**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。

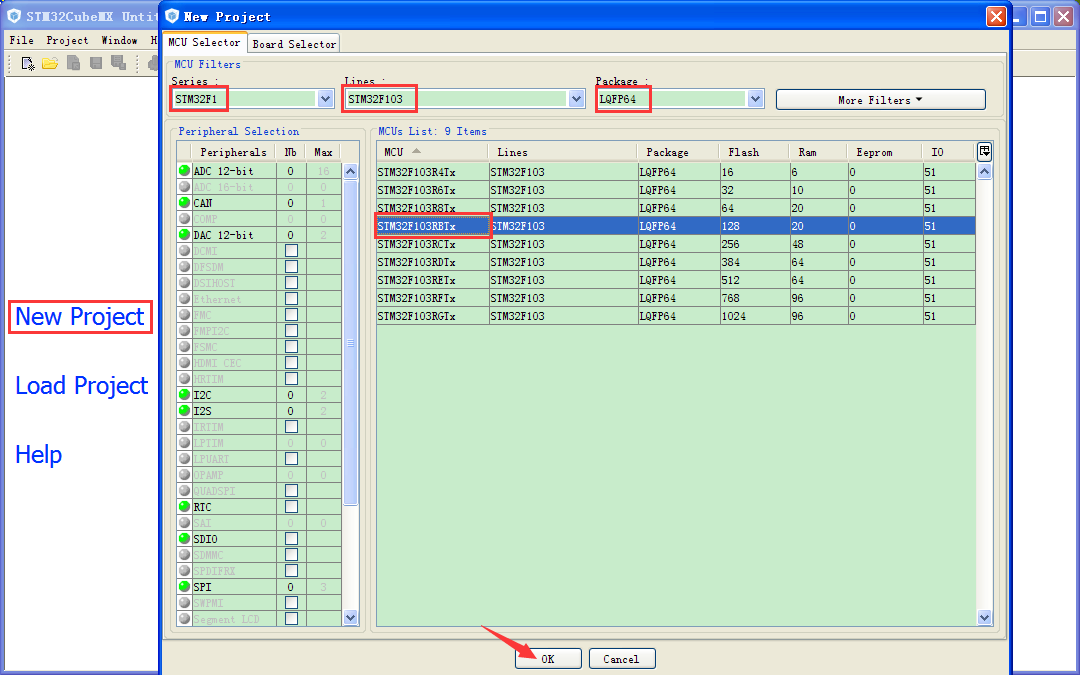
硬件平台：STM32F1xx系列。

目的：学习计数信号量的使用。

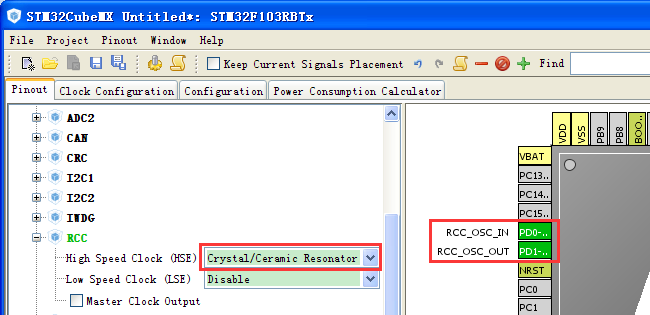
计数信号量的使用场景：一个二值信号量最多只可以锁存一个中断事件。在锁存的事件还未被处理之前，如果还有中断事件发生，那么后续发生的中断事件将会丢失。如果 用计数信号量代替二值信号量，那么，这种丢中断的情形将可以避免。

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。

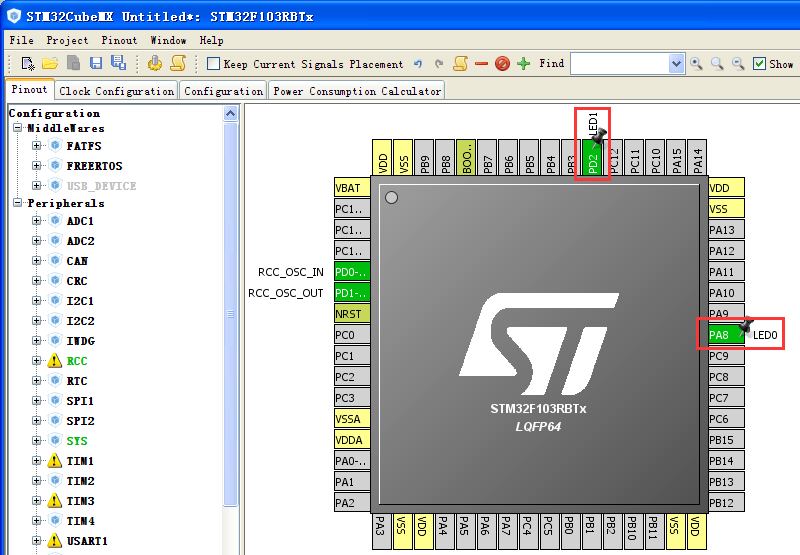
Step1.打开STM32CubeMX，点击“New Project”，选择芯片型号，STM32F103RBTx。



