# 实训报告书

实训名称 生产实习实训

院 (部) ： 智能装备学院

专业班级： 电子信息科学与技术 17-2

学生姓名： 张厚今

指导教师： 李涵

完成日期： 2021 年 1 月 8 日

山东科技大学

2021 年 1 月

| | |
|---|---|
| 实训课题 | 智能环境检测项目实训 |
| 实训人姓名 | 张厚今 |
| 实训日期 | **2020** 年 **12** 月 **21** 日至 **2021** 年 **1** 月 **8** 日 |
| 实训成绩 | |
| 指导<br>教师<br>评语 | 指导教师签名：_____<br>_____年_____月____日 |

# 目录



装

订

# 1 实训目的及内容

## 1.1 实训目的

该实训是在电子信息科学与技术专业课程结束后开设的实践性教学环节，通过完成具有一定工作量和难度的设计型综合实训课题，培养学生综合运用所学专业课程的基本知识、基本理论和掌握的基本技能，解决电子信息工程领域内有关智能信息处理或嵌入式与物联网工程问题的设计和实现能力，强化学生实践动手能力和工程实施的能力。使学生进一步巩固和综合运用所学专业知识的基础上，通过现状调研、文献检索、工程分析和设计等方法和手段，增强学生分析、解决实际问题的能力，尝试所学的内容解决实际工程问题，培养学生的工程实践能力。

## 1.2 实训内容

本次实训需要实现基于 STM32 单片机的火焰和电压检测系统。实训中主要涉及的内容包括 ADC 采集及数据转换、GPIO 控制、串口通信等。其中最主要的部分是 ADC 数模转换，需要用轮询方式分别读取电压及火焰传感器的数据，将电压数据转换为电压值，将火焰传感器的数据分成不同的等级，根据不同的等级控制对应的 LED。最后要将电压和火焰传感器的数值通过串口进行显示。

# 2 实训原理

## 2.1 STM32 主控芯片

本次实训使用的主控芯片为 STM32F051K8 芯片，STM32F051K8 系列集成了工作频率为 48 MHz 的高性能 ARM Cortex-M0 32 位 RISC 内核、高速嵌入式存储器以及各种增强型外设。所有器件都提供标准通信接口，提供 1 个 12 位的 ADC、1 个 12 位 DAC 以及最多 5 个通用 16 位定时器、1 个 32 位定时器和 1 个高级控制 PWM 定时器。STM32F051K8 系列可在-40℃至+85℃和-40℃至+105℃两种温度范围内工作，电源为 2.0V 至 3.6 V。本次实训中，主要用到了该芯片提供的 GPIO、UART、ADC 等功能。

## 2.2 GPIO 通用输入输出

GPIO 是通用输入输出端口的简称，STM32 单片机可以控制引脚的输入输出以实现丰富的功能。GPIO 的引脚与外部硬件设备连接，可实现与外部通讯、控制外部硬件或采集外部硬件数据等功能。

本次实验中，使用 GPIO 控制了三个 LED 的亮灭，通过调用 HAL 库函数的

方式，控制对应引脚的 GPIO 输出高低电平，从而实现 LED 的亮灭控制。

### 2.3 UART 串口通信

UART 是通用一部收发传输器的简称，用于在不同设备之间，将数据通过串行通讯和并行通讯进行数据传输和转换。串行通信的通信线路简单，只要一对传输线就可以实现双向通信，从而大大降低了成本，特别适用于远距离通信。本次实训中使用的全双工异步串行通信的数据格式包括：1 位的起始位，8 位数据位，1 位的奇偶校验位以及 1 位的停止位。

本次实训中，使用 UART 串口进行数据传输。将 ADC 采集到的电压数值、火焰传感器强度等信息，通过串口通信传输到电脑的串口调试界面，以便查看数据和调试程序。

### 2.4 ADC 模数转换

ADC 是模拟/数字转换的简称，用于将模拟信号转换为数字信号。将模拟信号转换为数字信号，一般分为 4 个步骤进行，即采样、保持、量化和编码。前两个步骤在采样保持电路中完成，后两步骤则在 ADC 中完成。ADC 是把经过与标准量比较处理后的模拟量转换成以二进制数值表示的离散信号的转换器，因此任何一个模数转换器都需要一个参考模拟量作为转换的标准。通常参考标准为最大的可转换信号大小，而输出的数字量则表示输入信号相对于参考信号的大小。

本次实验中，使用 ADC 将采集到的火焰强度、电压数值等模拟信号转换为数字信号，以便进行相应的处理。

## 3 实训过程

### 3.1 开发环境搭建

软件层面，本次实训使用 STM32CubeMX 加 Keil-MDK 的方式进行软件开发。STM32CubeMX 是由意法半导体公司开发的一款 STM32 图形化配置工具，它将很多函数都进行了封装，提供了硬件抽象层、中间层、示例代码等内容。其提供的 HAL 库极大地方便了 STM32 的开发过程，可以大大减轻开发工作。

硬件层面，使用的开发板是由华清远见公司提供的 NB-IoT 物联网开发板，利用 ST-Link 进行程序烧录，使用 CH340 芯片加 MiniUSB 的方式进行串口通信。NB-IoT 开发板搭载了 STM32F051K8 芯片，提供了 LCD 液晶屏、LED 灯、五向按键、电量检测接口、JLink 调试接口、Lierda 无线通讯模块、SD 卡槽等外设，

