**写在前面的话**

本系列笔记一共七篇，是我个人学习FreeRTOS的实验笔记。

学习过程中写笔记有几个好处：一是可以加深自己对FreeRTOS的理解；二是使学习更有成就感。笔记可以作为自己学习进步和知识储备的凭证，当然没人去查，关键是自己真的从中有更多的收获。

在开始学习FreeRTOS时，我就已经计划写成笔记并上传带百度文库，希望对初学者有所帮助。因为我的学习历程也非常依赖网络资源。

本人在学习FreeRTOS之前，已经学过μC/OS II，也上传了几篇学习笔记。这两个系统非常相似，都是开源的RTOS，但是一个是免费的，另一个是收费的。

笔记的主要内容就是学习FreeRTOS的各种通讯机制。

笔记的结构非常简单，就是通过简单的实例，演示FreeRTOS的各种通讯机制的使用方法。

跟随本笔记学习完，能够做到以下几点即可：

1.了解FreeRTOS程序的基本架构；

2.能够理解和应用信号量、消息队列、邮箱队列等相关知识。

特别说明：

本笔记以STM32为平台，任何STM32平台都可以。所有例程只用到简单的硬件资源：最小系统的资源，LED输出，UART输出。

为了开发简单，本笔记的例程全部使用STM32Cube配置生成，只需要添加很少的代码。如果不熟悉STM32Cube的使用，也没关系。只要在网上下载安装STM32CubeMX和相应芯片的支持包，然后跟着笔记的步骤操作即可，该笔记没有省略任何步骤。

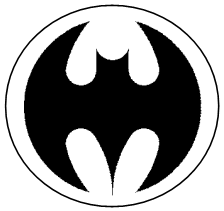
要学习STM32Cube，可到ST社区论坛http://www.stmcu.org/module/forum/forum.php，搜索STM32Cube，即可查看相关帖子。其中比较详细和全面的是微雪电子发布的帖子。

重要参考资料：

FreeRTOS实时内核实用指南.pdf (由Zou Changjun翻译并分享)，建议学习者先通读一遍该文档，这是翻译自FreeRTOS作者Richard Barry于2009年发布的手册。

最新最详细的资料当然是官网www.freertos.org发布的信息。

由于本人水平有限，错漏难免，欢迎指正，谢谢！



S.D.Lu 于 深圳

2016年8月

E-mail:547068172@qq.com

**FreeRTOS学习之一：任务的创建**

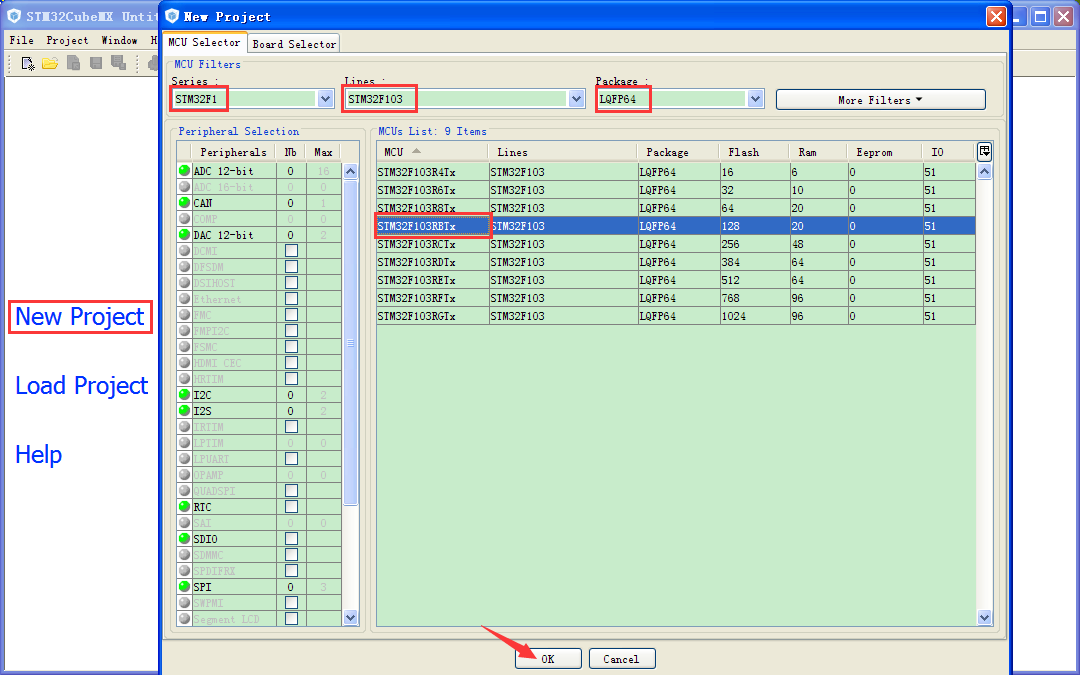
前提：默认已经装好MDK V5和STM32CubeMX，并安装了STM32F1xx系列的支持包。

