目录

目的

本文档介绍基于Huawei LiteOS如何移植到第三方开发板，并成功运行基础示例。

读者对象

本文档主要适用于Huawei LiteOS Kernel的开发者。 本文档主要适用于以下对象：

- 物联网端软件开发工程师
- 物联网架构设计师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

用于警示紧急的危险情形，若不避免，将会导致人员死亡或严重的人身伤害

用于警示潜在的危险情形，若不避免，可能会导致人员死亡或严重的人身伤害

用于警示潜在的危险情形，若不避免，可能会导致中度或轻微的人身伤害

用于传递设备或环境安全警示信息，若不避免，可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果“注意”不涉及人身伤害

| 说明 | “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息 |

修订记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新 内容。

日期	修订版本	描述
2017年1月17日	1.0	完成初稿
2017年3月17日	1.1	重构手册，并根据更新的代码结构刷新内容

修订时间： 2017-03-22 09:45:32

关于本文档的开源协议说明

您可以自由地：

分享

- 在任何媒介以任何形式复制、发行本文档

演绎

- 修改、转换或以本文档为基础进行创作。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，或以本文档为基础进行创作，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.

概述

目前在github上已开源的Huawei LiteOS内核源码已适配好STM32F412、STM32F429、STM32L476、GD32F450、GD32F190芯片，本手册将介绍LiteOS如何从零创建可以运行的工程的移植过程以及如何修改LiteOS的各种配置等内容。本文档使用的demo板是STM32F4291-DISCO单板。

修订时间： 2017-03-21 11:58:23

环境准备

基于Huawei LiteOS Kernel开发前，我们首先需要准备好单板运行的环境，包括软件环 境和硬件环境。

硬件环境：

所需硬件	描述
STM32F4291-DISCO单板	STM32开发板(芯片型号STM32F429ZIT6)
PC机	用于编译、加载并调试镜像
电源（5v）	开发板供电(使用Mini USB连接线)

软件环境：

软件	描述
Window 7 操作系统	安装Keil和st-link的操作系统
Keil(5.18以上版本)	用于编译、链接、调试程序代码 uVision V5.18.0.0 MDK-Lite
st-link_v2_usbdriver	开发板与pc连接的驱动程序，用户加载及调试程序代码

说 明

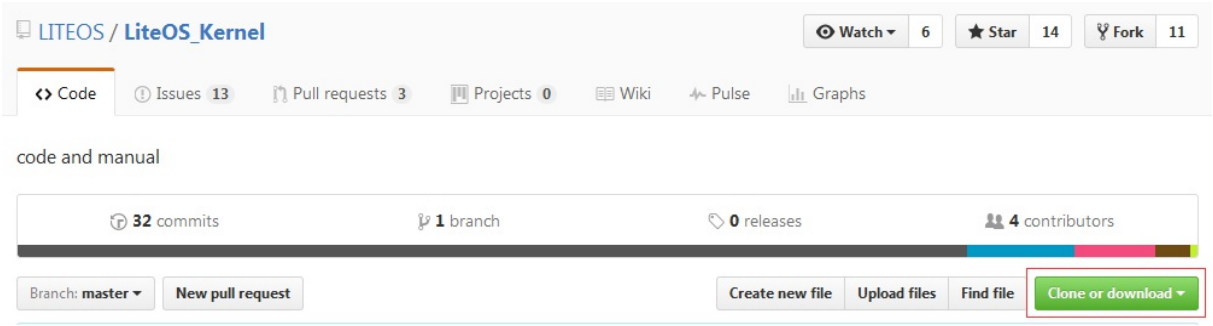
Keil工具需要开发者自行购买，ST-Link的驱动程序可以从st link的相关网站获取，采用J-Link还 是ST-Link需要根据具体使用的开发板来确定。这里以STM32F429为例，使用ST-Link。

修订时间： 2017-03-21 11:58:23

获取Huawei LiteOS 源码

首先我们需要通过网络下载获取Huawei LiteOS开发包。目前Huawei LiteOS的代码已经 开源，可以直接从网络上获取，步骤如下：

1. 仓库地址是https://github.com/LITEOS/LiteOS_Kernel.git



2. 点击”clone or download”按钮,下载源代码

   目录结构如下：Huawei LiteOS的源代码目录的各子目录包含的内容如下：

关于代码树中各个目录内容简介如下：

一级目录	二级目录	说明
doc		此目录存放的是LiteOS的使用文档和API说明文档
example	api	此目录存放的是内核功能测试用的相关用例的代码
	include	aip功能头文件存放目录
kernel	base	此目录存放的是与平台无关的内核代码，包含核心提供给外部调用的接口的头文件以及内核中进程调度、进程通信、内存管理等等功能的核心代码。用户一般不需要修改此目录下的相关内容。
	cmsis	LiteOS提供的cmsis接口
	config	此目录下是内核资源配置相关的代码，在头文件中配置了LiteOS所提供的各种资源所占用的内存池的总大小以及各种资源的数量，例如task的最大个数、信号量的最大个数等等
	cpu	此目录以及以下目录存放的是与体系架构紧密相关的适配LiteOS的代码。比如目前我们适配了arm/cortex-m4及arm/cortex-m3系列对

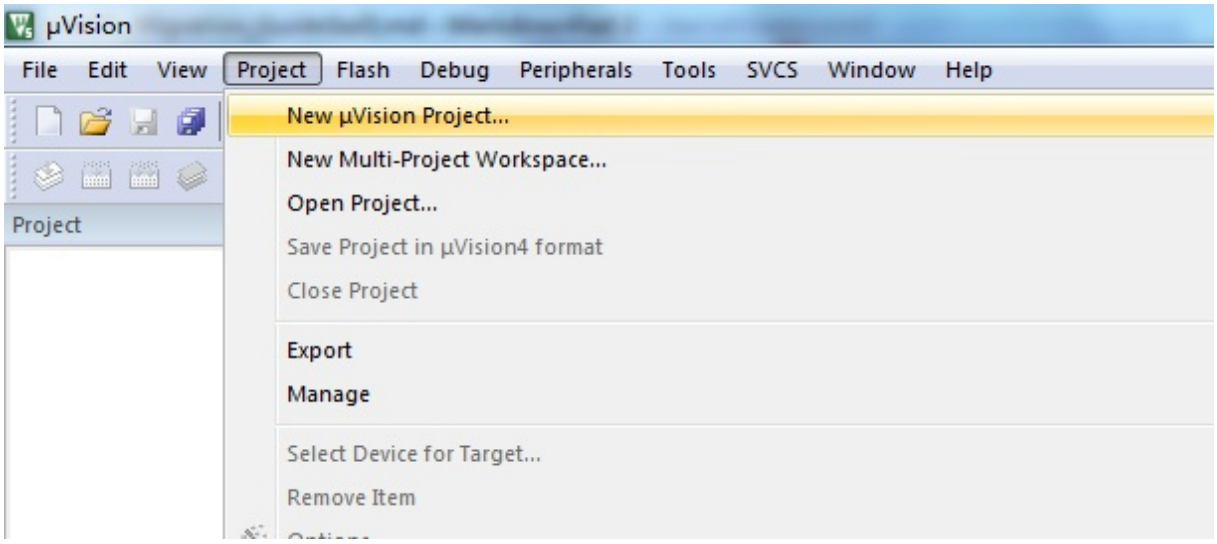
		应的初始化内容。
	include	内核的相关头文件存放目录
	link	与IDE相关的编译链接相关宏定义
platform	GD32F190R-EVAL	GD190开发板systick以及led、uart、key驱动bsp适配代码
	GD32F450i-EVAL	GD450开发板systick以及led、uart、key驱动bsp适配代码
	STM32F412ZG-NUCLEO	STM32F412开发板systick以及led、uart、key驱动bsp适配代码
	STM32F429I_DISCO	STM32F429开发板systick以及led、uart、key驱动bsp适配代码
	STM32L476RG_NUCLEO	STM32L476开发板systick以及led、uart、key驱动bsp适配代码
projects	STM32F412ZG-NUCLEO-KEIL	stm32f412开发板的keil工程目录
	STM32F429I_DISCO_IAR	stm32f429开发板的iar工程目录
	STM32F429I_DISCO_KEIL	stm32f429开发板的keil工程目录
	STM32L476R-Nucleo	stm32f476开发板的keil工程目录
	GD32F190R-EVAL-KEIL	gd32f190开发板的keil工程目录
	GD32F450i-EVAL-KEIL	gd32f450开发板的keil工程目录
user		此目录存放用户测试代码，LiteOS的初始化和使用示例在main.c中

