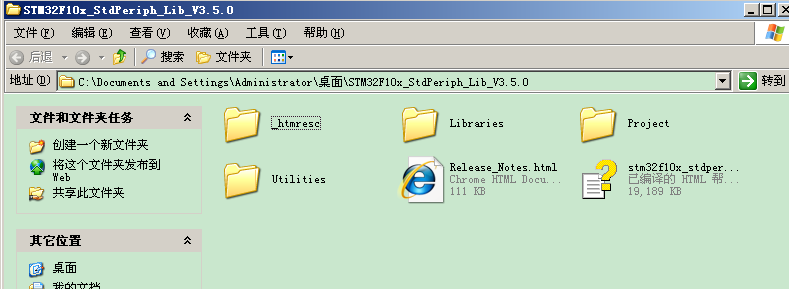
我用的是stm32f103ve的板子和IAR for ARM 6.3的编译器，由于本人水平有限，着重写移植的操作，至于为什么这么有这个操作，我能写多少写多少，大家见谅. (～ o ～)~

## 准备材料

在st官网上下载最新的固件库（现在是v3.5.0），这个库作用非常大，可以帮我们快速开发出想要的功能，但是网上有很多人说这个库有漏洞，本人水平有限就不得而知了，这里直接上地址。

<http://www.st.com/internet/com/SOFTWARE_RESOURCES/SW_COMPONENT/FIRMWARE/stm32f10x_stdperiph_lib.zip>

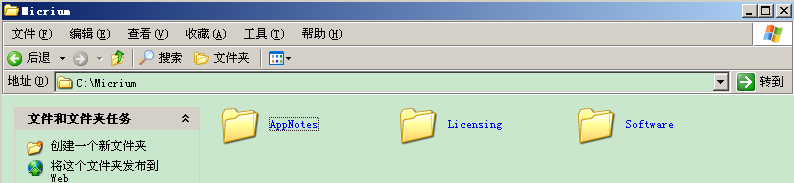
解压后得到目录如下图



在ucos官网上下载移植stm32移植版的ucosii代码，这个需要注册一下，不多说直接上接上地址

<http://micrium.com/download/uCOSII-ST-STM32F103ZE-SK.exe>

下载安装后得到目录如下图



## 建立工程

首先建三个文件目录：CMSIS用来存放cpu和系统的相关的文件、STM32F10x\_StdPeriph\_Driver用来存放cpu的硬件驱动、ucosii用来存放ucos的源代码。

复制固件库\STM32F10x\_StdPeriph\_Lib\_V3.5.0\Libraries\CMSIS\CM3\CoreSupport目录下的core\_cm3.c和core\_cm3.h到工程文件夹下的CMSIS目录中。

复制固件库\STM32F10x\_StdPeriph\_Lib\_V3.5.0\Libraries\CMSIS\CM3\DeviceSupport\ST

\STM32F10x目录下的stm32f10x.h、system\_stm32f10x.c、system\_stm32f10x.h三个文件到工程文件夹下的CMSIS目录中。

在固件库\STM32F10x\_StdPeriph\_Lib\_V3.5.0\Libraries\CMSIS\CM3\DeviceSupport\ST\

STM32F10x\startup\iar目录中选取相应的开始文件，这里我选取的是startup\_stm32f10x\_hd.s复制到到工程文件夹下的CMSIS目录中。

复制固件库\STM32F10x\_StdPeriph\_Lib\_V3.5.0\Project\STM32F10x\_StdPeriph\_Template目录中的stm32f10x\_conf.h、stm32f10x\_it.c、stm32f10x\_it.h三个文件到工程文件夹下的CMSIS目录中。其中stm32f10x\_conf.h是驱动的配置文件，可以根据实际需求注释掉不用的驱动，默认是全部启用的，这里就不做改变了。stm32f10x\_it.c、stm32f10x\_it.h是中断的相应函数所在的文件，但仅给出了一部分，如果用到其他的中断可以打开选取的startup\_stm32f10x\_hd.s文件，来确定相应的函数名称。

在工程目录STM32F10x\_StdPeriph\_Driver下面建立inc目录，并复制固件库\STM32F10x\_StdPeriph\_Lib\_V3.5.0\Libraries\STM32F10x\_StdPeriph\_Driver\inc目录下的所有文件到此文件夹中

