



Hi3861V100 / Hi3861LV100 SDK 开发环境搭建

用户指南

文档版本 04

发布日期 2020-07-03

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

本文档介绍海思Hi3861V100、Hi3861LV100芯片SDK开发环境（包括：SDK编译、应用程序的开发等），用于帮助用户在快速了解开发环境后编译出可执行文件进行二次开发。

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

本文档主要适用于以下工程师：



- 技术支持工程师
- 软件开发工程师

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
 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
04	2020-07-03	<ul style="list-style-type: none">在“2.2.2 Menuconfig配置”中更新“图2-2；表2-5中删除Harmony Settings菜单及其配置项和说明。在“2.3 编译SDK (Makefile)”中新增makefile编译结构、文件说明及新增组件makeifile配置说明。
03	2020-06-30	<ul style="list-style-type: none">在“2.1 SDK目录结构介绍”的表2-1中新增SDK根目录与说明。在“2.2.2 Menuconfig配置”中更新“图2-2；表2-5中新增Harmony Settings菜单及其配置项和说明。
02	2020-06-28	<ul style="list-style-type: none">在“2.2.2 Menuconfig配置”的表2-5中Security Settings菜单新增TEE HUKS demo support配置项。在“2.2.3 注意事项”中新增关于OTA Settings选择升级模式的建议。在“2.3 编译SDK (Makefile)”中更新•编译产测程序内容。新增“2.4 修改烧写镜像构成”章节。



文档版本	发布日期	修改说明
01	2020-04-30	<p>第一次正式版本发布。</p> <ul style="list-style-type: none">在“2.1 SDK目录结构介绍”的表2-1中新增目录Makefile、non_factory.mk、factory.mk的说明；更新图2-1。在“2.2.1 编译方法”的表2-2中新增参数factory的说明；更新表2-3。在“2.2.2 Menuconfig配置”中更新图2-2；更新表2-5。在“2.2.3 注意事项”中更新关于如果执行“./build.sh”提示无权限的说明。在“2.3 编译SDK (Makefile)”中新增编译厂测程序的说明；删除单独编译库文件的说明。更新“3.1 建立目录结构”的说明。在“3.2 开发代码”中新增关于编译my_demo进行代码调试的描述。新增“3.3 app_main接口”小节。在“4.2.2 编译配置”中更新步骤5。
00B08	2020-04-03	在“ 2.2.2 Menuconfig配置 ”的 表2-5 中新增OTA Settings菜单的说明。
00B07	2020-03-27	<ul style="list-style-type: none">更新“2.2 编译SDK (SCONS)”标题名称。新增“2.3 编译SDK (Makefile)”小节。
00B06	2020-03-06	<ul style="list-style-type: none">在“1.2.2 Python环境安装”中新增关于使用“python setup.py install”进行安装的描述，新增关于如果构建环境中包含多个python的说明。在“2.2.3 注意事项”中新增关于编译过程中报错找不到某个包的注意说明。
00B05	2020-02-27	<ul style="list-style-type: none">新增“表2-2”。更新“2.2.1 编译方法”指令运行脚本路径。更新“表2-3”中Hi3861_demo_ota_2.bin的说明。更新“表2-5”。更新“3.4.2 编写编译配置文件”的“步骤3”。新增“4 客制化SDK组件”。