资源丰富，非常适合 NB-IoT 开发。开发板实物如下图 3.1 所示。

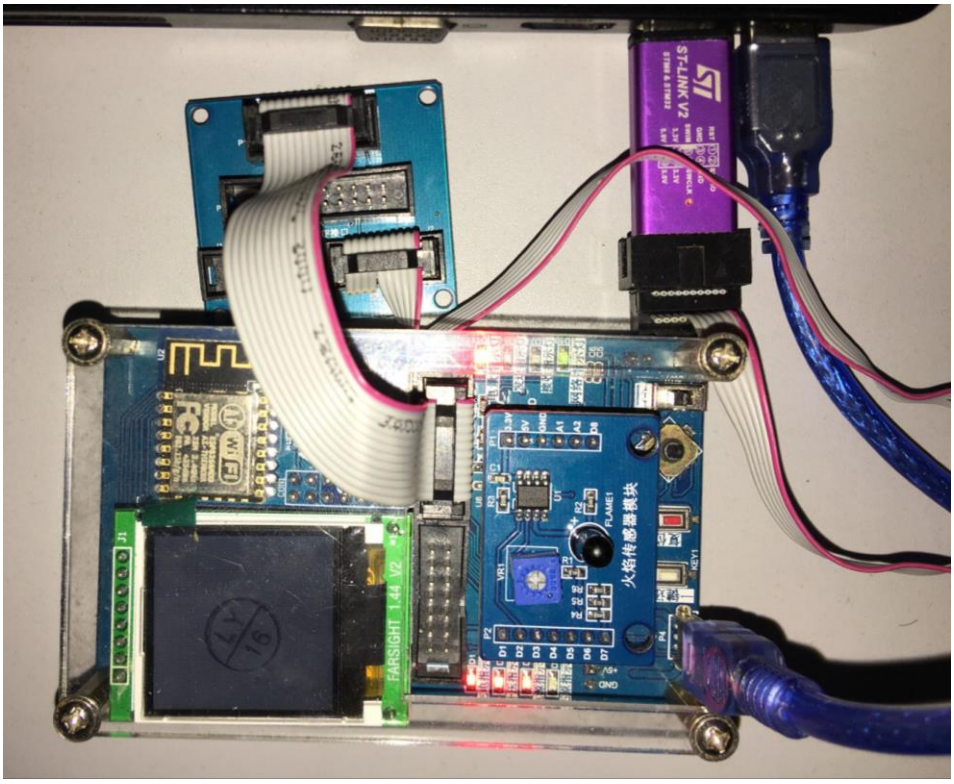


图 3.1 开发板实物图

## 3.2 原理图分析

### 3.2.1 LED 灯控制

实训使用的开发板中，搭载了三个用于显示状态的 LED 灯。根据底板原理图可以看出，三个 LED 灯分别为 LED2、LED3 和 LED4。根据下图 3.2 的原理图，三个 LED 共阳极，因此只要为其阴极提供高低电平即可控制 LED 的亮灭。

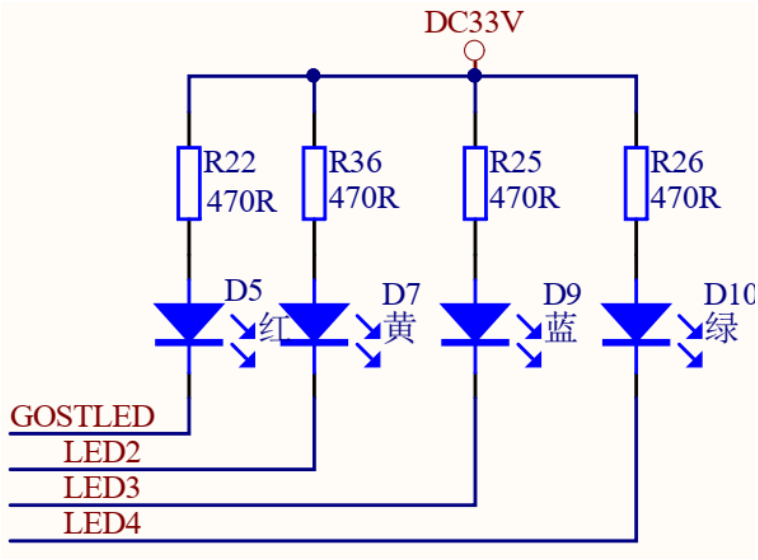


图 3.2 底板原理图 LED 部分

再根据核心板原理图，如下图 3.3 所示，可以看到 LED 的三个引脚是直接连接在了 STM32 芯片的 PB0~PB2 这三个引脚上。因此，只要控制 STM32 这三个引脚的 GPIO 电平高低，就可以控制三个 LED 的亮灭。

