南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Embedded System and Microcomputer Principle

LAB10 FreeRTOS

2021 Fall
wangq9@mail.sustech.edu.cn

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

CONTENTS

01

FreeRTOS Description

1. FreeRTOS Description

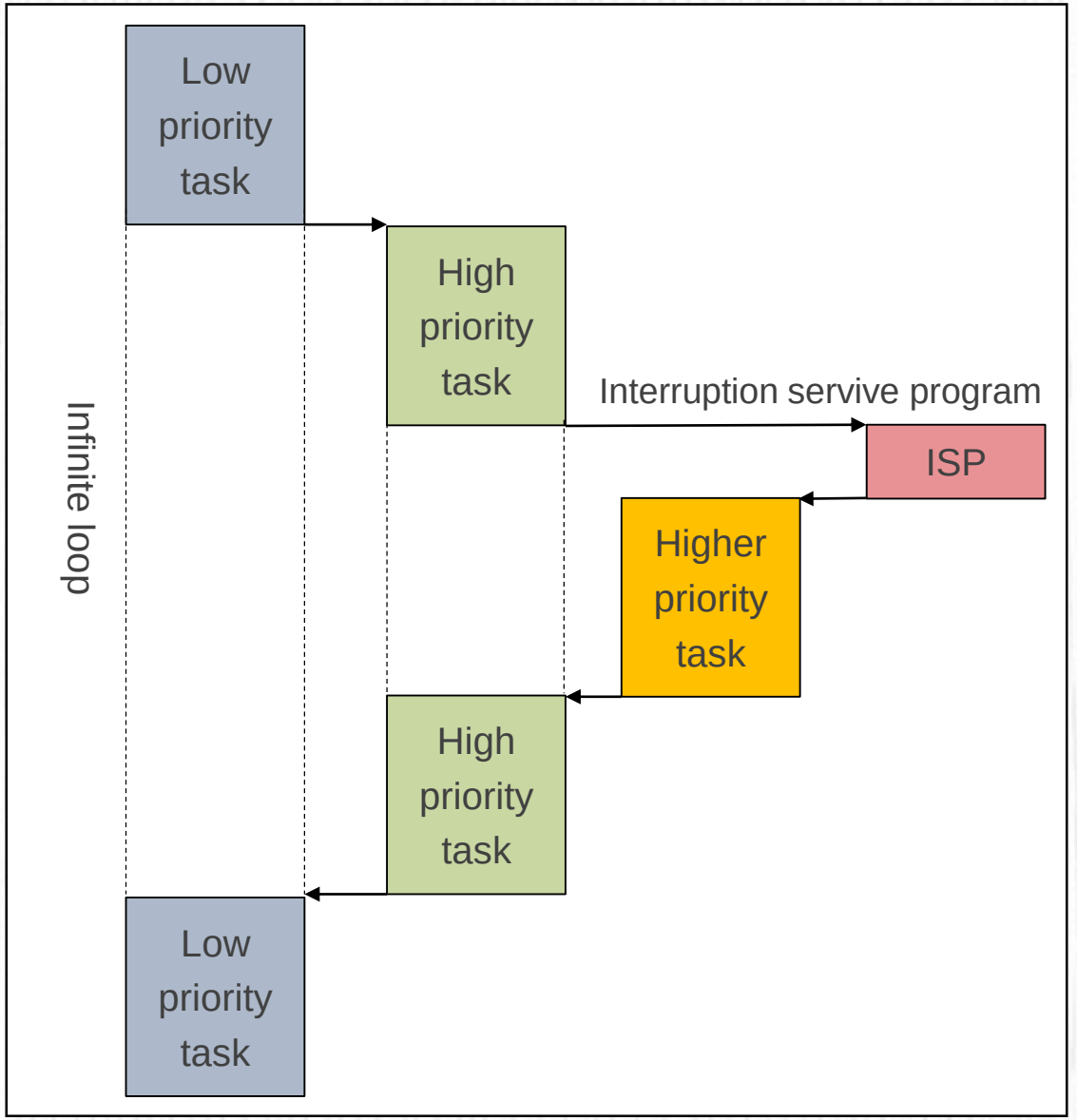
-- What is RTOS

- Real Time OS，即实时操作系统
- RTOS操作系统的核心内容：实时内核
- 在实时操作系统中，我们可以把要实现的功能划分为多个任务，每个任务负责实现其中的一部分，每个任务都是一个很简单的程序，通常是一个无限循环
- 常见的RTOS操作系统：FreeRTOS、μCOS、RT-Thread、RTX、ThreadX、uLinux，DJYOS等
- RTOS的内核负责管理所有的任务，内核决定了运行哪个任务，何时停止当前任务切换到其他任务，这个是内核的多任务管理能力。多任务管理给人的感觉就好像芯片有多个CPU，实现了CPU资源的最大化利用

