FreeRTOS 学习之三：计数信号量

前提：默认已经装好 MDK V5 和 STM32CubeMX，并安装了 STM32F1xx 系列的支持包。

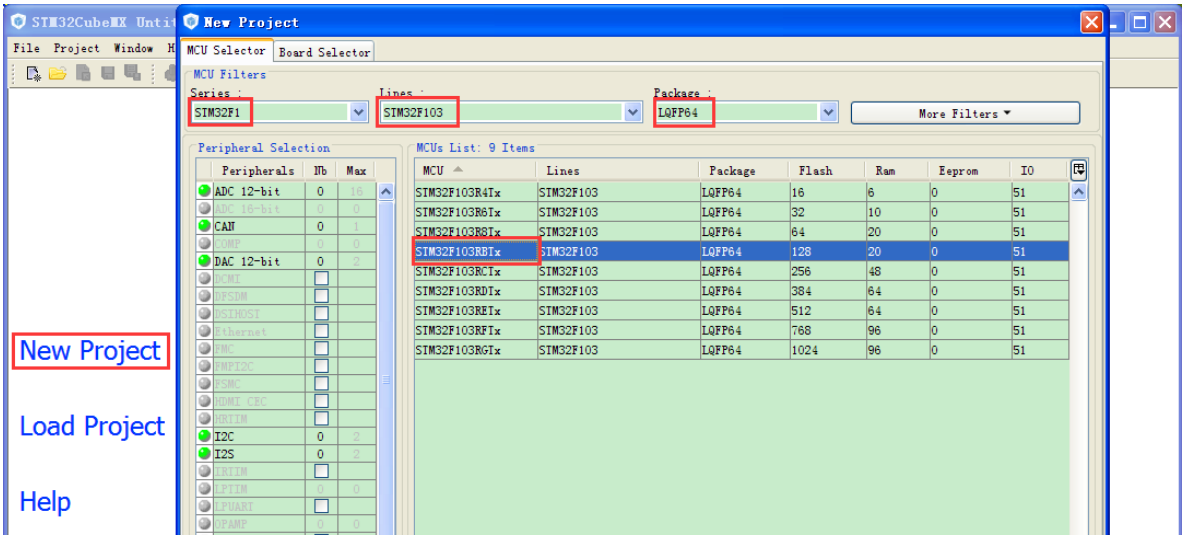
硬件平台：STM32F1xx 系列。

目的：学习计数信号量的使用。

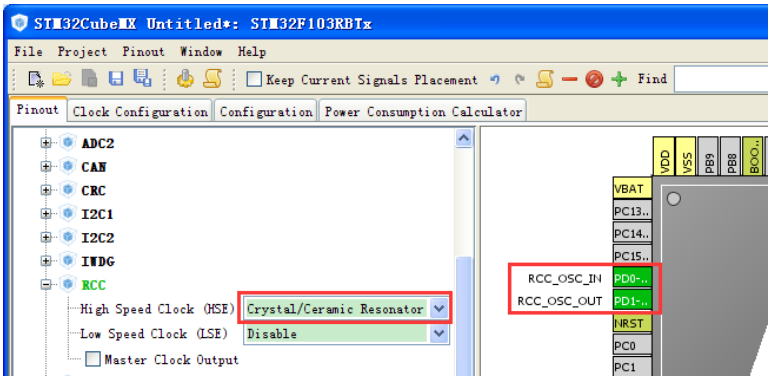
计数信号量的使用场景：一个二值信号量最多只可以锁存一个中断事件。在锁存的事件还未被处理之前，如果还有中断事件发生，那么后续发生的中断事件将会丢失。如果 用计数信号量代替二值信号量，那么，这种丢中断的情形将可以避免。

