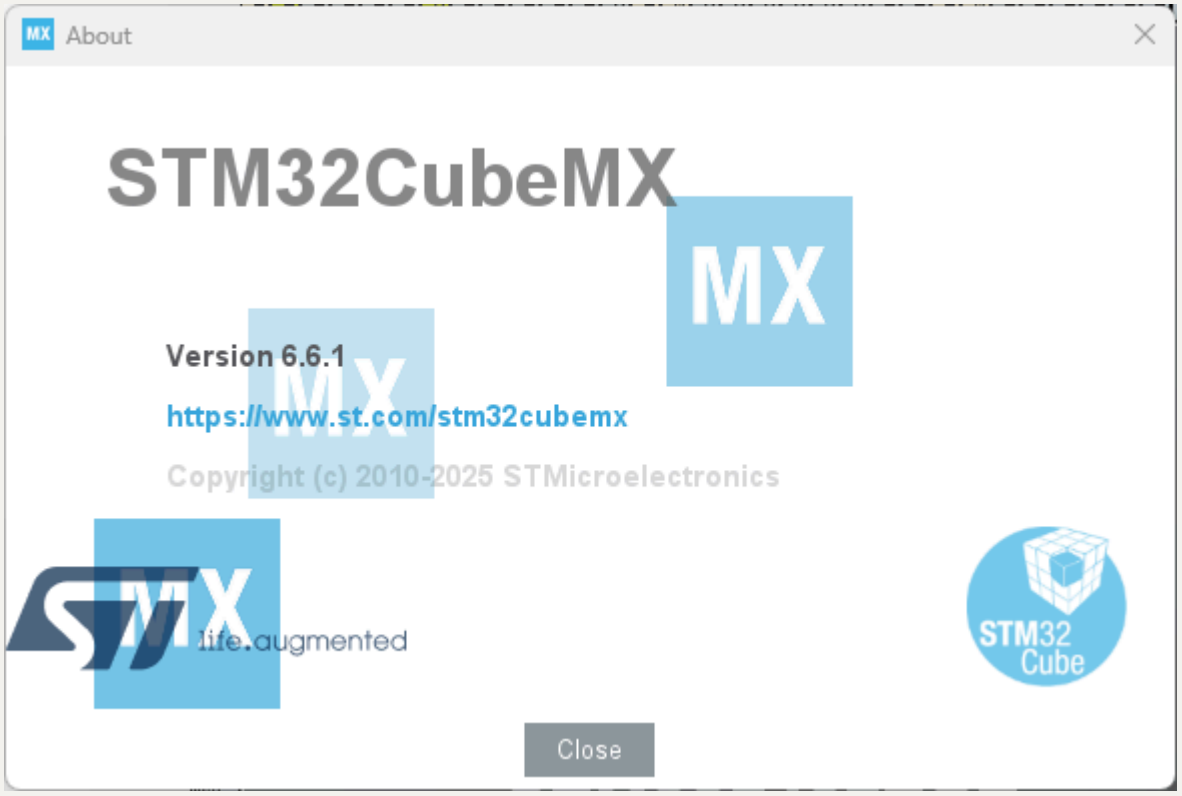
CubeMX导入RT-Thread-NANO

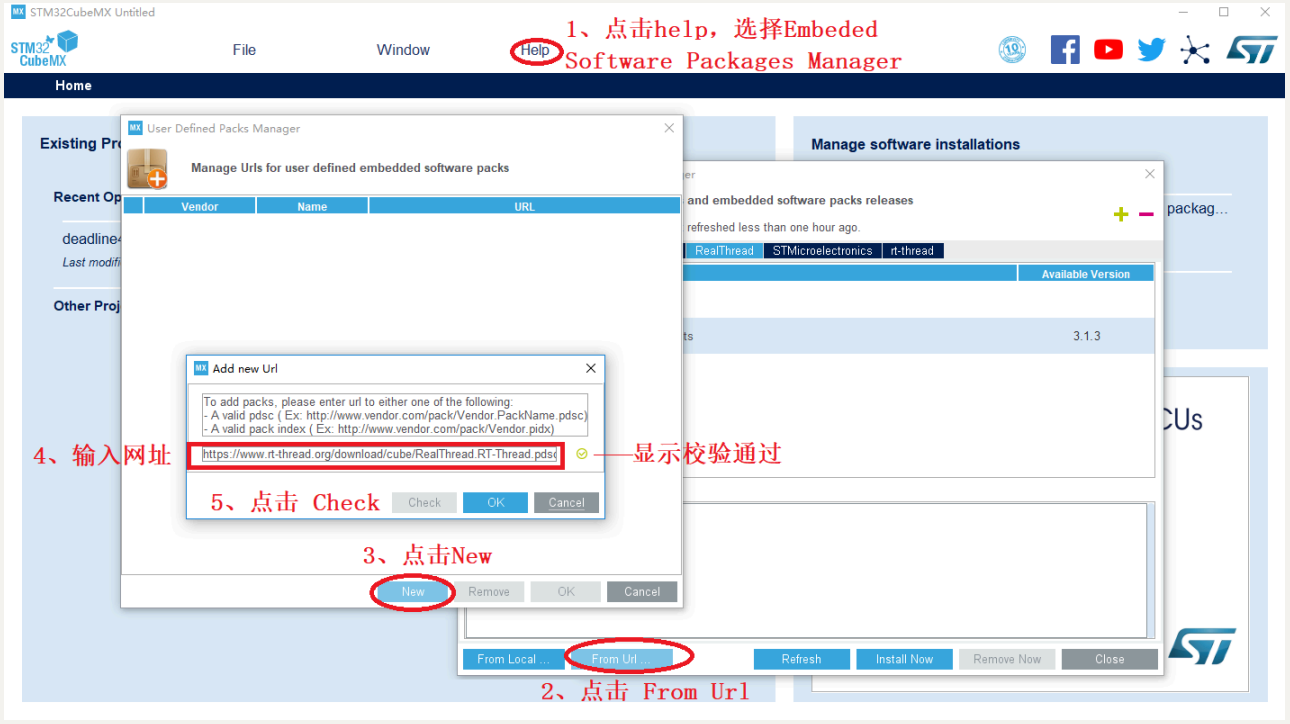
我的CubeMx版本号



下载RT-Thread-NANO

下载链接：RT-Thread下载链接

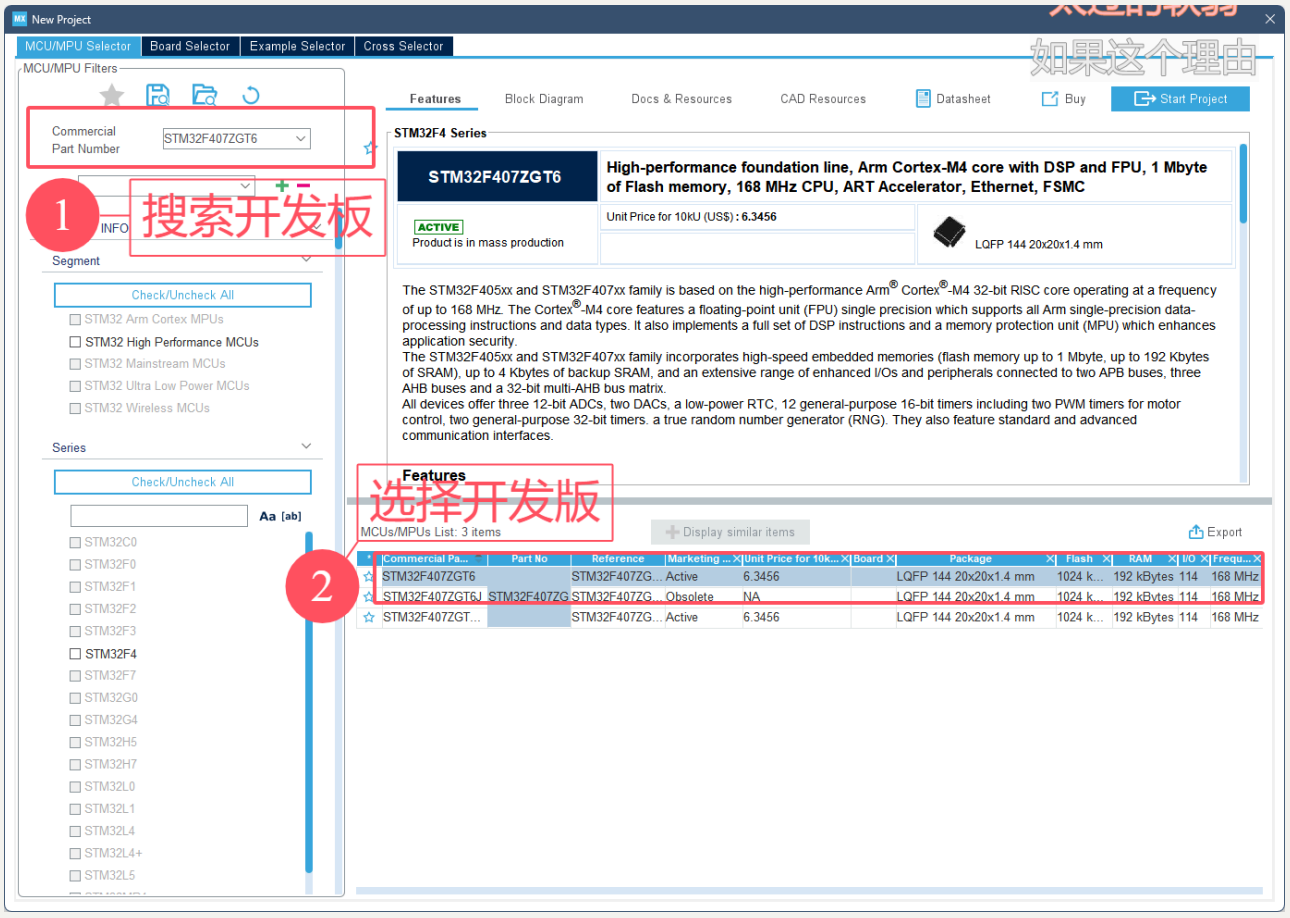
注: 这是Nano版本 3.1.5



创建工程

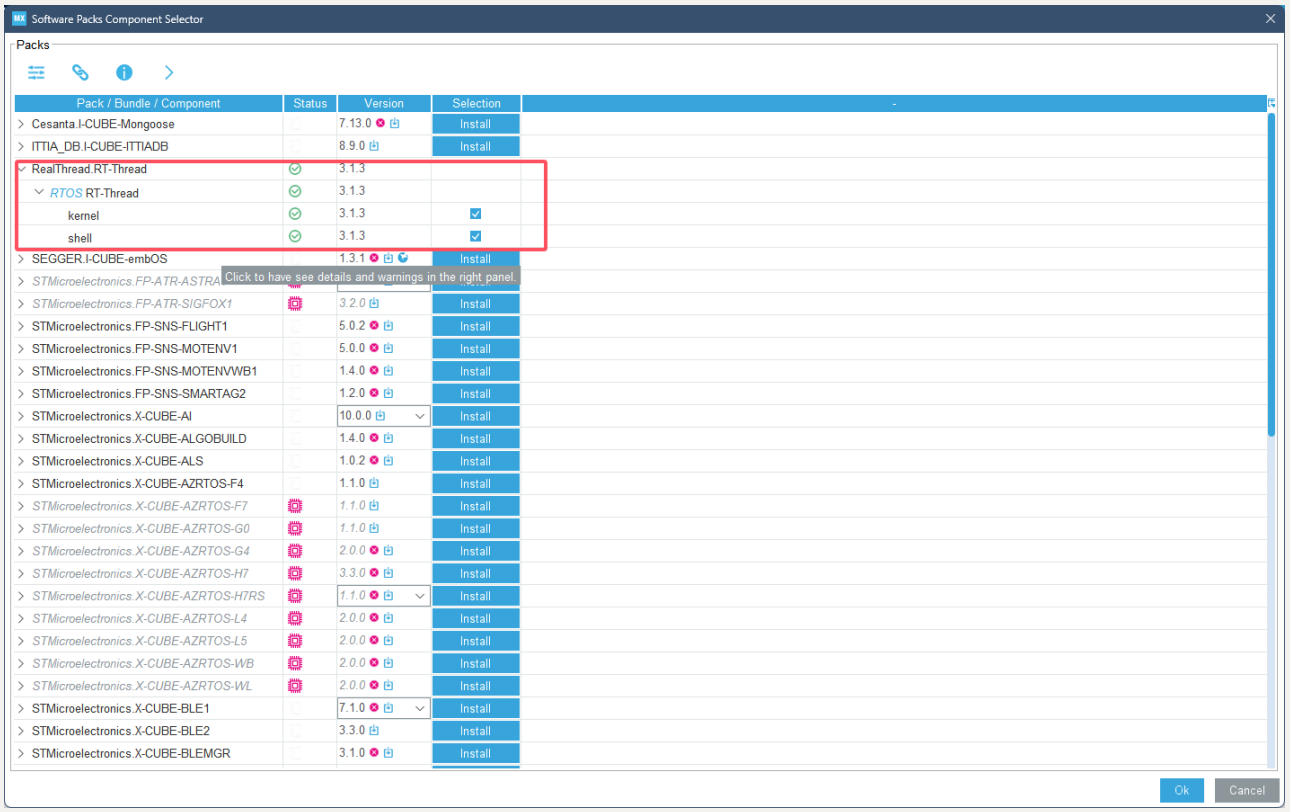
> *这里以 **STM32F407ZGT6** 开发板为例进行操作*

1. 选择开发板


New Project
MCU/MPU Selector
Board Selector
Example Selector
Cross Selector
MCU/MPU Filters
Commercial Part Number
STM32F407ZGT6
1
搜索开发板
INFO
Segment
Check/Uncheck All
STM32 Arm Cortex MPUs
STM32 High Performance MCUs
STM32 Mainstream MCUs
STM32 Ultra Low Power MCUs
STM32 Wireless MCUs
Series
Check/Uncheck All
STM32C0
STM32F0
STM32F1
STM32F2
STM32F3
STM32F4
STM32F7
STM32G0
STM32G4
STM32H5
STM32H7
STM32L0
STM32L1
STM32L4
STM32L4+
STM32L5
Features
Block Diagram
Docs & Resources
CAD Resources
Datasheet
Buy
Start Project
STM32F4 Series
STM32F407ZGT6
High-performance foundation line, Arm Cortex-M4 core with DSP and FPU, 1 Mbyte of Flash memory, 168 MHz CPU, ART Accelerator, Ethernet, FSMC
ACTIVE
Product is in mass production
Unit Price for 10kU (US$) : 6.3456
LQFP 144 20x20x1.4 mm