硬件平台：STM32F1xx系列。

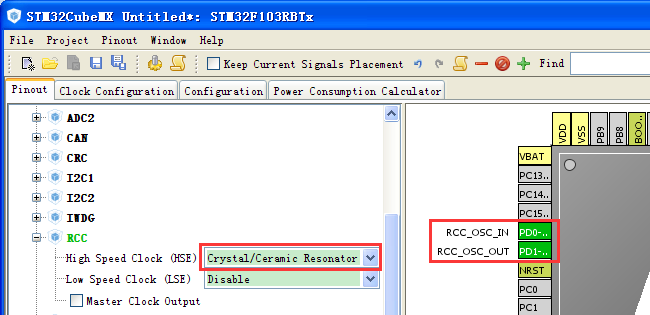
目的：学习FreeRTOS任务的创建。

创建任务是使用FreeRTOS的必要步骤，本文通过实例描述怎样使用STM32CubeMX配置创建FreeRTOS的任务。本文例子将创建两个任务，每个任务分别控制一个LED的闪烁。

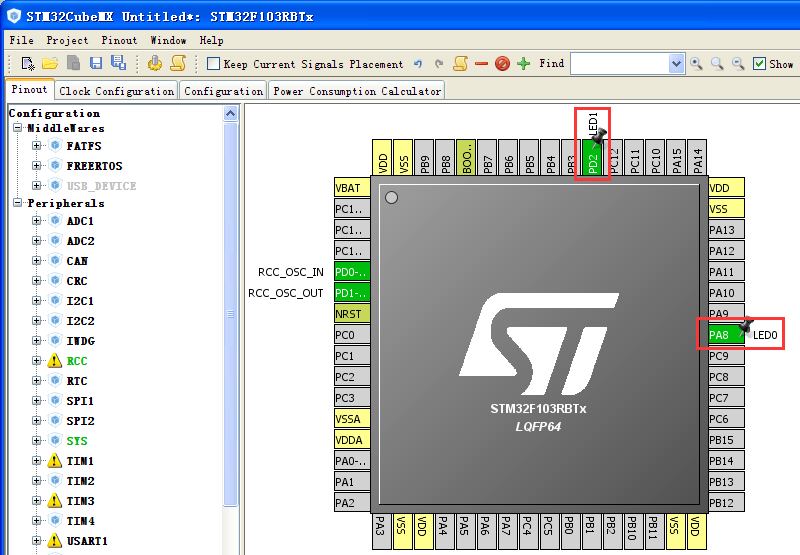
Step1.打开STM32CubeMX，点击“New Project”，选择芯片型号，STM32F103RBTx。



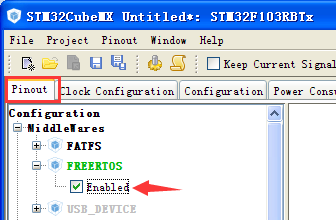
Step2.配置时钟引脚。



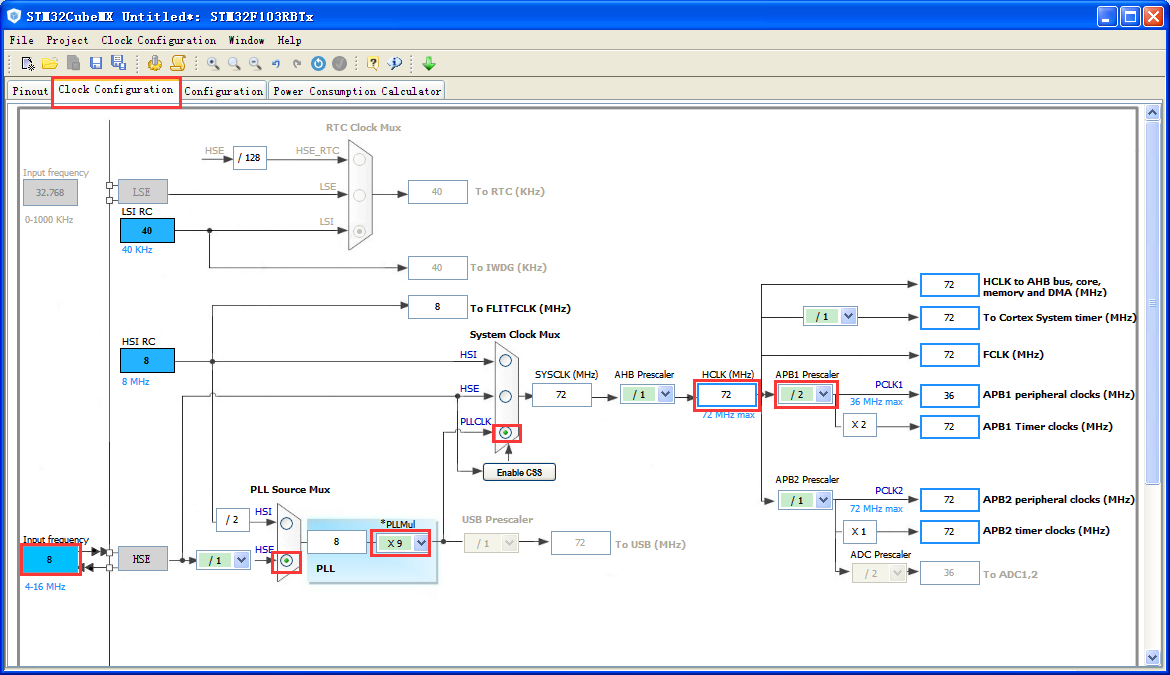
Step3.配置PA8和PD2为Output，并把用户标签分别改为LED0，LED1。



