# EtherKit SDK

开发文档

*EtherKit 开发板是 RT-Thread 推出基于瑞萨 Cortex-R52 架构 R9A07G084 芯片，为工程师们提供了一个灵活、全面的开发平台，助力开发者在EtherCAT工业以太网领域获得更深层次的体验。*

Version 1.0.0 | 中文 | 2025年10月23日

www.rt-thread.org

Copyright (c) 2006-2025, RT-Thread Development Team

# 目录


- 1. 快速上手
  - 1.1. EtherKit 开发板 BSP 说明
- 2. 基础篇
  - 2.1. 按键中断使用说明
  - 2.2. RTC及Alarm 使用说明
  - 2.3. RGB 使用说明
- 3. 驱动篇
  - 3.1. ADC 驱动例程
  - 3.2. CANFD 驱动例程
  - 3.3. Ethernet 驱动示例
  - 3.4. GPT 驱动例程
  - 3.5. HyperRAM 驱动例程
  - 3.6. IIC EEROM 驱动例程
  - 3.7. RS485 驱动例程
  - 3.8. SCI_SPI 驱动例程
  - 3.9. WDT 驱动例程
  - 3.10. USB-PCDC 驱动例程
  - 3.11. USB-PMSC 驱动例程
- 4. 组件篇
  - 4.1. Filesystem 例程
  - 4.2. MQTT 例程
  - 4.3. Netutils 例程
- 5. 工业协议篇
  - 5.1. CherryECAT 驱动示例
  - 5.2. EtherCAT CoE 例程
  - 5.3. EtherCAT EOE 例程
  - 5.4. Ethernet/IP 例程

- 5.5. Modbus-TCP/IP 例程
- 5.6. Modbus-UART 例程
- 5.7. PROFINET 例程

# 1. 快速上手

## 1.1. EtherKit 开发板 BSP 说明

**中文 | English**

### 简介

本文档为 RT-Thread EtherKit 开发板提供的 BSP (板级支持包) 说明。通过阅读快速上手章节开发者可以快速地上手该 BSP，将 RT-Thread 运行在开发板上。

主要内容如下：

- 开发板介绍
- BSP 快速上手指南

### 开发板介绍

基于瑞萨 RZ/N2L 开发的 EtherKit 开发板，通过灵活配置软件包和 IDE，对嵌入系统应用程序进行开发。

开发板正面外观如下图：

该开发板常用 **板载资源** 如下：

- MPU：R9A07G084M04GBG，最大工作频率 400MHz，Arm Cortex®-R52 内核，紧密耦合内存 128KB（带 ECC），内部 RAM 1.5 MB（带 ECC）
- 调试接口：板载 J-Link 接口
- 扩展接口：一个 PMOD 连接器

**更多详细资料及工具**

# 外设支持

本 BSP 目前对外设的支持情况如下：

| EtherCAT方案 | 支持情况 | EtherCAT方案 | 支持情况 |
| --- | --- | --- | --- |
| EtherCAT_IO | 支持 | EtherCAT_FOE | 支持 |

| **EtherCAT方案** | **支持情况** | **EtherCAT方案** | **支持情况** |
| --- | --- | --- | --- |
| EtherCAT_EOE | 支持 | EtherCAT_COE | 支持 |
| **PROFINET方案** | **支持情况** | **Ethernet/IP方案** | **支持情况** |
| P-Net（支持ProfiNET从站协议栈的开源评估软件包） | 支持 | EIP | 支持 |
| **片上外设** | **支持情况** | **组件** | **支持情况** |
| UART | 支持 | LWIP | 支持 |
| GPIO | 支持 | TCP/UDP | 支持 |
| HWIMER | 支持 | MQTT | 支持 |
| IIC | 支持 | TFTP | 支持 |
| WDT | 支持 | Modbus主从站协议 | 支持 |
| RTC | 支持 | | |
| ADC | 支持 | | |
| DAC | 支持 | | |
| SPI | 支持 | | |

# 使用说明

使用说明分为如下两个章节：

- 快速上手

本章节是为刚接触 RT-Thread 的新手准备的使用说明，遵循简单的步骤即可将 RT-Thread 操作系统运行在该开发板上，看到实验效果 。

- 进阶使用

本章节是为需要在 RT-Thread 操作系统上使用更多开发板资源的开发者准备的。通过使用 ENV 工具对 BSP 进行配置，可以开启更多板载资源，实现更多高级功能。

## 快速上手

本 BSP 目前提供 GCC/IAR 工程。下面以 IAR Embedded Workbench for Arm 开发环境为例，介绍如何将系统运行起来。

### 硬件连接

使用 USB 数据线连接开发板到 PC，使用 J-link 接口下载和 DEBUG 程序。

### 编译下载

- 进入 bsp 目录下，打开 ENV 使用命令 `scons --target=iar` 生成 IAR工程。
- 编译：双击 project.eww 文件，打开 IAR 工程，编译程序。
- 调试：IAR 左上方导航栏点击 `Project->Download and Debug` 下载并启动调试。

### 查看运行结果

下载程序成功之后，系统会自动运行并打印系统信息。

连接开发板对应串口到 PC , 在终端工具里打开相应的串口（115200-8-1-N），复位设备后，可以看到 RT-Thread 的输出信息。输入 help 命令可查看系统中支持的命令。

```
 \ | /
- 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 a iar project which mode is ram execution!
====================================================
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 >
```

### 应用入口函数

应用层的入口函数在 **src\hal_entry.c** 中 的 `void hal_entry(void)` 。用户编写的源文件可直接放在 src 目录下。

```
void hal_entry(void)
{
    rt_kprintf("\nHello RT-Thread!\n");
    rt_kprintf("==================================================
    rt_kprintf("This is a iar project which mode is ram executi
    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);
    }
}
```

## 进阶使用

### 资料及文档

- 开发板官网主页
- 开发板数据手册
- 开发板硬件手册
- RZ/N2L MCU 快速入门指南
- RZ/N2L Easy Download Guide
- Renesas RZ/N2L Group

### FSP 配置

需要修改瑞萨的 BSP 外设配置或添加新的外设端口，需要用到瑞萨的 FSP 配置工具。请务必按照如下步骤完成配置。配置中有任何问题可到RT-Thread 社区论坛中提问。

1. 下载灵活配置软件包 (FSP) | Renesas，请使用 FSP 2.0.0 版本

2. 如何将 ”**EtherKit板级支持包**“添加到 FSP 中，请参考文档如何导入板级支持包
3. 请参考文档：RA系列使用FSP配置外设驱动。

**ENV 配置**

- 如何使用 ENV 工具：RT-Thread env 工具用户手册

此 BSP 默认只开启了 UART0 的功能，如果需使用更多高级功能例如组件、软件包等，需要利用 ENV 工具进行配置。

步骤如下：

1. 在 bsp 下打开 env 工具。
2. 输入 `menuconfig` 命令配置工程，配置好之后保存退出。
3. 输入 `pkgs --update` 命令更新软件包。
4. 输入 `scons --target=iar` 命令重新生成工程。

# 联系人信息

在使用过程中若您有任何的想法和建议，建议您通过以下方式来联系到我们 RT-Thread 社区论坛

# 贡献代码

如果您对 EtherKit 感兴趣，并且有一些好玩的项目愿意与大家分享的话欢迎给我们贡献代码，您可以参考 如何向 RT-Thread 代码贡献。

# 2. 基础篇

## 2.1. 按键中断使用说明

**中文 | English**

## 简介

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

## 硬件说明

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

KEY 在开发板中的位置如下图所示：

## FSP配置

首先下载官方FSP代码生成工具：

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

安装成功之后我们双击eclipse下的rasc.exe，并依次根据下图打开工程配置文件configuration.xml：

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

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

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

# 示例代码说明

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

KEY1(LEFT) 、KEY2(RIGHT)对应的单片机引脚定义如下:

```
/* 配置 key irq 引脚 */
```

```
#define IRQ_TEST_PIN1 BSP_IO_PORT_14_PIN_2
#define IRQ_TEST_PIN2 BSP_IO_PORT_16_PIN_3
```