获取Huawei LiteOS源代码之后，我们就可以创建project然后编译调试我们的程序了， 详细可以参考后续的各个章节。详细的编程应用编程API请参考《HuaweiLiteOSKernelDevGuide》

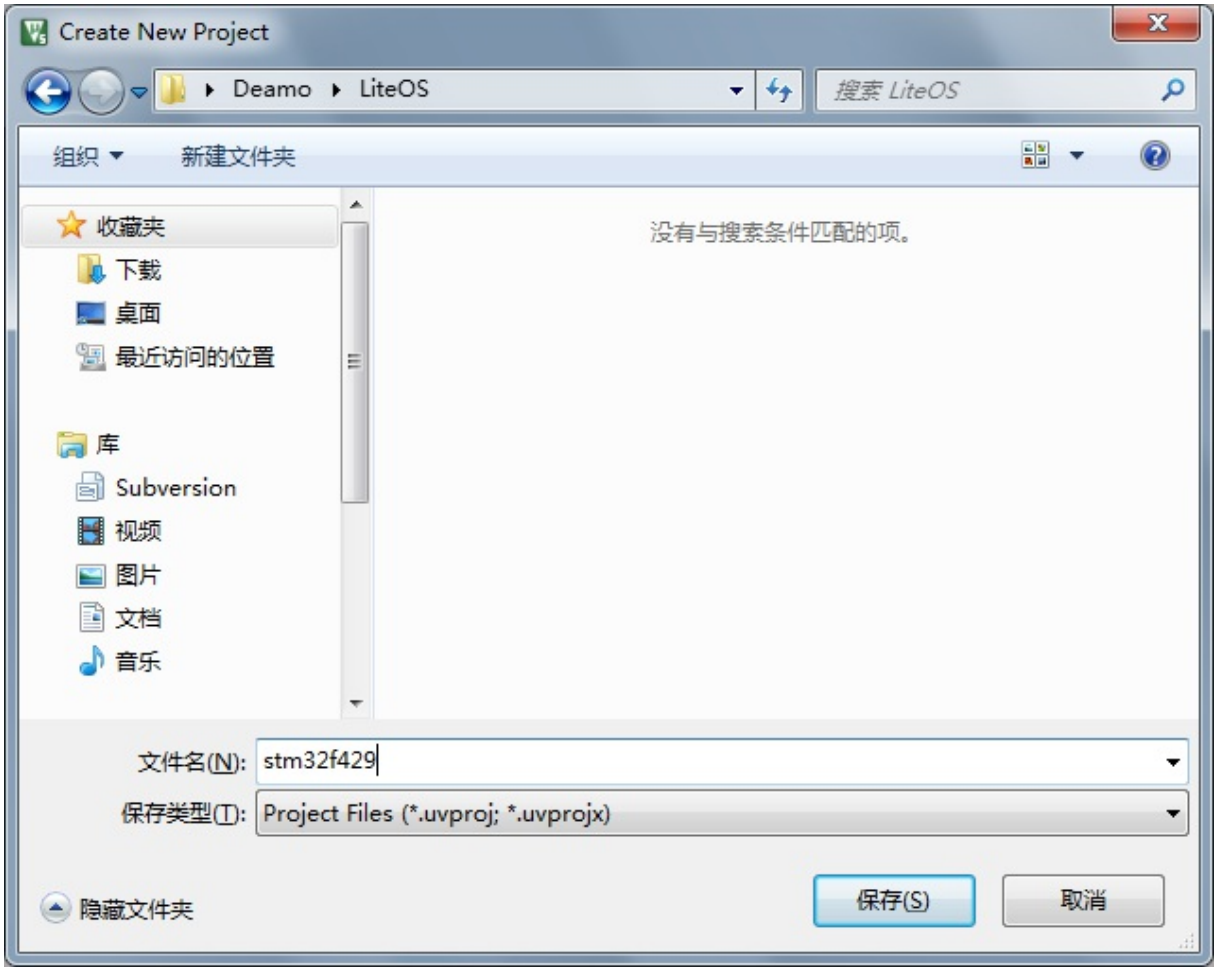
创建Huawei LiteOS 工程

在获取完成Huawei LiteOS的源代码和安装好Keil等相关的开发工具后，我们需要用Keil 集成开发环境创建编译Huawei LiteOS的工程，步骤如下：

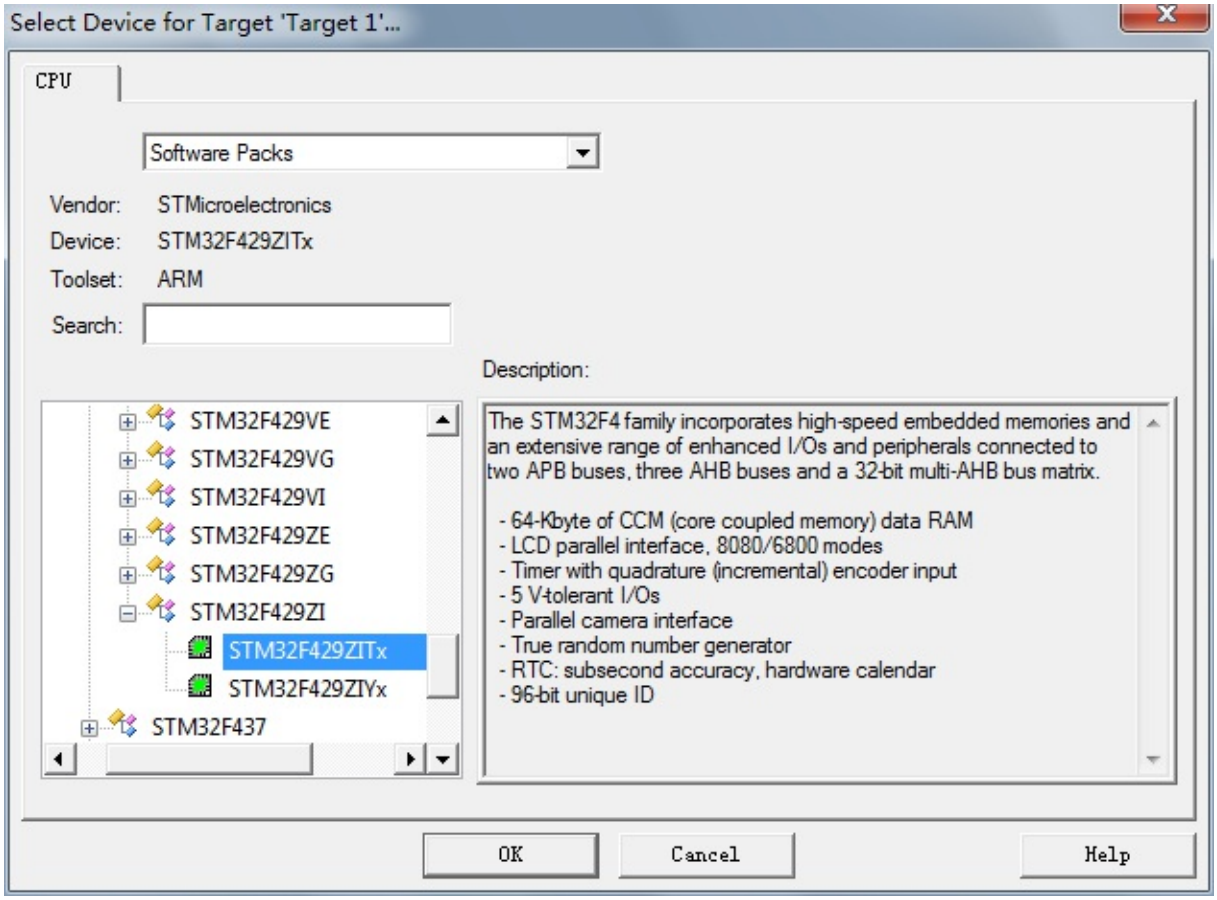
1. 打开Keil uVision5， 然后点击project->New uVision Project...创建一个新的工程



