## 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.

## 2前言

### 目的

本文档介如何移植Huawei LiteOS到Silicon LAB开发板 SLSTK3401A，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。

[](https://github.com/LITEOS/LiteOS_Kernel/blob/master/doc/meta/kernelapi/danger.png)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

[](https://github.com/LITEOS/LiteOS_Kernel/blob/master/doc/meta/kernelapi/warning.png)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

[](https://github.com/LITEOS/LiteOS_Kernel/blob/master/doc/meta/kernelapi/careful.png)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

[](https://github.com/LITEOS/LiteOS_Kernel/blob/master/doc/meta/kernelapi/notice.png)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

### 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04月10日 | 1.0 | 完成初稿 |
| 2017年04月11日 | 1.1 | 更新为通用移植指南，并增加移植相关注意事项及说明 |

## 3概述

目前在github上已开源的Huawei LiteOS\_Kernel源码已适配好SLSTK3401A芯片的iar示例工程，如果您使用的芯片(开发板)未在其中，可以参照本文档从零开始创建自己的发开工程，并验证移植的结果。

## 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 |
| 开发板 | 基于Cortex-M3或Cortex-M4内核的芯片开发板 |
| PC机 | 用于编译、加载并调试镜像 |
| 电源 | 开发板供电 |

软件环境：

|  |  |
| --- | --- |
| 软件 | 描述 |
| Window 10 操作系 | 安装IAR和相关调试驱动的操作系统 |
| IAR(8.10) | 用于编译、链接、调试程序代码 |
| Starter Kit | Silicon LAB 提供的一系列例程 |

**说明**

## I). IAR 获取[https://www.iar.com/iar-embedded-workbench/?focus=wbselector](https://www.iar.com/iar-embedded-workbench/?focus=wbselector" \t "http://developer.huawei.com/ict/forum/_blank)

II).Starter Kit获取：

[http://www.silabs.com/products/d ... l-gecko-starter-kit](http://www.silabs.com/products/development-tools/mcu/32-bit/efm32-pearl-gecko-starter-kit" \t "http://developer.huawei.com/ict/forum/_blank)

官方使用的Simplicity Studio IDE，需要注册账号，安装成功后，首次打开软件，登陆后，会弹出一个对话框，可以对SDK进行选择，下载完成后的，Starter Kit的路径是：

$(INSTALL\_DIR)\SiliconLabs\SimplicityStudio\v4\developer\sdks\exx32\v4.4.1\

III).安装完软件过程中，提示安装驱动，按照提示安装，安装完成后，J-Trace便已经支持了；

## 5获取Huawei LiteOS 源码

首先下载Huawei LiteOS开发包，步骤如下：

仓库地址：https://github.com/Songzhongrang/PortingContest.git

点击”clone or download”按钮,下载源代码

目录结构如下：

关于代码树中各个目录存放的源代码的相关内容简介如下：

|  |  |  |
| --- | --- | --- |
| 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 |
|  | **SLSTK3401A-SLab** | SLSTK3401A开发板systick以及led/uart/key/lcd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-KEIL | stm32l476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 |
|  | **SLSTK3401A-SLab-IAR** | Slstk3401A开发板的iar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文件中 |

获取Huawei LiteOS源代码之后，我们就可以开始创建自己的project开发工程了，详细内容请参考后续各章节。

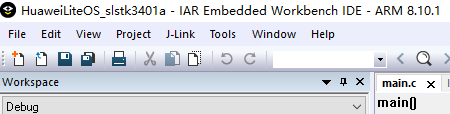
## 6创建Huawei LiteOS 工程

### 6.1 创建工程

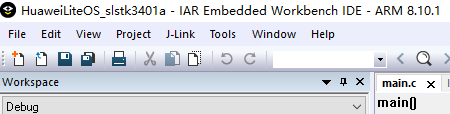
获取到LiteOS内核代码后，如果您本地有开发板相关的驱动代码，可以先将您本地的驱动代码库拷贝到platform\SLSTK3401A-SLab目录下，以便添加到工程中。