1. FreeRTOS Description

-- Preemptive kernel

- 在可剥夺型内核中，CPU总是运行多个任务中优先级别最高的那个任务，即使CPU正在运行某个低级别的任务，当有高优先级别的任务准备就绪时，该优先级别的任务就会剥夺正在运行任务的CPU使用权，从而使自己获得CPU的使用权
- 可剥夺型内核实时性较好
- 目前大多数嵌入式实时操作系统是可剥夺型内核



1. FreeRTOS Description

-- What is FreeRTOS

- FreeRTOS是一个可裁剪、可剥夺型的开源多任务内核，而且没有任务数限制
- FreeRTOS提供了实时操作系统所需的所有功能，包括资源管理、同步、任务通信等
- FreeRTOS使用C语言和汇编语言实现，其中大部分功能用C语言编写，极少数的与处理器密切相关的功能用汇编语言编写
- STM32CubeIDE的中间件即包含FreeRTOS
- FreeRTOS官网：http://www.freertos.org/

02

FreeRTOS Task Description

2. FreeRTOS Task Description

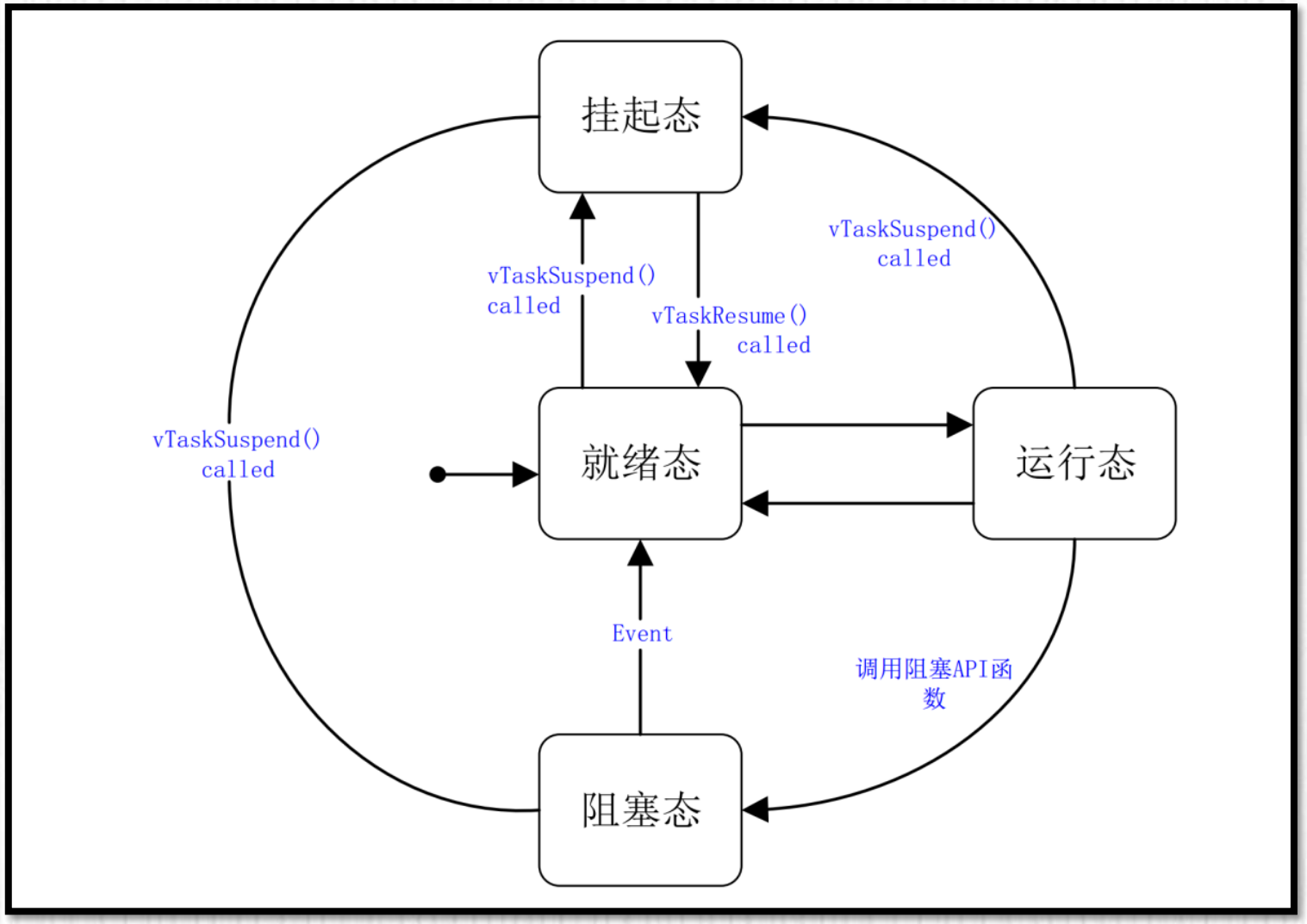
-- FreeRTOS task characteristics

- 简单
- 没有使用限制
- 支持抢占
- 支持优先级
- 每个任务都拥有堆栈导致了 RAM 使用量增大
- 如果使用抢占的话的必须仔细的考虑重入的问题

2. FreeRTOS Task Description

-- FreeRTOS task states

- 运行态
- 就绪态
- 阻塞态
- 挂起态



2. FreeRTOS Task Description
-- FreeRTOS task Priorities

- 任务优先级决定了任务的执行优先级别
- FreeRTOS中任务优先级可选范围为：

  0 ~ configMAX_PRIORITIES-1

- 数字越大，优先级越高

2. FreeRTOS Task Description
-- FreeRTOS API introduction

- Task Creation
  - TaskHandle_t (type)
  - xTaskCreate()
  - xTaskCreateStatic()
