



芯片开放社区
Open Chip Community

阿里云开发者社区
ALIBABA CLOUD DEVELOPER COMMUNITY

平头哥剑池CDK 快速上手指南

教你如何使用平头哥剑池CDK制作自定义SDK

作者:荣放



阿里云开发者电子书系列

- 最完整的适用IOT领域的开发工具指南
- 手把手教你创建自定义SDK
- 支持平头哥自研指令集C-SKY架构和RISC-V架构的芯片开发



平头哥芯片开放社区交流群
扫码关注获取更多信息



扫码注册平头哥 OCC 官网
观看各类蓝牙视频及课程



扫码关注芯片开放社区微信
可获取更多资讯信息



阿里云开发者“藏经阁”
海量免费电子书下载

目录

前言	4
剑池 CDK 开发工具介绍	5
1. 剑池 CDK 工具划分	5
2. 端云一体的开发方式	7
如何创建初始组件化 SDK 工程	8
1. 新建工程结构简介	8
2. 新建平台相关的组件	9
3. 修改链接脚本文件，与平台存储空间相符	10
如何创建一个 Flash 算法文件	12
1. 配置算法初始信息	12
2. 实现算法接口	13
3. 调试算法逻辑	14
4. 配置算法文件到 SDK 工程	18
如何修改初始 SDK 工程	20
1. 新增一个开发板组件	20
2. 更新当前的芯片组件	21
3. 增加两个硬件无关的 common 组件	23
4. 更新当前 solution 工程	25
5. 调试和 Flash 配置	26
如何将制作完成的 SDK 发布给其他开发者使用	27
1. 离线发布 SDK 工程	27
2. 在线发布 SDK 工程	27
常见问题以及解答	30

I 前言

之前，平头哥联合阿里云社区推出了蓝牙电子书及语音电子书：《无需从 0 开发平头哥教你 1 天上手蓝牙 Mesh 应用方案》，《无需从 0 开发 1 天上手智能语音离线方案》，受到了广泛好评。同时有许多用户反馈在开发过程中 SDK 的定义与生成方式的问题，为了解决用户在剑池 CDK 开发环境中制作自定义 SDK 的问题，我们特别推出本书《平头哥剑池 CDK 快速上手指南》，方便开发者可以自定义自己平台的 SDK。

在本书中，第一章介绍了剑池 CDK 工具的基本情况，及端云一体的开发方式。

第二章节开始，描述了如何通过 CDK 创建一个初始的方案，并且新建基于自身平台的相关组件。

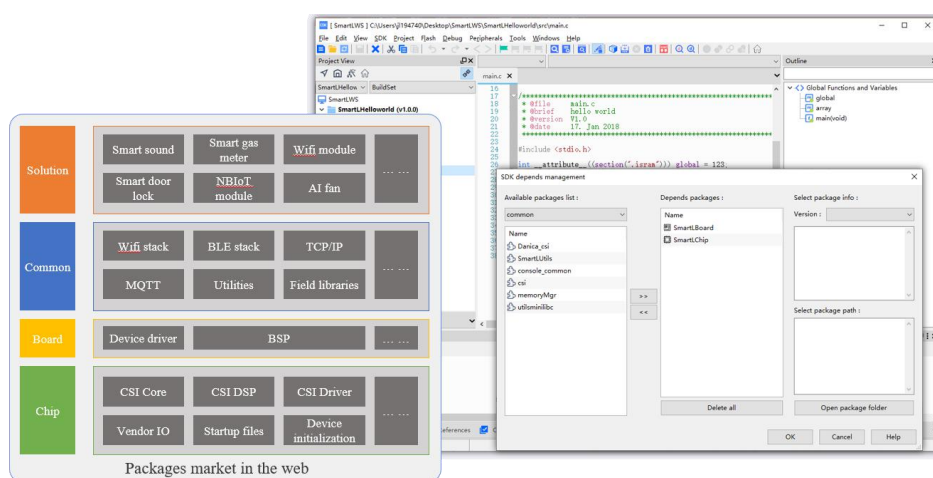
第三章节，描述了一些需要 Flash 烧写的平台，如何创建一个 Flash 算法文件，从而达到使用 CDK 进行 Flash 烧写、调试的目标。

第四章节描述了如何将之前创建的初始方案进一步修改，实现真正平台 SDK 的目标。

第五，第六章节描述了如何将制作完成的 SDK 发布给其他开发者使用，并且对于一些基本问题进行了回答。

I 剑池 CDK 开发工具介绍

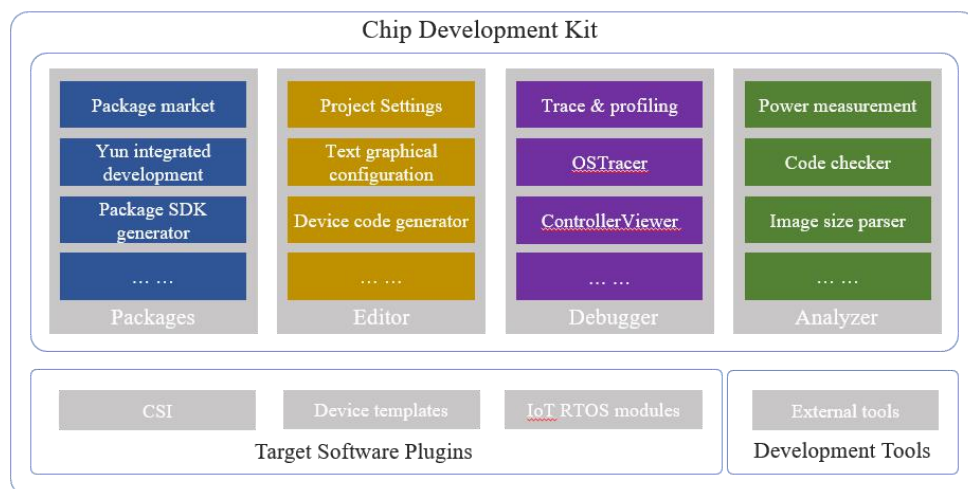
剑池 CDK 是平头哥推出的一款专业面向 IoT 开发领域的集成开发环境，该集成开发环境围绕平头哥“1 天上手，5 天出原型，20 天出产品”1520 技术理念，为开发者提供简洁统一的图形开发界面，帮助开发者进行应用开发。该开发环境目前已支持平头哥自研指令集 C-SKY 架构和 RISC-V 架构的芯片的开发。



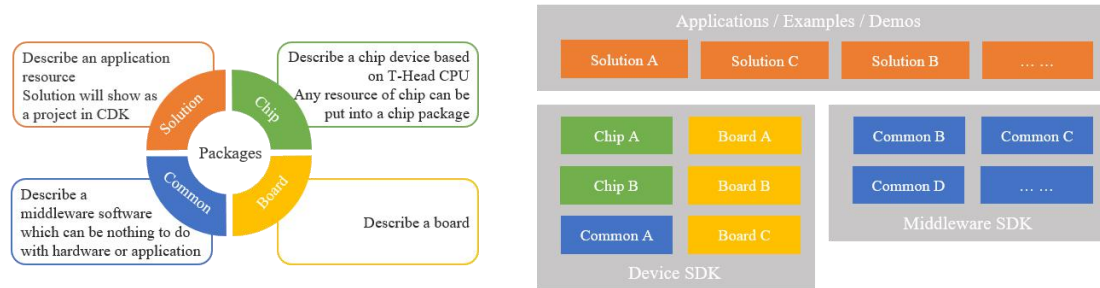
与传统的嵌入式集成开发环境不同的是，剑池 CDK 内部自动对接芯片开放平台，自动获取芯片开放平台上的开发资源。在芯片开放平台上，包含了一个网络组件超市，能够提供各种类型的组件，通过对接网络平台，开发者可以快速的形成自己的方案。

1. 从功能划分角度分析，剑池 CDK 工具分为四部分：

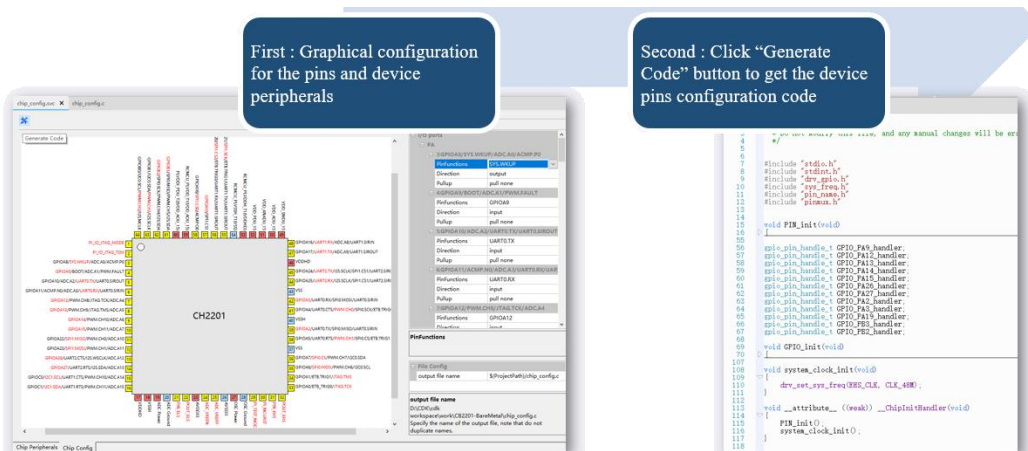
A professional IoT-oriented development tool



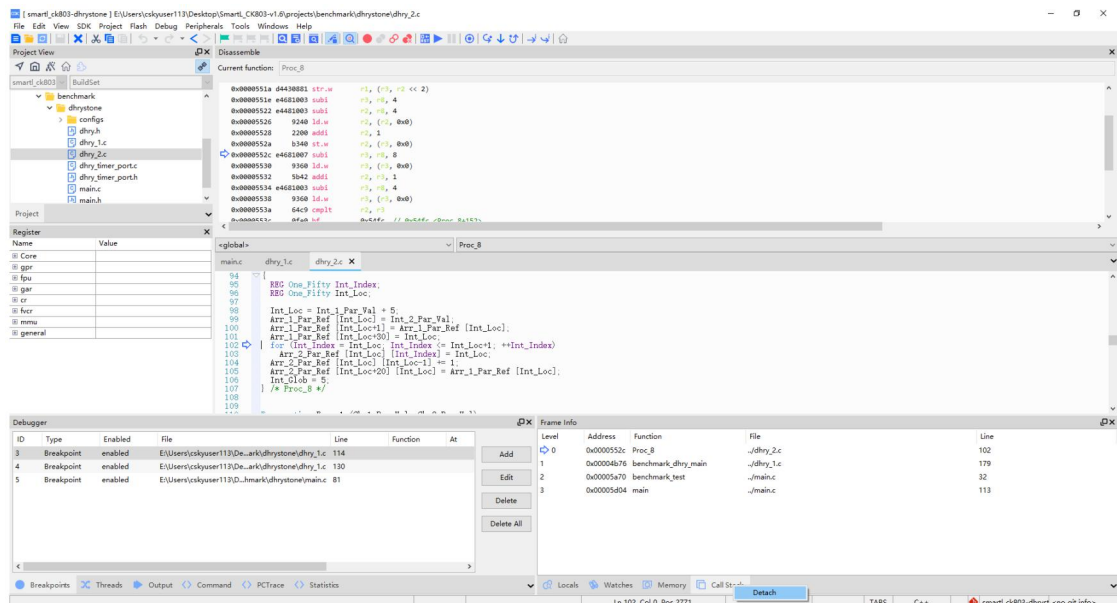
第一部分是 Packages：组件化模块；用于支撑 CDK 的组件化的开发，为开发者提供一个制作松耦合的软件 SDK 的工具。



