中科蓝讯 AB32VG1 开发实践指南

文档更新日志

2021-8-19 1.0.1 版本更新

1.更新 UART 文档

2.更新 RTC 文档

3.新增 WIFI 模块配置文档

近日，国内领先的自主物联网操作系统（RT-Thread）厂商睿赛德科技联合其高级会员国内领先 RISC-V 物联网芯片公司中科蓝讯正式发布基于 AB32VG1 RISC-V 评估板，AB32VG1 评估板原生搭载 RT-Thread 物联网操作系统，基于 RT-Thread Studio 提供 SDK，并配备了数百页开发实践指南，践行为开发者提供易获取、易用的 RISC-V 开发平台的初心。

蓝讯骄龙 AB32VG1 是中科蓝讯在 2020 RT-Thread 开发者大会上首度面向通用市场发布的其自主 RISC-V 内核 32 位 MCU 芯片，AB32VG1 主频 120M ，片上集成 RAM 192K, Flash 4Mbit，ADC，PWM，USB，UART，IIC 等资源。

在软件开发上，AB32VG1 的软件 SDK 内置 RT-Thread Studio IDE 中，可以让开发者毫无障碍的进行应用开发 ,搭配 RT-Thread 丰富的软件包可进一步降低开发门槛，助力开发者快速搭建自己的应用。

AB32VG1 评估板具有丰富的软硬件资源、详尽的例程文档和低成本等优势。在正式发布前已有数位开发者进行了内测尝鲜，并提供了宝贵的意见和建议，其中数位开发者提交了代码贡献如 mysterywolf、JiangYangJie 、iysheng 、yaoyufan 、leton-tian，多位小伙伴参与撰写和校对了实践指南，再次向他们表示感谢。

- AB32VG1 硬件相关的资料：
  https://gitee.com/bluetrum/AB32VG1_DOC

目录	原作者	整理人
零、实践指南说明	RT-Thread & 中科蓝讯	RT-Thread & 中科蓝讯
一、中科蓝讯 AB32VG1 上的 UART 实践	欧小龙	田振华
二、中科蓝讯 AB32VG1 上的 GPIO 实践	徐杰	徐杰
三、中科蓝讯 AB32VG1 上的 I2C 实践	李志青	李志青
四、中科蓝讯 AB32VG1 上的模拟 SPI 实践	秦韦忠	秦韦忠
五、中科蓝讯 AB32VG1 上的 Timer 实践	梁以江	秦韦忠
六、中科蓝讯 AB32VG1 上的 ADC 实践	杨永胜	秦韦忠
七、中科蓝讯 AB32VG1 上的 PWM 实践	江阳杰	江阳杰
八、中科蓝讯 AB32VG1 上的 WDT 实践	满鉴霆	江阳杰
九、中科蓝讯 AB32VG1 上的 RTC 实践	杨永胜	秦韦忠

十、中科蓝讯 AB32VG1 上的 SDIO 实践	黄治阳	**徐杰**
十一、中科蓝讯 AB32VG1 上的 Flash 实践	凌子健	**田振华**
十二、中科蓝讯 AB32VG1 上的 SD 实践	李健洪	**李志青**
十三、中科蓝讯 AB32VG1 上的 IRDA 实践	杨永胜	**杨永胜**
十四、中科蓝讯 AB32VG1 上的 Audio 实践	中科蓝讯	**徐杰**
十五、中科蓝讯 AB32VG1 上的 mic 实践	中科蓝讯	**田振华**
十六、中科蓝讯 AB32VG1 上的 WIFI 模块配置	徐杰	徐杰
项目 1：基于 AB32VG1 的智慧门禁系统	姚昱凡	**姚昱凡**
项目 2：基于 AB32VG1 的遥控台灯	张龙	**张龙**
项目 3：基于 AB32 的智能灯控	刘畅	**刘畅**
项目 4：WAV 音频播放	中科蓝讯	中科蓝讯
持续增加中		

以上信息如有错误，请联系官方人员微信改正：rtthread2020

零、实践指南说明

硬件介绍

AB32VG1 开发板是以中科蓝讯 (Bluetrum) 公司推出的基于 RISC-V 架构的高配置芯片 AB32VG1 为核心所组成的。

- CPU: AB32VG1 (LQFP48 封装，主频 120M ，片上集成 RAM 192K, flash 4Mbit，ADC，PWM，USB，UART，IIC 等资源)
- 搭载蓝牙模块
- 搭载 FM 模块
- 一路 TF Card 接口
- 一路 USB 接口
- 一路 IIC 接口

- 一路音频接口（美标 CTIA）
- 六路 ADC 输入引脚端子引出
- 六路 PWM 输出引脚端子引出
- 一个全彩 LED 灯模块，一个电源指示灯，三个烧录指示灯
- 一个 IRDA (红外接收端口)
- 一个 Reset 按键，三个功能按键（通用版为两个功能按键）
- 板子规格尺寸：6cm * 9cm
- I/O 口通过 2.54MM 标准间距引出，同时兼容 Arduino Uno 扩展接口，方便二次开发

开发环境

准备

实验前需要下载

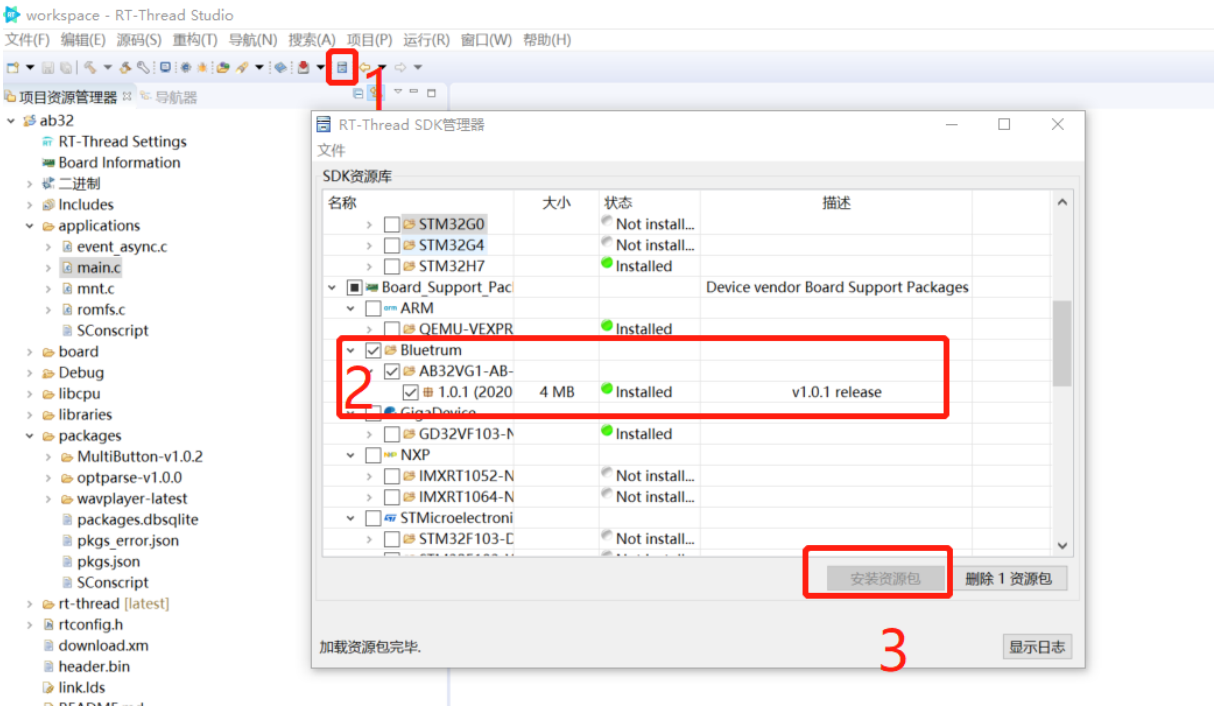
- rt-thread studio 安装包
- Downloader(下载软件)
- 配套的 USB 转串口驱动(V1 版本专用)

rt-thread studio 安装

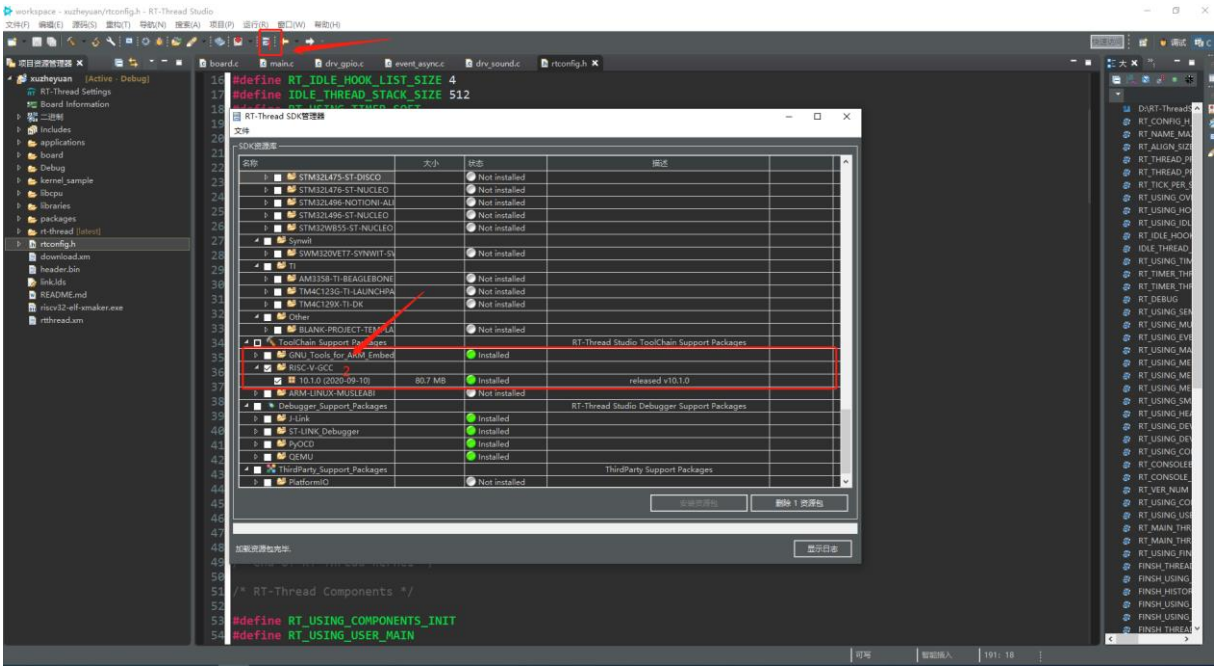
首先需要确保已经安装 rt-thread studio

在工具栏找到 SDK 管理器，点击后在弹出窗口，

Board_Support_Packages -> Bluetrum_AB32VG1-ab-prougen ,勾选，安装资源包，至此可以在 rt-thread studio 基于 AB32VG1 做开发了

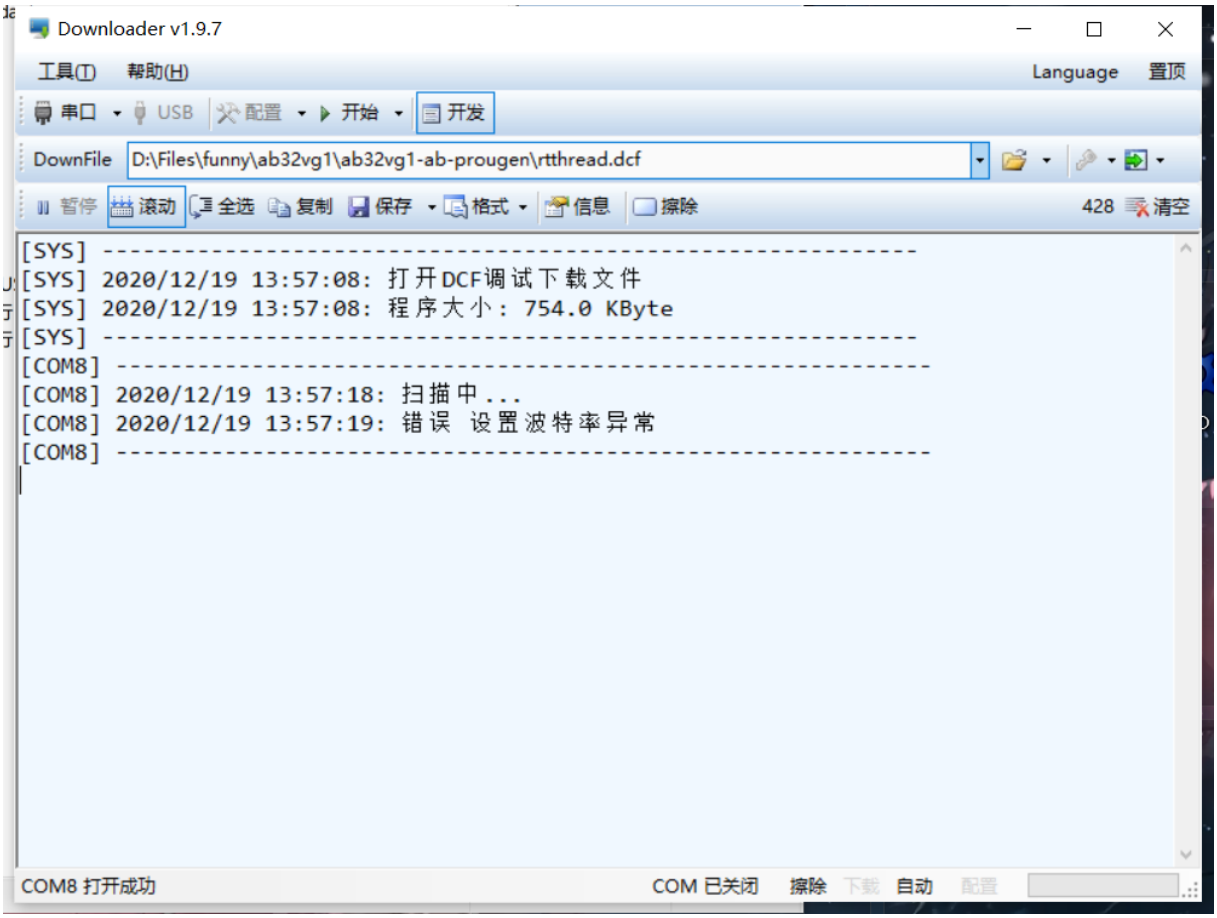


还需要在 SDK 管理器中安装 riscv 的工具链，否则无法编译

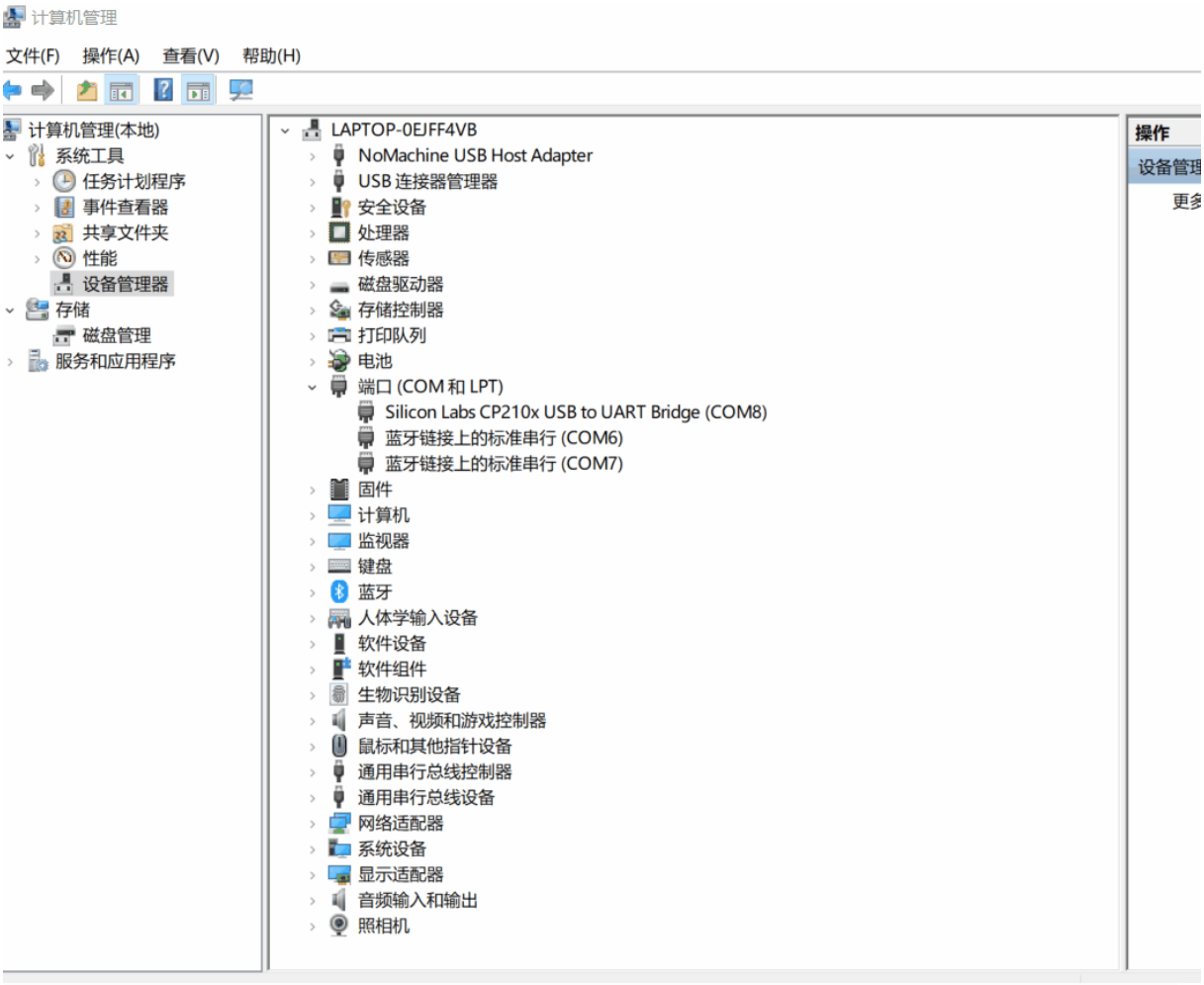


Downloader 安装

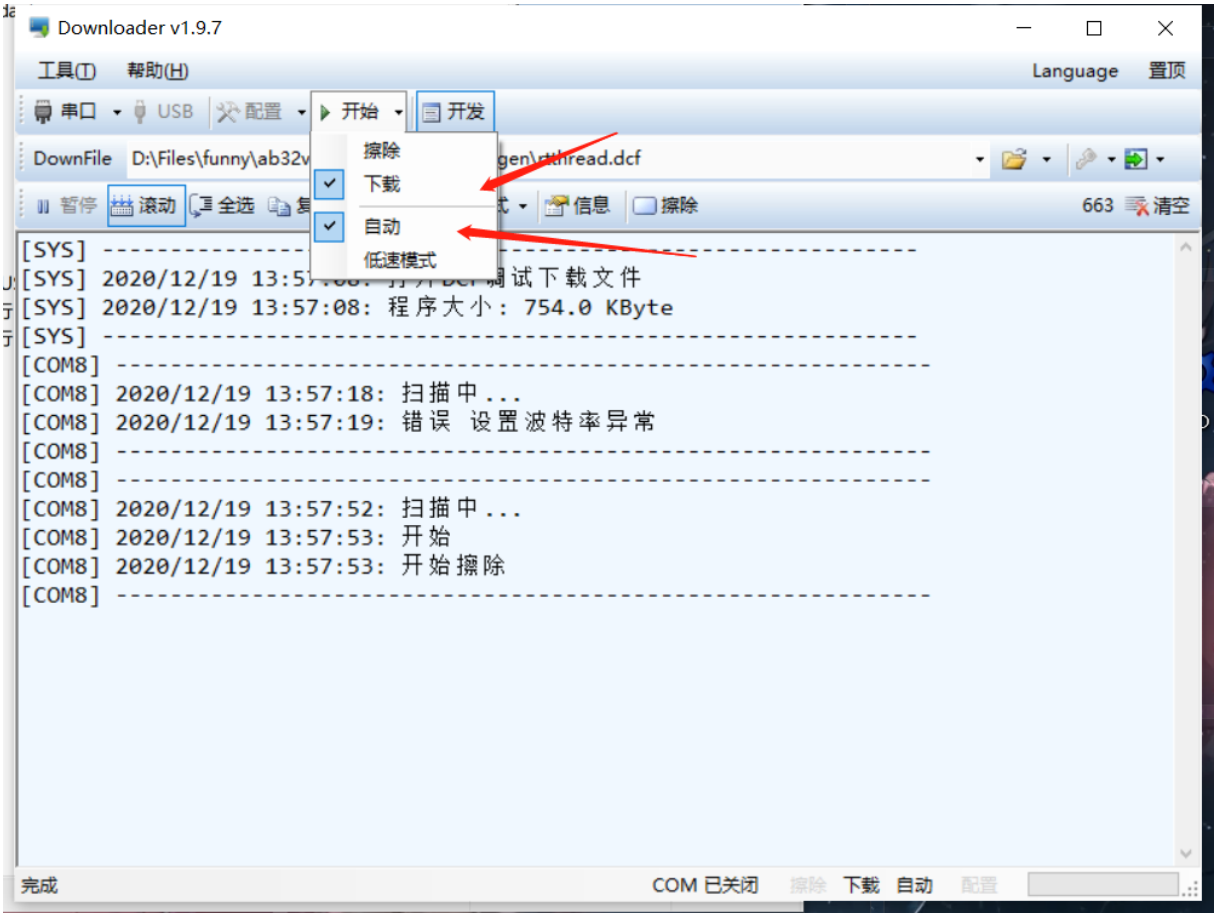
我们是使用 Downloader 进行程序的固件下载的，编译出来的固件后缀为.dcf,该文件位于工程 Debug 目录下。 Downloader 软件需要安装自己的 USB 转串口驱动，如果驱动不匹配，会报下面的这个错误，这时需要安装配套的 USB 转串口驱动 。



V2 版本的开发板使用的串口芯片是 CH340，可以直接使用系统自动安装的驱动。V1 版本的开发板使用的串口芯片是 CP2102 ，需要切换到我们的驱动，具体操作如下图



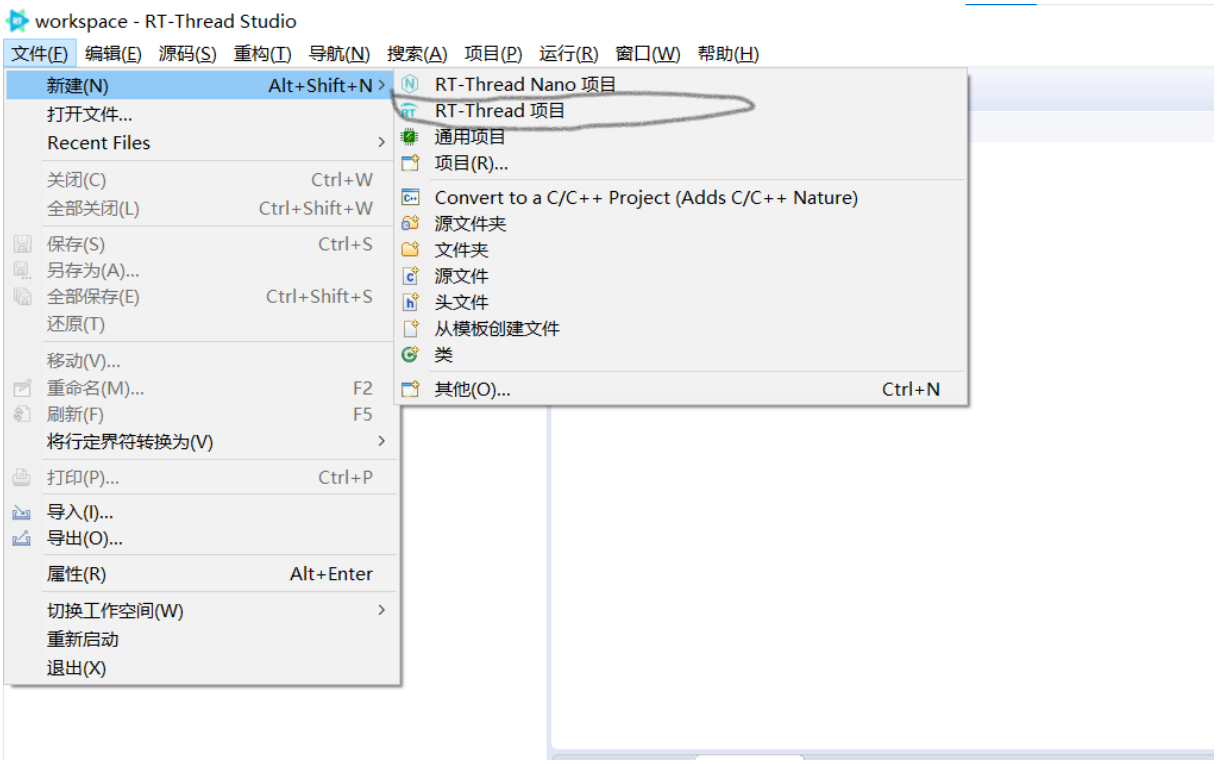
如何希望能够编译后自动下载，需要在 **Downloader** 中的下载的下拉窗中选择 **自动**



RT-Thread Studio IDE 使用的基础介绍

studio 新建工程

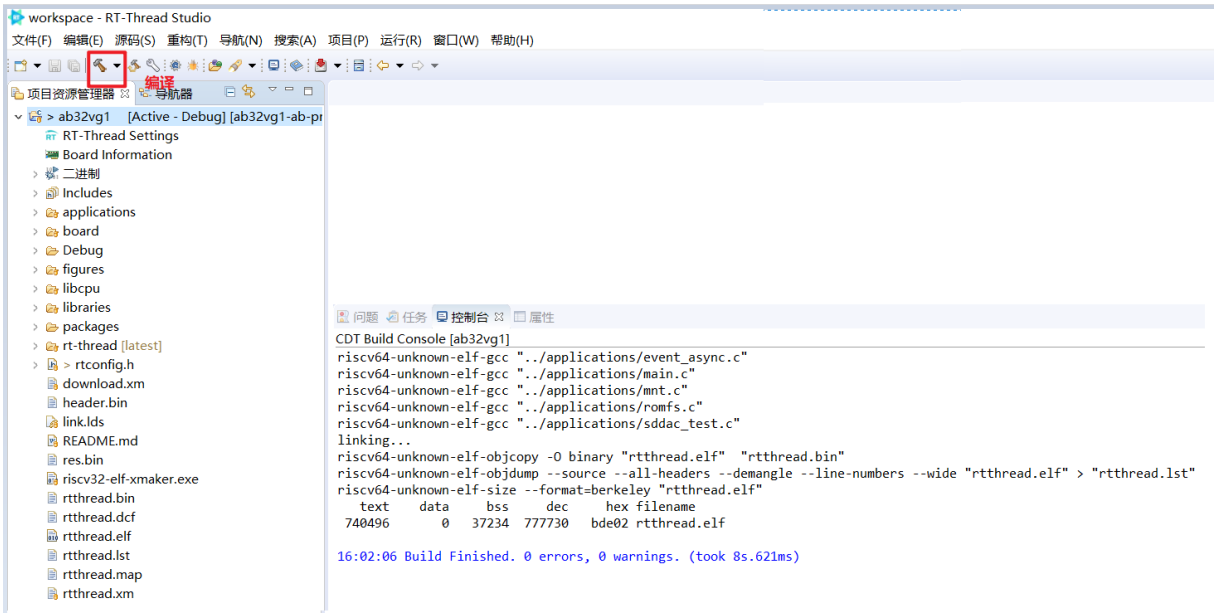
打开 studio，如下图所示，新建工程。



选择 基于开发板，然后选择 AB32VG1-AB-PROUGEN

编译

单击编译按键，编译工程，如下图所示。



一、 中科蓝讯 AB32VG1 上的 UART 实践

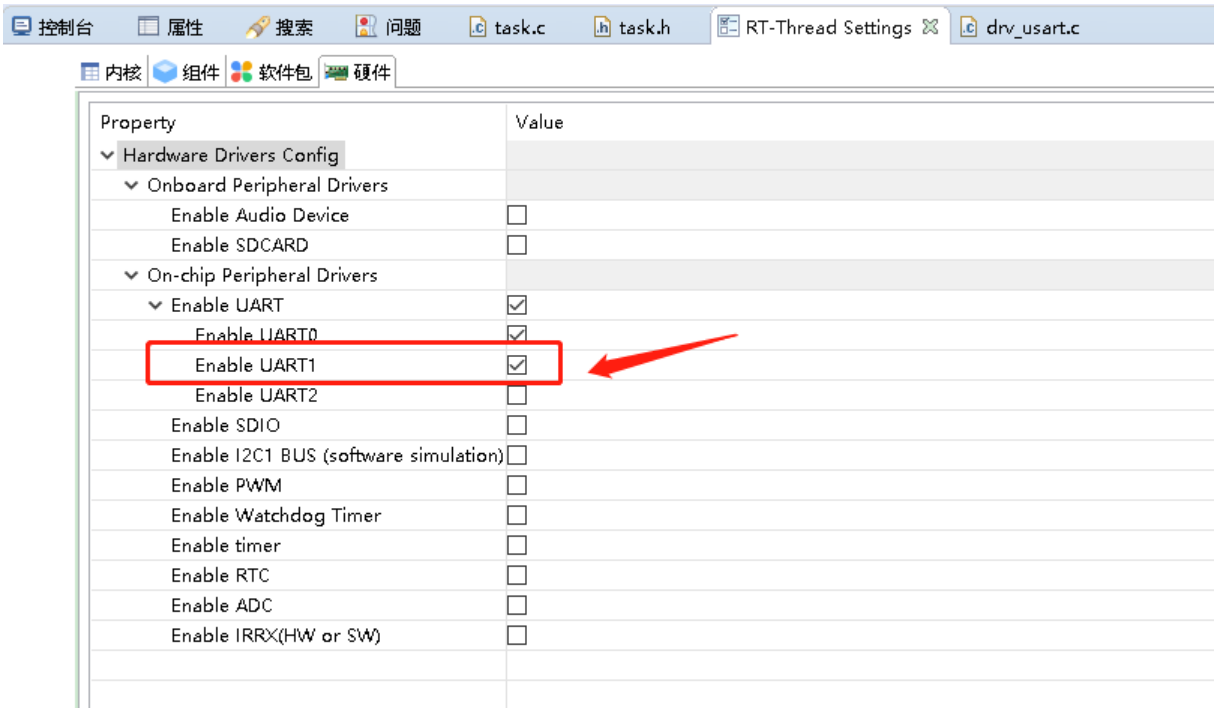
1. 前言说明

1.1. 本章内容

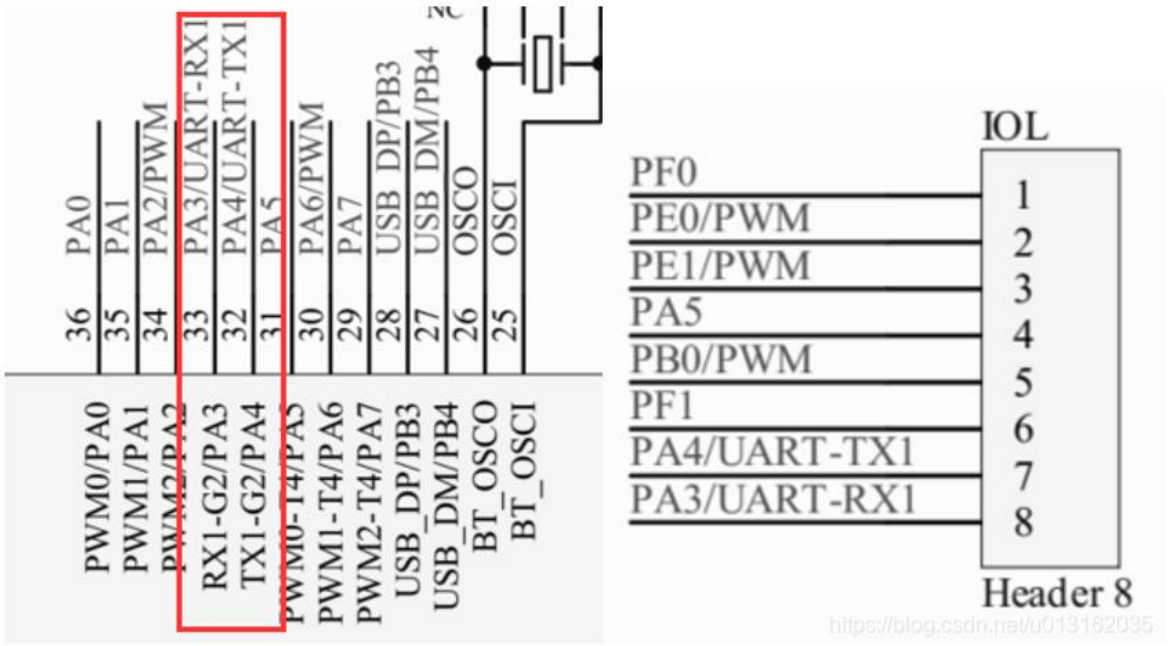
本章通过 RT-Thread Studio 配置 AB32VG1 片上外设 UART 的功能，实现开发板和 PC 进行通信。

1.2. 模块介绍

AB32VG1 的串口 0 被用作系统调试串口，串口 1 可以用作通讯端口。RT-Thread 里做好了 UART0 和 UART1 的驱动，只要打开相应的设备即可。



开发板上串口部分的电路图如下图所示：



从电路图上看，串口 1 使用的是 PA3 和 PA4。

1.3. 开发软件

开发环境：RT-Thread Studio

下载工具：Downloader.exe

2. 步骤说明

2.1. 新建工程

2.1.1.文件->新键->RT-Thread 项目。

2.1.2.选择基于开发板，填写工程名字。

2.1.3.开发板：AB32VG1-AB-PROUGEN。

2.1.4.BSP：1.0.8。

2.1.3.其他默认，点完成。一个新的项目就建成了。

2.2. 编写测试程序

在 applications 新键 task.c 文件。此例程源自 RT-Thread 文档中心，引用时有修改。

[task.c]

```
/*
 * 程序清单：这是一个 串口 设备使用例程
 * 例程导出了 uart_sample 命令到控制终端
 * 命令调用格式：uart_sample uart1
 * 命令解释：命令第二个参数是要使用的串口设备名称，为空则使用默认的串口设备
 * 程序功能：通过串口输出字符串"hello RT-Thread!"，然后错位输出输入的字符
*/

#include <rtthread.h>

#define SAMPLE_UART_NAME       "uart1"

/* 用于接收消息的信号量 */
static struct rt_semaphore rx_sem;
static rt_device_t serial;

/* 接收数据回调函数 */
static rt_err_t uart_input(rt_device_t dev, rt_size_t size)
{
    /* 串口接收到数据后产生中断，调用此回调函数，然后发送接收信号量 */
    rt_sem_release(&rx_sem);

    return RT_EOK;
```

```
}

static void serial_thread_entry(void *parameter)
{
    char ch;

    while (1)
    {
        /* 从串口读取一个字节的数据，没有读取到则等待接收信号量 */
        while (rt_device_read(serial, -1, &ch, 1) != 1)
        {
            /* 阻塞等待接收信号量，等到信号量后再次读取数据 */
            rt_sem_take(&rx_sem, RT_WAITING_FOREVER);
        }
        /* 读取到的数据通过串口错位输出 */
        ch = ch + 1;
        rt_device_write(serial, 0, &ch, 1);
    }
}

static int uart_sample(int argc, char *argv[])
{
    rt_err_t ret = RT_EOK;
    char uart_name[RT_NAME_MAX];
    char str[] = "hello RT-Thread!\r\n";

    if (argc == 2)
    {
        rt_strncpy(uart_name, argv[1], RT_NAME_MAX);
    }
    else
    {
        rt_strncpy(uart_name, SAMPLE_UART_NAME, RT_NAME_MAX);
    }

    /* 查找系统中的串口设备 */
    serial = rt_device_find(uart_name);
    if (!serial)
    {
        rt_kprintf("find %s failed!\n", uart_name);
        return RT_ERROR;
    }

    /* 初始化信号量 */
    rt_sem_init(&rx_sem, "rx_sem", 0, RT_IPC_FLAG_FIFO);
```

```
    /* 以中断接收及轮询发送模式打开串口设备 */
    rt_device_open(serial, RT_DEVICE_FLAG_INT_RX);
    /* 设置接收回调函数 */
    rt_device_set_rx_indicate(serial, uart_input);
    /* 发送字符串 */
    rt_device_write(serial, 0, str, (sizeof(str) - 1));

    /* 创建 serial 线程 */
    rt_thread_t thread = rt_thread_create("serial", serial_thread_entry, RT_NULL, 1024, 25, 10);
    /* 创建成功则启动线程 */
    if (thread != RT_NULL)
    {
        rt_thread_startup(thread);
    }
    else
    {
        ret = RT_ERROR;
    }

    return ret;
}
/* 导出到 msh 命令列表中 */
MSH_CMD_EXPORT(uart_sample, uart device sample);
```

由于在初始化串口时，默认波特率是 1500000，可以在 libraries->hal_drivers->drv_usart.c 中 int rt_hw_usart_init(void)做些修改。

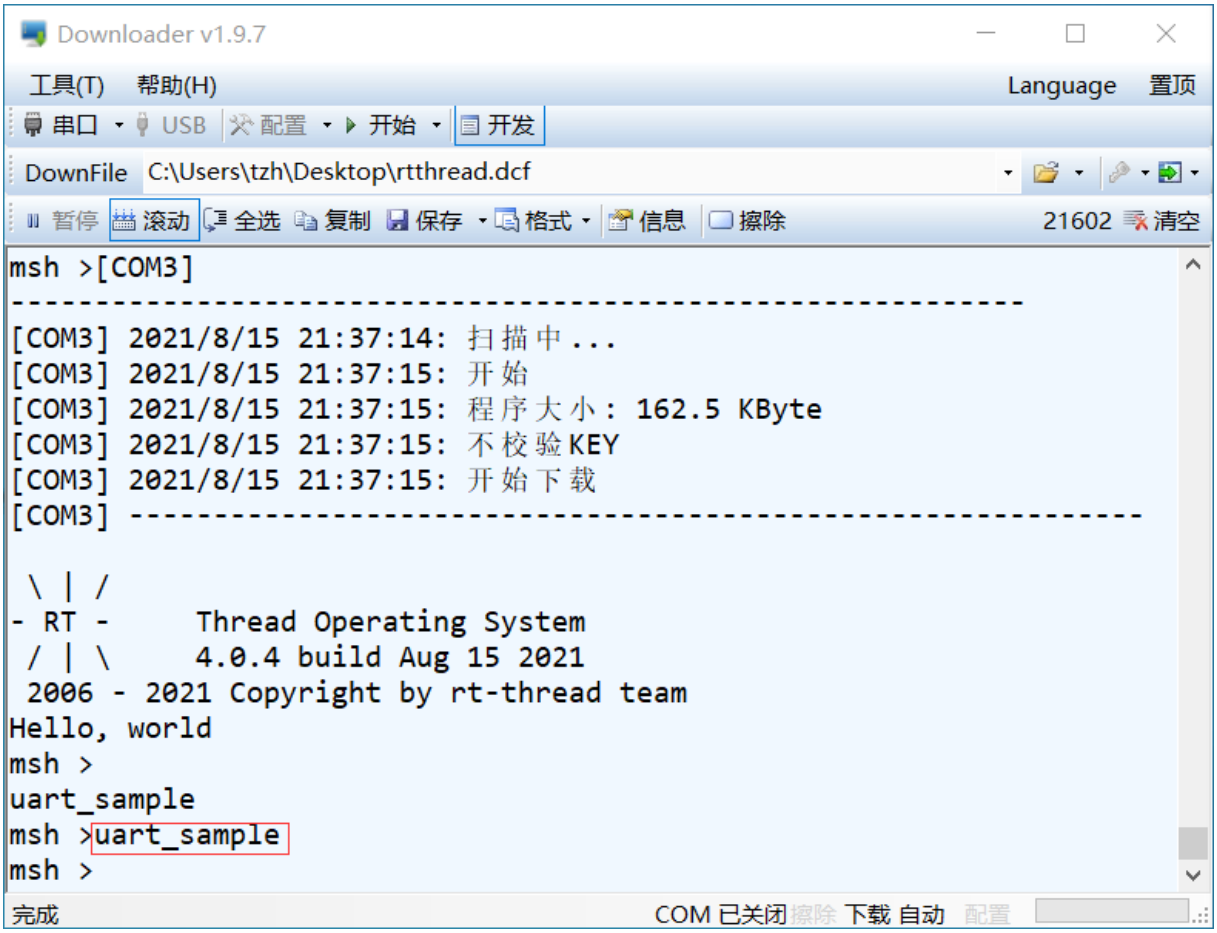
```
int rt_hw_usart_init(void)
{
    rt_size_t obj_num = sizeof(uart_obj) / sizeof(struct ab32_uart);
    struct serial_configure config = RT_SERIAL_CONFIG_DEFAULT;
    rt_err_t result = 0;

    rt_hw_interrupt_install(IRQ_UART0_2_VECTOR, uart_isr, RT_NULL, "ut_isr");

    for (int i = 0; i < obj_num; i++)
    {
        /* init UART object */
        uart_obj[i].config              = &uart_config[i];
        uart_obj[i].rx_idx              = 0;
        uart_obj[i].rx_idx_prev         = 0;
        uart_obj[i].serial.ops          = &ab32_uart_ops;
        uart_obj[i].serial.config       = config;
        uart_obj[i].serial.config.baud_rate = 1500000;
```

```
        uart_obj[i].rx_buf              = rt_malloc(uart_config[i].fifo_size);

        if (uart_obj[i].rx_buf == RT_NULL) {
            LOG_E("uart%d malloc failed!", i);
            continue;
        }
        //如果是串口 1，修改波特率位 115200
        if (i == 1)
        {
            uart_obj[i].serial.config.baud_rate = 115200;
        }
        //------------------
        /* register UART device */
        result = rt_hw_serial_register(&uart_obj[i].serial, uart_obj[i].config->name,
                                       RT_DEVICE_FLAG_RDWR
                                       | RT_DEVICE_FLAG_INT_RX
                                       | RT_DEVICE_FLAG_INT_TX
                                       | uart_obj[i].uart_dma_flag
                                       , NULL);
        RT_ASSERT(result == RT_EOK);
    }

    return result;
}
```

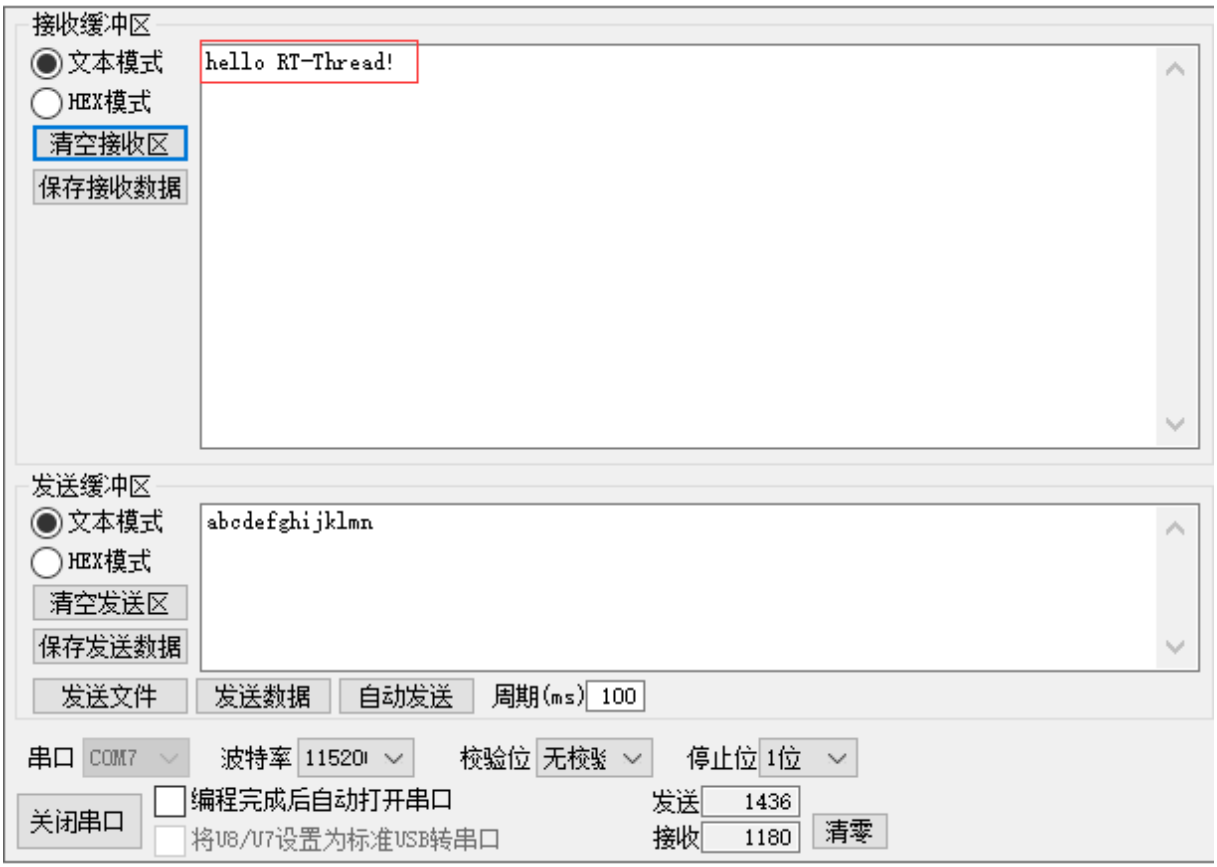
3. 代码验证

编译，下载。在 finish 终端输入

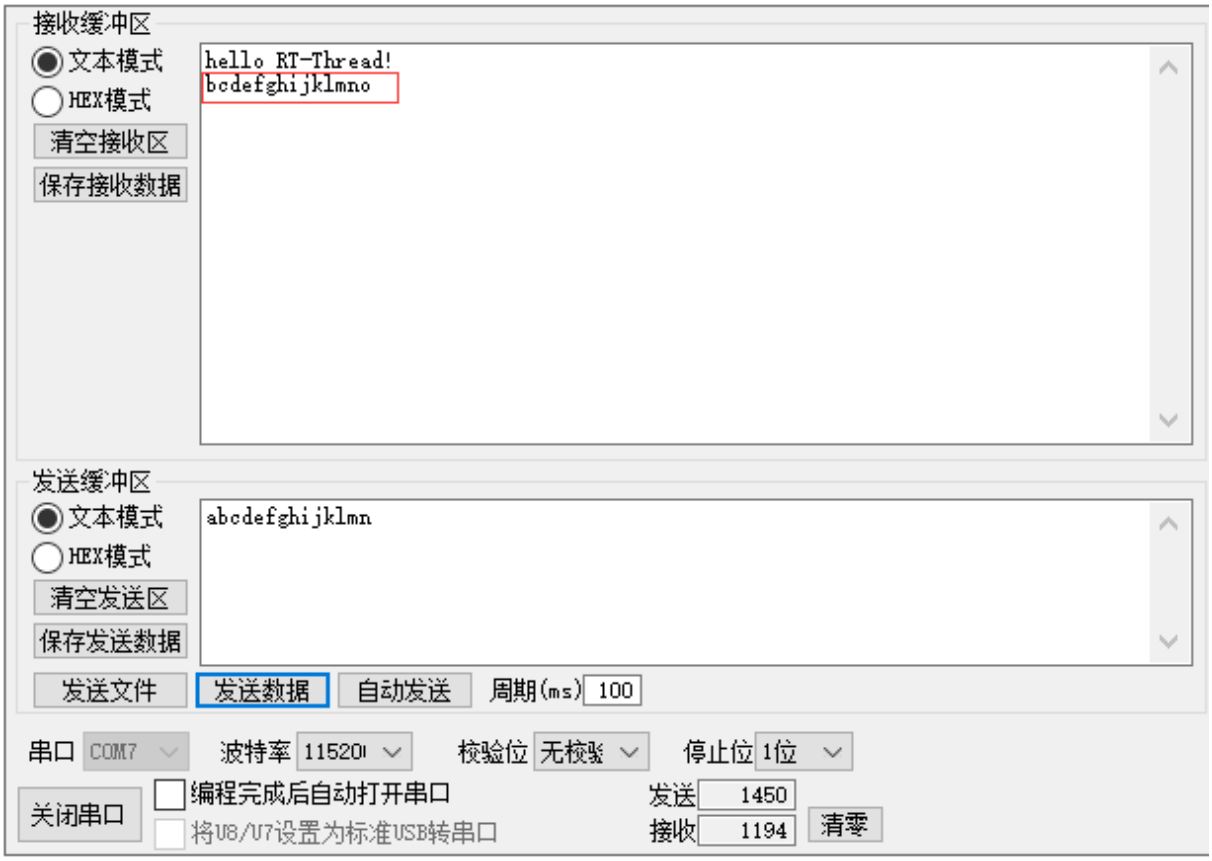
uart_sample



使用串口调试助手即可查看 uart1 的输出信息。



同样也可通过串口调试助手发送数据，开发板逐个字符加一后返回输出。



4. 章节总结

在开发板上，AB32VG1 的串口 0 被用作调试接口，用户只能使用串口 1 实现具体功能。RT-Thread Studio 在此开发板 BSP1.0.8 版本上对串口支持做了升级，用户使用时体验更方便。

二、中科蓝讯 AB32VG1 上的 GPIO 实践

1. 前言说明

1.1 本章内容

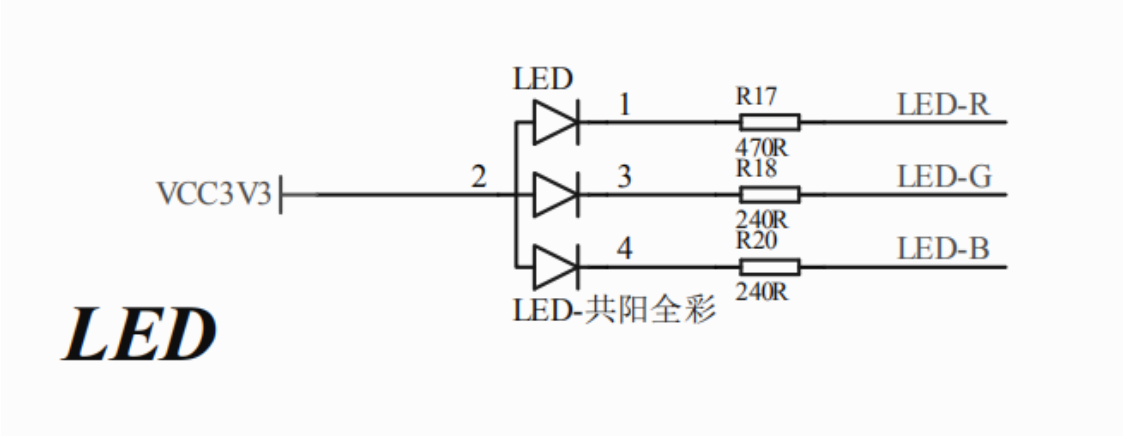
本章通过 RT-Thread Studio 配置 AB32VG1 片上外设 GPIO 的引脚，控制 RGB 彩灯

进行简单的颜色变换

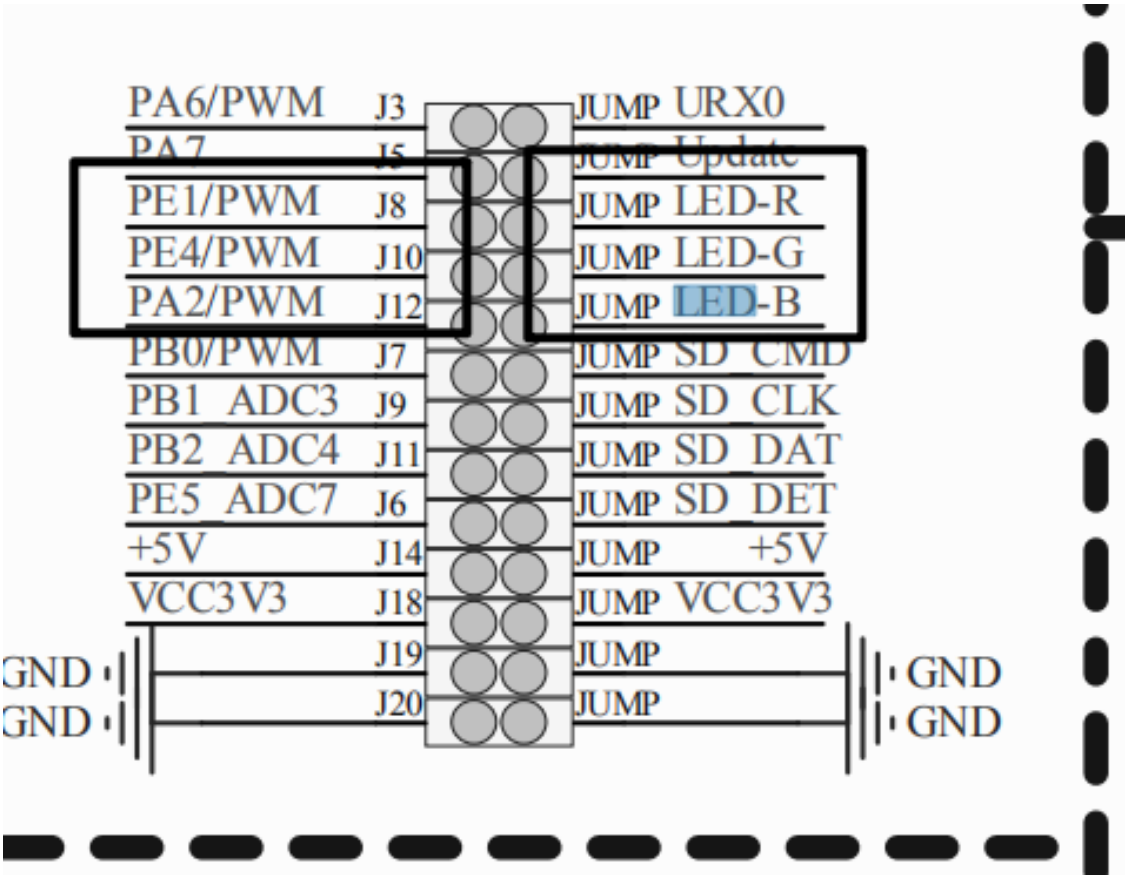
1.2 模块介绍

开发板上板载一个三色 RGB 彩灯

RGB 彩灯原理图如下



开发板引脚连接如下图，引脚 PA2 对应蓝灯，引脚 PE1 对应红灯，引脚 PE4 对应绿灯，RGB 为共阳极，当引脚拉低时，对应的 led 点亮



1.3 开发软件

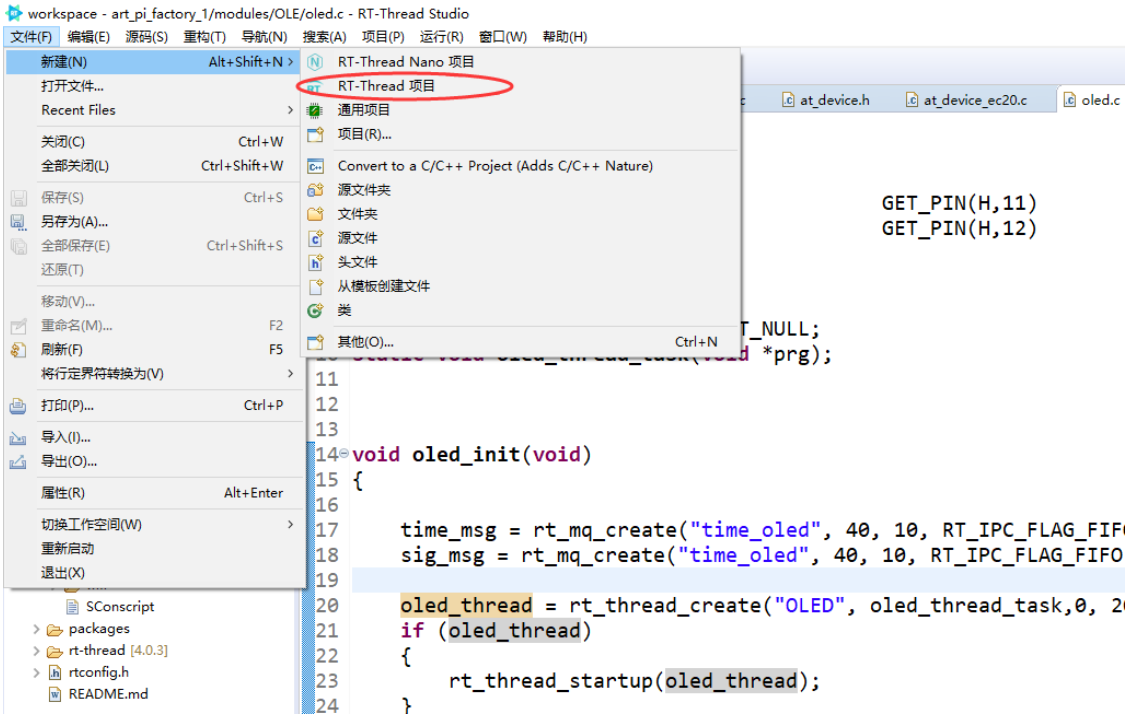
编译平台：**RT-Thread Studio：** 安装链接

下载平台：**Downloader:** 安装链接

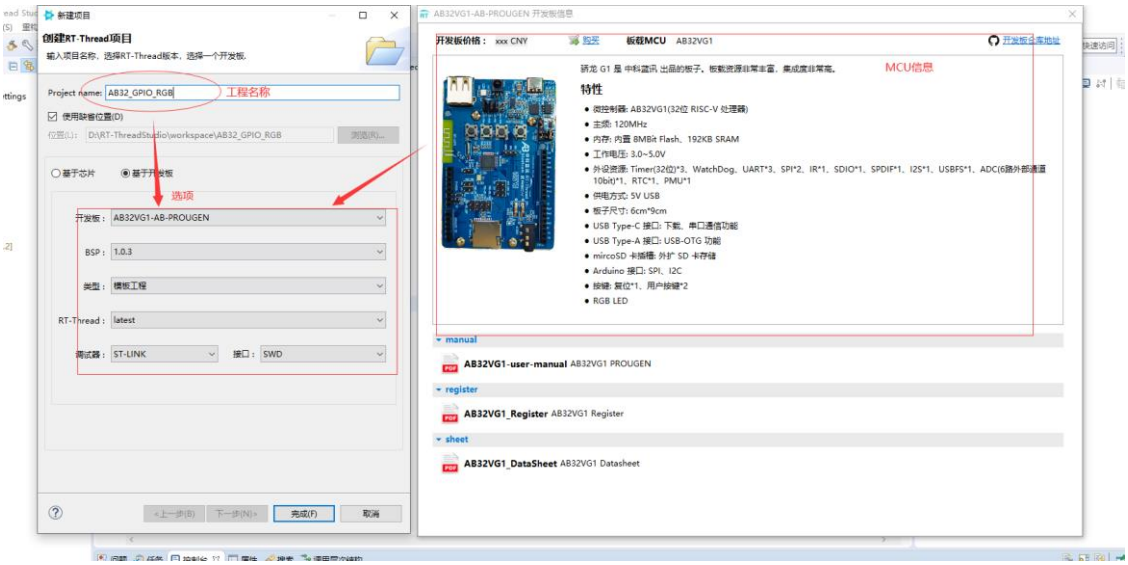
2. 步骤说明

2.1 新建工程

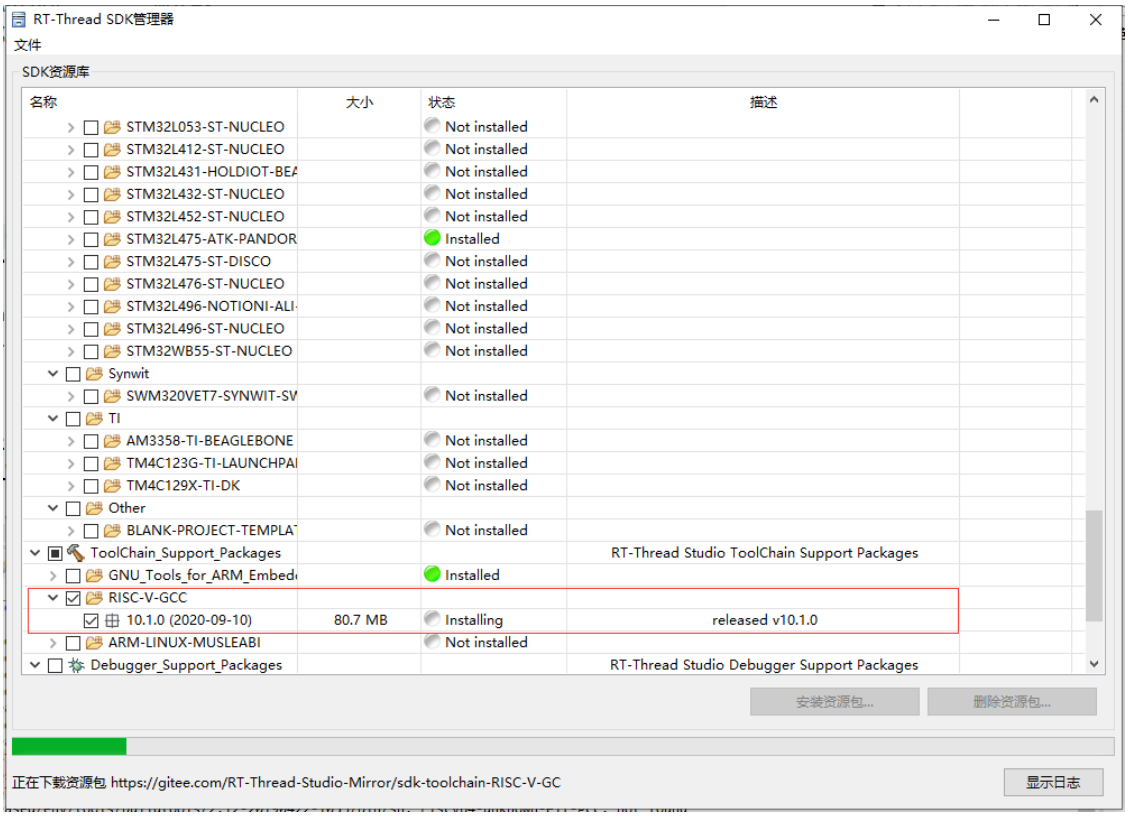
点击 文件-> 新建-> RT-Thread 项目控件



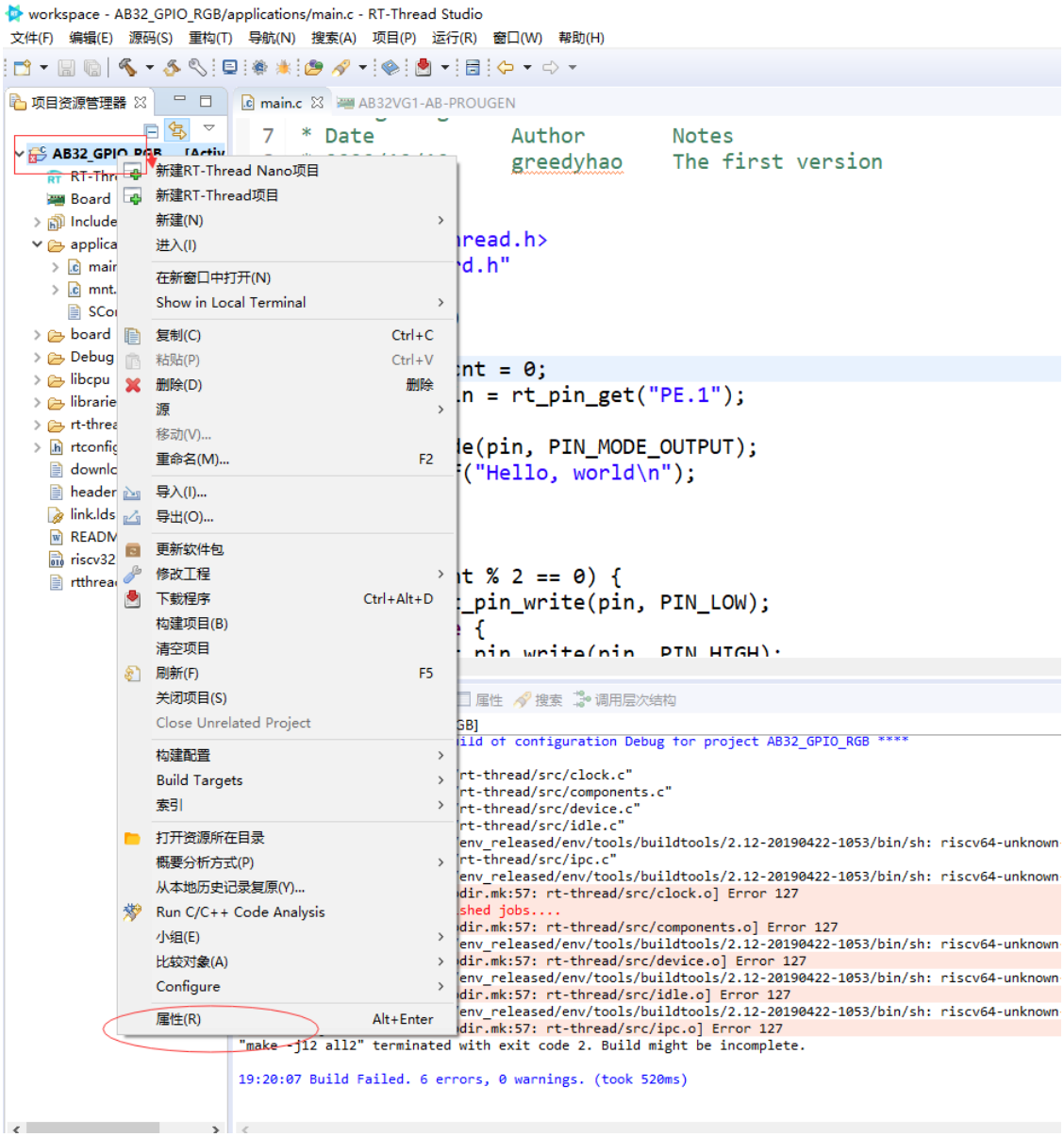
选择基于开发板的项目，填写工程名字，选择我们**使用到的开发板（AB32VG1）**，调试器我们随便选，下载方式不是通过此处下载



注意：如果第一次使用 RISC-V 芯片需要安装工具链，在 SDK 管理器中下载工具链

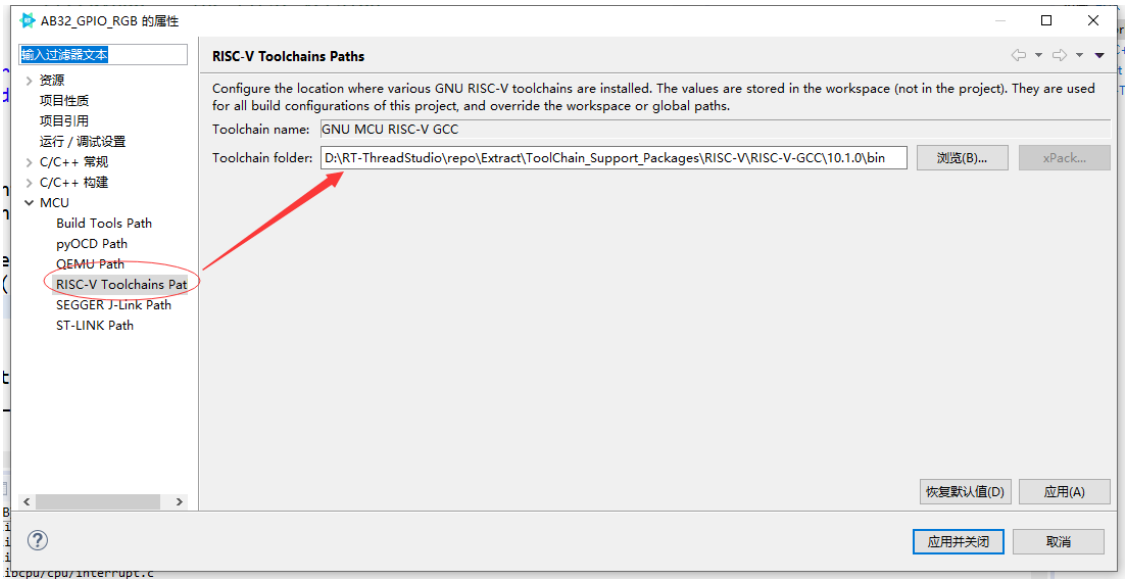


然右击项目名称，进入属性

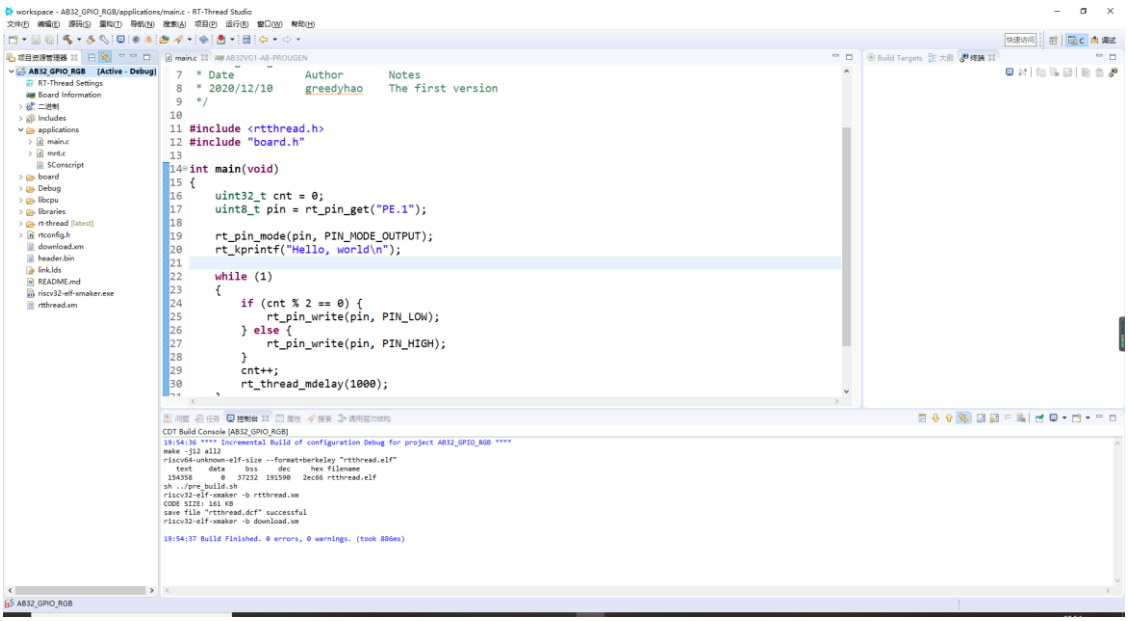


找到 MCU->RISC-V ToolchainsPat ，配置 Tool 的环境，在软件安装位置下面的路径中

软 件 安 装 位 置 \RT-ThreadStudio\repo\Extract\ToolChain_Support_Packages\RISC-V\RISC-V-GCC\10.1.0\bin



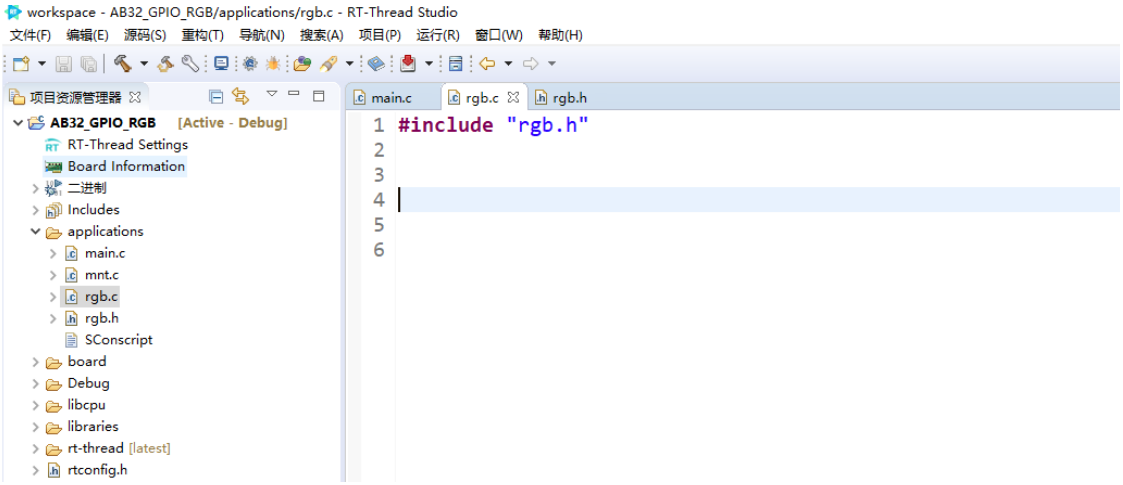
工程新建后左边的项目资源管理器会显示我们的工程，我们把他展开，编译一下，编译结果如下



编译无报错，新建工程完成了！

2.2 编写 RGB 驱动程序文件

新建程序文件：在 applications 文件夹下新建立一个 rgb.c 和.h 文件，建立后如下



rgb.c 内写入如下程序

添加头文件，定义一个 RGB 结构体，并声明 LED

```
#include "rgb.h"
#include <rtthread.h>
#include "board.h"

struct Led_s
{
    uint8_t LED_R;
    uint8_t LED_B;
    uint8_t LED_G;
};  // 定义一个 RGB 结构体

struct Led_s Led;
```

编写初始化驱动程序，调用 rt_pin_get 获取 led 句柄，通过句柄设置对应引脚模式为输出模式

```
void RGB_Init(void)
{
    // 获得 led
```

```
    Led.LED_R = rt_pin_get("PE.1");
    Led.LED_G = rt_pin_get("PE.4");
    Led.LED_B = rt_pin_get("PA.2");
    // 设置引脚为输出方式
    rt_pin_mode(Led.LED_R, PIN_MODE_OUTPUT);
    rt_pin_mode(Led.LED_G, PIN_MODE_OUTPUT);
    rt_pin_mode(Led.LED_B, PIN_MODE_OUTPUT);
}
```

编写 rgb 不同颜色点灯驱动，通过 rt_pin_write 来控制 gpio 口电平高低，点亮红灯即拉低红灯引脚，拉高其他两个颜色灯的引脚

```
//传入参数 on=1：对应亮，on=0:对应灭
//红灯驱动
void RGB_Red(rt_bool_t on)
{
    rt_pin_write(Led.LED_G, PIN_HIGH);
    rt_pin_write(Led.LED_B, PIN_HIGH);
    if (on) {
        rt_pin_write(Led.LED_R, PIN_LOW);
    }else {
        rt_pin_write(Led.LED_R, PIN_HIGH);
    }
}
//蓝灯驱动
void RGB_Blue(rt_bool_t on){
    rt_pin_write(Led.LED_G, PIN_HIGH);
    rt_pin_write(Led.LED_R, PIN_HIGH);
    if (on) {
        rt_pin_write(Led.LED_B, PIN_LOW);
    }else {
        rt_pin_write(Led.LED_B, PIN_HIGH);
    }
}
//绿灯驱动
void RGB_Green(rt_bool_t on)
{
    rt_pin_write(Led.LED_R, PIN_HIGH);
    rt_pin_write(Led.LED_B, PIN_HIGH);
    if (on) {
        rt_pin_write(Led.LED_G, PIN_LOW);
    }else {
        rt_pin_write(Led.LED_G, PIN_HIGH);
    }
```

```
}
```

2.3 编写主程序文件

编写 rgb 彩灯运行线程，三种颜色依次切换，中间延时 1s

```
static void rgb_thread_entry(void* p)
{
    RGB_Init();
    while(1)
    {
       rt_thread_mdelay(1000);
       RGB_Blue(1);
       rt_thread_mdelay(1000);
       RGB_Green(1);
       rt_thread_mdelay(1000);
       RGB_Red(1);
    }
}
```

创建线程启动函数，用于启动上一步编写的线程主体

```
static int Thread_RGB(void)
{
    rt_thread_t thread = RT_NULL;
    thread = rt_thread_create("rgb", rgb_thread_entry, RT_NULL, 512, 10, 10);
    if(thread == RT_NULL)
    {
        rt_kprintf("Thread_GRB Init ERROR");
        return RT_ERROR;
    }
    rt_thread_startup(thread);
}
```

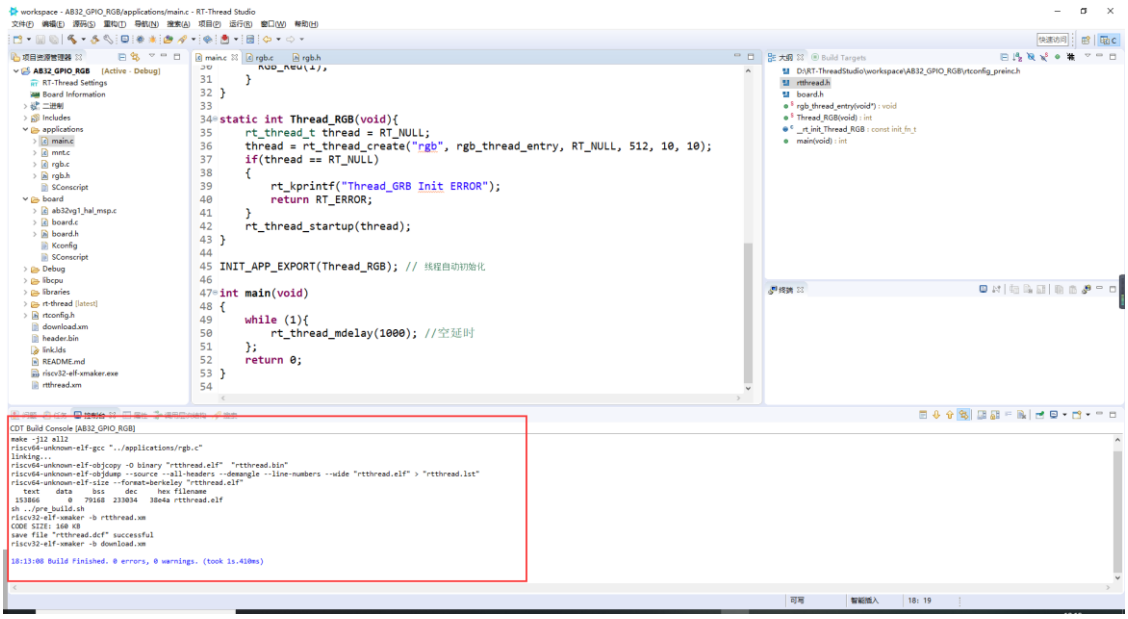
添加将线程初始化添加入系统初始化中

```
INIT_APP_EXPORT(Thread_RGB);
```

