鸿蒙-海思物联网系统开发板

（鸿蒙）物联网系统开发板（基于海思 Hi3863）

用户手册

四梯科技有限公司

目录

4T_HRM_QS2

一、总体概述

本产品的设计是基于海思 Hi3863V100 芯片设计的高性能、低功耗 Wi-Fi 物联网开发板。海思 Hi3863V100 是 2.4GHz Wi-Fi 6 星闪多模 IoT SoC 芯片，旨在为智能家居、工业控制、消费电子等领域提供高性价比、开发便捷的无线连接解决方案，适用于大小家电常用电器类物联网智能场景。

本硬件开发板构建了一个完整的物联网硬件生态系统，实现了物联网完整的运行过程，实现从 信息采集 -> 信息传输 -> 信息处理和分析 -> 决策执行 的一整套完整的物联网闭环流程。整套硬件包含如下硬件：

首先是核心计算与控制单元，作为硬件系统的大脑，负责执行逻辑、处理数据和控制外设，本产品使用的是海思 Hi3863V100 芯片 Soc。Soc 芯片内部集成 2.4GHz Wi-Fi 6 星闪多模，可以实现物联网开发的无线通信。

其次是常用的输入输出接口等典型外设，如 LED 指示灯、蜂鸣器、用户按键等，以及一个 SPI 接口的 2.4 寸 LCD 屏幕。常用的输入输出接口可以作为信息的执行机构，对数据处理的结果进行呈现。

最后是有部分物联网的存储模块，Soc 本身自带的 Flash ROM：4MB、SRAM：606KB、ROM：300KB。此外，板卡集成了 IIC 接口的 EEPROM 芯片 AT24C02，FLASH 芯片 W25Q16，以及一个 micro SD 卡卡座以供扩展设备的存储功能。

本产品所使用的海思 Soc 芯片 Hi3863V100 具有以下一些特点：

1. 集成 IEEE 802.11 b/g/n/ax 基带和 RF 基带，包括功率放大器 PA、低噪声放大器 LNA、RF balun、天线开关以及电源管理模块等。

2. 支持 20MHz 频宽，提供最大 114.7Mbps 物理层速率，支持更大的发射功率和更远的覆盖范围。

3. 支持星闪 SLE 1MHz/2MHz/4MHz 频宽、SLE1.0 协议、支持 SLE 网关功能，最大空口速率 12Mbps。

4. Wi-Fi6 主流 Wi-Fi 连接：提供更高速率、更大的用户并发数。Wi-Fi6 支持 OFDMA、空分复用技术（SR）和 BSS Coloring 等新技术，有效降低同频干扰，提升用户并发度。

5. 星闪极致互联：提供超低时延空口，极致体验。星闪 SLE 通过全新的物理层设计，支持更小的空口时隙，端到端时延大幅下降，获得超强的覆盖增益和抗干扰能力。

6. 简易配网，异常快速定位：提升配网成功率，降低不必要退货，支持星闪近距离发现，实现待配网设备的自动发现和密码获取，设备靠近一键配网。

1.1 产品特点

1）鸿蒙系统，完美支持：完美兼容 OpenHarmony，轻松接入华为鸿蒙生态，实现跨设备协同。

2）强劲性能，星闪互联：内置强大的 32 位 MCU，最大工作频率 240MHz，提供充足的算力和存储空间。

3）万物互联，完整生态：本产品可以实现从 信息采集 -> 信息传输 -> 信息处理和分析 -> 决策执行 的一整套完整的物联网闭环流程。

4）超低功耗，超长待机：支持多种低功耗模式，待机功耗低至微安级，满足各种严苛场景需求。

5）人机交互，体验超绝：配备有一个 2.4 英寸 TFT LCD 彩色显示屏；清晰地展示所有信息，也可通过按键输入来获取相关信息，LED 和蜂鸣器也都可以作为数据和状态的呈现。

1.2 资源配置

1）海思 Hi3863V100 核心处理器

2）USB 转串口芯片 CH340

3）2.4 英寸 TFT LCD 彩色显示屏

4）非接触式读写器系统

5）Micro SD 存储卡扩展电路

6）EEPROM 存储电路

7）实时时钟（RTC）电路

8）NOR Flash 存储电路

9）1 路蜂鸣器

10）3 路 LED

11）1 路复位按键

12）4 路 ADC 按键

13）USB-C 接口

1.3 开发环境

1）集成开发环境（IDE）： HiSpark Studio

2）开发工具：HiSpark Studio

3）下载工具：板载 USB 转串口芯片 CH340，可通过串口进行下载

1.4 订购信息

1).官方淘宝：https://gxct.taobao.com/

2).四梯商城：https://4t.wiki/mall

1.5 获取支持

请通过以下方式联系我们，获取更多硬件学习资源和技术支持。

1).技术支持：tech@4t.wiki

2).交流社区：https://www.4t.wiki/community

3).学习资源：https://www.4t.wiki/curriculum

4).Github 仓库地址：

5).Gitee 仓库地址：

6).打开 4t.wiki 网站，获取更多资讯。

7).打开 https://space.bilibili.com/1208440832，看 B 站官方视频学习。

1.6 版本信息

版本编号	日期	修改内容	页码
V1.0	2026-02	新修订	1-31

二、硬件规格详情

系统框图如下所示：

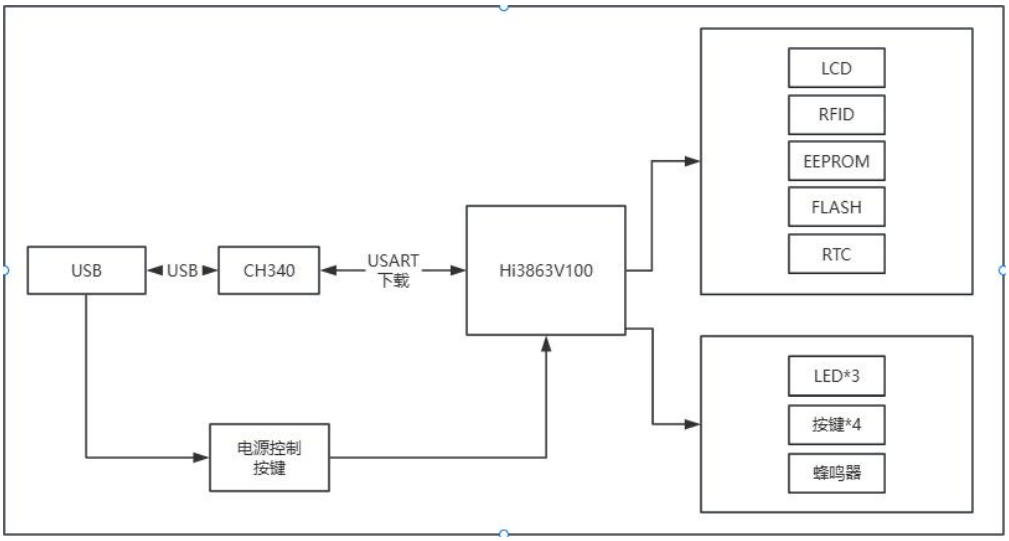

图 2.1　系统框图

2.1 供电方式

- 供电接口： Type-C USB 接口
  输入电压：5V DC ±5%
  额定电流：500mA

2.2 主控单元

- 型号：海思 Hi3863V100
- 架构：32 位 RISC-V 微处理器
- CPU 主频：240MHz
- 存储器：
  Flash ROM：4MB
  SRAM：606KB
  ROM：300KB
- 工作电压：3.3V

2.3 时钟与复位系统

- 时钟源：配备独立 24MHZ 晶振
  复位电路：配备复位电路与复位按键

2.4 通信接口

- USB-to-USART：CH340C，实现 USB 串口下载与调试
- Wi-Fi 6 无线通信
- 星闪无线通信

- RFID 非接触式通信接口

2.5 人机交互接口

- 显示输出：2.4 英寸 TFT LCD 彩色显示屏
  接口类型：SPI 接口
  分辨率：240*32（RGB）
  驱动 IC 型号：ST7789
- 按键输入
  复位按键 1 路
  ADC 按键 4 路

2.6 物理特性

- PCB 尺寸：120mm * 90mm * 1.6mm
- 产品尺寸：124mm * 94mm * 3.6mm

三、接线详情

海思 Hi3863 芯片			
PIN	名称	接线	说明
1	RFIO	天线接口	板载 PCB 天线和 IPEX-1 可选
2	AVDD33_RF1	电源接口	+3V3
3	AVDD33_RF0	电源接口	+3V3
4	GPIO_00	SPI_CS	SPI 片选信号线（扩展接口）
5	GPIO_01	SPI_MOSI	SPI 信号线
6	GPIO_02	SD_CS	SD 卡片选信号线
7	GPIO_03	LCD_WR/DC	LCD 读写控制线
8	GPIO_04	SPI_MISO	SPI 信号线
9	GPIO_05	LCS_CS	LCD 片选信号线
10	GPIO_06	SPI_CLK	SPI 时钟线
11	GPIO_07	BUZZ	蜂鸣器
12	GPIO_08	ADC_KEY	ADC 按键
13	GPIO_09	ADC	扩展接口 ADC
14	GPIO_10	LED1	红灯
15	GPIO_11	LED2	绿灯
16	GPIO_12	LED3	黄灯
17	VDD33 _OUT	电源接口	+3V3

