RT-THREAD EtherKit 用户手册

RT-THREAD

上海睿赛德电子科技有限公司版权@2024

版本和修订

Data	Version	Author	Note
2024-11-11	V0.1.0	RT-Thread	对应 SDK_V1.0.0

目录

RT-THREAD EtherKit 用户手册 1

第 1 章 简介 14

1.1 目录结构 15

1.2 使用介绍 16

1.2.1 RT-Thread Studio 开发 16

1.2.2 IAR 开发 16

1.3 交流平台（QQ 群） 16

1.4 开发板购买渠道 16

第 2 章 RGB 闪烁例程 18

2.1 简介 18

2.2 硬件说明 18

2.3 软件说明 19

2.4 运行 20

2.4.1 编译&下载 20

2.4.2 运行效果 20

2.5 注意事项 21

2.6 引用参考 21

第 3 章 按键中断例程 22

3.1 简介 22

3.2 硬件说明 22

3.3 软件说明 23

3.3.1 FSP 配置 23

3.3.2 示例代码说明 25

3.4 运行 27

3.4.1 编译&下载 27

3.4.2 运行效果 27

3.5 注意事项 28

3.6 引用参考 28

第 4 章 RTC 及 alarm 使用例程 29

4.1 简介 29

4.2 硬件说明 29

4.3 软件说明 29

4.3.1 FSP 配置说明 29

4.3.2 RT-Thread Settings 配置 30

4.3.2 示例代码说明 31

4.4 运行 33

4.4.1 编译&下载 33

4.4.2 运行效果 33

4.5 注意事项 34

4.6 引用参考 34

第 5 章 ADC 例程 35

5.1 简介 35

5.2 硬件说明 35

5.3 软件说明 35
5.3.1 FSP 配置说明 35
5.3.2 RT-Thread Settings 配置 36
5.3.3 示例工程说明 36
5.4 运行 38
5.4.1 编译&下载 38
5.4.2 运行效果 38
5.5 注意事项 39
5.6 引用参考 39
第 6 章 I2C 例程 40
6.1 简介 40
6.2 硬件说明 40
6.3 软件说明 40
6.3.1 FSP 配置说明 40
6.3.2 RT-Thread Settings 配置 41
6.3.3 示例工程说明 42
6.4 运行 44
6.4.1 编译&下载 44
6.4.2 运行效果 44
6.5 注意事项 45
6.6 引用参考 45
第 7 章 SPI 例程 46

7.1 简介 46
7.2 硬件说明 46
7.3 软件说明 46
7.3.1 FSP 配置说明 46
7.3.2 RT-Thread Settings 配置 47
7.3.3 示例工程说明 47
7.4 运行 49
7.4.1 编译&下载 49
7.4.2 运行效果 50
7.5 注意事项 50
7.6 引用参考 50
第 8 章 GPT 例程 51
8.1 简介 51
8.2 硬件说明 51
8.3 软件说明 51
8.3.1 FSP 配置说明 51
8.3.2 RT-Thread Settings 配置 52
8.3.3 示例工程说明 52
8.4 运行 54
8.4.1 编译&下载 54
8.4.2 运行效果 55
8.5 注意事项 55

8.6 引用参考 55
第 9 章 WDT 例程 56
9.1 简介 56
9.2 硬件说明 56
9.3 软件说明 56
9.3.1 FSP 配置说明 56
9.3.2 RT-Thread Settings 配置 56
9.3.3 示例工程说明 57
9.4 运行 58
9.4.1 编译&下载 58
9.4.2 运行效果 59
9.5 注意事项 59
9.6 引用参考 60
第 10 章 RS485 例程 61
10.1 简介 61
10.2 硬件说明 61
10.3 软件说明 61
10.3.1 FSP 配置说明 61
10.3.2 工程示例说明 62
10.4 运行 63
10.4.1 编译&下载 63
10.4.2 运行效果 63

10.5 注意事项 64

10.6 引用参考 64

第 11 章 以太网例程 65

11.1 简介 65

11.2 硬件说明 65

11.3 软件说明 67

11.3.1 FSP 配置 67

11.3.2 RT-Thread Settings 配置 70

11.3.3 示例代码说明 71

11.4 运行 71

11.4.1 编译&下载 71

11.4.2 运行效果 71

11.5 注意事项 72

11.6 引用参考 72

第 12 章 CANFD 例程 73

12.1 简介 73

12.2 硬件说明 73

12.3 软件说明 73

12.3.1 FSP 配置说明 73

12.3.2 RT-Thread Settings 配置 74

12.3.3 工程示例说明 74

12.4 运行 77

12.4.1 编译&下载 77
12.4.2 运行效果 78
12.5 注意事项 78
12.6 引用参考 78
第 13 章 Netutils 例程 79
13.1 简介 79
13.2 硬件说明 79
13.3 软件说明 79
13.3.1 FSP 配置说明 79
13.3.2 RT-Thread Settings 配置 79
13.4 运行 80
13.4.1 编译&下载 80
13.4.2 运行效果 81
13.5 注意事项 85
13.6 引用参考 85
第 14 章 MQTT 例程 86
14.1 简介 86
14.2 硬件说明 86
14.3 软件说明 86
14.3.1 FSP 配置 86
14.3.2 RT-Thread Settings 配置 86
14.3.3 示例代码说明 87

14.4 运行 89
14.4.1 编译&下载 89
14.4.2 MQTTX 配置 89
14.4.3 运行效果 91
14.5 其他说明 91
14.6 引用参考 92
第 15 章 Modbus-UART 例程 93
15.1 简介 93
15.2 硬件说明 93
15.3 软件说明 93
15.3.1 FSP 配置 93
15.3.2 RT-Thread Settings 配置 95
15.4 运行 96
15.4.1 编译&下载 96
15.4.2 运行效果 97
15.5 注意事项 99
15.6 引用参考 99
第 16 章 Modbus-TCP/IP 例程 100
16.1 简介 100
16.2 硬件说明 100
16.3 软件说明 100
16.3.1 FSP 配置 100

