

STM32Cube高效开发教程（高级篇）

第5章 信号量

王维波

中国石油大学（华东）控制科学与工程学院

STM32Cube高效开发教程（高级篇）

作者：王维波，鄢志丹，王钊

人民邮电出版社

2022年2月出版

如果有读者需要本书课件的PPT版本用于备课，可以给作者发邮件免费获取，并可加入专门的教学和技术交流QQ群

邮箱：wangwb@upc.edu.cn



5.1 信号量和互斥量概述

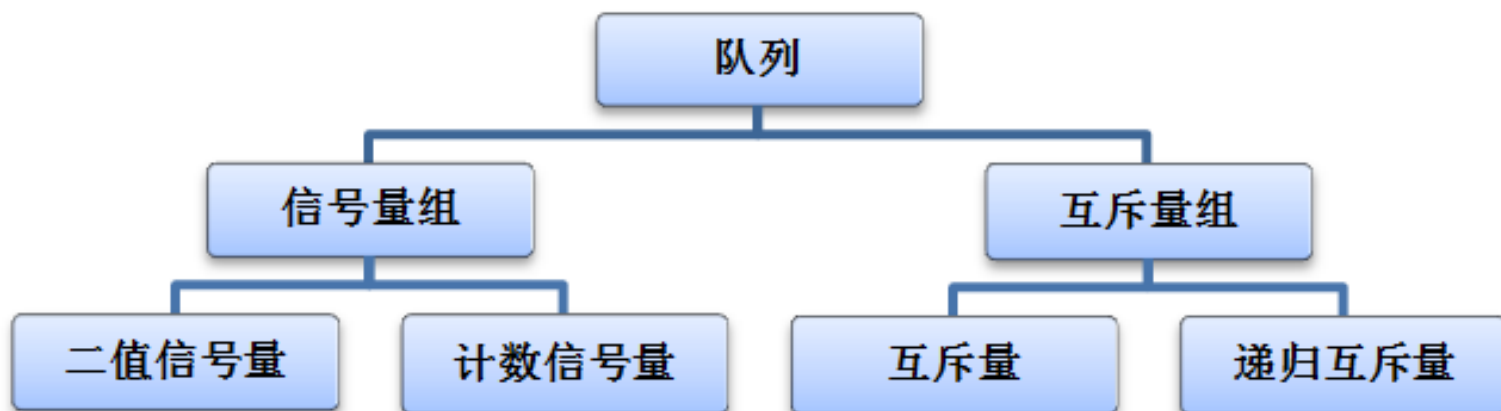
5.2 二值信号量使用示例

5.3 计数信号量使用示例

信号量（Semaphore）和互斥量（Mutex）都基于消息队列的基本数据结构，但是信号量和互斥量又有一些区别。

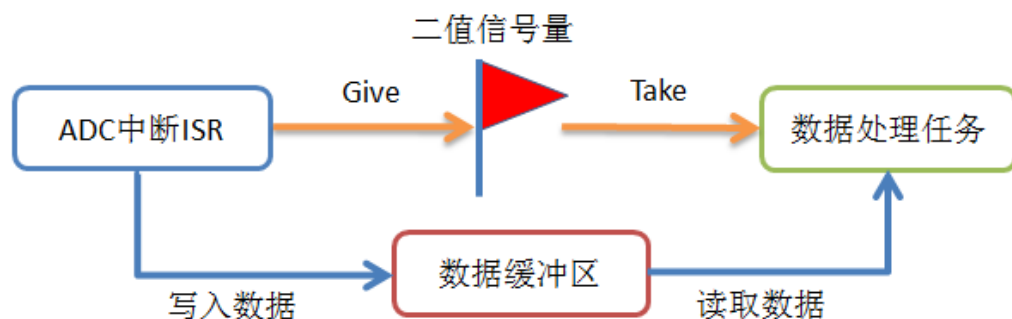
信号量没有优先级继承机制，使用二值信号量时容易出现优先级翻转问题，而二值信号量可以减缓优先级翻转问题。

二值信号量适用于进程间同步，计数信号量适用于多个共享资源的访问控制，互斥量适用于对一个资源的互斥访问控制。



5.1.1 二值信号量

二值信号量（Binary Semaphore）就是只有一个项的队列。
二值信号量就像是一个标志，适合用于进程间同步的通信。



- ADC中断ISR读取ADC转换结果后写入数据缓冲区，并且释放（Give）二值信号量。
- 数据处理任务总是获取（Take）二值信号量。无效时，任务在阻塞状态等待；有效后，任务立刻进入就绪状态参与任务调度，就可以读取缓冲区的数据并进行处理。

