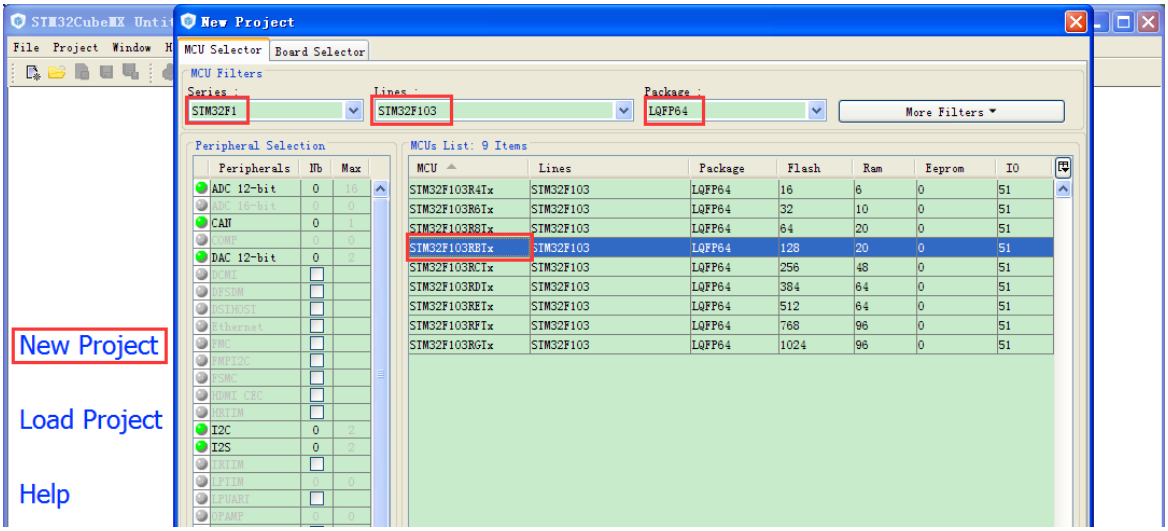
FreeRTOS 学习之二：二值信号量

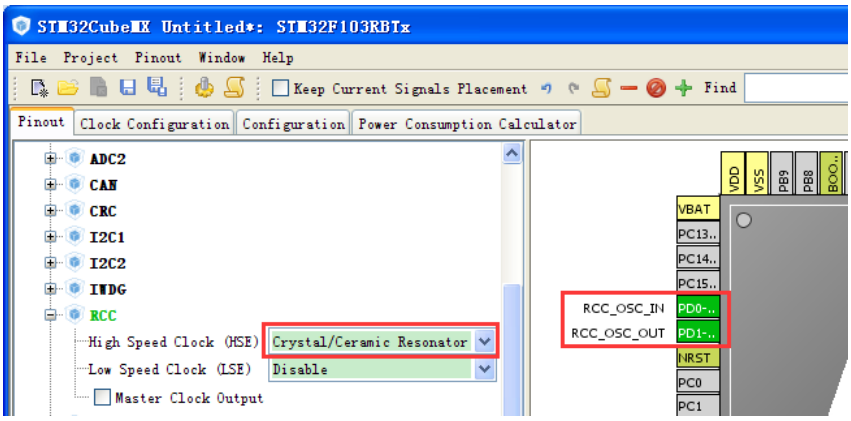
前提：默认已经装好 MDK V5 和 STM32CubeMX，并安装了 STM32F1xx 系列的支持包。
硬件平台：STM32F1xx 系列。
目的：学习使用二值信号量进行任务同步。