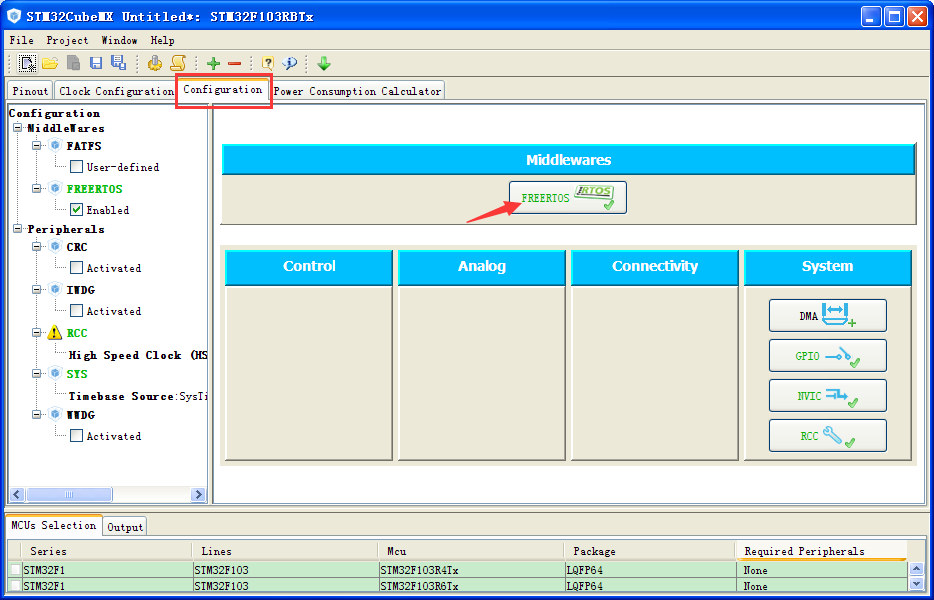
Step4.使能FreeRTOS。



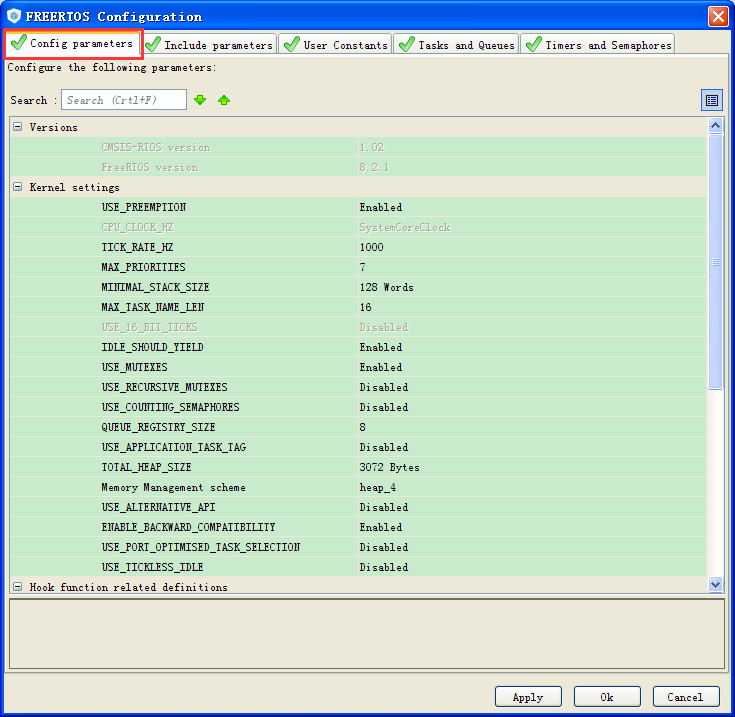
Step5.配置时钟树。8M输入时，通过PLL得到72M内部时钟。



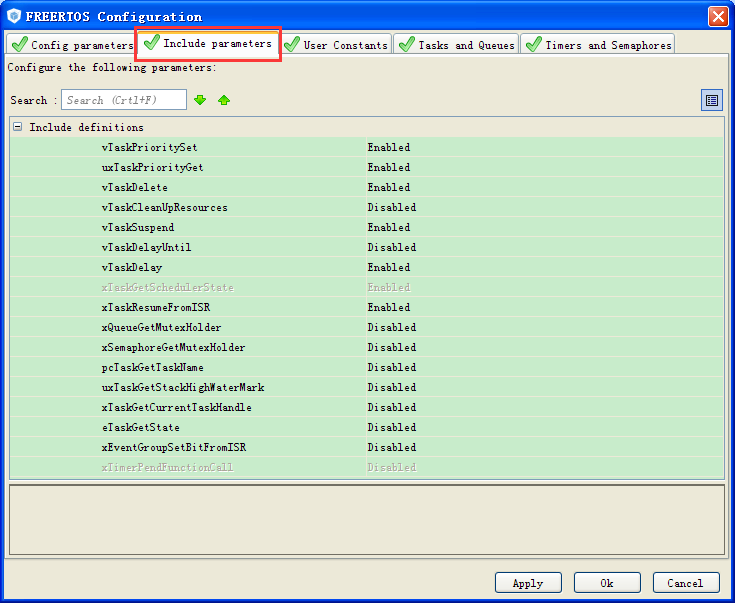
Step6.配置FreeRTOS。



Config parameters选项卡中是配置参数，其中列出了FreeRTOS的可配置参数，对应于FreeRTOSConfig.h文件中的配置参数。

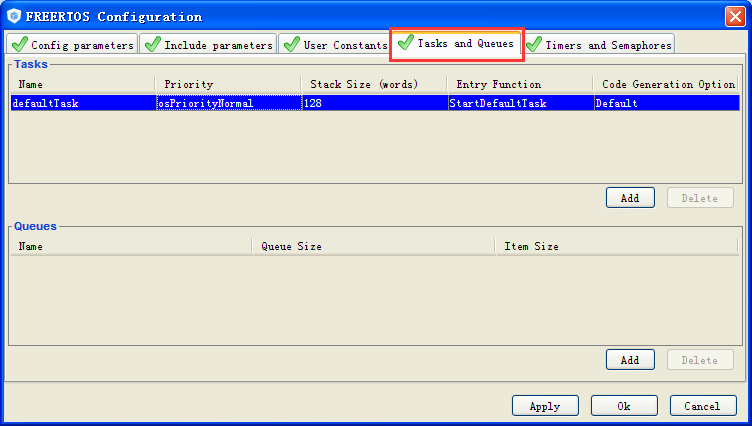