第二部分是 Editor：编辑器模块；图形化的方式解决嵌入式开发中晦涩、难懂的文本文档和代码编辑。

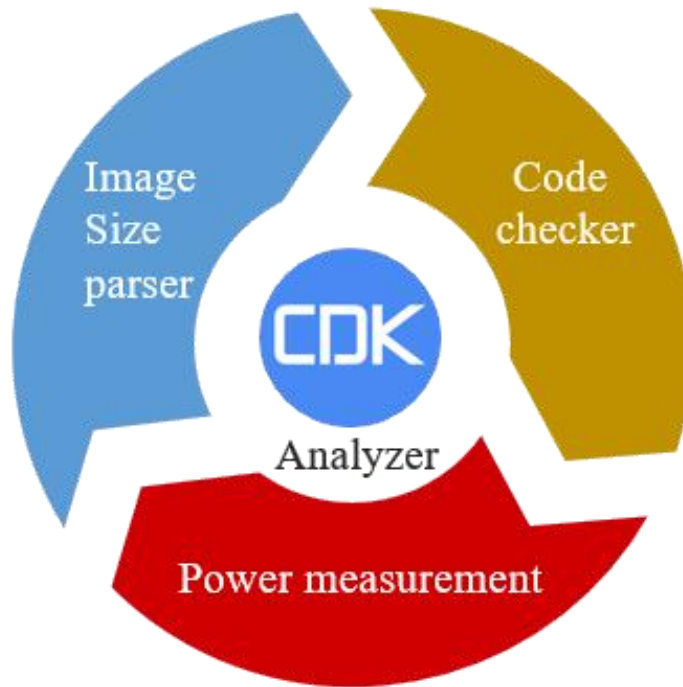


第三部分是 Debugger：调试器模块；图形化的方式提供芯片调试的查看和控制界面。



最后一部分是 Analyzer：分析器模块；为开发者开发出更高效的嵌入式程序而提供的工具。

2. 产品特性：端云一体的开发方式



剑池 CDK 开发工具目前在芯片开放社区有对外链接 (<https://occ.t-head.cn/community/download?id=575997419775328256&inviteUserId=3769057297817612288>，电脑复制链接到浏览器即可) 可供下载安装。

I 如何创建初始组件化 SDK 工程

开发者可以通过本章节中 SmartL 平台学习 SDK 的制作过程,展示 CDK 中制作一个 SDK 的通用做法,为其他 SDK 制作人员提供一些使用引导。

SmartL 平台的基本资料在 OCC 的 [SmartL 平台链接](#) 中。本文使用的 SDK 的源代码,是基于 SmartL 平台的旧版本的非组件化的 SDK 工程,源代码也在 OCC 的 [E802 SDK 链接](#) 中。

创建初始组件化 SDK 工程

1. 新建工程结构简介

菜单栏 Project->New SOC Project 窗口提供创建一个初始组件化的 SDK 工程的入口。

New SOC Project

Name: my_solution

Language: ☒ C ☐ CPP

Project Type: Solution Package

CPU Family: CK800 Series

CPU Name: CK802

Endian: Little

Pacakges Path: \$(ProjectPath)/../my_pack_pool

Project Description:

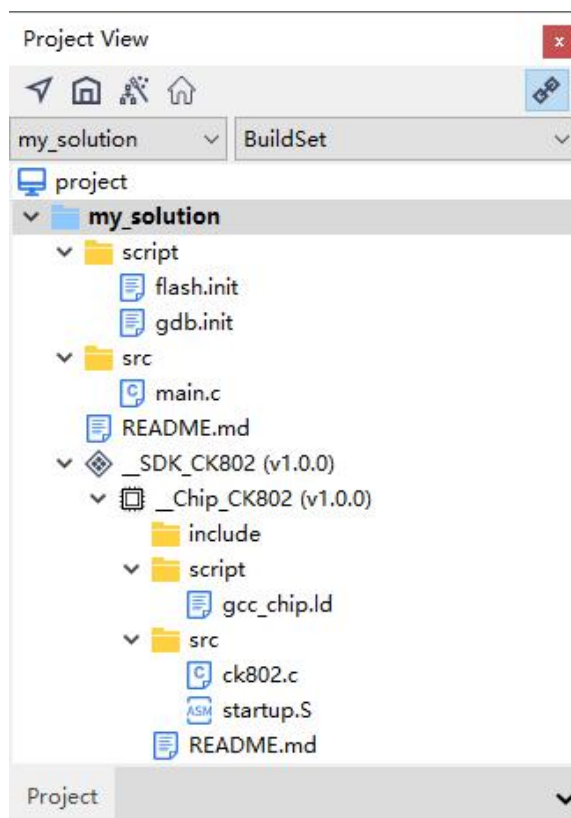
ROM Areas			
In use	Start	Size	Startup
<input type="checkbox"/> ROM1	0x0	0x20000	<input checked="" type="radio"/>
<input type="checkbox"/> ROM2			<input type="radio"/>
<input type="checkbox"/> ROM3			<input type="radio"/>
<input type="checkbox"/> ROM4			<input type="radio"/>
<input type="checkbox"/> ROM5			<input type="radio"/>

RAM Areas			
In use	Start	Size	NoInit
<input checked="" type="checkbox"/> RAM1	0x20000000	0x80000	<input type="checkbox"/>
<input type="checkbox"/> RAM2	0x50000000	0x800000	<input type="checkbox"/>
<input type="checkbox"/> RAM3	0x60000000	0x20000	<input type="checkbox"/>
<input type="checkbox"/> RAM4			<input type="checkbox"/>
<input type="checkbox"/> RAM5			<input type="checkbox"/>

OK Cancel Help

在打开的窗口中输入需要的 SDK 工程名称, Project Type 类型选择 Solution Package, Package Path 设置正确的路径, 该路径用来保存工程使用到的组件。

配置完成以后, 点击 OK, 即完成了一个初始的组件化的 SDK 工程。



整个视图，包含三个部分：

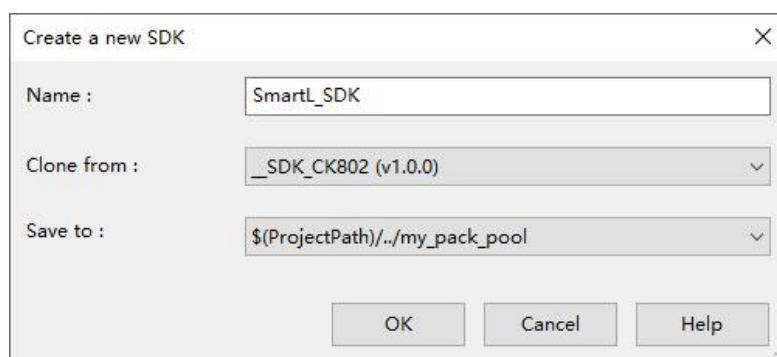
- 1、 my_solution 工程节点，包含 main 文件，以及可以配置的调试初始化脚本和 Flash 初始化脚本。
- 2、 __SDK_CK802 名称的 sdk 类型的组件，该组件依赖了一个 chip 组件。该组件的保存路径是之前新建工程时，配置的 Package Path 指定的路径。
- 3、 __Chip_CK802 名称的 chip 类型的组件，该组件包含了 CPU 的初始化启动代码，以及链接脚本文件。该组件的保存路径是之前新建工程时，配置的 Package Path 指定的路径。

至此，完成了一个初始的 SDK 工程。

2. 新建平台相关的组件

这里的芯片和 SDK 组件都是 CDK 默认的组件，这里需要创建自己的组件。可以根据当前组件，创建自己平台的组件。

右击 my_solution 工程节点，弹出菜单中选择 Create a new SDK，弹出窗口填写需要的 SDK 名称，以及使用 __SDK_CK802 的组件创建自己的新组件。

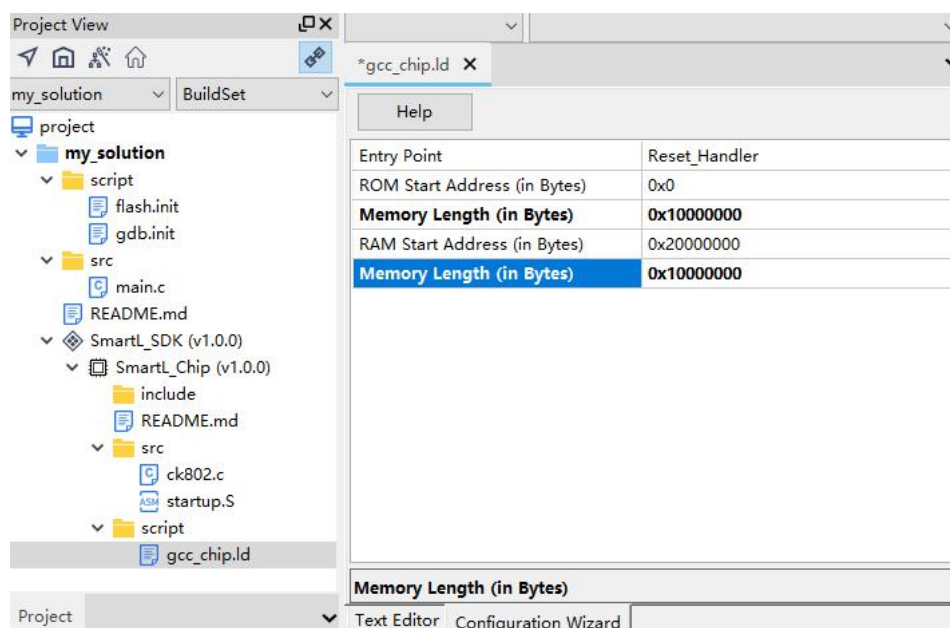


点击 OK，完成 SDK 组件创建，然后右击工程视图的 SDK 组件，选择 Create a new package 选项，然后弹出的窗口中，填写新建的芯片组件名，SmartL_Chip，并选择 __SDK_CK802 作为基础组件去创建。



3. 修改链接脚本文件，与平台存储空间相符

根据 SmartL 平台手册，将默认的 LD 文件的存储区域修改为与 SmartL 平台一致的区间。



CDK 默认的 LD 配置了图形文本修改的方式，如果平台存储区域与默认配置不能匹配（例如超过两段 Flash 或 RAM），那么此时需要根据 GCC 官方的语法要求，完成对应 LD 文件的配置。需要注意的地方在于，默认 LD 文件中的以下符号，不要去随意改动：

符号名	功能/作用
__s_ram_data_1	定义了程序数据区的起始位置
__e_rom	定义了程序只读数据区以后，4 对其的位置
__e_ram_data_1	定义了程序数据区的结束位置
__e_ram_bss_1	定义了程序的 BSS 区域的结束位置
__s_ram_bss_1	定义了程序的 BSS 区域的开始位置

这些符号，在组件 SmartL_Chip 中的 src/ck802.c 文件中使用到。

I 如何创建一个 Flash 算法文件

CDK 支持 Flash 烧写功能，为了能够适配不同的 Flash 器件的编程逻辑，CDK 允许开发人员通过 Flash 算法文件的适配，实现 Flash 器件的编程，从而实现对某个芯片平台的 Flash 烧写。

关于 Flash 算法文件的内容，可以参考《Flash 算法的开发和使用》视频教程进行详细了解。

1. 配置算法初始信息

要在 CDK 中制作一个特定芯片的方案，需要通过 New SOC Project 创建一个 Flash 类型的 CDK 工程。

New SOC Project

Name: SmartL_Flash

Language: ☒ C ☐ CPP

Project Type: Flash

CPU Family: CK800 Series

CPU Name: CK8002

Endian: Little

Pacakges Path:

Project Description:

ROM Areas

In use	Start	Size	Startup
<input type="checkbox"/> ROM1	0x0	0x20000	<input checked="" type="radio"/>
<input type="checkbox"/> ROM2			<input type="radio"/>
<input type="checkbox"/> ROM3			<input type="radio"/>
<input type="checkbox"/> ROM4			<input type="radio"/>
<input type="checkbox"/> ROM5			<input type="radio"/>

RAM Areas

In use	Start	Size	NoInit
<input checked="" type="checkbox"/> RAM1	0x20000000	0x80000	<input type="checkbox"/>
<input type="checkbox"/> RAM2	0x50000000	0x800000	<input type="checkbox"/>
<input type="checkbox"/> RAM3	0x60000000	0x20000	<input type="checkbox"/>
<input type="checkbox"/> RAM4			<input type="checkbox"/>
<input type="checkbox"/> RAM5			<input type="checkbox"/>