5.1.2 计数型信号量

计数型信号量（Counting Semaphore）就是有固定长度的队列，每个项是一个标志。计数型信号量通常用于多个共享资源的访问控制

- 一个计数型信号量被创建时设置了初值4，这个值只是个计数值。

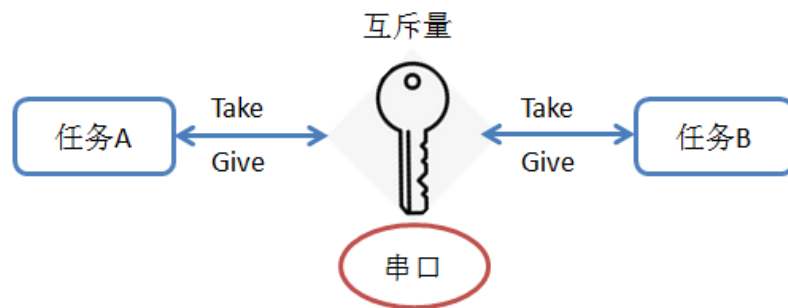


- 有1个客人进店时就是获取（Take）信号量，计数信号量的值减1。当计数信号量的值变为0时，再有客人要进店时就得等待。
- 如果有1个客人用餐结束离开了就是释放（Give）信号量，计数信号量的值加1，表示可用资源数量增加了1个。

5.1.3 互斥量

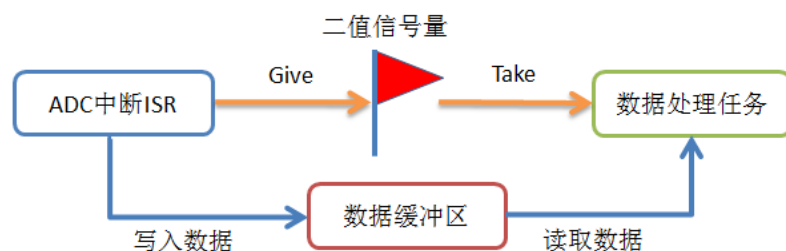
使用二值信号量时可能会出现优先级翻转的问题。互斥量引入了优先级继承机制，可以减缓优先级翻转问题。

- 两个任务互斥性地访问串口，即在任务A访问串口时，其他任务不能访问串口。

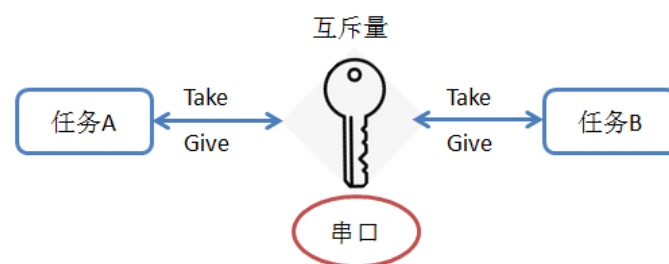


- 互斥量相当于管理串口的一把钥匙。一个任务可以获取（Take）互斥量，获得互斥量后将独占对串口的访问，访问完后要释放（Give）互斥量。
- 一个任务获得互斥量后，对资源进行访问时，其他想要获取互斥量的进程只能等待。

进程间同步



共享资源的互斥访问



- **进程间同步**：一个进程只负责释放信号量，另一个进程只负责获取信号量
- **共享资源互斥访问**：一个任务对互斥量既有获取操作，也有释放操作

注意，**互斥量不能在ISR函数中使用**，因为互斥量具有任务的优先级继承机制，而ISR不是任务。另外，ISR函数中不能设置阻塞等待时间，而获取互斥量时经常是需要等待的。

5.1.4 递归互斥量

递归互斥量（Recursive Mutex）是一种特殊的互斥量，可以用于需要递归调用的函数中。

一个任务在获得了互斥量之后就不能再获得互斥量，而一个任务获得递归互斥量之后，可以再次获得此递归互斥量，当然每次获取必须与一次释放配对使用。

递归互斥量同样不能在ISR函数中使用。

5.1.5 相关函数概述

1. 对象的创建与删除

每一种对象的创建都可以动态分配内存，或静态分配内存

函数名	功能描述
xSemaphoreCreateBinary()	创建二值信号量
xSemaphoreCreateBinaryStatic()	创建二值信号量，静态分配内存
xSemaphoreCreateCounting()	创建计数型信号量
xSemaphoreCreateCountingStatic()	创建计数型信号量，静态分配内存
xSemaphoreCreateMutex()	创建互斥量
xSemaphoreCreateMutexStatic()	创建互斥量，静态分配内存
xSemaphoreCreateRecursiveMutex()	创建递归互斥量
xSemaphoreCreateRecursiveMutexStatic()	创建递归互斥量，静态分配内存
vSemaphoreDelete()	删除这4种信号量或互斥量

