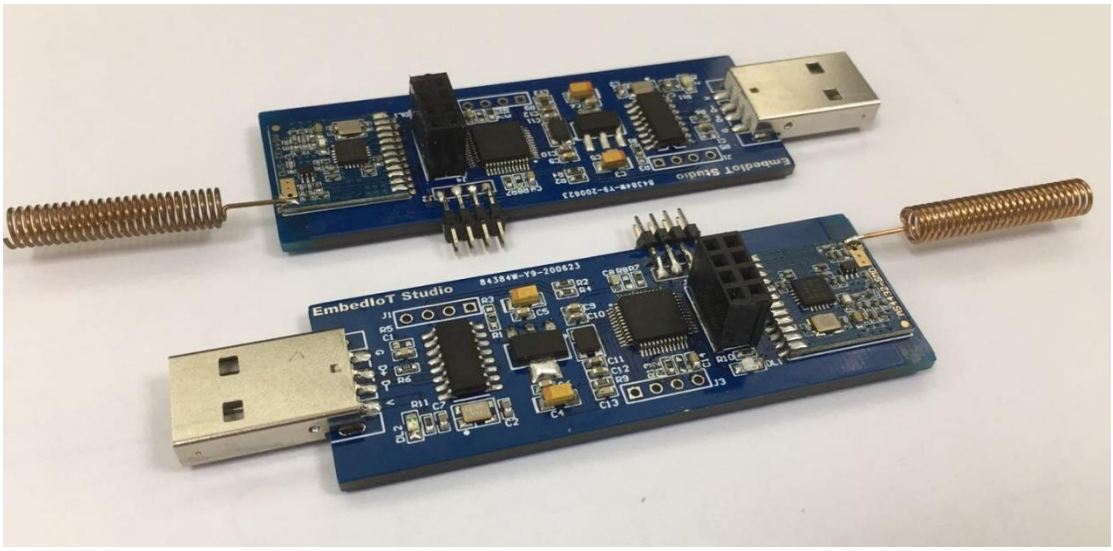
嵌入式物联网应用开发 – ARM 中控扩展模块

为了让 ARM-Linux 开发板具有 2.4GHz 和 433MHz 无线通信功能，微联智控工作室设计了一款通用的 ARM 中控扩展模块，该模块的实物图片，如下图所示。



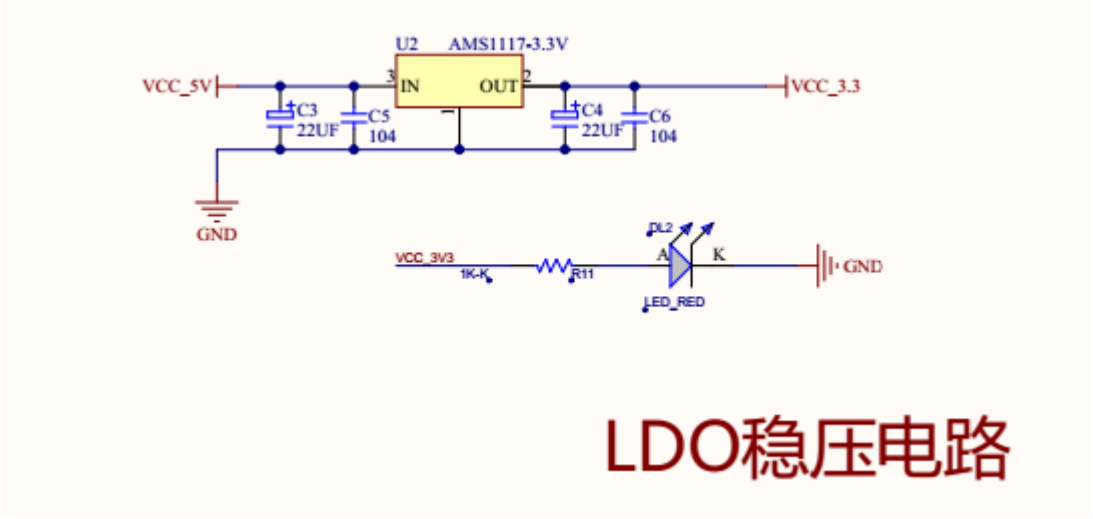
这款 ARM 中控扩展模块，主要使用了 USB-TTL 芯片和 STM32 单片机进行通信，再通过 SPI 接口外接 si4432 模块和 nRF24L01+PA+LAN 模块。使用这种方式，就可以避免移植 si4432 和 nRF24L01 的驱动到 ARM-Linux 开发板，大大降低了开发工作量和开发难度。

在 ARM-Linux 开发板或者在 PC 电脑上使用这款模块，只需要简单地在开发板移植或在 PC 机上安装 CH340G 的芯片驱动，就可以把这款模块当做串口使用。

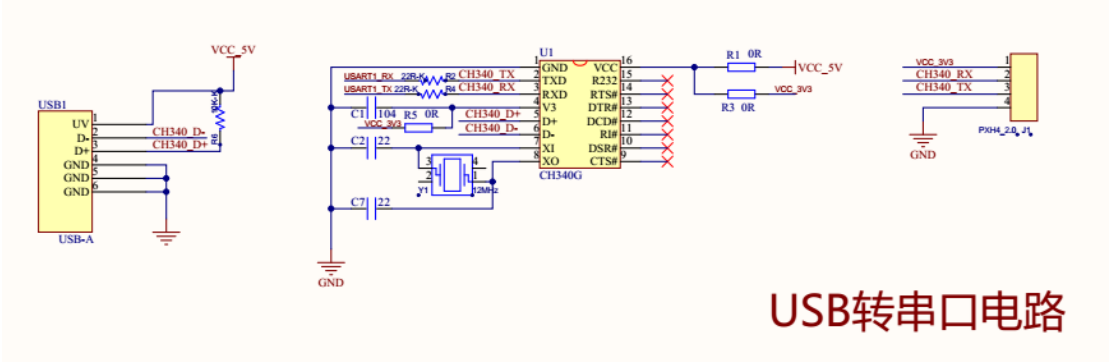
ARM-Linux 中控扩展模块具有以下功能：
1、USB 转串口功能，使用 CH340G 芯片，可以方便对接电脑和嵌入式 Linux 开发板。
2、引出 CH340G 的 TTL 接口，可以用作一个普通的 USB 转串口模块（去掉 R2 和 R4 电阻）。
3、引出单片机的 UART2 串口，TTL 电平，方便用户扩展（提供源码）。
4、板载 433MHz 无线通信模块，与单片机通过 SPI 接口进行通信，用户可编程。
5、引出 SPI 接口，对接市面上大部分 SPI 接口的 2.4GHz 通信模块，用户可编程。
6、模块尺寸（不含 USB 接口）：70mm * 25mm

