RT-THREAD EtherKit 用户手册

RT-THREAD

版本和修订

Data	Version	Author	Note
2025-02-07	V1.1.0	RT-Thread	对应 SDK_V1.2.0
2024-12-09	V1.0.0	RT-Thread	对应 SDK_V1.1.0
2024-11-11	V0.1.0	RT-Thread	对应 SDK_V1.0.0

目录

第 1 章 简介

EtherKit 是一款专为工业以太网应用设计的高性能开发板，基于瑞萨 RZ/N2L 系列微处理器，面向工业自动化和物联网领域，旨在帮助工程师高效实现工业以太网功能。EtherKit 配备支持 TSN 的三端口千兆以太网交换机和 EtherCAT 从站控制器，兼容 EtherCAT、PROFINET 等主流工业协议，以满足多协议应用需求。

RT-Thread 对 EtherKit 提供全面支持，使开发者能够在 RT-Thread 操作系统的生态下实现各种工业以太网功能。RT-Thread 为 EtherKit 提供完整的实时操作系统支持，包括实时内核、文件系统、中间件、网络协议栈（如 TCP/IP、IPv4/6 等），以及工业协议栈（如 Modbus、EtherCAT 协议等），确保系统的高效、稳定运行。此外，RT-Thread 的网络框架让 EtherKit 能够轻松支持各种工业通信协议，便于开发人员在物联网应用中实现快速网络集成。

EtherKit 的 RZ/N2L 微处理器搭载 Arm® Cortex®-R52 内核，以 400MHz 的频率运行，并配有 ECC 支持的大容量片上 RAM，从而可以实现对实时协议的独立处理，显著减轻应用程序 CPU 的负载。内置的三端口千兆以太网交换机支持 TSN，并具备 EtherCAT® 从属控制器，非常适合多种工业以太网场景。

此外，EtherKit 板载丰富的外设资源和扩展接口（如 GPT 定时器、HyperRAM、CAN 等），使其在远程 I/O、传感器集线器、逆变器和工业网关等场景中表现出色。RT-Thread 对 EtherKit 的驱动程序和示例代码提供了全面支持，使开发人员能够快速完成原型设计和功能评估，加速进入工业物联网领域，满足嵌入式工业通信的多样化需求。

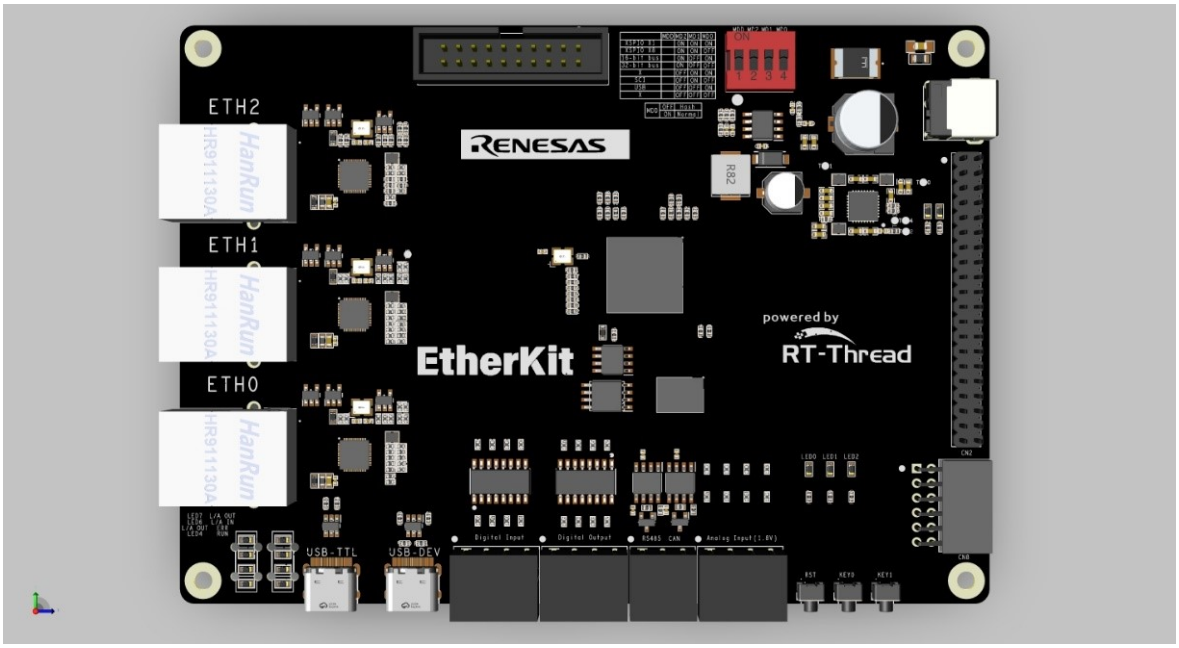


图 1-1 EtherKit 正面图

1.1 目录结构

```
|-- README.md
|-- docs
|-- libraries
| |-- HAL_Drivers
|-- projects
|-- rt-thread
`-- sdk-bsp-rzn2l-etherkit.yaml
```

- docs：EtherKit 原理图、用户手册等
- libraries：RZ 通用外接驱动程序
- projects：示例项目文件夹，包括各种示例代码
- rt-thread：rt-thread 源代码
- sdk-bsp-rzn2l-etherkit.yaml：描述 EtherKit 的硬件信息

1.2 快速上手

sdk-bsp-rzn2l-etherkit 支持 RT-Thread Studio 和 IAR 开发。

1.2.1 RT-Thread Studio 开发

打开 RT-Thread Studio ，安装 EtherKit 开发板支持包（如有最新建议安装最新版本，下图版本仅供参考）；**（如有最新建议安装最新版本，下图版本仅供参考）**

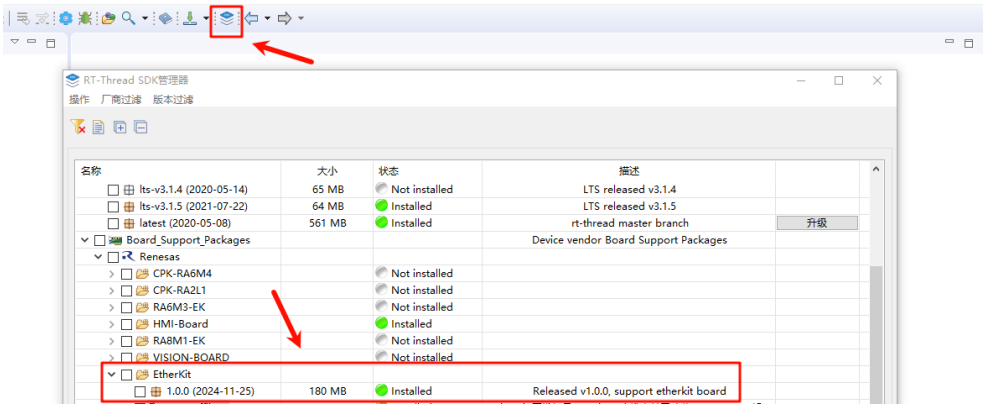


图 1-2 安装 etherkit SDK

新建 EtherKit 工程，选择左上角文件->新建->RT-Thread 项目->基于开发板，可以创建示例工程和模板工程；

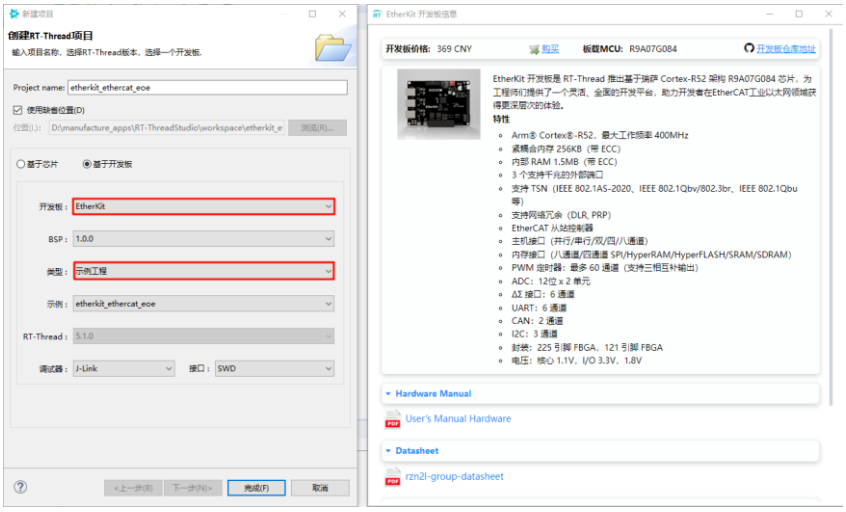


图 1-3 创建工程

进行工程的编译和下载：

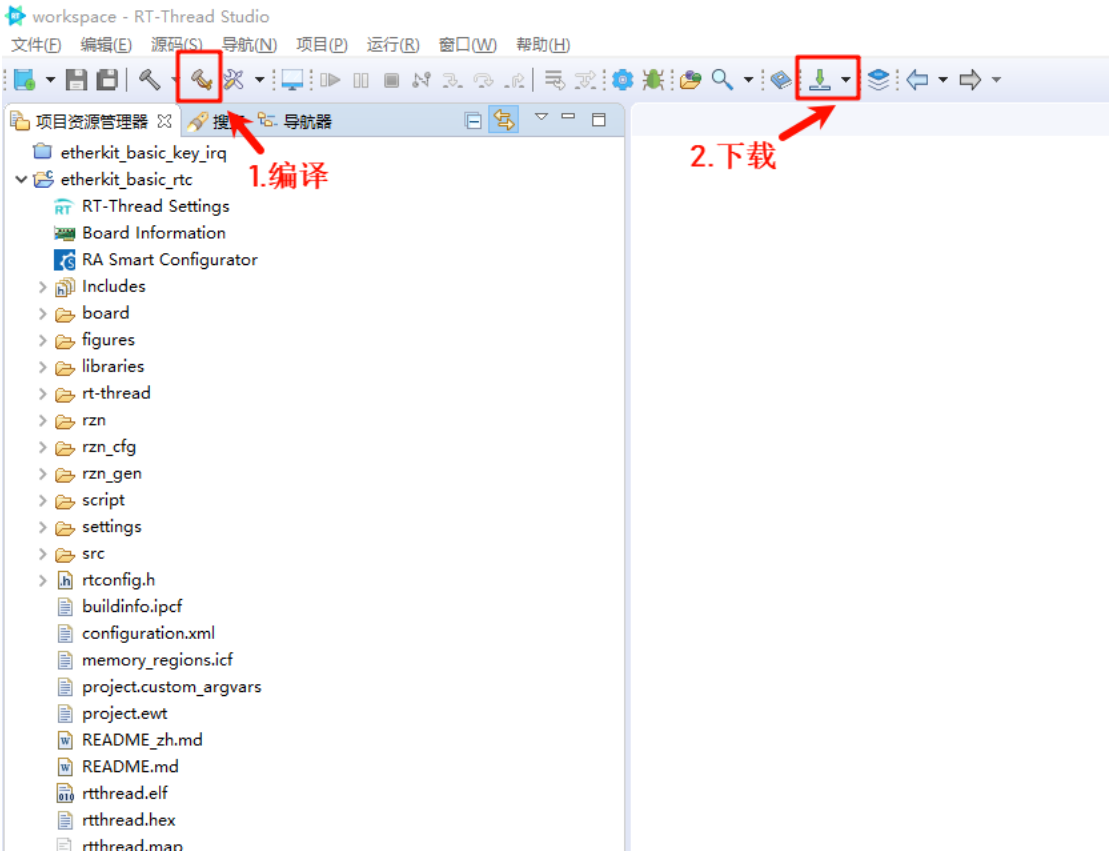


图 1-4 工程编译及下载

1.2.2 IAR 开发

首先本地克隆 EtherKit SDK 仓库：

```
git clone https://github.com/RT-Thread-Studio/sdk-bsp-rzn2l-etherkit.git
```

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

1. 双击某个 project 目录下的 mklinks.bat 文件，或者使用 Env 工具执行 mklinks.bat 命令，分别为 rt-thread 及 libraries 文件创建符号链接。

2. 查看目录下是否有 rt-thread 和 libraries 的文件夹图标。

3. 使用 Env 工具执行 scons –target=iar 更新 IAR 工程文件。

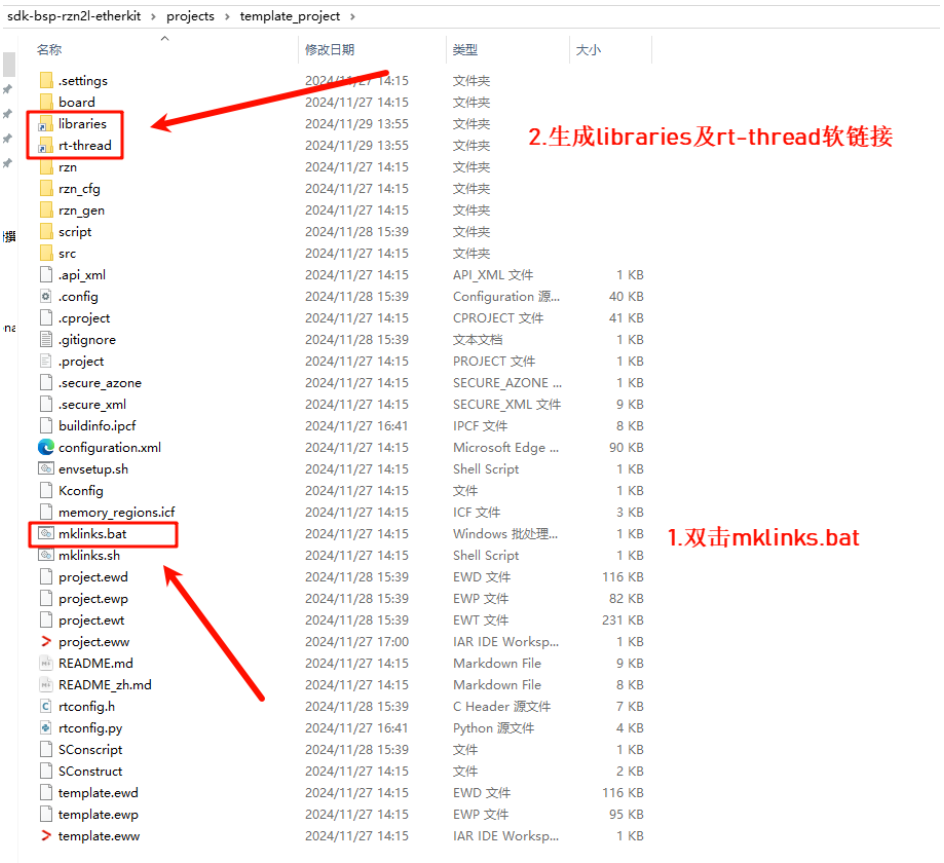


图 1-5 创建软链接

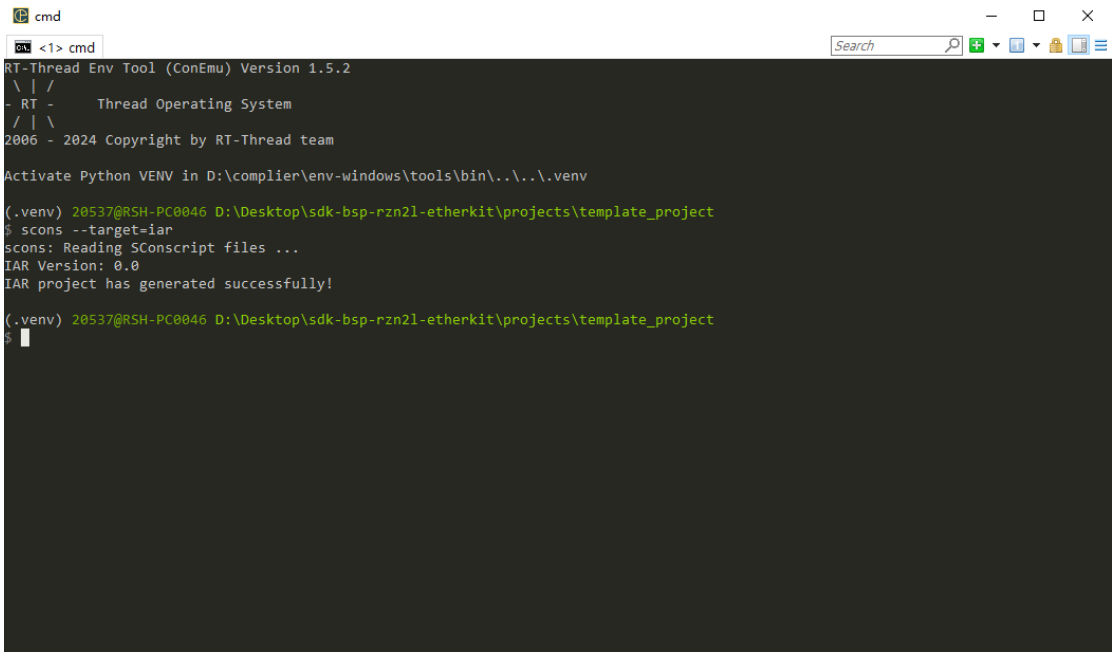


图 1-6 生成 IAR 工程

双击 project.eww 文件打开 IAR 工程，点击下图按钮进行项目全编译：

图 1-7 编译项目

点击下图按钮进行固件烧录：

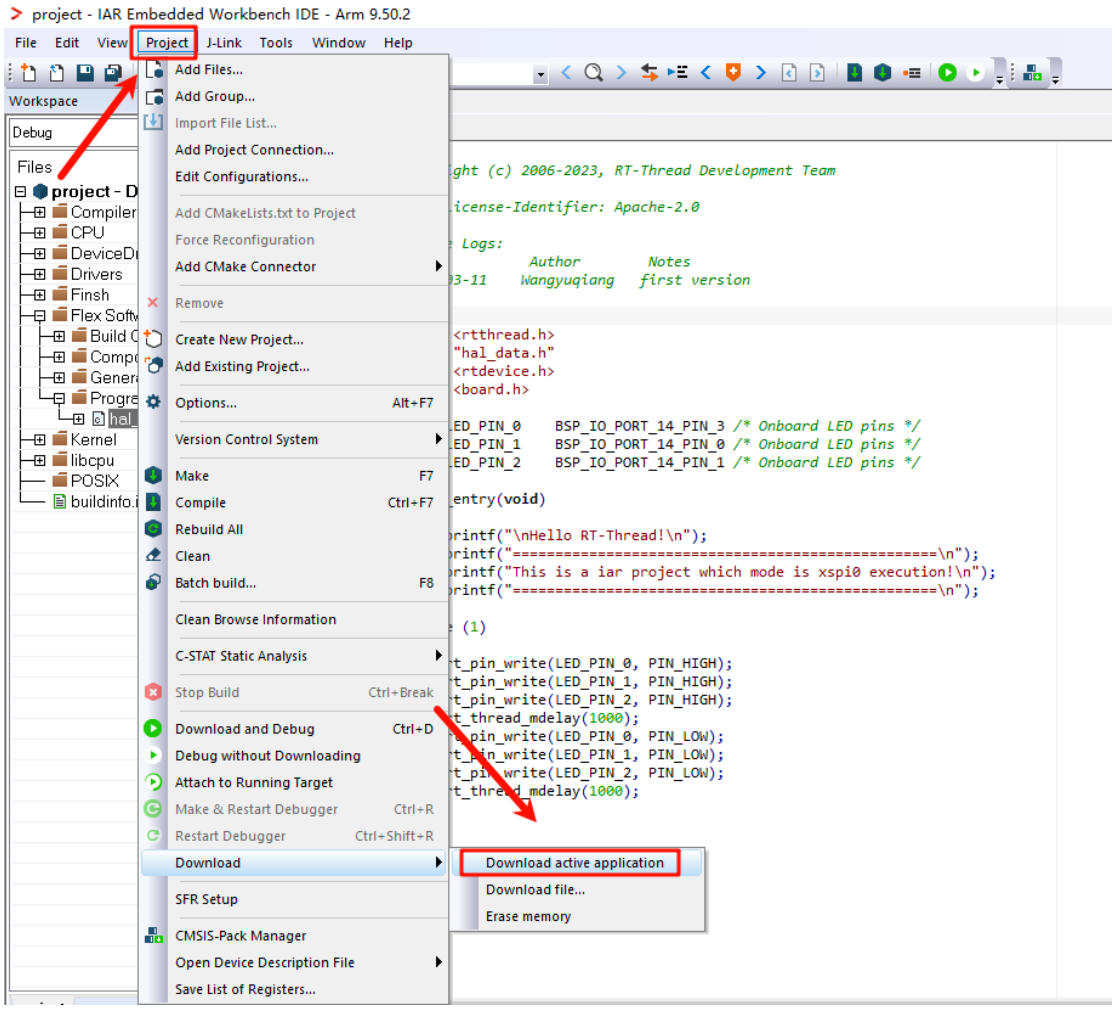


图 1-8 下载固件

1.2.3 开发板接线示意

EtherKit 接线需要用到一根 typec 数据线及一个 Jlink 调试器（Jlink v11 或 v

12 版本），接线示意如下：

图 1-9 EtherKit 接线示意

打开串口终端，可在 Finsh 终端查看 RT-Thread 版本 logo，以及一些预置指令；

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Dec 13 2024 14:15:31
 2006 - 2024 Copyright by RT-Thread team

Hello RT-Thread!
================================================
This is a iar project which mode is xspi0 execution!
================================================
msh >
RT-Thread shell commands:
backtrace        - print backtrace of a thread
list             - list objects
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
pin              - pin [option]
reboot           - Reboot System

msh >
```

图 1-10 finsh 终端

1.3 交流平台（QQ 群）

对 EtherKit 感兴趣的小伙伴可以加入 QQ 群-EtherKit 兴趣小组，群号：930079668。

1.4 开发板购买渠道

如果您对 EtherKit 开发板感兴趣，欢迎在睿赛德淘宝店铺购买此开发板，链接：https://item.taobao.com/item.htm?abbucket=17&id=855679103445.

第 2 章 RGB 闪烁例程

2.1 简介

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

2.2 硬件说明

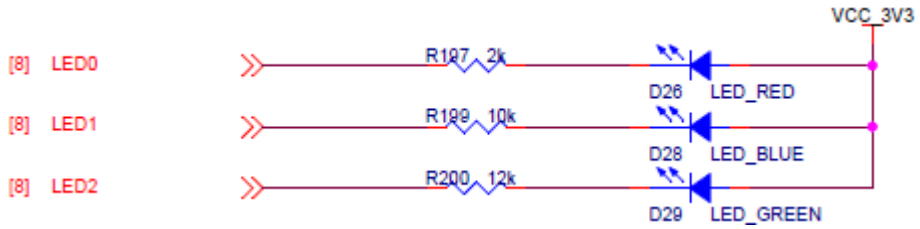


图 2-1 LED 电路原理图

图 2-2 LED 引脚示意图

如上图所示，EtherKit 提供三个用户 LED，分别为 LED0（RED）、LED1（BLUE）、LED2（GREEN），其中 LED_RED 对应引脚 P14_3。单片机引脚输出低电平即可点亮 LED ，输出高电平则会熄灭 LED。

LED 在开发板中的位置如下图所示：

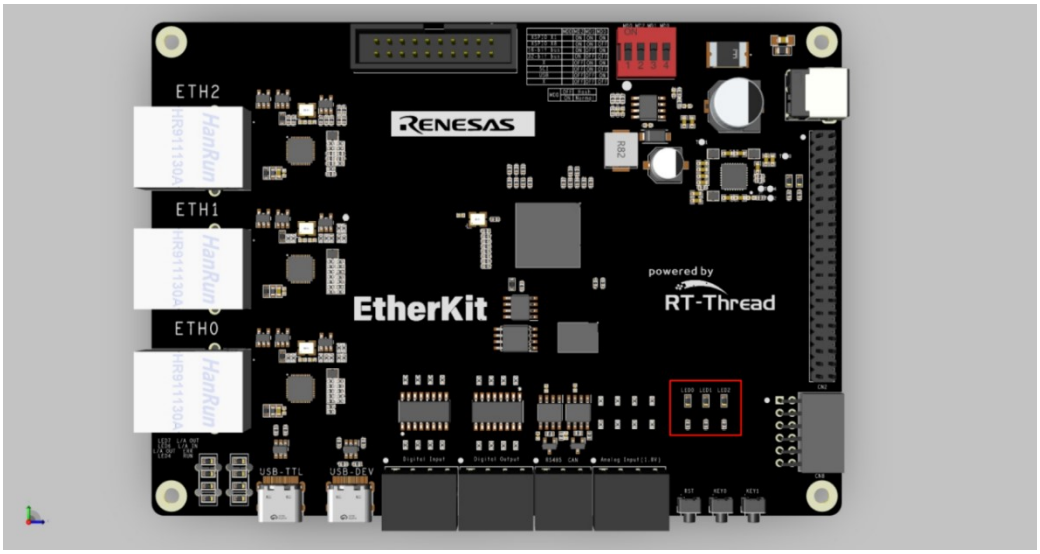


图 2-3 LED 位置

2.3 软件说明

本例程的源码位于/projects/etherkit_blink_led.

RGB-LED 对应的单片机引脚定义及 RGB 变换源码可以通过查阅 src/hal_data.c 中。

```
/* 配置 LED 灯引脚 */
#define LED_PIN_R    BSP_IO_PORT_14_PIN_3 /* Onboard RED LED pins */
#define LED_PIN_B    BSP_IO_PORT_14_PIN_0 /* Onboard BLUE LED pins */
#define LED_PIN_G    BSP_IO_PORT_14_PIN_1 /* Onboard GREEN LED pins */
    do
    {
        /* 获得组编号 */
        group_current = count % group_num;

        /* 控制 RGB 灯 */
        rt_pin_write(LED_PIN_R, _blink_tab[group_current][0]);
        rt_pin_write(LED_PIN_B, _blink_tab[group_current][1]);
        rt_pin_write(LED_PIN_G, _blink_tab[group_current][2]);

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

```
            _blink_tab[group_current][1] == LED_ON ? "ON" : "OFF",
            _blink_tab[group_current][2] == LED_ON ? "ON" : "OFF");

        count++;

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

2.4 运行

2.4.1 编译&下载

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

编译完成后，将开发板的 Jlink 接口与 PC 机连接，然后将固件下载至开发板。

2.4.2 运行效果

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

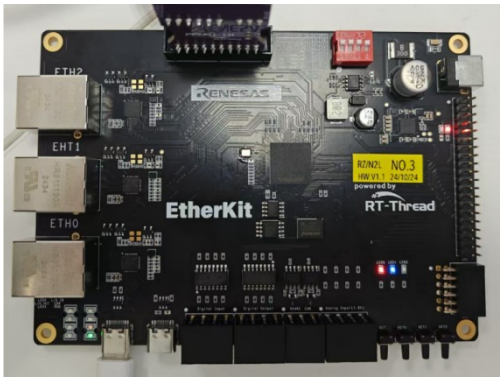


图 2-4 RGB-LED 演示

此时也可以在 PC 端使用终端工具打开开发板的默认配置的串口，设置波特率为 115200N。开发板的运行日志信息即可实时输出出来。

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

2.5 注意事项

暂无

2.6 引用参考

- 设备与驱动：PIN 设备

第 3 章 按键中断例程

3.1 简介

本例程主要功能是通过板载的按键 KEY 实现外部中断，当指定的 KEY 被按下时，打印相关信息，同时触发对应的 LED 亮起。

中断是计算机系统中的一个重要概念，用于处理来自外部设备或软件的事件或信号。当一个事件发生时，例如用户按下键盘上的一个键或者硬盘传输数据完成，系统会发出一个中断信号，以通知 CPU 停止当前执行的任务并处理该事件。中断的目的是实现多任务处理和异步事件处理。它允许计算机在执行某个任 务时，能够立即响应外部设备的输入或其他需要处理的事件。中断可以被看作是一种特殊的信号，它打断了正常的程序执行流程，使得处理器能够优先处理一些紧急的任务。当一个中断事件发生时，处理器会保存当前的执行状态，包括程序计数器和寄存器的值，并转而执行一个预先定义的中断处理程序（中断服务程序）。中断处理程序会根据不同的中断类型进行相应的处理，例如读取键盘输入、发送数据到打印机等。完成中断处理后，处理器会恢复之前保存的执行状态，继续执行被中断的任务。

3.2 硬件说明

图 3-1 key 电路原理图

图 3-2 key 引脚示意图

如上图所示， KEY1(LEFT)、KEY2(RIGHT)引脚分别连接单片机 P14_2(LEFT)和 P16_3(RIGHT)引脚，KEY 按键按下为高电平，松开为低电平。

KEY 在开发板中的位置如下图所示：

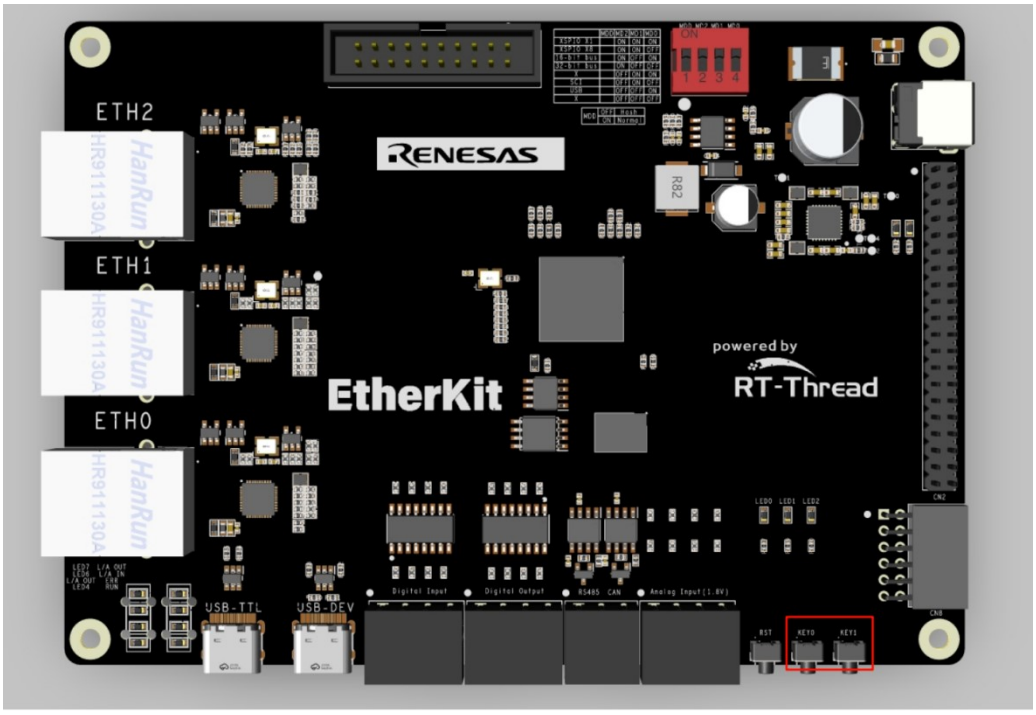


图 3-3 按键位置

3.3 软件说明

3.3.1 FSP 配置

首先下载官方 FSP 代码生成工具：https://github.com/renesas/rzn-fsp/releases/download/v2.0.0/setup_rznfsp_v2_0_0_rzsc_v2024-01.1.exe；安装成功之后我们双击 eclipse 下的 rasc.exe，并依次根据下图打开工程配置文件 configuration.xml：

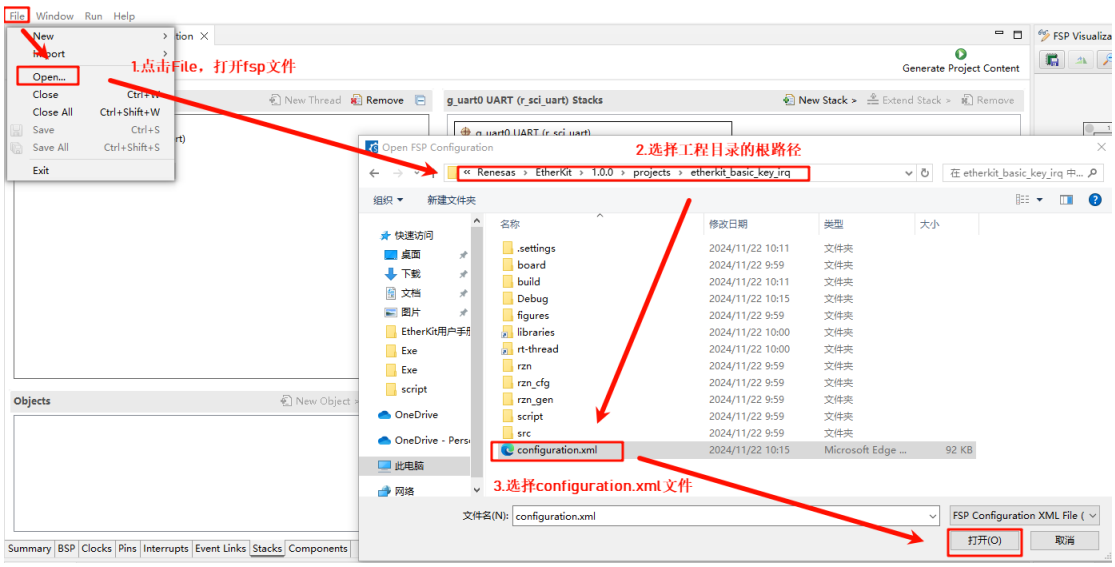


图 3-4 打开配置文件

下面我们新增两个 Stack：New Stack->Input->External IRQ(r_icu)；

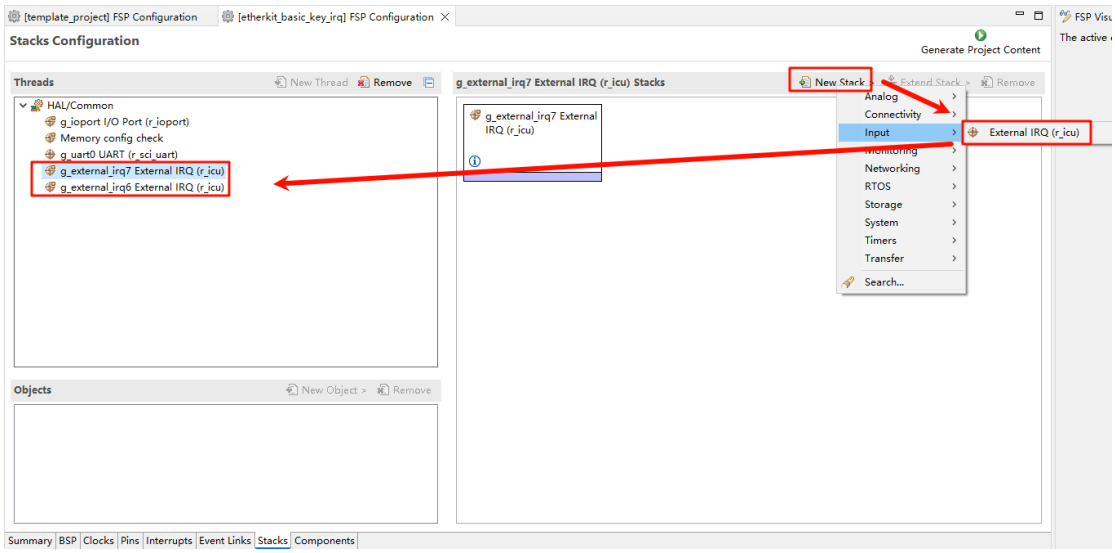


图 3-5 新增 IRQ Stack

接着我们需要在引脚配置那开启 IRQ 功能，根据下图选中我们要使能的两个中断引脚：KEY1(IRQ6)和 KEY2(IRQ7)；

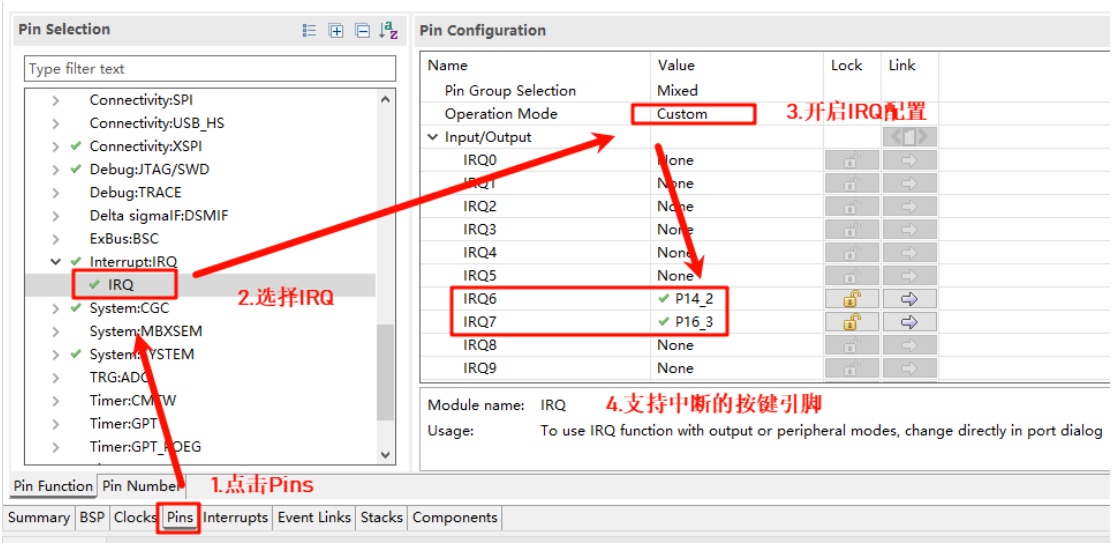


图 3-6 IRQ 开启

回到 Stacks 界面，这里分别设置 IRQ6 和 IRQ7，配置对应的中断名称、通道号以及中断回调函数；

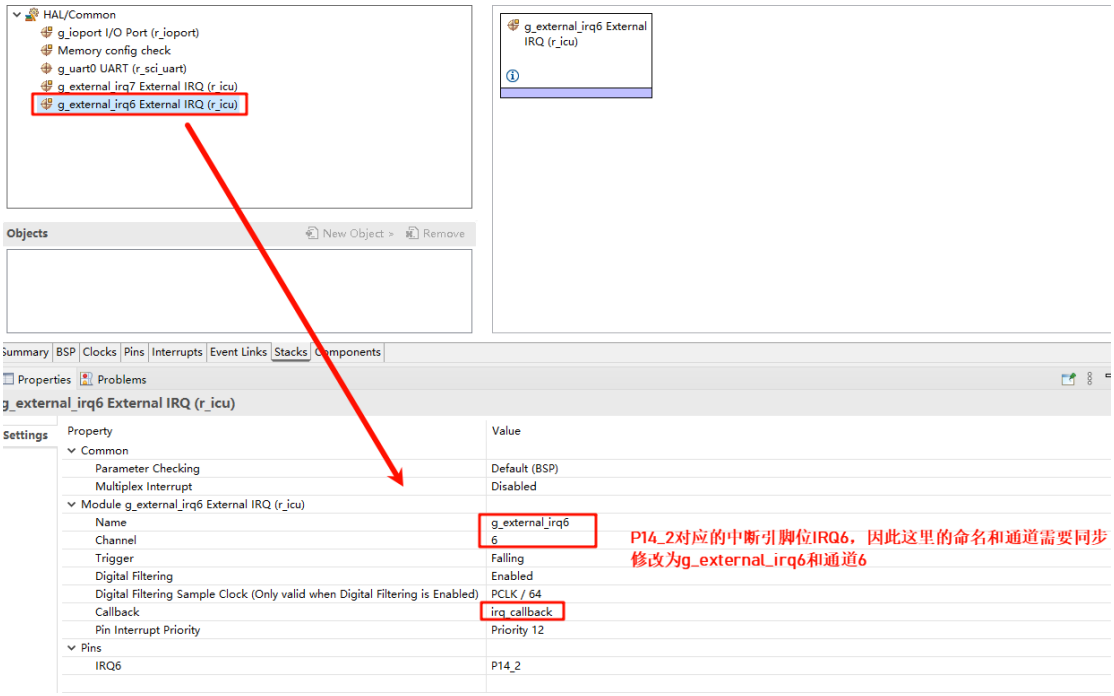


图 3-7 IRQ 配置

3.3.2 示例代码说明

本例程的源码位于/projects/etherkit_basic_key_irq。

KEY1(LEFT) 、KEY2(RIGHT)对应的单片机引脚定义如下。

```
/* 配置 key irq 引脚 */

#define IRQ_TEST_PIN1 BSP_IO_PORT_14_PIN_2
#define IRQ_TEST_PIN2 BSP_IO_PORT_16_PIN_3
```

LED 灯的单片机引脚定义如下。

```
/* 配置 LED 灯引脚 */
#define LED_PIN_B    BSP_IO_PORT_14_PIN_0 /* Onboard BLUE LED pins */
#define LED_PIN_G    BSP_IO_PORT_14_PIN_1 /* Onboard GREEN LED pins */
```

按键中断的源代码位于/projects/etherkit_basic_key_irq/src/hal_entry.c 中，当按下对应的中断按键，会触发相应的打印信息。

```
static void irq_callback_test(void *args)
{
    rt_kprintf("\n IRQ:%d triggered \n", args);
}

void hal_entry(void)
{
    rt_kprintf("\nHello RT-Thread!\n");
    rt_kprintf("================================================\n");
    rt_kprintf("This example project is an basic key irq routine!\n");
    rt_kprintf("================================================\n");

    /* init */
    rt_err_t err = rt_pin_attach_irq(IRQ_TEST_PIN1, PIN_IRQ_MODE_RISING,
irq_callback_test, (void *)1);
    if (RT_EOK != err)
    {
        rt_kprintf("\n attach irq failed. \n");
    }
    err = rt_pin_attach_irq(IRQ_TEST_PIN2, PIN_IRQ_MODE_RISING,
irq_callback_test, (void *)2);
    if (RT_EOK != err)
    {
        rt_kprintf("\n attach irq failed. \n");
    }

    err = rt_pin_irq_enable(IRQ_TEST_PIN1, PIN_IRQ_ENABLE);
    if (RT_EOK != err)
    {
        rt_kprintf("\n enable irq failed. \n");
```

```
    }
    err = rt_pin_irq_enable(IRQ_TEST_PIN2, PIN_IRQ_ENABLE);
    if (RT_EOK != err)
    {
        rt_kprintf("\n enable irq failed. \n");
    }
}
```

3.4 运行

3.4.1 编译&下载

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

编译完成后，将开发板的 Jlink 接口与 PC 机连接，然后将固件下载至开发板。

3.4.2 运行效果

按下复位按键重启开发板，初始状态下的 LED1 和 LED2 处于灭灯状态，当按下 KEY1 时，LED1(Blue)亮起；当按下 KEY2 时，LED2(Green)亮起。

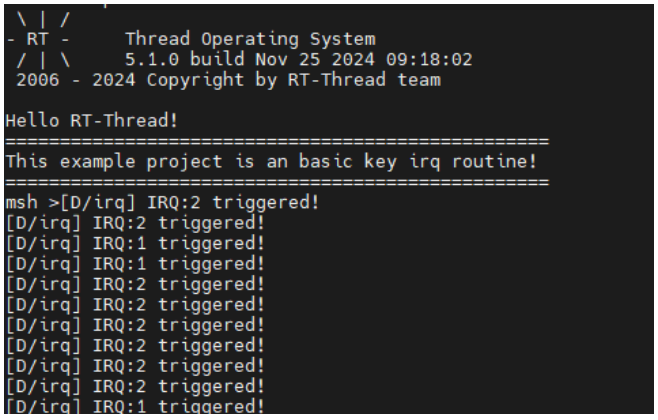


图 3-8 IRQ 运行示例

3.5 注意事项

暂无

3.6 引用参考

- 设备与驱动：PIN 设备

第 4 章 RTC 及 alarm 使用例程

4.1 简介

本例程主要介绍了如何在 EtherKit 上使用 RTC（Real-Time Clock）实时时钟，RTC 可以提供精确的实时时间，它可以用于产生年、月、日、时、分、秒等信息。目前实时时钟芯片大多采用精度较高的晶体振荡器作为时钟源。有些时钟芯片为了在主电源掉电时还可以工作，会外加电池供电，使时间信息一直保持有效。

RT-Thread 的 RTC 设备为操作系统的时间系统提供了基础服务。面对越来越多的 IoT 场景，RTC 已经成为产品的标配，甚至在诸如 SSL 的安全传输过程中，RTC 已经成为不可或缺的部分。

4.2 硬件说明

本例程使用的 RTC 设备依赖于 LSE 时钟，此外无需过多连接。

4.3 软件说明

4.3.1 FSP 配置说明

打开 FSP，选择对应的工程文件下的 configuration.xml，新增 RTC Stack;

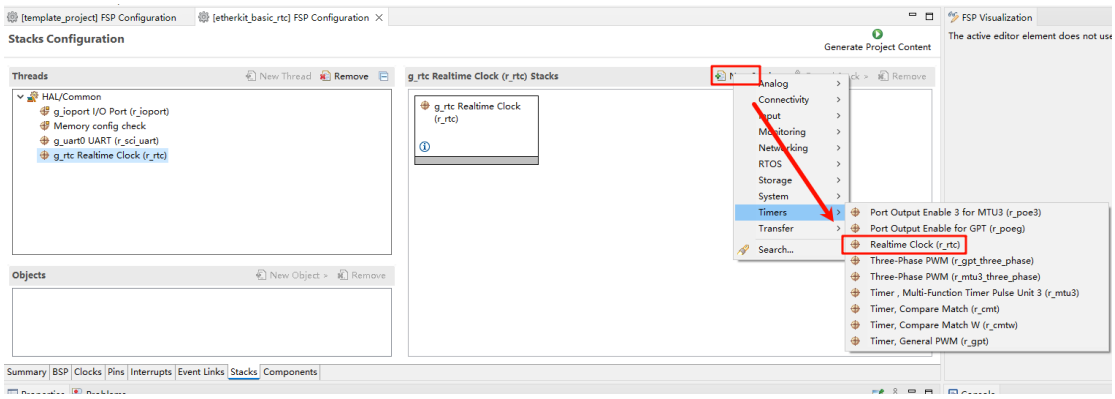


图 4-1 新增 RTC Stack

下面进行 RTC 参数的配置，设置 rtc stack name 为 g_rtc，设置 RTC 中断回调函数为 rtc_callback，并配置中断回调优先级；

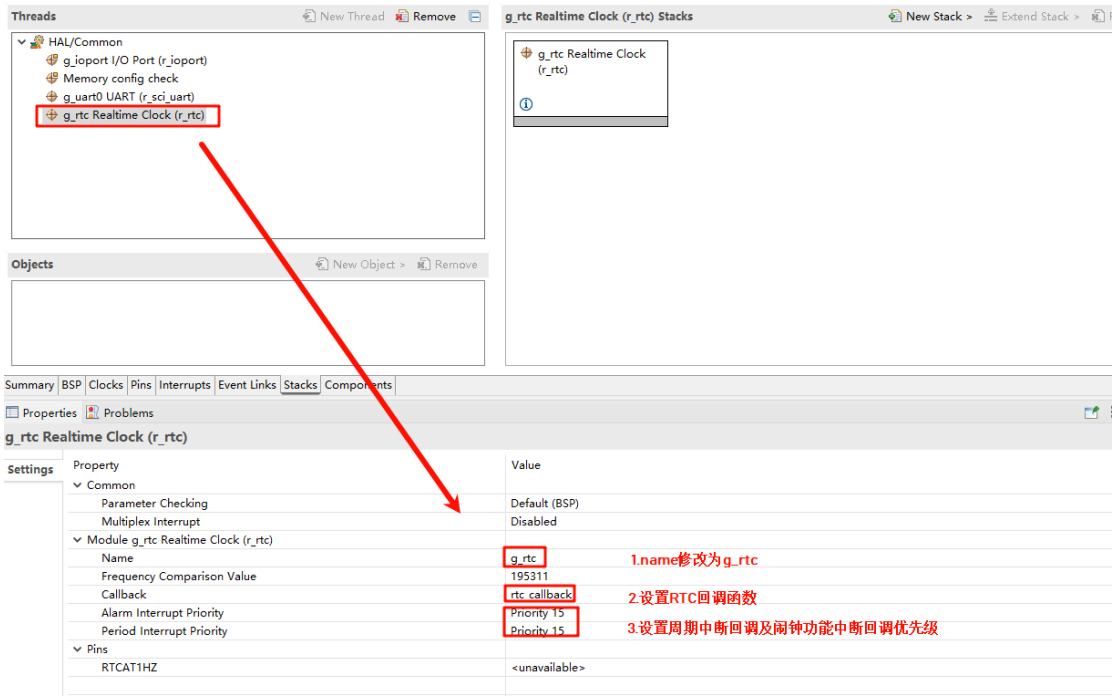


图 4-2 RTC 参数配置

4.3.2 RT-Thread Settings 配置

打开 RT-Thread Settings，找到硬件选项，使能 RTC；

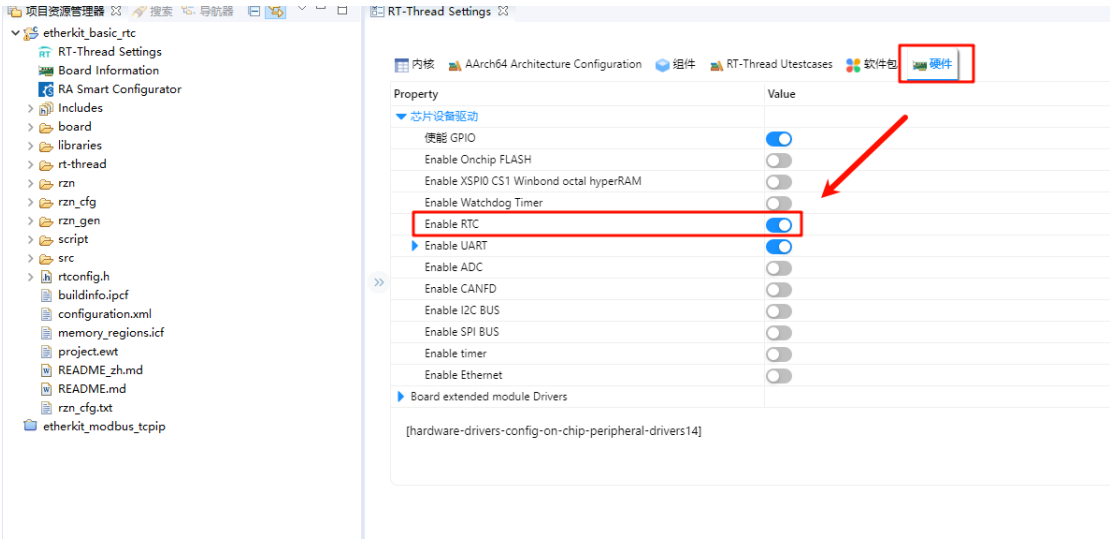


图 4-3 RTC 使能

接下来我们配置 RTC，首先需要使能 RT-Thread 的 RTC 设备框架，同时使能软件 alarm 功能（注：瑞萨 rzn 系列的 alarm 功能暂时存在一些问题，因此闹钟功能暂时使用软件模拟，不影响使用）；

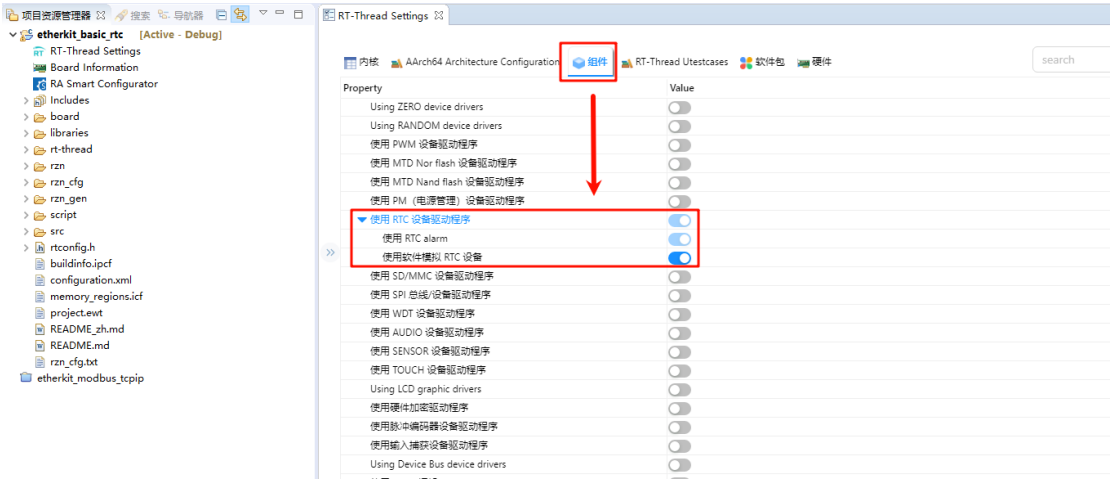


图 4-4 RTC 配置

4.3.2 示例代码说明

本例程的源码位于/projects/etherkit_basic_rtc。在 hal_entry()函数中，获取到了 RTC 设备，然后设置一次系统时间，随后获取一次系统时间以便检测时间是否设置成功，最后延时 1s 后再次获取系统时间。

```
    rt_err_t ret = RT_EOK;
    time_t now;
    rt_device_t device = RT_NULL;

    device = rt_device_find(RTC_NAME);
    if (!device)
    {
      rt_kprintf("find %s failed!\n", RTC_NAME);
    }

    if(rt_device_open(device, 0) != RT_EOK)
    {
      rt_kprintf("open %s failed!\n", RTC_NAME);
    }

    /* 设置日期 */
    ret = set_date(2024, 11, 11);
    rt_kprintf("set RTC date to 2024-11-11\n");
    if (ret != RT_EOK)
    {
        rt_kprintf("set RTC date failed\n");
    }

    /* 设置时间 */
    ret = set_time(15, 00, 00);
    if (ret != RT_EOK)
    {
        rt_kprintf("set RTC time failed\n");
    }

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

    /* 获取时间 */
    get_timestamp(&now);
    rt_kprintf("now: %.*s", 25, ctime(&now));
```

下面代码可创建一个 RTC 闹钟，然后设置 1 秒后唤醒，最后把该函数导入 msh 命令行中。

```
void user_alarm_callback(rt_alarm_t alarm, time_t timestamp)
{
    rt_kprintf("user alarm callback function.\n");
}
```

```
void alarm_sample(void)
{
    rt_device_t dev = rt_device_find("rtc");
    struct rt_alarm_setup setup;
    struct rt_alarm * alarm = RT_NULL;
    static time_t now;
    struct tm p_tm;

    if (alarm != RT_NULL)
        return;

    /* 获取当前时间戳，并把下一秒时间设置为闹钟时间 */
    now = get_timestamp(NULL) + 1;
    gmtime_r(&now,&p_tm);

    setup.flag = RT_ALARM_SECOND;
    setup.wktime.tm_year = p_tm.tm_year;
    setup.wktime.tm_mon = p_tm.tm_mon;
    setup.wktime.tm_mday = p_tm.tm_mday;
    setup.wktime.tm_wday = p_tm.tm_wday;
    setup.wktime.tm_hour = p_tm.tm_hour;
    setup.wktime.tm_min = p_tm.tm_min;
    setup.wktime.tm_sec = p_tm.tm_sec;

    alarm = rt_alarm_create(user_alarm_callback, &setup);
    if(RT_NULL != alarm)
    {
        rt_alarm_start(alarm);
    }
}
/* export msh cmd */
MSH_CMD_EXPORT(alarm_sample,alarm sample);
```

4.4 运行

4.4.1 编译&下载

- RT-Thread Studio：在 RT-Thread Studio 的包管理器中下载 EtherKit 资源包，然后创建新工程，执行编译。
- IAR：首先双击 mklinks.bat，生成 rt-thread 与 libraries 文件夹链接；再使用

Env 生成 IAR 工程；最后双击 project.eww 打开 IAR 工程，执行编译。

编译完成后，将开发板的 Jlink 接口与 PC 机连接，然后将固件下载至开发板。

4.4.2 运行效果

按下复位按键重启开发板，可以看到板子上会打印如下信息：

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Nov 13 2024 13:35:43
 2006 - 2024 Copyright by RT-Thread team

Hello RT-Thread!
==================================================
This example project is an rtc and alarm routine!
==================================================
set RTC date to 2024-11-11
msh >now: Sat Nov 19 07:42:42 3385
msh >alarm_sample
user alarm callback function.
user alarm callback function.
user alarm callback function.
user alarm callback function.
```

4.5 注意事项

暂无

4.6 引用参考

- 设备与驱动：RTC 设备

第 5 章 ADC 例程

5.1 简介

在嵌入式系统中，ADC（Analog-to-Digital Converter）是指一种将模拟信号（如传感器的输出）转换为数字信号的功能。

嵌入式系统中的 ADC 功能通常包括以下几个方面：

1. **模拟信号采集**：从传感器或其他模拟信号源中采集模拟信号。
2. **信号转换**：将采集到的模拟信号转换为数字信号。
3. **数字信号处理**：对转换后的数字信号进行处理，例如滤波、放大、缩小等。

ADC 功能在嵌入式系统中非常重要，因为它使得系统能够接收和处理来自外部世界的模拟信号，从而实现诸如测量、控制、监控等功能。

例如，在一个工业控制系统中，ADC 功能可以用于将传感器的模拟输出（如温度、压力、流量等）转换为数字信号，从而实现对这些物理量的监控和控制。

本例程主要介绍了如何在 EtherKit 上使用 rtthread 的 ADC 框架完成通过 ADC 采集模拟信号并进行数字信号的转换；

5.2 硬件说明

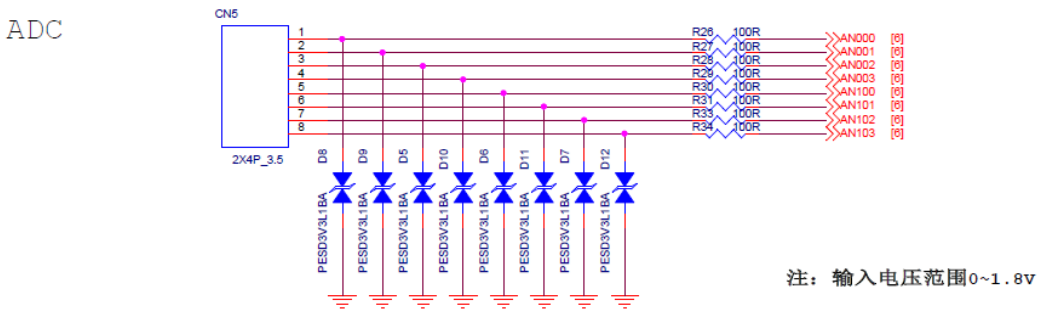


图 5-1 硬件原理图

如原理图所示：EtherKit 上留有 Analog Input 8 通道接口分别连接到单片机的 adc0、adc1 的通道 0、1、2、3；（注意，Analog Input 的耐压范围为 0~1.8 v）；

5.3 软件说明

5.3.1 FSP 配置说明

第一步：打开 FSP 导入 xml 配置文件；（或者直接点击 RT-Thread Studio 的 FSP 链接文件）；

第二步：新建 r_adc Stack 配置 adc 设备以及所用通道;

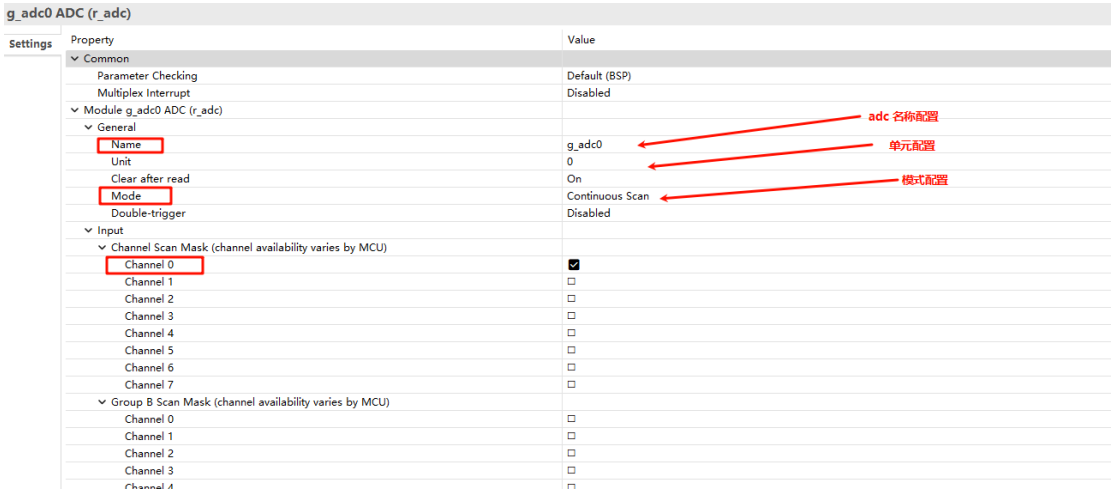


图 5-2 FSP 配置

5.3.2 RT-Thread Settings 配置

在 settings 里打开 adc0 的外设；

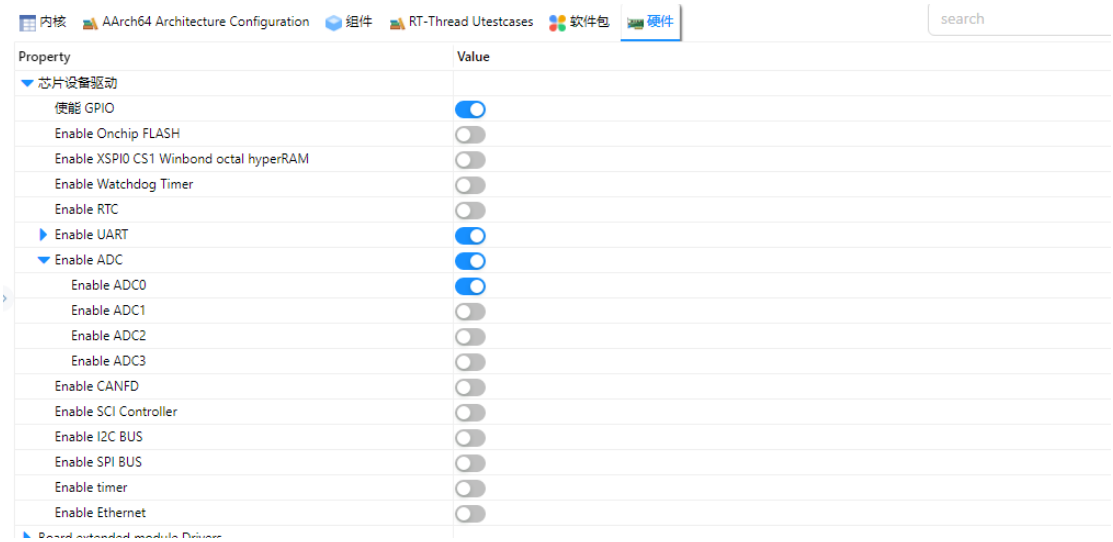


图 5-3 Settings 配置

5.3.3 示例工程说明

ADC 的源代码位于/projects/etherkit_driver_adc/src/hal_entry.c 中，使用的宏定义如下所示：

```
#define ADC_DEV_NAME        "adc0"      /* ADC 设备名称 */
#define ADC_DEV_CHANNEL     0           /* ADC 通道 */
#define REFER_VOLTAGE       180         /* 参考电压 1.8V,数据精度乘以100保留2位小数*/
#define CONVERT_BITS        (1 << 12)   /* 转换位数为12位 */
```

图 5-4 ADC 相关宏定义

具体功能为：每隔 1000ms 对 ADC0 的通道 0 采集一次模拟电压并进行一次转化；

具体代码如下：

```
static int adc_vol_sample()
{
    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;
}
```

示例中，While 循环每隔 1000ms 调用一次 adc_vol_sample；

5.4 运行

5.4.1 编译&下载

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

编译完成后，将开发板的 Jlink 接口与 PC 机连接，然后将固件下载至开发

板。

5.4.2 运行效果

使用 adc0 的 0 通道采集 1.8v 电压效果如下

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Nov 25 2024 14:28:51
 2006 - 2024 Copyright by RT-Thread team
[D/drv.adc] adc0 init success

Hello RT-Thread!
================================================
This example project is an driver adc routine!
================================================
msh >the value is :0
the voltage is :0.00
the value is :4095
the voltage is :1.79
the value is :4095
the voltage is :1.79
the value is :4095
the voltage is :1.79
the value is :4095
the voltage is :1.79
the value is :4095
the voltage is :1.79
the value is :4095
```

图 5-5 ADC 示例运行效果图

5.5 注意事项

暂无

5.6 引用参考

- 设备与驱动：ADC 设备

第 6 章 I2C 例程

6.1 简介

I2C（Inter-Integrated Circuit）是一种串行通信协议，用于连接和通信多个集成电路（IC）或设备。I2C 功能的原理如下：

- **基本原理**

I2C 协议使用两条线来进行通信：SCL（时钟线）和 SDA（数据线）。SCL 线用于传输时钟信号，SDA 线用于传输数据信号。

- **通信过程**

I2C 通信过程如下：

1. **主机初始化**：主机设备（通常是微控制器）初始化 I2C 总线，设置时钟频率和数据传输方向。
2. **从机地址**：主机设备发送从机设备的地址到 I2C 总线上。
3. **数据传输**：主机设备发送数据到从机设备，或者从机设备发送数据到主机设备。
4. **应答**：从机设备发送应答信号（ACK）到主机设备，确认数据传输成功。
5. **停止**：主机设备发送停止信号（STOP）到 I2C 总线上，结束通信。

- **通信模式**

I2C 协议支持两种通信模式：

1. **主机模式**：主机设备控制 I2C 总线，发送数据和命令到从机设备。
2. **从机模式**：从机设备响应主机设备的命令，发送数据到主机设备。

- **I2C 设备**

I2C 设备可以分为两类：

1. **主机设备**：控制 I2C 总线，发送数据和命令到从机设备。
2. **从机设备**：响应主机设备的命令，发送数据到主机设备。

I2C 功能在嵌入式系统中非常常见，因为它提供了一种简单、低成本的方式来连接和通信多个设备。

本例程主要介绍了如何在 EtherKit 上使用 RT-Thread 的 IIC 框架完成通过对板子上的 EEROM 的读写功能；

6.2 硬件说明

EtherKit 上的 EEROM 使用为 AT24C16 连接 R9A07G084M08GBG 芯片的 IIC0；

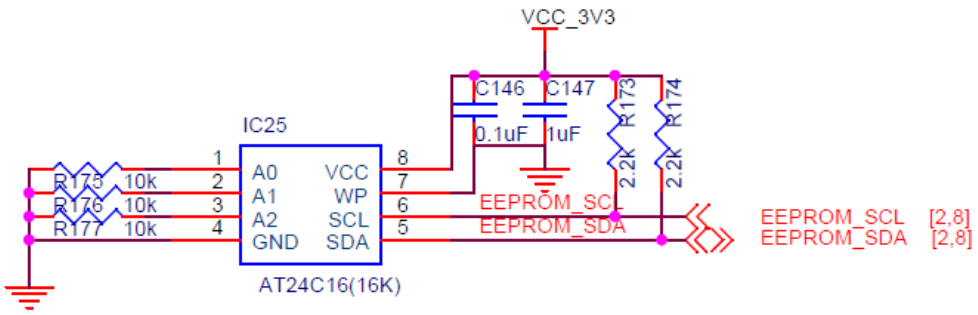


图 6-1 IIC 原理图

6.3 软件说明

6.3.1 FSP 配置说明

新建 stacks 选择 r_iic_master 并配置 IIC0 配置信息如下；

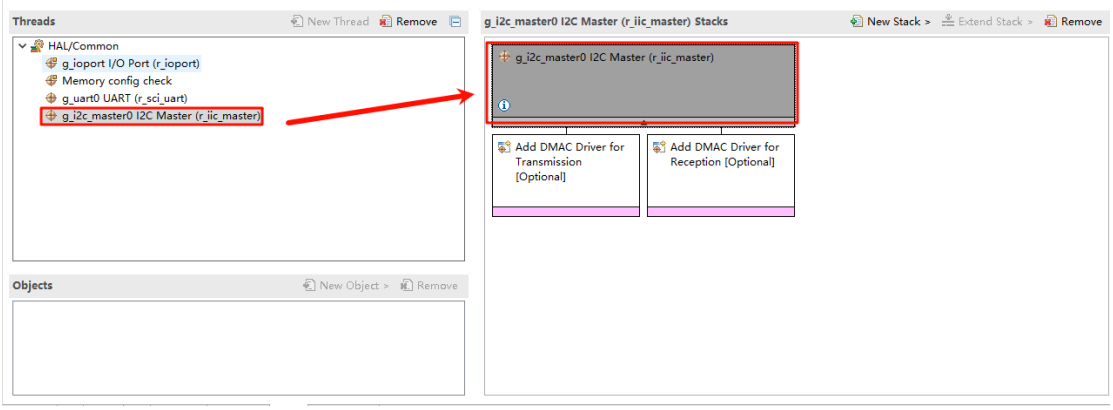


图 6-2 IIC master stack

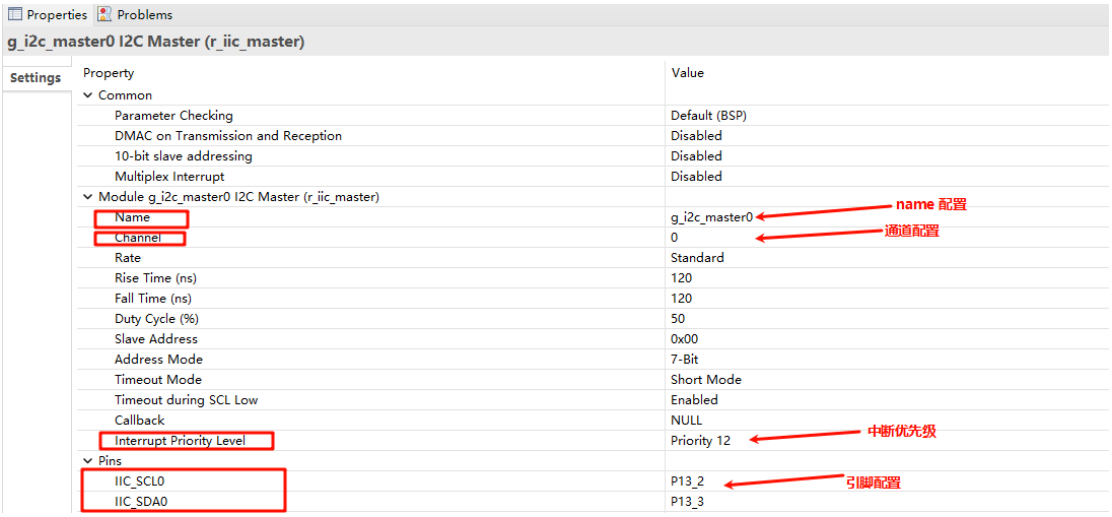


图 6-3 IIC 配置

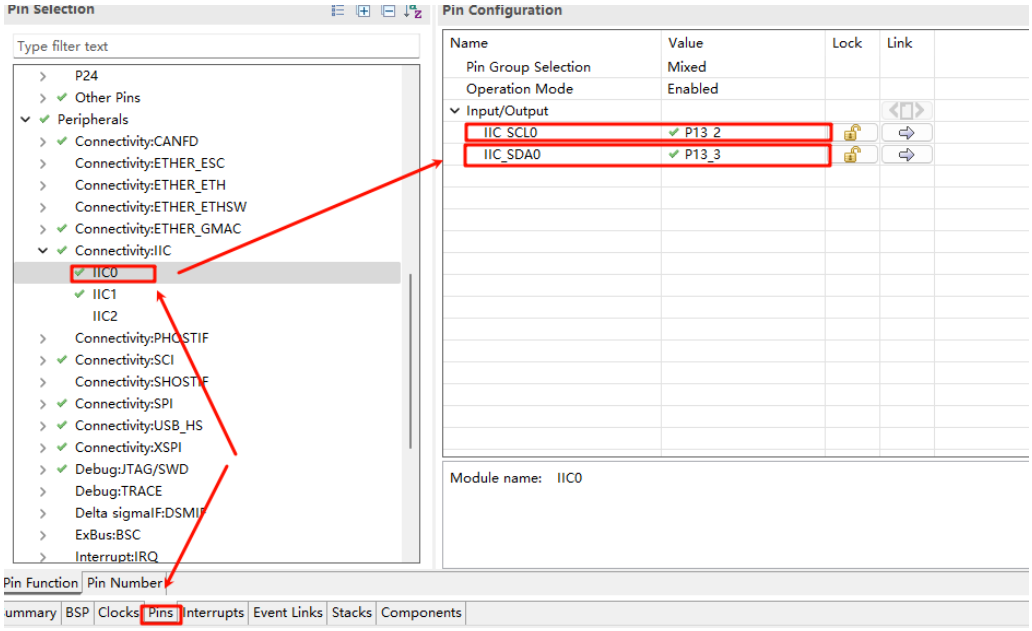


图 6-4 IIC 引脚配置

6.3.2 RT-Thread Settings 配置

在配置中打开 RT-Thread 的 IIC 驱动框架与 AT24C16 的驱动软件包；

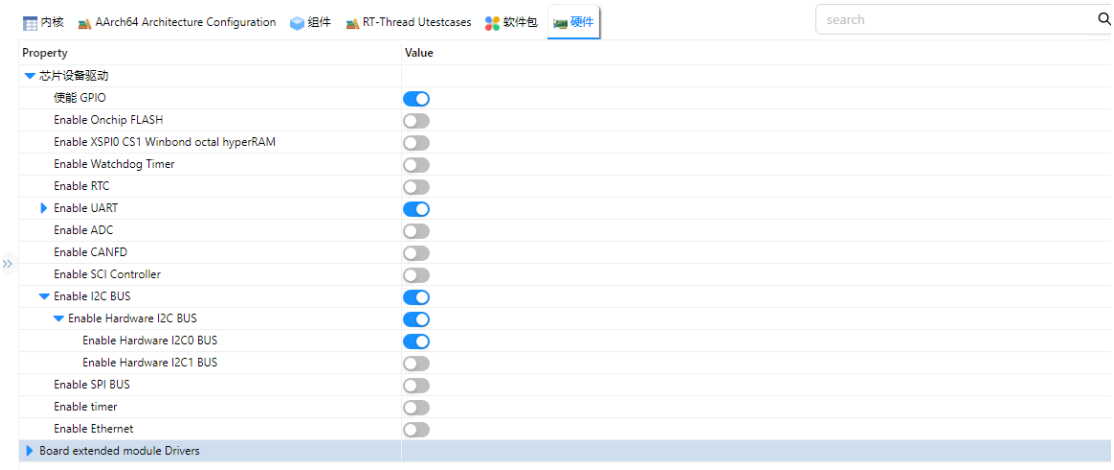


图 6-5 IIC 使能

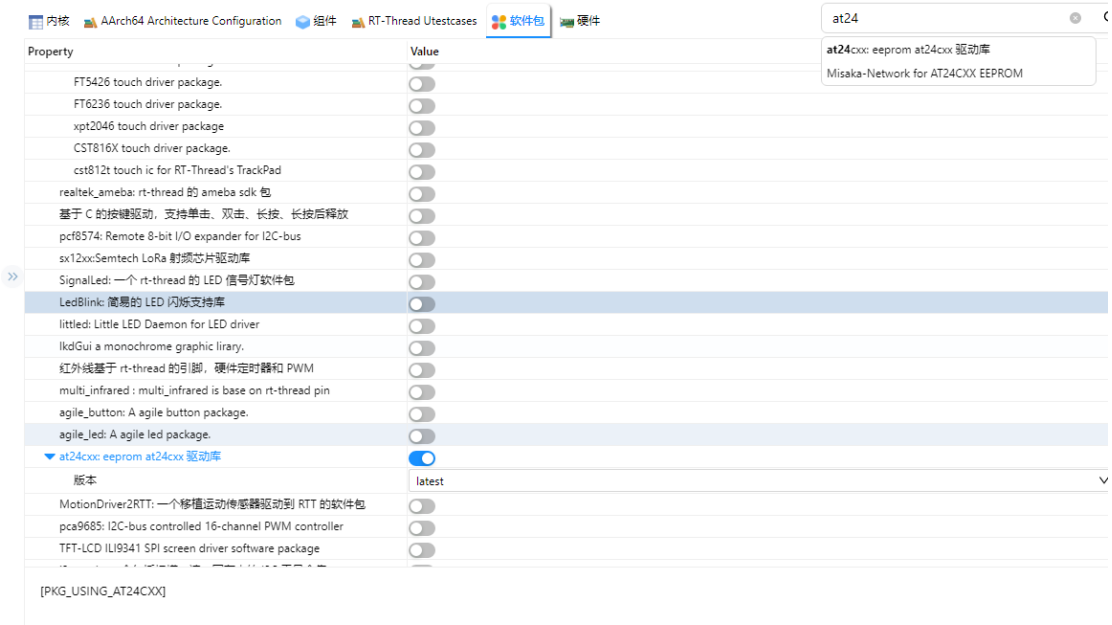


图 6-6 使能软件包

6.3.3 示例工程说明

本例程的源码位于/projects/etherkit_driver_iic 中；基于 AT24C16 的驱动软件包实现对 EEROM 的 0x00,0x20 地址写入与读出；