本文例子使用 STM32CubeMX 配置创建两个任务，一个任务每秒钟发送多次信号量，另一个等待信号量并控制 LED 的闪烁。

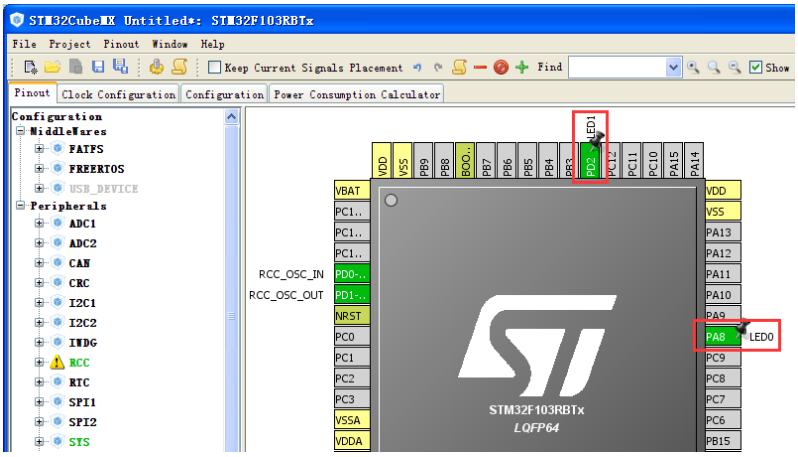
Step1.打开 STM32CubeMX，点击“New Project”，选择芯片型号，STM32F103RBTx。



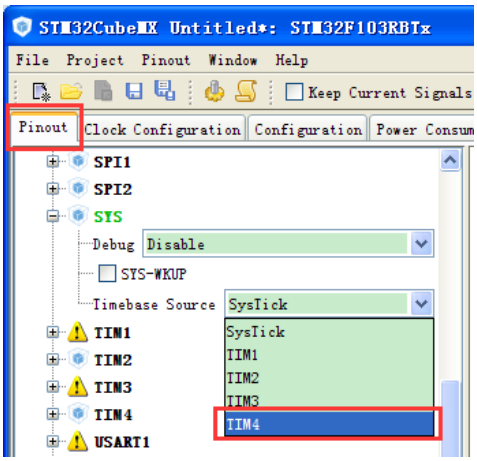
Step2.配置时钟引脚。



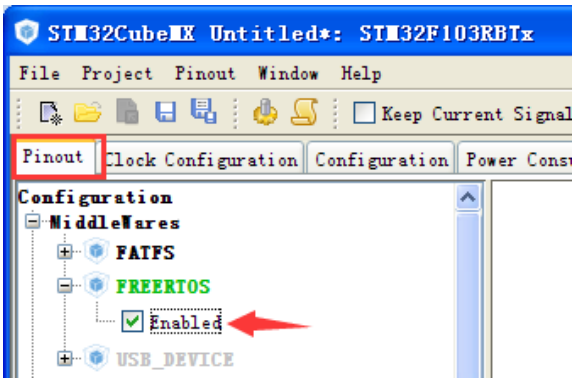
Step3.配置 PA8 和 PD2 为 Output，并把用户标签分别改为 LED0，LED1。