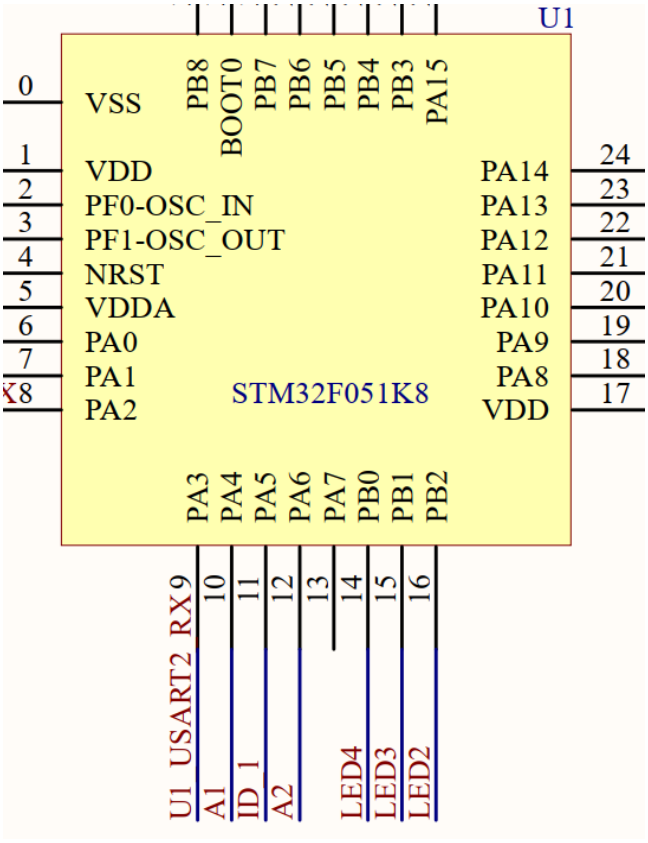


图 3.3 主控芯片 LED 部分

3.2.2 火焰传感器检测

火焰检测功能使用了独立的红外火焰检测模块，远红外火焰传感器可以用来探测火源或其它热源。在火焰传感器模块的原理图上，可以看到火焰传感器 FLAME 连接的引脚为 A1 引脚，如下图 3.4 所示。在 STM32 芯片的原理图上可以看到，A1 对应的引脚为 PA4 引脚。因此，只需要将 PA4 引脚设置为 ADC 模式，读取对应引脚的数值并进行转换，即可获得火焰程度数值。

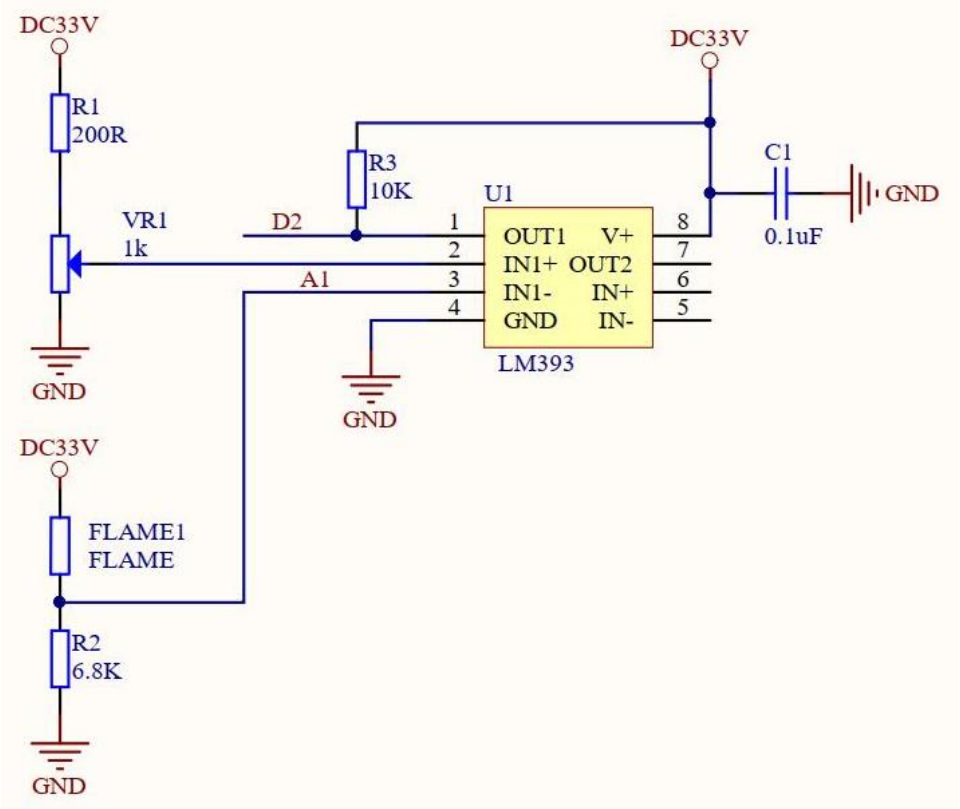


图 3.4 火焰传感器原理图

3.2.3 电池电量检测

本次实训使用的开发板可以通过电池和USB两种方式供电，通过ADC数模转换，可以测得实时的电池电量数值。由于 ADC 模拟输入的参考电压最大为3.3V，而电池电压最大值为 5V，所以不能直接将电池连接至 ADC 输入端口。本实训中使用的开发板，利用 R15 和 R13 电阻的分压，将检测电压分压至电池电压最大值的三分之二，即利用 BAT_ADC 引脚检测电压值，如下图 3.5 所示。

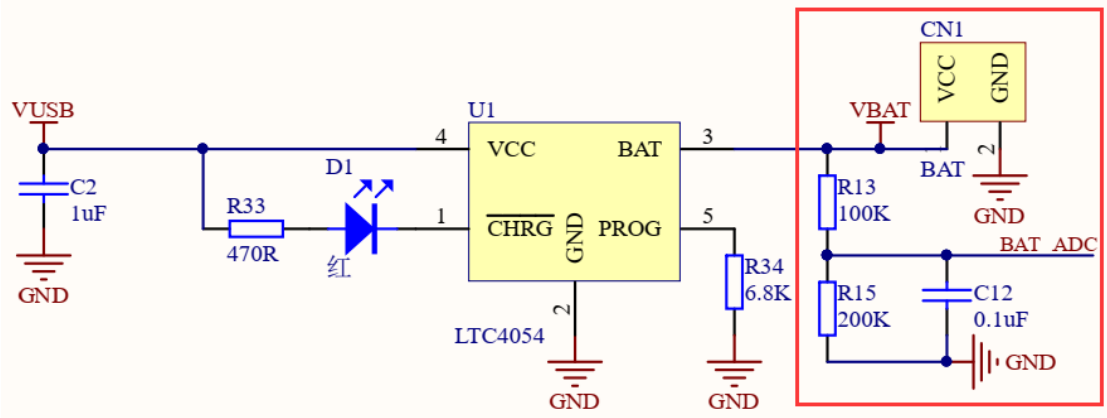


图 3.5 电压检测模块

3.3 程序编写

3.3.1 生成程序模板

首先在 STM32CubeMX 软件中生成程序模板。打开该软件，新建项目并选择相应的芯片型号。在图形编辑页面将 PB0~PB2 引脚设置为 GPIO 输出模式，用来控制 LED 灯。设置 PA0 和 PA4 分别为 ADC 检测引脚，分别表示电压检测和火焰强度检测。最后设置 PA10 和 PA9 为 UART 模式，以实现串口通信功能。设置完成的引脚界面如下图 3.6 所示。