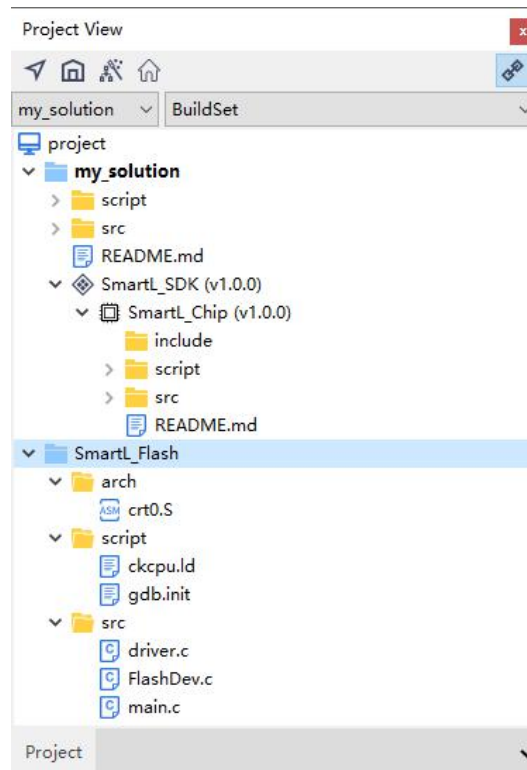
OK

Cancel

Help

此时 Project Type 选为 Flash，并在 RAM1 中填写 SmartL 的 RAM 区域。CPU 和对应 SDK 的 CPU 要完全一致。

完成工程创建以后，工程视图显示内容如下：



FlashDev.c 文件用来对算法基本信息进行描述。

```
/**
 * structure to describe flash device
 */
struct FlashDevice const FlashDevices INDEVSECTION = {
    6,                      // Reserved version description, do not modify!!!
    "SmartL_Flash",         // Flash name
    "ck802",                // CPU name, must in low case
    0x123456,                // Flash ID
    "NorFlash",             // type
    512*1024,               // Reserved
    1,                      // Access directly
    1,                      // RangeNumbers
    // {start address, the flash size, sector size}
    {{0x0, 0x80000, 0x200}}
};
```

将默认的内容修改为 SmartL 硬件信息相匹配即可。

2. 实现算法接口

Driver.c 文件用来实现算法逻辑，针对具体的硬件信息，实现对应的接口。

接口定义	功能说明
Int flashInit()	@brief Flash 编程初始化接口，运行 Flash 程序，开始编程之前会执行该接口； @return 返回 0 表示成功，否则失败；
int flashUnInit()	@brief Flash 编程结束接口，CDK 在 Flash 操作结束之后，会默认执行此接口； @return 返回 0 表示成功，否则失败；
int flashID(unsigned int* flashID)	@brief 获取 Flash ID 接口； @param flashID 返回 flash ID； @return 返回 0 表示成功，否则失败；
Int flashProgram(char* dst, char*src, int length)	@brief Flash 烧写接口； @param dst 烧写目标地址； @param src 烧写源数据地址； @param length 烧写源数据长度； @return 返回 0 表示成功，否则失败；
int flashRead(char* dst, char*src, int length)	@brief Flash 数据读取接口；当该 Flash 不是直接可读的 Flash 属性时，CDK 会调用该接口读取数据； @param dst 读取目标地址； @param src 数据保存地址； @param length 读取数据长度； @return 返回 0 表示成功，否则失败；
Int flashErase(char* dst, int length)	@brief Flash 擦除接口；用于擦除指定空间的数据； @param dst 待擦除目标地址；该地址始终是后续 FlashDev.c 文件中描述的 PageSize 对齐的地址； @param length 待擦除数据长度；该长度始终是后续 FlashDev.c 文件中描述的 PageSize 的倍数； @return 返回 0 表示成功，否则失败；
int flashChipErase()	@brief Flash 整片擦除接口； @return 返回 0 表示成功，否则失败；
int flashChecksum(char*dst, int length, int checksum)	@brief Flash 快速校验接口 @param dst 待校验目标地址 @param length 待校验数据长度 @param checksum 烧写镜像中的 checksum 值 @return 返回 0 表示成功，否则失败；

3. 调试算法逻辑

main.c 文件是算法工程的主控逻辑，main 函数完成此功能。

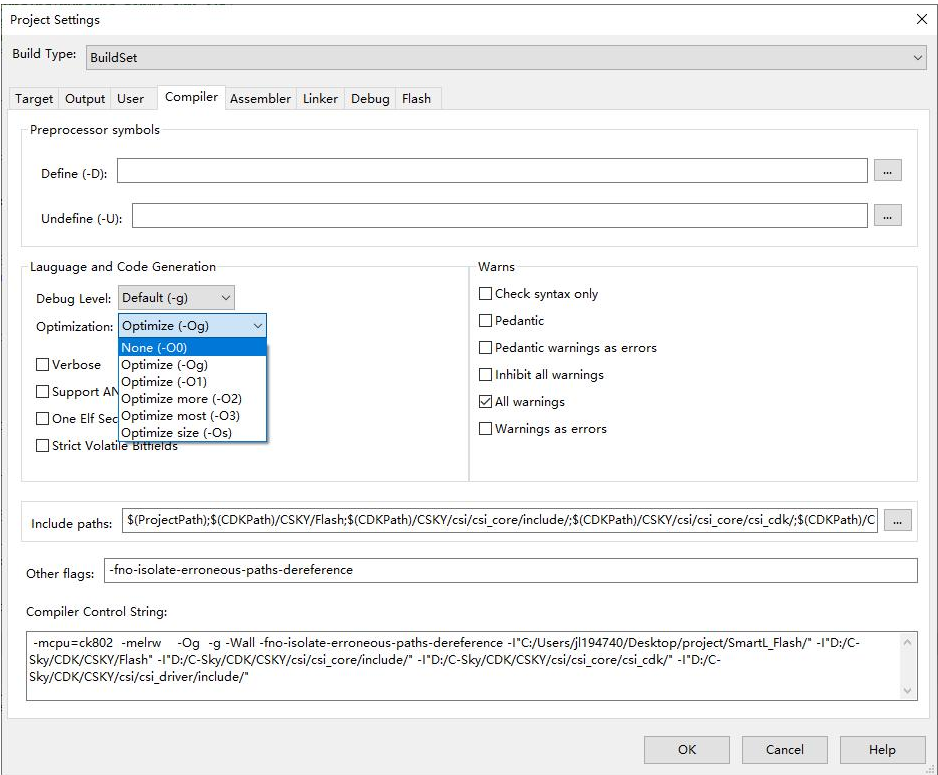

```

int main() {
    // call nor flash drivers to program
    g_error = flashInit();
    // for debug flash driver
    flashTest();
    do {
        __bkpt_label();
        switch (g_func) {
            case 0:
                g_error = flashID(&g_flashID);
                break;
            case 1:
                g_error = flashProgram((char *) g_dstAddress, (char *) g_rwBuffer,
                                        g_length);
                break;
            case 2:
                g_error = flashRead((char *) g_rwBuffer, (char *) g_dstAddress,
                                    g_length);
                break;
            case 3:
                g_error = flashErase((char *) g_dstAddress, g_length);
                break;
            case 4:
                g_error = flashChipErase();
                break;
            case 5:
                g_error = flashUnInit();
                break;
            case 6:
                g_error = flashChecksum((char*)g_dstAddress, g_length,
g_checksum);
                break;
            default:
                break;
        }
    } while (1);
}

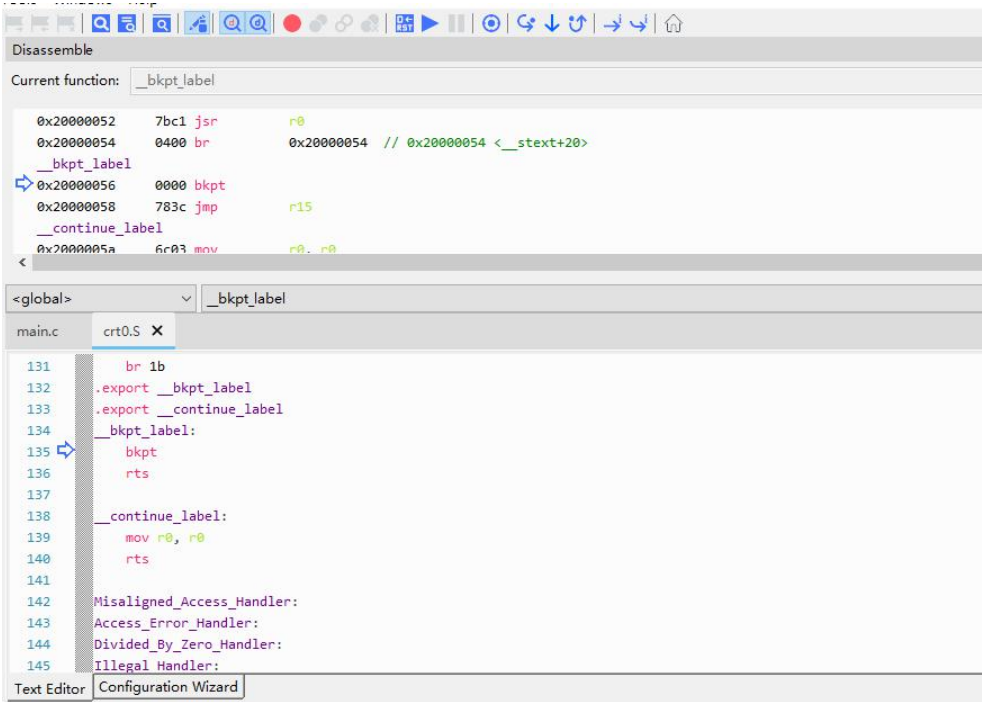
```

CDK 通过下载此算法工程到芯片 RAM 的 g_rwBuffer[] 数组中，然后通过控制 g_func, g_dstAddress, g_length, g_checksum 这些全局变量，实现对特定 Flash 器件的编程功能。这里需要调试算法文件，就是模拟整个烧写的过程，将每个函数调试通过即可。

为了能够更方便的调试算法文件, 右击算法工程, 在 Compiler 配置中, 修改 Optimization 修改为 `-O0` , 然后再次编译算法工程。



双击工程视图的算法工程节点, 设置为 active project, 然后点击调试按钮, 启动调试, 运行到 `main` 函数以后, 全速运行, 发现算法文件停止在 `__bkpt_label` 函数处。



然后，我们这里根据调试需求，设置相应的全局变量，进行验证。

例如，这里我们调试 flashProgram 接口的正确性，首先在 CDK Watch 界面将 g_func 设置为 1，g_dstAddress 设置为要进行编程的 Flash 区域的地址，g_length 设置为要烧写的长度，g_rwBuffer 设置为具体的烧写数据内容，一般来说，对于调试，g_length 数据量不需要太长。

Frame Info			
Expression	Value	Type	Location
g_func	0x00000001	int	0x20000164
g_dstAddress	0x00000000	int	0x20000170
g_length	0x00000008	int	0x2000016c
g_rwBuffer	[1024]	int [1024]	0x20000178
0	0x00000001	int	0x20000178
1	0x00000002	int	0x2000017c
2	0x00000003	int	0x20000180
3	0x00000004	int	0x20000184
4	0x00000005	int	0x20000188
5	0x00000006	int	0x2000018c
6	0x00000007	int	0x20000190
7	0x00000008	int	0x20000194
8	0x00000000	int	0x20000198
9	0x00000000	int	0x2000019c
10	0x00000000	int	0x200001a0
11	0x00000000	int	0x200001a4

Add Expression

Locals Watches Call Stack Memory

这里我们设置向 Flash 区域 0x0—0x8 区域烧写 0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8 特征数据，来验证算法文件的正确性。CDK 中设置好这些变量以后，把 PC 的值设置为当前 lr (r15) 的值，然后点击单步运行，