18	VBAT_IN	供电接口	+3V3
19	BUCK_IN	供电接口	+3V3
20	BUCK_LX	BUCK 方案	
21	AVSS_PGND	GND	GND 管脚
22	BUCK_OUT	BUCK 方案	
23	PWR_SEL	PMU 控制	+3V3
24	GPIO_13	RFID_CS	RFID 片选信号线
25	GPIO_14	FLASH_CS	FLASH 片选信号线
26	UART1_TX	IIC_SDA	IIC 数据线
27	UART1_RX	IIC_SCL	IIC 时钟线
28	UART0_TX	UART0_TX	USB 转串口
29	UART0_RX	UART0_RX	USB 转串口
30	NC	NC	悬空
31	XIN	XIN	晶振引脚
32	XOUT	XOUT	晶振引脚
33	VDD_DIG	NC	悬空
34	DVDD3318	供电接口	+3V3
35	IOLDO18	LDO 输出	LDO 输出
36	AVDD33_1	供电接口	+3V3
37	AVDD33_0	供电接口	+3V3
38	PWR_ON	PMU 控制	复位引脚
39	AVDD33_RF2	RF 电源输入	+3V3
40	RF1	ANA	RF 接口

四、测试程序简介

这段代码是一个综合性的嵌入式系统演示程序，它同时运行多个任务。

LED 控制；

PWM 蜂鸣器；

Flash 存储器；

EEPROM (AT24C02);

RTC 实时时钟（BM8563）；

SD 卡；

RFID 读卡器；

LCD 显示屏；

这些任务并发执行，但通过互斥锁（mutex）来保护共享资源（如 I2C 和 SPI 总线），以避免冲突。

函数功能说明：

定义每个任务的优先级，决定了任务被调度执行的先后顺序，数值越小的代表优先级越高。

```
#define LED_TASK_PRIORITY          20
#define PWM_TASK_PRIORITY          20
#define AT24C02_TASK_PRIORITY      22
#define RTC_TASK_PRIORITY          22
#define FLASH_TASK_PRIORITY        23
#define SDCARD_TASK_PRIORITY       23
#define RFID_TASK_PRIORITY         24
#define LCD_TASK_PRIORITY          26
#define DEMO_TASK_PRIORITY         24
```

初始化程序所需的所有互斥锁，创建 I2C 总线互斥锁，SPI 总线互斥锁，RFID 状态互斥锁。

```
static void init_mutexes(void)
{
    int ret;
    /* 初始化 I2C 互斥锁 */
    ret = osal_mutex_init(&i2c_mutex);
```

```
    if (ret != OSAL_SUCCESS) {
        osal_printk("Error: Failed to initialize I2C mutex, ret=%d\r\n", ret);
    } else {
        osal_printk("I2C mutex initialized successfully\r\n");
    }
    /* 初始化 SPI 互斥锁 */
    ret = osal_mutex_init(&spi_mutex);
    if (ret != OSAL_SUCCESS) {
        osal_printk("Error: Failed to initialize SPI mutex, ret=%d\r\n", ret);
    } else {
        osal_printk("SPI mutex initialized successfully\r\n");
    }
    /* 初始化 RFID 状态互斥锁 */
    ......
}
```

销毁互斥锁，销毁已经创建的互斥锁释放资源。

```
static void destroy_mutexes(void)
{
    osal_mutex_destroy(&i2c_mutex);
    osal_printk("I2C mutex destroyed\r\n");

    osal_mutex_destroy(&spi_mutex);
    osal_printk("SPI mutex destroyed\r\n");

    osal_mutex_destroy(&g_rfid_status_mutex);
    osal_printk("RFID status mutex destroyed\r\n");
}
```

LED 闪烁任务，循环亮灭，不需要互斥锁。

```
static void *led_blink_task_thread(void *arg)
{
    unused(arg);
    osal_printk("LED Blink Task Started\r\n");
    led_blink_task(NULL);
    osal_printk("LED Blink Task Completed\r\n");
    return NULL;
}
```

蜂鸣器任务，响 1 秒停止，不需要互斥锁。

```
static void *pwm_beep_task_thread(void *arg)
{
    unused(arg);
    osal_printk("PWM Beep Task Started\r\n");
    pwm_beep_task(NULL);
    osal_printk("PWM Beep Task Completed\r\n");
    return NULL;
}
```

EEPROM 读写任务：获取 I2C 互斥锁保护访问，执行 EEPROM 读写测试，释放 I2C 互斥锁，打印任务执行状态。

```
static void *i2c_at24c02_task_thread(void *arg)
{
    unused(arg);
    /* 获取 I2C 互斥锁 */
    int ret = osal_mutex_lock(&i2c_mutex);
    if (ret != OSAL_SUCCESS) {
        osal_printk("Error: Failed to lock I2C mutex, ret=%d\r\n", ret);
        return NULL;
    }
    osal_printk("24C02 I2C Task Started\r\n");
    i2c_at24c02_task(NULL);
    osal_printk("24C02 I2C Task Completed\r\n");
    /* 释放 I2C 互斥锁 */
    osal_mutex_unlock(&i2c_mutex);
    return NULL;
}
```

Flash 读写任务：获取 SPI 互斥锁保护访问，执行 Flash 读写测试，释放 SPI 互斥锁，打印任务执行状态。

```
static void *flash_spi_task_thread(void *arg)
{
    unused(arg);
    /* 获取 SPI 互斥锁 */
    int ret = osal_mutex_lock(&spi_mutex);
```

```
    if (ret != OSAL_SUCCESS) {
        osal_printk("Error: Failed to lock SPI mutex, ret=%d\r\n", ret);
        return NULL;
    }
    osal_printk("Flash SPI Task Started\r\n");
    flash_spi_task(NULL);
    osal_printk("Flash SPI Task Completed\r\n");

    /* 释放 SPI 互斥锁 */
    osal_mutex_unlock(&spi_mutex);
    return NULL;
}
```

BM8563 测试任务：获取 I2C 互斥锁保护访问，执行时钟读写测试，释放 I2C 互斥锁，打印任务执行状态。

```
static void *i2c_bm8563_task_thread(void *arg)
{
    unused(arg);
    /* 获取 I2C 互斥锁 */
    int ret = osal_mutex_lock(&i2c_mutex);
    if (ret != OSAL_SUCCESS) {
        osal_printk("Error: Failed to lock I2C mutex, ret=%d\r\n", ret);
        return NULL;
    }
    osal_printk("RTC (BM8563) Task Started\r\n");
    i2c_bm8563_task(NULL);
    osal_printk("RTC (BM8563) Task Completed\r\n");
    /* 释放 I2C 互斥锁 */
    osal_mutex_unlock(&i2c_mutex);
    return NULL;
}
```

SD 卡读写任务：获取 SPI 互斥锁保护访问，执行 SD 卡读写测试，释放 SPI 互斥锁，打印任务执行状态。

```
static void *sd_card_test_task_thread(void *arg)
{
    unused(arg);
    /* 获取 SPI 互斥锁 */
    int ret = osal_mutex_lock(&spi_mutex);
```

```
    if (ret != OSAL_SUCCESS) {
        osal_printk("Error: Failed to lock SPI mutex, ret=%d\r\n", ret);
        return NULL;
    }
    osal_printk("SD Card Task Started\r\n");
    sd_card_test_task(NULL);
    osal_printk("SD Card Task Completed\r\n");
    /* 释放 SPI 互斥锁 */
    osal_mutex_unlock(&spi_mutex);
    return NULL;
}
```

RFID 测试任务：循环检测 RFID 卡片，检测到卡片时显示，更新全局 RFID 状态信息，监控卡片离开状态，使用 SPI 互斥锁保护 RFID 模块访问。

```
static void get_rfid_status_string(char *buffer, size_t buffer_size)
{
    int ret = osal_mutex_lock(&g_rfid_status_mutex);
    if (ret == OSAL_SUCCESS) {
        strncpy(buffer, g_rfid_status.status_str, buffer_size);
        buffer[buffer_size - 1] = '\0';
        osal_mutex_unlock(&g_rfid_status_mutex);
    } else {
        strncpy(buffer, "RFID: error", buffer_size);
        buffer[buffer_size - 1] = '\0';
    }
}
```

