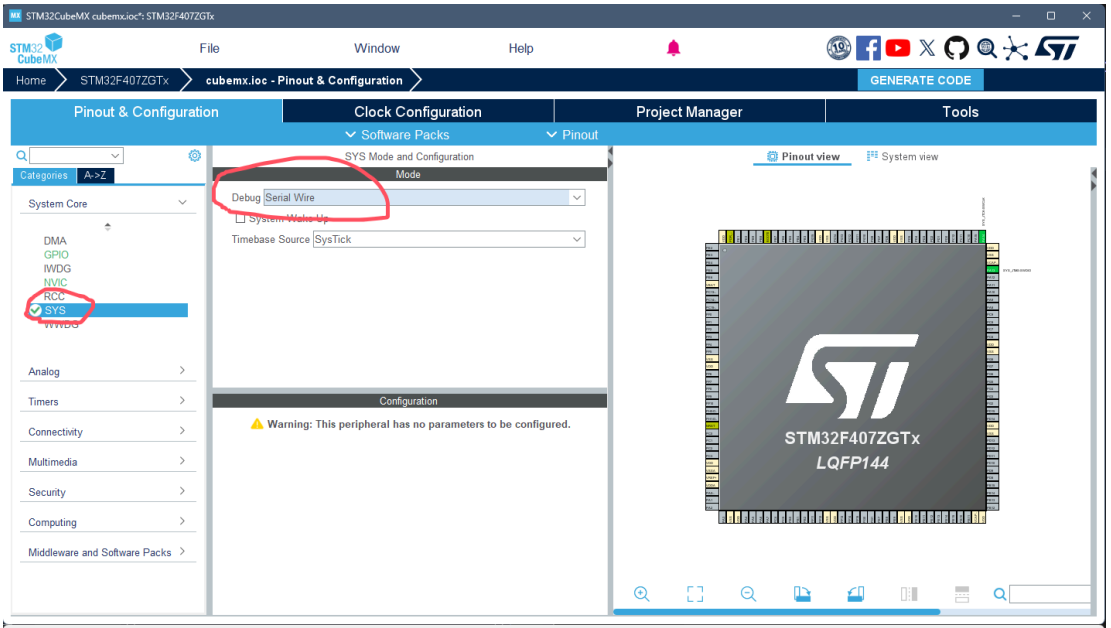
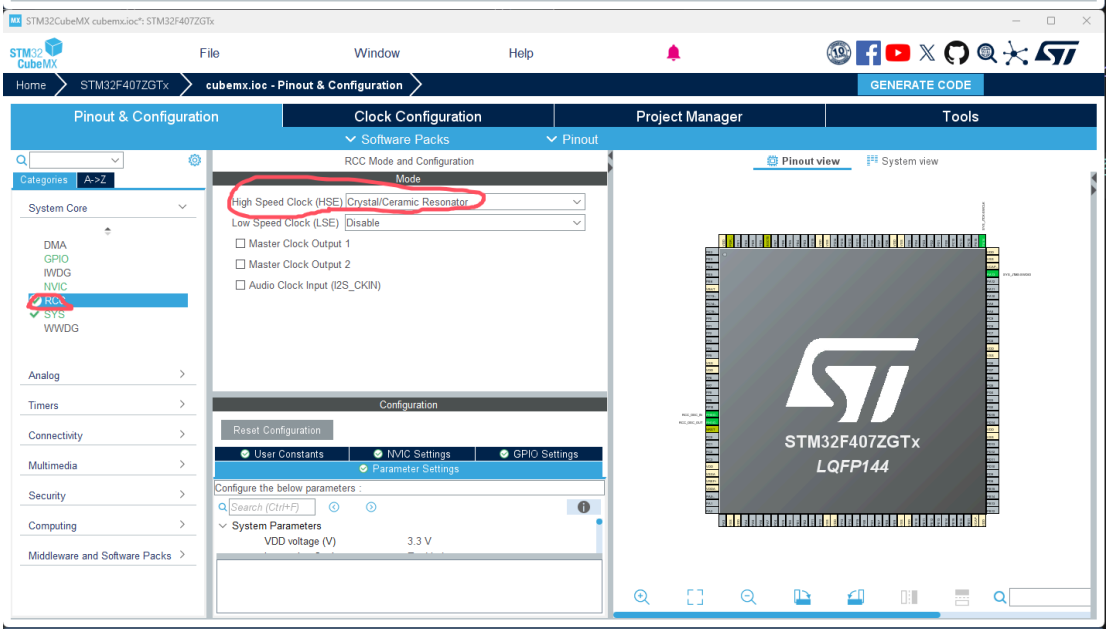
UART设备