16.3.2 RT-Thread Settings 配置 100
16.4 运行 101
16.4.1 编译&下载 101
16.4.2 运行效果 102
16.5 注意事项 104
16.6 引用参考 104
第 17 章 USB-PMSC 例程 105
17.1 简介 105
17.2 硬件说明 105
17.3 软件说明 106
17.3.1 FSP 配置 106
17.3.2 RT-Thread Settings 配置 107
17.4 运行 108
17.4.1 编译&下载 108
17.4.2 运行效果 108
17.5 注意事项 109
17.6 引用参考 109
第 18 章 USB-PCDC 例程 110
18.1 简介 110
18.2 硬件说明 110
18.3 软件说明 111
18.3.1 FSP 配置 111

18.3.2 RT-Thread Settings 配置 112
18.4 运行 113
18.4.1 编译&下载 113
18.4.2 运行效果 113
18.5 注意事项 114
18.6 引用参考 114
第 19 章 EtherCAT-EOE 例程 115
19.1 简介 115
19.2 前期准备 116
19.3 TwinCAT3 配置 116
19.3.1 安装 ESI 文件 117
19.3.2 添加 TwinCAT 网卡驱动 117
19.4 FSP 及 Studio 配置 119
19.4.1 FSP 配置 119
19.4.2 RT-Thread Studio 配置 124
19.5 EtherCAT EOE 配置 125
19.5.1 新建 TwinCAT 工程 125
19.5.2 从站启动 EOE App 126
19.5.3 从站设备扫描 126
19.5.4 更新 EEPROM 固件 127
19.6 EtherCAT EOE 通信 129
19.6.1 EIO 测试 130

19.6.2 EOE 测试 132

第 20 章 PROFIENT 例程 134

20.1 简介 134

20.2 前期准备 134

20.3 软件包配置 134

20.4 网络配置 135

20.5 软 PLC 启动 137

20.5.1 CODESYS 创建标准工程 137

20.5.2 Gateway 及 软 PLC 启动 139

20.5.3 profinet GSDML 文件添加 141

20.5.4 设备添加 142

20.5.5 任务响应 144

20.5.6 网络配置 145

20.5.7 工程编译并启动调试 146

20.6 profinet 从站应用启动 147

20.7 PN 协议栈运行 demo 149

第 21 章 FAQ 150

21.1 芯片状态异常 150

第 1 章 简介

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

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

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

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

图 1-1 EtherKit 正面图

1.1 目录结构

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

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

1.2 使用介绍

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

1.2.1 RT-Thread Studio 开发

1.打开 RT-Thread Studio 的包管理器，安装 EtherKit 开发板资源包；

2. 安装完成后，选择基于 BSP 创建工程即可。

1.2.2 IAR 开发

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

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

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

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

1.3 交流平台（QQ 群）

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

1.4 开发板购买渠道

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

第 2 章 RGB 闪烁例程

2.1 简介

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

2.2 硬件说明

图 2.1 LED 电路原理图

图 2.2 LED 引脚示意图

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

LED 在开发板中的位置如下图所示：

图 2-3 LED 位置

2.3 软件说明

本例程的源码位于/projects/etherkit_blink_led.

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

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

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

        /* 输出 LOG 信息 */
        LOG_D("group: %d | red led [%-3.3s] | | blue led [%-3.3s] | | green
led [%-3.3s]",
            group_current,
```

```
            _blink_tab[group_current][0] == LED_ON ? "ON" : "OFF",
            _blink_tab[group_current][1] == LED_ON ? "ON" : "OFF",
            _blink_tab[group_current][2] == LED_ON ? "ON" : "OFF");

        count++;

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

2.4 运行

2.4.1 编译&下载

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

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

2.4.2 运行效果

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

图 2-4 RGB-LED 演示

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

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

2.5 注意事项

暂无

2.6 引用参考

- 设备与驱动：PIN 设备

第 3 章 按键中断例程

3.1 简介

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

3.2 硬件说明

图 3-1 key 电路原理图

图 3-2 key 引脚示意图

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

KEY 在开发板中的位置如下图所示：

图 3-3 按键位置

3.3 软件说明

3.3.1 FSP 配置

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

图 3-4 打开配置文件

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

图 3-5 新增 IRQ Stack

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

图 3-6 IRQ 开启

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

图 3-7 IRQ 配置

3.3.2 示例代码说明

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

KEY1(LEFT) 、KEY2(RIGHT)对应的单片机引脚定义如下。

```
/* 配置 key irq 引脚 */

#define IRQ_TEST_PIN1 BSP_IO_PORT_14_PIN_2
#define IRQ_TEST_PIN2 BSP_IO_PORT_16_PIN_3
```

LED 灯的单片机引脚定义如下。

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

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

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

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

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

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

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

3.4 运行

3.4.1 编译&下载

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

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

3.4.2 运行效果

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

```
 \ | /
- 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!
```

图 3-8 IRQ 运行示例

3.5 注意事项

暂无

3.6 引用参考

- 设备与驱动：PIN 设备

第 4 章 RTC 及 alarm 使用例程

4.1 简介

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

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

4.2 硬件说明

本例程使用的 RTC 设备依赖于 LSE 时钟，此外无需过多连接。

4.3 软件说明

4.3.1 FSP 配置说明

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

图 5-1 新增 RTC Stack

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

图 4-2 RTC 参数配置

4.3.2 RT-Thread Settings 配置

打开 RT-Thread Settings，找到硬件选项，使能 RTC；

图 4-3 RTC 使能

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

图 4-4 RTC 配置

4.3.2 示例代码说明

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

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

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

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

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

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

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

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

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

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

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

    if (alarm != RT_NULL)
        return;