```
#ifdef PKG_USING_AT24CXX
#include "at24cxx.h"
#define EEPROM_I2C_NAME "i2c0"
static at24cxx_device_t at24c02_dev;
static void eeprom_test(void)
{
    char str1[] = "test string-hello rtthread\n";
    char str2[] = "test string-rzt2m eeprom testcase\n";
    uint8_t read_buffer1[50];
    uint8_t read_buffer2[50];
    at24c02_dev = at24cxx_init(EEPROM_I2C_NAME, 0x0);
    if (at24c02_dev == RT_NULL)
    {
        rt_kprintf("eeprom init failed\n");
        return;
    }
    rt_memset(read_buffer1, 0x0, sizeof(read_buffer1));
    rt_memset(read_buffer2, 0x0, sizeof(read_buffer2));
    at24cxx_write(at24c02_dev, 0x0, (uint8_t *)str1, (sizeof(str1) -
1));
    rt_kprintf("write eeprom data to 0x0: %s\n", str1);
    rt_thread_mdelay(1000);
    at24cxx_read(at24c02_dev, 0x0, read_buffer1, (sizeof(str1) - 1));
    rt_kprintf("read eeprom data from 0x0: %s\n", read_buffer1);
    at24cxx_write(at24c02_dev, 0x20, (uint8_t *)str2, (sizeof(str2) -
1));
    rt_kprintf("write eeprom data to 0x20: %s\n", str2);
    rt_thread_mdelay(1000);
    at24cxx_read(at24c02_dev, 0x20, read_buffer2, (sizeof(str2) - 1));
    rt_kprintf("read eeprom data from 0x20: %s\n", read_buffer2);
```

```
    if (rt_strcmp((const char *)str1, (const char *)read_buffer1) != 0 &&
 rt_strcmp((const char *)str2, (const char *)read_buffer2) != 0)
        rt_kprintf("eeprom test fail\n");
    else
        rt_kprintf("eeprom test success\n");
    at24cxx_deinit(at24c02_dev);
}
MSH_CMD_EXPORT(eeprom_test, eeprom test sample);
#endif
```

6.4 运行

6.4.1 编译&下载

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

编译完成后，将开发板的 Jlink 接口与 PC 机连接，然后将固件下载至开发板。

6.4.2 运行效果

在串口终端输入 eeprom_test 指令；

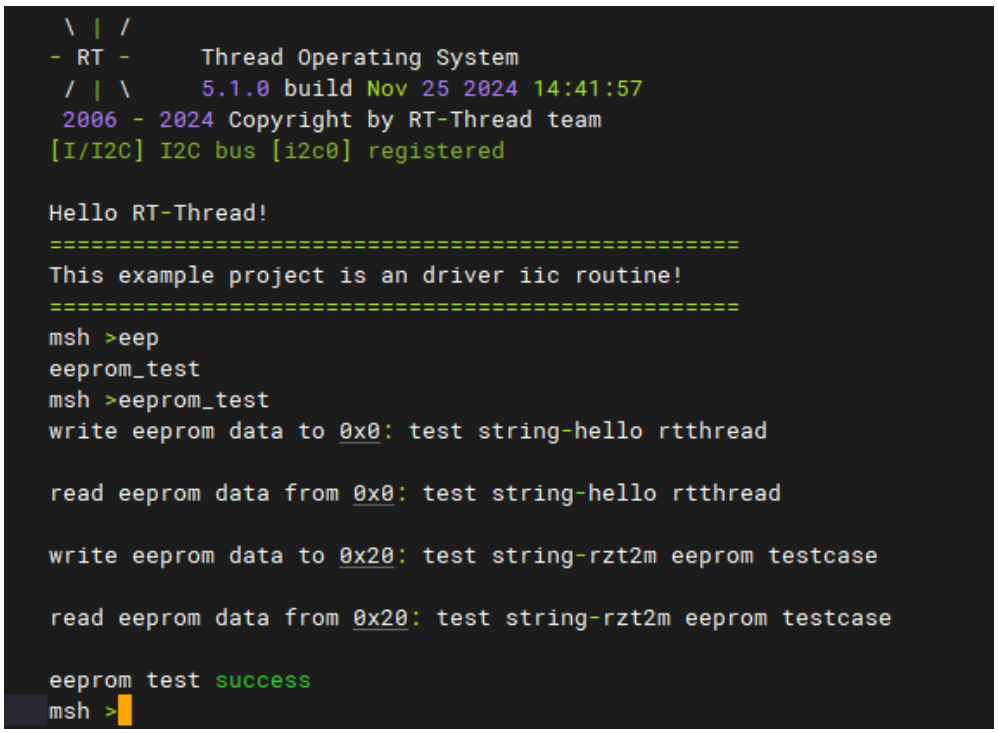


图 6-6 IIC 测试

6.5 注意事项

暂无

6.6 引用参考

- 设备与驱动：I2C 设备

第 7 章 SPI 例程

7.1 简介

SPI（Serial Peripheral Interface）是一种串行通信协议，用于连接和通信多个集成电路（IC）或设备。SPI 功能的原理如下：

- **基本原理**

SPI 协议使用四条线来进行通信：SCK（时钟线）、MOSI（主机输出从机输入）、MISO（主机输入从机输出）和 CS（片选线）。

- **通信过程**

SPI 通信过程如下：

1. **主机初始化**：主机设备（通常是微控制器）初始化 SPI 总线，设置时钟频率和数据传输方向。
2. **从机选择**：主机设备发送从机设备的选择信号（CS）到 SPI 总线上。
3. **数据传输**：主机设备发送数据到从机设备通过 MOSI 线，或者从机设备发送数据到主机设备通过 MISO 线。
4. **时钟信号**：主机设备发送时钟信号（SCK）到 SPI 总线上，用于同步数据传输。
5. **数据接收**：主机设备接收数据从从机设备通过 MISO 线，或者从机设备接收数据从主机设备通过 MOSI 线。
6. **结束**：主机设备结束通信，释放从机设备的选择信号（CS）。

- **通信模式**

SPI 协议支持两种通信模式：

1. **主机模式**：主机设备控制 SPI 总线，发送数据和命令到从机设备。
2. **从机模式**：从机设备响应主机设备的命令，发送数据到主机设备。

- **SPI 设备**

SPI 设备可以分为两类：

1. **主机设备**：控制 SPI 总线，发送数据和命令到从机设备。
2. **从机设备**：响应主机设备的命令，发送数据到主机设备。

SPI 功能在嵌入式系统中非常常见，因为它提供了一种高速、低成本的方式来连接和通信多个设备。

本例程主要介绍了如何在 EtherKit 上使用 RT-Thread 的 SCI_SPI 框架。

7.2 硬件说明

EtherKit 板载资源有 PMOD 接口，连接到 R9A07G084M08GBG 芯片的 SCI_SPI3；

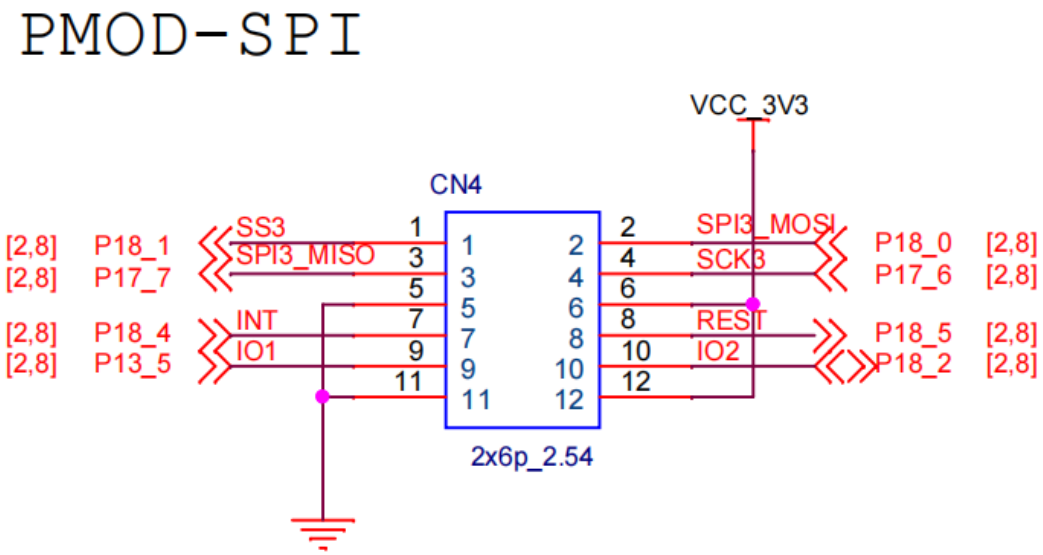


图 7-1 SPI 硬件原理图

7.3 软件说明

7.3.1 FSP 配置说明

打开 FSP 工具 新建 Stacks 选择 r_sci_spi3；

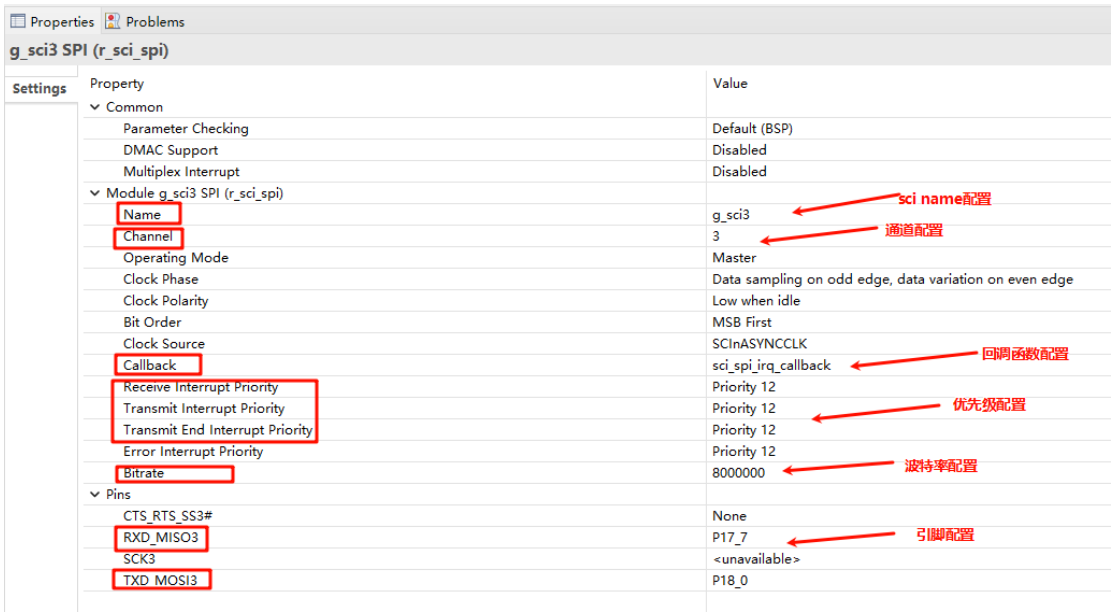


图 7-2 SPI 配置

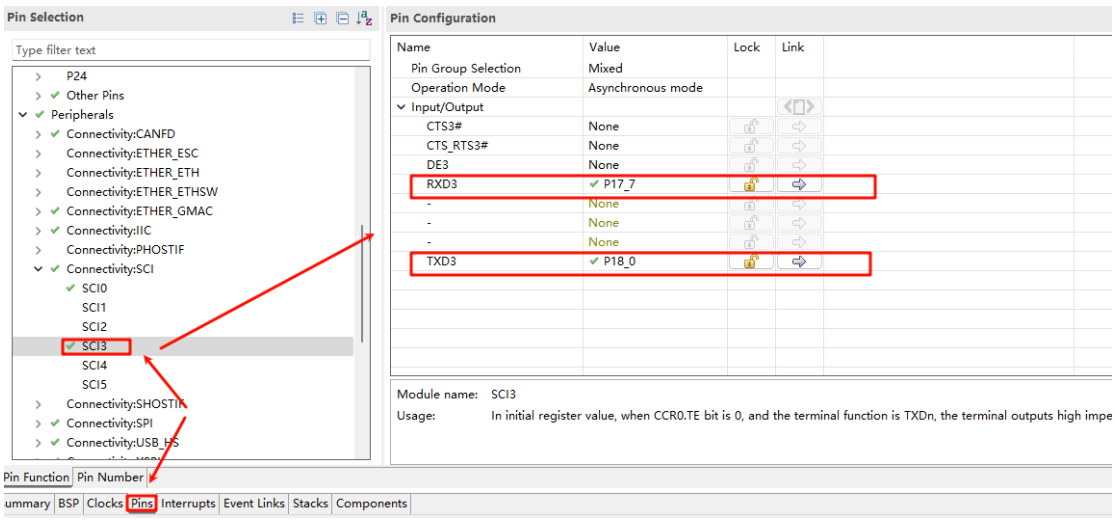


图 7-3 SCI-SPI 引脚配置

7.3.2 RT-Thread Settings 配置

打开 RT-Thread Settings，硬件选择 SCI，并配置 SCI3 模式为 SPI

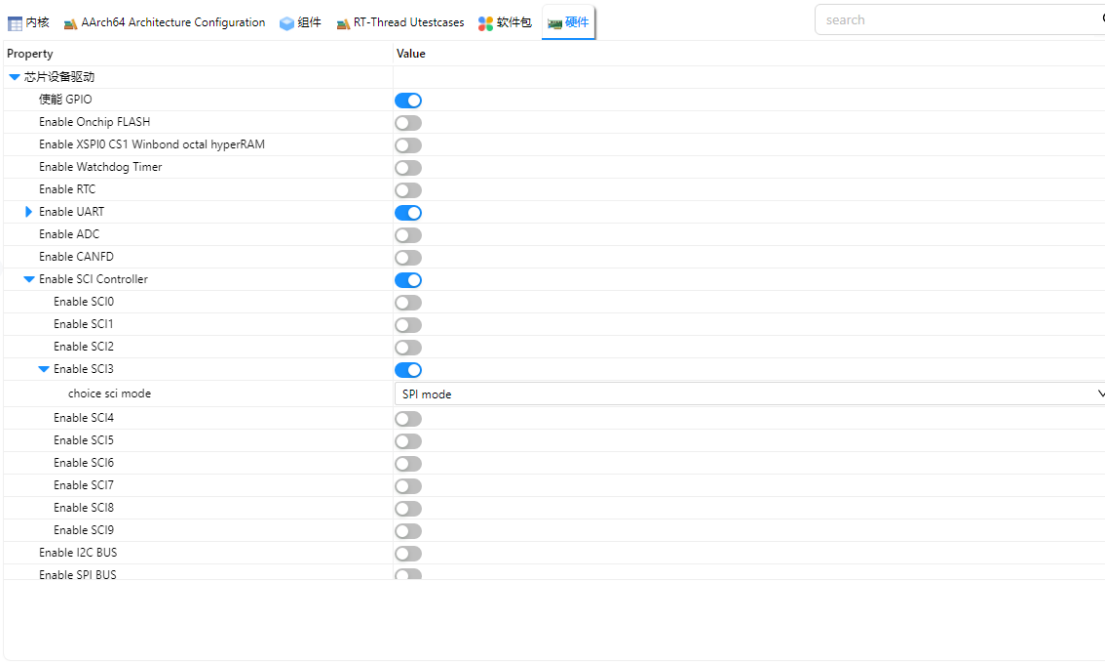


图 7-4 SPI 使能

7.3.3 示例工程说明

基于 RT-Thread 的 SCI 驱动框架实现对 PMODE 的 loop 回环测试；（将 PMOD 的 SPI3_MOSI 连接到 SPI3_MISO）；

代码如下：

```
void spi_loop_test(void)
{
#define TEXT_NUMBER_SIZE 1024
#define SPI_BUS_NAME "sci3s"
#define SPI_NAME "spi30"
    static uint8_t sendbuf[TEXT_NUMBER_SIZE] = {0};
    static uint8_t readbuf[TEXT_NUMBER_SIZE] = {0};
```

```
    for (int i = 0; i < sizeof(readbuf); i++)
    {
        sendbuf[i] = i;
    }
    static struct rt_spi_device *spi_dev = RT_NULL;
    struct rt_spi_configuration cfg;
    rt_hw_sci_spi_device_attach(SPI_BUS_NAME, SPI_NAME, NULL);
    cfg.data_width = 8;
    cfg.mode = RT_SPI_MASTER | RT_SPI_MODE_0 | RT_SPI_MSB | RT_SPI_NO_C
S;
    cfg.max_hz = 1 * 1000 * 1000;
    spi_dev = (struct rt_spi_device *)rt_device_find(SPI_NAME);
    if (RT_NULL == spi_dev)
    {
        rt_kprintf("spi sample run failed! can't find %s device!\n", SPI_
NAME);
        return;
    }
    rt_spi_configure(spi_dev, &cfg);
    rt_kprintf("%s send:\n", SPI_NAME);
    for (int i = 0; i < sizeof(sendbuf); i++)
    {
        rt_kprintf("%02x ", sendbuf[i]);
    }
    rt_spi_transfer(spi_dev, sendbuf, readbuf, sizeof(sendbuf));
    rt_kprintf("\n\n%s rcv:\n", SPI_NAME);
    for (int i = 0; i < sizeof(readbuf); i++)
    {
        if (readbuf[i] != sendbuf[i])
        {
```

```
            rt_kprintf("SPI test fail!!!\n");
            break;
        }
        else
            rt_kprintf("%02x ", readbuf[i]);
    }
    rt_kprintf("\n\n");
    rt_kprintf("SPI test end\n");
}
```

7.4 运行

7.4.1 编译&下载

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

编译完成后，将开发板的 Jlink 接口与 PC 机连接，然后将固件下载至开发板。

7.4.2 运行效果

打开串口工具，可以看到通过 spi 的发送与接收数据一致；

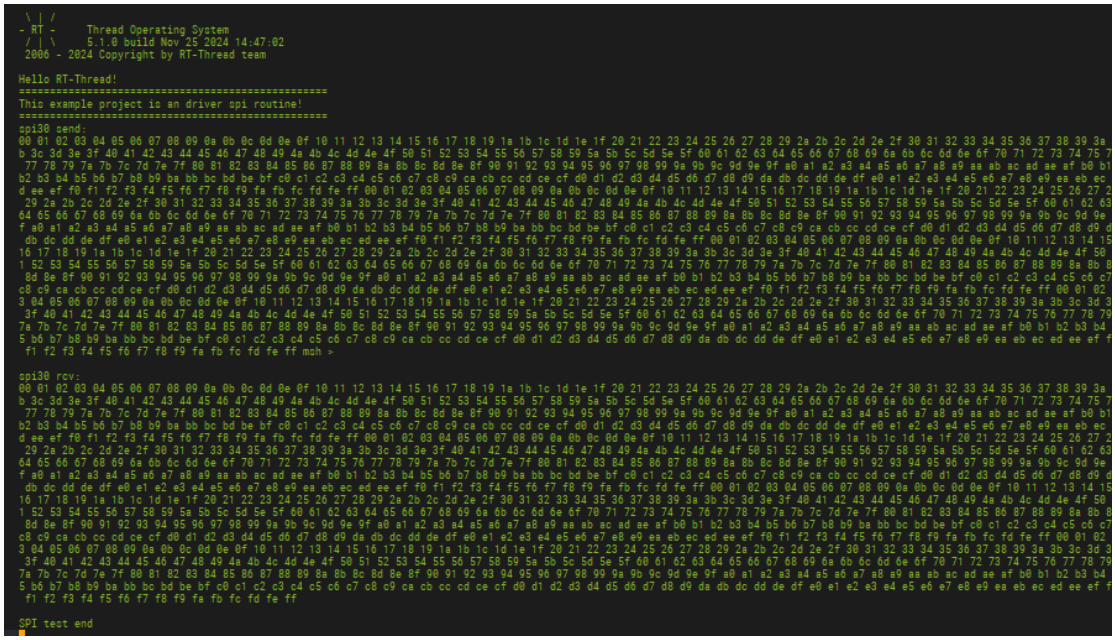


图 7-5 SPI 示例运行效果

7.5 注意事项

暂无

7.6 引用参考

- 设备与驱动：SPI 设备

第 8 章 GPT 例程

8.1 简介

定时器（Timer）是一种嵌入式系统中的硬件或软件组件，用于测量时间间隔或实现定时功能。定时器功能的原理如下：

- **基本原理**

定时器的基本原理是使用一个计数器来记录时间的流逝。计数器可以是硬件或软件实现的。

1. 硬件定时器：硬件定时器通常使用一个晶振或时钟信号作为计数器的输入。计数器会根据时钟信号的频率来增加或减少其值。当计数器达到预设值时，会产生一个中断信号，通知系统进行相应的处理。

2. 软件定时器：软件定时器则使用系统的时钟信号和软件算法来实现定时功能。软件定时器会周期性地检查当前时间，并在达到预设时间时执行相应的处理。

- **定时器模式**

定时器可以工作在以下几种模式：

1. **单次模式**：定时器只触发一次中断信号。
2. **周期模式**：定时器周期性地触发中断信号。
3. **延时模式**：定时器在延时时间后触发中断信号。

定时器应用

定时器在嵌入式系统中有许多应用，例如：

1. **任务调度**：定时器可以用于调度系统任务。
2. **数据采集**：定时器可以用于控制数据采集的时间间隔。
3. **控制系统**：定时器可以用于控制系统的状态和行为。

总之，定时器是嵌入式系统中的一个重要组件，用于实现定时功能和控制系统的行为。

本例程主要介绍了如何在 EtherKit 上使用 GPT 设备，也就是定时器，包括基本定时器的使用和 PWM 的使用；

8.2 硬件说明

本例程使用定时器功能，无需硬件连接。

8.3 软件说明

8.3.1 FSP 配置说明

FSP 分别配置使能 GPT0 为基本定时器模式，GPT5 为 PWM 模式；

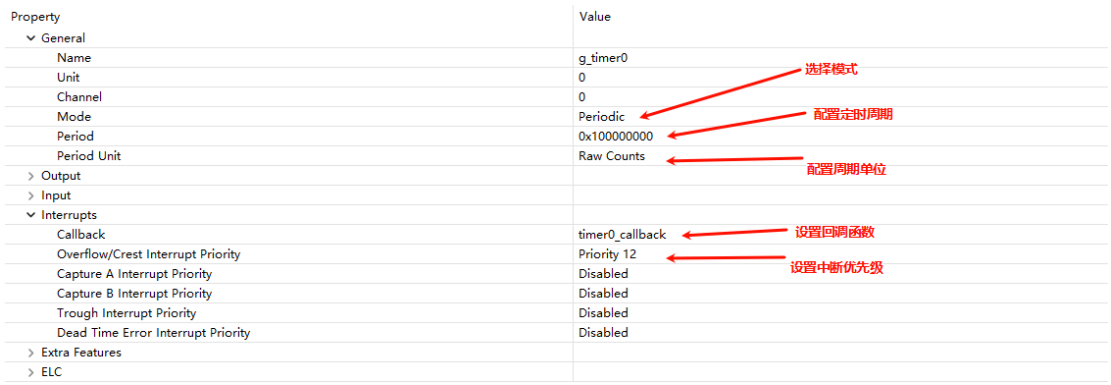


图 8-1 timer0 配置基本定时器模式

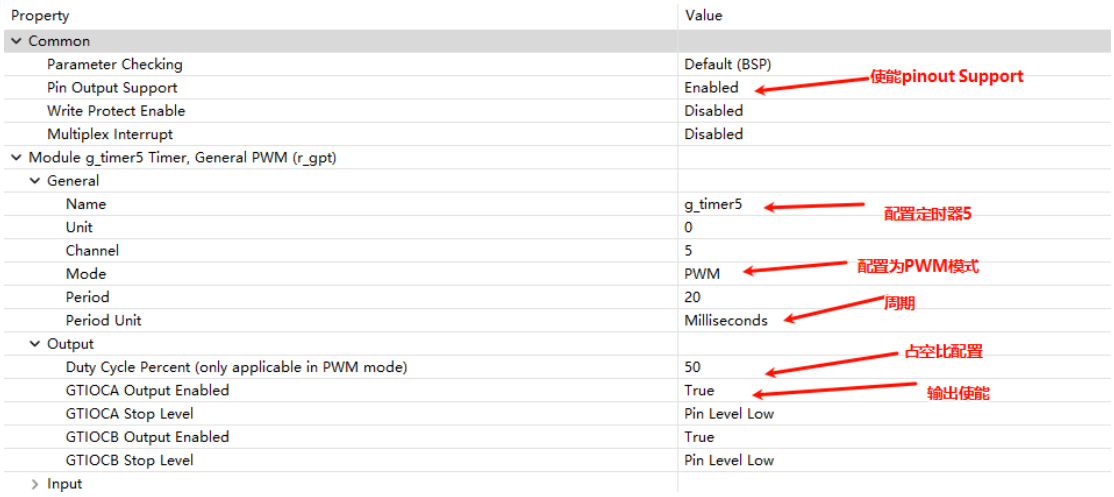


图 8-2 timer5 配置 pwm 模式

并配置 pins 使能 GPT0 GPT5；

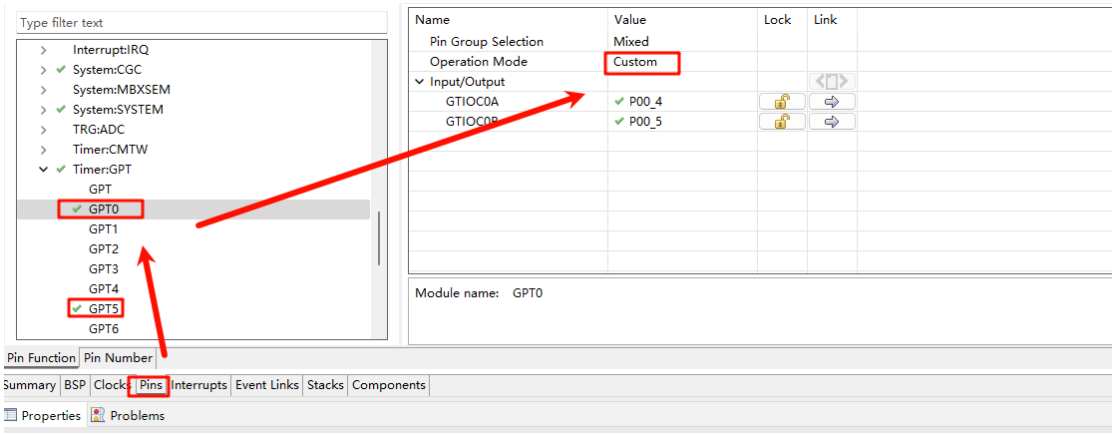


图 8-3 GPT 引脚配置

8.3.2 RT-Thread Settings 配置

在配置中打开 timer0 使能；

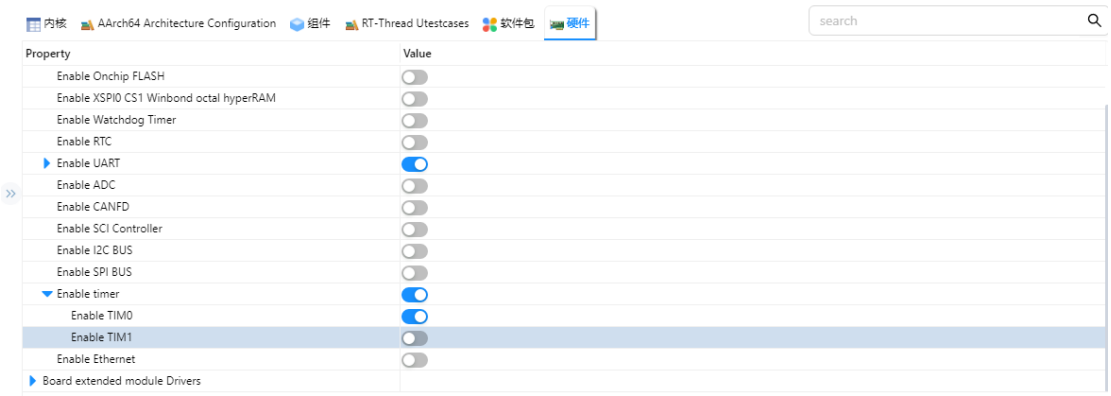


图 8-4 使能定时器

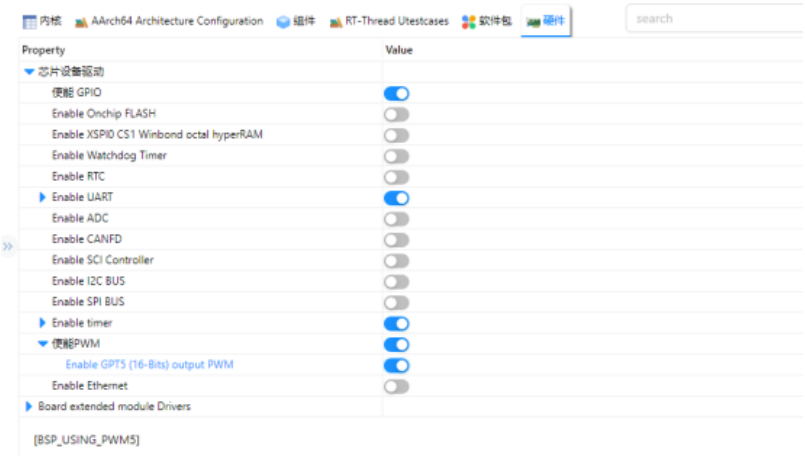


图 8-5 使能 PWM5

8.3.3 示例工程说明

本例程的源码位于/projects/etherkit_driver_gpt；

```
int hwtimer_sample(void)

{

    rt_err_t ret = RT_EOK;

    rt_hwtimerval_t timeout_s;

    rt_device_t hw_dev = RT_NULL;

    rt_hwtimer_mode_t mode;

    rt_uint32_t freq = 400000000; /* 1Mhz */

    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;
    }
    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;
    }
    rt_device_set_rx_indicate(hw_dev, timeout_cb);
    rt_device_control(hw_dev, HWTIMER_CTRL_FREQ_SET, &freq);
    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;
    }
    /* Example Set the timeout period of the timer */
    timeout_s.sec = 1;  /* secend */
    timeout_s.usec = 0; /* microsecend */
    if (rt_device_write(hw_dev, 0, &timeout_s, sizeof(timeout_s)) != siz
eof(timeout_s))
    {
        rt_kprintf("set timeout value failed\n");
        return -RT_ERROR;
    }
```

```
    /* read hwtimer value */
    rt_device_read(hw_dev, 0, &timeout_s, sizeof(timeout_s));
    rt_kprintf("Read: Sec = %d, Usec = %d\n", timeout_s.sec, timeout_s.u
sec);
    return ret;
}
MSH_CMD_EXPORT(hwtimer_sample, hwtimer sample);
```

每隔 1s 中触发一次中断回调函数打印输出，下面是 PWM 配置使能；

PWM 相关宏定义：当前版本的 PWM 驱动将每个通道都看做一个单独的 PWM 设备，每个设备都只有一个通道 0。使用 PWM5 设备，注意此处通道选择为 0 通道；

```
#define PWM_DEV_NAME        "pwm5"  /* PWM 设备名称 */
#define PWM_DEV_CHANNEL      0      /* PWM 通道 */
struct rt_device_pwm *pwm_dev;      /* PWM 设备句柄 */
```

配置 PWM 周期以及占空比：

```
static int pwm_sample(int argc, char *argv[])
{
    rt_uint32_t period, pulse, dir;
    period = 500000;    /* 周期为 0.5ms，单位为纳秒 ns */
    dir = 1;        /* PWM 脉冲宽度值的增减方向 */
    pulse = 100000;      /* PWM 脉冲宽度值，单位为纳秒 ns */

    /* 查找设备 */
    pwm_dev = (struct rt_device_pwm *)rt_device_find(PWM_DEV_NAME);
    if (pwm_dev == RT_NULL)
    {
        rt_kprintf("pwm sample run failed! can't find %s device!\n",
PWM_DEV_NAME);
        return RT_ERROR;
    }

    /* 设置 PWM 周期和脉冲宽度默认值 */
    rt_pwm_set(pwm_dev, PWM_DEV_CHANNEL, period, pulse);

    /* 使能设备 */
    rt_pwm_enable(pwm_dev, PWM_DEV_CHANNEL);
}
```

```
/* 导出到 msh 命令列表中 */
MSH_CMD_EXPORT(pwm_sample, pwm sample);
```

8.4 运行

8.4.1 编译&下载

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

编译完成后，将开发板的 Jlink 接口与 PC 机连接，然后将固件下载至开发板。

8.4.2 运行效果

每隔 1s 触发回调函数并打印输出；

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Nov 25 2024 14:51:10
 2006 - 2024 Copyright by RT-Thread team

Hello RT-Thread!
====================================================
This example project is an basic key routine!
====================================================
msh >hw
hwtimer_sample
msh >hwtimer_sample
Read: Sec = 1, Usec = 0
msh >this is hwtimer timeout callback fucntion!
tick is :3599 !
this is hwtimer timeout callback fucntion!
tick is :4599 !
this is hwtimer timeout callback fucntion!
tick is :5599 !
```

图 10-6 定时器示例运行

使用逻辑分析仪量取 pwm 输出波形如下所示；

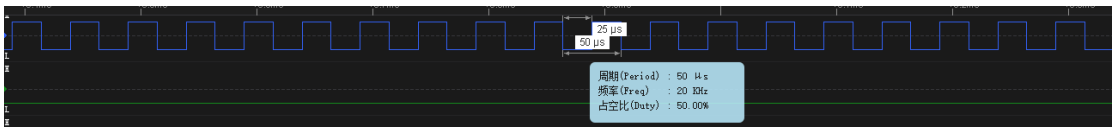


图 10-7 pwm 波形显示

8.5 注意事项

暂无

8.6 引用参考

- 设备与驱动：HWTIMER 设备

第 9 章 WDT 例程

9.1 简介

WDT（Watchdog Timer）是一种嵌入式系统中的硬件或软件组件，用于监视系统的运行状态和防止系统出现故障。WDT 功能的原理如下：

- **基本原理**

WDT 的基本原理是使用一个计数器来记录系统的运行时间。当系统正常运行时，计数器会被周期性地重置。如果系统出现故障或卡死，计数器不会被重置，会继续计数直到溢出。当计数器溢出时，WDT 会产生一个中断信号，通知系统进行相应的处理。

1. 硬件 WDT

硬件 WDT 通常使用一个独立的时钟信号和计数器来实现监视功能。当系统正常运行时，硬件 WDT 会被周期性地重置。如果系统出现故障或卡死，硬件 WDT 不会被重置，会继续计数直到溢出。

2. 软件 WDT

软件 WDT 则使用系统的时钟信号和软件算法来实现监视功能。软件 WDT 会周期性地检查系统的运行状态，并在系统出现故障或卡死时产生一个中断信号。

- **WDT 模式**

WDT 可以工作在以下几种模式：

1. **正常模式**：WDT 正常工作，监视系统的运行状态。
2. **测试模式**：WDT 用于测试系统的故障处理能力。
3. **禁用模式**：WDT 被禁用，不监视系统的运行状态。

- **WDT 应用**

WDT 在嵌入式系统中有许多应用，例如：

1. **系统故障处理**：WDT 可以用于处理系统故障和异常。
2. **系统重启**：WDT 可以用于重启系统。
3. **系统监视**：WDT 可以用于监视系统的运行状态。

总之，WDT 可以保证我们的代码在我们的预期中进行，可以有效防止我们的程序因为一些其它不可控因素导致代码”跑飞“；本例程主要介绍了如何在 EtherKit 上使用 WDT 设备;

9.2 硬件说明

本例程使用看门狗功能，无需硬件连接。

9.3 软件说明

9.3.1 FSP 配置说明

打开 FSP 工具 新建 Stacks 选择 r_wdt，并配置窗口看门狗的时间窗口；

Property	Value
Common	
Parameter Checking	Default (BSP)
Multiplex Interrupt	Disabled
Module g_wdt Watchdog (r_wdt)	
Name	g_wdt
Timeout	16,384 Cycles
Clock Division Ratio	PCLK/8192
Window Start Position	100 (Window Position Not Specified)
Window End Position	0 (Window Position Not Specified)
WDT Callback	NULL

图 9-1 看门狗配置

9.3.2 RT-Thread Settings 配置

打开 RT-Thread Settings，选择硬件配置并使能看门狗；

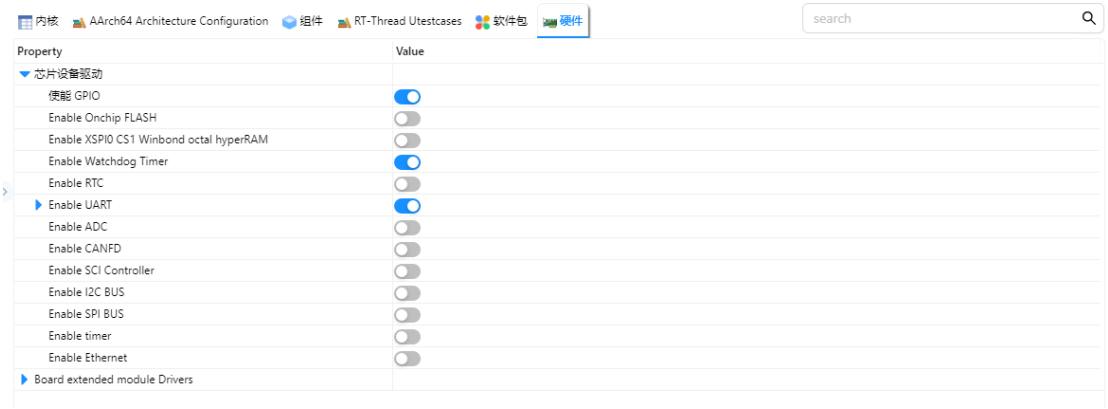


图 9-2 Settings 配置图

9.3.3 示例工程说明

本例程的源码位于/projects/etherkit_driver_wdt，通过在空闲函数中执行喂狗操作；

```
static void idle_hook(void)
{
    /* 在空闲线程的回调函数里喂狗 */
    rt_device_control(wdg_dev, RT_DEVICE_CTRL_WDT_KEEPALIVE, NULL);
    rt_kprintf("feed the dog!\n ");
}
static int wdt_test(int argc, char *argv[])
{
    rt_err_t ret = RT_EOK;
    char device_name[RT_NAME_MAX];
    /* 判断命令行参数是否给定了设备名称 */
    if (argc == 2)
    {
        rt_strncpy(device_name, argv[1], RT_NAME_MAX);
    }
    else
    {
```

```
        rt_strncpy(device_name, WDT_DEVICE_NAME, RT_NAME_MAX);
    }
    /* 根据设备名称查找看门狗设备，获取设备句柄 */
    wdg_dev = rt_device_find(device_name);
    if (!wdg_dev)
    {
        rt_kprintf("find %s failed!\n", device_name);
        return RT_ERROR;
    }
    /* 初始化设备 */
     rt_device_init(wdg_dev);
    /* 启动看门狗 */
    ret = rt_device_control(wdg_dev, RT_DEVICE_CTRL_WDT_START, RT_NULL);
    if (ret != RT_EOK)
    {
        rt_kprintf("start %s failed!\n", device_name);
        return -RT_ERROR;
    }
    /* 设置空闲线程回调函数 */
    rt_thread_idle_sethook(idle_hook);
    return ret;
}
MSH_CMD_EXPORT(wdt_test, wdt sample);
```

9.4 运行

9.4.1 编译&下载

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

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

编译完成后，将开发板的 Jlink 接口与 PC 机连接，然后将固件下载至开发板。

9.4.2 运行效果

打开串口工具，在命令行终端运行 wdt_test，执行看门狗示例；

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Nov 25 2024 14:54:44
 2006 - 2024 Copyright by RT-Thread team

Hello RT-Thread!
================================================
This example project is an driver wdt routine!
================================================
msh >wdt
wdt_test
msh >wdt_test
msh >feed the dog!
 feed the dog!
 feed the dog!
 feed the dog!
 feed the dog!
 feed the dog!
 feed the dog!
 feed the dog!
 feed the dog!
```

图 9-3 WDT 示例运行效果

9.5 注意事项

暂无

9.6 引用参考

- 设备与驱动：WDT 设备

第 10 章 RS485 例程

10.1 简介

RS485 是一种串行通信协议，用于连接多个设备到同一条总线上。RS485 功能的原理如下：

- **基本原理**

RS485 使用两条线来进行通信：数据线（Data+和 Data-）。数据线用于传输数据信号，包括发送和接收数据。

- **通信过程**

RS485 通信过程如下：

1. **设备连接**：多个设备连接到同一条 RS485 总线上。
2. **数据传输**：设备发送数据到总线上，其他设备可以接收数据。
3. **数据接收**：设备接收数据从总线上，其他设备可以发送数据。

- **RS485 模式**

RS485 可以工作在以下几种模式：

1. **单主机模式**：一个主机设备控制总线，其他设备作为从机。
2. **多主机模式**：多个主机设备控制总线，其他设备作为从机。

- **RS485 特点**

RS485 具有以下特点：

1. **高速通信**：RS485 支持高速通信，最高可达 10Mbps。
2. **远距离通信**：RS485 支持远距离通信，最高可达 1200 米。
3. **多设备连接**：RS485 支持多设备连接，最高可达 32 个设备。

总之，RS485 是一种串行通信协议，用于连接多个设备到同一条总线上。RS485 具有高速通信、远距离通信和多设备连接等特点，广泛应用于嵌入式系统中。

本例程主要介绍了如何在 EtherKit 上使用 RS485 设备；

10.2 硬件说明

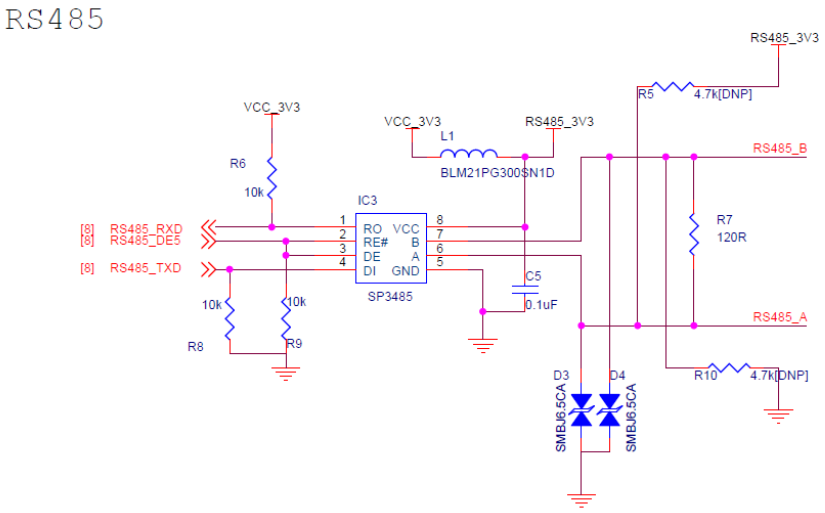


图 10-1 rs485 硬件原理图示

10.3 软件说明

10.3.1 FSP 配置说明

打开 FSP 工具 新建 Stacks 选择 r_sci_uart5，具体配置信息如下；

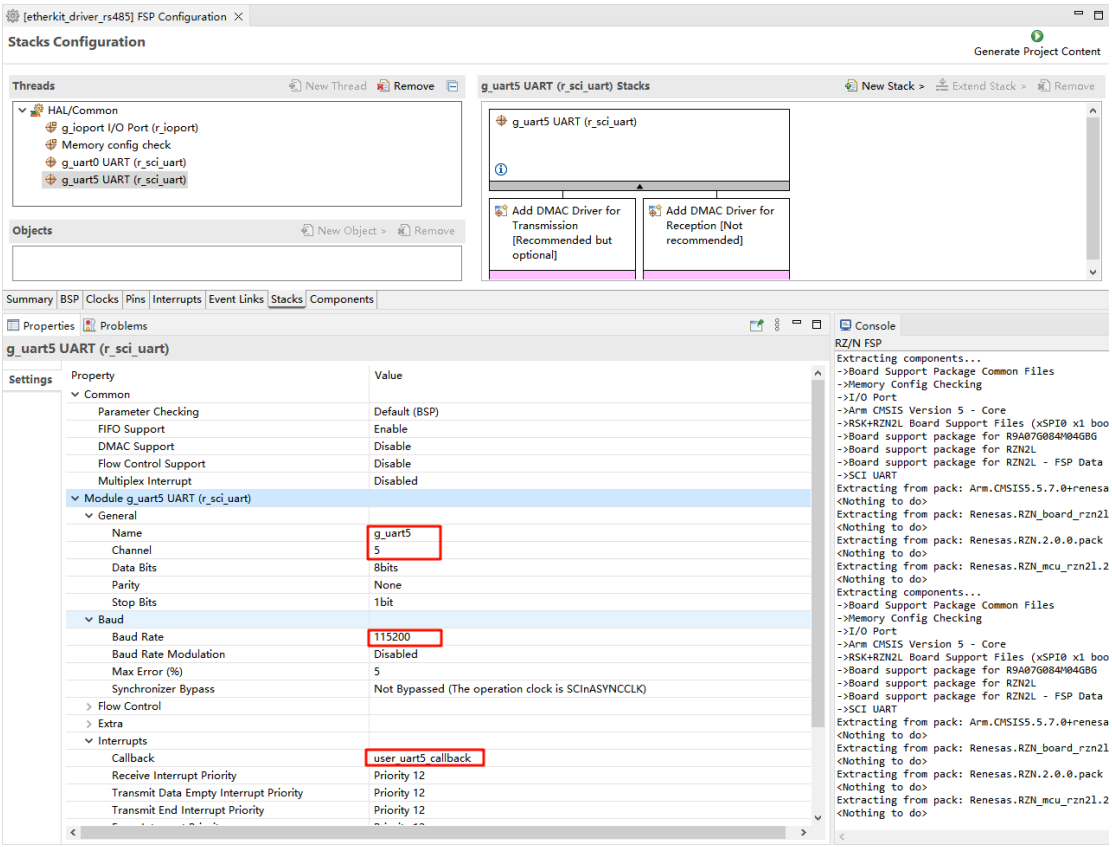


图 10-2 串口配置

10.3.2 工程示例说明

初始化 RS485 驱动，在 Finsh 终端打印从 rs485 串口终端来的字符，并且回显在 rs485 终端。

10.4 运行

10.4.1 编译&下载

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

编译完成后，将开发板的 Jlink 接口与 PC 机连接，然后将固件下载至开发板。

10.4.2 运行效果

串口输出指令 rs485_sample 指令，打开 rs485 串口终端查看收到的数据：

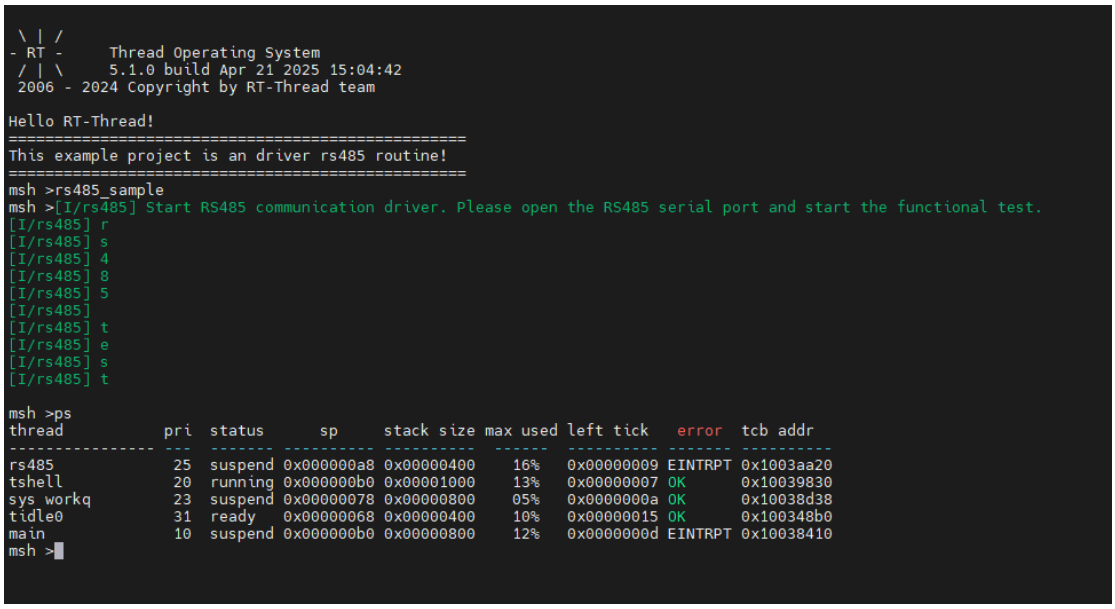


图 10-3 rs485 接收

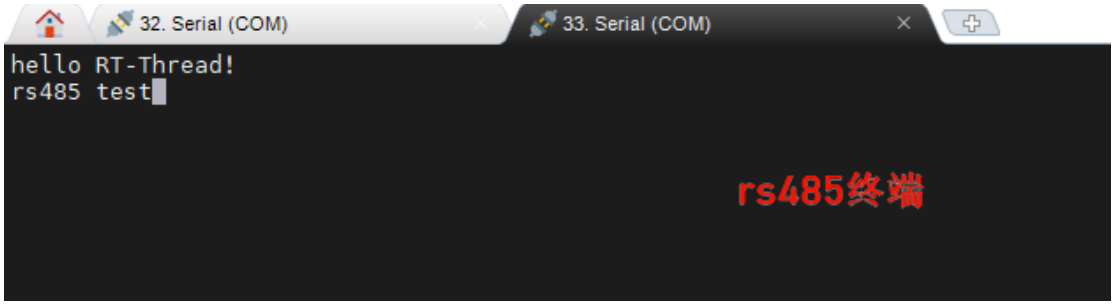


图 10-4 rs485 发送

10.5 注意事项

暂无

10.6 引用参考

- 设备与驱动：UART_V2 设备

第 11 章 以太网例程

11.1 简介

本例程主要介绍了如何在 EtherKit 上使用以太网进行网络连接。单片机以太网技术是现代嵌入式系统中广泛应用的通信技术，通过将以太网功能集成到单片机中，使设备能够方便地接入局域网或互联网，满足工业自动化、物联网、智能家居等领域对高速通信的需求。

11.2 硬件说明

EtherKit 使用的以太网芯片为 RTL8211，RTL8211 系列是由瑞昱半导体（Realtek Semiconductor）推出的一系列高性能千兆以太网 PHY（物理层）芯片，下面来看下以太网相关原理图设计：

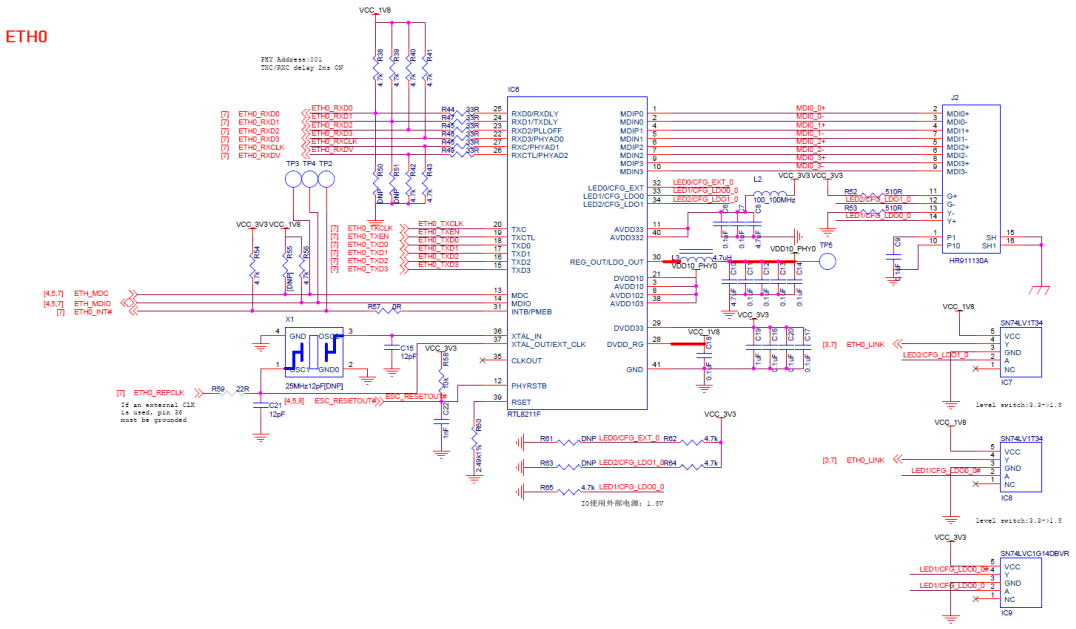


图 11-1 ETH0

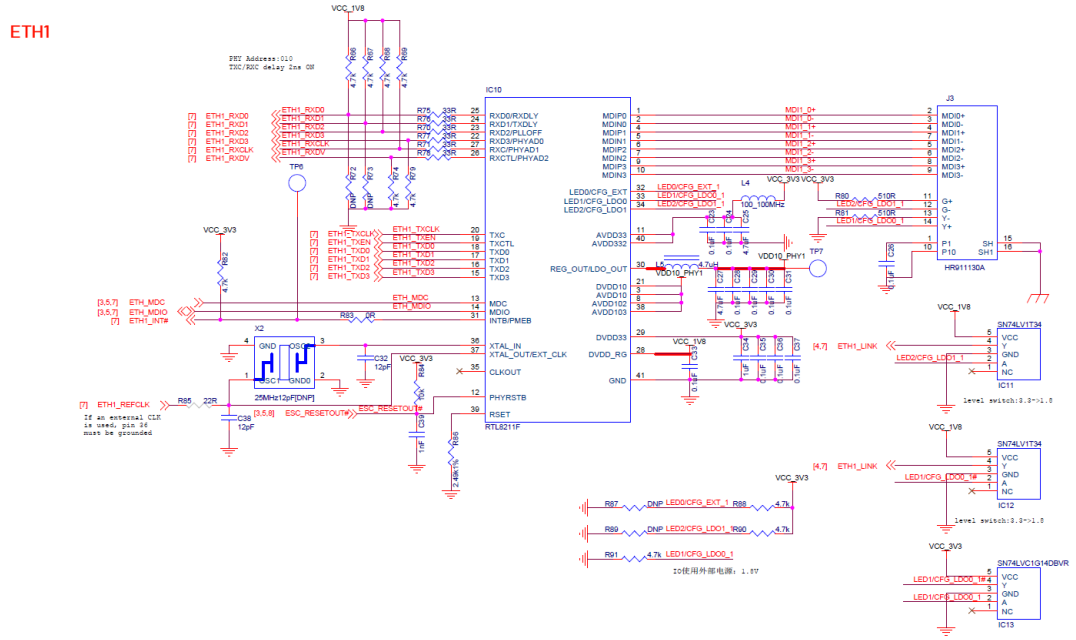


图 11-2 ETH1

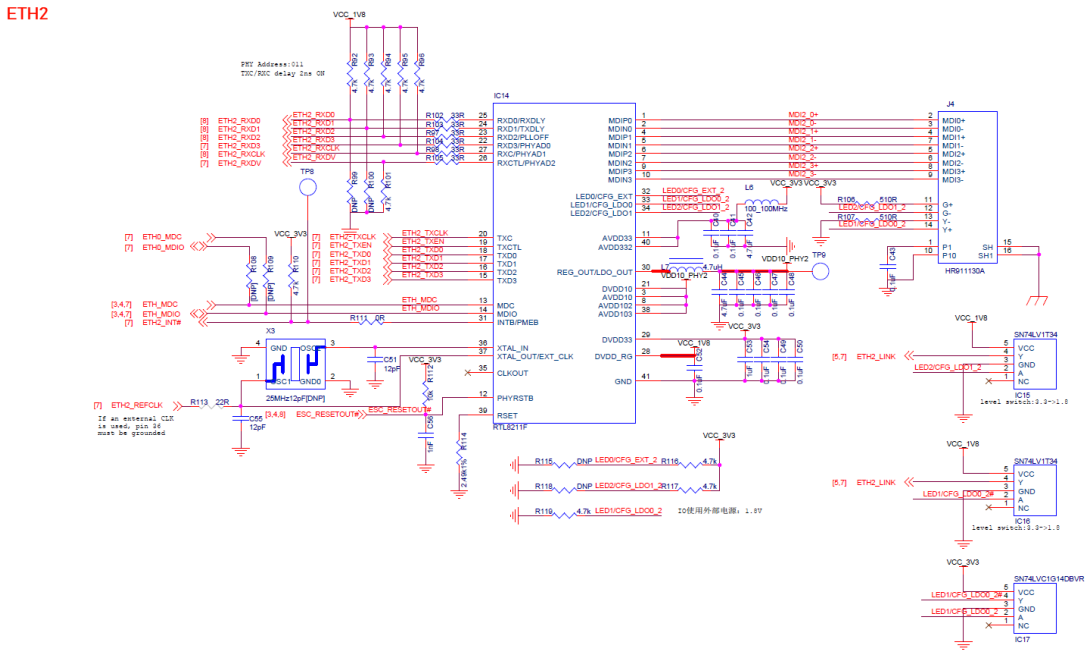


图 11-3 ETH2

11.3 软件说明

11.3.1 FSP 配置

打开工程配置文件 configuration.xml：

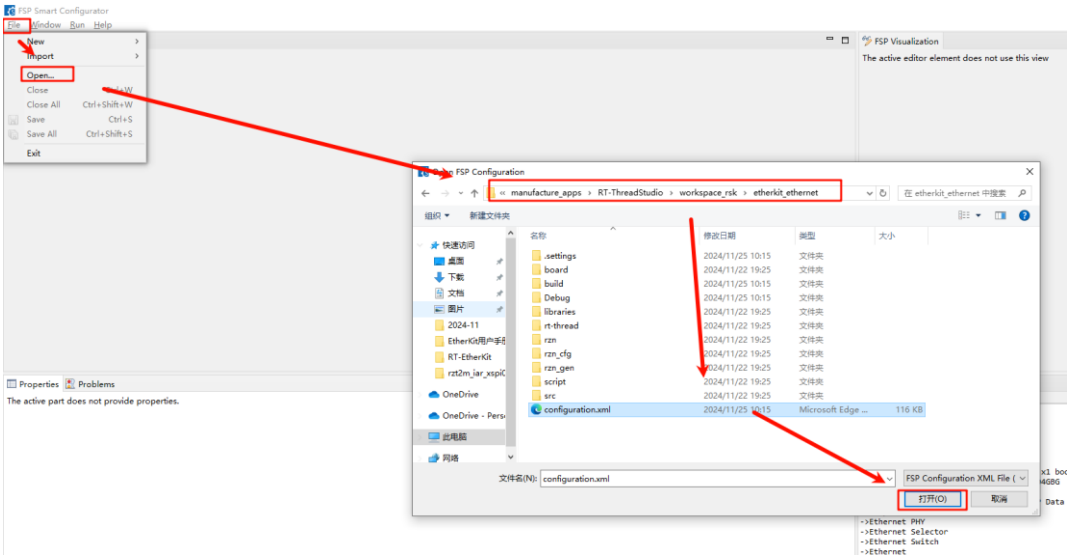


图 11-4 打开配置文件

新增 r_gamc Stack；

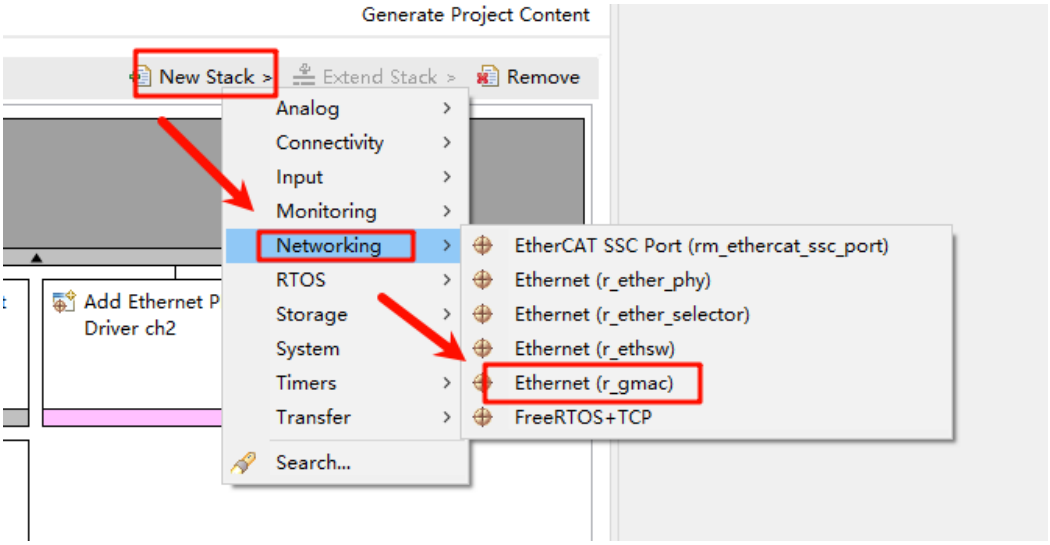


图 11-5 r_gmac stack

点击 g_ether0 Ethernet，配置中断回调函数为 user_ether0_callback；

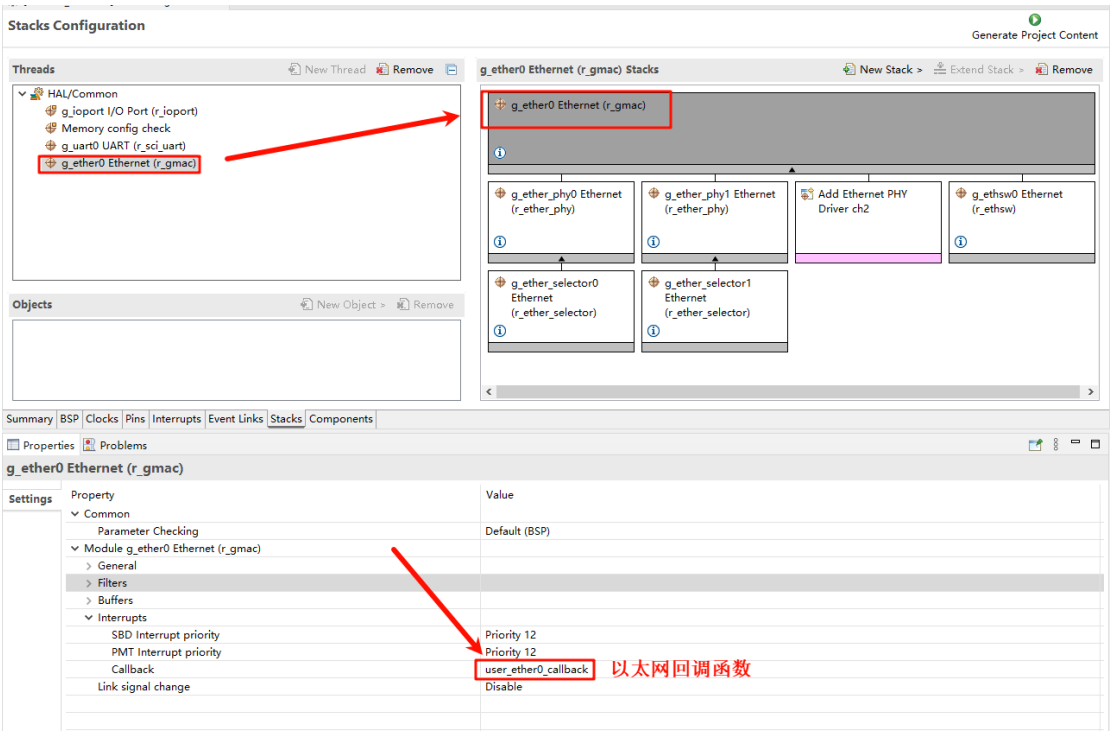


图 11-6 以太网回调函数

下面配置 phy 信息，选择 g_ether_phy0，Common 配置为 User Own Target；修改 PHY LSI 地址为 1（根据原理图查询具体地址）；设置 phy 初始化回调函数为 ether_phy_targets_initialize_rtl8211_rgmii()；同时设置 MDIO 为 GMAC。

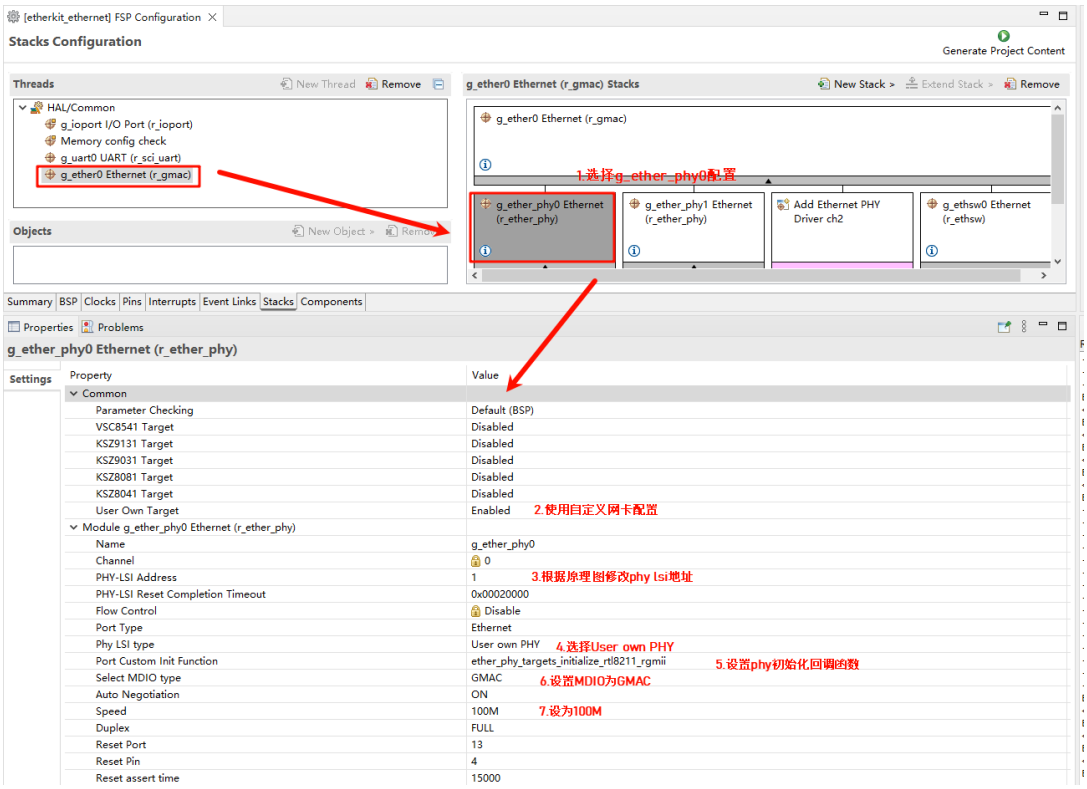


图 11-7 PHY 配置

配置 g_ether_selector0，选择以太网模式为交换机模式，PHY link 设置为默认 active-low，PHY 接口模式设置为 RGMII。

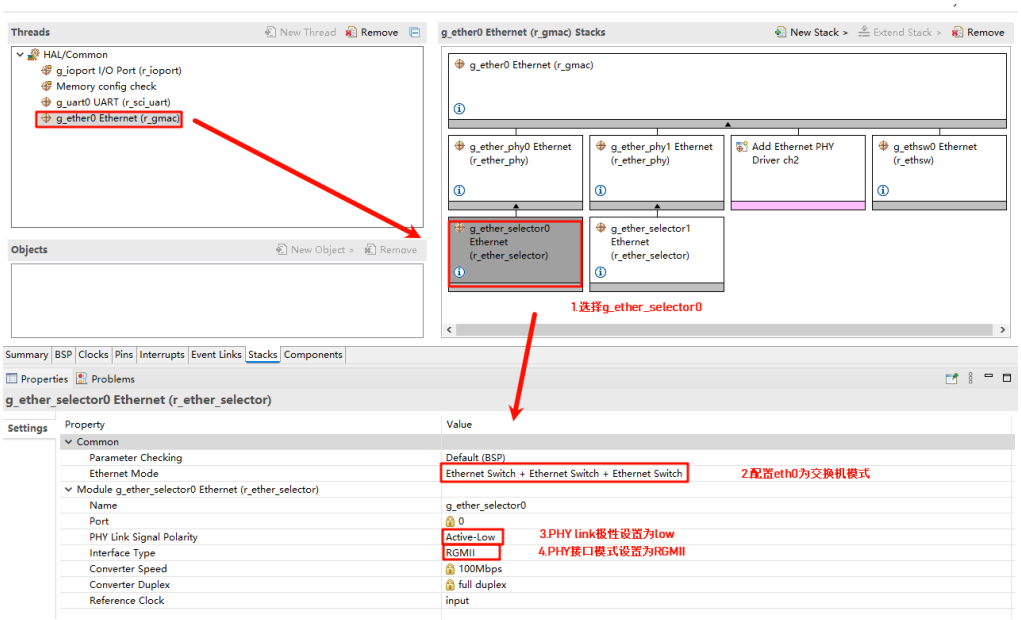


图 11-8 selector 配置

网卡引脚参数配置，选择操作模式为 RGMII：

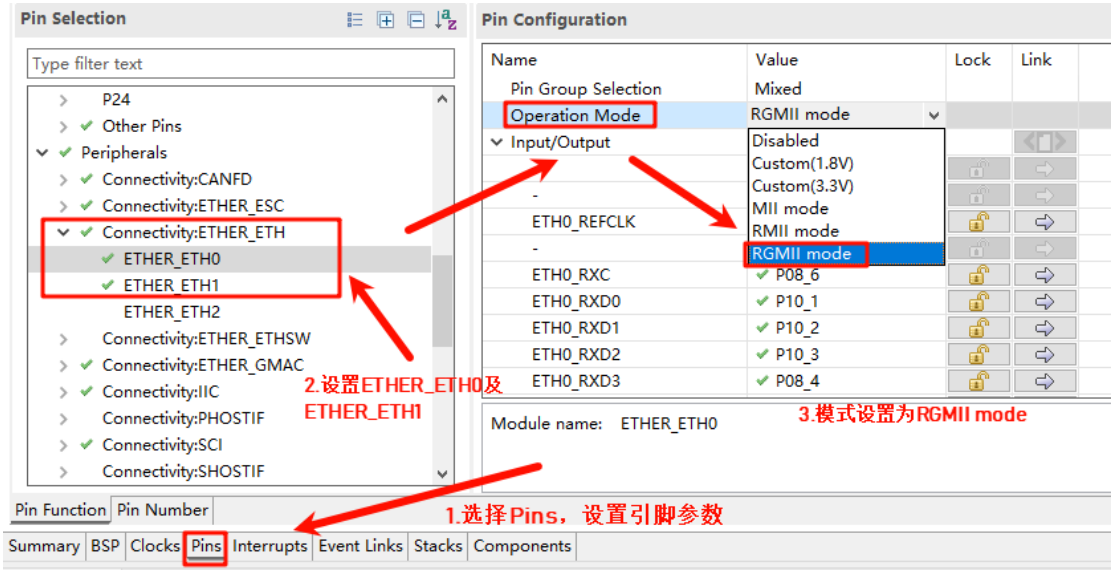


图 11-9 ETH 引脚配置

ETHER_GMAC 配置：

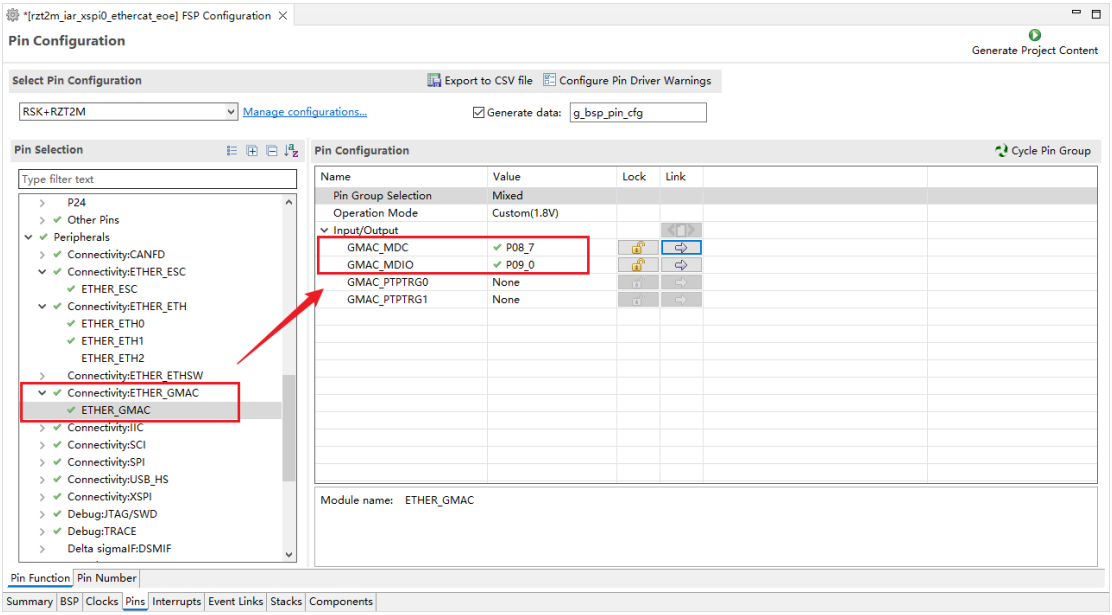


图 11-10 GMAC 引脚配置

11.3.2 RT-Thread Settings 配置

回到 Studio 工程，配置 RT-Thread Settings，点击选择硬件选项，找到芯片设备驱动，使能以太网；

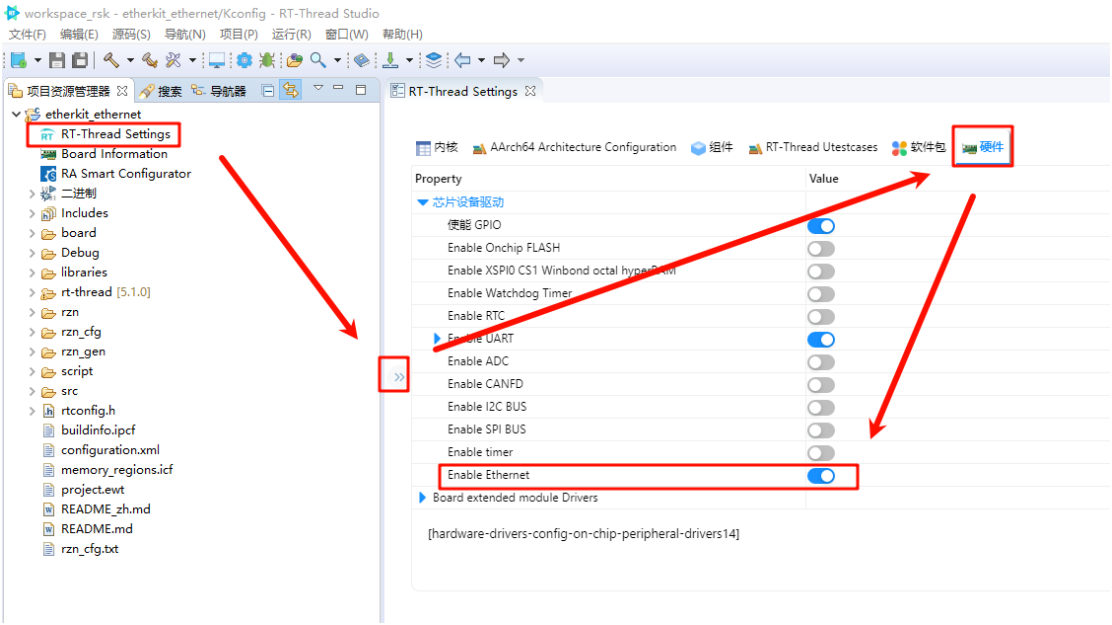


图 11-11 settings 配置

11.3.3 示例代码说明

将以太网网口连接至交换机，使用 netdev 的 ping 功能进行联通性测试；

11.4 运行

11.4.1 编译&下载

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

编译完成后，将开发板的 Jlink 接口与 PC 机连接，然后将固件下载至开发板。

11.4.2 运行效果

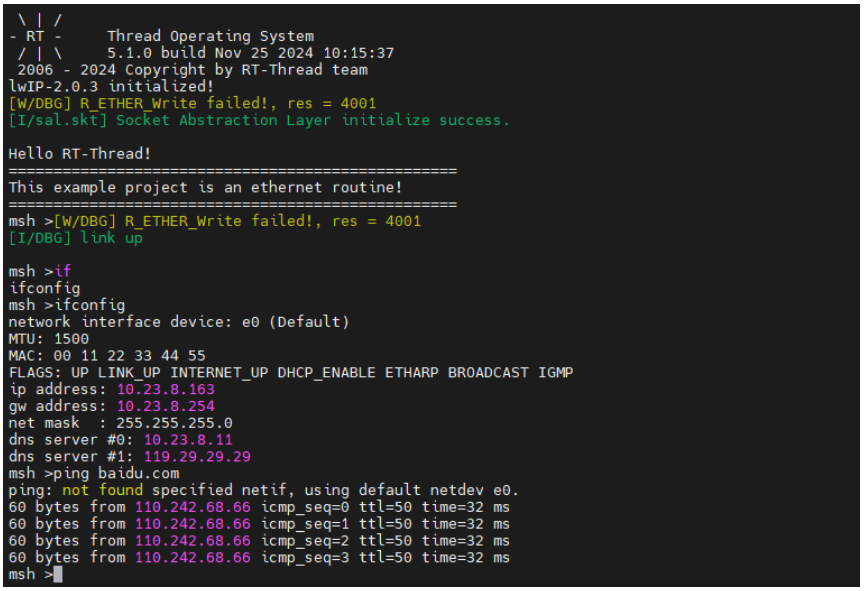


图 11-11 以太网 ping 命令

11.5 注意事项

暂无

11.6 引用参考

暂无

第 12 章 CANFD 例程

12.1 简介

CANFD（Controller Area Network Flexible Data-Rate）是一种汽车网络通信协议，用于连接多个电子控制单元（ECU）到同一条网络上。CANFD 功能的原理如下：

- **基本原理**

CANFD 使用一种称为 CSMA/CA（载波侦听多路访问/冲突避免）的协议来管理数据传输。该协议允许多个 ECU 共享同一条网络，并且可以自动检测和避免数据冲突。

- **通信过程**

CANFD 通信过程如下：

1. **ECU 连接**：多个 ECU 连接到同一条 CANFD 网络上。
2. **数据传输**：ECU 发送数据到网络上，其他 ECU 可以接收数据。
3. **数据接收**：ECU 接收数据从网络上，其他 ECU 可以发送数据。

- **CANFD 模式**

CANFD 可以工作在以下几种模式：

1. **标准模式**：CANFD 工作在标准模式下，数据传输速率为 500kbps。
2. **高速模式**：CANFD 工作在高速模式下，数据传输速率为 1000kbps 或更高。

- **CANFD 特点**

CANFD 具有以下特点：

1. **高速通信**：CANFD 支持高速通信，最高可达 1000kbps。

2. **远距离通信**：CANFD 支持远距离通信，最高可达 100 米。

3. **多 ECU 连接**：CANFD 支持多 ECU 连接，最高可达 128 个 ECU。

总之，CANFD 是一种汽车网络通信协议，用于连接多个 ECU 到同一条网络上。CANFD 具有高速通信、远距离通信和多 ECU 连接等特点，广泛应用于嵌入式系统中。

本例程主要介绍了如何在 EtherKit 上使用 canfd 设备。

12.2 硬件说明

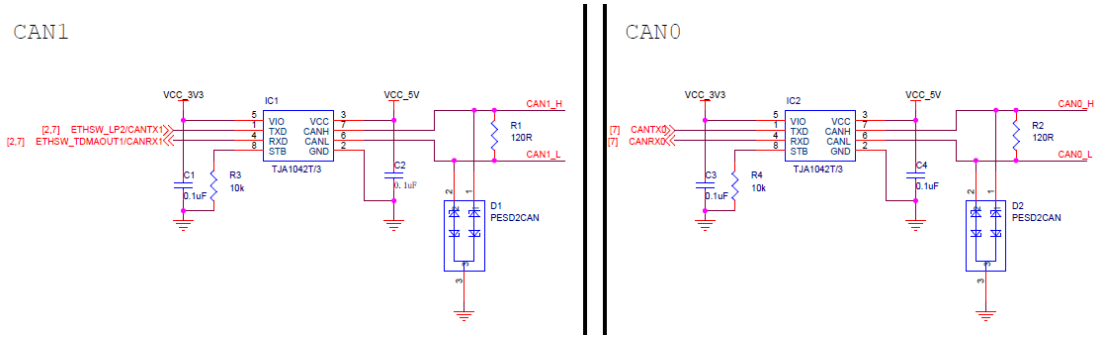


图 12-1 canfd 原理图示

12.3 软件说明

12.3.1 FSP 配置说明

选择新建工程下的 configuration.xml 文件，在 rzn-fsp 中打开；

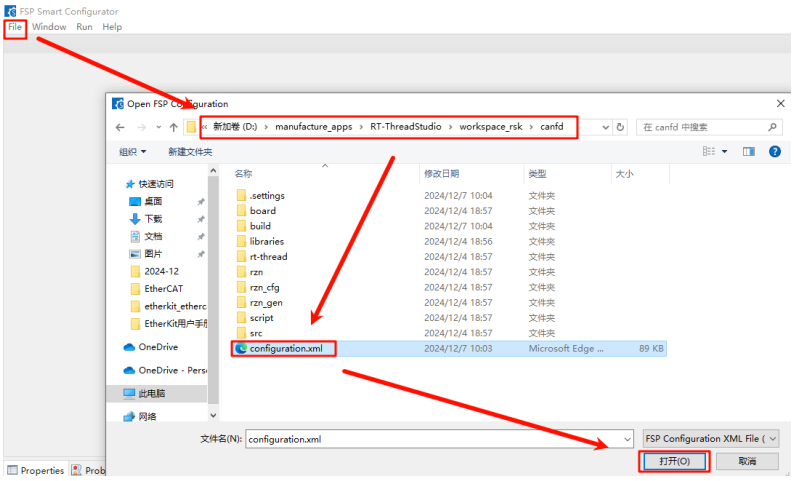


图 12-2 导入工程

点击添加 New Stack，搜索 canfd 并添加 r_canfd，这里我们需要添加两个 canfd_stack；

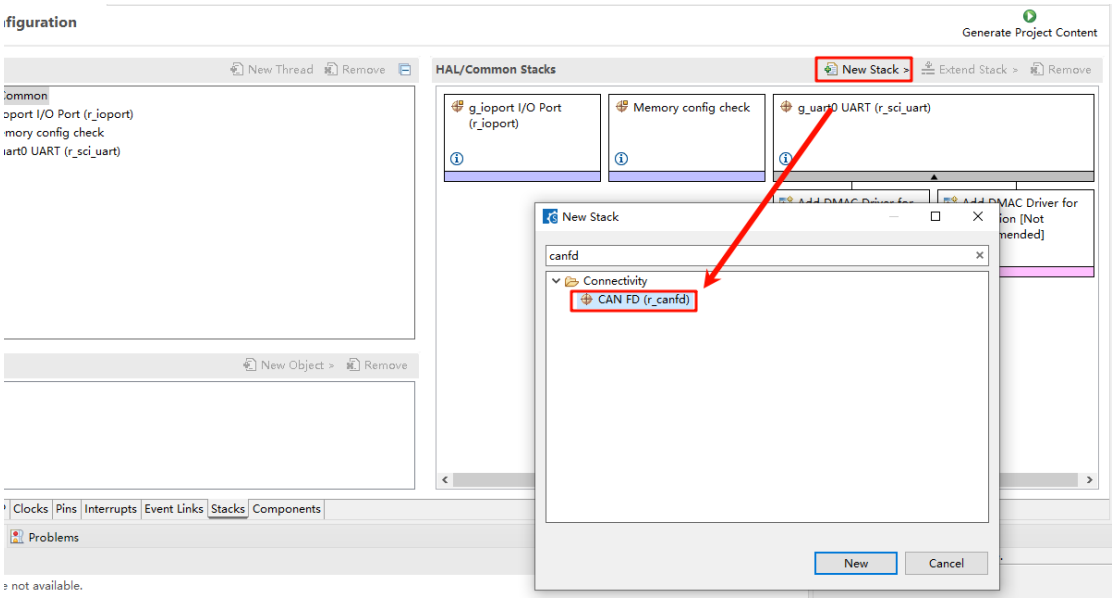


图 12-3 添加 canfd stack

在基本配置中我们为 canfd0 和 canfd1 分别使能接收 FIFO 中断，依次选择 Common->Reception->FIFOs->FIFO 0 / FIFO 1->Enable，其中为 canfd0 使能 FIFO 0 中断，canfd1 使能 FIFO 1 中断：

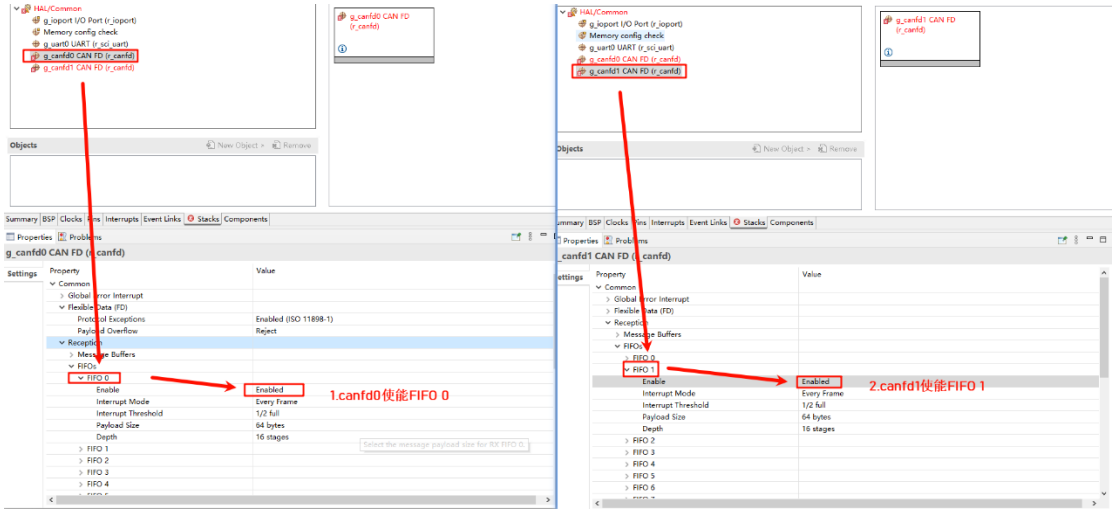


图 12-4 使能 canfd FIFO 中断

接下来我们需要为 CANFD 分别设置通道、中断回调函数及过滤器数组；

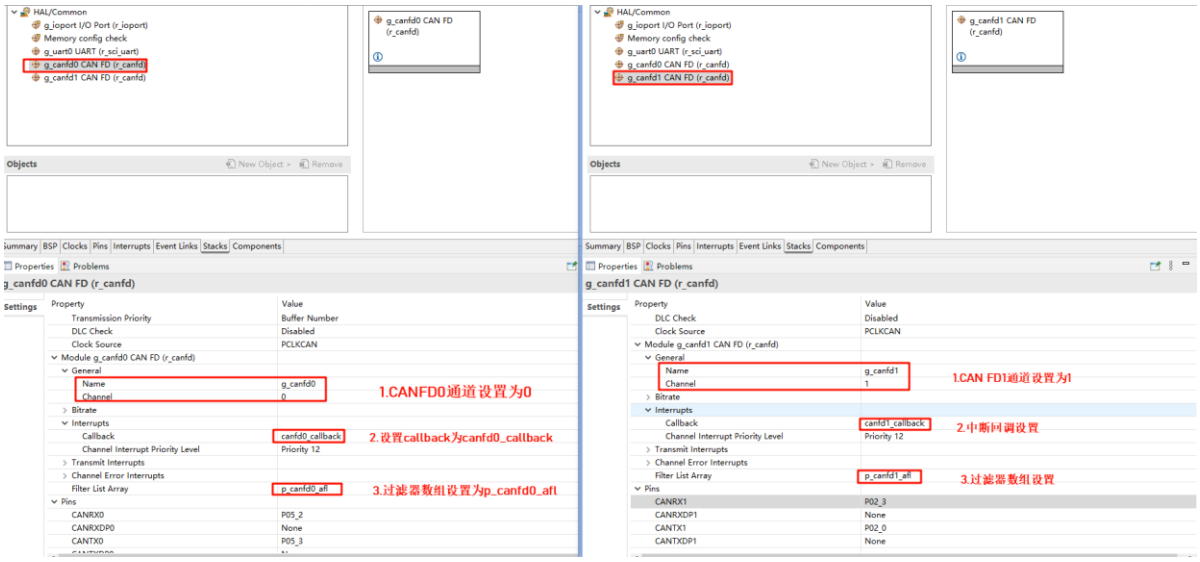


图 12-5 canfd stack 配置

对 CANFD 的引脚进行配置和使能；

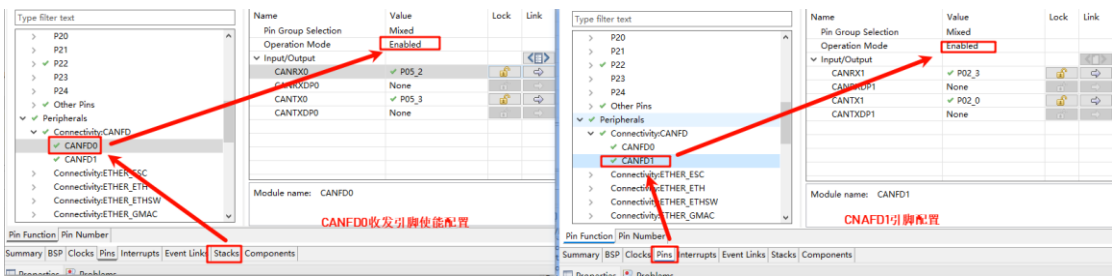


图 12-6 canfd 引脚配置

接下来还需要使能发送消息缓冲区中断配置，这能决定当传输完成时应该是哪个消息缓冲区触发中断：

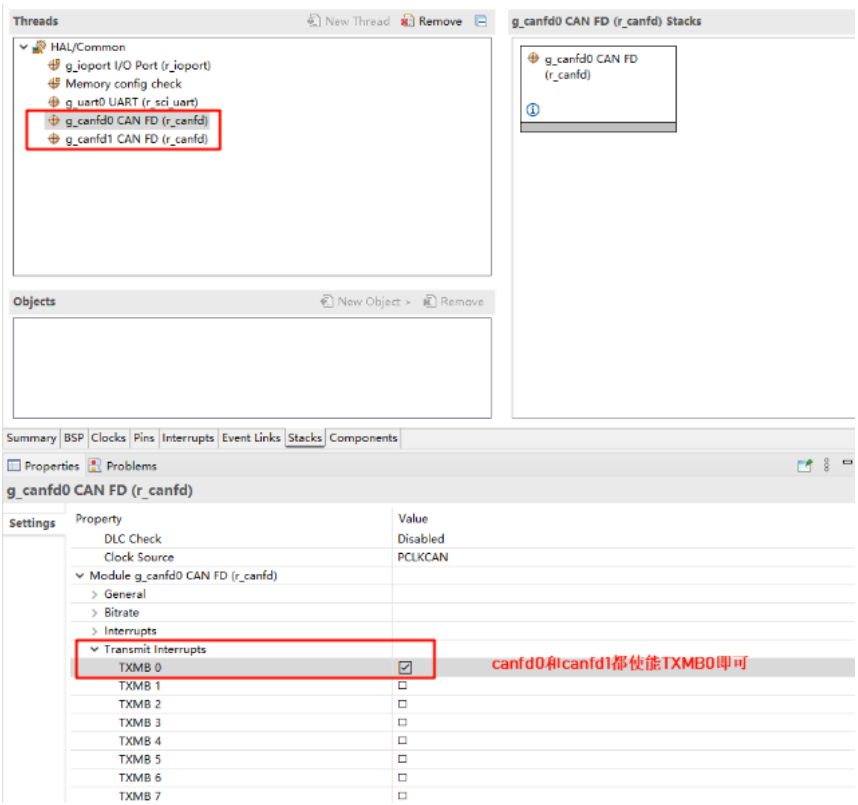


图 12-7 配置 TXMB

12.3.2 RT-Thread Settings 配置

打开 RT-Thread Settings，使能 canfd 配置；

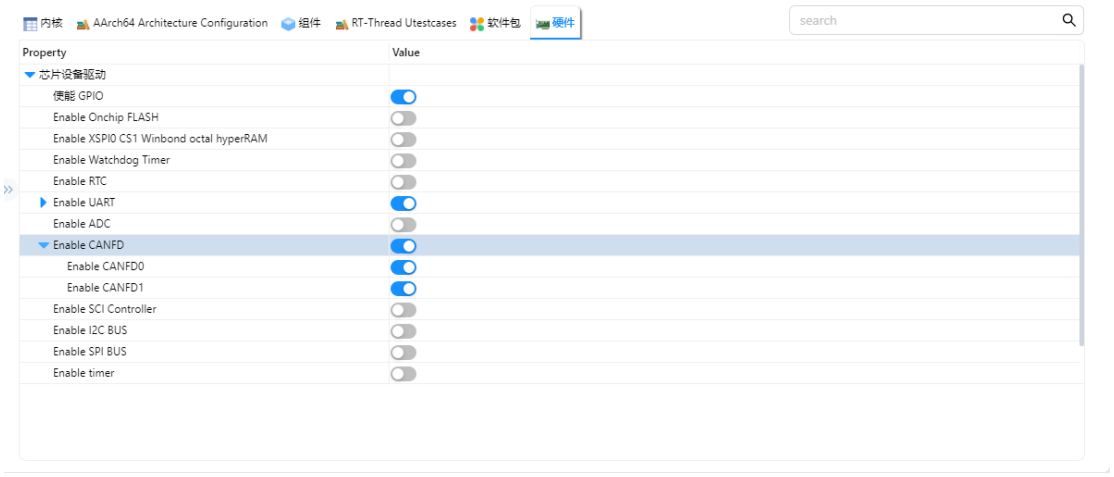


图 12-8 Settings 配置图

12.3.3 工程示例说明

工程通过 canfd0 发送报文 canfd1 接受报文并将其使用串口打印；发送代码示例如下；