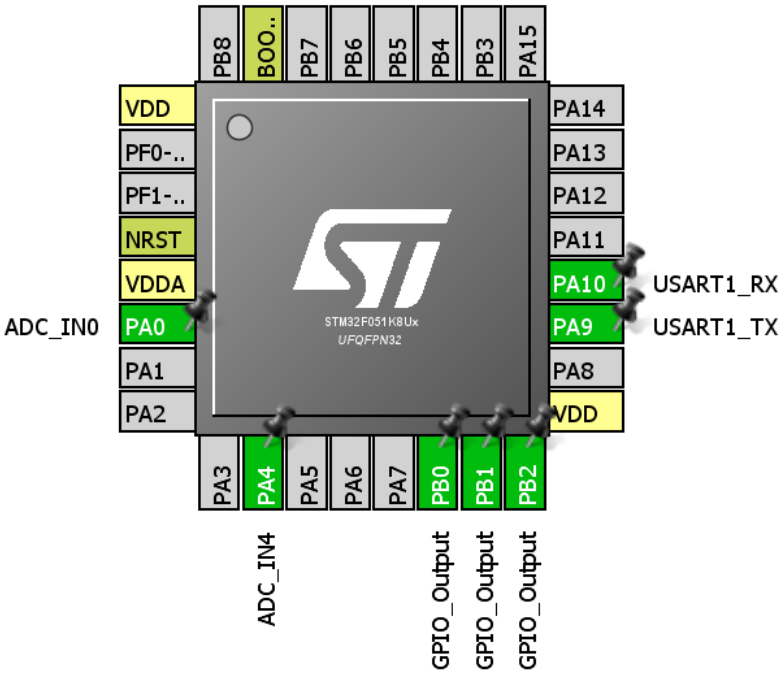


图 3.6 设置引脚状态

接下来在配置界面设置 UART 的波特率、数据位、校验、停止位等内容，如下图 3.7 所示。

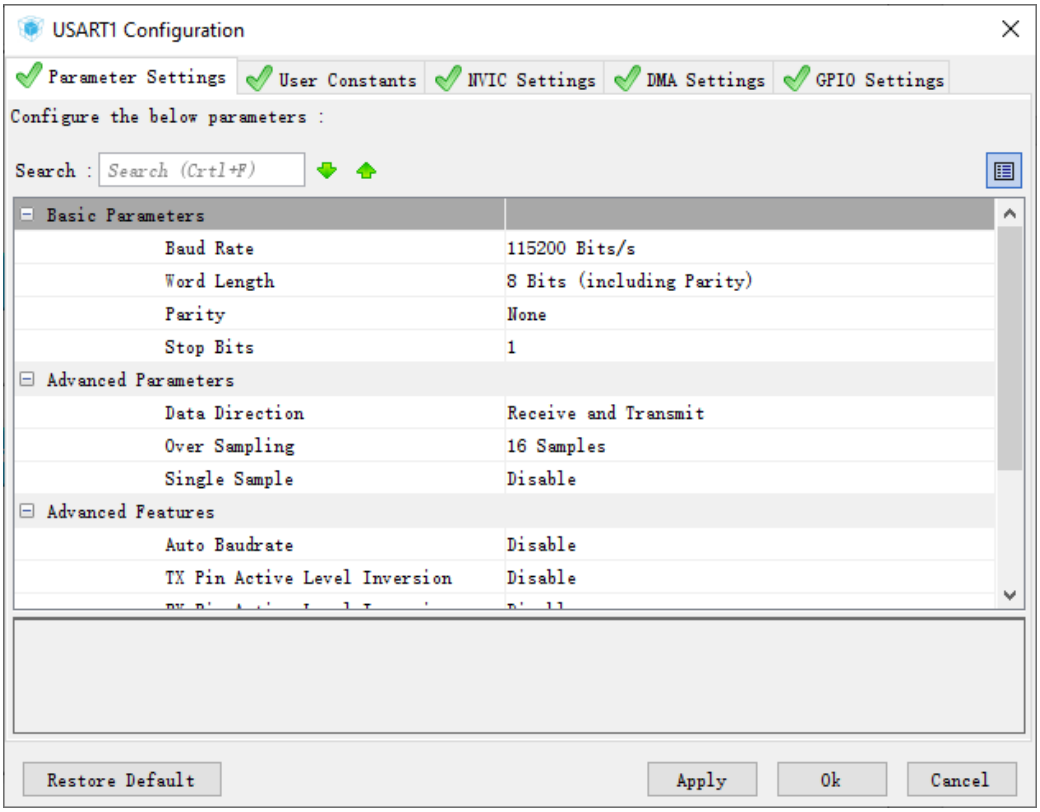


图 3.7 配置 UART

配置 ADC 时钟模式、转换精度、数据转换方式等信息，如下图 3.8 所示。

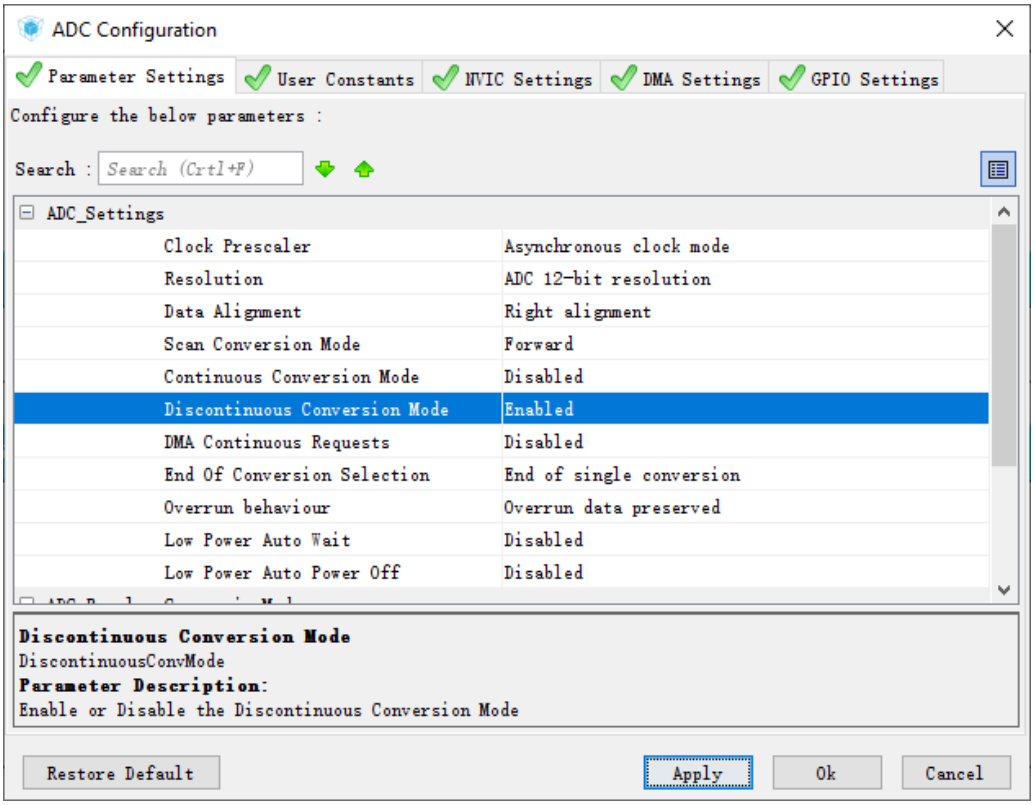


图 3.8 配置 ADC

经过配置之后，接下来需要对工程的名称、生成路径等代码生成过程进行配置，如下图 3.9 所示。

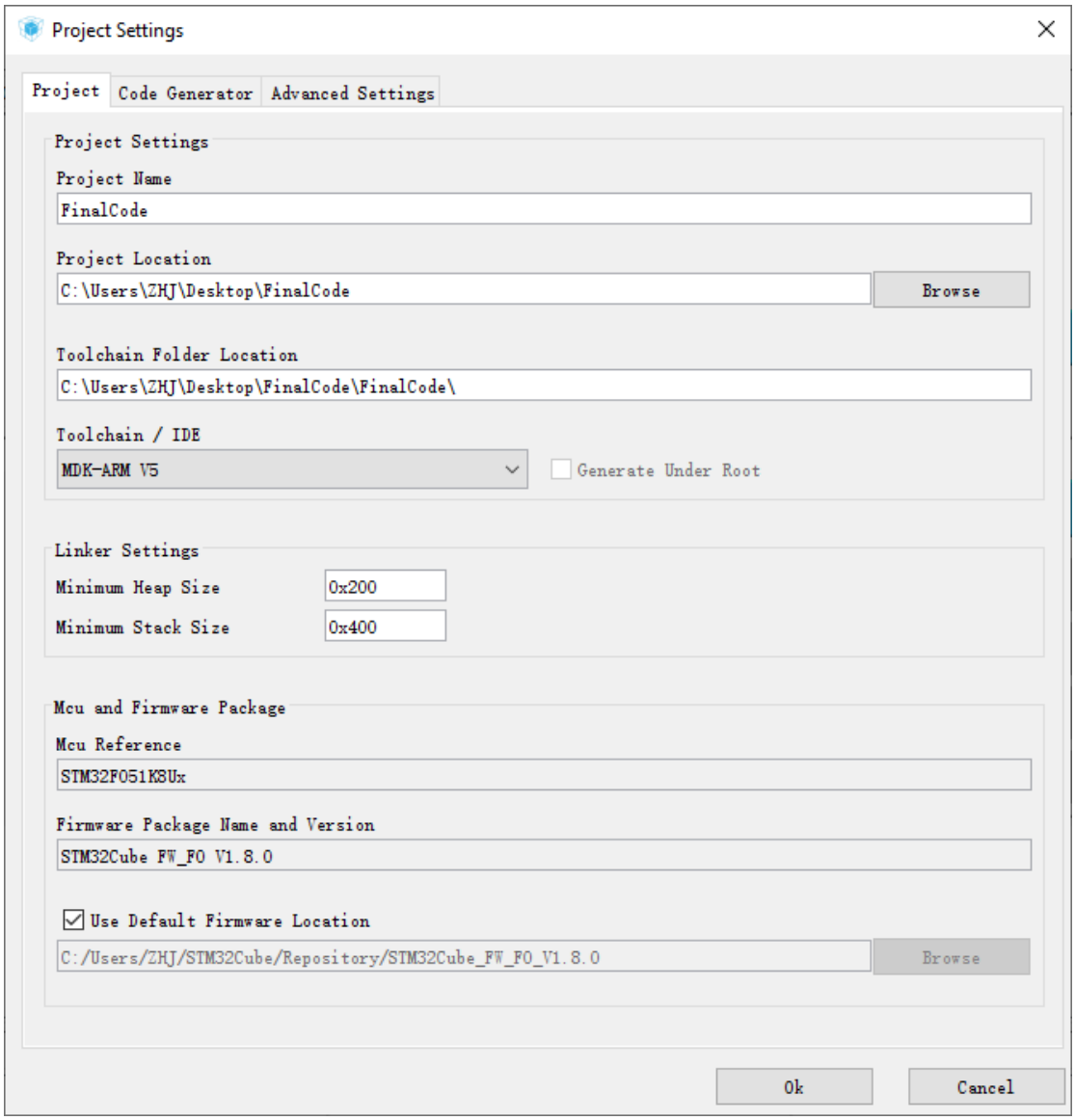


图 3.9 工程配置

配置完成后，就可以自动生成基本的工程模板了。

### 3.3.2 编写程序代码

首先引用头文件和宏定义，添加 string.h 文件用来在后面调用 strcpy 函数，添加三个火焰程度的字符串宏定义以及三个 LED 引脚的宏定义。

```
/* USER CODE BEGIN Includes */
#include "string.h"
#define LOW         "Safe"
#define MID         "Fine"
#define HIGH        "Danger!!!!"
#define GREEN       GPIO_PIN_0
#define BLUE        GPIO_PIN_1
#define YELLOW      GPIO_PIN_2
/* USER CODE END Includes */
```

重新编写 fputc 函数，用于将 printf 函数的输出重定向到串口输出。

```
/* 重定向 UART 的发送到标准输出 */
int fputc(int ch, FILE * file){
  while(__HAL_UART_GET_FLAG(&huart1,UART_FLAG_TXE) == RESET);
  huart1.Instance->TDR = ch;
  return ch;
}
```