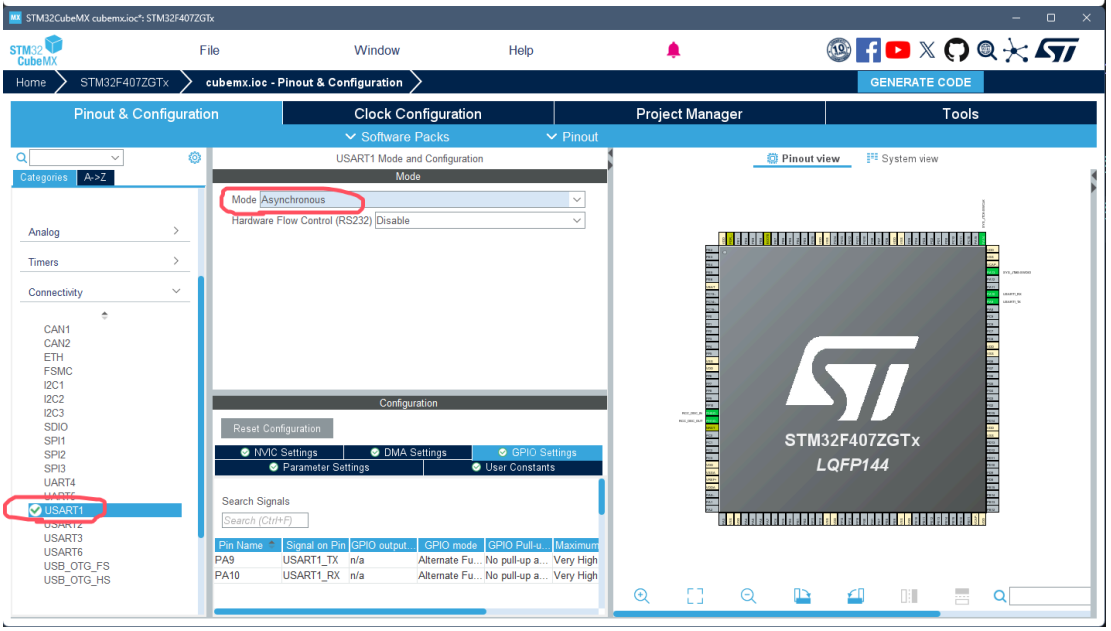
1.串口中断或轮询模式的操作流程

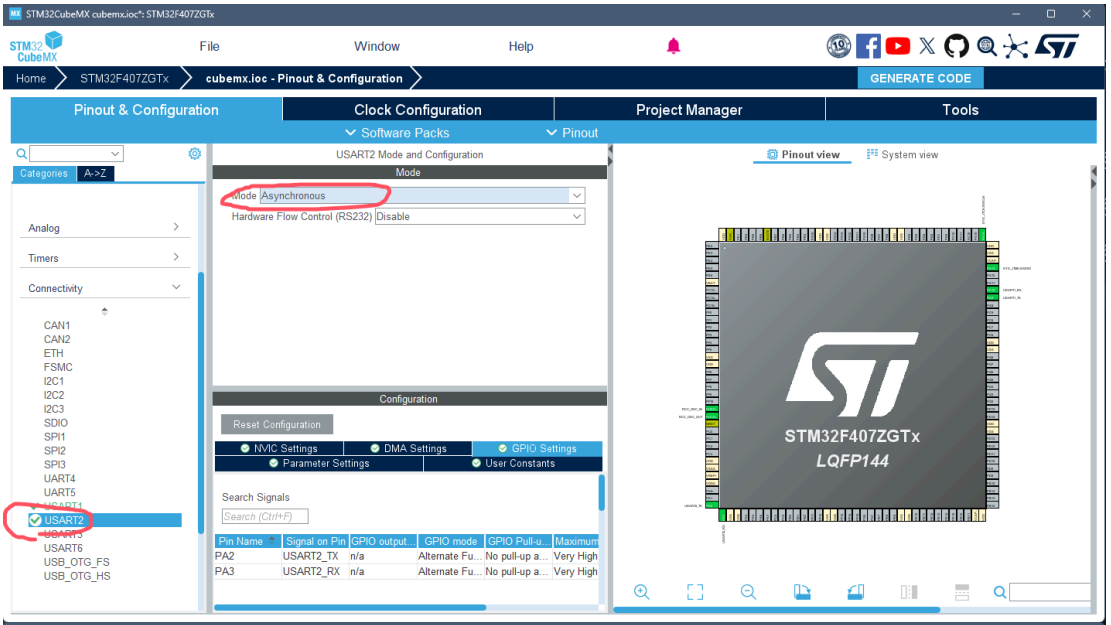
- 1. 新建rtthread项目

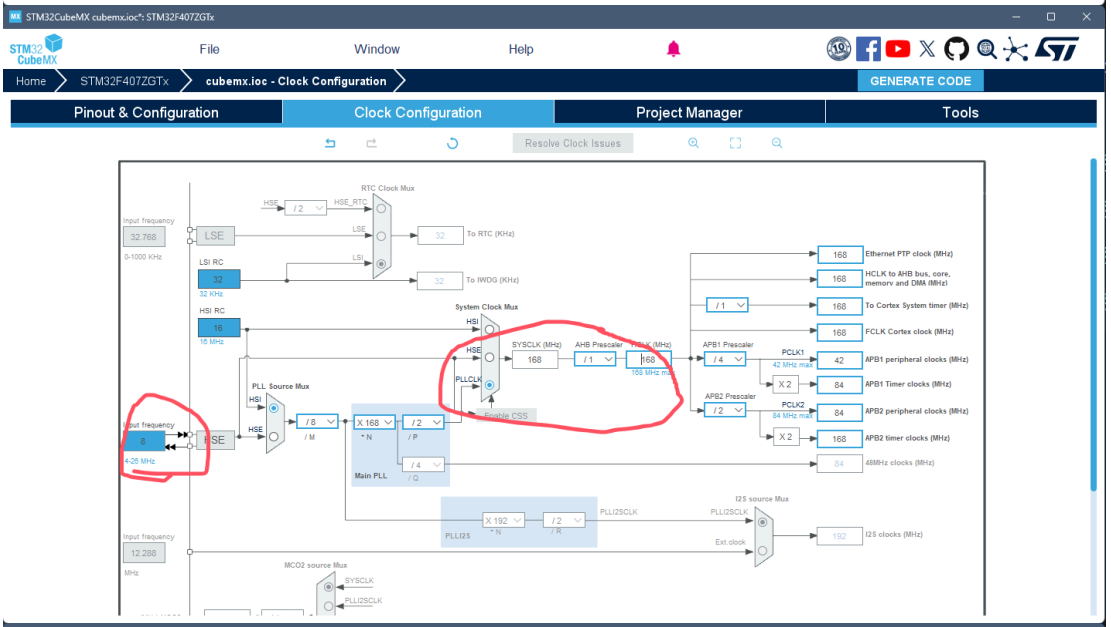
- 2. 使用CubeMx配置

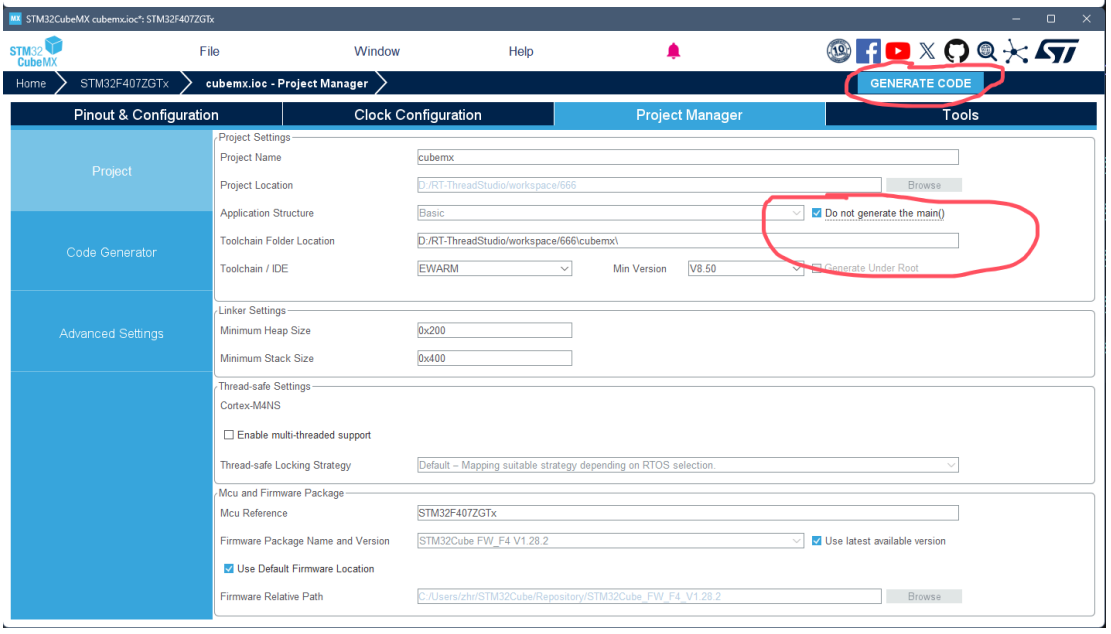




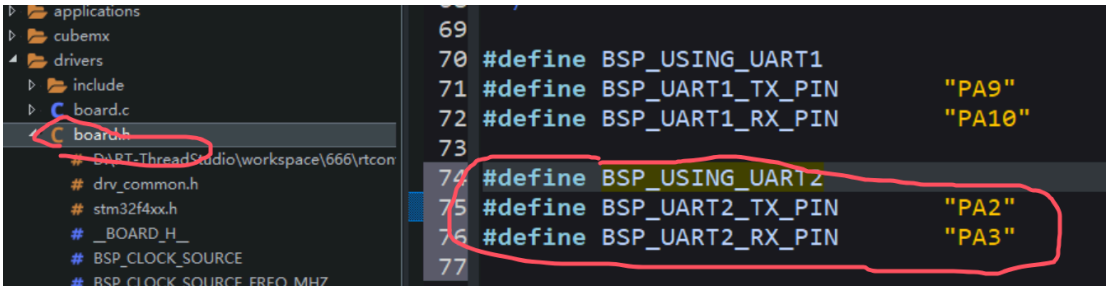



STM32CubeMX cubemx.ioc*: STM32F407ZGTx
File
Window
Help
Home
STM32F407ZGTx
cubemx.ioc - Pinout & Configuration
GENERATE CODE
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Software Packs
Pinout
Categories
A->Z
Analog
Timers
Connectivity
CAN1
CAN2
ETH
FSMC
I2C1
I2C2
I2C3
SDIO
SPI1
SPI2
SPI3
UART4
UART5
USART1
USART2
USART3
USART6
USB_OTG_FS
USB_OTG_HS
USART2 Mode and Configuration
Mode
Mode Asynchronous
Hardware Flow Control (RS232) Disable
Configuration
Reset Configuration
NVIC Settings
DMA Settings
GPIO Settings
Parameter Settings
User Constants
Search Signals
Search (Ctrl+F)
Pin Name
Signal on Pin
GPIO output...
GPIO mode
GPIO Pull-u...
Maximum
PA2
USART2_TX
n/a
Alternate Fu...
No pull-up a...
Very High
PA3
USART2_RX
n/a
Alternate Fu...
No pull-up a...
Very High
Pinout view
System view
STM32F407ZGTx
LQFP144

