FreeRTOS 学习之五：邮箱队列

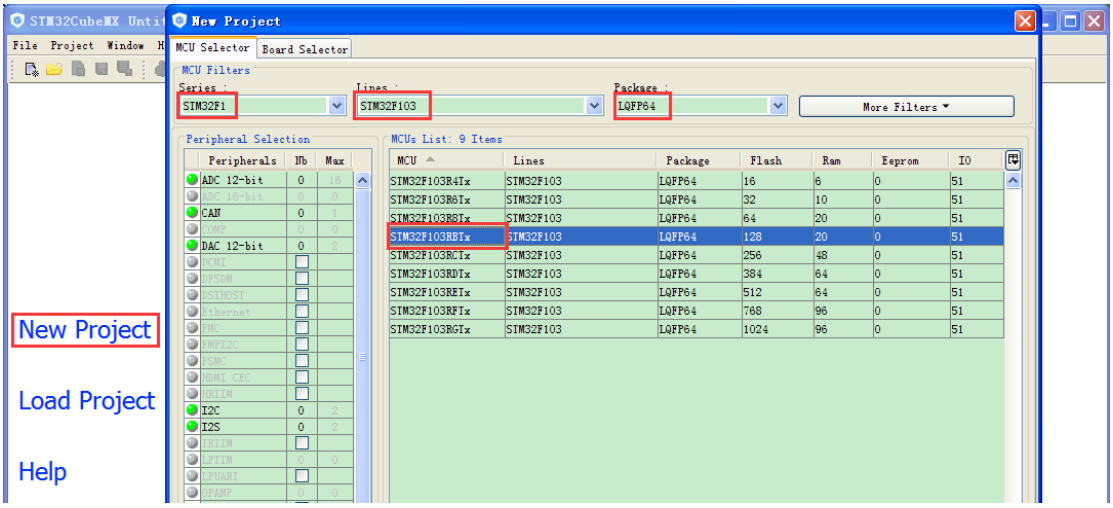
前提：默认已经装好 MDK V5 和 STM32CubeMX，并安装了 STM32F1xx 系列的支持包。
硬件平台：STM32F1xx 系列。
目的：学习邮箱队列的使用。

如果队列存储的数据单元尺寸较大，那最好是利用队列来传递数据的指针而不是对 数据本身在队列上一字节一字节地拷贝进或拷贝出。传递指针无论是在处理速度上还是 内存空间利用上都更有效。这个实现方式就是邮箱队列，它传递的是数据的指针。

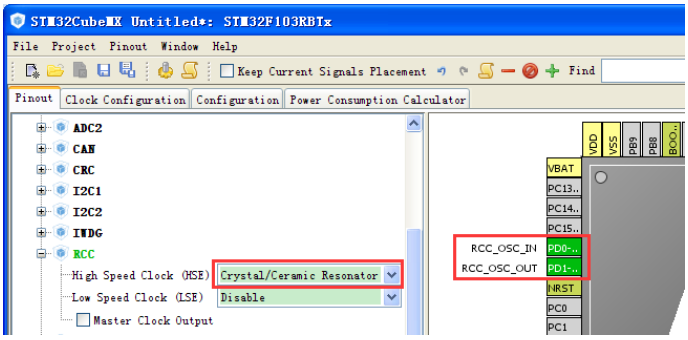
实际上，STM32Cube 所使用的 FreeRTOS 已经在 CMSIS 标准接口文件 cmsis_os.c 实现了邮箱队列。当前 STM32CubeMX 版本 V4.14，还不支持配置生成邮箱队列。但是可以参考 STM32Cube 提供的例程，学习使用邮箱队列。

本文例子使用 STM32CubeMX 配置创建两个任务，一个任务每隔一定时间发送一个消息到邮箱队列，另一个等待邮箱消息并根据消息的内容控制 LED 的闪烁次数。

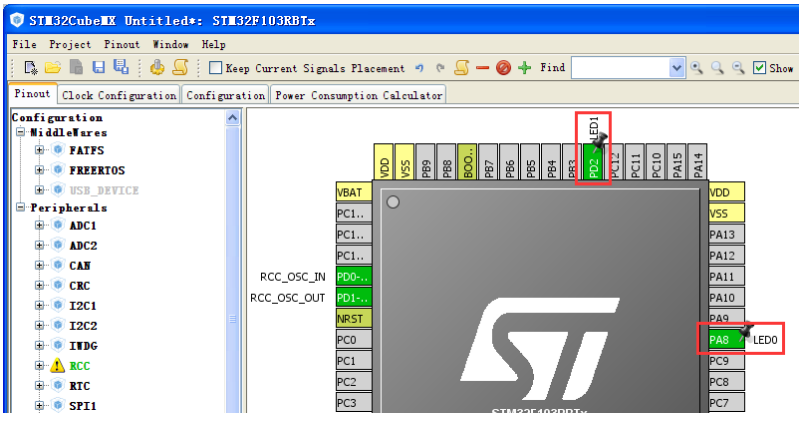
Step1.打开 STM32CubeMX，点击“New Project”，选择芯片型号，STM32F103RBTx。



