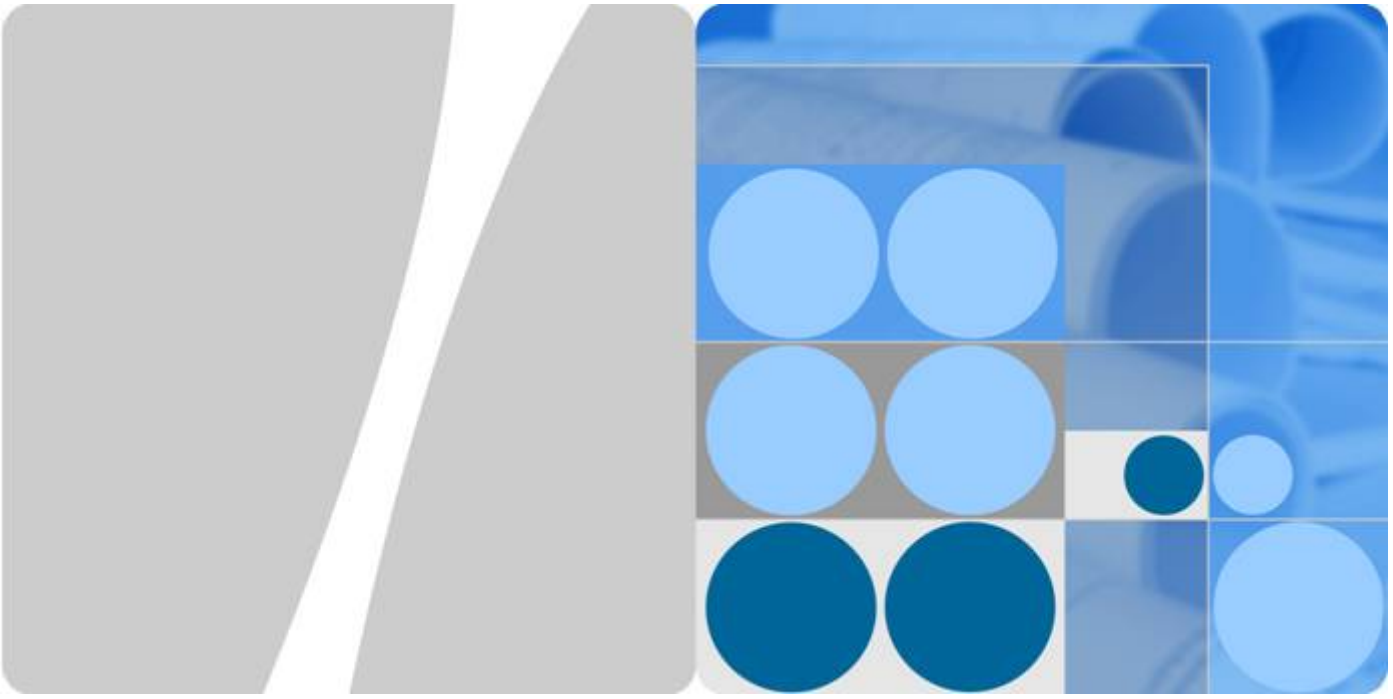


Huawei LiteOS 开源团队

Huawei LiteOS 第三方芯片移植指南（Keil 版）

文档版本　01

发布日期　2016-12-12

华为技术有限公司

HUAWEI

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

1 开源协议说明

您可以自由地：

分享 — 在任何媒介以任何形式复制、发行本作品

演绎 — 修改、转换或以本作品为基础进行创作

只要你遵守许可协议条款，许可人就无法收回你的这些权利。

惟须遵守下列条件：

署名 — 您必须提供适当的证书，提供一个链接到许可证，并指示是否作出更改。您可以以任何合理的方式这样做，但不是以任何方式表明，许可方赞同您或您的使用。

非商业性使用 — 您不得将本作品用于商业目的。

相同方式共享 — 如果您的修改、转换，或以本作品为基础进行创作，仅得依本素材的授权条款来散布您的贡献作品。

没有附加限制 — 您不能增设法律条款或科技措施，来限制别人依授权条款本已许可的作为。

声明：

当您使用本素材中属于公众领域的元素，或当法律有例外或限制条款允许您的使用，则您不需要遵守本授权条款。

未提供保证。本授权条款未必能完全提供您预期用途所需要的所有许可。例如：形象权、隐私权、著作人格权等其他权利，可能限制您如何使用本素材。

注意

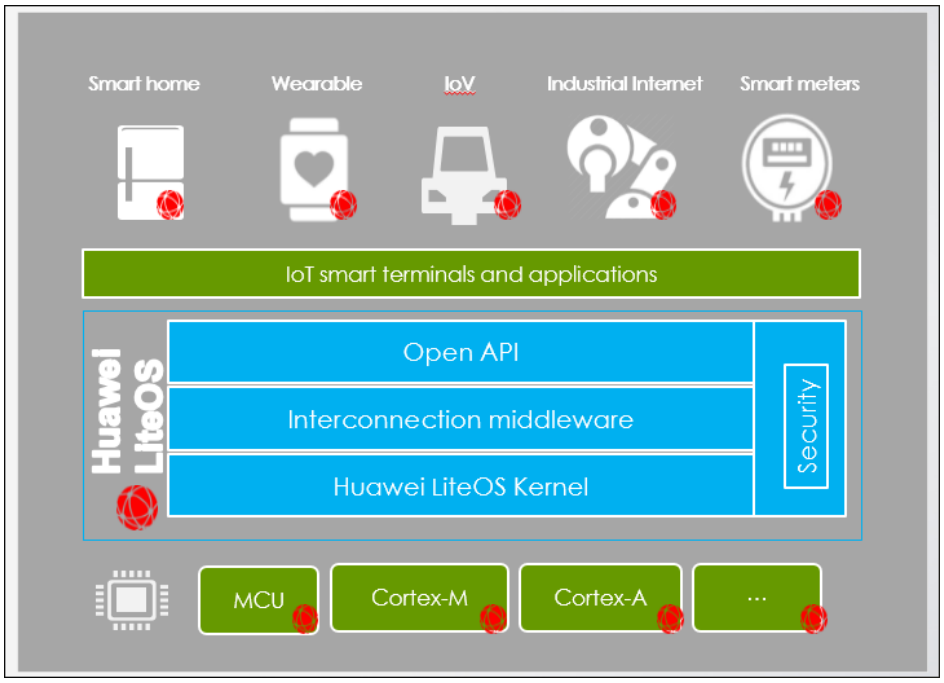
为了方便用户理解，这是协议的概述. 可以访问网址https://creativecommons.org/licenses/by-sa/3.0/legalcode了解完整协议内容.

2 概述

本文主要介绍如何在STM32系列处理器上移植Huawei LiteOS。

3 Huawei LiteOS 简介

Huawei LiteOS是华为轻量级物联网操作系统，其体系架构如下图所示：



Huawei LiteOS由Huawei LiteOS kernel、互联互通中间件、开放API以及安全组成：

- Huawei LiteOS Kernel为Huawei LiteOS基础内核，属于最精简RTOS，包括任务管理、内存管理、时间管理、通信机制、中断管理、队列管理、事件管理、定时器、异常管理等操作系统基础组件，可以单独运行；
- Huawei LiteOS互联互通中间件，可覆盖短距(Wifi、BT等)//广域(4G/NB-IoT)协议，可解决不同协议架构间互联互通、互操作问题；
- Huawei LiteOS提供面向不同IoT领域的业务Profile，并以开放API的方式提供给第三方开发者；
- Huawei LiteOS构建完备的设备侧安全、轻量级E2E传输安全能力。

3.1 Huawei LiteOS 的关键特性

3.2 Huawei LiteOS Kernel 主要模块简介

任务

提供任务的创建、删除、延迟、挂起、恢复等功能，以及锁定和解锁任务调度。任务按优先级可抢占、同优先级时间片轮转调度的方式调度。

任务同步

信号量：支持信号量的创建、删除、PV等功能。

互斥锁：支持互斥锁的创建、删除、PV等功能。

硬件相关

提供中断、定时器等功能。

中断：提供中断的创建、删除、使能、禁止、请求位的清除等功能。

定时器：提供定时器的创建、删除、启动、停止等功能。

IPC通信

提供事件、消息队列功能。

事件：支持读事件和写事件功能。

消息队列：支持消息队列的创建、删除、发送和接收功能。

时间管理

系统时间：系统时间是由定时/计数器产生的输出脉冲触发中断而产生的。

Tick时间：Tick是操作系统调度的基本时间单位，对应的时长由系统主频及每秒Tick数决定，由用户配置。

软件定时器：以Tick为单位的定时器功能，软件定时器的超时处理函数在系统创建的Tick软中断中被调用。

内存管理

提供静态内存和动态内存两种算法，支持内存申请、释放。目前支持的内存管理算法有固定大小的BOX算法、动态申请DLINK算法。

提供内存统计、内存越界检测功能。

异常接管

异常接管是指在系统运行过程中发生异常后，跳转到异常处理信息的钩子函数，打印当前发生异常函数调用栈信息，或者保存当前系统状态的一系列动作。

Huawei LiteOS的异常接管，会在异常后打印发生异常的任务ID号、栈大小，以及LR、PC等寄存器信息。

4 环境准备

4.1 硬件环境

1. 开发板：可选STM32F103、STM32F4、STM32F7全系列芯片，本教程将使用国内主流STM32学习板-秉火指南者STM32F103开发板进行移植，其他STM32开发板移植方法类似。
2. 仿真器：J-Link V9 或者ST-Link

4.2 软件环境

主流的ARM cortex M系列微控制器集成开发环境主要有MDK、IAR；由于华为开发者社区已经提供IAR版本的Huawei LiteOS源码，因此本教程不再介绍IAR环境的移植。本教程的集成开发环境为MDK5.21 ，大家可以去ARM官网下载：**https://www.keil.com/demo/eval/arm.htm**

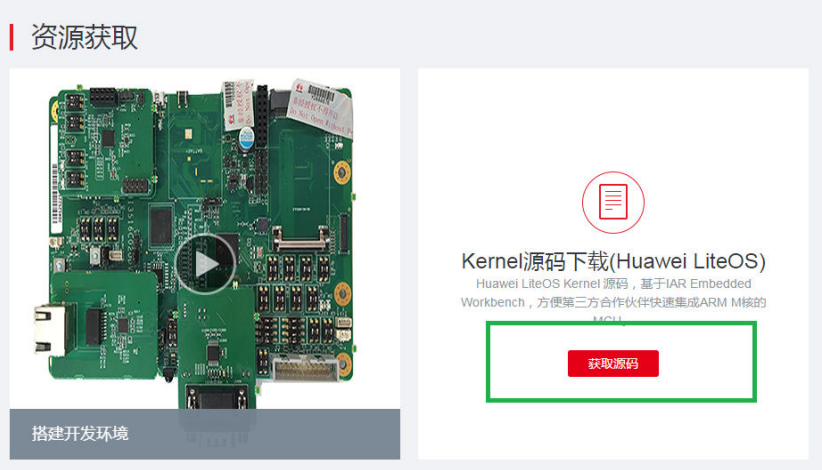
在 MDK5 安装完成后，要让 MDK5 支持 STM32F103 的开发，还要安装 STM32F103 的器件支持包： Keil.STM32F1xx_DFP.2.1.0.pack（ STM32F1 系列的器件包）。下载地址：**http://www.keil.com/dd2/Pack/#/eula-container**

Keil安装pack包的步骤可以从网上搜索了解。

4.3 源代码

1. Huawei LiteOS kernel源码

http://developer.huawei.com/ict/cn/site-iot/product/liteos



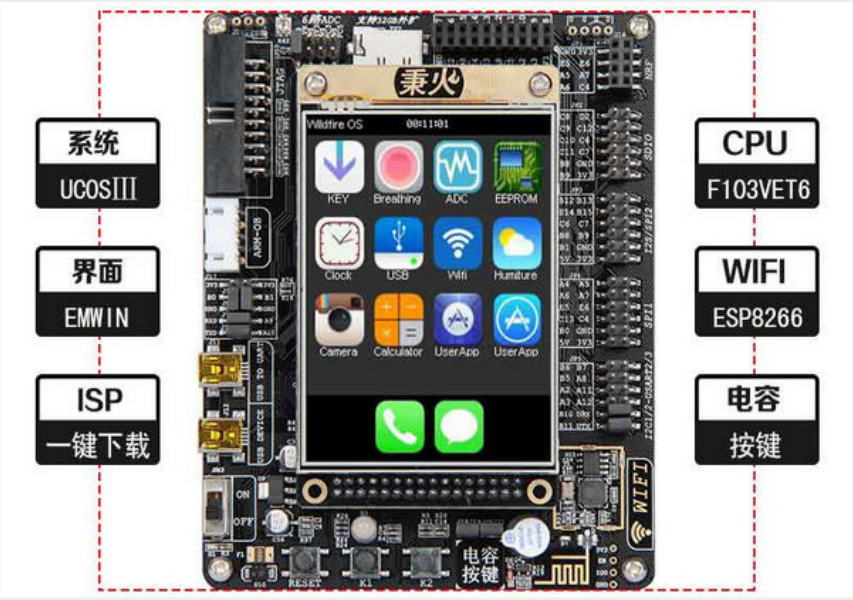
2. 秉火指南者STM32F103开发板开发板资料

http://www.firebbs.cn/forum.php?mod=viewthread&tid=12686&extra=page%3D1%26filter%3Dtypeid%26typeid%3D107