  - xTaskCreateRestrictedStatic()
  - vTaskDelete()

- Task Control
  - vTaskDelay()
  - vTaskDelayUntil()
  - xTaskDelayUntil()
  - uxTaskPriorityGet()
  - vTaskPrioritySet()
  - vTaskSuspend()
  - vTaskResume()
  - xTaskResumeFromISR()
  - xTaskAbortDelay()

2. FreeRTOS Task Description
-- FreeRTOS API introduction(continued)

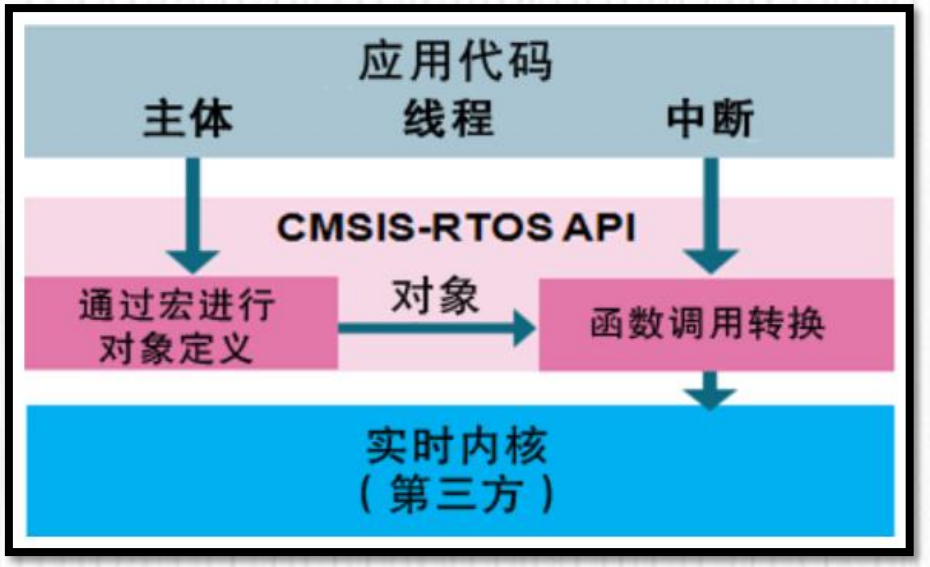
- Task Utilities
  - uxTaskGetSystemState()
  - vTaskGetInfo()
  - xTaskGetApplicationTaskTag()
  - xTaskGetCurrentTaskHandle()
  - xTaskGetHandle()
  - xTaskGetIdleTaskHandle()
  - uxTaskGetStackHighWaterMark()
  - eTaskGetState()
  - pcTaskGetName()
  - xTaskGetTickCount()
  - xTaskGetTickCountFromISR()
  - xTaskGetSchedulerState()
  - uxTaskGetNumberOfTasks()
  - vTaskList()
  - vTaskStartTrace()
  - ulTaskEndTrace()
  - vTaskGetRunTimeStats()
  - vTaskSetApplicationTaskTag()
  - xTaskCallApplicationTaskHook()
  - SetThreadLocalStoragePointer()
  - GetThreadLocalStoragePointer()
  - vTaskSetTimeOutState()
  - xTaskGetCheckForTimeOut()

