```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "adc.h"
#include "dma.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "led.h"
#include "interrupt.h"
#include "stdio.h"
#include "string.h"
#include "lcd.h"
#include "i2c_hal.h"
#include "seg.h"
#include "ds18b20.h"


/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
extern struct keys key[4];
extern char rx_data;
extern char rx_arry[50];
extern char rx_pointer;
extern uint PA1_freq,PA1_duty;

/* 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 ---------------------------------------------------------*/

/* USER CODE BEGIN PV */
char lcd_arry[50];
char lcd_view;
u16 ADC1_array[2];
u16 ADC2_array[3];

uint PA7_freq = 1000;
uint PA7_duty = 70;
uint PA7_autoreload,PA7_compare;
```

```c

//温度
float temp_float;
__IO uint32_t  ds18b20_uwTick;


char PARA_change_state = 0;//0:FH   1:AH    2:TH
char FSET_change_state = 0;//0:FH   1:VP    2:TT
char record_flag;//开始记录标志位
__IO uint32_t  key_uwTick;

int R37_adc_record_array[100];//ADC记录数组
int PA1_freq_record_array[100];//频率记录数组
int PA1_duty_record_array[100];//占空比记录数组

char RECD_clear_flag;//统计参数清零

int PARA_FH_value = 2000;
float PARA_AH_value = 3.0;
int PARA_TH_value = 30;

int FSET_FP_value = 1;
float FSET_VP_value = 0.9;
int FSET_TT_value = 6;

char FSET_voltage_flag;//开始记录电压标志位
char FSET_pulse_flag;//开始频率和占空比标志位
__IO uint32_t FSET_voltage_flag_uwTick;
__IO uint32_t FSET_pulse_flag_uwTick;
char FSET_finish_flag;//记录完毕标志位

char Reset_flag;//复位标志位
__IO uint32_t  led_uwTick;

char led_num;

char F_alarm_flag;//频率报警信号
char V_alarm_flag;//电压报警信号
char T_alarm_flag;//温度报警信号

char RECD_FN_value;
char RECD_AN_value;
char RECD_TN_value;
__IO uint32_t  signal_record_uwTick;
__IO uint32_t  signal_review_uwTick;
__IO uint32_t  TT_time_uwTick;

int FH_reg,TH_reg;//上限参数寄存变量
float AH_reg;


//char maopao_array[10] = {0,6,8,2,9,1,7,3,5,4};
char maopao_array[50];

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void key_proc(void);
void rx_proc(void);
void lcd_proc(void);
void pwm_proc(void);
void ds18b20_proc(void);
void led_proc(void);
void signal_record(void);//信号记录函数
void signal_review(void);//信号回放函数
void alarm_proc(void);//报警函数
void maopao_sort(int length,char maopao_array[]);

/* 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_DMA_Init();
  MX_TIM6_Init();
  MX_USART1_UART_Init();
  MX_TIM8_Init();
  MX_TIM16_Init();
  MX_ADC1_Init();
  MX_ADC2_Init();
  MX_TIM2_Init();
  MX_TIM3_Init();
  MX_TIM17_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  led_disp(0x00);

  HAL_TIM_Base_Start_IT(&htim6);

  HAL_UART_Receive_IT(&huart1,(uint8_t *)&rx_data,1);

  LCD_Init();
  LCD_Clear(Black);
  LCD_SetTextColor(White);
  LCD_SetBackColor(Black);

  I2CInit();
//
  HAL_TIM_IC_Start_IT(&htim16,TIM_CHANNEL_1);//按键扫描定时器中断
  HAL_TIM_IC_Start_IT(&htim8,TIM_CHANNEL_1);
  HAL_TIM_IC_Start_IT(&htim8,TIM_CHANNEL_2);
  HAL_TIM_IC_Start_IT(&htim2,TIM_CHANNEL_2);//测量PA1的频率和占空比
//  HAL_TIM_IC_Start_IT(&htim2,TIM_CHANNEL_3);

  HAL_ADC_Start_DMA(&hadc1,(uint32_t *)ADC1_array,2);
  HAL_ADC_Start_DMA(&hadc2,(uint32_t *)ADC2_array,3);

  HAL_TIM_PWM_Start(&htim17,TIM_CHANNEL_1);//PA7--PWM输出

  led_disp(0x00);
  ds18b20_init_x();
  while((int)ds18b20_read() == 85) {}

  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
     key_proc();
     rx_proc();
     lcd_proc();
     ds18b20_proc();
     pwm_proc();
     led_proc();
     signal_record();
     signal_review();
     alarm_proc();
  }
  /* 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_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = RCC_PLLM_DIV3;
  RCC_OscInitStruct.PLL.PLLN = 20;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
  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_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/* USER CODE BEGIN 4 */
```

```c
void maopao_sort(int length, char maopao_array[])
{
        char temp;//交换两个变量的值需要一个中间变量
        //如果length为10
        //总共比较：length-1次
        //i表示每轮比较结束元素的索引+1,
        //第一轮：8+1
        //第一轮：7+1
        //第一轮：6+1
        //...
