

阿熊的FreeRTOS教程系列！

哈喽大家好！我是你们的老朋友阿熊！STM32教程系列更新完结已经有一段时间了，视频反馈还是不错的，从今天开始我们将会更新我们的FreeRTOS的教程

由于东西真的太多了，也纠结了很久要不要讲这个系列，毕竟难度真的很大，怕在难以做到那么通俗易懂，经过一段时间的考虑，还是决定好了给大家做一个入门级的讲解使用，由于FreeRTOS的内容真的很多，作为还是学生的我使用的也相对较少，操作系统层面的东西，我会用最大的能力去让大家理解，主要讲述主要功能，学完以后保证大伙可以理解80%以上的FreeRTOS的使用场景，好了废话不多说，开始我们的课程吧！



第二章：FreeRTOS的移植以及做基本的使用

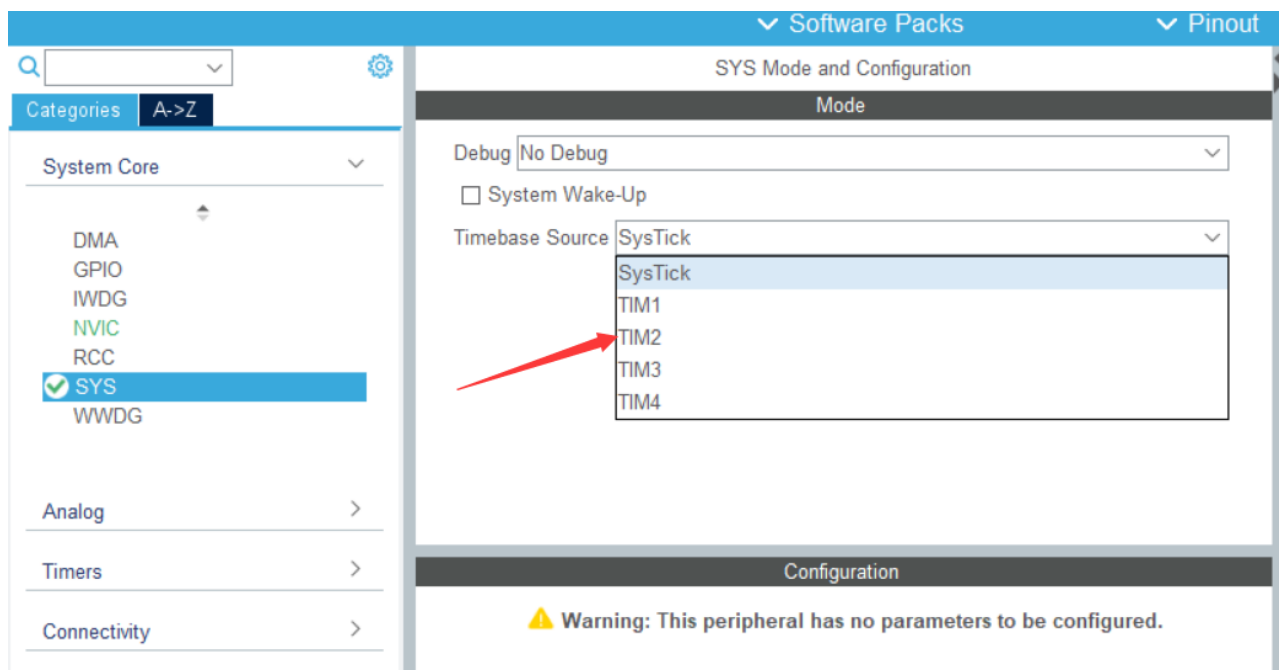
本节课我们将会带着大家进行我们的FreeRTOS的移植，本节课主要是实际操作，而且刚开始学习的话，暂时不用过于纠结原理，FreeRTOS只是一个使用的工具我们应该学习的是如何去使用，就像学习我们的STM32一样，先把项目建立出来，让我们LED亮起来再去研究原理和代码

壹：FreeRTOS的手动移植

这里我们将会教大家 如何进行我们手动的项目移植

①.建立STM32的空项目

这里使用STM32CubeMX快速创建项目，要注意的是我们完成最基本的配置以后，需要将我们的Timebase Source修改一下，修改成除了滴答滴答定时器的其他定时器，就像这样



这里我使用的TIM2作为Timebase Source

|为什么不可以使用滴答定时器呢？

在FreeRTOS中我们的SysTick定时器被用于了我们的始终基准，它用来实现我们的仍无切换，我们的SysTick定时器每次触发我们的中断（默认是一毫秒，可以自行修改为其他值）

|Timebase Source是干嘛的呢？

简单的说，正常裸机开发中我们的SysTick定时器是用来主要是用来进行我们的HAL_Delay()延时的，使用其作为基准，前面我们说了SysTick定时器被用于系统任务切换了，所以它就是一直在工作，或者说一直在触发中断，这样的话我们的HAL_Delay()就需用使用我们的其他定时器进行替代其作用了

为了方便移植展示，这里就生成一个Keil的项目吧！

②.FreeRTOS内核下载

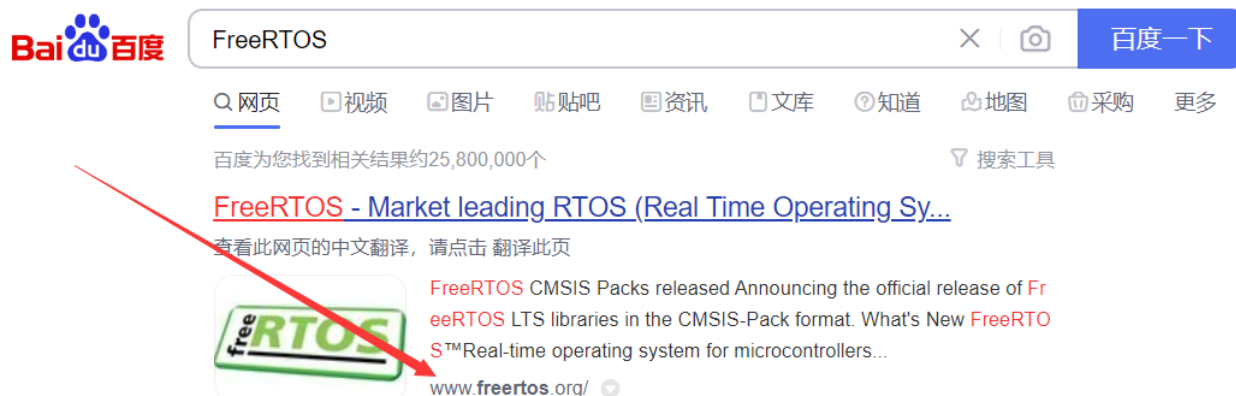
以我们自己的能力肯定是不可能自己写出来操作系统的，这一步将会教大家如何去下载FreeRTOS内核以及如何进行移植到我们的项目里面

进入FreeRTOS的官网

可以选择百度直接搜索FreeRTOS



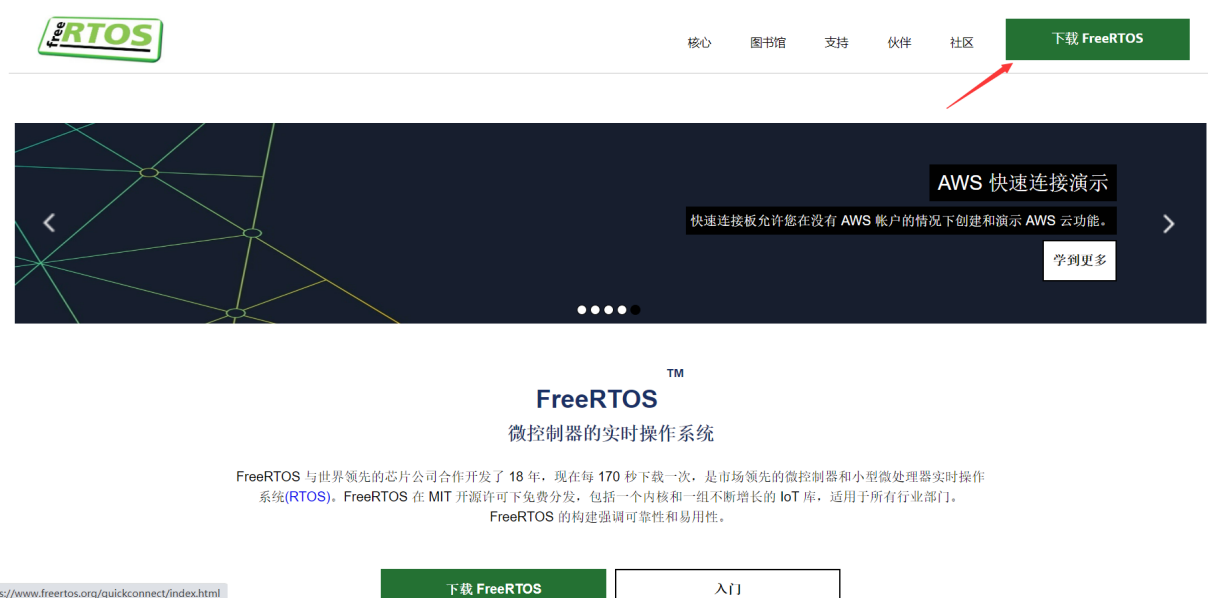
然后选择



也可以直接点击该链接进入

<https://www.freertos.org/>

然后点击右上角下载FreeRTOS



来到版本选择界面

Download FreeRTOS

Download the latest FreeRTOS and Long Term Support (LTS) packages below.