2. FreeRTOS Task Description
-- CMSIS

- CMSIS: Cortex Microcontroller Software Interface Standard Cortex 微控制器软件接口标准
- 是Cortex-M 处理器系列的与供应商无关的硬件抽象层
- 是ARM和一些编译器厂家以及半导体厂家共同遵循的一套标准，是由ARM专门针对CORTEX-M系列提出的标准。在该标准的约定下，ARM和芯片厂商会提供一些通用的API接口来访问Cortex内核以及一些专用外设
- 只要都是基于Cortex的芯片，代码均是可以复用的
- 使用 CMSIS可以为处理器和外设实现一致且简单的软件接口，从而简化软件的重用、缩短微控制器新开发人员的学习过程，并缩短新设备的上市时间

2. FreeRTOS Task Description
-- CMSIS-RTOS packaging layer

- CMSIS-RTOS是ARM公司为统一操作系统、降低嵌入式门槛而发布的操作系统标准软件接口。
- 通俗讲，CMSIS-RTOS将操作系统（不管是FreeRTOS还是RTX等）屏蔽起来，然后提供CMSIS-RTOS接口函数给最终使用者调用。
- 使用者只需要学习CMSIS-ROTS即可，从而降低学习门槛。
- 目前FreeRTOS和RTX能够很好的支持CMSIS-RTOS，其他有些RTOS还没有做适配
- CMSIS-RTOS官方教程 https://www.keil.com/pack/doc/CMSIS/RTOS/html/index.html



2. FreeRTOS Task Description
-- CMSIS-RTOS kernel control function

内核控制函数		
osStatus osKernelStart (void)	内核开始运行	
int32_t osKernelRunning(void)	返回值为 1 表示正在运行	作用：系统是否正常工作
uint32_t osKernelSysTick (void)	系统当前节拍数	作用：毫秒级计时
osKernelSysTickMicroSec(microsec)	微秒（宏函数）	作用：微秒级计时

2. FreeRTOS Task Description -- CMSIS-RTOS thread management function

线程管理函数	
osThreadDef(name, thread, priority, instances, stacksz)	定义 osThreadDef_t 结构体
osThread(name)	获取 os_thread_def_##name 结构体的指针
osThreadId osThreadCreate (const osThreadDef_t *thread_def, void *argument)	创建线程，错误则返回 NULL
osThreadId osThreadGetId (void)	返回当前线程
osStatus osThreadTerminate (osThreadId thread_id)	终结线程
osStatus osThreadYield (void)	调度一次
osStatus osThreadSetPriority (osThreadId thread_id, osPriority priority);	设置线程优先级
osPriority osThreadGetPriority (osThreadId thread_id)	获取线程优先级

2. FreeRTOS Task Description
-- CMSIS-RTOS wait function

等待函数	
osStatus osDelay (uint32_t millisec)	毫秒级延时
osEvent osWait (uint32_t millisec)	未实现

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

03

How to Program

3. How to Program

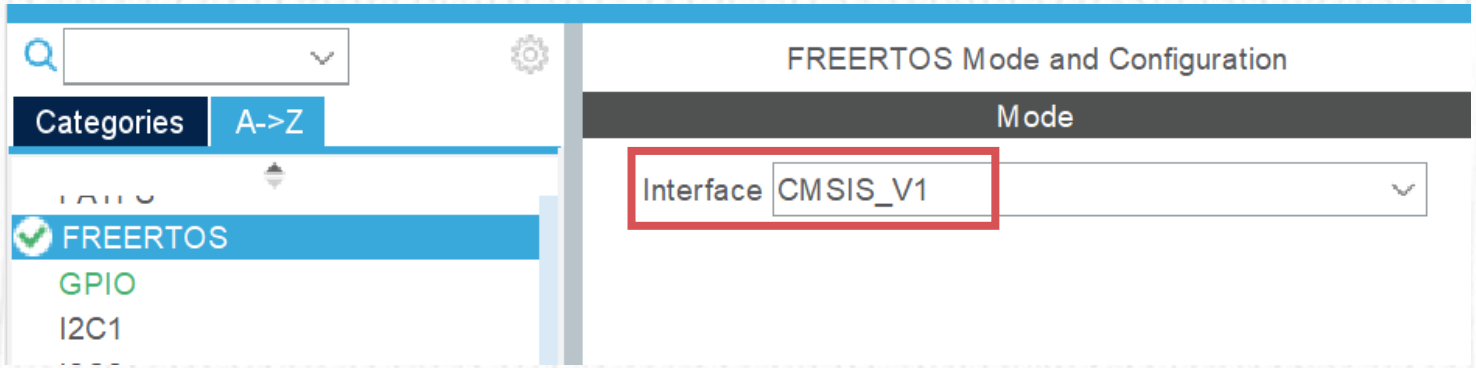
- Our Goal
  - Create two tasks LED0_Task and LED1_Task
  - In LED0_Task, LED0 reverse every 0.5s
  - In LED1_Task, LED1 reverse every 0.5s