        //第一轮：0+1
        for(int i=length-1;i>0;i--)
        {
                //每次都从第一个元素开始相邻比较，结束元素索引不同
                //0----8，<9:第0个元素和第1个元素比较，...，第8个元素和第9个元素比较
                //0----0，<1: 第0个元素和第1个元素比较
                for(int j=0;j<i;j++)
                {
                        if(maopao_array[j]> maopao_array[j+1])
                        {
                                temp = maopao_array[j+1];
                                maopao_array[j+1]= maopao_array[j];
                                maopao_array[j] = temp;
                        }

                }
        }

        //打印查看信息
        for(int i=0;i<length;i++)
        {
                printf("%c\n",maopao_array[i]);
        }
}
void alarm_proc(void)
{
    //从报警参数界面退出时，新的 FH、AH 和TH 参数生效
    if(lcd_view != 1) //非报警参数界面时更新参数
    {
        FH_reg = PARA_FH_value;
        AH_reg = PARA_AH_value;
        TH_reg = PARA_TH_value;
    }

    //大于上限，只累加一次
    //F_alarm_flag为0表示频率值低于上限或者初始状态
    //+10和-10是设置一个滞环，减小波动的影响
    //上大于上限+10，且当前频率值低于下限，则累加一次，同时标志位拉高说明此时频率值大于上限值
    if(PA1_freq > FH_reg+10 && (F_alarm_flag == 0))
    {
        F_alarm_flag = 1;
        RECD_FN_value++;
    }
    else if(PA1_freq <= FH_reg-10)
        F_alarm_flag = 0;

    //对电压也是同样的逻辑
    //但是电压是浮点数，不能判等
    if(ADC2_array[0]*3.3/4096 > AH_reg && (V_alarm_flag == 0))
    {
        V_alarm_flag = 1;
        RECD_AN_value++;
    }
    else if(ADC2_array[0]*3.3/4096  <= AH_reg)
        V_alarm_flag = 0;

    //温度带有小数，而上限参数是整型，所以放大10倍，小数也进行比较
    if((int)(temp_float*10) > (TH_reg*10) && (T_alarm_flag == 0))
    {
        T_alarm_flag = 1;
        RECD_TN_value++;
    }
    else if((int)(temp_float*10) <= (TH_reg*10))
        T_alarm_flag = 0;

    //统计值清零
    if(RECD_clear_flag)
    {
        RECD_FN_value = 0;
        RECD_AN_value = 0;
        RECD_TN_value = 0;
        RECD_clear_flag = 0;//统计值清零标志拉低
    }

    //复位，设备回到初始状态
    if(Reset_flag)
    {
        Reset_flag = 0;

        //统计值复位
        RECD_FN_value = 0;
        RECD_AN_value = 0;
        RECD_TN_value = 0;

        //上限值复位
        PARA_FH_value = 2000;
        PARA_AH_value = 3.0;
        PARA_TH_value = 30;

        //记录值复位
        FSET_FP_value = 1;
        FSET_VP_value = 0.9;
        FSET_TT_value = 6;

        //界面复位
        lcd_view = 0;
        PARA_change_state = 0;//0:FH    1:AH    2:TH
        FSET_change_state = 0;//0:FH    1:VP    2:TT

    }

}
void signal_record(void)
{
    //每100ms记录一次数据到数组中
    if(uwTick - signal_record_uwTick < 100) return;
        signal_record_uwTick = uwTick;
    static int i;

    //开始记录
    if(record_flag == 1)
    {
        R37_adc_record_array[i] = ADC2_array[0];
        PA1_freq_record_array[i] = PA1_freq;
        PA1_duty_record_array[i] = PA1_duty;

        i++;

        //记录时间达到
        if(uwTick - TT_time_uwTick > FSET_TT_value*1000)
        {
            i = 0;//数组索引复位
            record_flag = 0;//标志位拉低
            FSET_finish_flag = 1;//记录结束标志拉高
        }
    }
}

void signal_review(void)
{
    //每100ms回放一次数据
    if(uwTick - signal_review_uwTick < 100) return;
```

```
        signal_review_uwTick = uwTick;
    static int j;

    //开始电压回放
    if(FSET_voltage_flag)
    {
        PA7_freq = 1000;
        PA7_duty = (int)((R37_adc_record_array[j]*3.3/4096-3.3)*90.0/(3.3-FSET_VP_value) + 100.0);
        if((R37_adc_record_array[j]*3.3/4096) < FSET_VP_value)