Include parameters选项卡的参数则是用来配置裁剪FreeRTOS的。

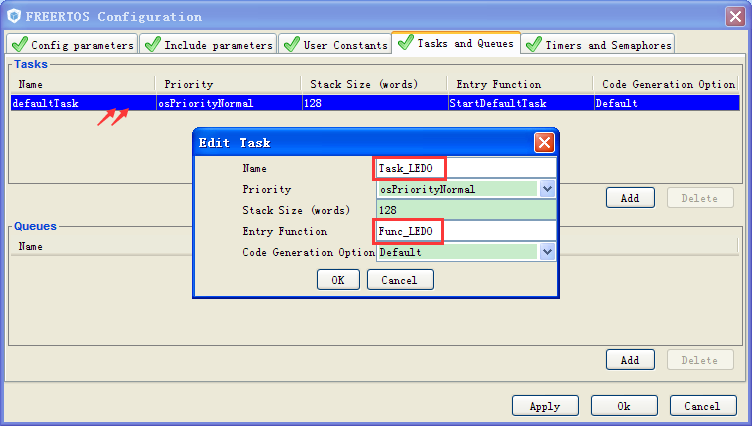


Tasks and Queues用于添加任务和队列。

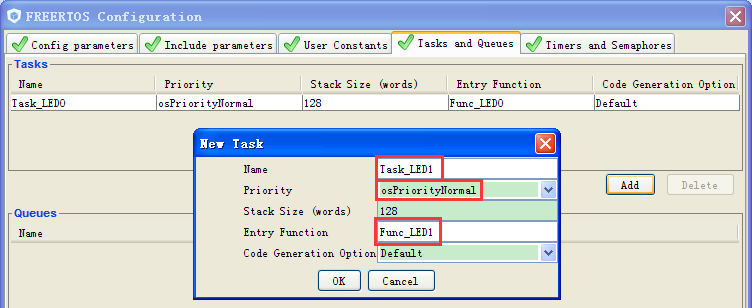
默认配置了一个名为defaultTask的任务，其优先级为普通，任务堆栈大小为128字，任务函数名为StartDefaultTask。



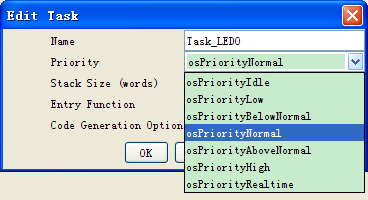
双击蓝色的地方，弹出对话框，将任务名修改为Task\_LED0，将任务函数名修改为Func\_LED0。