LED灯的单片机引脚定义如下:

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

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

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

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

```
    }
}
```

## 编译&下载

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

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

## 运行效果

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

```
 \ | /
- 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及Alarm 使用说明

**中文 | English**

# 简介

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

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

# 硬件说明

本例程使用的RTC 设备依赖于LSE 时钟，此外无需过多连接。

# FSP配置说明

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

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

# RT-Thread Settings配置

打开RT-Thread Settings，找到硬件选项，使能RTC；

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

## 示例代码说明

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

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

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

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

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

    /* 设置时间 */
    ret = set_time(15, 00, 00);
    if (ret != RT_EOK)
```

```
    {
        rt_kprintf("set RTC time failed\n");
    }

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

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

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

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

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

    if (alarm != RT_NULL)
        return;

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

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

    alarm = rt_alarm_create(user_alarm_callback, &setup);
    if(RT_NULL != alarm)
    {
        rt_alarm_start(alarm);
    }
}
```

```
/* export msh cmd */
MSH_CMD_EXPORT(alarm_sample,alarm sample);
```

## 编译&下载

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

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

## 运行效果

按下复位按键重启开发板，可以看到板子上会打印如下信息：

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

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

# 2.3. RGB 使用说明

**中文 | English**

## 简介

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

## 硬件说明

image-20241126095713622

image-20241126095728227

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

LED在开发板中的位置如下图所示：

image-20241126095740420

## 软件说明

本例程的源码位于/projects/etherkit_blink_led.RGB-LED对应的单片机引脚定义及RGB变换源码可以通过查阅 src/hal_data.c中。

```
/* 配置 LED 灯引脚 */
#define LED_PIN_R    BSP_IO_PORT_14_PIN_3 /* Onboard RED LED 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
    {
        /* 获得组编号 */
        group_current = count % group_num;
```

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

        /* 输出 LOG 信息 */
        LOG_D("group: %d | red led [%-3.3s] | | blue led [%-3.3
            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++;

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

## 编译&下载

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

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

## 运行效果

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

image-20241126095753974

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

```
[D/main] group: 0 | red led [OFF] | | blue led [OFF] | | green
[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. 驱动篇

## 3.1. ADC 驱动例程

**中文 | English**

## 简介

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

## 硬件说明

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

## FSP配置说明

- 第一步：打开FSP导入xml配置文件；（或者直接点击Rtthread Studio 的FSP链接文件）；
- 第二步：新建r_adc Stack 配置adc设备以及所用通道;

- 第三步：保存并点击Generate Project；生成的代码保存到hal_data.c中；

# env配置

使用env工具打开adc0的外设

# 工程实例说明

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

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

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

```
static int adc_vol_sample()
{
    rt_adc_device_t adc_dev;
    rt_uint32_t value, vol;
    rt_err_t ret = RT_EOK;
    /* 查找设备 */
    adc_dev = (rt_adc_device_t)rt_device_find(ADC_DEV_NAME);
    if (adc_dev == RT_NULL)
    {
        rt_kprintf("adc sample run failed! can't find %s device
        return RT_ERROR;
    }
    /* 使能设备 */
    ret = rt_adc_enable(adc_dev, ADC_DEV_CHANNEL);
    /* 读取采样值 */
    value = rt_adc_read(adc_dev, ADC_DEV_CHANNEL);
    rt_kprintf("the value is :%d \n", value);
    /* 转换为对应电压值 */
    vol = value * REFER_VOLTAGE / CONVERT_BITS;
    rt_kprintf("the voltage is :%d.%02d \n", vol / 100, vol % 1
    /* 关闭通道 */
    ret = rt_adc_disable(adc_dev, ADC_DEV_CHANNEL);
    return ret;
}
```

示例中While循环每隔1000ms调用一次adc_vol_sample;

## 编译&下载

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

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

## 运行效果

使用adc0 的0通道采集1.8v电压效果如下：

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

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

## 注意事项

> *R9A07G084M08GBG 芯片的ADC采集电压耐压为1.8v!*

# 3.2. CANFD 驱动例程

**中文 | English**

## 简介

本例程主要介绍了如何在EtherKit上使用canfd设备；

## 硬件说明

## 软件说明

### FSP配置说明

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

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

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

接下来我们需要为CANFD分别设置通道、中断回调函数及过滤器数组；

对CANFD的引脚进行配置和使能；

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

## RT-Thread Settings配置

打开RT-Thread Settings，使能canfd配置；

## 工程示例说明

工程通过canfd0发送报文canfd1接收报文并将其使用串口打印

发送代码示例：

```
int can0_sample_send(int argc, char *argv[])
{
    rt_err_t res;
    rt_thread_t thread;
    char can_name[RT_NAME_MAX];
    if (argc == 2)
    {
        rt_strncpy(can_name, argv[1], RT_NAME_MAX);
    }
    else
    {
        rt_strncpy(can_name, CAN0_DEV_NAME, RT_NAME_MAX);
    }
    /* 查找 CAN 设备 */
    can0_dev = rt_device_find(can_name);
    if (!can0_dev)
    {
        rt_kprintf("find %s failed!\n", can_name);
        return RT_ERROR;
    }
    /* 以中断接收及发送方式打开 CAN 设备 */
    res = rt_device_open(can0_dev, RT_DEVICE_FLAG_INT_TX | RT_D
    RT_ASSERT(res == RT_EOK);
    /* 创建数据接收线程 */
    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;
}
/* 导出到 msh 命令列表中 */
MSH_CMD_EXPORT(can0_sample_send, can device sample);
int can1_sample_receive(int argc, char *argv[])
{
    rt_err_t res;
    rt_thread_t thread;
    char can_name[RT_NAME_MAX];
    if (argc == 2)
```

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

    /* 以中断接收及发送方式打开 CAN 设备 */
    res = rt_device_open(can1_dev, RT_DEVICE_FLAG_INT_TX | RT_D
    RT_ASSERT(res == RT_EOK);
    /* 创建数据接收线程 */
    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;
}
```

# 运行

## 编译&下载

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

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

## 运行效果

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

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

```
 \ | /
- 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 驱动示例

**中文 | English**

# 简介

本工程提供 ethernet 的基础功能，比如 `ping` 功能。

# 硬件连接

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

# FSP配置说明

打开工程配置文件configuration.xml，新增r_gamc Stack：

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

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

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

网卡引脚参数配置，选择操作模式为RGMII：

ETHER_GMAC配置：

## RT-Thread Studio配置

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

## 以太网IP实验现象

烧录代码到开发板，打开串口终端查看日志：

