
南京理工大学

NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

# 计算机科学与工程学院

## “嵌入式系统”实验报告书

**题目：ex5_922106840127_ I2C**

**学号：922106840127**

**姓名：刘宇翔**

**成绩**

**日期：　　2025 年　　4 月　　8 日**

# 1 题目要求

### 1. 题目设计要求

（1）实验内容

通过 I2C 总线读写 AP3216C 传感器采集的环境光数据和距离数据，并将它们显示在 LCD 上，数据保留一位小数。

要求：采用 I2C2 的 HAL 库驱动函数(不是 GPIO 口模拟 I2C 总 线)进行操作 AP3216C；

（2）完成要求

工程名称命名：ex5_学号_I2C，并打包成：ex5_学号_ I2C.rar 压缩文件夹

实验报告 PDF 格式：ex5_学号_ I2C.pdf

### 2. 拟实现的具体功能

本次实验拟实现一个基于 STM32CubeMX 硬件设计平台的 I2C 传感器数据采集系统，通过 I2C2 总线读写 AP3216C 传感器采集环境光与距离数据，并实时将采集结果显示在 LCD 屏上，数据显示精确到一位小数。系统采用 I2C2 的 HAL 库驱动函数进行 AP3216C 的操作，确保数据采集的准确性和实时性。

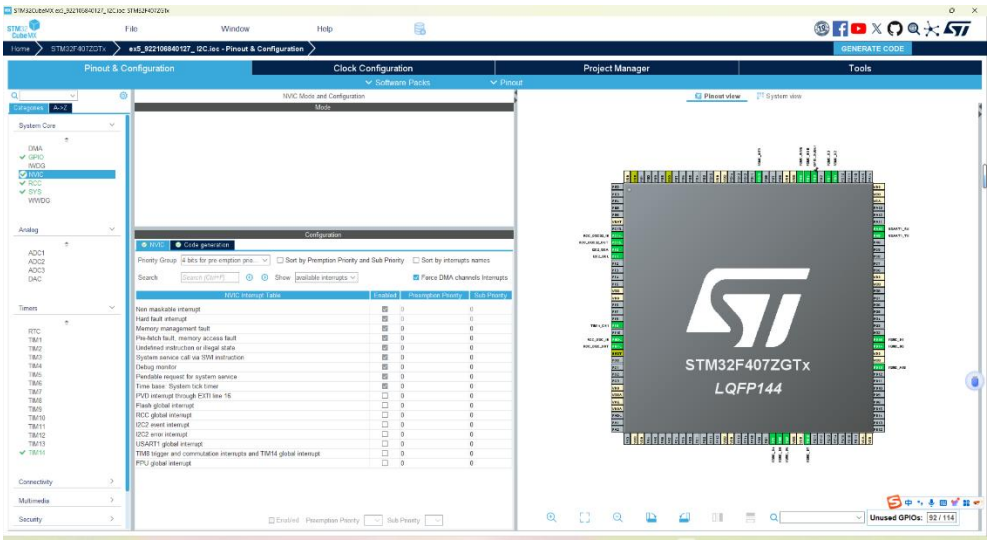
实验中，首先完成各项硬件初始化和外设配置，包括 I2C2 接口、LCD 显示模块及其它必要的资源；接着通过 AP3216C 的寄存器操作完成传感器初始化，并周期性读取环境光数据及距离数据；最后，将读取到的数据经过格式化处理后显示在 LCD 上，同时通过串口输出调试信息，为后续数据交互和功能扩展打下基础。