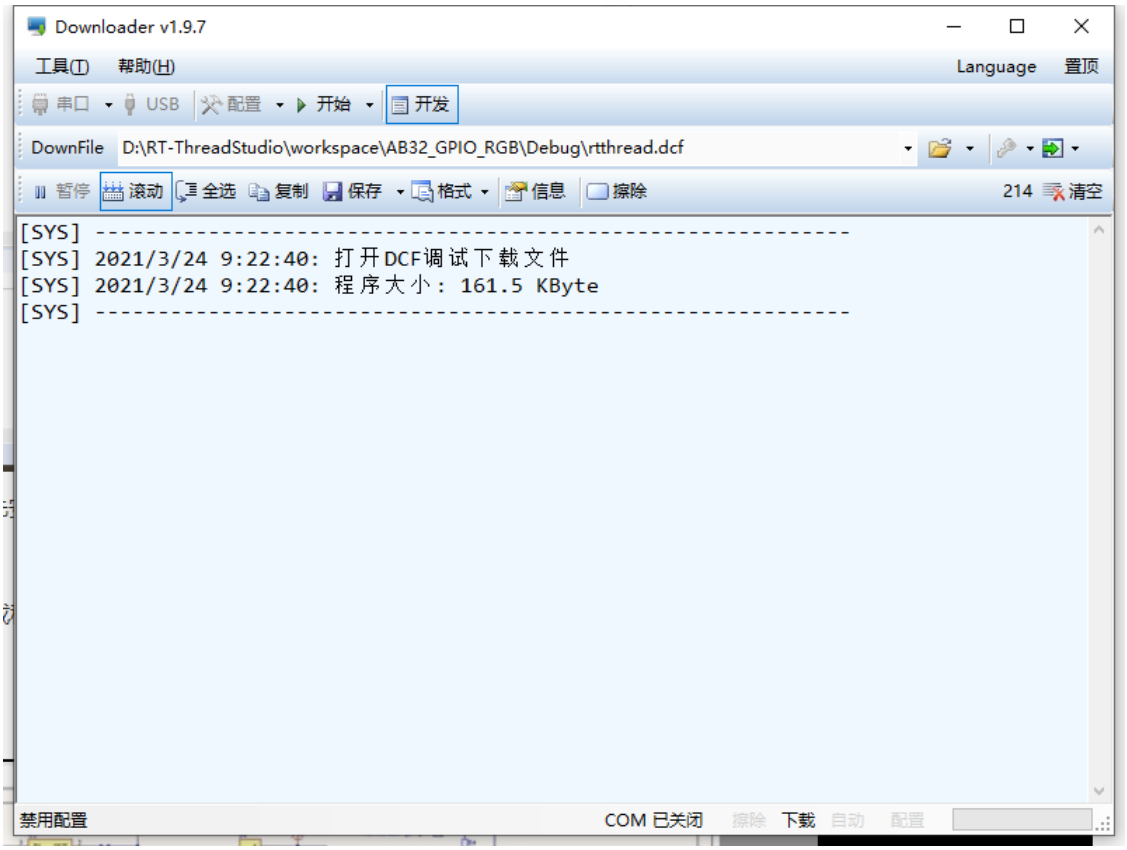
3. 代码验证

将 main.c 中的 while 里的代码改成 rt_thread_mdelay(1000);

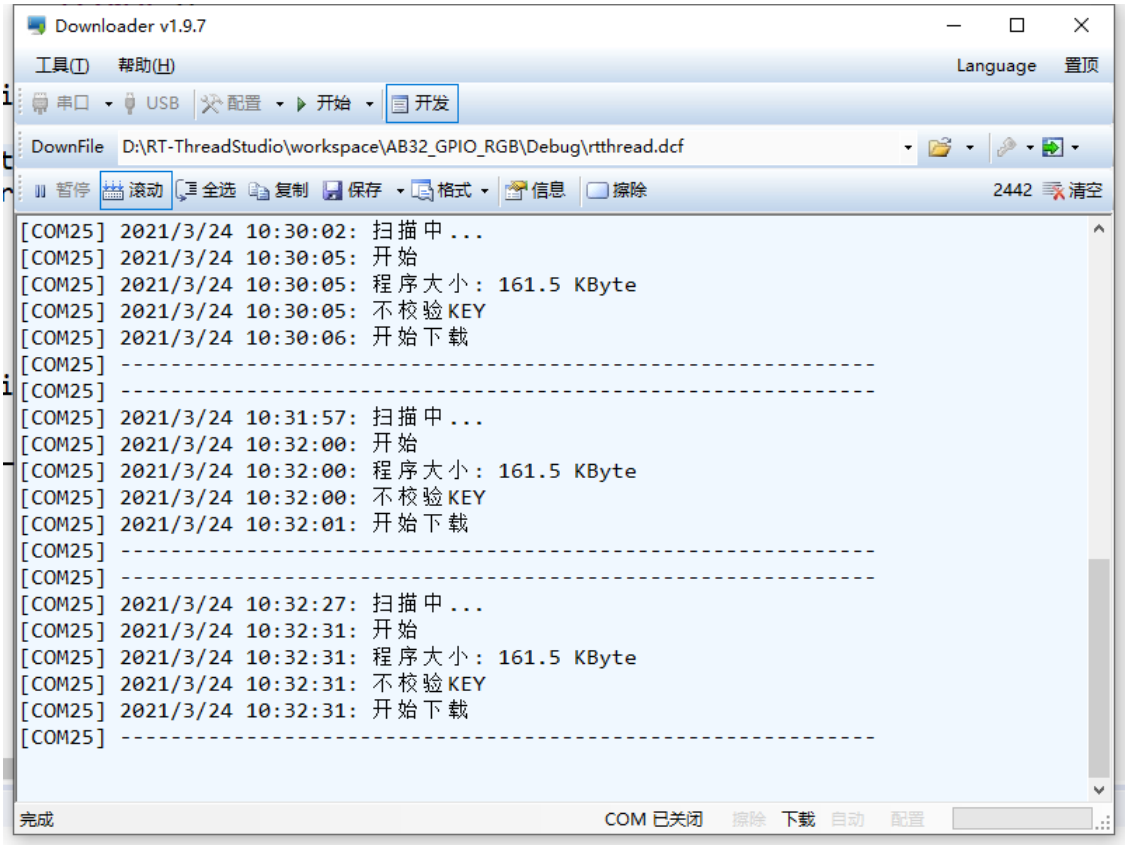
编译程序，可以看到无报错



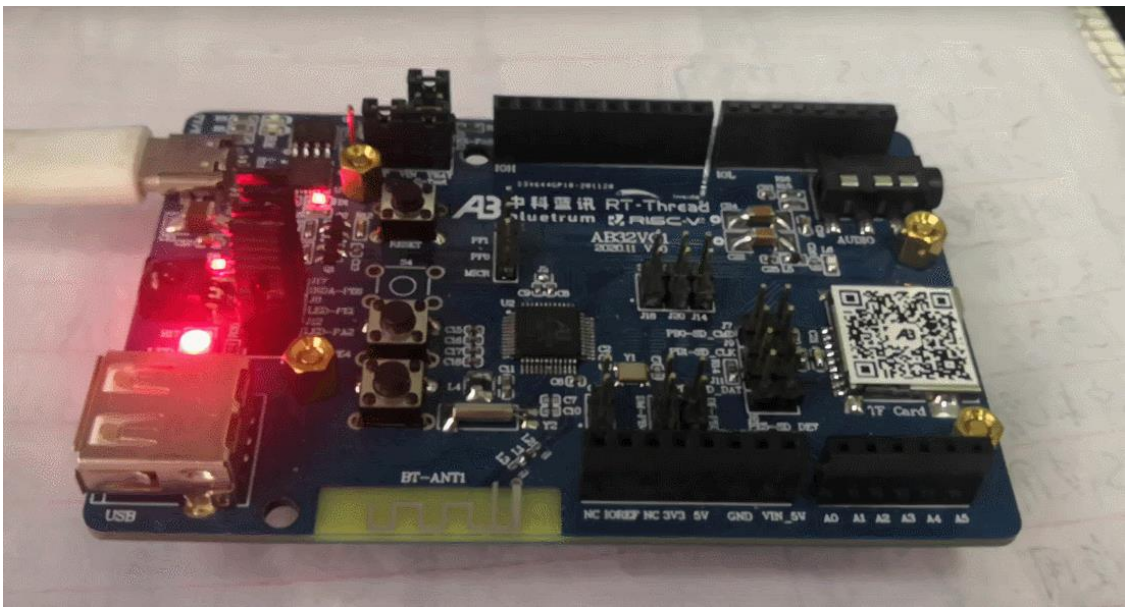
编译完成，打开 Downloaded 下载器，通过 download 下载生成的.dcf 文件（第一次使用前需要先安装串口驱动）



扫描串口，点击开始后，按一下板子上复位按键下载程序


Downloader v1.9.7
工具(T)
帮助(H)
Language
置顶
串口
USB
配置
开始
开发
DownFile
D:\RT-ThreadStudio\workspace\AB32_GPIO_RGB\Debug\rtthread.dcf
暂停
滚动
全选
复制
保存
格式
信息
擦除
2442
清空
[COM25] 2021/3/24 10:30:02: 扫描中...
[COM25] 2021/3/24 10:30:05: 开始
[COM25] 2021/3/24 10:30:05: 程序大小: 161.5 KByte
[COM25] 2021/3/24 10:30:05: 不校验KEY
[COM25] 2021/3/24 10:30:06: 开始下载
[COM25] ----------
[COM25] ----------
[COM25] 2021/3/24 10:31:57: 扫描中...
[COM25] 2021/3/24 10:32:00: 开始
[COM25] 2021/3/24 10:32:00: 程序大小: 161.5 KByte
[COM25] 2021/3/24 10:32:00: 不校验KEY
[COM25] 2021/3/24 10:32:01: 开始下载
[COM25] ----------
[COM25] ----------
[COM25] 2021/3/24 10:32:27: 扫描中...
[COM25] 2021/3/24 10:32:31: 开始
[COM25] 2021/3/24 10:32:31: 程序大小: 161.5 KByte
[COM25] 2021/3/24 10:32:31: 不校验KEY
[COM25] 2021/3/24 10:32:31: 开始下载
[COM25] ----------
完成
COM 已关闭
擦除
下载
自动
配置

程序现象


RT-Thread
BT-ANT1
USB
TF Card

4. 章节总结

本章节我们学会了如何在 RT-Thread 上配置 GPIO 口，总的来说 GPIO 的使用步骤很简单，第一步获取对应 GPIO 句柄，第二步配置 GPIO 模式，之后就可以调用 rtt 函数对 GPIO 进行读写操作了！

三、中科蓝讯 AB32VG1 上的 I2C 实践

1. 前言说明

1.1 本章内容

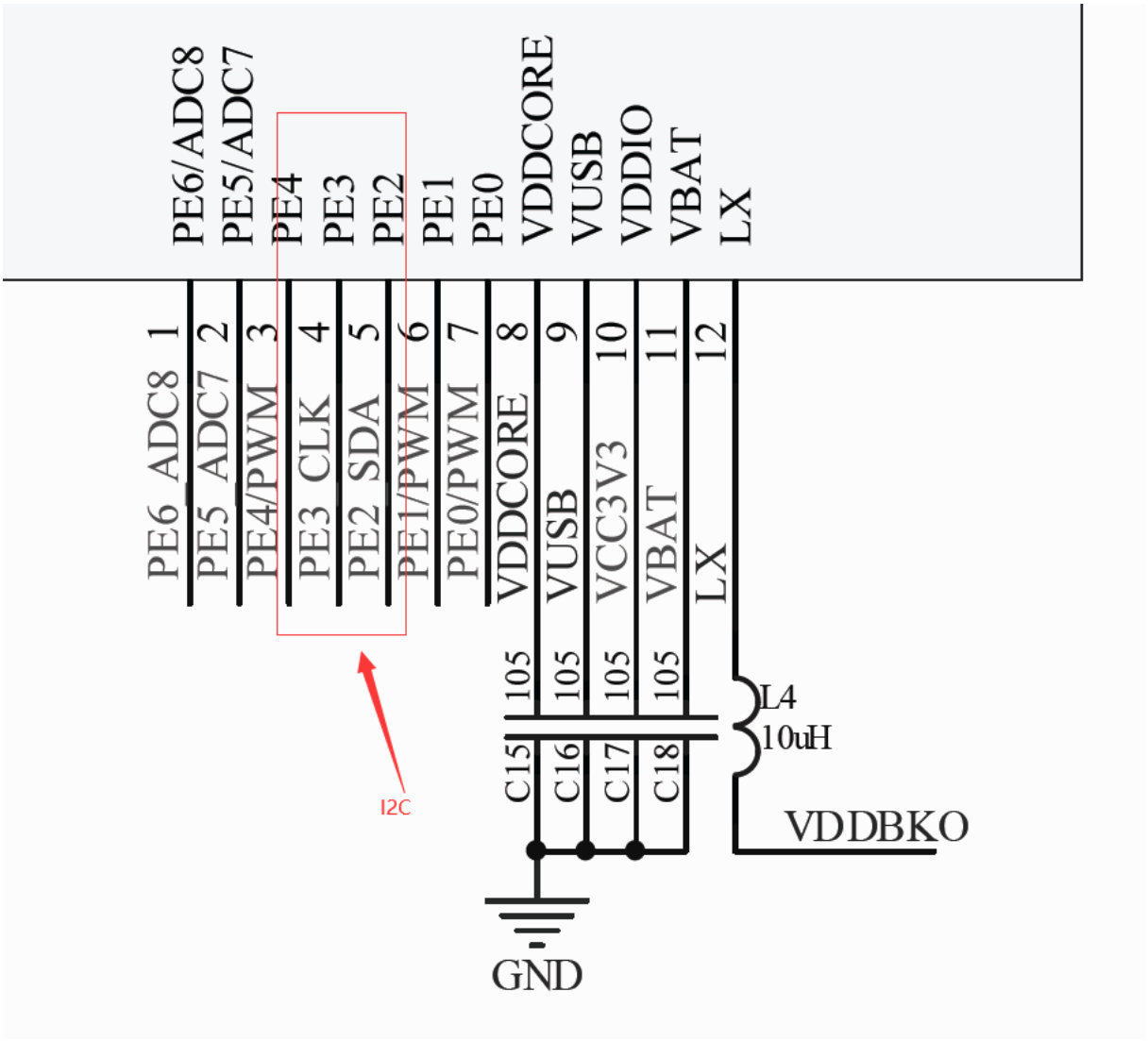
本章通过 RT-Thread Studio 配置 AB32VG1 片上外设 I2C 的引脚，控制 RGB 彩灯进行简单的颜色变换

1.2 模块介绍

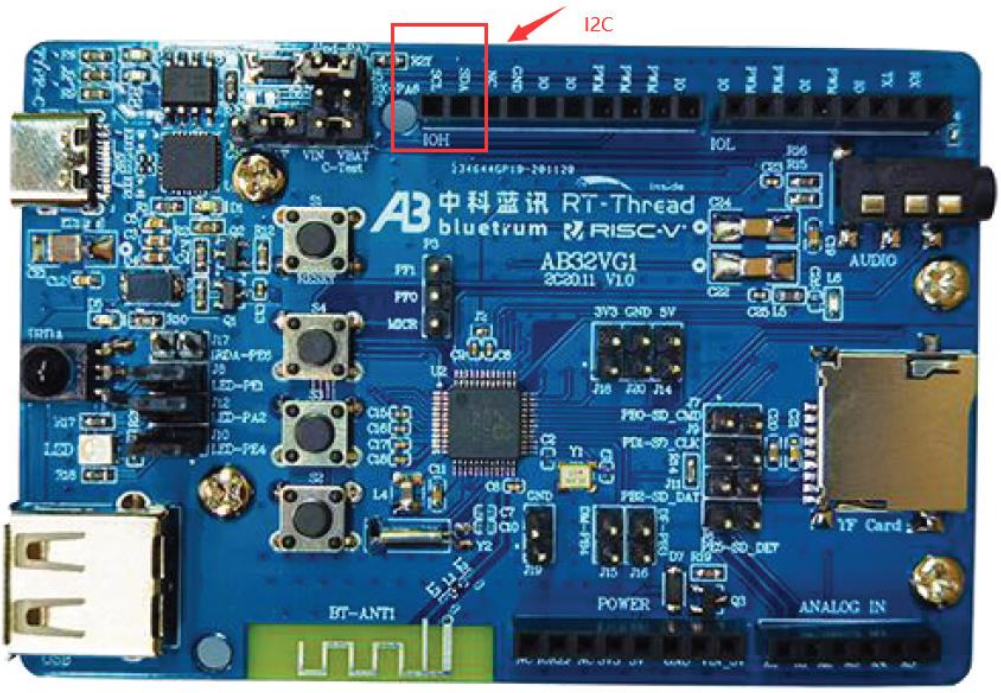
根据说明书，开发板上有一路 I2C

- 一路 IIC 接口
- 一路音频接口(美标 CTIA)
- 六路 ADC 输入引脚端子引出
- 六路 PWM 输出引脚端子引出
- 一个全彩 LED 灯模块，一个电源指示灯，三个烧录指示灯
- 一个 IRDA（红外接收端口）
- 一个 Reset 按键，三个功能按键(通用版为两个功能按键)
- 板子规格尺寸：6cm*9cm
- I/O 口通过 2.54MM 标准间距引出，同时兼容 Arduino Uno 扩展接口，方便二次开发

原理图如下：


PE6/ADC8
PE5/ADC7
PE4
PE3
PE2
PE1
PE0
VDDCORE
VUSB
VDDIO
VBAT
LX
PE6_ADC8 1
PE5_ADC7 2
PE4/PWM 3
PE3_CLK 4
PE2_SDA 5
PE1/PWM 6
PE0/PWM 7
VDDCORE 8
VUSB 9
VCC3V3 10
VBAT 11
LX 12
105
105
105
105
C15
C16
C17
C18
L4
10uH
VDDBKO
GND
I2C

开发板实物位置：

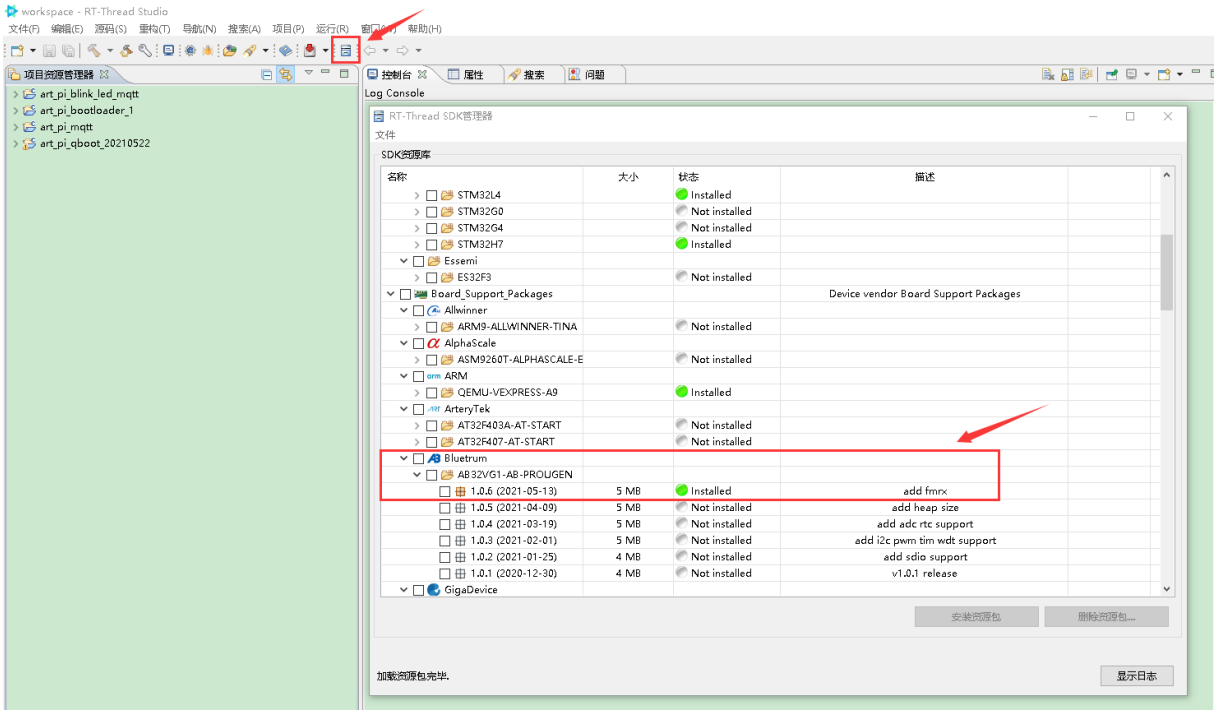


I2C 的 OLED 屏，芯片 SH1106

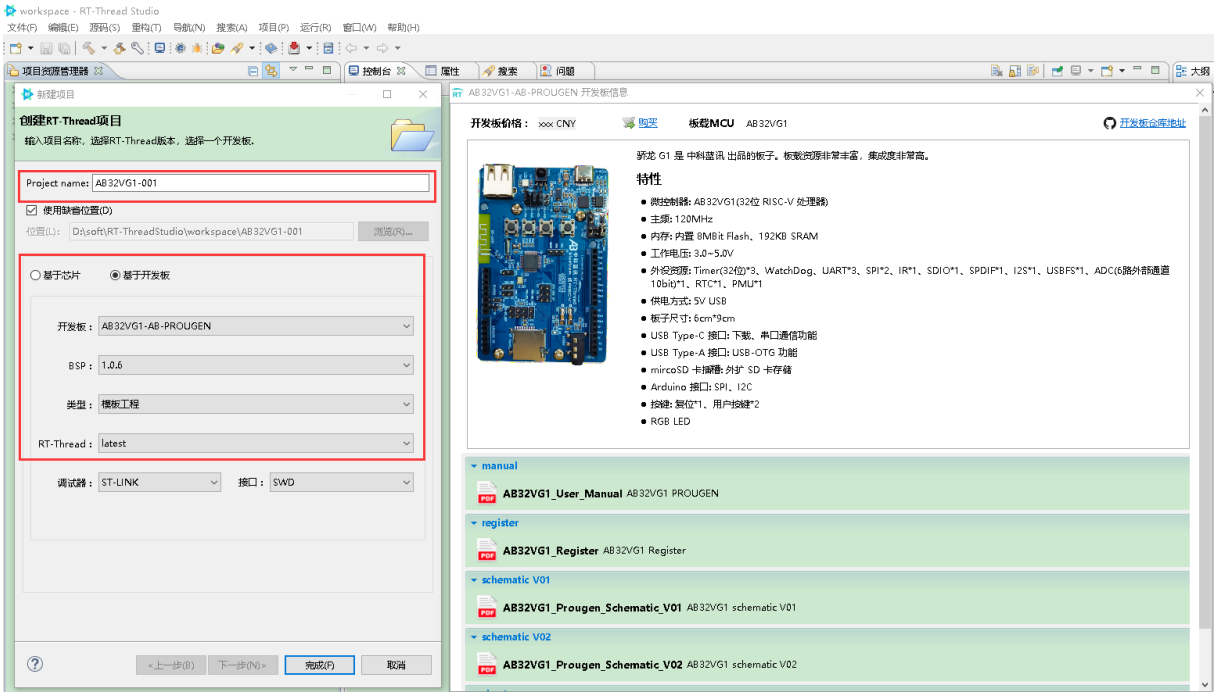
2. 步骤说明

2.1 创建工程

先要保证板子的 SDK 已下载好



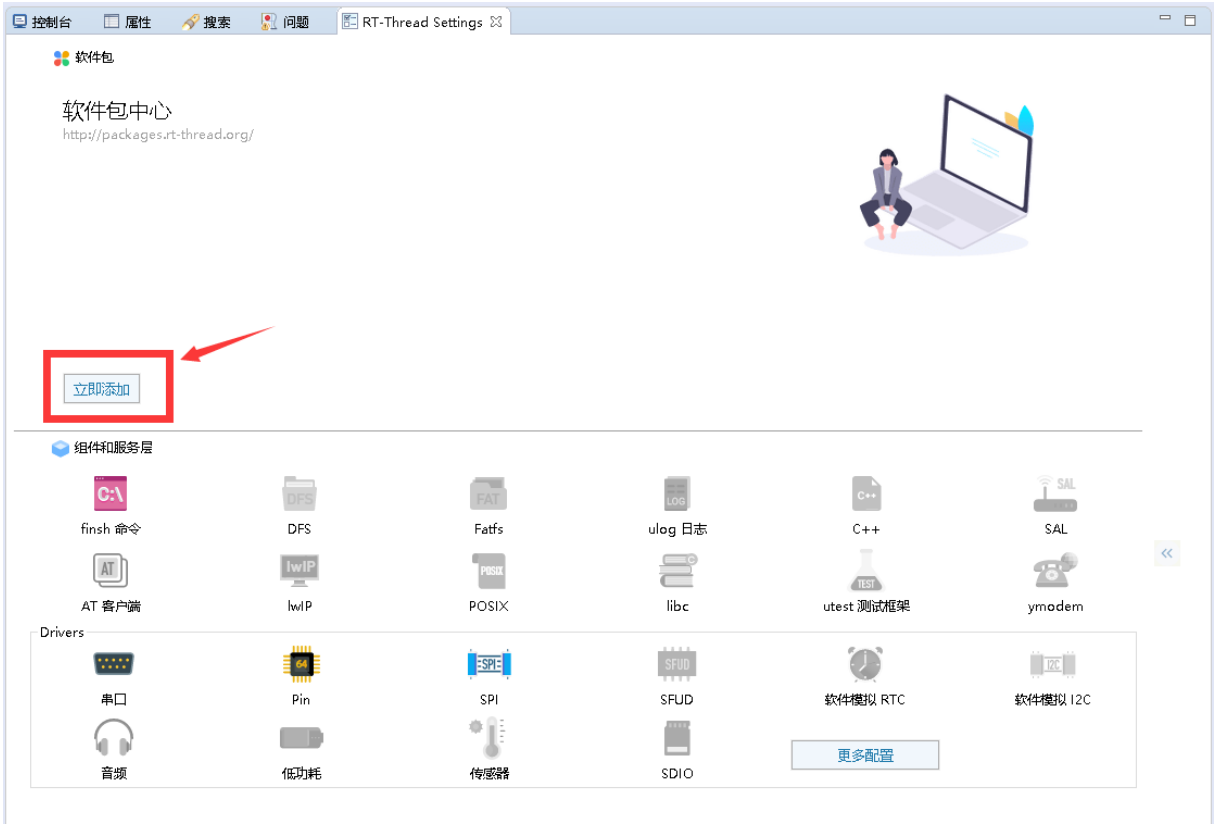
点击 文件-> 新建-> RT-Thread 项目



编译无报错，新建工程完成

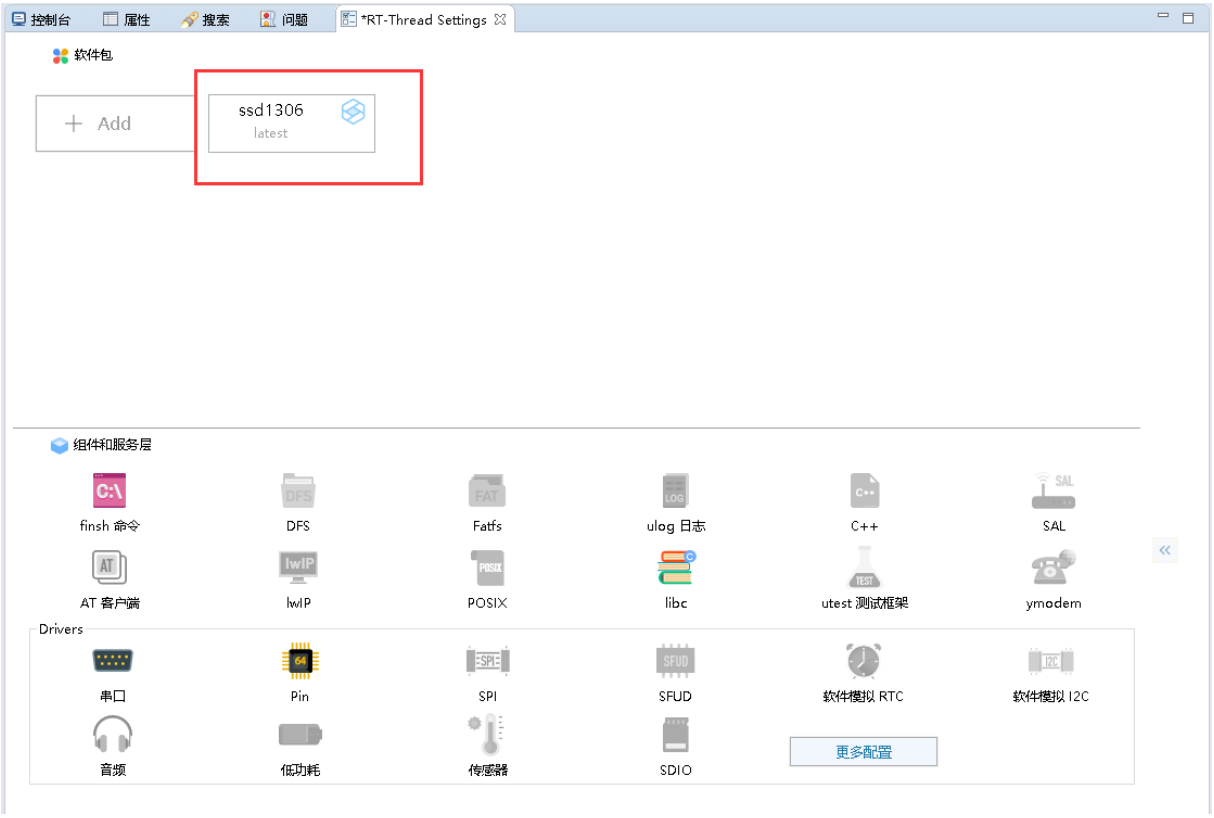
2.2 组件配置

打开 RT-Thread setting->立即添加 按钮

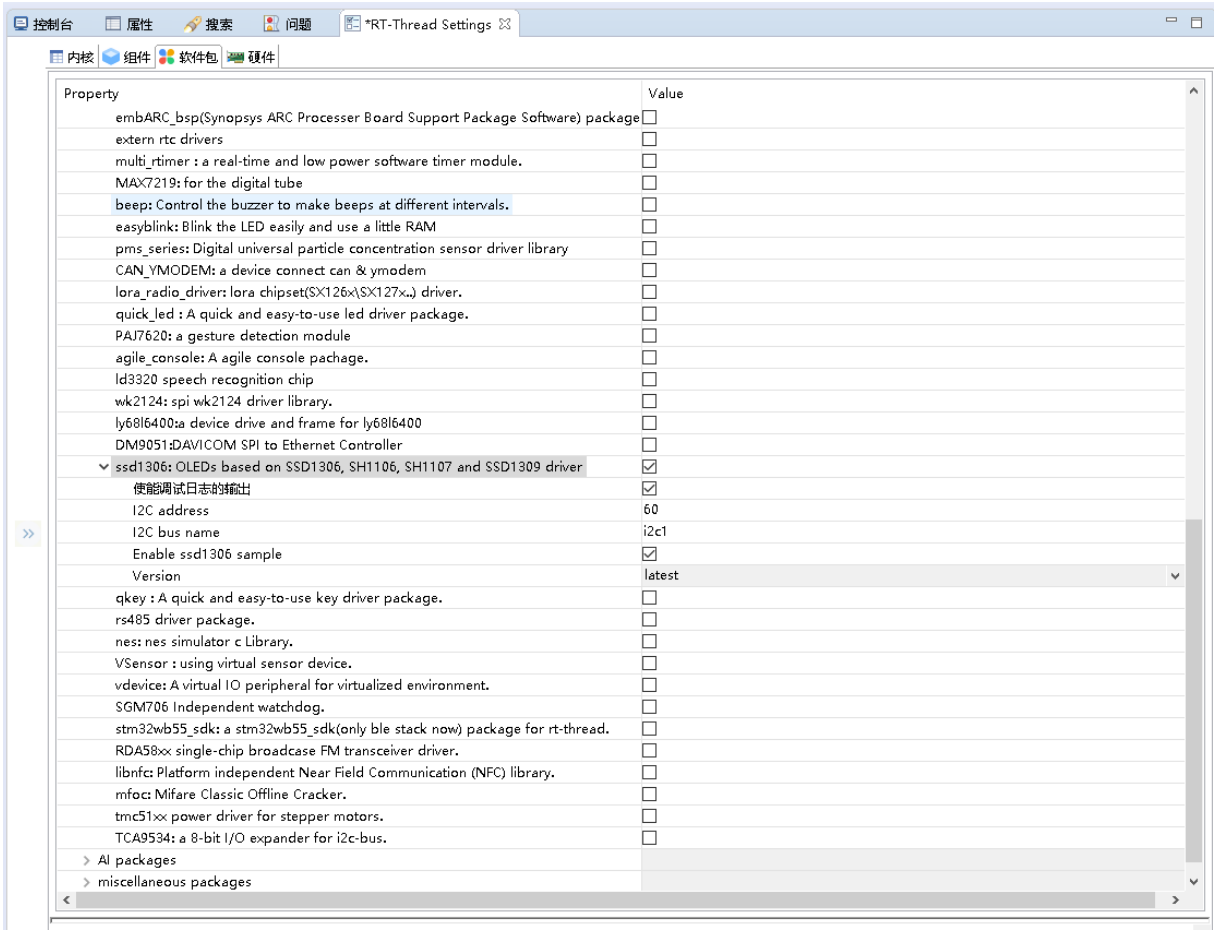


搜索框输入 SSD1306

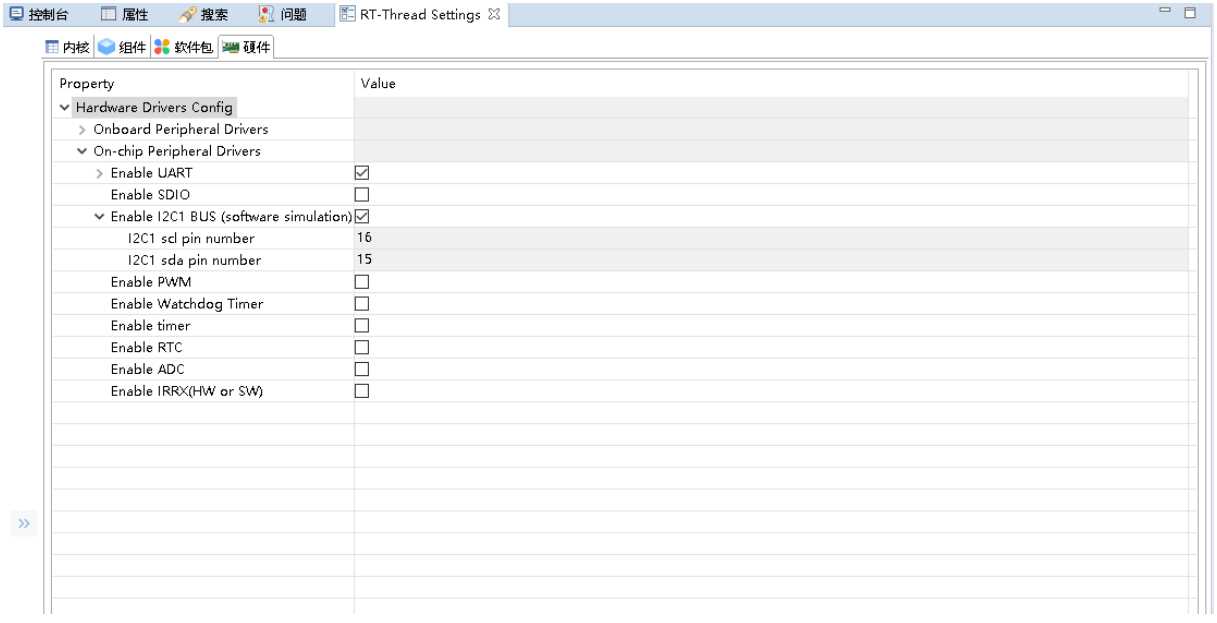
组件已添加



右键点击，选择->详细配置，软件包 选项卡配置如下



硬件 选项卡配置如下



保存，编译一下

2.3 代码检验

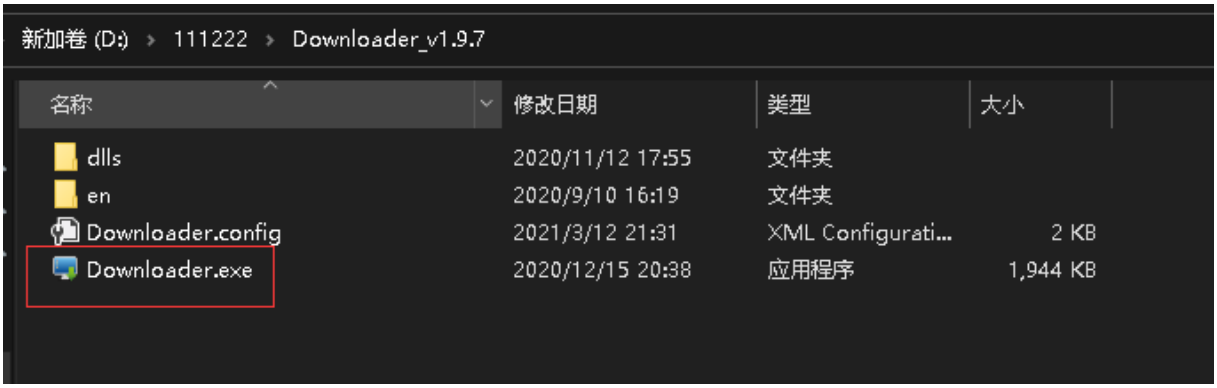
RT-ThreadStudio\workspace\AB32VG1-001\packages\ssd1306-latest\examples\ssd1306_tests.h 添加如下代码用于兼容 ssd1306 软件包中 HAL 代码部分

```
#ifdef AB32VG1_HAL_H__
#define HAL_GetTick()    rt_tick_get()
#define HAL_Delay(ms)    rt_thread_mdelay(ms)
#endif
```

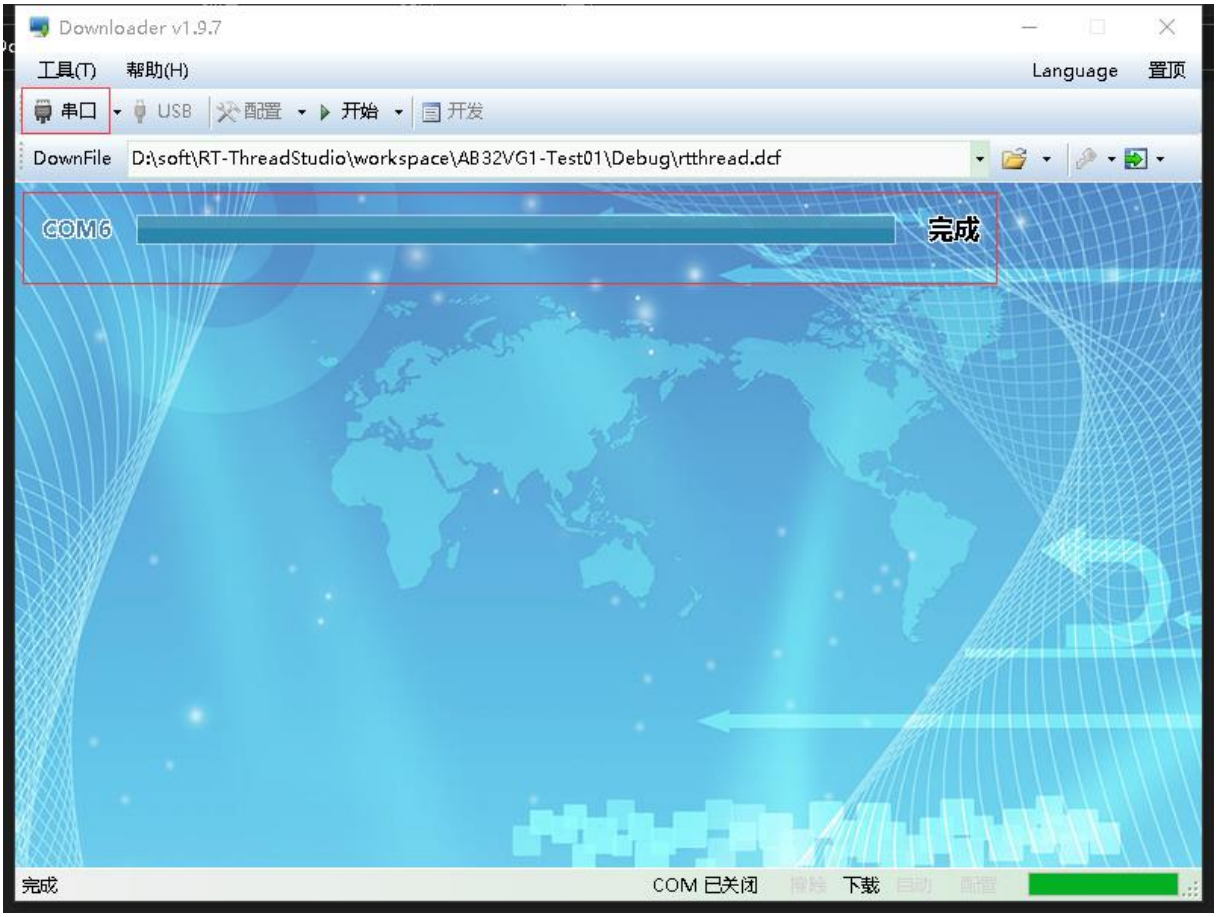
保存，编译一下，无报错

3. Downloaded 下载器使用

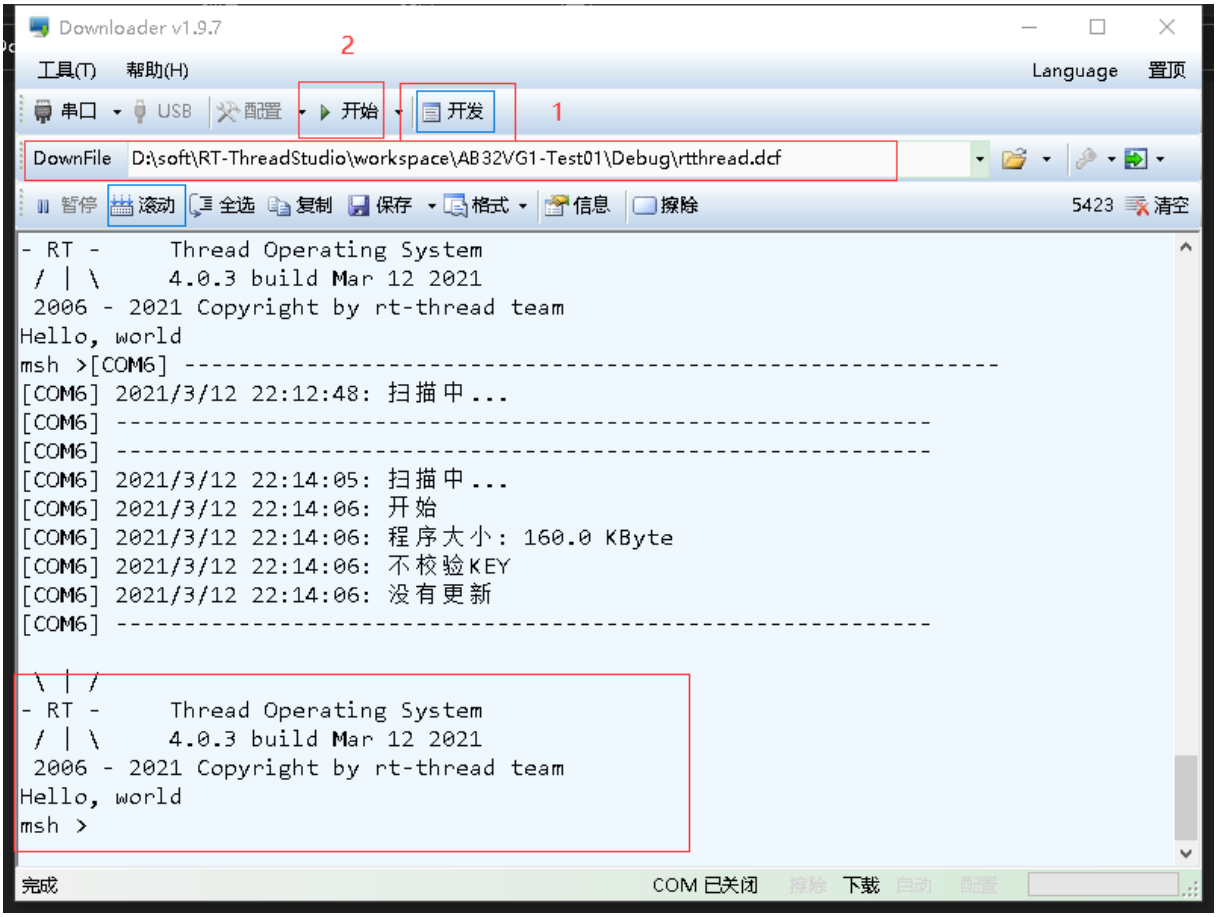
运行 Downloader.exe 下载器



点击串口按钮



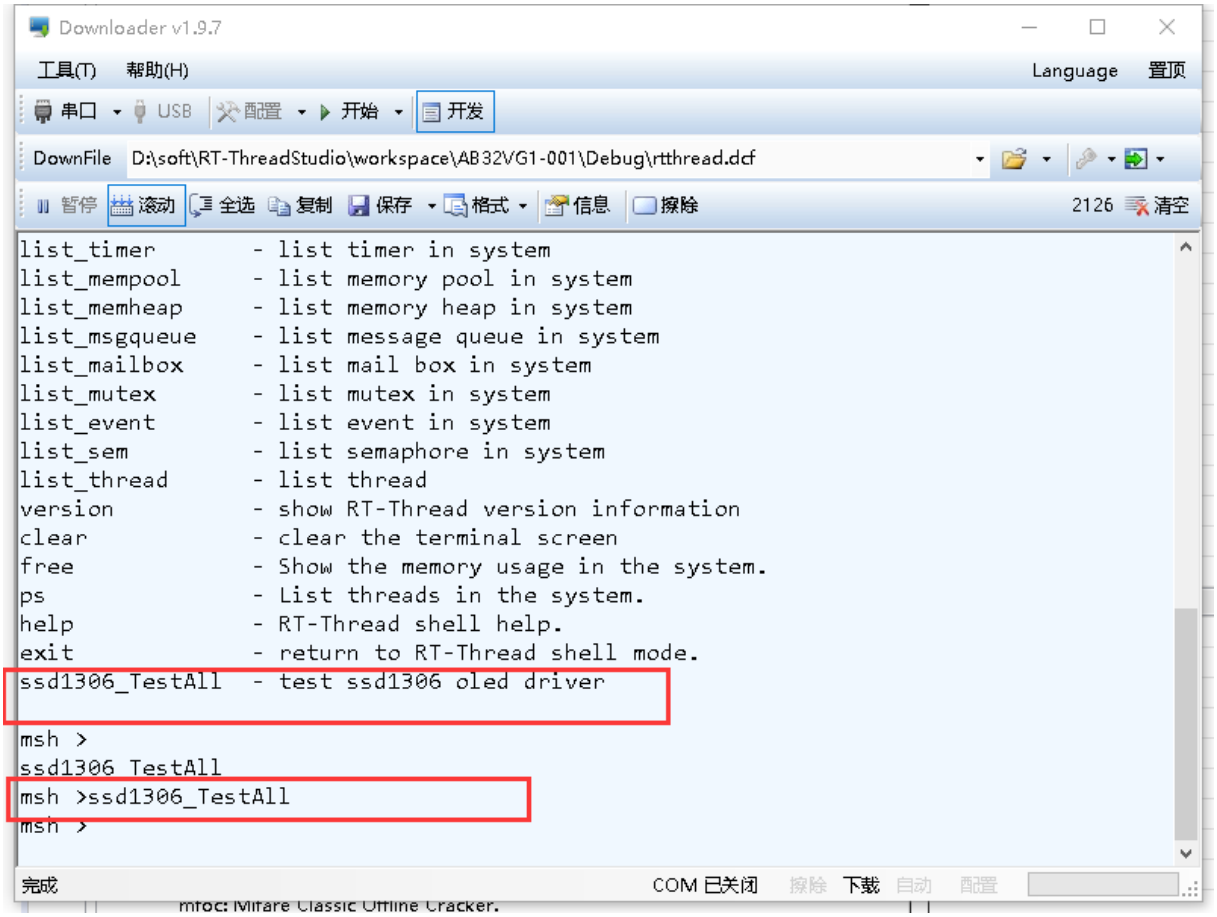
注意下图的操作顺序，不然不会出 rt-thread 的命令行



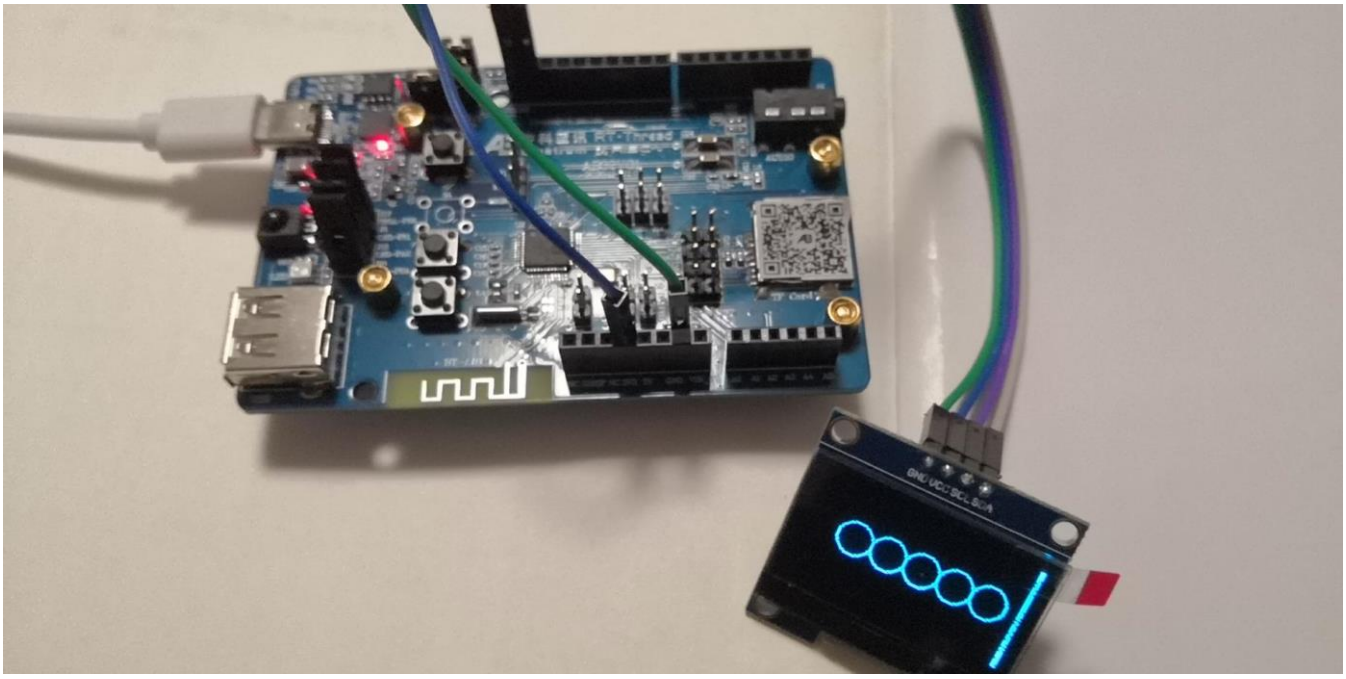
（**再次提醒注意串口驱动的安装最好是先用 type-C 接着板子，并 USB 连联接到电脑了安装驱动，如果提前没接板子就安装了驱动出现不正常，卸载再安装一遍即可**）比如下图 一直出现“扫描中”：


控制台
属性
搜索
问题
CDT Build Console [AB32VG1-Test01]
21:11:57 **** Incremental Build of configuration Debug for project AB32VG1-Test01 ****
make -j8 all2
riscv64-unknown-elf-size --format=berkeley "rtthread.elf"
text data bss dec hex filename
153842 0 37232 191074 2ea62 rtthread.elf
sh ../pre_build.sh
riscv32-elf-xmaker -b rtthread.xm
CODE SIZE: 160 KB
save file "rtthread.dcf" successful
riscv32-elf-xmaker -b download.xm
21:11:58 Build Finished. 0 errors, 0 warnings. (took 1s.205ms)
Downloader v1.9.7
工具(T)
帮助(H)
Language
置顶
串口
USB
配置
开始
开发
DownFile
D:\soft\RT-ThreadStudio\workspace\AB32VG1-Test01\Debug\rtthread.dcf
COM6
扫描中
截图(Alt + A)
COM6 打开成功
COM 已关闭
下载

msh 命令行下执行


Downloader v1.9.7
工具(T) 帮助(H)
Language 置顶
串口 USB 配置 开始 开发
DownFile D:\soft\RT-ThreadStudio\workspace\AB32VG1-001\Debug\rtthread.dcf
暂停 滚动 全选 复制 保存 格式 信息 擦除 2126 清空
list_timer - list timer in system
list_mempool - list memory pool in system
list_memheap - list memory heap in system
list_msgqueue - list message queue in system
list_mailbox - list mail box in system
list_mutex - list mutex in system
list_event - list event in system
list_sem - list semaphore in system
list_thread - list thread
version - show RT-Thread version information
clear - clear the terminal screen
free - Show the memory usage in the system.
ps - List threads in the system.
help - RT-Thread shell help.
exit - return to RT-Thread shell mode.
ssd1306_TestAll - test ssd1306 oled driver
msh >
ssd1306 TestAll
msh >ssd1306_TestAll
msh >
完成
COM 已关闭
擦除 下载 自动 配置

查看板子运行情况



4.章节总结

本章节我们学会了如何在 RT-Thread 上配置 I2C 以及 SSD1306 组件的用法，代码中一些函数要改成 调用 rtt 框架里的自带函数。

四、中科蓝讯 AB32VG1 上的模拟 SPI 实践

1. 前言说明

sdk 目前还不支持 spi，没有 spi 就失去了很多乐趣，如 easyflash、spi 的屏幕，蓝讯的这次活动我接到了模拟 spi 的任务，下面介绍如何写 rt-thread 的设备驱动层的驱动。（rt-thread 的设备 I/O 模型有设备管理层、设备驱动框架层、设备驱动层），我写过一篇使用 timer 的，就属于最接近用户那一层-设备管理层，我们调用 rt_device_find 根据名称查找句柄，之后根据句柄执行 rt_device_read、rt_device_write、rt_device_control 语句完成与底层设备的交互，而最底层的 timer 已经由中科蓝讯的工程师完成了。而这次的模拟 spi 则是写设备驱动层。

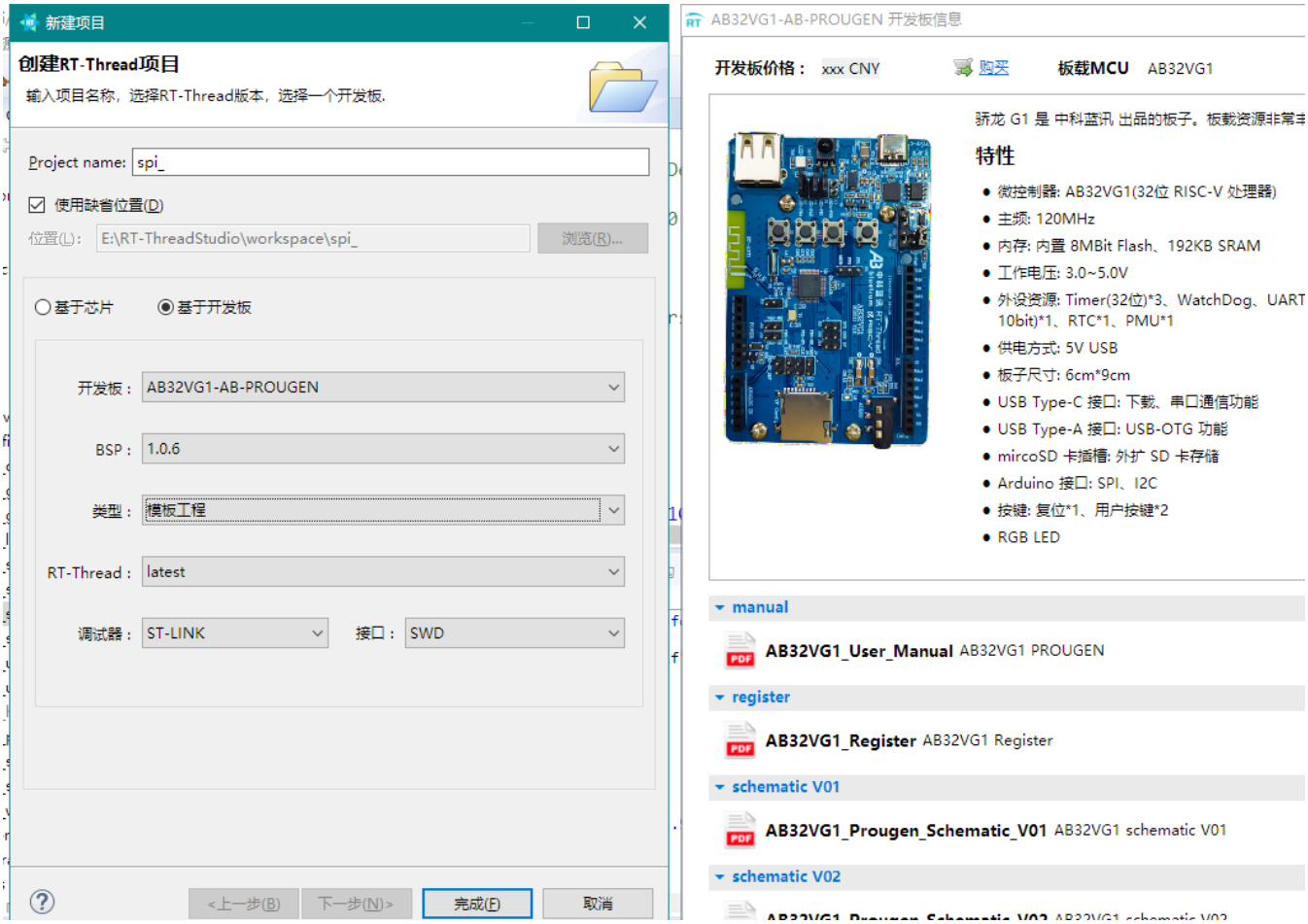
设备驱动层的编写有两步：

实现 spi 的驱动程序（模拟 spi 主要通过 io 口模拟 spi 的时序）

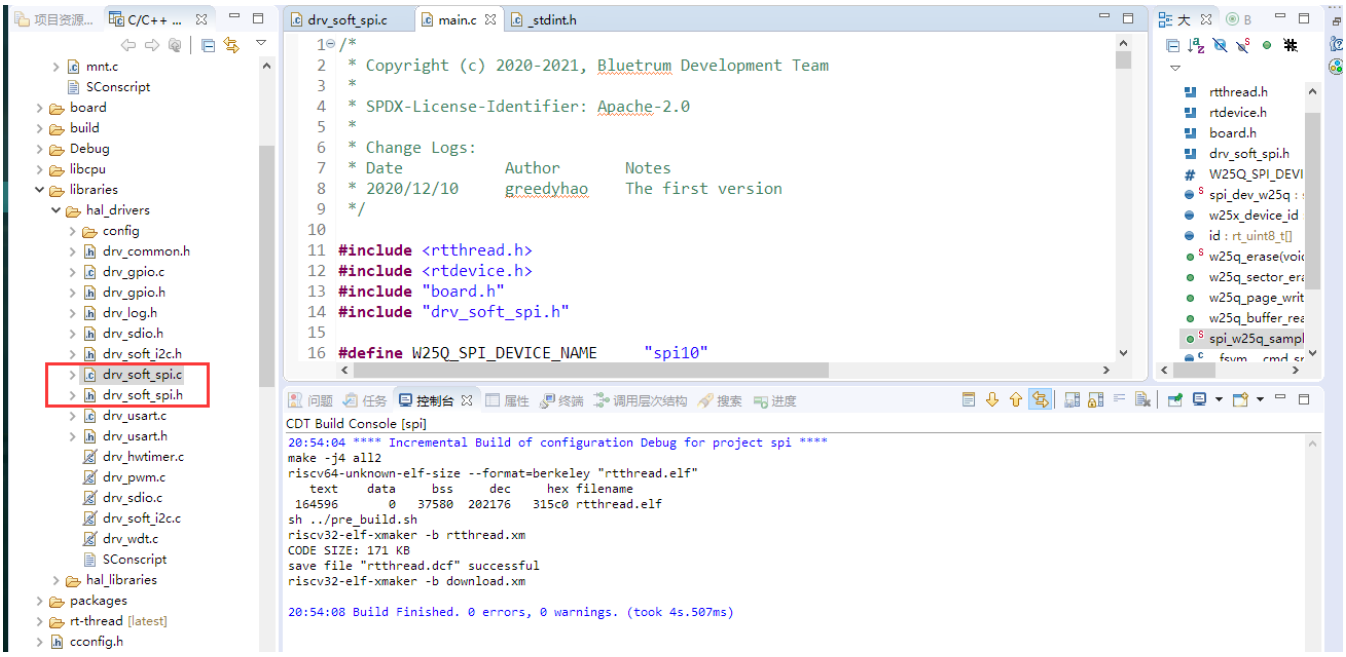
将裸机程序按 rt-thread 的设备驱动框架封装（主要是自己写的函数原型与 rt-thread 的接口对应上）

2. 步骤说明

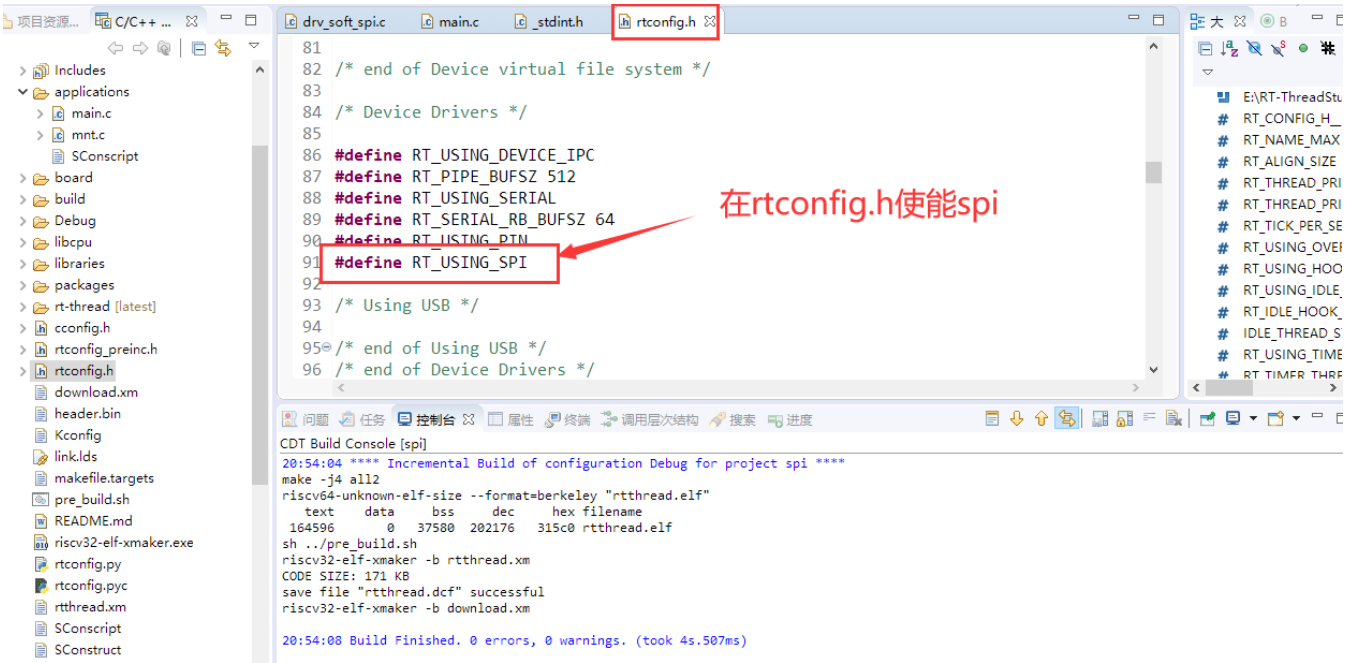
2.1 新建 rt-thread studio 开发板工程



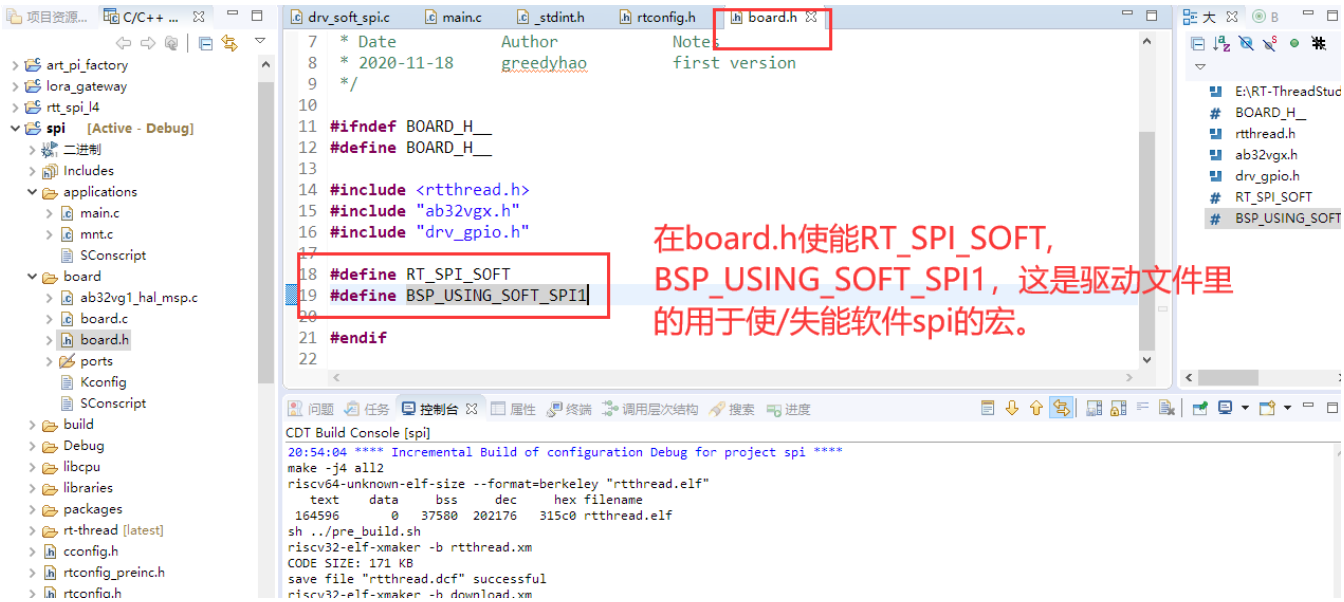
2.2 在 library 下添加 drv_soft_spi.c 和 drv_soft_spi.h



2.3 在 rtconfig.h 添加 RT_USING_SPI



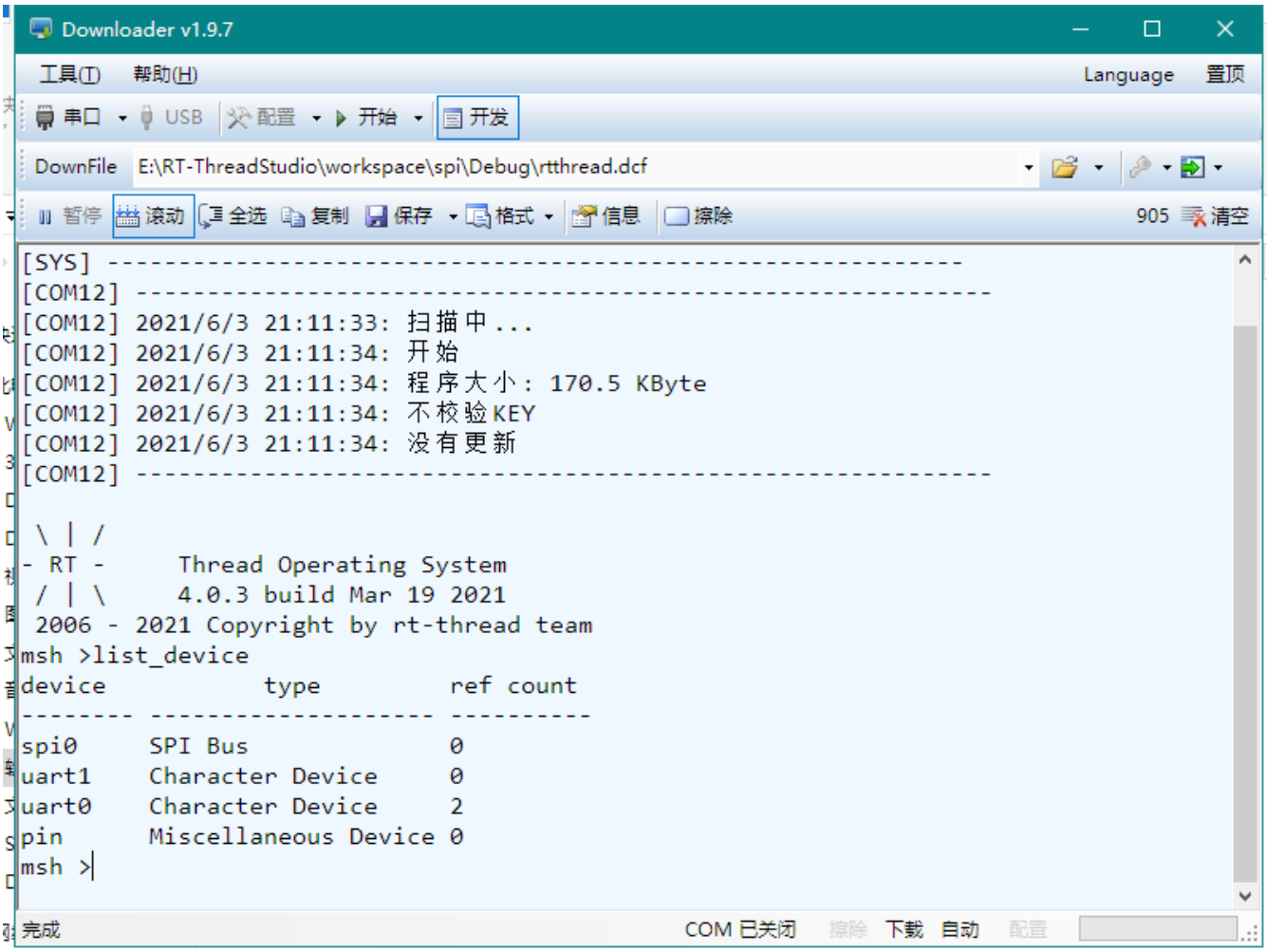
2.4 在 board.h 添加 RT_USING_SOFT_SPI



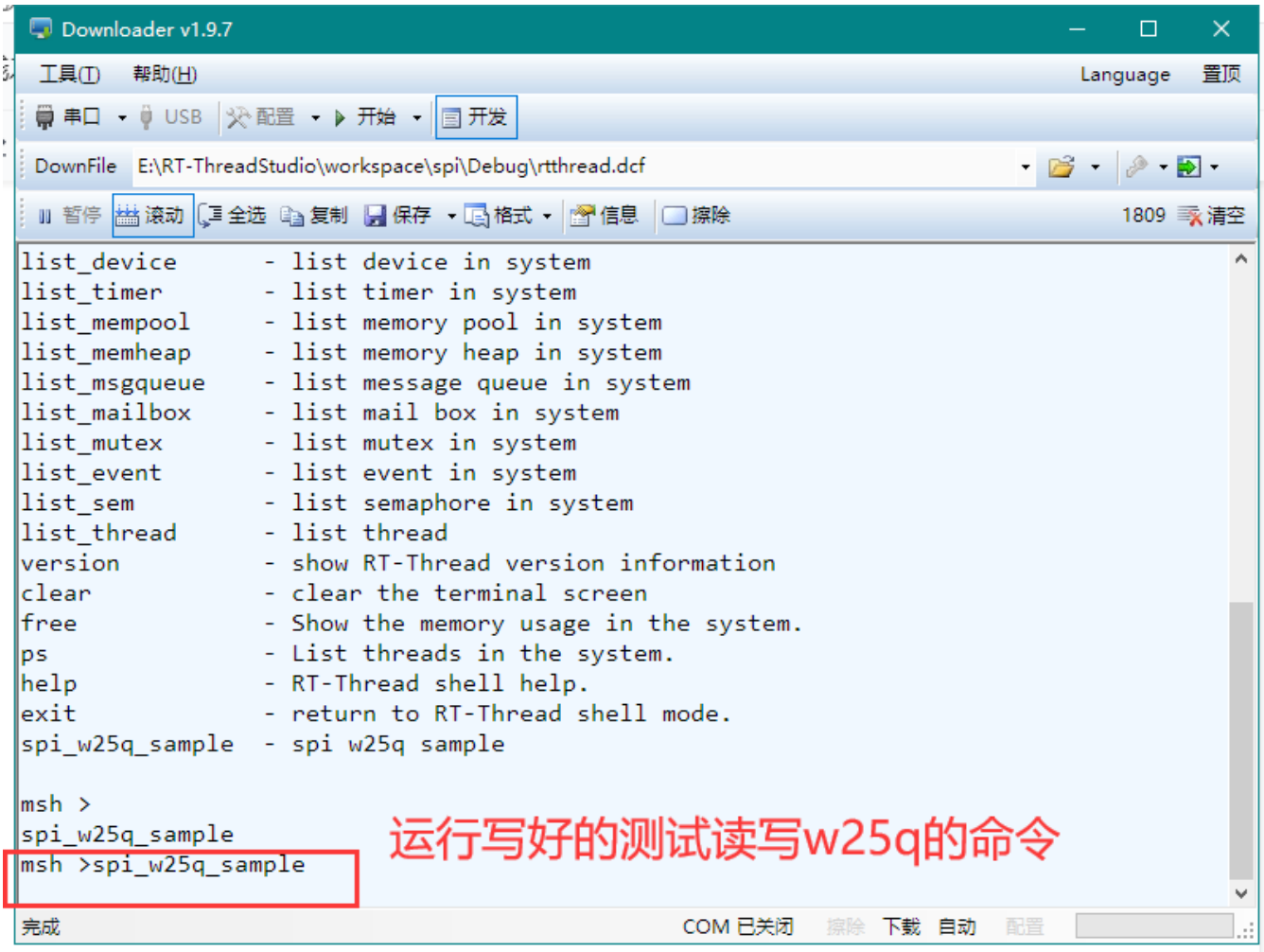
2.5 驱动验证

驱动完成了，我做的测试是读写 w25q64 来验证，你将神奇的看到，自己通过调用设备管理层的接口居然能够控制到底层的硬件，更加深刻的认识到设备 I/O 模型的三层框架。

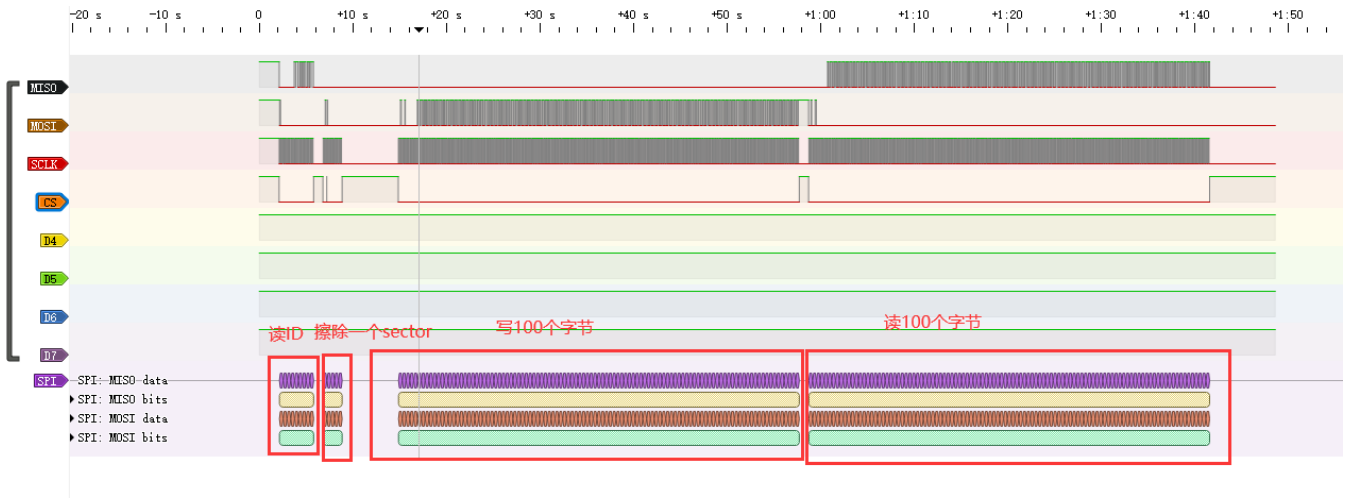
2.5.1 在终端中输入 list_device，查看自己写的设备驱动是否注册到 rt-thread。