The [FAQ](#) describes the difference between individual libraries and library packages, and provides [links to individual library repositories](#).

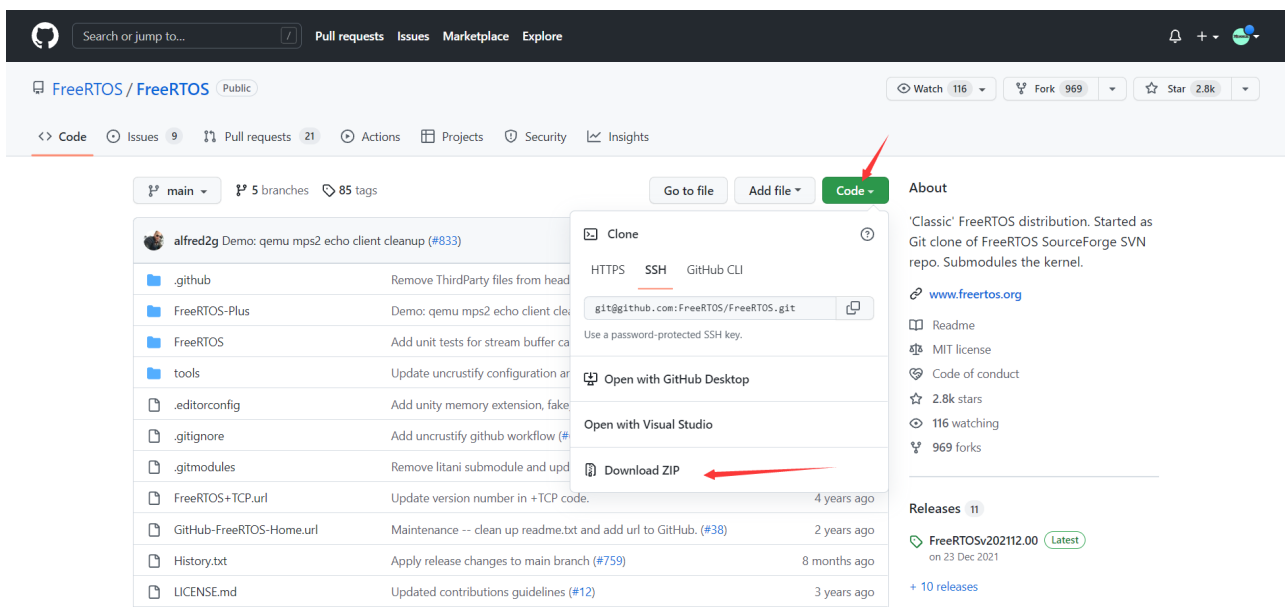
FreeRTOS 202112.00 Package containing the latest FreeRTOS Kernel , FreeRTOS+ libraries and AWS IoT libraries , along with example projects. Source code is also available on Github . Separately, the latest FreeRTOS Kernel can also be downloaded from here .	Download
FreeRTOS LTS 202012.04 Package containing the FreeRTOS LTS libraries, which includes the FreeRTOS kernel and IoT libraries without example projects. See the LTS Libraries page for additional details. Source code is also available on Github .	Download

上面的是FreeRTOS的最新版本，下面的就是长期维护的版本

我们直接点击上面这个Github的超链接

FreeRTOS 202112.00 Package containing the latest FreeRTOS Kernel , FreeRTOS+ libraries and AWS IoT libraries , along with example projects. Source code is also available on Github . Separately, the latest FreeRTOS Kernel can also be downloaded from here .	Download
---	--------------------------

然后就依次带点击Code和Download ZIP



不过我发现，这个下载下来好像是不完整的

Cloning this repository

This repo uses [Git Submodules](#) to bring in dependent components.

Note: If you download the ZIP file provided by the GitHub UI, you will not get the contents of the submodules. (The ZIP file is also not a valid git repository)

If using Windows, because this repository and its submodules contain symbolic links, set `core.symlinks` to true with the following command:

```
git config --global core.symlinks true
```

In addition to this, either enable [Developer Mode](#) or, whenever using a git command that writes to the system (e.g. `git pull`, `git clone`, and `git submodule update --init --recursive`), use a console elevated as administrator so that git can properly create symbolic links for this repository. Otherwise, symbolic links will be written as normal files with the symbolic links' paths in them as text. [This](#) gives more explanation.

To clone using HTTPS:

```
git clone https://github.com/FreeRTOS/FreeRTOS.git --recurse-submodules
```

这里也写了它的子模块是不会直接下载进去的，这里我们就需要继续往下翻

Kernel source code and example projects

`FreeRTOS/Source` contains the FreeRTOS kernel source code (submoduled from <https://github.com/FreeRTOS/FreeRTOS-Kernel>).

`FreeRTOS/Demo` contains pre-configured example projects that demonstrate the FreeRTOS kernel executing on different hardware platforms and using different compilers.

翻译过来就是这样的

内核源代码和示例项目

FreeRTOS/Source 包含 FreeRTOS 内核源代码（来自<https://github.com/FreeRTOS/FreeRTOS-Kernel>的子模块）。

FreeRTOS/Demo 包含预配置的示例项目，这些项目演示了在不同硬件平台上执行并使用不同编译器的 FreeRTOS 内核。

内核代码在这里我们点击超链接，同样的方法进行下载

这里就快速掠过，如果无法下载应该就是网络问题了













下载好，然后进行一下解压

 FreeRTOS-Kernel	2022/8/8 17:46	文件夹
 FreeRTOS-main	2022/8/8 16:53	文件夹

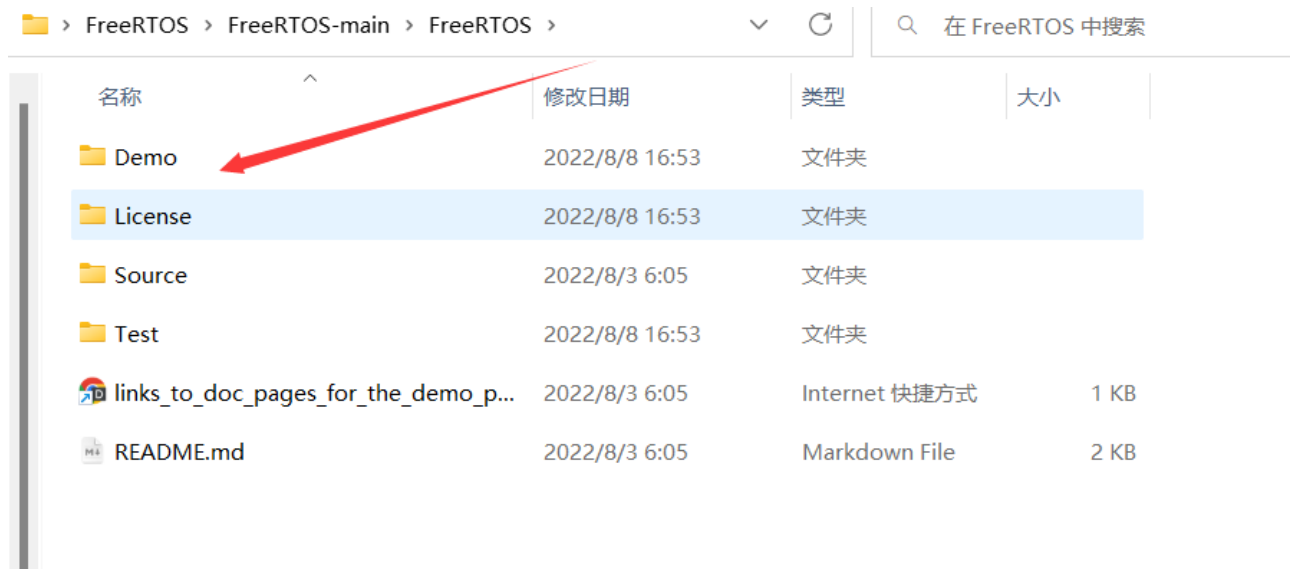
解压好了就是这个样子

FreeRTOS-Kernel是FreeRTOS的内核文件

FreeRTOS-main里面文件很多，我们主要是看里面的Demo

📁 > FreeRTOS > FreeRTOS-main >		⌵ ↻	🔍 在 FreeRTOS-main 中搜索	
名称	修改日期	类型	大小	
 .github	2022/8/8 16:53	文件夹		
 FreeRTOS	2022/8/8 16:53	文件夹		
 FreeRTOS-Plus	2022/8/8 16:53	文件夹		
 tools	2022/8/8 16:53	文件夹		
 .editorconfig	2022/8/3 6:05	Editor Config 源...	1 KB	
 .gitignore	2022/8/3 6:05	Git Ignore 源文件	1 KB	
 .gitmodules	2022/8/3 6:05	txtfile	4 KB	
 FreeRTOS+TCP	2022/8/3 6:05	Internet 快捷方式	1 KB	
 GitHub-FreeRTOS-Home	2022/8/3 6:05	Internet 快捷方式	1 KB	
 History.txt	2022/8/3 6:05	文本文档	141 KB	
 lexicon.txt	2022/8/3 6:05	文本文档	43 KB	
 LICENSE.md	2022/8/3 6:05	Markdown File	2 KB	