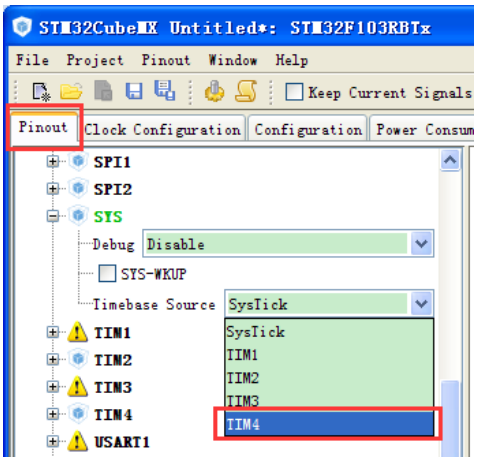
Step2.配置时钟引脚。



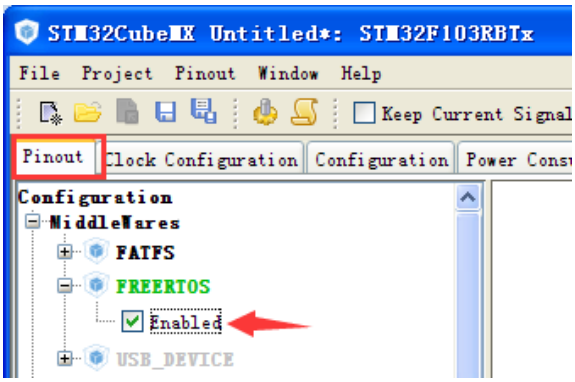
Step3.配置 PA8 和 PD2 为 Output，并把用户标签分别改为 LED0，LED1。



