南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Embedded System and Microcomputer Principle

LAB14 FreeRTOS Queues

2022 Fall
wangq9@mail.sustech.edu.cn

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

CONTENTS

01

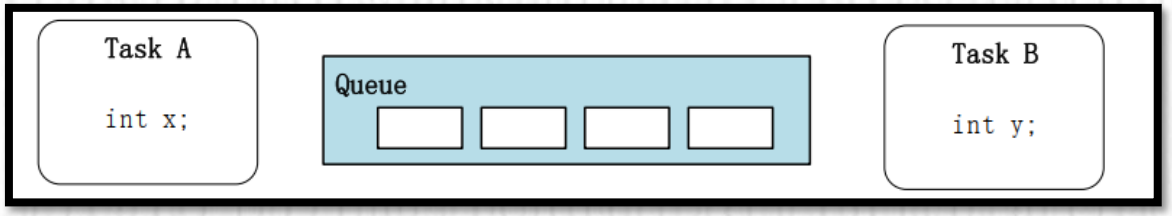
FreeRTOS Queue Description

1. FreeRTOS Queue Description

-- What is Queue

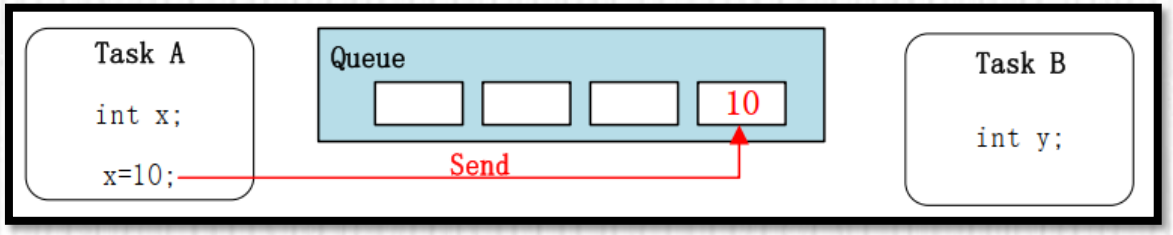
- 队列是为了任务与任务、任务与中断之间的通信而准备的，可以在任务与任务、任务与中断之间传递消息。
- 队列中可以存储有限的、大小固定的数据项目。
- 任务与任务、任务与中断之间要交流的数据保存在队列中，叫做队列项目。
- 队列所能保存的最大数据项目数量叫做队列的长度，创建队列的时候会指定数据项目的大小和队列的长度。
- 由于队列用来传递消息的，所以也称为消息队列。
- FreeRTOS 中的信号量的也是依据队列实现的。

1. FreeRTOS Queue Description
-- Queue operations



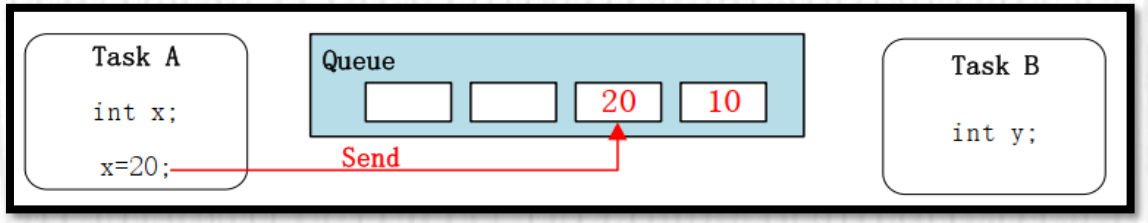
1. 创建队列

任务 A 要向 B 发送消息， 这个消息是 x 变量的值。首先创建一个队列，并指定队列长度和每条消息的长度。



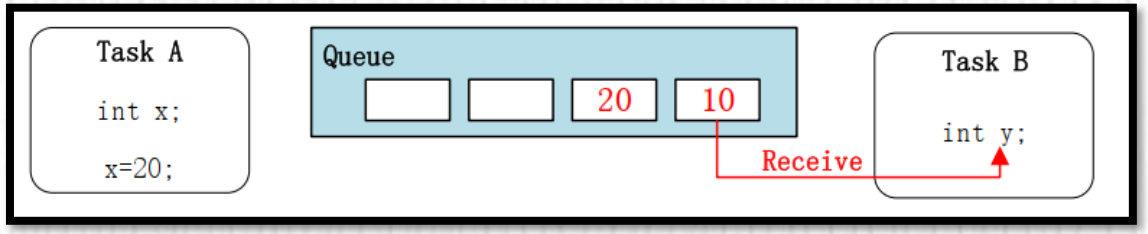
2. 向队列发送第一个消息

任务 A 的变量 x 值为 10，将这个值发送到消息队列中。此时队列剩余长度就是3了。前面说了向队列中发送消息是采用拷贝的方式，所以一旦消息发送完成变量 x 就可以再次被使用，赋其他的值。