LCD 测试任务：初始化 LCD 显示屏，实时收集外设状态信息，每 1 秒刷新一次显示，使用 SPI 互斥锁保护访问。

```
static void *lcd_display_task_thread(void *arg)
{
    unused(arg);
    char flash_buffer[64] = {0};
    char eeprom_buffer[64] = {0};
    char rtc_buffer[64] = {0};
    char sd_buffer[64] = {0};
    char rfid_buffer[64] = {0};
    char status_buffer[64] = {0};
    /* 获取 SPI 互斥锁 */
```

```
    int ret = osal_mutex_lock(&spi_mutex);
    if (ret != OSAL_SUCCESS) {
        osal_printk("Error: Failed to lock SPI mutex, ret=%d\r\n", ret);
        return NULL;
    }
    osal_printk("LCD Display Task Started\r\n");
    /* 初始化 LCD */
    spi_lcd_init();
    osal_printk("LCD Initialized\r\n");
    spi_lcd_clear(WHITE);
    /* 主显示循环 */
    while(1) {
        /* 更新显示内容 */
        snprintf(flash_buffer, sizeof(flash_buffer), "Flash: %02X %02X %02X %02X",
                 rx_buf[0], rx_buf[1], rx_buf[2], rx_buf[3]);
        // EEPROM 信息
                snprintf(eeprom_buffer, sizeof(eeprom_buffer), "EEPROM: %d bytes",
AT24C02_DATE_LEN);
        // RTC 信息
        if (read_time[0] != 0xFF) {
            snprintf(rtc_buffer, sizeof(rtc_buffer), "RTC: %02d:%02d:%02d",
                     bcd_to_dec(read_time[2] & 0x3F),
                     bcd_to_dec(read_time[1] & 0x7F),
                     bcd_to_dec(read_time[0] & 0x7F));
        } else {
            snprintf(rtc_buffer, sizeof(rtc_buffer), "RTC: error");
        }
        // SD 卡信息
        snprintf(sd_buffer, sizeof(sd_buffer), "SD OCR: 0x%08X", sd_get_ocr);
        // RFID 信息
        get_rfid_status_string(rfid_buffer, sizeof(rfid_buffer));
        // 系统状态
        static unsigned long counter = 0;
        counter++;
        snprintf(status_buffer, sizeof(status_buffer), "RUNNING... %lu", counter);
        /* 清屏并显示所有信息 */
        //spi_lcd_clear(WHITE);
        spi_lcd_display_string_line(0, 0, BLACK, WHITE, (uint8_t*)flash_buffer);
        spi_lcd_display_string_line(0, 1, BLACK, WHITE, (uint8_t*)eeprom_buffer);
        spi_lcd_display_string_line(0, 2, BLACK, WHITE, (uint8_t*)rtc_buffer);
        spi_lcd_display_string_line(0, 3, BLACK, WHITE, (uint8_t*)sd_buffer);
        spi_lcd_display_string_line(0, 4, BLACK, WHITE, (uint8_t*)rfid_buffer);
        spi_lcd_display_string_line(0, 5, BLACK, WHITE, (uint8_t*)status_buffer);
```

```
        /* 释放 SPI 互斥锁（短暂释放以允许其他 SPI 设备访问） */
        osal_mutex_unlock(&spi_mutex);

        /* 显示刷新间隔 */
        osal_msleep(1000);

        /* 重新获取 SPI 互斥锁进行下一次显示 */
        ret = osal_mutex_lock(&spi_mutex);
        if (ret != OSAL_SUCCESS) {
            osal_printk("Error: Failed to lock SPI mutex for LCD, ret=%d\r\n", ret);
            break;
        }
    }
    /* 释放 SPI 互斥锁 */
    osal_mutex_unlock(&spi_mutex);

    osal_printk("LCD Display Task Completed\r\n");
    return NULL;
}
```

任务管理函数：定义任务函数、名称、优先级数组，循环创建 8 个并发任务线程，为每个任务设置优先级，输出每个任务的创建状态，返回成功创建的任务数量。

```
static int create_concurrent_tasks(void)
{
    int created_count = 0;

    /* 任务函数数组 */
    osal_kthread_handler task_functions[MAX_TASKS] = {
        (osal_kthread_handler)led_blink_task_thread,
        (osal_kthread_handler)pwm_beep_task_thread,
        (osal_kthread_handler)flash_spi_task_thread,
        (osal_kthread_handler)i2c_at24c02_task_thread,
        (osal_kthread_handler)i2c_bm8563_task_thread,
        (osal_kthread_handler)sd_card_test_task_thread,
        (osal_kthread_handler)rfid_test_task_thread,      /* 修改为持续运行的 RFID 任务 */
        (osal_kthread_handler)lcd_display_task_thread     /* 修改为持续运行的 LCD 任务 */
```

```
};

/* 任务名称数组 */
const char *task_names[MAX_TASKS] = {
    "LED_Blink_Thread",
    "PWM_Beep_Thread",
    "Flash_SPI_Thread",
    "24C02_I2C_Thread",
    "RTC_BM8563_Thread",
    "SD_Card_Thread",
    "RFID_Thread",
    "LCD_Display_Thread"
};

/* 任务优先级数组 */
int task_priorities[MAX_TASKS] = {
    LED_TASK_PRIORITY,
    PWM_TASK_PRIORITY,
    FLASH_TASK_PRIORITY,
    AT24C02_TASK_PRIORITY,
    RTC_TASK_PRIORITY,
    SDCARD_TASK_PRIORITY,
    RFID_TASK_PRIORITY,
    LCD_TASK_PRIORITY
};

/* 批量创建线程 */
for (int i = 0; i < MAX_TASKS; i++) {
    osal_task *task_handle = NULL;

    /* 使用 osal_kthread_create 创建任务 */
    osal_kthread_lock();
    task_handle = osal_kthread_create(task_functions[i],
                                      NULL,
                                      task_names[i],
                                      TASK_STACK_SIZE);

    if (task_handle != NULL) {
        /* 使用每个任务特定的优先级 */
        osal_kthread_set_priority(task_handle, task_priorities[i]);
        osal_printk("Success: Created %s with priority %d\r\n",
                    task_names[i], task_priorities[i]);
        created_count++;
```

```
        } else {
            osal_printk("Error: Failed to create %s\r\n", task_names[i]);
        }
        osal_kthread_unlock();

        /* 添加短暂延时，避免线程启动冲突 */
        osal_msleep(10);
    }

    return created_count;
}
```

综合演示任务：初始化所有互斥锁，初始化 RFID 模块，创建所有并发线程任务，等待任务执行，打印并在 LCD 上显示信息。

```
static void *concurrent_demo_task(void *arg)
{
    unused(arg);
    osal_printk("\r\n\r\n===== Concurrent Demo Task Start =====\r\n");
        osal_printk("This demo will run all tasks concurrently with proper resource locking.\r\n");
       osal_printk("Tasks using same resources (I2C/SPI) will be serialized with mutex locks.\r\n");
    osal_printk("Each task has its own priority for better scheduling.\r\n");
    osal_printk("RFID and LCD tasks will run continuously.\r\n");

    /* 初始化互斥锁 */
    init_mutexes();

    /* 初始化 RFID 模块 */
    int ret = osal_mutex_lock(&spi_mutex);
    if (ret == OSAL_SUCCESS) {
        MFRC522_Init();
        osal_printk("RFID Module Initialized\r\n");
        osal_mutex_unlock(&spi_mutex);
    }
    /* 创建并发任务线程 */
    int created_tasks = create_concurrent_tasks();
    if (created_tasks > 0) {
        osal_printk("Successfully created %d concurrent tasks\r\n", created_tasks);
        /* 等待所有任务完成 */
        wait_for_all_tasks();
```

```
        osal_printk("\r\n===== Demo Task Completed =====\r\n");
        osal_printk("Note: RFID and LCD tasks continue running in background.\r\n");
        /* 注意：这里不销毁互斥锁，因为 RFID 和 LCD 任务还在运行 */
        /* 在实际应用中，应该设计一个优雅的退出机制 */
    } else {
        osal_printk("Error: Failed to create any concurrent tasks\r\n");
        /* 销毁互斥锁 */
        destroy_mutexes();
    }
    return NULL;
}
```

程序主入口函数：输出程序启动信息，创建并发演示任务线程，设置演示任务优先级。

```
static void main_application_entry(void)
{
    osal_task *demo_task_handle = NULL;

    osal_printk("\r\n\r\n===== Concurrent Demo Application Start =====\r\n");
       osal_printk("This demo will run all tasks concurrently with proper resource locking.\r\n");
      osal_printk("Tasks using same resources (I2C/SPI) will be serialized with mutex locks.\r\n");
    osal_printk("Each task has its own priority for better scheduling.\r\n");
    osal_printk("RFID and LCD tasks will run continuously until system shutdown.\r\n");

    /* 创建并发演示任务 */
    osal_kthread_lock();
                                                    demo_task_handle =
osal_kthread_create((osal_kthread_handler)concurrent_demo_task,
                                        NULL,
                                        "ConcurrentDemoTask",
                                        TASK_STACK_SIZE);

    if (demo_task_handle != NULL) {
        osal_kthread_set_priority(demo_task_handle, DEMO_TASK_PRIORITY);
        osal_printk("Success: Concurrent demo task created with priority %d\r\n",
                DEMO_TASK_PRIORITY);
    } else {
        osal_printk("Error: Failed to create concurrent demo task\r\n");
    }
```

```
    osal_kthread_unlock();

    osal_printk("===== Concurrent Demo Execution Started =====\r\n");
    osal_printk("Main application continues, all tasks run in background.\r\n\r\n");
}
```