在安装好IAR等开发工具后，我们使用IAR集成开发环境创建Huawei LiteOS工程，步骤如下：

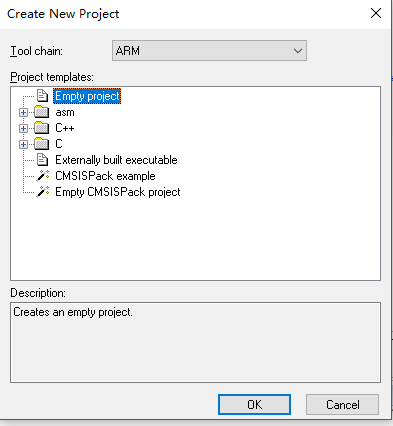
* 打开IAR， 然后点击File->New->Workspace创建一个新的工作空间



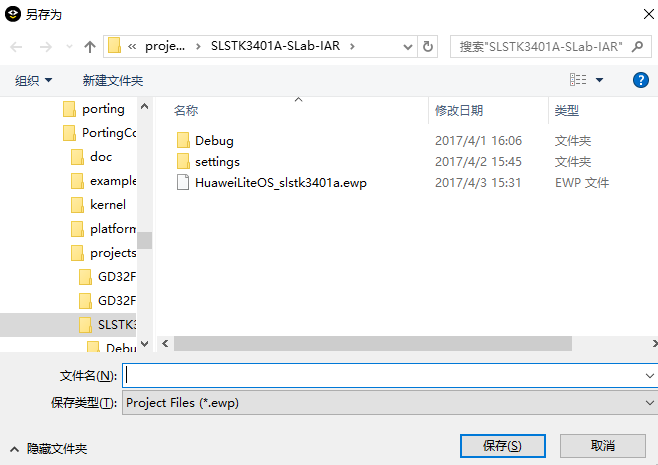
* 然后点击Project->Create New Project...创建一个新的project