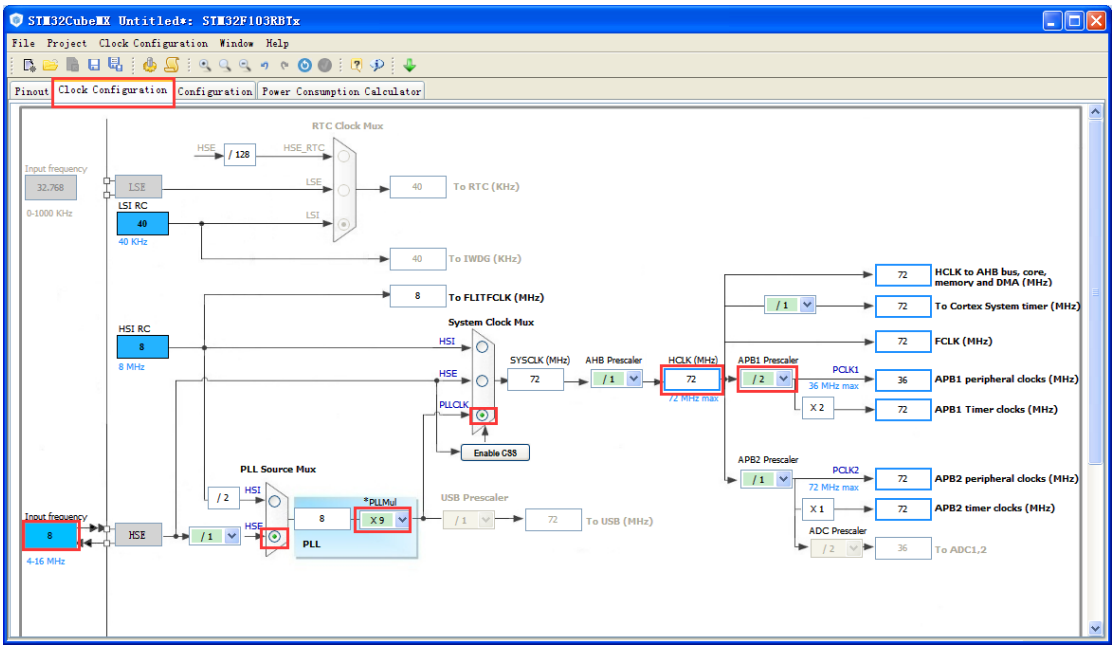
Step4.将系统时基源改为 TIM4。



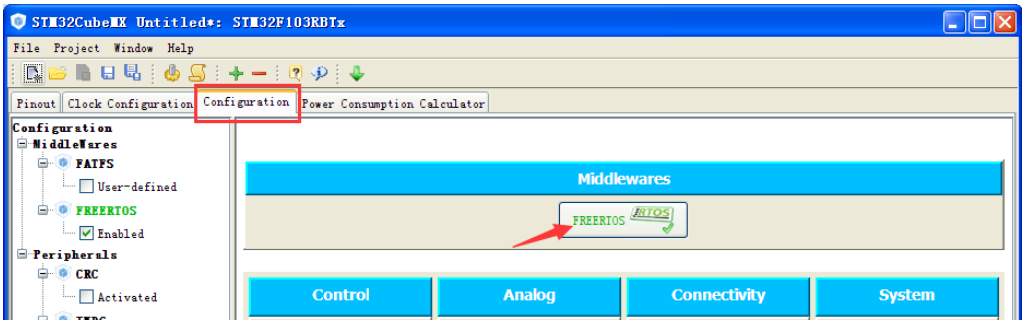
Step5.使能 FreeRTOS。



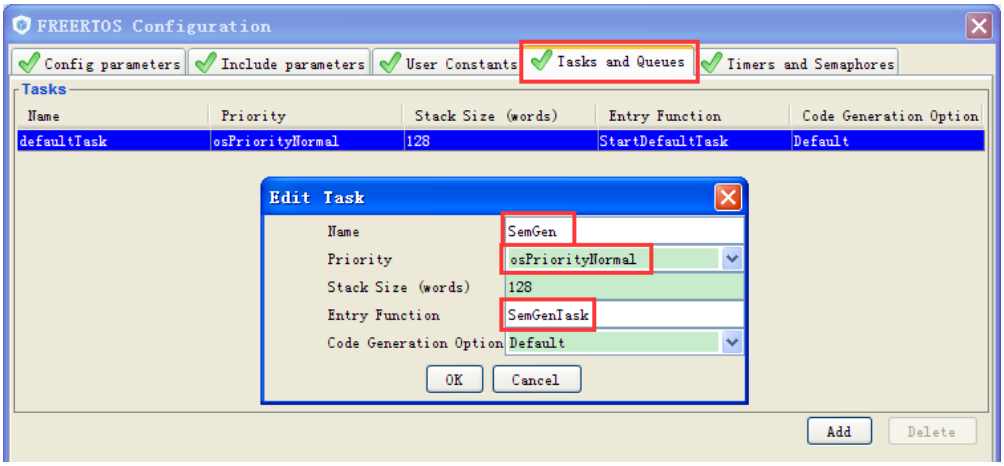
Step6.配置时钟树。8M 输入时，通过 PLL 得到 72M 内部时钟。



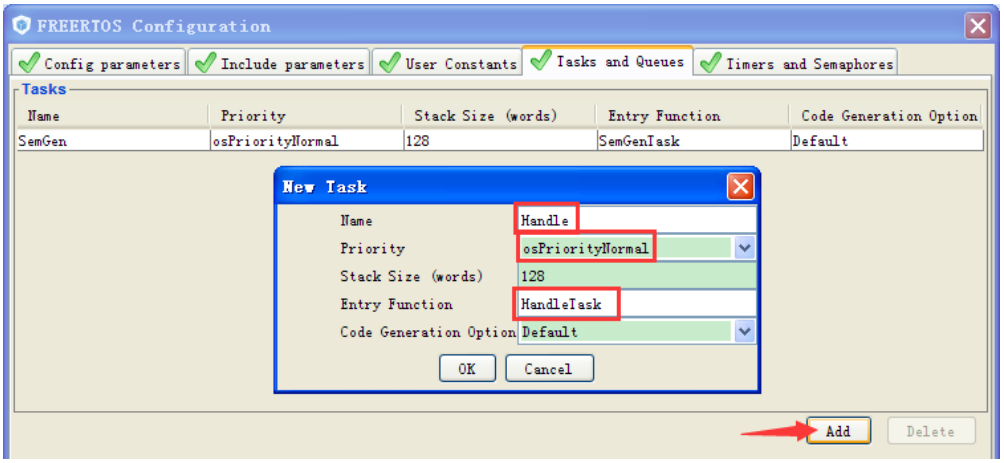