3. 向队列发送第二个消息

任务 A 又向队列发送了一个消息，即新的 x 的值，这里是 20。 此时队列剩余长度为 2。



4. 从队列中读取消息

任务 B 从队列中读取消息，并将读取到的消息值赋值给 y，这样y就等于10了。任务 B 从队列中读取消息完成以后可以选择清除掉这个消息或者不清除。 当选择清除这个消息的话其他任务或中断就不能获取这个消息了，而且队列剩余大小就会加一，变成 3。如果不清除的话其他任务或中断也可以获取这个消息， 而队列剩余大小依旧是 2。

1. FreeRTOS Queue Description
-- Characteristics of a Queue

- 数据存储
- 多任务访问
- 出队读操作阻塞
- 入队写操作阻塞
- 阻塞多个队列

1. FreeRTOS Queue Description

-- Data Storage

- 通常队列采用先进先出(FIFO)的存储缓冲机制，也就是往队列发送数据的时候(也叫入队)永远都是发送到队列的尾部，而从队列提取数据的时候(也叫出队)是从队列的头部提取的。
- 但是也可以使用LIFO的存储缓冲，也就是后进先出，FreeRTOS中的队列也提供了LIFO的存储缓冲机制。
- 数据发送到队列中会导致数据拷贝，也就是将要发送的数据拷贝到队列中，这就意味着在队列中存储的是数据的原始值，而不是原数据的引用(即只传递数据的指针)，这个也叫做值传递。
- 与之对应，UCOS 的消息队列采用的是引用传递，传递的是消息指针。

1. FreeRTOS Queue Description

-- Data Storage(continued)

- 值传递：导致数据拷贝，会浪费一点时间，但是一旦将消息发送到队列中原始的数据缓冲区就可以删除掉或者覆写，这样的话这些缓冲区就可以被重复的使用。
- 引用传递：消息内容必须有效，那么局部变量这种可能会随时被删掉的东西就不能用来传递消息。但是采用引用传递不用进行数据拷贝，会节省很多时间。

1. FreeRTOS Queue Description -- Access by Multiple Tasks

- 队列不是属于某个特别指定的任务的，任何知道队列存在的任务或ISR都可以访问它们。
- 任意数量的任务都可以写入同一队列，任意数量的任务都可以从同一队列中读取。
- 实际上，一个队列有多个写入程序是很常见的，但一个队列有多个读卡器则不太常见。

1. FreeRTOS Queue Description -- Blocking on Queue Reads

- 出队就是就从队列中读取消息，出队阻塞是针对从队列中读取消息的任务而言的。当任务尝试从一个队列中读取消息的时候可以指定一个阻塞时间，这个阻塞时间就是当任务从队列中读取消息无效的时候任务阻塞的时间。
- 比如任务 A 要从队列 Q 中读取数据，但如果此时队列 Q 是空的，A 获取不到任何东西，该如何处理？任务 A 有三种选择，1，直接放弃；2，在一个时间范围内等待；3，一直等待直至有数据。这3种选择就是由阻塞时间决定的，阻塞时间单位是时钟节拍数。
- 阻塞时间为 0 的话就是不阻塞，没有数据则马上返回任务，对应第1种选择。
- 如果阻塞时间为 0~ portMAX_DELAY-1，当任务没有从队列中获取到消息的话就进入阻塞态，当阻塞时间到了还没有接收到数据的话就退出阻塞态，返回任务接着运行下面的代码；如果在阻塞时间内接收到了数据就立即返回，执行任务中下面的代码，对应第2种选择。
- 当阻塞时间设置为portMAX_DELAY 的话，任务就会一直进入阻塞态等待，直到接收到数据为止，对应第3种选择。

