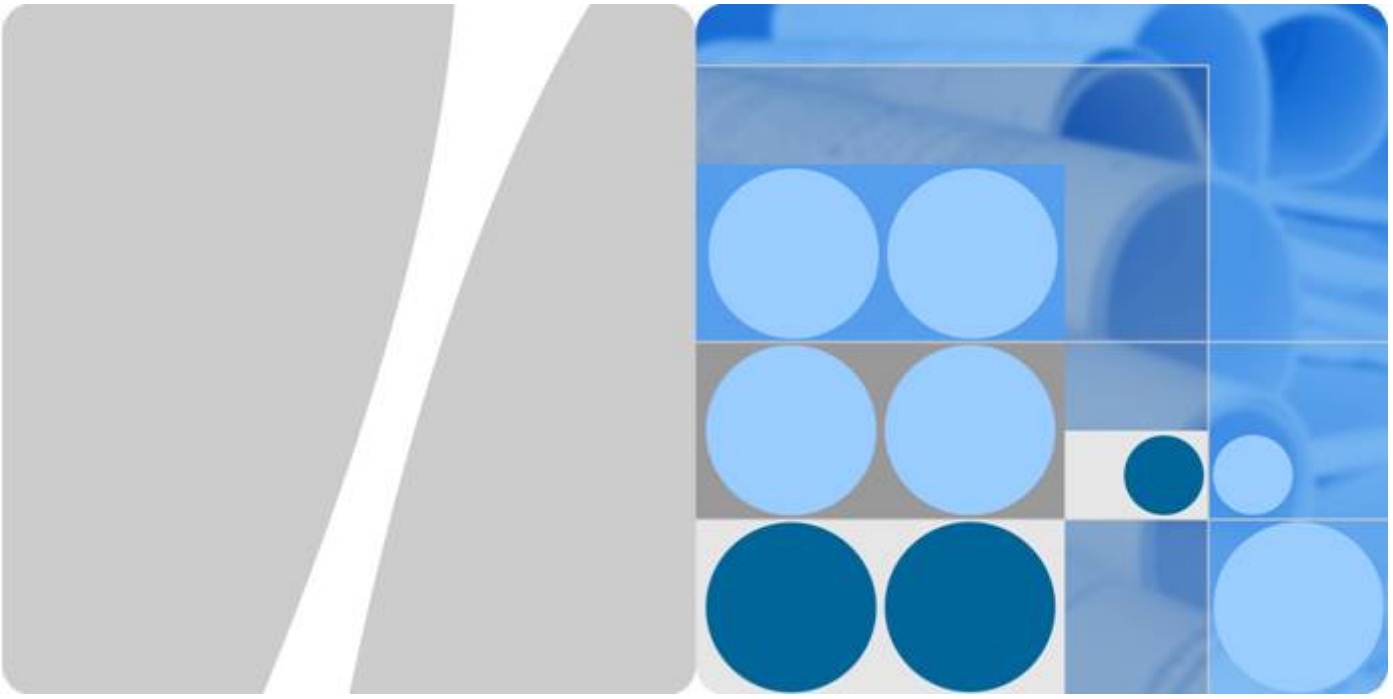


Huawei LiteOS 开源团队

# Huawei LiteOS 第三方芯片移植指南（IAR 版）

文档版本　01

发布日期　2016-11-29

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 前言

## 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：

- 物联网端侧软件开发工程师
- 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。

| 符号 | 说明 |
| --- | --- |
|  | 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<br>“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 |
| --- | --- | --- |
| 2016年10月28日 | 1.0 | 完成初稿 |

# 3 概述

一般来说，用户拿到Huawei LiteOS操作系统，想要基于Huawei LiteOS进行业务开发，主要有两种情况：

1. 将Huawei LiteOS在一款新的芯片上运行起来，根据芯片硬件参数，修改Huawei LiteOS，最终能简单的运行一个程序。
2. 拿到已适配好芯片的Huawei LiteOS代码包，在单板上运行起来，之后基于Huawei LiteOS开发新的模块业务，以及增加新的单板外设。

目前在github上已开源的Huawei LiteOS内核源码已适配好STM32F411芯片，本手册将以STM32F429ZI芯片为例，介绍基于Cortex M4核芯片的移植过程。

# 4 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IAR和st-link的操作系统 |
| IAR(7.30以上版本) | 用于编译、链接、调试程序代码。 |
| 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**

IAR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，需要使用ST-Link。

# 5 获取 Huawei LiteOS 源码

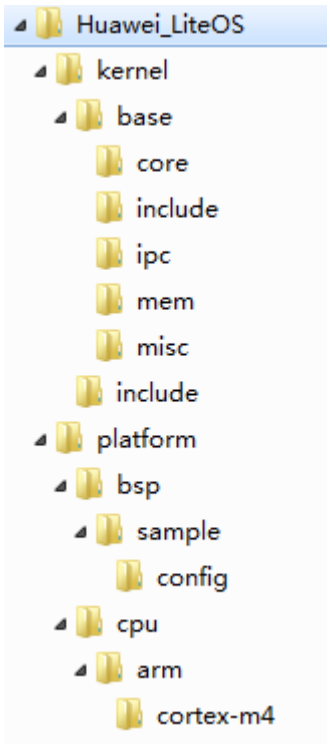
首先我们需要通过网络下载获取Huawei LiteOS开发包。目前Huawei LiteOS的代码已经开源，可以直接从网络上获取，步骤如下：

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


2. 点击”获取源码”按钮,进入源代码下载页面

注：如果想下载源代码，请先注册成为会员。

3. 下载完成所有的源代码之后，将其解压到自定义的一个目录中。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存放的源代码的相关内容详细介绍如下：

| 序号 | 一级目录 | 二级目录 | 说明 |
| --- | --- | --- | --- |
| 1 | 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。 |
| 2 | | Include | 内核的相关头文件存放目录 |
| | Platform | Bsp | 目录下则是应用入口相关示例代码。用户自己实现的相关应用程序源代码都可以放到此文件夹下的子目录或者拷贝sample目录更名为其他名称再添加新的源代码。(注：总入口函数是main函数) |

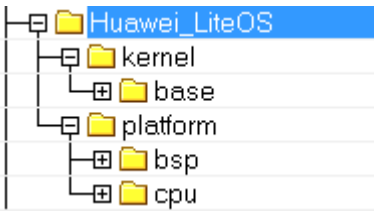
| | | | |
|---|---|---|---|
| | | Cpu | 该目录以及以下目录存放的是与体系架构紧密相关的硬件初始化的代码。此目录最好按照芯片的体系结构以及芯片型号进行命名方便区分。比如目前我们实现了arm/cortex-m4这个芯片对应的硬件初始化内容。用户最好按照这样的划分进行新的芯片型号的添加 |

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《Huawei LiteOS Kernel开发指南》

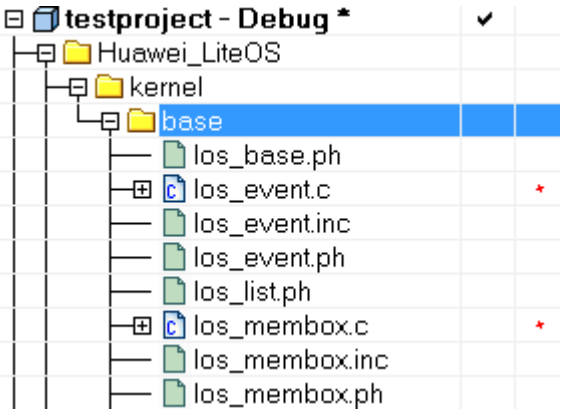
# 6 创建 Huawei LiteOS 工程

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

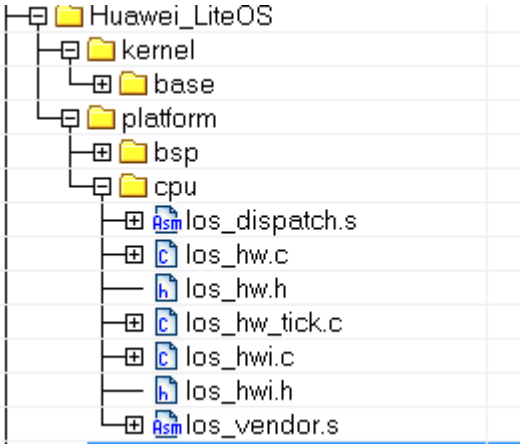
1. 打开IAR Embedded Workbench IDE， 然后点击File->New->Workspace创建一个新的workspace
2. 完成步骤1之后，需要创建一个project。点击Project->Create New Project 弹出一个窗口，选择Empty project然后点击OK按钮弹出保存窗口，输入project名称比如testproject，然后单击保存完成空工程的创建。
3. 步骤2完成后，需要将Huawei LiteOS的源代码添加到工程中。创建工程的过程比较繁琐，详细请参考以下内容。
4. 在空工程中添加如下目录结构（右键点击工程比如：testproject， 然后选择add->add Group，如果不了解请参考IAR的帮助手册）