            PA7_duty = 10;
        j++;

        //记录时间达到
        if(uwTick - FSET_voltage_flag_uwTick > FSET_TT_value*1000)
        {
            FSET_voltage_flag = 0;//标志位拉低
            j = 0;//数组索引复位
            PA7_duty = 0;//回放结束，输出低电平
        }
    }

    //开始脉冲回放
    if(FSET_pulse_flag)
    {
        PA7_freq = PA1_freq_record_array[j]/FSET_FP_value;
        PA7_duty = PA1_duty_record_array[j];
        j++;

        //记录时间达到
        if(uwTick - FSET_pulse_flag_uwTick > FSET_TT_value*1000)
        {
            FSET_pulse_flag = 0;//标志位拉低
            j = 0;//数组索引复位
            PA7_duty = 0;//回放结束，输出低电平
        }
    }
}
void key_proc(void)
{
    //每50ms执行一次按键处理
    if((uwTick - key_uwTick < 50)) return;
    key_uwTick = uwTick;

    for(int i=0;i<4;i++)
    {
        //正在记录数据，所有标志位清零，按键均无效
        if(record_flag)
        {
            key[i].short_flag = 0;
            key[i].long_flag = 0;
        }//记录结束，按键按下，清一次屏
        else if(key[i].short_flag == 1 || key[i].long_flag == 1)
            LCD_Clear(Black);
    }


    if(key[0].short_flag == 1)
    {
        key[0].short_flag = 0;
        lcd_view++;
        //界面切换的时候，复位参数初始状态
        if(lcd_view == 4) lcd_view = 0;
        if(lcd_view == 1) PARA_change_state = 0;
        if(lcd_view == 3) FSET_change_state = 0;
    }

    if(key[1].short_flag == 1)
    {
        key[1].short_flag = 0;

        if(lcd_view == 0)
        {
            //开始记录
            record_flag = 1;
            TT_time_uwTick = uwTick;//开始计时
            FSET_finish_flag = 0;//记录结束标志位拉低，初始状态没有记录数据的时候，不能执行回放
        }

        if(lcd_view == 1)
        {
            //选中参数状态自加，超出范围，复位
            PARA_change_state++;
            if(PARA_change_state == 3) PARA_change_state = 0;
        }

        //统计值清零
        if(lcd_view == 2) RECD_clear_flag = 1;

        if(lcd_view == 3)
        {
            FSET_change_state++;
            if(FSET_change_state == 3) FSET_change_state = 0;
        }
    }

    if(key[2].short_flag == 1)
    {
        key[2].short_flag = 0;
        if(lcd_view == 1)
        {
            //根据选中的参数执行自加或自减
            //超出范围需要置位
            switch(PARA_change_state)
            {
                case 0:
                    PARA_FH_value+=1000;
                    if(PARA_FH_value >= 10000) PARA_FH_value = 10000;
                    break;
                case 1:
                    PARA_AH_value+=0.3f;
                    if(PARA_AH_value >= 3.3f) PARA_AH_value = 3.3f;
                    break;
                case 2:
                    PARA_TH_value+=1;
                    if(PARA_TH_value >= 80) PARA_TH_value = 80;
                    break;
            }
        }

        if(lcd_view == 3)
        {
            switch(FSET_change_state)
            {
                case 0:
                    FSET_FP_value+=1;
                    if(FSET_FP_value >= 10) FSET_FP_value = 10;
                    break;
                case 1:
                    FSET_VP_value+=0.3f;
                    if(FSET_VP_value >= 3.3f) FSET_VP_value = 3.3f;
                    break;
                case 2:
                    FSET_TT_value+=2;
                    if(FSET_TT_value >= 10) FSET_TT_value = 10;
                    break;
            }
        }

        if(lcd_view == 0)
        {
            //FSET_finish_flag为高说明已经记录过数据了，同时此刻并没有开始记录数据