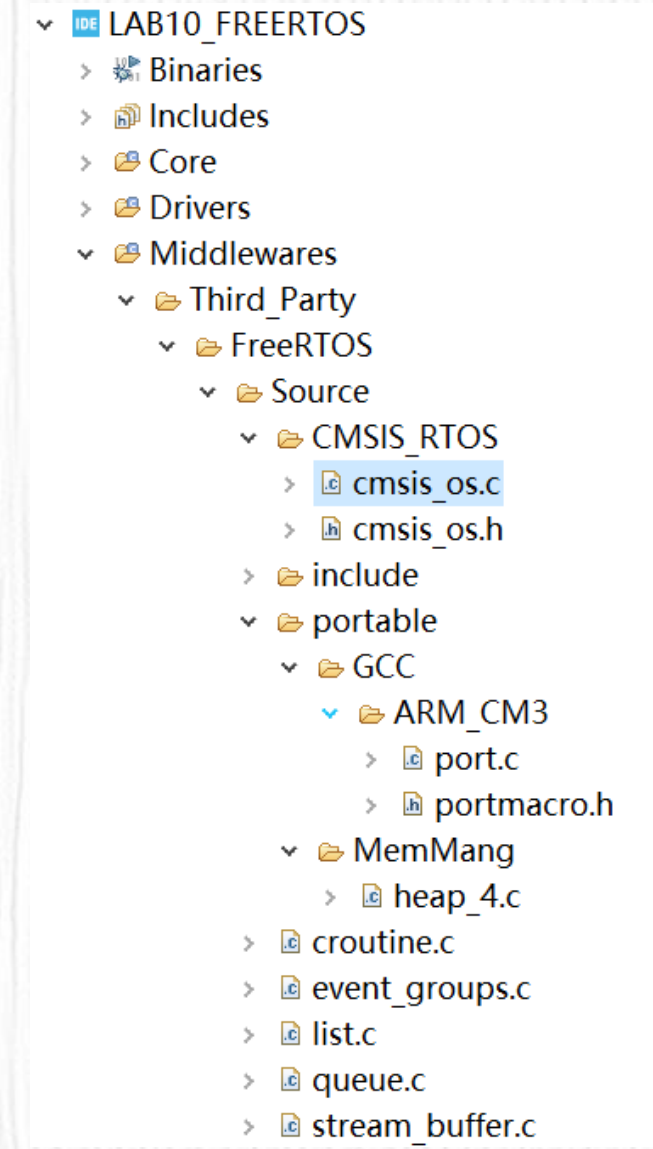
3. How to Program

- Configure SYS
- Configure RCC
- Configure GPIO

3. How to Program

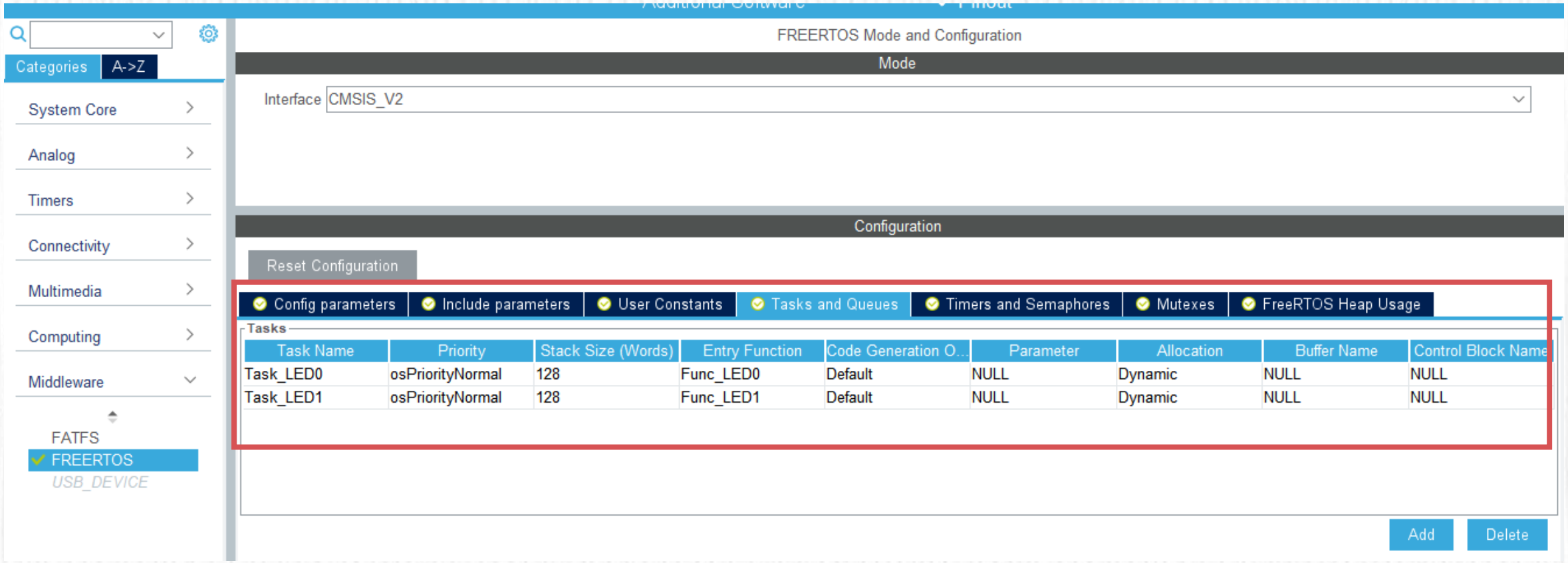
- Configure FREERTOS
  - Set the FreeRTOS API as CMSIS_v1