Step7.配置 FreeRTOS。



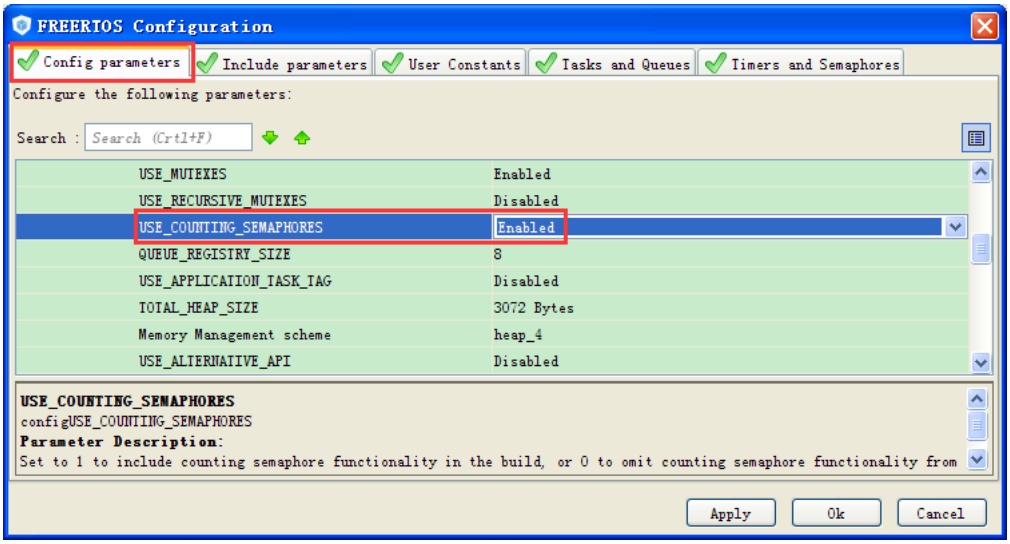
在 Tasks and Queues 选项卡，默认配置了一个名为 defaultTask 的任务，其优先级为普通，任务堆栈大小为 128 字，任务函数名为 StartDefaultTask。
双击蓝色的地方，弹出对话框，将任务名修改为 SemGen，将任务函数名修改为 SemGenTask。