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

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

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

4.4 运行

4.4.1 编译&下载

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

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

4.4.2 运行效果

按下复位按键重启开发板，可以看到板子上会打印如下信息：

```
 \ | /
- RT -     Thread Operating System
```

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

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

4.5 注意事项

暂无

4.6 引用参考

- 设备与驱动：RTC 设备

第 5 章 ADC 例程

5.1 简介

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

5.2 硬件说明

图 5-1 硬件原理图

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

5.3 软件说明

5.3.1 FSP 配置说明

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

第二步：新建 r_adc Stack 配置 adc 设备以及所用通道;

图 5-2 FSP 配置

5.3.2 RT-Thread Settings 配置

在 settings 里打开 adc0 的外设；

图 5-3 Settings 配置

5.3.3 示例工程说明

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

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

图 5-4 ADC 相关宏定义

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

具体代码如下：

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

```
    ret = rt_adc_disable(adc_dev, ADC_DEV_CHANNEL);

    return ret;

}
```

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

5.4 运行

5.4.1 编译&下载

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

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

5.4.2 运行效果

使用 adc0 的 0 通道采集 1.8v 电压效果如下

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

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

图 7-6 ADC 示例运行效果图

5.5 注意事项

暂无

5.6 引用参考

- 设备与驱动：ADC 设备

第 6 章 I2C 例程

6.1 简介

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

6.2 硬件说明

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

图 6-1 IIC 原理图

6.3 软件说明

6.3.1 FSP 配置说明

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

图 6-2 IIC master stack

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

图 6-3 IIC 配置

6.3.2 RT-Thread Settings 配置

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

图 6-4 IIC 使能

图 6-5 使能软件包

6.3.3 示例工程说明

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

```
#ifdef PKG_USING_AT24CXX
#include "at24cxx.h"
#define EEPROM_I2C_NAME "i2c0"
static at24cxx_device_t at24c02_dev;
static void eeprom_test(void)
{
    char str1[] = "test string-hello rtthread\n";
    char str2[] = "test string-rzt2m eeprom testcase\n";
    uint8_t read_buffer1[50];
    uint8_t read_buffer2[50];
    at24c02_dev = at24cxx_init(EEPROM_I2C_NAME, 0x0);
```

```
    if (at24c02_dev == RT_NULL)
    {
        rt_kprintf("eeprom init failed\n");
        return;
    }
    rt_memset(read_buffer1, 0x0, sizeof(read_buffer1));
    rt_memset(read_buffer2, 0x0, sizeof(read_buffer2));
    at24cxx_write(at24c02_dev, 0x0, (uint8_t *)str1, (sizeof(str1) -
1));
    rt_kprintf("write eeprom data to 0x0: %s\n", str1);
    rt_thread_mdelay(1000);
    at24cxx_read(at24c02_dev, 0x0, read_buffer1, (sizeof(str1) - 1));
    rt_kprintf("read eeprom data from 0x0: %s\n", read_buffer1);
    at24cxx_write(at24c02_dev, 0x20, (uint8_t *)str2, (sizeof(str2) -
1));
    rt_kprintf("write eeprom data to 0x20: %s\n", str2);
    rt_thread_mdelay(1000);
    at24cxx_read(at24c02_dev, 0x20, read_buffer2, (sizeof(str2) - 1));
    rt_kprintf("read eeprom data from 0x20: %s\n", read_buffer2);
    if (rt_strcmp((const char *)str1, (const char *)read_buffer1) != 0 &
& rt_strcmp((const char *)str2, (const char *)read_buffer2) != 0)
        rt_kprintf("eeprom test fail\n");
    else
        rt_kprintf("eeprom test success\n");
    at24cxx_deinit(at24c02_dev);
}
MSH_CMD_EXPORT(eeprom_test, eeprom test sample);
#endif
```

6.4 运行

6.4.1 编译&下载

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

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

6.4.2 运行效果

在串口终端输入 eeprom_test 指令；

```
 \ | /
- 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 >
```