硬件设计篇

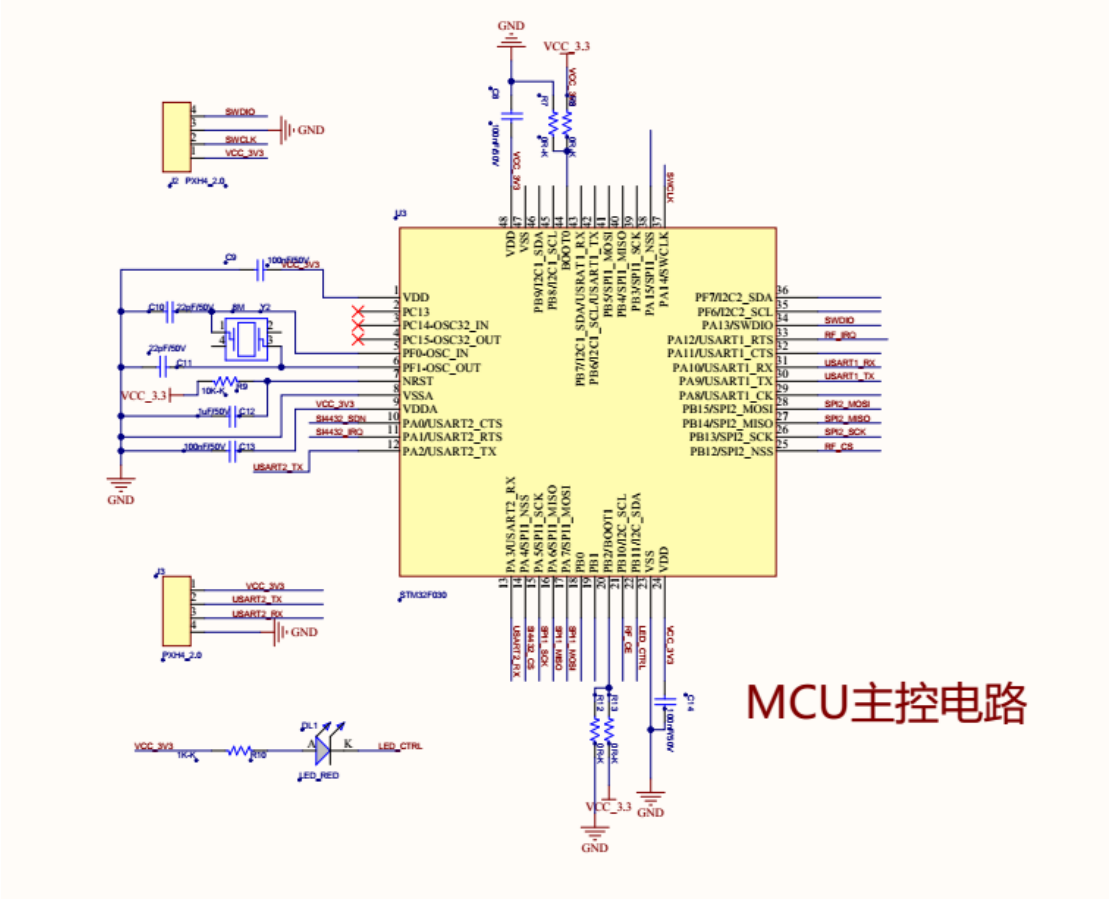
ARM 中控扩展模块使用 USB 接口进行 5V 供电，模块自带稳压芯片，5V 电源通过 LDO 稳压电路，为单片机及其他模块提供 3.3V 电源，模块还自带有红色的电源指示灯用来表示模块的通电情况，如下图所示。



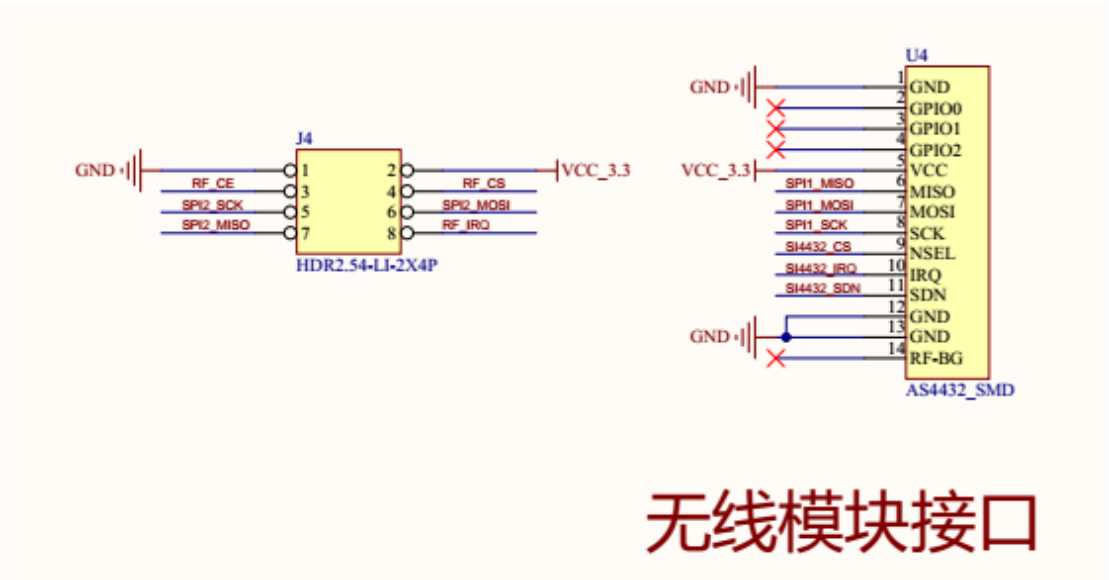
USB 转串口电路使用沁恒科技的 CH340G 作为转换芯片，该芯片可以通过 3.3V 或 5V 电源供电，对外引出芯片的 TTL 接口（使用时需要去掉 R2 和 R4 电阻），方便开发者扩展，USB 转串口电路如下图所示。



ARM 中控扩展模块采用 STM32F030C8T6 作为主控芯片，这款 MCU 芯片是 Cortex-M0 内核，主频 48MHz，芯片自带 64KB 可编程 Flash 和 8KB RAM，很适合物联网终端设备的控制器开发。模块的 MCU 控制电路如下图所示。