1. FreeRTOS Queue Description
-- Blocking on Queue Writes

- 入队说的是向队列中发送消息， 将消息加入到队列中。
- 任务可以选择在写入队列时指定块时间。在这种情况下，如果队列已满，则阻塞时间是任务在阻塞状态下等待队列上的可用空间的最长时间。其处理过程与出队阻塞是类似的，只不过一个是向队列 Q 发送消息，一个是从队列 Q 读取消息而已。
- 队列可以有多个写入程序，因此一个完整队列可能会有多个任务被阻塞，等待完成发送操作。在这种情况下，当队列上的空间可用时，只有一个任务将被取消阻塞。未阻塞的任务将始终是等待空间的优先级最高的任务。如果被阻止的任务具有相同的优先级，则等待空间最长的任务将被解除阻止。

1. FreeRTOS Queue Description

-- Blocking on Multiple Queues

- 队列可以分组到集合中，允许任务进入阻塞状态，以等待集合中任何队列上的数据可用。

1. FreeRTOS Queue Description -- Queue structure

- 有一个结构体用于描述队列，叫做 Queue_t，这个结构体在文件 queue.c 中定义

```
typedef struct QueueDefinition
{
    int8_t *pcHead;                 //指向队列存储区开始地址。
    int8_t *pcTail;                 //指向队列存储区最后一个字节。
    int8_t *pcWriteTo;              //指向存储区中下一个空闲区域。

    union
    {
        int8_t *pcReadFrom;         //当用作队列的时候指向最后一个出队的队列项首地址
        UBaseType_t uxRecursiveCallCount;//当用作递归互斥量的时候用来记录递归互斥量被
                                        //调用的次数。
    } u;

    List_t xTasksWaitingToSend;     //等待发送任务列表，那些因为队列满导致入队失败而进
                                    //入阻塞态的任务就会挂到此列表上。
    List_t xTasksWaitingToReceive;  //等待接收任务列表，那些因为队列空导致出队失败而进
                                    //入阻塞态的任务就会挂到此列表上。

    volatile UBaseType_t uxMessagesWaiting;  //队列中当前队列项数量，也就是消息数
    UBaseType_t uxLength;           //创建队列时指定的队列长度，也就是队列中最大允许的
                                    //队列项(消息)数量
    UBaseType_t uxItemSize;         //创建队列时指定的每个队列项(消息)最大长度，单位字节

    volatile int8_t cRxLock;        //当队列上锁以后用来统计从队列中接收到的队列项数
                                    //量，也就是出队的队列项数量，当队列没有上锁的话此字
                                    //段为 queueUNLOCKED

    volatile int8_t cTxLock;        //当队列上锁以后用来统计发送到队列中的队列项数量，
                                    //也就是入队的队列项数量，当队列没有上锁的话此字
                                    //段为 queueUNLOCKED

    #if( ( configSUPPORT_STATIC_ALLOCATION == 1 ) &&\
         ( configSUPPORT_DYNAMIC_ALLOCATION == 1 ) )
        uint8_t ucStaticallyAllocated;//如果使用静态存储的话此字段设置为 pdTURE。
    #endif
    #if ( configUSE_QUEUE_SETS == 1 )       //队列集相关宏
        struct QueueDefinition *pxQueueSetContainer;
    #endif

    #if ( configUSE_TRACE_FACILITY == 1 ) //跟踪调试相关宏
        UBaseType_t uxQueueNumber;
        uint8_t ucQueueType;
    #endif

} xQUEUE;

typedef    xQUEUE       Queue_t;
```

1. FreeRTOS Queue Description

-- Queue API

- xQueueCreate()
- xQueueCreateStatic()
- vQueueDelete()
- xQueueSend()
- xQueueSendFromISR()
- xQueueSendToBack()
- xQueueSendToBackFromISR()
- xQueueSendToFront()
- xQueueSendToFrontFromISR()
- xQueueReceive()
- xQueueReceiveFromISR()
- uxQueueMessagesWaiting()
- uxQueueMessagesWaitingFromISR()
- uxQueueSpacesAvailable()
- xQueueReset()
- xQueueOverwrite()
- xQueueOverwriteFromISR()
- xQueuePeek()
- xQueuePeekFromISR()
- vQueueAddToRegistry()
- vQueueUnregisterQueue()
- pcQueueGetName()
- xQueueIsQueueFullFromISR()
- xQueueIsQueueEmptyFromISR()