2. 获取与释放

函数名	功能描述
xSemaphoreGive()	释放二值信号量、计数型信号量、互斥量
xSemaphoreGiveFromISR()	xSemaphoreGive()的ISR版本，但不能用于互斥量
xSemaphoreGiveRecursive()	释放递归互斥量
xSemaphoreTake()	获取二值信号量、计数型信号量、互斥量
xSemaphoreTakeFromISR()	xSemaphoreTake()的ISR版本，但不用于互斥量
xSemaphoreTakeRecursive()	获取递归互斥量

注意，互斥量不能在ISR函数里使用，递归互斥量有专门的函数

3. 其他操作

函数名	功能描述
uxSemaphoreGetCount()	返回计数型信号量或二值信号量当前的值
xSemaphoreGetMutexHolder()	返回互斥量的当前保持者（holder）
xSemaphoreGetMutexHolderFromISR()	xSemaphoreGetMutexHolder()的ISR版本

uxSemaphoreGetCount(xSemaphore)返回信号量xSemaphore的当前值，xSemaphore可以是计数型信号量或二值信号量

- 如果是二值信号量，返回的值是1（信号量有效）或0（信号量无效）
- 如果是计数型信号量，返回值就是计数型信号量当前的值，也就是剩余可用资源个数

要在FreeRTOS中使用计数信号量、互斥量、递归互斥量，需要设置相应的config参数为1。

这几个参数在CubeMX里可以设置，且默认都是Enabled。

▼ Kernel settings	
USE_PREEMPTION	Enabled
CPU_CLOCK_HZ	SystemCoreClock
TICK_RATE_HZ	1000
MAX_PRIORITIES	56
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	Enabled
USE_COUNTING_SEMAPHORES	Enabled

5.2 二值信号量使用示例

5.2.1 二值信号量操作相关函数详解

5.2.2 示例功能和CubeMX项目设置

5.2.3 程序功能实现

5.2.1 二值信号量操作相关函数详解

1. 创建二值信号量

`xSemaphoreCreateBinary()`以动态分配内存方式创建二值信号量，调用函数`xQueueGenericCreate()`

```
#define xSemaphoreCreateBinary() xQueueGenericCreate( ( UBaseType_t ) 1,  
    semSEMAPHORE_QUEUE_ITEM_LENGTH,  
    queueQUEUE_TYPE_BINARY_SEMAPHORE )
```

- 第1个参数：数值1，是队列长度
- 第2个参数：`semSEMAPHORE_QUEUE_ITEM_LENGTH`，其值实际为0，表示数据类型
- 第3个参数：`queueQUEUE_TYPE_BINARY_SEMAPHORE`，表示创建的是二值信号量

2. 释放二值信号量

二值信号量被创建后是无效的，相当于值为0。释放二值信号量就是使其有效，相当于使其变为1。

```
#define xSemaphoreGive( xSemaphore )  
    xQueueGenericSend( ( QueueHandle_t ) ( xSemaphore ), NULL,  
        semGIVE_BLOCK_TIME, queueSEND_TO_BACK )
```

- 第1个参数：xSemaphore，是二值信号量的句柄
- 第2个参数：数值NULL。这个参数是需要向队列写入的数据，对于二值信号量来说不需要写数据到队列
- 第3个参数：常量semGIVE_BLOCK_TIME，其数值为0，释放二值信号量无需等待，所以数值为0
- 第4个参数：常量queueSEND_TO_BACK，这个参数是写入队列的方向

在ISR函数中释放信号量使用xSemaphoreGiveFromISR()

```
#define xSemaphoreGiveFromISR( xSemaphore, pxHigherPriorityTaskWoken )  
    xQueueGiveFromISR( ( QueueHandle_t ) ( xSemaphore ),  
                        ( pxHigherPriorityTaskWoken ) )
```

它调用了函数xQueueGiveFromISR(), 原型定义是:

```
BaseType_t xQueueGiveFromISR( QueueHandle_t xQueue,  
                             BaseType_t * const pxHigherPriorityTaskWoken )
```

参数pxHigherPriorityTaskWoken是指针类型，是一个返回数据，返回值为pdTRUE或pdFALSE。如果释放信号量导致一个任务解锁，而解锁的任务比当前任务优先级高，则返回值为pdTRUE，就需要在退出ISR函数之前申请进行任务调度，以便解锁的高优先级任务可以及时执行。