然后接着点击



里面的文件非常多，不过我们可以找到这样一个文件夹

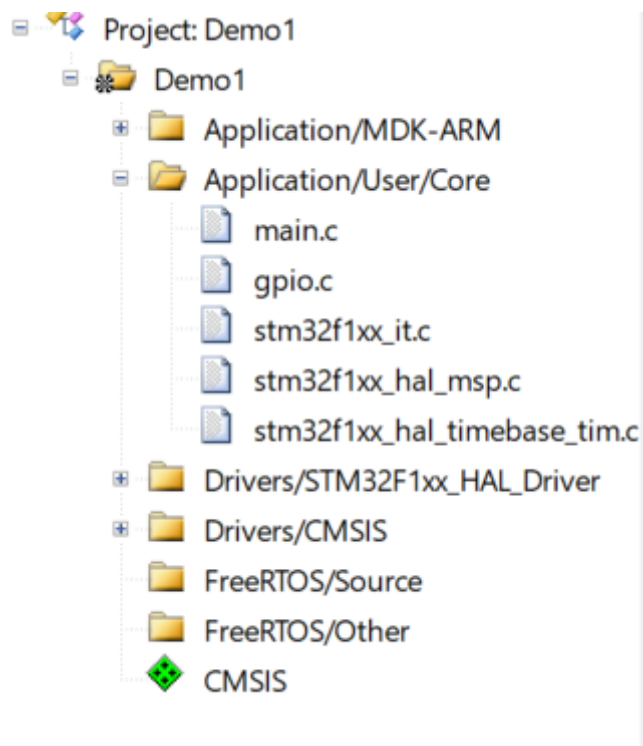


这里有系统已经移植好的实例，但是这并不是我们直接移植的，感兴趣的小伙伴可以先打看官方移植好的能不能看懂

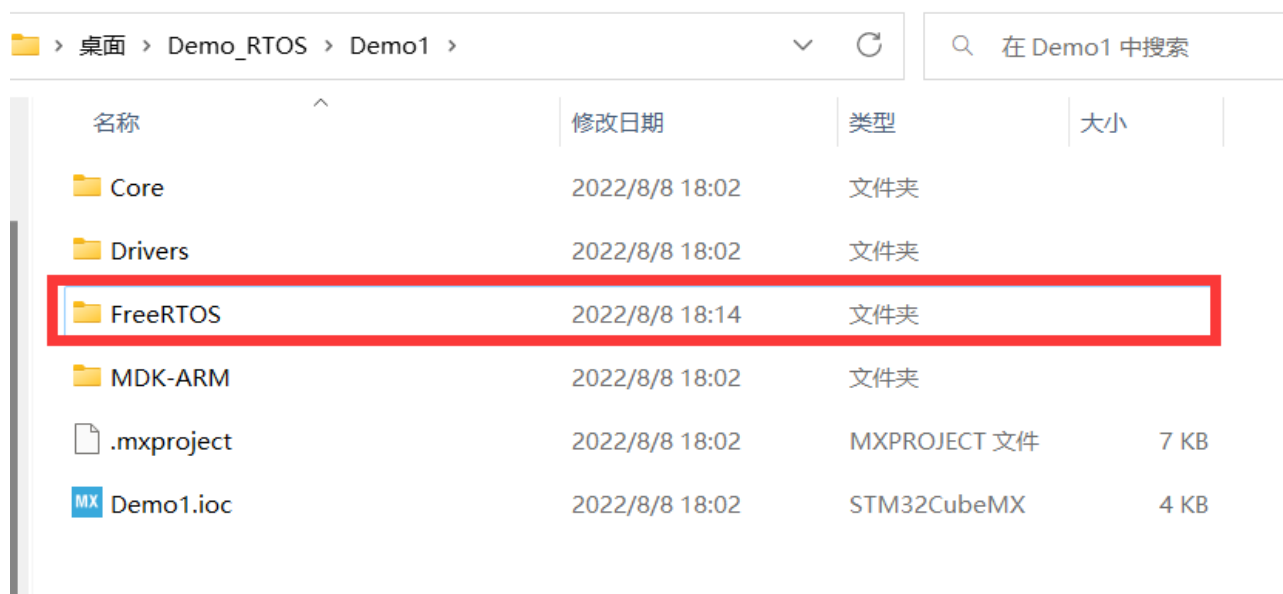
③.手动移植FreeRTOS内核文件

1.打开第一步建立的新项目

2.左侧项目文件添加两个新的分组



3.建立对应的FreeRTOS文件夹



4.打开FreeRTOS-Kernel文件夹

5.主目录下所有的.C后缀的文件全部复制我们的FreeRTOS文件夹下（一共七个）

桌面 > Demo_RTOS > Demo1 > FreeRTOS					在 FreeRTOS 中搜索
名称	修改日期	类型	大小		
croutine.c	2022/8/8 17:10	C 源文件	16 KB		
event_groups.c	2022/8/8 17:10	C 源文件	32 KB		
list.c	2022/8/8 17:10	C 源文件	11 KB		
queue.c	2022/8/8 17:10	C 源文件	124 KB		
stream_buffer.c	2022/8/8 17:10	C 源文件	62 KB		
tasks.c	2022/8/8 17:10	C 源文件	220 KB		
timers.c	2022/8/8 17:10	C 源文件	49 KB		

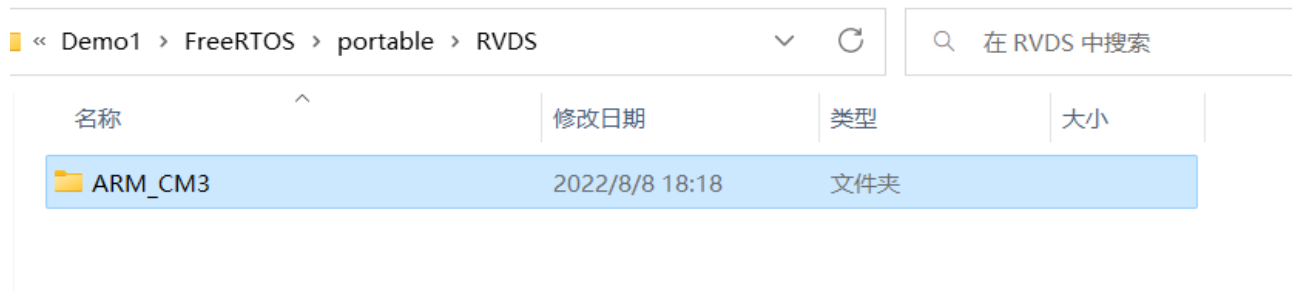
6.复制include文件夹以及portable文件夹到FreeRTOS文件夹下

桌面 > Demo_RTOS > Demo1 > FreeRTOS >					在 FreeRTOS 中搜索
名称	修改日期	类型	大小		
include	2022/8/8 18:18	文件夹			
portable	2022/8/8 18:18	文件夹			
croutine.c	2022/8/8 17:10	C 源文件	16 KB		
event_groups.c	2022/8/8 17:10	C 源文件	32 KB		
list.c	2022/8/8 17:10	C 源文件	11 KB		
queue.c	2022/8/8 17:10	C 源文件	124 KB		
stream_buffer.c	2022/8/8 17:10	C 源文件	62 KB		
tasks.c	2022/8/8 17:10	C 源文件	220 KB		
timers.c	2022/8/8 17:10	C 源文件	49 KB		

