南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

# Embedded System and Microcomputer Principle

## LAB13 FreeRTOS Semaphore

2022 Fall
wangq9@mail.sustech.edu.cn

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

# CONTENTS

01

# FreeRTOS Semaphore Description

# 1. FreeRTOS Semaphore Description

## -- What is Semaphore

- 信号量是由Dijkstra在1965年提出的，这是一种非常重要的技术，通过使用一个简单的整数值（称为信号量）来管理并发进程。
- 信号量的主要目的有两个
  - 共享资源访问
  - 任务同步
- FreeRTOS中信号量分为4种
  - 二值信号量
  - 计数型信号量
  - 互斥信号量
  - 递归互斥信号量

死锁

多个进程访问共享资源

# 1. FreeRTOS Semaphore Description -- P and V operations

- 信号量具有两种操作：P和V
- 操作V增加信号量的值
- 操作P减少信号量的值
- 当执行操作P但信号量值为零时，执行该操作的任务将被阻塞，并等待信号量值大于零

# 1. FreeRTOS Semaphore Description -- Binary semaphores

- 二值信号量通常用于互斥访问或同步，二值信号量和互斥信号量非常类似，但是还是有一些细微的差别，互斥信号量拥有优先级继承机制，二值信号量没有优先级继承。
- 二值信号另更适合用于同步（任务与任务间同步或任务与中断的同步)
- 二值信号量其实就是一个只有一个队列项的队列，这个特殊的队列要么是满的，要么是空的，这正好就是二值的。

# 1. FreeRTOS Semaphore Description -- Counting semaphores



- 有些资料中也将计数型信号量叫做数值信号量，二值信号量相当于长度为 1 的队列，那么计数型信号量就是长度大于 1 的队列。同二值信号量一样，用户不需要关心队列中存储了什么数据，只需要关心队列是否为空即可。
- 计数型信号量通常用于如下两个场合
  - 事件计数
  - 资源管理

# 1. FreeRTOS Semaphore Description
## -- Semaphores/Mutexes API

- xSemaphoreCreateBinary()
- xSemaphoreCreateBinaryStatic()
- vSemaphoreCreateBinary()
- xSemaphoreCreateCounting()
- xSemaphoreCreateCountingStatic()
- xSemaphoreCreateMutex()
- xSemaphoreCreateMutexStatic()
- xSem'CreateRecursiveMutex()
- xSem'CreateRecursiveMutexStatic()
- vSemaphoreDelete()
- xSemaphoreGetMutexHolder()
- uxSemaphoreGetCount()
- xSemaphoreTake()
- xSemaphoreTakeFromISR()
- xSemaphoreTakeRecursive()
- xSemaphoreGive()
- xSemaphoreGiveRecursive()
- xSemaphoreGiveFromISR()

# 1. FreeRTOS Semaphore Description -- API for binary semaphores



| 创建二值信号量函数 | |
|---|---|
| vSemaphoreCreateBinary() | 动态创建二值信号量，老版本FreeRTOS中使用 |
| xSemaphoreCreateBinary() | 动态创建二值信号量，新版本FreeRTOS中使用 |
| xSemaphoreCreateBinaryStatic() | 静态创建二值信号量 |
| **释放信号量函数** | |
| xSemaphoreGive() | 任务级释放信号量函数 |
| xSemaphoreGiveFromISR() | 中断级释放信号量函数 |
| **获取信号量函数** | |
| xSemaphoreTake() | 任务级获取信号量函数 |
| xSemaphoreTakeFromISR() | 中断级获取信号量函数 |

# 1. FreeRTOS Semaphore Description -- API for counting semaphores

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

| 创建计数型信号量函数 | |
|---|---|
| xSemaphoreCreateCounting() | 使用动态方法创建计数型信号量 |
| xSemaphoreCreateCountingStatic() | 使用静态方法创建计数型信号量 |

| 释放信号量函数(同二值信号量) | |
|---|---|
| xSemaphoreGive() | 任务级释放信号量函数 |
| xSemaphoreGiveFromISR() | 中断级释放信号量函数 |

| 获取信号量函数(同二值信号量) | |
|---|---|
| xSemaphoreTake() | 任务级获取信号量函数 |
| xSemaphoreTakeFromISR() | 中断级获取信号量函数 |