The STM32F405xx and STM32F407xx family is based on the high-performance Arm® Cortex®-M4 32-bit RISC core operating at a frequency of up to 168 MHz. The Cortex®-M4 core features a floating-point unit (FPU) single precision which supports all Arm single-precision data-processing instructions and data types. It also implements a full set of DSP instructions and a memory protection unit (MPU) which enhances application security.
The STM32F405xx and STM32F407xx family incorporates high-speed embedded memories (flash memory up to 1 Mbyte, up to 192 Kbytes of SRAM), up to 4 Kbytes of backup SRAM, and an extensive range of enhanced I/Os and peripherals connected to two APB buses, three AHB buses and a 32-bit multi-AHB bus matrix.
All devices offer three 12-bit ADCs, two DACs, a low-power RTC, 12 general-purpose 16-bit timers including two PWM timers for motor control, two general-purpose 32-bit timers. a true random number generator (RNG). They also feature standard and advanced communication interfaces.
Features
选择开发版
2
MCUs/MPUs List: 3 items
Display similar items
Export
Commercial Pa...
Part No
Reference
Marketing ...
Unit Price for 10k...
Board
Package
Flash
RAM
I/O
Frequ...
STM32F407ZGT6
STM32F407ZG...
Active
6.3456
LQFP 144 20x20x1.4 mm
1024 k...
192 kBytes
114
168 MHz
STM32F407ZGT6J
STM32F407ZG...
Obsolete
NA
LQFP 144 20x20x1.4 mm
1024 k...
192 kBytes
114
168 MHz
STM32F407ZGT...
STM32F407ZG...
Active
6.3456
LQFP 144 20x20x1.4 mm
1024 k...
192 kBytes
114
168 MHz