图 6-6 IIC 测试

6.5 注意事项

暂无

6.6 引用参考

- 设备与驱动：I2C 设备

第 7 章 SPI 例程

7.1 简介

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

7.2 硬件说明

EtherKit 板载资源有 PMOD 接口，连接到 R9A07G084M08GBG 芯片的 SCI _SPI3；

图 7-1 SPI 硬件原理图

7.3 软件说明

7.3.1 FSP 配置说明

打开 FSP 工具 新建 Stacks 选择 r_sci_spi3；

图 7-2 SPI 配置

7.3.2 RT-Thread Settings 配置

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

图 7-3 SPI 使能

7.3.3 示例工程说明

基于 RT-Thread 的 SCI 驱动框架实现对 PMODE 的 loop 回环测试；（将 PMO

D 的 SPI3_MOSI 连接到 SPI3_MISO）；

代码如下：

```
void spi_loop_test(void)
{
#define TEXT_NUMBER_SIZE 1024
#define SPI_BUS_NAME "sci3s"
#define SPI_NAME "spi30"
    static uint8_t sendbuf[TEXT_NUMBER_SIZE] = {0};
    static uint8_t readbuf[TEXT_NUMBER_SIZE] = {0};
    for (int i = 0; i < sizeof(readbuf); i++)
    {
        sendbuf[i] = i;
    }
    static struct rt_spi_device *spi_dev = RT_NULL;
    struct rt_spi_configuration cfg;
    rt_hw_sci_spi_device_attach(SPI_BUS_NAME, SPI_NAME, NULL);
    cfg.data_width = 8;
    cfg.mode = RT_SPI_MASTER | RT_SPI_MODE_0 | RT_SPI_MSB | RT_SPI_NO_C
S;
    cfg.max_hz = 1 * 1000 * 1000;
    spi_dev = (struct rt_spi_device *)rt_device_find(SPI_NAME);
    if (RT_NULL == spi_dev)
    {
        rt_kprintf("spi sample run failed! can't find %s device!\n", SPI_
NAME);
        return;
    }
    rt_spi_configure(spi_dev, &cfg);
    rt_kprintf("%s send:\n", SPI_NAME);
```

```
    for (int i = 0; i < sizeof(sendbuf); i++)
    {
        rt_kprintf("%02x ", sendbuf[i]);
    }
    rt_spi_transfer(spi_dev, sendbuf, readbuf, sizeof(sendbuf));
    rt_kprintf("\n\n%s rcv:\n", SPI_NAME);
    for (int i = 0; i < sizeof(readbuf); i++)
    {
        if (readbuf[i] != sendbuf[i])
        {
            rt_kprintf("SPI test fail!!!\n");
            break;
        }
        else
            rt_kprintf("%02x ", readbuf[i]);
    }
    rt_kprintf("\n\n");
    rt_kprintf("SPI test end\n");
}
```

7.4 运行

7.4.1 编译&下载

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

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

板。

7.4.2 运行效果

打开串口工具，可以看到通过 spi 的发送与接收数据一致；

图 7-4 SPI 示例运行效果

7.5 注意事项

暂无

7.6 引用参考

- 设备与驱动：SPI 设备

第 8 章 GPT 例程

8.1 简介

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；

8.2 硬件说明

本例程使用定时器功能，无需硬件连接。

8.3 软件说明

8.3.1 FSP 配置说明

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

Common	
Parameter Checking	Default (BSP)
Pin Output Support	Enabled
Write Protect Enable	Disabled
Multiplex Interrupt	Disabled
Module g_timer0 Timer, General PWM (r_gpt)	
General	
Name	g_timer0
Unit	0
Channel	0
Mode	Periodic
Period	0x100000000
Period Unit	Raw Counts
Output	
Input	
Interrupts	
Extra Features	
ELC	
Pins	
GTIOC0A	P00_4
GTIOC0B	P00_5

Property	Value
General	
Name	g_timer5
Unit	0
Channel	5
Mode	PWM
Period	100
Period Unit	Kilohertz
Output	
Input	
Interrupts	
Extra Features	
ELC	

图 8-1 FSP 配置

并配置 pins 使能 GPT0 GPT5；

图 8-2 GPT 引脚配置

8.3.2 RT-Thread Settings 配置

在配置中打开 timer0 使能；

图 8-3 使能定时器

8.3.3 示例工程说明

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

```
int hwtimer_sample(void)
{
    rt_err_t ret = RT_EOK;
    rt_hwtimerval_t timeout_s;
```

```
    rt_device_t hw_dev = RT_NULL;
    rt_hwtimer_mode_t mode;
    rt_uint32_t freq = 400000000; /* 1Mhz */
    hw_dev = rt_device_find(HWTIMER_DEV_NAME);
    if (hw_dev == RT_NULL)
    {
        rt_kprintf("hwtimer sample run failed! can't find %s device!\n",
HWTIMER_DEV_NAME);
        return -RT_ERROR;
    }
    ret = rt_device_open(hw_dev, RT_DEVICE_OFLAG_RDWR);
    if (ret != RT_EOK)
    {
        rt_kprintf("open %s device failed!\n", HWTIMER_DEV_NAME);
        return ret;
    }
    rt_device_set_rx_indicate(hw_dev, timeout_cb);
    rt_device_control(hw_dev, HWTIMER_CTRL_FREQ_SET, &freq);
    mode = HWTIMER_MODE_PERIOD;
    ret = rt_device_control(hw_dev, HWTIMER_CTRL_MODE_SET, &mode);
    if (ret != RT_EOK)
    {
        rt_kprintf("set mode failed! ret is :%d\n", ret);
        return ret;
    }
    /* Example Set the timeout period of the timer */
    timeout_s.sec = 1;  /* secend */
    timeout_s.usec = 0; /* microsecend */
    if (rt_device_write(hw_dev, 0, &timeout_s, sizeof(timeout_s)) != siz
eof(timeout_s))
```

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

每隔 1s 中触发一次中断回调函数打印输出，下面是 PWM 配置使能；

```
void GPT_PWM_Init(void)
{
    R_GPT_Open(&g_timer5_ctrl, &g_timer5_cfg);
    R_GPT_Start(&g_timer5_ctrl);
}
```

8.4 运行

8.4.1 编译&下载

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

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

8.4.2 运行效果

每隔 1s 触发回调函数并打印输出；

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

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

图 10-6 定时器示例运行

使用逻辑分析仪量取 pwm 输出波形如下所示；

图 10-6 pwm 波形显示

8.5 注意事项

暂无

8.6 引用参考

- 设备与驱动：HWTIMER 设备

第 9 章 WDT 例程

9.1 简介

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 WDT 设备;