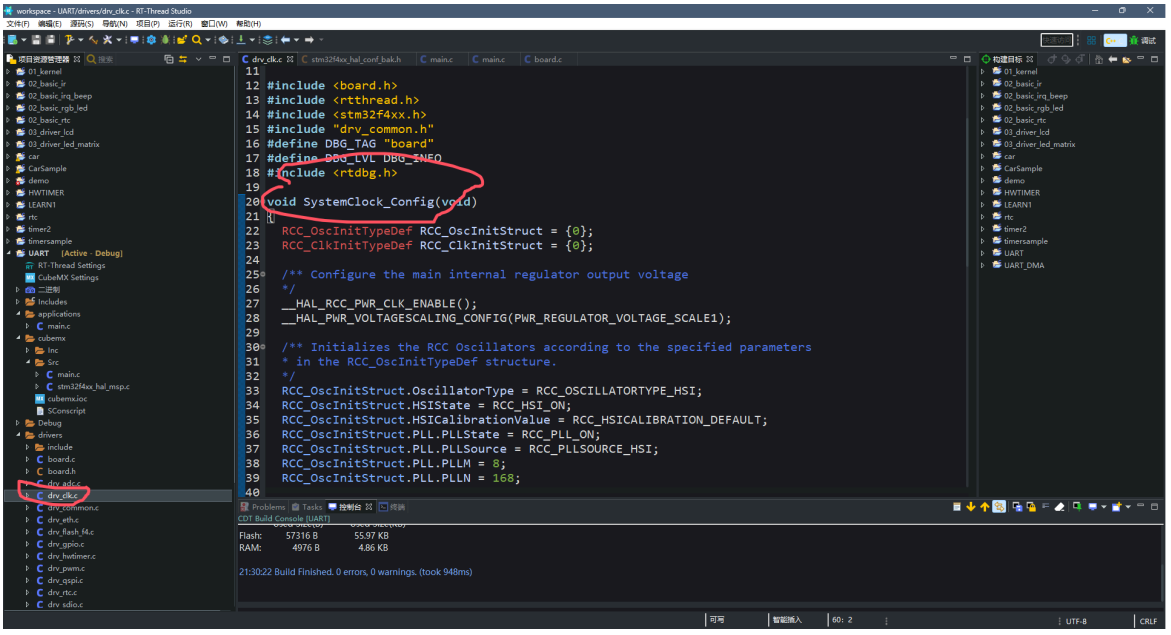

STM32CubeMX cubemx.ioc*: STM32F407ZGTx
File
Window
Help
Home
STM32F407ZGTx
cubemx.ioc - Clock Configuration
GENERATE CODE
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Resolve Clock Issues
RTC Clock Mux
HSE
HSE_RTC
LSE
LSI
To RTC (KHz)
To IWDG (KHz)
LSI RC
HSI RC
System Clock Mux
HSI
HSE
PLLCLK
SYSCLK (MHz)
AHB Prescaler
APB1 Prescaler
APB2 Prescaler
PCLK1
PCLK2
Enable CSS
PLL Source Mux
Main PLL
Ethernet PTP clock (MHz)
HCLK to AHB bus, core, memory and DMA (MHz)
To Cortex System timer (MHz)
FCLK Cortex clock (MHz)
APB1 peripheral clocks (MHz)
APB1 Timer clocks (MHz)
APB2 peripheral clocks (MHz)
APB2 timer clocks (MHz)
48MHz clocks (MHz)
I2S source Mux
PLLI2SCLK
I2S clocks (MHz)
MCO2 source Mux
SYSCLK


STM32CubeMX cubemx.ioc*: STM32F407ZGTx
File
Window
Help
Home
STM32F407ZGTx
cubemx.ioc - Project Manager
GENERATE CODE
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Project
Code Generator
Advanced Settings
Project Settings
Project Name
cubemx
Project Location
D:/RT-ThreadStudio/workspace/666
Browse
Application Structure
Basic
Do not generate the main()
Toolchain Folder Location
D:/RT-ThreadStudio/workspace/666\cubemx\
Toolchain / IDE
EWARM
Min Version
V8.50
Generate Under Root
Linker Settings
Minimum Heap Size
0x200
Minimum Stack Size
0x400
Thread-safe Settings
Cortex-M4NS
Enable multi-threaded support
Thread-safe Locking Strategy
Default – Mapping suitable strategy depending on RTOS selection.
Mcu and Firmware Package
Mcu Reference
STM32F407ZGTx
Firmware Package Name and Version
STM32Cube FW_F4 V1.28.2
Use latest available version
Use Default Firmware Location
Firmware Relative Path
C:/Users/zhr/STM32Cube/Repository/STM32Cube_FW_F4_V1.28.2
Browse