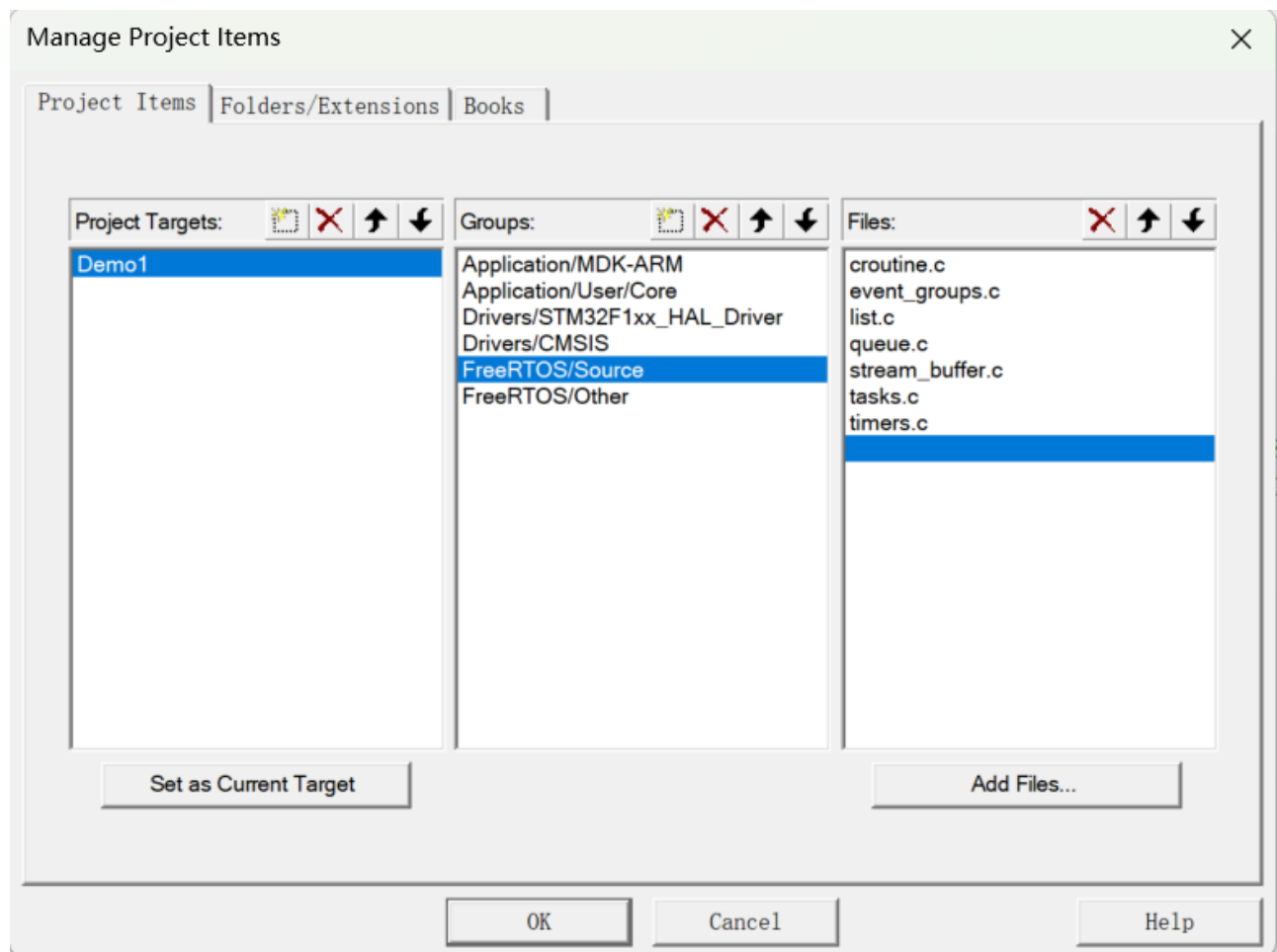
7.保留FreeRTOS/portable文件夹中的Keil、MemMang、RVDS文件夹，其余全部删除

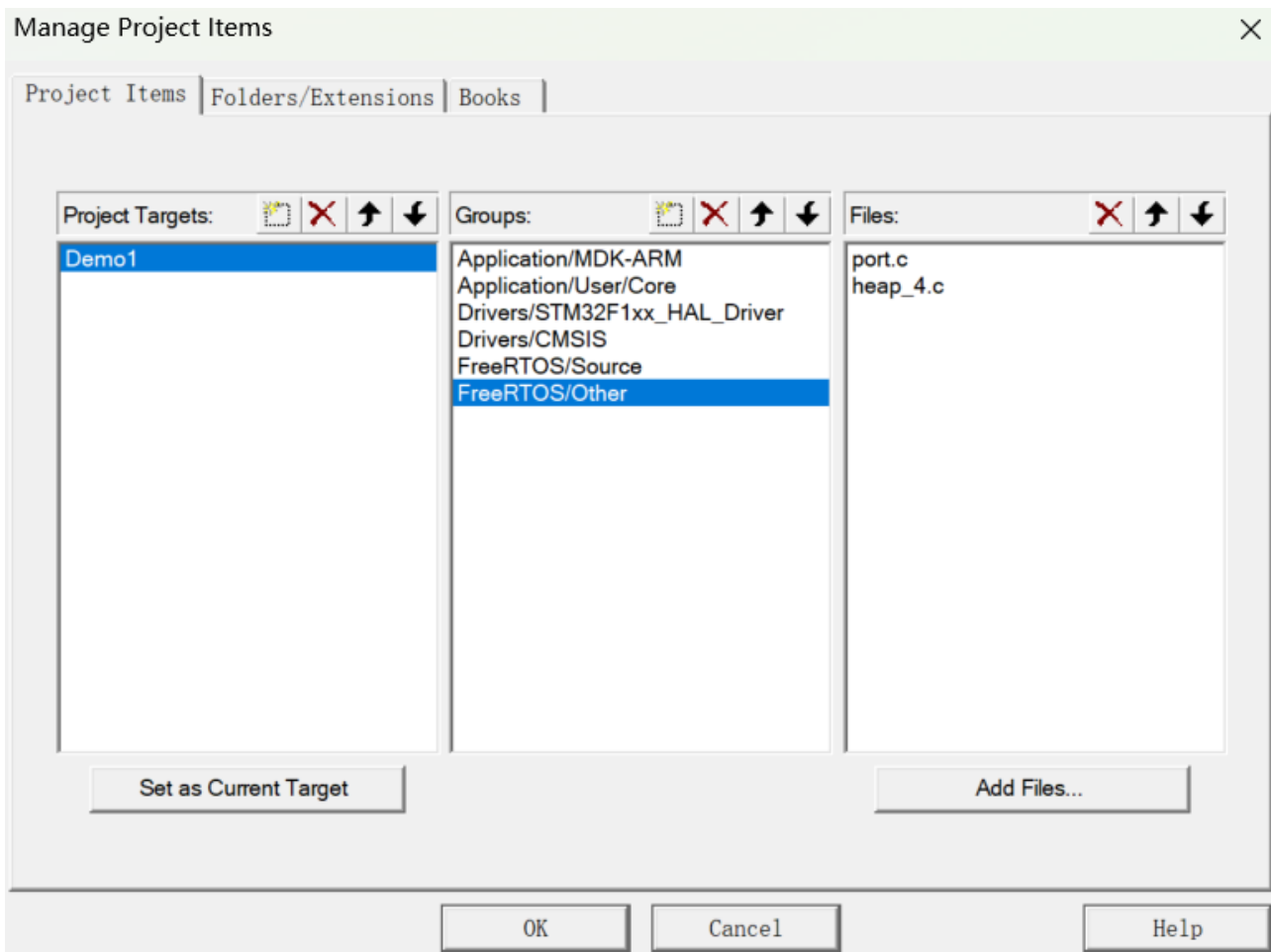
« Demo_RTOS > Demo1 > FreeRTOS > portable					在 portable 中搜索
名称	修改日期	类型	大小		
Keil	2022/8/8 18:18	文件夹			
MemMang	2022/8/8 18:18	文件夹			
RVDS	2022/8/8 18:18	文件夹			