在工程目录STM32F10x\_StdPeriph\_Driver下面建立src目录，并复制固件库\STM32F10x\_StdPeriph\_Lib\_V3.5.0\Libraries\STM32F10x\_StdPeriph\_Driver\src目录下的所有文件到此目录中

在工程目录ucosii下建立ports目录，并复制ucos安装目录\Micrium\Software\uCOS-II\

Ports\arm-cortex-m3\Generic\IAR下的所有文件到此文件夹中。

在工程目录ucosii下建立source目录，并复制ucos安装目录\Micrium\Software\uCOS-II

\Source下的所有文件到此文件夹中。

下面还需要系统运行所需的几个文件，一个是os\_cfg.h，用来配置ucos各个功能，一个是app\_cfg.h用来配置任务所需的各项参数。这两个文件我是从\Micrium\Software\EvalBoards\

ST\STM32F103ZE-SK\IAR\OS-Probe-LCD目录中借鉴的，用户也可以根据自己的需求来更改,具体更改方法网上有很多说明这里就不多说了。将这两个文件复制到工程根目录下。

我们还缺少一个main函数的文件，我们新建立一个文本文档，取名为test.c

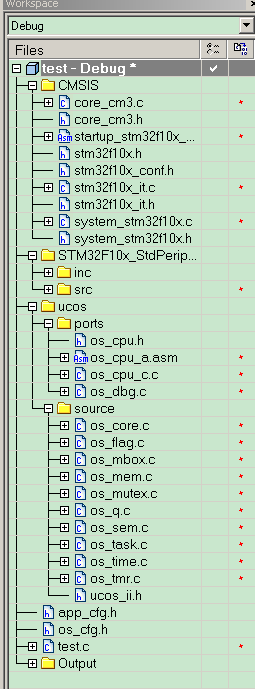
最后我们还缺少一个连接时的配置文件，这个文件我是从IAR给的例程里借鉴的，复制\IAR Systems\Embedded Workbench 6.0 Evaluation\arm\examples\ST\STM32F10x\IAR-STM32-SK\