编写三个函数，分别用来控制 LED 的全灭、单个 LED 亮和单个 LED 灭。控制 LED 亮灭使用了 HAL 库函数 HAL_GPIO_WritePin，该函数可以控制相应的 GPIO 端口为指定的高低电平。

```
/* 处理 LED 亮灭的函数 */
void LED_All_Off(void){
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_SET);
}
void LED_On(int Color){
    HAL_GPIO_WritePin(GPIOB, Color, GPIO_PIN_RESET);
}
void LED_Off(int Color){
    HAL_GPIO_WritePin(GPIOB, Color, GPIO_PIN_SET);
}
```

接下来编写主函数，首先添加变量定义，包括用于存储 ADC 轮询值的变量 value、存储转换后的电压值的变量 Battery、表示火焰程度的字符数组 Level、存储火焰传感器数值的变量 Fire。

```
/* USER CODE BEGIN 1 */
double value[2];        // 存储 ADC 检测到的数值
double Battery;         // 电池电压
char Level[10];         // 火焰程度
int Fire;               // 火焰传感器数值
/* USER CODE END 1 */
```

在主函数的 while 循环中需要循序执行三个部分，分别是 ADC 轮询检测、电压和火焰传感器数值的转化、以及根据火焰传感器强度分段处理 LED 和串口通信。ADC 采集转换过程，需要先调用 HAL_ADC_Start 函数开启 ADC 转换，

之后调用 HAL_ADC_PollForConversion 函数等待 ADC 转换完成，转换完成后，调用 HAL_ADC_GetValue 函数获取转换后的数值，最后调用 HAL_ADC_Stop 函数停止本轮 ADC 检测。由于本次实训要求检测两个 ADC 值，所以需要对两个 ADC 进行轮询检测。需要将前三个 ADC 库函数放到 for 循环中，循环两次之后再调用 HAL_ADC_Stop 函数结束本轮转换。以此来实现对两个 ADC 数据的轮询检测。

```