# 2 总体设计

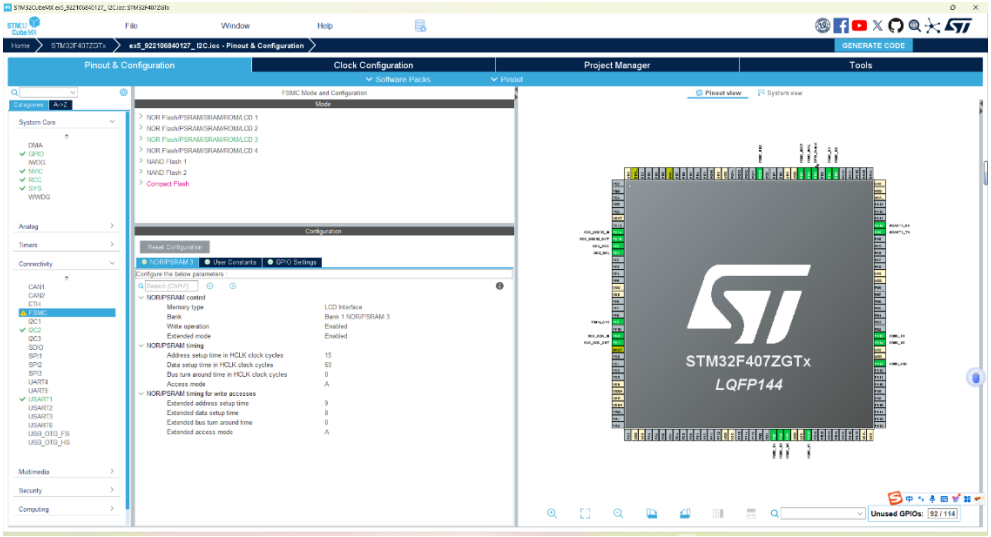
## 2.1 硬件设计

### 1.硬件设计思路

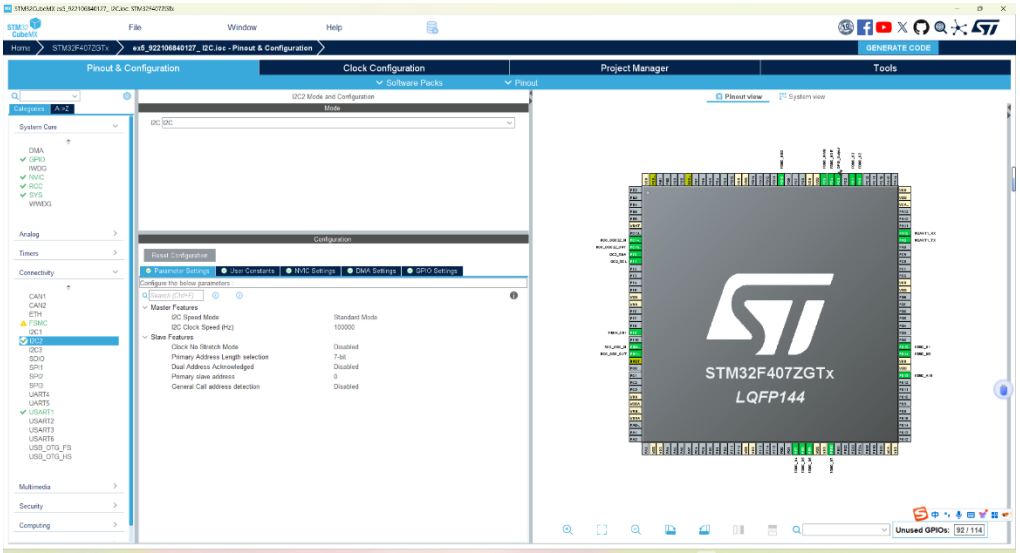
我通过查询相关开发板原理图确定了本次实验所需要的有关 GPIO 端口、NVIC 设置、USART 通信端口与 TIM 相关的设置，具体的代码工程配置硬件设计如下：