文档版本	发布日期	修改说明
00B04	2020-02-12	<ul style="list-style-type: none">更新“1.2 搭建Linux开发环境”中关于交叉编译器的说明。在“1.2.1 交叉编译器安装”中更新步骤1关于编译器安装包的说明，更新步骤3中新增关于编译器版本号的说明。在“1.2.3 Scons安装”中更新步骤1的推荐Scons版本、目录运行命令；更新“图1-4”；新增Scons安装后的常见问题说明。在“2.1 SDK目录结构介绍”中表2-1新增boot目录的说明，更新图2-1。更新“2.2.1 编译方法”的表2-3。更新“2.2.2 Menuconfig配置”的表2-5。
00B03	2020-01-15	<ul style="list-style-type: none">更新“图1-1”。在“1.2.2 Python环境安装”中的步骤5中增加安装six、ecdsa的说明。更新“表2-1”中目录app、tools的说明。更新“表2-3”。在“2.2.2 Menuconfig配置”中新增Menuconfig操作说明、Menuconfig命令帮助栏说明、Menuconfig配置项说明。更新“表3-1”中SDK链接脚本的文件名、app/my_demo/SConscript和app/my_demo/src/SConscript的说明。在“表3-2”中新增LD_DIRS的说明。
00B02	2019-12-19	<ul style="list-style-type: none">在“1.2.2 Python环境安装”中增加关于安装pycryptodome的步骤说明。在“1.2.3 Scons安装”的步骤1中增加推荐scons版本是3.0.1+的说明。更新“2.2.2 Menuconfig配置”的说明。在“3.1 建立目录结构”中删除关于新建app/my_demo/nv目录的步骤说明。更新“3.2 开发代码”的说明。在“表3-1”中更新build/scripts/common_env.py的说明、增加app/my_demo/app.json的说明。更新“3.4.2 编写编译配置文件”的说明。
00B01	2019-11-15	第一次临时版本发布。



目录

前言.....	i
1 开发环境搭建.....	1
1.1 SDK 开发环境简介.....	1
1.2 搭建 Linux 开发环境.....	2
1.2.1 交叉编译器安装.....	2
1.2.2 Python 环境安装.....	3
1.2.3 Scons 安装.....	4
2 编译 SDK.....	6
2.1 SDK 目录结构介绍.....	6
2.2 编译 SDK (SCONS)	7
2.2.1 编译方法.....	7
2.2.2 Menuconfig 配置.....	9
2.2.3 注意事项.....	11
2.3 编译 SDK (Makefile)	12
2.4 修改烧写镜像构成.....	14
2.4.1 修改 Hi3861_xxx_allinone.bin 构成.....	14
2.4.2 修改 Hi3861_xxx_burn.bin 构成.....	16
2.4.3 注意事项.....	16
3 新建 APP.....	18
3.1 建立目录结构.....	18
3.2 开发代码.....	18
3.3 app_main 接口.....	18
3.4 编译配置.....	19
3.4.1 编译基础脚本文件说明.....	19
3.4.2 编写编译配置文件.....	19
4 客制化 SDK 组件.....	21
4.1 构建系统概述.....	21
4.2 新建组件.....	23
4.2.1 编译输出.....	23
4.2.2 编译配置.....	23
4.3 客制化 Menuconfig.....	26