在ISR中调用xSemaphoreGiveFromISR()的示意代码如下：

```
BaseType_t highTaskWoken = pdFALSE;  
if (BinSem_DataReadyHandle != NULL)  
{  
    xSemaphoreGiveFromISR(BinSem_DataReadyHandle, &highTaskWoken);  
    portYIELD_FROM_ISR(highTaskWoken); //申请进行一次任务调度  
}
```

函数portYIELD_FROM_ISR()用于申请进行任务调度。

3. 获取二值信号量

在任务中获取二值信号量的函数是xSemaphoreTake()

```
#define xSemaphoreTake( xSemaphore, xBlockTime )  
    xQueueSemaphoreTake( ( xSemaphore ), ( xBlockTime ) )
```

- 参数xSemaphore，是二值信号量的句柄
- 参数xBlockTime，是阻塞等待的节拍数。如果二值信号量无效，可以设置一个超时等待时间。常数portMAX_DELAY表示一直等待，0表示不等待

如果成功获得了二值信号量，函数xSemaphoreTake()返回pdTRUE，否则返回pdFALSE。

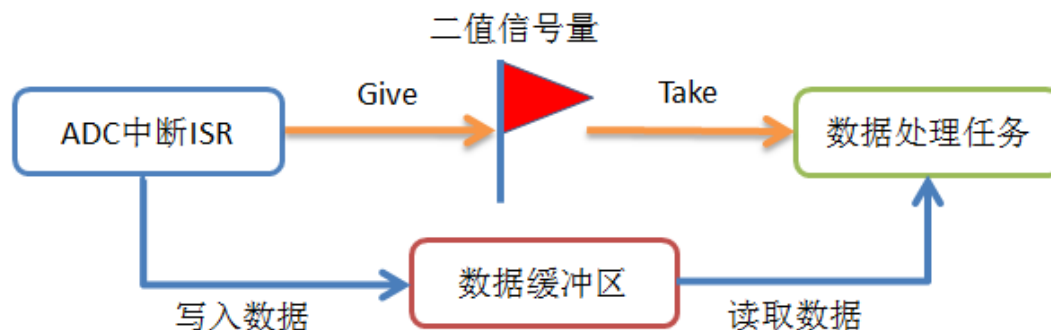
在ISR函数中获取二值信号量使用xSemaphoreTakeFromISR()

```
#define xSemaphoreTakeFromISR( xSemaphore, pxHigherPriorityTaskWoken )  
    xQueueReceiveFromISR( ( QueueHandle_t ) ( xSemaphore ), NULL,  
        ( pxHigherPriorityTaskWoken ) )
```

- 参数xSemaphore，是二值信号量或计数型信号量的句柄，不能是互斥量。
- 参数pxHigherPriorityTaskWoken是传递指针的返回数据，返回值为pdTRUE或pdFALSE，表示是否需要在退出ISR函数之前进行任务调度申请。

5.2.2 示例功能和CubeMX项目设置

示例Demo5_1BinarySemaphore，演示二值信号量的使用



- 创建一个二值信号量BinSem_DataReady
- ADC1在定时器TIM3的触发下进行周期为500ms的ADC数据采集，在ADC的ISR里将转换结果写入缓存变量，并释放信号量BinSem_DataReady
- 一个任务总是尝试获取信号量BinSem_DataReady。在获得信号量后，读取ADC转换结果缓存变量，然后在LCD上显示数据

(1) 定时器TIM3的设置

设置时钟树

- 使HCLK为100MHz
- APB1和APB2定时器时钟频率都是50MHz

设置TIM3

- 设置定时器时钟源为Internal Clock
- 使TIM3的更新周期为500ms
- 以更新事件作为TRGO信号, TIM3的TRGO信号可以作为ADC1的外部触发信号

TIM3 Mode and Configuration

Mode	
Slave Mode	Disable
Trigger Source	Disable
Clock Source	Internal Clock
Channel1	Disable

Configuration	
Reset Configuration	
Parameter Settings User Constants NVIC Settings DMA Settings	
Counter Settings	
Prescaler (PSC - 16 bits value)	49999
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	499
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable
Trigger Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection	Update Event

(2) ADC1的设置

✓ ADCs_Common_Settings	
Mode	Independent mode
✓ ADC_Settings	
Clock Prescaler	PCLK2 divided by 2
Resolution	12 bits (15 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	Disabled
Continuous Conversion Mode	Disabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Disabled
End Of Conversion Selection	EOC flag at the end of single channel conversion
✓ ADC_Regular_ConversionMode	
Number Of Conversion	1
External Trigger Conversion Source	Timer 3 Trigger Out event
External Trigger Conversion Edge	Trigger detection on the rising edge
✓ Rank	1
Channel	Channel 5
Sampling Time	15 Cycles