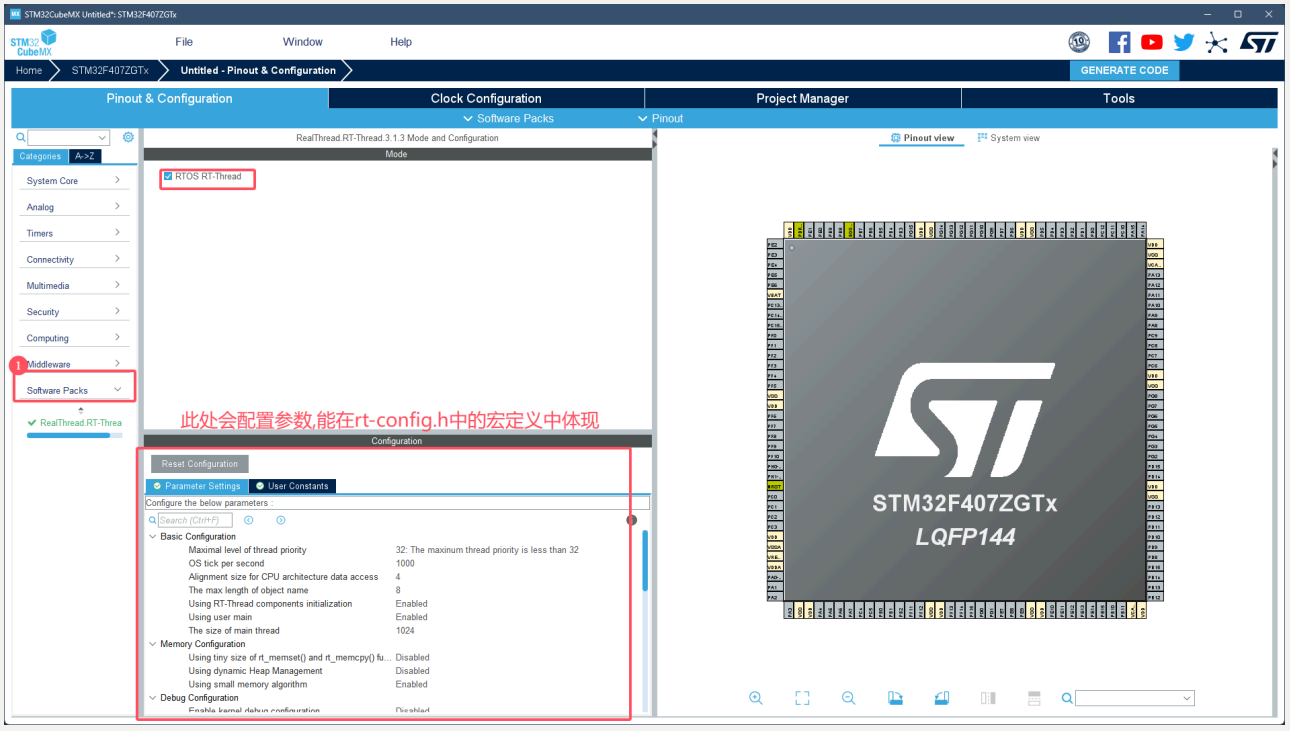
2. 选择操作系统

选择导入的RT-Thread软件包（首次安装需要手动添加，再勾选它），然后点击右下角“OK”。

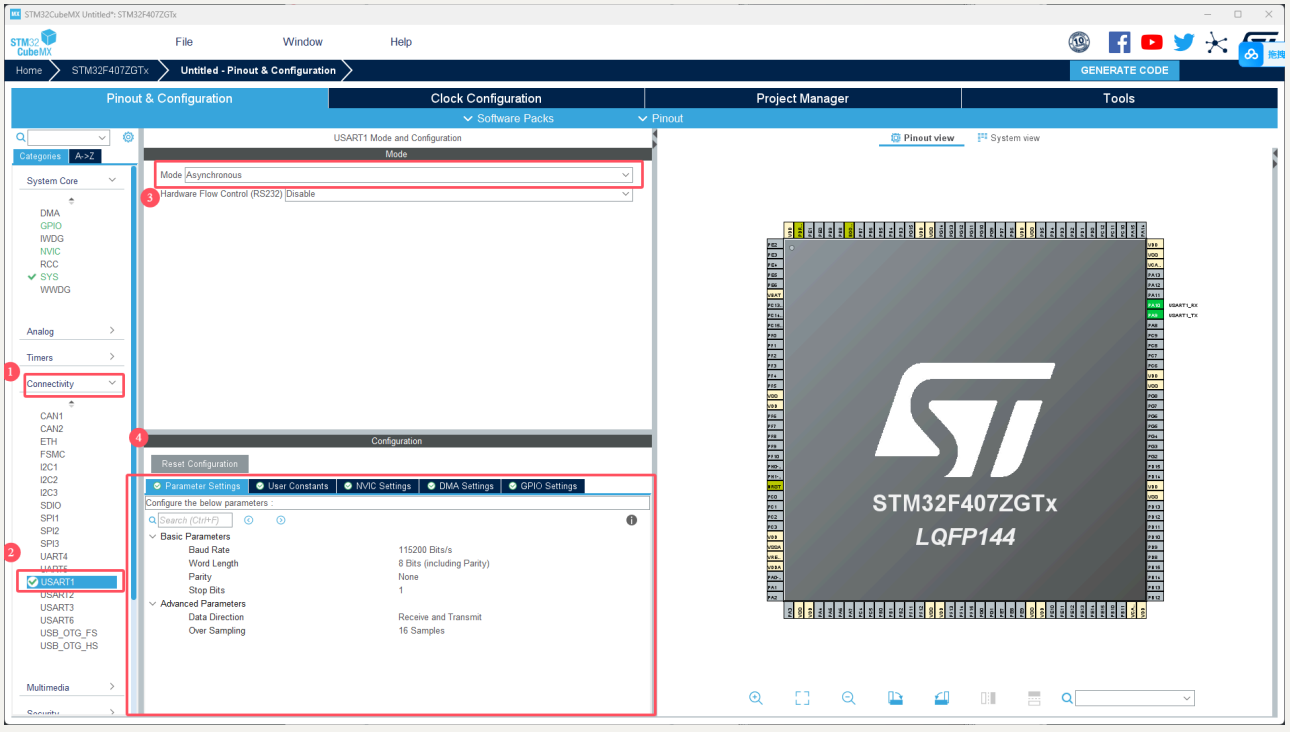


3. 配置组件

在工程界面的 `Pinout & Configuration` 中，进入所选组件的参数配置区域，按下图配置：



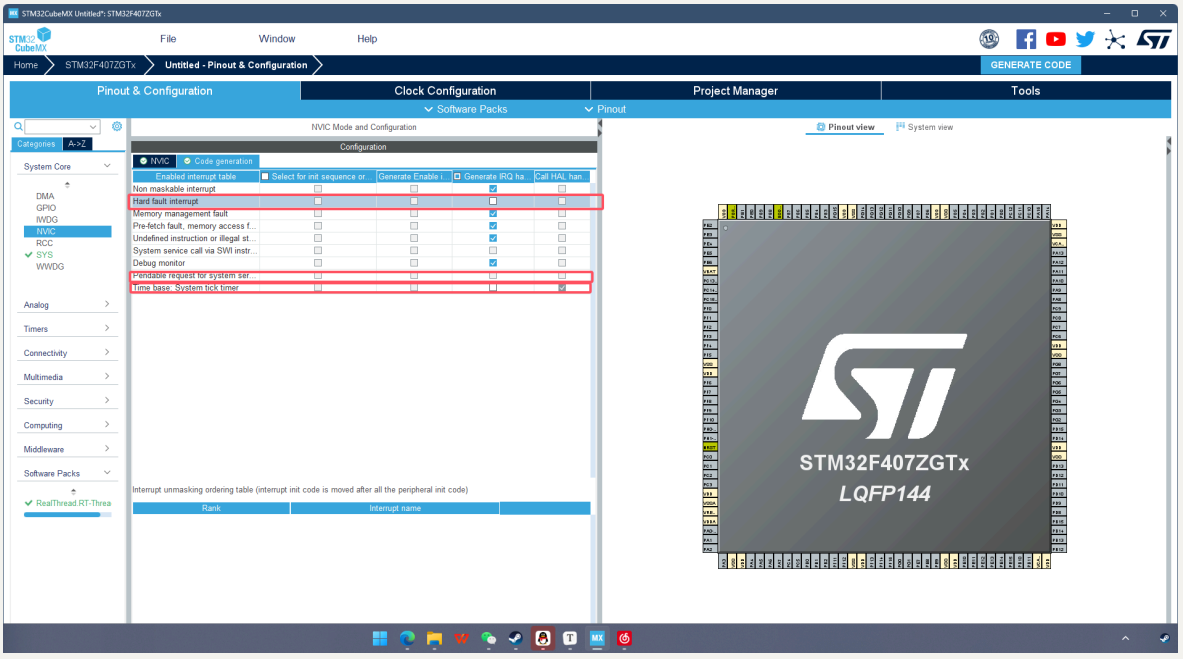
4. 配置串口



