EtherKit SDK

Documentation

The EtherKit Development Board is RT-Thread's R9A07G084 chip based on Renesas Cortex-R52 architecture, providing engineers with a flexible and comprehensive development platform to help developers gain a deeper experience in EtherCAT industrial Ethernet.

Version 1.0.0 | English | September 08, 2025

www.rt-thread.org

Contents

1. Getting Started

1.1. EtherKit Development Board BSP Documentation

English | **Chinese**

Introduction

This document provides the BSP (Board Support Package) documentation for the RT-Thread EtherKit development board. By following the Quick Start section, developers can quickly get started with this BSP and run RT-Thread on the development board.

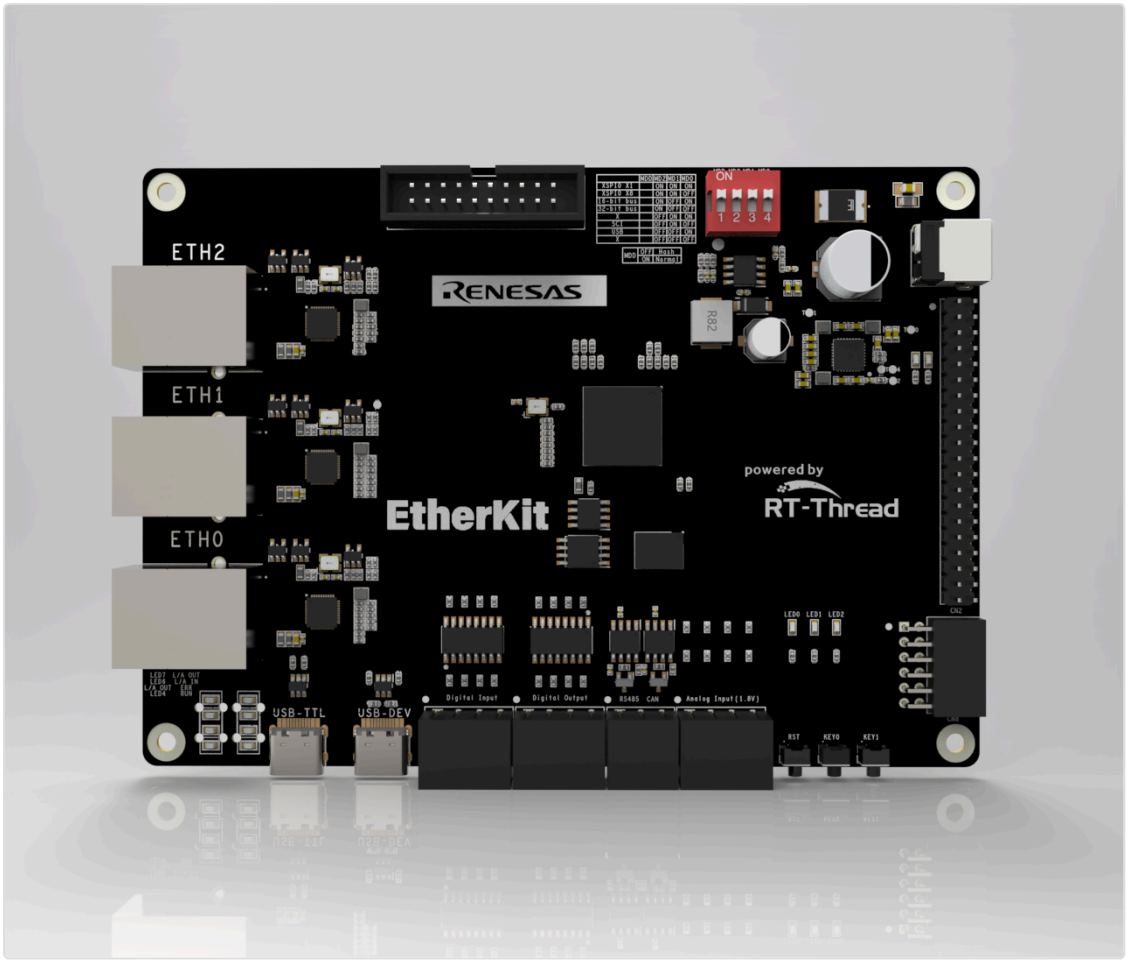
The main contents are as follows:

- Introduction to the Development Board
- BSP Quick Start Guide

Introduction to the Development Board

The EtherKit development board is based on the Renesas RZ/N2L and is designed to facilitate embedded system application development by offering flexible software package and IDE configurations.

The front view of the development board is shown below:



Key **onboard resources** include:

- MPU: R9A07G084M04GBG, maximum operating frequency of 400MHz, Arm Cortex®-R52 core, 128KB tightly coupled memory (with ECC), 1.5MB internal RAM (with ECC)
- Debug Interface: Onboard J-Link interface
- Expansion Interface: One PMOD connector

More detailed information and tools

Peripheral Support

This BSP currently supports the following peripherals:

Here is the translated text in English, keeping the markdown format:

EtherCAT Solution	**Support Status**	**EtherCAT Solution**	**Support Status**
EtherCAT_IO	Supported	EtherCAT_FOE	Supported
EtherCAT_EOE	Supported	EtherCAT_COE	Supported
PROFINET Solution	**Support Status**	**Ethernet/IP Solution**	**Support Status**
P-Net (Open source evaluation package supporting ProfiNET slave protocol stack)	Supported	EIP	Supported
On-chip Peripherals	**Support Status**	**Components**	**Support Status**
UART	Supported	LWIP	Supported
GPIO	Supported	TCP/UDP	Supported
HWIMER	Supported	MQTT	Supported
IIC	Supported	TFTP	Supported
WDT	Supported	Modbus Master/Slave Protocol	Supported
RTC	Supported		
ADC	Supported		
DAC	Supported		
SPI	Supported		

Usage Instructions

Usage instructions are divided into two sections:

- **Quick Start**

This section is designed for beginners who are new to RT-Thread. By following simple steps, users can run the RT-Thread OS on the development board and observe the experimental results.

- **Advanced Usage**

This section is for developers who need to use more of the development board's resources within the RT-Thread OS. By configuring the BSP using the ENV tool, additional onboard resources and advanced features can be enabled.

Quick Start

This BSP currently provides GCC/IAR project support. Below is a guide using the IAR Embedded Workbench for Arm development environment to run the system.

Hardware Connection

Connect the development board to the PC via a USB cable. Use the J-Link interface to download and debug the program.

Compilation and Download

- Navigate to the `bsp` directory and use the command `scons --target=iar` to generate the IAR project.
- Compile: Double-click the `project.eww` file to open the IAR project and compile the program.
- Debug: In the IAR navigation bar, click `Project -> Download and Debug` to download and start debugging.

Viewing the Run Results

After successfully downloading the program, the system will automatically run and print system information.

Connect the corresponding serial port of the development board to the PC. Open the relevant serial port (115200-8-1-N) in the terminal tool. After resetting the device, you can view the RT-Thread output. Enter the `help` command to see the list of supported system commands.

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Mar 14 2024 18:26:01
 2006 - 2024 Copyright by RT-Thread team
```

```
Hello RT-Thread!
================================================
This is an IAR project in RAM execution mode!
================================================
msh > help
RT-Thread shell commands:
clear            - clear the terminal screen
version          - show RT-Thread version information
list             - list objects
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 >
```

Application Entry Function

The entry function for the application layer is located in **src\hal_entry.c** within `void hal_entry(void)`. User source files can be placed directly in the `src` directory.

```
void hal_entry(void)
{
    rt_kprintf("\nHello RT-Thread!\n");
    rt_kprintf("================================================
    rt_kprintf("This is an IAR project in RAM execution mode!\n
    rt_kprintf("================================================

    while (1)
    {
        rt_pin_write(LED_PIN, PIN_HIGH);
        rt_thread_mdelay(500);
        rt_pin_write(LED_PIN, PIN_LOW);
        rt_thread_mdelay(500);
    }
}
```

Advanced Usage

Resources and Documentation

- Development Board Official Homepage
- Development Board Datasheet

- Development Board Hardware Manual
- RZ/N2L MCU Quick Start Guide
- RZ/N2L Easy Download Guide
- Renesas RZ/N2L Group

FSP Configuration

To modify Renesas BSP peripheral configurations or add new peripheral ports, the Renesas FSP configuration tool is required. Please follow the steps outlined below for configuration. For any questions regarding the configuration, please visit the RT-Thread Community Forum.

1. Download the Flexible Software Package (FSP) | Renesas, use FSP version 2.0.0.
2. To add the **"EtherKit Board Support Package"** to FSP, refer to the document How to Import a BSP.
3. For guidance on configuring peripheral drivers using FSP, refer to the document: Configuring Peripheral Drivers Using FSP for RA Series.

ENV Configuration

- To learn how to use the ENV tool, refer to the RT-Thread ENV Tool User Manual.

By default, this BSP only enables the UART0 functionality. To use more advanced features such as components, software packages, and more, the ENV tool must be used for configuration.

The steps are as follows:

1. Open the ENV tool in the `bsp` directory.
2. Use the `menuconfig` command to configure the project. Save and exit once the configuration is complete.
3. Run the `pkgs --update` command to update the software packages.
4. Run the `scons --target=iar` command to regenerate the project.

Contact Information

If you have any thoughts or suggestions during usage, please feel free to contact us via the RT-Thread Community Forum.

Contribute Code

If you're interested in EtherKit and have some exciting projects you'd like to share, we welcome code contributions. Please refer to How to Contribute to RT-Thread Code.

2. Basics

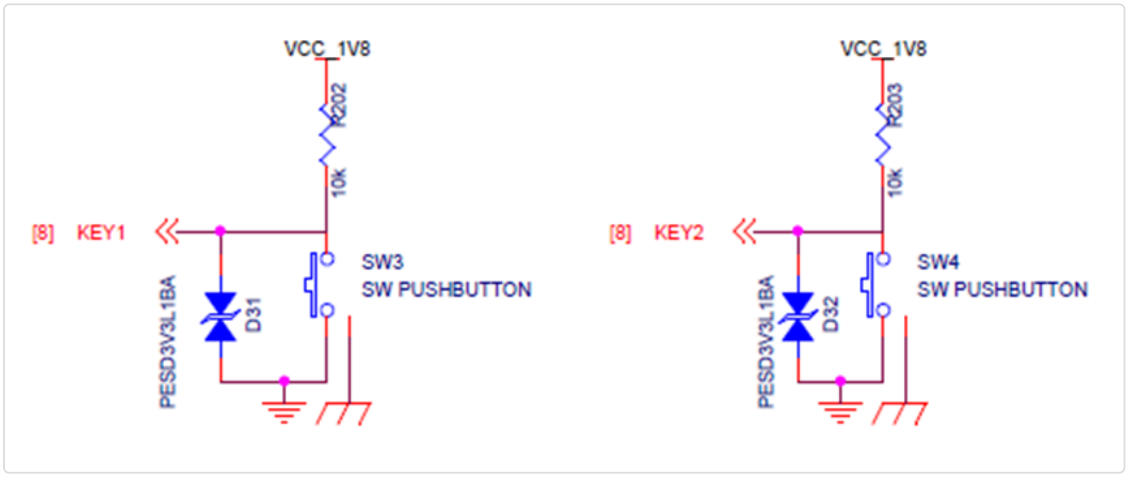
2.1. Button Interrupt Usage Instructions

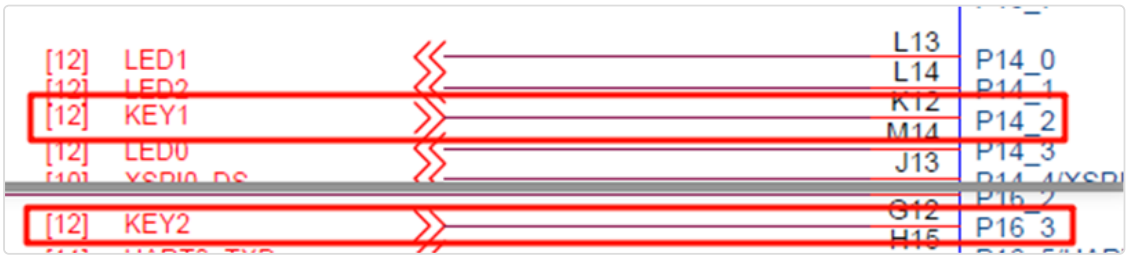
English | **中文**

Introduction

This example demonstrates how to use the onboard button (KEY) to trigger an external interrupt. When a specified KEY is pressed, related information is printed, and the corresponding LED is activated.

Hardware Description

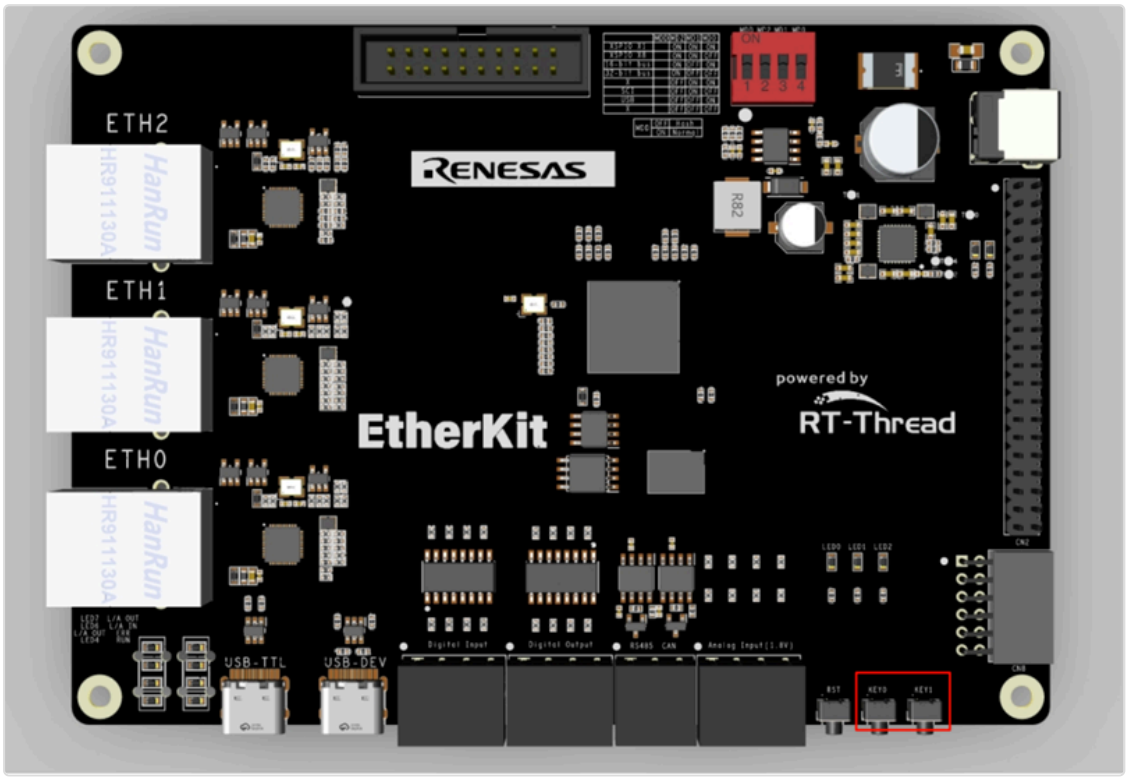




As shown in the diagram above, KEY1 (LEFT) and KEY2 (RIGHT) are connected to MCU pins P14_2 (LEFT) and P16_3 (RIGHT), respectively.

Pressing the KEY button generates a high signal, and releasing it generates a low signal.

The positions of the keys on the development board are shown in the following diagram:

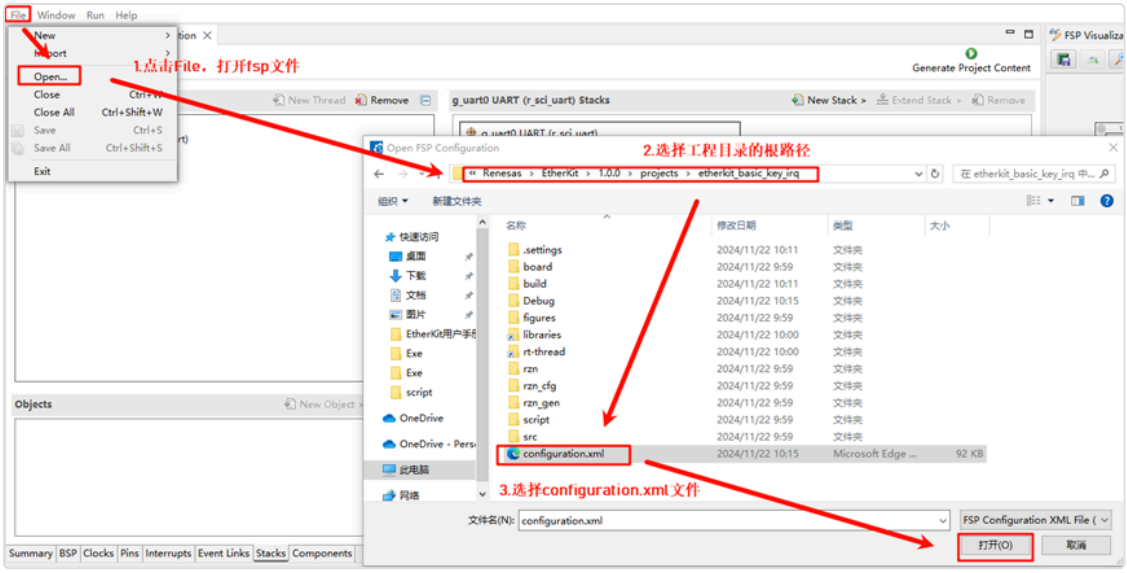


FSP Configuration

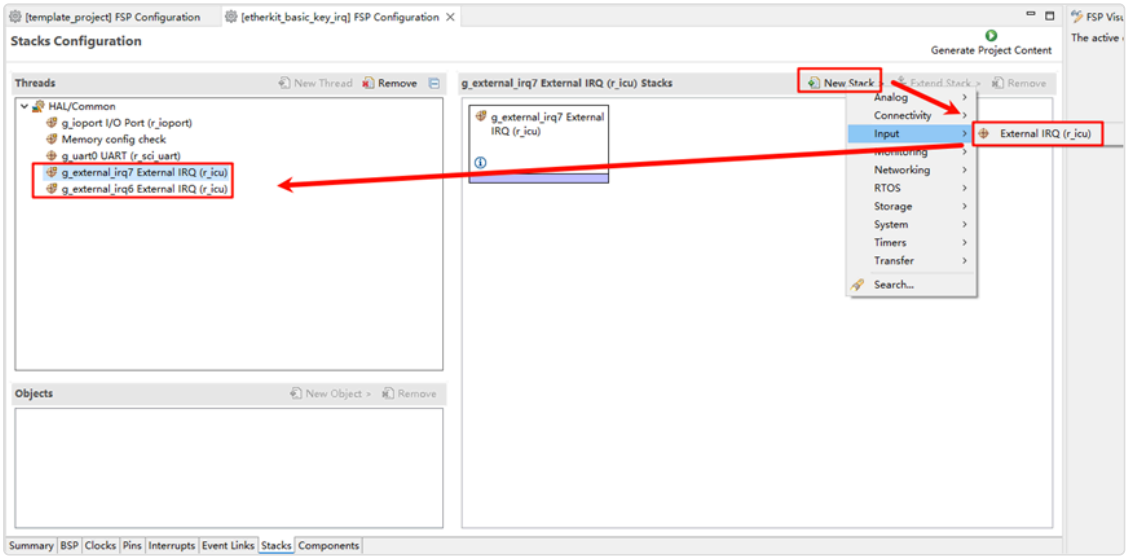
First, download the official FSP code generation tool:

- https://github.com/renesas/rzn-fsp/releases/download/v2.0.0/setup_rznfsp_v2_0_0_rzsc_v2024-01.1.exe

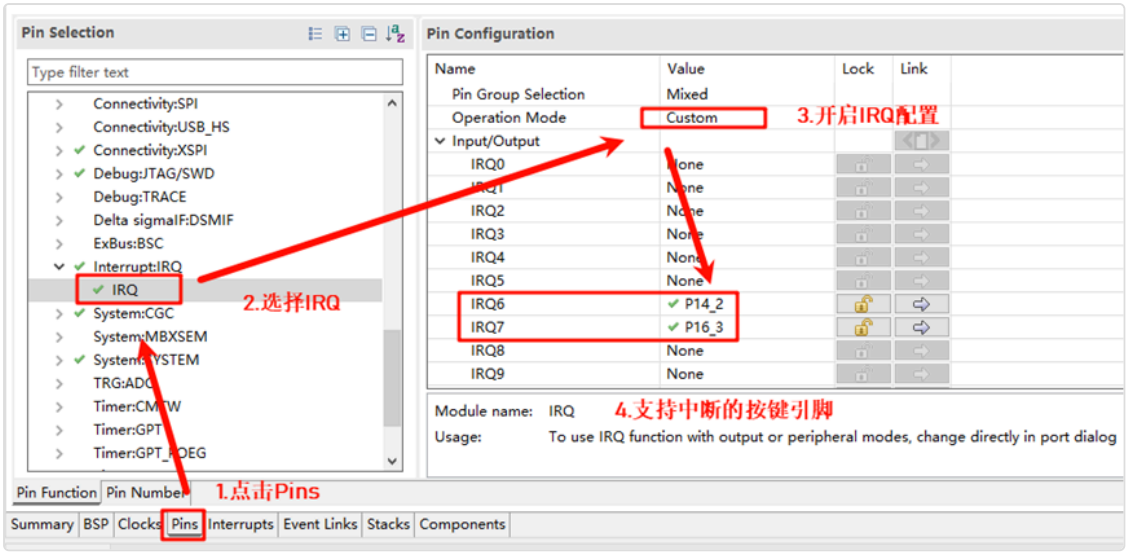
Once installed, double-click `rasc.exe` in Eclipse and open the project configuration file `configuration.xml` as shown in the following image:



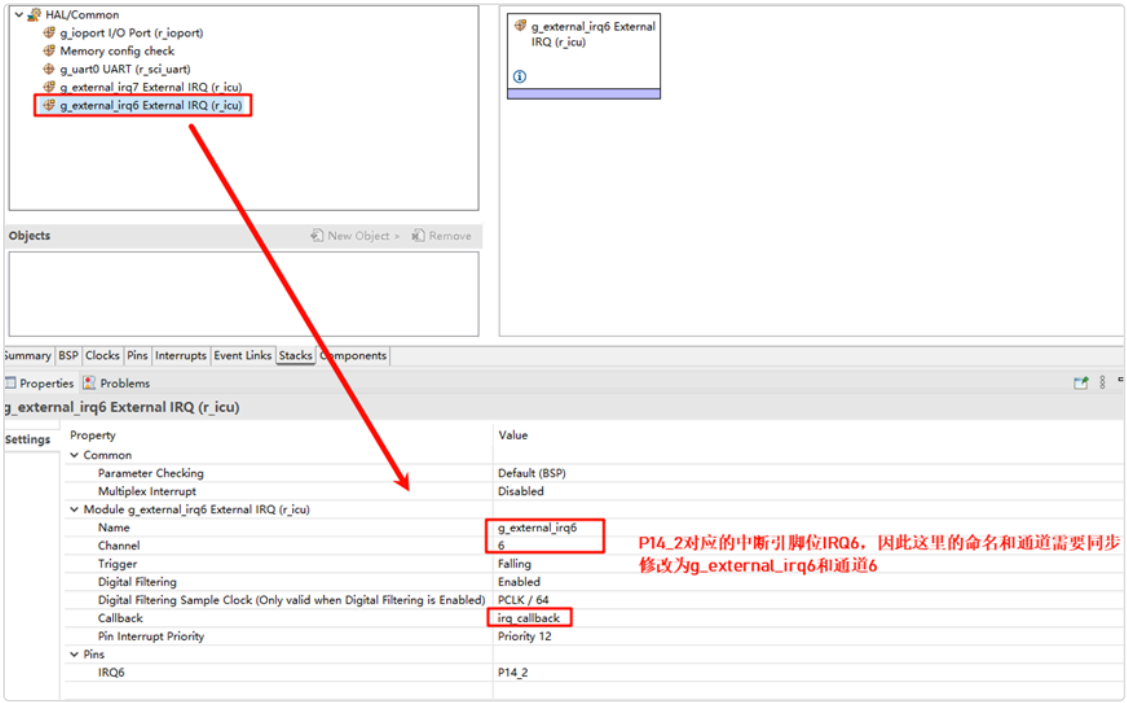
Next, add two stacks: New Stack -> Input -> External IRQ (r_icu):



Now, enable the IRQ functionality in the pin configuration. Select the two interrupt pins to be enabled: KEY1 (IRQ6) and KEY2 (IRQ7):



Return to the Stacks interface and configure IRQ6 and IRQ7 by specifying the interrupt names, channel numbers, and callback functions:



Example Code Description

The source code for this example is located at `/projects/etherkit_basic_key_irq`.

The MCU pin definitions for KEY1 (LEFT) and KEY2 (RIGHT) are as follows:

```
/* Configure key IRQ pins */

#define IRQ_TEST_PIN1 BSP_IO_PORT_14_PIN_2
#define IRQ_TEST_PIN2 BSP_IO_PORT_16_PIN_3
```

The MCU pin definitions for the LED are as follows:

```
/* Configure LED pins */
#define LED_PIN_B    BSP_IO_PORT_14_PIN_0 /* Onboard BLUE LED p
#define LED_PIN_G    BSP_IO_PORT_14_PIN_1 /* Onboard GREEN LED
```

The source code for the button interrupt is located at `/projects/etherkit_basic_key_irq/src/hal_entry.c`. When the corresponding interrupt button is pressed, it triggers the printing of related information.

```
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("================================================
    rt_kprintf("This example project is a basic key IRQ routine
    rt_kprintf("================================================

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

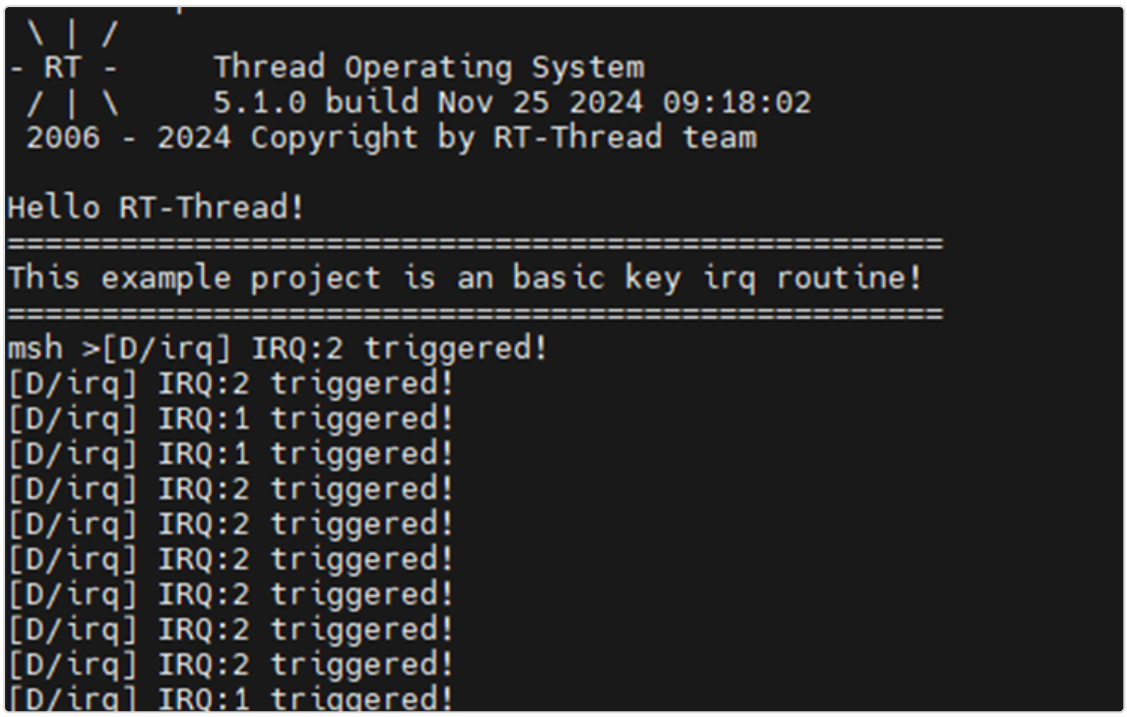
Compilation & Download

- **RT-Thread Studio**: In RT-Thread Studio's package manager, download the EtherKit resource package, create a new project, and compile it.
- **IAR**: First, double-click `mklinks.bat` to create symbolic links between RT-Thread and the libraries folder. Then, use the `Env` tool to generate the IAR project. Finally, double-click `project.eww` to open the IAR project and compile it.

After compiling, connect the development board's JLink interface to the PC and download the firmware to the development board.

Run Effect

After pressing the reset button to restart the development board, the initial state has both LED1 and LED2 turned off. When KEY1 is pressed, LED1 (Blue) will light up; when KEY2 is pressed, LED2 (Green) will light up.


\ | /
- RT -   Thread Operating System
/ | \   5.1.0 build Nov 25 2024 09:18:02
2006 - 2024 Copyright by RT-Thread team
Hello RT-Thread!
This example project is an basic key irq routine!
msh >[D/irq] IRQ:2 triggered!
[D/irq] IRQ:2 triggered!
[D/irq] IRQ:1 triggered!
[D/irq] IRQ:1 triggered!
[D/irq] IRQ:2 triggered!
[D/irq] IRQ:2 triggered!
[D/irq] IRQ:2 triggered!
[D/irq] IRQ:2 triggered!
[D/irq] IRQ:2 triggered!
[D/irq] IRQ:2 triggered!
[D/irq] IRQ:1 triggered!

2.2. RTC and Alarm Usage Instructions

English | **中文**

Introduction

This example demonstrates how to use the RTC (Real-Time Clock) on the EtherKit. RTC provides accurate real-time information, such as the year, month, day, hour, minute, and second. Most real-time clock chips use high-precision crystal oscillators as clock sources. Some RTC chips have a battery backup to keep time information valid when the main power supply is off.

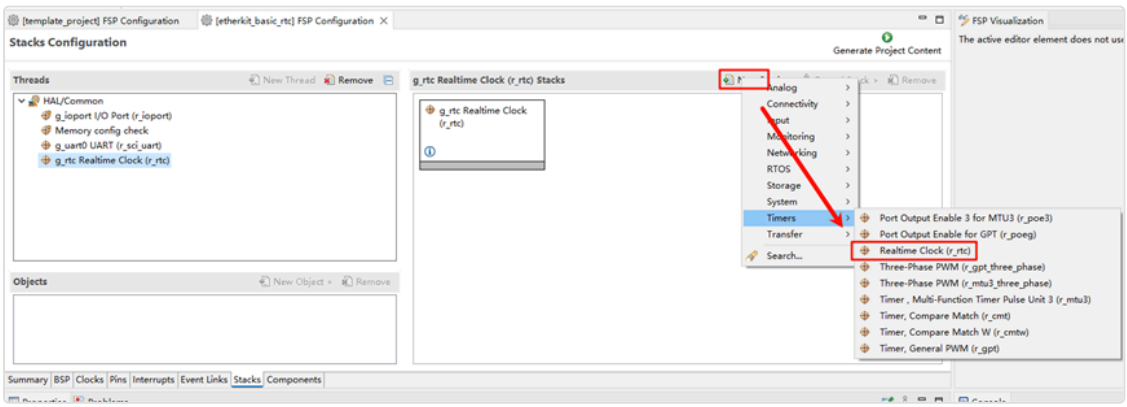
RT-Thread's RTC device provides foundational services for the operating system's time system. With the increasing number of IoT scenarios, RTCs have become standard in products, and are even indispensable in secure communication protocols like SSL.

Hardware Description

The RTC device used in this example relies on the LSE (Low-Speed External) clock. No other special connections are required.

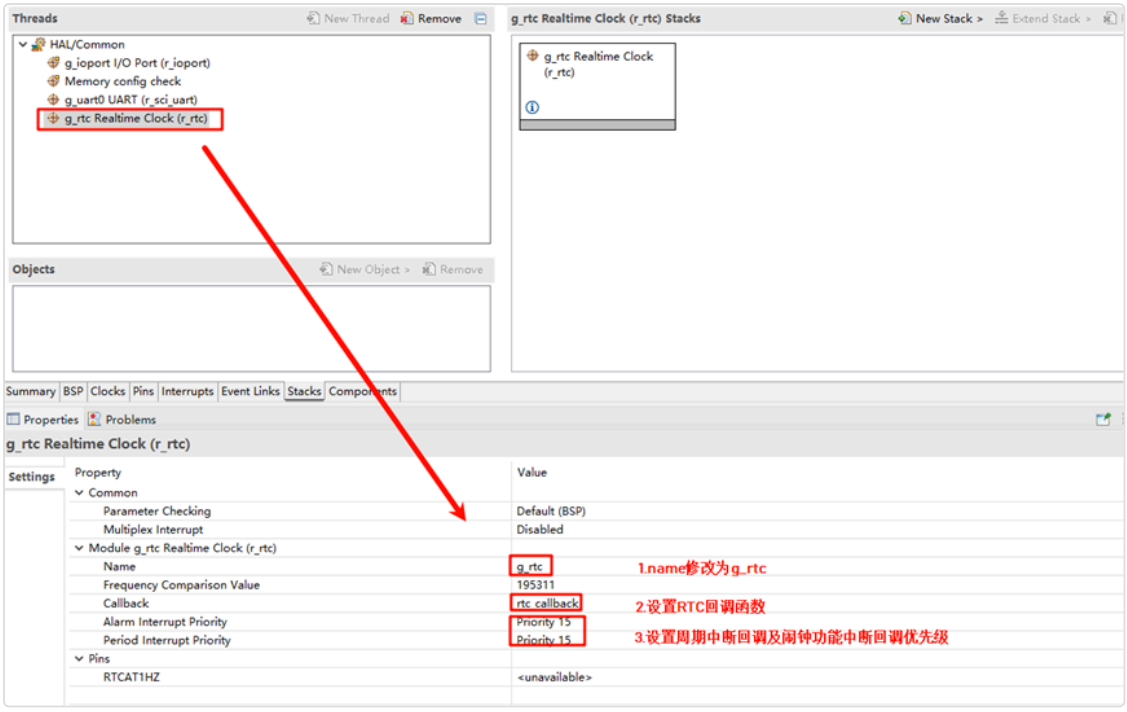
FSP Configuration Instructions

Open the FSP configuration tool and select the `configuration.xml` file under the corresponding project directory. Add the RTC Stack:



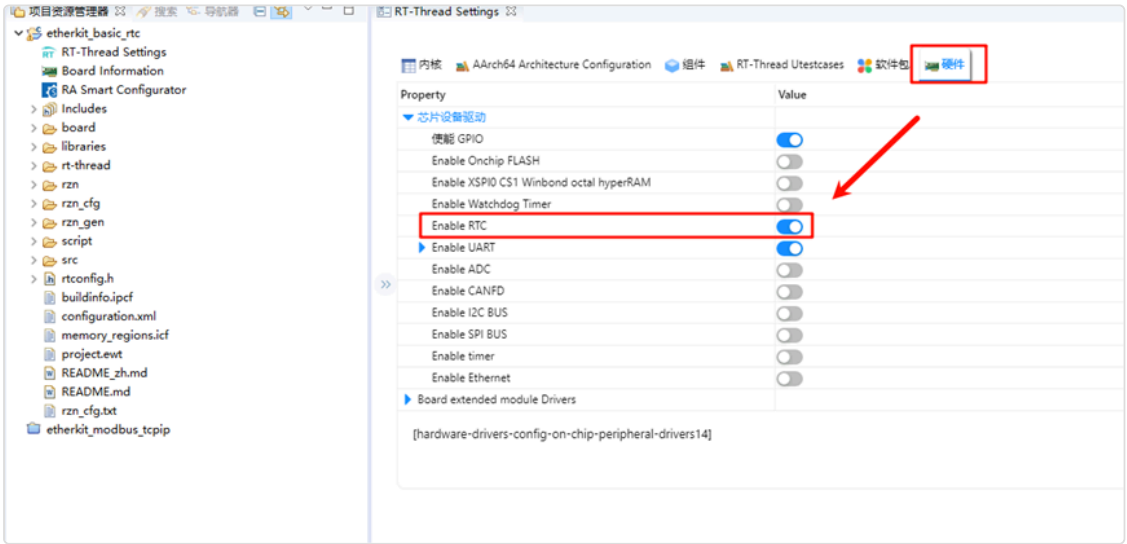
Next, configure the RTC parameters. Set the RTC stack name to `g_rtc`, configure the RTC interrupt callback function to `rtc_callback`, and set the

interrupt callback priority:

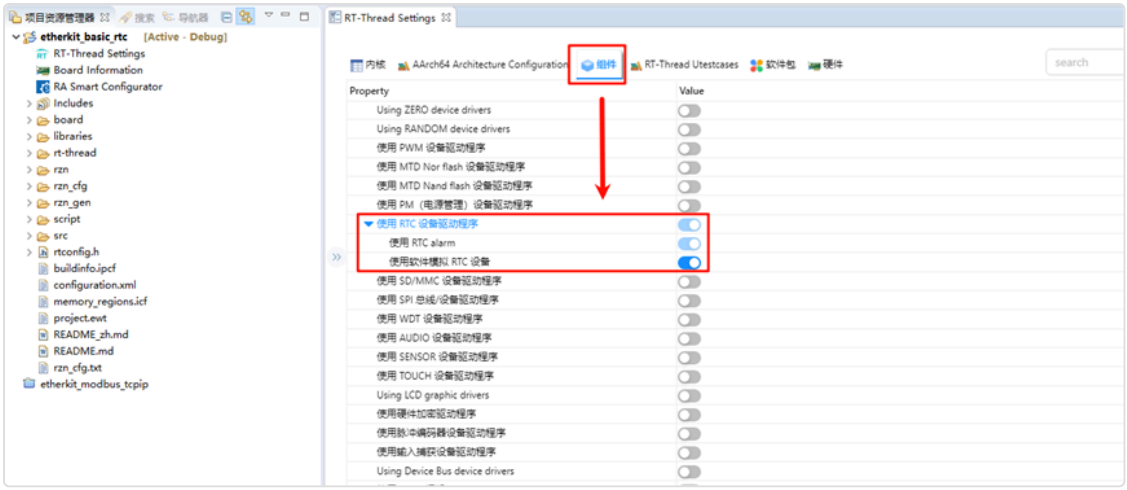


RT-Thread Settings Configuration

Open RT-Thread Settings and enable the RTC under the hardware options:



Next, configure the RTC by enabling RT-Thread's RTC device framework and the software alarm feature. (Note: Currently, the alarm feature in Renesas RZ series has some issues, so the alarm function is simulated with software, but this does not affect its usage):



Example Code Description

The source code for this example is located at `/projects/etherkit_basic_rtc`. In the `hal_entry()` function, the RTC device is accessed, the system time is set, and then the system time is retrieved to check if it has been set correctly. After a 1-second delay, the system time is retrieved again.

```
    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);
    }

    /* Set date */
    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");
    }

    /* Set time */
```

```
    ret = set_time(15, 00, 00);
    if (ret != RT_EOK)
    {
        rt_kprintf("set RTC time failed\n");
    }

    /* Delay for 3 seconds */
    rt_thread_mdelay(3000);

    /* Get time */
    get_timestamp(&now);
    rt_kprintf("now: %.*s", 25, ctime(&now));
```

The following code creates an RTC alarm and sets it to trigger in 1 second. Finally, the function is exported to the MSH command line.

```
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;

    /* Get the current timestamp and set the alarm to trigger i
    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);
```

Compilation & Download

- **RT-Thread Studio**: In RT-Thread Studio's package manager, download the EtherKit resource package, create a new project, and compile it.
- **IAR**: First, double-click `mklinks.bat` to create symbolic links between RT-Thread and the libraries folder. Then, use the `Env` tool to generate the IAR project. Finally, double-click `project.eww` to open the IAR project and compile it.

After compilation, connect the development board's JLink interface to the PC and download the firmware to the development board.

Run Effect

After pressing the reset button to restart the development board, you should see the following printed information on the board:

```
 \ | /
- 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.
```

2.3. RGB Usage Instructions

English | **中文**

Introduction

This example is the first and simplest example in the SDK, similar to the "Hello World" program that programmers encounter as their first program. The main function of this example is to make the onboard RGB-LED blink periodically.

Hardware Description

As shown in the diagram above, the EtherKit provides three user LEDs: LED0 (RED), LED1 (BLUE), and LED2 (GREEN). The RED LED corresponds to pin P14_3 on the MCU. To turn on the LED, a low level signal is output from the MCU pin, and to turn it off, a high level signal is output.

The positions of the LEDs on the development board are shown in the diagram below:

Software Description

The source code for this example is located in `/projects/etherkit_blink_led`. The MCU pin definitions for the RGB LEDs and the RGB control logic can be found in `src/hal_data.c`.

```
/* Configure LED pins */
#define LED_PIN_R    BSP_IO_PORT_14_PIN_3 /* Onboard RED LED pi
#define LED_PIN_B    BSP_IO_PORT_14_PIN_0 /* Onboard BLUE LED p
#define LED_PIN_G    BSP_IO_PORT_14_PIN_1 /* Onboard GREEN LED
    do
```

```
    {
        /* Get the group number */
        group_current = count % group_num;

        /* Control RGB LEDs */
        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]);

        /* Output LOG information */
        LOG_D("group: %d | red led [%-3.3s] | | blue led [%-3.3
            group_current,
            _blink_tab[group_current][0] == LED_ON ? "ON" : "OF
            _blink_tab[group_current][1] == LED_ON ? "ON" : "OF
            _blink_tab[group_current][2] == LED_ON ? "ON" : "OF

        count++;

        /* Delay for a short time */
        rt_thread_mdelay(500);
    } while(count > 0);
```

Compilation & Download

- **RT-Thread Studio**: In RT-Thread Studio's package manager, download the EtherKit resource package, create a new project, and compile it.
- **IAR**: First, double-click `mklinks.bat` to create symbolic links between RT-Thread and the libraries folder. Then, use the `Env` tool to generate the IAR project. Finally, double-click `project.eww` to open the IAR project and compile it.

After compilation, connect the development board's JLink interface to the PC and download the firmware to the development board.

Run Effect

After pressing the reset button to restart the development board, observe the actual effect of the RGB-LED on the development board. When the program is running normally, the RGB LEDs will change periodically, as shown in the image below:

At this point, you can also open the default serial port of the development board using a terminal tool on the PC, with the baud rate set to 115200N. The board's runtime log information will be output in real-time.

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

3. Drivers

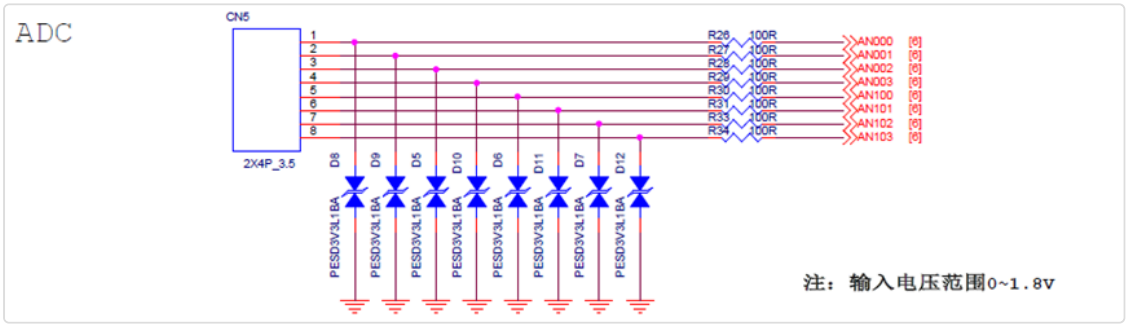
3.1. ADC Driver Usage Instructions

English | **中文**

Introduction

This example demonstrates how to use the RT-Thread ADC framework on the EtherKit to collect analog signals via ADC and perform digital signal conversion. The main content includes the following:

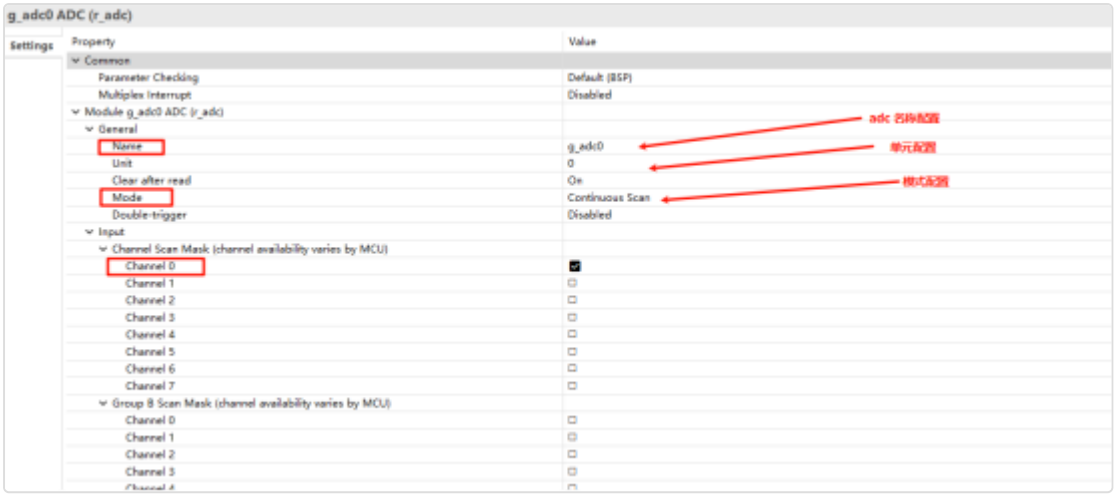
Hardware Description



As shown in the schematic above, the EtherKit provides an 8-channel analog input interface, which is connected to the MCU's ADC channels 0, 1, 2, and 3 for ADC0 and ADC1. (Note: The voltage range for the Analog Input is 0 to 1.8V.)

FSP Configuration Instructions

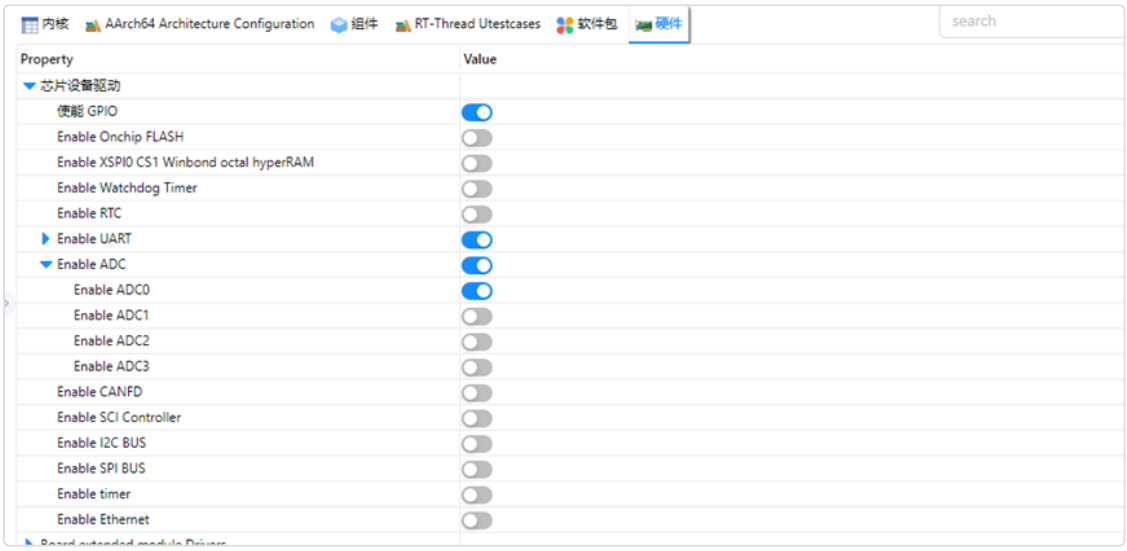
- Step 1: Open FSP and import the XML configuration file (or directly click the FSP link file in RT-Thread Studio).
- Step 2: Create a new `r_adc` stack, configure the ADC device, and select the channels to be used.



- Step 3: Save and click "Generate Project". The generated code will be saved in `hal_data.c`.

ENV Configuration

Use the `env` tool to enable ADC0 peripheral:



Example Code Description

The ADC source code is located in `/projects/etherkit_driver_adc/src/hal_entry.c`. The macros used are defined as follows:

```
#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 */
```

The specific function is to sample the analog voltage from ADC0 channel 0 every 1000ms and perform a conversion. The code is as follows:

```
static int adc_vol_sample()
{
    rt_adc_device_t adc_dev;
    rt_uint32_t value, vol;
    rt_err_t ret = RT_EOK;
    /* Find the device */
    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
        return RT_ERROR;
    }
    /* Enable the device */
    ret = rt_adc_enable(adc_dev, ADC_DEV_CHANNEL);
    /* Read the sampled value */
    value = rt_adc_read(adc_dev, ADC_DEV_CHANNEL);
    rt_kprintf("The value is :%d \n", value);
    /* Convert to corresponding voltage */
    vol = value * REFER_VOLTAGE / CONVERT_BITS;
    rt_kprintf("The voltage is :%d.%02d \n", vol / 100, vol % 1
    /* Disable the channel */
    ret = rt_adc_disable(adc_dev, ADC_DEV_CHANNEL);
    return ret;
}
```

In the example, the `while` loop calls `adc_vol_sample` every 1000ms.

Compilation & Download

- **RT-Thread Studio**: In RT-Thread Studio's package manager, download the EtherKit resource package, create a new project, and compile it.
- **IAR**: First, double-click `mklinks.bat` to create symbolic links between RT-Thread and the libraries folder. Then, use the `Env` tool to generate the IAR project. Finally, double-click `project.eww` to open the IAR project and compile it.

After compilation, connect the development board's JLink interface to the PC and download the firmware to the development board.

Run Effect

The effect when using ADC0 channel 0 to sample a 1.8V voltage is shown below:

```
 \ | /
- 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
```

Notes

> *The ADC voltage input tolerance for the R9A07G084M08GBG chip is 1.8V.*

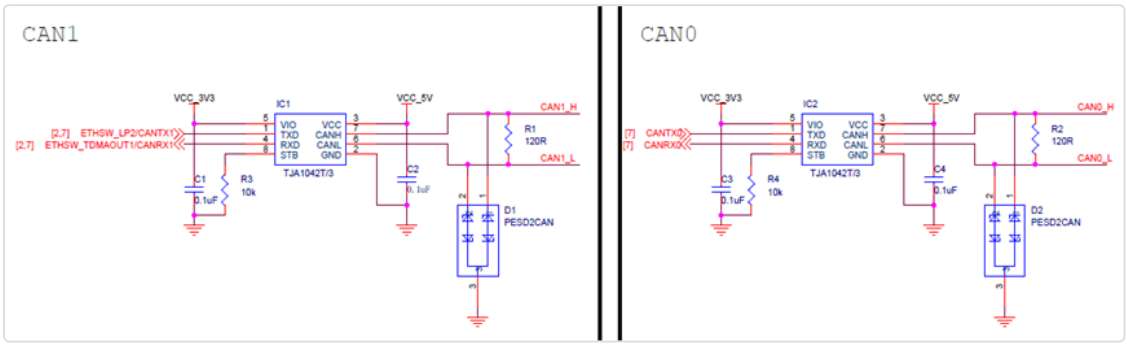
3.2. CANFD Driver Usage Instructions

English | **Chinese**

Introduction

This tutorial mainly introduces how to use the CANFD device on the EtherKit.

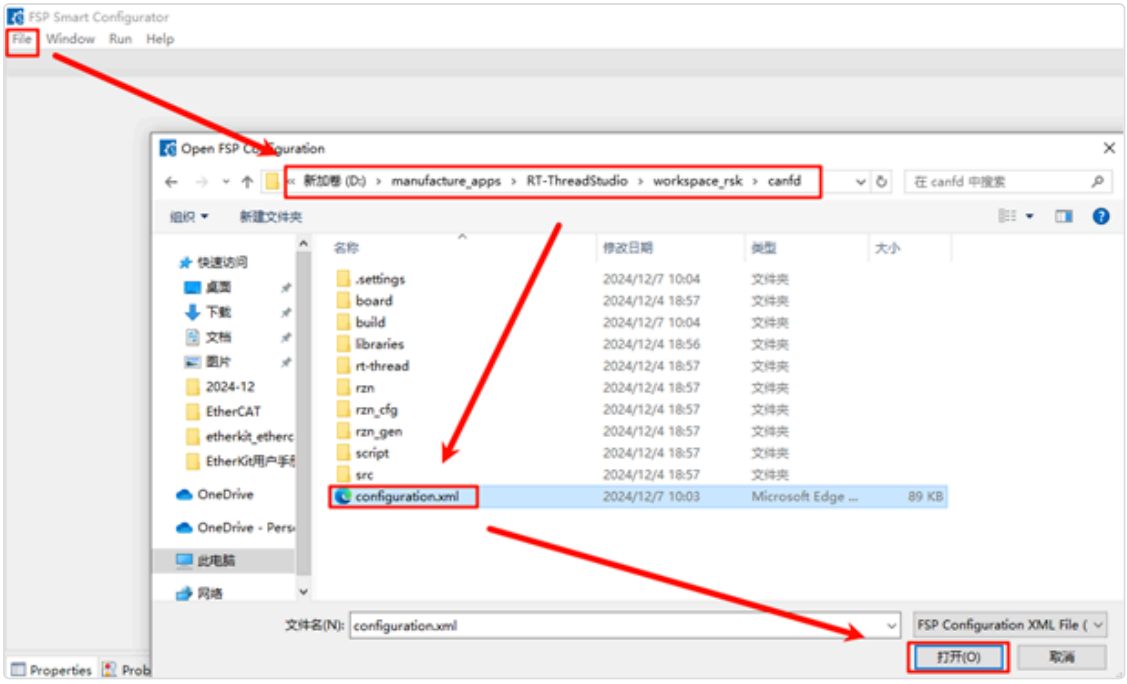
Hardware Description



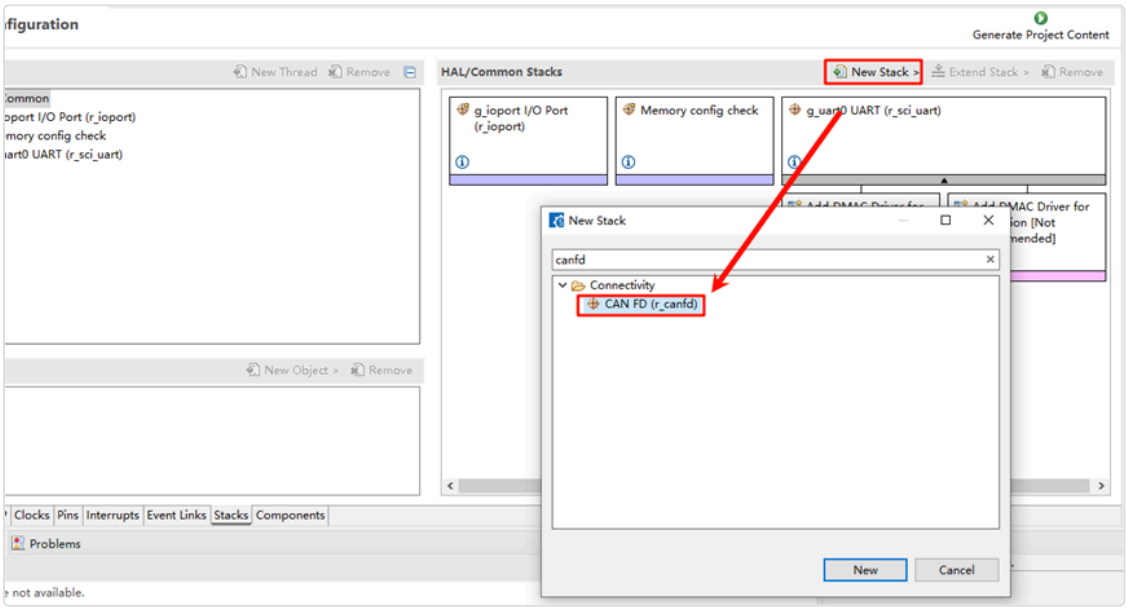
Software Description

FSP Configuration Guide

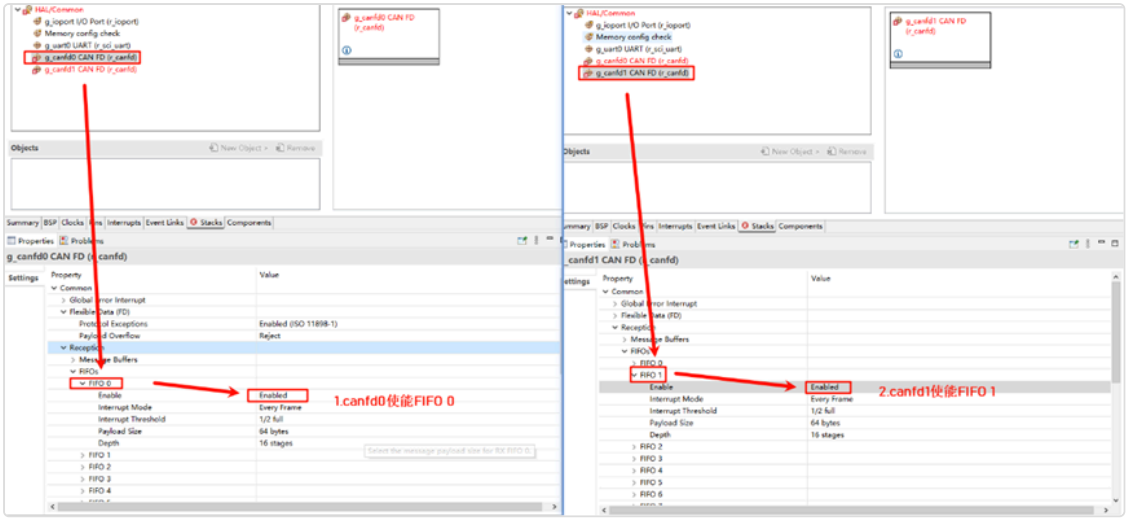
Select the configuration.xml file under the newly created project and open it in rzn-fsp.



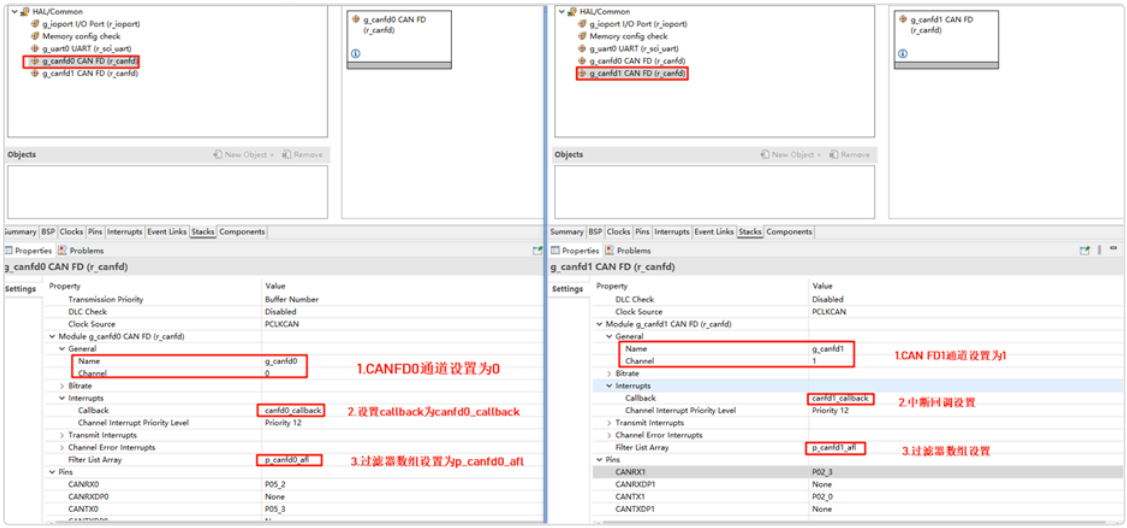
Click to add a new stack, search for `canfd`, and add `r_canfd`. Here, we need to add two `canfd_stack`.



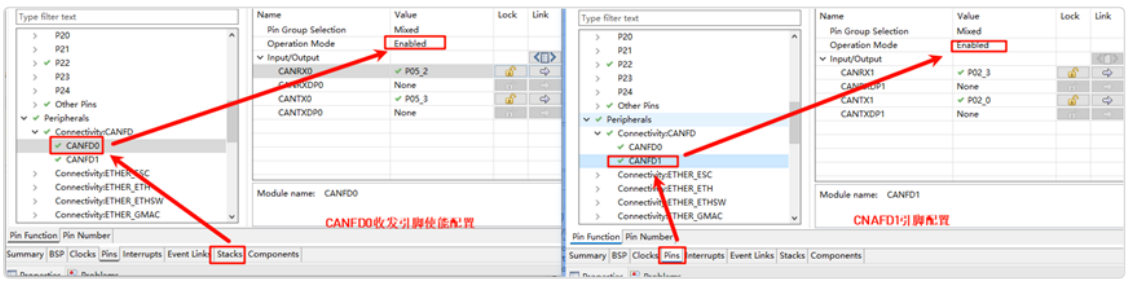
In the basic configuration, enable the reception FIFO interrupt for `canfd0` and `canfd1` by selecting Common -> Reception -> FIFOs -> FIFO 0 / FIFO 1 -> Enable. For `canfd0`, enable FIFO 0 interrupt, and for `canfd1`, enable FIFO 1 interrupt:



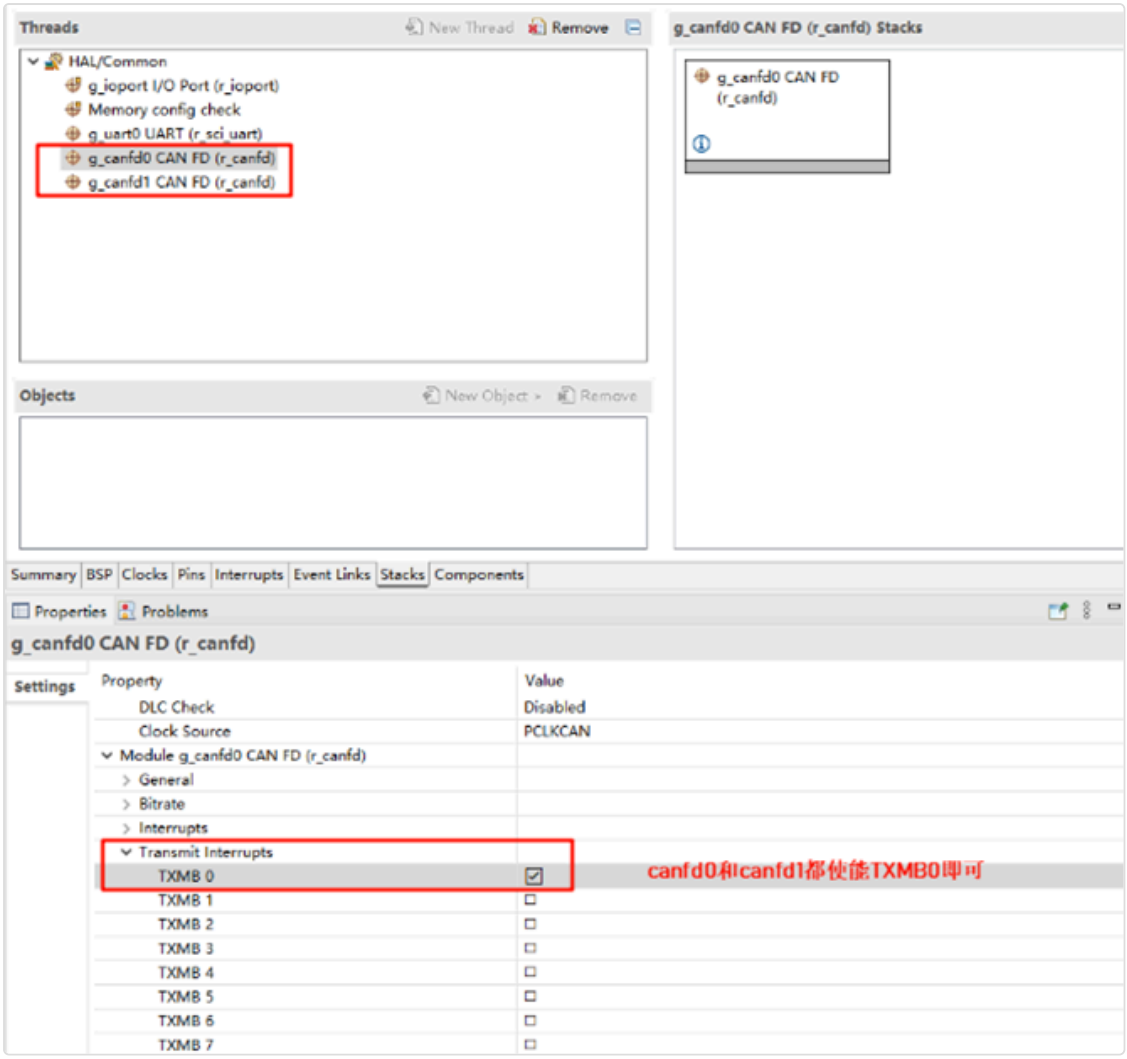
Next, we need to configure the channel, interrupt callback functions, and filter array for CANFD.



Configure and enable the CANFD pins.

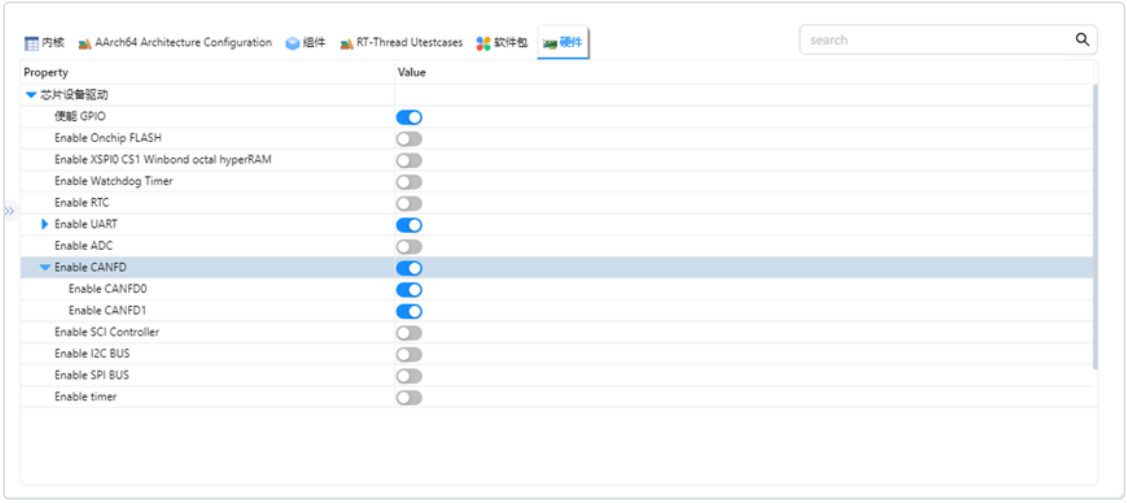


Next, enable the interrupt configuration for the message buffer, which determines which message buffer will trigger the interrupt when transmission is complete.



RT-Thread Settings Configuration

Open RT-Thread Settings and enable the CANFD configuration.



Project Example Description

The project sends a message using `canfd0` and receives it using `canfd1`, printing the result via serial port.

Example send code:

```
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);
    }
    /* Find the CAN device */
    can0_dev = rt_device_find(can_name);
    if (!can0_dev)
    {
        rt_kprintf("find %s failed!\n", can_name);
        return RT_ERROR;
    }
    /* Open the CAN device for interrupt-based receive and send
    res = rt_device_open(can0_dev, RT_DEVICE_FLAG_INT_TX | RT_D
    RT_ASSERT(res == RT_EOK);
    /* Create data reception thread */
    thread = rt_thread_create("can0_tx", can0_tx_thread, RT_NUL
    if (thread != RT_NULL)
    {
        rt_thread_startup(thread);
    }
    else
    {
        rt_kprintf("create can_rx thread failed!\n");
    }
    return res;
}
/* Export to msh command list */
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);
    }
    /* Find the CAN device */
    can1_dev = rt_device_find(can_name);
    if (!can1_dev)
    {
        rt_kprintf("find %s failed!\n", can_name);
        return RT_ERROR;
    }
    /* Initialize CAN reception semaphore */
    rt_sem_init(&rx_sem, "rx_sem", 0, RT_IPC_FLAG_FIFO);

    /* Open the CAN device for interrupt-based receive and send
    res = rt_device_open(can1_dev, RT_DEVICE_FLAG_INT_TX | RT_D
    RT_ASSERT(res == RT_EOK);
    /* Create data reception thread */
    thread = rt_thread_create("can1_rx", can1_rx_thread, RT_NUL
    if (thread != RT_NULL)
    {
        rt_thread_startup(thread);
    }
    else
    {
        rt_kprintf("create can_rx thread failed!\n");
    }
    return res;
}
```

Operation

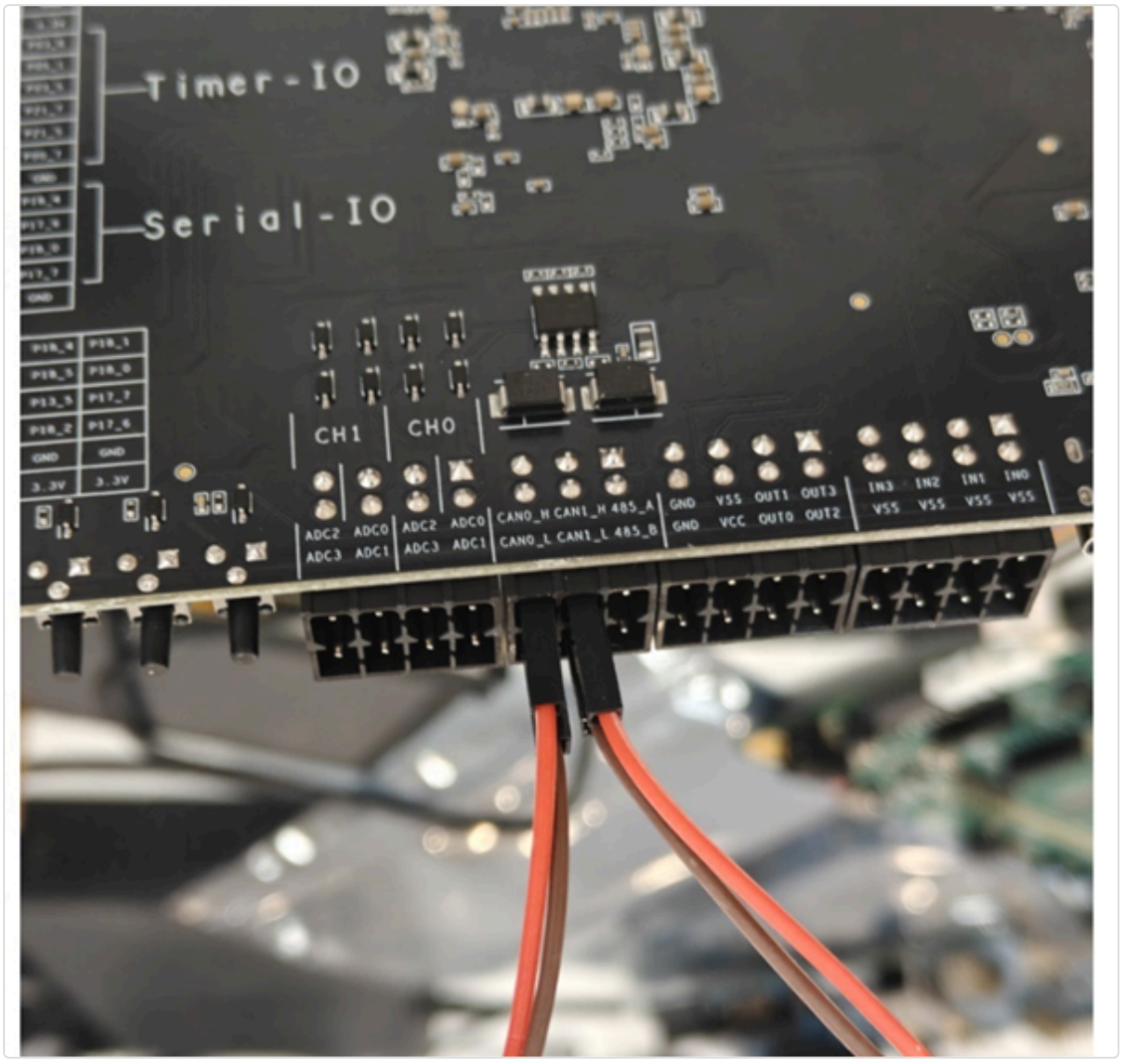
Compilation & Download

- **RT-Thread Studio**: Download the EtherKit resource package from the RT-Thread Studio package manager, then create a new project and compile.
- **IAR**: First, double-click `mklinks.bat` to generate links to the `rt-thread` and `libraries` folders. Then use Env to generate the IAR project. Finally, double-click `project.eww` to open the IAR project and compile.

After compiling, connect the Jlink interface of the development board to the PC and download the firmware to the development board.

Running Effect

Connect `CAN0_L` to `CAN1_L`, and `CAN0_H` to `CAN1_H`, as shown in the image below:



Use the serial port to send the `can0_sample_send` and `can1_sample_receive` commands for loopback testing:

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Dec  7 2024 10:48:06
 2006 - 2024 Copyright by RT-Thread team