8.保留FreeRTOS/portable/RVDS文件夹中的ARM_CM3文件夹

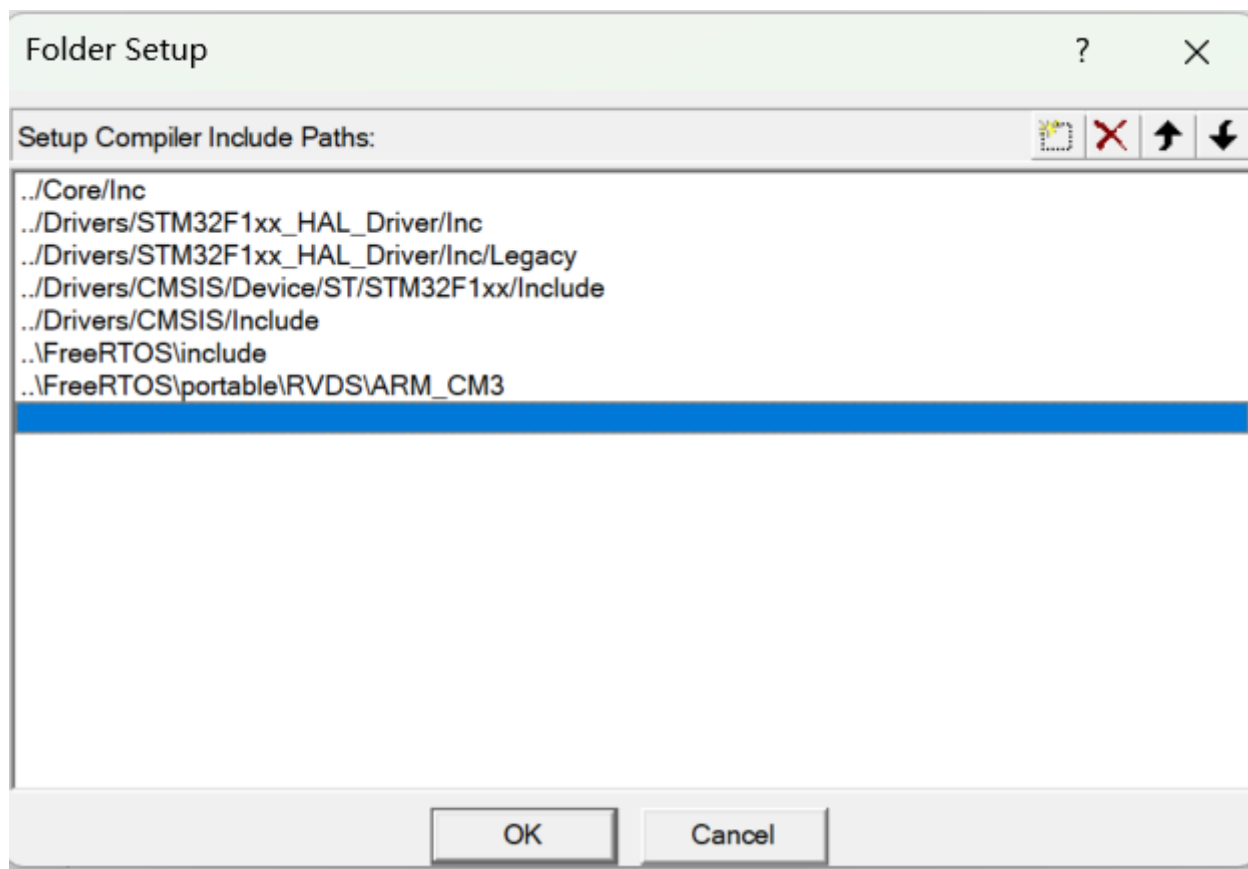


9.然后如图进行Keil项目的分组文件配置





10. 添加对应的头文件



11.直接编译（出现9个错误）

```
Build Output
compiling list.c...
..\FreeRTOS\include\FreeRTOS.h(59): error: #5: cannot open source input file "FreeRTOSConfig.h": No such file or directory
#include "FreeRTOSConfig.h"
..\FreeRTOS\list.c: 0 warnings, 1 error
compiling event_groups.c...
..\FreeRTOS\include\FreeRTOS.h(59): error: #5: cannot open source input file "FreeRTOSConfig.h": No such file or directory
#include "FreeRTOSConfig.h"
..\FreeRTOS\event_groups.c: 0 warnings, 1 error
compiling queue.c...
..\FreeRTOS\include\FreeRTOS.h(59): error: #5: cannot open source input file "FreeRTOSConfig.h": No such file or directory
#include "FreeRTOSConfig.h"
..\FreeRTOS\queue.c: 0 warnings, 1 error
compiling stream_buffer.c...
..\FreeRTOS\include\FreeRTOS.h(59): error: #5: cannot open source input file "FreeRTOSConfig.h": No such file or directory
#include "FreeRTOSConfig.h"
..\FreeRTOS\stream_buffer.c: 0 warnings, 1 error
compiling tasks.c...
..\FreeRTOS\include\FreeRTOS.h(59): error: #5: cannot open source input file "FreeRTOSConfig.h": No such file or directory
#include "FreeRTOSConfig.h"
..\FreeRTOS\tasks.c: 0 warnings, 1 error
compiling heap_4.c...
..\FreeRTOS\include\FreeRTOS.h(59): error: #5: cannot open source input file "FreeRTOSConfig.h": No such file or directory
#include "FreeRTOSConfig.h"
..\FreeRTOS\portable\MemMang\heap_4.c: 0 warnings, 1 error
compiling port.c...
..\FreeRTOS\include\FreeRTOS.h(59): error: #5: cannot open source input file "FreeRTOSConfig.h": No such file or directory
#include "FreeRTOSConfig.h"
..\FreeRTOS\portable\RVDS\ARM_CM3\port.c: 0 warnings, 1 error
compiling timers.c...
..\FreeRTOS\include\FreeRTOS.h(59): error: #5: cannot open source input file "FreeRTOSConfig.h": No such file or directory
#include "FreeRTOSConfig.h"
..\FreeRTOS\timers.c: 0 warnings, 1 error
compiling stm32f1xx_hal_flash_ex.c...
compiling system_stm32f1xx.c...
compiling stm32f1xx_hal_exti.c...
"Demol\Demol.axf" - 9 Error(s), 0 Warning(s).
Target not created.
Build Time Elapsed: 00:00:04
```

④.项目问题修复

缺少**FreeRTOSConfig.h**文件（**FreeRTOS**的配置文件）

« FreeRTOS > Demo > CORTEX_STM32F103_Keil >

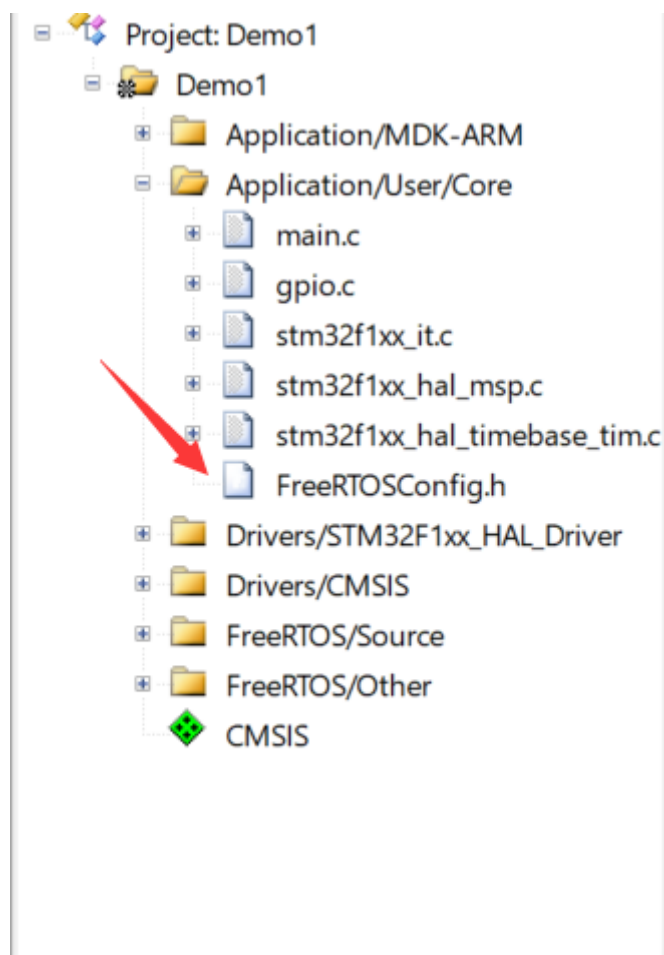
名称	修改日期	类型	大小
ParTest	2022/8/8 16:53	文件夹	
serial	2022/8/8 16:53	文件夹	
STM32F10xFWLib	2022/8/8 16:53	文件夹	
FreeRTOSConfig.h	2022/8/3 6:05	C Header 源文件	4 KB
LCD_Message.h	2022/8/3 6:05	C Header 源文件	58 KB
main.c	2022/8/3 6:05	C 源文件	14 KB
RTOSDemo.Opt	2022/8/3 6:05	OPT 文件	5 KB
RTOSDemo.plg	2022/8/3 6:05	PLG 文件	6 KB
RTOSDemo.sct	2022/8/3 6:05	Windows Script ...	1 KB
RTOSDemo.Uv2	2022/8/3 6:05	磳ision2 & 磳isio...	5 KB
spi_flash.c	2022/8/3 6:05	C 源文件	19 KB
STM32F10x.s	2022/8/3 6:05	S 文件	12 KB
stm32f10x_conf.h	2022/8/3 6:05	C Header 源文件	6 KB
timertest.c	2022/8/3 6:05	C 源文件	6 KB

示例项目中复制粘贴并且添加到项目文件夹中（记得添加头文件路径）

桌面 > Demo_RTOS > Demo1 > FreeRTOS >

在 FreeRTOS 中搜索

名称	修改日期	类型	大小
include	2022/8/8 18:18	文件夹	
portable	2022/8/8 18:21	文件夹	
croutine.c	2022/8/8 17:10	C 源文件	16 KB
event_groups.c	2022/8/8 17:10	C 源文件	32 KB
FreeRTOSConfig.h	2022/8/3 6:05	C Header 源文件	4 KB
list.c	2022/8/8 17:10	C 源文件	11 KB
queue.c	2022/8/8 17:10	C 源文件	124 KB
stream_buffer.c	2022/8/8 17:10	C 源文件	62 KB
tasks.c	2022/8/8 17:10	C 源文件	220 KB
timers.c	2022/8/8 17:10	C 源文件	49 KB



|编译，还是报错

```
Build Output
compiling stm32f1xx_hal_rcc.c...
compiling stm32f1xx_hal_flash_ex.c...
compiling stm32f1xx_hal_flash.c...
compiling stm32f1xx_hal_cortex.c...
compiling stm32f1xx_hal_tim_ex.c...
compiling stm32f1xx_hal_rcc_ex.c...
compiling croutine.c...
compiling list.c...
compiling stream_buffer.c...
..\FreeRTOS\StreamBuffer.c(40): error: #35: #error directive: INCLUDE xTaskGetCurrentTaskHandle must be set to 1 to build stream_buffer.c
#error INCLUDE xTaskGetCurrentTaskHandle must be set to 1 to build stream_buffer.c
..\FreeRTOS\stream_buffer.c: 0 warnings, 1 error
compiling timers.c...
compiling event_groups.c...
compiling heap_4.c...
compiling queue.c...
compiling port.c...
compiling tasks.c...
compiling stm32f1xx_hal_tim.c...
compiling stm32f1xx_hal_exti.c...
compiling system_stm32f1xx.c...
"Demo1\Demo1.axf" - 1 Error(s), 0 Warning(s).
Target not created.
Build Time Elapsed: 00:00:03
```

|在FreeRTOSConfig.h中添加#define INCLUDE_xTaskGetCurrentTaskHandle 1

|编译无错误

```
Build Output
compiling list.c...
compiling event_groups.c...
compiling timers.c...
compiling heap_4.c...
compiling stream_buffer.c...
compiling queue.c...
compiling port.c...
compiling tasks.c...
linking...
Program Size: Code=3544 RO-data=292 RW-data=16 ZI-data=1704
FromELF: creating hex file...
"Demo1\Demo1.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:01
```