ARM 中控扩展模块使用 SPI1 和 SPI2 接口跟无线模块进行通信，其中，SPI1 接口连接 SI4432 这款 433MHz 无线模块，SI4432 模块使用焊接方式固定在中控扩展模块上。SPI2 接口通过提供接口的方式外引出来，可以用来对接 nRF24L01 这款 2.4GHz 无线模块，如下图所示。

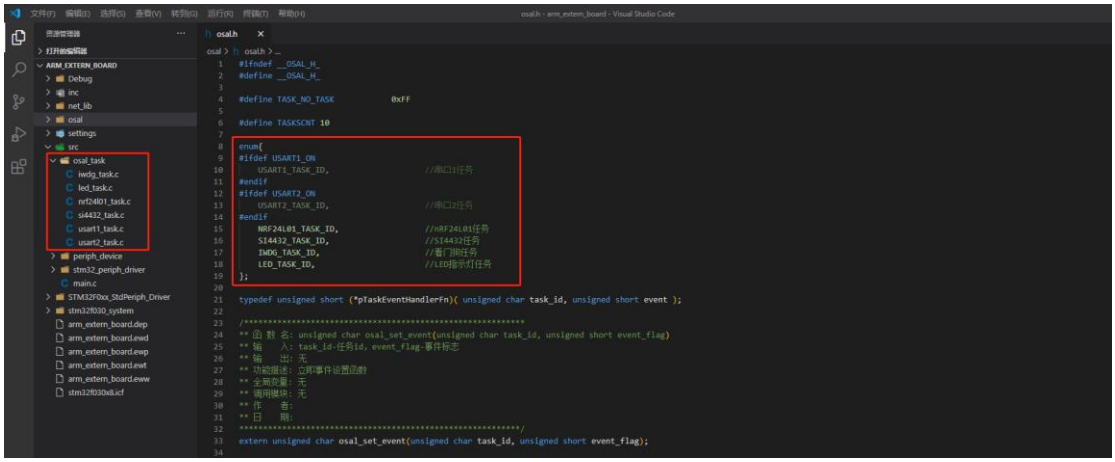


以上就是 ARM 中控扩展模块的原理图描述，各个模块的原理比较简单，具体原理图和 PCB 的硬件工程可以到 GitHub 或 Gitee 上下载。

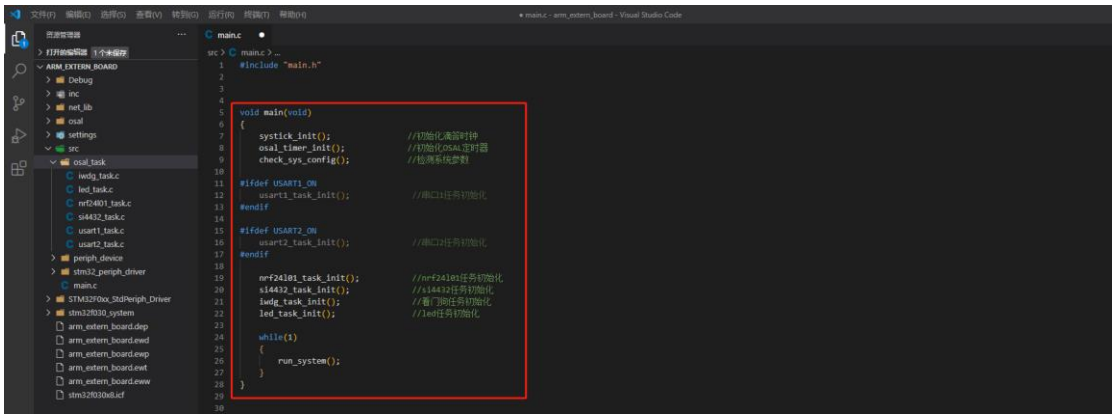
软件设计篇

ARM 中控扩展模块的软件框架，是基于任务和事件的 OSAL 调度器来进行开发，有关 OSAL 调度器的介绍，可以查看以下文章。
开源 | 嵌入式物联网应用开发 - 基于任务和事件的 OSAL 调度器

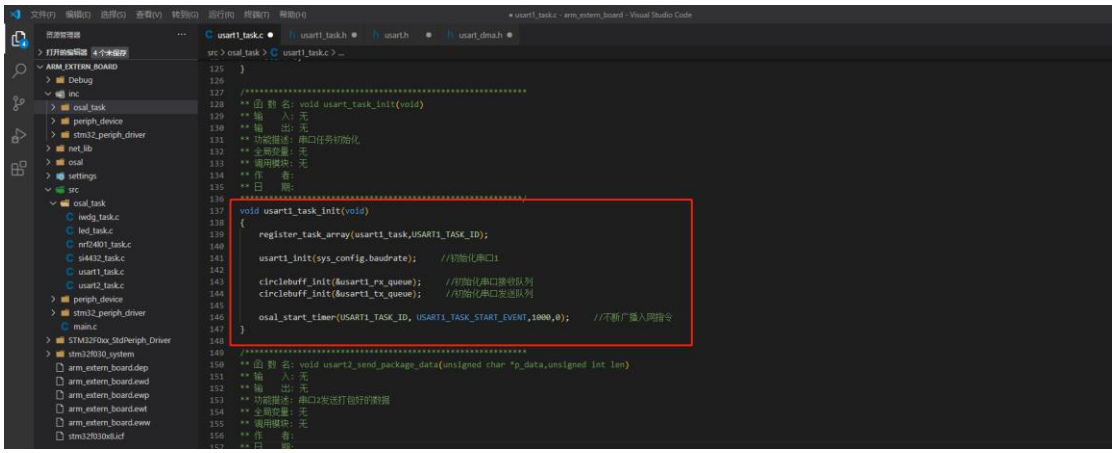
ARM 中控扩展模块，具有串口通信功能，2.4G 和 433 无线通信功能，看门狗系统监测功能，LED 指示灯功能。因此，在软件设计上，把这些功能划分为 6 个任务，在 osal.h 中定义好各个任务的 ID，并创建各个任务的源代码文件，如下图所示。