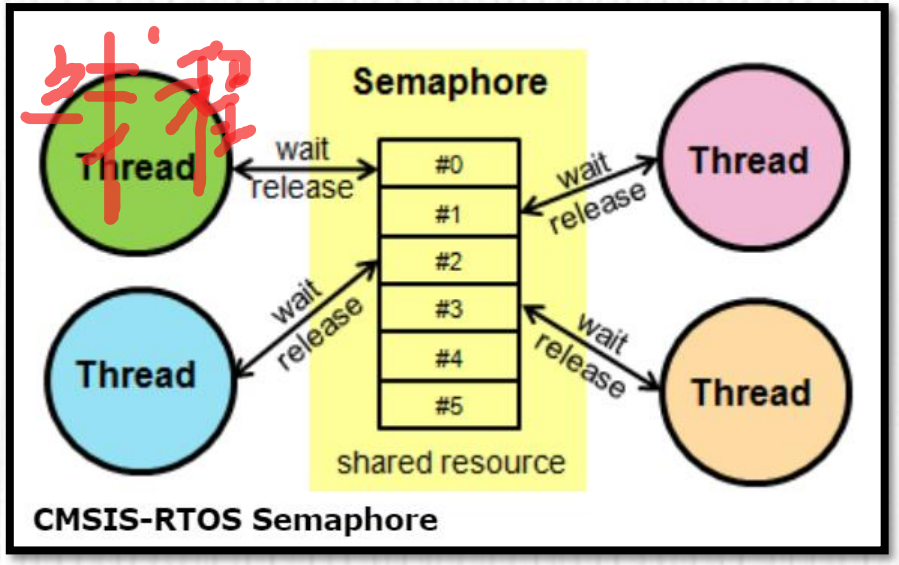
02

# CMSIS-RTOS Semaphore API

# 2. CMSIS-RTOS Semaphore API

## -- CMSIS-RTOS semaphore description

- A semaphore object should be initialized to the maximum number of available tokens. This number of available resources is specified as parameter of the osSemaphoreCreate function.
- Each time a semaphore token is obtained with osSemaphoreWait, the semaphore count is decremented.




- When the semaphore count is 0, no semaphore token can be obtained.
- The thread that tries to obtain the semaphore token needs to wait until the next token is free. Semaphores are released with osSemaphoreRelease function incrementing the semaphore count.
- Semaphore tokens can be acquired from threads and released from threads and ISRs.

# 2. CMSIS-RTOS Semaphore API
## -- CMSIS-RTOS semaphore macros



△宏定义

**#define osFeature_Semaphore 30**

A CMSIS-RTOS implementation may support semaphores. The value osFeature_Semaphore indicates the maximum index count for a semaphore.

**#define osSemaphore (name) &os_semaphore_def_##name**

Access to semaphore object for the functions osSemaphoreCreate.

**name** parameter: name of the semaphore object.

**#define osSemaphoreDef (name) const osSemaphoreDef_t os_semaphore_def_##name = { 0 }**

Define a semaphore object that is referenced by osSemaphore.

**name** parameter: name of the semaphore object.

# 2. CMSIS-RTOS Semaphore API
# -- CMSIS-RTOS semaphore functions

| **osSemaphoreId osSemaphoreCreate (const osSemaphoreDef_t *semaphore_def, int32_t count)** |
|---|
| Create and Initialize a Semaphore object used for managing resources. |
| **semaphore_def** parameter: semaphore definition referenced with osSemaphore.<br>**count** parameter: number of available resources.<br>returns: semaphore ID for reference by other functions or NULL in case of error. |
| **osStatus osSemaphoreDelete (osSemaphoreId semaphore_id)** |
| Delete a Semaphore that was created by osSemaphoreCreate. |
| **semaphore_id** parameter: semaphore object referenced with osSemaphoreCreate.<br>returns: status code that indicates the execution status of the function. |
| **osStatus osSemaphoreRelease (osSemaphoreId semaphore_id)** |
| Release a Semaphore token. |
| **int32_t osSemaphoreWait (osSemaphoreId semaphore_id, uint32_t millisec)** |
| Wait until a Semaphore token becomes available. |
| **millisec** parameter: Timout Value or 0 in case of no time-out.<br>returns: number of available tokens, or -1 in case of incorrect parameters. |