```
int can0_sample_send(int argc, char *argv[])
{
    rt_err_t res;
    rt_thread_t thread;
    char can_name[RT_NAME_MAX];
    if (argc == 2)
    {
        rt_strncpy(can_name, argv[1], RT_NAME_MAX);
    }
    else
    {
        rt_strncpy(can_name, CAN0_DEV_NAME, RT_NAME_MAX);
    }
    /* 查找 CAN 设备 */
    can0_dev = rt_device_find(can_name);
```

```
    if (!can0_dev)
    {
        rt_kprintf("find %s failed!\n", can_name);
        return RT_ERROR;
    }
    /* 以中断接收及发送方式打开 CAN 设备 */
    res = rt_device_open(can0_dev, RT_DEVICE_FLAG_INT_TX | RT_DEVICE_FLA
G_INT_RX);
    RT_ASSERT(res == RT_EOK);
    /* 创建数据接收线程 */
    thread = rt_thread_create("can0_tx", can0_tx_thread, RT_NULL, 1024,
25, 10);
    if (thread != RT_NULL)
    {
        rt_thread_startup(thread);
    }
    else
    {
        rt_kprintf("create can_rx thread failed!\n");
    }
    return res;
}
/* 导出到 msh 命令列表中 */
MSH_CMD_EXPORT(can0_sample_send, can device sample);
int can1_sample_receive(int argc, char *argv[])
{
    rt_err_t res;
    rt_thread_t thread;
    char can_name[RT_NAME_MAX];
    if (argc == 2)
```

```
    {
        rt_strncpy(can_name, argv[1], RT_NAME_MAX);
    }
    else
    {
        rt_strncpy(can_name, CAN1_DEV_NAME, RT_NAME_MAX);
    }
    /* 查找 CAN 设备 */
    can1_dev = rt_device_find(can_name);
    if (!can1_dev)
    {
        rt_kprintf("find %s failed!\n", can_name);
        return RT_ERROR;
    }
    /* 初始化 CAN 接收信号量 */
    rt_sem_init(&rx_sem, "rx_sem", 0, RT_IPC_FLAG_FIFO);


    /* 以中断接收及发送方式打开 CAN 设备 */
    res = rt_device_open(can1_dev, RT_DEVICE_FLAG_INT_TX | RT_DEVICE_FLA
G_INT_RX);
    RT_ASSERT(res == RT_EOK);
    /* 创建数据接收线程 */
    thread = rt_thread_create("can1_rx", can1_rx_thread, RT_NULL, 1024,
25, 10);
    if (thread != RT_NULL)
    {
        rt_thread_startup(thread);
    }
    else
    {
```

```
        rt_kprintf("create can_rx thread failed!\n");
    }
    return res;
}
```

12.4 运行

12.4.1 编译&下载

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

编译完成后，将开发板的 Jlink 接口与 PC 机连接，然后将固件下载至开发板。

12.4.2 运行效果

将 CAN0_L 与 CAN1_L 对接，CAN0_H 与 CAN1_H 对接，如下图所示；

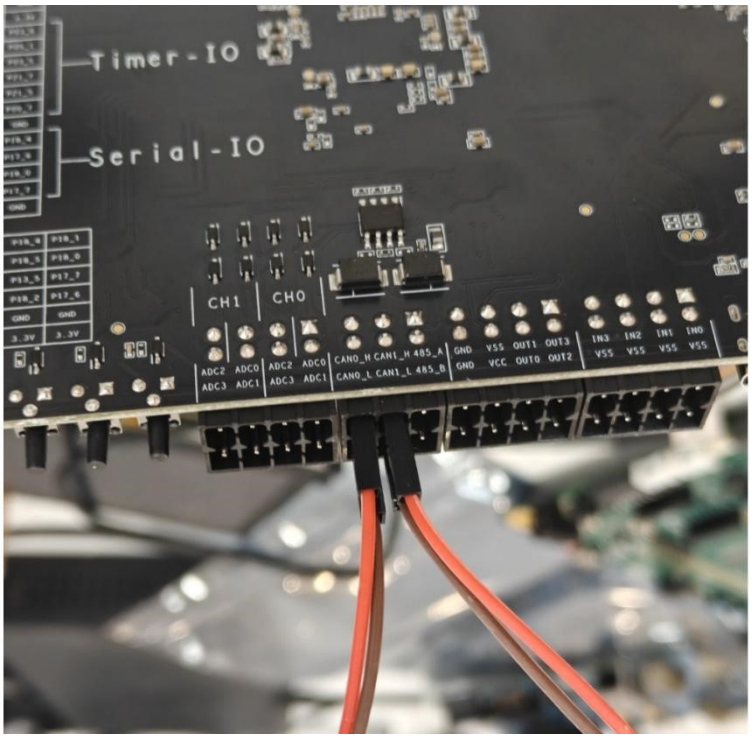


图 12-9 canfd 测试接线示意

使用串口分别发送 can0_sample_send 和 can1_sample_receive 命令进行回环测试；

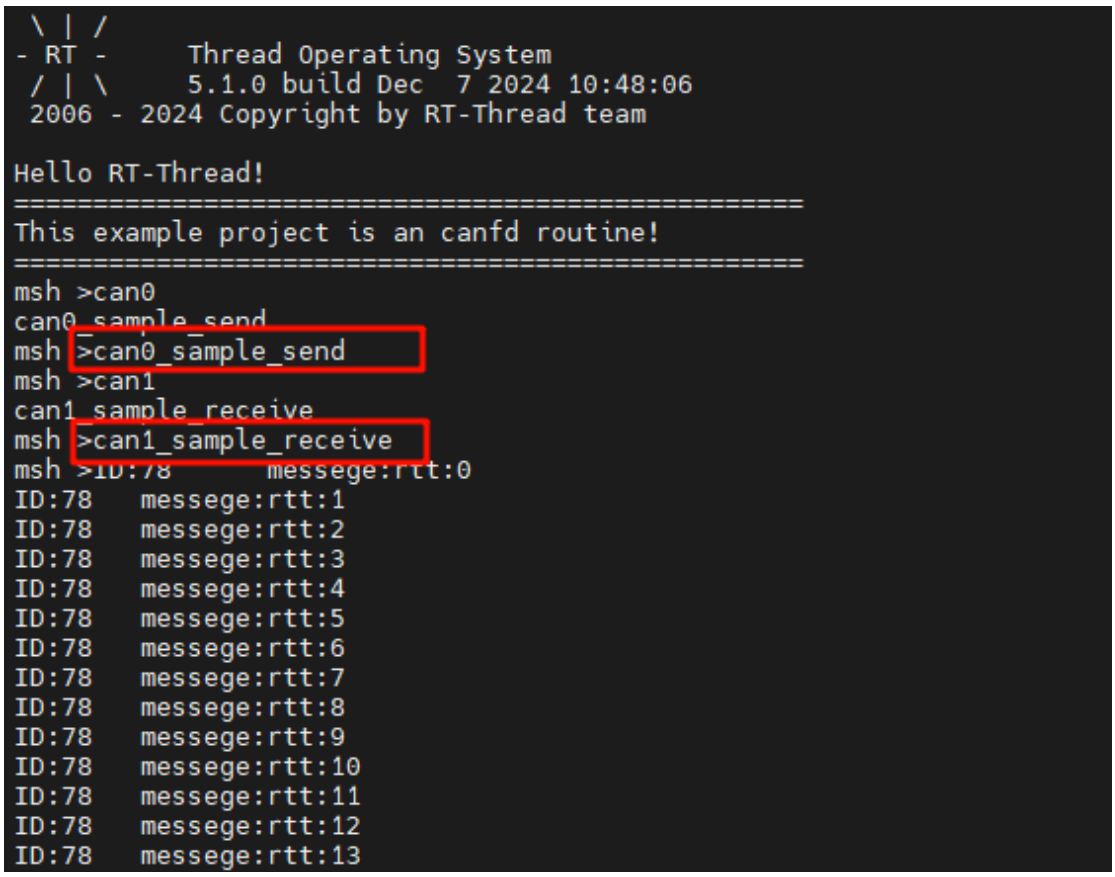


图 12-5 canfd 回环测试

12.5 注意事项

暂无

12.6 引用参考

- 设备与驱动：CAN 设备

第 13 章 Netutils 例程

13.1 简介

本工程提供 ethernet 的基础功能，包括 ping 、tftp、ntp 、iperf 等功能。

13.2 硬件说明

需要使用网线连接到开发板的三网口其中任意一个网口，另一头连接到可以联网的交换机上。

13.3 软件说明

13.3.1 FSP 配置说明

此处配置请参考第 11 章：11.3.1 FSP 配置。

13.3.2 RT-Thread Settings 配置

回到 Studio 工程，配置 RT-Thread Settings，点击选择硬件选项，找到芯片设备驱动，使能以太网；

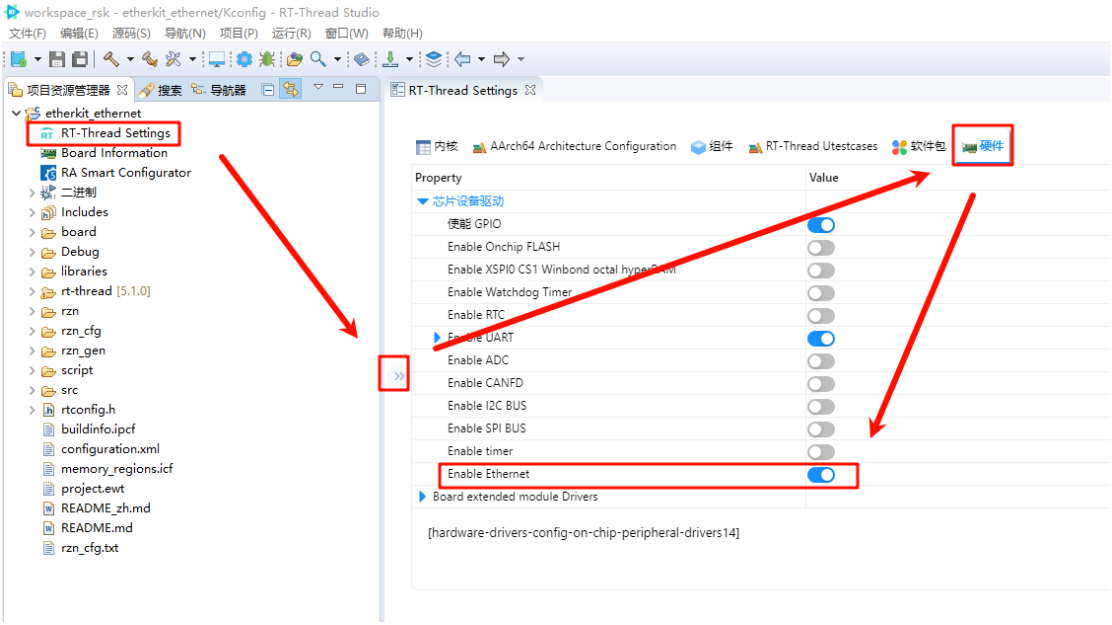


图 13-1 以太网使能

打开 RT-Thread Settings，软件包搜索 netutils 并使能 tftp、iperf、ntp 功能；

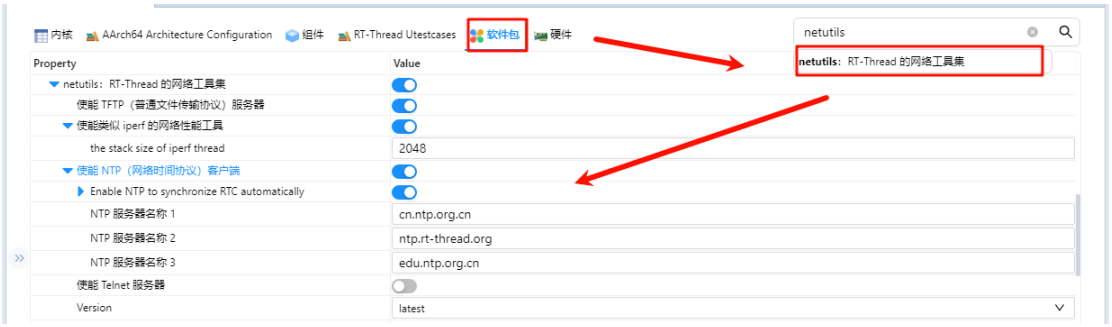


图 13-2 使能 netutils

13.4 运行

13.4.1 编译&下载

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

编译完成后，将开发板的 Jlink 接口与 PC 机连接，然后将固件下载至开发板。

13.4.2 运行效果

13.4.2.1 TFTP Server 发送测试

首先安装 netutils-v1.3.3\tools 下的 Tftpd64-4.60-setup 软件；

图 13-3 安装 tftp 工具

回到开发板串口终端，输入 tftp_server 命令开启 tftp-server 服务；

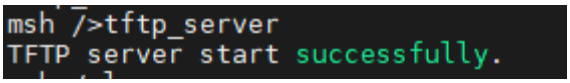


图 13-4 tftp 服务开启

然后打开安装好的 Tftpd64-4.60 软件，配置如下信息：

- Host 是开发板的 IP 地址；
- Port 是 TFTP 服务器端口号，默认： 69；
- Local File 是客户端发送文件的存放路径（包含文件名）；
- 最后点击 Put 按钮即可发送文件到设备端。

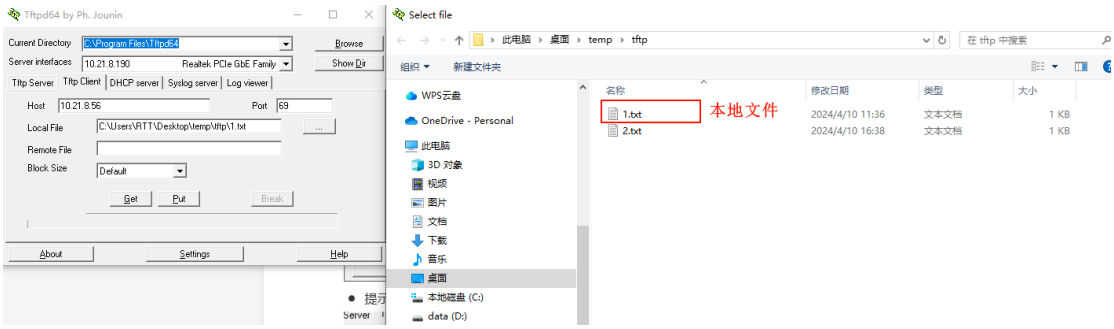


图 13-5 tftp 文件传输测试

点击 Put 后，会提示已经发送信息；

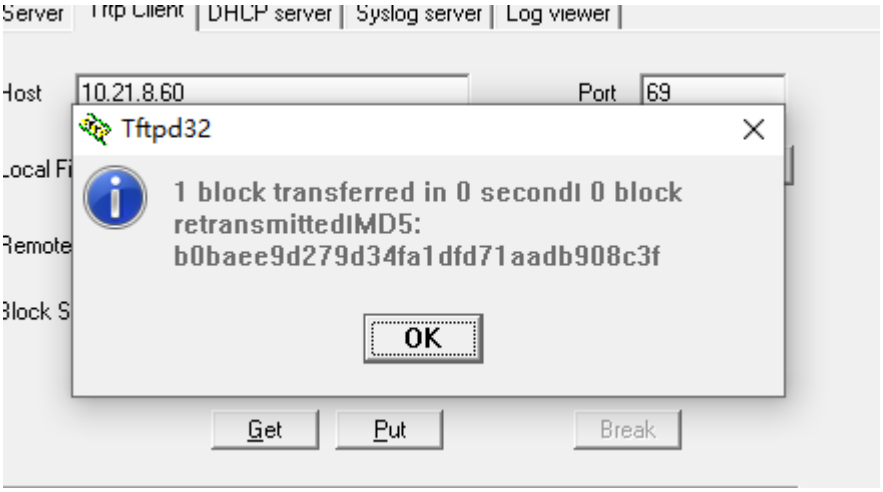


图 13-6 tftp 测试成功

返回开发板终端，输入 ls，可以看到已经接收到电脑发来的 1.txt 文件；可以输入 cat 1.txt 查看内容是否和我们发送文件的一致；

注意：由于使能的是 ramfs，因此不要传输超过 128KB 的文件！仅作为测试使用

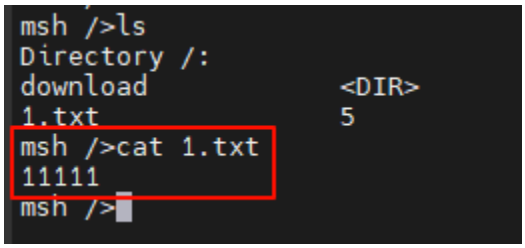


图 13-7 查看接收文件

13.4.2.2 TFTP 接收测试

回到开发板串口终端，输入：echo "rtthread" 2.txt 创建并向文件中写入自定义内容；

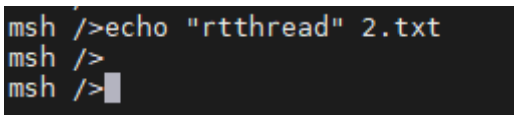


图 13-8 写入文件

可以验证下是否创建并写入成功；

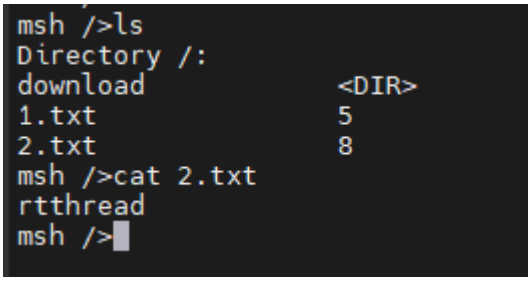


图 13-9 查看写入文件

打开安装好的 Tftpd64-4.60 软件，具体配置说明如下：

- Local File 是 客户端接收文件的存放路径（包含文件名）；
- Remote File 是服务器发送文件的路径（包括文件名），请输入我们想要接收的文件名称；
- 填写 TFTP 服务器端口号，默认： 69；
- 点击 Get 按钮；

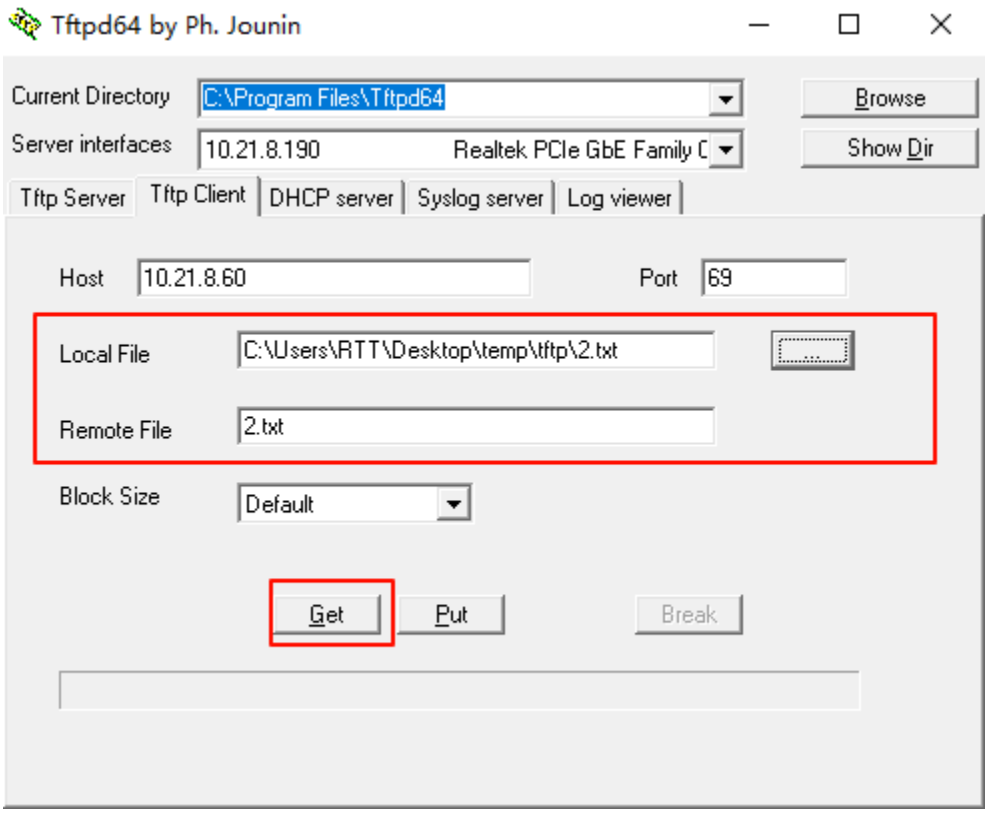


图 13-10 tftp 接收配置

可以看到 2.txt 已经接受成功，内容也是开发板文件系统中的文件内容；

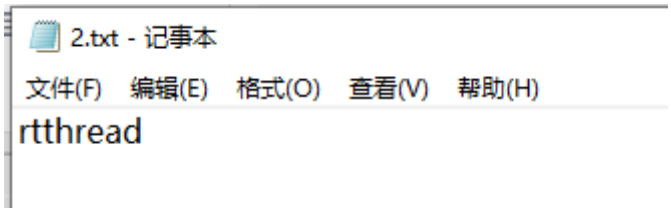


图 13-11 查看接收文件

13.4.2.3 NTP 联网校时

NTP（Network Time Protocol）是一种用于同步计算机时间的协议。它能够确保计算机时钟与全球统一的时间标准保持同步。

输入 ntp_sync 指令后，可以看到已经获取到网络时间，输入 date 指令后可以看到已经同步 RTC 的时间了。

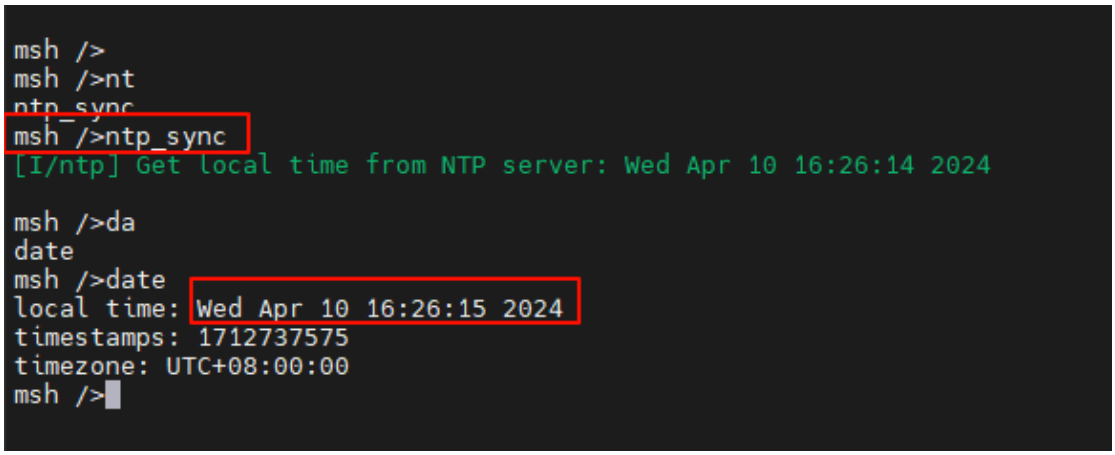


图 13-12 ntp 对时

13.5 注意事项

暂无

13.6 引用参考

第 14 章 MQTT 例程

14.1 简介

MQTT（Message Queuing Telemetry Transport）是一种轻量级的消息队列协议，用于连接多个设备到同一条网络上。MQTT 功能的原理如下：

- **基本原理**

MQTT 使用一种称为发布/订阅（Publish/Subscribe）的模式来管理数据传输。该模式允许多个设备共享同一条网络，并且可以自动检测和避免数据冲突。

- **通信过程**

MQTT 通信过程如下：

1. **设备连接**：多个设备连接到同一条 MQTT 网络上。
2. **主题订阅**：设备订阅一个或多个主题，用于接收数据。
3. **数据发布**：设备发布数据到一个或多个主题，用于发送数据。
4. **数据接收**：设备接收数据从订阅的主题，用于处理数据。

- **MQTT 模式**

MQTT 可以工作在以下几种模式：

1. **QoS 0**：MQTT 工作在 QoS 0 模式下，数据传输没有保证。
2. **QoS 1**：MQTT 工作在 QoS 1 模式下，数据传输有保证，但可能会重复。
3. **QoS 2**：MQTT 工作在 QoS 2 模式下，数据传输有保证，并且不会重复。

- **MQTT 特点**

MQTT 具有以下特点：

1. **轻量级**：MQTT 是一种轻量级的协议，适合于资源有限的设备。
2. **低延迟**：MQTT 支持低延迟的数据传输，适合于实时应用。
3. **高可靠性**：MQTT 支持高可靠性的数据传输，适合于关键应用。

总之，MQTT 是一种轻量级的消息队列协议，用于连接多个设备到同一条网络上。MQTT 具有轻量级、低延迟和高可靠性等特点，广泛应用于嵌入式系统中。

本例程基于 kawaii-mqtt 软件包，展示了通过 MQTTX 软件向服务器订阅主题和向指定主题发布消息的功能。

14.2 硬件说明

本例程需要依赖 EtherKit 板卡上的以太网模块完成网络通信，因此请确保硬件平台上的以太网模组可以正常工作。

14.3 软件说明

14.3.1 FSP 配置

本例程的源码位于 /projects/etherkit_component_mqtt，以太网驱动配置参考第 11 章：以太网例程。

14.3.2 RT-Thread Settings 配置

回到 Studio 工程，配置 RT-Thread Settings，点击选择硬件选项，找到芯片设备驱动，使能以太网；

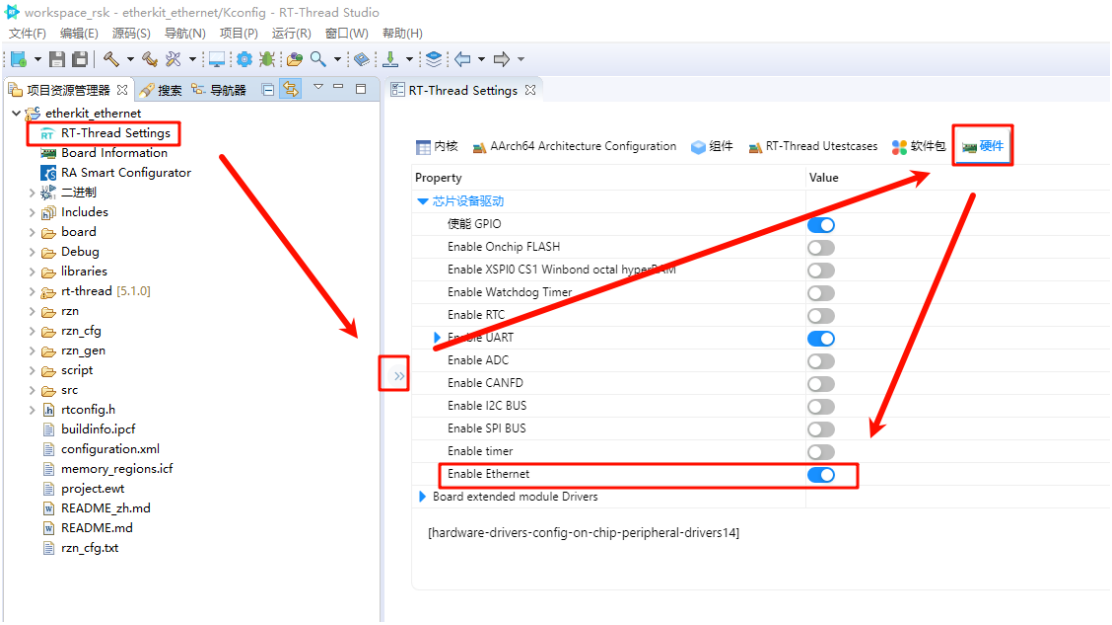


图 14-1 settings 配置

找到软件包界面，我们搜索 kawaii-mqtt 软件包，并使能 SAL 选项。

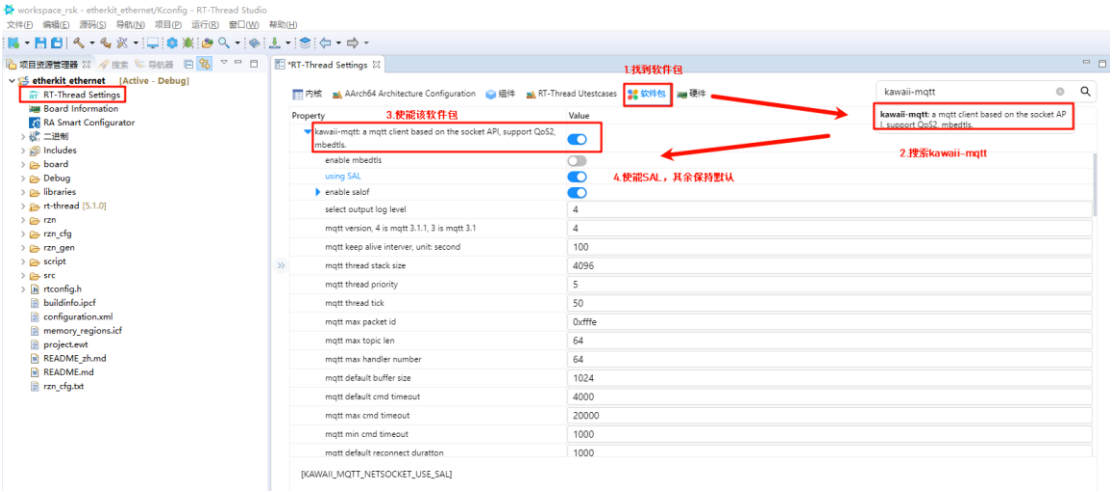


图 14-2 软件包配置

14.3.3 示例代码说明

这段代码实现了一个基于 Kawaii MQTT 客户端库的 MQTT 通信演示程序，用于连接到 MQTT 代理服务器（broker），订阅主题，并周期性发布消息。

```
static void sub_topic_handle1(void* client, message_data_t* msg)
```

```
{
    (void) client;
    KAWAII_MQTT_LOG_I("-----------------------------------------------
----------------------------------");
    KAWAII_MQTT_LOG_I("%s:%d %s()...\ntopic: %s\nmessage:%s", __FILE__,
__LINE__, __FUNCTION__, msg->topic_name, (char*)msg->message->payload);
    KAWAII_MQTT_LOG_I("-----------------------------------------------
----------------------------------");
}
static int mqtt_publish_handle1(mqtt_client_t *client)
{
    mqtt_message_t msg;
    memset(&msg, 0, sizeof(msg));
    msg.qos = QOS0;
    msg.payload = (void *)"this is a kawaii mqtt test ...";
    return mqtt_publish(client, "pub5323", &msg);
}
static char cid[64] = { 0 };
static void kawaii_mqtt_demo(void *parameter)
{
    mqtt_client_t *client = NULL;
    rt_thread_delay(6000);
    mqtt_log_init();
    client = mqtt_lease();
    rt_snprintf(cid, sizeof(cid), "rtthread-5323", rt_tick_get());
    mqtt_set_host(client, "broker.emqx.io");
    mqtt_set_port(client, "1883");
    mqtt_set_user_name(client, "RT-Thread");
    mqtt_set_password(client, "012345678");
    mqtt_set_client_id(client, cid);
    mqtt_set_clean_session(client, 1);
    KAWAII_MQTT_LOG_I("The ID of the Kawaii client is: %s ",cid);
    mqtt_connect(client);
    mqtt_subscribe(client, "sub5323", QOS0, sub_topic_handle1);
    while (1) {
        mqtt_publish_handle1(client);
        mqtt_sleep_ms(4 * 1000);
    }
}
int ka_mqtt(void)
{
    rt_thread_t tid_mqtt;
    tid_mqtt = rt_thread_create("kawaii_demo", kawaii_mqtt_demo,
RT_NULL, 2048, 17, 10);
```

```
    if (tid_mqtt == RT_NULL) {
        return -RT_ERROR;
    }
    rt_thread_startup(tid_mqtt);
    return RT_EOK;
}
MSH_CMD_EXPORT(ka_mqtt, Kawaii MQTT client test program);
```

14.4 运行

14.4.1 编译&下载

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

编译完成后，将开发板的 Jlink 接口与 PC 机连接，然后将固件下载至开发板。

14.4.2 MQTTX 配置

安装并运行 MQTTX，来到主界面，我们点击 New Connection 新增一个新连接；

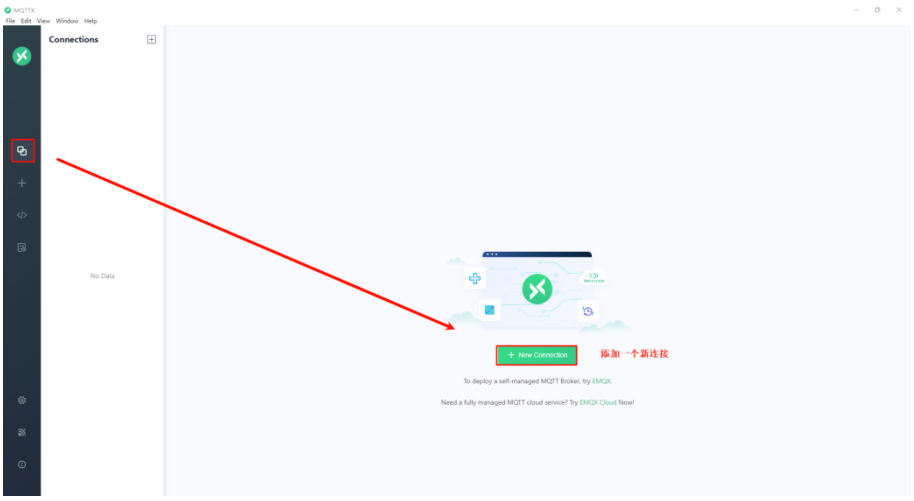


图 15-3 新增 MQTT 连接

配置 MQTT 客户端信息，注意 Client ID 不要和开发板端一致，点击后面的重置按钮随意生成一个 id 即可，其他配置参考下述说明，配置结束后，点击右上角的 Connect；

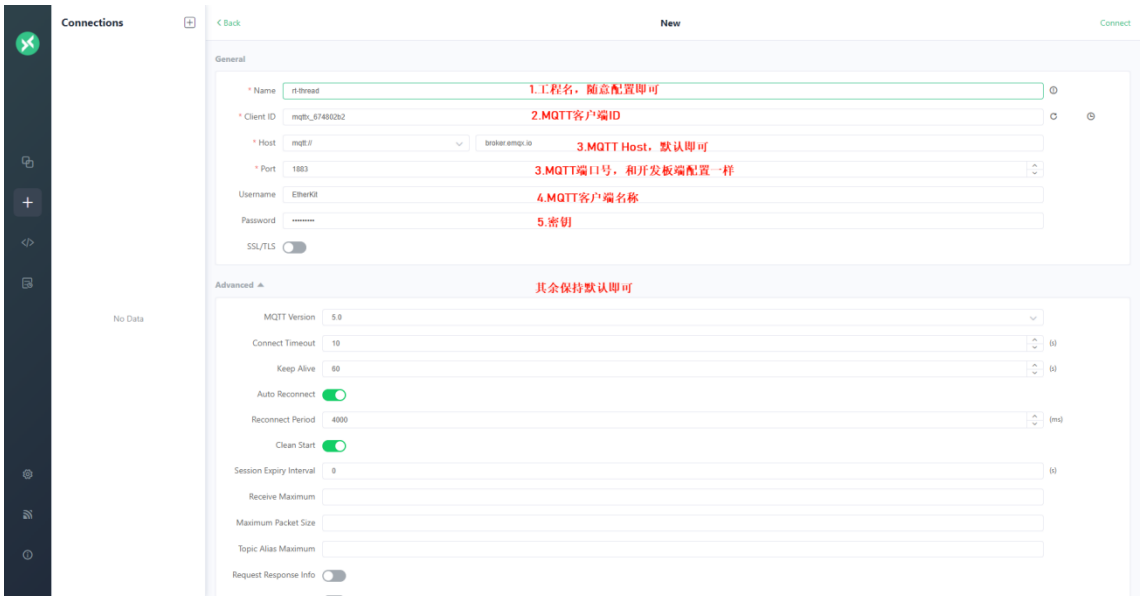


图 14-4 MQTT 客户端配置

点击 + New Subscription，修改 Topic name 为 sub5323 并确认；

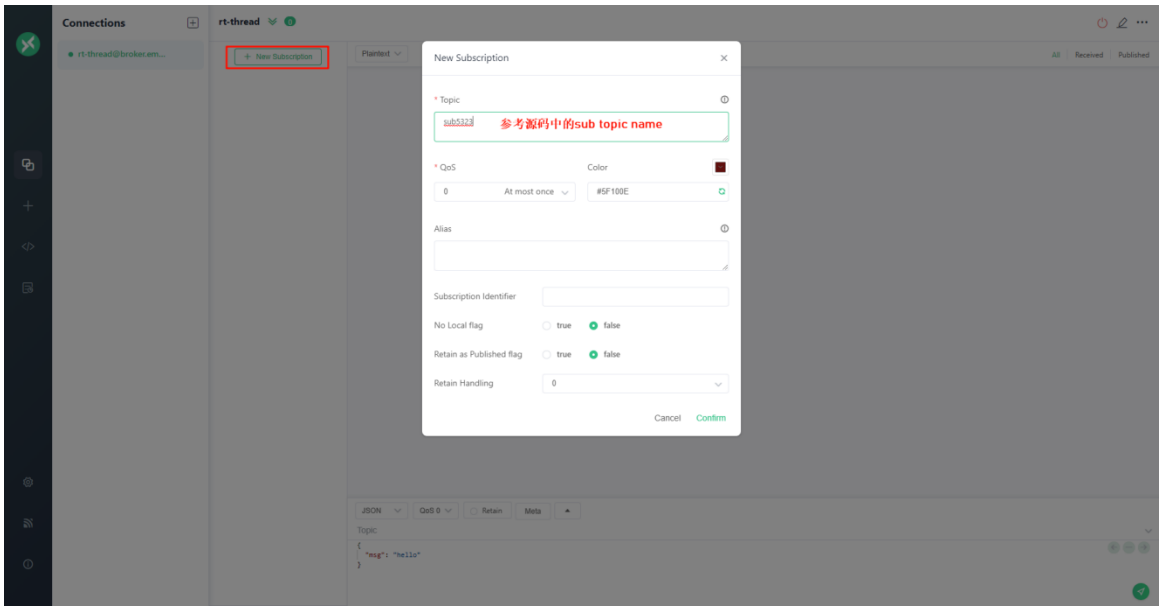


图 14-5 new Subscription

在下方功能框中编写订阅主题名称为 sub5323，订阅信息按自己需求配置：

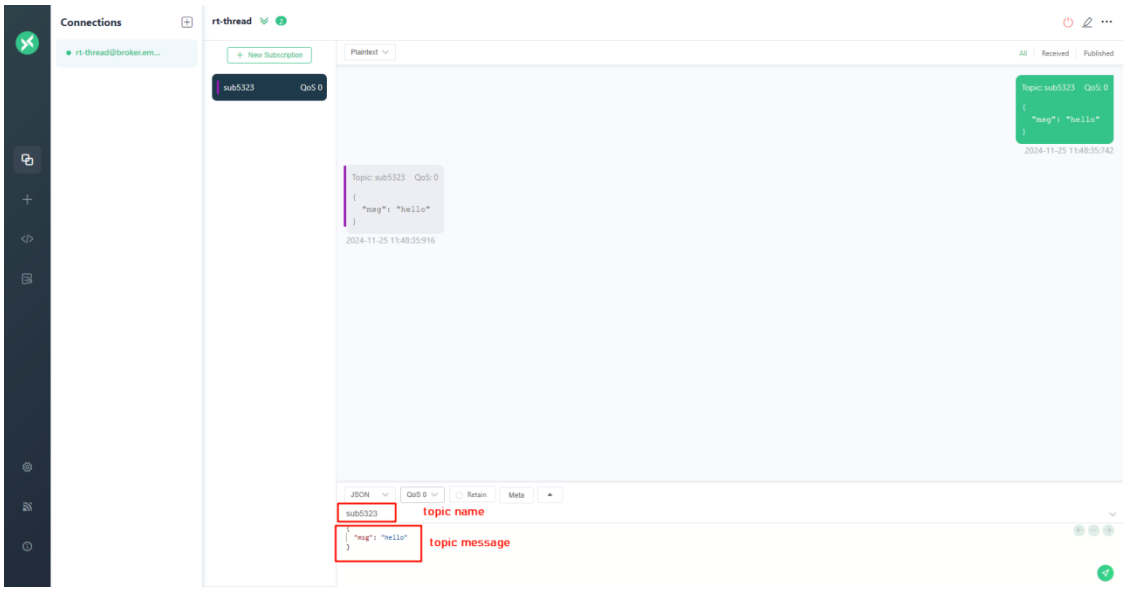


图 14-6 发布主题

14.4.3 运行效果

打开串口工具，运行 ka_mqtt 命令后查看：

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Nov 25 2024 11:30:10
 2006 - 2024 Copyright by RT-Thread team
lwIP-2.0.3 initialized!
[W/DBG] R_ETHER_Write failed!, res = 4001
[I/sal.skt] Socket Abstraction Layer initialize success.

Hello RT-Thread!
================================================
This example project is an mqtt component routine!
================================================
msh />[W/DBG] R_ETHER_Write failed!, res = 4001
[I/DBG] link up

msh />
msh />
msh />mq
msh />ka
ka_mqtt
msh />ka_mqtt
msh />
[I] >> The ID of the Kawaii client is: rtthread-5323
[I] >> ../packages/kawaii-mqtt-v1.1.0/mqttclient/mqttclient.c:976 mqtt_connect_with_results()... mqtt connect success...
[I] >> --------------------------------------------------------------------------------
[I] >> ../src/hal_entry.c:46 sub_topic_handle1()...
topic: sub5323
message:{
  "msg": "hello"
}
[I] >> --------------------------------------------------------------------------------
```

图 14-7 MQTT 运行

14.5 其他说明

MQTTX 下载链接：https://packages.emqx.net/MQTTX/v1.9.6/MQTTX-Setup-

1.9.6-x64.exe

14.6 引用参考

- 软件包：kawaii-mqtt

第 15 章 Modbus-UART 例程

15.1 简介

Modbus 是一种开放的通信协议，用于在控制设备之间传输数据，支持多种物理层，如 UART、TCP/IP 和 RS-485/232。Modbus UART 是一种通过串口通信实现的 Modbus 协议版本，广泛应用于工业自动化和控制系统中。

- **特点**

  1. **简单易用**：Modbus UART 是一种简单易用的协议，易于实现和维护。
  2. **低成本**：Modbus UART 是一种低成本的协议，不需要专用的硬件。
  3. **高可靠性**：Modbus UART 是一种高可靠性的协议，支持错误检测和纠正。

- **工作原理**

  1. **主从模式**：Modbus UART 工作在主从模式下，主设备发送命令，从设备响应。
  2. **UART 通信**：Modbus UART 使用 UART 作为物理层，发送和接收数据。
  3. **数据格式**：Modbus UART 使用一种特定的数据格式，包括地址、功能码、数据等。

- **优点**

  1. **易于实现**：Modbus UART 是一种易于实现的协议，不需要专用的硬件。
  2. **低成本**：Modbus UART 是一种低成本的协议，不需要专用的硬件。
  3. **高可靠性**：Modbus UART 是一种高可靠性的协议，支持错误检测和纠正。

- **缺点**

1. **速度慢**：Modbus UART 是一种速度慢的协议，数据传输速度有限。
2. **距离有限**：Modbus UART 是一种距离有限的协议，数据传输距离有限。
3. **安全性低**：Modbus UART 是一种安全性低的协议，数据传输不安全。

本例程基于 agile_modbus 软件包，展示了通过串口方式实现 modbus 协议通信的示例。

15.2 硬件说明

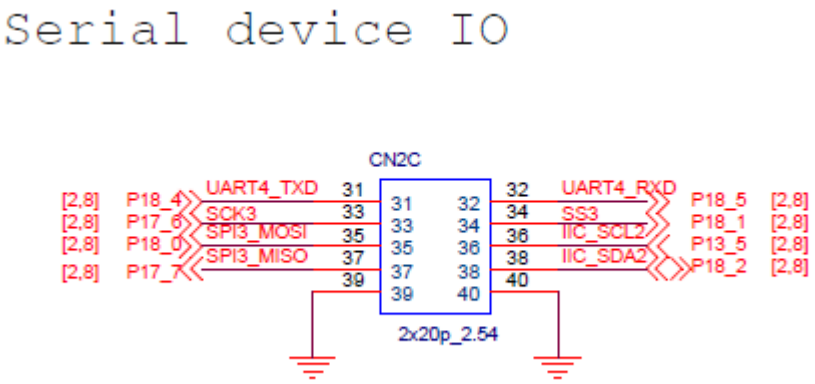


图 15-1 UART3 串口原理图

如上图所示，我们本次要使用到的外设为 SCI，其中复用 SCI3 为串口模式，因对应的 TX 引脚为 P18_0，RX 引脚为 P17_7。

15.3 软件说明

15.3.1 FSP 配置

打开工程下的 configuration.xml 文件，我们添加一个新的 stack：

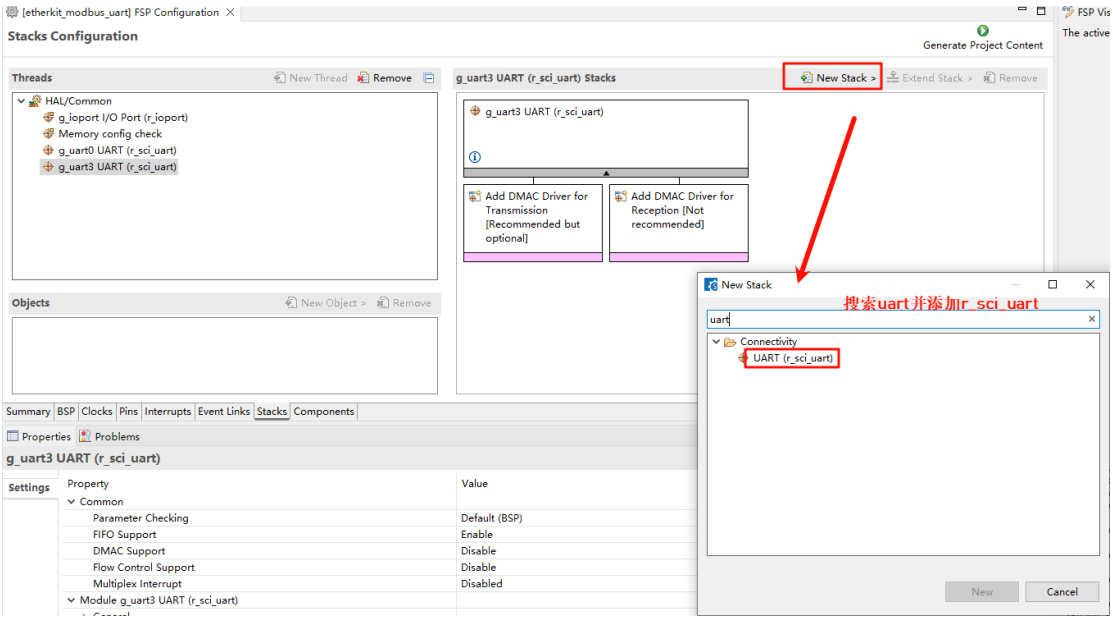


图 15-2 添加 sci_uart stack

打开 r_sci_uart 配置，使能 FIFO 支持，同时设置通道数为 3；

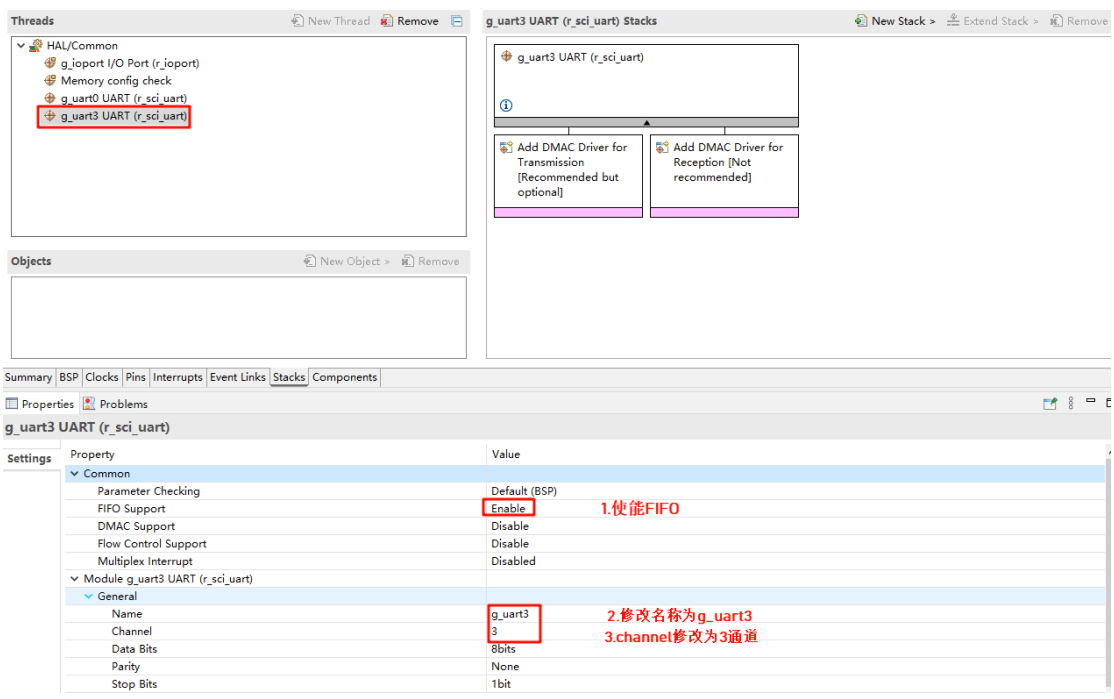


图 15-3 sci-uart 配置

点击选择 Pins，设置 SCI3，将 SCI mode 修改为 Asynchronous mode，同时可以对应看到相关引脚被使能；

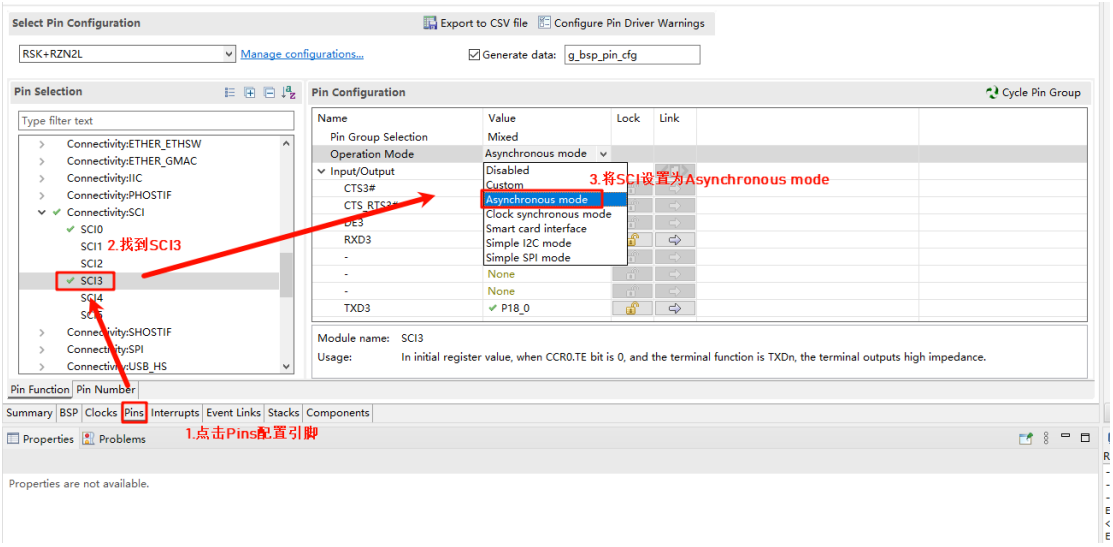


图 15-4 sci-uart 引脚模式配置

回到 stack 界面，展开并设置中断回调函数为 user_uart3_callback，同时在下方可以知道对应的串口引脚信息；

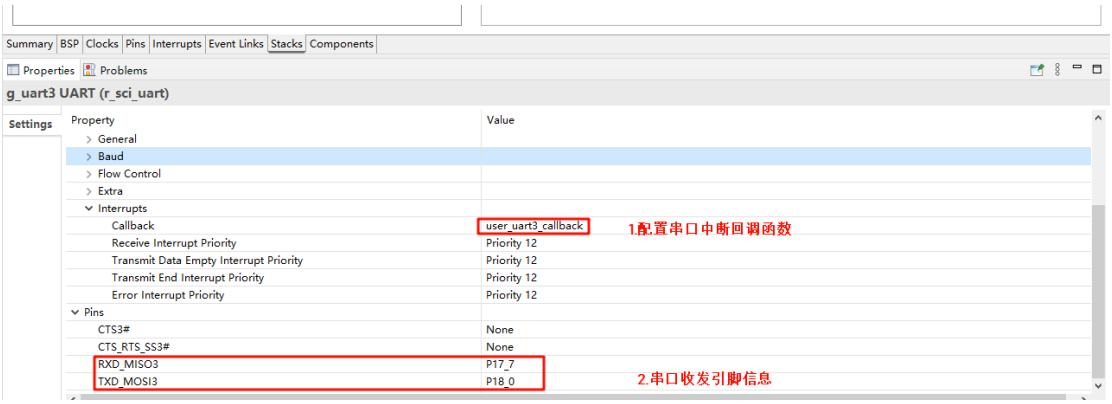


图 15-5 串口引脚信息

15.3.2 RT-Thread Settings 配置

回到 studio，点击 RT-Thread Settings，先配置串口，使能 UART3；

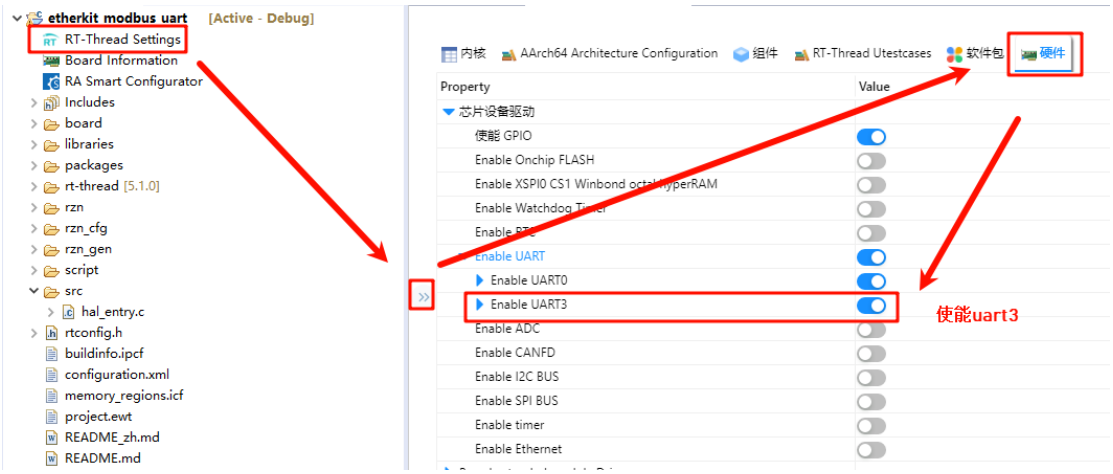


图 15-6 settings 配置串口

找到软件包界面，在搜索框搜索 modbus，并选择 agile_modbus 软件包后使能；

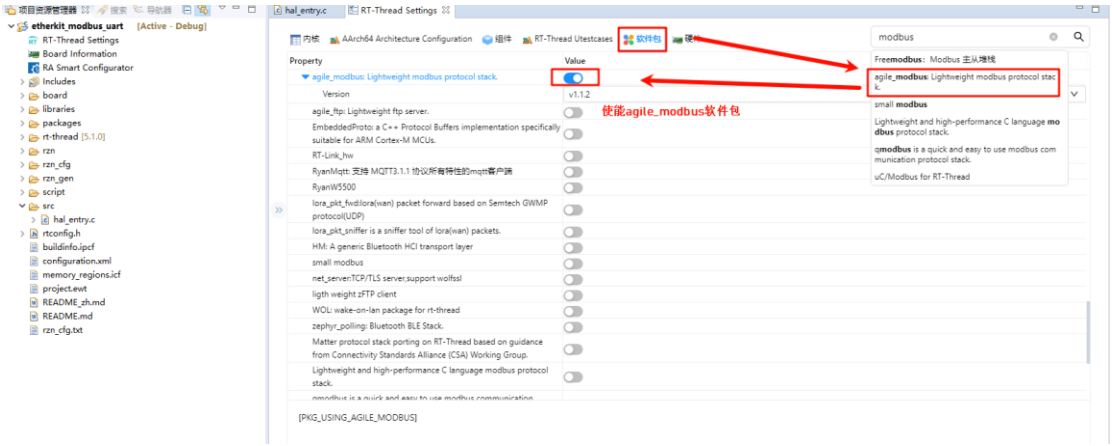


图 15-7 软件包使能

15.4 运行

15.4.1 编译&下载

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

编译完成后，将开发板的 Jlink 接口与 PC 机连接，然后将固件下载至开发板。

15.4.2 运行效果

首先我们需要使用一个 USB 转 TTL 模块，将其收发引脚与开发板串口 3 的收发引脚反接（RX-TX(P18_0)，TX-RX(P17_7)），如下图所示：

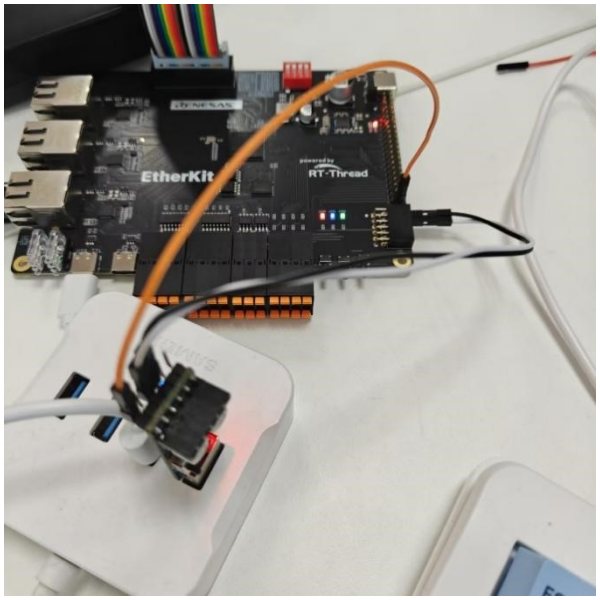


图 15-8 usb-ttl 接线示意

接着我们打开 modbus slaver 软件，点击连接；

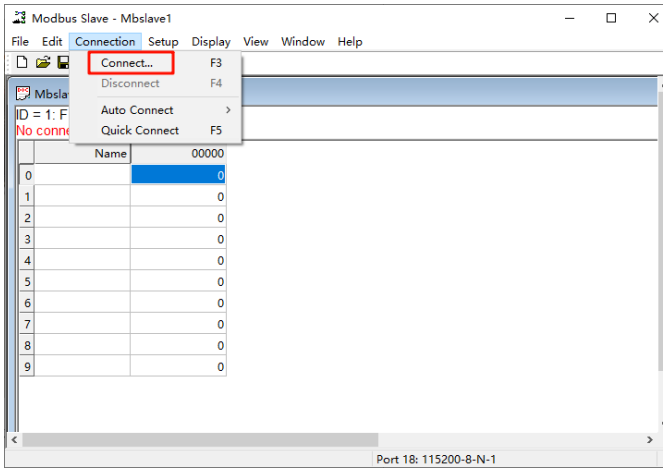


图 15-9 modbus slaver 连接

配置 modbus slaver 信息，首先选择连接为串口模式，串口设备为连接到开发板的 USB 转 TTL 模块，并设置 None Parity；

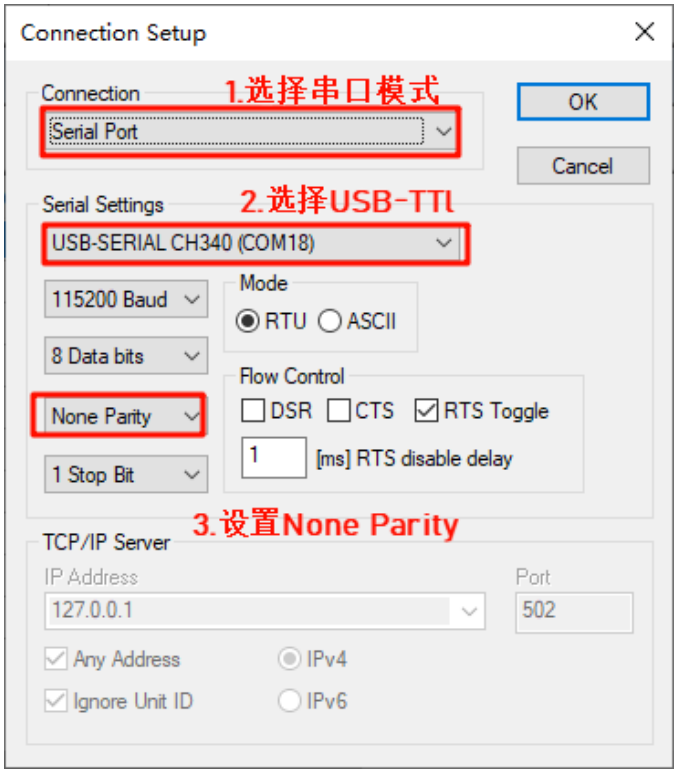


图 15-10 modbus slaver 设置

接着我们回到串口工具，输入命令 modbus_master_uart_sample 开启 modbus 主站示例；

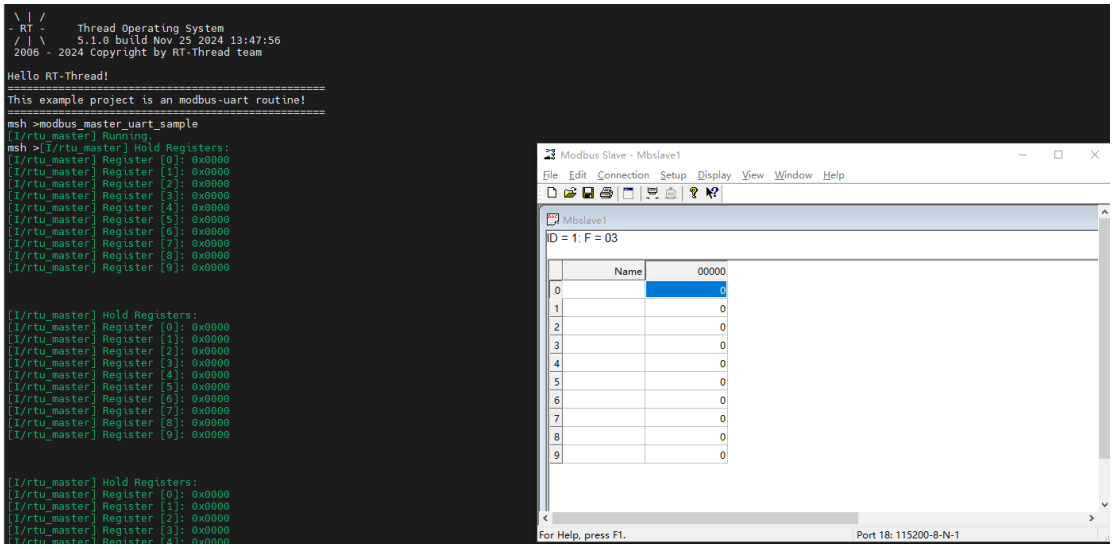


图 15-11 开启 modbus 主站示例

开发板的串口 3 作为主机，电脑作为从机，向站号写线圈，串口终端会同步显示寄存器的修改；

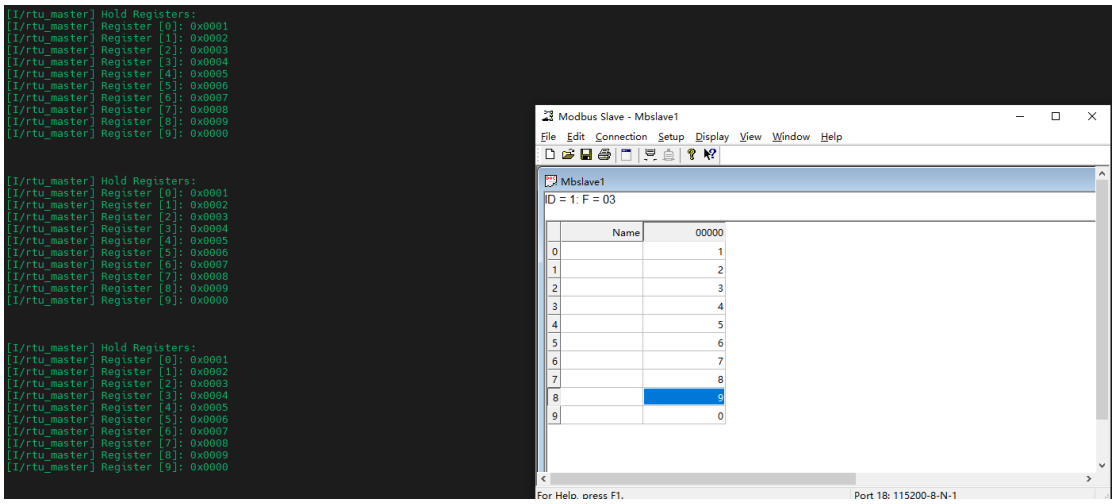


图 15-12 modbus 从站寄存器修改

15.5 注意事项

暂无

15.6 引用参考

- 软件包：agile_modbus

第 16 章 Modbus-TCP/IP 例程

16.1 简介

Modbus TCP/IP 是一种通过串口通信实现的 Modbus 协议版本，广泛应用于工业自动化和控制系统中。Modbus 是一种开放的通信协议，用于在控制设备之间传输数据，支持多种物理层，如 UART、TCP/IP 和 RS-485/232。

