Electrical and Computer Engineering Department

ECE 4510/5530 Microcontroller Applications

Project 2-Bonus

**Title: Laboratory Design Project**

Masoud Panahi, Donovan J Colo

TEAM 8

April 15, 2022

# Contents

## Introduction

The theme of the project is to move an object that has been placed on a conveyor belt from a start position to a specific end position. In this part of project a physical part of conveyor belt, LCD and ultrasonic sensor are used.

# Procedure

## Task a

In this project, dip switches are used to simulate start and stop switches. Green and Red LEDs are used as indicators. To provide 24 volts for conveyor belt system, relay is used. To divide voltage from 24V to 3.3V, two 12k ohm resistor in parallel made a divider with 1K ohm resistor. Ports and setting are listed below. C code is provided in appendix part, calculation for different type of timers are shown in comments.

**List of ports and settings:**

Start Switch : bounce-free switch, input, PE7 (EXTI7)

Stop Switch: active low, Input, IR Signal,PE8(EXTI8)

IR (Infra Red) Emitter: Red LED, Output, PG14

LED indicator : Green LED, output( Six times,1 blink/s) and ( 10 times,2 blink/s), PG13

Buzzer(a small speaker): Outputf(5.5 KHz)TIM3_CH4_OC_Buzzer_PB1

Ultrasonic sensor: PC2 and systick timer-TIM7 to make 20us trig every 50ms, Pulse = 25us*50MHz=1250, TIM2_CH4_PB11 input capture.

Forward direction control_Pin15: PG12

Reverse direction control_Pin16: PG11

Photo receiver sensor 1_Pin5: PG10

Photo receiver sensor 2_Pin6: PG9

LCD: PD0 to PD7 and PF0 to PF5 (control Pins)

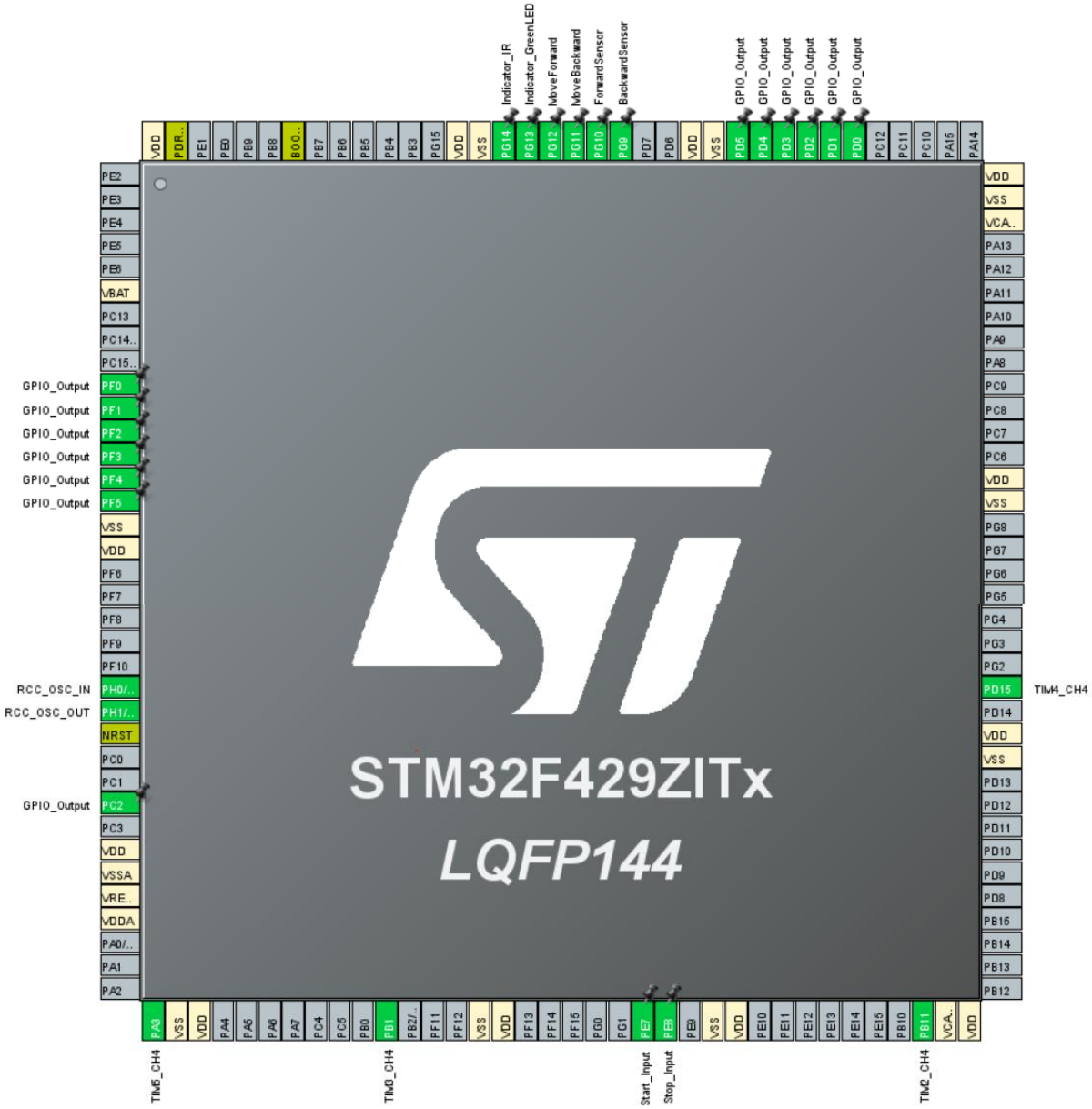



**Figure 1. port configuration**

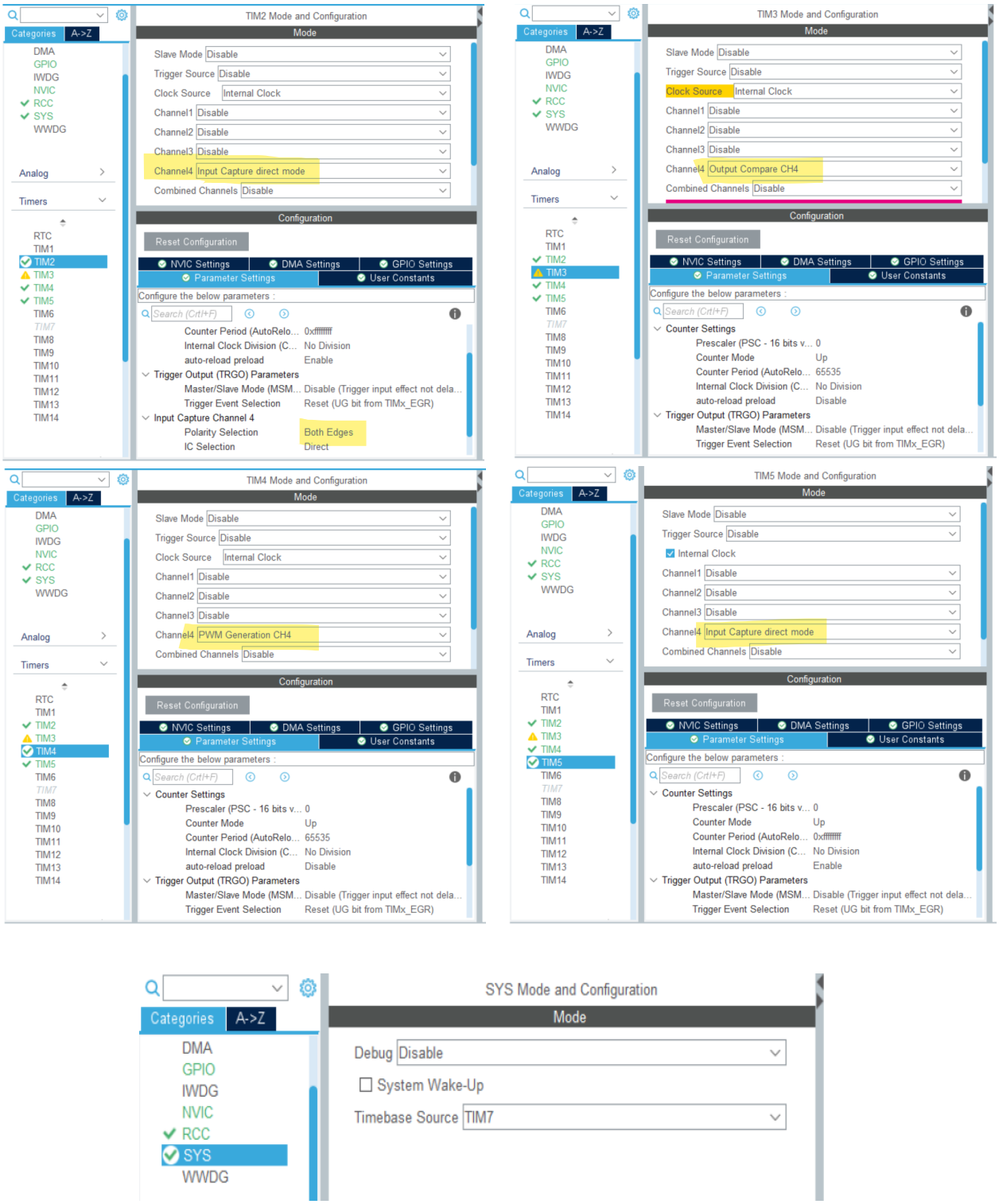



**Figure 2. Timers' configuration**

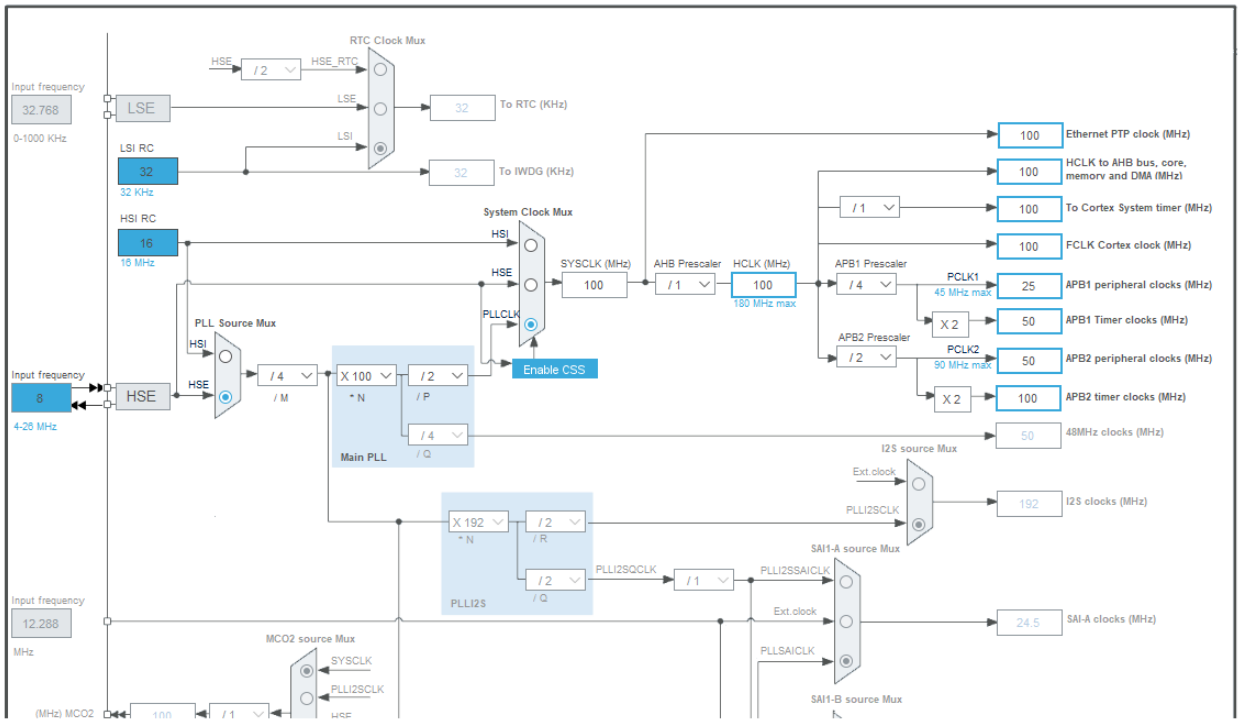



**Figure 3. Clock configuration**

# Task b

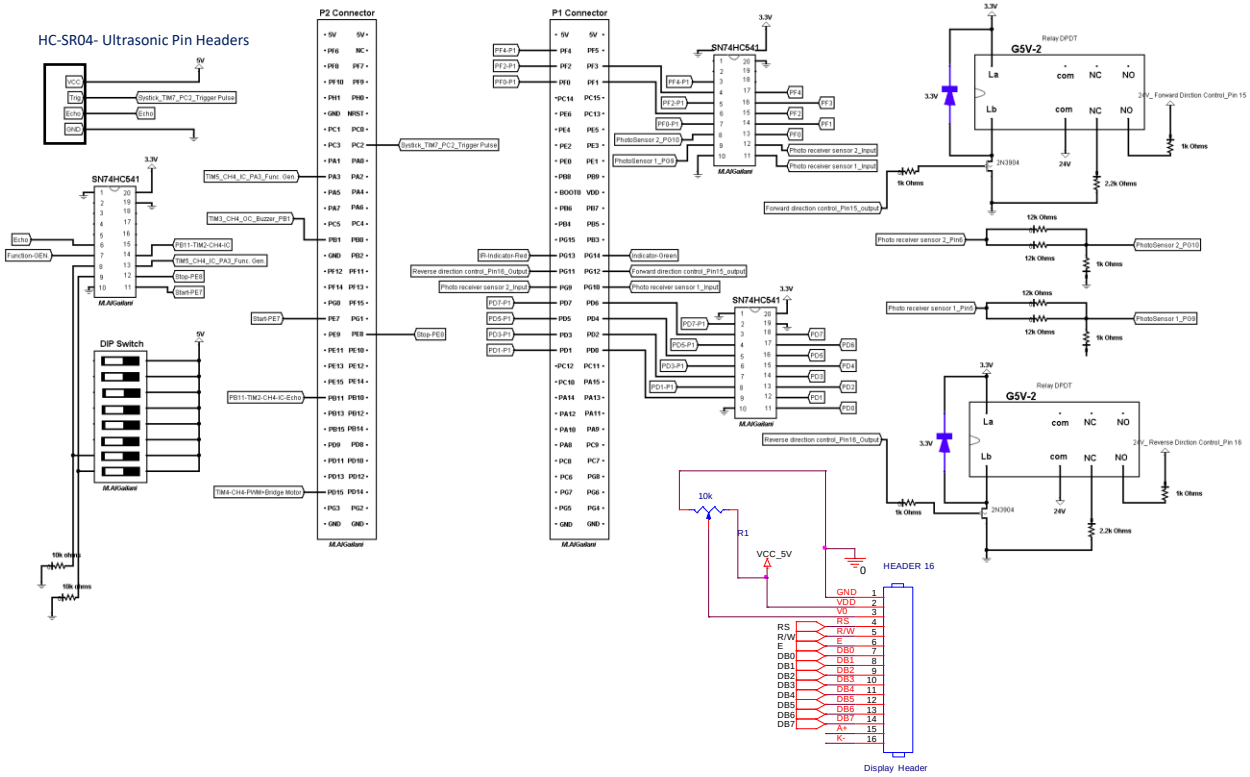
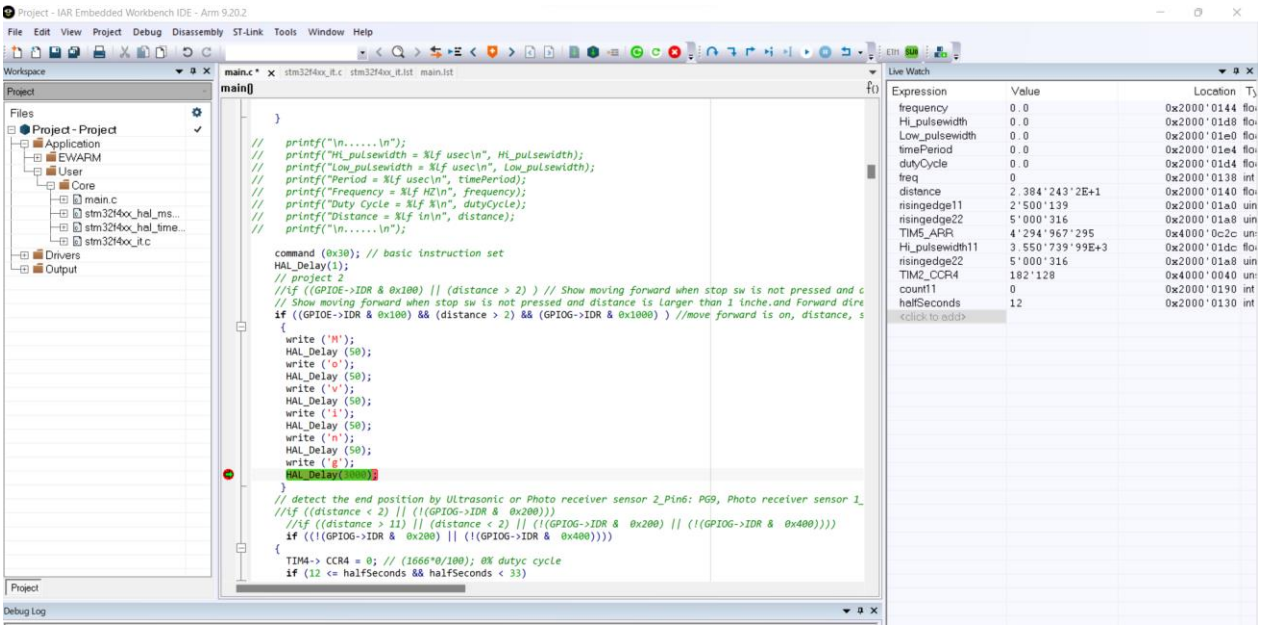
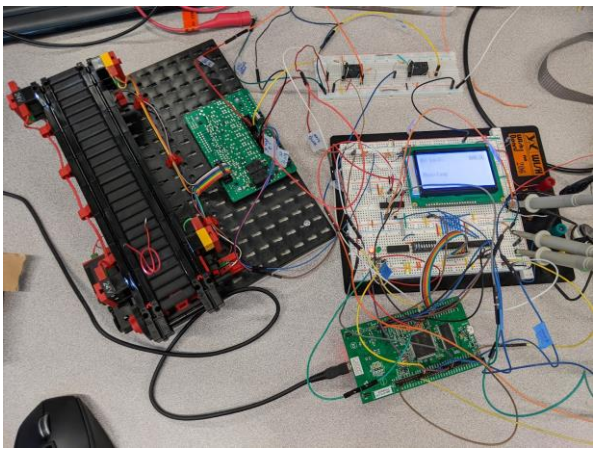



**Figure 4. Schematic**

# Task c

# Bonus Project #1



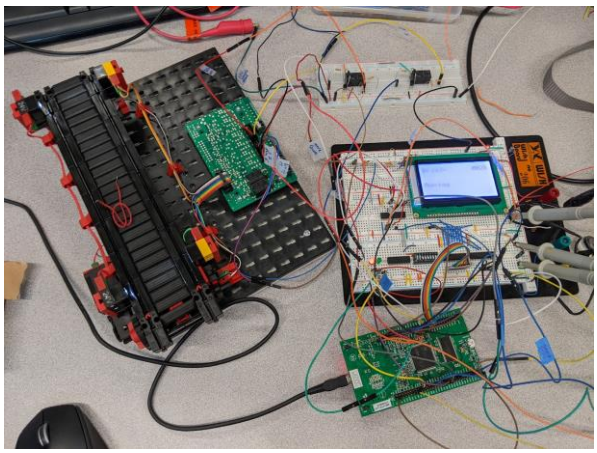
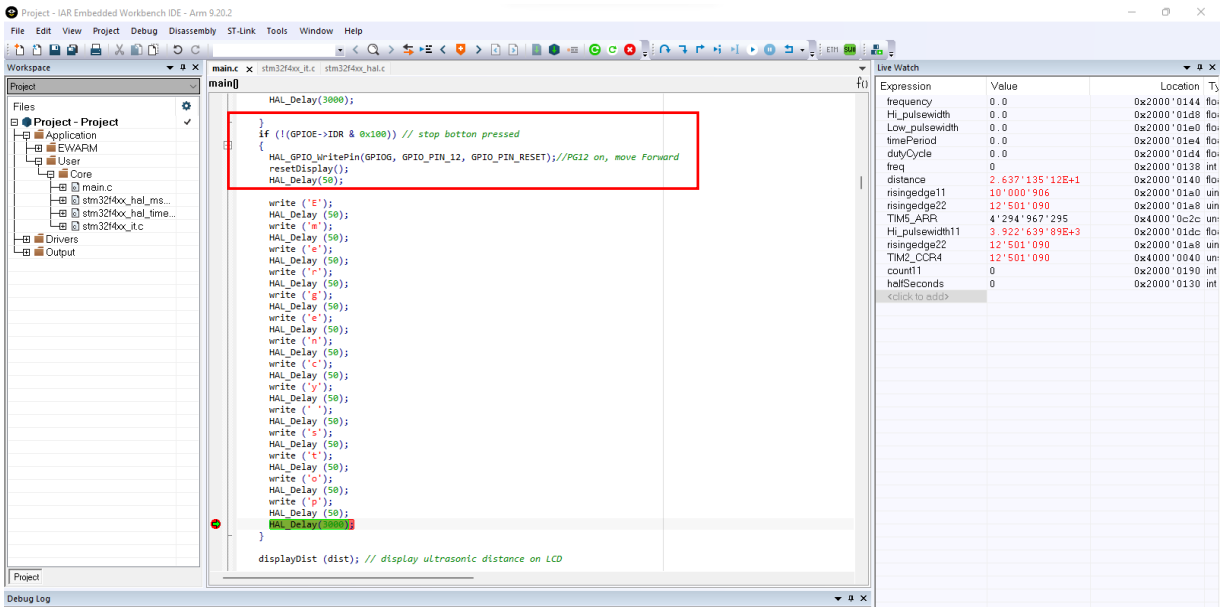


**Figure 5. Operate the prototype conveyor belt system moving forward and backward**

# Bonus Project #2



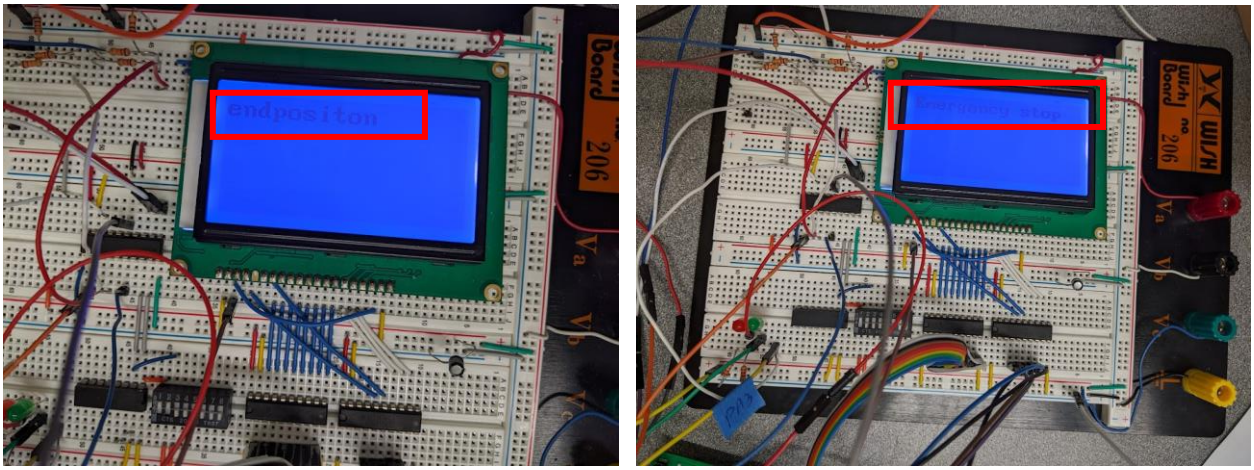
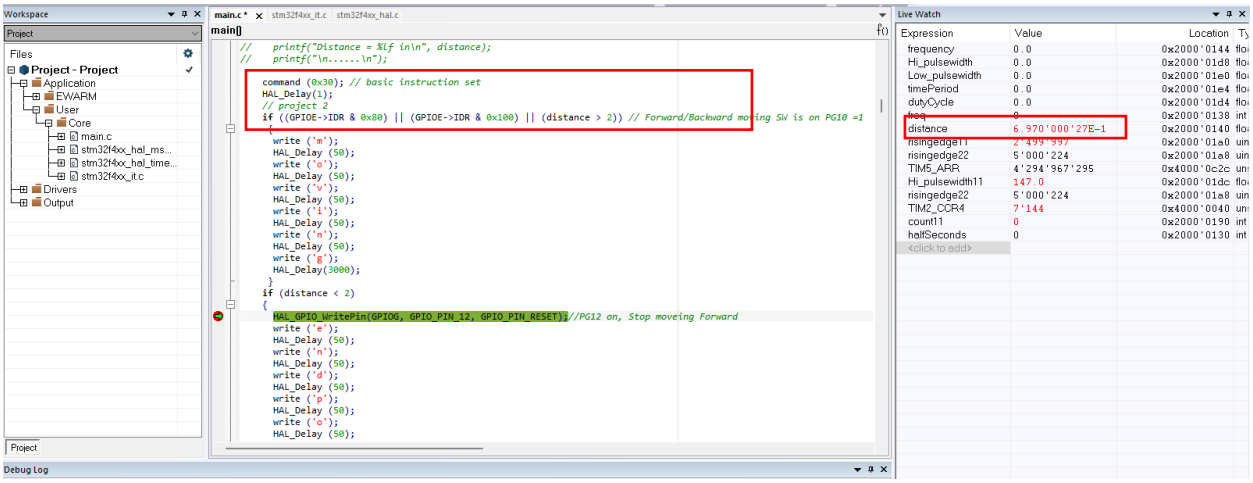
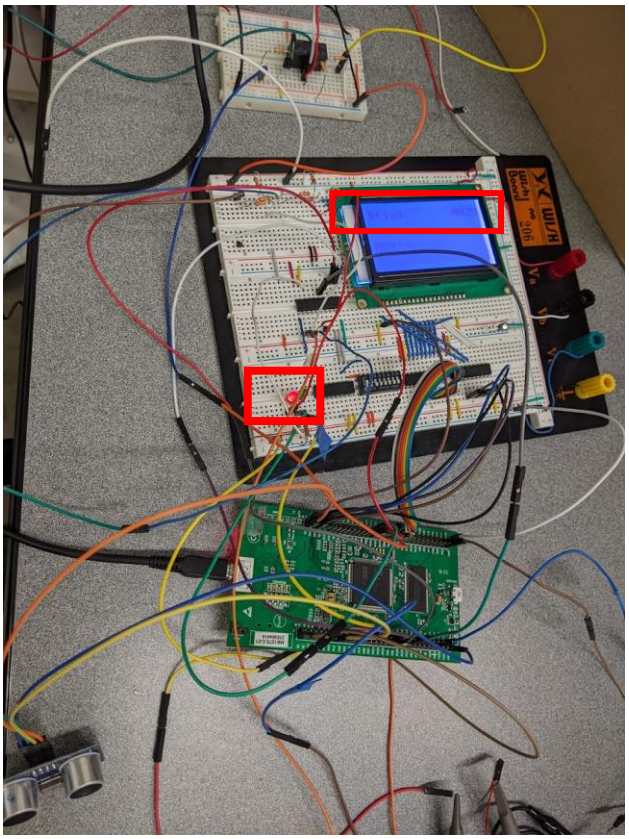