Hello RT-Thread!
================================================
This example project is an canfd routine!
================================================
msh >can0
can0_sample_send
msh >can0_sample_send
msh >can1
can1_sample_receive
msh >can1_sample_receive
msh >ID:78      messege:rtt:0
ID:78   messege:rtt:1
ID:78   messege:rtt:2
ID:78   messege:rtt:3
ID:78   messege:rtt:4
ID:78   messege:rtt:5
ID:78   messege:rtt:6
ID:78   messege:rtt:7
ID:78   messege:rtt:8
ID:78   messege:rtt:9
ID:78   messege:rtt:10
ID:78   messege:rtt:11
ID:78   messege:rtt:12
ID:78   messege:rtt:13
```

3.3. Ethernet Driver Usage Instructions

English | 中文

Introduction

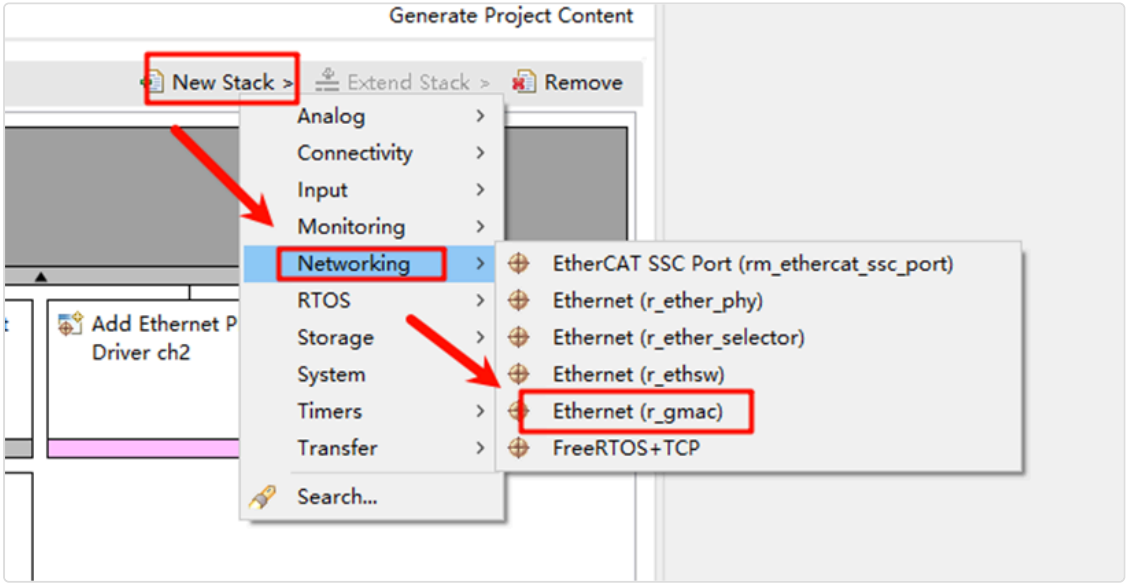
This project provides basic Ethernet functionalities, including `ping`, `tftp`, `ntp`, and `iperf`.

Hardware Connection

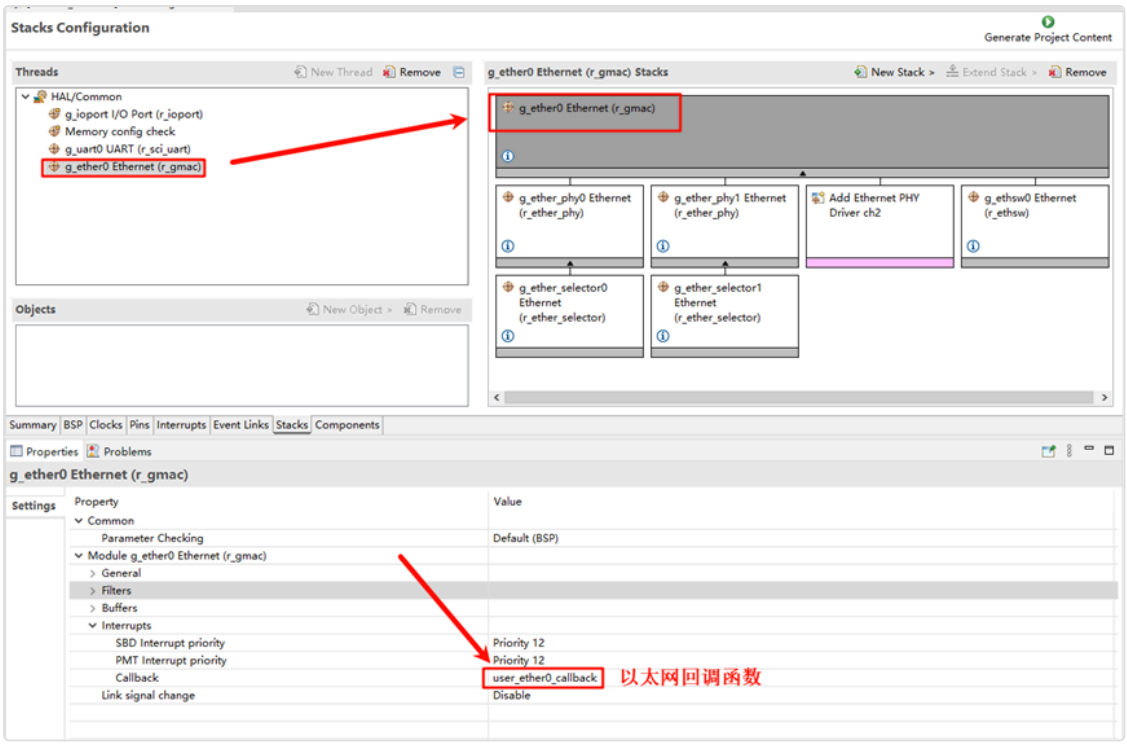
To use Ethernet, connect the development board to any one of the three network ports using an Ethernet cable, and the other end should be connected to a network switch that has internet access.

FSP Configuration Instructions

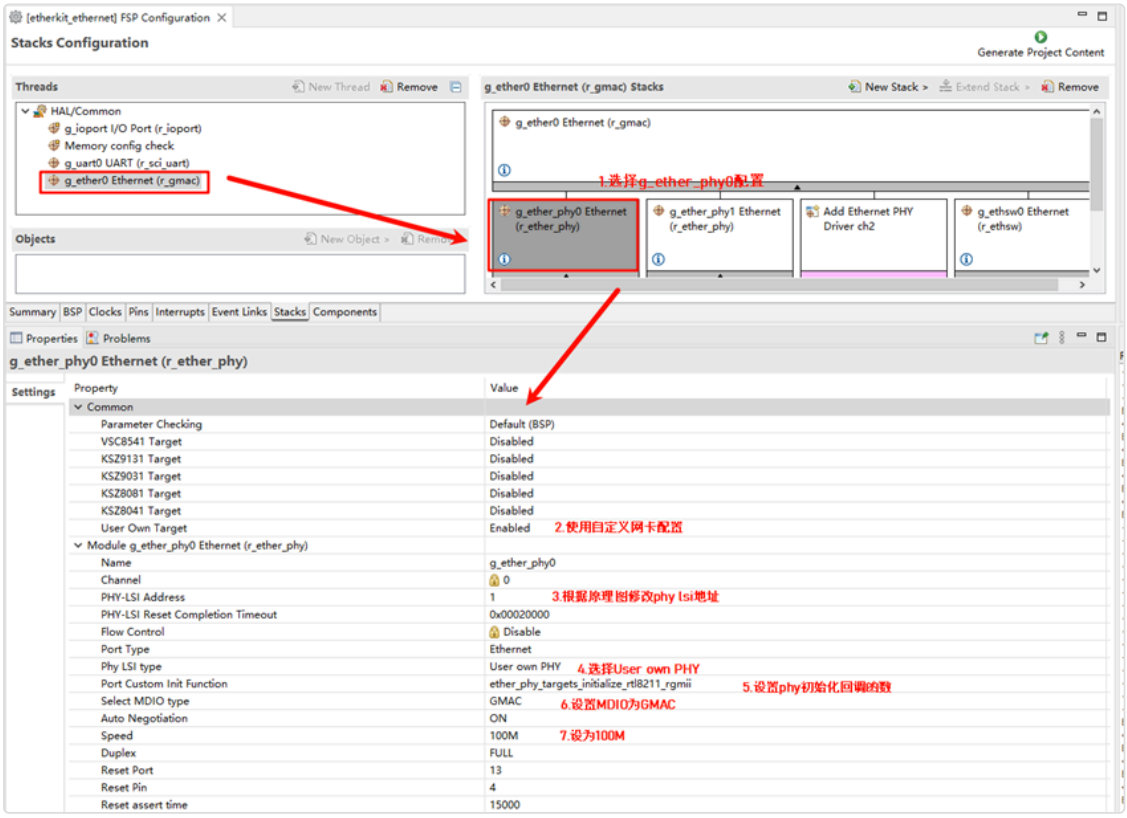
Open the project configuration file `configuration.xml` and add the `r_gamc` stack:



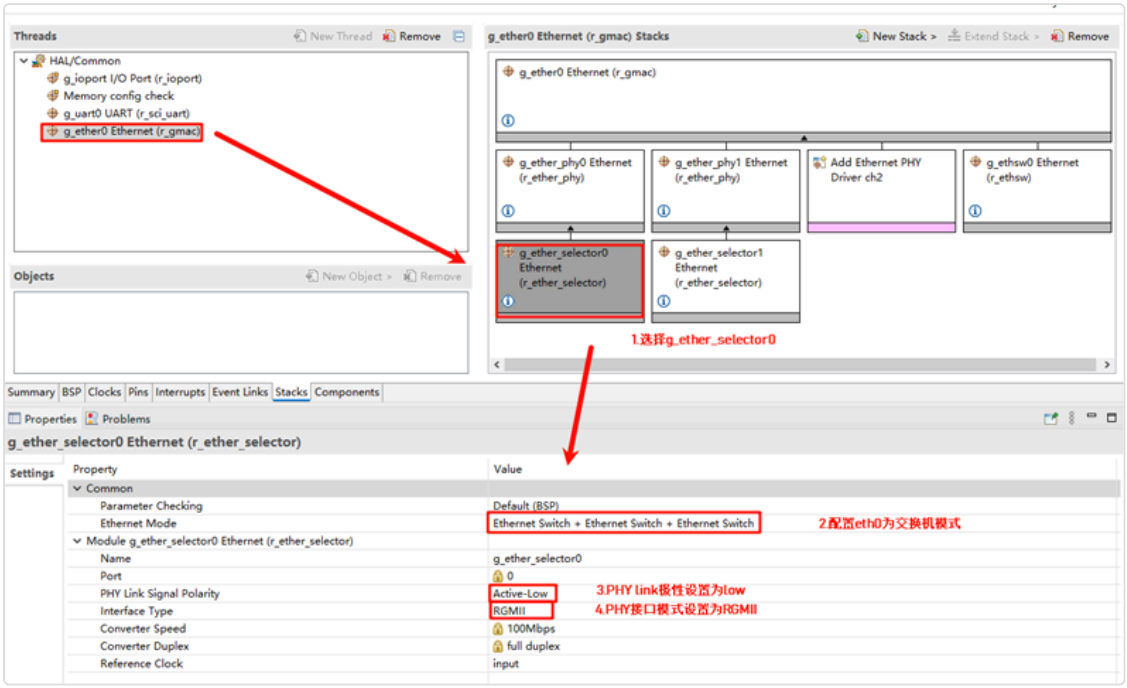
Next, click on `g_ether0 Ethernet`, and configure the interrupt callback function to `user_ether0_callback`:



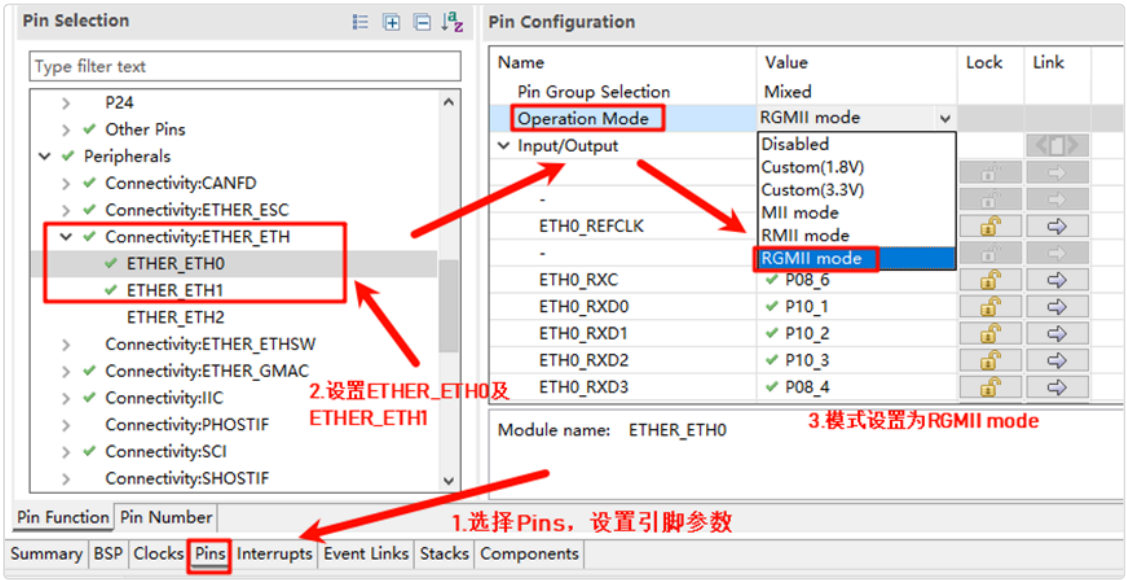
Now configure the PHY settings. Select `g_ether_phy0`, set the common configuration to "User Own Target", change the PHY LSI address to `1` (refer to the schematic for the exact address), and set the PHY initialization callback function to `ether_phy_targets_initialize_rtl8211_rgmii()`. Also, set the MDIO to GMAC.



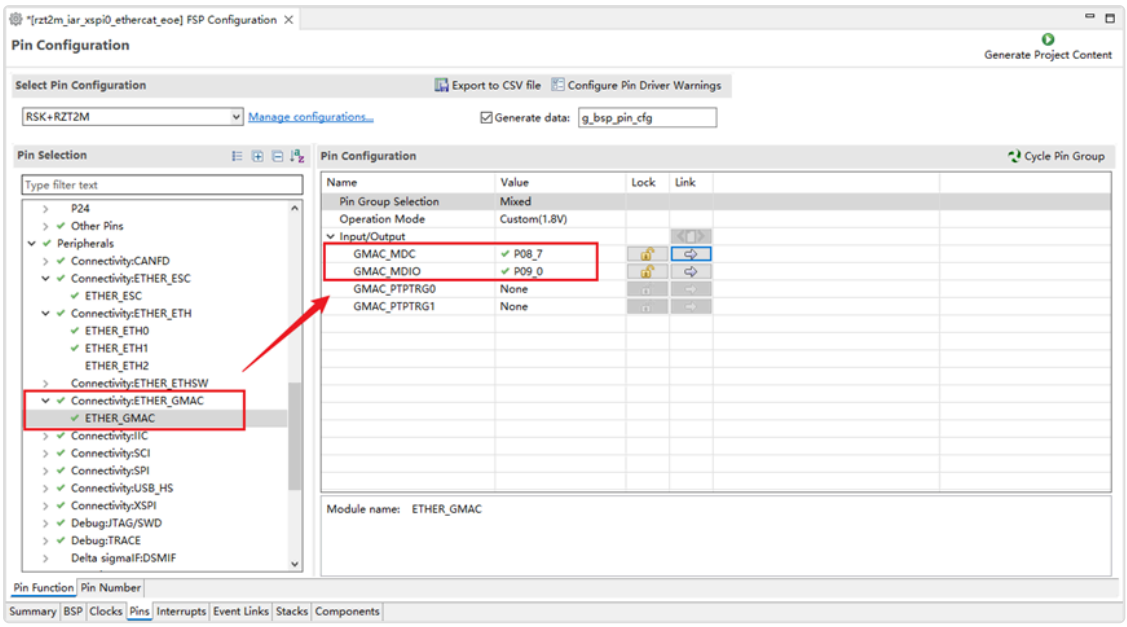
Next, configure `g_ether_selector0`, set the Ethernet mode to "Switch Mode", set the PHY link to "Default Active-Low", and choose "RGMII" for the PHY interface mode.



Configure the Ethernet pin parameters and select the operating mode to RGMII:

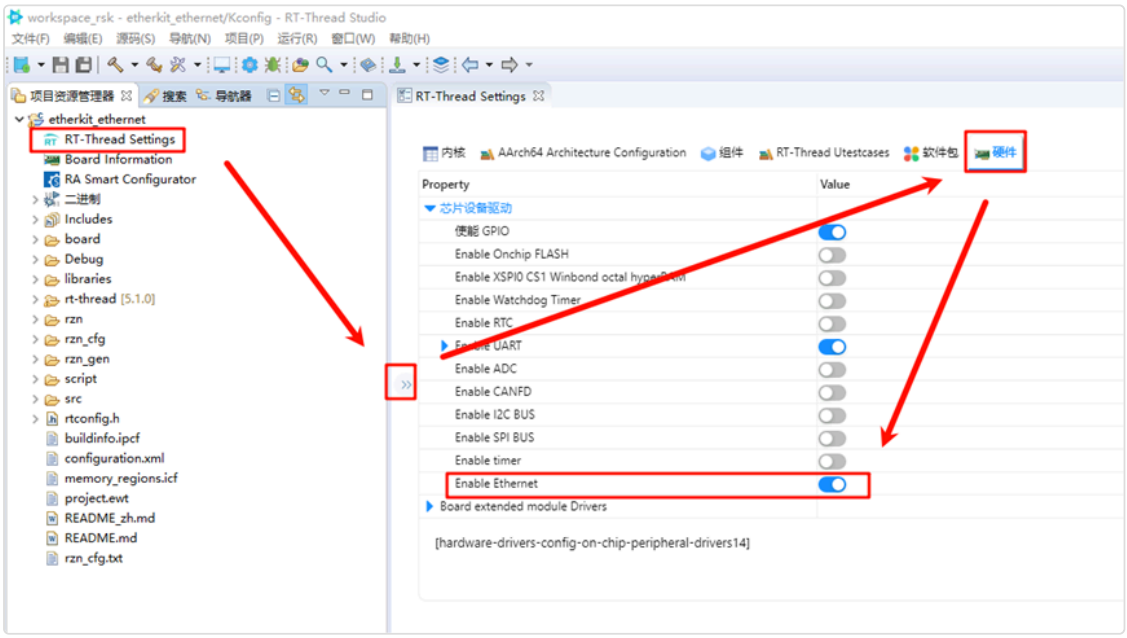


Finally, configure `ETHER_GMAC`:



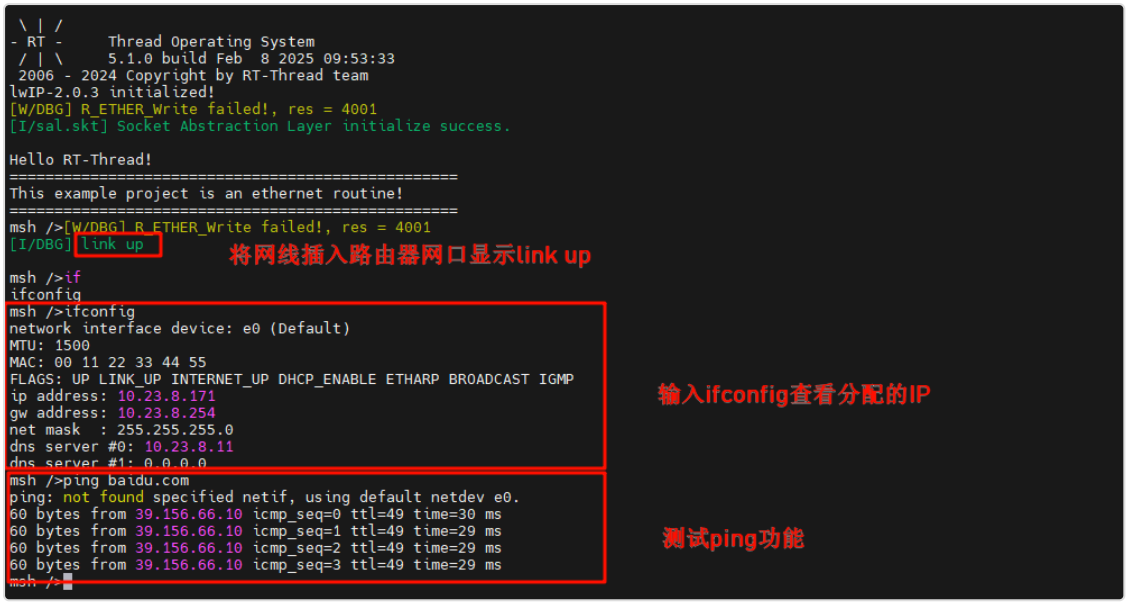
RT-Thread Studio Configuration

Return to the Studio project, and configure RT-Thread Settings. Click on "Hardware", find the chip device driver, and enable Ethernet:



Ethernet IP Experiment Results

After flashing the code to the development board, open the serial terminal to view the logs:


\ | /
- RT -   Thread Operating System
/ | \   5.1.0 build Feb 8 2025 09:53:33
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 ethernet routine!
msh />[W/DBG] R_ETHER_Write failed!, res = 4001
[I/DBG] link up
将网线插入路由器网口显示link up
msh />if
ifconfig
msh />ifconfig
network interface device: e0 (Default)
MTU: 1500
MAC: 00 11 22 33 44 55
FLAGS: UP LINK_UP INTERNET_UP DHCP_ENABLE ETHARP BROADCAST IGMP
ip address: 10.23.8.171
gw address: 10.23.8.254
net mask : 255.255.255.0
dns server #0: 10.23.8.11
dns server #1: 0.0.0.0
输入ifconfig查看分配的IP
msh />ping baidu.com
ping: not found specified netif, using default netdev e0.
60 bytes from 39.156.66.10 icmp_seq=0 ttl=49 time=30 ms
60 bytes from 39.156.66.10 icmp_seq=1 ttl=49 time=29 ms
60 bytes from 39.156.66.10 icmp_seq=2 ttl=49 time=29 ms
60 bytes from 39.156.66.10 icmp_seq=3 ttl=49 time=29 ms
msh />
测试ping功能

3.4. GPT Driver Usage Instructions

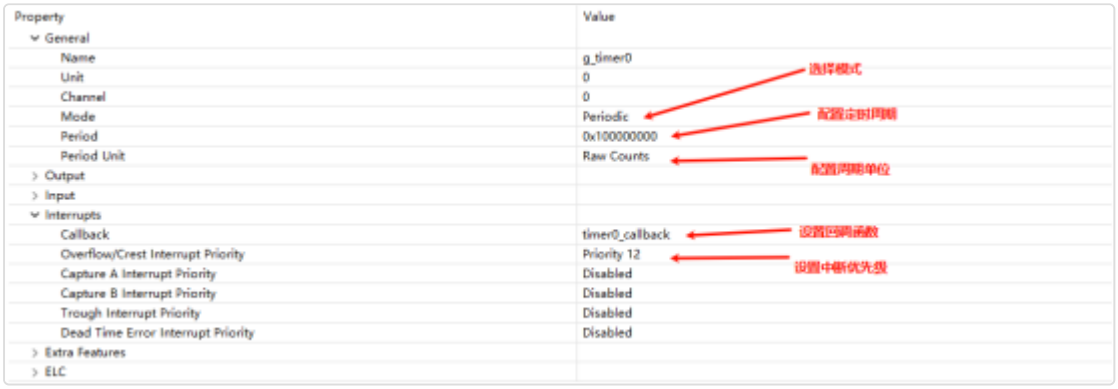
English | **中文**

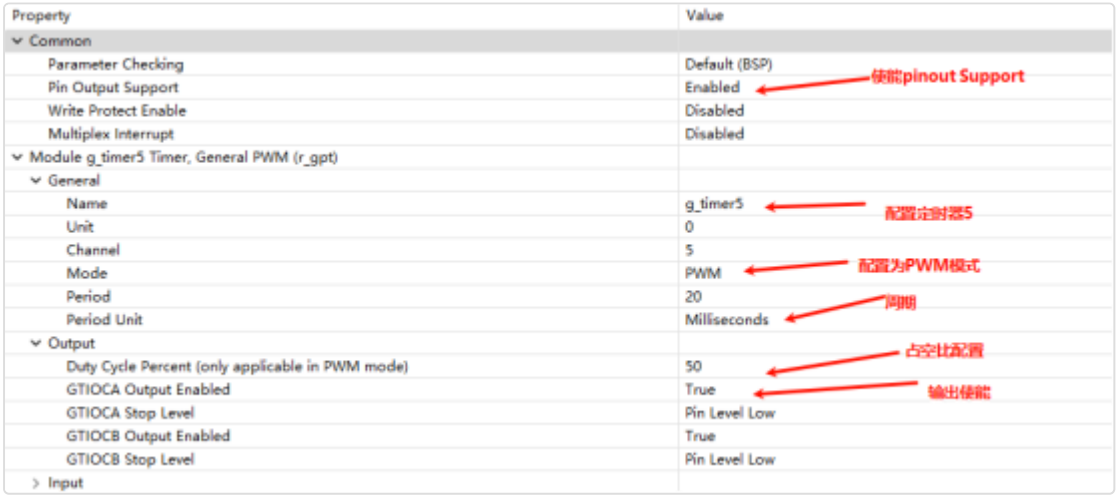
Introduction

In our specific application scenarios, the use of timers is often indispensable. This example mainly introduces how to use the GPT device on the EtherKit board, including the basic timer usage and PWM usage.

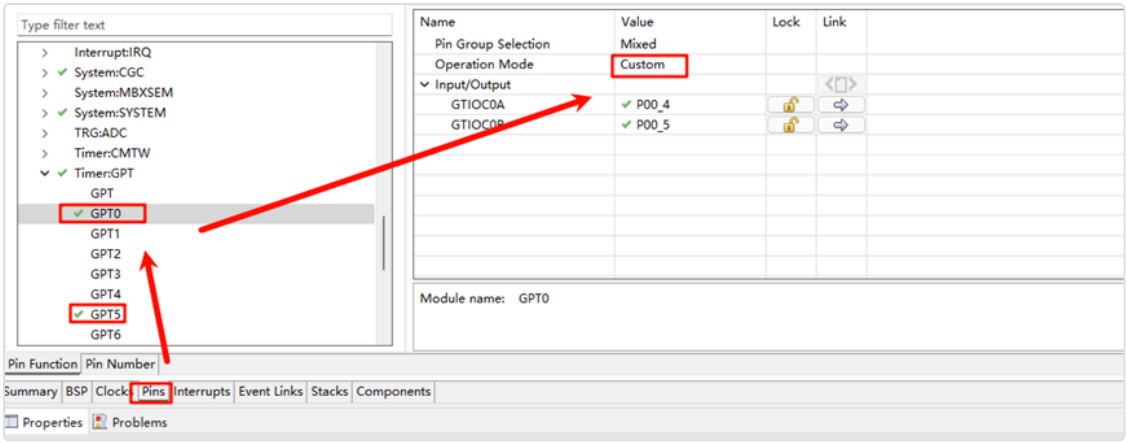
FSP Configuration Instructions

The FSP is configured to enable GPT0 as a basic timer mode and GPT5 as a PWM mode:



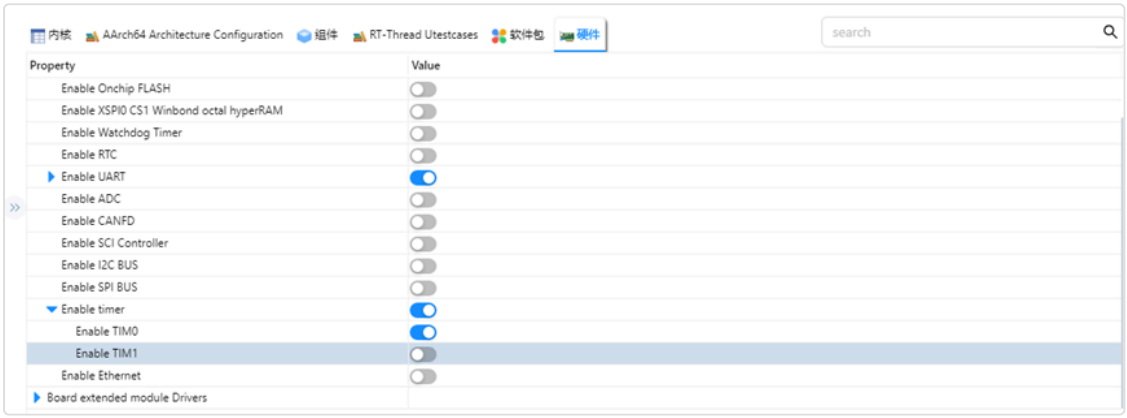


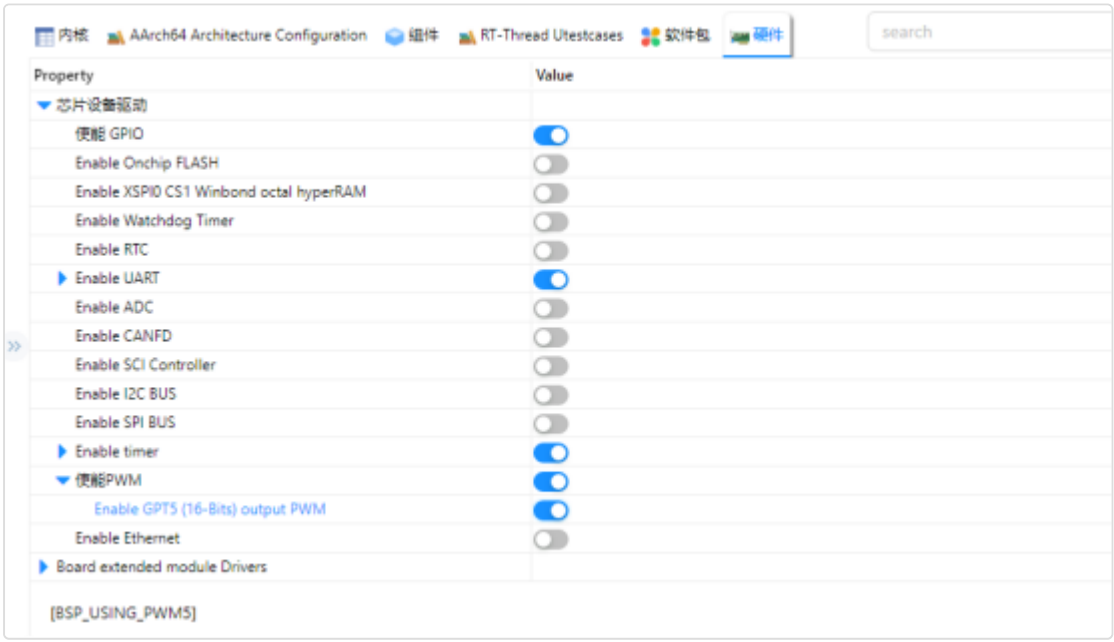
Additionally, the pins for GPT0 and GPT5 are enabled:



RT-Thread Settings Configuration

In the configuration, enable timer0 and PWM:





Example Project Instructions

The source code for this example is located in `/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; /* 1 MHz */
    hw_dev = rt_device_find(HWTIMER_DEV_NAME);
    if (hw_dev == RT_NULL)
    {
        rt_kprintf("hwtimer sample run failed! can't find %s de
        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, &mod
    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;  /* seconds */
    timeout_s.usec = 0; /* microseconds */
    if (rt_device_write(hw_dev, 0, &timeout_s, sizeof(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, ti
    return ret;
}
MSH_CMD_EXPORT(hwtimer_sample, hwtimer sample);
```

The interrupt callback function is triggered every 1 second, printing output. Below is the PWM configuration and enablement:

PWM-related macros:

The current version of the PWM driver treats each channel as a separate PWM device, with each device having only one channel (channel 0). Using the PWM5 device, note that channel 0 is selected here:

```
#define PWM_DEV_NAME        "pwm5"  /* PWM device name */
#define PWM_DEV_CHANNEL      0      /* PWM channel */
struct rt_device_pwm *pwm_dev;      /* PWM device handle */
```

Configure the PWM period and duty cycle:

```
static int pwm_sample(int argc, char *argv[])
{
    rt_uint32_t period, pulse, dir;
    period = 500000;    /* Period is 0.5 ms, in nanoseconds */
    dir = 1;            /* Direction of PWM pulse width increas
    pulse = 100000;     /* PWM pulse width value, in nanosecond
    /* Find device */
    pwm_dev = (struct rt_device_pwm *)rt_device_find(PWM_DEV_NA
    if (pwm_dev == RT_NULL)
    {
        rt_kprintf("pwm sample run failed! can't find %s device
        return RT_ERROR;
    }
    /* Set PWM period and default pulse width */
    rt_pwm_set(pwm_dev, PWM_DEV_CHANNEL, period, pulse);
    /* Enable device */
    rt_pwm_enable(pwm_dev, PWM_DEV_CHANNEL);
}
/* Export to msh command list */
MSH_CMD_EXPORT(pwm_sample, pwm sample);
```

Compilation & Download

- **RT-Thread Studio**: Download the EtherKit resource package in the RT-Thread Studio package manager, then create a new project and compile it.
- **IAR**: First, double-click `mklinks.bat` to generate the links for rt-thread and libraries folders. Then use Env to generate the IAR project, and finally, double-click `project.eww` to open the IAR project and compile it.

After compilation, connect the development board's Jlink interface to the PC, and download the firmware to the development board.

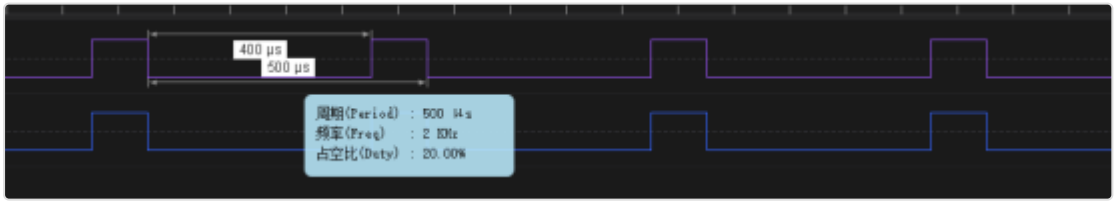
Running Results

In the serial terminal, input `pwm_sample` and `hwtimer_sample` to see the specific results.

The callback function is triggered every 1 second, and the output is printed:

```
 \ | /
- 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 !
```

Using a logic analyzer, the PWM output waveform is measured as follows:



3.5. HyperRAM Driver Usage Instructions

English | **中文**

Introduction

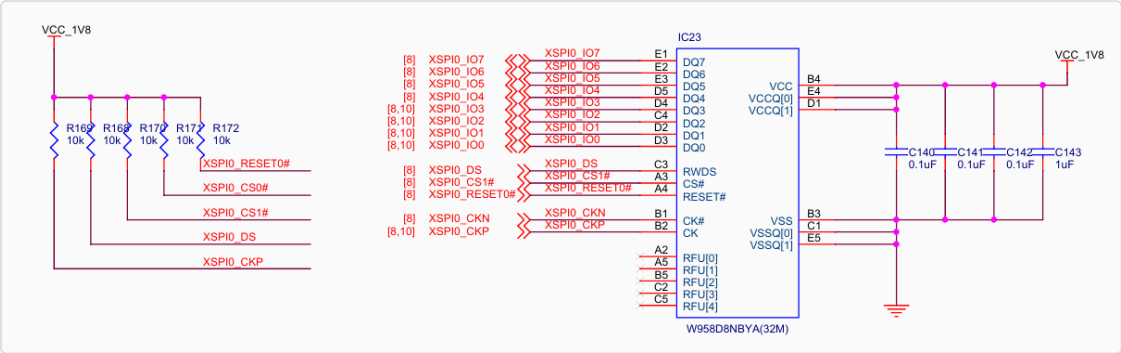
HyperRAM is a high-performance, low-latency external memory primarily used in embedded systems, providing additional storage capacity and fast data access. Its main features and principles are as follows:

- **Features**
- **High-Speed Data Access**: HyperRAM offers high data transfer rates, making it ideal for applications that require fast read/write operations, such as image processing and real-time data handling.
- **Low Power Consumption**: Compared to other types of memory, HyperRAM is designed with low power consumption, making it suitable for battery-powered embedded devices.
- **Extended Storage**: HyperRAM provides extra storage space for microcontrollers or other processors, addressing the issue of insufficient built-in RAM.
- **Simplified Design**: HyperRAM typically uses simple interfaces, simplifying hardware design and system integration.
- **Principles**
- **Serial Interface**: HyperRAM typically communicates via SPI or similar serial interfaces, making connections and data transfers efficient.
- **Dynamic Random Access Memory (DRAM)**: HyperRAM is based on DRAM technology, utilizing capacitors to store data, which allows for high density and large storage capacity.
- **Memory Controller**: A memory controller enables the microcontroller to manage data transfers with HyperRAM and control read/write operations.
- **Page Mode**: HyperRAM supports page mode operation, allowing more efficient data transfers and reduced access latency.

HyperRAM is widely used in embedded systems, particularly in applications where high storage speed and energy efficiency are critical.

This example demonstrates how to use the HyperRAM driver for read/write testing on the EtherKit development board.

Hardware Overview



Software Overview

The source code for this example is located in `../library/HAL_Drivers/drv_hyperram.c`:

```
/*
 * Copyright (c) 2006-2024, RT-Thread Development Team
 *
 * SPDX-License-Identifier: Apache-2.0
 *
 * Change Logs:
 * Date           Author       Notes
 * 2024-10-28     yuanjie      first version
 */

#include <rtthread.h>
#include "hal_data.h"
#ifdef BSP_USING_HYPERRAM

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

#define PSRAM_BANK_ADDR                 ((uint32_t)0x44000000UL
#define PSRAM_SIZE                      ((uint32_t)0x2000000UL)
#define PSRAM_DATA_WIDTH                16

#ifdef RT_USING_MEMHEAP_AS_HEAP
```

```
static struct rt_memheap system_heap;
#endif

static int HYPERRAM_Init(void)
{
    int result = RT_EOK;
    /* XSPI initial settings */
    /* Initialize the PSRAM controller */
    if (R_XSPI_HYPER_Open(&g_hyperbus0_ctrl, &g_hyperbus0_cfg)
    {
        LOG_E("HYPER RAM init failed!");
        result = -RT_ERROR;
    }
    else
    {
        LOG_D("PSRAM init success, mapped at 0x%X, size is %d b
#ifdef RT_USING_MEMHEAP_AS_HEAP
        /* If RT_USING_MEMHEAP_AS_HEAP is enabled, PSRAM is ini
        rt_memheap_init(&system_heap, "psram", (void *)PSRAM_BA
#endif
    }

    return result;
}
INIT_BOARD_EXPORT(HYPERRAM_Init);

#ifdef DRV_DEBUG
#ifdef FINSH_USING_MSH
int psram_test(void)
{
    int i = 0;
    uint32_t start_time = 0, time_cast = 0;
#if PSRAM_DATA_WIDTH == 8
    char data_width = 1;
    uint8_t data = 0;
#elif PSRAM_DATA_WIDTH == 16
    char data_width = 2;
    uint16_t data = 0;
#else
    char data_width = 4;
    uint32_t data = 0;
#endif

    /* write data */
    LOG_D("Writing the %ld bytes data, waiting....", PSRAM_SIZE
    start_time = rt_tick_get();
    for (i = 0; i < PSRAM_SIZE / data_width; i++)
    {
#if PSRAM_DATA_WIDTH == 8
```

```
        *(__IO uint8_t *)(PSRAM_BANK_ADDR + i * data_width) = (
#elif PSRAM_DATA_WIDTH == 16
        *(__IO uint16_t *)(PSRAM_BANK_ADDR + i * data_width) =
#else
        *(__IO uint32_t *)(PSRAM_BANK_ADDR + i * data_width) =
#endif
    }
    time_cast = rt_tick_get() - start_time;
    LOG_D("Write data success, total time: %d.%03dS.", time_cas
          time_cast % RT_TICK_PER_SECOND / ((RT_TICK_PER_SECOND

    /* read data */
    LOG_D("Start reading and verifying data, waiting....");
    for (i = 0; i < PSRAM_SIZE / data_width; i++)
    {
#if PSRAM_DATA_WIDTH == 8
        data = *(__IO uint8_t *)(PSRAM_BANK_ADDR + i * data_wid
        if (data != 0x55)
        {
            LOG_E("PSRAM test failed!");
            break;
        }
#elif PSRAM_DATA_WIDTH == 16
        data = *(__IO uint16_t *)(PSRAM_BANK_ADDR + i * data_wi
        if (data != 0x5555)
        {
            LOG_E("PSRAM test failed!");
            break;
        }
#else
        data = *(__IO uint32_t *)(PSRAM_BANK_ADDR + i * data_wi
        if (data != 0x55555555)
        {
            LOG_E("PSRAM test failed!");
            break;
        }
#endif
    }

    if (i >= PSRAM_SIZE / data_width)
    {
        LOG_D("PSRAM test success!");
    }

    return RT_EOK;
}
MSH_CMD_EXPORT(psram_test, XSPI XIP hyper ram test)
#endif /* FINSH_USING_MSH */
```

```
#endif /* DRV_DEBUG */
#endif /* BSP_USING_HYPERRAM */
```

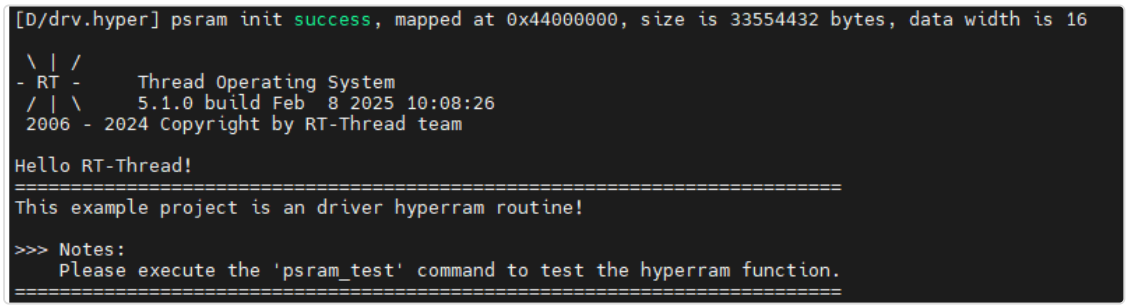
Compilation & Download

- **RT-Thread Studio**: Download the EtherKit resource package from the RT-Thread Studio package manager, create a new project, and then compile it.
- **IAR**: First, double-click `mklinks.bat` to generate links to the `rt-thread` and `libraries` folders. Then use the `Env` tool to generate an IAR project, and finally double-click `project.eww` to open the IAR project and compile it.

Once the compilation is completed, connect the development board's Jlink interface to the PC and download the firmware to the development board.

Run Results

Press the reset button to restart the development board and observe the terminal logs:



Execute the `psram_test` command to start the HyperRAM read/write test:

```
msh >psram_test
[D/drv.hyper] Writing the 33554432 bytes data, waiting....
[D/drv.hyper] Write data success, total time: 0.866S.
[D/drv.hyper] Start reading and verifying data, waiting....
[D/drv.hyper] PSRAM test success!
```

3.6. IIC EEPROM Driver Usage Instructions

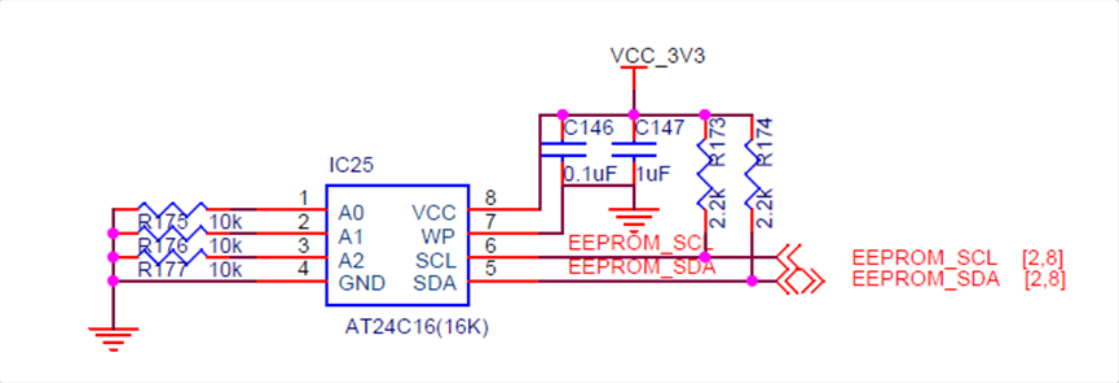
English | **中文**

Introduction

This example demonstrates how to use the RT-Thread I2C framework on the EtherKit to perform read and write operations on the onboard EEPROM.

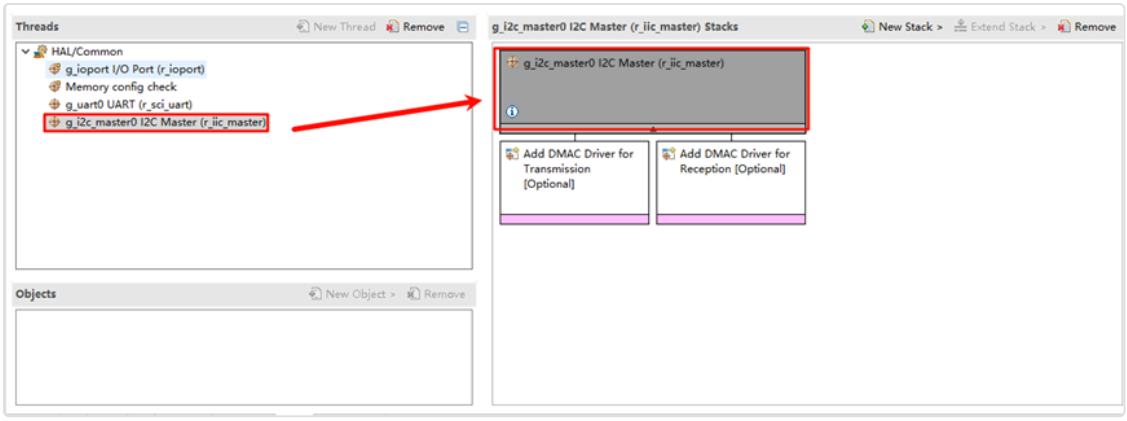
Hardware Description

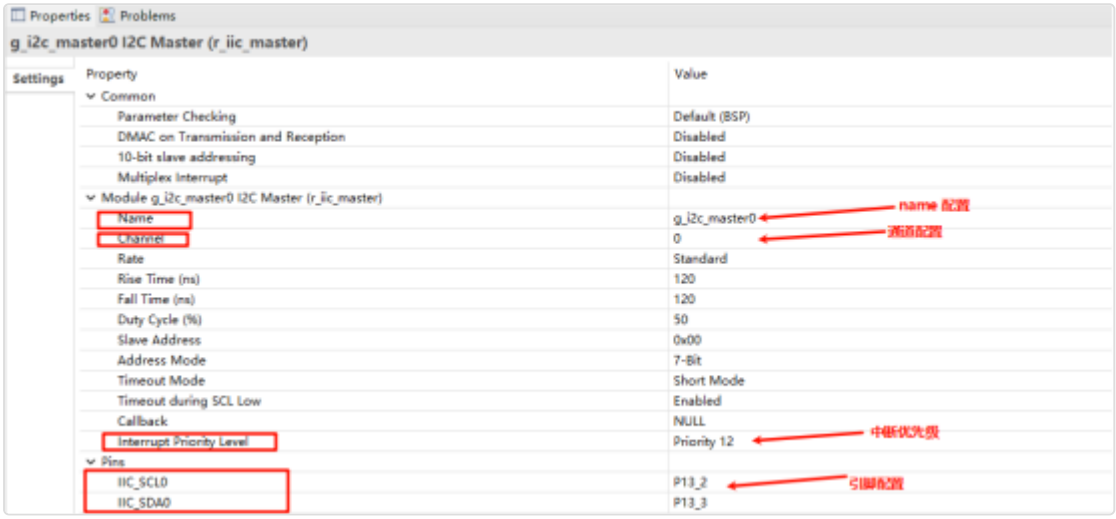
The EEPROM used on the EtherKit is an AT24C16, which is connected to the I2C0 of the R9A07G084M08GBG chip.

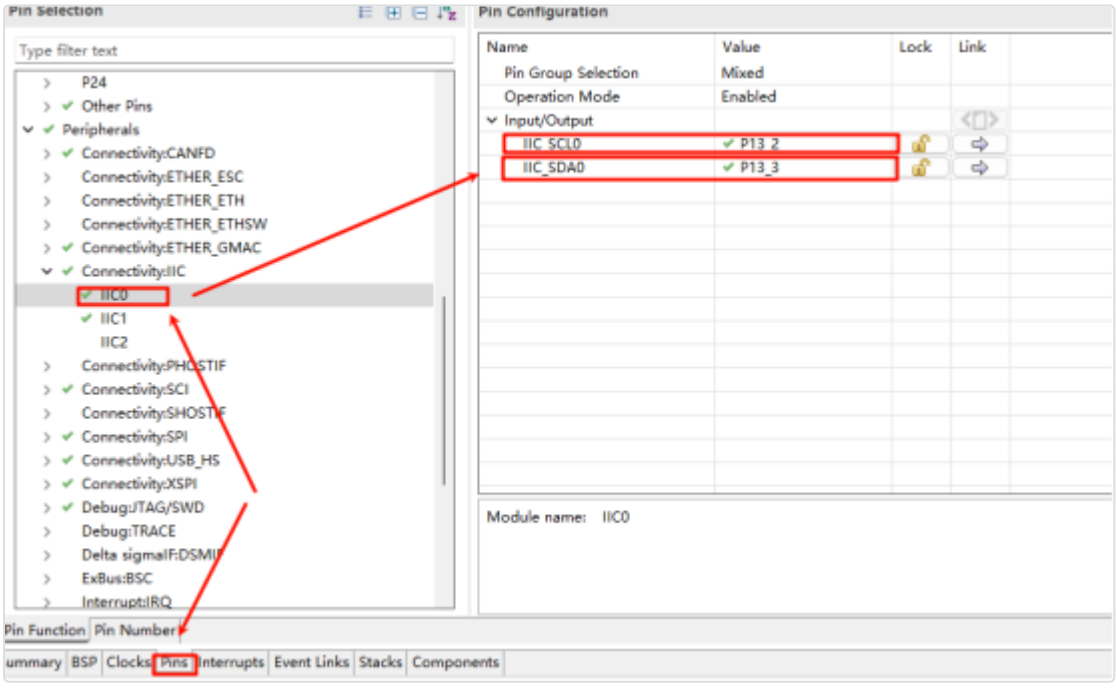


FSP Configuration Instructions

Create a new stack and select `r_iic_master`. Then, configure the I2C0 settings as shown below:

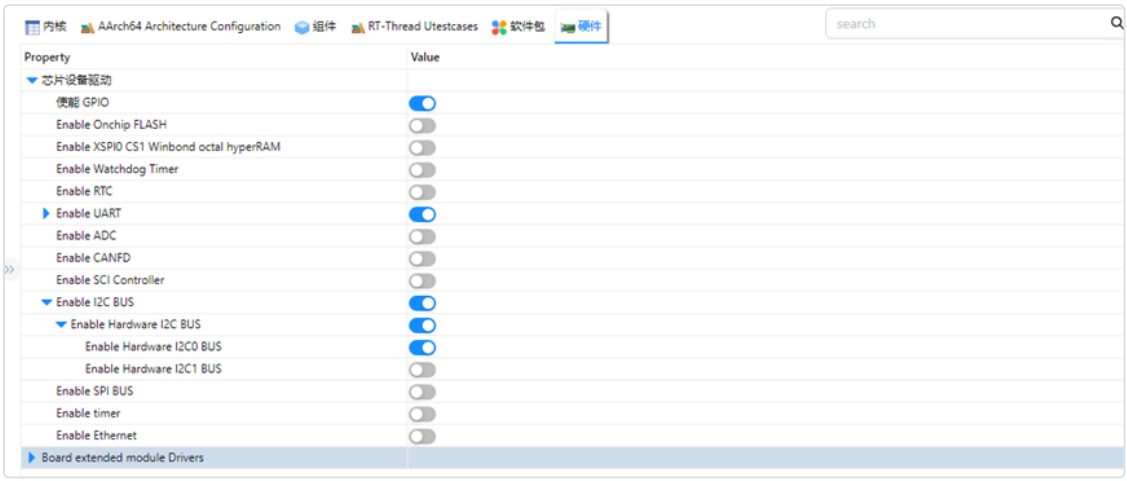

Threads
New Thread
Remove
HAL/Common
g_ioport I/O Port (r_ioport)
Memory config check
g_uart0 UART (r_sci_uart)
g_i2c_master0 I2C Master (r_iic_master)
Objects
New Object >
Remove
g_i2c_master0 I2C Master (r_iic_master) Stacks
New Stack >
Extend Stack >
Remove
g_i2c_master0 I2C Master (r_iic_master)
Add DMAC Driver for Transmission [Optional]
Add DMAC Driver for Reception [Optional]

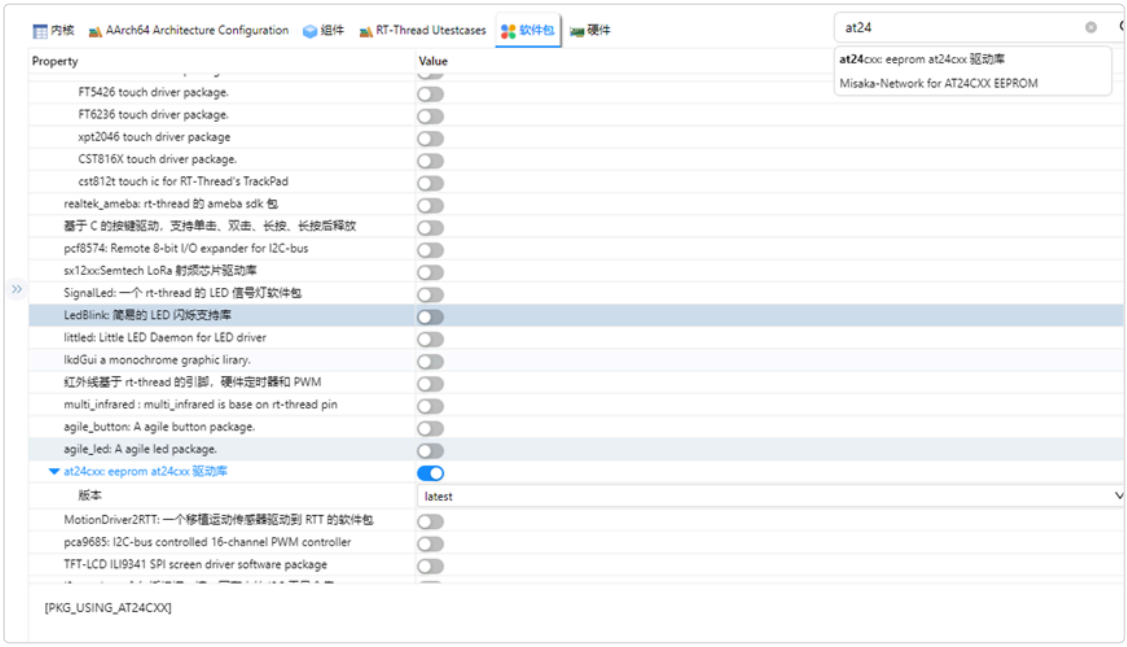

Properties
Problems
g_i2c_master0 I2C Master (r_iic_master)
Settings
Property
Value
Common
Parameter Checking
Default (BSP)
DMAC on Transmission and Reception
Disabled
10-bit slave addressing
Disabled
Multiplex Interrupt
Disabled
Module g_i2c_master0 I2C Master (r_iic_master)
Name
g_i2c_master0
name 配置
Channel
0
通道配置
Rate
Standard
Rise Time (ns)
120
Fall Time (ns)
120
Duty Cycle (%)
50
Slave Address
0x00
Address Mode
7-Bit
Timeout Mode
Short Mode
Timeout during SCL Low
Enabled
Callback
NULL
Interrupt Priority Level
Priority 12
中断优先级
Pins
IIC_SCL0
P13_2
IIC_SDA0
P13_3
引脚配置


Pin Selection
Type filter text
P24
Other Pins
Peripherals
Connectivity:CANFD
Connectivity:ETHER_ESC
Connectivity:ETHER_ETH
Connectivity:ETHER_ETHSW
Connectivity:ETHER_GMAC
Connectivity:IIC
IIC0
IIC1
IIC2
Connectivity:PHOSTIF
Connectivity:SCI
Connectivity:SHOSTIF
Connectivity:SPI
Connectivity:USB_HS
Connectivity:XSPI
Debug:JTAG/SWD
Debug:TRACE
Delta sigmaIF:DSMIF
ExBus:BSC
Interrupt:IRQ
Pin Configuration
Name
Value
Lock
Link
Pin Group Selection
Mixed
Operation Mode
Enabled
Input/Output
IIC_SCL0
P13_2
IIC_SDA0
P13_3
Module name: IIC0
Pin Function
Pin Number
ummary
BSP
Clocks
Pins
Interrupts
Event Links
Stacks
Components

RT-Thread Settings Configuration

Enable the RT-Thread I2C driver framework and the AT24C16 driver software package in the configuration.





Example Project Description

The example uses the AT24C16 driver package to perform read and write operations at EEPROM addresses `0x00` and `0x20`.

```
#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(st
    rt_kprintf("write eeprom data to 0x0: %s\n", str1);
    rt_thread_mdelay(1000);
    at24cxx_read(at24c02_dev, 0x0, read_buffer1, (sizeof(str1)
    rt_kprintf("read eeprom data from 0x0: %s\n", read_buffer1)
    at24cxx_write(at24c02_dev, 0x20, (uint8_t *)str2, (sizeof(s
    rt_kprintf("write eeprom data to 0x20: %s\n", str2);
    rt_thread_mdelay(1000);
    at24cxx_read(at24c02_dev, 0x20, read_buffer2, (sizeof(str2)
    rt_kprintf("read eeprom data from 0x20: %s\n", read_buffer2
    if (rt_strcmp((const char *)str1, (const char *)read_buffer
        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
```

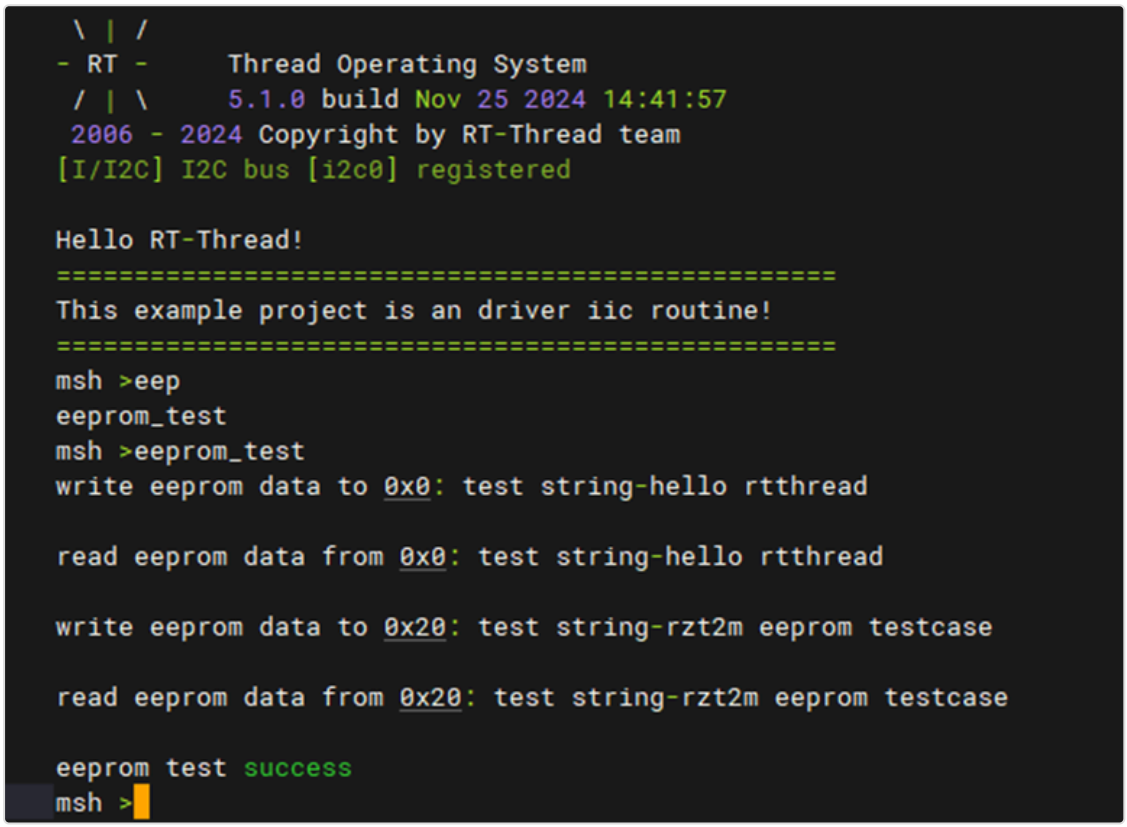
Compilation & Download

- **RT-Thread Studio**: In RT-Thread Studio's package manager, download the EtherKit resource package, create a new project, and compile it.
- **IAR**: First, double-click `mklinks.bat` to create symbolic links between RT-Thread and the libraries folder. Then, use the `Env` tool to generate the IAR project. Finally, double-click `project.eww` to open the IAR project and compile it.

After compilation, connect the development board's JLink interface to the PC and download the firmware to the development board.

Run Effect

To test the EEPROM, enter the `eeprom_test` command in the serial terminal:



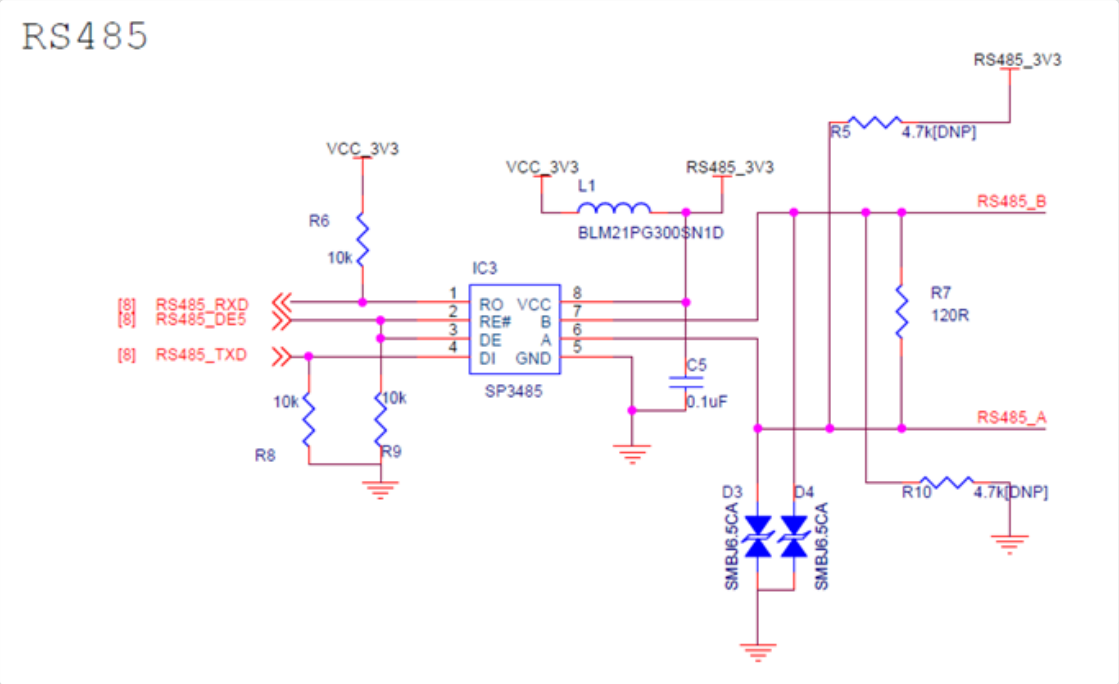
3.7. RS485 Driver Usage Instructions

English | **中文**

Introduction

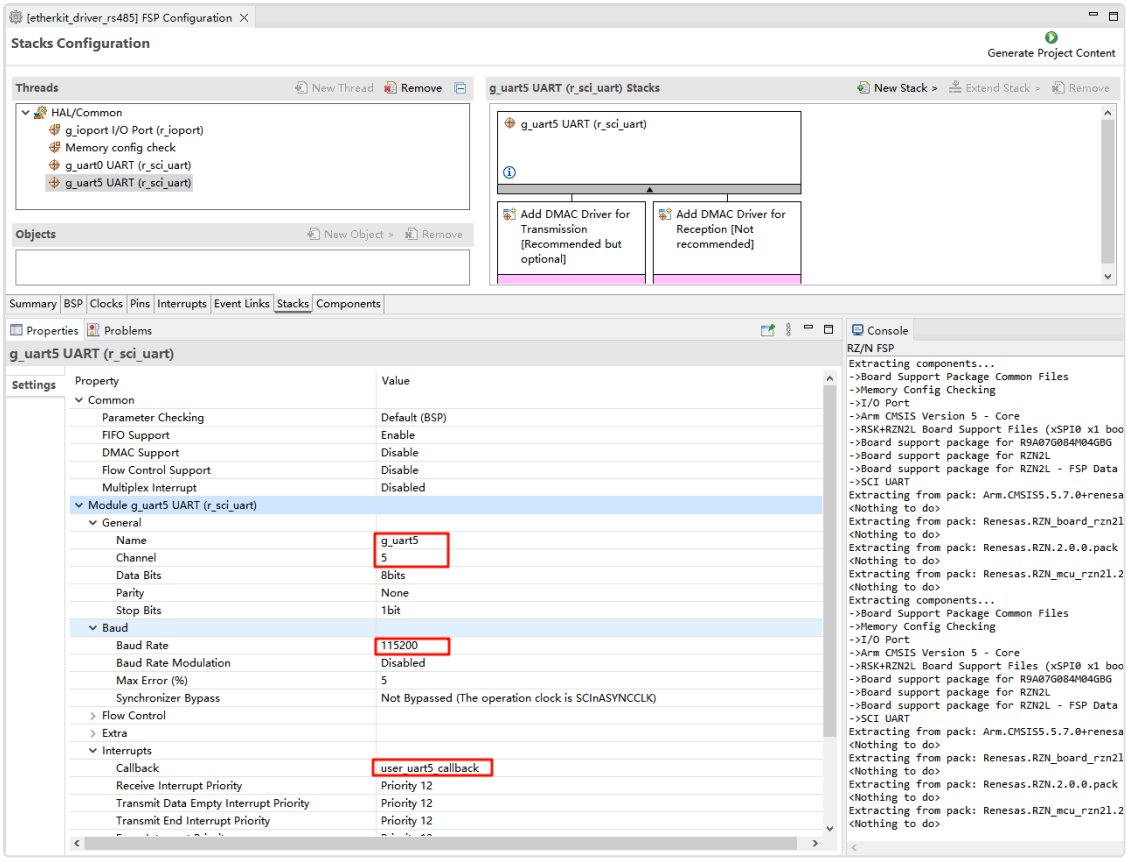
This example demonstrates how to use an RS485 device on the EtherKit.

Hardware Overview



FSP Configuration

Open the FSP tool, create a new Stack, and select **r_sci_uart5**. The specific configuration details are as follows:



Example Project Description

The RS485 driver is initialized, and characters received from the RS485 serial terminal are printed to the Finsh terminal, while being echoed back to the RS485 terminal.

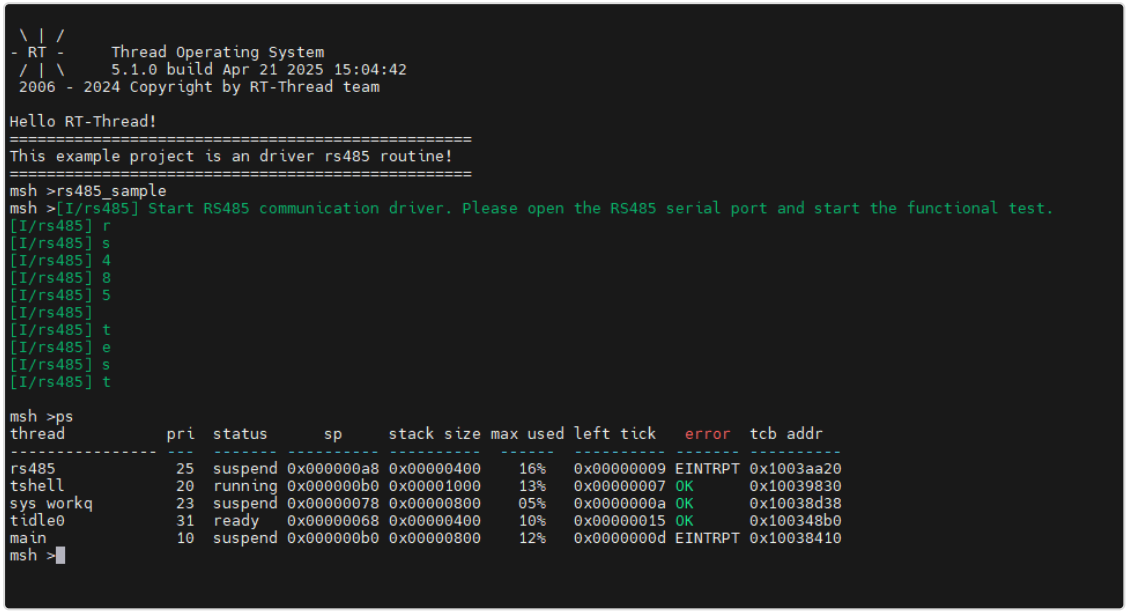
Build & Download

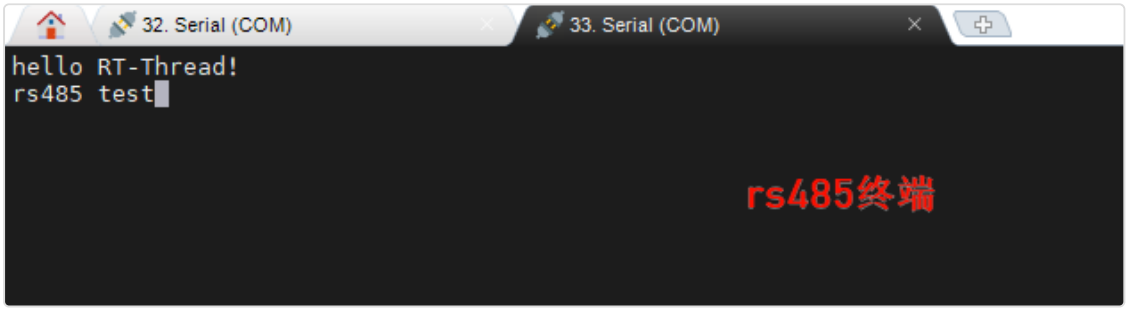
- **RT-Thread Studio:** Download the EtherKit resource pack from the RT-Thread Studio package manager, then create a new project and compile it.
- **IAR:** First, double-click `mklinks.bat` to generate links for the `rt-thread` and `libraries` folders. Then, use Env to generate the IAR project. Finally, double-click `project.eww` to open the IAR project and compile it.

Once compiled, connect the development board's JLink interface to the PC, and download the firmware to the development board.

Running Result

Run the `rs485_sample` command in the serial console, and open the RS485 serial terminal to view the received data:





3.8. SCI_SPI Driver Usage Instructions

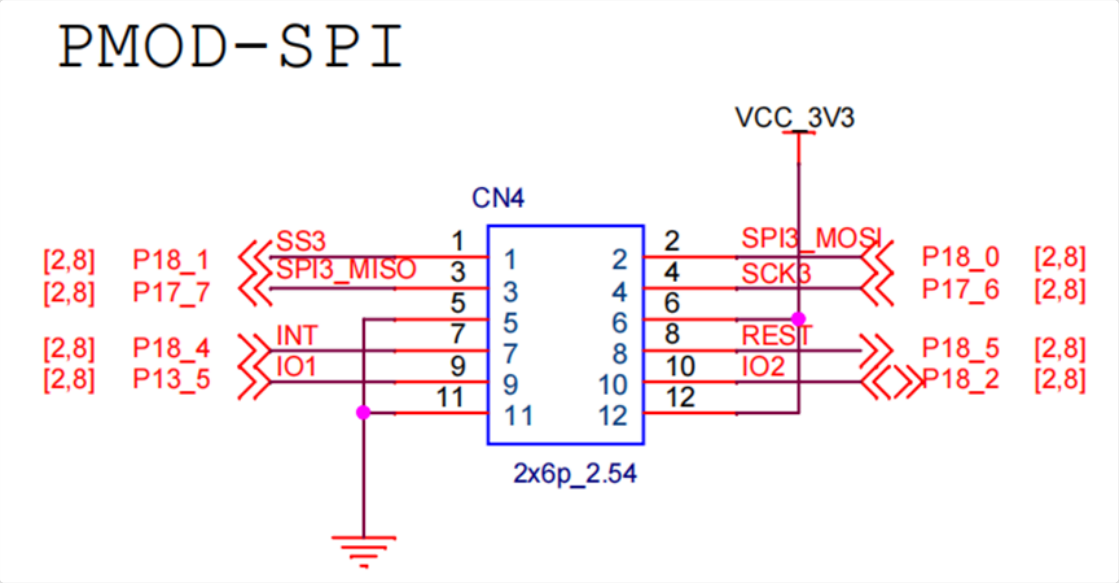
English | **中文**

Introduction

This example demonstrates how to use the RT-Thread SCI_SPI framework on the EtherKit.

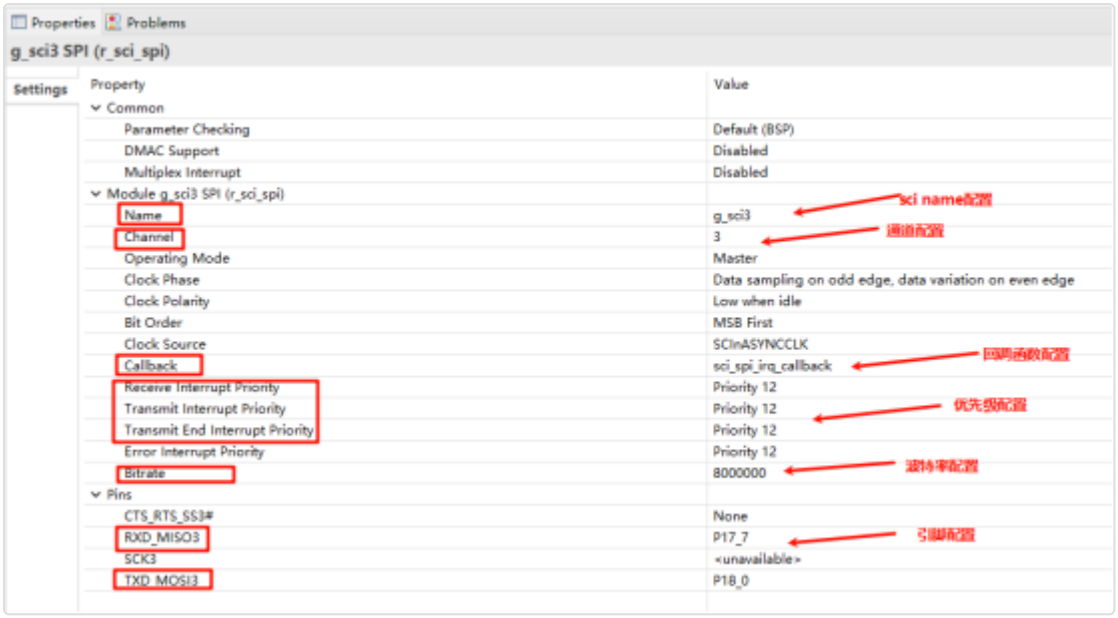
Hardware Description

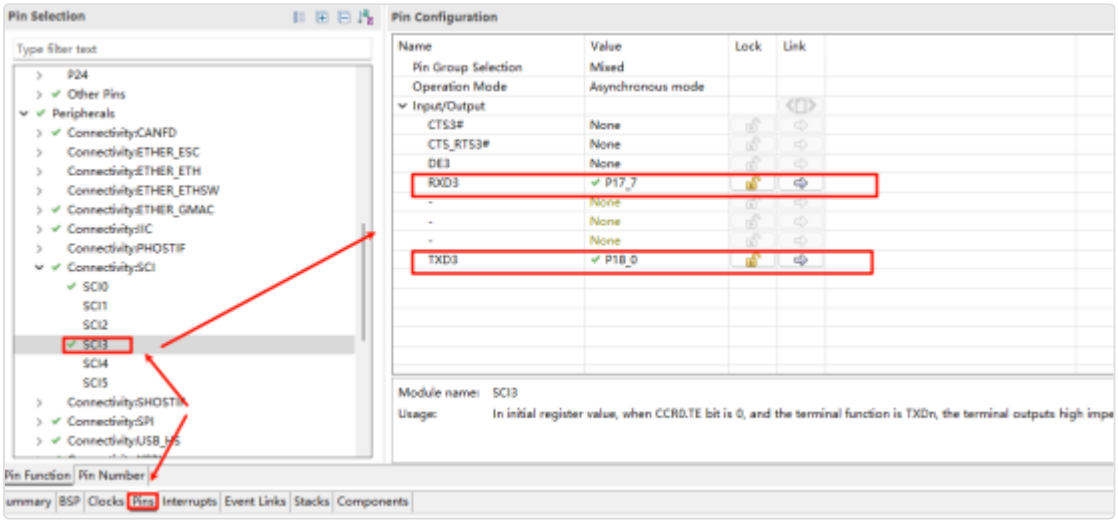
The EtherKit board has a PMOD interface, which is connected to the SCI_SPI3 of the R9A07G084M08GBG chip.

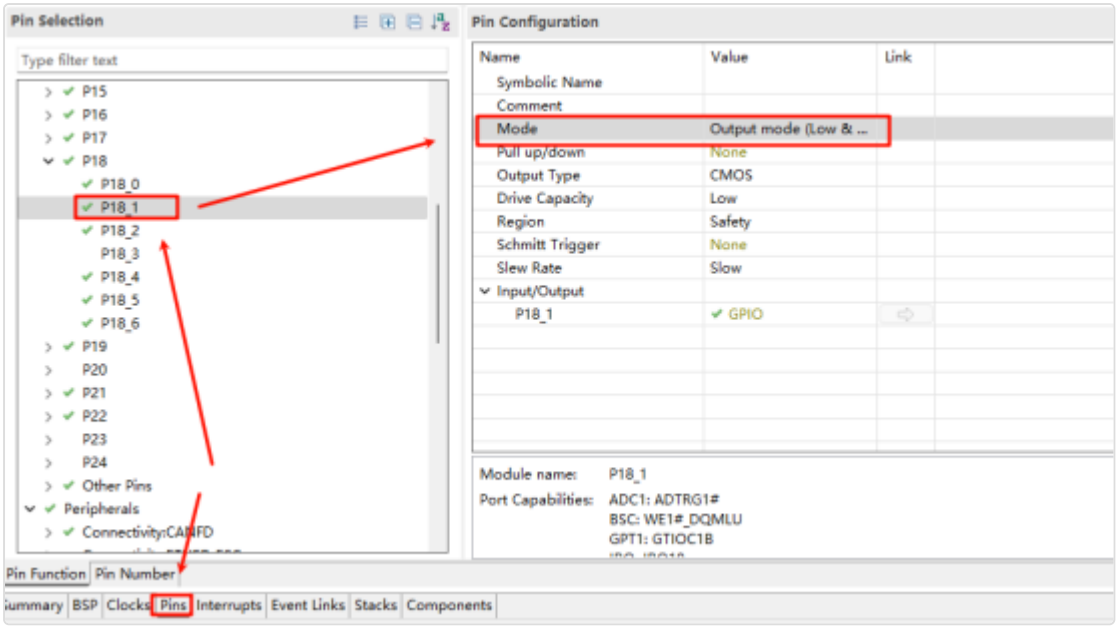


FSP Configuration Instructions

Open the FSP tool, create a new stack, and select `r_sci_spi3`:

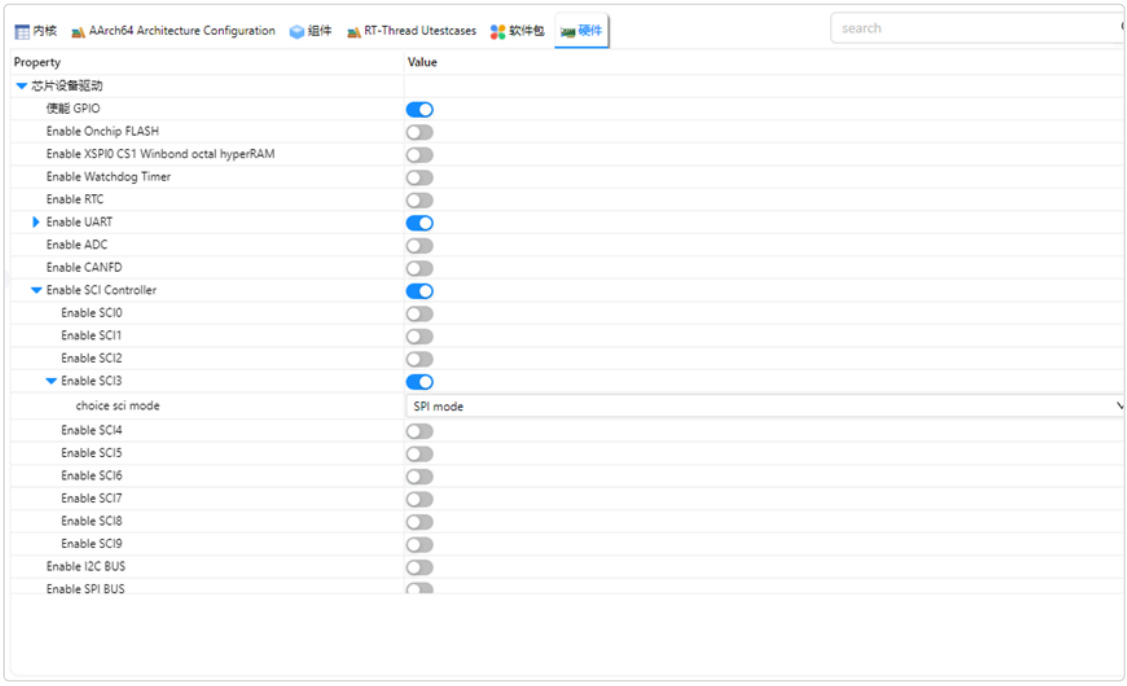

Properties
Problems
g_sci3 SPI (r_sci_spi)
Settings
Property
Value
Common
Parameter Checking
Default (BSP)
DMAC Support
Disabled
Multiplex Interrupt
Disabled
Module g_sci3 SPI (r_sci_spi)
Name
g_sci3
sci name配置
Channel
3
通道配置
Operating Mode
Master
Clock Phase
Data sampling on odd edge, data variation on even edge
Clock Polarity
Low when idle
Bit Order
MSB First
Clock Source
SCInASYNCCLK
Callback
sci_spi_irq_callback
回调函数配置
Receive Interrupt Priority
Priority 12
Transmit Interrupt Priority
Priority 12
优先级配置
Transmit End Interrupt Priority
Priority 12
Error Interrupt Priority
Priority 12
Bitrate
8000000
波特率配置
Pins
CTS_RTS_SS3#
None
RXD_MISO3
P17_7
引脚配置
SCK3
«unavailable»
TXD_MOSI3
P18_0


Pin Selection
Type filter text
P24
Other Pins
Peripherals
Connectivity:CANFD
Connectivity:ETHER_ESC
Connectivity:ETHER_ETH
Connectivity:ETHER_ETHSW
Connectivity:ETHER_GMAC
Connectivity:IIC
Connectivity:PHOSTIF
Connectivity:SCI
SCI0
SCI1
SCI2
SCI3
SCI4
SCI5
Connectivity:SHOSTIF
Connectivity:SPI
Connectivity:USB_HS
Pin Function
Pin Number
ummary
BSP
Clocks
Pins
Interrupts
Event Links
Stacks
Components
Pin Configuration
Name
Value
Lock
Link
Pin Group Selection
Mixed
Operation Mode
Asynchronous mode
Input/Output
CTS3#
None
CTS_RTS3#
None
DE3
None
RXD3
P17_7
None
None
None
TXD3
P18_0
Module name:
SCI3
Usage:
In initial register value, when CCR0.TE bit is 0, and the terminal function is TXDn, the terminal outputs high impe



RT-Thread Settings Configuration

In RT-Thread Settings, enable SCI hardware and configure SCI3 mode as SPI:



Example Project Description

This example uses the RT-Thread SCI driver framework to perform a loopback test on the PMOD interface (connecting PMOD's SPI3_MOSI to SPI3_MISO).

The code is as follows:

```
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_
    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
        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");
}
```

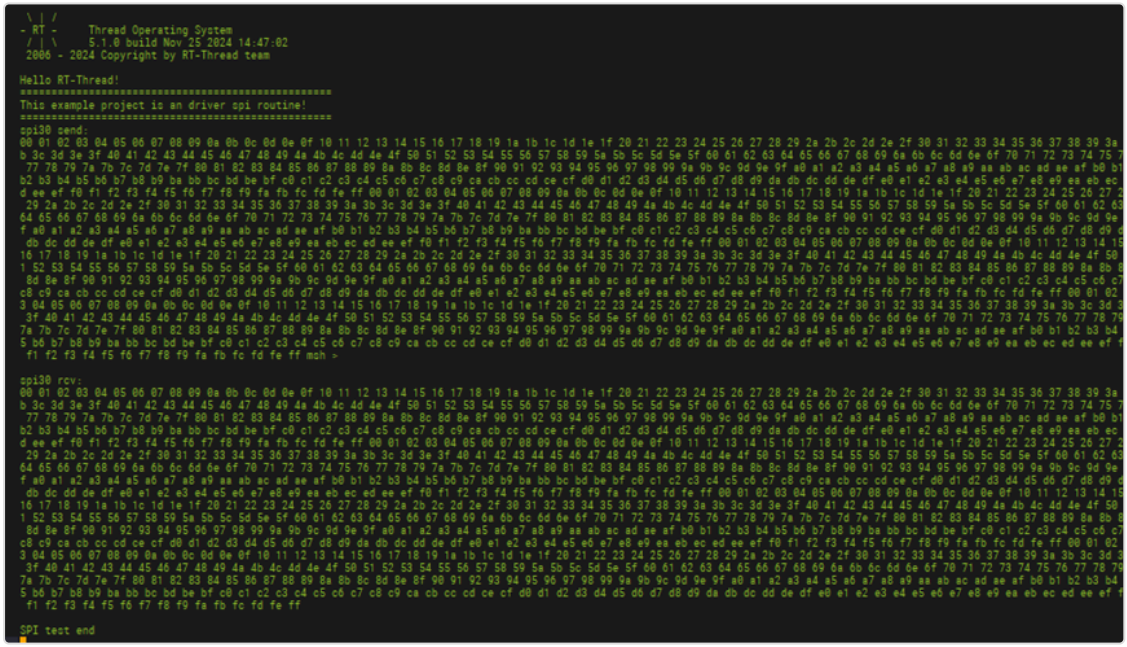
Compilation & Download

- **RT-Thread Studio**: In RT-Thread Studio's package manager, download the EtherKit resource package, create a new project, and compile it.
- **IAR**: First, double-click `mklinks.bat` to create symbolic links between RT-Thread and the libraries folder. Then, use the `Env` tool to generate the IAR project. Finally, double-click `project.eww` to open the IAR project and compile it.

After compilation, connect the development board's JLink interface to the PC and download the firmware to the development board.

Run Effect

When using a serial tool, you can observe that the data sent and received in the SPI loopback test matches:



3.9. WDT Driver Usage Instructions

English | **Chinese**

Introduction

This example mainly introduces how to use the WDT (Watchdog Timer) device on the EtherKit.

Hardware Description

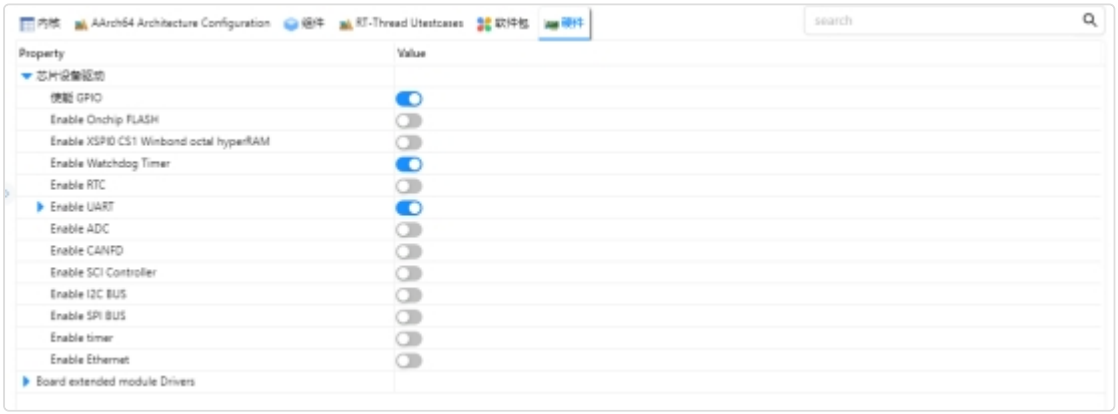
None

Software Description

FSP Configuration Instructions

Open the FSP tool and create a new stack, selecting `r_wdt`.

RT-Thread Settings Configuration



Example project description

By performing a watchdog operation in the idle function, we ensure that our program runs as expected.

```
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, NU
    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);
    }
}
```

Running

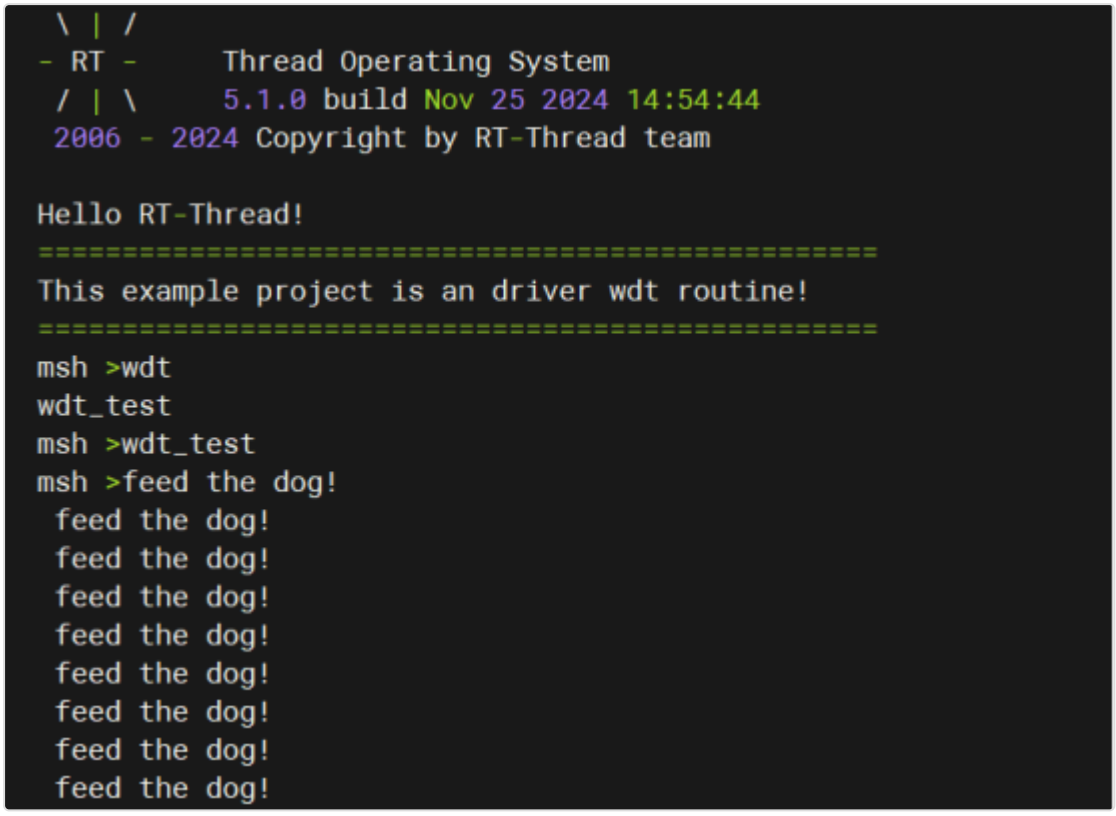
Compilation & Download

RT-Thread Studio: Download the EtherKit resource package in the RT-Thread Studio package manager, then create a new project and compile it.

IAR: First, double-click `mklinks.bat` to generate links for the rt-thread and libraries folders; then use Env to generate the IAR project; finally, double-click `project.eww` to open the IAR project and compile it.

After compilation, connect the Jlink interface of the development board to the PC, and download the firmware to the development board.

Running Effects



Notes

None

References

Device and Driver: WDT Device

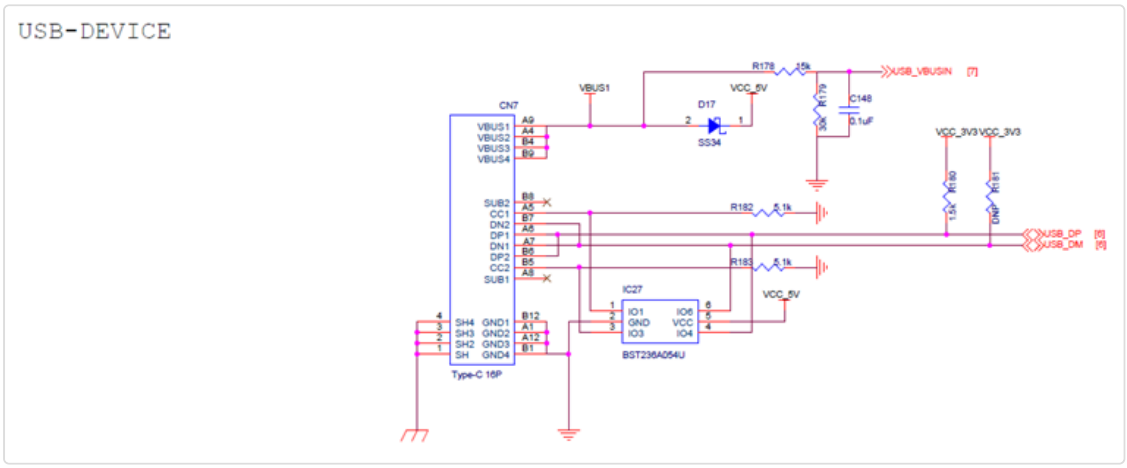
3.10. USB-PCDC Driver Usage Instructions

English | **中文**

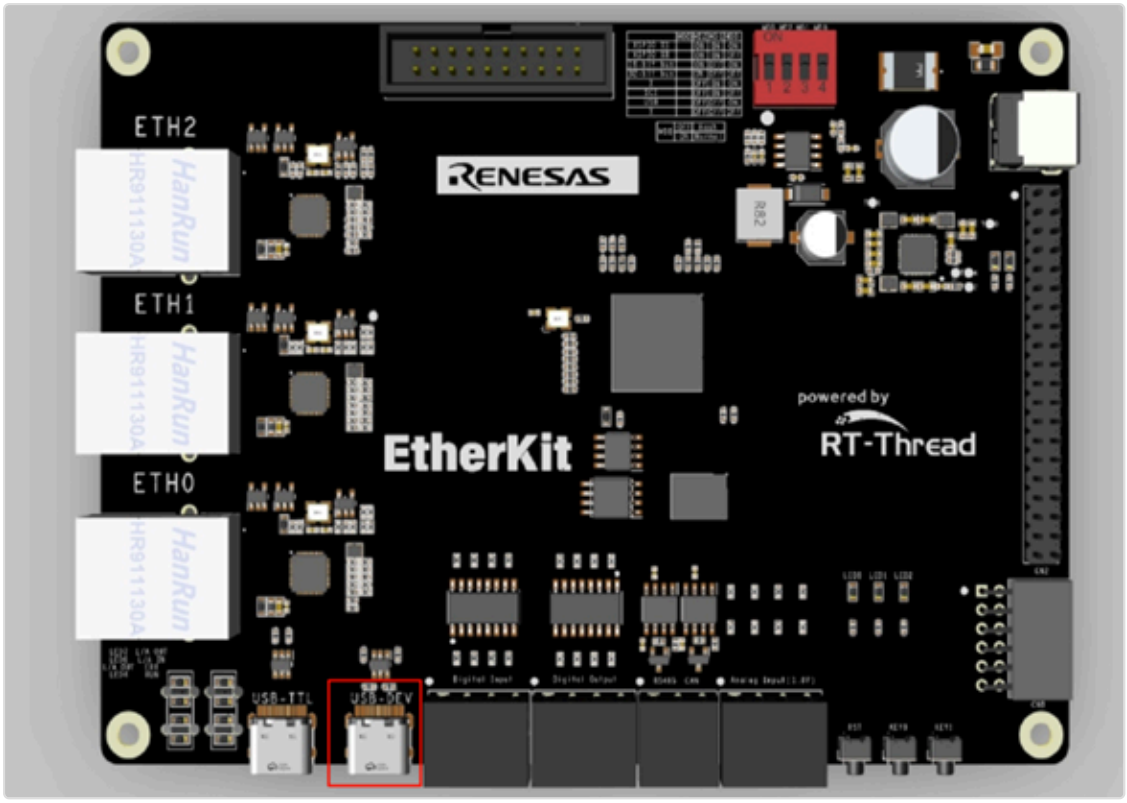
Introduction

This example demonstrates how to implement a virtual serial port over USB. USB PCDC (USB Communication Device Class) is a subclass of the USB communication device class (CDC) that is commonly used to implement virtual serial communication functionality. In embedded device development, USB PCDC is often used to simulate a serial communication port (such as a COM port) over a USB interface, allowing data interaction with a host.