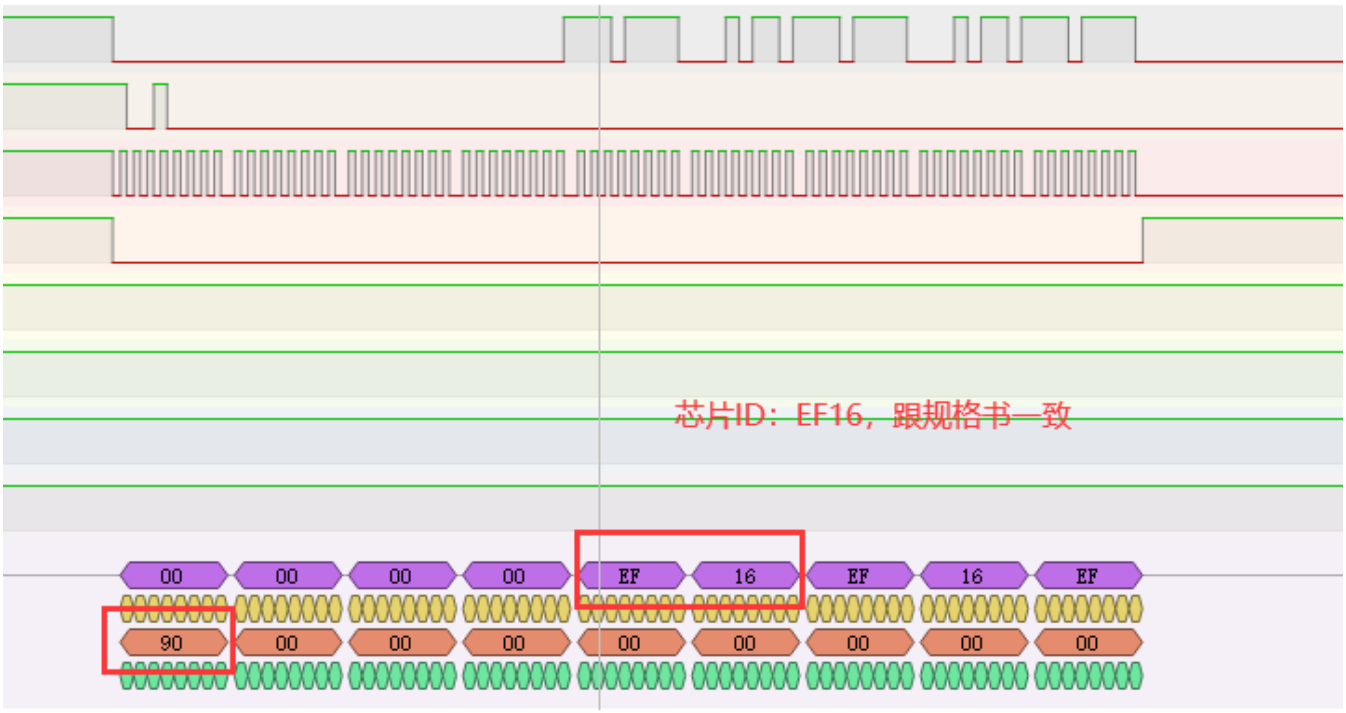
2.5.2 读写 spi flash 测试：先擦除 flash、再写入 100 个字节、再读出 100 个字节。



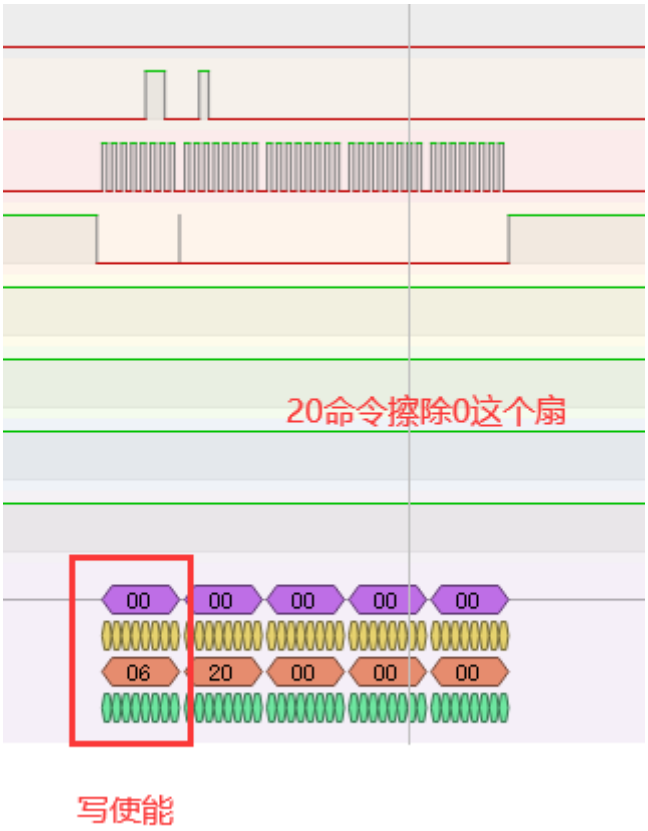
逻辑分析仪查看整个过程：读芯片 ID、擦除一个扇区、写 100 个字节、再读出来看是否跟写入的一致。



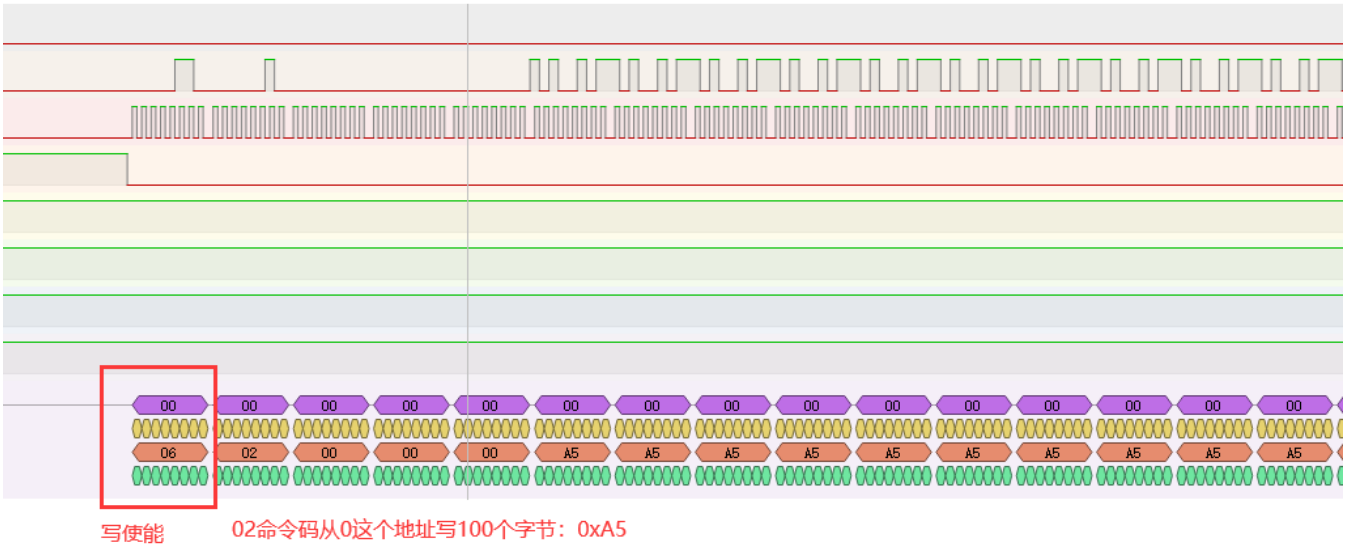
过程一：读 ID



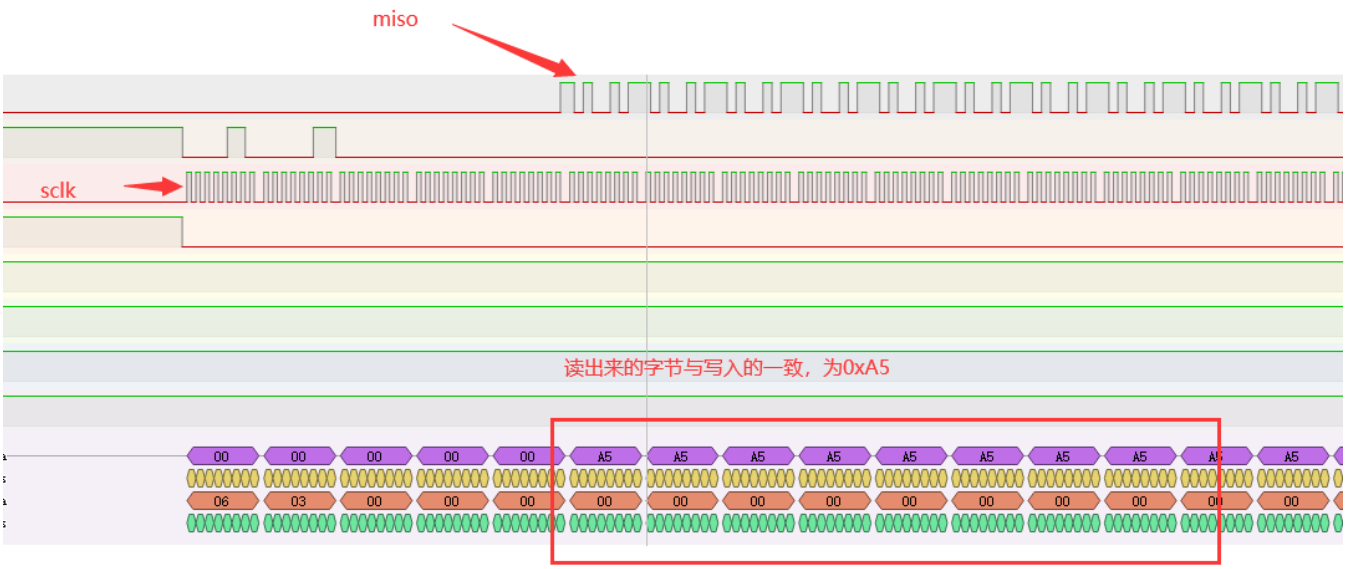
过程二：擦除一个扇区



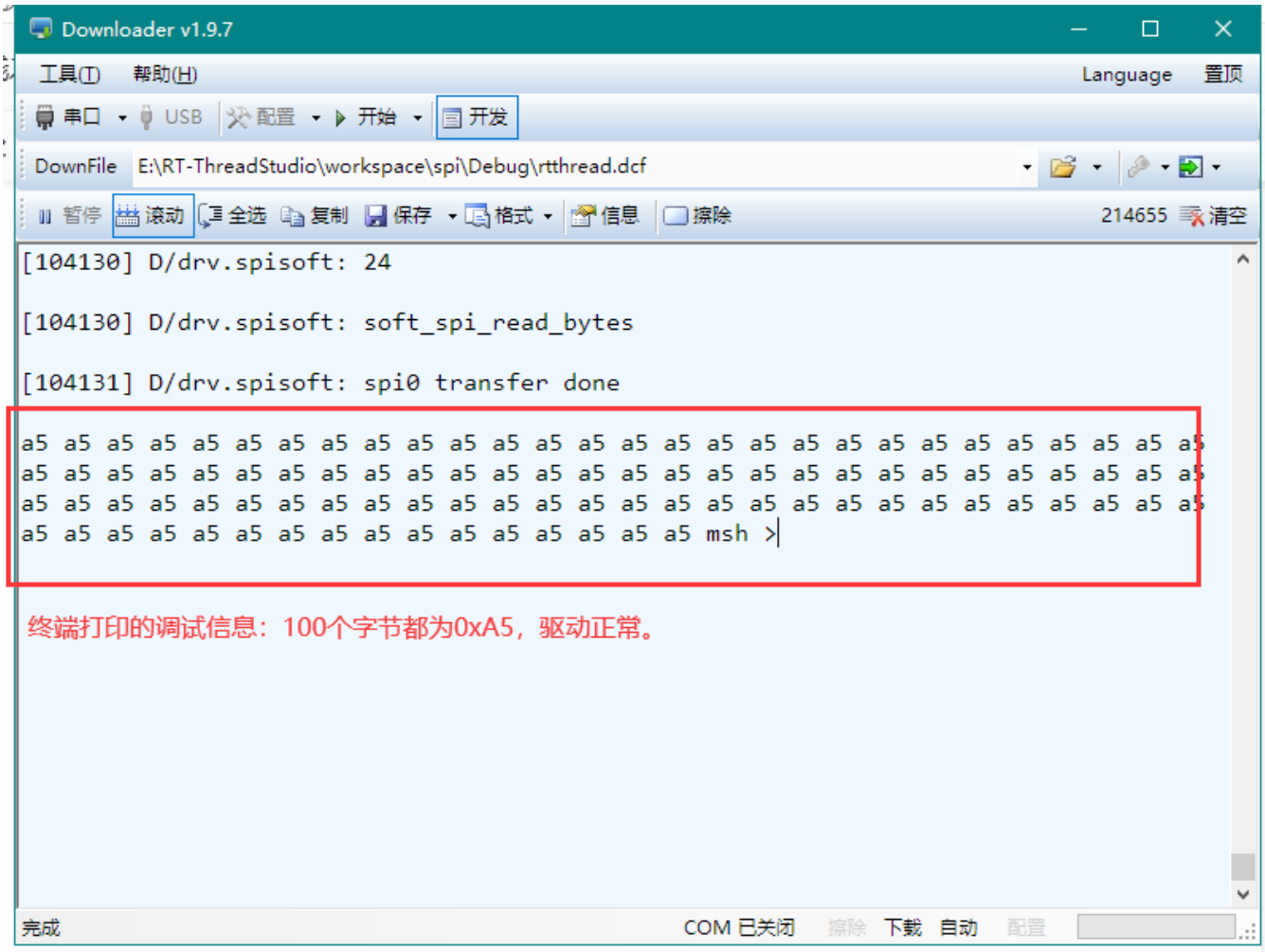
过程三：写字节



过程四：读字节


miso
sclk
读出来的字节与写入的一致，为0xA5

终端打印的读取信息


Downloader v1.9.7
工具(T) 帮助(H) Language 置顶
串口 USB 配置 开始 开发
DownFile E:\RT-ThreadStudio\workspace\spi\Debug\rtthread.dcf
暂停 滚动 全选 复制 保存 格式 信息 擦除 214655 清空
[104130] D/drv.spisoft: 24
[104130] D/drv.spisoft: soft_spi_read_bytes
[104131] D/drv.spisoft: spi0 transfer done
a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5
a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5
a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5
a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 a5 msh >
终端打印的调试信息：100个字节都为0xA5，驱动正常。
完成 COM 已关闭 擦除 下载 自动 配置

3. 代码验证

3.1 drv_soft_spi.c

```
/*
 * Change Logs:
 * Date           Author       Notes
 * 2021-06-03     qwz          first version
 */

#include "board.h"

#ifdef RT_USING_SPI
#ifdef RT_SPI_SOFT

#include "spi.h"
#include "drv_soft_spi.h"

#include <string.h>

#define DRV_DEBUG
#define LOG_TAG                "drv.spisoft"
#include <drv_log.h>

enum{
#ifdef BSP_USING_SOFT_SPI1
    SOFT_SPI1_INDEX,
#endif
};
//PB2 10 ;PE5 18;PE6 19;PB1 9;
#define SOFT_SPI1_BUS_CONFIG {                          \
    .mosi_pin = 18,                                     \
    .miso_pin = 10,                                     \
    .sclk_pin = 9,                                  \
    .bus_name = "spi0",                                 \
}

static struct ab32_soft_spi_config soft_spi_config[] ={
#ifdef BSP_USING_SOFT_SPI1
    SOFT_SPI1_BUS_CONFIG,
#endif
};
```

```
static struct ab32_soft_spi soft_spi_bus_obj[sizeof(soft_spi_config) / sizeof(soft_spi_config[0])] = {0};

static rt_err_t ab32_spi_init(struct ab32_soft_spi *spi_drv, struct rt_spi_configuration *cfg){
    RT_ASSERT(spi_drv != RT_NULL);
    RT_ASSERT(cfg != RT_NULL);

    //mode = master
    if (cfg->mode & RT_SPI_SLAVE){
        return RT_EIO;
    }
    else
        spi_drv->mode = RT_SPI_MASTER;

    if (cfg->mode & RT_SPI_3WIRE){
        return RT_EIO;
    }

    if (cfg->data_width == 8 || cfg->data_width == 16)
        spi_drv->data_width = cfg->data_width;
    else{
        return RT_EIO;
    }


    if (cfg->mode & RT_SPI_CPHA){
        spi_drv->cpha = 1;
    }
    else{
        spi_drv->cpha = 0;
    }

    if (cfg->mode & RT_SPI_CPOL){
        spi_drv->cpol = 1;
    }
    else{
        spi_drv->cpol = 0;
    }

    if (cfg->mode & RT_SPI_NO_CS){
    }
    else{
    }

    if (cfg->max_hz >= 1200000){
```

```
        spi_drv->spi_delay = 0;
    }else if (cfg->max_hz >= 1000000){
        spi_drv->spi_delay = 8;
    }else if (cfg->max_hz >= 830000){
        spi_drv->spi_delay = 16;
    }
    else {
        spi_drv->spi_delay = 24;
    }

    LOG_D("SPI limiting freq: %d, BaudRatePrescaler: %d",
          cfg->max_hz,
          spi_drv->max_hz);

    if (cfg->mode & RT_SPI_MSB){
        spi_drv->msb = 1;
    }
    else{
        spi_drv->msb = 0;
    }

    rt_pin_mode(spi_drv->config->mosi_pin,PIN_MODE_OUTPUT_OD);
    rt_pin_write(spi_drv->config->mosi_pin,PIN_HIGH);
    rt_pin_mode(spi_drv->config->miso_pin,PIN_MODE_INPUT_PULLDOWN);

    rt_pin_mode(spi_drv->config->sclk_pin,PIN_MODE_OUTPUT_OD);
    if(spi_drv->cpol)
        rt_pin_write(spi_drv->config->sclk_pin,PIN_HIGH);
    else
        rt_pin_write(spi_drv->config->sclk_pin,PIN_LOW);
    LOG_D("%s init done", spi_drv->config->bus_name);
    return RT_EOK;
}

static inline void spi_delay(rt_uint32_t us){
    rt_thread_mdelay(us);
}

static rt_uint32_t soft_spi_read_write_bytes(struct ab32_soft_spi *spi_drv, rt_uint8_t* send_buff,
rt_uint8_t* recv_buff, rt_uint32_t len){
    rt_uint8_t dataIndex = 0;
    rt_uint8_t time = 1;
    for(rt_uint32_t i = 0; i<len; i++){
        if(spi_drv->cpha){   //CPHA=1
            if(rt_pin_read(spi_drv->config->sclk_pin))
```

```
            {
                rt_pin_write(spi_drv->config->sclk_pin,PIN_LOW);
            }
            else {
                rt_pin_write(spi_drv->config->sclk_pin,PIN_HIGH);
            }
        }
        if(spi_drv->data_width == 16)
            time = 2;
        do{
            for(rt_uint8_t j = 0; j < 8; j++){
                if ((send_buff[dataIndex] & 0x80) != 0){
                    rt_pin_write(spi_drv->config->mosi_pin,PIN_HIGH);
                }else{
                    rt_pin_write(spi_drv->config->mosi_pin,PIN_LOW);
                }
                send_buff[dataIndex] <<= 1;
                spi_delay(spi_drv->spi_delay);
                if(rt_pin_read(spi_drv->config->sclk_pin))
                {
                    rt_pin_write(spi_drv->config->sclk_pin,PIN_LOW);
                }
                else {
                    rt_pin_write(spi_drv->config->sclk_pin,PIN_HIGH);
                }

                recv_buff[dataIndex] <<= 1;
                if (rt_pin_read(spi_drv->config->miso_pin))
                    recv_buff[dataIndex] |= 0x01;
                spi_delay(spi_drv->spi_delay);
                if(time != 0 || j != 7){
                    if(rt_pin_read(spi_drv->config->sclk_pin))
                    {
                        rt_pin_write(spi_drv->config->sclk_pin,PIN_LOW);
                    }
                    else {
                        rt_pin_write(spi_drv->config->sclk_pin,PIN_HIGH);
                    }
                }
            }
            dataIndex++;
        }while((--time)==1);
        time = 1;
        spi_delay(spi_drv->spi_delay);
    }
```

```
    return len;
}

static rt_uint32_t soft_spi_read_bytes(struct ab32_soft_spi *spi_drv, rt_uint8_t* recv_buff, rt_uint32_t len){
    rt_uint8_t send_buff = spi_drv->dummy_data;
    rt_uint32_t dataIndex = 0;
    rt_uint8_t time = 1;
        if(spi_drv->cpha){   //CPHA=1
            if(rt_pin_read(spi_drv->config->sclk_pin))
            {
                rt_pin_write(spi_drv->config->sclk_pin,PIN_LOW);
            }
            else {
                rt_pin_write(spi_drv->config->sclk_pin,PIN_HIGH);
            }
        }
    for(rt_uint32_t i = 0; i<len; i++){

        if(spi_drv->data_width == 16)
            time = 2;
        do{
            for(rt_uint8_t j = 0; j < 8; j++){
                if ((send_buff & 0x80) != 0){
                    rt_pin_write(spi_drv->config->mosi_pin,PIN_HIGH);
                }else{
                    rt_pin_write(spi_drv->config->mosi_pin,PIN_LOW);
                }
                send_buff <<= 1;
                spi_delay(spi_drv->spi_delay);
                if(rt_pin_read(spi_drv->config->sclk_pin))
                {
                    rt_pin_write(spi_drv->config->sclk_pin,PIN_LOW);
                }
                else {
                    rt_pin_write(spi_drv->config->sclk_pin,PIN_HIGH);
                }

                *recv_buff <<= 1;
                if (rt_pin_read(spi_drv->config->miso_pin))
                {
                    *recv_buff |= 0x01;
                }
                else
                {
```

```
                        *recv_buff &= 0xfe;
                    }
                    spi_delay(spi_drv->spi_delay);
                    if(time != 0 || j != 7){
                        if(rt_pin_read(spi_drv->config->sclk_pin))
                        {
                            rt_pin_write(spi_drv->config->sclk_pin,PIN_LOW);
                        }
                        else {
                            rt_pin_write(spi_drv->config->sclk_pin,PIN_HIGH);
                        }
                    }
                }
                recv_buff ++;
                dataIndex++;
            }while((--time)==1);
            time = 1;
            spi_delay(spi_drv->spi_delay);
            LOG_D("DONE ONE BYTE %d",dataIndex);
            LOG_D("%d",spi_drv->spi_delay);
        }
        return len;
}


static rt_uint32_t soft_spi_write_bytes(struct ab32_soft_spi *spi_drv, rt_uint8_t* send_buff, rt_uint32_t
len){
        rt_uint8_t recv_buff = 0;
        rt_uint32_t dataIndex = 0;
        rt_uint8_t time = 1;
        LOG_D("%x",send_buff[0]);
        if(spi_drv->cpha){   //CPHA=1
                if(rt_pin_read(spi_drv->config->sclk_pin))
                {
                    rt_pin_write(spi_drv->config->sclk_pin,PIN_LOW);
                }
                else {
                    rt_pin_write(spi_drv->config->sclk_pin,PIN_HIGH);
                }
            }
        for(uint32_t i = 0; i<len; i++){

            if(spi_drv->data_width == 16)
                time = 2;
            do{
```

```
            for(rt_uint8_t j = 0; j < 8; j++){
                if ((send_buff[dataIndex] & 0x80) != 0){
                    rt_pin_write(spi_drv->config->mosi_pin,PIN_HIGH);
                    LOG_D("PIN_HIGH");
                }else{
                    rt_pin_write(spi_drv->config->mosi_pin,PIN_LOW);
                    LOG_D("PIN_LOW");
                }
                send_buff[dataIndex] <<= 1;
                spi_delay(spi_drv->spi_delay);
                if(rt_pin_read(spi_drv->config->sclk_pin))
                {
                    rt_pin_write(spi_drv->config->sclk_pin,PIN_LOW);
                }
                else {
                    rt_pin_write(spi_drv->config->sclk_pin,PIN_HIGH);
                }

                recv_buff <<= 1;
                if (rt_pin_read(spi_drv->config->miso_pin))
                    recv_buff |= 0x01;
                spi_delay(spi_drv->spi_delay);
                if(time != 0 || j != 7){
                    if(rt_pin_read(spi_drv->config->sclk_pin))
                    {
                        rt_pin_write(spi_drv->config->sclk_pin,PIN_LOW);
                    }
                    else {
                        rt_pin_write(spi_drv->config->sclk_pin,PIN_HIGH);
                    }
                }
            }
            dataIndex++;
        }while((--time)==1);
        time = 1;
        spi_delay(spi_drv->spi_delay);
    }
    return len;
}

static rt_uint32_t spixfer(struct rt_spi_device *device, struct rt_spi_message *message){
    rt_uint32_t state;
    rt_size_t message_length;
    rt_uint8_t *recv_buf;
    const rt_uint8_t *send_buf;
```

```
    rt_uint8_t pin = rt_pin_get("PE.6");

    RT_ASSERT(device != RT_NULL);
    RT_ASSERT(device->bus != RT_NULL);
    RT_ASSERT(device->bus->parent.user_data != RT_NULL);
    RT_ASSERT(message != RT_NULL);

    struct ab32_soft_spi *spi_drv =    rt_container_of(device->bus, struct ab32_soft_spi, spi_bus);
    struct ab32_soft_spi_pin *cs = device->parent.user_data;

    if (message->cs_take){
        rt_pin_write(cs->GPIO_Pin,PIN_LOW);
    }

    LOG_D("%s transfer prepare and start", spi_drv->config->bus_name);
    LOG_D("%s sendbuf: %02x, recvbuf: %02x, length: %d",
            spi_drv->config->bus_name,
            (message->send_buf),
            ((rt_uint8_t *)(message->recv_buf)), message->length);

    message_length = message->length;
    recv_buf = message->recv_buf;
    send_buf = message->send_buf;
    if(message_length){
        if (message->send_buf && message->recv_buf){
            state = soft_spi_read_write_bytes(spi_drv, (rt_uint8_t *)send_buf, (rt_uint8_t *)recv_buf,
message_length);
            LOG_D("soft_spi_read_write_bytes");
        }
        else if (message->send_buf){
            state = soft_spi_write_bytes(spi_drv, (rt_uint8_t *)send_buf, message_length);
            LOG_D("soft_spi_write_bytes");
        }
        else{
            memset((rt_uint8_t *)recv_buf, 0x00, message_length);
            state = soft_spi_read_bytes(spi_drv, (rt_uint8_t *)recv_buf, message_length);
            LOG_D("soft_spi_read_bytes");
        }

        if (state != message_length){
            LOG_I("spi transfer error : %d", state);
            message->length = 0;
        }
        else{
            LOG_D("%s transfer done", spi_drv->config->bus_name);
```

```
            }
        }

        if (message->cs_release){
            rt_pin_write(cs->GPIO_Pin,PIN_HIGH);
        }

        return message->length;
}

static rt_err_t spi_configure(struct rt_spi_device *device,
                                        struct rt_spi_configuration *configuration){
        RT_ASSERT(device != RT_NULL);
        RT_ASSERT(configuration != RT_NULL);

        struct ab32_soft_spi *spi_drv =     rt_container_of(device->bus, struct ab32_soft_spi, spi_bus);
        spi_drv->cfg = configuration;

        return ab32_spi_init(spi_drv, configuration);
}

static const struct rt_spi_ops ab32_spi_ops ={
        .configure = spi_configure,
        .xfer = spixfer,
};

static int rt_soft_spi_bus_init(void){
        rt_err_t result;
        for (int i = 0; i < sizeof(soft_spi_config) / sizeof(soft_spi_config[0]); i++){
            soft_spi_bus_obj[i].config = &soft_spi_config[i];
            soft_spi_bus_obj[i].spi_bus.parent.user_data = &soft_spi_config[i];
            result      =      rt_spi_bus_register(&soft_spi_bus_obj[i].spi_bus,      soft_spi_config[i].bus_name,
&ab32_spi_ops);
            RT_ASSERT(result == RT_EOK);

            LOG_D("%s bus init done", soft_spi_config[i].bus_name);
        }

        return result;
}

/**
   * Attach the spi device to SPI bus, this function must be used after initialization.
   */
rt_err_t rt_soft_spi_device_attach(const char *bus_name, const char *device_name, hal_sfr_t cs_gpiox,
```

```
rt_uint8_t cs_gpio_pin){
    RT_ASSERT(bus_name != RT_NULL);
    RT_ASSERT(device_name != RT_NULL);

    rt_err_t result;
    struct rt_spi_device *spi_device;
    struct ab32_soft_spi_pin *cs_pin;

    /* attach the device to spi bus*/
    spi_device = (struct rt_spi_device *)rt_malloc(sizeof(struct rt_spi_device));
    RT_ASSERT(spi_device != RT_NULL);
    cs_pin = (struct ab32_soft_spi_pin *)rt_malloc(sizeof(struct ab32_soft_spi_pin));
    RT_ASSERT(cs_pin != RT_NULL);
    cs_pin->GPIOx = cs_gpiox;
    cs_pin->GPIO_Pin = cs_gpio_pin;
    rt_pin_mode(cs_pin->GPIO_Pin, PIN_MODE_OUTPUT);
    result = rt_spi_bus_attach_device(spi_device, device_name, bus_name, (void *)cs_pin);

    if (result != RT_EOK){
        LOG_E("%s attach to %s faild, %d\n", device_name, bus_name, result);
    }

    RT_ASSERT(result == RT_EOK);

    LOG_D("%s attach to %s done", device_name, bus_name);

    return result;
}

int rt_soft_spi_init(void){
    return rt_soft_spi_bus_init();
}
INIT_BOARD_EXPORT(rt_soft_spi_init);

#endif
#endif /* RT_USING_SPI */
```

3.2 drv_soft_spi.h

```
/*
 * Change Logs:
 * Date           Author       Notes
 * 2021-06-03     qwz      first version
 */
```

```
#ifndef __DRV_SOFT_SPI_H_
#define __DRV_SOFT_SPI_H_

#include <rtthread.h>
#include "rtdevice.h"
#include <rthw.h>
#include <drv_common.h>

rt_err_t rt_soft_spi_device_attach(const char *bus_name, const char *device_name, hal_sfr_t cs_gpiox,
rt_uint8_t cs_gpio_pin);

struct ab32_soft_spi_pin
{
    hal_sfr_t GPIOx;
    rt_uint8_t GPIO_Pin;
};

struct ab32_soft_spi_config
{
    rt_uint8_t mosi_pin;
    rt_uint8_t miso_pin;
    rt_uint8_t sclk_pin;
    char *bus_name;
};

struct ab32_soft_spi_device
{
    rt_uint8_t cs_pin;
    char *bus_name;
    char *device_name;
};

/* ab32 soft spi dirver class */
struct ab32_soft_spi
{
    uint8_t mode;
    uint8_t cpha;
    uint8_t cpol;
    uint8_t data_width;
    uint8_t msb;
    uint16_t dummy_data;
    uint32_t spi_delay;
    uint8_t max_hz;
    struct ab32_soft_spi_config *config;
```

```
    struct rt_spi_configuration *cfg;
    struct rt_spi_bus spi_bus;
};

#endif /*__DRV_SOFT_SPI_H_ */
```

3.3 w25q_sample.c

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

#define W25Q_SPI_DEVICE_NAME          "spi10"

static struct rt_spi_device *spi_dev_w25q;


rt_uint8_t w25x_device_id[4] = {0x90,0x00,0x00,0x00};
rt_uint8_t id[5]={0};

static void w25q_erase(void)
{
    rt_uint8_t chip_erase[1] = {0x20};
    struct rt_spi_message msg;
    msg.send_buf = chip_erase;
    msg.recv_buf = RT_NULL;
    msg.length = 1;
    msg.next = RT_NULL;
    msg.cs_take = 0;
    msg.cs_release = 1;
    rt_spi_transfer_message(spi_dev_w25q, &msg);
    rt_kprintf("erase success\r\n");
}

void w25q_sector_erase(rt_uint32_t SectorAddr)
{
    rt_uint8_t page_write_cmd[4];

    rt_uint8_t write_enable = 0x06;
    struct rt_spi_message msg3;
    msg3.send_buf = &write_enable;
    msg3.recv_buf = RT_NULL;
    msg3.length = 1;
```

```
    msg3.next = RT_NULL;
    msg3.cs_take = 1;
    msg3.cs_release = 1;
    rt_spi_transfer_message(spi_dev_w25q, &msg3);

    page_write_cmd[0] = 0x20;
    page_write_cmd[1] = (SectorAddr & 0xFF0000) >> 16;
    page_write_cmd[2] = (SectorAddr & 0xFF00) >> 8;
    page_write_cmd[3] = SectorAddr & 0xFF;

    struct rt_spi_message msg,msg2;
    msg.send_buf = page_write_cmd;
    msg.recv_buf = RT_NULL;
    msg.length = 4;
    msg.next = RT_NULL;
    msg.cs_take = 1;
    msg.cs_release = 1;
    rt_spi_transfer_message(spi_dev_w25q, &msg);
}

void w25q_page_write(rt_uint8_t* pBuffer, rt_uint32_t WriteAddr, rt_uint16_t NumByteToWrite)
{
    rt_uint8_t page_write_cmd[4];

    rt_uint8_t write_enable = 0x06;
    struct rt_spi_message msg3;
    msg3.send_buf = &write_enable;
    msg3.recv_buf = RT_NULL;
    msg3.length = 1;
    msg3.next = RT_NULL;
    msg3.cs_take = 1;
    msg3.cs_release = 1;
    rt_spi_transfer_message(spi_dev_w25q, &msg3);

    page_write_cmd[0] = 0x02;
    page_write_cmd[1] = (WriteAddr & 0xFF0000) >> 16;
    page_write_cmd[2] = (WriteAddr & 0xFF00) >> 8;
    page_write_cmd[3] = WriteAddr & 0xFF;

    struct rt_spi_message msg,msg2;
    msg.send_buf = page_write_cmd;
    msg.recv_buf = RT_NULL;
    msg.length = 4;
    msg.next = RT_NULL;
    msg.cs_take = 1;
```

```
    msg.cs_release = 0;
    rt_spi_transfer_message(spi_dev_w25q, &msg);


    msg2.send_buf = pBuffer;
    msg2.recv_buf = RT_NULL;
    msg2.length = NumByteToWrite;
    msg2.next = RT_NULL;
    msg2.cs_take = 0;
    msg2.cs_release = 1;
    rt_spi_transfer_message(spi_dev_w25q, &msg2);

    rt_kprintf("write done\r\n");
}

void w25q_buffer_read(rt_uint8_t* pBuffer, rt_uint32_t ReadAddr, rt_uint16_t NumByteToRead)
{
    rt_uint8_t buffer_read_cmd[4];

    rt_uint8_t write_enable = 0x06;
    struct rt_spi_message msg3;
    msg3.send_buf = &write_enable;
    msg3.recv_buf = RT_NULL;
    msg3.length = 1;
    msg3.next = RT_NULL;
    msg3.cs_take = 1;
    msg3.cs_release = 1;
    rt_spi_transfer_message(spi_dev_w25q, &msg3);


    buffer_read_cmd[0] = 0x03;
    buffer_read_cmd[1] = (ReadAddr & 0xFF0000) >> 16;
    buffer_read_cmd[2] = (ReadAddr & 0xFF00) >> 8;
    buffer_read_cmd[3] = ReadAddr & 0xFF;

    struct rt_spi_message msg,msg2;
    msg.send_buf = buffer_read_cmd;
    msg.recv_buf = RT_NULL;
    msg.length = 4;
    msg.next = RT_NULL;
    msg.cs_take = 1;
    msg.cs_release = 0;
    rt_spi_transfer_message(spi_dev_w25q, &msg);

    msg2.send_buf = RT_NULL;
```

```
    msg2.recv_buf = pBuffer;
    msg2.length = NumByteToRead;
    msg2.next = RT_NULL;
    msg2.cs_take = 0;
    msg2.cs_release = 1;
    rt_spi_transfer_message(spi_dev_w25q, &msg2);
}

static void spi_w25q_sample(int argc,char *argv[])
{
    rt_uint8_t write_buf[100] = {0};//{0x5a,0x5a,0x5a,0x5a,0x5a};
    memset(write_buf,0xa5,100);
    rt_uint8_t read_buf[100] = {0};
    rt_soft_spi_device_attach("spi0","spi10",RT_NULL,8);


    //查找 spi 设备获取地址
    spi_dev_w25q = (struct rt_spi_device *)rt_device_find(W25Q_SPI_DEVICE_NAME);
    if(!spi_dev_w25q)
    {
        rt_kprintf("spi sample run failed\n");
    }
    else {
        struct rt_spi_configuration cfg;
        cfg.data_width = 8;
        cfg.mode = RT_SPI_MASTER | RT_SPI_MSB | RT_SPI_MODE_0;
        cfg.max_hz = 1*1000; //1.92 M
        rt_spi_configure(spi_dev_w25q, &cfg);

        struct rt_spi_message msg1, msg2;
        w25x_device_id[0] = 0x90;
        w25x_device_id[1] = 0x00;
        w25x_device_id[2] = 0x00;
        w25x_device_id[3] = 0x00;
        msg1.send_buf = w25x_device_id;
        msg1.recv_buf = RT_NULL;
        msg1.length = 4;
        msg1.cs_take = 1;
        msg1.cs_release = 0;
        msg1.next = &msg2;
        msg2.send_buf = RT_NULL;
        msg2.recv_buf = id;
        msg2.length = 5;
        msg2.cs_take = 0;
        msg2.cs_release = 1;
```

```
        msg2.next = RT_NULL;

        rt_spi_transfer_message(spi_dev_w25q, &msg1);
        rt_kprintf("use rt_spi_transfer_message() read w25q ID is:%02x %02x %02x %02x %02x\n",
id[0],id[1],id[2],id[3], id[4]);
        rt_thread_mdelay(1000);
        w25q_sector_erase(0);
        rt_thread_mdelay(6000);
        w25q_page_write(write_buf,0,100);
        rt_thread_mdelay(1000);
        rt_kprintf("write done \r\n");
        w25q_buffer_read(read_buf,0,100);
        rt_thread_mdelay(1000);
        for(rt_uint8_t i=0;i<100;i++)
        {
            rt_kprintf("%x ",read_buf[i]);
        }
    }
}

MSH_CMD_EXPORT(spi_w25q_sample, spi w25q sample);
```

4. 章节总结

本篇文章在第二章先说明从新建工程开始到写一个设备驱动的步骤：新建 rtthread studio 工程、在工程中添加驱动文件（我已经写好，见第三章代码部分），在 rtconfig.h 使能 spi、在 board.h 使能软件 spi。第三章的代码部分给出了我写的模拟 spi 的设备驱动代码和测试代码。验证主要是通过逻辑分析仪和终端打印出的调试信息。list_device 可以检查自己写的驱动有没有注册到 rt-thread，之后进行 w25q 这块 spi flash 的读写测试。

drv_soft_spi.c 这个文件只做了一件事情：注册 spi 设备。注册 spi 设备到 rt-thread 主要有两部分内容：一是实现 spi 的传输 spixfer，二是实现 spi 的配置函数。

五、中科蓝讯 AB32VG1 上的 Timer 实践

1. 前言说明

1.1 开篇语

本章介绍中科蓝讯 ab32vg1 的 timer 的用法，1.3 节介绍 rt-thread 提供的 api，接着列出了使用 sdk 的详细步骤，最后分析一次代码。本章力求简短，逐渐引人入胜，最后希望读者能举一反三，领略中科蓝讯 sdk 中 rt-thread 的魅力。

1.2 ab32vg1 的 timer 介绍

1、timer0、timer1、timer2 只支持 32 位的定时器功能。%

2、timer3、timer4、timer5 能够被配置成定时器模式、计数器模式、输入捕获模式和 pwm 模式。

1.3 rt-thread 提供的 timer 的 api

rt-thread 是通过 I/O 设备模型来管理 soc 上的外设，从上到下分为三层：I/O 设备管理层、设备驱动框架层和设备驱动层。stm32 的 HAL 库就属于设备驱动层，比如熟知的 i2c、spi 的外设驱动在用 cubemx 生成代码的时候就已经准备好。中科蓝讯的 ab32vg1 的设备驱动已经在 sdk 中由蓝讯的工程师实现。而在设备驱动层之上的设备驱动框架层和设备 I/O 管理层要说明一下：设备驱动框架层提供了一些接口留给设备驱动开发者去实现，只在做驱动移植的时候需要，作为普通用户，只需要关心 I/O 管理层即可，rt-thread 的 I/O 管理层提供了类似于 linux 中文件 IO 的 ap ,常用的有 rt_device_find、rt_device_open、rt_device_read、rt_device_close 等。下面列举了 hwtimer 的 api，结合示例去理解如何将这些 api 用起来实现定时器的功能。

```
//查找设备
/*
name：设备名称
*/
rt_device_t rt_device_find(const char* name)

//打开定时器设备
/*
dev：定时器设备句柄
```

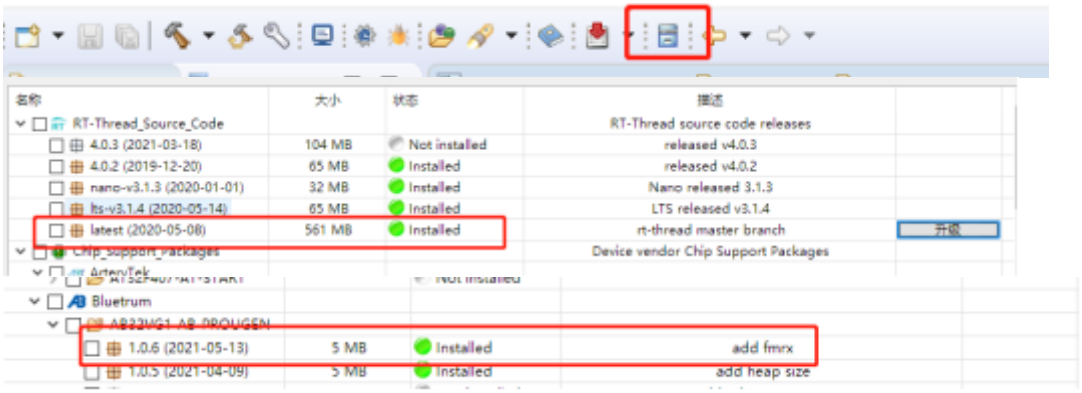
```
oflags：打开模式，一般取 RT_DEVICE_OFLAG_RDWR
*/
rt_err_t rt_device_open(rt_device_t dev, rt_uint16_t oflags);

//设置超时回调
/*
dev：定时器设备句柄
rx_ind：超时回调函数
*/
rt_err_t rt_device_set_rx_indicate(rt_device_t dev, rt_err_t (*rx_ind)(rt_device_t dev,rt_size_t size))

//控制定时器
/*
dev：定时器设备句柄
cmd：控制命令，可取
HWTIMER_CTRL_FREQ_SET 设置计数频率
HWTIMER_CTRL_STOP 停止定时器
HWTIMER_CTRL_INFO_GET 获取定时器特征信息
HWTIMER_CTRL_MODE_SET 设置定时器模式
arg：控制命令参数
设置定时器模式时，可取
HWTIMER_MODE_ONESHOT 单 次 定 时
HWTIMER_MODE_PERIOD 周 期 性 定 时
*/
rt_err_t rt_device_control(rt_device_t dev, rt_uint8_t cmd, void* arg);

//设置定时器超时值
/*
dev：定时器设备句柄
pos：偏移值，未使用，可取 0 值
buffer：指向超时时间结构体
size：超时时间结构体大小
*/
rt_size_t rt_device_write(rt_device_t dev,
rt_off_t pos,
const void* buffer,
rt_size_t size);
//获取定时器当前值
/*
dev：定时器句柄
pos：偏移值，未使用，可取 0 值
buffer：超时时间结构体
size：超时时间结构体大小
*/
rt_size_t rt_device_read(rt_device_t dev,
```

```
rt_off_t pos,
void* buffer,
rt_size_t size

);

//关闭定时器
/*
dev：定时器句柄
*/
rt_err_t rt_device_close(rt_device_t dev);
```

2. 步骤说明

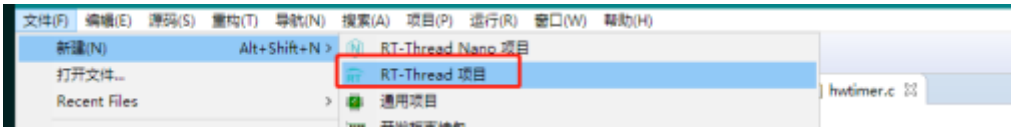
2.1 在 rt-thread studio 下载开发板 sdk

打开 sdk 管理器



2.2 rt-thread studio 新建 bsp 工程

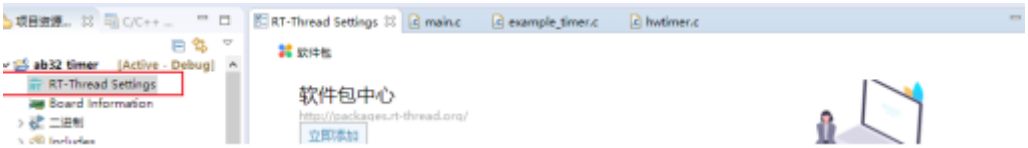
点击文件菜单下的新建工程按钮



新建 bsp 工程

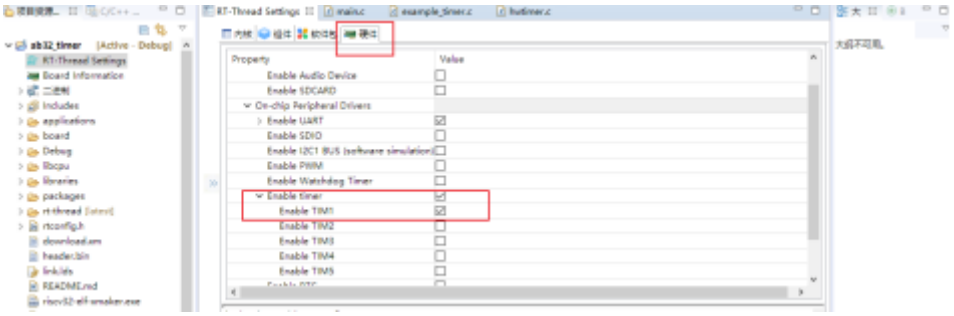
2.3 配置 rt-thread setting

双击 RT-Thread Setting



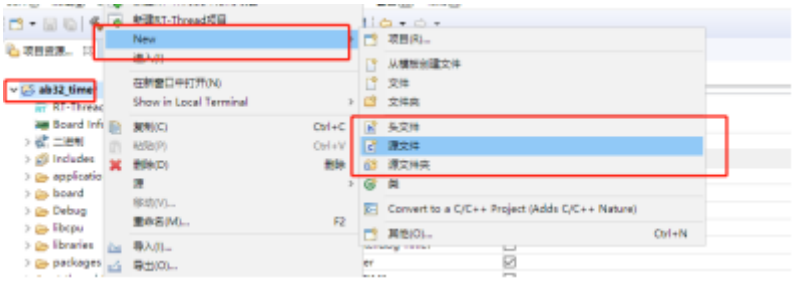
点击更多配置

在硬件选项下勾选定时器，使能定时器 1



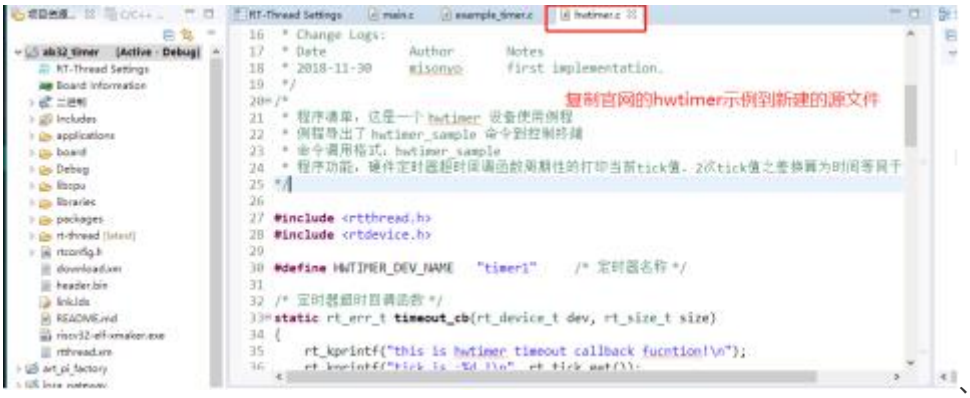
2.4 增加测试源文件

右键添加源文件

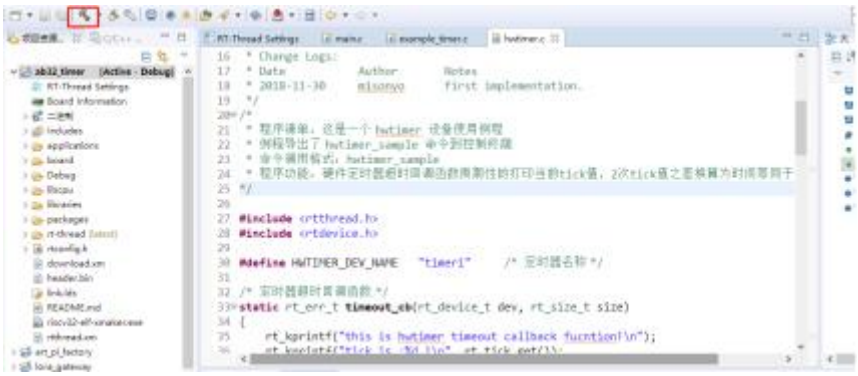


添加后点击更新软件包

复制官网的 hwtimer 示例

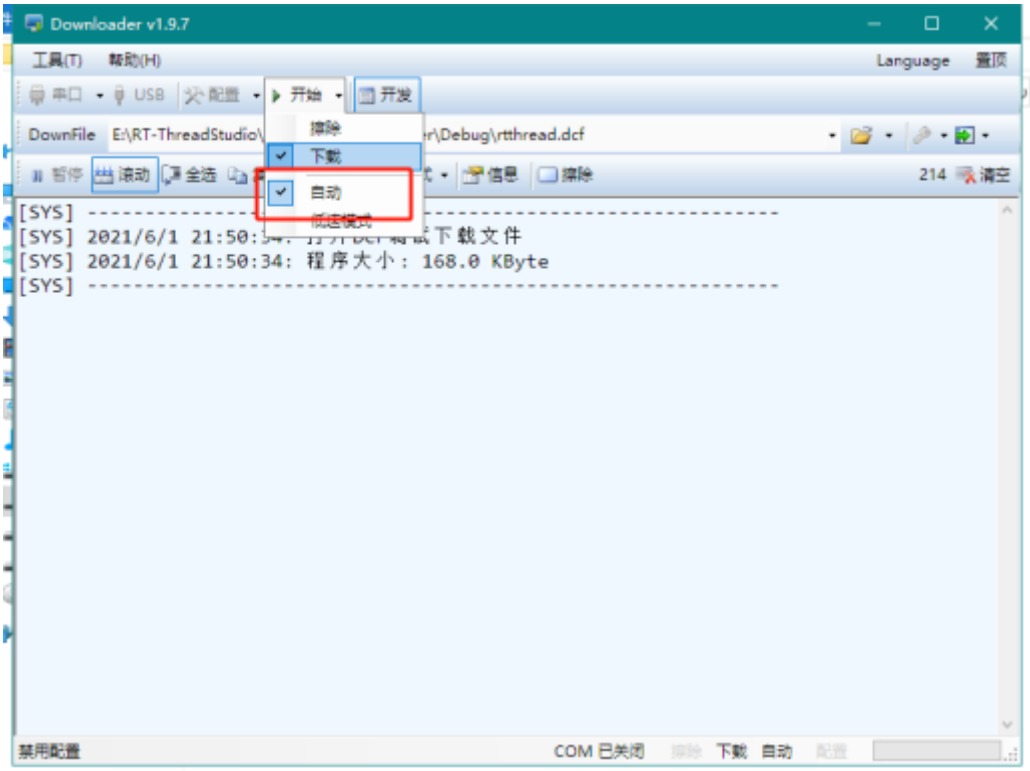


2.5 编译



2.6 下载

勾选上自动，每次编译出新的固件就会自动下载。



3. 代码验证

/*

* Copyright (c) 2006-2018, RT-Thread Development Team

*

* SPDX-License-Identifier: Apache-2.0

*

```
* Change Logs:
* Date           Author       Notes
* 2018-11-30     misonyo      first implementation.
*/
/*
* 程序清单：这是一个 hwtimer 设备使用例程
* 例程导出了 hwtimer_sample 命令到控制终端
* 命令调用格式：hwtimer_sample
* 程序功能：硬件定时器超时回调函数周期性的打印当前 tick 值，2 次 tick 值之差换算为时间等同于定时时间值。
*/

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

#define HWTIMER_DEV_NAME   "timer1"     /* 定时器名称 */

/* 定时器超时回调函数 */
static rt_err_t timeout_cb(rt_device_t dev, rt_size_t size)
{
rt_kprintf("this is hwtimer timeout callback fucntion!\n");
rt_kprintf("tick is :%d !\n", rt_tick_get());

return 0;
}

static int hwtimer_sample(int argc, char *argv[])
{
rt_err_t ret = RT_EOK;
rt_hwtimerval_t timeout_s;      /* 定时器超时值 */
rt_device_t hw_dev = RT_NULL;   /* 定时器设备句柄 */
rt_hwtimer_mode_t mode;         /* 定时器模式 */
rt_uint32_t freq = 10000;       /* 计数频率 */

/* 查找定时器设备 */
hw_dev = rt_device_find(HWTIMER_DEV_NAME);
if (hw_dev == RT_NULL)
{
rt_kprintf("hwtimer sample run failed! can't find %s device!\n", HWTIMER_DEV_NAME);
return RT_ERROR;
}

/* 以读写方式打开设备 */
ret = rt_device_open(hw_dev, RT_DEVICE_OFLAG_RDWR);
if (ret != RT_EOK)
```

```
{
rt_kprintf("open %s device failed!\n", HWTIMER_DEV_NAME);
return ret;
}

/* 设置超时回调函数 */
rt_device_set_rx_indicate(hw_dev, timeout_cb);

/* 设置计数频率(默认 1Mhz 或支持的最小计数频率) */
ret = rt_device_control(hw_dev, HWTIMER_CTRL_FREQ_SET, &freq);
if (ret != RT_EOK)
{
rt_kprintf("set frequency failed! ret is :%d\n", ret);
return ret;
}

/* 设置模式为周期性定时器 */
mode = HWTIMER_MODE_PERIOD;
ret = rt_device_control(hw_dev, HWTIMER_CTRL_MODE_SET, &mode);
if (ret != RT_EOK)
{
rt_kprintf("set mode failed! ret is :%d\n", ret);
return ret;
}

/* 设置定时器超时值为 5s 并启动定时器 */
timeout_s.sec = 5;      /* 秒 */
timeout_s.usec = 0;     /* 微秒 */

if (rt_device_write(hw_dev, 0, &timeout_s, sizeof(timeout_s)) != sizeof(timeout_s))
{
rt_kprintf("set timeout value failed\n");
return RT_ERROR;
}

/* 延时 3500ms */
rt_thread_mdelay(3500);

/* 读取定时器当前值 */
rt_device_read(hw_dev, 0, &timeout_s, sizeof(timeout_s));
rt_kprintf("Read: Sec = %d, Usec = %d\n", timeout_s.sec, timeout_s.usec);

return ret;
}
/* 导出到 msh 命令列表中 */
```

MSH_CMD_EXPORT(hwtimer_sample, hwtimer sample);

4. 章节总结

使用 rt-thread studio 进行 sdk 的开发是一件非常有效率的事情，新建 bsp 工程后只需要在 rt-thread setting 配置需要的硬件功能就可以使用 rt-thread 提供的设备 I/O 管理接口对底层的 soc 的外设进行控制。从示例中可以定时器的流程：先用 rt_device_find 根据设备名称查找到定时器句柄、使用定时器句柄打开定时器、接着设置定时器的回调函数、配置完定时器后设置定时器的定时值后定时器启动，之后每当定时器的计数器溢出就会执行一次定时器的回调函数。

六、中科蓝讯 AB32VG1 上的 ADC 实践

1. 前言说明

1.1 本章内容

本章介绍基于 rt-thread studio 的 sdk 开发 adc 的应用。

1.2 模块介绍

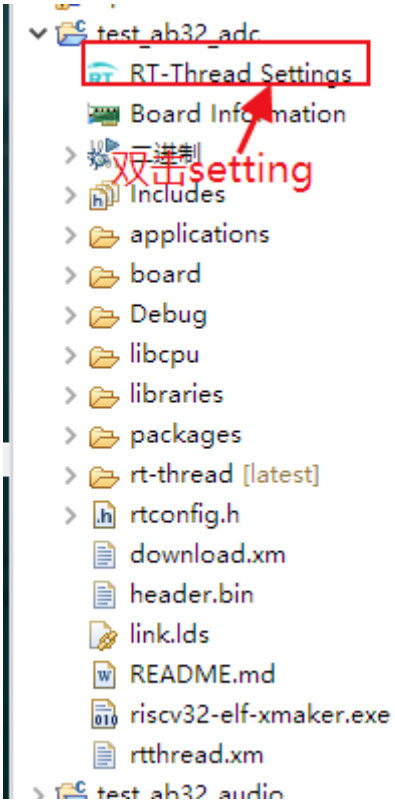
AB32VG1 有 16 个通道的 10 bit 的 ADC 模块。

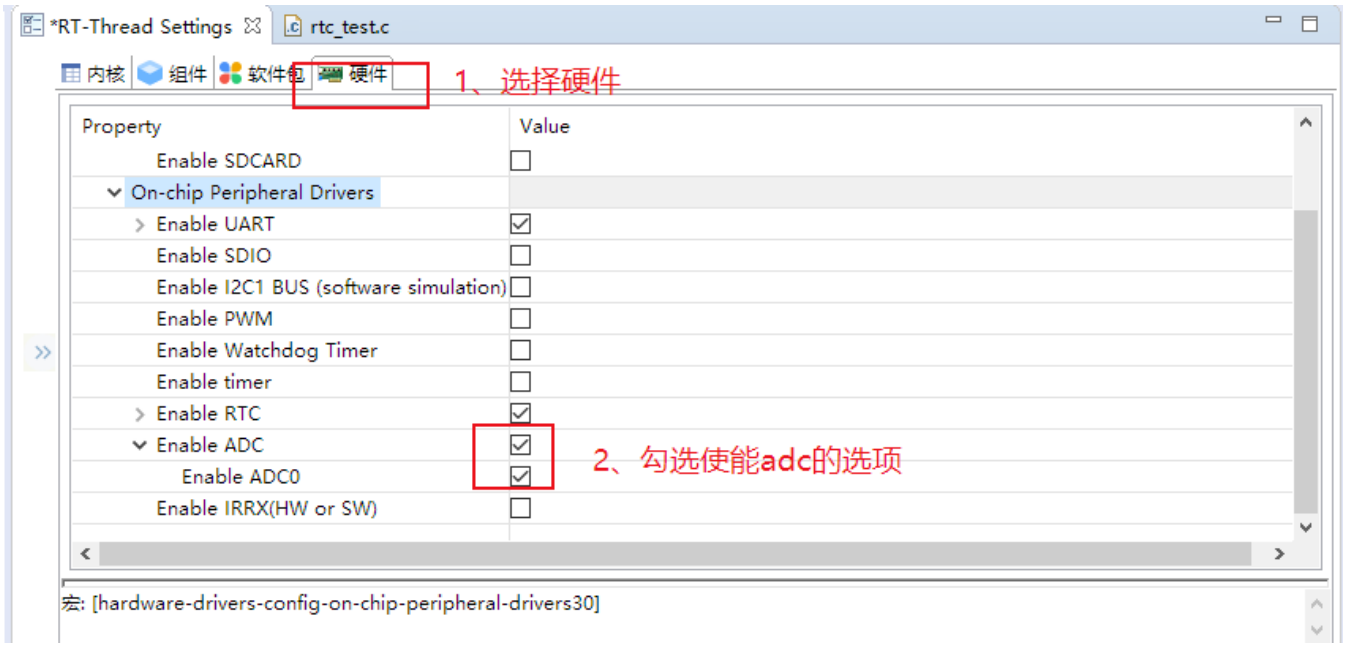
- 最大采样速度是 78k/s; ADC 模块时钟的最大速度是 1MHz

- 有内部 100k 的上拉电阻

2. 步骤说明

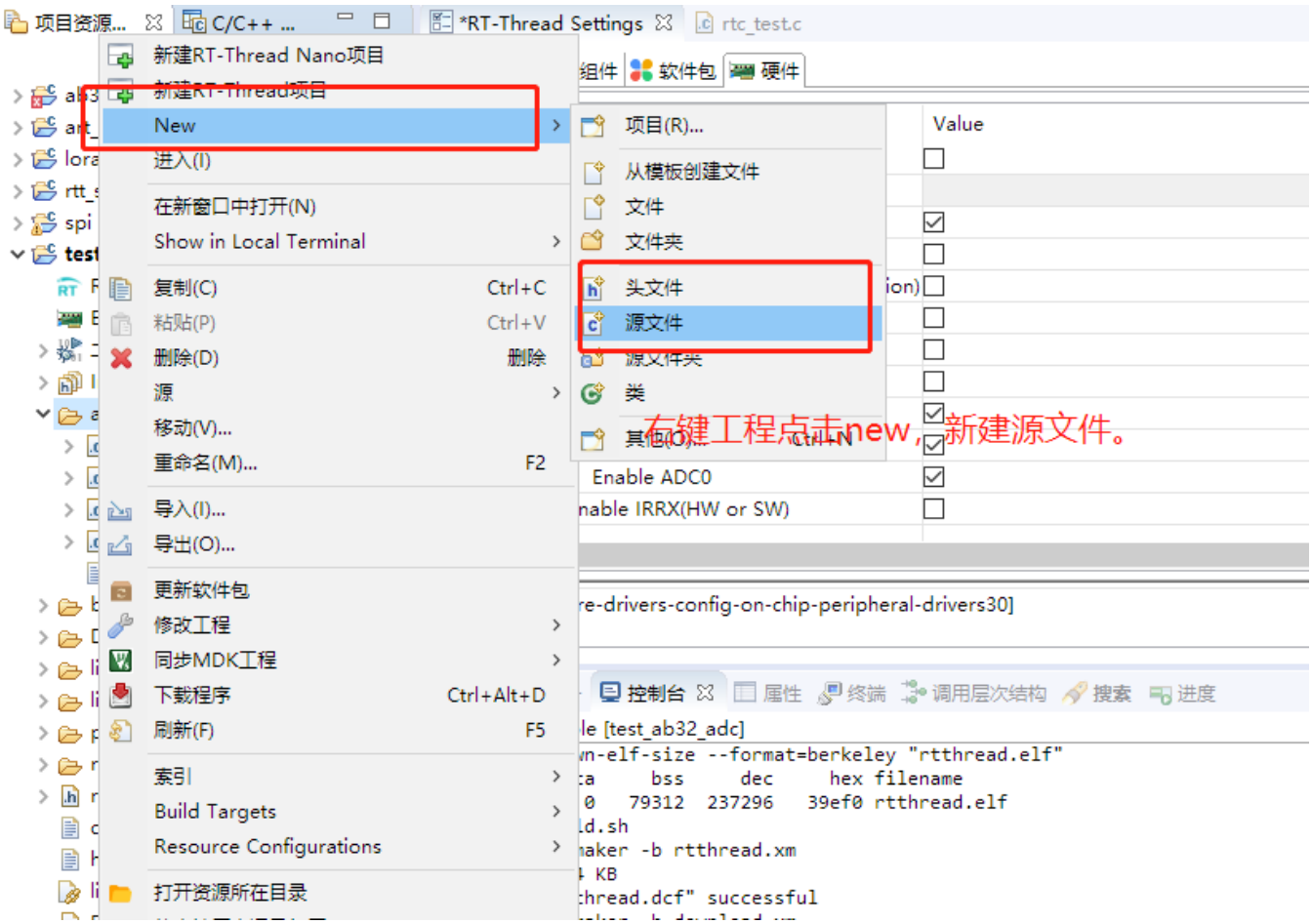
2.1 新建工程和配置

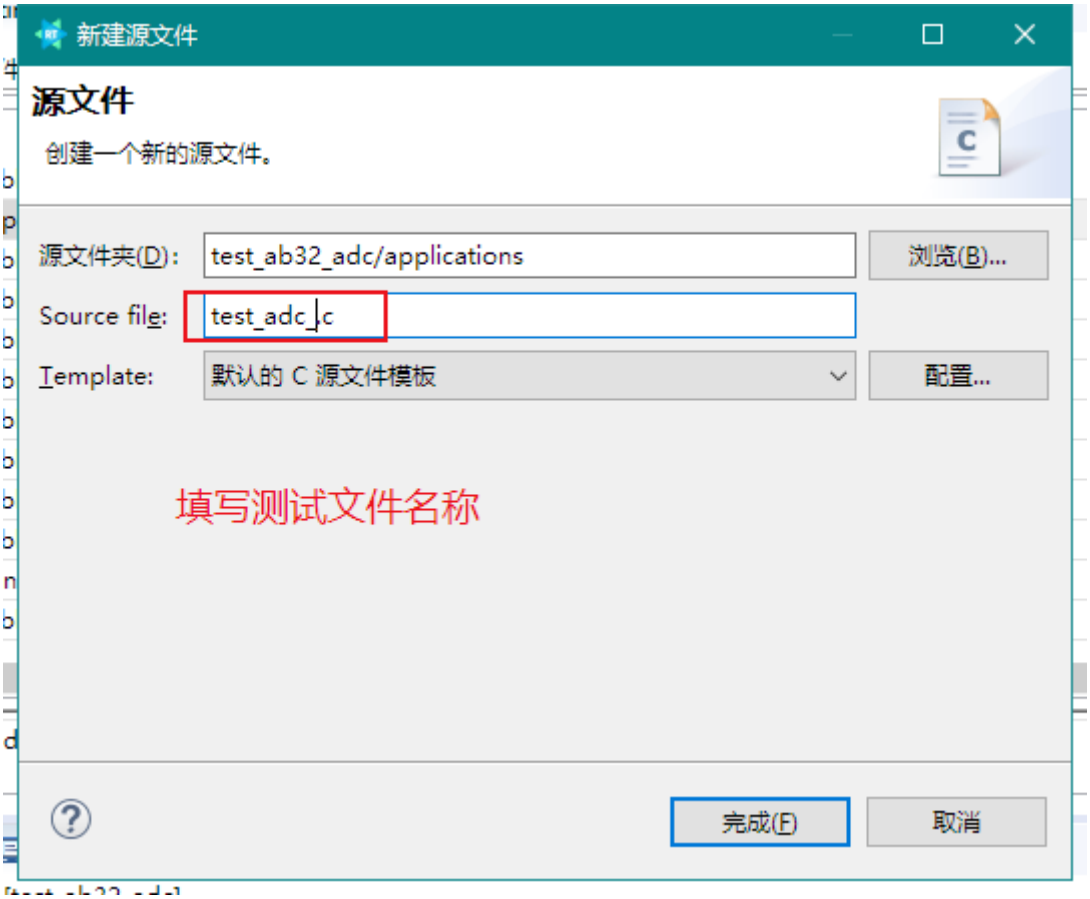




配置好后 ctrl+s 保存配置。

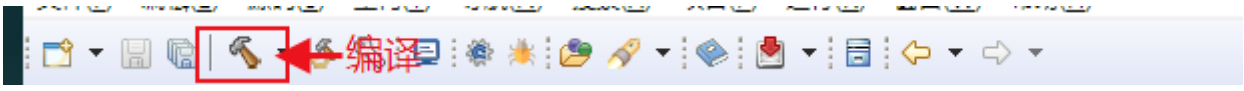
2.2 添加测试源文件

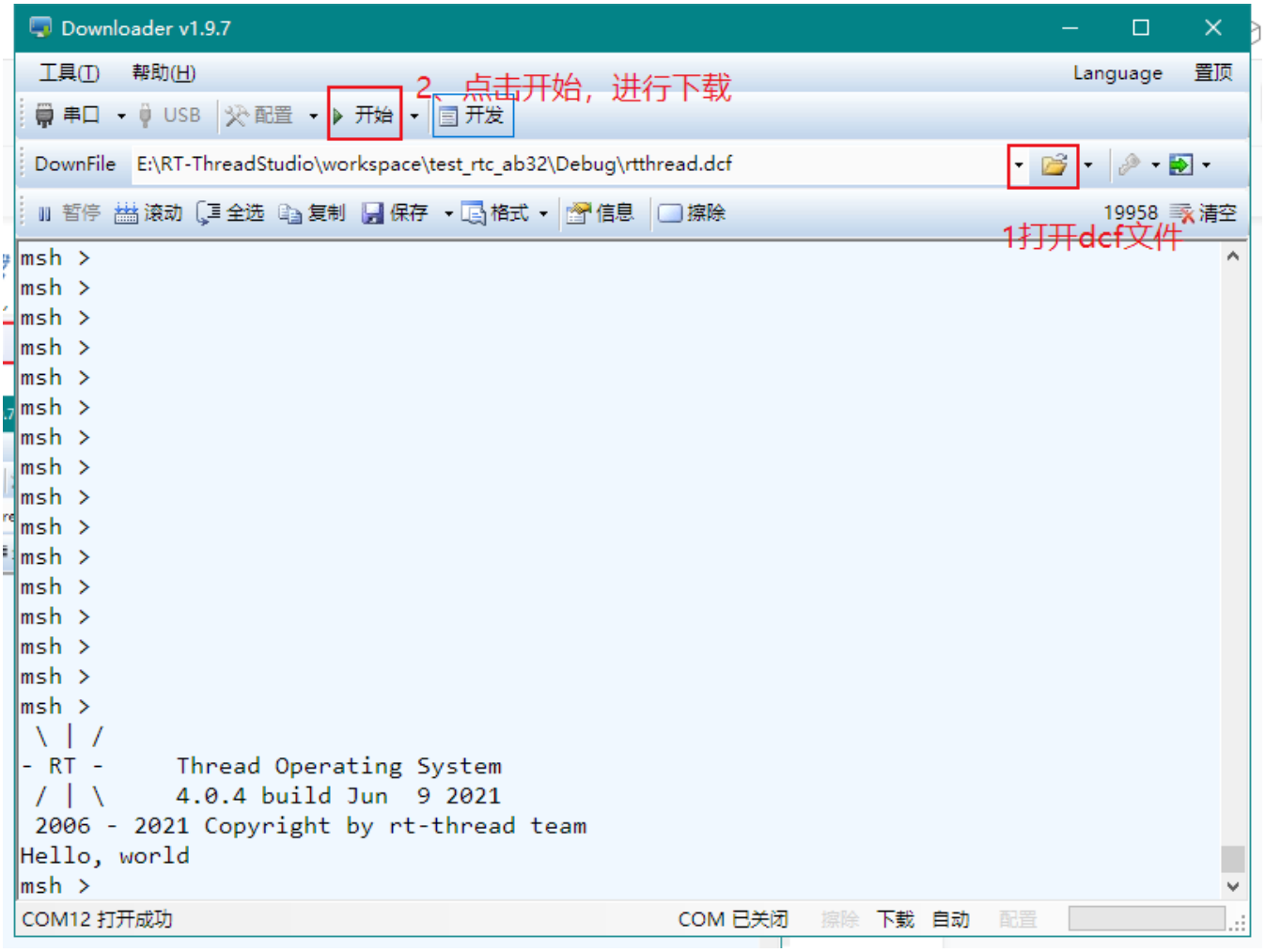

项目资源...
C/C++ ...
*RT-Thread Settings
rtc_test.c
新建RT-Thread Nano项目
新建RT-Thread项目
New
进入(I)
在新窗口中打开(N)
Show in Local Terminal
复制(C) Ctrl+C
粘贴(P) Ctrl+V
删除(D) 删除
源
移动(V)...
重命名(M)... F2
导入(I)...
导出(O)...
更新软件包
修改工程
同步MDK工程
下载程序 Ctrl+Alt+D
刷新(F) F5
索引
Build Targets
Resource Configurations
打开资源所在目录
项目(R)...
从模板创建文件
文件
文件夹
头文件
源文件
源文件夹
类
其他(O)... Ctrl+N
组件
软件包
硬件
Value
Enable ADC0
nable IRRX(HW or SW)
右键工程点击new，新建源文件。
控制台
属性
终端
调用层次结构
搜索
进度
le [test_ab32_adc]
n-elf-size --format=berkeley "rtthread.elf"
bss dec hex filename
79312 237296 39ef0 rtthread.elf
maker -b rtthread.xm
thread.dcf" successful


新建源文件
源文件
创建一个新的源文件。
源文件夹(D):
test_ab32_adc/applications
浏览(B)...
Source file:
test_adc.c
Template:
默认的 C 源文件模板
配置...
填写测试文件名称
完成(F)
取消

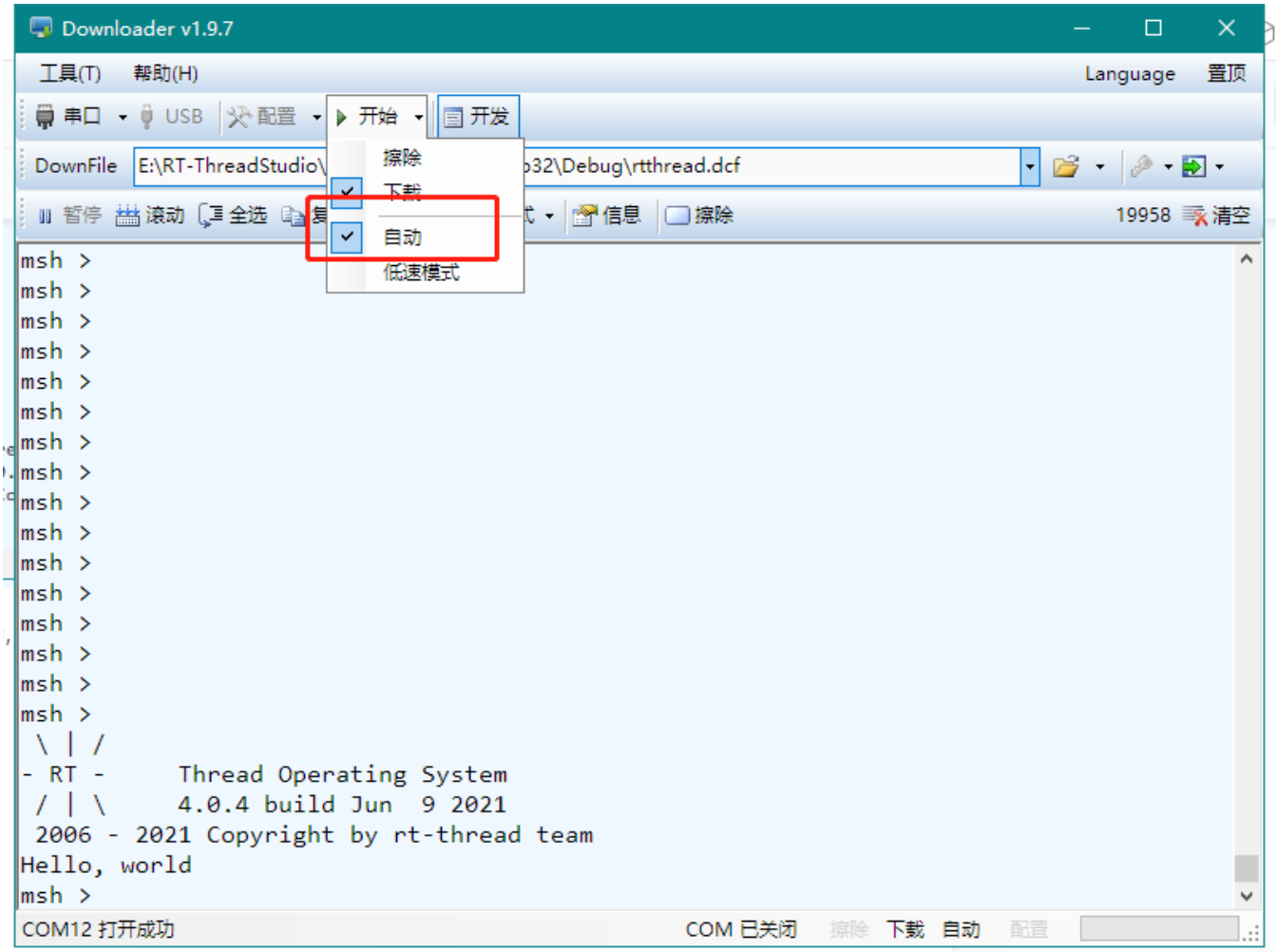

*RT-Thread Settings
rtc_test.c
test_adc.c
/*
* Copyright (c) 2006-2021, RT-Thread Development Team
*
* SPDX-License-Identifier: Apache-2.0
*
* Change Logs:
* Date Author Notes
* 2021-06-08 yuspu the first version
*/
#include <rtthread.h>
#include <rtdevice.h>
复制官网的adc设备的测试代码到测试文件。
#define ADC_DEV_NAME "adc0" /* ADC 设备名称 */
#define ADC_DEV_CHANNEL 7 /* ADC 通道 */
#define REFER_VOLTAGE 330 /* 参考电压 3.3V,数据精度乘以100保留2位小数*/
#define CONVERT_BITS (1 << 10) /* 转换位数为10位 */
static int adc_vol_sample(int argc, char *argv[])

2.3 编译和下载


编译



如果选择了自动，则编译程序后会自动下载。自动选项如下：



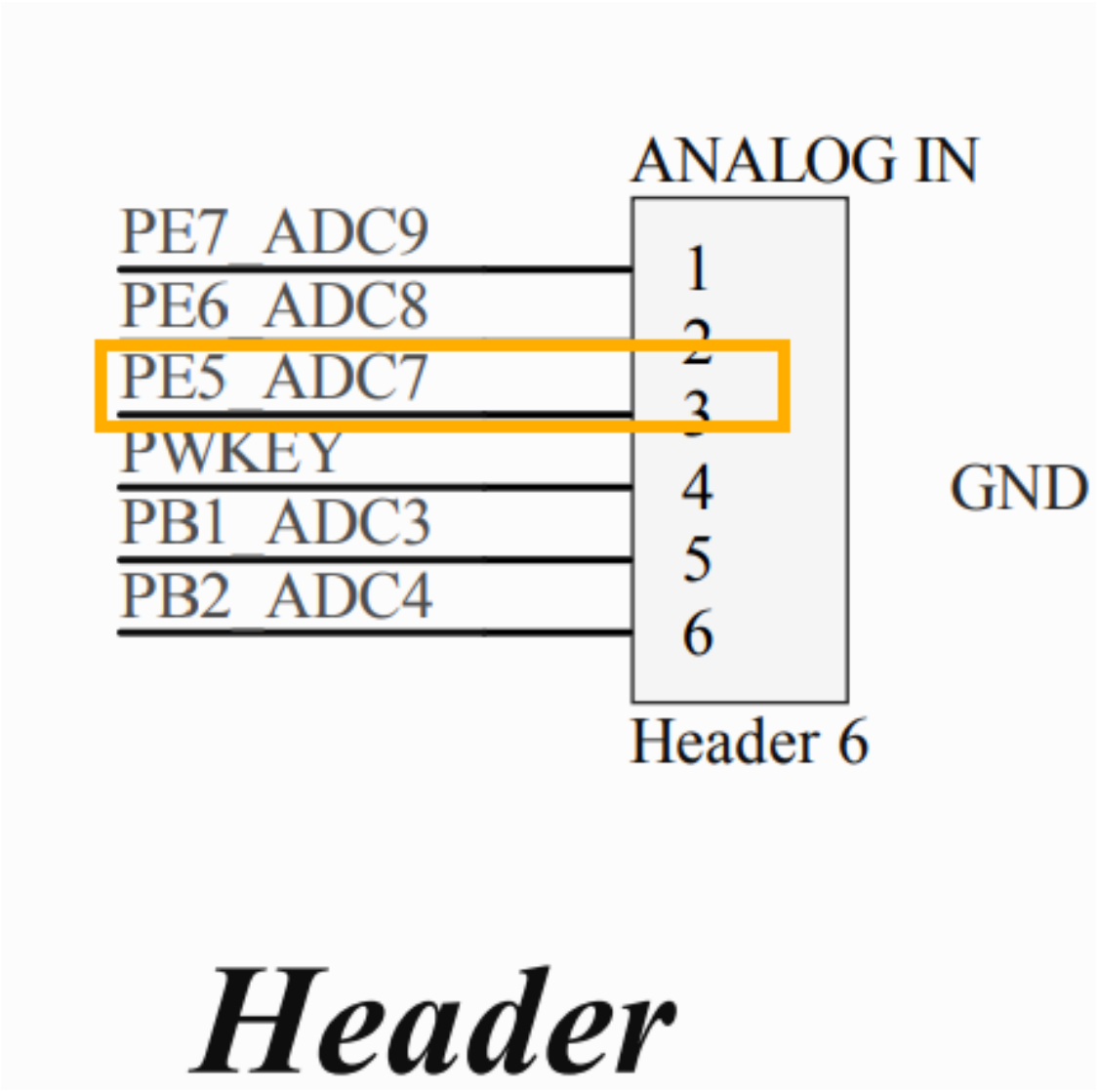
2.4 现象

开发板的 A3 连接到 3.3v，准备测 3.3.v 引脚的电压。在 finsh 命令行输入 tab 键，弹出支持的命令，输入 adc_vol_sample。