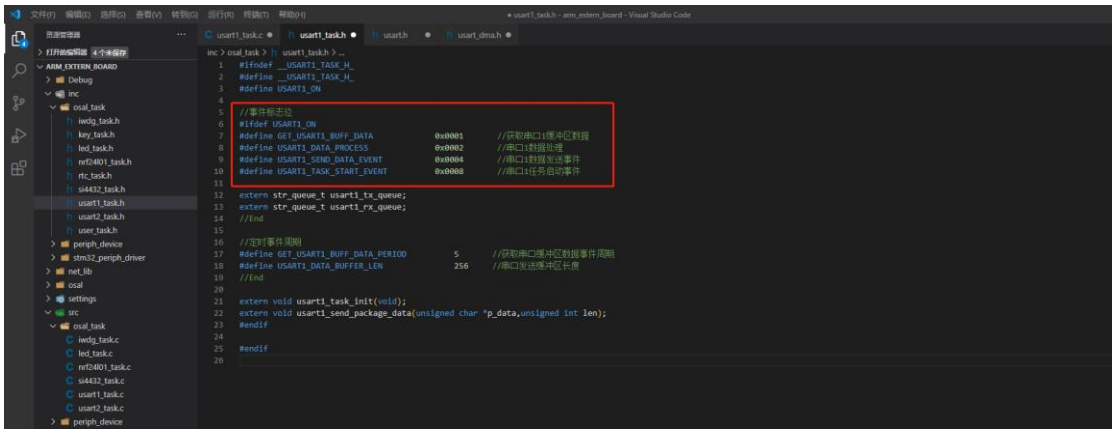
在 main.c 主函数里面，进行 OSAL 调度器和各个任务事件的初始化，由于有些设备不一定用到所有串口，因此，通过 USART1_ON 和 USART2_ON 这两个宏定义作为串口模块的开关，主函数的内容如下图所示。



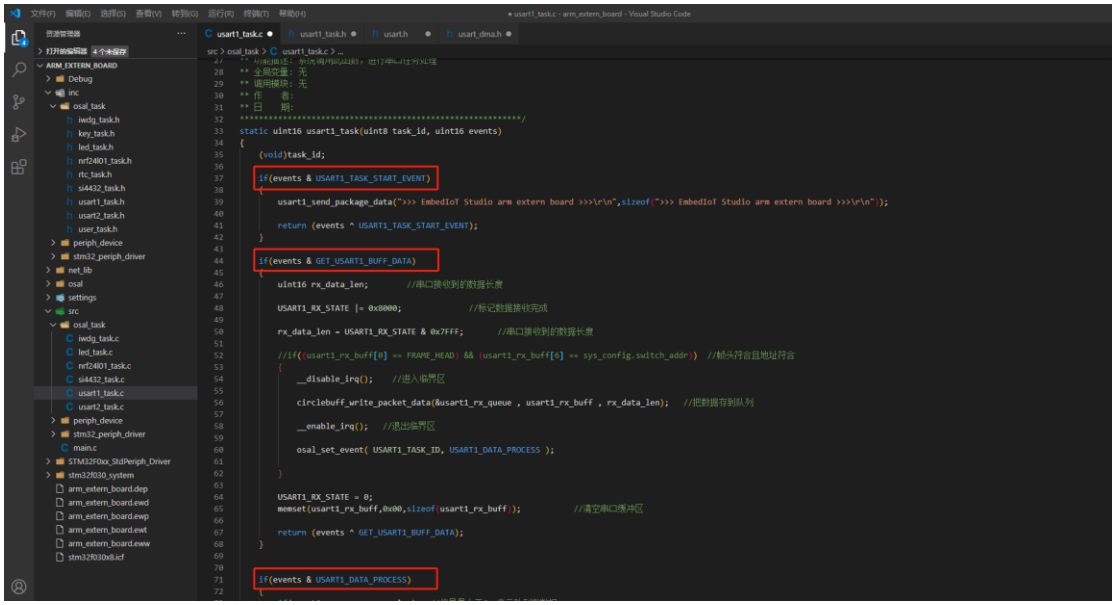
先来看看串口任务处理，ARM 中控扩展模块主要用到了串口 1 以及 DMA 通信。所以，需要在工程的宏定义那里定义 USART1_ON 宏和 USART_DMA 宏，以使能串口 1 的 DMA 通信方式。在源文件 usart1_task.c 中，对串口任务进行了注册和串口外设初始化，并且同时初始化了两个队列数组，用来缓冲串口的发送数据和接收数据，如下图所示。



在头文件 usart1_task.h 中，定义串口 1 任务相关的事件，主要有串口 1 数据收发和处理，串口任务启动事件，如下图所示。



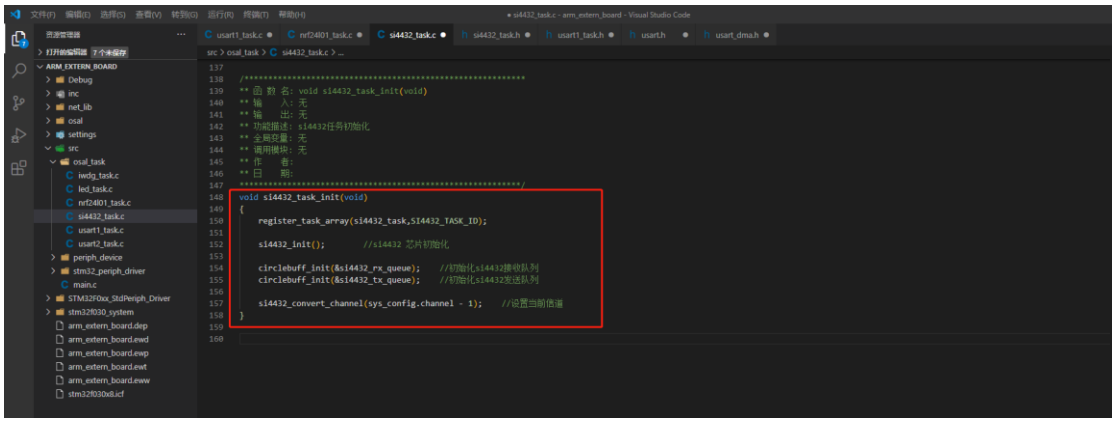
然后在源文件 usart1_task.c 的 usart1_task()函数中，处理各种串口任务事件，OSAL 调度器在收到串口的各种事件后，都会在这个函数里面处理各种事件，如下图所示。