Register	
Name	Value
r14	0x20001278
r15	0x20000092
r16	0x00000000
r17	0x00000000
r18	0x00000000
r19	0x00000000
r20	0x00000000
r21	0x00000000
r22	0x00000000
r23	0x00000000
r24	0x00000000
r25	0x00000000
r26	0x00000000
r27	0x00000000
r28	0x00000000
r29	0x00000000
r30	0x00000000
r31	0x00000000
pc	0x20000092
psr	0x80000100
epsr	0x00000000
epc	0x00000000
gpr	

点击单步运行，程序运行在 main.c 文件的 flashProgram 入口，然后通过常规的调试手段，进入该函数内部，运行函数完成以后，验证是否正确烧写。

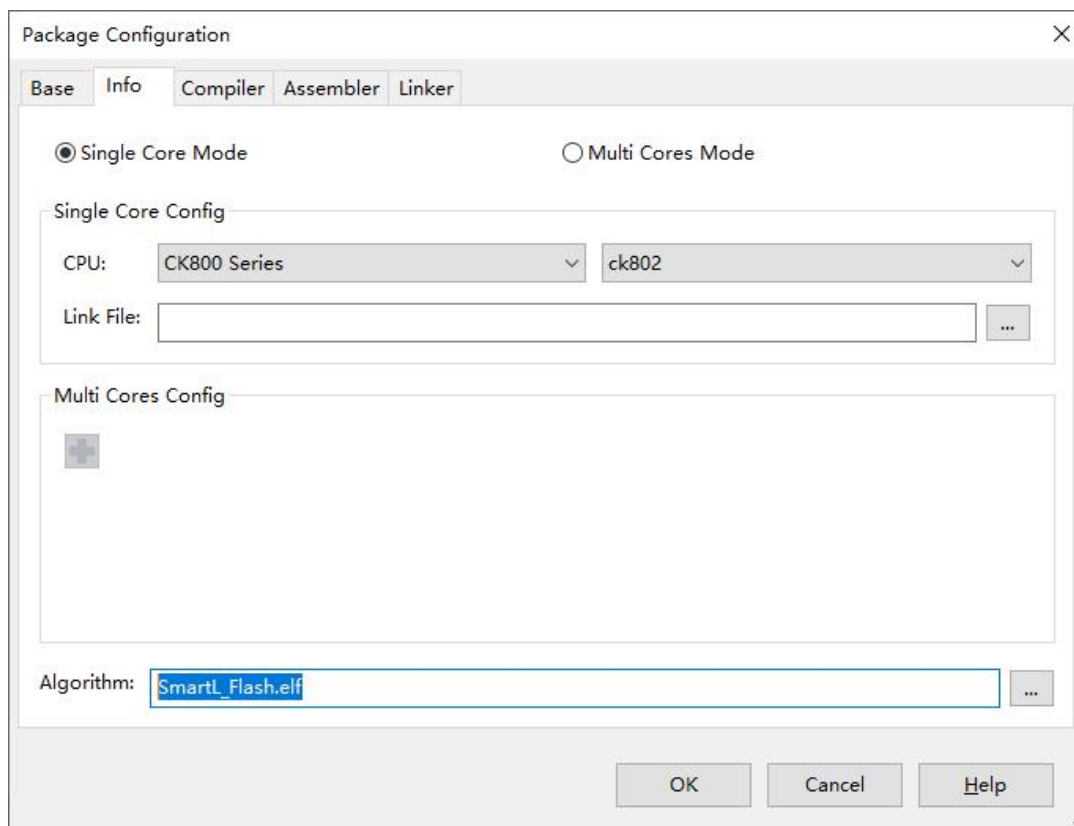
按照上述手段，分别验证每个函数的正确性即可。

4. 配置算法文件到 SDK 工程

当完成整个算法文件调试以后，为了更好的提升算法文件的性能，这里我们将算法工程的 Compiler 编译选项由 -O0 修改为 -O2，然后重新编译，编译完成以后，将 SmartL_Flash 工程根路径下 Obj/目录中生成的 SmartL_Flash.elf 算法文件 copy 到 my_solution 工程使用的 SmartL_Chip 组件的目录中去，CDK 工程视图右击菜单 Open Containing Folder 选项，可以直接打开工程、组件所在的目录。

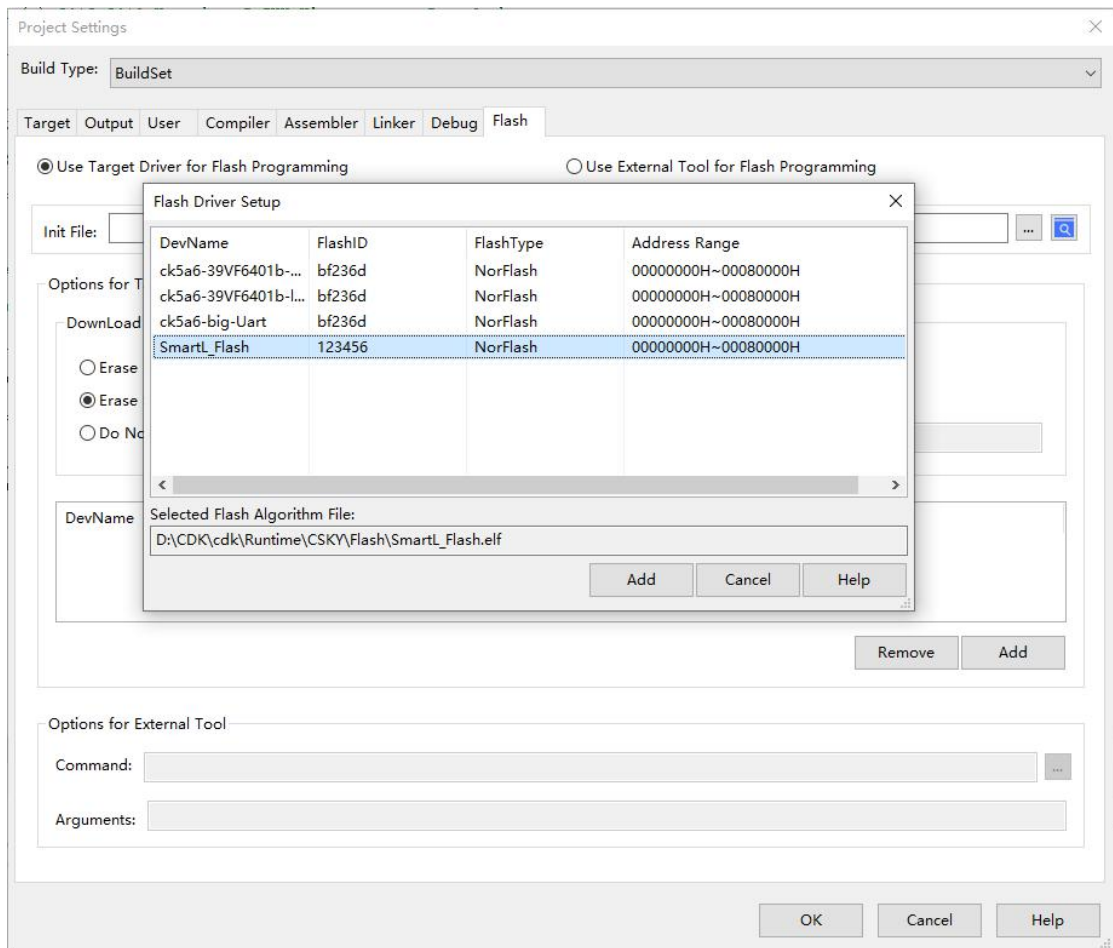
然后在工程视图 SmartL_Chip 组件节点的配置窗口中，配置 SmartL_Flash.elf 作为算法文件。

【情况1】



对于没有使用芯片组件或者使用旧版本的组件的情况，则需要将 SmartL_Flash.elf 拷贝到 my_solution 工程根路径下（与 my_solution.ckdproj 文件同目录），然后在工程配置的 Flash Tab 的算法选择中，选择该组件即可。

【情况 2】

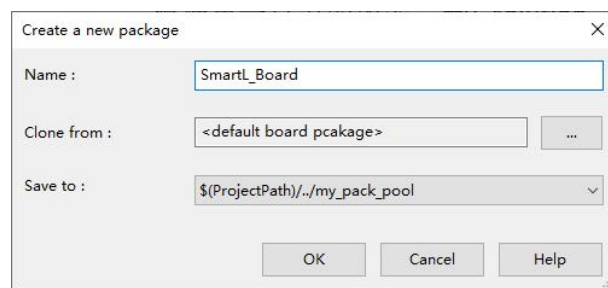


I 如何修改初始 SDK 工程

根据第一章的内容，my_solution 目前是根据初始组件化工程而来，这里需要根据 SmartL 平台的内容，对当前 SDK 工程进行必要的更新和修改。

1. 新增一个开发板组件

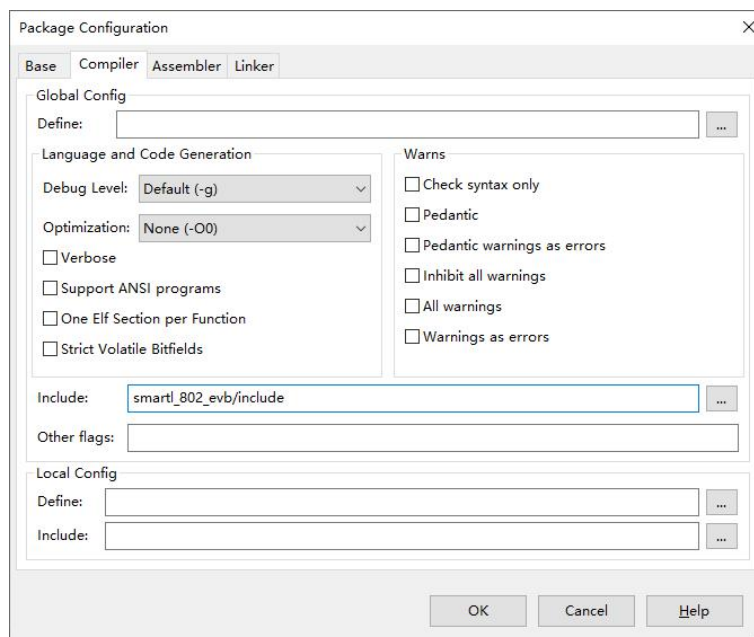
在 SmartL_SDK 节点下新增一个 SmartL_Board 的开发板组件。



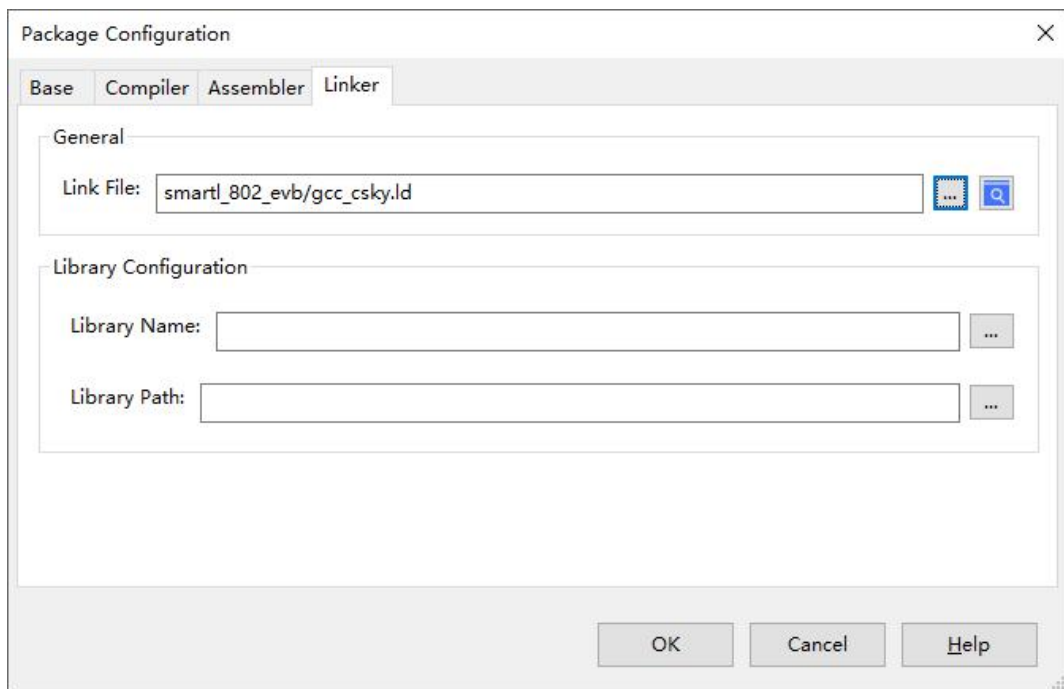
然后右击工程视图的 SmartL_Board 节点，点击 Add Source Folder 选项，选中已经存在的代码目录，点击 OK，此时将开发板组件的代码完全导入到新建的组件中了。