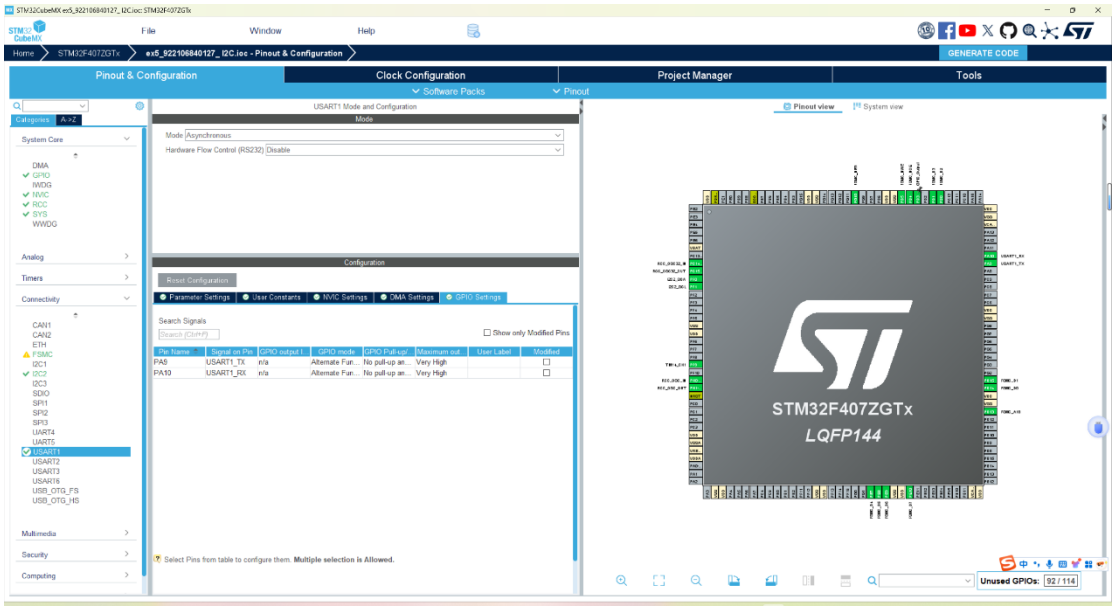

NVIC 设置




FSMC 设置




I2C 设置




USART 设置

如图所示，我在 NVIC 的相关设置界面对几种中断的相关属性做了具体的规定，USART 的通信模式采用的是异步通信，FSMC 是设置 LCD 相关的内容，I2C 的配置是本次实验的重点内容。

其他设置与前几次实验相同，如 RCC 采用 Crystal/Ceramic Resonator，使用 Serial Wire 用于调试对应的 debug 接口，对 Clock Configuration 的时钟配置进行规范，选择了 MDK-ARM V5.32 作为编译工具链，其他内容此处省略。 以上配置图是我作为 STM32CUBEMX 进行的配置设置，设置后点击 “Generate Code” 生成初始化代码。

## 2.2 软件设计

### 1.软件设计概述

本软件主要实现基于 I2C 总线的 AP3216C 传感器数据采集与 LCD 显示功能，通过 STM32 单片机利用 I2C2 接口读写 AP3216C 传感器，采集环境光数据和距离数据，并将经过格式化处理后的结果实时显示在 LCD 屏幕上，数据精度严格保留一位小数。系统基于 HAL 库进行外设初始化，采用模块化设计方式，整体结构清晰、易于维护且具备良好的实时性与扩展性。具体设计内容如下：

**（1） 系统初始化模块**

在 main 函数中，系统首先调用 HAL_Init()完成底层硬件初始化，并通过

SystemClock_Config()配置系统时钟，确保各外设的工作时序正确；随后依次初始化 GPIO、FSMC、I2C2、TIM14 以及 USART1，并调用 drv_lcd_init()初始化 LCD 显示模块，通过 lcd_clear()设置初始显示状态。

**（2） AP3216C 传感器操作模块**

利用 I2C2 接口，通过 HAL_I2C_Mem_Read/Write 函数实现对 AP3216C 传感器各寄存器的操作，在 AP3216C_Init()函数中完成传感器的复位和使能设置。通过 AP3216C_Read_PS()与 AP3216C_Read_ALS()分别读取距离数据和环境光数据，同时根据传感器的量程设置进行单位换算，确保采集数据的正确性。

**（3） 数据显示与格式化模块**

在主循环中，系统周期性调用 Display_Sensor_Data()函数，将采集到的距离数据以及经过 RoundToOneDecimal()函数处理后的环境光数据进行格式化，生成严格保留一位小数的字符串，再调用 lcd_show_string()将数据显示在 LCD 上；同时通过 printf()输出调试信息，便于实时监控数据变化。