```
 \ | /
- 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

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
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 />
```

将网线插入路由器网口显示link up

输入ifconfig查看分配的IP

测试ping功能

## 3.4. GPT 驱动例程

**中文 | English**

# 简介

在我们具体的应用场合中往往都离不开timer的使用，本例程主要介绍了如何在EtherKit上使用GPT设备，包括基本定时器的使用和PWM的使用。

# FSP配置说明

FSP 分别配置使能GPT0为基本定时器模式，GPT5为PWM 模式：

| Property | Value |
|---|---|
| General | |
| Name | g_timer0 |
| Unit | 0 |
| Channel | 0 |
| Mode | Periodic |
| Period | 0x100000000 |
| Period Unit | Raw Counts |
| Output | |
| Input | |
| Interrupts | |
| Callback | timer0_callback |
| Overflow/Crest Interrupt Priority | Priority 12 |
| Capture A Interrupt Priority | Disabled |
| Capture B Interrupt Priority | Disabled |
| Trough Interrupt Priority | Disabled |
| Dead Time Error Interrupt Priority | Disabled |
| Extra Features | |
| ELC | |

选择模式

配置定时周期

配置周期单位

设置回调函数

设置中断优先级

| Property | Value |
|---|---|
| Common | |
| Parameter Checking | Default (BSP) |
| Pin Output Support | Enabled |
| Write Protect Enable | Disabled |
| Multiplex Interrupt | Disabled |
| Module g_timer5 Timer, General PWM (r_gpt) | |
| General | |
| Name | g_timer5 |
| Unit | 0 |
| Channel | 5 |
| Mode | PWM |
| Period | 20 |
| Period Unit | Milliseconds |
| Output | |
| Duty Cycle Percent (only applicable in PWM mode) | 50 |
| GTIOCA Output Enabled | True |
| GTIOCA Stop Level | Pin Level Low |
| GTIOCB Output Enabled | True |
| GTIOCB Stop Level | Pin Level Low |
| Input | |

使能pinout Support

配置定时器5

配置为PWM模式

周期

占空比配置

输出使能

并配置pins 使能GPT0 GPT5：

## RT-Thread Settings配置

在配置中打开timer0使能与PWM使能：

## 示例工程说明

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

```
int hwtimer_sample(void)
{
    rt_err_t ret = RT_EOK;
    rt_hwtimerval_t timeout_s;
    rt_device_t hw_dev = RT_NULL;
    rt_hwtimer_mode_t mode;
    rt_uint32_t freq = 400000000; /* 1Mhz */
    hw_dev = rt_device_find(HWTIMER_DEV_NAME);
    if (hw_dev == RT_NULL)
    {
        rt_kprintf("hwtimer sample run failed! can't find %s 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;  /* secend */
    timeout_s.usec = 0; /* microsecend */
    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);
```

每隔1s 中触发一次中断回调函数打印输出，下面是PWM 配置使能：

PWM相关宏定义：

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

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

配置PWM周期以及占空比：

```
static int pwm_sample(int argc, char *argv[])
{
    rt_uint32_t period, pulse, dir;
    period = 500000;    /* 周期为0.5ms，单位为纳秒ns */
    dir = 1;            /* PWM脉冲宽度值的增减方向 */
    pulse = 100000;          /* PWM脉冲宽度值，单位为纳秒ns */
    /* 查找设备 */
    pwm_dev = (struct rt_device_pwm *)rt_device_find(PWM_DEV_NA
    if (pwm_dev == RT_NULL)
    {
        rt_kprintf("pwm sample run failed! can't find %s device
        return RT_ERROR;
    }
    /* 设置PWM周期和脉冲宽度默认值 */
    rt_pwm_set(pwm_dev, PWM_DEV_CHANNEL, period, pulse);
    /* 使能设备 */
    rt_pwm_enable(pwm_dev, PWM_DEV_CHANNEL);
}
/* 导出到 msh 命令列表中 */
MSH_CMD_EXPORT(pwm_sample, pwm sample);
```

## 编译&下载

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

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

# 运行效果

在串口终端分别输入pwm_sample、hwtimer_sample查看具体效果；

每隔1s触发回调函数并打印输出：

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

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

使用逻辑分析仪量取Pwm 输出波形如下所示：

# 3.5. HyperRAM 驱动例程

**中文 | English**

## 简介

HyperRAM 是一种高性能、低延迟的外部存储器，主要用于嵌入式系统中，提供额外的存储容量和快速的数据访问。其主要功能和原理如下：

- 功能
- **高速数据存取**：HyperRAM 提供较高的数据传输速率，适合需要快速读写操作的应用，如图像处理和实时数据处理。
- **低功耗**：相较于其他类型的存储器，HyperRAM 设计上注重低功耗，适合电池供电的嵌入式设备。
- **扩展存储**：HyperRAM 可以为微控制器或其他处理器提供额外的存储空间，解决内置RAM不足的问题。
- **简化设计**：HyperRAM 通常使用简单的接口，简化了硬件设计和系统集成。
- 原理
- **串行接口**：HyperRAM 通常采用 SPI 或类似的串行接口进行通信，这使得连接和数据传输变得高效。
- **动态随机存取存储器（DRAM）**：HyperRAM 基于 DRAM 技术，使用电容器存储数据，具有较高的密度和存储能力。
- **内存控制器**：通过内存控制器，微控制器能够管理与 HyperRAM 的数据传输，控制读写操作。
- **页面模式**：HyperRAM 支持页面模式操作，这允许更高效的数据传输，减少访问延迟。

在嵌入式系统中，HyperRAM 的应用广泛，尤其是在对存储速度和能效有较高要求的场合。

本例程主要介绍了如何在EtherKit上使用 Hyperram 驱动进行读写测试。

# 硬件说明

# 软件说明

本例程的源码驱动位于 ../library/HAL_Drivers/drv_hyperram.c 中：

```
/*
 * 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 */
```

## 编译&下载

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

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

## 运行效果

按下复位按键重启开发板，观察开发板终端日志。

```
[D/drv.hyper] psram init success, mapped at 0x44000000, size is 33554432 bytes, data width is 16

 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Feb  8 2025 10:08:26
 2006 - 2024 Copyright by RT-Thread team

Hello RT-Thread!
===========================================================================
This example project is an driver hyperram routine!

>>> Notes:
    Please execute the 'psram_test' command to test the hyperram function.
===========================================================================
```

执行 psram_test 指令开始 hyperram 读写测试：

```
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 EEROM 驱动例程

**中文 | English**

## 简介

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

## 硬件说明

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

## FSP配置说明

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

# RT-Thread Settings 配置

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

# 示例工程说明

基于AT24C16的驱动软件包实现对EEROM的0x00,0x20地址写入与读出；

```
#ifdef PKG_USING_AT24CXX
#include "at24cxx.h"
#define EEPROM_I2C_NAME "i2c0"
static at24cxx_device_t at24c02_dev;
static void eeprom_test(void)
{
    char str1[] = "test string-hello rtthread\n";
    char str2[] = "test string-rzt2m eeprom testcase\n";
    uint8_t read_buffer1[50];
    uint8_t read_buffer2[50];
```

```
    at24c02_dev = at24cxx_init(EEPROM_I2C_NAME, 0x0);
    if (at24c02_dev == RT_NULL)
    {
        rt_kprintf("eeprom init failed\n");
        return;
    }
    rt_memset(read_buffer1, 0x0, sizeof(read_buffer1));
    rt_memset(read_buffer2, 0x0, sizeof(read_buffer2));
    at24cxx_write(at24c02_dev, 0x0, (uint8_t *)str1, (sizeof(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
```

## 编译&下载

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

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

## 运行效果

串口终端输入 eeprom_test指令：

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Nov 25 2024 14:41:57
 2006 - 2024 Copyright by RT-Thread team
[I/I2C] I2C bus [i2c0] registered