```
list_thread      - list thread
version          - show RT-Thread version information
clear            - clear the terminal screen
free             - Show the memory usage in the system.
ps               - List threads in the system.
help             - RT-Thread shell help.
exit             - return to RT-Thread shell mode.
date             - get date and time or set (local timezone) [year month day hour
min sec]
list_date        - show date and time (local timezone)
adc              - adc function
adc_vol_sample   - adc voltage convert sample
rtc_sample       - RTC SAMPLE

msh >adc_vol_sample
the value is :0
the voltage is :0.00
msh >adc_vol_sample
the value is :1023
the voltage is :3.29
msh >
```

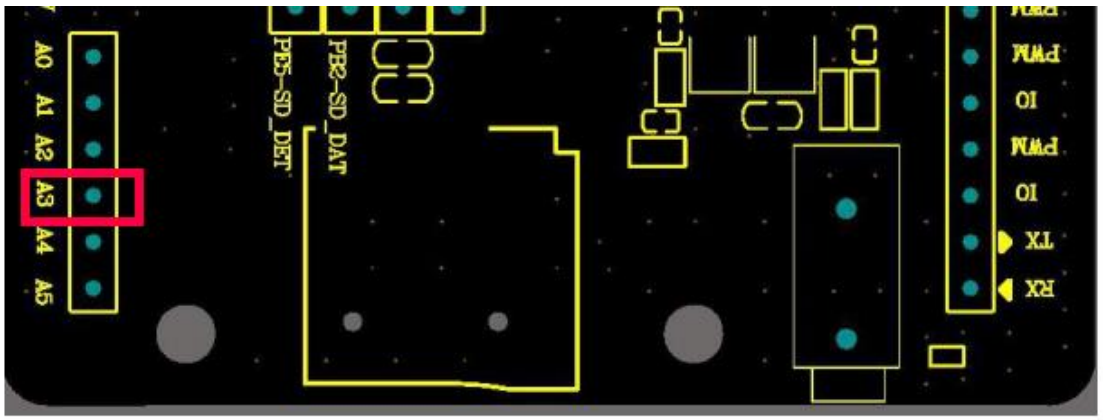
在finsh输入adc_vol_sample

说明：使用的是 ADC 的第 7 个通道,根据原理图可知对应的是 PE5 管脚:



特别要注意的是在板子上对应的丝印编号是 ANALOG IN 的 A3.千万不要接错了。

（A3 的 A 是模拟量的意思，那一排引脚都作为模拟输入）



3.代码

因为 ADC 驱动的核心代码并未完全公开,大部分内容都封装到了 libhal.a 库文件中。

```
ab32vg1_hal_adc.o:
00000000 T hal_adc_enable
00000000 T hal_adc_poll_for_conversion
00000000 T hal_adc_start
         U hal_mdelay
00000010 t .L2
0000000a t .L4
0000001c t .L6
0000000e t .L7
00000024 t .L9
         U __riscv_restore_3
         U __riscv_save_3
```

驱动层面的代码也不好过多分析.核心的使能某一个通道以及获取某个通道的数据源码都未可见.所以简单从 APP 层写一段测试代码.同样的,对 ADC 的测试代码还是参考 RT-Thread 的官网.并进行简单的适配修改.

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

#define ADC_DEV_NAME        "adc0"      /* ADC 设备名称 */
#define ADC_DEV_CHANNEL     7           /* ADC 通道 */
#define REFER_VOLTAGE       330         /* 参考电压 3.3V,数据精度乘以 100 保留 2 位小数*/
#define CONVERT_BITS        (1 << 10)   /* 转换位数为 10 位 */

static int adc_vol_sample(int argc, char *argv[])
{
    rt_adc_device_t adc_dev;
    rt_uint32_t value, vol;
    rt_err_t ret = RT_EOK;

    /* 查找设备 */
    adc_dev = (rt_adc_device_t)rt_device_find(ADC_DEV_NAME);
    if (adc_dev == RT_NULL)
    {
        rt_kprintf("adc sample run failed! can't find %s device!\n", ADC_DEV_NAME);
        return RT_ERROR;
    }
```

```
    /* 使能设备 */
    ret = rt_adc_enable(adc_dev, ADC_DEV_CHANNEL);

    /* 读取采样值 */
    value = rt_adc_read(adc_dev, ADC_DEV_CHANNEL);
    rt_kprintf("the value is :%d \n", value);

    /* 转换为对应电压值 */
    vol = value * REFER_VOLTAGE / CONVERT_BITS;
    rt_kprintf("the voltage is :%d.%02d \n", vol / 100, vol % 100);

    /* 关闭通道 */
    ret = rt_adc_disable(adc_dev, ADC_DEV_CHANNEL);

    return ret;
}
/* 导出到 msh 命令列表中 */
MSH_CMD_EXPORT(adc_vol_sample, adc voltage convert sample);
```

4. 章节总结

最后做一个总结，首先新建一个 rt-thread studio 的工程，接着配置 rt-thread setting，使能 sdk 的 adc，配置完后 crtl+s 保存，接着在 application 文件夹下新建测试源文件，在源文件中添加官方的 adc 设备测试代码，后面编译好后下载到开发板就可以开始测量电压了。

七、中科蓝讯 AB32VG1 上的 PWM 实践

1.前言说明

本次实验主要对 AB32VG1 开发板的 PWM 部分进行简单讲解，通过本次实验，我们将

会看到 AB32VG1 开发板上的 RGB 灯会展示出呼吸灯的效果。

1.1 实验必备

软件：

1. RT-Thread Studio v 2.1.0
2. AB32VG1 软件包 v1.0.6
3. Downloader v1.9.7

硬件：

1. AB32VG1 开发板 V1.0
2. 示波器一台
3. USB-TYPEC 数据线一条

1.2 模块介绍

AB32VG1 开发板提供六路 PWM 输出 ,分别对应的引脚为 PA2,PE4,PA6,PE0,PE1,PB0，其中，PE1,PE4,PA2 用跳线帽连接可使用全彩 LED 模块。在这六路 PWM 中，有三路为基本 PWM，由定时器产生，则另外三路为 LPWM，由专门的 pwm 外设产生，其中三路 LPWM 是互斥的。

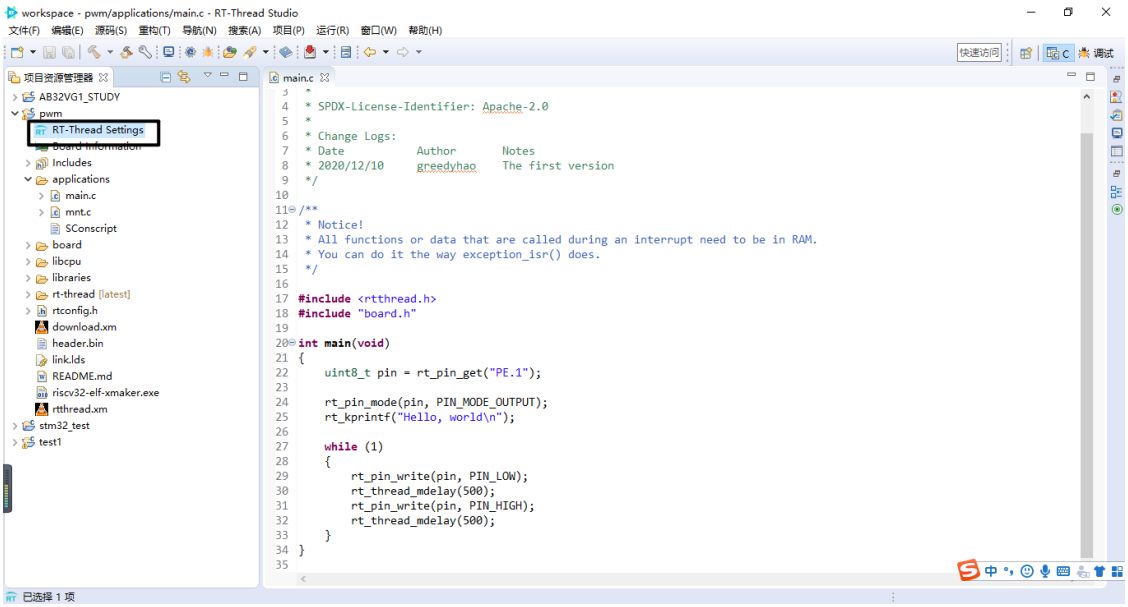
2.步骤说明

2.1 新建工程

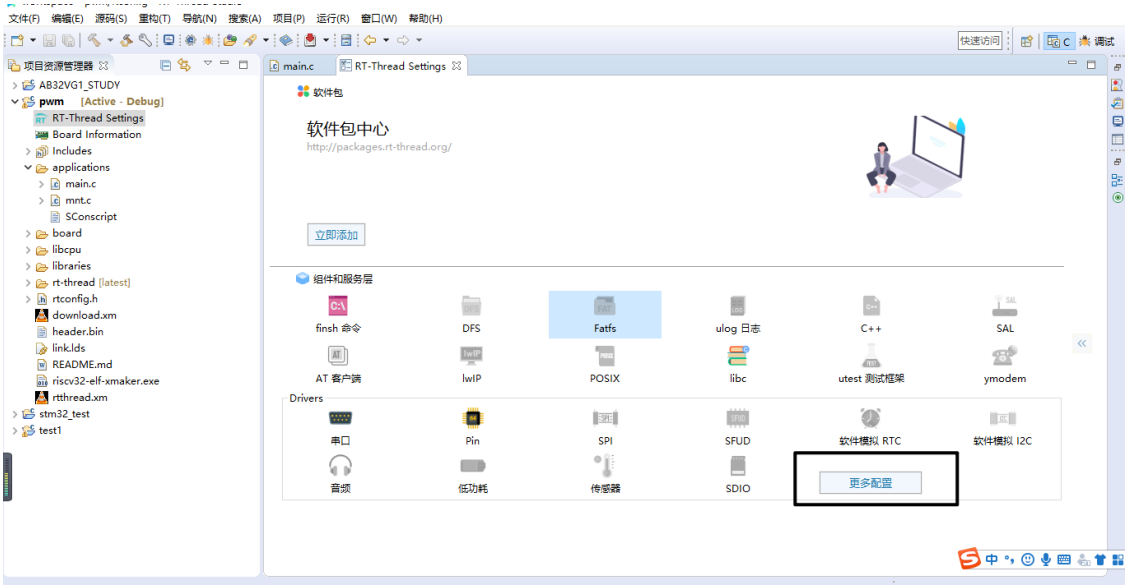
新建一个名为 pwm 的项目，在此不在赘述。请参考从内部 Flash 读取 WAV 音频播放 。

2.2 配置 PWM

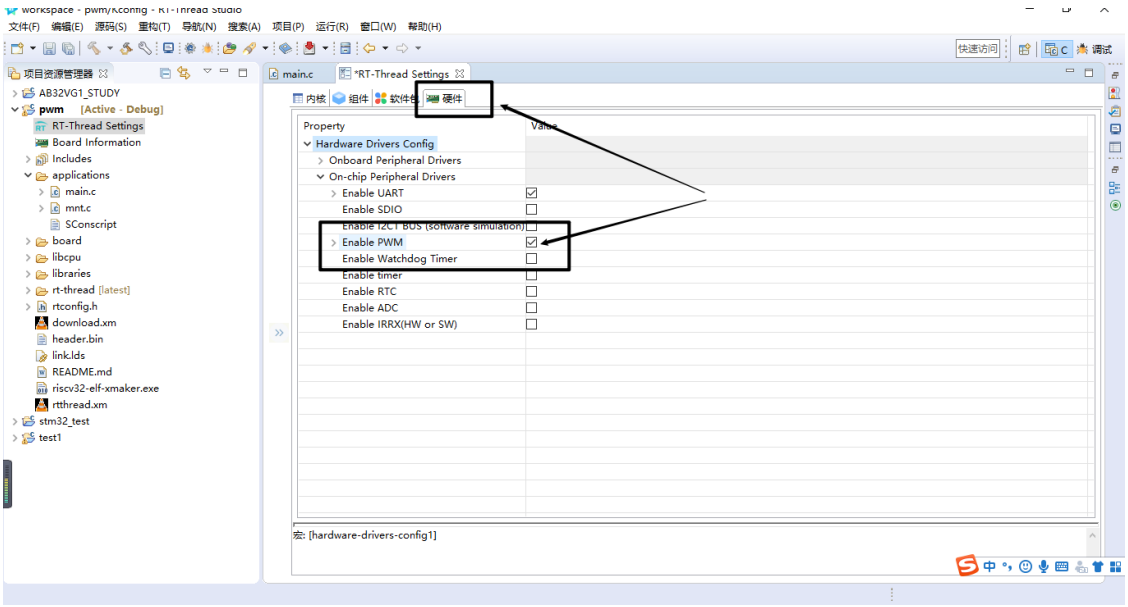
1.双击本项目的 RT-Thread Setting，进入配置软件包页面。



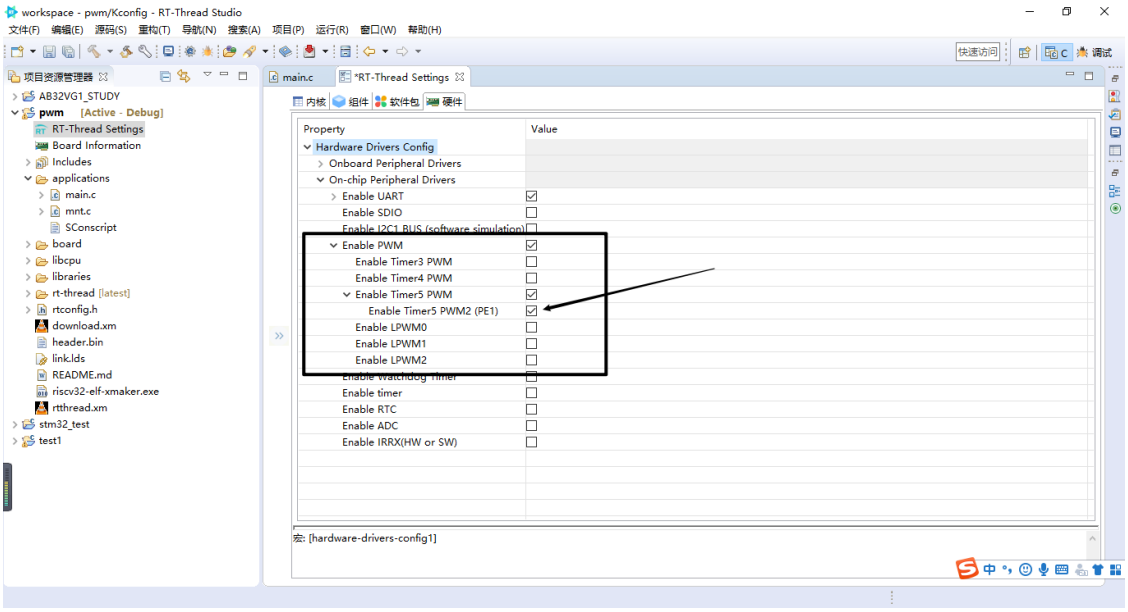
2.进入软件包配置界面，点击更多配置。



3.在进入更多配置后，选择硬件选项，使能 PWM。



4.配置用户需要的 PWM 引脚。本次使能 Timer5 PWM2 通道。

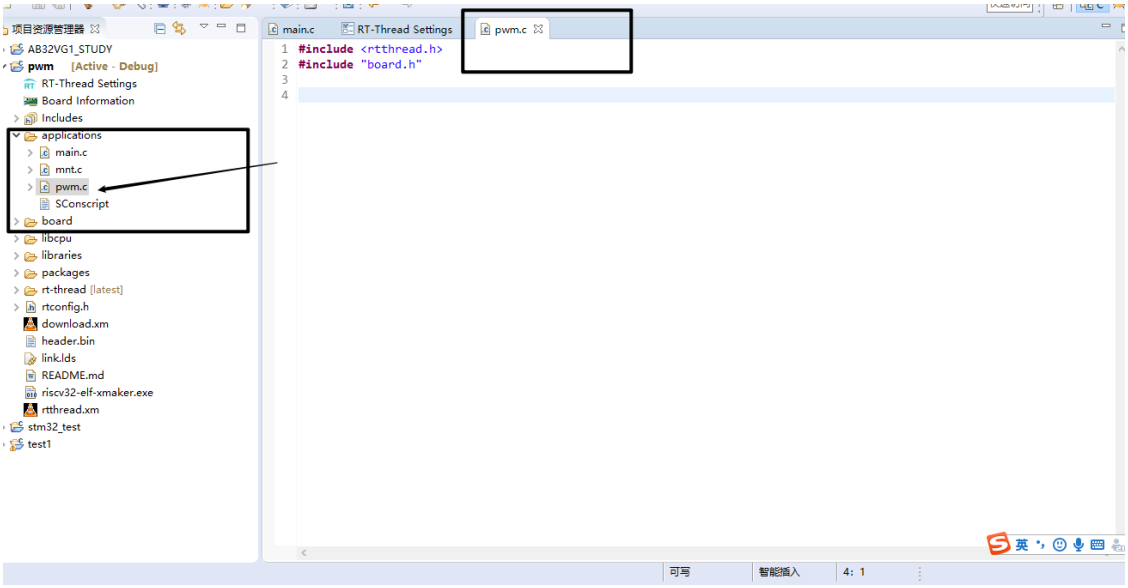


注：其中三路 LPWM 是互斥的，开发时需注意。

5.保存。

2.3 程序编写

1.在项目中的 application 文件夹中，新建 pwm.c

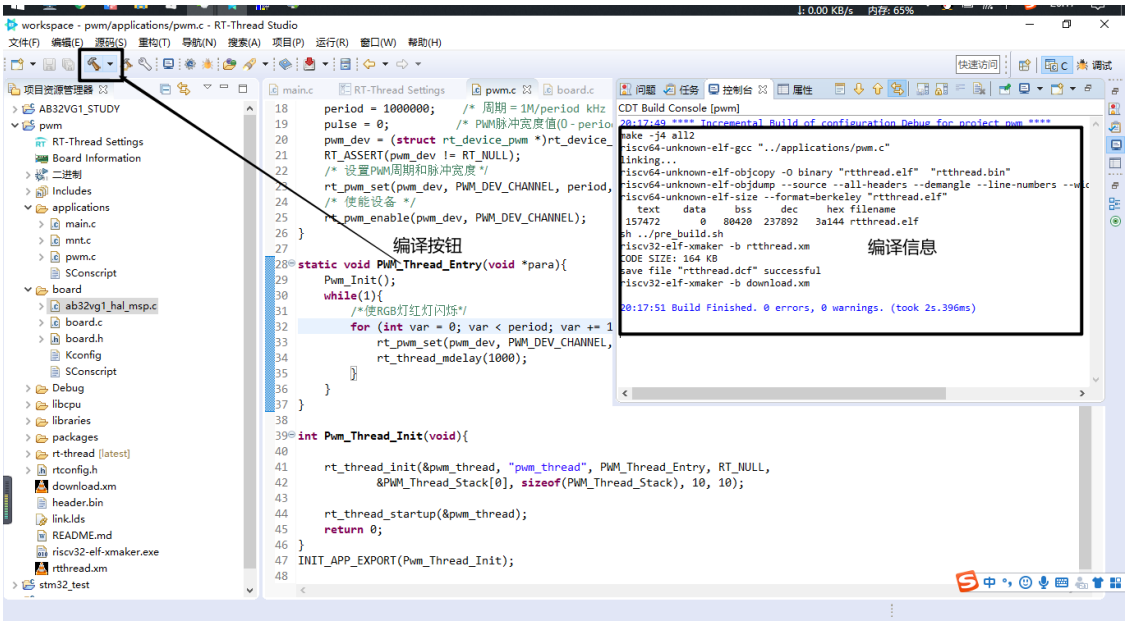


2.程序编写

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

/*************************************************************************************/
/*******************************    PWM_DEV_NAME ********************************/
/*** timer3 pwm    *** timer4 pwm *** timer5 pwm ***    lpwm0 *** lpwm1 *** lpwm2 ***/
/*** t3pwm            *** t4pwm          *** t5pwm          ***    lpwm0 *** lpwm1 *** lpwm2 ***/
/*************************************************************************************/
#define PWM_DEV_NAME             "t5pwm"    /* PWM 设备名称  */
#define PWM_DEV_CHANNEL        1            /* PWM 通道  */
struct rt_device_pwm *pwm_dev;          /* PWM 设备句柄  */
ALIGN(RT_ALIGN_SIZE)
static uint8_t PWM_Thread_Stack[1024];
static void PWM_Thread_Entry(void *para);
static struct rt_thread pwm_thread;
rt_uint32_t period, pulse;
void Pwm_Init(void){
    period = 1000000;       /*  周期  = 1M/period kHz */
    pulse = 0;                  /* PWM 脉冲宽度值(0 - period) */
    pwm_dev = (struct rt_device_pwm *)rt_device_find(PWM_DEV_NAME);
    RT_ASSERT(pwm_dev != RT_NULL);
    /*  设置 PWM 周期和脉冲宽度  */
    rt_pwm_set(pwm_dev, PWM_DEV_CHANNEL, period, pulse);
    /*  使能设备  */
    rt_pwm_enable(pwm_dev, PWM_DEV_CHANNEL);
}

static void PWM_Thread_Entry(void *para){
```

```
    Pwm_Init();
    while(1){
        /*使 RGB 灯红灯闪烁*/
        for (int var = 0; var < period; var += 10000) {
            rt_pwm_set(pwm_dev, PWM_DEV_CHANNEL, period, var);
            rt_thread_mdelay(1000);
        }
    }
}

int Pwm_Thread_Init(void){

    rt_thread_init(&pwm_thread, "pwm_thread", PWM_Thread_Entry, RT_NULL,
            &PWM_Thread_Stack[0], sizeof(PWM_Thread_Stack), 10, 10);

    rt_thread_startup(&pwm_thread);
    return 0;
}
INIT_APP_EXPORT(Pwm_Thread_Init);
```

3.代码编译

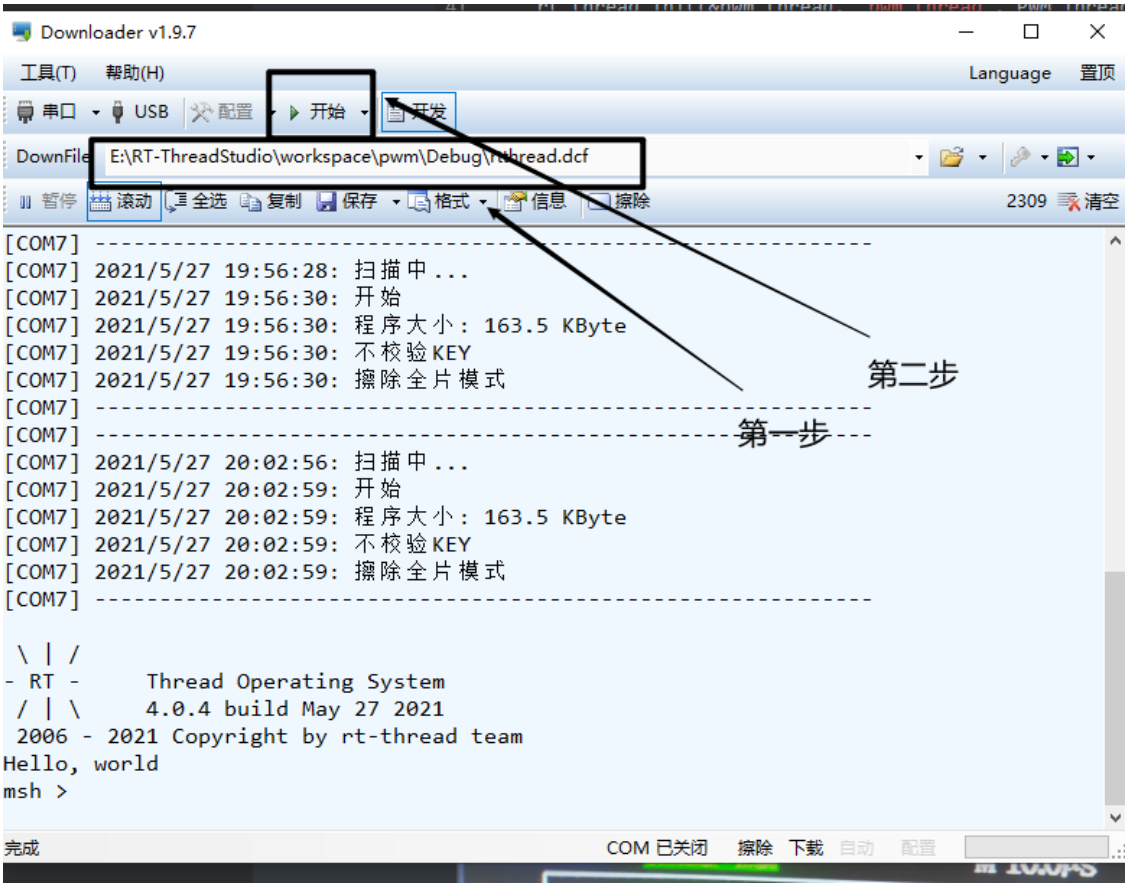


3.代码验证

3.1 下载

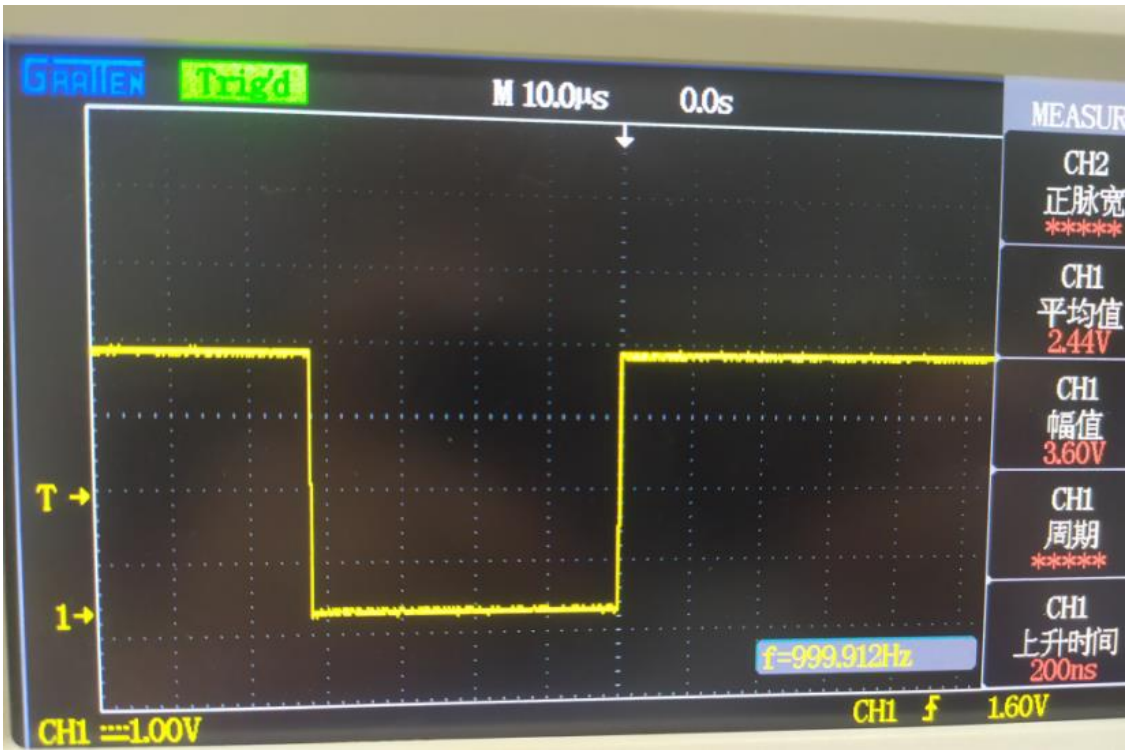
下载使用下载工具 Downloader。第一步：选择你的编译文件。第二步：点击下载即可。

在完成下载后，手动重启，Downloader 中会打印出 Hello Word，说明程序下载成功。

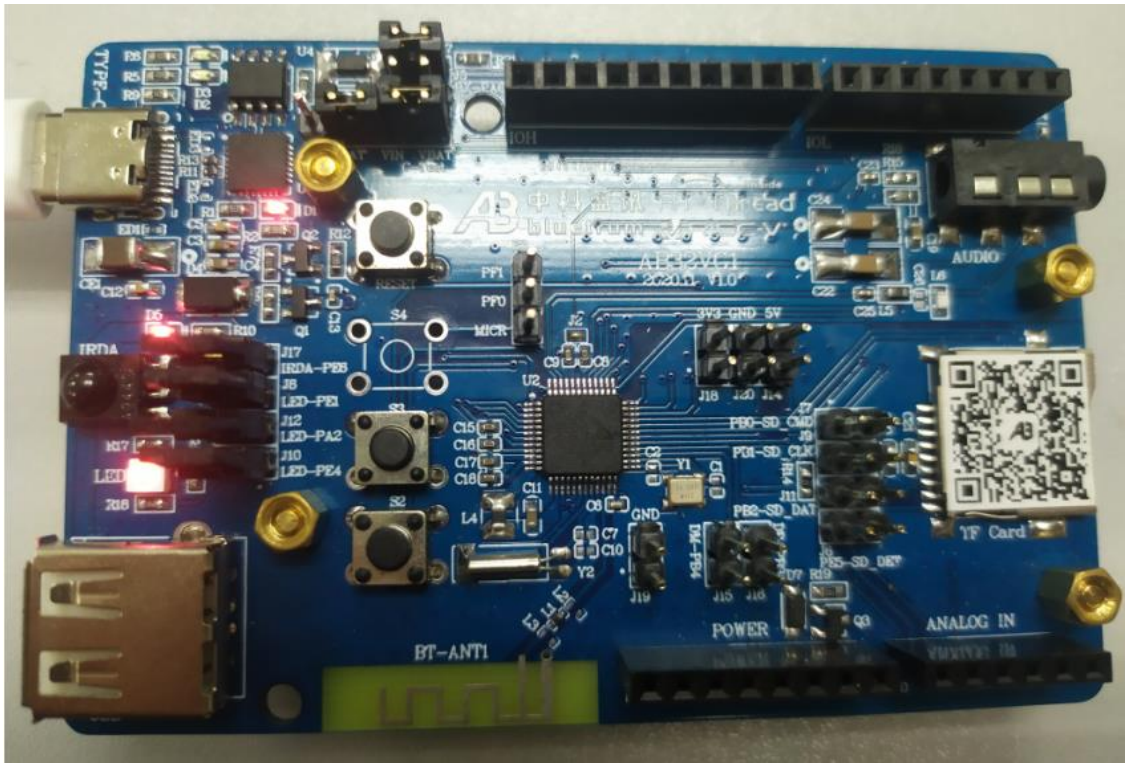


3.2 验证

验证一：通过示波器验证，如下图所示。



验证二：观察 RGB 灯转台验证，如下图所示。RGB 中的红灯会出现呼吸灯的效果。



4.章节总结

本次实现是对板载 PWM 进行测试，由于官方已经将 PWM 的配置完全图形化了，非常适合新手配置，因此对于 PWM 的配置难度不大。在本次实验中，最重要的是代码问题，在配置 PWM 的名字和通道时需要注意，这一部分会和官方的 PWM 教程有些出入，主要是注意 PWM 对象的名字和通道。

八、中科蓝讯 AB32VG1 上的 WDT 实践

1.前言说明

本次实验是对 AB32VG1 开发板上的看门狗进行试验。通过本次实验，我们可以学会如何使用看门狗。

1.1 实验必备

软件：

4. RT-Thread Studio v 2.1.0
5. AB32VG1 软件包 v1.0.6
6. Downloader v1.9.7

硬件：

1. AB32VG1 开发板 V1.0

2. USB-TYPEC 数据线一条

1.2 模块介绍

AB32VG1 开发版的主控芯片提供了片内看门狗，使用户不需外接看门狗外设，极大地较少了电路的复杂度。用户仅需操作相应寄存器便可控制看门狗。

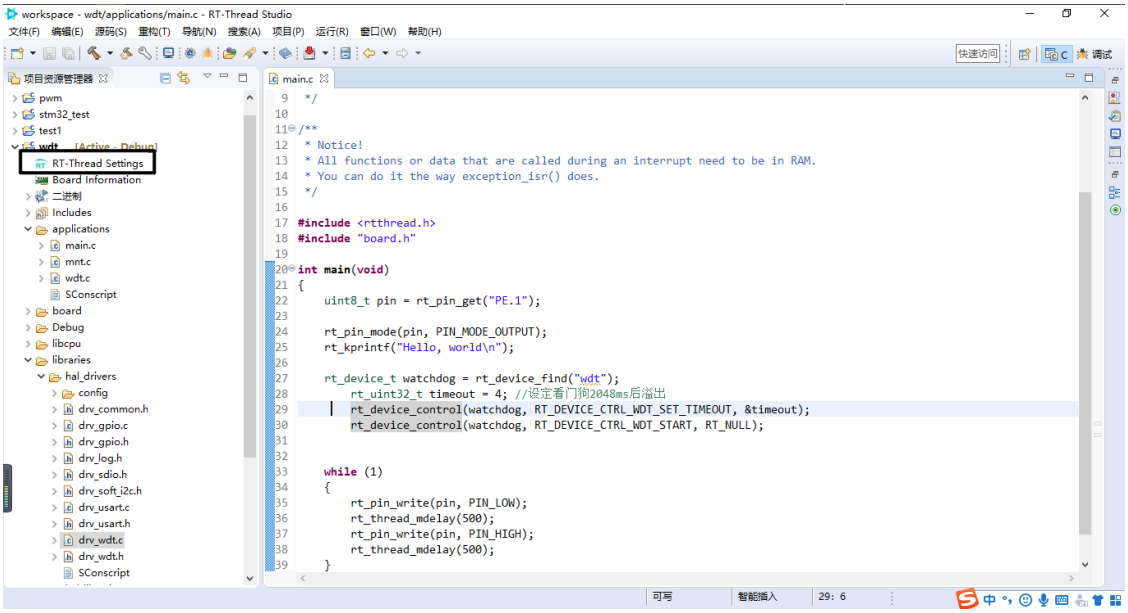
2.步骤说明

2.1 新建工程

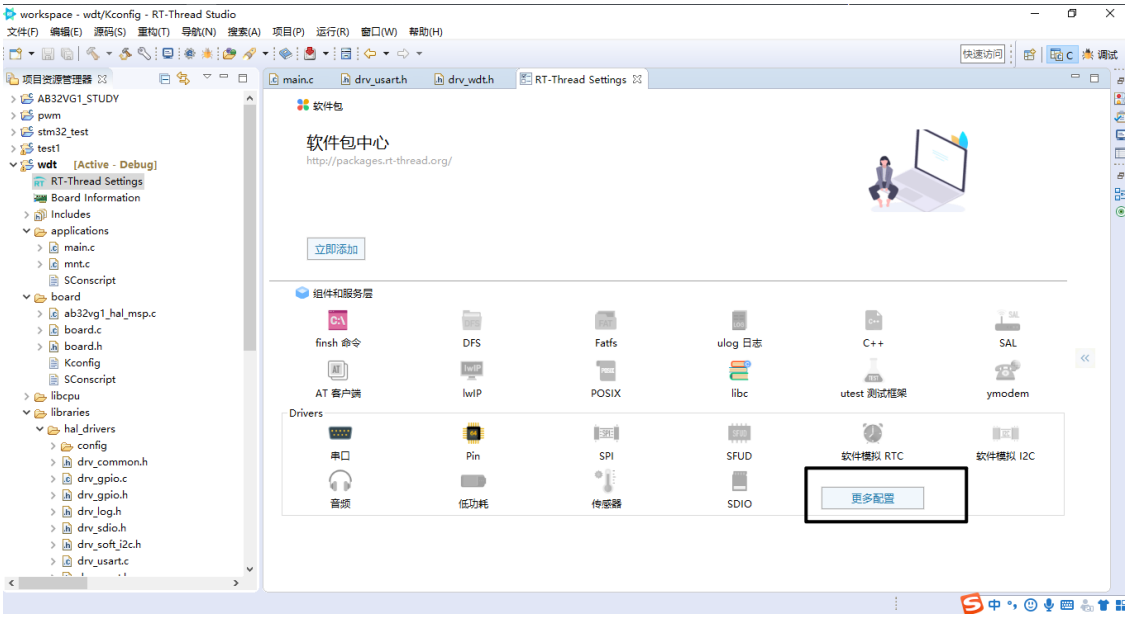
新建一个名为 wdt 的项目，在此不在赘述。请参考从内部 Flash 读取 WAV 音频播放 。

2.2 配置 WDT

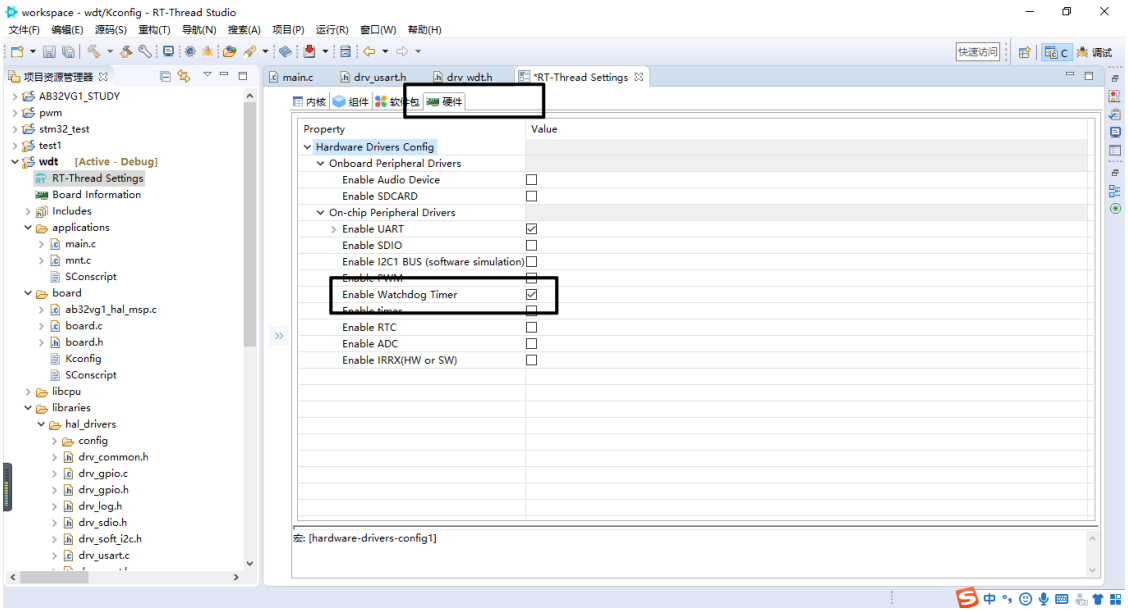
1.双击本项目的 RT-Thread Setting，进入配置软件包页面。



2.进入软件包配置界面，点击更多配置。



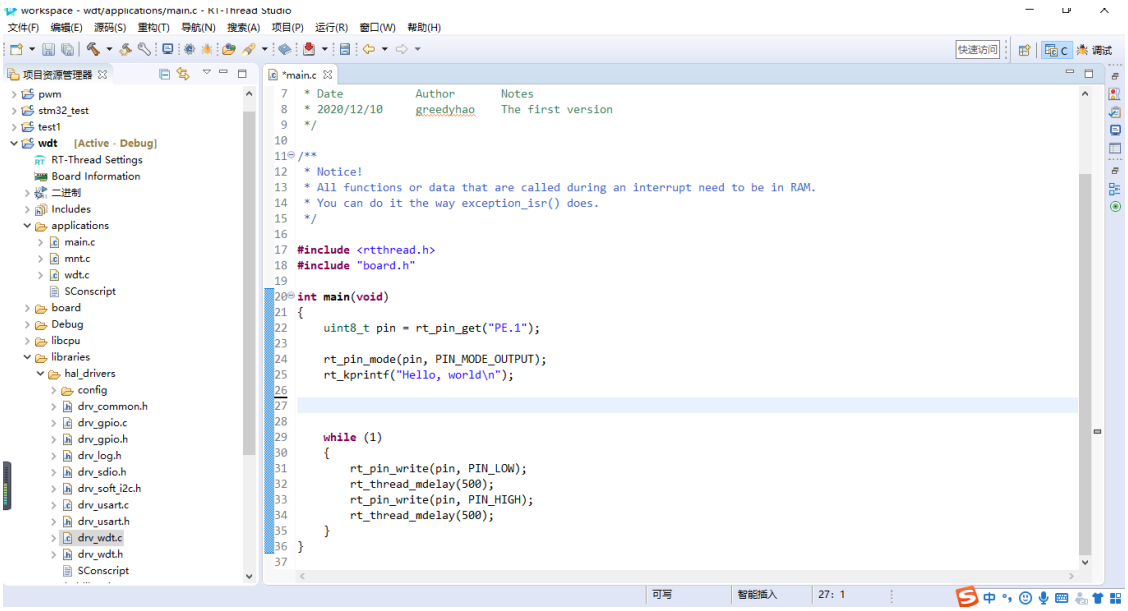
3.在进入更多配置后，选择硬件选项，使能看门狗。



5.保存。

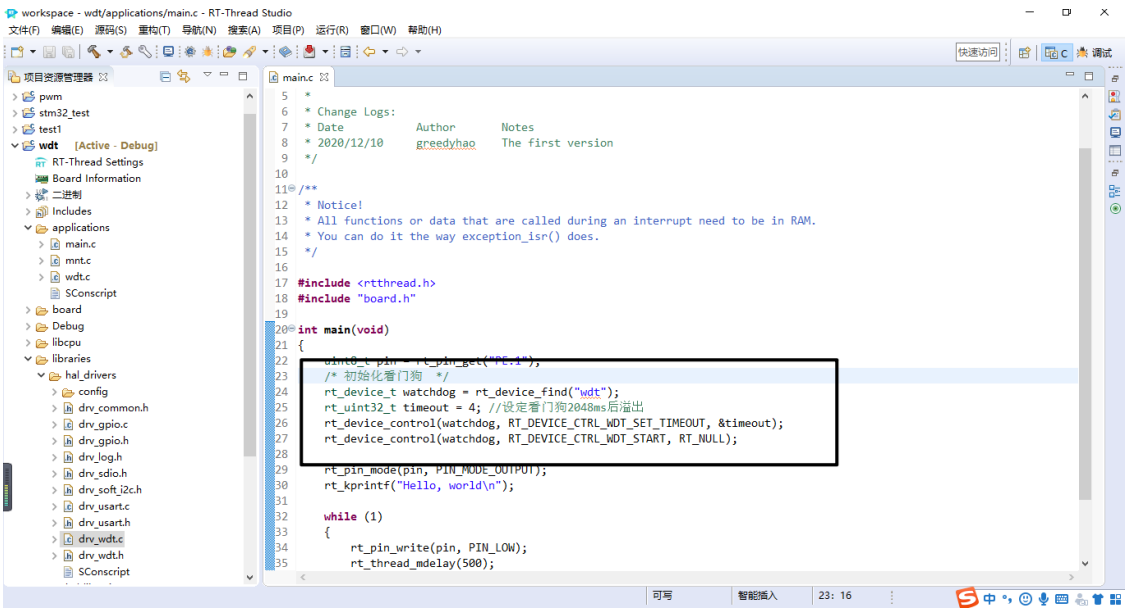
2.3 代码编写

1.进入 main.c



2.将下述代码粘贴 main 函数中，下述代码未进行喂狗操作，每隔 2048ms，硬件将会重启。

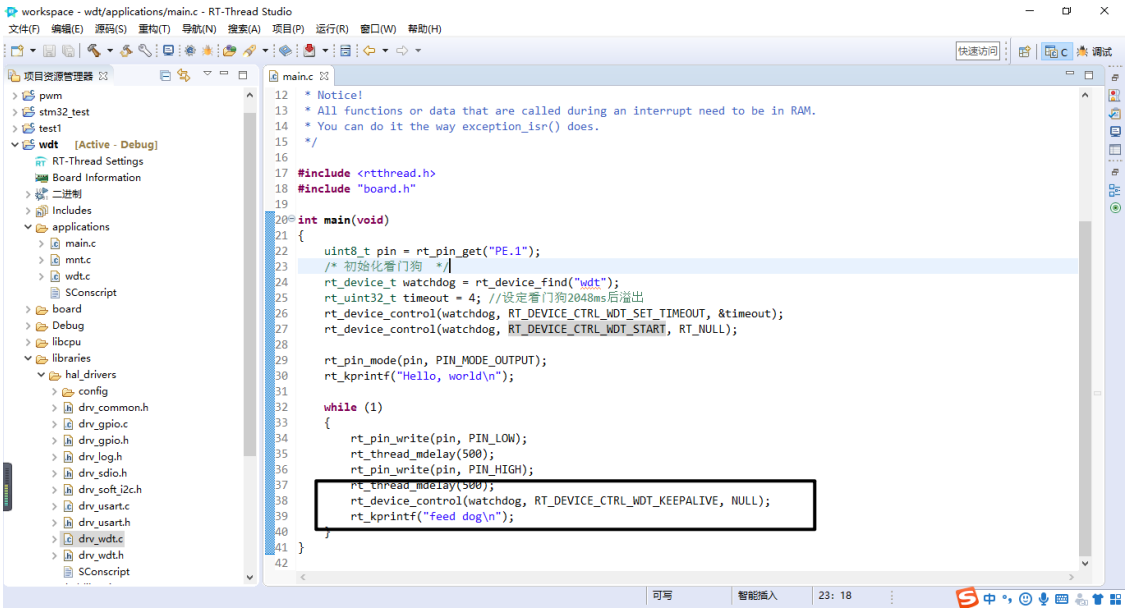
```
rt_device_t watchdog = rt_device_find("wdt");
rt_uint32_t timeout = 4; //设定看门狗 2048ms 后溢出
rt_device_control(watchdog, RT_DEVICE_CTRL_WDT_SET_TIMEOUT, &timeout);
rt_device_control(watchdog, RT_DEVICE_CTRL_WDT_START, RT_NULL);
```



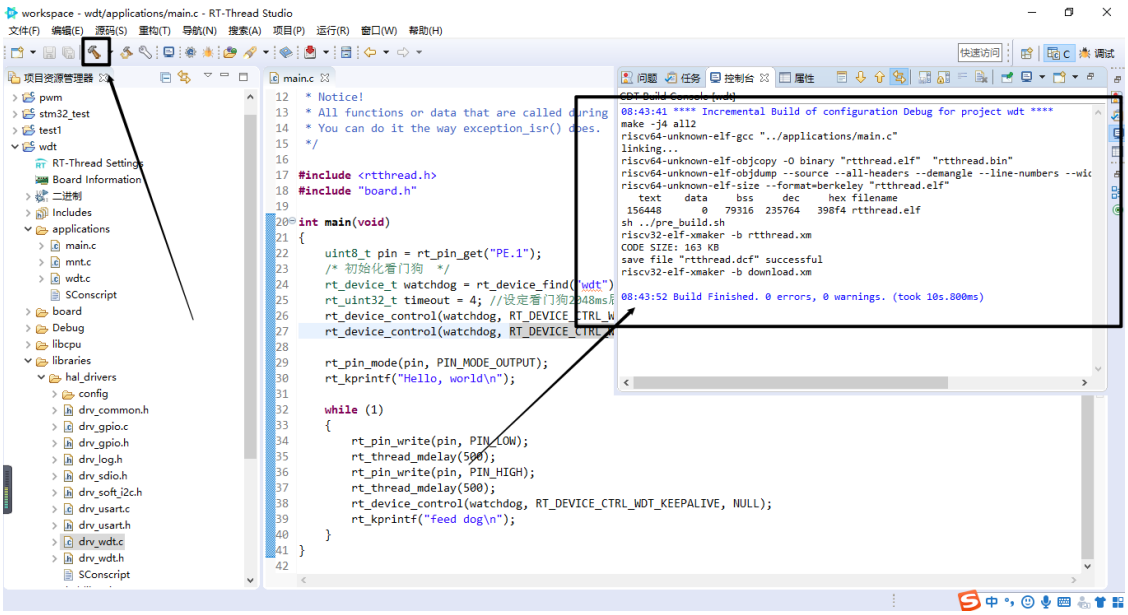
3.在 while 中，加入喂狗程序段。

rt_device_control(watchdog, RT_DEVICE_CTRL_WDT_KEEPALIVE, NULL);

rt_kprintf("feed dog\n");



4.编译

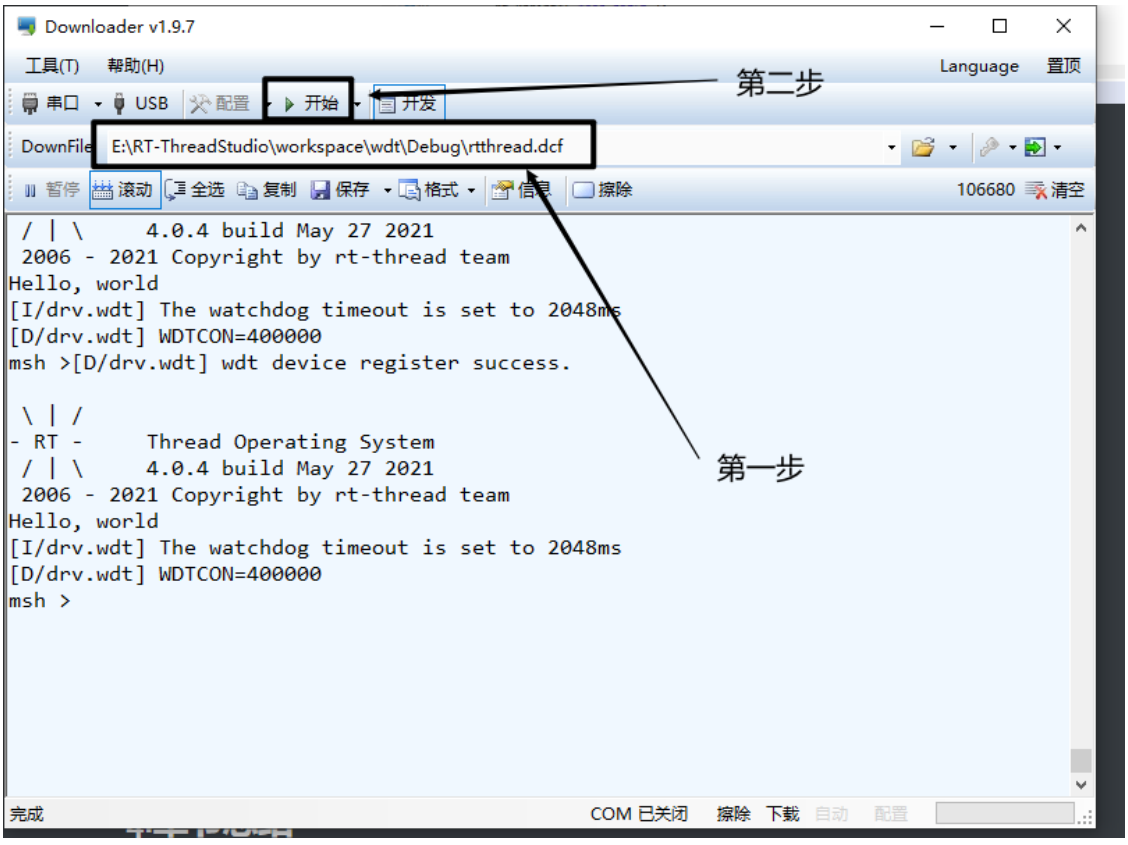


3.代码验证

3.1 下载

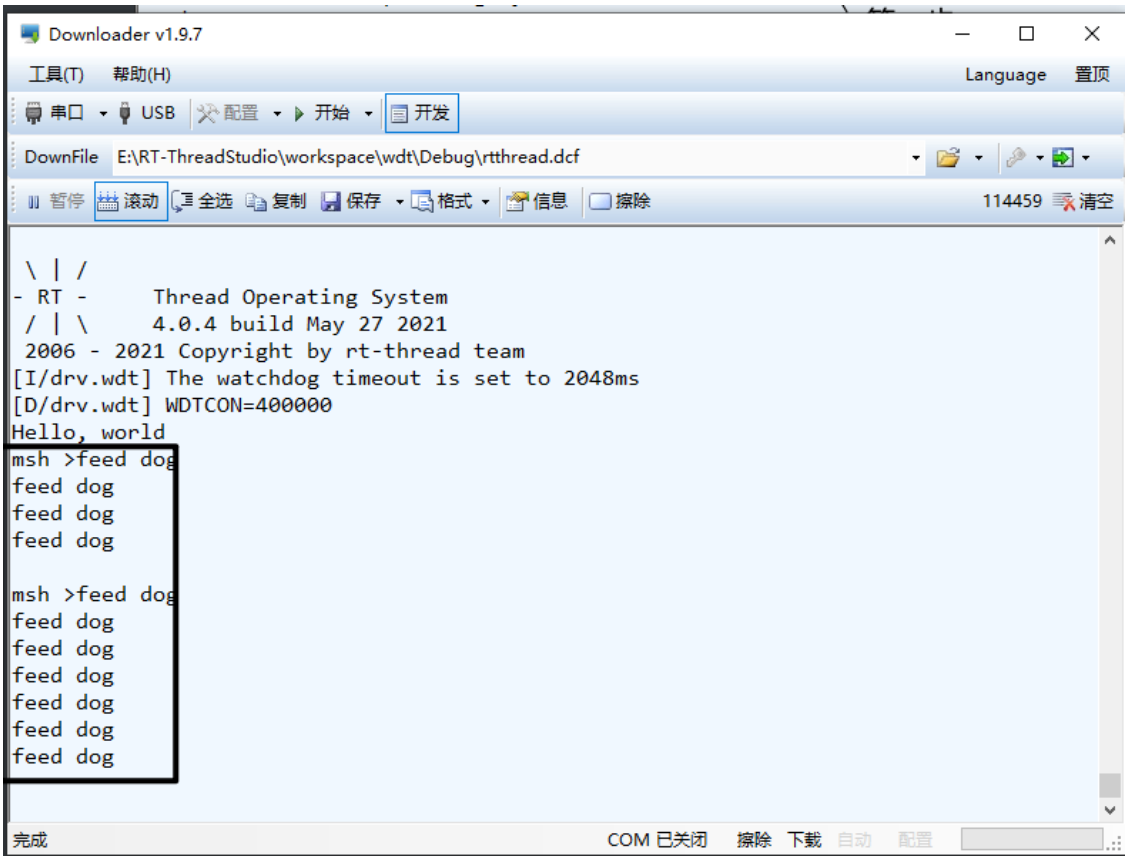
下载使用下载工具 Downloader。第一步：选择你的编译文件。第二步：点击下载即可。

在完成下载后，手动重启，Downloader 中会打印出 Hello Word，说明程序下载成功。



3.2 验证

在 Downloader 中观察，会发现串口会一直发送 feed dog，说明此时喂狗成功。



4.章节总结

本次开门狗实验须注意的是看门狗的最大喂狗时长，具体参数见 drv_wdt.h。其次，在使用看门狗时，要注意及时喂狗。

九、中科蓝讯 AB32VG1 上的 RTC 实践

1. 前言说明

1.1 □□□□

本章介绍基于 rtthread studio 的 sdk 开发 ab32vg1 的 rtc 外设。

1.2 □□□□

AB32VG1 有内置的 RTC 模块.

7. 支持 32 bit 的独立时钟供电
8. 支持闹钟中断以及秒中断

1.3 □□□□□

```
static void _init_rtc_clock(void)
{
    uint8_t rtccon0;
    uint8_t rtccon2;

    rtccon0 = irtc_sfr_read(RTCCON0_CMD);
    rtccon2 = irtc_sfr_read(RTCCON2_CMD);
#ifdef RTC_USING_INTERNAL_CLK
    rtccon0 &= ~RTC_CON0_XOSC32K_ENABLE;
    rtccon0 |= RTC_CON0_INTERNAL_32K;
    rtccon2 | RTC_CON2_32K_SELECT;
#else
    rtccon0 |= RTC_CON0_XOSC32K_ENABLE;
    rtccon0 &= ~RTC_CON0_INTERNAL_32K;
    rtccon2 & ~RTC_CON2_32K_SELECT;
#endif
    irtc_sfr_write(RTCCON0_CMD, rtccon0);
    irtc_sfr_write(RTCCON2_CMD, rtccon2);
}
```

针对设置 RTC 以及读取 RTC 的时钟数据,核心都是在读写寄存器:

Register 5-4 RTCCNT: RTC counter Register

Bit	Name	Mode	Default	Description
31:0	RTCCNT	WR	0x0	32bit RTC counter

在读去当前 RTC 时间时具体的代码为:

```
uint32_t irtc_time_read(uint32_t cmd)
{
```

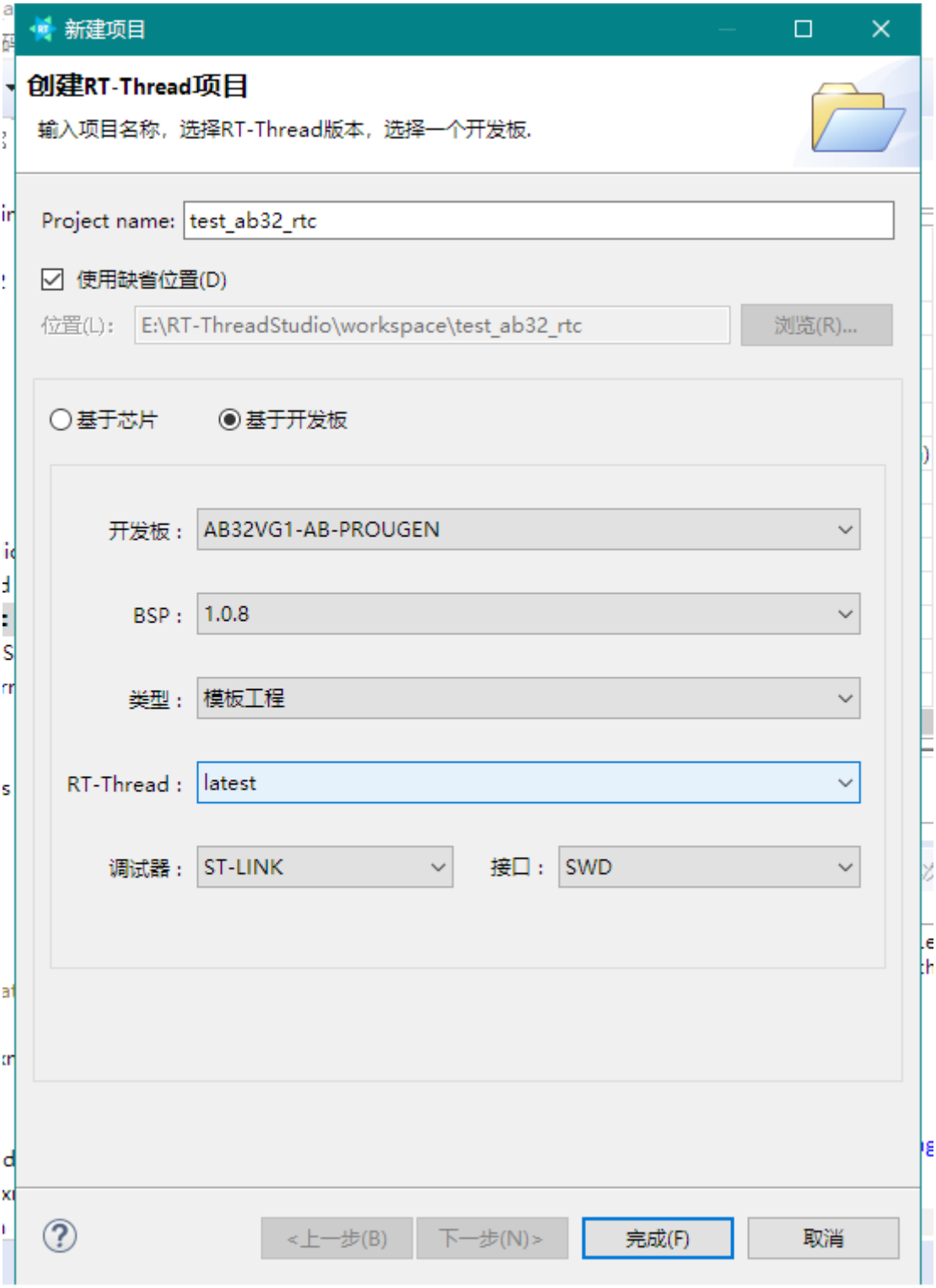
```
    uint32_t rd_val;
    IRTC_ENTER_CRITICAL();
    RTCCON |= RTC_CON_CHIP_SELECT;
    irtc_write(cmd | RTC_RD);
    *((uint8_t *)&rd_val + 3) = irtc_read();
    *((uint8_t *)&rd_val + 2) = irtc_read();
    *((uint8_t *)&rd_val + 1) = irtc_read();
    *((uint8_t *)&rd_val + 0) = irtc_read();
    RTCCON &= ~RTC_CON_CHIP_SELECT;
    IRTC_EXIT_CRITICAL();
    return rd_val;
}
```

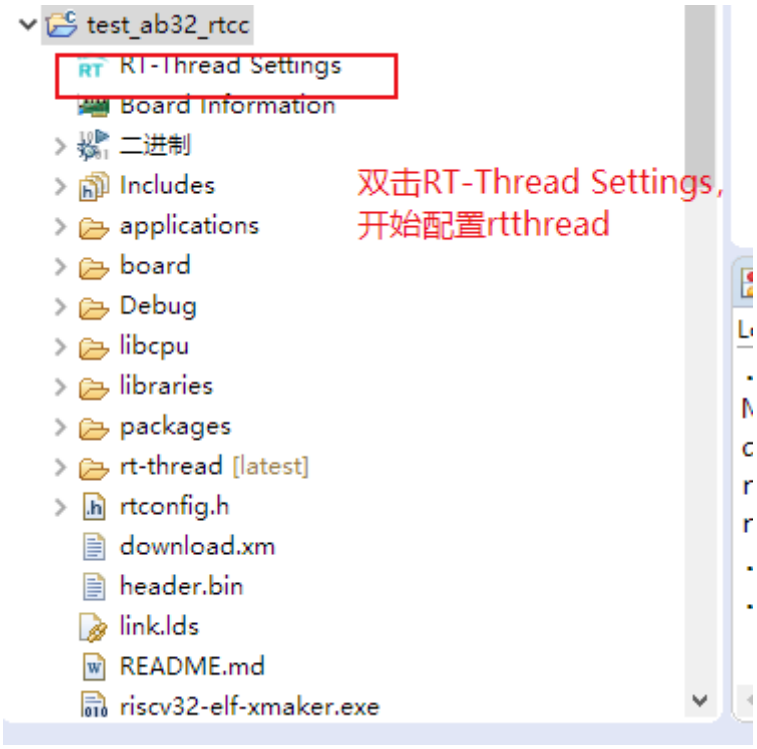
根据 RT-Thread 的 RTC 驱动框架的结构,读写 RTC 都会下发到具体 RTC 驱动 control 句柄的 **RT*DEVICE*CTRL*RTC*GET_TIM** 和 **RT*DEVICE*CTRL*RTC*SET_TIME** 命令参数中.读取时间就是获取 RTCCON 寄存器的数据,单位是 s, 从 1970-01-01 00:00:00 +0000 (UTC) 到当前时间经历的秒数.配置时间亦然,将从 1970-01-01 00:00:00 +0000 (UTC) 到指定的时间经历的秒数写到 RTCCON 寄存器.对应的代码是:

```
void irtc_time_write(uint32_t cmd, uint32_t dat)
{
    IRTC_ENTER_CRITICAL();
    RTCCON |= RTC_CON_CHIP_SELECT;
    irtc_write(cmd | RTC_WR);
    irtc_write((uint8_t)(dat >> 24));
    irtc_write((uint8_t)(dat >> 16));
    irtc_write((uint8_t)(dat >>  8));
    irtc_write((uint8_t)(dat >>  0));
    RTCCON &= ~RTC_CON_CHIP_SELECT;
    IRTC_EXIT_CRITICAL();
}
```

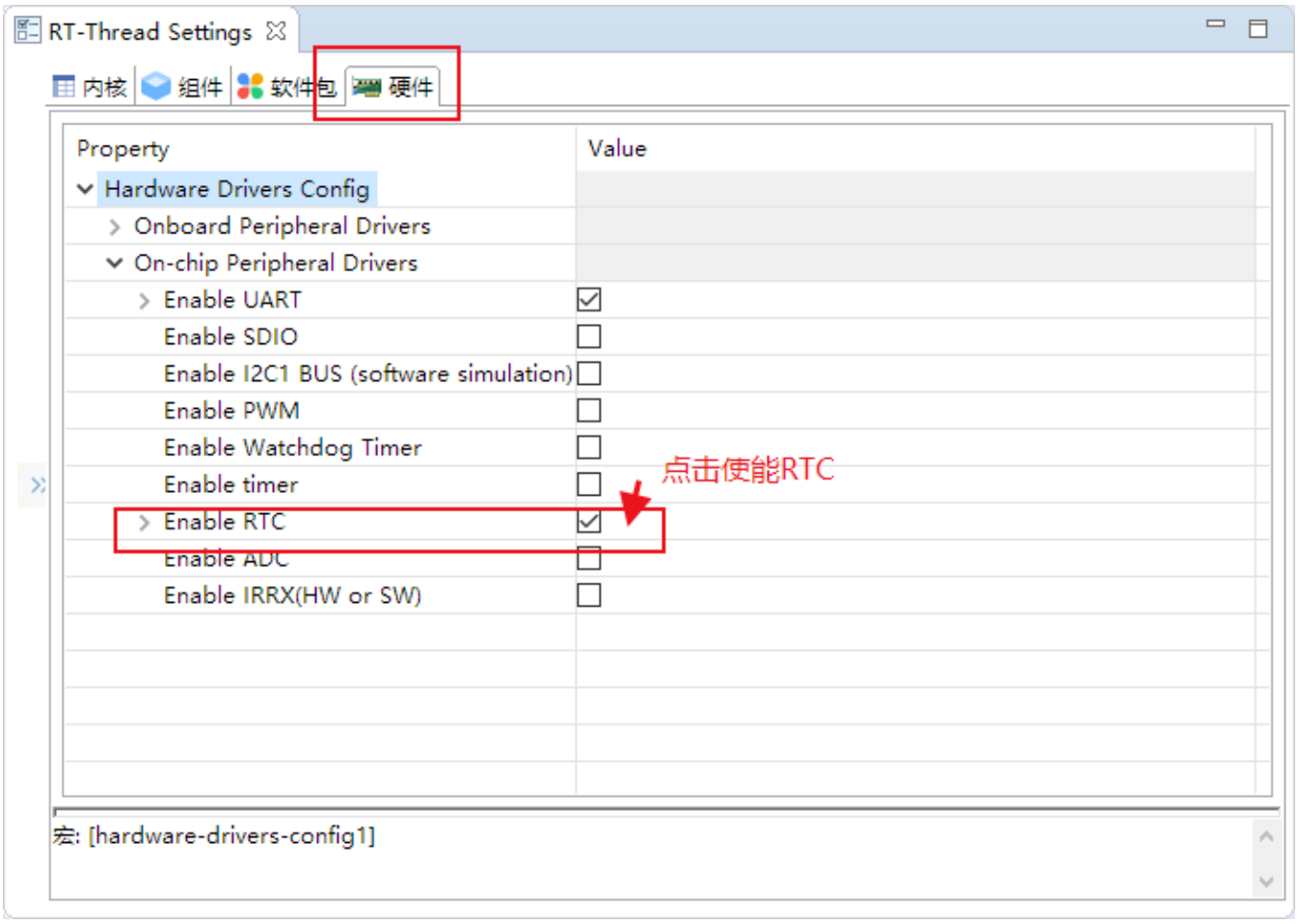
2. 步骤说明

2.1 新建工程和配置

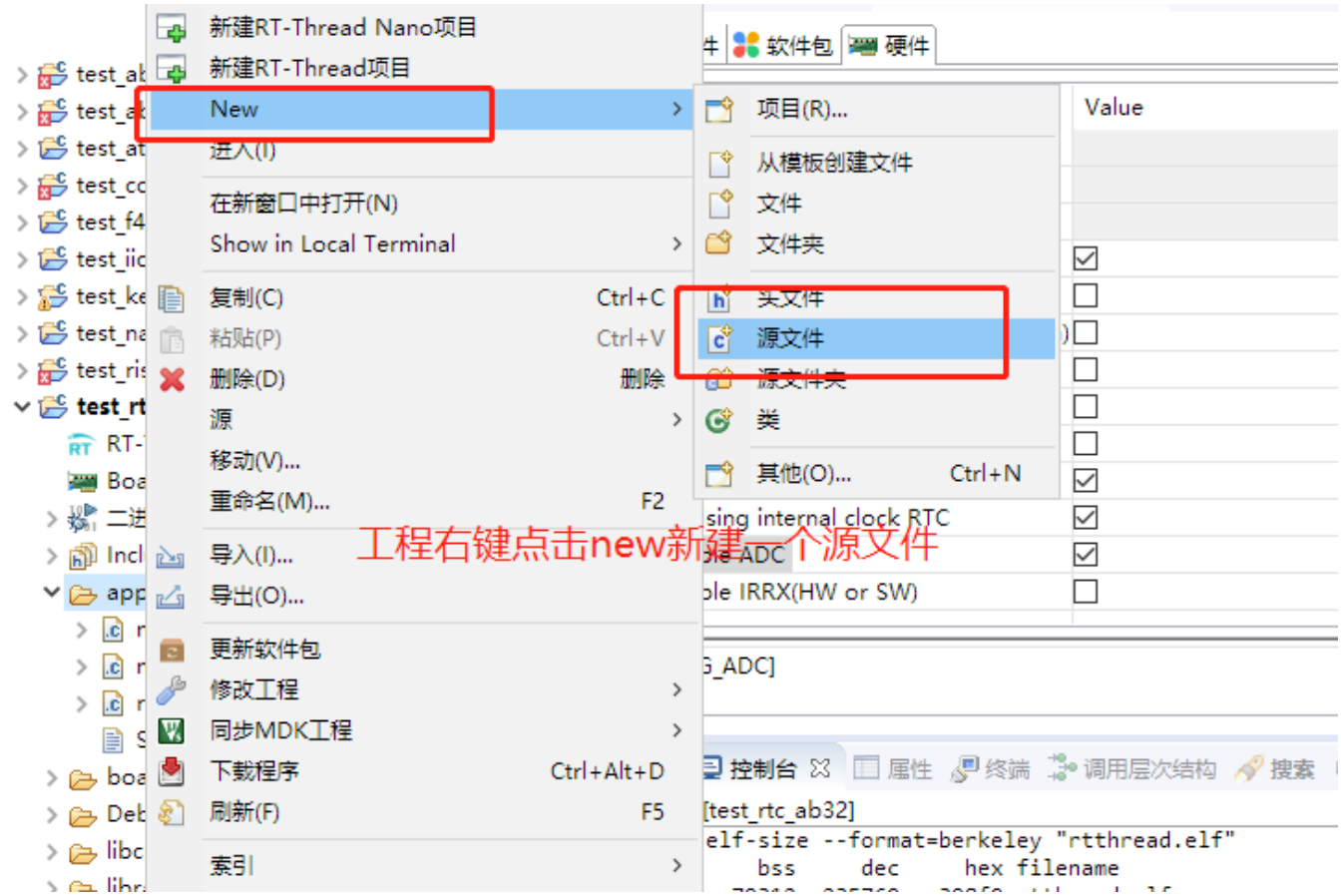


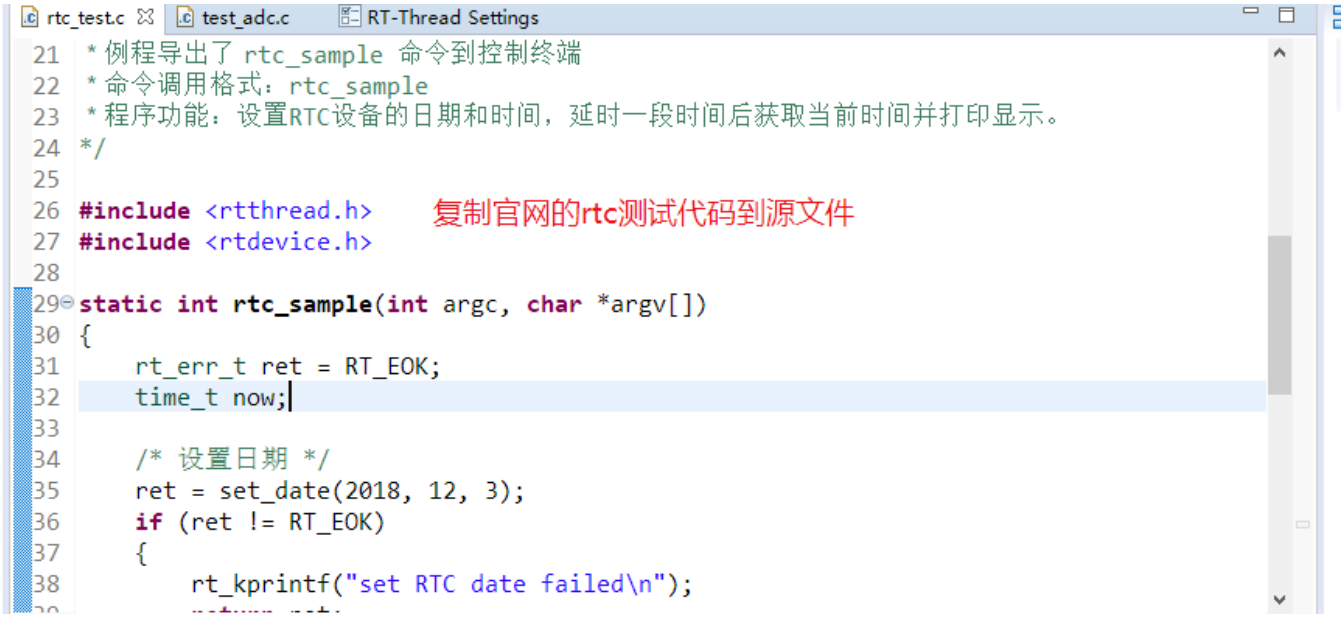

test_ab32_rtcc
RT-Thread Settings
Board Information
二进制
Includes
applications
board
Debug
libcpu
libraries
packages
rt-thread [latest]
rtconfig.h
download.xm
header.bin
link.lds
README.md
riscv32-elf-xmaker.exe
双击RT-Thread Settings，
开始配置rtthread


RT-Thread Settings
软件包
软件包中心
http://packages.rt-thread.org/
立即添加
组件和服务层
finsh 命令
DFS
Fatfs
ulog 日志
C++
SAL
AT 客户端
lwIP
POSIX
libc
utest 测试
ymodem
Drivers
串口
Pin
SPI
SFUD
软件模拟 RTC
软件模拟 I2C
音频
低功耗
传感器
SDIO
更多配置
点击更多配置，进入
配置界面

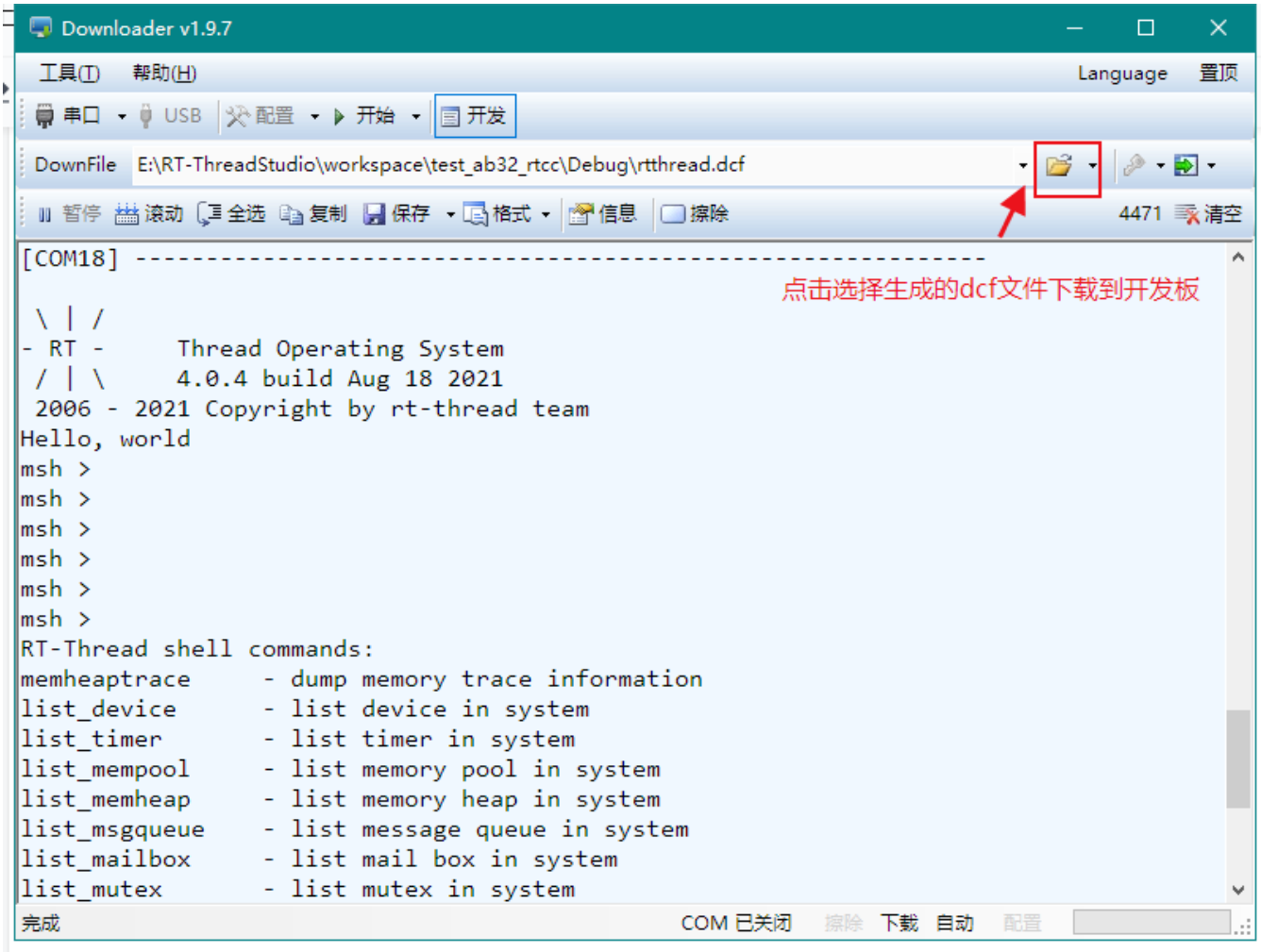

RT-Thread Settings
内核
组件
软件包
硬件
Property
Value
Hardware Drivers Config
Onboard Peripheral Drivers
On-chip Peripheral Drivers
Enable UART
Enable SDIO
Enable I2C1 BUS (software simulation)
Enable PWM
Enable Watchdog Timer
Enable timer
Enable RTC
Enable ADC
Enable IRRX(HW or SW)
点击使能RTC
宏: [hardware-drivers-config1]

2.2 添加测试文件





2.3 编译和下载



2.4 现象

在 finsh 输入 tab 键弹出支持的命令，输入 date 即可查看当前的时间。

```
list_device       - list device in system
list_timer        - list timer in system
list_mempool      - list memory pool in system
list_memheap      - list memory heap in system
list_msgqueue     - list message queue in system
list_mailbox      - list mail box in system
list_mutex        - list mutex in system
list_event        - list event in system
list_sem          - list semaphore in system
list_thread       - list thread
version           - show RT-Thread version information
clear             - clear the terminal screen
free              - Show the memory usage in the system.
ps                - List threads in the system.
help              - RT-Thread shell help.
exit              - return to RT-Thread shell mode.
date              - get date and time or set (local timezone) [year month day hour
min sec]

msh >date
Wed Aug 18 11:15:56 2021
msh >
```

输入date，查看时间，与设置的时间一致。

3 代码检验

通过对 RTC 驱动代码的见但描述,接下来将 RT-Thread 官网有关 RTC 部分的测试代码稍作改动,实现上电自动设置一次系统时间,然后终端打印出新的 RTC 时间.