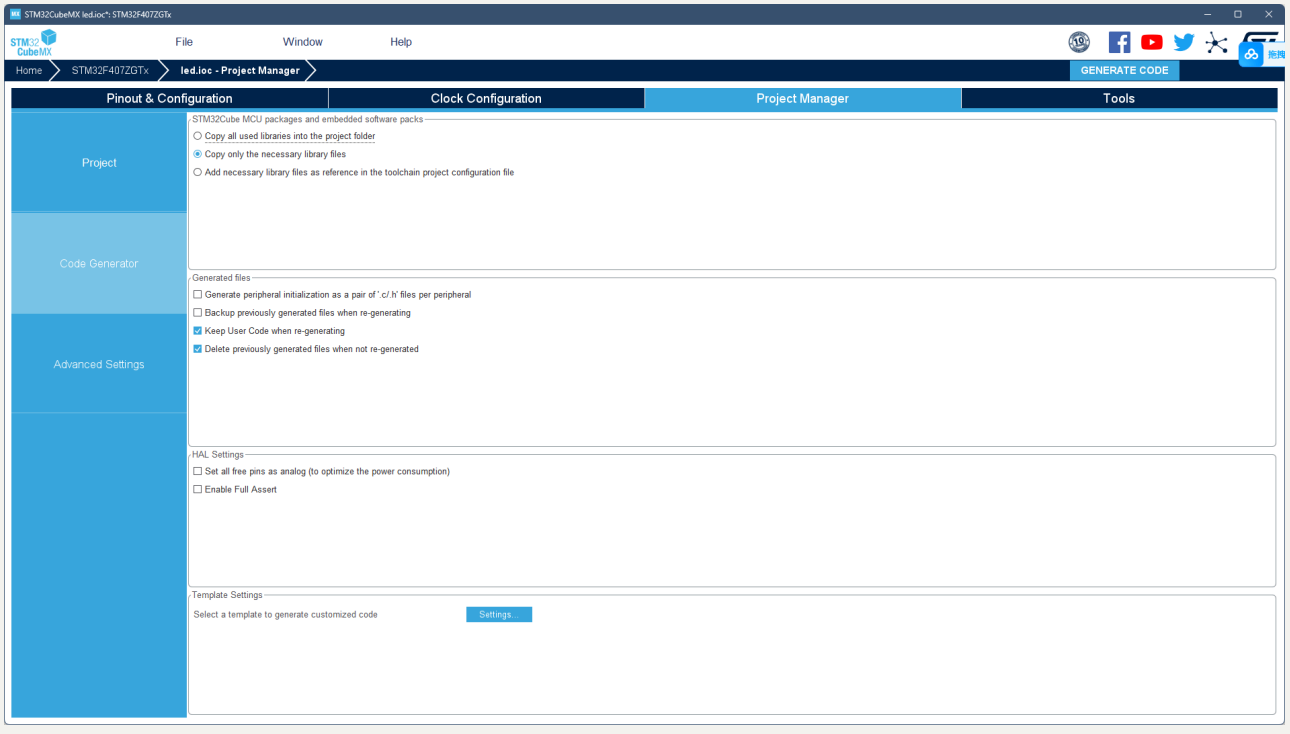
5. 中断和异常处理

> *RT-Thread 操作系统重定义了 `HardFault_Handler`、`PendSV_Handler`、`SysTick_Handler` 等中断函数。为避免重复定义问题，在生成工程之前，需在中断配置中取消这三个中断的使能（注释选项分别为 `Hard fault interrupt`、`Pendable request`、`Time base : System tick timer`）。然后点击“生成代码”。*