**Figure 6. Graphics LCD display to the system, two stop positions, emergency stop.**

# Bonus Project #3



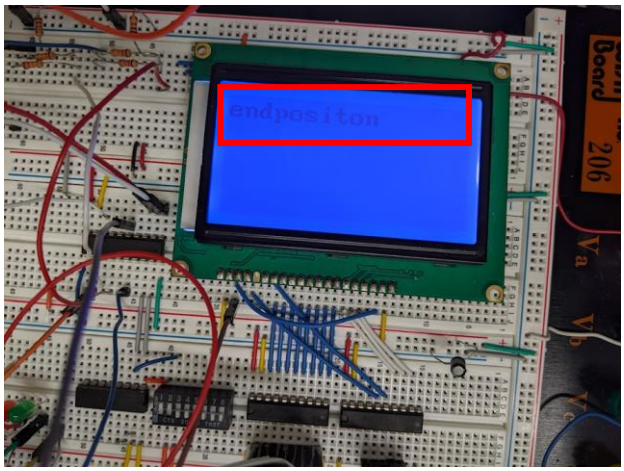
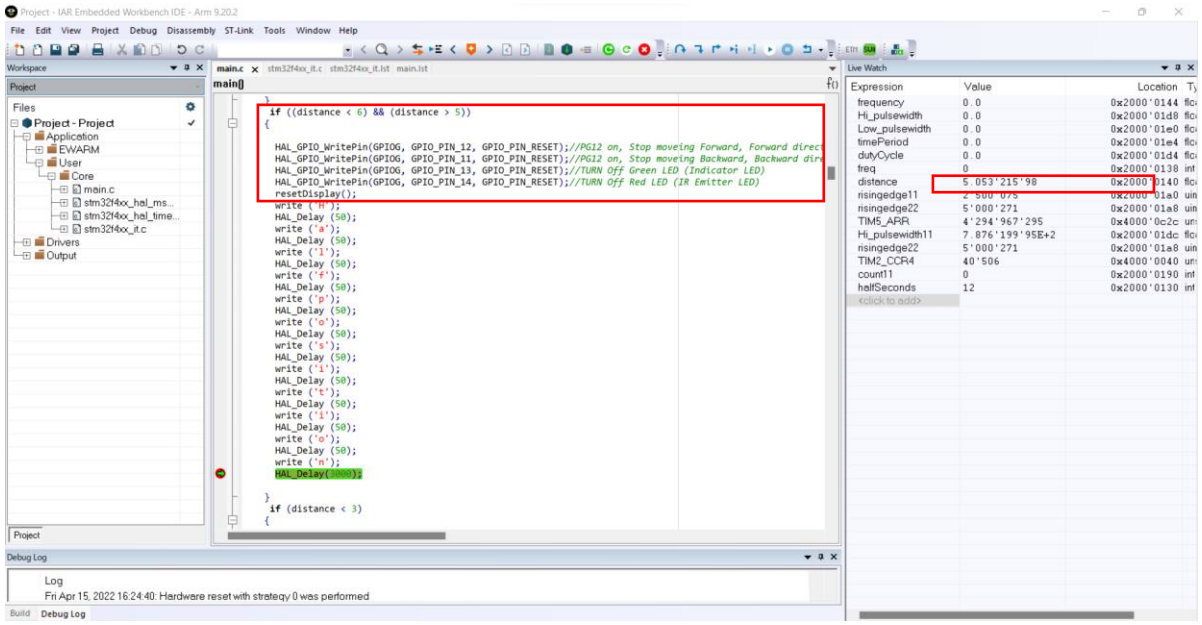



**Figure 7. Utilize the HC-SR04 ultrasonic sensor, Establish two locations at specified distances from the end of the belt**



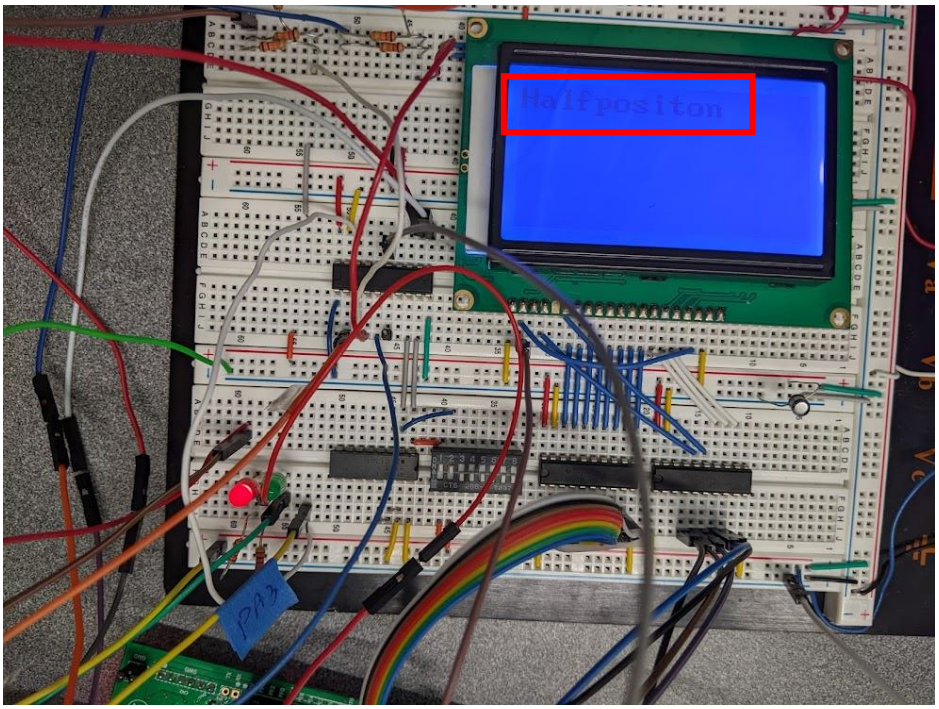
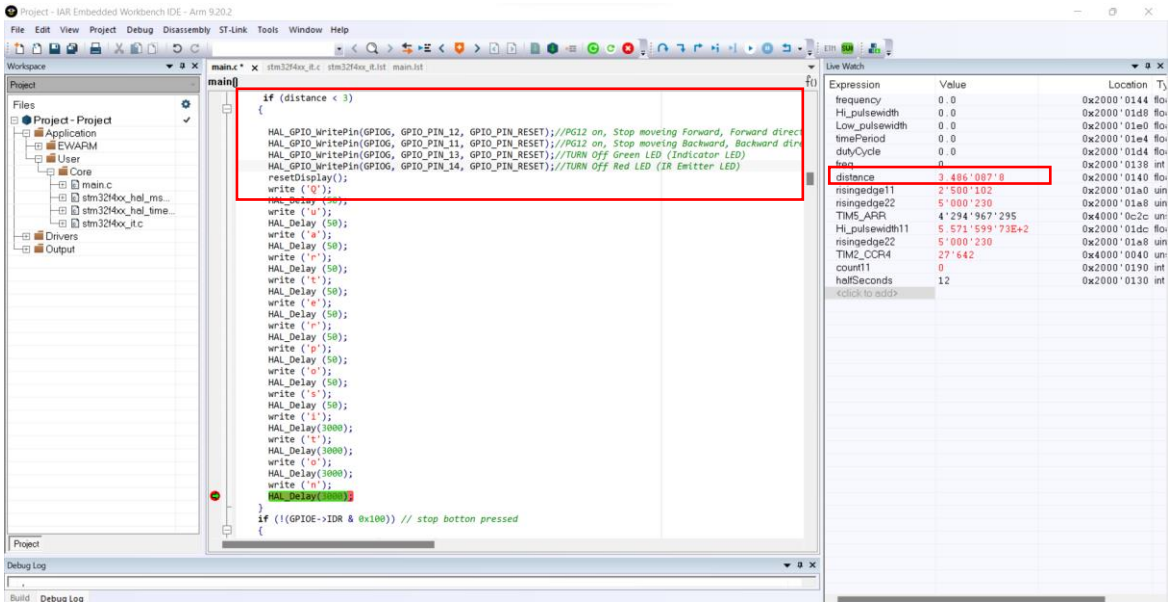




**Figure 8. Utilize the HC-SR04 ultrasonic sensor, Establish half position distance from the end of the belt**



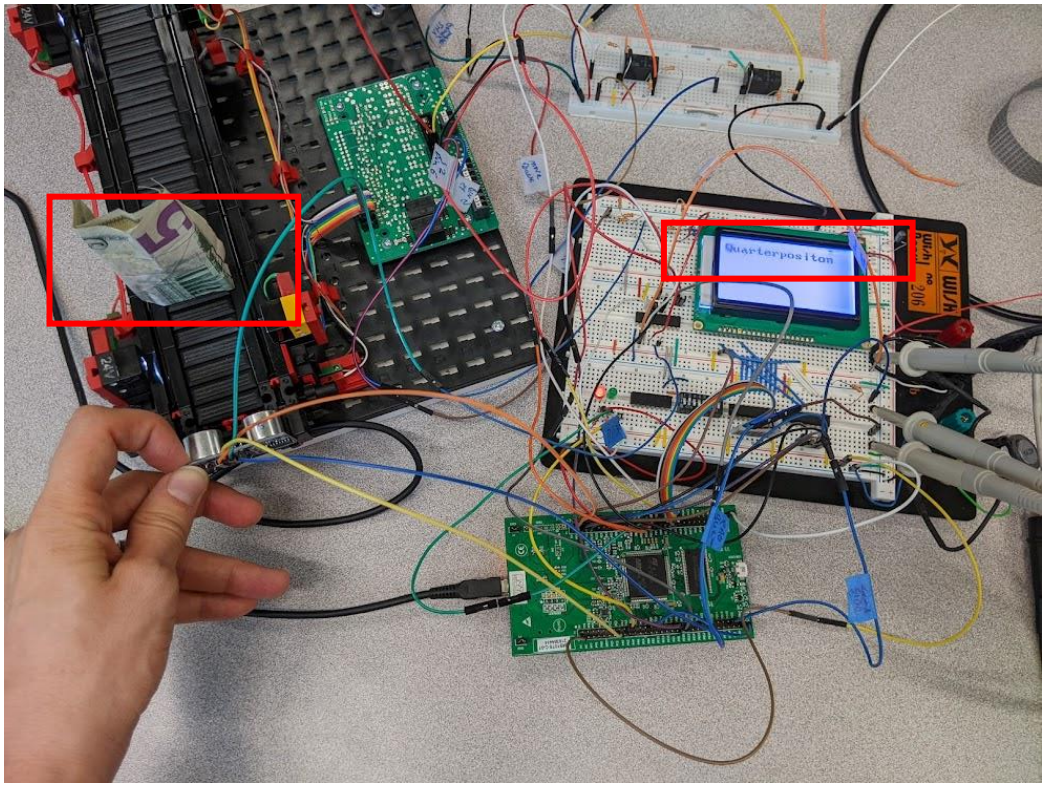




**Figure 9. Utilize the HC-SR04 ultrasonic sensor, Establish quarter position distance from the end of the belt**

# Appendix

## Main.c

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "stdio.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;
TIM_HandleTypeDef htim5;

/* USER CODE BEGIN PV */
int halfSeconds = 0;
int count4 = 0;
int freq;
int dist;

float distance = 0x0000;
uint8_t one = 0x0000;
uint8_t two = 0x0000;
uint8_t three = 0x0000;
//uint16_t i = 0x0000;
uint16_t R = 0x0000;
float Hi_pulsewidth = 0x0000;
float Hi_pulsewidth11 = 0x0000; // for Ultrasonic sensor to measure distance
float Low_pulsewidth= 0x0000;
```

```
float frequency = 0x0000;
float dutyCycle = 0x0000;
float timePeriod = 0x0000;
char temp;
uint16_t temp2;
uint16_t temp3;
uint32_t prevDuty = 0;
unsigned int blocks = 0;
unsigned int remainder = 0;
unsigned int ref, pos = 0;
//uint32_t freq;
unsigned int fd1, fd2, fd3, fd4, fd5, fd6 = 0;
char digits[6] = {0,0,0,0,0,0};
uint32_t duty;
char dcd1, dcd2 = 0;
char dcds[2] = {0,0};
unsigned int pulse = 0;
unsigned int t1, t2, t3;
char T[3] = {0,0,0};
unsigned int p1, p2, p3;
char P[3] = {0,0,0};
unsigned int bonus = 0;

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM5_Init(void);
static void MX_TIM4_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void command (char i);
void write(char i);
void init();
static void displayDist(int);
static void displayFreq(int);
static void displayDC(int);
void resetDisplay();
char number (int);
void graphicPosition(uint16_t, uint16_t );
void initGraphicMode(void);
void clearGFX();
void drawWave(uint32_t);
void drawRem(unsigned int);
void drawLowRem(unsigned int);
void drawPixel(unsigned int);
void drawFreq(uint32_t);
void drawDuty(int);
void drawPWT(unsigned int, unsigned int);
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
```

```
  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM3_Init();
  MX_TIM2_Init();
  MX_TIM5_Init();
  MX_TIM4_Init();
  /* USER CODE BEGIN 2 */
  GPIOE->IDR  = 0x80; //start is off when the power is On at the first time
  GPIOG->IDR = 0x0000;
  GPIOG->ODR = 0x0000;
  HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_4); //Starts the TIM Input Capture measurement in interrupt mode.
  //HAL_TIM_PWM_Start(&htim5, TIM_CHANNEL_3);// to make a pulse with 25 us width and 55 ms period. or
HAL_TIM_PWM_Start(&htim5, TIM_CHANNEL_4); check it !!!

  SysTick->CTRL = 0x0005; //"Choose the undividided clock source, disable the
  SysTick->LOAD = 2000; //ticks Number of ticks between two interrupts = 20us*100MHz= 2000
  SysTick->VAL = 0x000000;; // Current counter value
  SysTick->CTRL |= 0x0007; //The SysTick CTRL register enables the SysTick features

  //HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_3); //Starts the TIM Input Capture measurement in interrupt mode.
  init(); // initialize the LCD
  resetDisplay(); // reset dispaly
  initGraphicMode(); // initialize in graphic Mode
    HAL_Delay(10);
    clearGFX(); // clear Dispaly
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
     //conver float values to int

     freq = (int) frequency;
     int DC = (int) dutyCycle;
     int period = (int) timePeriod;
     int pulse = (int) Hi_pulsewidth;
     dist = (int) distance;

     if (freq == 4700) //frequency_KHz == 4.7
     {
      TIM4-> CCR4 = 1499; // (1666*90/100); 90% dutyc cycle
     }

    if (freq == 4900)//frequency_KHz == 4.9
     {
      TIM4-> CCR4 = 1332; // (1666*80/100); 80% dutyc cycle
     }
```

```
    if (freq == 5100)//frequency_KHz == 5.1
    {
      TIM4-> CCR4 = 1166; // (1666*70/100); 70% dutyc cycle
    }

    if (freq == 5300)//frequency_KHz == 5.3
    {
      TIM4-> CCR4 = 999; // (1666*60/100); 60% dutyc cycle
    }

    if (freq == 5500)//frequency_KHz == 5.5
    {
      TIM4-> CCR4 = 833; // (1666*50/100); 50% dutyc cycle
    }

    if (freq == 5700)//frequency_KHz == 5.7
    {
      TIM4-> CCR4 = 666; // (1666*40/100); 40% dutyc cycle
    }

    if (freq == 5900)//frequency_KHz == 5.9
    {
      TIM4-> CCR4 = 499; // (1666*30/100); 30% dutyc cycle
    }

    if (freq == 6100)//frequency_KHz == 6.1
    {
      TIM4-> CCR4 = 333; // (1666*20/100); 20% dutyc cycle
    }

    if (freq == 6300)//frequency_KHz == 6.3
    {
      TIM4-> CCR4 = 166; // (1666*10/100); 10% dutyc cycle
    }

    if (freq  > 6300) // end position detected; PWM signal should be turned off
    {
      count4 = 2;
      TIM4-> CCR4 = 0; // (1666*0/100); 0% dutyc cycle
      if (12 <= halfSeconds && halfSeconds < 33)
      {
        HAL_TIM_OC_Start_IT(&htim3, TIM_CHANNEL_4); // Start Buzzer and indicator
      }

      //HAL_TIM_PWM_Stop_IT(&htim4, TIM_CHANNEL_4); // Stop PWM
      //HAL_TIM_OC_Start_IT(&htim3, TIM_CHANNEL_4); // Start Buzzer

    }

//    printf("\n......\n");
//    printf("Hi_pulsewidth = %lf usec\n", Hi_pulsewidth);
//    printf("Low_pulsewidth = %lf usec\n", Low_pulsewidth);
//    printf("Period = %lf usec\n", timePeriod);
//    printf("Frequency = %lf HZ\n", frequency);
//    printf("Duty Cycle = %lf %\n", dutyCycle);
//    printf("Distance = %lf in\n", distance);
//    printf("\n......\n");

    command (0x30); // basic instruction set
    HAL_Delay(1);
    // project 2
    //if ((GPIOE->IDR & 0x100) || (distance > 2) ) // Show moving forward when stop sw is not pressed and distance is larger than 1 inches.
    // Show moving forward when stop sw is not pressed and distance is larger than 1 inche.and Forward direction control_Pin15: PG12 is on.
    if ((GPIOE->IDR & 0x100) && (distance > 2) && (GPIOG->IDR & 0x1000) ) //move forward is on, distance, stop sw is on
     {
      write ('M');
      HAL_Delay (50);
      write ('o');
```

```
      HAL_Delay (50);
      write ('v');
      HAL_Delay (50);
      write ('i');
      HAL_Delay (50);
      write ('n');
      HAL_Delay (50);
      write ('g');
      HAL_Delay(3000);
     }
    // detect the end position by Ultrasonic or Photo receiver sensor 2_Pin6: PG9, Photo receiver sensor 1_Pin5: PG10  ( falling edge detection by using polling technique)
    //if ((distance < 2) || (!(GPIOG->IDR &  0x200)))
      //if ((distance > 11) || (distance < 2) || (!(GPIOG->IDR &  0x200) || (!(GPIOG->IDR &  0x400))))
      if ((!(GPIOG->IDR &  0x200) || (!(GPIOG->IDR &  0x400))))
    {
      TIM4-> CCR4 = 0; // (1666*0/100); 0% dutyc cycle
      if (12 <= halfSeconds && halfSeconds < 33)
      {
       HAL_TIM_OC_Start_IT(&htim3, TIM_CHANNEL_4); // Start Buzzer and indicator
      }
      HAL_GPIO_WritePin(GPIOG, GPIO_PIN_12, GPIO_PIN_RESET);//PG12 on, Stop moveing Forward, Forward direction control_Pin15: PG12
      HAL_GPIO_WritePin(GPIOG, GPIO_PIN_11, GPIO_PIN_RESET);//PG12 on, Stop moveing Backward, Backward direction control_Pin16: PG11
      HAL_GPIO_WritePin(GPIOG, GPIO_PIN_13, GPIO_PIN_RESET);//TURN Off Green LED (Indicator LED)
      HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_RESET);//TURN Off Red LED (IR Emitter LED)
      resetDisplay();
      write ('E');
      HAL_Delay (50);
      write ('n');
      HAL_Delay (50);
      write ('d');
      HAL_Delay (50);
      write ('p');
      HAL_Delay (50);
      write ('o');
      HAL_Delay (50);
      write ('s');
      HAL_Delay (50);
      write ('i');
      HAL_Delay (50);
      write ('t');
      HAL_Delay (50);
      write ('o');
      HAL_Delay (50);
      write ('n');
      HAL_Delay(3000);
    }
     if ((distance < 6) && (distance > 5))
    {

      HAL_GPIO_WritePin(GPIOG, GPIO_PIN_12, GPIO_PIN_RESET);//PG12 on, Stop moveing Forward, Forward direction control_Pin15: PG12
      HAL_GPIO_WritePin(GPIOG, GPIO_PIN_11, GPIO_PIN_RESET);//PG12 on, Stop moveing Backward, Backward direction control_Pin16: PG11
      HAL_GPIO_WritePin(GPIOG, GPIO_PIN_13, GPIO_PIN_RESET);//TURN Off Green LED (Indicator LED)
      //HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_RESET);//TURN Off Red LED (IR Emitter LED)
      resetDisplay();
      write ('H');
      HAL_Delay (50);
      write ('a');
      HAL_Delay (50);
      write ('l');
      HAL_Delay (50);
      write ('f');
      HAL_Delay (50);
      write ('p');
```

```
      HAL_Delay (50);
      write ('o');
      HAL_Delay (50);
      write ('s');
      HAL_Delay (50);
      write ('i');
      HAL_Delay (50);
      write ('t');
      HAL_Delay (50);
      write ('i');
      HAL_Delay (50);
      write ('o');
      HAL_Delay (50);
      write ('n');
      HAL_Delay(3000);

    }
     if (distance < 3)
    {

      HAL_GPIO_WritePin(GPIOG, GPIO_PIN_12, GPIO_PIN_RESET);//PG12 on, Stop moveing Forward, Forward direction
control_Pin15: PG12
      HAL_GPIO_WritePin(GPIOG, GPIO_PIN_11, GPIO_PIN_RESET);//PG12 on, Stop moveing Backward, Backward direction
control_Pin16: PG11
      HAL_GPIO_WritePin(GPIOG, GPIO_PIN_13, GPIO_PIN_RESET);//TURN Off Green LED (Indicator LED)
      HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_RESET);//TURN Off Red LED (IR Emitter LED)
      resetDisplay();
      write ('Q');
      HAL_Delay (50);
      write ('u');
      HAL_Delay (50);
      write ('a');
      HAL_Delay (50);
      write ('r');
      HAL_Delay (50);
      write ('t');
      HAL_Delay (50);
      write ('e');
      HAL_Delay (50);
      write ('r');
      HAL_Delay (50);
      write ('p');
      HAL_Delay (50);
      write ('o');
      HAL_Delay (50);
      write ('s');
      HAL_Delay (50);
      write ('i');
      HAL_Delay(3000);
      write ('t');
      HAL_Delay(3000);
      write ('o');
      HAL_Delay(3000);
      write ('n');
      HAL_Delay(3000);
    }
    if (!(GPIOE->IDR & 0x100)) // stop botton pressed
    {
      TIM4-> CCR4 = 0; // (1666*0/100); 0% dutyc cycle
      if (12 <= halfSeconds && halfSeconds < 33)
      {
        HAL_TIM_OC_Start_IT(&htim3, TIM_CHANNEL_4); // Start Buzzer and indicator
      }
      HAL_GPIO_WritePin(GPIOG, GPIO_PIN_12, GPIO_PIN_RESET);//PG12 on, Stop moveing Forward, Forward direction
control_Pin15: PG12
      HAL_GPIO_WritePin(GPIOG, GPIO_PIN_11, GPIO_PIN_RESET);//PG12 on, Stop moveing Backward, Backward direction
control_Pin16: PG11
```

```
      resetDisplay();
      HAL_Delay(50);

      write ('E');
      HAL_Delay (50);
      write ('m');
      HAL_Delay (50);
      write ('e');
      HAL_Delay (50);
      write ('r');
      HAL_Delay (50);
      write ('g');
      HAL_Delay (50);
      write ('e');
      HAL_Delay (50);
      write ('n');
      HAL_Delay (50);
      write ('c');
      HAL_Delay (50);
      write ('y');
      HAL_Delay (50);
      write (' ');
      HAL_Delay (50);
      write ('s');
      HAL_Delay (50);
      write ('t');
      HAL_Delay (50);
      write ('o');
      HAL_Delay (50);
      write ('p');
      HAL_Delay (50);
      HAL_Delay(3000);
    }

    displayDist (dist); // display ultrasonic distance on LCD

//    displayFreq (freq);
//    displayDC (DC);
//    initGraphicMode();
//    HAL_Delay(10);
//    clearGFX();
//    HAL_Delay(10);
//    drawWave(DC);
//    HAL_Delay(1);
//     // Draw Period Time 'PT' followed by 3 digit period in microseconds
//    // Draw Pulse Width 'PW' followed by 3 digit width in microseconds
//    drawPWT(period, pulse);
//    HAL_Delay(1);
//
//    // Display letter 'F' followed by six digit frequency in Hz
//    drawFreq(freq);
//    HAL_Delay(1);
//
//    // Display letters 'DC' followed by two digit duty cycle in %
//    drawDuty(duty);
//    HAL_Delay(3000);


  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
```

```
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_LSI|RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.LSIState = RCC_LSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 4;
  RCC_OscInitStruct.PLL.PLLN = 100;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_IC_InitTypeDef sConfigIC = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 0;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 0xffffffff;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
```

```
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_IC_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_BOTHEDGE;
  sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
  sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
  sConfigIC.ICFilter = 0;
  if (HAL_TIM_IC_ConfigChannel(&htim2, &sConfigIC, TIM_CHANNEL_4) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */

}

/**
  * @brief TIM3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM3_Init(void)
{

  /* USER CODE BEGIN TIM3_Init 0 */

  /* USER CODE END TIM3_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM3_Init 1 */

  /* USER CODE END TIM3_Init 1 */
  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 0;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 65535;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_OC_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
```

```
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_TOGGLE;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_OC_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_4) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM3_Init 2 */

  /* USER CODE END TIM3_Init 2 */
  HAL_TIM_MspPostInit(&htim3);

}

/**
  * @brief TIM4 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM4_Init(void)
{

  /* USER CODE BEGIN TIM4_Init 0 */

  /* USER CODE END TIM4_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM4_Init 1 */

  /* USER CODE END TIM4_Init 1 */
  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 0;
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim4.Init.Period = 65535;
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim4) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_4) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM4_Init 2 */

  /* USER CODE END TIM4_Init 2 */
  HAL_TIM_MspPostInit(&htim4);

}

/**
  * @brief TIM5 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM5_Init(void)
{