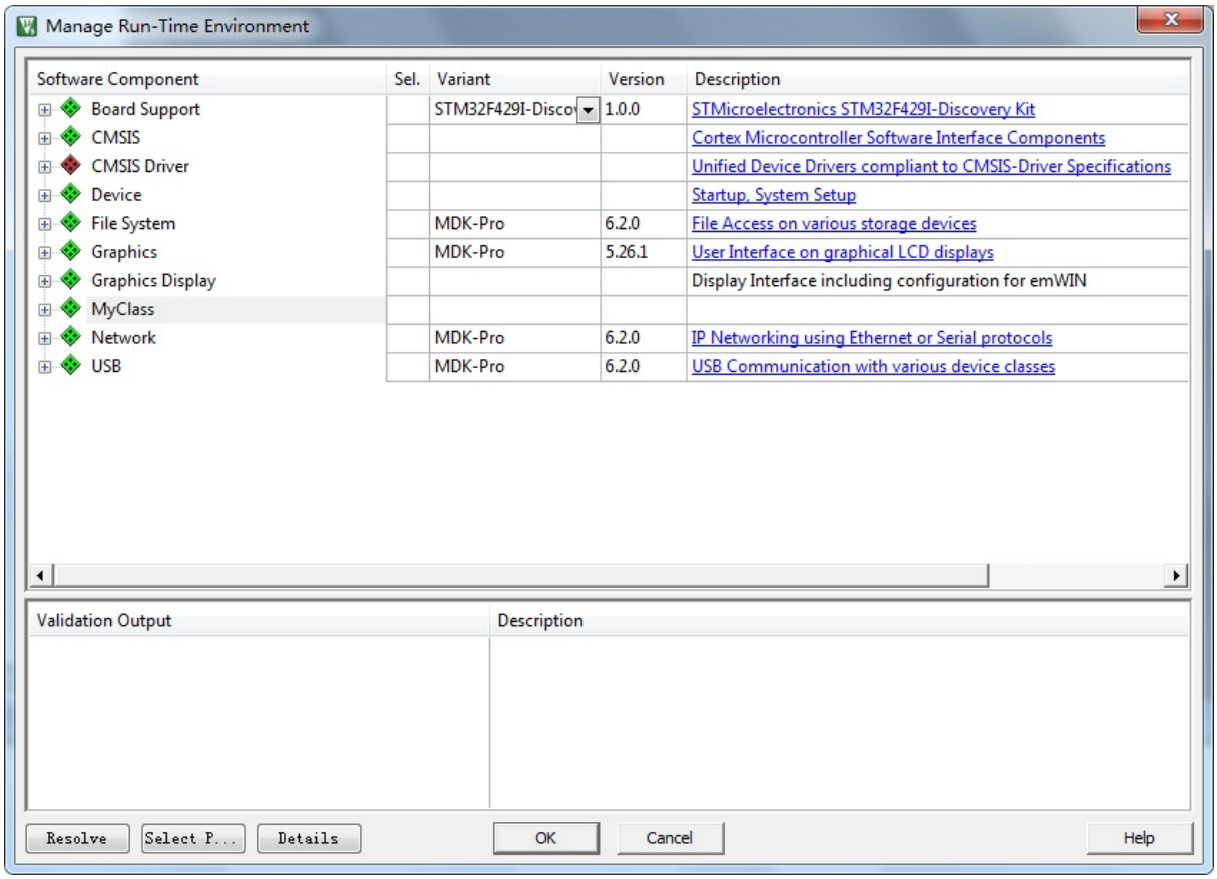
2. 保存工程名，比如stm32f429



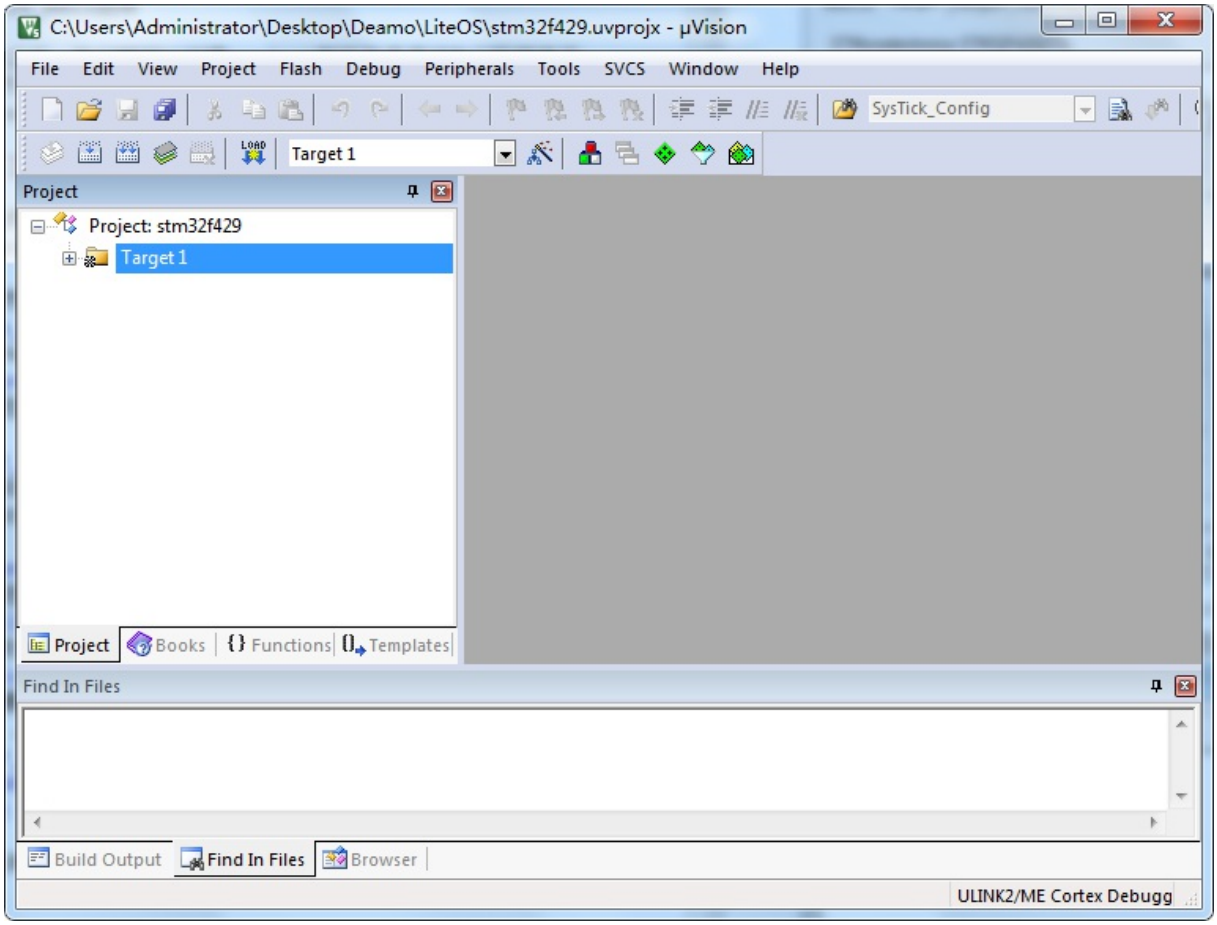
3. 保存后会立即弹出芯片型号选择的窗口，根据实际的开发板的芯片进行选择，比如stm32f429zi是目前demo使用的芯片。



