RT-THREAD 星火 1 号用户手册

RT-THREAD 文档中心

上海睿赛德电子科技有限公司版权 **@2023**

WWW.RT-THREAD.ORG

Friday 30th June, 2023

版本和修订

Date	Version	Author	Note
2023-06-30	v0.1.0	RT-Thread	初始版本

目录

第 1 章

简介

“星火 1 号”，一款专为工程师和高校学生设计的嵌入式 RTOS 开发学习板。在这个科技飞速发展的时代，嵌入式系统已经成为了现代工业、交通、通信等众多领域的核心驱动力。而 RTOS 实时操作系统作为嵌入式领域的基石，更是工程师们必须熟练掌握的核心技术。作为业界主流的 RTOS 实时操作系统 RT-Thread，我们有义务帮助更多开发者掌握这项技术。为此，我们精心打造了一款专为工程师和高校学生设计的嵌入式开发学习板。

星火 1 号主控选用了目前行业中比较常用且学习门槛较低的 STM32F407，性能强劲、功能丰富，完全能够满足嵌入式入门的需求。此开发板不仅具有众多的板载资源（Flash 存储、WIFI 通信、多个传感器），还支持丰富的扩展接口，让您轻松实现各种复杂的应用场景。通过使用这款开发学习板，您将能够深入了解 RTOS/RT-Thread 的工作原理，提升自己的技能水平，为当前以及未来的职业生涯做好充分准备。

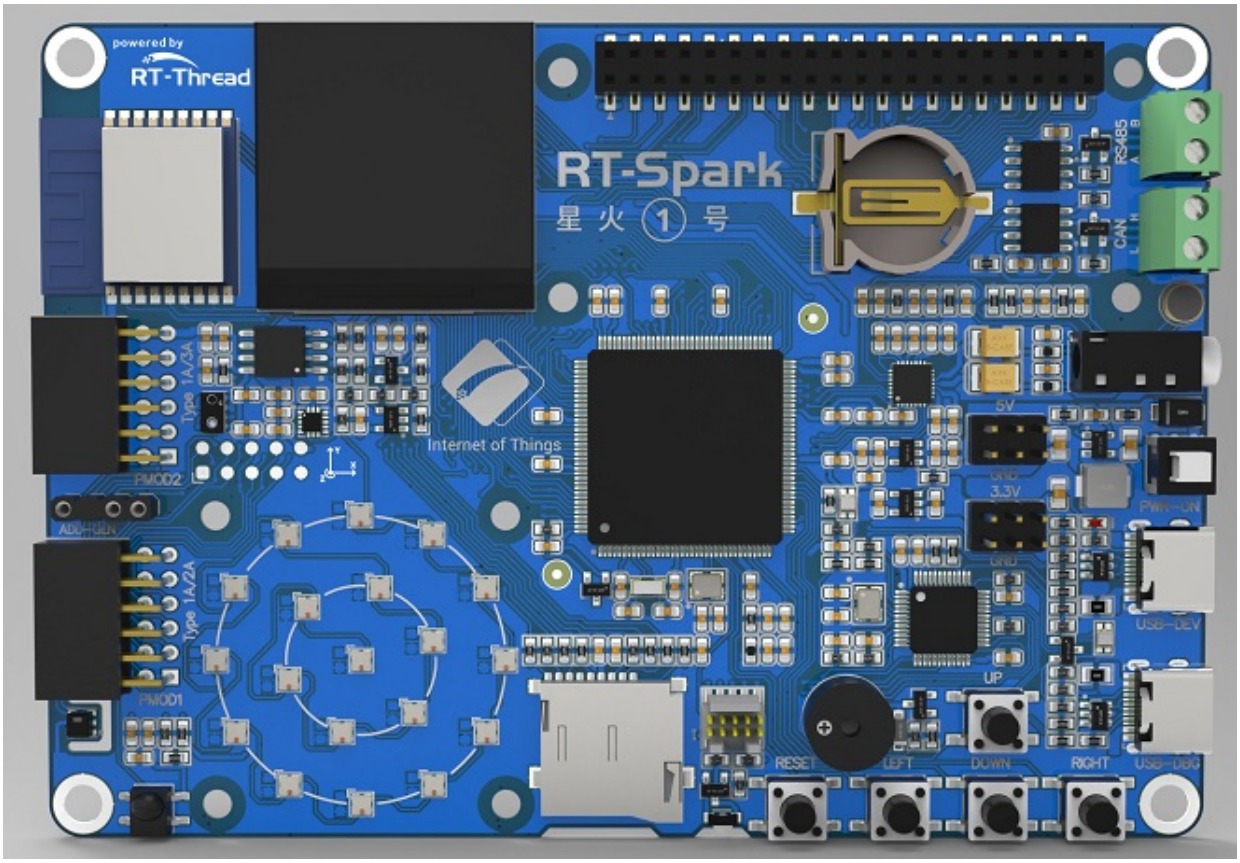

图 1.1: *星火 1 号开发板渲染图*

1.1 目录结构

```
|-- README.md
|-- docs
|   |-- SCH_LearnBoard_IoT_V0_0.pdf
|   `-- images
|-- libraries
|   |-- Board_Drivers
|   |-- HAL_Drivers
|   `-- STM32F4xx_HAL
|-- projects
|   |-- 01_basic_led_blink       # LED 闪烁例程
|   |-- 02_basic_rgb_led         # RGB LED 例程
|   |-- 03_basic_key             # 按键输入例程
|   |-- 04_basic_beep_led        # 蜂鸣器和 LED 控制例程
|   |-- 05_basic_ir              # 红外遥控例程
|   |-- 06_driver_lcd            # LCD 显示例程
|   |-- 07_driver_temp_humi      # AHT10 温湿度传感器例程
|   |-- 08_driver_als_ps         # AP3216C 接近与光强传感器例程
|   |-- 09_driver_axis           # ICM20608 六轴传感器例程
|   |-- 10_component_usb_mouse   # USB 鼠标例程
|   `-- README.md
```

```
|-- rt-thread
`-- sdk-bsp-rt-spark.yaml
```

- sdk-bsp-rt-spark.yaml：描述星火 1 号的硬件信息
- libraries：板级移植文件、通用外接驱动程序、STM32F4 固件库
- projects：示例项目文件夹，包括各种示例代码
- rt-thread：rt-thread 源代码

1.2 使用

sdk-bsp-stm32f407-spark 支持 RT-Thread Studio 和 MDK 开发。

1.2.1 RT-Thread Studio 开发

1. 打开 RT-Thread Studio 的包管理器，安装 星火1 号开发板 资源包
2. 安装完成后，选择基于 BSP 创建工程即可

1.2.2 MDK 开发

为了避免 SDK 在持续更新中，每一个 `projects` 都创建一份 `rt-thread` 文件夹和 `libraries` 文件夹导致的 SDK 越来越臃肿，所以这些通用文件夹被单独提取了出来。这样就会导致直接打开 `MDK` 的工程编译会提示缺少上述两个文件夹的文件，我们使用如下步骤解决这个问题：

1. 双击某个 `project` 目录下的 `mklinks.bat` 文件，或者使用 Env 工具执行 mklink 命令，分别为 `rt-thread` 及 `libraries` 文件创建符号链接。
2. 查看目录下是否有 `rt-thread` 和 `libraries` 的文件夹图标。
3. 使用 Env 工具执行 `scons --target=mdk5` 更新 MDK5 工程文件。

1.3 交流平台

对星火 1 号感兴趣的小伙伴可以加入 QQ 群 - RT-Thread 星火学习板群号: 839583041、852752783。

第 2 章

LED 闪烁例程

2.1 简介

本例程作为 SDK 的第一个例程，也是最简单的例程，类似于程序员接触的第一个程序 Hello World 一样简洁。它的主要功能是让板载的 RGB-LED 中的红色 LED 不间断闪烁。

2.2 硬件说明

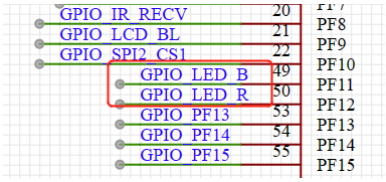

图 2.1: *LED 连接单片机引脚*

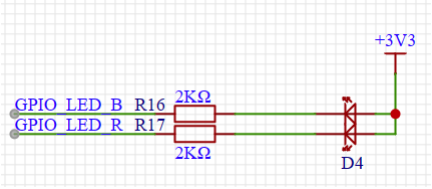

图 2.2: *LED 电路原理图*

如上图所示，RBG-LED 属于共阳 LED，阴极分别与单片机的引脚连接，其中红色 LED 对应 PF12 引脚。单片机引脚输出低电平即可点亮 LED，输出高电平则会熄灭 LED。

LED 在开发板中的位置如下图所示：

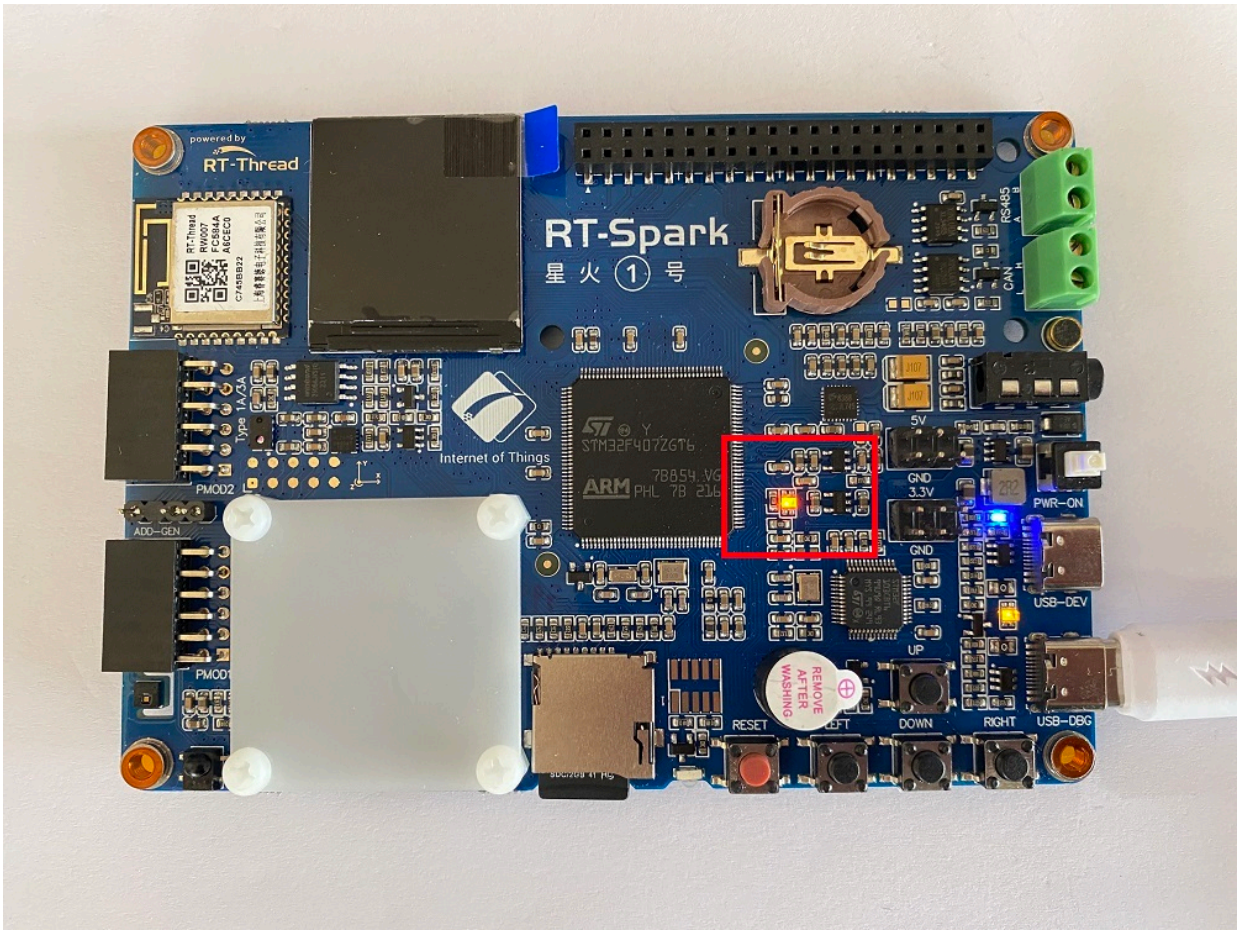

图 2.3: *LED 位置*

2.3 软件说明

闪灯的源代码位于 applications/main.c 中。首先定义了一个宏 LED_PIN ，代表闪灯的 LED 引脚编号，然后与 PIN_LED_R（PF12）对应：

```
/* 配 置 LED 灯 引 脚 */
#define LED_PIN PIN_LED_R
```

在 main 函数中，将该引脚配置为输出模式，并在下面的 while 循环中，周期性（500 毫秒）开关 LED，同时输出一些日志信息。

```
#include <rtthread.h>
#include <rtdevice.h>
#include <board.h>

#define DBG_TAG "main"
#define DBG_LVL         DBG_LOG
#include <rtdbg.h>

/* 配置 LED 灯引脚 */
#define PIN_LED_B               GET_PIN(F, 11)      // PF11 :  LED_B        --> LED
```

```
#define PIN_LED_R              GET_PIN(F, 12)      // PF12 :  LED_R        --> LED

int main(void)
{
    unsigned int count = 1;

    /* 设置 LED 引脚为输出模式 */
    rt_pin_mode(PIN_LED_R, PIN_MODE_OUTPUT);

    while (count > 0)
    {
        /* LED 灯亮 */
        rt_pin_write(PIN_LED_R, PIN_LOW);
        LOG_D("led on, count: %d", count);
        rt_thread_mdelay(500);

        /* LED 灯灭 */
        rt_pin_write(PIN_LED_R, PIN_HIGH);
        LOG_D("led off");
        rt_thread_mdelay(500);

        count++;
    }

    return 0;
}
```

2.4 运行

2.4.1 编译 & 下载

- RT-Thread Studio：在 RT-Thread Studio 的包管理器中下载 STM32F407-RT-SPARK 资源包，然后创建新工程，执行编译。
- MDK：首先双击 mklinks.bat，生成 rt-thread 与 libraries 文件夹链接；再使用 Env 生成 MDK5 工程；最后双击 project.uvprojx 打开 MDK5 工程，执行编译。