1 开发环境搭建

1.1 SDK开发环境简介

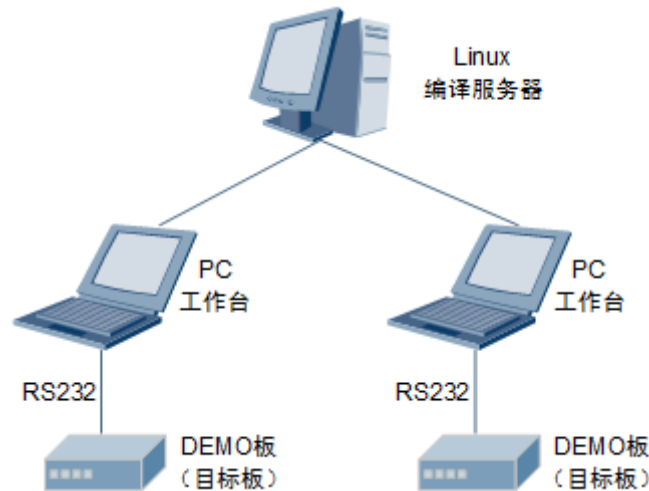
1.2 搭建Linux开发环境

1.1 SDK 开发环境简介

典型的SDK开发环境主要包括：

- Linux服务器
Linux服务器主要用于建立交叉编译环境，实现在Linux服务器上编译出可以在目标板上运行的可执行代码。
- 工作台
工作台主要用于目标板烧录和调试，通过串口与目标板连接，开发人员可以在工作台中烧录目标板的镜像、调试程序。工作台通常需要安装终端工具，用于登录Linux服务器和目标板，查看目标板的打印输出信息。工作台一般为Windows或Linux操作系统，在Windows或Linux工作台运行的终端工具通常有SecureCRT、Putty、miniCom等，这些软件需要从其官网下载。
- 目标板
本文的目标板以海思的DEMO板为例，DEMO板与工作台通过USB转串口连接。工作台将交叉编译出来的DEMO板镜像通过串口烧录到DEMO板。如图1-1所示。

图 1-1 SDK 开发环境



1.2 搭建 Linux 开发环境

linux系统推荐使用Ubuntu 16.04及以上版本，Shell使用bash，交叉编译器使用hcc_riscv32，编译工具链还包括Python、Scons等。

1.2.1 交叉编译器安装

步骤1 获取编译器安装包“hcc_riscv32.tar.gz”（安装包包含版本信息，文档使用简洁名称，实际名称以编译器发布版本为准）。

步骤2 编译器安装到系统目录下，并增加到环境变量中（需root/sudo权限执行）。

例如：将编译器解压缩到安装目录下，以使用“/toolchain”为例：

1. 复制压缩包到“/toolchain”目录下，解压缩“tar zxvf hcc_riscv32.tar.gz”。
2. 递归修改编译器安装目录权限：chmod -R 755 /toolchain。
3. 设置环境变量：vim /etc/profile，新增：export PATH=/toolchain/hcc_riscv32/bin:\$PATH。
4. 生效环境变量：source /etc/profile。

步骤3 Shell命令行中输入“riscv32-unknown-elf-gcc -v”，如果能正确显示编译器版本号，表明编译器安装成功。安装成功的结果如图1-2所示，图中编译器版本号可能有变动，需以编译器发布的实际版本为准。

图 1-2 编译器安装成功

```
wn-elf/7.3.0/lto-wrapper
Target: riscv32-unknown-elf
Configured with: /usr1/heterogeneous_compiler_codesign/build/hcc_riscv32/../../o
pen_source/hcc_riscv32_build_src/gcc-7.3.0/configure --build=x86_64-pc-linux-gnu
--host=x86_64-pc-linux-gnu --target=riscv32-unknown-elf --with-arch=rv32imc --w
ith-abi=ilp32 --disable-__cxa_atexit --disable-libgomp --disable-libmudflap --en
able-libssp --disable-libstdc++-pch --disable-nls --disable-shared --disable-thr
eads --disable-multilib --enable-poison-system-directories --enable-languages=c,
c++ --with-headers=/usr1/heterogeneous_compiler_codesign/build/hcc_riscv32/riscv
32_elf_build_dir/hcc_riscv32/riscv32-unknown-elf/include --with-gnu-as --with-gn
u-ld --with-newlib --with-pkgversion='Heterogeneous Compiler&Codesign V100R003C0
0SPC200B023' --with-build-sysroot=/usr1/heterogeneous_compiler_codesign/build/hc
c_riscv32/riscv32_elf_build_dir/hcc_riscv32/riscv32-unknown-elf --with-gmp=/usr1
/heterogeneous_compiler_codesign/build/hcc_riscv32/riscv32_elf_build_dir/obj/hos
t-libs/usr --with-mpfr=/usr1/heterogeneous_compiler_codesign/build/hcc_riscv32/r
iscv32_elf_build_dir/obj/host-libs/usr --with-mpc=/usr1/heterogeneous_compiler_c
odesign/build/hcc_riscv32/riscv32_elf_build_dir/obj/host-libs/usr --with-isl=/us
r1/heterogeneous_compiler_codesign/build/hcc_riscv32/riscv32_elf_build_dir/obj/h
ost-libs/usr --with-build-time-tools=/usr1/heterogeneous_compiler_codesign/build
/hcc_riscv32/riscv32_elf_build_dir/hcc_riscv32/riscv32-unknown-elf/bin --with-sy
stem-zlib
Thread model: single
gcc version 7.3.0 (Heterogeneous Compiler&Codesign V100R003C00SPC200B023)
```