**（4） 辅助数据格式化模块**

为保证 LCD 上显示的环境光数据始终严格保留一位小数，系统新增 RoundToOneDecimal()函数，对原始浮点数据进行四舍五入处理，使所有显示数据的精度符合要求，避免因数值变化导致多余小数位或显示异常。

总体而言，本软件依托 I2C 总线及 HAL 库驱动 AP3216C 传感器，实现了环境光与距离数据的实时采集和精确显示，模块划分合理、逻辑清晰，为后续进一步扩展如数据记录、报警控制等功能奠定了坚实基础。

**2.软件流程分解**

A. 初始阶段

开始 → 系统初始化

程序启动后，首先完成各项硬件与外设的初始化配置，包括：

- 系统底层初始化（调用 HAL_Init() 对 MCU 进行复位与基础设置）
- 系统时钟配置（通过 SystemClock_Config() 配置主频与外设时钟，确保各模块正常运行）
- 初始化 GPIO、FSMC、I2C2、TIM14 以及 USART1（分别调用 MX_GPIO_Init()、MX_FSMC_Init() 、 MX_I2C2_Init() 、 MX_TIM14_Init() 与

MX_USART1_UART_Init()，为传感器通信、LCD 显示及调试信息输出提供硬件支持）

• 初始化 LCD（调用 drv_lcd_init() 配置液晶模块，并通过 lcd_clear() 清屏设置初始显示状态）

• 初始化 AP3216C 传感器（调用 AP3216C_Init() 进行复位及使能设置，保证传感器进入正常工作状态）

B. 主循环结构

初始化完成 → 进入主循环

系统进入主循环后，周期性地读取 AP3216C 传感器采集的环境光和距离数据，并调用显示函数刷新 LCD 屏幕，同时通过串口输出调试信息。

C. I2C 数据采集流程

传感器数据读取 → 通过 I2C2 接口完成数据采集

利用 HAL 库提供的 I2C2 驱动函数，在 AP3216C_Read_PS() 和 AP3216C_Read_ALS()函数中，依次从 AP3216C 的各数据寄存器读取两字节数据，再进行数据解码和单位换算，得到实际的距离值与环境光亮度。

D. 数据处理与格式化流程

读取到的环境光数据经过传感器量程选择后换算出实际亮度值后，再调用辅助函数（如 RoundToOneDecimal()）对数据进行四舍五入处理，确保输出结果严格保留一位小数，从而使数据格式统一、显示规范。

E. LCD 显示模块

周期性刷新 LCD → 调用 Display_Sensor_Data()

主循环中每隔一定时间（如 2000ms）调用 Display_Sensor_Data() 函数，将处理后的距离数据和环境光数据格式化为字符串，使用 lcd_show_string() 显示在 LCD 屏的指定区域，使用户直观查看传感器采集结果。

F. 循环机制

数据采集及显示完成 → 返回主循环

每次读取与显示数据完成后，系统返回主循环，等待下一次数据采集与刷新，保证整个系统运行连续、稳定，实现实时监控 AP3216C 传感器采集的环境光和距离数据，并确保数据输出始终保留一位小数。