Hello RT-Thread!
==================================================
This example project is an driver iic routine!
==================================================
msh >eep
eeprom_test
msh >eeprom_test
write eeprom data to 0x0: test string-hello rtthread

read eeprom data from 0x0: test string-hello rtthread

write eeprom data to 0x20: test string-rzt2m eeprom testcase

read eeprom data from 0x20: test string-rzt2m eeprom testcase

eeprom test success
msh >
```

## 3.7. RS485 驱动例程

**中文 | English**

# 简介

本例程主要介绍了如何在EtherKit上使用RS485设备;

# 硬件说明

# FSP配置说明

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

# 工程示例说明

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

# 编译&下载

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

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

## 运行效果

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

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Apr 21 2025 15:04:42
 2006 - 2024 Copyright by RT-Thread team

Hello RT-Thread!
===================================================
This example project is an driver rs485 routine!
===================================================
msh >rs485_sample
msh >[I/rs485] Start RS485 communication driver. Please open the RS485 serial port and start the functional test.
[I/rs485] r
[I/rs485] s
[I/rs485] 4
[I/rs485] 8
[I/rs485] 5
[I/rs485]
[I/rs485] t
[I/rs485] e
[I/rs485] s
[I/rs485] t

msh >ps
thread           pri  status      sp     stack size max used left tick   error  tcb addr
---------------- ---  ------- ---------- ----------  ------  ---------- ------- ----------
rs485             25  suspend 0x000000a8 0x00000400    16%   0x00000009 EINTRPT 0x1003aa20
tshell            20  running 0x000000b0 0x00001000    13%   0x00000007 OK      0x10039830
sys workq         23  suspend 0x00000078 0x00000800    05%   0x0000000a OK      0x10038d38
tidle0            31  ready   0x00000068 0x00000400    10%   0x00000015 OK      0x100348b0
main              10  suspend 0x000000b0 0x00000800    12%   0x0000000d EINTRPT 0x10038410
msh >
```

# 3.8. SCI_SPI 驱动例程

**中文 | English**

## 简介

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

## 硬件说明

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

## FSP配置说明

打开FSP工具，新建 Stacks 并选择 r_sci_spi3：

Sci的片选引脚选择常低

## RT-Thread Settings配置

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

## 示例工程介绍

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

```
void spi_loop_test(void)
{
#define TEXT_NUMBER_SIZE 1024
#define SPI_BUS_NAME "sci3s"
#define SPI_NAME "spi30"
    static uint8_t sendbuf[TEXT_NUMBER_SIZE] = {0};
    static uint8_t readbuf[TEXT_NUMBER_SIZE] = {0};
    for (int i = 0; i < sizeof(readbuf); i++)
    {
        sendbuf[i] = i;
    }
    static struct rt_spi_device *spi_dev = RT_NULL;
    struct rt_spi_configuration cfg;
    rt_hw_sci_spi_device_attach(SPI_BUS_NAME, SPI_NAME, NULL);
    cfg.data_width = 8;
    cfg.mode = RT_SPI_MASTER | RT_SPI_MODE_0 | RT_SPI_MSB | RT_
    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");
}
```

## 编译&下载

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

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

## 运行效果

打开串口工具，可以看到通过spi回环测试中发送的数据与接收数据一致：

# 3.9. WDT 驱动例程

**中文 | English**

## 简介

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

## 硬件说明

无

## 软件说明

### FSP配置说明

打开FSP工具 新建Stacks选择r_wdt

### RT-Thread Settings配置

## 示例工程说明

通过在空闲函数中执行喂狗操作来保证我们的程序如期运转;

```
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);
    }
}
```

# 运行

## 编译&下载

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

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

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

### 运行效果

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

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

## 注意事项

暂无

## 引用参考

设备与驱动：WDT 设备

# 3.10. USB-PCDC 驱动例程

**中文 | English**

## 简介

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

## 硬件说明

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

## FSP配置说明

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

- 添加g_rm_block_media0；

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

- 下面配置USB引脚，找到USB_HS并使能：

## 构建配置

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

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

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

if rtconfig.PLATFORM in ['iccarm'] + 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/dri
                            cwd + '/fsp/src/r_usb_basic/src/hw/
```

```
                                cwd + '/fsp/src/r_usb_pcdc/src/inc'

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

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

## RT-Thread Settings配置

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

## 编译&下载

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

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

# 运行效果

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

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Nov 25 2024 16:06:26
 2006 - 2024 Copyright by RT-Thread team

Hello RT-Thread!
===================================================
This example project is an usb pcdc routine!
===================================================
msh >ps
thread           pri  status      sp     stack size max used left tick   error  tcb addr
---------------- ---  ------- ---------- ----------  ------  ---------- ------- ----------
PCD_TSK            0  suspend 0x000000c0 0x00001800    03%   0x00000001 EINTRPT 0x10042768
tshell            20  running 0x000000b0 0x00001000    13%   0x00000001 OK      0x10041440
usb_td            20  suspend 0x000000f0 0x00000800    19%   0x00000014 EINTRPT 0x10040948
sys_workq         23  suspend 0x00000078 0x00000800    05%   0x0000000a OK      0x10040080
tidle0            31  ready   0x00000048 0x00000400    10%   0x0000000e OK      0x1003d868
main              10  suspend 0x000000b0 0x00000800    12%   0x0000000d EINTRPT 0x1003f758
msh >
```

接着我们打开虚拟的串口设备，并测试字符输入：

```
9. Serial (COM)    11. Serial (COM)
EtherKit USB pcdc test
```

# 3.11. USB-PMSC 驱动例程

**中文 | English**

## 简介

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

## 硬件说明

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

## FSP配置说明

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

- 添加g_rm_block_media0；

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

- 下面配置USB引脚，找到USB_HS并使能：

## 构建配置

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

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

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

if rtconfig.PLATFORM in ['iccarm'] + 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')
```

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

## RT-Thread Settings配置

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

## 编译&下载

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

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

# 运行效果

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

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

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

# 4. 组件篇

## 4.1. Filesystem 例程

**中文 | English**

## 简介

FAL (Flash Abstraction Layer) Flash 抽象层，是对 Flash 及基于 Flash 的分区进行管理、操作的抽象层，对上层统一了 Flash 及 分区操作的 API (框架图如下所示)，并具有以下特性：

- 支持静态可配置的分区表，并可关联多个 Flash 设备；
- 分区表支持 **自动装载** 。避免在多固件项目，分区表被多次定义的问题；
- 代码精简，对操作系统 **无依赖** ，可运行于裸机平台，比如对资源有一定要求的 Bootloader；
- 统一的操作接口。保证了文件系统、OTA、NVM（例如：EasyFlash） 等对 Flash 有一定依赖的组件，底层 Flash 驱动的可重用性；
- 自带基于 Finsh/MSH 的测试命令，可以通过 Shell 按字节寻址的方式操作（读写擦） Flash 或分区，方便开发者进行调试、测试；

本示例中将使用 EtherKit 板载 GD Flash 结合 RT-Thread FAL 组件，使用 littlefs 构建文件系统。

## 硬件说明

## 软件说明

本例程的文件系统初始化源码位于 ../board/ports/filesystem/drv_filesystem.c 中：

```
/*
 * 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);
```

## 编译&下载

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

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

## 运行效果

按下复位按键重启开发板，观察开发板终端日志。

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Apr 21 2025 11:53:46
 2006 - 2024 Copyright by RT-Thread team
[D/FAL] (fal_flash_init:47) Flash device |           qspi0cs0_flash | addr: 0x60000000 | len: 0x00800000 | blk_size: 0x00000100 |initialized finish.
[I/FAL] ==================== FAL partition table ====================
[I/FAL] | name  | flash_dev      |   offset   |    length  |
[I/FAL] ------------------------------------------------------------
[I/FAL] | app   | qspi0cs0_flash | 0x00000000 | 0x00700000 |
[I/FAL] | param | qspi0cs0_flash | 0x00700000 | 0x00100000 |
[I/FAL] ============================================================
[I/FAL] RT-Thread Flash Abstraction Layer initialize success.
[I/FAL] The FAL MTD NOR device (param) created successfully
[I/drv.fs] mount lfs success