二值信号量主要用于任务的同步。本文例子使用 STM32CubeMX 配置创建两个任务，一个任务每秒钟发送一次信号量，另一个等待信号量并控制 LED 的输出状态。

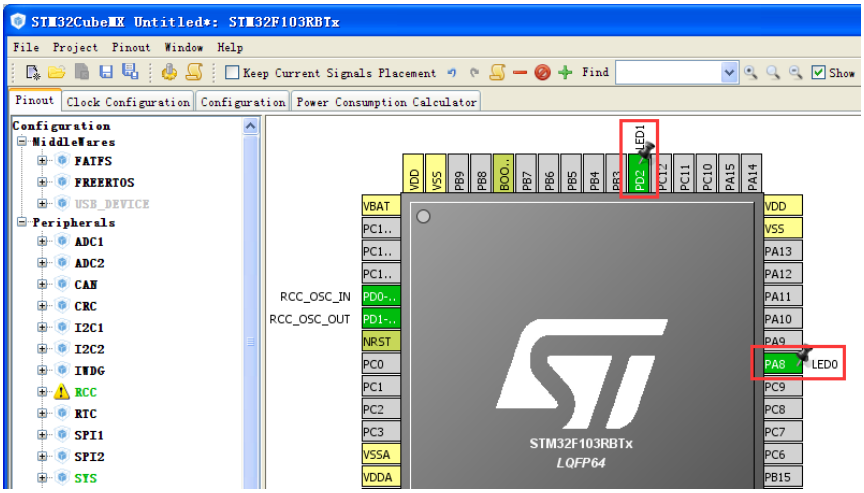
Step1.打开 STM32CubeMX，点击“New Project”，选择芯片型号，STM32F103RBTx。



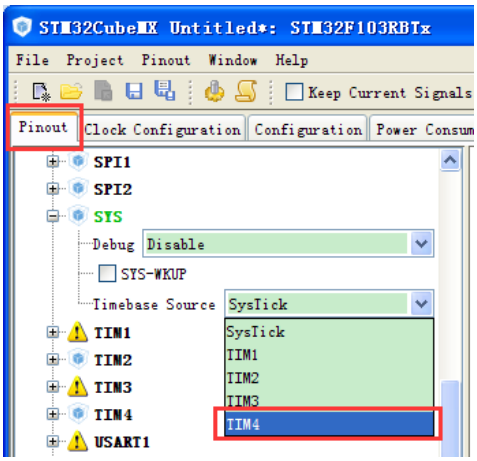
Step2.配置时钟引脚。



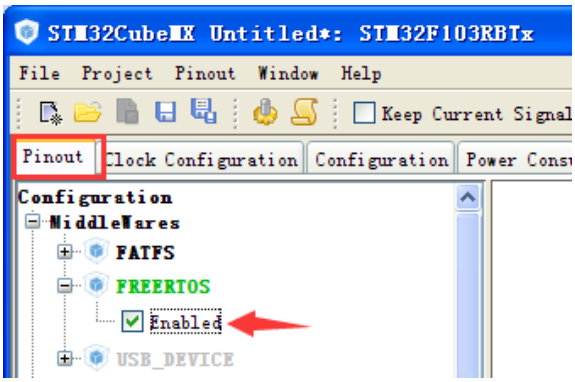
Step3.配置 PA8 和 PD2 为 Output，并把用户标签分别改为 LED0，LED1。