（限于篇幅，这里只列出部分源码）

对于 SI4432 和 nRF24L01 这两款无线模块的任务处理，其任务事件的处理框架都是大同小异的，这里以 SI4432 为例进行说明。

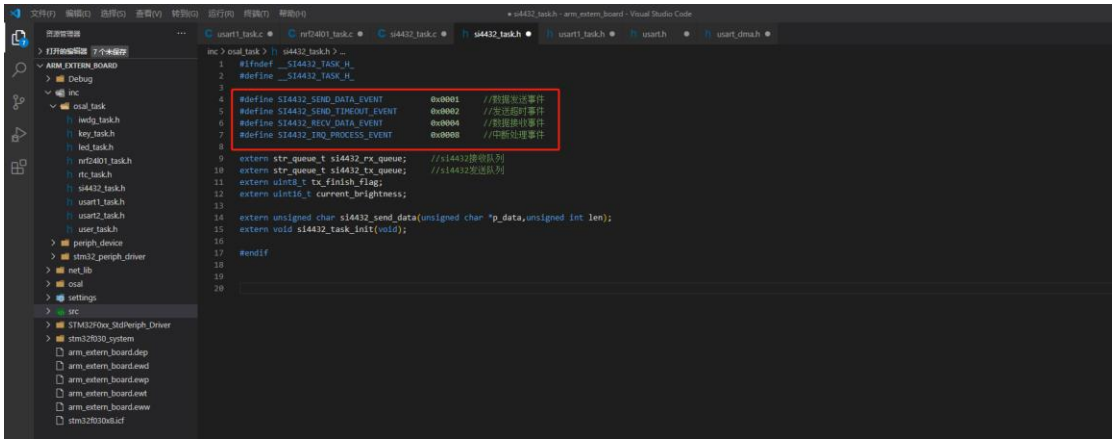
先来看看源文件 si4432_task.c 里面的任务初始化函数，如下图所示。



```
/*********************************************************
** 函 数 名: void si4432_task_init(void)
** 输    入: 无
** 输    出: 无
** 功能描述: si4432任务初始化
** 全局变量: 无
** 调用模块: 无
** 作    者:
** 日    期:
*********************************************************/
void si4432_task_init(void)
{
    register_task_array(si4432_task,SI4432_TASK_ID);

    si4432_init();        //si4432 芯片初始化

    circlebuff_init(&si4432_rx_queue);    //初始化si4432接收队列
    circlebuff_init(&si4432_tx_queue);    //初始化si4432发送队列

    si4432_convert_channel(sys_config.channel - 1);    //设置当前信道
}
```

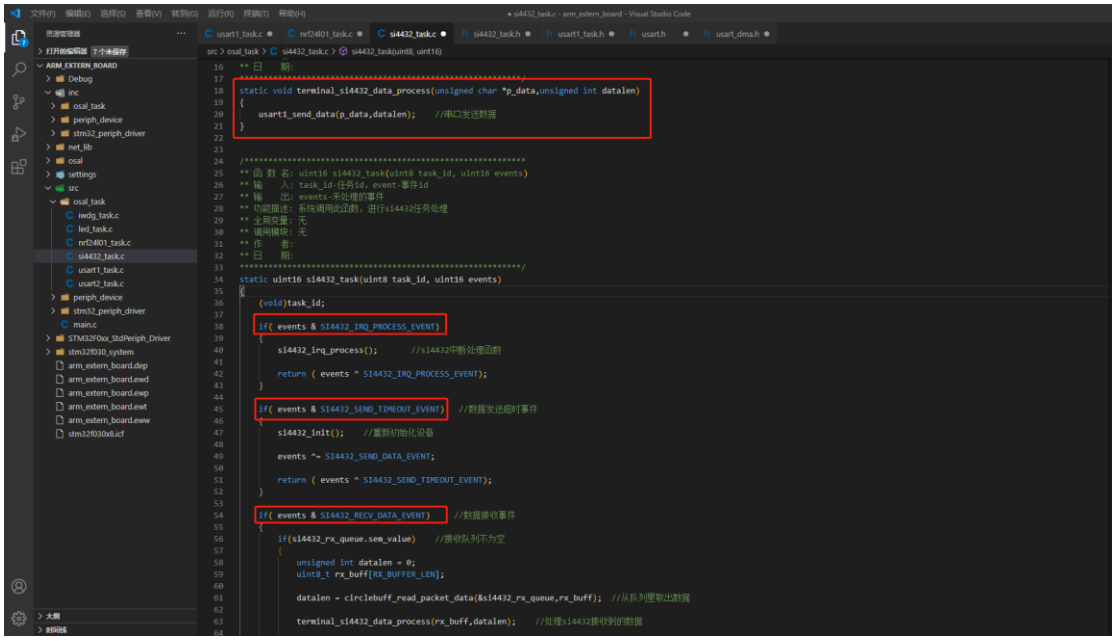
在初始化函数 si4432_task_init()中，先对 si4432 芯片的连接引脚和芯片寄存器进行初始化，然后再初始化 si4432 的数据发送和接收队列，然后再根据系统设置的参数，进行无线通信信道设置。

在头文件 si4432_task.h 中，定义了各种任务事件，主要有数据发送和接收，超时处理，以及中断事件，如下图所示。



```
#ifndef __SI4432_TASK_H_
#define __SI4432_TASK_H_

#define SI4432_SEND_DATA_EVENT          0x0001    //数据发送事件
#define SI4432_SEND_TIMEOUT_EVENT       0x0002    //发送超时事件
#define SI4432_RECV_DATA_EVENT          0x0004    //数据接收事件
#define SI4432_IRQ_PROCESS_EVENT        0x0008    //中断处理事件

extern str_queue_t si4432_rx_queue;     //si4432接收队列
extern str_queue_t si4432_tx_queue;     //si4432发送队列
extern uint8_t tx_finish_flag;
extern uint16_t current_brightness;

extern unsigned char si4432_send_data(unsigned char *p_data,unsigned int len);
extern void si4432_task_init(void);

#endif
```