右击开发板组件 SmartL_Board，在配置选项中，完成组件的编译配置：

1、在 Compiler 选项 Global Config 中，配置开发板组件需要提供给外部的头文件搜索路径；然后配置 Debug Level 也要配置包含调试信息的选项。



2、在 Linker 选项卡中配置使用的链接描述文件。



【备注】

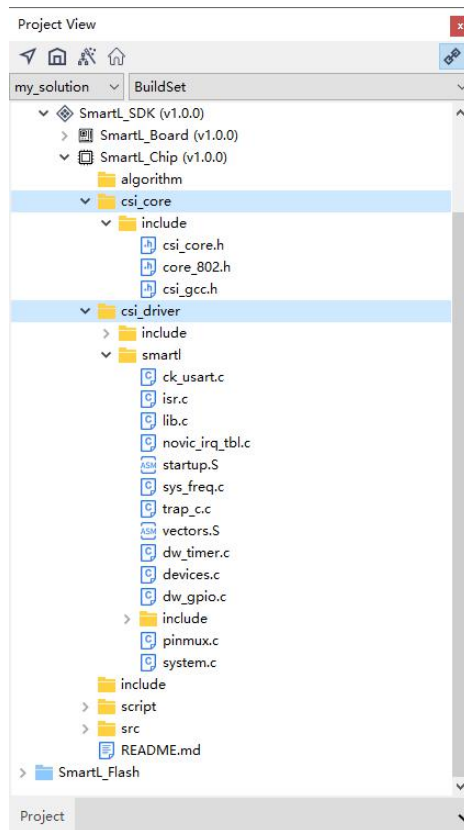
关于链接描述文件，solution、board、chip 类型的组件的配置窗口的 Linker 选项卡中都可以配置，CDK 会按照 solution > board > chip 的优先级，选择合适的链接文件使用。

在本例中，CDK 会使用 board 组件配置的 gcc_csky.ld 文件作为链接描述文件。

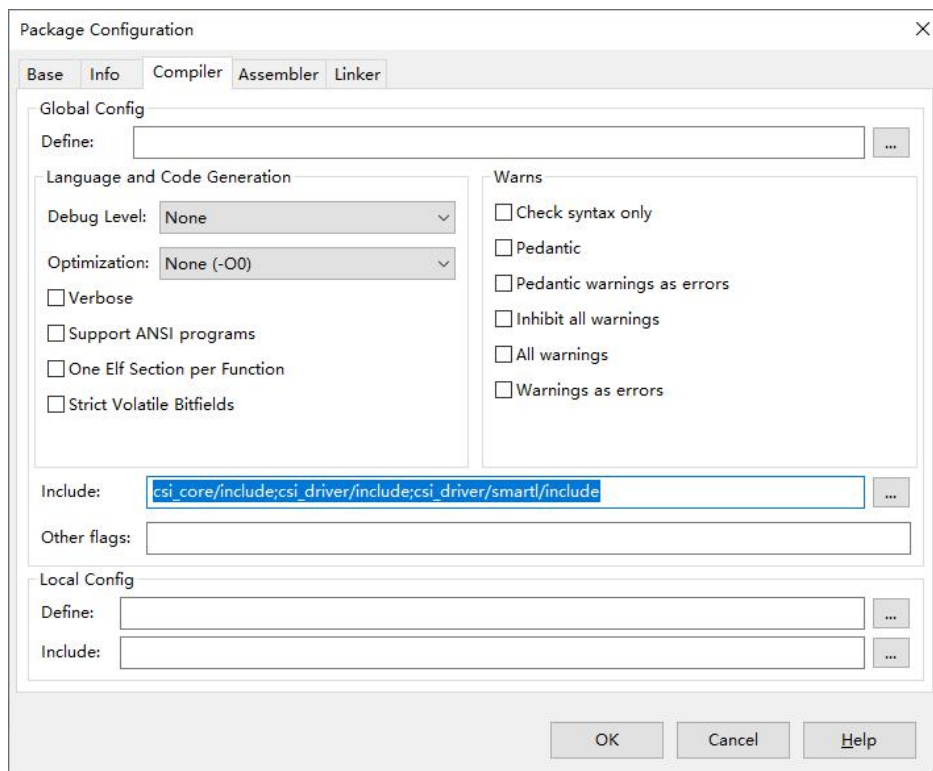
2. 更新当前的芯片组件

芯片组件的更新，主要包含芯片外设驱动的代码的编写，如果没有现成代码，需要逐个添加，编写代码，如果存在驱动代码，可以右击 SmartL_Chip 组件节点，选择 Add Source Folder，将芯片组件需要包含的内容添加到芯片组件中。

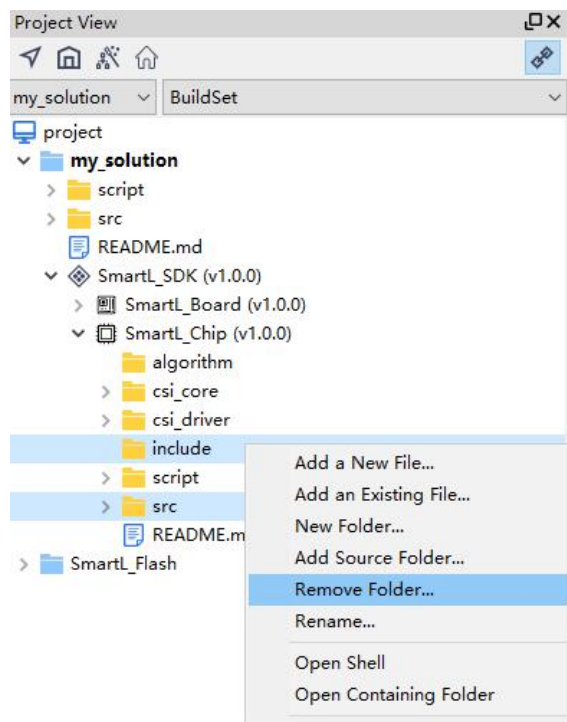
本例中，将 SmartL_Chip 组件使用的 csi_core 接口和驱动代码，通过 Add Source Folder 的方式添加到芯片组件中。



完成源代码的添加以后，需要将使用的头文件路径配置到芯片组件配置对话框中的 Compiler 选项卡中。



由于我们添加的芯片组件代码中，包含了 CPU 的初始化和启动逻辑，所以，我们把之前 SmartL_Chip 组件包含的默认提供的芯片初始化的代码逻辑删除。

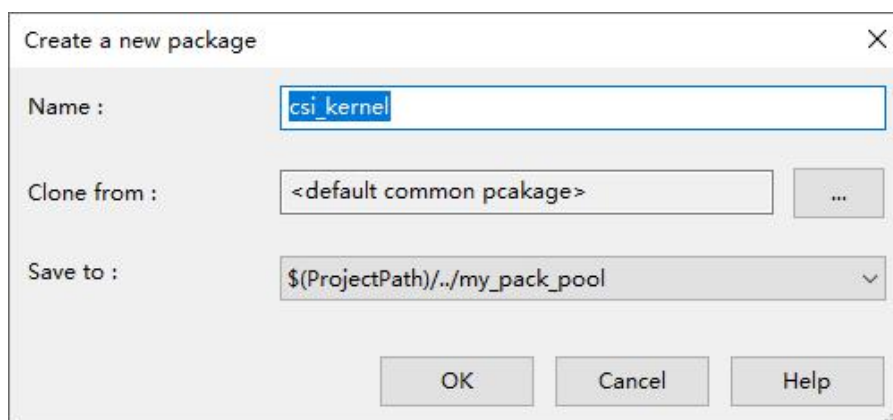


SmartL 的 SDK 的硬件部分已经完成。

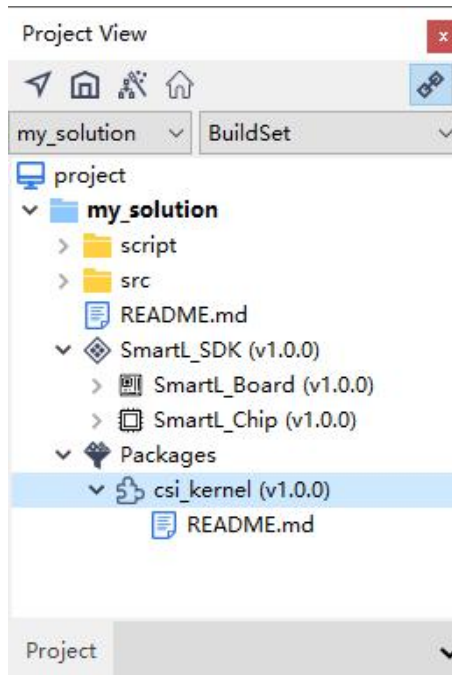
3. 增加两个硬件无关的 common 组件

此外，我们还有两个代码模块需要添加，一个是 RTOS 的 kernel 部分，另外一个是一些工具函数。两部分的添加逻辑类似，这里介绍 RTOS 的 kernel 的添加。

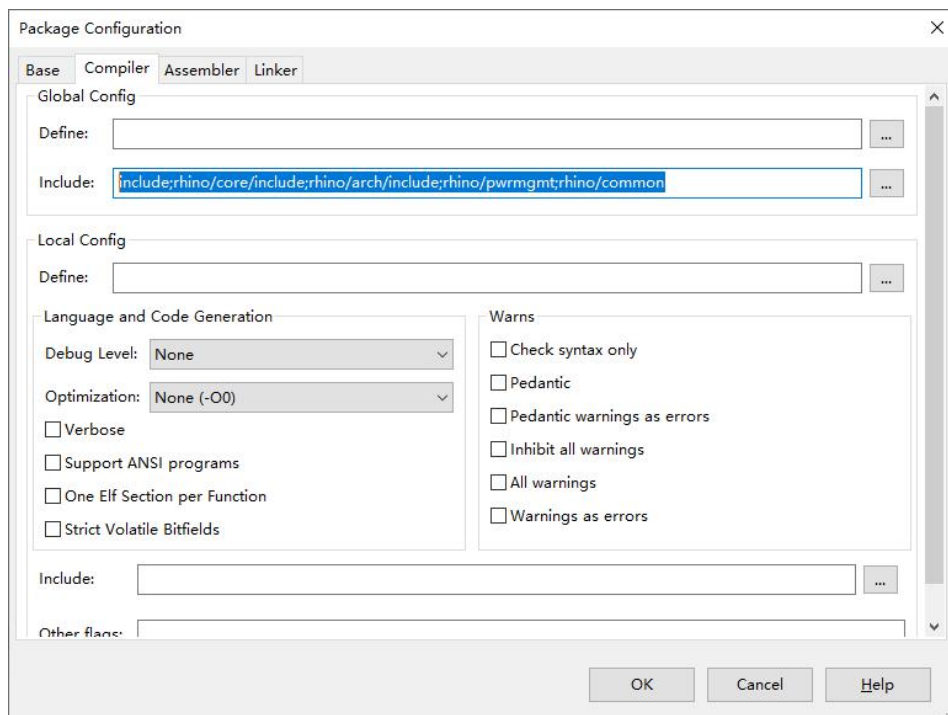
右击 my_solution 工程节点,选择 Create a Common Package 选项,选择 Clone from 为<default common package>, 然后输入组件名称, 点击 OK