2.4.2 运行效果

按下复位按键重启开发板，观察开发板上 RBG-LED 的实际效果。正常运行后，红色 LED 会周期性闪烁，如下图所示：

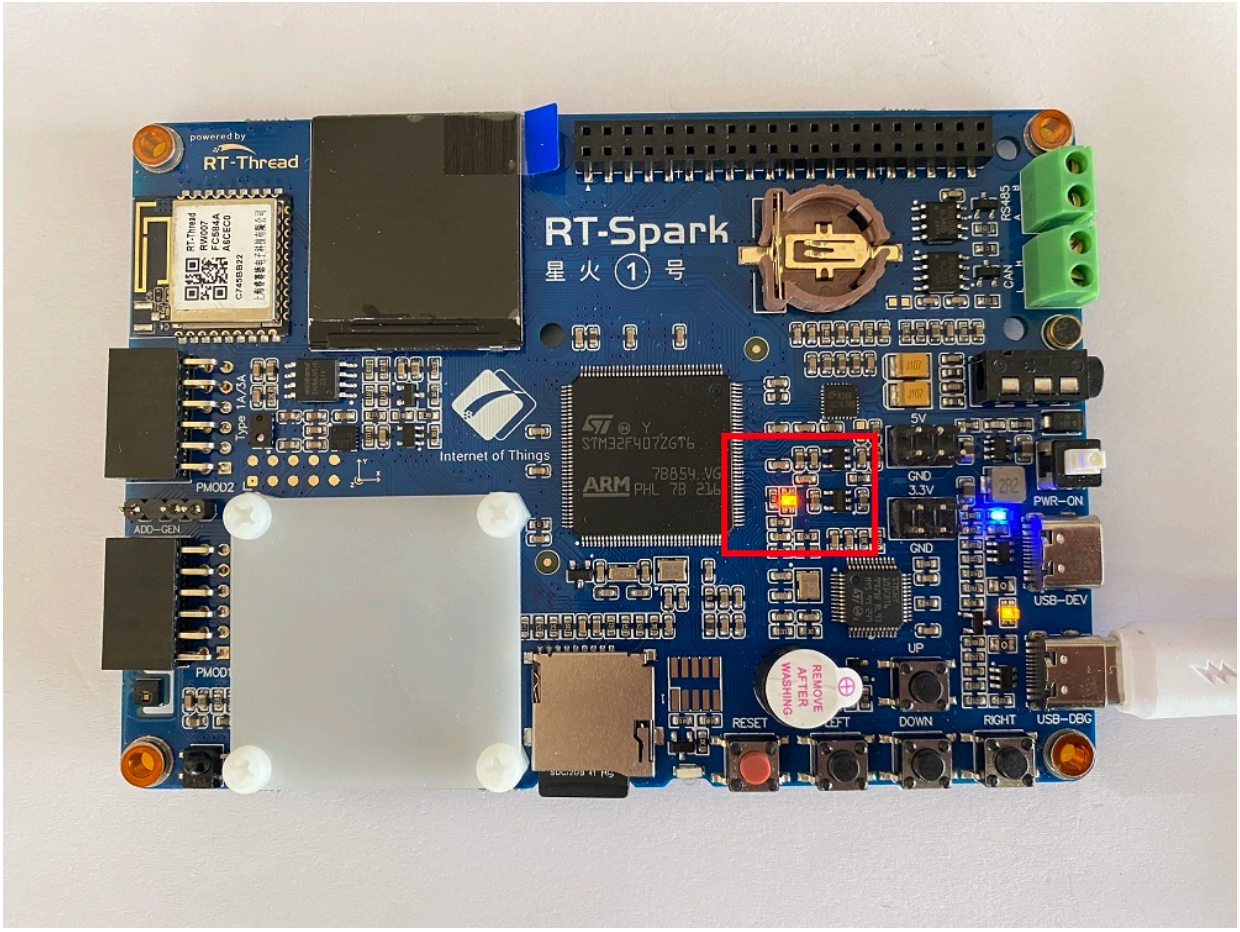

图 2.4: *红灯亮起*

此时也可以在 PC 端使用终端工具打开开发板的 ST-Link 提供的虚拟串口，设置 115200 8 1 N 。开发板的运行日志信息即可实时输出出来。

```
[D/main] led on, count: 1
[D/main] led off
[D/main] led on, count: 2
[D/main] led off
[D/main] led on, count: 3
[D/main] led off
[D/main] led on, count: 4
[D/main] led off
[D/main] led on, count: 5
[D/main] led off
[D/main] led on, count: 6
[D/main] led off
[D/main] led on, count: 7
[D/main] led off
[D/main] led on, count: 8
```

2.5 注意事项

如果想要修改 LED_PIN 宏定义，可以参考 /drivers/drv_gpio.h 文件，该文件中里有定义单片机的其它引脚编号。

2.6 引用参考

- 设备与驱动：PIN 设备

第 3 章

RGB LED 例程

3.1 简介

本例程主要功能是让板载的 RGB-LED 灯周期性地变换颜色。

3.2 硬件说明

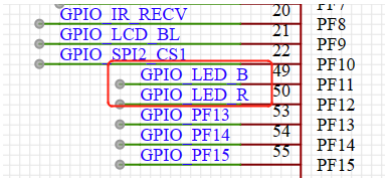

图 3.1: *RGB 连接单片机引脚*

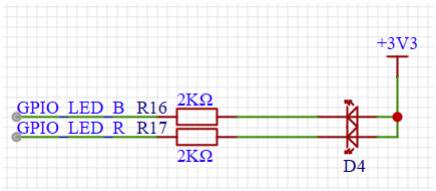

图 3.2: *RGB 电路原理图*

如上图所示，RGB-LED 属于共阳 LED，阴极分别与单片机的引脚连接，其中红色 LED 对应 PF12 号引脚，蓝色 LED 对应 PF11 号引脚。单片机对应的引脚输出低电平即可点亮对应的 LED，输出高电平则会熄灭对应的 LED。

注意：由于生产批次不同，具体 LED 灯的颜色可能会不一样，以实际开发板上颜色为准。

RGB-LED 在开发板中的位置如下图所示：

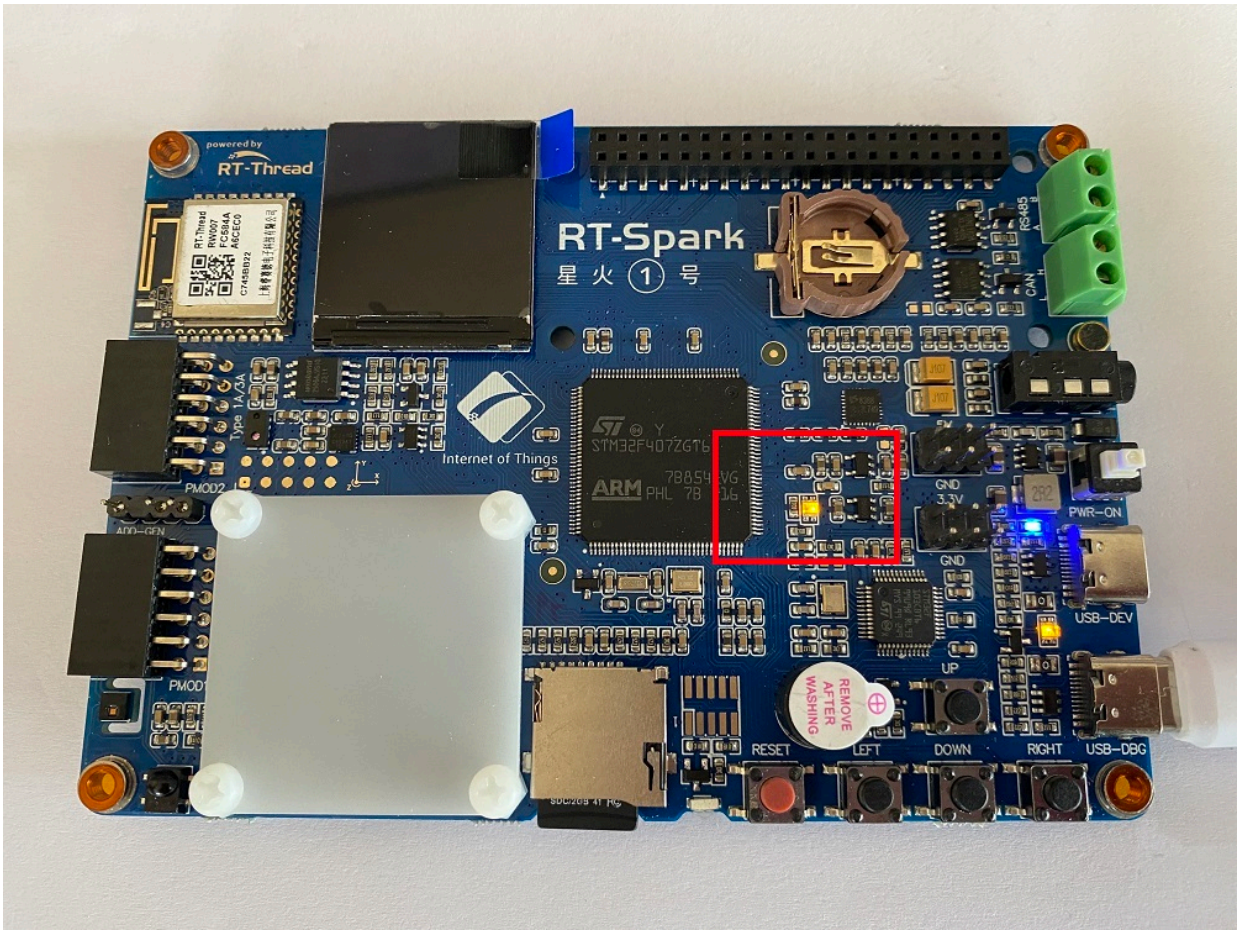

图 3.3: *RGB LED 位置*

3.3 软件说明

RGB-LED 对应的单片机引脚定义可以通过查阅头文件 /drivers/drv_gpio.h 获知。

```
/* 配置 LED 灯引脚 */
#define PIN_LED_B               GET_PIN(F, 11)      // PF11 :  LED_B        --> LED
#define PIN_LED_R               GET_PIN(F, 12)      // PF12 :  LED_R        --> LED
```

RGB-LED 灯变换的源代码位于 /projects/02_basic_rgb_led/applications/main.c 中。

```
int main(void)
{
    unsigned int count = 0;
    unsigned int group_num = sizeof(_blink_tab)/sizeof(_blink_tab[0]);
    unsigned int group_current;

    /* 设置 RGB 灯引脚为输出模式 */
    rt_pin_mode(PIN_LED_R, PIN_MODE_OUTPUT);
    rt_pin_mode(PIN_LED_B, PIN_MODE_OUTPUT);
    rt_pin_write(PIN_LED_R, LED_OFF);
    rt_pin_write(PIN_LED_B, LED_OFF);
```

```
    do
    {

        /* 获得组编号 */
        group_current = count % group_num;

        /* 控制 RGB 灯 */
        rt_pin_write(PIN_LED_R, _blink_tab[group_current][0]);
        rt_pin_write(PIN_LED_B, _blink_tab[group_current][1]);

        /* 输出 LOG 信息 */
        LOG_D("group: %d | red led [%-3.3s] | | blue led [%-3.3s]",
            group_current,
            _blink_tab[group_current][0] == LED_ON ? "ON" : "OFF",
            _blink_tab[group_current][1] == LED_ON ? "ON" : "OFF");

        count++;

        /* 延时一段时间 */
        rt_thread_mdelay(500);
    } while ( count > 0 );
    return 0;
}
```

3.4 运行

3.4.1 编译 & 下载

- RT-Thread Studio：在 RT-Thread Studio 的包管理器中下载 STM32F407-RT-SPARK 资源包，然后创建新工程，执行编译。
- MDK：首先双击 mklinks.bat，生成 rt-thread 与 libraries 文件夹链接；再使用 Env 生成 MDK5 工程；最后双击 project.uvprojx 打开 MDK5 工程，执行编译。