Hello RT-Thread!
==================================================
This is a iar project which mode is xspi0 execution!
==================================================
msh />
msh />
msh />
RT-Thread shell commands:
backtrace        - print backtrace of a thread
list             - list objects
version          - show RT-Thread version information
clear            - clear the terminal screen
free             - Show the memory usage in the system
ps               - List threads in the system
help             - RT-Thread shell help
tail             - print the last N - lines data of the given file
echo             - echo string to file
df               - disk free
umount           - Unmount the mountpoint
mount            - mount <device> <mountpoint> <fstype>
mkfs             - format disk with file system
mkdir            - Create the DIRECTORY.
pwd              - Print the name of the current working directory.
cd               - Change the shell working directory.
rm               - Remove(unlink) the FILE(s).
cat              - Concatenate FILE(s)
mv               - Rename SOURCE to DEST.
cp               - Copy SOURCE to DEST.
ls               - List information about the FILEs.
fal              - FAL (Flash Abstraction Layer) operate.
pin              - pin [option]
reboot           - Reboot System

msh />fal probe param param
Probed a flash partition | param | flash_dev: qspi0cs0_flash | offset: 7340032 | len: 1048576 |.
msh />fal bench 4096 yes
Erasing 1048576 bytes data, waiting...
Erase benchmark success, total time: 2.645S.
Writing 1048576 bytes data, waiting...
Write benchmark success, total time: 16.384S.
Reading 1048576 bytes data, waiting...
Read benchmark success, total time: 0.091S.
msh />
```

执行以下指令开始 FAL 读写测试：

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

## 4.2. MQTT 例程

**中文 | English**

# 简介

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

# 硬件说明

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

# FSP配置说明

打开工程配置文件configuration.xml，新增r_gamc Stack：

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

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

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

网卡引脚参数配置，选择操作模式为RGMII：

ETHER_GMAC配置：

# RT-Thread Studio配置

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

找到软件包界面，我们搜索kawaii-mqtt软件包，并使能SAL选项：

## 示例代码说明

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

```
static void sub_topic_handle1(void* client, message_data_t* msg
{
    (void) client;
    KAWAII_MQTT_LOG_I("----------------------------------------
    KAWAII_MQTT_LOG_I("%s:%d %s()...\ntopic: %s\nmessage:%s", _
    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);
```

## 编译&下载

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

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

## MQTTX配置

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

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

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

在下方功能框中编写订阅主题名称为sub5323，订阅信息按自己需求配置；

# 运行效果

打开串口工具，运行ka_mqtt命令后查看：

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

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

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

# 其他说明

> *MQTTX下载链接：https://packages.emqx.net/MQTTX/v1.9.6/MQTTX-Setup-1.9.6-x64.exe*

## 4.3. Netutils 例程

**中文 | English**

# 简介

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

# 硬件连接

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

# FSP配置说明

打开工程配置文件configuration.xml，新增r_gamc Stack：

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

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

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

网卡引脚参数配置，选择操作模式为RGMII：

ETHER_GMAC配置：

# RT-Thread Studio配置

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

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

## 以太网IP实验现象

烧录代码到开发板，打开串口终端查看日志：

## TFTP Server 发送测试

1. 安装 netutils-v1.3.3\tools 下的Tftpd64-4.60-setup软件

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

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

3. 然后打开安装好的Tftpd64-4.60软件

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

4. 点击Put后，会提示已经发送信息:

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

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

## TFTP接收测试

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

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

2. 可以验证下是否创建并写入成功:

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

3. 打开安装好的Tftpd64-4.60软件:

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

- 可以看到2.txt已经接受成功，内容也是开发板文件系统中的文件内容

# NTP联网校时

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

## NTP实验现象

烧录代码到开发板，打开串口终端查看日志：

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

```
msh />
msh />nt
ntp_sync
msh />ntp_sync
[I/ntp] Get local time from NTP server: Wed Apr 10 16:26:14 2024