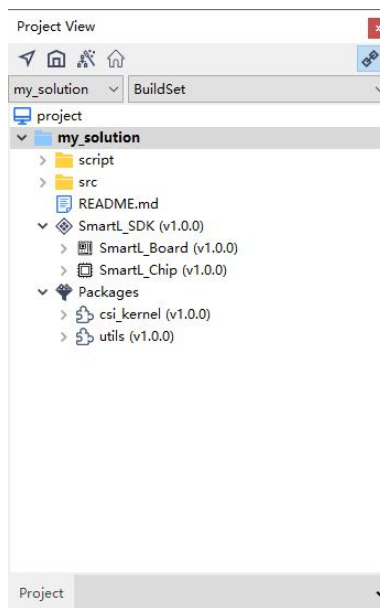
完成创建以后，工程视图会出现 Packages 节点，该节点表示直接被 solution 工程依赖的组件。该节点下面包含了刚刚创建出来的组件。



然后右击该节点，选择 Add Source Folder，将需要添加的代码加入到组件中。然后在组件配置中的 Compiler Tab 中配置需要提供的全局头文件搜索路径。



至此，工程依赖的组件的添加和修改工作完成。

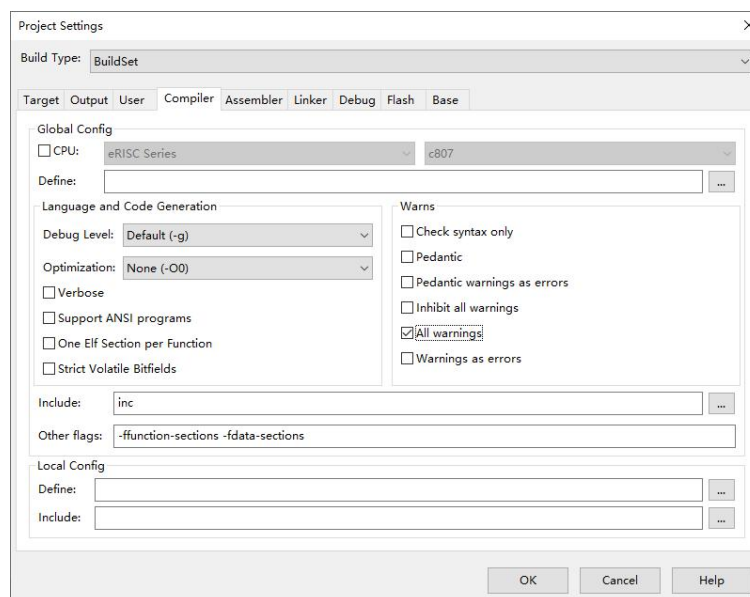


4. 更新当前 solution 工程

对于工程本身，也是一个 solution 类型的组件，这里我们添加一些 solution 组件需要的代码，由于这些代码包含了 main 函数，所以，我们需要将工程默认的 main 函数所在的文件删除。

右击工程，通过 New Folder 创建新的目录，通过 Add an Existing File 添加已经存在的某个源文件。

然后根据 solution 具体的编译需要，在工程节点的配置窗口中，对 Compiler、Linker 等选项卡中的内容进行配置。



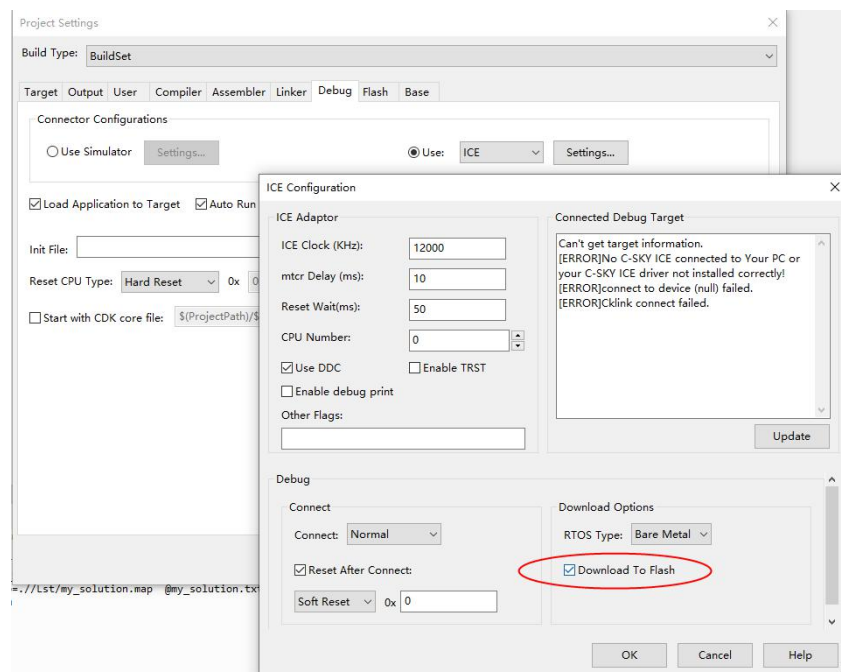
【备注】

如果想要了解这些选项卡中的每个字段的节点的含义，在界面显示该节点时，点击 Help 按钮，就会直接显示配置含义。

至此，完成了主要代码的编写和工程编译的配置工作。

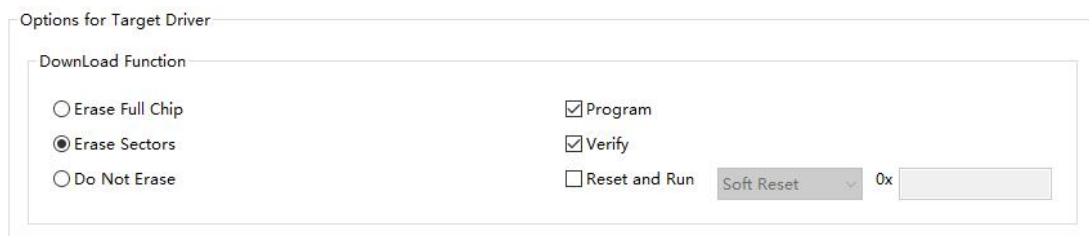
5. 调试和 Flash 配置

调试相关配置，SmartL 包含了 Flash 区间的调试，需要将工程配置窗口 Debug 选项卡中的 ICE Settings 子窗口中的 Download to Flash 勾选。



这样，工程就可以在 Flash 区间进行调试了。

对于 Flash 的配置，需要在工程配置窗口的 Flash 选项卡中进行配置，其中 Options for Target Deriver 区域需要配置 Flash 烧写的操作。



这里 SmartL 使用 Erase Sectors 的擦除方式，并在烧写完成进行 Verify 验证。

I 如何将制作完成的 SDK 发布给其他开发者使用

完成 SmartL SDK 制作以后，需要将该工程发布给开发者使用，这里有两种发布方式：离线发布和在线发布。

1. 离线发布 SDK 工程

工程 clean 完成以后，打开工程所在目录，关闭 CDK，然后将工程目录以及章节一中创建的 Package Path 设置的目录，共同打包。

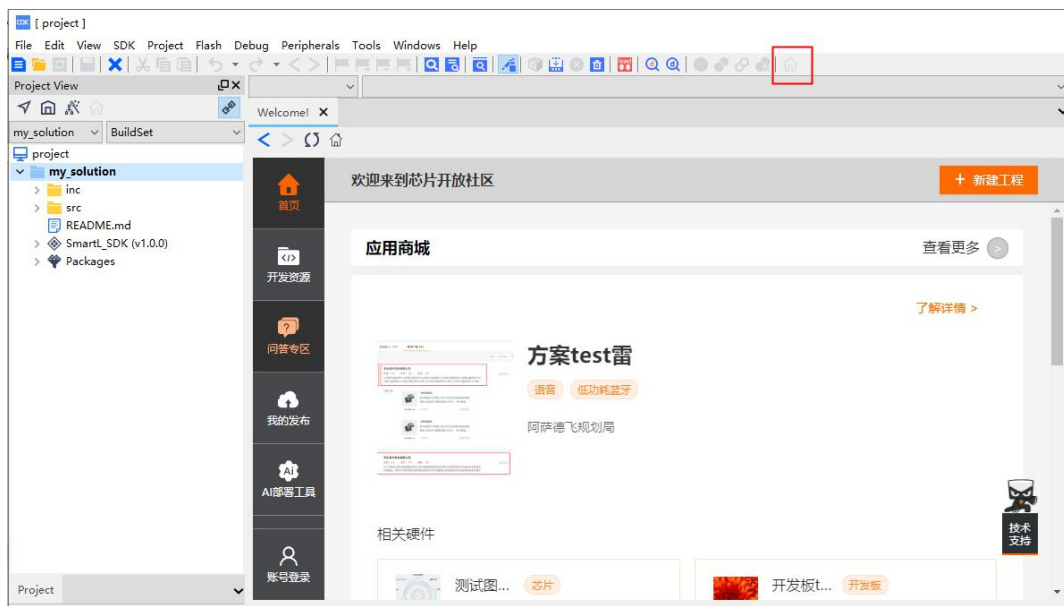
打包需要注意，在打包的目录结构中，工程目录与 Package Path 目录的相对路径关系，应该与 SDK 制作的时候保持一致。例如，当前例子中，SmartL 工程在桌面的 project/my_solution/ 目录下，而使用的 Package Path 目录是桌面的 project/my_pack_pool/ 目录，那么发布的时候，将桌面 project/my_solution/ 目录和桌面 project/my_pack_pool/ 目录共同拷贝到同一个目录 SmartLSDK 目录下，然后压缩该目录作为一个 SDK 即可。

2. 在线发布 SDK 工程

CDK 自动对接了芯片开放社区（Open Chip Community）网络平台（以下简称 OCC），能够帮助开发者将自己开发完成的 SDK，上传到 OCC 中，并选择对外发布与否。

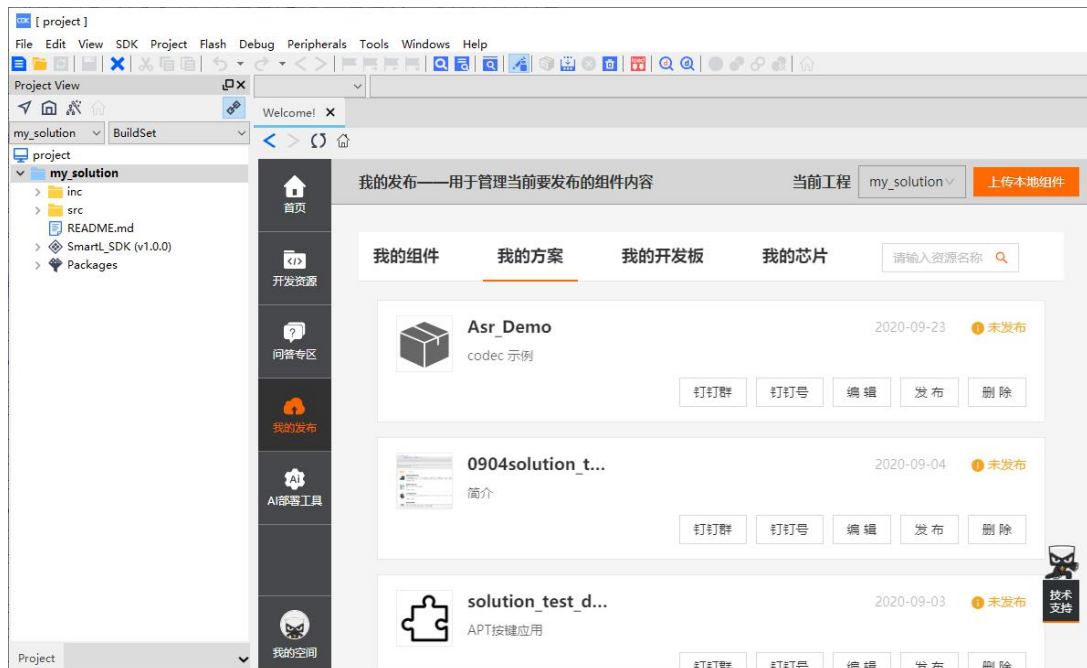
以 SmartL 为例，需要把当前平台使用的组件逐个发布到 OCC 平台中。

首先需要在 OCC 平台申请一个账号，并申请企业入驻资质，然后在 CDK 中打开 welcome 页面。



然后点击“账号登录”，在 CDK 中登录 OCC 账号。

选择“我的发布”，进入发布页面。



组件的发布中，需要先提交被依赖的组件，然后才能提交依赖的组件，以本工程为例，提交组件的顺序为：

首先是：csi_kernel、utils、SmartL_Board、SmartL_Chip，提交完成以后，才可以提交 SmartL_SDK，当这些组件全部提交完成，最后提交工程 my_solution