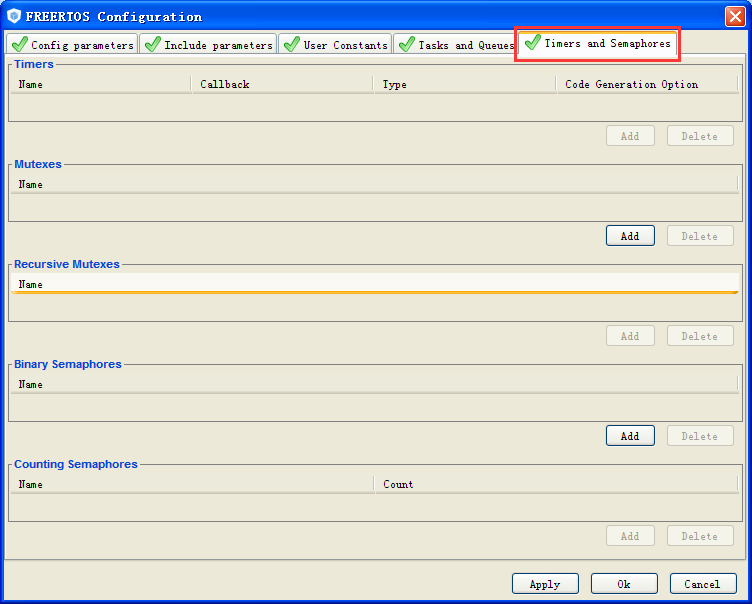
点击Add按钮，增加一个任务Task\_LED1，优先级设置为Normal，函数名为Func\_LED1。



需要注意的是，STM32Cube对FreeRTOS进行了一些修改，比如优先级只有7个，如下图。

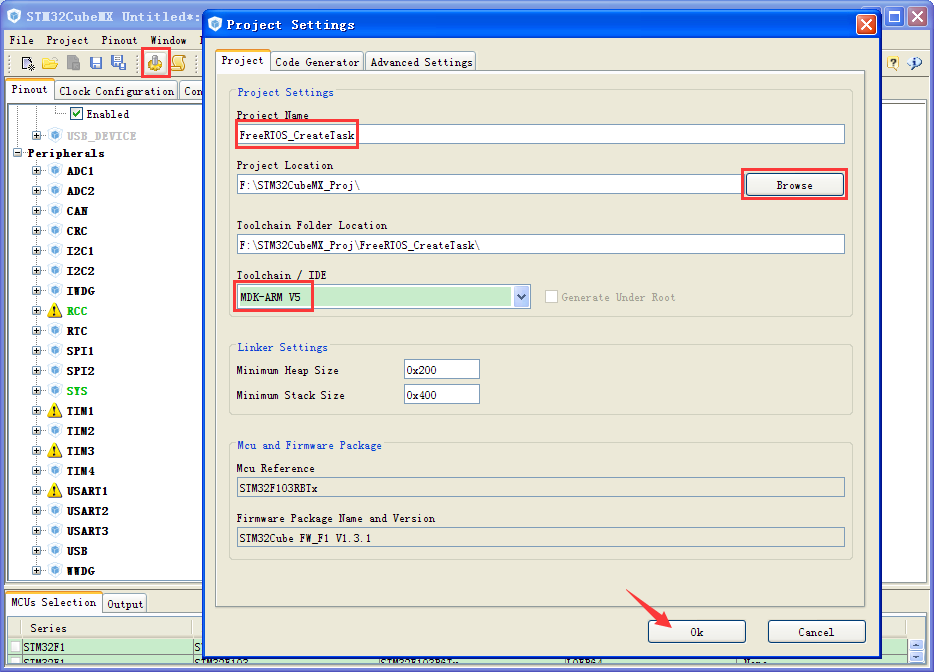


Timers and Semaphores是添加软件定时器和信号量的选项。

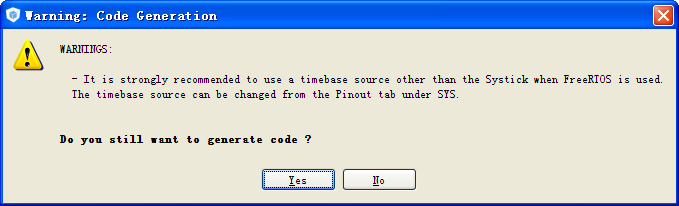


注：该步骤中，除了添加任务，其他的都使用默认参数。

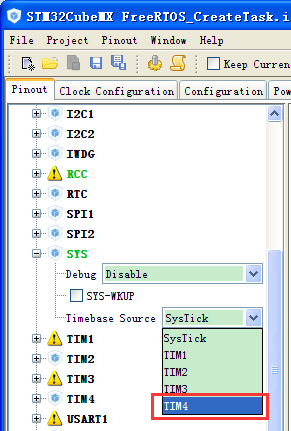
Step7.生成代码。



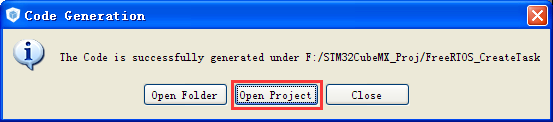
这时候会弹出一个警告。原因是FreeRTSO使用了Systick作为时钟节拍，而HAL库也使用了Systick作为HAL\_Delay()和各种timeout的时钟基准。因此需要将HAL的时钟基准改为其他TIMER。一般使用一个基本定时器。