1. FreeRTOS Queue Description

-- Queue Set API

- xQueueCreateSet()
- xQueueAddToSet()
- xQueueRemoveFromSet()
- xQueueSelectFromSet()
- xQueueSelectFromSetFromISR()

02

CMSIS-RTOS Queue API

2. CMSIS-RTOS Queue API

-- CMSIS-RTOS queue structures

- osStatus: CMSIS-RTOS函数返回的状态代码值。
- Event structure包含有关事件的详细信息。
- Message ID标识消息队列（指向消息队列控制块的指针）。

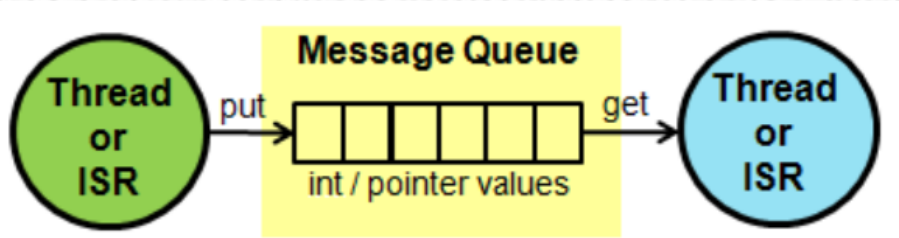
```
typedef enum  {
  osOK                     =     0,       ///< function completed; no error or event occurred.
  osEventSignal            =  0x08,       ///< function completed; signal event occurred.
  osEventMessage           =  0x10,       ///< function completed; message event occurred.
  osEventMail              =  0x20,       ///< function completed; mail event occurred.
  osEventTimeout           =  0x40,       ///< function completed; timeout occurred.
  osErrorParameter         =  0x80,       ///< parameter error: a mandatory parameter was missing or specified an incorrect object.
  osErrorResource          =  0x81,       ///< resource not available: a specified resource was not available.
  osErrorTimeoutResource   =  0xC1,       ///< resource not available within given time: a specified resource was not available within the timeout
  osErrorISR               =  0x82,       ///< not allowed in ISR context: the function cannot be called from interrupt service routines.
  osErrorISRRecursive      =  0x83,       ///< function called multiple times from ISR with same object.
  osErrorPriority          =  0x84,       ///< system cannot determine priority or thread has illegal priority.
  osErrorNoMemory          =  0x85,       ///< system is out of memory: it was impossible to allocate or reserve memory for the operation.
  osErrorValue             =  0x86,       ///< value of a parameter is out of range.
  osErrorOS                =  0xFF,       ///< unspecified RTOS error: run-time error but no other error message fits.
  os_status_reserved       =  0x7FFFFFFF  ///< prevent from enum down-size compiler optimization.
} osStatus;
```

```
typedef struct  {
  osStatus                status;     ///< status code: event or error information
  union  {
    uint32_t                v;     ///< message as 32-bit value
    void                   *p;     ///< message or mail as void pointer
    int32_t              signals;     ///< signal flags
  } value;                          ///< event value
  union  {
    osMailQId            mail_id;     ///< mail id obtained by \ref osMailCreate
    osMessageQId      message_id;     ///< message id obtained by \ref osMessageCreate
  } def;                            ///< event definition
} osEvent;
```

```
typedef QueueHandle_t osMessageQId;
```

```
typedef struct os_mailQ_cb *osMailQId;
```

2. CMSIS-RTOS Queue API
-- CMSIS-RTOS message queue



CMSIS-RTOS Message Queue

- 消息传递是线程之间的基本通信模型。在消息传递模型中，一个线程显式发送数据，而另一个线程接收数据。
- 该操作更像某种I/O，而不是直接访问要共享的信息。在CMSIS-RTOS中，这种机制称为s消息队列。数据通过类似FIFO的操作从一个线程传递到另一个线程。使用消息队列函数，您可以控制、发送、接收或等待消息。要传递的数据可以是整数或指针类型。
- 与内存池相比，消息队列通常效率较低，但可以解决更广泛的问题。有时，线程没有公共地址空间，或者使用共享内存会引发问题，例如互斥。

2. CMSIS-RTOS Queue API
-- Message queue macros

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

#define osFeature_MessageQ 1

A CMSIS-RTOS implementation may support message queues.
1 = available, 0 = not available

#define osMessageQ (name) &os_messageQ_def__##name

Access to the message queue definition for the function osMessageCreate.

name parameter: name of the queue.