9.2 硬件说明

本例程使用看门狗功能，无需硬件连接。

9.3 软件说明

9.3.1 FSP 配置说明

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

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

图 9-1 看门狗配置

9.3.2 RT-Thread Settings 配置

打开 RT-Thread Settings，选择硬件配置并使能看门狗；

图 9-2 Settings 配置图

9.3.3 示例工程说明

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

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

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

9.4 运行

9.4.1 编译&下载

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

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

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

9.4.2 运行效果

打开串口工具，在命令行终端运行 wdt_test，执行看门狗示例；

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

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

图 9-3 WDT 示例运行效果

9.5 注意事项

暂无

9.6 引用参考

- 设备与驱动：WDT 设备

第 10 章 RS485 例程

10.1 简介

本例程主要介绍了如何在 EtherKit 上使用 RS485 设备；

10.2 硬件说明

图 10-1 rs485 硬件原理图示

10.3 软件说明

10.3.1 FSP 配置说明

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

Property	Value
DMAC Support	Disable
Flow Control Support	Disable
Multiplex Interrupt	Disabled
Module g_uart5 UART (r_sci_uart)	
General	
Name	g_uart5
Channel	5
Data Bits	8bits
Parity	None
Stop Bits	1bit
Baud	
Baud Rate	9600
Baud Rate Modulation	Disabled
Max Error (%)	5
Synchronizer Bypass	Not Bypassed (The operation clock is SCInASYNCCLK)
Flow Control	
Extra	
Interrupts	
Callback	rs485_callback
Receive Interrupt Priority	Priority 12
Transmit Data Empty Interrupt Priority	Priority 12
Transmit End Interrupt Priority	Priority 12
Error Interrupt Priority	Priority 12
Pins	
CTS5#	None

图 10-2 串口配置

10.3.2 工程示例说明

rs485 发送函数：每隔 1s 发送一次，总共发送 10 次数据，每次发送为 1 个字节；

```
int rs485_send_test(void)
{
   static uint8_t i;

   for(i =1; i <= 10; i++)
   {
       /*发送数据*/
       RS485_Send_Example(i);
       rt_thread_delay(1000);
   }
   return 0;
}
```

rs485 接收中断函数（需要在 FSP 里提前配置接收中断名称）；

```
/*RS485_1 中断回调函数*/
void rs485_callback(uart_callback_args_t * p_args)
{
    rt_interrupt_enter();

    switch(p_args->event)
    {
        /*接收数据时将数据打印出来*/
        case UART_EVENT_RX_CHAR:
          {
```

```
            rt_kprintf("%d\n", p_args->data);
            break;
          }
        default:
            break;
    }
}
```

10.4 运行

10.4.1 编译&下载

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

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

10.4.2 运行效果

串口输出指令 rs485_send 指令，打开另一个串口的终端查看收到的数据：

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Nov 25 2024 14:57:08
 2006 - 2024 Copyright by RT-Thread team

Hello RT-Thread!
===================================================
This example project is an driver rs485 routine!
===================================================
msh >
msh >rs
rs485_send_test
msh >rs485_send_test
```

图 10-3 rs485 发送

```
msh >1
2
3
4
5
6
7
8
9
10
```

图 10-4 rs485 接收

10.5 注意事项

暂无

10.6 引用参考

- 设备与驱动：UART_V2 设备

第 11 章 以太网例程

11.1 简介

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

11.2 硬件说明

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

图 11-1 ETH0

图 11-2 ETH1

图 11-3 ETH2

11.3 软件说明

11.3.1 FSP 配置

打开工程配置文件 configuration.xml：

图 11-4 打开配置文件

新增 r_gamc Stack；

图 11-5 r_gmac stack

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

图 11-6 以太网回调函数

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

图 11-7 PHY 配置

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

图 11-8 selector 配置

网卡引脚参数配置，选择操作模式为 RGMII：

图 11-9 ETH 引脚配置

ETHER_GMAC 配置：

图 11-10 GMAC 引脚配置

11.3.2 RT-Thread Settings 配置

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

图 11-11 settings 配置

11.3.3 示例代码说明

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

11.4 运行

11.4.1 编译&下载

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

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

11.4.2 运行效果

图 11-11 以太网 ping 命令

11.5 注意事项

暂无

11.6 引用参考

暂无

第 12 章 CANFD 例程

12.1 简介

本例程主要介绍了如何在 EtherKit 上使用 canfd 设备。

12.2 硬件说明

图 12-1 canfd 原理图示

12.3 软件说明

12.3.1 FSP 配置说明

打开工程下的 configuration.xml 文件，添加 canfd stack，并完成下述配置；

Property	Value
Common	
Global Error Interrupt	
Flexible Data (FD)	
Reception	
Parameter Checking	Default (BSP)
Multiplex Interrupt	Disabled
Transmission Priority	Buffer Number
DLC Check	Disabled
Clock Source	PCLKCAN
Module g_canfd0 CAN FD (r_canfd)	
General	
Name	g_canfd0
Channel	0
Bitrate	
Interrupts	
Transmit Interrupts	
Channel Error Interrupts	
Filter List Array	p_canfd0_afl
Pins	
CANRX0	P05_2
CANRXDP0	None
CANTX0	P05_3
CANTXDP0	None

图 12-2 canfd0 配置

图 12-3 canfd1 配置

12.3.2 RT-Thread Settings 配置

打开 RT-Thread Settings，使能 canfd 配置；

图 12-4 Settings 配置图

12.3.3 工程示例说明

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

```
int can0_sample_send(int argc, char *argv[])
{
    rt_err_t res;
```

```
    rt_thread_t thread;
    char can_name[RT_NAME_MAX];
    if (argc == 2)
    {
        rt_strncpy(can_name, argv[1], RT_NAME_MAX);
    }
    else
    {
        rt_strncpy(can_name, CAN0_DEV_NAME, RT_NAME_MAX);
    }
    /* 查找 CAN 设备 */
    can0_dev = rt_device_find(can_name);
    if (!can0_dev)
    {
        rt_kprintf("find %s failed!\n", can_name);
        return RT_ERROR;
    }
    /* 以中断接收及发送方式打开 CAN 设备 */
    res = rt_device_open(can0_dev, RT_DEVICE_FLAG_INT_TX | RT_DEVICE_FLA
G_INT_RX);
    RT_ASSERT(res == RT_EOK);
    /* 创建数据接收线程 */
    thread = rt_thread_create("can0_tx", can0_tx_thread, RT_NULL, 1024,
25, 10);
    if (thread != RT_NULL)
    {
        rt_thread_startup(thread);
    }
    else
    {
```