  /* USER CODE BEGIN TIM5_Init 0 */

  /* USER CODE END TIM5_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_IC_InitTypeDef sConfigIC = {0};

  /* USER CODE BEGIN TIM5_Init 1 */

  /* USER CODE END TIM5_Init 1 */
  htim5.Instance = TIM5;
  htim5.Init.Prescaler = 0;
  htim5.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim5.Init.Period = 0xffffffff;
  htim5.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim5.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim5) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim5, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_IC_Init(&htim5) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim5, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_BOTHEDGE;
  sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
  sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
  sConfigIC.ICFilter = 0;
  if (HAL_TIM_IC_ConfigChannel(&htim5, &sConfigIC, TIM_CHANNEL_4) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM5_Init 2 */

  /* USER CODE END TIM5_Init 2 */

}
```

```
/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOG_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
                    |GPIO_PIN_4|GPIO_PIN_5, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_2, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
                    |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOG, MoveBackward_Pin|MoveForward_Pin|Indicator_GreenLED_Pin|Indicator_IR_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pins : PF0 PF1 PF2 PF3
                           PF4 PF5 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
                    |GPIO_PIN_4|GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOF, &GPIO_InitStruct);

  /*Configure GPIO pin : PC2 */
  GPIO_InitStruct.Pin = GPIO_PIN_2;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : Start_Input_Pin Stop_Input_Pin */
  GPIO_InitStruct.Pin = Start_Input_Pin|Stop_Input_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

  /*Configure GPIO pins : PD0 PD1 PD2 PD3
                           PD4 PD5 PD6 PD7 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
                    |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /*Configure GPIO pins : BackwardSW_Pin ForwardSW_Pin */
  GPIO_InitStruct.Pin = BackwardSW_Pin|ForwardSW_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
  HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);

  /*Configure GPIO pins : MoveBackward_Pin MoveForward_Pin Indicator_GreenLED_Pin Indicator_IR_Pin */
  GPIO_InitStruct.Pin = MoveBackward_Pin|MoveForward_Pin|Indicator_GreenLED_Pin|Indicator_IR_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

}

/* USER CODE BEGIN 4 */
static void displayDist(int num)
{
  char digit_one = number(num % 10); // reminder = 9
  char digit_two = number((num / 10) % 10); // reminder = 9
  char digit_three = number((num / 100) % 10);//reminder =  9
  char digit_four = number((num / 1000) % 10);//reminder =  1
   resetDisplay();
   HAL_Delay(50);

//    write (' ');
//    write (' ');
//    write (' ');
//    write (' ');
//    write (' ');
//    write (' ');
    write ('D');
    HAL_Delay (50);
    write ('(');
    HAL_Delay (50);
    write ('i');
    HAL_Delay (50);
    write ('n');
    HAL_Delay (50);
    write (')');
    HAL_Delay (50);
    write ('=');
    HAL_Delay (50);
    write (' ');
    write (' ');
    write (' ');
    write (' ');
    write (' ');
    write (' ');

    write(digit_four);
    HAL_Delay(50);
    write(digit_three);
    HAL_Delay(50);
    write(digit_two);
    HAL_Delay(50);
    write(digit_one);
    HAL_Delay(100);
}

static void displayFreq(int num)
{
  char digit_one = number(num % 10); // reminder = 9
  char digit_two = number((num / 10) % 10); // reminder = 9
  char digit_three = number((num / 100) % 10);//reminder =  9
  char digit_four = number((num / 1000) % 10);//reminder =  1
   resetDisplay();
   HAL_Delay(50);
```

```
    write ('f');
    HAL_Delay (50);
    write ('(');
    HAL_Delay (50);
    write ('H');
    HAL_Delay (50);
    write ('z');
    HAL_Delay (50);
    write (')');
    HAL_Delay (50);
    write ('=');
    HAL_Delay (50);

    write(digit_four);
    HAL_Delay(50);
    write(digit_three);
    HAL_Delay(50);
    write(digit_two);
    HAL_Delay(50);
    write(digit_one);
    HAL_Delay(100);
}
static void displayDC(int num)
{
  char digit_one = number(num % 10); // reminder = 9
  char digit_two = number((num / 10) % 10); // reminder = 9

    write (' ');
    write (' ');
    write (' ');
    write (' ');
    write (' ');
    write (' ');
    HAL_Delay (50);
    write ('D');
    HAL_Delay (50);
    write ('C');
    HAL_Delay (50);
    write ('(');
    HAL_Delay (50);
    write ('%');
    HAL_Delay (50);
    write (')');
    HAL_Delay (50);
    write ('=');

    HAL_Delay(50);
    write(digit_two);
    HAL_Delay(50);
    write(digit_one);
    HAL_Delay(500);
    resetDisplay();
}
char number (int n)
{
  char R;
  switch (n)
  {
  case 0:
    R = '0';
    break;
  case 1:
    R = '1';
    break;
  case 2:
    R = '2';
    break;
```

```
  case 3:
    R = '3';
    break;
  case 4:
    R = '4';
    break;
  case 5:
    R = '5';
    break;
  case 6:
    R = '6';
    break;
  case 7:
    R = '7';
    break;
  case 8:
    R = '8';
    break;
  case 9:
    R = '9';
    break;
  }
  return R;
}
void resetDisplay()
{
 command(0x01);        //clear dispaly
 HAL_Delay (1);
 command(0x02);        //return cursor home
 HAL_Delay(1);      //
}
void graphicPosition(uint16_t horizontal, uint16_t vertical)
{
  // Vertical Position with 0x80 appended for Set Graphic RAM Addr. Func.
  vertical = vertical + 0x80;

  command (vertical);
  HAL_Delay (1);
 // Horizontal Position with 0x80 appended for Set Graphic RAM Addr. Func.
  horizontal = horizontal + 0x80;
  temp3 = horizontal;
  command (horizontal);
  HAL_Delay (1);
}
void initGraphicMode()
{
  command(0x36);// Enable extended function set
  HAL_Delay(10);

  command(0x41); // Set vertical scroll address
  HAL_Delay(10);
}
void clearGFX()
{
//    int i,j;
//  for(j=0x80;j<0xa1;j++)
//  {
//    for(i=0x80;i<0x90;i++)
//    {
//      command(j);
//      command(i);
//      write(0x00);
//      write(0x00);
//    }
//  }
//  HAL_Delay(1);

  // Clear LCD
```

```
  for(uint16_t vert = 0; vert < 32; vert++)
  {
    for(uint16_t horiz = 0; horiz < 16; horiz++)
    {
      // Send graphics address
      graphicPosition(horiz, vert);
      // Send pixel fill data
      write(0x00); // First 8-bits
      write(0x00); // Second 8-bits
    }
  }
}
// Method to draw all needed characters in graphics mode
void drawChar(unsigned int ref, unsigned int pos)
{
  // Reference values for numbers 0, 1, 2, 3, 4, 5, 6
  // F, D, C, are 10, 11, 12, respectively
  // T, W, are 14, 15, respectively
  // Reference 13 will insert a blank space.
  switch(ref)
  {
  case 0:
    // Draw 0
    graphicPosition(pos,1);
    write(0x0E);
    graphicPosition(pos,2);
    write(0x11);
    graphicPosition(pos,3);
    write(0x13);
    graphicPosition(pos,4);
    write(0x15);
    graphicPosition(pos,5);
    write(0x19);
    graphicPosition(pos,6);
    write(0x11);
    graphicPosition(pos,7);
    write(0x0E);
    break;
  case 1:
    // Draw 1
    graphicPosition(pos,1);
    write(0x04);
    graphicPosition(pos,2);
    write(0x0C);
    graphicPosition(pos,3);
    write(0x04);
    graphicPosition(pos,4);
    write(0x04);
    graphicPosition(pos,5);
    write(0x04);
    graphicPosition(pos,6);
    write(0x04);
    graphicPosition(pos,7);
    write(0x0E);
    break;
  case 2:
    // Draw 2
    graphicPosition(pos,1);
    write(0x0E);
    graphicPosition(pos,2);
    write(0x11);
    graphicPosition(pos,3);
    write(0x01);
    graphicPosition(pos,4);
    write(0x02);
    graphicPosition(pos,5);
    write(0x04);
    graphicPosition(pos,6);
```

```
    write(0x08);
    graphicPosition(pos,7);
    write(0x1F);
    break;
  case 3:
    // Draw 3
    graphicPosition(pos,1);
    write(0x1F);
    graphicPosition(pos,2);
    write(0x02);
    graphicPosition(pos,3);
    write(0x04);
    graphicPosition(pos,4);
    write(0x02);
    graphicPosition(pos,5);
    write(0x01);
    graphicPosition(pos,6);
    write(0x11);
    graphicPosition(pos,7);
    write(0x0E);
    break;
  case 4:
    // Draw 4
    graphicPosition(pos,1);
    write(0x02);
    graphicPosition(pos,2);
    write(0x06);
    graphicPosition(pos,3);
    write(0x0A);
    graphicPosition(pos,4);
    write(0x12);
    graphicPosition(pos,5);
    write(0x1F);
    graphicPosition(pos,6);
    write(0x02);
    graphicPosition(pos,7);
    write(0x02);
    break;
  case 5:
    // Draw 5
    graphicPosition(pos,1);
    write(0x1F);
    graphicPosition(pos,2);
    write(0x10);
    graphicPosition(pos,3);
    write(0x1E);
    graphicPosition(pos,4);
    write(0x01);
    graphicPosition(pos,5);
    write(0x01);
    graphicPosition(pos,6);
    write(0x11);
    graphicPosition(pos,7);
    write(0x0E);
    break;
  case 6:
    // Draw 6
    graphicPosition(pos,1);
    write(0x06);
    graphicPosition(pos,2);
    write(0x08);
    graphicPosition(pos,3);
    write(0x10);
    graphicPosition(pos,4);
    write(0x1E);
    graphicPosition(pos,5);
    write(0x11);
    graphicPosition(pos,6);
```

```
    write(0x11);
    graphicPosition(pos,7);
    write(0x0E);
    break;
  case 7:
    // Draw 7
    graphicPosition(pos,1);
    write(0x1F);
    graphicPosition(pos,2);
    write(0x01);
    graphicPosition(pos,3);
    write(0x02);
    graphicPosition(pos,4);
    write(0x04);
    graphicPosition(pos,5);
    write(0x08);
    graphicPosition(pos,6);
    write(0x08);
    graphicPosition(pos,7);
    write(0x08);
    break;
  case 8:
    // Draw 8
    graphicPosition(pos,1);
    write(0x0E);
    graphicPosition(pos,2);
    write(0x11);
    graphicPosition(pos,3);
    write(0x11);
    graphicPosition(pos,4);
    write(0x0E);
    graphicPosition(pos,5);
    write(0x11);
    graphicPosition(pos,6);
    write(0x11);
    graphicPosition(pos,7);
    write(0x0E);
    break;
  case 9:
    // Draw 9
    graphicPosition(pos,1);
    write(0x0E);
    graphicPosition(pos,2);
    write(0x11);
    graphicPosition(pos,3);
    write(0x11);
    graphicPosition(pos,4);
    write(0x0F);
    graphicPosition(pos,5);
    write(0x01);
    graphicPosition(pos,6);
    write(0x02);
    graphicPosition(pos,7);
    write(0x0C);
    break;
  case 10:
    // Draw 'F'
    graphicPosition(pos,1);
    write(0x1F);
    graphicPosition(pos,2);
    write(0x10);
    graphicPosition(pos,3);
    write(0x10);
    graphicPosition(pos,4);
    write(0x1E);
    graphicPosition(pos,5);
    write(0x10);
    graphicPosition(pos,6);
```

```
    write(0x10);
    graphicPosition(pos,7);
    write(0x10);
    break;
  case 11:
    // Draw 'D'
    graphicPosition(pos,16);
    write(0x1C);
    graphicPosition(pos,17);
    write(0x12);
    graphicPosition(pos,18);
    write(0x11);
    graphicPosition(pos,19);
    write(0x11);
    graphicPosition(pos,20);
    write(0x11);
    graphicPosition(pos,21);
    write(0x12);
    graphicPosition(pos,22);
    write(0x1C);
    break;
  case 12:
    // Draw 'C'
    graphicPosition(pos,16);
    write(0x0E);
    graphicPosition(pos,17);
    write(0x11);
    graphicPosition(pos,18);
    write(0x10);
    graphicPosition(pos,19);
    write(0x10);
    graphicPosition(pos,20);
    write(0x10);
    graphicPosition(pos,21);
    write(0x11);
    graphicPosition(pos,22);
    write(0x0E);
    break;
  case 13:
    // Draw blank space
    graphicPosition(pos,1);
    write(0x00);
    graphicPosition(pos,2);
    write(0x00);
    graphicPosition(pos,3);
    write(0x00);
    graphicPosition(pos,4);
    write(0x00);
    graphicPosition(pos,5);
    write(0x00);
    graphicPosition(pos,6);
    write(0x00);
    graphicPosition(pos,7);
    write(0x00);
    break;

  case 14:
    // Draw 'T'
    graphicPosition(pos,1);
    write(0x1F);
    graphicPosition(pos,2);
    write(0x04);
    graphicPosition(pos,3);
    write(0x04);
    graphicPosition(pos,4);
    write(0x04);
    graphicPosition(pos,5);
    write(0x04);
```

```
    graphicPosition(pos,6);
    write(0x04);
    graphicPosition(pos,7);
    write(0x04);
    break;
  case 15:
    // Draw 'W'
    graphicPosition(pos,1);
    write(0x11);
    graphicPosition(pos,2);
    write(0x11);
    graphicPosition(pos,3);
    write(0x11);
    graphicPosition(pos,4);
    write(0x11);
    graphicPosition(pos,5);
    write(0x15);
    graphicPosition(pos,6);
    write(0x15);
    graphicPosition(pos,7);
    write(0x0E);
    break;
  }
}

// Method for printing DC in proper location, used by drawDuty
void drawDC(unsigned int ref, unsigned int pos)
{
  switch(ref)
  {
  case 0:
    // Draw 0
    graphicPosition(pos,16);
    write(0x0E);
    graphicPosition(pos,17);
    write(0x11);
    graphicPosition(pos,18);
    write(0x13);
    graphicPosition(pos,19);
    write(0x15);
    graphicPosition(pos,20);
    write(0x19);
    graphicPosition(pos,21);
    write(0x11);
    graphicPosition(pos,22);
    write(0x0E);
    break;
  case 1:
    // Draw 1
    graphicPosition(pos,16);
    write(0x04);
    graphicPosition(pos,17);
    write(0x0C);
    graphicPosition(pos,18);
    write(0x04);
    graphicPosition(pos,19);
    write(0x04);
    graphicPosition(pos,20);
    write(0x04);
    graphicPosition(pos,21);
    write(0x04);
    graphicPosition(pos,22);
    write(0x0E);
    break;
  case 2:
    // Draw 2
    graphicPosition(pos,16);
    write(0x0E);
```

```
    graphicPosition(pos,17);
    write(0x11);
    graphicPosition(pos,18);
    write(0x01);
    graphicPosition(pos,19);
    write(0x02);
    graphicPosition(pos,20);
    write(0x04);
    graphicPosition(pos,21);
    write(0x08);
    graphicPosition(pos,22);
    write(0x1F);
    break;
  case 3:
    // Draw 3
    graphicPosition(pos,16);
    write(0x1F);
    graphicPosition(pos,17);
    write(0x02);
    graphicPosition(pos,18);
    write(0x04);
    graphicPosition(pos,19);
    write(0x02);
    graphicPosition(pos,20);
    write(0x01);
    graphicPosition(pos,21);
    write(0x11);
    graphicPosition(pos,22);
    write(0x0E);
    break;
  case 4:
    // Draw 4
    graphicPosition(pos,16);
    write(0x02);
    graphicPosition(pos,17);
    write(0x06);
    graphicPosition(pos,18);
    write(0x0A);
    graphicPosition(pos,19);
    write(0x12);
    graphicPosition(pos,20);
    write(0x1F);
    graphicPosition(pos,21);
    write(0x02);
    graphicPosition(pos,22);
    write(0x02);
    break;
  case 5:
    // Draw 5
    graphicPosition(pos,16);
    write(0x1F);
    graphicPosition(pos,17);
    write(0x10);
    graphicPosition(pos,18);
    write(0x1E);
    graphicPosition(pos,19);
    write(0x01);
    graphicPosition(pos,20);
    write(0x01);
    graphicPosition(pos,21);
    write(0x11);
    graphicPosition(pos,22);
    write(0x0E);
    break;
  case 6:
    // Draw 6
    graphicPosition(pos,16);
    write(0x06);
```

```
    graphicPosition(pos,17);
    write(0x08);
    graphicPosition(pos,18);
    write(0x10);
    graphicPosition(pos,19);
    write(0x1E);
    graphicPosition(pos,20);
    write(0x11);
    graphicPosition(pos,21);
    write(0x11);
    graphicPosition(pos,22);
    write(0x0E);
    break;
  case 7:
    // Draw 7
    graphicPosition(pos,16);
    write(0x1F);
    graphicPosition(pos,17);
    write(0x01);
    graphicPosition(pos,18);
    write(0x02);
    graphicPosition(pos,19);
    write(0x04);
    graphicPosition(pos,20);
    write(0x08);
    graphicPosition(pos,21);
    write(0x08);
    graphicPosition(pos,22);
    write(0x08);
    break;
  case 8:
    // Draw 8
    graphicPosition(pos,16);
    write(0x0E);
    graphicPosition(pos,17);
    write(0x11);
    graphicPosition(pos,18);
    write(0x11);
    graphicPosition(pos,19);
    write(0x0E);
    graphicPosition(pos,20);
    write(0x11);
    graphicPosition(pos,21);
    write(0x11);
    graphicPosition(pos,22);
    write(0x0E);
    break;
  case 9:
    // Draw 9
    graphicPosition(pos,16);
    write(0x0E);
    graphicPosition(pos,17);
    write(0x11);
    graphicPosition(pos,18);
    write(0x11);
    graphicPosition(pos,19);
    write(0x0F);
    graphicPosition(pos,20);
    write(0x01);
    graphicPosition(pos,21);
    write(0x02);
    graphicPosition(pos,22);
    write(0x0C);
    break;
  }
 }

 // Method for drawing frequency in graphics mode
```

```
void drawFreq(uint32_t freq)
{
  // Draw 'F' = ref 7, in position 0
//  drawChar(10, 0);
  drawChar(10,8);
  // Split freq into six digits
  fd6 = freq % 10;
  freq = freq / 10;
  fd5 = freq % 10;
  freq = freq / 10;
  fd4 = freq % 10;
  freq = freq / 10;
  fd3 = freq % 10;
  freq = freq / 10;
  fd2 = freq % 10;
  freq = freq / 10;
  fd1 = freq % 10;

  // Assign freq digits to array
  digits[0] = fd1;
  digits[1] = fd2;
  digits[2] = fd3;
  digits[3] = fd4;
  digits[4] = fd5;
  digits[5] = fd6;

  // Print digits
  for(int i = 0; i < 6; i++)
  {
    switch(digits[i])
    {
    case 9:
      // Draw 9, add one to position because F is at 0
      drawChar(9, i+9);
      break;
    case 8:
      // Draw 8
      drawChar(8, i+9);
      break;
    case 7:
      // Draw 7
      drawChar(7, i+9);
      break;
    case 6:
      // Draw 6
      drawChar(6, i+9);
      break;
    case 5:
      // Draw 5
      drawChar(5, i+9);
      break;
    case 4:
      // Draw 4
      drawChar(4, i+9);
      break;
    case 3:
      // Draw 3
      drawChar(3, i+9);
      break;
    case 2:
      // Draw 2
      drawChar(2, i+9);
      break;
    case 1:
      // Draw 1
      drawChar(1, i+9);
      break;
    case 0:
```

```
      // Draw 0
      drawChar(0, i+9);
      break;

    }
  }

}
// Method to draw Period and Pulse Width
void drawPWT(unsigned int period, unsigned int pulse)
{
  // Draw Period label 'T'
  drawChar(14, 0);
//  drawChar(14,8);
  // Draw Period Time value as 3 digits in microseconds
  // Split period into 3 digits
  t1 = period % 10;
  period = period /10;
  t2 = period % 10;
  period = period / 10;
  t3 = period % 10;
  T[0] = t3;
  T[1] = t2;
  T[2] = t1;
  for(int i = 0; i < 3; i++)
  {
    switch(T[i])
    {
    case 9:
      // Draw 6
      drawChar(9, i+1);
      break;
    case 8:
      // Draw 6
      drawChar(8, i+1);
      break;
    case 7:
      // Draw 6
      drawChar(7, i+1);
      break;
    case 6:
      // Draw 6
      drawChar(6, i+1);
      break;
    case 5:
      // Draw 5
      drawChar(5, i+1);
      break;
    case 4:
      // Draw 4
      drawChar(4, i+1);
      break;
    case 3:
      // Draw 3
      drawChar(3, i+1);
      break;
    case 2:
      // Draw 2
      drawChar(2, i+1);
      break;
    case 1:
      // Draw 1
      drawChar(1, i+1);
      break;
    case 0:
      // Draw 0
      drawChar(0, i+1);
      break;
```

```
   }
  }
  // Draw Pulse Width label 'W'
  drawChar(15,4);
  // Draw Pulse Width value as 3 digits in microseonds
  p1 = pulse % 10;
  pulse = pulse / 10;
  p2 = pulse % 10;
  pulse = pulse / 10;
  p3 = pulse % 10;
  P[0] = p3;
  P[1] = p2;
  P[2] = p1;
  for(int i = 0; i < 3; i++)
  {
    switch(P[i])
    {
    case 9:
      // Draw 6
      drawChar(9, i+5);
      break;
    case 8:
      // Draw 6
      drawChar(8, i+5);
      break;
    case 7:
      // Draw 6
      drawChar(7, i+5);
      break;
    case 6:
      // Draw 6
      drawChar(6, i+5);
      break;
    case 5:
      // Draw 5
      drawChar(5, i+5);
      break;
    case 4:
      // Draw 4
      drawChar(4, i+5);
      break;
    case 3:
      // Draw 3
      drawChar(3, i+5);
      break;
    case 2:
      // Draw 2
      drawChar(2, i+5);
      break;
    case 1:
      // Draw 1
      drawChar(1, i+5);
      break;
    case 0:
      // Draw 0
      drawChar(0, i+5);
      break;

    }
   }
}
// Method for drawing duty cycle in graphics mode
void drawDuty(int DC)
{

  // Draw 'D' and 'C' label
  drawChar(11,10);
```

```
  drawChar(12,11);

  // Split duty cycle into two digits
  dcd1 = (((int) dutyCycle) % 10);
  dcd2 = (((int) dutyCycle) / 10) % 10;

  // Assign duty cycle digits to array
  dcds[0] = dcd2;
  dcds[1] = dcd1;

  // Print Duty Cycle Value
  for(int i = 0; i < 2; i++)
  {
   switch(dcds[i])
   {
   case 9:
     // Draw 6
     drawDC(9, i+12);
     break;
   case 8:
     // Draw 6
     drawDC(8, i+12);
     break;
   case 7:
     // Draw 6
     drawDC(7, i+12);
     break;
   case 6:
     // Draw 6
     drawDC(6, i+12);
     break;
   case 5:
     // Draw 5
     drawDC(5, i+12);
     break;
   case 4:
     // Draw 4
     drawDC(4, i+12);
     break;
   case 3:
     // Draw 3
     drawDC(3, i+12);
     break;
   case 2:
     // Draw 2
     drawDC(2, i+12);
     break;
   case 1:
     // Draw 1
     drawDC(1, i+12);
     break;
   case 0:
     // Draw 0
     drawDC(0, i+12);
     break;

    }
  }
}
void drawWave(uint32_t duty)
{
  // If the previous duty cycle is the same as the current, exit method
  if(duty == prevDuty)
  {
    prevDuty = duty;
              return;
  }
```

```
// New Duty Cycle input, clear wave
// For loop to clear the waveform area of the LCD
for(uint16_t vert = 11; vert < 27; vert++){
  for(uint16_t horiz = 0; horiz < 9; horiz++){
                // Send graphics address
                graphicPosition(horiz, vert);
                // Send pixel fill data
        write(0x00); // First 8-bits
                write(0x00); // Second 8-bits
  }
}

// Duty Cycle split into 16 bit blocks
// Duty 60 = 3.75 blocks = 3 blocks
blocks = duty / 16;
graphicPosition(0, 26); //25
for(int i = 0; i < 2; i++)
{
  write(0xFF);
}
//draw the rising edge
for(int i = 0; i < 15; i++)
{
  // 16 pixel vertical line starting from 12
  // Start of waveform at 1, add offset to center block
  graphicPosition(1, 26-i);
  drawPixel(0);
}
//MSD
// Remainder is 75 - (4*16) = 75 - 64 = 11
// 3 full blocks will be filled, then 12 pixels in the last block
remainder = duty - (blocks*16);

// Set graphics position to start of waveform
graphicPosition(1,11);

// For loop to fill all blocks in the top line
for(int i = 0; i < blocks; i++)
{
  write(0xFF);
  write(0xFF);
}

// Draws the remaining pixels in the block for the top line
drawRem(remainder);


// Draw Middle Line
// Middle line is at the duty cycle value = 75
// Line will be in block 3
for(int i = 0; i < 15; i++)
{
  // 15 pixel vertical line starting from 12
  // Start of waveform at 1, add offset to center block
  graphicPosition( (1+blocks), 12+i);
  drawPixel(remainder);
}

// Draw bottom line
// Fill rest of current block based on inverse of remainder
// Bottom line at vertical 26
graphicPosition( (1+blocks), 26);
drawLowRem((17-remainder)); //17-11=5 pixels

// Fill rest of bottom line, total blocks = 7
// 4 blocks filled so far
// Fill blocks 5, 6, and first 4 bits of 7
// 3 blocks remaining to fill for DC = 75
```

```
  // Set graphics position to proper block
  graphicPosition( ((1+blocks)+1), 26);
  // For loop to fill blocks through block 6
  for(int i = 0; i < 6 - (blocks + 1); i++)
  {
    write(0xFF);
    write(0xFF);
  }

  // Last block (7th) only needs 4 bits
  // This instruction is always executed because duty cycle only changes from 20-80%
  // Change this if duty cycle values exceed 96%
  // First 6 blocks = 96 pixels
  write(0xF0);

    for(int i = 0; i < 14; i++)
  {
    temp2 = blocks;
    // 15 pixel vertical line starting from 12
    // Start of waveform at 1, add offset to center block
    graphicPosition(7, 25-i);
    //drawPixel(4);
                write(0x10);
    write(0x00);
  }
  graphicPosition(7, 11);
  //drawRem(13);
  //drawLowRem(13);
  write(0x1F);
  write(0xFF);