#define osMessageQDef (name, queue_sz, type)

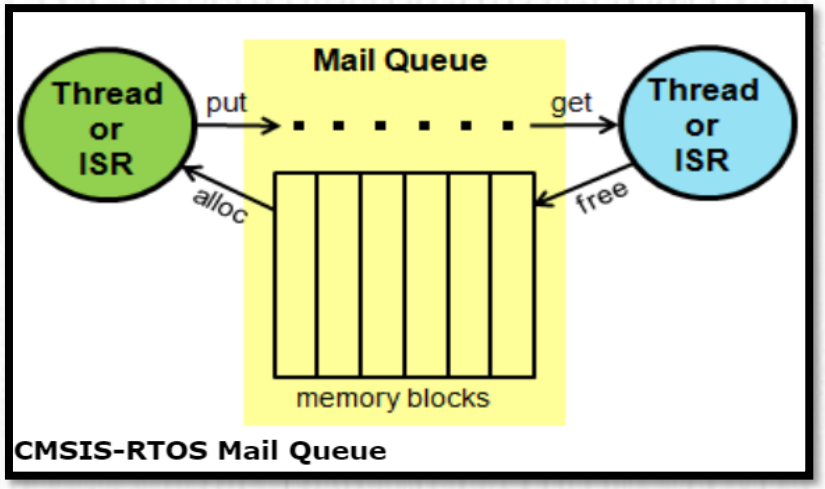
Define the attributes of a message queue created by the function osMessageCreate using osMessageQ.

name parameter: name of the queue.
queue_sz parameter: maximum number of messages in the queue.
type parameter: data type of a single message element (for debugger).

2. CMSIS-RTOS Queue API
-- Message queue functions

osMessageQId osMessageCreate (const osMessageQDef_t *queue_def, osThreadId thread_id)
Create and Initialize a Message Queue.
queue_def parameter: queue definition referenced with osMessageQ. **thread_id** parameter: thread ID (obtained by osThreadCreate or osThreadGetId) or NULL. returns: message queue ID for reference by other functions or NULL in case of error.
osEvent osMessageGet (osMessageQId queue_id, uint32_t millisec)
Get a Message or Wait for a Message from a Queue.
queue_id parameter: message queue ID obtained with osMessageCreate. **millisec** parameter: timeout value or 0 in case of no time-out. returns: event information that includes status code.
osStatus osMessagePut (osMessageQId queue_id, uint32_t info, uint32_t millisec)
Put a Message to a Queue.
queue_id parameter: message queue ID obtained with osMessageCreate. **info** parameter: message information. **millisec** parameter: timeout value or 0 in case of no time-out. returns: status code that indicates the execution status of the function.

2. CMSIS-RTOS Queue API
-- CMSIS-RTOS mail queue



- 邮件队列类似于消息队列，但正在传输的数据由需要分配（放入数据之前）和释放（取出数据之后）的内存块组成。
- 邮件队列类似于消息队列，但正在传输的数据由需要分配（放入数据之前）和释放（取出数据之后）的内存块组成。
- 邮件队列使用内存池创建格式化内存块，并在消息队列中传递指向这些块的指针。这允许数据保留在分配的内存块中，而只有一个指针在单独的线程之间移动。与只能传输32位值或指针的消息相比，这是一个优势。使用邮件队列功能，您可以控制、发送、接收或等待邮件。

2. CMSIS-RTOS Queue API -- Mail queue macros

#define osFeature_MailQ 1

A CMSIS-RTOS implementation may support mail queues.
1 = available, 0 = not available

#define osMailQ (name) &os_mailQ_def__##name

Access to the mail queue definition for the function osMailCreate.

name parameter: name of the queue.

#define osMailQDef (name, queue_sz, type)

Define the attributes of a mail queue created by the function osMailCreate using osMaiQ.

name parameter: name of the queue.
queue_sz parameter: maximum number of messages in the queue.
type parameter: data type of a single message element.