# 03

# How to Program

# 3. How to Program

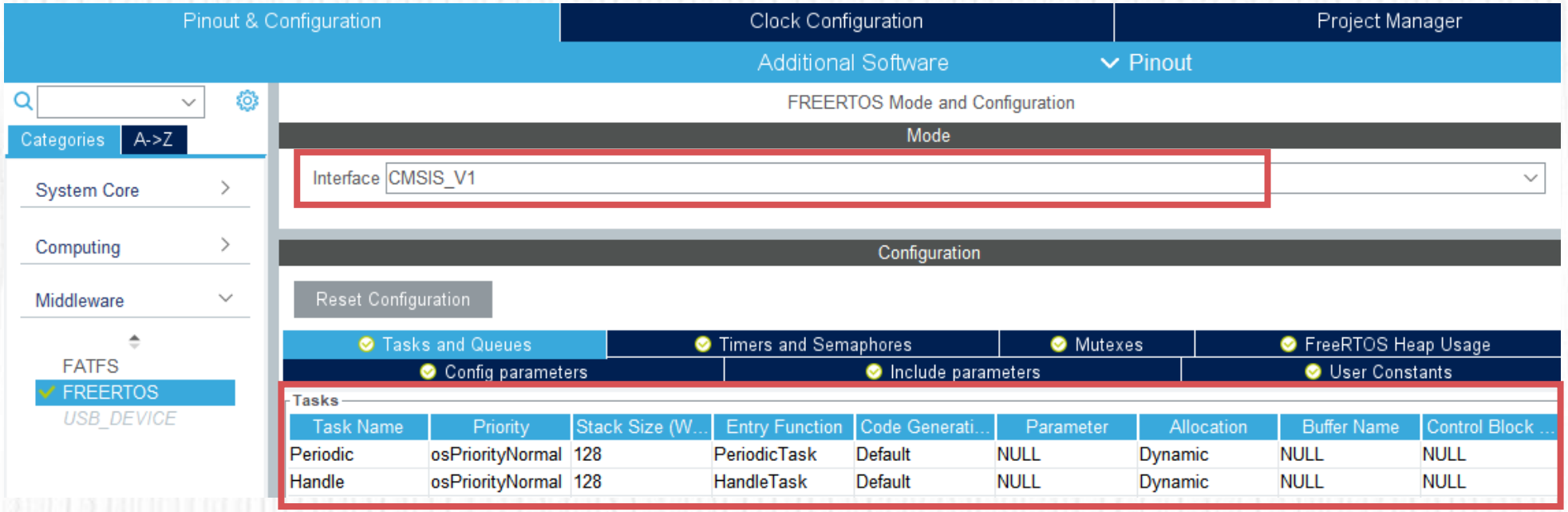
- Our Goal
  - Use binary semaphores to control two tasks.
  - Use counting semaphores to control two tasks.

# 3. How to Program

- Configure SYS
  - Remember to configure the basetime source
- Configure RCC
- Configure GPIO

# 3. How to Program

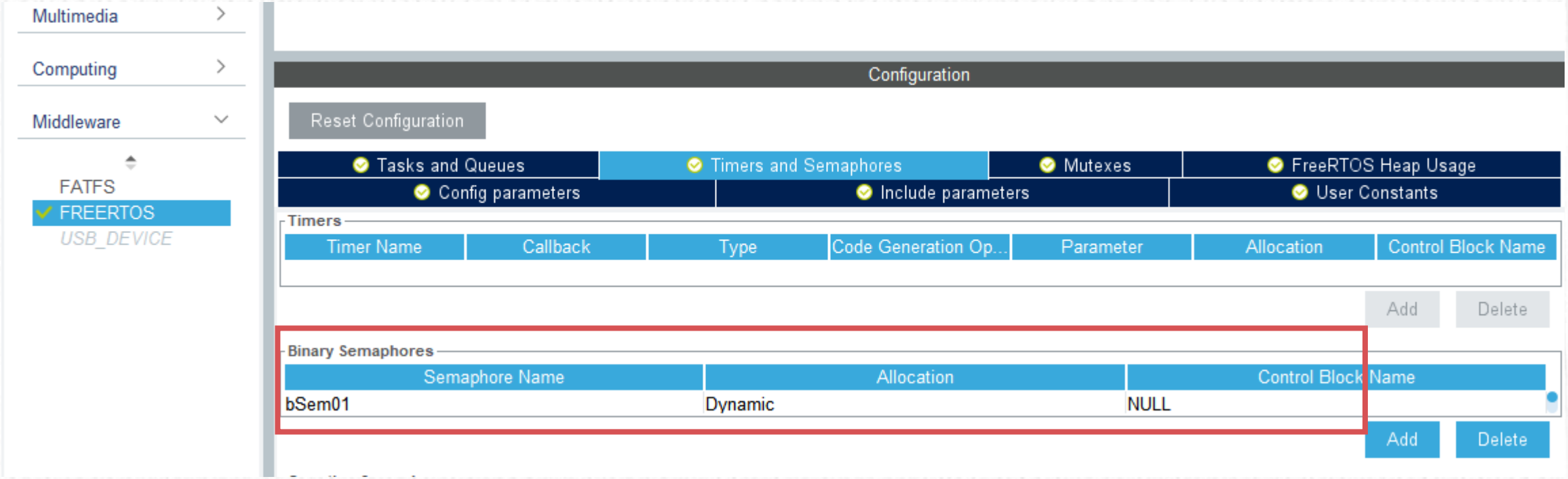
- Configure FREERTOS
  - Set the FreeRTOS API as CMSIS_v1
  - Create two tasks with the name **Periodic** and **Handle**