- 3. 配置board.h和board.c



将CubeMx中SystemClock_Config()函数剪切到drv_clk.c中替换同名函数



- 4.书写main函数代码

```c
#include <rtthread.h>
#include "rtdevice.h"

#define DBG_TAG "main"
#define DBG_LVL DBG_LOG
#include <rtdbg.h>
#include "drv_common.h"

static int uart_sample(int argc);
#define SAMPLE_UART_NAME "uart2" //创建串口名

/* 用于接收消息的信号量 */
static struct rt_semaphore rx_sem; //接受消息的信号量
static rt_device_t serial; //设备句柄

int main(void)
{
    int count = 1;
    uart_sample(2);

    while (count++)
    {
        rt_thread_mdelay(1000);
    }

    return RT_EOK;
}
```

```c

/*串口回调函数*/
static rt_err_t uart_input(rt_device_t dev, rt_size_t size)
{
    /*进入中断后,释放信号量*/
    rt_sem_release(&rx_sem);
    return RT_EOK;
}

static void serial_thread_entry(void *parameter)
{
    char ch;

    while (1)
    {
       /*从串口中接收一个字节,没有读取到则等待接收信号量*/
        while (rt_device_read(serial, -1, &ch, 1) != 1)
        {
           /*阻塞等待接收信号量,等到信号量后再次读取信号量*/
            rt_sem_take(&rx_sem, RT_WAITING_FOREVER);
        }

        /* 读取到的数据通过串口输出 */
        ch = ch;
        rt_device_write(serial, 0, &ch, 1);
    }
}

static int uart_sample(int argc)
{
    rt_err_t ret = RT_EOK;
    struct serial_configure config = RT_SERIAL_CONFIG_DEFAULT;//修改设置使用的结构体
    char uart_name[RT_NAME_MAX];
    char str[] = "hello RT-Thread!\r\n";

    if (argc == 2)
    {
        rt_strncpy(uart_name, SAMPLE_UART_NAME, RT_NAME_MAX);
    }
    else
    {
        rt_strncpy(uart_name, SAMPLE_UART_NAME, RT_NAME_MAX);
    }

    /* 查找系统中的串口设备 */
    serial = rt_device_find(uart_name);
    /*修改串口设置*/
    config.baud_rate = BAUD_RATE_9600;        // 修改波特率为 9600
    config.data_bits = DATA_BITS_8;           // 数据位 8
    config.stop_bits = STOP_BITS_1;           // 停止位 1
    config.bufsz     = 128;                // 修改缓冲区 rx buff size 为128
    config.parity    = PARITY_NONE;           // 无奇偶校验位
    rt_device_control(serial,RT_DEVICE_CTRL_CONFIG,&config);

    if (!serial)
```

```
    {
        rt_kprintf("find %s failed!\n", uart_name);
        return RT_ERROR;
    }

    /* 初始化信号量 */
    rt_sem_init(&rx_sem, "rx_sem", 0, RT_IPC_FLAG_FIFO);

    /* 以中断接收及轮询发送模式打开串口设备 */
    //rt_device_open(serial, RT_DEVICE_FLAG_INT_RX |
RT_DEVICE_OFLAG_RDWR);
    /* 以中断接受及中断发送模式打开串口设备 */
    rt_device_open(serial, RT_DEVICE_FLAG_INT_TX | RT_DEVICE_FLAG_INT_RX
| RT_DEVICE_OFLAG_RDWR);

    /* 设置接收回调函数 */
    rt_device_set_rx_indicate(serial, uart_input);

    /* 发送字符串 */
    rt_device_write(serial, 0, str, (sizeof(str) - 1));

    /* 创建 serial 线程 */
    rt_thread_t thread = rt_thread_create("serial", serial_thread_entry,
RT_NULL, 2048, 25, 10);

    /* 创建成功则启动线程 */
    if (thread != RT_NULL)
    {
        rt_thread_startup(thread);
        rt_kprintf("thread startes successfully!\n");
    }
    else
    {
        ret = RT_ERROR;
    }

    return ret;
}
```

打开串口助手:

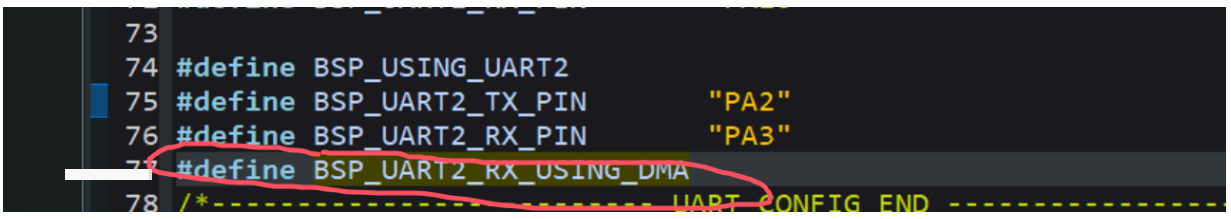
显示如图则成功:

2.UART设备关于DMA的使用:

操作流程:

- 

ctrl+s保存后进入board.h定义串口及DMA宏定义



- 配置CubeMX,与上面的配置完全相同

- 将main.c换为下面的代码

```c
 #include <rtthread.h>
#include <board.h>
#include <rtdevice.h>

#define DBG_TAG "main"
#define DBG_LVL DBG_LOG
#include <rtdbg.h>

#define SAMPLE_UART_NAME       "uart2"      /* 串口设备名称 */

/* 串口接收消息结构*/
struct rx_msg
{
    rt_device_t dev;
    rt_size_t size;
};
/* 串口设备句柄 */
static rt_device_t serial;

/* 初始化配置参数 */
struct serial_configure config = RT_SERIAL_CONFIG_DEFAULT;

/* 消息队列控制块 */
static struct rt_messagequeue rx_mq;

/* 接收数据回调函数 */
static rt_err_t uart_input(rt_device_t dev, rt_size_t size)
{
    struct rx_msg msg;
    rt_err_t result;
    msg.dev = dev;
    msg.size = size;

    result = rt_mq_send(&rx_mq, &msg, sizeof(msg));
    if ( result == -RT_EFULL)
    {
        /* 消息队列满 */
        rt_kprintf("message queue full! \n");
    }
    return result;
}

static void serial_thread_entry(void *parameter)
{
    struct rx_msg msg;
    rt_err_t result;
    rt_uint32_t rx_length;
    static char rx_buffer[RT_SERIAL_RB_BUFSZ + 1];

    while (1)
    {
        rt_memset(&msg, 0, sizeof(msg));
        /* 从消息队列中读取消息*/
        rt_thread_mdelay(5);
        result = rt_mq_recv(&rx_mq, &msg, sizeof(msg), RT_WAITING_FOREVER);
```

```
        //rt_thread_mdelay(50);
        if (result == RT_EOK)
        {
            /* 从串口读取数据*/
            rx_length = rt_device_read(msg.dev, 0, rx_buffer, msg.size);
            rx_buffer[rx_length] = '\0';
            /* 通过串口设备 serial 输出读取到的消息 */
            rt_device_write(serial, 0, rx_buffer, rx_length);
            /* 打印数据 */
            //rt_kprintf("%s\n",rx_buffer);
        }
    }
}

static int uart_dma_sample()
{
    rt_err_t ret = RT_EOK;
    char uart_name[RT_NAME_MAX];
    static char msg_pool[256];
    char str[] = "hello RT-Thread!\r\n";

    /* 查找串口设备 */
    serial = rt_device_find(SAMPLE_UART_NAME);
    if (!serial)
    {
        rt_kprintf("find %s failed!\n", uart_name);
        return RT_ERROR;
    }

    /* 初始化消息队列 */
    rt_mq_init(&rx_mq, "rx_mq",
               msg_pool,                 /* 存放消息的缓冲区 */
               sizeof(struct rx_msg),    /* 一条消息的最大长度 */
               sizeof(msg_pool),         /* 存放消息的缓冲区大小 */
               RT_IPC_FLAG_FIFO);        /* 如果有多个线程等待，按照先来先得到的方法分配消息 */

    /* 修改串口配置参数 */
    config.baud_rate = BAUD_RATE_115200;        //修改波特率为 9600
    config.data_bits = DATA_BITS_8;           //数据位 8
    config.stop_bits = STOP_BITS_1;           //停止位 1
    config.bufsz     = 128;                   //修改缓冲区 buff size 为 128
    config.parity    = PARITY_NONE;           //无奇偶校验位

    /*控制串口设备。通过控制接口传入命令控制字，与控制参数 */
    rt_device_control(serial, RT_DEVICE_CTRL_CONFIG, &config);

    /* 以 DMA 接收及轮询发送方式打开串口设备 */
    rt_device_open(serial, RT_DEVICE_FLAG_DMA_RX);
    /* 设置接收回调函数 */
    rt_device_set_rx_indicate(serial, uart_input);
    /* 发送字符串 */
    rt_device_write(serial, 0, str, (sizeof(str) - 1));

    /* 创建 serial 线程 */
    rt_thread_t thread = rt_thread_create("serial", serial_thread_entry, RT_NULL, 1024, 25, 10);
```

```
    /* 创建成功则启动线程 */
    if (thread != RT_NULL)
    {
        rt_thread_startup(thread);
    }
    else
    {
        ret = RT_ERROR;
    }
    return ret;
}

int main(void)
{
    uart_dma_sample();
   return RT_EOK;
}
```

发送数据,串口助手显示接收数据则成功实现功能