----结束

1.2.2 Python 环境安装

- 步骤1** 打开Linux终端，输入命令“python3 -V”，查看Python版本号，推荐python3.7以上版本。
- 步骤2** 如果Python版本太低，请使用命令“sudo apt-get update”更新系统到最新，或通过命令“sudo apt-get install python3 -y”安装Python3（需root/sudo权限安装），安装后再次确认Python版本。
- 如果仍不能满足版本要求，请从“<https://www.python.org/downloads/source/>”下载对应版本源码包，下载与安装方法请阅读 <https://wiki.python.org/moin/BeginnersGuide/Download> 和源码包内README内容。
- 步骤3** 安装Python包管理工具，运行命令“sudo apt-get install python3-setuptools python3-pip -y”（需root/sudo权限安装）。
- 步骤4** 安装Kconfiglib 13.2.0+，使用命令“sudo pip3 install kconfiglib”（需root/sudo权限安装），或从“<https://pypi.org/project/kconfiglib>”下载.whl文件（例如：kconfiglib-13.2.0-py2.py3-none-any.whl）后，使用“pip3 install kconfiglib-xxx.whl”进行安装（需root/sudo权限安装），或者下载源码包到本地并解压，使用“python setup.py install”进行安装（需root/sudo权限安装）。安装完成界面如图 1-3所示。

图 1-3 安装 Kconfiglib 组件包完成示例

```
ll kconfiglib-13.2.0-py2.py3-none-any.whl
Processing ./kconfiglib-13.2.0-py2.py3-none-any.whl
Installing collected packages: kconfiglib
Successfully installed kconfiglib-13.2.0
tools/menuconfig#
```



步骤5 安装升级文件签名依赖的Python组件包：

- 安装pycryptodome
从“<https://pypi.org/project/pycryptodome/#files>”下载.whl文件（例如：pycryptodome-3.7.3-cp37-cp37m-manylinux1_x86_64.whl）后，使用“pip3 install pycryptodome-xxx.whl”进行安装（需root/sudo权限安装），或者下载源码包到本地并解压，使用“python setup.py install”进行安装（需root/sudo权限安装）。安装完成后界面会提示“Successfully intalled pycryptodome-3.7.3”。
- 安装six
从“<https://pypi.org/project/six/>”下载.whl文件（例如：six-1.12.0-py2.py3-none-any.whl）后，使用“pip3 install six-xxx.whl”进行安装（需root/sudo权限安装），或者下载源码包到本地并解压，使用“python setup.py install”进行安装（需root/sudo权限安装）。安装完成后界面会提示"Successfully installed six-xxx"。
- 安装ecdsa
从“<https://pypi.org/project/ecdsa/>”下载.whl文件（ecdsa-0.14.1-py2.py3-none-any.whl）后，使用“pip3 install ecdsa-0.14.1-py2.py3-none-any.whl”进行安装（需root/sudo权限安装），或者下载源码包到本地并解压，使用“python setup.py install”进行安装（需root/sudo权限安装）。安装完成后界面会提示"Successfully installed ecdsa-0.14.1"。
说明：安装ecdsa依赖six。需要先安装six，再安装ecdsa。

----结束

说明

如果构建环境中包含多个python，特别是多个同版本的python，而用户无法辨认正在使用的是其中的哪个版本，此情况下，在安装python组件包时，推荐使用组件包源码进行安装。

1.2.3 Scons 安装

步骤1 打开Linux终端，输入命令“sudo apt-get install scons -y”（需root/sudo权限安装）。

如果无法找到安装包，请从“<https://scons.org/pages/download.html>”下载源码包。解压源码包到任意目录，进入此目录运行命令“sudo python3 setup.py install”（需root/sudo权限安装），等待安装完毕。推荐Scons版本是3.0.4+。