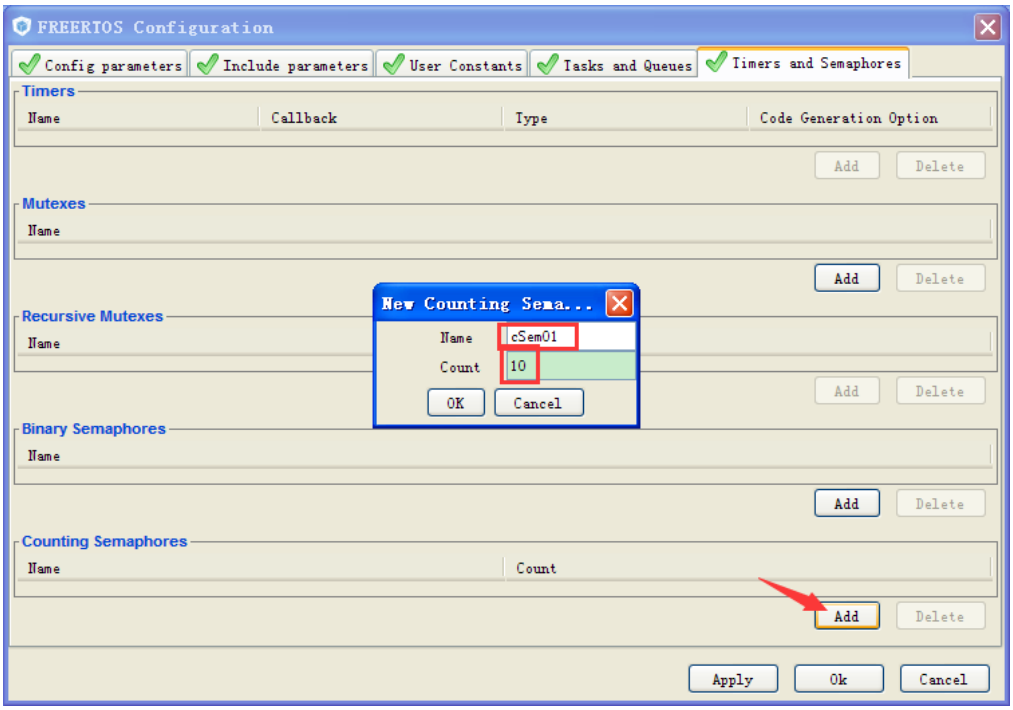
点击 Add 按钮，增加一个任务 Handle，优先级设置为 Normal，函数名为 HandleTask。