/* ADC 轮询检测 */
for(int i=0;i<2;i++){
   HAL_ADC_Start(&hadc);
   HAL_ADC_PollForConversion(&hadc,100);
   value[i] = HAL_ADC_GetValue(&hadc);
   HAL_Delay(200);
}
HAL_ADC_Stop(&hadc);
```

由于实验要求显示当前的电压值，所以需要将 ADC 检测到的数值进行相应的转换。实验中所配置的 ADC 转换精度为 12 位精度，因此装载满时的数值为 4095，即 ADC 可表示的数据范围为 0~4095。因此，只要将采集到的数值除以 4096，再乘以标准电压 3.3V，即可得到实际检测的 BAT_ADC 端口的电压值。另外，因为火焰强度检测到的数据没有实际意义，因此不必进行数据转换，直接通过串口输出即可。

```
/* 电压和火焰传感器的数值转换 */
Battery = value[0]*3.3/4096.0;
Fire = (int)value[1];
printf("BAT = %.2lf V\t", Battery);
printf("Fire = %d\n", Fire);
```

最后，根据不同的火焰强度数值，将所有火焰程度分成三个等级，每个等级再分别进行处理即可。其处理主要为：为表示火焰程度的字符数组 Level 赋值以及控制 LED 亮灭两部分。

```
/* 根据火焰传感器强度分别处理 */
if(Fire<1000){
    strcpy(Level, LOW);
    LED_On(GREEN);
    LED_Off(YELLOW);
    LED_Off(BLUE);
}
else if(Fire>1000 && Fire<1800){
    strcpy(Level, MID);
    LED_On(YELLOW);
    LED_Off(GREEN);
    LED_Off(BLUE);
}
else{
    strcpy(Level, HIGH);
    LED_On(BLUE);
    LED_Off(YELLOW);
    LED_Off(GREEN);
}
printf("Level = %s\n\n", Level);
```