- ADC1的输入通道只选择IN5，使用12bits精度，靠右对齐
- 外部触发源选择Timer 3 Trigger Out event
- 在ADC1的NVIC Settings启用ADC1全局中断

(3) FreeRTOS的设置

创建一个任务Task_Show,
用于ADC转换结果的显示。

Task Name	Task_Show
Priority	osPriorityNormal
Stack Size (Words)	128
Entry Function	AppTask_Show
Code Generation Option	Default
Parameter	NULL
Allocation	Dynamic
Buffer Name	NULL
Control Block Name	NULL

OK Cancel

创建一个二值信号量
BinSem_DataReady,
使用动态分配内存方式

Timer Name	Callback	Type	Code Gener...	Parameter	Allocation	Control Blo...
------------	----------	------	---------------	-----------	------------	----------------

Add Delete

Semaphore Name	Allocation	Control Block Name
BinSem_DataReady	Dynamic	NULL

Add Delete

Semaphore Name	Count	Allocation	Control Block Name
----------------	-------	------------	--------------------

Add Delete

(4) NVIC的设置

- 无需启用TIM3的中断，只启用ADC1的中断
- 由于要在ADC1的中断ISR函数里调用FreeRTOS的函数
xSemaphoreGiveFromISR()，其抢占优先级不能高于5

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority	Uses FreeRTOS functions
Non maskable interrupt	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Hard fault interrupt	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Memory management fault	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Pre-fetch fault, memory access fault	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
System service call via SWI instruction	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Debug monitor	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Pendable request for system service	<input checked="" type="checkbox"/>	15	0	<input checked="" type="checkbox"/>
System tick timer	<input checked="" type="checkbox"/>	15	0	<input checked="" type="checkbox"/>
ADC1, ADC2 and ADC3 global interrupts	<input checked="" type="checkbox"/>	5	0	<input checked="" type="checkbox"/>
Time base: TIM6 global interrupt, DAC1 and ...	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>

5.2.3 程序功能实现

1. 主程序

```
int main(void)
{
    HAL_Init();                //HAL初始化
    SystemClock_Config();      //系统时钟初始化
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_FSMC_Init();
    MX_ADC1_Init();            //ADC初始化
    MX_TIM3_Init();            //TIM3初始化

    /* USER CODE BEGIN 2 */
    TFTLCD_Init();             //LCD 初始化
    LCD_ShowString(10, 10, (uint8_t *)"Demo5_1:Binary Semaphore");
    HAL_ADC_Start_IT(&hadc1);   //以中断方式启动ADC
    HAL_TIM_Base_Start(&htim3); //启动定时器
    /* USER CODE END 2 */

    osKernelInitialize();
    MX_FREERTOS_Init();
    osKernelStart();
    while (1)
    {
    }
}
```

2. FreeRTOS初始化与对象创建

文件freertos.c中的定义和初始化代码

```
/* 文件: freertos.c -----*/
#include "FreeRTOS.h"
#include "task.h"
#include "main.h"
#include "cmsis_os.h"

/* Private variables -----*/
/* 任务Task_Show相关定义 */
osThreadId_t Task_ShowHandle; //任务句柄变量
const osThreadAttr_t Task_Show_attributes = { //任务属性
    .name = "Task_Show",
    .priority = (osPriority_t) osPriorityNormal,
    .stack_size = 128 * 4
};

/* 二值信号量BinSem_DataReady相关定义 */
osSemaphoreId_t BinSem_DataReadyHandle; //信号量句柄变量
const osSemaphoreAttr_t BinSem_DataReady_attributes = { //信号量属性
    .name = "BinSem_DataReady"
};
```

创建任务和二值信号量

```
void MX_FREERTOS_Init(void)
{
    /* 创建信号量 BinSem_DataReady */
    BinSem_DataReadyHandle = osSemaphoreNew(1, 1,
        &BinSem_DataReady_attributes);

    /* 创建任务 Task_Show */
    Task_ShowHandle = osThreadNew(AppTask_Show, NULL,
        &Task_Show_attributes);
}
```