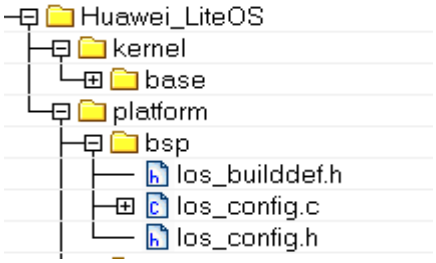

5. 将Huawei_LiteOS\kernel\base下的所有子目录下的文件添加到工程的kernel\base下



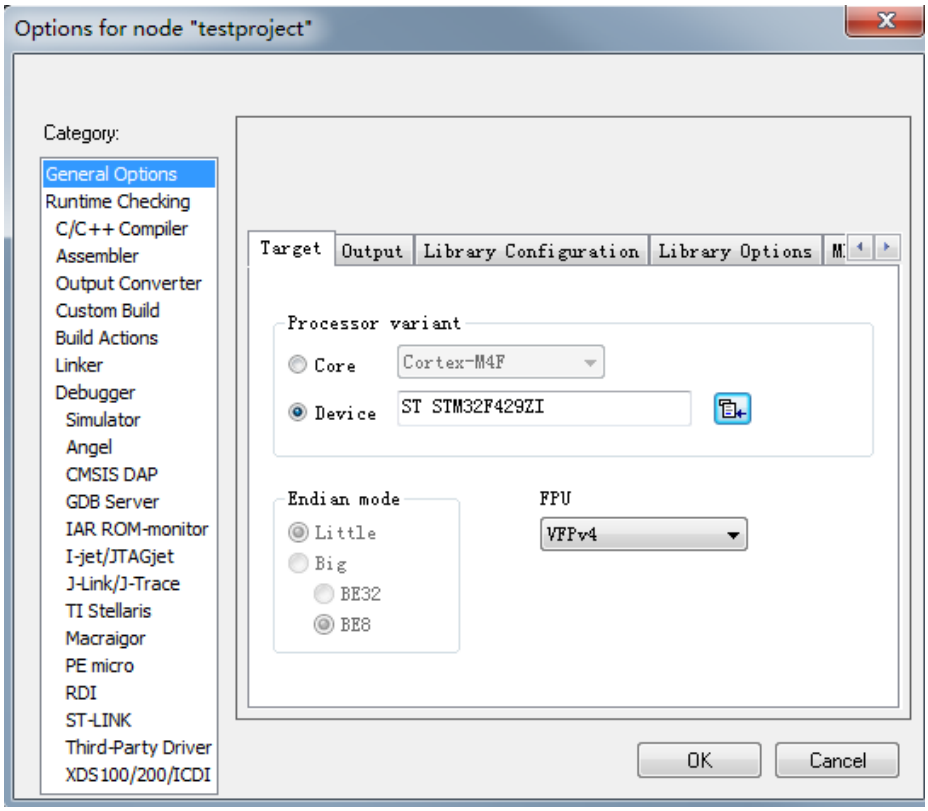

6. 将目前要使用的具体的芯片型号的代码添加到platform\cpu目录下（比如源代码中platform\cpu\arm\cortex-m4）。



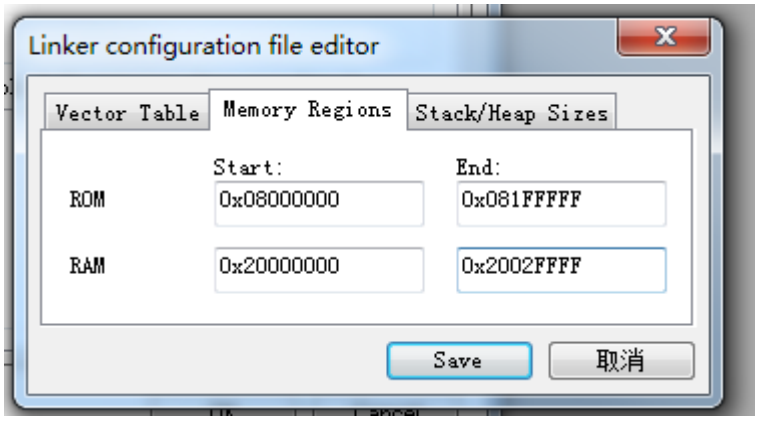

7. 将应用程序的相关代码 （比如platform\bsp\sample\config\ los_config.c)）添加到platform\bsp目录下。注：自己添加的新的应用功能代码如xxx.c之类的文件都可以添加到该目录。



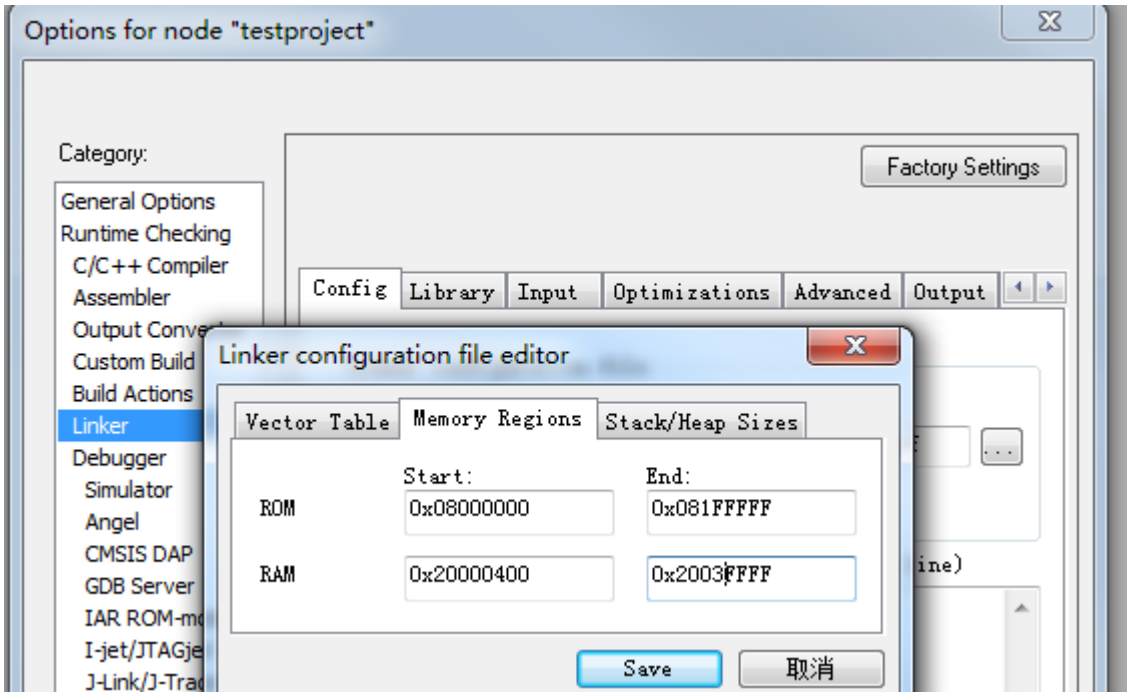

8. 完成代码添加之后需要对工程进行编译选项配置，详细内容如下：
9. 修改芯片平台：Options -> General Options -> Target，修改成对应的Device（比如我们用STM32F429ZI这个开发板）如下图所示：




10. 修改对应的rom和ram的大小，配置好芯片型号后默认的配置范围如下




实际上我们可以根据实际芯片STM32F429ZIT6的实际值来，比如rom是2M，ram是256k。




**说明**

- 以ST芯片为例，RAM不能从0x2000000开始，要预留0x400作为向量表空间。
- 关于向量表所占用的空间的计算：向量表的起始地址是有要求的，必须先求出系统中共有多少个向量，再把这个数字向上“圆整”到2的整次幂，而起始地址必须对齐到后者的边界上。例如，如果一共有32个中断，则共有32+16（系统异常）=48个向量，向上圆整到2的整次幂后值为64，因此向量表重定位的地址必须能被64*4=256整除，从而合法的起始地址可以是：0x0, 0x100, 0x200等。

11. 在链接配置文件中增加对向量VECTOR的相关修改

```
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x08000000;
define symbol __ICFEDIT_region_ROM_end__   = 0x081FFFFF;
define symbol __ICFEDIT_region_RAM_start__ = 0x20000400;
define symbol __ICFEDIT_region_RAM_end__   = 0x2003FFFF;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x2000;
define symbol __ICFEDIT_size_heap__   = 0x2000;
/**** End of ICF editor section. ###ICF###*/

/*-VECTOR SIZE-*/
define symbol __ICFEDIT_region_VECTOR_start__ = 0x20000000;
define symbol __ICFEDIT_region_VECTOR_end__ = 0x200003FF;
/**** End of VECTOR SIZE ###ICF###*/

define symbol __region_RAM1_start__ = 0x10000000;
define symbol __region_RAM1_end__   = 0x1000FFFF;

define memory mem with size = 4G;
define region ROM_region   = mem:[from __ICFEDIT_region_ROM_start__   to __ICFEDIT_region_ROM_end__];
define region RAM_region   = mem:[from __ICFEDIT_region_RAM_start__   to __ICFEDIT_region_RAM_end__];
define region RAM1_region  = mem:[from __region_RAM1_start__   to __region_RAM1_end__];
define region VECTOR_region  = mem:[from __ICFEDIT_region_VECTOR_start__   to __ICFEDIT_region_VECTOR_end_



define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };
define block HEAP      with alignment = 8, size = __ICFEDIT_size_heap__     { };

initialize by copy { readwrite };
do not initialize  { section .noinit };

place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

/*** VECTOR ***/
place in VECTOR_region  { section .vector };
/*** End of VECTOR ***/

place in ROM_region   { readonly };
place in RAM_region   { readwrite,
                        block CSTACK, block HEAP };
place in RAM1_region  { section .sram };
```