步骤2 输入命令“scons -v”，查看是否安装成功。安装成功结果如图1-4所示。

图 1-4 scons 安装成功

```
SCons by Steven Knight et al.:
  script: v3.0.4.3a41ed6b288cee8d085373ad7fa02894
kufra
  engine: v3.0.4.3a41ed6b288cee8d085373ad7fa02894
kufra
  engine path: ['/usr/local/lib/scons/SCons']
Copyright (c) 2001 - 2019 The SCons Foundation
```

----结束

Scons安装后的常见问题：



- 启动SDK编译时，终端提示“SCons 3.0.4 or greater required, but you have SCons 3.0.x”，说明编译主机Scons版本过低，需升级版本。
- 启动SDK编译时，终端提示“SCons version x.x.x does not run under Python version 3.x.x. Python 3 is not yet supported.”，说明编译主机Scons版本过低，不支持python3语法，需升级版本。
- 启动SDK编译时，终端提示“parallel builds are unsupported by this version of Python;”，说明由于python3.7版本移除了threadless模块，导致SCons 3.0.1之前版本无法支持多线程编译。可升级Scons版本解决此问题；也可继续使用单线程编译，对编译结果无实质影响。



2 编译 SDK

- 2.1 SDK目录结构介绍
- 2.2 编译SDK (SCONS)
- 2.3 编译SDK (Makefile)
- 2.4 修改烧写镜像构成

2.1 SDK 目录结构介绍

SDK根目录结构如表2-1所示。

表 2-1 SDK 根目录

目录	说明
app	应用层代码（其中包含demo程序为参考示例）。
boot	Flash bootloader代码。
build	SDK构建所需的库文件、链接文件、配置文件。
components	SDK组件目录。
config	SDK系统配置文件。
documents	文档目录（包括：SDK说明文档）。
include	API头文件存放目录。
output	编译时生成的目标文件与中间文件（包括：库文件、打印log、生成的二进制文件等）。
platform	SDK平台相关的文件（包括：镜像、内核驱动模块等）。
third_party	开源第三方软件目录。
tools	SDK提供的Linux系统和Windows系统上使用的工具（包括：NV制作工具、签名工具、Menuconfig等）。



目录	说明
SConstruct	SCons编译脚本。
build.sh	启动编译脚本，同时支持“sh build.sh menuconfig”进行客制化配置。
build_patch.sh	解压开源源码包和patch文件编译脚本。
Makefile	支持makefile编译，使用“make”或“make all”启动编译。
non_factory.mk	非厂测版本编译脚本。
factory.mk	厂测版本编译脚本。

解压缩SDK后的根目录，如图2-1所示。

图 2-1 解压缩 SDK 示例

```
.  
..  
app  
boot  
build  
build.sh  
components  
config  
documents  
factory.mk  
include  
Makefile  
non_factory.mk  
platform  
SConstruct  
third_party  
tools
```

2.2 编译 SDK (SCONS)

2.2.1 编译方法

根目录下执行“./build.sh”指令运行脚本编译，即可编译出对应的SDK程序。编译命令列表如表2-2所示。



表 2-2 build.sh 参数列表

参数	示例	说明
无	./build.sh	启动增量编译，默认编译app工程是demo。
all	./build.sh all	启动全量编译，默认编译app工程是demo。
app工程 目录名 称	./build.sh demo	参数输入app工程目录名称，启动增量编译，编译app工程是所输入的工程目录名称。默认是demo工程。
clean	./build.sh clean	清理编译过程中生成的中间文件和烧写文件。
menuco nfig	./build.sh menuconfi g	启动menuconfig图形配置界面。