> *操作示例：取消这三个中断在CubeMX中的使能。*



6. 生成文件并编译



编译错误信息：

```
linking...
led\led.axf: Error: L6218E: Undefined symbol rt_hw_console_getchar (referred from shell.o).
Not enough information to list image symbols.
Not enough information to list load addresses in the image map.
Finished: 2 information, 0 warning and 1 error messages.
"led\led.axf" - 1 Error(s), 0 Warning(s).
Target not created.
```

错误原因是未定义相关函数，但RT-Thread中已有该函数的接口，因此需要手动添加。

```
extern UART_HandleTypeDef huart1;
char rt_hw_console_getchar(void)
{
    int ch = -1;
    if (__HAL_UART_GET_FLAG(&huart1, UART_FLAG_RXNE) != RESET)
    {
        ch = huart1.Instance->DR & 0xff;
    }
    else
    {
        rt_thread_mdelay(10);
    }
    return ch;
}
```

编译通过后，LED点亮程序与串口的基本配置完成。

7. 书写LED点亮程序与串口发送检验

1. 新建文件夹 **BSP** 并将其加入编译路径
2. 在 **BSP** 文件夹中创建 `Finsh.c` 和 `Finsh.h`，并加入编译路径
3. 使能 `rt-kprintf` 功能

```
void rt_hw_console_output(const char *str)
{
    rt_size_t i = 0, size = 0;
    char a = '
';


    __HAL_UNLOCK(&huart1);
```

```
    size = rt_strlen(str);
    for (i = 0; i < size; i++)
    {
        if (*(str + i) == '
')
        {
            HAL_UART_Transmit(&huart1, (uint8_t *)&a, 1, 1);
        }
        HAL_UART_Transmit(&huart1, (uint8_t *)(str + i), 1, 1);
    }
}
```

综合代码

```
static rt_thread_t led_thread;

void test_thread_entry(void *parameter)
{
    while(1)
    {
        HAL_GPIO_WritePin(GPIOF, GPIO_PIN_11, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOF, GPIO_PIN_12, GPIO_PIN_RESET);
        rt_kprintf("hello rtthread
");
        rt_thread_mdelay(500);
        HAL_GPIO_WritePin(GPIOF, GPIO_PIN_11, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOF, GPIO_PIN_12, GPIO_PIN_SET);
        rt_thread_mdelay(500);
    }
}

void TaskInit(void)
{
    led_thread = rt_thread_create("ledThread",        /* 线程名字 */
                                  test_thread_entry,  /* 线程入口函数
*/
                                  RT_NULL,            /* 线程入口参数
*/
                                  256,                /* 线程栈大小
*/
```

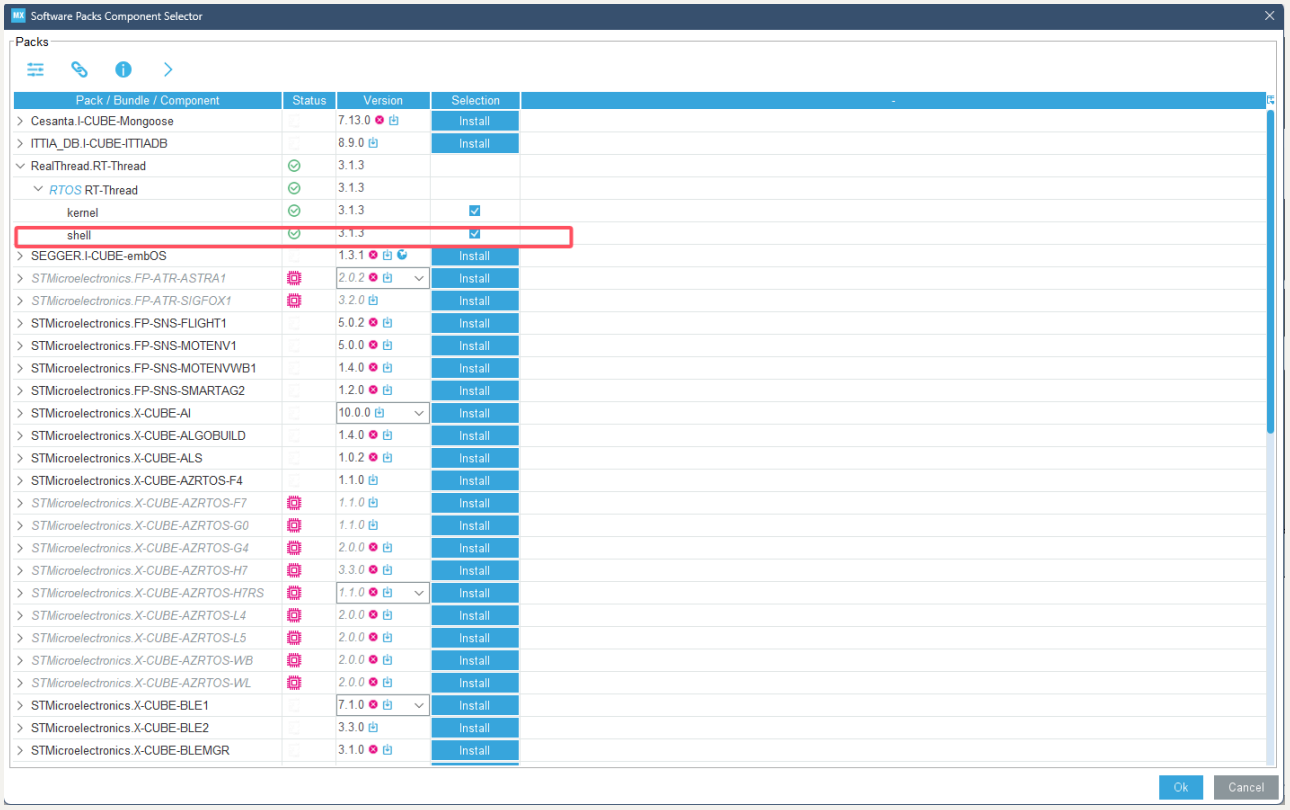
```
                                    2,                  /* 线程优先级
*/
                                    10                  /* 线程时间片
*/
                                   );
    if(led_thread != RT_NULL)  // 分配成功，加入到就绪队列中
    {
        rt_thread_startup(led_thread);
    }
}

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_USART1_UART_Init();
    TaskInit();
    while (1)
    {
    }
}
```

LED点亮成功,串口发送正常,则基本配置完成。

加入FinSH功能

1.确认FinSH被加入



2.将串口与FinSH对接(放在main函数前)

```
static int uart_init(void)
{
    MX_USART1_UART_Init();
    return 0;
}
INIT_BOARD_EXPORT(uart_init);
```