}
// Method for drawing remainder of top line of waveform
void drawRem(unsigned int rem)
{
  switch (rem)
  {
    case 0:
          // Print nothing
          write(0x00);
          write(0x00);
          break;
    case 1:
          // Print 1 pixel
          write(0x80);
          write(0x00);
          break;
    case 2:
          // Print 2 pixels
          write(0xC0);
          write(0x00);
          break;
    case 3:
          // Print 3 pixels
          write(0xE0);
          write(0x00);
          break;
    case 4:
          // Print 4 pixels
          write(0xF0);
          write(0x00);
          break;
    case 5:
          // Send 5 pixels
          write(0xF8);
          write(0x00);
          break;
    case 6:
```

```
            write(0xFC);
            write(0x00);
            break;
    case 7:
            write(0xFE);
            write(0x00);
            break;
    case 8:
            write(0xFF);
            write(0x00);
            break;
    case 9:
            write(0xFF);
            write(0x80);
            break;
    case 10:
            write(0xFF);
            write(0xC0);
            break;
    case 11:
            write(0xFF);
            write(0xE0);
            break;
    case 12:
            write(0xFF);
            write(0xF0);
            break;
    case 13:
            write(0xFF);
            write(0xF8);
            break;
    case 14:
            write(0xFF);
            write(0xFC);
            break;
    case 15:
            write(0xFF);
            write(0xFE);
            break;
  }
}
// Method for drawing vertical line of waveform
void drawPixel(unsigned int rem)
{
  switch (rem)
    {
      case 0:
            // If there is no remainder, print line on 16th bit of block
            write(0x80);
            write(0x00);
            break;
      case 1:
            // Print 1 pixel
            write(0x80);
            write(0x00);
            break;
      case 2:
            // Print 2 pixels
            write(0x40);
            write(0x00);
            break;
      case 3:
            // Print 3 pixels
            write(0x20);
            write(0x00);
            break;
      case 4:
            // Print 4 pixels
```

```
            write(0x10);
            write(0x00);
            break;
      case 5:
            // Send 5 pixels
            write(0x08);
            write(0x00);
            break;
      case 6:
            write(0x04);
            write(0x00);
            break;
      case 7:
            write(0x02);
            write(0x00);
            break;
      case 8:
            write(0x01);
            write(0x00);
            break;
      case 9:
            write(0x00);
            write(0x80);
            break;
      case 10:
            write(0x00);
            write(0x40);
            break;
      case 11:
            write(0x00);
            write(0x20);
            break;
      case 12:
            write(0x00);
            write(0x10);
            break;
      case 13:
            write(0x00);
            write(0x08);
            break;
      case 14:
            write(0x00);
            write(0x04);
            break;
      case 15:
            write(0x00);
            write(0x02);
            break;
    }
}
// Method for drawing remainder of bottom line of waveform after vertical line
void drawLowRem(unsigned int rem)
{
  switch (rem)
  {
    case 0:
          // Print nothing
          write(0xFF);
          write(0xFF);
          break;
    case 1:
          // Print 1 pixel
          write(0x00);
          write(0x01);
          break;
    case 2:
          // Print 2 pixels
          write(0x00);
```

```
            write(0x03);
            break;
    case 3:
            // Print 3 pixels
            write(0x00);
            write(0x07);
            break;
    case 4:
            // Print 4 pixels
            write(0x00);
            write(0x0F);
            break;
    case 5:
            // Send 5 pixels
            write(0x00);
            write(0x1F);
            break;
    case 6:
            write(0x00);
            write(0x3F);
            break;
    case 7:
            write(0x00);
            write(0x7F);
            break;
    case 8:
            write(0x00);
            write(0xFF);
            break;
    case 9:
            write(0x01);
            write(0xFF);
            break;
    case 10:
            write(0x03);
            write(0xFF);
            break;
    case 11:
            write(0x07);
            write(0xFF);
            break;
    case 12:
            write(0x0F);
            write(0xFF);
            break;
    case 13:
            write(0x1F);
            write(0xFF);
            break;
    case 14:
            write(0x3F);
            write(0xFF);
            break;
    case 15:
            write(0x7F);
            write(0xFF);
            break;
    case 16:
          write(0xFF);
          write(0xFF);
          break;
  }
}

void command(char i)
{
  GPIOD->ODR = i; //put data on output Port
  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0, GPIO_PIN_RESET); //PF0 = RS = LOW : send instruction, AO
```

```
  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_1, GPIO_PIN_RESET); // /PF1= R/W, WR in 8080 mode; R/W in 6800 mode, WRT
  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_2, GPIO_PIN_SET); //PF2 = E, active low, CS activated, CS1
  HAL_Delay (1);
  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_2, GPIO_PIN_RESET); //PF2 = E, active low, CS activated, CS1

}
void write (char i)
{
  GPIOD->ODR = i; //put data on output Port
  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0, GPIO_PIN_SET); //PF0 = RS = LOW : send instruction, AO
  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_1, GPIO_PIN_RESET); // /PF1= R/W, WR in 8080 mode; R/W in 6800 mode, WRT
  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_2, GPIO_PIN_SET); //PF2 = E, active low, CS activated, CS1
  HAL_Delay (1);
  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_2, GPIO_PIN_RESET); //PF2 = E, active low, CS activated, CS1
  HAL_Delay(1);

}
void init ()
{
  HAL_Delay (100);
  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_3, GPIO_PIN_SET); // ??, PF3 = PSB, ??
//  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_2, GPIO_PIN_RESET); //PF2 = E, active low, CS activated, CS1
  //HAL_Delay (100); //Wait >15 msec after power is applied
  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_4, GPIO_PIN_RESET); //PF2 = E, active low, CS activated, CS1
  HAL_Delay (150); //Wait >15 msec after power is applied
  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_4, GPIO_PIN_SET); //PF2 = E, active low, CS activated, CS1
  command(0x30);              //command 0x30 = Wake up
  HAL_Delay (1);             //Wait time >100uS
  command(0x30);              //command 0x30 = Wake up #2
  //command(0x34);              //Function set: 8-bit/RE=1: extended instruction
  HAL_Delay (1);  //must wait 160us, busy flag not available
  command(0x0C);
  HAL_Delay (1);
  command(0x01);              //Clear dispaly
  HAL_Delay (15);
  command(0x06);              //Entry mode set
  HAL_Delay (1);


}

/* USER CODE END 4 */

/**
  * @brief  Period elapsed callback in non blocking mode
  * @note   This function is called  when TIM7 interrupt took place, inside
  * HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to increment
  * a global variable "uwTick" used as application time base.
  * @param  htim : TIM handle
  * @retval None
  */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* USER CODE BEGIN Callback 0 */

  /* USER CODE END Callback 0 */
  if (htim->Instance == TIM7) {
    HAL_IncTick();
  }
  /* USER CODE BEGIN Callback 1 */

  /* USER CODE END Callback 1 */
}

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
```

```
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */

  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

## stm32f4xx_it.c

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file    stm32f4xx_it.c
  * @brief   Interrupt Service Routines.
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f4xx_it.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN TD */

/* USER CODE END TD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
```

```
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */
//double pulsewidth = 0;

//EXTI
extern int halfSeconds;

//TIM5
int count1s = 0;
int count1s2 = 0;

int count1s, seconds = 0;
int count = 0;
int count2 = 0;
int count3 = 0;
uint32_t risingedge1 = 0x0000;
uint32_t fallingedge = 0x0000;
uint32_t risingedge2 = 0x0000;
extern float Hi_pulsewidth ;
extern float Low_pulsewidth ;
extern float frequency;
extern float dutyCycle;
extern float timePeriod;

float frequency_KHz;
int count11 = 0;
int countHi;
int countLow;
int countPeriod;
uint32_t risingedge11 = 0;
uint32_t fallingedge11 = 0;
uint32_t risingedge22 = 0;
extern float Hi_pulsewidth11;
extern float Low_pulsewidth11;
extern float distance;
int count20us = 0;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/* External variables --------------------------------------------------------*/
extern TIM_HandleTypeDef htim2;
extern TIM_HandleTypeDef htim3;
extern TIM_HandleTypeDef htim4;
extern TIM_HandleTypeDef htim5;
extern TIM_HandleTypeDef htim7;

/* USER CODE BEGIN EV */

/* USER CODE END EV */

/******************************************************************************/
/*           Cortex-M4 Processor Interruption and Exception Handlers          */
/******************************************************************************/
/**
  * @brief This function handles Non maskable interrupt.
```

```
  */
void NMI_Handler(void)
{
  /* USER CODE BEGIN NonMaskableInt_IRQn 0 */

  /* USER CODE END NonMaskableInt_IRQn 0 */
  /* USER CODE BEGIN NonMaskableInt_IRQn 1 */

  /* USER CODE END NonMaskableInt_IRQn 1 */
}

/**
  * @brief This function handles Hard fault interrupt.
  */
void HardFault_Handler(void)
{
  /* USER CODE BEGIN HardFault_IRQn 0 */

  /* USER CODE END HardFault_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_HardFault_IRQn 0 */
    /* USER CODE END W1_HardFault_IRQn 0 */
  }
}

/**
  * @brief This function handles Memory management fault.
  */
void MemManage_Handler(void)
{
  /* USER CODE BEGIN MemoryManagement_IRQn 0 */

  /* USER CODE END MemoryManagement_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_MemoryManagement_IRQn 0 */
    /* USER CODE END W1_MemoryManagement_IRQn 0 */
  }
}

/**
  * @brief This function handles Pre-fetch fault, memory access fault.
  */
void BusFault_Handler(void)
{
  /* USER CODE BEGIN BusFault_IRQn 0 */

  /* USER CODE END BusFault_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_BusFault_IRQn 0 */
    /* USER CODE END W1_BusFault_IRQn 0 */
  }
}

/**
  * @brief This function handles Undefined instruction or illegal state.
  */
void UsageFault_Handler(void)
{
  /* USER CODE BEGIN UsageFault_IRQn 0 */

  /* USER CODE END UsageFault_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_UsageFault_IRQn 0 */
    /* USER CODE END W1_UsageFault_IRQn 0 */
```

```
  }
}

/**
  * @brief This function handles System service call via SWI instruction.
  */
void SVC_Handler(void)
{
  /* USER CODE BEGIN SVCall_IRQn 0 */

  /* USER CODE END SVCall_IRQn 0 */
  /* USER CODE BEGIN SVCall_IRQn 1 */

  /* USER CODE END SVCall_IRQn 1 */
}

/**
  * @brief This function handles Debug monitor.
  */
void DebugMon_Handler(void)
{
  /* USER CODE BEGIN DebugMonitor_IRQn 0 */

  /* USER CODE END DebugMonitor_IRQn 0 */
  /* USER CODE BEGIN DebugMonitor_IRQn 1 */

  /* USER CODE END DebugMonitor_IRQn 1 */
}

/**
  * @brief This function handles Pendable request for system service.
  */
void PendSV_Handler(void)
{
  /* USER CODE BEGIN PendSV_IRQn 0 */

  /* USER CODE END PendSV_IRQn 0 */
  /* USER CODE BEGIN PendSV_IRQn 1 */

  /* USER CODE END PendSV_IRQn 1 */
}

/**
  * @brief This function handles System tick timer.
  */
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */
    count20us+=1;
    if (count20us == 1)
  {
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_2, GPIO_PIN_RESET);//Set PC2 for 20us
  }
    if (count20us == (2500-1))
  {
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_2, GPIO_PIN_SET);//Set PC2 for 20us
    count20us = 0;
  }


  /* USER CODE END SysTick_IRQn 0 */

  /* USER CODE BEGIN SysTick_IRQn 1 */

  /* USER CODE END SysTick_IRQn 1 */
}

/*****************************************************************************/
```

SysTick->CTRL = 0x0005; //"Choose the undividided clock source, disable the
SysTick->LOAD = 2000; //ticks Number of ticks between two interrupts = 20us*100MHz= 2000
SysTick->VAL = 0x000000;; // Current counter value
SysTick->CTRL |= 0x0007; //The SysTick CTRL register enables the SysTick features

```
/* STM32F4xx Peripheral Interrupt Handlers                        */
/* Add here the Interrupt Handlers for the used peripherals.      */
/* For the available peripheral interrupt handler names,          */
/* please refer to the startup file (startup_stm32f4xx.s).        */
/******************************************************************************/

/**
  * @brief This function handles EXTI line[9:5] interrupts.
  */
void EXTI9_5_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI9_5_IRQn 0 */

  if ((!(GPIOE->IDR & 0x80)) && (halfSeconds == 0)) // Falling Edge detection, start PE7 (EXTI7)
  {
   HAL_TIM_OC_Start_IT(&htim3, TIM_CHANNEL_4); // Start Buzzer and indicator
   HAL_GPIO_WritePin(GPIOG, GPIO_PIN_12, GPIO_PIN_SET);// Forward direction control_Pin15: PG12

   //HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_3); //Starts the TIM Input Capture measurement in interrupt mode.

   // HC-SR04 ultrasonic sensor
//   HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_4); //Starts the TIM Input Capture measurement in interrupt mode.
//   HAL_TIM_PWM_Start(&htim5, TIM_CHANNEL_3);// to make a pulse with 25 us width and 55 ms period. or
HAL_TIM_PWM_Start(&htim5, TIM_CHANNEL_4); check it !!!

  }
  if (!(GPIOE->IDR & 0x100)) // Falling Edge detection on Stop pin  PE8(EXTI8); wait for start
  {
   HAL_GPIO_WritePin(GPIOG, GPIO_PIN_12, GPIO_PIN_RESET);// Forward direction control_Pin15: PG12
   HAL_GPIO_WritePin(GPIOG, GPIO_PIN_11, GPIO_PIN_RESET);//Reverse direction control_Pin16: PG11
   halfSeconds = 0;
   HAL_TIM_PWM_Stop_IT(&htim4, TIM_CHANNEL_4);// stop PWM
   HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_RESET);//TURN Off Red LED (IR Emitter LED)
  }


  /* USER CODE END EXTI9_5_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(Start_Input_Pin);
  HAL_GPIO_EXTI_IRQHandler(Stop_Input_Pin);
  /* USER CODE BEGIN EXTI9_5_IRQn 1 */

  /* USER CODE END EXTI9_5_IRQn 1 */
}

/**
  * @brief This function handles TIM2 global interrupt.
  */
void TIM2_IRQHandler(void)
{
  /* USER CODE BEGIN TIM2_IRQn 0 */
 if ((HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_11)) && (count11 == 0)) // Rising-1 edge detection
  {

   risingedge11 = TIM2->CCR4; // record the count11er value
   count11 = 1;
  }
  else if((!(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_11))) && (count11 == 1))// Falling Edge detection
  {
   fallingedge11 = TIM2->CCR4; //record the count11er value
   count11 = 2;
  }
  else if ((HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_11)) && (count11 == 2)) // Rising-1 edge detection
  {

   risingedge22 = TIM2->CCR4; // record the count11er value
   countHi = (fallingedge11 - risingedge11);
   countLow = (risingedge22 - fallingedge11);
   countPeriod = countHi + countLow;
```

```
    Hi_pulsewidth11 = ((fallingedge11 - risingedge11)* 0.02); // Hi_Pulsewidth (us)= CCRx * Clock period,
    distance = ((Hi_pulsewidth11 *  0.0068 ) - 0.3026); // Distance (in) equation culculated from calibration graph
    TIM2->CNT = 0; // Reset the count11er register
    count11 = 0;
  }

  /* USER CODE END TIM2_IRQn 0 */
  HAL_TIM_IRQHandler(&htim2);
  /* USER CODE BEGIN TIM2_IRQn 1 */

  /* USER CODE END TIM2_IRQn 1 */
}

/**
  * @brief This function handles TIM3 global interrupt.
  */
void TIM3_IRQHandler(void)
{
  /* USER CODE BEGIN TIM3_IRQn 0 */

  if (halfSeconds < 12)
  {
    if (count2 == 0)
    {
      TIM3->CCR4 += 4545;// Tcnt for one periode = (1/f) * fapb1_clock = (1/5.5kHz)* 50MHz = 9090
      count2 = 1;
    }
    else if (count2 == 1)
    {
    TIM3->CCR4 += (9090-4545);// 50% duty cycle:(50/100)*9090=4545
    count2 = 0;
    count1s += 1;
    }
    count3= 1;
  }
  if ((count1s == 2750) && (count3 != 2)) // To mak 1 halfSeconds, 1/5.5KHz =0.1818ms; 0.1818*X/2 = 1000ms; x = 5500/2=2750
  //else if ((count1s == 2750) && (halfSeconds <= 12)) // To mak 1 halfSeconds, 1/5.5KHz =0.1818ms; 0.1818*X/2 = 1000ms; x =
5500/2=2750
  {
    count1s = 0;
    HAL_GPIO_TogglePin(GPIOG, GPIO_PIN_13); //Toggle GREEN LED
    halfSeconds +=1;

  }

  ////////////////////////////////////////////////////
  //if (((GPIOE->IDR & 0x80)) && (halfSeconds == 12))// Falling Edge is not detected on start and after 12 halfSeconds
  if (((count3== 1)) && (halfSeconds == 12))// Falling Edge is not detected on start and after 12 halfSeconds
  //if (count3 == 1)
  {
    HAL_TIM_OC_Stop_IT(&htim3, TIM_CHANNEL_4); // Turn off Buzzer
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_SET);//TURN ON Red LED (IR Emitter LED)

    //PWM signal driving the motor on
    HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_4);// to make a pulse with f= 30KHz and duty cycle=50%  0.016666 ms width and
0.03333 ms period.
    TIM4-> ARR = 1666; // (1/30KHz)*50MHz=1666
    TIM4-> CCR4 = 833; // (1/2*30KHz)*50MHz=833; 50% dutyc cycle
    count3= 2;
  }
////////////////////////////////////////////////////////////////

//  if ((TIM4->CCR4 == 0) && ( 12 <= halfSeconds && halfSeconds <= 32))
  //if ((TIM4->CCR4 == 0) && (halfSeconds <= 32))
  //count1s2 = 0;
  if (TIM4->CCR4 == 0)
  {
    if (count2 == 0)
```

```
    {
      TIM3->CCR4 += 7143;// Tcnt for one periode = (1/f) * fapb1_clock = (1/3.5kHz)* 50MHz = 14286
      count2 = 1;
    }
    else if (count2 == 1)
    {
    TIM3->CCR4 += (14286-7143);// 50% duty cycle:(50/100)*14286=7143
    count3 = 0;
    count1s2 += 1;
    }
  }
   if((count1s2 == 1375) && (halfSeconds <= 32)) // To make 1 halfSeconds, 1/3.5KHz =0.2857ms; 0.2857*2X = 1000ms; x = 3500/4 for 2
blinks/sec
     {
       count1s2 = 0;
       HAL_GPIO_TogglePin(GPIOG, GPIO_PIN_13); //Toggle GREEN LED
       halfSeconds +=1;
     }

  else if ( halfSeconds > 32)
  {
   HAL_TIM_OC_Stop_IT(&htim3, TIM_CHANNEL_4); // Stop Buzzer Start
   HAL_GPIO_WritePin(GPIOG, GPIO_PIN_13, GPIO_PIN_RESET);//TURN Off Green LED (Indicator LED)
   HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_RESET);//TURN Off Red LED (IR Emitter LED)
   halfSeconds = 0;
   count2 = 0;
   count3 = 0;
   HAL_GPIO_WritePin(GPIOG, GPIO_PIN_12, GPIO_PIN_RESET);//PG12 on, move Forward
  }

  ////////////////////////////////////////////////////////////////////////////
//    if ((GPIOB->IDR & 0x01) && (count == 0)) // Rising-1 edge detection
//  {
//    risingedge1 = TIM3->CCR3;
//    count = 1;
//  }
//   else if ((!(GPIOB->IDR & 0x01)) && (count == 1))// Falling Edge detection
//  {
//    fallingedge = TIM3->CCR3;
//    count = 2;
//  }
//  else if ((GPIOB->IDR & 0x01) && (count == 2)) //Rising-2 edge detection
//  {
//    risingedge2 = TIM3->CCR3;
//    timePeriod = ((risingedge2 - risingedge1)*0.02) ; // HLCK=100MHz  and APB1=50MHz. Period (us).1/50MHz = 0.02us
//    frequency = (1000000/timePeriod); // frequency (HZ)=1/timePeriod(us)= 10*6/timePeriod
//    Hi_pulsewidth = ((fallingedge - risingedge1)* 0.02); // Hi_Pulsewidth (ms).
//
//    Low_pulsewidth = ((risingedge2 - fallingedge) * 0.02); // Low_pulsewidth (ms).
//    //printf("Period = %lf msec\n", timePeriod);
//    dutyCycle = (Hi_pulsewidth/timePeriod)*100;
//
//
//    count = 0;
//    TIM3->CNT = 0; // Reset the counter register
//
//    frequency_KHz = frequency/1000;
//  }
////////////////////////////////////////////////////////////////////////////

  /* USER CODE END TIM3_IRQn 0 */
  HAL_TIM_IRQHandler(&htim3);
  /* USER CODE BEGIN TIM3_IRQn 1 */

  /* USER CODE END TIM3_IRQn 1 */
}

/**
```

```
  * @brief This function handles TIM4 global interrupt.
  */
void TIM4_IRQHandler(void)
{
  /* USER CODE BEGIN TIM4_IRQn 0 */

  /* USER CODE END TIM4_IRQn 0 */
  HAL_TIM_IRQHandler(&htim4);
  /* USER CODE BEGIN TIM4_IRQn 1 */

  /* USER CODE END TIM4_IRQn 1 */
}

/**
  * @brief This function handles TIM5 global interrupt.
  */
void TIM5_IRQHandler(void)
{
  /* USER CODE BEGIN TIM5_IRQn 0 */


if ((GPIOA->IDR & 0x0008) && (count == 0)) // Rising-1 edge detection
  {
    risingedge1 = TIM5->CCR4;
    count = 1;
  }
  else if ((!(GPIOA->IDR & 0x0008)) && (count == 1))// Falling Edge detection
  {
    fallingedge = TIM5->CCR4;
    count = 2;
  }
else if ((GPIOA->IDR & 0x0008) && (count == 2)) //Rising-2 edge detection
  {
    risingedge2 = TIM5->CCR4;
    timePeriod = ((risingedge2 - risingedge1)*0.02) ; // HLCK=100MHz  and APB1=50MHz. Period (us).1/50MHz = 0.02us
    frequency = (1000000/timePeriod); // frequency (HZ)=1/timePeriod(us)= 10*6/timePeriod
    Hi_pulsewidth = ((fallingedge - risingedge1)* 0.02); // Hi_Pulsewidth (ms).

    Low_pulsewidth = ((risingedge2 - fallingedge) * 0.02); // Low_pulsewidth (ms).
    //printf("Period = %lf msec\n", timePeriod);
    dutyCycle = (Hi_pulsewidth/timePeriod)*100;


    count = 0;
    TIM5->CNT = 0; // Reset the counter register

    frequency_KHz = frequency/1000;
  }
////////////////////////////////////////////////////////////////////////////////
//  while (seconds <= 5)
//  {
//    if (count2 == 0)
//    {
//      TIM5->CCR1 += 4545;// Tcnt for one periode = (1/f) * fapb1_clock = (1/5.5kHz)* 50MHz = 9090
//      count2 = 1;
//    }
//    else if (count2 == 1)
//    {
//    TIM5->CCR1 += (9090-4545);// 50% duty cycle:(50/100)*9090=4545
//    count2 = 0;
//    count1s += 1;
//    }
//    if(count1s == 5500) // To mak 1 seconds, 1/5.5KHz =0.1818ms; 0.1818*X = 1000ms; x = 5500
//      {
//        count1s = 0;
//        HAL_GPIO_TogglePin(GPIOG, Indicator_GreenLED_Pin); //Toggle GREEN LED
//        seconds +=1;
//      }
```

```
//  }
//
//
//
//  while (5 < seconds && seconds <= 16)
//  {
//    if (count2 == 0)
//    {
//      TIM5->CCR1 += 7143;// Tcnt for one periode = (1/f) * fapb1_clock = (1/3.5kHz)* 50MHz = 14286
//      count2 = 1;
//    }
//    else if (count2 == 1)
//    {
//    TIM5->CCR1 += (14286-7143);// 50% duty cycle:(50/100)*14286=7143
//    count2 = 0;
//    count1s += 1;
//    }
//    if(count1s == 1750) // To mak 1 seconds, 1/3.5KHz =0.2857ms; 0.2857*X = 1000ms; x = 3500/2 for 2 blinks/sec
//      {
//        count1s = 0;
//        HAL_GPIO_TogglePin(GPIOG, Indicator_GreenLED_Pin); //Toggle GREEN LED
//        seconds +=1;
//      }
//  }
//
//  if ( seconds > 16)
//  {
//   HAL_TIM_OC_Stop_IT(&htim5, TIM_CHANNEL_1); // Stop Buzzer Start
//   HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_RESET);//TURN Off Red LED (IR Emitter LED)
//   seconds = 0;
//   count2 = 0;
//  }


  /* USER CODE END TIM5_IRQn 0 */
  HAL_TIM_IRQHandler(&htim5);
  /* USER CODE BEGIN TIM5_IRQn 1 */

  /* USER CODE END TIM5_IRQn 1 */
}

/**
  * @brief This function handles TIM7 global interrupt.
  */
void TIM7_IRQHandler(void)
{
  /* USER CODE BEGIN TIM7_IRQn 0 */

  /* USER CODE END TIM7_IRQn 0 */
  HAL_TIM_IRQHandler(&htim7);
  /* USER CODE BEGIN TIM7_IRQn 1 */

  /* USER CODE END TIM7_IRQn 1 */
}

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
```

## Main.lst

```
###############################################################################
#
# IAR ANSI C/C++ Compiler V9.20.2.320/W64 for ARM       15/Apr/2022  16:32:04
```