编译结果输出到“output\bin”目录下（如表2-3所示）。

表 2-3 编译 demo 结果示例（以压缩升级为例）

文件名	说明
Hi3861_boot_signed.bin	带签名的bootloader文件。
Hi3861_boot_signed_B.bin	带签名的bootloader备份文件。
Hi3861_demo.asm	Kernel asm文件。
Hi3861_demo.map	Kernel map文件。
Hi3861_demo.out	Kernel 输出文件。
Hi3861_demo_allinone.bin	产线工装烧写文件（已经包含独立烧写程序和loader程序）。
Hi3861_demo_burn.bin	烧写文件，烧写程序建议使用“Hi3861_demo_allinone.bin”。
Hi3861_demo_flash_boot_ota.bin	Flash Boot升级文件。
Hi3861_demo_ota.bin	Kernel升级文件。
Hi3861_demo_vercfg.bin	Kernel和Boot的版本号文件。
Hi3861_loader_signed.bin	烧写工具使用的加载文件，烧写程序建议使用“Hi3861_demo_allinone.bin”。

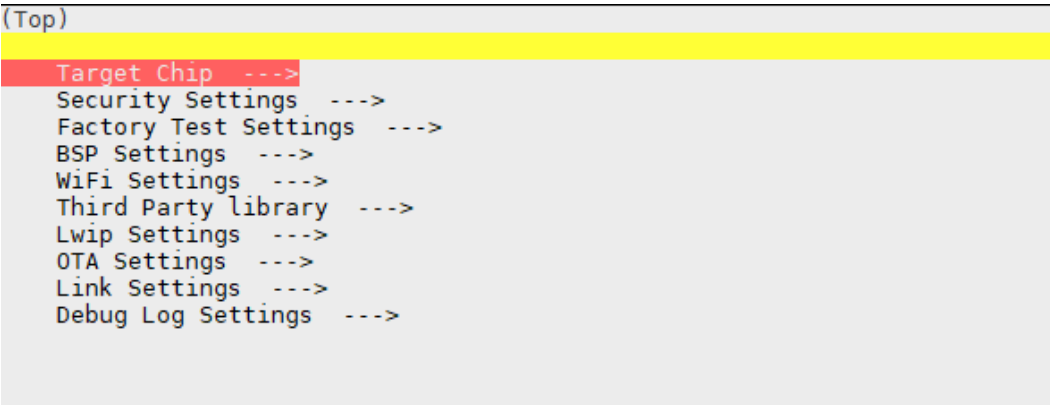
注：Hi3861_demo_allinone.bin为将要烧写到单板中的程序。



2.2.2 Menuconfig 配置

运行“sh build.sh menuconfig”脚本会启动Menuconfig程序，用户可通过Menuconfig对编译和系统功能进行配置。SDK集成了默认配置，但建议用户首次运行时进行相应配置，从而减少因为配置原因引起的问题。用户随时可以运行“sh build.sh menuconfig”更改配置。

图 2-2 Menuconfig 运行界面



注：界面如存在差异，以实际版本为准。

Menuconfig操作说明如表2-4所示，在Menuconfig界面中可输入快捷键进行配置。

表 2-4 Menuconfig 常用操作命令

快捷键	说明
空格，回车	选中，反选
ESC	返回上级菜单，退出界面
Q	退出界面
S	保存配置
F	显示帮助菜单

所有命令可在Menuconfig界面的下方查看Menuconfig官方说明解释（如图2-3所示）。

图 2-3 Menuconfig 命令帮助栏

[Space/Enter] Toggle/enter	[ESC] Leave menu	[S] Save
[O] Load	[?] Symbol info	[/] Jump to symbol
[F] Toggle show-help mode	[C] Toggle show-name mode	[A] Toggle show-all mode
[Q] Quit (prompts for save)	[D] Save minimal config (advanced)	