Hardware Requirements

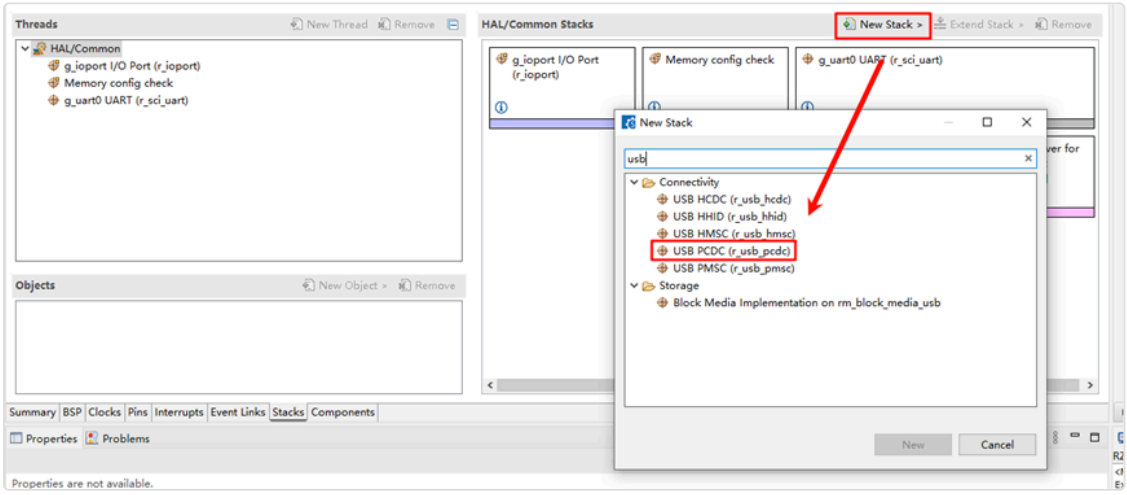


EtherKit provides a USB-Device peripheral, located on the development board as shown below:

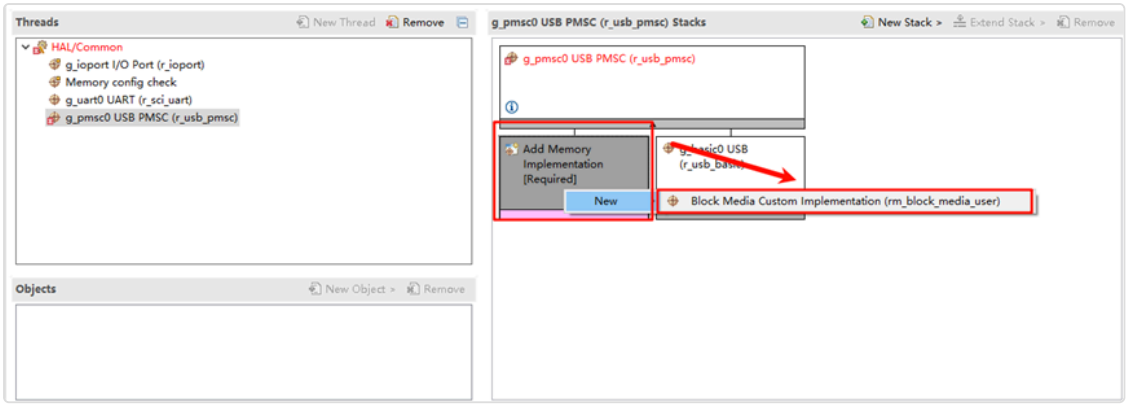


FSP Configuration Instructions

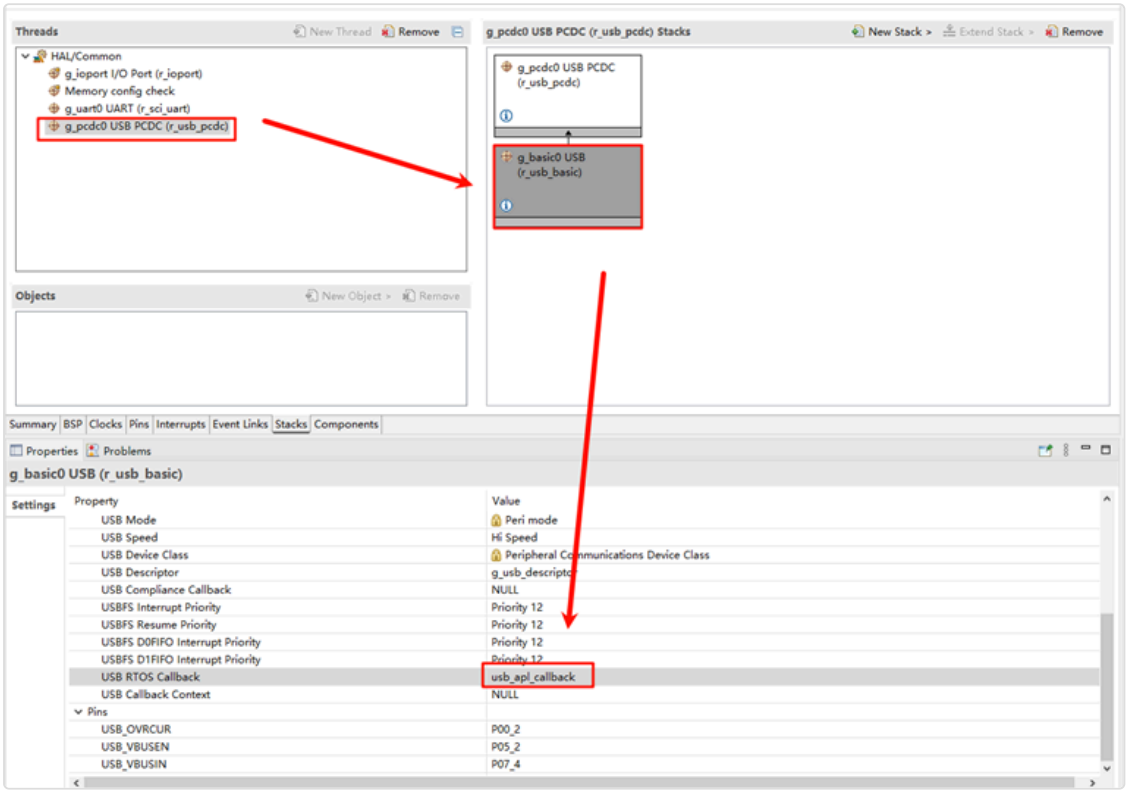
- Open the project's `configuration.xml` file using FSP and add the `usb_pmcs` stack:



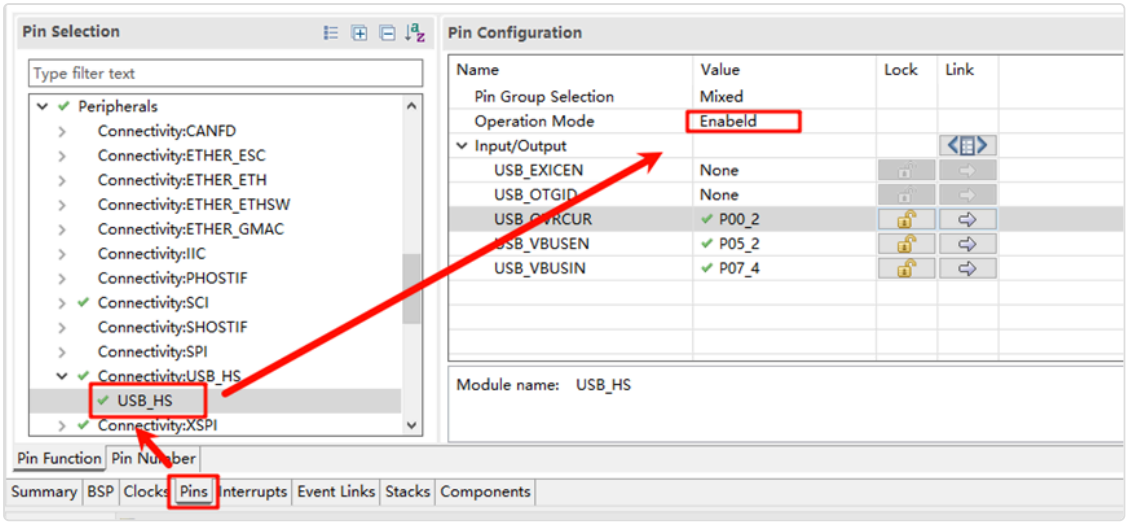
- Add `g_rm_block_media0` :



- Select `g_basic0_usb` and set the interrupt callback function to `usb_apl_callback`:



- Next, configure the USB pins. Find `USB_HS` and enable it:



Build Configuration

Locate the file at the specified path in the project: `.\rzn\SConscript`, and replace its content with the following:

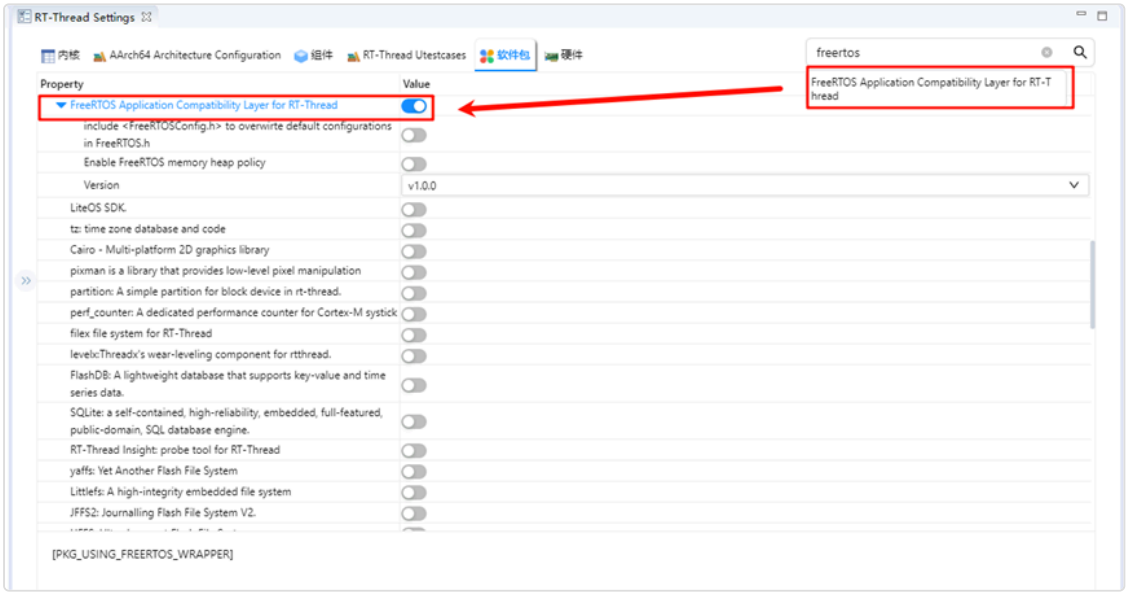
```
Import('RTT_ROOT')
Import('rtconfig')
from building import *
from gcc import *

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

if rtconfig.PLATFORM in ['iccarm'] + GetGCCLikePLATFORM():
    if rtconfig.PLATFORM == 'iccarm' or GetOption('target') !=
        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/
        src += Glob('./fsp/src/bsp/cmsis/Device/RENESAS/Source/
        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 = CPPPAT
Return('group')
```

If you are using Studio for development, right-click on the project and select **Sync SCons Configuration to Project**.
For IAR development, right-click within the current project to open the environment, and run the following command to regenerate the configuration:
`scons --target=iar`

RT-Thread Settings Configuration

The USB example currently uses the FreeRTOS interface driver, so we also need to enable the FreeRTOS compatibility layer package:



Compilation & Download

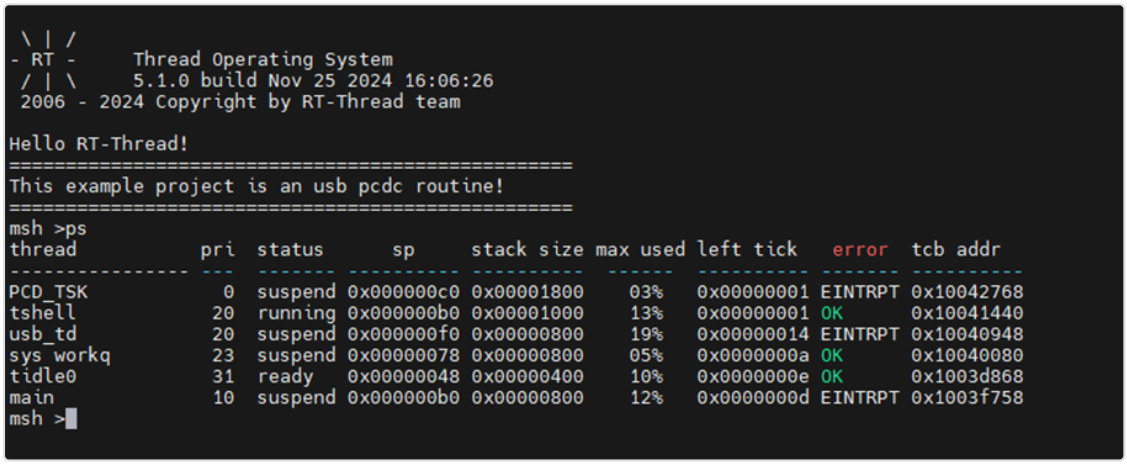
- **RT-Thread Studio**: Download the EtherKit resource package in the RT-Thread Studio package manager, create a new project, and compile it.
- **IAR**: First, double-click `mklinks.bat` to create the link between the rt-thread and libraries folders. Then, use Env to generate the IAR project.

Finally, double-click `project.eww` to open the IAR project and compile it.

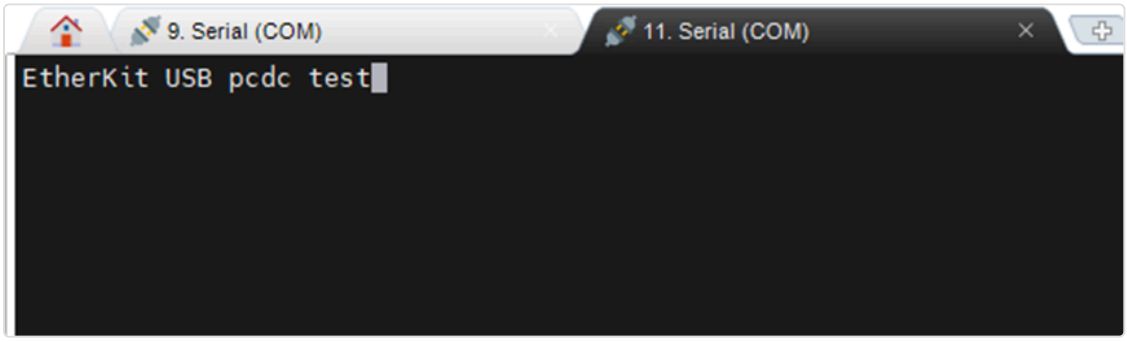
Once the compilation is complete, connect the Jlink interface of the development board to the PC and download the firmware to the board.

Running Results

After generating the FSP configuration, compile and download the firmware to the development board. The example will automatically start, and you will see an additional USB device in the file manager:



Next, open the virtual serial port device and test character input:



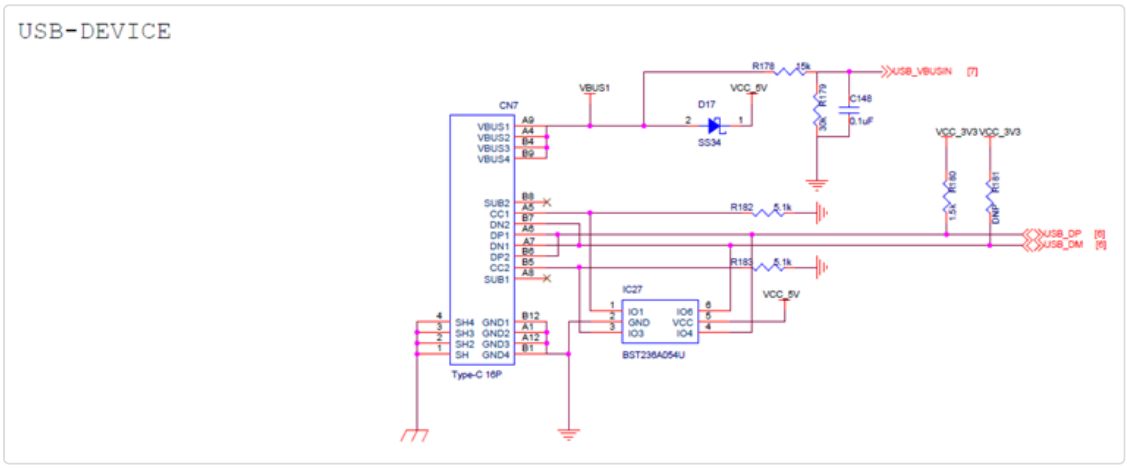
3.11. USB-PMSC Driver Usage Instructions

English | 中文

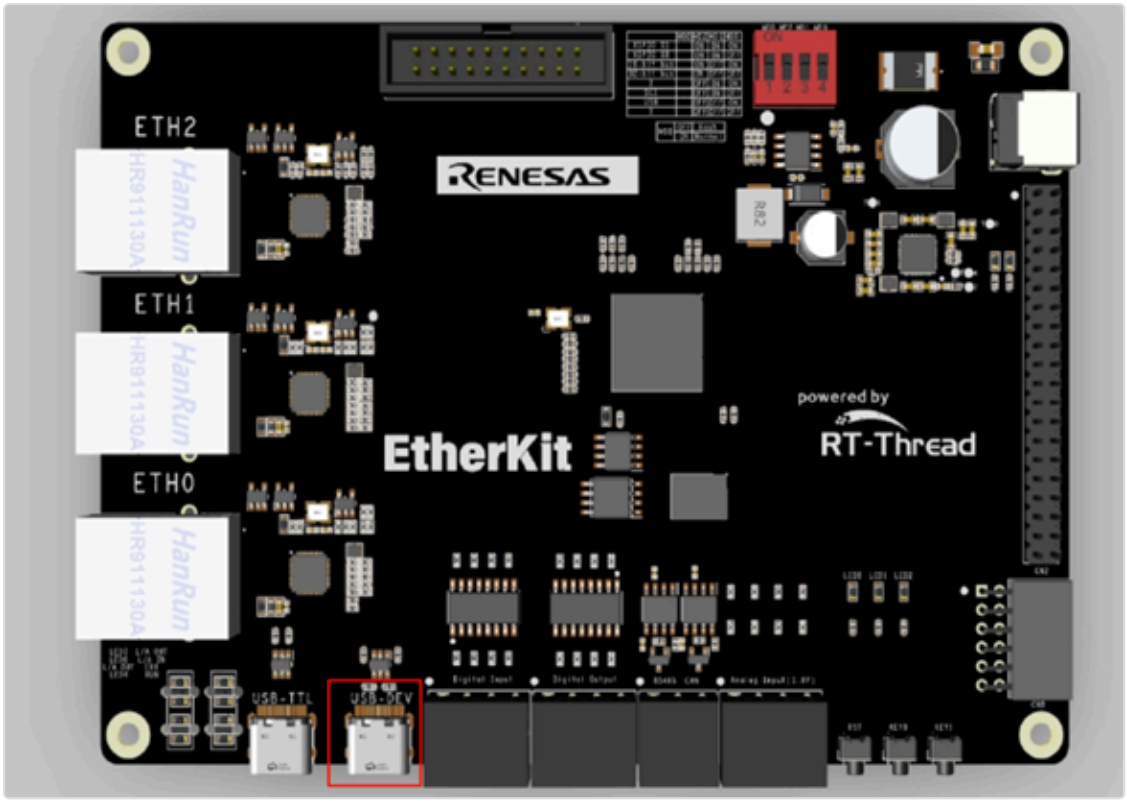
Introduction

This example demonstrates how to implement a USB flash drive using the USB-PMSC (USB Peripheral Mass Storage Class). USB-PMSC is a USB device class that enables USB-based storage device functionality.

Hardware Requirements

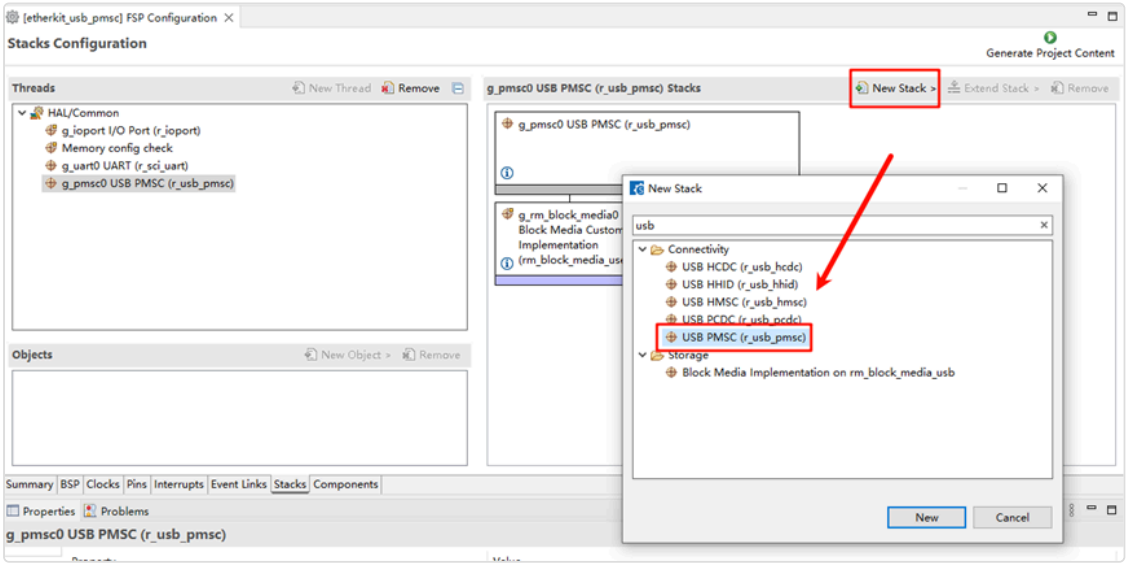


EtherKit provides a USB-Device peripheral, located on the development board as shown below:

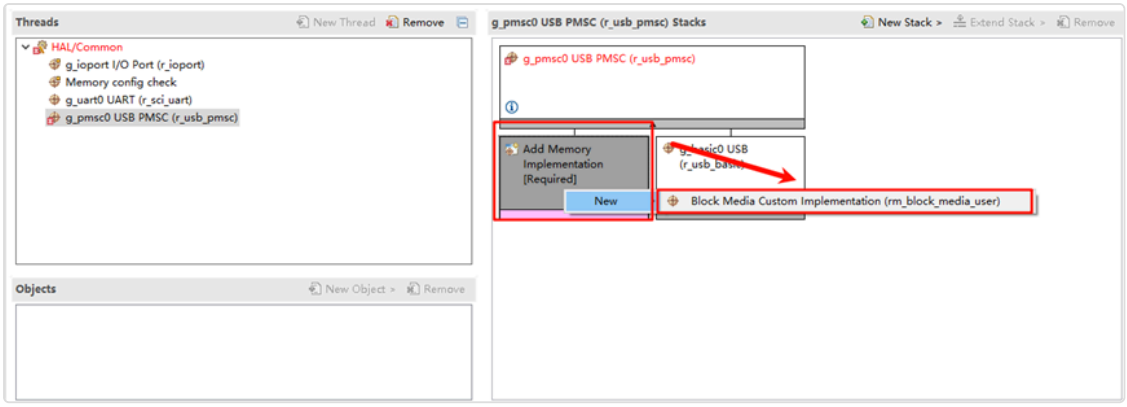


FSP Configuration Instructions

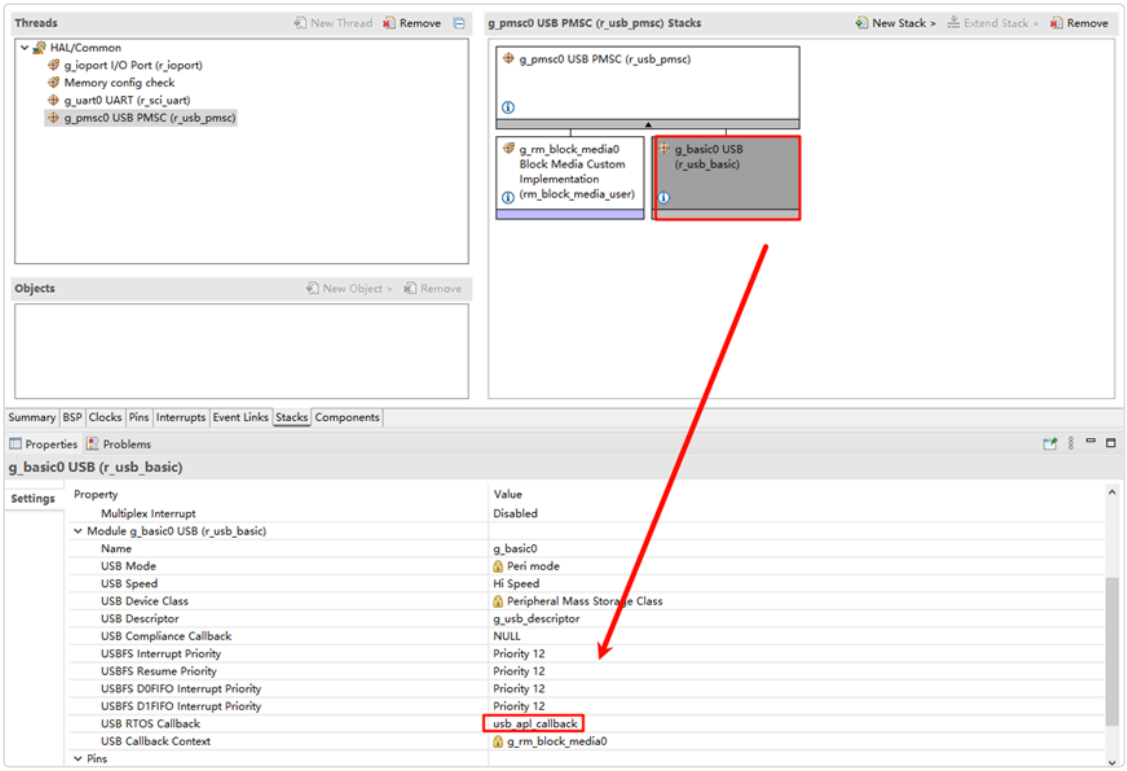
- Open the project's `configuration.xml` file using FSP and add the `usb_pmcs` stack:



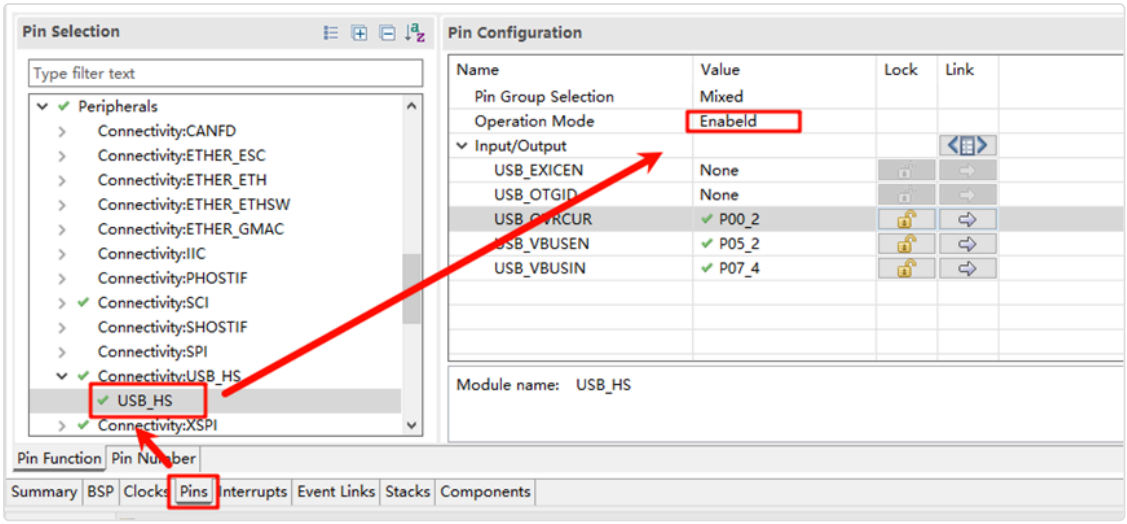
- Add `g_rm_block_media0` :



- Select `g_basic0_usb` and set the interrupt callback function to `usb_apl_callback`:



- Next, configure the USB pins. Find `USB_HS` and enable it:



Build Configuration

Locate the file at the specified path in the project: `.\rzn\SConscript` and replace its content with the following:

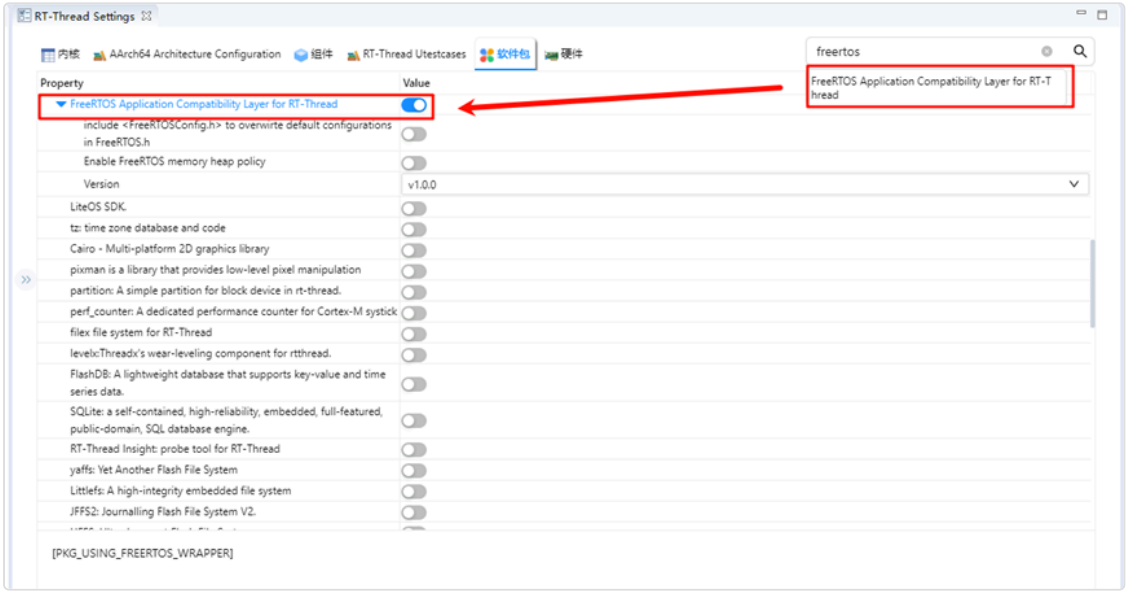
```
Import('RTT_ROOT')
Import('rtconfig')
from building import *
from gcc import *

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

if rtconfig.PLATFORM in ['iccarm'] + GetGCCLikePLATFORM():
    if rtconfig.PLATFORM == 'iccarm' or GetOption('target') !=
        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/
        src += Glob('./fsp/src/bsp/cmsis/Device/RENESAS/Source/
        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/dri
                                cwd + '/fsp/src/r_usb_basic/src/hw/
                                cwd + '/fsp/src/r_usb_pmsc/src/inc'

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

If you are using Studio for development, right-click the project and select **Sync SCons Configuration to Project**. For IAR development, right-click within the current project to open the environment, and run the command: `scons --target=iar` to regenerate the configuration.

RT-Thread Settings Configuration

The USB example currently uses the FreeRTOS interface driver, so we also need to enable the FreeRTOS compatibility layer package:



Compilation & Download

- **RT-Thread Studio**: Download the EtherKit resource package in the RT-Thread Studio package manager, create a new project, and compile it.
- **IAR**: First, double-click `mklinks.bat` to create the link between the rt-thread and libraries folders. Then, use Env to generate the IAR project. Finally, double-click `project.eww` to open the IAR project and compile it.

Once the compilation is complete, connect the Jlink interface of the development board to the PC and download the firmware to the board.

Running Results

After generating the FSP configuration, compile and download the firmware to the development board. The example will automatically start. In the file manager, you will see an additional USB device:

```
 \ | /
- 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 >
```

4. Components

4.1. Filesystem Usage Instructions

English | **中文**

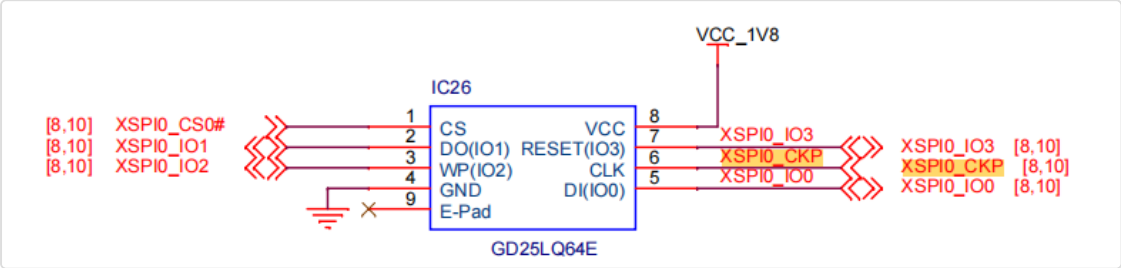
Introduction

FAL (Flash Abstraction Layer) is an abstraction layer for managing and operating Flash devices and Flash-based partitions. It provides a unified API for upper-layer Flash and partition operations (as shown in the framework diagram below), with the following features:

- Supports a statically configurable partition table, which can be associated with multiple Flash devices.
- The partition table supports **automatic loading**, avoiding the issue of the partition table being repeatedly defined in multi-firmware projects.
- Lightweight code with **no dependency on an operating system**, allowing it to run on bare-metal platforms, such as bootloaders with limited resources.
- A unified operation interface ensures the reusability of underlying Flash drivers for components that rely on Flash, such as file systems, OTA, and NVM (e.g., EasyFlash).
- Built-in Finsh/MSH-based test commands, enabling developers to read, write, and erase Flash or partitions via shell commands with byte-level addressing, which facilitates debugging and testing.

In this example, the EtherKit onboard GD Flash is combined with the RT-Thread FAL component to build a file system using littlefs.

Hardware Overview



Software Overview

The source code for file system initialization in this example is located at: `../board/ports/filesystem/drv_filesystem.c`

```
/*
 * Copyright (c) 2006-2021, RT-Thread Development Team
 *
 * SPDX-License-Identifier: Apache-2.0
 *
 * Change Logs:
 * Date           Author                Notes
 * 2025-02-07     newflydd@gmail.com    the first version
 */
#define DBG_TAG "drv.fs"
```

```
#define DBG_LVL DBG_LOG
#include <rtdbg.h>

#include <rtthread.h>
#include <fal.h>
#include <dfs_fs.h>

int initFileSystem()
{
    fal_init();

    // register onchip flash tail area as littlefs
    struct rt_device* flashDev = fal_mtd_nor_device_create("par
    if (RT_NULL == flashDev)
    {
        LOG_W("create fal device failed");
        return RT_ERROR;
    }

    if (RT_EOK != dfs_mount("param", "/", "lfs", 0, RT_NULL))
    {
        LOG_W("mount lfs failed once, try to format it");
        if (RT_EOK != dfs_mkfs("lfs", "param"))
        {
            LOG_W("mkfs lfs failed");
            return RT_ERROR;
        }
        LOG_I("mkfs lfs success");

        if (RT_EOK != dfs_mount("param", "/", "lfs", 0, RT_NULL
        {
            LOG_W("mount lfs failed");
            return RT_ERROR;
        }
    }
    LOG_I("mount lfs success");
    return RT_EOK;
}
INIT ENV EXPORT(initFileSystem);
```

Build & Download

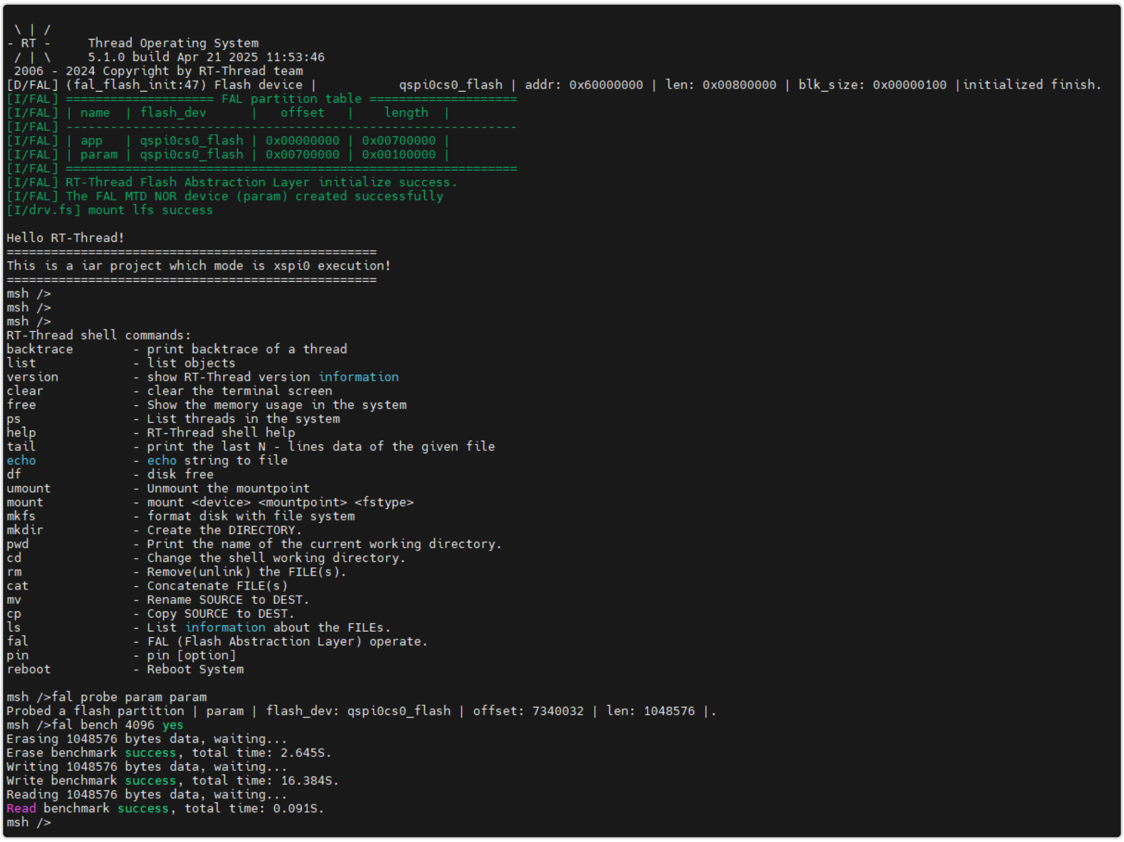
- **RT-Thread Studio:** Download the EtherKit resource pack from the RT-Thread Studio package manager, then create a new project and compile it.

- **IAR:** First, double-click `mklinks.bat` to generate links for the `rt-thread` and `libraries` folders. Then, use Env to generate the IAR project. Finally, double-click `project.eww` to open the IAR project and compile it.

Once compiled, connect the development board's JLink interface to the PC, and download the firmware to the development board.

Running Result

Press the reset button to restart the development board and observe the terminal logs from the board.



Run the following commands to start the FAL read/write test:

```
> fal probe param param
> fal bench 4096 yes
```

4.2. MQTT Usage Instructions

English | **中文**

Introduction

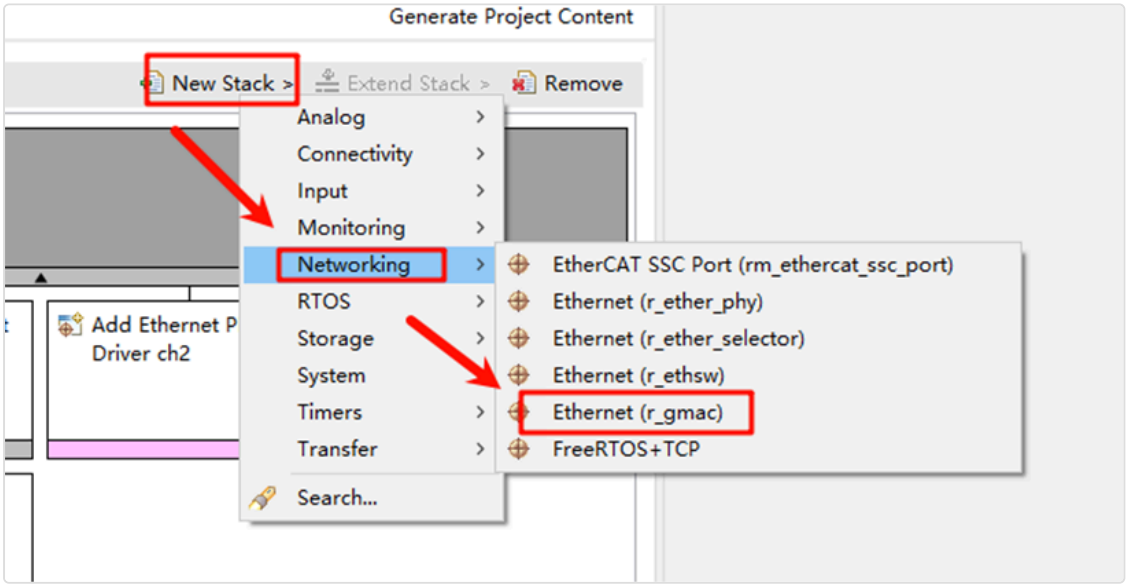
This example is based on the kawaii-mqtt package and demonstrates the functionality of subscribing to topics and publishing messages to specific topics using the MQTTX software.

Hardware Requirements

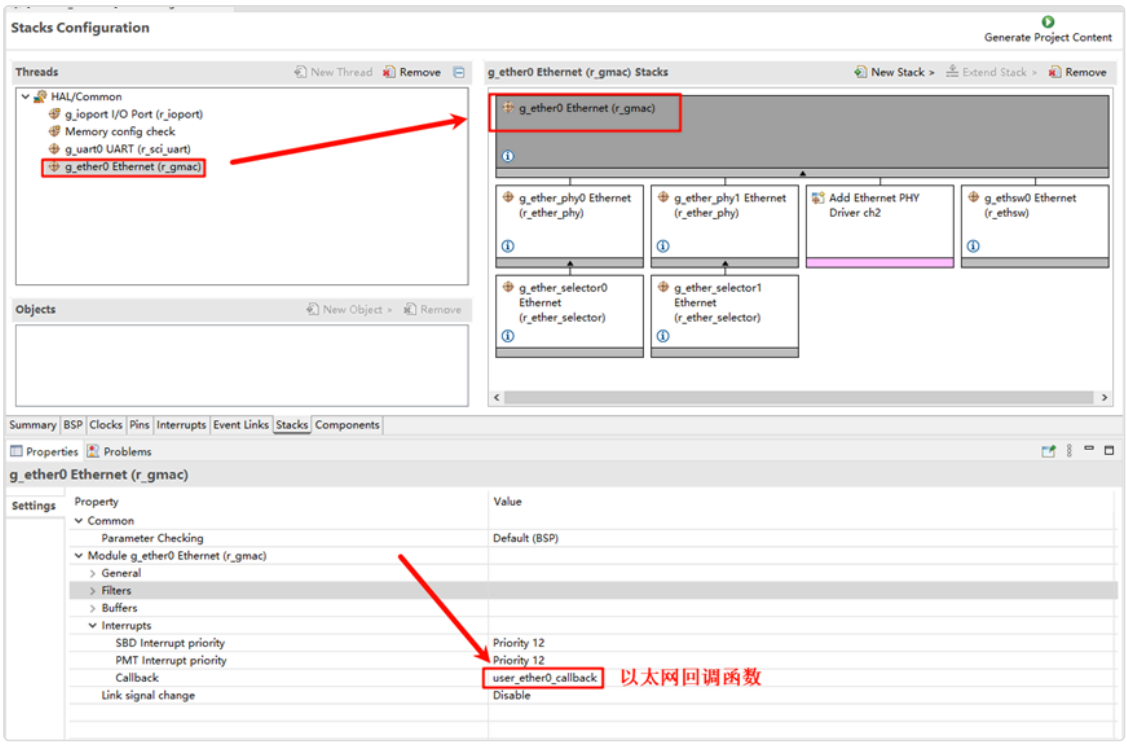
This example requires the Ethernet module on the EtherKit board for network communication, so please ensure that the Ethernet module on the hardware platform is functioning properly.

FSP Configuration Instructions

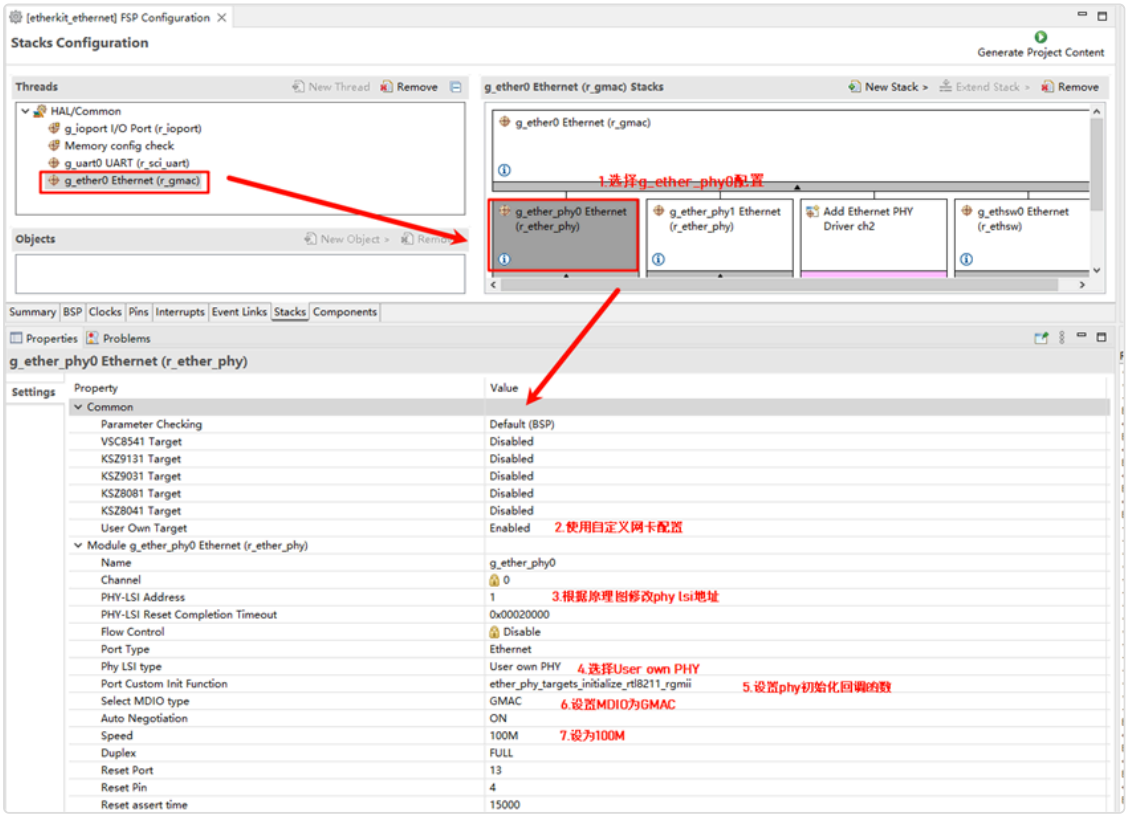
Open the project configuration file `configuration.xml` and add the `r_gamc` stack:



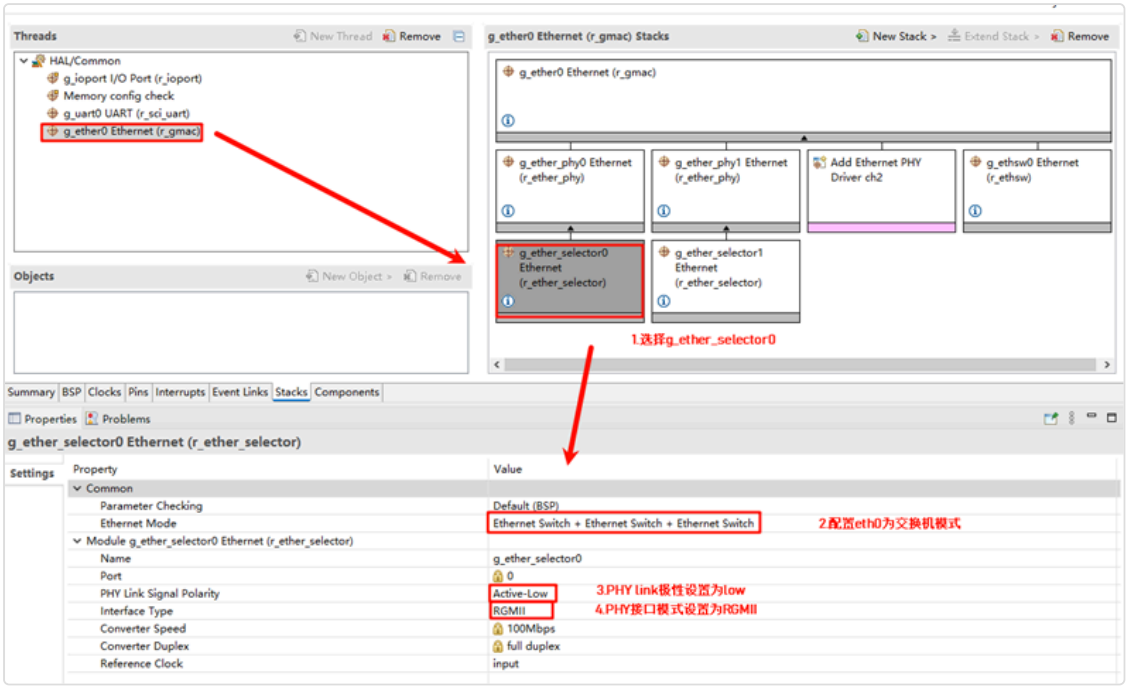
Click on `g_ether0` Ethernet and configure the interrupt callback function as `user_ether0_callback`:



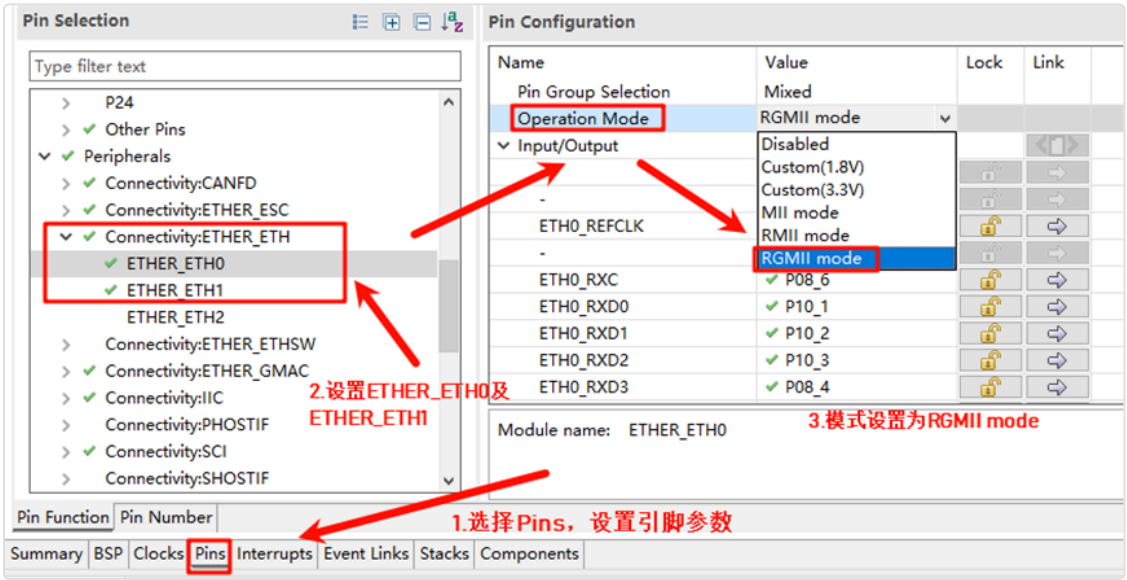
Next, configure the PHY information. Select `g_ether_phy0`, set the Common configuration to "User Own Target", change the PHY LSI address to 1 (refer to the schematic for the specific address), and set the PHY initialization callback function to `ether_phy_targets_initialize_rtl8211_rgmii()`. Also, set MDIO to GMAC.



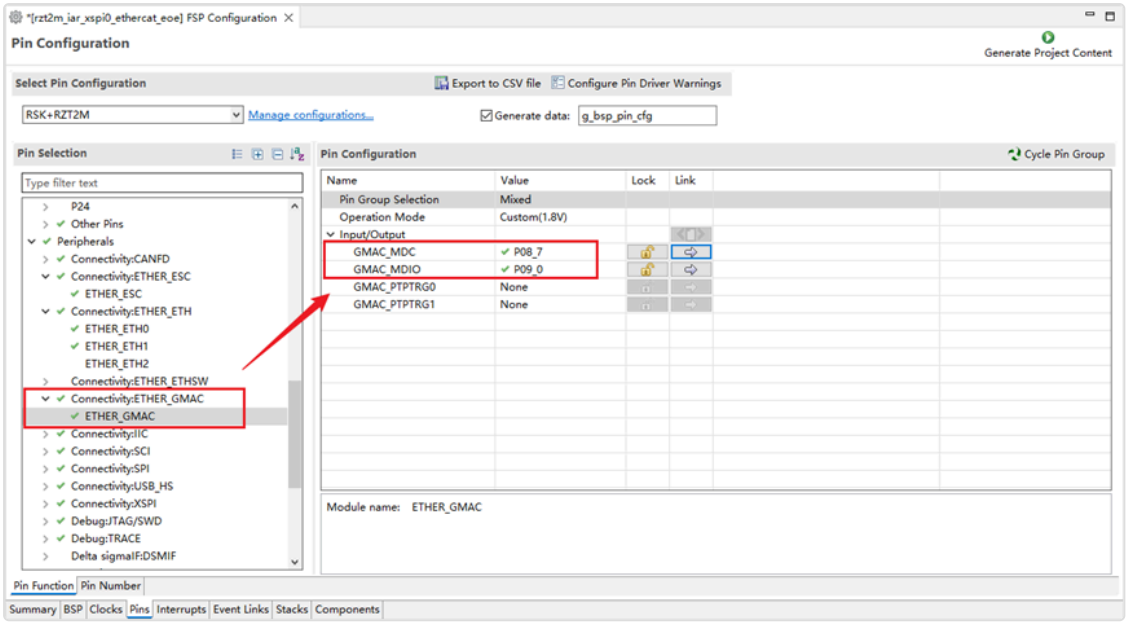
Configure `g_ether_selector0`, set the Ethernet mode to switch mode, configure PHY link as default active-low, and set the PHY interface mode to RGMII.



Configure the network card pin parameters, selecting the operation mode as RGMII:

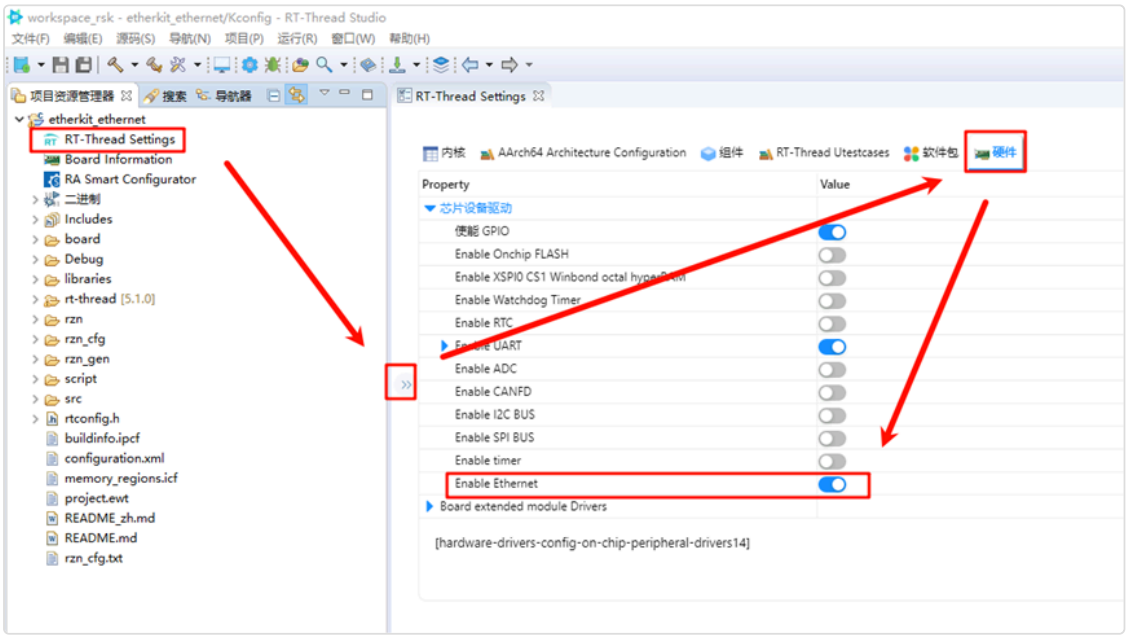


Configure `ETHER_GMAC`:

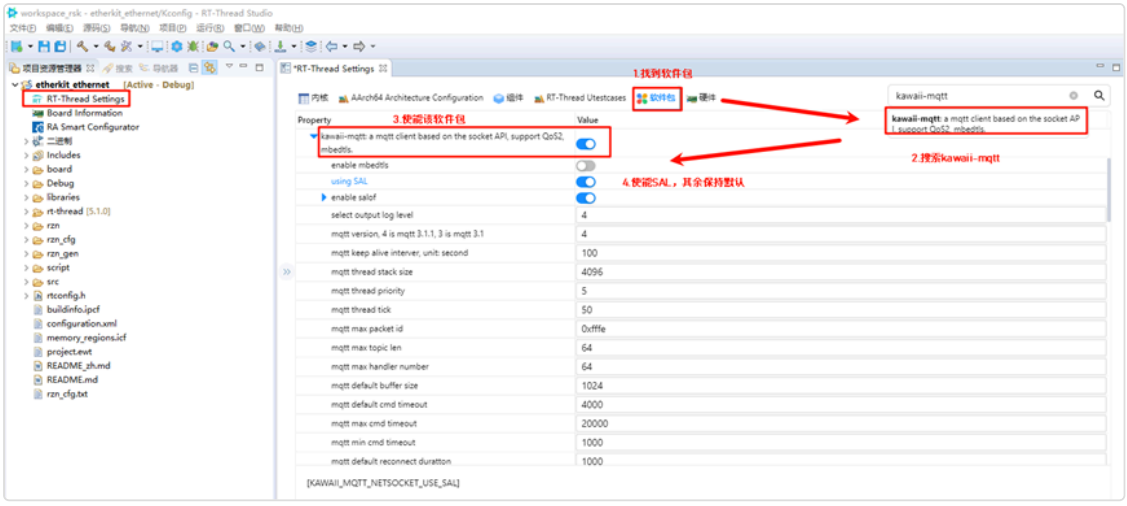


RT-Thread Studio Configuration

Return to the Studio project, configure RT-Thread Settings, select hardware options, and enable Ethernet:



In the software package interface, search for the kawaii-mqtt package and enable the SAL option:



Example Code Explanation

This code implements an MQTT communication demonstration program based on the Kawaii MQTT client library. It connects to an MQTT broker, subscribes to topics, and periodically publishes messages.

```
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", _
    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
    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);
```

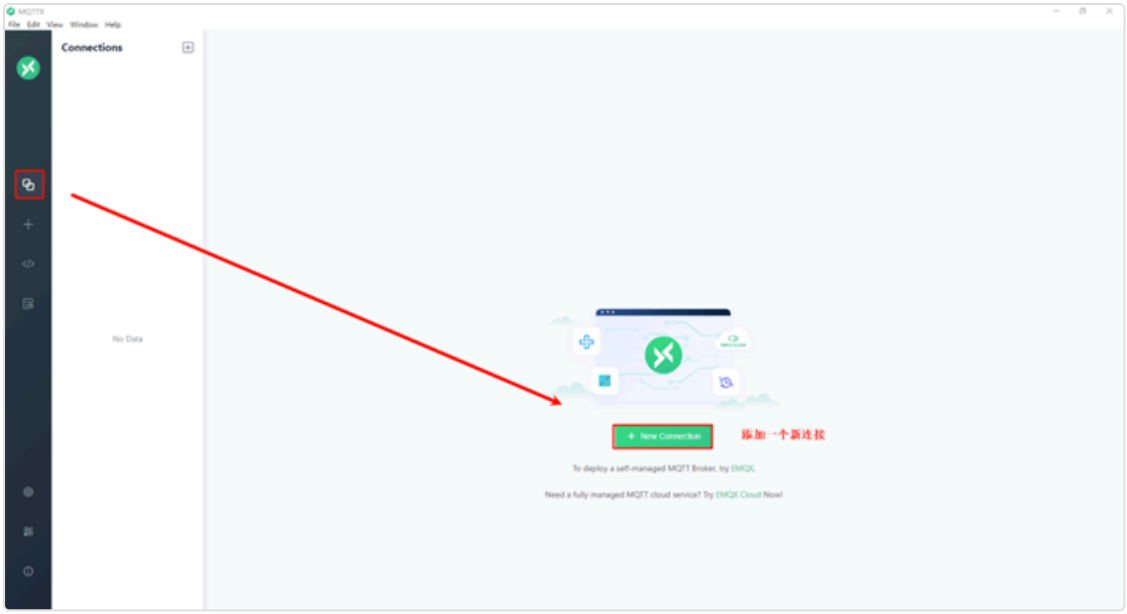
Compilation & Download

- **RT-Thread Studio**: Download the EtherKit resource package in RT-Thread Studio's package manager, create a new project, and then compile it.
- **IAR**: First, double-click `mklinks.bat` to create the link between rt-thread and libraries folders. Then, use Env to generate the IAR project. Finally, double-click `project.eww` to open the IAR project and compile it.

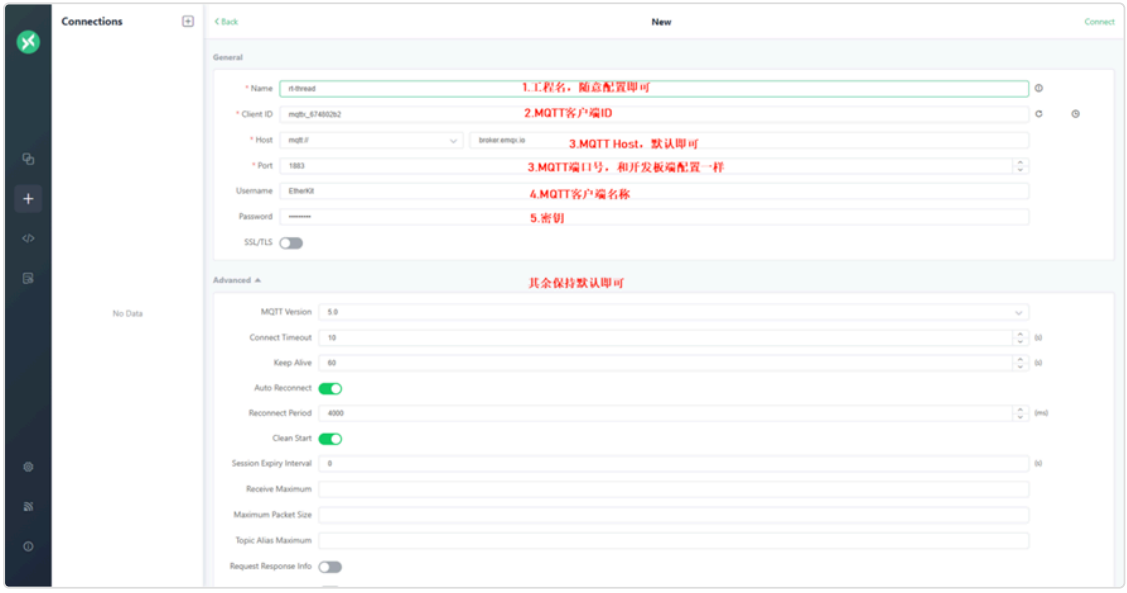
Once compilation is complete, connect the development board's Jlink interface to the PC, and download the firmware to the board.

MQTTX Configuration

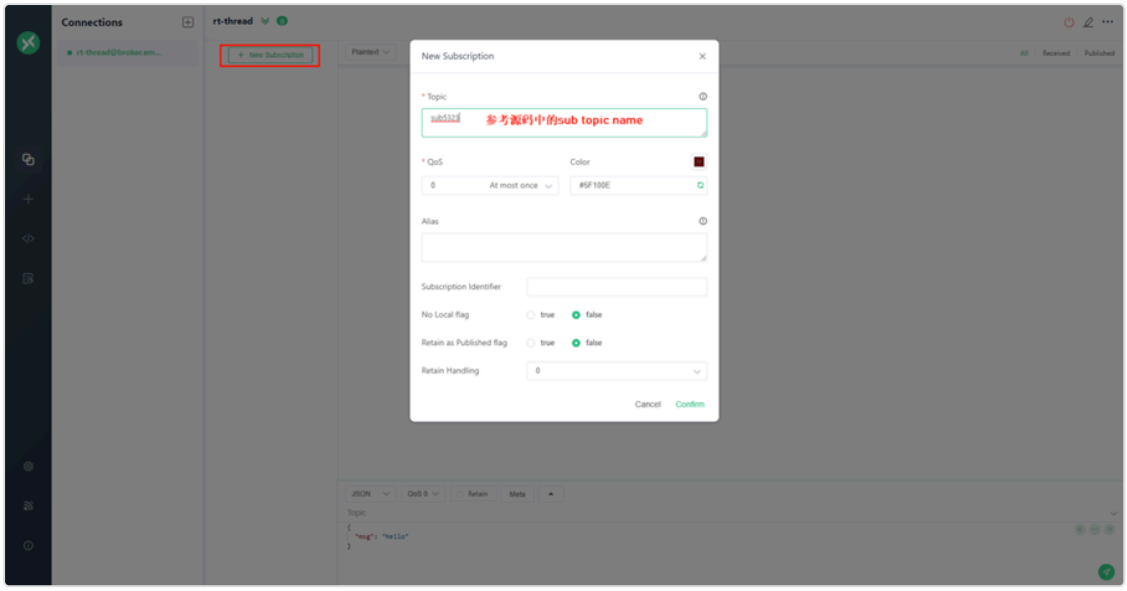
Install and run MQTTX. On the main interface, click **New Connection** to create a new connection:



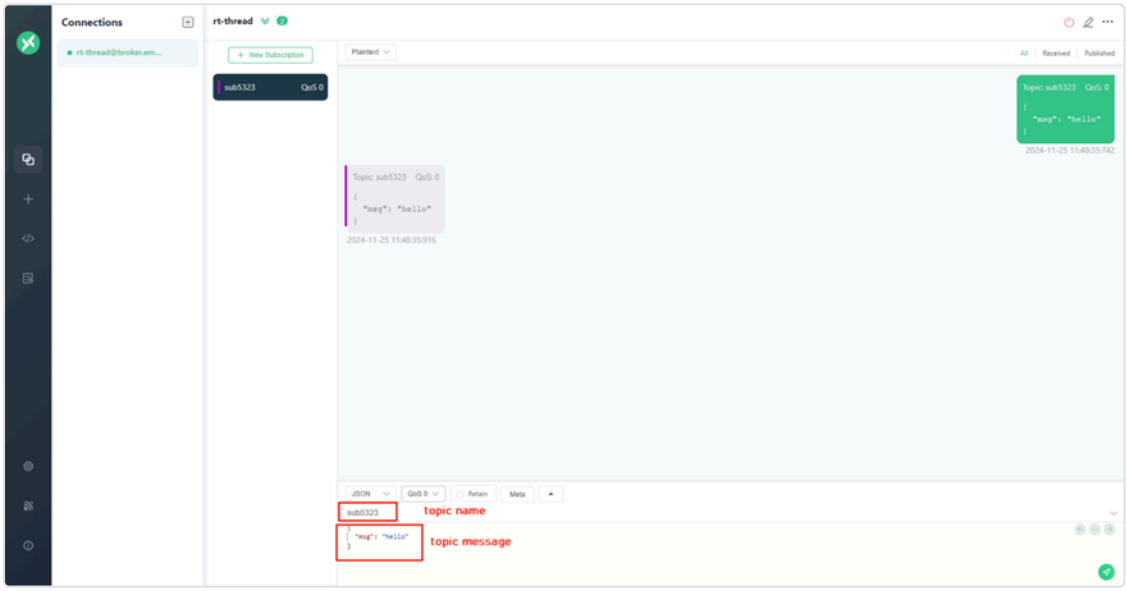
Configure the MQTT client information. Note that the Client ID should not be the same as the one on the development board. You can generate a random ID by clicking the reset button. After configuring the settings, click **Connect** in the top right corner:



Click **+ New Subscription**, change the Topic name to `sub5323`, and confirm:

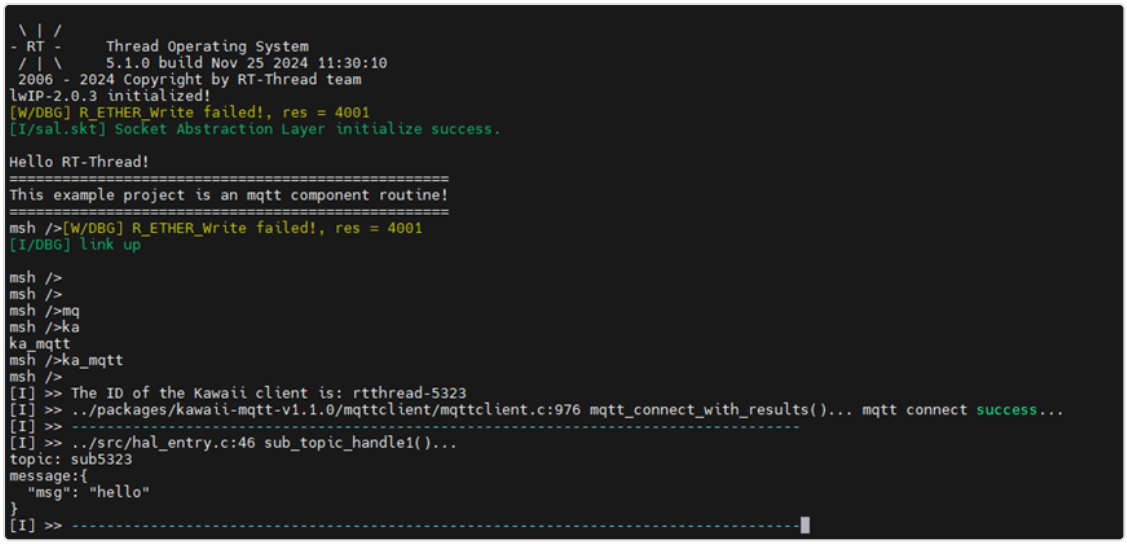


In the function box below, set the subscription topic name to `sub5323` and configure the subscription settings as needed:



Running Results

Open a serial tool and run the `ka_mqtt` command to check the output:



Additional Notes

> *MQTTX download link: https://packages.emqx.net/MQTTX/v1.9.6/MQTTX-Setup-1.9.6-x64.exe*

4.3. Netutils Usage Instructions

English | **中文**

Introduction

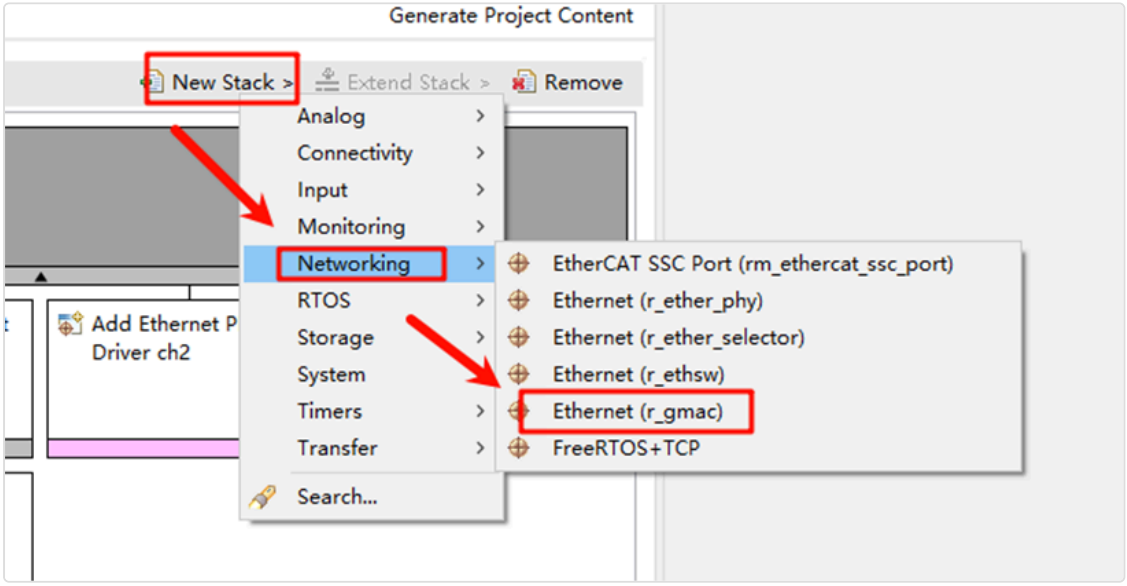
This project provides basic Ethernet functionalities, including `ping`, `tftp`, `ntp`, and `iperf`.

Hardware Connection

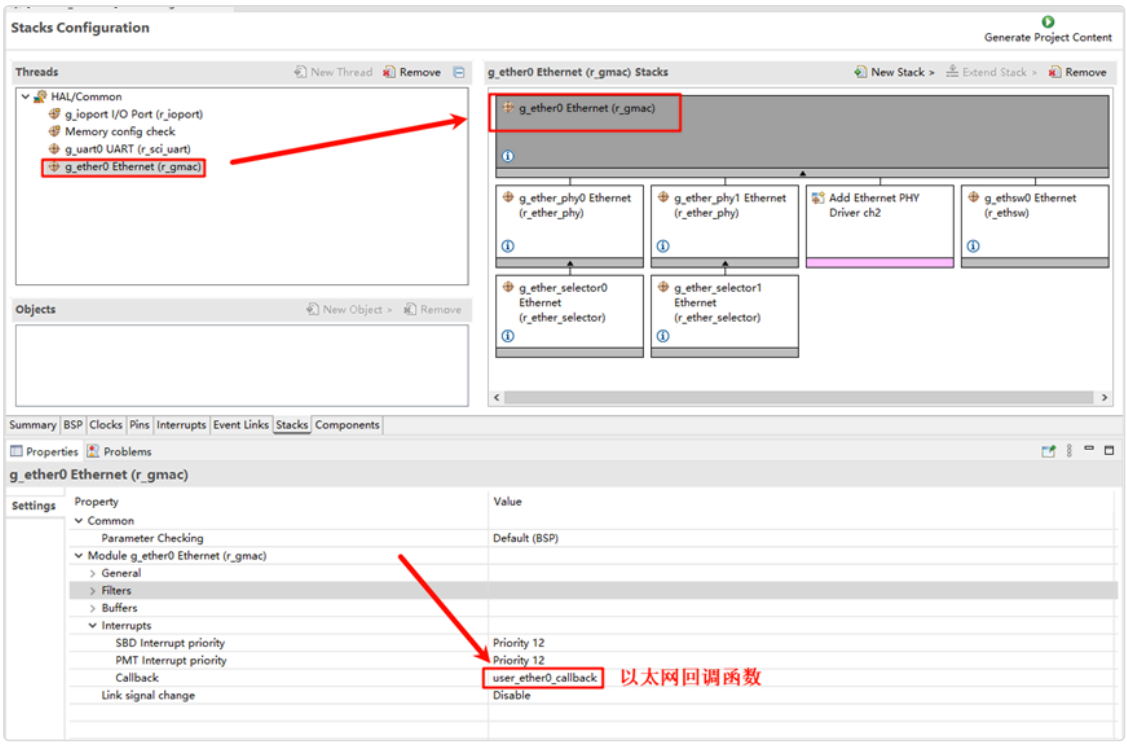
To use Ethernet, connect the development board to any one of the three network ports using an Ethernet cable, and the other end should be connected to a network switch that has internet access.

FSP Configuration Instructions

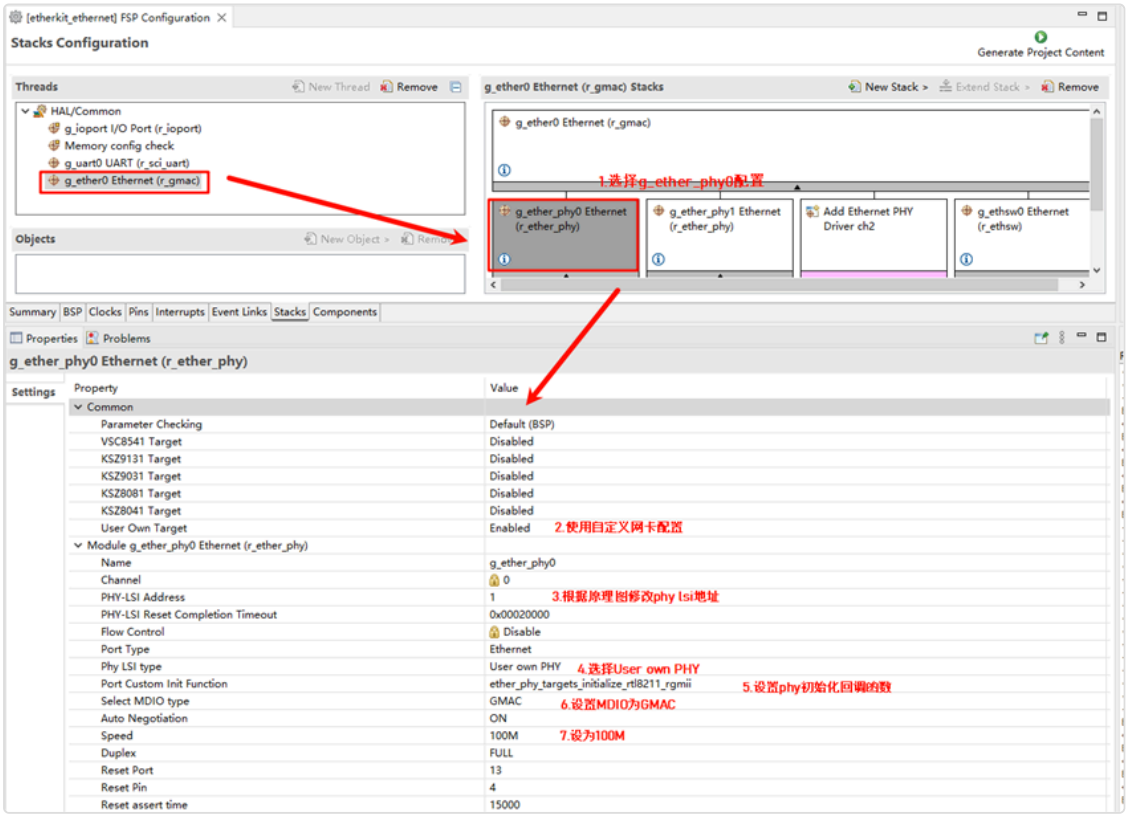
Open the project configuration file `configuration.xml` and add the `r_gamc` stack:



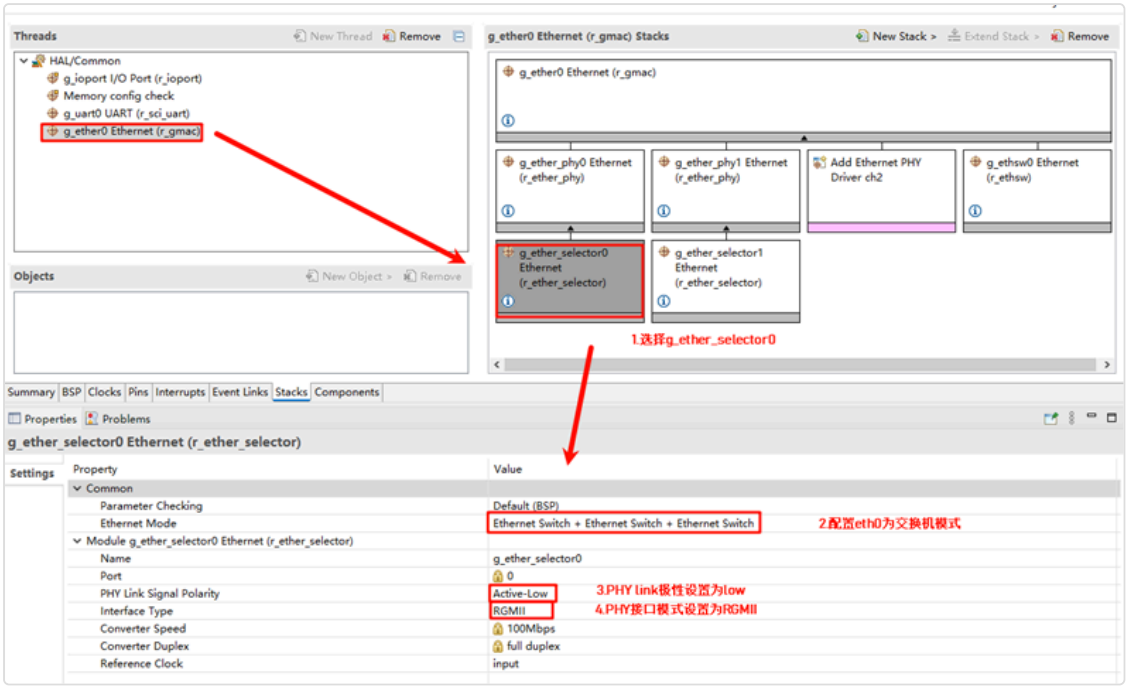
Next, click on `g_ether0 Ethernet`, and configure the interrupt callback function to `user_ether0_callback`:



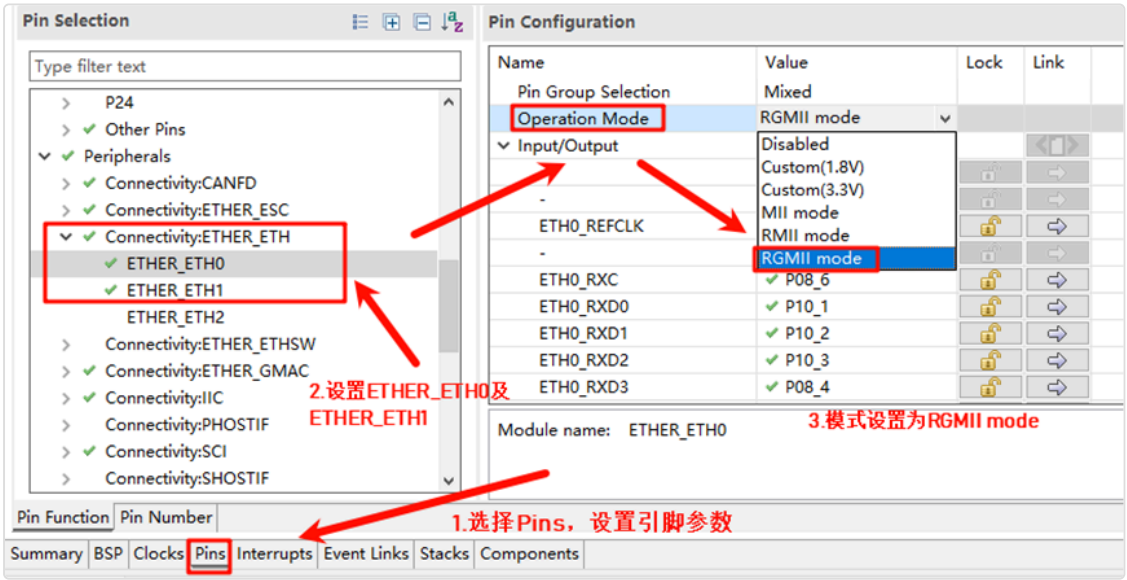
Now configure the PHY settings. Select `g_ether_phy0`, set the common configuration to "User Own Target", change the PHY LSI address to `1` (refer to the schematic for the exact address), and set the PHY initialization callback function to `ether_phy_targets_initialize_rtl8211_rgmii()`. Also, set the MDIO to GMAC.



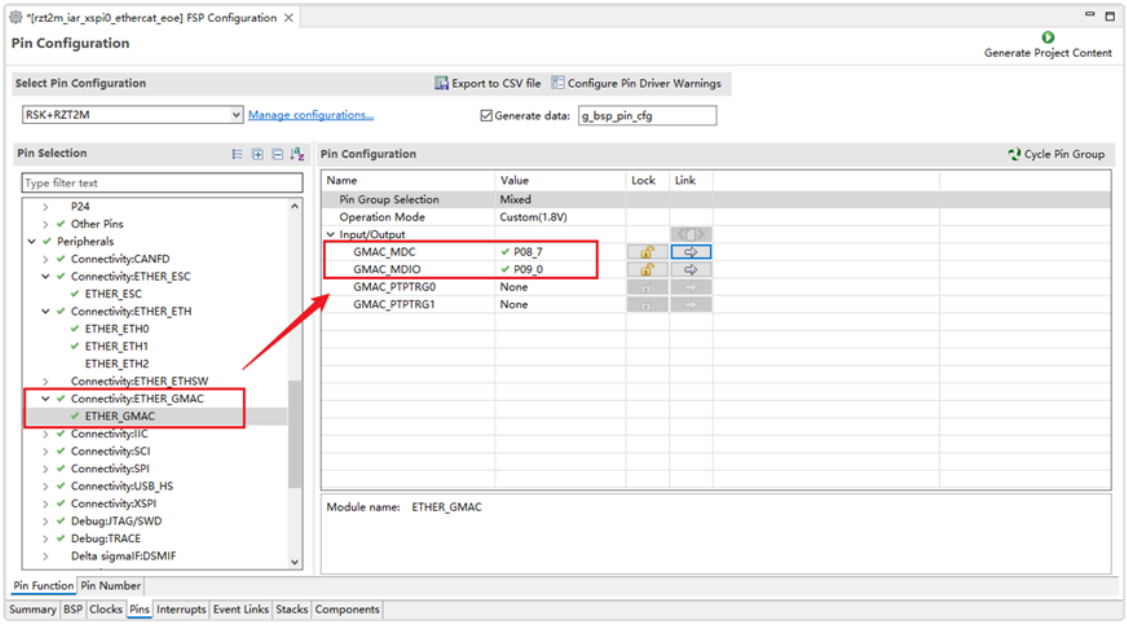
Next, configure `g_ether_selector0`, set the Ethernet mode to "Switch Mode", set the PHY link to "Default Active-Low", and choose "RGMII" for the PHY interface mode.



Configure the Ethernet pin parameters and select the operating mode to RGMII:

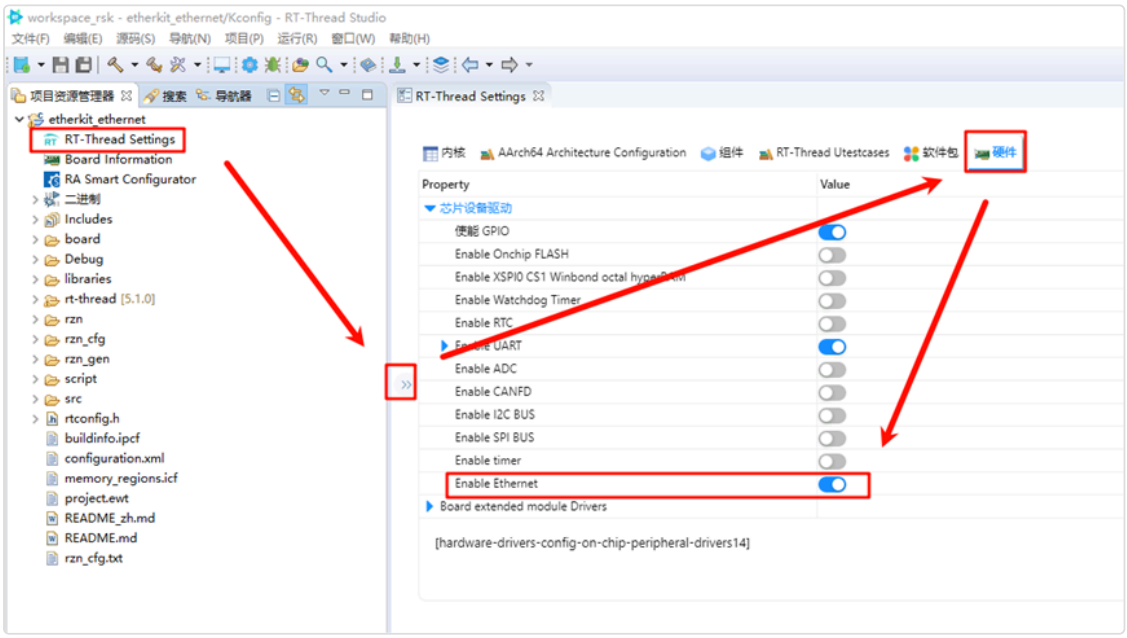


Finally, configure `ETHER_GMAC`:

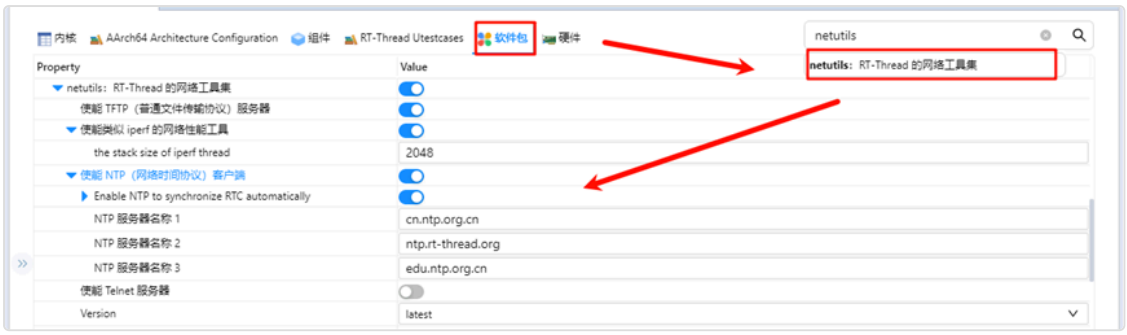


RT-Thread Studio Configuration

Return to the Studio project, and configure RT-Thread Settings. Click on "Hardware", find the chip device driver, and enable Ethernet:

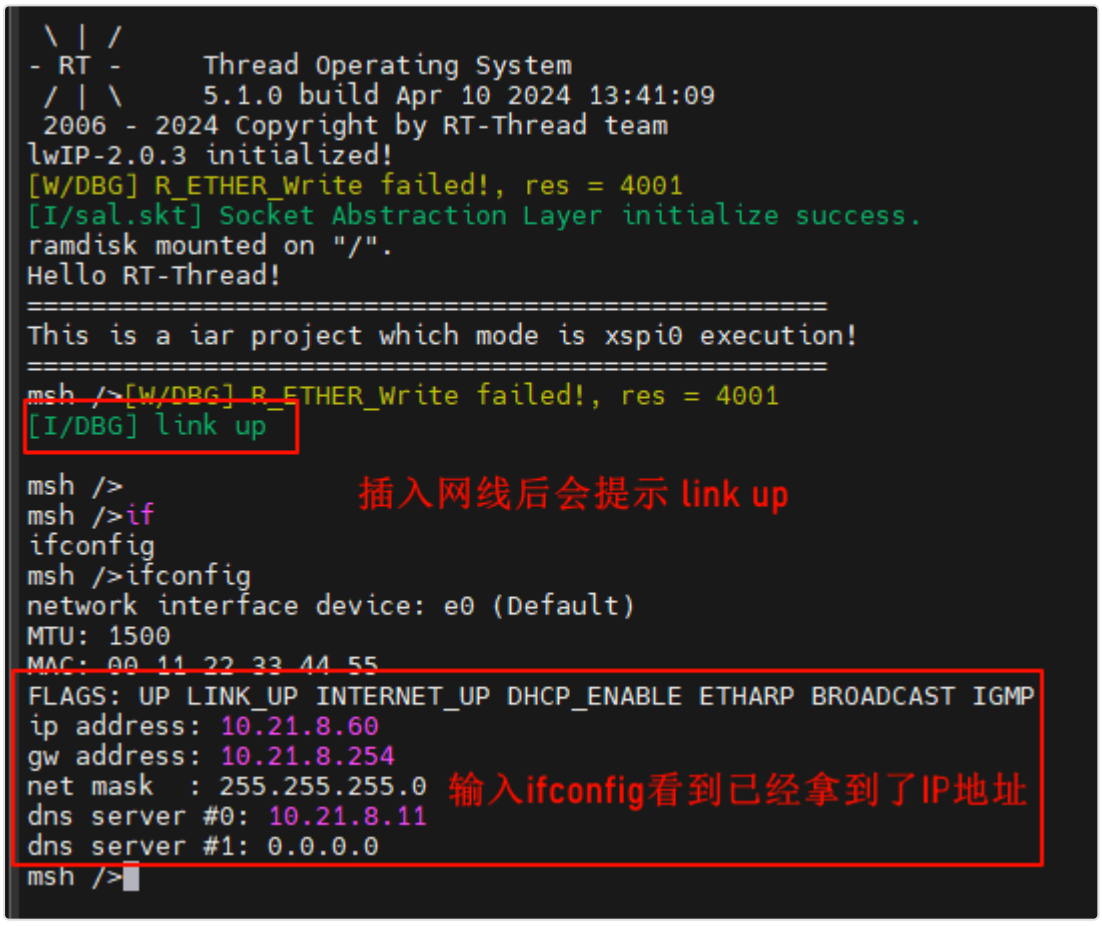


In the RT-Thread Settings, search for the `netutils` software package and enable the `tftp`, `iperf`, and `ntp` features:



Ethernet IP Experiment Results

After flashing the code to the development board, open the serial terminal to view the logs:



TFTP Server Send Test

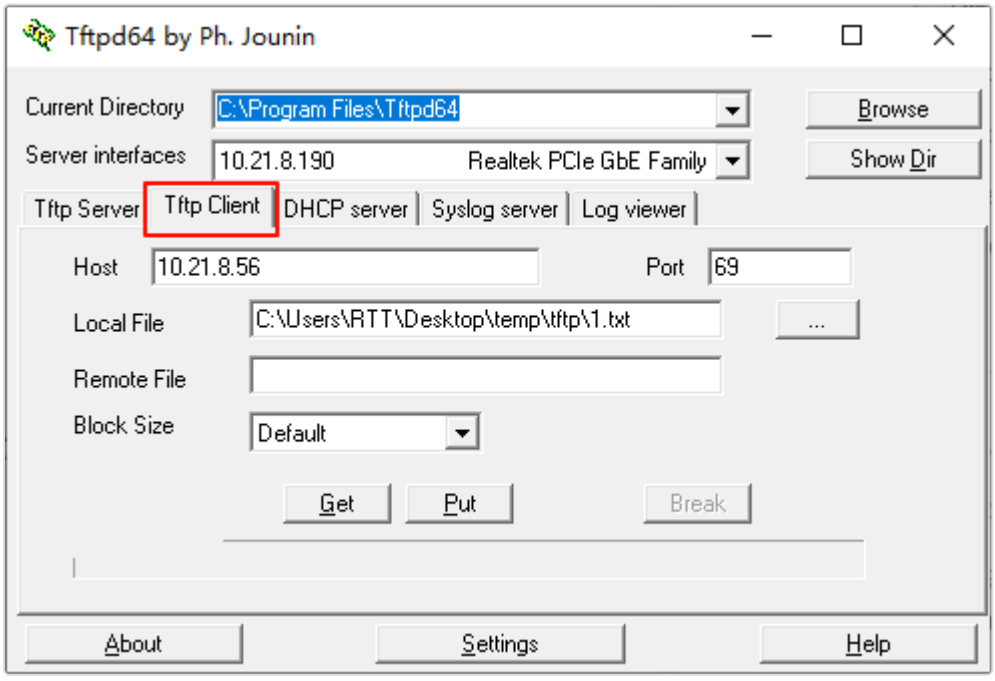
1. Install the Tftpd64-4.60-setup software from `netutils-v1.3.3\tools`:

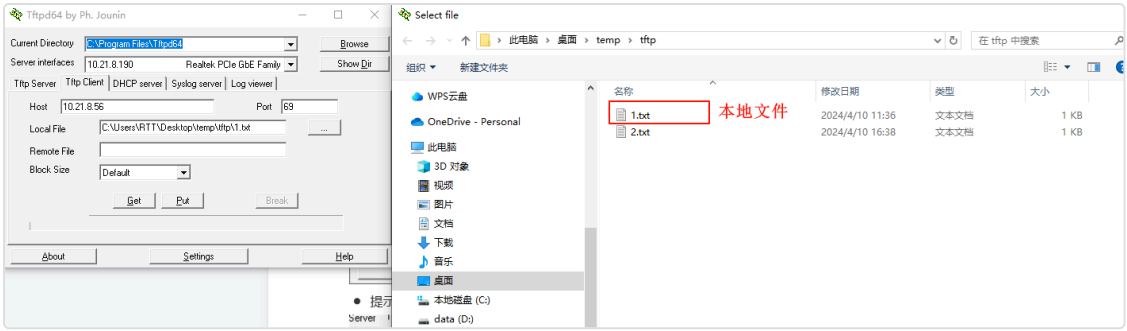
2. Go back to the development board serial terminal and input the `tftp_server` command to start the TFTP server service:

```
msh />tftp_server
TFTP server start successfully.
```

3. Open the installed Tftpd64-4.60 software:

- Set **Host** to the development board's IP address;
- Set **Port** to the TFTP server's port (default is 69);
- Set **Local File** to the file path where the client will send the file (including filename);
- Click **Put** to send the file to the device.

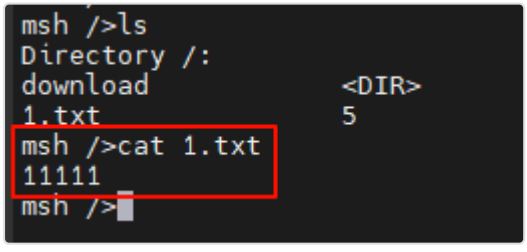




4. After clicking "Put", a message will indicate that the file has been sent:

5. Back at the development board terminal, input `ls` to see that the `1.txt` file has been received. You can input `cat 1.txt` to check if the file content matches what was sent:

- Note: Since **ramfs** is enabled, do not send files larger than 128KB. This is for testing purposes only.



TFTP Receive Test

1. Back at the development board serial terminal, input `echo "rtthread" 2.txt` to create a file with custom content:

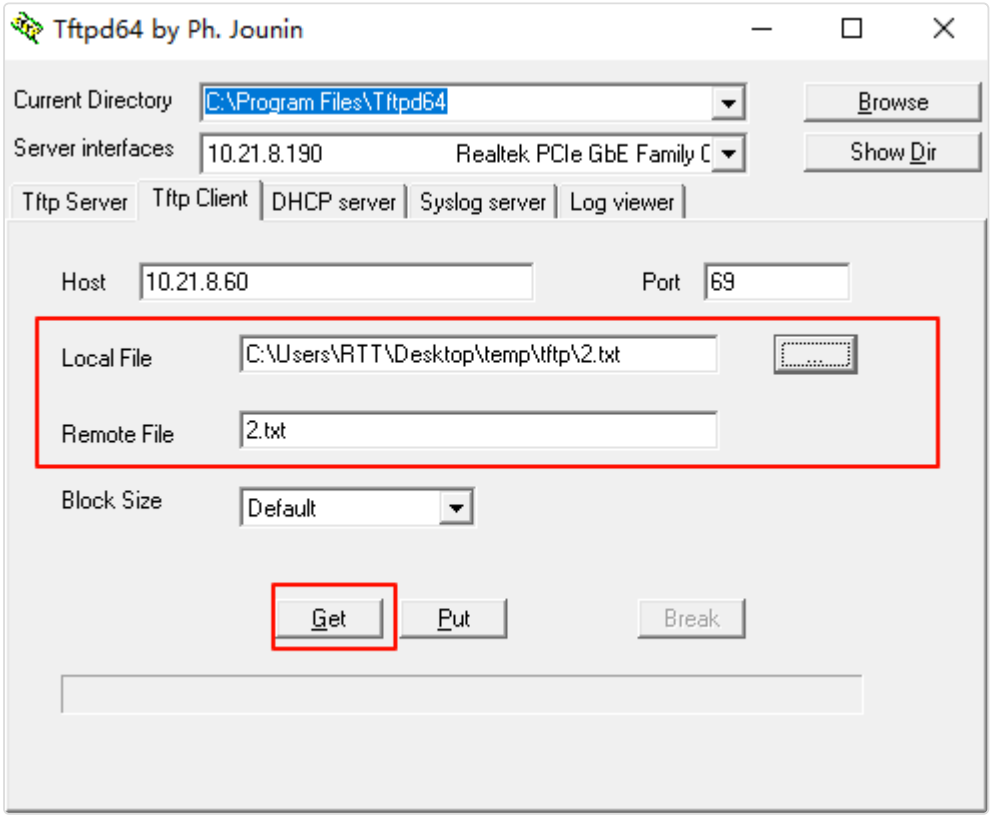
```
msh />echo "rtthread" 2.txt
msh />
msh />
```

2. You can verify the file creation and content:

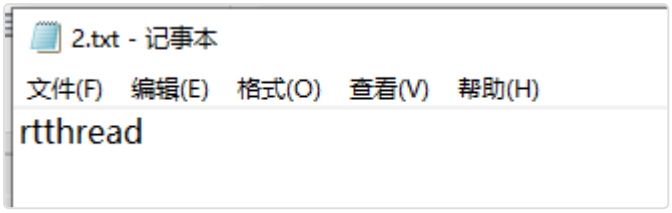
```
msh />ls
Directory /:
download            <DIR>
1.txt               5
2.txt               8
msh />cat 2.txt
rtthread
msh />
```

3. Open the installed Tftpd64-4.60 software:

- Set **Local File** to the path where the client will save the received file (including filename);
- Set **Remote File** to the path of the file on the server (including filename), i.e., the file to be received;
- Set the TFTP server port to 69;
- Click **Get** to receive the file.



- You should see that `2.txt` has been successfully received, and its content matches the file in the development board's file system:

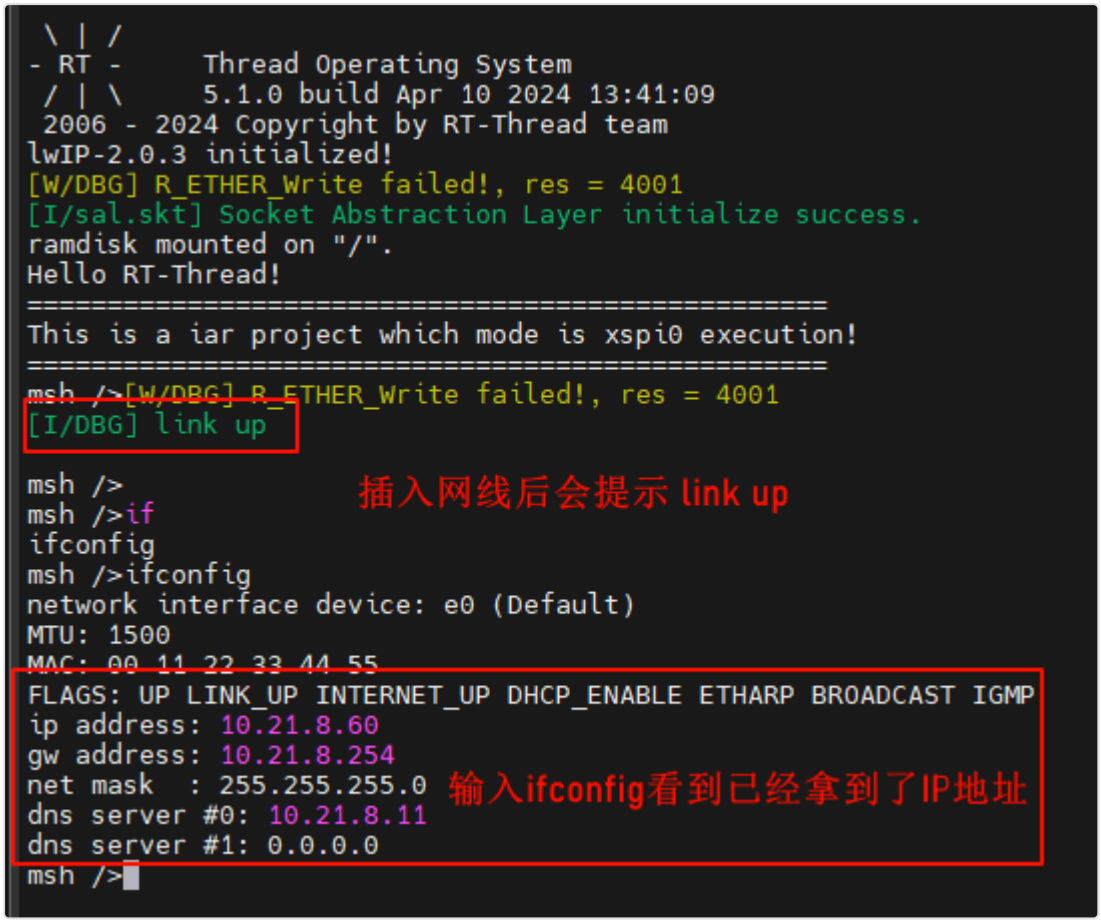


NTP Network Time Synchronization

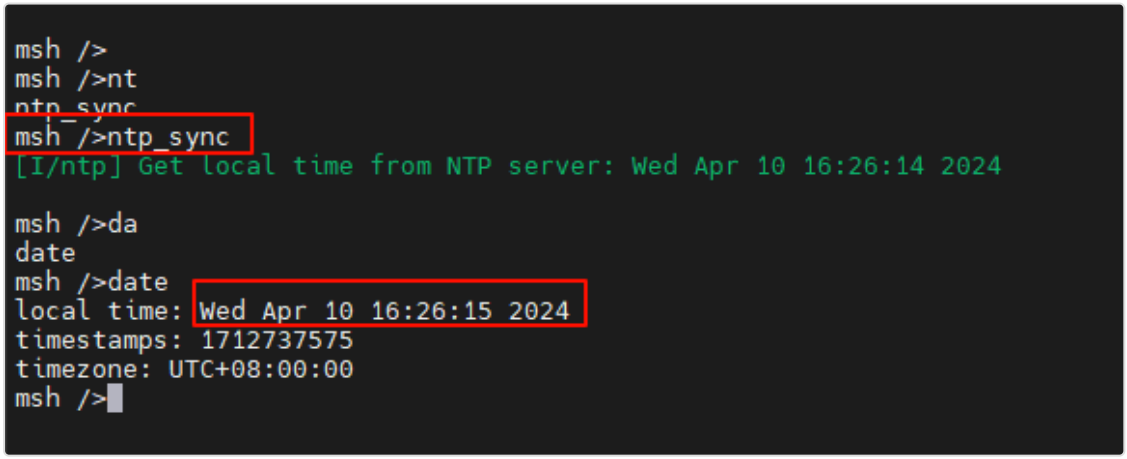
> *NTP (Network Time Protocol) is a protocol used to synchronize computer clocks with a global time standard.*

NTP Experiment Results

After flashing the code to the development board, open the serial terminal to view the logs:



Input the `ntp_sync` command, and you should see the network time synchronization result. Input the `date` command to check that the RTC time has been synchronized:



5. Industrial Protocol

5.1. EtherCAT CoE Usage Instructions

English | **中文**

Introduction

EtherCAT CoE (**CAN over EtherCAT**) is a communication protocol within the EtherCAT protocol that integrates the CANopen application layer protocol into the EtherCAT network for device control and data exchange in distributed systems. It combines the ease of use of CANopen with the high performance of EtherCAT, making it widely used in industrial automation, motion control, and sensor networks.

The following are the main features and functions of CoE:

Based on CANopen:

- The CoE application layer directly adopts the device protocol of CANopen, including the structure and services of the Object Dictionary.
- Device parameters, communication objects, and control data are defined via the Object Dictionary, ensuring interoperability between devices.

Supports Standard Services:

- **SDO (Service Data Object)**: Used for point-to-point configuration and diagnostic communication, allowing the master to exchange large amounts of data (e.g., parameter configuration) with the slave.
- **PDO (Process Data Object)**: Used for real-time communication, transmitting small amounts of periodic process data with quick response times.
- **Emergency (EMCY) Messages**: Used to report abnormal device conditions.
- **NMT (Network Management)**: Provides network management functions such as starting, stopping, and resetting devices.

Efficient Data Transfer:

- The EtherCAT bus structure and high-speed frame processing capabilities allow CoE to exchange data with lower latency and higher efficiency.

Supports Various Application Scenarios:

- Suitable for industrial equipment configuration, real-time monitoring, parameter diagnostics, and system integration.

Object Dictionary Mapping:

- The Object Dictionary organizes device data and functions in a hierarchical structure.
- EtherCAT uses the CoE protocol to access variables in the Object Dictionary for parameter reading, writing, and real-time control.

Typical Applications:

- Used in drives (e.g., servo drives) supporting complex control logic.
- Used in engineering tools for monitoring, debugging, and configuring devices.

This section demonstrates how to implement EtherCAT CoE master-slave communication using Beckhoff TwinCAT3 and the EtherKit development board. The example project supports both CSP and CSV operating modes.

Preliminary Setup

Software Environment:

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

Hardware Environment:

- EtherKit Development Board
- One Ethernet Cable
- Jlink Debugger

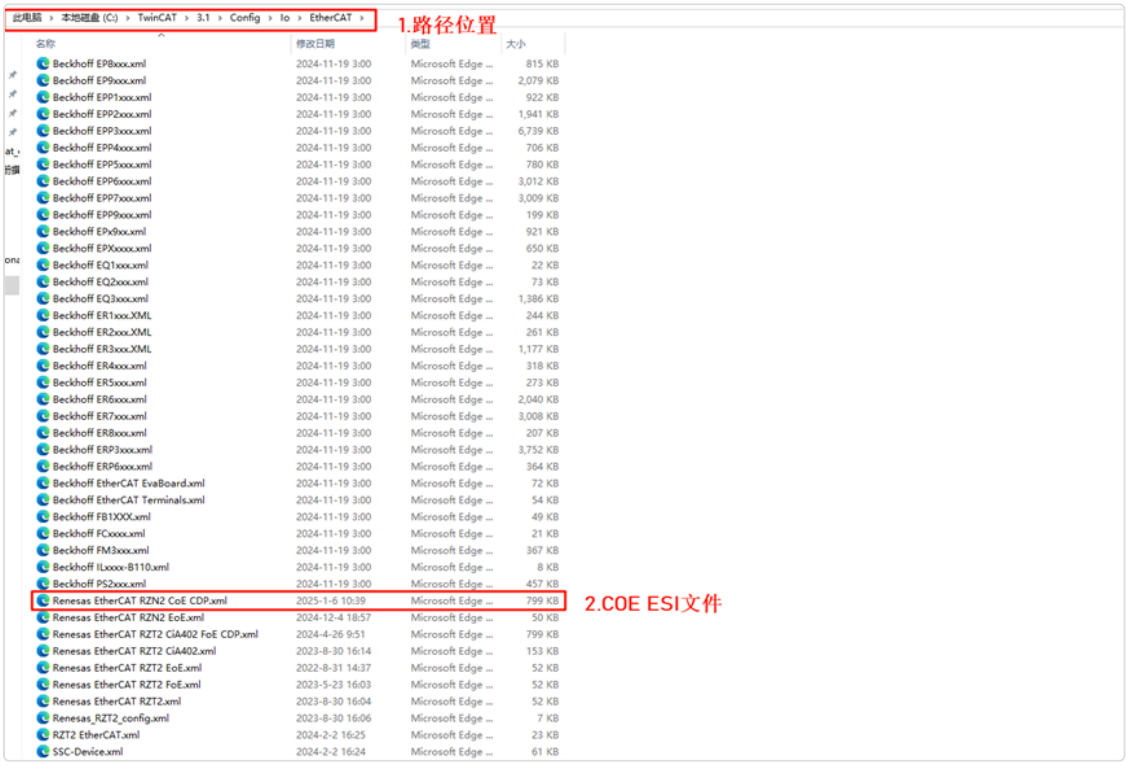
TwinCAT3 Configuration

> *Before starting TwinCAT3, we need to perform some configuration tasks:*

Install ESI Files

Before starting TwinCAT, copy the ESI files contained in the release folder to the TwinCAT target directory: **..\TwinCAT\3.x\Config\IO\EtherCAT**

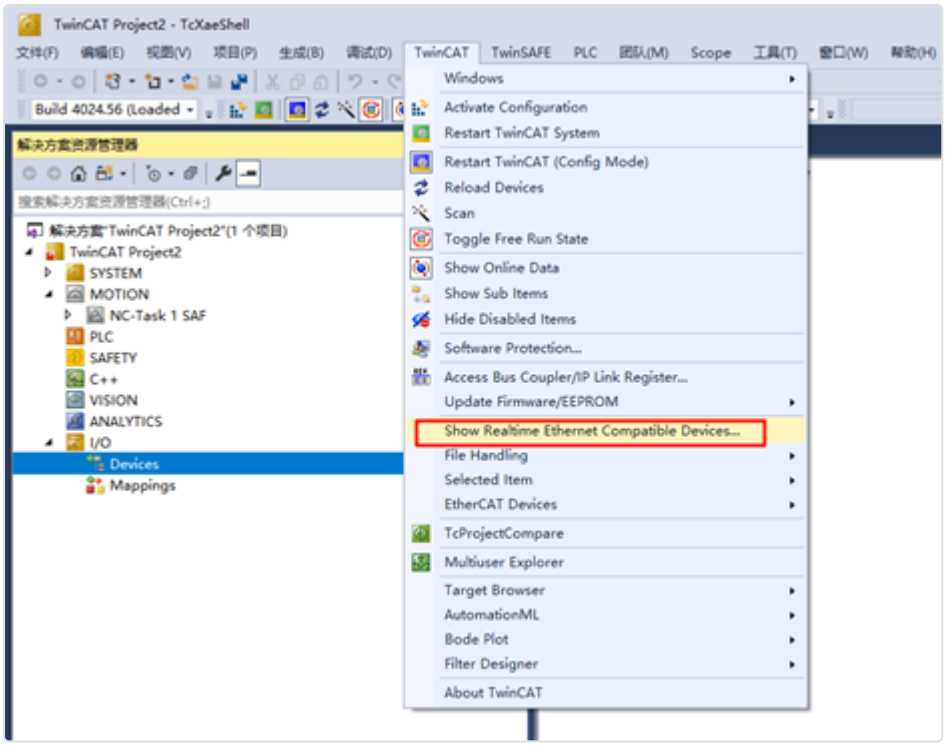
> *Note: The current version of the ESI file is located at: ..\board\ports\ESI_File\Renesas EtherCAT RZN2 CoE CDP.xml*



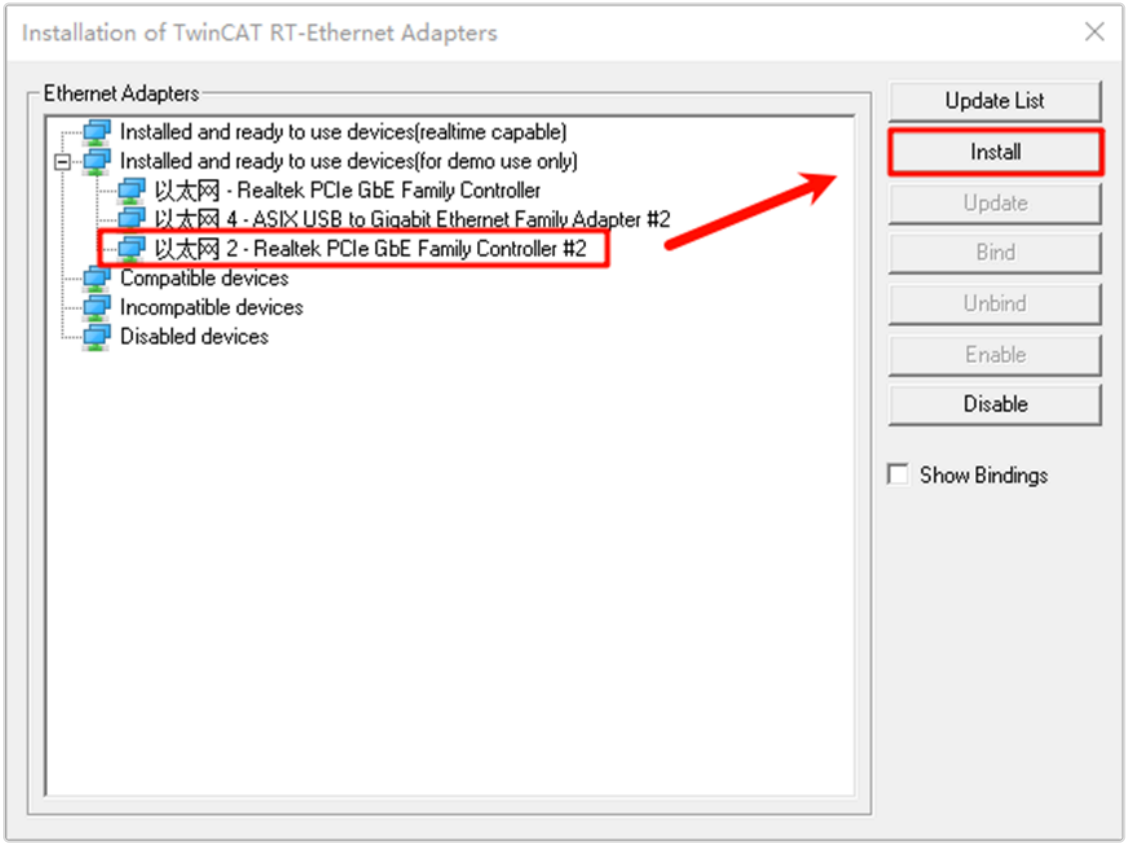
Add TwinCAT Ethernet Driver

Add the TwinCAT Ethernet driver (this step is only needed for the initial setup); from the Start menu, select [TwinCAT] → [Show Realtime Ethernet Compatible

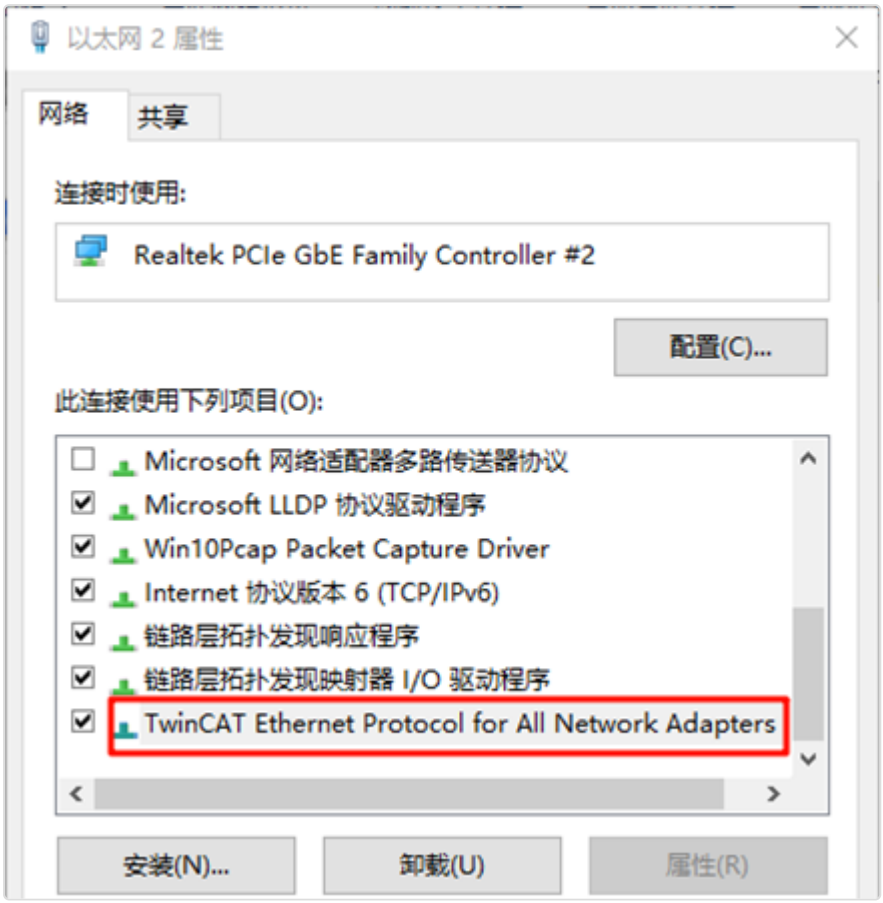
Devices…], choose the connected Ethernet port from the communication port list and install it.



Here we can see all the Ethernet adapters on the PC. Select the port you will be testing with, then click Install:

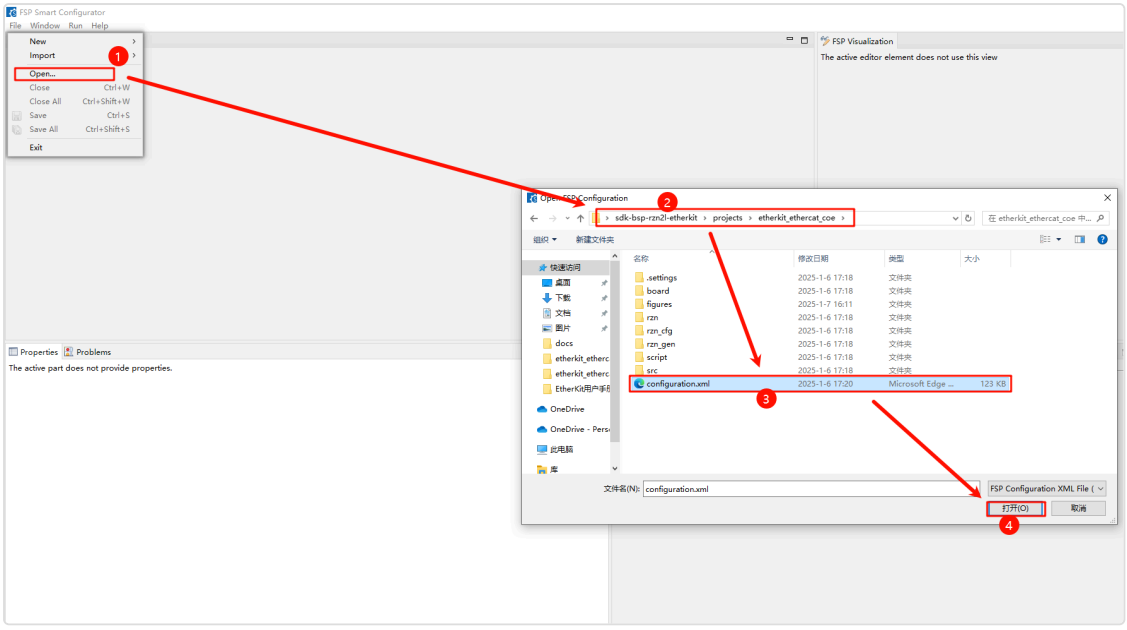


After checking the network adapter, we can see that the installation was successful:

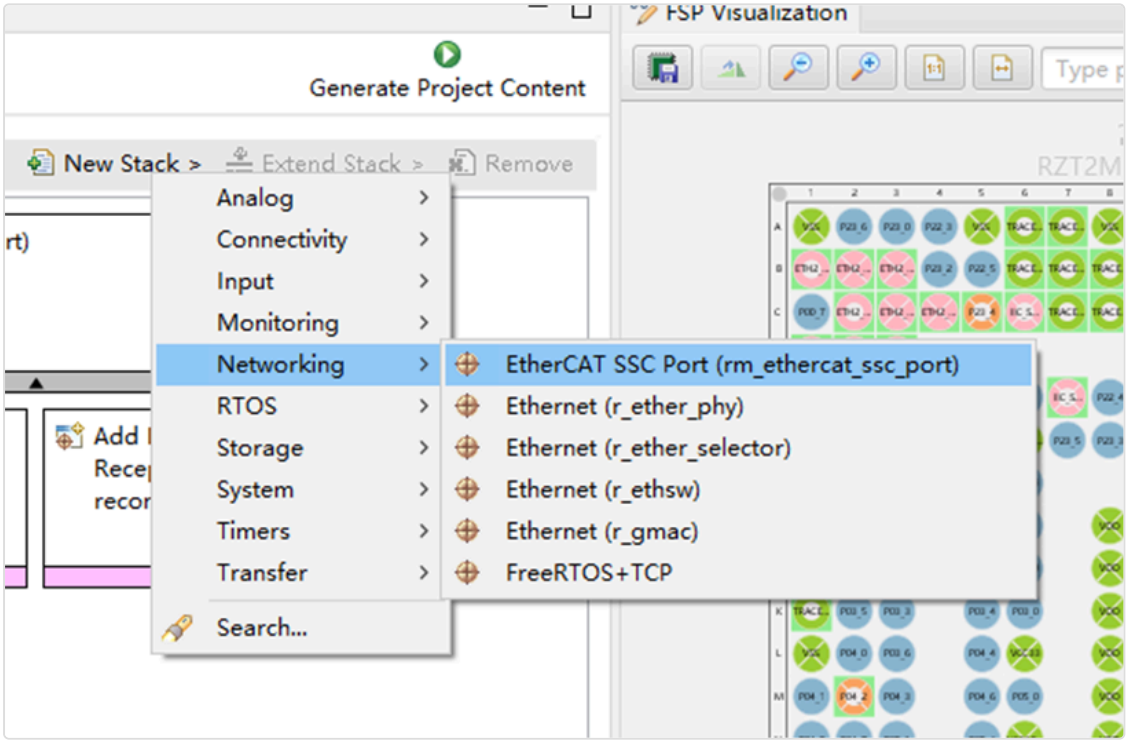


FSP Configuration Instructions

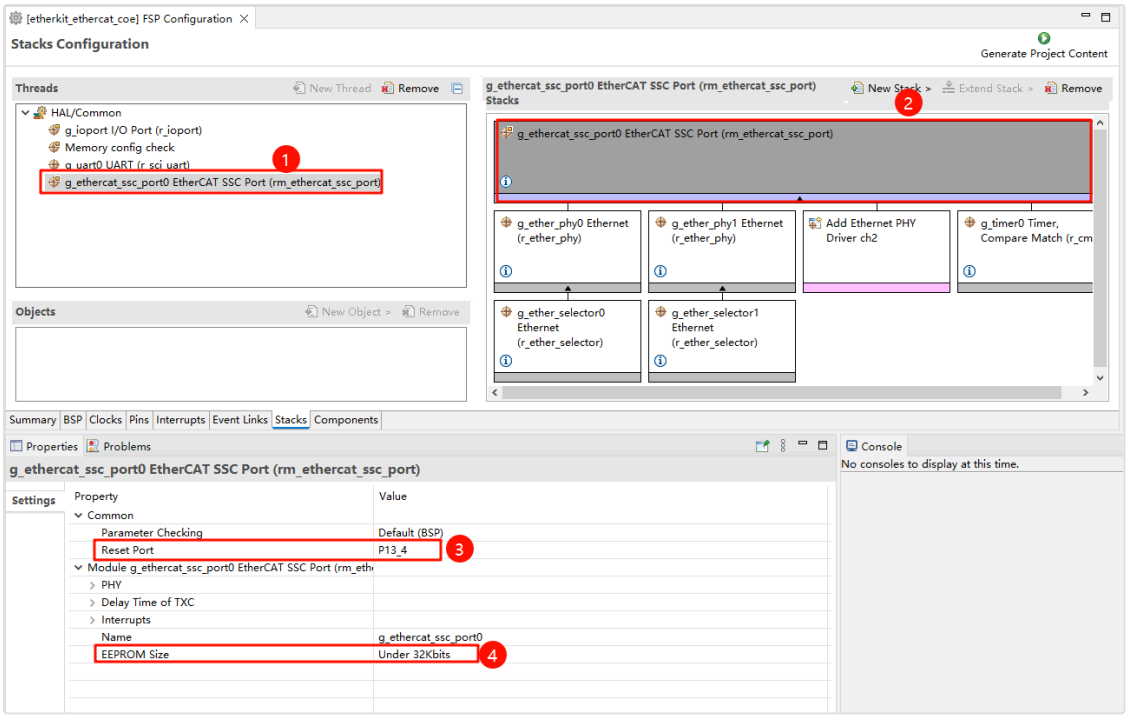
Next, we proceed with the pin initialization configuration. Open the installed RZN-FSP 2.0.0 and select the root directory of our project:



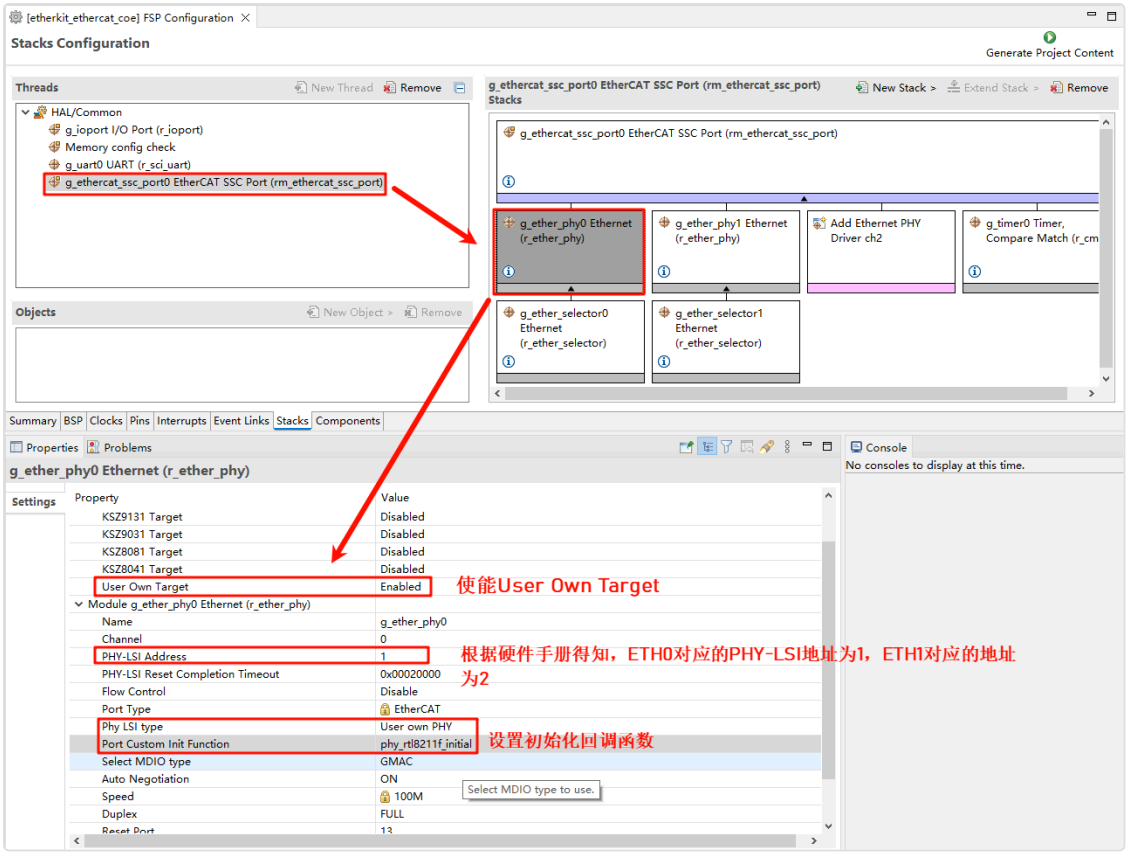
We configure the peripherals and pins as follows: click New Stack and add the ethercat_ssc_port peripheral:



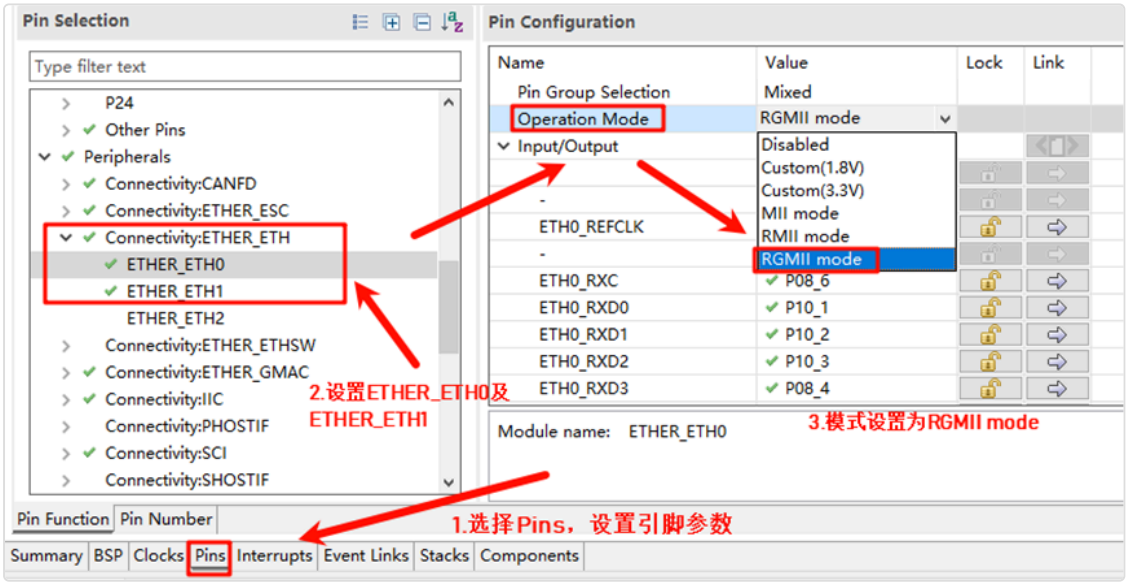
Configure ethercat_ssc_port: modify the Reset Port to P13_4, and set the EEPROM_Size to Under 32Kbits:



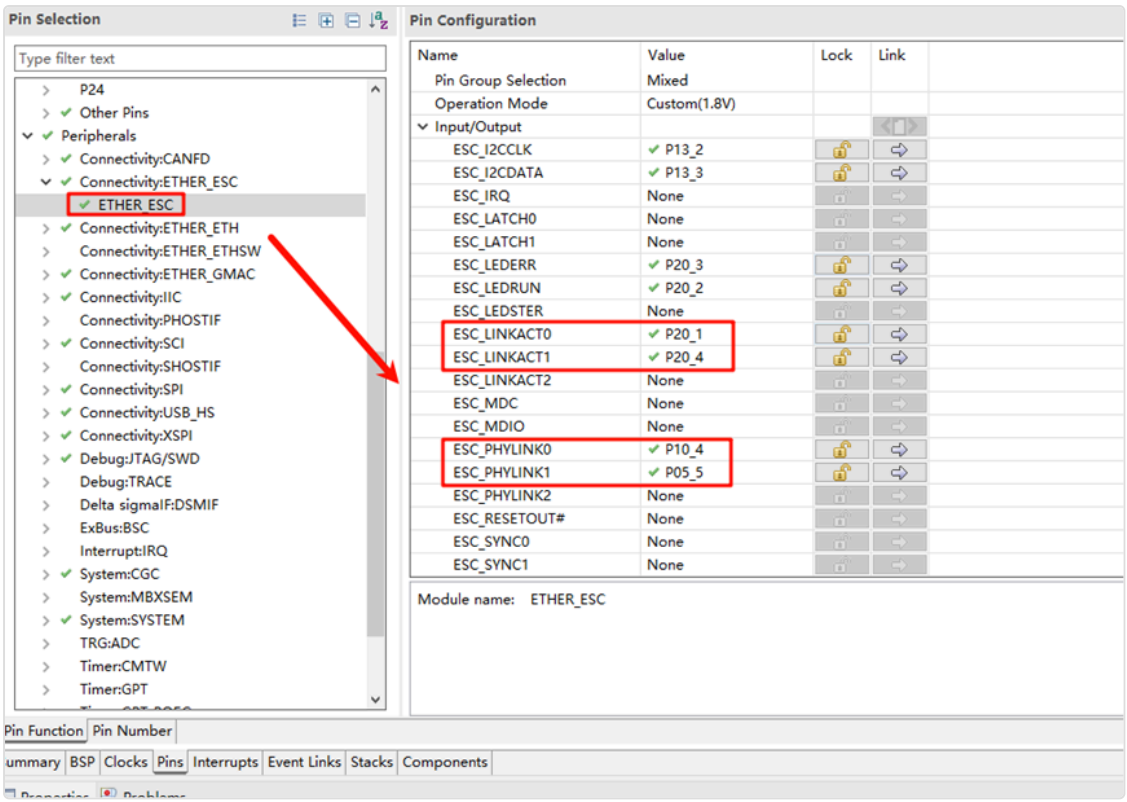
Enable the Ethernet card type and configure the Ethernet device parameters. Here, we add two PHYs (phy0 and phy1). It should be noted that EtherKit uses the RTL8211 Ethernet card, which is not supported by the Renesas FSP. However, Renesas has reserved user-defined network interfaces, so we configure the network card as follows and set the MDIO type to GMAC, with the network card initialization callback function `phy_rtl8211f_initial();`:



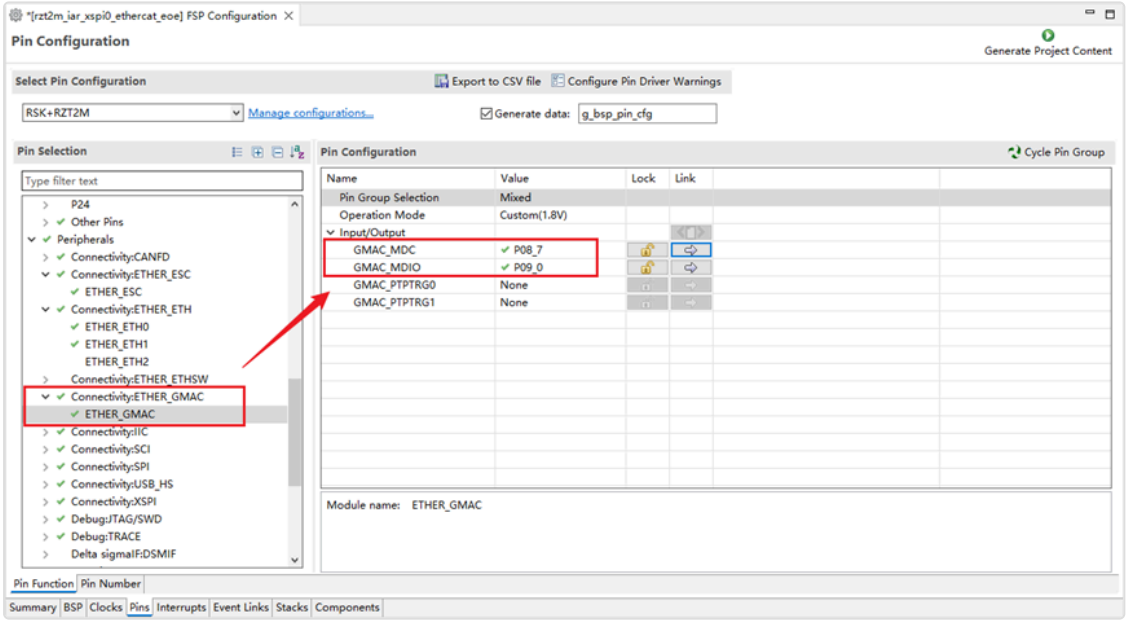
Configure the network card pin parameters and set the operating mode to RGMII:



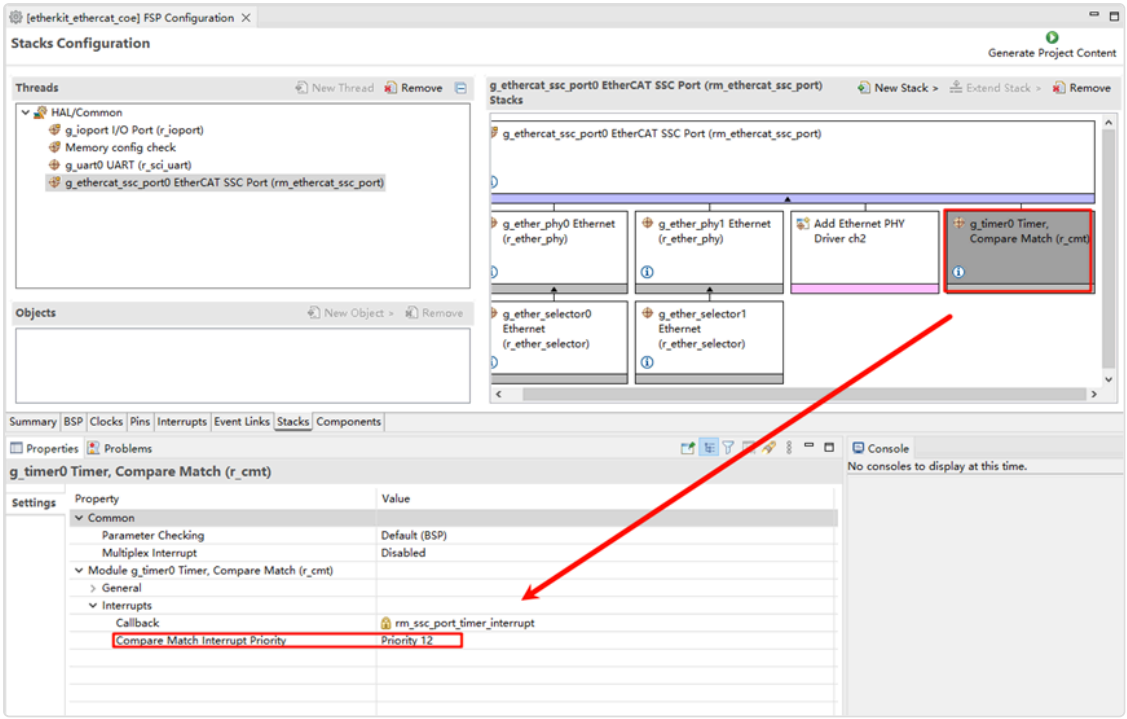
ETHER_ESC Settings:



ETHER_GMAC Configuration:



Add a CMT timer to ethercat_ssc_port and configure the interrupt priority:



Finally, click Generate Project Content to generate the low-level driver source code.

Build Configuration

1. Modify the `sconscript`: Enter the project and find the file at the specified path: `.\rzn\SConscript`, and replace it with the following content:

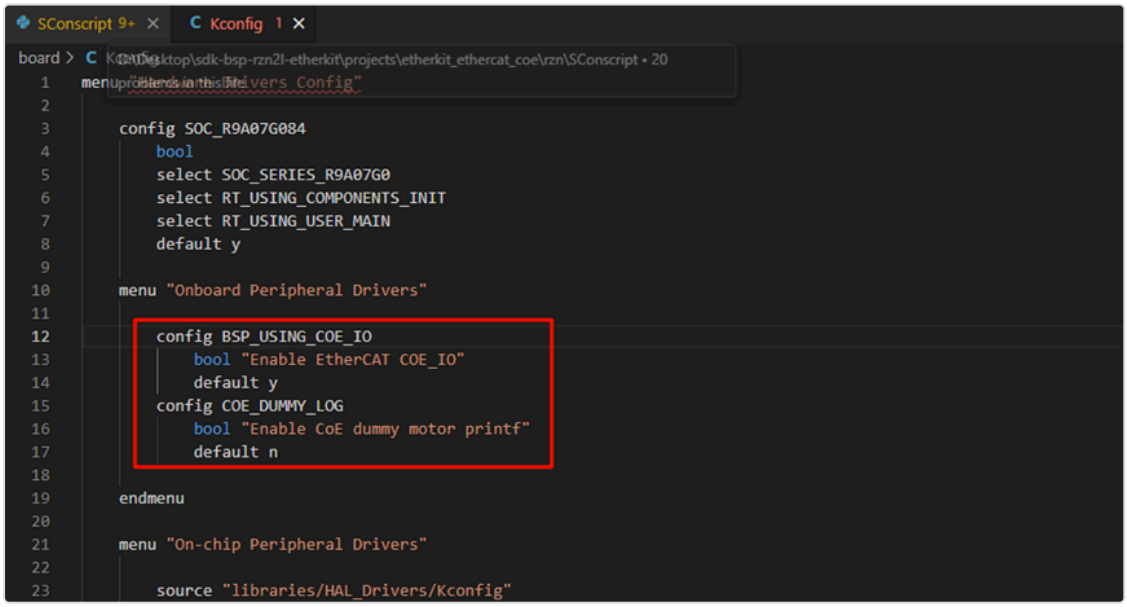
```
Import('RTT_ROOT')
Import('rtconfig')
from building import *
from gcc import *

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

if rtconfig.PLATFORM in ['iccarm'] + GetGCCLikePLATFORM():
    if rtconfig.PLATFORM == 'iccarm' or GetOption('target') !=
        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/
        src += Glob('./fsp/src/bsp/cmsis/Device/RENESAS/Source/
```

```
        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 = CPPPAT
Return('group')
```

2. Modify Kconfig: Open the file under the project directory (projects\etherkit_ethercat_coe\board\Kconfig), and add the CoE configuration under the Onboard Peripheral Drivers option:

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

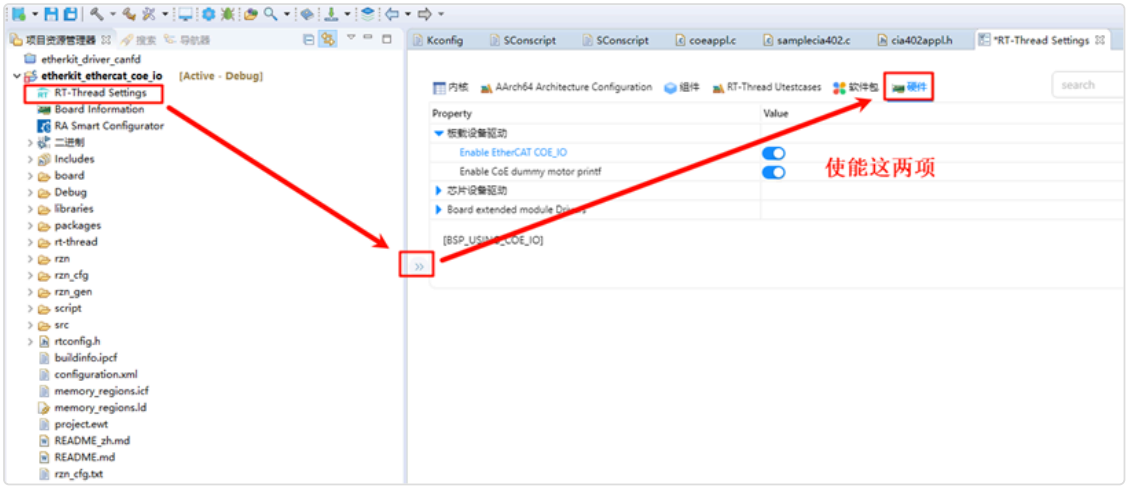
As shown below:



3. If using Studio for development, right-click the project and select **Sync Scons Configuration to Project**; if using IAR, right-click the project folder and execute: `scons –target=iar` to regenerate the configuration.

RT-Thread Studio Configuration

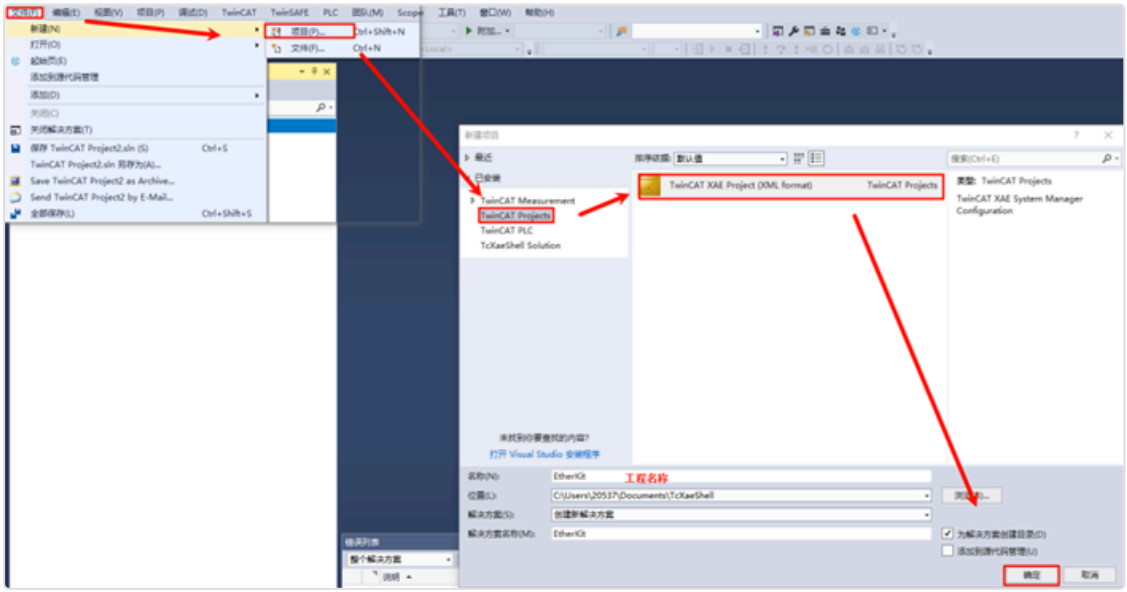
After completing the FSP configuration, the pin and peripheral initialization is mostly complete. Next, we need to enable the EtherCAT EOE example. Open Studio, click RT-Thread Settings, and enable the EOE example:



EtherCAT CoE Configuration

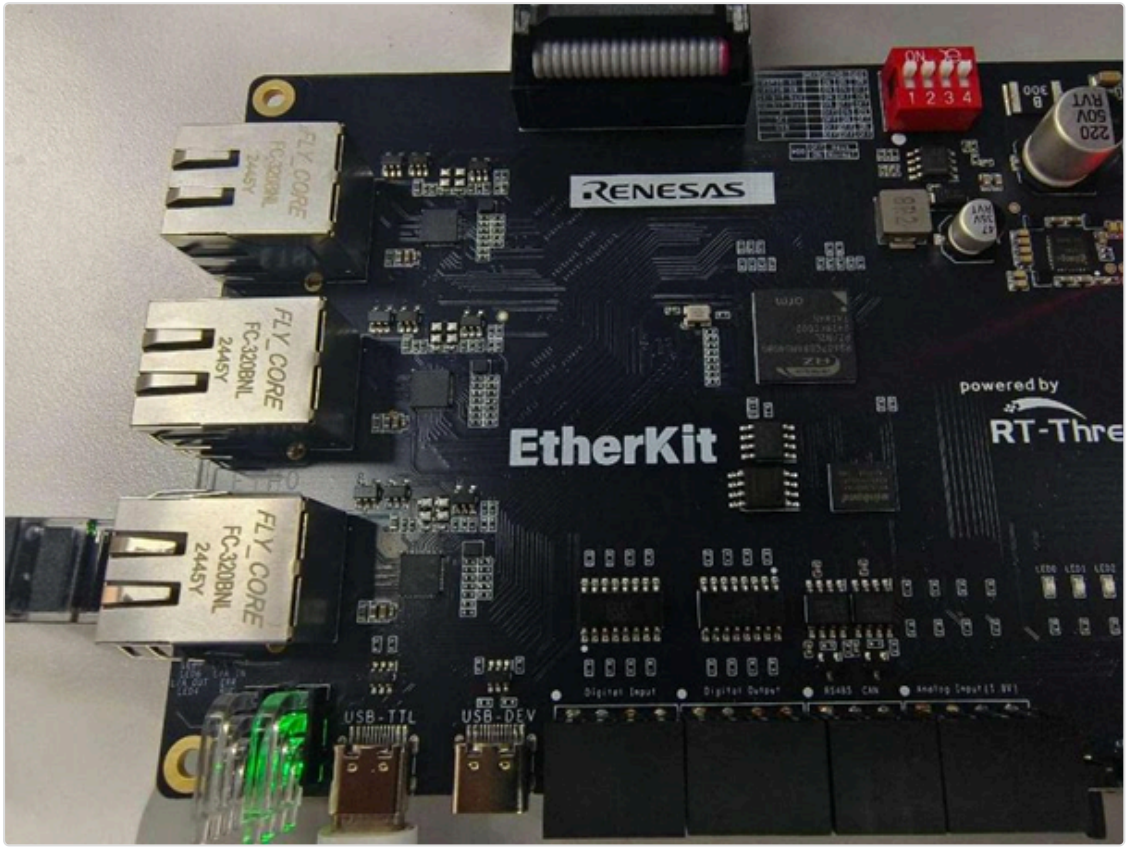
Create a New TwinCAT Project

Open TwinCAT software, click File -> New -> New Project, select TwinCAT Projects, and create a TwinCAT XAR Project (XML format):



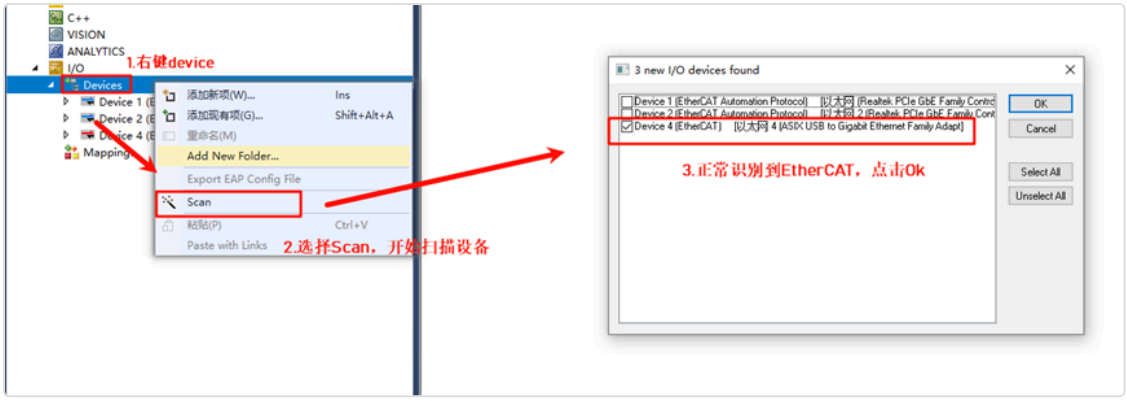
Slave Startup CoE App

After powering up the EtherKit development board, use an Ethernet cable to connect to the ETH0 port. EtherCAT will run by default.



Slave Device Scanning

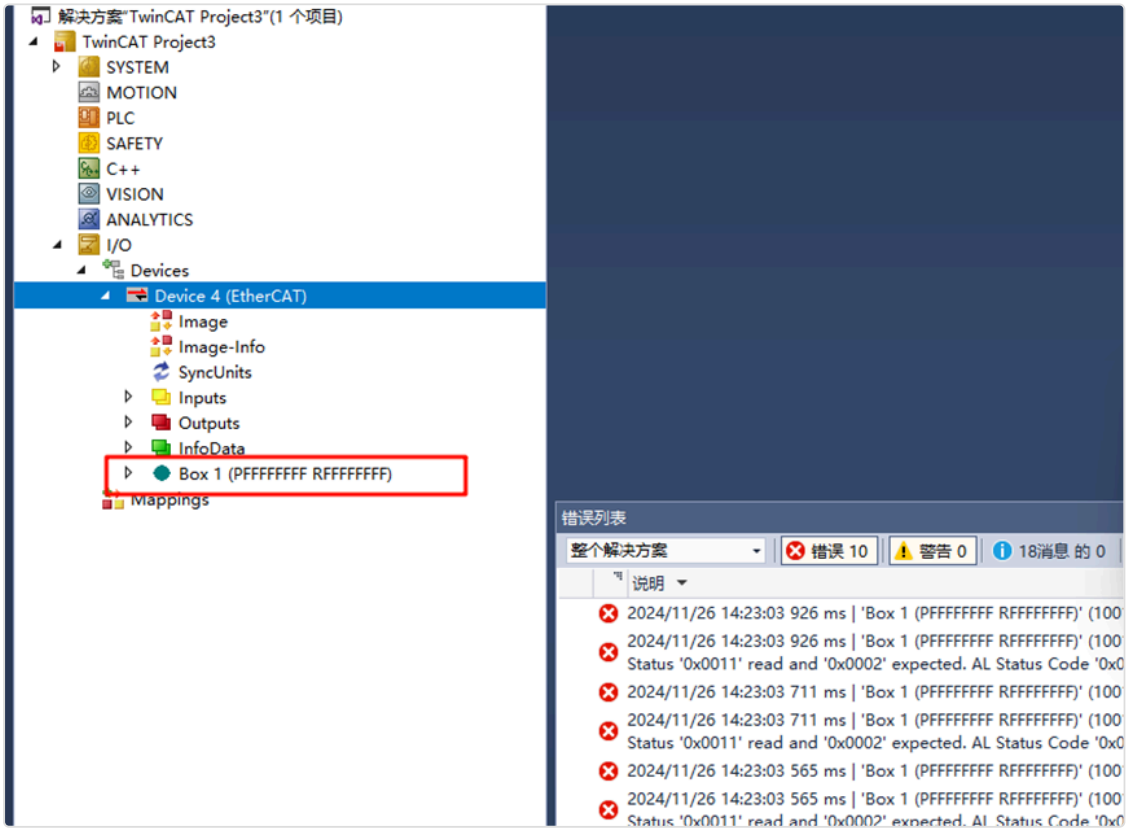
After creating a new project, find **Devices** in the left navigation bar, right-click, and select **Scan for devices**. Normally, if scanning the slave device is successful, it will show: **Device x[EtherCAT]**. If the scan fails, it will show: **Device x[EtherCAT Automation Protocol]**, indicating that the slave initialization has failed.



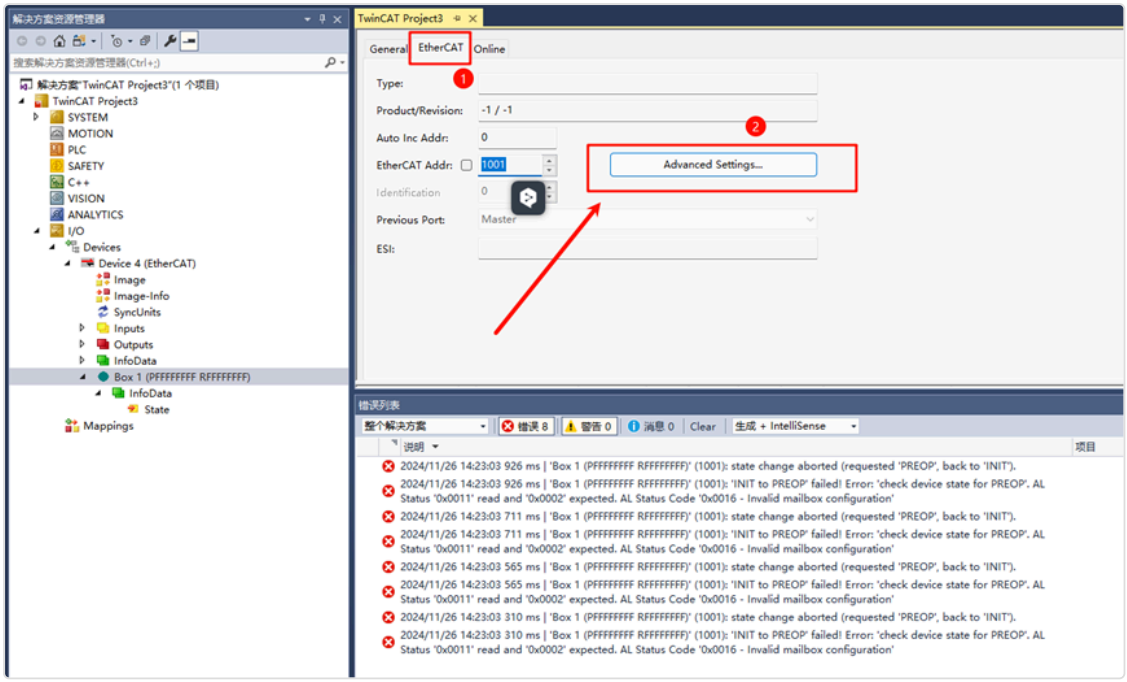
Click **Ok** and a window will pop up: **Scan for boxes**. After confirming, another window will pop up: **Activate Free Run**. Since we are using CoE for the first time and need to update the EEPROM firmware, we will not activate it for now.

Update EEPROM Firmware

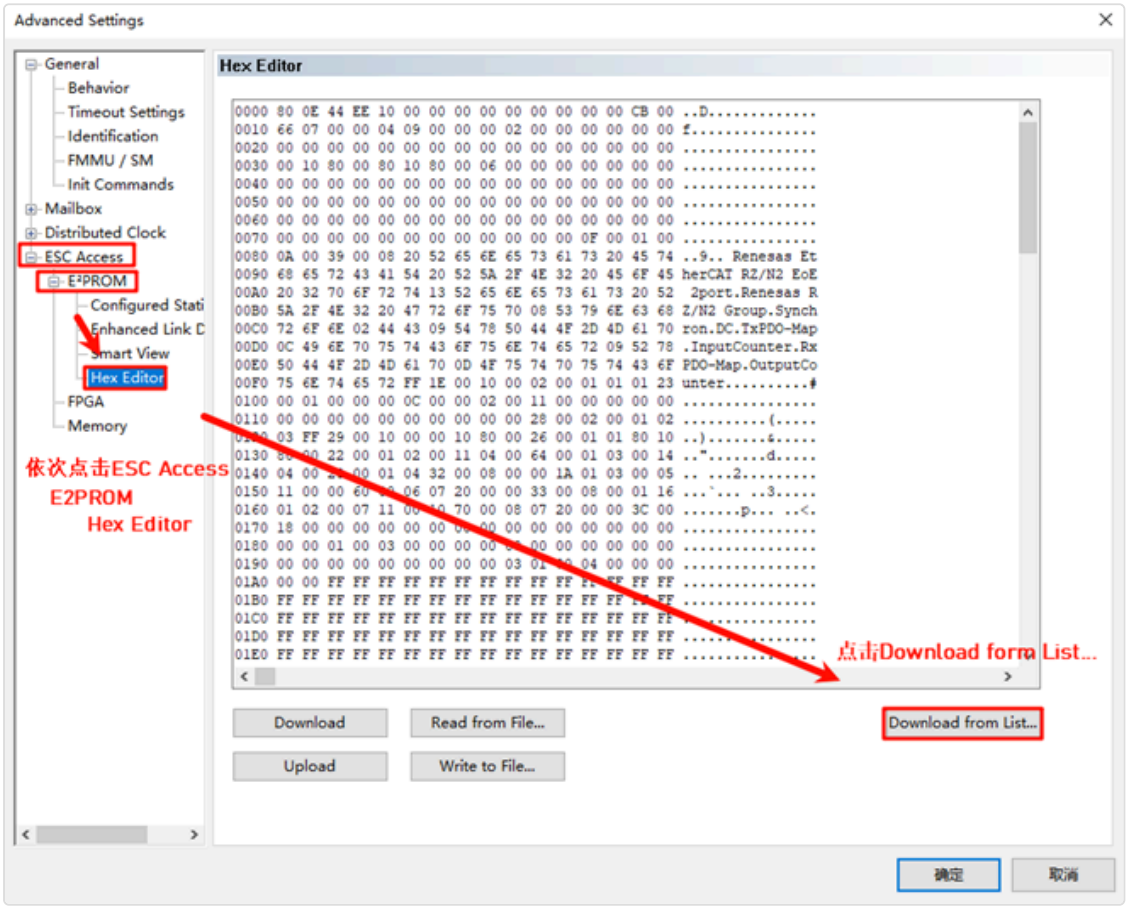
Return to TwinCAT. In the left navigation bar, since we have successfully scanned the slave device, the master-slave configuration interface is now visible:



Double-click **Box 1** and in the top navigation bar of the middle interface, click **EtherCAT** and then click **Advanced Settings…**:

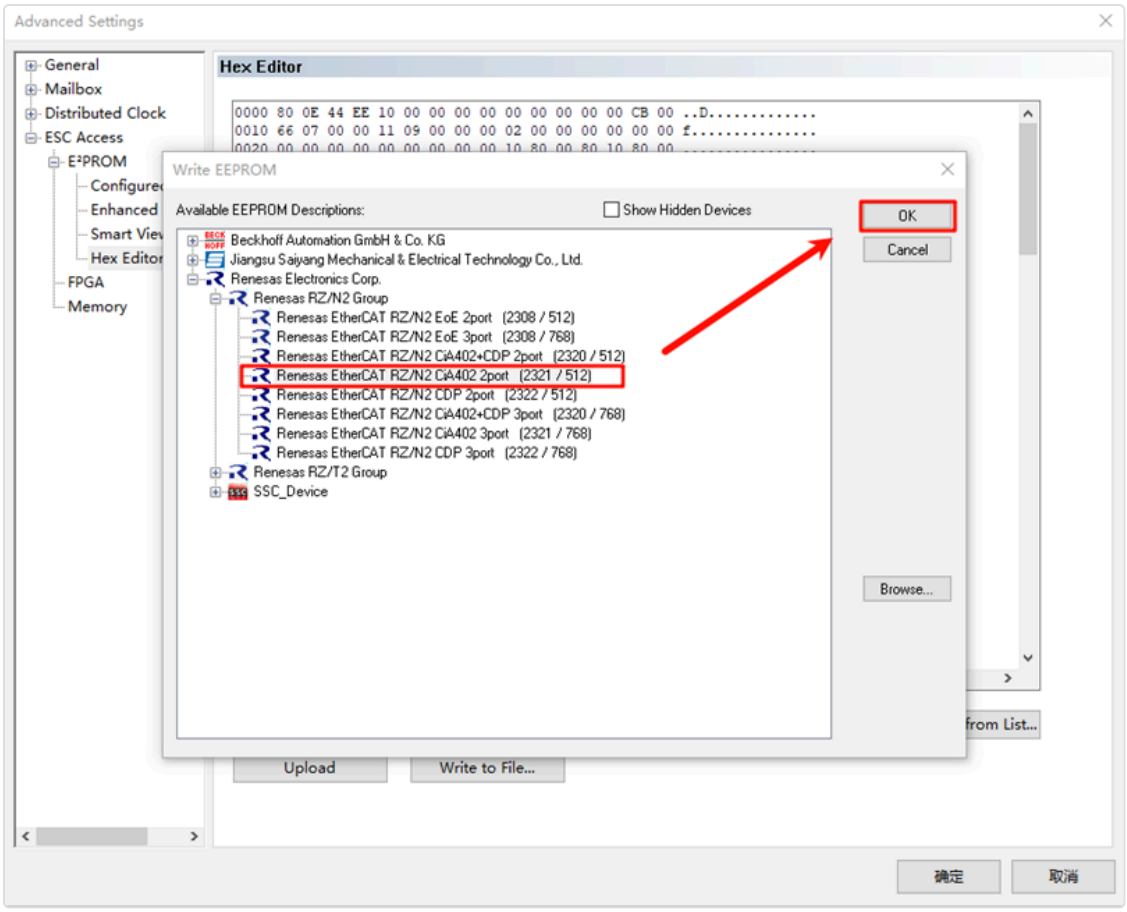


Click **Download from List…** as shown in the image below:

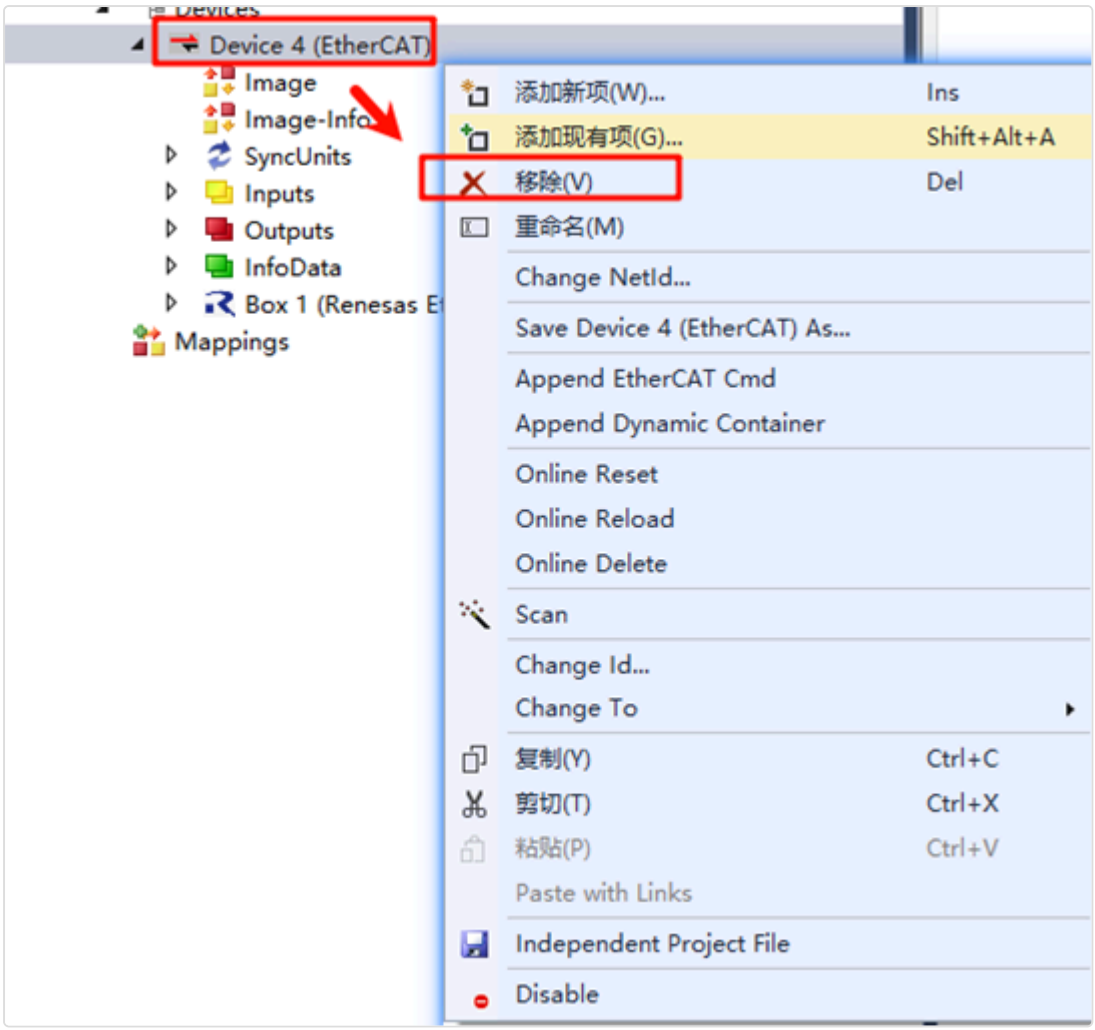


We will write the ESI firmware to the EEPROM. Since we are using a dual Ethernet port configuration, select **Renesas EtherCAT RZ/N2 COE 2port**. If

you are using a three-port configuration, select the ESI file with the **3port** suffix for download.



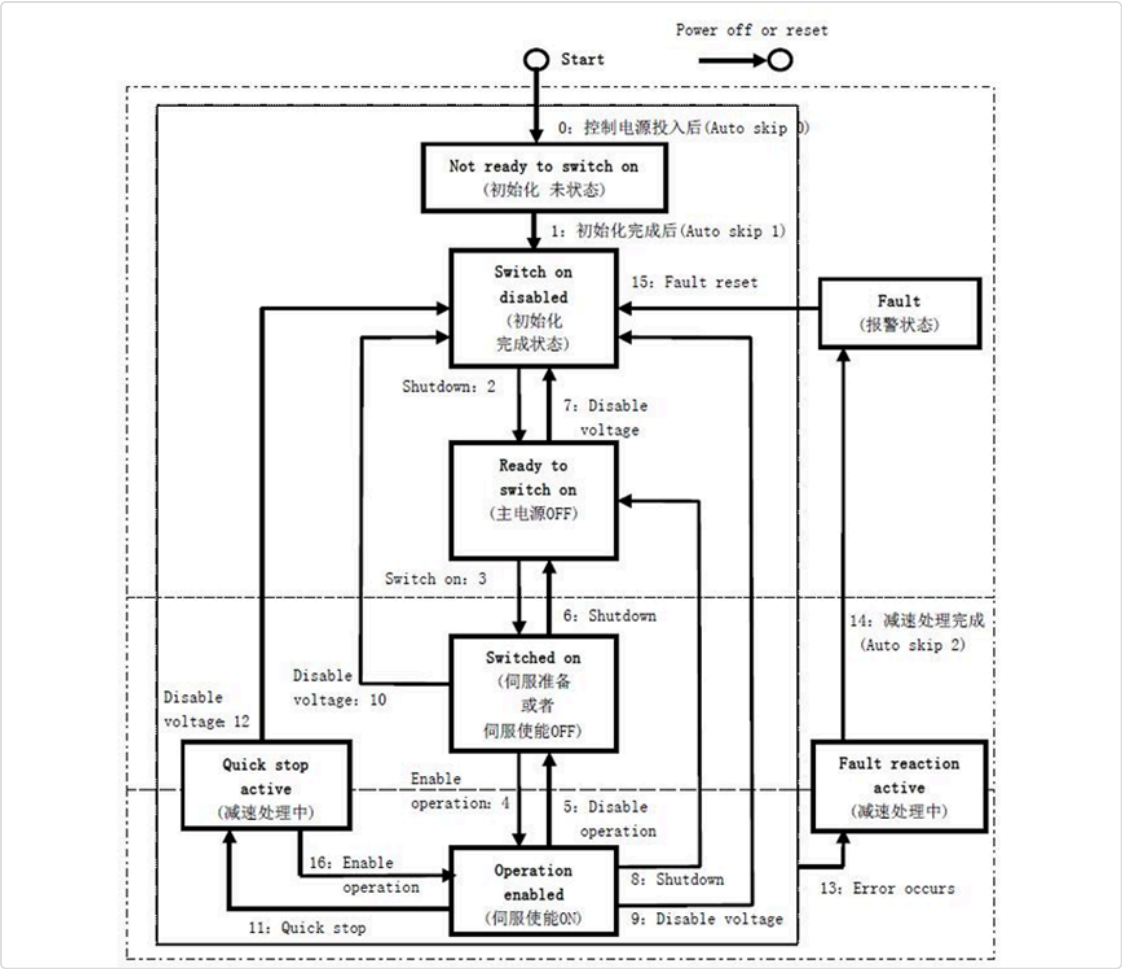
After the download is complete, right-click **Device x(EtherCAT)**, remove the device, rescan, and add the device back. Finally, complete the activation process (as mentioned above).



CiA402 Servo Usage Instructions

First, let's look at the CiA402 protocol: The CiA402 protocol (Communication Interface for Drive Systems) is defined by the CiA (CAN in Automation) organization and is a standardized protocol used in industrial automation, especially for motor control systems. CiA402 is the drive and motion controller sub-protocol for CANopen, defining the interface for inverters, servo controllers, and stepper motors. It is part of the international standard IEC 61800-7 series. The CiA402 protocol is based on the CANopen communication protocol and extends and optimizes the functionality for motion control systems. It is mainly used for controlling servo motors, stepper motors, and other types of electric drive systems.

Next, we look at the FSA (Finite State Automaton) showing different states of the drive and how to perform transitions between them.



Below is a detailed description of each state from the above diagram:

State	Description
Initialization	Servo initialization: Parameters cannot be set, and drive commands cannot be executed.
Initialization Complete	Servo initialization complete, parameters can be set.
Servo Ready	The main power is on, and servo parameters can be set, but the drive is in a deactivated state.
Waiting for Servo Enable	Main power is OK, parameters can be set, waiting for servo enable.
Servo Enabled	Servo is enabled and running according to the set mode.

State	Description
Quick Stop Active	Quick stop function is active, the drive is performing a quick stop function.
Fault Stop	Drive has encountered a fault, performing fault stop procedure.
Alarm State	Fault stop is complete, all drive functions are disabled, but the parameters can be changed to troubleshoot the fault.

For the controller, in each communication cycle, the master station sends a **Control Word** to the slave station and receives the **Status Word** from the slave to confirm its status. For example, in this project, the state transitions of CiA402 are implemented through **CiA402_StateMachine()**:

```
/*----------------------------------------------
-    CiA402 State machine
----------------------------------------------*/
#define STATE_NOT_READY_TO_SWITCH_ON        0x0001 /**< \brief
#define STATE_SWITCH_ON_DISABLED            0x0002 /**< \brief
#define STATE_READY_TO_SWITCH_ON            0x0004 /**< \brief
#define STATE_SWITCHED_ON                   0x0008 /**< \brief
#define STATE_OPERATION_ENABLED             0x0010 /**< \brief
#define STATE_QUICK_STOP_ACTIVE             0x0020 /**< \brief
#define STATE_FAULT_REACTION_ACTIVE         0x0040 /**< \brief
#define STATE_FAULT                         0x0080 /**< \brief
```

Meanwhile, the master station reads the **Status Word** (0x6041) from the slave to understand the current running state of the slave. Through the **Status Word**, the master can obtain detailed information about the current state of the slave and any potential faults or warnings:

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

The master station sends control commands to the slave using the **Control Word** (0x6040) to change its operational state or trigger specific actions:

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

CiA402 Object Dictionary Definition

Below is a list of CiA402 object dictionary items supported in the EtherKit CoE project. The position mode and speed mode are supported, and the master station can interact with the slave's process data by setting the control word, completing read and write operations based on the CoE protocol:

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

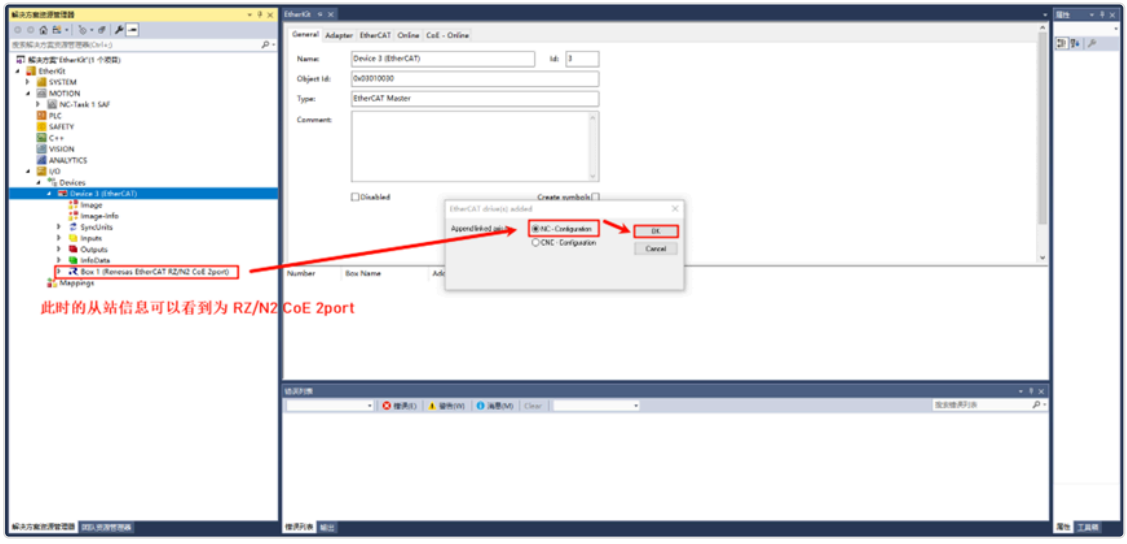
Function Group	OBJECT Name	INDEX	Category	Access	Data Type	PDO Mapping
Torque Limiting	Positive torque limit value	0x60E0	Conditional	rw	UINT16	Yes
	Negative torque limit value	0x60E1	Conditional	rw	UINT16	Yes
Homing	Home Offset	0x607C	Optional	rw	INT32	No
	Homing speeds	0x6099	Conditional	c,rw	UINT32	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

EtherCAT CoE Testing

First, ensure that the program has been successfully downloaded to the project, and the ESI file has been successfully programmed. Below is the terminal output from the development board’s serial port:

```
 \ | /
- 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
```

Next, open the previously created ESC project and scan for devices. The system will pop up **EtherCAT drive(s) added**. Select **NC – configuration**, click **OK**, and activate the device:



After activation, the EtherCAT state machine will go through **Init** → **Pre-Op** → **Safe-Op**, and finally to **Op (Operational)**. The EtherKit CoE project by default enables **csp** (Cyclic Synchronous Position Mode) and supports **csv** (Cyclic Synchronous Velocity Mode).

When the system is powered on, the drive will automatically complete the initialization and enter the **STATE_SWITCH_ON_DISABLED** state. At this point, you can set the operating mode of the drive, such as setting it to **csp** or

csv mode. Meanwhile, the CiA402 state machine information for **Axis 1** will be continuously printed on the development board:

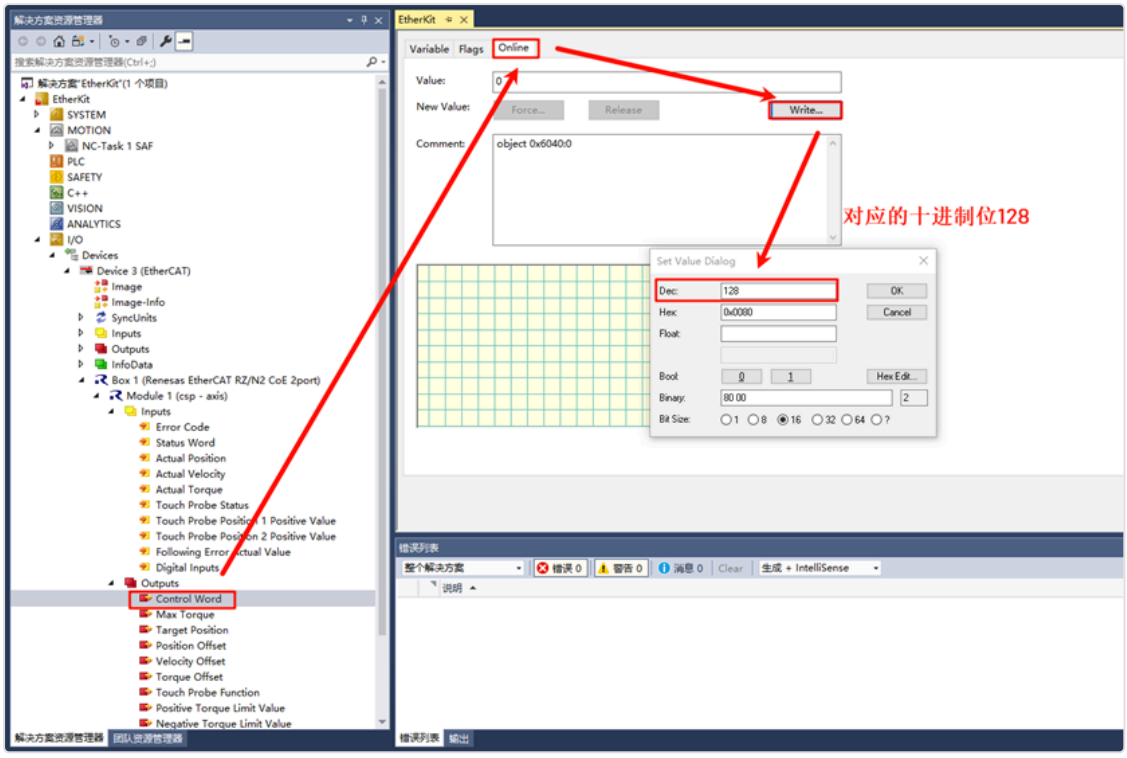
```
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7 Pass
Axis1 :StateTransition2 Pass
Axis1 :StateTransition7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

csp Position Mode Control

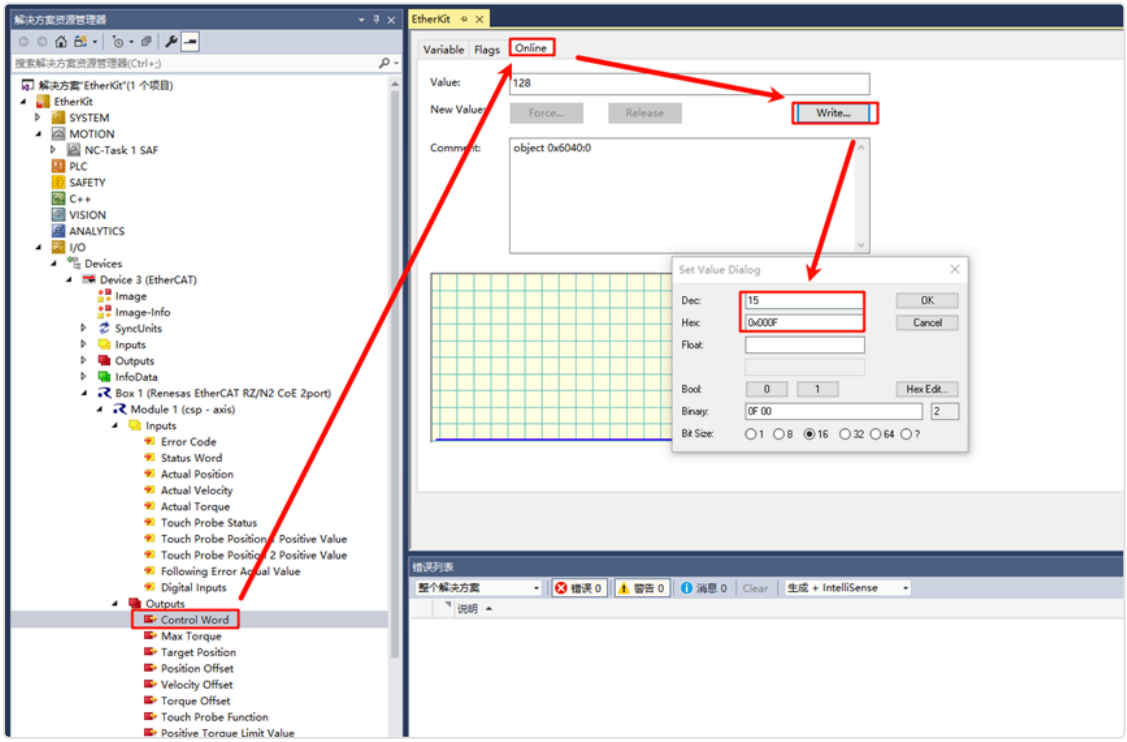
Let’s take a look at the controller in **csp** mode. In position mode, you can set the target position by writing the desired position to the control word **0x607A**, and the actual feedback position can be obtained via the status word **0x6064**.