```
# Copyright 1999-2021 IAR Systems AB.
#
#   Cpu mode               =  thumb
#   Endian                 =  little
#   Source file            =
#       C:\Users\Student\OneDrive - Western Michigan
#       University\Documents\MIcrocontroller\Project\Core\Src\main.c
#   Command line           =
#       -f "C:\Users\Student\OneDrive - Western Michigan
#       University\Documents\MIcrocontroller\Project\EWARM\Project\Obj\Application\User\Core\main.o.rsp"
#       ("C:\Users\Student\OneDrive - Western Michigan
#       University\Documents\MIcrocontroller\Project\Core\Src\main.c" -D
#       USE_HAL_DRIVER -D STM32F429xx -lcN "C:\Users\Student\OneDrive -
#       Western Michigan
#       University\Documents\MIcrocontroller\Project\EWARM\Project\List\Application\User\Core"
#       -o "C:\Users\Student\OneDrive - Western Michigan
#       University\Documents\MIcrocontroller\Project\EWARM\Project\Obj\Application\User\Core"
#       --debug --endian=little --cpu=Cortex-M4 -e --fpu=VFPv4_sp
#       --dlib_config "C:\Program Files\IAR Systems\Embedded Workbench
#       9.0\arm\inc\c\DLib_Config_Full.h" -I "C:\Users\Student\OneDrive -
#       Western Michigan
#       University\Documents\MIcrocontroller\Project\EWARM/../Core/Inc\\" -I
#       "C:\Users\Student\OneDrive - Western Michigan
#       University\Documents\MIcrocontroller\Project\EWARM/../Drivers/STM32F4xx_HAL_Driver/Inc\\"
#       -I "C:\Users\Student\OneDrive - Western Michigan
#       University\Documents\MIcrocontroller\Project\EWARM/../Drivers/STM32F4xx_HAL_Driver/Inc/Legacy\\"
#       -I "C:\Users\Student\OneDrive - Western Michigan
#       University\Documents\MIcrocontroller\Project\EWARM/../Drivers/CMSIS/Device/ST/STM32F4xx/Include\\"
#       -I "C:\Users\Student\OneDrive - Western Michigan
#       University\Documents\MIcrocontroller\Project\EWARM/../Drivers/CMSIS/Include\\"
#       -Ohz) --dependencies=n "C:\Users\Student\OneDrive - Western Michigan
#       University\Documents\MIcrocontroller\Project\EWARM\Project\Obj\Application\User\Core\main.o.d"
#   Locale                 =  C
#   List file              =
#       C:\Users\Student\OneDrive - Western Michigan
#       University\Documents\MIcrocontroller\Project\EWARM\Project\List\Application\User\Core\main.lst
#   Object file            =
#       C:\Users\Student\OneDrive - Western Michigan
#       University\Documents\MIcrocontroller\Project\EWARM\Project\Obj\Application\User\Core\main.o
#   Runtime model:
#     __CPP_Runtime         =  1
#     __SystemLibrary       =  DLib
#     __dlib_file_descriptor =  1
#     __dlib_version        =  6
#     __size_limit          =  32768|ARM.EW.LINKER
#
###############################################################################

C:\Users\Student\OneDrive - Western Michigan University\Documents\MIcrocontroller\Project\Core\Src\main.c
      1        /* USER CODE BEGIN Header */
      2        /**
      3          ******************************************************************************
      4          * @file          : main.c
      5          * @brief         : Main program body
      6          ******************************************************************************
      7          * @attention
      8          *
      9          * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
     10           * All rights reserved.</center></h2>
     11           *
     12           * This software component is licensed by ST under BSD 3-Clause license,
     13           * the "License"; You may not use this file except in compliance with the
     14           * License. You may obtain a copy of the License at:
     15           *                        opensource.org/licenses/BSD-3-Clause
     16           *
     17           ******************************************************************************
     18           */
     19         /* USER CODE END Header */
```

```
20  /* Includes ------------------------------------------------------------------*/
21  #include "main.h"
22
23  /* Private includes ----------------------------------------------------------*/
24  /* USER CODE BEGIN Includes */
25  #include "stdio.h"
26  /* USER CODE END Includes */
27
28  /* Private typedef -----------------------------------------------------------*/
29  /* USER CODE BEGIN PTD */
30
31  /* USER CODE END PTD */
32
33  /* Private define ------------------------------------------------------------*/
34  /* USER CODE BEGIN PD */
35  /* USER CODE END PD */
36
37  /* Private macro -------------------------------------------------------------*/
38  /* USER CODE BEGIN PM */
39
40  /* USER CODE END PM */
41
42  /* Private variables ---------------------------------------------------------*/
43  TIM_HandleTypeDef htim2;
44  TIM_HandleTypeDef htim3;
45  TIM_HandleTypeDef htim4;
46  TIM_HandleTypeDef htim5;
47
48  /* USER CODE BEGIN PV */
49  int halfSeconds = 0;
50  int count4 = 0;
51  int freq;
52  int dist;
53
54  float distance = 0x0000;
55  uint8_t one = 0x0000;
56  uint8_t two = 0x0000;
57  uint8_t three = 0x0000;
58  //uint16_t i = 0x0000;
59  uint16_t R = 0x0000;
60  float Hi_pulsewidth = 0x0000;
61  float Hi_pulsewidth11 = 0x0000; // for Ultrasonic sensor to measure distance
62  float Low_pulsewidth= 0x0000;
63  float frequency = 0x0000;
64  float dutyCycle = 0x0000;
65  float timePeriod = 0x0000;
66  char temp;
67  uint16_t temp2;
68  uint16_t temp3;
69  uint32_t prevDuty = 0;
70  unsigned int blocks = 0;
71  unsigned int remainder = 0;
72  unsigned int ref, pos = 0;
73  //uint32_t freq;
74  unsigned int fd1, fd2, fd3, fd4, fd5, fd6 = 0;
75  char digits[6] = {0,0,0,0,0,0};
76  uint32_t duty;
77  char dcd1, dcd2 = 0;
78  char dcds[2] = {0,0};
79  unsigned int pulse = 0;
80  unsigned int t1, t2, t3;
81  char T[3] = {0,0,0};
82  unsigned int p1, p2, p3;
83  char P[3] = {0,0,0};
84  unsigned int bonus = 0;
85
86  /* USER CODE END PV */
87
```

```
88  /* Private function prototypes -----------------------------------------------*/
89  void SystemClock_Config(void);
90  static void MX_GPIO_Init(void);
91  static void MX_TIM3_Init(void);
92  static void MX_TIM2_Init(void);
93  static void MX_TIM5_Init(void);
94  static void MX_TIM4_Init(void);
95  /* USER CODE BEGIN PFP */
96
97  /* USER CODE END PFP */
98
99  /* Private user code ---------------------------------------------------------*/
100  /* USER CODE BEGIN 0 */
101  void command (char i);
102  void write(char i);
103  void init();
104  static void displayDist(int);
105  static void displayFreq(int);
106  static void displayDC(int);
107  void resetDisplay();
108  char number (int);
109  void graphicPosition(uint16_t, uint16_t );
110  void initGraphicMode(void);
111  void clearGFX();
112  void drawWave(uint32_t);
113  void drawRem(unsigned int);
114  void drawLowRem(unsigned int);
115  void drawPixel(unsigned int);
116  void drawFreq(uint32_t);
117  void drawDuty(int);
118  void drawPWT(unsigned int, unsigned int);
119  /* USER CODE END 0 */
120
121  /**
122    * @brief  The application entry point.
123    * @retval int
124    */
125  int main(void)
126  {
127    /* USER CODE BEGIN 1 */
128
129    /* USER CODE END 1 */
130
131    /* MCU Configuration--------------------------------------------------------*/
132
133    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
134    HAL_Init();
135
136    /* USER CODE BEGIN Init */
137
138    /* USER CODE END Init */
139
140    /* Configure the system clock */
141    SystemClock_Config();
142
143    /* USER CODE BEGIN SysInit */
144
145    /* USER CODE END SysInit */
146
147    /* Initialize all configured peripherals */
148    MX_GPIO_Init();
149    MX_TIM3_Init();
150    MX_TIM2_Init();
151    MX_TIM5_Init();
152    MX_TIM4_Init();
153    /* USER CODE BEGIN 2 */
154    GPIOE->IDR  = 0x80; //start is off when the power is On at the first time
155    GPIOG->IDR = 0x0000;
```

```
156         GPIOG->ODR = 0x0000;
157         HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_4); //Starts the TIM Input Capture measurement in interrupt mode.
158         //HAL_TIM_PWM_Start(&htim5, TIM_CHANNEL_3);// to make a pulse with 25 us width and 55 ms period. or
HAL_TIM_PWM_Start(&htim5, TIM_CHANNEL_4); check it !!!
159
160         SysTick->CTRL = 0x0005; //"Choose the undividided clock source, disable the
161         SysTick->LOAD = 2000; //ticks Number of ticks between two interrupts = 20us*100MHz= 2000
162         SysTick->VAL = 0x000000;; // Current counter value
163         SysTick->CTRL |= 0x0007; //The SysTick CTRL register enables the SysTick features
164
165         //HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_3); //Starts the TIM Input Capture measurement in interrupt mode.
166         init(); // initialize the LCD
167         resetDisplay(); // reset dispaly
168         initGraphicMode(); // initialize in graphic Mode
169           HAL_Delay(10);
170           clearGFX(); // clear Dispaly
171         /* USER CODE END 2 */
172
173         /* Infinite loop */
174         /* USER CODE BEGIN WHILE */
175         while (1)
176         {
177           /* USER CODE END WHILE */
178
179           /* USER CODE BEGIN 3 */
180            //conver float values to int
181
182            freq = (int) frequency;
183            int DC = (int) dutyCycle;
184            int period = (int) timePeriod;
185            int pulse = (int) Hi_pulsewidth;
186            dist = (int) distance;
187
188            if (freq == 4700) //frequency_KHz == 4.7
189           {
190             TIM4-> CCR4 = 1499; // (1666*90/100); 90% dutyc cycle
191           }
192
193           if (freq == 4900)//frequency_KHz == 4.9
194           {
195             TIM4-> CCR4 = 1332; // (1666*80/100); 80% dutyc cycle
196           }
197
198           if (freq == 5100)//frequency_KHz == 5.1
199           {
200             TIM4-> CCR4 = 1166; // (1666*70/100); 70% dutyc cycle
201           }
202
203           if (freq == 5300)//frequency_KHz == 5.3
204           {
205             TIM4-> CCR4 = 999; // (1666*60/100); 60% dutyc cycle
206           }
207
208           if (freq == 5500)//frequency_KHz == 5.5
209           {
210             TIM4-> CCR4 = 833; // (1666*50/100); 50% dutyc cycle
211           }
212
213           if (freq == 5700)//frequency_KHz == 5.7
214           {
215             TIM4-> CCR4 = 666; // (1666*40/100); 40% dutyc cycle
216           }
217
218           if (freq == 5900)//frequency_KHz == 5.9
219           {
220             TIM4-> CCR4 = 499; // (1666*30/100); 30% dutyc cycle
221           }
222
```

```
223             if (freq == 6100)//frequency_KHz == 6.1
224             {
225               TIM4-> CCR4 = 333; // (1666*20/100); 20% dutyc cycle
226             }
227
228             if (freq == 6300)//frequency_KHz == 6.3
229             {
230               TIM4-> CCR4 = 166; // (1666*10/100); 10% dutyc cycle
231             }
232
233             if (freq  > 6300) // end position detected; PWM signal should be turned off
234             {
235               count4 = 2;
236               TIM4-> CCR4 = 0; // (1666*0/100); 0% dutyc cycle
237               if (12 <= halfSeconds && halfSeconds < 33)
238               {
239                 HAL_TIM_OC_Start_IT(&htim3, TIM_CHANNEL_4); // Start Buzzer and indicator
240               }
241
242               //HAL_TIM_PWM_Stop_IT(&htim4, TIM_CHANNEL_4); // Stop PWM
243               //HAL_TIM_OC_Start_IT(&htim3, TIM_CHANNEL_4); // Start Buzzer
244
245             }
246
247         //    printf("\n.....\n");
248         //    printf("Hi_pulsewidth = %lf usec\n", Hi_pulsewidth);
249         //    printf("Low_pulsewidth = %lf usec\n", Low_pulsewidth);
250         //    printf("Period = %lf usec\n", timePeriod);
251         //    printf("Frequency = %lf HZ\n", frequency);
252         //    printf("Duty Cycle = %lf %\n", dutyCycle);
253         //    printf("Distance = %lf in\n", distance);
254         //    printf("\n.....\n");
255
256             command (0x30); // basic instruction set
257             HAL_Delay(1);
258             // project 2
259             //if ((GPIOE->IDR & 0x100) || (distance > 2) ) // Show moving forward when stop sw is not pressed and distance is larger than
1 inches.
260             // Show moving forward when stop sw is not pressed and distance is larger than 1 inche.and Forward direction control_Pin15:
PG12 is on.
261             if ((GPIOE->IDR & 0x100) && (distance > 2) && (GPIOG->IDR & 0x1000) ) //move forward is on, distance, stop sw is on
262              {
263               write ('M');
264               HAL_Delay (50);
265               write ('o');
266               HAL_Delay (50);
267               write ('v');
268               HAL_Delay (50);
269               write ('i');
270               HAL_Delay (50);
271               write ('n');
272               HAL_Delay (50);
273               write ('g');
274               HAL_Delay(3000);
275              }
276             // detect the end position by Ultrasonic or Photo receiver sensor 2_Pin6: PG9, Photo receiver sensor 1_Pin5: PG10  ( falling
edge detection by using polling technique)
277             //if ((distance < 2) || (!(GPIOG->IDR &  0x200)))
278               //if ((distance > 11) || (distance < 2) || (!(GPIOG->IDR &  0x200) || (!(GPIOG->IDR &  0x400))))
279               if ((!(GPIOG->IDR &  0x200) || (!(GPIOG->IDR &  0x400))))
280             {
281               TIM4-> CCR4 = 0; // (1666*0/100); 0% dutyc cycle
282               if (12 <= halfSeconds && halfSeconds < 33)
283               {
284                 HAL_TIM_OC_Start_IT(&htim3, TIM_CHANNEL_4); // Start Buzzer and indicator
285               }
286               HAL_GPIO_WritePin(GPIOG, GPIO_PIN_12, GPIO_PIN_RESET);//PG12 on, Stop moveing Forward, Forward direction
control_Pin15: PG12
```

```
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_11, GPIO_PIN_RESET);//PG12 on, Stop moveing Backward, Backward direction control_Pin16: PG11
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_13, GPIO_PIN_RESET);//TURN Off Green LED (Indicator LED)
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_RESET);//TURN Off Red LED (IR Emitter LED)
    resetDisplay();
    write ('E');
    HAL_Delay (50);
    write ('n');
    HAL_Delay (50);
    write ('d');
    HAL_Delay (50);
    write ('p');
    HAL_Delay (50);
    write ('o');
    HAL_Delay (50);
    write ('s');
    HAL_Delay (50);
    write ('i');
    HAL_Delay (50);
    write ('t');
    HAL_Delay (50);
    write ('o');
    HAL_Delay (50);
    write ('n');
    HAL_Delay(3000);
   }
   if ((distance < 6) && (distance > 5))
   {

    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_12, GPIO_PIN_RESET);//PG12 on, Stop moveing Forward, Forward direction control_Pin15: PG12
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_11, GPIO_PIN_RESET);//PG12 on, Stop moveing Backward, Backward direction control_Pin16: PG11
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_13, GPIO_PIN_RESET);//TURN Off Green LED (Indicator LED)
    //HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_RESET);//TURN Off Red LED (IR Emitter LED)
    resetDisplay();
    write ('H');
    HAL_Delay (50);
    write ('a');
    HAL_Delay (50);
    write ('l');
    HAL_Delay (50);
    write ('f');
    HAL_Delay (50);
    write ('p');
    HAL_Delay (50);
    write ('o');
    HAL_Delay (50);
    write ('s');
    HAL_Delay (50);
    write ('i');
    HAL_Delay (50);
    write ('t');
    HAL_Delay (50);
    write ('i');
    HAL_Delay (50);
    write ('o');
    HAL_Delay (50);
    write ('n');
    HAL_Delay(3000);

   }
   if (distance < 3)
   {

    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_12, GPIO_PIN_RESET);//PG12 on, Stop moveing Forward, Forward direction control_Pin15: PG12
```

```
350             HAL_GPIO_WritePin(GPIOG, GPIO_PIN_11, GPIO_PIN_RESET);//PG12 on, Stop moveing Backward, Backward
direction control_Pin16: PG11
351             HAL_GPIO_WritePin(GPIOG, GPIO_PIN_13, GPIO_PIN_RESET);//TURN Off Green LED (Indicator LED)
352             //HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_RESET);//TURN Off Red LED (IR Emitter LED)
353             resetDisplay();
354             write ('Q');
355             HAL_Delay (50);
356             write ('u');
357             HAL_Delay (50);
358             write ('a');
359             HAL_Delay (50);
360             write ('r');
361             HAL_Delay (50);
362             write ('t');
363             HAL_Delay (50);
364             write ('e');
365             HAL_Delay (50);
366             write ('r');
367             HAL_Delay (50);
368             write ('p');
369             HAL_Delay (50);
370             write ('o');
371             HAL_Delay (50);
372             write ('s');
373             HAL_Delay (50);
374             write ('i');
375             HAL_Delay(3000);
376             write ('t');
377             HAL_Delay(3000);
378             write ('o');
379             HAL_Delay(3000);
380             write ('n');
381             HAL_Delay(3000);
382           }
383           if (!(GPIOE->IDR & 0x100)) // stop botton pressed
384           {
385             TIM4-> CCR4 = 0; // (1666*0/100); 0% dutyc cycle
386             if (12 <= halfSeconds && halfSeconds < 33)
387             {
388               HAL_TIM_OC_Start_IT(&htim3, TIM_CHANNEL_4); // Start Buzzer and indicator
389             }
390             HAL_GPIO_WritePin(GPIOG, GPIO_PIN_12, GPIO_PIN_RESET);//PG12 on, Stop moveing Forward, Forward direction
control_Pin15: PG12
391             HAL_GPIO_WritePin(GPIOG, GPIO_PIN_11, GPIO_PIN_RESET);//PG12 on, Stop moveing Backward, Backward
direction control_Pin16: PG11
392
393             resetDisplay();
394             HAL_Delay(50);
395
396             write ('E');
397             HAL_Delay (50);
398             write ('m');
399             HAL_Delay (50);
400             write ('e');
401             HAL_Delay (50);
402             write ('r');
403             HAL_Delay (50);
404             write ('g');
405             HAL_Delay (50);
406             write ('e');
407             HAL_Delay (50);
408             write ('n');
409             HAL_Delay (50);
410             write ('c');
411             HAL_Delay (50);
412             write ('y');
413             HAL_Delay (50);
414             write (' ');
```

```
415             HAL_Delay (50);
416             write ('s');
417             HAL_Delay (50);
418             write ('t');
419             HAL_Delay (50);
420             write ('o');
421             HAL_Delay (50);
422             write ('p');
423             HAL_Delay (50);
424             HAL_Delay(3000);
425           }
426
427           displayDist (dist); // display ultrasonic distance on LCD
428
429         //    displayFreq (freq);
430         //    displayDC (DC);
431         //    initGraphicMode();
432         //    HAL_Delay(10);
433         //    clearGFX();
434         //    HAL_Delay(10);
435         //    drawWave(DC);
436         //    HAL_Delay(1);
437         //     // Draw Period Time 'PT' followed by 3 digit period in microseconds
438         //    // Draw Pulse Width 'PW' followed by 3 digit width in microseconds
439         //    drawPWT(period, pulse);
440         //    HAL_Delay(1);
441         //
442         //    // Display letter 'F' followed by six digit frequency in Hz
443         //    drawFreq(freq);
444         //    HAL_Delay(1);
445         //
446         //    // Display letters 'DC' followed by two digit duty cycle in %
447         //    drawDuty(duty);
448         //    HAL_Delay(3000);
449
450
451           }
452           /* USER CODE END 3 */
453         }
454
455         /**
456           * @brief System Clock Configuration
457           * @retval None
458           */
459         void SystemClock_Config(void)
460         {
461           RCC_OscInitTypeDef RCC_OscInitStruct = {0};
462           RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
463
464           /** Configure the main internal regulator output voltage
465           */
466           __HAL_RCC_PWR_CLK_ENABLE();
467           __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);
468           /** Initializes the RCC Oscillators according to the specified parameters
469           * in the RCC_OscInitTypeDef structure.
470           */
471           RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_LSI|RCC_OSCILLATORTYPE_HSE;
472           RCC_OscInitStruct.HSEState = RCC_HSE_ON;
473           RCC_OscInitStruct.LSIState = RCC_LSI_ON;
474           RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
475           RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
476           RCC_OscInitStruct.PLL.PLLM = 4;
477           RCC_OscInitStruct.PLL.PLLN = 100;
478           RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
479           RCC_OscInitStruct.PLL.PLLQ = 4;
480           if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
481           {
482             Error_Handler();
```

```
483         }
484         /** Initializes the CPU, AHB and APB buses clocks
485         */
486         RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
487                                     |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
488         RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
489         RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
490         RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
491         RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
492
493         if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
494         {
495           Error_Handler();
496         }
497       }
498
499       /**
500         * @brief TIM2 Initialization Function
501         * @param None
502         * @retval None
503         */
504       static void MX_TIM2_Init(void)
505       {
506
507         /* USER CODE BEGIN TIM2_Init 0 */
508
509         /* USER CODE END TIM2_Init 0 */
510
511         TIM_ClockConfigTypeDef sClockSourceConfig = {0};
512         TIM_MasterConfigTypeDef sMasterConfig = {0};
513         TIM_IC_InitTypeDef sConfigIC = {0};
514
515         /* USER CODE BEGIN TIM2_Init 1 */
516
517         /* USER CODE END TIM2_Init 1 */
518         htim2.Instance = TIM2;
519         htim2.Init.Prescaler = 0;
520         htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
521         htim2.Init.Period = 0xffffffff;
522         htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
523         htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
524         if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
525         {
526           Error_Handler();
527         }
528         sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
529         if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
530         {
531           Error_Handler();
532         }
533         if (HAL_TIM_IC_Init(&htim2) != HAL_OK)
534         {
535           Error_Handler();
536         }
537         sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
538         sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
539         if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
540         {
541           Error_Handler();
542         }
543         sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_BOTHEDGE;
544         sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
545         sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
546         sConfigIC.ICFilter = 0;
547         if (HAL_TIM_IC_ConfigChannel(&htim2, &sConfigIC, TIM_CHANNEL_4) != HAL_OK)
548         {
549           Error_Handler();
550         }
```

```
551       /* USER CODE BEGIN TIM2_Init 2 */
552
553       /* USER CODE END TIM2_Init 2 */
554
555     }
556
557     /**
558       * @brief TIM3 Initialization Function
559       * @param None
560       * @retval None
561       */
562     static void MX_TIM3_Init(void)
563     {
564
565       /* USER CODE BEGIN TIM3_Init 0 */
566
567       /* USER CODE END TIM3_Init 0 */
568
569       TIM_ClockConfigTypeDef sClockSourceConfig = {0};
570       TIM_MasterConfigTypeDef sMasterConfig = {0};
571       TIM_OC_InitTypeDef sConfigOC = {0};
572
573       /* USER CODE BEGIN TIM3_Init 1 */
574
575       /* USER CODE END TIM3_Init 1 */
576       htim3.Instance = TIM3;
577       htim3.Init.Prescaler = 0;
578       htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
579       htim3.Init.Period = 65535;
580       htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
581       htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
582       if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
583       {
584         Error_Handler();
585       }
586       sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
587       if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
588       {
589         Error_Handler();
590       }
591       if (HAL_TIM_OC_Init(&htim3) != HAL_OK)
592       {
593         Error_Handler();
594       }
595       sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
596       sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
597       if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
598       {
599         Error_Handler();
600       }
601       sConfigOC.OCMode = TIM_OCMODE_TOGGLE;
602       sConfigOC.Pulse = 0;
603       sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
604       sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
605       if (HAL_TIM_OC_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_4) != HAL_OK)
606       {
607         Error_Handler();
608       }
609       /* USER CODE BEGIN TIM3_Init 2 */
610
611       /* USER CODE END TIM3_Init 2 */
612       HAL_TIM_MspPostInit(&htim3);
613
614     }
615
616     /**
617       * @brief TIM4 Initialization Function
618       * @param None
```

```
619         * @retval None
620         */
621        static void MX_TIM4_Init(void)
622        {
623
624          /* USER CODE BEGIN TIM4_Init 0 */
625
626          /* USER CODE END TIM4_Init 0 */
627
628          TIM_ClockConfigTypeDef sClockSourceConfig = {0};
629          TIM_MasterConfigTypeDef sMasterConfig = {0};
630          TIM_OC_InitTypeDef sConfigOC = {0};
631
632          /* USER CODE BEGIN TIM4_Init 1 */
633
634          /* USER CODE END TIM4_Init 1 */
635          htim4.Instance = TIM4;
636          htim4.Init.Prescaler = 0;
637          htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
638          htim4.Init.Period = 65535;
639          htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
640          htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
641          if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
642          {
643            Error_Handler();
644          }
645          sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
646          if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
647          {
648            Error_Handler();
649          }
650          if (HAL_TIM_PWM_Init(&htim4) != HAL_OK)
651          {
652            Error_Handler();
653          }
654          sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
655          sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
656          if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
657          {
658            Error_Handler();
659          }
660          sConfigOC.OCMode = TIM_OCMODE_PWM1;
661          sConfigOC.Pulse = 0;
662          sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
663          sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
664          if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_4) != HAL_OK)
665          {
666            Error_Handler();
667          }
668          /* USER CODE BEGIN TIM4_Init 2 */
669
670          /* USER CODE END TIM4_Init 2 */
671          HAL_TIM_MspPostInit(&htim4);
672
673        }
674
675        /**
676          * @brief TIM5 Initialization Function
677          * @param None
678          * @retval None
679          */
680        static void MX_TIM5_Init(void)
681        {
682
683          /* USER CODE BEGIN TIM5_Init 0 */
684
685          /* USER CODE END TIM5_Init 0 */
686
```

```
687       TIM_ClockConfigTypeDef sClockSourceConfig = {0};
688       TIM_MasterConfigTypeDef sMasterConfig = {0};
689       TIM_IC_InitTypeDef sConfigIC = {0};
690
691       /* USER CODE BEGIN TIM5_Init 1 */
692
693       /* USER CODE END TIM5_Init 1 */
694       htim5.Instance = TIM5;
695       htim5.Init.Prescaler = 0;
696       htim5.Init.CounterMode = TIM_COUNTERMODE_UP;
697       htim5.Init.Period = 0xffffffff;
698       htim5.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
699       htim5.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
700       if (HAL_TIM_Base_Init(&htim5) != HAL_OK)
701       {
702         Error_Handler();
703       }
704       sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
705       if (HAL_TIM_ConfigClockSource(&htim5, &sClockSourceConfig) != HAL_OK)
706       {
707         Error_Handler();
708       }
709       if (HAL_TIM_IC_Init(&htim5) != HAL_OK)
710       {
711         Error_Handler();
712       }
713       sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
714       sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
715       if (HAL_TIMEx_MasterConfigSynchronization(&htim5, &sMasterConfig) != HAL_OK)
716       {
717         Error_Handler();
718       }
719       sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_BOTHEDGE;
720       sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
721       sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
722       sConfigIC.ICFilter = 0;
723       if (HAL_TIM_IC_ConfigChannel(&htim5, &sConfigIC, TIM_CHANNEL_4) != HAL_OK)
724       {
725         Error_Handler();
726       }
727       /* USER CODE BEGIN TIM5_Init 2 */
728
729       /* USER CODE END TIM5_Init 2 */
730
731     }
732
733     /**
734       * @brief GPIO Initialization Function
735       * @param None
736       * @retval None
737       */
738     static void MX_GPIO_Init(void)
739     {
740       GPIO_InitTypeDef GPIO_InitStruct = {0};
741
742       /* GPIO Ports Clock Enable */
743       __HAL_RCC_GPIOF_CLK_ENABLE();
744       __HAL_RCC_GPIOH_CLK_ENABLE();
745       __HAL_RCC_GPIOC_CLK_ENABLE();
746       __HAL_RCC_GPIOA_CLK_ENABLE();
747       __HAL_RCC_GPIOB_CLK_ENABLE();
748       __HAL_RCC_GPIOE_CLK_ENABLE();
749       __HAL_RCC_GPIOD_CLK_ENABLE();
750       __HAL_RCC_GPIOG_CLK_ENABLE();
751
752       /*Configure GPIO pin Output Level */
753       HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
754                         |GPIO_PIN_4|GPIO_PIN_5, GPIO_PIN_RESET);
```