在 Config parameters 选项卡，使能计数信号量。

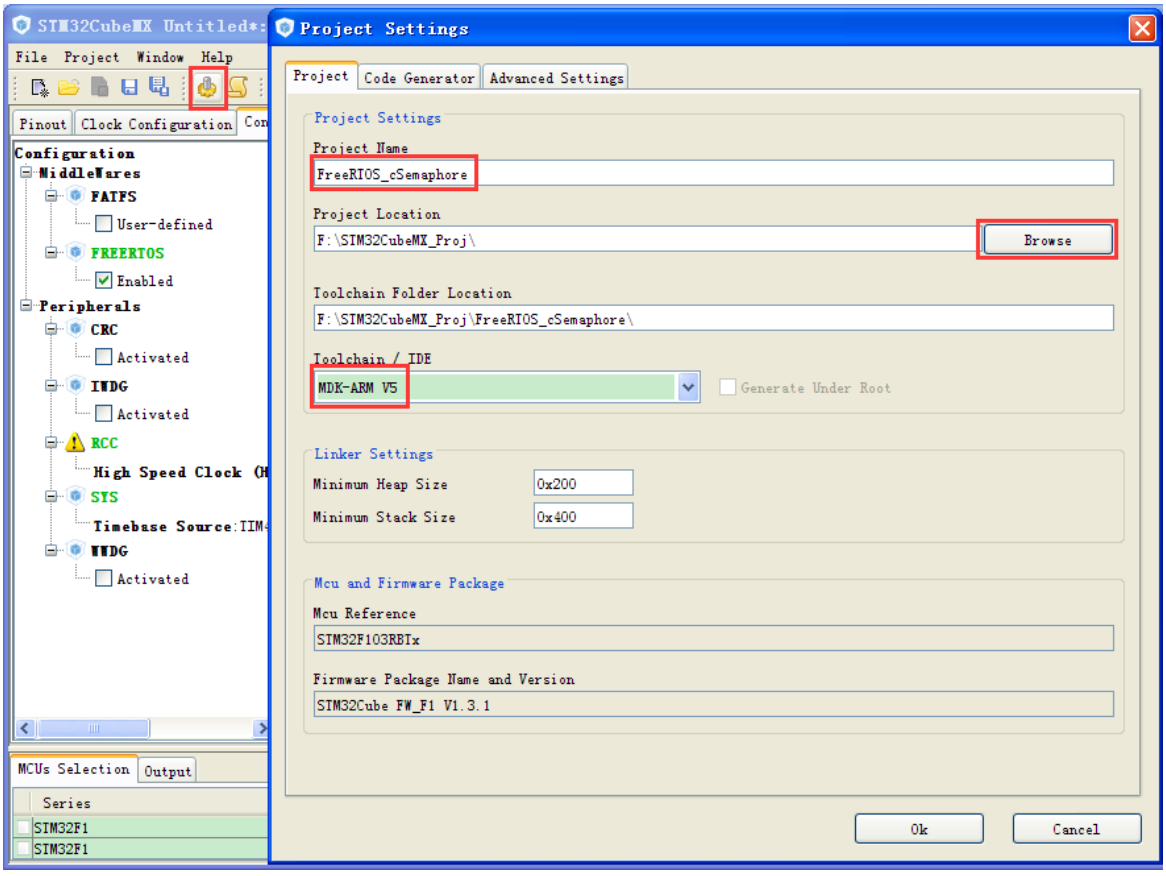


在 Timers and Semaphores 选项卡，点击 Counting Semaphores 项右边的“Add”按钮，添加一个信号量，名称改为 cSem01，并把最大计数值改为 10。

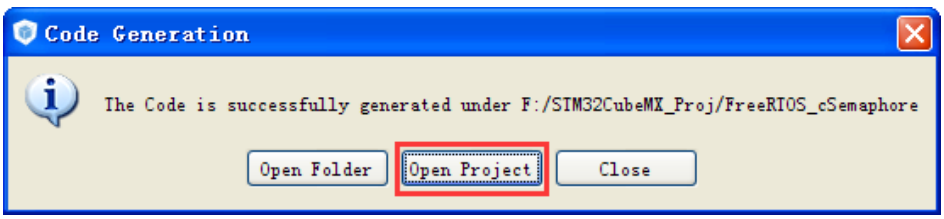


注：其他的都使用默认参数。

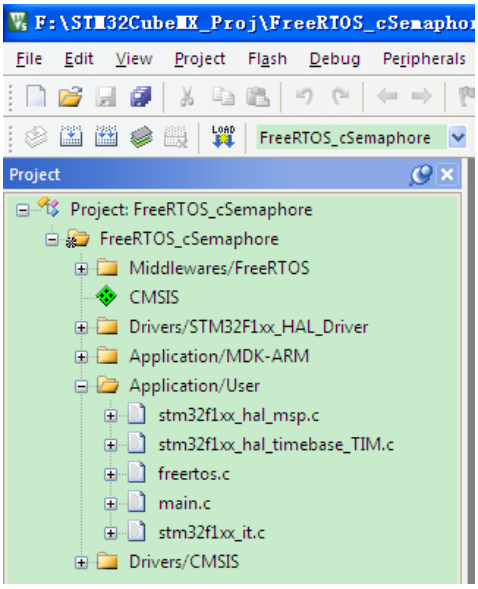
Step8.生成代码。



等完成后直接打开工程。



工程基本组织结构如下图，其中 Application/User 组中的文件是用户可以修改的，而其他组中的文件一般不进行修改。



Step9.分析程序结构。

在进入 main 函数之前，先定义了几个变量，声明了几个函数。

```
/* Private variables ---------------------
osThreadId SemGenHandle;
osThreadId HandleHandle;
osSemaphoreId cSem01Handle;

/* USER CODE BEGIN PV */
/* Private variables ---------------------

/* USER CODE END PV */

/* Private function prototypes ----------
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
void SemGenTask(void const * argument);
void HandleTask(void const * argument);
```