代码编写完成后，编译程序，无错误警告即可。

# 4 实训结果及分析

代码编写完成后，通过 ST-Link 将代码烧录至开发板中。通过 MiniUSB 连接电脑，利用串口助手查看开发板输出的信息。

## 4.1 安全模式

当火焰传感器检测到的强度较小时，可看作火焰为正常水平，即没有火焰。此时 LED 绿灯常亮（如图 4.1 所示），串口会显示显示此时的火焰强度、电压值以及当前的火焰程度等级（如图 4.2 所示）。

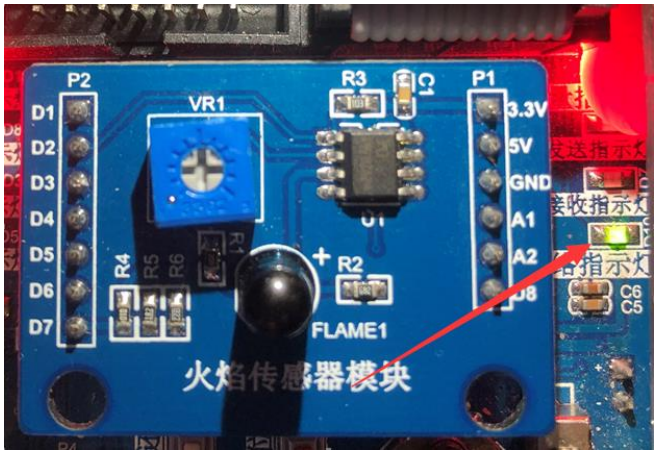


图 4.1 LED 状态

接收区：共接收561709字节，速度82字节/秒。共发送0字节。

```
BAT = 2.79 V     Fire = 496
Level = Safe

BAT = 2.78 V     Fire = 538
Level = Safe

BAT = 2.78 V     Fire = 539
Level = Safe

BAT = 2.79 V     Fire = 538
Level = Safe

BAT = 2.78 V     Fire = 537
Level = Safe

BAT = 2.78 V     Fire = 537
Level = Safe
```

图 4.2 串口通信状态

4.2 正常模式

当火焰传感器的强度大于 1000 且小于 1800 时，可看作此时光线强度较强而导致的火焰传感器数据较高，LED 黄灯常量（如图 4.3 所示）。此时火焰强度等级为 Fine 等级，串口同样会发送火焰强度、电压值以及当前的火焰程度等级信息（如图 4.4 所示）。

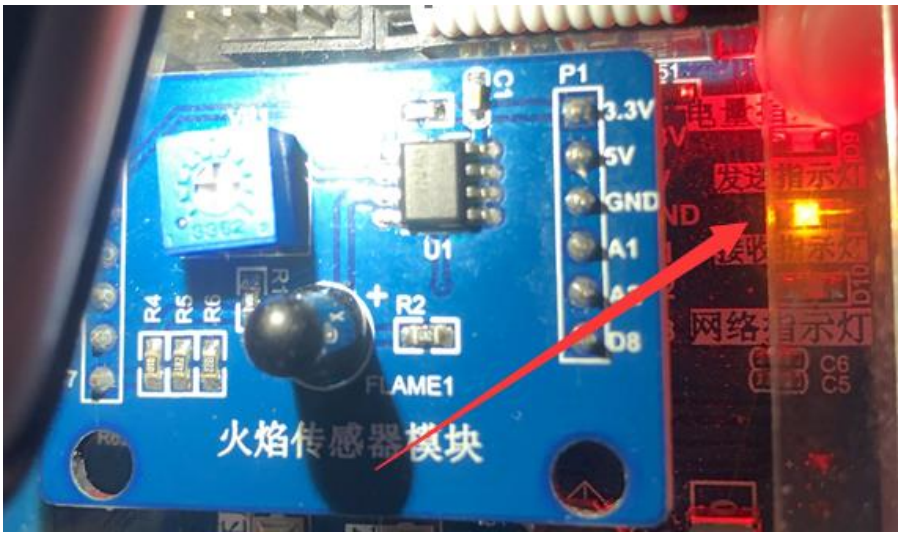


图 4.3 LED 状态

接收区：共接收563928字节，速度84字节/秒。共发送0字节。

```
BAT = 2.79 V     Fire = 1003
Level = Fine

BAT = 2.78 V     Fire = 1384
Level = Fine

BAT = 2.78 V     Fire = 1314
Level = Fine

BAT = 2.78 V     Fire = 1271
Level = Fine

BAT = 2.78 V     Fire = 1187
Level = Fine

BAT = 2.78 V     Fire = 1162
Level = Fine
```

图 4.4 串口通信状态

4.3 危险模式

当火焰强度大于 1800 时，可看作此时有较大可能产生了火焰，此时 LED 蓝灯常亮，串口发送相关的信息。效果如图 4.5 和图 4.6 所示。


图 4.5 LED 状态

接收区：共接收566731字节，速度144字节/秒。共发送0字节。

```
BAT = 2.79 V     Fire = 2170
Level = Danger!!!!

BAT = 2.78 V     Fire = 2486
Level = Danger!!!!

BAT = 2.79 V     Fire = 2345
Level = Danger!!!!

BAT = 2.78 V     Fire = 2222
Level = Danger!!!!

BAT = 2.79 V     Fire = 2113
Level = Danger!!!!

BAT = 2.78 V     Fire = 2095
Level = Danger!!!!
```

图 4.6 串口通信状态

# 5 实训心得

为期两个周的生产实习实训在老师的指导下即将结束，我们也顺利的完成了智能环境监测项目。通过这次实训，我们收获了很多，一方面接触并学习了以前没学过的专业知识并对学习到的知识进行应用，另一方面提高了自己动手做项目的能力。