1 2 4

**说明**

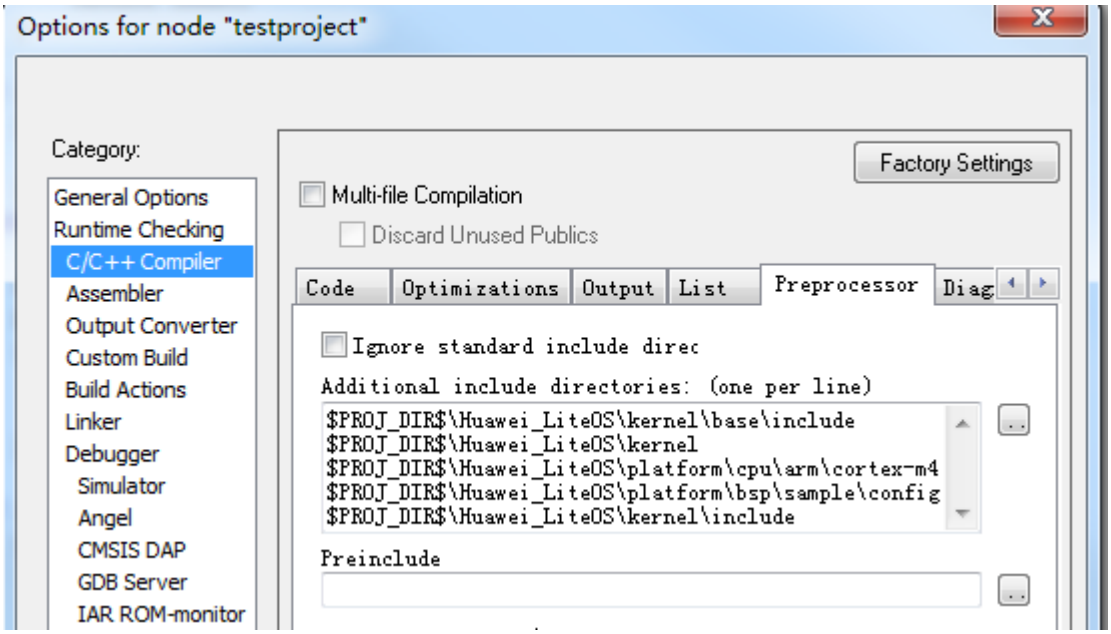
1标识的内容是ram的起始结束地址

2 标识的内容是增加的向量表的起始结束地址

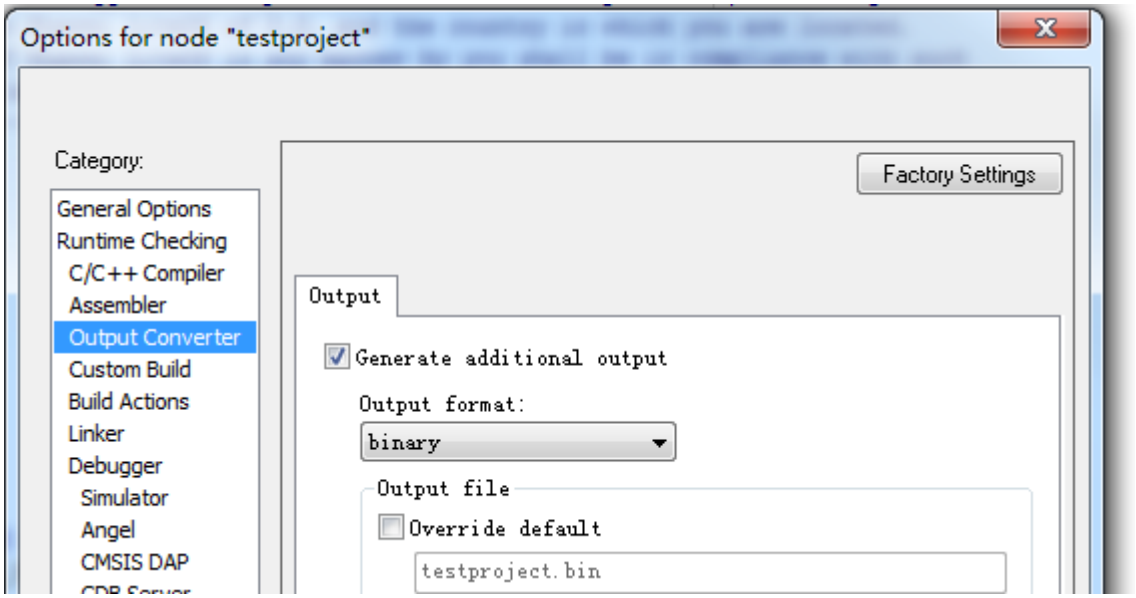
3标识的内容是定义了vector 的区域，

4 是增加的是将向量区域放入到section .vector中。

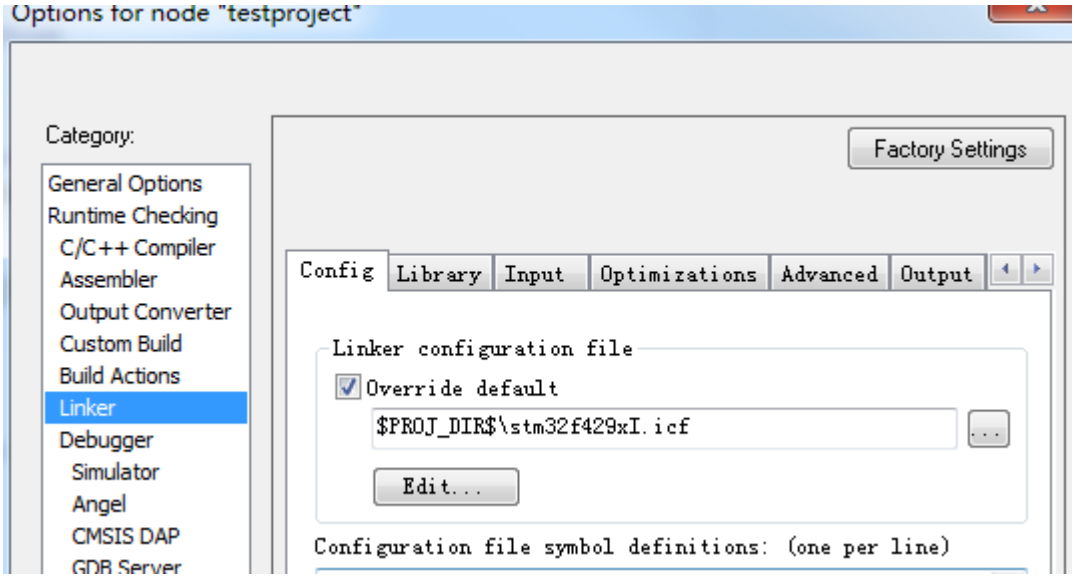
12. 配置编译器的头文件包含路径。我们需要将工程中的所有目录都添加到头文件搜索目录中确保c文件包含的.h文件以及.ph都能够被搜索得到。包含内容如下图所示：