To operate in **csp** or **csv** mode, you must first change the state to **STATE_OPERATION_ENABLED** (operational mode).

Expand the left navigation bar and click **Box 1 (Renesas EtherCAT RZ/N2 CoE 2port)** → **Module 1 (csp - axis)** → **Outputs** → **Control Word**. First, switch the state to **Servo Fault Free Mode** by writing **0x0080** (dec: 128) to control word **0x6040**. This changes the servo controller to an unobstructed state:



At this point, you will see that the serial terminal on the slave will stop printing **State Transition2** and **State Transition7**. Then, write **0x000F** (dec: 15) to control word **0x6040**:

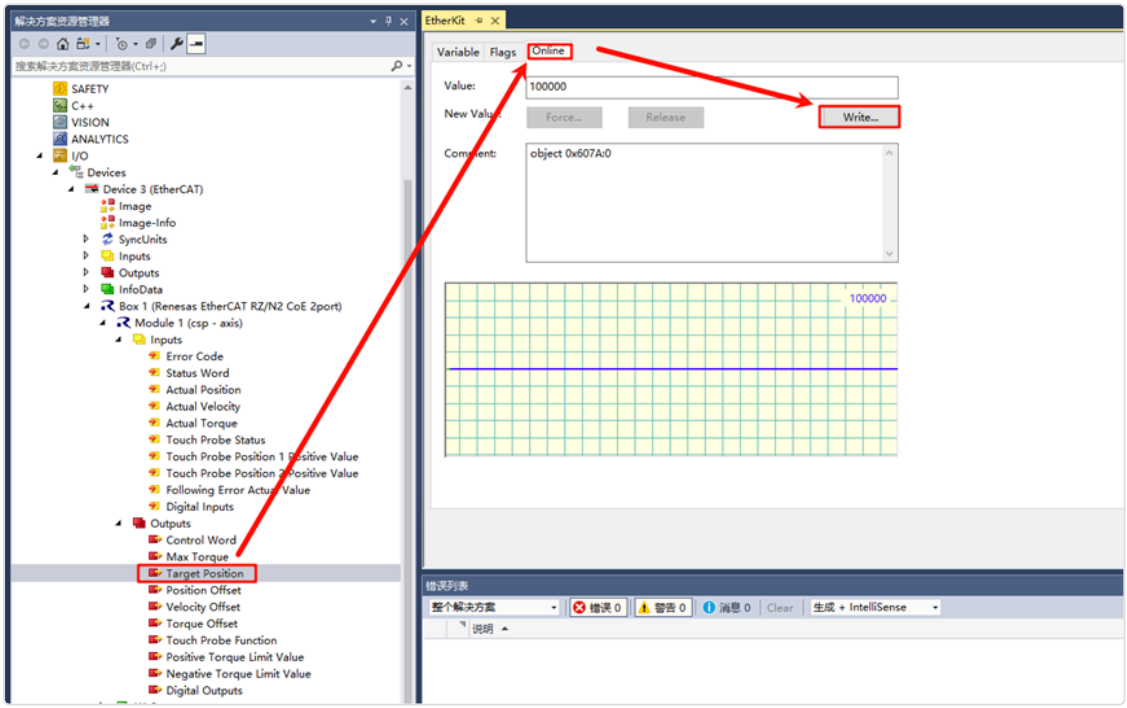


The servo controller will transition from waiting for servo enable to the **Servo Running** state. Meanwhile, the slave's serial terminal will print **State**

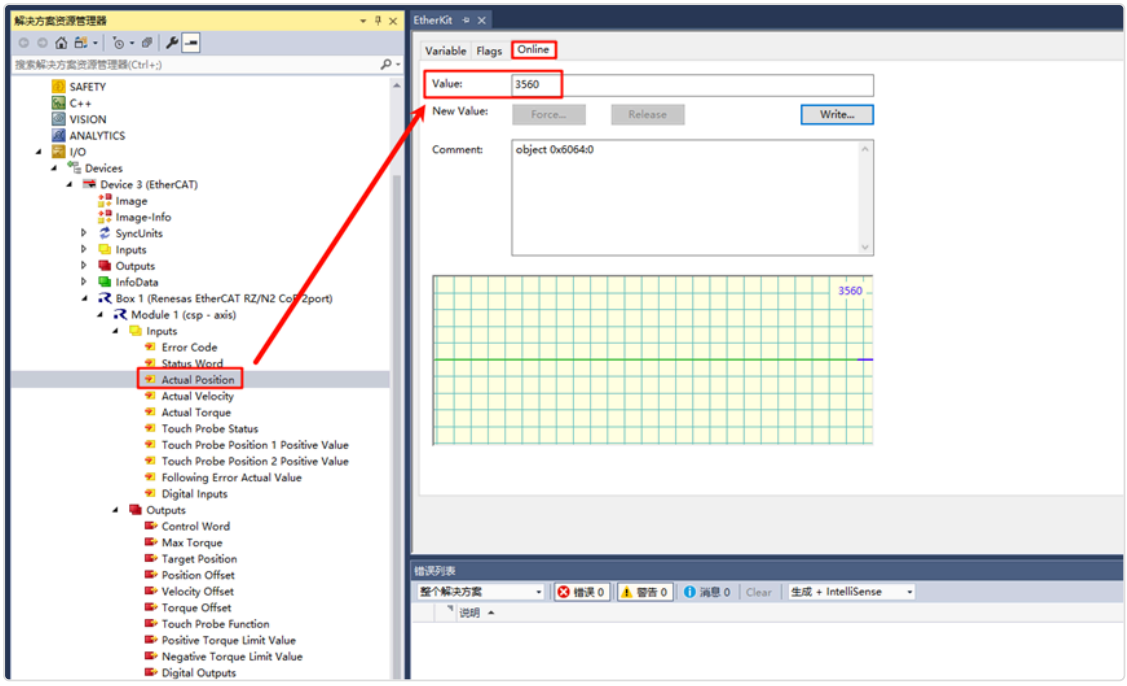
Transition2, **State Transition3**, and **State Transition4**. After these state transitions, the CiA402 state machine

will enter **STATE_OPERATION_ENABLED**, and the controller can now be controlled.

For example, in position mode, write a position value (e.g., **100000**) to **Index: 0x607A**:

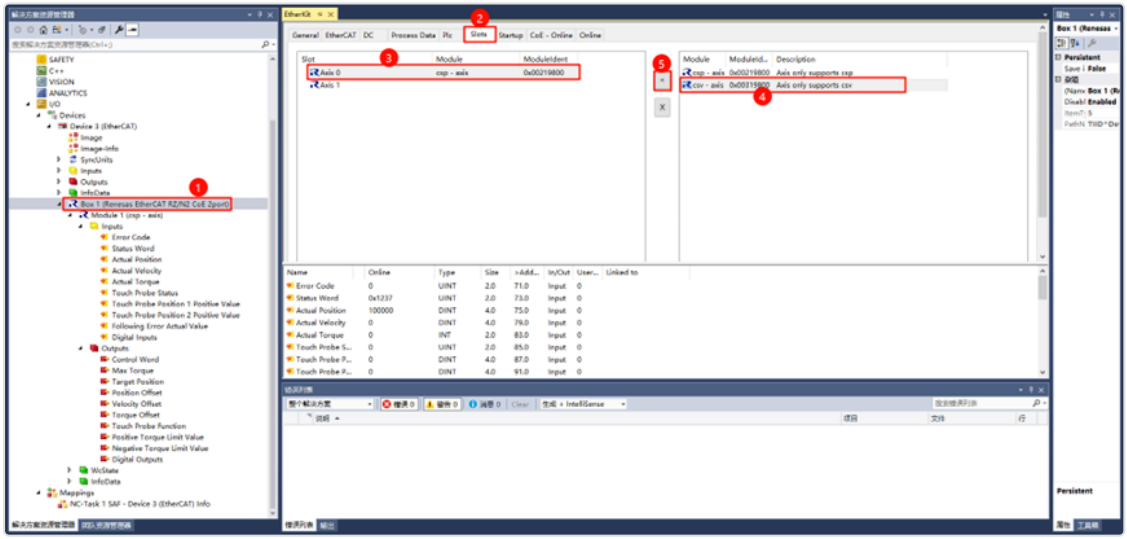


Then, click **Box 1 (Renesas EtherCAT RZ/N2 CoE 2port) → Module 1 (csp - axis) → Inputs → Actual Position** to view the actual feedback position. You will see that the value of **Index 0x6064** increases incrementally until it reaches **100000**:

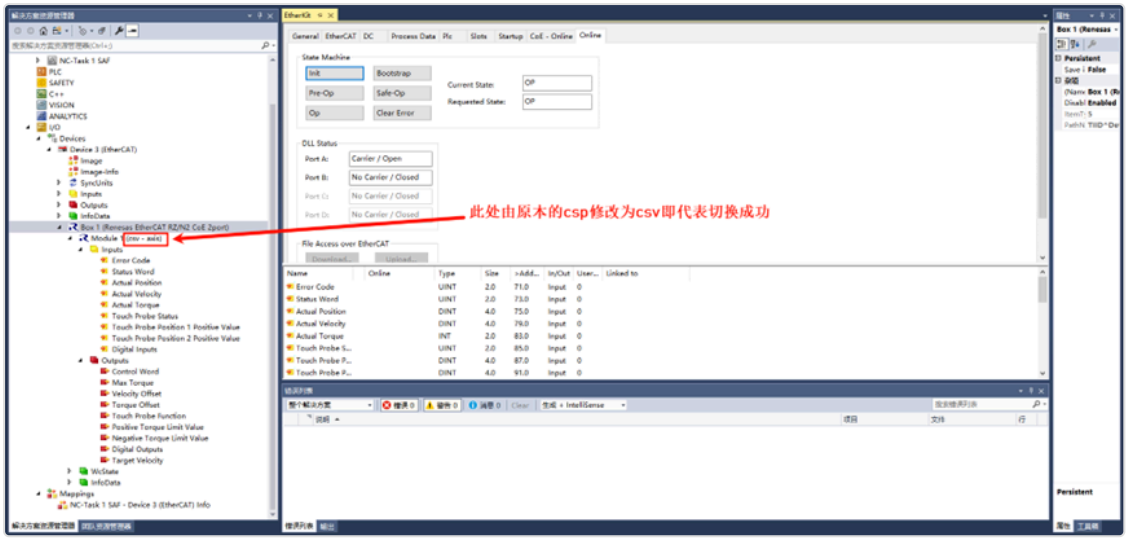


csv Speed Mode Control

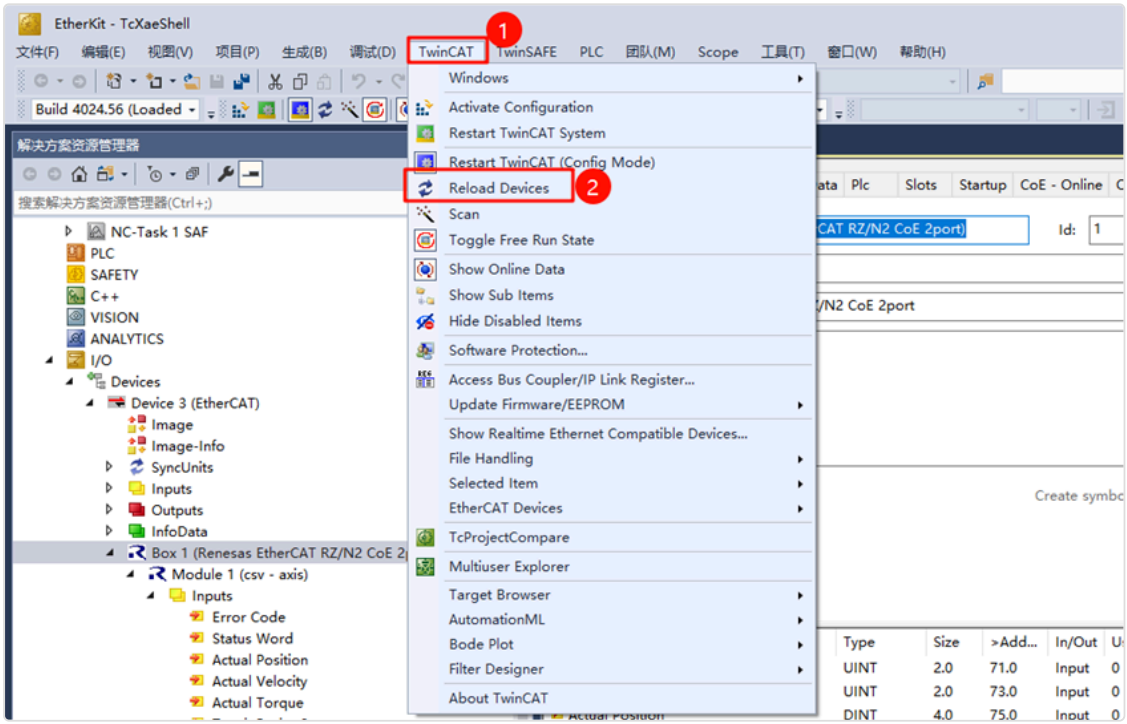
First, switch the controller mode from **csp** to **csv** mode. Click **Box 1 (Renesas EtherCAT RZ/N2 CoE 2port)** in the left navigation bar, then find **Axis 0** in the middle panel, and change the supported module to **csv**. Click the **<** symbol:



Observe if the module information on the left has been updated to **csv** mode:



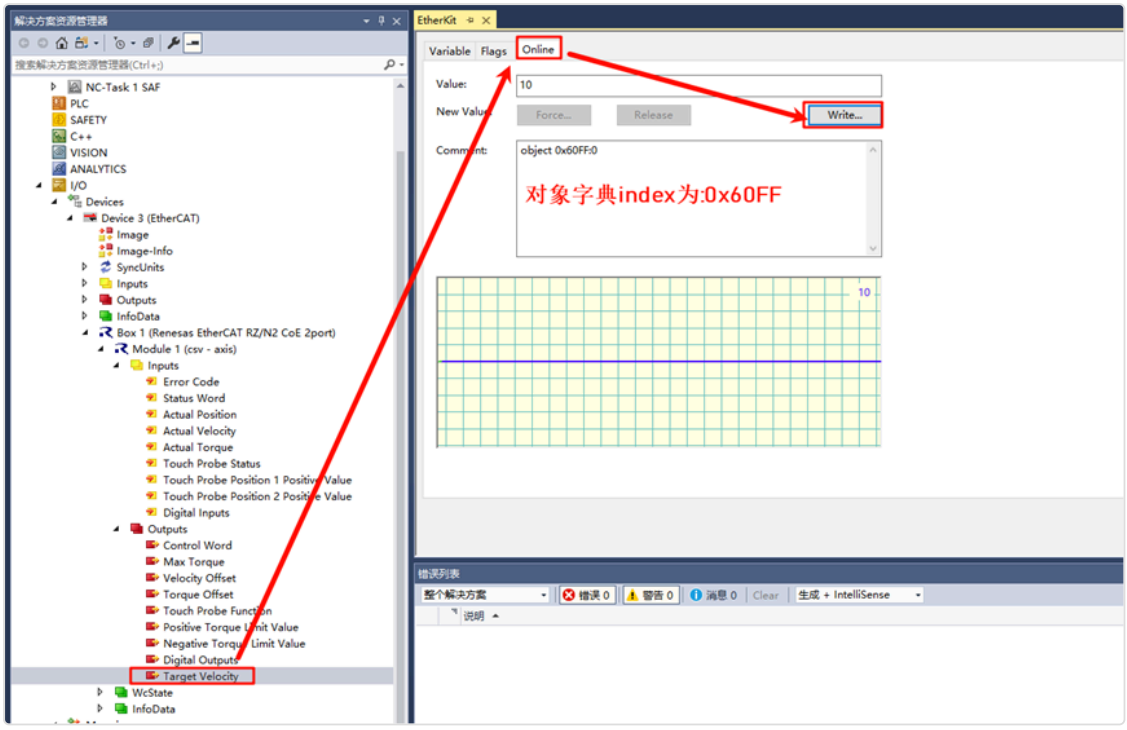
Once the mode has been switched, reload the device by clicking **TwinCAT → Reload Devices** in the top navigation bar:



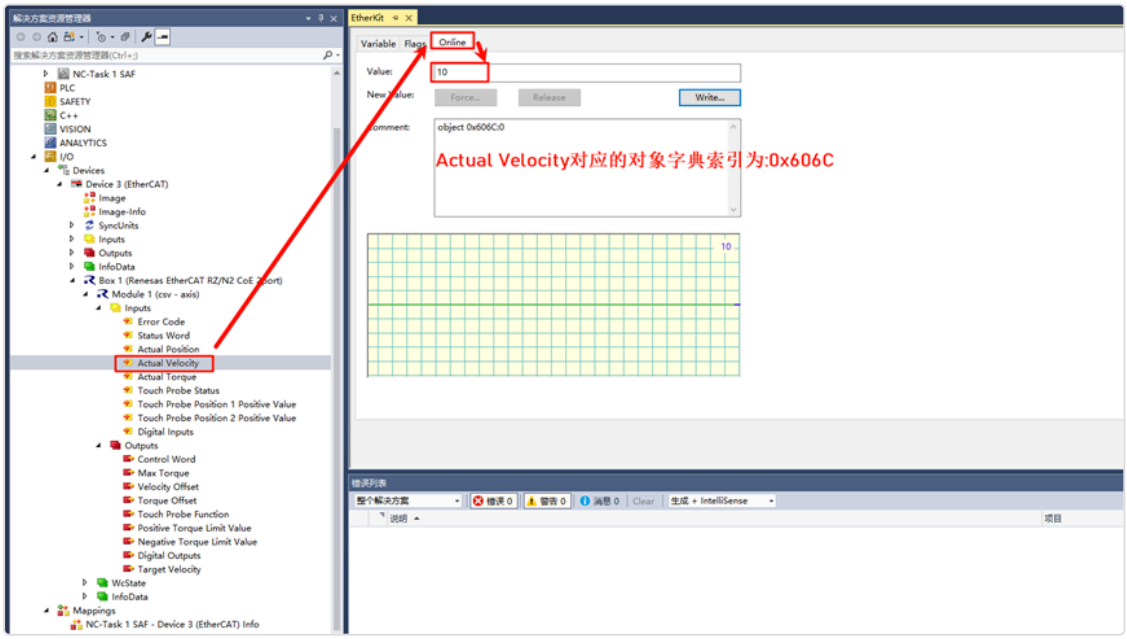
Now, switch the controller to **STATE_OPERATION_ENABLED** (operational mode, as mentioned above). Write **0x0080** (dec: 128) to change the state to an unobstructed state, then write **0x000F** (dec: 15) to transition from waiting for servo enable to servo running state.

Next, check the **Status Word (0x6041)**. If the value is **0x1237**, it indicates that the system is in **STATE_OPERATION_ENABLED**. If the value is **0x1208**, it indicates that the status is in **Fault**. Reconfigure the control word to **0x0080** (dec: 128) and repeat the above operations.

Now you can write the **Target Velocity** value to control the speed:



Also, check the actual set speed information in the inputs to ensure it is consistent:



5.2. EtherCAT EOE Usage Instructions

English | **中文**

Introduction

EtherCAT EoE (**Ethernet over EtherCAT**) is a communication protocol in the EtherCAT standard, designed to transfer standard Ethernet packets over an EtherCAT network. It allows non-real-time Ethernet communication to coexist with real-time EtherCAT communication, providing flexible network integration for industrial automation systems.

The main features and functions of EoE are as follows:

1. **Ethernet Tunnel Transmission**:

- EoE encapsulates standard Ethernet packets within EtherCAT communication frames, enabling standard Ethernet protocols (such as TCP/IP, UDP, HTTP, etc.) to be transmitted over an EtherCAT network.

2. **Extended Network Functionality**:

- EoE allows EtherCAT slaves to function as virtual Ethernet devices within a TCP/IP network.
- It enables remote standard Ethernet devices to be accessed via the EtherCAT communication link.

3. **Efficient Integration**:

- The implementation of EoE does not affect the real-time performance of EtherCAT.
- Non-real-time Ethernet communication and real-time EtherCAT data exchange can coexist, each serving its respective purpose.

4. **Usage Scenarios**:

- **Device Management**: Access EtherCAT slave devices via IP for remote configuration, diagnostics, and firmware updates.
- **Hybrid Networks**: Integrate devices that require standard Ethernet communication (such as cameras, sensors, or industrial PCs).

5. **Simplified Network Wiring**:

- In industrial automation scenarios, EoE allows Ethernet devices to be accessed over the EtherCAT network, reducing the need for separate Ethernet cabling.

6. **Typical Applications**:

- Remote monitoring and diagnostics in factory automation systems.
- Communication bridging between industrial robots or production equipment and external IT systems.

This section demonstrates how to implement EtherCAT EoE master-slave communication using Beckhoff TwinCAT3 and the EtherKit development board.

Prerequisites

Software Environment:

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

Hardware Environment:

- EtherKit development board
- One Ethernet cable
- Jlink debugger

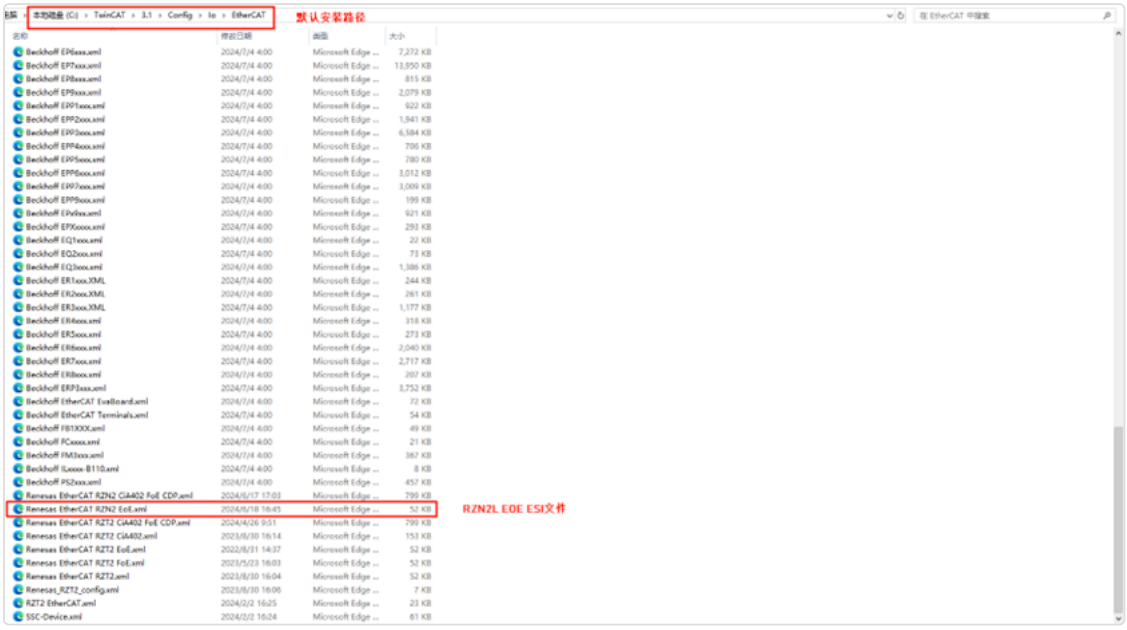
TwinCAT3 Configuration

> *Before launching TwinCAT3, some configuration steps are necessary:*

Install the ESI File

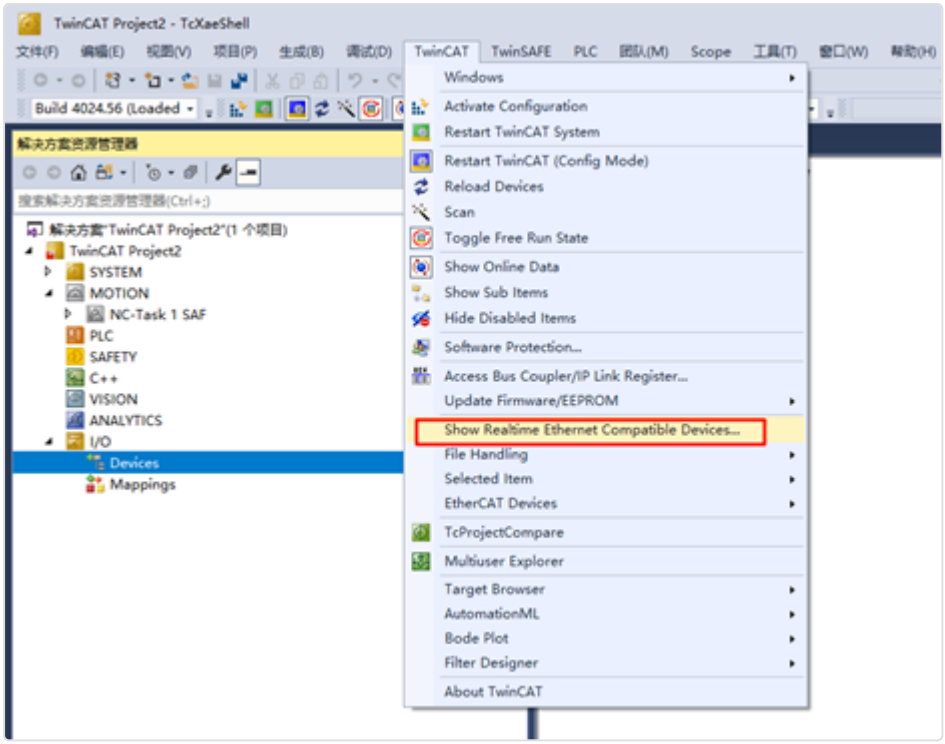
Before launching TwinCAT, copy the ESI file included in the release folder to the TwinCAT target directory: `..\TwinCAT\3.x\Config\IO\EtherCAT`

> *Note: The current version of the ESI file is located at:* `..\board\ports\ethercat\ESI_File\Renesas EtherCAT RZT2 EoE.xml`

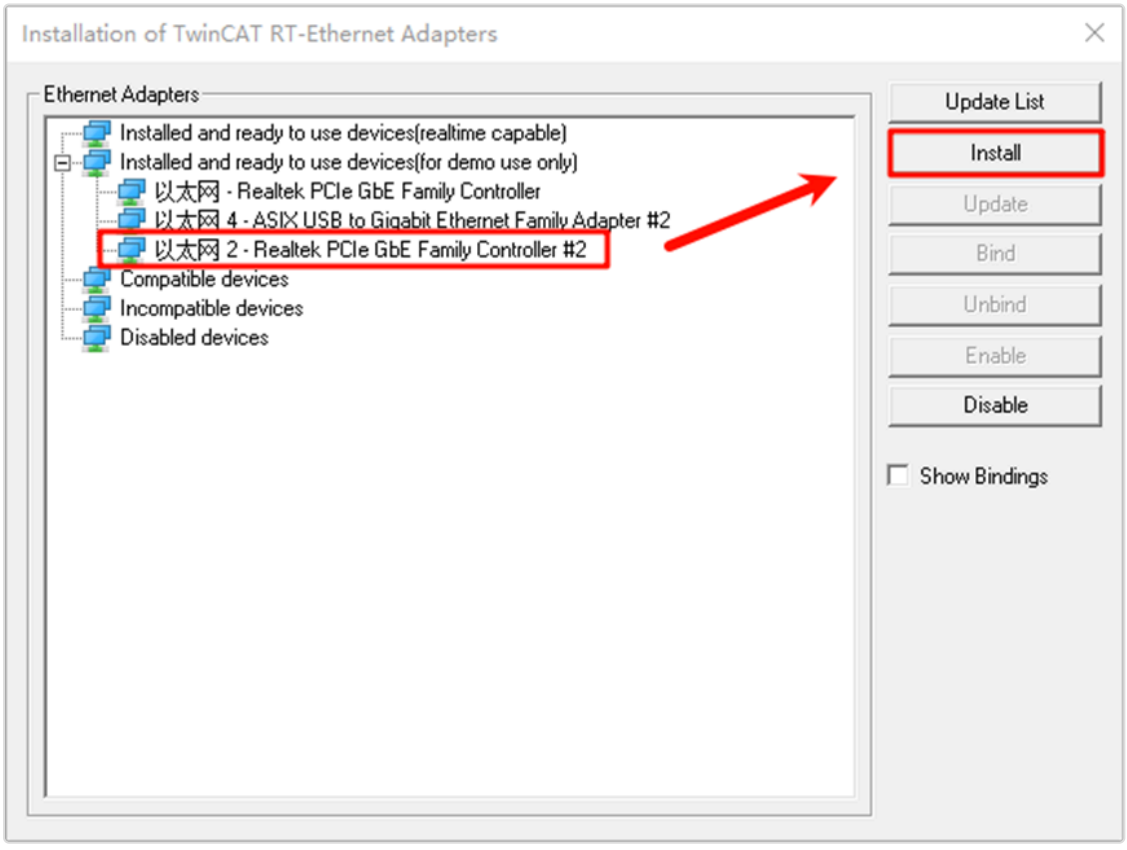


Add TwinCAT Ethernet Driver

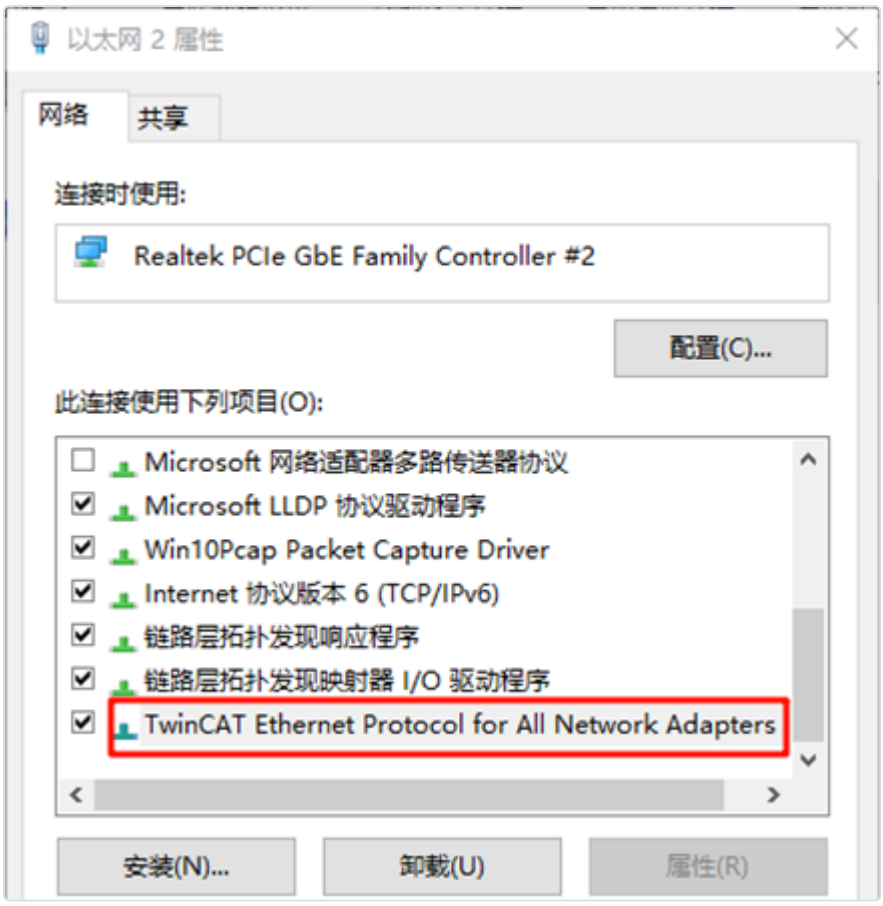
Add the TwinCAT Ethernet driver (only required for initial configuration). From the Start menu, select [TwinCAT] → [Show Realtime Ethernet Compatible Device…], choose the connected Ethernet port from the communication ports and install it.



Here, we can see all the Ethernet adapters on the PC. After selecting the port we are testing, click Install:

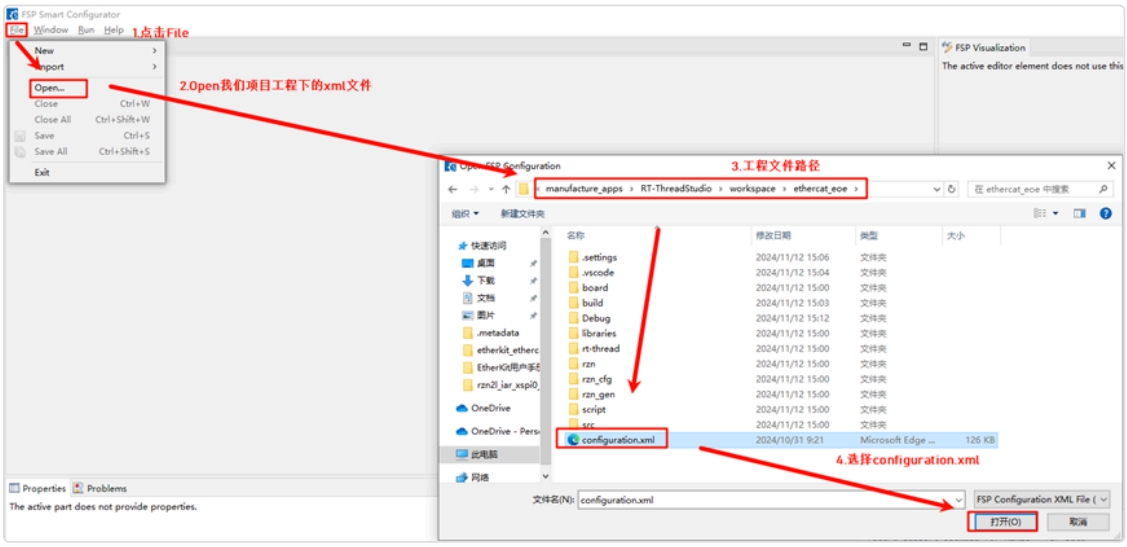


Check the network adapter to see that it has been successfully installed:

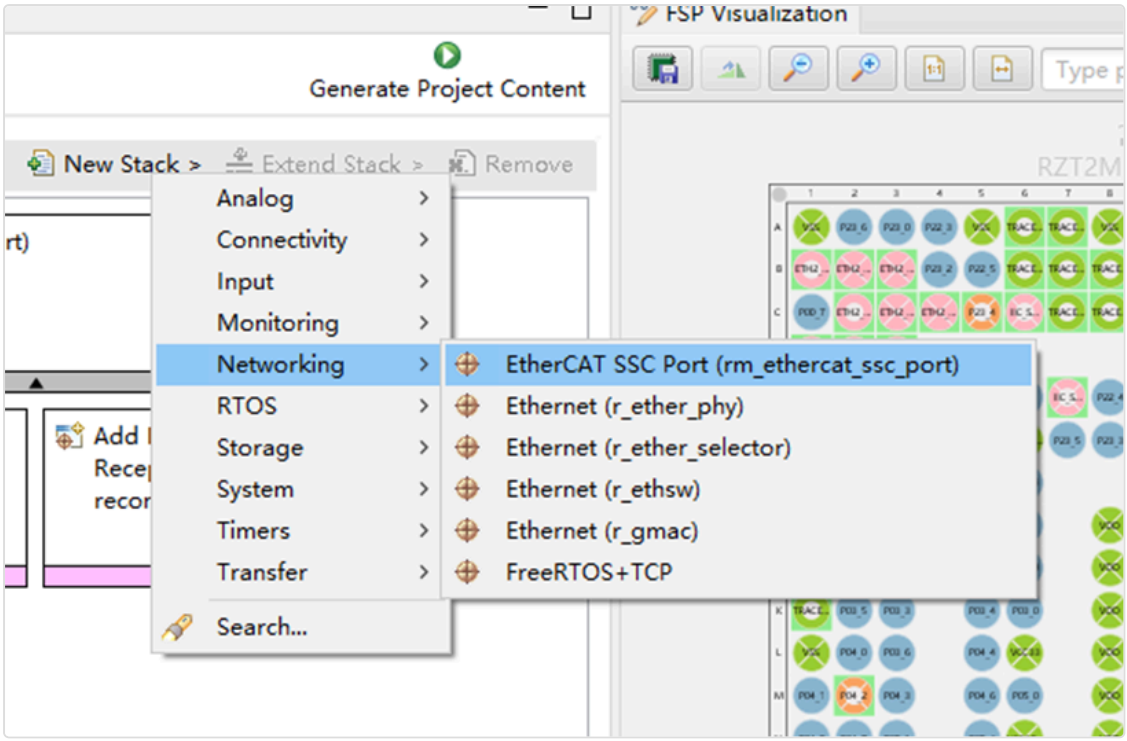


FSP Configuration Instructions

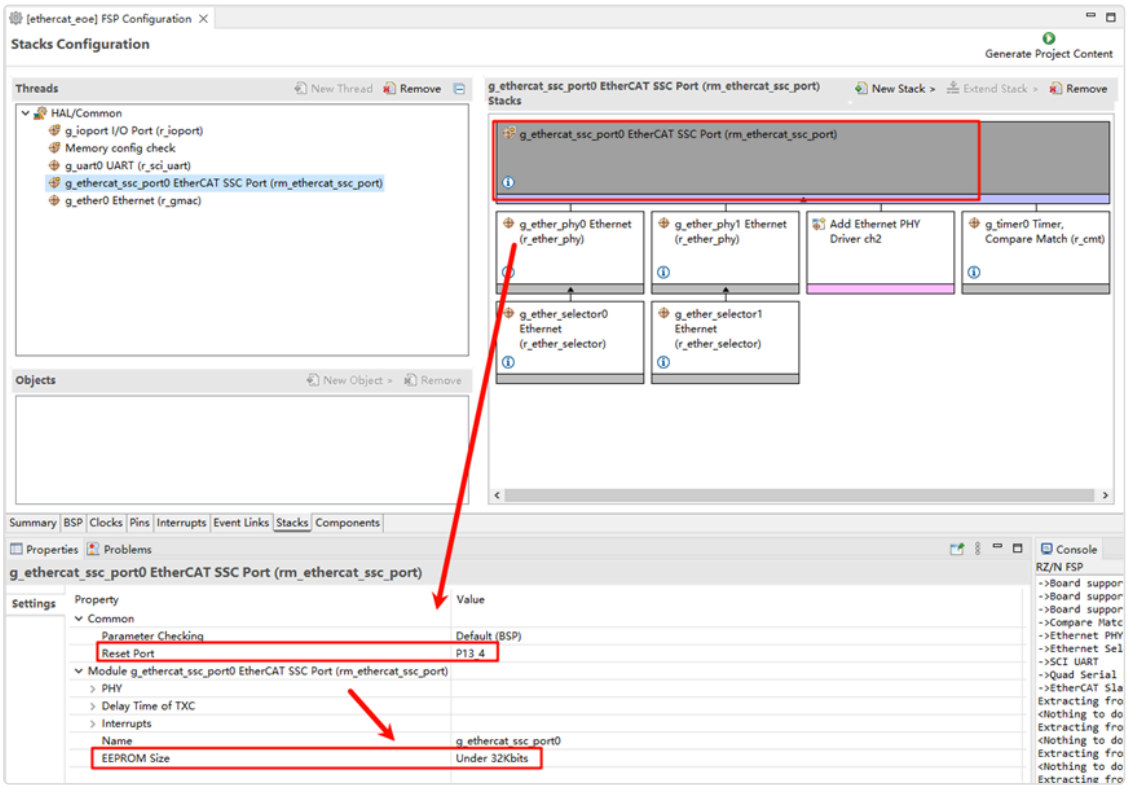
Next, configure the pin initialization. Open the installed RZN-FSP 2.0.0 and select the root directory of your project:



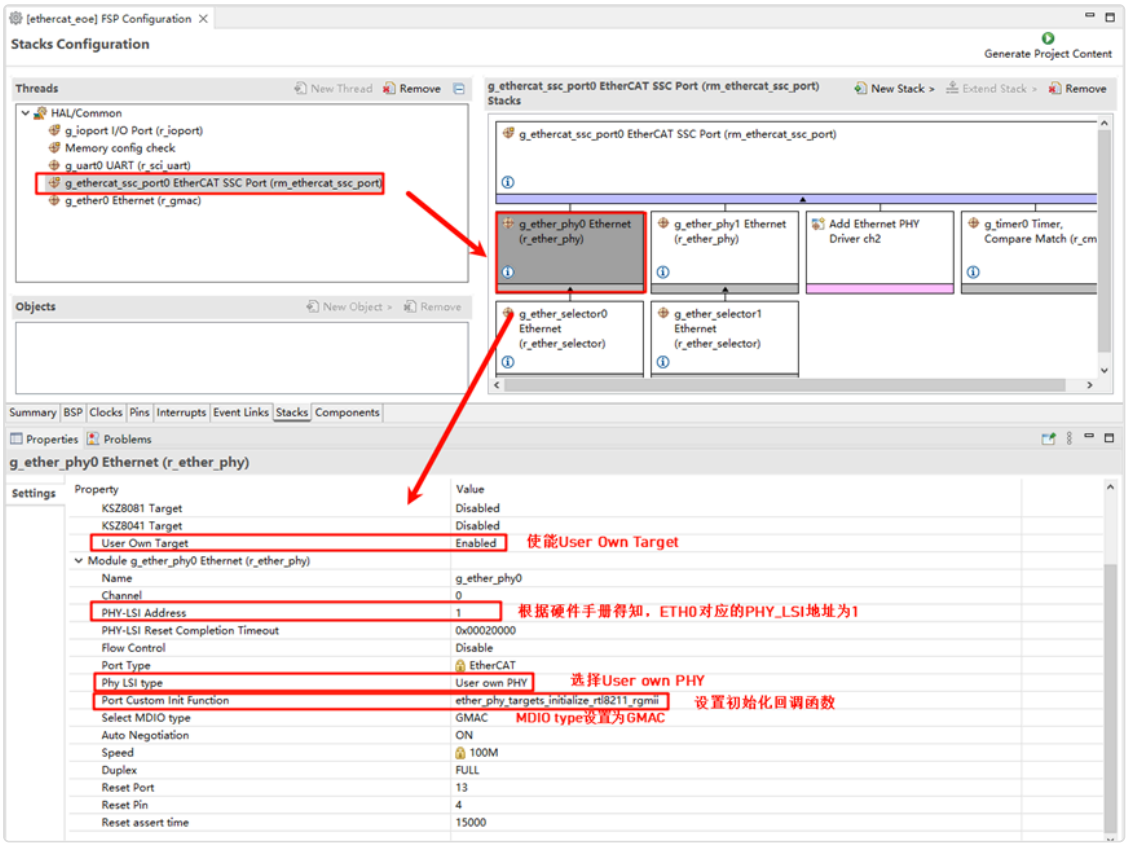
We will configure the following peripherals and pins: Click New Stack and add the `ethercat_ssc_port` peripheral:



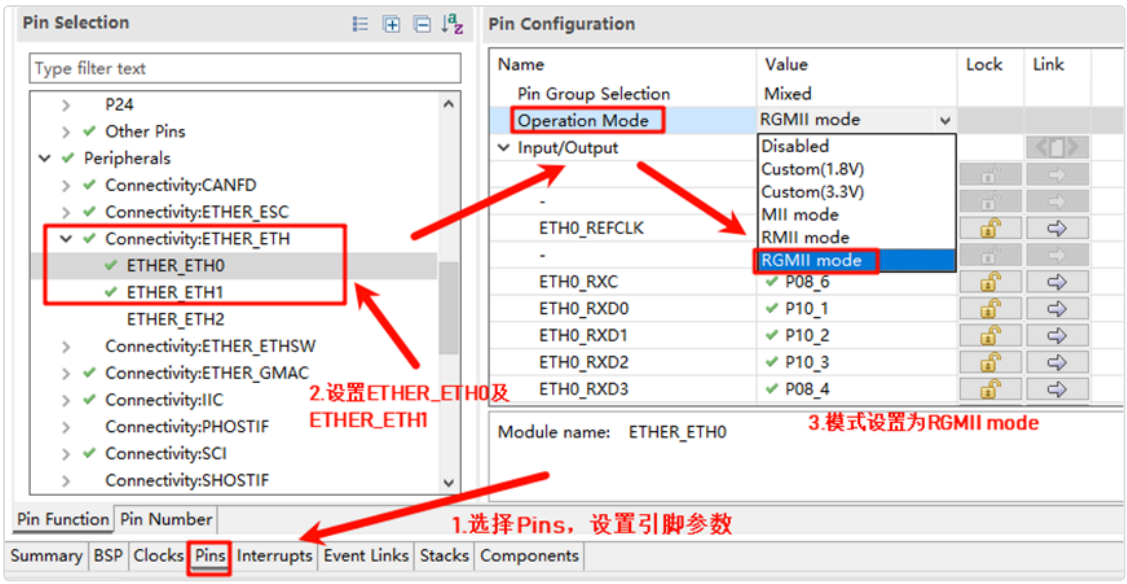
Configure `ethercat_ssc_port`: Change the Reset Port to `P13_4` and set the EEPROM size to "Under 32Kbits":



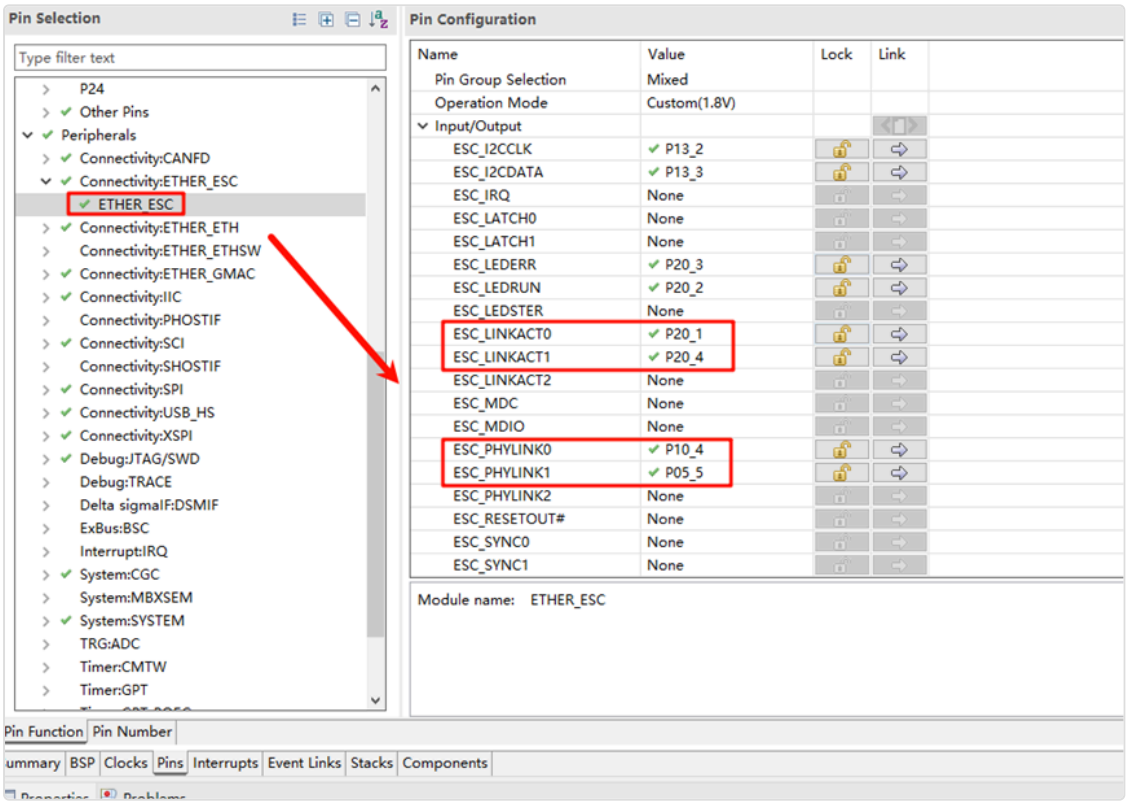
Enable the Ethernet card type and configure the network card device parameters. Here we add two PHYs (`phy0` and `phy1`). Note that EtherKit uses the RTL8211 Ethernet card, which is not supported by Renesas FSP. However, Renesas provides an interface for user-defined network cards, so we configure the card accordingly, setting the MDIO type to GMAC and the network card initialization callback function to `ether_phy_targets_initialize_rtl8211_rgmii()` :



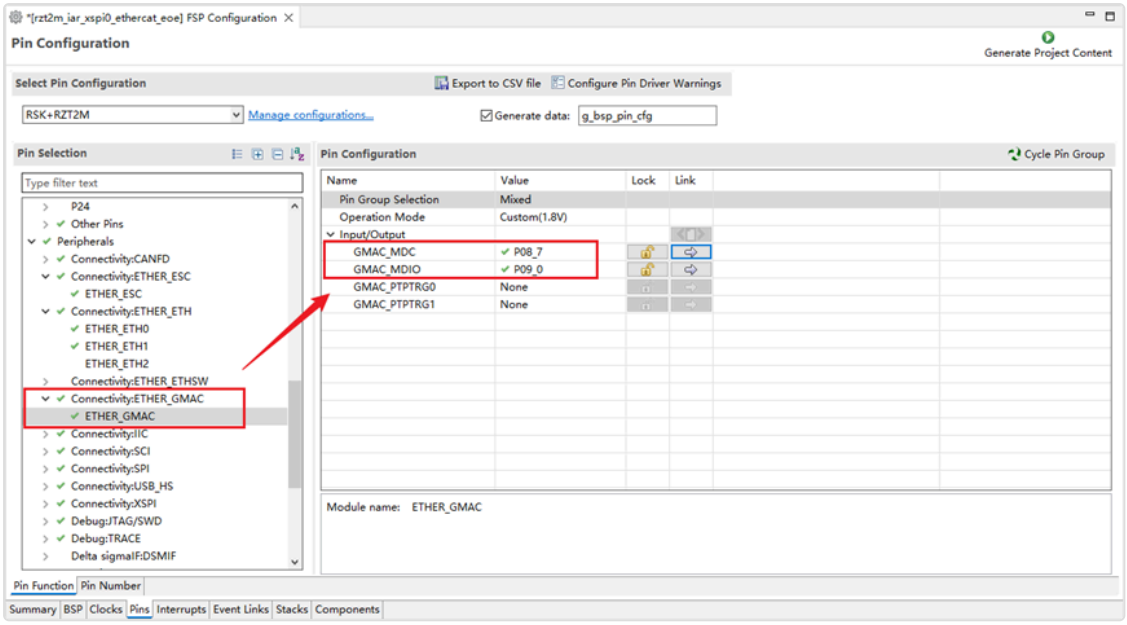
Configure the Ethernet pin parameters, setting the operation mode to RGMII:



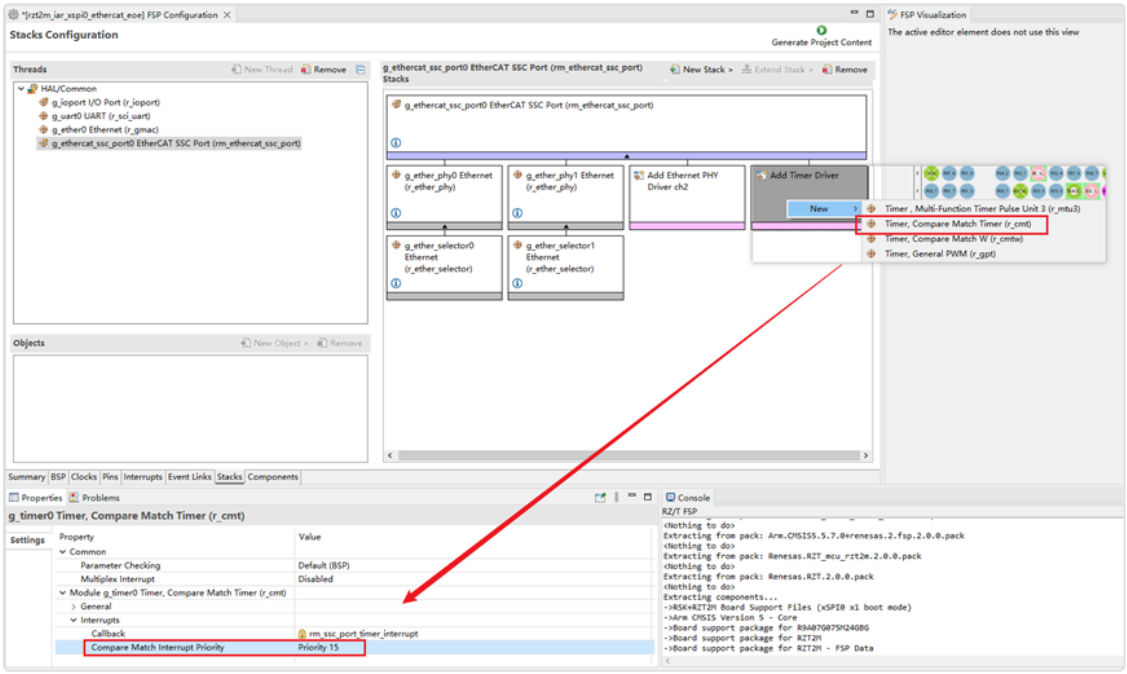
ETHER_ESC Settings:



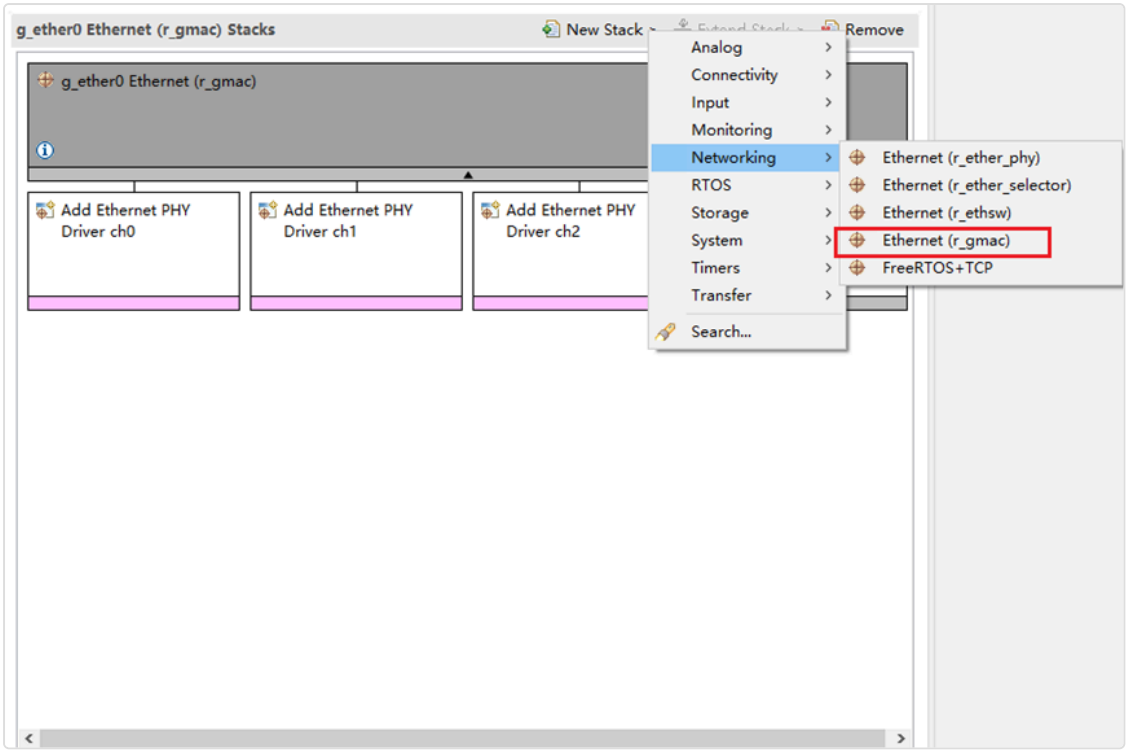
ETHER_GMAC Configuration:



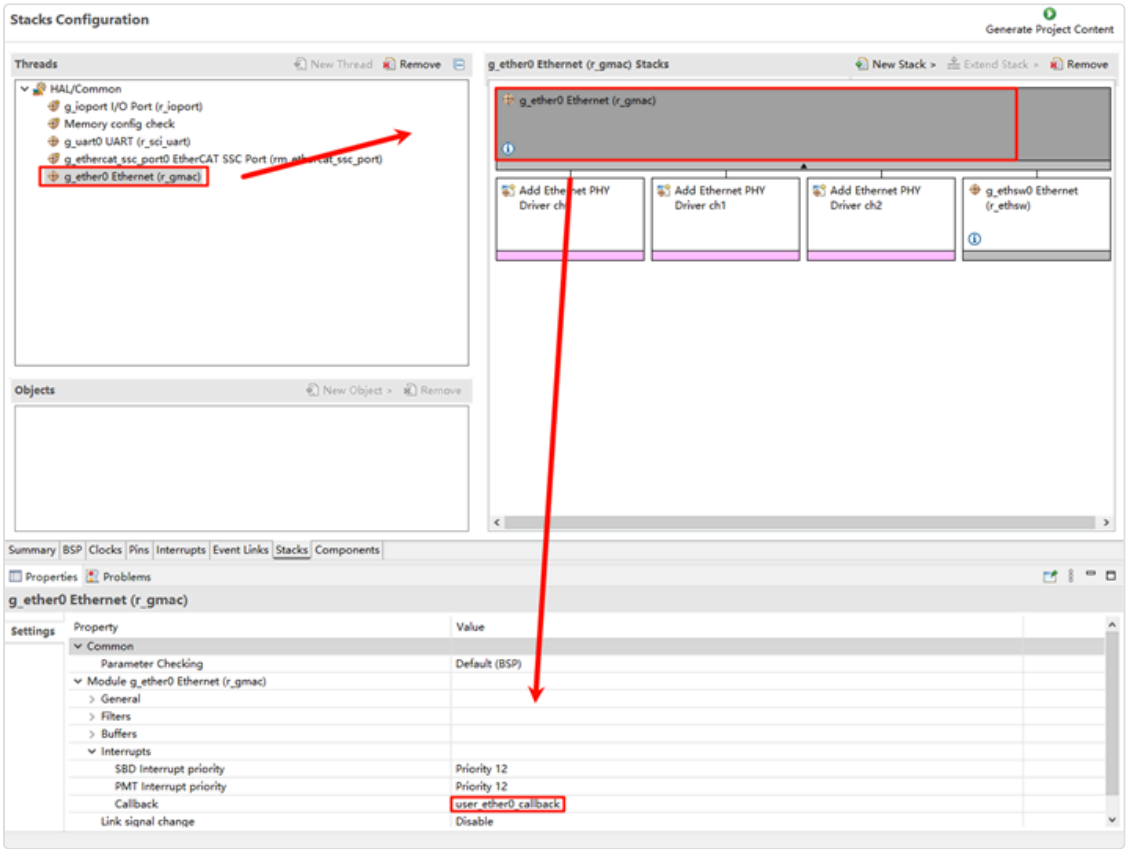
Add a CMT timer to `ethercat_ssc_port` and configure the interrupt priority:



Add the Ethernet peripheral:



Set the Ethernet interrupt callback to: `user_ether0_callback`



Finally, click Generate Project Content to generate the underlying driver source code.

Build Configuration

1. **Modify SConscript**: Navigate to the project and locate the file at the specified path: `.\rzn\SConscript`. Replace the file content with the following:

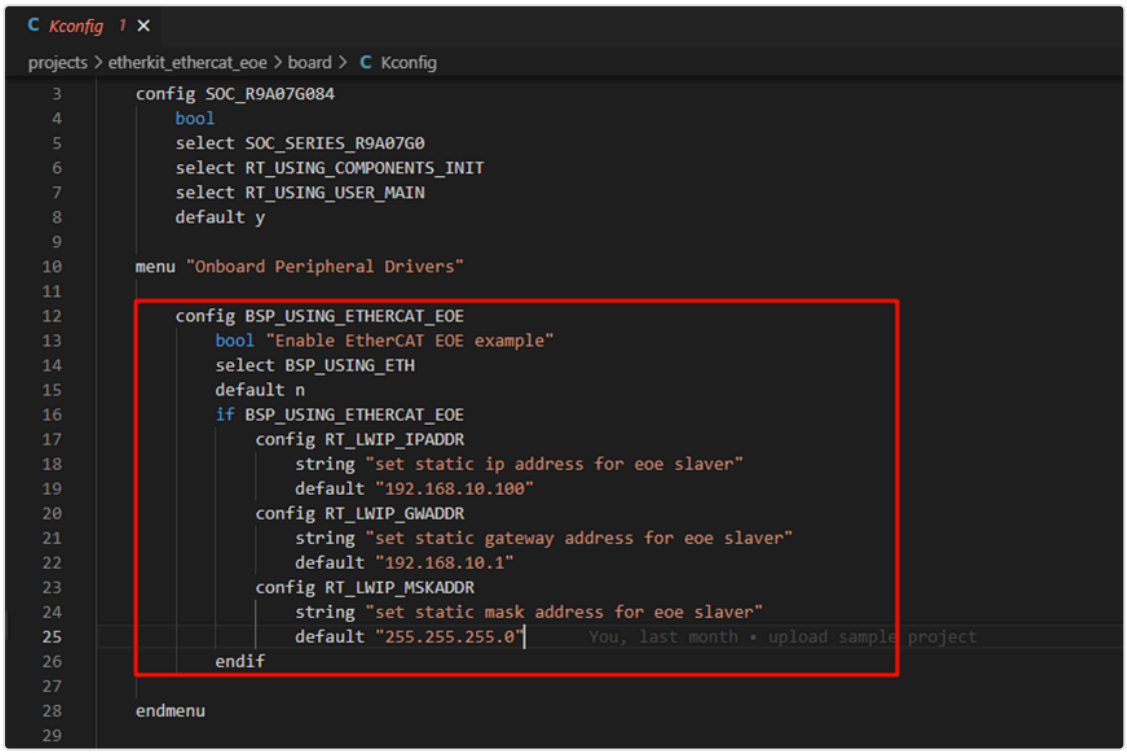
```
Import('RTT_ROOT')
Import('rtconfig')
from building import *
from gcc import *

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

if rtconfig.PLATFORM in ['iccarm'] + GetGCCLikePLATFORM():
    if rtconfig.PLATFORM == 'iccarm' or GetOption('target') !=
        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/
        src += Glob('./fsp/src/bsp/cmsis/Device/RENESAS/Source/
        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 = CPPPAT
Return('group')
```

2. **Modify Kconfig**: Open the file located at `projects\etherkit_ethercat_eoe\board\Kconfig`. Add the EOE configuration under the *Onboard Peripheral Drivers* section:

```
        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 slave
                    default "192.168.10.100"
                config RT_LWIP_GWADDR
                    string "set static gateway address for eoe
                    default "192.168.10.1"
                config RT_LWIP_MSKADDR
                    string "set static mask address for eoe sla
                    default "255.255.255.0"
            endif
```

As shown in the following figure:



3. **Development Environment**:

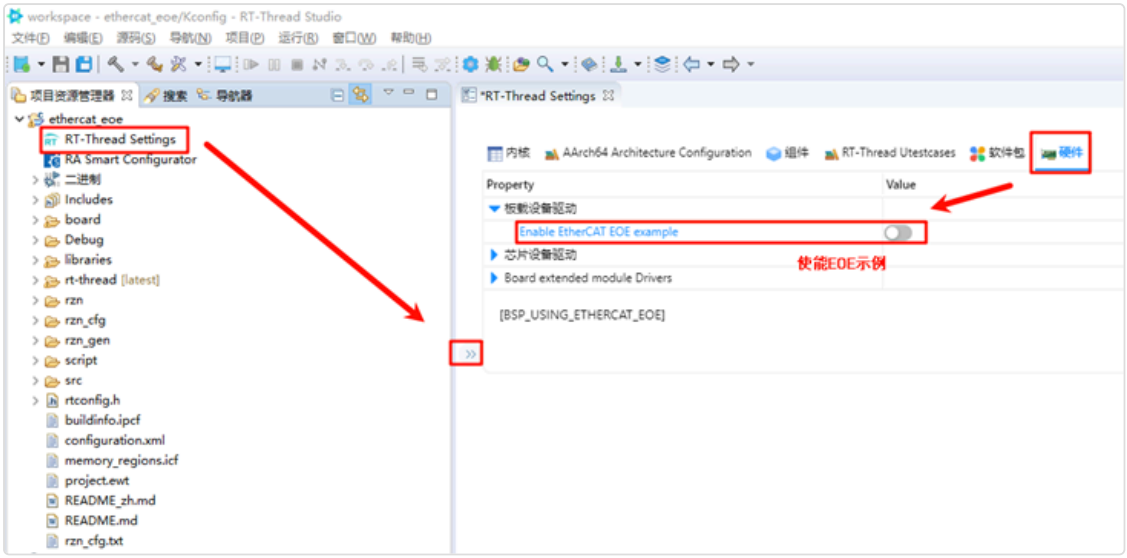
- If you are using Studio for development, right-click the project and select **Sync SCons Configuration to Project**.
- If you are using IAR for development, right-click in the current project directory to open the environment and execute:

```
    scons --target=iar
```

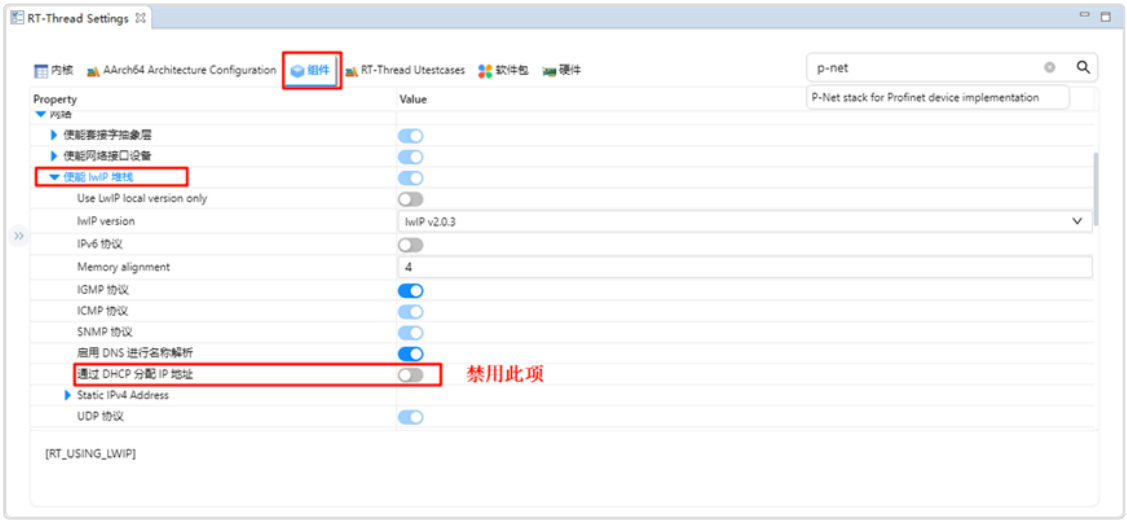
```
to regenerate the configuration.
```

RT-Thread Studio Configuration

After completing the FSP configuration, pin and peripheral initialization is done. Now, we need to enable the EtherCAT EOE example. Open Studio, click RT-Thread Settings, and enable the EOE example:



Next, we need to configure the system to disable DHCP and use a static IP. Click on the component -> enable the lwip stack, and select to disable DHCP:



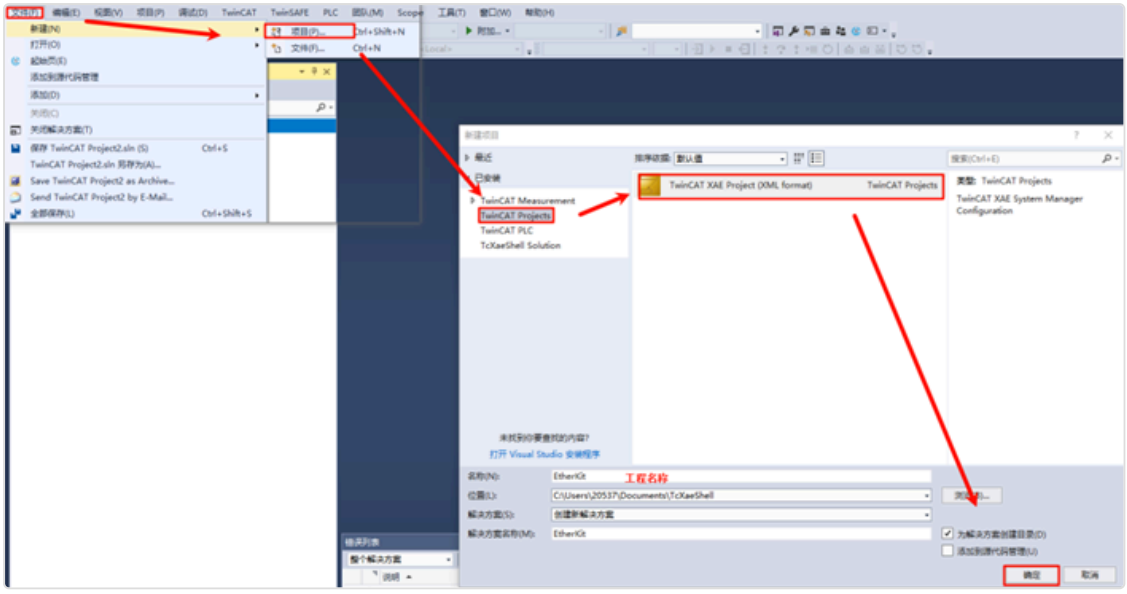
Once enabled, save the settings and synchronize the scons configuration. Then compile and download the program. After resetting the development board, observe the serial log:

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Nov 26 2024 16:00:59
 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: 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 >
```

EtherCAT EOE Configuration

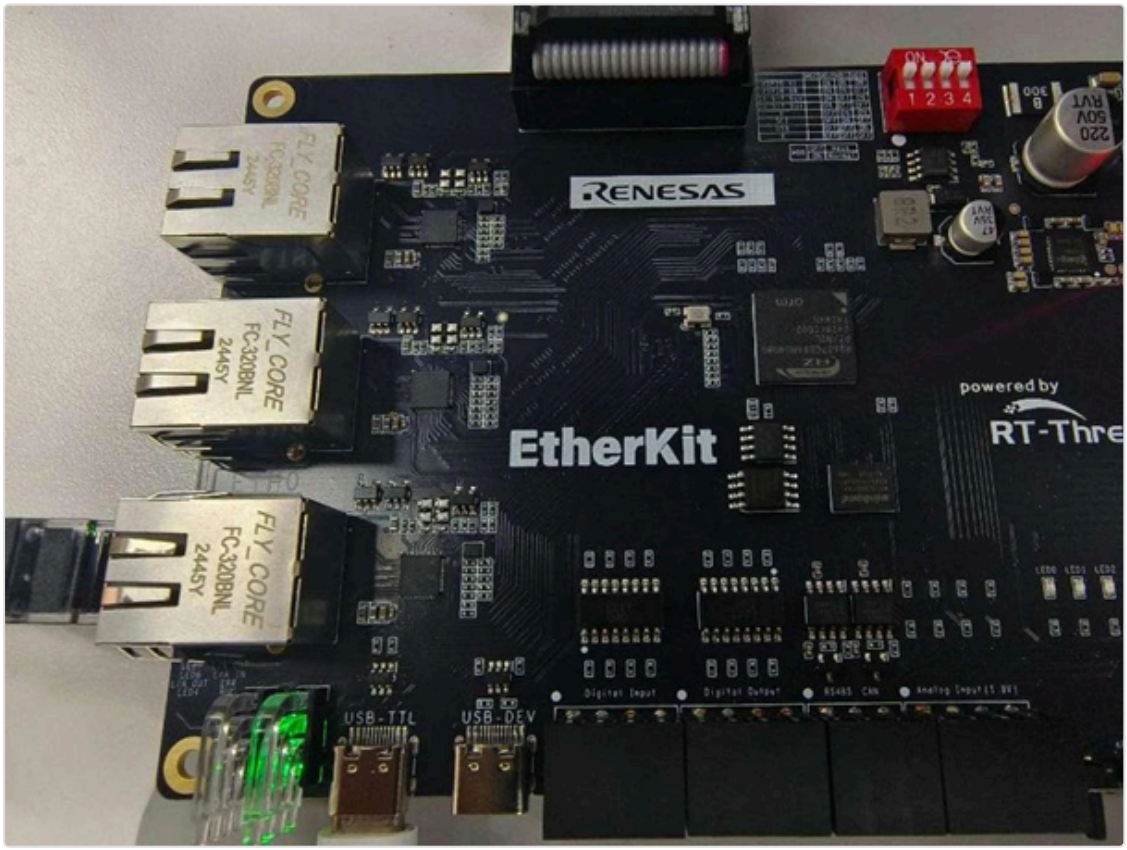
Create a New TwinCAT Project

Open TwinCAT software, click File → New → New Project, select TwinCAT Projects, and create a TwinCAT XAR Project (XML format):



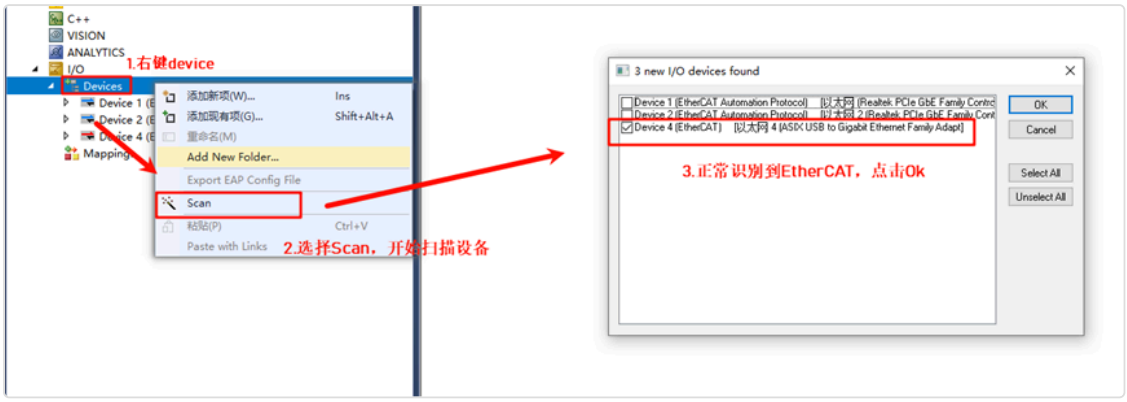
Start the EOE App on the Slave

After powering on the EtherKit development board, connect the Ethernet cable to the ETH0 port. EtherCAT will run by default.



Scan for Slave Devices

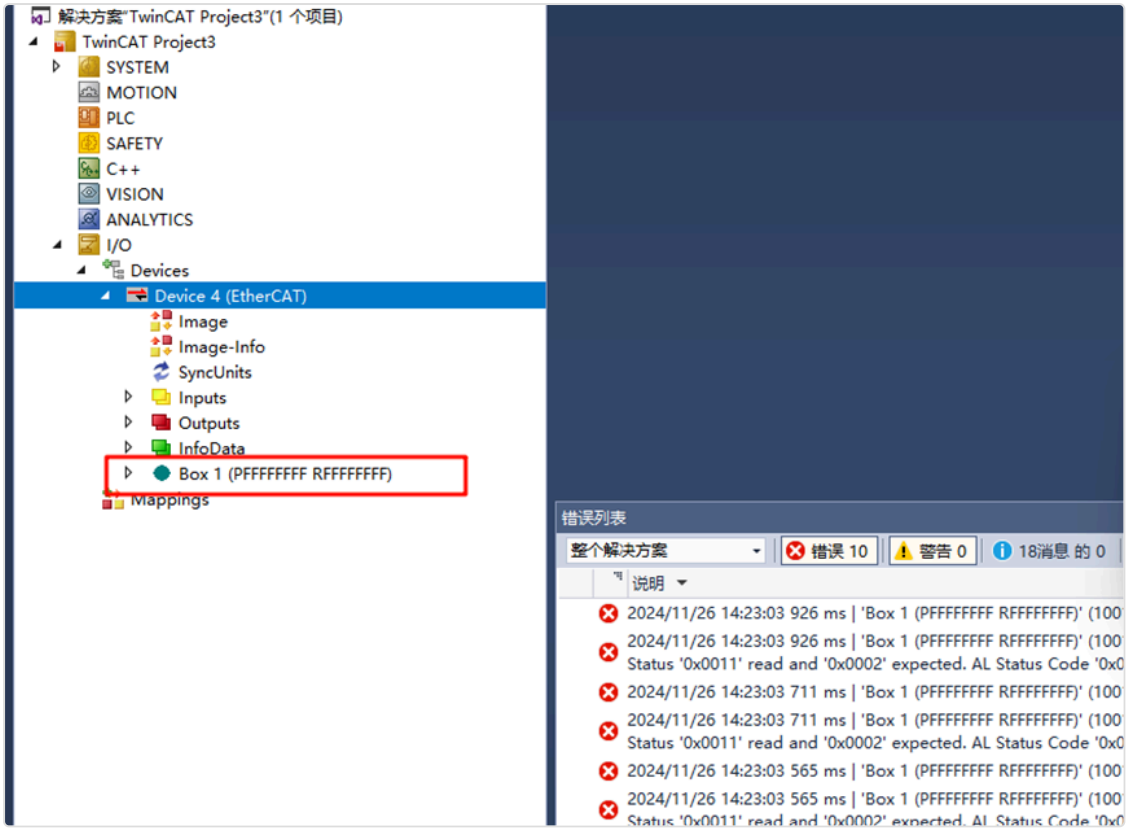
After creating the project, find Devices in the left sidebar, right-click, and select Scan Devices. Normally, if the slave device is scanned successfully, it will display: Device x [EtherCAT]. If the scan fails, it will show: Device x [EtherCAT Automation Protocol], indicating that the slave initialization failed.



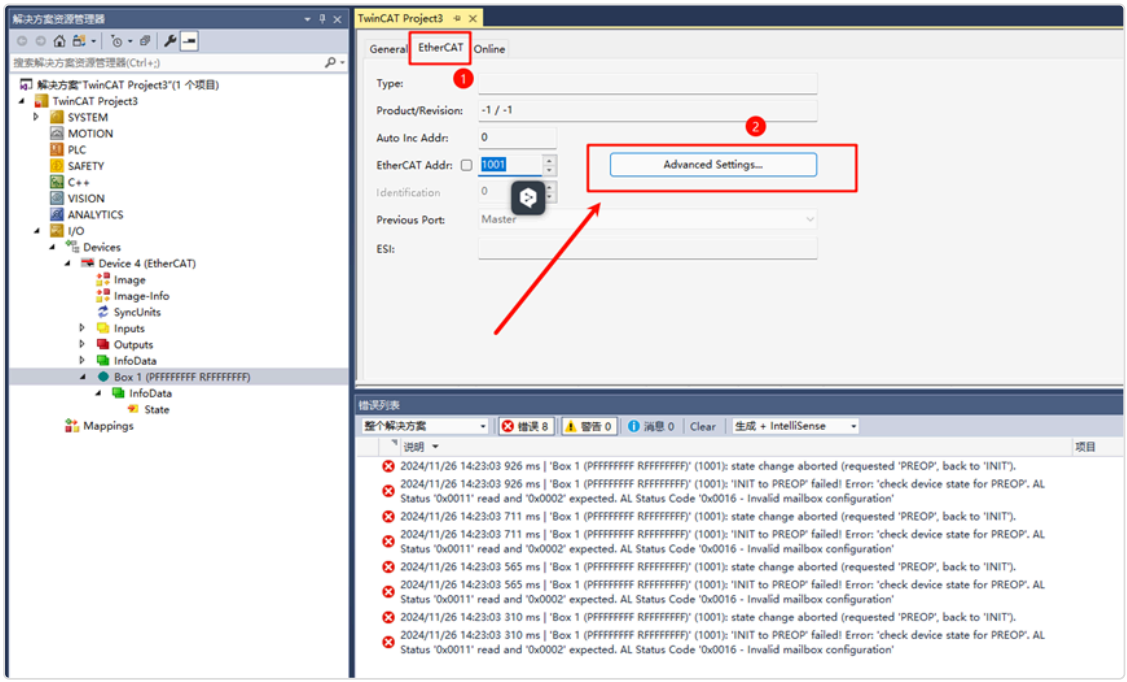
Click OK, and a window will appear: Scan for Boxes. Click Confirm, and another window will pop up: Activate Free Run. Since we are using EOE for the first time and need to update the EEPROM firmware, do not activate it yet.

Update EEPROM Firmware

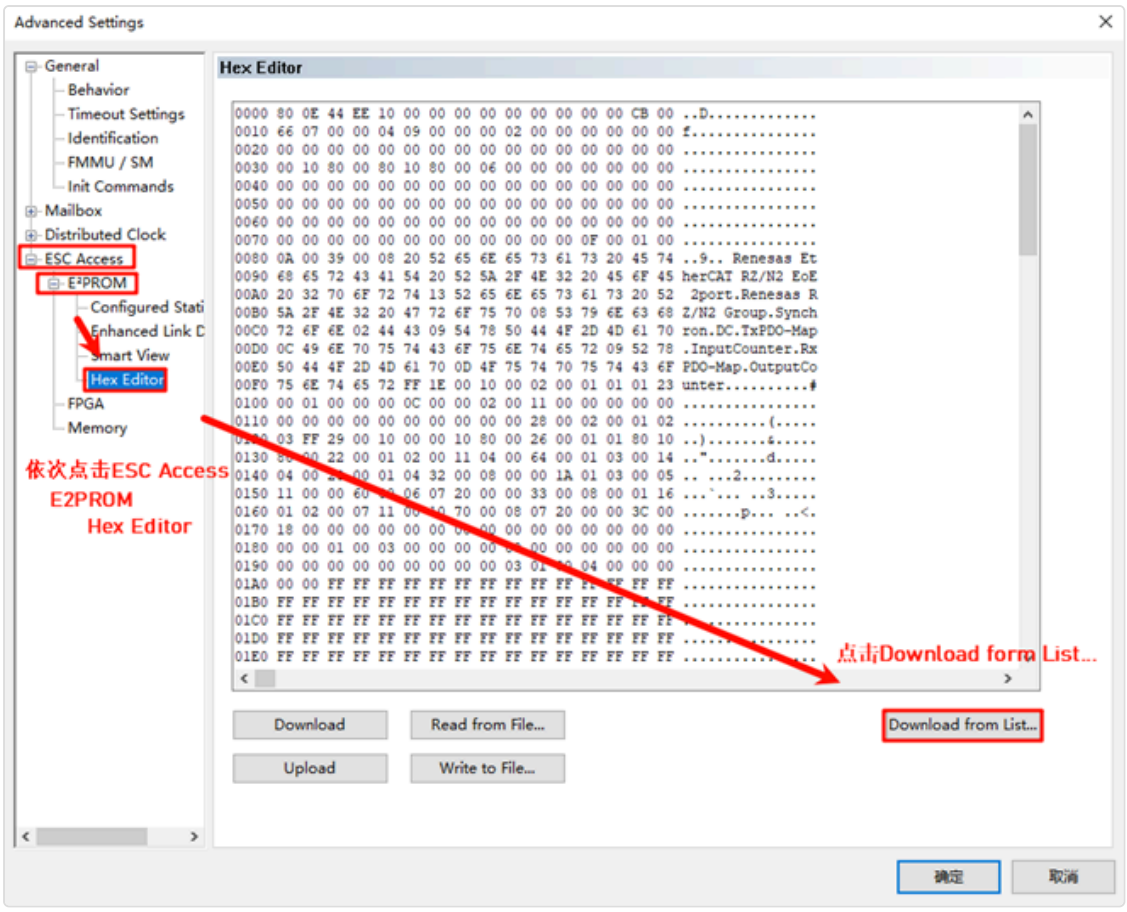
Back in TwinCAT, in the left sidebar, we can see the slave device after successfully scanning it, so we can access the master-slave configuration interface:



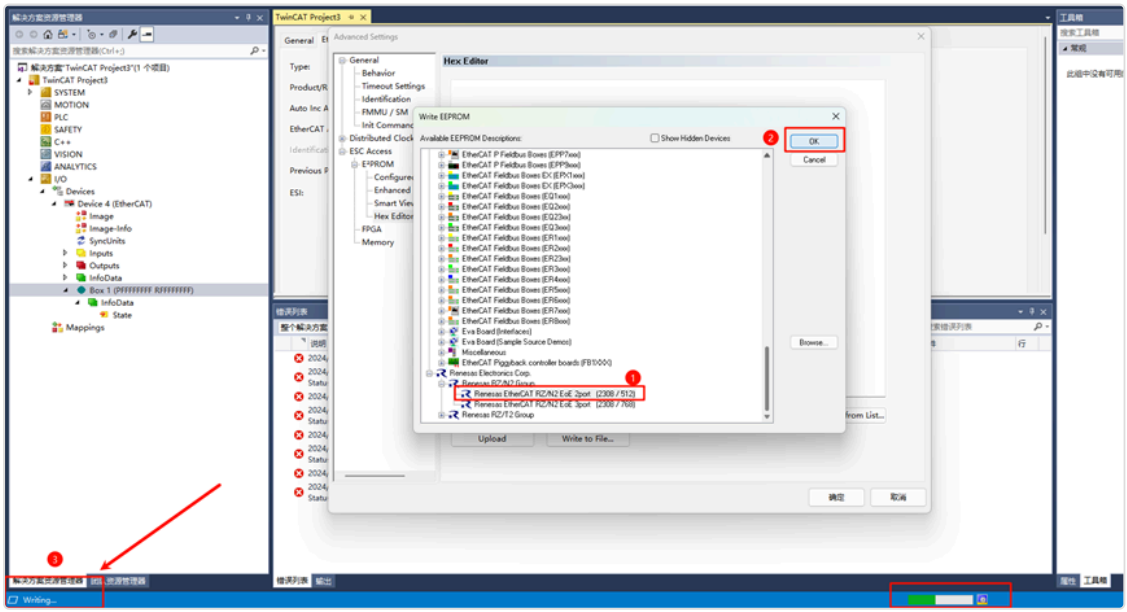
Double-click Box 1, then click EtherCAT in the top navigation bar and select Advanced Settings…:



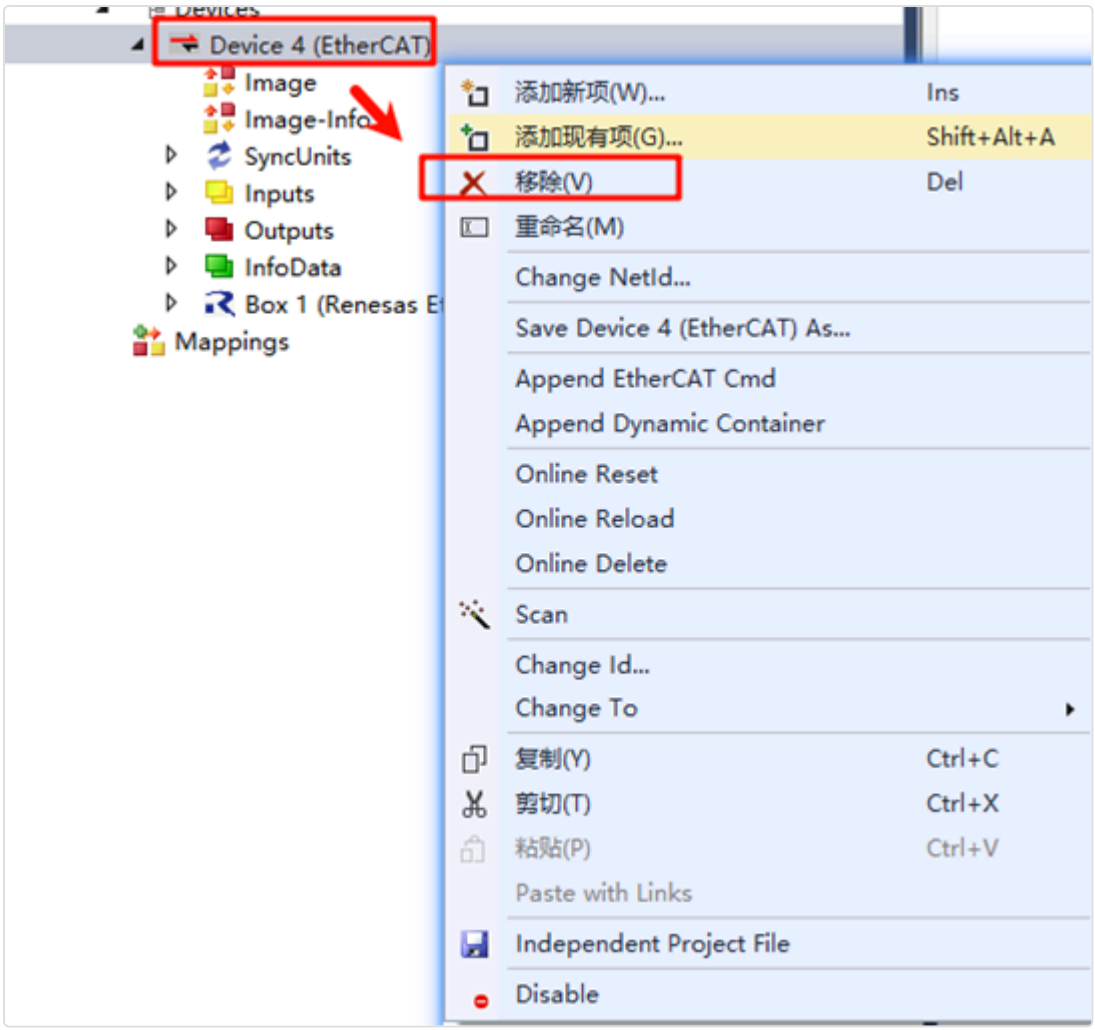
Click Download from List… as shown:



Write the ESI firmware to the EEPROM. Since we are configuring a dual-port setup, select `Renesas EtherCAT RZ/N2 EOE 2port`. If you are configuring a three-port setup, select the ESI file with the `3port` suffix.



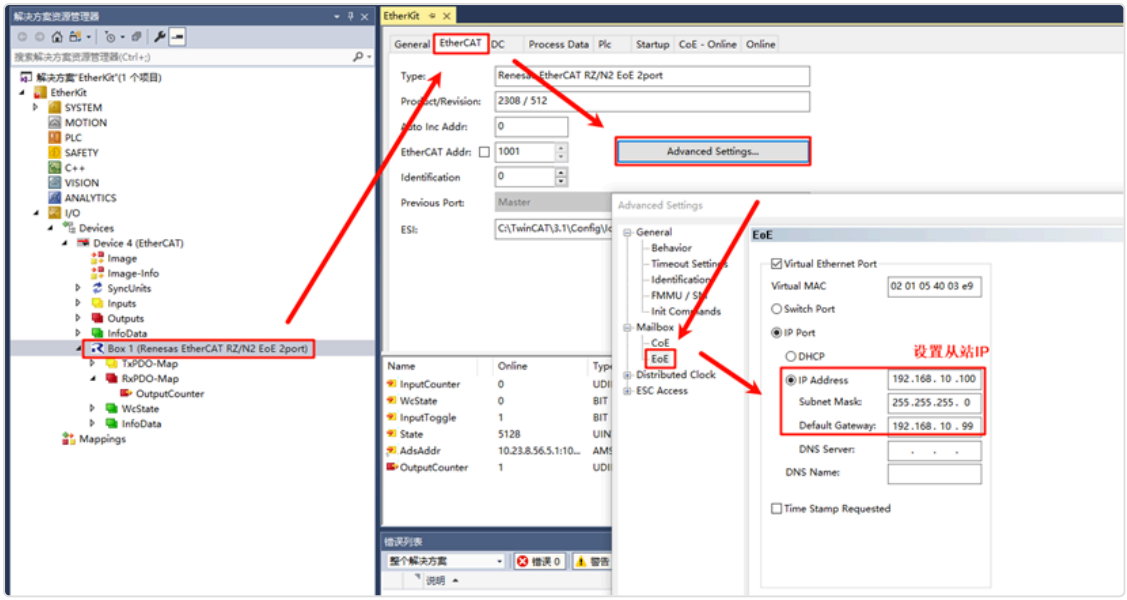
After downloading, right-click Device x (EtherCAT), remove the device, re-scan, add the device again, and activate it (as described above).



EtherCAT EOE Communication

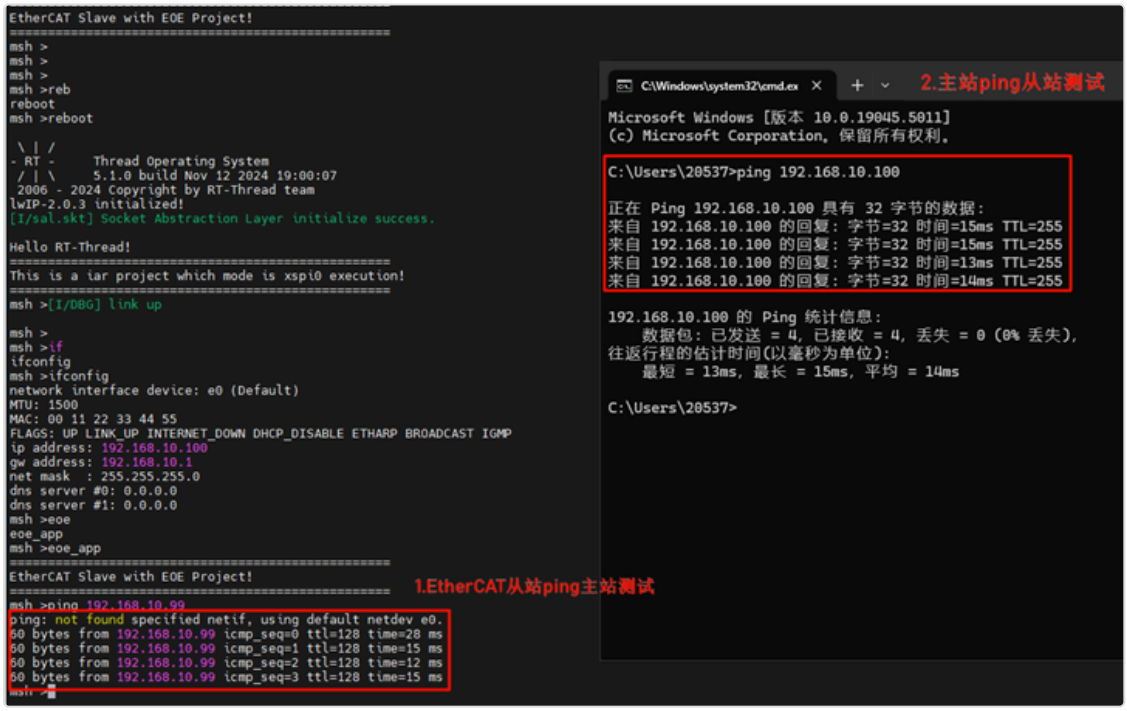
After downloading the EEPROM ESI firmware and re-scanning and adding the device, we can activate the device. We will observe two green LEDs on the board (indicating normal communication). One LED flashes at a high frequency, while the other stays steady. At

this point, double-click the EOE tab on the TwinCAT device page to access the EOE communication configuration.



Enable DHCP or Assign an IP Address

At this point, you can use DHCP to configure an automatic IP or manually assign a static IP address to the EtherCAT slave node.



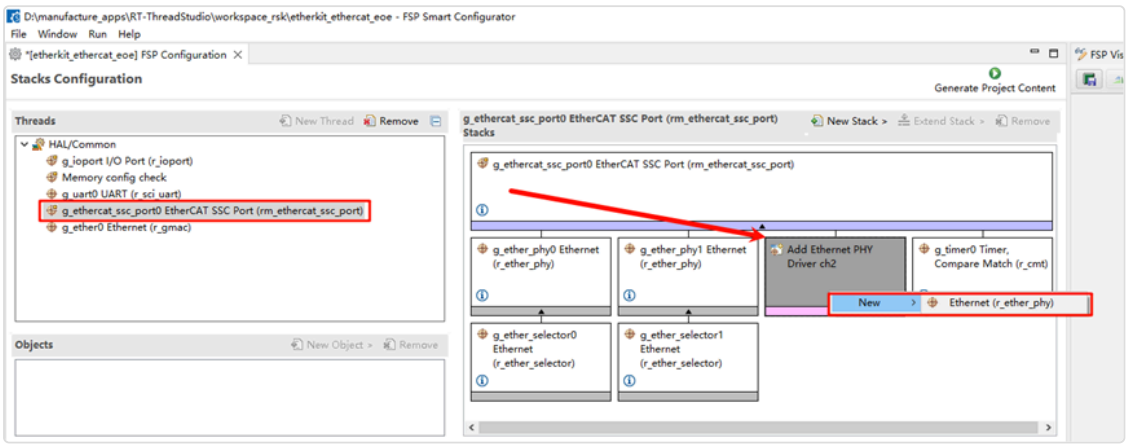
Once configured, the EOE App will work, allowing communication over EtherCAT!

Extension Explanation: 3-Port Ethernet EOE Communication

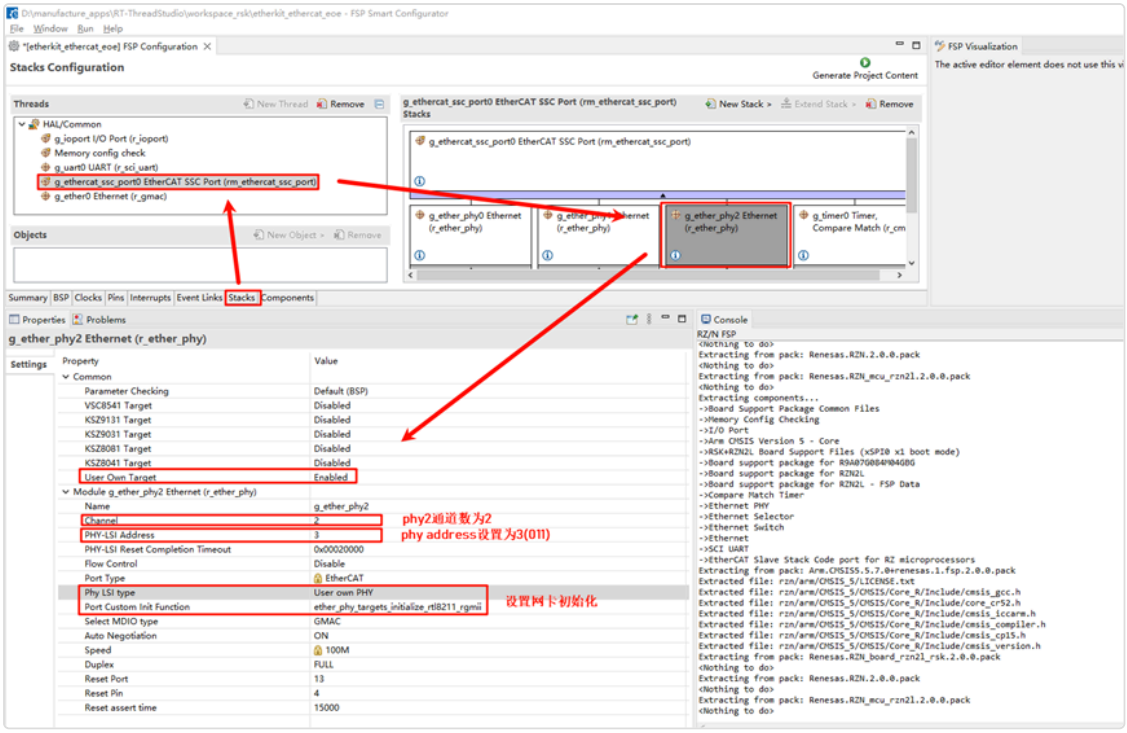
The example project currently defaults to a 2-port Ethernet EOE configuration. If you need to use a 3-port EOE communication setup, please follow the instructions in this chapter for configuration.

FSP Configuration

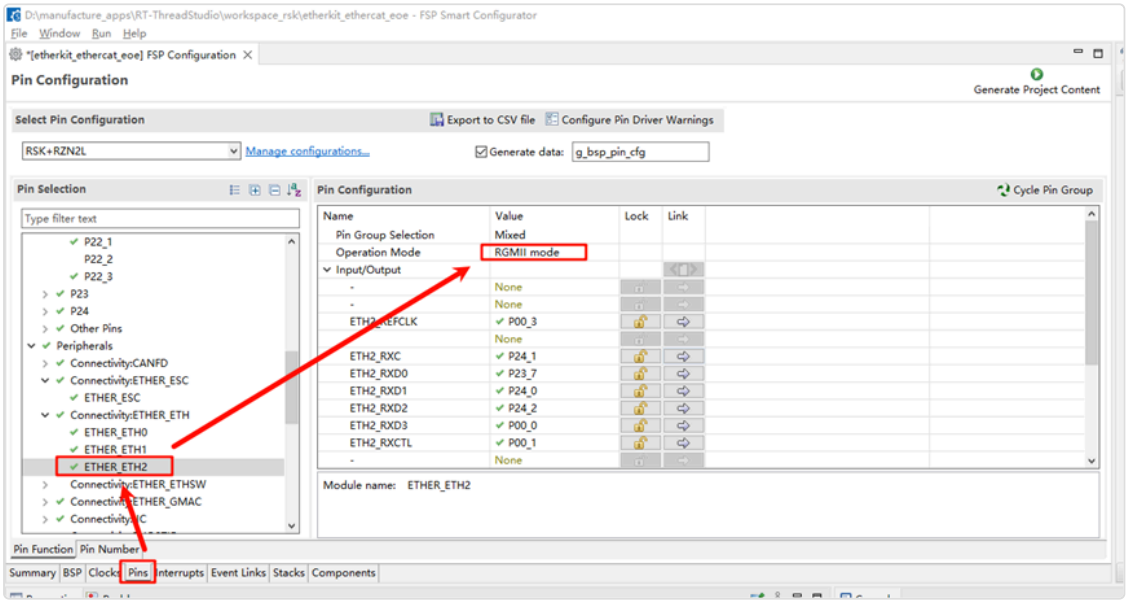
First, open the FSP configuration file in the project. We will add a third PHY for the SSC stack.



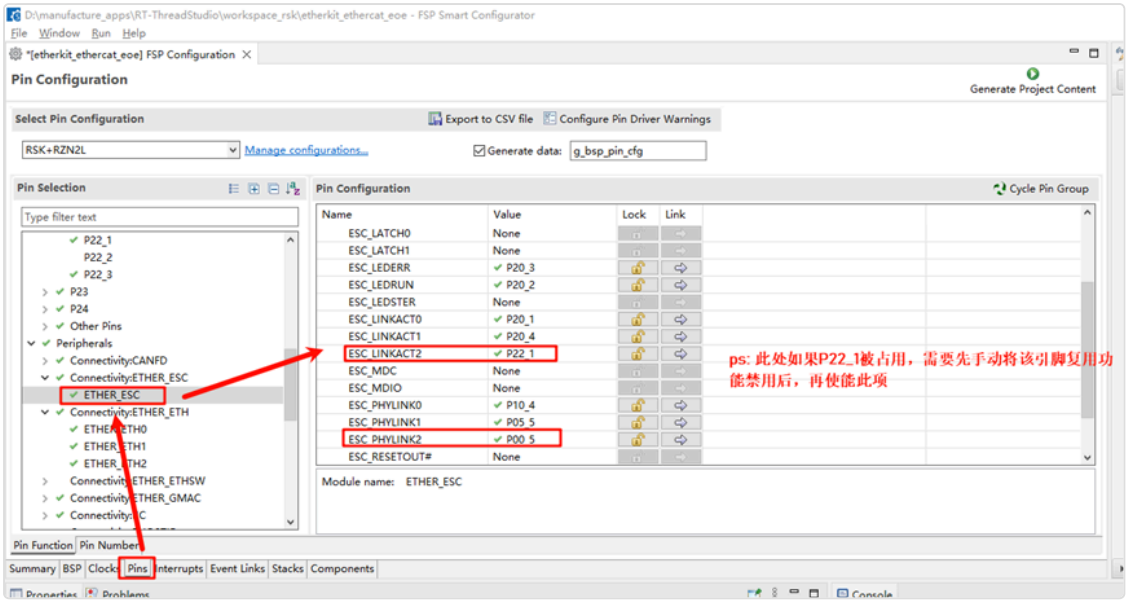
Next, configure the channel count for PHY2 to 2, and set the PHY address to 3 (as referenced from the schematic manual). Also, configure the network card model as user-defined and set the Ethernet initialization callback function.



Then, configure the pins to enable ETH2.



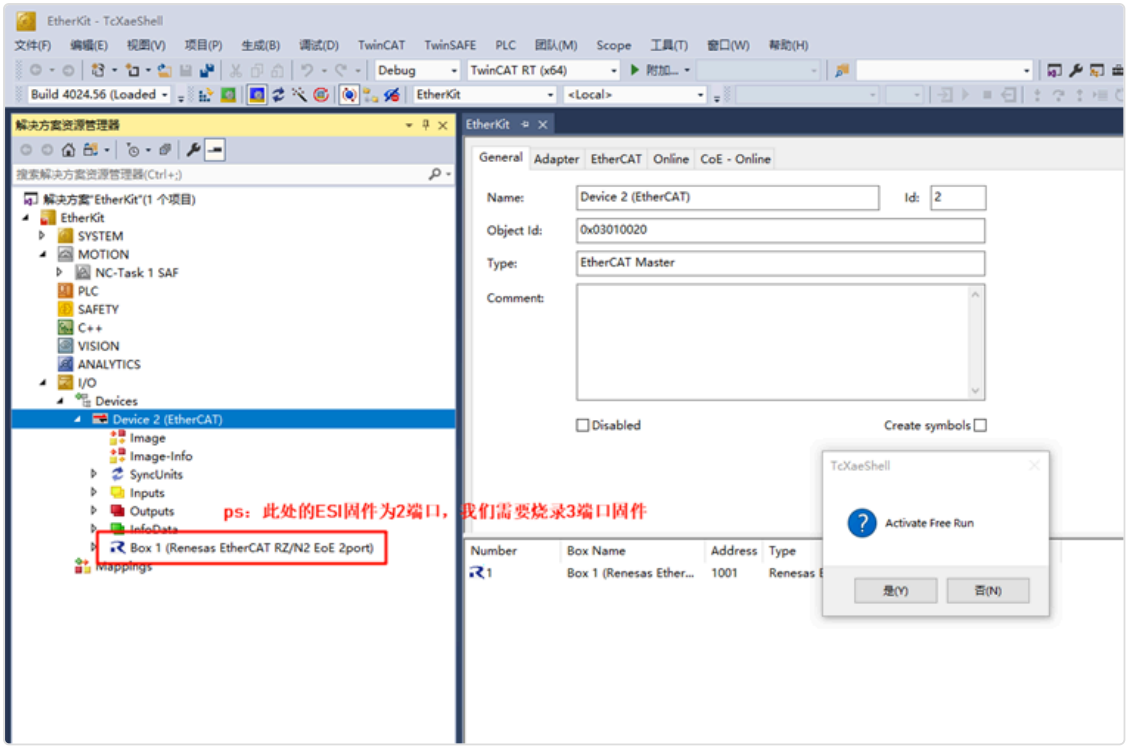
Next, configure the ESC corresponding to the ETH2 LINK pins, setting ESC_LINKACT2 (P22_1) and ESC_PHYLINK2 (P00_5). Note: **If P22_1 is already in use, you must first manually disable its multiplexing function before enabling this option**.



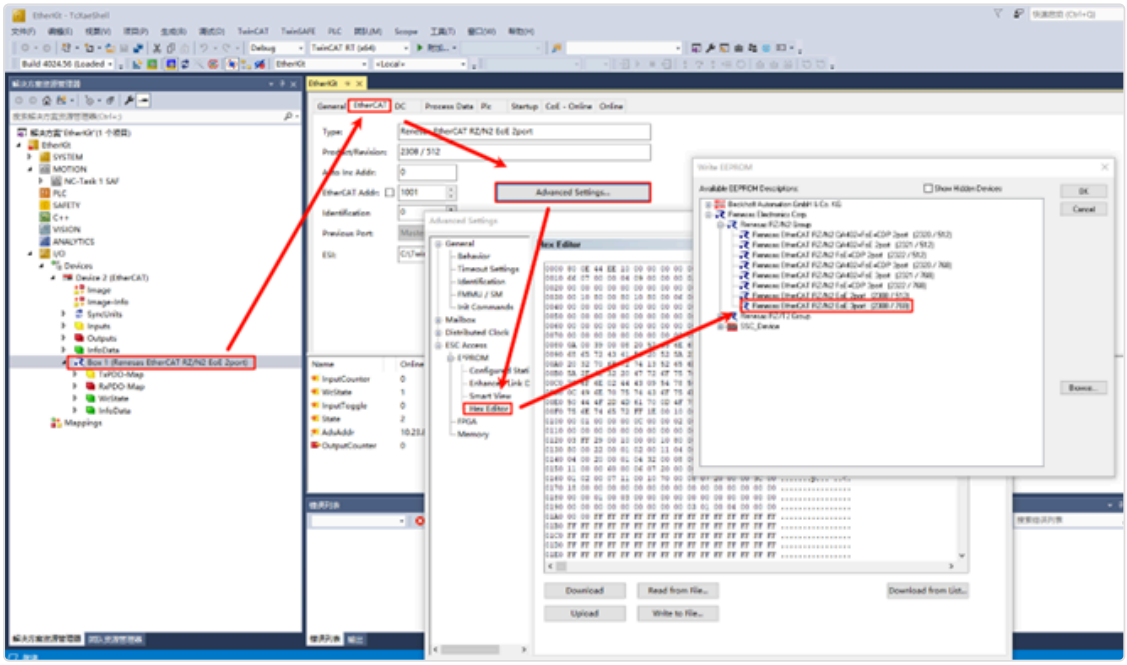
After completing the above configurations, you can click to generate the source code, return to the project, compile it, and download the program to the development board.

ESI Firmware Update

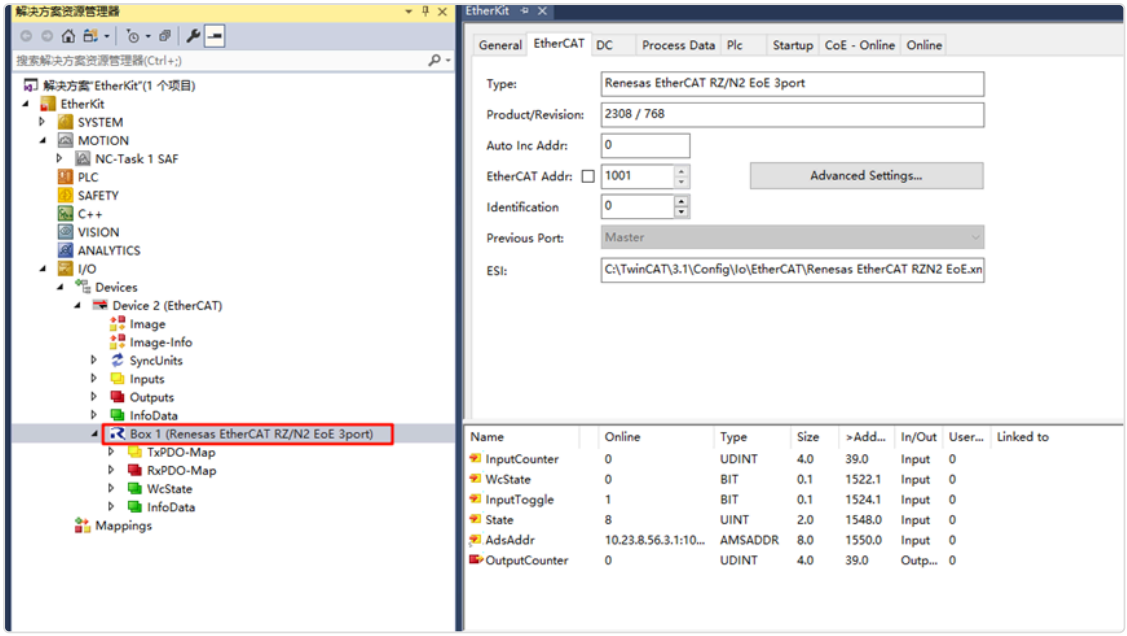
Similarly, we need to wait for the development board's EOE slave to successfully run. Then, open the TwinCAT 3 software to scan for devices. Once the EtherCAT device is found, do not activate it immediately. In the pop-up window, click "No."



Refer to the **"Update EEPROM Firmware"** section. Follow the same steps, but this time, select the firmware to be updated as: Renesas EtherCAT RZ/N2 EoE 3port [2308 / 768], and click to flash the firmware.



Once the flashing is complete, we need to delete the device again and scan it once more. You should see that the slave device description has been updated to "Box 1 (Renesas EtherCAT RZ/N2 EoE 3port)."



For further EOE development, please refer to the previous chapters.

5.3. Ethernet/IP Usage Instructions

English | **中文**

Introduction

Ethernet/IP (Ethernet Industrial Protocol) is an industrial communication protocol based on standard Ethernet architecture, widely used in automation and control systems. It combines the TCP/IP protocol and the CIP (Common Industrial Protocol) standard to provide high-speed, reliable data transmission and supports real-time communication between various industrial devices. Since Ethernet/IP is compatible with existing Ethernet hardware and networks, enterprises can achieve interconnectivity between industrial devices without the need for specialized hardware, thereby enhancing production efficiency and system reliability.

OpENer is the EtherNet/IP™ stack for I/O adapter devices, supporting multiple I/O and explicit connections. It includes objects and services for creating EtherNet/IP™ compatible products as defined in the Ethernet/IP specification and published by ODVA.

This example will use the adapted OpENer package to implement Ethernet/IP communication.

Prerequisites

Software Environment:

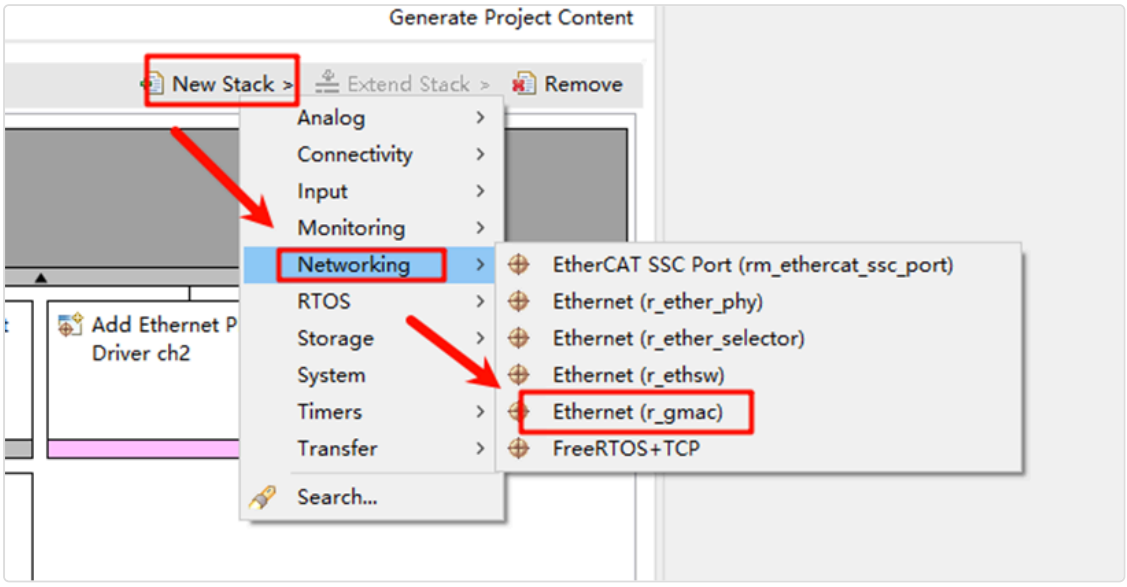
- CODESYS (Ethernet/IP Communication Simulation)
- CODESYS
- CODESYS Gateway (Gateway Device)
- CODESYS Control Win SysTray (Soft PLC Device)
- Npcap (This software is necessary to run CODESYS and must be installed beforehand.)

Hardware Environment:

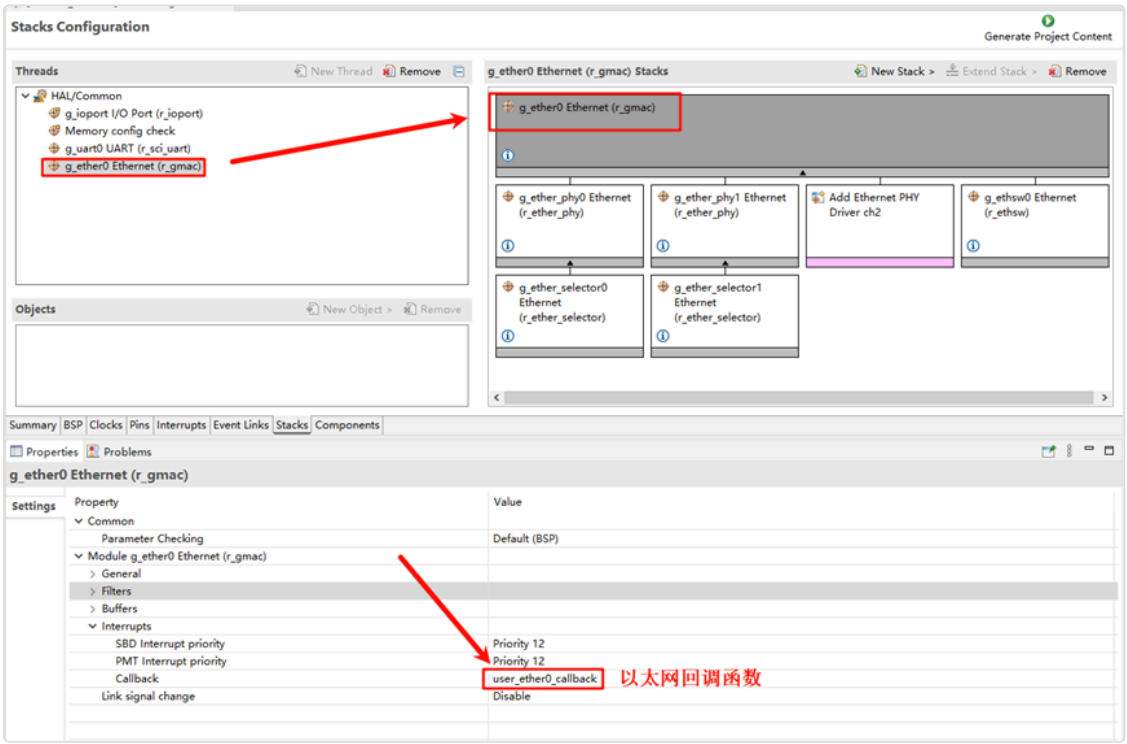
- EtherKit Development Board

FSP Configuration

Open the project configuration file `configuration.xml` and add the `r_gamc Stack`:

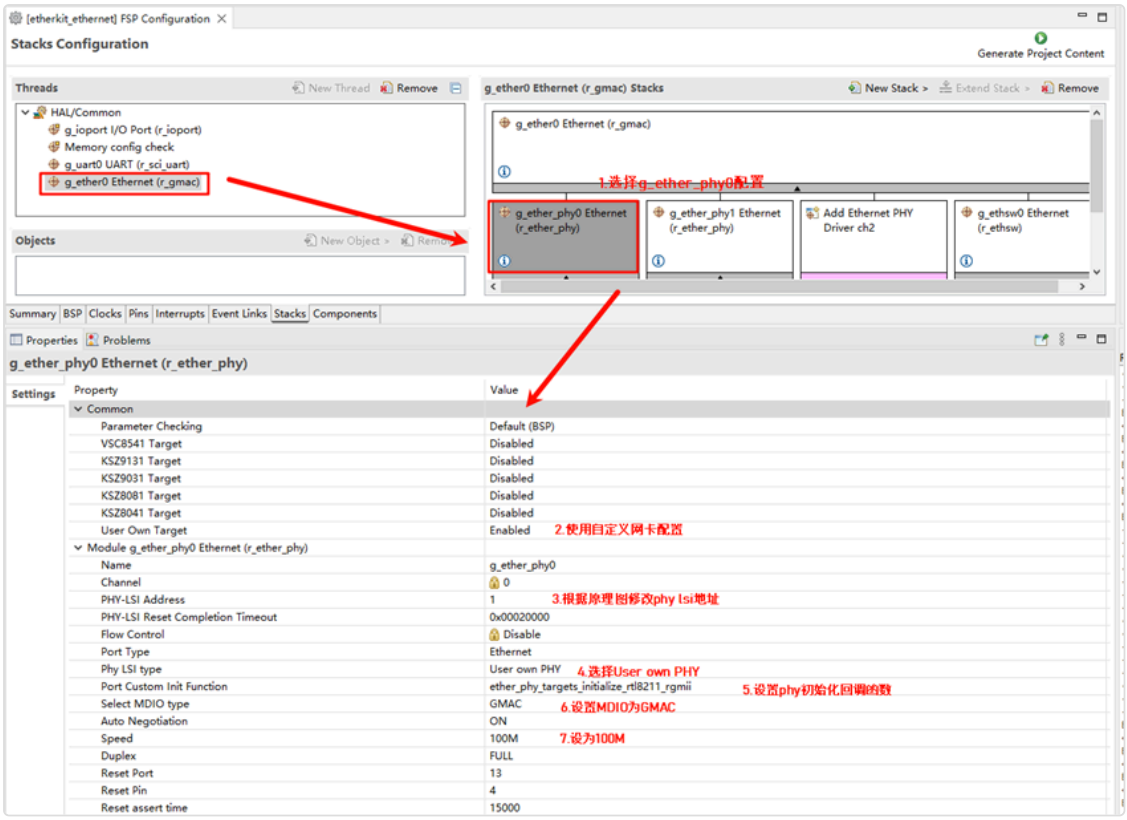


Click `g_ether0 Ethernet`, and set the interrupt callback function to `user_ether0_callback`:

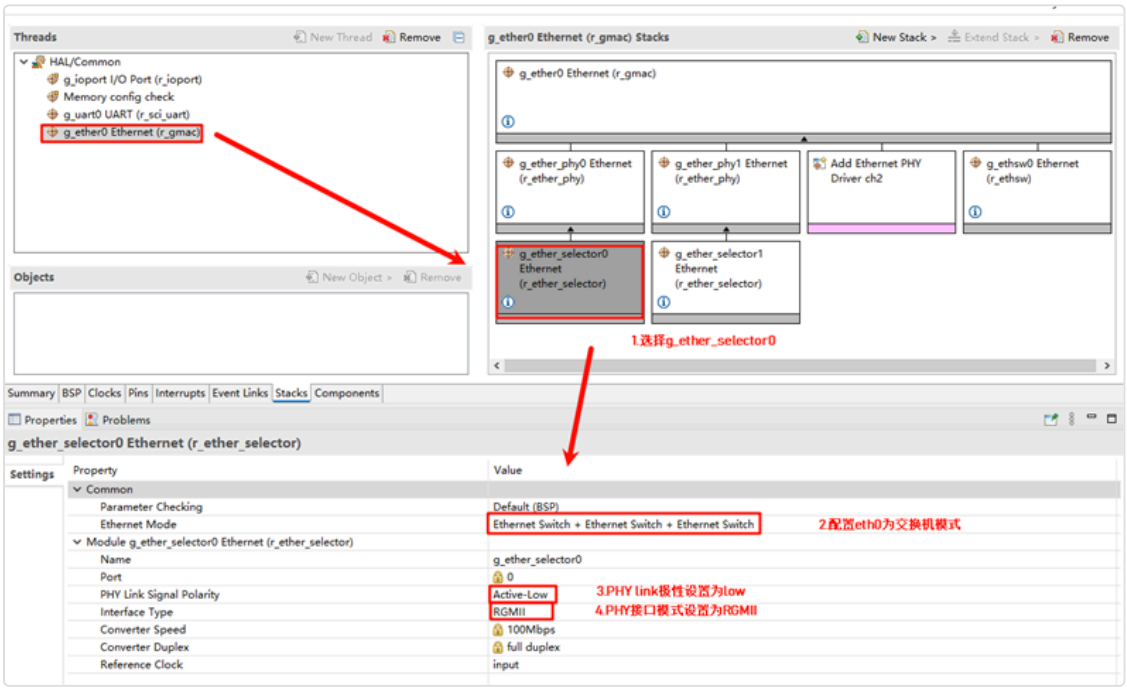


Next, configure the PHY information, select `g_ether_phy0`, set the Common configuration to `User Own Target`; change the PHY LSI address to 1 (refer to the schematic for the specific address); set the PHY initialization callback

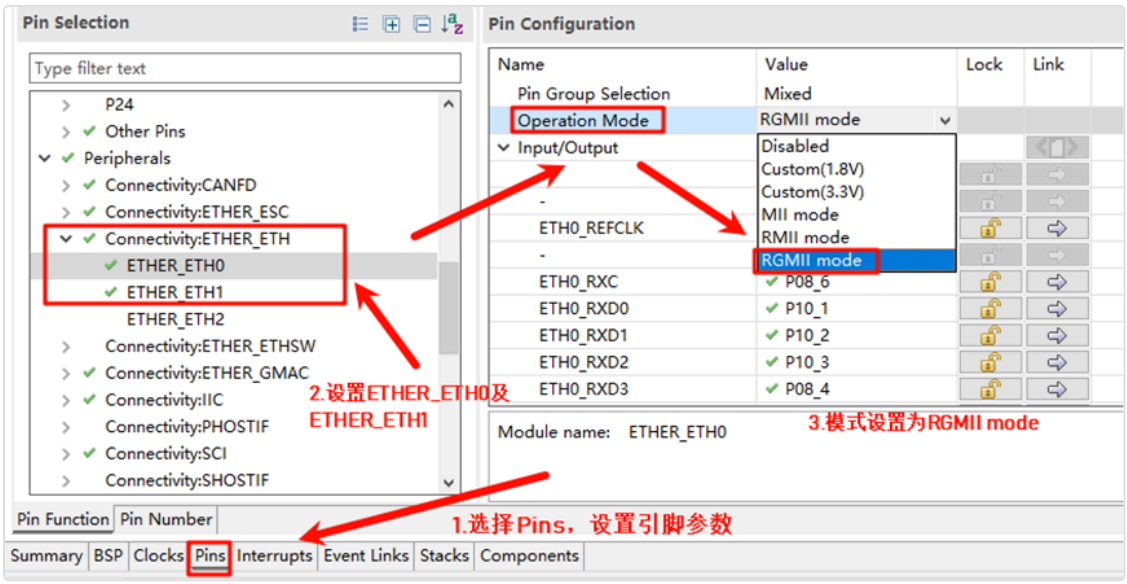
function to `ether_phy_targets_initialize_rtl8211_rgmii()`; and set MDIO to GMAC.



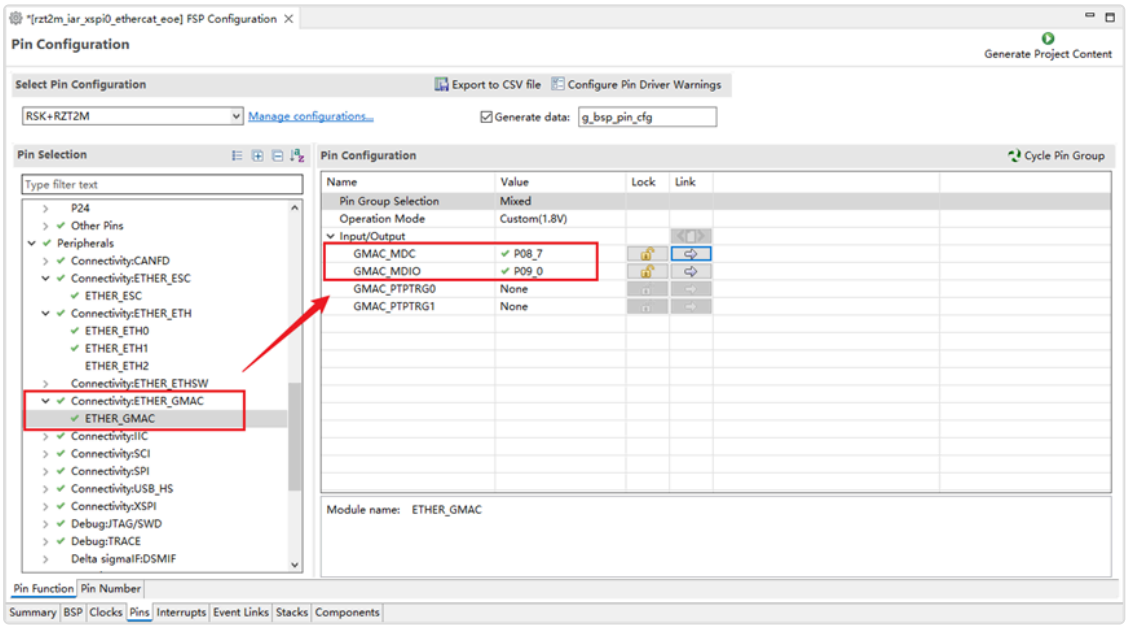
Configure `g_ether_selector0`, set the Ethernet mode to switch mode, set PHY link to active-low, and PHY interface mode to RGMII.



Configure the NIC pin parameters, select the operation mode as RGMII:

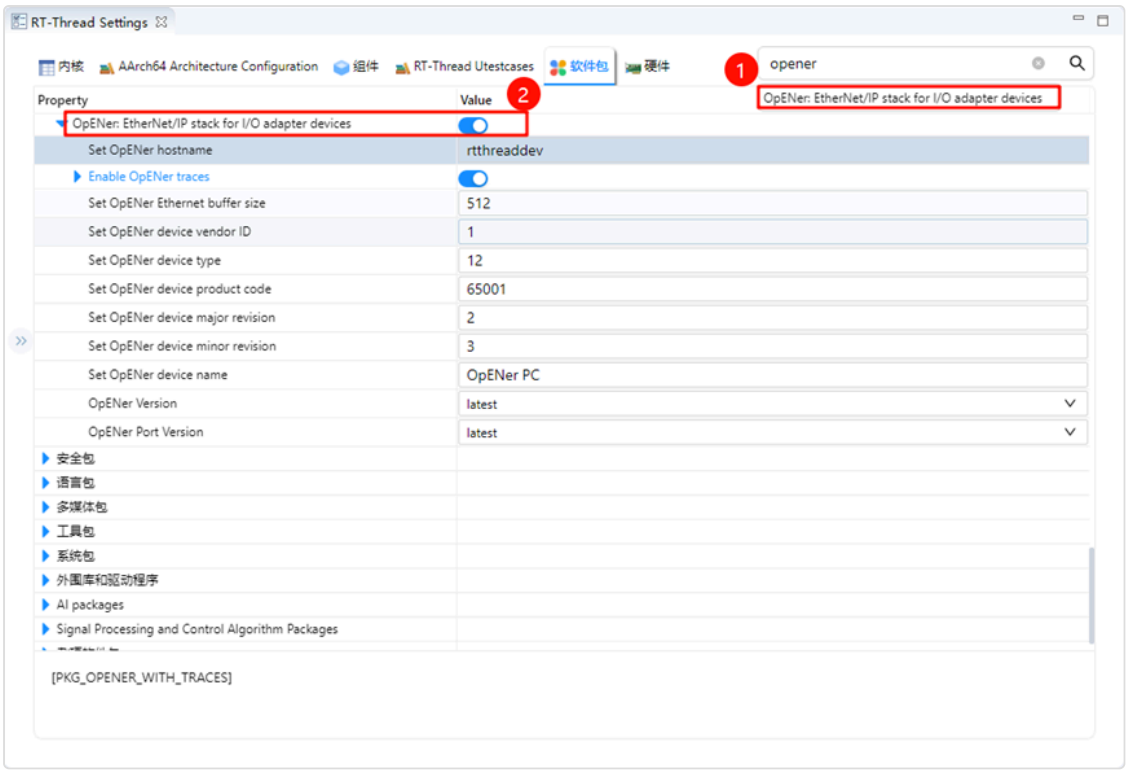


ETHER_GMAC Configuration:

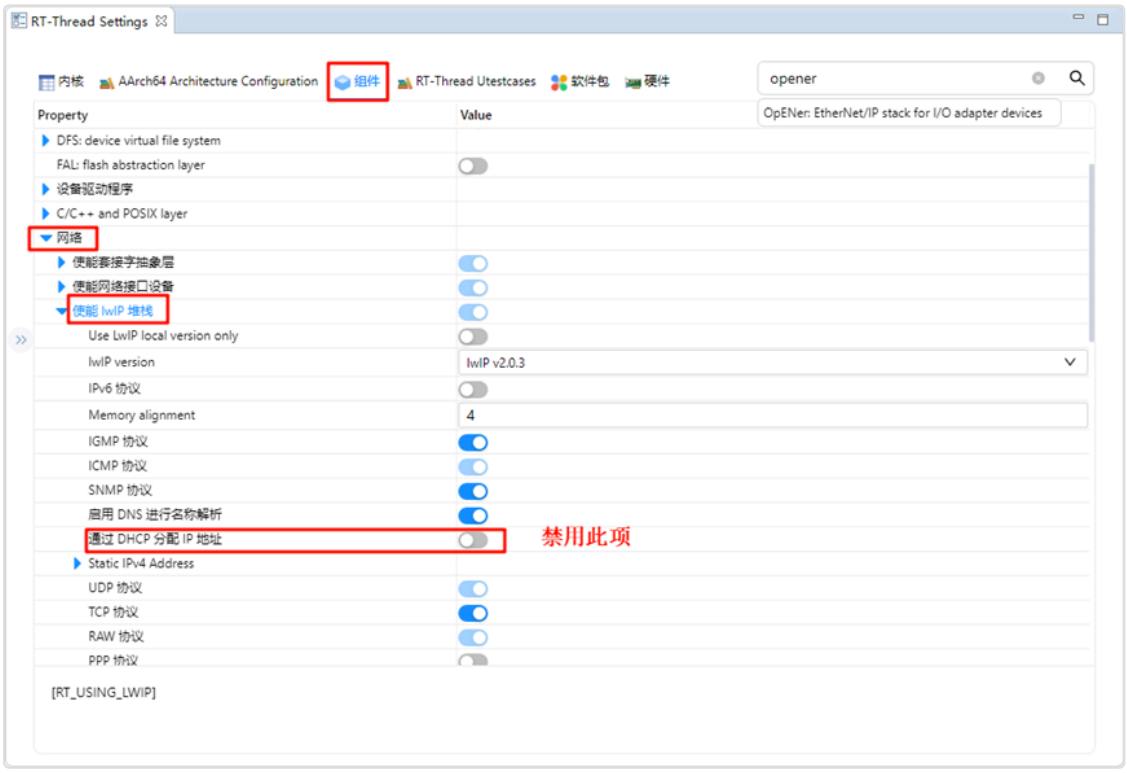


RT-Thread Settings Configuration

Double-click to open RT-Thread Settings, search for the OpENer package, and enable it. The following is the related user configuration information:



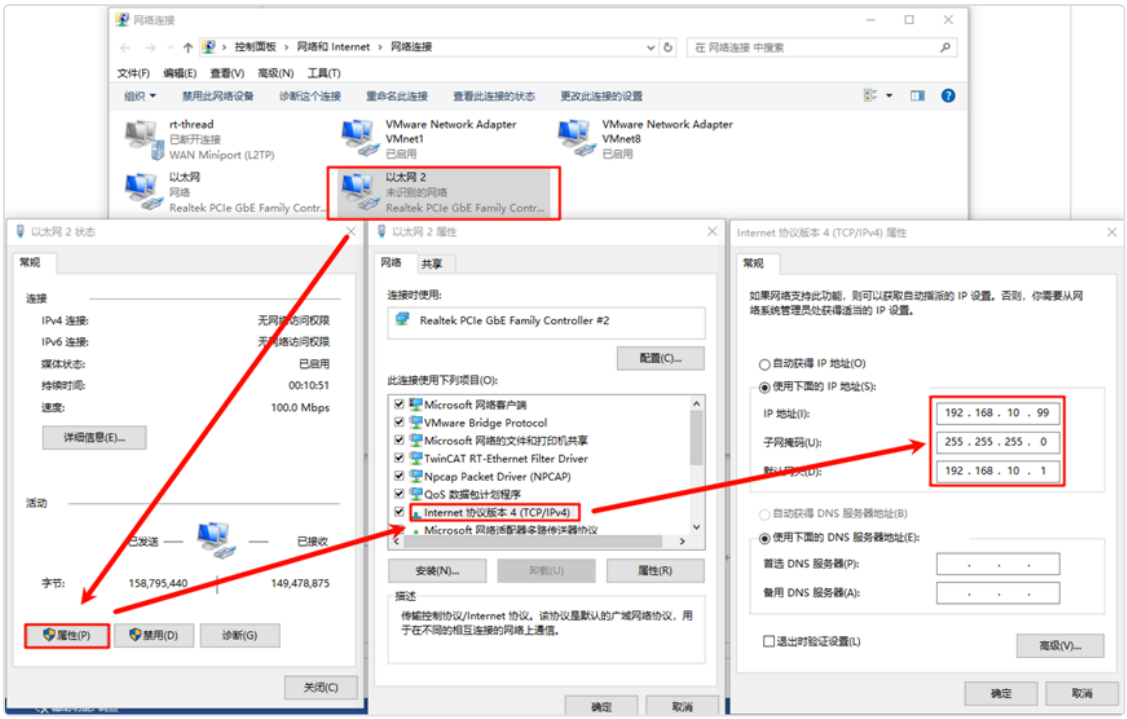
Next, disable DHCP and use a static IP. Go to Components -> Enable lwIP stack, and select disable DHCP:



After completing the configuration, compile the program and download it to the development board.

Network Configuration

We use a network cable to connect the development board to the PC, and set a static IP on the PC:



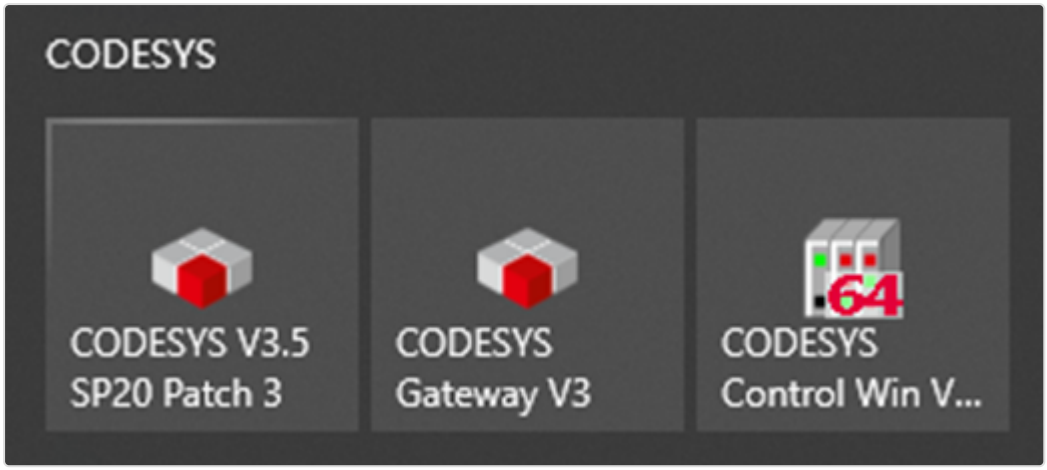
Soft PLC Startup

Introduction to CODESYS:

CODESYS is a PLC software developed by the German company 3S, which integrates PLC logic, motion control, configuration display, and other functionalities. CODESYS, short for "Controller Development System," is an industrial automation programming tool based on the IEC 61131-3 standard. It supports multiple programming languages (such as Ladder Logic, Structured Text, Function Block Diagram, etc.) and provides a rich set of libraries and function blocks, helping engineers rapidly develop and debug PLCs (Programmable Logic Controllers) and industrial control systems. The flexibility and powerful features of CODESYS make it a widely used development platform in the field of industrial automation.

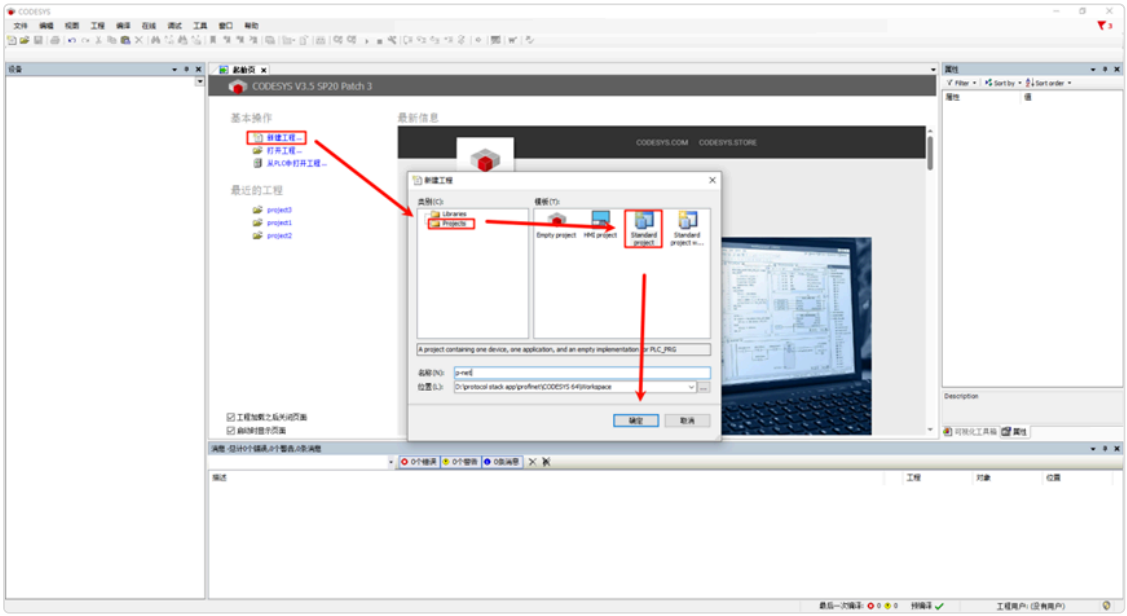
Create a Standard Project in CODESYS

Ensure that CODESYS software is installed. After installation, the following three components are required:

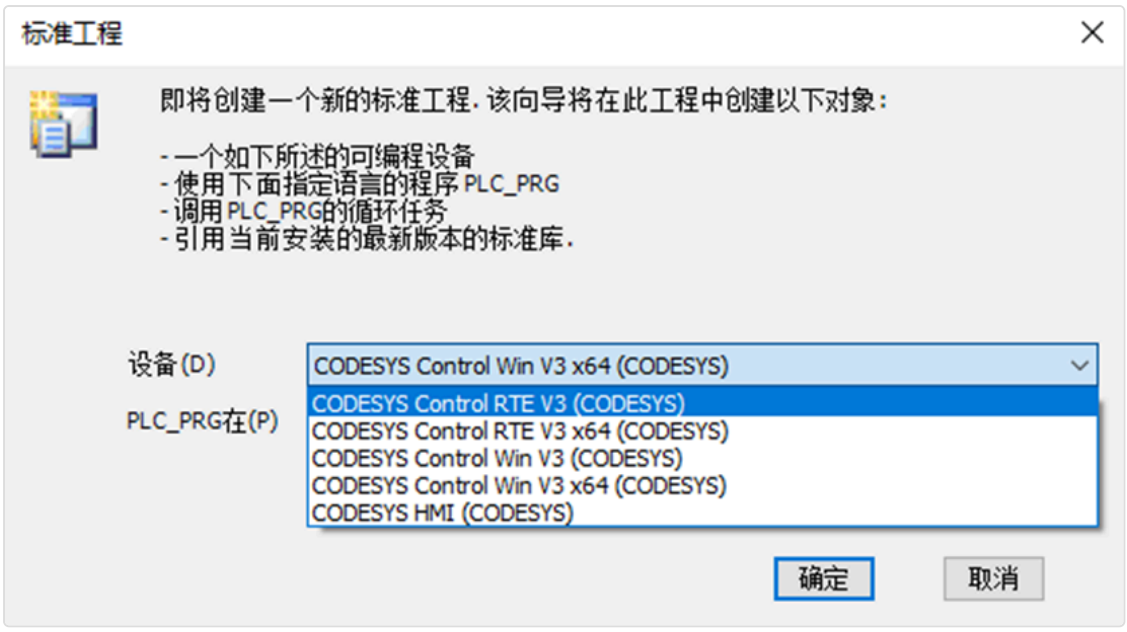


- CODESYS V3.5 SP20 Patch 3: Ethernet/IP Communication Simulation
- CODESYS Gateway V3: Gateway Device
- CODESYS Control Win V3 -x64 SysTray: Soft PLC Device

First, open **CODESYS V3.5 SP20 Patch 3**, then select -> New Project -> Projects -> Standard Project, and configure the project name and location before clicking OK:

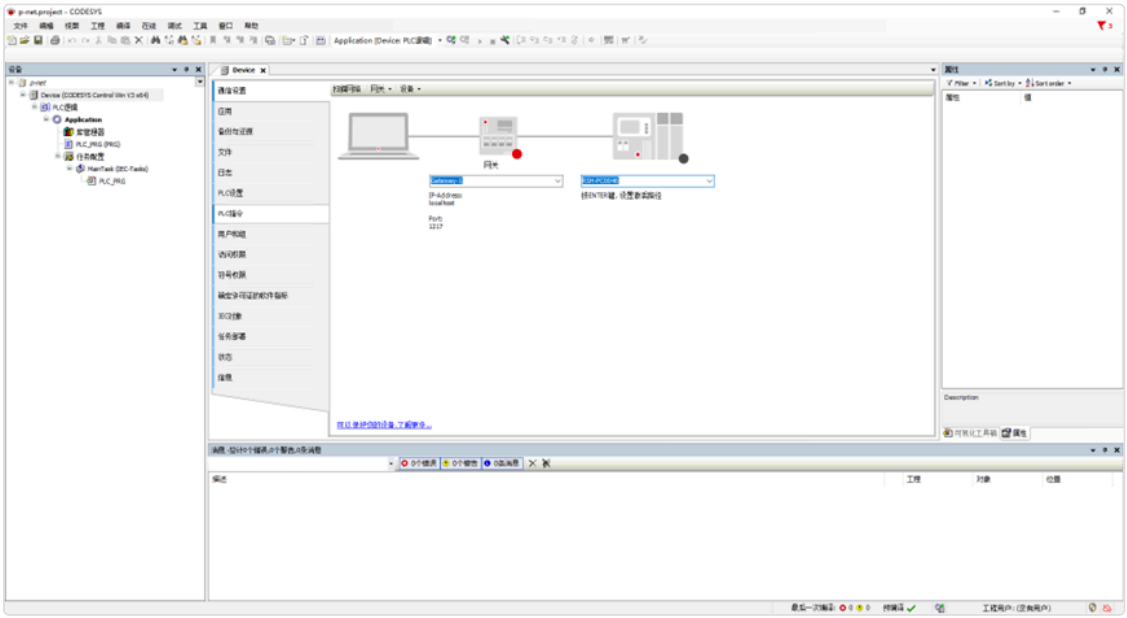


After the pop-up window appears, keep the default configuration (CODESYS Control Win V3 (CODESYS) / x64 (CODESYS)) and click OK:



> *Note: If you purchased **CODESYS Control RTE SL** (http://store.codesys.cn/codesys/store/detail.html?productId=58), you can select the device: CODESYS Control RTE V3 (CODESYS) / x64 (CODESYS). For normal evaluation purposes, you can choose not to install this extension package and create the project using the CODESYS Control Win V3 (CODESYS) / x64 (CODESYS) device.*

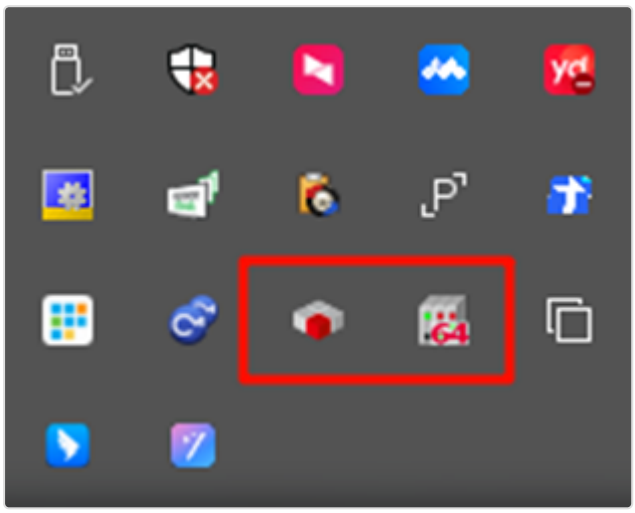
Once the project is created, the main interface will be displayed:



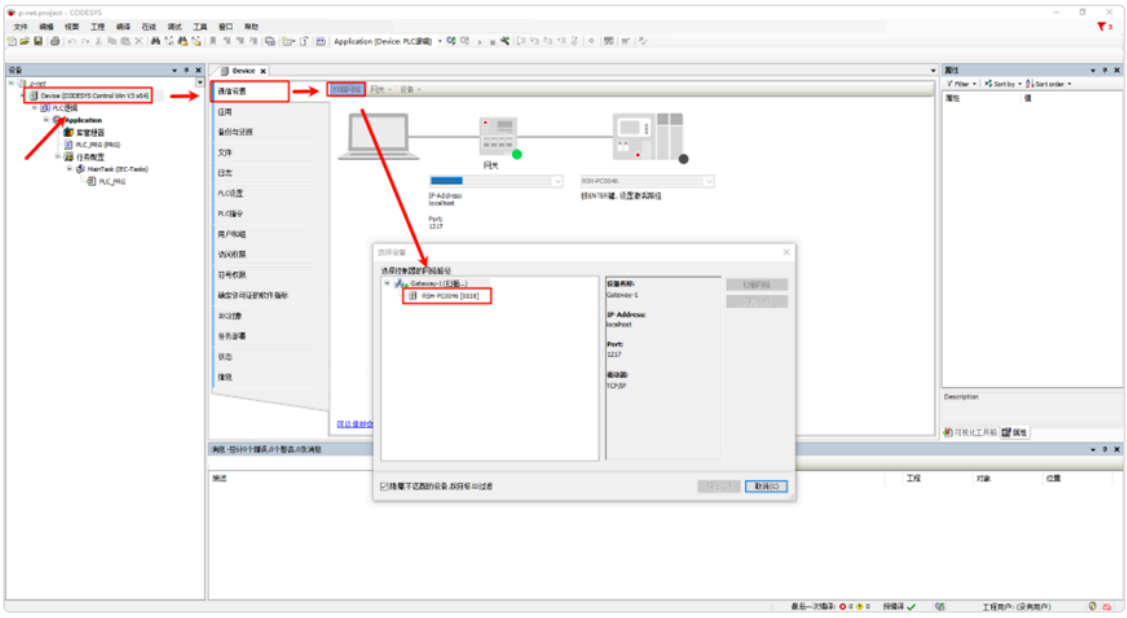
Gateway and Soft PLC Startup

Open the following two software programs:

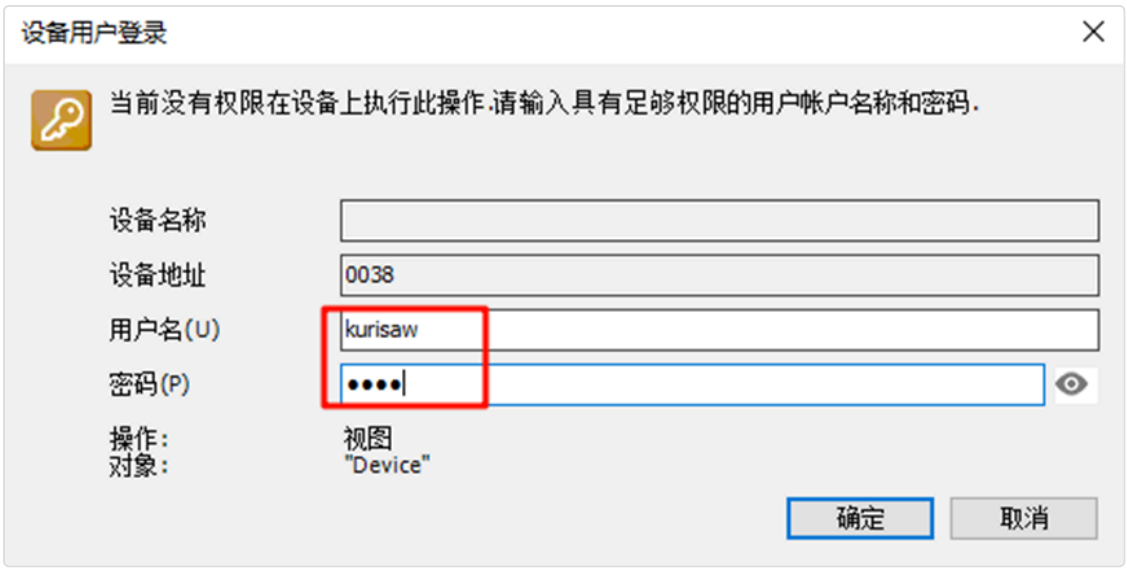
- CODESYS Gateway V3 (Right-click Start Gateway)
- CODESYS Control Win V3 -x64 SysTray (Right-click Start PLC)



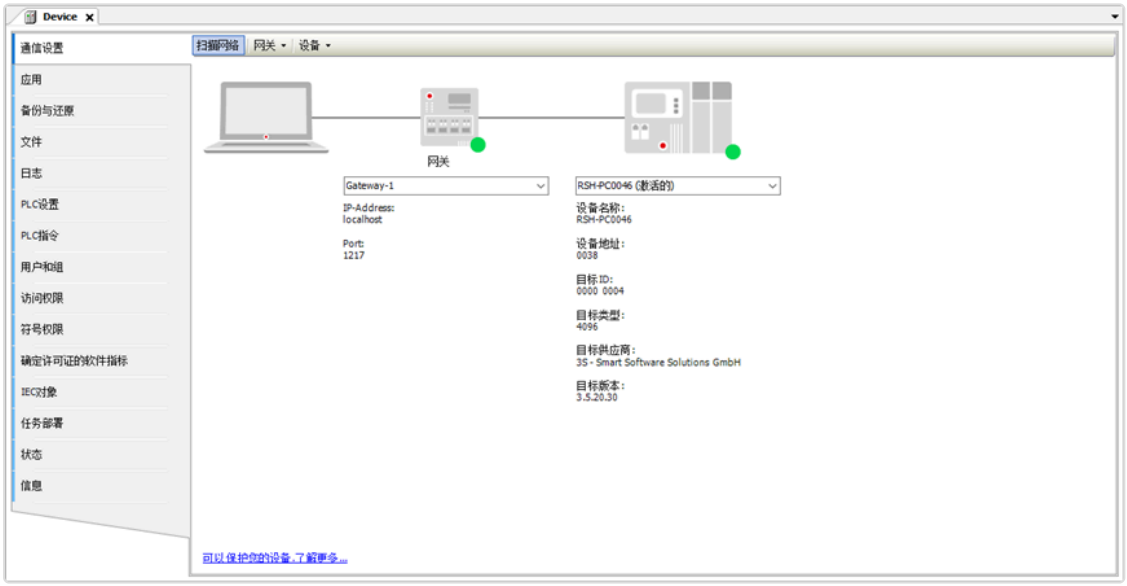
Return to the CODESYS main software, double-click **Device (CODESYS Control Win V3 x64)** -> **Communication Settings** -> **Scan Network**:



A device user login window will pop up. Enter the username and password (customized by the user):



Check whether the gateway device and soft PLC device are online:



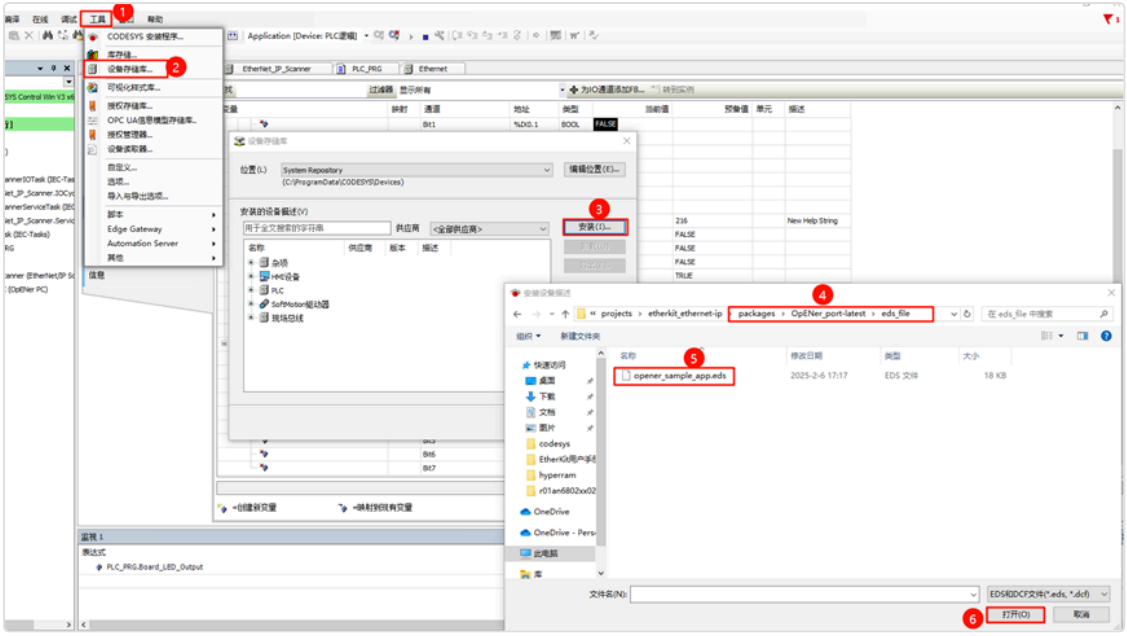
Add Ethernet/IP EDS File

An **EDS file** (Electronic Data Sheet) is a standard file format in Ethernet/IP used to describe device characteristics and communication parameters. It contains detailed information about the device, including its type, supported services, input/output definitions, parameter settings, device status, and configuration options.

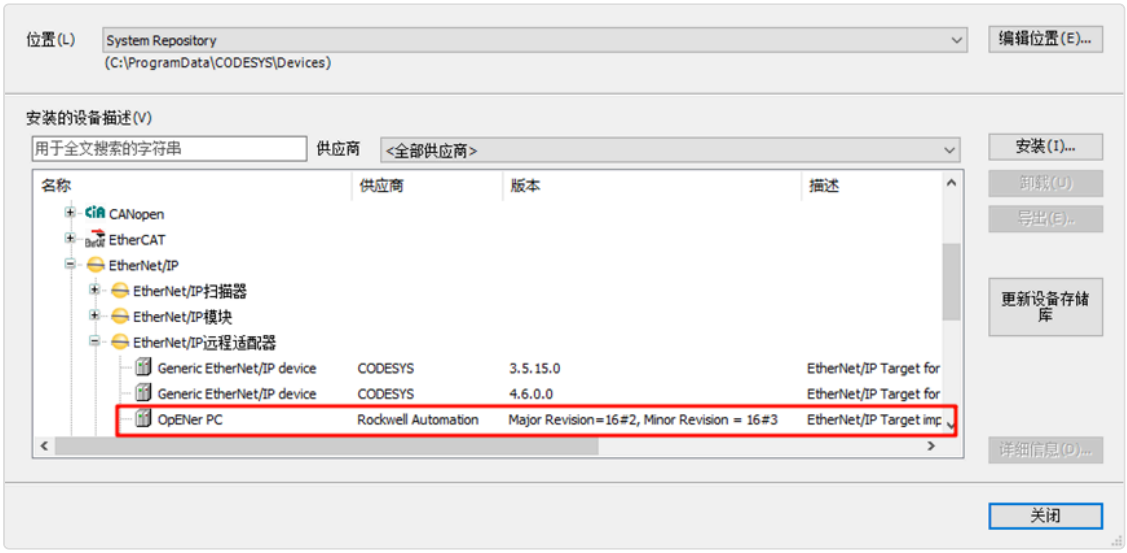
The EDS file for this project is located at:

- `..\packages\OpENer_port-latest\eds_file`

Select the device repository installation description file and choose the **opener_sample_app.eds** file from the above path.

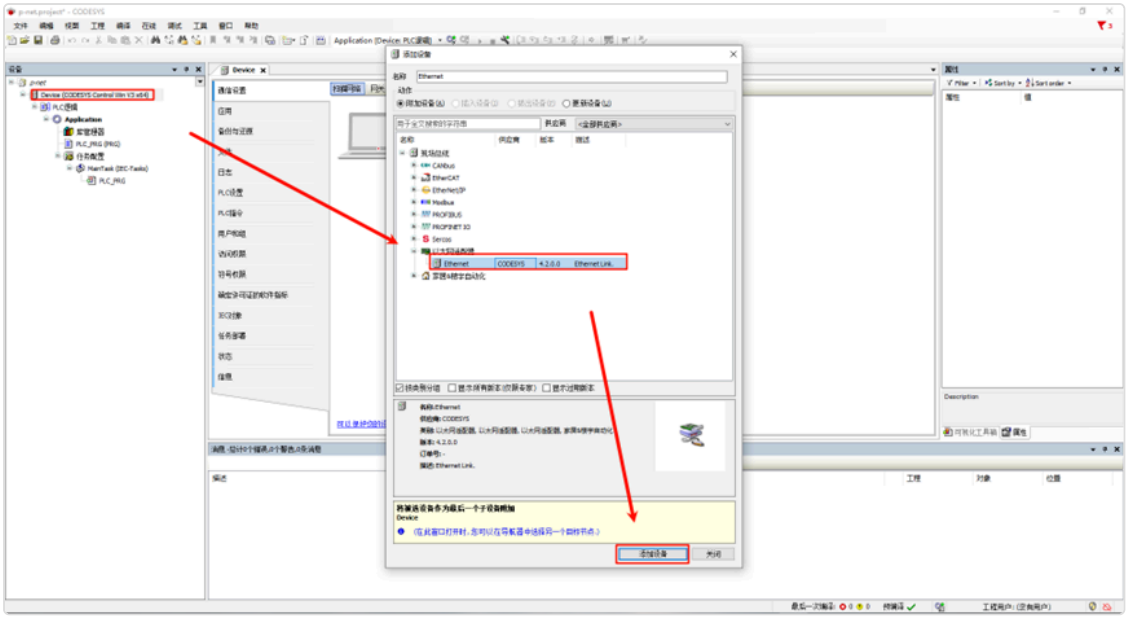


After successful installation, you can see the OpENer PC slave description file:

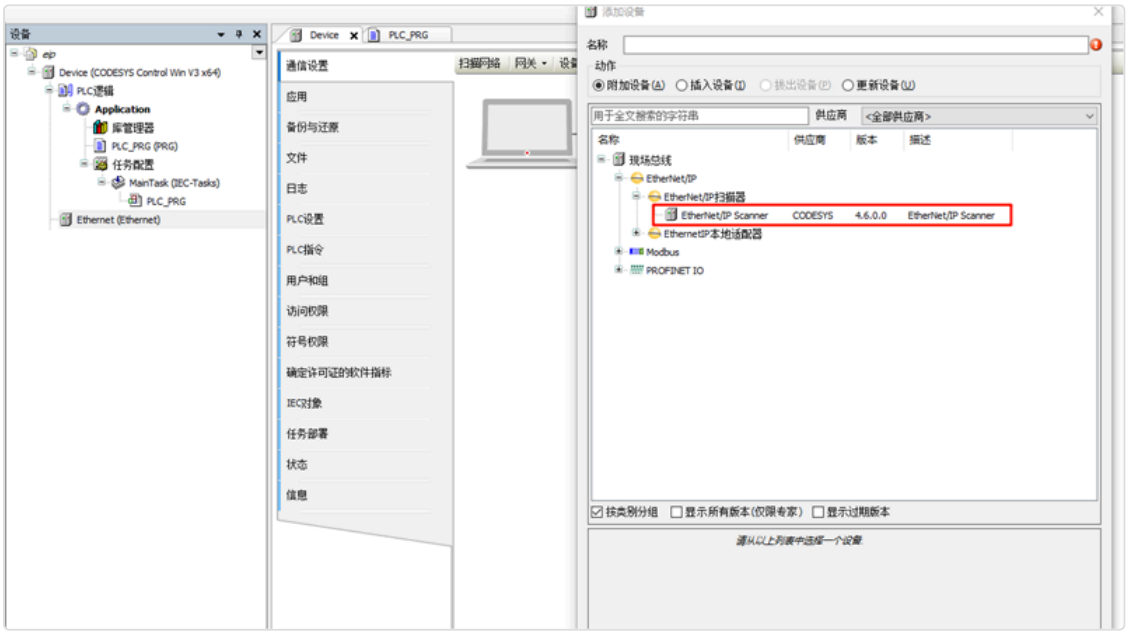


Add Devices

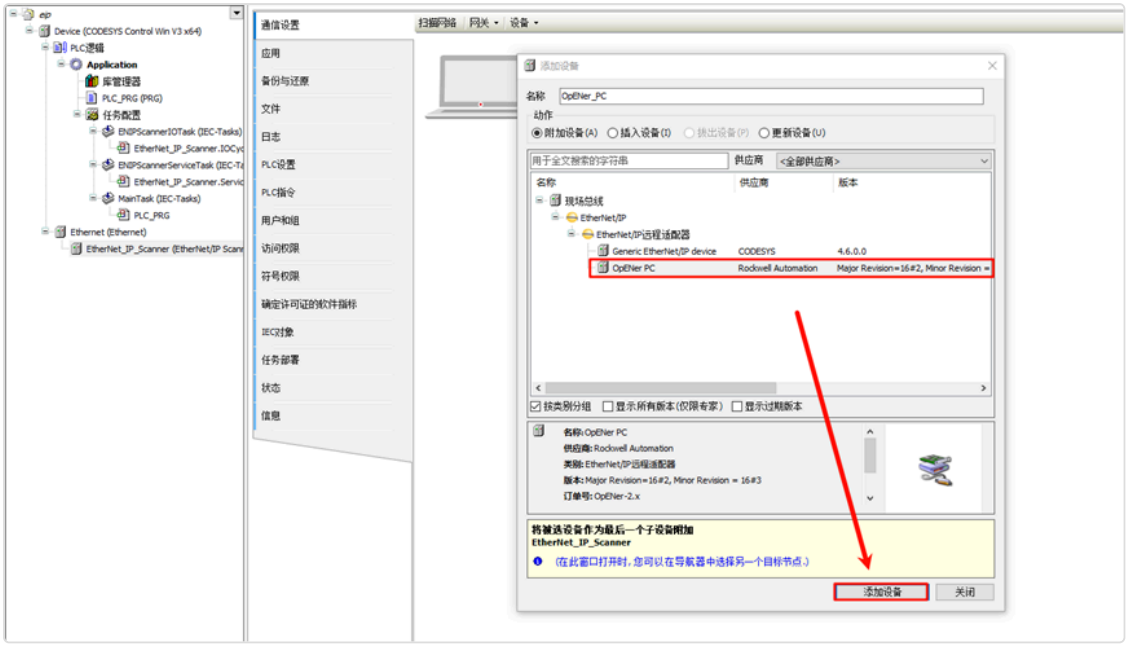
- Add Ethernet: In the left navigation pane, right-click **Device** and select **Add Device**, then choose **Ethernet Adapter**:



- Add EtherNet/IP Scanner: Right-click **Ethernet** in the left navigation pane and choose **EtherNet/IP Scanner**:



- Add EtherNet/IP Bus Device: Right-click **EtherNet/IP Scanner** in the left navigation pane and choose **OpENer PC**:

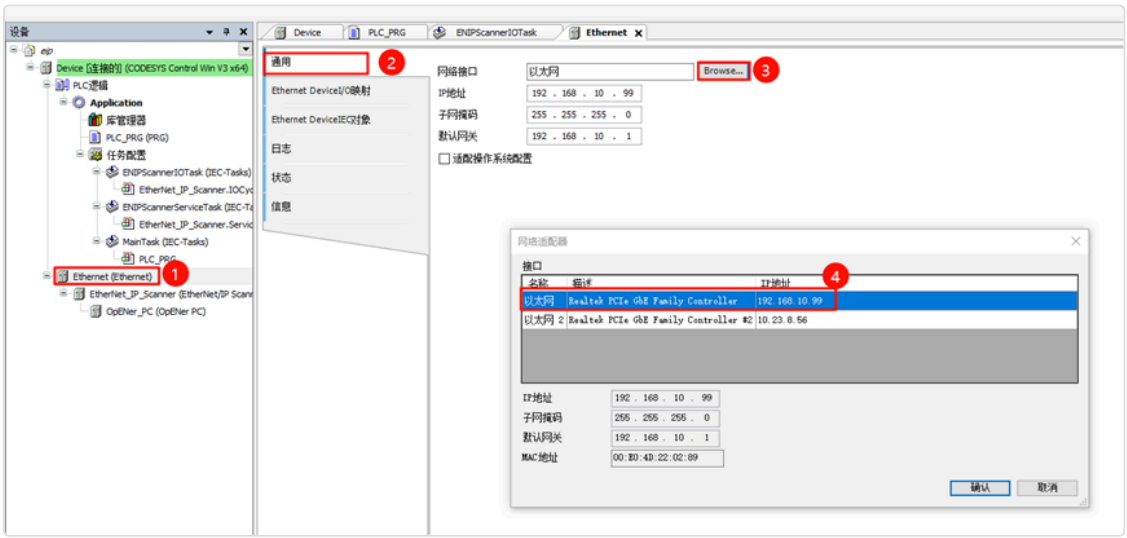


Task Response

Keep the default configuration.

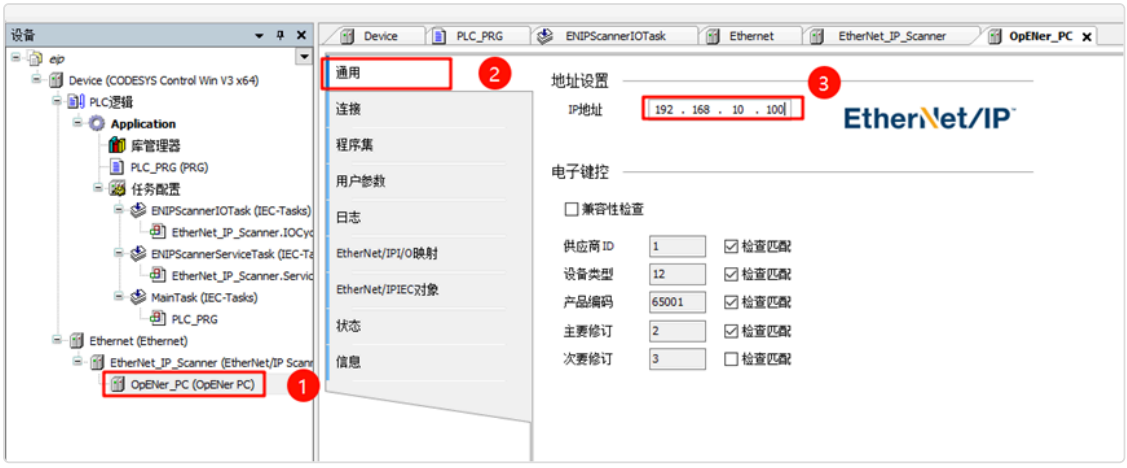
Network Configuration

- Ethernet Configuration: Double-click **Ethernet (Ethernet)** in the left navigation pane -> **General**, change the network interface to the Ethernet port connected to the development board:



- EtherNet/IP Bus Device Network Configuration: Double-click **OpENer_PC (OpENer PC)** in the left navigation pane -> **General** -> **Address Settings**,

change the IP parameters to the development board's IP address.



EtherNet/IP Thread Application Startup

Once the development board is powered on, it will automatically start the OpENer thread when the network card link is detected:

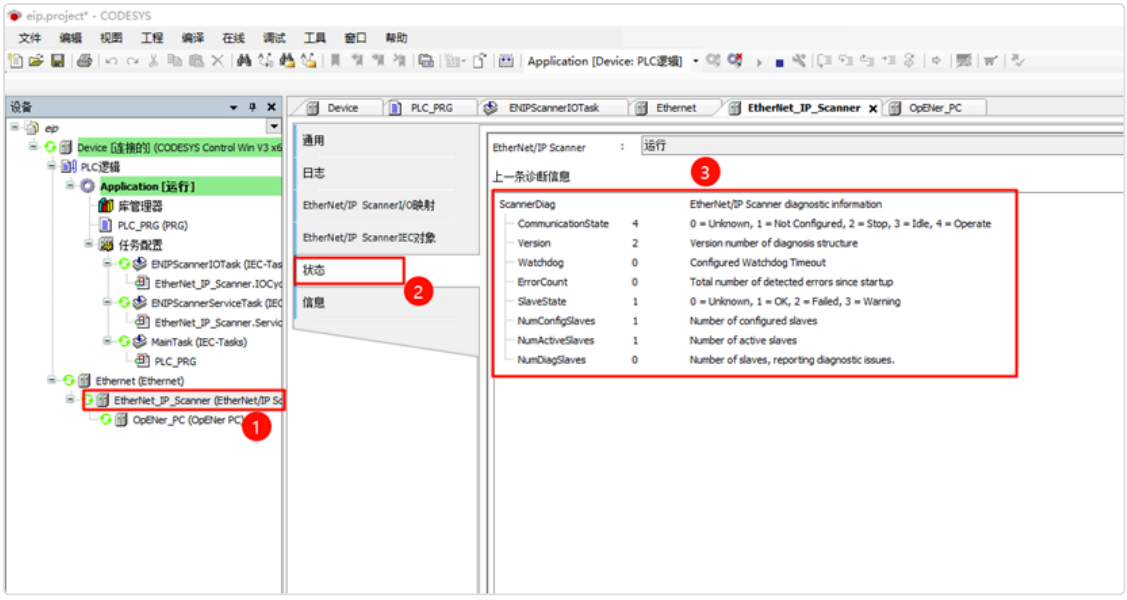
```
msh />
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Feb  7 2025 14:55:26
 2006 - 2024 Copyright by RT-Thread team
lwIP-2.0.3 initialized!
[I/sal.skt] Socket Abstraction Layer initialize success.