我们只需要下载其中的某个基础例程即可：

秉火F103-指南者-光盘资料（资料盘+视频盘）

百度网盘 下载链接：https://pan.baidu.com/s/1jIuKi2a

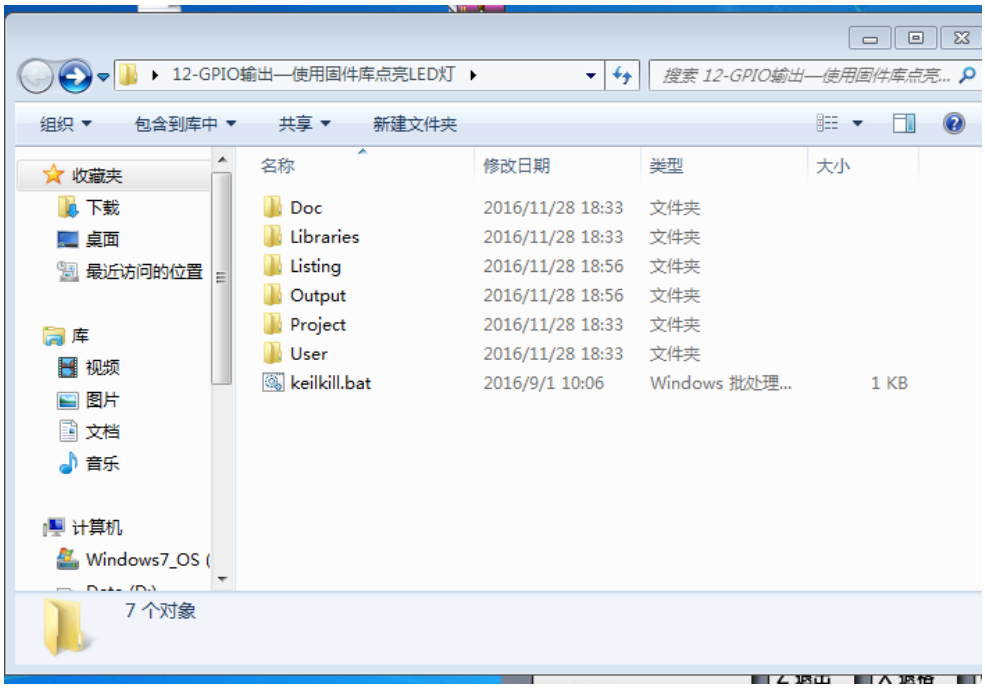


5 Huawei LiteOS 操作系统移植

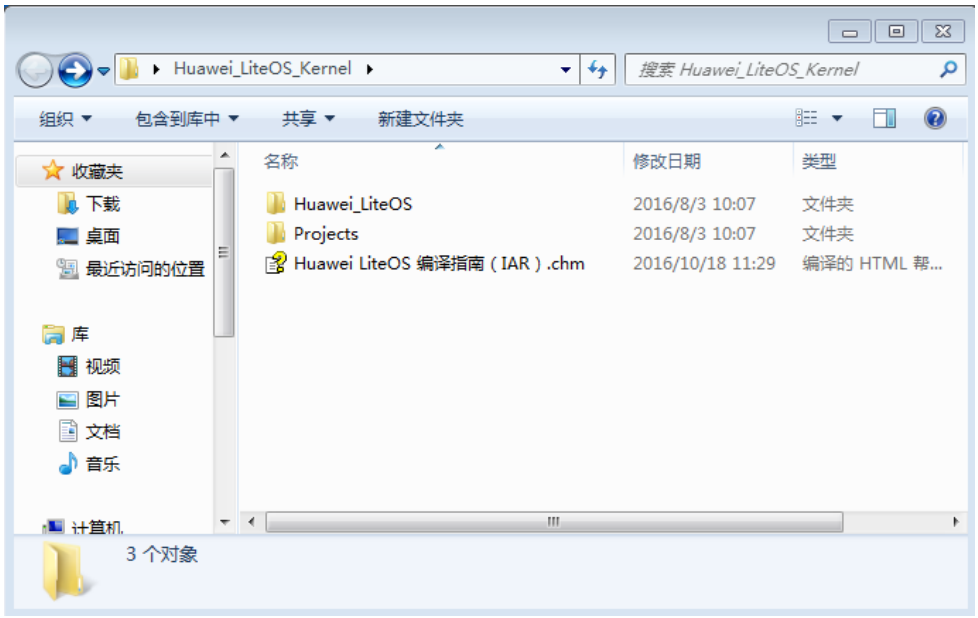
5.1 新建 STM32 裸机工程模板

首先，我们从秉火STM32官方例程中找一个最简单例程作为参考，来新建STM32裸机工程模板，作为Huawei LiteOS移植的基础环境。（当然，大家也可以使用例程中现有的模板工程或者其他开发板的裸机工程模板，不限制）。

这里我们选取GPIO输出—使用固件库点亮LED灯的例程，如图所示：

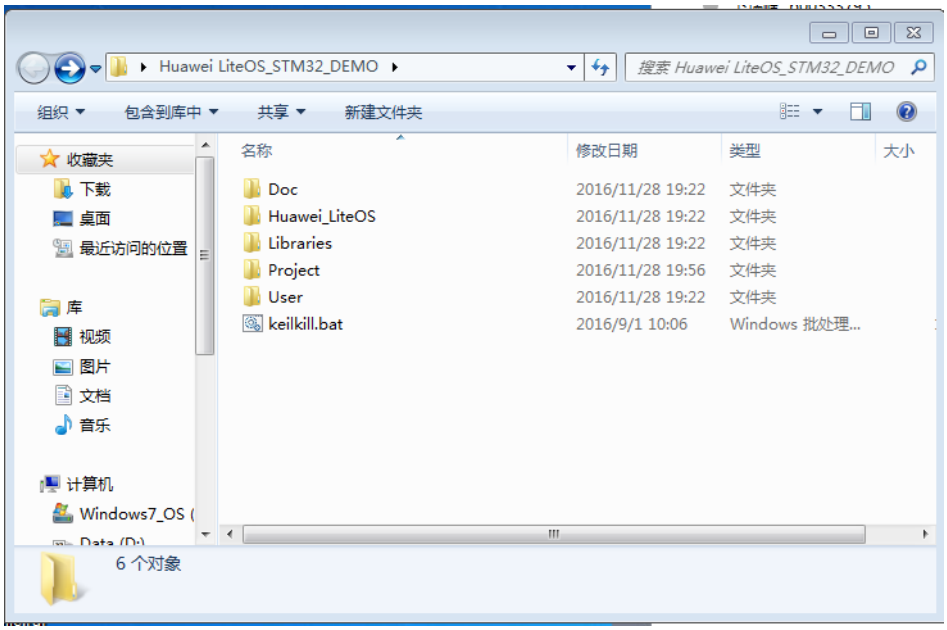


然后解压下载下来的Huawei LiteOS Kernel源码，如下图所示，Huawei_LiteOS文件夹是操作系统内核源码，移植的代码在这里，Project为IAR工程文件，我们移植的软件平台是MDK，不需要关注。



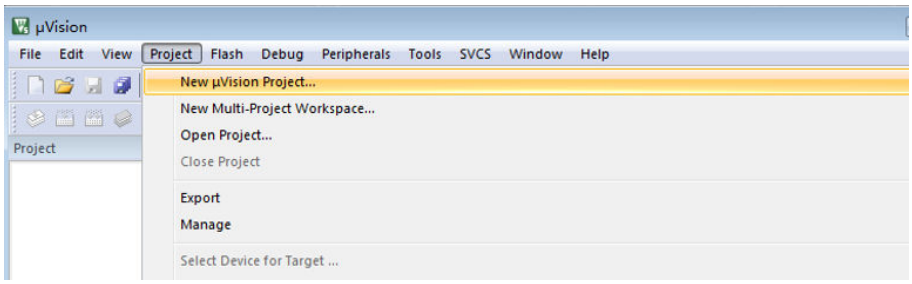
步骤1 新建工程目录

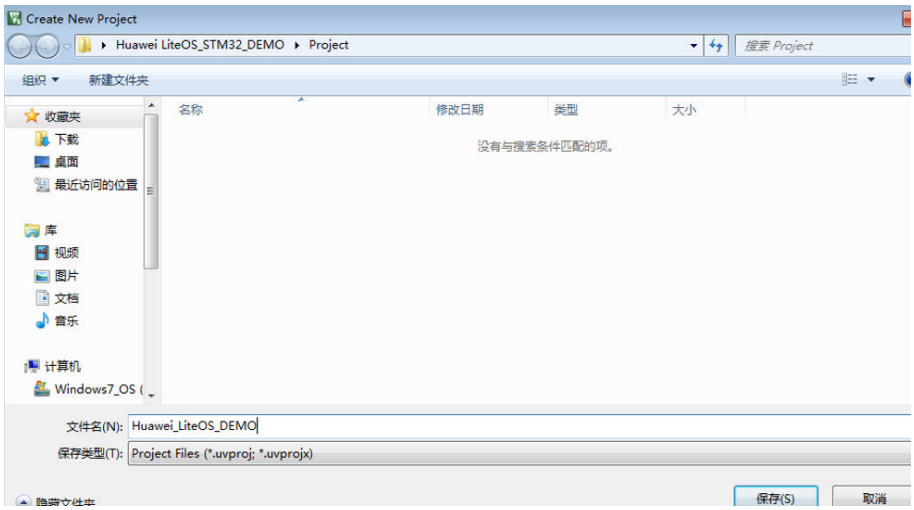
在电脑上新建一个目录，这里命名为Huawei LiteOS_STM32_DEMO，将秉火的LED灯例程的文件全部拷贝到该目录下，并将Huawei LiteOSKernel源码文件夹Huawei_LiteOS也拷贝到该目录下，删除Listing、Output两个文件夹，并将Project文件夹目录下面的文件全部删除，一会我们新建的工程会生成自己的工程文件和输出文件，最终的文件目录如下图所示：



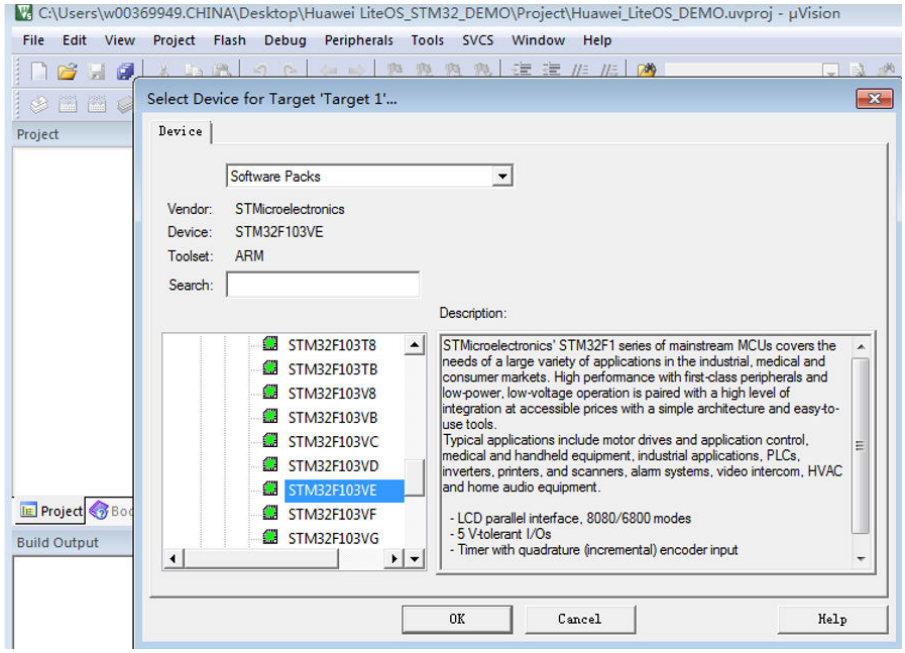
步骤2 添加工程

打开MDK软件，点击菜单Project －>New Uvision Project，然后将目录定位到刚才建立的文件夹 Huawei LiteOS_STM32_DEMO 之下的 Project 子目录， 工程取名为 Huawei_LiteOS_DEMO 之后点击保存， 工程文件就都保存到 Project 文件夹下面。 操作过程如下图所示：

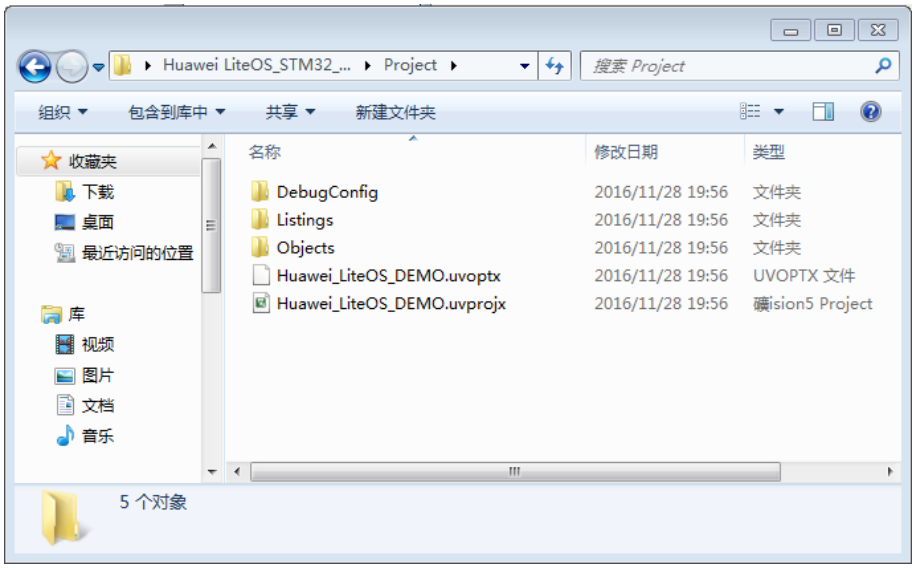




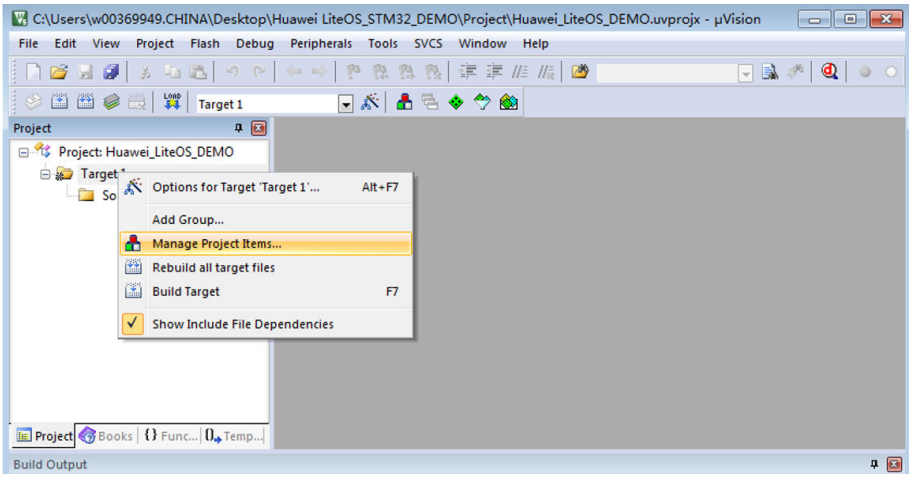
之后会出现一个选择 Device 的界面，选择开发板对应的芯片，秉火指南者开发板使用的是STM32F103VET6，所以我们选择如下图所示，注意，MDK5需要安装器件的pack才会显示这些内容。