4. 然后选择要包含的开发基础库，比如CMSIS、DEVICE两个选项可以选择平台提供的支持包和启动汇编文件，不过目前LiteOS有自己的启动文件，并且不需要额外的驱动，所以可以直接点击OK跳过。



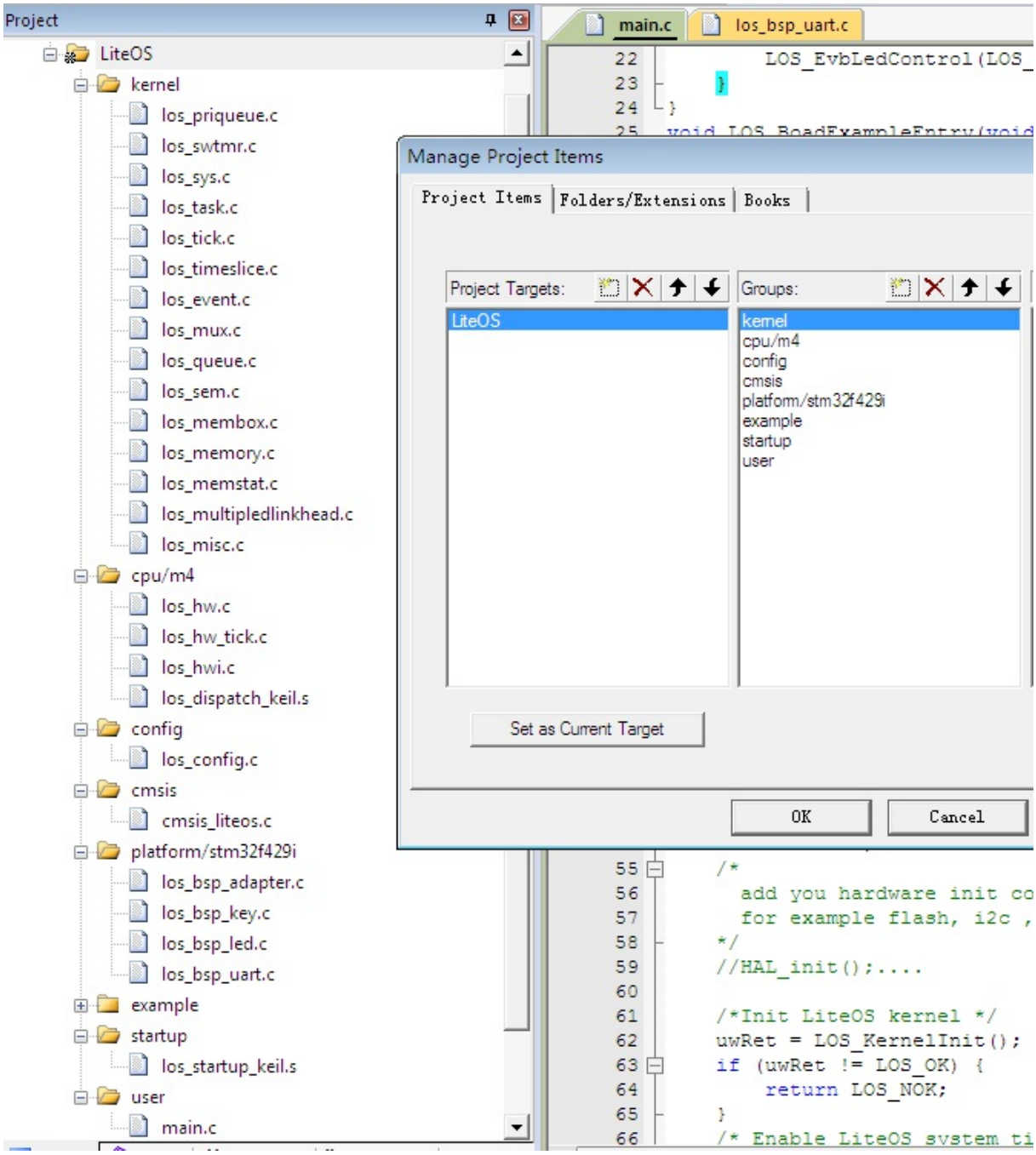
5. 至此，我们的工程已经创建完成，如下图所示：



完成上面的芯片和支持包选择之后，可以将源代码添加到工程中。

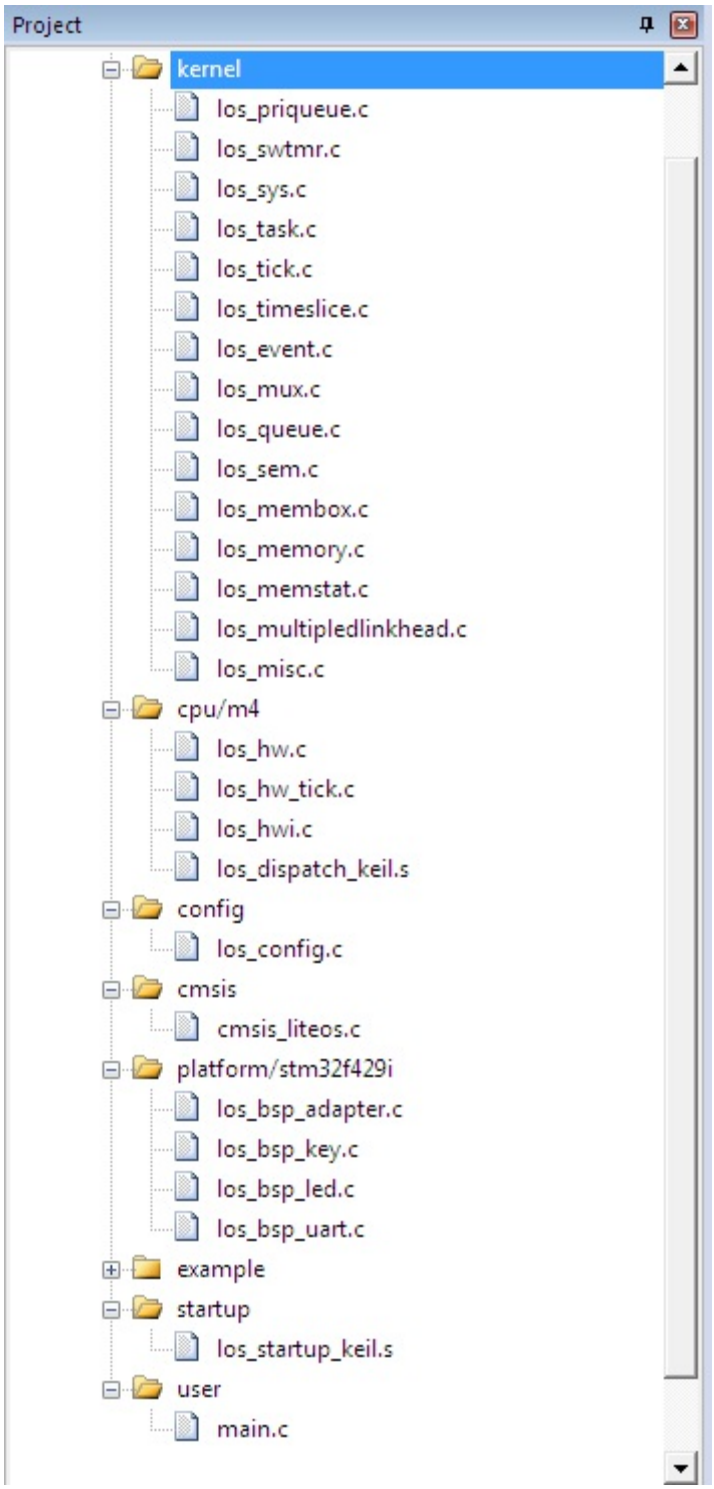
添加**kernel**代码到工程

1. 创建LiteOS的相关目录层级



2. 创建完成目录树之后我们逐个添加源代码到目录树中，最终添加完成的内容如下：

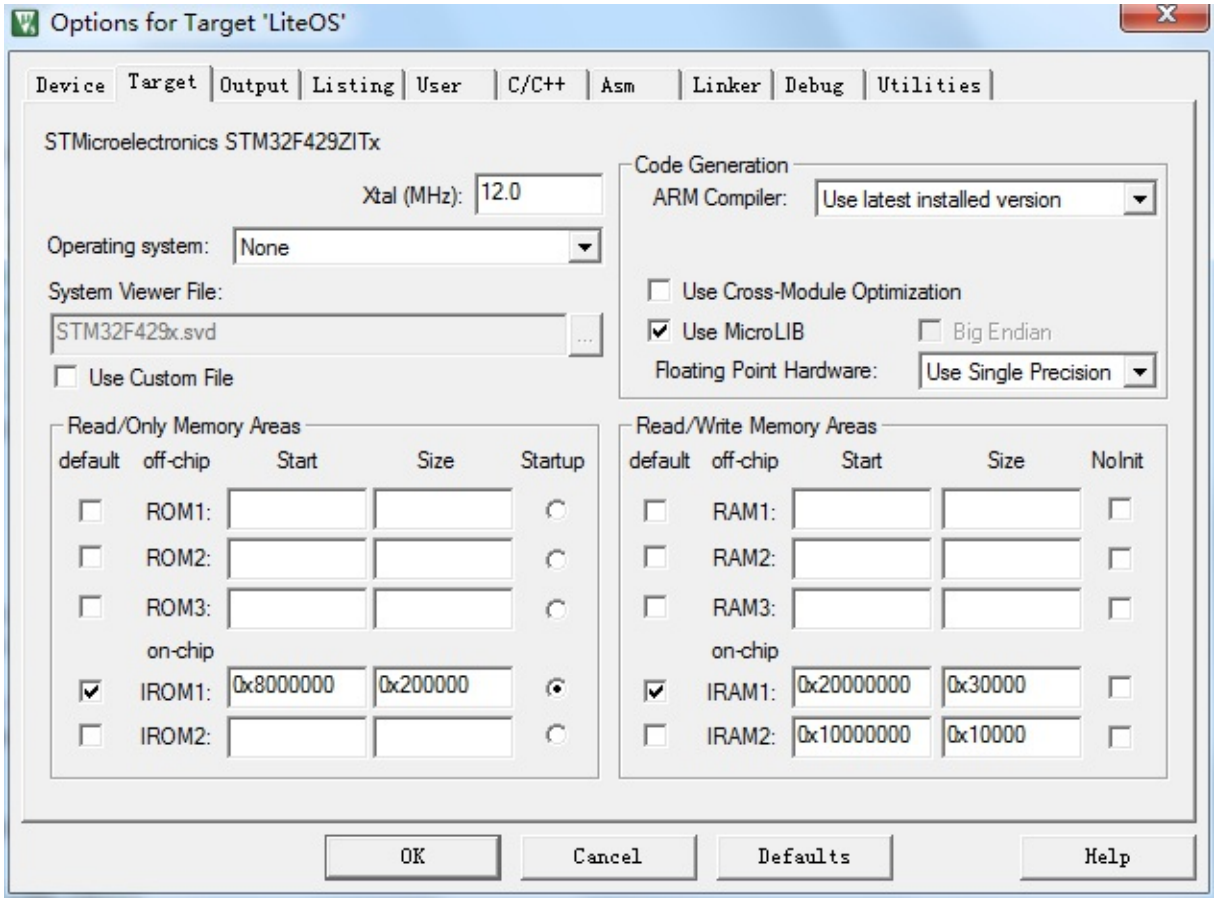
   - 将kernel/base目录下的所有C代码添加到工程中的kernel下