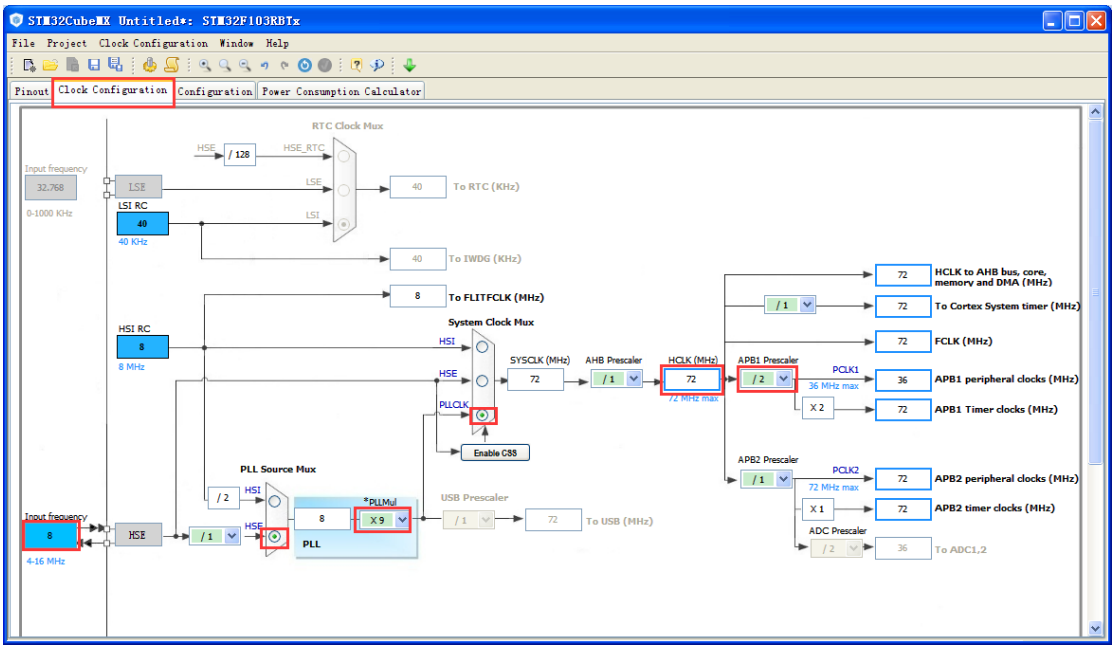
Step4.将系统时基源改为 TIM4。



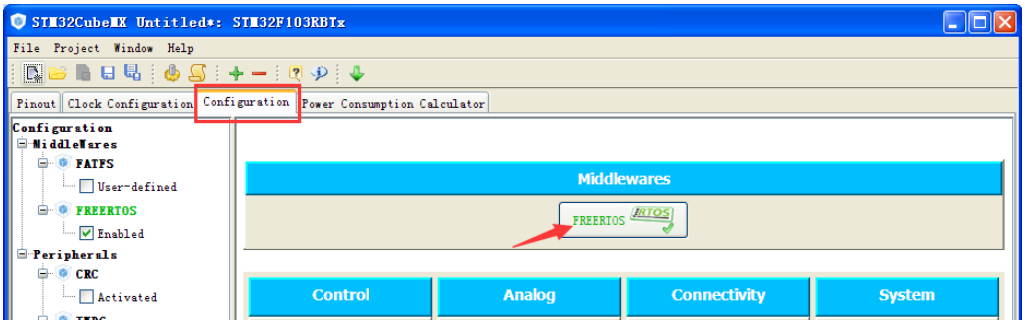
Step5.使能 FreeRTOS。



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

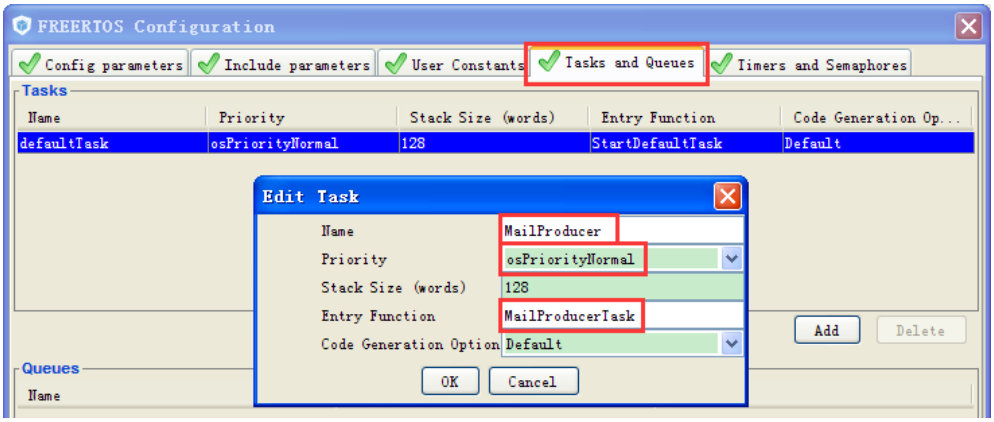


Step7.配置 FreeRTOS。

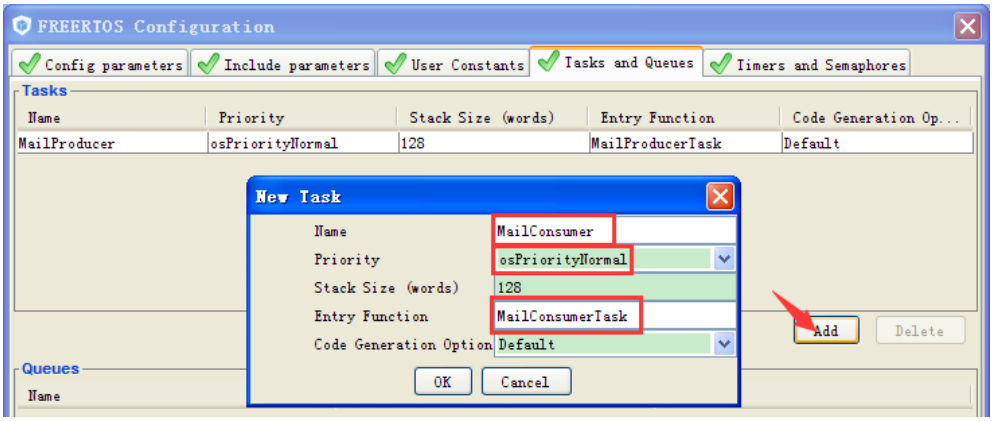


在 Tasks and Queues 选项卡，默认配置了一个名为 defaultTask 的任务，其优先级为普通，任务堆栈大小为 128 字，任务函数名为 StartDefaultTask。