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 />
```

Compile and Start Debugging the Project

- Step 1: In the project toolbar, select **Build** -> **Generate Code**
- Step 2: Select **Online** -> **Login**
- Step

3: Click **Debug** -> **Start**

At this point, you can see that the EtherNet/IP Scanner is running properly:



PLC Programming and CIP IO Control

First, click **Device** -> **PLC Logic** -> **Application** -> **PLC_PRG (PRG)** in the left panel to open the PLC program, then use Structured Text (ST) to write variable definitions and program code:

- Variable Definition: The following variables include two key variables: `Board_SW_Input` (representing the controller's onboard switch array in bit-level) and `Board_LED_Output` (representing the controller's onboard LED in bit-level).

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

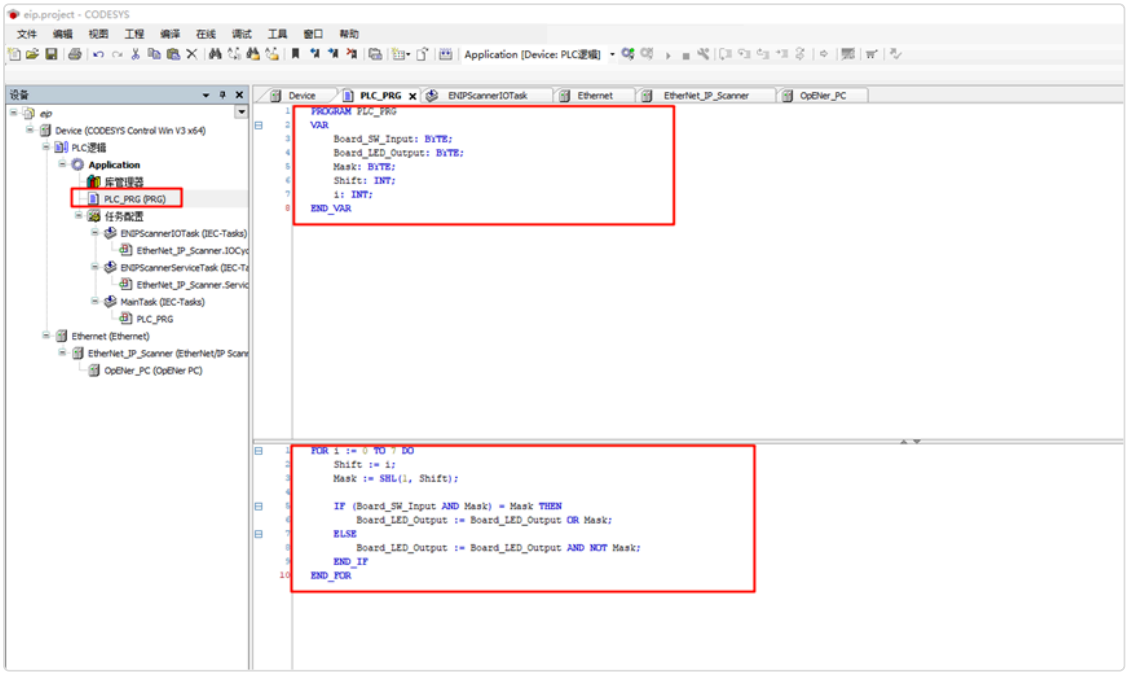
- Program Definition: This code sets the corresponding bits of `Board_LED_Output` based on the state of each bit of `Board_SW_Input`:
- If a bit of `Board_SW_Input` is 1, the corresponding bit of `Board_LED_Output` is set to 1.

- If a bit of `Board_SW_Input` is 0, the corresponding bit of `Board_LED_Output` is set to 0.

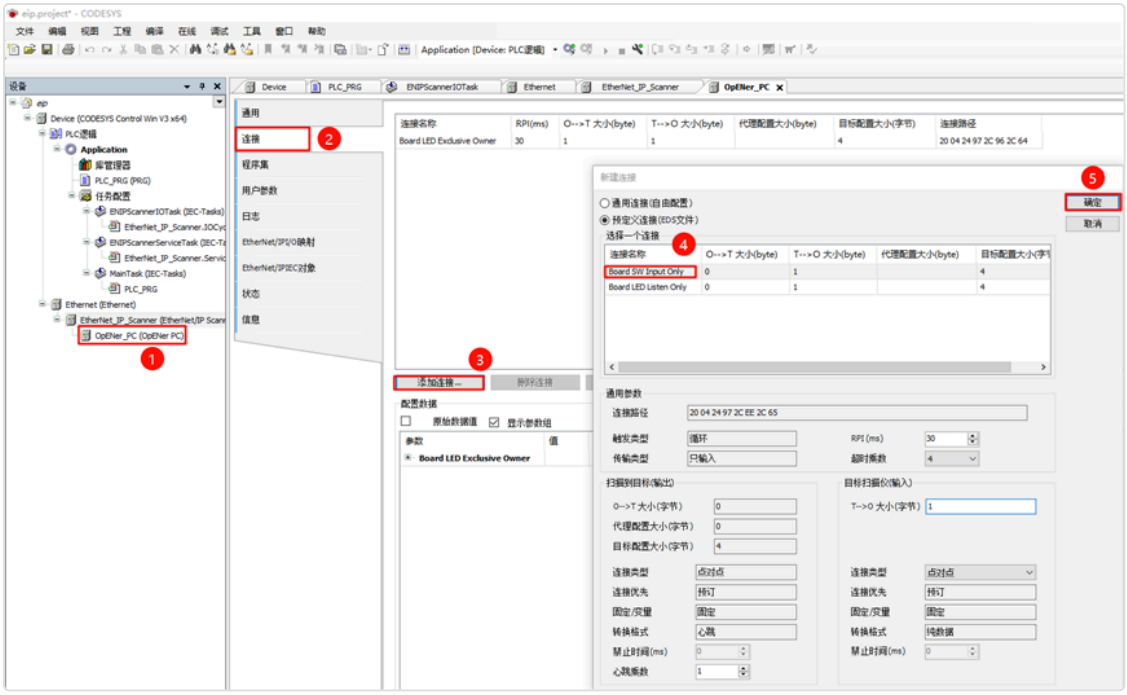
By iterating through all 8 bits, the input states are mapped to the output.

```
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
```

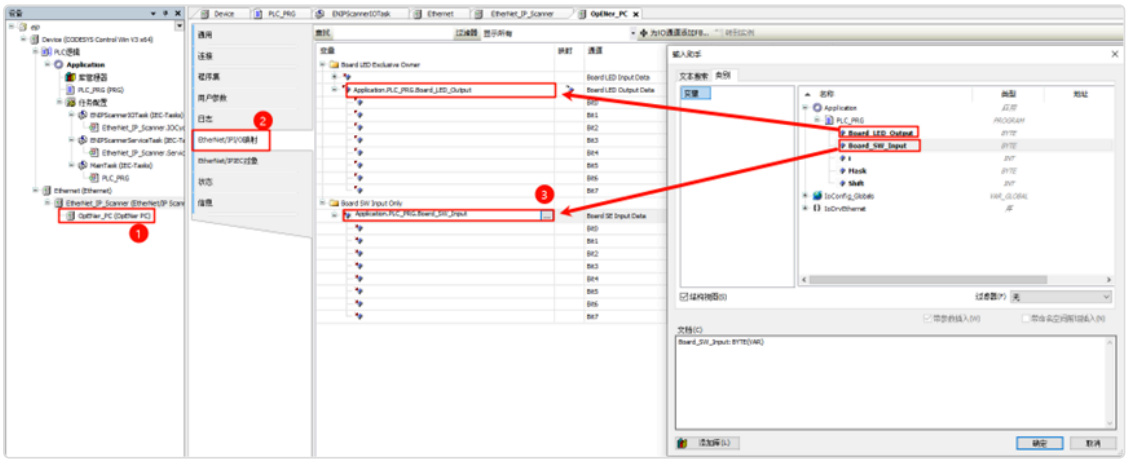
The configuration location in the project is as follows:



After loading the EDS file, only one connection configuration (Board LED Exclusive Owner) is shown by default. We also need to load another configuration embedded in the EDS file. Click **OpENer_PC (OpENer PC)** in the left menu -> **Connections**, click **Add Connection...**, and choose **Board SW Input Only**.



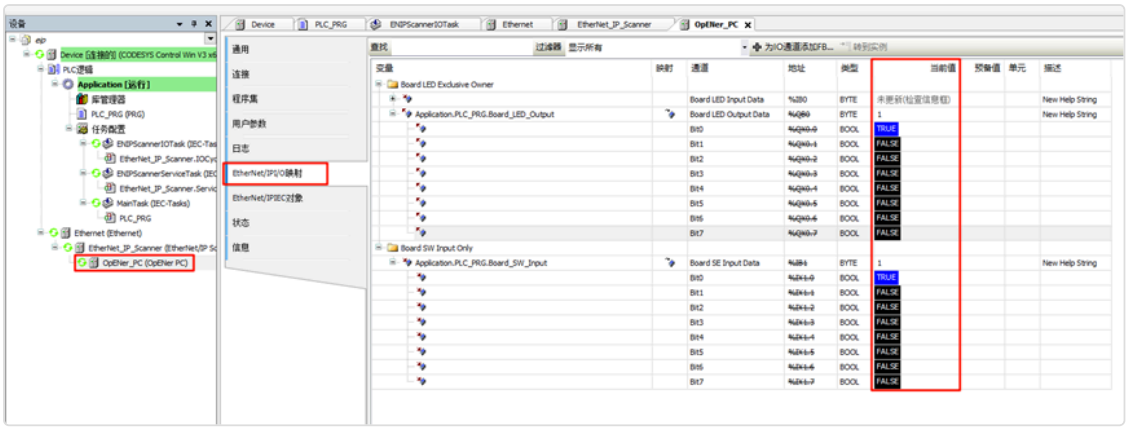
Next, click **Ethernet/IP I/O Mapping**, where we need to map the previously defined ST variables to the variables in this section. Map `Board_LED_Output` to the channel: **Board LED Output Data**; map `Board_SW_Input` to the channel: **Board SE Input Data**.



Finally, click **Build** -> **Generate Code**, then select **Online** -> **Login**. At this point, you can dynamically observe the program's running status. For example, when holding down KEY1 on the EtherKit development board, you will notice that LED0 (red) is off, and when releasing KEY1, LED0 stays on. When holding down KEY2, LED2 (green) will be off, and when releasing KEY2, LED2 stays on.

At the same time, you can also observe the current value of the bit in **OpENer_PC (OpENer PC)** -> **EtherNet/IP I/O Mapping**. When the

corresponding bit of the key is TRUE, the key is pressed, and the corresponding LED bit turns on, showing the current value as TRUE:



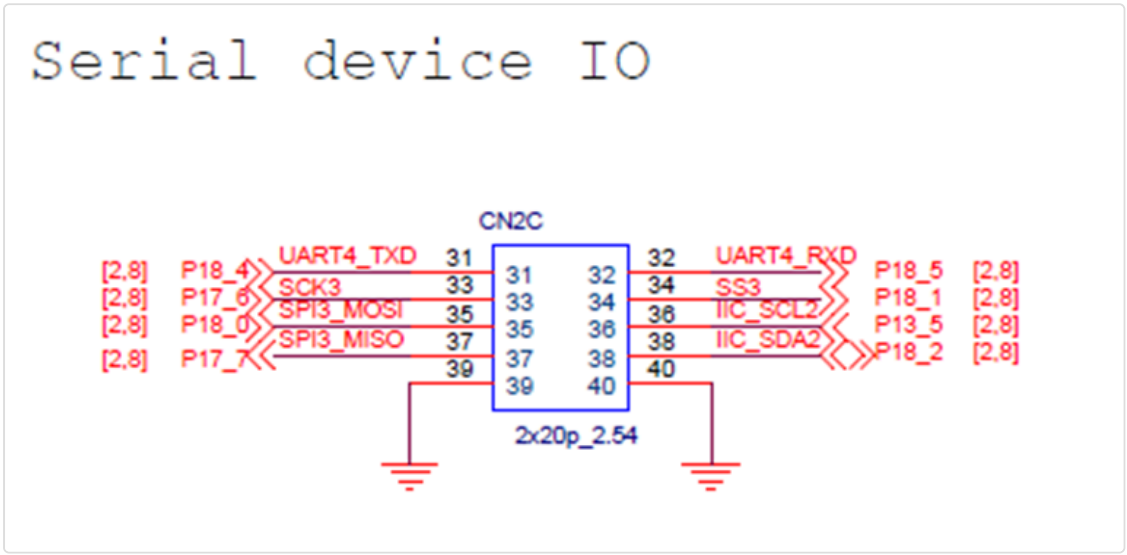
5.4. Modbus-TCP/IP Usage Instructions

English | **中文**

Introduction

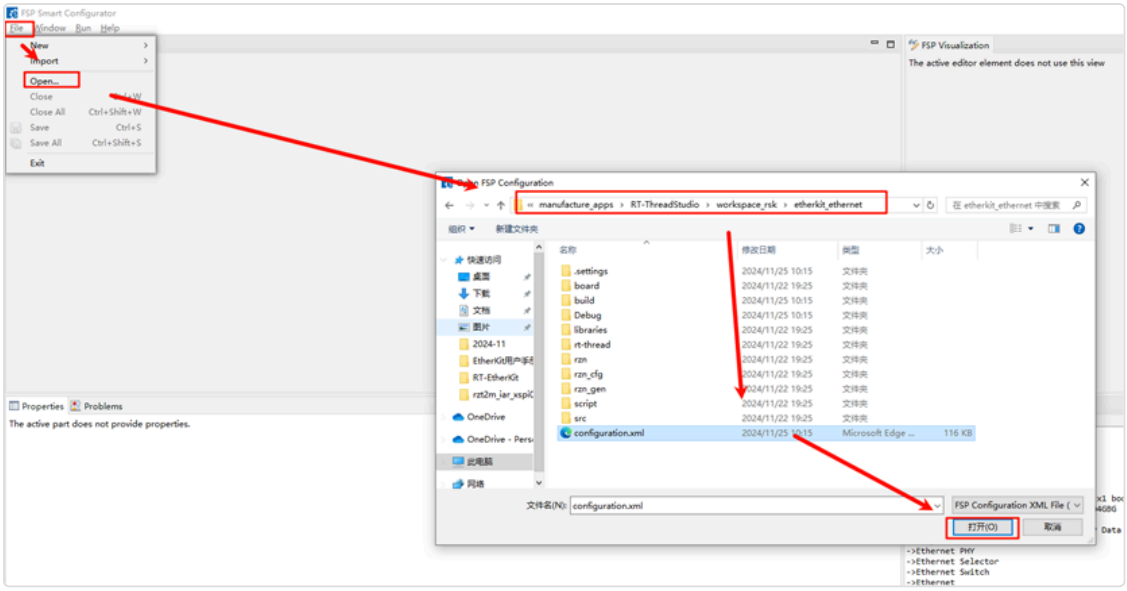
This example is based on the agile_modbus package and demonstrates Modbus protocol communication over TCP/IP.

Hardware Requirements

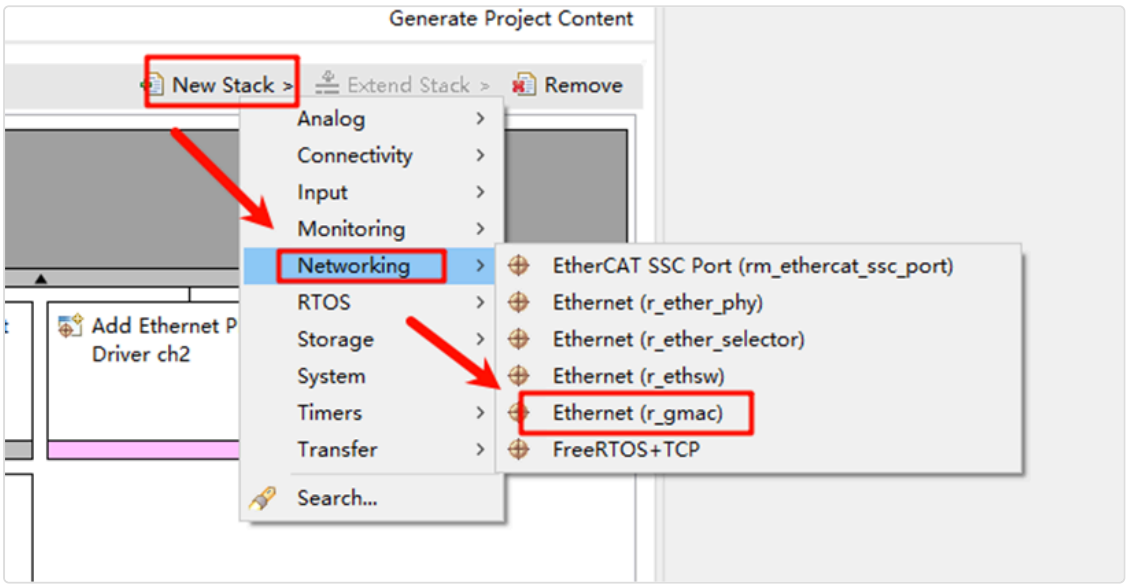


FSP Configuration Instructions

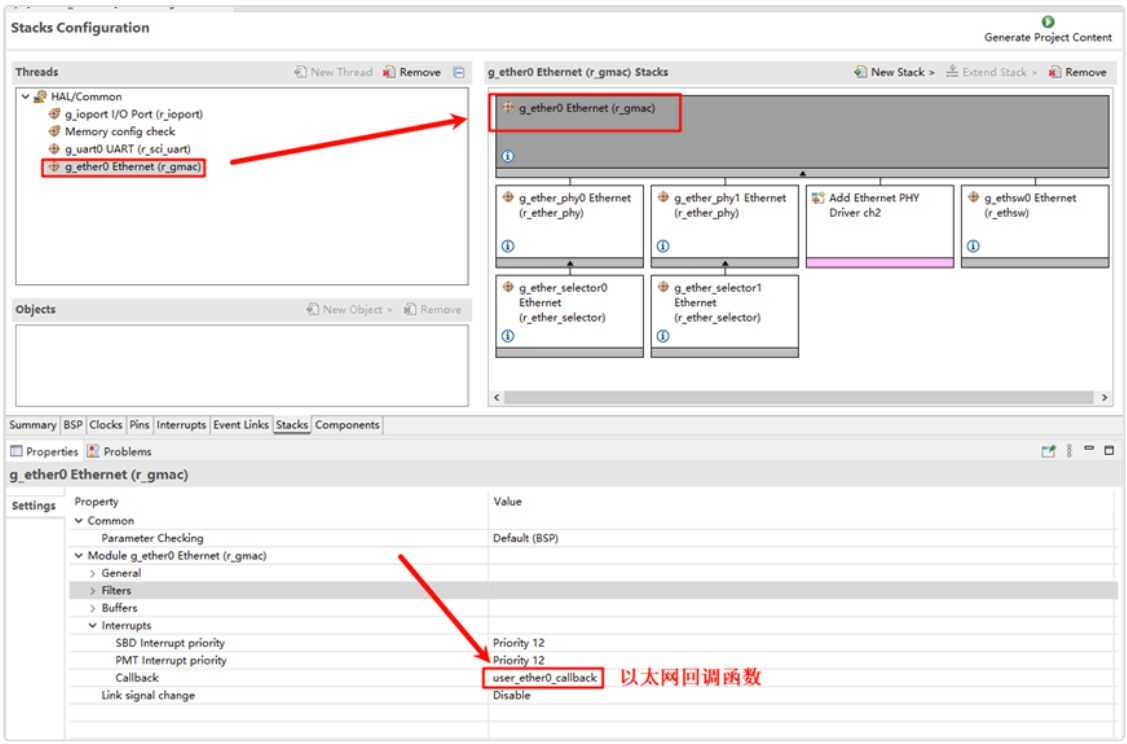
- Open the project configuration file `configuration.xml`:



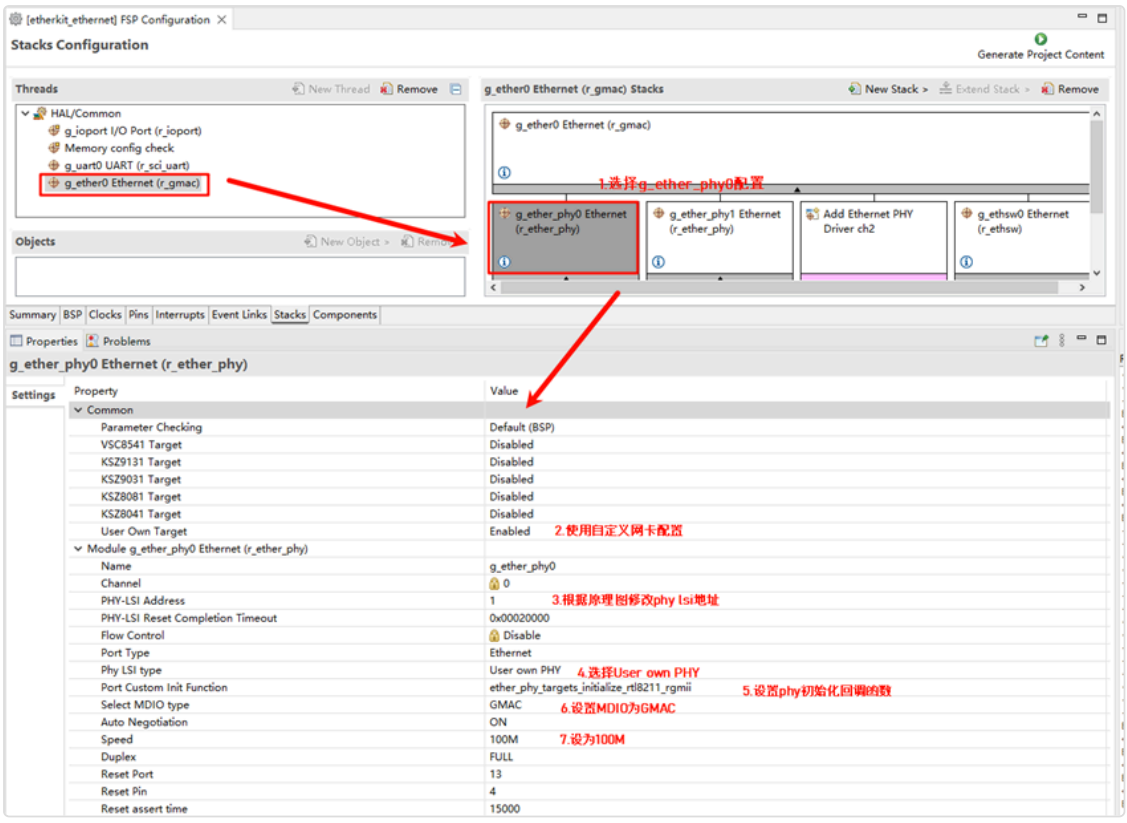
- Add the `r_gamc` stack:



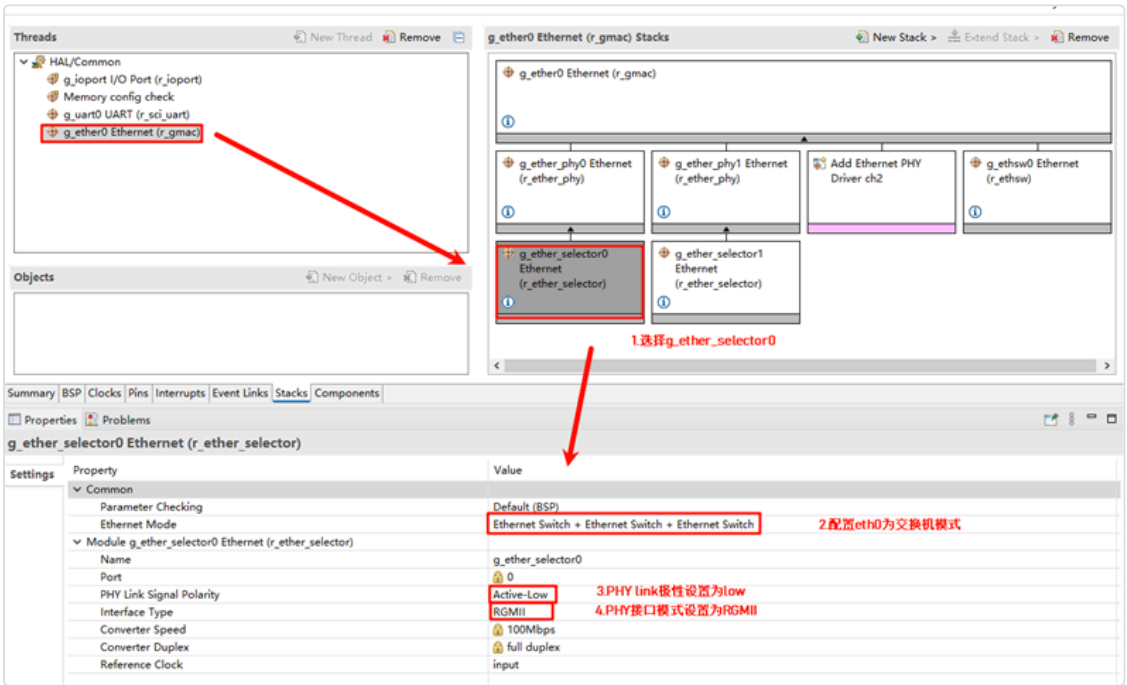
- Click on `g_ether0` Ethernet and configure the interrupt callback function as `user_ether0_callback`:



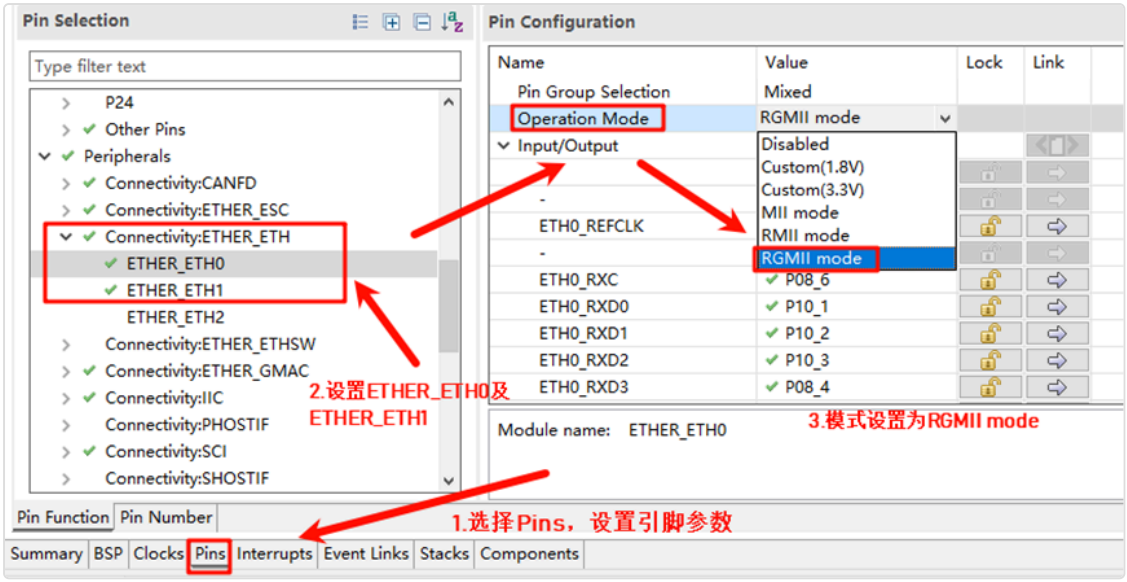
- Next, configure the PHY information. Select `g_ether_phy0`, set the Common configuration to "User Own Target," change the PHY LSI address to 1 (refer to the schematic for the exact address), and set the PHY initialization callback function to `ether_phy_targets_initialize_rtl8211_rgmii()`. Also, set MDIO to GMAC.



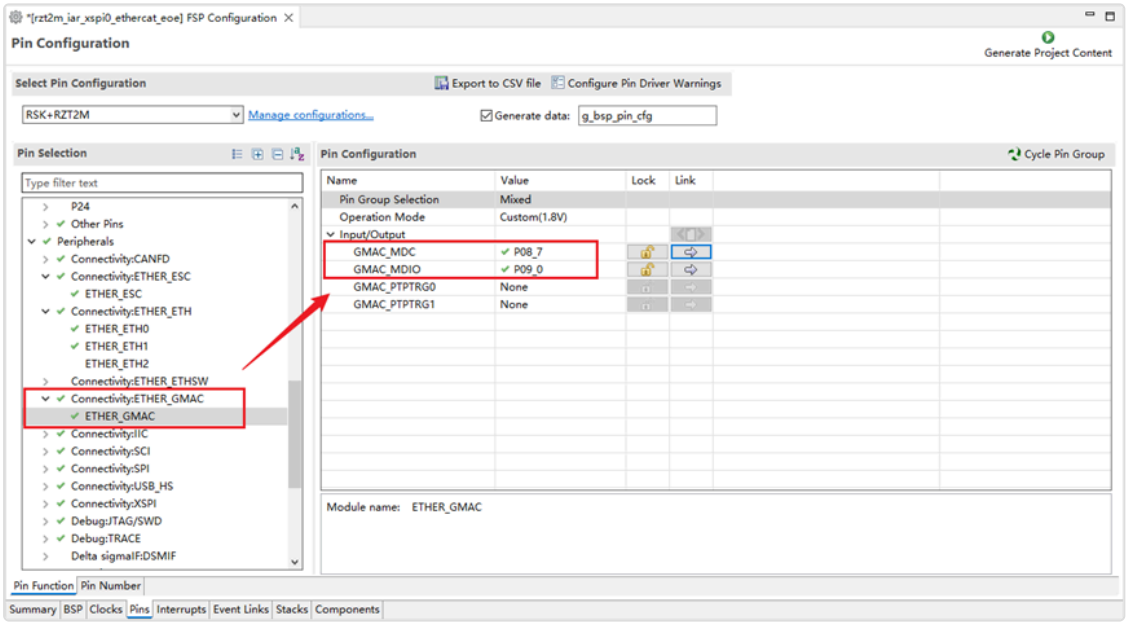
- Configure `g_ether_selector0`, set the Ethernet mode to **Switch mode**, set the PHY link to default **active-low**, and configure the PHY interface mode to **RGMII**.



Configure the network card pin parameters, setting the operation mode to RGMII:

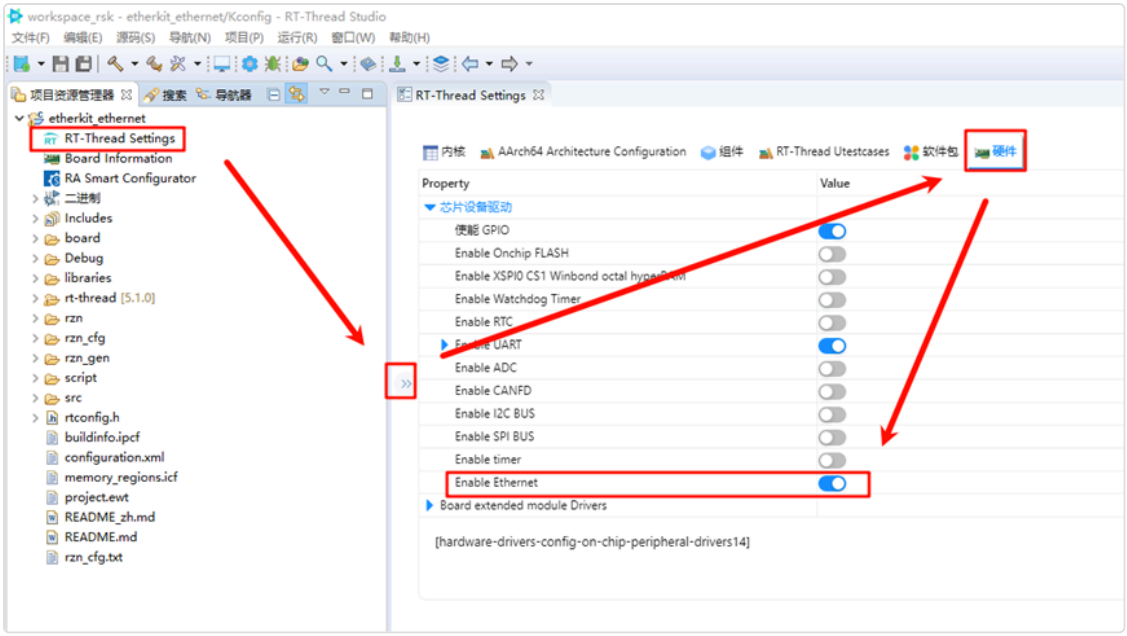


- ETHER_GMAC configuration:

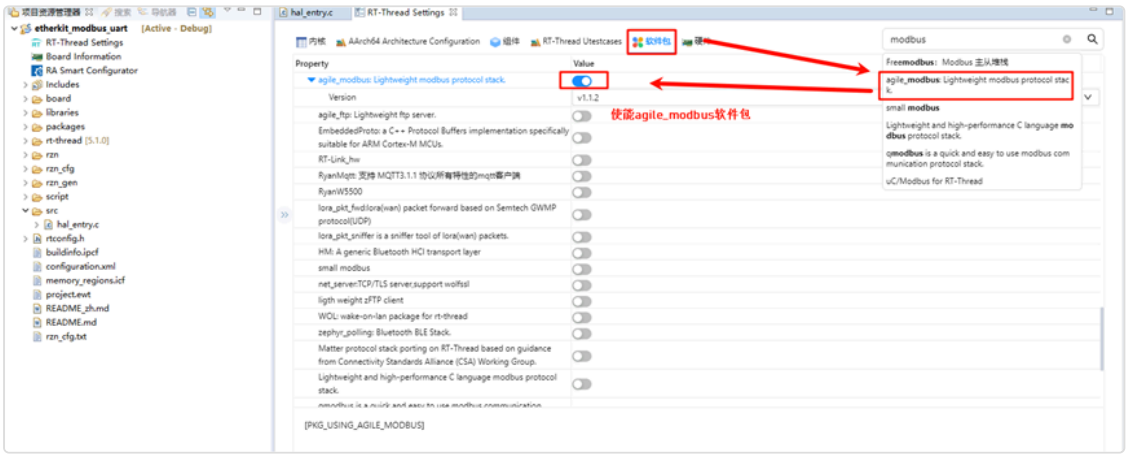


RT-Thread Settings Configuration

Return to the Studio project, configure **RT-Thread Settings**, select hardware options, and enable Ethernet by finding the chip device driver:



In the software package interface, search for "modbus," select the `agile_modbus` package, and enable it:



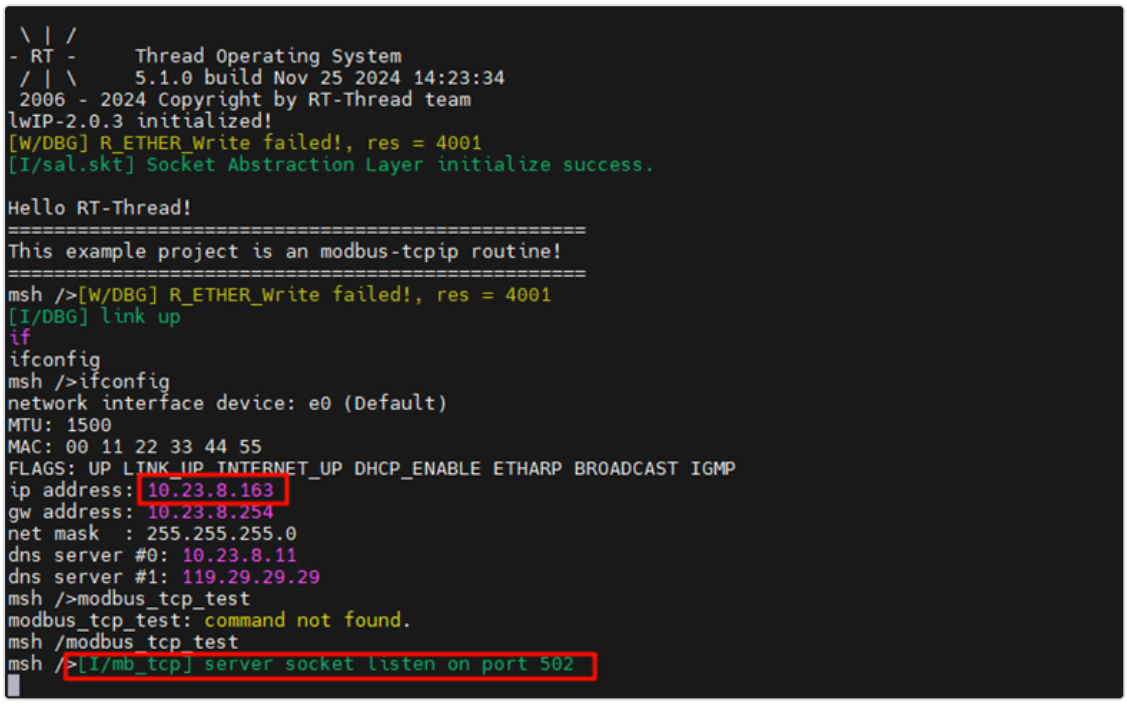
Compilation & Download

- **RT-Thread Studio**: Download the EtherKit resource package in the RT-Thread Studio package manager, create a new project, and compile it.
- **IAR**: First, double-click `mklinks.bat` to create the link between the rt-thread and libraries folders. Then, use Env to generate the IAR project. Finally, double-click `project.eww` to open the IAR project and compile it.

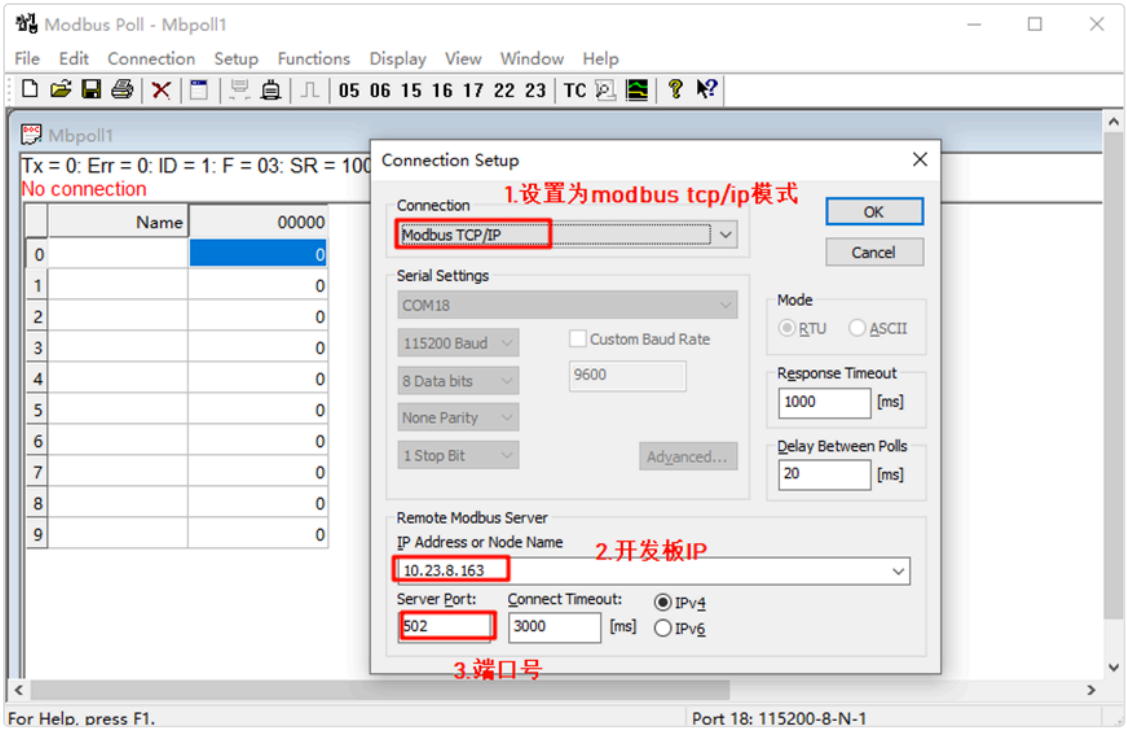
Once the compilation is complete, connect the Jlink interface of the development board to the PC and download the firmware to the board.

Running Results

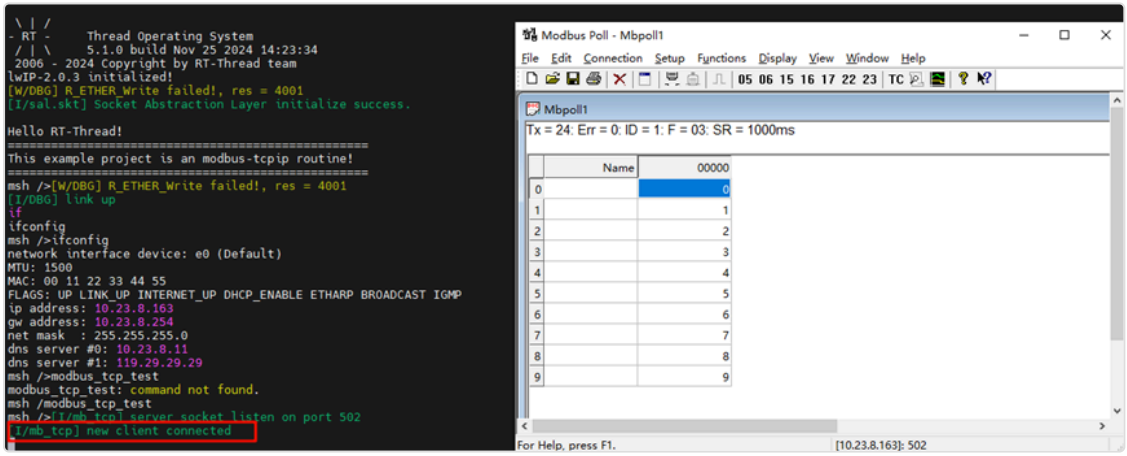
First, use an Ethernet cable to connect the development board's network port to a switch (if your computer has an extra Ethernet port, you can also use a shared adapter). Then, enter the command `modbus_tcp_test` in the serial tool to start the Modbus-TCP example:



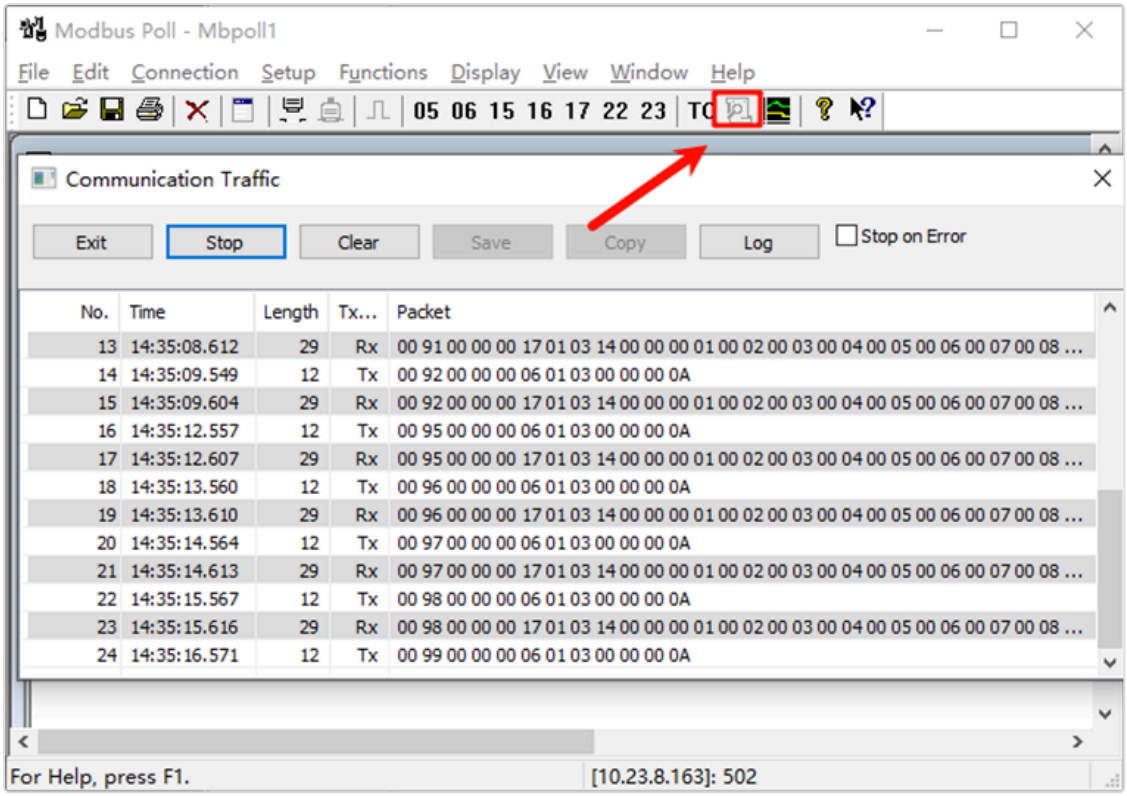
Open the **Modbus Poll** software, connect to the development board, set the mode to **Modbus TCP/IP**, set the IP to the development board's IP address, and the port number to 502:



After a successful connection, the development board's terminal will display that the Modbus client is connected:



Return to the **Modbus Poll** software, and you will see that the read and write coil functions are working correctly:


Modbus Poll - Mbpoll1
File Edit Connection Setup Functions Display View Window Help
05 06 15 16 17 22 23
Communication Traffic
Exit
Stop
Clear
Save
Copy
Log
Stop on Error
No. Time Length Tx... Packet
13 14:35:08.612 29 Rx 00 91 00 00 00 17 01 03 14 00 00 00 01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 ...
14 14:35:09.549 12 Tx 00 92 00 00 00 06 01 03 00 00 00 0A
15 14:35:09.604 29 Rx 00 92 00 00 00 17 01 03 14 00 00 00 01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 ...
16 14:35:12.557 12 Tx 00 95 00 00 00 06 01 03 00 00 00 0A
17 14:35:12.607 29 Rx 00 95 00 00 00 17 01 03 14 00 00 00 01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 ...
18 14:35:13.560 12 Tx 00 96 00 00 00 06 01 03 00 00 00 0A
19 14:35:13.610 29 Rx 00 96 00 00 00 17 01 03 14 00 00 00 01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 ...
20 14:35:14.564 12 Tx 00 97 00 00 00 06 01 03 00 00 00 0A
21 14:35:14.613 29 Rx 00 97 00 00 00 17 01 03 14 00 00 00 01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 ...
22 14:35:15.567 12 Tx 00 98 00 00 00 06 01 03 00 00 00 0A
23 14:35:15.616 29 Rx 00 98 00 00 00 17 01 03 14 00 00 00 01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 ...
24 14:35:16.571 12 Tx 00 99 00 00 00 06 01 03 00 00 00 0A
For Help, press F1.
[10.23.8.163]: 502

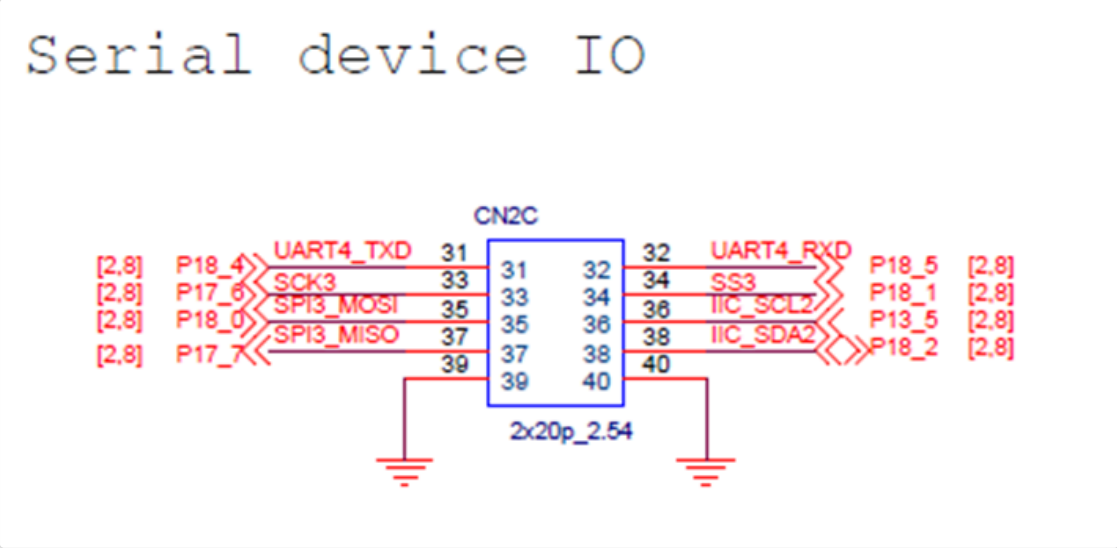
5.5. Modbus-UART Usage Instructions

English | **中文**

Introduction

This example is based on the agile_modbus package and demonstrates Modbus protocol communication over UART. Modbus UART is a version of the Modbus protocol implemented via serial communication and is widely used in industrial automation and control systems. Modbus is an open communication protocol used to transfer data between control devices and supports multiple physical layers such as UART, TCP/IP, and RS-485/232.

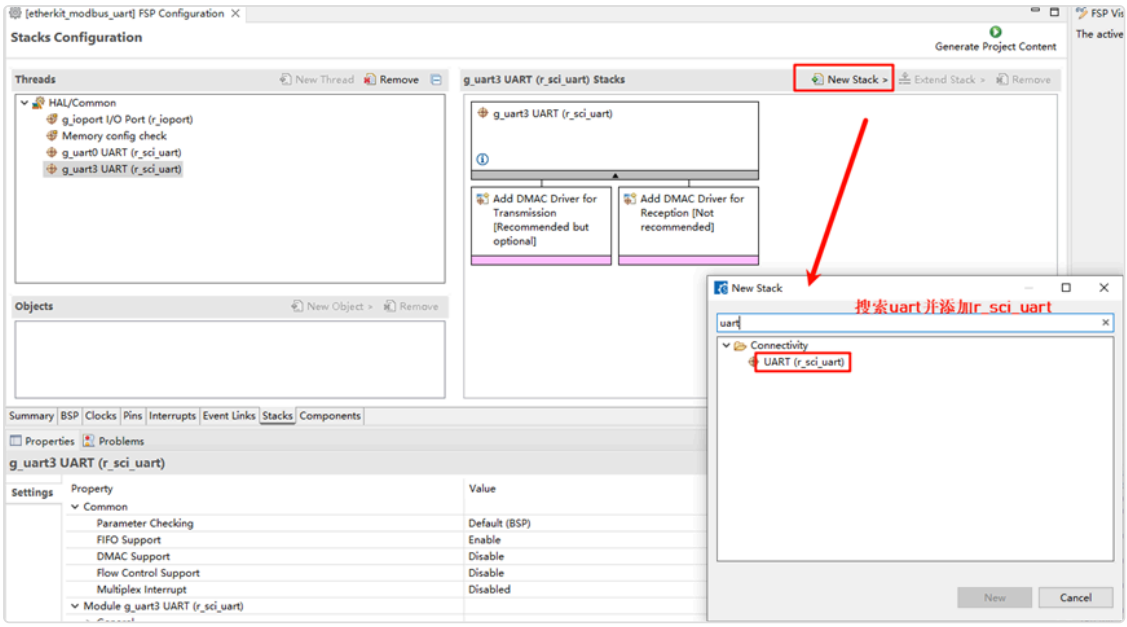
Hardware Requirements



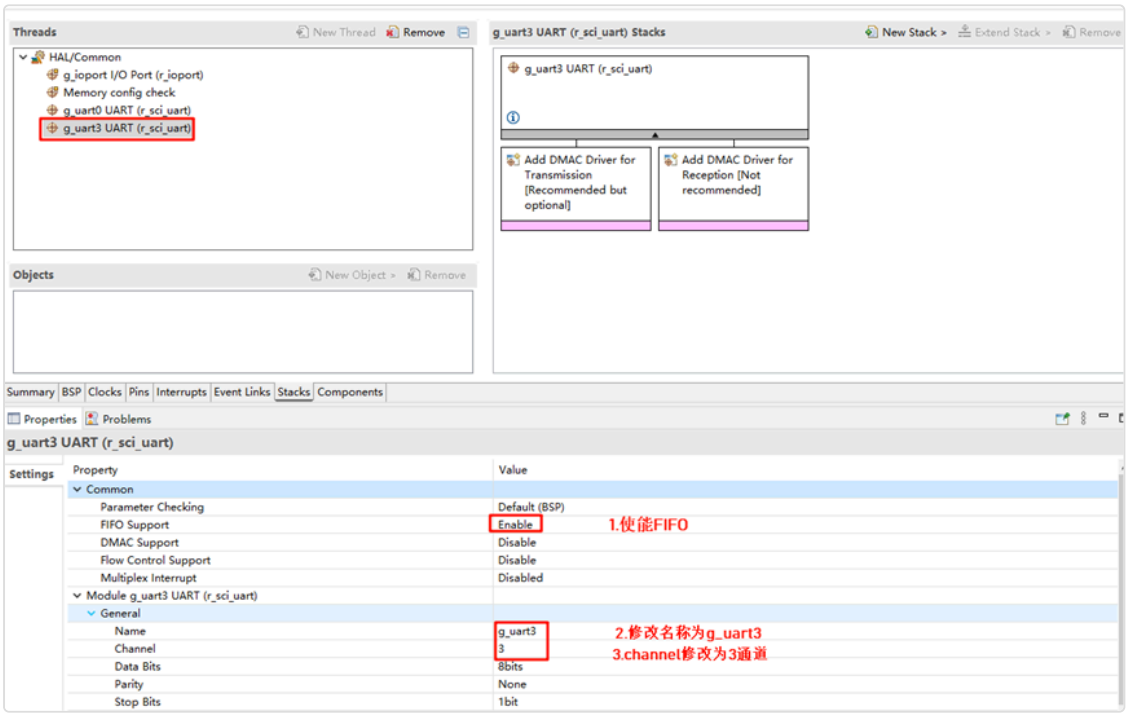
As shown in the image above, the peripheral used in this example is the SCI module. SCI3 is configured in UART mode, with TX pin as P18_0 and RX pin as P17_7.

FSP Configuration Instructions

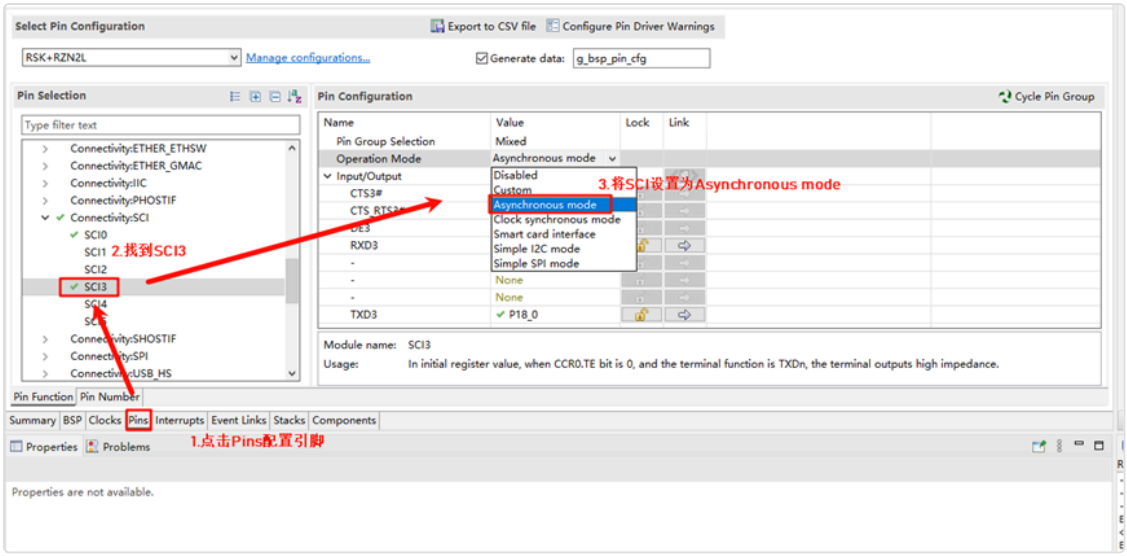
Open the `configuration.xml` file in the project and add a new stack:



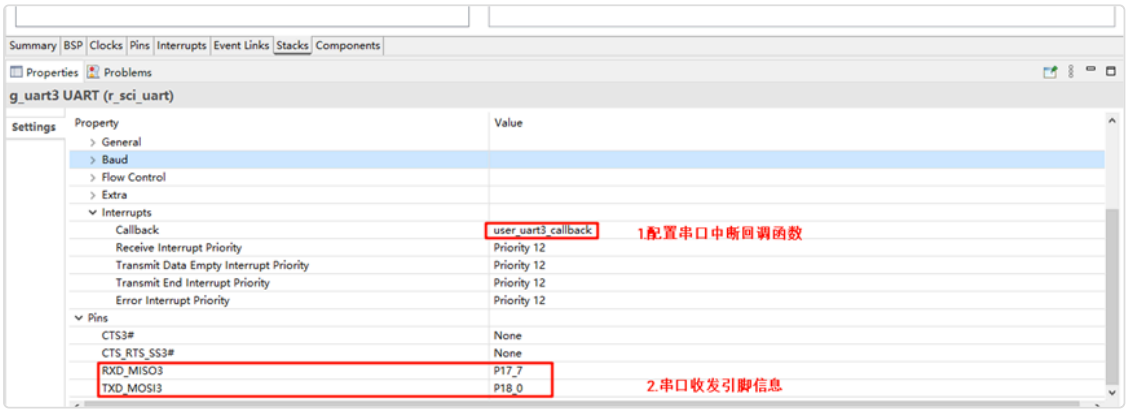
Enable the `r_sci_uart` configuration and support FIFO. Set the channel number to 3:



Click on **Pins**, configure SCI3, and set the SCI mode to **Asynchronous mode**. The corresponding pins will be enabled accordingly:

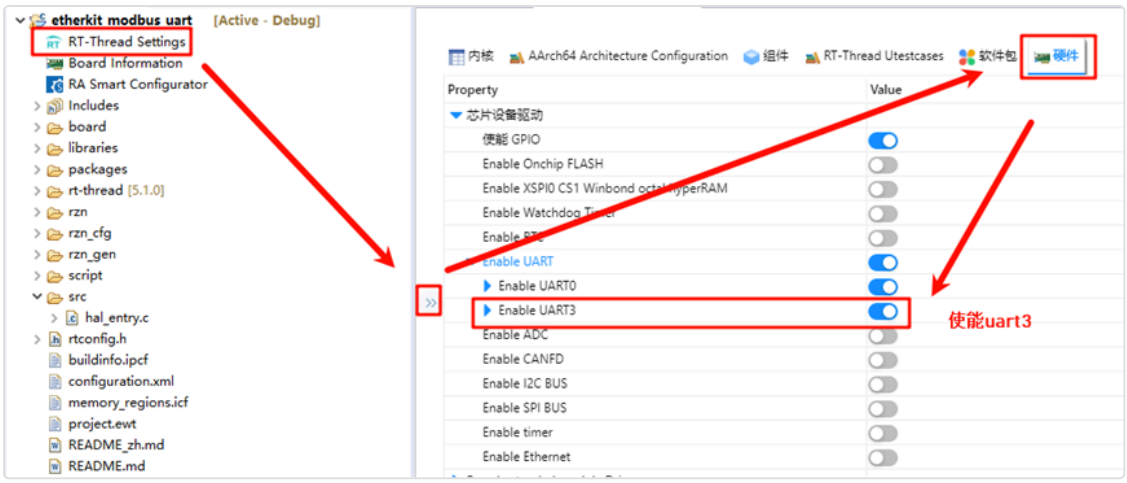


Return to the stack configuration page, expand and set the interrupt callback function to `user_uart3_callback`. The corresponding UART pin information will be shown below:



RT-Thread Settings Configuration

Back in RT-Thread Studio, go to **RT-Thread Settings**, enable UART3 for the serial port configuration:



In the software package interface, search for "modbus," select the `agile_modbus` package, and enable it:

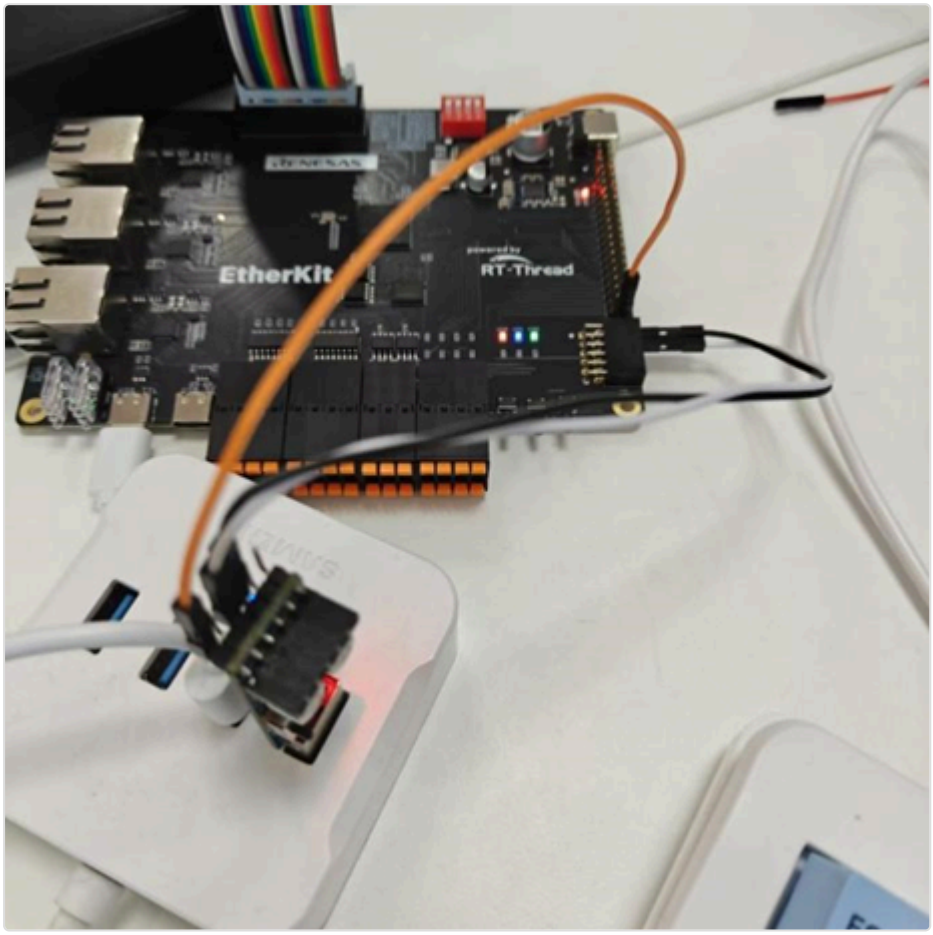
Compilation & Download

- **RT-Thread Studio**: Download the EtherKit resource package in the RT-Thread Studio package manager, create a new project, and compile it.
- **IAR**: First, double-click `mklinks.bat` to create the link between the rt-thread and libraries folders. Then, use Env to generate the IAR project. Finally, double-click `project.eww` to open the IAR project and compile it.

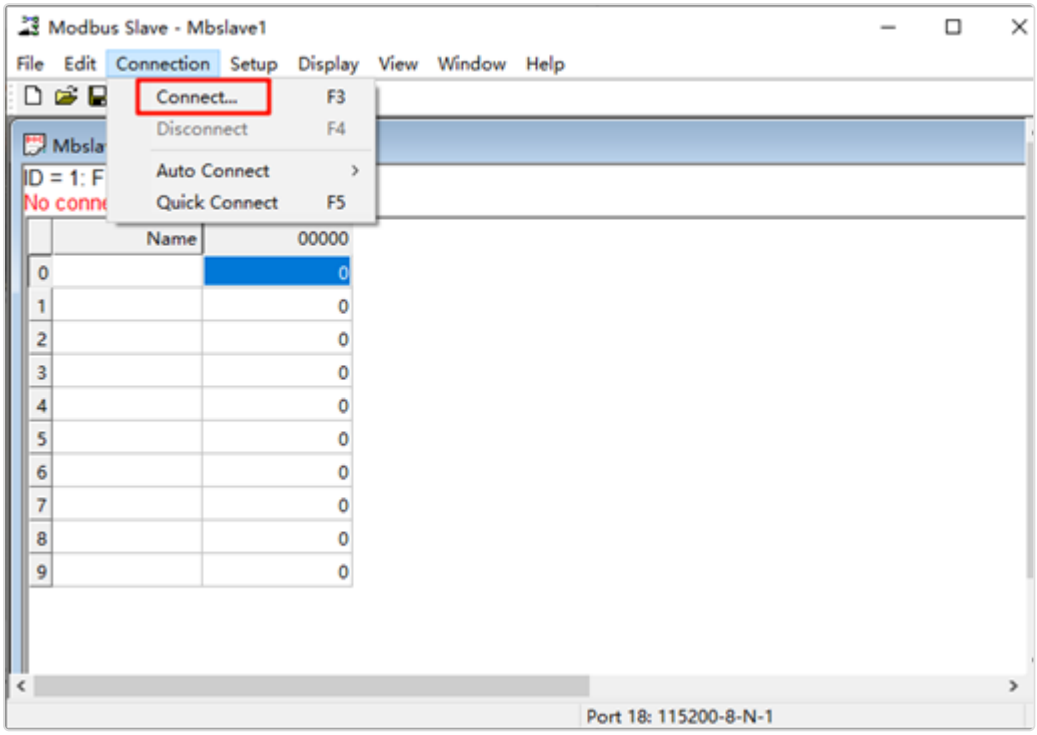
Once the compilation is complete, connect the Jlink interface of the development board to the PC and download the firmware to the board.

Running Results

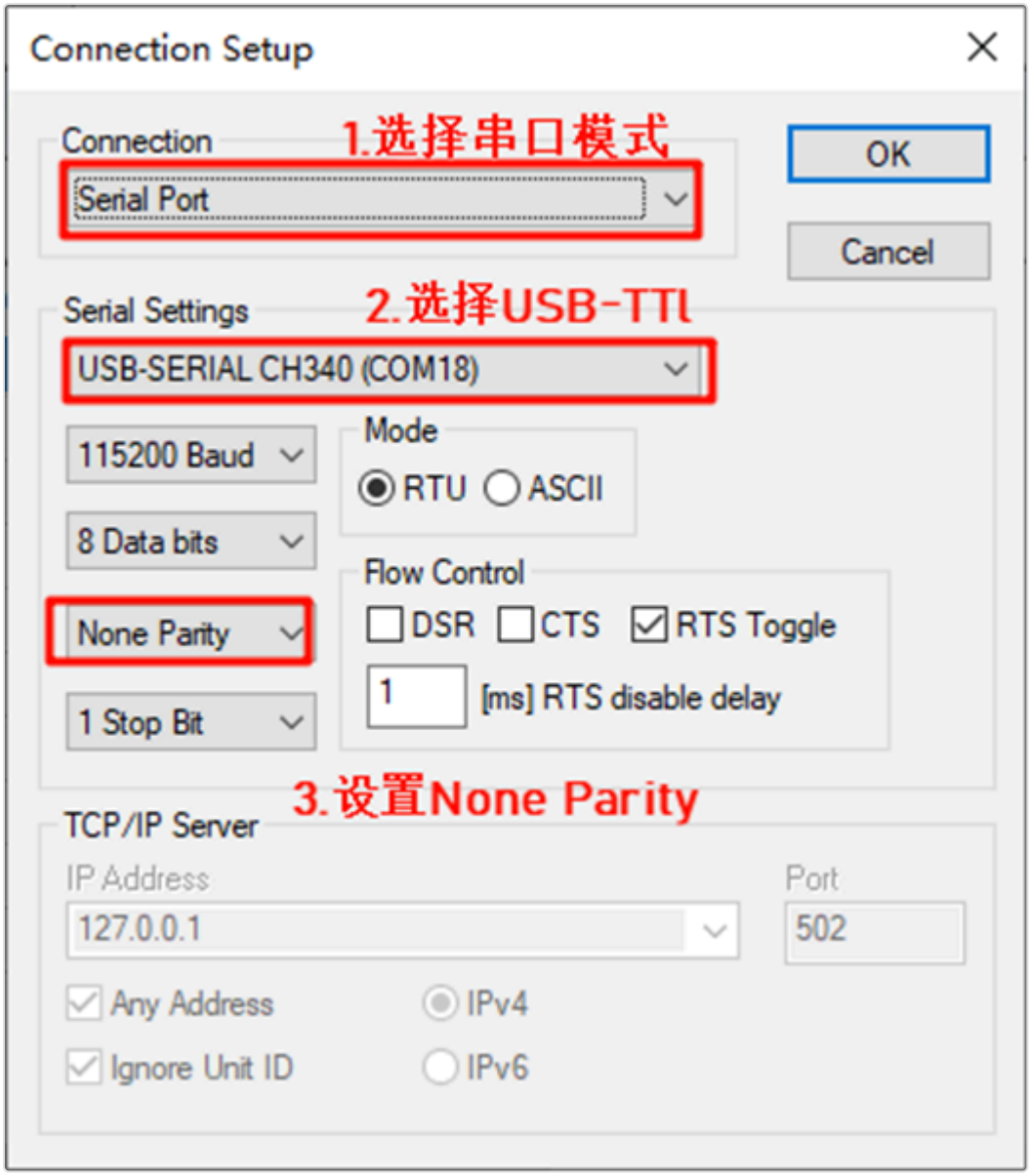
First, you need a USB-to-TTL module. Connect the TX and RX pins of the USB-to-TTL module to the RX and TX pins (P18_0 and P17_7) of the development board, respectively, as shown below:



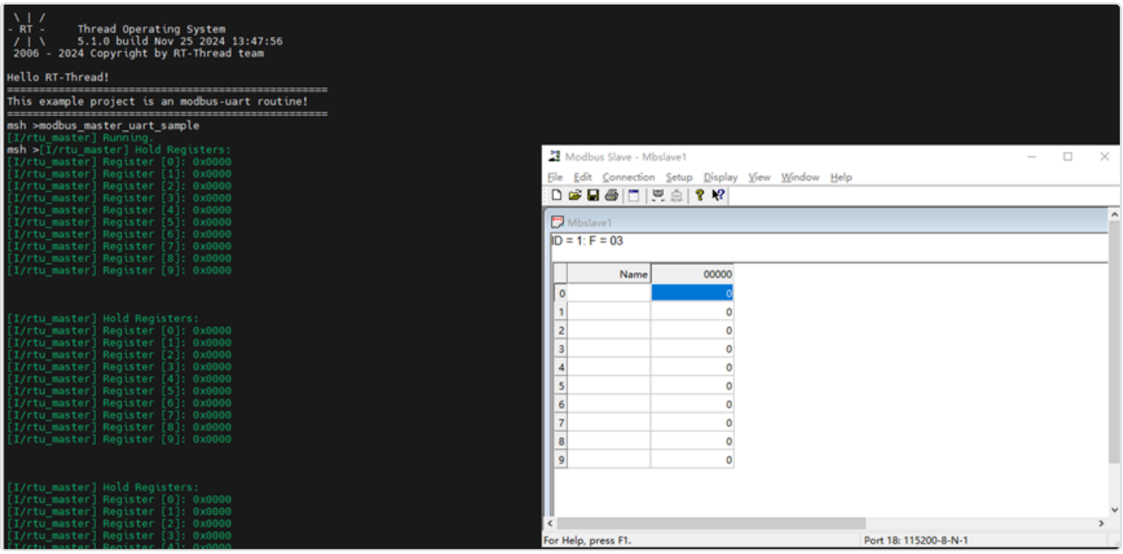
Next, open the Modbus Slave software and click **Connect**:



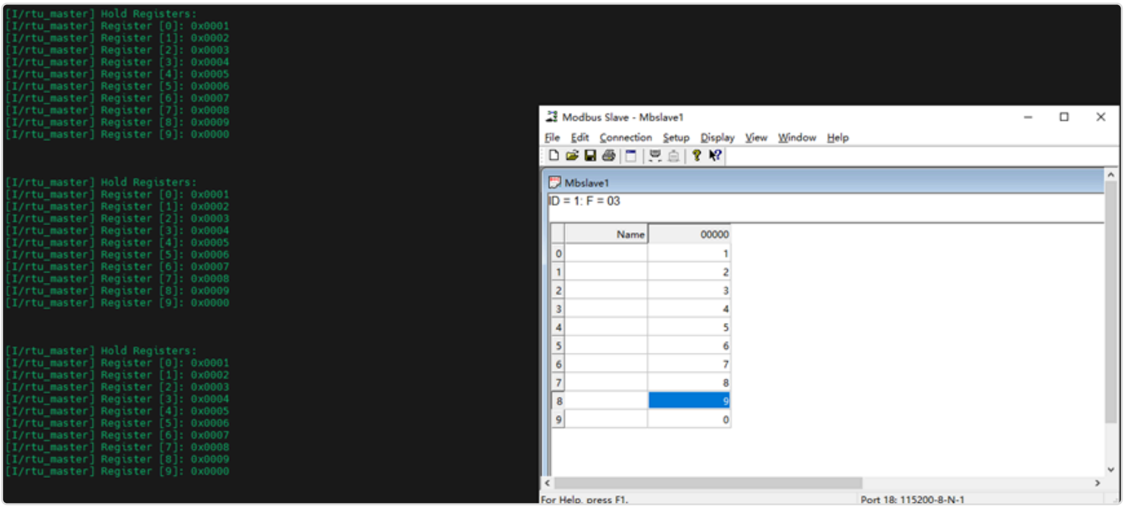
Configure the Modbus Slave settings. Select **Serial Mode** for the connection type, choose the USB-to-TTL module connected to the development board, and set **None Parity**:



Then, return to the serial terminal and enter the command `modbus_master_uart_sample` to start the Modbus master example:



The development board's UART3 functions as the master, and the PC acts as the slave. The master writes to the coil at the station number, and the terminal will display the updated registers:



5.6. PROFINET Usage Instructions

English | **中文**

Introduction

PROFINET is an industrial Ethernet standard developed and promoted by PI (PROFIBUS and PROFINET International) and is widely used in the industrial automation field.

P-Net is an open-source PROFINET implementation, specifically designed for real-time network communication in embedded devices. It is a lightweight PROFINET protocol stack aimed at providing developers with a quick and efficient way to integrate PROFINET functionality into embedded platforms.

In this example, we will use the P-Net software package to implement PROFINET master-slave communication.

Prerequisites

Software Environment:

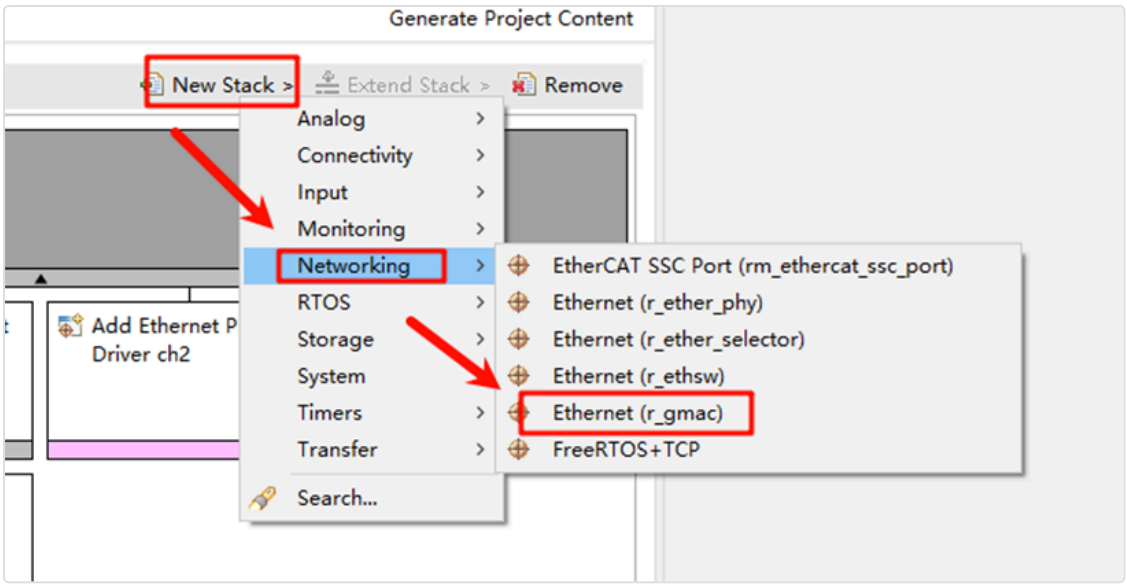
- CODESYS (PROFINET master simulation)
  - CODESYS
  - CODESYS Gateway (Gateway device)
  - CODESYS Control Win SysTray (Soft PLC device)
- Npcap(The software is running CODESYS must, need installed in advance!)

Hardware Environment:

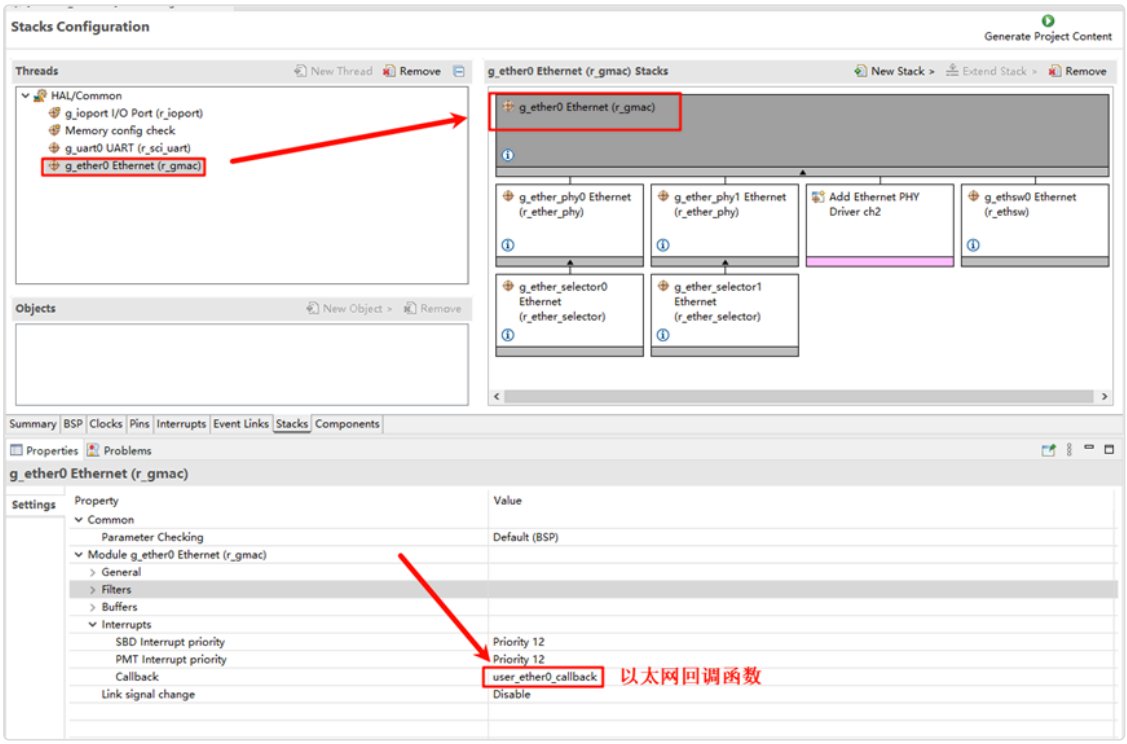
- EtherKit development board

FSP Configuration Instructions

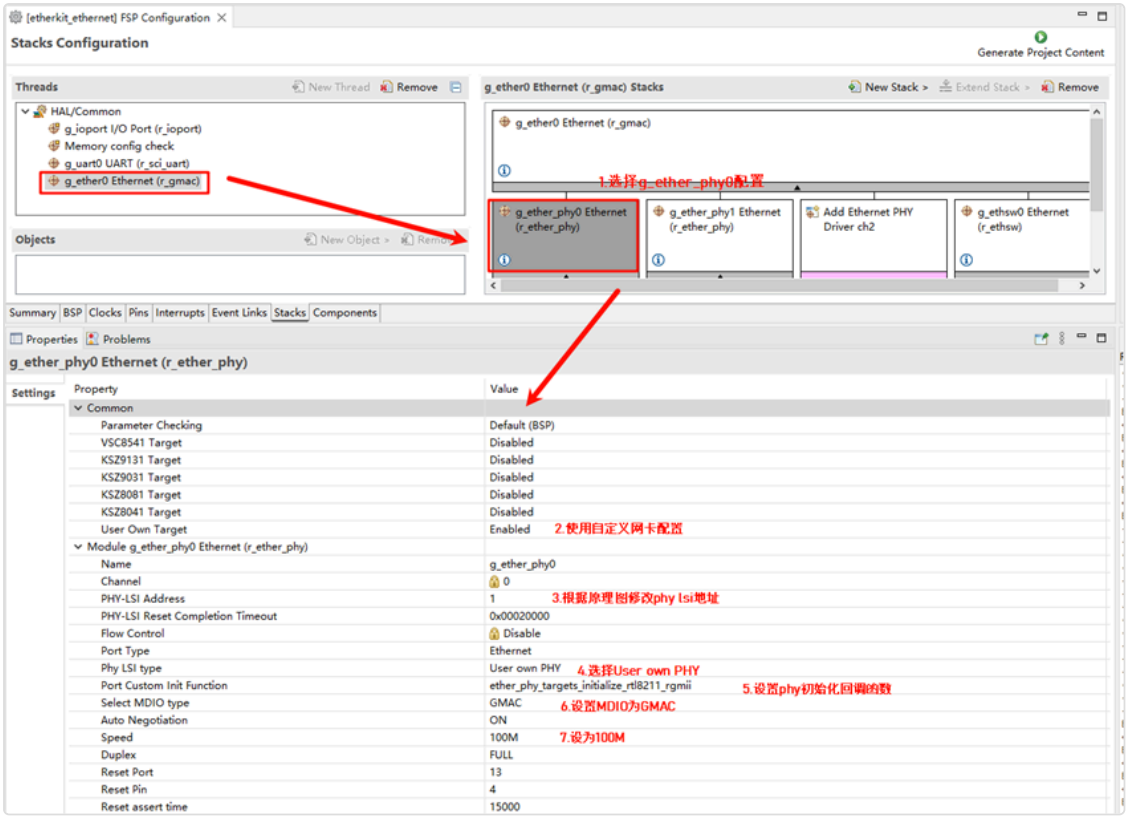
Open the project configuration file `configuration.xml` and add the `r_gamc` stack:



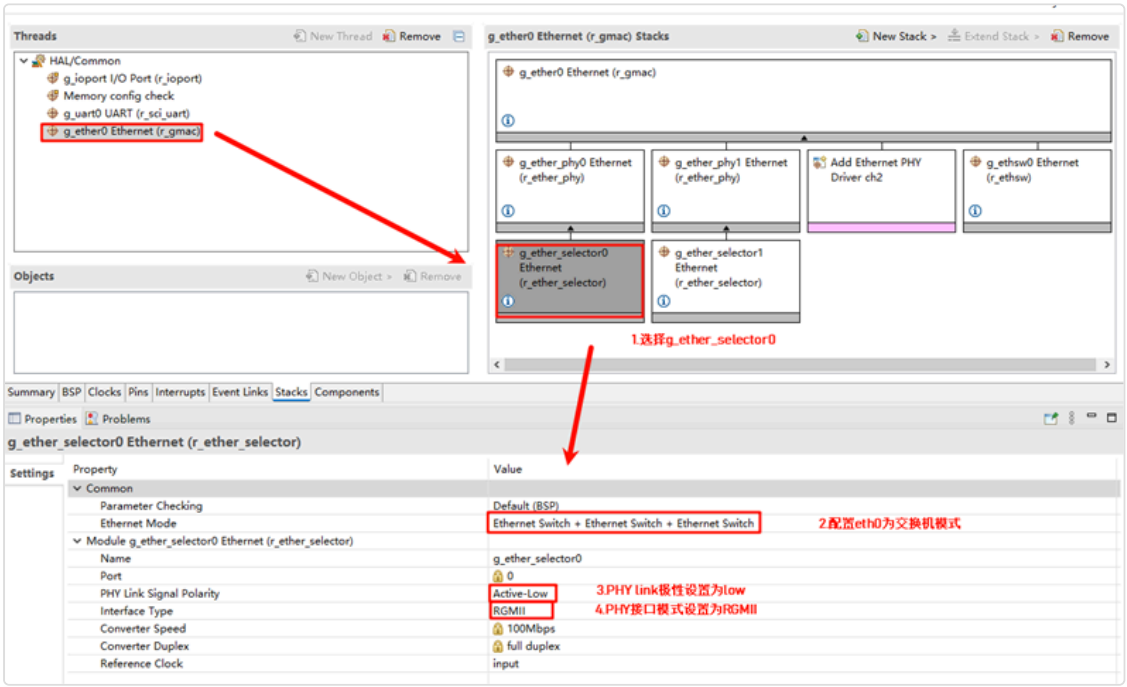
Next, click on `g_ether0 Ethernet`, and configure the interrupt callback function to `user_ether0_callback`:



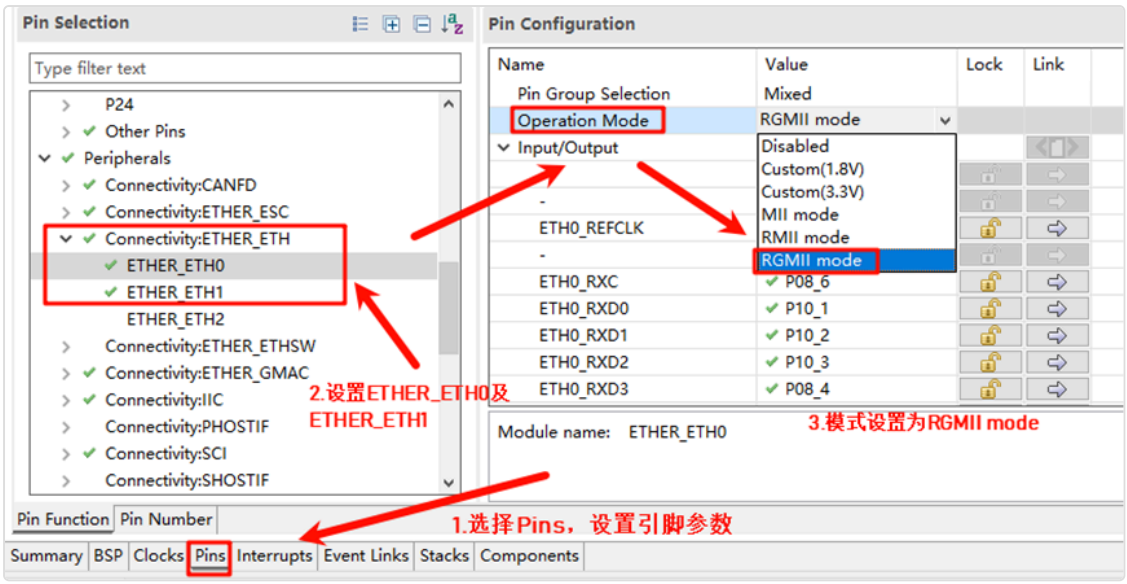
Now configure the PHY settings. Select `g_ether_phy0`, set the common configuration to "User Own Target", change the PHY LSI address to `1` (refer to the schematic for the exact address), and set the PHY initialization callback function to `ether_phy_targets_initialize_rtl8211_rgmii()`. Also, set the MDIO to GMAC.



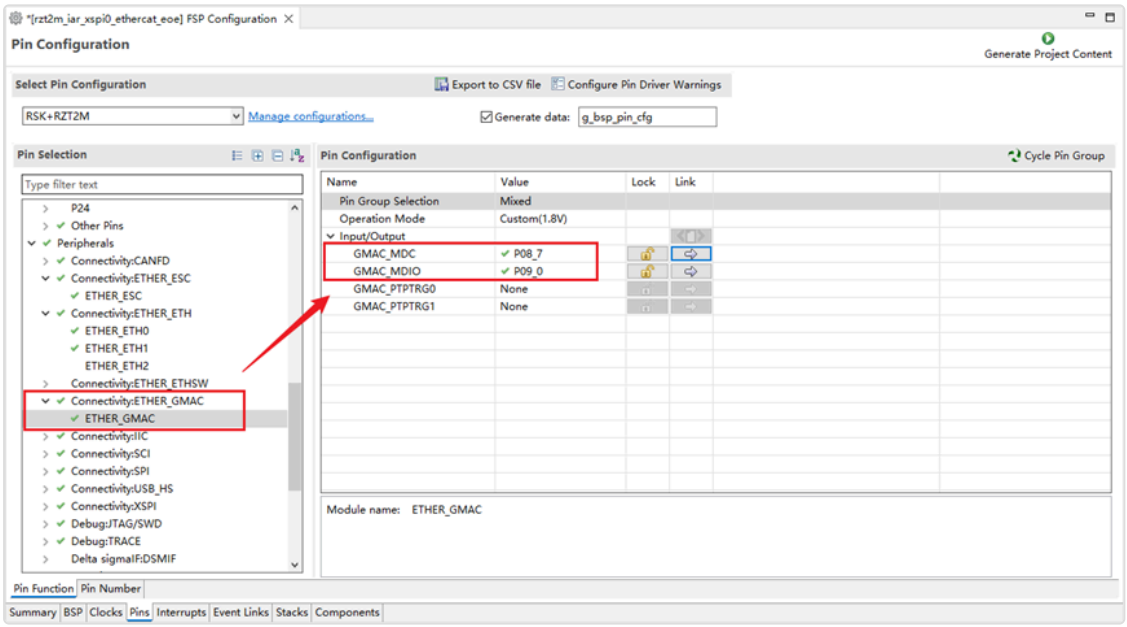
Next, configure `g_ether_selector0`, set the Ethernet mode to "Switch Mode", set the PHY link to "Default Active-Low", and choose "RGMII" for the PHY interface mode.



Configure the Ethernet pin parameters and select the operating mode to RGMII:

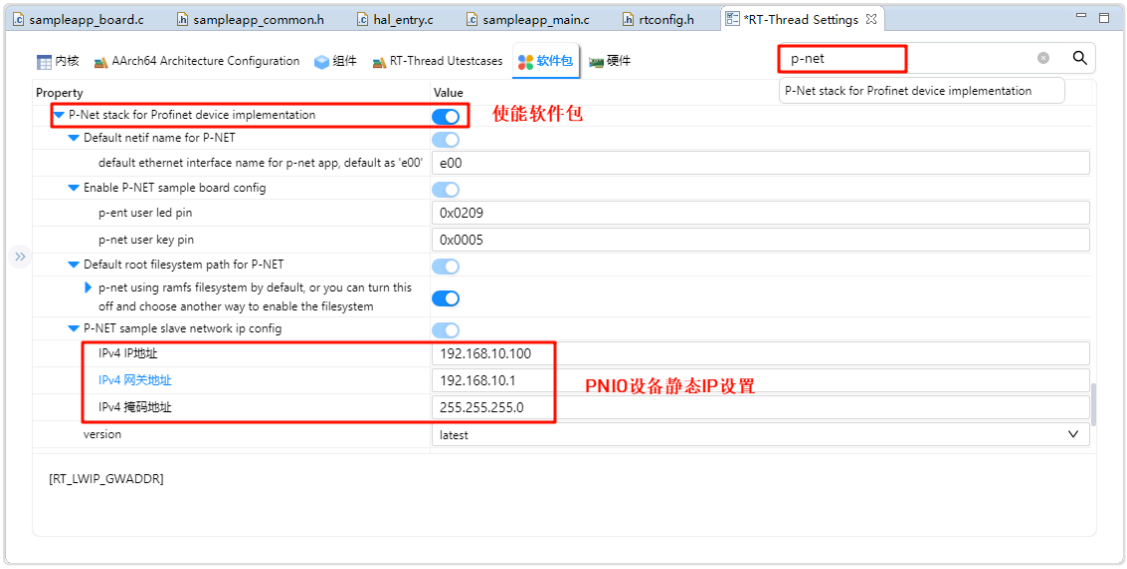


Finally, configure `ETHER_GMAC`:



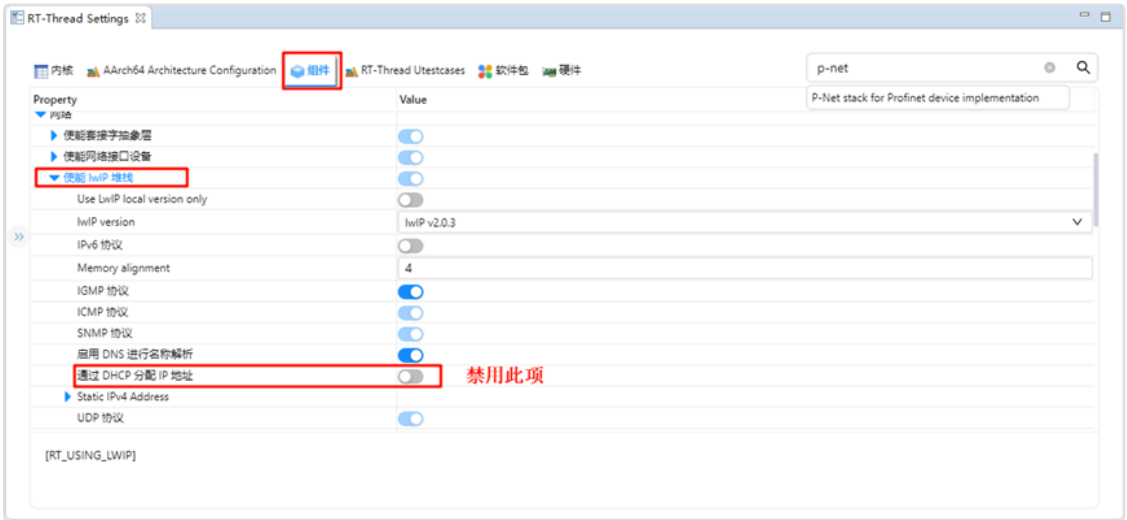
RT-Thread Settings Configuration

Double-click to open RT-Thread Settings, search for the p-net package in the search bar and enable it. Below are the related user configuration details:



- **Default netif name for p-net**: The interface name for the p-net network card device, default is e00.
- **Enable pnet sample board config**: Configuration for user LED and button on the p-net application.
- **Default root filesystem path for p-net**: Configuration for the p-net filesystem, default is using ramfs, with 8K of memory space allocated by default.
- **P-NET sample slave network ip config**: Static IP configuration for the p-net slave device (**Please disable RT_LWIP_DHCP functionality and use static IP**).

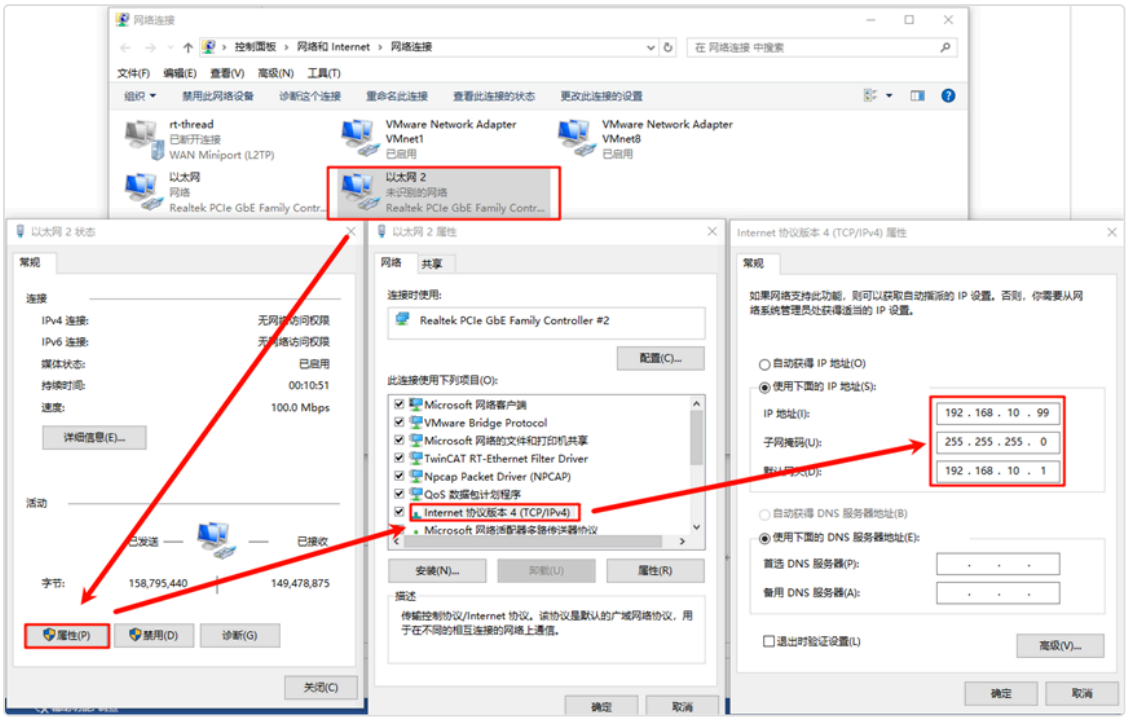
Next, we need to configure the system to disable DHCP and use a static IP. Click on the component -> enable the lwip stack, and select to disable DHCP:



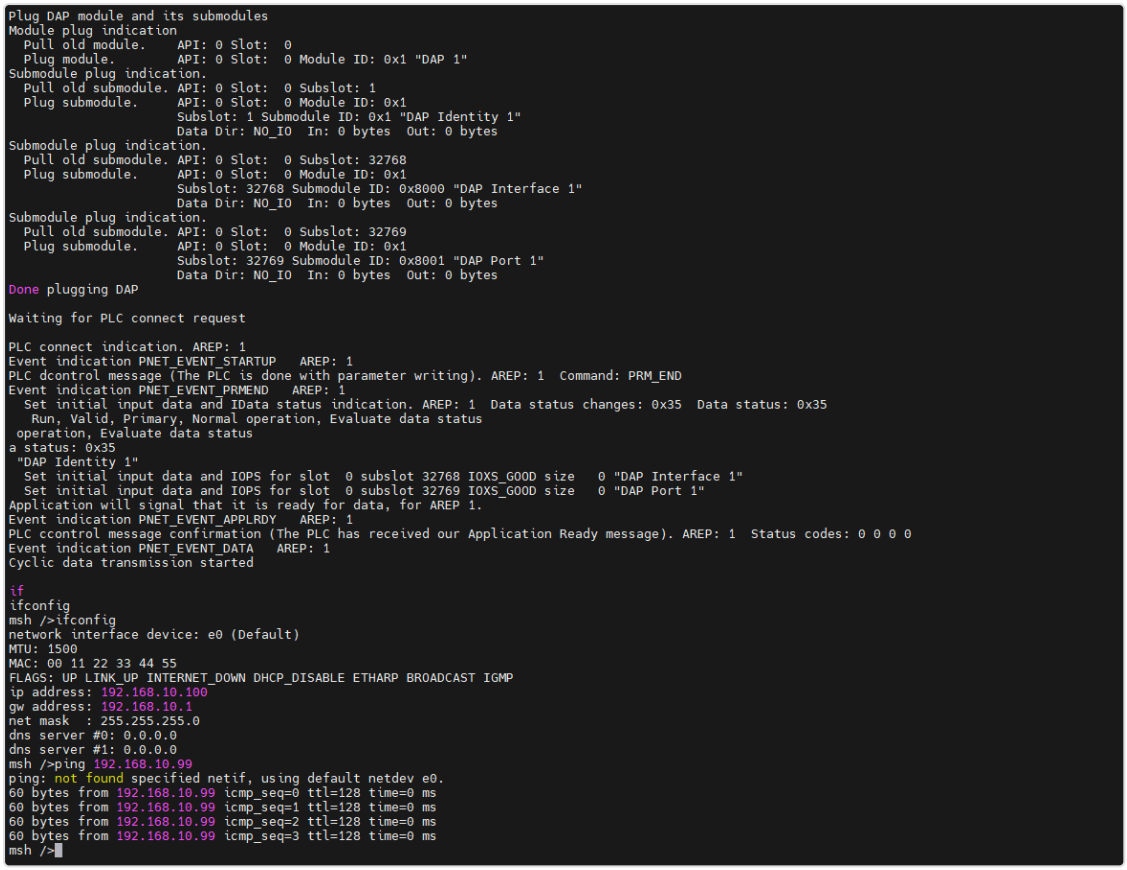
After completing the above configurations, compile the program and download it to the development board.

Network Configuration

Connect the development board to the PC using an Ethernet cable, and configure a static IP on the PC:



Check the IP information on the development board and test connectivity:



Soft PLC Startup

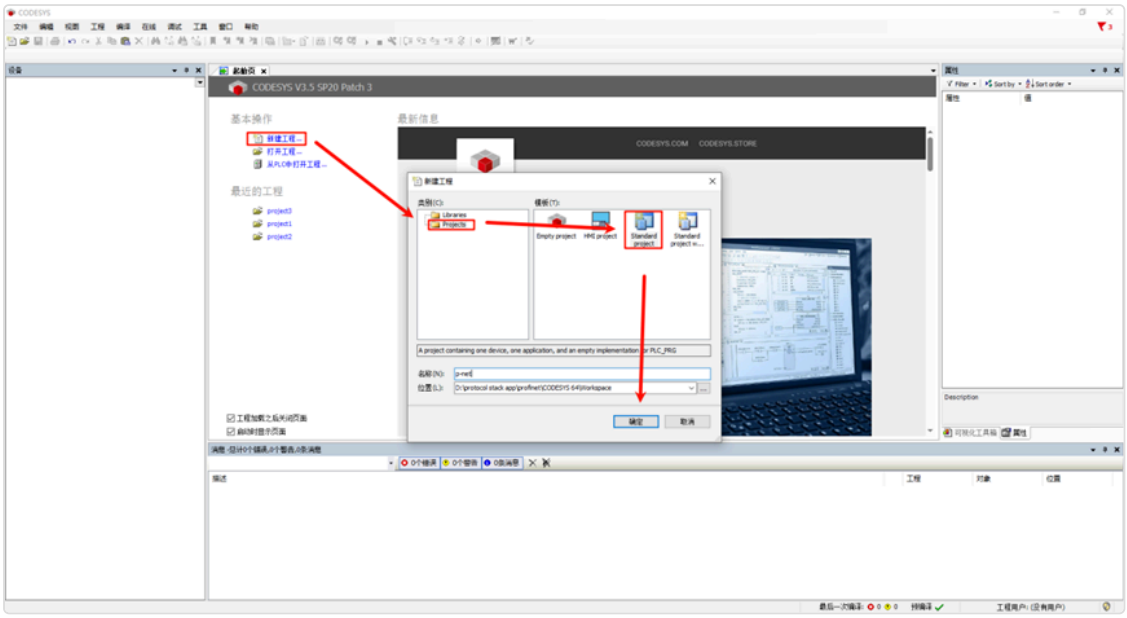
CODESYS Overview: CODESYS is a PLC software developed by 3S (Germany), integrating PLC logic, motion control, configuration, display, and other functions. CODESYS, short for **Controller Development System**, is an industrial automation programming tool based on the IEC 61131-3 standard. It supports various programming languages (e.g., Ladder Diagram, Structured Text, Function Block Diagram) and provides libraries and function modules, making it a widely used platform in industrial automation for developing and debugging PLCs and control systems.

Creating a Standard Project in CODESYS

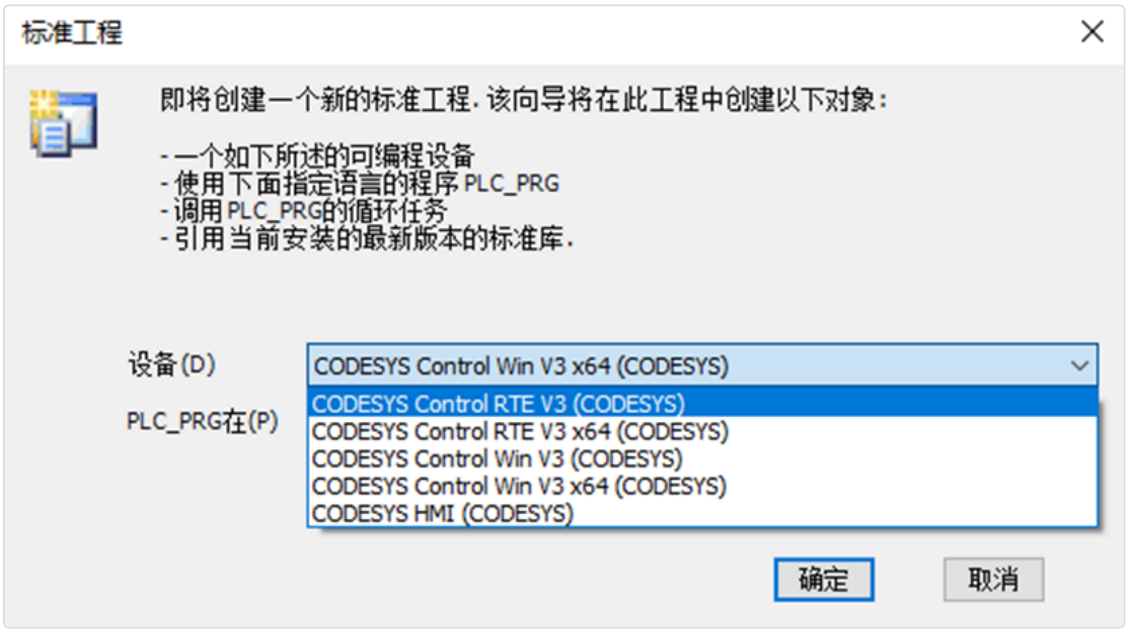
Ensure that you have CODESYS installed. After installation, the following three software components are required:

- **CODESYS V3.5 SP20 Patch 3**: PROFINET master simulation
- **CODESYS Gateway V3**: Gateway device
- **CODESYS Control Win V3 -x64 SysTray**: Soft PLC device

First, open **CODESYS V3.5 SP20 Patch 3**, choose **New Project** -> **Projects** -> **Standard project**, configure the project name and location, and click **OK**:

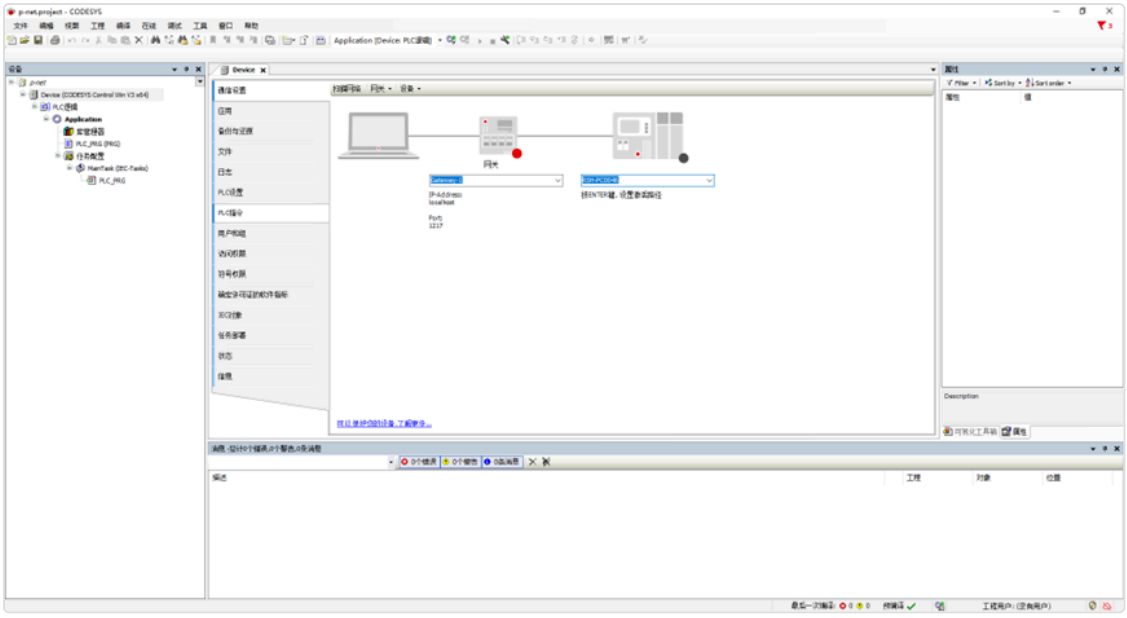


After the following popup, keep the default Settings (CODESYS Control Win V3 (CODESYS)/x64 (CODESYS)) and click OK:



> ***Note**: If you have purchased **CODESYS Control RTE SL**, select the device: **CODESYS Control RTE V3 (CODESYS) / x64 (CODESYS)**. For evaluation purposes, you can choose **CODESYS Control Win V3 (CODESYS) / x64** to create the project.*

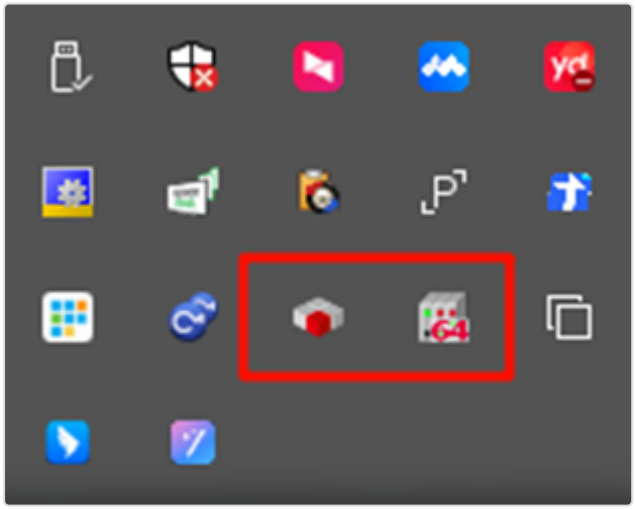
After creation, you will see the main interface:



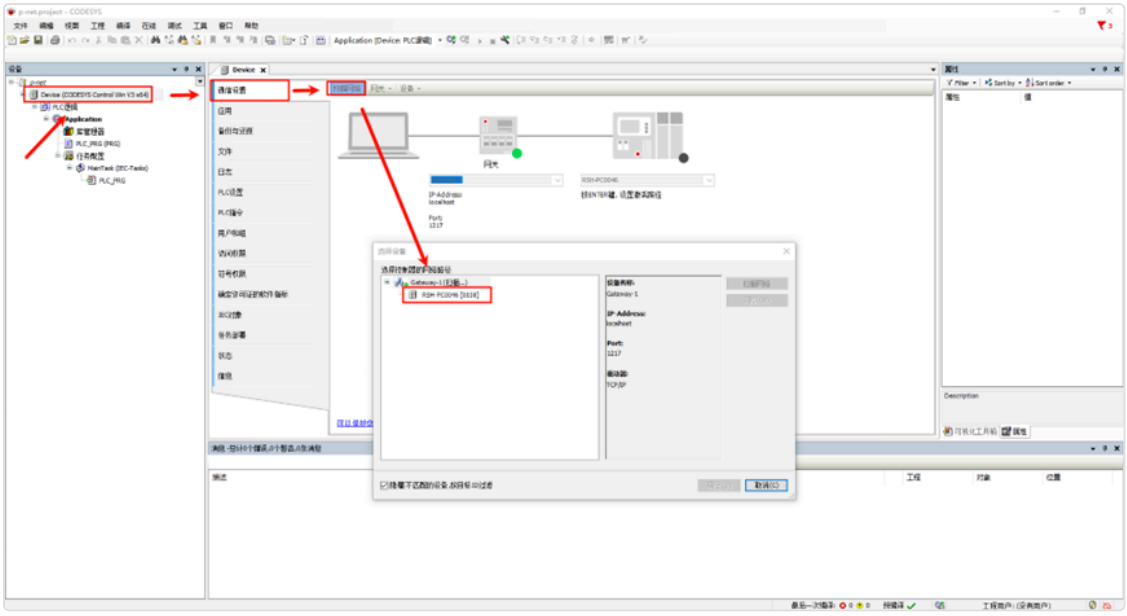
Starting Gateway and Soft PLC

Open the following two software components:

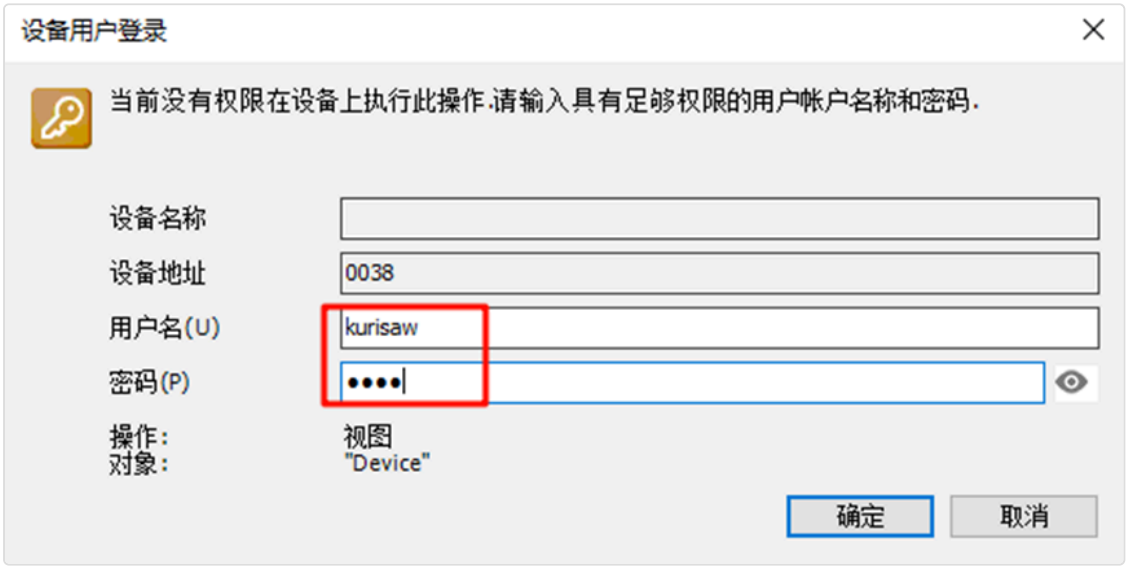
- **CODESYS Gateway V3** (right-click **Start Gateway**)
- **CODESYS Control Win V3 -x64 SysTray** (right-click **Start PLC**)



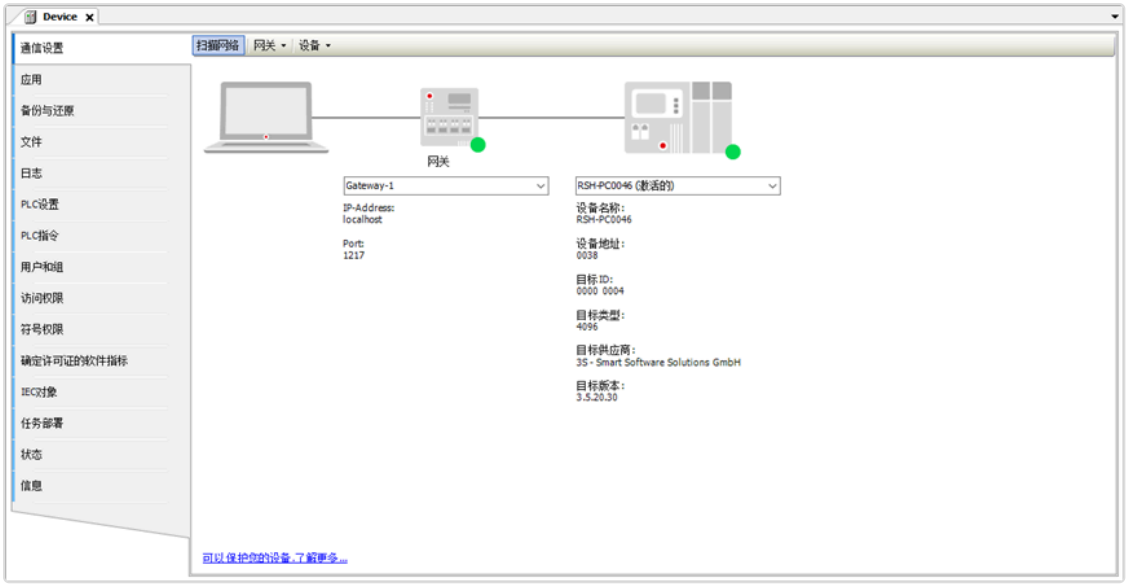
Back in CODESYS, double-click **Device(CODESYS Control Win V3 x64)** -> **Communication Settings** -> **Scan Network**:



In the user login window, configure the username and password (customizable):



Check if the gateway and soft PLC devices are online:



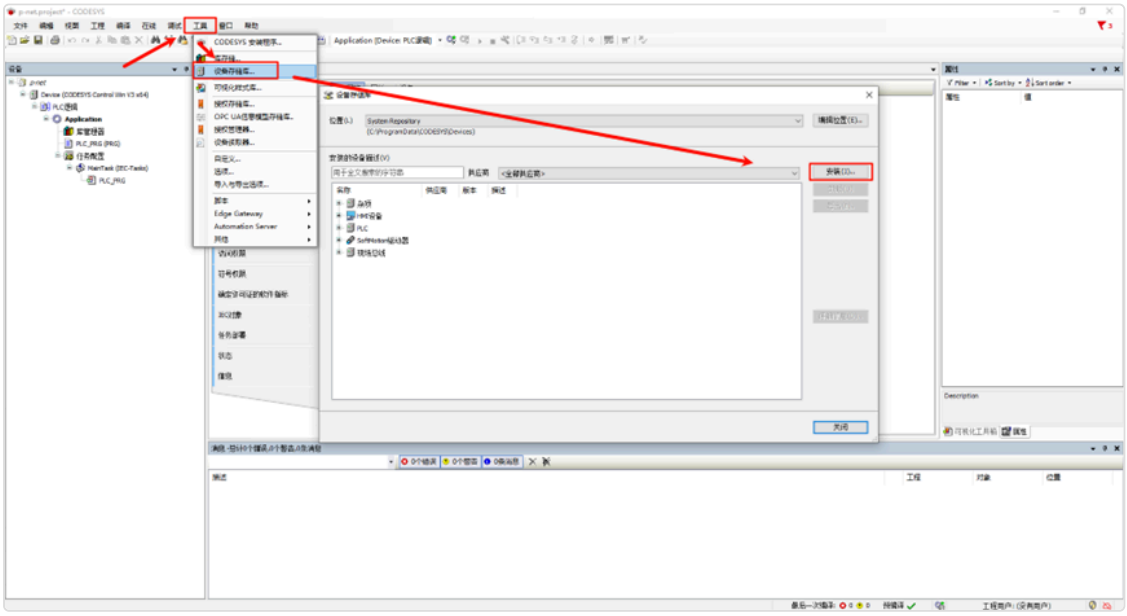
Adding PROFINET GSDML Files

GSD (Generic Station Description) files are used for PROFIBUS DP and PROFINET IO communication, describing parameters, diagnostic data, and user-defined data between the PLC and I/O modules.

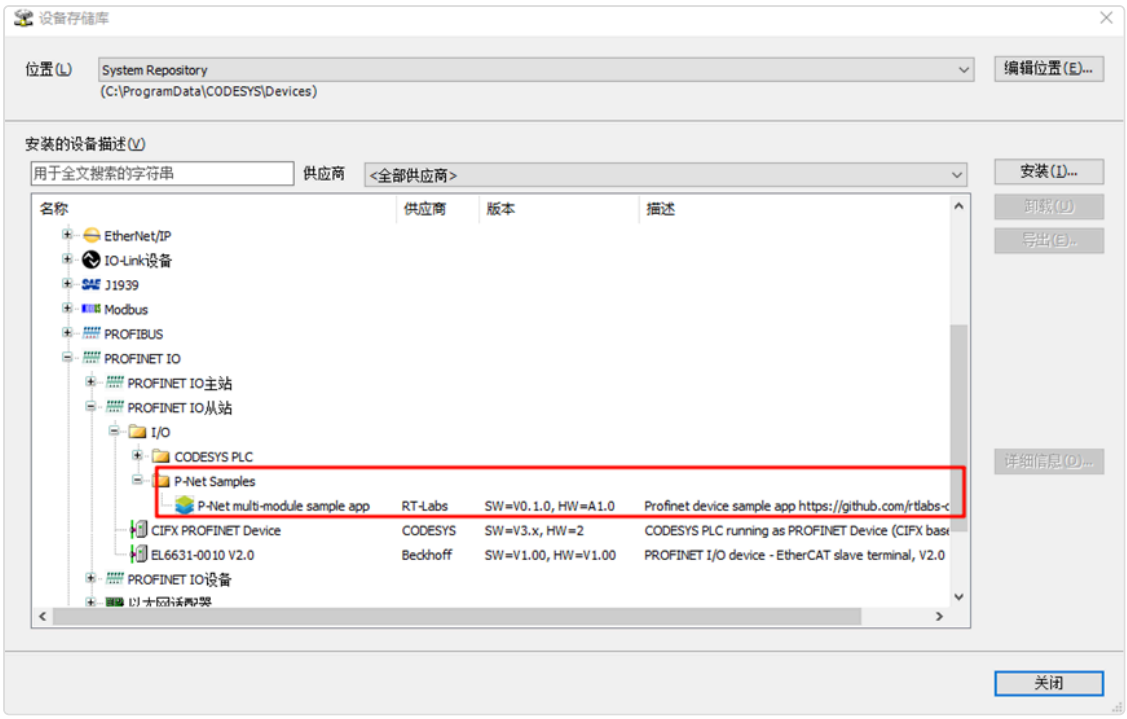
The GSDML file for this project is located at:

- `..\src\ports\rtthread\pn_dev`

Install the GSDML file from the above path: **GSDML-V2.4-RT-Labs-P-Net-Sample-App-20220324.xml**.

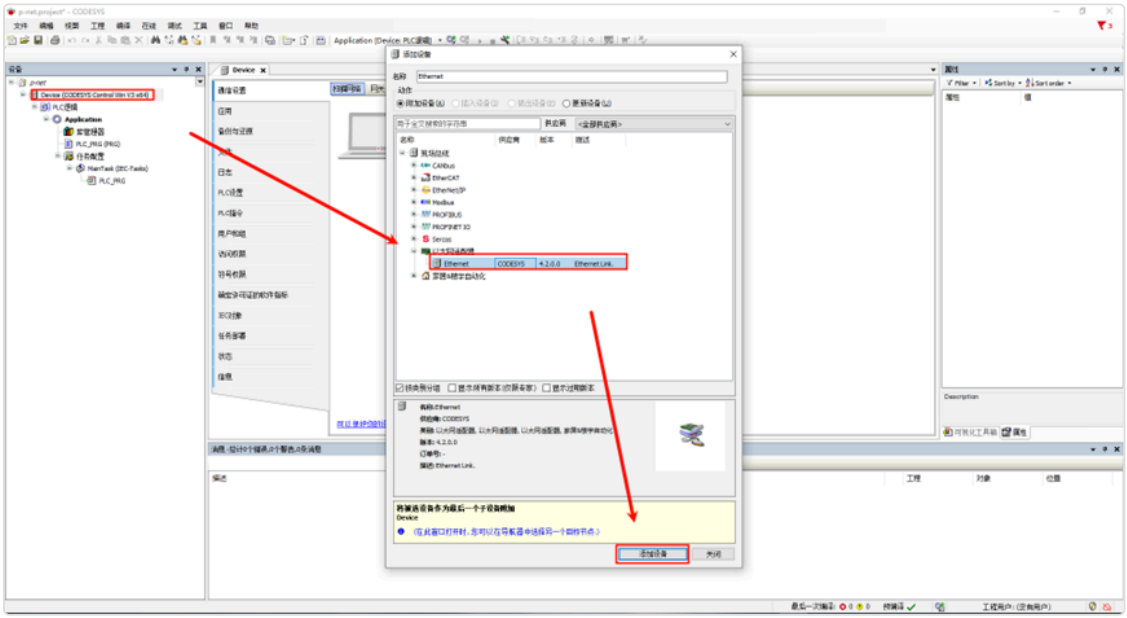


After installation, you should see the **P-Net slave description file**:

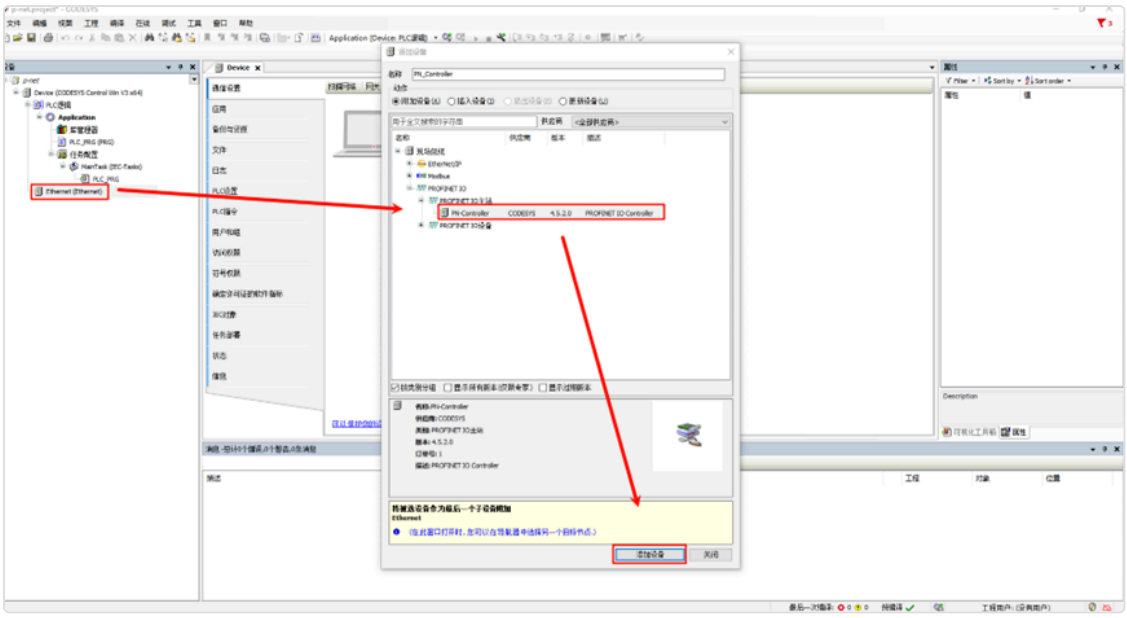


Adding Devices

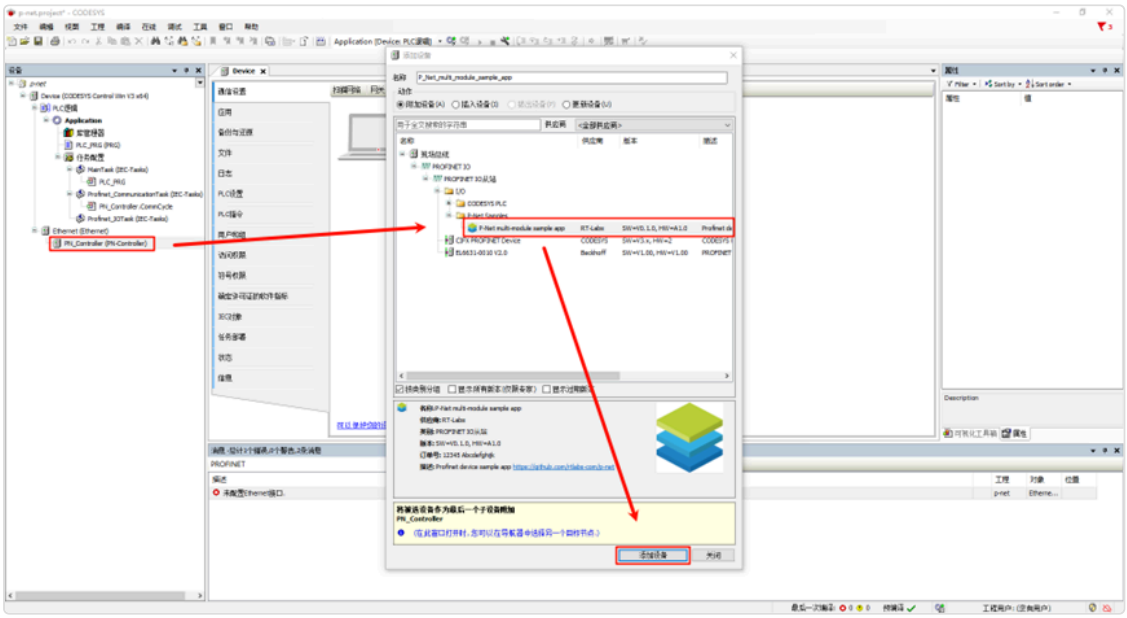
- **Adding Ethernet**: Right-click **Device** in the left navigation bar and select **Ethernet Adapter**:



- **Adding PROFINET IO Master**: Right-click **Ethernet** in the left navigation bar and select **PN-Controller**:

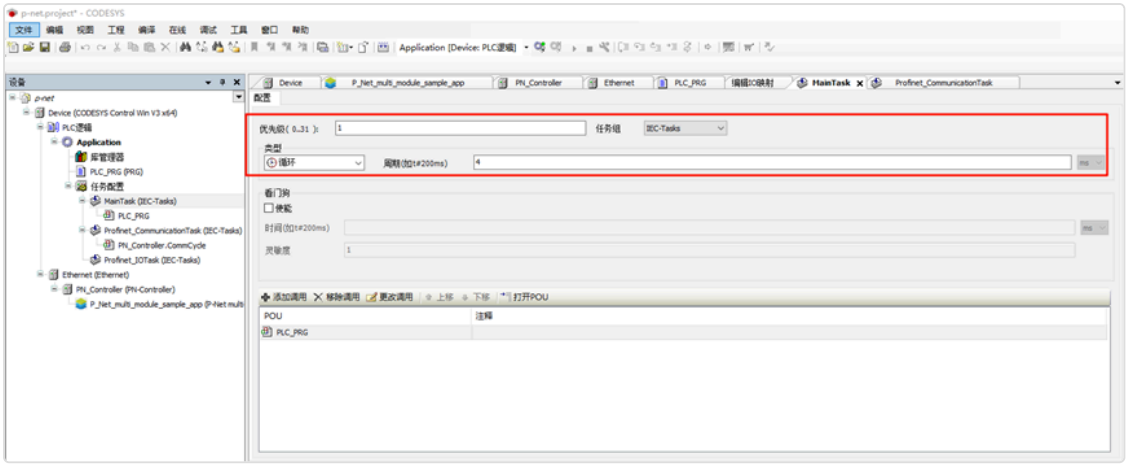


- **Adding PROFINET IO Slave**: Right-click **PN-Controller** in the left navigation bar and select **P-Net-multiple-module sample app**:

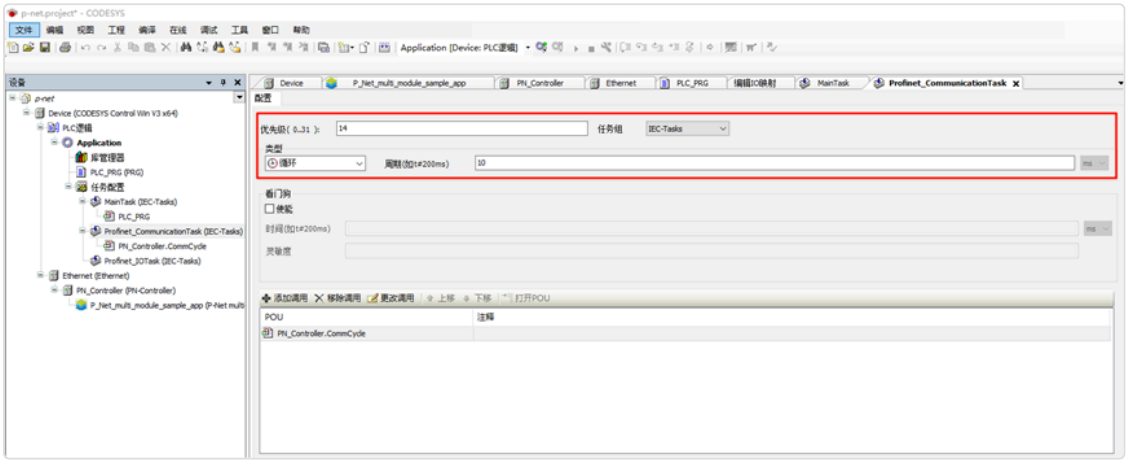


Configuring Tasks

- **Main Tasks Configuration**: Select **Application** -> **Task Configuration** in the left navigation bar, double-click **MainTask (IEC-Tasks)**, set the priority to 1, type to **Cyclic**, and cycle time to 4ms:

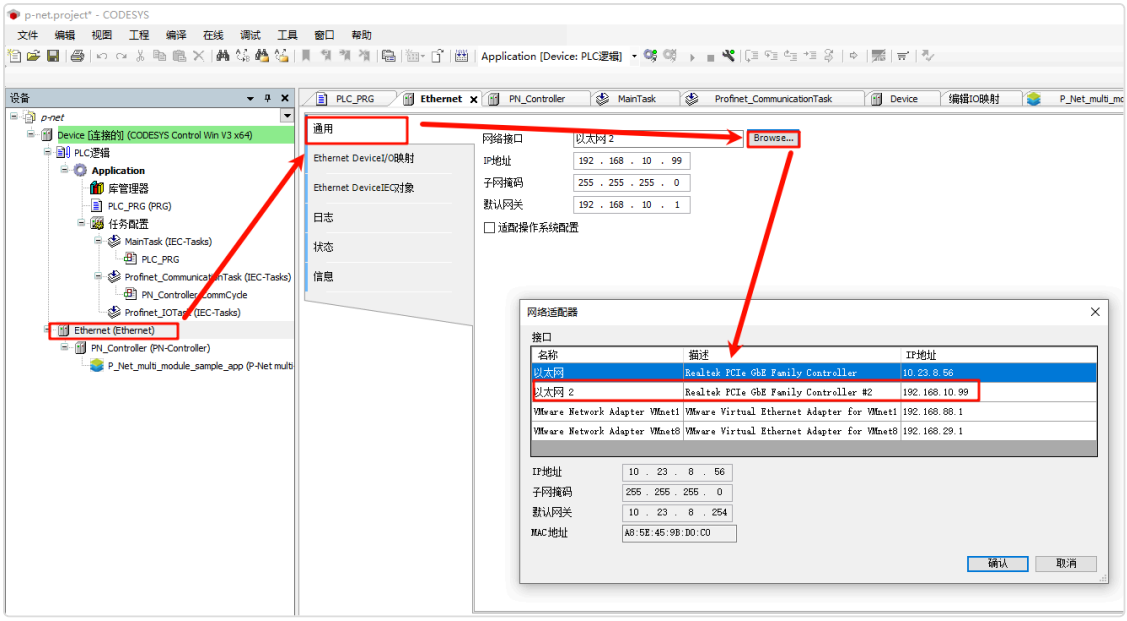


- **Profinet_CommunicationTask Configuration**: Double-click **Profinet_CommunicationTask (IEC-Tasks)**, set the priority to 14, type to **Cyclic**, and cycle time to 10ms:

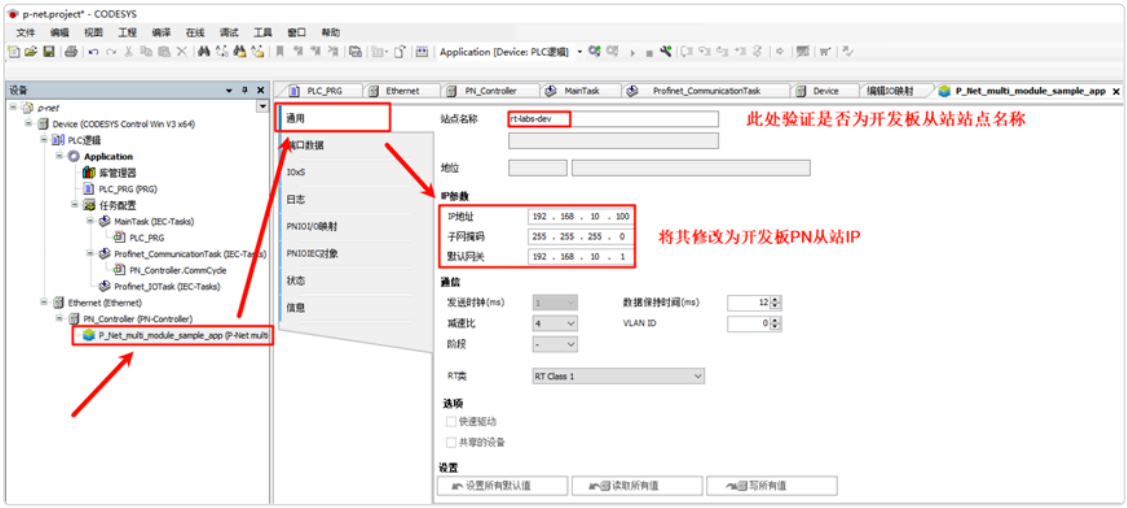


Network Configuration

- **Ethernet Configuration**: Double-click **Ethernet (Ethernet)** in the left navigation bar -> **General**, and modify the network interface to the one connected to the development board.



- **PN_Controller Configuration**: Double-click **PN_Controller (PN-Controller)** -> **General**, and modify the default slave IP parameters as needed.
- **P-Net Slave Network Configuration**: Double-click **P-Net-multiple-module sample app** -> **General**, and modify the IP parameters to match the development board's IP:

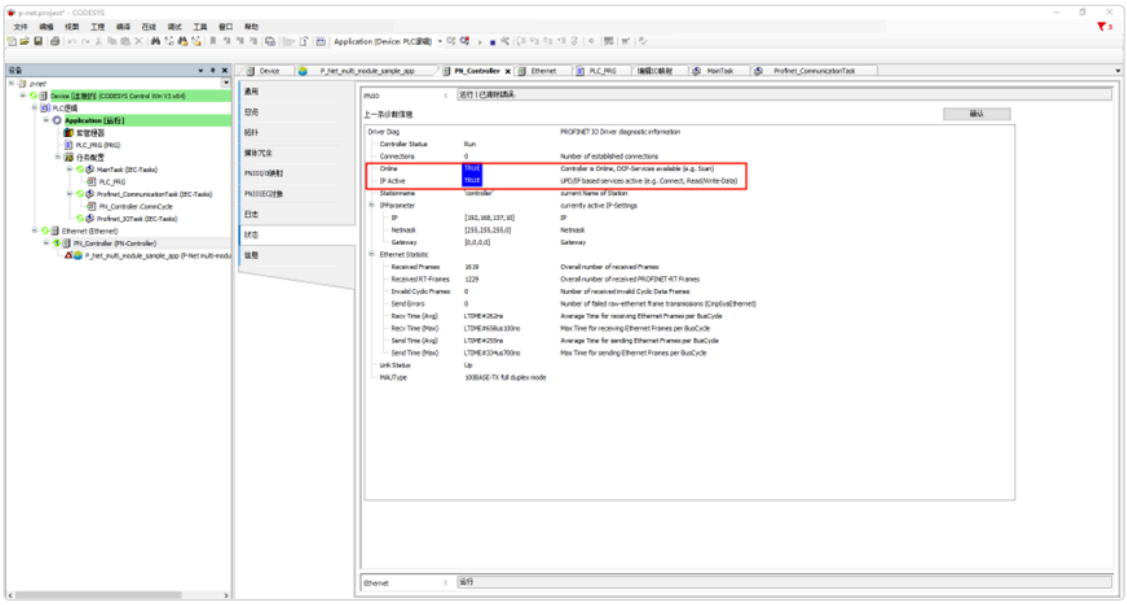