GettingStarted\config目录下的STM32F10x\_FLASH.icf文件到工程根目录下。

至此我们所有的文件已经准备好了。

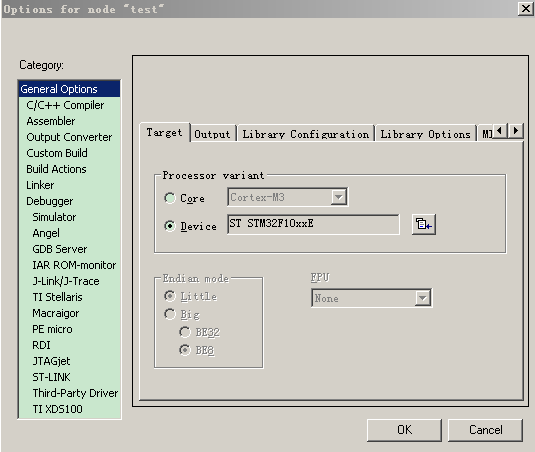
## 建立和配置工程

打开IAR建立一个ARM的空工程 取名test。向工程添加文件，格式我是按照目录形式安排的这个随便大家，直接上图。



配置工程：

选取相应的芯片



添加相应的包含目录和宏定义

我的包含目录为

$PROJ\_DIR$\

$PROJ\_DIR$\CMSIS

$PROJ\_DIR$\STM32F10x\_StdPeriph\_Driver\inc

$PROJ\_DIR$\STM32F10x\_StdPeriph\_Driver\src

$PROJ\_DIR$\ucosii\ports

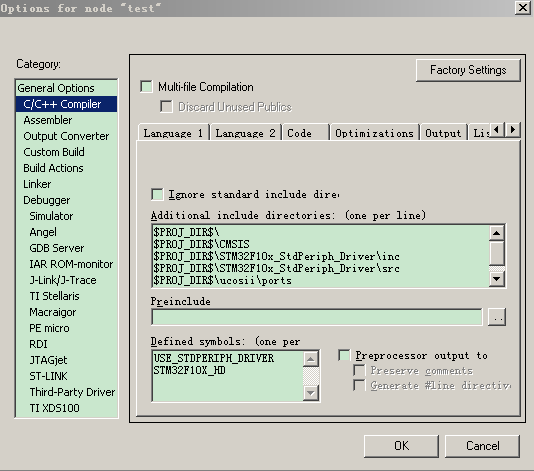
$PROJ\_DIR$\ucosii\source

需要的宏定义

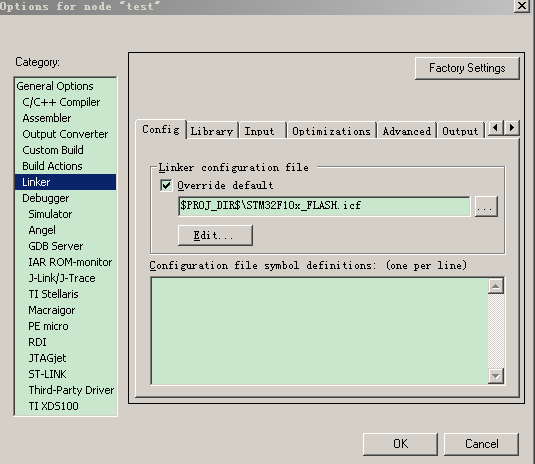
USE\_STDPERIPH\_DRIVER

STM32F10X\_HD

其中USE\_STDPERIPH\_DRIVER表示要使用固件库，TM32F10X\_HD表示选取的芯片为大容量芯片，这两个宏定义也可以在stm32f10x.h中的70行和105行取消相应的注释来选择，两种方法二选一。



选择链接配置文件



最后配置一下仿真，就此完成。

## 移植配置

由于这个版本是官方已经移植好的，我们所需要修改的东西很少只有两个地方。

打开startup\_stm32f10x\_hd.s(注意要去掉文件默认的只读属性,否则修改不成功)

把所有出现PendSV\_Handler的地方替换成OS\_CPU\_PendSVHandler

把所有出现SysTick\_Handler的地方替换成OS\_CPU\_SysTickHandler

之所以要修改这两个地方，是因为我们一般的中断向量用的是CMSIS给的，而ucos的中断向量是自己写的，我们修改CMSIS的向量表来统一这两个向量这种方法最简单，当然你也可以，修改ucos系统的函数名，这种方法我没有试，不过我见过另一个同事这么修改的也成功了（我不推荐这种方法，因为不知道修改ucos函数调用复杂，修改以后可能会编译出错，有心情的可以试一下）。

当然也有网上别人的移植教程是重写void OS\_CPU\_SysTickHandler(void);

void OS\_CPU\_SysTickInit(void)；两个函数，同时替换了PendSV\_Handler。这种方法也可以。

具体再深一步的话要研究这几个函数在系统中的作用，可以参见邵老师的书或者百度，这里不多说了（因为我说不明白-\_-!）。

编译一下，咦发现出错了，仔细研究发现是因为core\_cm3.h中的函数声明和intrinsics.h中的函数声明重复了,这里把1049行#elif (defined (\_\_ICCARM\_\_)) 直接修改#elif (defined (\_\_ICCARM\_\_USR)) 不让编译器编译。

还要修改os\_cfg.h中的

#define OS\_APP\_HOOKS\_EN 0 /\* Application-defined hooks are called from the uC/OS-II hooks\*/

#define OS\_ARG\_CHK\_EN 0 /\*Enable (1) or Disable (0) argument checking \*/

#define OS\_TMR\_EN 0 /\* Enable (1) or Disable (0) code generation for TIMERS \*/

三个地方，这些功能我们暂时不用，要用的话要定义一些东西，这里不做介绍，参见邵老师的书吧。

还缺少一个函数定义如下

INT32U OS\_CPU\_SysTickClkFreq(void)

{

RCC\_ClocksTypeDef

rcc\_clocks;

RCC\_GetClocksFreq(&rcc\_clocks);

return ((INT32U)rcc\_clocks.HCLK\_Frequency);

//return SYSCLK\_FREQ\_72MHz;

}

这个函数主要是用来获得cpu频率，用来确定系统滴答的时间。如果知道系统时钟是多少可以不用这么复杂，直接

INT32U OS\_CPU\_SysTickClkFreq(void)

{

return 72000000;

}

也可以。

当然还有个错误是缺少main函数，下面我们就来写。

首先把app\_cfg.h所有内容删掉替换成下面的（因为这个文件是借鉴的下面要自己写了）

#ifndef \_\_APP\_CFG\_H\_\_

#define \_\_APP\_CFG\_H\_\_

/\* task stack size \*/

#define STARTUP\_TASK\_STK\_SIZE 128

#define TASK1\_STK\_SIZE 128

#define TASK2\_STK\_SIZE 128

#define TASK3\_STK\_SIZE 128

#define TASK4\_STK\_SIZE 128

/\* task priority \*/

#define STARTUP\_TASK\_PRIO 4

#define TASK1\_PRIO 5

#define TASK2\_PRIO 6

#define TASK3\_PRIO 7

#define TASK4\_PRIO 8

#endif

这个文件主要就是每个任务的优先级和堆栈大小。

我们建立5个任务，其中一个是启动任务，其余四个分别闪一个led等。代码如下。

#include "stm32f10x.h"