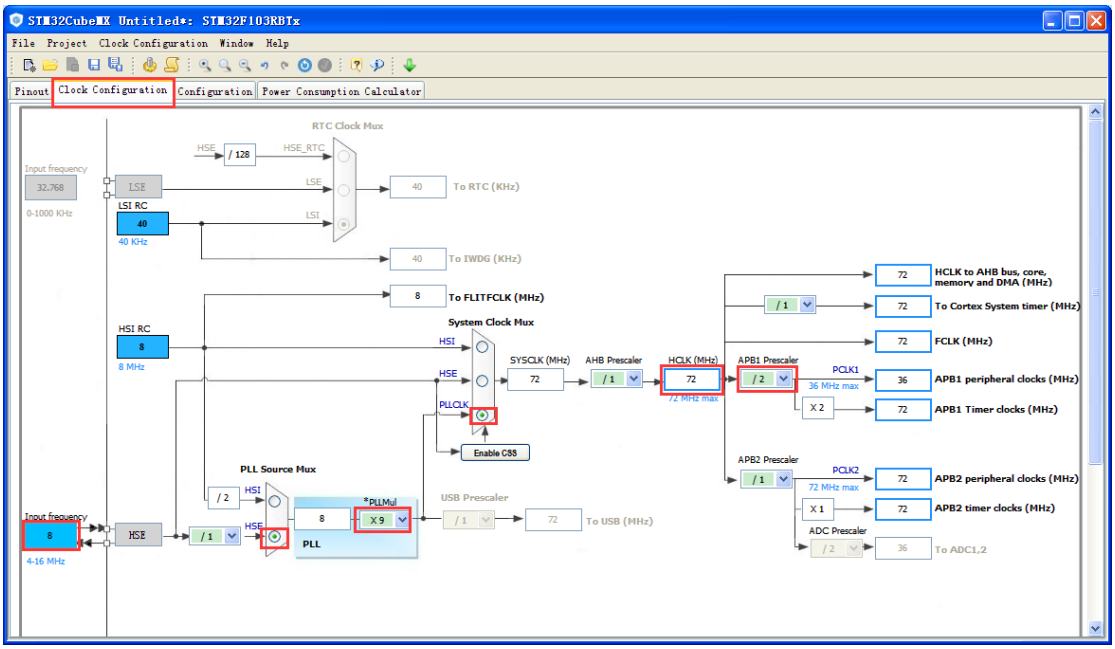
Step4.将系统时基源改为 TIM4。



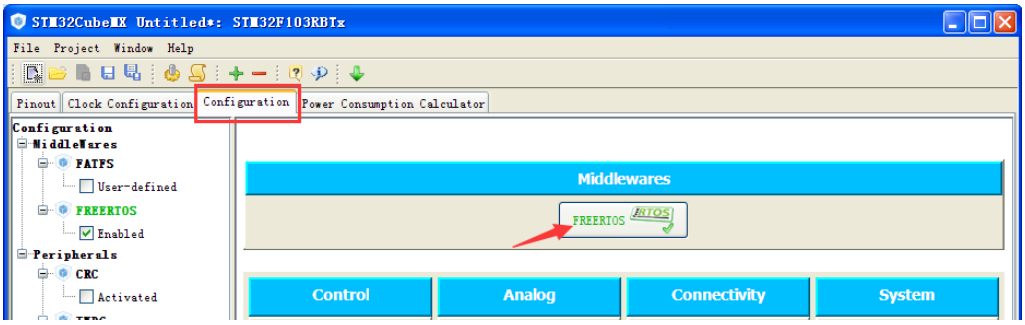
Step5.使能 FreeRTOS。



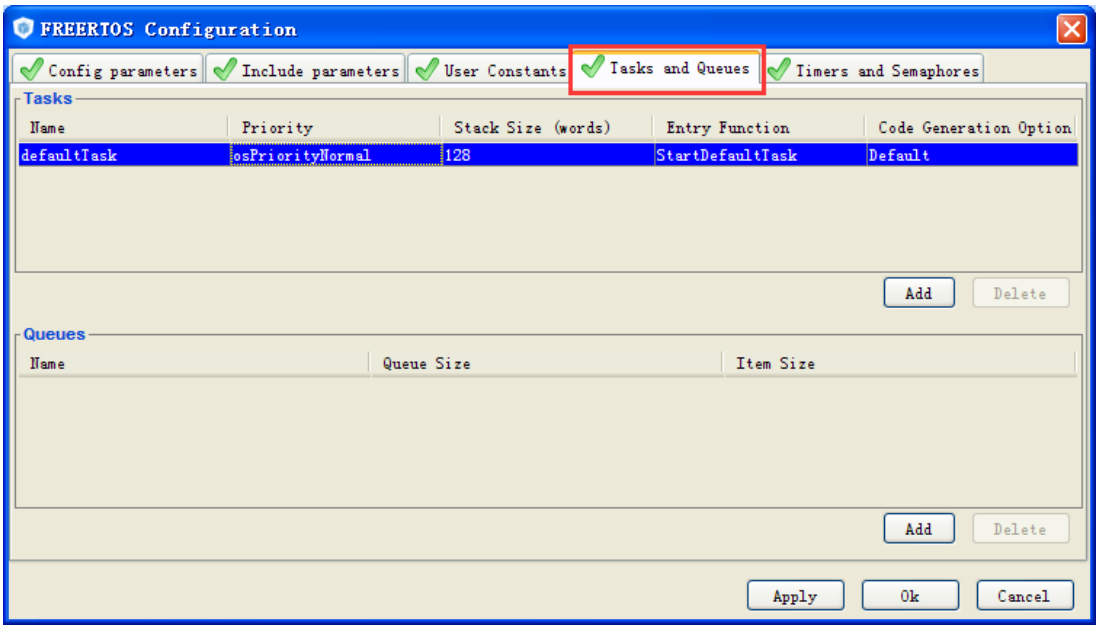
Step6.配置时钟树。8M 输入时，通过 PLL 得到 72M 内部时钟。