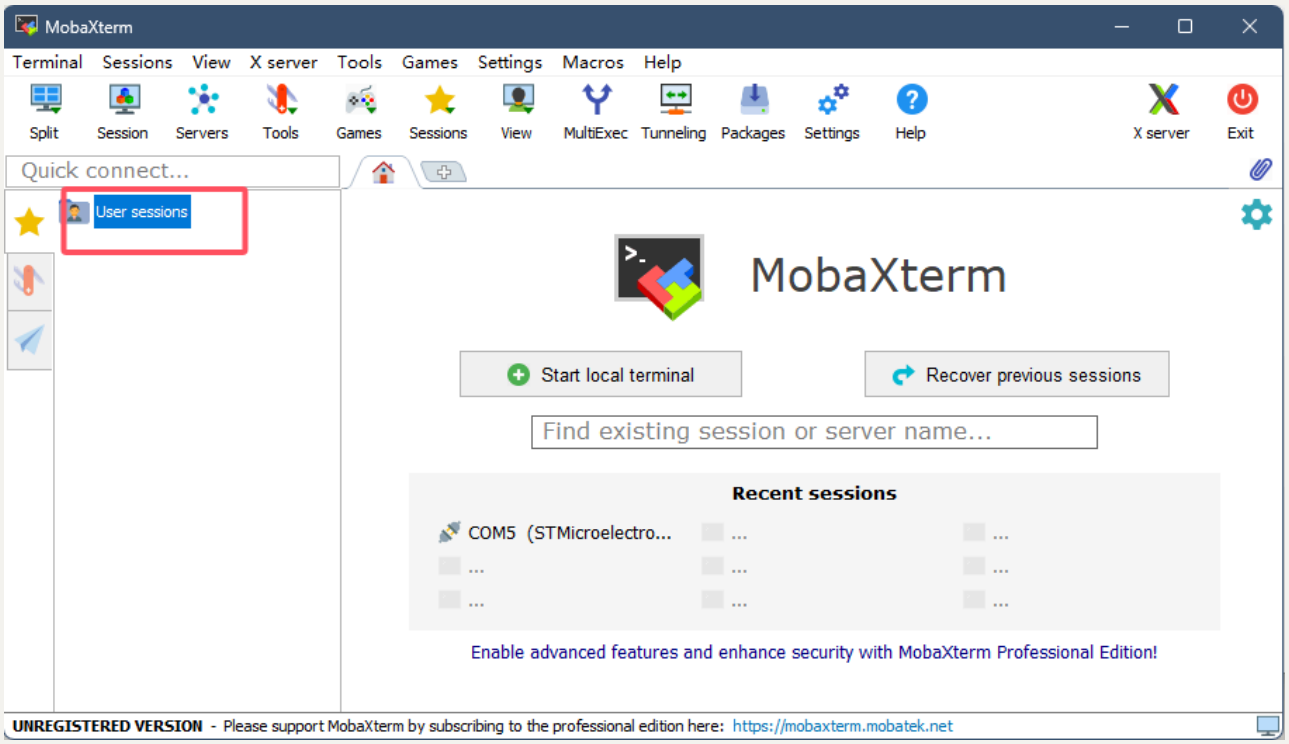
1. 在 main 线程的 while 中加入 `rt_thread_mdelay(100)`

2. 编译下载，打开 MobaXterm

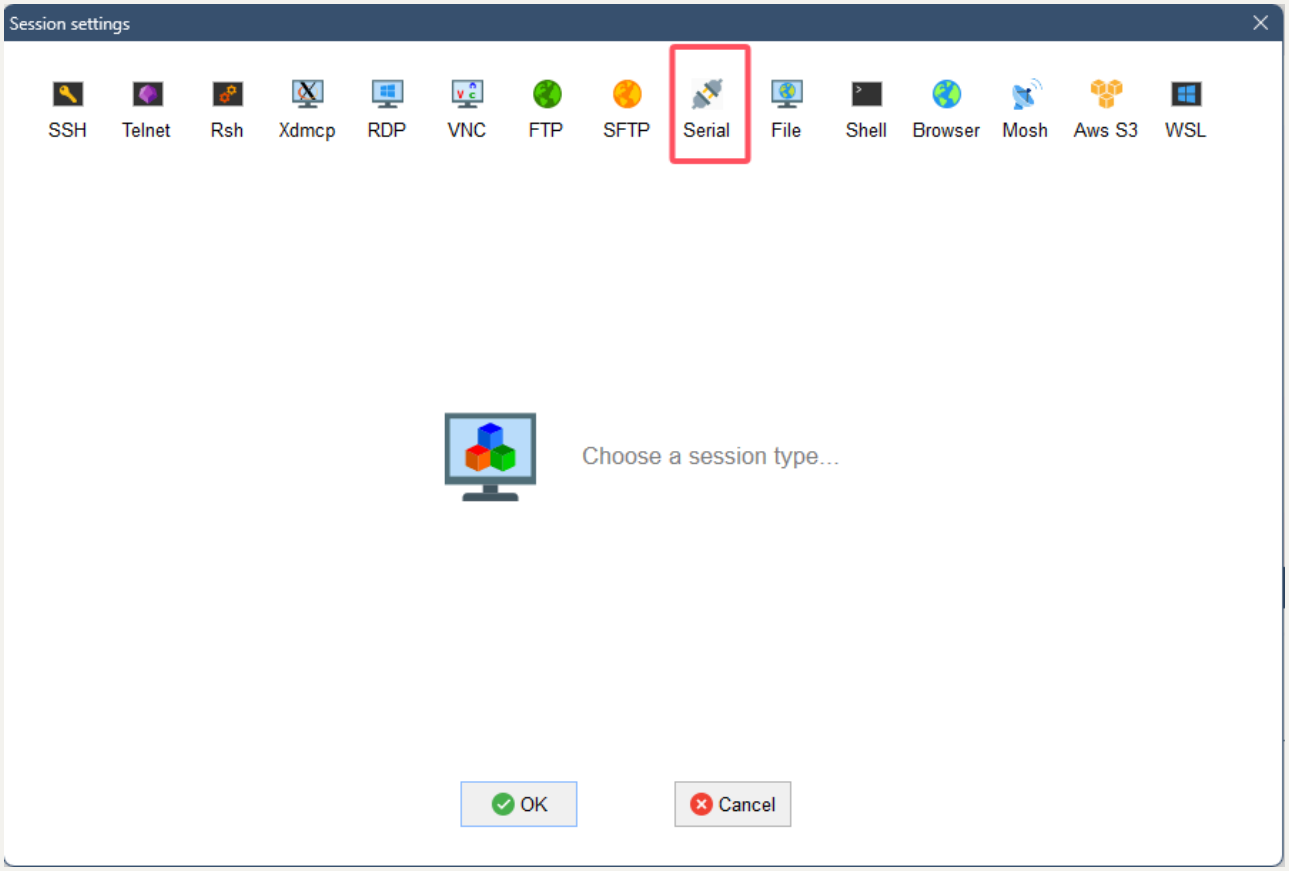
下载链接:

MobaXterm Xserver with SSH, telnet, RDP, VNC and X11 - Home Edition

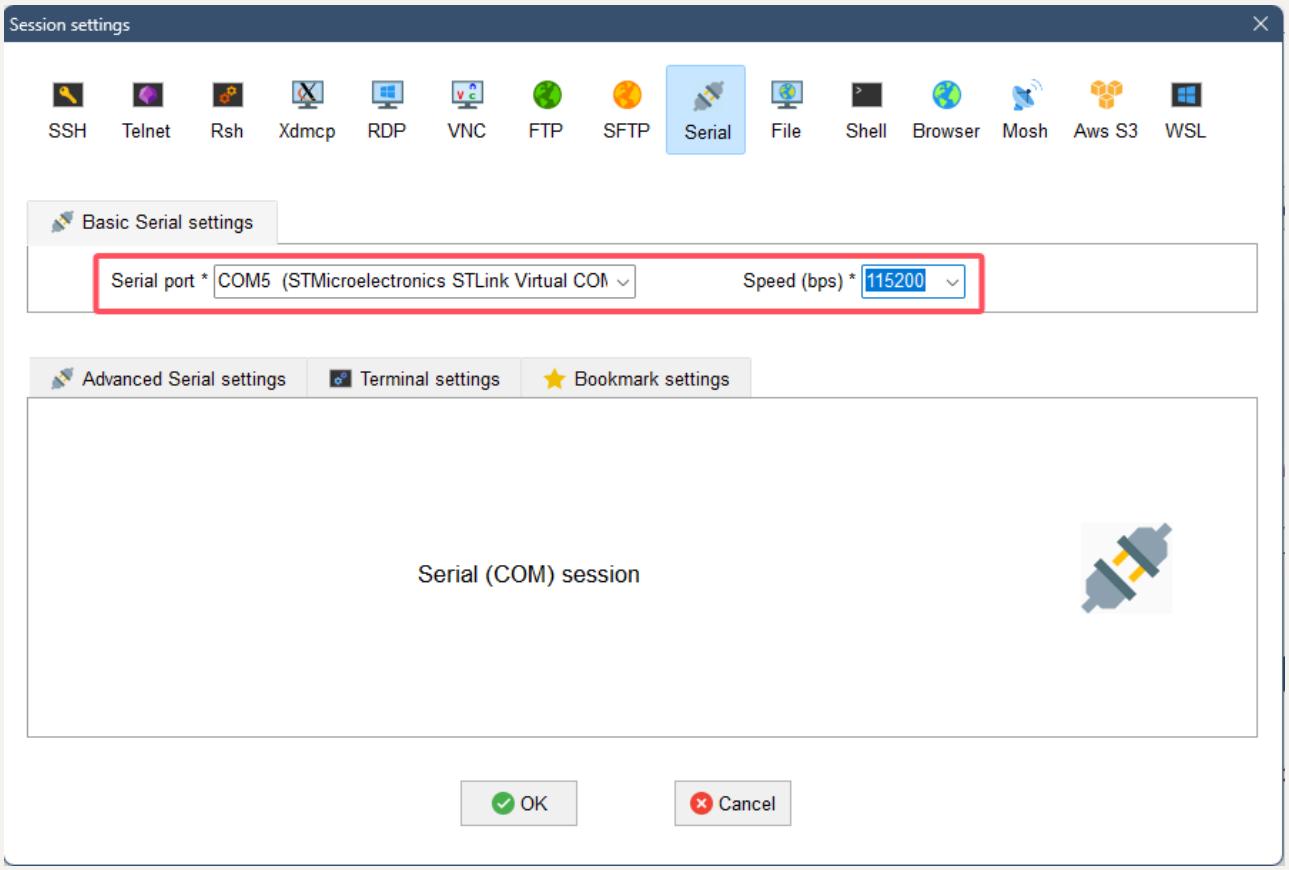
3. 选择 User sessions



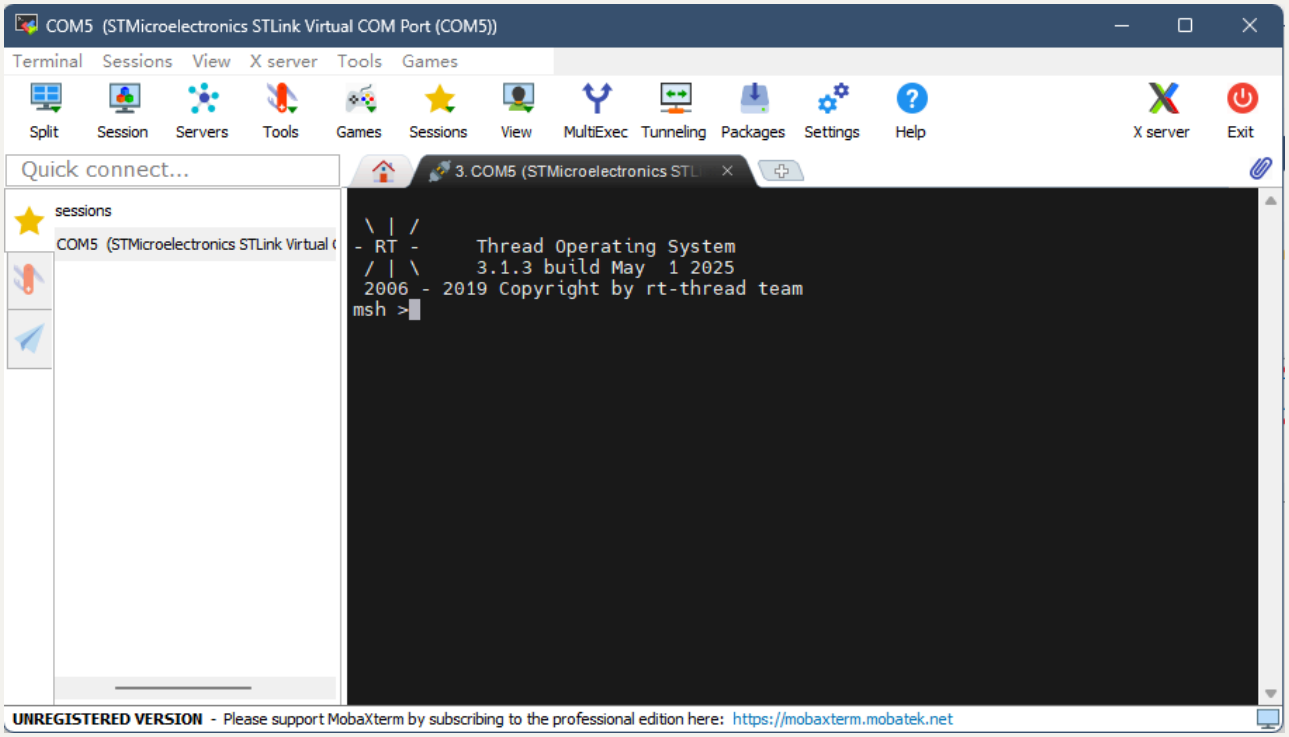
4. 选择 Serial(串口)