此次实训，在老师的带领下对 Linux C 的基本操作进行了复习和巩固，了解了以前没有接触过的 STM32CubeMX 软件的安装和使用，学习了 STM32F051K8 芯片相关引脚的功能和控制，以及通过完成 LED 灯点亮、LED 呼吸灯、火焰模块和电池电量数据的串口通信等相关小程序来进一步加深理解其相关功能控制并提高自己的动手能力。本次实训实现了基于 STM32 单片机的火焰和电压检测系统。实训中主要涉及的内容包括 ADC 采集及数据转换、GPIO 控制、串口通信等。其中最主要的部分是 ADC 数模转换，需要用轮询方式分别读取电压及火焰传感器的数据，将电压数据转换为电压值，将火焰传感器的数据分成不同的等级，根据不同的等级控制对应的 LED。最后要将电压和火焰传感器的数值通过串口进行显示。整个实训过程较为顺利，本组同学积极配合，在遇到不会的问题时，都积极讨论，最终解决了问题，实训取得了理想的成绩。

最后非常感谢各位老师，在实训过程中，认真仔细地讲解专业知识，对于同学们的问题耐心解答，注重实际操作，提高了我们的动手能力，提高了我们的专业素养。相信本次实训定会对我们未来项目实践起到积极作用，在今后的工作和生活中，我将继续学习，深入实践，不断提升自我，继续创造更多价值。

## 附录 1：参考文献

[1]杨秀芝,汪晴晴.基于 STM32 程控电源的设计与实现[J].三门峡职业技术学院学报,2020,19(04):125-131.

[2]刘辉应. STM32 初学应用笔记实例六则(二)[N]. 电子报,2020-12-06(007).

[3] 文涛. stm32f103vet6 串口 2 使用设置问题探讨[N]. 电子报,2020-11-22(007).

[4] 齐翔. STM32F103 AD 采样波动幅度大的问题[N]. 电子报,2020-10-18(007).

[5]王沛雪,张水利,崔佳萌,任卓,李江,闫子瑜,张波.基于 STM32F103 单片机的远程无线瓦斯浓度监测系统[J].电脑与电信,2020(10):53-56.

[6]崔海朋.基于 STM32 和 AD5791 的高精度数模转换电路设计[J].电子产品世界,2020,27(06):39-42.

[7]张海超,张北伟.基于 STM32 的多串口通信系统设计[J].国外电子测量技术,2019,38(02):99-102.

[8]刘宁,陈冬琼,杨克磊.基于STM32最小系统串口通信显示系统设计[J].工业控制计算机,2017,30(08):33-34+36.

[9]石发强.基于紫红外线检测原理的火焰传感器的设计[J].煤炭与化工,2015,38(09):101-104+107.

[10]刘磊,孙晓菲,唐含,范超,盛婕.电气实验室安全预警系统设计[J].科技创新与应用,2015(20):28.

## 附录 2：主要代码及注释

（1）项目仓库

GitHub 地址：https://github.com/ZHJ0125/STM32F051K8

Gitee 地址：https://gitee.com/zhj0125/STM32F051K8

（2）主要代码

```
/* USER CODE BEGIN Includes */
#include "string.h"
#define LOW         "Safe"
#define MID         "Fine"
#define HIGH        "Danger!!!!"
#define GREEN       GPIO_PIN_0
#define BLUE        GPIO_PIN_1
#define YELLOW  GPIO_PIN_2
/* USER CODE END Includes */
/* 重定向 UART 的发送到标准输出 */
int fputc(int ch, FILE * file){
  while(__HAL_UART_GET_FLAG(&huart1,UART_FLAG_TXE) == RESET);
  huart1.Instance->TDR = ch;
  return ch;
}
/* 处理 LED 亮灭的函数 */
void LED_All_Off(void){
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_SET);
}
void LED_On(int Color){
    HAL_GPIO_WritePin(GPIOB, Color, GPIO_PIN_RESET);
}
void LED_Off(int Color){
    HAL_GPIO_WritePin(GPIOB, Color, GPIO_PIN_SET);
}
/* USER CODE BEGIN 1 */
  double value[2];          // 存储 ADC 检测到的数值
  double Battery;               // 电池电压
  char Level[10];               // 火焰程度
  int Fire;                 // 火焰传感器数值
/* USER CODE END 1 */

/* USER CODE BEGIN WHILE */

```

```
while (1){
    /* ADC 轮询检测 */
    for(int i=0;i<2;i++){
      HAL_ADC_Start(&hadc);
      HAL_ADC_PollForConversion(&hadc,100);
      value[i] = HAL_ADC_GetValue(&hadc);
      HAL_Delay(200);
    }
    HAL_ADC_Stop(&hadc);

    /* 电压和火焰传感器的数值转换 */
    Battery = value[0]*3.3/4096.0;
    Fire = (int)value[1];
    printf("BAT = %.2lf V\t   ", Battery);
    printf("Fire = %d\n", Fire);

    /* 根据火焰传感器强度分别处理 */
    if(Fire<1000){
      strcpy(Level, LOW);
      LED_On(GREEN);
      LED_Off(YELLOW);
      LED_Off(BLUE);
    }
    else if(Fire>1000 && Fire<1800){
      strcpy(Level, MID);
      LED_On(YELLOW);
      LED_Off(GREEN);
      LED_Off(BLUE);
    }
    else{
      strcpy(Level, HIGH);
      LED_On(BLUE);
      LED_Off(YELLOW);
      LED_Off(GREEN);
    }
    printf("Level = %s\n\n", Level);
}
/* USER CODE END WHILE */

```