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

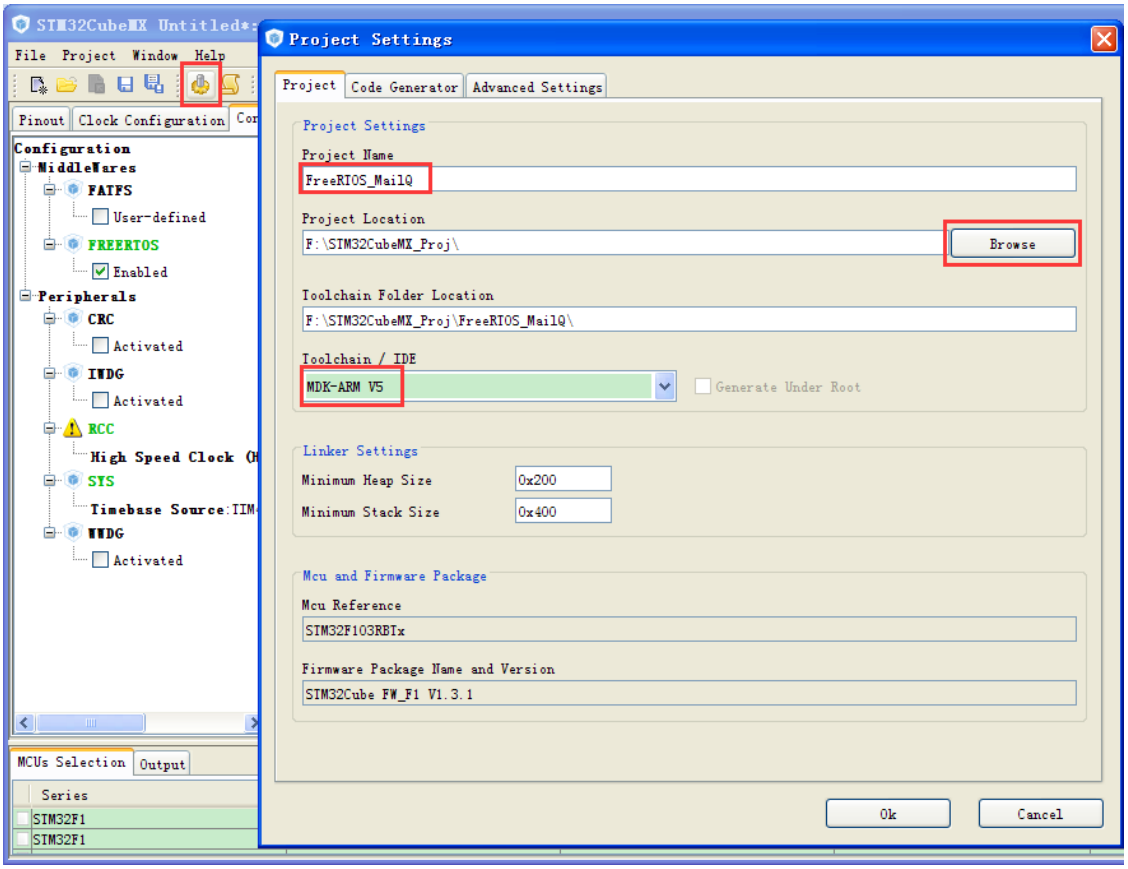


点击 Add 按钮，增加一个任务 MailConsumer，优先级设置为 Normal，函数名为 MailConsumerTask。

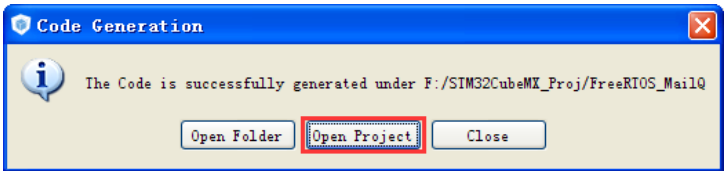


注：其他的都使用默认参数。

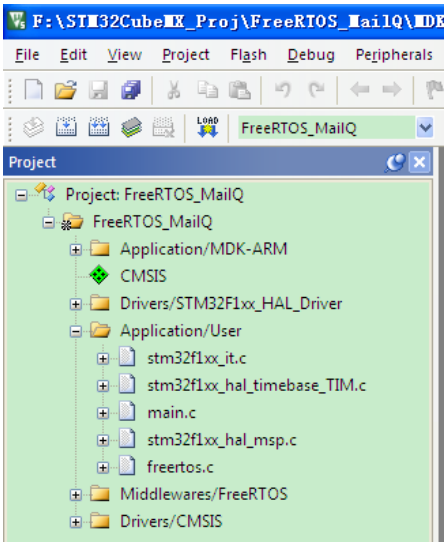
Step8.生成代码。



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。

```
41  /* Private variables ---------------------------
42  osThreadId MailProducerHandle;
43  osThreadId MailConsumerHandle;
44
45  /* USER CODE BEGIN PV */
46  /* Private variables ---------------------------
47
48  /* USER CODE END PV */
49
50  /* Private function prototypes -----------------
51  void SystemClock_Config(void);
52  static void MX_GPIO_Init(void);
53  void MailProducerTask(void const * argument);
54  void MailConsumerTask(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。

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

其中第①部分，是硬件配置；第②部分，创建两个任务；第③部分，启动调度器。

Step10.添加代码。

参考官方的 Mail 例程，添加应用代码。下载 STM32CubeF1 的支持包 stm32cubef1 V1.4.0.zip 解压，Mail 例程路径为 STM32Cube_FW_F1_V1.4.0\Projects\STM3210E_EVAL\Applications\FreeRTOS\FreeRTOS_Mail。

在 main.c 文件中，自定义一个邮箱结构体，和一个 osMailQId 变量(实际上是一个指针)。

```
/* USER CODE BEGIN PV */
/* Private variables -----------------------
typedef struct
{ /* Mail object structure */
  uint32_t var1; /* var1 is a uint32_t */
  uint32_t var2; /* var2 is a uint32_t */
  uint8_t var3; /* var3 is a uint8_t */
} Amail_TypeDef;

osMailQId mailQ01Handle;
```

在 main 文件的/* USER CODE BEGIN 2 */和/* USER CODE END 2 */两个宏之间，创建邮箱队列，队列深度为 15。

```
 /* USER CODE BEGIN 2 */
 /* Create the mail queue used by the two tasks to pass the struct Amail_TypeDef */
 osMailQDef(mailQ01, 15, Amail_TypeDef); /* Define mail queue */