本例程基于 agile_modbus 软件包，展示了通过 TCP/IP 方式实现 modbus 协议通信的示例。

16.2 硬件说明

本例程使用的硬件为以太网接口，请确保以太网功能正常

16.3 软件说明

16.3.1 FSP 配置

注：该小节配置同 11.3.1 FSP 配置。

16.3.2 RT-Thread Settings 配置

回到 studio，点击 RT-Thread Settings，先配置以太网，使能 Ethernet；

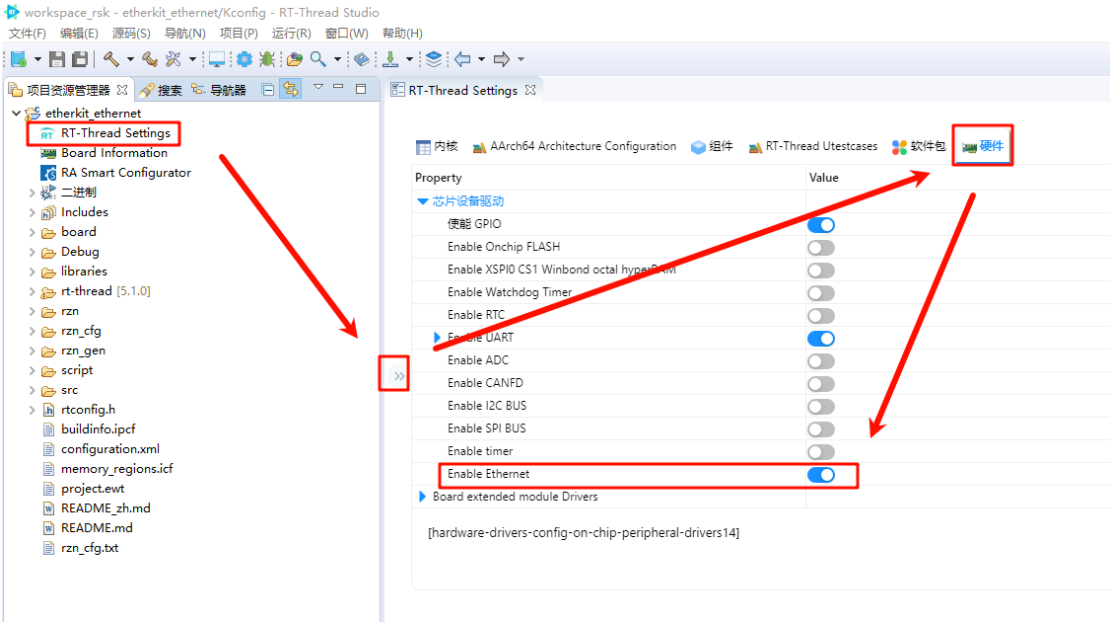


图 16-1 settings 配置以太网

找到软件包界面，在搜索框搜索 modbus，并选择 agile_modbus 软件包后使能；

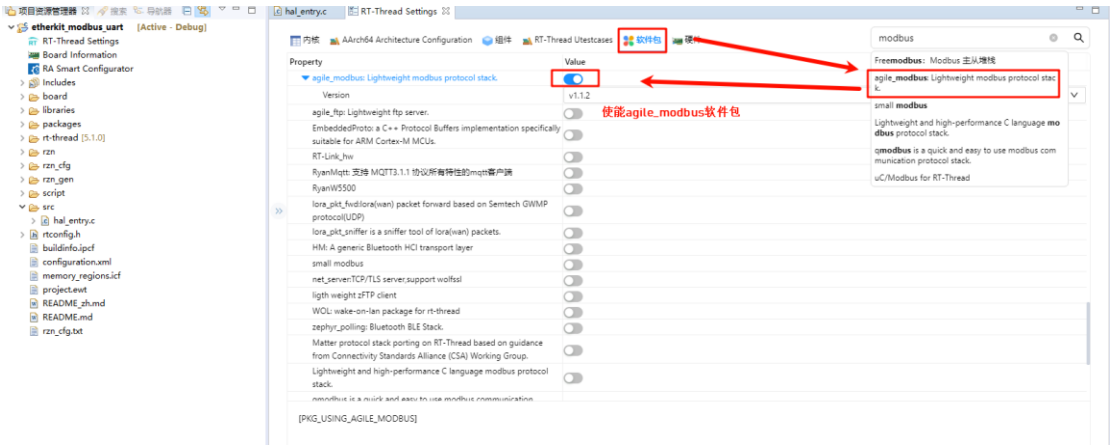


图 16-2 软件包使能

16.4 运行

16.4.1 编译&下载

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

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

编译完成后，将开发板的 Jlink 接口与 PC 机连接，然后将固件下载至开发板。

16.4.2 运行效果

首先找一根网线连接开发板网口及交换机（自己电脑有多余网口的也可以使用共享适配器操作），接着在串口工具输入命令：modbus_tcp_test，开启 modbus-tcp 示例；

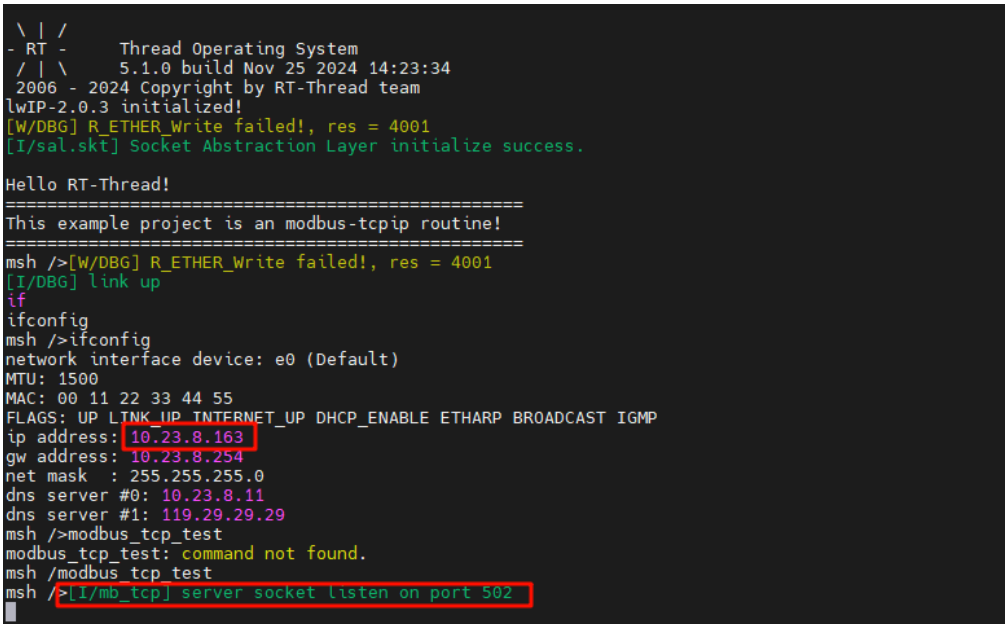


图 16-1 启动 mdobus tcp 测试

打开 Modbus Poll 软件，连接并设置其模式为 Modbus TCP/IP，IP 为开发板 IP 信息，同时端口号为 502；

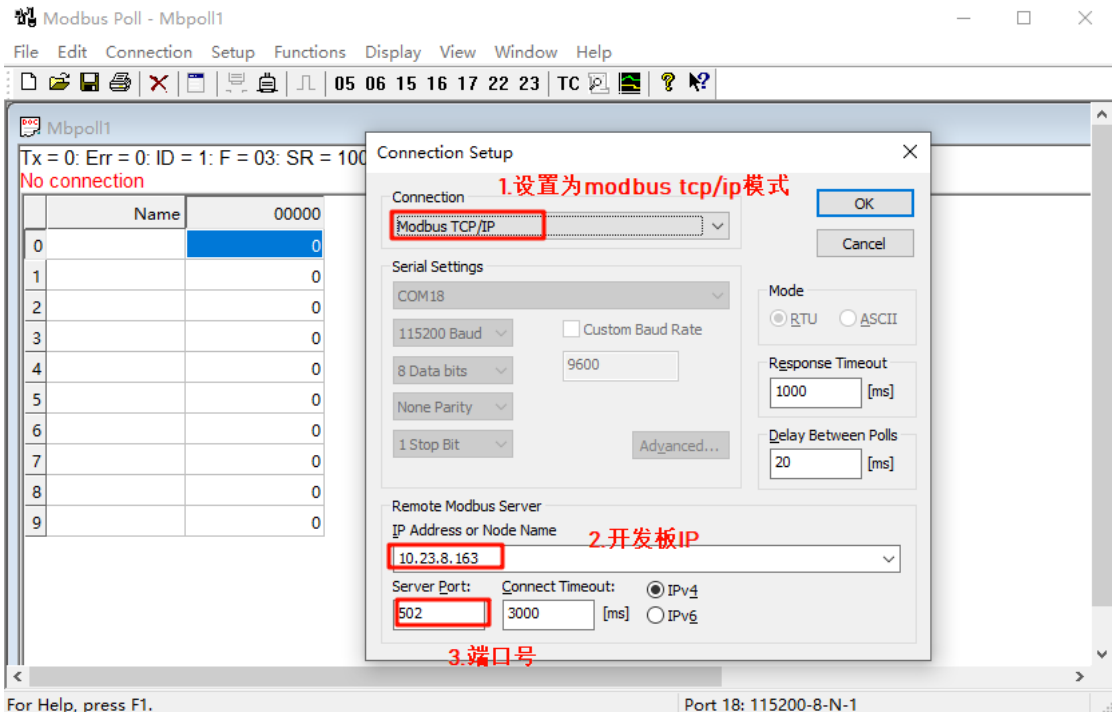


图 16-2 mdobus tcp/ip 设置

连接成功后在开发板终端可以看到 modbus 客户端已连接；

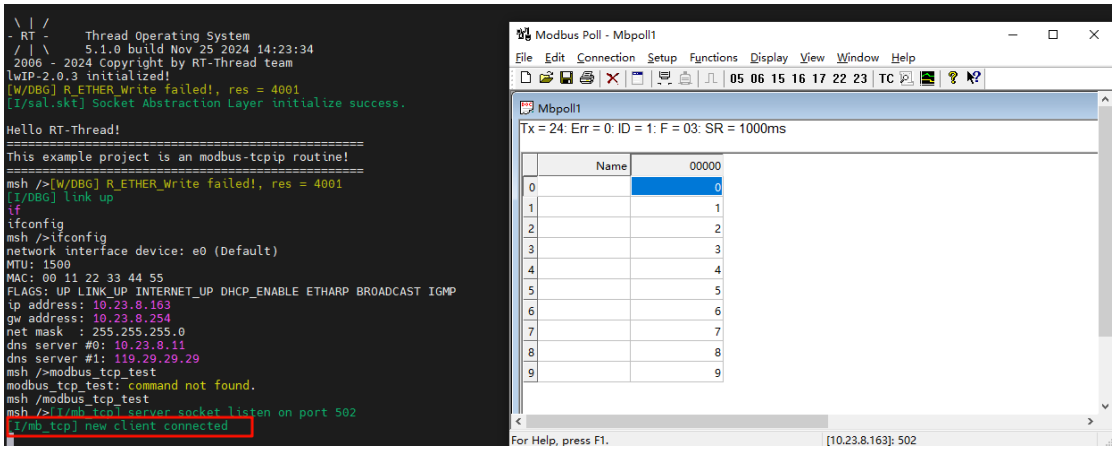


图 16-3 modbus tcp client 连接

回到 Modbus Poll 软件可以看到读写线圈功能都是正常的；

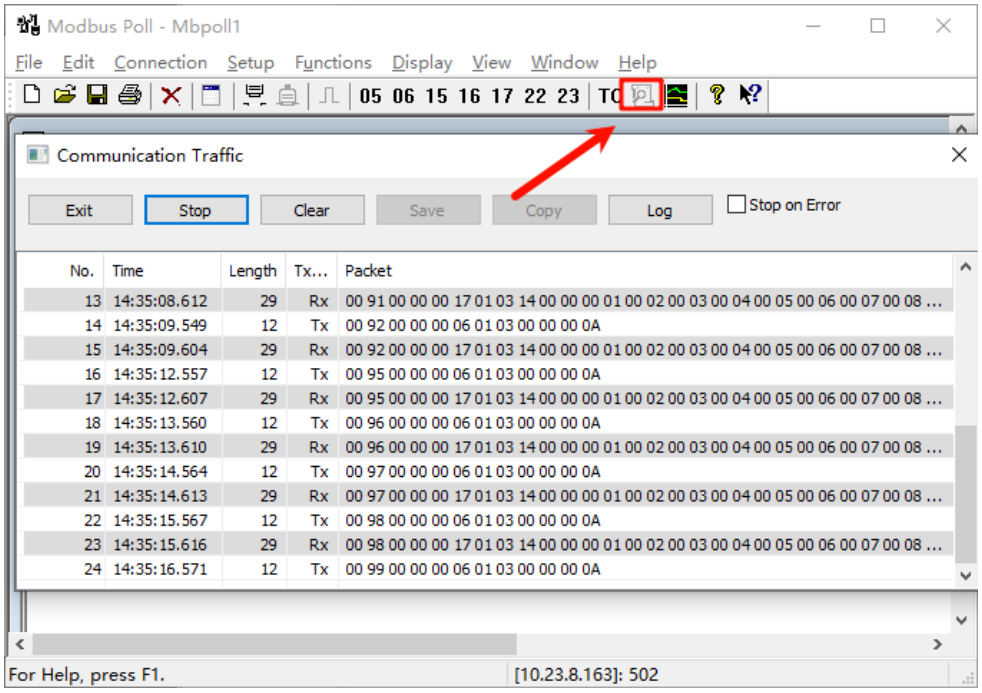


图 16-4 modbus 线圈读取

16.5 注意事项

暂无

16.6 引用参考

- 软件包：kawaii-mqtt

第 17 章 USB-PMSC 例程

17.1 简介

本例程展示了通过 USB 方式实现 U 盘的示例。USB-PMSC（USB Peripheral Mass Storage Class）是一种通用串行总线（USB）设备类，用于实现基于 USB 的存储设备功能。

17.2 硬件说明

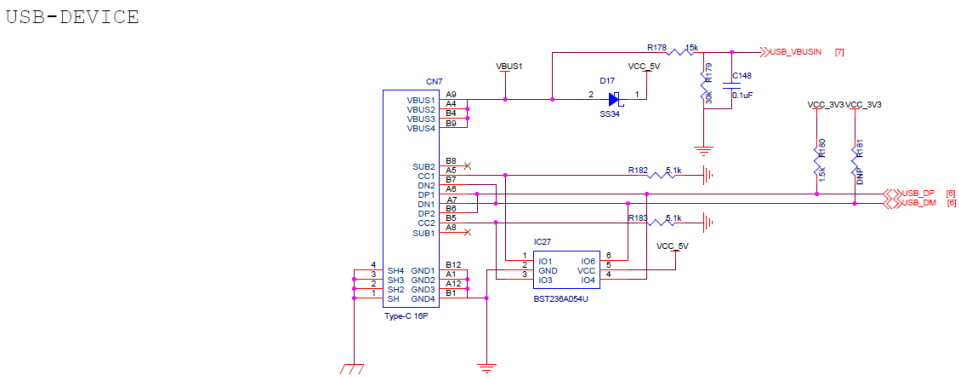


图 17-1 USB-Device 原理图

EtherKit 提供一个 USB-Device 外设，位于开发板的位置如下：

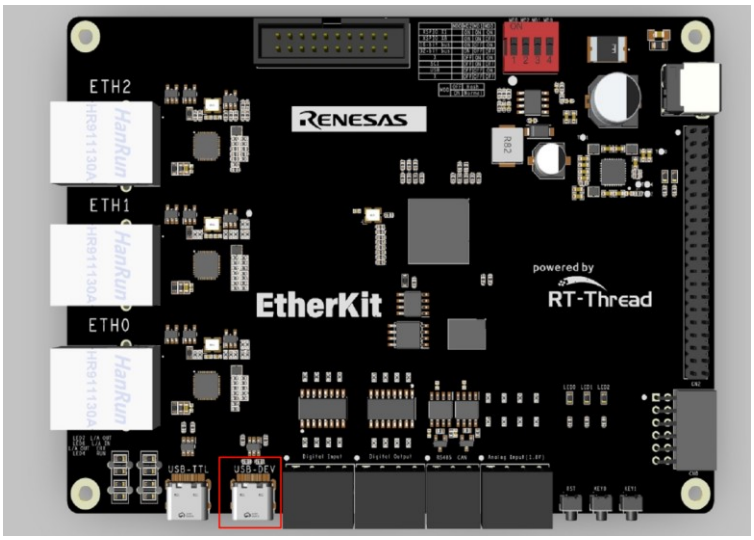


图 17-2 USB-Device 位置

17.3 软件说明

17.3.1 FSP 配置

使用 fsp 打开工程下的 configuration.xml 文件，并添加 usb_pmcs stack；

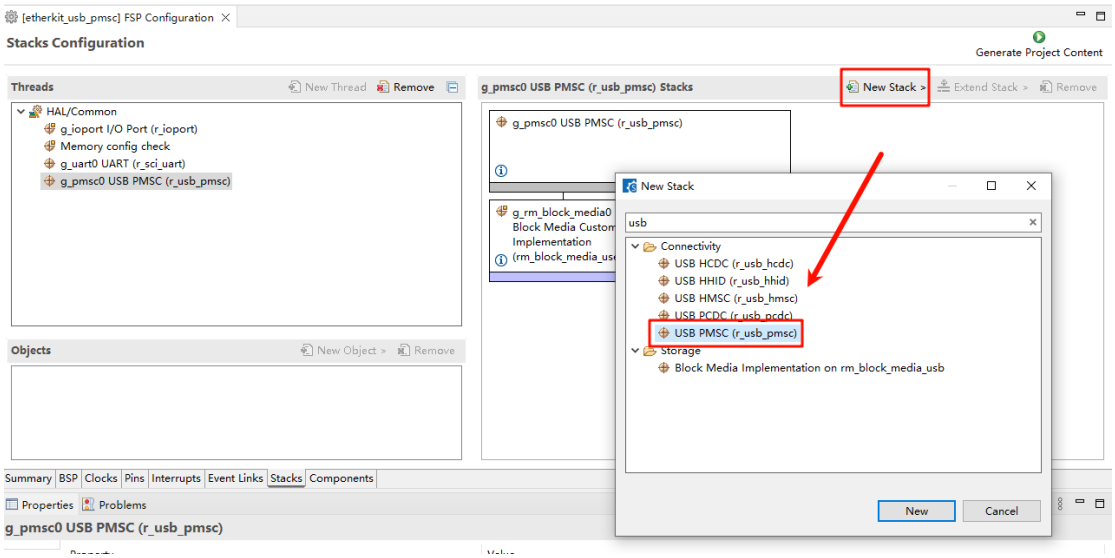


图 17-3 usb_pmsc stack 添加

添加 g_rm_block_media0；

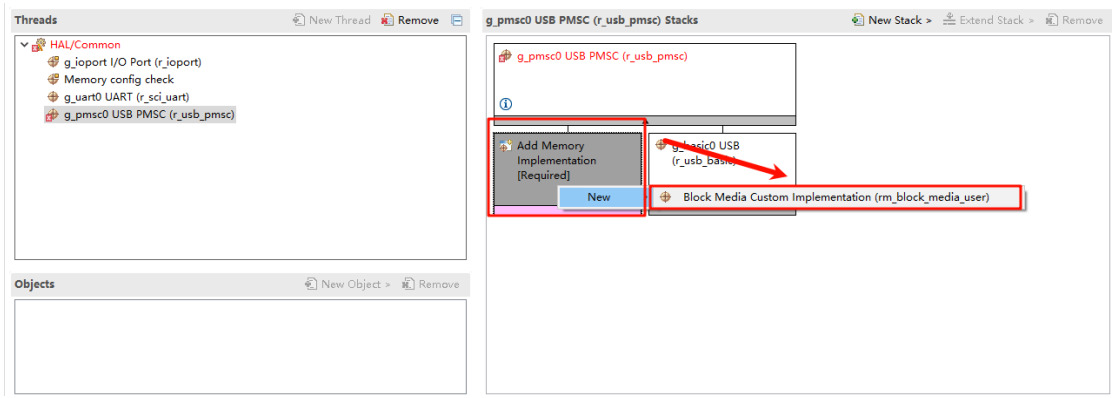


图 17-4 block media0 添加

选择 g_basic0_usb，设置其中断回调函数为 usb_apl_callback；

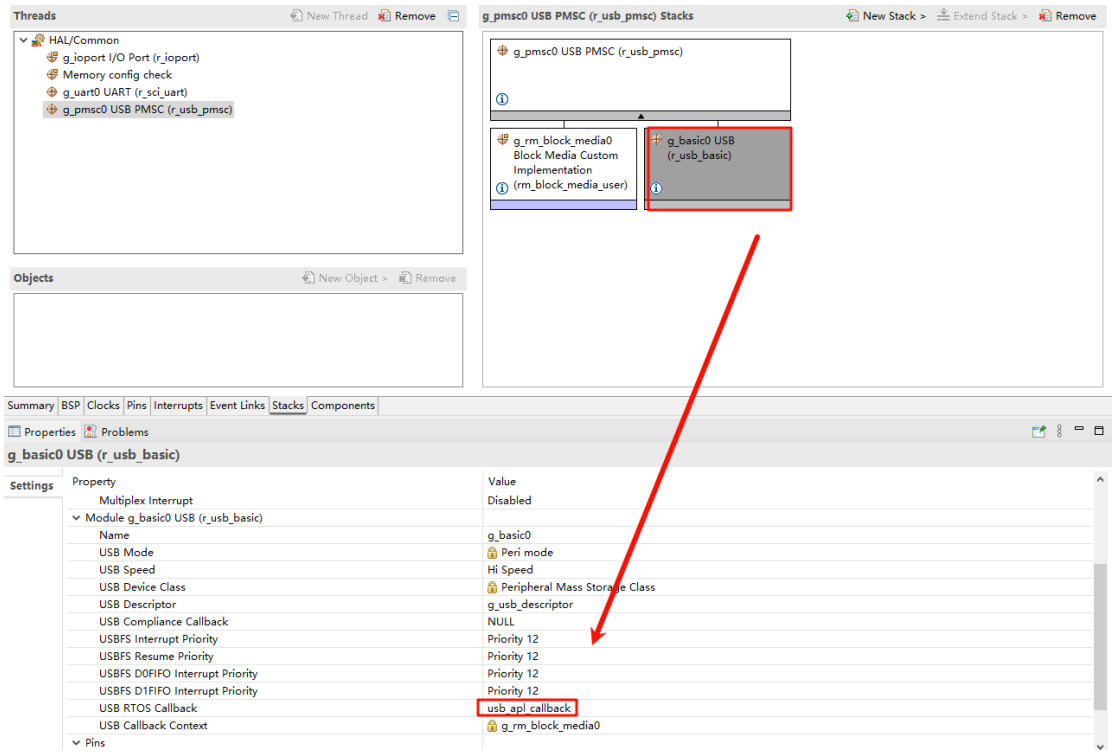


图 17-5 添加 USB callback

下面配置 USB 引脚，找到 USB_HS 并使能；

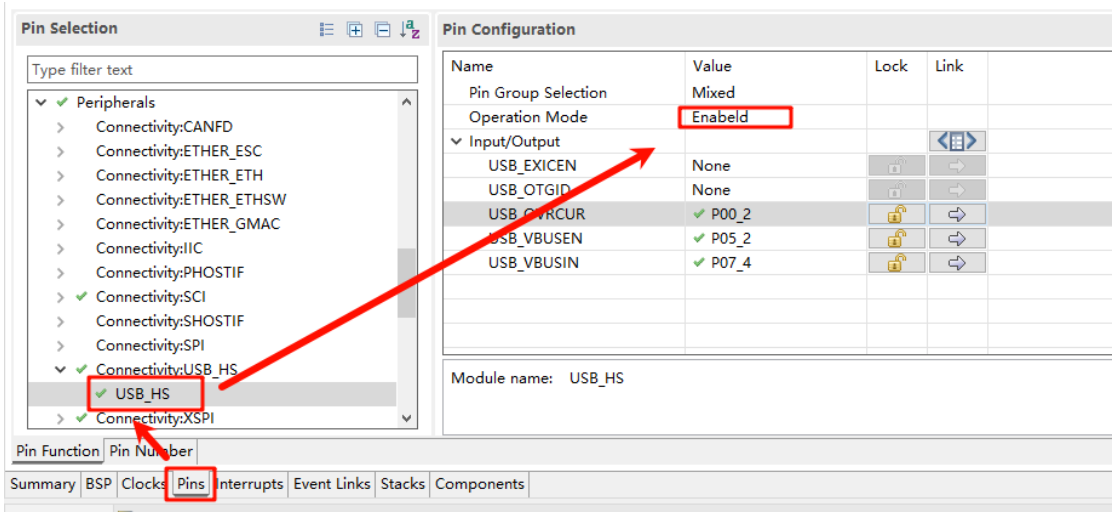


图 17-6 USB 引脚配置

17.3.2 构建配置

进入工程找到指定路径下的文件：.\rzn\SConscript，替换该文件为如下内

容：

```
Import('RTT_ROOT')
Import('rtconfig')
from building import *
from gcc import *

cwd = GetCurrentDir()
src = []
group = []
CPPPATH = []

if rtconfig.PLATFORM in ['iccarm']:
    Return('group')
elif rtconfig.PLATFORM in GetGCCLikePLATFORM():
    if GetOption('target') != 'mdk5':
        src += Glob('./fsp/src/bsp/mcu/all/*.c')
        src += Glob('./fsp/src/bsp/mcu/all/cr/*.c')
        src += Glob('./fsp/src/bsp/mcu/r*/*.c')
        src += Glob('./fsp/src/bsp/cmsis/Device/RENESAS/Source/*.c')
        src += Glob('./fsp/src/bsp/cmsis/Device/RENESAS/Source/cr/*.c')
        src += Glob('./fsp/src/r_*/*.c')
        src += Glob('./fsp/src/r_usb_basic/src/driver/*.c')
        src += Glob('./fsp/src/r_usb_basic/src/hw/*.c')
        src += Glob('./fsp/src/r_usb_pmsc/src/*.c')
        CPPPATH = [ cwd + '/arm/CMSIS_5/CMSIS/Core_R/Include',
                        cwd + '/fsp/inc',
                        cwd + '/fsp/src/inc',
                        cwd + '/fsp/inc/api',
                        cwd + '/fsp/inc/instances',
                        cwd + '/fsp/src/r_usb_basic/src/driver/inc',
                        cwd + '/fsp/src/r_usb_basic/src/hw/inc',
                        cwd + '/fsp/src/r_usb_pmsc/src/inc',]

group = DefineGroup('rzn', src, depend = [''], CPPPATH = CPPPATH)
Return('group')
```

使用 studio 开发的话需要右键工程点击 **同步 scons 配置至项目**；如果是使用 IAR 开发请在当前工程下右键打开 env，执行：scons –target=iar 重新生成配置。

17.3.3 RT-Thread Settings 配置

USB 示例目前使用的是 freertos 接口驱动，因此我们还需要使能 Freertos 兼容层软件包；

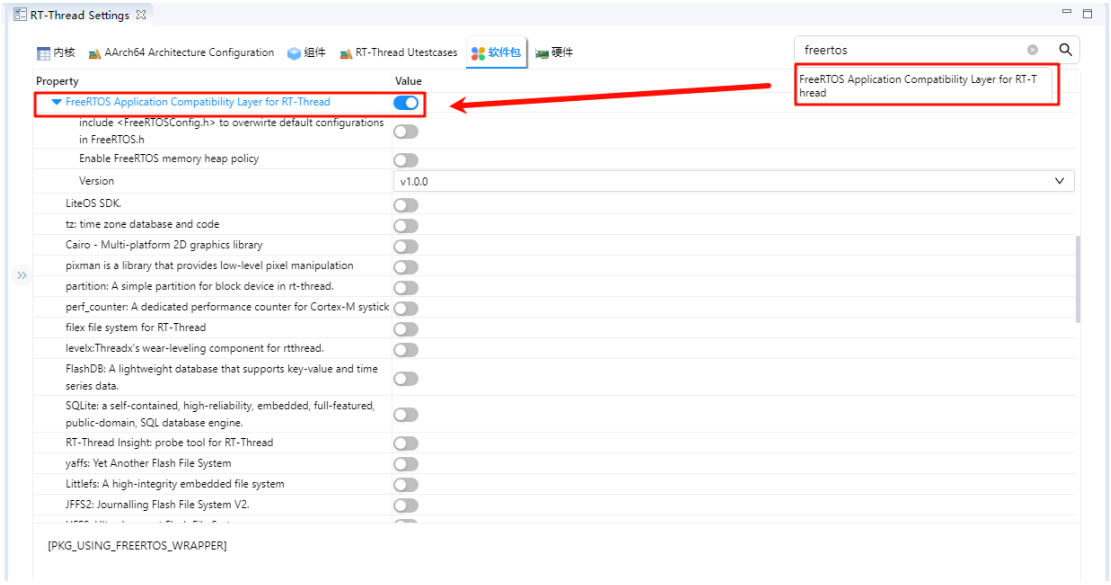


图 17-6 使能 freertos 兼容层

17.4 运行

17.4.1 编译&下载

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

编译完成后，将开发板的 Jlink 接口与 PC 机连接，然后将固件下载至开发板。

17.4.2 运行效果

生成 FSP 配置之后进行编译下载，将代码烧录到开发板即可自动启动该示例，同时在文件管理器中可以发现多出了一个 U 盘设备；

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Nov 25 2024 15:26:40
 2006 - 2024 Copyright by RT-Thread team

Hello RT-Thread!
================================================
This example project is an usb pmsc routine!
================================================
msh >ps
thread           pri  status      sp     stack size max used left tick   error  tcb addr
---------------- ---  ------- ---------- ----------  ------  ---------- ------- ----------
PMSC_TSK           1  suspend 0x000000c0 0x00000800    17%   0x00000001 EINTRPT 0x10048230
PCD_TSK            0  suspend 0x000000c0 0x00001800    04%   0x00000001 EINTRPT 0x10046820
tshell            20  running 0x000000b0 0x00001000    13%   0x00000003 OK      0x100454f8
usb_td            20  suspend 0x000000e8 0x00000800    19%   0x00000014 EINTRPT 0x10044a00
sys workq         23  suspend 0x00000078 0x00000800    05%   0x0000000a OK      0x10044138
tidle0            31  ready   0x00000048 0x00000400    10%   0x00000015 OK      0x10041d24
main              10  suspend 0x000000b0 0x00000800    12%   0x0000000d EINTRPT 0x10043810
msh >
```

图 17-7 USB-PMSC 终端

图 17-8 USB 模拟 U 盘

17.5 注意事项

暂无

17.6 引用参考

第 18 章 USB-PCDC 例程

18.1 简介

本例程展示了通过 USB 方式实现模拟串口的示例。USB PCDC 是 USB 通信设备类（Communication Device Class, CDC）的一种子类，通常用于实现虚拟串口通信功能。在嵌入式设备开发中，USB PCDC 常被用于通过 USB 接口将设备模拟为串行通信端口（如 COM 端口），以便与主机进行数据交互。

18.2 硬件说明

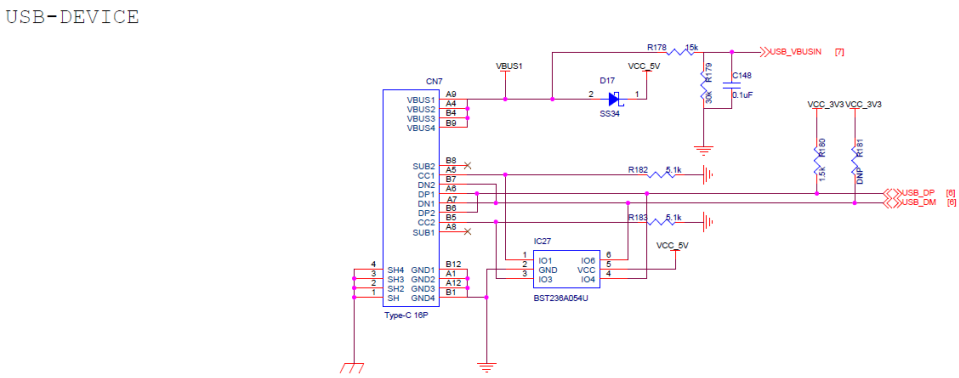


图 18-1 USB-Device 原理图

EtherKit 提供一个 USB-Device 外设，位于开发板的位置如下：

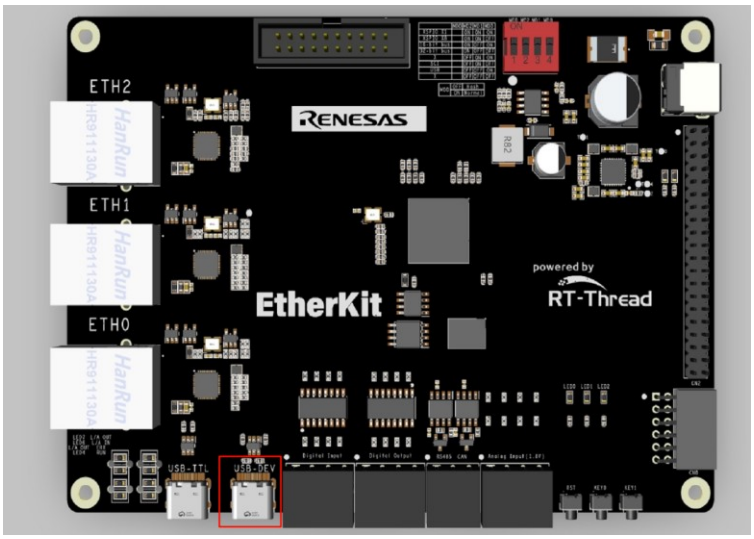


图 18-2 USB-Device 位置

18.3 软件说明

18.3.1 FSP 配置

使用 fsp 打开工程下的 configuration.xml 文件，并添加 usb_pcdc stack；

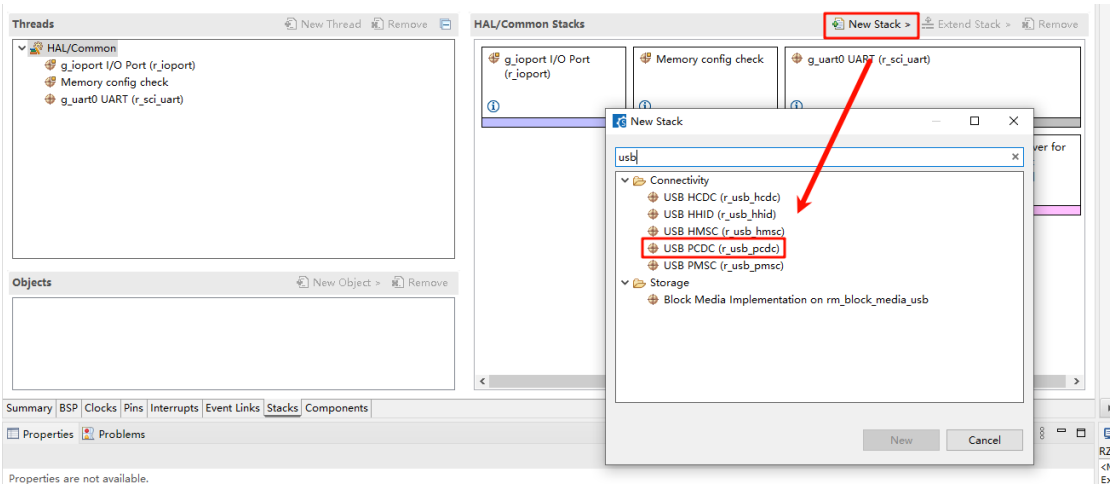


图 18-3 usb_pcdc stack 添加

选择 g_basic0_usb，设置其中断回调函数为 usb_apl_callback；

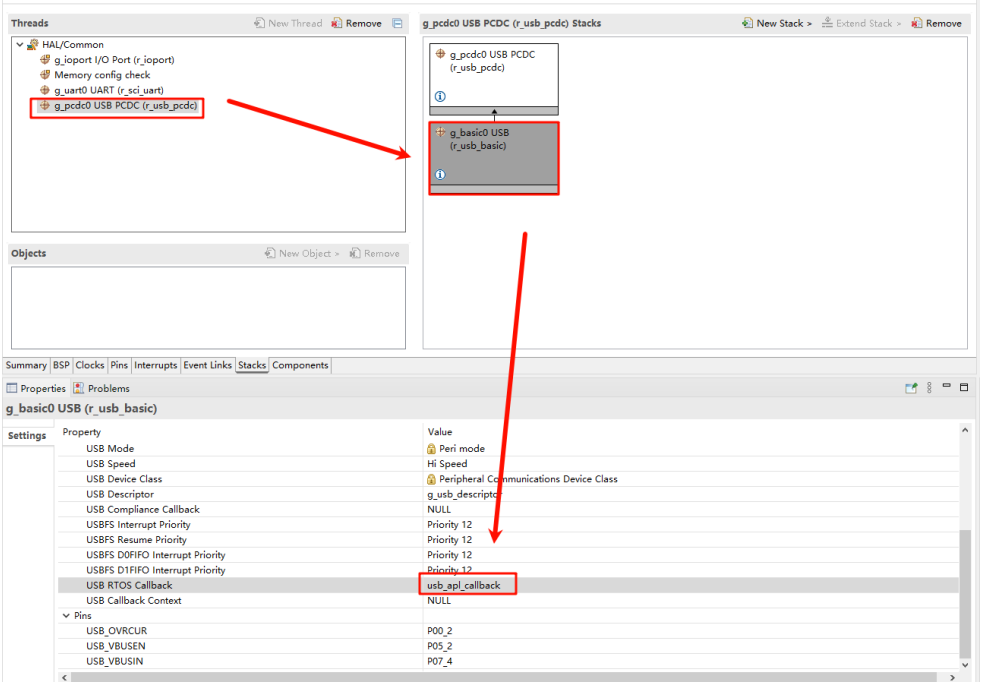


图 18-4 添加 USB callback

下面配置 USB 引脚，找到 USB_HS 并使能；

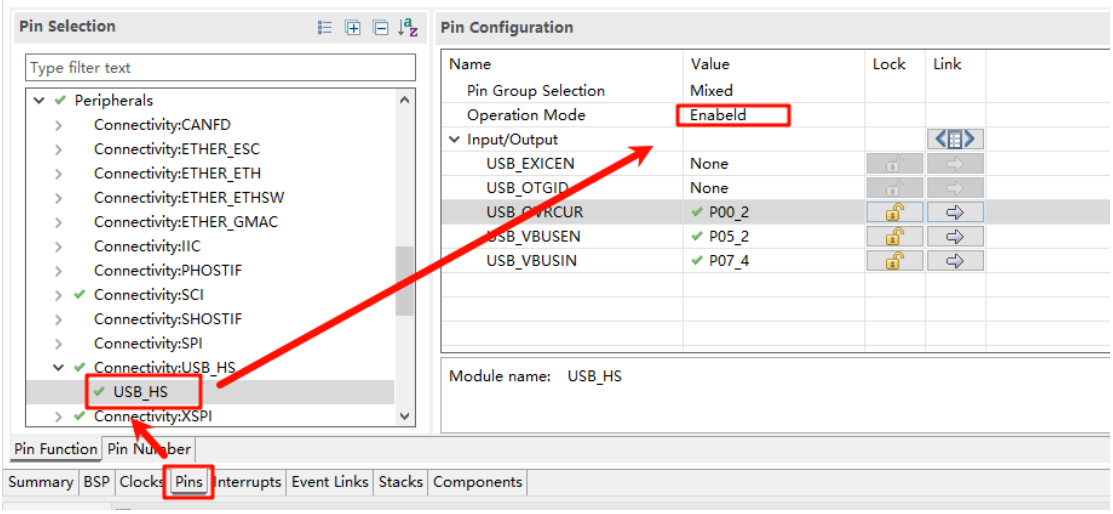


图 18-5 USB 引脚配置

18.3.2 构建配置

进入工程找到指定路径下的文件：.\rzn\SConscript，替换该文件为如下内容：

```
Import('RTT_ROOT')
Import('rtconfig')
from building import *
from gcc import *

cwd = GetCurrentDir()
src = []
group = []
CPPPATH = []

if rtconfig.PLATFORM in ['iccarm']:
    Return('group')
elif rtconfig.PLATFORM in GetGCCLikePLATFORM():
    if GetOption('target') != 'mdk5':
        src += Glob('./fsp/src/bsp/mcu/all/*.c')
        src += Glob('./fsp/src/bsp/mcu/all/cr/*.c')
        src += Glob('./fsp/src/bsp/mcu/r*/*.c')
```

```
        src += Glob('./fsp/src/bsp/cmsis/Device/RENESAS/Source/*.c')
        src += Glob('./fsp/src/bsp/cmsis/Device/RENESAS/Source/cr/*.c')
        src += Glob('./fsp/src/r_*/*.c')
        src += Glob('./fsp/src/r_usb_basic/src/driver/*.c')
        src += Glob('./fsp/src/r_usb_basic/src/hw/*.c')
        src += Glob('./fsp/src/r_usb_pcdc/src/*.c')
        CPPPATH = [ cwd + '/arm/CMSIS_5/CMSIS/Core_R/Include',
                          cwd + '/fsp/inc',
                          cwd + '/fsp/src/inc',
                          cwd + '/fsp/inc/api',
                          cwd + '/fsp/inc/instances',
                          cwd + '/fsp/src/r_usb_basic/src/driver/inc',
                          cwd + '/fsp/src/r_usb_basic/src/hw/inc',
                          cwd + '/fsp/src/r_usb_pcdc/src/inc',]

group = DefineGroup('rzn', src, depend = [''], CPPPATH = CPPPATH)
Return('group')
```

使用 studio 开发的话需要右键工程点击 **同步 scons 配置至项目**；如果是使用 IAR 开发请在当前工程下右键打开 env，执行：scons –target=iar 重新生成配置。

18.3.3 RT-Thread Settings 配置

USB 示例目前使用的是 freertos 接口驱动，因此我们还需要使能 Freertos 兼容层软件包；

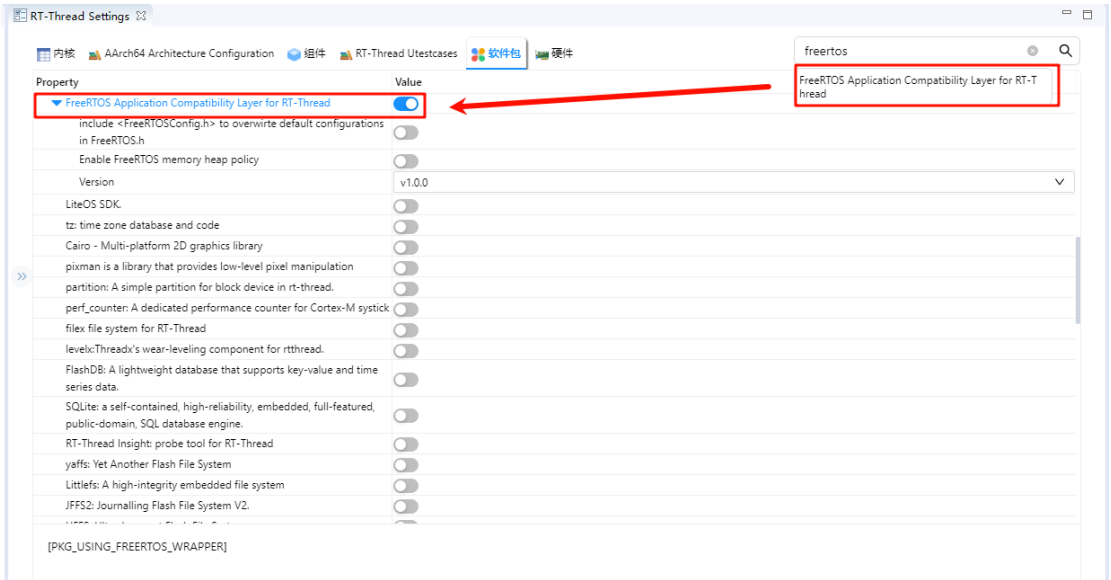


图 17-6 使能 freertos 兼容层

18.4 运行

18.4.1 编译&下载

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

编译完成后，将开发板的 Jlink 接口与 PC 机连接，然后将固件下载至开发板。

18.4.2 运行效果

打开串口工具，系统会自动初始化 USB-Device 为虚拟串口设备；

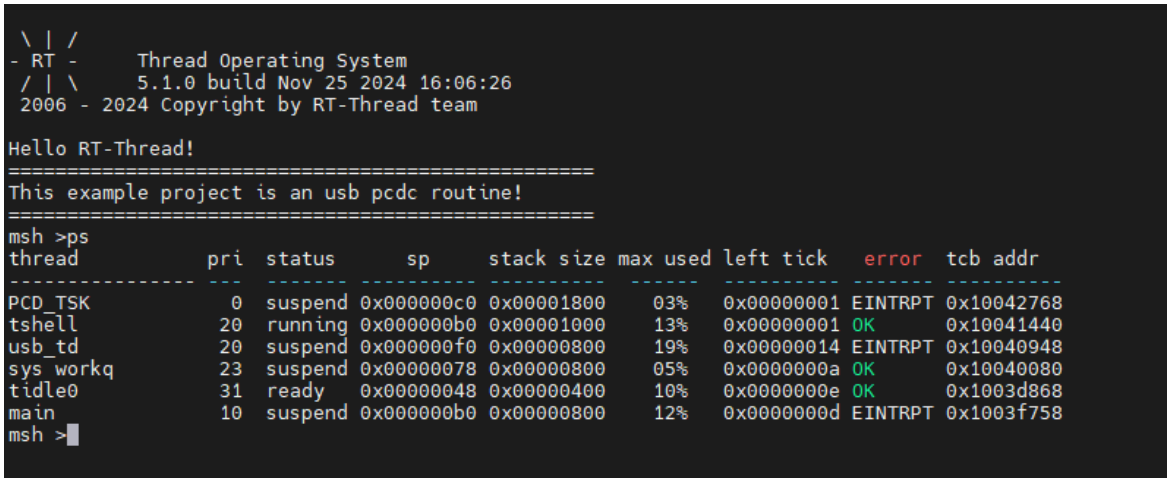


图 17-7 pcdc 线程信息

接着我们打开虚拟的串口设备，并测试字符输入；

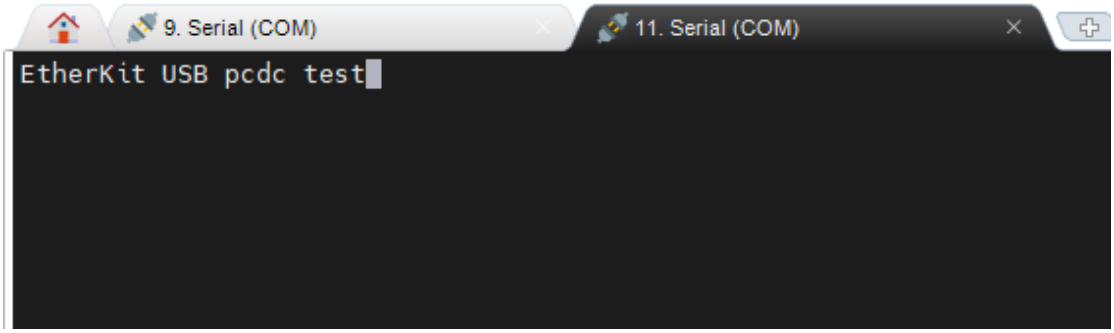


图 17-8 usb pcdc 测试

18.5 注意事项

暂无

18.6 引用参考

第 19 章 EtherCAT-EOE 例程

19.1 简介

EtherCAT EoE（**Ethernet over EtherCAT**）是 EtherCAT 协议中的一种通信协议，用于在 EtherCAT 网络上传输标准以太网数据包。它允许非实时的以太网通信与实时的 EtherCAT 通信共存，为工业自动化系统提供了灵活的网络集成能力。

以下是 EoE 的主要特点和功能：

1. **以太网隧道传输：**
    - EoE 在 EtherCAT 通信帧中封装标准的以太网数据包，使标准以太网通信协议（如 TCP/IP、UDP、HTTP 等）可以通过 EtherCAT 网络传输。
2. **扩展网络功能：**
    - 支持将 EtherCAT 从站作为虚拟以太网设备加入到 TCP/IP 网络中。
    - 允许通过 EtherCAT 通信链路访问远程的标准以太网设备。
3. **高效整合：**
    - EoE 的实现不会影响 EtherCAT 的实时性能。
    - 非实时的以太网通信与实时的 EtherCAT 数据交换能够共存，各司其职。
4. **使用场景：**
    - **设备管理：**通过 IP 协议访问 EtherCAT 从站设备（如远程配置、诊断和固件更新）。
    - **混合网络：**集成需要标准以太网通信的设备（如摄像头、传感器

或工控机）。

5. **简化网络布线：**
    - 在工业自动化场景中，EoE 允许通过 EtherCAT 网络访问以太网设备，从而减少了独立以太网布线的需求。
6. **典型应用：**
    - 工厂自动化系统中的远程监控和诊断。
    - 工业机器人或生产设备与外部 IT 系统的通信桥接。

本节将演示如何使用 Beckhoff TwinCAT3 和 EtherKit 开发板实现 EtherCAT EOE 主从站通信。

19.2 前期准备

软件环境：

- RT-Thread Studio
- RZN-FSP v2.0.0
- Beckhoff Automation TwinCAT3

硬件环境：

- EtherKit 开发板
- 网线一根
- Jlink 调试器

19.3 TwinCAT3 配置

在启动 TwinCAT3 之前，我们还需要做一些配置操作：

19.3.1 安装 ESI 文件

启动 TwinCAT 之前，将发布文件夹中包含的 ESI 文件复制到 TwinCAT 目标位置：“..\TwinCAT\3.x\Config\IO\EtherCAT”

注意：当前版本的 ESI 文件位于：..\board\ports\ethercat\ESI_File\Renesas EtherCAT RZN2 EoE.xml”

图 19-1 安装 ESI 文件

19.3.2 添加 TwinCAT 网卡驱动

添加 TwinCAT 的以太网驱动程序。（仅限首次使用配置即可）。从开始菜单中，选择 [TwinCAT] → [Show Realtime Ethernet Compatible Devise…]，从通信端口中选择连接的以太网端口并安装。

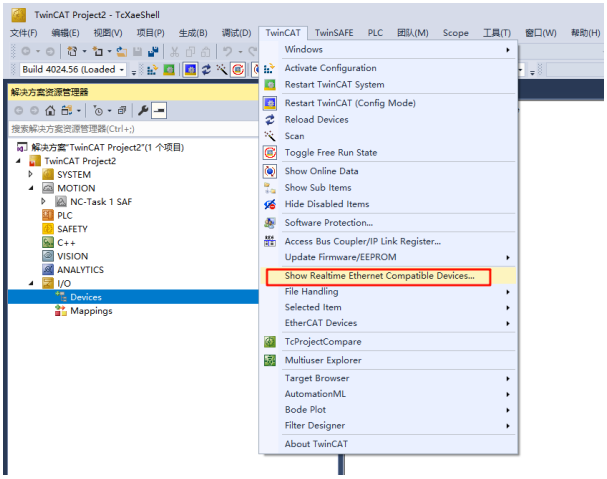


图 19-2 查看支持的以太网适配器

在这里我们能看到目前 PC 端的所有以太网适配器信息，选择我们测试要用的端口后，点击安装：

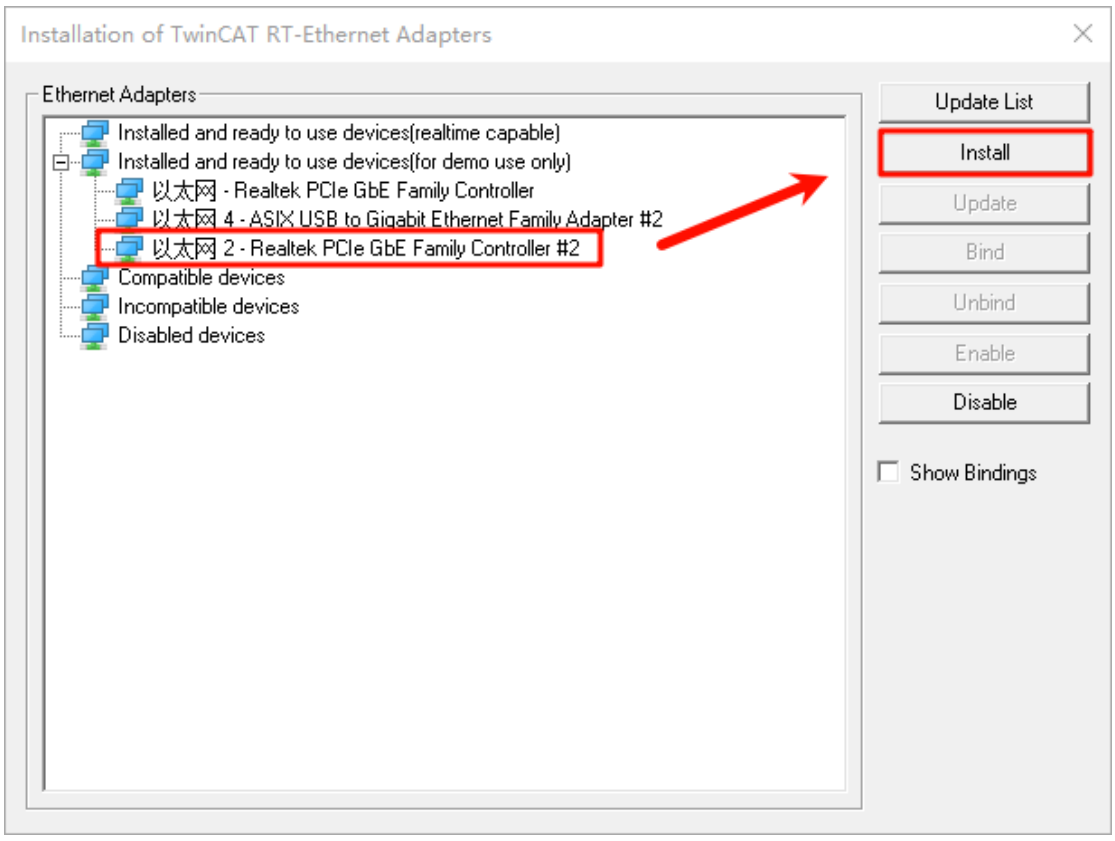


图 19-3 安装 TwinCAT 网卡驱动

检查网络适配器，可以看到已经成功安装了

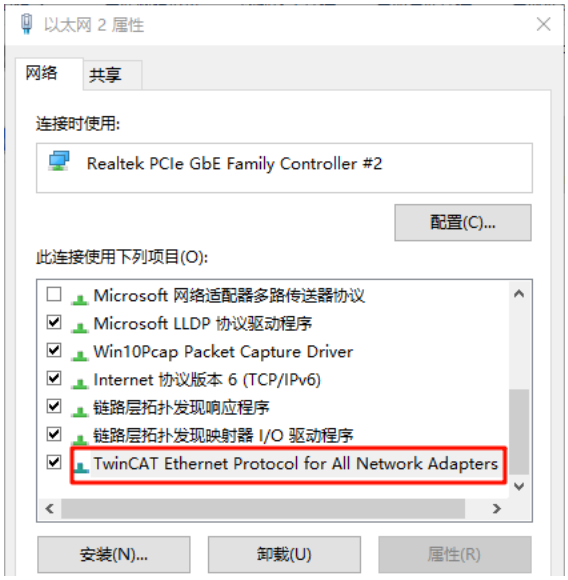


图 19-4 检查 TwinCAT 网卡驱动

19.4 FSP 及 Studio 配置

19.4.1 FSP 配置

接下来就是引脚初始化配置了，打开安装的 RZN-FSP 2.0.0，选择我们工程的根目录：

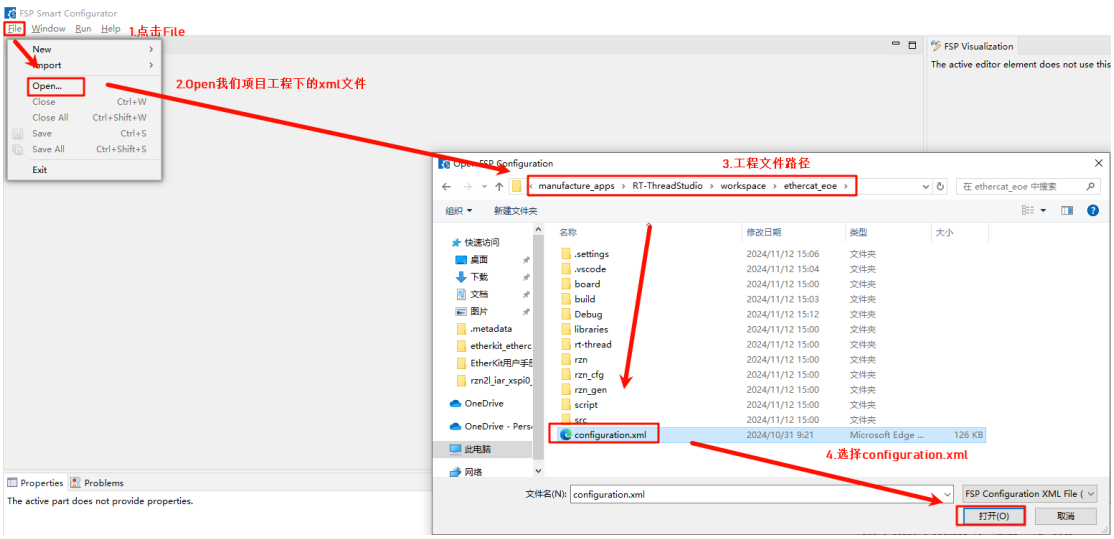


图 19-5 打开 fsp 配置

我们进行以下外设及引脚的配置：点击 New Stack，并添加 ethercat_ssc_port 外设：

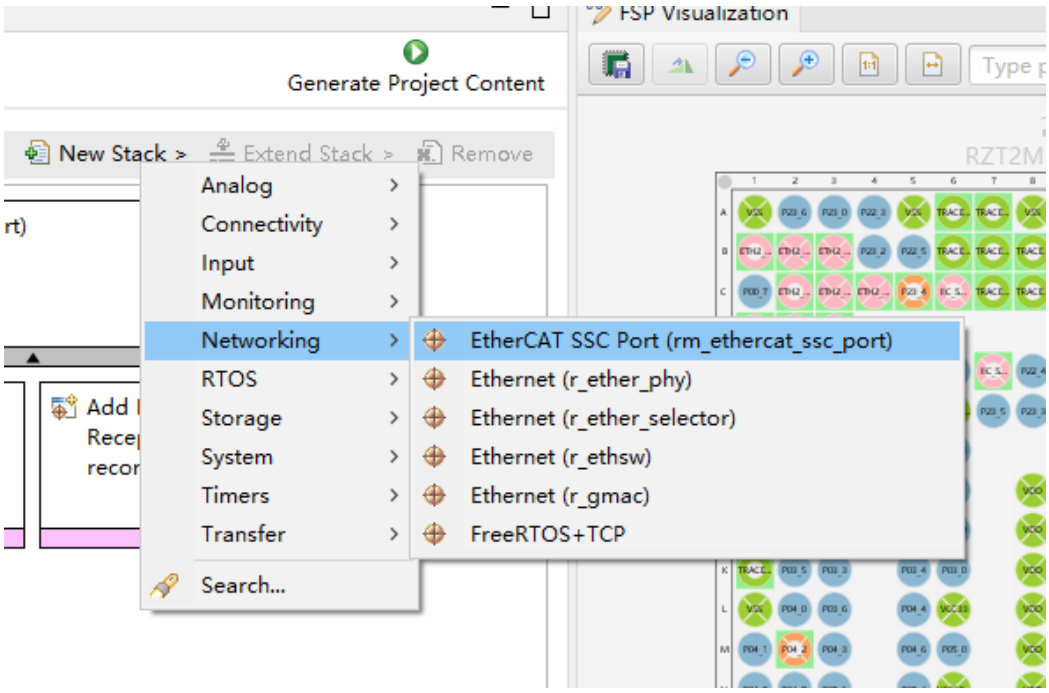


图 19-6 添加 ethercat_ssc_port 外设

配置 ethercat_ssc_port：修改 Reset Port 为 P13_4，同时 EEPROM_Size 大小设置为 Under 32Kbits；

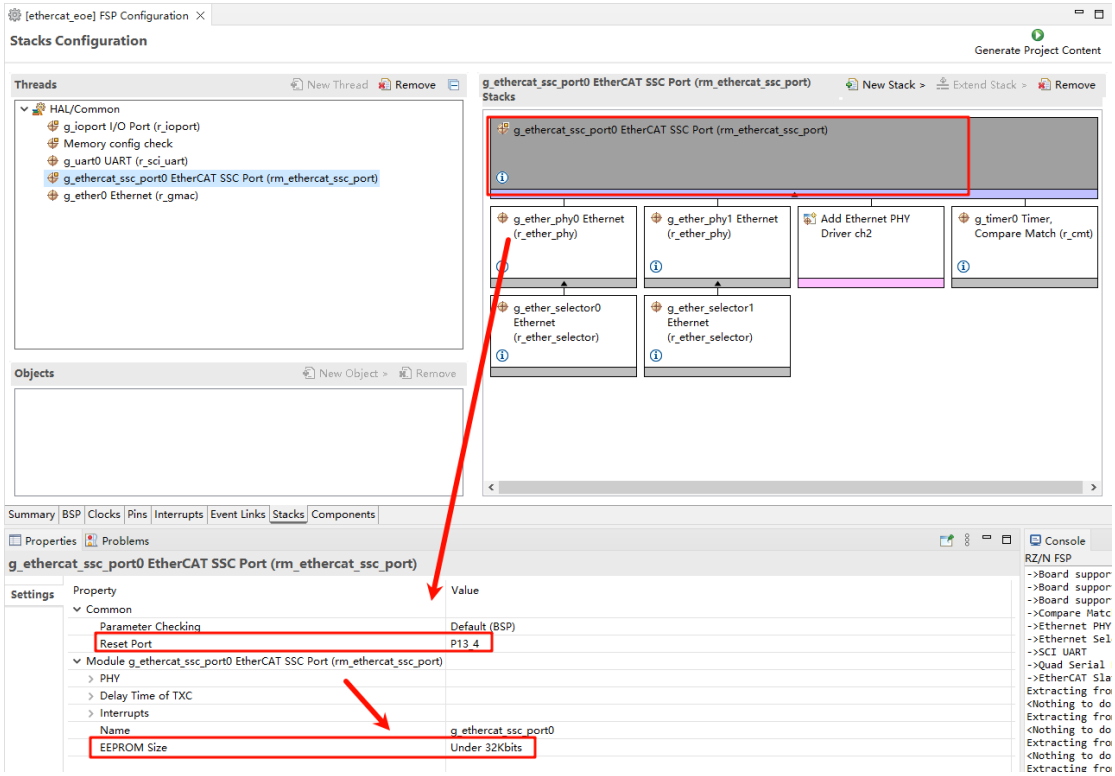


图 19-7 ethercat_ssc_port 配置

使能网卡类型、配置网卡设备参数，这里我们添加两个 phy（phy0 和 phy

1），其中需要注意的是，EtherKit 使用的是 rtl8211 网卡，并不在瑞萨 FSP 的支持范围内，但好在瑞萨预留了用户自定义网卡接口，因此按照如下设置来配置网卡，同时设置 MDIO 类型为 GMAC，设置网卡初始化回调函数 ether_phy_targets_initialize_rtl8211_rgmii()；

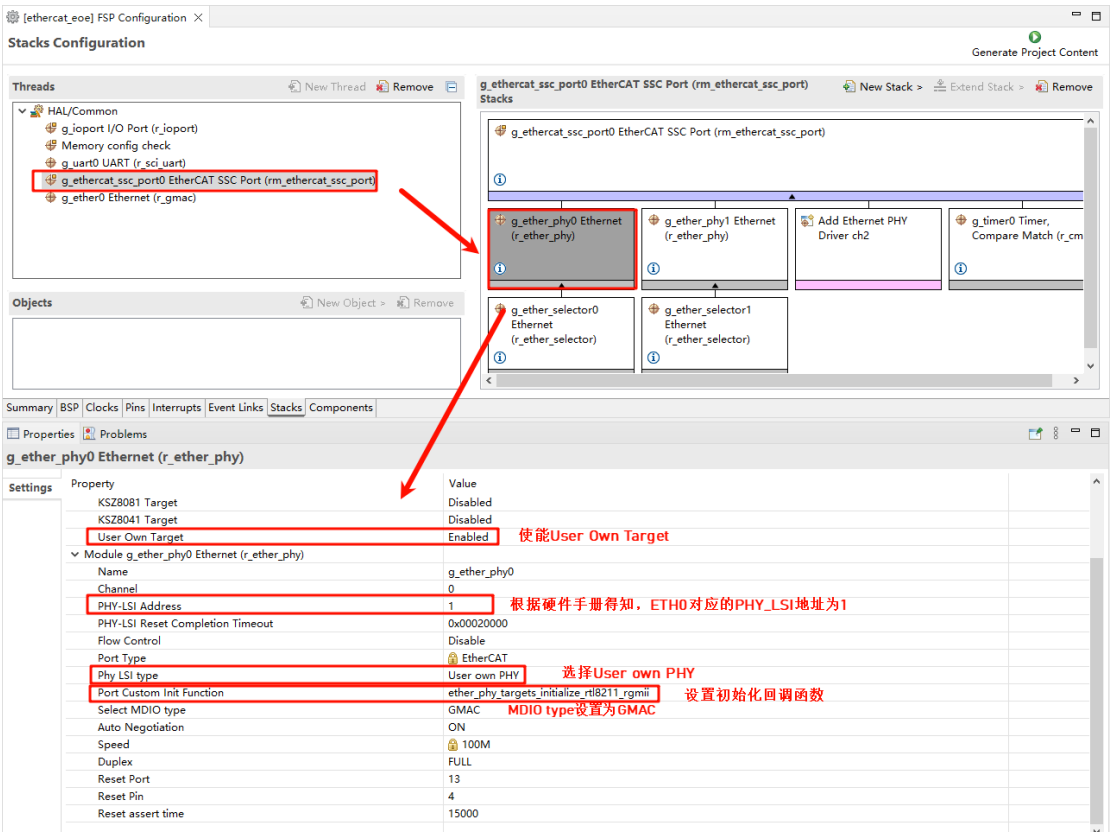


图 19-8 PHY 配置

网卡引脚参数配置，选择操作模式为 RGMII：

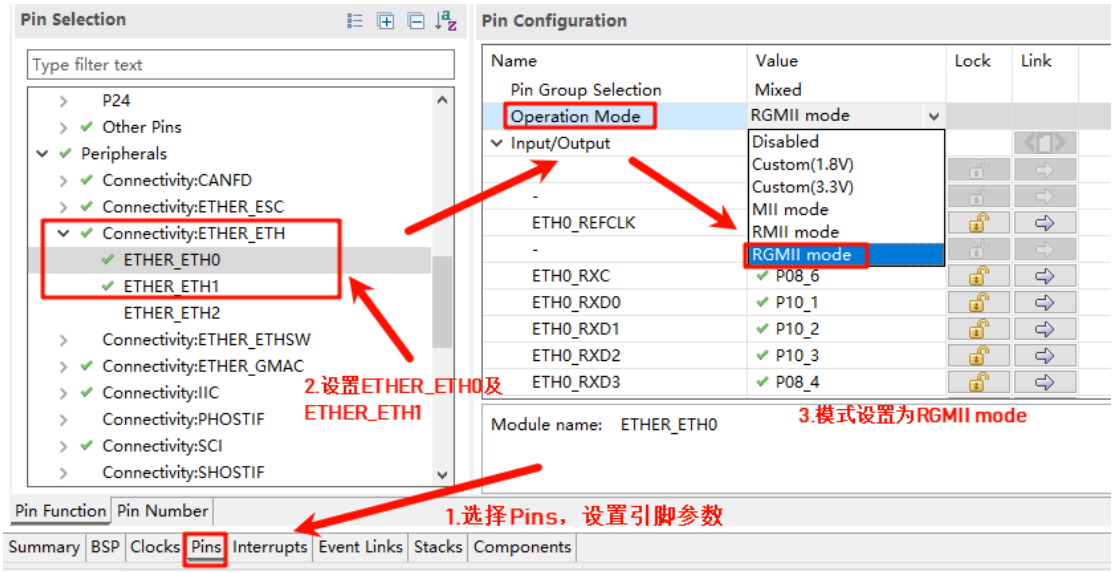


图 19-9 ETH 引脚配置

ETHER_ESC 设置：

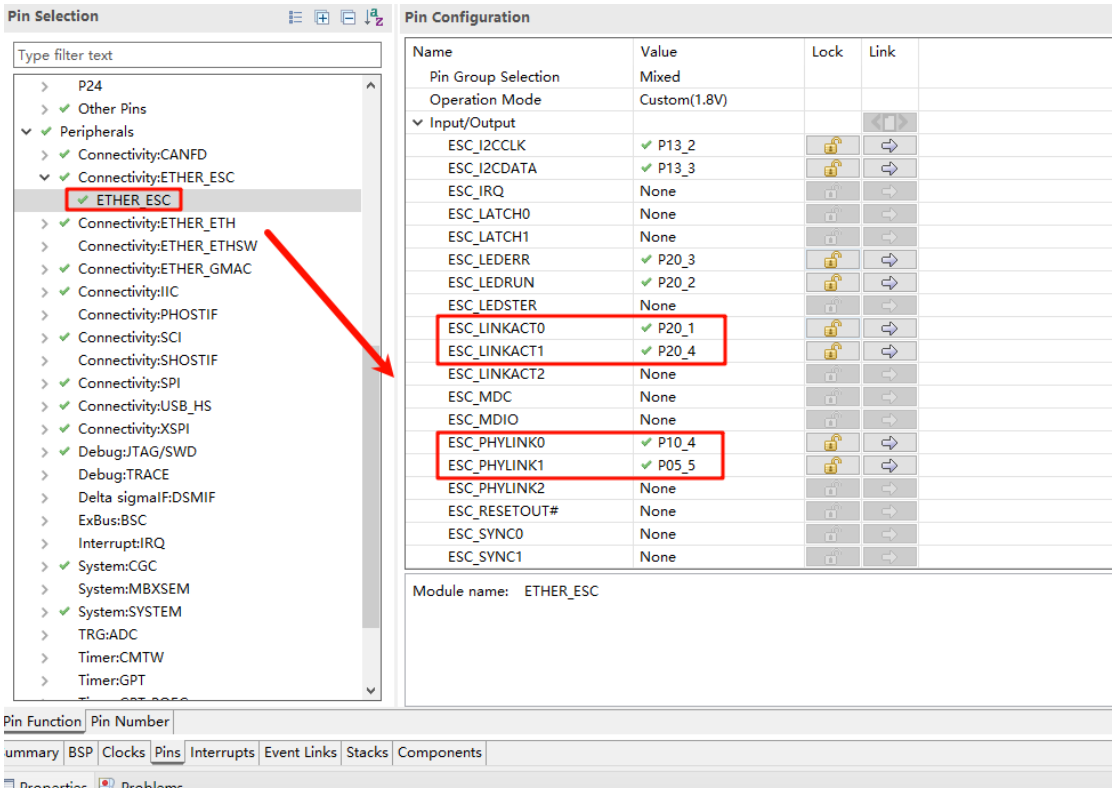


图 19-10 ESC 引脚配置

ETHER_GMAC 配置：

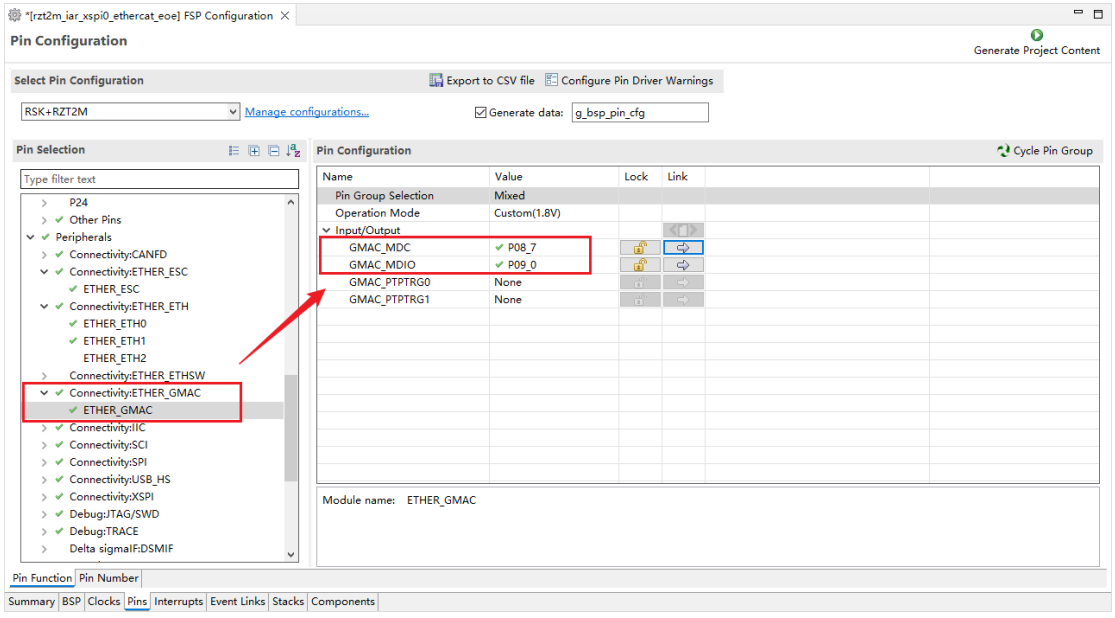


图 19-11 GMAC 引脚配置

为 ethercat_ssc_port 添加 cmt 定时器并配置中断优先级：

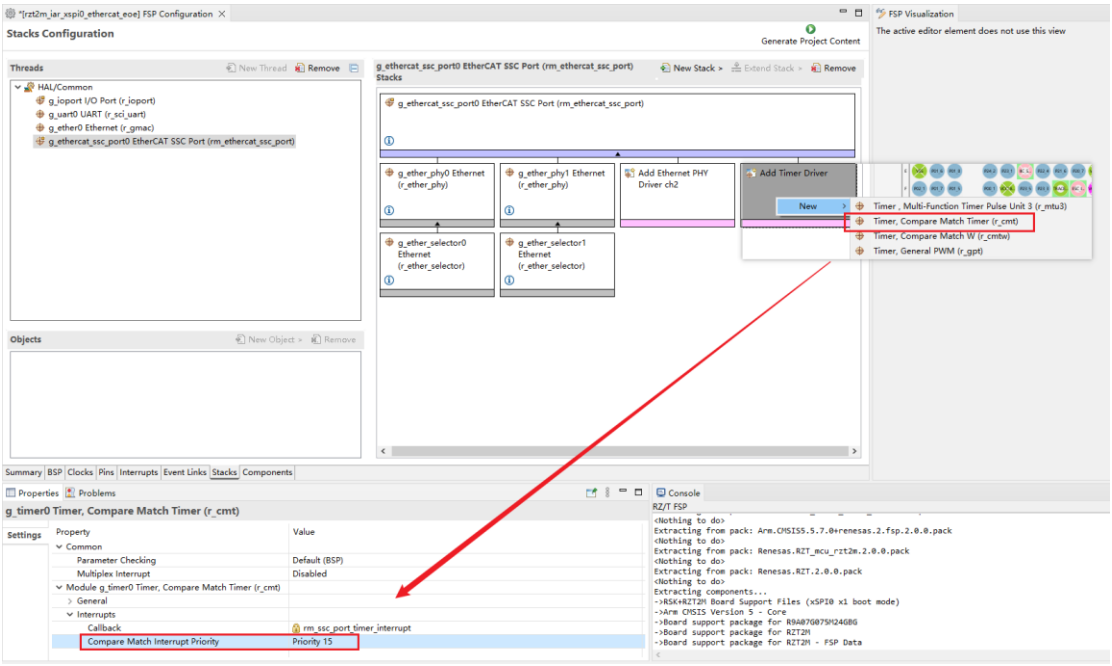


图 19-12 添加 CMT 定时器

添加 Ethernet 外设：

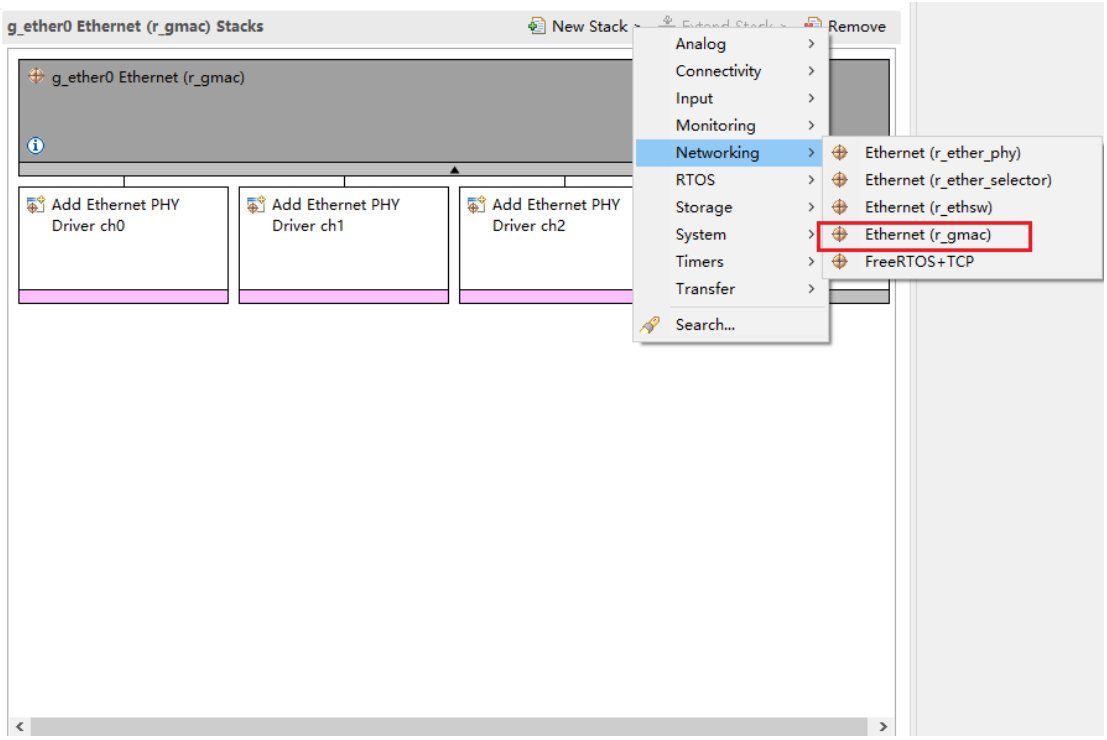


图 19-13 添加 Ethernet 外设

ethernet 中断触发回调设置为：user_ether0_callback

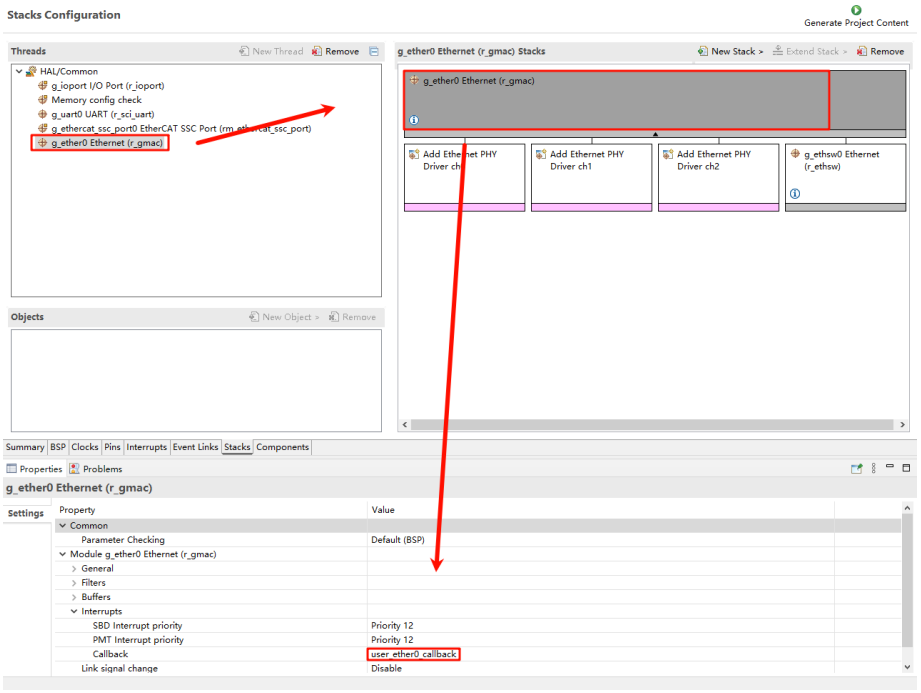


图 21-14 ETH 中断回调

最后点击 Generate Project Content 生成底层驱动源码。

19.4.2 构建配置

1.修改 sconscript：进入工程找到指定路径下的文件：.\rzn\SConscript，替换该文件为如下内容：