五、硬件资源布局

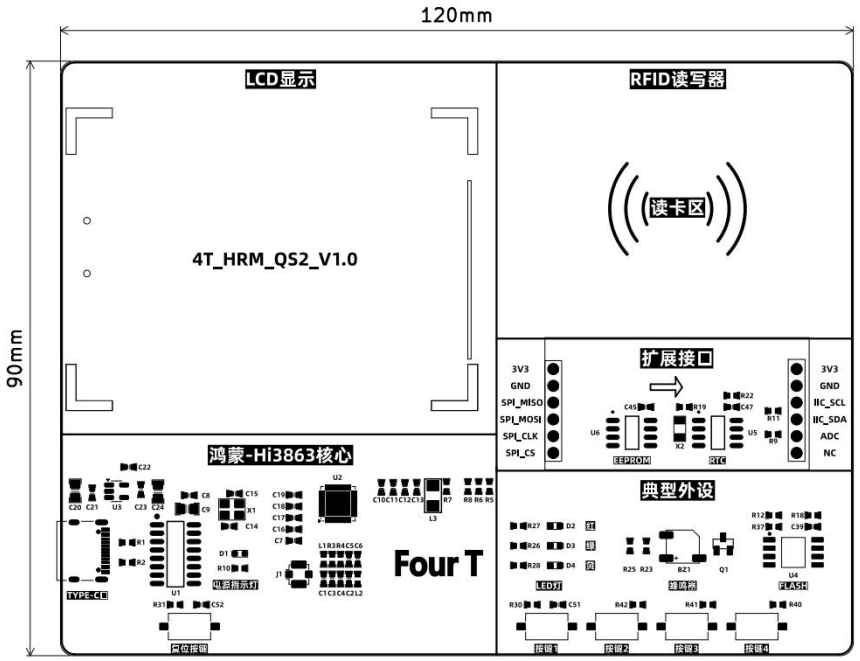

图 5.1 4T_HRM_QS2 硬件资源布局正面图

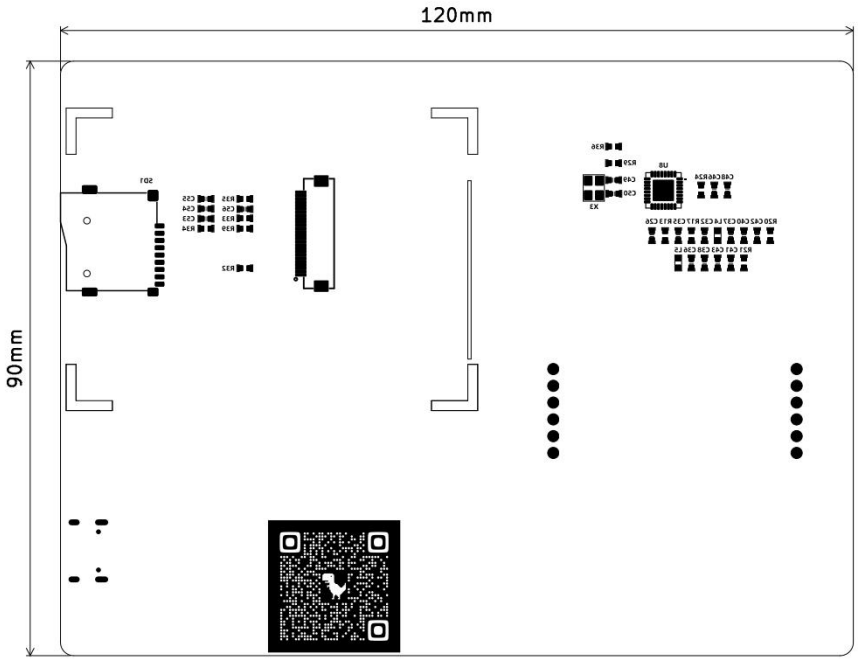

图 5.2 4T_HRM_QS2 硬件资源布局背面图

六、开发环境与工具

Window 环境下开发环境搭建。

点击网站

https://www.bearpi.cn/core_board/bearpi/pico/h3863/filebrowser/?path=7036653&fileID=7332219

下载 HiSparkStudio 一键环境安装_v1.0.3.exe，下载后将“.bak”后缀删除，双击打开

文件名	修改时间	大小	文件号
QCOM_V1.6.exe	2024-07-16	380KB	7036656
BurnTool_H3863.zip	2025-02-28	12MB	7036657
CH341SER.EXE	2024-07-16	383KB	7036684
DevTools_CFBB_V1.0.12.zip	2024-07-16	72MB	7036685
deveco-device-tool-all-in-one-1.1.7.exe	2024-07-16	497MB	7036686
MobaXterm_Installer_v21.4.zip	2024-07-16	26MB	7036687
VMware-player-full-16.2.1-18811642.exe	2024-07-16	585MB	7036688
RaiDrive.rar	2024-07-16	21MB	7036689
VSCodeUserSetup-x64-1.90.2.exe	2024-07-16	96MB	7036690
TCP UDP Socket调试工具.exe	2024-07-29	2MB	7236091
HiSparkStudioSetup-1.0.0.6.exe	2025-02-18	240MB	7328896
HiSparkStudio一键环境安装_v1.0.3.exe	2025-02-18	59MB	7332219