```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Dec 17 2024 13:52:54
 2006 - 2024 Copyright by RT-Thread team
lwIP-2.0.3 initialized!
[I/sal.skt] Socket Abstraction Layer initialize success.

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          PNIO站点名称
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
```

Compile and Debug the Project

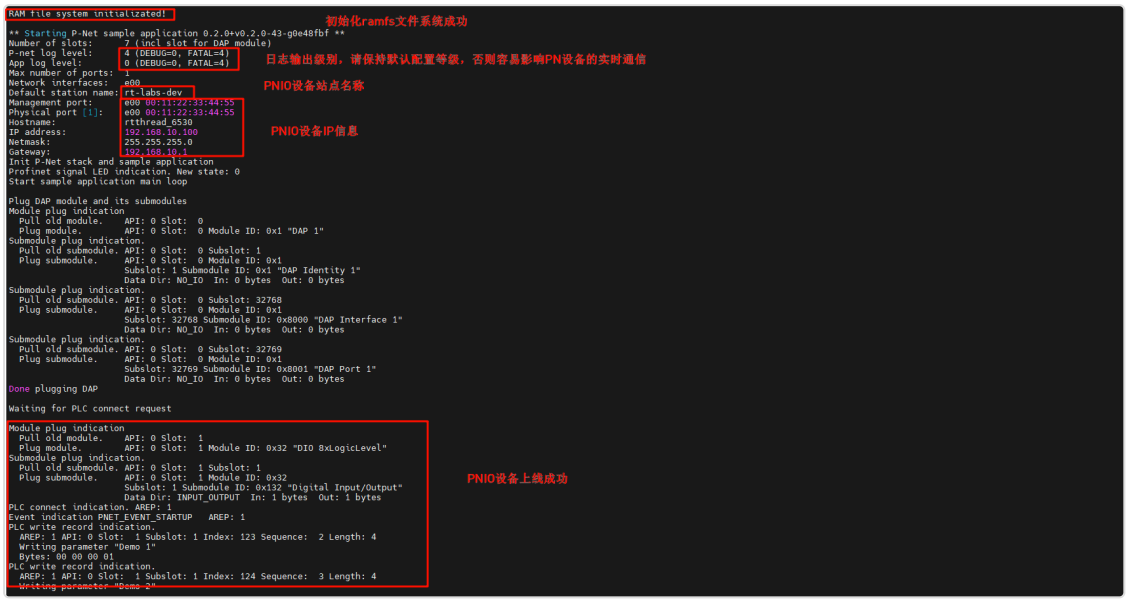
- Step 1: From the navigation bar, select **Compile** -> **Generate Code**
- Step 2: Select **Online** -> **Login**
- Step 3: Click **Debug** -> **Start**

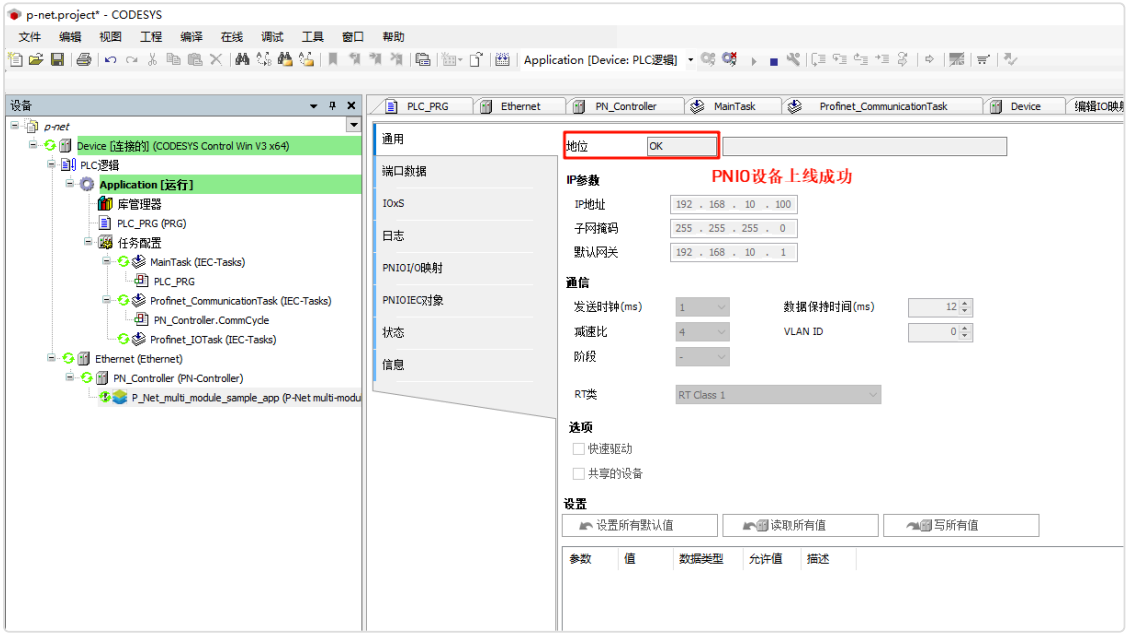
You should see the PN master successfully online:



profinet starts from the station application

After the development board is powered on, once the NIC link up is detected, the secondary PN station is automatically started:



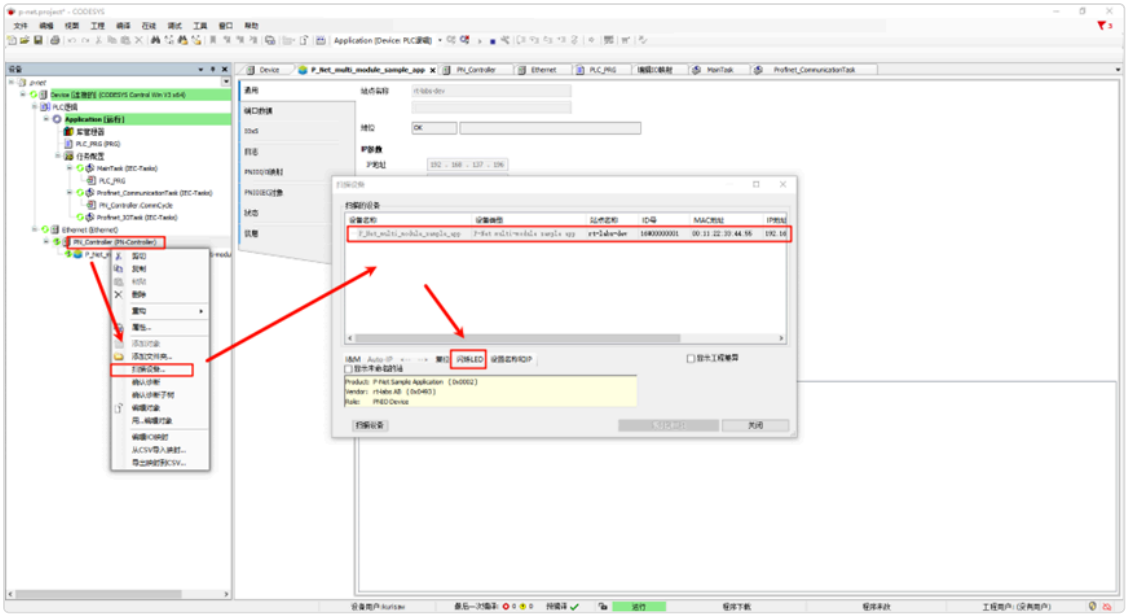


PROFINET Slave Application Startup

Here we use the CODESYS software to test the interaction between the PN master and slave stations.

LED Blinking

Back in the CODESYS software, in the left navigation panel, select `PN_Controller`, right-click and scan the devices. After clicking on the device name, click on "Blink LED":

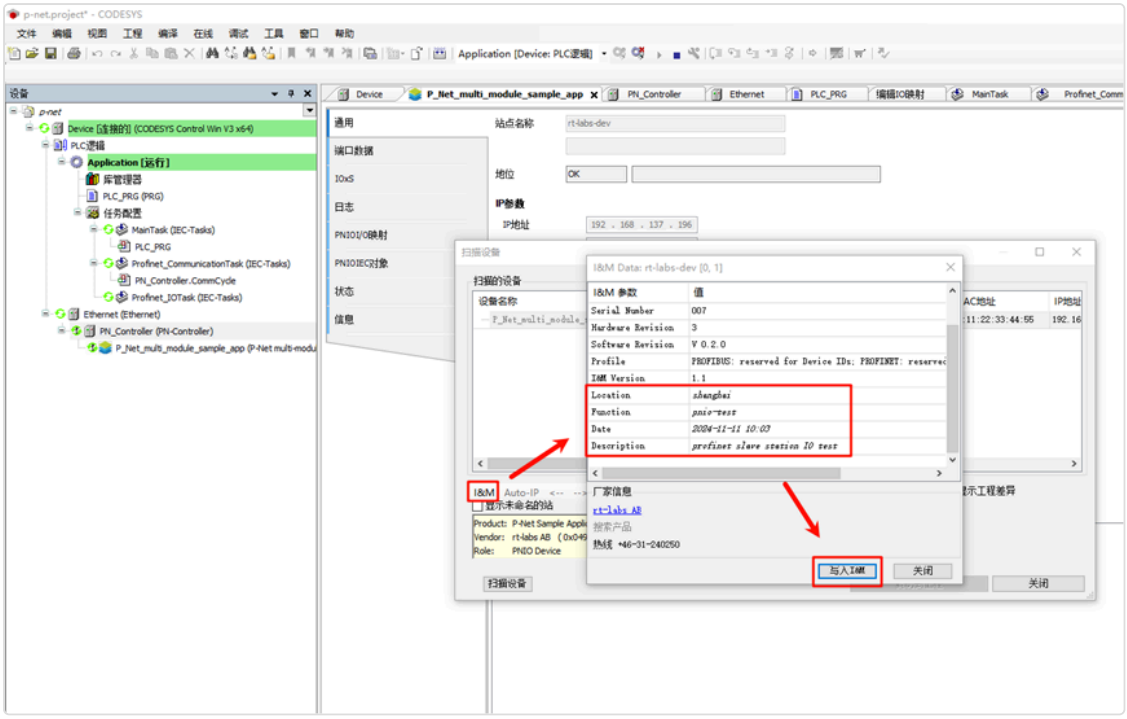


At this point, the development board (PN slave I/O) will display log outputs, accompanied by the onboard User LED blinking:

```
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
```

Modifying Slave I&M (Identification and Maintenance) Data

In the same device scanning interface, click on the I&M section in the lower left corner, modify the information, and write it to the I&M:



At the same time, PNIO will update the slave station configuration:

```
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
```

We can click to view the I&M again, and we will see that the I&M data has been successfully modified!

PLC Programming and PNIO Control

First, we click on the left panel under Device -> PLC Logic -> Application -> PLC_PRG (PRG), and use ST language to program the variable and program code:

- **Variable Definition**: These variables define the input state of the button (in_pin_button_LED), the output state of the LED (out_pin_LED), and the state variable controlling whether the LED should blink (flashing). The oscillator state (oscillator_state) and oscillator cycle counter (oscillator_cycles) are used to achieve a timed blinking effect.

```
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
```

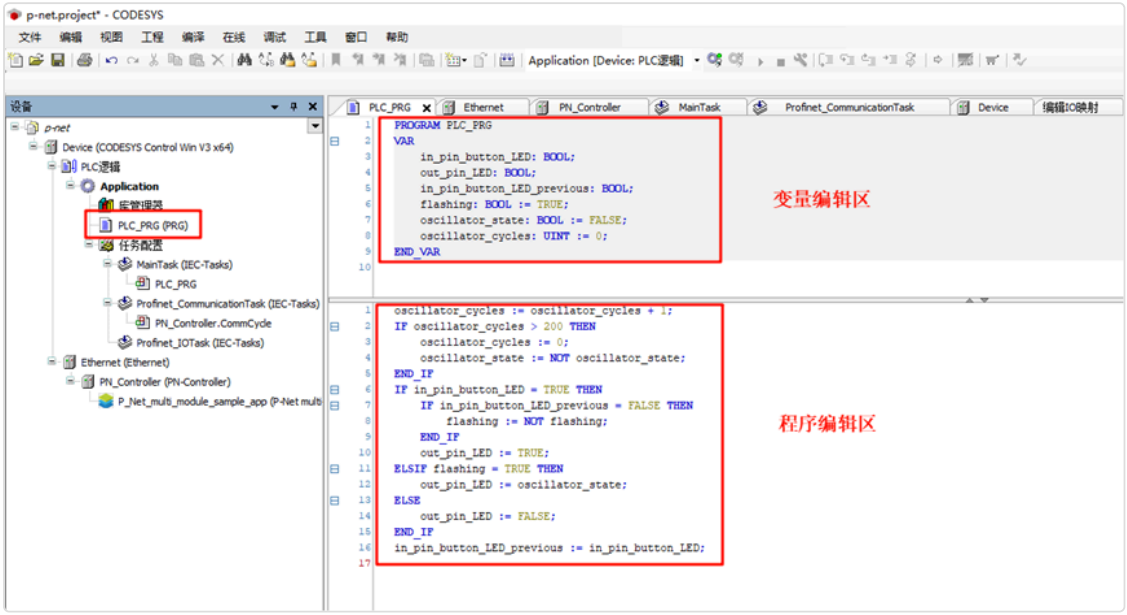
- **Program Definition**:

1. First, in each cycle, the `oscillator_cycles` increases by 1. When the counter exceeds 200, the counter is reset, and the `oscillator_state` is toggled (TRUE or FALSE), achieving a periodic change.
2. If the button is pressed (in_pin_button_LED is TRUE), and the button state in the previous cycle was FALSE, the `flashing` state is toggled. That is, every time the button is pressed, the LED blinking state is toggled.
3. If `flashing` is TRUE, the LED will blink according to the oscillator state (`oscillator_state`). If `flashing` is FALSE, the LED will turn off directly.
4. At the end of each cycle, the current button state is saved in `in_pin_button_LED_previous` to check the button press event in the next cycle.

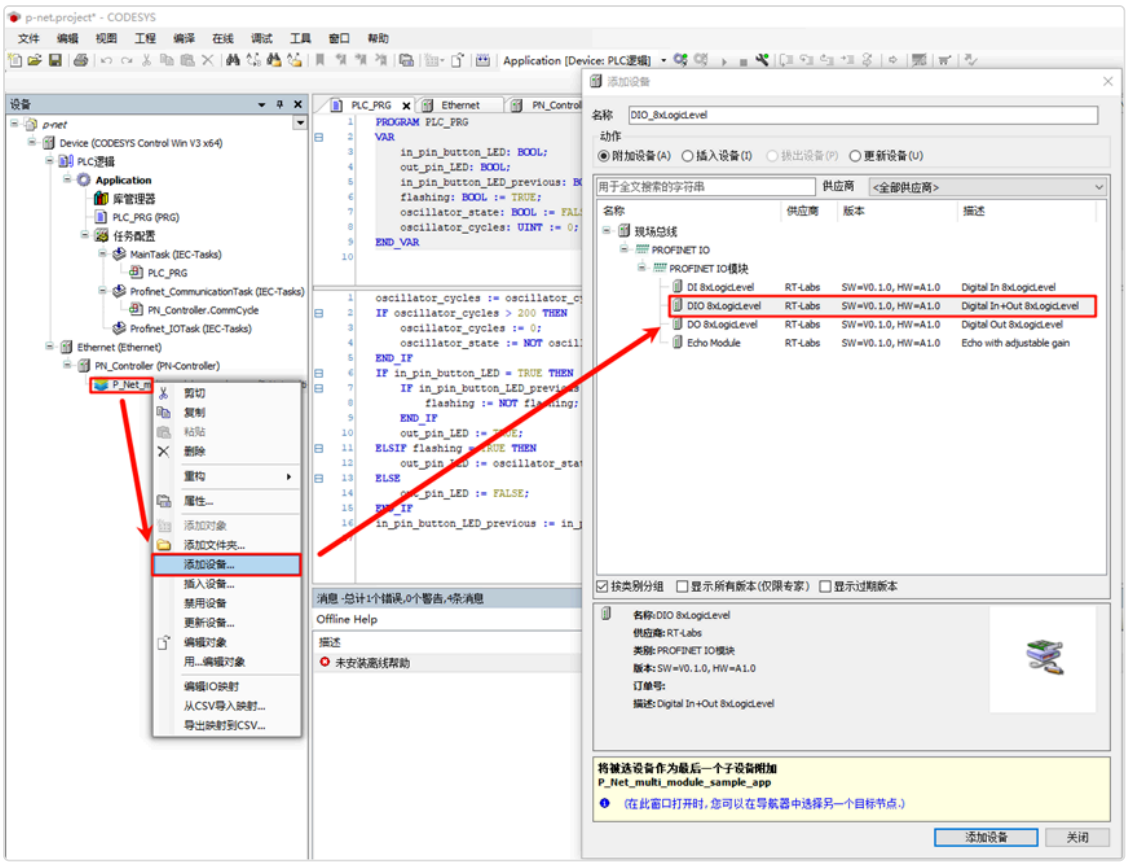
```
oscillator_cycles := oscillator_cycles + 1;
IF oscillator_cycles > 200 THEN
    oscillator_cycles := 0;
    oscillator_state := NOT oscillator_state;
END_IF
IF in_pin_button_LED = TRUE THEN
```

```
    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;
```

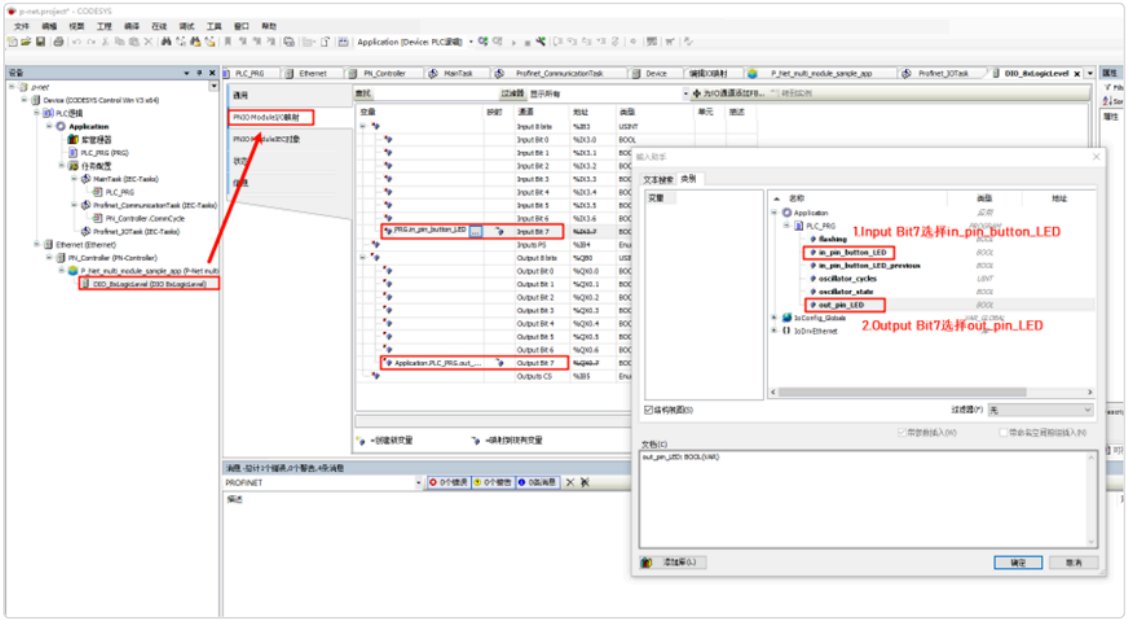
The configuration in the project is shown in the image below:



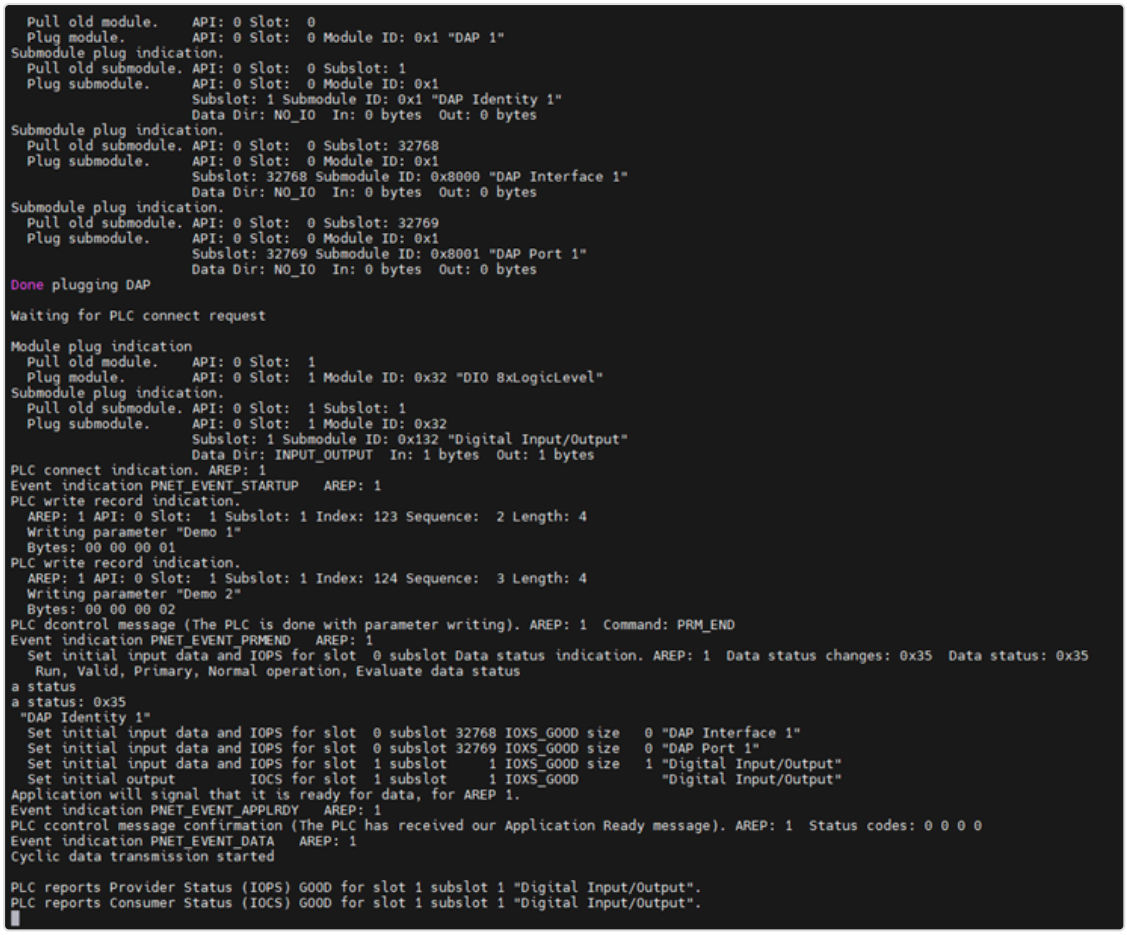
Next, we need to add a built-in IO module. Right-click on `P_Net_multi_module_sample_app` and add an IO module (DIO 8xLogicLevel), as shown in the image below:



Then, double-click on the `DIO_8xLogicLevel` node, select the `PNIO Module I/O Mapping`, edit Input Bit 7 and Output Bit 7, and bind the PLC variables:



Next, click on the **Build -> Generate Code** in the navigation bar, then select **Online -> Login**, and run to observe the behavior:



Next, go back to CODESYS, double-click **Device -> PLC Logic -> Application** and open the `PLC_PRG (PRG)`. At this point, you can dynamically observe the program's running state. For example, if you press and hold the KEY0 on the EtherKit development board, you will find that both `in_pin_button_LED` and `in_pin_button_LED_previous` are FALSE. When you release KEY0, you will see that the `flashing` value toggles once.