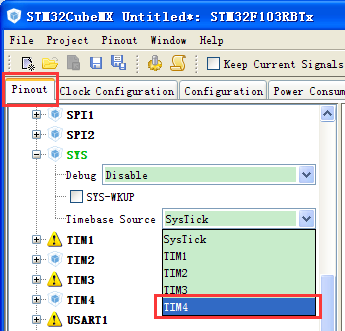
Step2.配置时钟引脚。



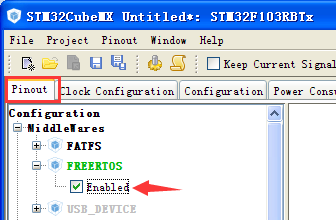
Step3.配置PA8和PD2为Output，并把用户标签分别改为LED0，LED1。



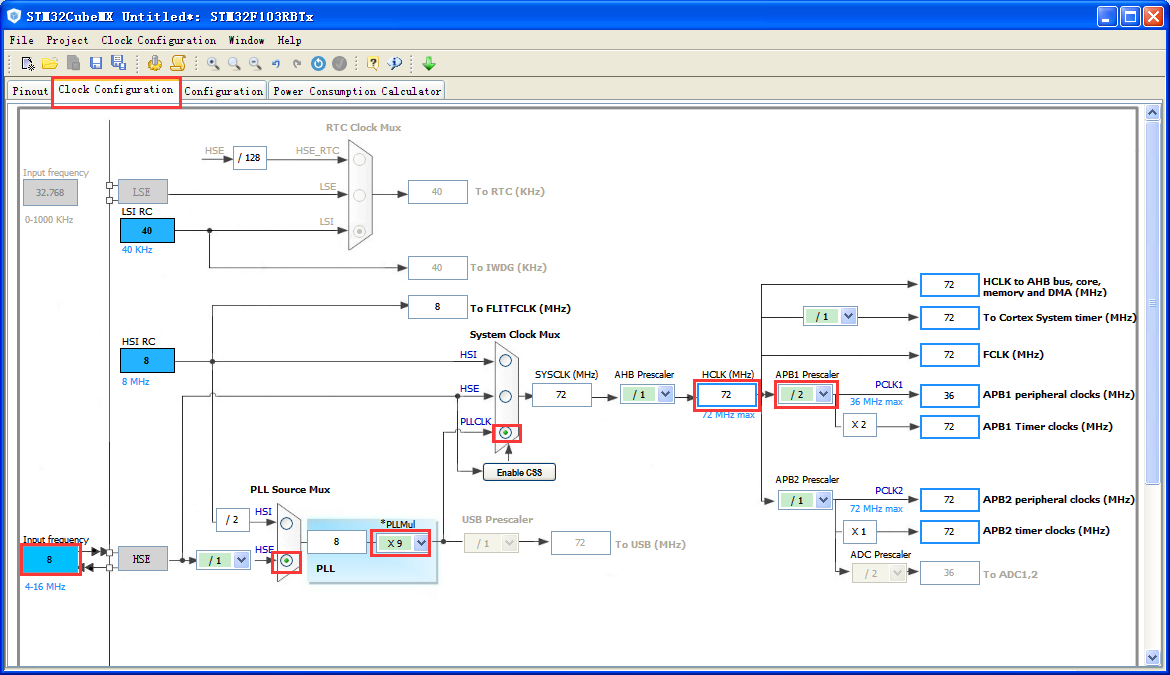
Step4.将系统时基源改为TIM4。



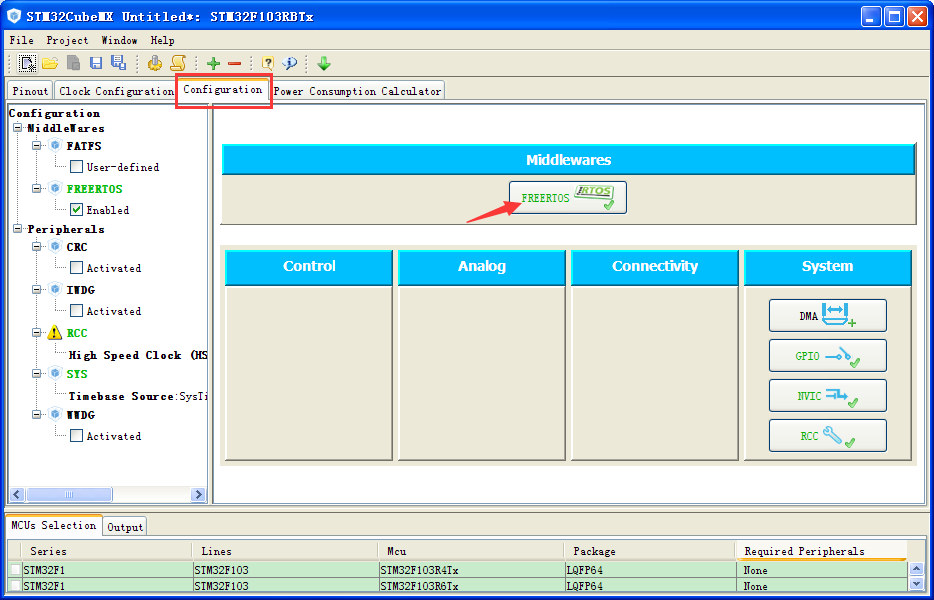
Step5.使能FreeRTOS。



Step6.配置时钟树。8M输入时，通过PLL得到72M内部时钟。