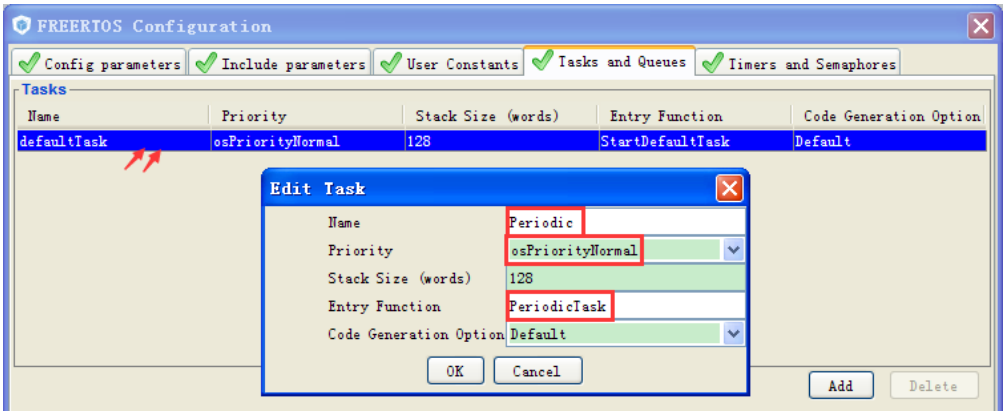
Step7.配置 FreeRTOS。



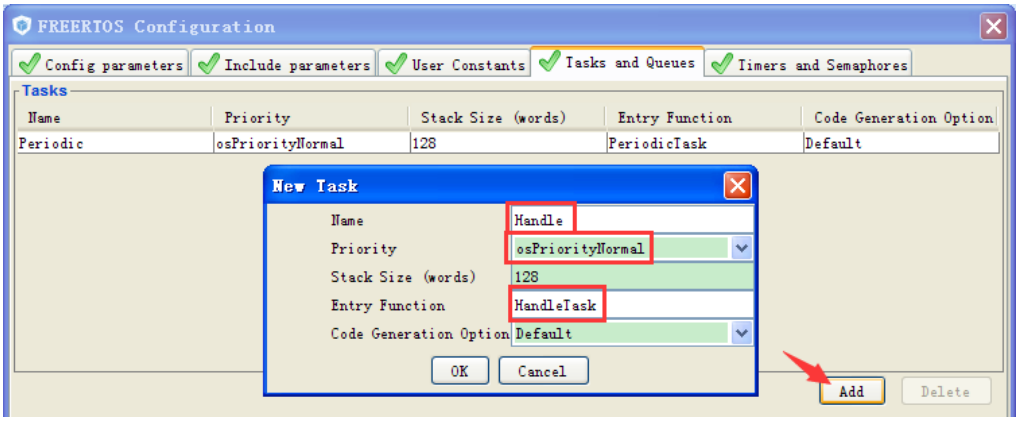
在 Tasks and Queues 选项卡中，默认配置了一个名为 defaultTask 的任务，其优先级为普通，任务堆栈大小为 128 字，任务函数名为 StartDefaultTask。