```
/*
 * 程序清单：这是一个 RTC 设备使用例程
 * 例程导出了 rtc_sample 命令到控制终端
 * 命令调用格式：rtc_sample
 * 程序功能：设置 RTC 设备的日期和时间，延时一段时间后获取当前时间并打印显示。
*/
#include <rtthread.h>
#include <rtdevice.h>

static int rtc_sample(void)
{
    rt_err_t ret = RT_EOK;
    time_t now;

    /* 设置日期 */
    ret = set_date(2021, 06, 02);
    if (ret != RT_EOK)
    {
        rt_kprintf("set RTC date failed\n");
```

```
        return ret;
    }

    /* 设置时间 */
    ret = set_time(11, 15, 50);
    if (ret != RT_EOK)
    {
        rt_kprintf("set RTC time failed\n");
        return ret;
    }

    /* 延时 3 秒 */
    rt_thread_mdelay(3000);

    /* 获取时间 */
    now = time(RT_NULL);
    rt_kprintf("%s\n", ctime(&now));

    return ret;
}
/* 在 APP 级自动执行该函数 */
INIT_APP_EXPORT(rtc_sample);
```

4. 章节总结

本章介绍 rt-thread studio 开发 ab32vg1 的 rtc 外设，新建工程后配置 rtthread setting，接着保存，新建测试文件，编译下载即可完成一个 rtc 应用。

十、中科蓝讯 AB32VG1 上的 SDIO 实践

1.前言说明

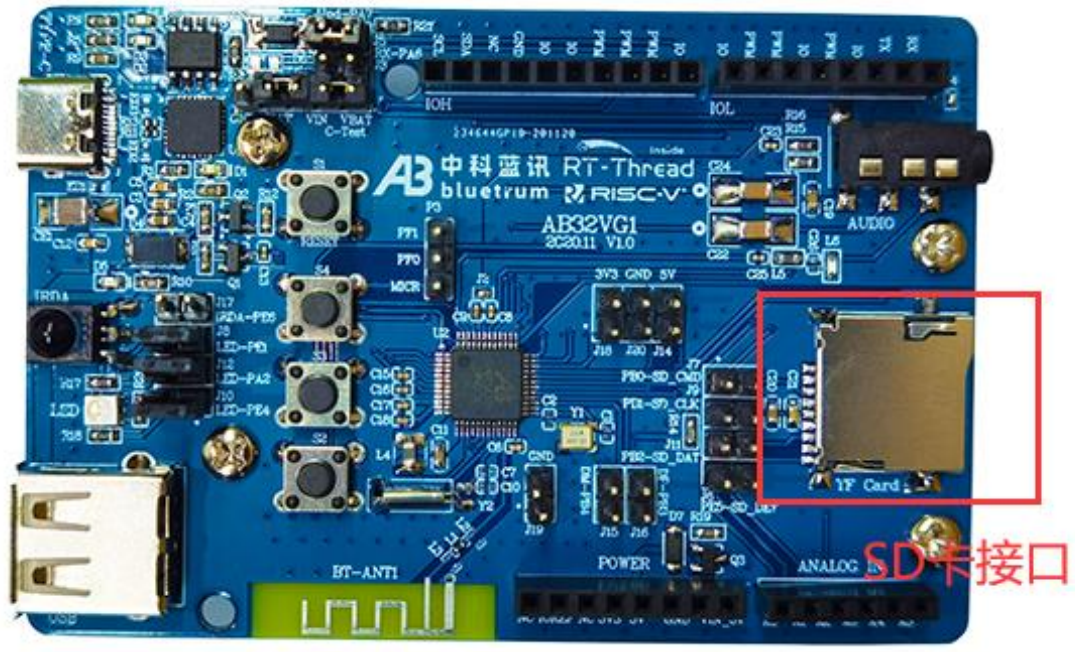
1.1 本章内容

本章通过 RT-Thread Studio 配置 AB32VG1 片上 SDIO 外设，快速使用 SDIO 功能，

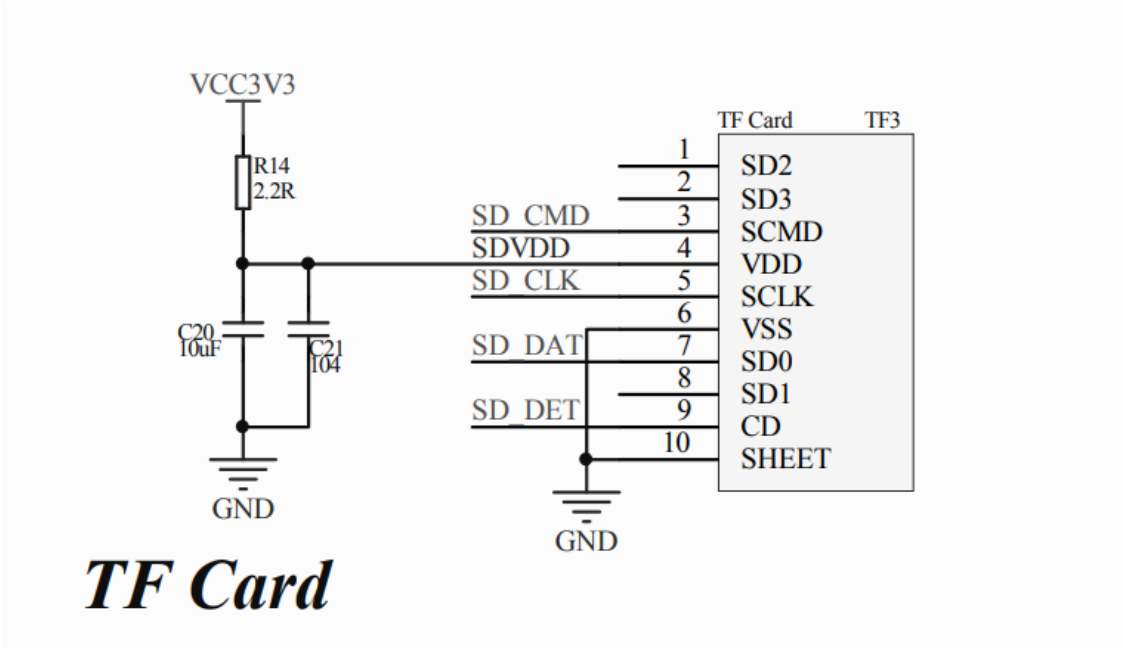
读写 SD 卡。

1.2 模块介绍

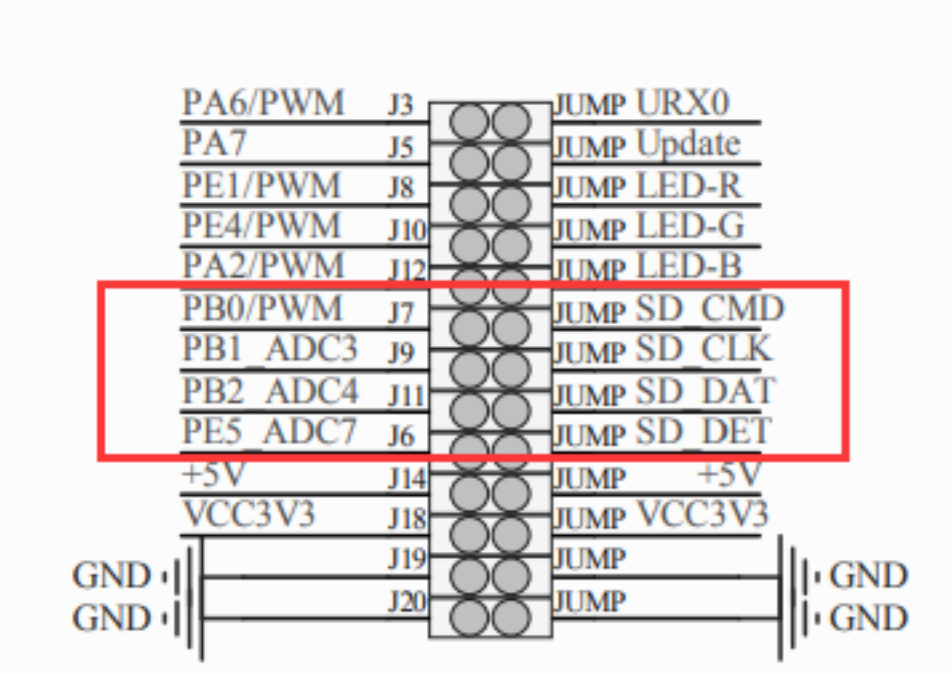
SD 卡接口位置



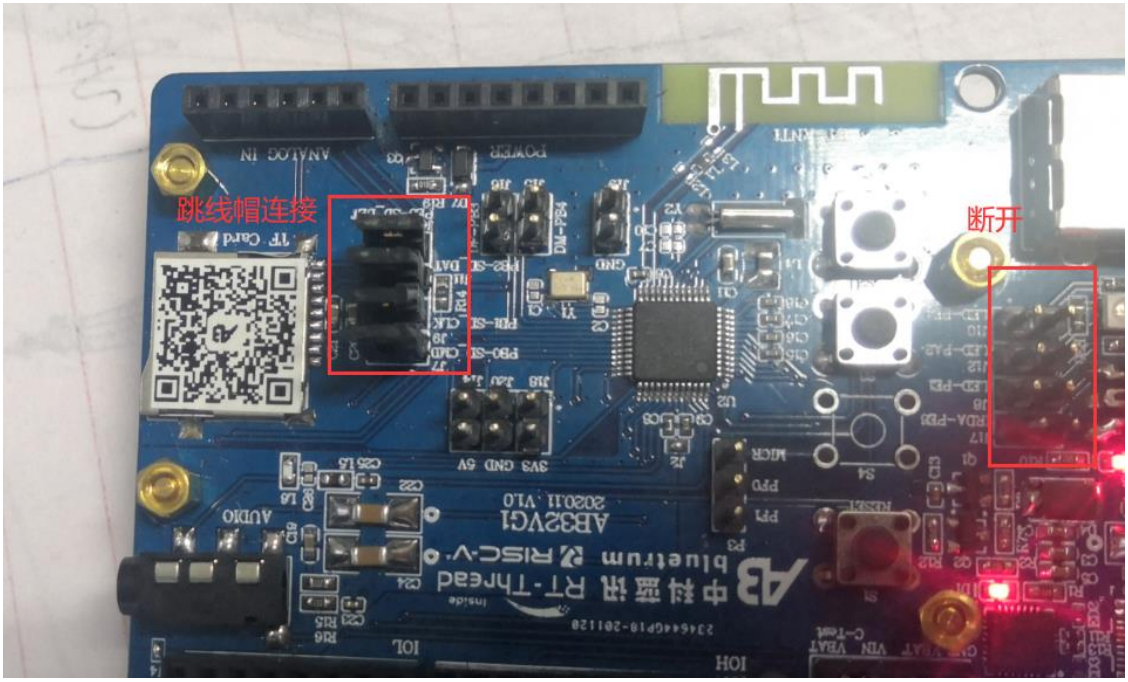
TF Card 接口原理图



通过跳线帽将 TF Card 接口与芯片 SDIO 接口连接



因为开发板引脚冲突，需要将图上右侧位置跳线帽断开，SD 卡处的跳线帽插上



1.3 开发软件

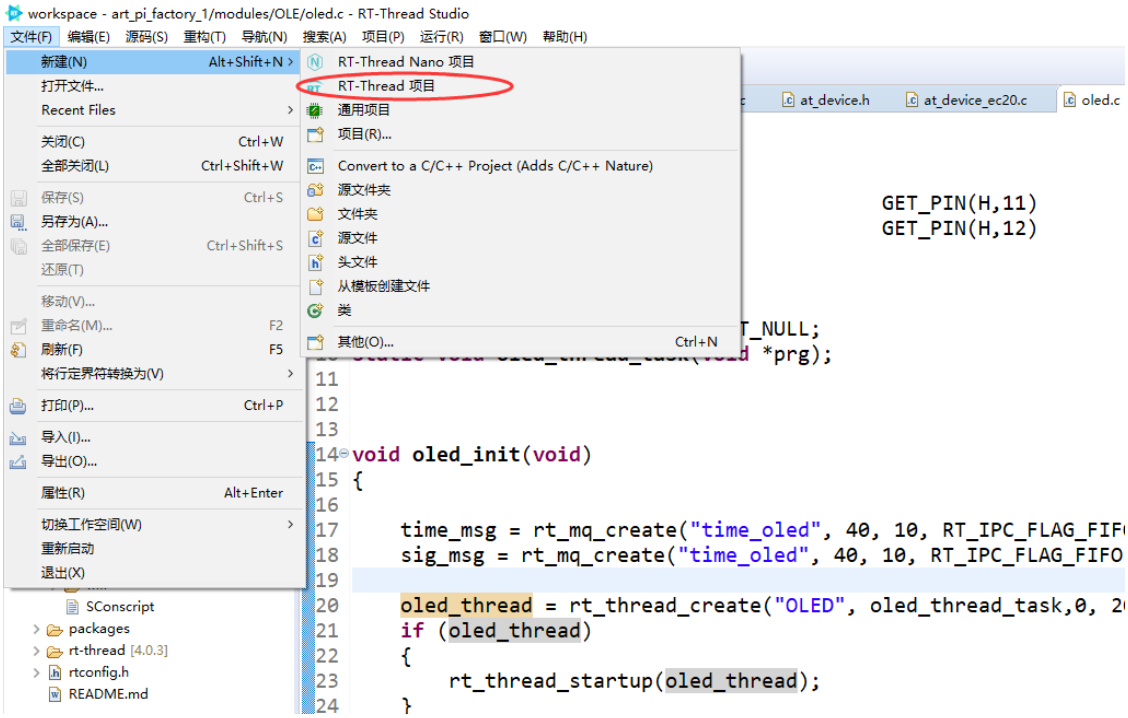
编译平台：**RT-Thread Studio：** 安装链接

下载平台：**Downloader:** 安装链接

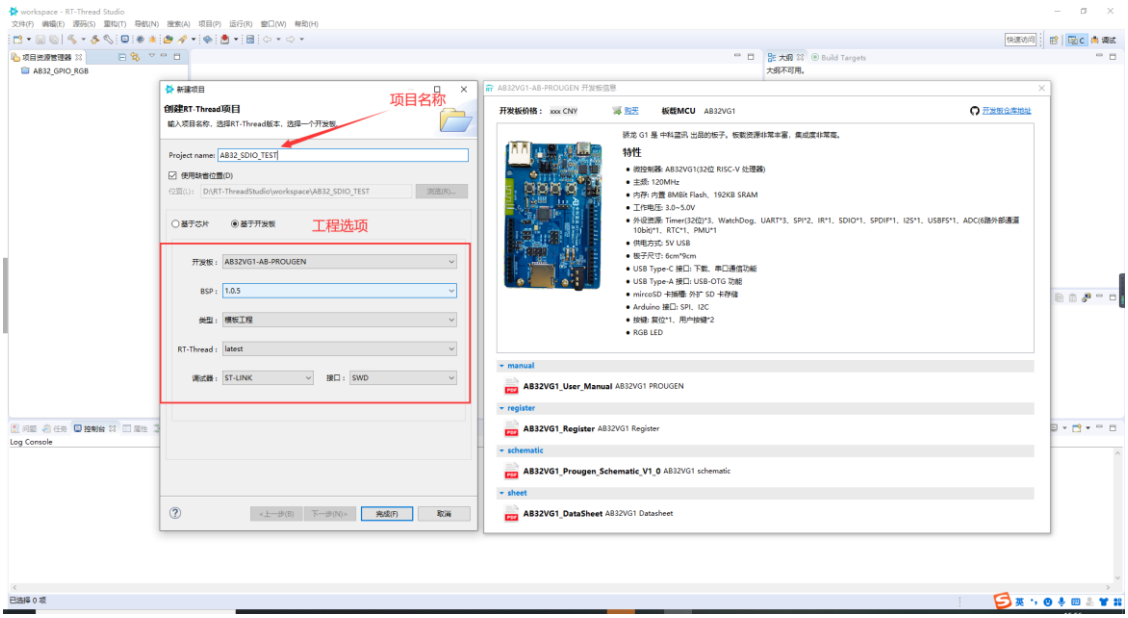
2.步骤说明

2.1 新建工程

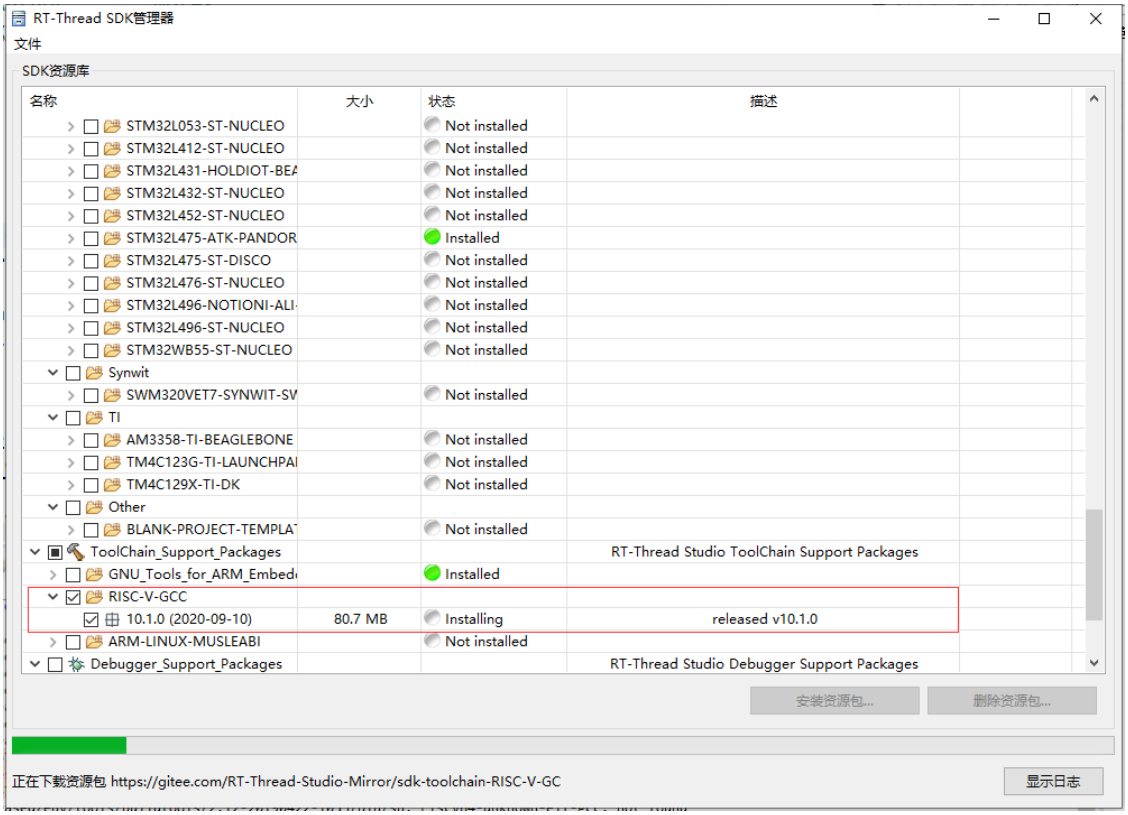
点击 文件-> 新建-> RT-Thread 项目控件



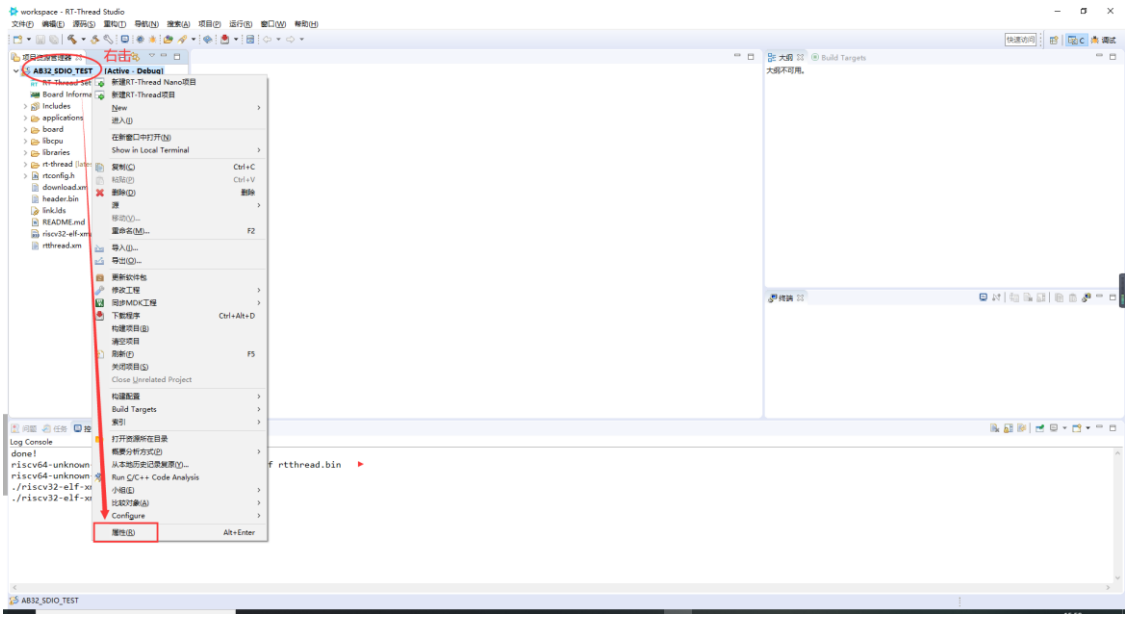
选择基于开发板的项目，填写工程名字，选择我们**使用到的开发板（AB32VG1）**，调试器我们随便选，下载方式不是通过此处下载



注意：如果第一次使用 RISC-V 芯片需要安装工具链，在 SDK 管理器中下载工具链

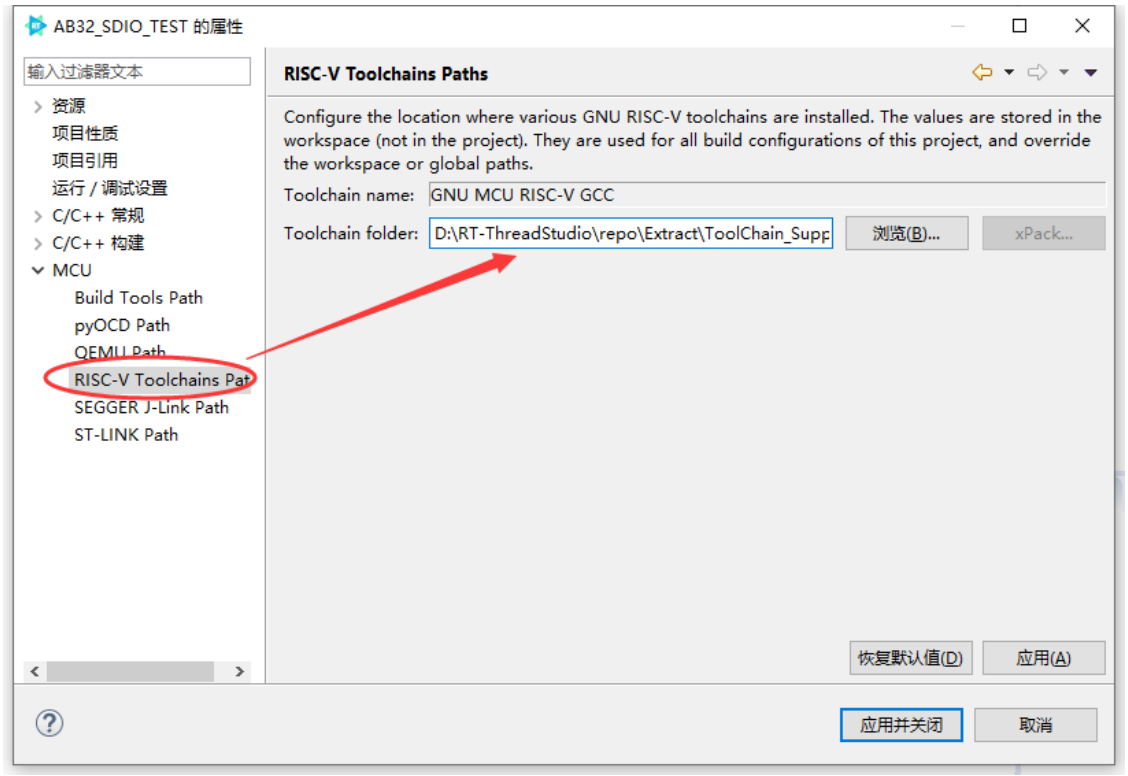


然右击项目名称，进入属性

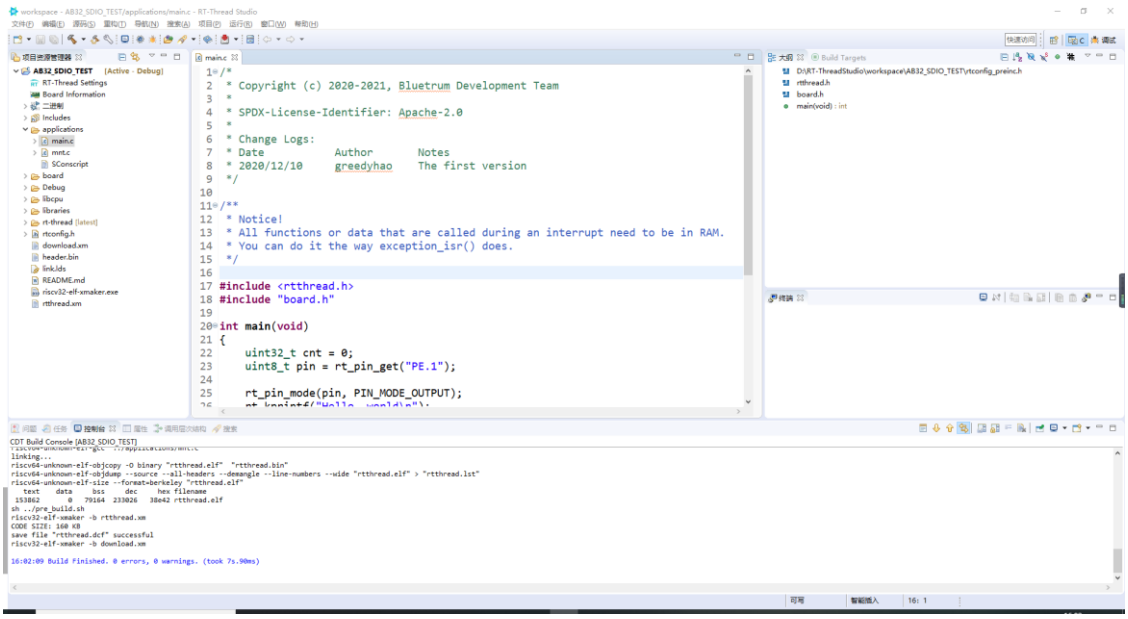


找到 MCU->RISC-V ToolchainsPat ，配置 Tool 的环境，在软件安装位置下面的路径中

软件安装位置 \RT-ThreadStudio\repo\Extract\ToolChain_Support_Packages\RISC-V\RISC-V-GCC\10.1.0\bin



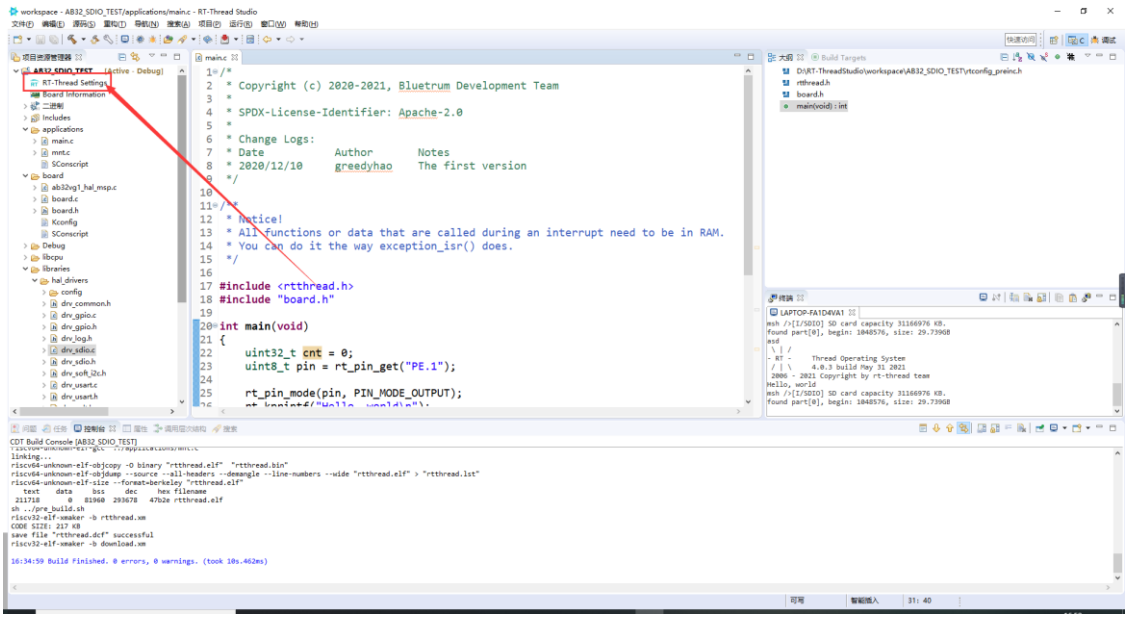
工程新建后左边的项目资源管理器会显示我们的工程，我们把他展开，点击小锤子图标编译一下，编译结果如下



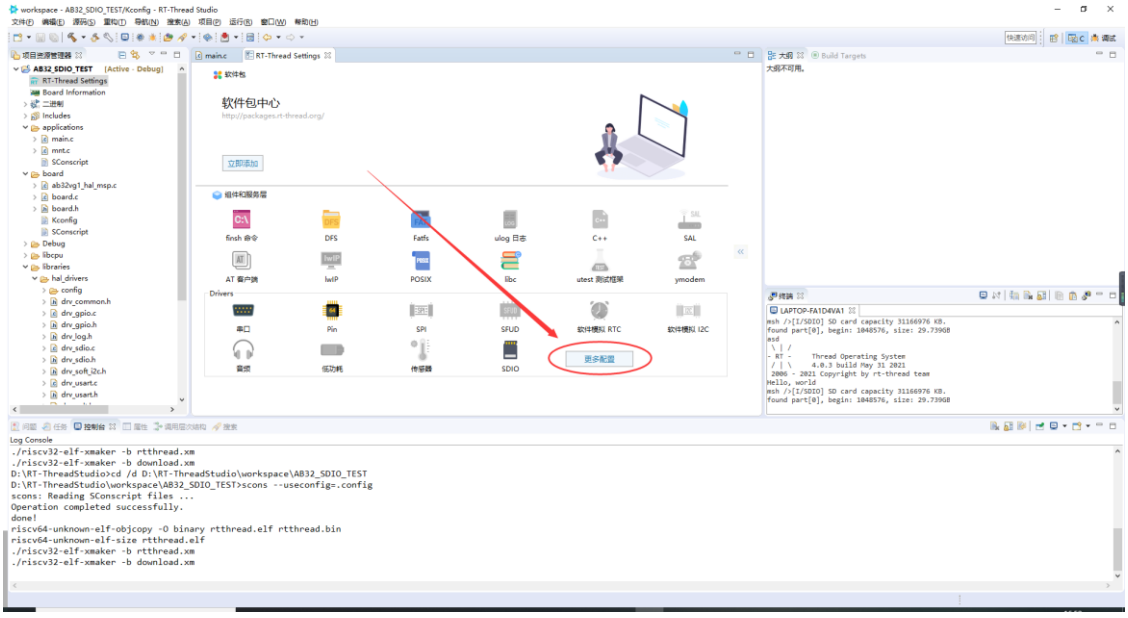
编译无报错，新建工程完成了！

2.2 RT-Thread Studio 配置 SDIO

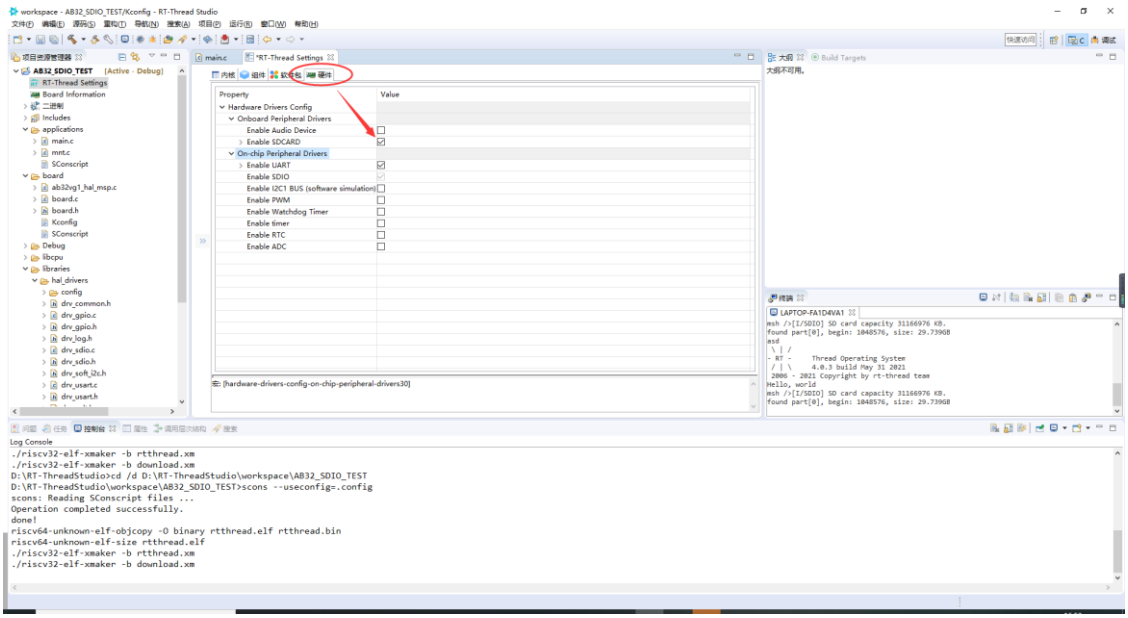
点击 RT-Thread Setting



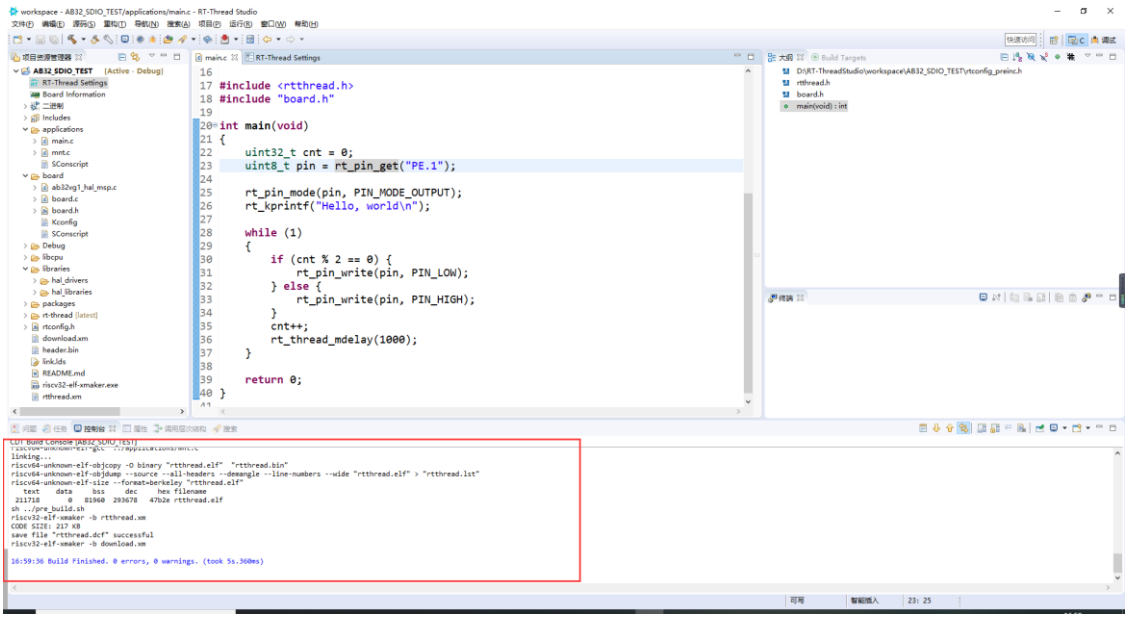
选择更多配置



点击硬件，展开后选择使能 SD 卡



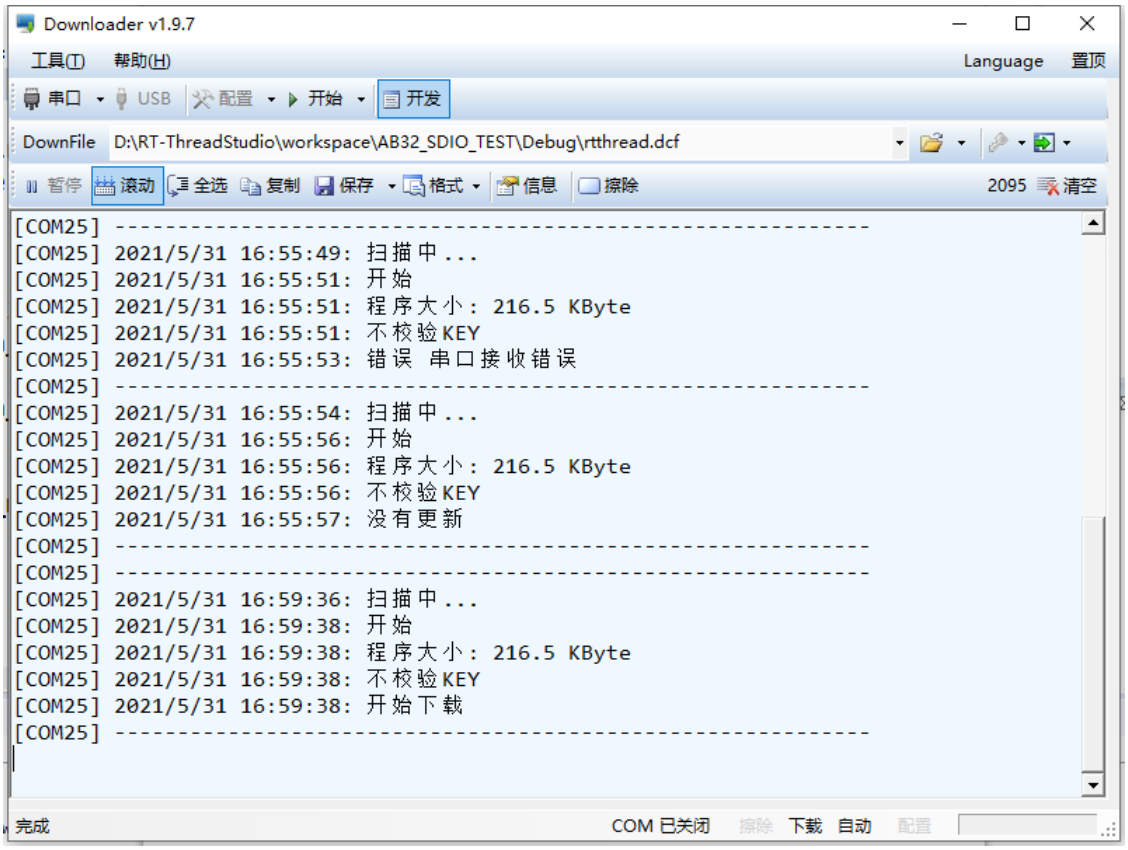
Ctrl + S 保存，RT-Thread 会自动生成代码，生成之后，我们回到工程文件，编译一下工程



编译无报错，SDIO 功能添加完成，下面就是验证 SD 卡功能

3.代码验证

编译完成，打开 Downloaded 下载器，通过 download 下载生成的.dcf 文件（第一次使用前需要先安装串口驱动），扫描串口，点击开始后，按一下板子上复位按键下载程序



程序下载完成后可以通过 Downloader 使用命令行进行在线调试，如下 ls 列出挂载的 SD 卡上的列表，mkdir 创建新的目录

```
[COM4] ----------------------------------------------------------
[COM4] 2021/3/24 11:59:29: 扫描中...
[COM4] 2021/3/24 11:59:30: 开始
[COM4] 2021/3/24 11:59:30: 程序大小: 218.5 KByte
[COM4] 2021/3/24 11:59:30: 不校验KEY
[COM4] 2021/3/24 11:59:30: 没有更新
[COM4] ----------------------------------------------------------

 \ | /
- RT -     Thread Operating System
 / | \     4.0.3 build Mar 24 2021
 2006 - 2021 Copyright by rt-thread team
Hello, world
msh />[I/SDIO] SD card capacity 15558144 KB.
[D/SDIO] probe mmcsd block device!
found part[0], begin: 4194304, size: 14.853GB

msh />
msh />
msh />ls
Directory /:
msh />mkdir hello
msh />
msh />cd hello
msh /hello>mkdir 1
msh /hello>mkdir 2
msh /hello>ls
Directory /hello:
1                   <DIR>
2                   <DIR>
msh /hello>df
disk free: 14.8 GB [ 31099648 block, 512 bytes per block ]
msh /hello>
```

4.章节总结

本章节我们使用 RTT Studio 配置文件系统，然后通过终端进行接口调用，不需要写很多代码，因为挂载文件系统的代码已经封装完成，这样的机制有利于降低开发者的开发时间，使开发者不用了解底层也可快速通过终端对 SD 卡进行配置，减少开发时间。

十一、中科蓝讯 AB32VG1 上的 Flash 实

践

1. 前言说明

1.1. 本章内容

本章通过 RT-Thread Studio 配置 AB32VG1 片上 Flash 的功能，实现文件读写。

1.2. 模块介绍

AB32VG1 内部集成 8M bit 即 1M byte flash，每个扇区 4096 Bytes，由于程序整体不大，所以将后 512KB 用于用户存储区，并通过文件系统挂载。

1.3. 开发软件

开发环境：RT-Thread Studio

下载工具：Downloader.exe

2. 步骤说明

2.1. 新键工程

2.1.1.文件->新键->RT-Thread 项目。

2.1.2.选择基于开发板，填写工程名字。

2.1.3.点完成。一个新的项目就建成了。

2.2. 编写驱动文件

2.2.1.接口函数

关于 AB32VG1 片上 flash 的信息目前还没有公开，目前由@greedyhao 通过编译库的形式提供，函数接口如下：

uint16_t os_spiflash_read(void *buf, uint32_t addr, uint16_t len);

void os_spiflash_program(void *buf, uint32_t addr, uint16_t len);

void os_spiflash_erase(uint32_t addr);

2.2.2 扩展 Flash 驱动

为满足文件系统的正常挂载、使用，对 flash 驱动进行了进一步封装

[flash.h]

```
/*
 * Copyright (c) 2006-2021, RT-Thread Development Team
 *
 * SPDX-License-Identifier: Apache-2.0
 *
 * Change Logs:
 * Date           Author       Notes
 * 2021-03-28     12804        the first version
 */
#ifndef APPLICATIONS_FLASH_H_
#define APPLICATIONS_FLASH_H_

#define  DATA_ADDRESS_START      0x80000
#define  DATA_ADDRESS_END        0x100000

#define  SECTOR_SIZE             4096


void FlashWrite(uint32_t addr, const uint8_t *data, size_t len);
void FlashRead(uint32_t addr, uint8_t *data, size_t len);
void FlashErase(uint32_t addr, size_t len);

#endif /* APPLICATIONS_FLASH_H_ */
```

[flash.c]

```
#include <rtthread.h>
#include "flash.h"
#include <stdlib.h>
uint16_t os_spiflash_read(void *buf, uint32_t addr, uint16_t len);
void os_spiflash_program(void *buf, uint32_t addr, uint16_t len);
void os_spiflash_erase(uint32_t addr);

static uint8_t buff[SECTOR_SIZE];

uint8_t FlashCheckSectorErase(uint32_t addr, uint8_t *data)
{
	uint16_t i;
	FlashRead(addr, data, SECTOR_SIZE);
	for(i=0;i<SECTOR_SIZE;i++)
	{
		if(data[i] != 0xFF)
			return 1;
	}
	return 0;
}

uint32_t	FlashGetSectorAddr(uint32_t addr)
{
	return (addr - (addr % SECTOR_SIZE));
}

uint8_t FlashWritePage(uint32_t pageAddress, uint8_t *data)
{
	uint16_t i;
	for(i=0;i<SECTOR_SIZE/256;i++)
	{
		os_spiflash_program(&data[256*i], pageAddress+256*i, 256);
	}
	return 1;
}


void FlashWrite(uint32_t addr, const uint8_t *data, size_t len)
{
	int i,j;
		if((addr >= DATA_ADDRESS_START) && ((addr + len) < DATA_ADDRESS_END))
		{
			uint32_t sectorAssress;
			uint16_t writed = 0;
			uint16_t toWrite;
```

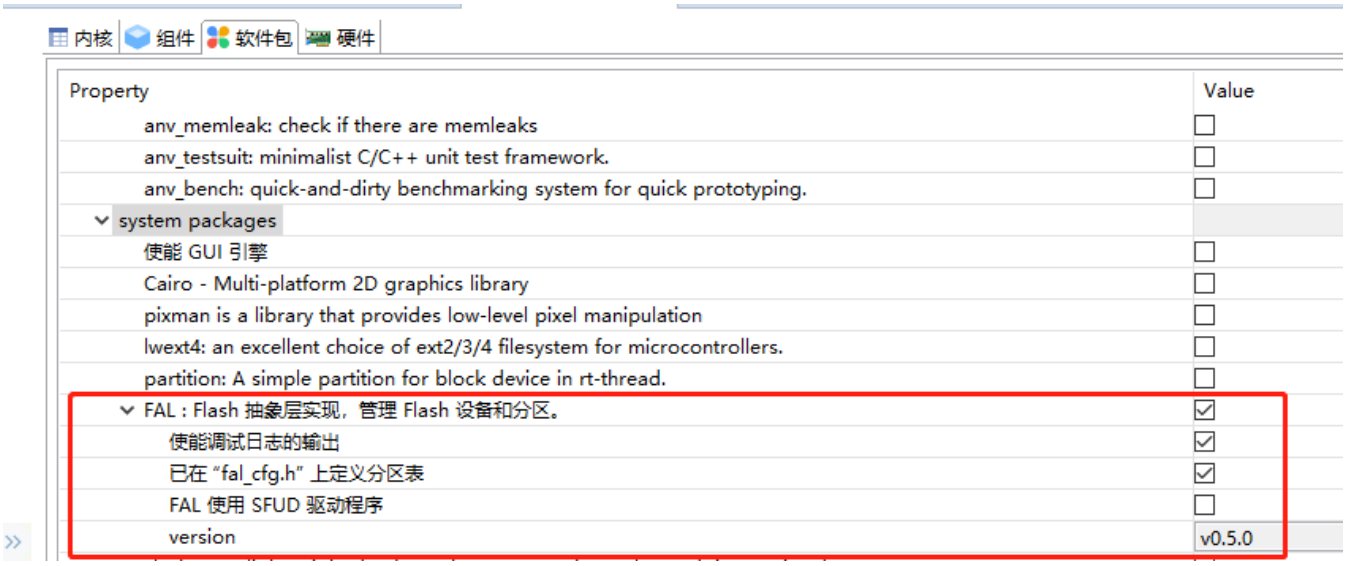
```
            uint16_t offset;

            while(writed < len)
            {
                sectorAssress = FlashGetSectorAddr(addr);
                if(FlashCheckSectorErase(sectorAssress, buff) == 1)
                {
                    os_spiflash_erase(sectorAssress);
                }
                offset = addr - sectorAssress;
                toWrite = ((len - writed) < (SECTOR_SIZE - offset) ? (len - writed) : (SECTOR_SIZE - offset));
                rt_memmove(buff + offset, data + writed, toWrite);
                if(FlashWritePage(sectorAssress, buff)==0)
                    break;

                writed += toWrite;
            }
        }
}

void FlashErase(uint32_t addr, size_t len)
{
        int i,j;
        if((addr >= DATA_ADDRESS_START) && ((addr + len) < DATA_ADDRESS_END))
        {
            uint32_t sectorAssress;
            uint16_t writed = 0;
            uint16_t toWrite;
            uint16_t offset;

            while(writed < len)
            {
                sectorAssress = FlashGetSectorAddr(addr);
                if(FlashCheckSectorErase(sectorAssress, buff) == 1)
                {
                    os_spiflash_erase(sectorAssress);
                }
                offset = addr - sectorAssress;
                toWrite = ((len - writed) < (SECTOR_SIZE - offset) ? (len - writed) : (SECTOR_SIZE - offset));
                rt_memset(buff + offset, 0xFF, toWrite);
                if(FlashWritePage(sectorAssress, buff)==0)
                    break;

                writed += toWrite;
            }
```

```
        }
}

void FlashRead(uint32_t addr, uint8_t *data, size_t len)
{
    os_spiflash_read(data, addr, len);
}
```

2.3. 编写主程序文件

2.3.1.FAL 移植

2.3.1.1 软件包添加 FAL



2.3.1.2.在 board 文件夹中新建 fal_cfg.h，并完成驱动接口。

[fal_cfg.h]

```
#ifndef _FAL_CFG_H_
#define _FAL_CFG_H_

#include <rtconfig.h>
#include <board.h>

/* ===================== Flash device Configuration ========================= */
extern const struct fal_flash_dev ab32vg1_onchip_flash;

/* flash device table */
#define FAL_FLASH_DEV_TABLE                                                    \
{                                                                              \
    &ab32vg1_onchip_flash,                                                     \
}

/* ====================== Partition Configuration ========================== */
#ifdef FAL_PART_HAS_TABLE_CFG
/* partition table */
#define FAL_PART_TABLE                                                                 \
{                                                                                      \
    {FAL_PART_MAGIC_WORD,        "flash1",     "AB32_onchip",        0,    512*1024, 0}, \
}
#endif /* FAL_PART_HAS_TABLE_CFG */

#endif /* _FAL_CFG_H_ */
```

2.3.1.3.新建 fal_flash_ab32vg1_port.c 文件并写入以下内容

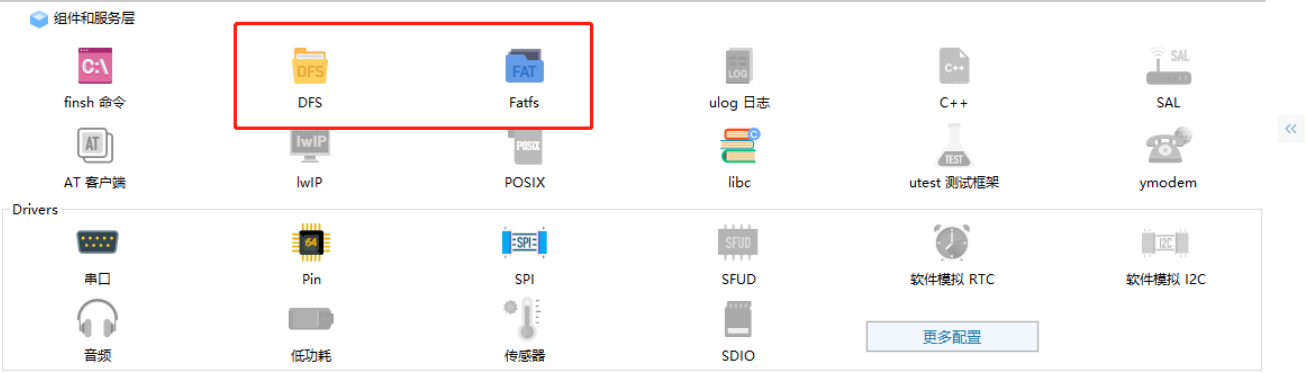
[fal_flash_ab32vg1_port.c]

```
#include <fal.h>
#include "flash.h"
#include "rtdbg.h"
#include "dfs_fs.h"
/**
 * Get the sector of a given address
 *
 * @param address flash address
 *
 * @return The sector of a given address
 */

static int init(void)
{
```

```
    /* do nothing now */
}

static int read(long offset, uint8_t* buf, size_t size)
{
    size_t i;
    uint32_t addr = ab32vg1_onchip_flash.addr + offset;
    FlashRead(addr, buf, size);
    return size;
}

static int write(long offset, const uint8_t* buf, size_t size)
{
    size_t i;
    uint32_t read_data;
    uint32_t addr = ab32vg1_onchip_flash.addr + offset;
    FlashWrite(addr, buf, size);
    return size;
}

static int erase(long offset, size_t size)
{

    FlashErase(ab32vg1_onchip_flash.addr + offset, size);
    return size;
}

const struct fal_flash_dev ab32vg1_onchip_flash =
{
    .name = "AB32_onchip",
    .addr = DATA_ADDRESS_START,
    .len = DATA_ADDRESS_END - DATA_ADDRESS_START,
    .blk_size = 512,
    .ops = {init, read, write, erase},
    .write_gran = 8
};
```

2.3.2.开启文件系统

配置打开 DFS 及 Fatfs



2.3.3.初始化 FAL 并挂载 Flash 至根目录 ‘/'

```
int ab32_flash_mount(void)
{
    struct rt_device *flash_dev;
    fal_init();
    flash_dev = fal_blk_device_create("flash1");
    if(flash_dev == NULL)
    {
        LOG_E("Failed to create flash device!");
        return -1;
    }
    retry:
    if (dfs_mount(flash_dev->parent.name, "/", "elm", 0, 0) == 0)
   {
        LOG_D("Filesystem initialized!");
        return 0;
   }
   else
   {
        if(dfs_mkfs("elm", "flash1") == 0)
        {
            LOG_D("mkfs ok!");
            goto retry;
        }
        LOG_E("Failed to initialize filesystem!");
        LOG_D("You should create a filesystem on the block device first!");
   }
    return -1;
}
INIT_APP_EXPORT(ab32_flash_mount);
```

3. 代码验证

3.1.FAL 初始化成功，并完成挂载

```
 \ | /
- RT -     Thread Operating System
 / | \     4.0.3 build Mar 29 2021
 2006 - 2021 Copyright by rt-thread team
[D/FAL] (fal_flash_init:61) Flash device |              AB32_onchip | addr:
0x00080000 | len: 0x00080000 | blk_size: 0x00000200 |initialized finish.
x1B[32;22m[I/FAL] ==================== FAL partition table ====================x1B
[0m
x1B[32;22m[I/FAL] | name   | flash_dev   |   offset   |    length  |x1B[0m
x1B[32;22m[I/FAL] ------------------------------------------------------------x1B
[0m
x1B[32;22m[I/FAL] | flash1 | AB32_onchip | 0x00000000 | 0x00080000 |x1B[0m
x1B[32;22m[I/FAL] ============================================================x1B
[0m
x1B[32;22m[I/FAL] RT-Thread Flash Abstraction Layer (V0.5.0) initialize success.x1B
[0m
x1B[32;22m[I/FAL] The FAL block device (flash1) created successfullyx1B[0m
msh />df
disk free: 493.0 KB [ 987 block, 512 bytes per block ]
msh />
```

3.2.文件操作

```
x1B[32;22m[I/FAL] The FAL block device (flash1) created successfullyx1B[0m
msh />cd /
msh />ls
Directory /:
msh />echo "Hello RT-Thread!" rt.txt
msh />ls
Directory /:
rt.txt               16
msh />
msh />cat rt.txt
Hello RT-Thread!
msh />mkdir AB32VG1
msh />ls
Directory /:
rt.txt               16
AB32VG1              <DIR>
msh />
```

写文件

读文件

新建文件夹

4. 章节总结

AB32VG1 内部有 1M byte flash，如此大的容量，在通用单片机上是比较少见的。但是目前开放出来的资料较少，深度定制开发比较困难。不过在 RT-Thread 上提供了一些支持，方便后续开发。

十二、中科蓝讯 AB32VG1 上的 SD 实践

1. 前言说明

1.1 本章内容

本章通过 RT-Thread Studio 配置 AB32VG1 使用 SDIO 驱动 SD 卡

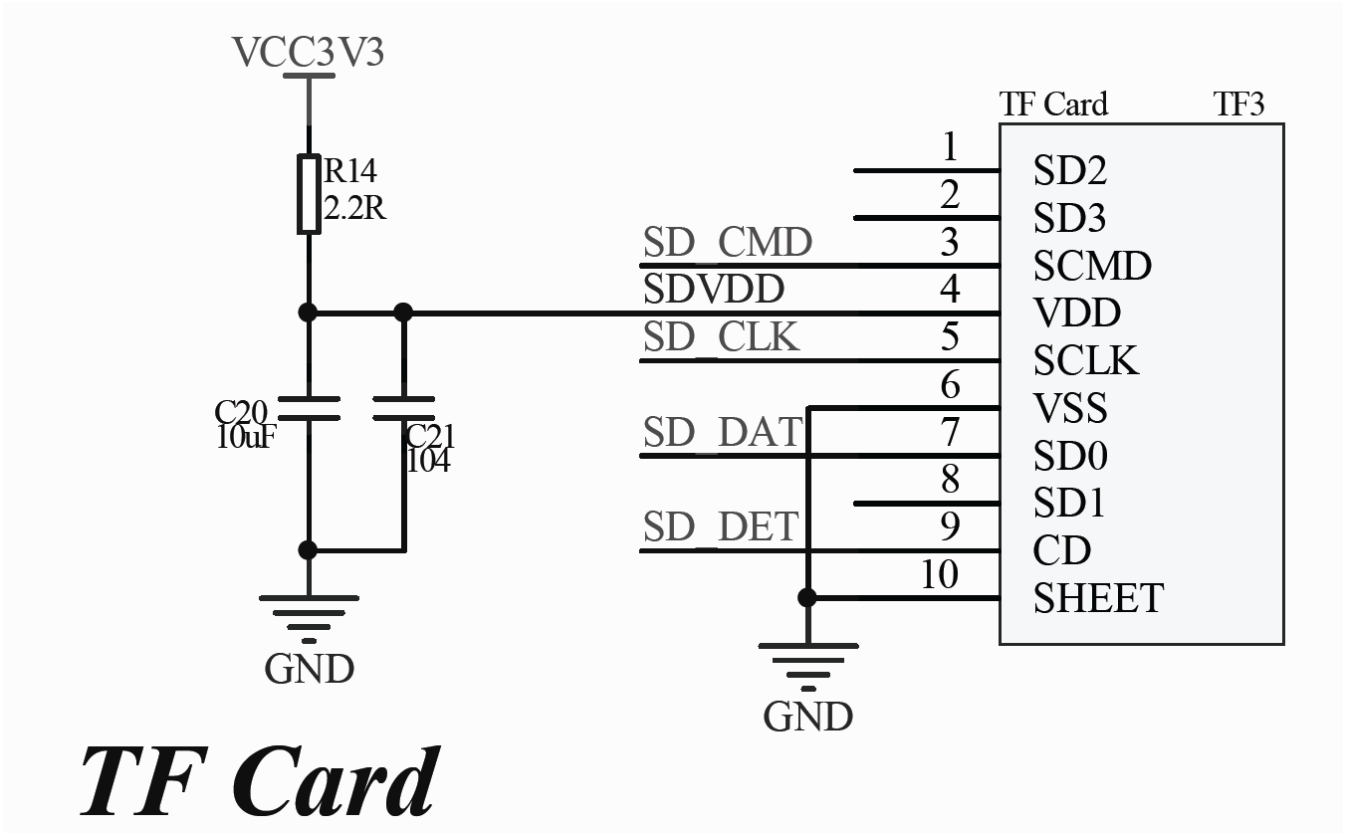
1.2 模块介绍

根据说明书，开发板上有一路 TF CARD 接口

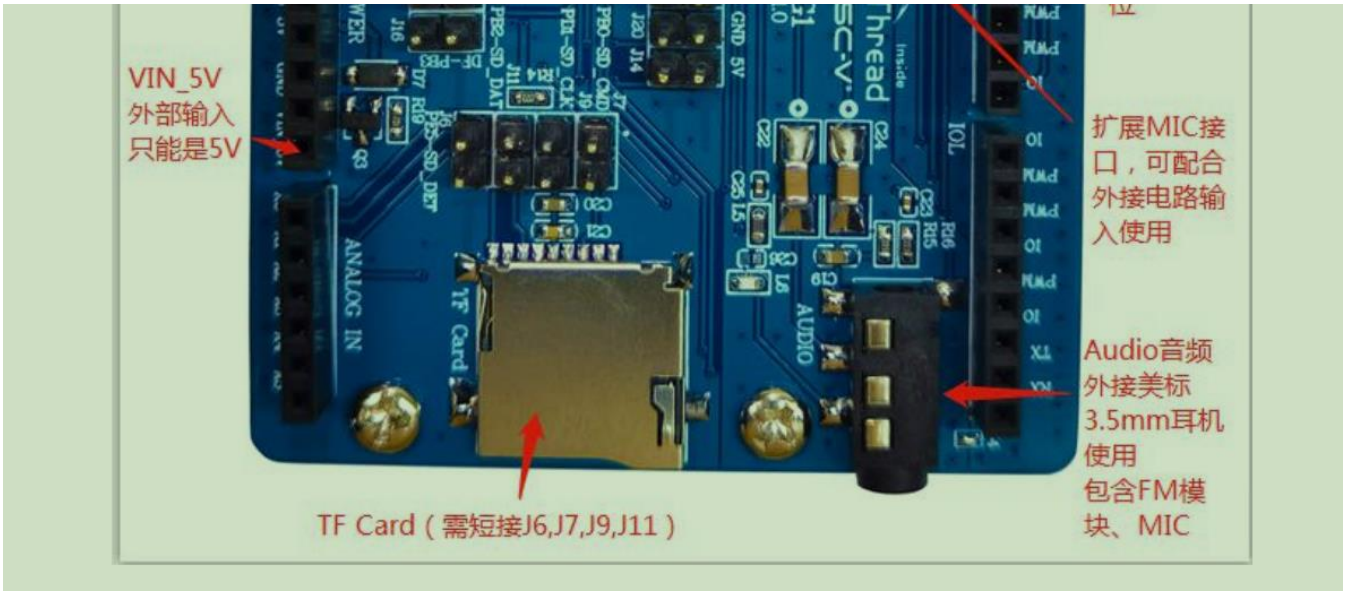
板上资源：

- **CPU：AB5301A；（LQFP48 封装，主频 120M，片上集成 RAM 192K, flash 8 Mbit，ADC PWM，USB，UART，IIC 等资源）**
- 搭载蓝牙模块
- 搭载 **FM** 模块
- 一路 **TF Card** 接口
- 一路 **USB** 接口
- 一路 **IIC** 接口
- 一路音频接口**(**美标 **CTIA)**
- 六路 **ADC** 输入引脚端子引出
- 六路 **PWM** 输出引脚端子引出
- 一个全彩 **LED** 灯模块，一个电源指示灯，三个烧录指示灯
- 一个 **IRDA**（红外接收端口）
- 一个 Reset 按键，三个功能按键(通用版为两个功能按键)
- 板子规格尺寸：**6cm*9cm**
- I/O 口通过 2.54MM 标准间距引出，同时兼容 **Arduino Uno** 扩展接口，方便二次开发

原理图如下：



开发板实物位置：



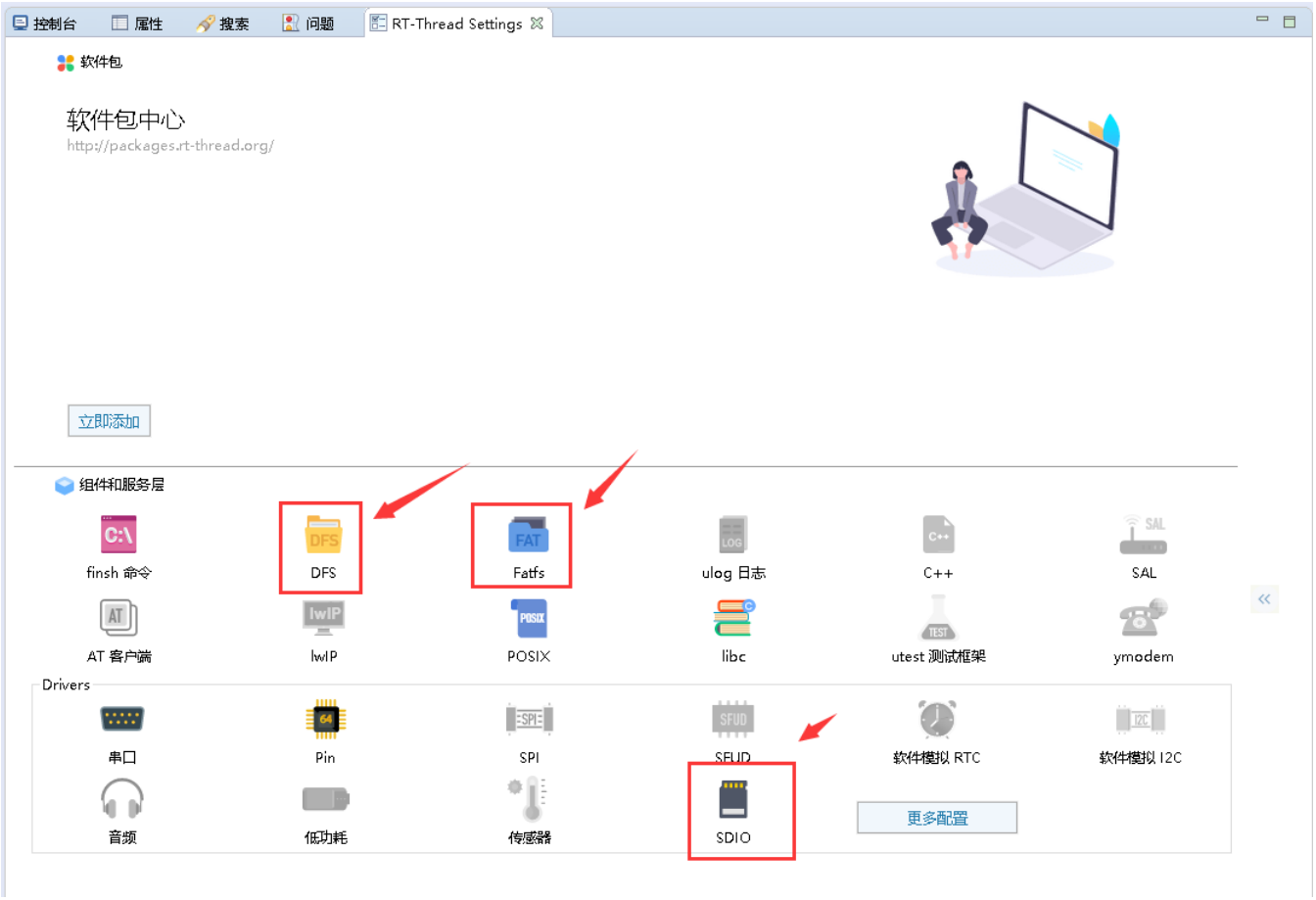
2. 步骤说明

2.1 创建工程

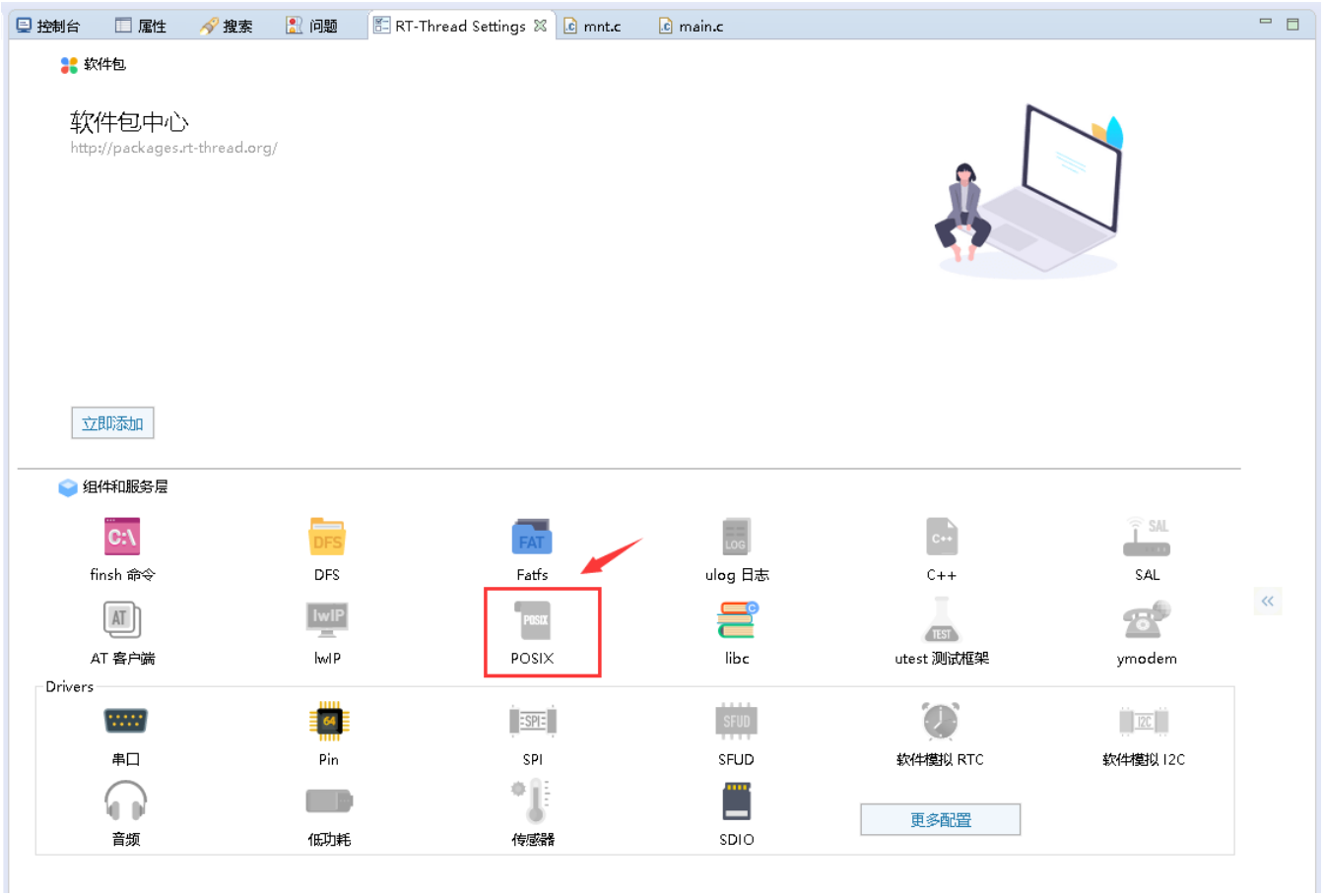
点击 文件-> 新建-> RT-Thread 项目

2.2 组件配置

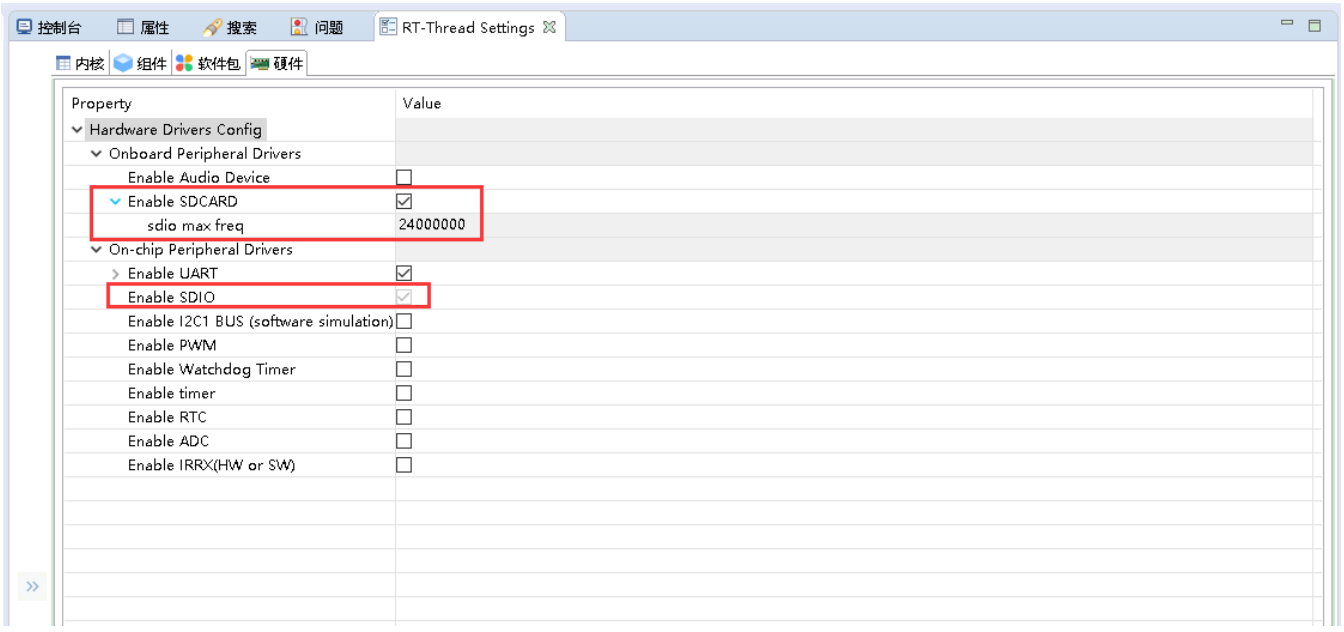
双击 RT-Thread Settings 文件，打开 RT-Thread 项目配置界面，启用 DFS, Fatfs 和 SDIO 驱动。


控制台
属性
搜索
问题
RT-Thread Settings
软件包
软件包中心
http://packages.rt-thread.org/
立即添加
组件和服务层
finsh 命令
DFS
Fatfs
ulog 日志
C++
SAL
AT 客户端
lwIP
POSIX
libc
utest 测试框架
ymodem
Drivers
串口
Pin
SPI
SFUD
软件模拟 RTC
软件模拟 I2C
音频
低功耗
传感器
SDIO
更多配置

禁用 POSIX 组件

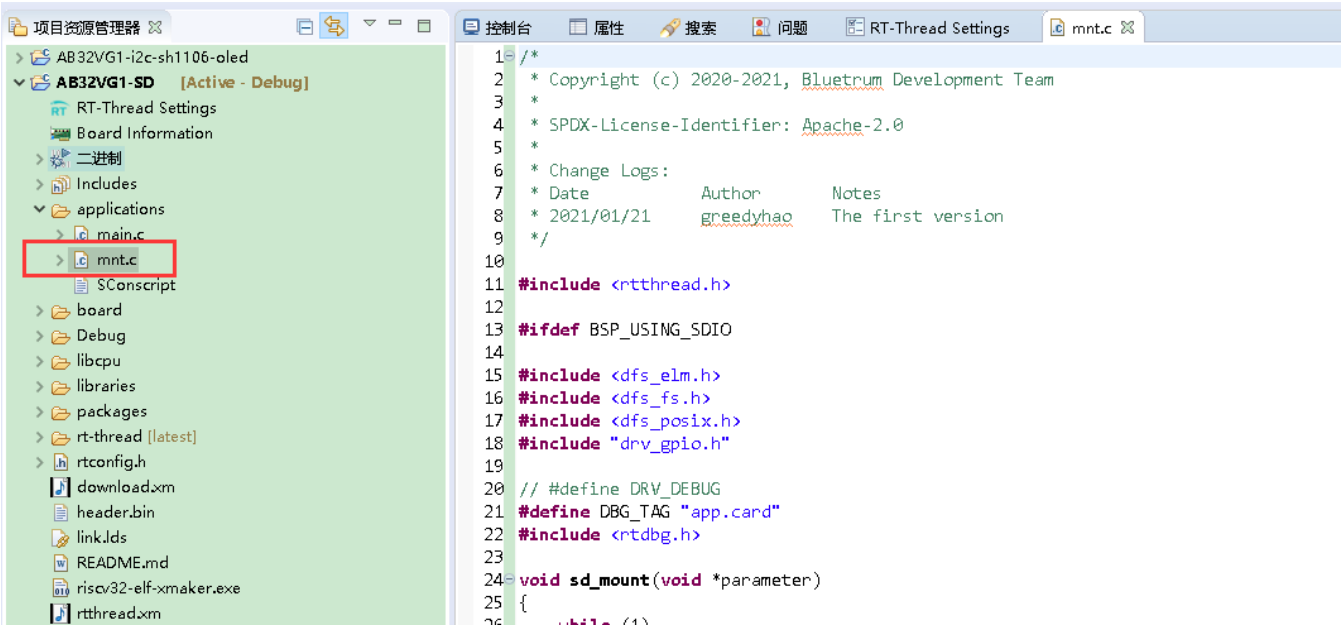


点击 更多设置，选择硬件选项，勾选 Onboard Peripheral Drivers 下的 Enable SDCARD 选项，使能 SDCARD，勾选后 On-chip Peripheral Drivers 下的 Enable SDIO 会默认勾选的。然后保存即可。



2.3 代码检验

挂载文件 elm-fat 系统，工程配置好后， 在 applications 目录下有个 mnt.c 文件，这个文件就是把 SD 卡挂载到文件 elm-fat 系统， 无须重新写挂载！这个 mnt.c 使用了线程是实现挂载，SD 初始化的时候存在延时，这也就是使用线程去实现挂载的原因吧！



测试代码， 参考了 RT-Thread 文档中心-虚拟文件系统 （https://www.rt-thread.org/document/site/#/rt-thread-version/rt-thread-standard/programming-manual/filesystem/filesystem.md ）