   - 将kernel/cmsis目录下的所有C代码添加到工程中的cmsis下。
   - 将platform\STM32F429I_DISCO目录下的所有C代码添加到工程中的platform/stm32f429i下
   - 将kernel\cpu\arm\cortex-m4目录下的所有C代码以及汇编代码添加到工程中的cpu/m4下
   - 将kernel\config目录下的所有C代码添加到工程中的config下
   - 将user目录下的所有C代码添加到工程中的user下
   - 将platform\STM32F429I_DISCO目录下keil版本的startup汇编代码添加到工程中的startup下



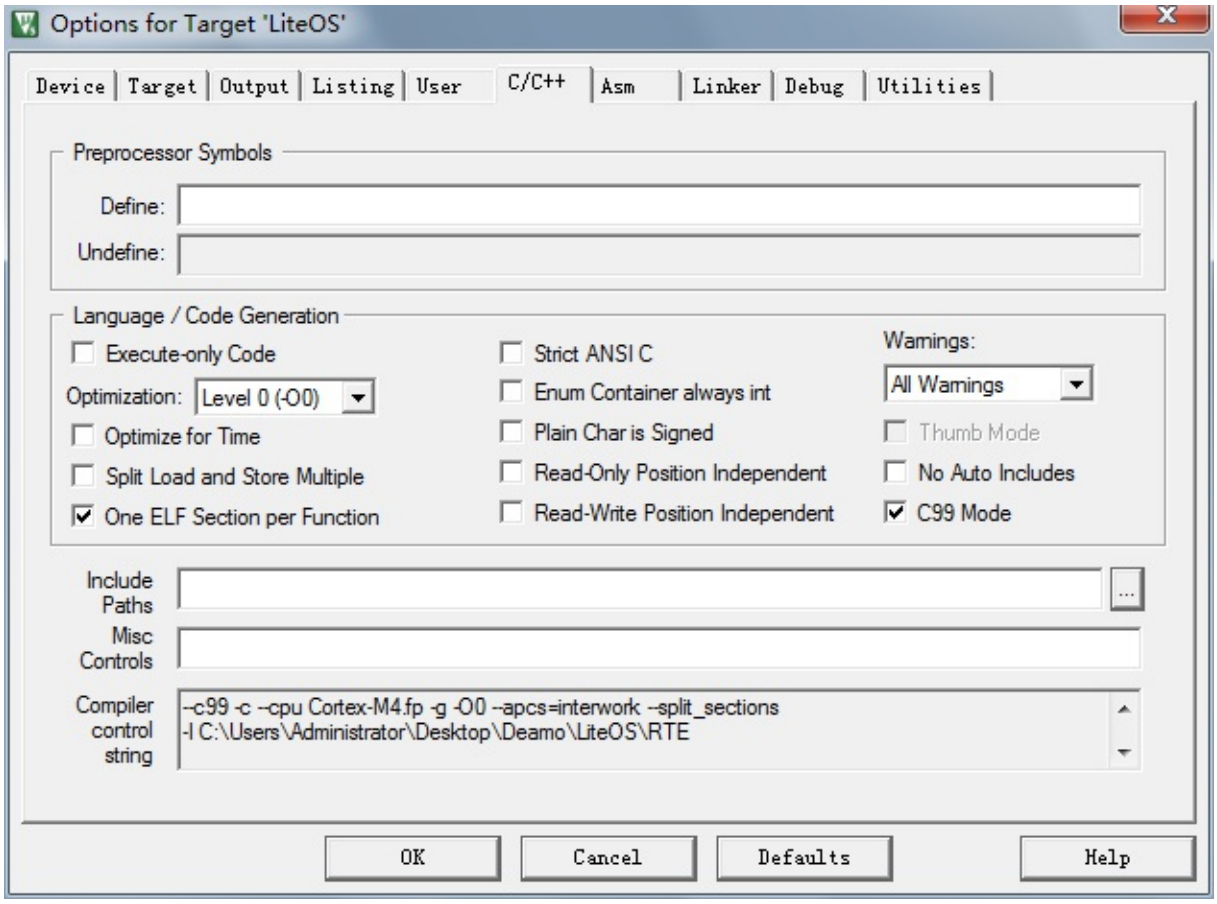
说明：已经创建好的工程中还增加了example的添加，example下的内容是用来测试kernel api接口的。