```
  755
  756         /*Configure GPIO pin Output Level */
  757         HAL_GPIO_WritePin(GPIOC, GPIO_PIN_2, GPIO_PIN_RESET);
  758
  759         /*Configure GPIO pin Output Level */
  760         HAL_GPIO_WritePin(GPIOD, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
  761                           |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7, GPIO_PIN_RESET);
  762
  763         /*Configure GPIO pin Output Level */
  764         HAL_GPIO_WritePin(GPIOG, MoveBackward_Pin|MoveForward_Pin|Indicator_GreenLED_Pin|Indicator_IR_Pin,
GPIO_PIN_RESET);
  765
  766         /*Configure GPIO pins : PF0 PF1 PF2 PF3
  767                           PF4 PF5 */
  768         GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
  769                           |GPIO_PIN_4|GPIO_PIN_5;
  770         GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  771         GPIO_InitStruct.Pull = GPIO_NOPULL;
  772         GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  773         HAL_GPIO_Init(GPIOF, &GPIO_InitStruct);
  774
  775         /*Configure GPIO pin : PC2 */
  776         GPIO_InitStruct.Pin = GPIO_PIN_2;
  777         GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  778         GPIO_InitStruct.Pull = GPIO_NOPULL;
  779         GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  780         HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
  781
  782         /*Configure GPIO pins : Start_Input_Pin Stop_Input_Pin */
  783         GPIO_InitStruct.Pin = Start_Input_Pin|Stop_Input_Pin;
  784         GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  785         GPIO_InitStruct.Pull = GPIO_NOPULL;
  786         HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);
  787
  788         /*Configure GPIO pins : PD0 PD1 PD2 PD3
  789                           PD4 PD5 PD6 PD7 */
  790         GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
  791                           |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
  792         GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  793         GPIO_InitStruct.Pull = GPIO_NOPULL;
  794         GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  795         HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
  796
  797         /*Configure GPIO pins : BackwardSW_Pin ForwardSW_Pin */
  798         GPIO_InitStruct.Pin = BackwardSW_Pin|ForwardSW_Pin;
  799         GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  800         GPIO_InitStruct.Pull = GPIO_NOPULL;
  801         HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
  802
  803         /*Configure GPIO pins : MoveBackward_Pin MoveForward_Pin Indicator_GreenLED_Pin Indicator_IR_Pin */
  804         GPIO_InitStruct.Pin = MoveBackward_Pin|MoveForward_Pin|Indicator_GreenLED_Pin|Indicator_IR_Pin;
  805         GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  806         GPIO_InitStruct.Pull = GPIO_NOPULL;
  807         GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  808         HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
  809
  810         /* EXTI interrupt init*/
  811         HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
  812         HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);
  813
  814       }
  815
  816       /* USER CODE BEGIN 4 */
  817       static void displayDist(int num)
  818       {
  819         char digit_one = number(num % 10); // reminder = 9
  820         char digit_two = number((num / 10) % 10); // reminder = 9
  821         char digit_three = number((num / 100) % 10);//reminder =  9
```

```
822         char digit_four = number((num / 1000) % 10);//reminder =  1
823          resetDisplay();
824          HAL_Delay(50);
825
826       //    write (' ');
827       //    write (' ');
828       //    write (' ');
829       //    write (' ');
830       //    write (' ');
831       //    write (' ');
832           write ('D');
833           HAL_Delay (50);
834           write ('(');
835           HAL_Delay (50);
836           write ('i');
837           HAL_Delay (50);
838           write ('n');
839           HAL_Delay (50);
840           write (')');
841           HAL_Delay (50);
842           write ('=');
843           HAL_Delay (50);
844           write (' ');
845           write (' ');
846           write (' ');
847           write (' ');
848           write (' ');
849           write (' ');
850
851           write(digit_four);
852           HAL_Delay(50);
853           write(digit_three);
854           HAL_Delay(50);
855           write(digit_two);
856           HAL_Delay(50);
857           write(digit_one);
858           HAL_Delay(100);
859       }
860
861       static void displayFreq(int num)
862       {
863         char digit_one = number(num % 10); // reminder = 9
864         char digit_two = number((num / 10) % 10); // reminder = 9
865         char digit_three = number((num / 100) % 10);//reminder =  9
866         char digit_four = number((num / 1000) % 10);//reminder =  1
867          resetDisplay();
868          HAL_Delay(50);
869
870           write ('f');
871           HAL_Delay (50);
872           write ('(');
873           HAL_Delay (50);
874           write ('H');
875           HAL_Delay (50);
876           write ('z');
877           HAL_Delay (50);
878           write (')');
879           HAL_Delay (50);
880           write ('=');
881           HAL_Delay (50);
882
883           write(digit_four);
884           HAL_Delay(50);
885           write(digit_three);
886           HAL_Delay(50);
887           write(digit_two);
888           HAL_Delay(50);
889           write(digit_one);
```

```
890           HAL_Delay(100);
891         }
892         static void displayDC(int num)
893         {
894           char digit_one = number(num % 10); // reminder = 9
895           char digit_two = number((num / 10) % 10); // reminder = 9
896
897             write (' ');
898             write (' ');
899             write (' ');
900             write (' ');
901             write (' ');
902             write (' ');
903             HAL_Delay (50);
904             write ('D');
905             HAL_Delay (50);
906             write ('C');
907             HAL_Delay (50);
908             write ('(');
909             HAL_Delay (50);
910             write ('%');
911             HAL_Delay (50);
912             write (')');
913             HAL_Delay (50);
914             write ('=');
915
916             HAL_Delay(50);
917             write(digit_two);
918             HAL_Delay(50);
919             write(digit_one);
920             HAL_Delay(500);
921             resetDisplay();
922         }
923         char number (int n)
924         {
925           char R;
926           switch (n)
927           {
928           case 0:
929             R = '0';
930             break;
931           case 1:
932             R = '1';
933             break;
934           case 2:
935             R = '2';
936             break;
937           case 3:
938             R = '3';
939             break;
940           case 4:
941             R = '4';
942             break;
943           case 5:
944             R = '5';
945             break;
946           case 6:
947             R = '6';
948             break;
949           case 7:
950             R = '7';
951             break;
952           case 8:
953             R = '8';
954             break;
955           case 9:
956             R = '9';
957             break;
```

```
958          }
959          return R;
960        }
961        void resetDisplay()
962        {
963         command(0x01);       //clear dispaly
964         HAL_Delay (1);
965         command(0x02);       //return cursor home
966         HAL_Delay(1);      //
967        }
968        void graphicPosition(uint16_t horizontal, uint16_t vertical)
969        {
970          // Vertical Position with 0x80 appended for Set Graphic RAM Addr. Func.
971          vertical = vertical + 0x80;
972
973          command (vertical);
974          HAL_Delay (1);
975         // Horizontal Position with 0x80 appended for Set Graphic RAM Addr. Func.
976          horizontal = horizontal + 0x80;
977          temp3 = horizontal;
978          command (horizontal);
979          HAL_Delay (1);
980        }
981        void initGraphicMode()
982        {
983          command(0x36);// Enable extended function set
984          HAL_Delay(10);
985
986          command(0x41); // Set vertical scroll address
987          HAL_Delay(10);
988        }
989        void clearGFX()
990        {
991        //    int i,j;
992        //  for(j=0x80;j<0xa1;j++)
993        //  {
994        //    for(i=0x80;i<0x90;i++)
995        //    {
996        //      command(j);
997        //      command(i);
998        //      write(0x00);
999        //      write(0x00);
1000       //    }
1001       //  }
1002       //  HAL_Delay(1);
1003
1004         // Clear LCD
1005         for(uint16_t vert = 0; vert < 32; vert++)
1006         {
1007           for(uint16_t horiz = 0; horiz < 16; horiz++)
1008           {
1009             // Send graphics address
1010             graphicPosition(horiz, vert);
1011             // Send pixel fill data
1012             write(0x00); // First 8-bits
1013             write(0x00); // Second 8-bits
1014           }
1015         }
1016       }
1017       // Method to draw all needed characters in graphics mode
1018       void drawChar(unsigned int ref, unsigned int pos)
1019       {
1020         // Reference values for numbers 0, 1, 2, 3, 4, 5, 6
1021         // F, D, C, are 10, 11, 12, respectively
1022         // T, W, are 14, 15, respectively
1023         // Reference 13 will insert a blank space.
1024         switch(ref)
1025         {
```

```
1026        case 0:
1027          // Draw 0
1028          graphicPosition(pos,1);
1029          write(0x0E);
1030          graphicPosition(pos,2);
1031          write(0x11);
1032          graphicPosition(pos,3);
1033          write(0x13);
1034          graphicPosition(pos,4);
1035          write(0x15);
1036          graphicPosition(pos,5);
1037          write(0x19);
1038          graphicPosition(pos,6);
1039          write(0x11);
1040          graphicPosition(pos,7);
1041          write(0x0E);
1042          break;
1043        case 1:
1044          // Draw 1
1045          graphicPosition(pos,1);
1046          write(0x04);
1047          graphicPosition(pos,2);
1048          write(0x0C);
1049          graphicPosition(pos,3);
1050          write(0x04);
1051          graphicPosition(pos,4);
1052          write(0x04);
1053          graphicPosition(pos,5);
1054          write(0x04);
1055          graphicPosition(pos,6);
1056          write(0x04);
1057          graphicPosition(pos,7);
1058          write(0x0E);
1059          break;
1060        case 2:
1061          // Draw 2
1062          graphicPosition(pos,1);
1063          write(0x0E);
1064          graphicPosition(pos,2);
1065          write(0x11);
1066          graphicPosition(pos,3);
1067          write(0x01);
1068          graphicPosition(pos,4);
1069          write(0x02);
1070          graphicPosition(pos,5);
1071          write(0x04);
1072          graphicPosition(pos,6);
1073          write(0x08);
1074          graphicPosition(pos,7);
1075          write(0x1F);
1076          break;
1077        case 3:
1078          // Draw 3
1079          graphicPosition(pos,1);
1080          write(0x1F);
1081          graphicPosition(pos,2);
1082          write(0x02);
1083          graphicPosition(pos,3);
1084          write(0x04);
1085          graphicPosition(pos,4);
1086          write(0x02);
1087          graphicPosition(pos,5);
1088          write(0x01);
1089          graphicPosition(pos,6);
1090          write(0x11);
1091          graphicPosition(pos,7);
1092          write(0x0E);
1093          break;
```

```
1094        case 4:
1095          // Draw 4
1096          graphicPosition(pos,1);
1097          write(0x02);
1098          graphicPosition(pos,2);
1099          write(0x06);
1100          graphicPosition(pos,3);
1101          write(0x0A);
1102          graphicPosition(pos,4);
1103          write(0x12);
1104          graphicPosition(pos,5);
1105          write(0x1F);
1106          graphicPosition(pos,6);
1107          write(0x02);
1108          graphicPosition(pos,7);
1109          write(0x02);
1110          break;
1111        case 5:
1112          // Draw 5
1113          graphicPosition(pos,1);
1114          write(0x1F);
1115          graphicPosition(pos,2);
1116          write(0x10);
1117          graphicPosition(pos,3);
1118          write(0x1E);
1119          graphicPosition(pos,4);
1120          write(0x01);
1121          graphicPosition(pos,5);
1122          write(0x01);
1123          graphicPosition(pos,6);
1124          write(0x11);
1125          graphicPosition(pos,7);
1126          write(0x0E);
1127          break;
1128        case 6:
1129          // Draw 6
1130          graphicPosition(pos,1);
1131          write(0x06);
1132          graphicPosition(pos,2);
1133          write(0x08);
1134          graphicPosition(pos,3);
1135          write(0x10);
1136          graphicPosition(pos,4);
1137          write(0x1E);
1138          graphicPosition(pos,5);
1139          write(0x11);
1140          graphicPosition(pos,6);
1141          write(0x11);
1142          graphicPosition(pos,7);
1143          write(0x0E);
1144          break;
1145        case 7:
1146          // Draw 7
1147          graphicPosition(pos,1);
1148          write(0x1F);
1149          graphicPosition(pos,2);
1150          write(0x01);
1151          graphicPosition(pos,3);
1152          write(0x02);
1153          graphicPosition(pos,4);
1154          write(0x04);
1155          graphicPosition(pos,5);
1156          write(0x08);
1157          graphicPosition(pos,6);
1158          write(0x08);
1159          graphicPosition(pos,7);
1160          write(0x08);
1161          break;
```

```
1162        case 8:
1163          // Draw 8
1164          graphicPosition(pos,1);
1165          write(0x0E);
1166          graphicPosition(pos,2);
1167          write(0x11);
1168          graphicPosition(pos,3);
1169          write(0x11);
1170          graphicPosition(pos,4);
1171          write(0x0E);
1172          graphicPosition(pos,5);
1173          write(0x11);
1174          graphicPosition(pos,6);
1175          write(0x11);
1176          graphicPosition(pos,7);
1177          write(0x0E);
1178          break;
1179        case 9:
1180          // Draw 9
1181          graphicPosition(pos,1);
1182          write(0x0E);
1183          graphicPosition(pos,2);
1184          write(0x11);
1185          graphicPosition(pos,3);
1186          write(0x11);
1187          graphicPosition(pos,4);
1188          write(0x0F);
1189          graphicPosition(pos,5);
1190          write(0x01);
1191          graphicPosition(pos,6);
1192          write(0x02);
1193          graphicPosition(pos,7);
1194          write(0x0C);
1195          break;
1196        case 10:
1197          // Draw 'F'
1198          graphicPosition(pos,1);
1199          write(0x1F);
1200          graphicPosition(pos,2);
1201          write(0x10);
1202          graphicPosition(pos,3);
1203          write(0x10);
1204          graphicPosition(pos,4);
1205          write(0x1E);
1206          graphicPosition(pos,5);
1207          write(0x10);
1208          graphicPosition(pos,6);
1209          write(0x10);
1210          graphicPosition(pos,7);
1211          write(0x10);
1212          break;
1213        case 11:
1214          // Draw 'D'
1215          graphicPosition(pos,16);
1216          write(0x1C);
1217          graphicPosition(pos,17);
1218          write(0x12);
1219          graphicPosition(pos,18);
1220          write(0x11);
1221          graphicPosition(pos,19);
1222          write(0x11);
1223          graphicPosition(pos,20);
1224          write(0x11);
1225          graphicPosition(pos,21);
1226          write(0x12);
1227          graphicPosition(pos,22);
1228          write(0x1C);
1229          break;
```

```
1230        case 12:
1231          // Draw 'C'
1232          graphicPosition(pos,16);
1233          write(0x0E);
1234          graphicPosition(pos,17);
1235          write(0x11);
1236          graphicPosition(pos,18);
1237          write(0x10);
1238          graphicPosition(pos,19);
1239          write(0x10);
1240          graphicPosition(pos,20);
1241          write(0x10);
1242          graphicPosition(pos,21);
1243          write(0x11);
1244          graphicPosition(pos,22);
1245          write(0x0E);
1246          break;
1247        case 13:
1248          // Draw blank space
1249          graphicPosition(pos,1);
1250          write(0x00);
1251          graphicPosition(pos,2);
1252          write(0x00);
1253          graphicPosition(pos,3);
1254          write(0x00);
1255          graphicPosition(pos,4);
1256          write(0x00);
1257          graphicPosition(pos,5);
1258          write(0x00);
1259          graphicPosition(pos,6);
1260          write(0x00);
1261          graphicPosition(pos,7);
1262          write(0x00);
1263          break;
1264
1265        case 14:
1266          // Draw 'T'
1267          graphicPosition(pos,1);
1268          write(0x1F);
1269          graphicPosition(pos,2);
1270          write(0x04);
1271          graphicPosition(pos,3);
1272          write(0x04);
1273          graphicPosition(pos,4);
1274          write(0x04);
1275          graphicPosition(pos,5);
1276          write(0x04);
1277          graphicPosition(pos,6);
1278          write(0x04);
1279          graphicPosition(pos,7);
1280          write(0x04);
1281          break;
1282        case 15:
1283          // Draw 'W'
1284          graphicPosition(pos,1);
1285          write(0x11);
1286          graphicPosition(pos,2);
1287          write(0x11);
1288          graphicPosition(pos,3);
1289          write(0x11);
1290          graphicPosition(pos,4);
1291          write(0x11);
1292          graphicPosition(pos,5);
1293          write(0x15);
1294          graphicPosition(pos,6);
1295          write(0x15);
1296          graphicPosition(pos,7);
1297          write(0x0E);
```

```
1298          break;
1299        }
1300      }
1301
1302      // Method for printing DC in proper location, used by drawDuty
1303      void drawDC(unsigned int ref, unsigned int pos)
1304      {
1305        switch(ref)
1306        {
1307        case 0:
1308          // Draw 0
1309          graphicPosition(pos,16);
1310          write(0x0E);
1311          graphicPosition(pos,17);
1312          write(0x11);
1313          graphicPosition(pos,18);
1314          write(0x13);
1315          graphicPosition(pos,19);
1316          write(0x15);
1317          graphicPosition(pos,20);
1318          write(0x19);
1319          graphicPosition(pos,21);
1320          write(0x11);
1321          graphicPosition(pos,22);
1322          write(0x0E);
1323          break;
1324        case 1:
1325          // Draw 1
1326          graphicPosition(pos,16);
1327          write(0x04);
1328          graphicPosition(pos,17);
1329          write(0x0C);
1330          graphicPosition(pos,18);
1331          write(0x04);
1332          graphicPosition(pos,19);
1333          write(0x04);
1334          graphicPosition(pos,20);
1335          write(0x04);
1336          graphicPosition(pos,21);
1337          write(0x04);
1338          graphicPosition(pos,22);
1339          write(0x0E);
1340          break;
1341        case 2:
1342          // Draw 2
1343          graphicPosition(pos,16);
1344          write(0x0E);
1345          graphicPosition(pos,17);
1346          write(0x11);
1347          graphicPosition(pos,18);
1348          write(0x01);
1349          graphicPosition(pos,19);
1350          write(0x02);
1351          graphicPosition(pos,20);
1352          write(0x04);
1353          graphicPosition(pos,21);
1354          write(0x08);
1355          graphicPosition(pos,22);
1356          write(0x1F);
1357          break;
1358        case 3:
1359          // Draw 3
1360          graphicPosition(pos,16);
1361          write(0x1F);
1362          graphicPosition(pos,17);
1363          write(0x02);
1364          graphicPosition(pos,18);
1365          write(0x04);
```

```
1366          graphicPosition(pos,19);
1367          write(0x02);
1368          graphicPosition(pos,20);
1369          write(0x01);
1370          graphicPosition(pos,21);
1371          write(0x11);
1372          graphicPosition(pos,22);
1373          write(0x0E);
1374          break;
1375        case 4:
1376          // Draw 4
1377          graphicPosition(pos,16);
1378          write(0x02);
1379          graphicPosition(pos,17);
1380          write(0x06);
1381          graphicPosition(pos,18);
1382          write(0x0A);
1383          graphicPosition(pos,19);
1384          write(0x12);
1385          graphicPosition(pos,20);
1386          write(0x1F);
1387          graphicPosition(pos,21);
1388          write(0x02);
1389          graphicPosition(pos,22);
1390          write(0x02);
1391          break;
1392        case 5:
1393          // Draw 5
1394          graphicPosition(pos,16);
1395          write(0x1F);
1396          graphicPosition(pos,17);
1397          write(0x10);
1398          graphicPosition(pos,18);
1399          write(0x1E);
1400          graphicPosition(pos,19);
1401          write(0x01);
1402          graphicPosition(pos,20);
1403          write(0x01);
1404          graphicPosition(pos,21);
1405          write(0x11);
1406          graphicPosition(pos,22);
1407          write(0x0E);
1408          break;
1409        case 6:
1410          // Draw 6
1411          graphicPosition(pos,16);
1412          write(0x06);
1413          graphicPosition(pos,17);
1414          write(0x08);
1415          graphicPosition(pos,18);
1416          write(0x10);
1417          graphicPosition(pos,19);
1418          write(0x1E);
1419          graphicPosition(pos,20);
1420          write(0x11);
1421          graphicPosition(pos,21);
1422          write(0x11);
1423          graphicPosition(pos,22);
1424          write(0x0E);
1425          break;
1426        case 7:
1427          // Draw 7
1428          graphicPosition(pos,16);
1429          write(0x1F);
1430          graphicPosition(pos,17);
1431          write(0x01);
1432          graphicPosition(pos,18);
1433          write(0x02);
```

```
1434            graphicPosition(pos,19);
1435            write(0x04);
1436            graphicPosition(pos,20);
1437            write(0x08);
1438            graphicPosition(pos,21);
1439            write(0x08);
1440            graphicPosition(pos,22);
1441            write(0x08);
1442            break;
1443          case 8:
1444            // Draw 8
1445            graphicPosition(pos,16);
1446            write(0x0E);
1447            graphicPosition(pos,17);
1448            write(0x11);
1449            graphicPosition(pos,18);
1450            write(0x11);
1451            graphicPosition(pos,19);
1452            write(0x0E);
1453            graphicPosition(pos,20);
1454            write(0x11);
1455            graphicPosition(pos,21);
1456            write(0x11);
1457            graphicPosition(pos,22);
1458            write(0x0E);
1459            break;
1460          case 9:
1461            // Draw 9
1462            graphicPosition(pos,16);
1463            write(0x0E);
1464            graphicPosition(pos,17);
1465            write(0x11);
1466            graphicPosition(pos,18);
1467            write(0x11);
1468            graphicPosition(pos,19);
1469            write(0x0F);
1470            graphicPosition(pos,20);
1471            write(0x01);
1472            graphicPosition(pos,21);
1473            write(0x02);
1474            graphicPosition(pos,22);
1475            write(0x0C);
1476            break;
1477          }
1478        }
1479
1480        // Method for drawing frequency in graphics mode
1481        void drawFreq(uint32_t freq)
1482        {
1483          // Draw 'F' = ref 7, in position 0
1484        //  drawChar(10, 0);
1485          drawChar(10,8);
1486          // Split freq into six digits
1487          fd6 = freq % 10;
1488          freq = freq / 10;
1489          fd5 = freq % 10;
1490          freq = freq / 10;
1491          fd4 = freq % 10;
1492          freq = freq / 10;
1493          fd3 = freq % 10;
1494          freq = freq / 10;
1495          fd2 = freq % 10;
1496          freq = freq / 10;
1497          fd1 = freq % 10;
1498
1499          // Assign freq digits to array
1500          digits[0] = fd1;
1501          digits[1] = fd2;
```

```
1502        digits[2] = fd3;
1503        digits[3] = fd4;
1504        digits[4] = fd5;
1505        digits[5] = fd6;
1506
1507        // Print digits
1508        for(int i = 0; i < 6; i++)
1509        {
1510          switch(digits[i])
1511          {
1512          case 9:
1513            // Draw 9, add one to position because F is at 0
1514            drawChar(9, i+9);
1515            break;
1516          case 8:
1517            // Draw 8
1518            drawChar(8, i+9);
1519            break;
1520          case 7:
1521            // Draw 7
1522            drawChar(7, i+9);
1523            break;
1524          case 6:
1525            // Draw 6
1526            drawChar(6, i+9);
1527            break;
1528          case 5:
1529            // Draw 5
1530            drawChar(5, i+9);
1531            break;
1532          case 4:
1533            // Draw 4
1534            drawChar(4, i+9);
1535            break;
1536          case 3:
1537            // Draw 3
1538            drawChar(3, i+9);
1539            break;
1540          case 2:
1541            // Draw 2
1542            drawChar(2, i+9);
1543            break;
1544          case 1:
1545            // Draw 1
1546            drawChar(1, i+9);
1547            break;
1548          case 0:
1549            // Draw 0
1550            drawChar(0, i+9);
1551            break;
1552
1553          }
1554        }
1555
1556      }
1557      // Method to draw Period and Pulse Width
1558      void drawPWT(unsigned int period, unsigned int pulse)
1559      {
1560        // Draw Period label 'T'
1561        drawChar(14, 0);
1562      //  drawChar(14,8);
1563        // Draw Period Time value as 3 digits in microseconds
1564        // Split period into 3 digits
1565        t1 = period % 10;
1566        period = period /10;
1567        t2 = period % 10;
1568        period = period / 10;
1569        t3 = period % 10;
```

```
1570        T[0] = t3;
1571        T[1] = t2;
1572        T[2] = t1;
1573        for(int i = 0; i < 3; i++)
1574        {
1575          switch(T[i])
1576          {
1577          case 9:
1578            // Draw 6
1579            drawChar(9, i+1);
1580            break;
1581          case 8:
1582            // Draw 6
1583            drawChar(8, i+1);
1584            break;
1585          case 7:
1586            // Draw 6
1587            drawChar(7, i+1);
1588            break;
1589          case 6:
1590            // Draw 6
1591            drawChar(6, i+1);
1592            break;
1593          case 5:
1594            // Draw 5
1595            drawChar(5, i+1);
1596            break;
1597          case 4:
1598            // Draw 4
1599            drawChar(4, i+1);
1600            break;
1601          case 3:
1602            // Draw 3
1603            drawChar(3, i+1);
1604            break;
1605          case 2:
1606            // Draw 2
1607            drawChar(2, i+1);
1608            break;
1609          case 1:
1610            // Draw 1
1611            drawChar(1, i+1);
1612            break;
1613          case 0:
1614            // Draw 0
1615            drawChar(0, i+1);
1616            break;
1617
1618          }
1619        }
1620        // Draw Pulse Width label 'W'
1621        drawChar(15,4);
1622        // Draw Pulse Width value as 3 digits in microseonds
1623        p1 = pulse % 10;
1624        pulse = pulse / 10;
1625        p2 = pulse % 10;
1626        pulse = pulse / 10;
1627        p3 = pulse % 10;
1628        P[0] = p3;
1629        P[1] = p2;
1630        P[2] = p1;
1631        for(int i = 0; i < 3; i++)
1632        {
1633          switch(P[i])
1634          {
1635          case 9:
1636            // Draw 6
1637            drawChar(9, i+5);
```

