编者序

很高兴终于完成了 uC/OS-III 嵌入式系统的翻译并移植到 stm32 中，翻译从 2011 年 10 月 15 日开始到 2011 年 11 月 3 日为止，共 20 天时间，平均每天 5 个小时。本想将 uC/OS-III 函数的 API 部分也翻译的，但毕竟考研更甚于爱好，我得为 2013 年 1 月的考研做准备呀~~。

在此，我要感谢：

1、我的导师：乐光学教授。是您经常带我去公司拓展视野，并让我坚定不移地往嵌入式方面发展。

2、我的师傅：张雪强博士。是您无偿提供给我一些开发板，作为回报，帮您的店铺宣传一下 http://as-robot.taobao.com/

3、还有我的亲朋好友们。

今天晚上，我将移植的步骤也分享给大家。

我的 QQ 号码是 522430192，我的邮箱是 522430192@qq.com，希望大家多多联系我，共同学习，共同进步。

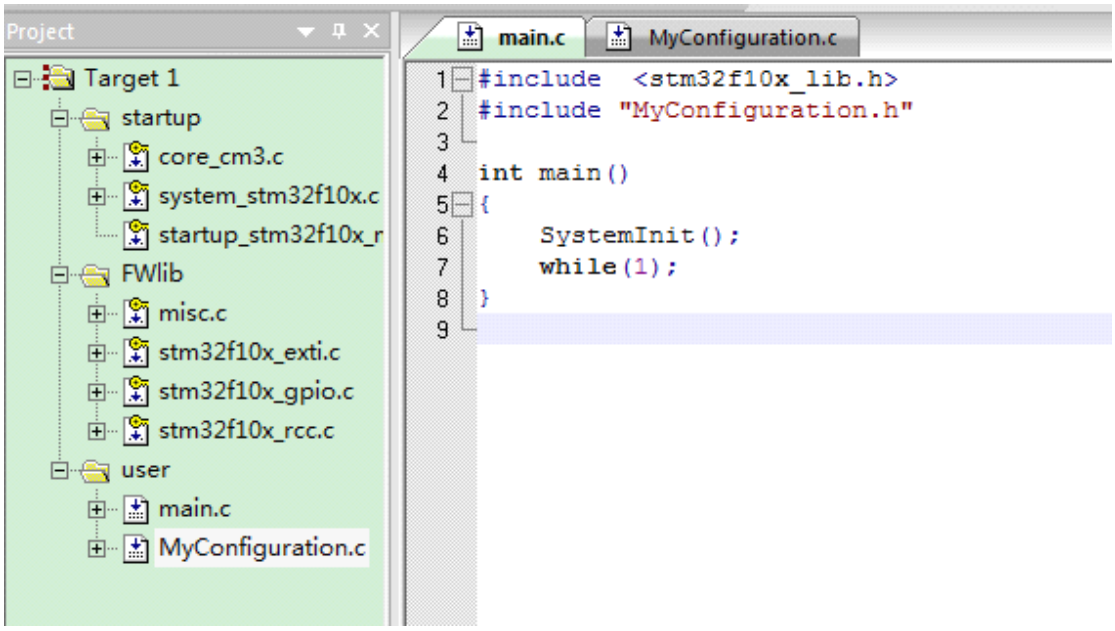
：屈环宇

：嘉兴学院

：2011 年 11 月 4 日晚

本次移植使用的处理器为 stm32f103rb，编译器为 Keil uVision4。

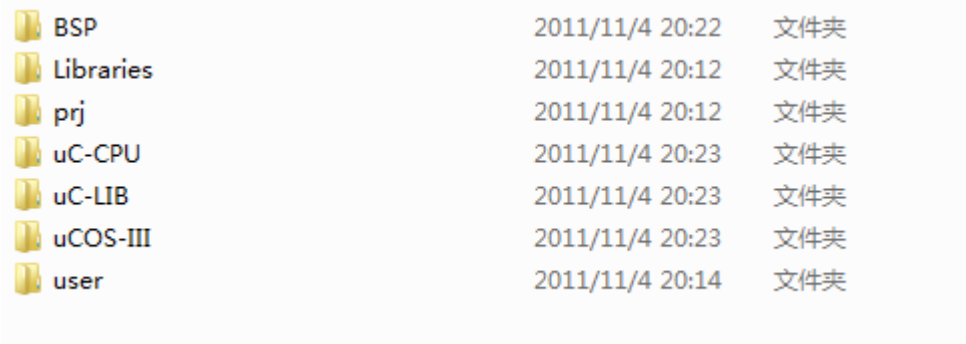
1、建立 MDK 工程模板



保证编译成功才能进行下一步。

这是我自己编写的文件，其实就是库函数的调用，这样写起来后使用就会方便很多（为了大家容易看懂，我把这文件中的其它函数都删除了）。

2、将 uC/OS-III 文件移植到工程文件夹中



原本有三个文件夹

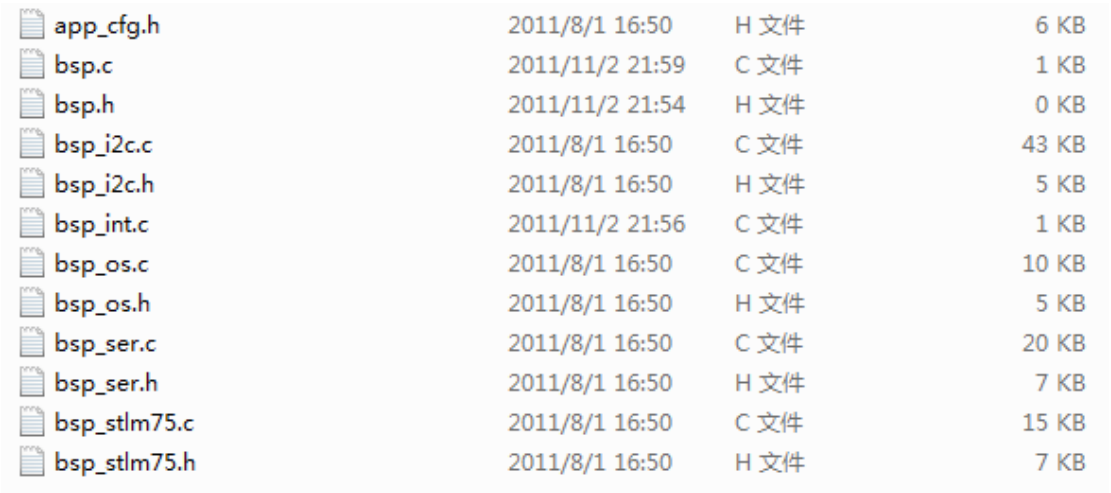
Libraries	存放着 stm32 的启动文件及库文件
User	存放着用户的文件
pri	存放着工程文件，链接文件，目标文件