编译完成后，将开发板的 ST-Link USB 口与 PC 机连接，然后将固件下载至开发板。

3.4.2 运行效果

按下复位按键重启开发板，观察开发板上 RBG-LED 的实际效果。正常运行后，RGB 会周期性变化，如下图所示：

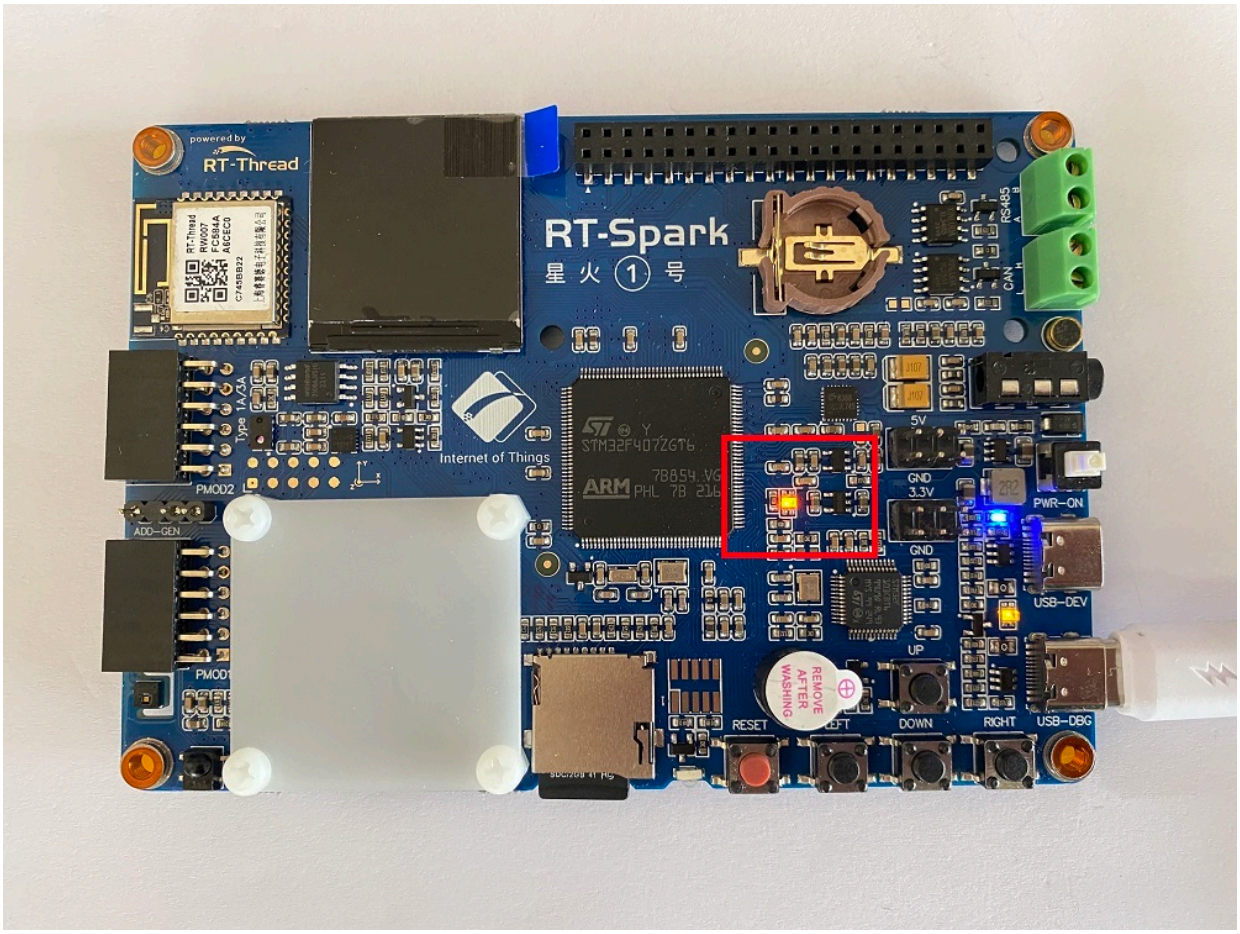

图 3.4: *红色（或其他颜色）灯运行*

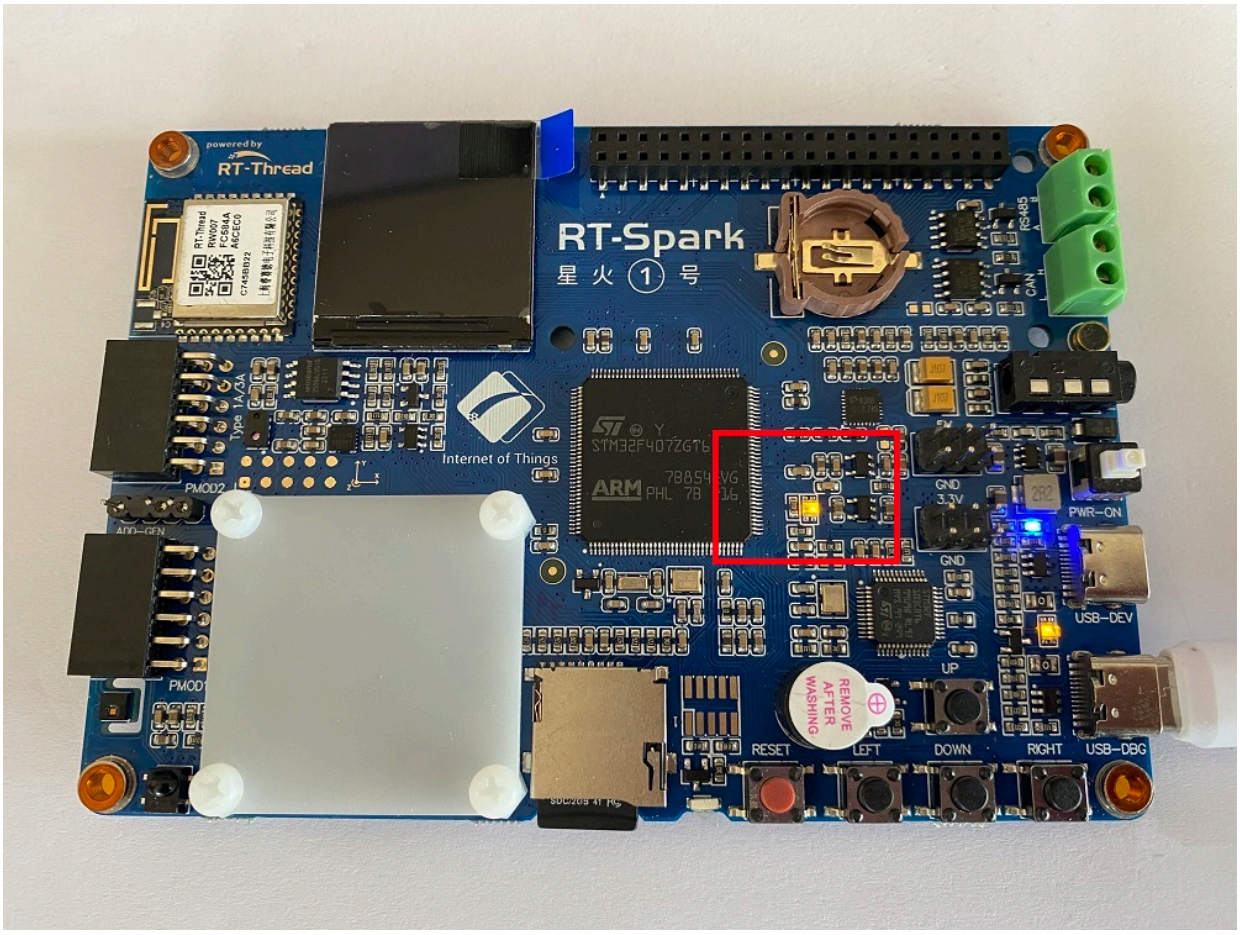

图 3.5: *蓝色（或其他颜色）灯运行*

此时也可以在 PC 端使用终端工具打开开发板的 ST-Link 提供的虚拟串口，设置 115200 8 1 N 。开发板的运行日志信息即可实时输出出来。

```
[D/main] group: 0 | red led [ON ] | | blue led [ON ]
[D/main] group: 1 | red led [OFF] | | blue led [ON ]
[D/main] group: 2 | red led [ON ] | | blue led [OFF]
[D/main] group: 3 | red led [ON ] | | blue led [ON ]
[D/main] group: 4 | red led [OFF] | | blue led [OFF]
[D/main] group: 5 | red led [ON ] | | blue led [OFF]
[D/main] group: 6 | red led [OFF] | | blue led [ON ]
[D/main] group: 7 | red led [OFF] | | blue led [OFF]
[D/main] group: 0 | red led [ON ] | | blue led [ON ]
[D/main] group: 1 | red led [OFF] | | blue led [ON ]
[D/main] group: 2 | red led [ON ] | | blue led [OFF]
[D/main] group: 3 | red led [ON ] | | blue led [ON ]
[D/main] group: 4 | red led [OFF] | | blue led [OFF]
[D/main] group: 5 | red led [ON ] | | blue led [OFF]
[D/main] group: 6 | red led [OFF] | | blue led [ON ]
[D/main] group: 7 | red led [OFF] | | blue led [OFF]
```

3.5 注意事项

暂无

3.6 引用参考

- 设备与驱动：PIN 设备

第 4 章

按键输入例程

4.1 简介

本例程主要功能是通过板载的按键 KEY0(LEFT) 控制 RGB-LED 中的红色 LED 的亮灭。

4.2 硬件说明

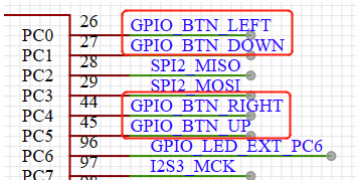

图 4.1: *连接单片机引脚*

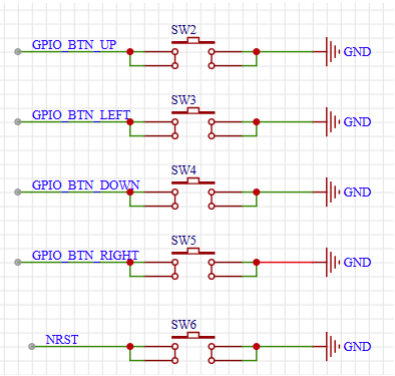

图 4.2: *电路原理图*

如上图所示，KEY0(LEFT) 引脚连接单片机 PC0(LEFT) 引脚, KEY0(LEFT) 按键按下为低电平，松开为高电平。

KEY 在开发板中的位置如下图所示：

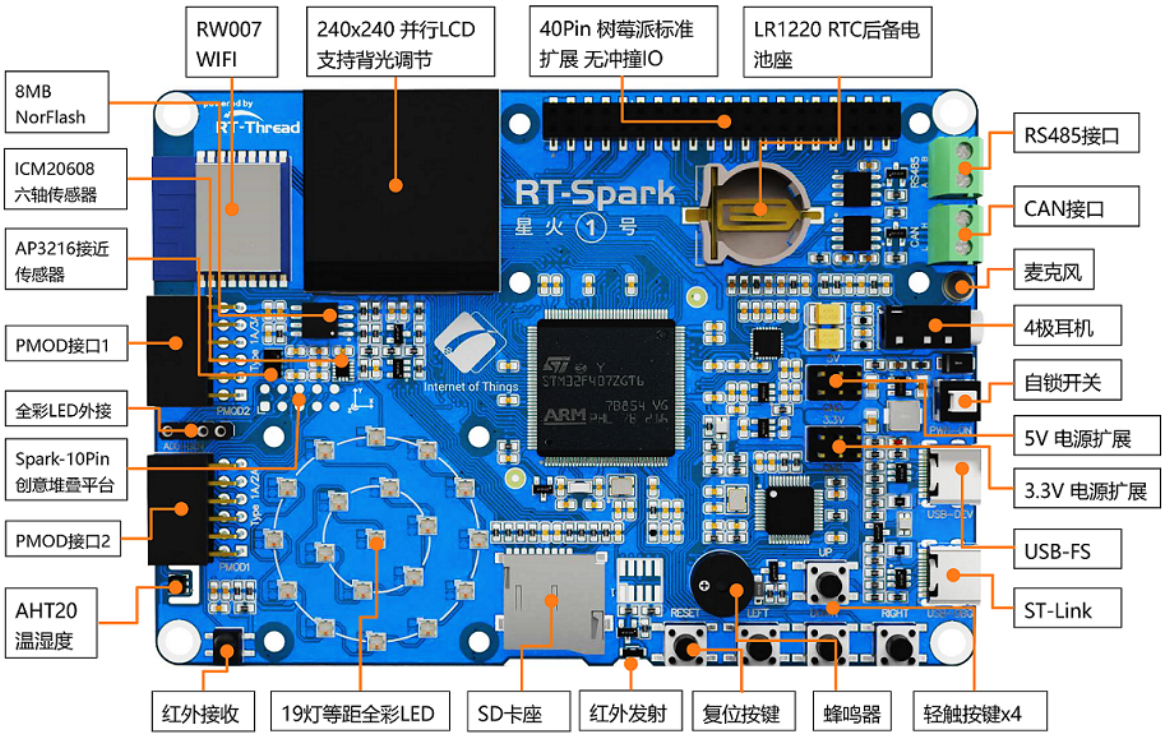

图 4.3: *按键位置*

4.3 软件说明

KEY0(LEFT) 对应的单片机引脚定义。

```
#define PIN_KEY0        GET_PIN(C, 0)     // PC0:  KEY0         --> KEY
```

按键输入的源代码位于 /projects/03_basic_key/applications/main.c 中。在 main 函数中，首先为了实验效果清晰可见，板载 RGB 红色 LED 作为 KEY0(LEFT) 的状态指示灯，设置 RGB 红灯引脚的模式为输出模式，然后设置 PIN_KEY0 引脚为输入模式，最后在 while 循环中通过 rt_pin_read(PIN_KEY0) 判断 KEY0(LEFT) 的电平状态，并作 50ms 的消抖处理，如果成功判断 KEY0(LEFT) 为低电平状态（即按键按下）则打印输出 “KEY0 pressed!” 并且指示灯亮，否则指示灯熄灭。

```
int main(void)
{
    unsigned int count = 1;

    /* 设置 RGB 红灯引脚的模式为输出模式 */
    rt_pin_mode(PIN_LED_R, PIN_MODE_OUTPUT);
    /* 设置 KEY0 引脚的模式为输入上拉模式 */
    rt_pin_mode(PIN_KEY0, PIN_MODE_INPUT_PULLUP);

    while (count > 0)
    {
```

```
        /* 读取按键 KEY0 的引脚状态 */
        if (rt_pin_read(PIN_KEY0) == PIN_LOW)
        {
            rt_thread_mdelay(50);
            if (rt_pin_read(PIN_KEY0) == PIN_LOW)
            {
                /* 按键已被按下，输出 log，点亮 LED 灯 */
                LOG_D("KEY0 pressed!");
                rt_pin_write(PIN_LED_R, PIN_LOW);
            }
        }
        else
        {
            /* 按键没被按下，熄灭 LED 灯 */
            rt_pin_write(PIN_LED_R, PIN_HIGH);
        }
        rt_thread_mdelay(10);
        count++;
    }
    return 0;
}
```

4.4 运行

4.4.1 编译 & 下载

- RT-Thread Studio：在 RT-Thread Studio 的包管理器中下载 STM32F407-RT-SPARK 资源包，然后创建新工程，执行编译。
- MDK：首先双击 mklinks.bat，生成 rt-thread 与 libraries 文件夹链接；再使用 Env 生成 MDK5 工程；最后双击 project.uvprojx 打开 MDK5 工程，执行编译。

编译完成后，将开发板的 ST-Link USB 口与 PC 机连接，然后将固件下载至开发板。

4.4.2 运行效果

按下复位按键重启开发板，按住 KEY0(LEFT) 可以观察到开发板上 RBG 红色 LED 指示灯的亮起，松开 KEY0(LEFT) 可以观察到开发板上的 RBG 红色 LED 指示灯熄灭。按住 KEY0(LEFT) 按键后如下图所示：