msh />da
date
msh />date
local time: Wed Apr 10 16:26:15 2024
timestamps: 1712737575
timezone: UTC+08:00:00
msh />
```

# 5. 工业协议篇

## 5.1. CherryECAT 驱动示例

**中文 | English**

### 简介

本工程提供 cherryecat 的基础功能

### 硬件连接

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

### FSP配置说明

打开工程配置文件configuration.xml，新增r_gamc Stack：

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

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

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

网卡引脚参数配置，选择操作模式为RGMII：

ETHER_GMAC配置：

# RT-Thread Studio配置

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

# 运行效果

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Sep 30 2025 10:40:07
 2006 - 2024 Copyright by RT-Thread team

Hello RT-Thread!
===================================================
This example project is an ethernet routine!
===================================================
msh >[I/ec_master] Detect link up on ec-netdev0(main)
[I/ec_master] 1 slaves responding on ec-netdev0(main) device
[I/ec_master] Slaves state on ec-netdev0(main) device: PREOP
[I/ec_master] Rescanning bus...
[I/ec_slave] Scanning slave 0 on ec-netdev0(main)
[I/ec_slave] Slave 0 mbxprot support: COE
[I/ec_slave] Slave 0 parse eeprom success
[I/ec_slave] Slave 0 State changed to INIT
[I/ec_slave] Slave 0 State changed to PREOP
[I/ec_master] Bus scanning completed in 81 ms
[I/ec_master] Using slave 0 as DC reference clock

msh >ethercat master
Master0
  Phase: Idle
  Slaves: 1
  Ethernet net devices:
    Main: 00:11:22:33:44:55
      Link: UP
      Tx frames:   7257
      Tx bytes:    435548
      Rx frames:   7257
      Rx bytes:    435548
      Tx errors:   0
      Tx frame rate [1/s]:      202      196       90
      Tx rate [KByte/s]:       11.8     11.5      5.3
      Rx frame rate [1/s]:      202      196       90
      Rx rate [KByte/s]:       11.8     11.5      5.3
    Common:
      Tx frames:   7257
      Tx bytes:    435548
      Rx frames:   7257
      Rx bytes:    435548
      Lost frames: 0
      Tx frame rate [1/s]:      202      196       90
      Tx rate [KByte/s]:       11.8     11.5      5.3
      Rx frame rate [1/s]:      202      196       90
      Rx rate [KByte/s]:       11.8     11.5      5.3
      Loss rate [1/s]:            0        0        0
      Frame loss [%]:           0.0      0.0      0.0
msh >
```

```
msh >
msh >ec_start 2000
[I/ec_slave] Slave 0: Logical address 0x00000000, obyte 28, ibyte 32 expected working counter 3
msh >[I/ec_slave] Slave 0 State changed to INIT
[I/ec_slave] Slave 0 State changed to PREOP
[I/ec_slave] Slave 0 DC time diff: 8 ns
[I/ec_slave] Slave 0 State changed to SAFEOP
[I/ec_slave] Slave 0 State changed to OP
[I/ec_master] Slaves state on ec-netdev0(main) device: OP

msh >ethercat slaves -v
=== Master 0, Slave 0 ===
Device: ec-netdev0(main)
State: OP
Identity:
  Vendor Id:       0x00000766
  Product code:    0x00000911
  Revision number: 0x00000200
  Serial number:   0x00000000
DL information:
  FMMU bit operation: no
  Distributed clocks: yes, 64 bit
  DC system time transmission delay: 0 ns
Port  Type  Link  Loop    Signal  NextSlave  RxTime [ns]  Diff [ns]   NextDc [ns]
   0  MII   up    open    yes     -           2251932621          0           0
   1  MII   down  closed  no      -                    -          -           -
   2  MII   down  closed  no      -                    -          -           -
   3  N/A   down  closed  no      -                    -          -           -
Mailboxes:
  Bootstrap RX: 0x1000/128, TX: 0x1080/128
  Standard  RX: 0x1000/256, TX: 0x1400/256
  Supported protocols: COE
General:
  Group: Renesas RZ/N2 Group
  Image name:
  Order number: Renesas EtherCAT RZ/N2 CoE 2port
  Device name: Renesas EtherCAT RZ/N2 CoE 2port
  CoE details:
    Enable SDO: yes
    Enable SDO Info: yes
    Enable PDO Assign: yes
    Enable PDO Configuration: yes
    Enable Upload at startup: no
    Enable SDO complete access: yes
  Flags:
    Enable SafeOp: no
    Enable notLRW: no
  Current consumption: 0 mA

msh >
```

```
msh >
msh >
msh >ethercat pdos
=== Master 0, Slave 0 ===
SM0: physaddr 0x1000, length 0x0100, control 0x26, enable 1
SM1: physaddr 0x1400, length 0x0100, control 0x22, enable 1
SM2: physaddr 0x1800, length 0x001c, control 0x64, enable 1
        RxPDO 0x1601
                PDO entry 0x6040:0x00, bitlen 0x10
                PDO entry 0x6072:0x00, bitlen 0x10
                PDO entry 0x607a:0x00, bitlen 0x20
                PDO entry 0x60b0:0x00, bitlen 0x20
                PDO entry 0x60b1:0x00, bitlen 0x20
                PDO entry 0x60b2:0x00, bitlen 0x10
                PDO entry 0x60b8:0x00, bitlen 0x10
                PDO entry 0x60e0:0x00, bitlen 0x10
                PDO entry 0x60e0:0x00, bitlen 0x10
                PDO entry 0x60fe:0x01, bitlen 0x20
SM3: physaddr 0x1c00, length 0x0020, control 0x20, enable 1
        TxPDO 0x1a01
                PDO entry 0x603f:0x00, bitlen 0x10
                PDO entry 0x6041:0x00, bitlen 0x10
                PDO entry 0x6064:0x00, bitlen 0x20
                PDO entry 0x606c:0x00, bitlen 0x20
                PDO entry 0x6077:0x00, bitlen 0x10
                PDO entry 0x60b9:0x00, bitlen 0x10
                PDO entry 0x60ba:0x00, bitlen 0x20
                PDO entry 0x60bc:0x00, bitlen 0x20
                PDO entry 0x60f4:0x00, bitlen 0x20
                PDO entry 0x60fd:0x00, bitlen 0x20
msh >
```

```
msh >
msh >ethercat sii_read -p 0
00000000:  80 0E 44 EE 10 00 00 00   00 00 00 00 00 00 CB 00  ..D.............
00000010:  66 07 00 00 11 09 00 00   00 02 00 00 00 00 00 00  f...............
00000020:  00 00 00 00 00 00 00 00   00 10 80 00 80 10 80 00  ................
00000030:  00 10 00 01 00 14 00 01   04 00 00 00 00 00 00 00  ................
00000040:  FF FF FF FF FF FF 00 00   00 00 00 00 00 00 00 00  ................
00000050:  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00000060:  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00000070:  00 00 00 00 00 00 00 00   00 00 00 00 0F 00 01 00  ................
00000080:  0A 00 21 00 04 20 52 65   6E 65 73 61 73 20 45 74  ..!.. Renesas Et
00000090:  68 65 72 43 41 54 20 52   5A 2F 4E 32 20 43 6F 45  herCAT RZ/N2 CoE
000000a0:  20 32 70 6F 72 74 13 52   65 6E 65 73 61 73 20 52   2port.Renesas R
000000b0:  5A 2F 4E 32 20 47 72 6F   75 70 02 44 43 08 53 79  Z/N2 Group.DC.Sy
000000c0:  6E 63 68 72 6F 6E 1E 00   10 00 02 00 01 01 01 2F  nchron........./
000000d0:  00 00 00 00 00 0C 00 00   02 00 11 00 00 00 00 00  ................
000000e0:  00 00 00 00 00 00 00 00   00 00 28 00 02 00 01 02  ..........(.....
000000f0:  03 FF 29 00 10 00 00 10   00 01 26 00 01 01 00 14  ..).......&.....
00000100:  00 01 22 00 01 02 00 18   1C 00 64 00 01 03 00 1C  ..".......d.....
00000110:  20 00 20 00 01 04 3C 00   18 00 00 00 00 00 00 00   . ...<.........
00000120:  00 00 00 00 00 00 00 00   00 03 01 00 03 00 00 00  ................
00000130:  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00000140:  00 00 01 00 04 00 00 00   00 00 FF FF FF FF FF FF  ................
msh >
```

```
msh >
msh >
msh >ethercat coe_read -p 0 0x6041 0x00
00000000:  21 12                                              !.
msh >ethercat pdo_write -p 0 0 0x8000
Slave 0 pdo write offset 0x0000, size 2
msh >ethercat coe_read -p 0 0x6041 0x00
00000000:  21 12                                              !.
msh >ethercat pdo_write -p 0 0 0x0f00
Slave 0 pdo write offset 0x0000, size 2
msh >ethercat coe_read -p 0 0x6041 0x00
00000000:  37 12                                              7.
msh >
```

# 5.2. EtherCAT CoE 例程

**中文 | English**

## 简介

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

以下是 CoE 的主要特点和功能：

**基于CANopen**：

- CoE 的应用层直接采用了 CANopen 的设备协议，包括对象字典（Object Dictionary）的结构和服务。
- 通过对象字典定义设备参数、通信对象和控制数据，确保了设备间的互操作性。

**支持标准服务**：

- **SDO（Service Data Object）**：用于点对点的配置和诊断通信，允许主站与从站交换大容量数据（如参数配置）。
- **PDO（Process Data Object）**：用于实时通信，传输小数据量的周期性过程数据，支持快速响应。
- **Emergency（EMCY）消息**：用于报告设备异常情况。
- **NMT（Network Management）**：提供网络管理功能，如启动、停止和复位设备。

**高效传输**：

- EtherCAT 的总线结构和高速帧处理能力，使 CoE 能以更低的延迟和更高的效率进行数据交换。

**支持多种应用场景**：

- 适用于工业设备配置、实时监控、参数诊断和系统集成等。

**对象字典映射**：

- 对象字典以层级结构组织设备的数据和功能。

- EtherCAT 使用 CoE 协议访问对象字典中的变量，以实现参数读取、写入和实时控制。

**典型应用**：

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

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

## 前期准备

软件环境：

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

硬件环境：

- EtherKit开发板
- 网线一根
- Jlink调试器

## TwinCAT3配置

> *在启动TwinCAT3之前，我们还需要做一些配置操作：*

### 安装ESI文件

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

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

## 添加TwinCAT网卡驱动

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

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

检查网络适配器，可以看到已经成功安装了：

## FSP配置说明

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

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

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

使能网卡类型、配置网卡设备参数，这里我们添加两个phy（phy0和phy1），其中需要注意的是，EtherKit使用的是rtl8211网卡，并不在瑞萨FSP的支持范围内，但好在瑞萨预留了用户自定义网卡接口，因此按照如下设置来配置网卡，

同时设置MDIO类型为GMAC，设置网卡初始化回调函数phy_rtl8211f_initial()（）；

网卡引脚参数配置，选择操作模式为RGMII：

ETHER_ESC设置：

ETHER_GMAC配置：

为ethercat_ssc_port添加cmt定时器并配置中断优先级：

最后点击Generate Project Content生成底层驱动源码。

## 构建配置

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

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

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

if rtconfig.PLATFORM in ['iccarm'] + 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. Kconfig修改：打开工程下的文件（projects\etherkit_ethercat_coe\board\Kconfig），在Onboard Peripheral Drivers选项中加入CoE配置：