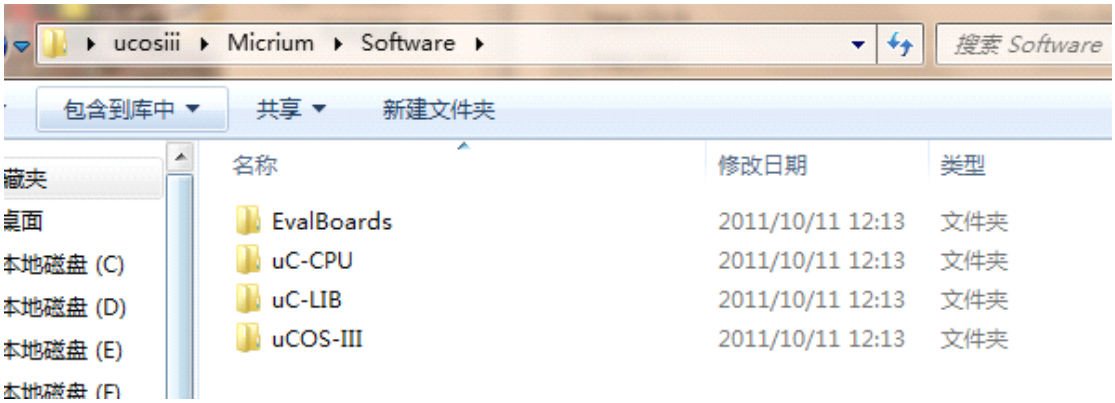
新添移植相关的四个文件

BSP	存放开发板外设的初始化文件
uC-CPU	存放与 CPU 相关的文件
uC-LIB	存放与一些通用文件
uCOS-III	存放 uC/OS-III 的源文件