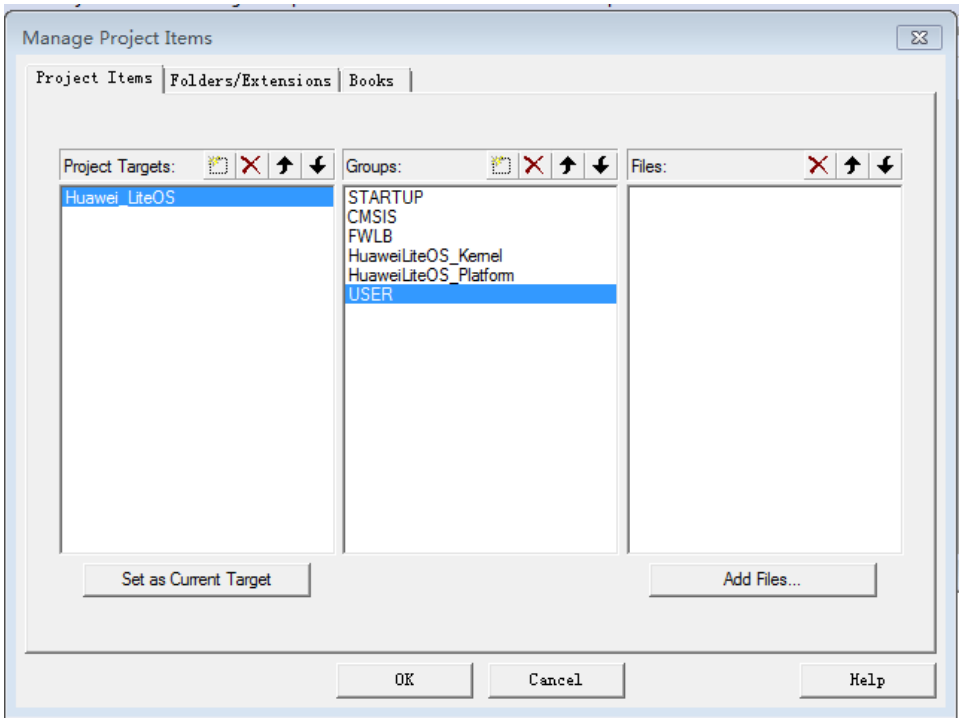
点击 OK， MDK 会弹出 Manage Run-Time Environment 对话框，这里我们不配置，直接跳过即可。这样Project目录下会生成以下文件，其中Huawei_LiteOS_DEMO.uvprojx是工程文件，DebugConfig， Listings 和 Objects 三个文件夹是 MDK 自动生成的文件夹。其中 DebugConfig 文件夹用于存储一些调试配置文件， Listings 和 Objects 文件夹用来存储 MDK 编译过程的一些中间文件。

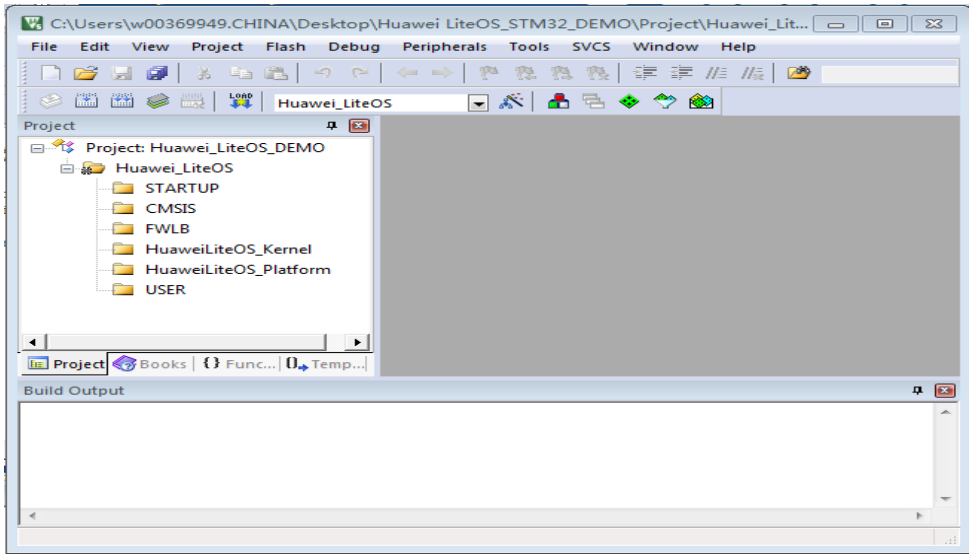


步骤3 配置工程

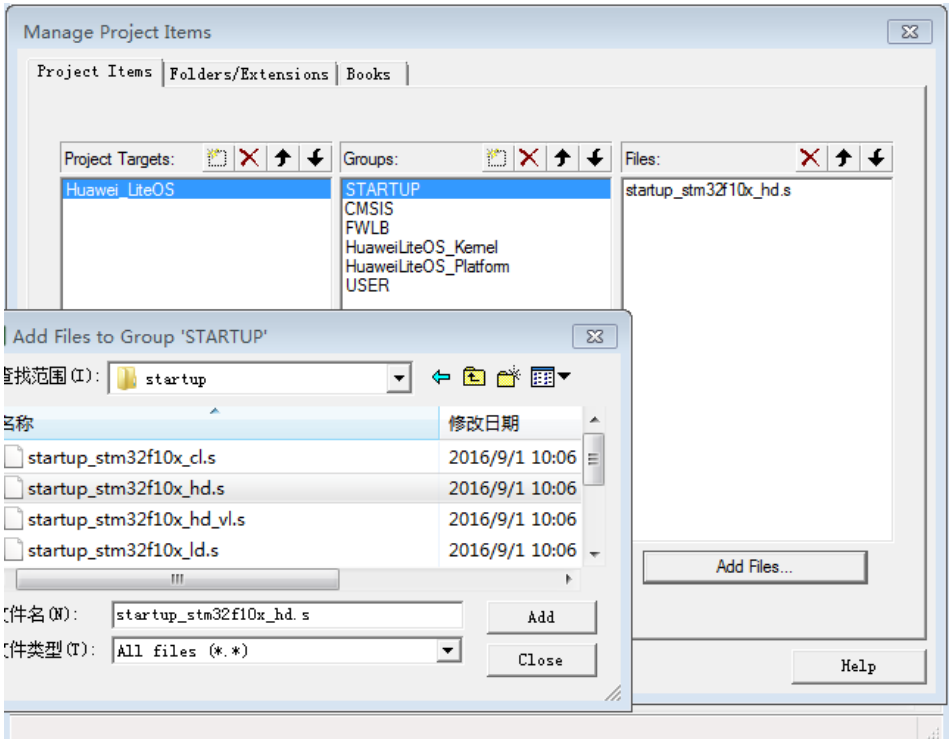


工程中右键点击 Target1，选择 Manage Project Items，如上图所示，接下来弹出新的对话框，在Project Targets 一栏，我们将 Target 名字修改为Huawei_LiteOS,然后在 Groups 一栏删掉 Source Group1，建立六个 Groups：分别为 STARTUP、CMSIS、FWLB、HuaweiLiteOS_Kernel、HuaweiLiteOS_Platform、USER六个文件夹。然后点击 OK，可以看到我们的 Target 名字以及 Groups 情况如下图：



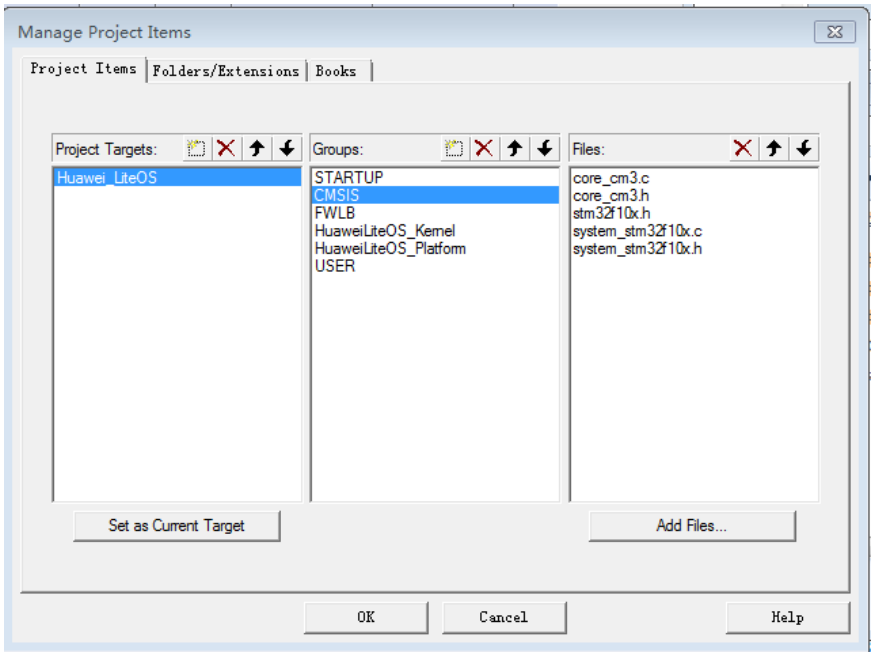


接下来，我们开始往Group里面添加我们所需要的文件。我们重复上个步骤，选择Manage Project Items，然后选择需要添加文件的Group，这里我们第一步选择STARTUP，点击右下角的Add Files，在出来的对话框内指定文件类型为All files，指定到Huawei LiteOS_STM32_DEMO\Libraries\CMSIS\startup目录下，将STM32芯片的汇编启动文件startup_stm32f10x_hd.s添加进来（注意：只添加这一个文件），如下图所示

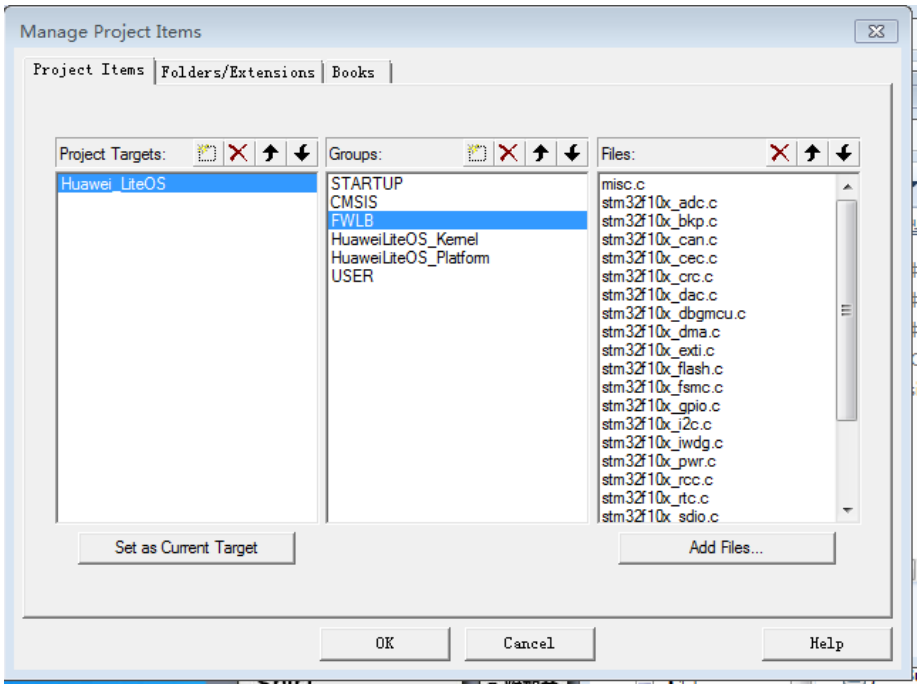


然后继续按照同样的方法给其余5个文件夹添加文件：

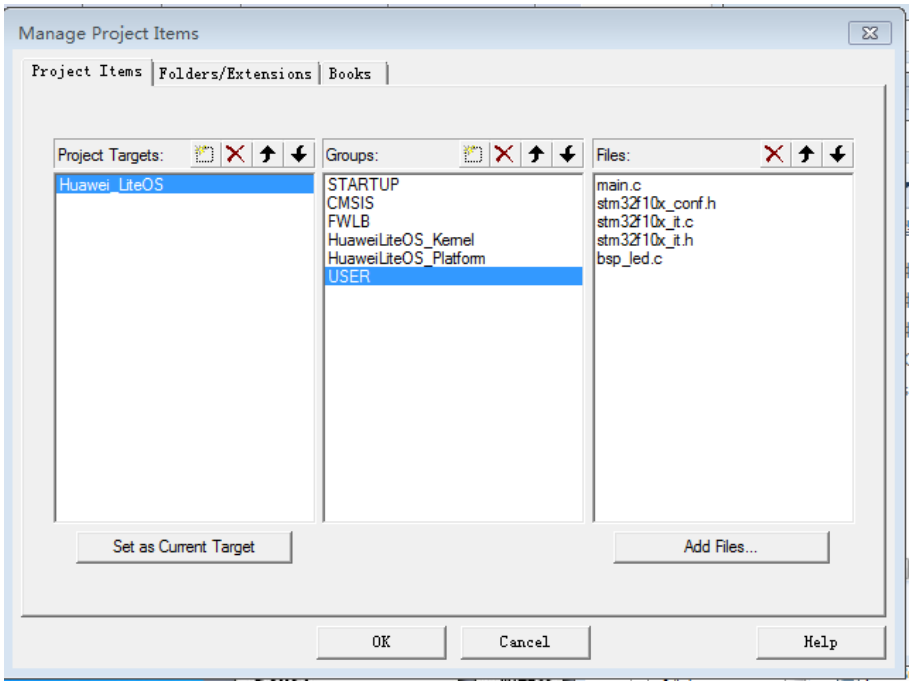
CMSIS为ARM提供的库文件，在Huawei LiteOS_STM32_DEMO\Libraries\CMSIS目录下，全部添加到CMSIS这个Group：