13. 配置OutputConverter，如下图所示：



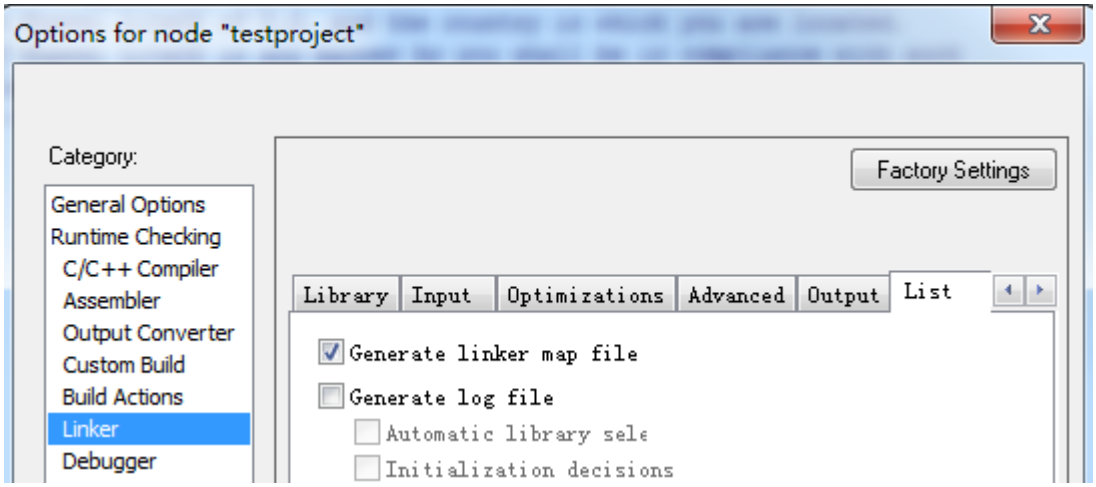

14. 配置Linker使用的配置文件，如下图所示：



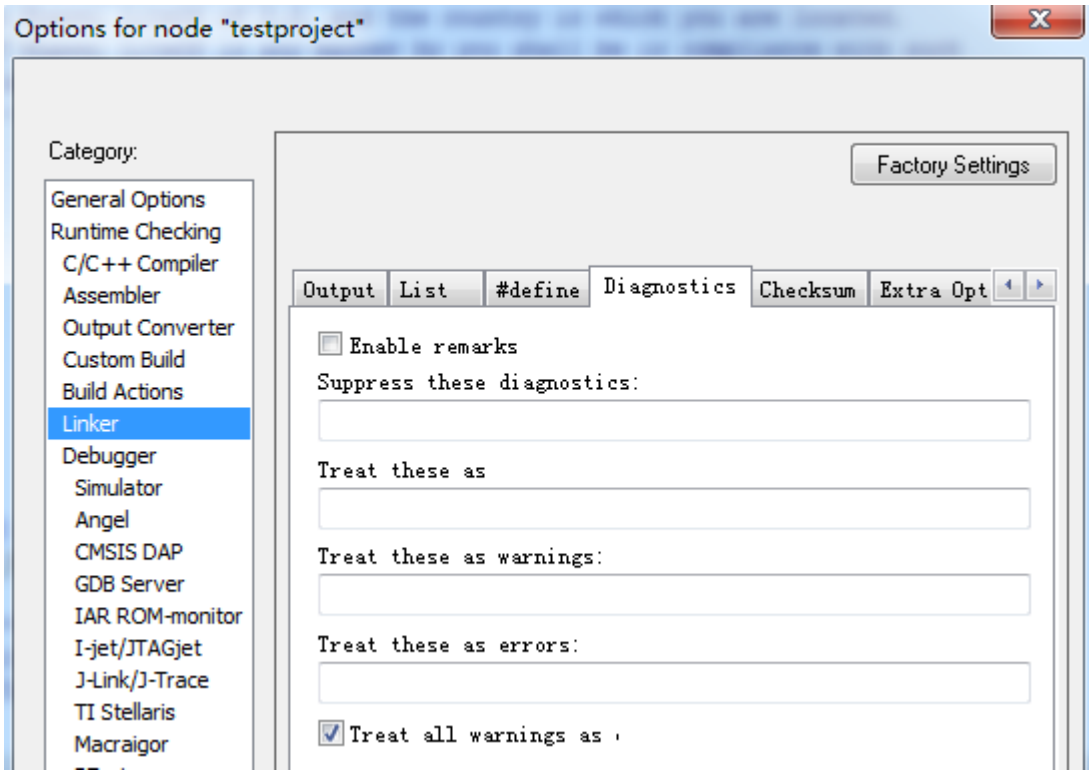

注意：

- 上面的路径选择时是绝对路径(比如：D:\xx\xxx\xxx.icf)可以手动修改成为相对于工程的路径，比如$PROJ_DIR$\xxx\xxx\stm32f429xI.icf.
- 此处选择的文件就是之前配置ram大小的那个配置文件。