* 创建一个空的工程

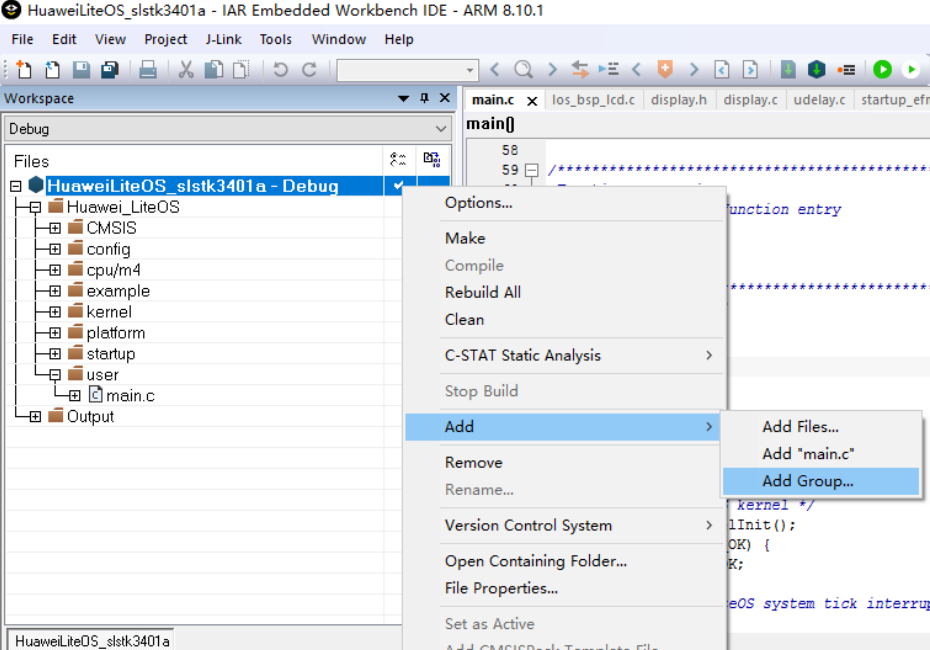


* 保存工程名，比如HuaweiLiteOS\_slstk3401a

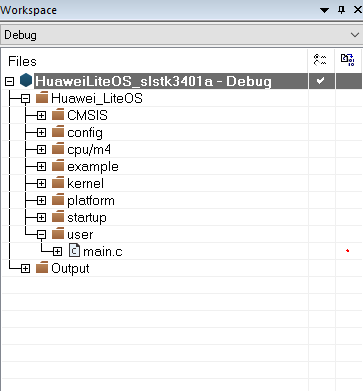


### 6.2 添加kernel代码到工程

* 创建工程中的目录结构



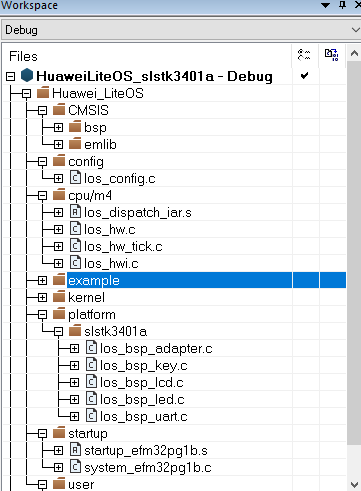
如上图所示，通过add Group 操作来创建一个在工程中显示的目录树，方便我们区分代码功能，比如我们创建如下结构的目录树



创建完成目录树之后我们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文件添加到工程中的kernel下
* 将platform/SLSTK3401A-SLab目录下的所有驱动适配文件，平台无关文件，添加到工程中的platform/slstk3401a下(slstk3401a文件夹名字可自行修改)
* 根据芯片内核型号，将kernel\cpu\arm\cortex-m4（或者cortex-m3）目录下的所有C文件以及汇编代码添加到工程中的cpu/m4（cpu/m3）下
* 将kernel\config目录下的所有C文件添加到工程中的config下
* 将user目录下的所有C文件添加到工程中的user下
* 从Starter Kit中找到例程，可以使用其中的startup\_efm32pg1b.s，因为，SLSTK3401A属于[EFM32PG1B200F256GM48](http://www.silabs.com/products/mcu/32-bit/efm32-pearl-gecko/device.efm32pg1b200f256gm48" \t "http://developer.huawei.com/ict/forum/_blank)系列；添加到platform/startup目录下；
* 将\platform\SLSTK3401A-SLab下驱动文件，平台相关文件，拷贝到工程中的CMSIS目录，可以设置子目录；
* 添加example/api目录下的所有C文件到工程的example目录下