```
Import('RTT_ROOT')
Import('rtconfig')
from building import *
from gcc import *

cwd = GetCurrentDir()
src = []
group = []
CPPPATH = []

if rtconfig.PLATFORM in ['iccarm']:
    Return('group')
elif rtconfig.PLATFORM in GetGCCLikePLATFORM():
    if GetOption('target') != 'mdk5':
        src += Glob('./fsp/src/bsp/mcu/all/*.c')
```

```
        src += Glob('./fsp/src/bsp/mcu/all/cr/*.c')
        src += Glob('./fsp/src/bsp/mcu/r*/*.c')
        src += Glob('./fsp/src/bsp/cmsis/Device/RENESAS/Source/*.c')
        src += Glob('./fsp/src/bsp/cmsis/Device/RENESAS/Source/cr/*.c')
        src += Glob('./fsp/src/r_*/*.c')
        CPPPATH = [ cwd + '/arm/CMSIS_5/CMSIS/Core_R/Include',
                          cwd + '/fsp/inc',
                          cwd + '/fsp/inc/api',
                          cwd + '/fsp/inc/instances',]

if GetDepend('BSP_USING_ETHERCAT_EOE'):
    src += Glob('./fsp/src/rm_ethercat_ssc_port/*.c')
    CPPPATH += [cwd + '/fsp/src/rm_ethercat_ssc_port']

group = DefineGroup('rzn', src, depend = [''], CPPPATH = CPPPATH)
Return('group')
```

2.Kconfig 修改：打开工程下的文件（projects\etherkit_ethercat_eoe\board\Kconfig），在 Onboard Peripheral Drivers 选项中加入 EOE 配置：

```
        config BSP_USING_ETHERCAT_EOE
            bool "Enable EtherCAT EOE example"
            select BSP_USING_ETH
            default n
            if BSP_USING_ETHERCAT_EOE
                config RT_LWIP_IPADDR
                    string "set static ip address for eoe slaver"
                    default "192.168.10.100"
                config RT_LWIP_GWADDR
                    string "set static gateway address for eoe slaver"
                    default "192.168.10.1"
                config RT_LWIP_MSKADDR
                    string "set static mask address for eoe slaver"
                    default "255.255.255.0"
            endif
```

如下图所示：

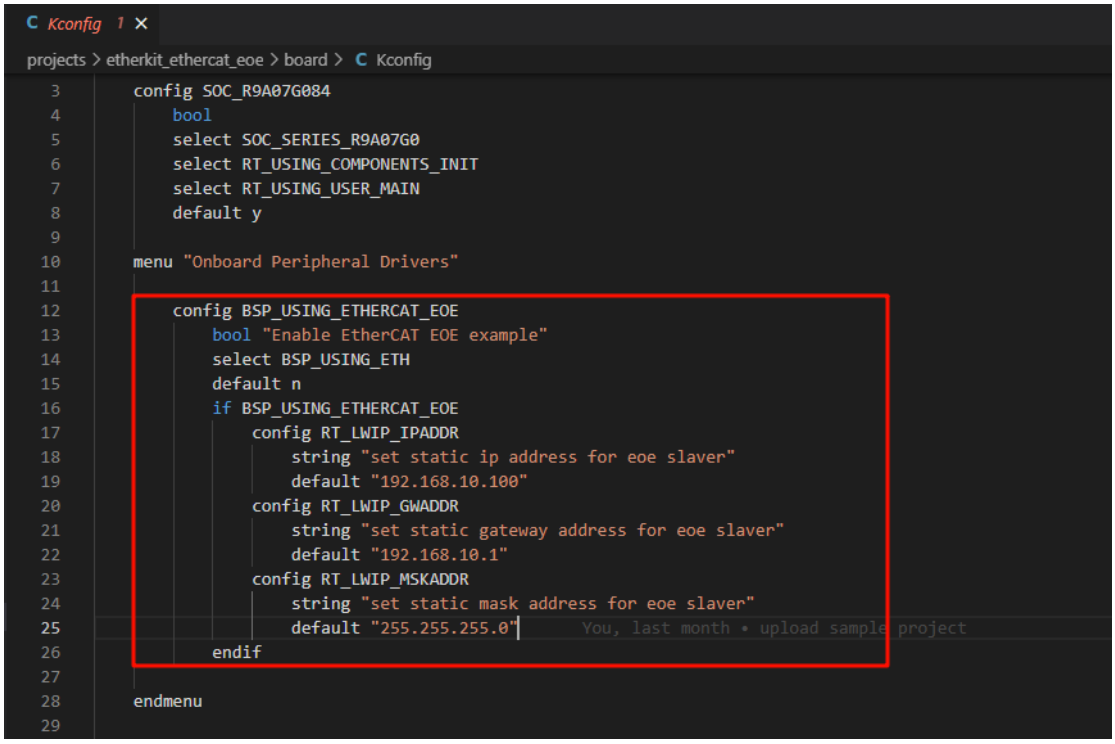


图 19-15 EOE 配置

3.使用 studio 开发的话需要右键工程点击 **同步 scons 配置至项目**；如果是使用 IAR 开发请在当前工程下右键打开 env，执行：scons –target=iar 重新生成配置。

19.4.3 RT-Thread Studio 配置

完成 FSP 配置之后，引脚及外设的初始化就暂告一段落了，接下来需要我们使能 EtherCAT EOE 示例，打开 Studio，点击 RT-Thread Settings，使能 EOE 示例：

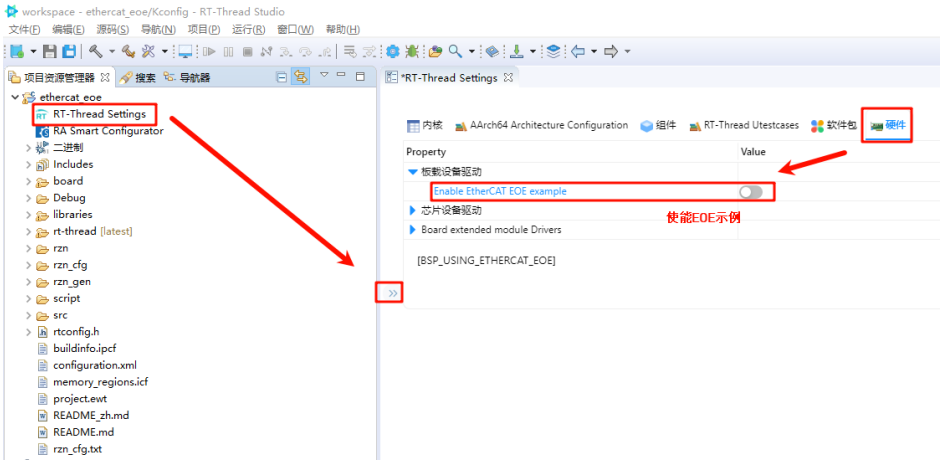


图 19-16 settings 使能 EOE

下面我们还需要配置禁用 dhcp 功能并使用静态 IP，点击组件->使能 lwip 堆栈，选择禁用 DHCP；

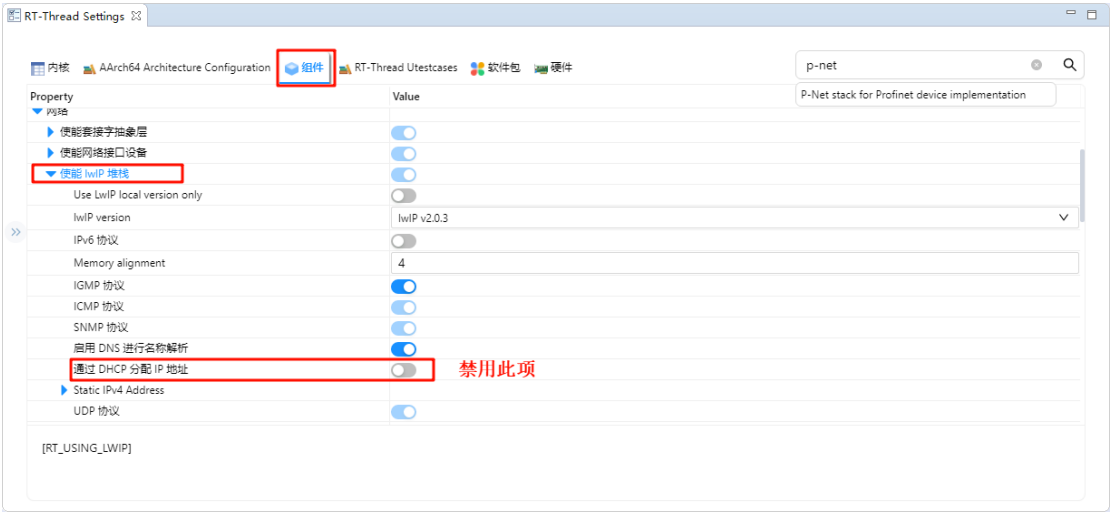


图 19-17 禁用 dhcp

使能完毕后我们保存 settings 配置并同步 scons 配置，同时编译并下载程序，复位开发板后观察串口日志：

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Nov 26 2024 16:27:24
 2006 - 2024 Copyright by RT-Thread team
lwIP-2.0.3 initialized!
[I/sal.skt] Socket Abstraction Layer initialize success.

Hello RT-Thread!
=================================================
This example project is an ethercat eoe routine!
=================================================
msh >[I/DBG] link up
=================================================
EtherCAT Slave with EOE Project!
=================================================

msh >if
ifconfig
msh >ifconfig
network interface device: e0 (Default)
MTU: 1500
MAC: 00 11 22 33 44 55
FLAGS: UP LINK_UP INTERNET_DOWN DHCP_DISABLE ETHARP BROADCAST IGMP
ip address: 10.21.8.22
gw address: 10.21.8.190
net mask   : 255.255.255.0
dns server #0: 0.0.0.0
dns server #1: 0.0.0.0
msh >
```

图 19-18 EOE 串口日志

19.5 EtherCAT EOE 配置

19.5.1 新建 TwinCAT 工程

打开 TwinCAT 软件，点击文件->新建->新建项目，选择 TwinCAT Projects，创建 TwinCAT XAR Project(XML format)工程：

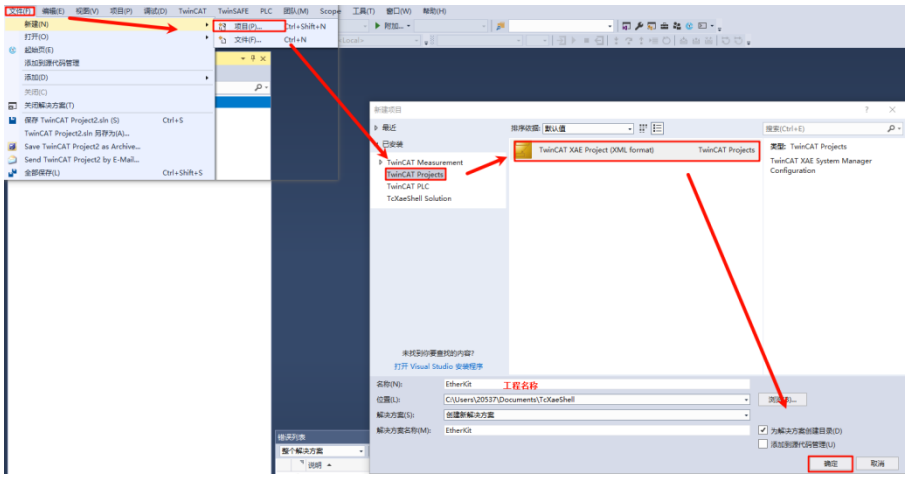


图 19-19 新建 TwinCAT 工程

19.5.2 从站启动 EOE App

将 EtherKit 开发板上电后，需要使用网线连接 ETH0 网口，ethercat 会默认运行。

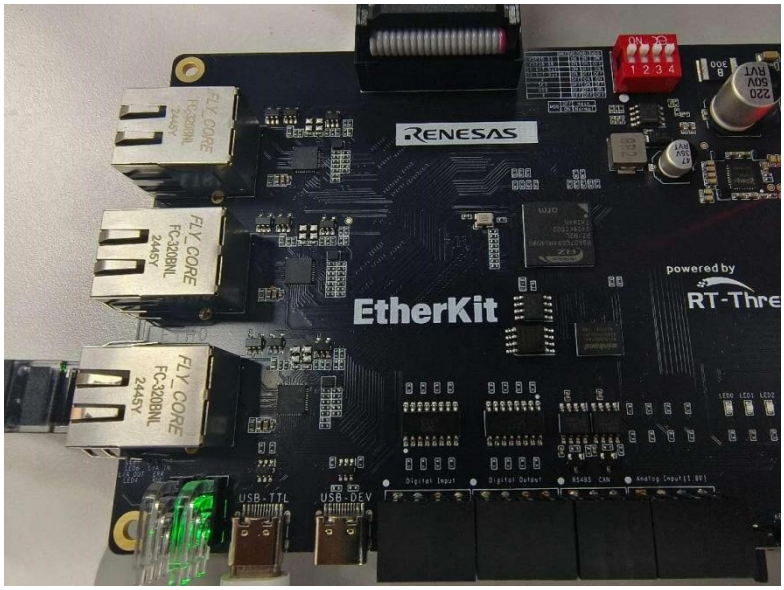


图 19-20 从站 EOE 启动

19.5.3 从站设备扫描

新建工程之后，在左侧导航栏找到 Devices，右键选择扫描设备。正常来说如果扫描从站设备成功的话是会显示：Device x[EtherCAT]；而扫描失败则显示的是：Device x[EtherCAT Automation Protocol]，此时就代表从站初始化失败。

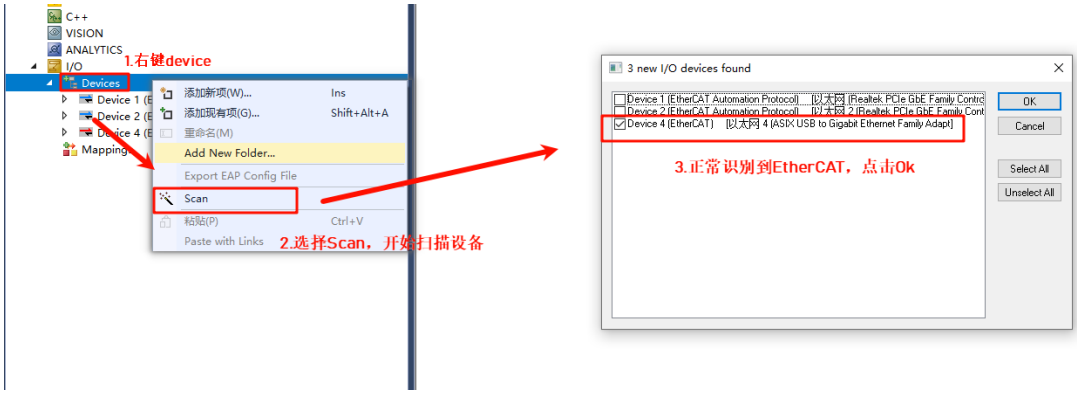


图 19-21 扫描设备

点击 Ok 后会弹出一个窗口：Scan for boxes，点击确认后，会再次弹出窗口：Activate Free Run，由于我们**首次使用 EOE 还需要更新 EEPROM 固件**，所以暂时先不激活。

19.5.4 更新 EEPROM 固件

回到 TwinCAT，在左侧导航栏中，由于我们已经成功扫描到从站设备，因此可以看到主从站的配置界面：

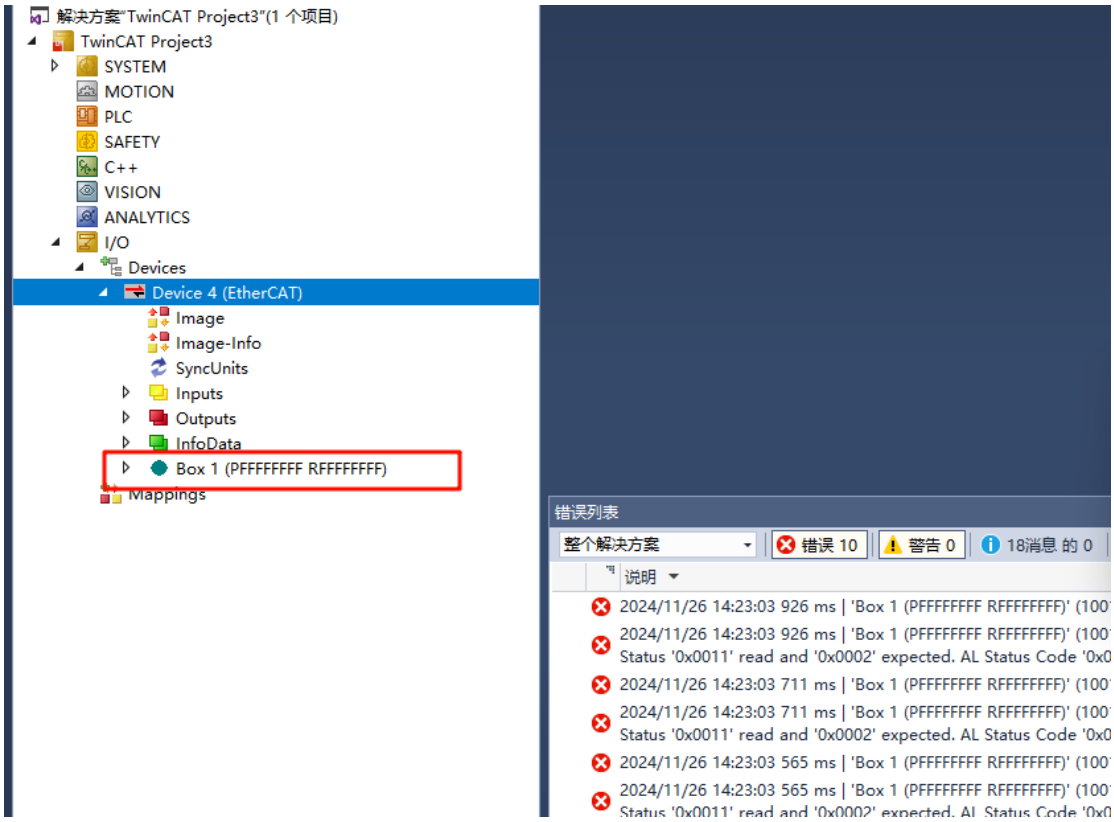


图 19-22 TwinCAT 配置界面

我们双击 Box 1，在中间界面的上方导航栏中单击 EtherCAT，并点击 Advanced Settings…：

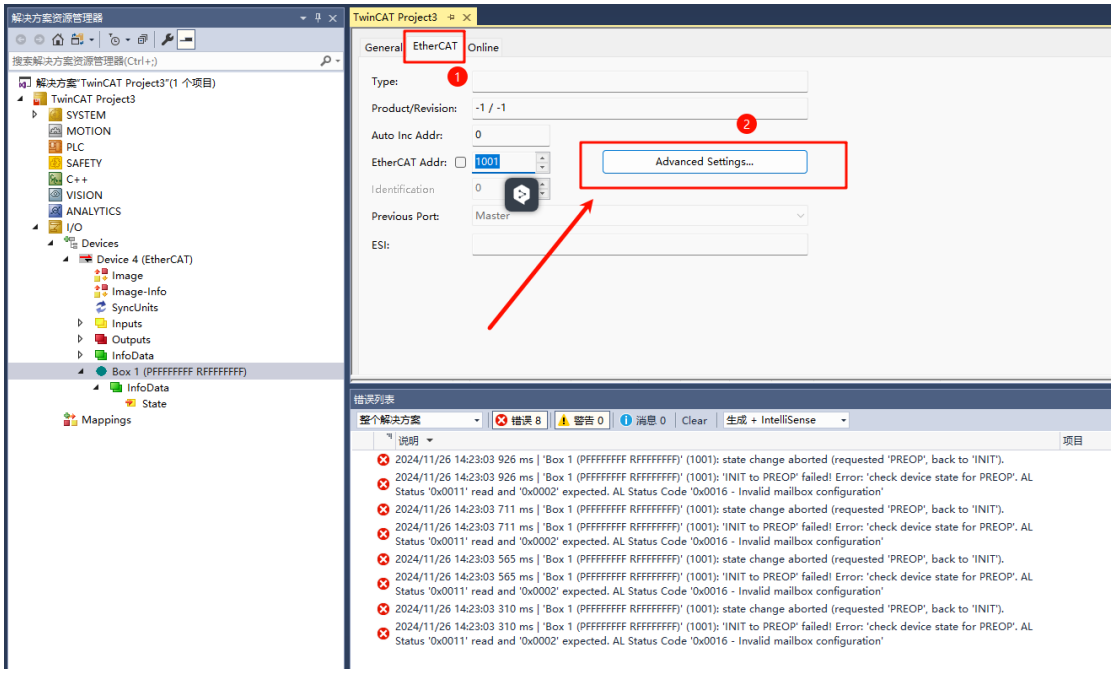


图 19-23 Advanced Settings

这里按图示点击 Download from List…：

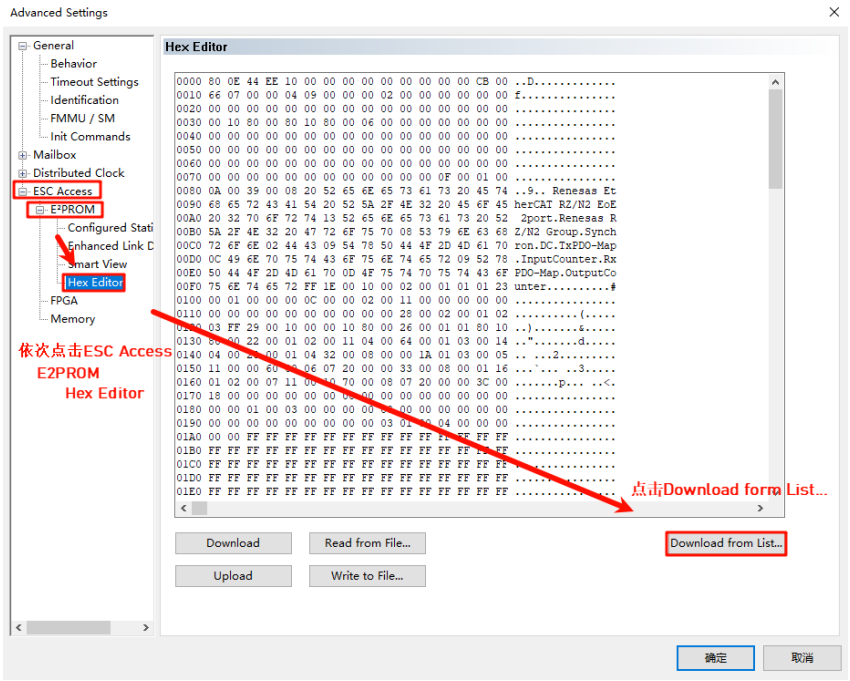


图 19-24 Hex Editor

我们写入 ESI 固件到 EEPROM 中，这里由于我们配置的是双网口，所以选择 Renesas EtherCAT RZ/N2 EOE 2port，如果你配置的是三网口的话则选择 3 port 后缀的 ESI 文件进行下载。

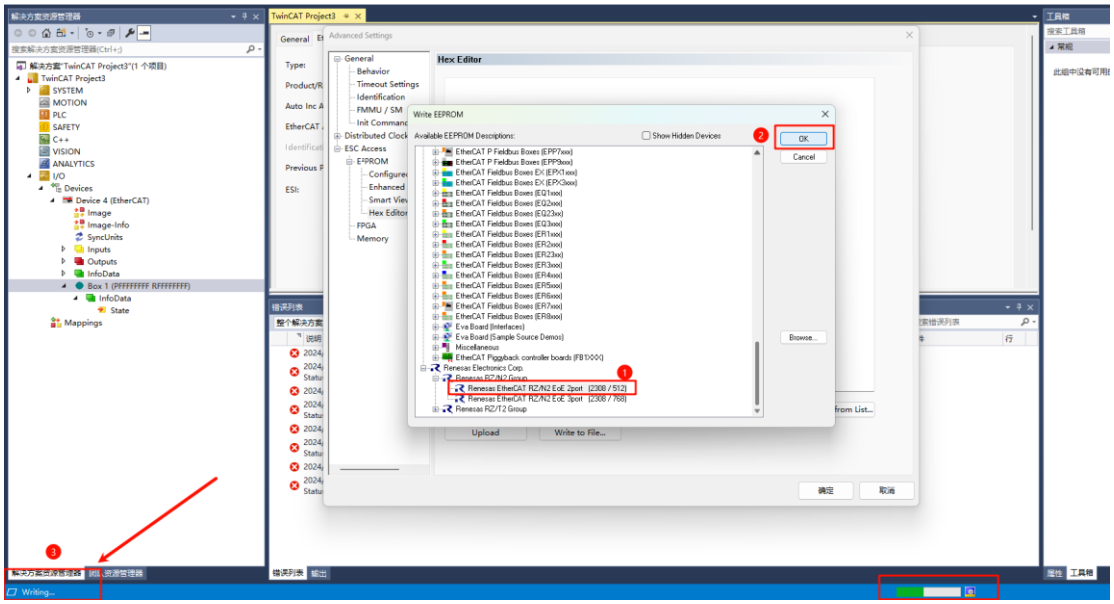


图 19-25 ESI 固件下载

下载完成之后，我们右键 Device x(EtherCAT)移除设备后重新扫描并添加设备，并完成激活工作（参考上文）。

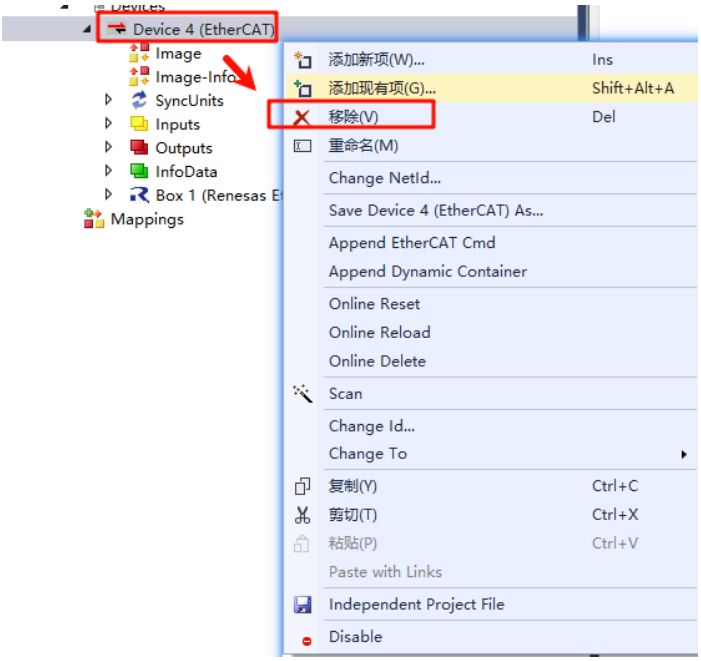


图 19-26 移除 Device

19.6 EtherCAT EOE 通信

在完成 EEPROM 下载 ESI 固件并重新扫描添加设备后，激活 Device 我们

可以观察到，板载有两颗绿色 LED 亮起（通信正常），并且其中一颗保持高频率闪烁一颗保持常亮，此时主从站就可以建立起正常的通信了。

图 19-27 EtherCAT 状态灯

19.6.1 EIO 测试

由于我们提供的 EOE 工程集成了 EIO 协议，因此可直接进行 EIO 测试，在本例程中，我们提供三个 USER LED 作为 EIO 的输入，回到 TwinCAT，依次点击 Device x(EtherCAT)->Box 1(Renesas EtherCAT RZ/N2 EOE 2port)->RxPD 0-Map->OutputCounter：

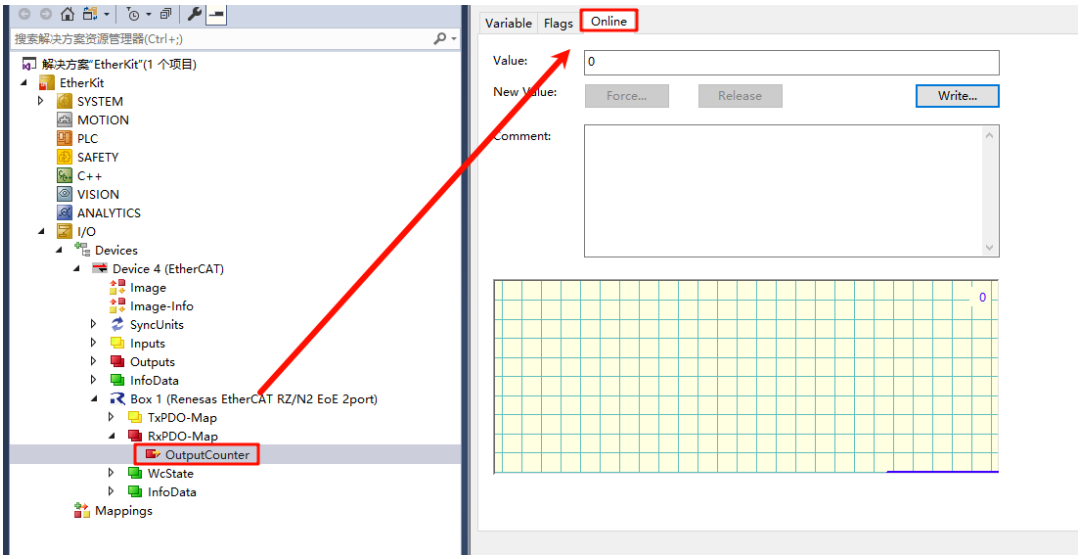


图 19-28 EtherCAT 从站输出

此时的开发板默认的三颗 USER LED 还处于灭灯状态，这里我们点击左上角的 Online，并且 Write Value：1

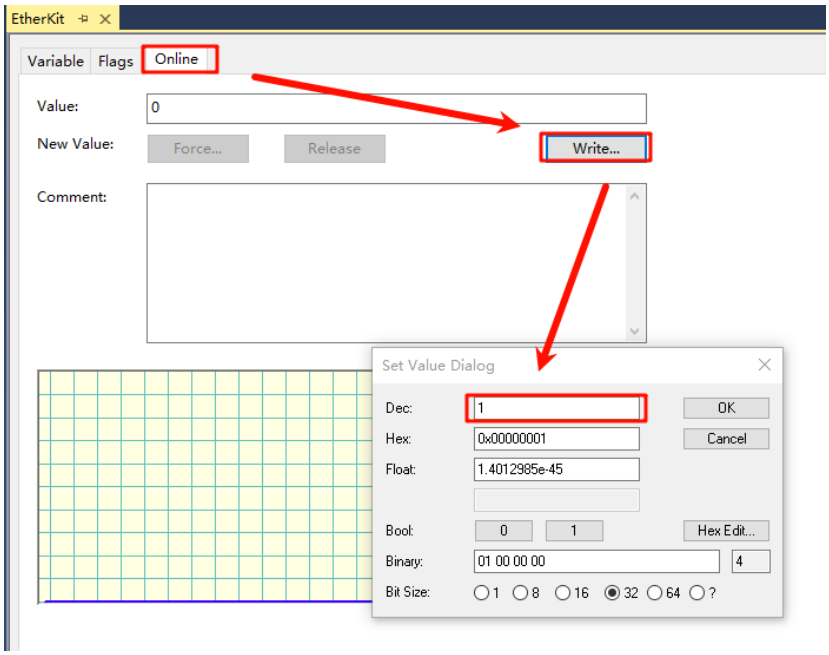


图 19-29 EIO 控制从站 LED

此时可以发现从站开发板同时亮起 LED0（红灯），EIO 测试正常，当然也可以随意尝试其他 value 组合，会有不同的 LED 阵列亮暗行为。

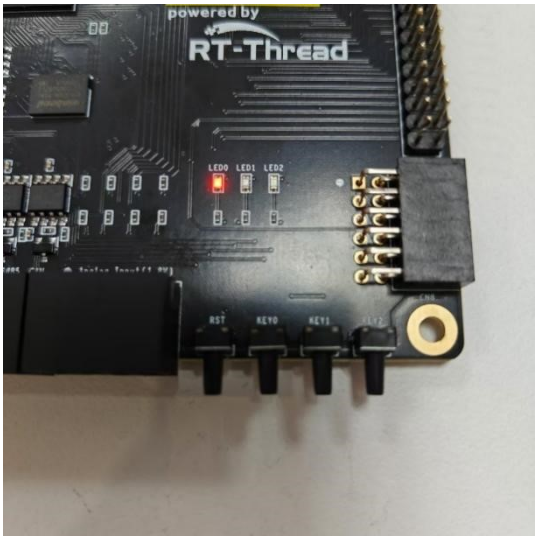


图 19-30 EIO 控制从站 LED

19.6.2 EOE 测试

打开以太网适配器，选择主站所使用的适配器并设置静态 IP：

图 19-31 主站 IP 设置

回到 TwinCAT，我们点击 Box 1，选择 EtherCAT->Advanced Settings…->MailBox->EOE->设置 IP Port，设置从站 IP 信息：

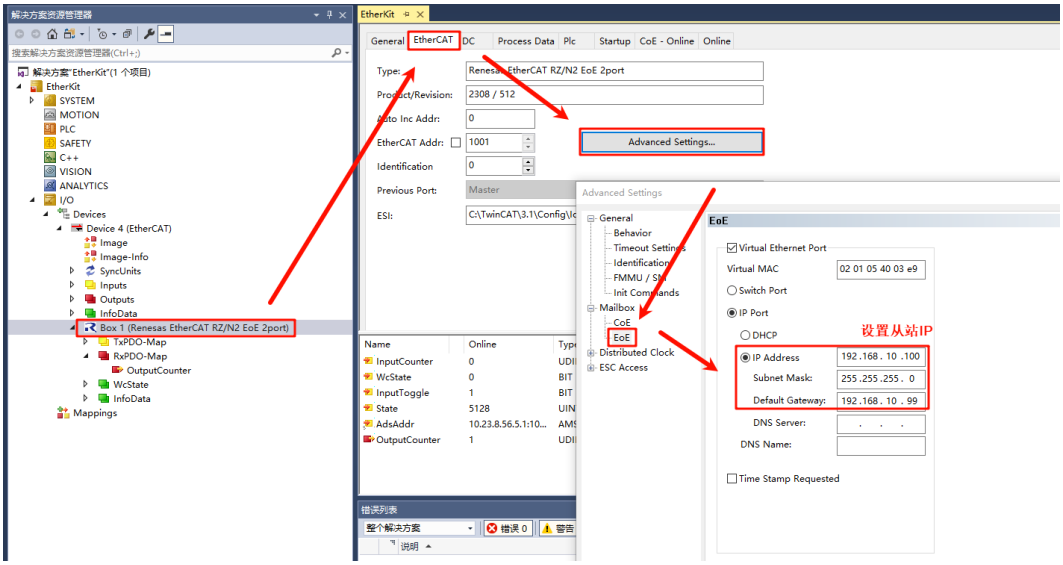


图 19-32 从站 IP 设置

完成这些配置后，我们就能测试使用 EtherCAT EOE 对主从站进行 ping 测试了：

- 主站 IP：192.168.10.99
- 从站 IP：192.168.10.100

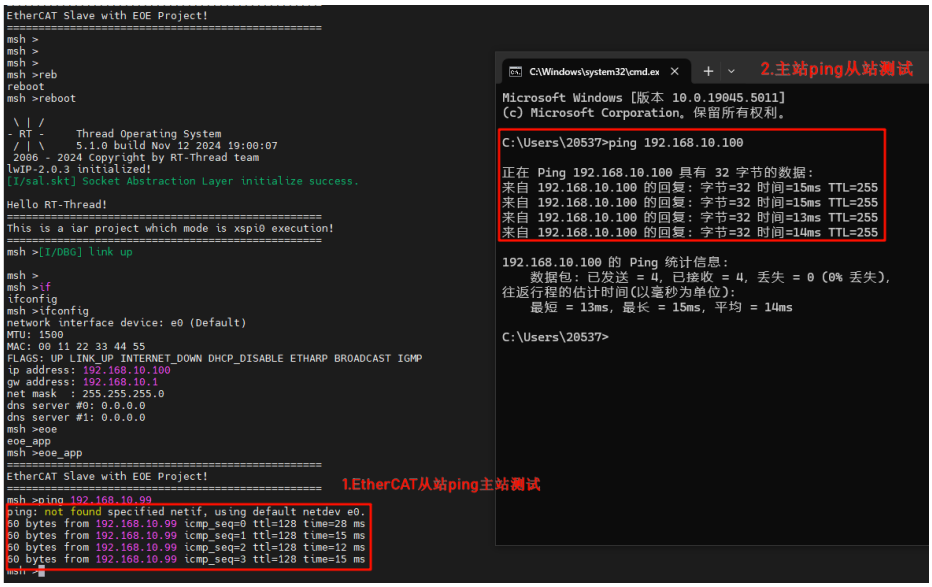


图 19-33 主从站使用 EOE 进行 ping 测试

19.7 拓展说明：3 端口以太网 EOE 通信

目前示例工程默认为 2 端口以太网 EOE，如需使用三网口 EOE 通信请遵循本章说明进行配置；

19.7.1 FSP 配置

首先仍然是打开工程下的 FSP 配置文件，我们为 SSC stack 添加第三个 phy；

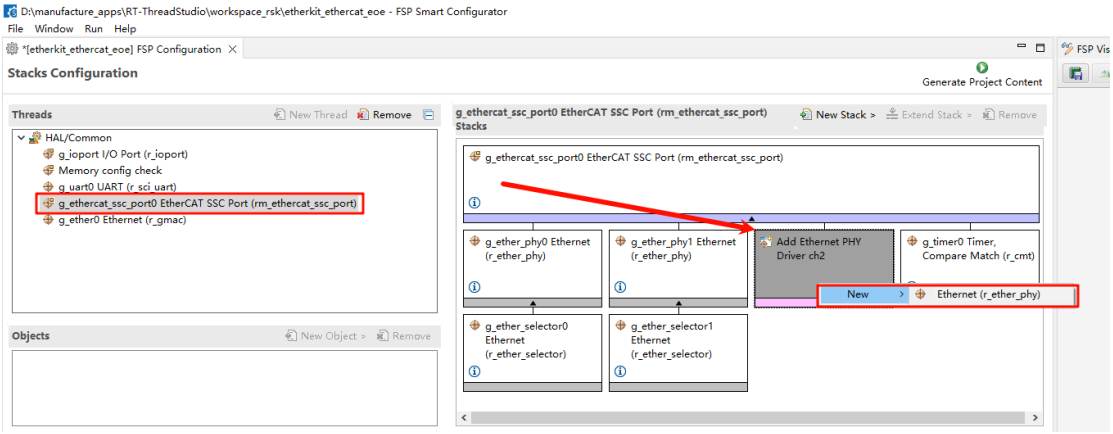


图 19-34 phy2 添加

然后配置 phy2 的通道数为 2，phy address 为 3（根据原理图手册查询可知），同时配置网卡型号为用户自定义，并且设置以太网初始化回调函数；

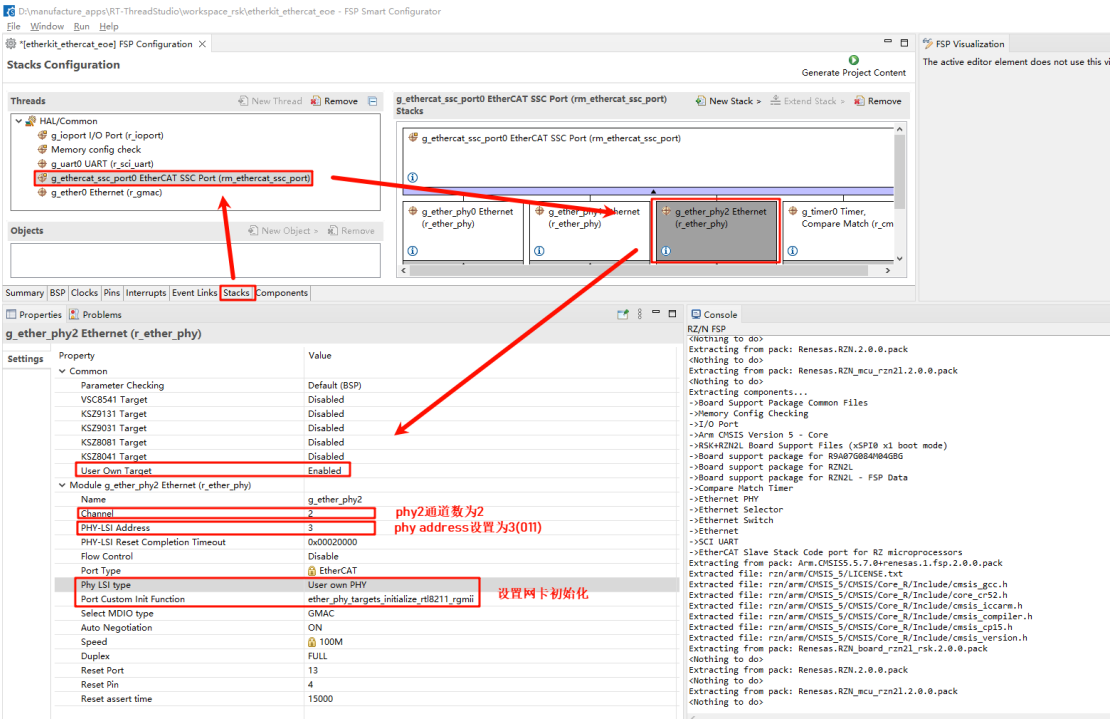


图 19-35 phy 参数配置

接下来配置引脚，使能 ETH2；

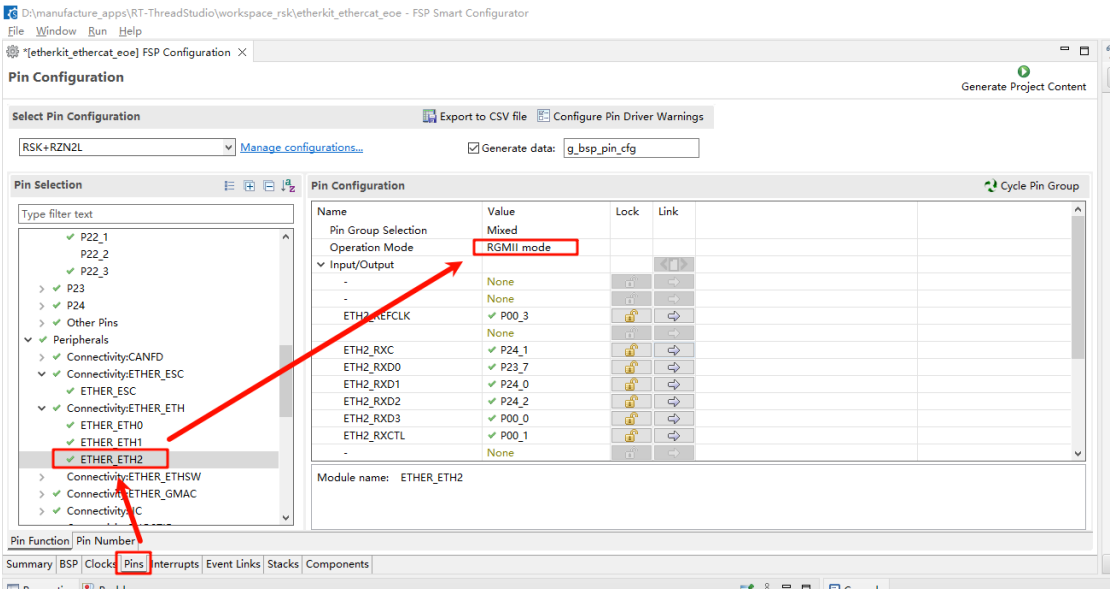


图 19-36 使能 ETH2

接着我们配置 ESC 对应 ETH2 的 LINK 引脚，分别配置 ESC_LINKACT2(P22_1)和 ESC_PHYLINK2(P00_5)；此处需要注意：**此处如果 P22_1 被占用，需要先手动将该引脚复用功能禁用后，再使能此项；**

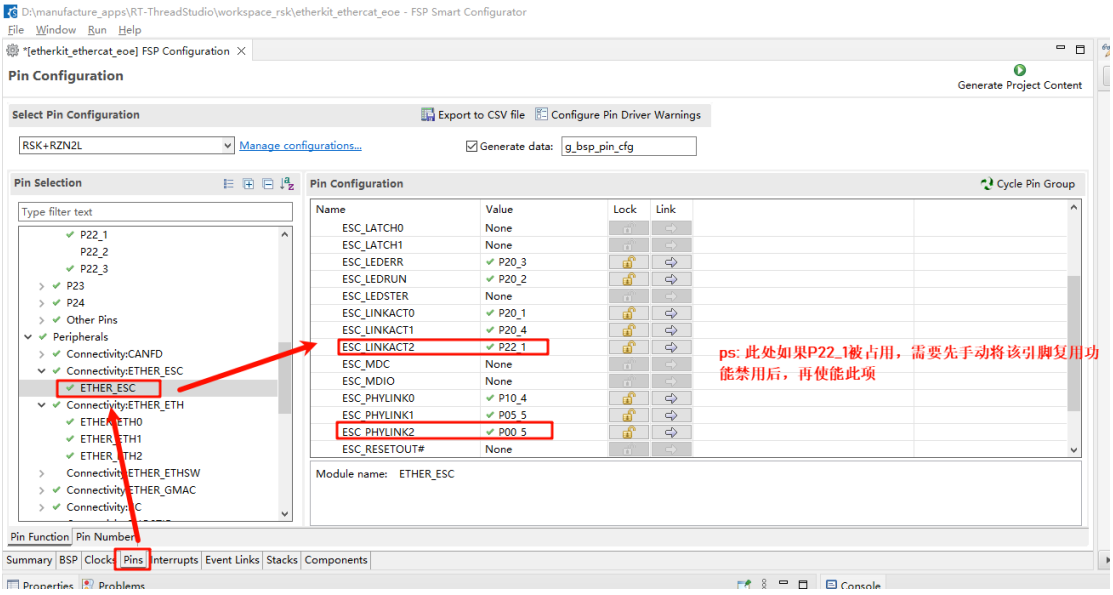


图 19-37 配置 ESC

完成上述配置后就可以点击生成源码了，回到工程编译并将程序下载开发板中；

19.7.2 ESI 固件更新

同样首先我们需要等待开发板 EOE 从站成功运行，接着我们打开 TwinCAT 3 软件扫描设备，扫描到 EtherCAT 设备后先暂时不激活，弹窗点击否即可；

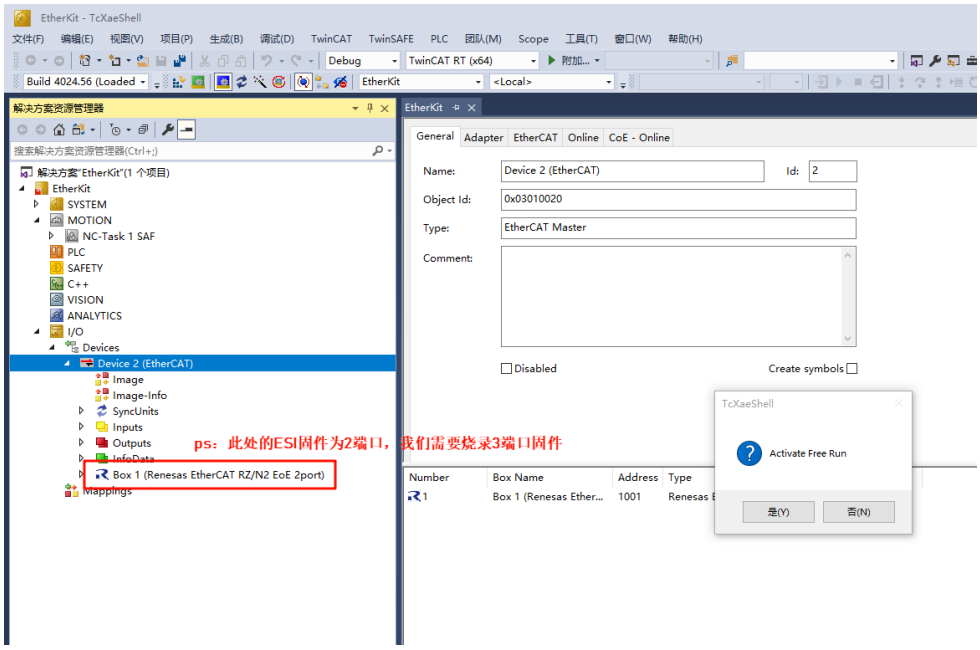


图 19-38 扫描从站设备

参考 19.5.4 章：更新 EEPROM 固件，一样的步骤，只不过这次需要选择更新的固件为：Renesas EtherCAT RZ/N2 EoE 3port [2308 / 768]，点击烧录固件；

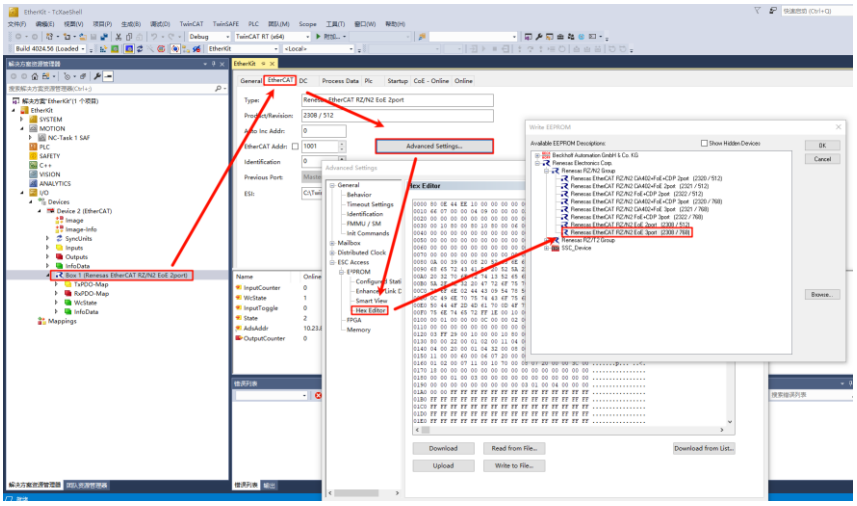


图 19-39 更新 ESI 固件

烧录完成后我们需要重新删除设备并再次扫描，可以看到从站设备描述已经更新为 Box 1 (Renesas EtherCAT RZ/N2 EoE 3port)；

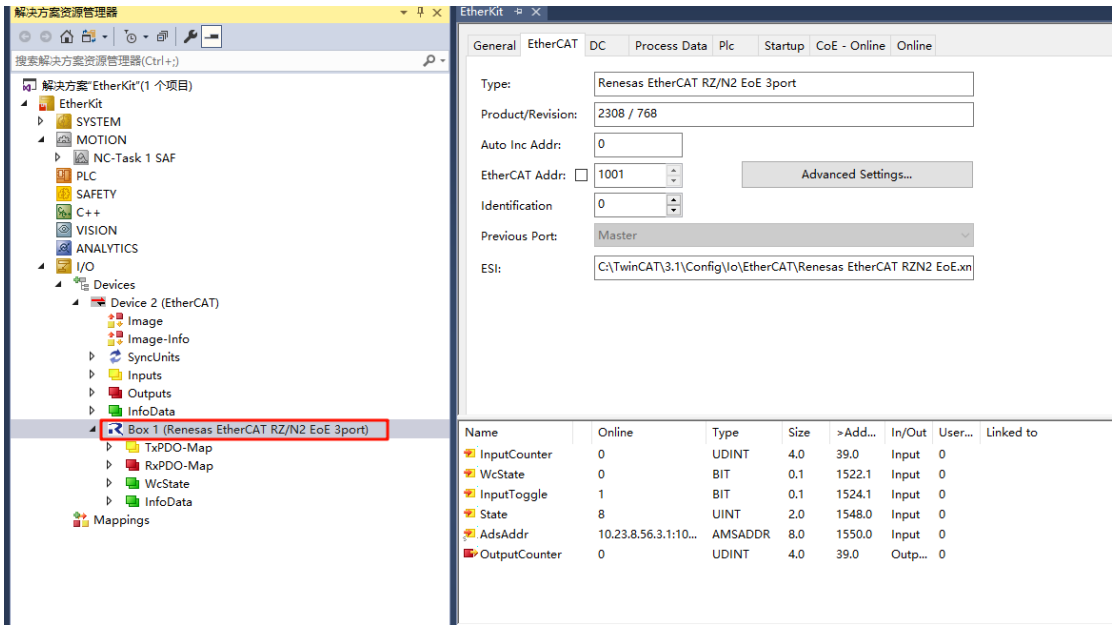


图 19-40 三端口 EOE

后续 EOE 开发请参考前几章节。

第 20 章 EtherCAT-COE 例程

20.1 简介

EtherCAT CoE（**CAN over EtherCAT**）是 EtherCAT 协议中的一种通信协议，它将 CANopen 应用层协议集成到 EtherCAT 网络中，用于分布式系统中的设备控制和数据交换。它结合了 CANopen 的易用性和 EtherCAT 的高性能优势，广泛用于工业自动化、运动控制和传感器网络等领域。

以下是 CoE 的主要特点和功能：

1. **基于 CANopen：**
    - CoE 的应用层直接采用了 CANopen 的设备协议，包括对象字典（Object Dictionary）的结构和服务。
    - 通过对象字典定义设备参数、通信对象和控制数据，确保了设备间的互操作性。
2. **支持标准服务：**
    - **SDO（Service Data Object）：**用于点对点的配置和诊断通信，允许主站与从站交换大容量数据（如参数配置）。
    - **PDO（Process Data Object）：**用于实时通信，传输小数据量的周期性过程数据，支持快速响应。
    - **Emergency（EMCY）消息：**用于报告设备异常情况。
    - **NMT（Network Management）：**提供网络管理功能，如启动、停止和复位设备。
3. **高效传输：**
    - EtherCAT 的总线结构和高速帧处理能力，使 CoE 能以更低的延迟和更高的效率进行数据交换。
4. **支持多种应用场景：**
    - 适用于工业设备配置、实时监控、参数诊断和系统集成等。
5. **对象字典映射：**
    - 对象字典以层级结构组织设备的数据和功能。
    - EtherCAT 使用 CoE 协议访问对象字典中的变量，以实现参数读

取、写入和实时控制。

6. **典型应用：**
   - 用于支持复杂控制逻辑的驱动器（如伺服驱动）。
   - 用于监控、调试和配置设备的工程工具。

本节将演示如何使用 Beckhoff TwinCAT3 和 EtherKit 开发板实现 EtherCAT COE 主从站通信，该示例工程已支持 CSP 及 CSV 两种操作模式。

20.2 前期准备

参考第 19 章：EtherCAT-EOE 例程，此处不再赘述。

20.3 TwinCAT3 配置

20.3.1 安装 ESI 文件

启动 TwinCAT 之前，将发布文件夹中包含的 ESI 文件复制到 TwinCAT 目标位置：“..\TwinCAT\3.x\Config\IO\EtherCAT”

注意：当前版本的 ESI 文件位于：..\board\ports\ESI_File\Renesas EtherCAT RZN2 CoE CDP.xml”