⑤.项目添加处理

虽然没有错误了，但是我们的移植没有完成，还有一些小步骤需要完成，的确有些繁琐，好在逻辑性还是比较强的，理解起来相对容易

|在FreeRTOSConfig.h中添加#define xPortPendSVHandler PendSV_Handler

|在FreeRTOSConfig.h中添加#define xPortSysTickHandler SysTick_Handler

|在FreeRTOSConfig.h中添加#define vPortSVCHandler SVC_Handler

左边三个是我们Free RTOS中定义好的函数，右边的是系统项目本来就定义好的函数，他们的作用小伙伴们可以去了解一下，作为初学者这里不进行拓展，反正就是和我们操作系统的中断还有任务切换有关的函数

|编译，还是报错，重复定义

```
Build Output
compiling stream_buffer.c...
compiling port.c...
compiling tasks.c...
linking...
Demo1\Demo1.axf: Error: L6200E: Symbol SVC_Handler multiply defined (by port.o and stm32f1xx_it.o).
Demo1\Demo1.axf: Error: L6200E: Symbol PendSV_Handler multiply defined (by port.o and stm32f1xx_it.o).
Demo1\Demo1.axf: Error: L6200E: Symbol SysTick_Handler multiply defined (by port.o and stm32f1xx_it.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 3 error messages.
"Demo1\Demo1.axf" - 3 Error(s), 0 Warning(s).
Target not created.
Build Time Elapsed: 00:00:01
```

|处理错误

|进入对应的文件**stm32f1xx_it.c**删除重复的**3**个函数

|OK啦！

⑥.FreeRTOS点灯项目创建

弄了这么久，终于移植完成了，我们现在开始创建我们的任务吧！

首先回到项目中我们需要

相关的头文件引入

用到的头文件有

“FreeRTOS.h”（操作系统相关）

"task.h"（任务相关）

```
9  /* Includes -----
0  #include "main.h"
1  #include "gpio.h"
2
3  /* Private includes -----
4  /* USER CODE BEGIN Includes */
5  #include "FreeRTOS.h"
6  #include "task.h"
7  /* USER CODE END Includes */
8
```

任务函数创建

```
//任务1
void vTask1( void *pvParameters )
{
    for( ;; )
    {
        HAL_GPIO_WritePin(LED1_GPIO_Port,LED1_Pin,0);
        HAL_Delay(1000);
        HAL_GPIO_WritePin(LED1_GPIO_Port,LED1_Pin,1);
        HAL_Delay(1000);
    }
}

//任务2
```

```

void vTask2( void *pvParameters )
{
    for( ;; )
    {
        HAL_GPIO_WritePin(LED2_GPIO_Port,LED2_Pin,0);
        HAL_Delay(500);
        HAL_GPIO_WritePin(LED2_GPIO_Port,LED2_Pin,1);
        HAL_Delay(500);
    }
}

```

主函数创建任务，以及开启任务调度器

```

int main(void)
{
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration-----
    -----*/

    /* Reset of all peripherals, Initializes the Flash interface and
    the Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */

    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    /* USER CODE BEGIN 2 */
    //创建任务1

```



```

xTaskCreate(vTask1, "LED1", 128, NULL, 1, NULL);
//创建任务2
xTaskCreate(vTask2, "LED2", 128, NULL, 1, NULL);
//启动任务调度器
vTaskStartScheduler();
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
}