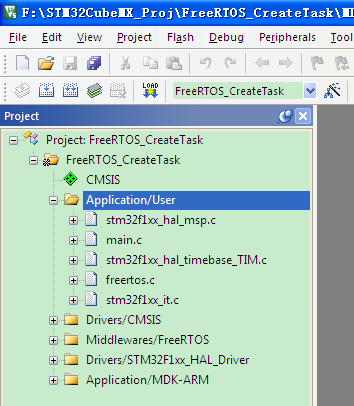
点击“No”按钮，然后在Pinout设置页面选择时基源为TIM4



再次点击代码生成按钮，等完成后直接打开工程。

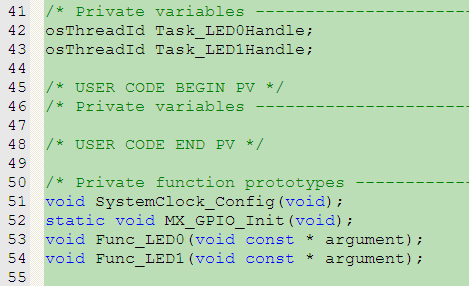


工程基本组织结构如下图，其中Application/User组中的文件是用户可以修改的，而其他组中的文件一般不进行修改。

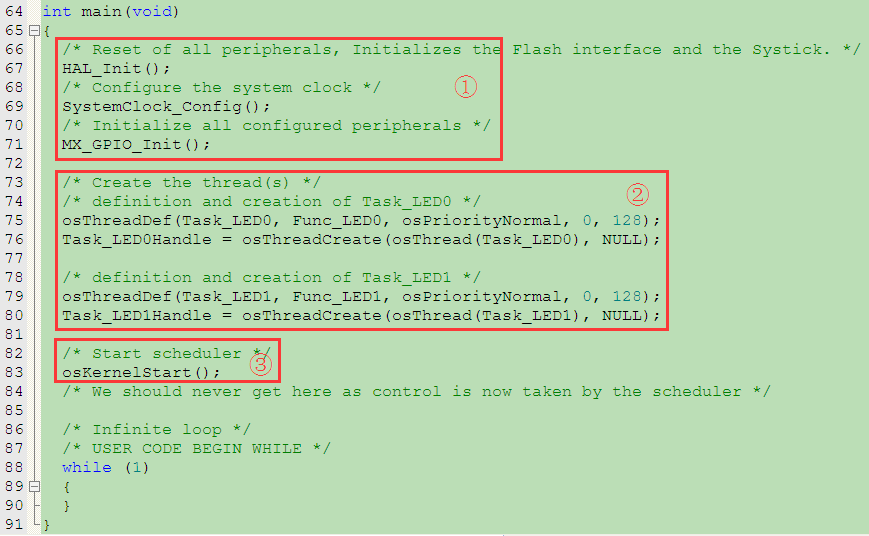


Step8.分析程序结构。

在进入main函数之前，先定义了两个变量，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。

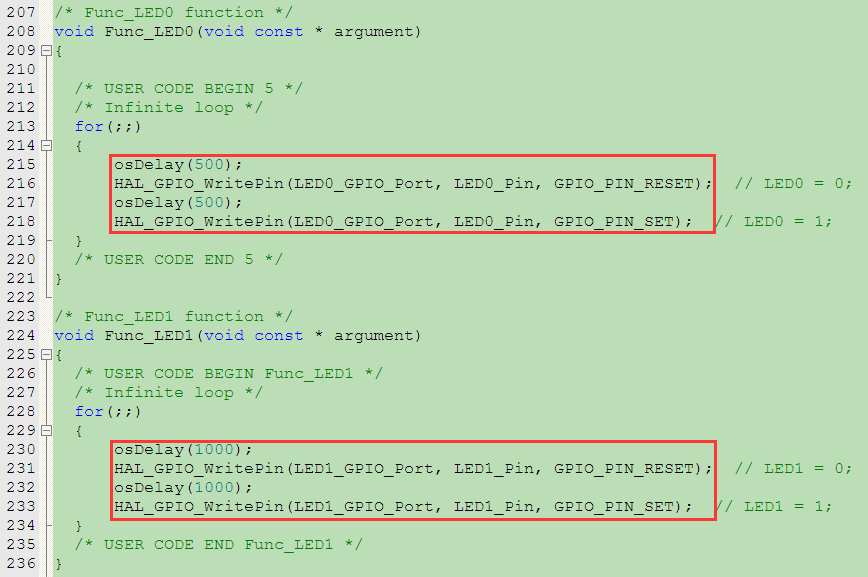
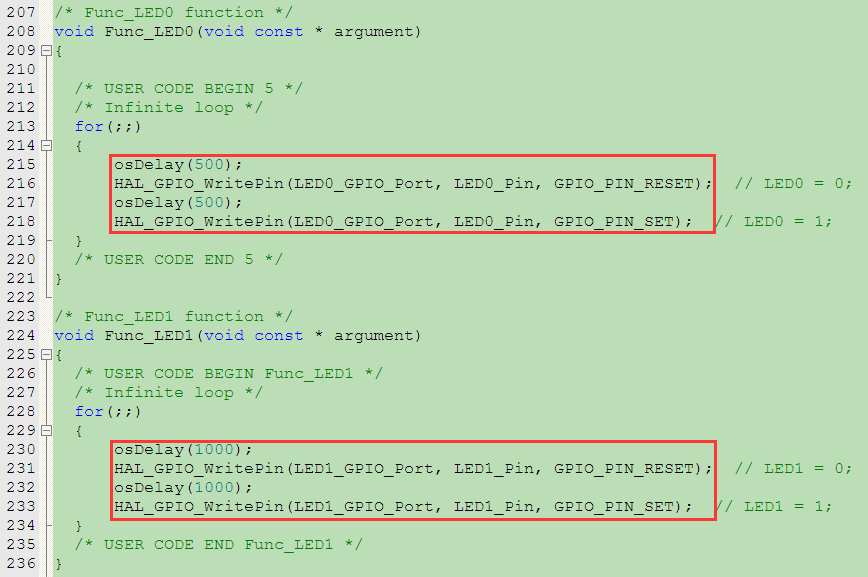