#include "ucos\_ii.h"

OS\_STK startup\_stk[STARTUP\_TASK\_STK\_SIZE];

OS\_STK task1\_stk[TASK1\_STK\_SIZE];

OS\_STK task2\_stk[TASK2\_STK\_SIZE];

OS\_STK task3\_stk[TASK3\_STK\_SIZE];

OS\_STK task4\_stk[TASK4\_STK\_SIZE];

u8 os\_err;

static void task1(void \*p\_arg)

{

for (;;)

{

GPIO\_WriteBit(GPIOA,GPIO\_Pin\_4 ,Bit\_SET);

OSTimeDly(500);

GPIO\_WriteBit(GPIOA,GPIO\_Pin\_4 ,Bit\_RESET);

OSTimeDly(500);

}

}

static void task2(void \*p\_arg)

{

for (;;)

{

GPIO\_WriteBit(GPIOA,GPIO\_Pin\_5 ,Bit\_SET);

OSTimeDly(500);

GPIO\_WriteBit(GPIOA,GPIO\_Pin\_5 ,Bit\_RESET);

OSTimeDly(500);

}

}

static void task3(void \*p\_arg)

{

for (;;)

{

GPIO\_WriteBit(GPIOA,GPIO\_Pin\_6 ,Bit\_SET);

OSTimeDly(500);

GPIO\_WriteBit(GPIOA,GPIO\_Pin\_6 ,Bit\_RESET);

OSTimeDly(500);

}

}

static void task4(void \*p\_arg)

{

for (;;)

{

GPIO\_WriteBit(GPIOA,GPIO\_Pin\_7 ,Bit\_SET);

OSTimeDly(500);

GPIO\_WriteBit(GPIOA,GPIO\_Pin\_7 ,Bit\_RESET);

OSTimeDly(500);

}

}

static void startup(void \*p\_arg)

{

OS\_CPU\_SysTickInit(); //初始化时钟滴答

os\_err = OSTaskCreate(task1, //任务1

(void \*)0, //不带参数

(OS\_STK \*)&task1\_stk[TASK1\_STK\_SIZE - 1],//堆栈指针

(INT8U)TASK1\_PRIO); //优先级

os\_err = OSTaskCreate(task2, //任务2

(void \*)0, //不带参数

(OS\_STK \*)&task2\_stk[TASK2\_STK\_SIZE - 1],//堆栈指针

(INT8U)TASK2\_PRIO); //优先级

os\_err = OSTaskCreate(task3, //任务3

(void \*)0, //不带参数

(OS\_STK \*)&task3\_stk[TASK3\_STK\_SIZE - 1],//堆栈指针

(INT8U)TASK3\_PRIO); //优先级

//

os\_err = OSTaskCreate(task4, //任务4

(void \*)0, //不带参数

(OS\_STK \*)&task4\_stk[TASK4\_STK\_SIZE - 1],//堆栈指针

(INT8U)TASK4\_PRIO);

OSTaskDel(OS\_PRIO\_SELF); //任务建立完成删掉这个任务

}

所用main函数为

int main(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

SystemInit();

NVIC\_SetVectorTable(NVIC\_VectTab\_FLASH, 0x0);

NVIC\_PriorityGroupConfig(NVIC\_PriorityGroup\_4);

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA ,

ENABLE);

RCC\_APB2PeriphResetCmd(RCC\_APB2Periph\_GPIOA ,

DISABLE);

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_Out\_PP;

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_4 | GPIO\_Pin\_5 | GPIO\_Pin\_6 |

GPIO\_Pin\_7;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

OSInit();

OSTaskCreate(startup, (void \*)0, &startup\_stk[STARTUP\_TASK\_STK\_SIZE-1], STARTUP\_TASK\_PRIO);

OSStart();

return 0;

}

编译通过,下载到目标板上,运行4个灯同时闪了起来.移植成功.

本人刚学stm32 和ucos不久,希望大家能多多指正.