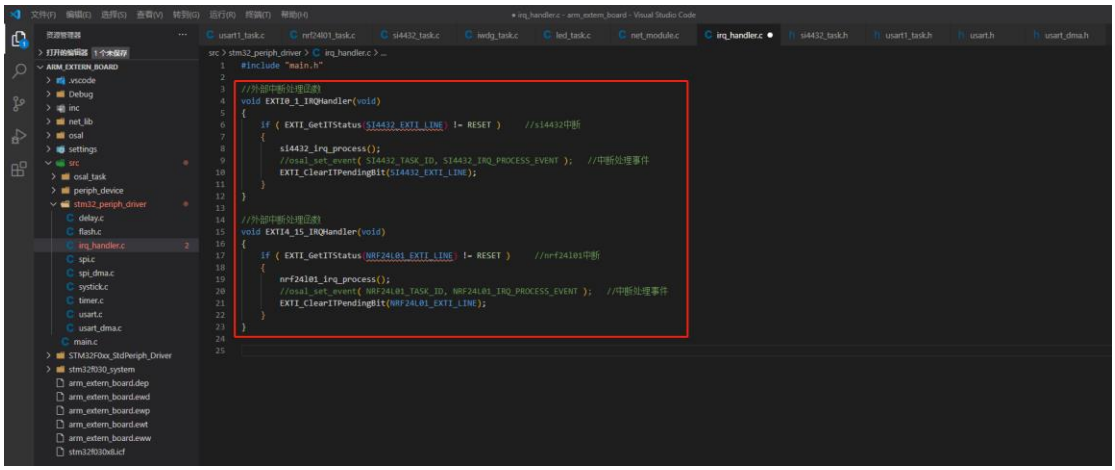
各种任务事件是在 si4432_task()这个函数中进行处理的，对于 SI4432 无线模块接收到的数据，目前都是通过串口透传出去，交给上位机进行处理，各个任务事件的函数内容，如下图所示。



```
/*********************************************************
** 函 数 名: uint16 si4432_task(uint8 task_id, uint16 events)
** 输    入: task_id-任务id, event-事件id
** 输    出: events-未处理的事件
** 功能描述: 系统调用此函数，进行si4432任务处理
** 全局变量: 无
** 调用模块: 无
** 作    者:
** 日    期:
*********************************************************/
static uint16 si4432_task(uint8 task_id, uint16 events)
{
    (void)task_id;

    if( events & SI4432_IRQ_PROCESS_EVENT)
    {
        si4432_irq_process();       //si4432中断处理函数

        return ( events ^ SI4432_IRQ_PROCESS_EVENT);
    }

    if( events & SI4432_SEND_TIMEOUT_EVENT)   //数据发送超时事件
    {
        si4432_init();    //重新初始化设备

        events ^= SI4432_SEND_DATA_EVENT;

        return ( events ^ SI4432_SEND_TIMEOUT_EVENT);
    }

    if( events & SI4432_RECV_DATA_EVENT)    //数据接收事件
    {
        if(si4432_rx_queue.sem_value)    //接收队列不为空
        {
            unsigned int datalen = 0;
            uint8_t rx_buff[RX_BUFFER_LEN];

            datalen = circlebuff_read_packet_data(&si4432_rx_queue,rx_buff);  //从队列里取出数据

            terminal_si4432_data_process(rx_buff,datalen);    //处理si4432接收到的数据
```

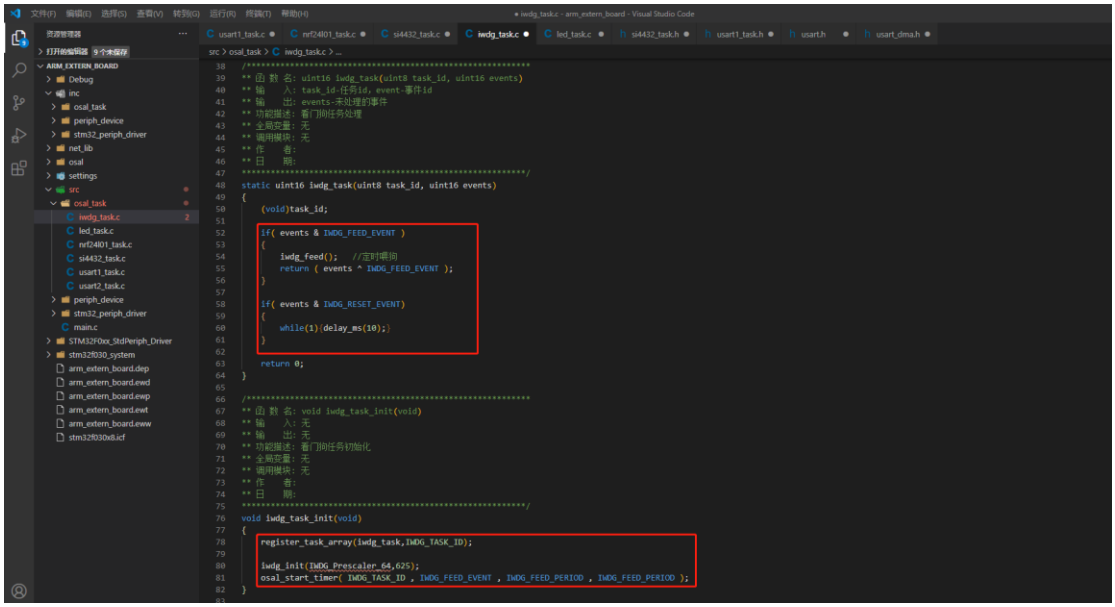
（限于篇幅，这里只列出部分源码）

由硬件原理图可知，不管是 SI4432 或 nRF24L01 模块，一旦接收到无线数据，都是通过引脚中断的方式来通知 STM32 进行处理的，因此，需要在 STM32 的外部中断引脚处理函数中，取出无线模块接收到的数据，如下图所示。



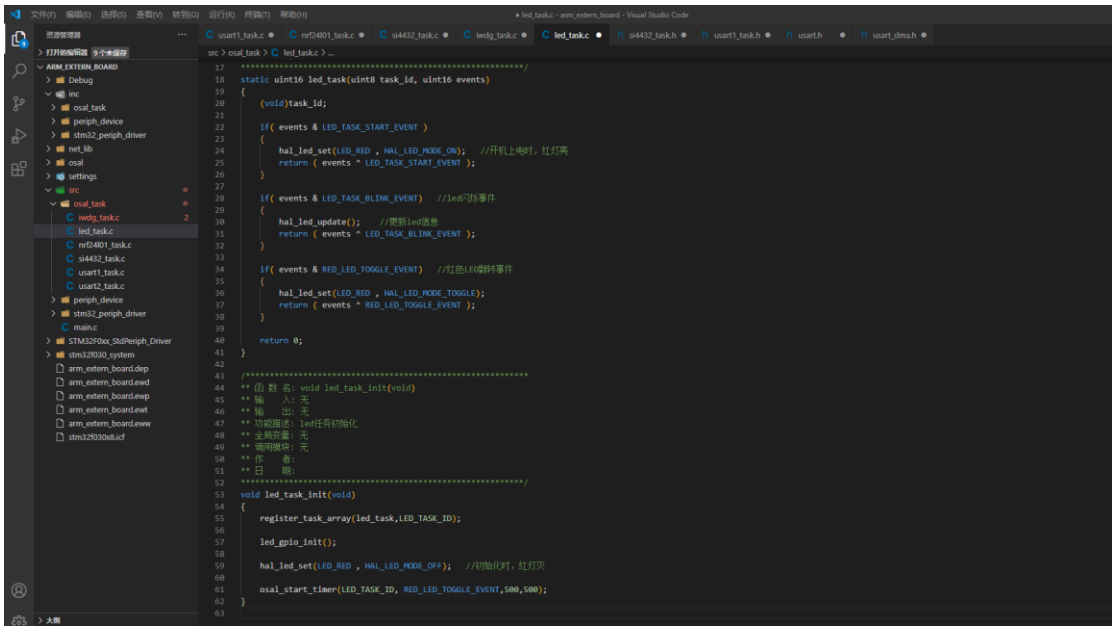
```
#include "main.h"

//外部中断处理函数
void EXTI0_1_IRQHandler(void)
{
    if ( EXTI_GetITStatus(SI4432_EXTI_LINE) != RESET )    //si4432中断
    {
        si4432_irq_process();
        //osal_set_event( SI4432_TASK_ID, SI4432_IRQ_PROCESS_EVENT );  //中断处理事件
        EXTI_ClearITPendingBit(SI4432_EXTI_LINE);
    }
}

//外部中断处理函数
void EXTI4_15_IRQHandler(void)
{
    if ( EXTI_GetITStatus(NRF24L01_EXTI_LINE) != RESET )    //nrf24l01中断
    {
        nrf24l01_irq_process();
        //osal_set_event( NRF24L01_TASK_ID, NRF24L01_IRQ_PROCESS_EVENT );  //中断处理事件
        EXTI_ClearITPendingBit(NRF24L01_EXTI_LINE);
    }
}
```