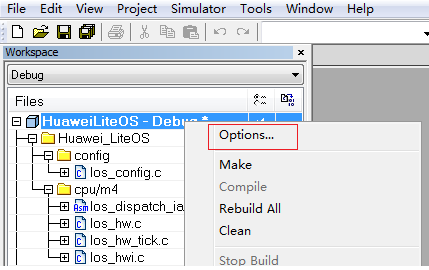
完成代码添加后的工程目录如下图



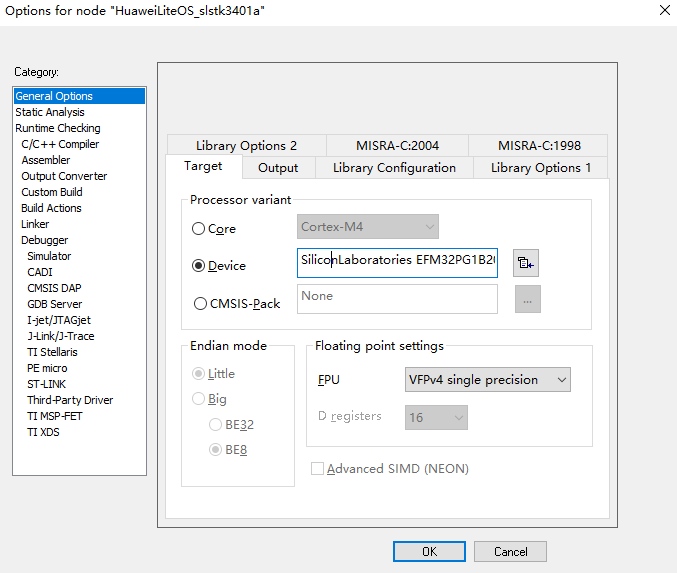
* 完成上面内容后我们先保存一下

### 6.3 配置工程属性

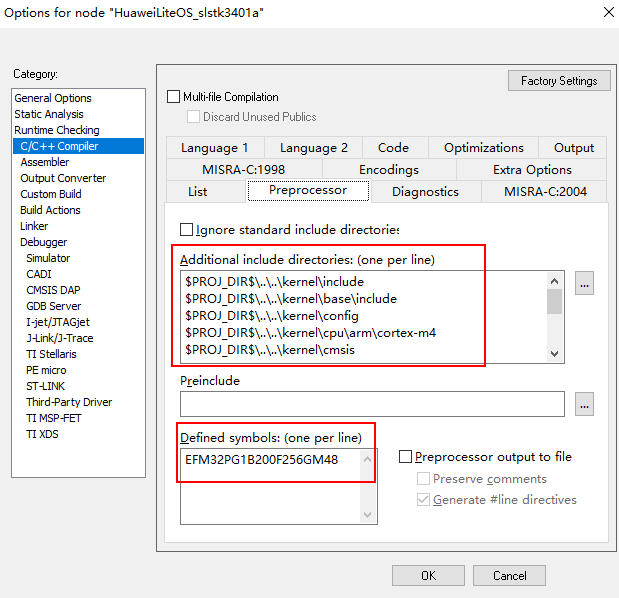
* 完成添加文件之后，需要对工程进行详细的配置，配置内容步骤如下所示：



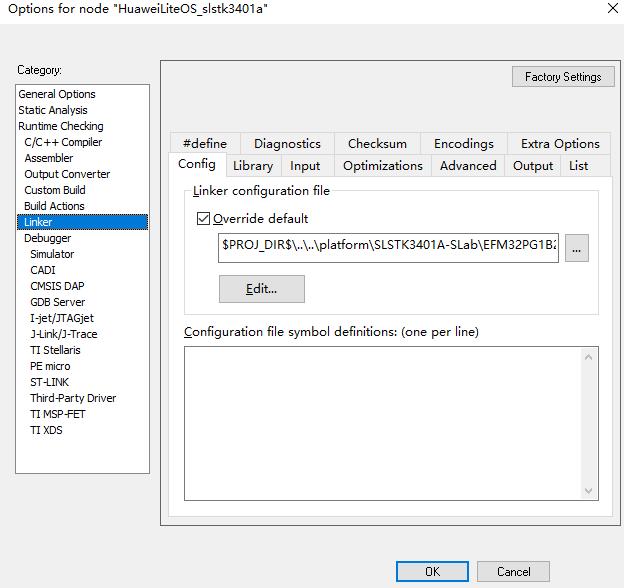
* 配置芯片型号,假如您使用的是SiliconLaboratories EFM32PG1B200F256GM48芯片



* 配置C文件头文件搜索路径，需要将所有的头文件路径都包含进来，定义宏；



* 配置分散加载文件，就是icf文件，IAR默认指向了一个，可以从中拷贝出来，放到某个路径下，然后，覆盖就可以；分散加载的意思，是划分了ro,rw,vector这些section的位置；



如果您需要使用中断动态注册功能，则需要配套使用分散机制。本示例中未使用分散加载机制，可参考源码中其他工程的分散加载文件自己编写。

SLSTK3401A的配置文件内容如下：

/\*###ICF### Section handled by ICF editor, don't touch! \*\*\*\*/

/\*-Editor annotation file-\*/

/\* IcfEditorFile="$TOOLKIT\_DIR$\config\ide\IcfEditor\cortex\_v1\_0.xml" \*/

/\*-Specials-\*/

define symbol \_\_ICFEDIT\_intvec\_start\_\_ = 0x00000000;