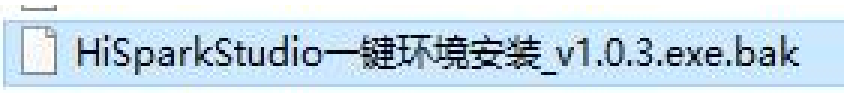

图 6.1 HiSparkStudio 一键环境下载

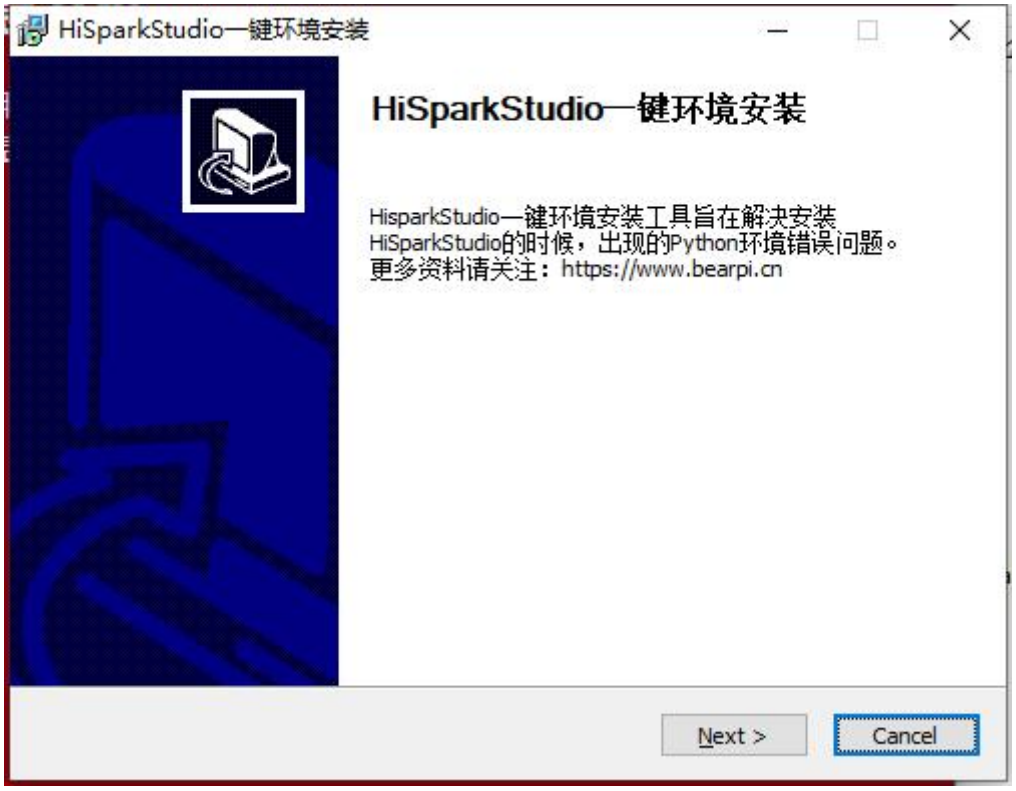

图 6.2 环境安装（1）

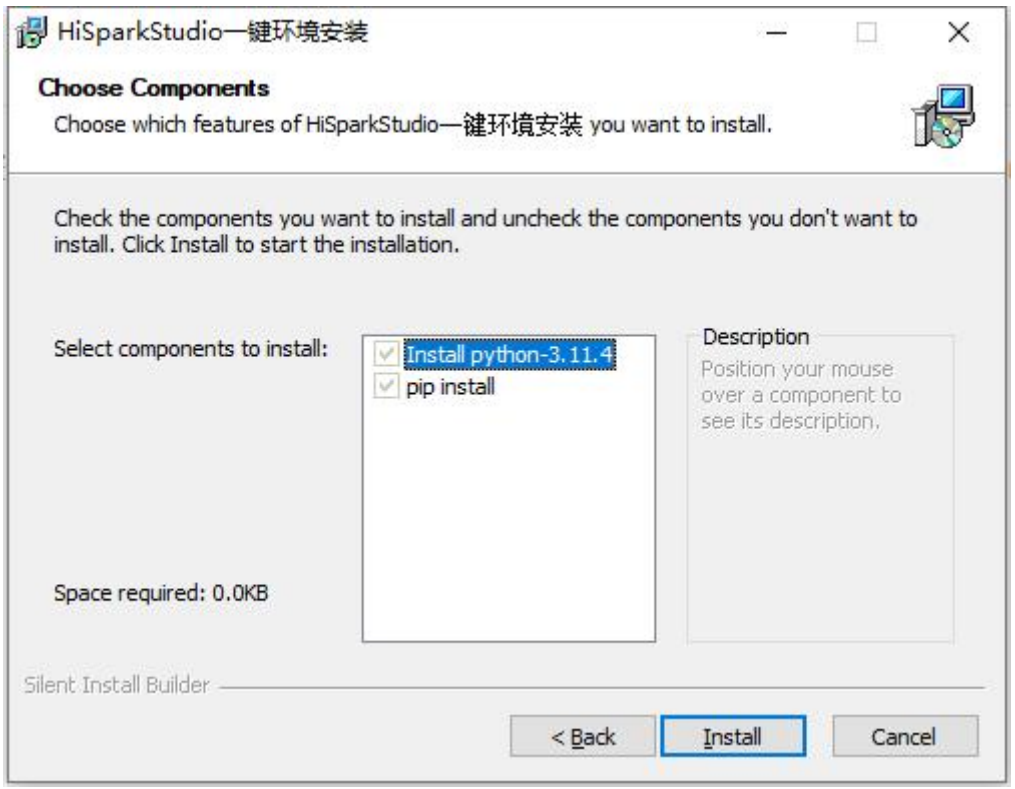

图 6.3 环境安装（2）

等待环境安装完毕，点击 Close

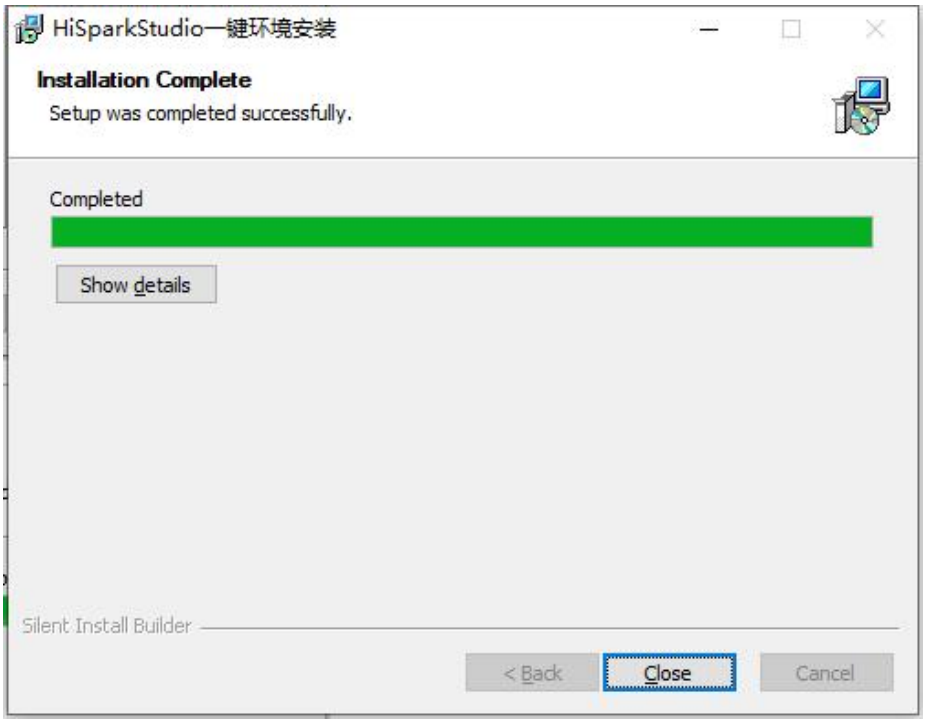

图 6.4 环境安装完毕

安装 HiSpark Studio IDE，双击下载的应用程序。

点击“我同意此协议”，下一步

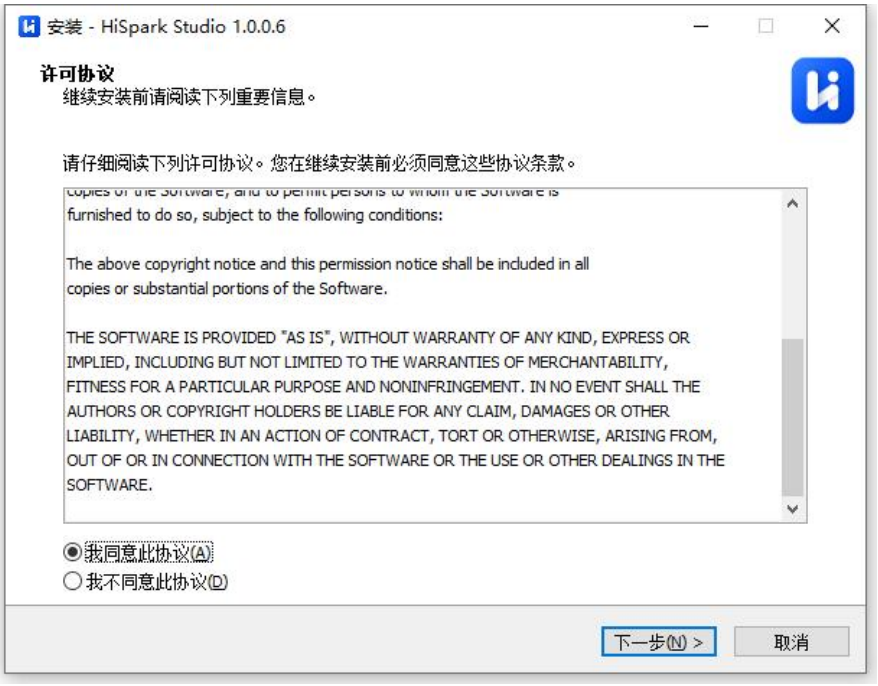

图 6.5 HiSpark Studio 安装许可协议

选择目标位置，注意安装路径不要有中文，下一步。

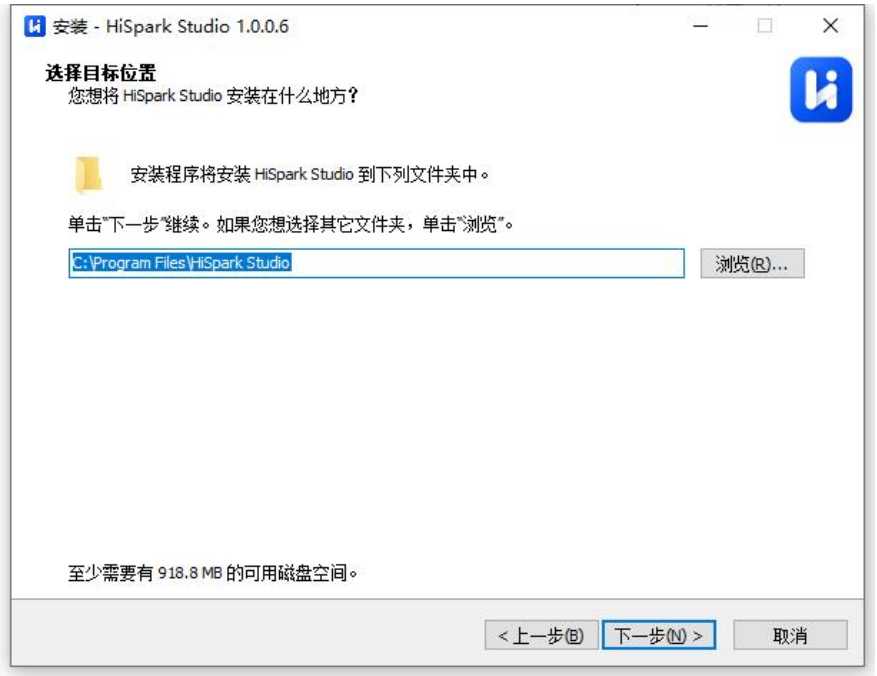

图 6.6 HiSpark Studio 安装路径选择

全部选择，下一步

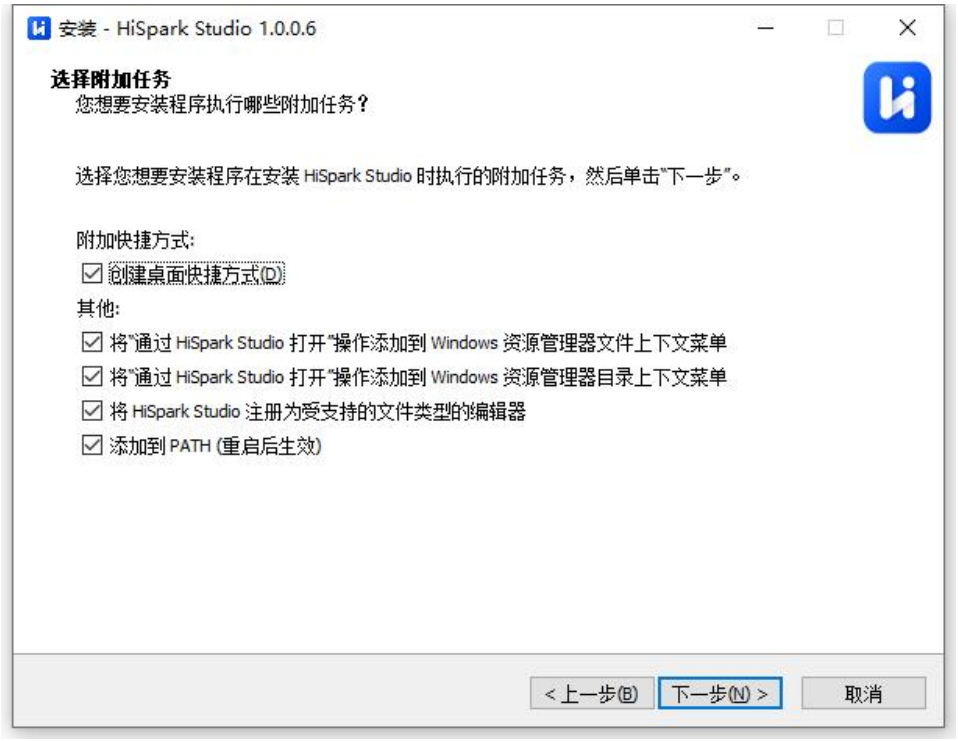

图 6.7 安装附加任务选择

点击安装。