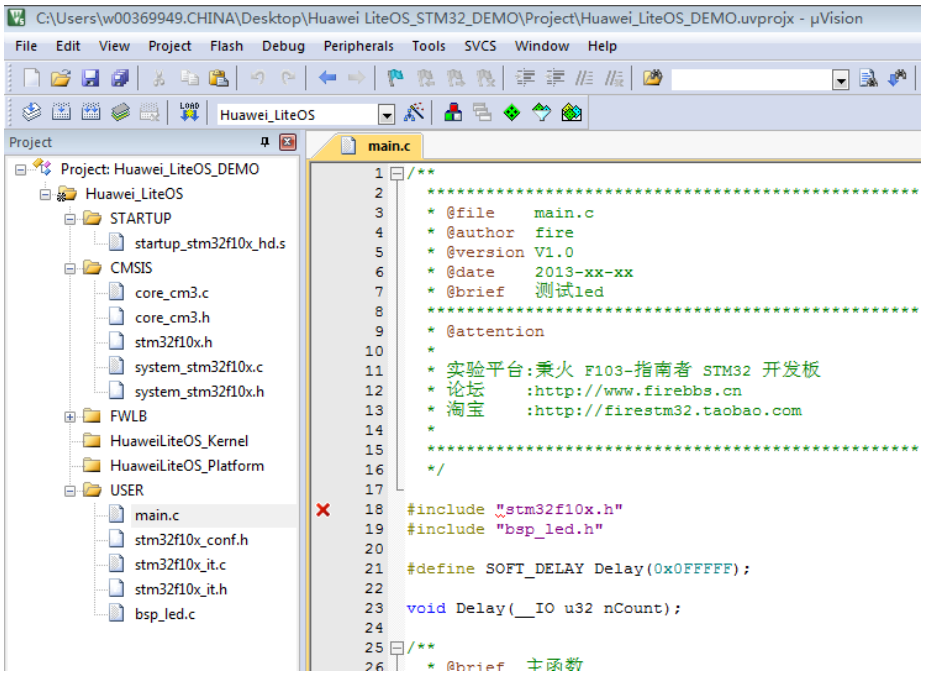
FWlib为STM32官方提供的驱动库，在Huawei LiteOS_STM32_DEMO\Libraries\FWlib目录下，我们把src目录下的c文件全部添加到FWLB这个Group:



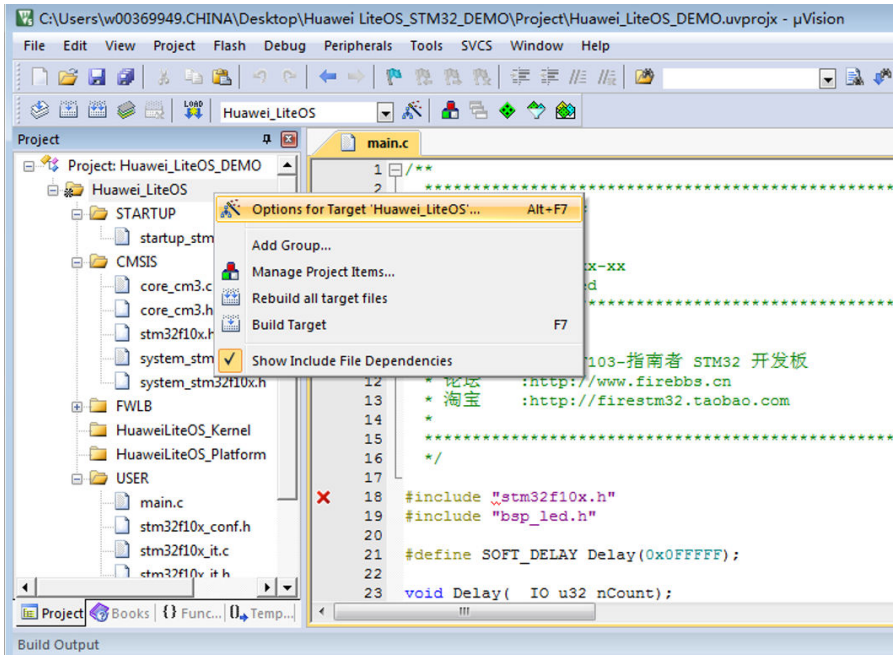
USER目录为用户应用代码，这里将秉火LED示例应用程序文件添加进来，在Huawei LiteOS_STM32_DEMO\User目录下，其中led驱动在led子目录下，一并添加进来：



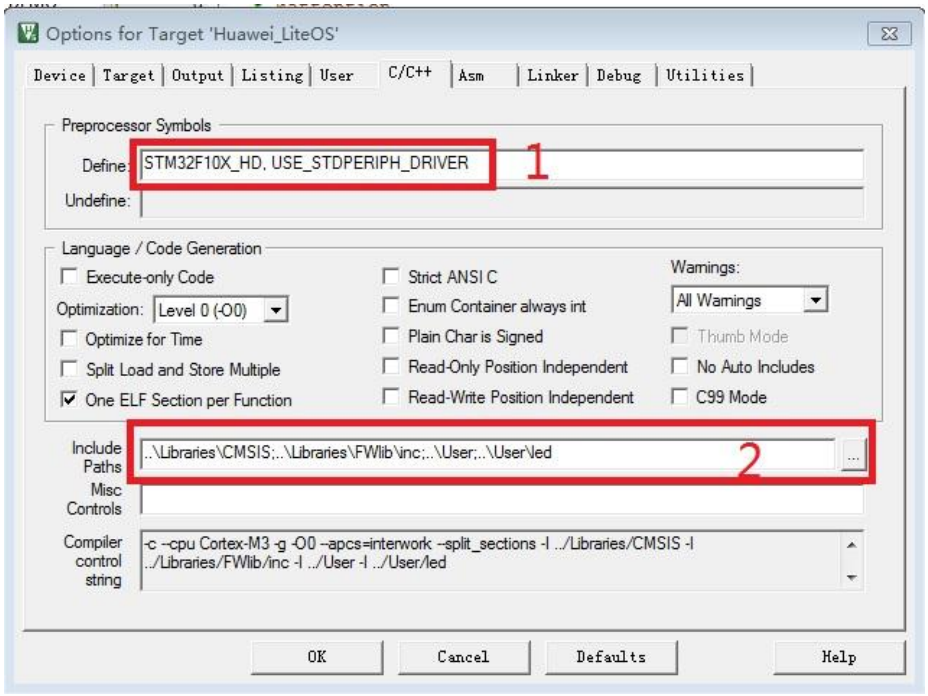
添加完所有文件到工程后，我们点击OK，回到MDK主界面，可以看到工程文件目录如下图所示：



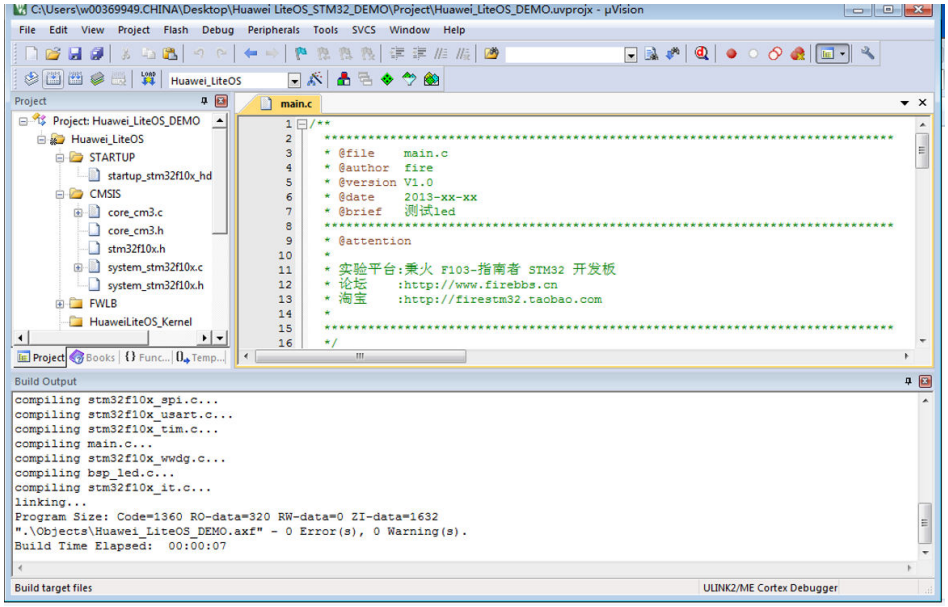
接下来，我们需要在MDK里面设置头文件存放路径，也就是告诉编译器去哪里找对应的头文件，如图所示，工程上右键选择Options for Target…，弹出工程配置对话框：



在工程配置对话框里面切到C++选项框，如下图所示，设置宏定义STM32F10X_HD, USE_STDPERIPH_DRIVER，并添加头文件路径。



完成后点击OK，回到主界面，按F7或者编译按钮进行工程编译，如果配置正确，应该会出现如下图所示的编译结果：



----结束

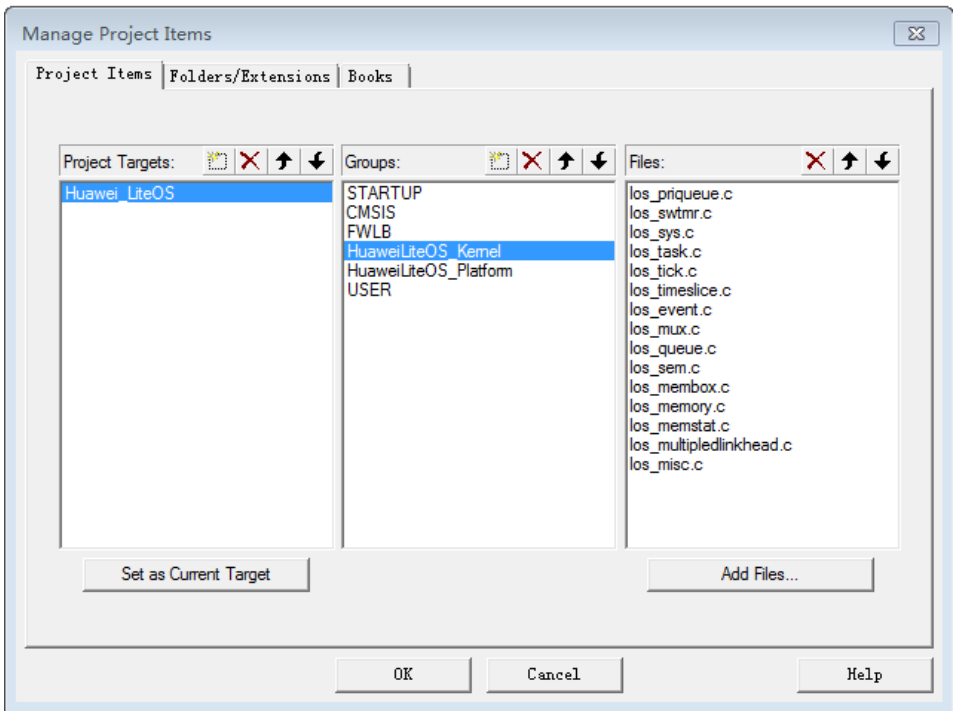
5.2 基于裸机模板工程，编译 Huawei LiteOS 内核代码

接下来，我们开始正式移植Huawei LiteOS

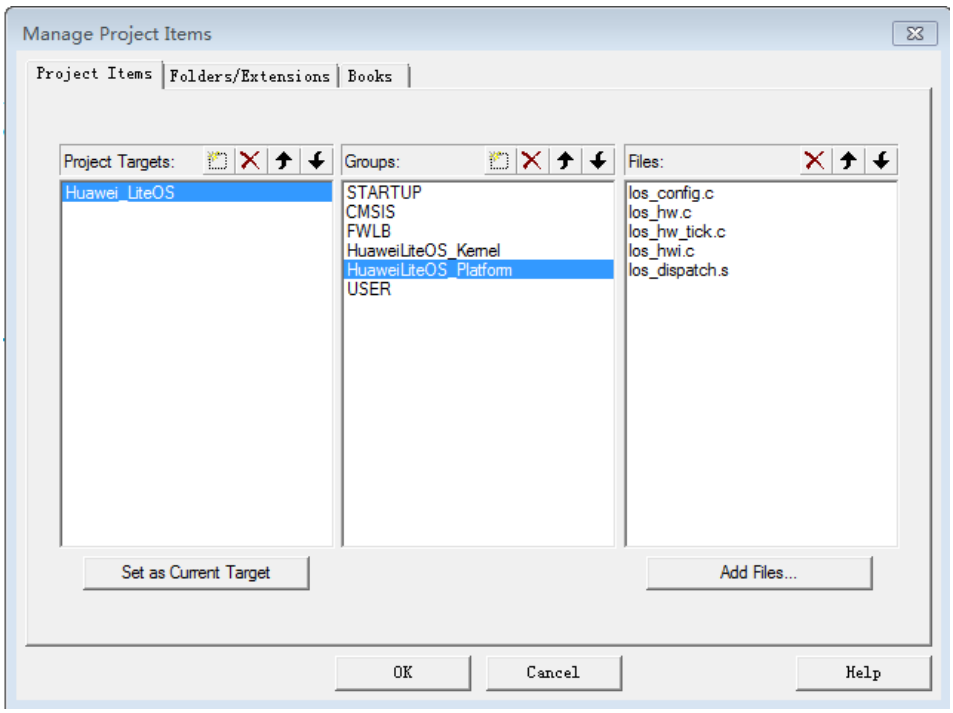
步骤1 向裸机模板工程中添加Huawei LiteOS源码

3.1中我们已经新建了HuaweiLiteOS_Kernel、HuaweiLiteOS_Platform两个Group，但是并木有添加源码，现在我们来一起完成。

HuaweiLiteOS_Kernel这个Group用来存放内核源码，移植的时候一般不需要修改内核源代码，内核源码位于Huawei LiteOS_STM32_DEMO\Huawei_LiteOS\kernel\base目录下，我们把子目录core、ipc、mem、misc目录下的c文件全部添加进来，一共15个文件，如下图所示：