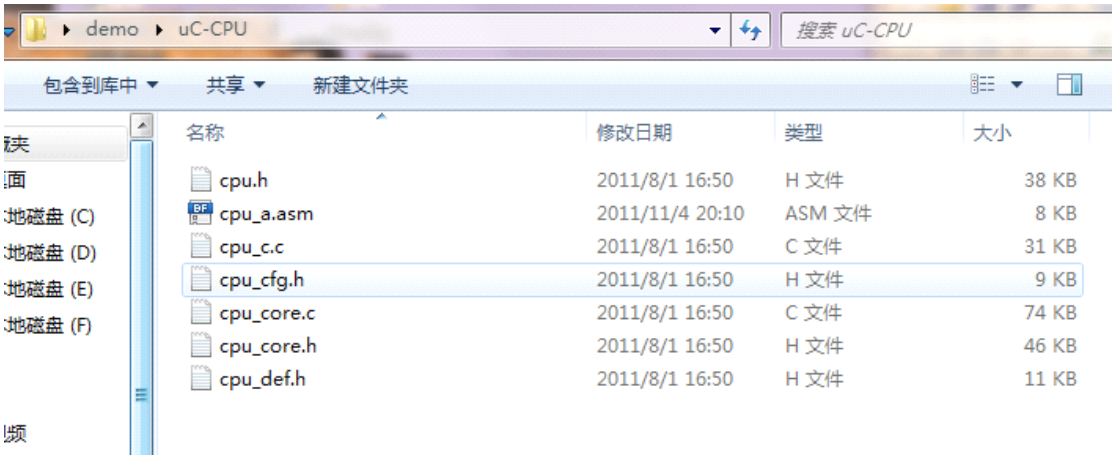
以下文件都可以在 Micrium 提供的压缩包中找到

首先介绍 BSP 文件夹

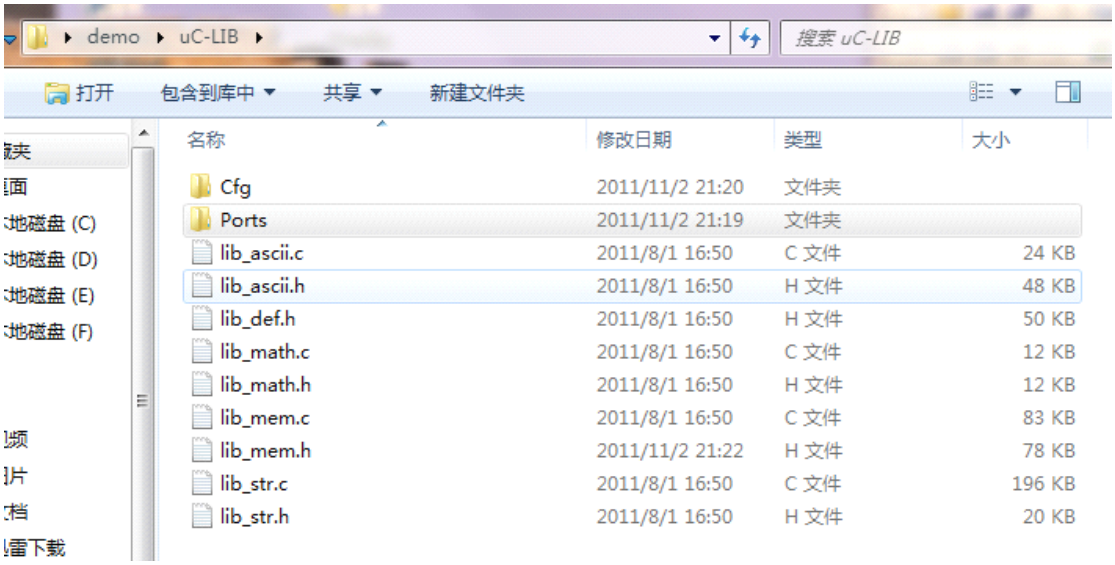


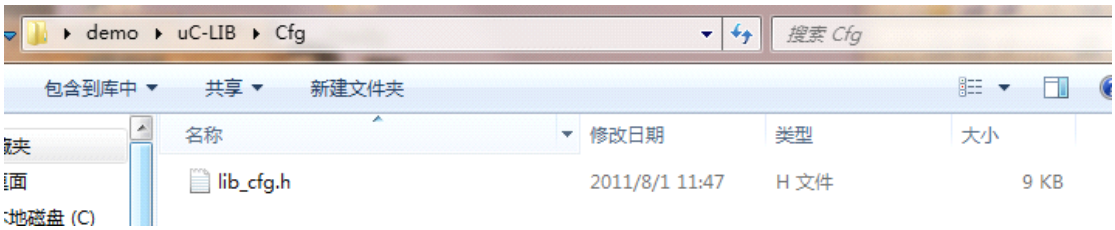


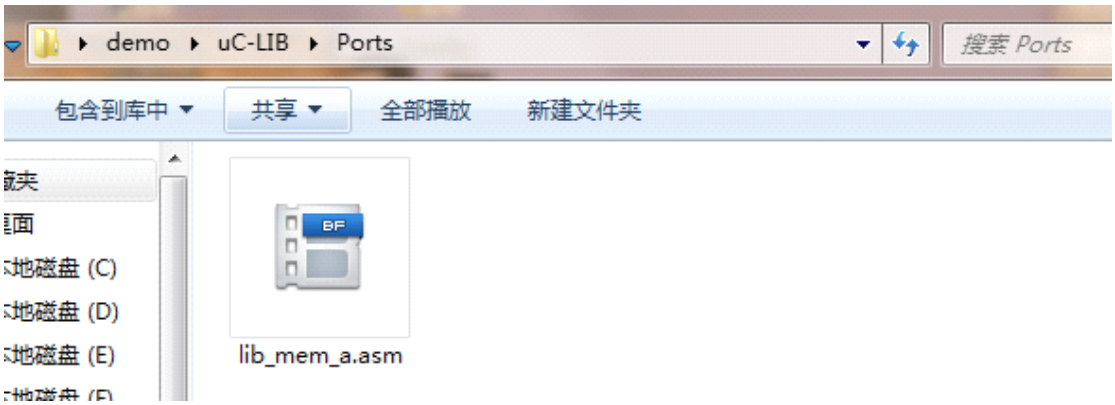
uC/CPU 文件夹中的文件



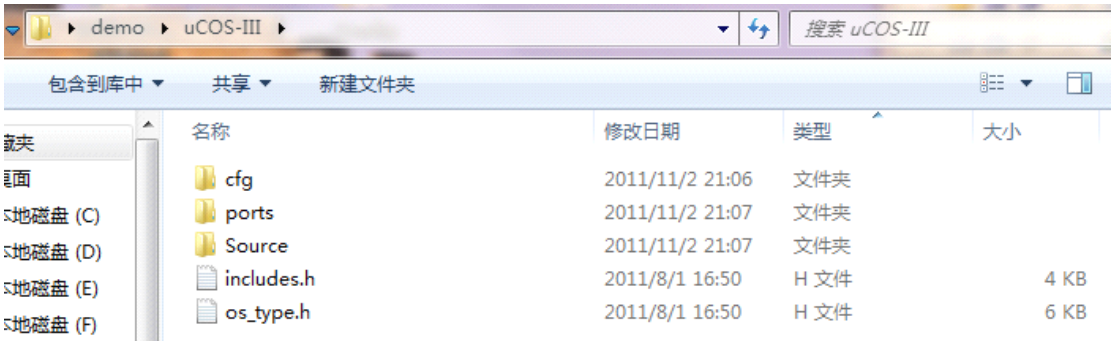