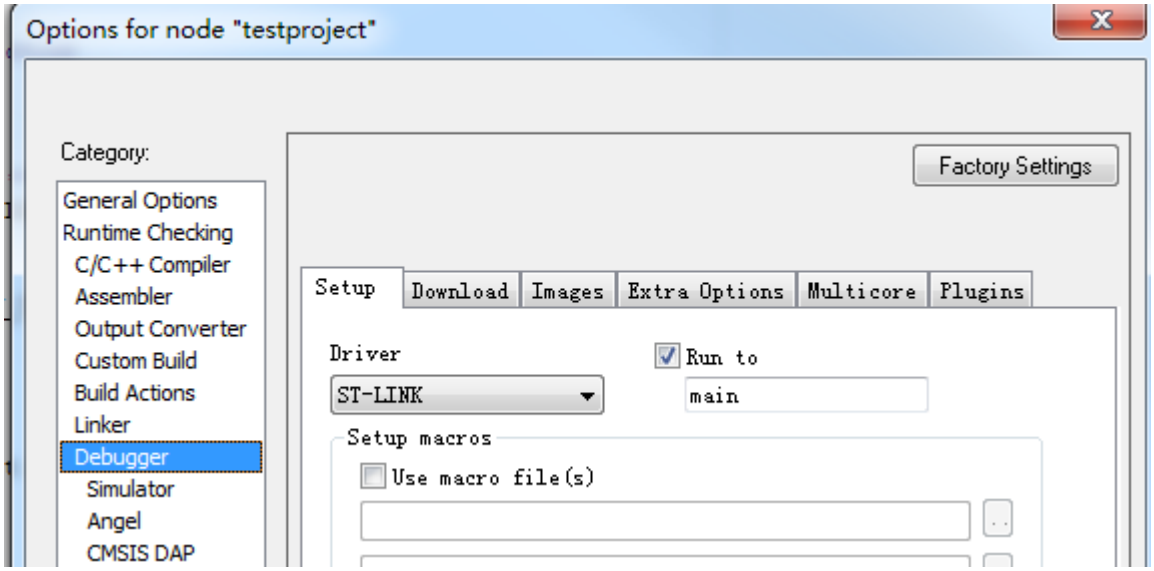
15. 将Linker选项中List选项页配置成如下模式：



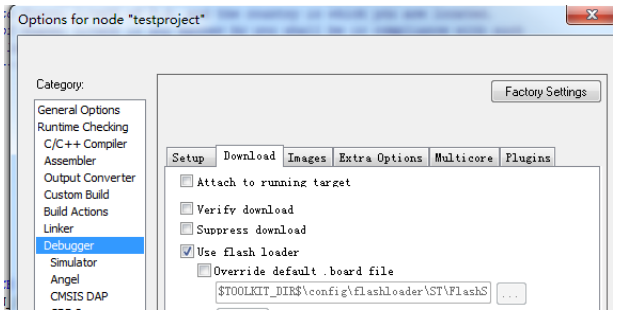

16. 将Linker选项中Diagnostics选项页配置成如下模式：



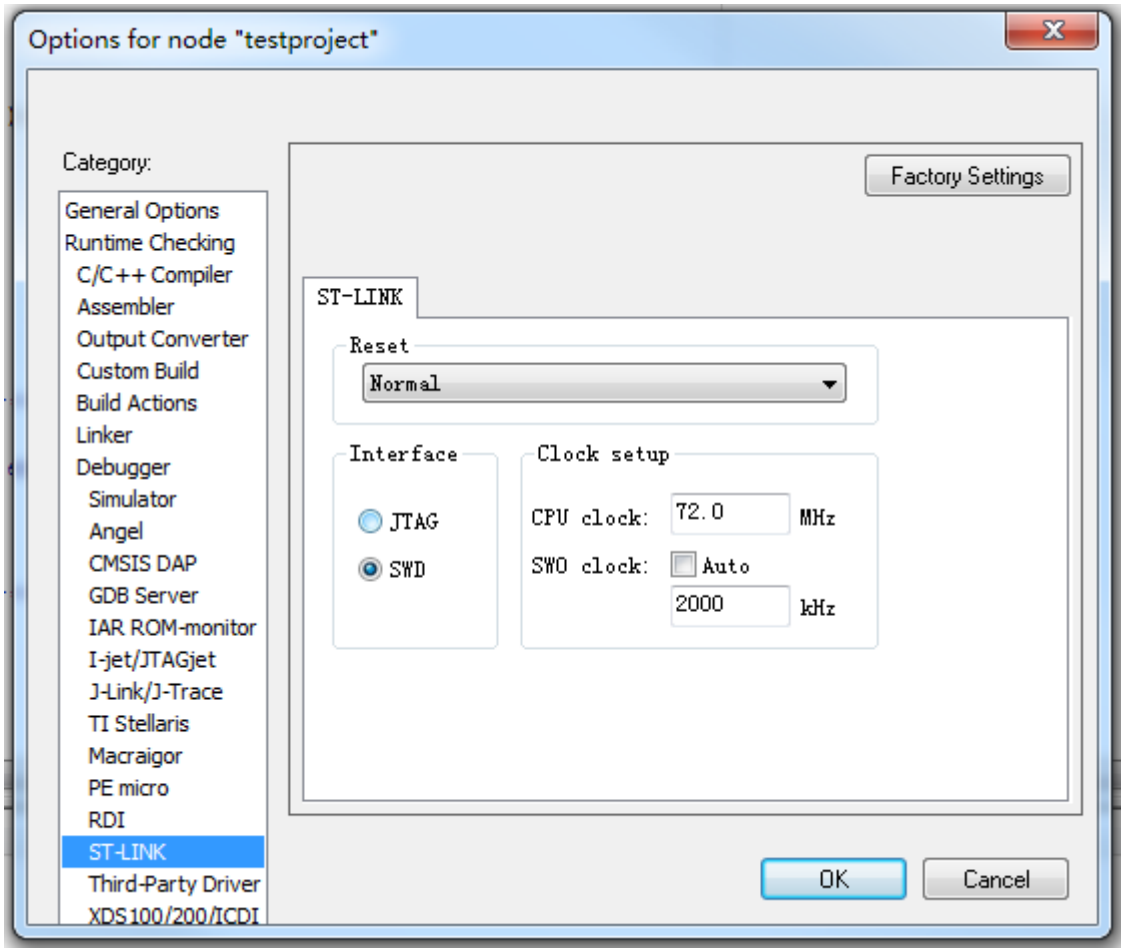

17. 配置Debugger为ST-LINK，此处要根据实际使用的设备进行选择比如有些芯片使用的J-link




18. 配置Debugger下的Download，如下图所示：



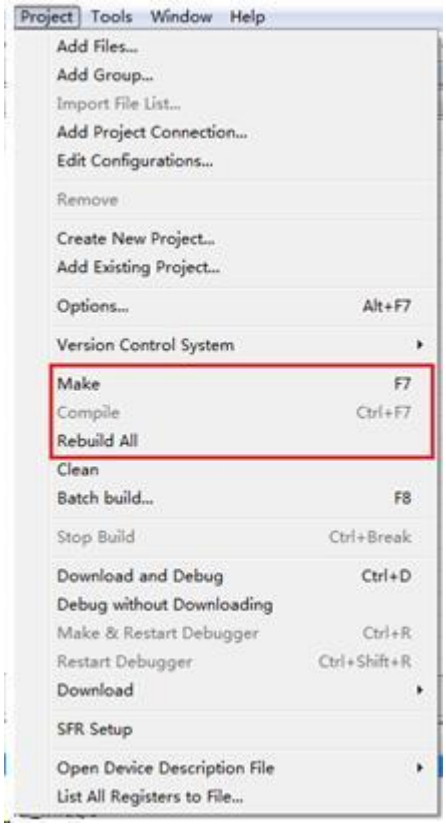

19. 配置ST-link，如下图所示：




以上内容配置正确之后，开发者就可以编译源代码了。

# 7 编译镜像

1. 打开工程后，菜单栏Project→Make、Compile、Rebuild All，可编译文件。这里点击Rebuild All，编译全部文件。

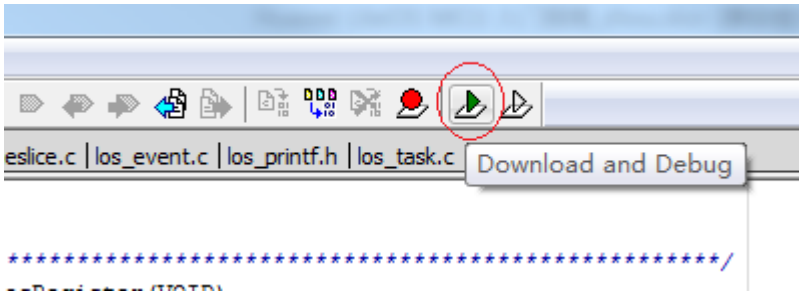



**说明**

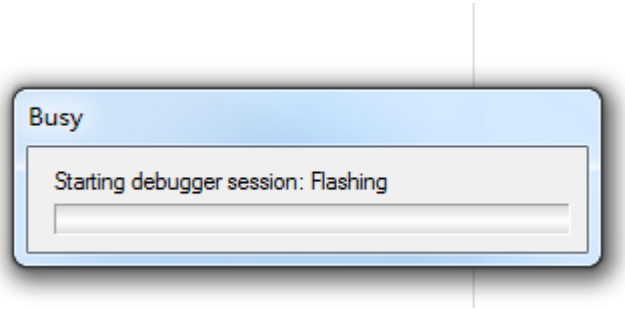
- Make：编译，连接当前工程。（编译只编译有改动文件，或者设置变动的文件，工程窗口文件右边会有个*号）
- Compile: 只编译当前源文件。（不管文件是否改动，或者设置是否变动）
- Rebuild all: 编译，连接当前工程。（不管文件是否改动，或者设置是否变动）

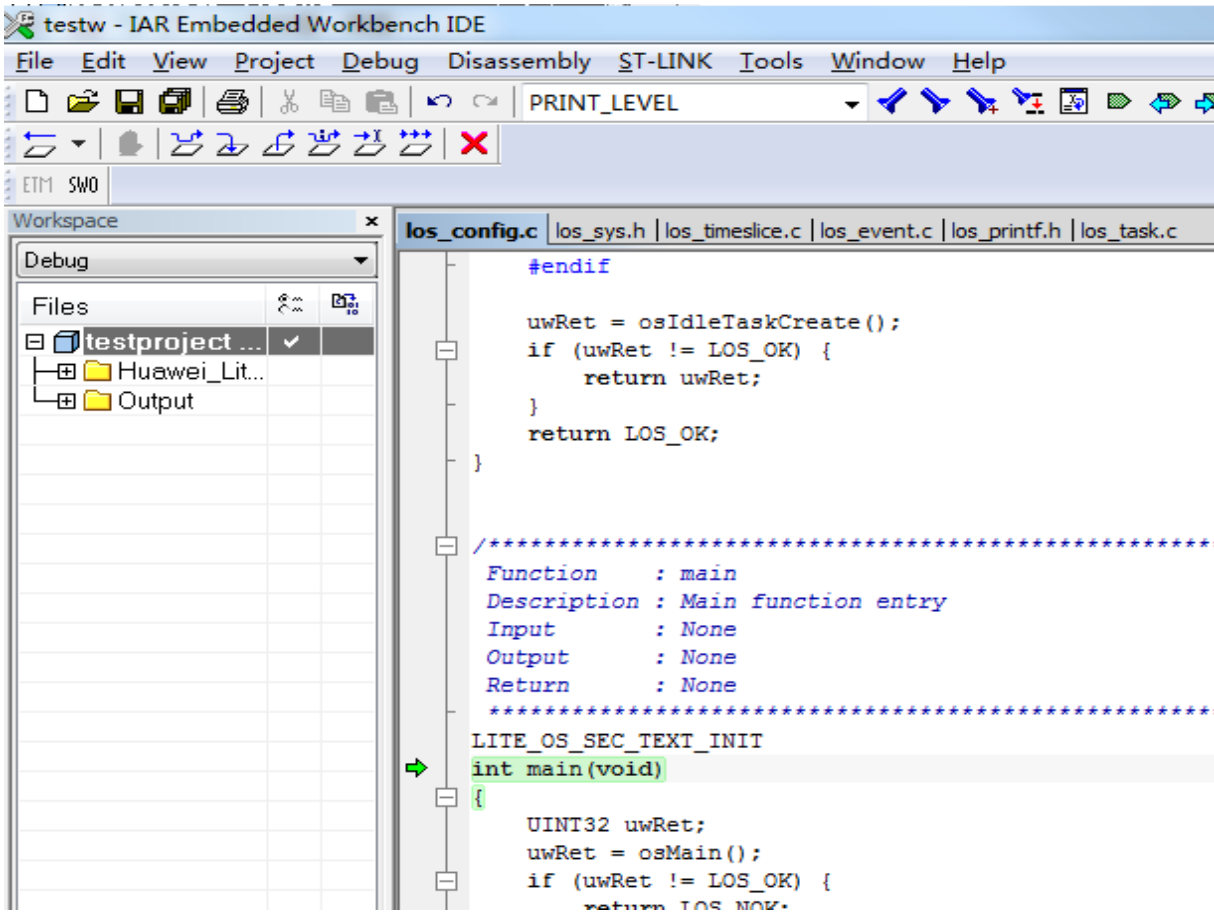
2. 编译成功，error为0；


3. 将开发板与PC使用mini usb连线链接在一起，然后点击Download and Debug可以进行调试。




4. 设备连接正常则会出现如下的界面，






这样就可以开始加断点调试代码了。

# 8 如何基于 Huawei LiteOS 开发

在实际的应用场景中我们需要开发的内容主要包括两个方面：与硬件相关的驱动程序以及基于驱动和内核功能的应用程序。下面将分别进行基于Huawei LiteOS的应用程序和硬件驱动程序进行介绍。

在开始介绍之前我们需要对我们开发的代码的存放位置进行相关的规划，否则代码管理将比较混乱不利于代码阅读以及管理。

1. 应用代码存放位置：建议将应用的代码存在到platform\bsp目录或者该目录下的自定义目录中。
2. 驱动程序存放位置：建议将硬件驱动代码按照设备创建子文件夹放入到platform\cpu\xxx\driver 这样的目录下，比如目前arm的cpu可以是platform\cpu\arm\driver这样的目录。目录不存在可以自己创建。
3. 多个驱动都需要用到的通用头文件可以创建专门的include文件夹来存放。

本章节将以通过GPIO接口控制两个板载LED灯交替闪灭来演示如何基于Huawei LiteOS开发嵌入式应用。

# 8.1 第三方资料获取

在开发之前我们需要获取STM32F429I-DISCO开发板的相关官方文档，

- 用户使用手册
- 芯片的data sheet
- 基于demo板的示例软件包等等。

在将Huawei LiteOS移植到任何其他的平台时，我们都需要获取到芯片厂商提供的demo板以及基于demo板的各种硬件资料和芯片资料等。

STM32F429I-DISCOdemo板的相关资料我们可以根据需要从这里获取**http://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-discovery-kits/32f429idiscovery.html**