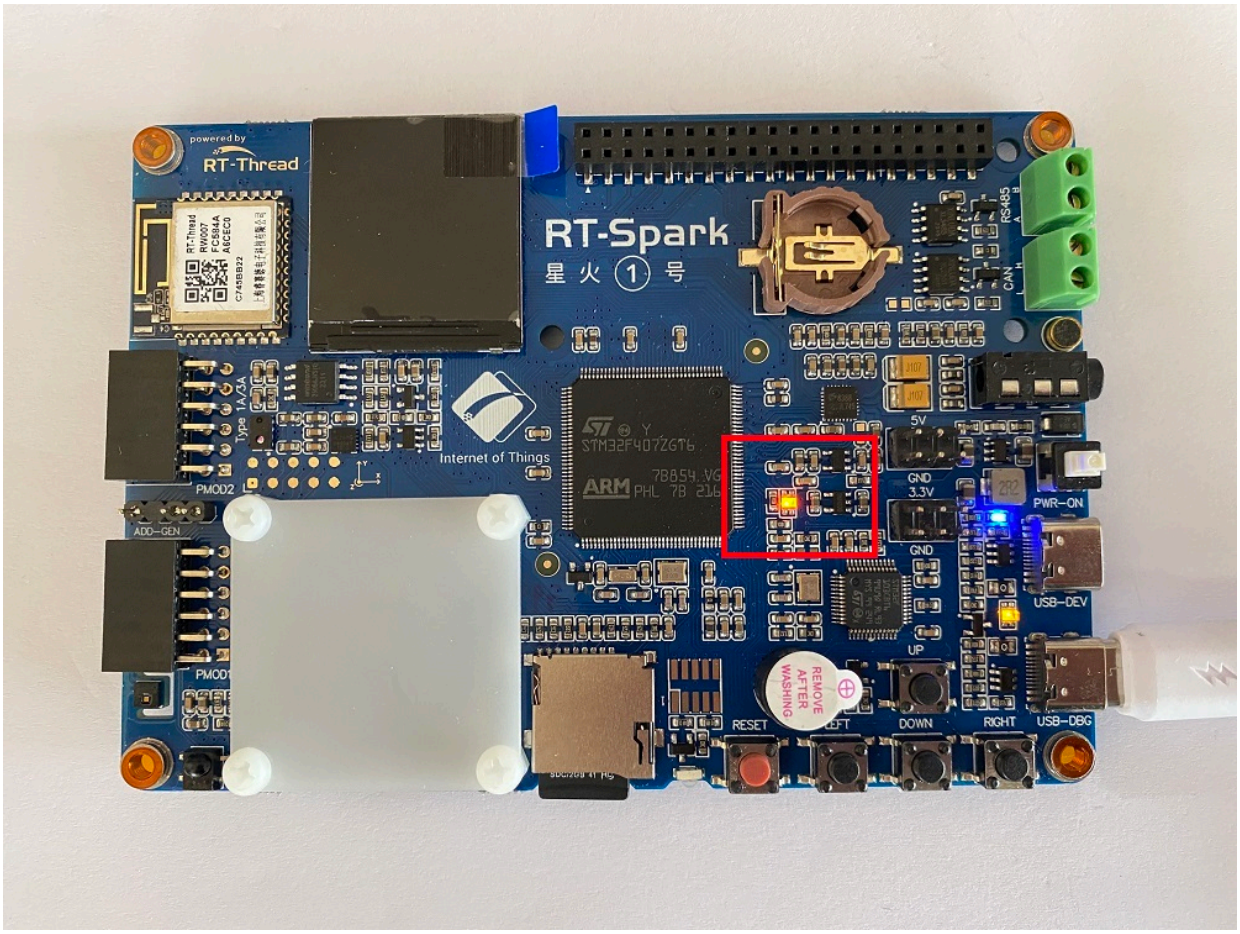

图 4.4: *按住 KEY0(LEFT) 红灯亮起*

此时也可以在 PC 端使用终端工具打开开发板的 ST-Link 提供的虚拟串口，设置 115200 8 1 N 。开发板的运行日志信息即可实时输出出来。

```
[D/main] KEY0 pressed!
[D/main] KEY0 pressed!
[D/main] KEY0 pressed!
```

4.5 注意事项

暂无。

4.6 引用参考

- 设备与驱动：PIN 设备

第 5 章

蜂鸣器和 LED 控制例程

5.1 简介

本例程主要功能为使用按键控制蜂鸣器和 led，当按下 KEY0 蓝色 LED 亮起，当按下 KEY1 后红色 LED 亮起，当按下 KEY2 后 LED 熄灭，当按住 WK_UP 时蜂鸣器鸣叫，松开 WK_UP 后蜂鸣器关闭。其中 KEY0 KEY1 KEY2 三个按键会触发中断，通过 pin 设备的中断回调函数控制 LED，WK_UP 按键通过轮询的方式控制蜂鸣器鸣叫。

5.2 硬件说明

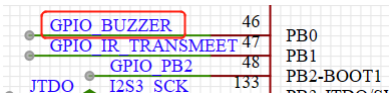

图 5.1: *蜂鸣器连接单片机引脚*

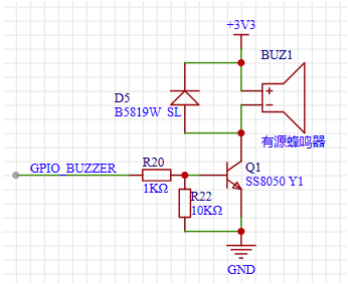

图 5.2: *蜂鸣器电路原理图*

蜂鸣器在开发板位置如下图所示：

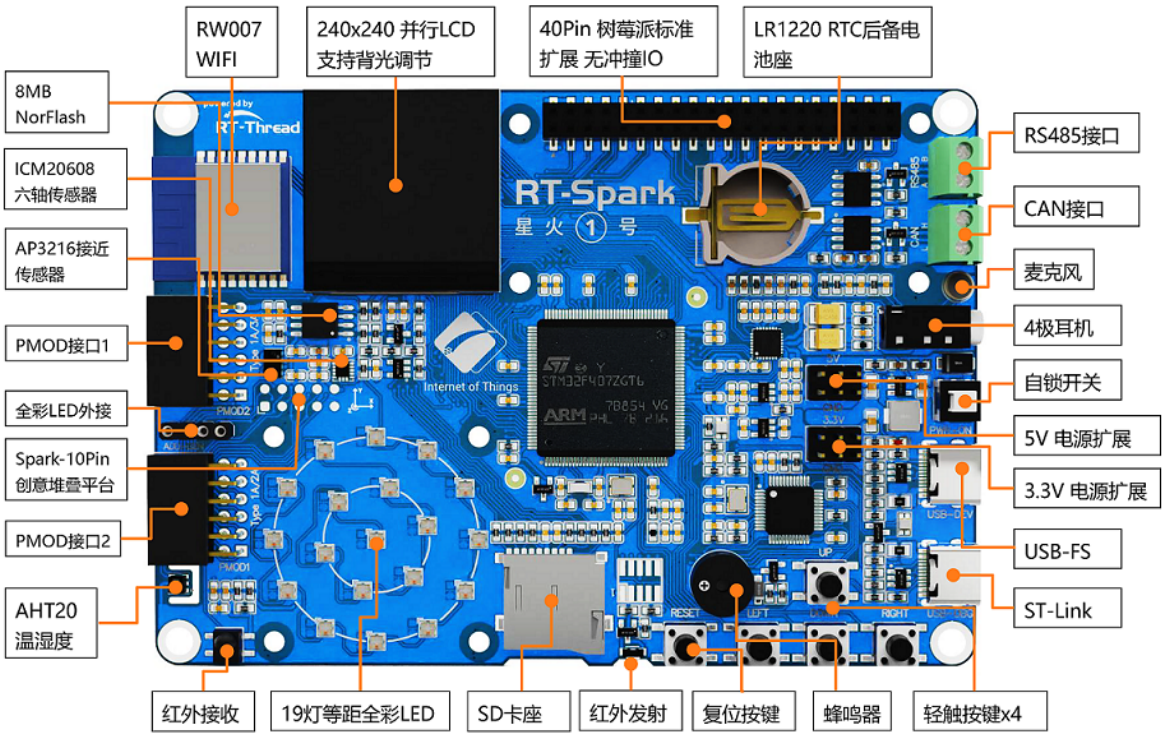

图 5.3: *蜂鸣器位置*

5.3 软件说明

按键控制蜂鸣器和 LED 的源代码位于 /projects/04_basic_beep_led/applications/main.c 中。

1. 查对应原理图可知，KEY0 、KEY1 、KEY2 按下为低电平，松开为高电平，WK_UP 按下为高电平，松开为低电平。所以在 main 函数中，首先将 KEY0 、KEY1 、KEY2 三个按键引脚配置为上拉输入模式，WK_UP 按键设置为下拉输入模式，将 PIN_LED_B、PIN_LED_R、PIN_BEEP 引脚设置为输出模式。
2. 然后使用 rt_pin_attach_irq 函数分别设置 KEY0 、KEY1 、KEY2 按键中断为下降沿触发中断并且绑定回调函数、设置回调函数相应的入参, 使用 rt_pin_irq_enable 函数使能这三个按键中断。
3. 最后在 while 循环里轮询 WK_UP 的按键状态，当成功判断 WK_UP 按键按下时调用 beep_ctrl(1) 蜂鸣器鸣叫，否则调用 beep_ctrl(0) 蜂鸣器关闭。

```
int main(void)
{
    unsigned int count = 1;

    /* 设置按键引脚为输入模式 */
    rt_pin_mode(PIN_KEY0, PIN_MODE_INPUT_PULLUP);
    rt_pin_mode(PIN_KEY1, PIN_MODE_INPUT_PULLUP);
    rt_pin_mode(PIN_KEY2, PIN_MODE_INPUT_PULLUP);
    rt_pin_mode(PIN_WK_UP, PIN_MODE_INPUT_PULLUP);
```

```
    /* 设置 LED 控制引脚为输出模式 */
    rt_pin_mode(PIN_LED_B, PIN_MODE_OUTPUT);
    rt_pin_mode(PIN_LED_R, PIN_MODE_OUTPUT);

    /* 设置蜂鸣器引脚为输出模式 */
    rt_pin_mode(PIN_BEEP, PIN_MODE_OUTPUT);

    /* 设置按键中断模式与中断回调函数 */
    rt_pin_attach_irq(PIN_KEY0, PIN_IRQ_MODE_FALLING, irq_callback, (void *)PIN_KEY0
        );
    rt_pin_attach_irq(PIN_KEY1, PIN_IRQ_MODE_FALLING, irq_callback, (void *)PIN_KEY1
        );
    rt_pin_attach_irq(PIN_KEY2, PIN_IRQ_MODE_FALLING, irq_callback, (void *)PIN_KEY2
        );

    /* 使能中断 */
    rt_pin_irq_enable(PIN_KEY0, PIN_IRQ_ENABLE);
    rt_pin_irq_enable(PIN_KEY1, PIN_IRQ_ENABLE);
    rt_pin_irq_enable(PIN_KEY2, PIN_IRQ_ENABLE);

    while (count > 0)
    {
        if (rt_pin_read(PIN_WK_UP) == PIN_LOW)
        {
            rt_thread_mdelay(50);
            if (rt_pin_read(PIN_WK_UP) == PIN_LOW)
            {
                LOG_D("WK_UP pressed. beep on.");
                beep_ctrl(1);
            }
        }
        else
        {
            beep_ctrl(0);
        }
        rt_thread_mdelay(10);
        count++;
    }
    return 0;
}
```

在中断回调函数中判断入参，根据不同入参进行相应的操作。

```
/* 中断回调 */
void irq_callback(void *args)
{
    rt_uint32_t sign = (rt_uint32_t)args;
    switch (sign)
```

```
    {
    case PIN_KEY0:
        led_ctrl(LED_BLUE);
        LOG_D("KEY0 interrupt. blue light on.");
        break;
    case PIN_KEY1:
        led_ctrl(LED_RED);
        LOG_D("KEY1 interrupt. red light on.");
        break;
    case PIN_KEY2:
        led_ctrl(LED_STOP);
        LOG_D("KEY2 interrupt. light off.");
        break;
    default:
        LOG_E("error sign= %d !", sign);
        break;
    }
}
```

5.4 运行

5.4.1 编译 & 下载

- RT-Thread Studio：在 RT-Thread Studio 的包管理器中下载 `STM32F407-RT-SPARK` 资源包，然后创建新工程，执行编译。
- MDK：首先双击 mklinks.bat，生成 rt-thread 与 libraries 文件夹链接；再使用 Env 生成 MDK5 工程；最后双击 project.uvprojx 打开 MDK5 工程，执行编译。

编译完成后，将开发板的 ST-Link USB 口与 PC 机连接，然后将固件下载至开发板。

5.4.2 运行效果

```
[D/main] KEY0 interrupt. blue light on.
[D/main] KEY0 interrupt. blue light on.
[D/main] KEY0 interrupt. blue light on.
[D/main] KEY0 interrupt. blue light on.
[D/main] KEY0 interrupt. blue light on.
[D/main] KEY1 interrupt. red light on.
[D/main] KEY1 interrupt. red light on.
[D/main] KEY1 interrupt. red light on.
[D/main] KEY1 interrupt. red light on.
[D/main] KEY1 interrupt. red light on.
[D/main] KEY1 interrupt. red light on.
[D/main] KEY1 interrupt. red light on.
[D/main] KEY1 interrupt. red light on.
```

```
[D/main] WK_UP pressed. beep on.
[D/main] WK_UP pressed. beep on.
[D/main] WK_UP pressed. beep on.
[D/main] WK_UP pressed. beep on.
[D/main] WK_UP pressed. beep on.
[D/main] WK_UP pressed. beep on.
[D/main] KEY2 interrupt. light off.
[D/main] KEY2 interrupt. light off.
[D/main] KEY2 interrupt. light off.
[D/main] KEY2 interrupt. light off.
[D/main] KEY2 interrupt. light off.
[D/main] KEY2 interrupt. light off.
```

5.5 注意事项

暂无

5.6 引用参考

- 设备与驱动：PIN 设备

第 6 章

红外遥控例程

6.1 简介

本例程主要功能是通过板载的红外接收头接收红外遥控器信号以及通过板载的红外发射头发送红外信号。

6.2 硬件说明

图 6.1: *红外接收引脚*

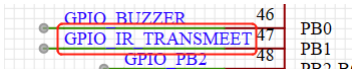

图 6.2: *红外发射连接单片机引脚*

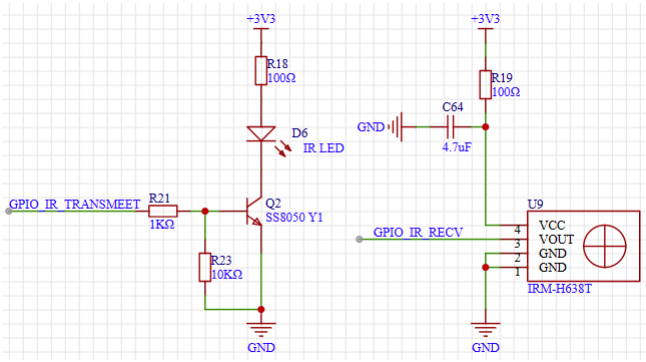

图 6.3: *红外发射接收电路原理图*

如上图所示，红外发射脚 EMISSION 接单片机引脚 PB1，红外接收头引脚 RECEPTION 接单片机引脚 PF8（TIM3_CH4 通道）。红外传感器在开发板中的位置如下图所示：