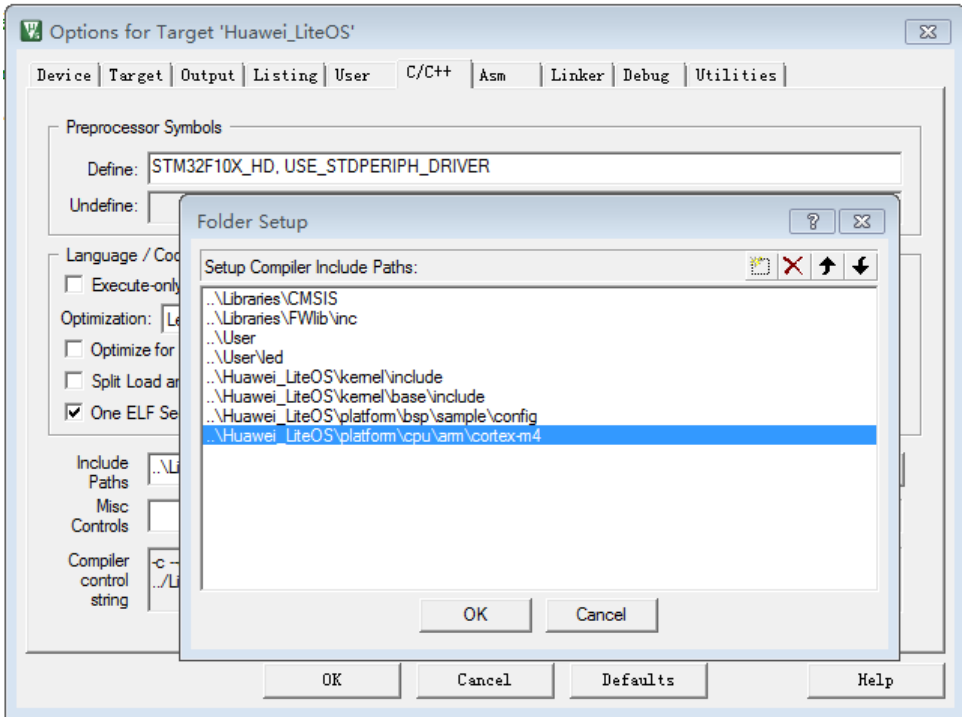


HuaweiLiteOS_Platform这个Group用来存放CPU相关配置文件以及操作系统的配置文件，源码位于Huawei LiteOS_STM32_DEMO\Huawei_LiteOS\platform目录下，我们把子目录bsp\sample\config下的los_config.c添加进来，再把cpu\arm\cortex-m4子目录下的los_dispatch.s、los_hw.c、los_hw_tick.c、los_hwi.c添加进来，一共5个文件，如下图所示：



步骤2 添加Huawei LiteOS头文件目录

按照3.1提到的方法，在工程上右键选择Options for Target…，然后在弹出工程配置对话框中选择C++选项卡，添加包含路径,工程的所有包含目录如下图所示：



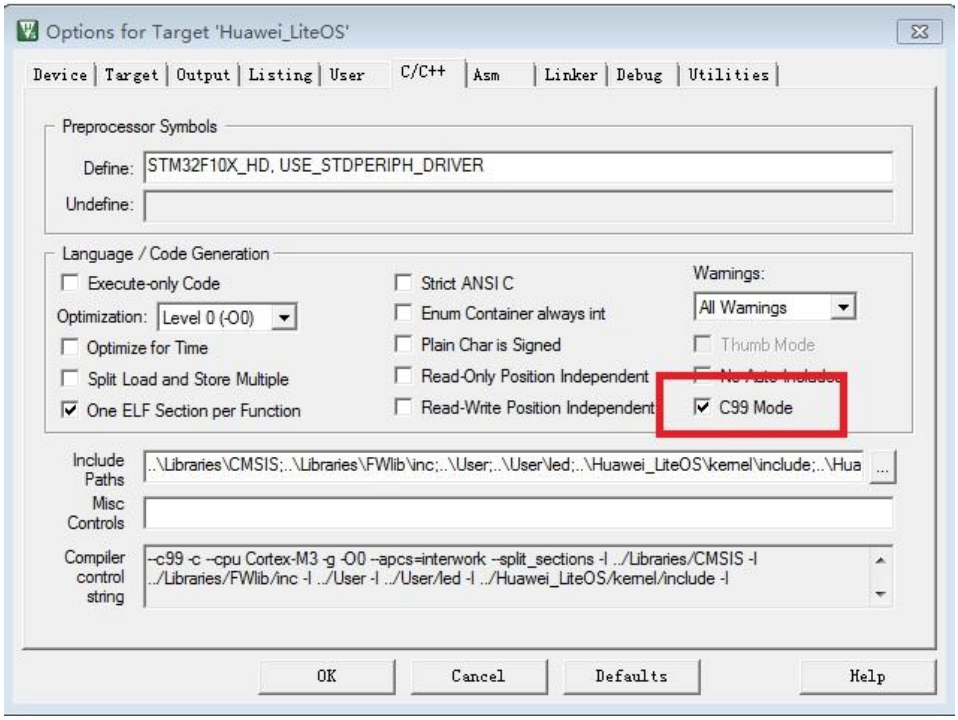
点击OK完成配置，回到主界面按F7进行编译，你会发现有200多个编译错误，不用着急，移植的过程肯定会遇到各种问题的，我们要一个个消灭它们，在解决error和warning的同时不断学习新知识。

步骤3 解决编译错误

错误1：

LITE_OS_SEC_ALW_INLINE INLINE 不识别，报错原因为Huawei LiteOS在los_builddef.h文件中宏定义了#define INLINE static inline，MDK默认环境下不认识static inline，我们只需要在工程中配置C99标准就可以支持了.

在工程上右键选择Options for Target…，然后在弹出工程配置对话框中选择C++选项卡，勾选C99 Mode, 如下图所示：



错误2：

..\Huawei_LiteOS\platform\bsp\sample\config\los_config.h(710): warning: #1295-D: Deprecated declaration osBackTrace - give arg types

解决办法：在los_config.h的710行代码中函数声明加上参数类型，改成extern void osBackTrace(void);

错误3：

..\Huawei_LiteOS\platform\cpu\arm\cortex-m4\los_hwi.c(81): error: #18: expected a ")" __asm ("mrs %0, ipsr" : "=r" (uwIntNum)); 这个是因为官方提供的是IAR工程，IAR的内嵌汇编跟MDK有区别

解决办法：将los_hwi.c文件中77行的函数LITE_OS_SEC_TEXT_MINOR UINT32 osIntNumGet(VOID)按照下面给出的进行修改：

```
70 /*****************************************************************************
71  Function    : osIntNumGet
72  Description : Get a interrupt number
73  Input       : None
74  Output      : None
75  Return      : Interrupt Indexes number
76  *****************************************************************************/
77 LITE_OS_SEC_TEXT_MINOR __asm UINT32 osIntNumGet(VOID)
78 {
79     MRS R0, IPSR
80     BX LR
81 }
82
```

错误4：

..\Huawei_LiteOS\platform\cpu\arm\cortex-m4\los_hw.c(99): error: #18: expected a ")" __asm ("cpsid i" : : : "memory"); 跟错误3类似

解决办法： los_hw.c 90行处的函数修改如下：

```
/*****************************************************************************
 Function    : osTaskExit
 Description : Task exit function
 Input       : None
 Output      : None
 Return      : None
 *****************************************************************************/
LITE_OS_SEC_TEXT_MINOR VOID osTaskExit(VOID)
{
    __disable_irq();
    while(1);
}
```

另外los_hw.c文件的112行的osTskStackInit函数中请将浮点运算相关的代码注释掉，因为我们移植的是M3处理器，如果是M4或者M7可以不做修改。注释只需将125行和160行的 #if 1改成#if 0即可。对应的请修改los_hw.h文件中的TSK_CONTEXT_S结构体，删除浮点相关寄存器，具体修改如下图所示：

```
typedef struct tagTskContext
{
    UINT32 uwR4;
    UINT32 uwR5;
    UINT32 uwR6;
    UINT32 uwR7;
    UINT32 uwR8;
    UINT32 uwR9;
    UINT32 uwR10;
    UINT32 uwR11;
    UINT32 uwPriMask;
    UINT32 uwR0;
    UINT32 uwR1;
    UINT32 uwR2;
    UINT32 uwR3;
    UINT32 uwR12;
    UINT32 uwLR;
    UINT32 uwPC;
    UINT32 uwxPSR;
} TSK_CONTEXT_S;
```

错误5：

..\Huawei_LiteOS\platform\cpu\arm\cortex-m4\los_dispatch.s(54): error: A1163E: Unknown opcode SECTION , expecting opcode or Macro 原因：MDK和IAR汇编的字段定义方法不同，需要修改，同时los_dispatch.s文件为操作系统调度的主要文件，M3和M4的支持的指定集会有一些不同，这里一并进行修改：

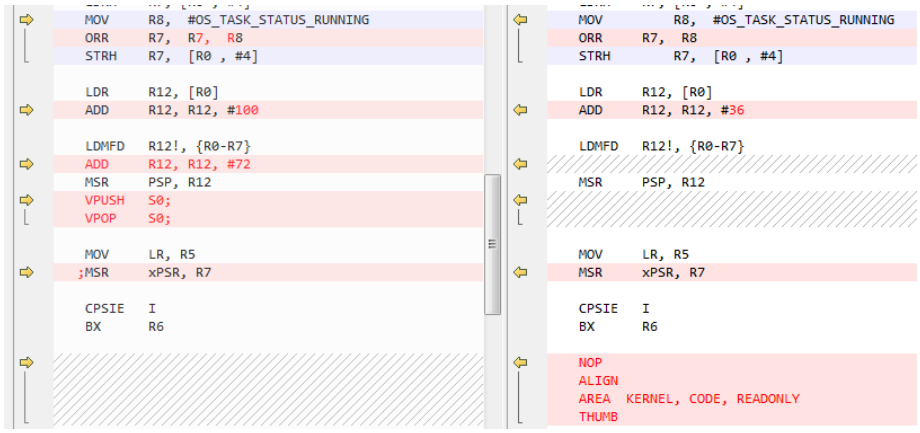
解决办法：请参考下图，将左边的改成右边的

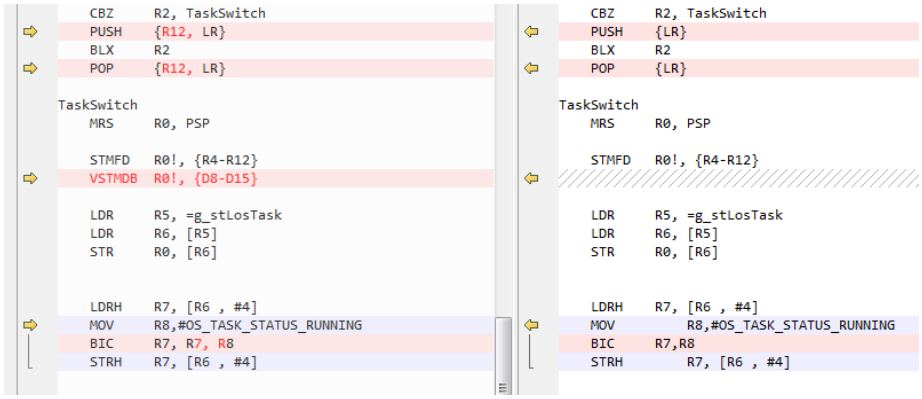
```
    PRESERVE8

    EXPORT  LOS_IntLock
    EXPORT  LOS_IntUnLock
    EXPORT  LOS_IntRestore
    EXPORT  LOS_StartToRun
    EXPORT  osTaskSchedule
    EXPORT  osPendSV

    IMPORT  g_stLosTask
    IMPORT  g_pfnTskSwitchHook
    IMPORT  g_bTaskScheduled
```

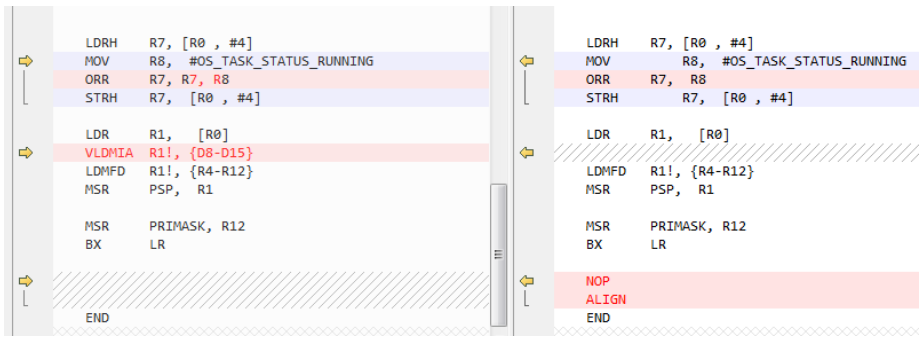
```
    PRESERVE8

    EXPORT  LOS_IntLock
    EXPORT  LOS_IntUnLock
    EXPORT  LOS_IntRestore
    EXPORT  LOS_StartToRun
    EXPORT  osTaskSchedule
    EXPORT  PendSV_Handler

    IMPORT  g_stLosTask
    IMPORT  g_pfnTskSwitchHook
    IMPORT  g_bTaskScheduled
```

```
OS_NVIC_PENDSVSET       EQU     0x10000000
OS_TASK_STATUS_RUNNING  EQU     0x0010

    SECTION    .text:CODE(2)
    THUMB
    REQUIRE8

LOS_StartToRun
```

```
OS_NVIC_PENDSVSET       EQU     0x10000000
OS_TASK_STATUS_RUNNING  EQU     0x0010

    AREA  |.text|, CODE, READONLY
    THUMB

LOS_StartToRun
```