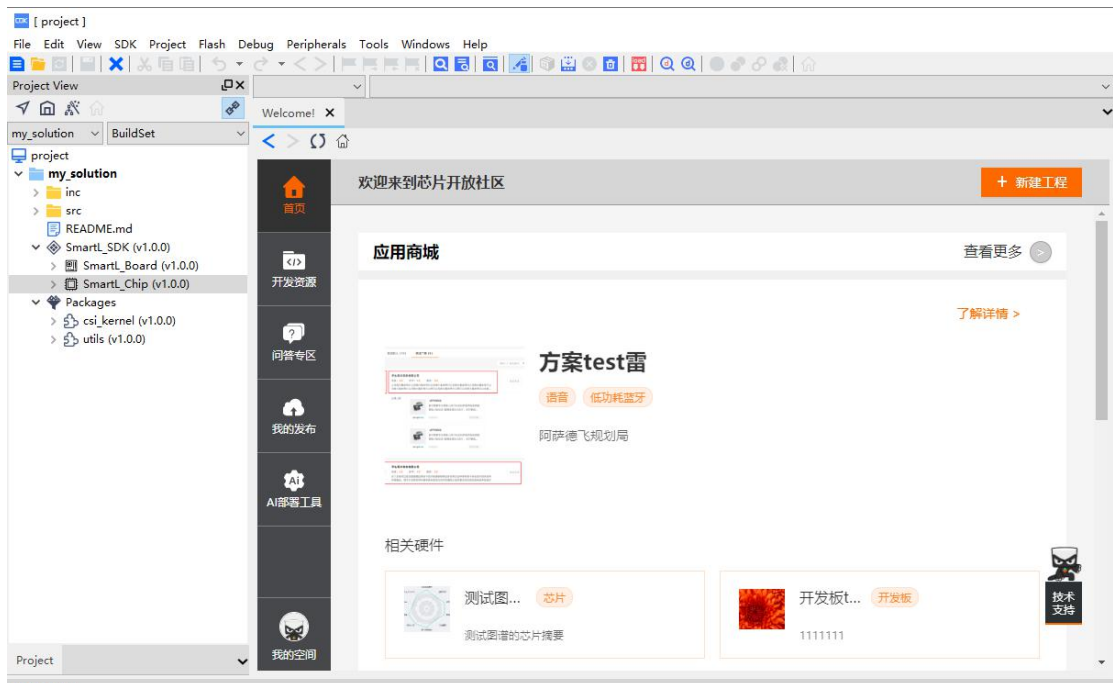
【备注】

1、除了 welcome 页面可以提供上传组件的入口,每个组件节点右击菜单栏中,Upload for xxx 也可以上传该组件。

2、目前 OCC 账号管理中,对于上传方案、开发板、芯片类型的组件,要求该账号必须为企业账号资质。

2.1 在线发布验证

当完成这些组件的上传以后,需要验证是否正确上传,在 welcome 页面的首页中,“新建工程”入口,会有上传成功的方案工程。



选择该工程,点击“创建工程”,然后输入工程名,并选择对应的版本号,完成工程的创建,创建完成以后,在 CDK 中该工程应该是可以正确编译和运行的。

I 常见问题以及解答

问题 1

Package Path 是否可以设置多个？

回答：可以。在 New SOC Project 对话框中，Package Path 只允许设置一个，工程设置完成以后，可以在工程节点右击，选择 Packages path setting，就可以设置多个 Package Path，并且可以设置每个 Package Path 的优先级，CDK 根据优先级去查找、使用组件。

问题 2

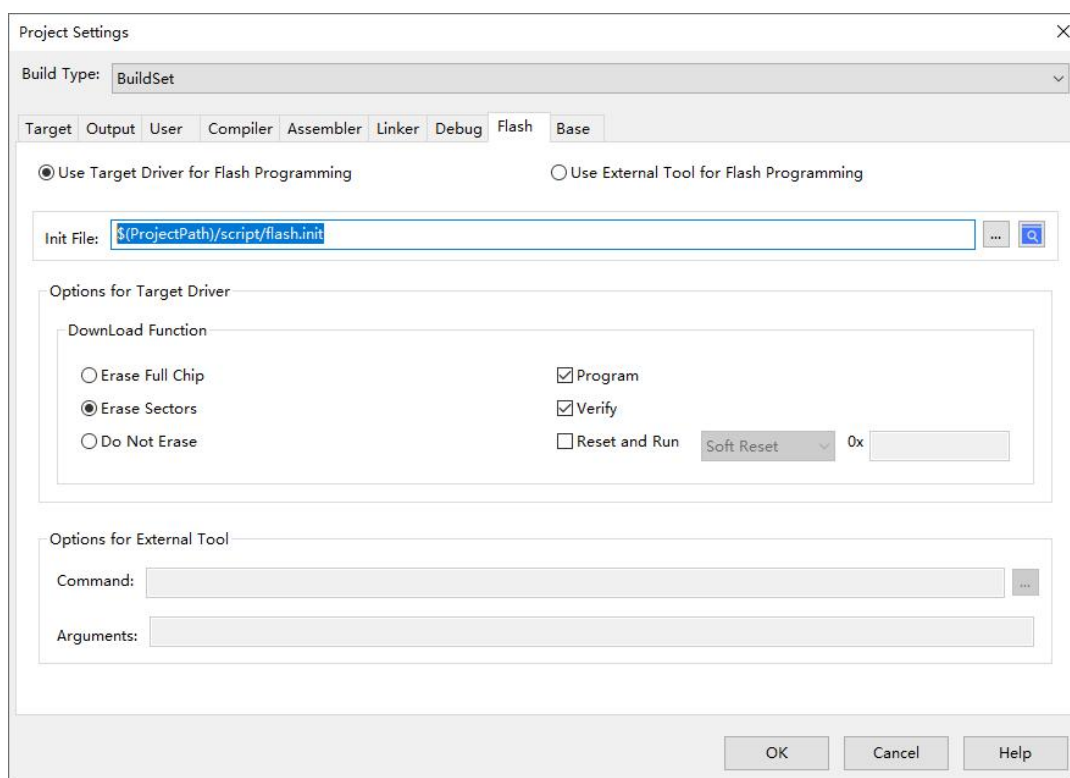
为何工程视图有代码文件，但是 CDK 没有编译该文件？

回答：确认该文件是否正确添加到 CDK 的编译规则中。目前，CDK 识别组件源文件的编译与否，是在组件对应的 package.yaml 文件中的 source_file 节点，正常情况下，CDK 不建议用户去手动修改该 yaml 文件，用户通过工程视图，Add Source Folder，Add an Existing File，Add a new File 等操作中，CDK 会自动更新该节点。如果遇到问题中提及的情况，可以直接检查一下该文件是否在 yaml 文件的 source_file 节点中，如果没有，可以根据 yaml 语法规则，手动添加进去，然后重新打开 CDK 即可。

问题 3

Flash 下载需要下载的内容不仅仅是工程 elf 文件本身，怎么将额外的内容下载到目标中？

回答：工程配置的 Debug、Flash 选项卡中，包含了 Init File 的配置：



该文件会在烧写工程 elf 文件之前被执行,文件规则可以参考用户文档,即上图中点击 Help 即可查看。

该脚本 download 命令支持 Flash 的区域的烧写,支持的文件类型有 elf、hex 和 bin 三种:

【elf 格式和 hex 格式】

download <format> [verify=yes/no] <filename>

<format>: 格式说明,根据下载的文件类型,这里填写 elf 或者 ihex

[verify=yes/no]: 选择是否对 download 的文件进行 verify 操作;可选,如果不写,则默认使用工程配置中的 Flash 配置中的 Verify 选项的值;

<filename>: 文件名称,支持\$(ProjectPath)表示工程根路径。

【bin 格式】

download bin [verify=yes/no] address=hex number <filename>

[verify=yes/no]: 选择是否对 download 的文件进行 verify 操作;可选,如果不写,则默认使用工程配置中的 Flash 配置中的 Verify 选项的值;

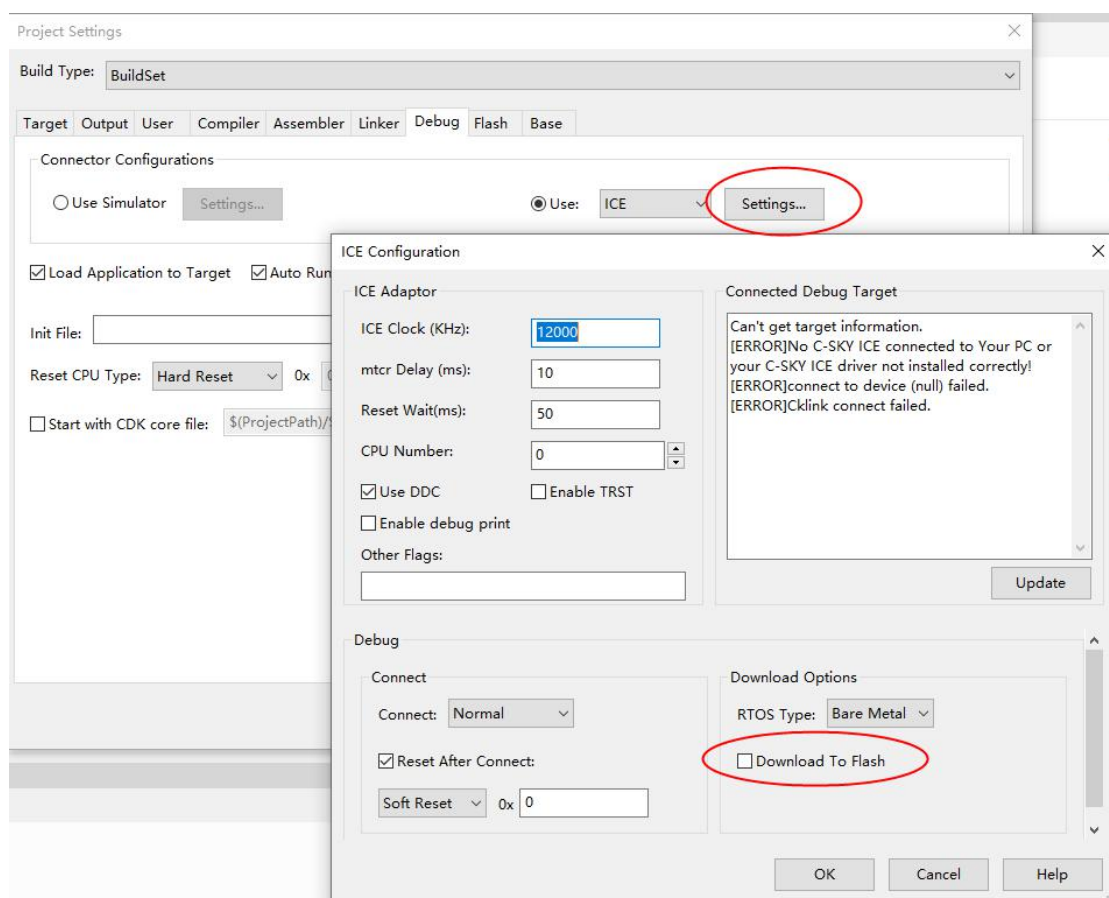
address: 设置 bin 文件烧录的起始地址,必须为 16 进制数据;