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

如下图所示：

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

# RT-Thread Studio配置

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

# EtherCAT CoE配置

## 新建TwinCAT工程

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

## 从站启动CoE App

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

## 从站设备扫描

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

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

## 更新EEPROM固件

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

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

这里按图示点击Download from List…：

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

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

## CiA402伺服使用说明

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

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

下面是对应上图各个状态的详细说明：

| 状态 | 说明 |
| --- | --- |
| 初始化 | 伺服初始化：伺服的参数不能设置，不能执行驱动指令功能 |
| 初始化完成 | 伺服初始化完成，可以设置伺服参数 |
| 伺服准备好 | 当前状态可以开启主电源，可以设置伺服参数，驱动器处于未激活状态 |
| 等待伺服使能 | 主电源OK，可以设置伺服参数，等待伺服使能 |
| 伺服使能 | 伺服使能，按照设置的模式运行 |
| 快速停机 | 快速停机功能被激活，驱动器正在执行快速停机功能 |
| 故障停机 | 驱动器发生故障，正在执行故障停机过程中 |

| 状态 | 说明 |
| --- | --- |
| 报警状态 | 故障停机完成，所有驱动功能均被禁止，同时允许更改驱动器参数以便排除故障 |

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

```
/*----------------------------------------------
-    CiA402 State machine
-----------------------------------------------*/
#define STATE_NOT_READY_TO_SWITCH_ON        0x0001 /**< \brief
#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
```

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

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

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

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

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

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

# CiA402对象字典定义

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

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

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

## EtherCAT COE测试

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

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

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

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

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

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

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

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

## csp位置模式控制

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

而如果想要在csp或csv模式下操作，必须先将其状态修改为STATE_OPERATION_ENABLED（可操作模式）。

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

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

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

比如说当前是位置模式，通过向Index:0x607A写入位置数值，我们写入100000：

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

## csv速度模式控制

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

同时我们也可以观察左侧对应的模块信息是否更新，并切换为csv模式：

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

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

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

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

同时可在输入中查看实际设置的速度信息是否一致：

# 5.3. EtherCAT EOE 例程

**中文 | English**

## 简介

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

以下是 EoE 的主要特点和功能：

1. **以太网隧道传输**：
2. 
    1. EoE 在 EtherCAT 通信帧中封装标准的以太网数据包，使标准以太网通信协议（如 TCP/IP、UDP、HTTP 等）可以通过 EtherCAT 网络传输。
3. **扩展网络功能**：
4. 
    1. 支持将 EtherCAT 从站作为虚拟以太网设备加入到 TCP/IP 网络中。
- 允许通过 EtherCAT 通信链路访问远程的标准以太网设备。
5. **高效整合**：
6. 
    1. EoE 的实现不会影响 EtherCAT 的实时性能。
- 非实时的以太网通信与实时的 EtherCAT 数据交换能够共存，各司其职。
7. **使用场景**：
8. 
    1. **设备管理**：通过 IP 协议访问 EtherCAT 从站设备（如远程配置、诊断和固件更新）。
- **混合网络**：集成需要标准以太网通信的设备（如摄像头、传感器或工控机）。
9. **简化网络布线**：
10. 
    1. 在工业自动化场景中，EoE 允许通过 EtherCAT 网络访问以太网设备，从而减少了独立以太网布线的需求。
11. **典型应用**：
12. 
    1. 工厂自动化系统中的远程监控和诊断。
    2. 工业机器人或生产设备与外部 IT 系统的通信桥接。

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

# 前期准备

软件环境：

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

硬件环境：

- EtherKit开发板
- 网线一根
- Jlink调试器

# TwinCAT3配置

> *在启动TwinCAT3之前，我们还需要做一些配置操作：*

## 安装ESI文件

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

> *注意：当前版本的ESI文件位于：..\board\ports\ethercat\ESI_File\Renesas EtherCAT RZT2 EoE.xml”*

## 添加TwinCAT网卡驱动

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

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

检查网络适配器，可以看到已经成功安装了：

# FSP配置说明

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

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

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

使能网卡类型、配置网卡设备参数，这里我们添加两个phy（phy0和phy1），其中需要注意的是，EtherKit使用的是rtl8211网卡，并不在瑞萨FSP的支持范围内，但好在瑞萨预留了用户自定义网卡接口，因此按照如下设置来配置网卡，同时设置MDIO类型为GMAC，设置网卡初始化回调函数ether_phy_targets_initialize_rtl8211_rgmii()；

网卡引脚参数配置，选择操作模式为RGMII：

ETHER_ESC设置：

ETHER_GMAC配置：

为ethercat_ssc_port添加cmt定时器并配置中断优先级：

添加Ethernet外设：

ethernet中断触发回调设置为：user_ether0_callback

最后点击Generate Project Content生成底层驱动源码。

## 构建配置

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

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

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

if rtconfig.PLATFORM in ['iccarm'] + 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. Kconfig修改：打开工程下的文件（projects\etherkit_ethercat_eoe\board\Kconfig），在Onboard Peripheral Drivers选项中加入EOE配置：

```
        config BSP_USING_ETHERCAT_EOE
            bool "Enable EtherCAT EOE example"
            select BSP_USING_ETH
            default n
            if BSP_USING_ETHERCAT_EOE
                config RT_LWIP_IPADDR
                    string "set static ip address for eoe 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
```

如下图所示：

```
Kconfig 1 ×
projects > etherkit_ethercat_eoe > board > Kconfig
3    config SOC_R9A07G084
4        bool
5        select SOC_SERIES_R9A07G0
6        select RT_USING_COMPONENTS_INIT
7        select RT_USING_USER_MAIN
8        default y
9
10   menu "Onboard Peripheral Drivers"
11
12       config BSP_USING_ETHERCAT_EOE
13           bool "Enable EtherCAT EOE example"
14           select BSP_USING_ETH
15           default n
16           if BSP_USING_ETHERCAT_EOE
17               config RT_LWIP_IPADDR
18                   string "set static ip address for eoe slaver"
19                   default "192.168.10.100"
20               config RT_LWIP_GWADDR
21                   string "set static gateway address for eoe slaver"
22                   default "192.168.10.1"
23               config RT_LWIP_MSKADDR
24                   string "set static mask address for eoe slaver"
25                   default "255.255.255.0"      You, last month • upload sample project
26           endif
27
28   endmenu
29
```

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

## RT-Thread Studio配置

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

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

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

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

## 新建TwinCAT工程

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

## 从站启动EOE App

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

## 从站设备扫描

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

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

## 更新EEPROM固件

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

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

这里按图示点击Download from List…：

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

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

## EtherCAT EOE通信

在完成EEPROM下载ESI固件并重新扫描添加设备后，激活Device我们可以观察到，板载有两颗绿色LED亮起（通信正常），并且其中一颗保持高频率闪烁一颗保持常亮，此时主从站就可以建立起正常的通信了。

## EIO测试

由于我们提供的EOE工程集成了EIO协议，因此可直接进行EIO测试，在本例程中，我们提供三个USER LED作为EIO的输入，回到TwinCAT，依次点击Device x(EtherCAT)->Box 1(Renesas EtherCAT RZ/N2 EOE 2port)->RxPD0-Map->OutputCounter：

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

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

## EOE测试

打开以太网适配器，选择主站所使用的适配器并设置静态IP：

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

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

- 主站IP：192.168.10.99
- 从站IP：192.168.10.100

# 拓展说明：3端口以太网EOE通信

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

## FSP配置

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

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

接下来配置引脚，使能ETH2；

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

完成上述配置后就可以点击生成源码了，回到工程编译并将程序下载开发板中；

## ESI固件更新

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

参考“**更新EEPROM固件**”一节，一样的步骤，只不过这次需要选择更新的固件为：Renesas EtherCAT RZ/N2 EoE 3port [2308 / 768]，点击烧录固件；