修改后的汇编代码如下：

```
PRESERVE8
        EXPORT  LOS_IntLock
        EXPORT  LOS_IntUnLock
        EXPORT  LOS_IntRestore
        EXPORT  LOS_StartToRun
        EXPORT  osTaskSchedule
        EXPORT  PendSV_Handler
        IMPORT  g_stLosTask
        IMPORT  g_pfnTskSwitchHook
        IMPORT  g_bTaskScheduled
OS_NVIC_INT_CTRL            EQU     0xE000ED04
OS_NVIC_SYSPRI2             EQU     0xE000ED20
OS_NVIC_PENDSV_PRI          EQU     0xF0F00000
OS_NVIC_PENDSVSET           EQU     0x10000000
OS_TASK_STATUS_RUNNING      EQU     0x0010
    AREA  |.text|, CODE, READONLY
    THUMB
LOS_StartToRun
    LDR     R4, =OS_NVIC_SYSPRI2
    LDR     R5, =OS_NVIC_PENDSV_PRI
```

```
    STR     R5, [R4]
    LDR     R0, =g_bTaskScheduled
    MOV     R1, #1
    STR     R1, [R0]
    MOV     R0, #2
    MSR     CONTROL, R0
    LDR     R0, =g_stLosTask
    LDR     R2, [R0, #4]
    LDR     R0, =g_stLosTask
    STR     R2, [R0]
    LDR     R3, =g_stLosTask
    LDR     R0, [R3]
    LDRH    R7, [R0 , #4]
    MOV R8,  #OS_TASK_STATUS_RUNNING
    ORR     R7,  R8
    STRH R7,  [R0 , #4]
    LDR     R12, [R0]
    ADD     R12, R12, #36
    LDMFD   R12!, {R0-R7}
    MSR     PSP, R12
    MOV     LR, R5
    MSR     xPSR, R7
    CPSIE   I
    BX      R6
    NOP
    ALIGN
    AREA  KERNEL, CODE, READONLY
    THUMB
LOS_IntLock
    MRS     R0, PRIMASK
    CPSID   I
    BX      LR
LOS_IntUnLock
    MRS     R0, PRIMASK
    CPSIE   I
    BX      LR
LOS_IntRestore
    MSR     PRIMASK, R0
    BX      LR
osTaskSchedule
    LDR     R0, =OS_NVIC_INT_CTRL
    LDR     R1, =OS_NVIC_PENDSVSET
    STR     R1, [R0]
    BX      LR
PendSV_Handler
    MRS     R12, PRIMASK
    CPSID   I
    LDR     R2, =g_pfnTskSwitchHook
    LDR     R2, [R2]
    CBZ     R2, TaskSwitch
    PUSH    {LR}
    BLX     R2
    POP     {LR}
TaskSwitch
    MRS     R0, PSP
    STMFD   R0!, {R4-R12}
    LDR     R5, =g_stLosTask
    LDR     R6, [R5]
    STR     R0, [R6]
    LDRH    R7, [R6 , #4]
    MOV R8,#OS_TASK_STATUS_RUNNING
    BIC     R7,R8
    STRH R7, [R6 , #4]
    LDR     R0, =g_stLosTask
    LDR     R0, [R0, #4]
    STR     R0, [R5]
    LDRH    R7, [R0 , #4]
    MOV R8,  #OS_TASK_STATUS_RUNNING
    ORR     R7,  R8
```

```
    STRH R7,  [R0 , #4]
    LDR     R1,   [R0]
    LDMFD   R1!, {R4-R12}
    MSR     PSP,  R1
    MSR     PRIMASK, R12
    BX      LR
    NOP
    ALIGN
    END
```

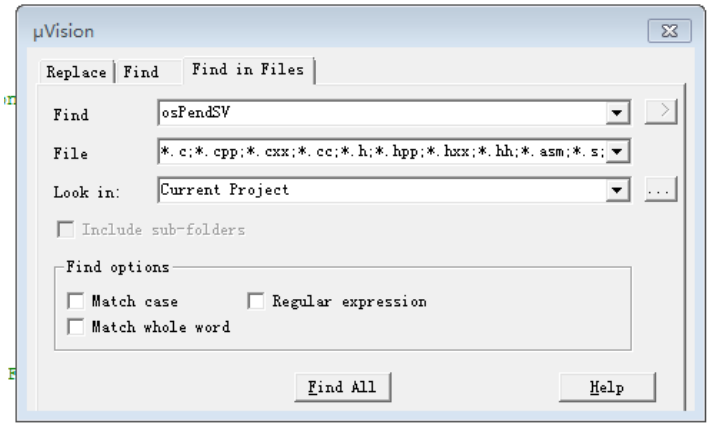
警告1：

..\Huawei_LiteOS\platform\cpu\arm\cortex-m4\los_hwi.c(48): warning: #161-D: unrecognized #pragma

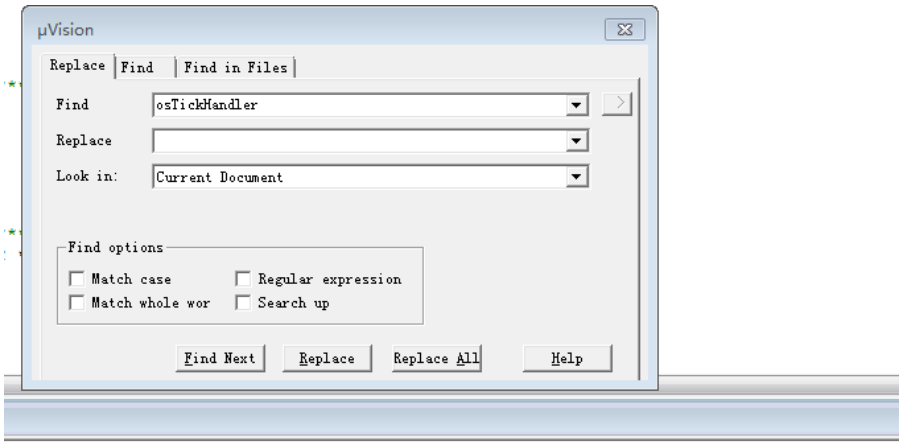
解决办法：注释掉48行的#pragma location = ".vector"

步骤4 根据STM32启动文件，修改相应平台文件

除了编译错误之外，我们还需要根据STM32启动文件修改PendSV_Handler异常向量和SysTick_Handler向量的名称，在Huawei LiteOS源码中，他们分别叫osPendSV、osTickHandler。这里我们在MDK主界面下使用Ctrl+H将查找到的osPendSV和osTickHandler都分别替换成PendSV_Handler和SysTick_Handler。如图所示：




```
,arm\cortex-m4\los_hwi.h(582) : extern VOID  osPendSV(VOID);
,arm\cortex-m4\los_hwi.c(65) :   osPendSV,            // [14] PendSV Handler
,arm\cortex-m4\los_dispatch.s(119) : osPendSV
```




```
:ortex-m4\los_hw_tick.c(68) :     m_pstHwiSlaveForm[OS_EXC_SYS_TICK] = osTickHandler;
.os_tick.c(60) : LITE_OS_SEC_TEXT VOID osTickHandler(VOID)
```

注意，查找的时候*.ph类型文件也要查找，比如los_tick.ph文件中的extern VOID osTickHandler(VOID);也要改成extern VOID SysTick_Handler(VOID)。

los_hwi.h文件中的243行改成extern UINT32 __Vectors[];与启动文件对应起来；

同时，请将STM32F10X_it.c文件中的void PendSV_Handler(void)和void SysTick_Handler(void);两个函数前面加上__weak关键字，定义为弱函数，否则会跟Huawei LiteOS源码中的同名函数重定义，或者你也可以选择将这两个函数注释掉。

修改完成后，再次编译应该只有2个错误：

.\Objects\demo.axf: Error: L6200E: Symbol __ARM_use_no_argv multiply defined (by main.o and los_config.o).

.\Objects\Huawei_LiteOS_DEMO.axf: Error: L6200E: Symbol main multiply defined (by main.o and los_config.o).

报错原因是因为裸机工程的main.c和Huawei LiteOS中的los_config.c都定义了main函数，这个问题，我们一会新建操作系统任务的时候会一并修改掉。

----结束

5.3 在 los_config.h 中配置系统参数

常用参数如下，这里示例采用的是STM32F103芯片，因此将OS_SYS_CLOCK设为系统主频72Mhz。

```
#DEFINE OS_SYS_CLOCK 72000000
#DEFINE LOSCFG_BASE_CORE_TSK_LIMIT 15
#DEFINE OS_SYS_MEM_SIZE 0X00008000
#DEFINE LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE SIZE(0X2D0)
#DEFINE LOSCFG_BASE_CORE_SWTMR_LIMIT 16
```

5.4 创建 Huawei LiteOS 任务，实现 LED 指示灯 DEMO

首先，我们了解下Huawei LiteOS启动流程

STM32芯片上电启动后，先执行startup.s汇编文件中的代码，执行堆和栈的初始化配置、中断向量表的配置，然后将程序引导到los_config.c文件中main( )函数，开始进行Huawei LiteOS初始化，main函数中实现Huawei LiteOS系统任务注册，内存池初始化等，然后调用Huawei LiteOS提供给开发者的入口函数osAppInit()。开发者需要在用户程序代码中实现osAppInit()函数，在此函数中添加用户任务。Huawei LiteOS的main函数会调用LOS_Start()函数来启动Huawei LiteOS任务调度。

注意

此处仅对Huawei LiteOS的启动做简单介绍，详细流程开发者请参考华为开发者社区的API文档，下载地址：**http://developer.huawei.com/cn/ict/products/iot/components/liteos/content/api**

Huawei LiteOS是一个支持多任务的操作系统。在Huawei LiteOS中，一个任务表示一个线程。任务可以使用或等待CPU、使用内存空间等系统资源，并独立于其它任务运行。Huawei LiteOS可以给开发者提供多个任务，实现了任务之间的切换和通信，帮助开发者管理业务程序流程。这样可以将更多的精力投入到业务功能的实现中。

在Huawei LiteOS中，我们通过函数LOS_TaskCreate()来创建任务，LOS_TaskCreate()函数原型在los_task.c文件中定义。调用LOS_TaskCreate()创建一个任务以后，任务就会进入就绪状态。

我们在原裸机工程的main.c文件中编写任务代码来创建任务，先删除main.c文件中原有代码，然后按照如下流程进行任务创建：

步骤1 开发者编写用户任务函数

如下图，我们创建一个LED指示灯任务

```
31  VOID task1(void)
32  {
33    UINT32 uwRet = LOS_OK;
34    UINT32 count=0;
35    while(1)
36    {
37      count++;
38      //printf("this is task 1,count is %d\r\n",count);//串口打印计数值
39      LED1_TOGGLE;//LED指示灯翻转
40      uwRet = LOS_TaskDelay(1000);//操作系统延时函数
41      if(uwRet !=LOS_OK)
42        return;
43    }
44  }
45
```

步骤2 配置任务参数并创建任务，如图所示：

```
UINT32 creat_task1(void)
{
  UINT32 uwRet = LOS_OK;
  TSK_INIT_PARAM_S task_init_param;//任务参数配置结构体
  task_init_param.usTaskPrio = 0;//任务优先级
  task_init_param.pcName = "task1";//任务名
  task_init_param.pfnTaskEntry = (TSK_ENTRY_FUNC)task1;//指定入口函数
  //设置任务堆栈大小
  task_init_param.uwStackSize = LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE;
  //设置任务执行完毕后自动删除，默认值，一般不修改
  task_init_param.uwResved = LOS_TASK_STATUS_DETACHED;
  //调用create函数创建任务
  uwRet = LOS_TaskCreate(&g_TestTskHandle,&task_init_param);
  if(uwRet !=LOS_OK)
  {
    return uwRet;
  }
  return uwRet;
}
```

步骤3 在Huawei LiteOS提供的用户入口函数osAppInit()中初始化硬件及用户应用，添加前面已经创建的全部任务，等待操作系统启动后进行任务调度。

```
UINT32 osAppInit(void)
{
  UINT32 uwRet = 0;
  hardware_init();//硬件模块初始化
  uwRet = creat_task1();//添加任务1
  if(uwRet !=LOS_OK)
  {
    return uwRet;
  }

  uwRet = creat_task2();//添加任务2
  if(uwRet !=LOS_OK)
  {
    return uwRet;
  }
  return LOS_OK;
}
```