配置工程属性

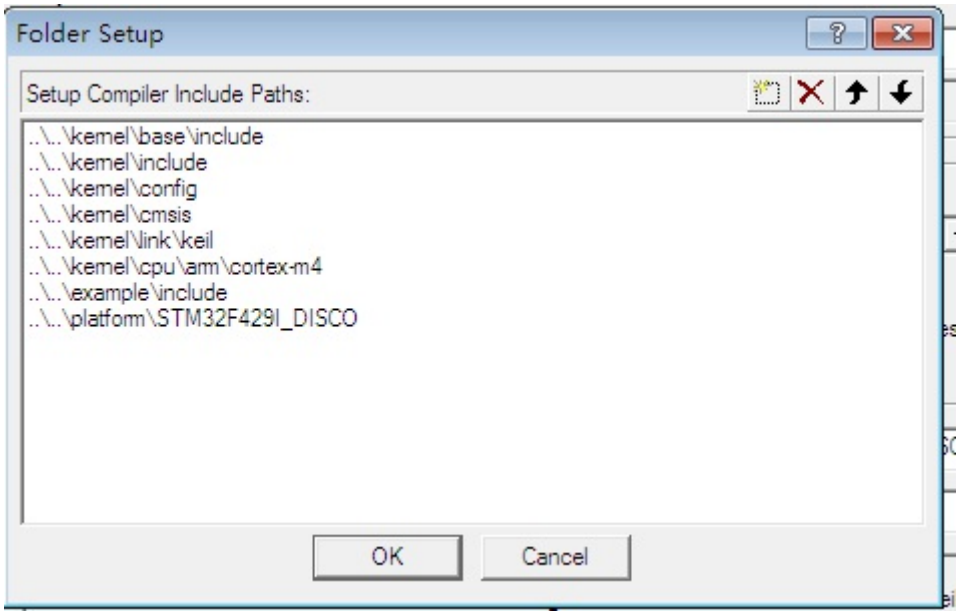
1. 配置target，如果需要调试log输出（printf）到IDE，可以选择Use MicroLib。



2. 编译C/C++设置中勾选C99选项

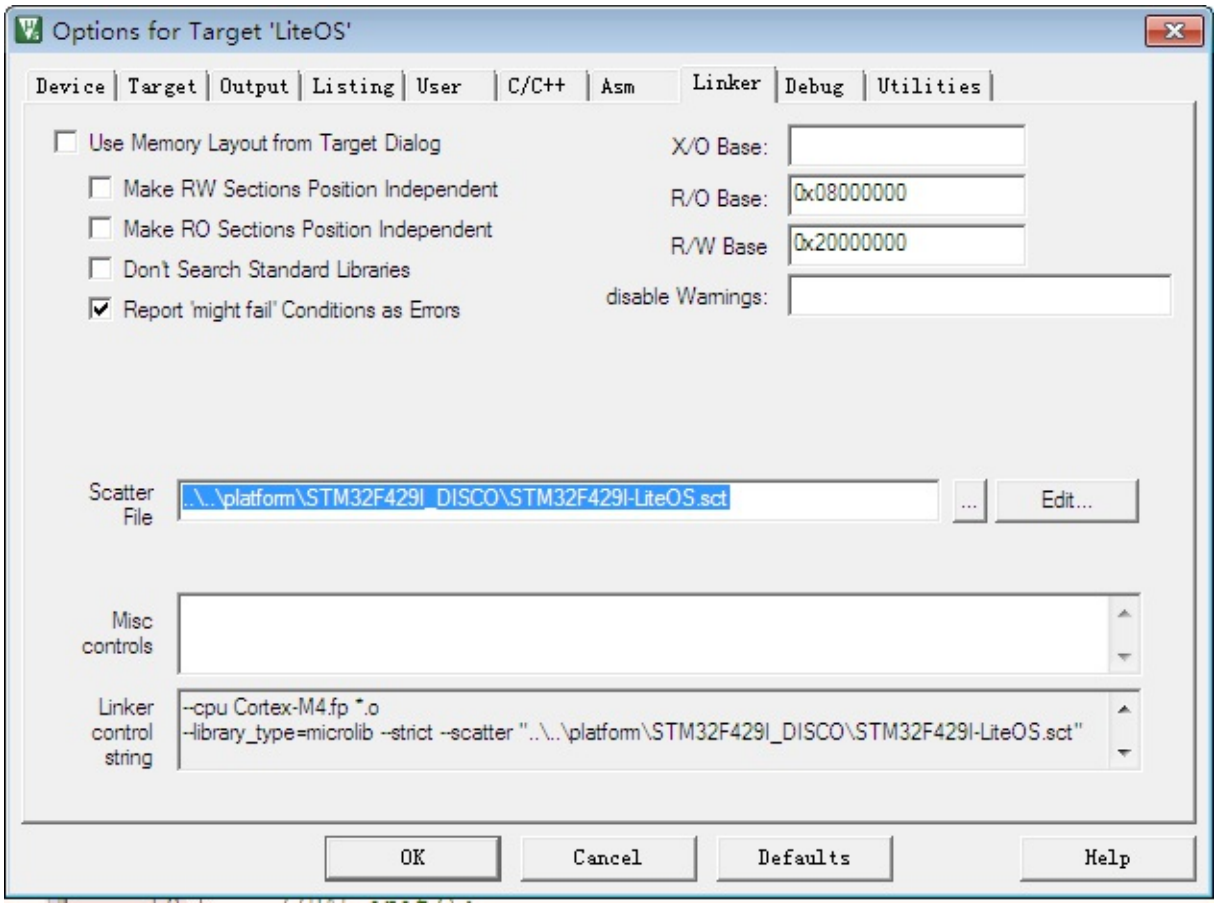


3. 配置头文件搜索路径，需要....\kernel\base\include;....\kernel\include;....\kernel\config;....\kernel\cmsis;....\kernel\link\keil;....\kernel\cpu\arm\cortex-m4;....\example\include;....\platform\STM32F429I_DISCO等等，详细参考图片所示内容。