# 3. How to Program

## -- Binary semaphores

- Go to the Timers and Semaphores tab and add a **binary semaphore** named **bSem01.**



# 3. How to Program



- Generate the code and implement the function PeriodicTask and HandleTask.

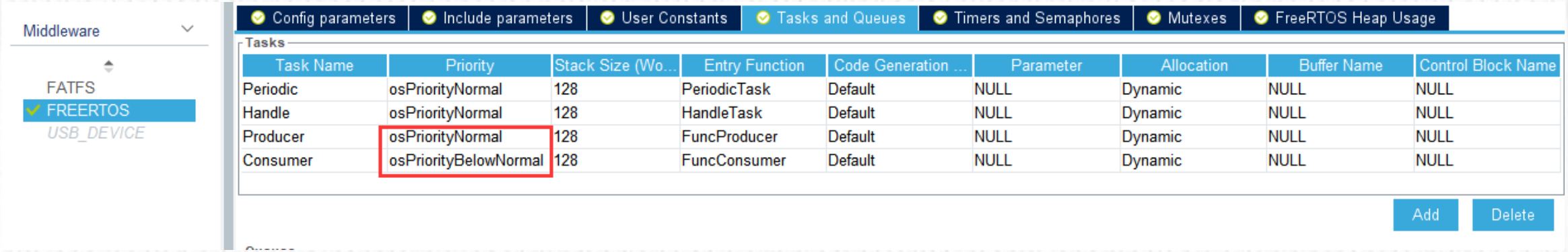
```
void PeriodicTask(void const * argument)
{
  /* USER CODE BEGIN PeriodicTask */
  /* Infinite loop */
  for(;;)
  {
    osDelay(1000);
    osSemaphoreRelease(bSem01Handle); //V operation, increase the semaphore count
  }    /* USER CODE END PeriodicTask */
}
```

```
void HandleTask(void const * argument)
{
  /* USER CODE BEGIN HandleTask */
  /* Infinite loop */
  for(;;)
  {
    osSemaphoreWait(bSem01Handle, osWaitForever); //P operation, decrease the semaphore count
    HAL_GPIO_TogglePin(LED0_GPIO_Port, LED0_Pin);
    HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
  }    /* USER CODE END HandleTask */
}
```

- Since **bSem01** is a binary semaphore, osSemaphoreWait can be executed once and the Handle task will be blocked because the value of semaphore is zero. It won't wake up until Periodic task does operation V. The LED0 and LED1 will blink in different period of time if the delay time changes in **PeriodicTask**().