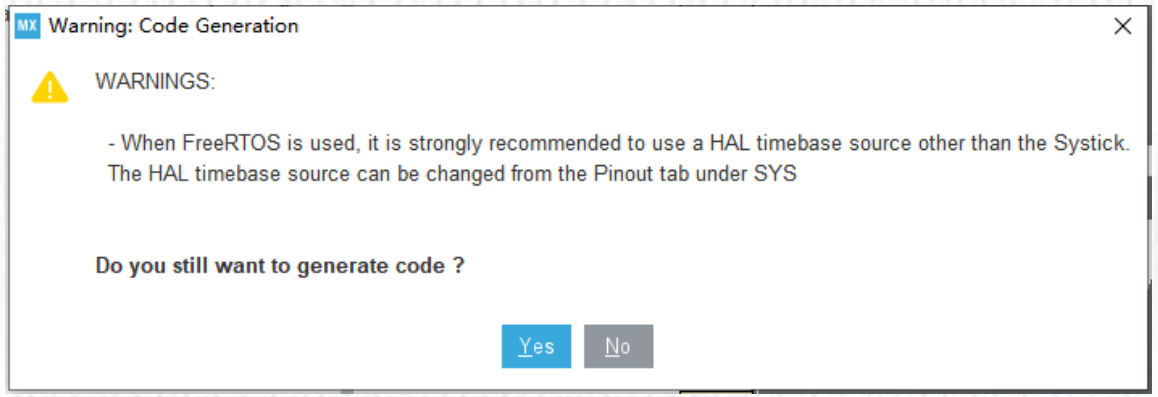
3. How to Program

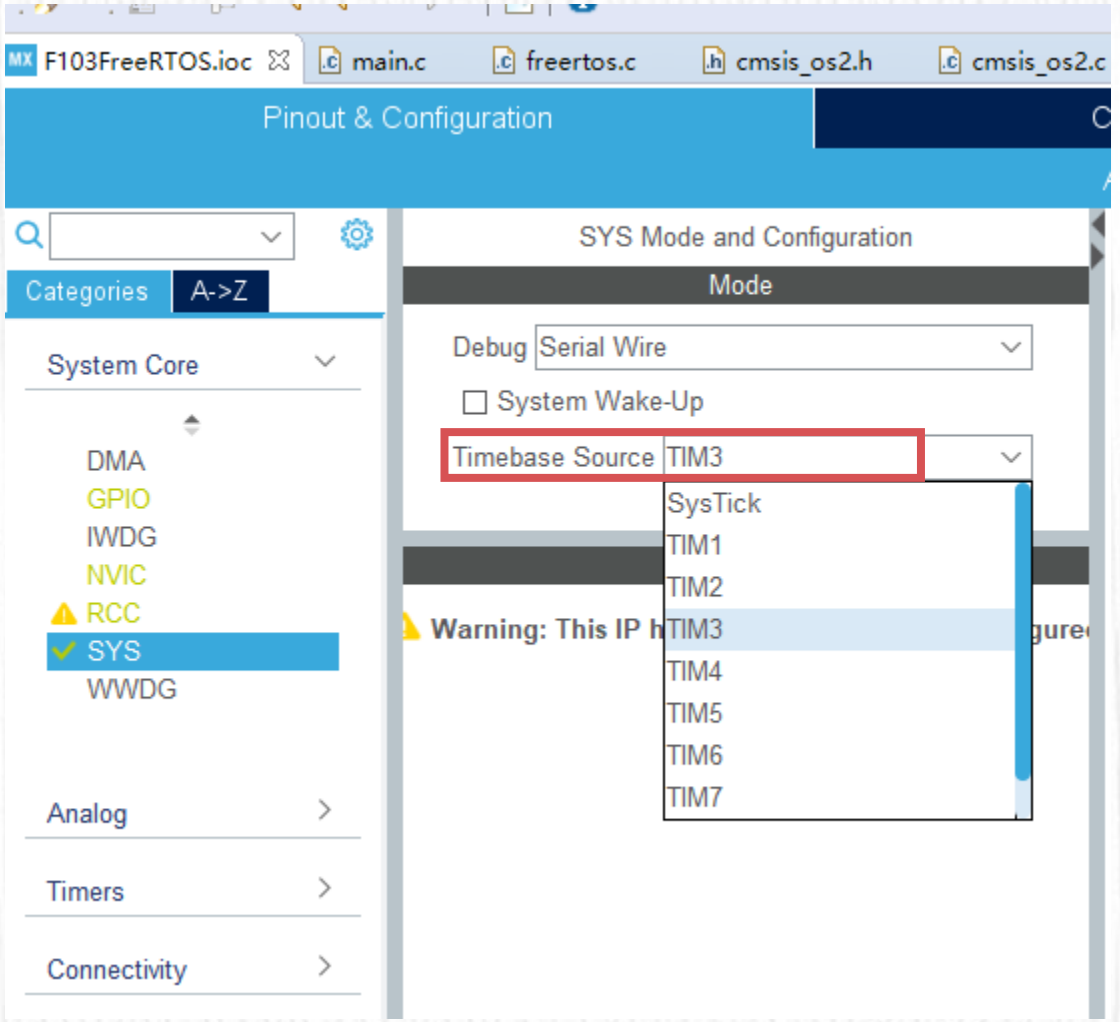
- Create 2 tasks
  - Go to the Tasks and Queues, and add two tasks



3. How to Program

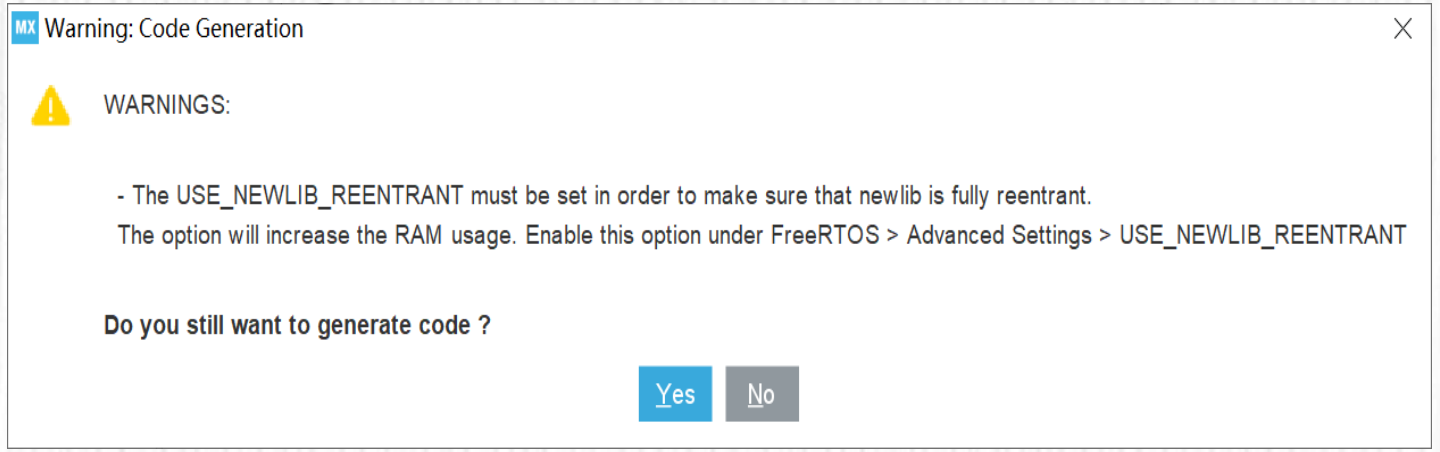
- Warning 1
  - The HAL has its own timebase which uses for timeout detection. It originally uses Systick but Systick is occupied by FreeRTOS. So, we need to change the timebase source to a timer, for example, TIM3