其中第①部分，是硬件配置；第②部分，创建两个线程(或称任务)；第③部分，启动调度器。这就是程序的基本结构。

启动调度器后，程序就由FreeRTOS的调度器管理了，将会被执行的是两个已经创建的任务函数Func\_LED0和Func\_LED1，后面的while(1)是不会执行到的。

Step9.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，Func\_LED0和Func\_LED1，然后在里面分别添加LED0和LED1的控制代码。



Step10.编译下载运行。LED0和LED1分别闪烁，LED0闪烁周期是1秒，LED1的周期是2秒。

程序分析：

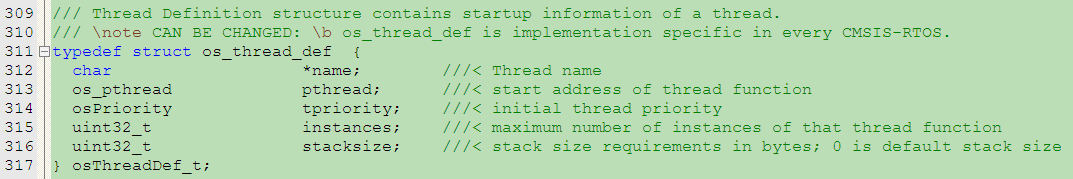
1.分析语句：osThreadDef(Task\_LED0, Func\_LED0, osPriorityNormal, 0, 128);

osThreadDef(…)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osThreadDef\_t结构体。



在cmsis\_os.h文件中，osThreadDef\_t结构体的定义如下：



因此，将osThreadDef(Task\_LED0, Func\_LED0, osPriorityNormal, 0, 128);展开

结果就是

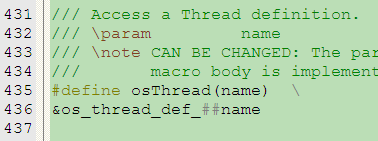
const osThreadDef\_t os\_thread\_def\_Task\_LED0 =

{ Task\_LED0, (Func\_LED0), (osPriorityNormal), (0), (128) };

即，定义了一个名为os\_thread\_def\_Task\_LED0的osThreadDef\_t类型结构体，并赋值给各个成员变量。

2.分析语句：Task\_LED0Handle = osThreadCreate(osThread(Task\_LED0), NULL);

同样的，osThread(…)也是一个宏定义，在cmsis\_os.h文件中可查到。



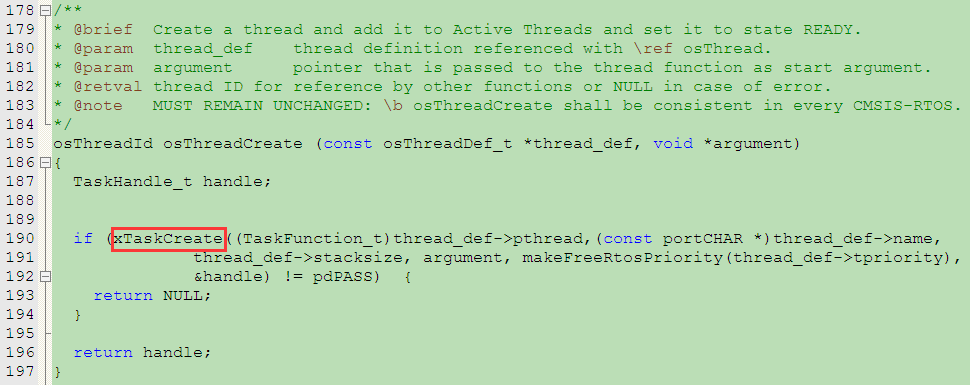
osThread(Task\_LED0)展开的结果就是 &os\_thread\_def\_ Task\_LED0。