```
static void readwrite_sample(void)
{
    int fd, size;
    char s[] = "RT-Thread Programmer!", buffer[80];

    rt_kprintf("Write string %s to test.txt.\n", s);

    /* 以创建和读写模式打开 /text.txt 文件，如果该文件不存在则创建该文件 */
    fd = open("/text.txt", O_WRONLY | O_CREAT);
    if (fd>= 0)
    {
        write(fd, s, sizeof(s));
        close(fd);
        rt_kprintf("Write done.\n");
    }
```

```
        /* 以只读模式打开 /text.txt 文件 */
    fd = open("/text.txt", O_RDONLY);
    if (fd>= 0)
    {
        size = read(fd, buffer, sizeof(buffer));
        close(fd);
        rt_kprintf("Read from file test.txt : %s \n", buffer);
        if (size < 0)
            return ;
    }
  }
/* 导出到 msh 命令列表中 */
MSH_CMD_EXPORT(readwrite_sample, readwrite sample);


static void rename_sample(void)
{
    rt_kprintf("%s => %s", "/text.txt", "/text1.txt");

    if (rename("/text.txt", "/text1.txt") < 0)
        rt_kprintf("[error!]\n");
    else
        rt_kprintf("[ok!]\n");
}
/* 导出到 msh 命令列表中 */
MSH_CMD_EXPORT(rename_sample, rename sample);


static void stat_sample(void)
{
    int ret;
     struct stat buf;
     ret = stat("/text.txt", &buf);
    if(ret == 0)
    rt_kprintf("text.txt file size = %d\n", buf.st_size);
    else
    rt_kprintf("text.txt file not fonud\n");
}
/* 导出到 msh 命令列表中 */
MSH_CMD_EXPORT(stat_sample, show text.txt stat sample);


static void mkdir_sample(void)
{
    int ret;
```

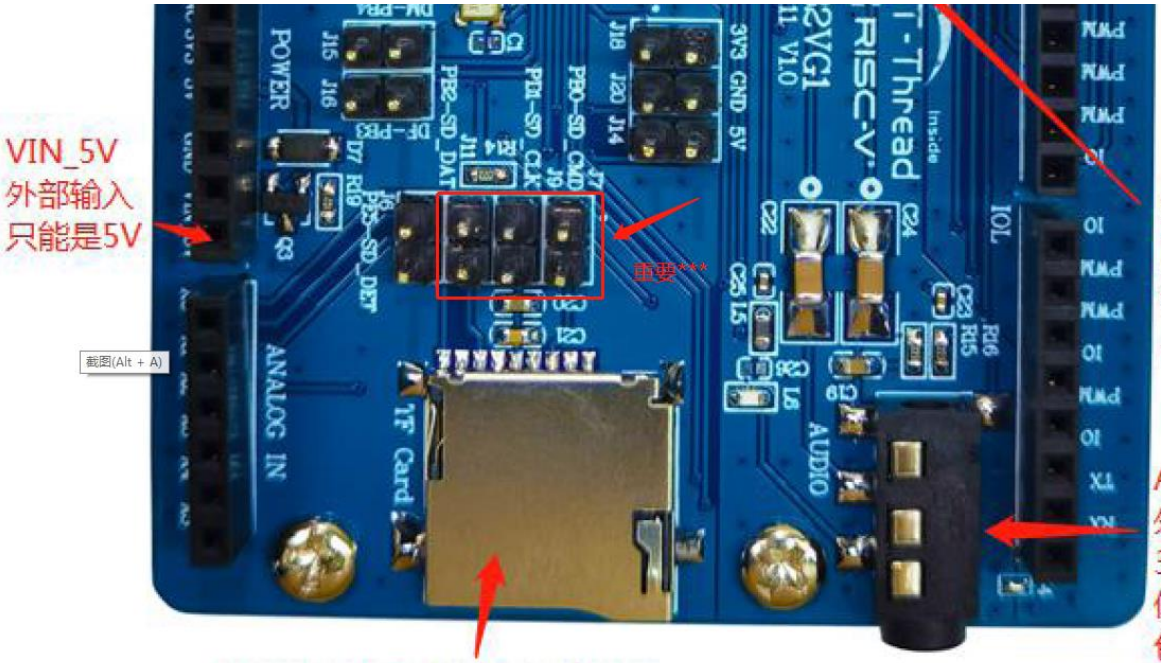
```
    /* 创建目录 */
    ret = mkdir("/dir_test", 0x777);
    if (ret < 0)
    {
        /* 创建目录失败 */
        rt_kprintf("dir error!\n");
    }
    else
    {
        /* 创建目录成功 */
        rt_kprintf("mkdir ok!\n");
    }
}
/* 导出到 msh 命令列表中 */
MSH_CMD_EXPORT(mkdir_sample, mkdir sample);
```

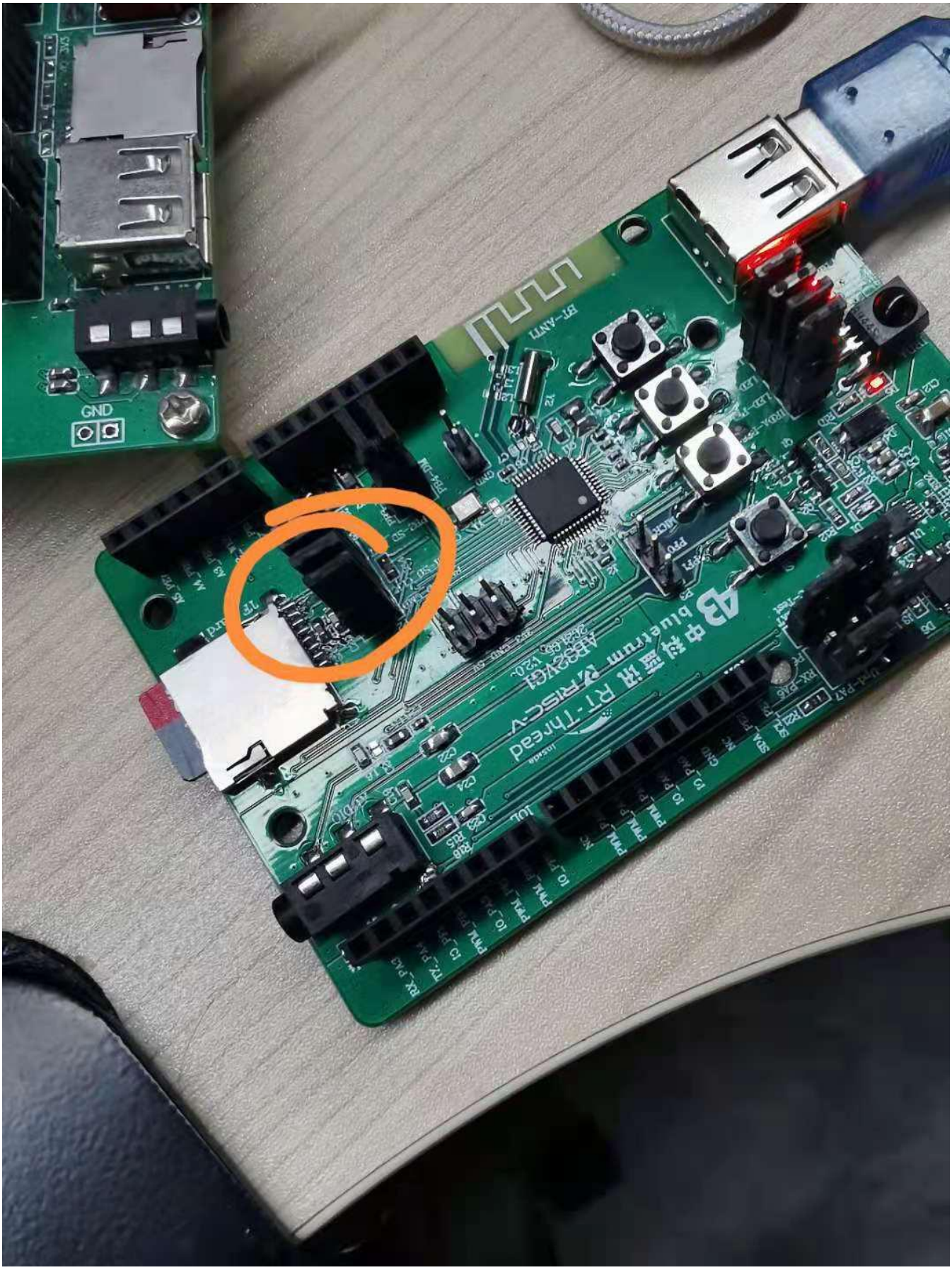
3. Downloaded 下载器使用

请参考 《中科蓝讯 AB32VG1 上的 I2C 实践》一文

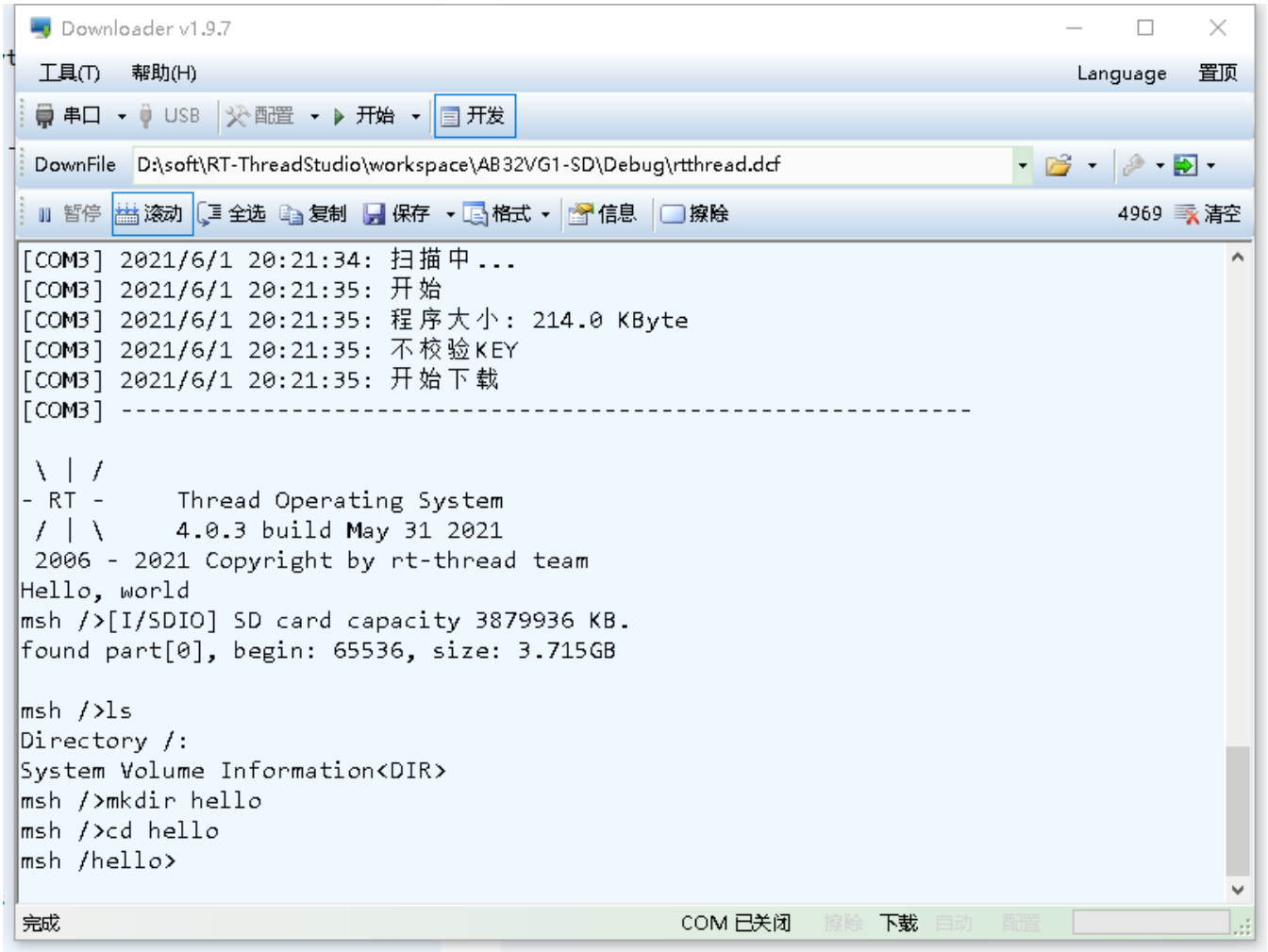
4. 演示

注意根据原理图要跳针


及烧录复位
VIN_5V
外部输入
只能是5V
扩展MIC接口，可配合外接电路输入使用
重要***
Audio音频
外接美标
3.5mm耳机
使用
包含FM模块、MIC
TF Card（需短接J6,J7,J9,J11）
截图(Alt + A)



试验中用的是 4GB TF 卡



可以通过 list_device 命令可以看到 sd0 的使用，可以通过上面的写好代码，已经使用想要的命令 readwrite_sample，rename_sample，mkdir_sample，stat_sample， 可以实现读写，改名，新建文件夹，SD 卡里面的文件，查询文件相关信息。

5. 章节总结

介绍了 SDIO 的使用。主要是挂载文件系统的时候，由于 SD 卡初始化会延时，需要延时挂载，或者想项目中使用独立线程挂载，挂载成功后退出线程。有个缺点，fatfs 文件系统无法读取大文件。

十三、中科蓝讯 AB32VG1 上的 IRDA

实践

1. 前言说明

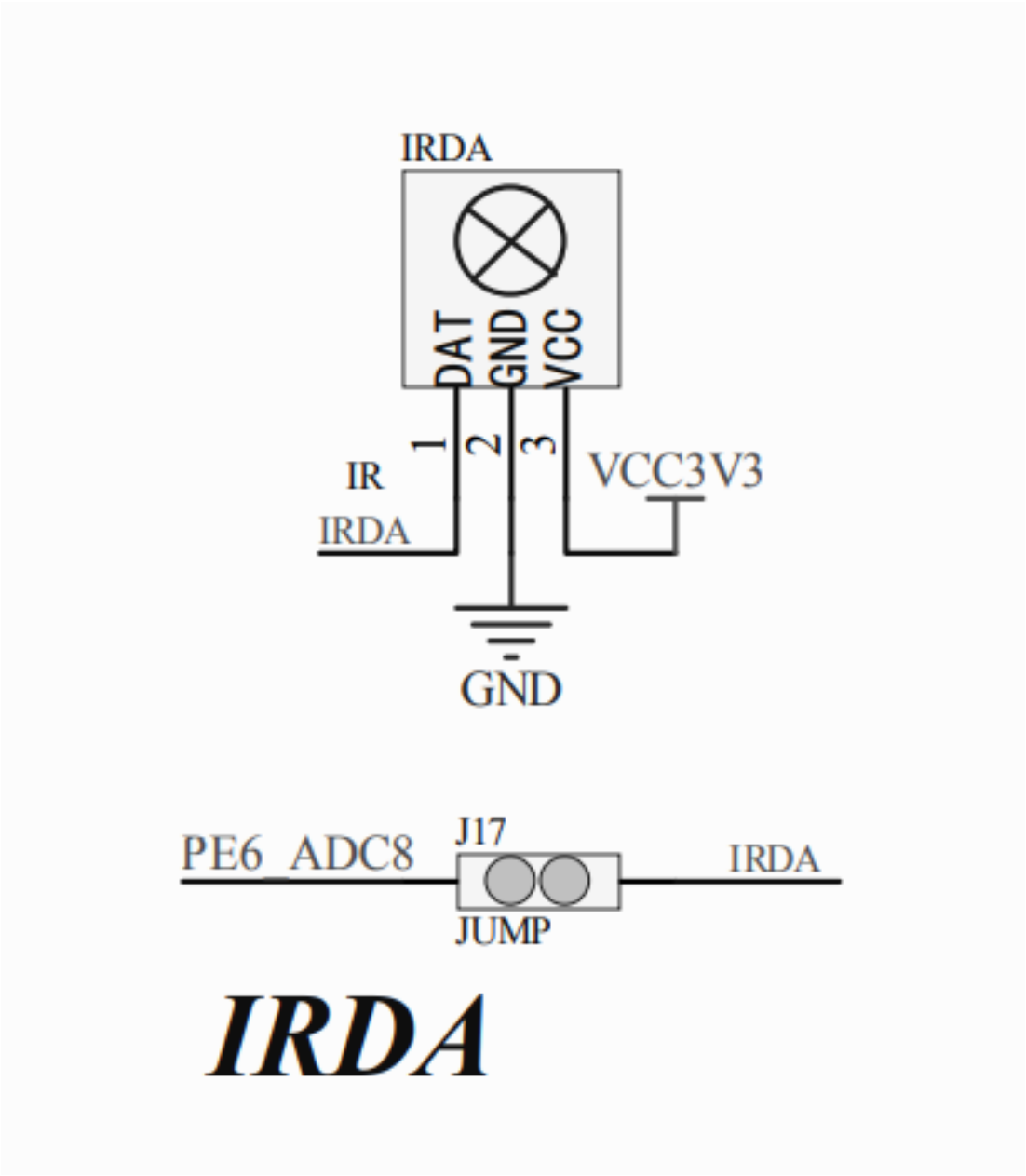
IRDA 模块

1.1 本章内容

本章描述在 linux 环境下开发测试 IRDA 模块的方法.

1.2 模块介绍

AB32VG1 有内置的红外接收控制器.根据原理图,红外接收模块关联的管脚是 PE6 脚.



测试红外时,需要将 J17 跳帽接上(默认都是接上的).

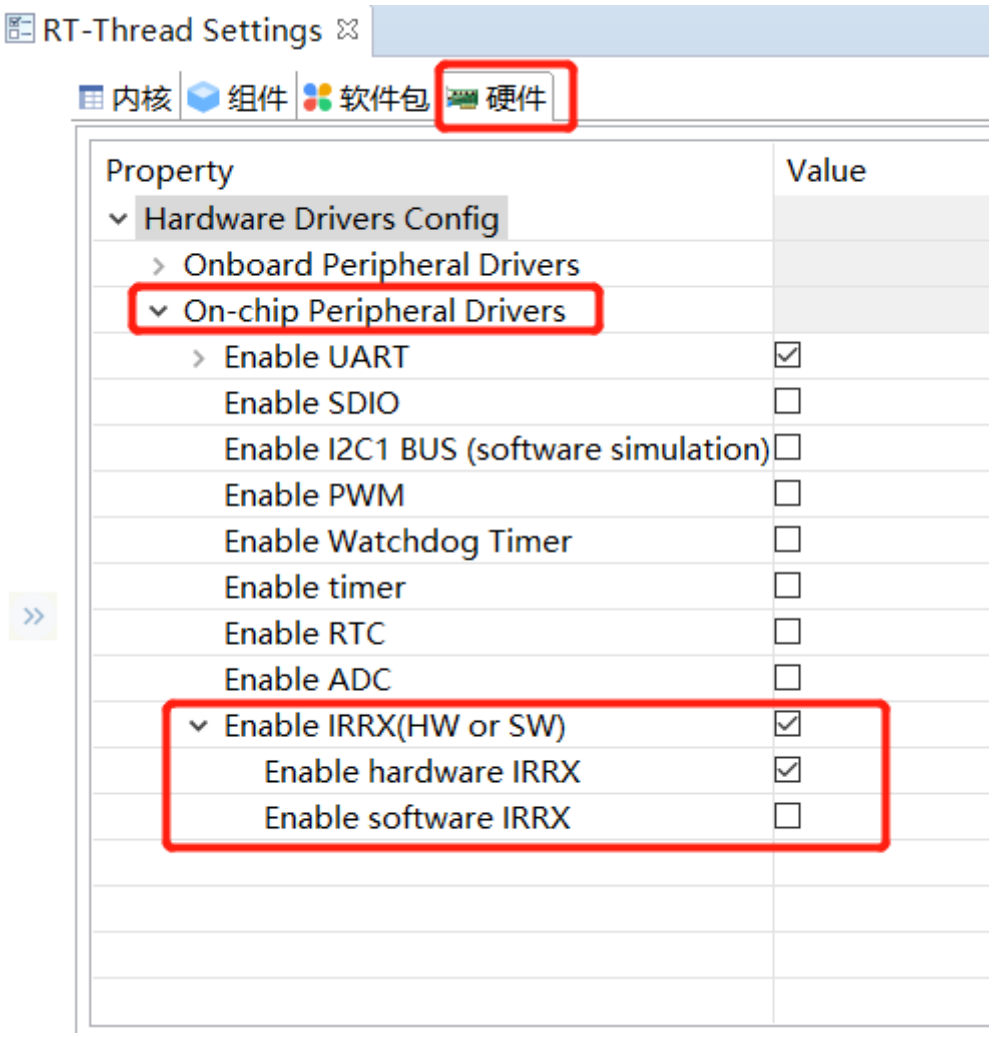
2. 步骤说明

2.1 打开 IRDA 模块

在 RT-Thread Studio 左侧资源管理器中选择 RT-Thread Settings，右侧点更多配置。

硬件--->Hardware Drivers Config--->On-chip Peripheral Drivers--->Enable IRRX(HW or SW)

可以选择 Enable hardware IRRX 或 Enable software IRRX。



2.2 组件配置以及代码分析

以硬件解码为例对驱动文件 drv_ir.c 文件进行解读.

```
static void _irrx_hw_init(void)
{
    /* 初始化红外管脚映射到 PE6 */
    GPIOEDE |= BIT(6);
    GPIOEPU |= BIT(6);
    GPIOEDIR |= BIT(6);
    FUNCMCON2 |= 0xf << 20;
```

```
    FUNCMCON2 |= (7 << 20);
    rt_memset(&_irrx, 0, sizeof(_irrx));

/* 根据 32K 或者 1M 不同的始终频率初始化配置 */
    IRRXERR0 = (RPTERR_CNT << 16) | DATERR_CNT;
    IRRXERR1 = (TOPR_CNT << 20) | (ONEERR_CNT << 10) | ZEROERR_CNT;

/* 如果选择了 32K 的时钟频率,需要额外配置时钟相关的寄存器以及开启红外对应寄存器的管脚 */
#if IR32KSEL_EN
    CLKCON1 &= ~BIT(5);
    CLKCON1 |= BIT(4);
    IRRXCON |= BIT(3);
#endif

/* 加载红外管脚中断处理函数 */
    rt_hw_interrupt_install(IRQ_IRRX_VECTOR, irrx_isr, RT_NULL, "irrx_isr");
 /* 使能红外以及红外中断 */
    IRRXCON = 0x03;
}
```

硬件红外驱动的处理函数相对来说简单些,就是从寄存器中读取红外的数据

```
static void irrx_isr(int vector, void *param)
{
    rt_interrupt_enter();
```

```
    /* 如果有接收到有效的红外数据 */
    if (IRRXCON & BIT(16)) {
    /* 清除接收到数据的标志 */
        IRRXCPND = BIT(16);
        /* 从红外控制器对应的寄存器中读取接收到的数据 */
        _irrx.addr = (uint16_t)IRRXDAT;
        _irrx.cmd = (uint16_t)(IRRXDAT >> 16);
        /* 标记红外数据长度是 32 bit */
        _irrx.cnt = 32;
    }

    /* 如果检测到红外按键释放了,清零接收数据的长度 */
    if (IRRXCON & BIT(17)) {
        IRRXCPND = BIT(17);
        _irrx.cnt = 0;
    }

    rt_interrupt_leave();
}
```

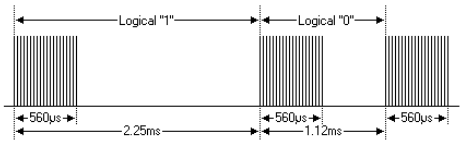
聊完了硬件解码的红外驱动,再聊下软件解码的红外驱动,初始化和中断处理函数都已经不同了,首先看初始化部分:

```
static void _irrx_hw_init(void)
{
```

```
    GPIOEDE |= BIT(6);
    GPIOEPU |= BIT(6);
    GPIOEDIR |= BIT(6);
    /* 将 PE6 设置为输入捕获模式 */
    FUNCMCON2 |= 0xf << 4;
    FUNCMCON2 |= (7 << 4);
    rt_memset(&_irrx, 0, sizeof(_irrx));
    /* 初始化定时器 3 */
    timer3_init();
}
static void timer3_init(void)
{
    /* 关联红外管脚的中断处理函数 */
    rt_hw_interrupt_install(IRQ_IRRX_VECTOR, irrx_isr, RT_NULL, "irrx_isr");
    TMR3CNT  = 0;
    /* 设置溢出的周期是 110ms */
    TMR3PR   = TMR3_RCLK*110 - 1;
    /* 配置定时器 3 工作在输入捕获模式,统计输入的上升沿以及下降边沿时捕获 */
    TMR3CON  = BIT(8) | BIT(7) | BIT(5) | BIT(2) | BIT(1) | BIT(0);
}
```

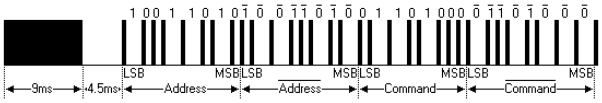
上面为什么设计定时器 3 的溢出周期为 110 ms 呢?这就需要参考 IR 的协议了,常用的是 NEC 格式的.

Modulation

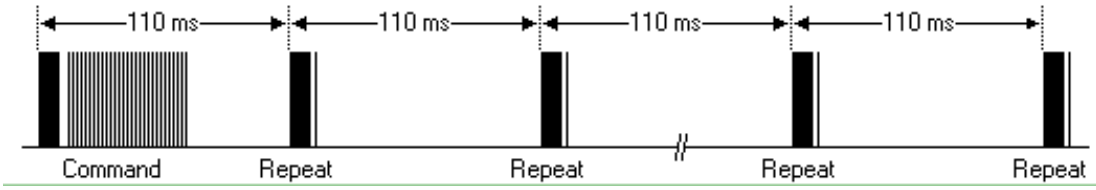


The NEC protocol uses pulse distance encoding of the bits. Each pulse is a 560µs long 38kHz carrier burst (about 21 cycles). A logical "1" takes 2.25ms to transmit, while a logical "0" is only half of that, being 1.125ms. The recommended carrier duty-cycle is 1/4 or 1/3.

Protocol



The picture above shows a typical pulse train of the NEC protocol. With this protocol the LSB is transmitted first. In this case Address $59 and Command $16 is transmitted. A message is started by a 9ms AGC burst, which was used to set the gain of the earlier IR receivers. This AGC burst is then followed by a 4.5ms space, which is then followed by the Address and Command. Address and Command are transmitted twice. The second time all bits are inverted and can be used for verification of the received message. The total transmission time is constant because every bit is repeated with its inverted length. If you're not interested in this reliability you can ignore the inverted values, or you can expand the Address and Command to 16 bits each!
Keep in mind that one extra 560µs burst has to follow at the end of the message in order to be able to determine the value of the last bit.



根据 NEC 协议的标准,每一帧数据的周期为 110 ms.并且上图也描述了逻辑 0 和 逻辑 1 对应的波形.其中逻辑 1 对应的两个高电平的时间间隔为 2.25ms.同样地逻辑 0 对应的两个高电平的时间间隔为 1.12ms.

软解码情况下中断处理函数为:

```
static void irrx_isr(int vector, void *param)
{
    rt_uint32_t tmrcnt;

/* 捕获到下降沿,开始触发统计 */
    if (TMR3CON & BIT(17)) {
        /* 定时器 3 捕获中断,计算当前下降沿最近的高电平的持续时间 */
        TMR3CNT  = TMR3CNT - TMR3CPT;
        /* 记录捕获的相邻的两个上升沿的计数周期 */
```

```
        tmrcnt = TMR3CPT;
        /* 清除捕获的标志 */
        TMR3CPND = BIT(17);
        /* 计算最近的两个上升沿的周期单位为 ms */
        tmrcnt /= TMR3_RCLK;
    } else if (TMR3CON & BIT(16)){
        /* 如果发生了溢出,那么清零溢出中断,标记计数值为 110 */
        TMR3CPND = BIT(16);
        tmrcnt = 110;
    } else {
        return;
    }

    /* 如果已经记录了 32 个 数据位,说明已经正常解析了红外数据 */
    if (_irrx.cnt == 32) {
        /* 如果最近的两个上升沿的周期时间满足条件,那么表示是重复帧的开始 */
        if ((tmrcnt >= 10) && (tmrcnt <= 12)) {
            /* 清零重复帧的次数 */
            _irrx.rpt_cnt = 0;
        } else {
            _irrx.rpt_cnt += tmrcnt;
            /*  如果计数值超过 108,表示已经超时,说明按键已经释放 */
            if (_irrx.rpt_cnt > 108) {
                _irrx.rpt_cnt = 0;
                _irrx.cnt = 0;              //ir key release
```

```
            }
        }
        return;
        /* 如果周期比 7ms 长,那么表示是红外数据的开始帧 */
    } else if ((tmrcnt > 7) || (tmrcnt == 0)) {
        _irrx.rpt_cnt = 0;
        _irrx.cnt = 0;                          //ir key message started
        return;
    }

    /* 红外的数据低位在前 */
    _irrx.cmd >>= 1;
    _irrx.cnt++;
    /* 如果最近的两个上升沿的周期计数为 2,说明接收周期 >= 2ms ,对应的红外的逻辑数据是 1 */
    if (tmrcnt == 2) {
        _irrx.cmd |= 0x8000;
    }

    /* 如果接收到的上升沿的数量已经为 16 说明已经获取到开始的 16 bit 的地址信息 */
    if (_irrx.cnt == 16) {
        _irrx.addr = _irrx.cmd;             //save address data
    } else if (_irrx.cnt == 32) {
        if ((rt_uint8_t)_irrx.cmd > 96) {
            _irrx.cmd = NO_KEY;
```

```
            }
        }
}
```

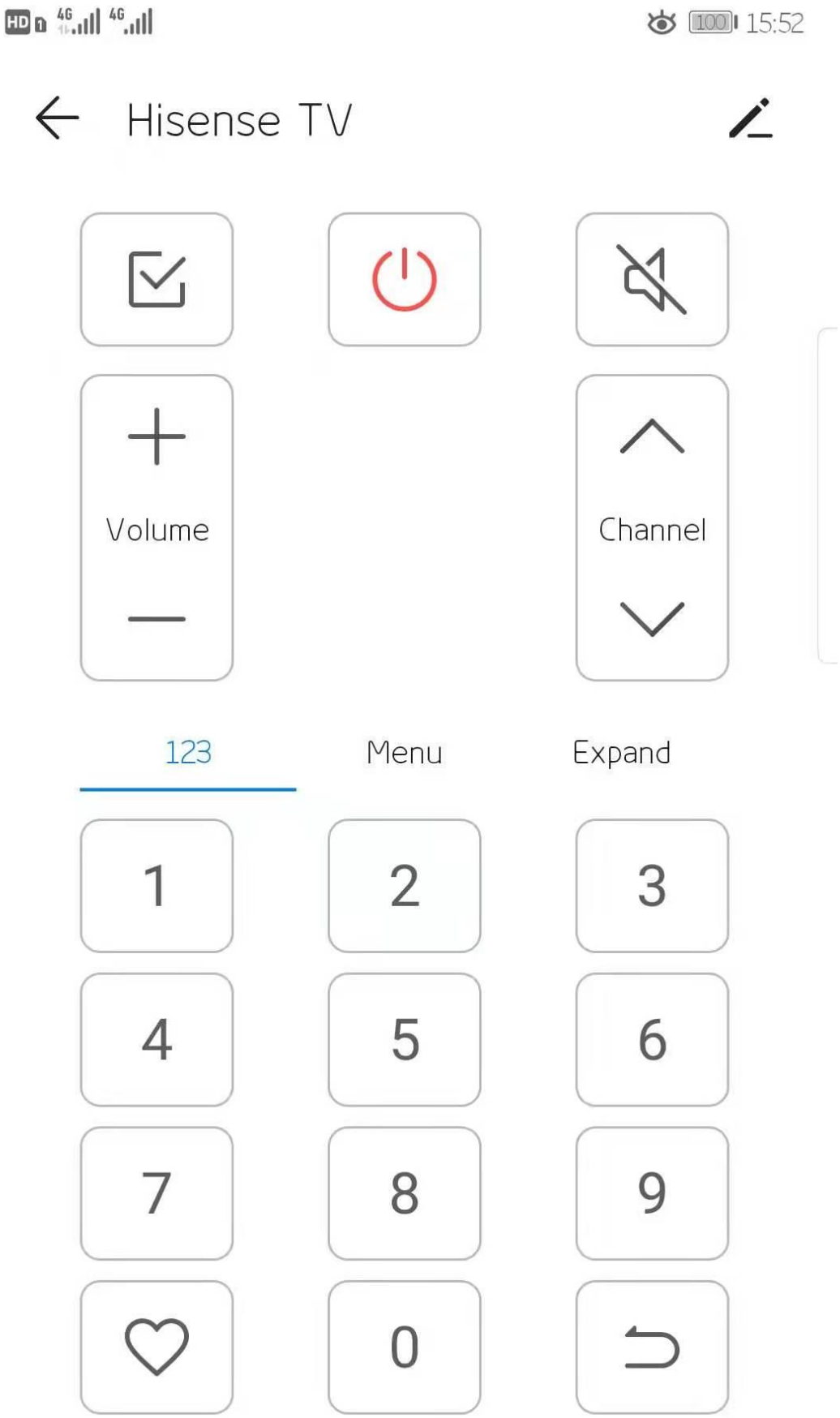
2.3 代码检验

了解了红外驱动的部分代码,那就需要写一段代码来测试了.实现将获取的键值通过终端打印出来.简单的方法就是直接在中断处理函数中将键值打印出来。

在测试过程中,通过添加一个 test_irda 命令来测试红外驱动，具体代码为,(**备注:**需要将改函数放在文件 drv_irrx.c 中):

```
static void test_irda(void)
{
    while (1)
    {
        if (_irrx.addr != 0 || _irrx.cmd != 0 )
        {
            rt_kprintf("cmd=%hx addr=%hx\n", _irrx.addr, _irrx.cmd);
            rt_memset(&_irrx, 0, sizeof(_irrx));
        }
        rt_thread_mdelay(100);
    }
}
MSH_CMD_EXPORT(test_irda, test irda sample);
```

通过将重新生成的 rtthread.bin 复制到 windows 然后添加头部以及加密处理,烧录到 AB32VG1 中,使用手机上的红外遥控器（海信 TV）。


15:52
Hisense TV
Volume
Channel
123
Menu
Expand
1
2
3
4
5
6
7
8
9
0

测试结果为在 msh 下输入命令 test_irda 之后每按下一个按键可以显示对应的键值:

```
[COM4] 2021/6/8 11:51:09: 开始下载
[COM4] ------------------------------------------------------------
[D/drv.irrx] irrx init success
[I/I2C] I2C bus [i2c1] registered

 \ | /
- RT -     Thread Operating System
 / | \     4.0.3 build Jun  2 2021
 2006 - 2021 Copyright by rt-thread team
Hello, world
msh />
test_irda
msh />test_irda
cmd=1 addr=fe01
cmd=1 addr=fd02
cmd=1 addr=fc03
cmd=1 addr=f906
cmd=1 addr=fa05
cmd=1 addr=fb04
cmd=1 addr=f708
```

3. 章节总结

本章节重点描述了在 RT-Thread Studio 中配置红外以及简单测试红外接收键值的方法。

介绍了 NEC 的红外协议特别是如何通过定时器中断实现软解码处理红外信号的。

十四、中科蓝讯 AB32VG1 上的 Audio 实践

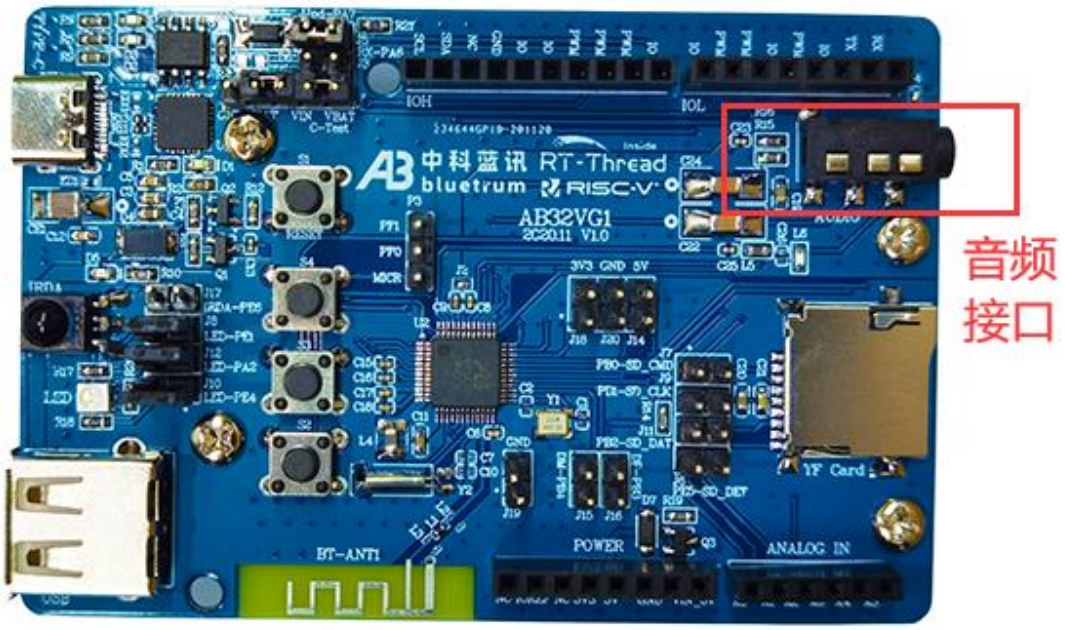
1.前言说明

1.1 本章内容

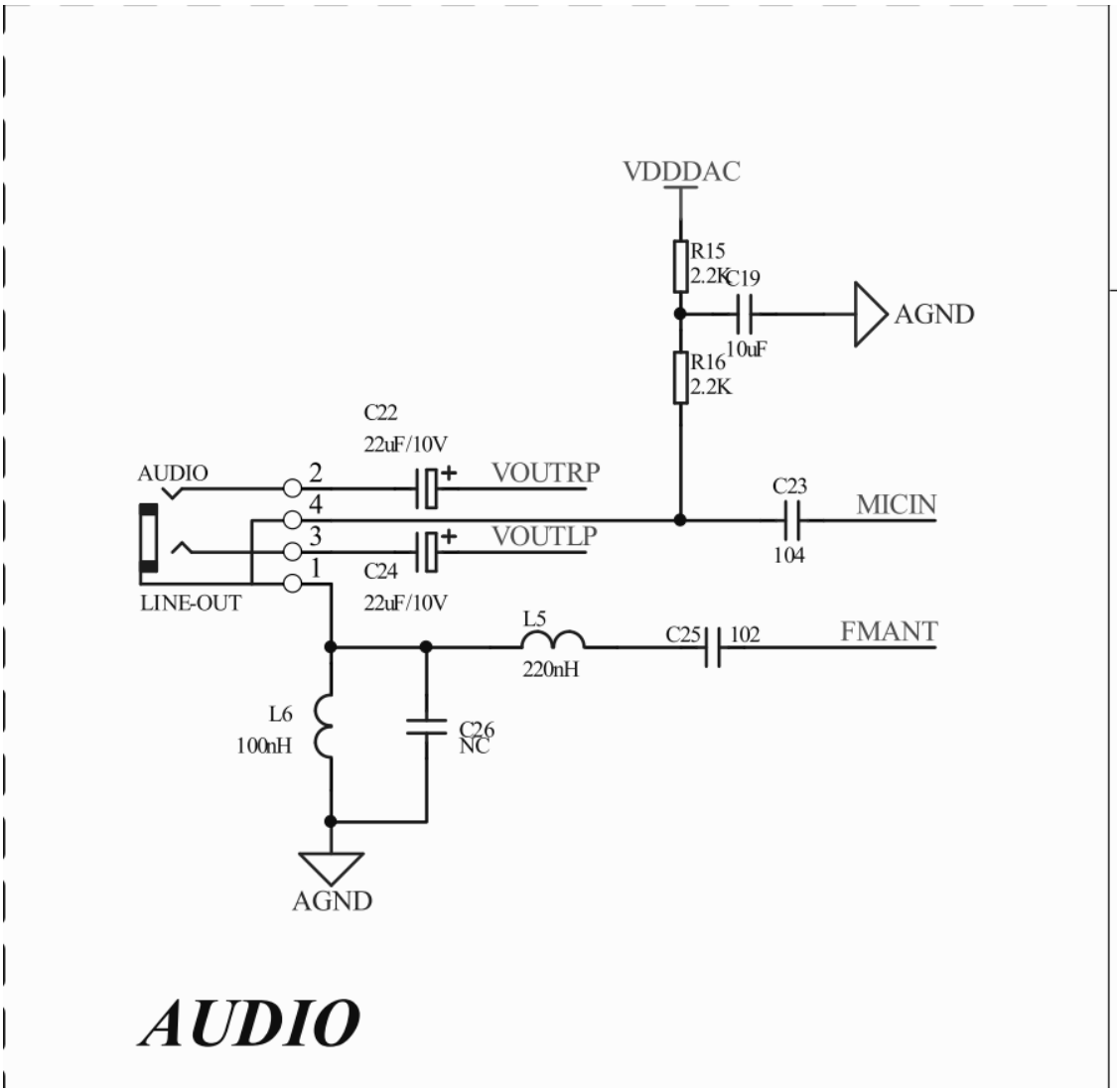
本次测评 AB32VG1 音频输出，从内部 Flash 读取 Wav 音频进行播放，按键控制音频的切换

1.2 模块介绍

开发板板载一个 3.5mm 的音频接口，接口是直连芯片的 DAC 输出和一个麦克风输入采样和收音机天线



原理图如下



芯片音频特性：

9. 具有 16 位立体声 DAC 和两个通道 16 位 ADC 的音频编解码器。
10. 支持灵活的音频 EQ 调节，支持 8、11.025、12、16、22.05、32、44.1 和 48khz 的采样率。
11. 4 通道立体声模拟 MUX。
12. 双通道 MIC 放大器输入。
13. 具有 90dB 信噪比的高性能立体声 ADC。

14. 高性能立体声音频 DAC，带 95dBSNR，带耳机放大器输出。

1.3 开发软件

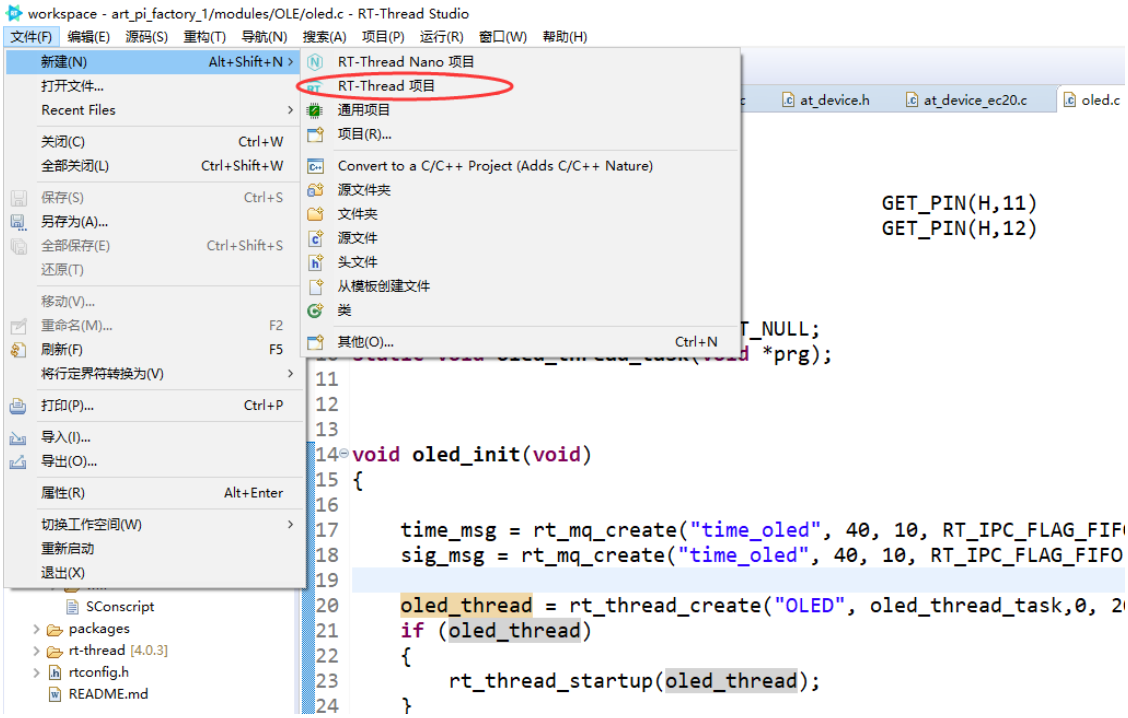
编译平台：**RT-Thread Studio：** 安装链接

下载平台：**Downloader:** 安装链接

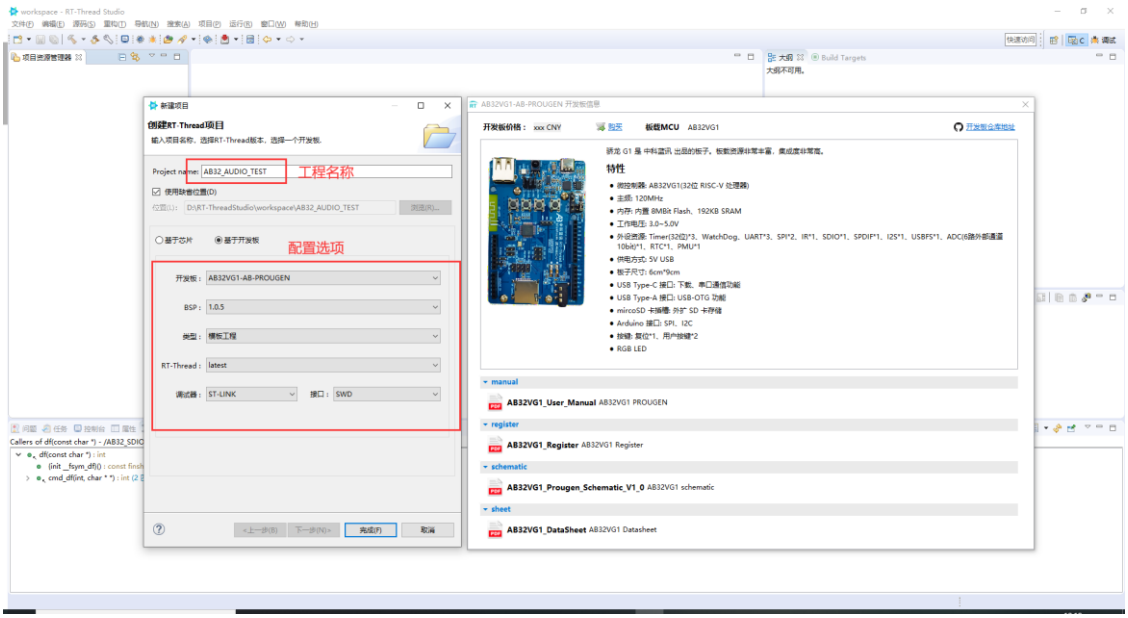
2.步骤说明

2.1 新建工程

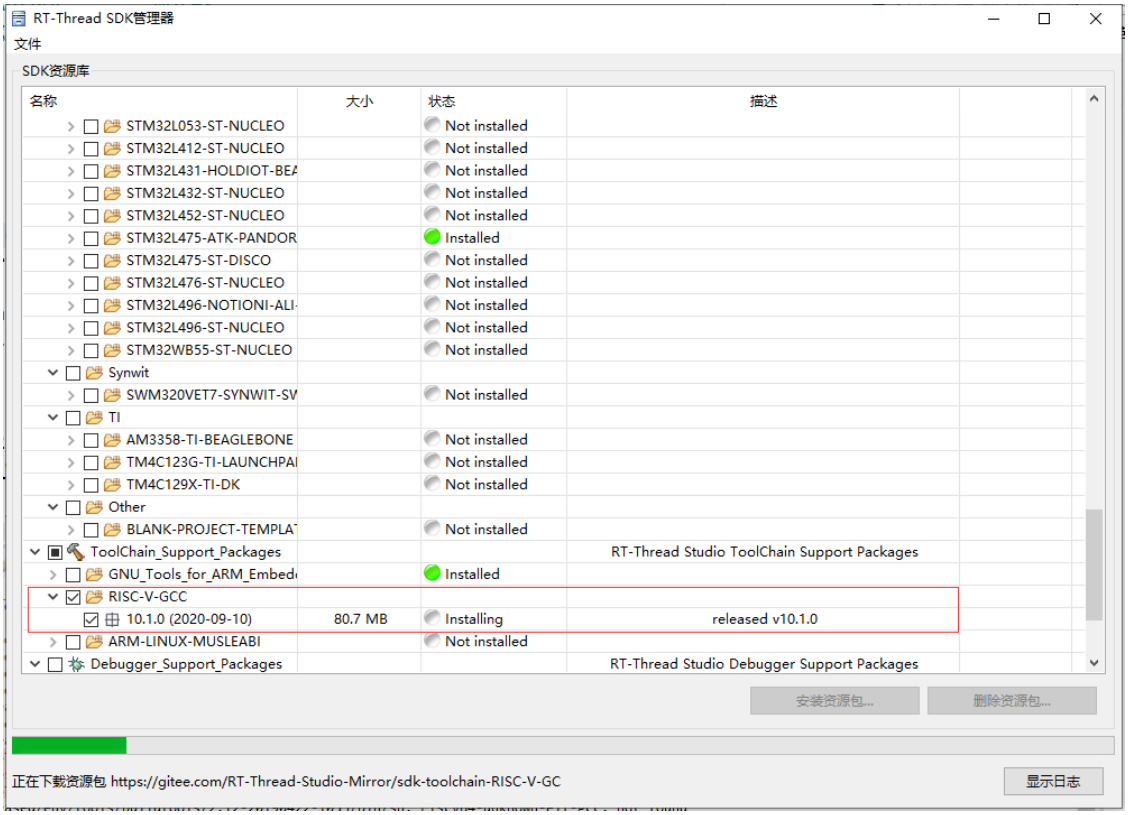
点击 文件-> 新建-> RT-Thread 项目控件



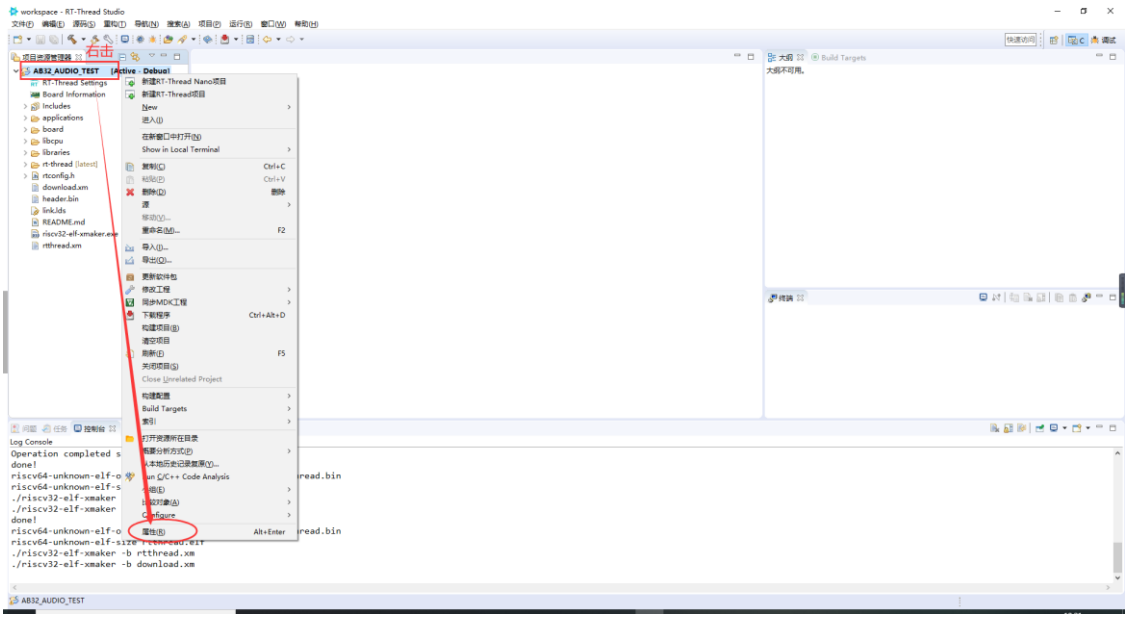
选择基于开发板的项目，填写工程名字，选择我们**使用到的开发板（AB32VG1）**，调试器我们随便选，下载方式不是通过此处下载



注意：如果第一次使用 RISC-V 芯片需要安装工具链，在 SDK 管理器中下载工具链



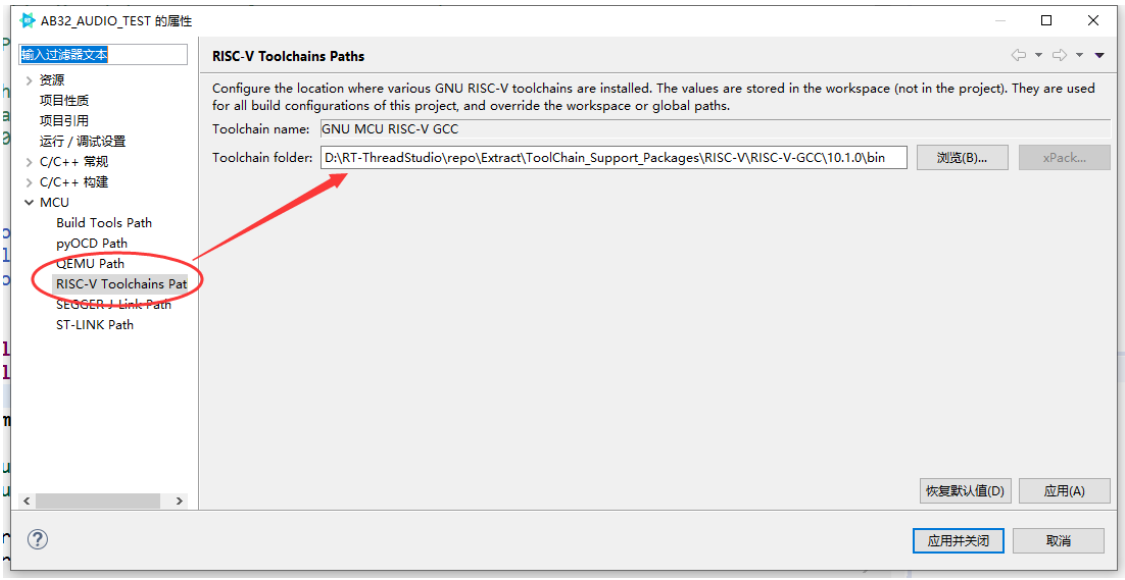
然右击项目名称，进入属性



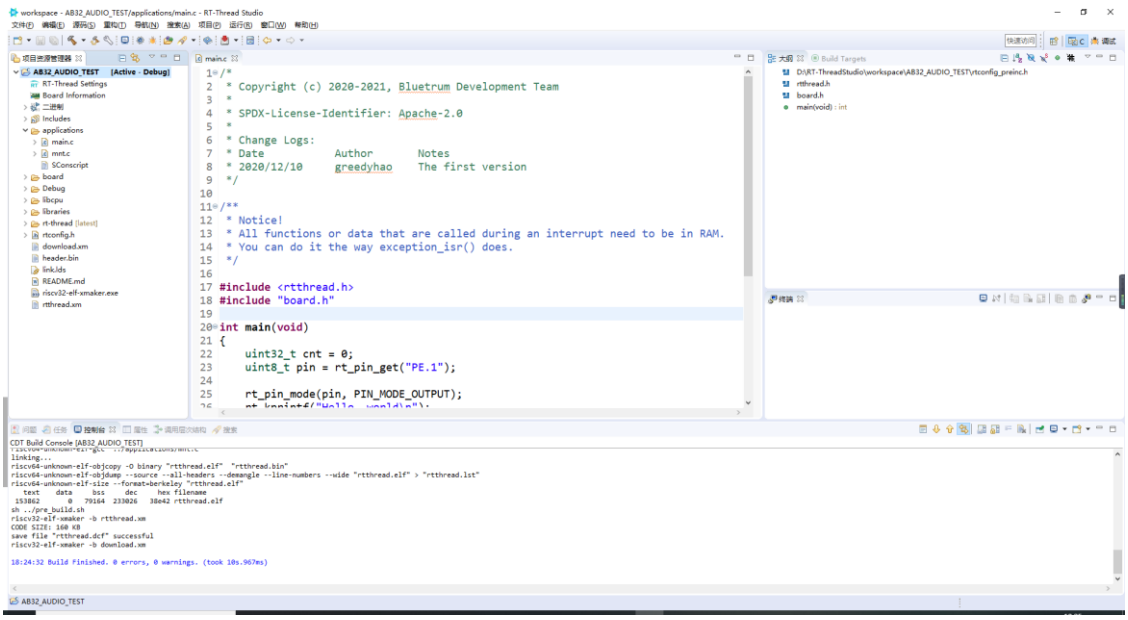
找到 MCU->RISC-V ToolchainsPat ，配置 Tool 的环境，在软件安装位置下面的路径

中

软件安装位置 \RT-ThreadStudio\repo\Extract\ToolChain_Support_Packages\RISC-V\RISC-V-GCC\10.1.0\bin



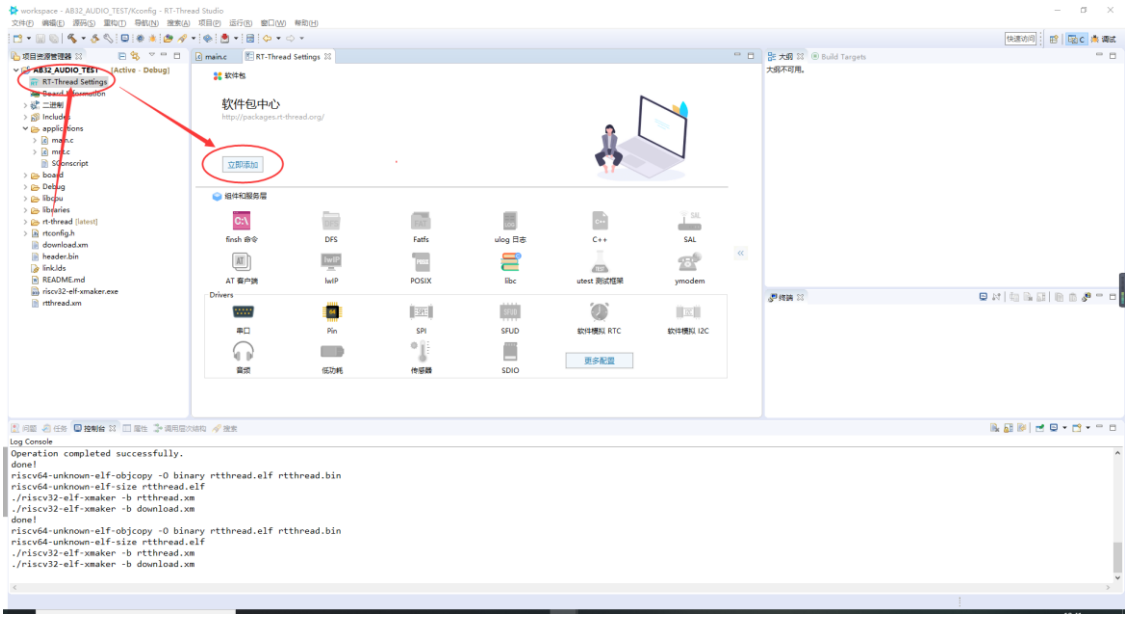
工程新建后左边的项目资源管理器会显示我们的工程，我们把他展开，点击小锤子图标编译一下，编译结果如下



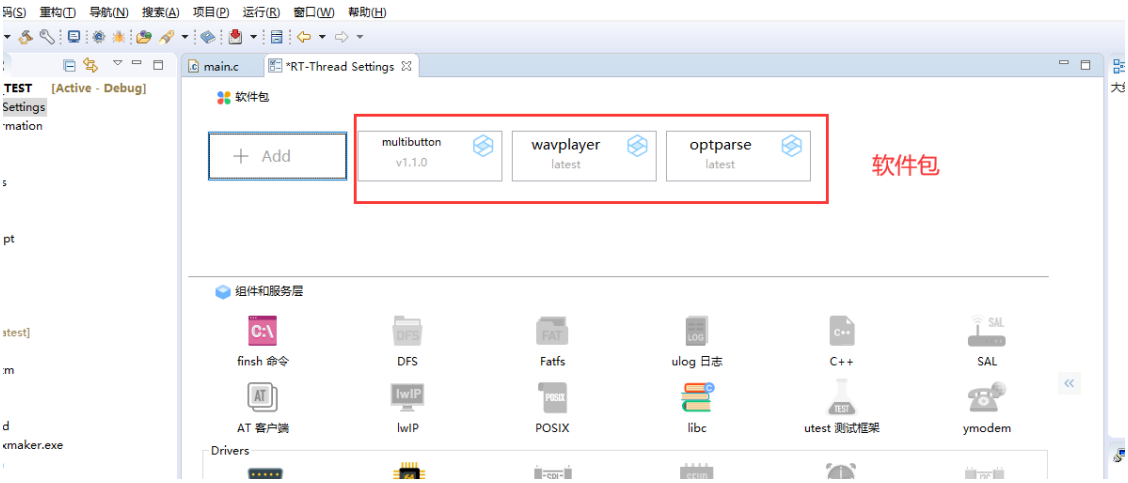
编译无报错，新建工程完成了！

2.2 RT-Thread Studio 配置 Audio

点击 RT Thread Setting -> 添加软件包



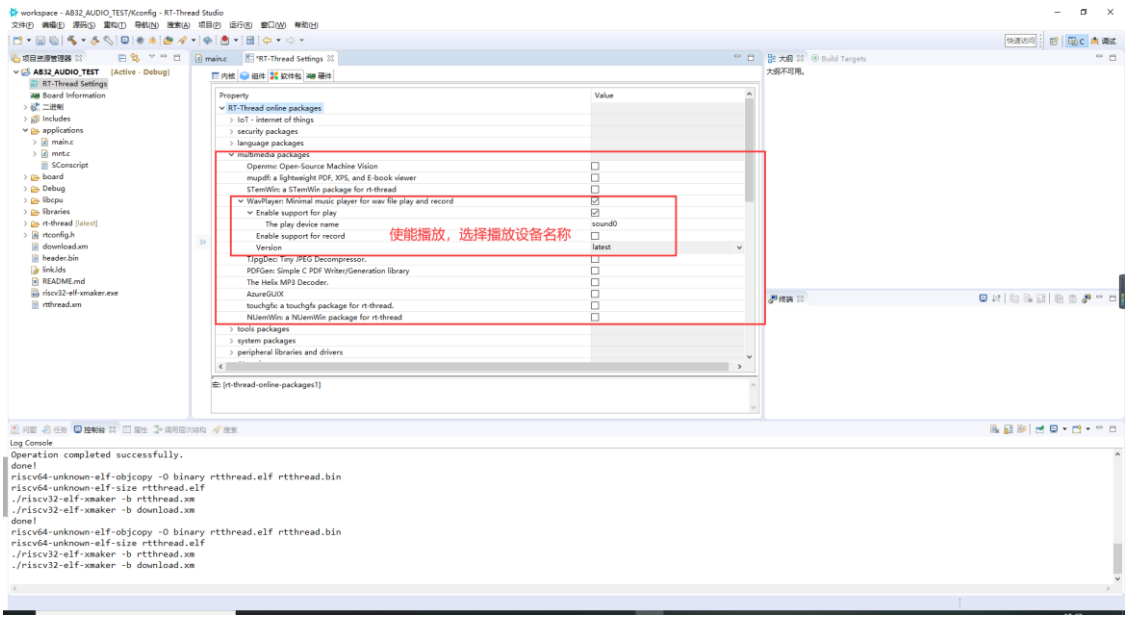
依次添加 multibutton、wavplayer、optoarse 三个软件包



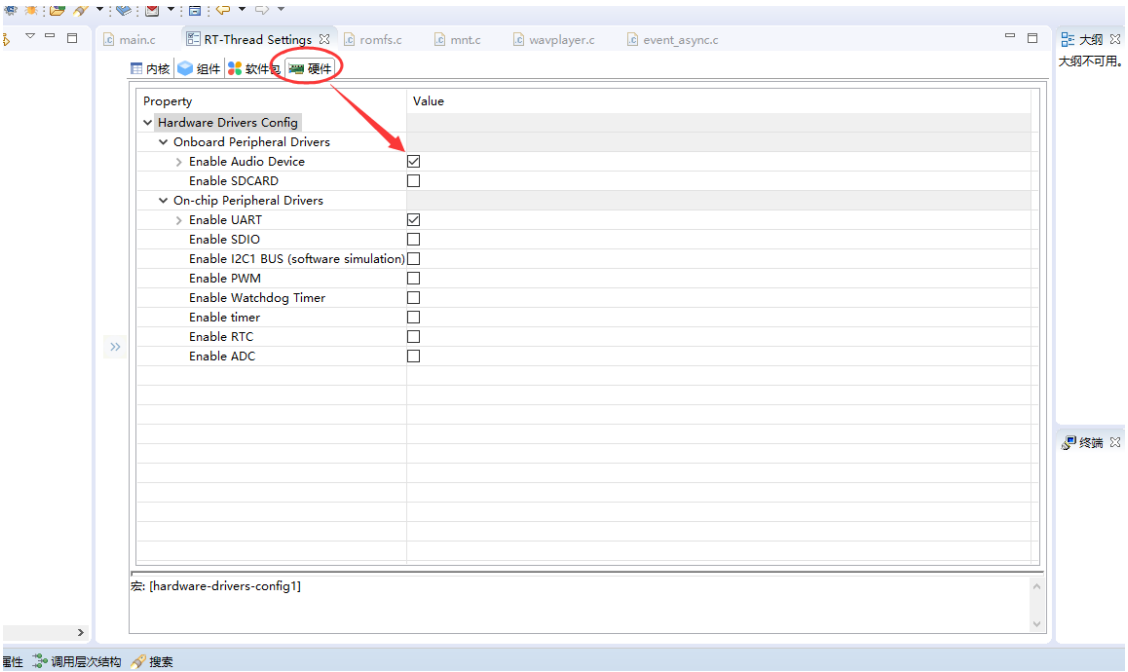
点击更多配置 -> 进入软件包 -> 配置使用的软件包

optparse 是 WavPlay 软件包依赖，因此 optparse 软件包在 wavplayer 勾选后，自动选择。optparse 模块主要用来为脚本传递命令参数，采用预先定义好的选项来解析命令行参数，所以我吗只要配置 WavPlay 软件包就行

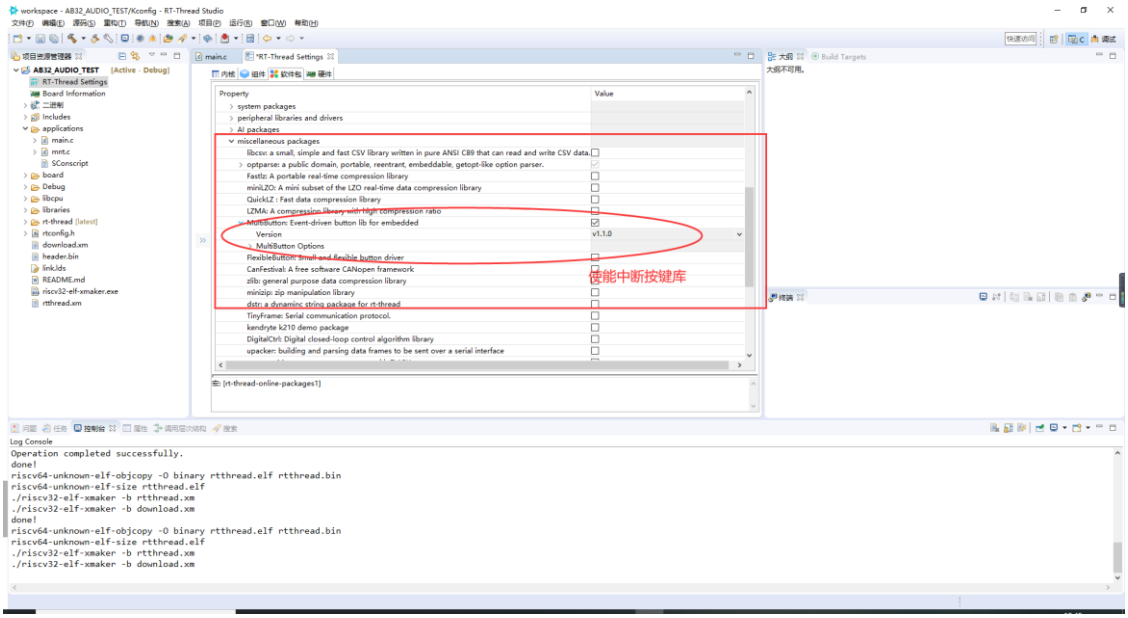
3. wavplayer wav 播放软件包安装



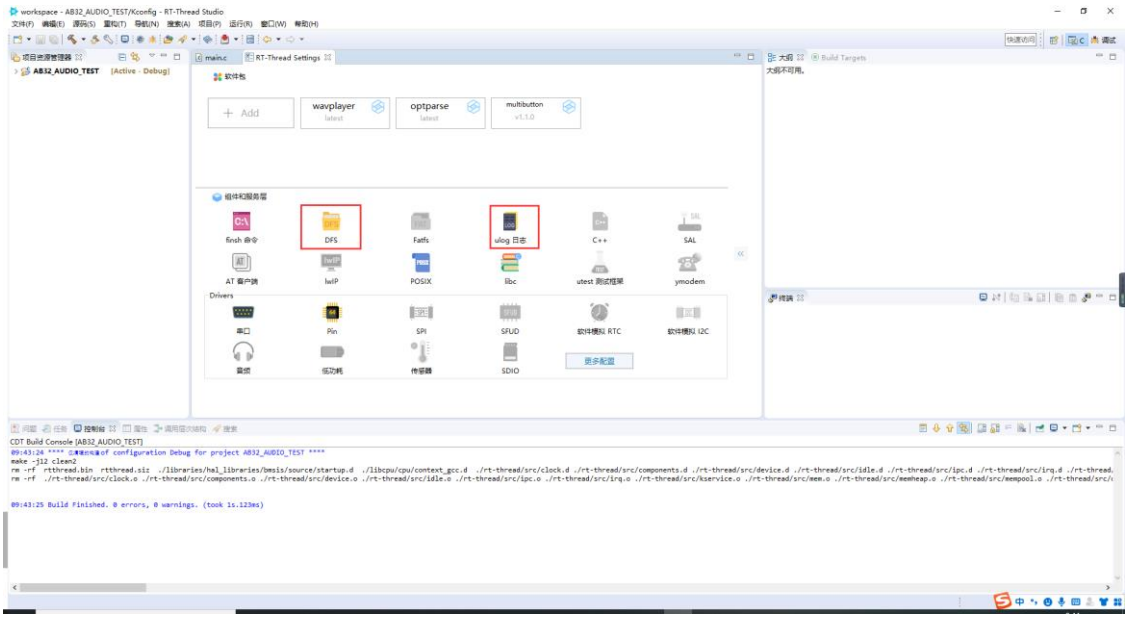
点击硬件->Enable Audio Device 使能硬件



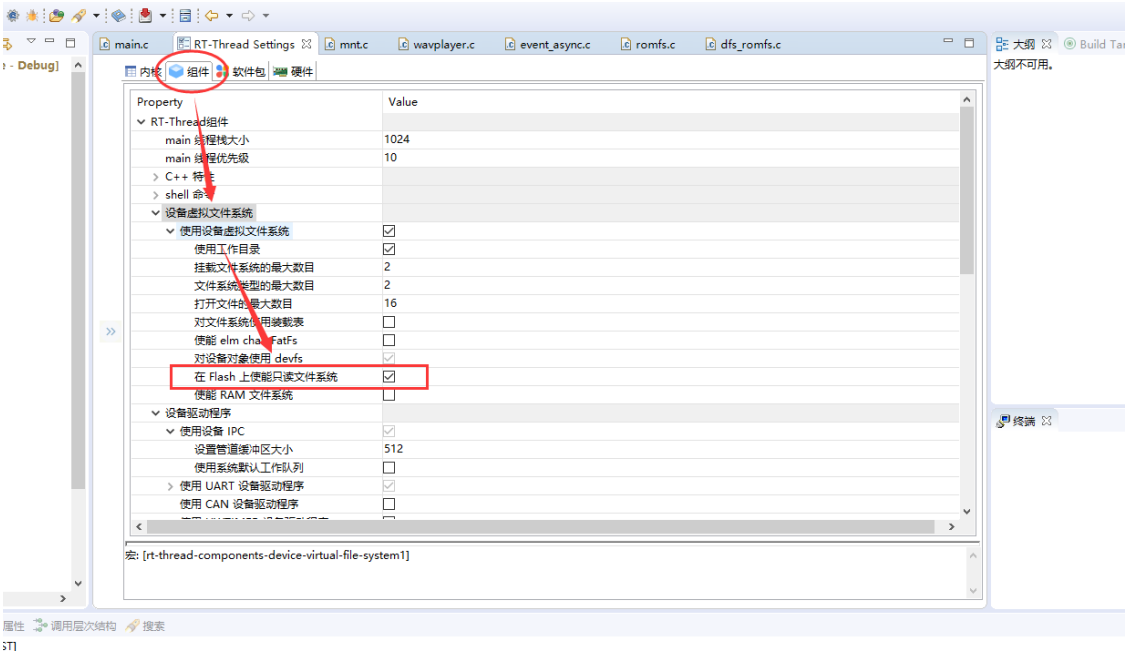
- multibutton 多按键软件包安装



使能虚拟文件系统 DFS，开启 Ulog 调试日志



点击组件->设备虚拟文件系统->使能 Flash 上只读文件系统（ROMFS）



点击保存，使 RT Thread 的配置生效，下一步进入代码修改

2.2 代码编写

首先需要下载 romfs.c（本文件包含了两个音频文件用于 demo 播放）放到 applications 下：下载地址

然后需要注意的一点是，需要修改 mnt.c 的内容，对 ROMFS 进行挂载，在 mnt 文件中添加下面代码即可