```
1638            break;
1639          case 8:
1640            // Draw 6
1641            drawChar(8, i+5);
1642            break;
1643          case 7:
1644            // Draw 6
1645            drawChar(7, i+5);
1646            break;
1647          case 6:
1648            // Draw 6
1649            drawChar(6, i+5);
1650            break;
1651          case 5:
1652            // Draw 5
1653            drawChar(5, i+5);
1654            break;
1655          case 4:
1656            // Draw 4
1657            drawChar(4, i+5);
1658            break;
1659          case 3:
1660            // Draw 3
1661            drawChar(3, i+5);
1662            break;
1663          case 2:
1664            // Draw 2
1665            drawChar(2, i+5);
1666            break;
1667          case 1:
1668            // Draw 1
1669            drawChar(1, i+5);
1670            break;
1671          case 0:
1672            // Draw 0
1673            drawChar(0, i+5);
1674            break;
1675
1676          }
1677        }
1678      }
1679      // Method for drawing duty cycle in graphics mode
1680      void drawDuty(int DC)
1681      {
1682
1683        // Draw 'D' and 'C' label
1684        drawChar(11,10);
1685        drawChar(12,11);
1686
1687        // Split duty cycle into two digits
1688        dcd1 = (((int) dutyCycle) % 10);
1689        dcd2 = (((int) dutyCycle) / 10) % 10;
1690
1691        // Assign duty cycle digits to array
1692        dcds[0] = dcd2;
1693        dcds[1] = dcd1;
1694
1695        // Print Duty Cycle Value
1696        for(int i = 0; i < 2; i++)
1697        {
1698          switch(dcds[i])
1699          {
1700          case 9:
1701            // Draw 6
1702            drawDC(9, i+12);
1703            break;
1704          case 8:
1705            // Draw 6
```

```
1706            drawDC(8, i+12);
1707            break;
1708          case 7:
1709            // Draw 6
1710            drawDC(7, i+12);
1711            break;
1712          case 6:
1713            // Draw 6
1714            drawDC(6, i+12);
1715            break;
1716          case 5:
1717            // Draw 5
1718            drawDC(5, i+12);
1719            break;
1720          case 4:
1721            // Draw 4
1722            drawDC(4, i+12);
1723            break;
1724          case 3:
1725            // Draw 3
1726            drawDC(3, i+12);
1727            break;
1728          case 2:
1729            // Draw 2
1730            drawDC(2, i+12);
1731            break;
1732          case 1:
1733            // Draw 1
1734            drawDC(1, i+12);
1735            break;
1736          case 0:
1737            // Draw 0
1738            drawDC(0, i+12);
1739            break;
1740
1741          }
1742        }
1743      }
1744      void drawWave(uint32_t duty)
1745      {
1746        // If the previous duty cycle is the same as the current, exit method
1747        if(duty == prevDuty)
1748        {
1749          prevDuty = duty;
1750                return;
1751        }
1752
1753        // New Duty Cycle input, clear wave
1754        // For loop to clear the waveform area of the LCD
1755        for(uint16_t vert = 11; vert < 27; vert++){
1756          for(uint16_t horiz = 0; horiz < 9; horiz++){
1757                // Send graphics address
1758                graphicPosition(horiz, vert);
1759                // Send pixel fill data
1760            write(0x00); // First 8-bits
1761                write(0x00); // Second 8-bits
1762          }
1763        }
1764
1765        // Duty Cycle split into 16 bit blocks
1766        // Duty 60 = 3.75 blocks = 3 blocks
1767        blocks = duty / 16;
1768        graphicPosition(0, 26); //25
1769        for(int i = 0; i < 2; i++)
1770        {
1771          write(0xFF);
1772        }
1773        //draw the rising edge
```

```
1774        for(int i = 0; i < 15; i++)
1775        {
1776          // 16 pixel vertical line starting from 12
1777          // Start of waveform at 1, add offset to center block
1778          graphicPosition(1, 26-i);
1779          drawPixel(0);
1780        }
1781        //MSD
1782        // Remainder is 75 - (4*16) = 75 - 64 = 11
1783        // 3 full blocks will be filled, then 12 pixels in the last block
1784        remainder = duty - (blocks*16);
1785
1786        // Set graphics position to start of waveform
1787        graphicPosition(1,11);
1788
1789        // For loop to fill all blocks in the top line
1790        for(int i = 0; i < blocks; i++)
1791        {
1792          write(0xFF);
1793          write(0xFF);
1794        }
1795
1796        // Draws the remaining pixels in the block for the top line
1797        drawRem(remainder);
1798
1799
1800        // Draw Middle Line
1801        // Middle line is at the duty cycle value = 75
1802        // Line will be in block 3
1803        for(int i = 0; i < 15; i++)
1804        {
1805          // 15 pixel vertical line starting from 12
1806          // Start of waveform at 1, add offset to center block
1807          graphicPosition( (1+blocks), 12+i);
1808          drawPixel(remainder);
1809        }
1810
1811        // Draw bottom line
1812        // Fill rest of current block based on inverse of remainder
1813        // Bottom line at vertical 26
1814        graphicPosition( (1+blocks), 26);
1815        drawLowRem((17-remainder)); //17-11=5 pixels
1816
1817        // Fill rest of bottom line, total blocks = 7
1818        // 4 blocks filled so far
1819        // Fill blocks 5, 6, and first 4 bits of 7
1820        // 3 blocks remaining to fill for DC = 75
1821        // Set graphics position to proper block
1822        graphicPosition( ((1+blocks)+1), 26);
1823        // For loop to fill blocks through block 6
1824        for(int i = 0; i < 6 - (blocks + 1); i++)
1825        {
1826          write(0xFF);
1827          write(0xFF);
1828        }
1829
1830        // Last block (7th) only needs 4 bits
1831        // This instruction is always executed because duty cycle only changes from 20-80%
1832        // Change this if duty cycle values exceed 96%
1833        // First 6 blocks = 96 pixels
1834        write(0xF0);
1835
1836          for(int i = 0; i < 14; i++)
1837        {
1838          temp2 = blocks;
1839          // 15 pixel vertical line starting from 12
1840          // Start of waveform at 1, add offset to center block
1841          graphicPosition(7, 25-i);
```

```
1842          //drawPixel(4);
1843                write(0x10);
1844          write(0x00);
1845         }
1846         graphicPosition(7, 11);
1847         //drawRem(13);
1848         //drawLowRem(13);
1849         write(0x1F);
1850         write(0xFF);
1851
1852        }
1853        // Method for drawing remainder of top line of waveform
1854        void drawRem(unsigned int rem)
1855        {
1856         switch (rem)
1857         {
1858           case 0:
1859                // Print nothing
1860                write(0x00);
1861                write(0x00);
1862                break;
1863           case 1:
1864                // Print 1 pixel
1865                write(0x80);
1866                write(0x00);
1867                break;
1868           case 2:
1869                // Print 2 pixels
1870                write(0xC0);
1871                write(0x00);
1872                break;
1873           case 3:
1874                // Print 3 pixels
1875                write(0xE0);
1876                write(0x00);
1877                break;
1878           case 4:
1879                // Print 4 pixels
1880                write(0xF0);
1881                write(0x00);
1882                break;
1883           case 5:
1884                // Send 5 pixels
1885                write(0xF8);
1886                write(0x00);
1887                break;
1888           case 6:
1889                write(0xFC);
1890                write(0x00);
1891                break;
1892           case 7:
1893                write(0xFE);
1894                write(0x00);
1895                break;
1896           case 8:
1897                write(0xFF);
1898                write(0x00);
1899                break;
1900           case 9:
1901                write(0xFF);
1902                write(0x80);
1903                break;
1904           case 10:
1905                write(0xFF);
1906                write(0xC0);
1907                break;
1908           case 11:
1909                write(0xFF);
```

```
1910                write(0xE0);
1911                break;
1912            case 12:
1913                write(0xFF);
1914                write(0xF0);
1915                break;
1916            case 13:
1917                write(0xFF);
1918                write(0xF8);
1919                break;
1920            case 14:
1921                write(0xFF);
1922                write(0xFC);
1923                break;
1924            case 15:
1925                write(0xFF);
1926                write(0xFE);
1927                break;
1928          }
1929        }
1930        // Method for drawing vertical line of waveform
1931        void drawPixel(unsigned int rem)
1932        {
1933          switch (rem)
1934            {
1935              case 0:
1936                  // If there is no remainder, print line on 16th bit of block
1937                  write(0x80);
1938                  write(0x00);
1939                  break;
1940              case 1:
1941                  // Print 1 pixel
1942                  write(0x80);
1943                  write(0x00);
1944                  break;
1945              case 2:
1946                  // Print 2 pixels
1947                  write(0x40);
1948                  write(0x00);
1949                  break;
1950              case 3:
1951                  // Print 3 pixels
1952                  write(0x20);
1953                  write(0x00);
1954                  break;
1955              case 4:
1956                  // Print 4 pixels
1957                  write(0x10);
1958                  write(0x00);
1959                  break;
1960              case 5:
1961                  // Send 5 pixels
1962                  write(0x08);
1963                  write(0x00);
1964                  break;
1965              case 6:
1966                  write(0x04);
1967                  write(0x00);
1968                  break;
1969              case 7:
1970                  write(0x02);
1971                  write(0x00);
1972                  break;
1973              case 8:
1974                  write(0x01);
1975                  write(0x00);
1976                  break;
1977              case 9:
```

```
1978                write(0x00);
1979                write(0x80);
1980                break;
1981            case 10:
1982                write(0x00);
1983                write(0x40);
1984                break;
1985            case 11:
1986                write(0x00);
1987                write(0x20);
1988                break;
1989            case 12:
1990                write(0x00);
1991                write(0x10);
1992                break;
1993            case 13:
1994                write(0x00);
1995                write(0x08);
1996                break;
1997            case 14:
1998                write(0x00);
1999                write(0x04);
2000                break;
2001            case 15:
2002                write(0x00);
2003                write(0x02);
2004                break;
2005          }
2006        }
2007        // Method for drawing remainder of bottom line of waveform after vertical line
2008        void drawLowRem(unsigned int rem)
2009        {
2010          switch (rem)
2011          {
2012            case 0:
2013                // Print nothing
2014                write(0xFF);
2015                write(0xFF);
2016                break;
2017            case 1:
2018                // Print 1 pixel
2019                write(0x00);
2020                write(0x01);
2021                break;
2022            case 2:
2023                // Print 2 pixels
2024                write(0x00);
2025                write(0x03);
2026                break;
2027            case 3:
2028                // Print 3 pixels
2029                write(0x00);
2030                write(0x07);
2031                break;
2032            case 4:
2033                // Print 4 pixels
2034                write(0x00);
2035                write(0x0F);
2036                break;
2037            case 5:
2038                // Send 5 pixels
2039                write(0x00);
2040                write(0x1F);
2041                break;
2042            case 6:
2043                write(0x00);
2044                write(0x3F);
2045                break;
```

```
2046            case 7:
2047                  write(0x00);
2048                  write(0x7F);
2049                  break;
2050            case 8:
2051                  write(0x00);
2052                  write(0xFF);
2053                  break;
2054            case 9:
2055                  write(0x01);
2056                  write(0xFF);
2057                  break;
2058            case 10:
2059                  write(0x03);
2060                  write(0xFF);
2061                  break;
2062            case 11:
2063                  write(0x07);
2064                  write(0xFF);
2065                  break;
2066            case 12:
2067                  write(0x0F);
2068                  write(0xFF);
2069                  break;
2070            case 13:
2071                  write(0x1F);
2072                  write(0xFF);
2073                  break;
2074            case 14:
2075                  write(0x3F);
2076                  write(0xFF);
2077                  break;
2078            case 15:
2079                  write(0x7F);
2080                  write(0xFF);
2081                  break;
2082            case 16:
2083                write(0xFF);
2084                write(0xFF);
2085                break;
2086          }
2087        }
2088
2089        void command(char i)
2090        {
2091          GPIOD->ODR = i; //put data on output Port
2092          HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0, GPIO_PIN_RESET); //PF0 = RS = LOW : send instruction, AO
2093          HAL_GPIO_WritePin(GPIOF, GPIO_PIN_1, GPIO_PIN_RESET); // /PF1= R/W, WR in 8080 mode; R/W in 6800 mode,
WRT
2094          HAL_GPIO_WritePin(GPIOF, GPIO_PIN_2, GPIO_PIN_SET); //PF2 = E, active low, CS activated, CS1
2095          HAL_Delay (1);
2096          HAL_GPIO_WritePin(GPIOF, GPIO_PIN_2, GPIO_PIN_RESET); //PF2 = E, active low, CS activated, CS1
2097
2098        }
2099        void write (char i)
2100        {
2101          GPIOD->ODR = i; //put data on output Port
2102          HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0, GPIO_PIN_SET); //PF0 = RS = LOW : send instruction, AO
2103          HAL_GPIO_WritePin(GPIOF, GPIO_PIN_1, GPIO_PIN_RESET); // /PF1= R/W, WR in 8080 mode; R/W in 6800 mode,
WRT
2104          HAL_GPIO_WritePin(GPIOF, GPIO_PIN_2, GPIO_PIN_SET); //PF2 = E, active low, CS activated, CS1
2105          HAL_Delay (1);
2106          HAL_GPIO_WritePin(GPIOF, GPIO_PIN_2, GPIO_PIN_RESET); //PF2 = E, active low, CS activated, CS1
2107          HAL_Delay(1);
2108
2109        }
2110        void init ()
2111        {
```

```
2112          HAL_Delay (100);
2113          HAL_GPIO_WritePin(GPIOF, GPIO_PIN_3, GPIO_PIN_SET); // ??, PF3 = PSB, ??
2114        //  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_2, GPIO_PIN_RESET); //PF2 = E, active low, CS activated, CS1
2115          //HAL_Delay (100); //Wait >15 msec after power is applied
2116          HAL_GPIO_WritePin(GPIOF, GPIO_PIN_4, GPIO_PIN_RESET); //PF2 = E, active low, CS activated, CS1
2117          HAL_Delay (150); //Wait >15 msec after power is applied
2118          HAL_GPIO_WritePin(GPIOF, GPIO_PIN_4, GPIO_PIN_SET); //PF2 = E, active low, CS activated, CS1
2119          command(0x30);              //command 0x30 = Wake up
2120          HAL_Delay (1);              //Wait time >100uS
2121          command(0x30);              //command 0x30 = Wake up #2
2122          //command(0x34);              //Function set: 8-bit/RE=1: extended instruction
2123          HAL_Delay (1);  //must wait 160us, busy flag not available
2124          command(0x0C);
2125          HAL_Delay (1);
2126          command(0x01);              //Clear dispaly
2127          HAL_Delay (15);
2128          command(0x06);              //Entry mode set
2129          HAL_Delay (1);
2130
2131
2132        }
2133
2134        /* USER CODE END 4 */
2135
2136        /**
2137          * @brief  Period elapsed callback in non blocking mode
2138          * @note   This function is called  when TIM7 interrupt took place, inside
2139          * HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to increment
2140          * a global variable "uwTick" used as application time base.
2141          * @param  htim : TIM handle
2142          * @retval None
2143          */
2144        void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
2145        {
2146          /* USER CODE BEGIN Callback 0 */
2147
2148          /* USER CODE END Callback 0 */
2149          if (htim->Instance == TIM7) {
2150            HAL_IncTick();
2151          }
2152          /* USER CODE BEGIN Callback 1 */
2153
2154          /* USER CODE END Callback 1 */
2155        }
2156
2157        /**
2158          * @brief  This function is executed in case of error occurrence.
2159          * @retval None
2160          */
2161        void Error_Handler(void)
2162        {
2163          /* USER CODE BEGIN Error_Handler_Debug */
2164          /* User can add his own implementation to report the HAL error return state */
2165
2166          /* USER CODE END Error_Handler_Debug */
2167        }
2168
2169        #ifdef  USE_FULL_ASSERT
2170        /**
2171          * @brief  Reports the name of the source file and the source line number
2172          *         where the assert_param error has occurred.
2173          * @param  file: pointer to the source file name
2174          * @param  line: assert_param error line source number
2175          * @retval None
2176          */
2177        void assert_failed(uint8_t *file, uint32_t line)
2178        {
2179          /* USER CODE BEGIN 6 */
```

```
2180           /* User can add his own implementation to report the file name and line number,
2181              tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
2182           /* USER CODE END 6 */
2183         }
2184         #endif /* USE_FULL_ASSERT */
2185

Maximum stack usage in bytes:

.cstack Function
------- --------
   0   Error_Handler
   0   HAL_TIM_PeriodElapsedCallback
     0   -> HAL_IncTick
  80   SystemClock_Config
    80   -> HAL_RCC_ClockConfig
    80   -> HAL_RCC_OscConfig
    80   -> memset
  16   clearGFX
    16   -> graphicPosition
    16   -> write
   8   command
     8   -> HAL_Delay
     0   -> HAL_GPIO_WritePin
     8   -> HAL_GPIO_WritePin
   8   drawChar
     8   -> graphicPosition
     0   -> write
     8   -> write
   8   drawDC
     8   -> graphicPosition
     0   -> write
     8   -> write
  16   drawDuty
    16   -> drawDC
    16   -> graphicPosition
    16   -> write
  24   drawFreq
    24   -> drawChar
   8   drawLowRem
     0   -> write
     8   -> write
  24   drawPWT
    24   -> drawChar
   8   drawPixel
     0   -> write
     8   -> write
   8   drawRem
     0   -> write
     8   -> write
  24   drawWave
    24   -> drawLowRem
    24   -> drawPixel
    24   -> drawRem
    24   -> graphicPosition
     0   -> write
    24   -> write
   8   graphicPosition
     0   -> HAL_Delay
     8   -> HAL_Delay
     8   -> command
   8   init
     0   -> HAL_Delay
     8   -> HAL_Delay
     8   -> HAL_GPIO_WritePin
     8   -> command
   8   initGraphicMode
     0   -> HAL_Delay
```

```
      8   -> HAL_Delay
      8   -> command
     88   main
      88   -> HAL_Delay
      88   -> HAL_GPIO_Init
      88   -> HAL_GPIO_WritePin
      88   -> HAL_Init
      88   -> HAL_NVIC_EnableIRQ
      88   -> HAL_NVIC_SetPriority
      88   -> HAL_TIMEx_MasterConfigSynchronization
      88   -> HAL_TIM_Base_Init
      88   -> HAL_TIM_ConfigClockSource
      88   -> HAL_TIM_IC_ConfigChannel
      88   -> HAL_TIM_IC_Init
      88   -> HAL_TIM_IC_Start_IT
      88   -> HAL_TIM_MspPostInit
      88   -> HAL_TIM_OC_ConfigChannel
      88   -> HAL_TIM_OC_Init
      88   -> HAL_TIM_OC_Start_IT
      88   -> HAL_TIM_PWM_ConfigChannel
      88   -> HAL_TIM_PWM_Init
      88   -> SystemClock_Config
      88   -> clearGFX
      88   -> command
      88   -> init
      88   -> initGraphicMode
      88   -> memset
      88   -> number
      88   -> resetDisplay
      88   -> write
     0   number
     8   resetDisplay
      0   -> HAL_Delay
      8   -> HAL_Delay
      8   -> command
     8   write
      0   -> HAL_Delay
      8   -> HAL_Delay
      8   -> HAL_GPIO_WritePin


Section sizes:

Bytes  Function/Label
-----  --------------
    4  ??DataTable0
    4  ??DataTable0_1
    4  ??DataTable2
    4  ??DataTable2_1
    4  ??DataTable2_2
    4  ??DataTable2_3
    4  ??DataTable2_4
    4  ??DataTable2_5
    4  ??DataTable2_6
    4  ??DataTable2_7
    4  ??DataTable2_8
    4  ??DataTable2_9
    4  ??DataTable3
    4  ??DataTable6
    4  ??DataTable6_1
    4  ??DataTable9
    4  ??DataTable9_1
    4  ??DataTable9_2
    4  ??DataTable9_3
    4  ??DataTable9_4
    4  ??DataTable9_5
    4  ??DataTable9_6
    4  ??DataTable9_7
```

```
 10  ?Subroutine0
  6  ?Subroutine1
  8  ?Subroutine10
 10  ?Subroutine11
 12  ?Subroutine12
 12  ?Subroutine13
 12  ?Subroutine14
 10  ?Subroutine15
 16  ?Subroutine16
 10  ?Subroutine17
  6  ?Subroutine18
 10  ?Subroutine19
  6  ?Subroutine2
  8  ?Subroutine20
 10  ?Subroutine21
  6  ?Subroutine22
  6  ?Subroutine23
 18  ?Subroutine24
  6  ?Subroutine25
  6  ?Subroutine26
  6  ?Subroutine27
  6  ?Subroutine28
  4  ?Subroutine29
  4  ?Subroutine3
 10  ?Subroutine30
  4  ?Subroutine31
  4  ?Subroutine32
  4  ?Subroutine33
  4  ?Subroutine34
  4  ?Subroutine35
  4  ?Subroutine36
  4  ?Subroutine37
  6  ?Subroutine38
  4  ?Subroutine39
 10  ?Subroutine4
  6  ?Subroutine40
  6  ?Subroutine41
  6  ?Subroutine42
  4  ?Subroutine43
  6  ?Subroutine44
  6  ?Subroutine45
  6  ?Subroutine46
  4  ?Subroutine47
  6  ?Subroutine48
  6  ?Subroutine49
 10  ?Subroutine5
  6  ?Subroutine50
  6  ?Subroutine51
  6  ?Subroutine52
  8  ?Subroutine53
  8  ?Subroutine54
 16  ?Subroutine6
 12  ?Subroutine7
 10  ?Subroutine8
 20  ?Subroutine9
  2  Error_Handler
 14  HAL_TIM_PeriodElapsedCallback
  4  Hi_pulsewidth
  4  Hi_pulsewidth11
  4  Low_pulsewidth
  2  R
156  SystemClock_Config
 32  T
     P
     t1
     t2
     t3
     p1
```

```
        p2
        p3
     4  bonus
    40  clearGFX
    38  command
     8  dcds
        dcd1
        dcd2
        dutyCycle
    32  digits
        fd1
        fd2
        fd3
        fd4
        fd5
        fd6
   934  drawChar
   634  drawDC
   270  drawDuty
   176  drawFreq
   130  drawLowRem
   280  drawPWT
   124  drawPixel
   128  drawRem
   282  drawWave
     4  duty
    36  graphicPosition
   312  htim2
        htim3
        htim4
        htim5
        halfSeconds
        count4
        freq
        dist
        distance
        frequency
    90  init
    30  initGraphicMode
 1'874  main
    60  number
     1  one
     4  pos
     4  pulse
     4  ref
    26  resetDisplay
     1  temp
    16  temp2
        prevDuty
        blocks
        remainder
     2  temp3
     1  three
     4  timePeriod
     1  two
    30  write


   444 bytes in section .bss
 5'870 bytes in section .text

 5'870 bytes of CODE memory
   444 bytes of DATA memory

Errors: none
Warnings: 5
```

## stm32f4xx_it.lst

```
###############################################################################
#
# IAR ANSI C/C++ Compiler V9.20.2.320/W64 for ARM         15/Apr/2022  14:33:51
# Copyright 1999-2021 IAR Systems AB.
#
#    Cpu mode         =  thumb
#    Endian           =  little
#    Source file      =
#        C:\Users\Student\OneDrive - Western Michigan
#        University\Documents\MIcrocontroller\Project\Core\Src\stm32f4xx_it.c
#    Command line     =
#        -f "C:\Users\Student\OneDrive - Western Michigan
#        University\Documents\MIcrocontroller\Project\EWARM\Project\Obj\Application\User\Core\stm32f4xx_it.o.rsp"
#        ("C:\Users\Student\OneDrive - Western Michigan
#        University\Documents\MIcrocontroller\Project\Core\Src\stm32f4xx_it.c"
#        -D USE_HAL_DRIVER -D STM32F429xx -lcN "C:\Users\Student\OneDrive -
#        Western Michigan
#        University\Documents\MIcrocontroller\Project\EWARM\Project\List\Application\User\Core"
#        -o "C:\Users\Student\OneDrive - Western Michigan
#        University\Documents\MIcrocontroller\Project\EWARM\Project\Obj\Application\User\Core"
#        --debug --endian=little --cpu=Cortex-M4 -e --fpu=VFPv4_sp
#        --dlib_config "C:\Program Files\IAR Systems\Embedded Workbench
#        9.0\arm\inc\c\DLib_Config_Full.h" -I "C:\Users\Student\OneDrive -
#        Western Michigan
#        University\Documents\MIcrocontroller\Project\EWARM/../Core/Inc\\" -I
#        "C:\Users\Student\OneDrive - Western Michigan
#        University\Documents\MIcrocontroller\Project\EWARM/../Drivers/STM32F4xx_HAL_Driver/Inc\\"
#        -I "C:\Users\Student\OneDrive - Western Michigan
#        University\Documents\MIcrocontroller\Project\EWARM/../Drivers/STM32F4xx_HAL_Driver/Inc/Legacy\\"
#        -I "C:\Users\Student\OneDrive - Western Michigan
#        University\Documents\MIcrocontroller\Project\EWARM/../Drivers/CMSIS/Device/ST/STM32F4xx/Include\\"
#        -I "C:\Users\Student\OneDrive - Western Michigan
#        University\Documents\MIcrocontroller\Project\EWARM/../Drivers/CMSIS/Include\\"
#        -Ohz) --dependencies=n "C:\Users\Student\OneDrive - Western Michigan
#        University\Documents\MIcrocontroller\Project\EWARM\Project\Obj\Application\User\Core\stm32f4xx_it.o.d"
#    Locale           =  C
#    List file        =
#        C:\Users\Student\OneDrive - Western Michigan
#        University\Documents\MIcrocontroller\Project\EWARM\Project\List\Application\User\Core\stm32f4xx_it.lst
#    Object file      =
#        C:\Users\Student\OneDrive - Western Michigan
#        University\Documents\MIcrocontroller\Project\EWARM\Project\Obj\Application\User\Core\stm32f4xx_it.o
#    Runtime model:
#      __CPP_Runtime   =  1
#      __SystemLibrary =  DLib
#      __dlib_version  =  6
#      __size_limit    =  32768|ARM.EW.LINKER
#
###############################################################################