uC/LIB 中的文件

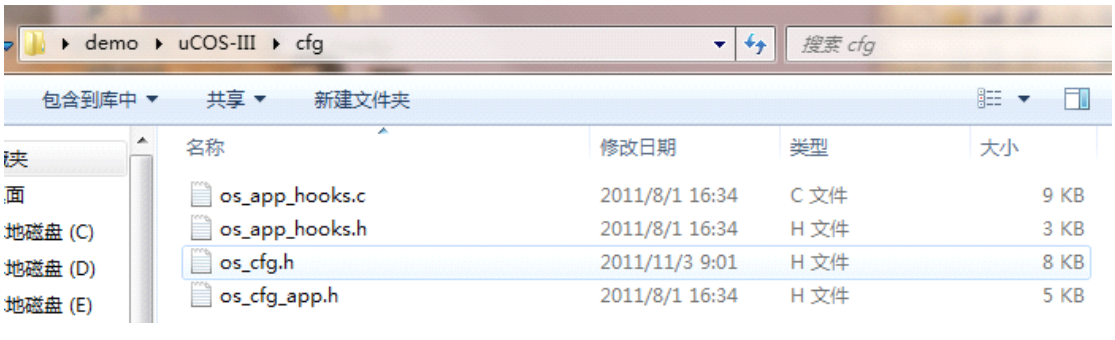


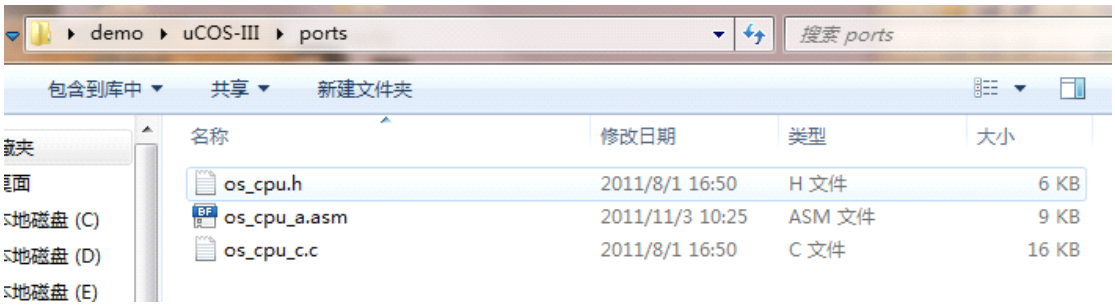


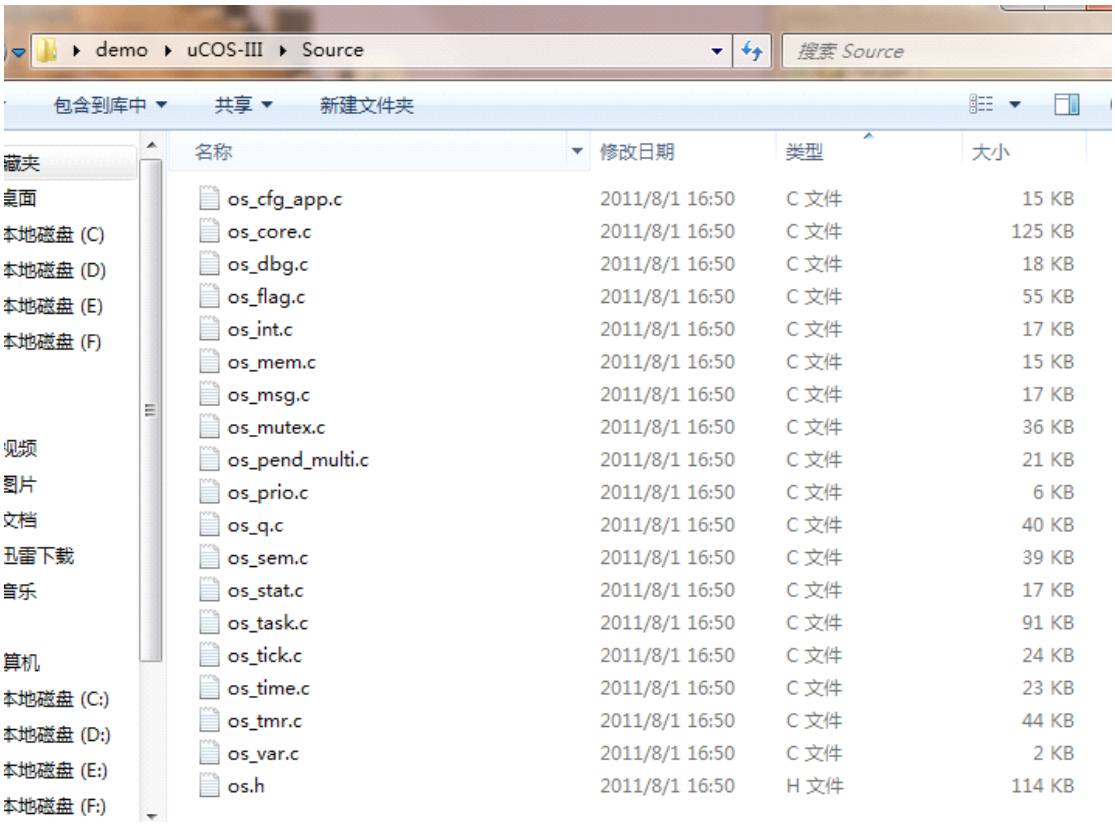


uC/OS-III 文件夹中的文件



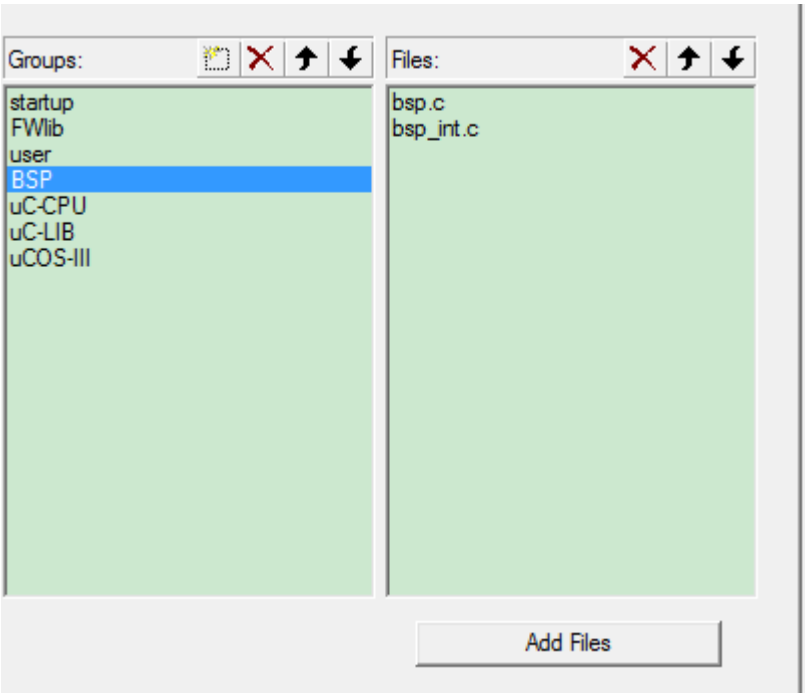


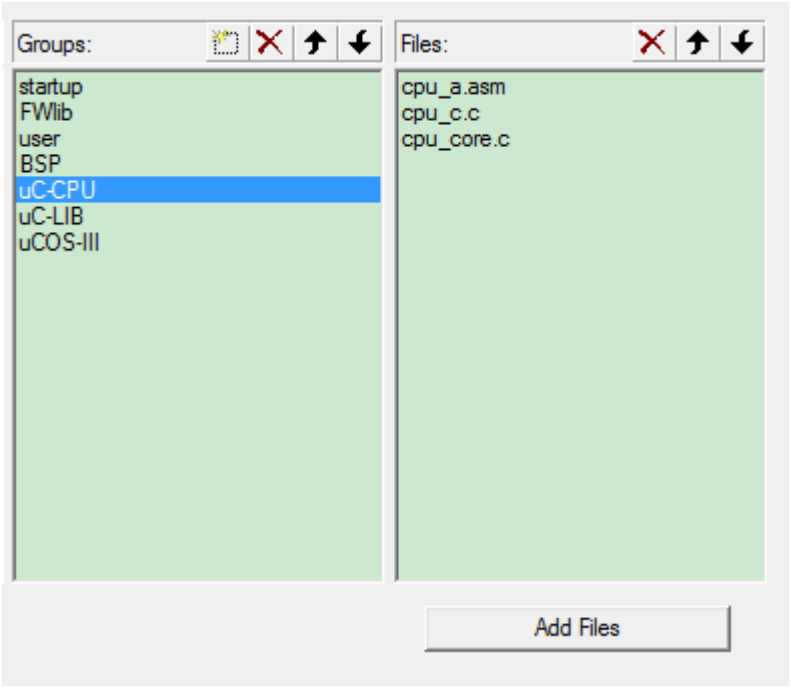


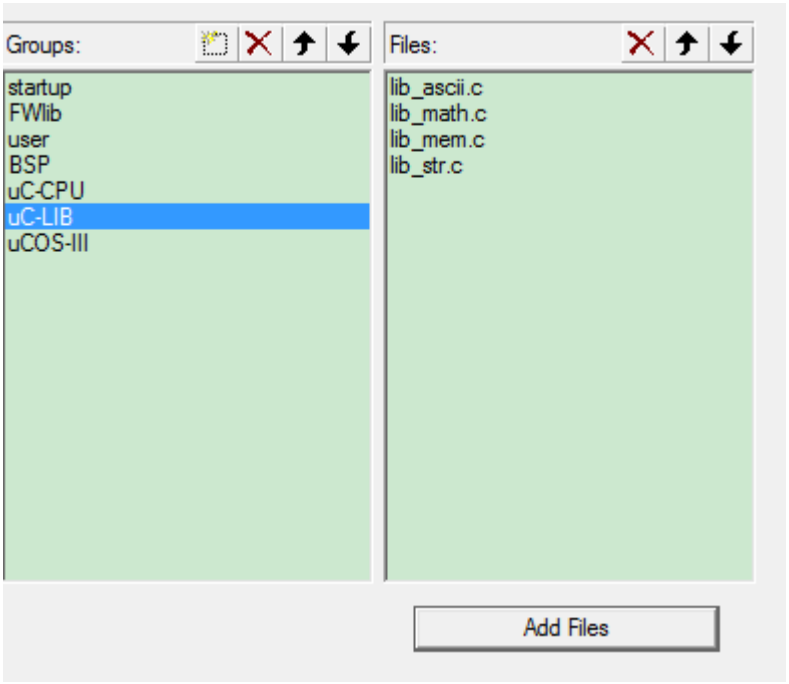