**3. μ visoin 详细代码**

```
#include "main.h"
#include "i2c.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"
#include "fsmc.h"
#include "stdio.h"
#include "./BSP/LCD/drv_lcd.h"
#include "./BSP/LCD/rttlogo.h"

#define AP3216C_CONFIG_REG          0x00
#define AP3216C_IR_DATA_L           0x0A
#define AP3216C_ALS_DATA_L          0x0C
#define AP3216C_PS_DATA_L           0x0E
#define AP3216C_ALS_CONFIG_REG    0x10
#define AP3216C_ADDR                0x3C

extern I2C_HandleTypeDef hi2c2;
void SystemClock_Config(void);

enum als_range {
    AP3216C_ALS_RANGE_20661, //Resolution = 0.35 lux/count(default)
    AP3216C_ALS_RANGE_5162,    //Resolution = 0.0788 lux/count
    AP3216C_ALS_RANGE_1291,    //Resolution = 0.0197 lux/count
    AP3216C_ALS_RANGE_323       //Resolution = 0.0049 lux/count
};

//fprintf()
int fputc(int ch, FILE *f) {
```

```
    HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 1000);
    return ch;
}

uint8_t AP3216C_WriteOneByte(uint8_t reg, uint8_t data) {
    return HAL_I2C_Mem_Write(&hi2c2, AP3216C_ADDR, reg,
I2C_MEMADD_SIZE_8BIT, &data, 1, 100);
}

uint8_t AP3216C_ReadOneByte(uint8_t reg) {
    uint8_t data;
    HAL_I2C_Mem_Read(&hi2c2, AP3216C_ADDR, reg,
I2C_MEMADD_SIZE_8BIT, &data, 1, 100);
    return data;
}

float RoundToOneDecimal(float value) {
    return ((int)(value * 10 + 0.5)) / 10.0f;
}

void AP3216C_Init(void) {
    AP3216C_WriteOneByte(AP3216C_CONFIG_REG, 0x04); // Reset
    HAL_Delay(50);
    AP3216C_WriteOneByte(AP3216C_CONFIG_REG, 0x03); // Enable ALS + PS
+ IR

    if (AP3216C_ReadOneByte(AP3216C_CONFIG_REG) == 0x03) {
        printf("AP3216C Init OK!\r\n");
    } else {
        printf("AP3216C Init Failed!\r\n");
```

```
    }
}

uint16_t AP3216C_Read_PS(void) {
    uint32_t buf[2];
    uint32_t read_data;
    uint16_t proximity = 0;

    for (int i = 0; i < 2; i++) {
        buf[i] = AP3216C_ReadOneByte(AP3216C_PS_DATA_L + i);
    }

    read_data = buf[0] + (buf[1] << 8);
    proximity = (read_data & 0x000F) + (((read_data >> 8) & 0x3F) << 4);

    return proximity;
}

float AP3216C_Read_ALS(void) {
    uint32_t buf[2];
    uint32_t read_data;
    float brightness = 0;

    for (int i = 0; i < 2; i++) {
        buf[i] = AP3216C_ReadOneByte(AP3216C_ALS_DATA_L + i);
    }

    read_data = buf[0] + (buf[1] << 8);
    uint8_t range = (AP3216C_ReadOneByte(AP3216C_ALS_CONFIG_REG) >> 4)
& 0x03;
```

```
    switch (range) {
        case AP3216C_ALS_RANGE_20661: brightness = 0.35 * read_data; break;
        case AP3216C_ALS_RANGE_5162:  brightness = 0.0788 * read_data;
break;
        case AP3216C_ALS_RANGE_1291:  brightness = 0.0197 * read_data;
break;
        case AP3216C_ALS_RANGE_323:   brightness = 0.0049 * read_data;
break;
        default: break;
    }
    return brightness;
}

void LCD_Init_Display(void) {
    drv_lcd_init();
    lcd_clear(WHITE);
    lcd_set_color(WHITE, BLACK);
}

void Display_Sensor_Data(uint16_t ps_data, float als_data) {
        char buffer[32];

    float als_formatted = RoundToOneDecimal(als_data);

    sprintf(buffer, "PS: %.1f", (float)ps_data);
    lcd_show_string(30, 70, 32, buffer);

    sprintf(buffer, "ALS: %.1f", als_formatted);
    lcd_show_string(30, 110, 32, buffer);
```