            //记录数据和回放不能同时进行
            if(FSET_finish_flag)
            {
                FSET_voltage_flag = 1;
                FSET_voltage_flag_uwTick = uwTick;//开始计时
            }
        }

    }

    if(key[3].short_flag == 1)
    {
        key[3].short_flag = 0;
        if(lcd_view == 1)
        {
            switch(PARA_change_state)
            {
                case 0:
                    PARA_FH_value-=1000;
                    if(PARA_FH_value <= 1000) PARA_FH_value = 1000;
                    break;
                case 1:
                    PARA_AH_value-=0.3f;
                    if(PARA_AH_value <= 0.0f) PARA_AH_value = 0.0f;
                    break;
                case 2:
                    PARA_TH_value-=1;
                    if(PARA_TH_value <= 0) PARA_TH_value = 0;
                    break;
            }
        }

        if(lcd_view == 3)
        {
            switch(FSET_change_state)
            {
                case 0:
                    FSET_FP_value-=1;
                    if(FSET_FP_value <= 1) FSET_FP_value = 1;
                    break;
                case 1:
                    FSET_VP_value-=0.3f;
                    if(FSET_VP_value <= 0.0f) FSET_VP_value = 0.0f;
                    break;
                case 2:
                    FSET_TT_value-=2;
                    if(FSET_TT_value <= 2) FSET_TT_value = 2;
                    break;
            }
        }

        if(lcd_view == 0)
        {
            if(FSET_finish_flag)
            {
                FSET_pulse_flag = 1;
                FSET_pulse_flag_uwTick = uwTick;
            }

        }

    }

    //复位
    if(key[2].long_flag && key[3].long_flag)
    {
        key[2].long_flag = 0;
        key[3].long_flag = 0;
        Reset_flag = 1;
    }

}
```

```
void led_proc(void)
{
    //每0.1s执行一次，实现翻转功能
    if(uwTick - led_uwTick < 100) return;
        led_uwTick = uwTick;

    //满足条件，翻转，否则清零
    if(record_flag == 1)
        led_num ^= 0x01;
    else led_num &= 0xfe;

    if(FSET_pulse_flag == 1)
        led_num ^= 0x02;
    else led_num &= 0xfd;

    if(FSET_voltage_flag == 1)
        led_num ^= 0x04;
    else led_num &= 0xfb;

    if(F_alarm_flag == 1)
        led_num |= 0x08;
    else led_num &= 0xf7;

    if(V_alarm_flag == 1)
        led_num |= 0x10;
    else led_num &= 0xef;

    if(T_alarm_flag == 1)
        led_num |= 0x20;
    else led_num &= 0xdf;

    //显示LED
    led_disp(led_num);

}
void ds18b20_proc(void)
{
    //每0.1s读取一次温度值
    if(uwTick - ds18b20_uwTick < 100)    return;
    ds18b20_uwTick = uwTick;
    temp_float = ds18b20_read();
}


void pwm_proc(void)
{
    //配置PWM输出频率与占空比
    PA7_autoreload = 1000000/PA7_freq;
    PA7_compare = PA7_autoreload*PA7_duty/100;
    __HAL_TIM_SetAutoreload(&htim17,PA7_autoreload);
    __HAL_TIM_SetCompare(&htim17,TIM_CHANNEL_1,PA7_compare);
}

void lcd_proc(void)
{
    //显示
    if(lcd_view == 0)
    {
        //避免高亮的影响，保持原本的背景色
        LCD_SetBackColor(Black);
        sprintf(lcd_arry,"        DATA");
        LCD_DisplayStringLine(Line1, (u8 *)lcd_arry);

        sprintf(lcd_arry,"     F = %-6d",PA1_freq);
        LCD_DisplayStringLine(Line3, (u8 *)lcd_arry);

        sprintf(lcd_arry,"     D = %d%%",PA1_duty);
        LCD_DisplayStringLine(Line4, (u8 *)lcd_arry);

        sprintf(lcd_arry,"     A = %3.1f",ADC2_array[0]*3.3/4096);