# 8.2 开发驱动程序

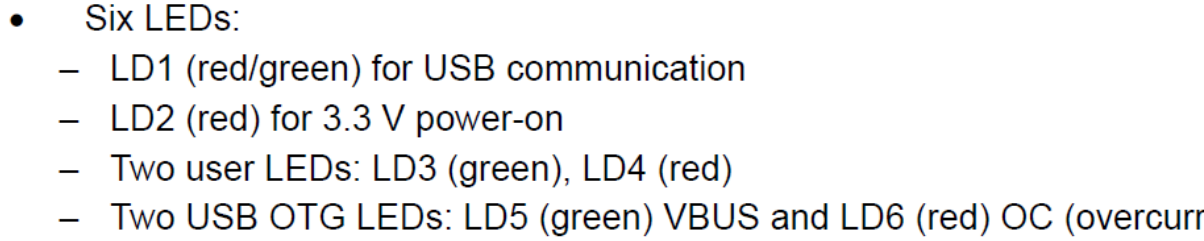
## 8.2.1 Huawei LiteOS 的中断使用

在驱动开发的过程中我们可能会使用到中断，因此我们需要注册自己的中断处理程序。在Huawei LiteOS中有一套自己的中断的逻辑。

- 在OS启动后，ram的起始地址0x20000000到0x20000400是用来存放中断向量表的，并且系统启动的汇编代码中只讲reset功能写入到了对应的区域。并且用了一个全局的m_pstHwiForm[ ] 来管理中断。
- 开发者需要使用某些中断时，可以通过LOS_HwiCreate (…)接口来添加自己的中断处理函数。如果驱动卸载还可以通过LOS_HwiDelete(….)来删除自己的中断处理函数。系统还提供了LOS_IntLock()关中断，LOS_IntRestore()恢复到中断前状态等接口。

## 8.2.2 根据硬件设计开发驱动程序

1. 首先，根据需求我们需要查看开发板上有哪些LED灯。通过查看demo板的用户手册，我们发现有两个用户可以使用的LED灯LD3和LD4。




**说明**

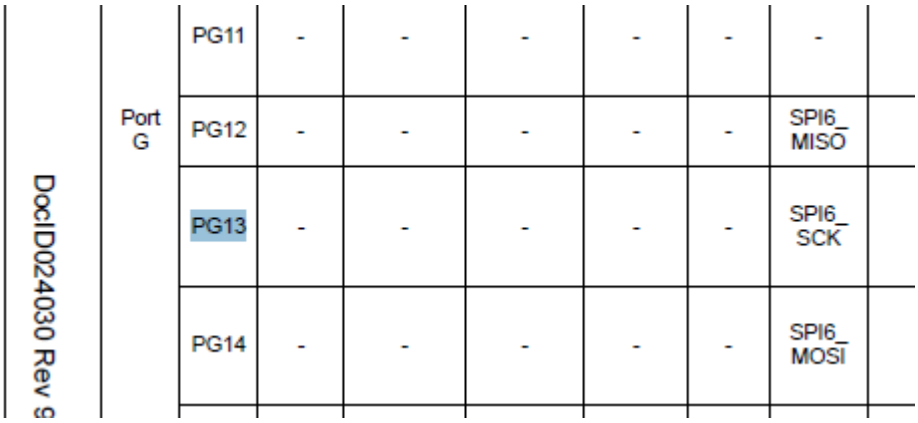
如果是开发其他的设备的驱动程序，需要根据实际硬件原理图进行查找，或者直接让硬件工程师告诉开发者。

2. 然后，在用户手册中找到LD3和LD4属于哪个IO口控制的，比如我们查看到如下内容：

- User LD3:
  The green LED is a user LED connected to the I/O PG13 of the STM32F429ZIT6.
- User LD4:
  The red LED is a user LED connected to the I/O PG14 of the STM32F429ZIT6.

我们可以看到LD3是I/O PG13控制的，LD4是I/O PG14控制的。

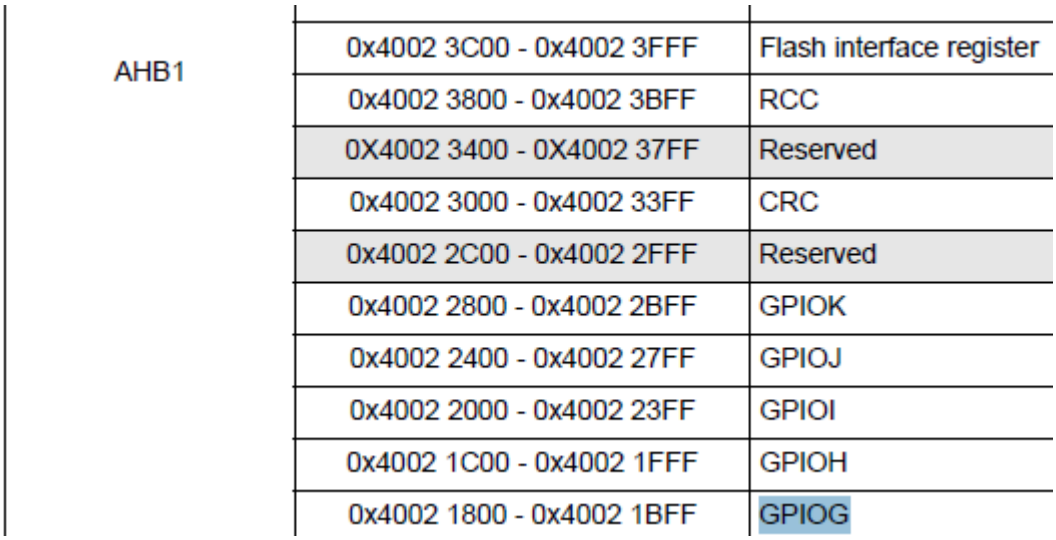
3. 之后我们在stm32f429zit6这个芯片的芯片手册上找到PG14和14的相关IO寄存器信息



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| DocID024030 Rev 9 | Port G | PG11 | - | - | - | - | - | - | |
| | | PG12 | - | - | - | - | - | SPI6_ MISO | |
| | | PG13 | - | - | - | - | - | SPI6_ SCK | |
| | | PG14 | - | - | - | - | - | SPI6_ MOSI | |

从上面我们可以看到PG13和PG14属于Port G（即属于GPIO G控制）

4. 然后继续在芯片手册中找GPIO G的地址相关信息。



| | | |
|---|---|---|
| AHB1 | 0x4002 3C00 - 0x4002 3FFF | Flash interface register |
| | 0x4002 3800 - 0x4002 3BFF | RCC |
| | 0X4002 3400 - 0X4002 37FF | Reserved |
| | 0x4002 3000 - 0x4002 33FF | CRC |
| | 0x4002 2C00 - 0x4002 2FFF | Reserved |
| | 0x4002 2800 - 0x4002 2BFF | GPIOK |
| | 0x4002 2400 - 0x4002 27FF | GPIOJ |
| | 0x4002 2000 - 0x4002 23FF | GPIOI |
| | 0x4002 1C00 - 0x4002 1FFF | GPIOH |
| | 0x4002 1800 - 0x4002 1BFF | GPIOG |

如上图所示，我们可以看到起相关的起始地址和结束地址等信息。我们需要对GPIO控制以及AHB1中其他的与GPIO控制相关的寄存器都进行相关的配置，比如RCC。在芯片手册中没有的话可以在st的官网去查找该芯片的寄存器手册。里面有详细的介绍。

5. 将GPIOG的相关寄存器找到后，我们需要将这些寄存器进行初始化以及设置相应的bit位来控制对应的GPIO的拉高拉低来操控外部的led器件。例如我们在代码中定义

#define PERIPH_BASE 0x40000000U /*!< Peripheral base addr in the alias region*/

#define APB2PERIPH_BASE (PERIPH_BASE + 0x00010000U)

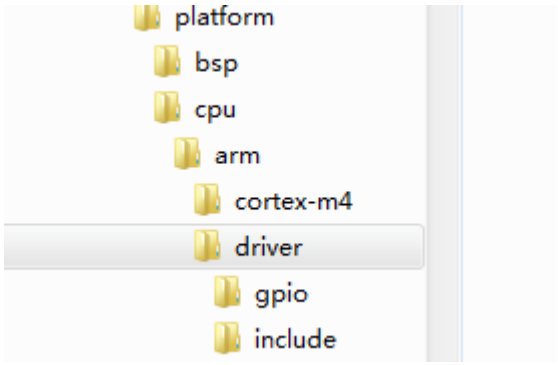
#define AHB1PERIPH_BASE (PERIPH_BASE + 0x00020000U)

#define RCC_BASE (AHB1PERIPH_BASE + 0x3800U)

#define GPIOG_BASE (AHB1PERIPH_BASE + 0x1800U)

注：为了快速的使用相关内容，我们可以将st官方提供的示例代码中的GPIO初始的相关内容都直接复用过来，不需要的代码直接去掉。比如stm32f4xx_hal_gpio.c stm32f4xx_hal_gpio.h文件就是GPIO初始化相关内容，我们可以将其需要用到的基地址定义以及寄存器集合而成的结构体等拷贝到我们的自定义公共头文件los_hw_driver_com.h中。

6. 按照功能和模块将代码放入到不同的目录下




比如我们在platform\cpu\arm目录下创建了driver目录，并在driver下创建了include和gpio两个子目录。然后将GPIO的初始化和设置等功能的实现放入到gpio目录，通用的寄存器地址配置文件放入到include目录下。