再看 main 函数。将 main 函数整理，删除很多注释之后，得到下图所示内容。

```
int main(void)
{
  /* Reset of all peripherals, Initializes the Flash interface and the
  HAL_Init();
  /* Configure the system clock */          ①
  SystemClock_Config();
  /* Initialize all configured peripherals */
  MX_GPIO_Init();

  /* Create the semaphores(s) */
  /* definition and creation of cSem01 */                ②
  osSemaphoreDef(cSem01);
  cSem01Handle = osSemaphoreCreate(osSemaphore(cSem01), 10);
  /* Create the thread(s) */
  /* definition and creation of SemGen */
  osThreadDef(SemGen, SemGenTask, osPriorityNormal, 0, 128);
  SemGenHandle = osThreadCreate(osThread(SemGen), NULL);
  /* definition and creation of Handle */
  osThreadDef(Handle, HandleTask, osPriorityNormal, 0, 128);
  HandleHandle = osThreadCreate(osThread(Handle), NULL);
  /* Start scheduler */
  osKernelStart();      ③

  /* We should never get here as control is now taken by the scheduler

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

其中第①部分，是硬件配置；第②部分，创建一个信号量和两个任务；第③部分，启动调度器。

Step10.添加代码。

在 main.c 文件中，找到前面配置添加的两个任务函数，并在其中分别添加代码。

SemGenTask 的功能是，发送 1 次信号量，间隔一秒后发送 2 次，再间隔一秒发送 3 次，然后等待 2 秒。

```
/* SemGenTask function */
void SemGenTask(void const * argument)
{

  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
      osDelay(1000);
      osSemaphoreRelease(cSem01Handle); // 释放(发送)1次信号量
      osDelay(1000);
      osSemaphoreRelease(cSem01Handle); // 释放(发送)2次信号量
      osSemaphoreRelease(cSem01Handle);
      osDelay(1000);
      osSemaphoreRelease(cSem01Handle); // 释放(发送)3次信号量
      osSemaphoreRelease(cSem01Handle);
      osSemaphoreRelease(cSem01Handle);
      osDelay(2000);
  }
  /* USER CODE END 5 */
}
```

HandleTask 的功能的，等待信号量，然后控制 LED0 和 LED1 闪烁一次。

```
/* HandleTask function */
void HandleTask(void const * argument)
{
  /* USER CODE BEGIN HandleTask */
  /* Infinite loop */
  for(;;)
  {
      osSemaphoreWait(cSem01Handle , osWaitForever);    // 等待信号量
      HAL_GPIO_WritePin(LED0_GPIO_Port, LED0_Pin, GPIO_PIN_RESET);  //LED0亮
      HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);  //LED1亮
      osDelay(100);
      HAL_GPIO_WritePin(LED0_GPIO_Port, LED0_Pin, GPIO_PIN_SET);    //LED0灭
      HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET);    //LED1灭
      osDelay(100);
  }
  /* USER CODE END HandleTask */
}
```

Step11.编译下载运行。现象是，LED 闪 1 次，一秒后闪 2 次，再过一秒闪 3 次，再等三秒，LED 闪 1 次…如此循环。

程序分析：

在 SemGenTask()任务中，连续调用 osSemaphoreRelease(cSem01Handle);两次或者三次耗时是很短的，而 HandleTask()任务执行一次 LED 闪烁耗时大约是 200 毫秒，会造成信号量值的累积。

如果把信号量的定义语句 cSem01Handle = osSemaphoreCreate(osSemaphore(cSem01), 10);的最后一个参数改为 1，即定义改为 cSem01Handle = osSemaphoreCreate(osSemaphore(cSem01), 1);，这样就变成了二值信号量。运行结果是，LED 闪 1 次，一秒后闪 1 次，再过一秒闪 1 次，再等三秒，LED 闪 1 次…如此循环。

由此可知，在连续释放二值信号时，如果处理信号量相关事件的函数来不及处理，就会造成事件的丢失。

S.D.Lu 于 深圳

2016 年 8 月