



Hi3861V100 / Hi3861LV100 第三方软件

移植指南

文档版本 02

发布日期 2020-06-28

版权所有 © 上海海思技术有限公司2020。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HISILICON、海思和其他海思商标均为海思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、服务或特性等应受海思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，海思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。

上海海思技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编： 518129

网址： <https://www.hisilicon.com/cn/>

客户服务邮箱： support@hisilicon.com



前言

概述

本文档详细介绍了移植第三方软件到SDK中的构建过程中的操作指导，同时提供了常见的问题解答及故障处理方法。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3861	V100
Hi3861L	V100



读者对象

本文档主要适用于软件开发人员。软件人员必须具备以下经验和技能：



- 熟悉构建工具、语法（包括SCons、Makefile）
- 有构建环境搭建经验

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
 危险	表示如不可避免则将会导致死亡或严重伤害的具有高等级风险的危害。
 警告	表示如不可避免则可能导致死亡或严重伤害的具有中等级风险的危害。



符号	说明
 注意	表示如不可避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不可避免则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。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修改记录

文档版本	发布日期	修改说明
02	2020-06-28	• 更新“ 1.3.1 第三方软件独立构建 ”中 步骤4 的内容。
01	2020-04-30	第一次正式版本发布。 • 更新“ 1.2 SCons构建 ”中的移植步骤说明。 • 更新“ 1.3.1 第三方软件独立构建 ”中的移植步骤说明。
00B01	2020-01-15	第一次临时版本发布。



目 录

前言..... i

1 移植指引..... 1

 1.1 概述..... 1

 1.2 SCons 构建..... 1

 1.3 Make 构建..... 3

 1.3.1 第三方软件独立构建..... 3

2 常见问题..... 5



1 移植指引

- 1.1 概述
- 1.2 SCons构建
- 1.3 Make构建

1.1 概述

Hi3861V100与Hi3861LV100的SDK使用SCons作为构建工具，因此，推荐优先使用SCons进行第三方库移植，从而保持编译的完整性和连贯性。由于有些库使用Make作为构建工具，并且使用SCons替代移植代价较高，本文档将提供使用make构建的移植方法与思路，开发者可根据具体的开发情况，自由选择移植方案。

1.2 SCons 构建

📖 说明

推荐移植方法：使用SCons构建。

以移植“mbedtls-x.x”为例，移植的步骤如下：

- 步骤1** 将第三方软件目录放置于“third_party”目录下（例如：third_party/mbedtls-x.x）。
- 步骤2** 在“mbedtls-x.x”目录下，添加第一层SConscript。建议从SDK中包含的第三方软件目录中复制，例如：复制“third_party/cjson/SConscript”至“third_party/mbedtls-x.x”下。此SConscript脚本将按照配置查询源码目录，并生成库(.a)文件。
- 步骤3** 在“mbedtls-x.x/library”目录下，添加次级SConscript。建议从SDK中包含的第三方软件目录中复制次级目录中的SConscript，例如：复制“cjson/cjson_utils/SConscript”至“mbedtls-x.x/library/SConscript”下。当第三方软件有多个源码目录时，需要为每个源码目录复制一份次级SConscript，并执行**步骤5**。如果源码目录结构复杂，请参考**步骤4**。
- 步骤4** （此步骤要求用户熟悉SCons脚本的编写）如果源码目录层次较深或目录较多，需要用户独立编写SConscript，以达到编译的目的。为了降低移植难度，建议用户将所有源码及源码目录放置在新建的“src”中，这样可以不修改第一级的SConscript，只需要小幅度修改次级SConscript，使其使用遍历的方式访问每个源码目录，并编译所有的.c或.s文件。



以下示例代码是次级SConscript遍历所有目录并生成.o文件的脚本，供开发者参考。

```
#!/usr/bin/env python3
# coding=utf-8

import os
Import('env')

env = env.Clone()

src_path = []
for root, dirs, files in os.walk('.'):
    src_path.append(root)

objs = []
for src in src_path:
    objs += env.Object(Glob(os.path.join(src, '*.c')))
    objs += env.Object(Glob(os.path.join(src, '*.S')))

Return('objs')
```

步骤5 修改配置文件“build/script/common_env.py”。

1. （必选）添加新增的第三方软件名称到“compile_module”。

```
compile_module = ['drv', 'sys', 'os', 'wpa', 'lwip', 'at', 'mbedtls']
```

2. （必选）添加新增的第三方软件的路径到“module_dir”。填写相对于工程根目录的相对路径，此路径下包含第一层SConscript。

```
module_dir = {
    'drv': os.path.join('platform', 'drivers'),
    'mbedtls': os.path.join('third_party', 'mbedtls-x.x'), #填写相对于工程根目录的相对
    # 路径，此路径下包含第一层SConscript
    ...
}
```

3. （必选）添加新增的第三方软件的库文件名称与源码文件目录到“proj_lib_cfg”。添加格式是‘模块名’:{‘生成库文件名’:‘源码目录路径’}，源码目录路径填写相对于第一层SConscript的相对路径。源码目录路径中，包含次级SConscript。

```
proj_lib_cfg = {
    #os modules
    'os':{
        'res_cfg': [
            os.path.join('kernel', 'redirect')
        ]
    },
    #third parties and components
    'mbedtls':{ 'mbedtls': ['library'] }, #格式 '模块名':{ '生成库文件名': '源码目录路径' }, 源码目录路
    # 径填写相对于第一层SConscript的相对路径。源码目录路径中，包含次级SConscript。
    ...
}
```