2. CMSIS-RTOS Queue API -- Mail queue functions

osMailQId osMailCreate (const osMailQDef_t *queue_def, osThreadId thread_id)
Create and Initialize a Mail Queue.
queue_def parameter: reference to the mail queue definition obtain with osMailQ. **thread_id** parameter: thread ID (obtained by osThreadCreate or osThreadGetId) or NULL. returns: mail queue ID for reference by other functions or NULL in case of error.
void *osMailAlloc (osMailQId queue_id, uint32_t millisec)
Allocate a memory block from a mail.
queue_id parameter: mail queue ID obtained with osMailCreate. **millisec** parameter: timeout value or 0 in case of no time-out. returns: pointer to memory block that can be filled with mail or NULL in case of error.
void *osMailCAlloc (osMailQId queue_id, uint32_t millisec)
Allocate a memory block from a mail and set memory block to zero.
queue_id parameter: mail queue ID obtained with osMailCreate. **millisec** parameter: timeout value or 0 in case of no time-out. returns: pointer to memory block that can be filled with mail or NULL in case of error.

2. CMSIS-RTOS Queue API
-- Mail queue functions(continued)

osStatus osMailPut (osMailQId queue_id, void *mail)
Put a mail to a queue.
queue_id parameter: mail queue ID obtained with osMailCreate. **mail** parameter: memory block previously allocated with osMailAlloc or osMailCAlloc. returns: status code that indicates the execution status of the function.
osEvent osMailGet (osMailQId queue_id, uint32_t millisec)
Get a mail from a queue.
queue_id parameter: mail queue ID obtained with osMailCreate. **millisec** parameter: timeout value or 0 in case of no time-out. returns: event that contains mail information or error code.
osStatus osMailFree (osMailQId queue_id, void *mail)
Free a memory block from a mail.
queue_id parameter: mail queue ID obtained with osMailCreate. **mail** parameter: pointer to the memory block that was obtained with osMailGet. returns: status code that indicates the execution status of the function.

03

How to Program

3. How to Program

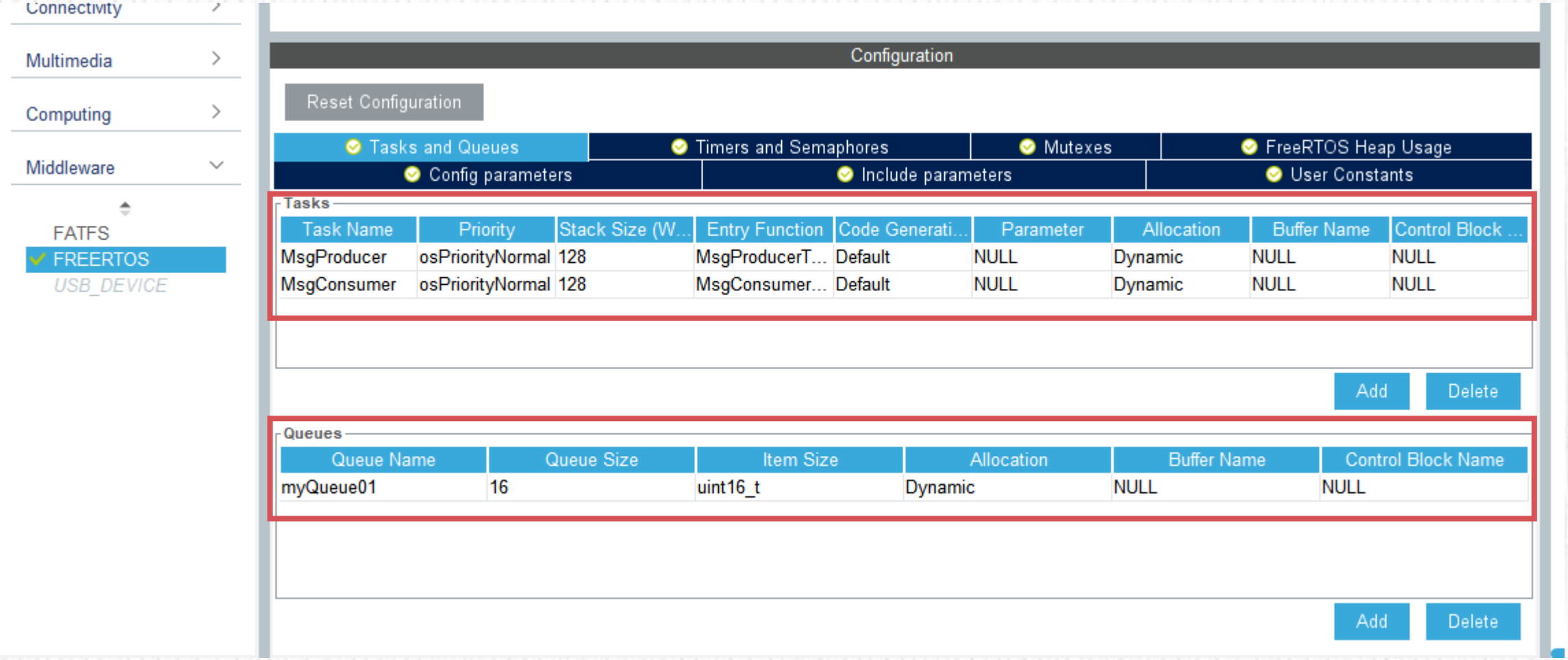
- Our Goal
  - Use message queue to communicate between tasks.
  - Use mail queue to communicate between tasks.