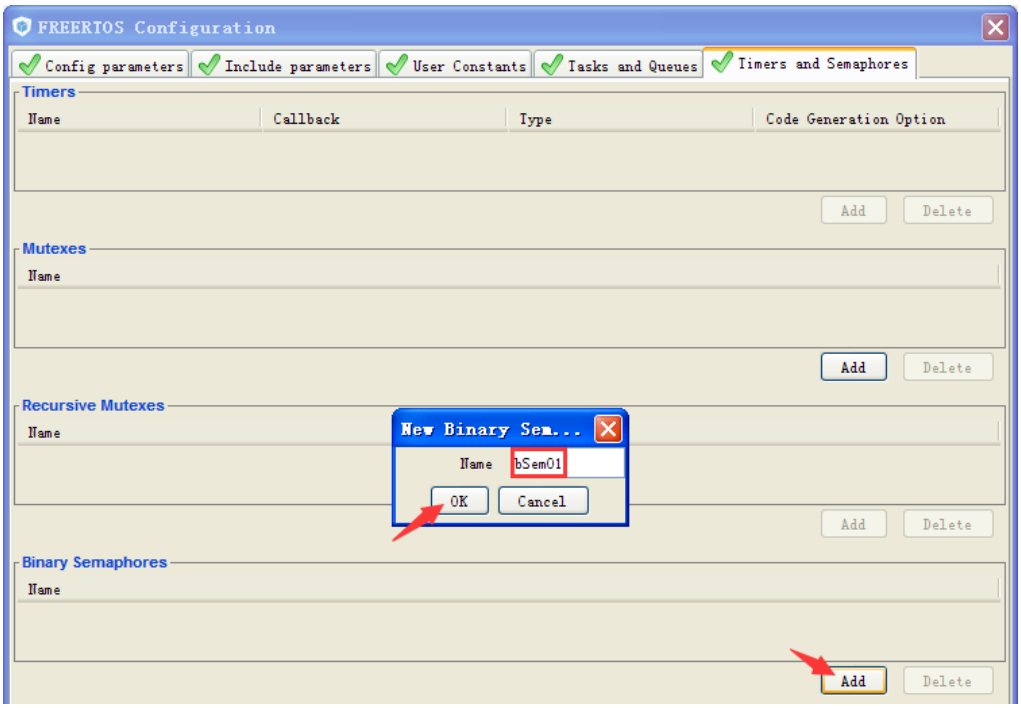
双击蓝色的地方，弹出对话框，将任务名修改为 Periodic，将任务函数名修改为 PeriodicTask。