```

⑦.程序烧录到开发板查看效果

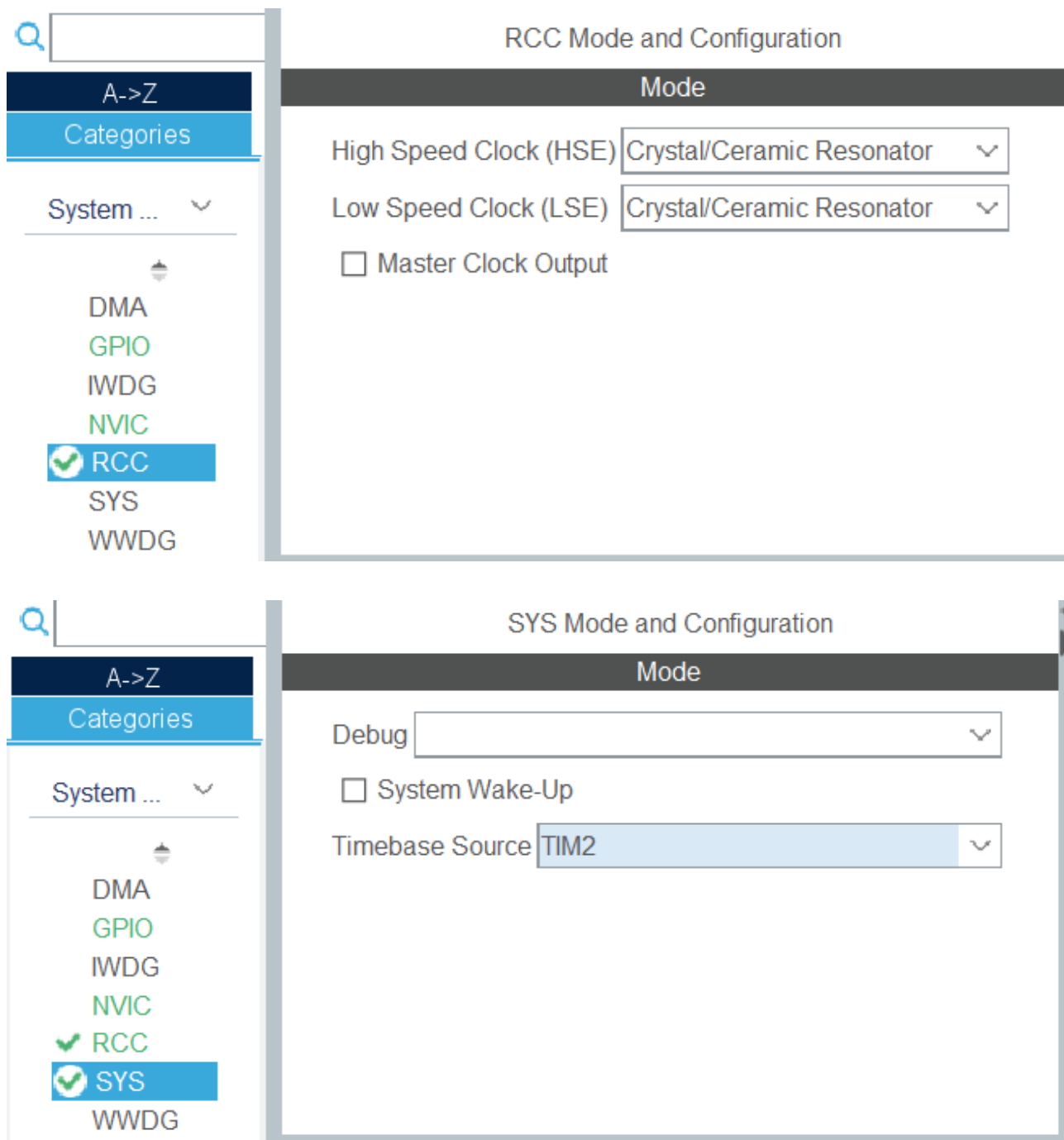
编译项目，然后进行简单的烧录，并且查看效果

贰：CubeMX快速生成FreeRTOS项目

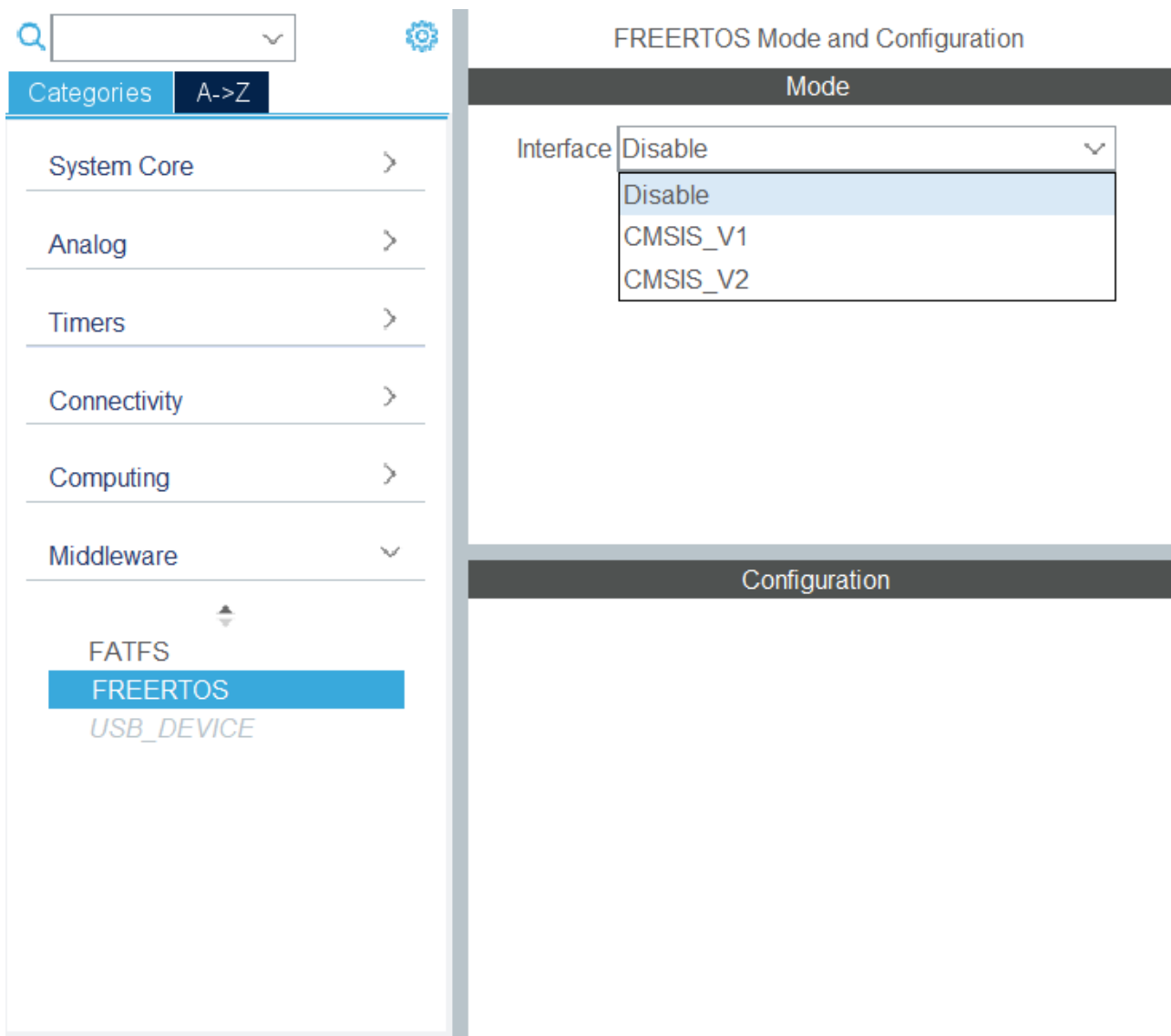
前面我们已经成功手动移植了我们的FreeRTOS的操作系统，步骤相对来说还是比较繁琐的，在我们的开发中肯定不可能每次都去这样的移植，所以我们本期视频就是教大家一个更快捷高效的方法，也就是使用我们的CubeMX快速生产FreeRTOS的项目，并且达到我们上期视频同样的效果，好了我们现在开始，这里我将会使用CubeIDE来生成项目

①正常流程初始化项目

我们就按照平常初始化项目，把我们的时钟频率设置好以及我们上节课讲到了基本时钟源改为其他的定时器



②找到**Middleware**选项然后点击**FREERTOS**



③选择**CMSIS_V1**

版本解释

V1的这个版本是老版本，然后V2版本是新版本，其实区别没有很大版本也没有差很多，不过相对来说我们直接选择V1就可以了，因为使用的人是比较多的

④参数介绍

FREERTOS Mode and Configuration

Mode

Interface

CMSIS_V1

Configuration

Reset Configuration

<input checked="" type="checkbox"/> Mutexes	<input checked="" type="checkbox"/> Events	<input checked="" type="checkbox"/> FreeRTOS Heap Usage
<input checked="" type="checkbox"/> User Constants	<input checked="" type="checkbox"/> Tasks and Queues	<input checked="" type="checkbox"/> Timers and Semaphores
<input checked="" type="checkbox"/> Config parameters	<input checked="" type="checkbox"/> Include parameters	<input checked="" type="checkbox"/> Advanced settings

可以看到我们需要设置的参数是非常非常多的，这里阿熊就给大家简单的解释一些常用的后续碰到会陆续的进行解释

Config parameters（配置参数）

FreeRTOS version	10.0.1
CMSIS-RTOS version	1.02
✖ Kernel settings	
USE_PREEMPTION	Enabled
CPU_CLOCK_HZ	SystemCoreClock
TICK_RATE_HZ	1000
MAX_PRIORITIES	7
MINIMAL_STACK_SIZE	128 Words
MAX_TASK_NAME_LEN	16
USE_16_BIT_TICKS	Disabled
IDLE_SHOULD_YIELD	Enabled
USE_MUTEXES	Enabled
USE_RECURSIVE_MUTEXES	Disabled
USE_COUNTING_SEMAPHORES	Disabled
QUEUE_REGISTRY_SIZE	8
USE_APPLICATION_TASK_HOOK	Disabled
ENABLE_BACKWARD_COMPATIBILITY	Enabled
USE_PORT_OPTIMISED_TASK_SELECTION	Enabled
USE_TICKLESS_IDLE	Disabled
USE_TASK_NOTIFICATIONS	Enabled
RECORD_STACK_HIGH_ADDRESS	Disabled
✖ Memory management settings	
Memory Allocation	Dynamic / Static
TOTAL_HEAP_SIZE	3072 Bytes
Memory Management scheme	heap_4
✖ Hook function related definitions	
USE_IDLE_HOOK	Disabled
USE_TICK_HOOK	Disabled
USE_MALLOC_FAILED_HOOK	Disabled
USE_DAEMON_TASK_STARTUP_HOOK	Disabled
CHECK_FOR_STACK_OVERFLOW	Disabled
✖ Run time and task stats gathering related definitions	
GENERATE_RUN_TIME_STATS	Disabled
USE_TRACE_FACILITY	Disabled
USE_STATS_FORMATTING_FUNCTIONS	Disabled
✖ Co-routine related definitions	
USE_CO_ROUTINES	Disabled
MAX_CO_ROUTINE_PRIORITIES	2
✖ Software timer definitions	
USE_TIMERS	Disabled
✖ Interrupt nesting behaviour configuration	
LIBRARY_LOWEST_INTERRUPT_PRIORITY	15

这里就是我们Free RTOS的配置参数，可以选择性的开启或者关闭某些功能，我们前面有讲过FreeRTOS的可裁剪性，是它的一大优势，像我们的这些设置里就是开启或关闭它的一些功能

Include parameters（内部参数）

▼ Include definitions

vTaskPrioritySet	Enabled
uxTaskPriorityGet	Enabled
vTaskDelete	Enabled
vTaskCleanUpResources	Disabled
vTaskSuspend	Enabled
vTaskDelayUntil	Disabled
vTaskDelay	Enabled
xTaskGetSchedulerState	Enabled
xTaskResumeFromISR	Enabled
xQueueGetMutexHolder	Disabled
xSemaphoreGetMutexHolder	Disabled
pcTaskGetTaskName	Disabled
uxTaskGetStackHighWaterMark	Disabled
xTaskGetCurrentTaskHandle	Disabled
eTaskGetState	Disabled
xEventGroupSetBitFromISR	Disabled
xTimerPendFunctionCall	Disabled
xTaskAbortDelay	Disabled
xTaskGetHandle	Disabled

这里面是我们开发中可能会用到的一些函数，我们如果用不到的话可以就选择性的将其关闭，这样的话我们的内核大小也会减小很多

Tasks and Queues（任务和队列）

Tasks

Task N...	Priority	Stack ...	Entry F...	Code ...	Param...	Allocati...	Buffer ...	Control...
default...	osPrior...	128	StartD...	Default	NULL	Dynamic	NULL	NULL

Add

Delete

Queues

Queue Name	Queue Size	Item Size	Allocation	Buffer Name	Control Bloc...
------------	------------	-----------	------------	-------------	-----------------

Add

Delete

我们操作系统中的任务还有我们的队列（后续会介绍到）都是可以直接在这里进行一个初始化的，我们只需要去更新内部的函数内容就可以高效的完成我们的项目

Other（其他设置）

这里我们就只介绍我们的这3个比较重要的，其他的全部统称为其他设置，后续的课程中，我们遇到了再讲解

⑤添加Task

直接点击Add或者双击默认的项目

Edit Task	
Task Name	defaultTask
Priority	osPriorityNormal
Stack Size (Words)	128
Entry Function	StartDefaultTask
Code Generation Option	Default
Parameter	NULL
Allocation	Dynamic
Buffer Name	NULL
Control Block Name	NULL
<div>OK Cancel</div>	