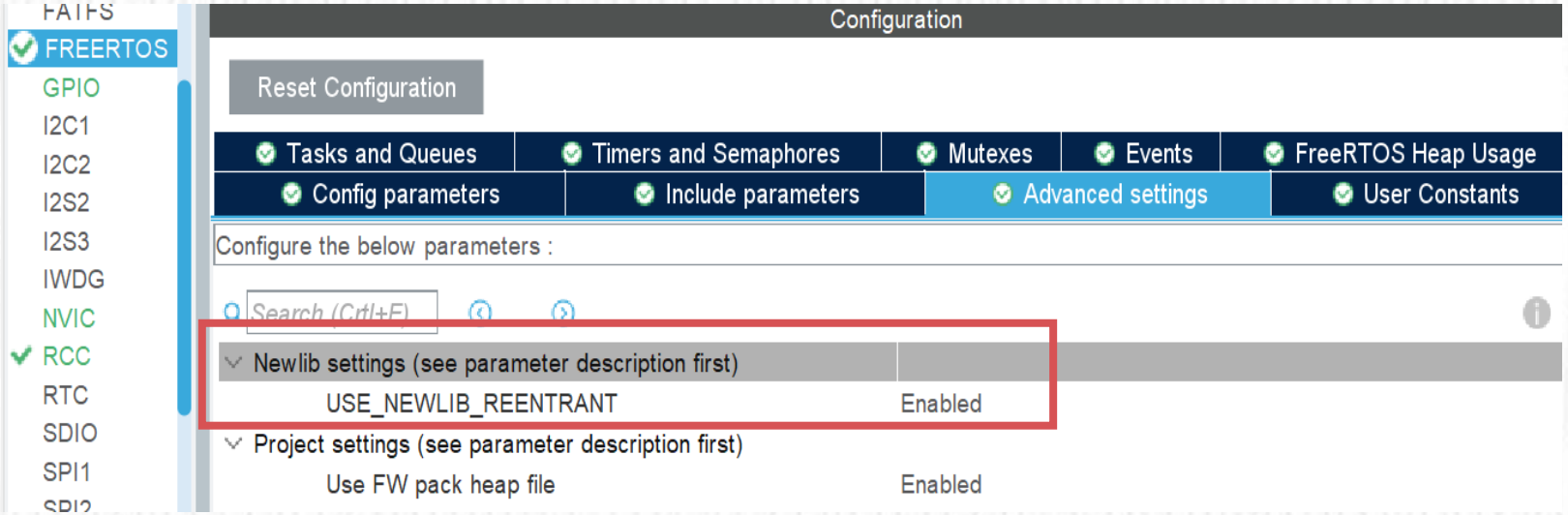




3. How to Program

- Warning 2





3. How to Program

- In the main routine of the main.c file, we find osThreadCreate() and osKernelStart()
- osThreadCreate() creates a task thread
- osKernelStart() starts the task-scheduler

```
/* Create the thread(s) */
/* definition and creation of Task_LED0 */
osThreadDef(Task_LED0, Func_LED0, osPriorityNormal, 0, 128);
Task_LED0Handle = osThreadCreate(osThread(Task_LED0), NULL);

/* definition and creation of Task_LED1 */
osThreadDef(Task_LED1, Func_LED1, osPriorityIdle, 0, 128);
Task_LED1Handle = osThreadCreate(osThread(Task_LED1), NULL);

/* USER CODE BEGIN RTOS_THREADS */
/* add threads, ... */
/* USER CODE END RTOS_THREADS */

/* Start scheduler */
osKernelStart();
```

3. How to Program

- Find Func_LED0 and Func_LED1 in main.c and implement them

```
void Func_LED0(void const * argument)
{
    /* USER CODE BEGIN Func_LED0 */
    /* Infinite loop */
    for(;;)
    {
      HAL_GPIO_TogglePin(LED0_GPIO_Port, LED0_Pin);
      osDelay(500);
    }
  /* USER CODE END Func_LED0 */
}
```

```
void Func_LED1(void const * argument)
{
    /* USER CODE BEGIN Func_LED1 */
    /* Infinite loop */
    for(;;)
    {
      HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
      osDelay(500);
    }
  /* USER CODE END Func_LED1 */
}
```

- osDelay is used to block the task, and execute the task in 500 milliseconds. It will suspend the current task, and CPU can execute other task when the pervious task is blocked

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

04

Practice

4. Practice

- Run the FREERTOS demo on MiniSTM32 board
- Complete assignment 2 on Sakai site and submit the whole project before DDL (30st, Nov)