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

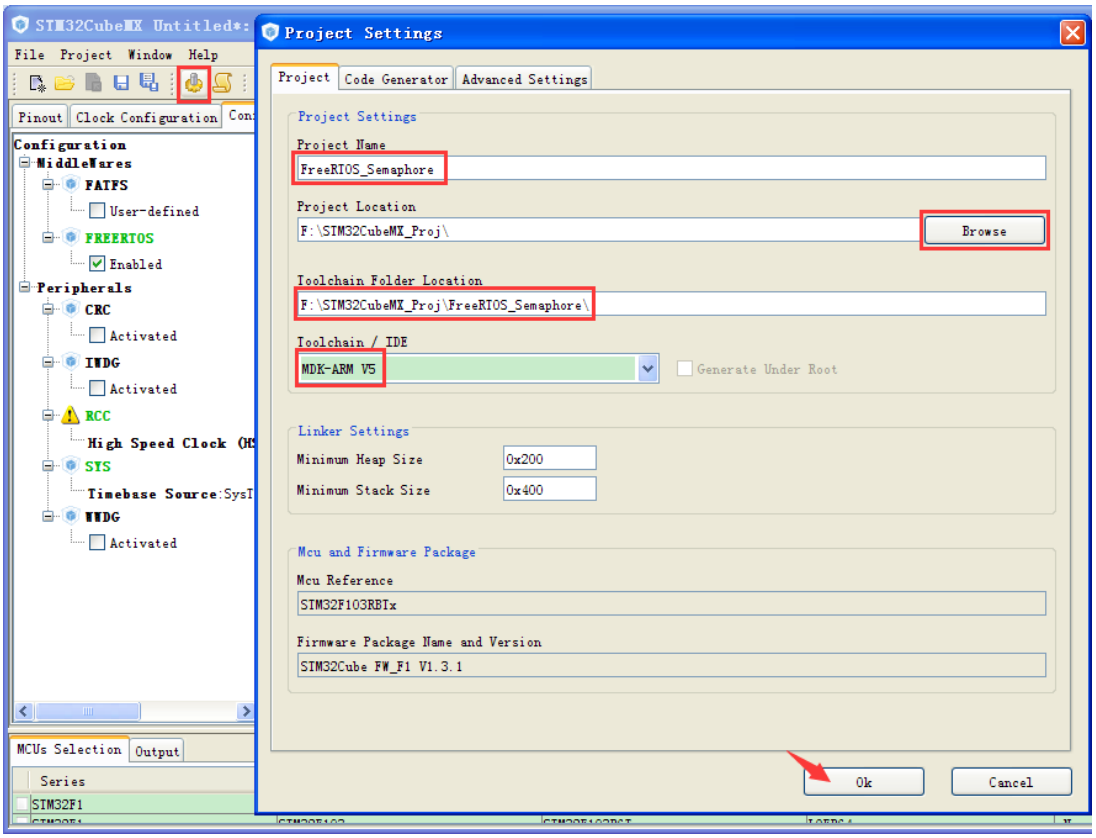


在 Timers and Semaphores 选项卡，点击 Binary Semaphores 项右边的“Add”按钮，添加一个信号量，名称改为 bSem01。

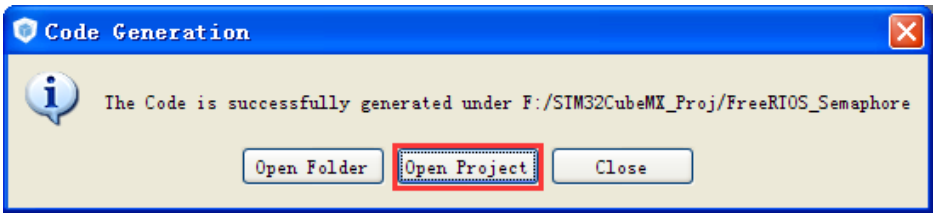


注：该步骤中，除了添加任务和信号量，其他的都使用默认参数。

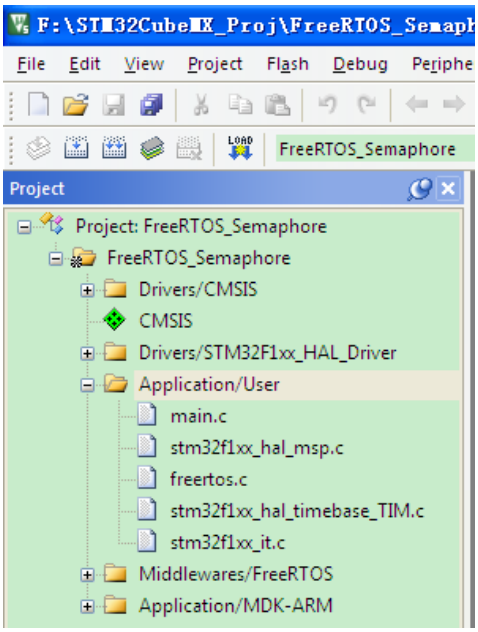
Step8.生成代码。



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。

```
41  /* Private variables ---------------------
42  osThreadId PeriodicHandle;
43  osThreadId HandleHandle;
44  osSemaphoreId bSem01Handle;
45
46  /* USER CODE BEGIN PV */
47  /* Private variables ---------------------
48
49  /* USER CODE END PV */
50
51  /* Private function prototypes -----------
52  void SystemClock_Config(void);
53  static void MX_GPIO_Init(void);
54  void PeriodicTask(void const * argument);
55  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。