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

后续EOE开发请参考前几章节。

# 5.4. Ethernet/IP 例程

**中文 | English**

## 简介

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

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

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

## 前期准备

软件环境：

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

硬件环境：

- EtherKit开发板

## FSP配置

打开工程配置文件configuration.xml，新增r_gamc Stack：

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

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

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

网卡引脚参数配置，选择操作模式为RGMII：

ETHER_GMAC配置：

# RT-Thread Settings 配置

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

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

完成上述配置后，将程序编译下载至开发板。

# 网络配置

我们使用一根网线连接开发板与PC，同时在PC端配置静态IP：

# 软PLC启动

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

## CODESYS创建标准工程

请确保已安装CODESYS软件，安装之后下面这三个是我们需要用到的软件：

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

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

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

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

创建成功后就可以看到主界面了：

## Gateway 及 软PLC 启动

依次打开下面两个软件：

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

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

弹出设备用户登录窗口后，配置用户名和密码（用户自定义）：

检查网关设备及软PLC设备是否在线：

## Ethernet/IP EDS文件添加

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

本项目的EDS文件位于如下路径：

- ..\packages\OpENer_port-latest\eds_file

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

安装成功后可以看到 OpENer PC 从站描述文件：

## 设备添加

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

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

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

## 任务响应

保持默认配置即可。

## 网络配置

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

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

## EtherNet/IP线程应用启动

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

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

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

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

## 工程编译并启动调试

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

此时就可以看到 EtherNet/IP Scanner已经正常运行了：

## PLC编程及CIP IO控制

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

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

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

- 程序定义：这段代码的功能是：根据Board_SW_Input的每一位的状态，设置Board_LED_Output的相应位。具体来说：
- 如果Board_SW_Input的某一位为1，则对应的Board_LED_Output的该位为1。
- 如果Board_SW_Input的某一位为0，则对应的Board_LED_Output的该位为0。

通过循环遍历所有8个位，实现了将输入的每一位状态映射到输出的每一位。

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

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

工程中的配置位置如下图所示：

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

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

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

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

设备
Device
PLC_PRG
ENIPScanner10Task
Ethernet
EtherNet_IP_Scanner
OpEtNer_PC
通用
连接
程序集
用户参数
日志
EtherNet/IP I/O映射
EtherNet/IP IEC对象
状态
信息
查找
过滤器 显示所有
为IO通道添加FB...
转到实例
变量
映射
通道
地址
类型
当前值
预备值
单元
描述
Board LED Exclusive Owner
Board LED Input Data
Board LED Output Data
Application.PLC_PRG.Board_LED_Output
Board SW Input Only
Board SE Input Data
Application.PLC_PRG.Board_SW_Input
BYTE
BOOL
TRUE
FALSE
New Help String

## 5.5. Modbus-TCP/IP 例程

**中文 | English**

# 简介

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

# 硬件说明

# FSP配置说明

- 打开工程配置文件configuration.xml：

- 新增r_gamc Stack；

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

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

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

网卡引脚参数配置，选择操作模式为RGMII：

- ETHER_GMAC配置：

# RT-Thread Settings配置

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

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

## 编译&下载

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

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

## 运行效果

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

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Nov 25 2024 14:23:34
 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 modbus-tcpip routine!
===================================================
msh />[W/DBG] R_ETHER_Write failed!, res = 4001
[I/DBG] link up
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: 10.23.8.163
gw address: 10.23.8.254
net mask  : 255.255.255.0
dns server #0: 10.23.8.11
dns server #1: 119.29.29.29
msh />modbus_tcp_test
modbus_tcp_test: command not found.
msh /modbus_tcp_test
msh />[I/mb_tcp] server socket listen on port 502
```

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

连接成功后在开发板终端可以看到modbus客户端已连接：

回到Modbus Poll软件可以看到读写线圈功能都是正常的：

# 5.6. Modbus-UART 例程

**中文 | English**

## 简介

本例程基于agile_modbus软件包，展示了通过串口方式实现modbus协议通信的示例。Modbus UART 是一种通过串口通信实现的 Modbus 协议版本，广泛应用于工业自动化和控制系统中。Modbus 是一种开放的通信协议，用于在控制设备之间传输数据，支持多种物理层，如 UART、TCP/IP 和 RS-485/232。

## 硬件说明

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

## FSP配置

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

打开r_sci_uart配置，使能FIFO支持，同时设置通道数为3；

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

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

# RT-Thread Settings配置

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

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

## 编译&下载

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

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

## 运行效果

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

接着我们打开modbus slaver软件，点击连接：

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

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

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

# 5.7. PROFINET 例程

**中文 | English**

## 简介

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

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

在本示例中将使用P-Net软件包来实现PROFINET主从站通信。

## 前期准备

软件环境：

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

硬件环境：

- EtherKit开发板

## FSP配置

打开工程配置文件configuration.xml，新增r_gamc Stack：

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

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

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

网卡引脚参数配置，选择操作模式为RGMII：

ETHER_GMAC配置：

# RT-Thread Settings 配置

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

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

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

完成上述配置后，将程序编译下载至开发板。

# 网络配置

我们使用一根网线连接开发板与PC，同时在PC端配置静态IP：

检查开发板端的 IP 信息，并测试联通性：

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

Waiting for PLC connect request

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

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

# 软PLC启动

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

## CODESYS创建标准工程

请确保已安装CODESYS软件，安装之后下面这三个是我们需要用到的软件：

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

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

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

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

创建成功后就可以看到主界面了：

## Gateway 及 软PLC 启动

依次打开下面两个软件：

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

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

弹出设备用户登录窗口后，配置用户名和密码（用户自定义）：

检查网关设备及软PLC设备是否在线：

## profinet GSDML文件添加

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

本项目的 GSDML文件位于如下路径：

- ..\src\ports\rtthread\pn_dev

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

安装成功后可以看到 p-net 从站描述文件：

## 设备添加

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

图20-13 Ethernet添加

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

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

## 任务响应

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

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

## 网络配置

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

- PN_Controller 配置：双击左侧导航栏 PN_Controller(PN-Controller) -> 通用，并正确修改默认从站IP参数的区间，根据提示修改即可。
- P-Net 从站网络配置：双击左侧导航栏 P-Net-multiple-module sample app -> 通用， 修改IP参数为开发板IP

```
 \ | /
- 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 initializated!

** 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
```

## 工程编译并启动调试

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

此时就可以看到 PN 主站已经上线成功

# profinet 从站应用启动

开发板端上电后，一旦检测到网卡 link up，则会自动启动 PN 从站：

# PN协议栈运行demo

这里我们使用 CODESYS软件来测试 PN 的主从站交互。

## LED闪烁

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

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

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

## 从站 I&M(标识和维护) 数据修改

依然是扫描设备界面，我们点击左下角的 I&M，修改信息并写入 I&M：

同时 PNIO 会更新从站配置信息：

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

我们再次点击查看 I&M，即可发现 I&M 修改成功！

## PLC编程及PNIO控制

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

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

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

- 程序定义：

1. 首先在每次循环中，oscillator_cycles 增加 1。当计数器超过 200 时，重置计数器并切换 oscillator_state 的状态（TRUE 或 FALSE），实现周期性变化；
2. 如果按钮被按下（in_pin_button_LED 为 TRUE），并且在上一周期按钮状态是 FALSE，则切换 flashing 状态。即每次按钮按下时，切换 LED 是否闪烁的状态。
3. 如果 flashing 为 TRUE，则 LED 会根据振荡器状态 (oscillator_state) 闪烁；如果 flashing 为 FALSE，LED 直接关闭。
4. 在每次循环结束时，将当前按钮的状态保存在 in_pin_button_LED_previous 中，以便在下次判断按钮按下的事件。

```
oscillator_cycles := oscillator_cycles + 1;
IF oscillator_cycles > 200 THEN
    oscillator_cycles := 0;
    oscillator_state := NOT oscillator_state;
END_IF
IF in_pin_button_LED = TRUE THEN
    IF in_pin_button_LED_previous = FALSE THEN
        flashing := NOT flashing;
    END_IF
    out_pin_LED := TRUE;
ELSIF flashing = TRUE THEN
    out_pin_LED := oscillator_state;
ELSE
    out_pin_LED := FALSE;
```

```
END_IF
in_pin_button_LED_previous := in_pin_button_LED;
```

工程中的配置位置如下图所示：

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

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

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

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

Waiting for PLC connect request

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

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

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