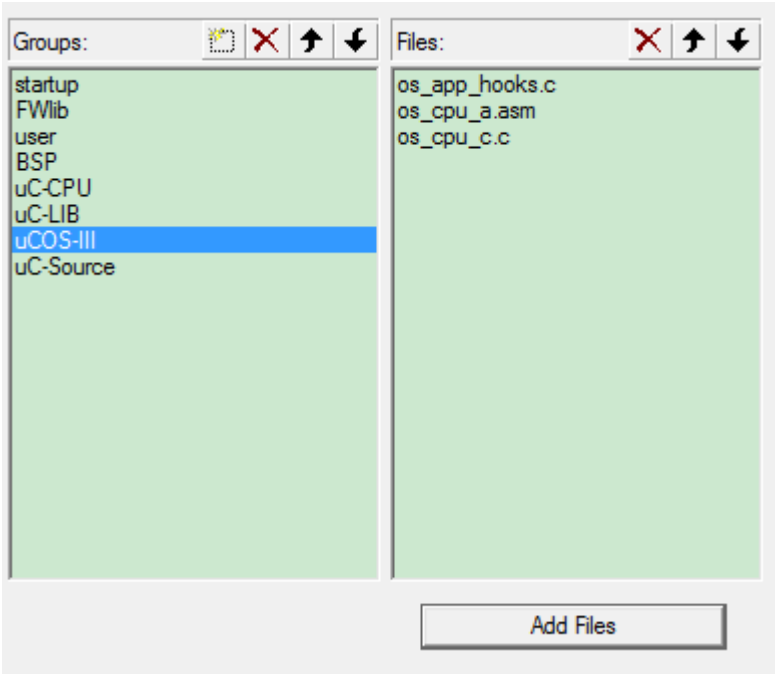
至此 所有的文件都添加完毕，接下来将这些文件添加到工程

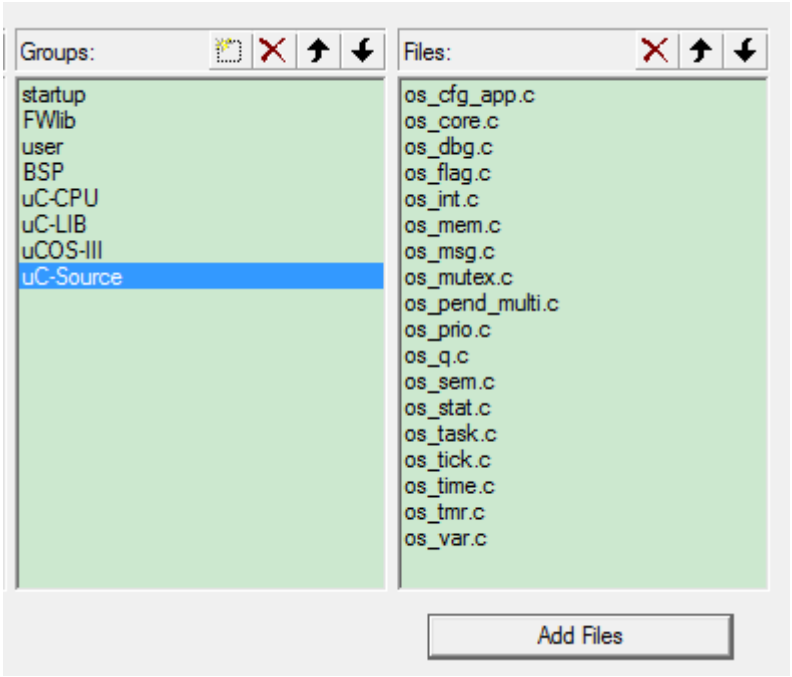
3、工程框架



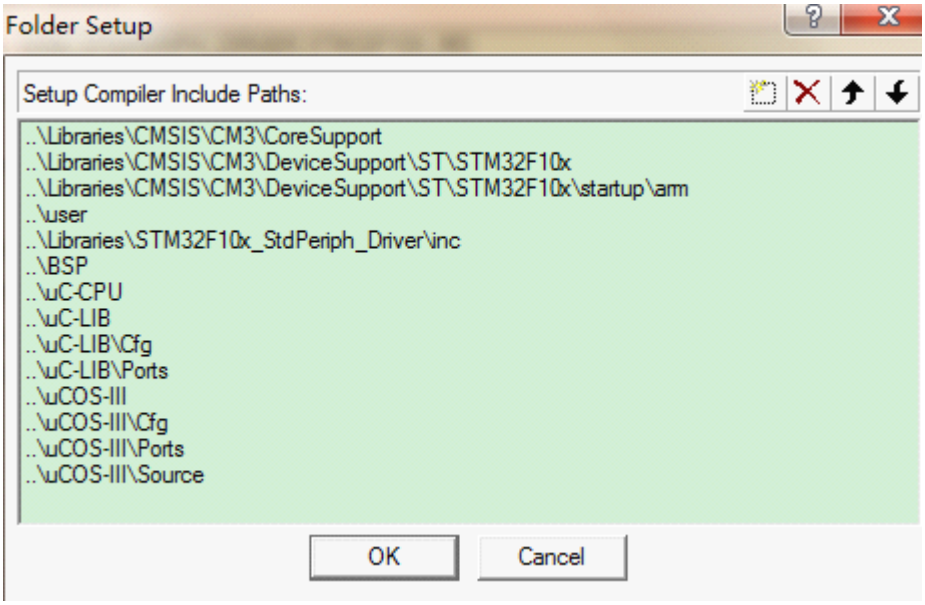

Groups:
startup
FWlib
user
BSP
uC-CPU
uC-LIB
uCOS-III
Files:
cpu_a.asm
cpu_c.c
cpu_core.c
Add Files