3. How to Program

- Configure SYS
  - Remember to configure the basetime source
- Configure RCC
- Configure USART
- Configure FreeRTOS
  - Set the FreeRTOS API as CMSIS_v1

3. How to Program

- Add a queue(message queue defaulted) and two tasks.



3. How to Program

- Generate the code and implement the function **MsgProducerTask** and **MsgConsumerTask**.

```
void MsgProducerTask(void const * argument)
{
  /* USER CODE BEGIN MsgProducerTask */
  /* Infinite loop */
  for(;;)
  {
    osDelay(1000);
    osMessagePut(myQueue01Handle, 2, osWaitForever);
    osDelay(2000);
    osMessagePut(myQueue01Handle, 4, osWaitForever);
  }
  /* USER CODE END MsgProducerTask */
}
```

```
void MsgConsumerTask(void const * argument)
{
  /* USER CODE BEGIN MsgConsumerTask */
  osEvent event;
  char msg[20];
  /* Infinite loop */
  for(;;)
  {
    event = osMessageGet(myQueue01Handle, osWaitForever);
    if (event.status == osEventMessage)
    {
      sprintf(msg, "Msg value: %d\r\n", (uint16_t)event.value.v);
      HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg),
HAL_MAX_DELAY);
    }
  }
  /* USER CODE END MsgConsumerTask */
}
```

3. How to Program

- Use **mail queue.**
- Although API about mail queues has been integrated into CMSIS-RTOS, STM32Cube doesn't provide the graphical configuration of mail queues. We need to create it by coding.
- As the macro in definition and creation is similar to message queue, so the code can be like following:

```
/* USER CODE BEGIN Variables */
typedef struct
{
  uint16_t var;
  } mailStruct;
osMailQId mail01Handle;
/* USER CODE END Variables */
```

```
// ...
/* USER CODE BEGIN RTOS_QUEUES */
/* add queues, ... */
  osMailQDef(mail01, 16, mailStruct);
  mail01Handle = osMailCreate(osMailQ(mail01),
NULL);
/* USER CODE END RTOS_QUEUES */
// ...
```

- We use memory pool to allocate and free the memories use to store mail.

3. How to Program

- Implement the task functions.

```
void MsgProducerTask(void const * argument)
{
  /* USER CODE BEGIN MsgProducerTask */
  mailStruct * mail;
  /* Infinite loop */
  for(;;)
  {
    osDelay(1000);
    mail = (mailStruct *)osMailAlloc(mail01Handle,
osWaitForever);
    mail->var = 1;
    osMailPut(mail01Handle, mail);
    osDelay(2000);
    mail = (mailStruct *)osMailAlloc(mail01Handle,
osWaitForever);
    mail->var = 2;
    osMailPut(mail01Handle, mail);
  }
  /* USER CODE END MsgProducerTask */
}
```

```
void MsgConsumerTask(void const * argument)
{
  /* USER CODE BEGIN MsgConsumerTask */
  osEvent event;
  mailStruct * pMail;
  char msg[20];
  /* Infinite loop */
  for(;;)
  {
    event = osMailGet(mail01Handle, osWaitForever);
    if (event.status == osEventMail)
    {
      pMail = event.value.p;
      sprintf(msg, "Mail value: %d\r\n", pMail->var);
      HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg),
HAL_MAX_DELAY);
      osMailFree(mail01Handle, pMail);
    }
  }
  /* USER CODE END MsgConsumerTask */
}
```

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.
- Using mail queues to solve the Producer-Consumer problem.