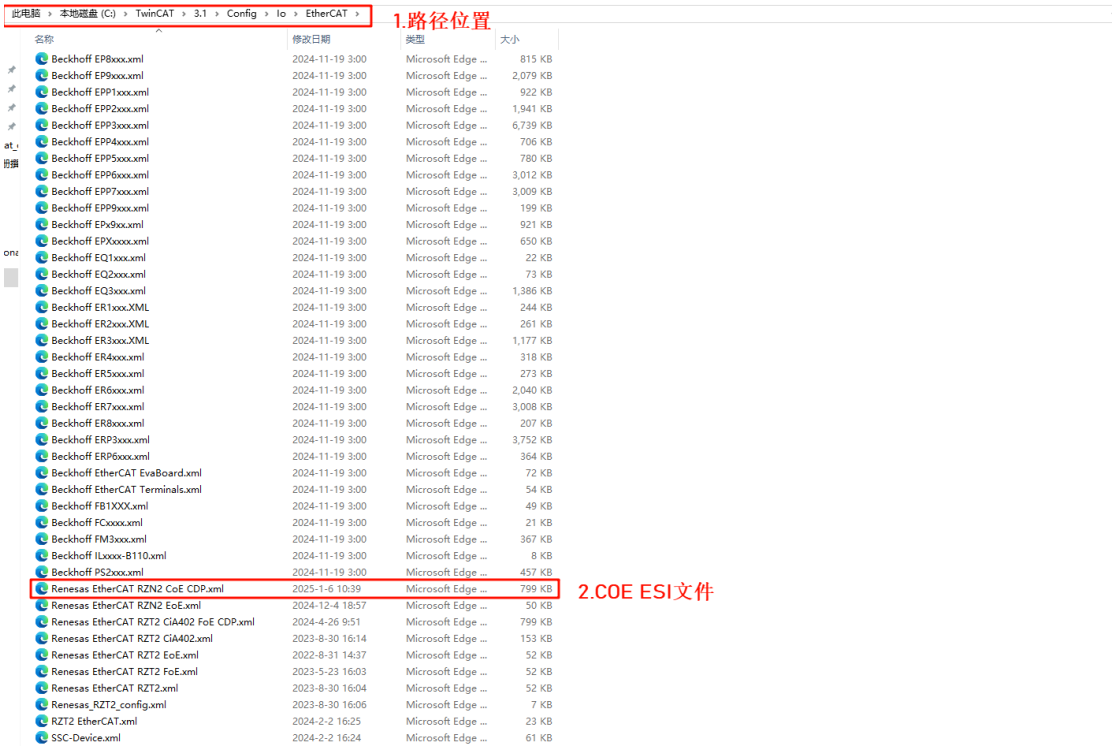


图 20-1 安装 ESI 文件

20.3.2 添加 TwinCAT 网卡驱动

参考 19.3.2 小节，此处不再说明。

20.4 FSP 及 Studio 配置

20.4.1 FSP 配置

接下来就是引脚初始化配置了，打开安装的 RZN-FSP 2.0.0，选择我们工程的根目录：

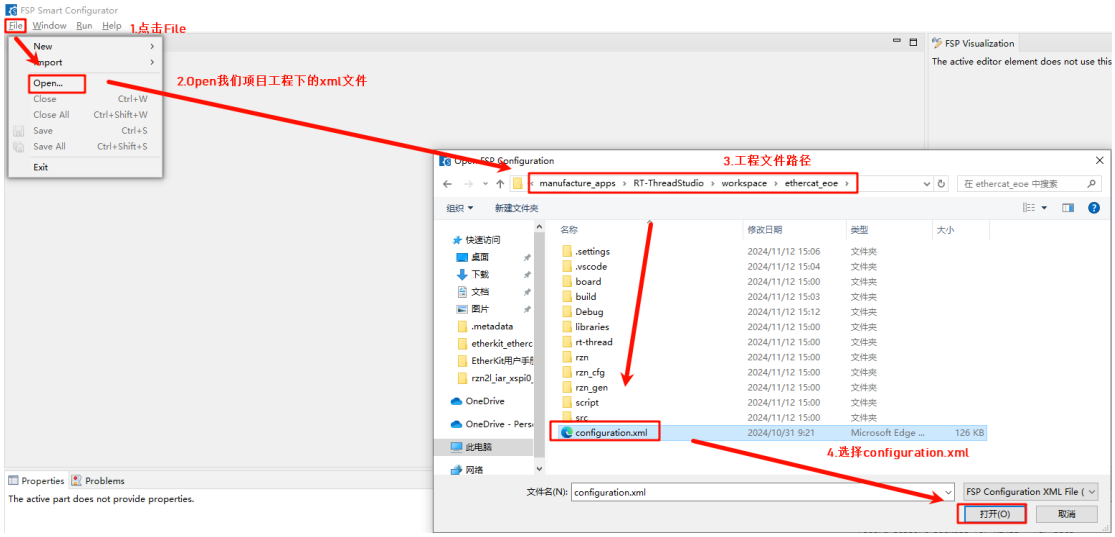


图 20-2 打开 fsp 配置

我们进行以下外设及引脚的配置：点击 New Stack，并添加 ethercat_ssc_port 外设：

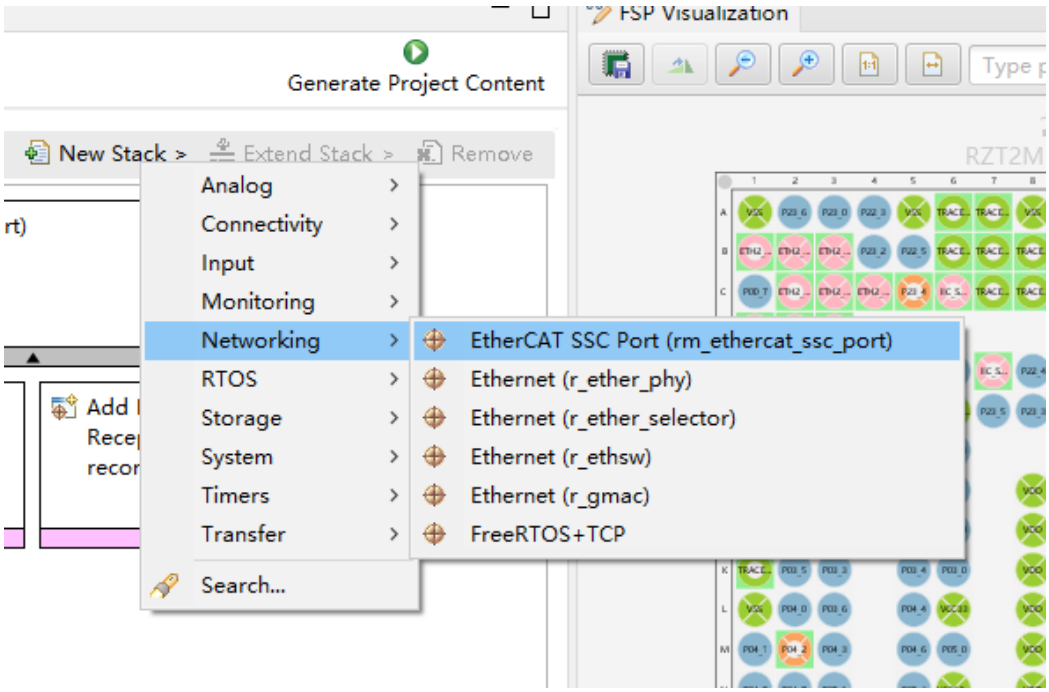


图 20-3 添加 ethercat_ssc_port 外设

配置 ethercat_ssc_port：修改 Reset Port 为 P13_4，同时 EEPROM_Size 大小设置为 Under 32Kbits；

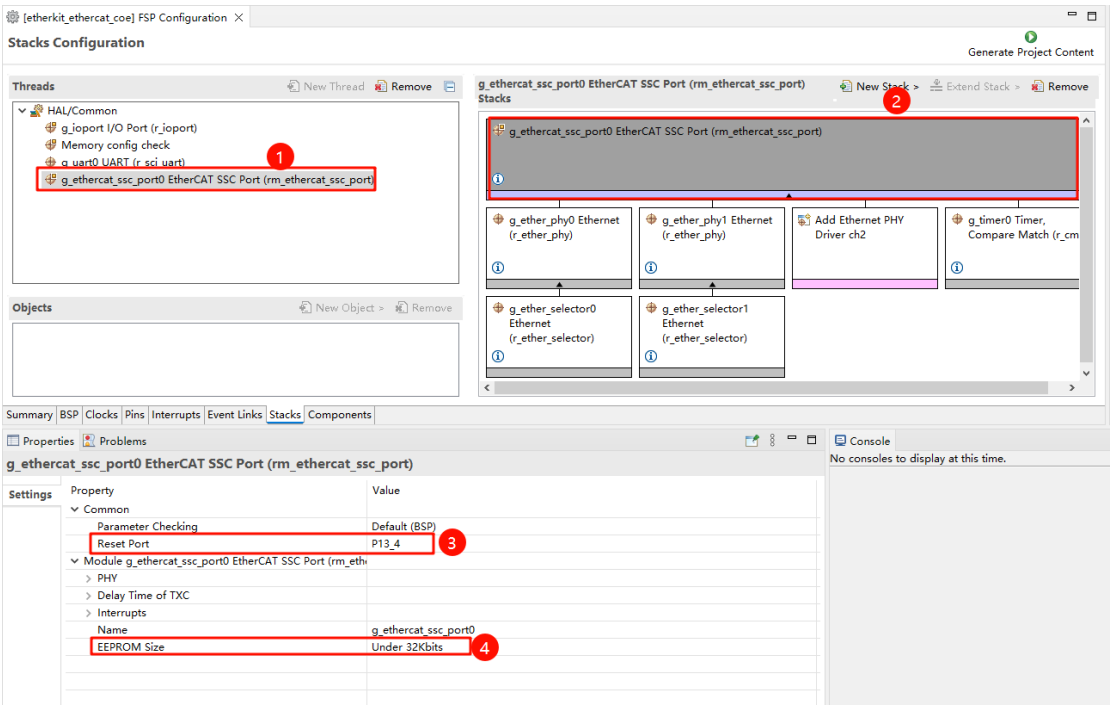


图 20-4 ethercat_ssc_port 配置

使能网卡类型、配置网卡设备参数，这里我们添加两个 phy（phy0 和 phy 1），其中需要注意的是，EtherKit 使用的是 rtl8211 网卡，并不在瑞萨 FSP 的支持范围内，但好在瑞萨预留了用户自定义网卡接口，因此按照如下设置来配置网卡，同时设置 MDIO 类型为 GMAC，设置网卡初始化回调函数 phy_rtl8211f_initial()（注意此处与 EOE 工程设置不同）；

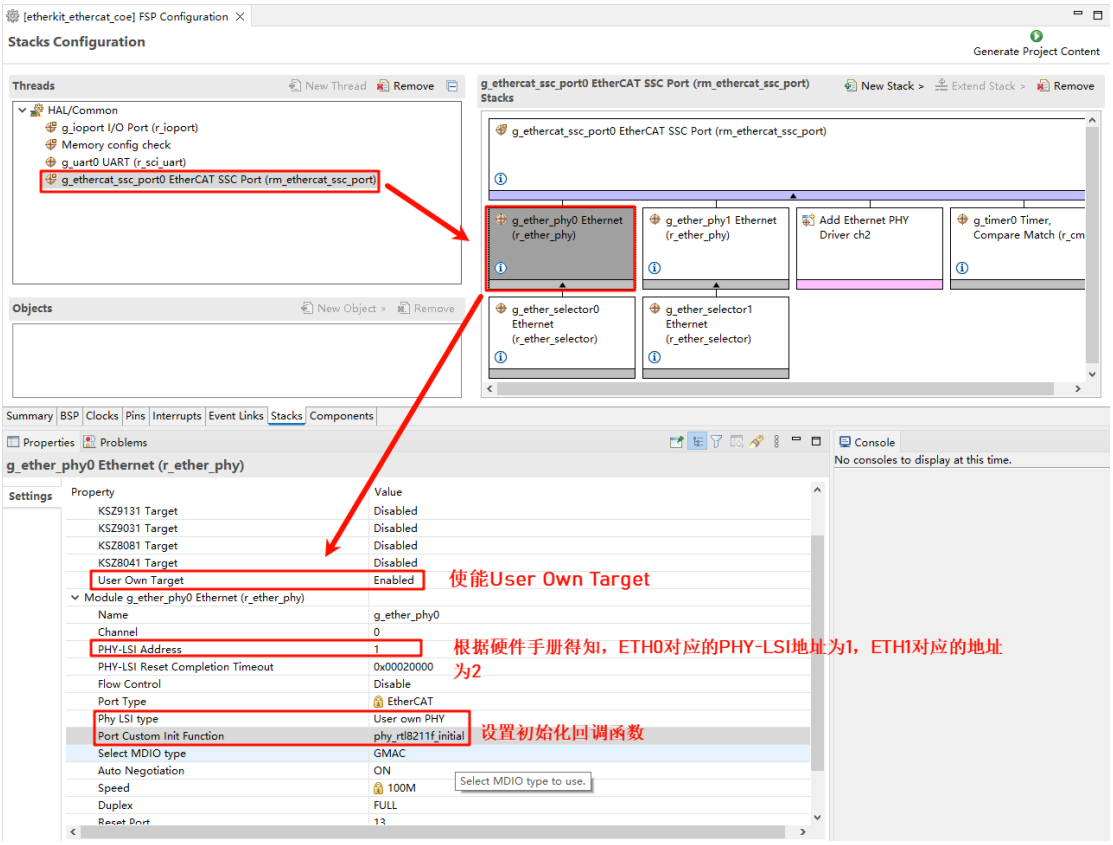


图 20-5 PHY 配置

网卡引脚参数配置，选择操作模式为 RGMII：

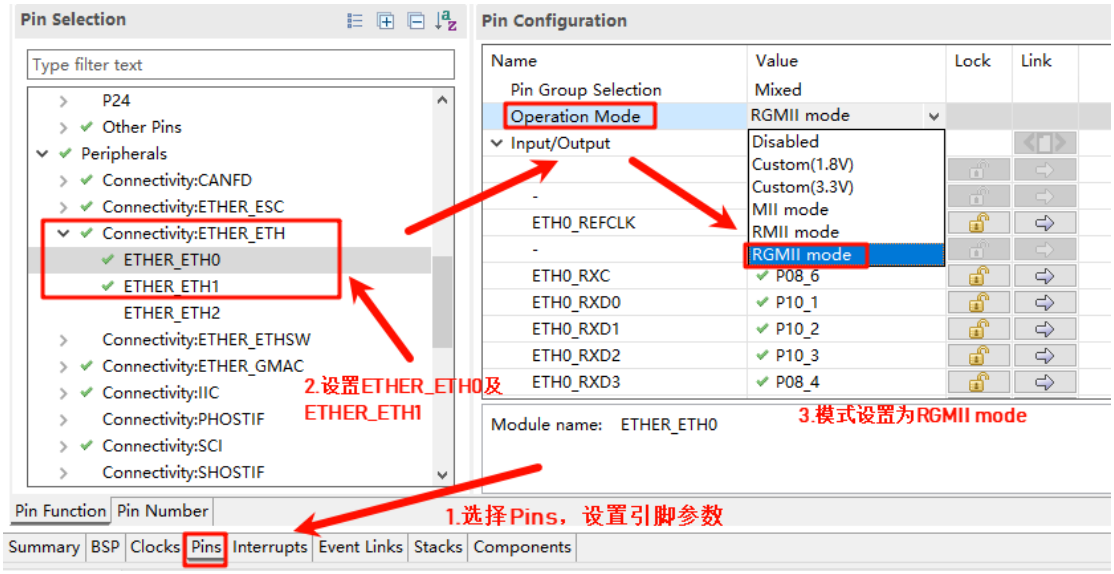


图 20-6 ETH 引脚配置

ETHER_ESC 设置：

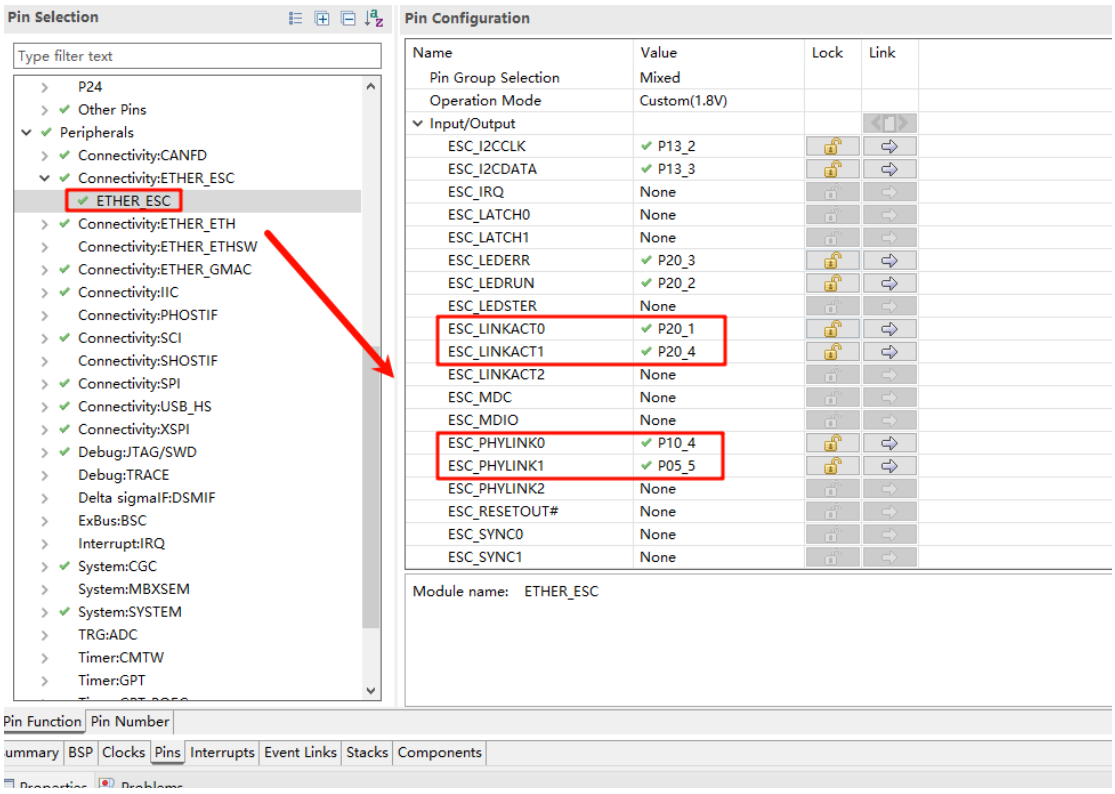


图 20-7 ESC 引脚配置

ETHER_GMAC 配置：

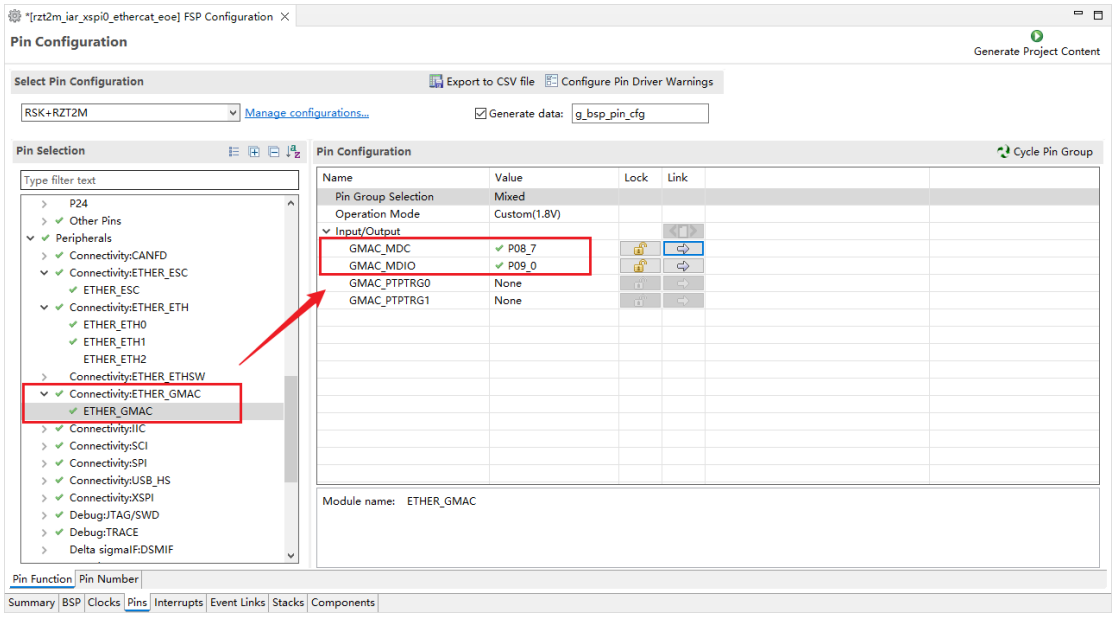


图 20-8 GMAC 引脚配置

为 ethercat_ssc_port 添加 cmt 定时器并配置中断优先级：

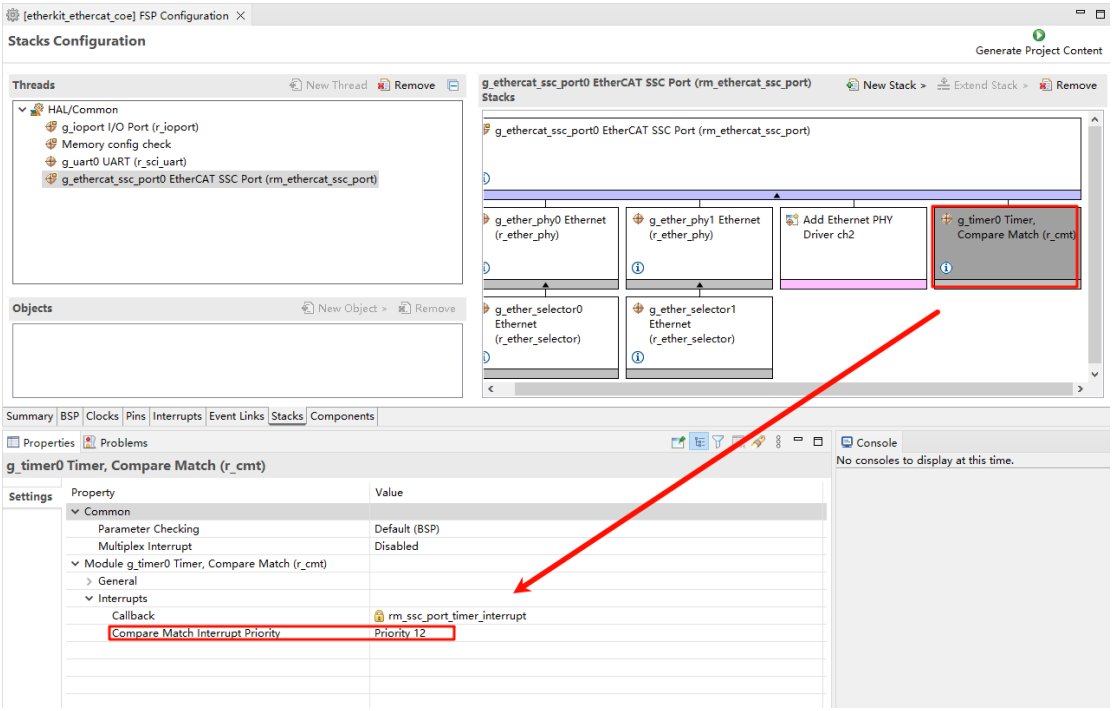


图 20-9 添加 CMT 定时器

最后点击 Generate Project Content 生成底层驱动源码。

20.4.2 构建配置

1.修改 sconscript：进入工程找到指定路径下的文件：.\rzn\SConscript，替换该文件为如下内容：

```
Import('RTT_ROOT')
Import('rtconfig')
from building import *
from gcc import *

cwd = GetCurrentDir()
src = []
group = []
CPPPATH = []

if rtconfig.PLATFORM in ['iccarm']:
    Return('group')
elif rtconfig.PLATFORM in GetGCCLikePLATFORM():
    if GetOption('target') != 'mdk5':
        src += Glob('./fsp/src/bsp/mcu/all/*.c')
        src += Glob('./fsp/src/bsp/mcu/all/cr/*.c')
```

```
        src += Glob('./fsp/src/bsp/mcu/r*/*.c')
        src += Glob('./fsp/src/bsp/cmsis/Device/RENESAS/Source/*.c')
        src += Glob('./fsp/src/bsp/cmsis/Device/RENESAS/Source/cr/*.c')
        src += Glob('./fsp/src/r_*/*.c')
        CPPPATH = [ cwd + '/arm/CMSIS_5/CMSIS/Core_R/Include',
                          cwd + '/fsp/inc',
                          cwd + '/fsp/inc/api',
                          cwd + '/fsp/inc/instances',]

if GetDepend('BSP_USING_COE_IO'):
    src += Glob('./fsp/src/rm_ethercat_ssc_port/*.c')
    CPPPATH += [cwd + '/fsp/src/rm_ethercat_ssc_port']

group = DefineGroup('rzn', src, depend = [''], CPPPATH = CPPPATH)
Return('group')
```

2.Kconfig 修改：打开工程下的文件（projects\etherkit_ethercat_coe\board\Kconfig），在 Onboard Peripheral Drivers 选项中加入 COE 配置：

```
        config BSP_USING_COE_IO
            bool "Enable EtherCAT COE_IO"
            default y
        config COE_DUMMY_LOG
            bool "Enable CoE dummy motor printf"
            default n
```

如下图所示：

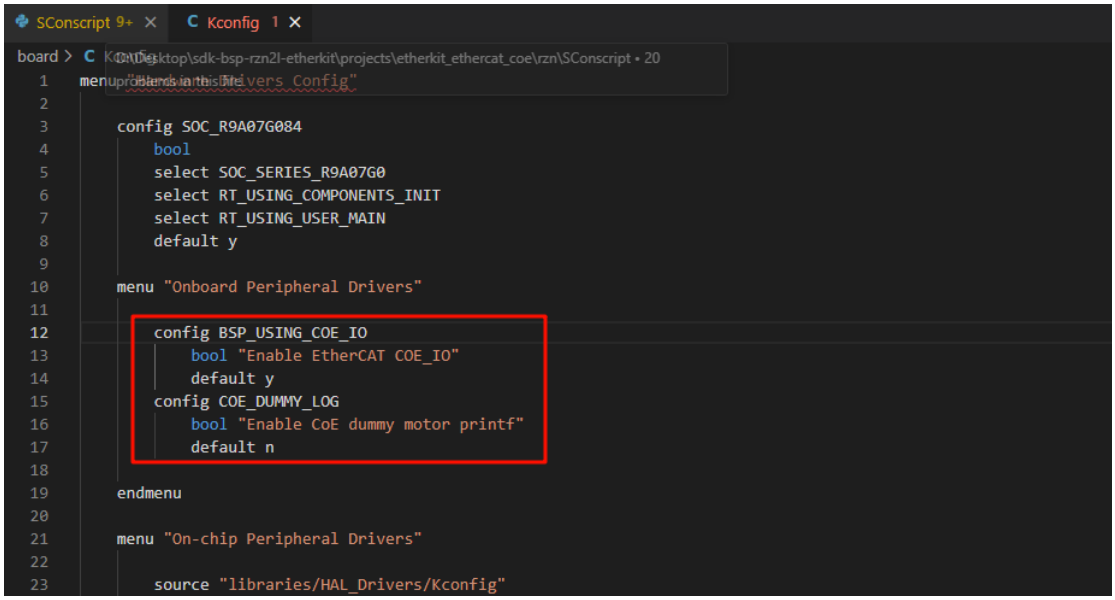


图 20-10 COE 配置

3.使用 studio 开发的话需要右键工程点击 **同步 scons 配置至项目**；如果是使用 IAR 开发请在当前工程下右键打开 env，执行：scons –target=iar 重新生成配置。

20.4.3 RT-Thread Studio 配置

完成 FSP 配置之后，引脚及外设的初始化就暂告一段落了，接下来需要我们使能 EtherCAT CoE 示例，打开 Studio，点击 RT-Thread Settings，使能 COE 示例，其中第二项可以开启 Canopen 状态打印：

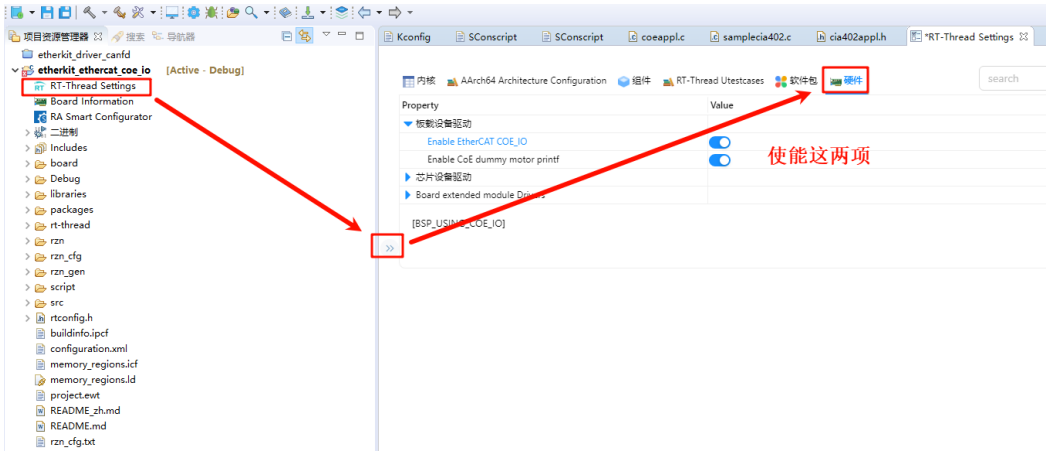


图 20-11 COE 配置

20.5 EtherCAT COE 配置

20.5.1 新建 TwinCAT 工程

打开 TwinCAT 软件，点击文件->新建->新建项目，选择 TwinCAT Projects，创建 TwinCAT XAR Project(XML format)工程：

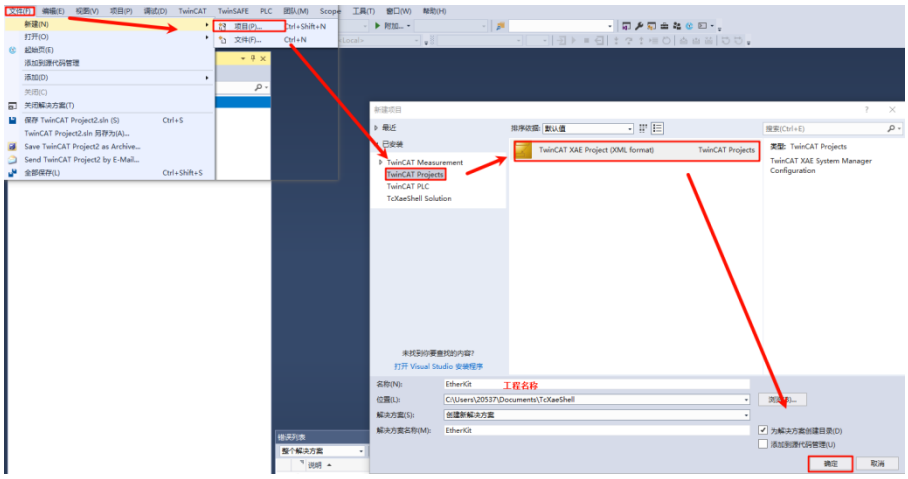


图 20-12 新建 TwinCAT 工程

20.5.2 从站启动 CoE App

将 EtherKit 开发板上电后，需要使用网线连接 ETH0/ETH1 网口，ethercat 会默认运行 CoE 程序。

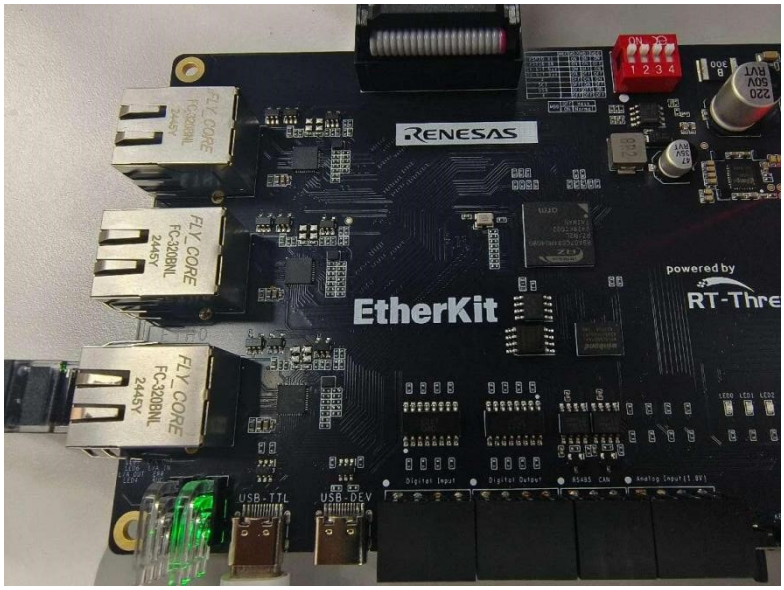


图 20-13 从站 COE 启动

20.5.3 从站设备扫描

新建工程之后，在左侧导航栏找到 Devices，右键选择扫描设备。正常来说如果扫描从站设备成功的话是会显示：Device x[EtherCAT]；而扫描失败则显示的是：Device x[EtherCAT Automation Protocol]，此时就代表从站初始化失败。

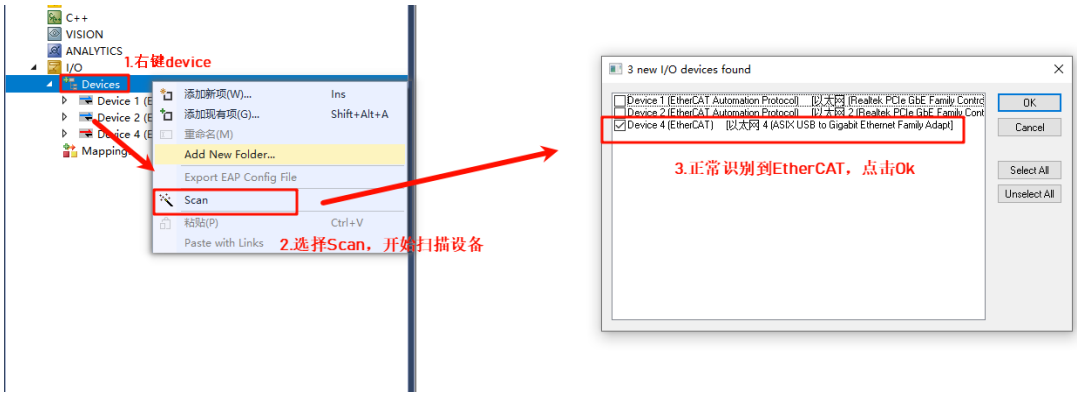


图 20-14 扫描设备

点击 Ok 后会弹出一个窗口：Scan for boxes，点击确认后，会再次弹出窗口：Activate Free Run，由于我们**首次使用 COE 还需要更新 EEPROM 固件**，所以暂时先不激活。

20.5.4 更新 EEPROM 固件

回到 TwinCAT，在左侧导航栏中，由于我们已经成功扫描到从站设备，因此可以看到主从站的配置界面：

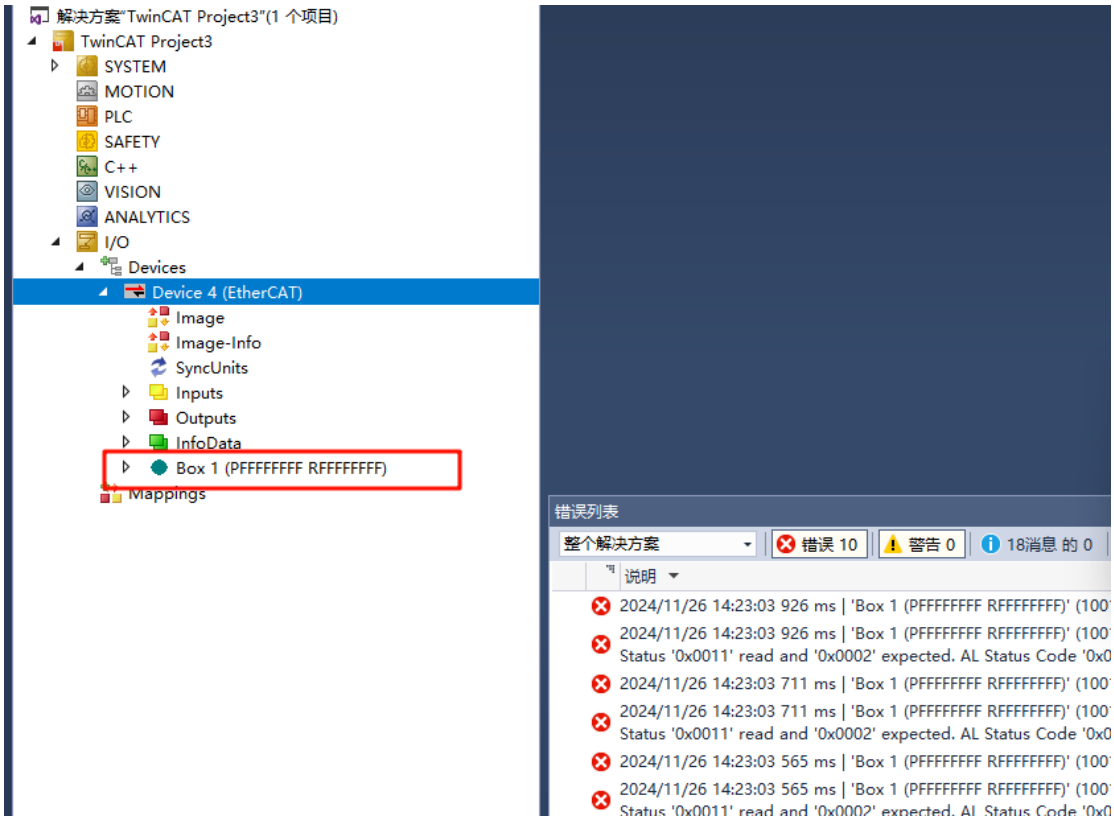


图 20-15 TwinCAT 配置界面

我们双击 Box 1，在中间界面的上方导航栏中单击 EtherCAT，并点击 Advanced Settings…：

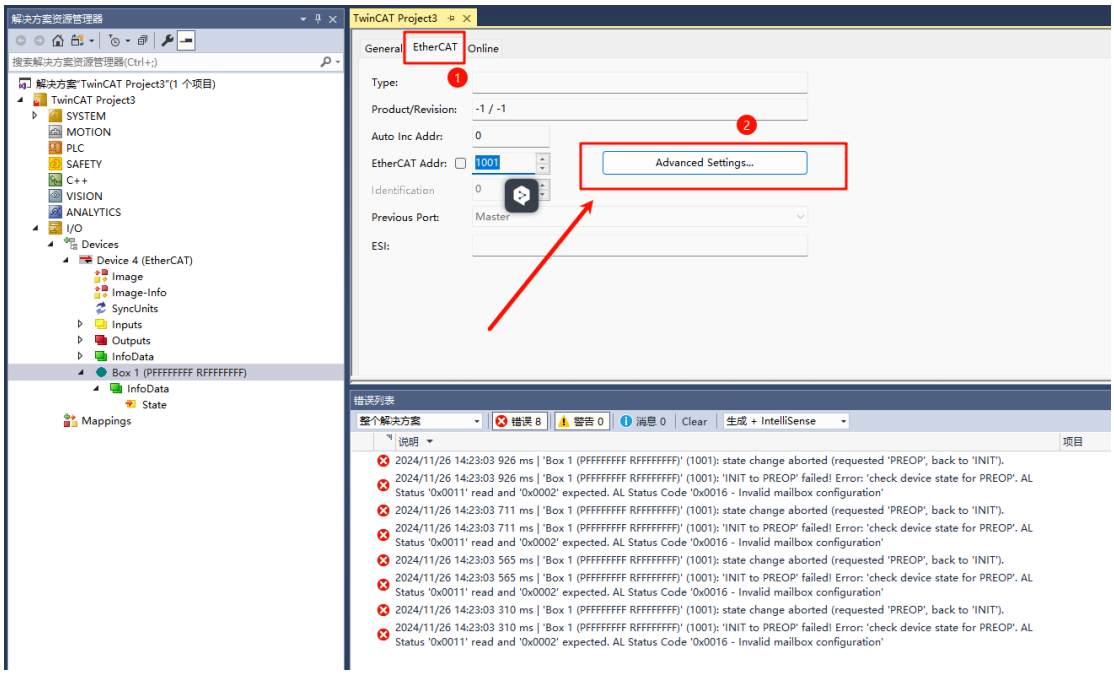


图 20-16 Advanced Settings

这里按图示点击 Download from List…：

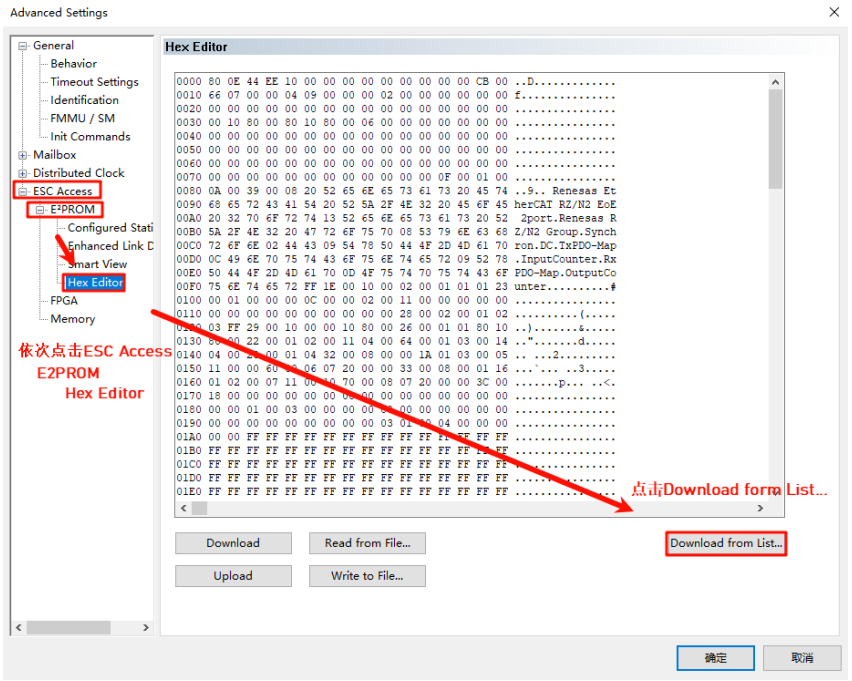


图 20-17 Hex Editor

我们写入 ESI 固件到 EEPROM 中，这里由于我们配置的是双网口，所以选择 Renesas EtherCAT RZ/N2 COE 2port，如果你配置的是三网口的话则选择 3 port 后缀的 ESI 文件进行下载。

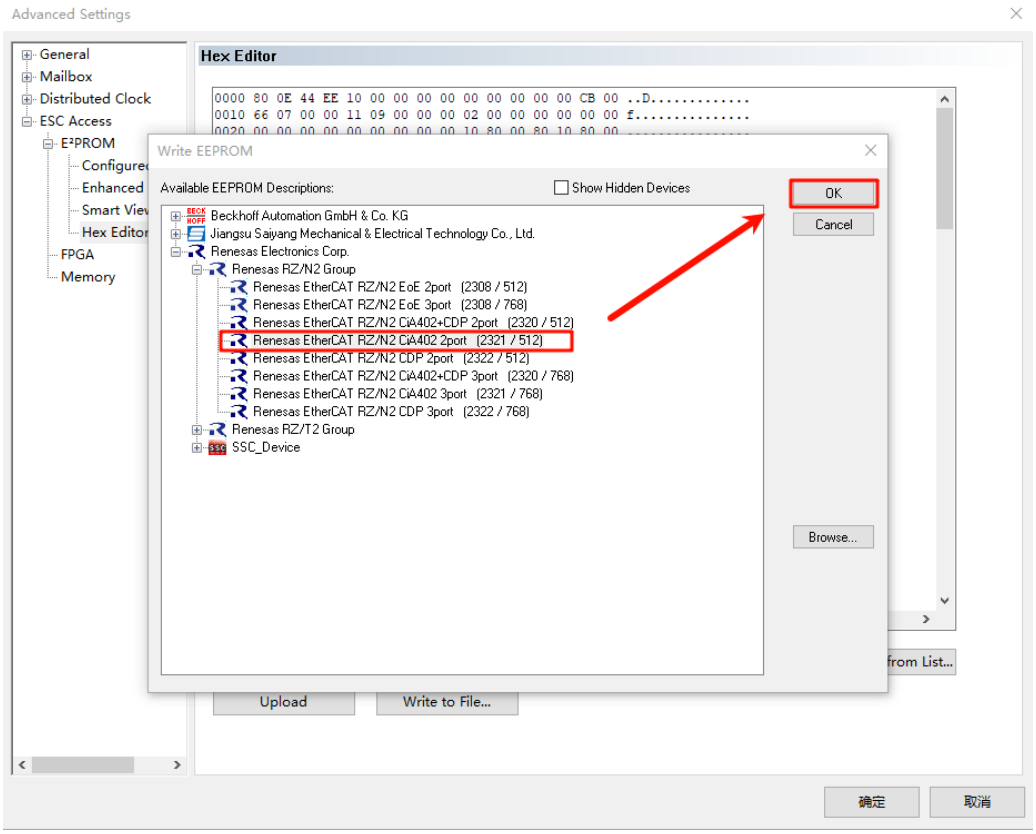


图 20-18 ESI 固件下载

下载完成之后，我们右键 Device x(EtherCAT)移除设备后重新扫描并添加设备，并完成激活工作（参考上文）。

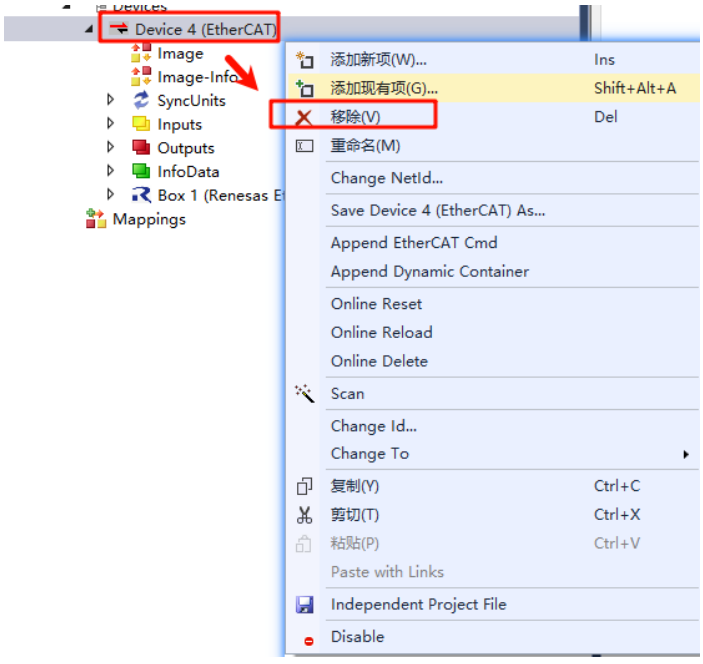


图 20-19 移除 Device

20.6 CiA402 伺服使用说明

首先来看下 CiA402 协议：CiA402 协议（Communication Interface for Drive Systems）是由 CiA (CAN in Automation) 组织定义的，用于工业自动化领域，特别是针对电机控制系统的标准化协议。CiA402 是驱动器和运动控制器 CANopen 设置子协议，定义变频器、伺服控制器以及步进的接口，它是国际标准 IEC 61800-7 系列的组成部分。CiA402 协议基于 CANopen 通信协议，并在此基础上扩展和优化了用于运动控制系统的功能。它主要用于伺服电机、步进电机以及其他类型的电动驱动系统的控制。

接下来看下 FSA（有限状态自动机）显示驱动器的不同状态以及如何执行它们之间的转换。

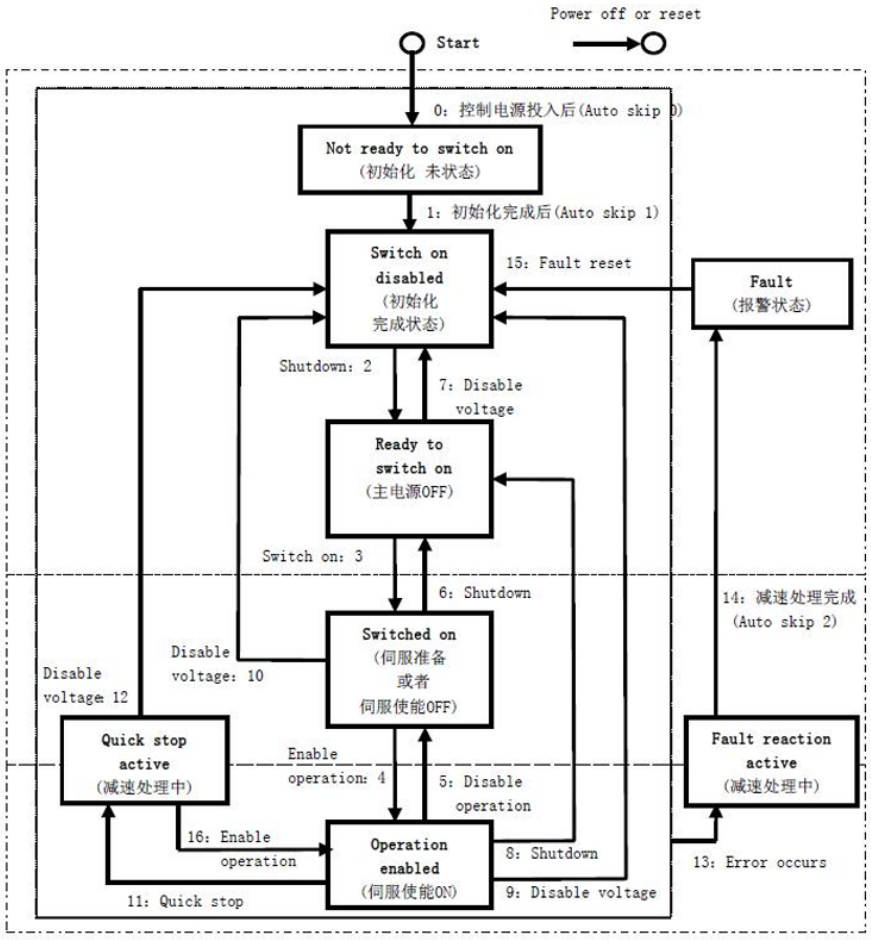


图 20-20 CiA402 状态机

下面是对应上图各个状态的详细说明：

表 20-1 CiA402 状态机详述

状态	说明
初始化	伺服初始化：伺服的参数不能设置，不能执行驱动指令功能
初始化完成	伺服初始化完成，可以设置伺服参数
伺服准备好	当前状态可以开启主电源，可以设置伺服参数，驱动器处于未激活状态
等待伺服使能	主电源 OK，可以设置伺服参数，等待伺服使能
伺服使能	伺服使能，按照设置的模式运行
快速停机	快速停机功能被激活，驱动器正在执行快速停机功能
故障停机	驱动器发生故障，正在执行故障停机过程中
报警状态	故障停机完成，所有驱动功能均被禁止，同时允许更改驱动器参数以便排除故障

对于控制器来说，在通信的每个周期内，都需要主站向从站发送控制字(control word)，并且接收从站的状态字进行确认，比如说本工程中通过 CiA402_StateMachine()实现 CiA402 的状态切换：

```
/*----------------------------------------------
-    CiA402 State machine
----------------------------------------------*/
#define STATE_NOT_READY_TO_SWITCH_ON       0x0001 /**< \brief Not ready
to switch on (optional)*/
#define STATE_SWITCH_ON_DISABLED           0x0002 /**< \brief Switch on
but disabled (optional)*/
```

```
#define STATE_READY_TO_SWITCH_ON          0x0004 /**< \brief Ready to
switch on (mandatory)*/
#define STATE_SWITCHED_ON                 0x0008 /**< \brief Switch on
(mandatory)*/
#define STATE_OPERATION_ENABLED           0x0010 /**< \brief Operation
enabled (mandatory)*/
#define STATE_QUICK_STOP_ACTIVE           0x0020 /**< \brief Quick stop
active (optional)*/
#define STATE_FAULT_REACTION_ACTIVE       0x0040 /**< \brief Fault
reaction active (mandatory)*/
#define STATE_FAULT                       0x0080 /**< \brief Fault state
(mandatory)*/
```

与此同时，主站通过读取从站的状态字(status word, 0x6041)来了解从站当前正在运行的状态，通过 status word 可以了解关于从机当前状态和可能发生的故障或警告的详细信息：

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
Reserved (Manufacture Specification)		Reserved (Operation Mode Specification)	Target Value Ignored	Internal Limit Active	Target Reached	Remote	Reserved (Maker Specification)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Warning	Switch On Disabled	Quick Stop	Voltage Enabled	Fault	Operation Enabled	Switched On	Ready to Switch on

图 20-21 CiA402 状态字帧格式

而主站通过控制字(control word, 0x6040)向从站发送控制命令，以此来改变其操作状态或触发指定的动作：

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
Manufacturer Specific (Manufacture Specification)					Reserved	Operation mode Specific	Halt

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Fault Reset	Operation mode Specific (Operation Mode Specification)			Enable Operation	Quick Stop	Enable Voltage	Switch On

图 20-22 CiA402 控制字帧格式

20.7 CiA402 对象字典定义

下面是有关 CiA402 对象字典在 EtherKit CoE 工程中支持的列表，其中已经支持了位置模式及速度模式，可通过主站去设置控制字来与从站的过程数据进行交互，基于 CoE 协议完成对控制器的读写：

Operation Mode	OBJECT Name	INDEX	Category	Access	Data Type	PDO Mapping
Cyclic synchronous position mode + Cyclic synchronous velocity mode	Position actual value	0x6064	Mandatory	ro	INT32	Yes
	Following error window	0x6065	Optional	rw	UINT32	No
	Following error time out	0x6066	Conditional	rw	UINT16	No
	Velocity actual value	0x606C	Conditional	ro	INT32	Yes
	Max torque	0x6072	Optional	rw	UINT16	Yes
	Torque actual value	0x6077	Conditional	ro	INT16	Yes
	Target position	0x607A	Optional	rw	INT32	Yes
	Position range limit	0x607B	Conditional	c,rw	INT32	Yes
	Software position limit	0x607D	Optional	c,rw	INT16	Yes
	Position offset	0x60B0	Optional	rw	INT32	Yes
	Velocity offset	0x60B1	Optional	rw	INT32	Yes
	Torque offset	0x60B2	Optional	rw	INT16	Yes
	Interpolation time period	0x60C2	Conditional	c,rw	UINT8	Yes
	Following error actual value	0x60F4	Optional	ro	INT32	Yes
	Target velocity	0x60FF	Conditional	rw	INT32	Yes

Function Group	OBJECT Name	INDEX	Category	Access	Data Type	PDO Mapping
Torque Limiting	Positive torque limit value	0x60E0	Conditional	rw	UINT16	Yes
	Negative torque limit value	0x60E1	Conditional	rw	UINT16	Yes
Homing	Home Offset	0x607C	Optional	rw	INT32	No
	Homing speeds	0x6099	Conditional	c,rw	UINT32	No
Touch Probe	Touch probe function	0x60B8	Optional	rw	UINT16	Yes
	Touch probe status	0x60B9	Optional	ro	UINT16	Yes
	Touch probe position 1 positive value	0x60BA	Optional	ro	INT32	Yes
	Touch probe position 1 negative value	0x60BB	Optional	ro	INT32	Yes
	Touch probe source	0x60D0	Conditional	c,rw	INT16	No
Gear ratio	Gear ratio	0x6091	Optional	c,rw	UINT32	No
Other object	**OBJECT Name**	**INDEX**	**Category**	**Access**	**Data Type**	**PDO Mapping**
Controlling the power drive system	Error code	0x603F	Optional	ro	UINT16	Yes
	Controlword	0x6040	Mandatory	rw	UINT16	Yes
	Statusword	0x6041	Mandatory	ro	UINT16	Yes
	Quick stop option code	0x605A	Optional	rw	INT16	No
	Shutdown option code	0x605B	Optional	rw	INT16	No
	Disable operation option code	0x605C	Optional	rw	INT16	No
	Halt option code	0x605D	Optional	rw	INT16	No
	Fault reaction option code	0x605E	Optional	rw	INT16	No
	Modes of operation	0x6060	Optional	rw	INT8	Yes
	Modes of operation display	0x6061	Optional	ro	INT8	Yes
	Supported drive modes	0x6502	Mandatory	ro	INT32	No
General object	Motor type	0x6402	Optional	rw	INT16	No
Position control function	Position demand value	0x6062	Optional	ro	INT32	No
	Position actual internal value	0x6063	Optional	ro	INT32	No
	Position window	0x6067	Optional	rw	UINT32	No
Optional application FE	Digital inputs	0x60FD	Optional	ro	UINT32	Yes
	Digital outputs	0x60FE	Optional	c,rw	UINT16	No,Yes

图 20-23 支持的对象字典列表

20.8 EtherCAT COE 测试

首先我们需要确保程序已经正常下载至工程中，同时 ESI 文件已经成功烧录，下面是开发板串口终端打印信息：

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Jan  6 2025 17:25:06
 2006 - 2024 Copyright by RT-Thread team
=================================================
EtherCAT Slave with CoE Project!
=================================================

Hello RT-Thread!
=================================================
This example project is an ethercat CoE_IO routine!
=================================================
msh >
RT-Thread shell commands:
clear            - clear the terminal screen
version          - show RT-Thread version information
list             - list objects
reboot           - Reboot System
backtrace        - print backtrace of a thread
help             - RT-Thread shell help
ps               - List threads in the system
free             - Show the memory usage in the system
pin              - pin [option]

msh >ps
thread           pri  status      sp     stack size max used left tick   error  tcb addr
---------------- ---  ------- ---------- ----------  ------  ---------- ------- ----------
tshell            20  running 0x00000210 0x00001000     13%   0x00000002 OK      0x1001b368
ethercat_thread   16  suspend 0x0000008c 0x00001000     05%   0x0000000a EINTRPT 0x100164c4
sys workq         23  suspend 0x00000070 0x00000800     05%   0x0000000a OK      0x1001a870
tidle0            31  ready   0x00000048 0x00000400     11%   0x00000004 OK      0x10014d70
```

图 20-24 CoE 日志

同时我们打开前面新建的 ESC 工程，并且扫描设备，此时会弹出 EtherCAT drive(s) added，我们选择 NC – configuration，点击 OK 后并激活设备：

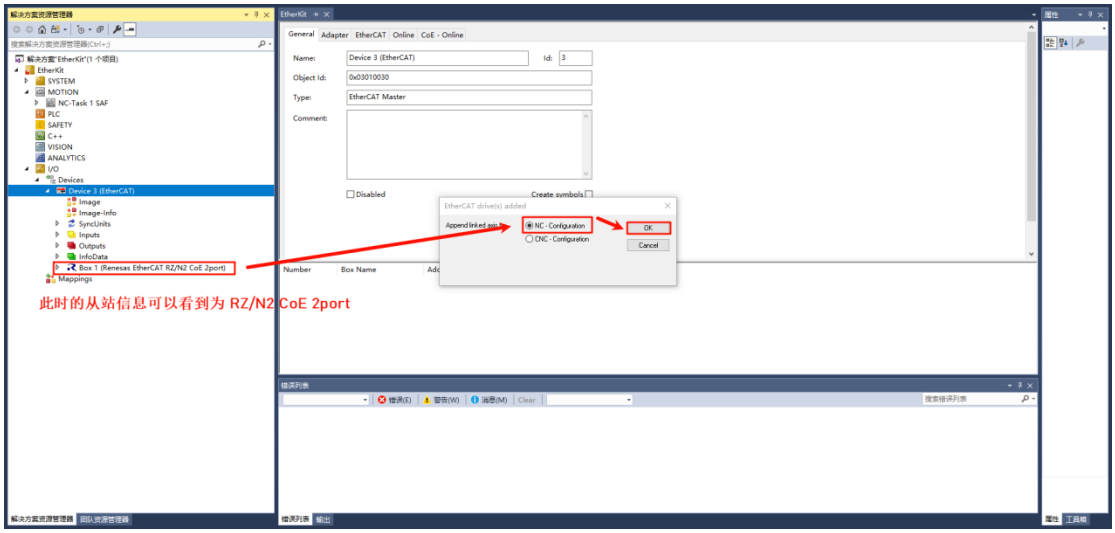


图 20-25 NC configuration 选中

成功激活后，EtherCAT 状态机会依次经历 Init->Pre-Op->Safe-Op，最后到 Op(Operational，可操作状态)，EtherKit CoE 工程默认开启 csp（周期同步位置模式），并且支持 csv（同步周期速度模式）。

当系统上电后，驱动器自动完成初始化，然后进入 STATE_SWITCH_ON_DISABLED 状态，此时可以对驱动器的工作状态模式进行设置，比如说设置为 csp 或 csv 模式等等；同时对应在开发板端能看到当前轴 1 对应的 CiA402 状态机信息在不断打印：

```
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
```

图 20-26 CoE 串口终端

20.8.1 csp 位置模式控制

先来看下 csp 模式下的控制器：在位置模式下，我们可以将规划好的目标位置通过写入控制字 0x607A 设置目标位置，并且对应状态字 0x6064 可以得到实际反馈的位置信息。

而如果想要在 csp 或 csv 模式下操作，必须先将其状态修改为 STATE_OPE

RATION_ENABLED（可操作模式）。

展开左侧导航栏，依次点击 Box 1(Renesas EtherCAT RZ/N2 CoE 2port)->Module 1(csp - axis)->Outputs->Control Word，首先需要将状态切换为伺服无故障模式，主站通过向控制字 0x6040 写入值 0x0080(dec:128)，将伺服控制器转变为无障碍状态：

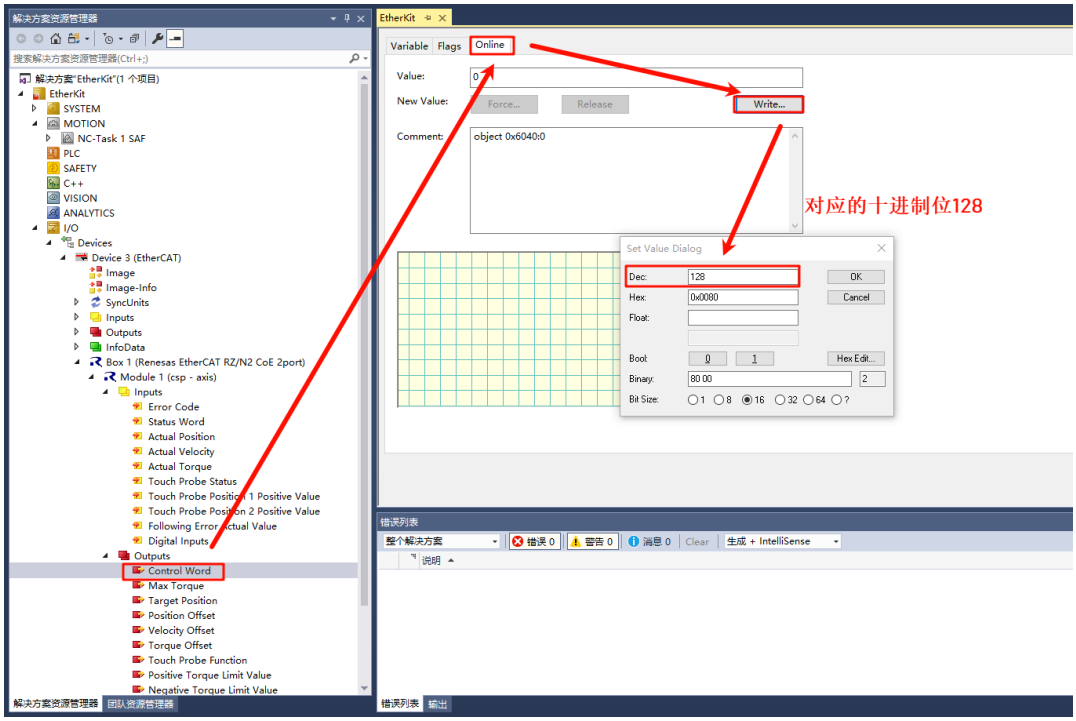


图 20-27 控制字写入 0x008

此时可以看到从站串口终端会停止 State Transition2、State Transition7 的打印，接着我们再次向控制字 0x6040 写入值 0x000F(dec:15)：

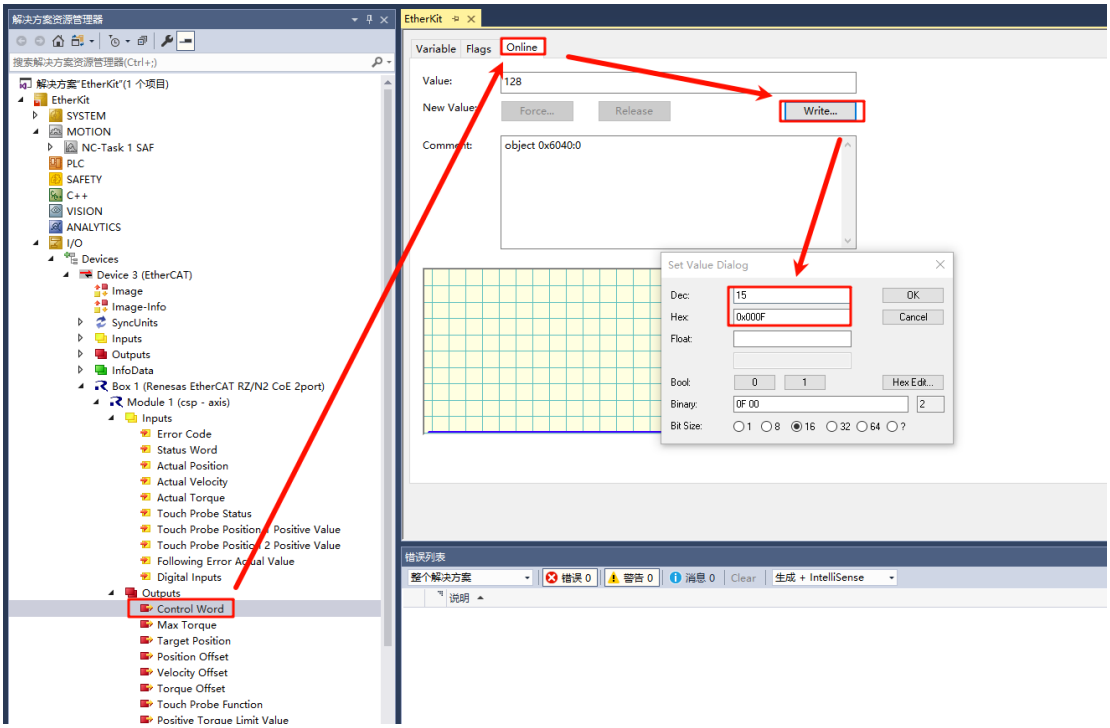


图 20-28 控制字写入 0x00F

此时伺服控制器由等待打开伺服使能切换到伺服运行的状态，同时在从站串口中断打印 StateTransition2、State Transition3、State Transition4，在经过状态传输 2 3 4 后，CiA402 状态机进入 STATE_OPERATION_ENABLED，此时就可以对控制器进行控制了。

比如说当前是位置模式，通过向 Index:0x607A 写入位置数值，我们写入 10 0000：

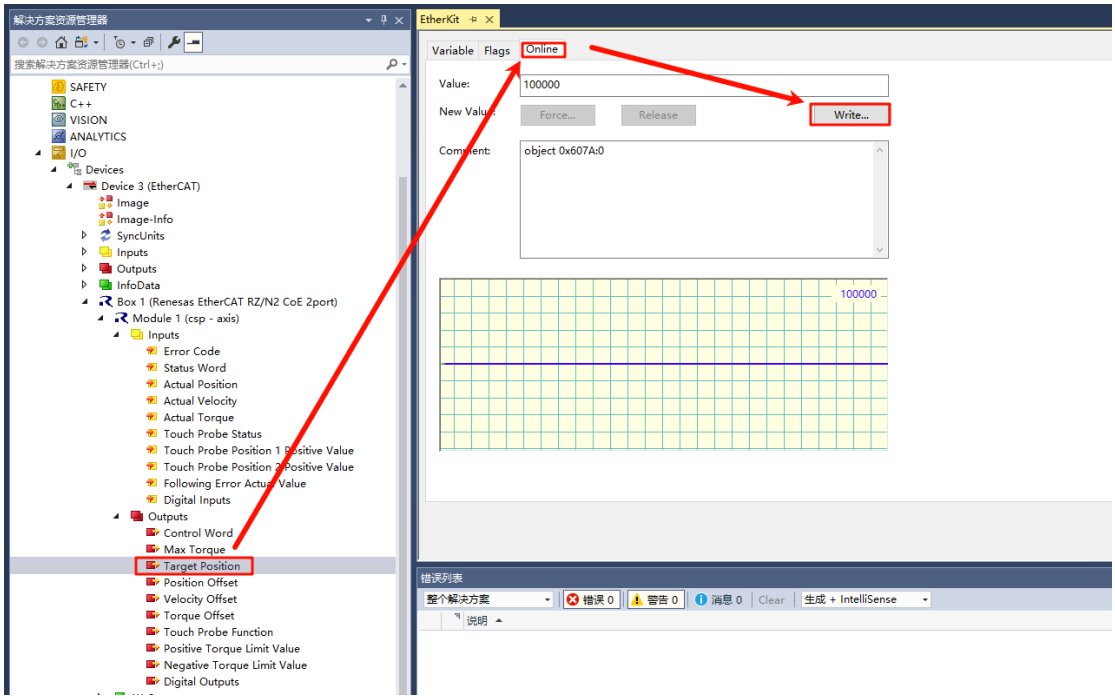


图 20-29 写入目标位置

此时依次点击 Box 1(Renesas EtherCAT RZ/N2 CoE 2port)->Module 1(csp - axis)->Inputs->Actual Position，查看实际反馈的位置，会发现 Index 0x6064 对应的 value 会不断自增，直到 100000 停止：

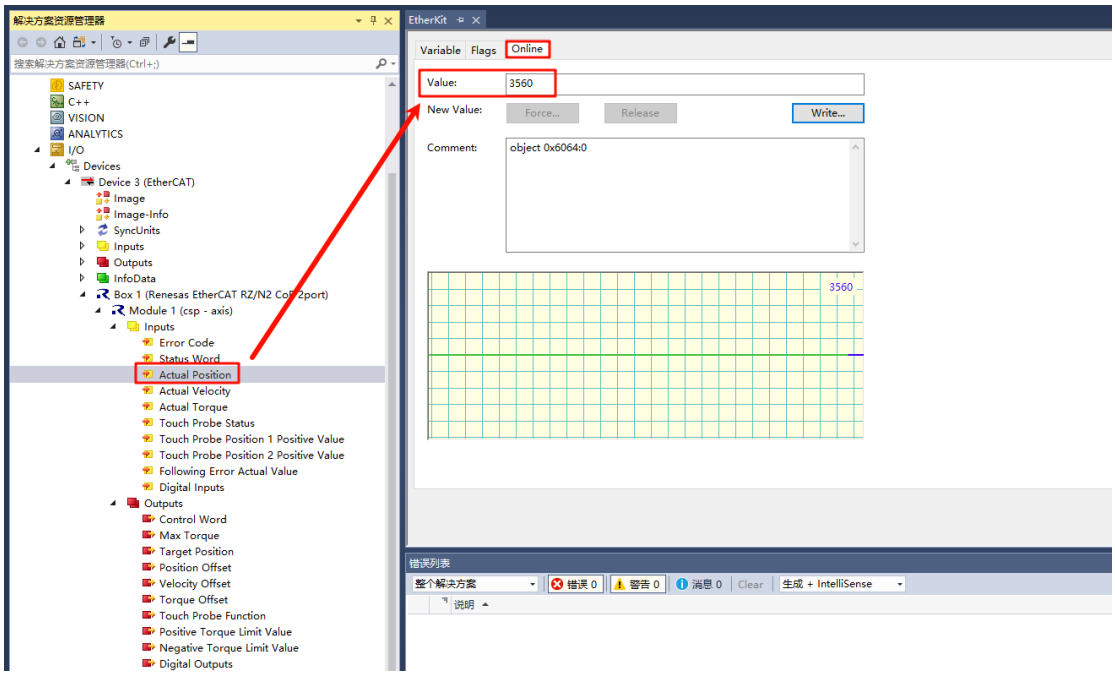


图 20-30 实际反馈位置

20.8.2 csv 速度模式控制

首先需要将控制器模式由默认的 csp 切换为 csv 模式，点击左侧导航栏中的 Box 1(Renesas EtherCAT RZ/N2 CoE 2port)，接着在中间的页面中找到上方的 Slots 选择 Axis 0，在右边预设支持的 module 修改为 csv，并点击‘<’标志：

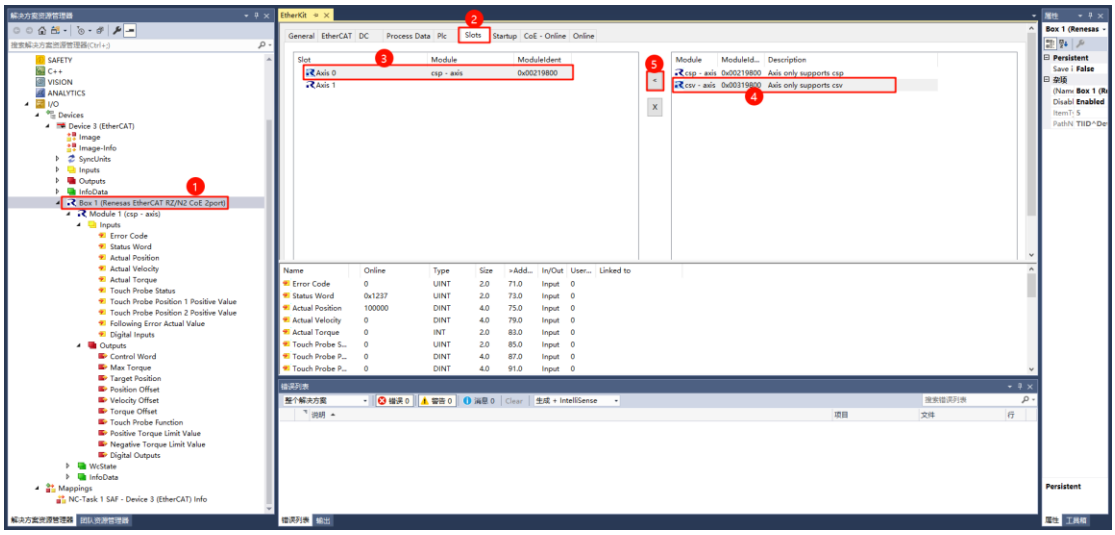


图 20-31 控制器模式切换

同时我们也可以观察左侧对应的模块信息是否更新，并切换为 csv 模式：

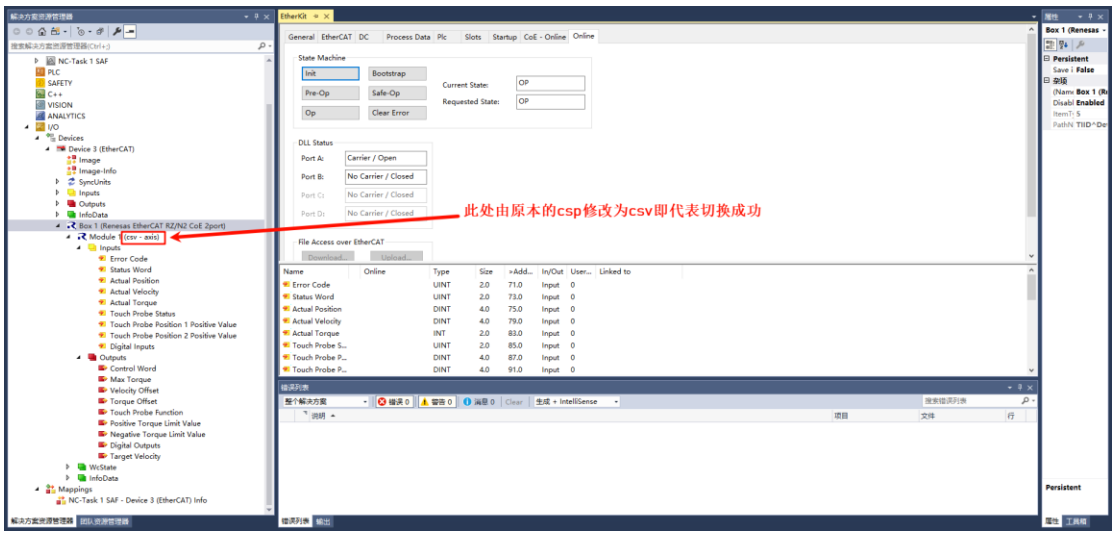


图 20-32 csv 模式

切换好模式后，我们需要重新加载设备，点击 TwinCAT3 上方导航栏的 TwinCAT->Reload Devices：

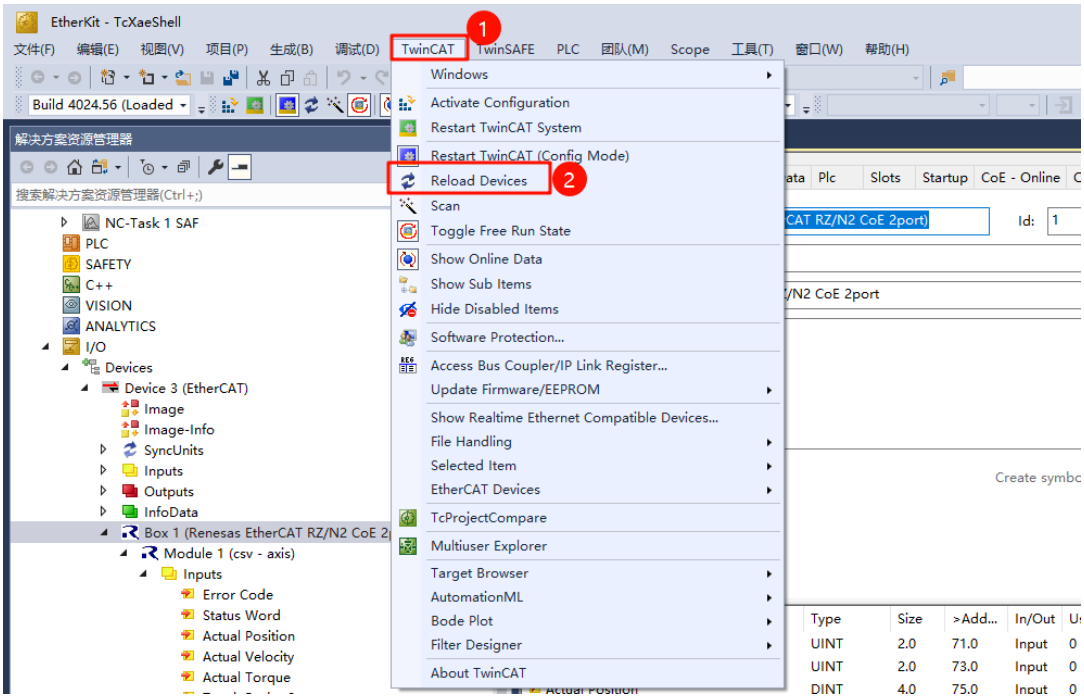


图 20-33 重新加载设备

然后需要使控制器进入 STATE_OPERATION_ENABLED（可操作模式，参考 20.8.1 章），同样是对控制字依次写入 0x0080（转变为无障碍状态）、0x000F（由等待打开伺服使能切换到伺服运行状态）。

此时我们查看输入的状态字 0x6041，如果对应的 value 值为 0x1237，那么就代表当前处于可操作模式(STATE_OPERATION_ENABLED)；如果显示的值为 0x1208，那么代表当前 status 处于 Fault，重新设置 control word 为 0x0080（dec:128），并且在重复上述操作即可。

此时我们便可对 Target Velocity 值进行写入实际想要控制的速度值：

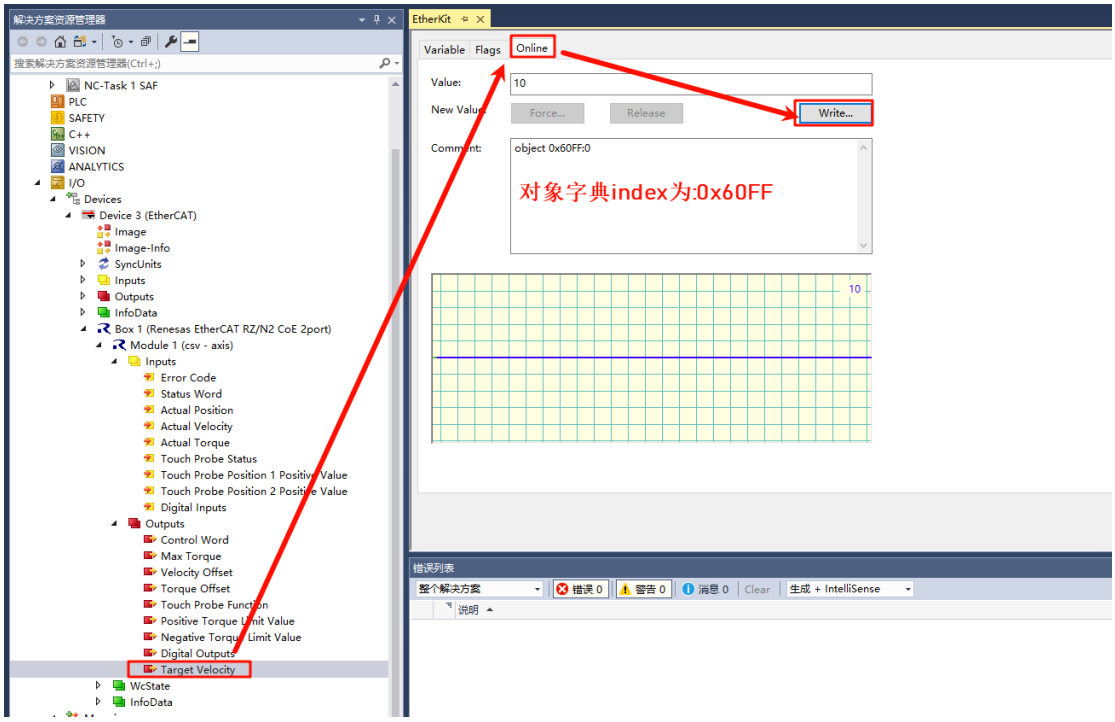


图 20-34 设置目标速度

同时可在输入中查看实际设置的速度信息是否一致：

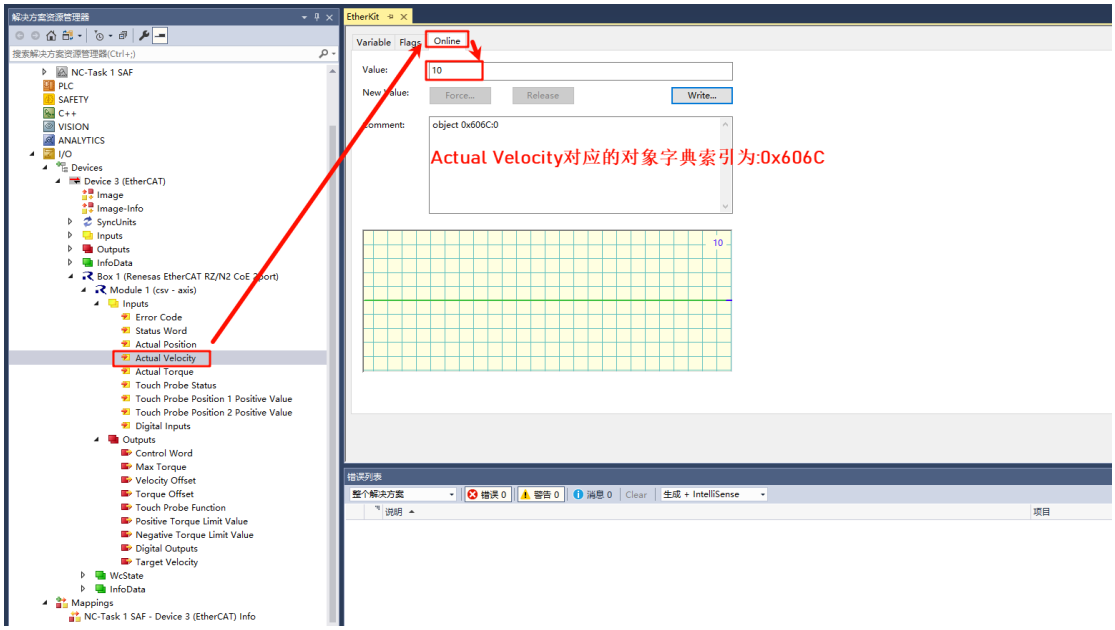


图 20-35 实际速度查看

第 21 章 PROFIENT 例程

21.1 简介

PROFINET 是由 PI（PROFIBUS 和 PROFINET International）组织开发和推广的工业以太网标准，广泛应用于工业自动化领域。

P-Net 协议是一个开源的 PROFINET 实现，专门用于嵌入式设备的实时网络通信。它是一个开源项目（p-net），目标是提供一个轻量级的 PROFINET 协议栈实现，使得开发者能够在嵌入式平台上快速集成 PROFINET 功能。

在本示例中将使用 P-Net 软件包来实现 PROFINET 主从站通信。

21.2 前期准备

软件环境：

- CODESYS（profinet 主站模拟）
    - CODESYS
    - CODESYS Gateway（网关设备）
    - CODESYS Control Win SysTray（软 PLC 设备）
- Npcap（该软件是运行 CODESYS 必须的，需要提前安装好！）

硬件环境：

- EtherKit 开发板

21.3 FSP 配置

此处配置请参考第 11 章：11.3.1 FSP 配置。

21.4 RT-Thread Settings 配置

双击打开 RT-Thread Settings，在搜索栏检索 p-net 软件包并使能，下面是相关用户配置信息说明；

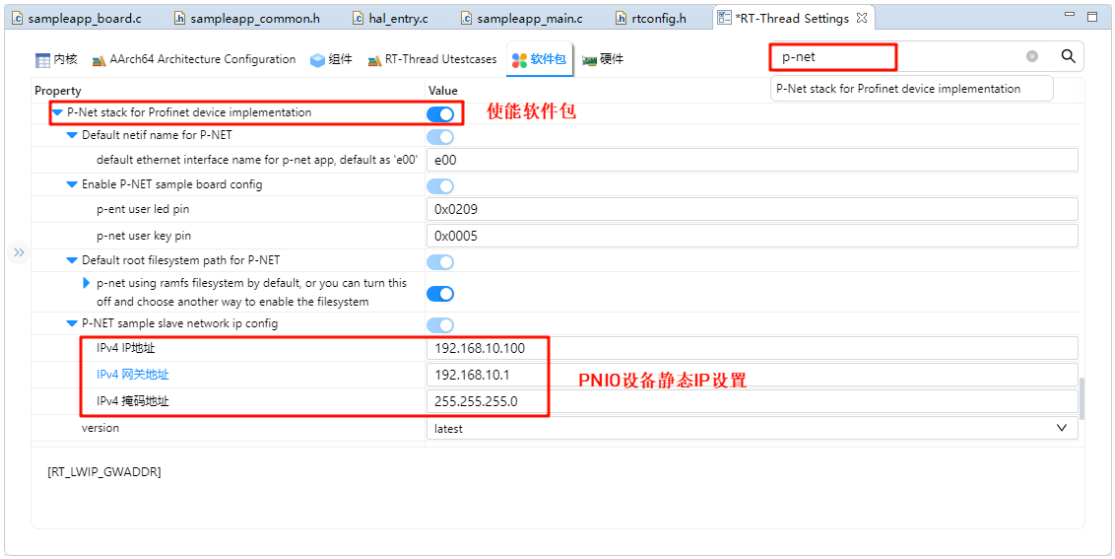


图 21-1 使能 p-net 软件包

- **Default netif name for p-net**：p-net 网卡设备接口名称，默认为 e00；
- **Enable pnet sample board config**：p-net app 用户 LED 及按键配置；
- **Default root filesystem path for p-net**：p-net 文件系统配置，默认使用 ramfs ，默认分配 8K 内存空间；
- **P-NET sample slave network ip config**：p-net 从站设备静态 IP 配置（**请关闭 RT_LWIP_DHCP 功能，使用静态 IP**）

下面我们还需要配置禁用 dhcp 功能并使用静态 IP，点击组件->使能 lwip 堆栈，选择禁用 DHCP；

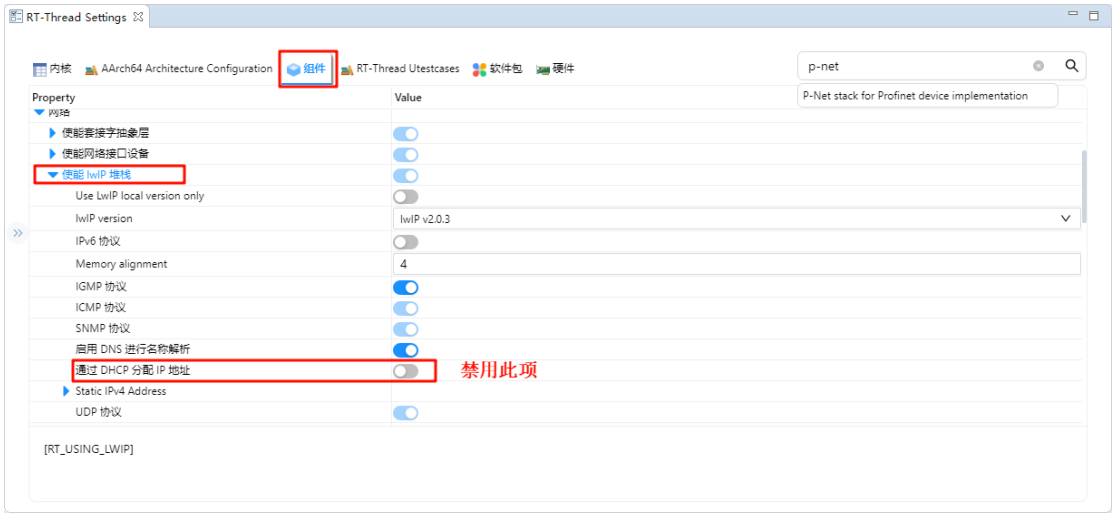


图 21-2 lwip 设置

完成上述配置后，将程序编译下载至开发板。

21.5 网络配置

我们使用一根网线连接开发板与 PC，同时在 PC 端配置静态 IP：

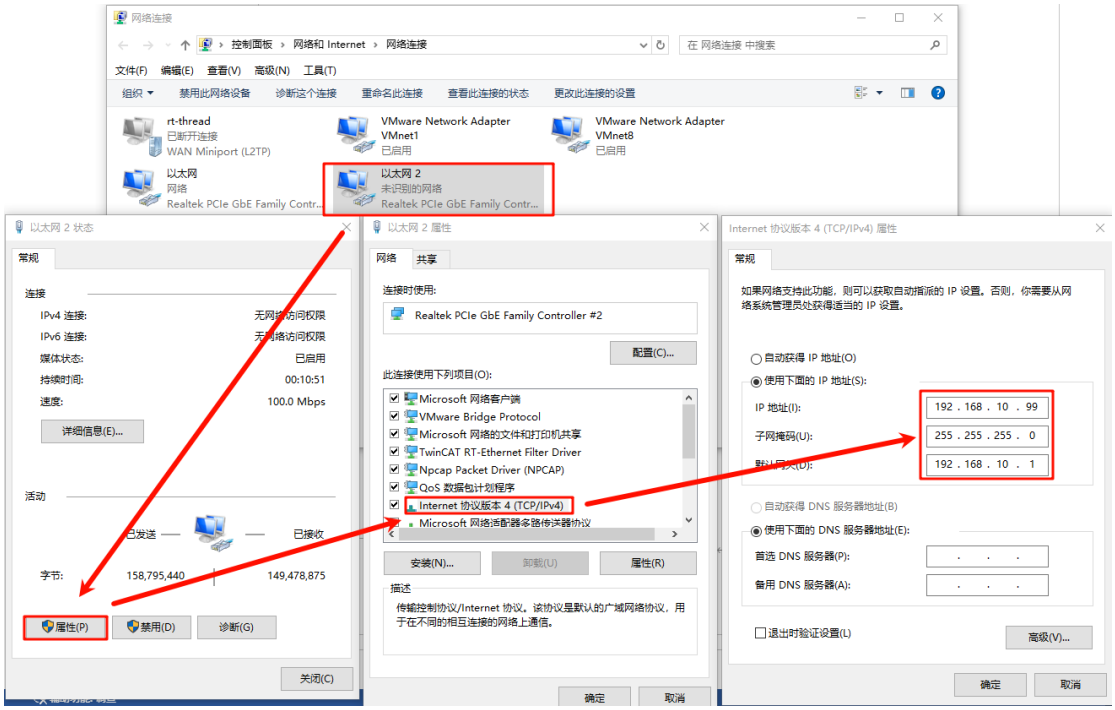


图 21-3 以太网静态 IP 配置

检查开发板端的 IP 信息，并测试联通性：