其分别对应

任务名称、任务优先级、任务堆栈大小、进入函数名称、代码生成方式、参数、任务类型

任务名称：

就是单纯的名称，一般随便取一个就好了，没有什么实质性的作用，毕竟我们执行的时候用的是函数名称

任务优先级：

任务优先级就很像我们那个中断优先级，不过它与我们的中断优先级是相反的，我们的中断优先级中的中段是数字越小，然后它的优先级就会越高，我们的操作系统中数字越大，优先级越高，并且会优先执行

任务堆栈大小：

就是可以理解为我们的任务执行所需要的空间大小，太大了不是很好，太小了反而会不够，所以这个需要一定的经验去判断

进入函数名称：

就是我们执行任务时对应的那个函数，它的名称是什么？我们的系统会自动给我们写好一个框架

任务类型：

我们任务生成的类型有动态生成，还有静态生成，动态生成的话就是系统，你只要告诉他空间大小系统会自己划分好，而我们静态生成是直接要指定一块地点，相对来说使用是非常少的，而且作为初学者几乎是完全理解不了的，所以我们本视频系列教程是不会涉及到静态生成任务，只会讲解动态任务

Other:

前面就是我们需要理解的，其他的我们后续遇到了会再讲解

添加**LED**的任务：

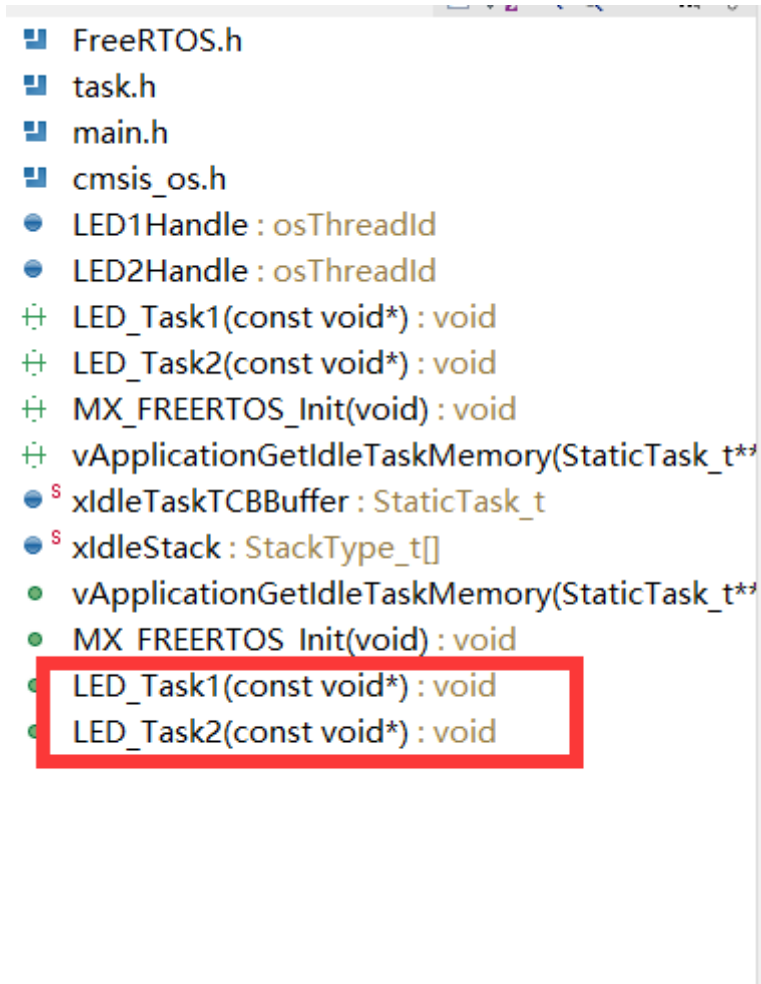
Edit Task		×
Task Name	LED1	
Priority	osPriorityNormal	
Stack Size (Words)	128	
Entry Function	LED_Task1	
Code Generation Option	Default	
Parameter	NULL	
Allocation	Dynamic	
Buffer Name	NULL	
Control Block Name	NULL	
OK		Cancel

Edit Task		×
Task Name	LED2	
Priority	osPriorityNormal	
Stack Size (Words)	128	
Entry Function	LED_Task2	
Code Generation Option	Default	
Parameter	NULL	
Allocation	Dynamic	
Buffer Name	NULL	
Control Block Name	NULL	
OK		Cancel

保存一下，然后生成项目

⑥完成任务函数

可以看到我们生成的项目中，main.C中他并没有那个任务的函数，我们可以找到左边的那个文件管理来，可以找到一个freeRTOS.C的项目文件，然后我们双击打开



然后我们就可以快速的完成一下两个任务，上节课视频我们也有讲过，如何去书写

```
/* USER CODE BEGIN Header_LED_Task1 */
/**
 * @brief Function implementing the LED1 thread.
 * @param argument: Not used
 * @retval None
 */
/* USER CODE END Header_LED_Task1 */
void LED_Task1(void const * argument)
{
    /* USER CODE BEGIN LED_Task1 */
    /* Infinite loop */
```

```

for(;;)
{
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin,1);
    HAL_Delay(1000);
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin,0);
    HAL_Delay(1000);
}
/* USER CODE END LED_Task1 */
}

/* USER CODE BEGIN Header_LED_Task2 */
/**
 * @brief Function implementing the LED2 thread.
 * @param argument: Not used
 * @retval None
 */
/* USER CODE END Header_LED_Task2 */
void LED_Task2(void const * argument)
{
    /* USER CODE BEGIN LED_Task2 */
    /* Infinite loop */
    for(;;)
    {
        HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin,1);
        HAL_Delay(500);
        HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin,0);
        HAL_Delay(500);
    }
    /* USER CODE END LED_Task2 */
}

```

⑦编译，烧录查看现象

可以看到我们的那个实验现象和上期手动移植的现象是完全一样的，我们成功完成了快速生成FreeRTOS的项目并且完成了最基本的任务

⑧代码讲解

```
int main(void)
{
    /* Reset of all peripherals, Initializes the Flash interface and
    the Systick. */
    HAL_Init();
    /* Configure the system clock */
    SystemClock_Config();
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    /* Call init function for freertos objects (in freertos.c) */
    MX_FREERTOS_Init();
    /* Start scheduler */
    osKernelStart();
    /* We should never get here as control is now taken by the
    scheduler */
    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {
        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
}
```

main函数中，我们最开始还是普通的初始化，和我们的普通项目没有区别

接着是

MX_FREERTOS_Init();

我们转跳过去看看

```

void MX_FREERTOS_Init(void) {
    /* Create the thread(s) */
    /* definition and creation of LED1 */
    osThreadDef(LED1, LED_Task1, osPriorityNormal, 0, 128);
    LED1Handle = osThreadCreate(osThread(LED1), NULL);

    /* definition and creation of LED2 */
    osThreadDef(LED2, LED_Task2, osPriorityNormal, 0, 128);
    LED2Handle = osThreadCreate(osThread(LED2), NULL);
}

```

内容也很简单，其实他就是创建了我们的两个项目，然后相对来说小伙伴们可以发现和我们手动一直是不一样的，因为我们的 Free rtos 的那个操作系统在我们那个 ST 官方的那个优化下，他自己将很多函数都进行了重新的宏定义以及方法的改写

我们甚至可以转跳过去看看他是怎么实现的

```

osThreadId osThreadCreate (const osThreadDef_t *thread_def, void
*argument)
{
    TaskHandle_t handle;

    #if( configSUPPORT_STATIC_ALLOCATION == 1 ) && (
configSUPPORT_DYNAMIC_ALLOCATION == 1 )
        if((thread_def->buffer != NULL) && (thread_def->controlblock !=
NULL)) {
            handle = xTaskCreateStatic((TaskFunction_t)thread_def->pthread,
(const portCHAR *)thread_def->name,
                thread_def->stacksize, argument,
makeFreeRtosPriority(thread_def->tpriority),
                thread_def->buffer, thread_def->controlblock);
        }
        else {
            if (xTaskCreate((TaskFunction_t)thread_def->pthread, (const
portCHAR *)thread_def->name,
                thread_def->stacksize, argument,
makeFreeRtosPriority(thread_def->tpriority),
                &handle) != pdPASS) {
                return NULL;
            }
        }
}

```

```

#elif( configSUPPORT_STATIC_ALLOCATION == 1 )

    handle = xTaskCreateStatic((TaskFunction_t)thread_def->pthread,
    (const portCHAR *)thread_def->name,
        thread_def->stacksize, argument,
    makeFreeRtosPriority(thread_def->tpriority),
        thread_def->buffer, thread_def->controlblock);
#else
    if (xTaskCreate((TaskFunction_t)thread_def->pthread, (const
    portCHAR *)thread_def->name,
        thread_def->stacksize, argument,
    makeFreeRtosPriority(thread_def->tpriority),
        &handle) != pdPASS) {

        return NULL;
    }
#endif

    return handle;
}

```

虽然他看起来很复杂，但是我们几乎可以一眼就找到我们上节课用到的那个xTaskCreate函数

在我们系统生成的FreeRTOS的框架下，它是宏定义了很多的函数，但其实用法师和我们的没有什么区别，我们甚至可以直接用我们上节课所写的那些东西,也可以实现同样的效果

这里就不拓展开了，碰到了我们再讲解