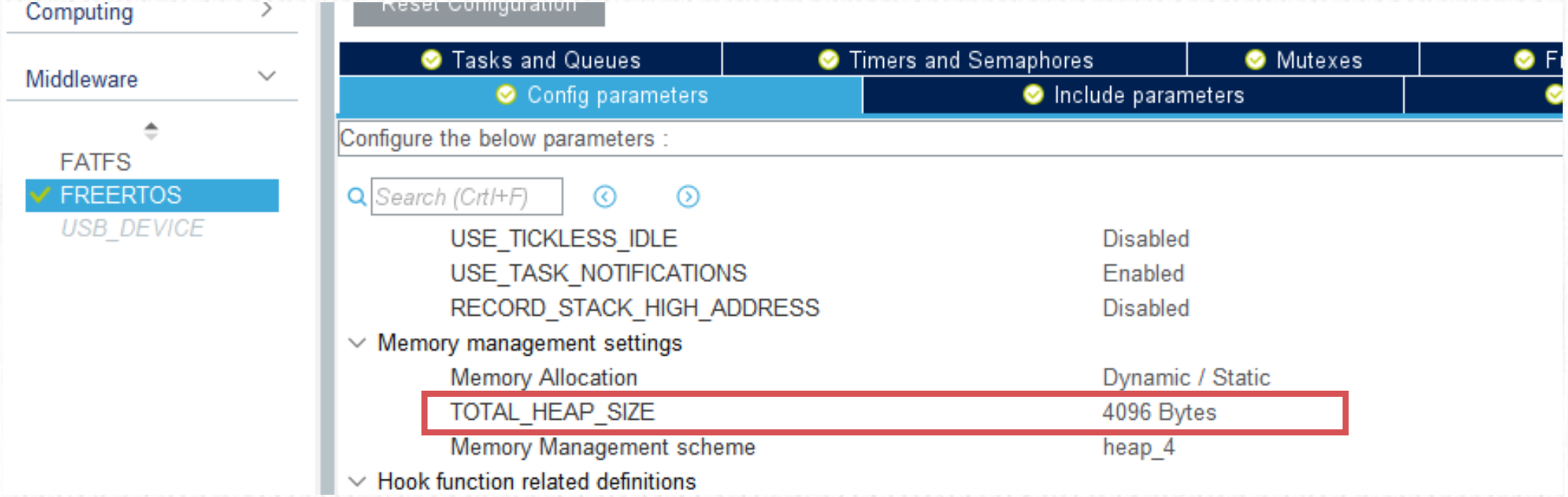
# 3. How to Program

- Use **binary semaphores** to solve simple producer-consumer problem.
  - Producer-consumer problem is a classical multi-process synchronization problem in operation system. If the buffer size is **one**, we can use binary semaphore to solve this problem.
  - Back to configuration and add two more tasks.
  - Remember to set the priority of the Consumer task as **osPriorityBelowNormal**, which will make sure the Producer task will be executed before the Consumer task.



# 3. How to Program

- We have added two more tasks but too small available heap size will cause unexpected problems. In case of this situation, we can go to the Config parameters tab to increase the TOTAL_HEAP_SIZE to 4096 Bytes.



# 3. How to Program

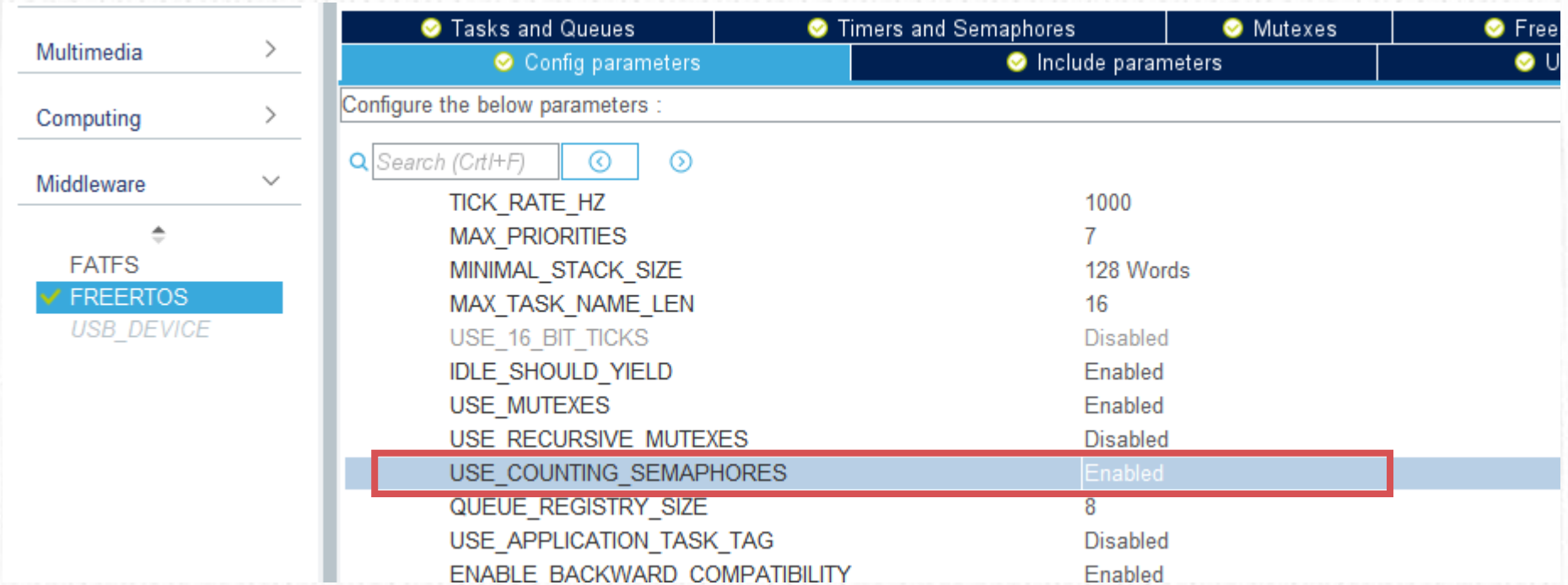
- Add two more semaphores named **bSemEmpty** and **bSemFilled**(binary semaphores) and generate the code.
- Find FuncProducer and FuncConsumer and implement them.

```
void FuncProducer(void const * argument)
{
  /* USER CODE BEGIN FuncProducer */
  /* Infinite loop */
  for(;;)
  {
    osSemaphoreWait(bSemEmptyHandle, osWaitForever);
    sprintf(msg, "Producer produce data\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)msg,
strlen(msg), HAL_MAX_DELAY);
    HAL_Delay(500);
    osSemaphoreRelease(bSemFilledHandle);
  }
/* USER CODE END FuncProducer */
}
```

```
void FuncConsumer(void const * argument)
{
  /* USER CODE BEGIN FuncConsumer */
  /* Infinite loop */
  for(;;)
  {
    osSemaphoreWait(bSemFilledHandle, osWaitForever);
    sprintf(msg, "Consumer consume data\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)msg,
strlen(msg), HAL_MAX_DELAY);
    HAL_Delay(500);
    osSemaphoreRelease(bSemEmptyHandle);
  }
/* USER CODE END FuncConsumer */
}
```