Groups:
startup
FWlib
user
BSP
uC-CPU
uC-LIB
uCOS-III
Files:
lib_ascii.c
lib_math.c
lib_mem.c
lib_str.c
Add Files


Groups:
Files:
startup
FWlib
user
BSP
uC-CPU
uC-LIB
uCOS-III
uC-Source
os_app_hooks.c
os_cpu_a.asm
os_cpu_c.c
Add Files

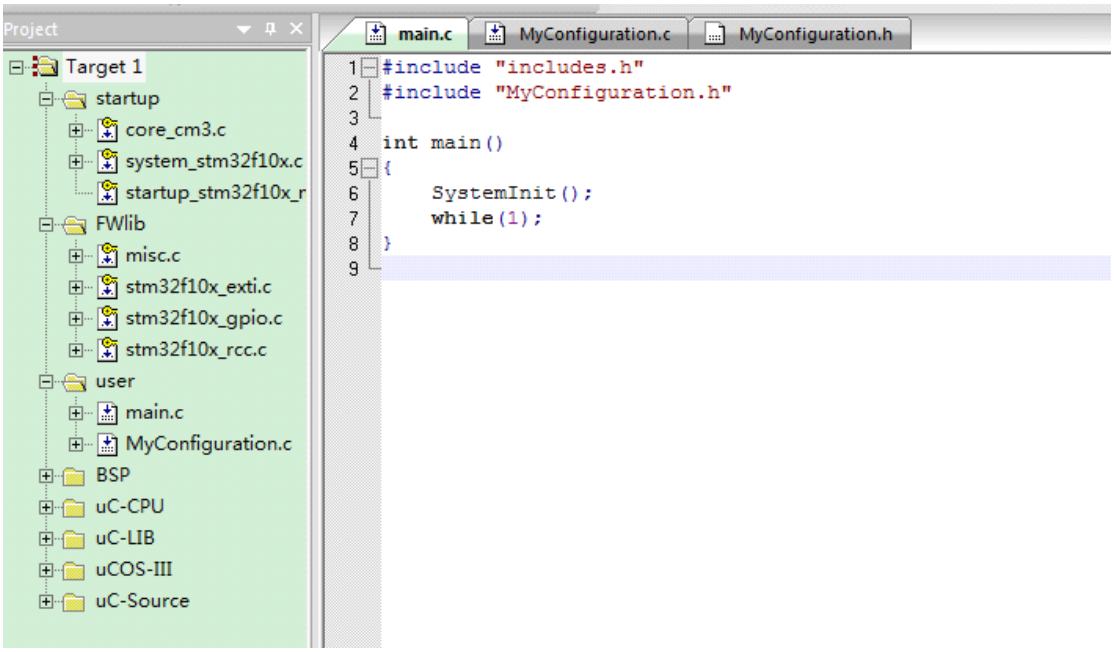

Groups:
Files:
startup
FWlib
user
BSP
uC-CPU
uC-LIB
uCOS-III
uC-Source
os_cfg_app.c
os_core.c
os_dbg.c
os_flag.c
os_int.c
os_mem.c
os_msg.c
os_mutex.c
os_pend_multi.c
os_prio.c
os_q.c
os_sem.c
os_stat.c
os_task.c
os_tick.c
os_time.c
os_tmr.c
os_var.c
Add Files

路径设置：



工程的框架搭好了，接下来就是修改文件了

4、移植配置



编译之后，错误一大堆~~~

不要怕，一个一个改~~

首先，我们是负责 Micrium 提供的 BSP 文件，但是这些 BSP 文件都是基于 Micrium 的评估板写的。为了大家看懂，我尽量让 BSP.c 中的文件越少越好。

main.c | bsp.c | bsp.h

```
1  #ifndef __BSP_H
2  #define __BSP_H
3
4  #endif
5
```

main.c | bsp.c | bsp.h

```
01 #include "bsp.h"
02
03 void CPU_TS_TmrInit()
04 {
05 }
06
07 void CPU_TS_TmrRd()
08 {
09 }
10
```

main.c | bsp.c | bsp.h | bsp_int.c

```
1
```

bsp_int.c 中直接为空

最后空一行是防止 MDK 的编译警告，至于为什么要添加这两个函数呢？
因为这两个是更软件定时器任务相关的函数，uC/OS-III 内核中与软件定时器相关的代码需要用户定义这两个函数，不然文件链接时会出错。

移植的难点，汇编文件的修改!!

```
  PUBLIC  CPU_IntDis
  PUBLIC  CPU_IntEn

  PUBLIC  CPU_SR_Save
  PUBLIC  CPU_SR_Restore

  PUBLIC  CPU_CntLeadZeros
  PUBLIC  CPU_RevBits

  PUBLIC  CPU_WaitForInt
  PUBLIC  CPU_WaitForExcept
```

将两个汇编文件 cpu_a.asm 和 os_cpu_a.asm 中的 PUBLIC 关键字全部改为 EXPORT。

因为 Micrium 是在 IAR 环境中编译的，所以关键字是不一样的。如下：

```
  EXPORT  CPU_IntDis
  EXPORT  CPU_IntEn

  EXPORT  CPU_SR_Save
  EXPORT  CPU_SR_Restore

  EXPORT  CPU_CntLeadZeros
  EXPORT  CPU_RevBits

  EXPORT  CPU_WaitForInt
  EXPORT  CPU_WaitForExcept
```

重新编译，错误是不是只有十几行了!!!