/\*-Memory Regions-\*/

define symbol \_\_ICFEDIT\_region\_ROM\_start\_\_ = 0x00000000;

define symbol \_\_ICFEDIT\_region\_ROM\_end\_\_ = (0x00000000+0x00040000-1);

define symbol \_\_ICFEDIT\_region\_RAM\_start\_\_ = 0x20000000;

define symbol \_\_ICFEDIT\_region\_RAM\_end\_\_ = (0x20000000+0x00008000-1);

/\*-Sizes-\*/

if ( !isdefinedsymbol( \_\_ICFEDIT\_size\_cstack\_\_ ) )

{ define symbol \_\_ICFEDIT\_size\_cstack\_\_ = 0x400; }

if ( !isdefinedsymbol( \_\_ICFEDIT\_size\_heap\_\_ ) )

{ define symbol \_\_ICFEDIT\_size\_heap\_\_ = 0x800; }

/\*\*\*\* End of ICF editor section. ###ICF###\*/

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 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 };

keep { section .intvec };

place at address mem:\_\_ICFEDIT\_intvec\_start\_\_ { readonly section .intvec };

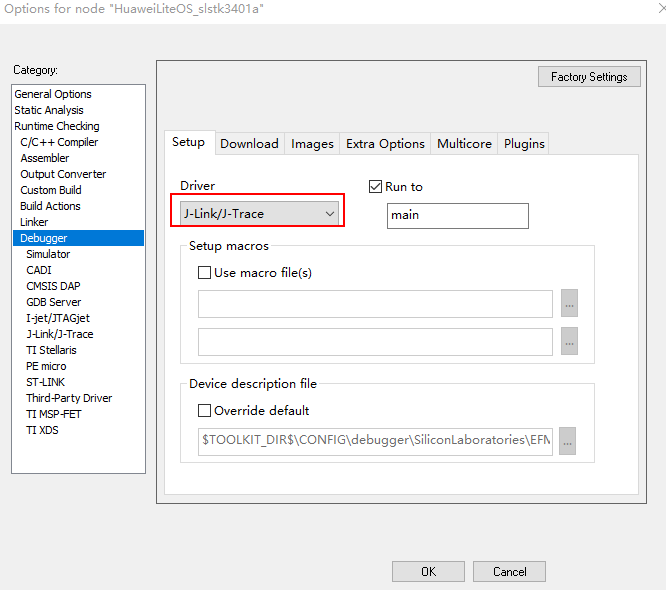
place in ROM\_region { readonly };

place in RAM\_region { readwrite,

block CSTACK,

block HEAP };

* 配置debug相关选项，比如您使用的是slstk3401A芯片，则使用选择,J-Link/J-Trace



## 7适配驱动代码

如果您不需要适配驱动代码到工程，可忽略此章。

* 内核代码中提供了bsp适配的框架代码，存放在platform文件夹下
* 根据本地Library代码提供的接口函数，来实现LiteOS中相关的bsp接口函数

（1）修改los\_bsp\_adapter.c文件，配置系统时钟及SysTick，适配sysTick\_Handler函数；

（2）实现los\_bsp\_led.c、los\_bsp\_key.c、los\_bsp\_uart.c等文件中提供的空函数。

空函数的具体实现可参考源码中已适配的其他工程中同名的文件，如有其他需要增加的驱动功能，可以在同级目录下添加相关文件。

将驱动代码添加到内核工程更详细的过程可参考源码doc目录下其他开发板的移植指南文档。

## 8如何验证移植后的工程

如果您需要验证移植后的LiteOS内核功能，可以参考本章内容。

### 8.1 API测试代码使用

目前LiteOS提供了单独测试每个功能的api代码，可在main()函数中调用los\_demo\_entry.c文件中的LOS\_Demo\_Entry()函数，并放开相应的宏定义。