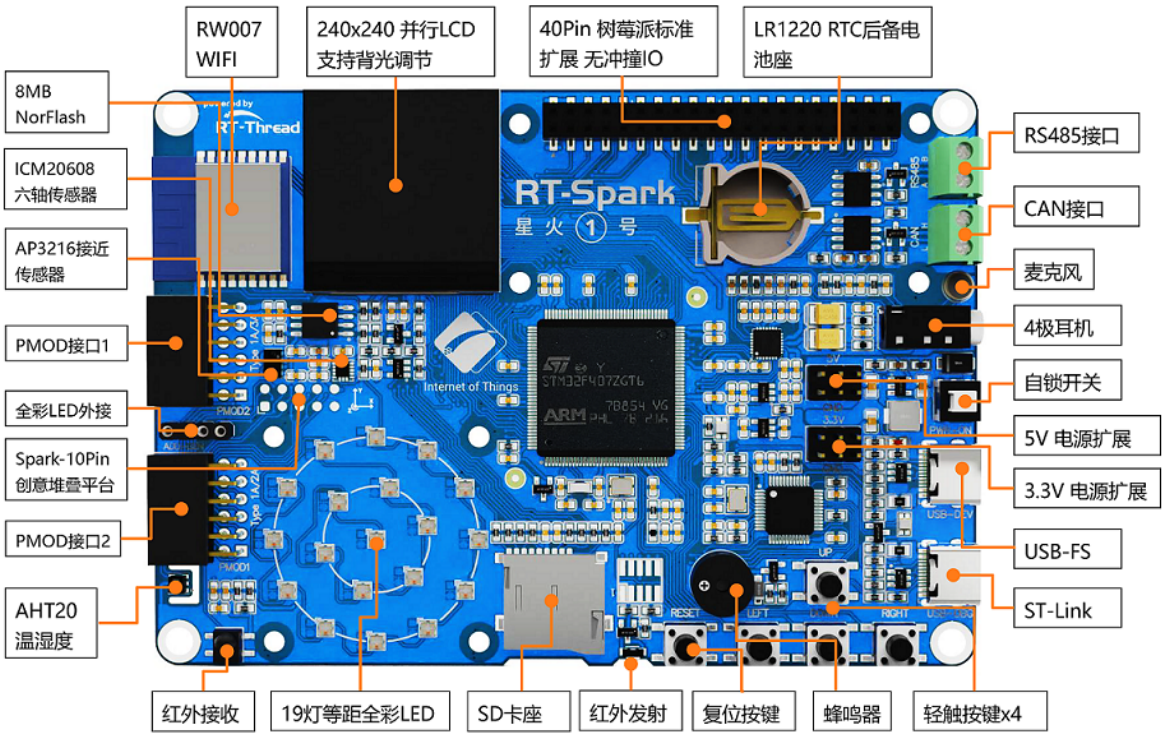

图 6.4: *红外位置*

6.3 软件说明

红外例程的示例代码位于 /projects/05_basic_ir/applications/main.c 中，主要流程：选择 NEC 解码器，初始化 GPIO 引脚。然后在 while 循环中扫描按键、打印输出接收到的红外数据，当 KEY0 按下后将会把最近一次接收到的红外数据通过红外发射头发送出去。

```
int main(void)
{
    unsigned int count = 1;
    rt_int16_t key;
    struct infrared_decoder_data infrared_data;

    /* 选择 NEC 解码器 */
    ir_select_decoder("nec");

    /* 设置按键引脚为输入模式 */
    rt_pin_mode(PIN_KEY0, PIN_MODE_INPUT_PULLUP);

    /* 设置 RGB 引脚为输出模式 */
    rt_pin_mode(PIN_LED_R, PIN_MODE_OUTPUT);
    rt_pin_mode(PIN_LED_B, PIN_MODE_OUTPUT);

    rt_pin_write(PIN_LED_R, PIN_HIGH);
```

```
    rt_pin_write(PIN_LED_B, PIN_HIGH);

    while (count > 0)
    {
        /* 按键扫描 */
        key = key_scan();
        if(key == PIN_KEY0)
        {
            /* 有按键按下，蓝灯亮起 */
            rt_pin_write(PIN_LED_B, PIN_LOW);
            infrared_data.data.nec.repeat = 0;
            /* 发送红外数据 */
            infrared_write("nec",&infrared_data);
            rt_thread_mdelay(200);
            LOG_I("SEND    OK: addr:0x%02X key:0x%02X repeat:%d",
                infrared_data.data.nec.addr, infrared_data.data.nec.key,
                    infrared_data.data.nec.repeat);
        }
        else if(infrared_read("nec",&infrared_data) == RT_EOK)
        {
            /* 读取到红外数据，红灯亮起 */
            rt_pin_write(PIN_LED_R, PIN_LOW);
            LOG_I("RECEIVE OK: addr:0x%02X key:0x%02X repeat:%d",
                infrared_data.data.nec.addr, infrared_data.data.nec.key,
                    infrared_data.data.nec.repeat);
        }
        rt_thread_mdelay(10);

        /* 熄灭蓝灯 */
        rt_pin_write(PIN_LED_B, PIN_HIGH);
        /* 熄灭红灯 */
        rt_pin_write(PIN_LED_R, PIN_HIGH);
        count++;
    }
    return 0;
}
```

本例程的实现主要使用了红外软件包的以下三个函数：

```
/* 选 择 解 码 器 */
rt_err_t ir_select_decoder(const char* name);
/* 读 取 红 外 数 据。 */
rt_err_t infrared_read(const char* decoder_name,struct infrared_decoder_data* data);
/* 接 收 红 外 数 据。 */
rt_err_t infrared_write(const char* decoder_name, struct infrared_decoder_data* data
    );
```

6.4 运行

6.4.1 编译 & 下载

- RT-Thread Studio：在 RT-Thread Studio 的包管理器中下载 STM32F407-RT-SPARK 资源包，然后创建新工程，执行编译。
- MDK：首先双击 mklinks.bat，生成 rt-thread 与 libraries 文件夹链接；再使用 Env 生成 MDK5 工程；最后双击 project.uvprojx 打开 MDK5 工程，执行编译。

编译完成后，将开发板的 ST-Link USB 口与 PC 机连接，然后将固件下载至开发板。

6.4.2 运行效果

按下复位按键重启开发板，观察板载 RGB 灯，用户可以使用红外遥控器对准板载红外接收头发送红外信号。在接收红外信号的时候，RGB 红灯闪烁。按下 KEY0 键，开发板将会通过红外发射头发送最近一次接收到的红外数据，发送红外数据时 RGB 蓝灯亮起。

此时也可以在 PC 端使用终端工具打开开发板的 ST-Link 提供的虚拟串口，设置波率：115200 ，数据位：8 ，停止位：1 N 。开发板的运行日志信息即可实时输出来。

```
[I/main] RECEIVE OK: addr:0x00 key:0x18 repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0x18 repeat:1
[I/main] RECEIVE OK: addr:0x00 key:0x18 repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0x18 repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0x98 repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0x98 repeat:1
[I/main] RECEIVE OK: addr:0x00 key:0x5A repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0x5A repeat:1
[I/main] RECEIVE OK: addr:0x00 key:0x7A repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0xB0 repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0x42 repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0xA8 repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0xA8 repeat:1
[I/main] RECEIVE OK: addr:0x00 key:0xE0 repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0x68 repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0x30 repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0x02 repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0x02 repeat:1
[I/main] RECEIVE OK: addr:0x00 key:0x02 repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0x42 repeat:0
[I/main] SEND    OK: addr:0x00 key:0x42 repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0x42 repeat:0
[I/main] SEND    OK: addr:0x00 key:0x42 repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0x42 repeat:0
[I/main] SEND    OK: addr:0x00 key:0x42 repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0x42 repeat:0
[I/main] SEND    OK: addr:0x00 key:0x42 repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0x42 repeat:0
```

```
[I/main] SEND    OK: addr:0x00 key:0x42 repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0x42 repeat:0
[I/main] SEND    OK: addr:0x00 key:0x42 repeat:0
[I/main] RECEIVE OK: addr:0x00 key:0x42 repeat:0
```

6.5 注意事项

请使用 38KHZ 载波的红外遥控器实验。

6.6 引用参考

- 设备与驱动：PIN 设备
- 红外框架：https://github.com/RT-Thread-packages/infrared_framework

第 7 章

LCD 显示例程

7.1 简介

本例程主要介绍了如何在 LCD 上显示文字和图片。

7.2 硬件说明

星火 1 号开发板板载的是一块 1.3 寸，分辨率为 240x240 的 LCD 显示屏，显示效果十分细腻。显示屏的驱动芯片是 ST7789 v3, 通信接口使用的是 8080 并口，通过 fsmc 模拟出驱动时序和单片机进行通讯。使用了 8 根数据线传输数据，一根地址选择线作为芯片的使能信号。

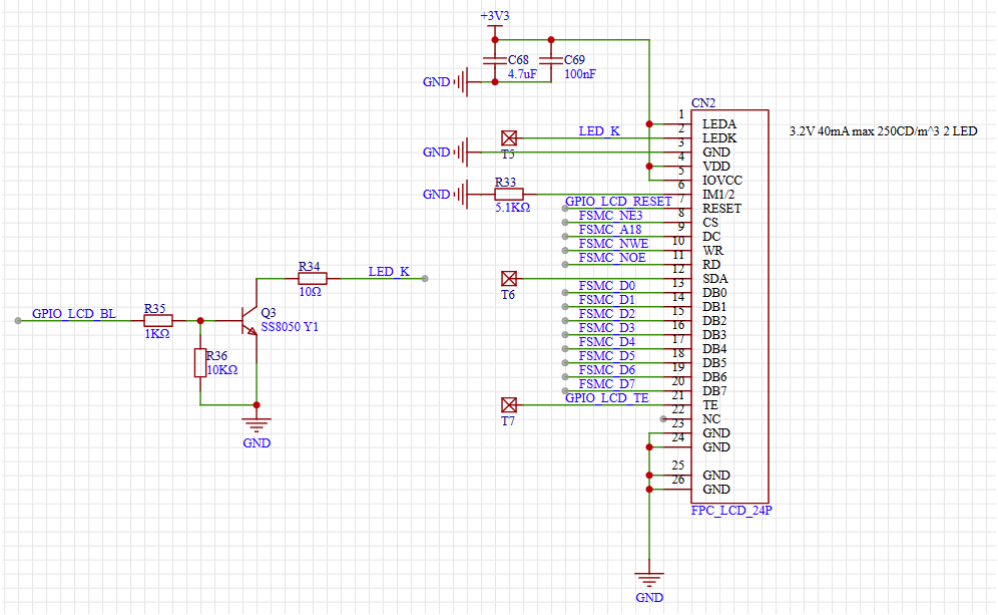

图 7.1: *LCD 原理图*

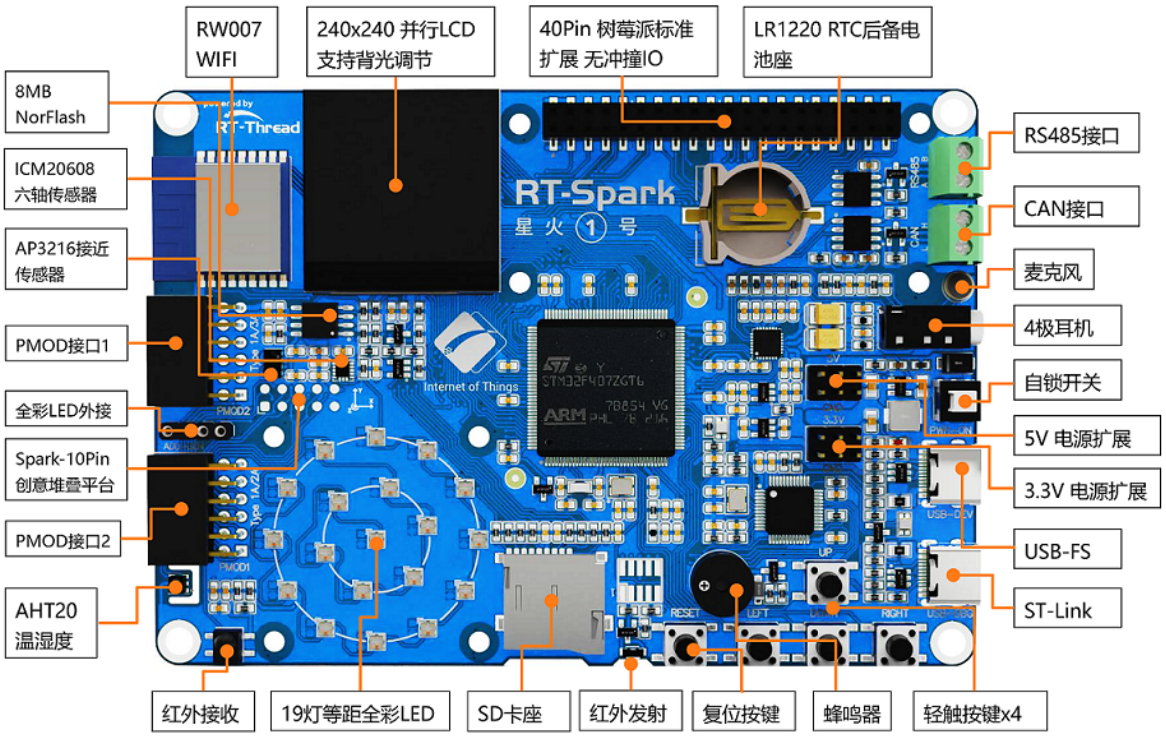

图 7.2: *LCD 实物图*

7.3 软件说明

显示图片和文字的源代码位于 libraries/Board_Drivers/lcd/drv_lcd.c 中。

在 main 函数中，通过调用已经封装好的 LCD API 函数，首先执行的是清屏操作，将 LCD 全部刷成白色。然后设置画笔的颜色为黑色，背景色为白色。接着显示 RT-Thread 的 LOGO。最后会显示一些信息，包括 16x16 像素，24x24 像素和 32x32 像素的三行英文字符，一条横线和一个同心圆。

```
int main(void)
{
    lcd_clear(WHITE);

    /* show RT-Thread logo */
    lcd_show_image(0, 0, 240, 69, image_rttlogo);

    /* set the background color and foreground color */
    lcd_set_color(WHITE, BLACK);

    /* show some string on lcd */
    lcd_show_string(10, 69, 16, "Hello, RT-Thread!");
    lcd_show_string(10, 69 + 16, 24, "RT-Thread");
    lcd_show_string(10, 69 + 16 + 24, 32, "RT-Thread");

    /* draw a line on lcd */
```

```
    lcd_draw_line(0, 69 + 16 + 24 + 32, 240, 69 + 16 + 24 + 32);

    /* draw a concentric circles */
    lcd_draw_point(120, 194);
    for (int i = 0; i < 46; i += 4)
    {
        lcd_draw_circle(120, 194, i);
    }
    return 0;
}
```

7.4 运行

7.4.1 编译 & 下载

- RT-Thread Studio：在 RT-Thread Studio 的包管理器中下载 STM32F407-RT-SPARK 资源包，然后创建新工程，执行编译。
- MDK：首先双击 mklinks.bat，生成 rt-thread 与 libraries 文件夹链接；再使用 Env 生成 MDK5 工程；最后双击 project.uvprojx 打开 MDK5 工程，执行编译。

编译完成后，将开发板的 ST-Link USB 口与 PC 机连接，然后将固件下载至开发板。

7.4.2 运行效果

按下复位按键重启开发板，观察开发板上 LCD 的实际效果。正常运行后，LCD 上会显示 RT-Thread LOGO，下面会显示 3 行大小为 16、24、32 像素的文字，文字下面是一行直线，直线的下方是一个同心圆。如下图所示：

图 7.3: *lcd 显示图案*

7.5 注意事项

屏幕的分辨率是 240x240，输入位置参数时要注意小于 240，不然会出现无法显示的现象。图像的取模方式为自上而下，自左向右，高位在前，16 位色（RGB-565）。本例程未添加中文字库，不支持显示中文。