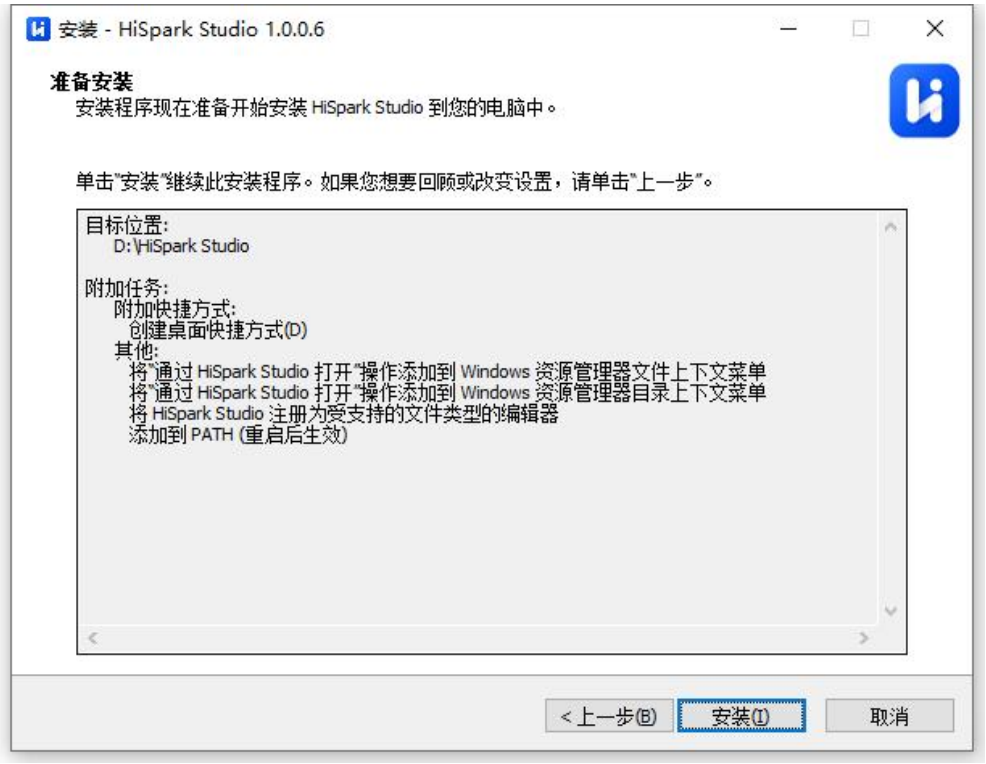

图 6.8 准备安装

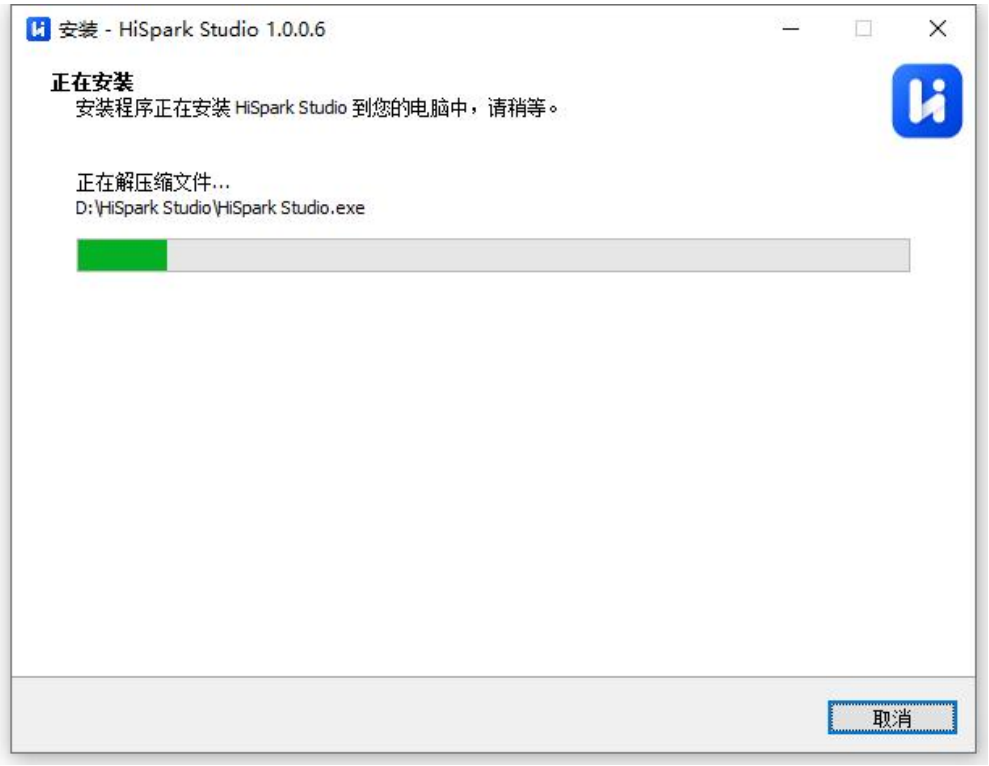

图 6.9 正在安装

图 6.10 安装完成

七、建立新工程

首先通过 zip 方式下载官方 SDK：https://gitee.com/HiSpark/fbb_ws63

点击克隆/下载

图 7.1 SDK 下载

下载 SDK 的 ZIP 压缩包。

图 7.2 下载 ZIP

下载后，解压“fbb_ws63_master”，在解压过程中可能需要关闭电脑杀毒软件

（防止有些文件当作病毒被删除掉），解压方式选择“Extract Here”即可（建议解压到 D 盘、E 盘等根目录，路径不要太深，且不要有中文路径）。

打开 HiSpark Studio，新建工程。

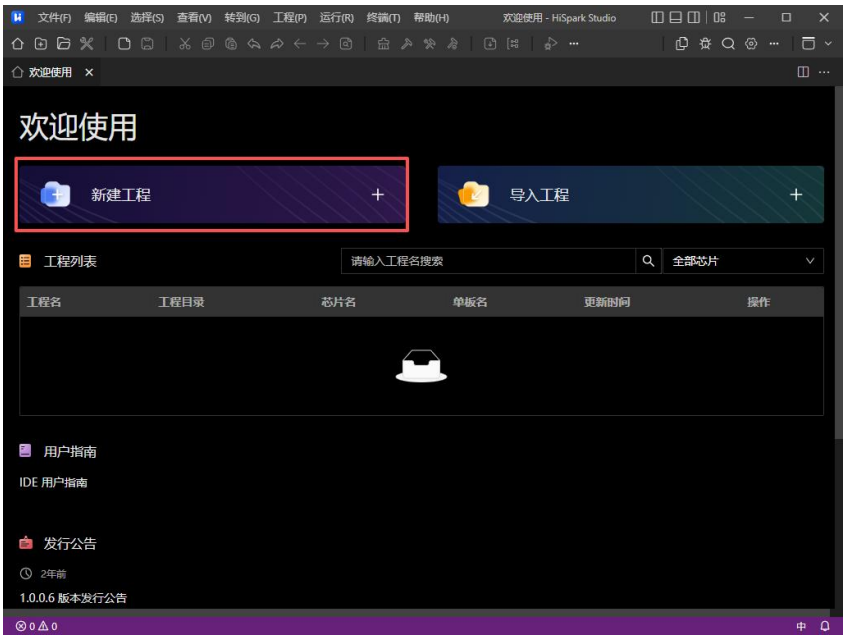

图 7.3 打开 HiSpark Studio

新建工程界面中芯片：“WS63”，工程名：“xxx”（用户自定义，但是不能带中文，特殊符号），软件包：“xxx/fbb_ws63/src”（SDK 软件包存放路径，这个地方一定要选到 src 层级，否则编译会失败），配置选择完成后，点击“完成按钮”。