```
#include <dfs_fs.h>
#include "dfs_romfs.h"

int mnt_init(void)
{
    if (dfs_mount(RT_NULL, "/", "rom", 0, &(romfs_root)) == 0)
    {
        rt_kprintf("ROM file system initializated!\n");
    }
    else
    {
        rt_kprintf("ROM file system initializate failed!\n");
```

```
    }

    return 0;
}
INIT_ENV_EXPORT(mnt_init);
```

然后在 applications 下新建 event_async.c 文件，编写下述所有代码：

```
//头文件包含
#include <rtthread.h>
#include <rtdevice.h>
#include "board.h"
#include <multi_button.h>
#include "wavplayer.h"
//按键获取
#define BUTTON_PIN_0 rt_pin_get("PF.0")
#define BUTTON_PIN_1 rt_pin_get("PF.1")
//按键编号
#define NUM_OF_SONGS        (2u)
//按键结构体
static struct button btn_0;
static struct button btn_1;

static uint32_t cnt_0 = 0;
static uint32_t cnt_1 = 0;
//音乐名称
static char *table[2] =
{
    "wav_1.wav",
    "wav_2.wav",
};

void saia_channels_set(uint8_t channels);
void saia_volume_set(rt_uint8_t volume);
uint8_t saia_volume_get(void);
```

按键驱动编写

```
//读取按键驱动
static uint8_t button_read_pin_0(void)
{
    return rt_pin_read(BUTTON_PIN_0);
```

```
}

static uint8_t button_read_pin_1(void)
{
    return rt_pin_read(BUTTON_PIN_1);
}
```

按键 0 回调函数

```
//按键 0 回调函数
static void button_0_callback(void *btn)
{
    uint32_t btn_event_val;

    btn_event_val = get_button_event((struct button *)btn);

    switch(btn_event_val)
    {
    case SINGLE_CLICK:
        if (cnt_0 == 1) {
            saia_volume_set(30);
        }else if (cnt_0 == 2) {
            saia_volume_set(50);
        }else {
            saia_volume_set(100);
            cnt_0 = 0;
        }
        cnt_0++;
        rt_kprintf("vol=%d\n", saia_volume_get());
        rt_kprintf("button 0 single click\n");
    break;

    case DOUBLE_CLICK:
        if (cnt_0 == 1) {
            saia_channels_set(1);
        }else {
            saia_channels_set(2);
            cnt_0 = 0;
        }
        cnt_0++;
        rt_kprintf("button 0 double click\n");
    break;

    case LONG_PRESS_START:
        rt_kprintf("button 0 long press start\n");
```

```
        break;

        case LONG_PRESS_HOLD:
            rt_kprintf("button 0 long press hold\n");
        break;
        }
}
```

按键 1 回调函数

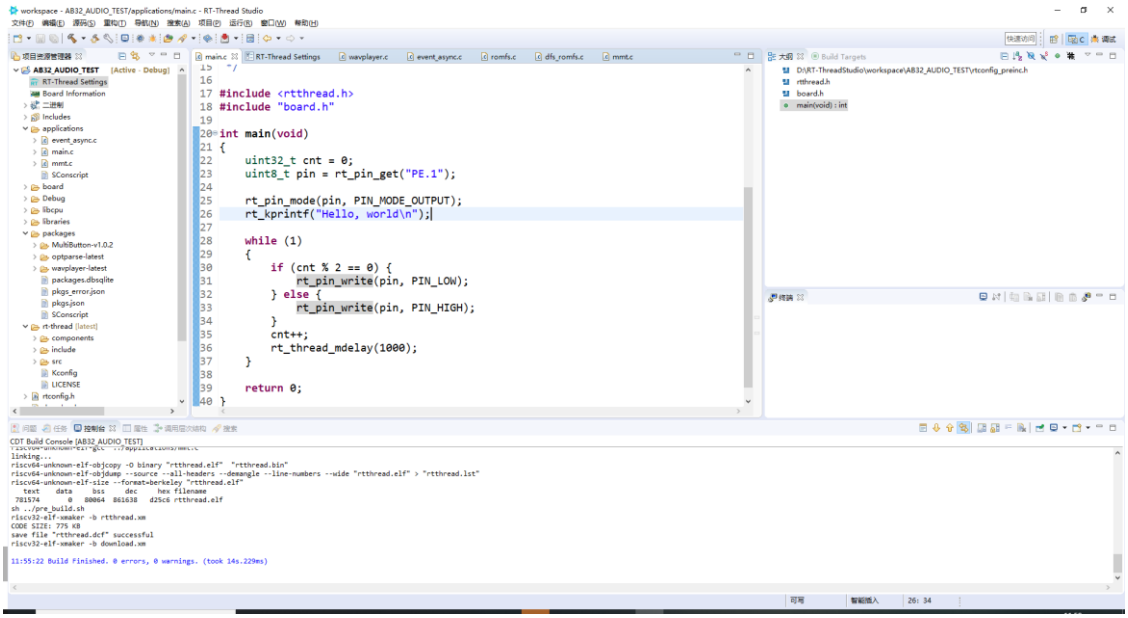
```
//按键 1 回调函数
static void button_1_callback(void *btn)
{
        uint32_t btn_event_val;

        btn_event_val = get_button_event((struct button *)btn);

        switch(btn_event_val)
        {
        case SINGLE_CLICK:
            wavplayer_play(table[(cnt_1++) % NUM_OF_SONGS]);
            rt_kprintf("button 1 single click\n");
        break;

        case DOUBLE_CLICK:
            rt_kprintf("button 1 double click\n");
        break;

        case LONG_PRESS_START:
            rt_kprintf("button 1 long press start\n");
        break;

        case LONG_PRESS_HOLD:
            rt_kprintf("button 1 long press hold\n");
        break;
        }
}
```

任务实体

```
//按键任务实体
static void btn_thread_entry(void* p)
{
    while(1)
    {
        /* 5ms */
        rt_thread_delay(RT_TICK_PER_SECOND/200);
        button_ticks();
    }
}
//按键初始化
static int multi_button_test(void)
{
    rt_thread_t thread = RT_NULL;

    /* Create background ticks thread */
    thread = rt_thread_create("btn", btn_thread_entry, RT_NULL, 1024, 10, 10);
    if(thread == RT_NULL)
    {
        return RT_ERROR;
    }
    rt_thread_startup(thread);

    /* low level drive */
    rt_pin_mode    (BUTTON_PIN_0, PIN_MODE_INPUT_PULLUP);
    button_init    (&btn_0, button_read_pin_0, PIN_LOW);
    button_attach(&btn_0, SINGLE_CLICK,         button_0_callback);
    button_attach(&btn_0, DOUBLE_CLICK,         button_0_callback);
    button_attach(&btn_0, LONG_PRESS_START, button_0_callback);
    button_attach(&btn_0, LONG_PRESS_HOLD,    button_0_callback);
    button_start (&btn_0);

    rt_pin_mode    (BUTTON_PIN_1, PIN_MODE_INPUT_PULLUP);
    button_init    (&btn_1, button_read_pin_1, PIN_LOW);
    button_attach(&btn_1, SINGLE_CLICK,         button_1_callback);
    button_attach(&btn_1, DOUBLE_CLICK,         button_1_callback);
    button_attach(&btn_1, LONG_PRESS_START, button_1_callback);
    button_attach(&btn_1, LONG_PRESS_HOLD,    button_1_callback);
    button_start (&btn_1);

    return RT_EOK;
}
//添加到系统初始化
```

INIT_APP_EXPORT(multi_button_test);

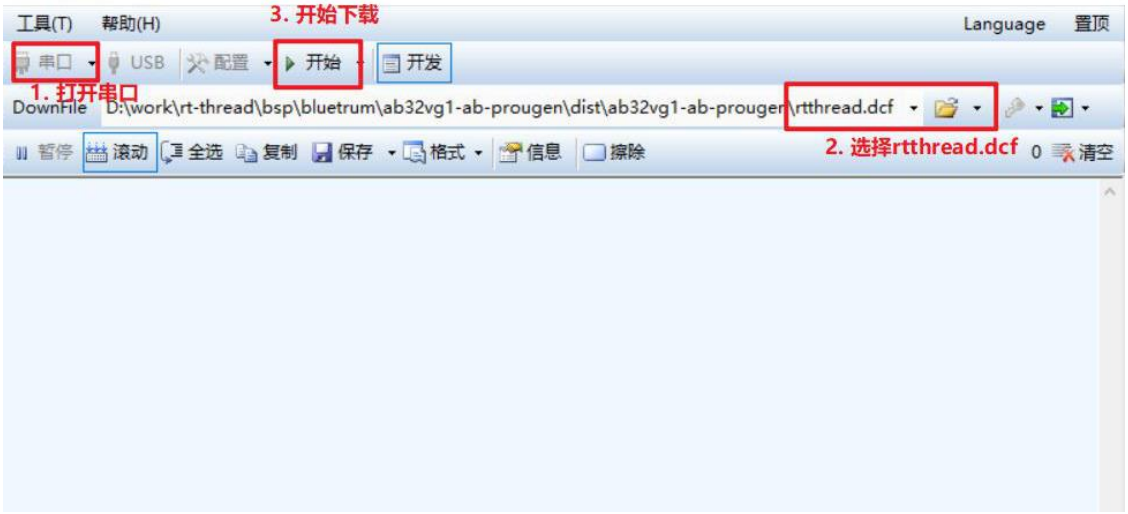
编译一下，无报错



3.代码验证

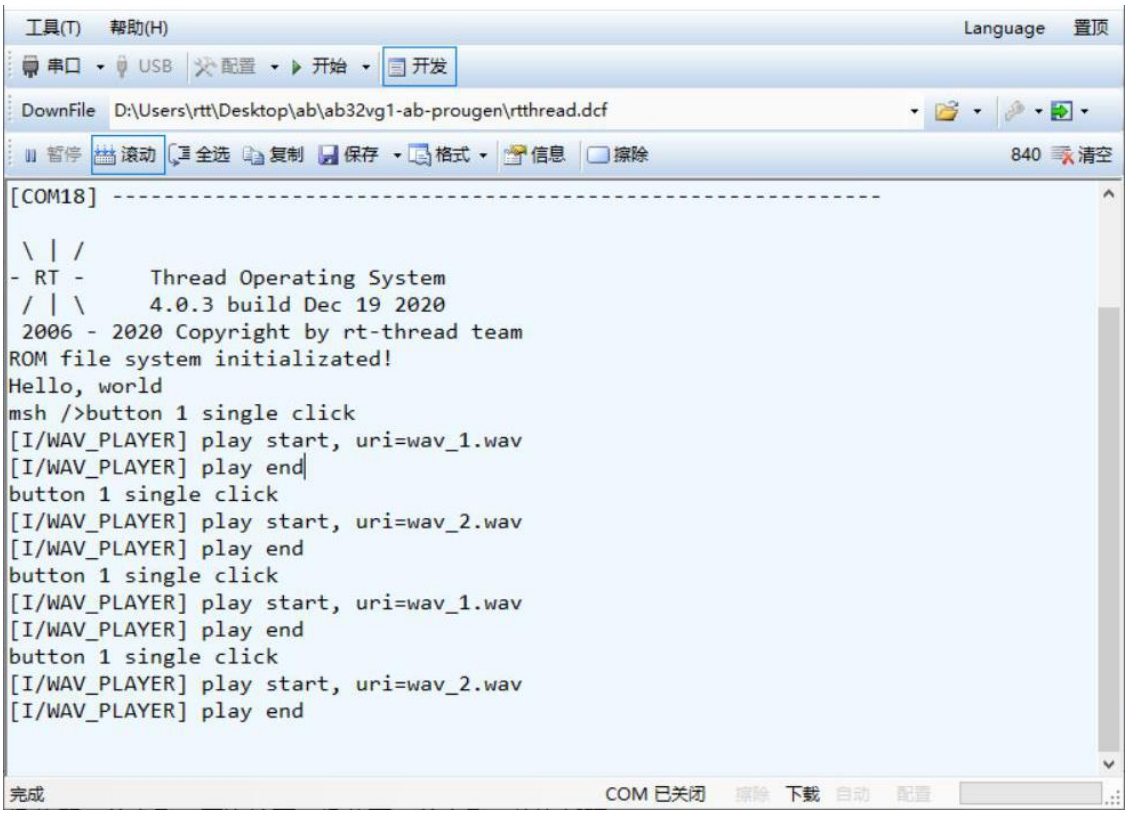
demo 编写完成后，单击编译按钮开始编译，编译成功后下载编译后生成的 .dcf 固件到芯片；

双击打开 Downloader v1.9.7。



下载成功后会在串口界面打印"Hello World"， 并会有 led 灯闪烁

此时按下按键 S2，会播放第一首音乐，再次按下，播放下一首音乐，依次循环，并且打印信息到调试终端



4.章节总结

得益于 RT-Thread 丰富的软件层与中间件，音频开发基本上不需要用户过多操作，在不知道芯片的底层原理的情况下仍能开发音频应用，只需要少量代码将功能整合起来即可，使用起来非常的方便快捷，大大提高了开发效率。

十五、中科蓝讯 AB32VG1 上的 mic 实践

1. 前言说明

1.1 本章内容

本章通过 RT-Thread Studio 配置 AB32VG1 片上 mic 的功能，实现通过 mic 采集声音后使用耳机播放的功能。

1.2 模块介绍

AB32VG1 内部集成 AUDIO 功能。

16 位立体声数模转换器和双通道 16 位模数转换器的音频编解码器；

支持灵活的音频均衡调节；

支持采样率 8、11.025、12、16、22.05、32、44.1 和 48KHz；

4 路立体声模拟多路复用器；

双通道麦克风放大器输入；

高性能立体声音频 ADC，信噪比 90dB；

高性能立体声音频数模转换器，95 分贝信噪比，耳机放大器输出；

1.3 开发软件

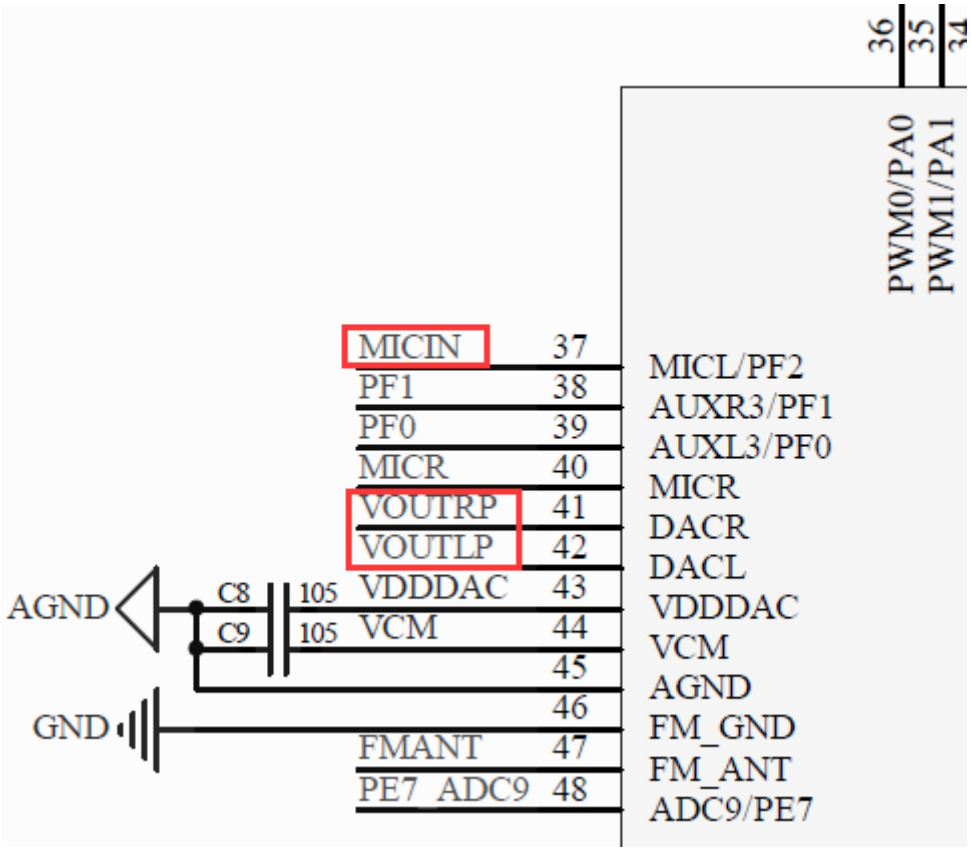
开发环境：RT-Thread Studio

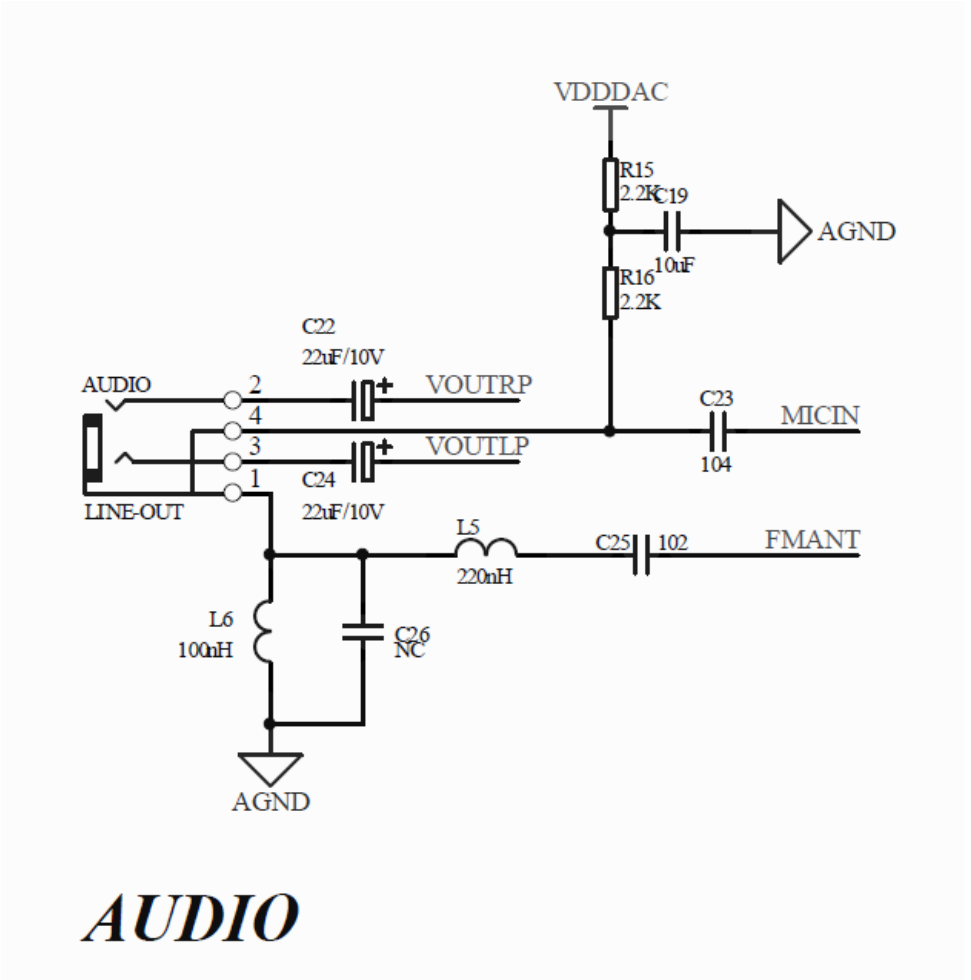
下载工具：Downloader.exe

2. 步骤说明

2.1 原理图

针对音频功能，AB32VG1 有专用的 IO 对应，并且外围电路非常简单。


36
35
34
PWM0/PA0
PWM1/PA1
MICIN
37
MICL/PF2
PF1
38
AUXR3/PF1
PF0
39
AUXL3/PF0
MICR
40
MICR
VOUTRP
41
DACR
VOUTLP
42
DACL
AGND
C8
105
VDDDAC
43
VDDDAC
C9
105
VCM
44
VCM
45
AGND
46
GND
FM_GND
FMANT
47
FM_ANT
PE7_ADC9
48
ADC9/PE7


VDDDAC
R15
2.2K
C19
AGND
10uF
R16
2.2K
C22
22uF/10V
AUDIO
2
VOUTRP
C23
MICIN
4
3
VOUTLP
104
1
C24
22uF/10V
LINE-OUT
L5
C25
102
FMANT
220nH
L6
100nH
C26
NC
AGND

AUDIO

2.2 新建工程

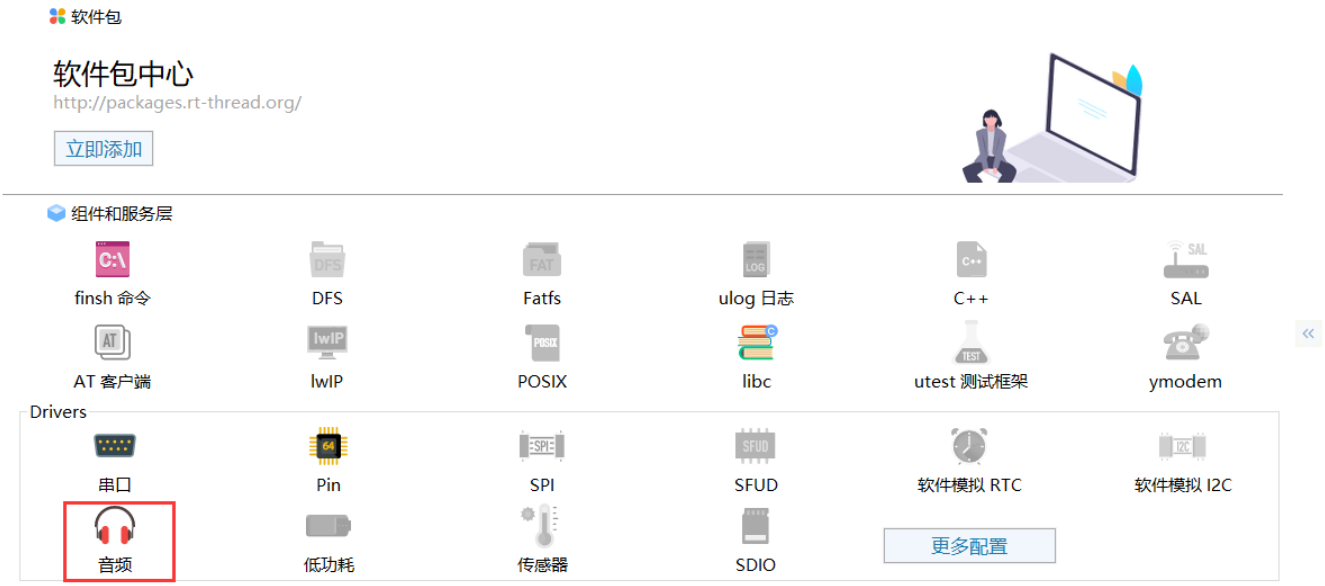
2.1.1.文件->新键->RT-Thread 项目。

2.1.2.选择基于开发板，填写工程名字。

2.1.3.点完成。一个新的项目就建成了。

2.3 加载音频模块

RT-Thread Settings 设置中，软件包中心->Drivers->音频。图标变成彩色后，保存工程文件。



2.4 编写测试文件

在 applications 里新键文件 sdadc_test.c 和 api_sdadc.h。

[sdadc_test.c]

```
#include <rtthread.h>
#include <rthw.h>
#include <rtdevice.h>
#include "api_sdadc.h"
#include "ab32vg1_hal.h"

#define SDADCSIZE 256
```

```
#define SOUND_DEVICE_NAME       "sound0"       /* Audio 设备名称 */
static rt_device_t snd_dev;                    /* Audio 设备句柄 */
rt_mq_t sdadc_mq = RT_NULL;

/**
 * @brief mono to dual
 *
 * @param channel 0:left channel 1:right channel
 * @param dual_buf
 * @param mono_buf
 * @param len
 */
RT_SECTION(".irq.dsp")
static void music_data_mono_2_dual(uint8_t channel, void *dual_buf, void *mono_buf, uint16_t len)
{
    uint16_t i;
    uint8_t *buf1 = (uint8_t *)mono_buf;
    uint8_t *buf2 = (uint8_t *)dual_buf;
    for (i = 0; i < (len/4); i++) {
        buf2[i*4] = buf1[i*4 + channel*2];
        buf2[i*4 + 1] = buf1[i*4 + 1 + channel*2];
        buf2[i*4 + 2] = buf1[i*4 + channel*2];
        buf2[i*4 + 3] = buf1[i*4 + 1 + channel*2];
    }
}

RT_SECTION(".irq.sdadc.str")
const char dbg_str[] = "notice %d\n";

RT_SECTION(".irq.sdadc")
void sdadc_dma_notice(enum sdadc_msg msg)
{
    uint8_t buf = (uint8_t)msg;
    rt_mq_send(sdadc_mq, &buf, 1);
}

RT_SECTION(".irq.sdadc")
void sdadc_isr_wrapper(int vector, void *param)
{
    rt_interrupt_enter();
    sdadc_isr();
    rt_interrupt_leave();
}

void sdadc_test(void)
```

```
{
    rt_err_t ret = RT_EOK;

    rt_uint32_t *sdadc_ptr = RT_NULL;
    sdadc_ptr = rt_calloc(SDADCSIZE, 4);
    if (sdadc_ptr == RT_NULL) {
        rt_kprintf("No memory\n");
    }

    rt_uint32_t *dual_ptr = RT_NULL;
    dual_ptr = rt_calloc(SDADCSIZE, 4);
    if (dual_ptr == RT_NULL) {
        rt_kprintf("No memory\n");
    }

    snd_dev = rt_device_find(SOUND_DEVICE_NAME);
    ret = rt_device_open(snd_dev, RT_DEVICE_OFLAG_WRONLY);
    if (ret == RT_EOK) {
        rt_kprintf("sound0 open successful\n");
    }

    struct rt_audio_caps caps =
    {
        .main_type = AUDIO_TYPE_OUTPUT,
        .sub_type = AUDIO_DSP_SAMPLERATE,
        .udata.config.samplerate = 44100,
    };
    rt_device_control(snd_dev, AUDIO_CTL_CONFIGURE, (void *)&caps);

    sdadc_mq = rt_mq_create("sdadc", 1, 16, RT_IPC_FLAG_FIFO);
    if (sdadc_mq == RT_NULL) {
        rt_kprintf("No memory!\n");
    }

    sdadc_set_channel(CH_MICL0);
    sdadc_set_sample_rate(44100);
    sdadc_set_gain(14 << 5);
    sdadc_set_dma_samples(SDADCSIZE);
    sdadc_start((void *)sdadc_ptr);

    rt_hw_interrupt_install(IRQ_SDADC_VECTOR, sdadc_isr_wrapper, RT_NULL, "sdadc_isr");

    uint8_t mq_buf = 0;
    #define SIZE_MUL        (2u)
    while (1) {
```

```
        if (rt_mq_recv(sdadc_mq, &mq_buf, 1, RT_WAITING_FOREVER) == RT_EOK) {
            if (mq_buf == MSG_SDADC_LHALF_DONE) {
                music_data_mono_2_dual(0, dual_ptr, sdadc_ptr, SDADCSIZE*SIZE_MUL);
                rt_device_write(snd_dev, 0, dual_ptr, SDADCSIZE*SIZE_MUL);
            } else if (mq_buf == MSG_SDADC_LALL_DONE) {
                music_data_mono_2_dual(0, dual_ptr, sdadc_ptr + SDADCSIZE/2,
SDADCSIZE*SIZE_MUL);
                rt_device_write(snd_dev, 0, dual_ptr, SDADCSIZE*SIZE_MUL);
            }
        }
    }
}
MSH_CMD_EXPORT(sdadc_test, sdadc_test);
```

[api_sdadc.h]

```
#ifndef API_SDADC_H__
#define API_SDADC_H__

/**
 * @defgroup SDADC_Channel
 * @{
 */
//left sdadc channel config
#define CH_AUXL_PA6          0x01    //AUXL0(PA6) -> left aux  -> sdadc left channel
#define CH_AUXL_PB1          0x02    //AUXL1(PB1) -> left aux  -> sdadc left channel
#define CH_AUXL_PE6          0x03    //AUXL2(PE6) -> left aux  -> sdadc left channel
#define CH_AUXL_PF0          0x04    //AUXL3(PF0) -> left aux  -> sdadc left channel
#define CH_AUXL_VOUTLN       0x05    //VOUTLN      -> left aux  -> sdadc left channel
#define CH_AUXL_MICL         0x06    //MICL(PF2)   -> left aux  -> sdadc left channel
#define CH_AUXL_MICR         0x07    //MICR        -> left aux  -> sdadc left channel
#define CH_AUXL_VOUTRP       0x08    //VOUTRP      -> left aux  -> sdadc left channel
#define CH_MICL0             0x0c    //MICL(PF2)   -> left mic  -> sdadc left channel
#define CH_MICL1             0x0d    //MICR        -> left mic  -> sdadc left channel

//right sdadc channel config
#define CH_AUXR_PA7          0x10    //AUXR0(PA7) -> right aux -> sdadc right channel
#define CH_AUXR_PB2          0x20    //AUXR1(PB2) -> right aux -> sdadc right channel
#define CH_AUXR_PE7          0x30    //AUXR2(PE7) -> right aux -> sdadc right channel
#define CH_AUXR_PF1          0x40    //AUXR3(PF1) -> right aux -> sdadc right channel
#define CH_AUXR_VOUTLN       0x50    //VOUTLN      -> right aux -> sdadc right channel
#define CH_AUXR_MICL         0x60    //MICL(PF2)   -> right aux -> sdadc right channel
#define CH_AUXR_MICR         0x70    //MICR        -> right aux -> sdadc right channel
#define CH_AUXR_VOUTRN       0x80    //VOUTRN      -> right aux -> sdadc right channel
#define CH_MICR0             0xc0    //MICR        -> right mic -> sdadc right channel
```

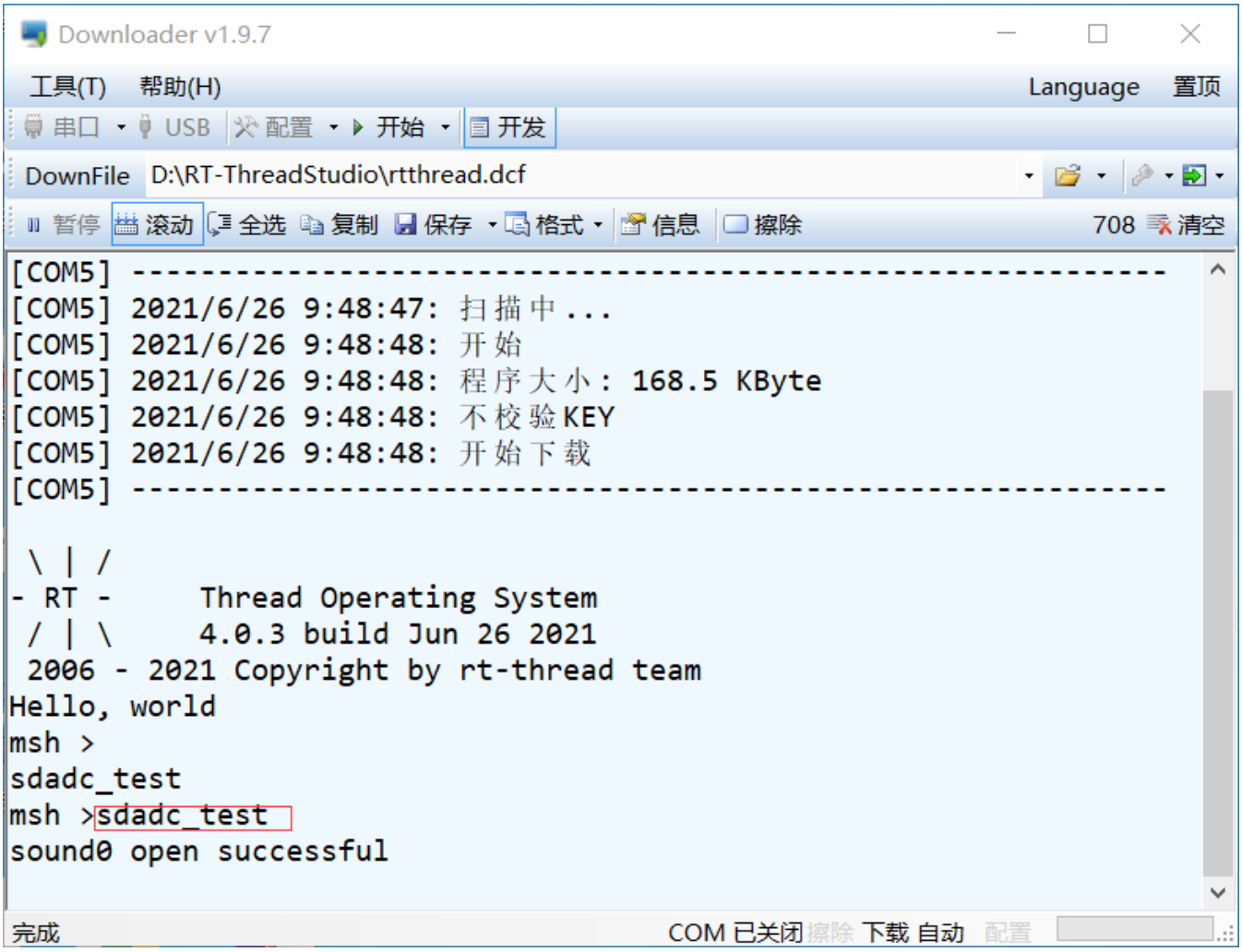
```
#define CH_MICR1                    0xd0        //MICL(PF2)    -> right mic -> sdadc right channel
/**
 * @}
 *
 */

#define CHANNEL_L      0x0F
#define CHANNEL_R      0xF0

/**
 * @defgroup SDADC_Message
 * @{
 */
enum sdadc_msg {
      MSG_SDADC_LHALF_DONE = 1,
      MSG_SDADC_LALL_DONE,
      MSG_SDADC_RHALF_DONE,
      MSG_SDADC_RALL_DONE,
};
/**
 * @}
 *
 */

/**
 * @brief      Set the SDADC channel.
 *
 * @param      channel This parameter can be a value of @ref SDADC_Channel
 * @return     int
 *                 0: OK
 *                -1: ERROR
 */
int sdadc_set_channel(uint8_t channel);

/**
 * @brief      Set the SDADC sample rate.
 *
 * @param      sample_rate support 48000, 44100, 38000, 32000, 24000, 22050, 16000, 12000, 11025,
8000.
 * @return     int
 *                 0: OK
 *                -1: ERROR
 */
int sdadc_set_sample_rate(uint16_t sample_rate);
```

```
/**
 * @brief      Set the SDADC gain.
 *
 * @param      gain Analog gain will only be set before the SDADC works.
 *                  [16:5] analog gain. Range from 0 to 23. Step is 3DB.(-6db ~ +63db)
 *                  [ 4:0] digital gain. Range from 0 to 31. Step is 3/32 DB.(0db ~ +3db)
 * @return     int
 *                  0: OK
 *                 -1: ERROR
 */
int sdadc_set_gain(uint16_t gain);

/**
 * @brief      Set the SDADC DMA samples.
 *
 * @param      samples
 * @return     int
 *                  0: OK
 *                 -1: ERROR
 */
int sdadc_set_dma_samples(uint16_t samples);

/**
 * @brief      Get the SDADC channel.
 *
 * @return     int channel
 */
int sdadc_get_channel(void);

/**
 * @brief      Get the SDADC sample rate.
 *
 * @return     int sample_rate
 */
int sdadc_get_sample_rate(void);

/**
 * @brief      Get the SDADC gain.
 *
 * @return     int gain
 */
int sdadc_get_gain(void);

/**
 * @brief      Get the SDADC DMA samples.
```

```
 *
 * @return    int samples
 */
int sdadc_get_dma_samples(void);

/**
 * @brief      Start the SDADC.
 *
 * @param      rx_buf The SDADC DMA buffer address.
 * @return    int
 *                0: OK
 *               -1: ERROR
 */
int sdadc_start(void *rx_buf);

/**
 * @brief      Close the SDADC.
 *
 * @return    int
 *                0: OK
 *               -1: ERROR
 */
int sdadc_exit(void);

/**
 * @brief      Will be called when SDADC DMA done.
 *                It needs to be in RAM and reimplemented.
 *
 * @param      msg This parameter can be a value of @ref SDADC_Message
 */
void sdadc_dma_notice(enum sdadc_msg msg);

/**
 * @brief      The SDADC DMA interrupt. Need to be registered.
 *                It needs to be in RAM and registered.
 *
 */
void sdadc_isr(void);

#endif
```

3. 代码验证

下载成功后，输入测试命令 sdadc_test。插上带有 mic 功能的耳机，就可以听到自己说话的声音了。此乃学习英语的神器啊！



4. 总结

AB32VG1 内部有 AUDIO 功能，并且在 RT-Thread Studio 上提供了对音频功能的支持，使得原本复杂的移植工作变成点点点。

十六、中科蓝讯 AB32VG1 上的 WIFI 模块

1. 前言说明

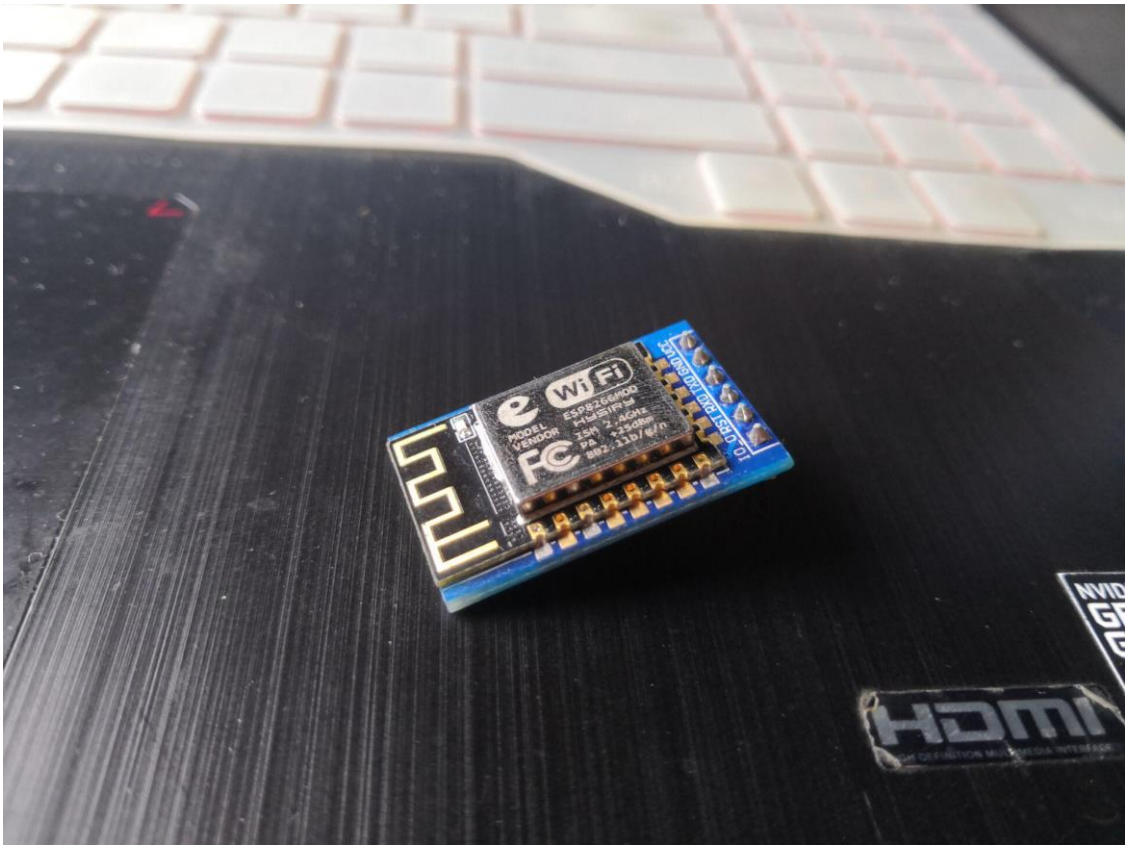
1.1 本章内容

本章通过 RT-Thread Studio 配置 AB32VG1 片上外设 UART1 ，搭载 at_device 软件包连接 WIFI 模块

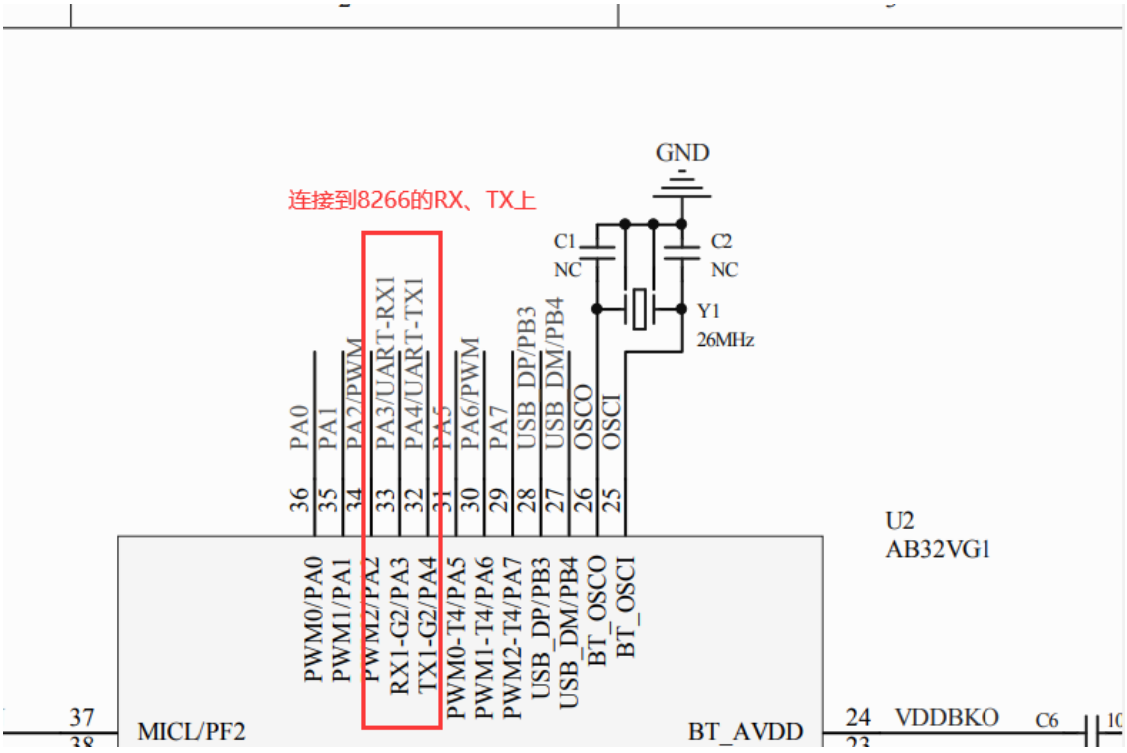
1.2 模块介绍

使用 AB32VG1 开发板做主控，芯片为 AB5301A（LQFP48 封装，主频 120M，片上集成 RAM 192K，flash 8 Mbit，ADC，PWM，USB，UART，IIC 等资源）

WIFI 模块使用 ESP8266：



对照 AB32 原理图接线：



1.3 开发软件

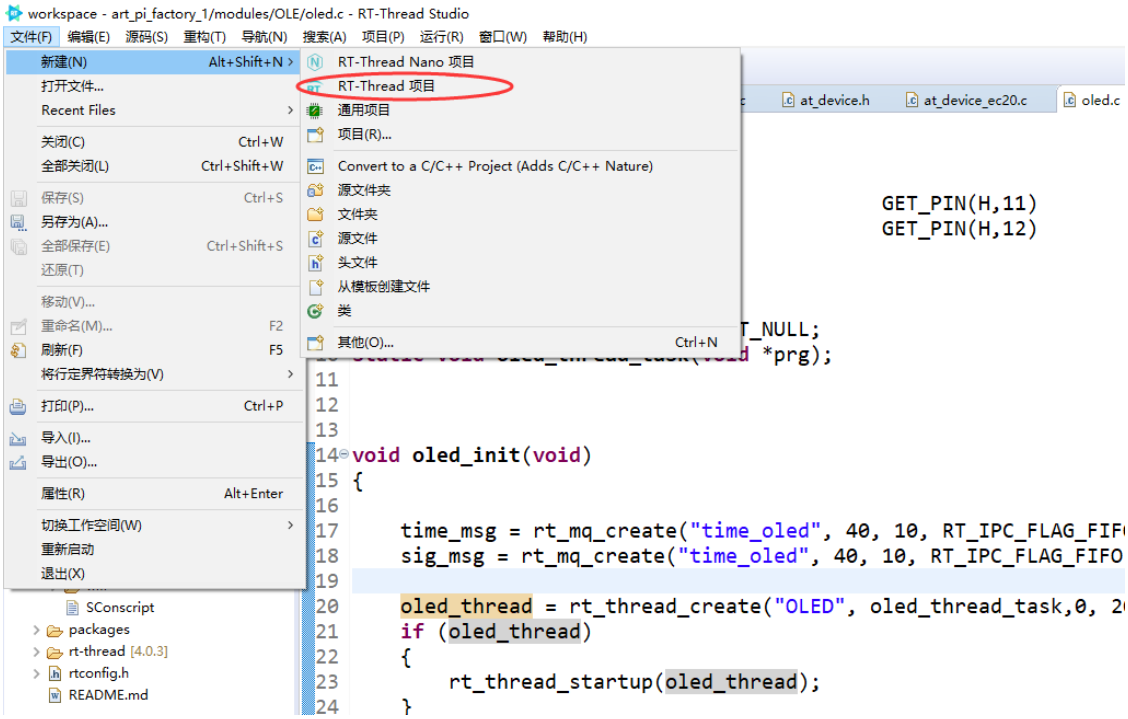
编译平台：RT-Thread Studio： 安装链接

下载平台：Downloader: 安装链接

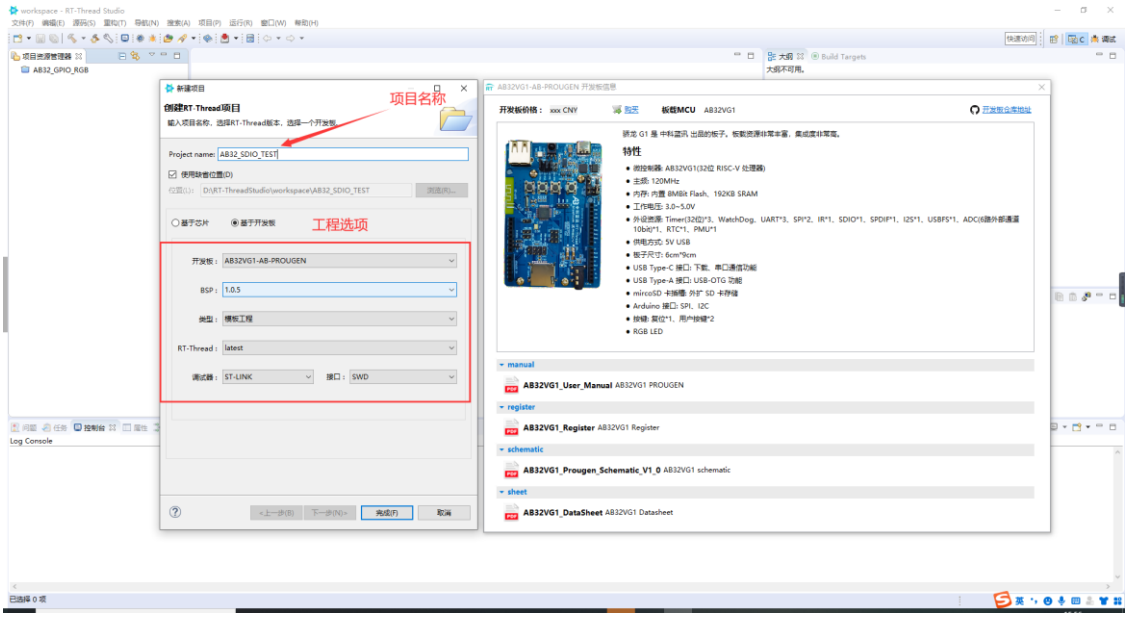
2. 步骤说明

2.1 新建工程

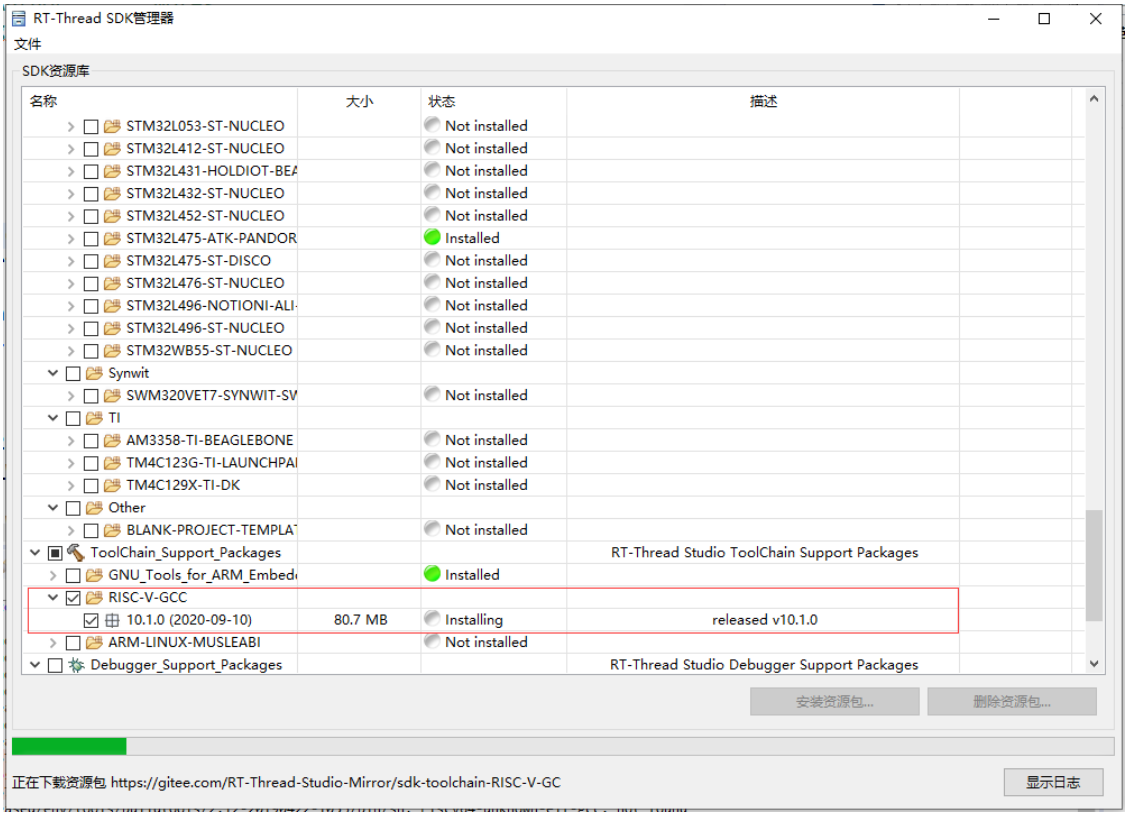
点击 文件-> 新建-> RT-Thread 项目控件



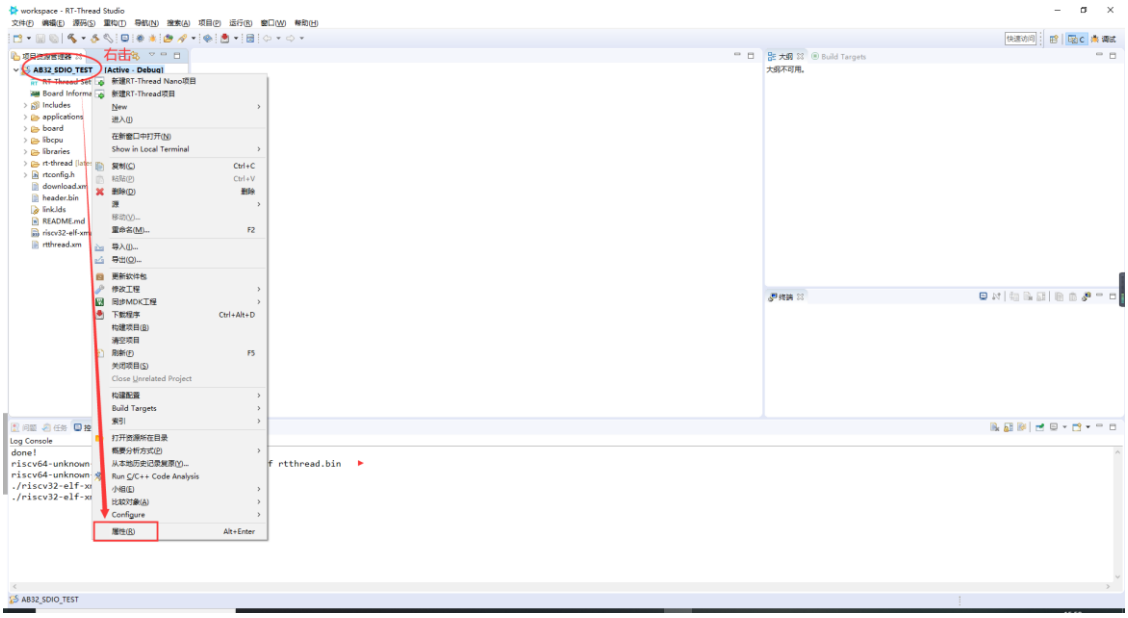
选择基于开发板的项目，填写工程名字，选择我们使用到的开发板（AB32VG1），调试器我们随便选，下载方式不是通过此处下载



注意：如果第一次使用 RISC-V 芯片需要安装工具链，在 SDK 管理器中下载工具链

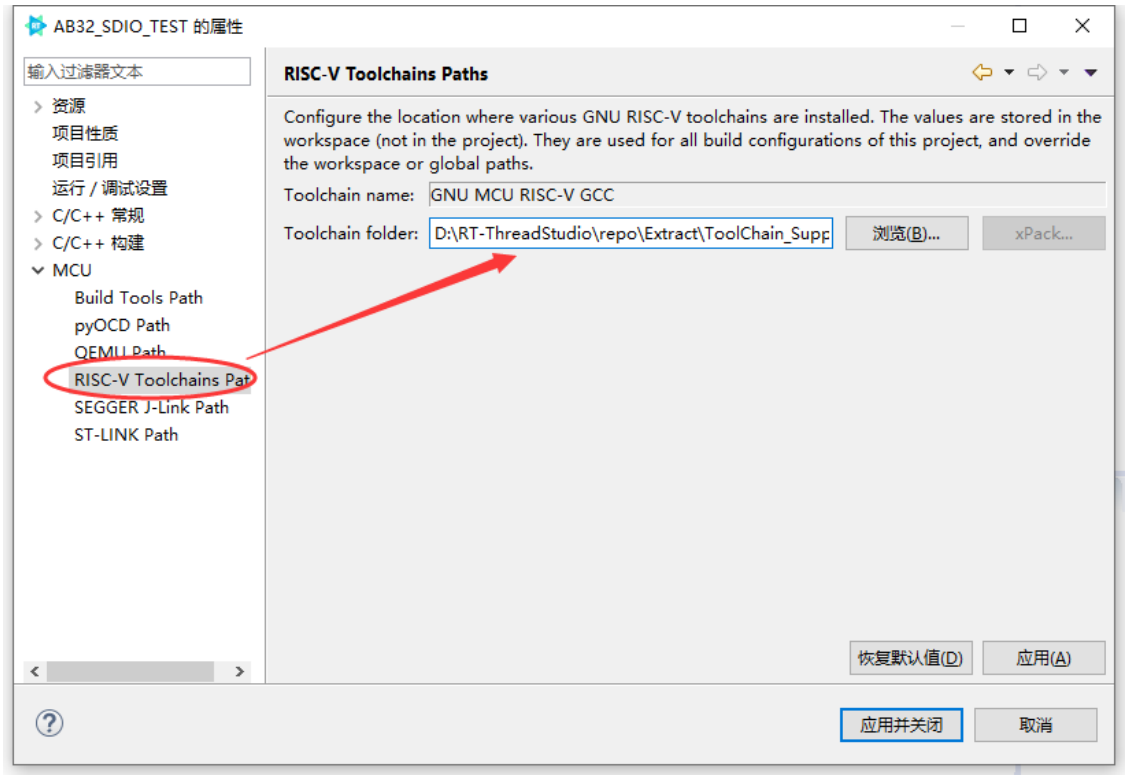


然右击项目名称，进入属性

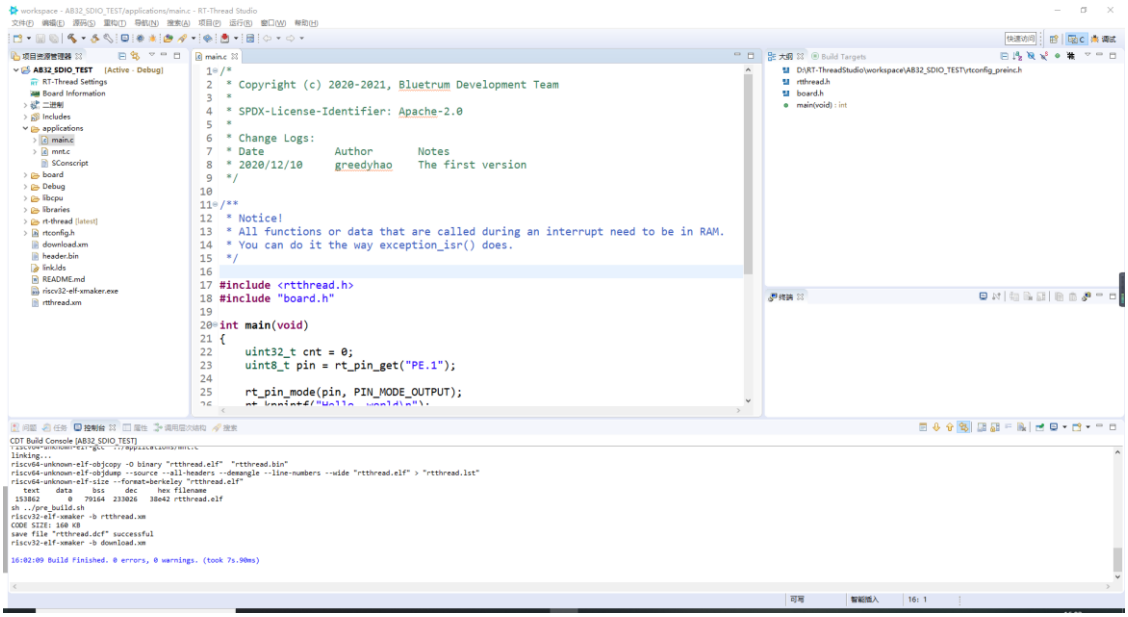


找到 MCU->RISC-V ToolchainsPat ，配置 Tool 的环境，在软件安装位置下面的路径中

软件安装位置 \RT-ThreadStudio\repo\Extract\ToolChain_Support_Packages\RISC-V\RISC-V-GCC\10.1.0\bin



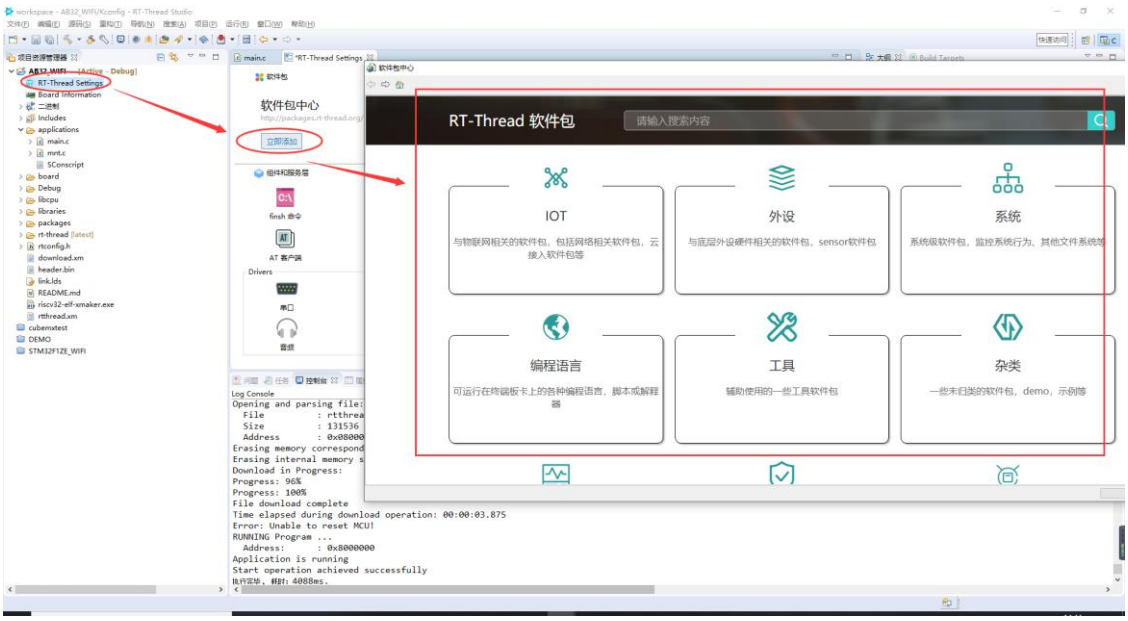
工程新建后左边的项目资源管理器会显示我们的工程，我们把他展开，点击小锤子图标编译一下，编译结果如下



编译无报错，新建工程完成了！

2.2 RT-Thread Studio 配置连接 WIFI

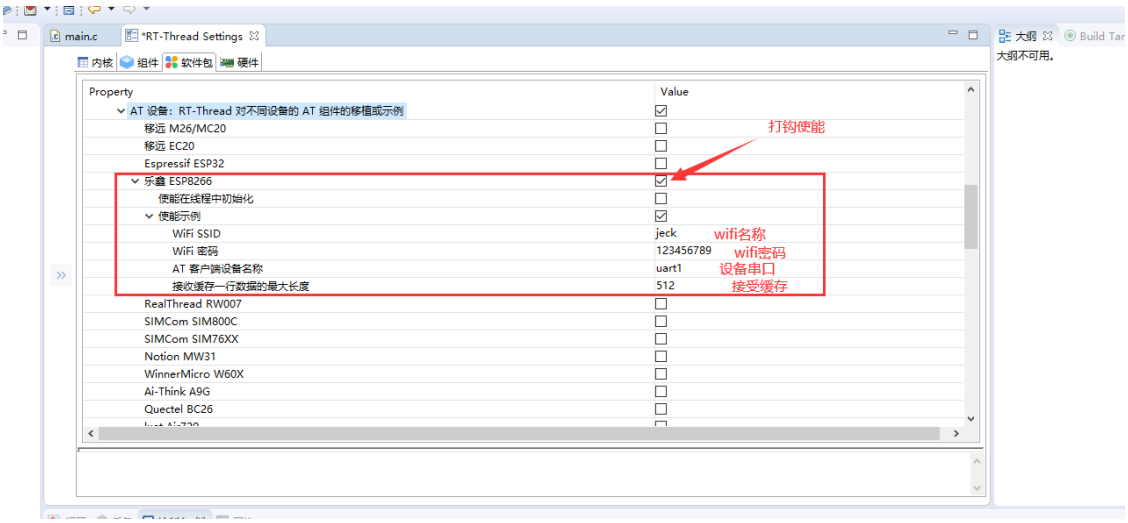
点击 RT-Thread Setting -> 添加软件包



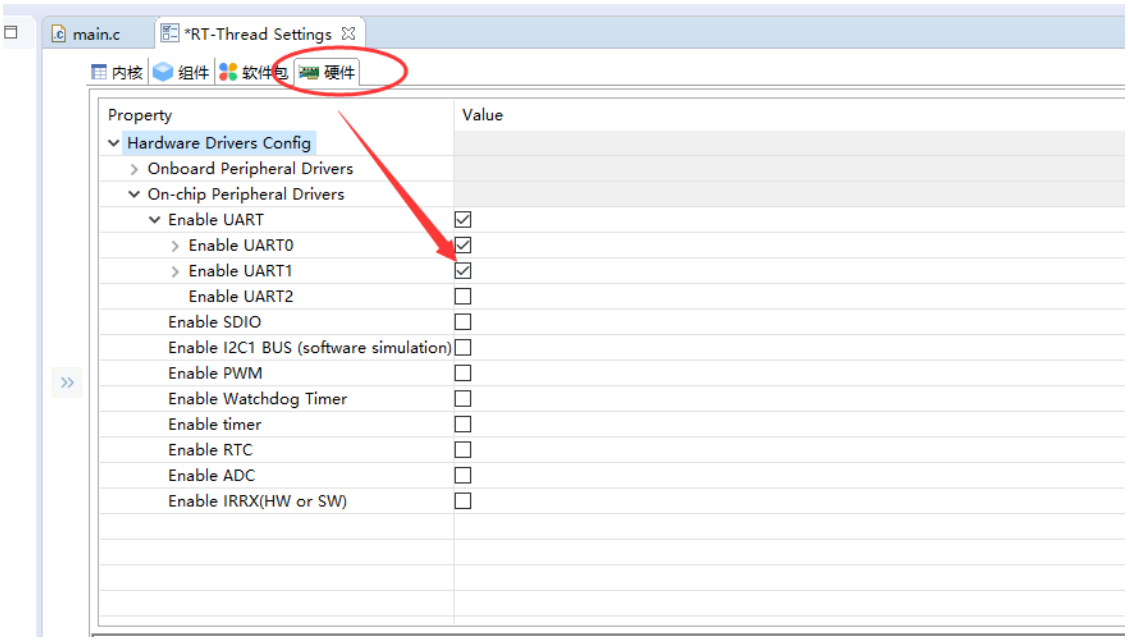
搜索 at_device -> 点击添加

回到 RT-Thread Setting 右击软件包 -> 点击详细配置

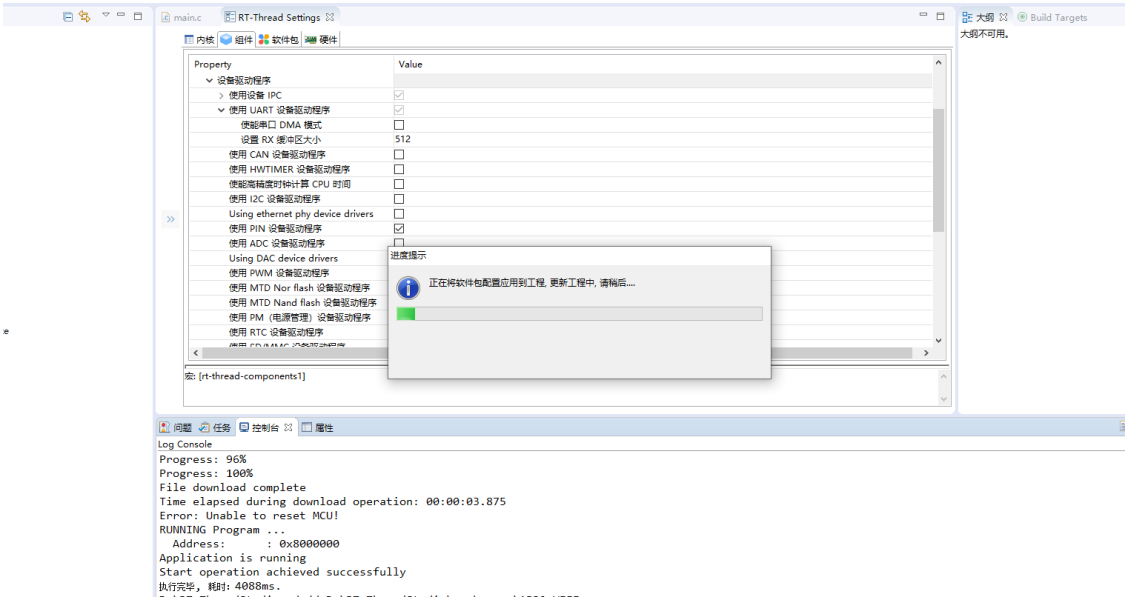
在详细配置里面使能 ESP8266，然后配置我们要连的 WIFI 名称和使用的串口设备



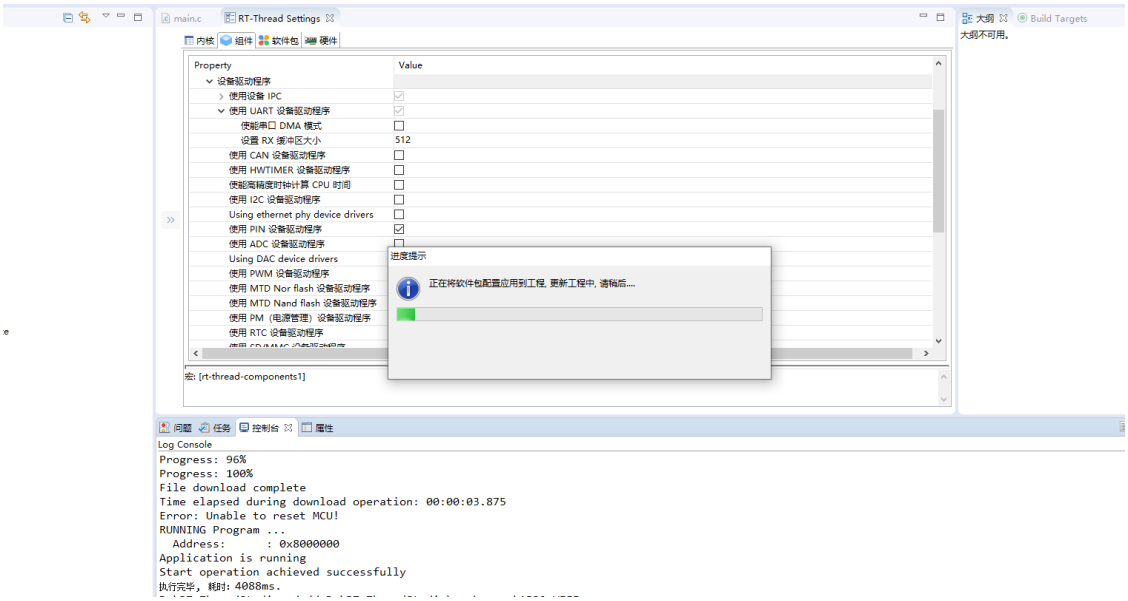
点击硬件，使能芯片外设驱动中的 UART1



配置完成后 Ctrl + S 保存配置，生成配置代码



编译一下代码，没有保存，配置完成



3. 代码验证

编译完成，打开 Downloaded 下载器，通过 download 下载生成的.dcf 文件（第一次使用前需要先安装串口驱动），扫描串口，点击开始后，按一下板子上复位按键下载程序

```
msh >ping www.baidu.com
32 bytes from 36.152.44.95 icmp_seq=0 time=51 ms
32 bytes from 36.152.44.95 icmp_seq=1 time=25 ms
32 bytes from 36.152.44.95 icmp_seq=2 time=28 ms
32 bytes from 36.152.44.95 icmp_seq=3 time=31 ms
msh >
```

连接板子串口，复位观察命令行，可以看到 8266 初始化成功，这里我有一个报错是因为 8266 固件和 at 软件包的对不上，问题不大，有需要可以去乐鑫官网下载更新

```
 \ | /
- RT -     Thread Operating System
 / | \     4.0.3 build Aug 15 2021
 2006 - 2020 Copyright by rt-thread team
[I/sal.skt] Socket Abstraction Layer initialize success.
[I/at.clnt] AT client(V1.3.1) on device uart3 initialize success.
msh >[I/at.dev.esp] esp0 device wifi is disconnect.
[I/at.dev.esp] esp0 device wifi is connected.
[I/at.dev.esp] esp0 device network initialize successfully.
[E/at.clnt] execute command (AT+CIPDNS_CUR?) failed!
[W/at.dev.esp] please check and update esp0 device firmware to support the "AT+CIPDNS_
CUR?" cmd.
```

初始化成功

查看一下模块网口信息：

```
msh >ifconfig
network interface device: esp0 (Default)
MTU: 1500
MAC: 2c 3a e8 0c 07 84
FLAGS: UP LINK_UP INTERNET_UP DHCP_DISABLE
ip address: 192.168.43.235
gw address: 192.168.43.1
net mask  : 255.255.255.0
dns server #0: 0.0.0.0
dns server #1: 0.0.0.0
```

ping 一下百度网址