```
Plug DAP module and its submodules
Module plug indication
  Pull old module.    API: 0 Slot:  0
  Plug module.        API: 0 Slot:  0 Module ID: 0x1 "DAP 1"
Submodule plug indication.
  Pull old submodule. API: 0 Slot:  0 Subslot: 1
  Plug submodule.     API: 0 Slot:  0 Module ID: 0x1
                      Subslot: 1 Submodule ID: 0x1 "DAP Identity 1"
                      Data Dir: NO_IO  In: 0 bytes  Out: 0 bytes
Submodule plug indication.
  Pull old submodule. API: 0 Slot:  0 Subslot: 32768
  Plug submodule.     API: 0 Slot:  0 Module ID: 0x1
                      Subslot: 32768 Submodule ID: 0x8000 "DAP Interface 1"
                      Data Dir: NO_IO  In: 0 bytes  Out: 0 bytes
Submodule plug indication.
  Pull old submodule. API: 0 Slot:  0 Subslot: 32769
  Plug submodule.     API: 0 Slot:  0 Module ID: 0x1
                      Subslot: 32769 Submodule ID: 0x8001 "DAP Port 1"
                      Data Dir: NO_IO  In: 0 bytes  Out: 0 bytes
Done plugging DAP

Waiting for PLC connect request

PLC connect indication. AREP: 1
Event indication PNET_EVENT_STARTUP   AREP: 1
PLC dcontrol message (The PLC is done with parameter writing). AREP: 1  Command: PRM_END
Event indication PNET_EVENT_PRMEND    AREP: 1
  Set initial input data and IData status indication. AREP: 1  Data status changes: 0x35  Data status: 0x35
   Run, Valid, Primary, Normal operation, Evaluate data status
 operation, Evaluate data status
a status: 0x35
 "DAP Identity 1"
  Set initial input data and IOPS for slot  0 subslot 32768 IOXS_GOOD size   0 "DAP Interface 1"
  Set initial input data and IOPS for slot  0 subslot 32769 IOXS_GOOD size   0 "DAP Port 1"
Application will signal that it is ready for data, for AREP 1.
Event indication PNET_EVENT_APPLRDY   AREP: 1
PLC ccontrol message confirmation (The PLC has received our Application Ready message). AREP: 1  Status codes: 0 0 0 0
Event indication PNET_EVENT_DATA   AREP: 1
Cyclic data transmission started

if
ifconfig
msh />ifconfig
network interface device: e0 (Default)
MTU: 1500
MAC: 00 11 22 33 44 55
FLAGS: UP LINK_UP INTERNET_DOWN DHCP_DISABLE ETHARP BROADCAST IGMP
ip address: 192.168.10.100
gw address: 192.168.10.1
net mask   : 255.255.255.0
dns server #0: 0.0.0.0
dns server #1: 0.0.0.0
msh />ping 192.168.10.99
ping: not found specified netif, using default netdev e0.
60 bytes from 192.168.10.99 icmp_seq=0 ttl=128 time=0 ms
60 bytes from 192.168.10.99 icmp_seq=1 ttl=128 time=0 ms
60 bytes from 192.168.10.99 icmp_seq=2 ttl=128 time=0 ms
60 bytes from 192.168.10.99 icmp_seq=3 ttl=128 time=0 ms
msh />
```

图 21-4 开发板端 IP 信息

21.6 软 PLC 启动

CODESYS 简介：CODESYS 是德国 3S 公司开发的 PLC 软件，集成了 PLC 逻辑、运动控制、组态显示等功能。CODESYS，全称为“Controller Development System”，是一种基于 IEC 61131-3 标准的工业自动化编程工具。它不仅支持多种编程语言（如梯形图、结构化文本、功能块图等），还提供了丰富的库和功能模块，帮助工程师快速开发和调试 PLC（可编程逻辑控制器）和工业控制系统。CODESYS 的灵活性和强大功能使其成为工业自动化领域广泛使用的开发平台。

21.6.1 CODESYS 创建标准工程

请确保已安装 CODESYS 软件，安装之后下面这三个是我们需要用到的软件：

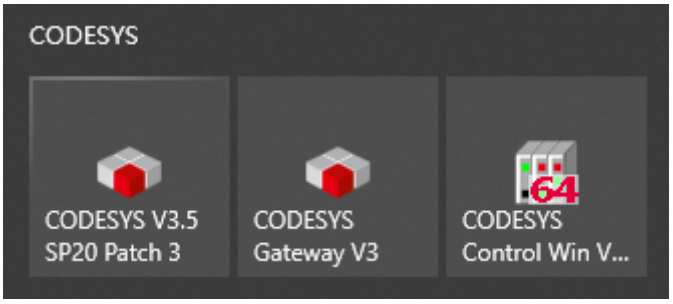


图 21-5 CODESYS 软件

- CODESYS V3.5 SP20 Patch 3：Profinet 主站模拟
- CODESYS Gateway V3：网关设备
- CODESYS Control Win V3 -x64 SysTray：软 PLC 设备

首先打开 **CODESYS V3.5 SP20 Patch 3**，依次选择 -> 新建工程 -> Projects -> Standard project ，配置工程名称及位置后点击确定：

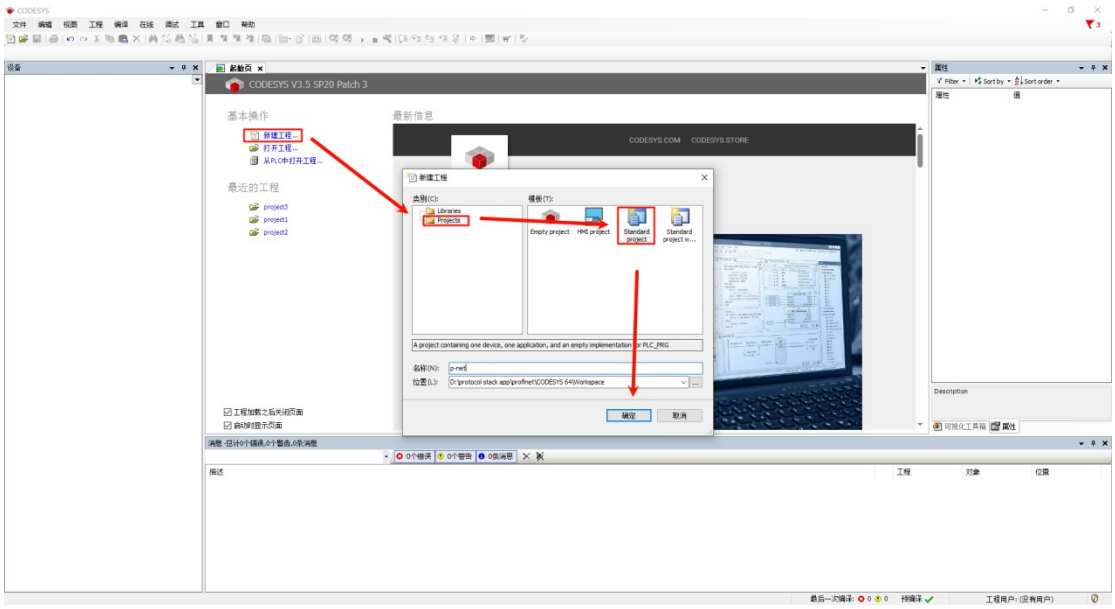


图 21-6 新建工程

弹出下面这个弹窗后保持默认配置(CODESYS Control Win V3 (CODESYS) / x64 (CODESYS))点击确定：

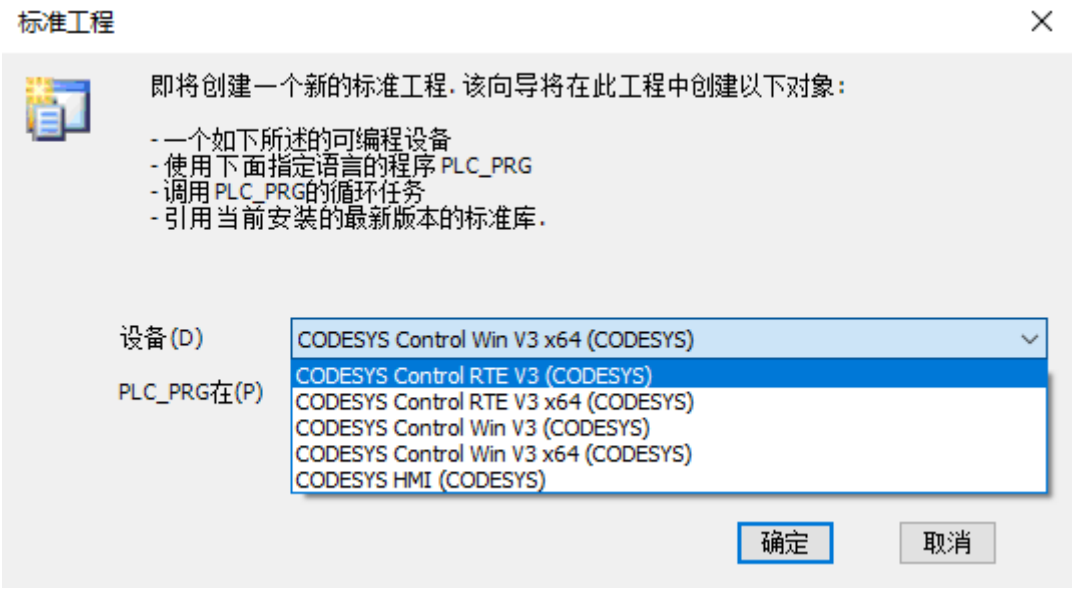


图 21-7 创建对象

注意：如果您购买了 CODESYS Control RTE SL，可选择设备：CODESYS Control RTE V3 (CODESYS) / x64 (CODESYS)，正常评估用途可选择不安装此扩展包，选择 CODESYS Control Win V3 (CODESYS) / x64 (CODESYS) 设备创建即可。

创建成功后就可以看到主界面了：

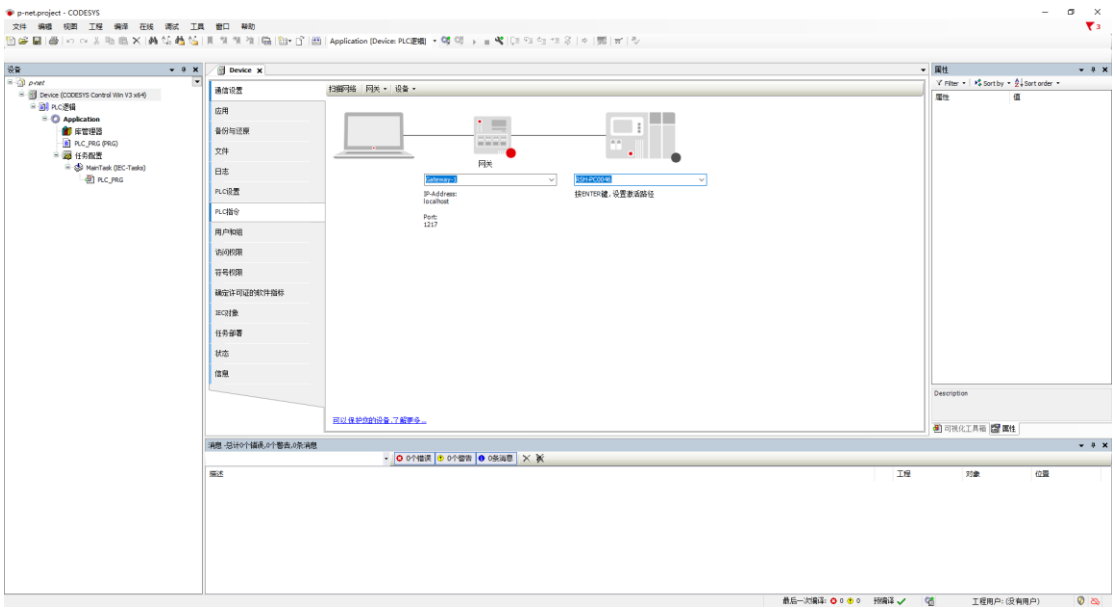


图 21-8 CODESYS 主界面

21.6.2 Gateway 及 软 PLC 启动

依次打开下面两个软件：

- CODESYS Gateway V3（右键 Start Gateway）
- CODESYS Control Win V3 -x64 SysTray（右键 Start PLC）

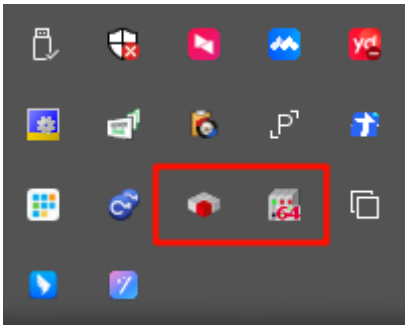


图 21-9 启动 CODESYS 软件

回到 CODESYS 主站软件，双击 Device(CODESYS Control Win V3 x64) -> 通信设置 -> 扫描网络：

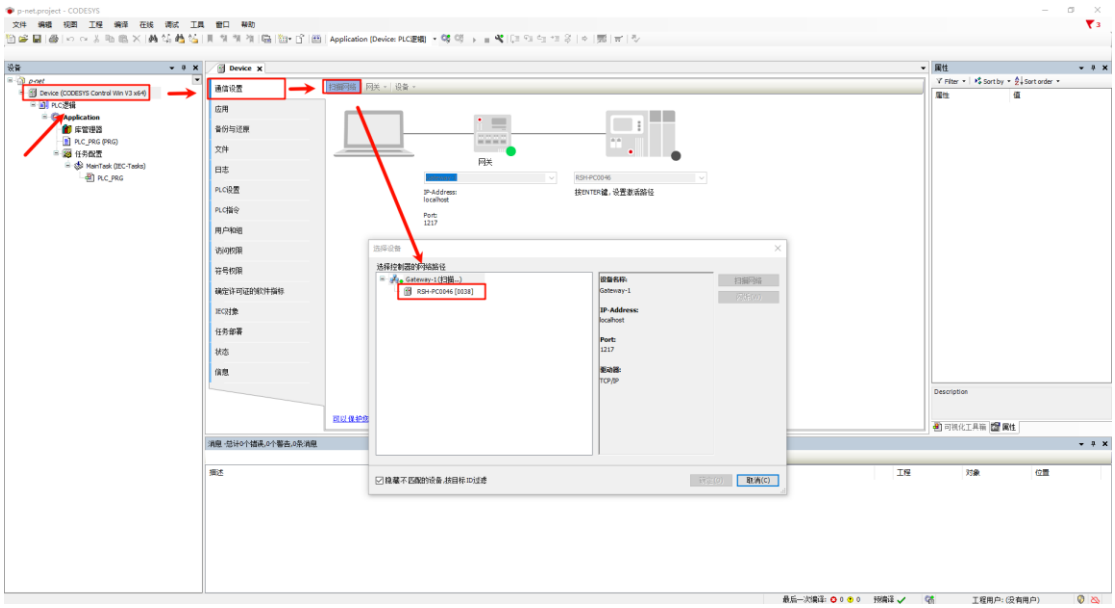


图 21-10 扫描网关设备

弹出设备用户登录窗口后，配置用户名和密码（用户自定义）：

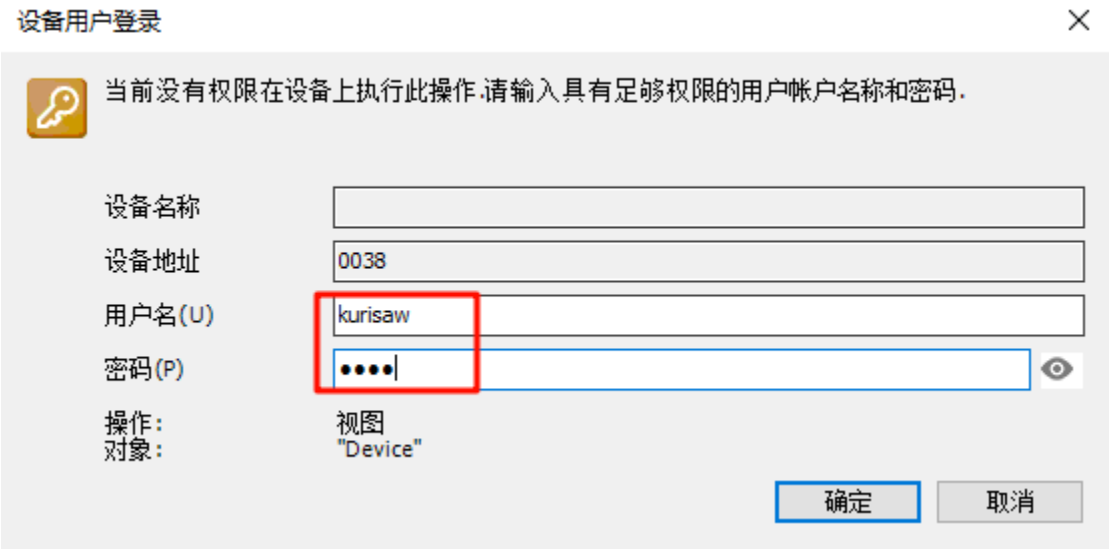


图 21-11 设备用户登录

检查网关设备及软 PLC 设备是否在线：

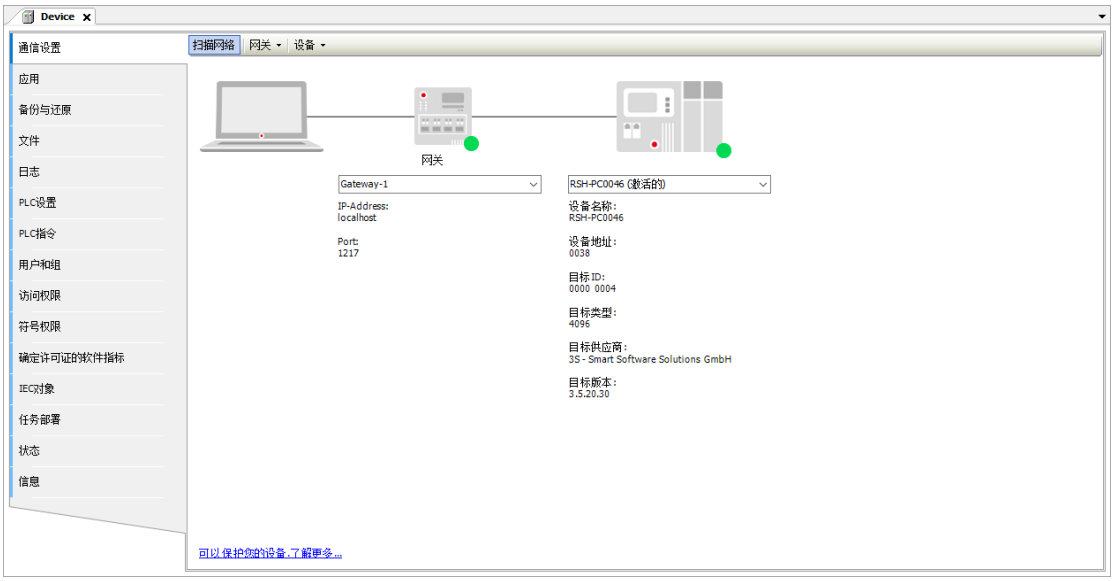


图 21-12 查看网关设备及软 PLC 设备在线

21.6.3 profinet GSDML 文件添加

GSD(Generic Station Description file)：即通用站点描述文件，主要用于 PROFIBUS DP（GSD 文件）和 PROFINET IO（GSDML 文件）通信，作为描述文件，是 PLC 系统中 CPU 模块和 IO 模块之间的桥梁，通常包括通道数据、参数数

据、诊断数据以及用户自定义数据。

本项目的 GSDML 文件位于如下路径：

- ..\src\ports\rtthread\pn_dev

选择设备存储库安装描述文件，选择上述路径下的 GSDML-V2.4-RT-Labs-P-Net-Sample-App-20220324.xml 文件。

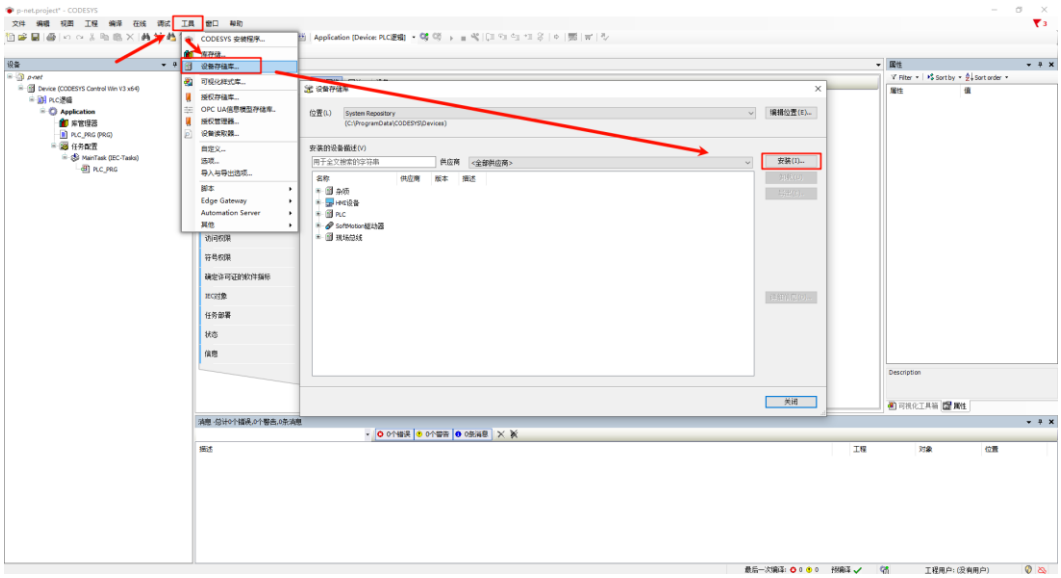


图 21-13 安装 XML 文件

安装成功后可以看到 p-net 从站描述文件：

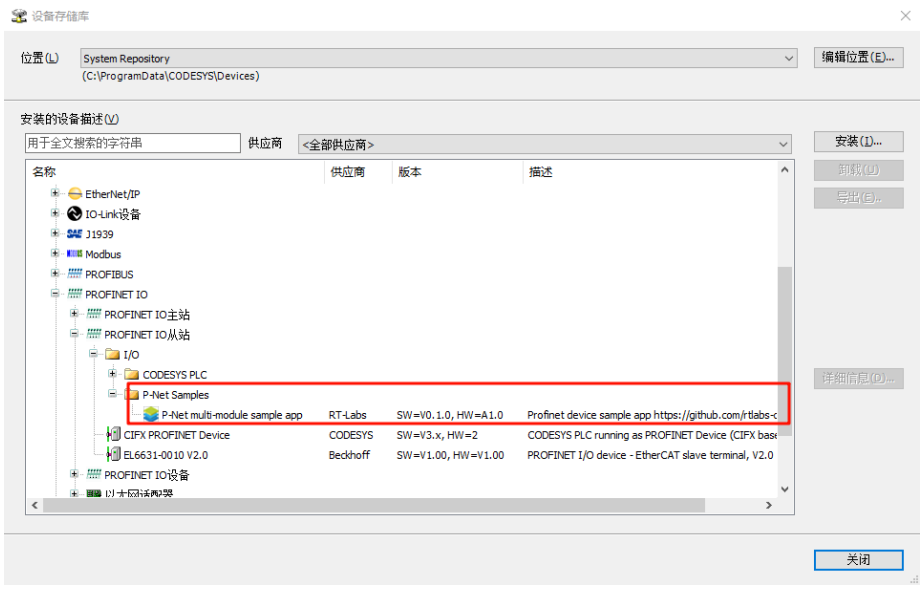


图 21-14 P-Net 描述文件安装

21.6.4 设备添加

- Ethernet 添加：左侧导航栏点击 Device 并右键添加设备，选择以太网适配器；

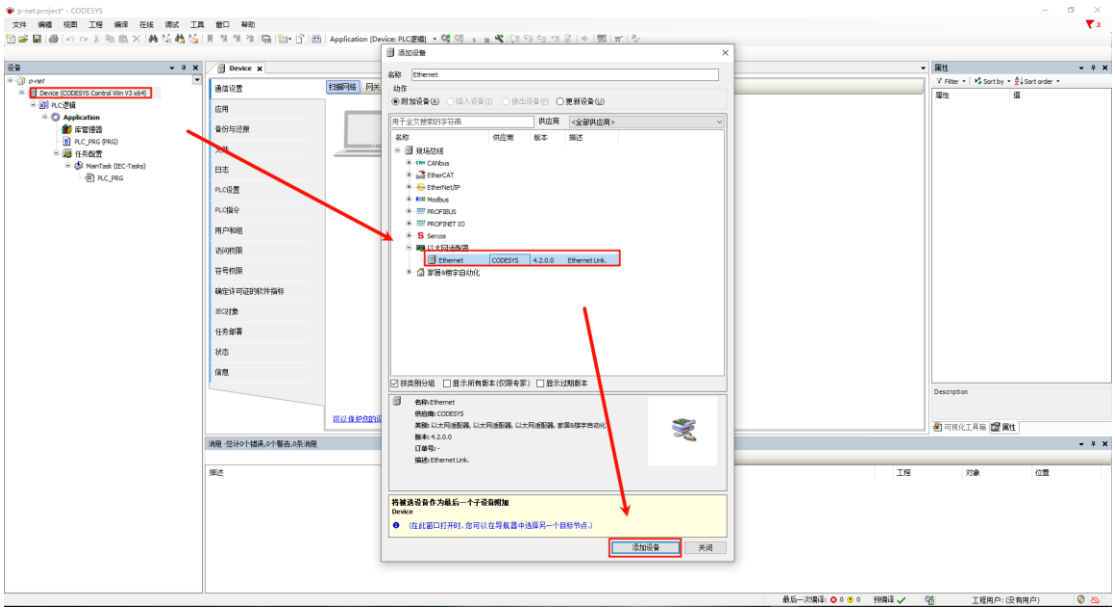


图 21-15 Ethernet 添加

- PROFINET IO 主站添加：右键左侧导航栏中的 Ethernet，选择 PN-Controller

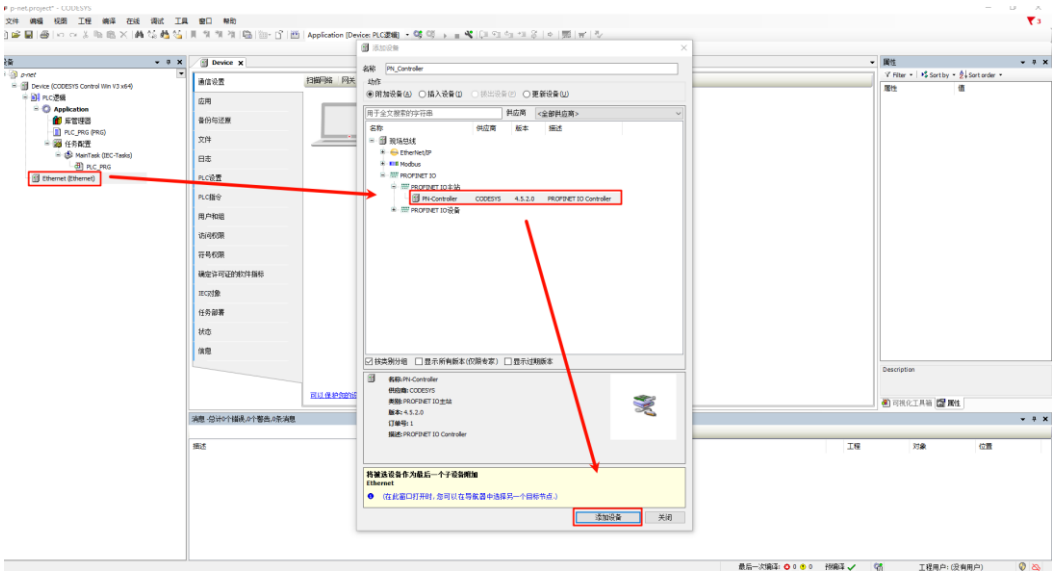


图 21-16 PROFINET IO 主站添加

- PROFINET IO 从站添加：右键左侧导航栏中的 PN-Controller，选择 P-Net-multiple-module sample app

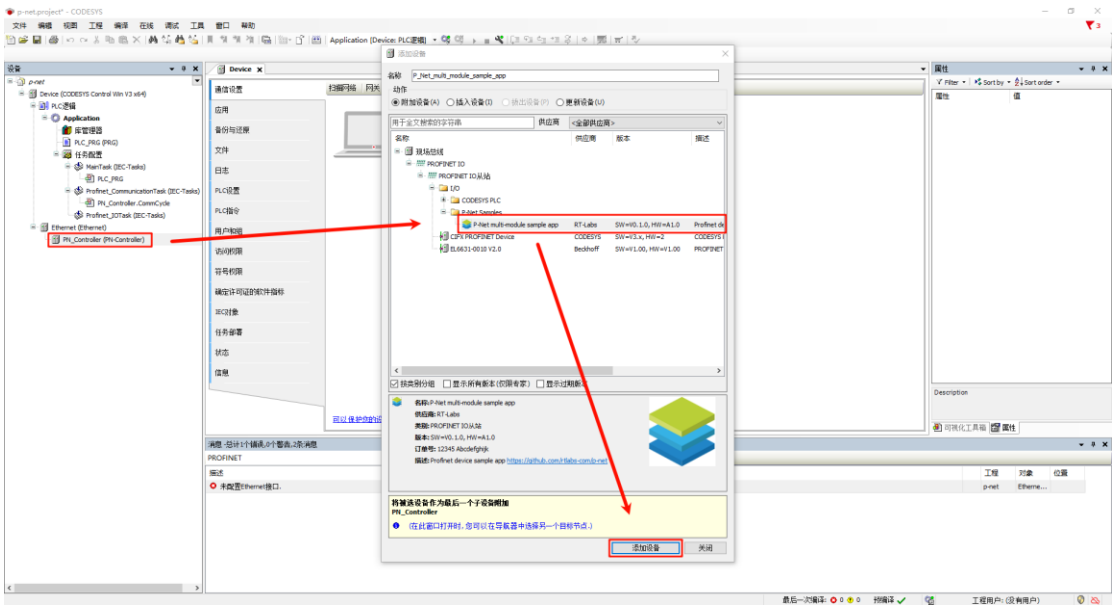


图 21-17 PROFINET IO 从站添加

21.6.5 任务响应

- Main Tasks 配置：左侧导航栏选择 Application -> 任务配置 -> 双击 MainTask(IEC-Tasks)，优先级设置为 1，类型选择循环，周期选择 4ms；

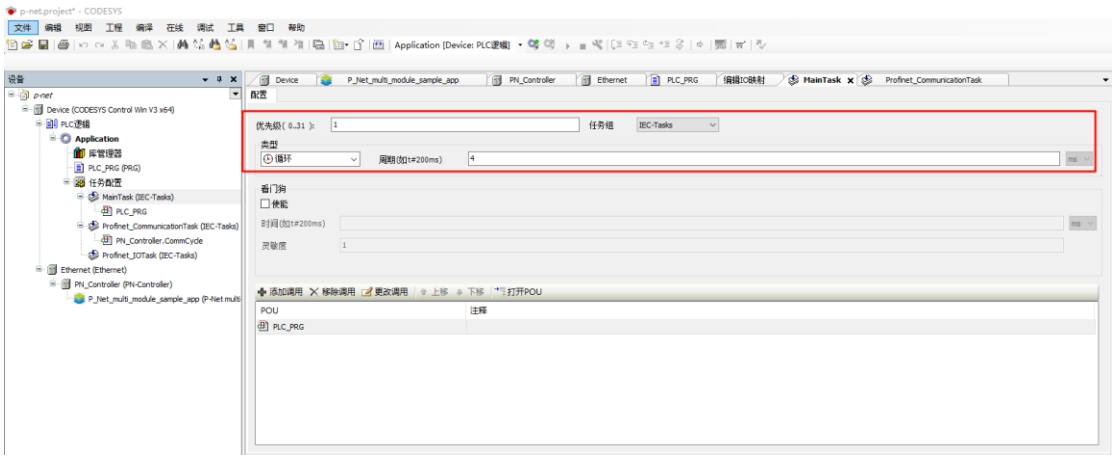


图 21-18 Main Tasks

- Profinet_CommunicationTask 配置：双击 Profinet_CommunicationTask(IEC-Tasks)，优先级设置为 14，类型选择循环，周期设置为 10ms。

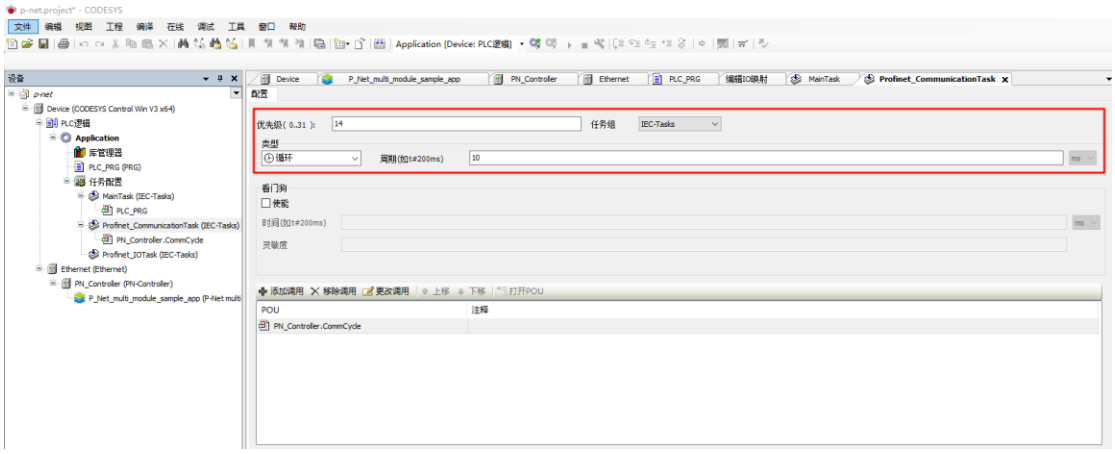


图 21-19 Profinet_CommunicationTask

21.6.6 网络配置

- Ethernet 配置：双击左侧导航栏中的 Ethernet(Ethernet) -> 通用，修改网络接口为连接到开发板的以太网端口；

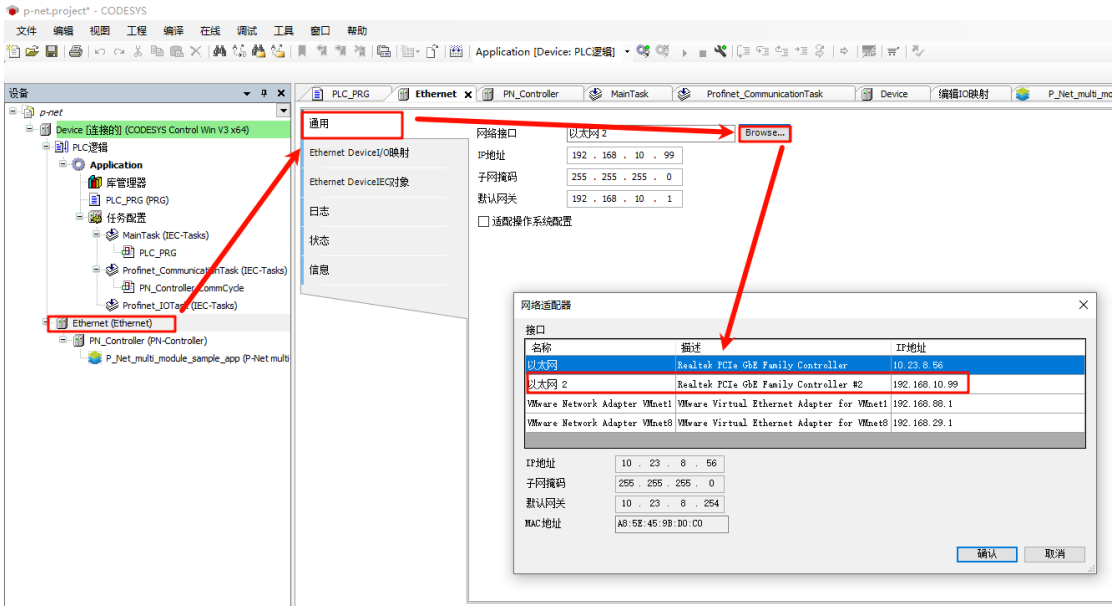


图 21-20 Ethernet 配置

- PN_Controller 配置：双击左侧导航栏 PN_Controller(PN-Controller)

-> 通用，并正确修改默认从站 IP 参数的区间，根据提示修改即可。

- P-Net 从站网络配置：双击左侧导航栏 P-Net-multiple-module sample app -> 通用， 修改 IP 参数为开发板 IP。

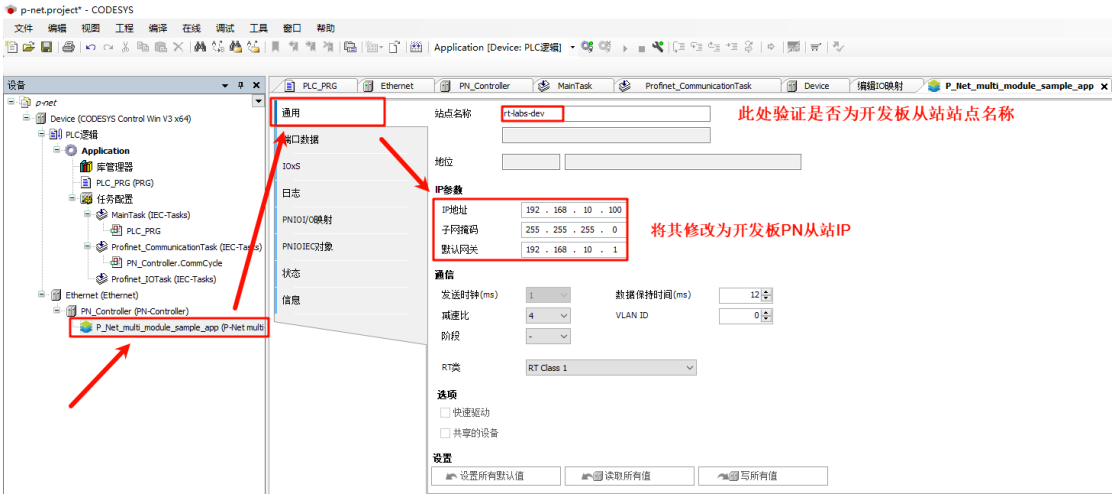
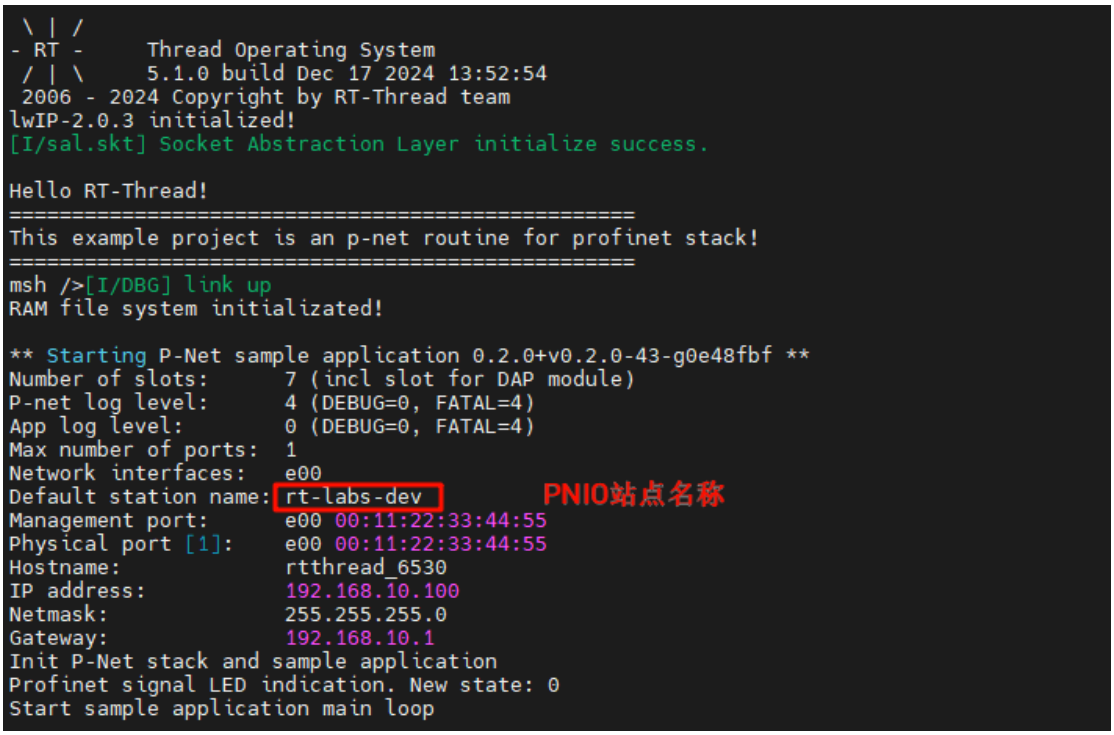


图 21-21 P-Net 从站网络配置




```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Dec 17 2024 13:52:54
 2006 - 2024 Copyright by RT-Thread team
lwIP-2.0.3 initialized!
[I/sal.skt] Socket Abstraction Layer initialize success.

Hello RT-Thread!
================================================
This example project is an p-net routine for profinet stack!
================================================
msh />[I/DBG] link up
RAM file system initialized!

** Starting P-Net sample application 0.2.0+v0.2.0-43-g0e48fbf **
Number of slots:      7 (incl slot for DAP module)
P-net log level:      4 (DEBUG=0, FATAL=4)
App log level:        0 (DEBUG=0, FATAL=4)
Max number of ports:  1
Network interfaces:   e00
Default station name: rt-labs-dev
Management port:      e00 00:11:22:33:44:55
Physical port [1]:    e00 00:11:22:33:44:55
Hostname:             rtthread_6530
IP address:           192.168.10.100
Netmask:              255.255.255.0
Gateway:              192.168.10.1
Init P-Net stack and sample application
Profinet signal LED indication. New state: 0
Start sample application main loop
```

图 21-22 PNIO 网络信息

21.6.7 工程编译并启动调试

- step1：工程上方导航栏选择 编译-> 生成代码
- step2：选择 在线 -> 登录
- step3：点击 调试 -> 启动

此时就可以看到 PN 主站已经上线成功

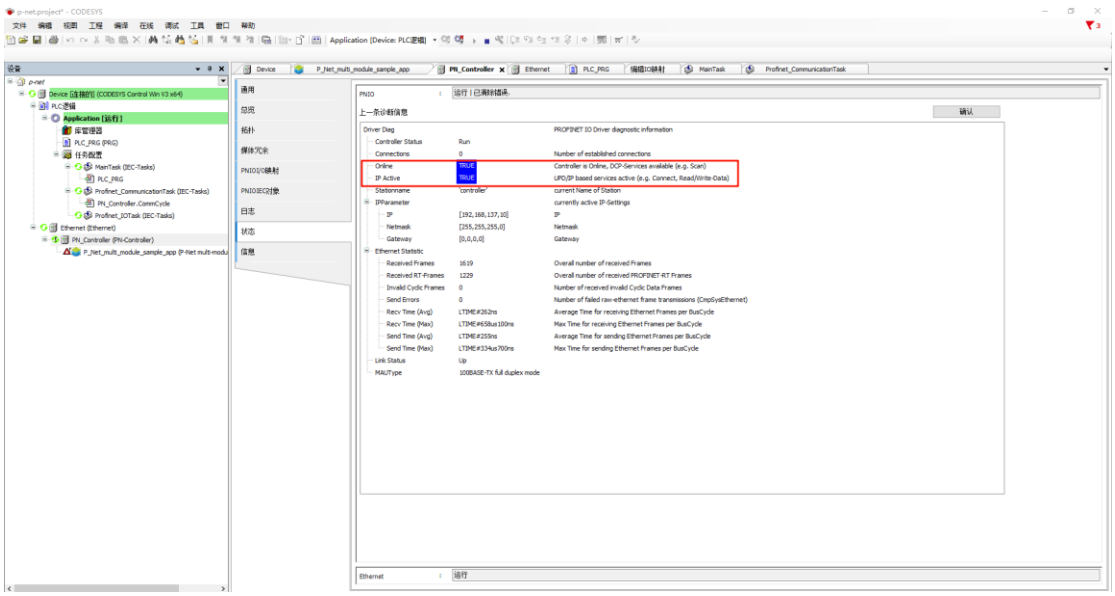


图 21-23 PN 上线成功

21.7 profinet 从站应用启动

开发板端上电后，一旦检测到网卡 link up，则会自动启动 PN 从站：

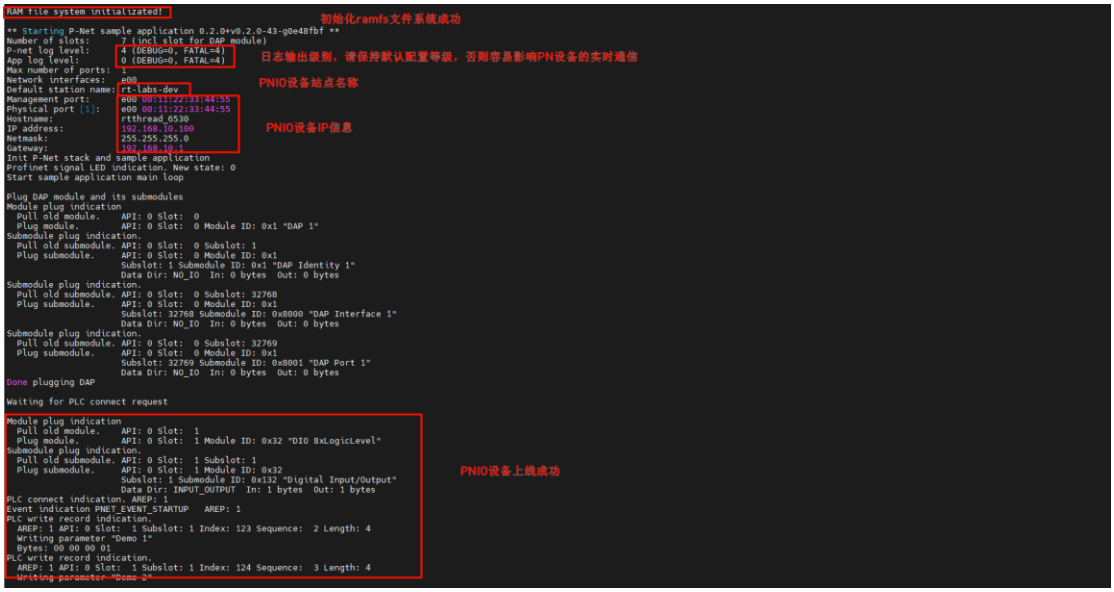


图 21-24 PNIO 响应日志

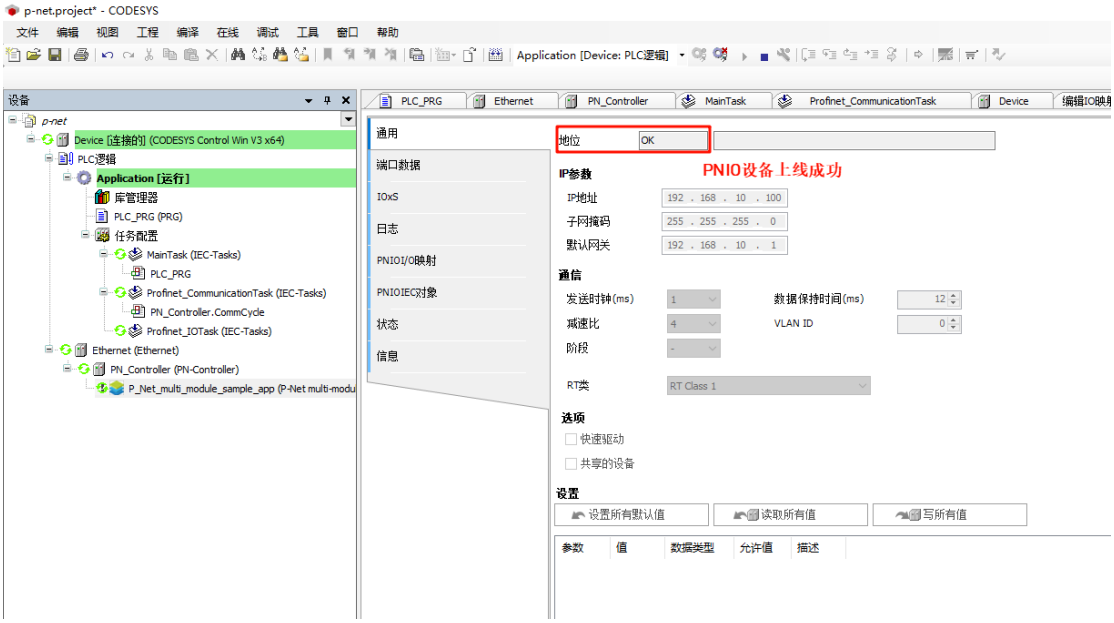


图 21-25 PN 从站状态

21.8 PN 协议栈运行 demo

这里我们使用 CODESYS 软件来测试 PN 的主从站交互。

21.8.1 LED 闪烁

回到 CODESYS 软件，左侧导航栏选择 PN_Controller，右键点击扫描设备，单击设备名后点击闪烁 LED：

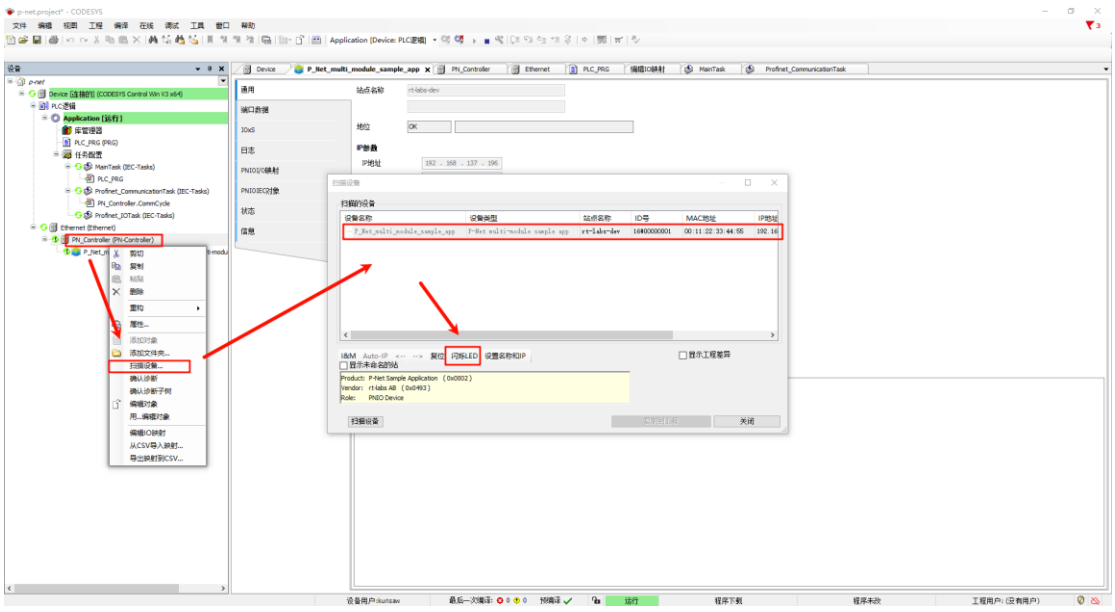


图 21-26 闪烁 PNIO LED

此时的开发板端（PN 从站 IO）可以看到日志输出，并伴随板载 User LED 闪烁：

```
Profinet signal LED indication. New state: 1
Profinet signal LED indication. New state: 0
Profinet signal LED indication. New state: 1
Profinet signal LED indication. New state: 0
Profinet signal LED indication. New state: 1
Profinet signal LED indication. New state: 0
```

图 21-27 PNIO 日志

21.8.2 从站 I&M(标识和维护) 数据修改

依然是扫描设备界面，我们点击左下角的 I&M，修改信息并写入 I&M：

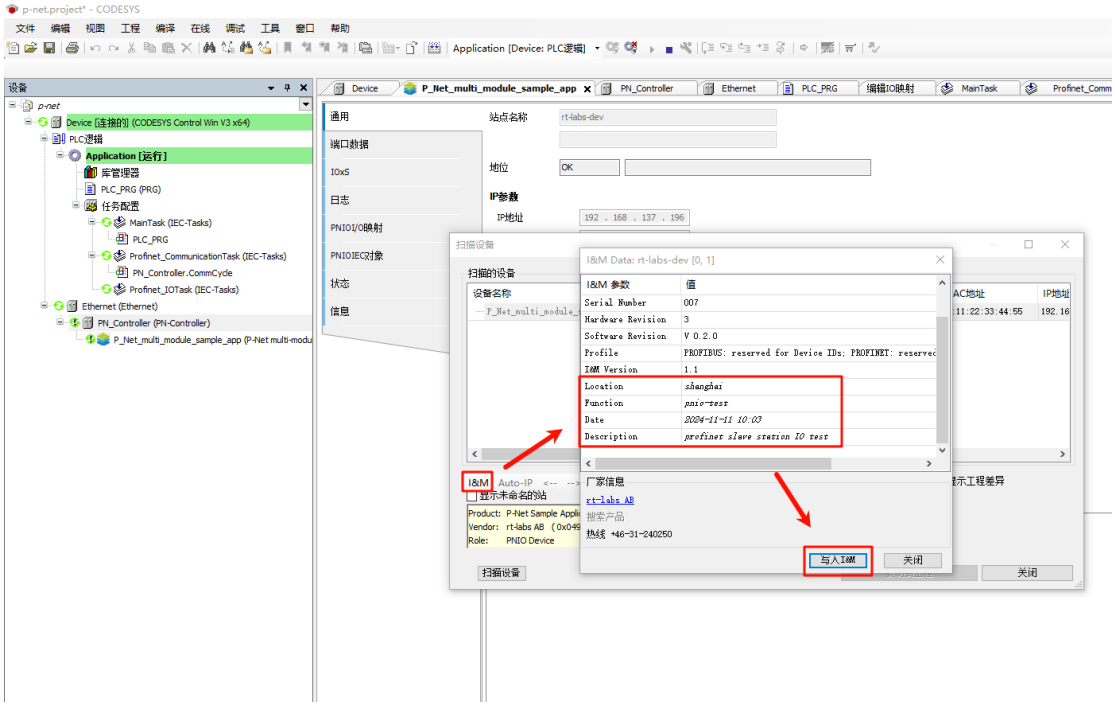


图 21-28 I&M 修改

同时 PNIO 会更新从站配置信息：

```
PLC connect indication. AREP: 2
Event indication PNET_EVENT_STARTUP   AREP: 2
Event indication PNET_EVENT_ABORT   AREP: 2
    Error class: 0xfd Real-Time Acyclic Protocol
    Error code:  0x0f AR release indication received
Setting outputs to default values.
Event indication PNET_EVENT_ABORT   AREP: 1
    Error class: 0xfd Real-Time Acyclic Protocol
    Error code:  0x05 Device missed cyclic data deadline, device terminated AR
Setting outputs to default values.
Connection (AREP 1) closed
PLC connect indication. AREP: 1
Event indication PNET_EVENT_STARTUP   AREP: 1
PLC dcontrol message (The PLC is done with parameter writing). AREP: 1  Command: PRM_END
Event indication PNET_EVENT_PRMEND   AREP: 1
  Set initial input data and IOPS for slot  0 subslot     1 IOXS_GOOD size   0 "DAP Identity 1"
  Set initial input data and IOPS for slot  0 subslot 32768 IOXS_GOOD size   0 "DAP Interface 1"
  Set initial input data and IOPS for slot  0 subslot 32769 IOXS_GOOD size   0 "DAP Port 1"
Application will signal that it is ready for data, for AREP 1.
Event indication PNET_EVENT_APPLRDY   AREP: 1
Data status indication. AREP: 1  Data status changes: 0x35  Data status: 0x35
   Run, Valid, Primary, Normal operation, Evaluate data status
Event indication PNET_EVENT_ABORT   AREP: 1
    Error class: 0x00 Not decoded
    Error code:  0x00 Not decoded
Setting outputs to default values.
Connection (AREP 1) closed
PLC connect indication. AREP: 1
Event indication PNET_EVENT_STARTUP   AREP: 1
PLC dcontrol message (The PLC is done with parameter writing). AREP: 1  Command: PRM_END
Event indication PNET_EVENT_PRMEND   AREP: 1
  Set initial input data and IOPS for slot  0 subslot     1 IOXS_GOOD size   0 "DAP Identity 1"
  Set initial input data and IOPS for slot  0 subslot 32768 IOXS_GOOD size   0 "DAP Interface 1"
  Set initial input data and IOPS for slot  0 subslot 32769 IOXS_GOOD size   0 "DAP Port 1"
Application will signal that it is ready for data, for AREP 1.
Event indication PNET_EVENT_APPLRDY   AREP: 1
Data status indication. AREP: 1  Data status changes: 0x35  Data status: 0x35
   Run, Valid, Primary, Normal operation, Evaluate data status
PLC ccontrol message confirmation (The PLC has received our Application Ready message). AREP: 1  Status codes: 0 0 0 0
Event indication PNET_EVENT_DATA   AREP: 1
Cyclic data transmission started
```

图 21-29 PNIO 更新日志

我们再次点击查看 I&M，即可发现 I&M 修改成功！

21.8.3 PLC 编程及 PNIO 控制

首先我们点击左侧面板的 Device->PLC 逻辑->Application->PLC_PRG(PRG)，使用 ST 语言编程，编写变量及程序代码：

- 变量定义：这些变量定义了按钮的输入状态（in_pin_button_LED），LED 的输出状态（out_pin_LED）以及控制 LED 是否闪烁的状态变量（flashing）。振荡器状态（oscillator_state）和振荡器周期计数器（oscillator_cycles）用来实现定时闪烁效果。

```
PROGRAM PLC_PRG
VAR
    in_pin_button_LED: BOOL;
    out_pin_LED: BOOL;
    in_pin_button_LED_previous: BOOL;
    flashing: BOOL := TRUE;
    oscillator_state: BOOL := FALSE;
    oscillator_cycles: UINT := 0;
END_VAR
```

- 程序定义：

1. 首先在每次循环中，oscillator_cycles 增加 1。当计数器超过 200 时，重置计数器并切换 oscillator_state 的状态（TRUE 或 FALSE），实现周期性变化；

2. 如果按钮被按下（in_pin_button_LED 为 TRUE），并且在上一周期按钮状态是 FALSE，则切换 flashing 状态。即每次按钮按下时，切换 LED 是否闪烁的状态。

3. 如果 flashing 为 TRUE，则 LED 会根据振荡器状态 (oscillator_state) 闪烁；如果 flashing 为 FALSE，LED 直接关闭。

4. 在每次循环结束时，将当前按钮的状态保存在 in_pin_button_LED_previous 中，以便在下次判断按钮按下的事件。

```
oscillator_cycles := oscillator_cycles + 1;
IF oscillator_cycles > 200 THEN
    oscillator_cycles := 0;
```