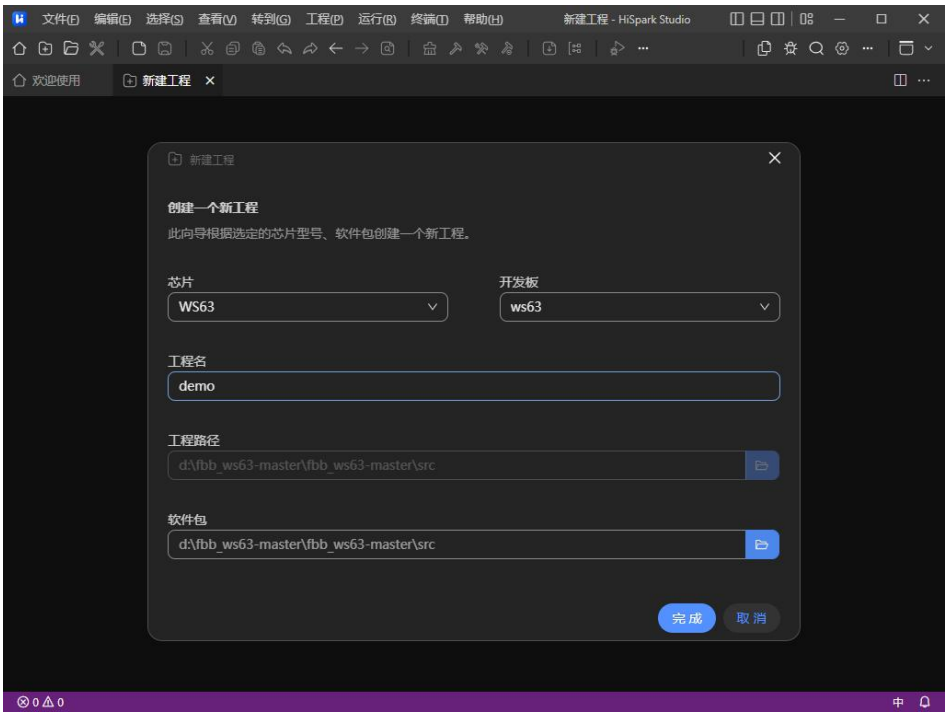

图 7.4 创建工程配置

点击完成后，点击 build 或 rebuild，编译一下。

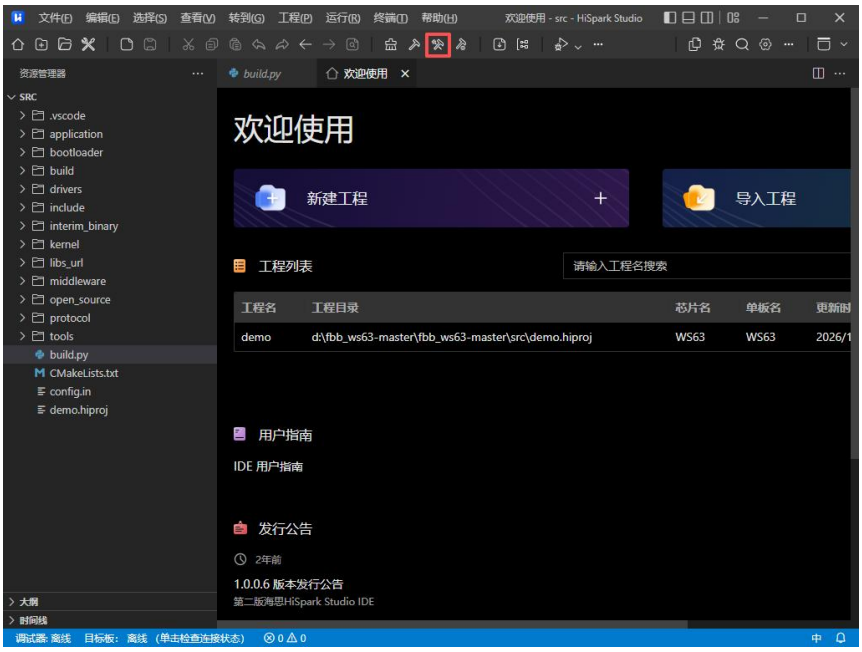

图 7.5 编译 SDK

编译完成显示 “SUCCESS”，说明编译成功。

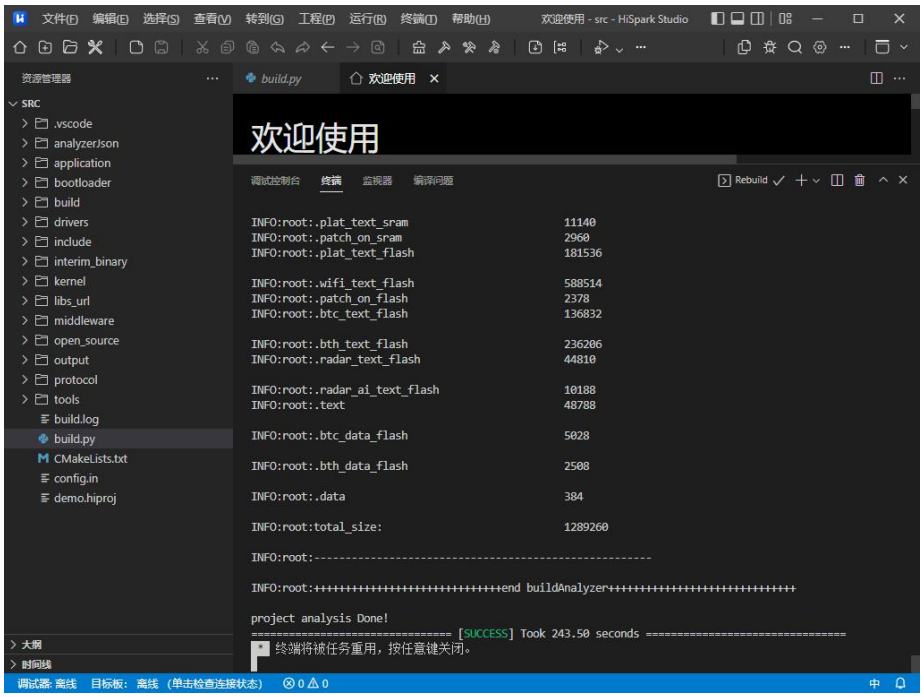
文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 工程(P) 运行(R) 终端(T) 帮助(H)
欢迎使用 - src - HiSpark Studio
资源管理器
SRC
.vscode
analyzerJson
application
bootloader
build
drivers
include
interim_binary
kernel
libs_url
middleware
open_source
output
protocol
tools
build.log
build.py
CMakeLists.txt
config.in
demo.hiproj
build.py
欢迎使用
欢迎使用
调试控制台 终端 监视器 编译问题
Rebuild
INFO:root:.plat_text_sram 11140
INFO:root:.patch_on_sram 2960
INFO:root:.plat_text_flash 181536
INFO:root:.wifi_text_flash 588514
INFO:root:.patch_on_flash 2378
INFO:root:.btc_text_flash 136832
INFO:root:.bth_text_flash 236206
INFO:root:.radar_text_flash 44810
INFO:root:.radar_ai_text_flash 10188
INFO:root:.text 48788
INFO:root:.btc_data_flash 5028
INFO:root:.bth_data_flash 2508
INFO:root:.data 384
INFO:root:total_size: 1289260
INFO:root:----------
INFO:root:++++++++++++++++++++++++++++++end buildAnalyzer++++++++++++++++++++++++++++++
project analysis Done!
[SUCCESS] Took 243.50 seconds
终端将被任务重用，按任意键关闭。
大纲
时间线
调试器: 离线 目标板: 离线 (单击检查连接状态)
0 0
中