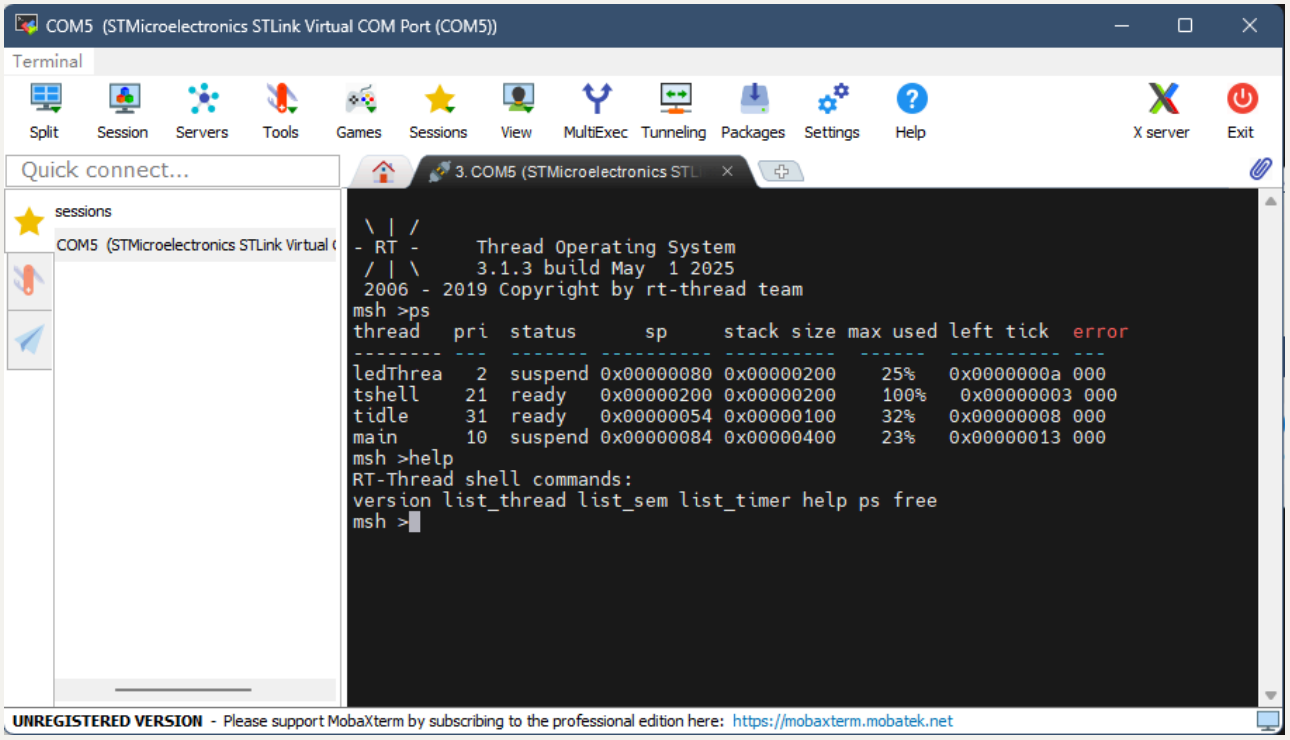
5. 选择对应波特率和端口，并点击 OK



6. 进入后点击 复位，出现如下图则表示配置正确



7. 输入 `ps`，会出现下图情形，显示当前线程

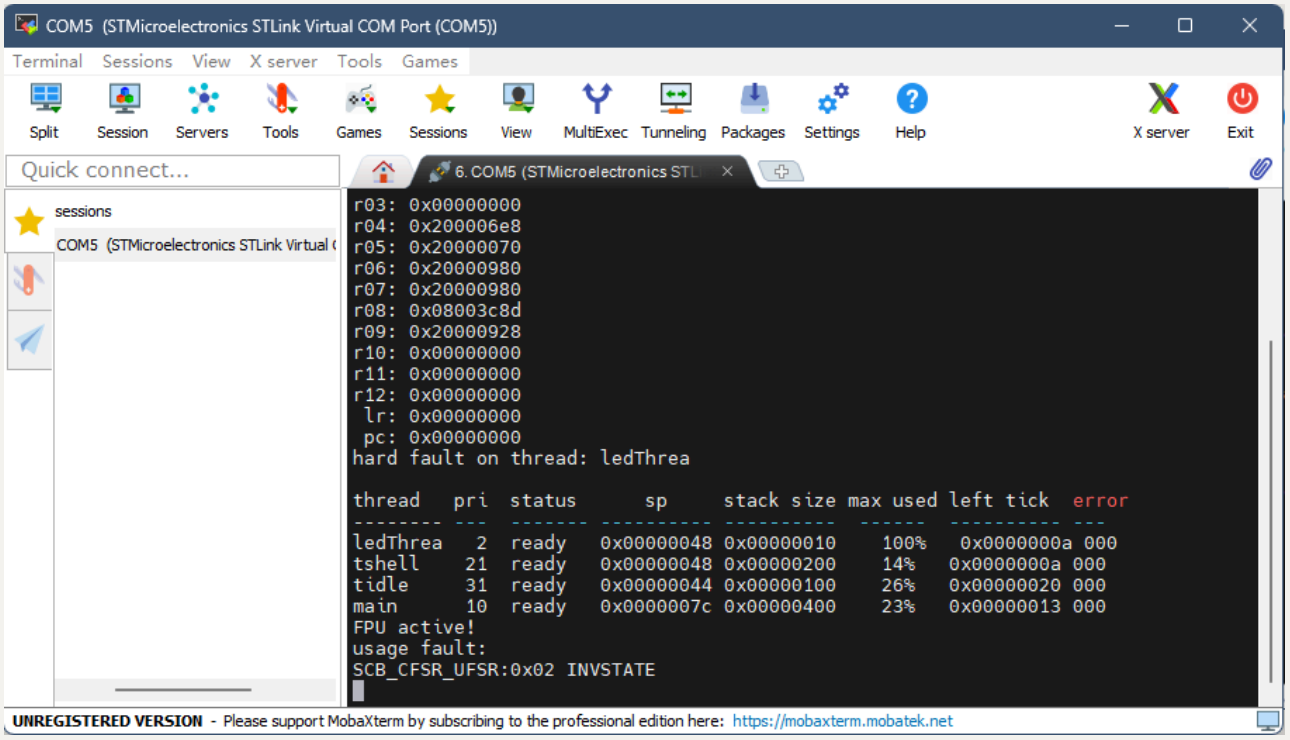


说明完全移植成功。

常见问题：

---

1. 输入命令后出现如下图情况



解决办法: ledthread 的线程栈空间太小, 可以将其调大

```
void TaskInit(void)
{
    // 返回值还是需要的，否则无法通过rt_thread_startup启动
    led_thread = rt_thread_create("ledThread",        /* 线程名字 */
                                  test_thread_entry,  /* 线程入口函数
*/
                                  RT_NULL,            /* 线程入口函数参
数 */
                                  16,                 /* 线程栈大小 */
                                  2,                  /* 线程的优先级
*/
                                  10                  /* 线程时间片 */
                                 );
    if(led_thread != RT_NULL)  // 分配成功，加入到就绪队列中
    {
        rt_thread_startup(led_thread);
    }
}
```

改为

```
void TaskInit(void)
{
//返回值还是需要的，否则无法通过rt_thread_startup启动
    led_thread = rt_thread_create("ledThread",        /* 线程名字 */
                                  test_thread_entry,  /* 线程入口函数
*/
                                  RT_NULL,            /* 线程入口函数参数
*/
                                  256,                /* 线程栈大小 */
                                  2,                  /* 线程的优先级 */
                                  10                  /* 线程时间片 */
                                  );
    if(led_thread != RT_NULL)//分配成功，加入到就绪队列中。
    {
        rt_thread_startup(led_thread);
    }
}
```