最终的main.c文件中没有main函数，只有任务创建相关函数，全部代码附件如下：

```
/* Includes ------------------------------------------------------------------*/
// Huawei LiteOS相关头文件
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "los_base.h"
#include "los_config.h"
#include "los_typedef.h"
#include "los_hwi.h"
#include "los_task.ph"
#include "los_sem.h"
#include "los_event.h"
#include "los_memory.h"
#include "los_queue.ph"
// STM32硬件及驱动相关头文件
#include "stm32f10x.h"
#include "bsp_led.h"
void Delay(__IO u32 nCount);
#define SOFT_DELAY Delay(0x0FFFFF);
UINT32 g_TestTskHandle;
void Delay(__IO uint32_t nCount) //简单延时函数
{
```

```
for(; nCount != 0; nCount--);
}
void hardware_init(void)
{
/* LED端口初始化 */
LED_GPIO_Config();
}
VOID task1(void)
{
UINT32 uwRet = LOS_OK;
UINT32 count=0;
while(1)
{
count++;
//printf("this is task 1,count is %d\r\n",count);//串口打印计数
LED1_TOGGLE;//LED指示灯翻转
uwRet = LOS_TaskDelay(1000);//操作系统延时
if(uwRet !=LOS_OK)
return;
}
}

UINT32 creat_task1(void)
{
UINT32 uwRet = LOS_OK;
TSK_INIT_PARAM_S task_init_param;
task_init_param.usTaskPrio = 0;//任务优先级
task_init_param.pcName = "task1";//任务名
task_init_param.pfnTaskEntry = (TSK_ENTRY_FUNC)task1;//指定任务入口函数
task_init_param.uwStackSize = LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE;//设置任务堆栈大小
task_init_param.uwResved = LOS_TASK_STATUS_DETACHED;
uwRet = LOS_TaskCreate(&g_TestTskHandle,&task_init_param);//调用任务创建函数
if(uwRet !=LOS_OK)
{
return uwRet;
}
return uwRet;
}

VOID task2(void)
{
UINT32 uwRet = LOS_OK;
UINT32 count=0;
while(1)
{
count++;
//printf("This is task 2,count is %d\r\n",count);
LED2_TOGGLE;
LED3_TOGGLE;
uwRet = LOS_TaskDelay(2000);
if(uwRet !=LOS_OK)
return;
}
}

UINT32 creat_task2(void)
{
UINT32 uwRet = LOS_OK;
TSK_INIT_PARAM_S task_init_param;
task_init_param.usTaskPrio = 0;
task_init_param.pcName = "task2";
task_init_param.pfnTaskEntry = (TSK_ENTRY_FUNC)task2;
task_init_param.uwStackSize = LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE;
task_init_param.uwResved = LOS_TASK_STATUS_DETACHED;
uwRet = LOS_TaskCreate(&g_TestTskHandle,&task_init_param);
if(uwRet !=LOS_OK)
{
return uwRet;
}
```

```
return uwRet;
}

UINT32 osAppInit(void)
{
UINT32 uwRet = 0;
hardware_init();
LED1_ON;
SOFT_DELAY;
uwRet = creat_task1();
if(uwRet !=LOS_OK)
{
return uwRet;
}
uwRet = creat_task2();
if(uwRet !=LOS_OK)
{
return uwRet;
}
return LOS_OK;
}
/**********************************************END OF FILE**********************/
```

注意

osAppInit函数为用户添加的任务函数，所以需要在los_config.c文件的main函数中调用，并在los_config.h文件中使用extern UINT32 osAppInit(VOID);声明。Los_config.c文件中的main函数调用osAppInit()函数，修改如下：

```
int main(void)
{
UINT32 uwRet;
uwRet = osMain();
if (uwRet != LOS_OK) {
return LOS_NOK;
}
osAppInit();
LOS_Start();

for (;;);
/* Replace the dots (...) with your own code. */
}
```

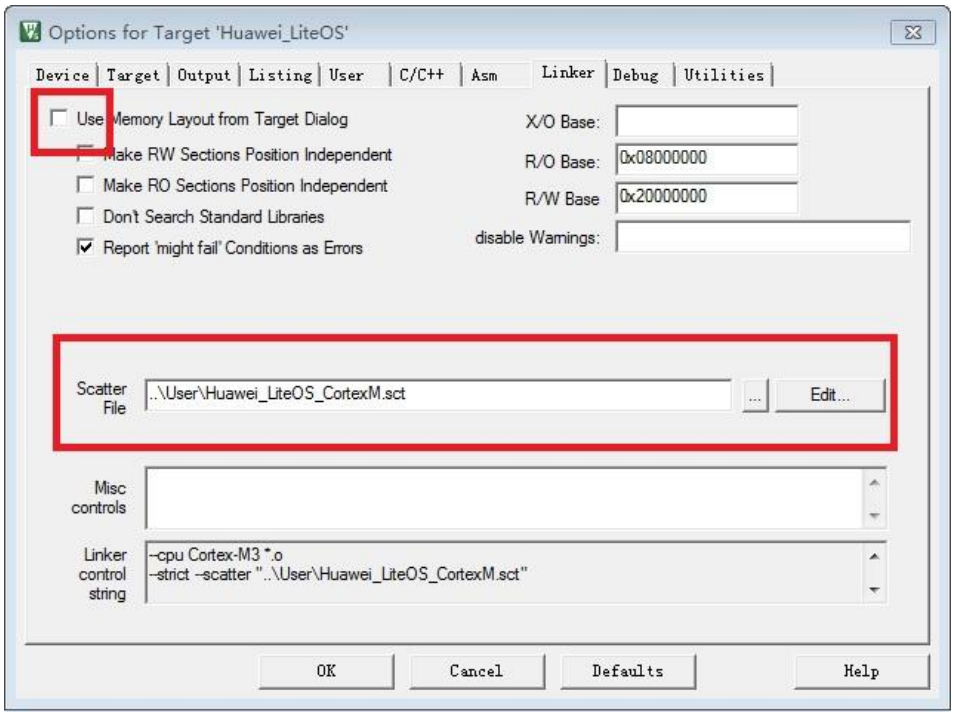
说明

以上步骤只是指导开发者完成基本任务的创建，如果开发者需要用到互斥锁和信号量以及其他操作系统组件，请参考详细的Huawei LiteOS文档，上文已给出下载地址。

----结束

5.5 移植的最后操作

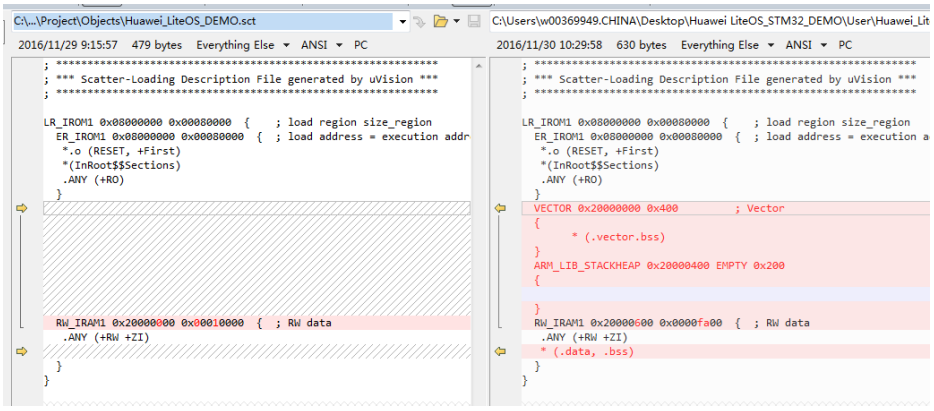
完成前三部分的操作后，我们的代码移植基本完成，最后一步我们还需要修改分散加载文件sct。首先，我们来看一下工程自动生成的sct文件，文件在Huawei LiteOS_STM32_DEMO\Project\Objects目录下（在下图的工程配置中，勾选“Use Memory Layout from Target Dialog”，则Keil会自动生成默认的sct文件）



一个普通的分散加载文件配置如下：

```
LR_IROM1 0x00000000 0x00040000 {                ; 定义一个加载时域，域基址：0x00000000，域大
                                                ; 小为 0x00040000，对应实际 Flash 的大小
  ER_IROM1 0x00000000 0x00040000 {              ; 定义一个运行时域，第一个运行时域必须和加载
                                                ; 时域起始地址相同，否则库不能加载到该时域的
                                                ; 错误，其域大小一般也和加载时域大小相同
    *.o (RESET, +First)                         ; 将 RESET 段最先加载到本域的起始地址外，即
                                                ; RESET 的起始地址为 0，RESET 存储的是向量表
    .ANY (+RO)                                  ; 加载所有匹配目标文件的只读属性数据，包含：
                                                ; Code、RW-Code、RO-Data。
  }

  RW_IRAM1 0x10000000 0x00008000 {              ; 定义一个运行时域，域基址：0x10000000，域大
                                                ; 小为 0x00008000，对应实际 RAM 大小
    * (+RW +ZI)                                 ; 加载所有区配目标文件的 RW-Data、ZI-Data
                                                ; 这里也可以用.ANY 替代*号
  }
}
```

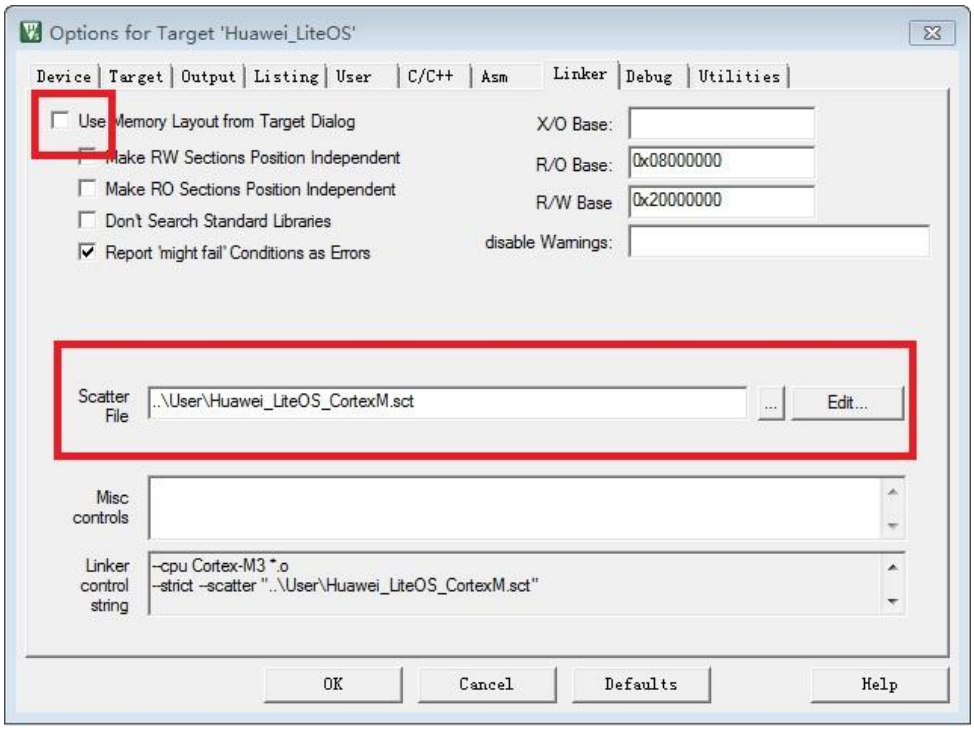
由于Huawei LiteOS中对数据和代码位置进行了控制，代码和数据会放在多个不同的内存区域，因此需要使用分散加载文件进行描述，要是系统准确运行起来，需要重新编写一个分散加载文件，配置MDK的链接器选择指定的分散加载文件，分散加载文件修改细节如下：



右边为修改后的文件，主要多加载了两个段，其中.vector.bss需要在los_builddef.h文件中进行配置，将该文件第90行的宏定义注释取消掉，修改后如下：

```
#define LITE_OS_SEC_VEC    __attribute__ ((section(".vector.bss")))
```

我们另存为Huawei_LiteOS_CortexM.sct文件放到Huawei LiteOS_STM32_DEMO \User目录下，然后在工程名上右键选择Options for Target…，弹出工程配置对话框，选择Linker选项，按下图进行配置：



完成后点击OK，回到主界面按F7编译工程，就可以下载到板子上验证功能了，如果移植正确，任务会根据优先级正确调度。

6 使用 DAP 仿真器下载程序

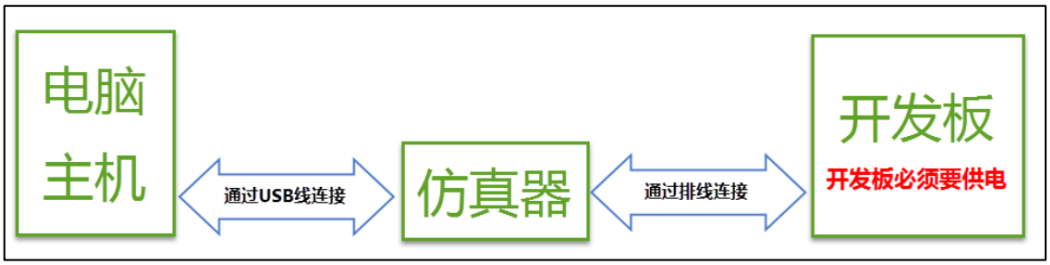
6.1 仿真器简介

仿真器为秉火STM32开发板配套的 Fire-Debugger，遵循 ARM 公司的 CMSIS-DAP 标准，支持所有基于 Cortex-M 内核的单片机，常见的 M3、 M4 和 M7 都可以完美支持。

Fire-Debugger 支持下载和在线仿真程序，支持 XP/WIN7/WIN8/WIN10 这四个操作系统，免驱，不需要安装驱动即可使用，支持 KEIL 和 IAR 直接下载，非常方便。

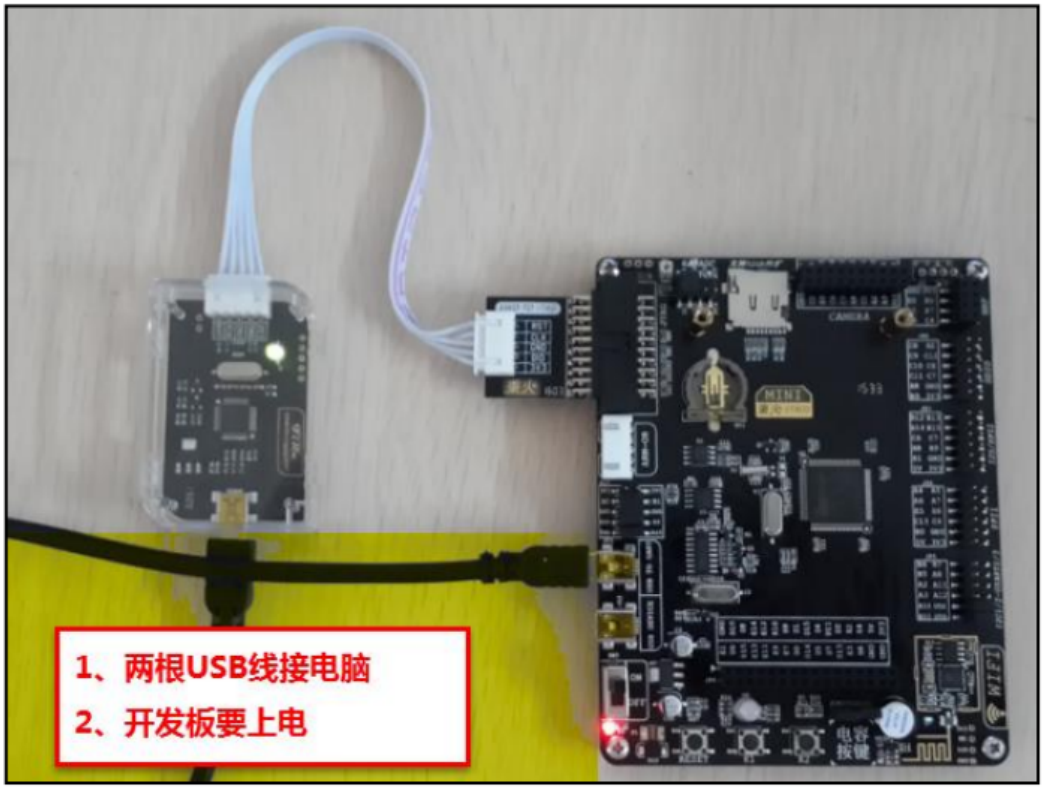
6.2 硬件连接

把仿真器用 USB 线连接电脑，如果仿真器的灯亮则表示正常，可以使用。然后把仿真器的另外一端连接到开发板， 给开发板上电，然后就可以通过软件 KEIL给开发板下载程序。