说明： platform\STM32F429I_DISCO以及kernel\cpu\arm\cortex-m4则需要根据实际使用的cpu和开发板目录来添加。

4. 配置分散加载文件



例子：stm32f429的配置文件内容如下：

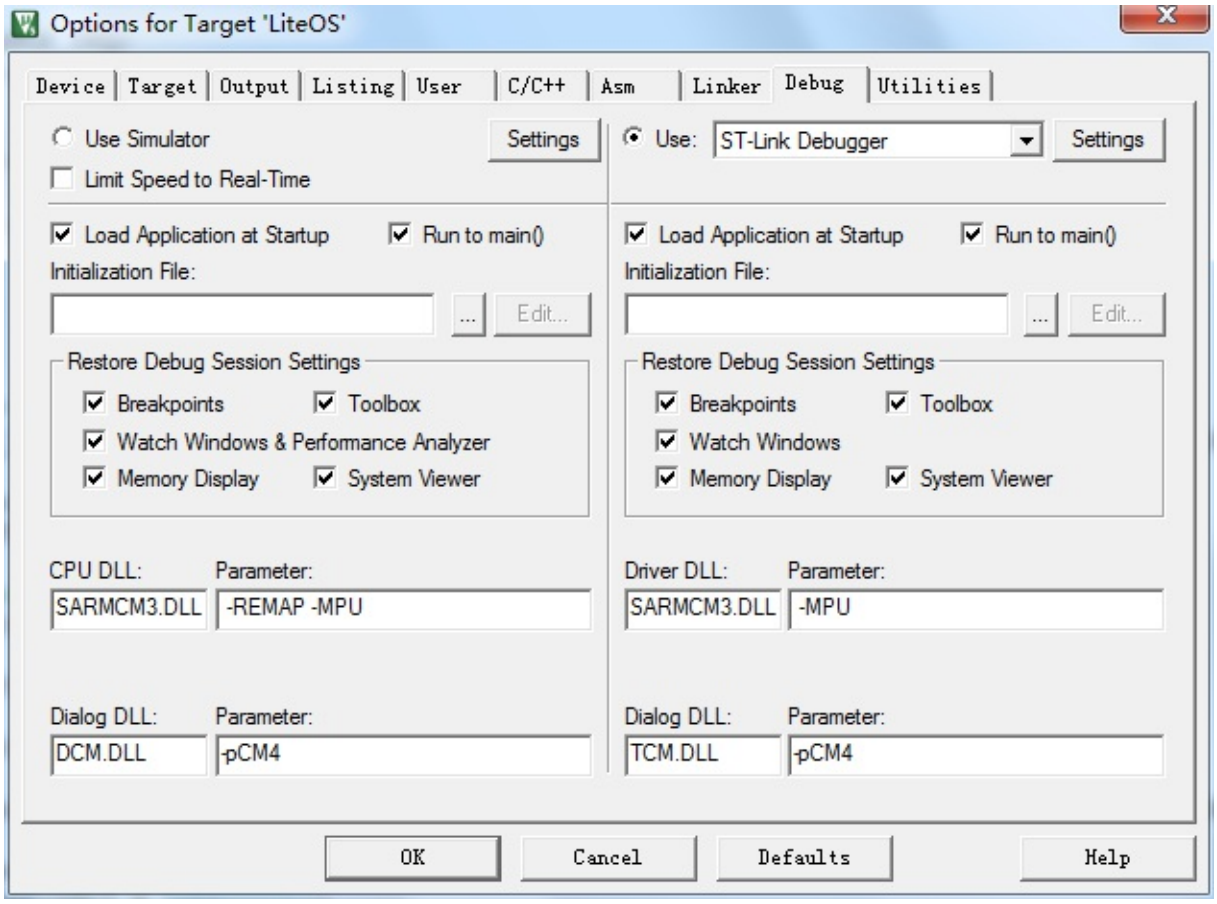
server.c | new 1 | STM32F429I-LiteOS.sct

```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

LR_IROM1 0x08000000 0x00200000  {    ; load region size_region
  ER_IROM1 0x08000000 0x00200000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  VECTOR 0x20000000 0x400 { ;vector
    * (.vector.bss)
  }

  RW_IRAM1 0x20000400 0x0002FC00  {  ; RW data
   .ANY (+RW +ZI)
   * (.data, .bss)
  }
}
```

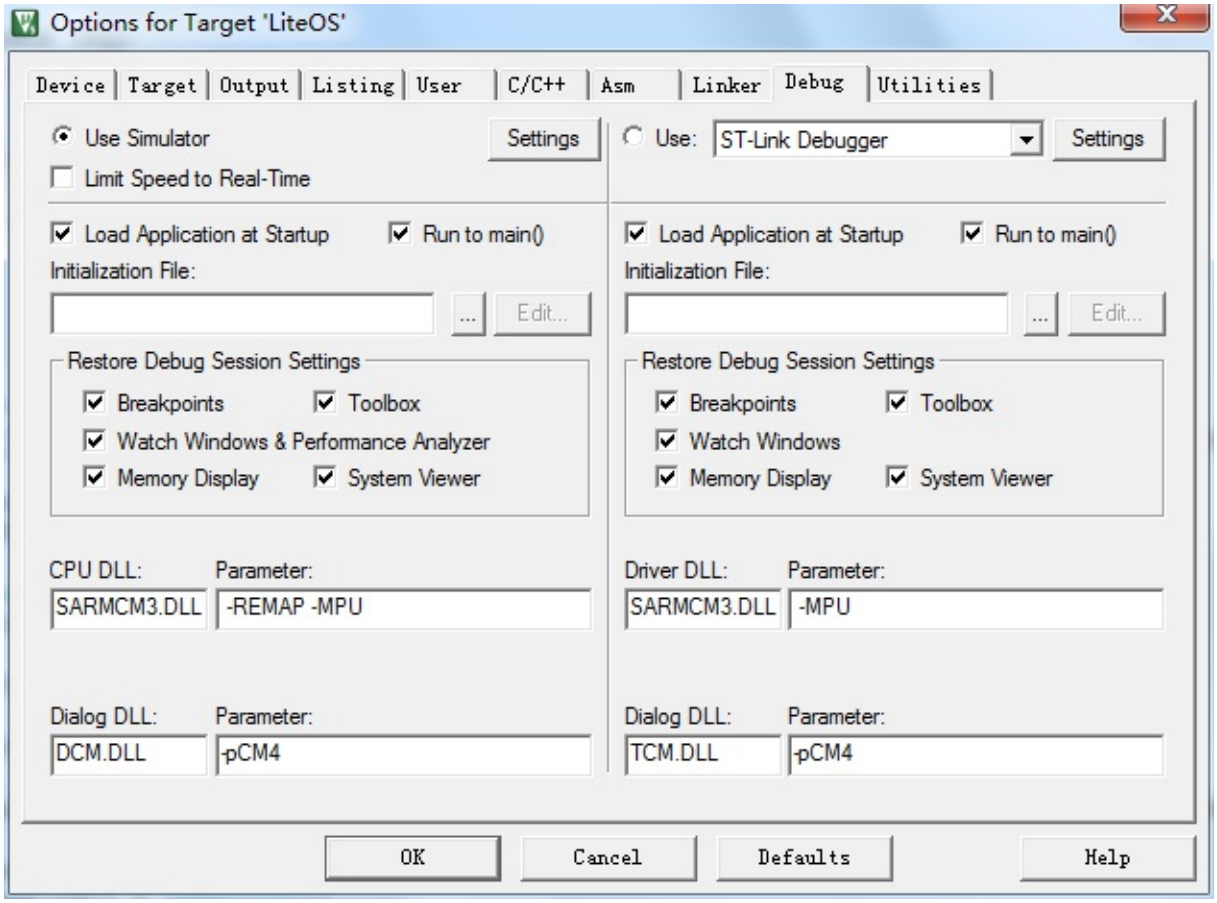
说明：

- 分散加载文件在每个开发板目录下，比如stm32f429的是\platform\STM32F429I_DISCO\STM32F429I-LiteOS.sct
- 分散配置文件中增加的是vector（中断向量表）的内容，LiteOS的中断向量表在stm32f429ZI这个芯片中定义的是0x400大小。如果不了解分散加载文件可以参考IDE的help中有关sct文件的说明。或者baidu、google分散加载文件相关内容。

5. 配置debug使用的驱动，选择ST-Link。



1. 对于需要使用printf输出调试log的场景，可以使用软件仿真。



测试代码使用

测试代码入口是los_demo_entry.c中的LOS_Demo_Entry()这个接口，使用方法los_config.c的main中调用

示例如下：

```
extern void LOS_Demo_Entry(void);
int main(void)
{
    UINT32 uwRet;
    /*
            add you hardware init code here
            for example flash, i2c , system clock ....
    */
    //HAL_init();....

    /*Init LiteOS kernel */
    uwRet = LOS_KernelInit();
    if (uwRet != LOS_OK) {
        return LOS_NOK;
    }
        /* Enable LiteOS system tick interrupt */
    LOS_EnableTick();


    /*
        Notice: add your code here
        here you can create task for your function
        do some hw init that need after systemtick init
    */
    //LOS_EvbSetup();
    //LOS_BoadExampleEntry();

    LOS_Demo_Entry();
    /* Kernel start to run */
    LOS_Start();
    for (;;);
    /* Replace the dots (...) with your own code.  */
}
```

如何选择测试的功能：

- 在example\include\los_demo_entry.h 打开要测试的功能的宏开关LOS_KERNEL_TEST_xxx，比如测试task调度打开 LOS_KERNEL_TEST_TASK 即可（//#define LOS_KERNEL_TEST_TASK 修改为 #define LOS_KERNEL_TEST_TASK）
- 如果需要printf，则将los_demo_debug.h中的LOS_KERNEL_DEBUG_OUTLOS_KERNEL_TEST_KEIL_SWSIMU打开。如果是在IAR工程中则不需要打开LOS_KERNEL_TEST_KEIL_SWSIMU
- 中断测试无法在软件仿真的情况下测试, 中断测试请自行添加中断的初始化相关内容。