```
    oscillator_state := NOT oscillator_state;
END_IF
IF in_pin_button_LED = TRUE THEN
    IF in_pin_button_LED_previous = FALSE THEN
        flashing := NOT flashing;
    END_IF
    out_pin_LED := TRUE;
ELSIF flashing = TRUE THEN
    out_pin_LED := oscillator_state;
ELSE
    out_pin_LED := FALSE;
END_IF
in_pin_button_LED_previous := in_pin_button_LED;
```

工程中的配置位置如下图所示：

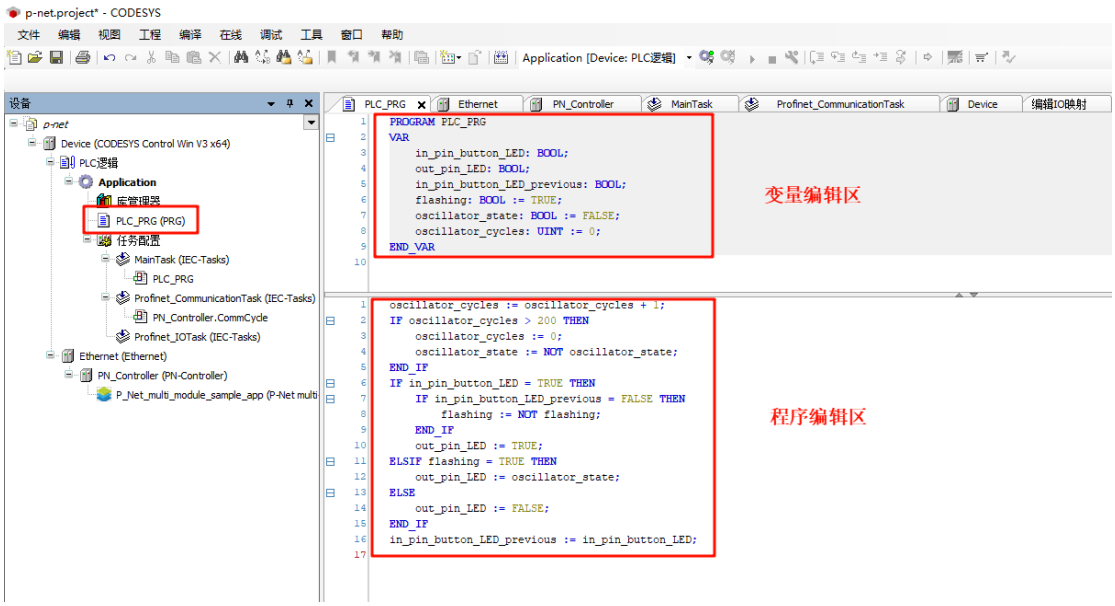


图 21-30 PLC 编程

接下来我们还需要添加一个内置的 IO 模块，右键点击 P_Net_multi_module_sample_app 然后添加一个 IO 模块（DIO 8xLogicLevel），如下图所示：

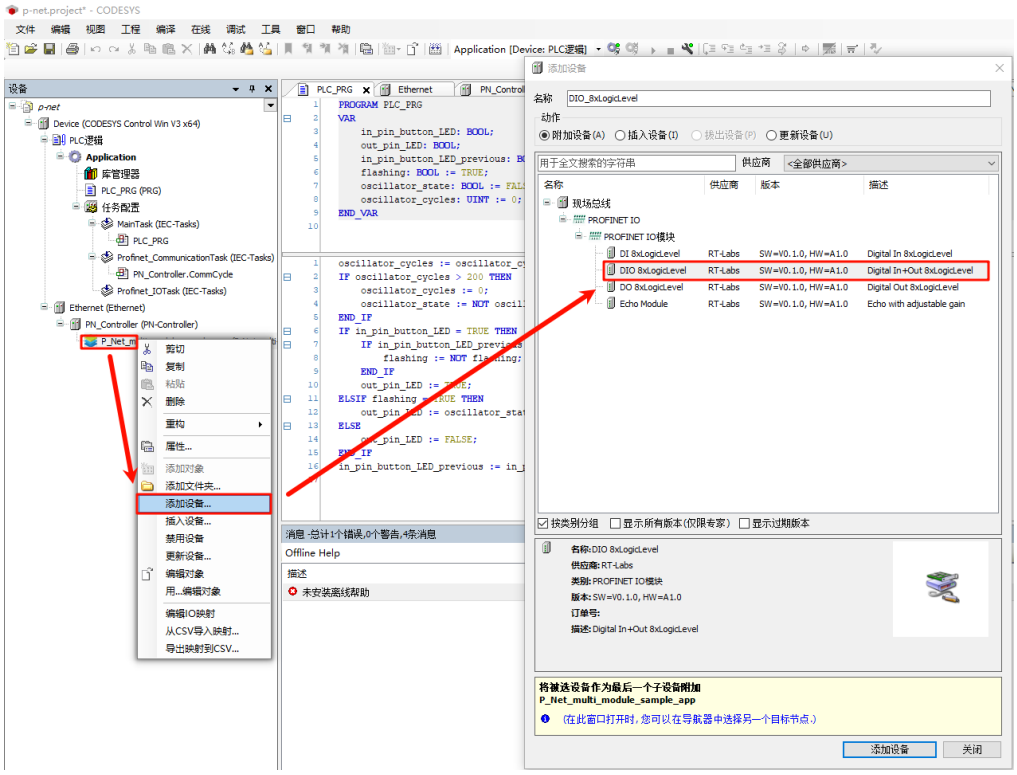


图 21-31 添加 IO 模块

接下来双击 DIO_8xLogicLevel 节点，选择 PNIO Module I/O 映射，编辑 Input Bit 7 和 Output Bit 7 并绑定 PLC 变量：

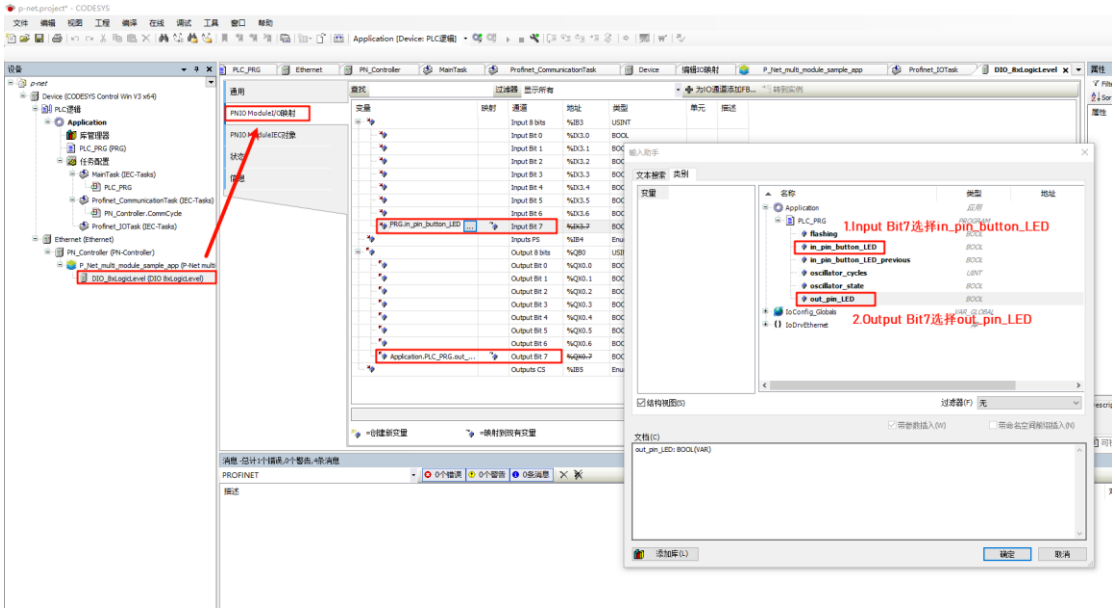


图 21-32 IO 映射编辑

接着我们点击上方导航栏的编译->生成代码，然后选择在线->登录，运行查看现象；

```
  Pull old module.    API: 0 Slot:  0
  Plug module.        API: 0 Slot:  0 Module ID: 0x1 "DAP 1"
Submodule plug indication.
  Pull old submodule. API: 0 Slot:  0 Subslot: 1
  Plug submodule.     API: 0 Slot:  0 Module ID: 0x1
                      Subslot: 1 Submodule ID: 0x1 "DAP Identity 1"
                      Data Dir: NO_IO  In: 0 bytes  Out: 0 bytes
Submodule plug indication.
  Pull old submodule. API: 0 Slot:  0 Subslot: 32768
  Plug submodule.     API: 0 Slot:  0 Module ID: 0x1
                      Subslot: 32768 Submodule ID: 0x8000 "DAP Interface 1"
                      Data Dir: NO_IO  In: 0 bytes  Out: 0 bytes
Submodule plug indication.
  Pull old submodule. API: 0 Slot:  0 Subslot: 32769
  Plug submodule.     API: 0 Slot:  0 Module ID: 0x1
                      Subslot: 32769 Submodule ID: 0x8001 "DAP Port 1"
                      Data Dir: NO_IO  In: 0 bytes  Out: 0 bytes
Done plugging DAP

Waiting for PLC connect request

Module plug indication
  Pull old module.    API: 0 Slot:  1
  Plug module.        API: 0 Slot:  1 Module ID: 0x32 "DIO 8xLogicLevel"
Submodule plug indication.
  Pull old submodule. API: 0 Slot:  1 Subslot: 1
  Plug submodule.     API: 0 Slot:  1 Module ID: 0x32
                      Subslot: 1 Submodule ID: 0x132 "Digital Input/Output"
                      Data Dir: INPUT_OUTPUT  In: 1 bytes  Out: 1 bytes
PLC connect indication. AREP: 1
Event indication PNET_EVENT_STARTUP    AREP: 1
PLC write record indication.
  AREP: 1 API: 0 Slot:  1 Subslot: 1 Index: 123 Sequence:  2 Length: 4
  Writing parameter "Demo 1"
  Bytes: 00 00 00 01
PLC write record indication.
  AREP: 1 API: 0 Slot:  1 Subslot: 1 Index: 124 Sequence:  3 Length: 4
  Writing parameter "Demo 2"
  Bytes: 00 00 00 02
PLC dcontrol message (The PLC is done with parameter writing). AREP: 1  Command: PRM_END
Event indication PNET_EVENT_PRMEND    AREP: 1
  Set initial input data and IOPS for slot  0 subslot Data status indication. AREP: 1  Data status changes: 0x35  Data status: 0x35
   Run, Valid, Primary, Normal operation, Evaluate data status
a status
a status: 0x35
 "DAP Identity 1"
  Set initial input data and IOPS for slot  0 subslot 32768 IOXS_GOOD size   0 "DAP Interface 1"
  Set initial input data and IOPS for slot  0 subslot 32769 IOXS_GOOD size   0 "DAP Port 1"
  Set initial input data and IOPS for slot  1 subslot     1 IOXS_GOOD size   1 "Digital Input/Output"
  Set initial output        IOCS for slot  1 subslot     1 IOXS_GOOD          "Digital Input/Output"
Application will signal that it is ready for data, for AREP 1.
Event indication PNET_EVENT_APPLRDY    AREP: 1
PLC ccontrol message confirmation (The PLC has received our Application Ready message). AREP: 1  Status codes: 0 0 0 0
Event indication PNET_EVENT_DATA    AREP: 1
Cyclic data transmission started

PLC reports Provider Status (IOPS) GOOD for slot 1 subslot 1 "Digital Input/Output".
PLC reports Consumer Status (IOCS) GOOD for slot 1 subslot 1 "Digital Input/Output".
```

图 21-33 msh 终端

接下来回到 CODESYS，再次双击 Device->PLC 逻辑->Application 下的 PLC_PRG(PRG)，此时便可动态观察程序运行状态，例如我们按住 etherkit 开发板上的 KEY0，可以发现 in_pin_button_LED 及 in_pin_button_LED_previous 这两个变量值为 FALSE，此时再松开 KEY0，可以发现 flashing 值反转一次。

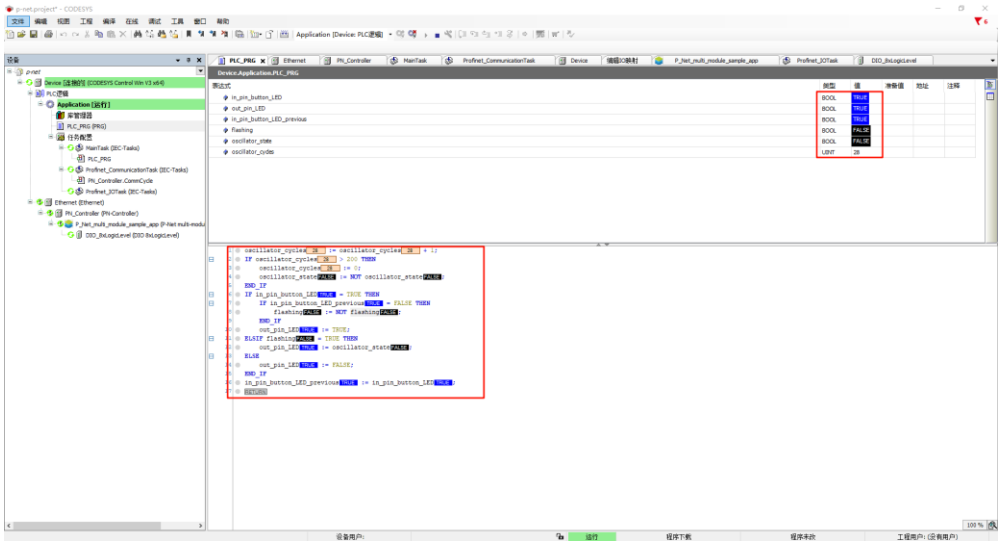


图 21-34 IO 映射测试

第 22 章 Ethernet/IP 例程

22.1 简介

Ethernet/IP（以太网工业协议）是一种基于标准以太网架构的工业通信协议，广泛应用于自动化和控制系统中。它结合了 TCP/IP 协议和 CIP（通用工业协议）标准，提供高速、可靠的数据传输，支持各种工业设备之间的实时通信。由于 Ethernet/IP 兼容现有的以太网硬件和网络，企业能够在不需要专用硬件的情况下，实现工业设备间的互联互通，提升生产效率和系统可靠性。

OpENer 是用于 I/O 适配器设备的 EtherNet/IP™ 堆栈；支持多个 I/O 和显式连接；包括用于制作符合以太网/IP 规范中定义并由 ODVA 发布的 EtherNet/IP™ 兼容产品的对象和服务。

在本示例中将使用已经适配的 OpENer 软件包来实现 Ethernet/IP 通讯。

22.2 前期准备

软件环境：

- CODESYS（Ethernet/IP 通信模拟）
    - CODESYS
    - CODESYS Gateway（网关设备）
    - CODESYS Control Win SysTray（软 PLC 设备）
- Npcap（该软件是运行 CODESYS 必须的，需要提前安装好！）

硬件环境：

- EtherKit 开发板

22.3 FSP 配置

此处配置请参考第 11 章：11.3.1 FSP 配置。

22.4 RT-Thread Settings 配置

双击打开 RT-Thread Settings，在搜索栏检索 OpENer 软件包并使能，下面是相关用户配置信息说明；

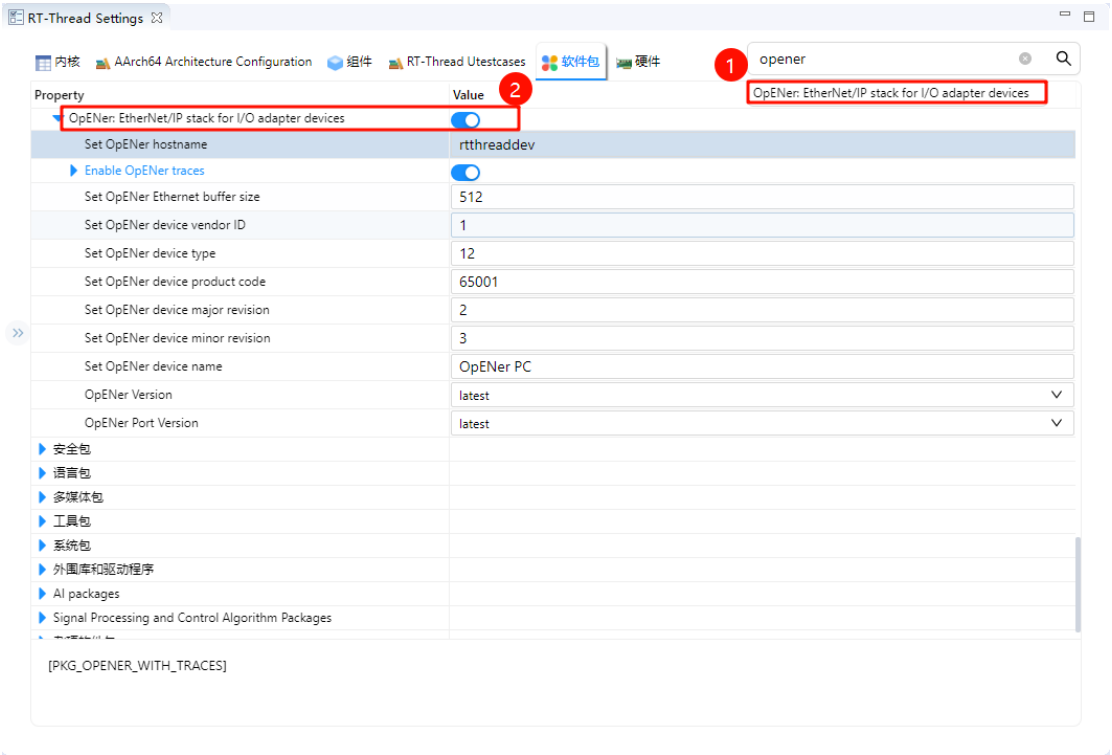


图 22-1 使能 OpENer 软件包

下面我们还需要配置禁用 dhcp 功能并使用静态 IP，点击组件->使能 lwip 堆栈，选择禁用 DHCP；

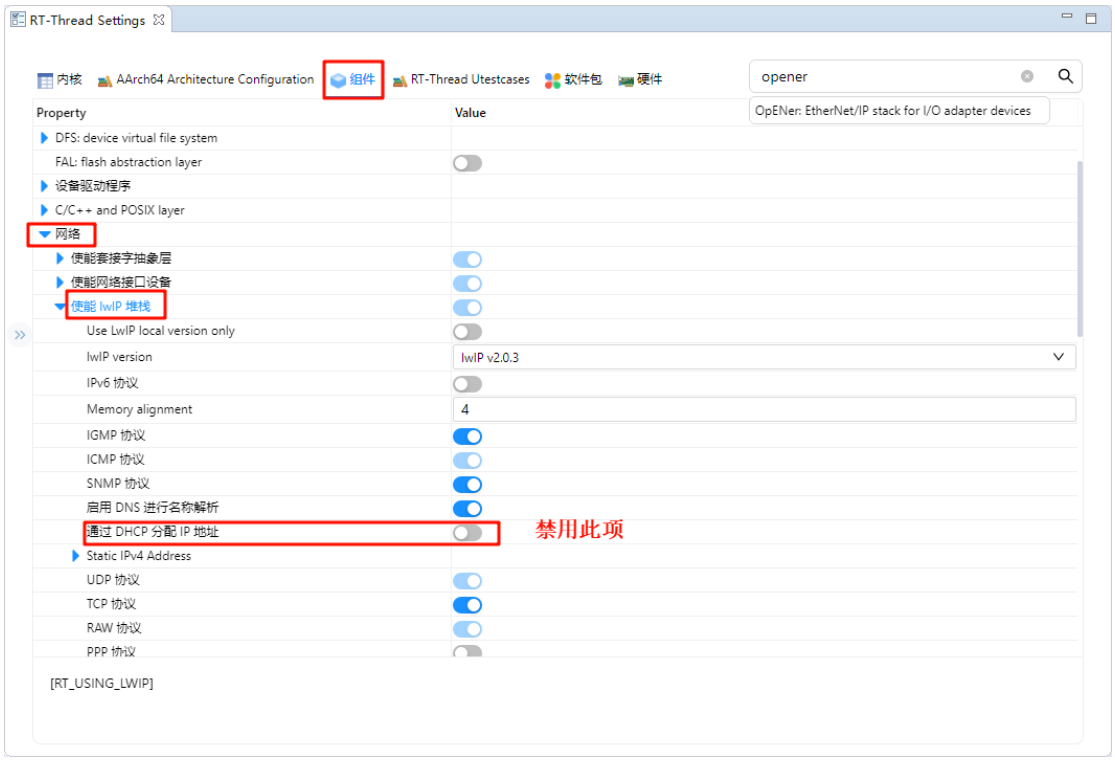


图 22-2 lwip 设置

完成上述配置后，将程序编译下载至开发板。

22.5 网络配置

我们使用一根网线连接开发板与 PC，同时在 PC 端配置静态 IP：

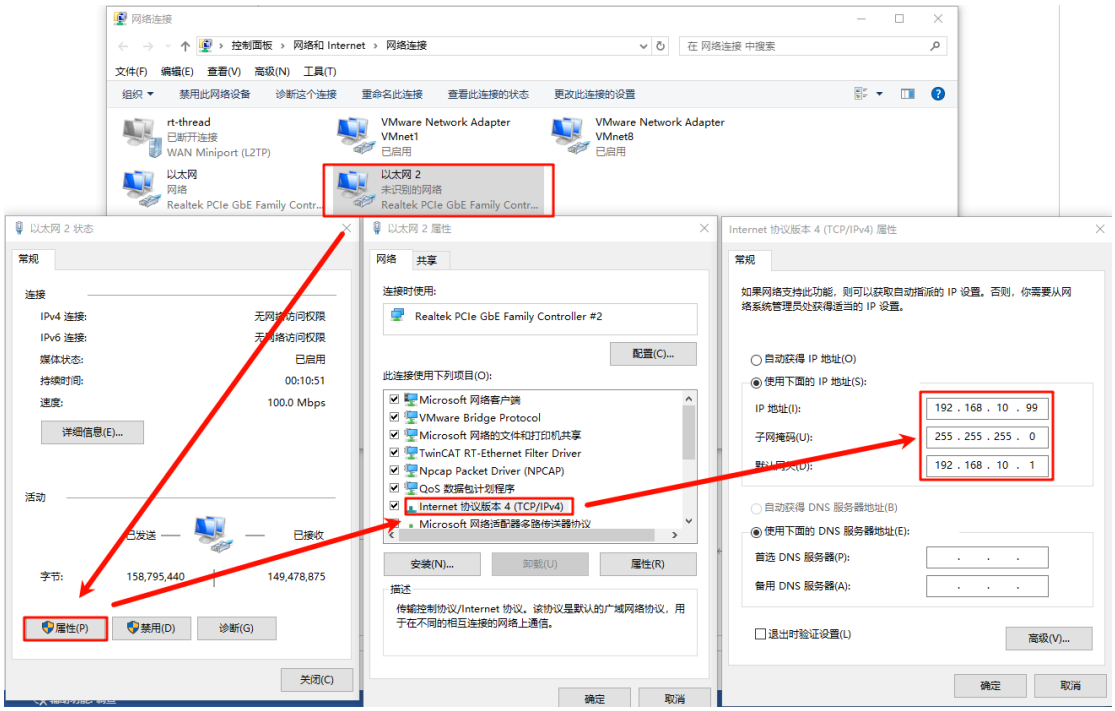


图 22-3 以太网静态 IP 配置

22.6 软 PLC 启动

CODESYS 简介：CODESYS 是德国 3S 公司开发的 PLC 软件，集成了 PLC 逻辑、运动控制、组态显示等功能。CODESYS，全称为“Controller Development System”，是一种基于 IEC 61131-3 标准的工业自动化编程工具。它不仅支持多种编程语言（如梯形图、结构化文本、功能块图等），还提供了丰富的库和功能模块，帮助工程师快速开发和调试 PLC（可编程逻辑控制器）和工业控制系统。CODESYS 的灵活性和强大功能使其成为工业自动化领域广泛使用的开发平台。

22.6.1 CODESYS 创建标准工程

请确保已安装 CODESYS 软件，安装之后下面这三个是我们需要用到的软件：

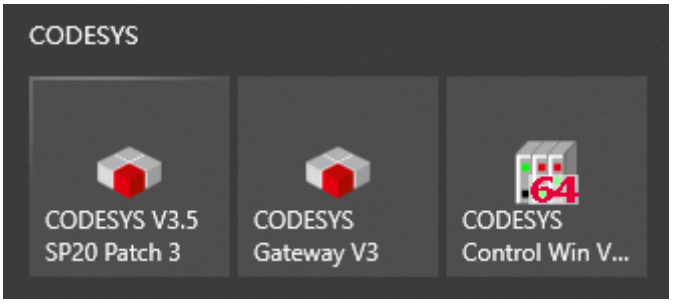


图 22-4 CODESYS 软件

- CODESYS V3.5 SP20 Patch 3：Ethernet/IP 通信模拟
- CODESYS Gateway V3：网关设备
- CODESYS Control Win V3 -x64 SysTray：软 PLC 设备

首先打开 **CODESYS V3.5 SP20 Patch 3**，依次选择 -> 新建工程 -> Projects -> Standard project ，配置工程名称及位置后点击确定：

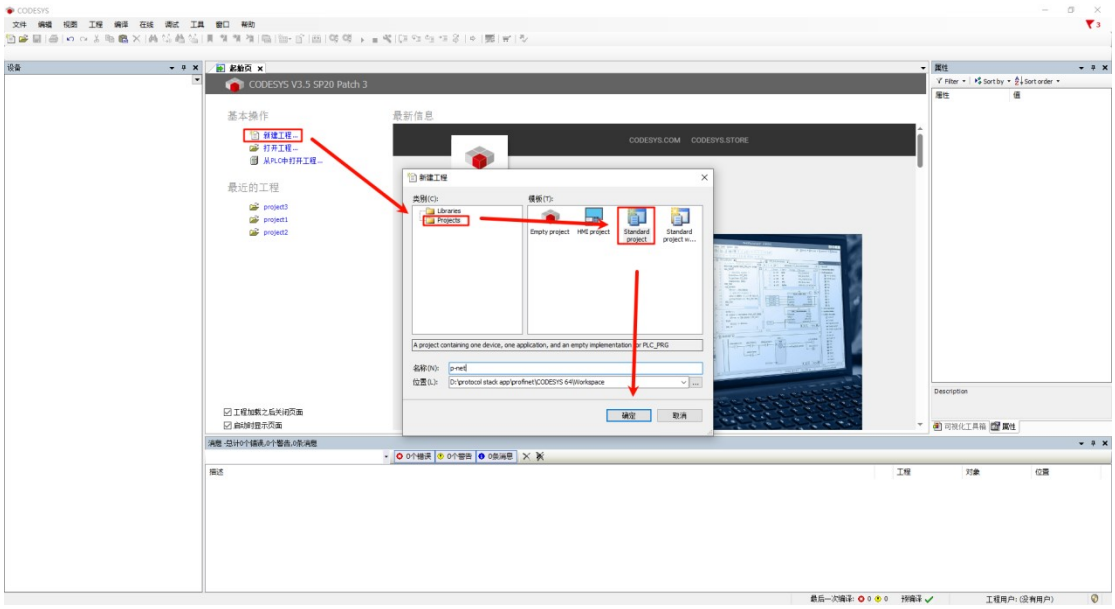


图 22-5 新建工程

弹出下面这个弹窗后保持默认配置(CODESYS Control Win V3 (CODESYS) / x64 (CODESYS))点击确定：

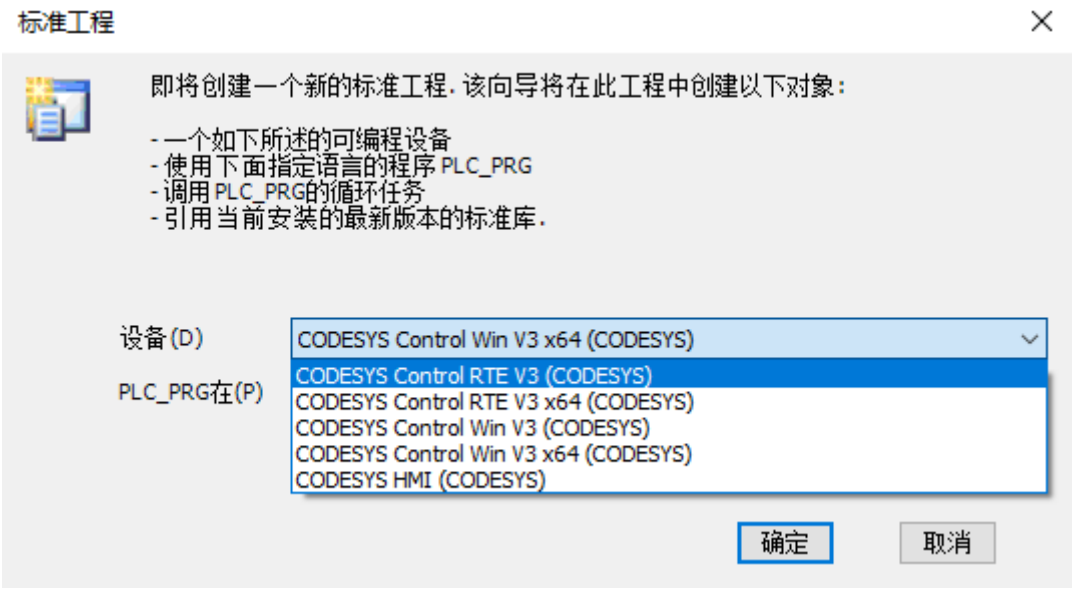


图 22-6 创建对象

注意：如果您购买了 CODESYS Control RTE SL，可选择设备：CODESYS Control RTE V3 (CODESYS) / x64 (CODESYS)，正常评估用途可选择不安装此扩展包，选择 CODESYS Control Win V3 (CODESYS) / x64 (CODESYS) 设备创建即可。

创建成功后就可以看到主界面了：

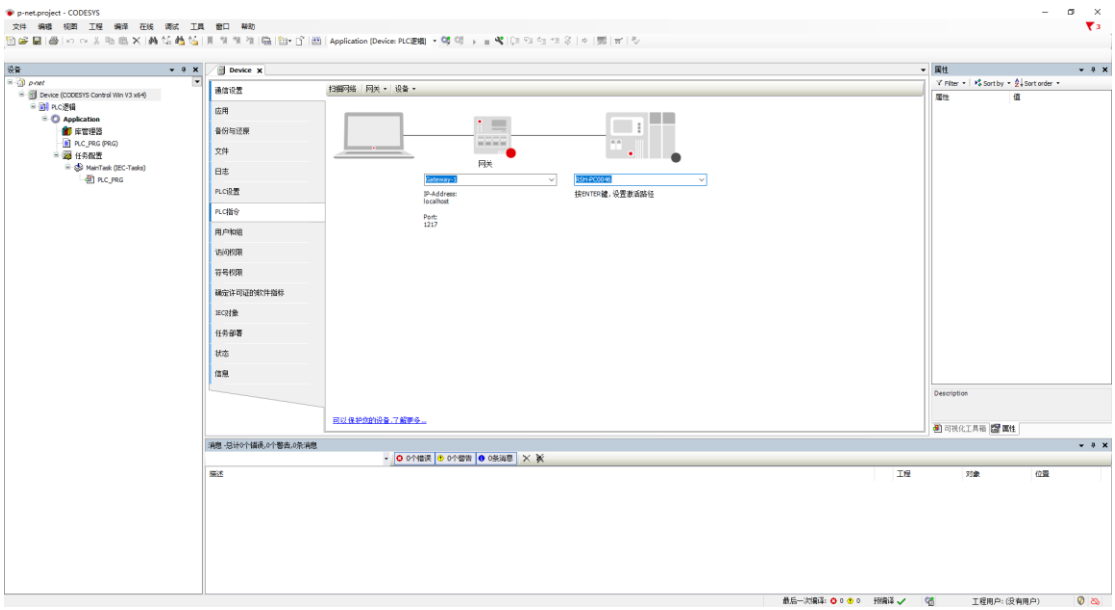


图 22-7 CODESYS 主界面

22.6.2 Gateway 及 软 PLC 启动

依次打开下面两个软件：

- CODESYS Gateway V3（右键 Start Gateway）
- CODESYS Control Win V3 -x64 SysTray（右键 Start PLC）

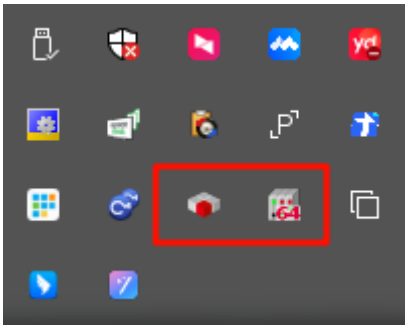


图 22-8 启动 CODESYS 软件

回到 CODESYS 主站软件，双击 Device(CODESYS Control Win V3 x64) -> 通信设置 -> 扫描网络：

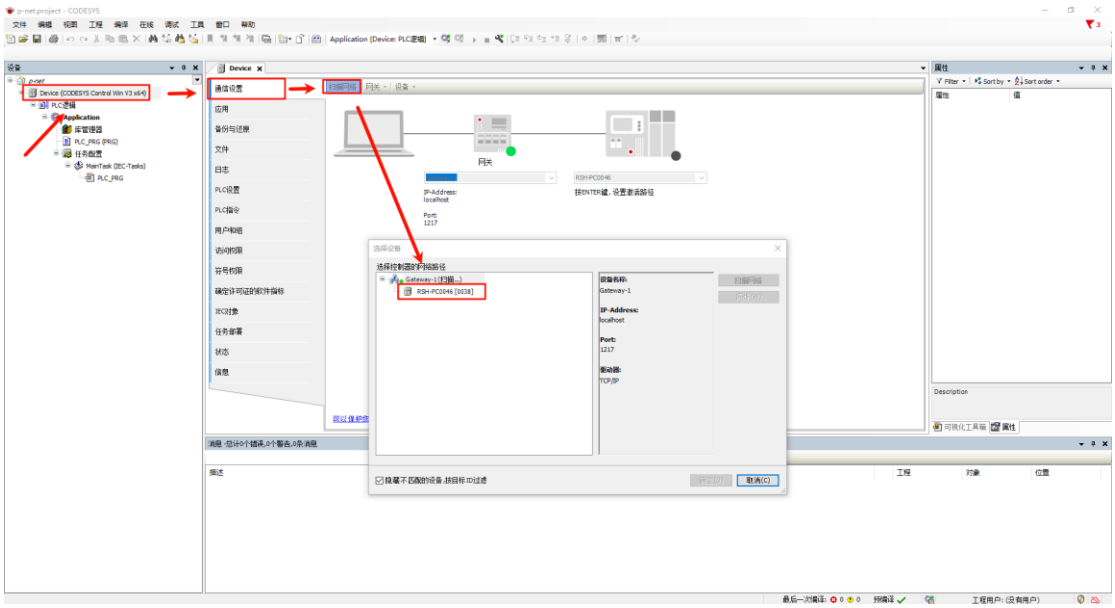


图 22-9 扫描网关设备

弹出设备用户登录窗口后，配置用户名和密码（用户自定义）：

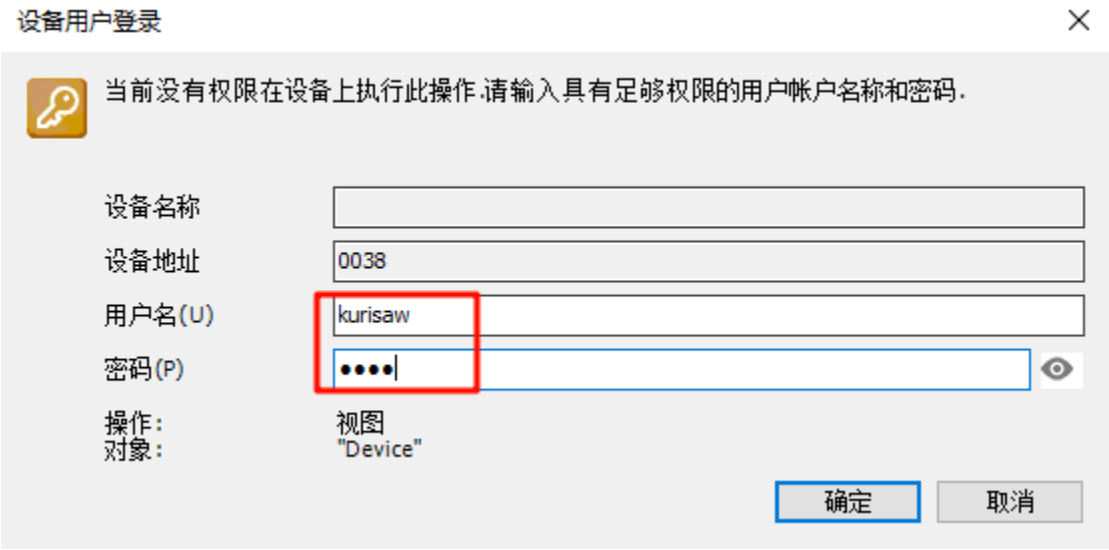


图 22-10 设备用户登录

检查网关设备及软 PLC 设备是否在线：

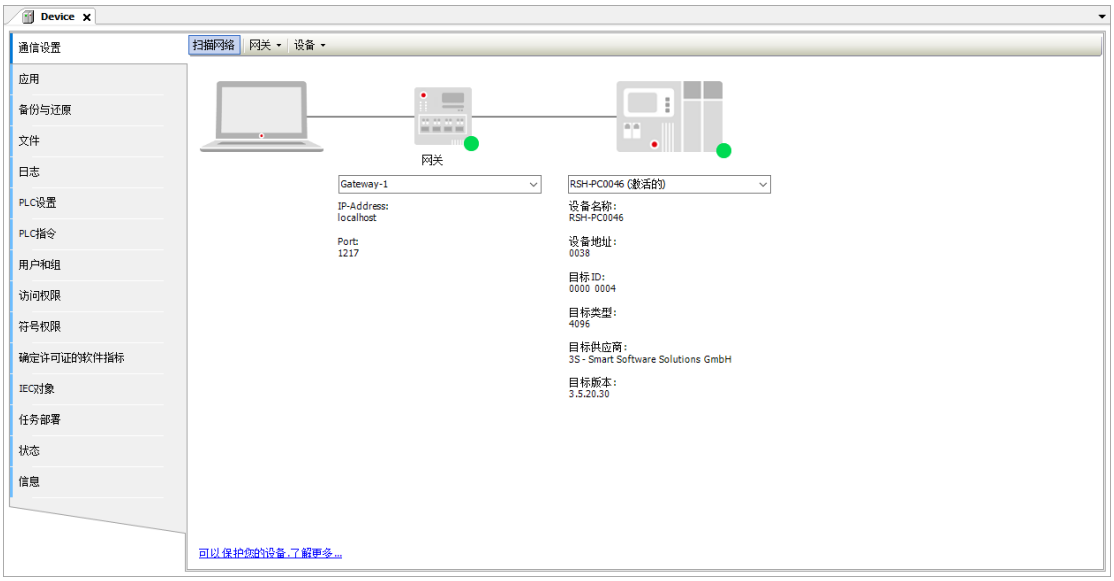


图 22-11 查看网关设备及软 PLC 设备在线

22.6.3 Ethernet/IP EDS 文件添加

EDS 文件（Electronic Data Sheet）是 **Ethernet/IP** 中用于描述设备特性和通信参数的标准文件格式。它包含了有关设备的详细信息，包括设备类型、支持的服务、输入输出的定义、参数设置、设备的状态和配置选项等。

本项目的 EDS 文件位于如下路径：

- ..\packages\OpENer_port-latest\eds_file

选择设备存储库安装描述文件，选择上述路径下的 opener_sample_app.eds 文件。

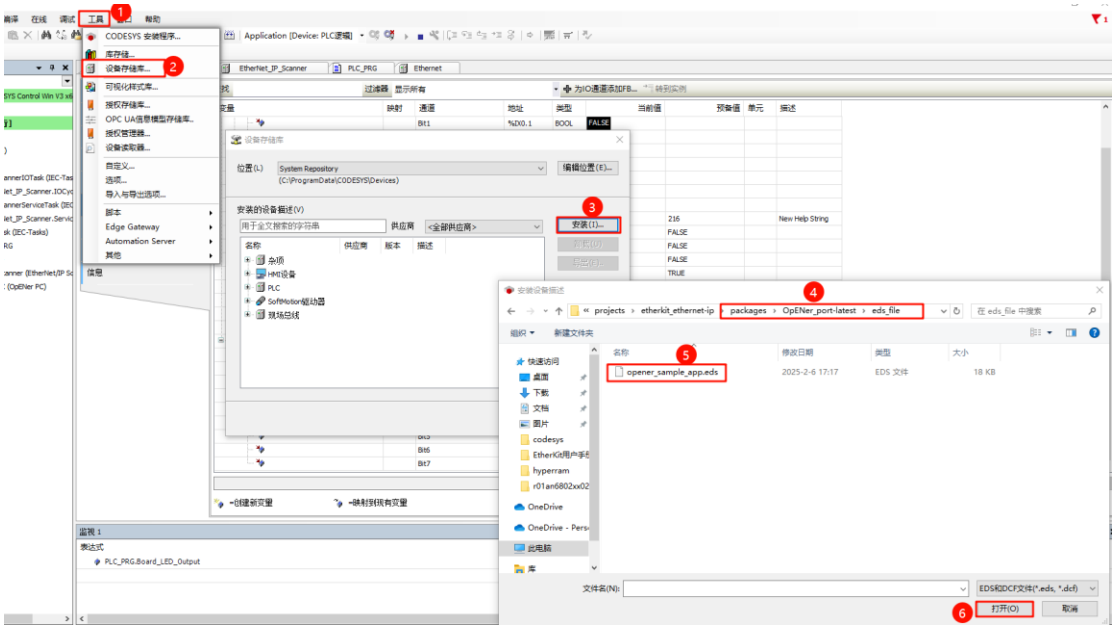


图 22-12 安装 eds 文件

安装成功后可以看到 OpENer PC 从站描述文件：

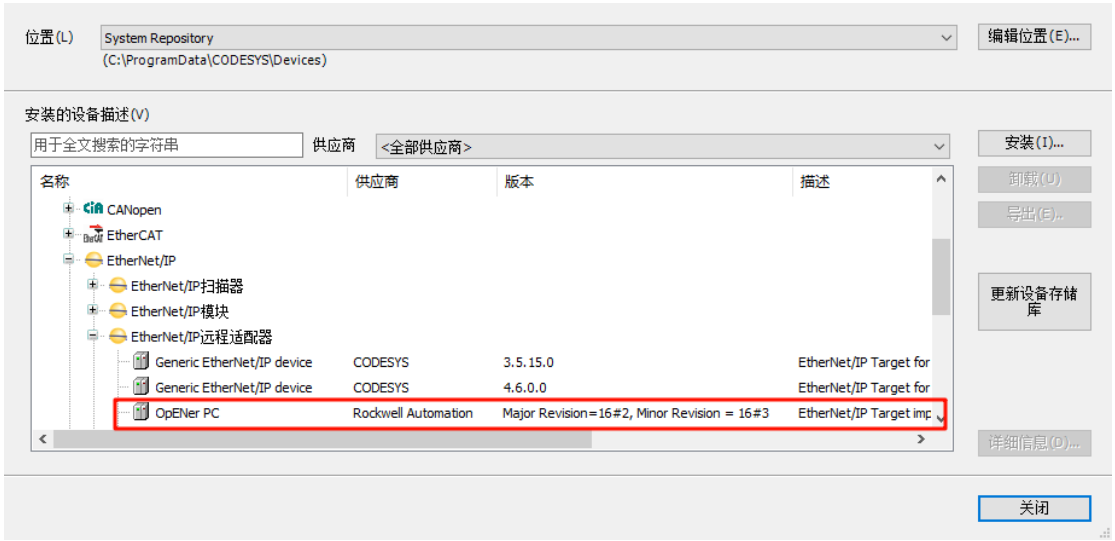


图 22-13 OpENer PC 文件安装

22.6.4 设备添加

- Ethernet 添加：左侧导航栏点击 Device 并右键添加设备，选择以太网适配器；

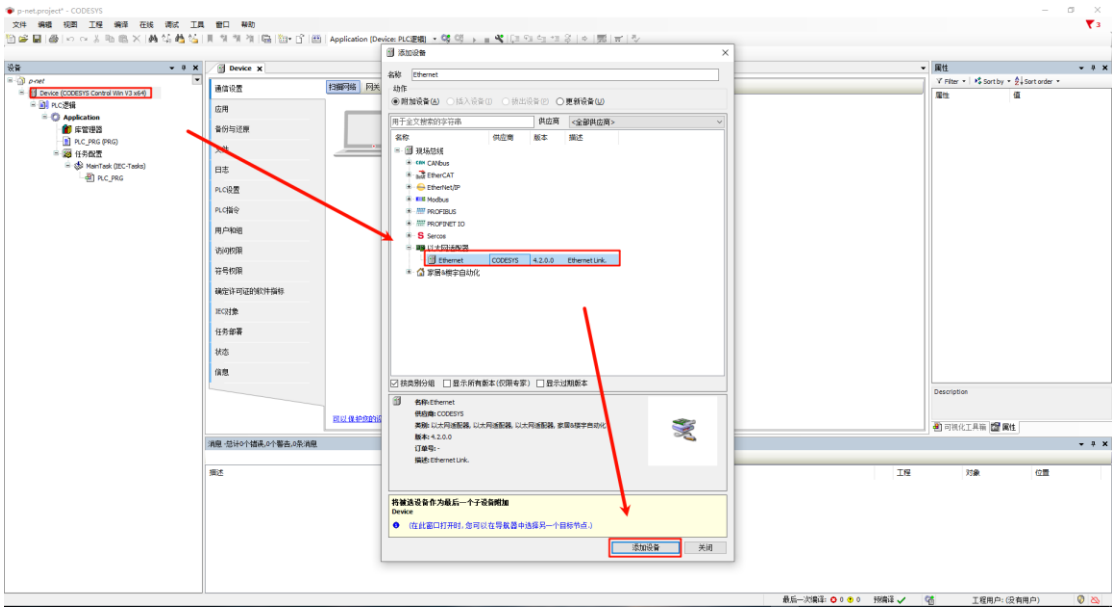


图 22-14 Ethernet 添加

- EtherNet/IP 扫描器添加：右键左侧导航栏中的 Ethernet，选择 EtherNet/IP Scanner

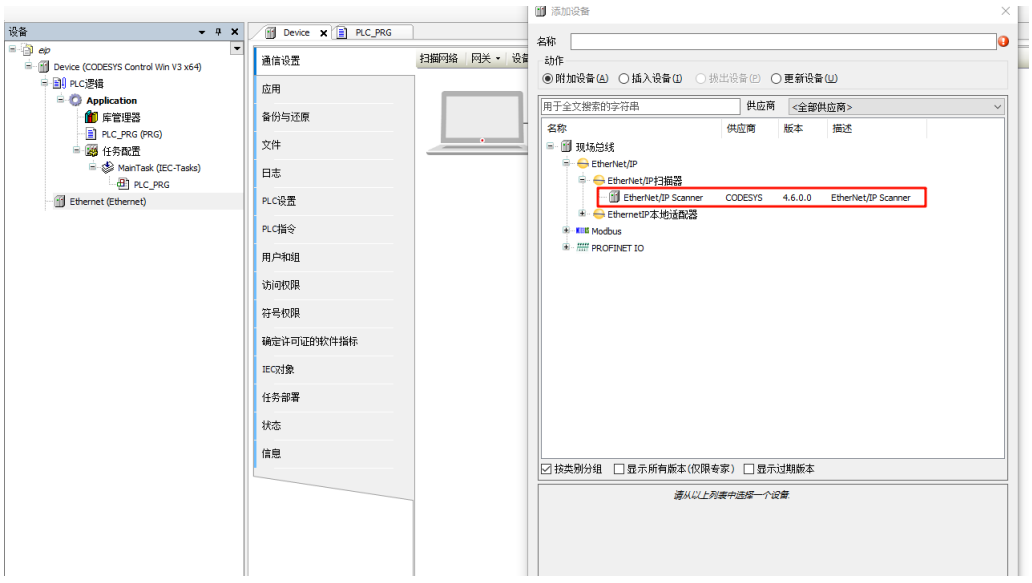


图 22-15 EtherNet/IP Scanner 添加

- EtherNet/IP 总线设备添加：右键左侧导航栏中的 EtherNet/IP Scanner，选择 OpENer PC

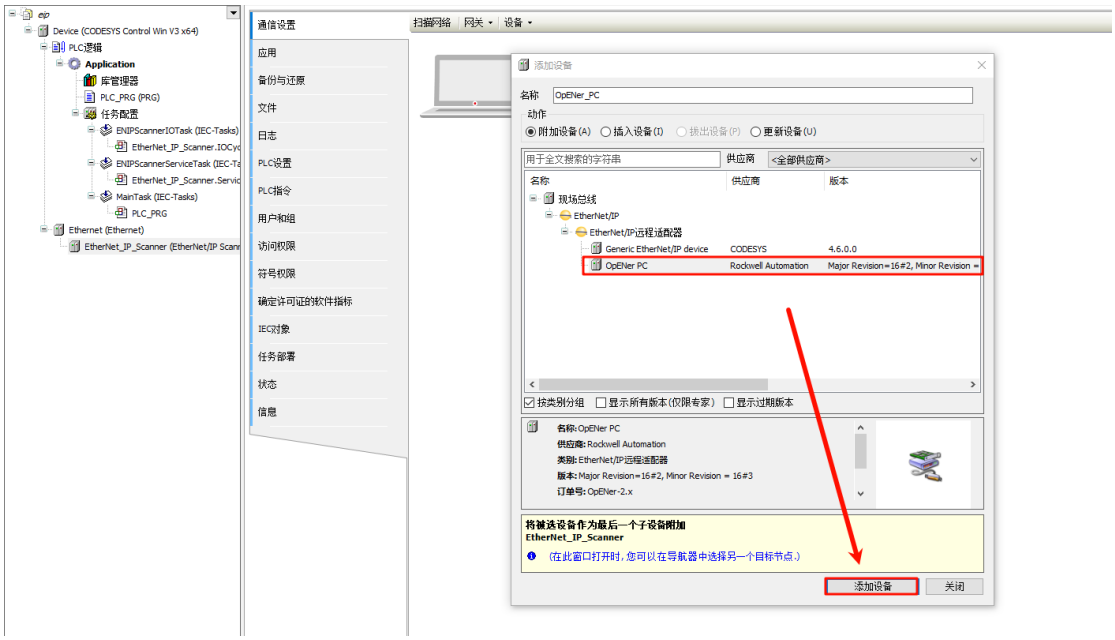


图 22-16 OpENer PC 设备添加

22.6.5 任务响应

保持默认配置即可。

22.6.6 网络配置

- Ethernet 配置：双击左侧导航栏中的 Ethernet(Ethernet) -> 通用，修改网络接口为连接到开发板的以太网端口；

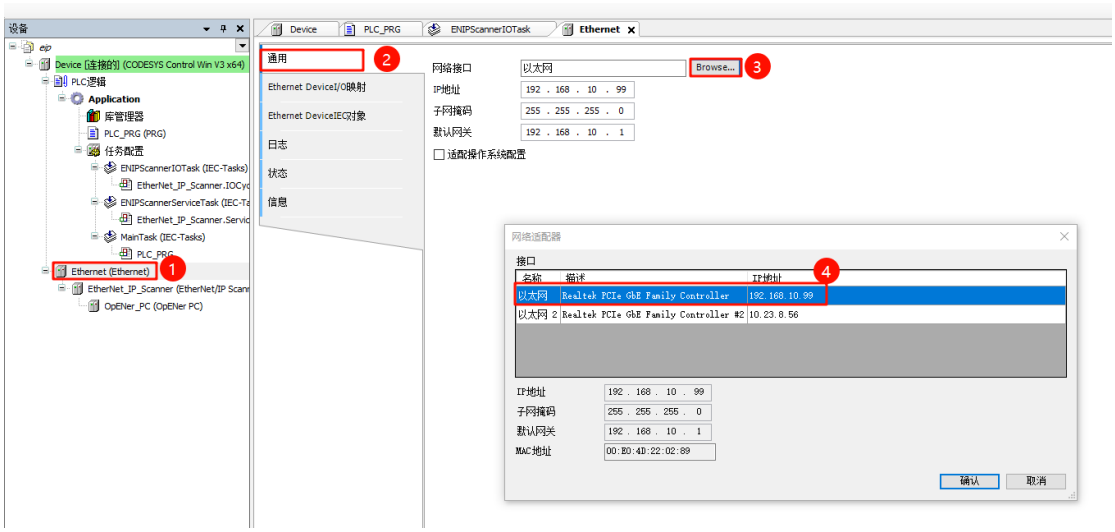


图 22-17 Ethernet 配置

- EtherNet/IP 总线设备网络配置：双击左侧导航栏 OpENer_PC(OpENer PC) -> 通用->地址设置， 修改 IP 参数为开发板 IP。

图 22-18 EtherNet/IP 总线设备网络配置

22.6.7 EtherNet/IP 线程应用启动

开发板端上电后，一旦检测到网卡 link up，则会自动启动 OpENer 线程：

```
msh />
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Feb  7 2025 14:55:26
 2006 - 2024 Copyright by RT-Thread team
lwIP-2.0.3 initialized!
[I/sal.skt] Socket Abstraction Layer initialize success.

Hello RT-Thread!
================================================
This example project is an Ethernet/IP routine!
================================================
msh />[I/DBG] link up
================================================
EtherNet/IP Bus device with OpENer Project!
================================================

msh />ps
thread           pri  status      sp     stack size max used left tick   error  tcb addr
---------------- ---  ------- ---------- ----------  ------  ---------- ------- ----------
OpENer            15  suspend 0x00000100 0x00002800    22%   0x0000000a ETIMOUT 0x1008b2e8
tshell            20  running 0x00000238 0x00001000    13%   0x00000001 OK      0x10087ef8
sys workq         23  suspend 0x00000078 0x00000800    22%   0x0000000a OK      0x10085fd0
tcpip             10  suspend 0x000000c8 0x00001000    16%   0x0000000a EINTRPT 0x10084ea8
etx               12  suspend 0x000000a4 0x00000400    16%   0x00000010 EINTRPT 0x1007536c
erx               12  suspend 0x000000a4 0x00000400    35%   0x0000000b EINTRPT 0x10074e64
tidle0            31  ready   0x00000048 0x00000400    10%   0x0000000f OK      0x100716ec
main              10  suspend 0x000000b0 0x00000800    18%   0x00000011 EINTRPT 0x10084510
msh />
```

图 22-19 OpENer 线程启动

22.6.8 工程编译并启动调试

- step1：工程上方导航栏选择 编译-> 生成代码
- step2：选择 在线 -> 登录
- step3：点击 调试 -> 启动

此时就可以看到 EtherNet/IP Scanner 已经正常运行了：

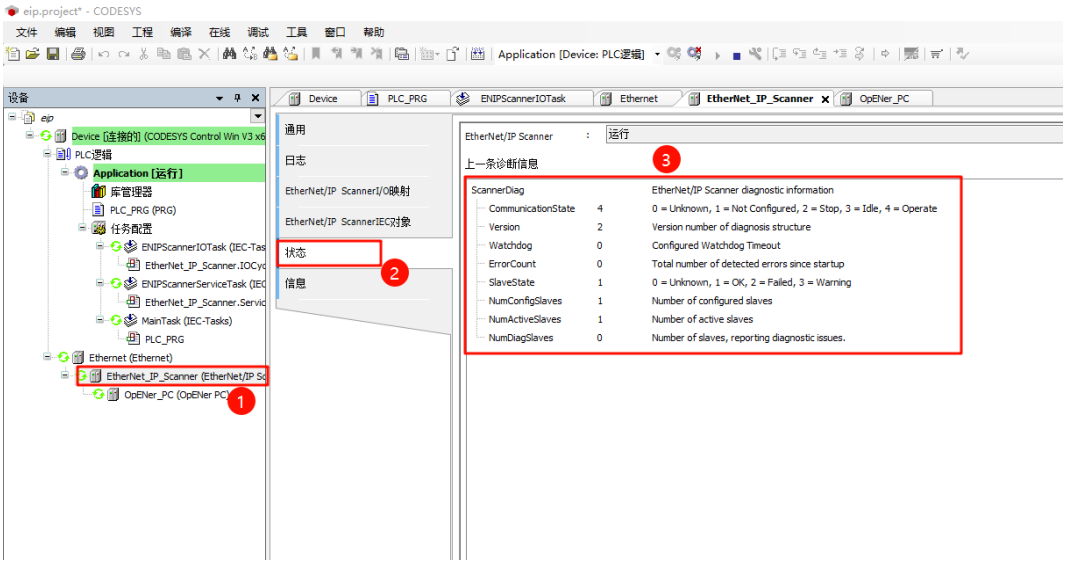


图 22-20 EtherNet/IP 总线设备运行正常

22.7 PLC 编程及 CIP IO 控制

首先我们点击左侧面板的 Device->PLC 逻辑->Application->PLC_PRG(PRG)，使用 ST 语言编程，编写变量及程序代码：

- 变量定义：下面这段变量中包含两个关键变量：Board_SW_Input（按 Bit 位标识控制器板载按键阵列）和 Board_LED_Output（按 Bit 位标识控制器板载 LED）。

```
PROGRAM PLC_PRG
VAR
    Board_SW_Input: BYTE;
    Board_LED_Output: BYTE;
    Mask: BYTE;
    Shift: INT;
    i: INT;
END_VAR
```

- 程序定义：这段代码的功能是：根据 Board_SW_Input 的每一位的状态，设置 Board_LED_Output 的相应位。具体来说：

  1. 如果 Board_SW_Input 的某一位为 1，则对应的 Board_LED_Output 的该位为 1。

  2. 如果 Board_SW_Input 的某一位为 0，则对应的 Board_LED_Output 的该位为 0。

  通过循环遍历所有 8 个位，实现了将输入的每一位状态映射到输出的每一位。

```
FOR i := 0 TO 7 DO
    Shift := i;
    Mask := SHL(1, Shift);

    IF (Board_SW_Input AND Mask) = Mask THEN
        Board_LED_Output := Board_LED_Output OR Mask;
    ELSE
        Board_LED_Output := Board_LED_Output AND NOT Mask;
    END_IF
END_FOR
```

工程中的配置位置如下图所示：

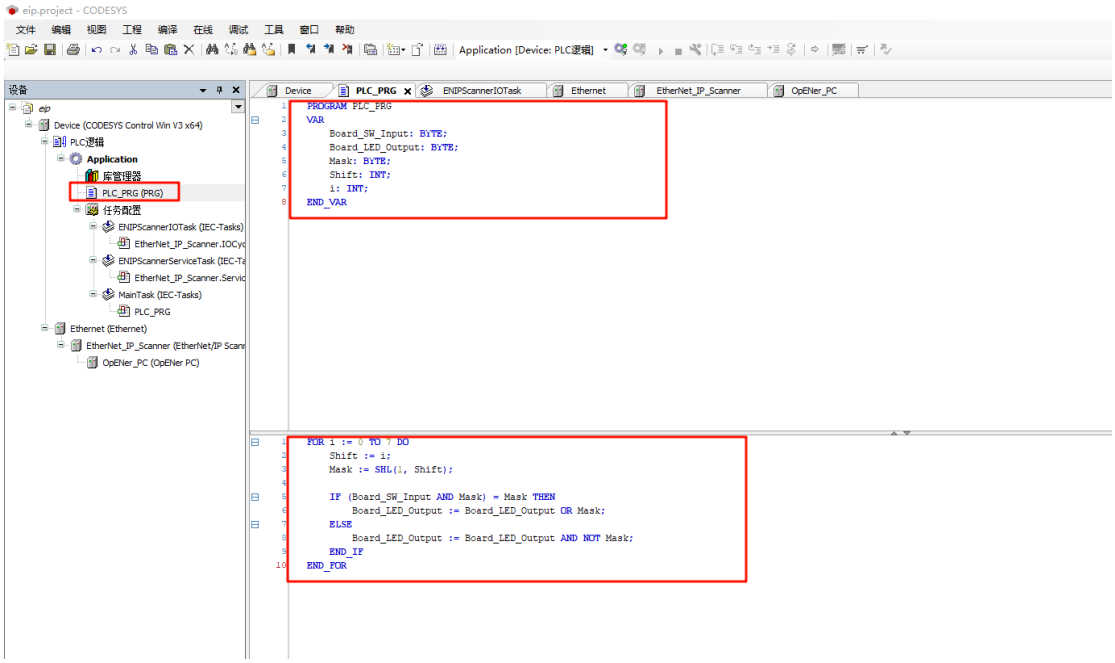


图 22-21 PLC 编程

由于加载 eds 文件后默认只会显示一个连接配置（Board LED Exclusive Owner），我们还需要将 eds 内置的另外一个配置加载出来，点击左侧菜单栏选择 OpENer_PC(OpENer PC)->连接，点击添加连接...，并选择 Board SW Input Only。

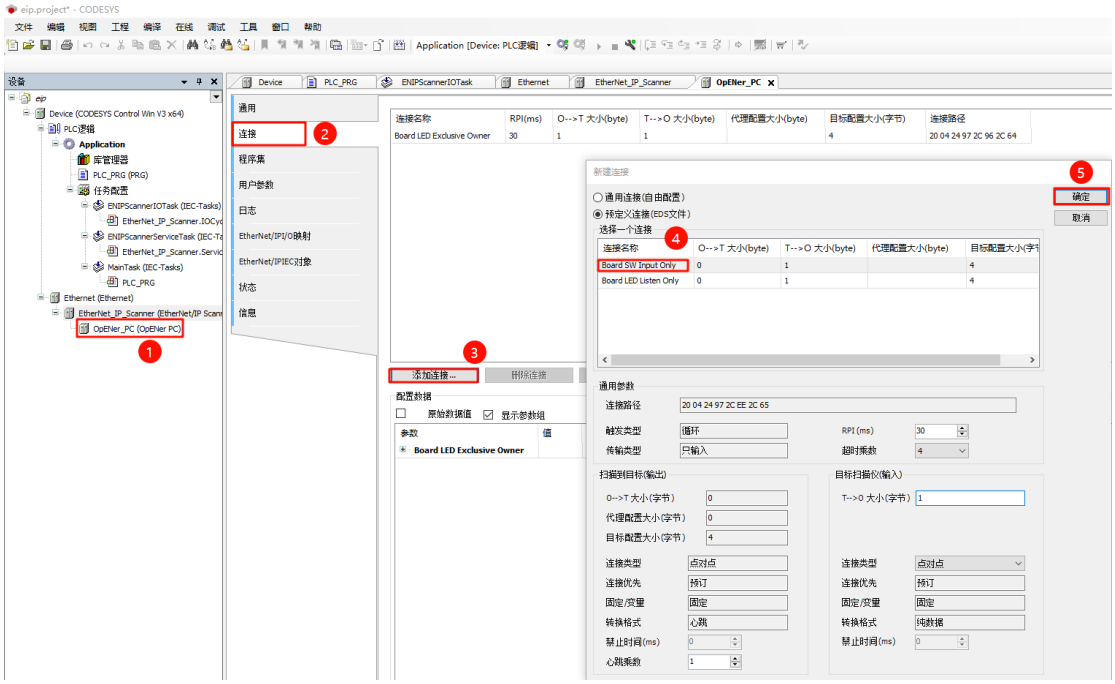


图 22-22 添加连接配置

接下来点击 Ethernet/IPI/O 映射，这里我们需要把前面定义的 ST 变量映射到此处的变量中，将 Board_LED_Output 映射到通道：Board LED Output Data；Board_SW_Input 映射到通道：Board SE Input Data。

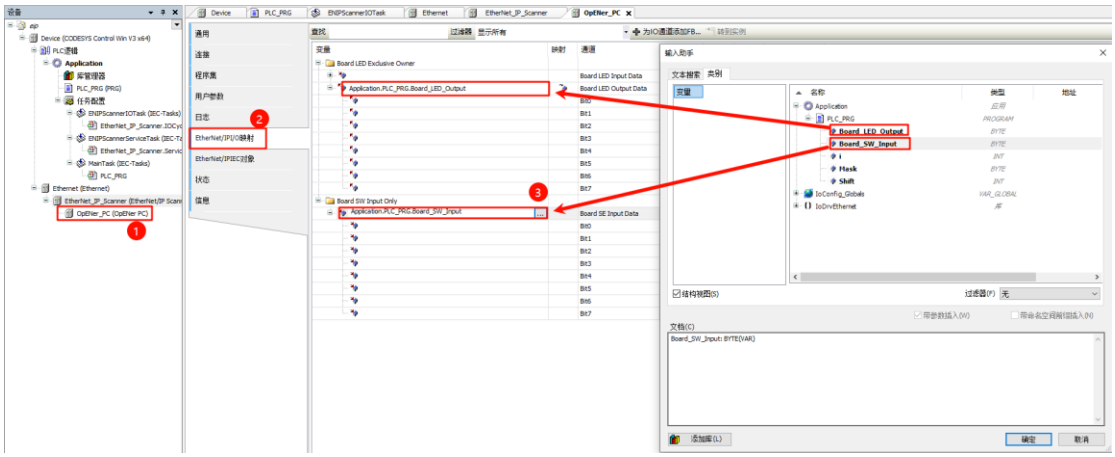


图 22-23 IO 映射编辑

接着我们点击上方导航栏的编译->生成代码，然后选择在线->登录，此时便可动态观察程序运行状态，例如我们按住 etherkit 开发板上的 KEY1，可以发现板载 LED0（红灯）处于灭灯状态，当我们松开 KEY1，LED0 保持常亮；按住开发板的 KEY2，板载 LED2（绿灯）处于灭灯状态，松开 KEY2，LED2 保持常亮。

同时在 OpENer_PC(OpENer PC)->EtherNet/IPI/O 映射也可以观察 Bit 位的当前值，当对应按键的 Bit 位为 TRUE 时，即代表按键按下，同时对应的 Bit 位 LED 亮起，并显示当前值为 TRUE：

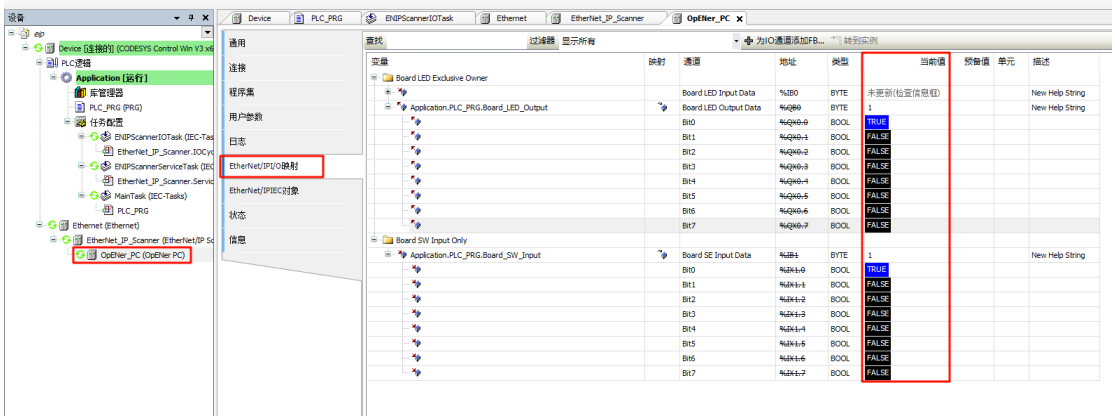


图 22-24 IO 映射测试

第 23 章 FAQ

本章节主要对用户使用 EtherKit 的开发过程中遇到的一些问题进行收集与解答。

23.1 芯片状态异常

当芯片无法正常下载程序，但是使用 J-Link Commander 可以正常读取到芯片状态时，大概率此时芯片已经处于异常状态，通过 BOOT 重置可解决此问题：

首先安装 Segger J-Link 驱动，推荐版本为 V7.98a；然后将板载拨码开关全部切换到 OFF 档（全都拨下去），如下图所示位置：

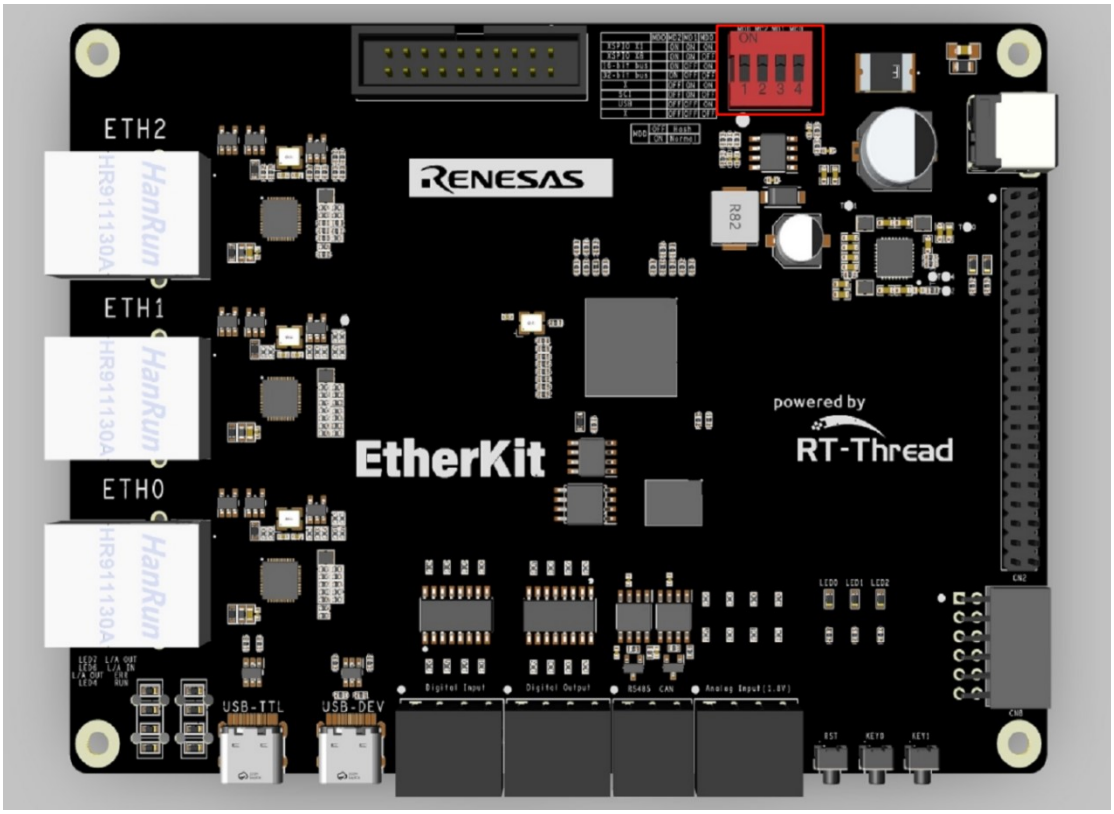


图 23-1 拨码开关

打开 Segger J-Link Commander，依次按下图所示操作；

```
SEGGER J-Link Commander V7.98a (Compiled Jul 19 2024 15:01:50)
DLL version V7.98a, compiled Jul 19 2024 15:01:03

Connecting to J-Link via USB...O.K.
Firmware: J-Link Vll compiled Jun 21 2023 09:20:55
Hardware version: V11.00
J-Link uptime (since boot): 0d 00h 02m 56s
S/N: 601012427
License(s): RDI, FlashBP, FlashDL, JFlash, GDB
USB speed mode: High speed (480 MBit/s)
VTref=3.251V


Type "connect" to establish a target connection, '?' for help
J-Link>connect                                          1.输入connect连接
Please specify device / core. <Default>: R9A07G084M04
Type '?' for selection dialog
Device>
Please specify target interface:                        2.默认配置回车即可
  J) JTAG (Default)
  S) SWD
  T) cJTAG
TIF>S                                                   3.输入S，选择SWD
Specify target interface speed [kHz]. <Default>: 4000 kHz
Speed>
Device "R9A07G084M04" selected.                         4.回车
```

图 23-2 连接 Jlink commander

下面执行如下命令清除 flash，如下所示：

```
exec EnableEraseAllFlashBanks
erase 0x60000000 0x63FFFFFF
```

```
  EL3 support: N/A
  FPU support: Single + Double + Conversion
ARMv8-A/R: The connected J-Link (S/N 601012427) uses an old firmware module V9 with known problems / limitations.
Add. info (CPU temp. halted)
Current exception level: EL1
Exception level AArch usage:
  EL0: AArch32
  EL1: AArch32
  EL2: AArch32
  EL3: AArch32
Non-secure status: Non-secure
Cache info:
  Inner cache boundary: none
  LoU Uniprocessor: 1
  LoC: 1
  LoU Inner Shareable: 1
I-Cache L1: 16 KB, 64 Sets, 64 Bytes/Line, 4-Way
D-Cache L1: 16 KB, 64 Sets, 64 Bytes/Line, 4-Way
Memory zones:
  Zone: "Default" Description: Default access mode
  Zone: "AP0" Description: MEM-AP (APB-AP)
  Zone: "AP1" Description: MEM-AP (APB-AP)
  Zone: "AP2" Description: MEM-AP (AXI-AP)
Cortex-R52 identified.
J-Link>exec EnableEraseAllFlashBanks
J-Link>erase 0x60000000 0x63FFFFFF
'erase': Performing implicit reset & halt of MCU.
ResetTarget() start
Reset: Halt core immediately after reset using reset catch.
Authenticated device detected. Skipping authentication process.
  OCDREG_STATUS: 0x00000001
Disabled core power domain detected.
Enabling debug mode...
ResetTarget() end - Took 251ms
Erasing selected range...
```

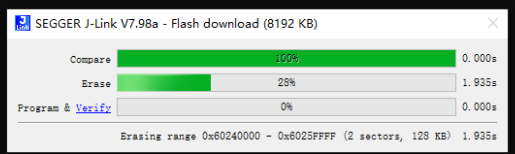


图 23-3 清除 flash

Flash 清除完成之后，先将开发板**断电**并将拨码全部切换为 ON 档后，再重

新上电即可再次下载程序。

23.2 Studio/IAR 调试断点无法停住

此次进入工程的如下路径并打开该文件：..\ rzn\fsp\src\bsp\cmsis\Device\RENESAS\Source\cr\startup_core.c，在 system_init()这个函数首行添加如下代码：

```
#if 1
    __asm volatile (
        "    mov   r0, #0                              \n"
        "    movw  r1, #0xf07f                      \n"
        "    movt  r1, #0x2fa                         \n"
        "software_loop:                               \n"
        "    adds  r0, #1                                  \n"
        "    cmp   r0, r1                                  \n"
        "    bne   software_loop                    \n"
        ::: "memory");
#endif
```

如下图所示，这段汇编功能主要实现了一段延时，在系统启动时加上这段延时，可方便调试进入断点处停下：

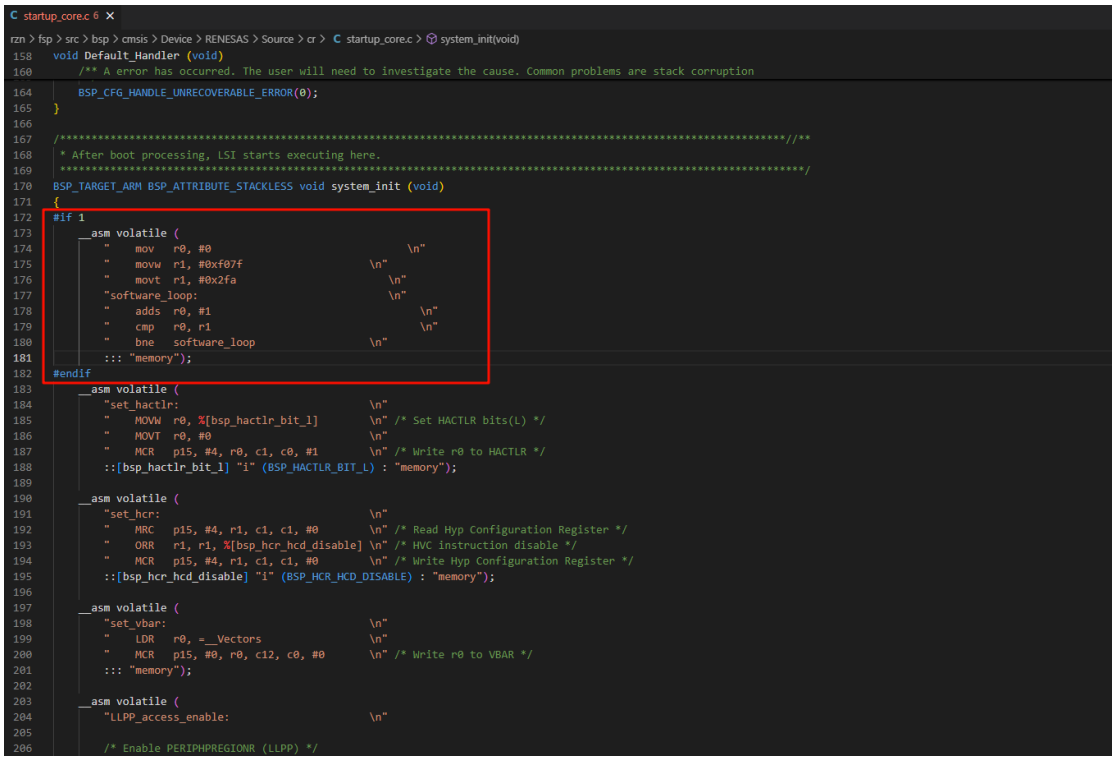


图 23-4 延时函数