7.6 引用参考

- 设备与驱动：SPI 设备

第 8 章

AHT10 温湿度传感器例程

8.1 简介

本例程主要功能是利用 RT-Thread 的 AHT10 软件包的读取传感器 aht10(aht21) 所测量的温度（temperature）与湿度（humidity）。因为两款传感器的驱动互相兼容，所以下文所说的 aht10 指的是我们板载的 aht21。

8.2 AHT10 软件包简介

AHT10 软件包提供了使用温度与湿度传感器 aht10 基本功能，并且提供了软件平均数滤波器可选功能，如需详细了解该软件包，请参考 AHT10 软件包中的 README。

8.3 硬件说明

aht10 硬件原理图如下所示：

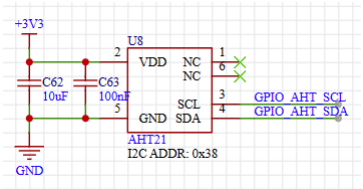

图 8.1: *温湿度传感器模块电路*

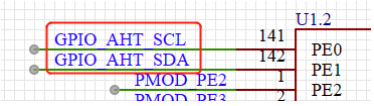

图 8.2: *温湿度传感器与芯片连接图*

如上图所示，单片机通过软件 iic I2C3(soft) scl(PE0)、I2C3(soft) sda(PE1) 对传感器 aht10 发送命令、读取数据等。温度与湿度传感器在开发板中的位置如下图所示：

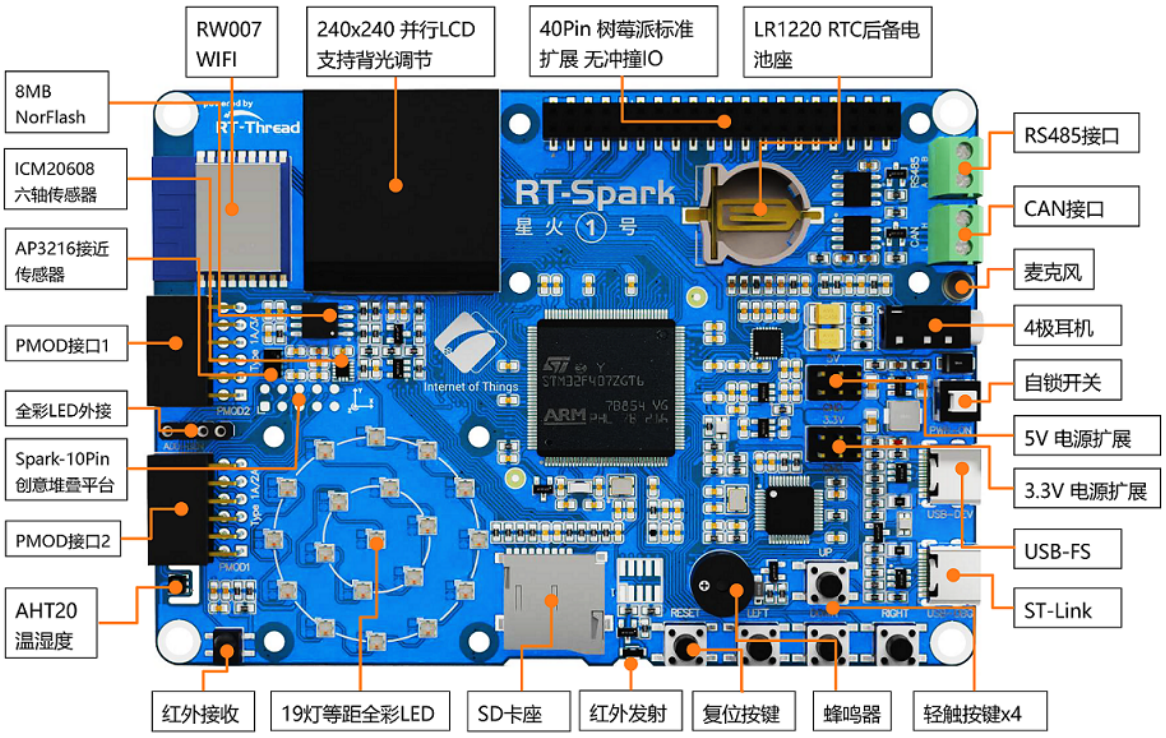

图 8.3: *温湿度传感器位置*

该传感器输入电压范围为 1.8v - 3.3v，测量温度与湿度的量程、精度如下表所示：

功能	量程	精度	单位
温度	-40 - 85	±0.5	摄氏度
相对湿度	0 - 100	±3	%

8.4 软件说明

温度与湿度传感器的示例代码位于 applications/main.c 中，主要流程：初始化传感器 aht10，传入参数 i2c2 为该传感器挂载的 i2c 总线的名称；初始化若失败，则返回空，程序不会被执行，若成功，则返回传感器设备对象；然后将返回的设备对象分别传入读取湿度与温度的函数，获取测量的湿度与温度值（详细的 API 介绍参考 aht10 软件包读取温度与湿度章节，源码参考 aht10.c）。示例代码如下：

```
int main(void)
{
    float humidity, temperature;
    aht10_device_t dev;

    /* 总线名称 */
    const char *i2c_bus_name = "i2c3";
    int count = 0;
```

```
    /* 等待传感器正常工作 */
    rt_thread_mdelay(2000);

    /* 初始化 aht10 */
    dev = aht10_init(i2c_bus_name);
    if (dev == RT_NULL)
    {
        LOG_E("The sensor initializes failure");
        return 0;
    }

    while (count++ < 100)
    {
        /* 读取湿度 */
        humidity = aht10_read_humidity(dev);
        LOG_D("humidity   : %d.%d %%", (int)humidity, (int)(humidity * 10) % 10);

        /* 读取温度 */
        temperature = aht10_read_temperature(dev);
        LOG_D("temperature: %d.%d", (int)temperature, (int)(temperature * 10) % 10);

        rt_thread_mdelay(1000);
    }
    return 0;
}
```

8.5 运行

8.5.1 编译 & 下载

- RT-Thread Studio：在 RT-Thread Studio 的包管理器中下载 STM32F407-RT-SPARK 资源包，然后创建新工程，执行编译。
- MDK：首先双击 mklinks.bat，生成 rt-thread 与 libraries 文件夹链接；再使用 Env 生成 MDK5 工程；最后双击 project.uvprojx 打开 MDK5 工程，执行编译。

编译完成后，将开发板的 ST-Link USB 口与 PC 机连接，然后将固件下载至开发板。

8.5.2 运行效果

烧录完成后，此时可以在 PC 端使用终端工具打开开发板的 ST-Link 提供的虚拟串口，设置串口波特率为 115200，数据位 8 位，停止位 1 位，无流控，开发板的运行日志信息即可实时输出出来，显示如下所示：

```
 \ | /
- RT -     Thread Operating System
 / | \     4.1.1 build Jun  9 2023 13:18:28
```

```
 2006 - 2022 Copyright by RT-Thread team
msh >[D/main] humidity   : 0.0 %
[D/main] temperature: -50.0
[D/main] humidity   : 58.8 %
[D/main] temperature: 26.2
[D/main] humidity   : 58.7 %
[D/main] temperature: 26.3
[D/main] humidity   : 58.6 %
[D/main] temperature: 26.3
[D/main] humidity   : 58.5 %
[D/main] temperature: 26.3
[D/main] humidity   : 58.5 %
[D/main] temperature: 26.3
[D/main] humidity   : 58.5 %
[D/main] temperature: 26.3
[D/main] humidity   : 58.4 %
[D/main] temperature: 26.3
[D/main] humidity   : 58.3 %
[D/main] temperature: 26.3
[D/main] humidity   : 58.4 %
[D/main] temperature: 26.3
[D/main] humidity   : 58.2 %
[D/main] temperature: 26.3
[D/main] humidity   : 58.1 %
[D/main] temperature: 26.3
[D/main] humidity   : 58.1 %
[D/main] temperature: 26.3
[D/main] humidity   : 57.9 %
[D/main] temperature: 26.4
[D/main] humidity   : 57.7 %
[D/main] temperature: 26.4
[D/main] humidity   : 57.4 %
[D/main] temperature: 26.3
[D/main] humidity   : 57.3 %
[D/main] temperature: 26.4
[D/main] humidity   : 57.1 %
[D/main] temperature: 26.3
[D/main] humidity   : 57.0 %
[D/main] temperature: 26.4
[D/main] humidity   : 57.0 %
[D/main] temperature: 26.4
```

8.6 注意事项

暂无。

8.7 引用参考

- 设备与驱动：I2C 设备
- aht10 软件包：https://github.com/RT-Thread-packages/aht10

第 9 章

AP3216C 接近与光强传感器例程

9.1 简介

本例程主要功能是利用 RT-Thread 的 AP3216C 软件包读取传感器 ap3216c 测量的接近感应（ps，proximity sensor）与光照强度（als，ambient light sensor）。

9.2 AP3216C 软件包简介

AP3216C 软件包提供了使用接近感应（ps）与光照强度（als）传感器 ap3216c 基本功能，并且提供了硬件中断的可选功能，如需详细了解该软件包，请参考 AP3216C 软件包中的 README。

9.3 硬件说明

ap3216c 硬件原理图如下所示：

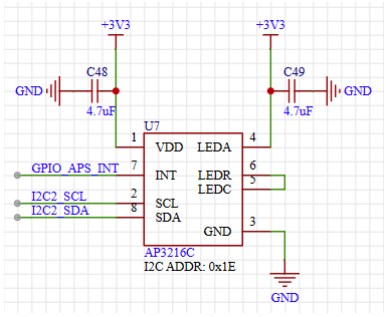

图 9.1: *接近与光强传感器模块原理图*

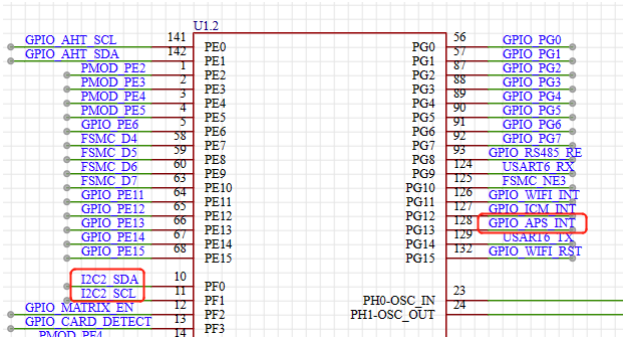

图 9.2: *接近与光强传感器模块与芯片连接图*

如上图所示，单片机通过 I2C2(soft) scl(PF1)、I2C2(soft) sda(PF0) 对传感器 ap3216c 发送命令、读取数据等，AP_INT(PG13) 为硬件中断引脚。

接近感应与光照强度传感器在开发板中的位置如下图所示：

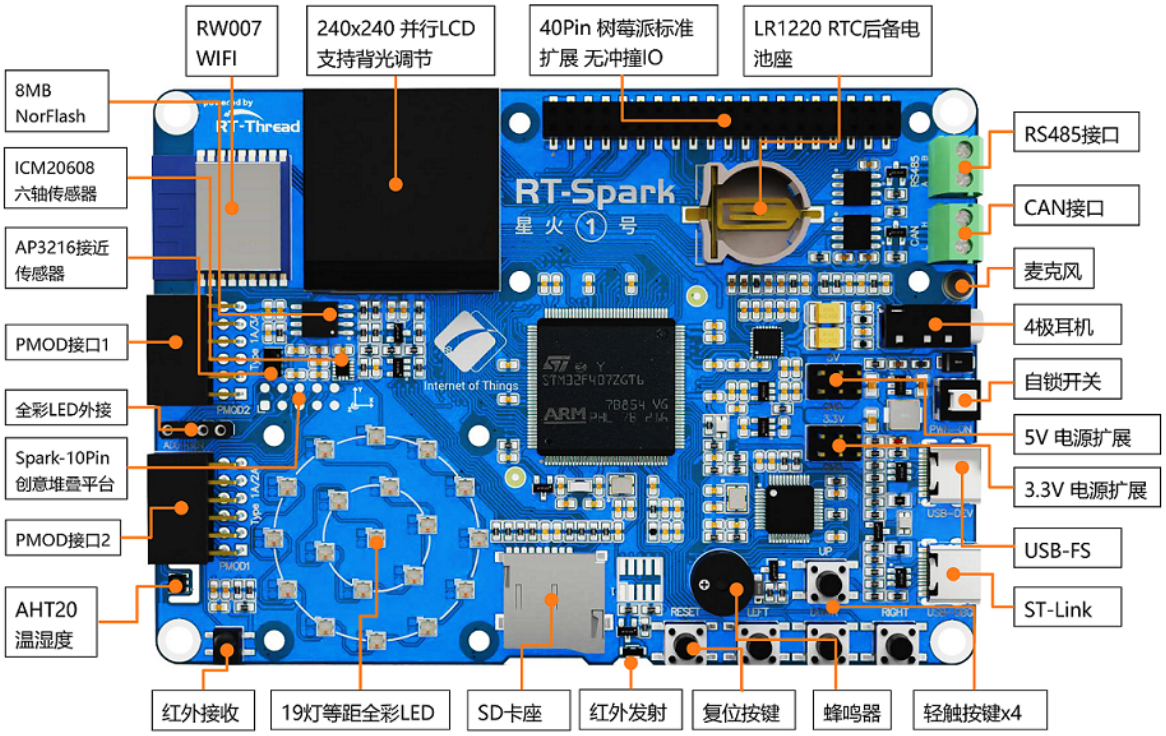

图 9.3: *接近与光强传感器位置*

该传感器能够实现如下功能：

- 光照强度：支持 4 个量程
- 接近感应：支持 4 种增益
- 中断触发：光照强度及接近感应同时支持高于阈值或低于阈值的两种硬件中断触发方式

9.4 软件说明

接近感应与光照强度传感器 ap3216c 的示例代码位于 applications/main.c 中，主要流程：初始化传感器 ap3216c，传入参数 i2c1 为该传感器挂载的 i2c 总线的名称；初始化若失败，则返回空，程序不会被执行，若成功，则返回传感器设备对象；然后将返回设备对象分别传入获取 als 与 ps 函数，获取测量的 als 与 ps 值（详细的 API 介绍参考 ap3216c 软件包读取接近感应与光照强度章节，源码参考 ap3216c.c）。示例代码如下：

```
int main(void)
{
    ap3216c_device_t dev;
    const char *i2c_bus_name = "i2c2";
    int count = 0;

    /* 初始化 ap3216c */
    dev = ap3216c_init(i2c_bus_name);
    if (dev == RT_NULL)
    {
        LOG_E("The sensor initializes failure.");
        return 0;
    }

    while (count++ < 100)
    {
        rt_uint16_t ps_data;
        float brightness;

        /* 读接近感应值 */
        ps_data = ap3216c_read_ps_data(dev);
        if (ps_data == 0)
        {
            LOG_D("object is not proximity of sensor.");
        }
        else
        {
            LOG_D("current ps data   : %d.", ps_data);
        }

        /* 读光照强度值 */
        brightness = ap3216c_read_ambient_light(dev);
        LOG_D("current brightness: %d.%d(lux).", (int)brightness, ((int)(10 *
            brightness) % 10));

        rt_thread_mdelay(1000);
    }
    return 0;
}
```