 mailQ01Handle = osMailCreate(osMailQ(mailQ01), NULL); /* create mail queue */
 /* USER CODE END 2 */
```

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。

MailProducerTask 的功能是，发送 1 次消息到邮箱队列，间隔一秒后发送 1 次，再间隔一秒发送 1 次，然后等待 2 秒。

```
/* MailProducerTask function */
void MailProducerTask(void const * argument)
{
  /* USER CODE BEGIN 5 */
  Amail_TypeDef Mail[3];

    Mail[0].var1 = 0x01; /* 设置邮件内容 */
    Mail[0].var2 = 0x02;
    Mail[0].var3 = 0x03;

    Mail[1].var1 = 0x02;
    Mail[1].var2 = 0x03;
    Mail[1].var3 = 0x04;

    Mail[2].var1 = 0x03;
    Mail[2].var2 = 0x04;
    Mail[2].var3 = 0x05;
  /* Infinite loop */
  for(;;)
  {
      osMailPut(mailQ01Handle, &Mail[0]);  // 发送邮件
      osDelay(1000);
      osMailPut(mailQ01Handle, &Mail[1]);  // 发送邮件
      osDelay(1000);
      osMailPut(mailQ01Handle, &Mail[2]);  // 发送邮件
      osDelay(3000);
  }
  /* USER CODE END 5 */
}
```

MailConsumerTask 的功能的，等待邮箱消息，然后根据消息的内容控制 LED0 和 LED1 闪烁次数。

```c
/* MailConsumerTask function */
void MailConsumerTask(void const * argument)
{
  /* USER CODE BEGIN MailConsumerTask */
  osEvent event;
  Amail_TypeDef *pMail;
  uint32_t cnt;
  /* Infinite loop */
  for(;;)
  {
      event = osMailGet(mailQ01Handle, osWaitForever);  // 获取邮件
      if(event.status == osEventMail)
      {
          pMail = event.value.p;     // 获取邮件指针
          cnt = pMail->var2;         // 提取邮件中的数据var2(也可以是其他数据)
          while(cnt--)
          {
              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 MailConsumerTask */
}
```

Step11.编译下载运行。现象是，LED 闪 2 次，一秒后闪 3 次，再过一秒闪 4 次，再等三秒，LED 闪 2 次…如此循环。

程序分析：

和消息队列相比，邮箱队列的效率更高。因为消息队列传递消息时，是把消息的内容拷贝到队列存储空间中，而邮箱队列只传递消息的指针。所以，使用邮箱队列过程中，要保证邮件的内容是有效的，且发送至邮箱队列后到邮件被提取之前，不应该被修改。

S.D.Lu 于 深圳
2016 年 8 月