**说明**

GPIO的外部接口至少需要提供3个：初始化接口、读、写。目前我们直接复用官方提供给的接口。

7. 代码实现后我们将所有代码以及新增加的目录添加到IAR的工程中以及的头文件搜索路径中。

上述步骤完成之后，我们就可以进行代码的编译，解决在代码移植过程中的编译错误等等。编译完成，驱动程序的第一步就算完成了，剩下的就是开发测试用例测试驱动程序。

我们将在下一个章节中介绍通过一个应用程序来对驱动进行测试。

# 8.3 开发应用程序

## 8.3.1 系统资源介绍

应用开发过程中可能会用到Huawei LiteOS所提供的任务、内存、任务间通信等模块，这里主要介绍经常会用到内存和任务。

- 在Huawei LiteOS的内核中已经实现了很多功能比如task、semphore、mutex、queue、timer等等。这些资源所使用的内存都是在已经分配好的内存池中分配的

（总大小OS_SYS_MEM_SIZE来决定的）。这些系统资源的初始化都在osMain()这个函数中。其中osMemSystemInit()就是系统内存池的初始化。从内存池中申请空间用LOS_MemAlloc()接口。

- 创建task，用户可以参考osIdleTaskCreate()来创建自己的task。Task的优先级别是从0到31，值越小优先级越高。可创建的task的多少受内存池的大小限制。
- 如果想使用semphore、mutex等等的资源，请参考Huawei_LiteOS\kernel\base目录下其子目录下的c代码所提供的相关实现。

## 8.3.2 开发一个 led 灯的示例

1. 首先我们可以在Huawei_LiteOS\platform\bsp\sample\config目录下创建一个c文件用来编写自己的demo。比如los_led_test.c。
2. 在C文件中实现一些接口通过调用GPIO的驱动程序来实现LED的控制，比如实现BSP_LED_init(), BSP_LED_on(), BSP_LED_off()
3. 在C文件中参考osIdleTaskCreate()创建一个自己的task，在task的处理函数中调用我们的LED控制的相关接口。Task函数参考如下：

```
LITE_OS_SEC_TEXT VOID led_test_Task(VOID)
{

    BSP_LED_Init(LED3);
    BSP_LED_Init(LED4);
    while (1)
    {
        BSP_LED_On(LED3);
        printf("led 3 on\n");
        LOS_TaskDelay(500);

        BSP_LED_Off(LED3);
        printf("led 3 off\n");
        printf("led 4 on\n");
        BSP_LED_On(LED4);
        LOS_TaskDelay(500);

        BSP_LED_Off(LED4);
        printf("led 4 off\n");
    }
} ? end led_test_Task ?
```

```
LITE_OS_SEC_TEXT_INIT UINT32 led_test_TaskCreate(VOID)
{
  UINT32 uwRet;
  TSK_INIT_PARAM_S stTaskInitParam;

  (VOID)memset((void *)(&stTaskInitParam), 0, sizeof(TSK_INIT_PARAM_S));
  stTaskInitParam.pfnTaskEntry = (TSK_ENTRY_FUNC)led_test_Task;
  stTaskInitParam.uwStackSize = LOSCFG_BASE_CORE_TSK_IDLE_STACK_SIZE;
  stTaskInitParam.pcName = "ledtest";
  stTaskInitParam.usTaskPrio = 10;
  uwRet = LOS_TaskCreate(&g_uwledTaskID, &stTaskInitParam);

  if (uwRet ! = LOS_OK)
  {
    return uwRet;
  }

  return LOS_OK;
}
```

4. 在完成上面的功能接口以后，还需要在程序的主入口中进行相关的接口调用。比如目前Huawei LiteOS的入口Huawei_LiteOS\platform\bsp\sample\config\los_config.c文件中的main()函数。

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

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

5. 在main中调用led测试应用程序入口led_test_TaskCreate()。这样就完成了应用程序的添加。

在上述步骤完成后，将led的应用代码添加到IAR工程，编译运行就可以看到实际的效果了，LD3和LD4会交替点亮。到此从驱动到应用开发的一个完整流程就完成了。

# 9 如何移植 Huawei LiteOS 到已有系统

本章节将讲述如何在已有平台的基础上使用Huawei LiteOS提供的功能。

# 9.1 修改 icf 配置文件

这个文件的修改是根据芯片提供的硬件来修改的，主要修改片上的rom的大小以及ram的大小。由于Huawei LiteOS使用了自己的中断向量管理机制，所以在配置文件中还需要加入Huawei LiteOS需要的向量的相关区域的定义。详细的修改方法可以参考“创建Huawei LiteOS工程”章节进行配置。

**说明**

这个文件中定义了分散加载的相关内容，Huawei LiteOS主要就是要在其中加入中断向量所需要的地址范围。中断向量的大小根据芯片实际的中断数目进行决定的。如果需要加入其它内容，开发者可以自己定义。

# 9.2 修改 startup_xxx.s 文件

此文件是系统启动文件，目前示例用的demo板可以直接使用开源代码中Huawei_LiteOS\platform\cpu\arm\cortex-m4目录下提供的los_vendor.s，该文件中主要定义的就是Reset相关内容。

比如测试工程中，将stm32官方提供的startup_stm32f429xx.s直接在工程中remove掉，直接使用Huawei LiteOS提供的.s文件。

**说明**

此文件中通常还会定义很多系统用到的中断处理函数。不过建议将中断在一个新的c文件实现，并用Huawei LiteOS的中断向量表m_pstHwiForm[]管理或者LOS_HwiCreate()函数动态添加。

# 9.3 修改 los_dispatch.s 文件

此文件中是与锁以及进程调度相关的一些内容的实现，不同的芯片类型汇编代码以及寄存器都不一样这样需要根据实际的芯片来进行相关的修改。但是功能必须保证跟Huawei LiteOS中提供的汇编代码功能一致。当然用户也可以增加其他需要的功能。

目前Huawei LiteOS的los_dispatch.s中定义的osPendSV()这个函数是比较重要的与进程调度相关。如果涉及到不同芯片的汇编指令不一样需要特别注意。

# 9.4 修改 los_config.h 配置系统时钟主频

此文件中列出了所有的OS配置项，其中包含一个芯片移植过程中必配的选项OS_SYS_CLOCK，需要将其修改为时钟源主频，以保证所有与时钟相关的操作均能按时触发，例如tick中断、任务delay等。

# 9.5 移植系统中断和外部中断

在los_vendor.s中Huawei LiteOS定义了自己的中断向量表，以STM32F429移植LCD示例为例，开发者需要重点关注系统中断里的tick处理，以及注册板载按键触发的外部中断。

1. 系统tick的相关中断。Huawei_LiteOS\platform\bsp\sample\config\los_config.c中的LOS_Start()函数中调用osTickStart()完成了tick的中断处理函数的注册。这样

Huawei LiteOS的调度相关的内容都会在每个tick中断到来时被执行。注：目前tick的配置是通过LOSCFG_BASE_CORE_TICK_PER_SECOND宏控制的，每秒1000次。即两个tick间隔是1毫秒。如果平台底层驱动有需要在tick中断中处理的，请修改osTickHandler()增加相关内容。

例如：stm32的lcd测试程序中注册了tick中断，调用SysTick_Handler()来更新驱动中用的tick。那么使用Huawei LiteOS后，中断处理程序由osTickHandler()接管，我们可以在osTickHandler中直接调用SysTick_Handler()来完成移植。

2. 注册按键响应的外部中断，调用LOS_HwiCreate(EXTI0_IRQn, 0,0,EXTI0_IRQHandler,0)注册（经阅读HAL代码，板载按键的中断号为EXTI0_IRQn，响应函数为EXTI0_IRQHandler）

**说明**

1. Huawei LiteOS启动的时候即Huawei_LiteOS\platform\bsp\sample\config\los_config.c中的main()调用的osMain()再继续调用到osHwiInit()将中断OS_M4_SYS_VECTOR_CNT开始的其他中断都注册成了defaut的处理函数，用户在需要使用外部中断的话（即中断号大于OS_M4_SYS_VECTOR_CNT的中断），则需要通过调用LOS_HwiCreate()来动态的添加。
2. 在OS_M4_SYS_VECTOR_CNT之前的中断都在m_pstHwiForm[]静态地添加。目前最重要的Reset的的中断处理中只是调用osEnableFPU来使能浮点运算，并没有初始化一些芯片其他的配置，如果需要我们可以将某些寄存器的初始化也放到Reset中进行实现。

# 9.6 修改 los_config.h 文件

目前los_config.h中的内容比较匹配arm cortex M4这种类型的芯片。该文件中在进行不同芯片平台移植时需要适配修改的内容如下：

- OS_SYS_CLOCK 这个定义的是芯片的时钟
- LOSCFG_BASE_CORE_TICK_PER_SECOND 每秒的tick数，需要写入到相关寄存器中。
- LOSCFG_PLATFORM_HWI_LIMIT 中断的最大数目，这个需要根据实际的芯片进行定义
- LOSCFG_BASE_CORE_TSK_LIMIT 定义系统中能够创建的task的数目。