        LCD_DisplayStringLine(Line5, (u8 *)lcd_arry);
```

```c

        sprintf(lcd_arry,"     T = %4.1f",temp_float);
        LCD_DisplayStringLine(Line6, (u8 *)lcd_arry);
    }
    else if(lcd_view == 1)
    {
        LCD_SetBackColor(Black);
        sprintf(lcd_arry,"        PARA");
        LCD_DisplayStringLine(Line1, (u8 *)lcd_arry);

        //高亮选中的参数，否则保持原本的背景色
        if(PARA_change_state == 0)  LCD_SetBackColor(Red);
        else LCD_SetBackColor(Black);
        sprintf(lcd_arry,"     FH = %-5d",PARA_FH_value);
        LCD_DisplayStringLine(Line3, (u8 *)lcd_arry);

        if(PARA_change_state == 1)  LCD_SetBackColor(Red);
        else LCD_SetBackColor(Black);
        sprintf(lcd_arry,"     AH = %3.1f",PARA_AH_value);
        LCD_DisplayStringLine(Line4, (u8 *)lcd_arry);

        if(PARA_change_state == 2)  LCD_SetBackColor(Red);
        else LCD_SetBackColor(Black);
        sprintf(lcd_arry,"     TH = %-2d",PARA_TH_value);
        LCD_DisplayStringLine(Line5, (u8 *)lcd_arry);

    }
    else if(lcd_view == 2)
    {
        //避免高亮的影响，保持原本的背景色
        LCD_SetBackColor(Black);
        sprintf(lcd_arry,"        RECD");
        LCD_DisplayStringLine(Line1, (u8 *)lcd_arry);

        sprintf(lcd_arry,"     FN = %-2d",RECD_FN_value);
        LCD_DisplayStringLine(Line3, (u8 *)lcd_arry);

        sprintf(lcd_arry,"     AN = %-2d",RECD_AN_value);
        LCD_DisplayStringLine(Line4, (u8 *)lcd_arry);

        sprintf(lcd_arry,"     TN = %-2d",RECD_TN_value);
        LCD_DisplayStringLine(Line5, (u8 *)lcd_arry);
    }
    else if(lcd_view == 3)
    {
        //避免高亮的影响，保持原本的背景色
        LCD_SetBackColor(Black);
        sprintf(lcd_arry,"        FSET");
        LCD_DisplayStringLine(Line1, (u8 *)lcd_arry);

        if(FSET_change_state == 0)  LCD_SetBackColor(Red);
        else LCD_SetBackColor(Black);
        sprintf(lcd_arry,"     FP = %-2d",FSET_FP_value);
        LCD_DisplayStringLine(Line3, (u8 *)lcd_arry);

        if(FSET_change_state == 1)  LCD_SetBackColor(Red);
        else LCD_SetBackColor(Black);
        sprintf(lcd_arry,"     VP = %3.1f",FSET_VP_value);
        LCD_DisplayStringLine(Line4, (u8 *)lcd_arry);

        if(FSET_change_state == 2)  LCD_SetBackColor(Red);
        else LCD_SetBackColor(Black);
        sprintf(lcd_arry,"     TT = %-2d",FSET_TT_value);
        LCD_DisplayStringLine(Line5, (u8 *)lcd_arry);
    }
}
void rx_proc(void)
{
    //判断数据是否接受完毕
    if(rx_pointer!= 0)
    {
        int temp = rx_pointer;
        //接收一次数据需要9个Bit
        HAL_Delay(1);//如果数据没有接受完毕，那么在这1ms内一定会发生中断，rx_pointer一定会变化
        //之所以1ms内一定会发生中断是因为最小的时间是每个字节接收结束到下个字节开始接收的这段时间
        //显然这段时间小于1ms，1ms能处理9bit，间隔时间一定小于9bit，
        if(temp == rx_pointer)
        {
            //串口接收处理部分
            maopao_sort(strlen(rx_arry),rx_arry);
            printf("data_lenth:%d\n",strlen(rx_arry));
//          printf("%s\n",rx_arry);
            led_disp(temp);
            rx_pointer=0;memset(rx_arry,0,50);
        }
    }
}

int fputc(int ch, FILE *f)
{
  /* Your implementation of fputc(). */
    HAL_UART_Transmit(&huart1, (const uint8_t *)&ch,  1, 50);
    return ch;
}


/* USER CODE END 4 */

/**
  * @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 */
  __disable_irq();
  while (1)
  {
  }
  /* 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,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

```