看门狗任务是用来监测整个系统的运行情况的，一旦发现系统运行异常，看门狗由于喂狗不及时，系统就会进行超时重启。因此，看门狗任务比较简单，主要是初始化 STM32 的看门狗外设，并设置一个定时喂狗任务，定时进行看门狗计数器复位操作，防止系统复位。与此同时，也可以利用看门狗超时的特性，来定义一个复位任务，方便一些系统重启操作，如下图所示。



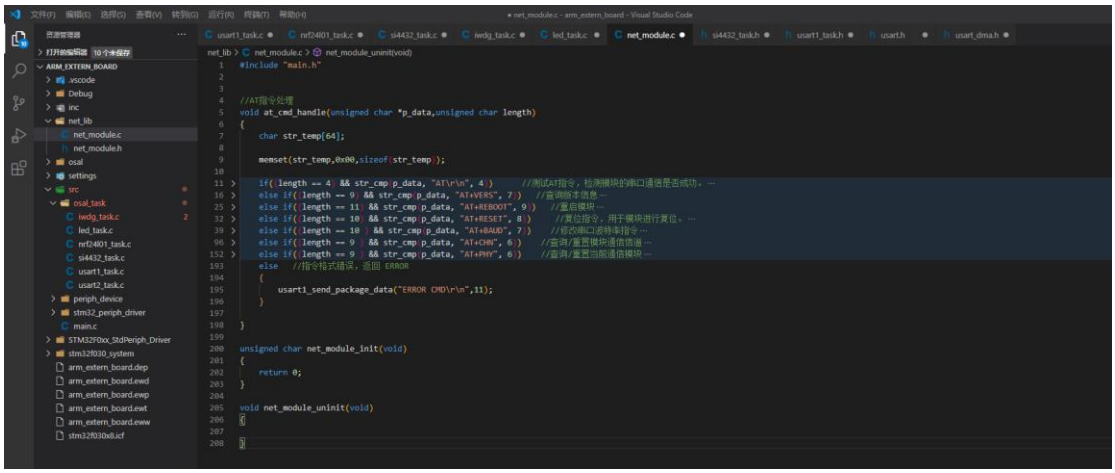
```c
/*********************************************************
** 函 数 名: uint16 iwdg_task(uint8 task_id, uint16 events)
** 输    入: task_id-任务id, event-事件id
** 输    出: events-未处理的事件
** 功能描述: 看门狗任务处理
** 全局变量: 无
** 调用模块: 无
** 作    者:
** 日    期:
*********************************************************/
static uint16 iwdg_task(uint8 task_id, uint16 events)
{
    (void)task_id;

    if( events & IWDG_FEED_EVENT )
    {
        iwdg_feed();  //定时喂狗
        return ( events ^ IWDG_FEED_EVENT );
    }

    if( events & IWDG_RESET_EVENT)
    {
        while(1){delay_ms(10);}
    }

    return 0;
}

/*********************************************************
** 函 数 名: void iwdg_task_init(void)
** 输    入: 无
** 输    出: 无
** 功能描述: 看门狗任务初始化
** 全局变量: 无
** 调用模块: 无
** 作    者:
** 日    期:
*********************************************************/
void iwdg_task_init(void)
{
    register_task_array(iwdg_task,IWDG_TASK_ID);

    iwdg_init(IWDG_Prescaler_64,625);
    osal_start_timer( IWDG_TASK_ID , IWDG_FEED_EVENT , IWDG_FEED_PERIOD , IWDG_FEED_PERIOD );
}
```

指示灯任务是用来指示系统无线网络状态的，当无线网络处于连接成功/连接断开/数据广播等不同状态的时候，指示灯会以不同的频率进行闪烁。网络状态指示灯的任务处理，如下图所示。