结构体osSemaphoreAttr_t的定义如下

```
typedef struct {  
    const char      *name; //信号量的名称  
    uint32_t        attr_bits; // 属性位  
    void            *cb_mem; //控制块的存储空间  
    uint32_t        cb_size; //控制块的存储空间大小，单位：字节  
} osSemaphoreAttr_t;
```

osSemaphoreNew()是CMSIS RTOS的接口函数 ()，这个函数不仅可以创建二值信号量，还可以创建计数信号量。

```
osSemaphoreId_t osSemaphoreNew (uint32_t max_count, uint32_t initial_count,  
    const osSemaphoreAttr_t *attr);
```

max_count是最多可用标志（token）个数，也就是队列存储项个数；initial_count是初始可用标志个数；attr是信号量的属性

3. ADC1的中断服务函数

ADC转换完成回调函数是HAL_ADC_ConvCpltCallback(),
在文件freertos.c中实现这个回调函数

```
/* USER CODE BEGIN Variables */
uint32_t adc_value;          //ADC转换原始数据
/* USER CODE END Variables */

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    if (hadc->Instance == ADC1)
    {
        adc_value=HAL_ADC_GetValue(hadc); //ADC转换原始数据
        BaseType_t highTaskWoken=pdFALSE;
        if (BinSem_DataReadyHandle != NULL)
        {
            xSemaphoreGiveFromISR(BinSem_DataReadyHandle, &highTaskWoken);
            portYIELD_FROM_ISR(highTaskWoken); //申请进行一次任务调度
        }
    }
}
```

4. 数据读取与显示任务函数

任务Task_Show总是获取二值信号量，当二值信号量变为有效时，表示有新的转换结果数据了

```
void AppTask_Show(void *argument)
{
    /* USER CODE BEGIN AppTask_Show */
    LCD_ShowString(10, 50, (uint8_t *)"ADC Value=");
    LCD_ShowString(10, 80, (uint8_t *)"Voltage(mV)=");
    for(;;)
    {
        if (xSemaphoreTake(BinSem_DataReadyHandle, portMAX_DELAY)==pdTRUE)
        {
            uint32_t tmpValue=adc_value;
            LCD_ShowUIntX(130,50,tmpValue,4); //显示ADC原始值
            uint32_t Volt=3300*tmpValue;      //mV
            Volt=Volt>>12; //除以2^12
            LCD_ShowUIntX(130,80,Volt,4);    //显示电压, mV
        }
    }
    /* USER CODE END AppTask_Show */
}
```

示例运行时的LCD显示

Demo5_1:Binary Semaphore

ADC Value = 2470

Voltage(mV)= 1989



5.3 计数信号量使用示例

5.3.1 计数信号量操作相关函数详解

5.3.2 示例功能和CubeMX项目设置

5.3.3 程序功能实现

5.3.1 计数信号量操作相关函数详解

1. 创建计数信号量

以动态分配内存方式创建计数信号量的函数

```
#define xSemaphoreCreateCounting( uxMaxCount, uxInitialCount )  
    xQueueCreateCountingSemaphore( ( uxMaxCount ), ( uxInitialCount ) )
```

- `uxMaxCount` 是计数信号量能达到的最大计数值
- `uxInitialCount` 是初始计数值。
- 函数返回数据类型是`QueueHandle_t`，返回值是所创建计数信号量的句柄

创建计数信号量时，一般使其初始值等于最大值，如

```
semb=xSemaphoreCreateCounting(5, 5)
```

2. 获取计数信号量

函数`xSemaphoreTake()`获取计数信号量，与获取二值信号量的函数就是同一个。

获取计数信号量就是申请一个资源，申请成功后计数信号量的计数值减1，表示可用资源减少一个。当计数信号量的计数值变为0时，表示没有资源再可被申请，再申请计数信号量的任务就需要等待。



3. 释放计数信号量

使用函数 `xSemaphoreGive()` 释放计数信号量，这个函数与释放二值信号量的函数就是同一个。

释放计数信号量就是释放一个资源，计数信号量的计数值会增加1，表示可用资源增加了1个。



4. 获取计数信号量当前计数值

可以使用函数uxSemaphoreGetCount()获取计数信号量当前的计数值，其定义是

```
#define uxSemaphoreGetCount( xSemaphore )  
    uxQueueMessagesWaiting( ( QueueHandle_t ) ( xSemaphore ) )
```