```
msh >ping www.baidu.com
32 bytes from 36.152.44.95 icmp_seq=0 time=51 ms
32 bytes from 36.152.44.95 icmp_seq=1 time=25 ms
32 bytes from 36.152.44.95 icmp_seq=2 time=28 ms
32 bytes from 36.152.44.95 icmp_seq=3 time=31 ms
msh >
```

一切完成

4. 章节总结

本章节我们使用 RTT Studio 配置 at 软件包来连接 wifi 模块，只需要几个步骤就可以配置完串口和软件包，开启 at 例程后，软件包例程自动把 8266 初始化放到系统 APP 初始化里面了，软件包默认添加了几个 Fish 命令到命令行里面，方便我们快速使用 8266 检测功能，如果需要更多功能的话则需要自己编写程序，调用 at 软件包的接口完成功能

项目 1：基于 AB32VG1 的智慧门禁系统

1 前言说明

1.1 本章内容

本章是基于 AB32VG1 的 DIY 作品之一——智慧门禁系统，使用 RC522 射频模块来读取 IC 卡卡号，卡号分为管理员卡和普通卡，管理员卡无视时间限制通行，普通卡在合理时间内能够通行，通过对比 SD 卡中存储的卡号信息来验证卡号是否合法，然后在 OLED 屏上显示时间和卡号验证信息，能够在 finish 终端对门禁系统进行管理。

2 模块介绍

2.1 硬件组成

LED 灯 ---------- 红灯表示卡号非法，绿灯表示卡号合法

RC522 模块 ---------- 用来读取 IC 卡卡号

OLED 屏 ---------- 显示时间和卡号验证信息

UART0 串口 ---------- finish 终端，可以输入命令查看卡号记录等

SD 卡 ----------存储卡号，日志存储

RC522 模块使用软件模拟 SPI 与 AB32VG1 进行通信，使用引脚：

SDA ---------- PF0

SCLK ---------- PE0

MOSI ---------- PF1

MISO ---------- PA5

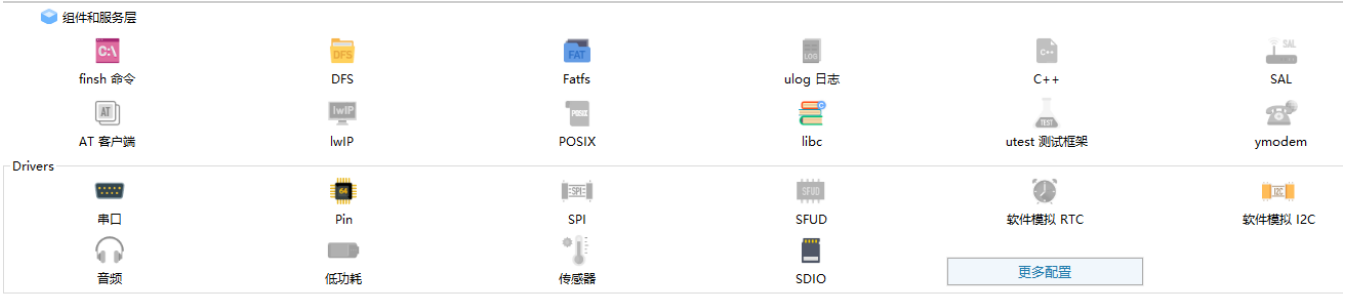
OLED 屏使用软件模拟 I2C 与 AB32VG1 进行通行，使用引脚：

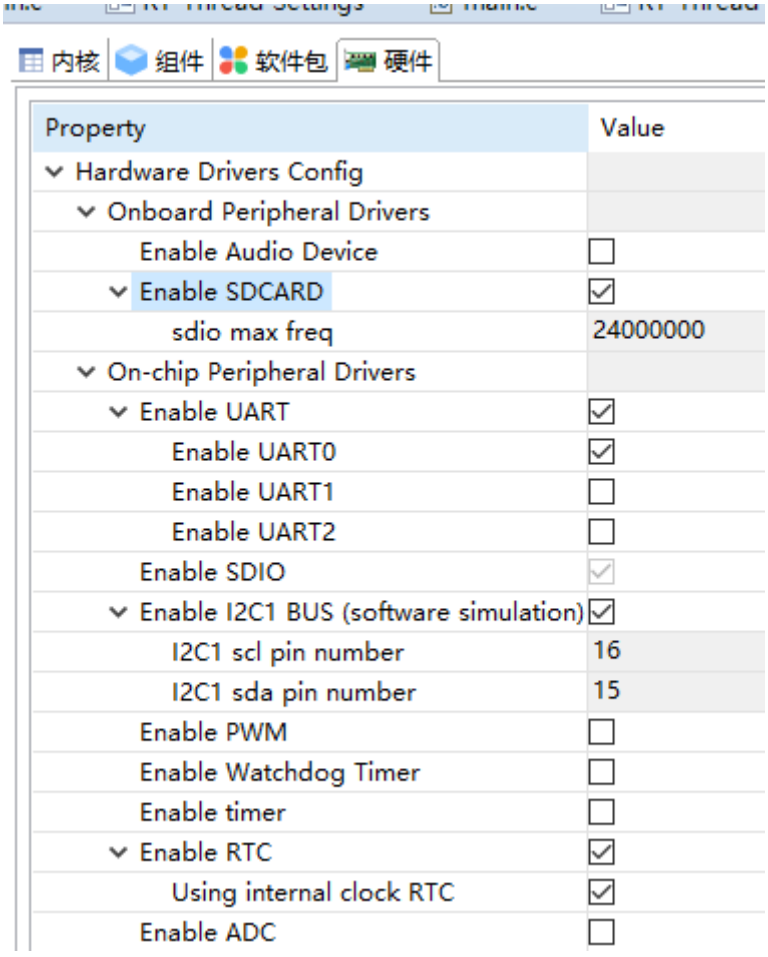
SDA ---------- PE2

SCL ---------- PE3

2.2 软件设计

软件设计中主要修改了 RC522 驱动以及 SSD1306 驱动，其中 RC522 驱动是裸机里的驱动，SSD1306 驱动以及其他驱动开关如下图：





整个程序流程主要分为以下几个步骤：

1.UID 卡号读取，读取卡号，将卡号通过邮箱发送给 UID 处理线程；

2.UID 卡号处理，判断卡号与 SD 卡中存储的卡号是否符合，符合的话 OLED 屏幕显示 YES 并 3 且亮绿灯，不符合显示 NO 并亮红灯；

3.OLED 屏幕显示当前时间；

4.Finish 终端可以进行门禁系统的管理，一共有 10 个命令来进行管理，具体命令下节介绍；

程序中创建了四个线程：UID 卡号读取线程、UID 处理线程、OLED 显示线程和 UID 卡号验证信息线程，详细程序请看源程序。本门禁管理系统，在 SD 卡中存放卡号信息，刷卡记录以及通行时间，一共四个文件，如下图：

在 Finish 终端中设计了 10 个命令来进行门禁系统的管理：

1.date----------设置时间以及显示时间

例：date 2021 05 11 23 33 30

2.search_is_common_card----------查询卡号是否普通卡

例：search_is_common_card 777ed460

3.search_is_manage_card----------查询卡号是否为管理员卡

例：search_is_manage_card 777ed460

4.read_opentime----------读取通行时间

例：read_opentime 08001600

5.set_opentime----------设置通行时间

例：set_opentime 08001600

6.delete_manage_card----------删除管理员卡号

例：delete_manage_card 777ed460

7.delete_common_card----------删除普通卡

例：delete_common_card 777ed460

8.add_common_card----------增加普通卡

例：add_common_card 777ed460

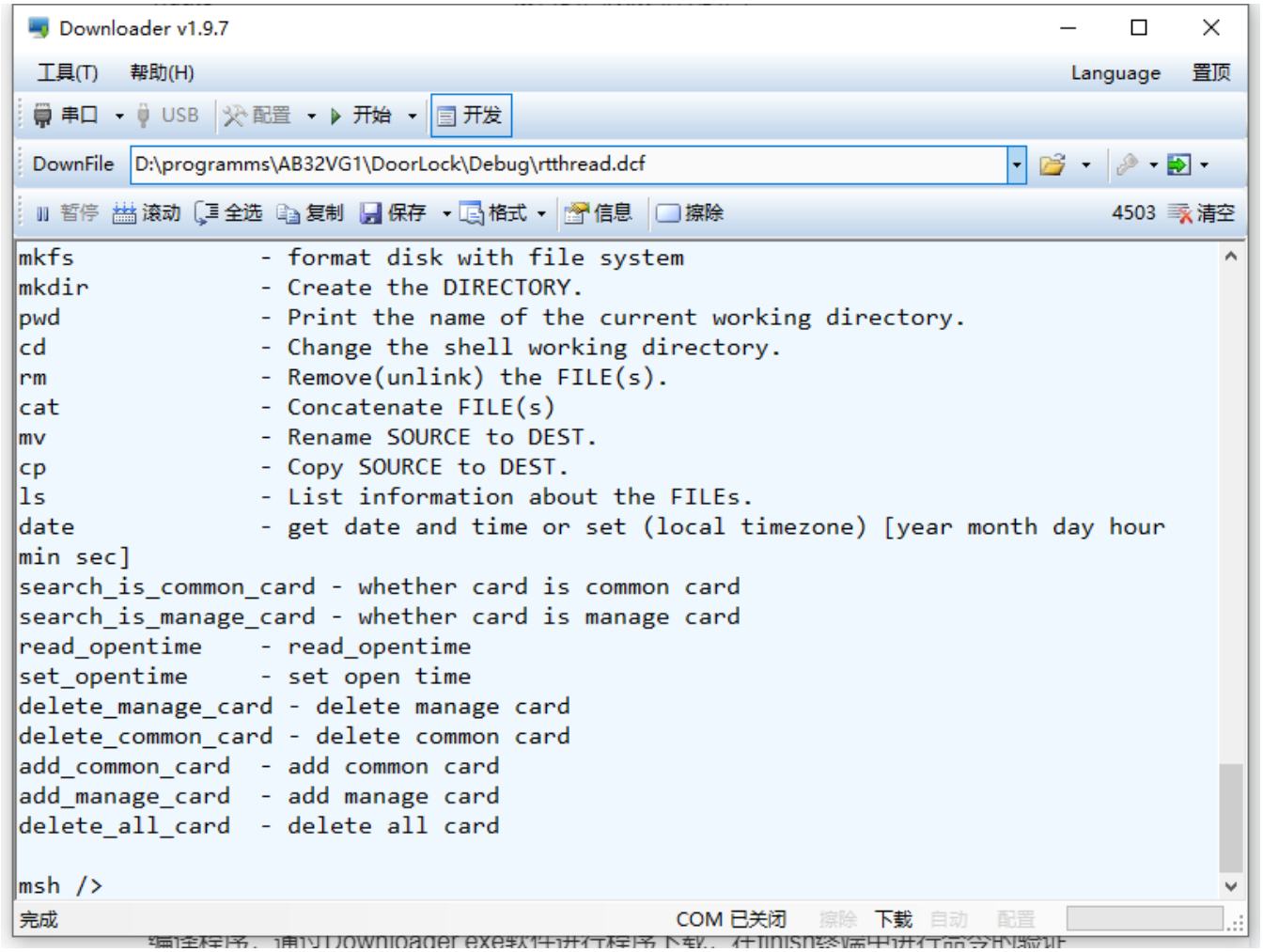
9.add_manage_card----------增加管理员卡

例：add_manage_card 777ed460

delete_all_card----------删除所有卡号

3　代码验证

编译程序，通过 Downloader.exe 软件进行程序下载，当刷卡合法、非法 LED 灯以及 OLED 显示屏均会有提示，功能已验证，此处不方便贴图，可自行尝试。在 finish 终端中进行命令的验证结果如下：



- finsh 终端可添加权限，输对密码才能正常使用终端

```
 \ | /
- RT -     Thread Operating System
 / | \     4.0.3 build Jun  9 2021
 2006 - 2021 Copyright by rt-thread team
Password for login:
RC522_Init Success
[I/SDIO] SD card capacity 15159296 KB.
found part[0], begin: 4194304, size: 14.464GB

Sorry, try again.
Password for login: *******
Sorry, try again.
Password for login: ******
msh />
```

- 查询卡号是否为管理员卡号

```
msh />add_manage_card 777ed460
Add Manage Card Success!
msh />
msh />
search_is_common_card
search_is_manage_card
msh />
search_is_manage_card
msh />search_is_manage_card 777ed460
This Card is Manage Card!
msh />
delete_manage_card
delete_common_card
delete_all_card
msh />
delete_manage_card
msh />delete_manage_card 777ed460
Delete Manage Card Success!
msh />search_is_manage_card 777ed460
This Card is not Manage Card!
msh />
```

- 查询卡号是否为普通卡

```
msh />search_is_common_card 319ef142
This Card is Common Card!
msh />
delete_manage_card
delete_common_card
delete_all_card
msh />
delete_common_card
msh />delete_common_card 319ef142
Delete Common Card Success!
msh />
search_is_common_card
search_is_manage_card
msh />
search_is_common_card
msh />search_is_common_card 319ef142
This Card is not Common Card!
msh />
```

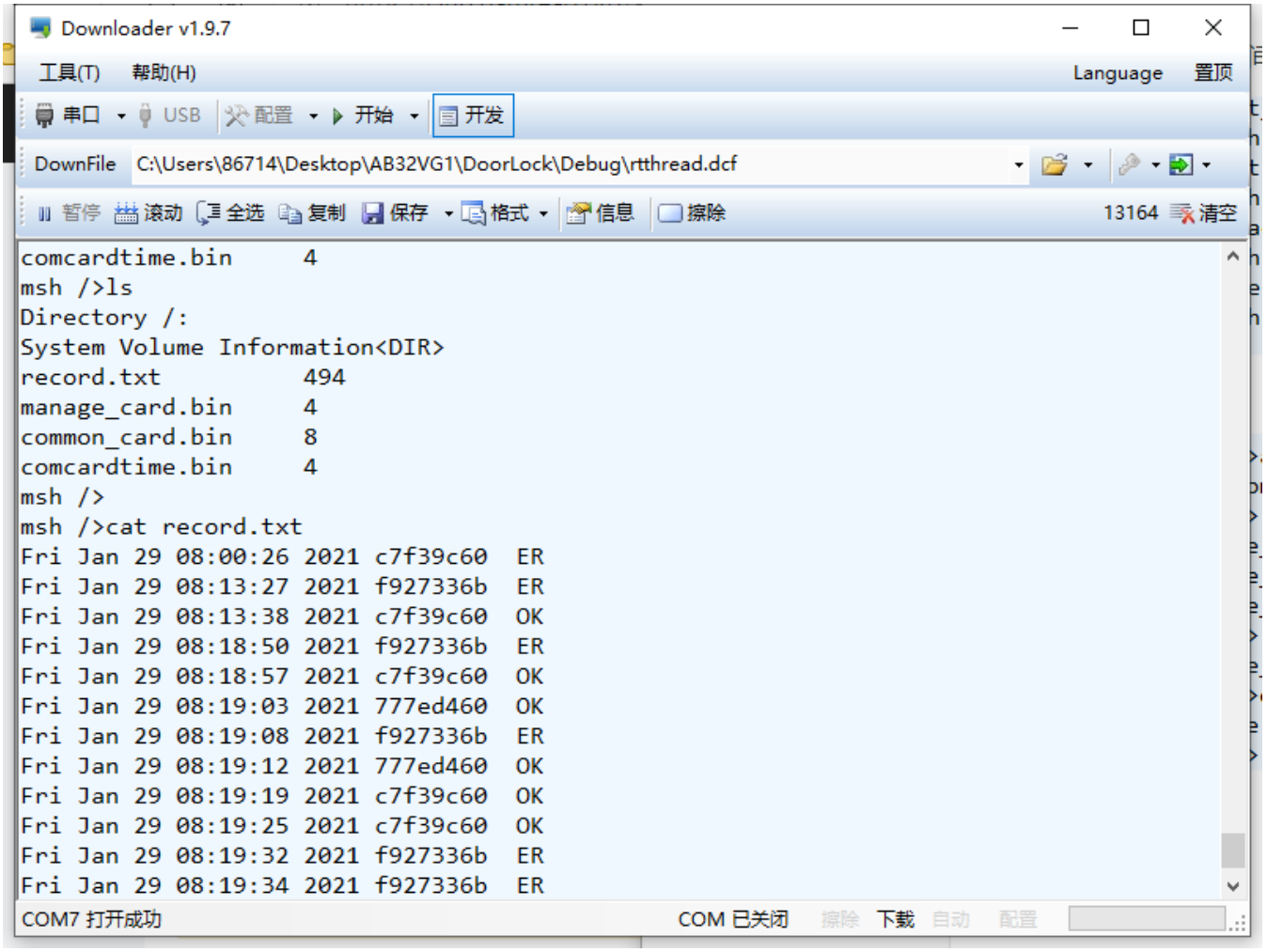
- 设置、读取开门时间

```
set_opentime
msh />set_opentime 08001600
Set Open Time Success!
msh />
read_opentime
msh />read_opentime
Opentime:08:00-22:00
msh />
```

- 增加、删除普通卡

```
msh />add_common_card c7f39c60
Add Common Card Success!
msh />
delete_manage_card
delete_common_card
delete_all_card
msh />
delete_common_card
msh />delete_common_card 777ed460
Delete Common Card Fail!
msh />
```

- 读取刷卡记录



- 设置、读取时间

```
msh />date
Fri Jan 29 08:13:24 2021 msh />
msh />date 2021 06 09 22 02 00
msh />date
Wed Jun  9 22:02:03 2021 msh />
```

4 章节总结

使用 AB32VG1 开发板，再搭配几个模块就能够完成一个 DIY 项目很有成就感。在 RT-Thread Studio 上开发项目搭配上操作系统十分方便，对底层不懂也能够快速上手，将代码迅速跑起来非常不错。

5 附录代码及视频

代码链接

视频链接

项目 2：基于 AB32VG1 的遥控台灯

1. 前言说明

本章主要为通过 RT-Thread Studio 配置 AB32VG1 代码，实现接收手机通过蓝牙模块发送的命令，用 PWM 功能驱动 MOSFET 控制流经 LED 灯条的电流从而达到控制灯条亮度的目的。

2. 模块介绍

本章使用的模块如下：

AB32VG1 开发板

HC-05 蓝牙模块

MOSFET 模块

3. 开发软件

开发环境：RT-Thread Studio

下载工具：Downloader.exe

4.步骤说明

4.1 新建工程

1. 打开 RT-Thread Studio 软件；
2. 点击"文件"->"新键"->"RT-Thread 项目"；

3. 填写项目名称,选择基于开发板；

4. 选择开发板，选择 BSP 后点击完成,即完成新建项目。

4.2 编写代码

1. 编写代码 pwm.c 如下：(此处代码参考自 https://club.rt-thread.org/ask/article/2636.html 进行部分代码添加修改等)

```
#include <rtthread.h>
#include <rtdevice.h>
#define PWM_DEV_NAME        "t5pwm"   /* PWM 设备名称 */
#define PWM_DEV_CHANNEL     1         /* PWM 通道 */
struct rt_device_pwm *pwm_dev;        /* PWM 设备句柄 */
rt_uint32_t pulse=0;
rt_uint32_t liangdu[]={188,0,64,84,106};
ALIGN(RT_ALIGN_SIZE)
static uint8_t PWM_Thread_Stack[1024];
static void PWM_Thread_Entry(void *para);
static struct rt_thread pwm_thread;
rt_uint32_t period, pulse;
void Pwm_Init(void){
    period = 200;      /* 周期 */
    pulse = 0;         /* PWM 脉冲宽度值(0 - 1000) */
    pwm_dev = (struct rt_device_pwm *)rt_device_find(PWM_DEV_NAME);
    RT_ASSERT(pwm_dev != RT_NULL);
    /* 设置 PWM 周期和脉冲宽度 */
    rt_pwm_set(pwm_dev, PWM_DEV_CHANNEL, period, 188);
    /* 使能设备 */
    rt_pwm_enable(pwm_dev, PWM_DEV_CHANNEL);
}

static void PWM_Thread_Entry(void *para){
    Pwm_Init();
    while(1)
    {
        rt_thread_mdelay(1000);
    }
}

int Pwm_Thread_Init(void){

    rt_thread_init(&pwm_thread, "pwm_thread", PWM_Thread_Entry, RT_NULL,
            &PWM_Thread_Stack[0], sizeof(PWM_Thread_Stack), 10, 10);
```

```
    rt_thread_startup(&pwm_thread);
    return 0;
}
INIT_APP_EXPORT(Pwm_Thread_Init);
int ledadd(void)
{
    if(pulse == 4)
    {
        pulse = 4;
    }
    else pulse ++;
    pwm_dev = (struct rt_device_pwm *)rt_device_find(PWM_DEV_NAME);
    rt_pwm_set(pwm_dev, PWM_DEV_CHANNEL, 200, liangdu[pulse]);rt_kprintf("now is %d\n",pulse);
    rt_pwm_enable(pwm_dev, PWM_DEV_CHANNEL);
}
MSH_CMD_EXPORT(ledadd, ledadd sample);
int ledabate(void)
{
    if(pulse == 0)
    {
        pulse =0;
    }
    else pulse --;
    pwm_dev = (struct rt_device_pwm *)rt_device_find(PWM_DEV_NAME);
    rt_pwm_set(pwm_dev, PWM_DEV_CHANNEL, 200, liangdu[pulse]);rt_kprintf("now is %d\n",pulse);
    rt_pwm_enable(pwm_dev, PWM_DEV_CHANNEL);
}
MSH_CMD_EXPORT(ledabate, ledadd sample);
int ledoff(void)
{
    pulse = 0;
    pwm_dev = (struct rt_device_pwm *)rt_device_find(PWM_DEV_NAME);
    rt_pwm_set(pwm_dev, PWM_DEV_CHANNEL, 200, liangdu[pulse]);
    rt_pwm_enable(pwm_dev, PWM_DEV_CHANNEL);
}
MSH_CMD_EXPORT(ledoff, ledoff sample);
int ledon(void)
{
    pulse = 4;
    pwm_dev = (struct rt_device_pwm *)rt_device_find(PWM_DEV_NAME);
    rt_pwm_set(pwm_dev, PWM_DEV_CHANNEL, 200, liangdu[pulse]);
    rt_pwm_enable(pwm_dev, PWM_DEV_CHANNEL);
}
MSH_CMD_EXPORT(ledon, ledoff sample);
```

2. 编写代码 pwm.h 如下：(此处代码完全参考自 https://club.rt-thread.org/ask/article/2636.html)

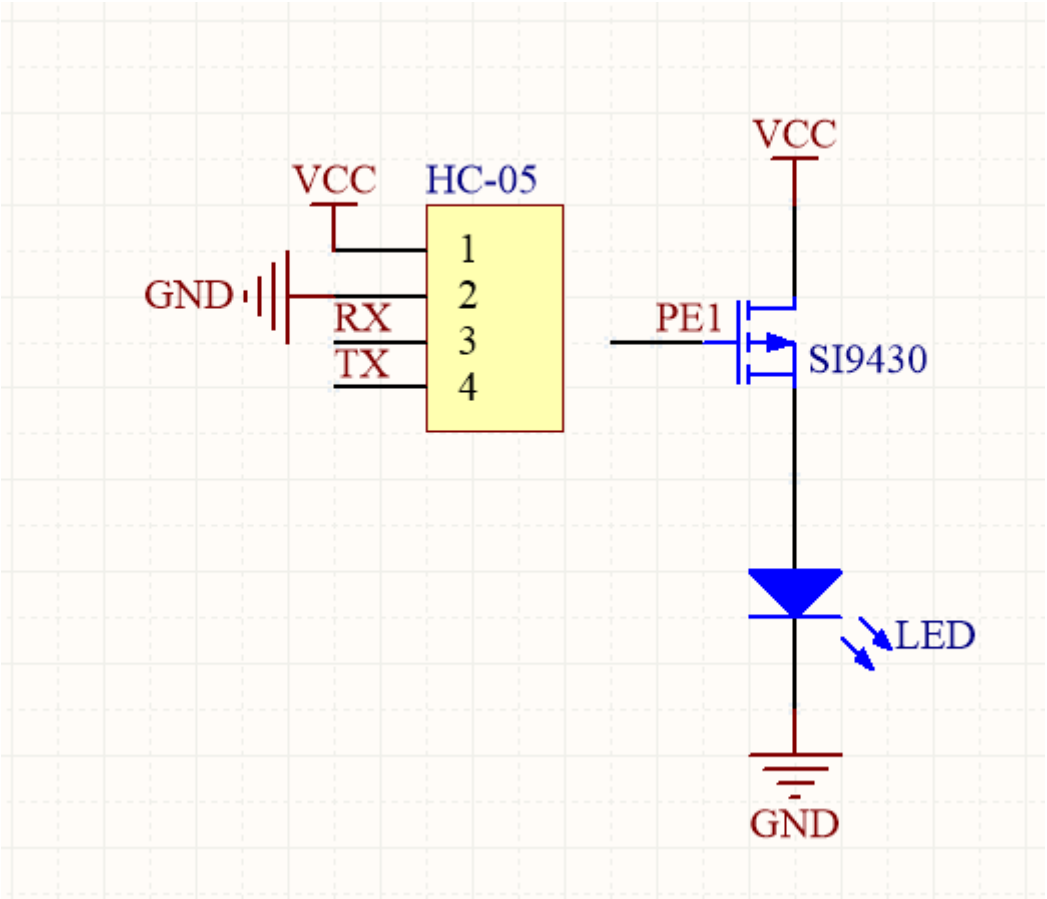
```
#ifndef APPLICATIONS_PWM_H_
#define APPLICATIONS_PWM_H_

#ifdef BSP_USING_T3_PWM     // timer3  的 pwm 发生器
#ifndef T3_PWM_CONFIG
#define T3_PWM_CONFIG                                       \
    {                                                       \
        .pwm_handle             = TIM3_BASE,        \
        .name                   = "t3pwm",           \
        .channel                = 0                 \
    }
#endif /* T3_PWM_CONFIG */
#endif /* BSP_USING_T3_PWM */

#ifdef BSP_USING_T4_PWM    // timer4  的 pwm 发生器
#ifndef T4_PWM_CONFIG
#define T4_PWM_CONFIG                                       \
    {                                                       \
        .pwm_handle             = TIM4_BASE,        \
        .name                   = "t4pwm",           \
        .channel                = 0                 \
    }
#endif /* T4_PWM_CONFIG */
#endif /* BSP_USING_T4_PWM */

#ifdef BSP_USING_T5_PWM     // timer5  的 pwm 发生器
#ifndef T5_PWM_CONFIG
#define T5_PWM_CONFIG                                       \
    {                                                       \
        .pwm_handle             = TIM5_BASE,        \
        .name                   = "t5pwm",           \
        .channel                = 0                 \
    }
#endif /* T5_PWM_CONFIG */
#endif /* BSP_USING_T5_PWM */

#ifdef BSP_USING_LPWM0     // pwm  特殊发生器 0
#ifndef LPWM0_CONFIG
#define LPWM0_CONFIG                                        \
    {                                                       \
```

```
        .pwm_handle             = PWM_BASE,         \
        .name                   = "lpwm0",          \
        .channel                = 0                 \
    }
#endif /* LPWM0_CONFIG */
#endif /* BSP_USING_LPWM0 */

#ifdef BSP_USING_LPWM1     // pwm  特殊发生器 1
#ifndef LPWM1_CONFIG
#define LPWM1_CONFIG                                \
    {                                               \
        .pwm_handle             = PWM_BASE,         \
        .name                   = "lpwm1",          \
        .channel                = 0                 \
    }
#endif /* LPWM1_CONFIG */
#endif /* BSP_USING_LPWM1 */

#ifdef BSP_USING_LPWM2    // pwm  特殊发生器 2
#ifndef LPWM2_CONFIG
#define LPWM2_CONFIG                                \
    {                                               \
        .pwm_handle             = PWM_BASE,         \
        .name                   = "lpwm2",          \
        .channel                = 0                 \
    }
#endif /* LPWM2_CONFIG */
#endif /* BSP_USING_LPWM2 */

#ifdef BSP_USING_LPWM3    // pwm  特殊发生器 3
#ifndef LPWM3_CONFIG
#define LPWM3_CONFIG                                \
    {                                               \
        .pwm_handle             = PWM_BASE,         \
        .name                   = "lpwm3",          \
        .channel                = 0                 \
    }
#endif /* LPWM3_CONFIG */
#endif /* BSP_USING_LPWM3 */
static void pwm_get_channel(void)
{
#ifdef BSP_USING_T3_PWM0     // timer3 pwm 0 通道   -》  PB0
    ab32_pwm_obj[T3_PWM_INDEX].channel |= 1 << 0;
#endif
#ifdef BSP_USING_T3_PWM1     // timer3 pwm 1 通道
```

```
    ab32_pwm_obj[T3_PWM_INDEX].channel |= 1 << 1;
#endif
#ifdef BSP_USING_T3_PWM2    // timer3 pwm 2 通道
    ab32_pwm_obj[T3_PWM_INDEX].channel |= 1 << 2;
#endif
#ifdef BSP_USING_T4_PWM0    // timer4 pwm 0 通道
    ab32_pwm_obj[T4_PWM_INDEX].channel |= 1 << 0;
#endif
#ifdef BSP_USING_T4_PWM1    // timer4 pwm 1 通道 -》 PA6
    ab32_pwm_obj[T4_PWM_INDEX].channel |= 1 << 1;
#endif
#ifdef BSP_USING_T4_PWM2    // timer4 pwm 2 通道
    ab32_pwm_obj[T4_PWM_INDEX].channel |= 1 << 2;
#endif
#ifdef BSP_USING_T5_PWM0    // timer5 pwm 0 通道  -》 PE1
    ab32_pwm_obj[T5_PWM_INDEX].channel |= 1 << 0;
#endif
#ifdef BSP_USING_T5_PWM1    // timer5 pwm 1 通道
    ab32_pwm_obj[T5_PWM_INDEX].channel |= 1 << 1;
#endif
#ifdef BSP_USING_T5_PWM2    // timer5 pwm 2 通道
    ab32_pwm_obj[T5_PWM_INDEX].channel |= 1 << 2;
#endif
#ifdef BSP_USING_LPWM0      // lpwm0 pwm 0 通道 ——》 PE4(G1)   G1,G2.G3 之间相互互斥
    ab32_pwm_obj[LPWM0_INDEX].channel |= 1 << 0;
#endif
#ifdef BSP_USING_LPWM1      // lpwm1 pwm 0 通道， -》 PA1(G3)
    ab32_pwm_obj[LPWM1_INDEX].channel |= 1 << 0;
#endif
#ifdef BSP_USING_LPWM2      // lpwm2 pwm 0 通道，-》PE0(G2)/PA2(G3)
    ab32_pwm_obj[LPWM2_INDEX].channel |= 1 << 0;
#endif
#ifdef BSP_USING_LPWM3      // lpwm3 pwm 0 通道
    ab32_pwm_obj[LPWM3_INDEX].channel |= 1 << 0;
#endif
}
#endif /* APPLICATIONS_PWM_H_ */
```

4.3 硬件电路

电路比较简单如下图所示：



5.代码验证

5.1 模拟测试

代码编写后先用串口助手分别发送如下命令测试：

ledadd	亮度增加
ledabate	亮度减小
ledoff	关闭
ledon	点亮

5.2 项目联调

串口验证通过后将整个系统搭建起来进行测试，演示视频 https://v.douyin.com/ePp8jpU/

6.章节总结

开发板本身有蓝牙功能但是由于并未开放所以只能再挂一个蓝牙去做，由于近期事情很多仓促间制作有些粗糙，制作效果并不是特别满意，亮度等级分的不是很多，只选取了比明显的几个等级好在基本现了功能。

项目 3：基于 AB32 的智能灯控

1. 前言说明

本章主要为通过 RT-Thread Studio 配置 AB32VG1 代码，实现接收手机通过蓝牙模块发送的命令，用 PWM 功能驱动舵机控制 86 开关从而达到控制灯管开关的目的。

2. 模块介绍

本章使用的模块如下：

AB32VG1 开发板

JDY-31 蓝牙模块

舵机模块

3. 开发软件

开发环境：RT-Thread Studio

下载工具：Downloader.exe

4.步骤说明

4.1 新建工程

5. 打开 RT-Thread Studio 软件；
6. 点击"文件"->"新键"->"RT-Thread 项目"；
7. 填写项目名称,选择基于开发板；
8. 选择开发板，选择 BSP 后点击完成,即完成新建项目。

4.2 编写代码

3. 编写代码 pwm.c 如下：(此处代码参考自 https://club.rt-thread.org/ask/article/2620.html 进行部分代码添加修改等)

```
#include "pwm_task.h"

struct rt_device_pwm *pwm1_dev;          /* PWM 设备句柄 */
struct rt_device_pwm *pwm2_dev;          /* PWM 设备句柄 */
struct rt_semaphore pwm_sem;/* 用于 PWM 消息的信号量 */

bool PWM;/* PWM 输出状态 */

static void pwm_task_entry(void *parameter)
{
      while (1)
      {
//            rt_pwm_disable(pwm1_dev, PWM1_DEV_CHANNEL);   // 关闭 PWM
//            rt_pwm_disable(pwm2_dev, PWM2_DEV_CHANNEL);   // 关闭 PWM
            /* 阻塞等待接收信号量，等到信号量后再次读取数据 */
            rt_sem_take(&pwm_sem, RT_WAITING_FOREVER);

            if(PWM == 1){   // 开灯
                  rt_pwm_enable(pwm1_dev, PWM1_DEV_CHANNEL);
                  rt_pwm_set(pwm1_dev,    PWM1_DEV_CHANNEL,    10000000,    10000000-2000000);//
10000000-2500000 180°
                  rt_thread_mdelay(700);
                  rt_pwm_disable(pwm1_dev, PWM1_DEV_CHANNEL);   // 关闭 PWM
                  rt_thread_mdelay(100);
                  rt_pwm_enable(pwm2_dev, PWM2_DEV_CHANNEL);
```

```
                rt_pwm_set(pwm2_dev, PWM2_DEV_CHANNEL, 10000000, 900000);// 10000000-2500000
180°
                rt_thread_mdelay(700);
                rt_pwm_disable(pwm2_dev, PWM2_DEV_CHANNEL);   // 关闭 PWM
            }
            else if (PWM == 0) {   // 关灯
                rt_pwm_enable(pwm1_dev, PWM1_DEV_CHANNEL);
                rt_pwm_set(pwm1_dev,    PWM1_DEV_CHANNEL,    10000000,    10000000-900000);//
10000000-500000 0°
                rt_thread_mdelay(700);
                rt_pwm_disable(pwm1_dev, PWM1_DEV_CHANNEL);   // 关闭 PWM
                rt_thread_mdelay(100);
                rt_pwm_enable(pwm2_dev, PWM2_DEV_CHANNEL);
                rt_pwm_set(pwm2_dev,   PWM2_DEV_CHANNEL,   10000000,   2000000);//   10000000-
2500000 180°
                rt_thread_mdelay(700);
                rt_pwm_disable(pwm2_dev, PWM2_DEV_CHANNEL);   // 关闭 PWM
            }
        }
}
/**
  * @brief    thread_sg90
  * @param    None
  * @retval ret
  */
int pwm_task(void)
{
      rt_err_t ret = RT_EOK;
      /* 查找设备 */
      rt_uint32_t period, pulse,t;

    period = 10000000;     /* 周期为 10ms，单位为纳秒 ns */
    pulse = 0;                 /* PWM 脉冲宽度值，单位为纳秒 ns */


    pwm1_dev = (struct rt_device_pwm *)rt_device_find(PWM1_DEV_NAME);   /* 查找设备 */
    /* 查看设备开启状况 */
     if (pwm1_dev == RT_NULL)
     {
          rt_kprintf("steering gear control run failed! can't find %s device!\n", PWM1_DEV_NAME);
          return RT_ERROR;
     }

     pwm2_dev = (struct rt_device_pwm *)rt_device_find(PWM2_DEV_NAME);   /* 查找设备 */
     /* 查看设备开启状况 */
```

```
        if (pwm2_dev == RT_NULL)
        {
            rt_kprintf("steering gear control run failed! can't find %s device!\n", PWM2_DEV_NAME);
            return RT_ERROR;
        }
//      rt_pwm_enable(pwm1_dev, PWM1_DEV_CHANNEL);   // 关闭 PWM
//      rt_pwm_enable(pwm2_dev, PWM2_DEV_CHANNEL);   // 关闭 PWM
//        rt_pwm_set(pwm1_dev, PWM1_DEV_CHANNEL, 10000000, 10000000-1500000);// 10000000-2500000 180°
//      rt_thread_mdelay(1000);
//      rt_pwm_set(pwm2_dev, PWM2_DEV_CHANNEL, 10000000, 1500000);// 10000000-2500000 180°
//      rt_thread_mdelay(1000);
    rt_pwm_disable(pwm1_dev, PWM1_DEV_CHANNEL);   // 关闭 PWM
    rt_pwm_disable(pwm2_dev, PWM2_DEV_CHANNEL);   // 关闭 PWM

    /* 初始化信号量 */
    rt_sem_init(&pwm_sem, "pwm_sem", 0, RT_IPC_FLAG_FIFO);

    /* 创建 task 线程 */
    rt_thread_t thread = rt_thread_create("pwm_task", pwm_task_entry, RT_NULL, 512, 25, 5);
    /* 创建成功则启动线程 */
    if (thread != RT_NULL)
    {
        rt_thread_startup(thread);
    }
    else
    {
        ret = RT_ERROR;
    }

    return ret;
}
```

4. 编写代码 pwm.h 如下：

```
/*
 * Copyright (c) 2006-2021, RT-Thread Development Team
 *
 * SPDX-License-Identifier: Apache-2.0
 *
 * Change Logs:
 * Date           Author       Notes
 * 2021-04-22     Administrator      the first version
 */
```

```
#ifndef APPLICATIONS_PWM_TASK_H_
#define APPLICATIONS_PWM_TASK_H_

/* Standard includes. */
#include <stdio.h>

/* rtthread includes. */
#include <rtdevice.h>
#include <board.h>

#define PWM1_DEV_NAME           "lpwm0" /* PWM 设备名称 */
#define PWM1_DEV_CHANNEL        0           /* PWM 通道 */
//#define SG1_PIN_NUM               rt_pin_get("PE.4")   /* LED PIN 脚编号，查看驱动文件 drv_gpio.c 确定 */

#define PWM2_DEV_NAME           "lpwm2" /* PWM 设备名称 */
#define PWM2_DEV_CHANNEL        3           /* PWM 通道 */


extern struct rt_semaphore pwm_sem;/* 用于 PWM 消息的信号量 */
extern bool PWM;

int pwm_task(void);

#endif /* APPLICATIONS_PWM_TASK_H_ */
```

5. 编写代码 usart.c 如下：(此处代码参考自 https://club.rt-thread.org/ask/article/2686.html 进行部分代码添加修改等)

```
/*Includes*************************************************************************/
#include "usart1_task.h"
#include "pwm_task.h"

struct serial_configure config = RT_SERIAL_CONFIG_DEFAULT;  /* 初始化配置参数 */

/* 用于接收消息的信号量 */
static struct rt_semaphore rx_sem;
static rt_device_t serial;      /* 串口设备句柄 */
/**
  * @brief    uart_input //接收数据回调函数
  * @param    dev
  *               size
  * @retval RT_EOK
  */
```

```
static rt_err_t uart_input(rt_device_t dev, rt_size_t size)
{
    /* 串口接收到数据后产生中断，调用此回调函数，然后发送接收信号量 */
    rt_sem_release(&rx_sem);
    return RT_EOK;
}

/**
  * @brief    serial_thread_entry
  * @param    parameter
  * @retval None
  */
static void usart1_task_entry(void *parameter)
{
    char ch;

    while (1)
    {
        /* 从串口读取一个字节的数据，没有读取到则等待接收信号量 */
        while (rt_device_read(serial, -1, &ch, 1) != 1)
        {
            /* 阻塞等待接收信号量，等到信号量后再次读取数据 */
            rt_sem_take(&rx_sem, RT_WAITING_FOREVER);
        }
        /* 读取到的数据做动作 */
        if(ch == 'o' && PWM == 0)   // 开灯动作
        {
            PWM = 1;
            rt_device_write(serial, 0, "open\r\n", 6);
            /* 释放 PWM 信号量 */
            rt_sem_release(&pwm_sem);
        }
        else if(ch == 'c' && PWM == 1)   // 关灯动作
        {
            PWM = 0;
            rt_device_write(serial, 0, "close\r\n", 7);
            /* 释放 PWM 信号量 */
            rt_sem_release(&pwm_sem);
        }
        else if (ch == 'r')   // 请求当前开关状态
        {
            if(PWM == 0)
                rt_device_write(serial, 0, "close\r\n", 7);
            else if (PWM == 1)
                rt_device_write(serial, 0, "open\r\n", 6);
```

```
            }
        }
    }

/**
  * @brief    thread_serial
  * @param    None
  * @retval ret
  */
int usart1_task(void)
{
    rt_err_t ret = RT_EOK;
    char uart_name[RT_NAME_MAX];

    /*串口相关*/
    rt_strncpy(uart_name, SAMPLE_UART_NAME, RT_NAME_MAX);
    /* 查找系统中的串口设备 */
    serial = rt_device_find(uart_name);
    if (!serial)
    {
        rt_kprintf("find %s failed!\n", uart_name);
        return RT_ERROR;
    }
    /* 修改串口配置参数 */
    config.baud_rate = BAUD_RATE_9600;        //修改波特率为 115200
    config.data_bits = DATA_BITS_8;           //数据位 8
    config.stop_bits = STOP_BITS_1;           //停止位 1
    config.bufsz     = 64;                    //修改缓冲区 buff size 为 128
    config.parity    = PARITY_NONE;           //无奇偶校验位

    /* 控制串口设备。通过控制接口传入命令控制字，与控制参数 */
    rt_device_control(serial, RT_DEVICE_CTRL_CONFIG, &config);

    /* 初始化信号量 */
    rt_sem_init(&rx_sem, "rx_sem", 0, RT_IPC_FLAG_FIFO);
    /* 以中断接收及轮询发送模式打开串口设备 */
    rt_device_open(serial, RT_DEVICE_FLAG_INT_RX);
    /* 设置接收回调函数 */
    rt_device_set_rx_indicate(serial, uart_input);
    /* 发送字符串 */
    rt_device_write(serial, 0, "task running \r\n", 16);

    /* 创建 task 线程 */
    rt_thread_t thread = rt_thread_create("usart1_task", usart1_task_entry, RT_NULL, 512, 25, 10);
    /* 创建成功则启动线程 */
```

```
    if (thread != RT_NULL)
    {
        rt_thread_startup(thread);
    }
    else
    {
        ret = RT_ERROR;
    }

    return ret;
}
```

6. 编写代码 usart.h 如下：

```
/*
 * Copyright (c) 2006-2021, RT-Thread Development Team
 *
 * SPDX-License-Identifier: Apache-2.0
 *
 * Change Logs:
 * Date           Author       Notes
 * 2021-04-20     Administrator      the first version
 */
#ifndef APPLICATIONS_USART1_TASK_H_
#define APPLICATIONS_USART1_TASK_H_

/* Standard includes. */
#include <stdio.h>

/* rtthread includes. */
#include <rtdevice.h>
#include <board.h>

#define SAMPLE_UART_NAME       "uart1" /* 串口名称 */

int usart1_task(void);   /* 主线任务函数 */


#endif /* APPLICATIONS_USART1_TASK_H_ */
```

7. 按键控制代码

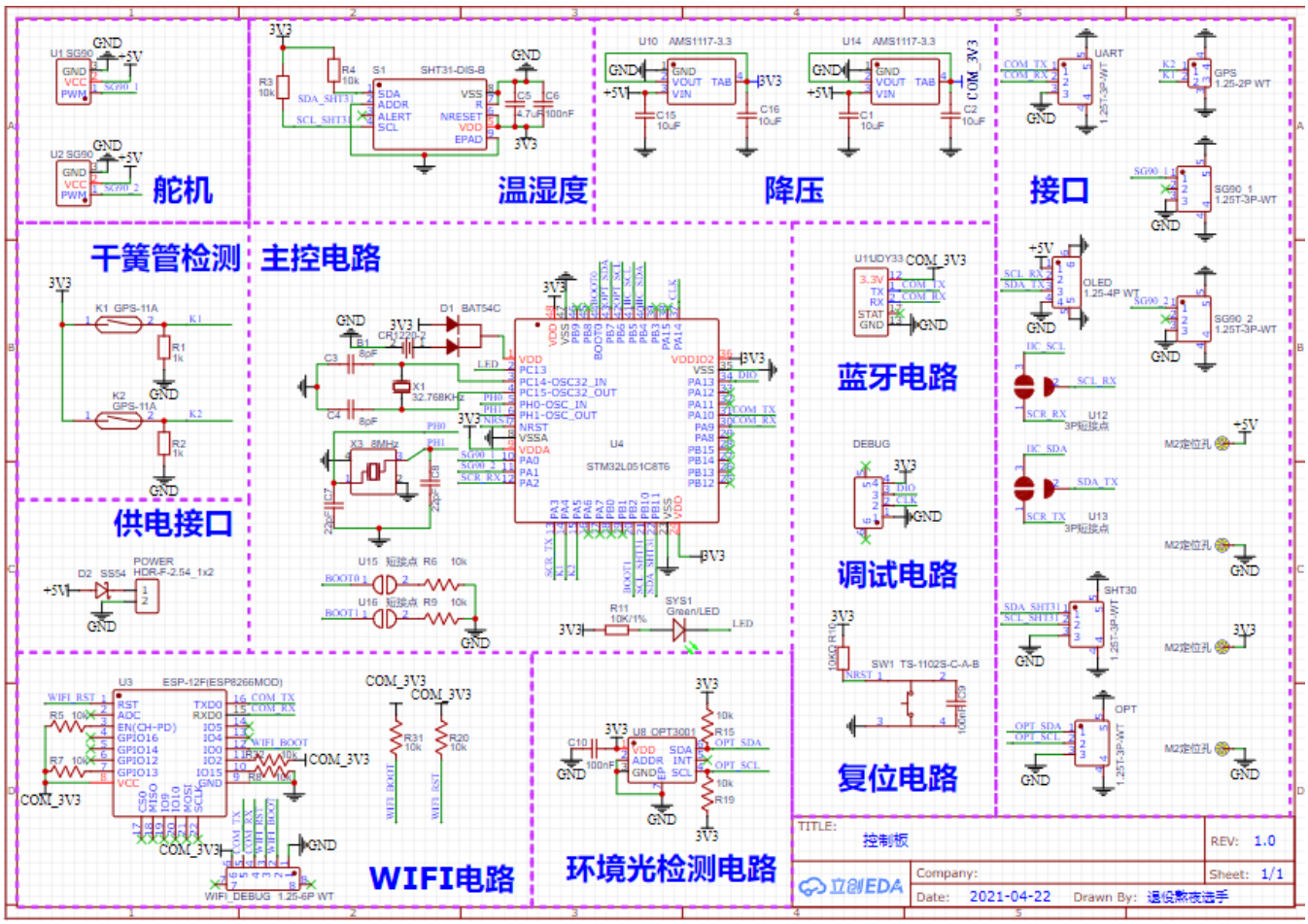
```
#include <rtthread.h>
```

```
#include <usart1_task.h>
#include <pwm_task.h>
#include "board.h"

/* 按键任务 */
int main(void)
{
    uint8_t time = 0;
    uint8_t flag = 0;
    uint8_t key1 = rt_pin_get("PF.1");   //按键 1
    uint8_t key2 = rt_pin_get("PF.1");   //按键 2

    uint8_t pin_b = rt_pin_get("PA.2");   // 蓝灯

    rt_pin_mode(key1, PIN_MODE_INPUT_PULLUP);
    rt_pin_mode(key2, PIN_MODE_INPUT_PULLUP);

    rt_pin_mode(pin_b, PIN_MODE_OUTPUT);

    usart1_task();   // 启动 串口 任务
    pwm_task();   // 启动 舵机 任务
    while (1)
    {
        if (!rt_pin_read(key1)) {
            rt_thread_mdelay(30);
            if (!rt_pin_read(key1)){
                while(!rt_pin_read(key1));
                PWM = 1 - PWM;
                rt_sem_release(&pwm_sem);
            }
        }
        rt_thread_mdelay(10);
        time++;
        if(time >= 100)
        {
            flag = 1-flag;
            if(flag == 0)
                rt_pin_write(pin_b, flag);
            else
                rt_pin_write(pin_b, flag);
            time = 0;
        }

    }
```

```
    return 0;
}
```

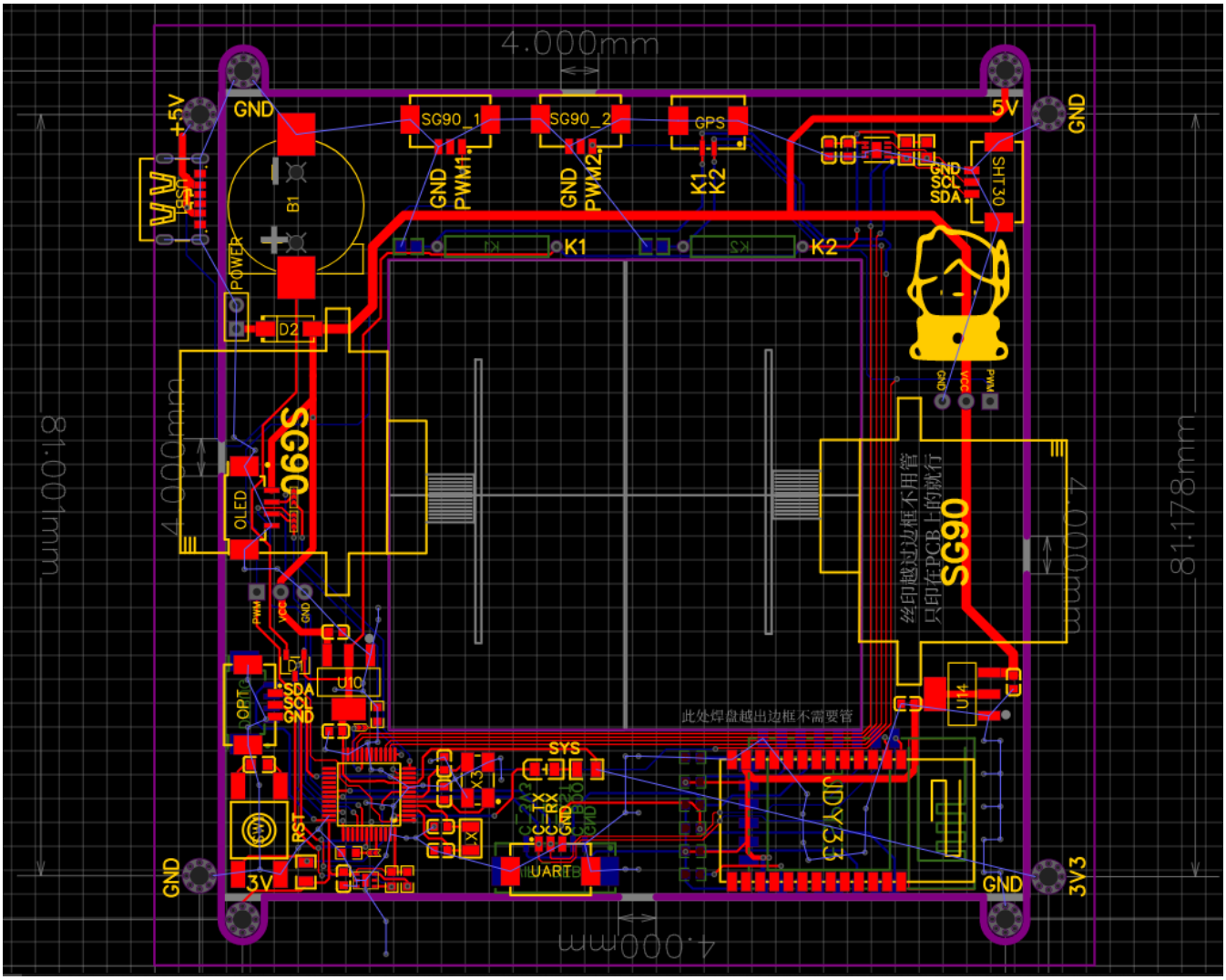
4.3 硬件电路

1.原理图电路比较简单如下图所示：

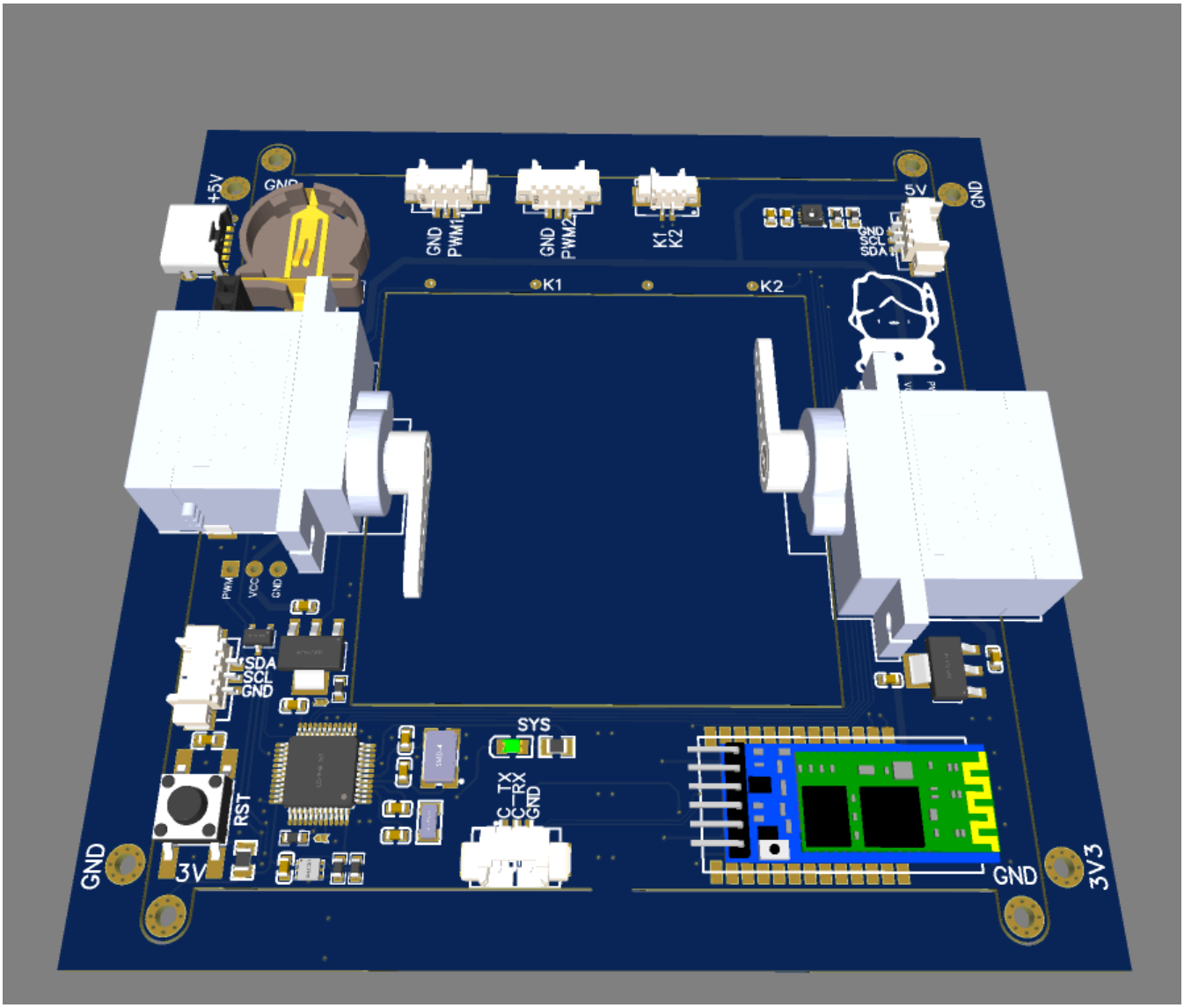


有大量的冗余设计，在此次项目中没有用到。

2.PCB 布局


4.000mm
81.001mm
81.178mm
4.000mm
+5V
GND
SG90_1
SG90_2
GPS
5V
GND
SHT30
K1
K2
POWER
D2
OLED
SG90
SG90
此处焊盘越出边框不需要管
只印在PCB上的就行
JDY23
SYS
UART
RST
3V
GND
3V3
GND

3.3D 模型



5.代码验证

5.1 模拟测试

代码编写后先用串口助手分别发送如下命令测试：

r	返回当前灯控开关状态
o	开灯
c	关灯

5.2 项目联调

串口验证通过后将整个系统搭建起来进行测试，演示视频

https://www.bilibili.com/video/BV1q54y1V7eC/

6.章节总结

通过这次 RT-Thread 的活动，基本掌握了一点 RT-Thread 物联网操作系统的用法，非常感谢官方能给这次实操 RT-Thread 的机会，同时也感受到 RT-Thread 的“美”，多任务运行起来还是非常给力的。

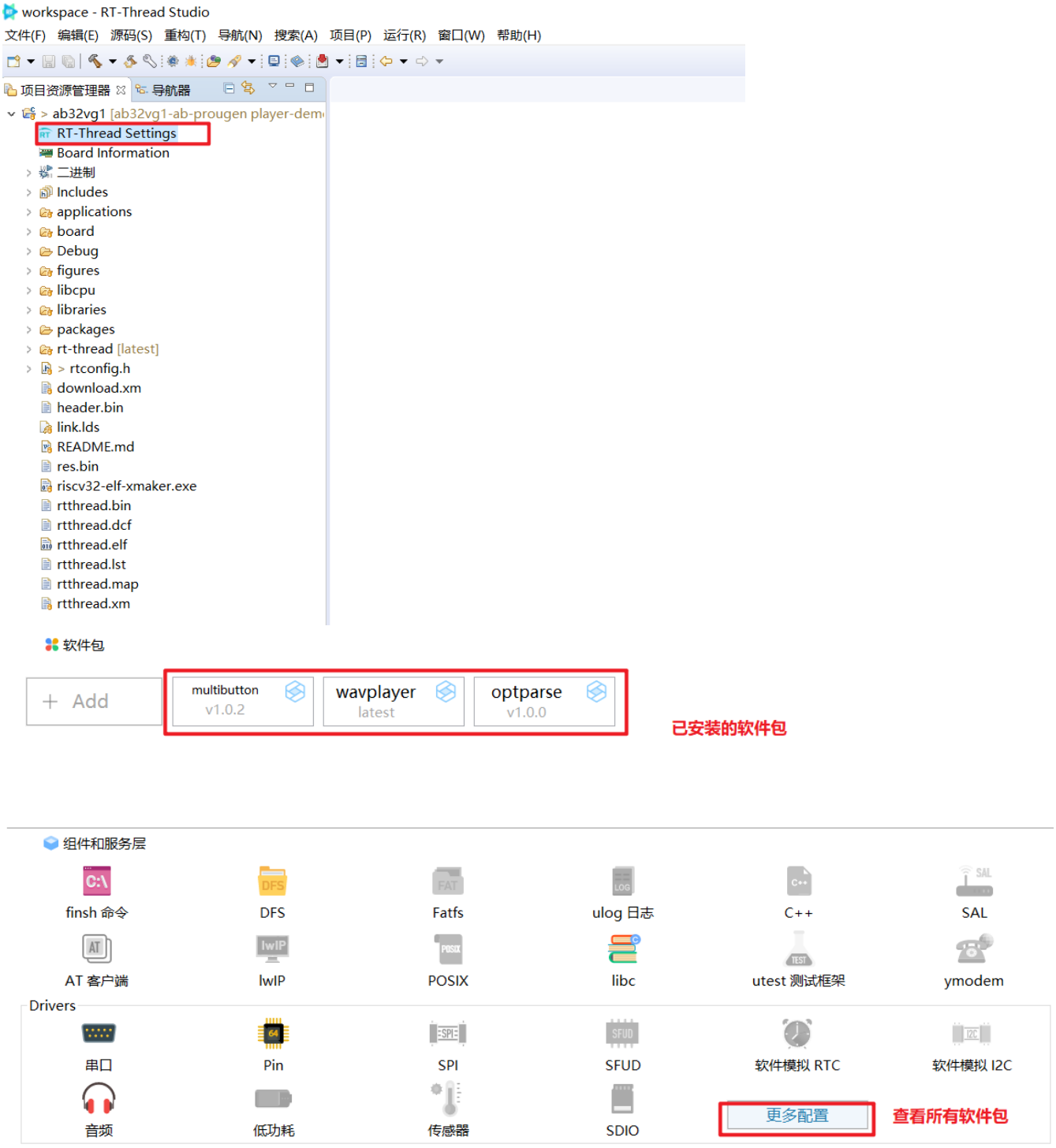
这次活动中调试 AB32 也遇到了一些 PWM 方面问题，也一并在此记录下，主要是“lpwm0”和“lpwm3”在一个关闭另一个启动的情况下(具体在 PWM 任务中)，极性是相反的，这部分是通过示波器来看到的，还有就是高级定时器输出的频率最低大概在 800HZ。

项目 4：WAV 音频播放

软件包安装

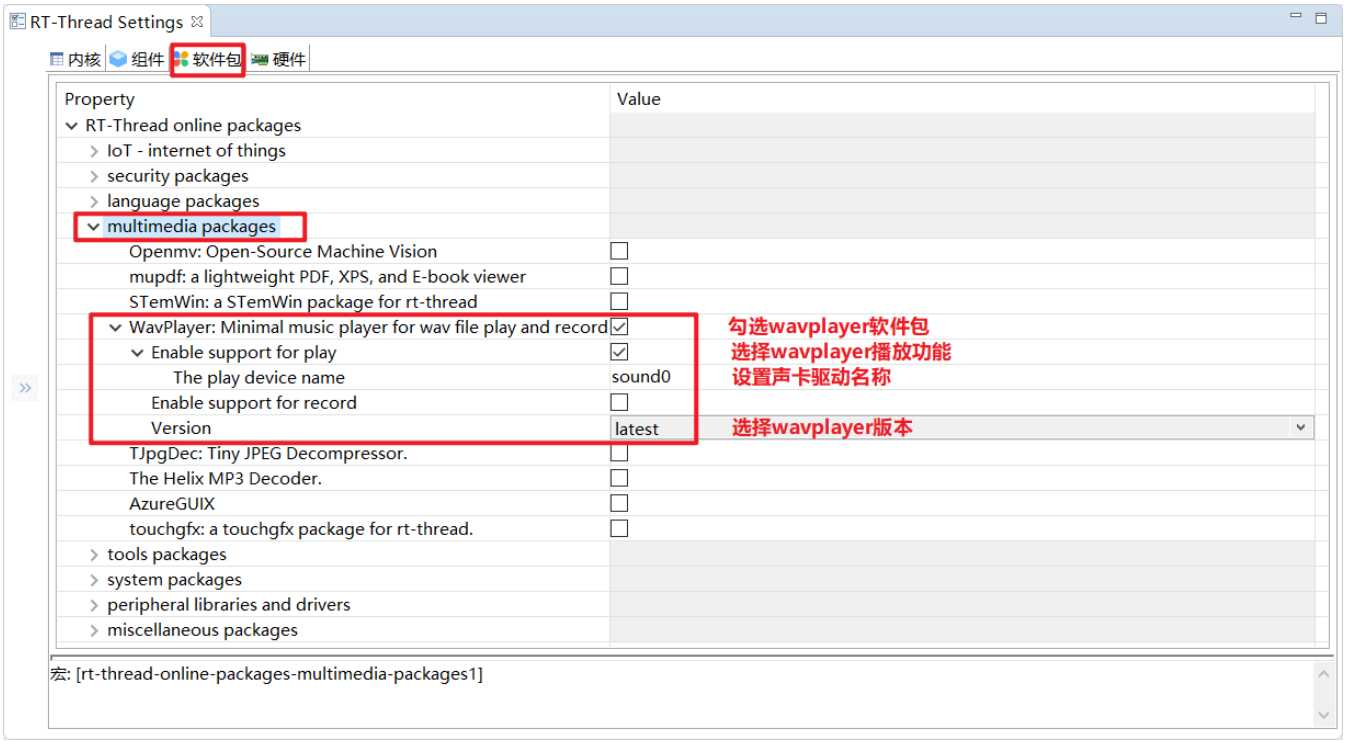
本次实验实现音乐播放功能，单击按键进行音乐切换。需要安装的软件包有 wavplayer/optparse/multibutton 三个软件包。其中 optparse 在 wavplayer 勾选后，自动选择。

进入软件包选择界面.

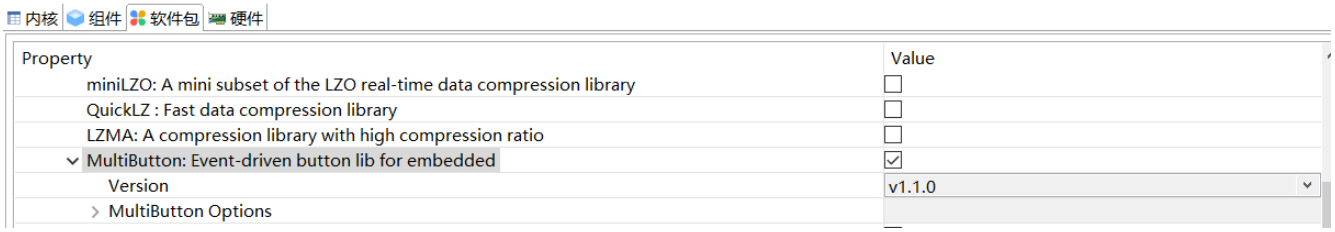


也可以通过`更多配置`查看所有软件包来选择个软件包：

wavplayer 软件包安装



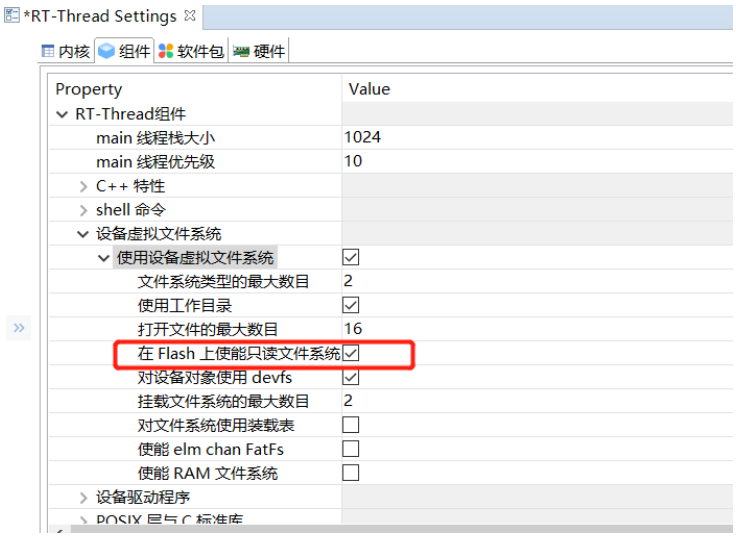
multibutton 软件包安装



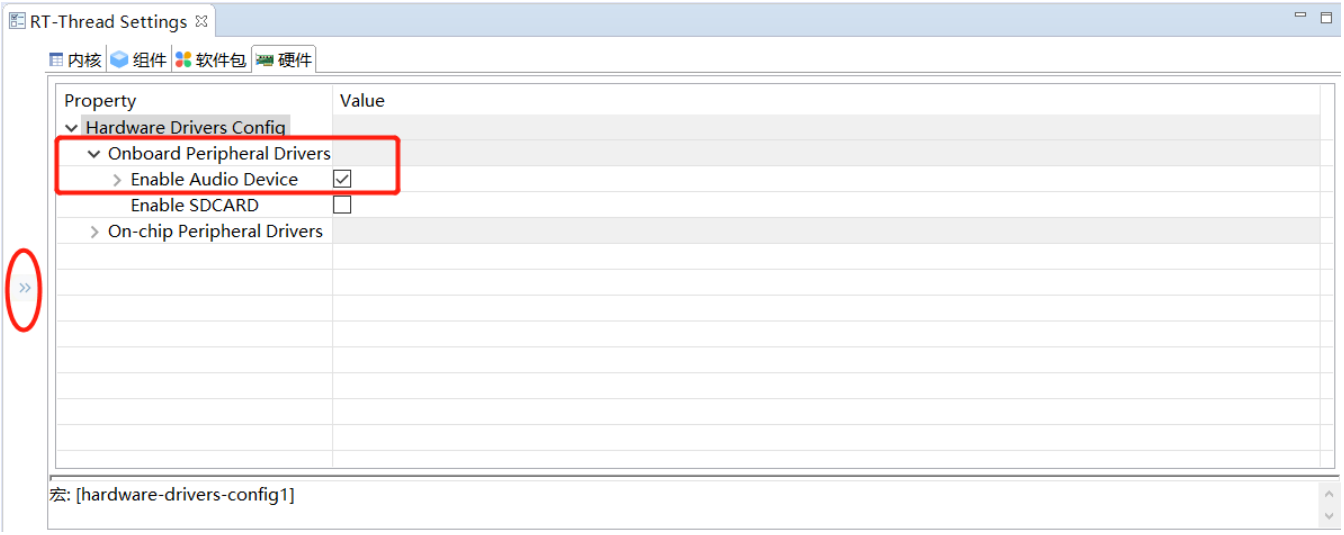
使能 DFS 组件


*RT-Thread Settings
软件包
Add
wavplayer
latest
optparse
latest
multibutton
v1.1.0
组件和服务层
右键启用DFS组件
finsh 命令
DFS
Fatfs
ulog 日志
C++
SAL
AT 客户端
lwIP
POSIX
libc
utest 测试框架
ymodem
右键停用POSIX组件
Drivers
串口
Pin
SPI
SFUD
软件模拟 RTC
软件模拟 I2C
音频
低功耗
传感器
SDIO
更多配置

右键 DFS 详细配置

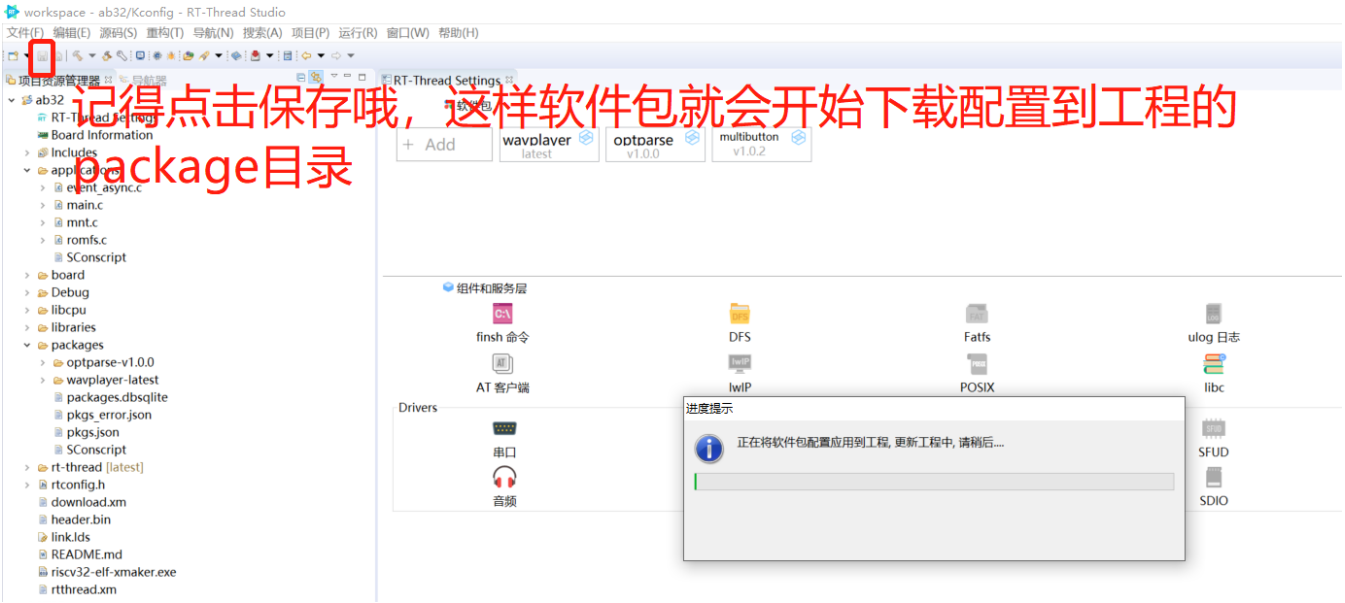

*RT-Thread Settings
内核
组件
软件包
硬件
Property
Value
RT-Thread组件
main 线程栈大小
1024
main 线程优先级
10
C++ 特性
shell 命令
设备虚拟文件系统
使用设备虚拟文件系统
文件系统类型的最大数目
2
使用工作目录
打开文件的最大数目
16
在 Flash 上使能只读文件系统
对设备对象使用 devfs
挂载文件系统的最大数目
2
对文件系统使用装载表
使能 elm chan FatFs
使能 RAM 文件系统
设备驱动程序

使能 Audio 设备



保存,下载软件包到工程

软件包选择完成后，点击 保存 按钮，将配置保存并应用到工程中。保存的时候会弹出进度提示框，提示保存进度，会自动下载到 package 目录下。



demo 编写

安装完 wavplayer/optparse/multibutton 三个软件包之后，就完成此次试验所需要的依赖的软件包。接下来开始编写 demo。

首先需要下载 romfs.c（本文件包含了两个音频文件用于 demo 播放） 替换 applications

下原有的 romfs.c

romfs.c：https://ab32vg1-example.readthedocs.io/zh/latest/_downloads/c80ffd3057bc4e3e621c37859aec34f0/romfs.c

检查一下 mnt.c 这个文件里的挂载信息，看看是否挂载的是 romfs，不是的话进行下面的修改

```
#include <dfs_fs.h>
#include "dfs_romfs.h"
int mnt_init(void)
 {
       if (dfs_mount(RT_NULL, "/", "rom", 0, &(romfs_root)) == 0)
       {
             rt_kprintf("ROM file system initializated!\n");
       }
      else
      {
            rt_kprintf("ROM file system initializate failed!\n");
      }

      return 0;
}
INIT_ENV_EXPORT(mnt_init);
```

然后在 applications 下新建 event_async.c 文件，复制以下代码

```
#include <rtthread.h>
#include <rtdevice.h>
#include "board.h"
#include <multi_button.h>
#include "wavplayer.h"

#define BUTTON_PIN_0 rt_pin_get("PF.0")
#define BUTTON_PIN_1 rt_pin_get("PF.1")

#define NUM_OF_SONGS       (2u)

static struct button btn_0;
static struct button btn_1;

static uint32_t cnt_0 = 0;
static uint32_t cnt_1 = 0;

static char *table[2] =
{
      "wav_1.wav",
      "wav_2.wav",
};

void saia_channels_set(uint8_t channels);
```

```
void saia_volume_set(rt_uint8_t volume);
uint8_t saia_volume_get(void);

static uint8_t button_read_pin_0(void)
{
    return rt_pin_read(BUTTON_PIN_0);
}

static uint8_t button_read_pin_1(void)
{
    return rt_pin_read(BUTTON_PIN_1);
}

static void button_0_callback(void *btn)
{
    uint32_t btn_event_val;

    btn_event_val = get_button_event((struct button *)btn);

    switch(btn_event_val)
    {
    case SINGLE_CLICK:
        if (cnt_0 == 1) {
            saia_volume_set(30);
        }else if (cnt_0 == 2) {
```

```
            saia_volume_set(50);
        }else {
            saia_volume_set(100);
            cnt_0 = 0;
        }
        cnt_0++;
        rt_kprintf("vol=%d\n", saia_volume_get());
        rt_kprintf("button 0 single click\n");
    break;

    case DOUBLE_CLICK:
        if (cnt_0 == 1) {
            saia_channels_set(1);
        }else {
            saia_channels_set(2);
            cnt_0 = 0;
        }
        cnt_0++;
        rt_kprintf("button 0 double click\n");
    break;

    case LONG_PRESS_START:
        rt_kprintf("button 0 long press start\n");
    break;
```

```
        case LONG_PRESS_HOLD:
            rt_kprintf("button 0 long press hold\n");
        break;
        }
}

static void button_1_callback(void *btn)
{
        uint32_t btn_event_val;

        btn_event_val = get_button_event((struct button *)btn);

        switch(btn_event_val)
        {
        case SINGLE_CLICK:
            wavplayer_play(table[(cnt_1++) % NUM_OF_SONGS]);
            rt_kprintf("button 1 single click\n");
        break;

        case DOUBLE_CLICK:
            rt_kprintf("button 1 double click\n");
        break;

        case LONG_PRESS_START:
            rt_kprintf("button 1 long press start\n");
```

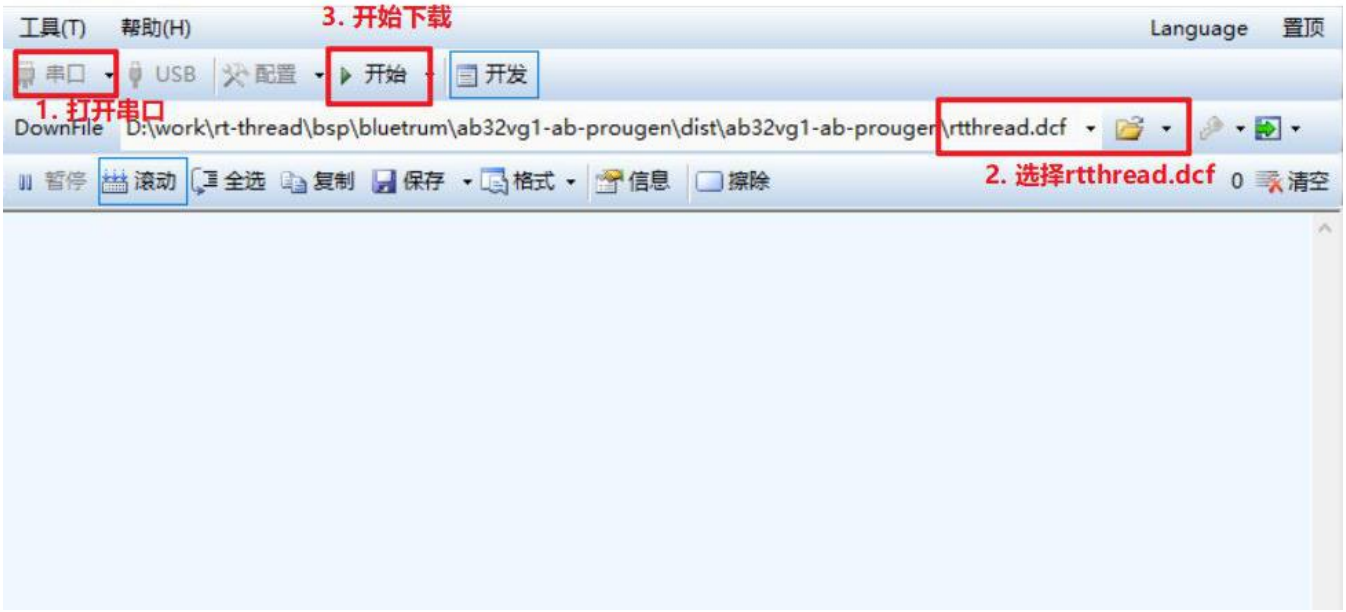
```
        break;

        case LONG_PRESS_HOLD:
            rt_kprintf("button 1 long press hold\n");
        break;
        }
}

static void btn_thread_entry(void* p)
{
        while(1)
        {
            /* 5ms */
            rt_thread_delay(RT_TICK_PER_SECOND/200);
            button_ticks();
        }
}

static int multi_button_test(void)
{
        rt_thread_t thread = RT_NULL;

        /* Create background ticks thread */
        thread = rt_thread_create("btn", btn_thread_entry, RT_NULL, 1024, 10, 10);
        if(thread == RT_NULL)
```

```
    {
        return RT_ERROR;
    }
    rt_thread_startup(thread);

    /* low level drive */
    rt_pin_mode    (BUTTON_PIN_0, PIN_MODE_INPUT_PULLUP);
    button_init    (&btn_0, button_read_pin_0, PIN_LOW);
    button_attach(&btn_0, SINGLE_CLICK,         button_0_callback);
    button_attach(&btn_0, DOUBLE_CLICK,         button_0_callback);
    button_attach(&btn_0, LONG_PRESS_START, button_0_callback);
    button_attach(&btn_0, LONG_PRESS_HOLD,    button_0_callback);
    button_start (&btn_0);

    rt_pin_mode    (BUTTON_PIN_1, PIN_MODE_INPUT_PULLUP);
    button_init    (&btn_1, button_read_pin_1, PIN_LOW);
    button_attach(&btn_1, SINGLE_CLICK,         button_1_callback);
    button_attach(&btn_1, DOUBLE_CLICK,         button_1_callback);
    button_attach(&btn_1, LONG_PRESS_START, button_1_callback);
    button_attach(&btn_1, LONG_PRESS_HOLD,    button_1_callback);
    button_start (&btn_1);

    return RT_EOK;
}
INIT_APP_EXPORT(multi_button_test);
```

程序下载

demo 编写完成后，单击编译按钮开始编译，编译成功后下载编译后生成的 .dcf 固件到芯片；

双击打开 Downloader v1.9.7。



下载成功后会在串口界面打印"Hello World"， 并会有 led 灯闪烁

此时按下按键 S2，会播放第一首音乐，再次按下，播放下一首音乐，依次循环。