将上述两个汇编文件中如下两行代码（共两处）

```
;*******************************************************************************
;                                       CODE GENERATION DIRECTIVES
;*******************************************************************************

        RSEG CODE:CODE:NOROOT(2)
        THUMB

$PAGE
;*******************************************************************************
;                                    DISABLE and ENABLE INTERRUPTS
```

改为

```
;*******************************************************************************
                                        CODE GENERATION DIRECTIVES
;*******************************************************************************

        PRESERVE8

        AREA    |.text|, CODE, READONLY
        THUMB
$PAGE
;*******************************************************************************
                                      DISABLE and ENABLE INTERRUPTS
```

原因也是编译器不同所致

还是编译器不同导致的错误：

```
;                       #define d in "cpu_cfg.h" or "cpu.h"
;*******************************************************************************

CPU_CntLeadZeros:
        CLZ     R0, R0                          ; Count leading zeros
        BX      LR


;*******************************************************************************
```

将所有函数名的冒号删除

```
;

CPU_CntLeadZeros
        CLZ     R0, R0                          ; Count leading zeros
        BX      LR


;*******************************************************************************
```

修改完成后，再次编译，没有错误!!!　是不是很高兴呀 O(∩_∩)O 哈哈~

那么，接下来创建任务：

main.c* | bsp.c | bsp.h | bsp_int.c | cpu_a.asm | os_cpu_a.asm

```
01 #include "includes.h"
02 #include "MyConfiguration.h"
03
04 static  OS_TCB  Task1TCB;
05 static  CPU_STK Task1Stk[128];
06 static void  Task1(void* p_arg);
07
08 int main()
09 {
10     OS_ERR err;
11     SystemInit();
12     OSInit(&err);
13     OSTaskCreate((OS_TCB     *)&Task1TCB,
14                  (CPU_CHAR   *)"Task1 Start",
15                  (OS_TASK_PTR )Task1,
16                  (void       *)0,
17                  (OS_PRIO     )2,
18                  (CPU_STK    *)&Task1Stk[0],
19                  (CPU_STK_SIZE)12,
20                  (CPU_STK_SIZE)128,
21                  (OS_MSG_QTY  )0,
22                  (OS_TICK     )0,
23                  (void       *)0,
24                  (OS_OPT      )(OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
25                  (OS_ERR     *)&err);
26
27     OSStart(&err);
28 }
29 static  void  Task1(void *p_arg)
30 {
31     OS_ERR err;
32     while(1)
33     {
34        OSTimeDly(200,OS_OPT_TIME_DLY,&err);
35     }
36 }
```

再次编译，还是没有错误，那么 是否 uC/OS-III 真的能运行了呢?_?

答案是否定的 ╭∩╮（ ̄ ̄）╭∩╮

不要忘记了，uC/OS-III 的运行是需要时基中断的，也就是需要被提供时钟周期。在 stm32 中，是由 SysTick 提供的。

新增两个函数：

```
static  OS_TCB  Task1TCB;
static  CPU_STK Task1Stk[128];
static void  Task1(void* p_arg);

void SysTick_Configuration(void);

int main()
{
    OS_ERR err;
    SystemInit();
    SysTick_Configuration();
    OSInit(&err);
    OSTaskCreate((OS_TCB     *)&Task1TCB,
```

```
40
41 //系统时钟中断服务函数
42 void SysTick_Handler(void)
43 {
44     OSTimeTick();         /* Call uC/OS-II's OSTimeTick()               */
45     OSIntExit();          /* Tell uC/OS-II that we are leaving the ISR  */
46 }
47 //系统时钟配置，设计1ms产生一次中断
48 void SysTick_Configuration(void)
49 {
50
51
52     SysTick->CTRL&=~(1<<2);//SYSTICK使用外部时钟源
53     SysTick->CTRL|=1<<1;   //开启SYSTICK中断
54     SysTick->LOAD=9000;    //产生1ms中断
55     NVIC_Configuration(3,3,SystemHandler_SysTick);
56     SysTick->CTRL|=1<<0;   //开启SYSTICK
57 }
58
```

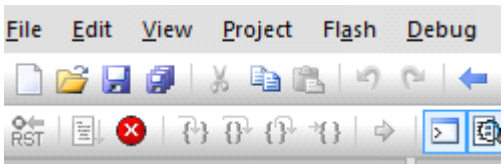
（中断服务函数不需要申明）

再次编译，没有错误 o(≧v≦)o~~好棒

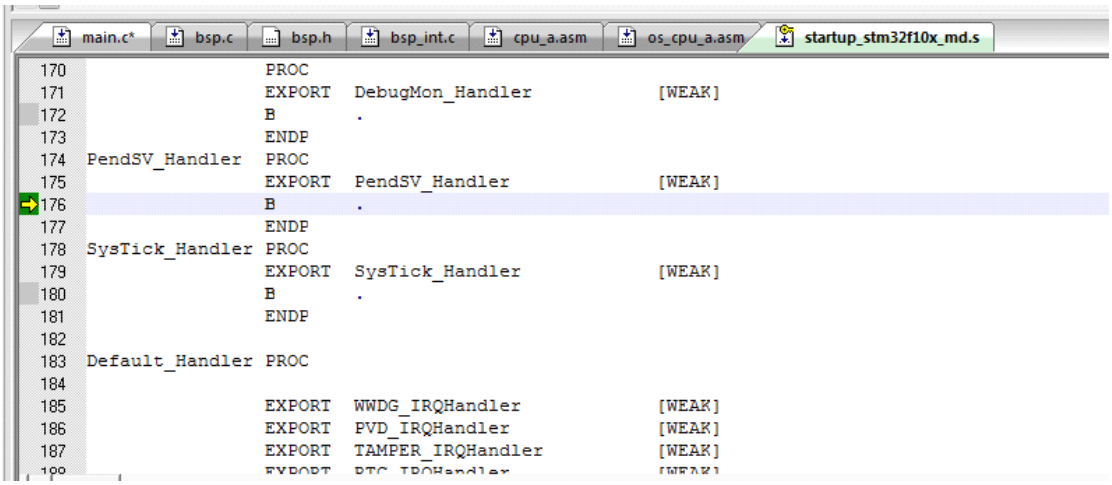
但是，也别高兴地太早了

```
31 }
32 static  void  Task1(void *p_arg)
33 {
34     OS_ERR err;
35     while(1)
36     {
37         OSTimeDly(200,OS_OPT_TIME_DLY,&err);
38     }
39 }
```

在任务 1 处设置断点，软件调试，程序是否能运行到任务~~



调式，并运行，过了好久、好久……
还是没停下了，没办法，强制停止!!