仿真器与电脑和开发板连接方式

图 6-1 仿真器与开发板连接示意图

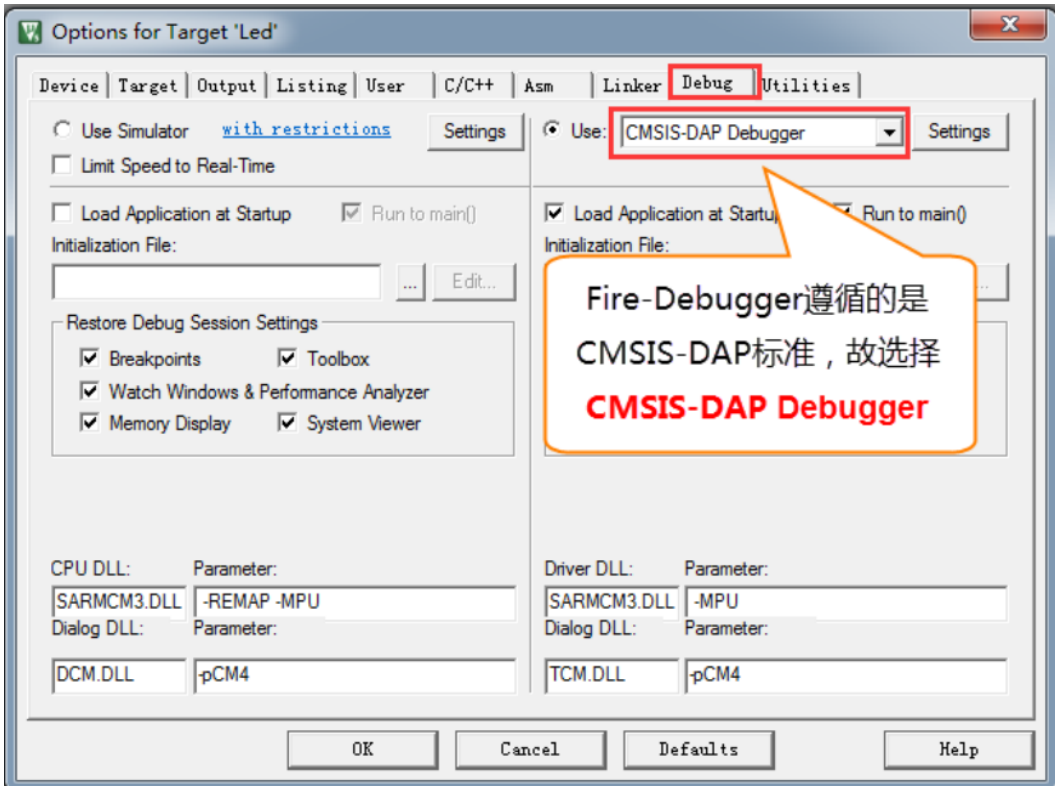


6.3 MDK 中仿真器的配置

在仿真器连接好电脑和开发板且开发板供电正常的情况下， 打开编译软件 KEIL， 在工程配置选项卡里面选择仿真器的型号，具体过程看图示：

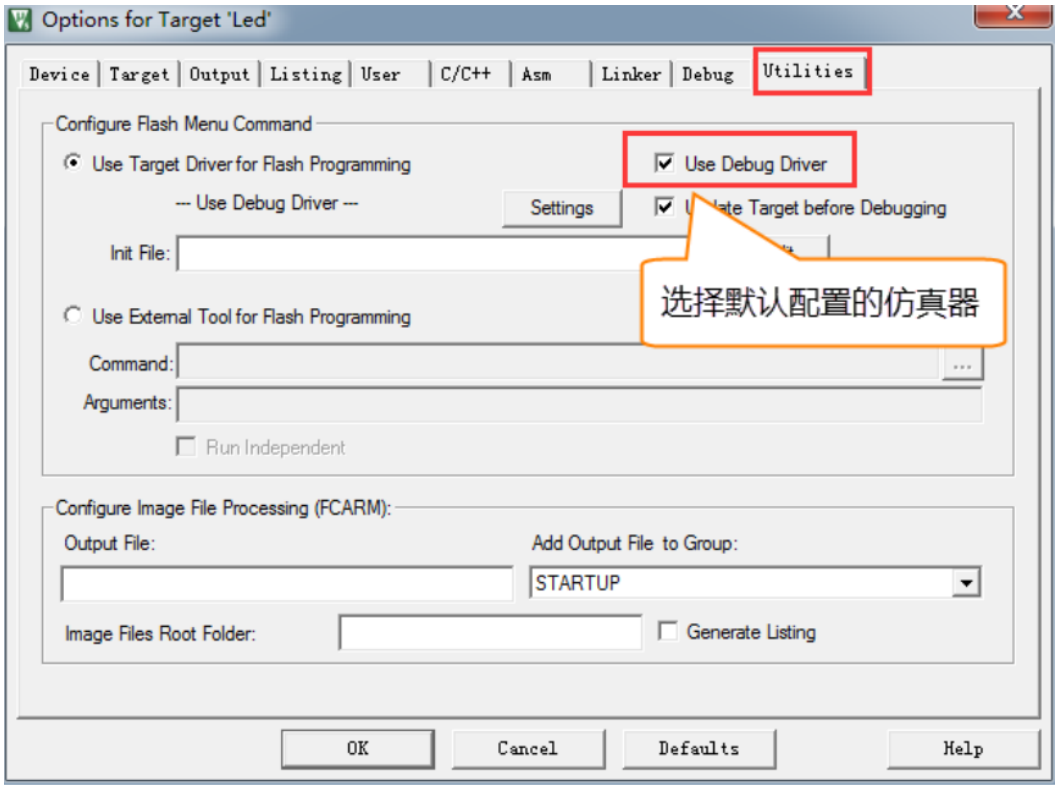
步骤1 Debug选项卡配置

Debug 选择 CMSIS-DAP Debugger

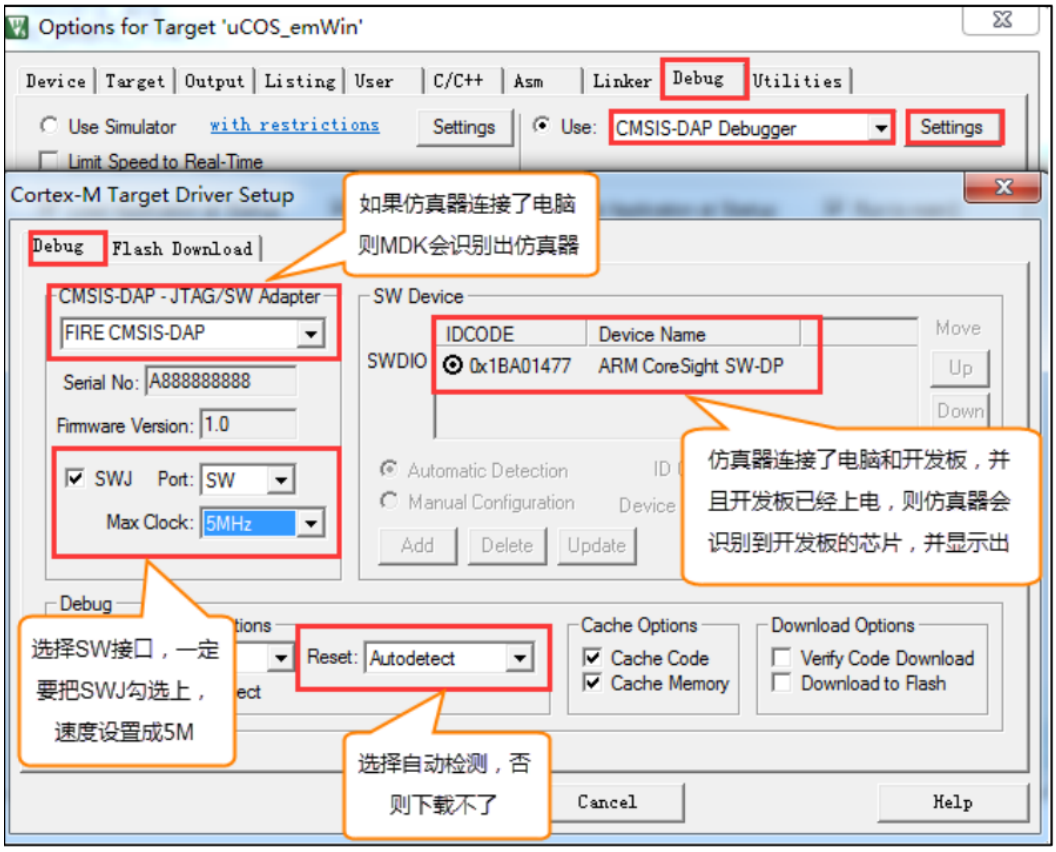


步骤2 Utilities 选项配置

Utilities 选择 Use Debug Driver



步骤3 Debug Settings 选项配置

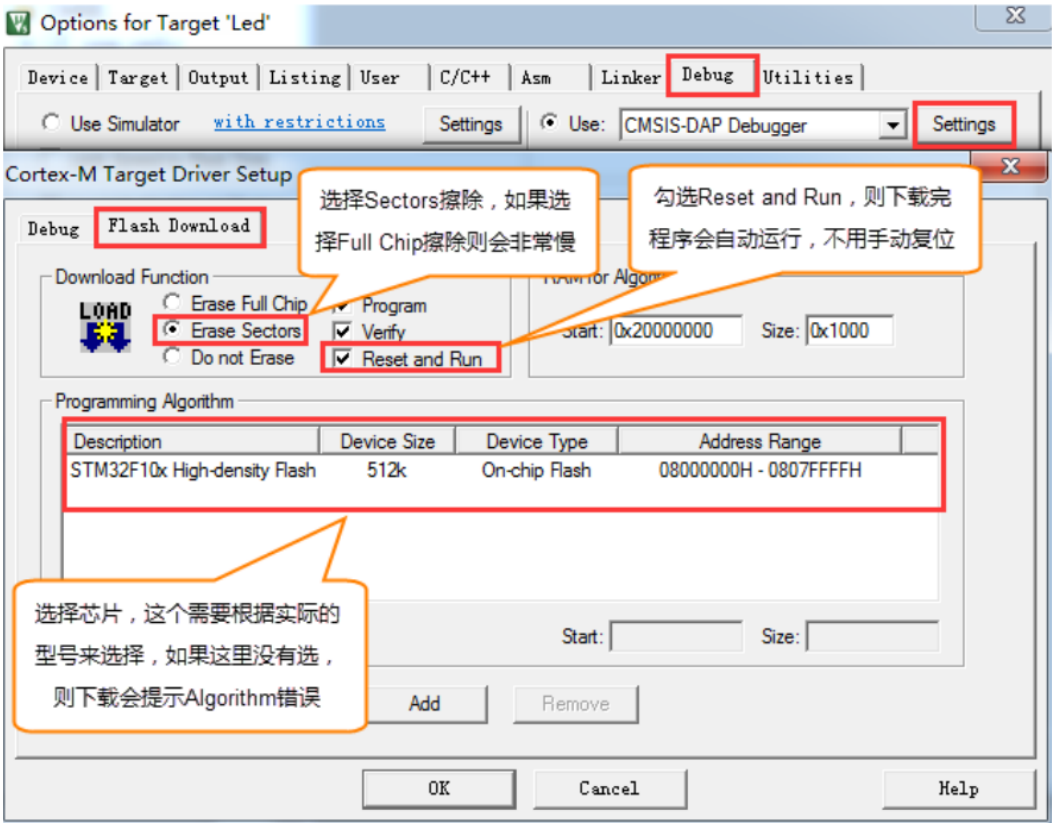


----结束

6.4 选择目标板

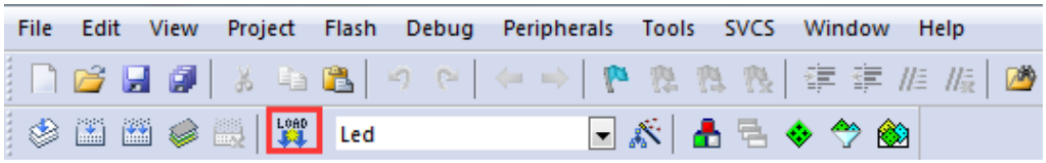
选择目标板，具体选择多大的 FLASH 要根据板子上的芯片型号决定。秉火 STM32 开发板的配置是： F1 选 512K， F4 选 1M。

这里面有个小技巧就是把 Reset and Run 也勾选上，这样程序下载完之后就会自动运行，否则需要手动复位。擦除的 FLASH 大小选择 Sectors即可，不要选择 Full Chip，不然下载会比较慢。



6.5 下载程序

如果前面步骤都成功了，接下来就可以把编译好的程序下载到开发板上运行。下载程序不需要其他额外的软件，直接点击 KEIL 中的 LOAD 按钮即可。



程序下载后， Build Output 选项卡如果打印出 Application running…则表示程序下载成功。如果没有出现实验现象，按复位键试试。