4. （可选）添加需要在编译阶段指定的编译宏定义到“proj_environment['defines']”。如果没有，无需添加。

```
'defines':{
    'common':[ ('PRODUCT_CFG_SOFT_VER_STR', r'\"%s\""%product_soft_ver_str),
                ('PRODUCT_CFG_BUILD_DATE', r'\"%s\""%cur_date),
                ('PRODUCT_CFG_BUILD_TIME', r'\"%s\""%cur_time),
                ('PRODUCT_CFG_BUILD_TIME_YEAR', r'\"%s\""%cur_time_year),
                ('PRODUCT_CFG_BUILD_TIME_MONTH', r'\"%s\""%cur_time_month),
                ('PRODUCT_CFG_BUILD_TIME_DAY', r'\"%s\""%cur_time_day),
                'CYGPKG_POSIX_SIGNALS',
                '__ECOS__',
```



```
'_RTOS_',
'PRODUCT_CFG_HAVE_FEATURE_SYS_ERR_INFO',
'_LITEOS_',
'LIB_CONFIGURABLE',
'LOSCFG_SHELL',
'CONFIG_DRIVER_HI1131',
'HISI_CODE_CROP',
'LOSCFG_CACHE_STATICS',#This option is used to control whether Cache hit
ratio statistics are supported.
#'LOG_PRINT_SZ',#This option is used to control whether print on shell
'CUSTOM_AT_COMMAND',
'LOS_COMPILE_LDM'
],
'mbedtls':[], # 可留空列表
...
}
```

5. （可选）添加编译选项到“proj_environment['opts']”。如果没有，无需添加。
6. （可选）添加新增的第三方软件所引用的liteOS中的头文件到“proj_environment['liteos_inc_path']”。如果没有，无需添加。
7. （必选）添加新增的第三方软件所引用的非liteOS头文件到“proj_environment['common_inc_path']”，并注意格式('#')。

```
'common_inc_path':{
    'common':[
        os.path.join('#', 'include'),
        os.path.join('#', 'platform', 'include'),
        os.path.join('#', 'platform', 'system', 'include'),
        os.path.join('#', 'config'),
        os.path.join('#', 'config', 'nv'),
    ],
    'mbedtls':[os.path.join('#', 'third_party', 'mbedtls-x.x', 'include', 'mbedtls')],
    ...
}
```

----结束

1.3 Make 构建

📖 说明

如果第三方软件基于Make构建，并且移植代价较高，用户可参考本节内容进行移植。

1.3.1 第三方软件独立构建

使用第三方软件原有的makefile独立构建，不需要依赖SCons编译脚本，需要单独手动执行make命令编译第三方库文件，以移植“mbedtls-x.x”为例，步骤如下：

- 步骤1** 将第三方软件目录放置于“third_party”目录下（例如：“third_party/mbedtls-x.x”）。
- 步骤2** 在“mbedtls-x.x”目录下，执行make命令。
- 步骤3** 将使用make编译生成的库文件（例如“libmbedtls.a”）复制到路径“build/libs”下，或复制至app工程目录下，并修改app.json文件，进一步减少对SDK的依赖。构建系统自动按需链接“build/libs”下所有库文件。
- 步骤4** 如果SDK开源代码有依赖第三方库的头文件，需要在相应的模块中添加头文件搜索目录，修改配置文件“build/scripts/common_env.py”。



例如: "components/at"下的源码依赖头文件"third_party/mbedtls-x.x/include/mbedtls/certs.h", 则需要在 "build/scripts/common_env.py" 中增加 "at" 模块的头文件搜索路径。

```
...
'common_inc_path':{
    ...
    'at':[
        os.path.join('#', 'third_party', 'mbedtls-x.x', 'include', 'mbedtls'),
        ...
    ],
    ...
}
```

----结束



2 常见问题

- 使用SCons构建时，编译过程中，报“无法找到头文件”错误。
解决方法：将头文件所在路径添加到“build/script/common_env.py”的“proj_environment['common_inc_path']”中，详细操作请参见“[步骤5.7](#)”。
- 使用SCons构建时，编译过程中，没有生成新添加的第三方软件库文件。
解决方法：
 - 如果编译过程中没有编译到第三方软件的源码，请确定“[步骤5.1](#)”已经配置完整。
 - 如果编译过程中只编译了部分第三方软件的源码，请确定“[步骤5.3](#)”已经配置完整。
- 使用SCons构建时，新增第三方软件的源码目录下，只需要编译部分.c文件的情况。
提供三种思路：
 - 修改次级SConscript脚本，将遍历所有.c文件改为需要编译的.c文件（与makefile的编写类似）。以下示例代码，可供参考。

```
wps_srcs = ['wps.c']
utils_srcs = ['base64.c']
all_srcs = []
all_srcs.extend(wps_srcs)
all_srcs.extend(utils_srcs)
objs = env.Object(all_srcs)
Return('objs')
```
 - 将需要编译的.c文件单独存放到另外一个目录下，并将次级SConscript放到新目录中，同时修改“build/script/comm_env.py”中的源码路径。
 - 从目录中删除不需要的源码文件。
- 使用SCons构建时，新增第三方软件目录结构中，源码文件距离源码目录存在多层的情况。例如：example/app/good/src/example.c，第一层SConscript在“example/app”下，次级SConscript在“example/app/good/src”下。
解决方法：在“build/script/common_env.py”的“proj_lib_cfg”中添加源码路径(['good/src'])即可。