9.4.1 编译 & 下载

- RT-Thread Studio：在 RT-Thread Studio 的包管理器中下载 STM32F407-RT-SPARK 资源包，然后创建新工程，执行编译。
- MDK：首先双击 mklinks.bat，生成 rt-thread 与 libraries 文件夹链接；再使用 Env 生成 MDK5 工程；最后双击 project.uvprojx 打开 MDK5 工程，执行编译。

编译完成后，将开发板的 ST-Link USB 口与 PC 机连接，然后将固件下载至开发板。

9.4.2 运行效果

烧录完成后，此时可以在 PC 端使用终端工具打开开发板的 ST-Link 提供的虚拟串口，设置串口波特率为 115200，数据位 8 位，停止位 1 位，无流控，开发板的运行日志信息即可实时输出出来，显示如下所示：

```
 \ | /
- RT -     Thread Operating System
 / | \     4.1.1 build Jun  9 2023 13:18:36
 2006 - 2022 Copyright by RT-Thread team
msh >[D/main] object is not proximity of sensor.
[D/main] current brightness: 32.9(lux).
[D/main] object is not proximity of sensor.
[D/main] current brightness: 33.6(lux).
[D/main] current ps data   : 7.
[D/main] current brightness: 33.9(lux).
[D/main] object is not proximity of sensor.
[D/main] current brightness: 33.9(lux).
[D/main] current ps data   : 7.
[D/main] current brightness: 33.9(lux).
[D/main] current ps data   : 11.
[D/main] current brightness: 33.9(lux).
[D/main] current ps data   : 1.
[D/main] current brightness: 33.9(lux).
[D/main] object is not proximity of sensor.
[D/main] current brightness: 33.9(lux).
[D/main] object is not proximity of sensor.
[D/main] current brightness: 33.9(lux).
[D/main] current ps data   : 1023.
[D/main] current brightness: 0.0(lux).
[D/main] current ps data   : 1023.
[D/main] current brightness: 0.0(lux).
[D/main] object is not proximity of sensor.
[D/main] current brightness: 32.9(lux).
[D/main] object is not proximity of sensor.
[D/main] current brightness: 33.6(lux).
[D/main] current ps data   : 1.
[D/main] current brightness: 33.6(lux).
[D/main] current ps data   : 8.
[D/main] current brightness: 33.6(lux).
```

```
[D/main] object is not proximity of sensor.
[D/main] current brightness: 33.6(lux).
```

9.5 注意事项

暂无。

9.6 引用参考

- 设备与驱动：I2C 设备
- ap3216c 软件包：https://github.com/RT-Thread-packages/ap3216c

第 10 章

ICM20608 六轴传感器例程

10.1 简介

本例程主要功能是利用 RT-Thread 的 ICM20608 软件包读取传感器 icm20608 所测量的三轴加速度（three accelerate）、三轴陀螺仪（three gyroscope）。

10.2 ICM20608 软件包简介

ICM20608 软件包是 RT-Thread 针对六轴传感器 icm20608 功能使用的实现，使用这个软件包，可以让该传感器在 RT-Thread 上非常方便使用 icm20608 的基本功能，包括读取三轴加速度（3-axisaccelerometer）、三轴陀螺仪（3-axis gyroscope）、零值校准等功能，如需详细了解该软件包，请参考 ICM20608 软件包中的 README。

10.3 硬件说明

icm20608 硬件原理图如下所示：

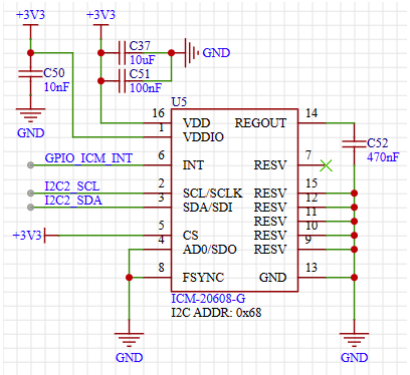

图 10.1: *六轴传感器原理图*

图 **10.2:** *六轴传感器与芯片连接图*

如上图所示，单片机通过 I2C2(soft) scl(PF1)、I2C2(soft) sda(PF0) 对传感器 icm20608 发送命令、读取数据等，ICM_INT(PG12) 为硬件中断引脚。

六轴传感器在开发板中的位置如下图所示：

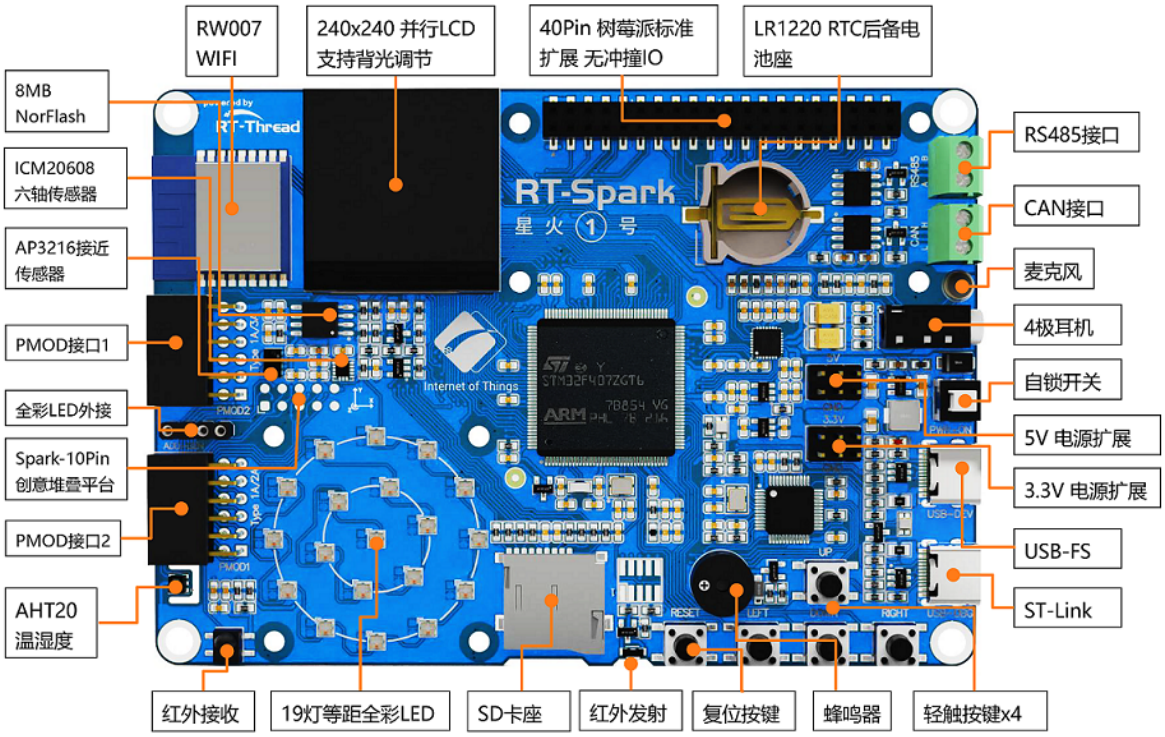

图 **10.3:** *六轴传感器位置*

该传感器能够实现如下功能：

- 支持 4 种三轴加速度量程
- 支持 4 种三轴陀螺仪量程
- 支持零值校准

10.4 软件说明

六轴传感器 icm20608 的示例代码位于 applications/main.c 中，主要流程：初始化传感器 -> 零值校准 -> 读取三轴加速度与三轴陀螺仪，分别展开如下所述：

1. 初始化传感器

初始化函数 icm20608_init 传入的参数 i2c_bus_name 为该传感器挂载的 i2c 总线名称，进行初始化；初始化若失败，则返回空，若成功，则返回六轴传感器的设备对象 dev。

2. 零值校准

首先在进行零值校准时，x 轴、y 轴应处于水平状态，且传感器处于静态；其次使用零值校准函数 icm20608_calib_level 进行零值校准时，传入设备对象 dev 与读取零值次数（此处为 10 次，可以改动），若失败，释放资源，提示失败，释放资源，若成功，返回 RT_EOK，零值数据存放在设备对象 dev 中，详细零值存放参考 icm20608 软件包中零值校准章节。

3. 读取三轴加速度与三轴陀螺仪

成功校准后，进行数据读取。如果失败，提示传感器不正常工作；如果成功，打印读取的三轴加速度与三轴陀螺仪的测量值（详细的 API 介绍参考 icm20608 软件包读取三轴加速度与三轴陀螺仪章节，源码参考 icm20608.c）。

示例代码如下：

```
int main(void)
{
    icm20608_device_t dev = RT_NULL;
    const char *i2c_bus_name = "i2c2";
    int count = 0;
    rt_err_t result;

    /* 初始化 icm20608 传感器 */
    dev = icm20608_init(i2c_bus_name);
    if (dev == RT_NULL)
    {
        LOG_E("The sensor initializes failure");

        return 0;
    }
    else
    {
        LOG_D("The sensor initializes success");
    }

    /* 对 icm20608 进行零值校准：采样 10 次，求取平均值作为零值 */
    result = icm20608_calib_level(dev, 10);
    if (result == RT_EOK)
    {
        LOG_D("The sensor calibrates success");
        LOG_D("accel_offset: X%6d  Y%6d  Z%6d", dev->accel_offset.x, dev->
            accel_offset.y, dev->accel_offset.z);
        LOG_D("gyro_offset : X%6d  Y%6d  Z%6d", dev->gyro_offset.x, dev->gyro_offset
            .y, dev->gyro_offset.z);
    }
    else
```

```
    {
        LOG_E("The sensor calibrates failure");
        icm20608_deinit(dev);

        return 0;
    }

    while (count++ < 100)
    {
        rt_int16_t accel_x, accel_y, accel_z;
        rt_int16_t gyros_x, gyros_y, gyros_z;

        /* 读取三轴加速度 */
        result = icm20608_get_accel(dev, &accel_x, &accel_y, &accel_z);
        if (result == RT_EOK)
        {
            LOG_D("current accelerometer: accel_x%6d, accel_y%6d, accel_z%6d",
                accel_x, accel_y, accel_z);
        }
        else
        {
            LOG_E("The sensor does not work");
            break;
        }

        /* 读取三轴陀螺仪 */
        result = icm20608_get_gyro(dev, &gyros_x, &gyros_y, &gyros_z);
        if (result == RT_EOK)
        {
            LOG_D("current gyroscope    : gyros_x%6d, gyros_y%6d, gyros_z%6d",
                gyros_x, gyros_y, gyros_z);
        }
        else
        {
            LOG_E("The sensor does not work");
            break;
        }
        rt_thread_mdelay(1000);
    }

    return 0;
}
```

10.4.1 编译 & 下载

- RT-Thread Studio：在 RT-Thread Studio 的包管理器中下载 STM32F407-RT-SPARK 资源包，然后创建新工程，执行编译。

- MDK：首先双击 mklinks.bat，生成 rt-thread 与 libraries 文件夹链接；再使用 Env 生成 MDK5 工程；最后双击 project.uvprojx 打开 MDK5 工程，执行编译。

编译完成后，将开发板的 ST-Link USB 口与 PC 机连接，然后将固件下载至开发板。

10.4.2 运行效果

烧录完成后，此时可以在 PC 端使用终端工具打开开发板的 ST-Link 提供的虚拟串口，设置串口波特率为 115200，数据位 8 位，停止位 1 位，无流控，开发板的运行日志信息即可实时输出出来，显示如下所示：

```
[D/main] The sensor initializes success
[D/main] The sensor calibrates success
[D/main] accel_offset: X   718  Y  1524  Z  1199
[D/main] gyro_offset : X   -89  Y    71  Z   -54
[D/main] current accelerometer: accel_x   122, accel_y   -68, accel_z 16441
[D/main] current gyroscope    : gyros_x    45, gyros_y    -9, gyros_z   -19
[D/main] current accelerometer: accel_x    94, accel_y   -36, accel_z 16561
[D/main] current gyroscope    : gyros_x    10, gyros_y   -15, gyros_z    -7
[D/main] current accelerometer: accel_x   -18, accel_y   -40, accel_z 16421
[D/main] current gyroscope    : gyros_x     5, gyros_y    -3, gyros_z    13
[D/main] current accelerometer: accel_x    42, accel_y     4, accel_z 16549
[D/main] current gyroscope    : gyros_x    10, gyros_y    -6, gyros_z    -5
[D/main] current accelerometer: accel_x    14, accel_y    32, accel_z 16489
[D/main] current gyroscope    : gyros_x    13, gyros_y    -4, gyros_z    -5
[D/main] current accelerometer: accel_x    38, accel_y    40, accel_z 16381
[D/main] current gyroscope    : gyros_x    12, gyros_y    11, gyros_z    13
[D/main] current accelerometer: accel_x    70, accel_y   -20, accel_z 16369
[D/main] current gyroscope    : gyros_x    -5, gyros_y     9, gyros_z     0
[D/main] current accelerometer: accel_x    30, accel_y   -52, accel_z 16457
[D/main] current gyroscope    : gyros_x    11, gyros_y     5, gyros_z    -3
[D/main] current accelerometer: accel_x   -94, accel_y   -24, accel_z 16473
[D/main] current gyroscope    : gyros_x    42, gyros_y     0, gyros_z    13
[D/main] current accelerometer: accel_x   -38, accel_y   -64, accel_z 16421
[D/main] current gyroscope    : gyros_x    27, gyros_y    -7, gyros_z    -3
[D/main] current accelerometer: accel_x    78, accel_y    52, accel_z 16513
[D/main] current gyroscope    : gyros_x    11, gyros_y    16, gyros_z   -23
[D/main] current accelerometer: accel_x    26, accel_y   -16, accel_z 16409
[D/main] current gyroscope    : gyros_x     5, gyros_y    -8, gyros_z    -2
```

10.5 注意事项

暂无。

10.6 引用参考

- 设备与驱动：PIN 设备

- icm20608 软件包：https://github.com/RT-Thread-packages/icm20608

第 11 章

USB 鼠标例程

11.1 例程简介

本例程使用板载的六轴传感器 icm20608 获取开发板的旋转方向及角度，然后转换为鼠标的位移信息。同时使用板载按键实现鼠标的左右键，最后将这些鼠标信息通过 RT-Thread 的 USB 组件发送至电脑，从而实现开发板模拟 USB 鼠标的功能。

11.2 相关组件与软件包简介

11.2.1 RT-Thread USB 组件

该组件位于 /rt-thread/components/drivers/usb，是 RT-Thread 依据 USB2.0 协议规范将 USB 协议栈逻辑高度抽象，支持 host（主机）模式、device（从机）模式。