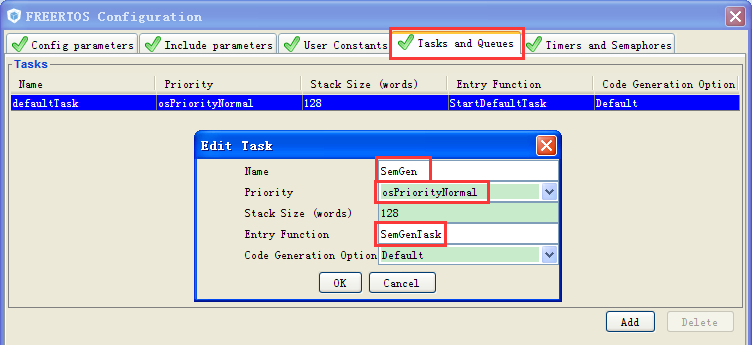


Step7.配置FreeRTOS。

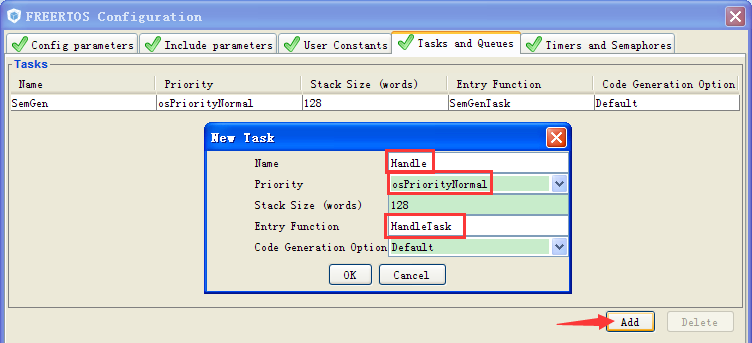


在Tasks and Queues选项卡，默认配置了一个名为defaultTask的任务，其优先级为普通，任务堆栈大小为128字，任务函数名为StartDefaultTask。

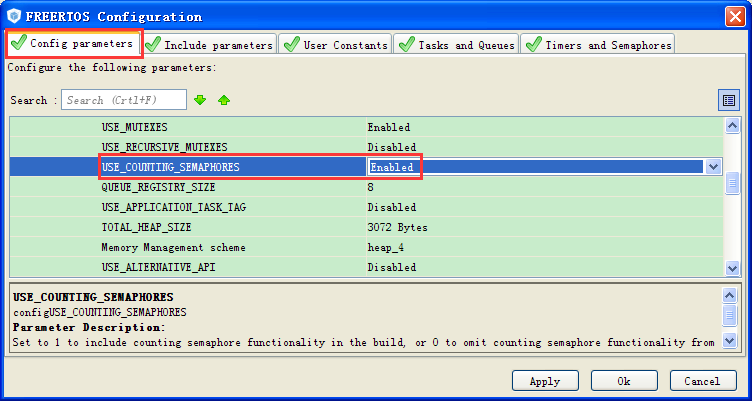
双击蓝色的地方，弹出对话框，将任务名修改为SemGen，将任务函数名修改为SemGenTask。