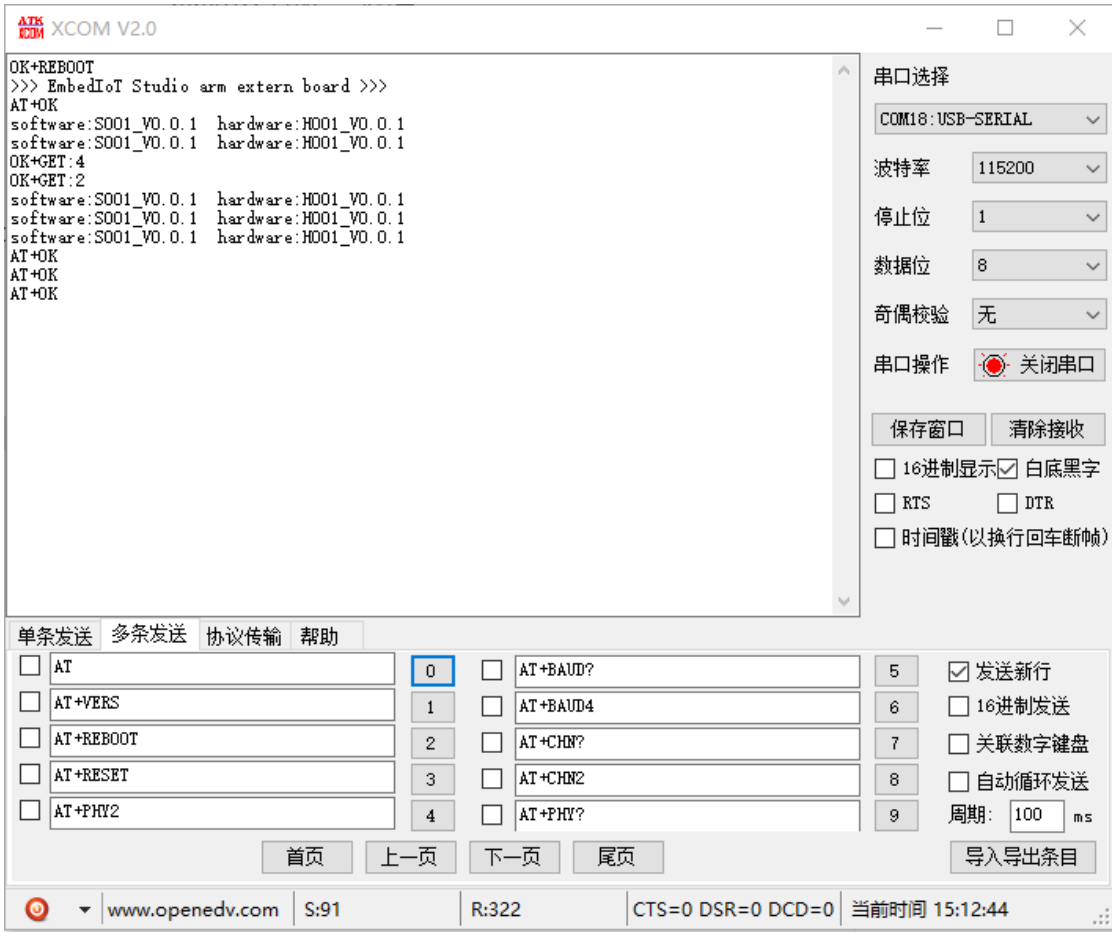


```c
/*********************************************************
** 函 数 名: uint16 led_task(uint8 task_id, uint16 events)
** 输    入: task_id-任务id, event-事件id
** 输    出: events-未处理的事件
** 功能描述: led任务处理
** 全局变量: 无
** 调用模块: 无
** 作    者:
** 日    期:
*********************************************************/
static uint16 led_task(uint8 task_id, uint16 events)
{
    (void)task_id;

    if( events & LED_TASK_START_EVENT )
    {
        hal_led_set(LED_RED , HAL_LED_MODE_ON);  //开机上电时，红灯亮
        return ( events ^ LED_TASK_START_EVENT );
    }

    if( events & LED_TASK_BLINK_EVENT)  //led闪烁事件
    {
        hal_led_update();  //更新led信息
        return ( events ^ LED_TASK_BLINK_EVENT );
    }

    if( events & RED_LED_TOGGLE_EVENT)  //红色LED翻转事件
    {
        hal_led_set(LED_RED , HAL_LED_MODE_TOGGLE);
        return ( events ^ RED_LED_TOGGLE_EVENT );
    }

    return 0;
}

/*********************************************************
** 函 数 名: void led_task_init(void)
** 输    入: 无
** 输    出: 无
** 功能描述: led任务初始化
** 全局变量: 无
** 调用模块: 无
** 作    者:
** 日    期:
*********************************************************/
void led_task_init(void)
{
    register_task_array(led_task,LED_TASK_ID);

    led_gpio_init();

    hal_led_set(LED_RED , HAL_LED_MODE_OFF);  //初始化时，红灯灭

    osal_start_timer(LED_TASK_ID, RED_LED_TOGGLE_EVENT,500,500);
}
```

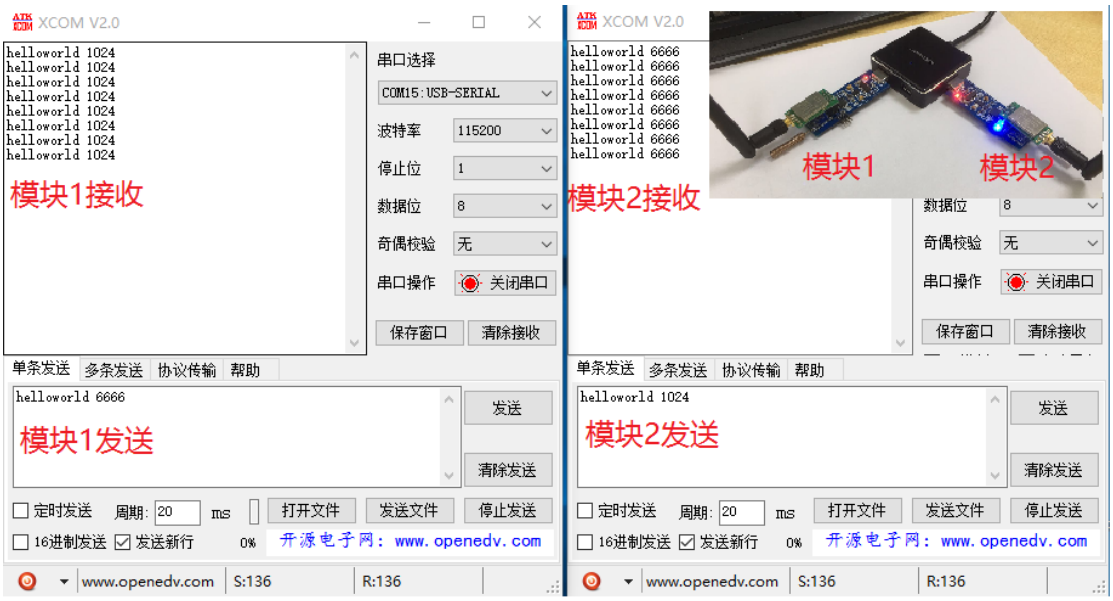
为了方便开发者对 ARM 中控扩展模块进行设置，模块还支持 AT 指令扩展，开发者可以通过 AT 指令对模块进行波特率参数，信道，恢复出厂设置，查询软硬件版本信息等操作，也可以根据实际需要，修改源码扩展自己的 AT 指令集，目前模块支持的 AT 指令如下图所示。



以上就是 ARM 中控模块的软件开发内容，编译源码并下载到模块中运行，可以使用串口工具测试模块的 AT 指令是否工作正常，如下图所示。



使用两个 ARM 中控扩展模块，并同时接到 PC 电脑上，用串口调试工具可以在这两个模块直接互相透传数据，如下图所示。



项目的开源地址：

https://github.com/embediot/Embedded-IoT-Project

https://gitee.com/embediot/Embedded-IoT-Project

感谢阅读！

点击这里，访问作者博客

欢迎关注【微联智控工作室】