C:\Users\Student\OneDrive - Western Michigan University\Documents\MIcrocontroller\Project\Core\Src\stm32f4xx_it.c
      1        /* USER CODE BEGIN Header */
      2        /**
      3          ******************************************************************************
      4          * @file    stm32f4xx_it.c
      5          * @brief   Interrupt Service Routines.
      6          ******************************************************************************
      7          * @attention
      8          *
      9          * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
     10           * All rights reserved.</center></h2>
     11           *
     12           * This software component is licensed by ST under BSD 3-Clause license,
     13           * the "License"; You may not use this file except in compliance with the
     14           * License. You may obtain a copy of the License at:
```

```
 15          *                        opensource.org/licenses/BSD-3-Clause
 16          *
 17          ******************************************************************************
 18          */
 19        /* USER CODE END Header */
 20
 21        /* Includes ------------------------------------------------------------------*/
 22        #include "main.h"
 23        #include "stm32f4xx_it.h"
 24        /* Private includes ----------------------------------------------------------*/
 25        /* USER CODE BEGIN Includes */
 26        /* USER CODE END Includes */
 27
 28        /* Private typedef -----------------------------------------------------------*/
 29        /* USER CODE BEGIN TD */
 30
 31        /* USER CODE END TD */
 32
 33        /* Private define ------------------------------------------------------------*/
 34        /* USER CODE BEGIN PD */
 35
 36        /* USER CODE END PD */
 37
 38        /* Private macro -------------------------------------------------------------*/
 39        /* USER CODE BEGIN PM */
 40
 41        /* USER CODE END PM */
 42
 43        /* Private variables ---------------------------------------------------------*/
 44        /* USER CODE BEGIN PV */
 45        //double pulsewidth = 0;
 46
 47        //EXTI
 48        extern int halfSeconds;
 49
 50        //TIM5
 51        int count1s = 0;
 52        int count1s2 = 0;
 53
 54        int count1s, seconds = 0;
 55        int count = 0;
 56        int count2 = 0;
 57        int count3 = 0;
 58        uint32_t risingedge1 = 0x0000;
 59        uint32_t fallingedge = 0x0000;
 60        uint32_t risingedge2 = 0x0000;
 61        extern float Hi_pulsewidth ;
 62        extern float Low_pulsewidth ;
 63        extern float frequency;
 64        extern float dutyCycle;
 65        extern float timePeriod;
 66
 67        float frequency_KHz;
 68        int count11 = 0;
 69        int countHi;
 70        int countLow;
 71        int countPeriod;
 72        uint32_t risingedge11 = 0;
 73        uint32_t fallingedge11 = 0;
 74        uint32_t risingedge22 = 0;
 75        extern float Hi_pulsewidth11;
 76        extern float Low_pulsewidth11;
 77        extern float distance;
 78        int count20us = 0;
 79        /* USER CODE END PV */
 80
 81        /* Private function prototypes -----------------------------------------------*/
 82        /* USER CODE BEGIN PFP */
```

```
83
84      /* USER CODE END PFP */
85
86      /* Private user code ---------------------------------------------------------*/
87      /* USER CODE BEGIN 0 */
88
89      /* USER CODE END 0 */
90
91      /* External variables --------------------------------------------------------*/
92      extern TIM_HandleTypeDef htim2;
93      extern TIM_HandleTypeDef htim3;
94      extern TIM_HandleTypeDef htim4;
95      extern TIM_HandleTypeDef htim5;
96      extern TIM_HandleTypeDef htim7;
97
98      /* USER CODE BEGIN EV */
99
100      /* USER CODE END EV */
101
102      /******************************************************************************/
103      /*           Cortex-M4 Processor Interruption and Exception Handlers          */
104      /******************************************************************************/
105      /**
106        * @brief This function handles Non maskable interrupt.
107        */
108      void NMI_Handler(void)
109      {
110        /* USER CODE BEGIN NonMaskableInt_IRQn 0 */
111
112        /* USER CODE END NonMaskableInt_IRQn 0 */
113        /* USER CODE BEGIN NonMaskableInt_IRQn 1 */
114
115        /* USER CODE END NonMaskableInt_IRQn 1 */
116      }
117
118      /**
119        * @brief This function handles Hard fault interrupt.
120        */
121      void HardFault_Handler(void)
122      {
123        /* USER CODE BEGIN HardFault_IRQn 0 */
124
125        /* USER CODE END HardFault_IRQn 0 */
126        while (1)
127        {
128          /* USER CODE BEGIN W1_HardFault_IRQn 0 */
129          /* USER CODE END W1_HardFault_IRQn 0 */
130        }
131      }
132
133      /**
134        * @brief This function handles Memory management fault.
135        */
136      void MemManage_Handler(void)
137      {
138        /* USER CODE BEGIN MemoryManagement_IRQn 0 */
139
140        /* USER CODE END MemoryManagement_IRQn 0 */
141        while (1)
142        {
143          /* USER CODE BEGIN W1_MemoryManagement_IRQn 0 */
144          /* USER CODE END W1_MemoryManagement_IRQn 0 */
145        }
146      }
147
148      /**
149        * @brief This function handles Pre-fetch fault, memory access fault.
150        */
```

```
151 void BusFault_Handler(void)
152 {
153   /* USER CODE BEGIN BusFault_IRQn 0 */
154
155   /* USER CODE END BusFault_IRQn 0 */
156   while (1)
157   {
158     /* USER CODE BEGIN W1_BusFault_IRQn 0 */
159     /* USER CODE END W1_BusFault_IRQn 0 */
160   }
161 }
162
163 /**
164   * @brief This function handles Undefined instruction or illegal state.
165   */
166 void UsageFault_Handler(void)
167 {
168   /* USER CODE BEGIN UsageFault_IRQn 0 */
169
170   /* USER CODE END UsageFault_IRQn 0 */
171   while (1)
172   {
173     /* USER CODE BEGIN W1_UsageFault_IRQn 0 */
174     /* USER CODE END W1_UsageFault_IRQn 0 */
175   }
176 }
177
178 /**
179   * @brief This function handles System service call via SWI instruction.
180   */
181 void SVC_Handler(void)
182 {
183   /* USER CODE BEGIN SVCall_IRQn 0 */
184
185   /* USER CODE END SVCall_IRQn 0 */
186   /* USER CODE BEGIN SVCall_IRQn 1 */
187
188   /* USER CODE END SVCall_IRQn 1 */
189 }
190
191 /**
192   * @brief This function handles Debug monitor.
193   */
194 void DebugMon_Handler(void)
195 {
196   /* USER CODE BEGIN DebugMonitor_IRQn 0 */
197
198   /* USER CODE END DebugMonitor_IRQn 0 */
199   /* USER CODE BEGIN DebugMonitor_IRQn 1 */
200
201   /* USER CODE END DebugMonitor_IRQn 1 */
202 }
203
204 /**
205   * @brief This function handles Pendable request for system service.
206   */
207 void PendSV_Handler(void)
208 {
209   /* USER CODE BEGIN PendSV_IRQn 0 */
210
211   /* USER CODE END PendSV_IRQn 0 */
212   /* USER CODE BEGIN PendSV_IRQn 1 */
213
214   /* USER CODE END PendSV_IRQn 1 */
215 }
216
217 /**
218   * @brief This function handles System tick timer.
```

```
219          */
220         void SysTick_Handler(void)
221         {
222           /* USER CODE BEGIN SysTick_IRQn 0 */
223             count20us+=1;
224             if (count20us == 1)
225           {
226             HAL_GPIO_WritePin(GPIOC, GPIO_PIN_2, GPIO_PIN_RESET);//Set PC2 for 20us
227           }
228             if (count20us == (2500-1))
229           {
230             HAL_GPIO_WritePin(GPIOC, GPIO_PIN_2, GPIO_PIN_SET);//Set PC2 for 20us
231             count20us = 0;
232           }
233
234
235           /* USER CODE END SysTick_IRQn 0 */
236
237           /* USER CODE BEGIN SysTick_IRQn 1 */
238
239           /* USER CODE END SysTick_IRQn 1 */
240         }
241
242         /******************************************************************************/
243         /* STM32F4xx Peripheral Interrupt Handlers                                    */
244         /* Add here the Interrupt Handlers for the used peripherals.                  */
245         /* For the available peripheral interrupt handler names,                      */
246         /* please refer to the startup file (startup_stm32f4xx.s).                    */
247         /******************************************************************************/
248
249         /**
250           * @brief This function handles EXTI line[9:5] interrupts.
251           */
252         void EXTI9_5_IRQHandler(void)
253         {
254           /* USER CODE BEGIN EXTI9_5_IRQn 0 */
255
256           if ((!(GPIOE->IDR & 0x80)) && (halfSeconds == 0)) // Falling Edge detection, start PE7 (EXTI7)
257           {
258            HAL_TIM_OC_Start_IT(&htim3, TIM_CHANNEL_4); // Start Buzzer and indicator
259            HAL_GPIO_WritePin(GPIOG, GPIO_PIN_12, GPIO_PIN_SET);// Forward direction control_Pin15: PG12
260
261            //HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_3); //Starts the TIM Input Capture measurement in interrupt mode.
262
263            // HC-SR04 ultrasonic sensor
264         //   HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_4); //Starts the TIM Input Capture measurement in interrupt mode.
265         //   HAL_TIM_PWM_Start(&htim5, TIM_CHANNEL_3);// to make a pulse with 25 us width and 55 ms period. or
HAL_TIM_PWM_Start(&htim5, TIM_CHANNEL_4); check it !!!
266
267           }
268            if (!(GPIOE->IDR & 0x100)) // Falling Edge detection on Stop pin  PE8(EXTI8); wait for start
269           {
270             HAL_GPIO_WritePin(GPIOG, GPIO_PIN_12, GPIO_PIN_RESET);// Forward direction control_Pin15: PG12
271             HAL_GPIO_WritePin(GPIOG, GPIO_PIN_11, GPIO_PIN_RESET);//Reverse direction control_Pin16: PG11
272             halfSeconds = 0;
273             HAL_TIM_PWM_Stop_IT(&htim4, TIM_CHANNEL_4);// stop PWM
274             HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_RESET);//TURN Off Red LED (IR Emitter LED)
275           }
276
277
278           /* USER CODE END EXTI9_5_IRQn 0 */
279           HAL_GPIO_EXTI_IRQHandler(Start_Input_Pin);
280           HAL_GPIO_EXTI_IRQHandler(Stop_Input_Pin);
281           /* USER CODE BEGIN EXTI9_5_IRQn 1 */
282
283           /* USER CODE END EXTI9_5_IRQn 1 */
284         }
285
```

```
286 /**
287   * @brief This function handles TIM2 global interrupt.
288   */
289 void TIM2_IRQHandler(void)
290 {
291   /* USER CODE BEGIN TIM2_IRQn 0 */
292  if ((HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_11)) && (count11 == 0)) // Rising-1 edge detection
293   {
294
295     risingedge11 = TIM2->CCR4; // record the count11er value
296     count11 = 1;
297   }
298    else if((!(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_11))) && (count11 == 1))// Falling Edge detection
299   {
300     fallingedge11 = TIM2->CCR4; //record the count11er value
301     count11 = 2;
302   }
303    else if ((HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_11)) && (count11 == 2)) // Rising-1 edge detection
304   {
305
306     risingedge22 = TIM2->CCR4; // record the count11er value
307     countHi = (fallingedge11 - risingedge11);
308     countLow = (risingedge22 - fallingedge11);
309     countPeriod = countHi + countLow;
310     Hi_pulsewidth11 = ((fallingedge11 - risingedge11)* 0.02); // Hi_Pulsewidth (us)= CCRx * Clock period,
311     distance = ((Hi_pulsewidth11 *  0.0068 ) - 0.3026); // Distance (in) equation culculated from calibration graph
312     TIM2->CNT = 0; // Reset the count11er register
313     count11 = 0;
314   }
315
316   /* USER CODE END TIM2_IRQn 0 */
317   HAL_TIM_IRQHandler(&htim2);
318   /* USER CODE BEGIN TIM2_IRQn 1 */
319
320   /* USER CODE END TIM2_IRQn 1 */
321 }
322
323 /**
324   * @brief This function handles TIM3 global interrupt.
325   */
326 void TIM3_IRQHandler(void)
327 {
328   /* USER CODE BEGIN TIM3_IRQn 0 */
329
330   if (halfSeconds < 12)
331   {
332     if (count2 == 0)
333     {
334       TIM3->CCR4 += 4545;// Tcnt for one periode = (1/f) * fapb1_clock = (1/5.5kHz)* 50MHz = 9090
335       count2 = 1;
336     }
337     else if (count2 == 1)
338     {
339     TIM3->CCR4 += (9090-4545);// 50% duty cycle:(50/100)*9090=4545
340     count2 = 0;
341     count1s += 1;
342     }
343     count3= 1;
344   }
345   if ((count1s == 2750) && (count3 != 2)) // To mak 1 halfSeconds, 1/5.5KHz =0.1818ms; 0.1818*X/2 = 1000ms; x =
5500/2=2750
346   //else if ((count1s == 2750) && (halfSeconds <= 12)) // To mak 1 halfSeconds, 1/5.5KHz =0.1818ms; 0.1818*X/2 = 1000ms; x
= 5500/2=2750
347   {
348     count1s = 0;
349     HAL_GPIO_TogglePin(GPIOG, GPIO_PIN_13); //Toggle GREEN LED
350     halfSeconds +=1;
351
```

```
352          }
353
354          ////////////////////////////////////////////////////
355          //if (((GPIOE->IDR & 0x80)) && (halfSeconds == 12))// Falling Edge is not detected on start and after 12 halfSeconds
356          if (((count3== 1)) && (halfSeconds == 12))// Falling Edge is not detected on start and after 12 halfSeconds
357          //if (count3 == 1)
358          {
359            HAL_TIM_OC_Stop_IT(&htim3, TIM_CHANNEL_4); // Turn off Buzzer
360            HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_SET);//TURN ON Red LED (IR Emitter LED)
361
362            //PWM signal driving the motor on
363            HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_4);// to make a pulse with f= 30KHz and duty cycle=50%  0.016666 ms
width and 0.03333 ms period.
364            TIM4-> ARR = 1666; // (1/30KHz)*50MHz=1666
365            TIM4-> CCR4 = 833; // (1/2*30KHz)*50MHz=833; 50% dutyc cycle
366            count3= 2;
367          }
368        ////////////////////////////////////////////////////////////////
369
370        //  if ((TIM4->CCR4 == 0) && ( 12 <= halfSeconds && halfSeconds <= 32))
371          //if ((TIM4->CCR4 == 0) && (halfSeconds <= 32))
372          //count1s2 = 0;
373          if (TIM4->CCR4 == 0)
374          {
375            if (count2 == 0)
376            {
377              TIM3->CCR4 += 7143;// Tcnt for one periode = (1/f) * fapb1_clock = (1/3.5kHz)* 50MHz = 14286
378              count2 = 1;
379            }
380            else if (count2 == 1)
381            {
382            TIM3->CCR4 += (14286-7143);// 50% duty cycle:(50/100)*14286=7143
383            count3 = 0;
384            count1s2 += 1;
385            }
386          }
387           if((count1s2 == 1375) && (halfSeconds <= 32)) // To make 1 halfSeconds, 1/3.5KHz =0.2857ms; 0.2857*2X = 1000ms; x =
3500/4 for 2 blinks/sec
388             {
389               count1s2 = 0;
390               HAL_GPIO_TogglePin(GPIOG, GPIO_PIN_13); //Toggle GREEN LED
391               halfSeconds +=1;
392             }
393
394          else if ( halfSeconds > 32)
395          {
396           HAL_TIM_OC_Stop_IT(&htim3, TIM_CHANNEL_4); // Stop Buzzer Start
397           HAL_GPIO_WritePin(GPIOG, GPIO_PIN_13, GPIO_PIN_RESET);//TURN Off Green LED (Indicator LED)
398           HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_RESET);//TURN Off Red LED (IR Emitter LED)
399           halfSeconds = 0;
400           count2 = 0;
401           count3 = 0;
402           HAL_GPIO_WritePin(GPIOG, GPIO_PIN_12, GPIO_PIN_RESET);//PG12 on, move Forward
403          }
404
405          ////////////////////////////////////////////////////////////////////////////
406        //    if ((GPIOB->IDR & 0x01) && (count == 0)) // Rising-1 edge detection
407        //  {
408        //    risingedge1 = TIM3->CCR3;
409        //    count = 1;
410        //  }
411        //   else if ((!(GPIOB->IDR & 0x01)) && (count == 1))// Falling Edge detection
412        //  {
413        //    fallingedge = TIM3->CCR3;
414        //    count = 2;
415        //  }
416        //  else if ((GPIOB->IDR & 0x01) && (count == 2)) //Rising-2 edge detection
417        //  {
```

```
418       //    risingedge2 = TIM3->CCR3;
419       //    timePeriod = ((risingedge2 - risingedge1)*0.02) ; // HLCK=100MHz  and APB1=50MHz. Period (us).1/50MHz = 0.02us
420       //    frequency = (1000000/timePeriod); // frequency (HZ)=1/timePeriod(us)= 10*6/timePeriod
421       //    Hi_pulsewidth = ((fallingedge - risingedge1)* 0.02); // Hi_Pulsewidth (ms).
422       //
423       //    Low_pulsewidth = ((risingedge2 - fallingedge) * 0.02); // Low_pulsewidth (ms).
424       //    //printf("Period = %lf msec\n", timePeriod);
425       //    dutyCycle = (Hi_pulsewidth/timePeriod)*100;
426       //
427       //
428       //    count = 0;
429       //    TIM3->CNT = 0; // Reset the counter register
430       //
431       //    frequency_KHz = frequency/1000;
432       //  }
433       ////////////////////////////////////////////////////////////////////////////////
434
435         /* USER CODE END TIM3_IRQn 0 */
436         HAL_TIM_IRQHandler(&htim3);
437         /* USER CODE BEGIN TIM3_IRQn 1 */
438
439         /* USER CODE END TIM3_IRQn 1 */
440       }
441
442       /**
443         * @brief This function handles TIM4 global interrupt.
444         */
445       void TIM4_IRQHandler(void)
446       {
447         /* USER CODE BEGIN TIM4_IRQn 0 */
448
449         /* USER CODE END TIM4_IRQn 0 */
450         HAL_TIM_IRQHandler(&htim4);
451         /* USER CODE BEGIN TIM4_IRQn 1 */
452
453         /* USER CODE END TIM4_IRQn 1 */
454       }
455
456       /**
457         * @brief This function handles TIM5 global interrupt.
458         */
459       void TIM5_IRQHandler(void)
460       {
461         /* USER CODE BEGIN TIM5_IRQn 0 */
462
463
464       if ((GPIOA->IDR & 0x0008) && (count == 0)) // Rising-1 edge detection
465         {
466           risingedge1 = TIM5->CCR4;
467           count = 1;
468         }
469         else if ((!(GPIOA->IDR & 0x0008)) && (count == 1))// Falling Edge detection
470         {
471           fallingedge = TIM5->CCR4;
472           count = 2;
473         }
474       else if ((GPIOA->IDR & 0x0008) && (count == 2)) //Rising-2 edge detection
475         {
476           risingedge2 = TIM5->CCR4;
477           timePeriod = ((risingedge2 - risingedge1)*0.02) ; // HLCK=100MHz  and APB1=50MHz. Period (us).1/50MHz = 0.02us
478           frequency = (1000000/timePeriod); // frequency (HZ)=1/timePeriod(us)= 10*6/timePeriod
479           Hi_pulsewidth = ((fallingedge - risingedge1)* 0.02); // Hi_Pulsewidth (ms).
480
481           Low_pulsewidth = ((risingedge2 - fallingedge) * 0.02); // Low_pulsewidth (ms).
482           //printf("Period = %lf msec\n", timePeriod);
483           dutyCycle = (Hi_pulsewidth/timePeriod)*100;
484
485
```

```
486           count = 0;
487           TIM5->CNT = 0; // Reset the counter register
488
489           frequency_KHz = frequency/1000;
490         }
491       ////////////////////////////////////////////////////////////////////////
492       //  while (seconds <= 5)
493       //  {
494       //    if (count2 == 0)
495       //    {
496       //      TIM5->CCR1 += 4545;// Tcnt for one periode = (1/f) * fapb1_clock = (1/5.5kHz)* 50MHz = 9090
497       //      count2 = 1;
498       //    }
499       //    else if (count2 == 1)
500       //    {
501       //    TIM5->CCR1 += (9090-4545);// 50% duty cycle:(50/100)*9090=4545
502       //    count2 = 0;
503       //    count1s += 1;
504       //    }
505       //    if(count1s == 5500) // To mak 1 seconds, 1/5.5KHz =0.1818ms; 0.1818*X = 1000ms; x = 5500
506       //      {
507       //        count1s = 0;
508       //        HAL_GPIO_TogglePin(GPIOG, Indicator_GreenLED_Pin); //Toggle GREEN LED
509       //        seconds +=1;
510       //      }
511       //  }
512       //
513       //
514       //
515       //  while (5 < seconds && seconds <= 16)
516       //  {
517       //    if (count2 == 0)
518       //    {
519       //      TIM5->CCR1 += 7143;// Tcnt for one periode = (1/f) * fapb1_clock = (1/3.5kHz)* 50MHz = 14286
520       //      count2 = 1;
521       //    }
522       //    else if (count2 == 1)
523       //    {
524       //    TIM5->CCR1 += (14286-7143);// 50% duty cycle:(50/100)*14286=7143
525       //    count2 = 0;
526       //    count1s += 1;
527       //    }
528       //    if(count1s == 1750) // To mak 1 seconds, 1/3.5KHz =0.2857ms; 0.2857*X = 1000ms; x = 3500/2 for 2 blinks/sec
529       //      {
530       //        count1s = 0;
531       //        HAL_GPIO_TogglePin(GPIOG, Indicator_GreenLED_Pin); //Toggle GREEN LED
532       //        seconds +=1;
533       //      }
534       //  }
535       //
536       //  if ( seconds > 16)
537       //  {
538       //   HAL_TIM_OC_Stop_IT(&htim5, TIM_CHANNEL_1); // Stop Buzzer Start
539       //   HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_RESET);//TURN Off Red LED (IR Emitter LED)
540       //   seconds = 0;
541       //   count2 = 0;
542       //  }
543
544
545         /* USER CODE END TIM5_IRQn 0 */
546         HAL_TIM_IRQHandler(&htim5);
547         /* USER CODE BEGIN TIM5_IRQn 1 */
548
549         /* USER CODE END TIM5_IRQn 1 */
550       }
551
552       /**
553         * @brief This function handles TIM7 global interrupt.
```

```
554           */
555         void TIM7_IRQHandler(void)
556         {
557           /* USER CODE BEGIN TIM7_IRQn 0 */
558
559           /* USER CODE END TIM7_IRQn 0 */
560           HAL_TIM_IRQHandler(&htim7);
561           /* USER CODE BEGIN TIM7_IRQn 1 */
562
563           /* USER CODE END TIM7_IRQn 1 */
564         }
565
566         /* USER CODE BEGIN 1 */
567
568         /* USER CODE END 1 */
569

Maximum stack usage in bytes:

.cstack Function
------- --------
    0   BusFault_Handler
    0   DebugMon_Handler
   16   EXTI9_5_IRQHandler
     0   -> HAL_GPIO_EXTI_IRQHandler
    16   -> HAL_GPIO_EXTI_IRQHandler
    16   -> HAL_GPIO_WritePin
    16   -> HAL_TIM_OC_Start_IT
    16   -> HAL_TIM_PWM_Stop_IT
    0   HardFault_Handler
    0   MemManage_Handler
    0   NMI_Handler
    0   PendSV_Handler
    0   SVC_Handler
   16   SysTick_Handler
    16   -> HAL_GPIO_WritePin
   24   TIM2_IRQHandler
    24   -> HAL_GPIO_ReadPin
     0   -> HAL_TIM_IRQHandler
    24   -> __aeabi_d2f
    24   -> __aeabi_dadd
    24   -> __aeabi_dmul
    24   -> __aeabi_f2d
    24   -> __aeabi_ui2d
   32   TIM3_IRQHandler
    32   -> HAL_GPIO_TogglePin
    32   -> HAL_GPIO_WritePin
     0   -> HAL_TIM_IRQHandler
    32   -> HAL_TIM_OC_Stop_IT
    32   -> HAL_TIM_PWM_Start
    0   TIM4_IRQHandler
     0   -> HAL_TIM_IRQHandler
   48   TIM5_IRQHandler
     0   -> HAL_TIM_IRQHandler
    48   -> __aeabi_d2f
    48   -> __aeabi_dmul
    48   -> __aeabi_ui2d
    0   TIM7_IRQHandler
     0   -> HAL_TIM_IRQHandler
    0   UsageFault_Handler


Section sizes:

Bytes  Function/Label
-----  --------------
    4  ??DataTable7
    4  ??DataTable7_1
```

```
   4  ??DataTable7_10
   4  ??DataTable7_11
   4  ??DataTable7_12
   4  ??DataTable7_13
   4  ??DataTable7_14
   4  ??DataTable7_15
   4  ??DataTable7_16
   4  ??DataTable7_17
   4  ??DataTable7_18
   4  ??DataTable7_19
   4  ??DataTable7_2
   4  ??DataTable7_20
   4  ??DataTable7_21
   8  ??DataTable7_22
   4  ??DataTable7_23
   4  ??DataTable7_24
   4  ??DataTable7_25
   4  ??DataTable7_26
   4  ??DataTable7_27
   4  ??DataTable7_28
   4  ??DataTable7_29
   4  ??DataTable7_3
   4  ??DataTable7_30
   4  ??DataTable7_31
   4  ??DataTable7_32
   4  ??DataTable7_33
   4  ??DataTable7_34
   4  ??DataTable7_35
   4  ??DataTable7_4
   4  ??DataTable7_5
   4  ??DataTable7_6
   4  ??DataTable7_7
   4  ??DataTable7_8
   4  ??DataTable7_9
  10  ?Subroutine0
  10  ?Subroutine1
  12  ?Subroutine2
  10  ?Subroutine3
  10  ?Subroutine4
   8  ?Subroutine5
   2  BusFault_Handler
   2  DebugMon_Handler
 104  EXTI9_5_IRQHandler
   2  HardFault_Handler
   2  MemManage_Handler
   2  NMI_Handler
   2  PendSV_Handler
   2  SVC_Handler
  56  SysTick_Handler
 160  TIM2_IRQHandler
 268  TIM3_IRQHandler
   6  TIM4_IRQHandler
 212  TIM5_IRQHandler
   6  TIM7_IRQHandler
   2  UsageFault_Handler
  20  count
      risingedge1
      fallingedge
      risingedge2
      frequency_KHz
  28  count11
      countHi
      countLow
      countPeriod
      risingedge11
      fallingedge11
      risingedge22
  16  count1s
```

```
        count1s2
        count2
        count3
     4  count20us
     4  seconds


    72 bytes in section .bss
 1'036 bytes in section .text

 1'036 bytes of CODE memory
    72 bytes of DATA memory

Errors: none
Warnings: none
```