# 3. How to Program
## -- Counting semaphores

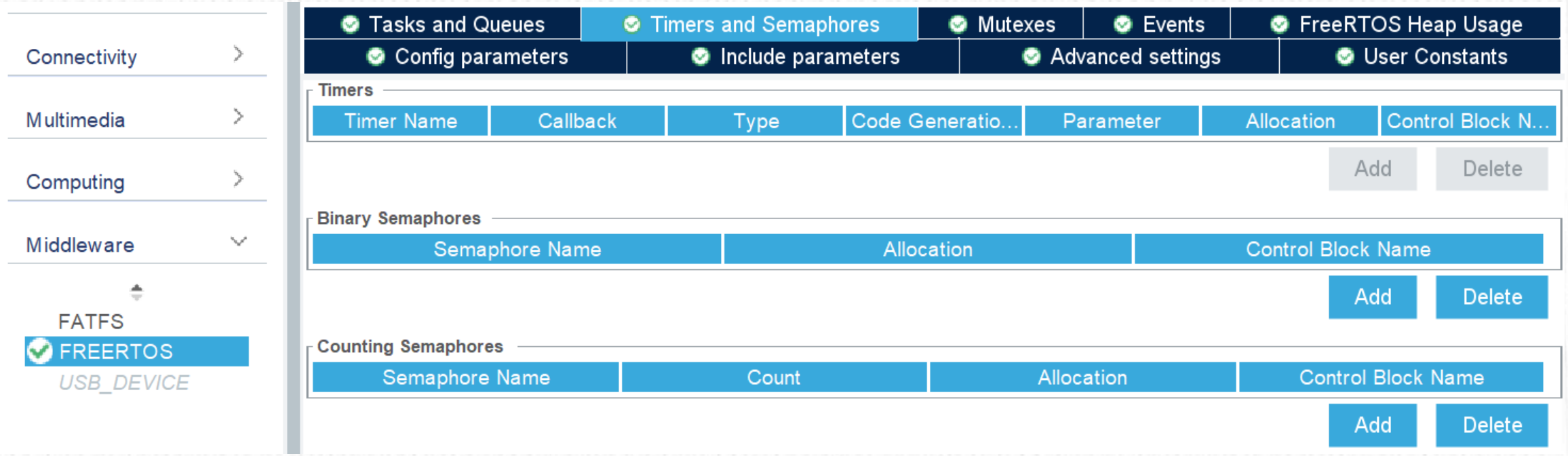
- To use counting semaphore, enable it in the Config parameters tab.



# 3. How to Program

## -- Counting semaphores

- If we want to add counting semaphores, we should do this in the Timers and Semaphores tab.



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

# 04

# Practice

# 4. Practice



- Suppose the buffer size of the Producer-Consumer problem is 4. Try to use counting semaphore to solve it.
- Remember to set the priority of the Consumer task as osPriorityBelowNormal to make sure the Producer task executed first.
- Try to figure out what the output should be and why before running the program, and compare it with the actual output.