该组件允许用户通过宏 RT_USB_DEVICE_COMPOSITE 开启复合功能，无需额外的代码即可对多个设备类型进行复合，虚拟串口、以太网卡、人体学输入设备、大容量存储设备、微软通用 USB 等。

该组件在驱动移植方面提供了非常友好的移植接口，用户可将厂商 SDK（软件开发工具包）中 PCD（端口连接检测）驱动直接接入到 RT-Thread，实现 0 代码使用 USB。

11.2.2 ICM20608 软件包

该软件包位于 /projects/10_component_usb_mouse/packages/icm20608-v1.0.0 中，是 RT-Thread 针对六轴传感器 icm20608 功能使用的实现，使用这个软件包，可以让该传感器在 RT-Thread 上非常方便使用 icm20608 的基本功能，包括读取三轴加速度（3-axis accelerometer）、三轴陀螺仪（3-axis gyroscope）、零值校准等功能，如需详细了解该软件包，请参考 ICM20608 软件包中的 README。

11.3 硬件说明

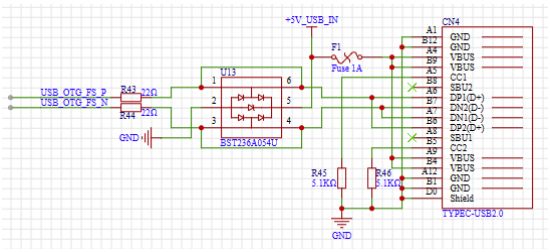

图 11.1: *USB 原理图*

如 USB 原理图所示：

USB-DEV 是本例程使用的 USB OTG 接口，其中 USB_D-(PA11) 、USB_D+(PA12) 为 USB 主机与从机的数据交互接口，本例程中开发板作为从机，电脑作为主机。

USB 接口、鼠标按键与传感器 icm20608 在开发板中的位置如下图所示:

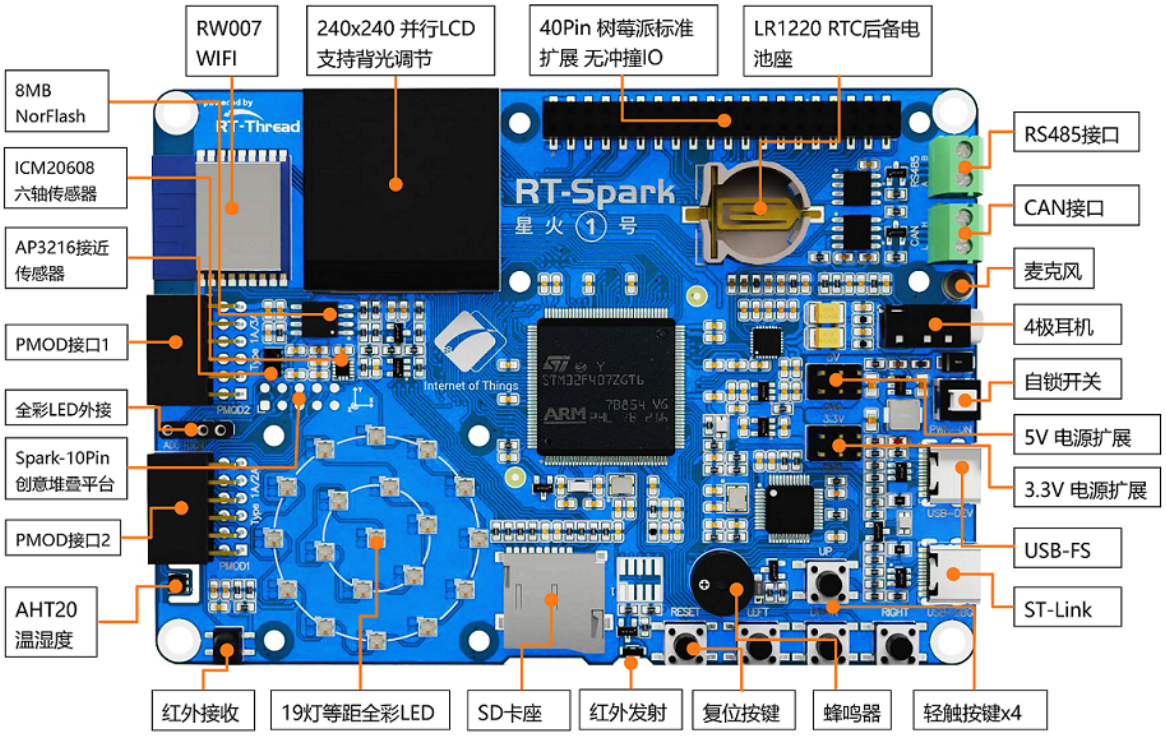

图 11.2: *硬件位置图*

11.4 软件说明

USB 鼠标例程位于 /projects/10_component_usb_mouse 目录下，重要文件摘要说明如下所示：

文件	说明
applications	应用
applications/main.c	app 入口
applications/SD_mouse.c	USB 鼠标应用主要文件
packages	内含简易好用的官方软件包
packages/icm20608-v1.0.0	六轴传感器

例程主要流程如下：

1. 初始化设备或者查找设备

   查找设备名称为 hidd 的 hid 设备，初始化 icm20608 传感器，初始化鼠标按键。

2. 打开设备

   打开查找到的 hid 设备。

3. 创建线程

   分别创建鼠标数据发送处理线程、传感器数据读取与处理线程、按键检测线程，并且启动这些创建的线程。

11.4.1 USB 鼠标功能指标定义

```
const static float mouse_rang_scope = 6.0f; /* 变 动 识 别 值 */
```

这个值设定开发板上下、左右移动变动识别值，可以自定义，值越小，鼠标越灵敏，越大鼠标越迟钝，但是应该大于 0，小于 45。

```
const static float mouse_angle_range = 80.0f; /* 读 取 角 度 有 效 角 度 */
```

这个值决定了六轴传感器 icm20608 读取角度控制值，范围为 0 - 90 度。

```
const static float mouse_move_range = 127.0f; /* 移 动 值 的 最 大 值 */
```

这个值决定了开发板倾斜角度转换成鼠标移动值的最大值，默认为最大值。

```
#define mouse_ratio (mouse_move_range / mouse_angle_range) /* 角 度 移 动 比 */
```

这个值由上面两个值决定，是将传感器读到的角度值转换成鼠标移动距离的比率。在角度一定的情况下，值越大，鼠标移动的距离越大，值越小，鼠标移动距离就小。

```
const static rt_uint8_t mouse_pixel_len = 5; /* 移 动 步 长 */
```

这个值决定了每次鼠标移动的步长，值越小，鼠标单次移动的就小，鼠标指针精度就越高，取值范围 0 - 127。

```
const static rt_uint32_t mouse_sample_times = 0; /* 鼠 标 响 应 时 间 */
```

这个值决定了鼠标响应时间，默认立即响应程序调度。初次使用，可以调大，便于观察日志。

11.4.2 原理性介绍

USB HID 鼠标实现流程图如下所示：

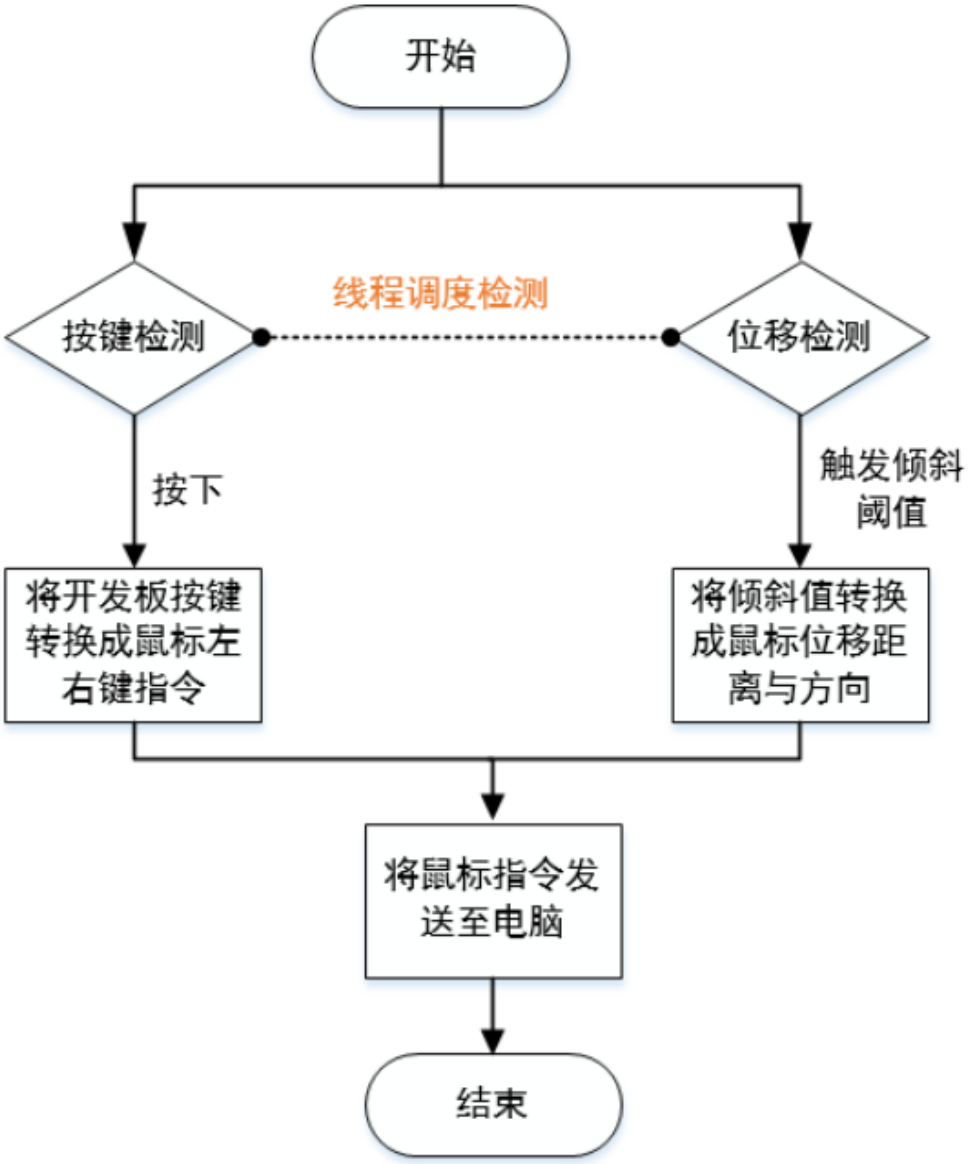

图 11.3: *原理流程图*

开发板开始正常运行后，RT-Thread 操作系统的线程调度函数，管理程序的运行，调度检测过程如下：

传感器数据读取与处理线程负责读取三轴加速度与三轴陀螺仪的测量值，在将测量值超过设置的上下、左右移动变动识别值，将读取到测量值，也就是倾斜值，转换成鼠标位移信息，其中倾斜值的正负号对应鼠标位移的方向，倾斜值的大小对应鼠标位移的大小，处理完成后将数据发送至电脑，完成鼠标位移操作。

按键检测线程负责检测开发板上按键是否被按下，若被按下，将响应的按键转换成鼠标的左键或者右键指令，处理完成后将数据发送至电脑，完成鼠标按键操作。

11.4.3 USB 数据例程程序入口

```
static int application_usb_init(void)
{
    /* 查找名称为 hidd 的设备 */
    rt_device_t device = rt_device_find("hidd");
    /* 初始化六轴传感器设备 */
    icm_device = mouse_init_icm();
    /* 初始化按键 */
    mouse_init_key();

    RT_ASSERT(device != RT_NULL);
    RT_ASSERT(icm_device != RT_NULL);

    /* 打开查找到的 hid 设备 */
    rt_device_open(device, RT_DEVICE_FLAG_WRONLY);

    /* 初始化 USB 线程 */
    rt_thread_init(&usb_thread,
                   "hidd",
                   usb_thread_entry, device,
                   usb_thread_stack, sizeof(usb_thread_stack),
                   10, 20);
    rt_thread_startup(&usb_thread);

    /* 初始化六轴传感器线程 */
    rt_thread_init(&icm_thread,
                   "icm20608",
                   icm_thread_entry, RT_NULL,
                   icm_thread_stack, sizeof(icm_thread_stack),
                   10, 20);
    rt_thread_startup(&icm_thread);

    /* 初始化按键线程 */
    rt_thread_init(&key_thread,
                   "key",
                   key_thread_entry, device,
                   key_thread_stack, sizeof(key_thread_stack),
                   10,20);
    rt_thread_startup(&key_thread);

    return 0;
}
```

11.4.4 编译 & 下载

- RT-Thread Studio：在 RT-Thread Studio 的包管理器中下载 STM32F407-RT-SPARK 资源包，然后创建新工程，执行编译。
- MDK：首先双击 mklinks.bat，生成 rt-thread 与 libraries 文件夹链接；再使用 Env 生成 MDK5 工程；最后双击 project.uvprojx 打开 MDK5 工程，执行编译。

编译完成后，将开发板的 ST-Link USB 口与 PC 机连接，然后将固件下载至开发板。

11.4.5 运行效果

烧写程序之后，并将开发板的 USB OTG 接口与电脑连接，电脑通过设备管理器查询，可以发现多了一个鼠标设备，如下图所示：

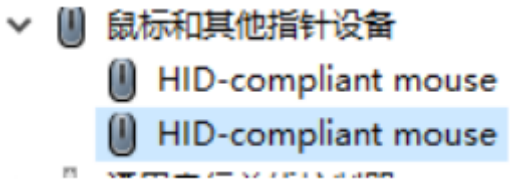

图 11.4: *运行日志*

打开串口，重启后，分别进行相关操作即可显示响应的日志，具体解释如下：

```
\ | /
- RT - Thread Operating System
/ | \ 4.0.1 build Mar 28 2019
2006 - 2019 Copyright by rt-thread team
[D/3D_mouse] The 3D mouse initializes success                    #鼠标初始化成
    功
msh >[D/3D_mouse] left down                                      #左键按下
[D/3D_mouse] left down
[D/3D_mouse] left down
[D/3D_mouse] right down                                          #右键按下
[D/3D_mouse] right down
[D/3D_mouse] right down
[D/3D_mouse] right down
[D/3D_mouse] move_max :   5, x:    0, y :   5                    #鼠标移动
[D/3D_mouse] move_max :   7, x:    3, y :   7
[D/3D_mouse] move_max :   7, x:    0, y :   7
[D/3D_mouse] move_max :  14, x:    0, y :  14
[D/3D_mouse] move_max :  14, x:    0, y :  14
[D/3D_mouse] move_max :  21, x:    0, y :  21
```

11.5 注意事项

暂无。