因此，将Task\_LED0Handle = osThreadCreate(osThread(Task\_LED0), NULL);展开

结果就是 Task\_LED0Handle = osThreadCreate(&os\_thread\_def\_Task\_LED0, NULL);

所以上面分析的两句话，其过程就是定义一个结构体变量，然后将结构体作为参数传递给osThreadCreate()函数，创建一个任务。

3.分析osThreadCreate()函数。



查看其源码，可以发现，这个函数实际上调用了xTaskCreate()函数，这才是原生FreeRTOS的API函数。

STM32CubeMX的工程师根据CMSIC接口标准对FreeRTOS的API函数进行了二次封装，使用户开发更加容易。封装后的函数接口都放在cmsis\_os.h文件中。

其实，在开发过程中，不需要像上面的分析过程那样，将函数或者宏定义展开进行详细分析。我们知道每个接口参数的意义，并会使用该接口就行了。

附加内容：FreeRTOS任务调度策略探讨

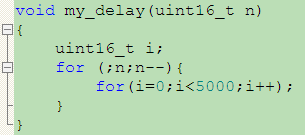
本例中的两个任务函数Func\_LED0和Func\_LED1，他们实际占用CPU的时间很少，在调用osDelay()函数之后，它们就进入阻塞状态了，它们在等待“定时时间到”事件。在用户任务都进入阻塞状态时，运行的是空闲任务。空闲任务是启动调度器时自动创建的。

本例中，两个任务的优先级是一样的，都是osPriorityNormal。但是由于调用了osDelay()函数，它们进入阻塞状态时就让出了CPU的使用权。因此，两个任务看上去就像并行执行的一样。

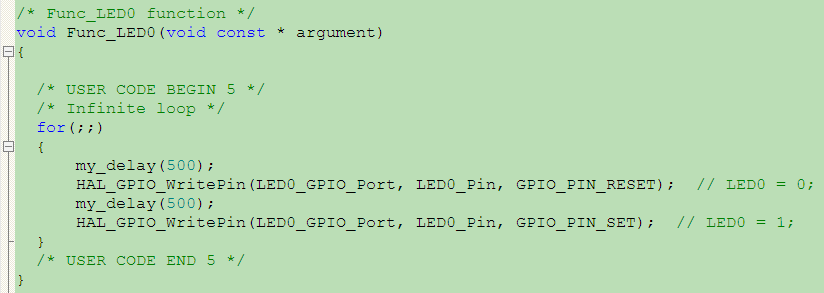
如果把其中的一任务的优先级设置成osPriorityLow或者osPriorityHigh，让两个任务的优先级不同，会怎样呢？结果是，运行起来还是像并行执行的一样。

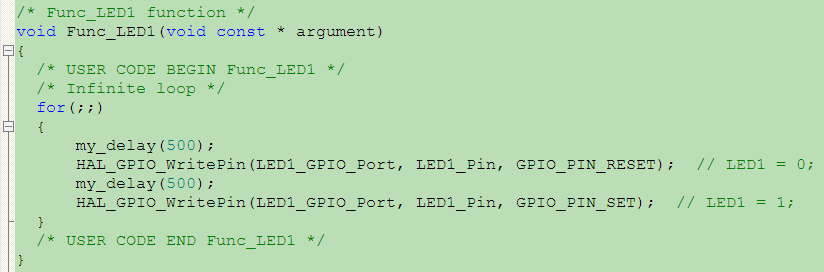
我们都知道，如果两个就绪的任务优先级不同，那么优先级高的任务得到运行。那么，如果两个就绪任务优先级相同，且一直处于就绪状态，那么在FreeRTOS中将如何运行呢？下面将通过实验检验其运行过程。

写一个用for循环实现的延时函数：



然后在任务中替代原来的osDelay()函数函数。这样，两个任务会一直处于可运行的状态，因为他们“总是有事情要做”。

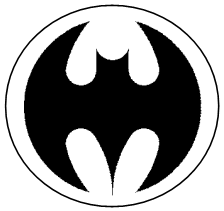




a).如果两个任务的优先级不同，则运行结果是：只有优先级高的任务得到运行。

b).如果两个任务的优先级相同，则运行结果是：两个任务都得到运行，但是LED的闪烁频率比单独运行时的低。按照上述代码，两个任务调用的都是my\_delay(500);，结果就是LED闪烁频率减半，相当于单个任务运行时的my\_delay(1000);。

从b)的情况可知，FreeRTOS在任务优先级相同时，会分配给各任务相同的CPU时间，即任务会轮流执行相等的时间片。



S.D.Lu 于 深圳

2016年8月