其他还有许多的宏，其中代码中有相关的注释可以参考，如果有需要可以修改其中的定义，不过修改这些值需要注意不要超过系统中能够使用的内存的大小。

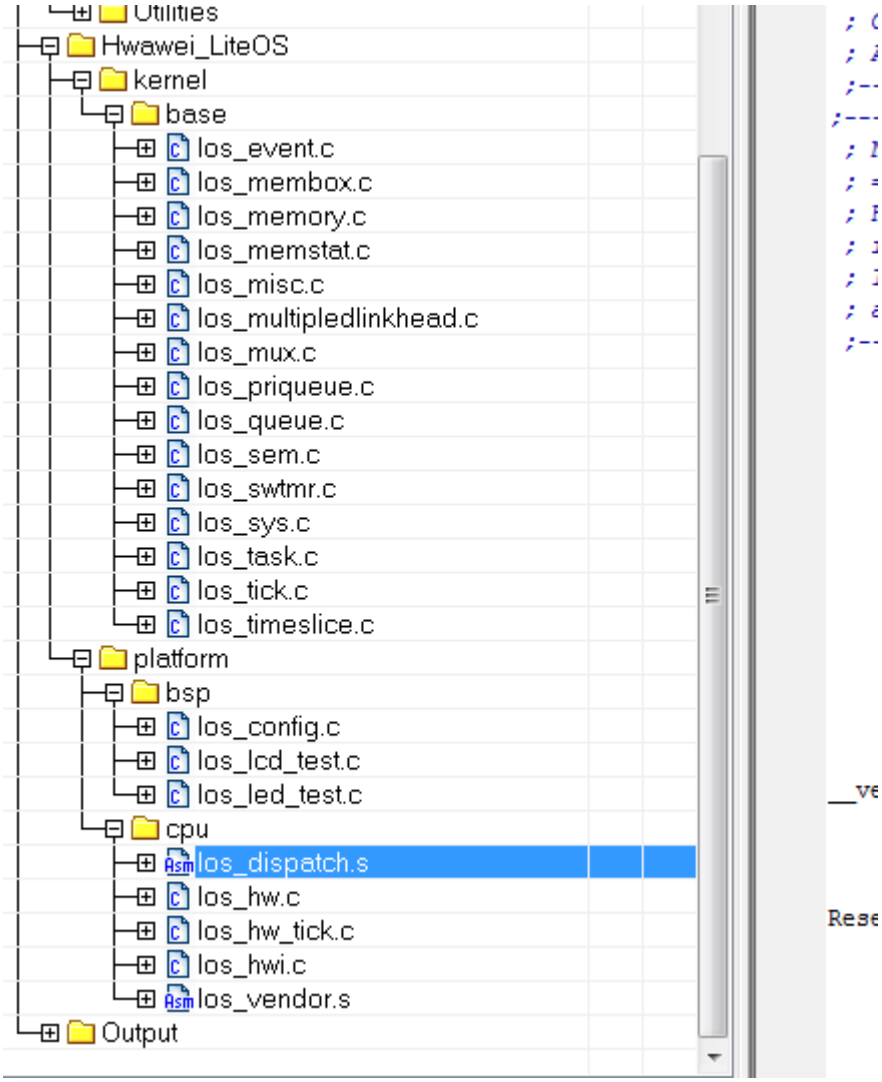
# 9.7 关注 los_hw.c 文件

此文件中的osTskStackInit()中初始化了task stack的 context相关内容，其中这个保存的是task切换时寄存器等相关的内容，具体保存的内容是在los_dispatch.s中实现的。任务切换需要保存哪些内容到这个里面就是根据实际的芯片来决定的了。

# 9.8 添加应用程序

在修改好以上各个章节的内容之后，可以向Huawei LiteOS中添加应用层序了。比如我们创建LED和LCD的测试程序，可以如下实现：

1. 首先将Huawei LiteOS中的相关文件都添加到IAR的工程中，如下图所示

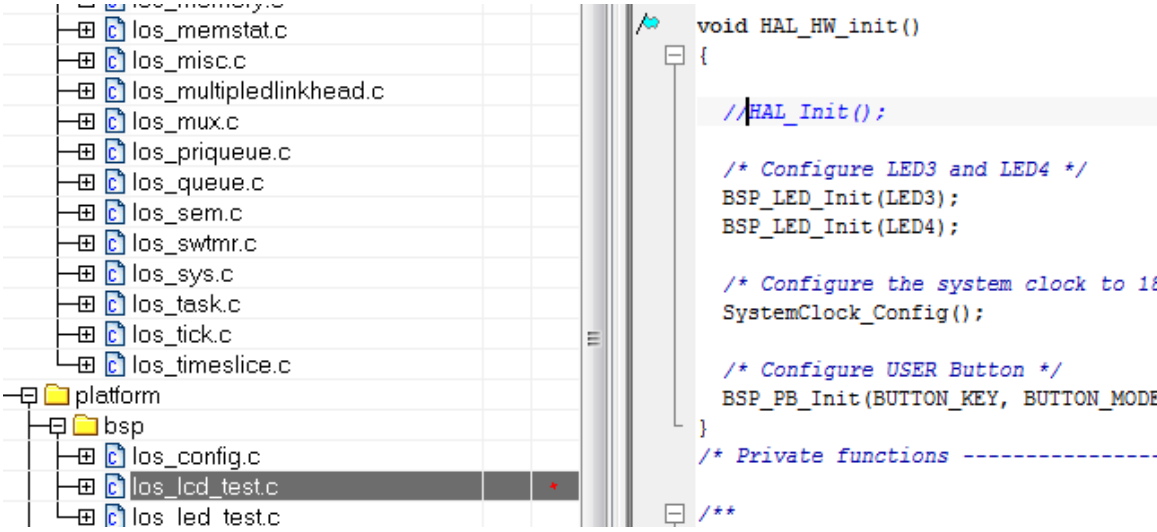



**注意**

Huawei LiteOS中的所有头文件的路径也请在工程的Option->c/c++ Compiler->Preprocessor中添加好，否则编译时会提示无法找到头文件。

2. 参考stm32的官方代码包中提供的LCD的演示代码实现一个LCD测试应用程序，即STM32Cube_FW_F4_V1.13.0\Projects\STM32F429I-Discovery\Examples\BSP\Src\main.c中的内容。比如在bsp下添加的los_lcd_test.c和los_led_test.c分别是lcd的应用和led(LD3\LD4)的应用.

将main.c中的部分硬件初始化抽取出来形成一个硬件初始化函数。




3. 将HAL_HW_init()和SystemInit()函数在los_config.c中的osMain()开头进行调用.

```
{
    UINT32 uwRet;

#if 1
        SystemInit();
        HAL_HW_init();
#endif
```

SystemInit()函数是LCD的驱动程序system_stm32f4xx.c中定义的一些芯片寄存器初始化的内容。这个就根据具体的芯片不一样初始化的内容也将不同。

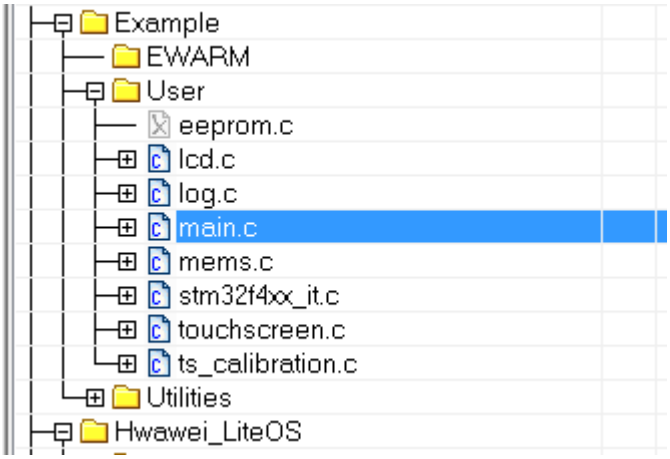
4. 完成上面两个步骤，则可以创建应用task，在task中调用LED的初始化、LCD的初始化，比如lcd中创建lcd_test_TaskCreate()和led中的led_test_TaskCreate()，然后在los_config.c中的main函数中添加调用，如下图所示：

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

#if 1
    uwRet = lcd_test_TaskCreate();
    if (uwRet != LOS_OK) {
        return uwRet;
    }
#endif
#if 1
    uwRet = led_test_TaskCreate();
    if (uwRet != LOS_OK) {
        return uwRet;
    }
#endif
    LOS_Start();

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

5. 以上步骤完成后，将工程中的example/user目录下的main.c进行Remove或者将其中定义的接口都if 0就可以避免和新添加的lcd代码出现重复定义的错误。




最后执行编译运行测试。我们可以看到LCD显示了内容，并且LD3和LD4在交替的亮和灭。

# 10 其他说明

- 示例程序仅仅是示范将stm32官方提供的程序和Huawei LiteOS对接的过程，并没有对程序的可靠性等内容进行详细地验证，因此原始平台中的一些不必要的初始化可能会对程序的正确性和稳定性造成影响，还请忽略这些bug。
- 如果使用非IAR工具，比如使用Keil或者其他编译工具，配置文件将会不同，请参考不同的工具使用的配置文件，对比IAR修改的icf文件增加的内容修改其配置。