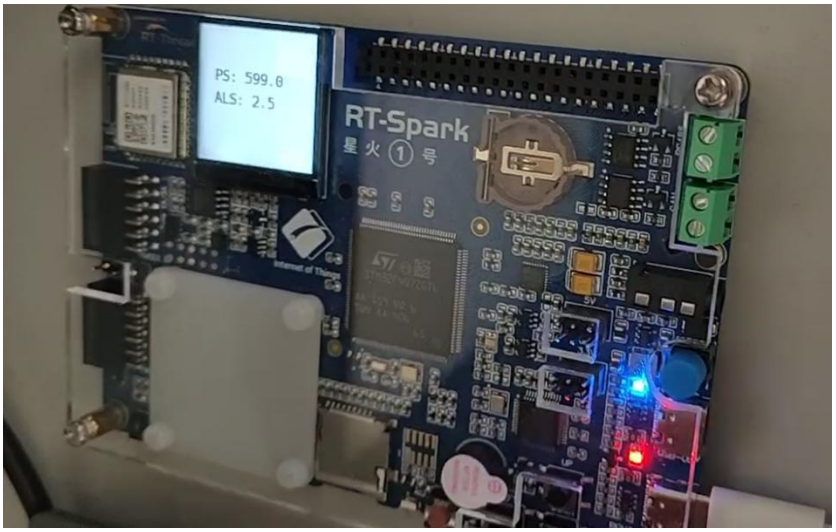
```
    printf("PS: %d\r\n", ps_data);
    printf("ALS: %.1f\r\n", als_formatted);
}

int main(void) {
    HAL_Init();
    SystemClock_Config();

    MX_GPIO_Init();
    MX_FSMC_Init();
    MX_I2C2_Init();
    MX_TIM14_Init();
    MX_USART1_UART_Init();

    HAL_TIM_Base_Start_IT(&htim14);
    HAL_TIM_PWM_Start(&htim14, TIM_CHANNEL_1);

    LCD_Init_Display();
    AP3216C_Init();

    while (1) {
        uint16_t ps = AP3216C_Read_PS();
        float als = AP3216C_Read_ALS();
        Display_Sensor_Data(ps, als);
        HAL_Delay(2000);
    }
}

void SystemClock_Config(void)
```

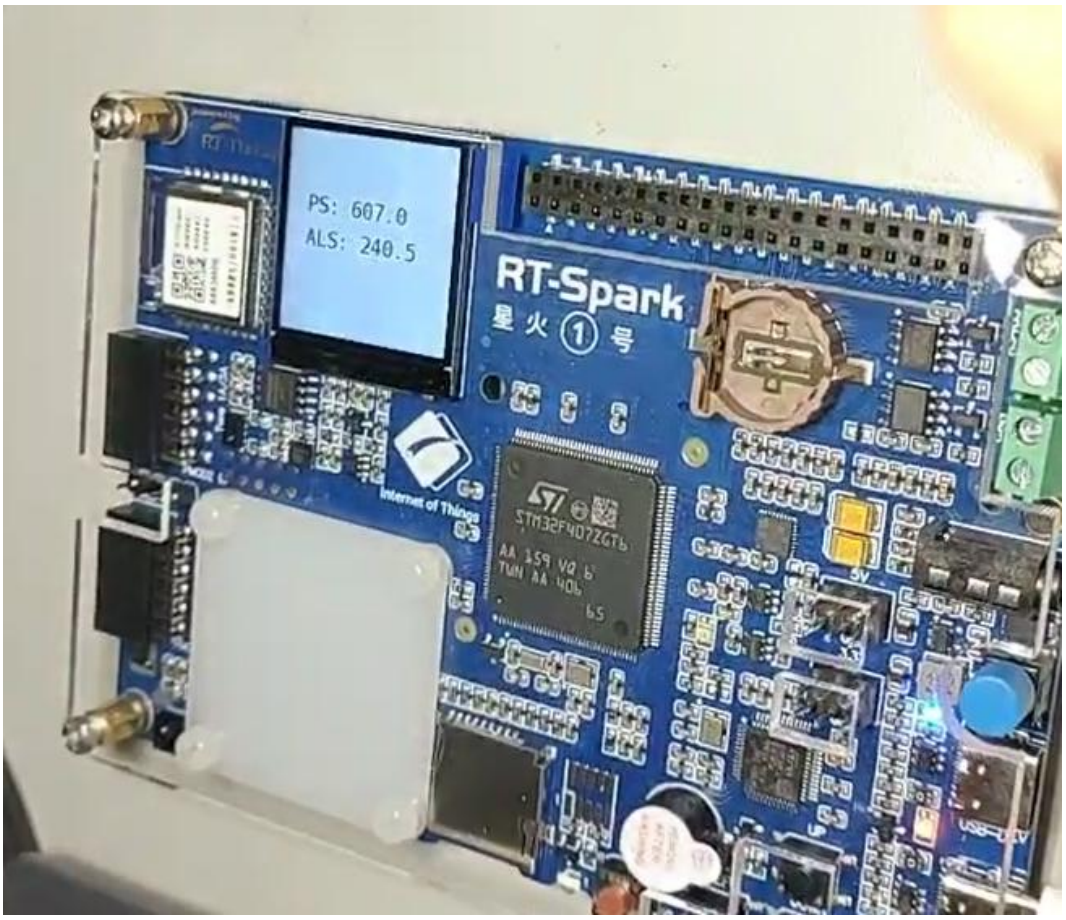
```
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_S
CALE1);

  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 = 4;
  RCC_OscInitStruct.PLL.PLLN = 168;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  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;
```