```
        rt_kprintf("create can_rx thread failed!\n");
    }
    return res;
}
/* 导出到 msh 命令列表中 */
MSH_CMD_EXPORT(can0_sample_send, can device sample);
int can1_sample_receive(int argc, char *argv[])
{
    rt_err_t res;
    rt_thread_t thread;
    char can_name[RT_NAME_MAX];
    if (argc == 2)
    {
        rt_strncpy(can_name, argv[1], RT_NAME_MAX);
    }
    else
    {
        rt_strncpy(can_name, CAN1_DEV_NAME, RT_NAME_MAX);
    }
    /* 查找 CAN 设备 */
    can1_dev = rt_device_find(can_name);
    if (!can1_dev)
    {
        rt_kprintf("find %s failed!\n", can_name);
        return RT_ERROR;
    }
    /* 初始化 CAN 接收信号量 */
    rt_sem_init(&rx_sem, "rx_sem", 0, RT_IPC_FLAG_FIFO);
```

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

12.4 运行

12.4.1 编译&下载

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

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

12.4.2 运行效果

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

```
Hello RT-Thread!
===================================================
This example projec
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Nov 25 2024 14:59:33
 2006 - 2024 Copyright by RT-Thread team

Hello RT-Thread!
===================================================
This example project is an canfd routine!
===================================================
msh >can
can0_sample_send
can1_sample_receive
canstat
msh >can0_
can0_sample_send
msh >can0_sample_send
msh >can1_sa
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
```

图 12-5 canfd 回环测试

12.5 注意事项

暂无

12.6 引用参考

- 设备与驱动：CAN 设备

第 13 章 Netutils 例程

13.1 简介

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

13.2 硬件说明

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

13.3 软件说明

13.3.1 FSP 配置说明

此处配置请参考第 11 章：11.3.1 FSP 配置。

13.3.2 RT-Thread Settings 配置

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

图 13-1 以太网使能

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

图 13-2 使能 netutils

13.4 运行

13.4.1 编译&下载

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

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

13.4.2 运行效果

13.4.2.1 TFTP Server 发送测试

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

图 13-3 安装 tftp 工具

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

图 13-4 tftp 服务开启

然后打开安装好的 Tftpd64-4.60 软件，配置如下信息：

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

图 13-5 tftp 文件传输测试

点击 Put 后，会提示已经发送信息；

图 13-6 tftp 测试成功

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

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

图 13-7 查看接收文件

13.4.2.2 TFTP 接收测试

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

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

图 13-8 写入文件

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 />
```

图 13-9 查看写入文件

打开安装好的 Tftpd64-4.60 软件，具体配置说明如下：

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

图 13-10 tftp 接收配置

可以看到 2.txt 已经接受成功，内容也是开发板文件系统中的文件内容；

图 13-11 查看接收文件

13.4.2.3 NTP 联网校时

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

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

```
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 />
```

图 13-12 ntp 对时

13.5 注意事项

暂无

13.6 引用参考

第 14 章 MQTT 例程

14.1 简介

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

14.2 硬件说明

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

14.3 软件说明

14.3.1 FSP 配置

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

14.3.2 RT-Thread Settings 配置

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

图 14-1 settings 配置

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

图 14-2 软件包配置

14.3.3 示例代码说明

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

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

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

```
    }
    rt_thread_startup(tid_mqtt);
    return RT_EOK;
}
MSH_CMD_EXPORT(ka_mqtt, Kawaii MQTT client test program);
```

14.4 运行

14.4.1 编译&下载

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

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

14.4.2 MQTTX 配置

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

图 15-3 新增 MQTT 连接

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

图 14-4 MQTT 客户端配置

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

图 14-5 new Subscription

在下方功能框中编写订阅主题名称为 sub5323，订阅信息按自己需求配置；

图 14-6 发布主题

14.4.3 运行效果

打开串口工具，运行 ka_mqtt 命令后查看：

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

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

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

图 14-7 MQTT 运行

14.5 其他说明

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

14.6 引用参考

- 软件包：kawaii-mqtt

第 15 章 Modbus-UART 例程

15.1 简介

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。

15.2 硬件说明

图 15-1 UART3 串口原理图

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

15.3 软件说明

15.3.1 FSP 配置

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

图 15-2 添加 sci_uart stack

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

图 15-3 sci-uart 配置

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

图 15-4 sci-uart 引脚模式配置

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

图 15-5 串口引脚信息

15.3.2 RT-Thread Settings 配置

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

图 15-6 settings 配置串口

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

图 15-7 软件包使能

15.4 运行

15.4.1 编译&下载

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

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

15.4.2 运行效果

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

图 15-8 usb-ttl 接线示意

接着我们打开 modbus slaver 软件，点击连接；

图 15-9 modbus slaver 连接

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

图 15-10 modbus slaver 设置

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

图 15-11 开启 modbus 主站示例

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

图 15-12 modbus 从站寄存器修改

15.5 注意事项

暂无

15.6 引用参考

- 软件包：agile_modbus

第 16 章 Modbus-TCP/IP 例程

16.1 简介

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。

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

16.2 硬件说明

本例程使用的硬件为以太网接口，请确保以太网功能正常

16.3 软件说明

16.3.1 FSP 配置

注：该小节配置同 11.3.1 FSP 配置。

16.3.2 RT-Thread Settings 配置

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

图 16-1 settings 配置以太网

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

图 16-2 软件包使能

16.4 运行

16.4.1 编译&下载

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

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

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

16.4.2 运行效果

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

```
 \ | /
- 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
```