<filename>: 文件名称,支持\$(ProjectPath)表示工程根路径。

问题 4

如何支持 Flash 区域的调试，如何自动的设置硬断点？

回答：在工程配置的 Debug 选项卡中的 ICE Settings 子选项卡中，勾选了 Download to Flash，即开启了 Flash 区域调试的功能，同时，CDK 中设置的断点默认情况下都是硬断点。



问题 5

上传组件提示“组件名已经其他账户 xxx”的错误提示？

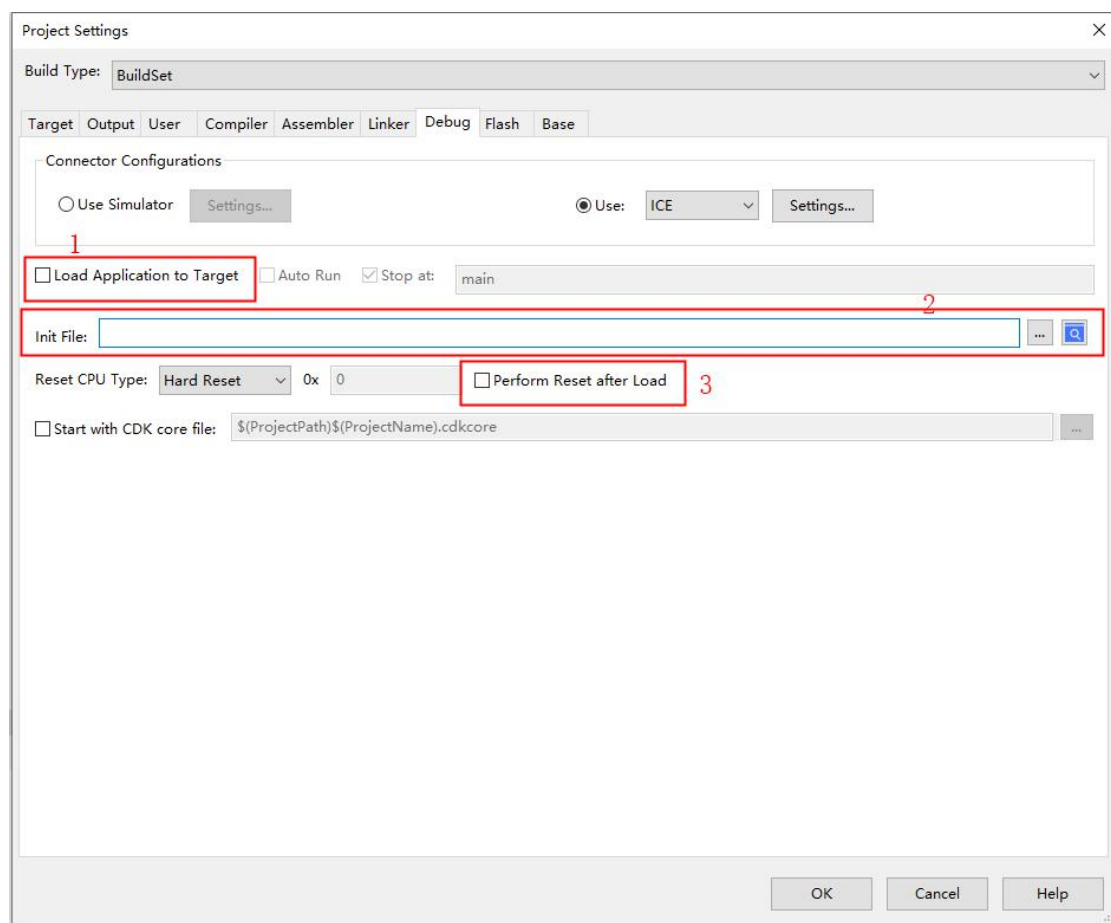
回答：组件名是稀缺的，如果一个组件名被某个账号使用，那么其他账户将无权限使用该组件名。解决方式：修改本地组件名，重新上传。

问题 6

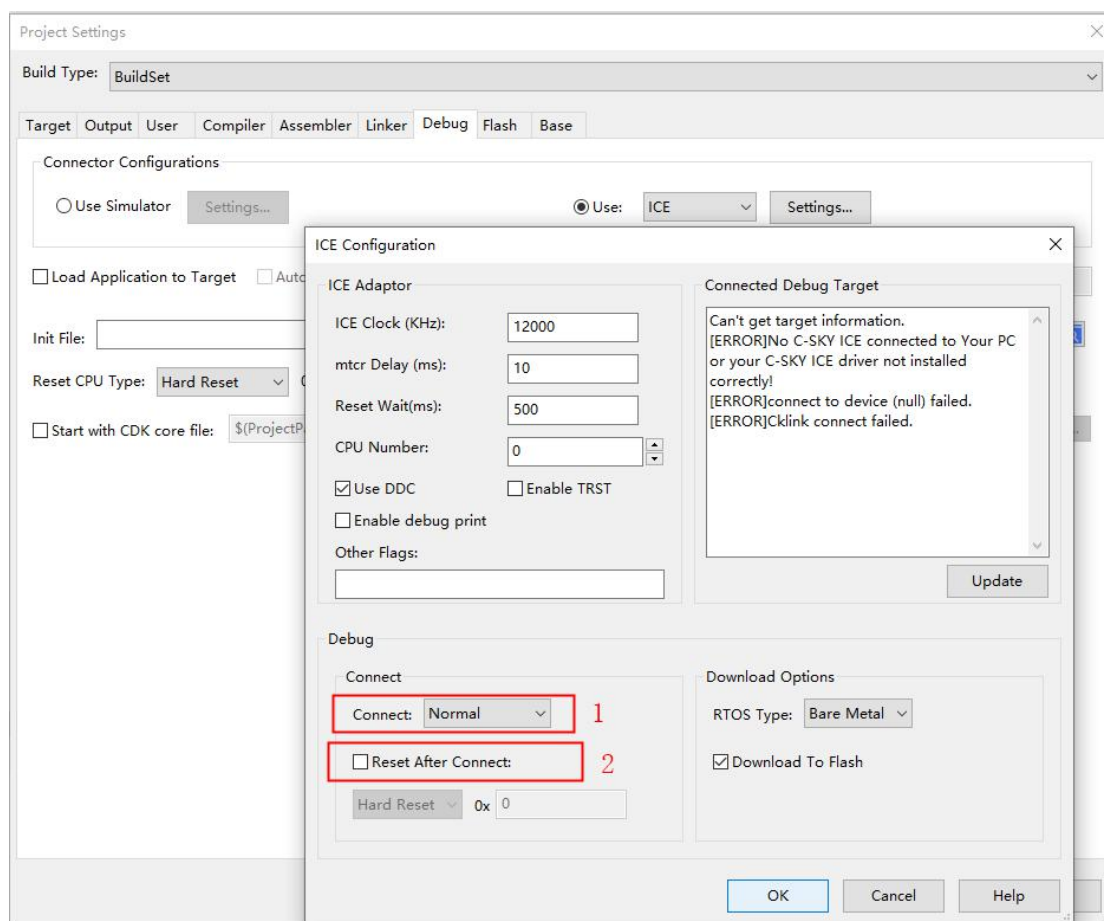
组件工程在调试目标中运行，并在终端输出运行错误信息以后，希望能够通过 CDK attach 到调试目标，直接查看出现错误的现场，应该如何操作？

回答：按照如下操作方式进行：

1、工程视图打开出现问题的组件工程的工程配置窗口，在 Debug 选项卡中，将 Load Application to Target 取消勾选，然后 Init File 不要勾选，最后，Perform Reset after Load 也不要勾选；



2、打开 ICE 或 Remote ICE 的 Settings 窗口，确保 Connect 方式为 Normal，并且不要勾选 Reset after connect；



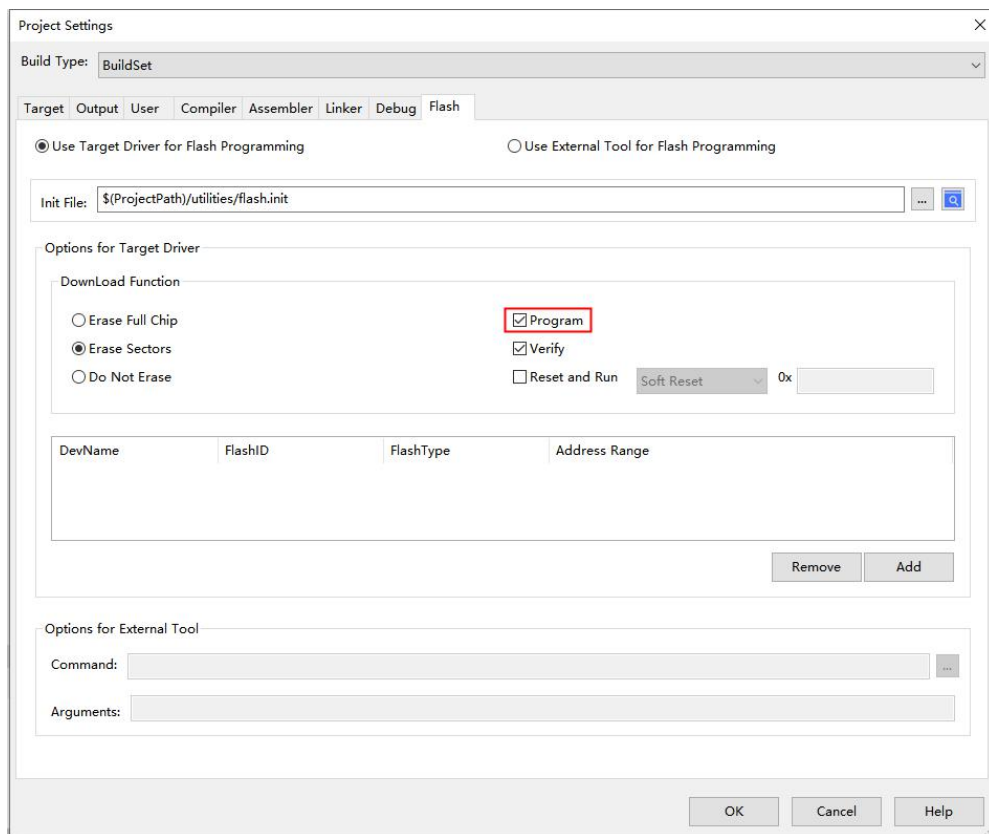
3、点击 OK，然后点击启动调试按钮，就可以直接 attach 到调试目标，在 CDK 的调试界面上查看各种需要的现场了。

问题 7

遇到烧写显示成功，但是程序没有正常启动的情况，如何去检查错误？

回答：按照如下的检查步骤，逐个检查：

1、查看方案工程配置窗口的 Flash 选项卡中的配置是否正确设置。



2、确认 Program 选项是否勾选，只有勾选了 Program，点击下载或调试按钮，才会进行 Flash 的烧写操作。

3、确认 Flash 算法文件是否选择正确，具体配置规则，可以参考【[2.4 配置算法文件到 SDK 工程](#)】章节的配置方式检查。

4、如果前面检查确认无误，需要按照【问题 6】描述的方法，attach 到烧写显示成功以后的嵌入式设备中，然后将预期的 Flash 的区域的数据 dump 出来，与本机的镜像内容进行 diff，查看具体的烧写错误的内容区域。

如有任何其他问题，欢迎加入钉钉芯片开放社区群，进一步沟通。



平头哥芯片开放社区交流群
扫码关注获取更多信息



扫码注册平头哥 OCC 官网
观看各类蓝牙视频及课程



扫码关注芯片开放社区微信
可获取更多资讯信息



阿里云开发者“藏经阁”
海量免费电子书下载