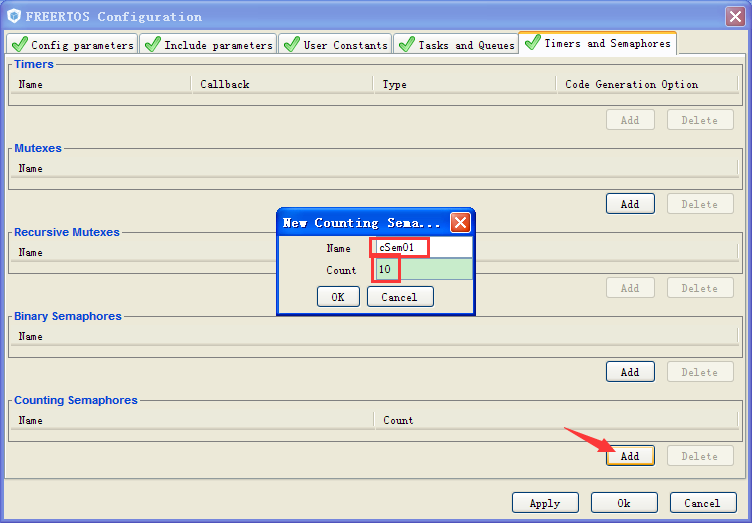
点击Add按钮，增加一个任务Handle，优先级设置为Normal，函数名为HandleTask。



在Config parameters选项卡，使能计数信号量。

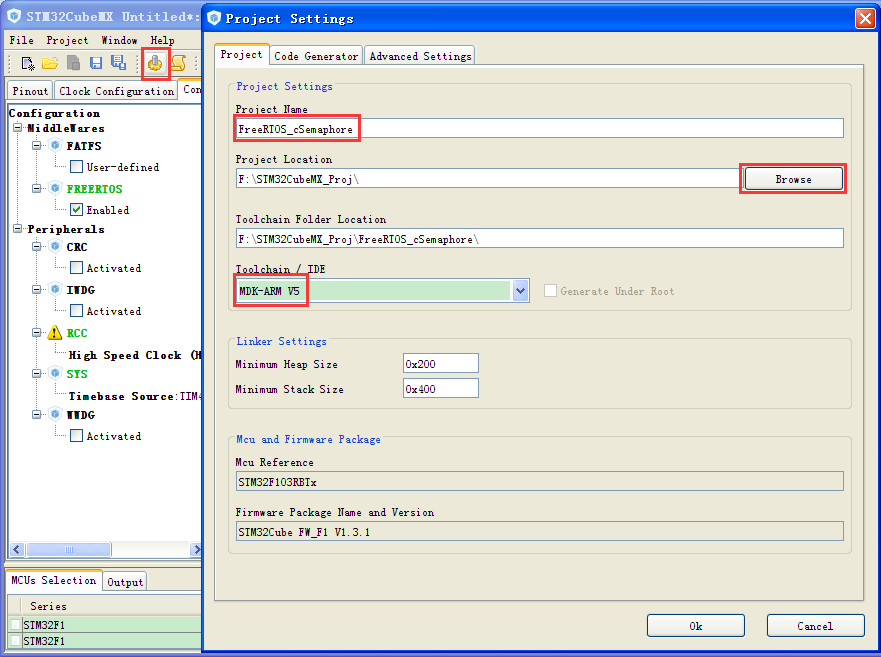


在Timers and Semaphores选项卡，点击Counting Semaphores项右边的“Add”按钮，添加一个信号量，名称改为cSem01，并把最大计数值改为10。