如果需要一次测试内核所有的功能，则可调用los\_inspect\_entry.c文件中的LOS\_Inspect\_Entry()函数。在本例中，是采用的这种方式；

LiteOS最小需要占用8K的RAM,使用Inspect巡检功能需要再增加1k RAM，不满足此要求的芯片请使用API单项测试功能。SLSTK3401A的RAM是32K，因此，满足要求，我们可以直接调用LOS\_Inspect\_Entry函数；

**使用printf打印的方法**

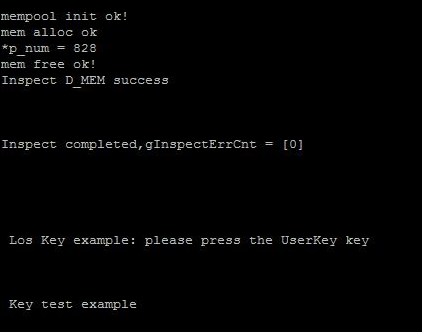
将printf重定向到uart输出，需要uart驱动支持，如果没有适配串口驱动代码，则不建议使用该方法。我们已经适配了串口驱动代码，因此，可以尝试使用printf，具体，可以参考[retargetio.c](https://github.com/Songzhongrang/PortingContest/blob/master/PortingContest_SLSTK3401A-SLab_songzhongrang_0/platform/SLSTK3401A-SLab/driver/retargetio.c" \o "retargetio.c)里的处理；

### 8.2编译调试

* 打开工程后，菜单栏Project→Clean 、Rebuild All，可clean和build 文件。这里点 击Rebuild All，编译全部文件
* 调试运行代码，查看测试结果输出：

(1)如果调用LOS\_Demo\_Entry()函数进行测试，可根据《HuaweiLiteOSKernelDevGuide》文档中列出每项API功能测试结果来进行对比判断。

(2)如果调用LOS\_Inspect\_Entry()函数进行功能巡检，gInspectErrCnt值为0则代表移植成功。



## 9 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。

### 9.1 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。

### 9.2 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。

**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。

### 9.3 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SLSTK3401A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，比如SLSTK3401A的最大时钟是40M HZ。

### 9.4 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。而留给用户创建的task的的个数则是LOSCFG\_BASE\_CORE\_TSK\_LIMIT

根据RAM\_SIZE\_LEVEL\_X宏进行控制，需要根据实际情况进行配置，否则，超过RAM空间限制，在link阶段会报错误；可以通过查看icf分散加载文件和查看生成的map文件进行分析；

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，由于未涉及向量表映射，因此，这部分并没有处理；

关于这里，有些疑问，会在另一篇issue文档中进行阐述；

### 9.5 移植cortex-m3/m4以外其他内核的芯片

* 移植LiteOS到其他内核的芯片时，需要在kernel\cpu下去添加一个芯片所属系列的目录，并且在该新增加的目录下添加los\_dispatch，los\_hw.c、los\_hw\_tick、los\_hwi这些文件。dispatch文件主要实现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

对于RAM较小的芯片，请参照los\_config.h文件中的注释，在工程中定义相关的编译宏，否则会出现链接失败（RAM大于32k可不定义）。

/\* default LiteOS ram size level

RAM\_SIZE\_LEVEL\_0 means kernel ram < 8k ,

RAM\_SIZE\_LEVEL\_1 means kernel ram < 16k,

RAM\_SIZE\_LEVEL\_2 means means kernel ram>=32k

\*/

目前在LiteOS的源代码中有一些已经创建好了的工程，移植到新的开发板(芯片)时可参考源码中的这些工程。

详细的应用编程API请参考《HuaweiLiteOSKernelDevGuide》。

FatFs文件系统移植请参考《LiteOS\_Migration\_Guide\_FatFs\_Keil.md》。