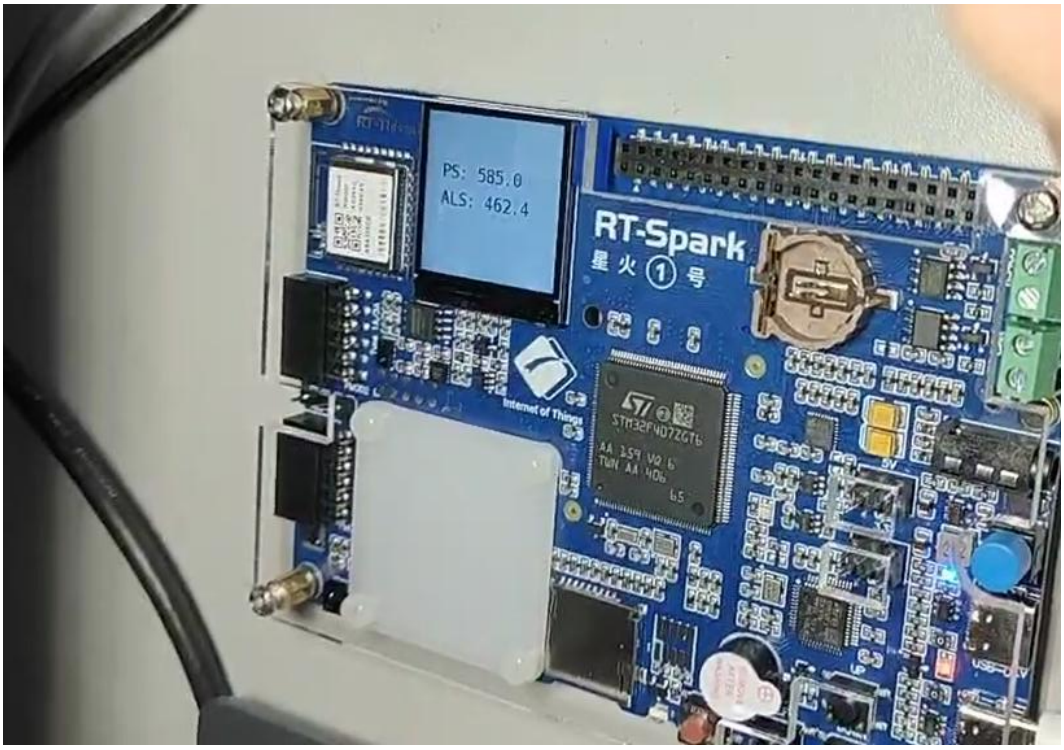
```
  if  (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,  FLASH_LATENCY_5)  !=
HAL_OK)
  {
    Error_Handler();
  }
}

void Error_Handler(void)
{
  __disable_irq();
  while (1){ }
}

#ifdef  USE_FULL_ASSERT
void assert_failed(uint8_t *file, uint32_t line){}
#endif /* USE_FULL_ASSERT */
```

# 3 实验结果分析与总结

代码工程编译完成上板运行，可以看到在不加外部强光下 ALS 值很小，仅为 2.5，后续加了强光，ALS 值飙升值 242 甚至 600，传感器显示无误，完成实验要求。详细内容请见附带的视频：




PS: 607.0
ALS: 240.5
RT-Spark
星火①号
Internet of Things
STM32F407ZGT6


PS: 585.0
ALS: 462.4
RT-Spark
星火①号
Internet of Things
STM32F407ZGT6