它就是调用了函数uxQueueMessagesWaiting()，而这个函数是查询队列中等待被读取消息条数的函数。

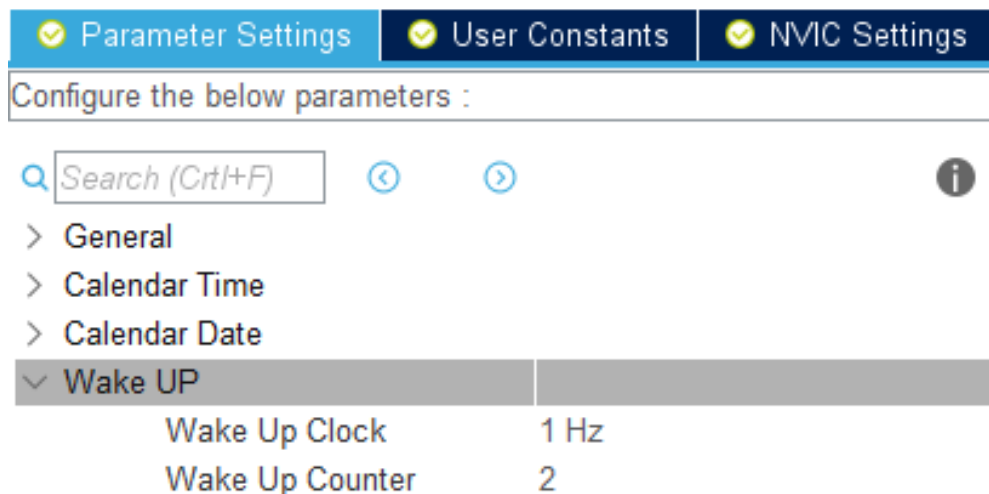
5.3.2 示例功能和CubeMX项目设置

演示计数信号量的使用，示例模拟下图的功能



- 创建一个计数信号量Sem_Tables，设置最大值为5，初始值为5，表示5张餐桌
- 设置RTC的唤醒周期为3秒。在RTC周期唤醒中断里用xSemaphoreGive()释放信号量，模拟有客人离开饭店
- 在一个任务里总是检测KeyRight的状态，按下按键时调用xSemaphoreTake()，模拟客人进店。

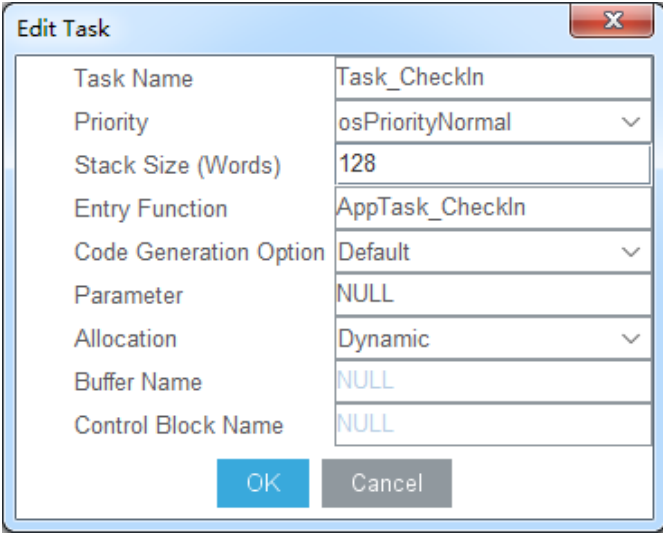
(1) RTC的设置



- 启用RTC，并将其WakeUp设置为Internal WakeUp。
- 周期唤醒使用的时钟频率是1Hz，每计数到2时唤醒一次，所以唤醒周期是3秒。
- 启用RTC的周期唤醒中断，将其中断优先级设置为5，因为要在其ISR函数中使用FreeRTOS API

(2) FreeRTOS的设置

创建一个任务Task_CheckIn



Task Name	Task_CheckIn
Priority	osPriorityNormal
Stack Size (Words)	128
Entry Function	AppTask_CheckIn
Code Generation Option	Default
Parameter	NULL
Allocation	Dynamic
Buffer Name	NULL
Control Block Name	NULL

OK Cancel

创建一个计数信号量Sem_Tables，采用动态分配内存方式创建计数信号量，Count表示最大计数值，也是初始的计数值。

Counting Semaphores			
Semaphore Name	Count	Allocation	Control Block Name
Sem_Tables	5	Dynamic	NULL

Add Delete

(3) 其他设置

与KeyRight连接的引脚PE2设置为GPIO_Input，设置为上拉

5.3.3 程序功能实现

1. 主程序

```
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    /* Initialize all configured peripherals */
    MX_GPIO_Init();          //GPIO初始化
    MX_FSMC_Init();
    MX_RTC_Init();           //RTC初始化

    /* USER CODE BEGIN 2 */
    TFTLCD_Init();           //LCD 初始化
    LCD_ShowString(10, 10, (uint8_t *)"Demo5_2:Counting Semaphore");
    /* USER CODE END 2 */

    osKernelInitialize();
    MX_FREERTOS_Init();
    osKernelStart();
    while (1)
    {
    }
}
```