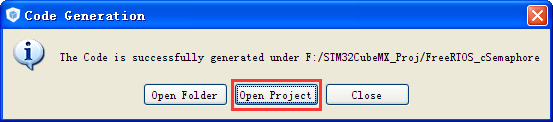


注：其他的都使用默认参数。

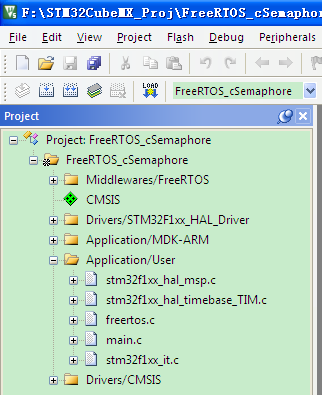
Step8.生成代码。



等完成后直接打开工程。

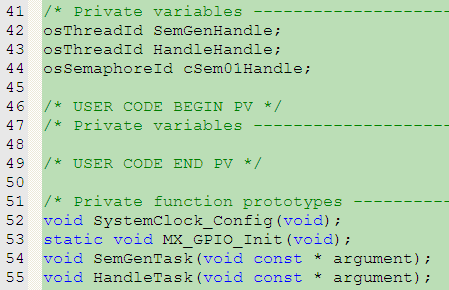


工程基本组织结构如下图，其中Application/User组中的文件是用户可以修改的，而其他组中的文件一般不进行修改。

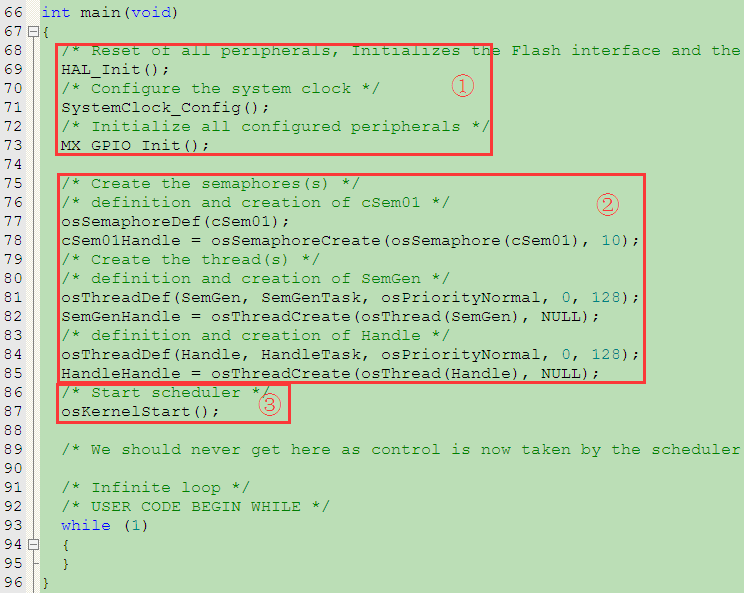


Step9.分析程序结构。

在进入main函数之前，先定义了几个变量，声明了几个函数。



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。

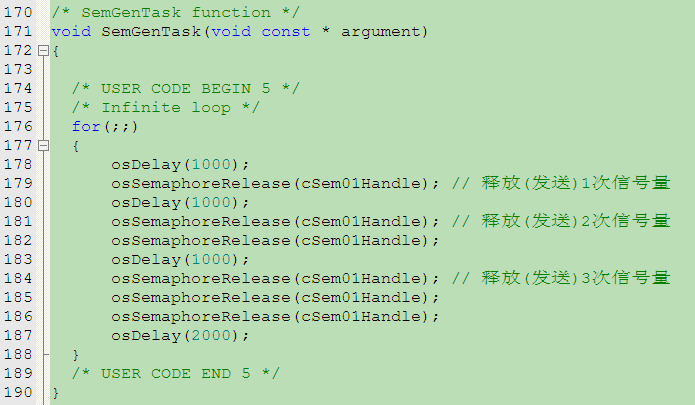


其中第①部分，是硬件配置；第②部分，创建一个信号量和两个任务；第③部分，启动调度器。

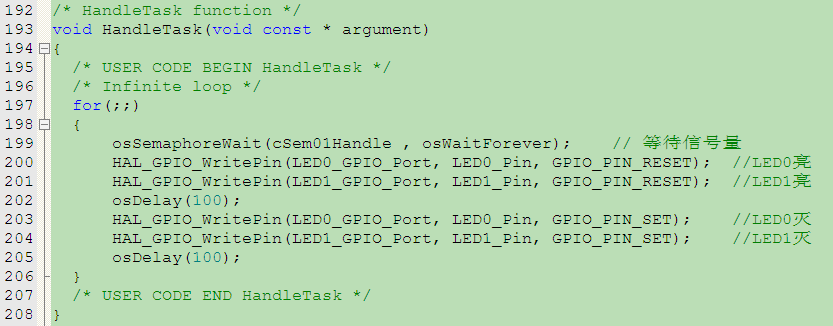
Step10.添加代码。

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。



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



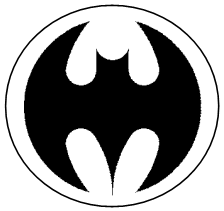
Step11.编译下载运行。现象是，LED闪1次，一秒后闪2次，再过一秒闪3次，再等三秒，LED闪1次…如此循环。

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