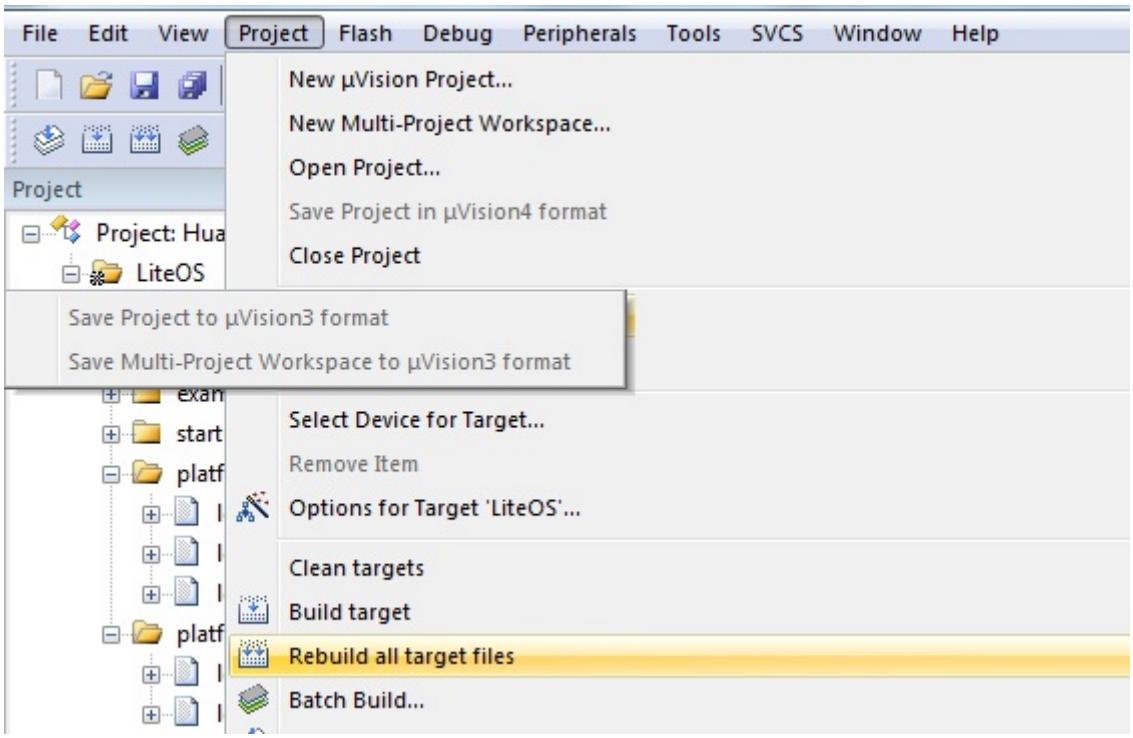
在**keil**中需要使用**printf**打印可以有几种方法

- 将printf重定位到uart输出，这个需要uart驱动支持，如果只有liteOS而没有相关驱动加入工程则不建议使用该方法。
- 使用软件仿真的方式在keil IDE的debug printf view中查看。

修订时间： 2017-03-21 11:58:23

编译调试

打开工程后，菜单栏Project→Clean Targets、Build target、Rebuild All target files，可编译文件。这里点 击Rebuild All target file，编译全部文件



关于中断向量位置选择

在los_bsp_adapter.c中，g_use_ram_vect变量控制了LiteOS中是否使用vector向量表（中断向量表）重定向功能。如果g_use_ram_vect设置为 1 ，则需要在配置分散加载文件，如果配置为0，则不配置分散加载文件（即在上面的配置步骤中可以不进行分散加载文件配置），系统启动后默认中断向量表在Rom的0x00000000地址。

关于工程创建

目前在LiteOS的源代码中已经存在了一些已经创建好的工程，用户可以直接使用，它们都在projects目录下。建议用户使用projects下已经建立好的工程作为LiteOS运行是否正常的参考工程使用。

修订时间： 2017-03-22 09:45:35

如何使用**LiteOS** 开发

LiteOS中提供的功能包括如下内容： 任务创建与删除、任务同步（信号量、互斥锁）、动态中断注册机制 等等内容，更详细的内容可以参考“HuaweiLiteOSKernelDevGuide”中描述的相关内容。下面章节将对任务和中断进行说明。

创建任务

用户使用LOS_TaskCreate(...)等接口来进行任务的创建。具体可以参考example/api/los_api_task.c中的使用方法来创建管理任务。

中断处理

Huawei LiteOS 的中断使用

在驱动开发的过程中我们通常会使用到中断，Huawei LiteOS有一套自己的中断的逻辑，在使用每个中断前需要为其注册相关的中断处理程序。

- OS启动后，RAM起始地址是0x20000000到0x20000400，用来存放中断向量表，系统启动的汇编代码中只将reset功能写入到了对应的区域，系统使用一个全局的m_pstHwiForm[ ]来管理中断。m3以及m4核的前16个异常处理程序都是直接写入m_pstHwiForm[]这个数组的。
- 开发者需要使用某些中断(m3以及m4中非前16个异常)时，可以通过LOS_HwiCreate (…)接口来注册自己的中断处理函数。如果驱动卸载还可以通过LOS_HwiDelete(….)来删除已注册的中断处理函数。系统还提供了LOS_IntLock()关中断及LOS_IntRestore()恢复到中断前状态等接口。详细的使用方法可以参考LiteOS中已经使用的地方。
- LiteOS中断机制会额外地使用2K的RAM，跟大部分开发板bsp代码包中的机制不一样。如果没有动态修改中断处理函数的需要，用户可以选择不使用该中断机制，简单的方法是在los_bsp_adapter.c中将g_use_ram_vect变量设置为0，并且在配置工程时不配置分散加载文件。这样就可以使用demo板bsp包中提供的中断方式。
- 如果使用LiteOS的中断机制，那么在启动LiteOS之前，请先将所有用到的中断都用LOS_HwiCreate()完成注册，否则在完成中断注册前就初始化了相关的硬件以及中断会直接进入osHwiDefaultHandler()导致程序无法正常运行。
- los_bsp_adapter.c中LosAdapIntInit() LosAdapIrpEnable() LosAdapIrqDisable（）等接口都可以调用BSP包中的接口实现。

系统**tick**中断配置修改

- los_bsp_adapter.c中修改后的osTickStart()函数，比如在该函数中直接调用BSP包中的接口配置system tick，在stm32中可以调用SysTick_Config(g_ucycle_per_tick);
- 根据实际配置的system clock 修改sys_clk_freq的值，工程中给出的值都是默认时钟频率。比如stm32f429的默认时钟是16M HZ。

LiteOS资源配置

- 对于嵌入式系统来说，内存都是比较宝贵的资源，因此一般的程序都会严格管理内存使用，LiteOS也一样。在LiteOS中系统资源使用g_ucMemStart[OS_SYS_MEM_SIZE]作为内存池，来管理任务、信号量等等资源的创建，总共是32K。而留给用户创建的task的的个数则是LOSCFG_BASE_CORE_TSK_LIMIT（15）.

- LiteOS中的内存使用都是在los_config.h中进行配置的，需要使用多大的内存，可以根据实际的task个数、信号量、互斥锁、timer、消息队列、链表等内容的个数来决定的（根据各自的结构体大小以及个数计算），总的内存池的大小是OS_SYS_MEM_SIZE来定义的。
- LiteOS的中断机制，目前使用了2K的内存。

移植到不同的芯片

- 移植LiteOS到不同的芯片时，需要在kernel\cpu下去添加一个芯片系列的目录，并且在该新增加的目录下添加los_dispatch，los_hw.c、los_hw_tick、los_hwi这些内容。dispatch主要实现task调度相关的处理以及开关中断获取中断号等内容，los_hw.c中实现的task调度时需要保存的寄存器等内容，los_hwi则是中断的相关内容，los_hw_tick则是系统tick中断处理以及获取tick等的实现

其他说明

目前git上提供的代码中直接提供了IAR和Keil的示例工程，可以直接用来进行参考；将用户自己的代码适配到LiteOS内核工程进行开发的过程，可参考各自开发板移植指导文档。