2. FreeRTOS初始化与任务函数

变量定义与MX_FREERTOS_Init()函数

```
/* 任务Task_CheckIn相关定义 */
osThreadId_t Task_CheckInHandle;    //任务句柄
const osThreadAttr_t Task_CheckIn_attributes = { //任务属性
    .name = "Task_CheckIn",
    .priority = (osPriority_t) osPriorityNormal,
    .stack_size = 128 * 4
};

/* 计数信号量Sem_Tables相关定义 */
osSemaphoreId_t Sem_TablesHandle; //信号量句柄
const osSemaphoreAttr_t Sem_Tables_attributes = { //信号量属性
    .name = "Sem_Tables"
};

void MX_FREERTOS_Init(void)
{
    /* 创建计数信号量Sem_Tables */
    Sem_TablesHandle = osSemaphoreNew(5, 5, &Sem_Tables_attributes);

    /* 创建任务Task_CheckIn */
    Task_CheckInHandle = osThreadNew(AppTask_CheckIn, NULL,
    &Task_CheckIn_attributes);
}
```

```
Sem_TablesHandle = osSemaphoreNew(5, 5, &Sem_Tables_attributes);
```

osSemaphoreNew()根据设置的最大计数值确定创建二值信号量或计数信号量。这里设置最大计数值为5，初始计数值为5，所以创建的是计数信号量。

osSemaphoreNew()内部会根据信号量的属性设置自动调用xSemaphoreCreateCounting()或xSemaphoreCreateCountingStatic()

任务Task_CheckIn的任务函数

```
void AppTask_CheckIn(void *argument)
{
    UBaseType_t totalTables=uxSemaphoreGetCount(Sem_TablesHandle);    //当前计数值
    LCD_ShowStr(10, 40, (uint8_t *)"Total tables=");
    LCD_ShowUint(LCD_CurPosX, 40, totalTables);    //显示初始的计数值
    LCD_ShowStr(10, 70, (uint8_t *)"Available tables=");    //动态过程中的计数值
    for(;;)
    {
        GPIO_PinState keyState=HAL_GPIO_ReadPin(GPIOE, GPIO_PIN_2); //PE2=KeyRight
        if (keyState==GPIO_PIN_RESET) //KeyRight 是低输入有效
        {
            BaseType_t result=xSemaphoreTake(Sem_TablesHandle, pdMS_TO_TICKS(100));
            if (result==pdTRUE)
                LCD_ShowString(10, 120, (uint8_t *)"Check in OK ");
            else
                LCD_ShowString(10, 120, (uint8_t *)"Check in fail");
            vTaskDelay(pdMS_TO_TICKS(300)); //延时，去除抖动,同时让任务调度执行
        }
        UBaseType_t availableTables=uxSemaphoreGetCount(Sem_TablesHandle);
        LCD_ShowUint(150, 70, availableTables); //剩下的桌子
        vTaskDelay(pdMS_TO_TICKS(10));
    }
}
```

3. RTC周期唤醒中断ISR函数


就在文件freertos.c中实现RTC周期唤醒事件的回调函数

HAL_RTCEx_WakeUpTimerEventCallback()

```
/* Private application code -----*/  
  
/* USER CODE BEGIN Application */  
  
void HAL_RTCEx_WakeUpTimerEventCallback(RTC_HandleTypeDef *hrtc)  
{ //定时释放信号量，相当于空出一个桌子  
    if (Sem_TablesHandle != NULL)  
    {  
        BaseType_t highTaskWoken=pdFALSE;  
        xSemaphoreGiveFromISR(Sem_TablesHandle, &highTaskWoken); //释放信号量  
        portYIELD_FROM_ISR(highTaskWoken); //申请进行一次任务调度  
    }  
}  
  
/* USER CODE END Application */
```

4. 程序运行测试

- ◆ 复位后显示桌子总数为5,
可用剩余桌子数为5



```
Demo5_2:Counting Semaphore  
Press KeyRight to check in  
Auto Check out every 3sec  
Total tables= 5  
Available tables= 5
```

- ◆ 按KeyRight键，显示“Check in OK”时会看到剩余桌数减少，这是模拟有客人进店了

```
Demo5_2:Counting Semaphore
Press KeyRight to check in
Auto Check out every 3sec

Total tables= 5

Available tables= 4
Check in OK
```

- ◆ 可以快速多按几次按键，当剩余桌数为0时再按键，就会显示“Check in fail”

```
Demo5_2:Counting Semaphore
Press KeyRight to check in
Auto Check out every 3sec

Total tables= 5
Available tables= 0
Check in fail
```


- ◆ RTC唤醒中断里会释放计数信号量，使其值加1，当然最后不会超过5。RTC唤醒周期为3秒

```
Demo5_2:Counting Semaphore
Press KeyRight to check in
Auto Check out every 3sec

Total tables= 5
Available tables= 3
Check in OK
```

练习任务

1. 看教材，练习本章的示例。