图 16-1 启动 mdobus tcp 测试

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

图 16-2 mdobus tcp/ip 设置

连接成功后在开发板终端可以看到 modbus 客户端已连接；

图 16-3 modbus tcp client 连接

回到 Modbus Poll 软件可以看到读写线圈功能都是正常的；

图 16-4 modbus 线圈读取

16.5 注意事项

暂无

16.6 引用参考

- 软件包：kawaii-mqtt

第 17 章 USB-PMSC 例程

17.1 简介

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

17.2 硬件说明

USB-DEVICE

图 17-1 USB-Device 原理图

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

图 17-2 USB-Device 位置

17.3 软件说明

17.3.1 FSP 配置

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

图 17-3 usb_pmsc stack 添加

添加 g_rm_block_media0；

图 17-4 block media0 添加

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

图 17-5 添加 USB callback

下面配置 USB 引脚，找到 USB_HS 并使能；

图 17-6 USB 引脚配置

17.3.2 RT-Thread Settings 配置

USB 示例目前使用的是 freertos 接口驱动，因此我们还需要使能 Freertos 兼

容层软件包；

图 17-6 使能 freertos 兼容层

17.4 运行

17.4.1 编译&下载

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

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

17.4.2 运行效果

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

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

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

图 17-7 USB-PMSC 终端

图 17-8 USB 模拟 U 盘

17.5 注意事项

暂无

17.6 引用参考

第 18 章 USB-PCDC 例程

18.1 简介

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

18.2 硬件说明

USB-DEVICE

图 18-1 USB-Device 原理图

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

图 18-2 USB-Device 位置

18.3 软件说明

18.3.1 FSP 配置

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

图 18-3 usb_pcdc stack 添加

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

图 18-4 添加 USB callback

下面配置 USB 引脚，找到 USB_HS 并使能；

图 18-5 USB 引脚配置

18.3.2 RT-Thread Settings 配置

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

图 17-6 使能 freertos 兼容层

18.4 运行

18.4.1 编译&下载

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

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

18.4.2 运行效果

打开串口工具，系统会自动初始化 USB-Device 为虚拟串口设备；

```
 \ | /
- 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 >
```

图 17-7 pcdc 线程信息

接着我们打开虚拟的串口设备，并测试字符输入；

图 17-8 usb pcdc 测试

18.5 注意事项

暂无

18.6 引用参考

第 19 章 EtherCAT-EOE 例程

19.1 简介

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

以下是 EoE 的主要特点和功能：

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

工控机）。

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

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

19.2 前期准备

软件环境：

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

硬件环境：

- EtherKit 开发板
- 网线一根
- Jlink 调试器

19.3 TwinCAT3 配置

在启动 TwinCAT3 之前，我们还需要做一些配置操作：

19.3.1 安装 ESI 文件

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

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

图 21-1 安装 ESI 文件

19.3.2 添加 TwinCAT 网卡驱动

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

图 21-2 查看支持的以太网适配器

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

图 21-3 安装 TwinCAT 网卡驱动

检查网络适配器，可以看到已经成功安装了

图 21-4 检查 TwinCAT 网卡驱动

19.4 FSP 及 Studio 配置

19.4.1 FSP 配置

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

图 21-5 打开 fsp 配置

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

图 21-6 添加 ethercat_ssc_port 外设

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

图 21-7 ethercat_ssc_port 配置

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

图 21-8 PHY 配置

网卡引脚参数配置，选择操作模式为 RGMII：

图 21-9 ETH 引脚配置

ETHER_ESC 设置：

图 21-10 ESC 引脚配置

ETHER_GMAC 配置：

图 21-11 GMAC 引脚配置

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

图 21-12 添加 CMT 定时器

添加 Ethernet 外设：

图 21-13 添加 Ethernet 外设

ethernet 中断触发回调设置为：user_ether0_callback

图 21-14 ETH 中断回调

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

19.4.2 RT-Thread Studio 配置

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

图 21-15 settings 使能 EOE

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

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

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

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

图 21-16 EOE 串口日志

19.5 EtherCAT EOE 配置

19.5.1 新建 TwinCAT 工程

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

图 21-17 新建 TwinCAT 工程

19.5.2 从站启动 EOE App

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

图 21-18 从站 EOE 启动

19.5.3 从站设备扫描

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

图 21-19 扫描设备

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

19.5.4 更新 EEPROM 固件

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

图 21-20 TwinCAT 配置界面

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

图 21-21 Advanced Settings

这里按图示点击 Download from List…：

图 21-22 Hex Editor

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

图 21-23 ESI 固件下载

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

图 21-24 移除 Device

19.6 EtherCAT EOE 通信

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

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

图 21-25 EtherCAT 状态灯

19.6.1 EIO 测试

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

图 21-26 EtherCAT 从站输出

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

图 21-27 EIO 控制从站 LED

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

图 21-28 EIO 控制从站 LED

19.6.2 EOE 测试

打开以太网适配器，选择主站所使用的适配器并设置静态 IP：

图 19-29 主站 IP 设置

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

图 19-30 从站 IP 设置

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

- 主站 IP：192.168.10.99
- 从站 IP：192.168.10.100

图 19-31 主从站使用 EOE 进行 ping 测试

第 20 章 PROFIENT 例程

20.1 简介

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

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

在本示例中将使用 P-Net 软件包来实现 PROFINET 主从站通信。

20.2 前期准备

软件环境：

- CODESYS（profinet 主站模拟）
    - CODESYS
    - CODESYS Gateway（网关设备）
    - CODESYS Control Win SysTray（PLC 设备）硬件环境：
- EtherKit 开发板