发现程序指针始终在这里。

显然　PendSV_Handler　函数出问题了!!

```
133 ;********************************************************
134
135 OS_CPU_PendSVHandler
136     CPSID   I
137     MRS     R0, PSP
138     CBZ     R0, OS_CPU_PendSVHandler_nosave
139
140     SUBS    R0, R0, #0x20
141     STM     R0, {R4-R11}
142
143     LDR     R1, =OSTCBCurPtr
144     LDR     R1, [R1]
145     STR     R0, [R1]
146
```

改为

```
134
135 PendSV_Handler
136     CPSID   I
137     MRS     R0, PSP
138     CBZ     R0, OS_CPU_PendSVHandler_nosave
139
140     SUBS    R0, R0, #0x20
141     STM     R0, {R4-R11}
142
143     LDR     R1, =OSTCBCurPtr
144     LDR     R1, [R1]
145     STR     R0, [R1]
146
147
```

```
036
037     EXPORT  OSStartHighRdy
038     EXPORT  OS_CPU_PendSVHandler
039
040 ;PAGE□
041 ;*********************************************************************
```

改为

```
036
037     EXPORT  OSStartHighRdy
038     EXPORT  PendSV_Handler
039
040 ;PAGE□
041 ;*********************************************
042 ;
```

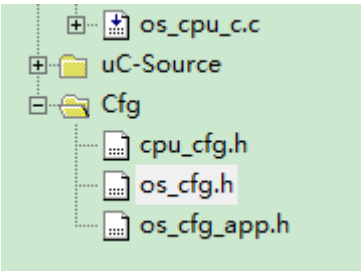
然后再编译，调试~~~~~

结果发现

```
35      while(1)
36      {
37          OSTimeDly(200,OS_OPT_TIME_DLY,&err);
38      }
39  }
40
41  //系统时钟中断服务函数
42  void SysTick_Handler(void)
```

只有一次能进入任务！

这时，就需要考虑到 uC/OS-III 内部机制了，这时，我们设置 uC/OS-III 的配置文件



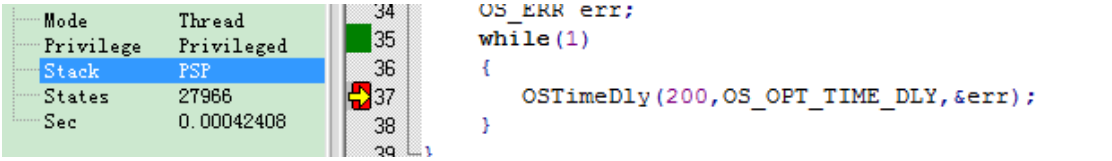
我们再加入这个目录，打开 os_cfg.h 文件

```
037                                         /* ---------------------------- MISCELLANEOUS ------------
038 #define OS_CFG_APP_HOOKS_EN             1u   /* Enable (1) or Disable (0) application specific hooks
039 #define OS_CFG_ARG_CHK_EN               1u   /* Enable (1) or Disable (0) argument checking
040 #define OS_CFG_CALLED_FROM_ISR_CHK_EN   1u   /* Enable (1) or Disable (0) check for called from ISR
041 #define OS_CFG_DBG_EN                   1u   /* Enable (1) debug code/variables
042 #define OS_CFG_ISR_POST_DEFERRED_EN     0u   /* Enable (1) or Disable (0) Deferred ISR posts
043 #define OS_CFG_OBJ_TYPE_CHK_EN          1u   /* Enable (1) or Disable (0) object type checking
044 #define OS_CFG_TS_EN                    1u   /* Enable (1) or Disable (0) time stamping
045
```

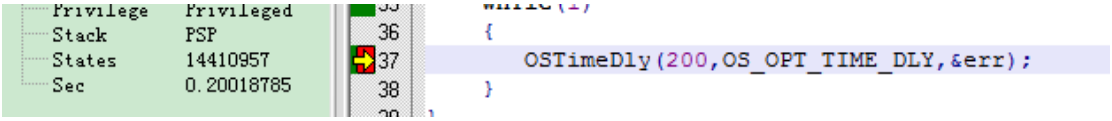
设置关闭中断延迟提交方式（在我翻译 uC/OS-III 中文资料中有介绍）

再编译，调试

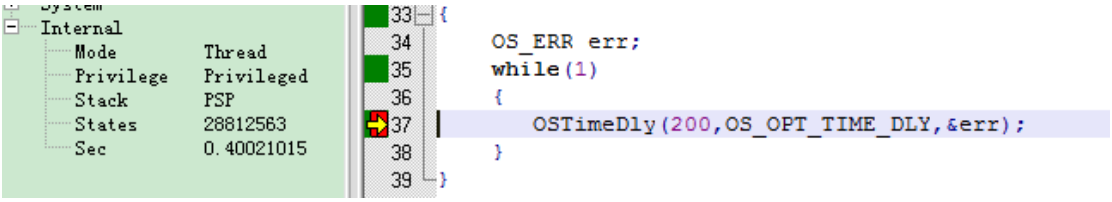
第一次运行到断点：



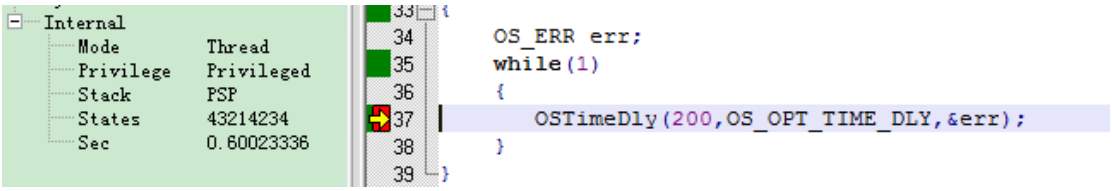
第二次运行到断点：



第三次运行到断点：



第四次运行到断点：



注意到图片中的 Sec，间隔都十分接近 0.2 秒的

至此 uC/OS-III 移植完成。

在 BSP 中编写板子 LED 灯的驱动程序，放在任务中开启，关闭。就能在硬件上测试 uC/OS-III 是否能成功运行了。

注意： hook 函数应该定义为空（或者根据自己的开发板设置），或者修改 os_cfg.h 中的 OS_CFG_APP_HOOKS_EN 为 0（关闭 hook 函数）。

注意：文件应该存放到对应的位置。文件名与文件的作用密切相关，函数名与函数的功能密切相关，这对于大型项目、团队合作会很有好处的。

在此，仅创建了一个任务。多个任务，信号量、消息等内核对象的创建是简单的，相信大家都能自己解决的，我就不多介绍了。

uC/OS-III 的详细说明，见 uC/OS-III 中文资料。我把它放到百度文库中了，只有百度搜索"uC/OS-III 中文资料"就能找到。

我当时移植的时候就是这种思想的，这样写出来可能更利于大家理解。