图 7.6 编译成功

八、烧录代码

1.使用一条带数据传输功能的 typec 线将开发板与电脑连接。

2.下载并安装 ch340 驱动“CH341SER.EXE”，资料包中含有。

3.查看开发板与电脑连接后映射出来的端口，USB-SERIAL CH340 表示开发板串口。COM10 代表串口号，不同电脑数值可能不同。

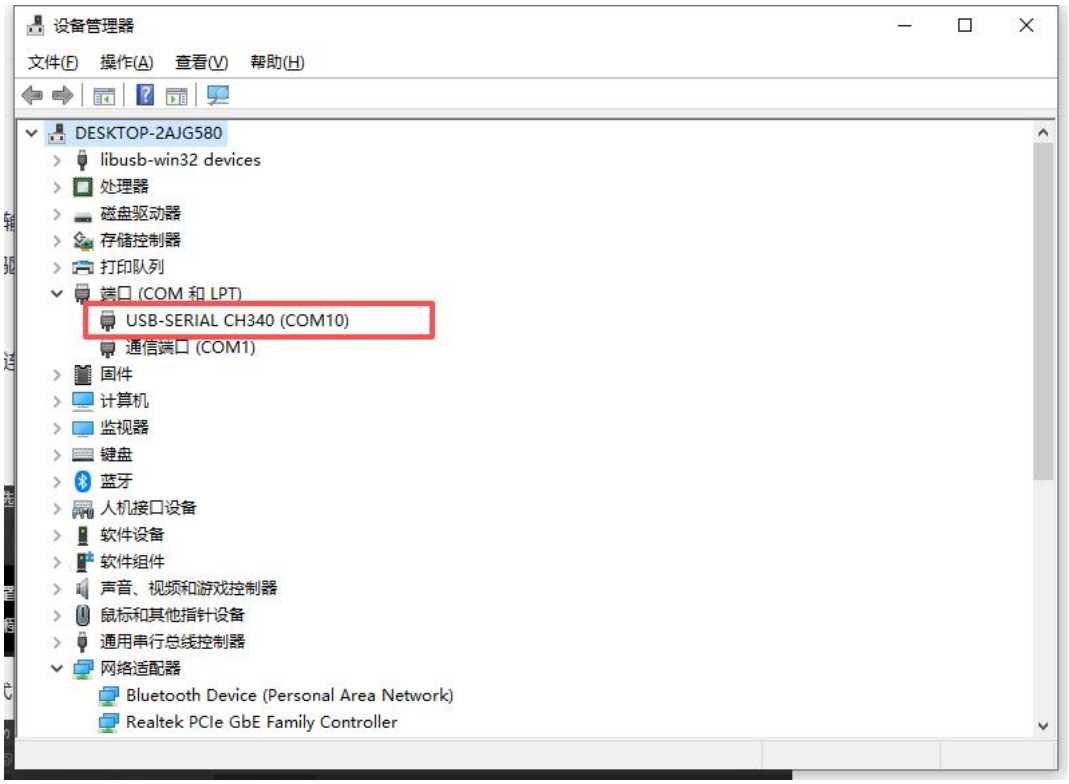

图 8.1 查看设备管理器端口

点击工程配置

图 8.2 配置工程参数

点击程序加载，传输方式 serial，端口选择对应的开发板端口号

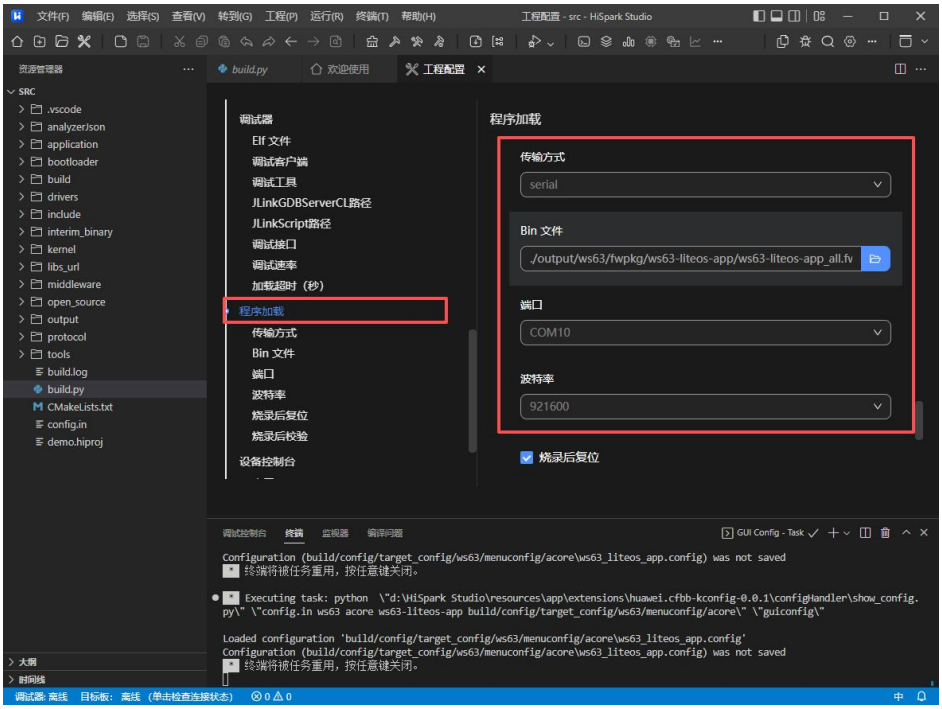

图 8.3 配置参数

配置成功后，点击系统配置，选择要下载的程序。

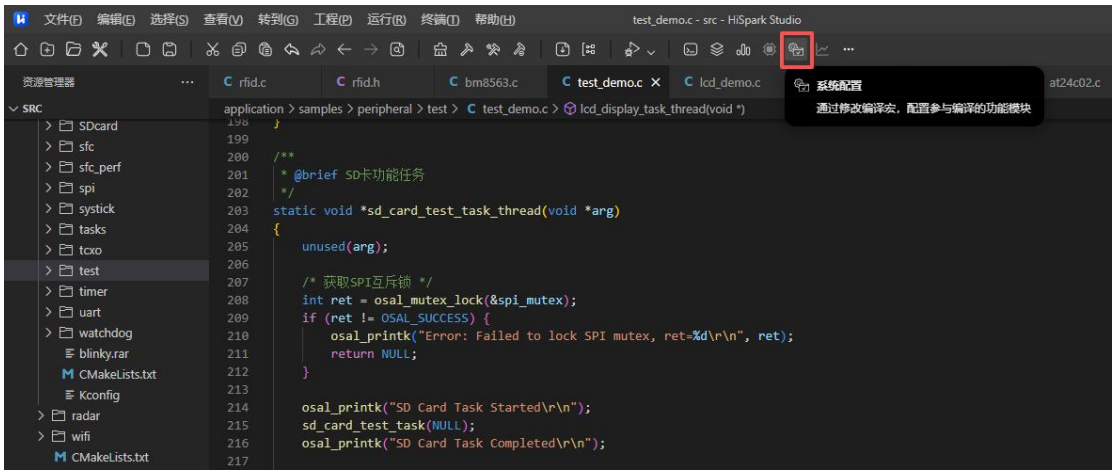

图 8.4 点击系统配置

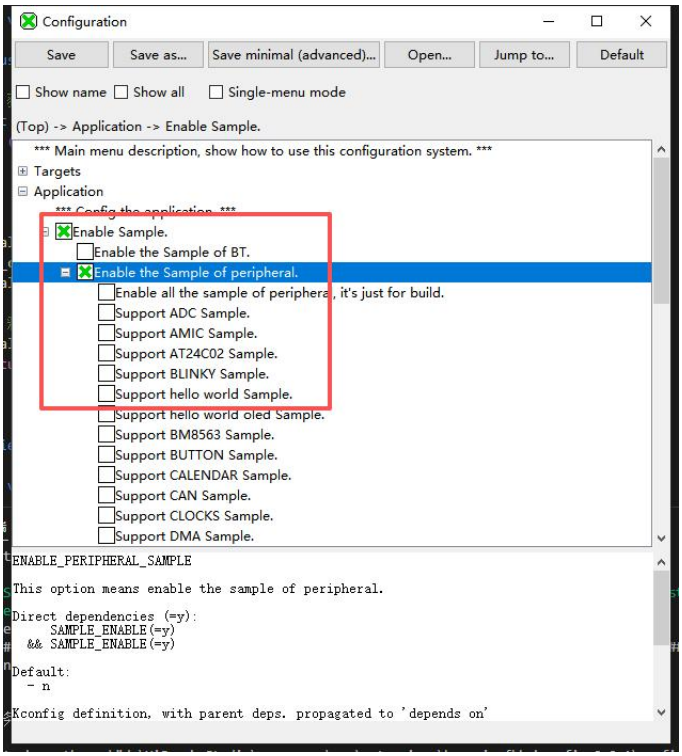

图 8.5 配置要下载的程序

选择后编译，再点击程序加载。

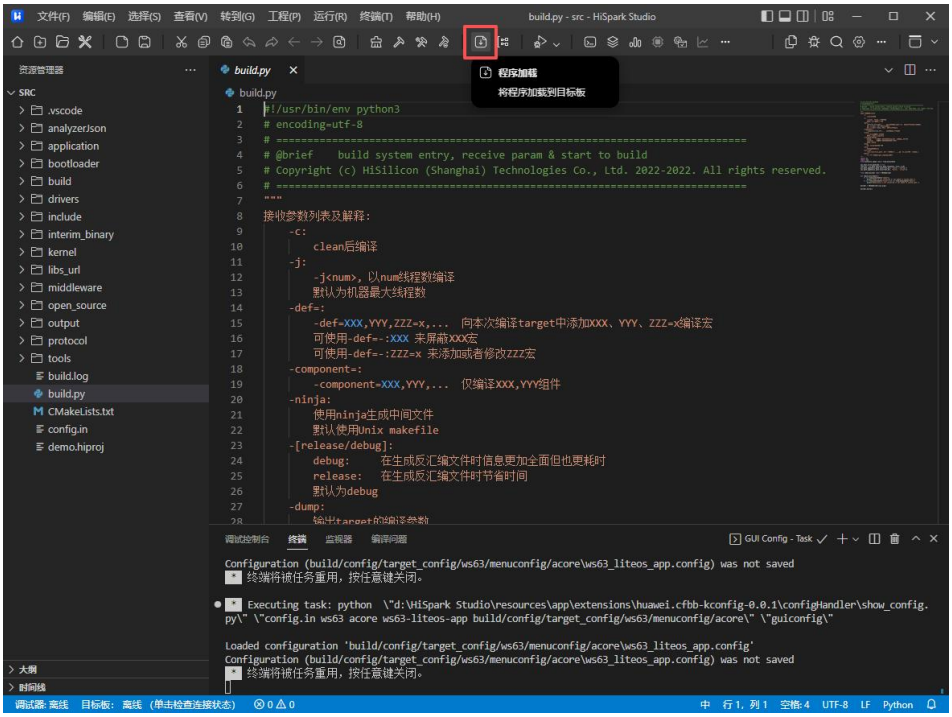

图 8.6 点击程序加载

终端提示“Connecting ，please reset device...",按下复位按键，开始烧录。

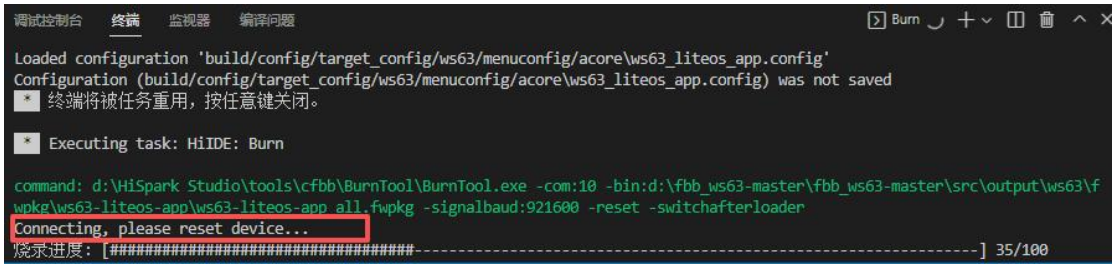

图 8.7 烧录中

烧录进度 100%，显示 All images burn successfully reset device，说明烧录成功。

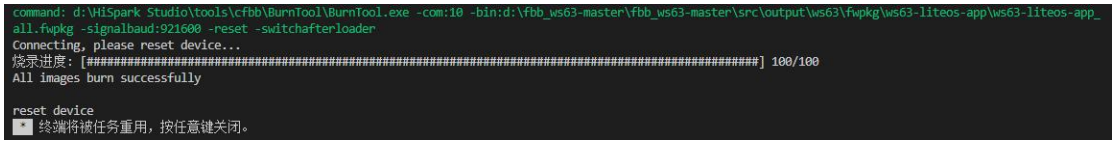

图 8.8 烧录中