20.3 软件包配置

打开 bsp 下的 env 界面，进入 ->RT-Thread online packages->IoT，找到 [*] P-Net stack for Profinet device implementation ---> 使能，同时留出配置提供给用户：

```
-*- Default netif name for P-NET  --->
```

```
    -> (e00) default ethernet interface name for p-net app, default as 'e
00
-*- Enable P-NET sample board config  --->
    -> (0x0209) p-ent user led pin
    -> (0x0005) p-net user key pin
-*- Default root filesystem path for P-NET  --->
    -> [*] p-net using ramfs filesystem by default, or you can turn this
off and choose another way to enable the filesystem
    -> (8192)  default memory size for ramfs
-*- P-NET sample slave network ip config  --->
    -> (192.168.137.196) set static ip address for profinet slaver
    -> (192.168.137.1) set static gateway address for profinet slaver
    -> (255.255.255.0) set static mask address for profinet slaver
    version (latest)  --->
```

- **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**）

完成上述配置后，将程序编译下载至开发板。

20.4 网络配置

我们使用一根网线连接开发板与 PC，同时在 PC 端配置静态 IP：

图 24-1 以太网静态 IP 配置

检查开发板端的 IP 信息，并测试联通性：

```
 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Nov  1 2024 16:09:58
 2006 - 2024 Copyright by RT-Thread team
lwIP-2.1.2 initialized!
[I/sal.skt] Socket Abstraction Layer initialize success.

Hello RT-Thread!
msh />[I/DBG] link up

msh />ifcon
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: 192.168.137.196
gw address: 192.168.137.1
net mask  : 255.255.255.0
dns server #0: 0.0.0.0
dns server #1: 0.0.0.0
msh />ping 192.168.137.1
ping: not found specified netif, using default netdev e0.
60 bytes from 192.168.137.1 icmp_seq=0 ttl=128 time=0 ms
60 bytes from 192.168.137.1 icmp_seq=1 ttl=128 time=0 ms
60 bytes from 192.168.137.1 icmp_seq=2 ttl=128 time=0 ms
60 bytes from 192.168.137.1 icmp_seq=3 ttl=128 time=0 ms
msh />
```

图 24-2 开发板端 IP 信息

20.5 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。

20.5.1 CODESYS 创建标准工程

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

图 20-3 CODESYS 软件

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

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

图 20-4 新建工程

弹出下面这个弹窗后保持默认配置点击确定：

图 20-5 创建对象

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

创建成功后就可以看到主界面了：

图 20-6 CODESYS 主界面

20.5.2 Gateway 及 软 PLC 启动

依次打开下面两个软件：

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

图 20-7 启动 CODESYS 软件

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

图 20-8 扫描网关设备

弹出设备用户登录窗口后，配置用户名和密码（用户自定义）：

图 20-9 设备用户登录

检查网关设备及软 PLC 设备是否在线：

图 20-10 查看网关设备及软 PLC 设备在线

20.5.3 profinet GSDML 文件添加

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

本项目的 GSDML 文件位于如下路径：

- ..\src\ports\rtthread\pn_dev

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

图 20-11 安装 XML 文件

安装成功后可以看到 p-net 从站描述文件：

图 20-12 P-Net 描述文件安装

20.5.4 设备添加

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

配器；

图 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

图 20-14 PROFINET IO 主站添加

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

图 20-13 PROFINET IO 从站添加

20.5.5 任务响应

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

图 20-14 Main Tasks

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

图 20-15 Profinet_CommunicationTask

20.5.6 网络配置

- Ethernet 配置：双击左侧导航栏中的 Ethernet(Ethernet) -> 通用，修改网络接口为连接到开发板的以太网端口（这里由于我开启了 PRONETA，所以在同一网段下分配了两个主站 IP，这里需要注意选择正确的那一个）

图 20-16 Ethernet 配置

- PN_Controller 配置：双击左侧导航栏 PN_Controller(PN-Controller) -> 通用，并正确修改默认从站 IP 参数的区间，根据提示修改即可。

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

图 20-17 P-Net 从站网络配置

```
msh />pnet_app
RAM file system initialized!

** Starting P-Net sample application 0.2.0+v0.2.0-43-g0e48fbf **
Number of slots:       7 (incl slot for DAP module)
P-net log level:       4 (DEBUG=0, FATAL=4)
App log level:         0 (DEBUG=0, FATAL=4)
Max number of ports:   1
Network interfaces:    e00
Default station name:  rt-labs-dev   PN从站名称
Management port:       e00
Physical port [1]:     e00
Hostname:              rtthread_6530
IP address:            192.168.137.196
Netmask:               255.255.255.0
Gateway:               192.168.137.1
Init P-Net stack and sample application
Profinet signal LED indication. New state: 0
Start sample application main loop
```

图 20-18 PNIO 网络信息

20.5.7 工程编译并启动调试

- step1：工程上方导航栏选择 编译-> 生成代码

- step2：选择 在线 -> 登录
- step3：点击 调试 -> 启动

此时就可以看到 PN 主站已经上线成功

图 20-19 PN 上线成功

20.6 profinet 从站应用启动

开发板端启动 PN 从站，执行命令：pnet_app：

```
msh />pnet app
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
Management port:       e00 00:11:22:33:44:55
Physical port [1]:     e00 00:11:22:33:44:55
Hostname:              rtthread_6530
IP address:            192.168.137.196
Netmask:               255.255.255.0
Gateway:               192.168.137.1
Init P-Net stack and sample application
Profinet signal LED indication. New state: 0
Start sample application main loop

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

msh />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
```

初始化ramfs文件系统成功

当前PN日志等级

从站站点名称

从站网络信息

主从站响应成功

图 20-20 PNIO 响应日志

图 20-21 PN 从站状态

20.7 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。

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

图 20-22 闪烁 PNIO LED

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

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

图 20-23 PNIO 日志

第 21 章 FAQ

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

21.1 芯片状态异常

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

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

图 21-1 拨码开关

打开 Segger J-Link Commander，依次按下图所示操作；

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

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


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

图 21-2 连接 Jlink commander

下面执行如下命令清除 flash，如下所示：

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

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

图 21-3 清除 flash

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

新上电即可再次下载程序。