```
66  int main(void)
67  {
68    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
69    HAL_Init();
70    /* Configure the system clock */          ①
71    SystemClock_Config();
72    /* Initialize all configured peripherals */
73    MX_GPIO_Init();
74
75    /* Create the semaphores(s) */
76    /* definition and creation of bSem01 */                    ②
77    osSemaphoreDef(bSem01);
78    bSem01Handle = osSemaphoreCreate(osSemaphore(bSem01), 1);
79    /* Create the thread(s) */
80    /* definition and creation of Periodic */
81    osThreadDef(Periodic, PeriodicTask, osPriorityNormal, 0, 128);
82    PeriodicHandle = osThreadCreate(osThread(Periodic), NULL);
83    /* definition and creation of Handle */
84    osThreadDef(Handle, HandleTask, osPriorityNormal, 0, 128);
85    HandleHandle = osThreadCreate(osThread(Handle), NULL);
86    /* Start scheduler */
87    osKernelStart();     ③
88
89    /* We should never get here as control is now taken by the scheduler */
90
91    /* Infinite loop */
92    /* USER CODE BEGIN WHILE */
93    while (1)
94    {
95    }
96  }
```

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。
启动调度器后，程序就由 FreeRTOS 的调度器管理了，后面的 while(1)是不会执行到的。
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。
PeriodicTask 的功能是，每 1 秒钟发送一次信号量。

```
168 /* USER CODE END 4 */
169
170 /* PeriodicTask function */
171 void PeriodicTask(void const * argument)
172 {
173
174   /* USER CODE BEGIN 5 */
175   /* Infinite loop */
176   for(;;)
177   {
178       osDelay(1000);
179       osSemaphoreRelease(bSem01Handle); // 释放(发送)信号量
180   }
181   /* USER CODE END 5 */
182 }
183
```

HandleTask 的功能的，等待信号量，然后翻转 LED0 和 LED1 的输出状态。

```
184 /* HandleTask function */
185 void HandleTask(void const * argument)
186 {
187   /* USER CODE BEGIN HandleTask */
188   /* Infinite loop */
189   for(;;)
190   {
191       osSemaphoreWait(bSem01Handle , osWaitForever);     // 等待信号量
192       HAL_GPIO_TogglePin(LED0_GPIO_Port, LED0_Pin);
193       HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
194   }
195   /* USER CODE END HandleTask */
196 }
```

Step11.编译下载运行。LED0 和 LED1 分别闪烁，周期都是 2 秒。

程序分析：
1.分析语句：osSemaphoreDef(bSem01);
和 osThreadDef(…)类似，osSemaphoreDef(…)并不是一个函数，而是一个宏。
其定义在 cmsis_os.h 文件中，作用是定义一个 osSemaphoreDef_t 结构体。

```
626 #define osSemaphoreDef(name)  \
627 const osSemaphoreDef_t os_semaphore_def_##name = { 0 }
```

2.分析语句：bSem01Handle = osSemaphoreCreate(osSemaphore(bSem01), 1);
同样的，osSemaphore (…)也是一个宏定义，在 cmsis_os.h 文件中可查到。
该语句的作用就是创建一个信号量，最后一个参数=1 时，创建的就是一个二值信号量。

3.HandleTask 任务在执行 osSemaphoreWait(bSem01Handle , osWaitForever);语句后，进入阻塞状态，等待 PeriodicTask 任务释放信号量。在实际运用中，二值信号量主要用于任务同步。

如果把上面的 PeriodicTask 任务换成硬件定时器的 ISR 函数，那么 HandleTask 任务就相当于定时器的延迟处理函数。这是 RTOS 种常用的方法。因为 RTOS 调度本身使用了较低优先级的定时器中断实现，如果在硬件 ISR 中执行过多的代码，就会造成 RTOS 响应受到很大影响。因此，在 ISR 中只发送信号量，然后在其延迟处理函数中实现真正的用户功能。这本质上就是，将任务函数和硬件中断进行同步。

需要注意的一点是，在原生的 FreeRTOS 中，信号量的发送和获取在 ISR 中要使用以 FromISR 结尾的特定函数。在 STM32Cube 生成的代码中，ST 的工程师已经把接口进行了统一，在相应的函数中，通过查询程序状态寄存器判断当前是在 ISR 中还是在普通函数中。所以用户在使用时，统一使用 osSemaphoreWait()和 osSemaphoreRelease()函数操作即可。同样的，FreeRTOS 的其他通信方式如队列、互斥量等，都进行了这样的处理。

S.D.Lu 于 深圳
2016 年 8 月