Menuconfig配置项说明如表2-5所示。



表 2-5 Menuconfig 配置项说明

菜单	配置项	说明
Target Chip	Hi3861	产品类型。
	Hi3861L	
Security Settings	Signature Algorithm for bootloader and upgrade file	Boot以及升级文件签名方式。支持RSA V15、RSA PSS、ECC、SHA256签名方式。
	boot ver(value form 0 to 16)	支持Boot版本号设置。
	kernel ver(value form 0 to 48)	支持Kernel版本号设置。
	TEE HUKS support	支持TEE HUKS接口。
	TEE HUKS demo support	支持TEE HUKS DEMO AT命令，详见《Hi3861V100 / Hi3861LV100 AT命令 使用指南》：安全存储相关AT指令章节。
	FLASH ENCRPT suppot	支持Flash安全加密。
Factory Test Settings	factory test enable	使能工厂产测。
BSP Settings	i2c driver support	支持I2C驱动。
	i2s driver support	支持I2S驱动。
	SPI driver support	支持SPI驱动。
	DMA driver support	支持DMA驱动。
	SDIO driver support	支持SDIO驱动。
	SPI support DMA	SPI驱动支持DMA传输模式。
	UART support DMA	UART驱动支持DMA传输模式。
	PWM driver support	支持PWM驱动。
	PWM hold after reboot	支持软复位后维持PWM状态。
	Enable AT command	支持AT命令。
	Enable file system	支持文件系统。
	Enable uart0 IO mux config	支持UART0。
	Enable uart1 IO mux config	支持UART1。
	Enable uart2 IO mux config	支持UART2。



菜单	配置项	说明
WiFi Settings	Enable WPS	WPA支持WPS（WIFI Protected Setup）。支持通过pin码和PBC连接网络。
	Authentication Option of Radio Parameters	选择验证方式，支持CE版本、FCC版本配置版本验证。
	Enable MESH	支持MESH功能。
Third Party library	cJson support	支持CJSON库编译。
	COAP support	支持libcoap库编译。
	MQTT support	支持MQTT库编译。
	iperf support	支持iperf库编译。
Lwip Settings	Enable Option Router (Option3)	支持设置网关地址选项。
	Enable DHCP Hostname (Option12)	支持设置/获取netif主机名。
	Enable DHCP Vendorname (Option60)	支持设置/获取供应商类别标识符信息。
	Lwip Support Lowpower Mode	配置LwIP支持低功耗模式（低功耗模式需要配置）。
OTA Settings	compression ota support	支持压缩升级。
	dual-partition ota support	支持双分区升级。
Link Settings	Hilink support	支持Hilink连接。
Debug Log Settings	Enable debug log	支持debug日志模式。

2.2.3 注意事项

- 如果执行“./build.sh”提示无权限，可执行命令“chmod +x build.sh”添加执行权限或执行“sh ./build.sh”。
- 编译过程中，报错找不到某个包，请检查环境中的python是否已经安装了相应组件。如果构建环境中包含多个python，特别是多个同版本的python，而用户无法辨认正在使用的是其中的哪个版本，此情况下，在安装python组件包时，推荐使用组件包源码进行安装。
- 系统优先使用用户通过Menuconfig所做的配置，如果用户未配置，系统将使用默认配置进行编译。



- 程序的烧写请参见《Hi3861V100 / Hi3861LV100 HiBurn工具 使用指南》。
- OTA Settings建议优先选择压缩升级模式，其相比双分区升级，镜像可用空间更大：默认多250KB+。如果选择双分区升级，需要考虑后续功能扩展情况，为后续的升级镜像预留足够空间，升级镜像不支持压缩升级和双分区升级混用。

2.3 编译 SDK (Makefile)

- 编译命令：
Wi-Fi顶层Makefile的使用说明：
 - 编译整个工程：
make / make all
 - 编译产测程序：
make clean;make factory;make clean_factory;make all
 - 清除整个工程的编译文件：
make clean
- makefile编译结构：



- 文件说明



表 2-6

文件	文件说明
顶层Makefile	编译入口
factory.mk	编译产测程序的makefile
non_factory.mk	编译正常程序的makefile
模块路径/Makefile	模块编译脚本
模块路径/ module_config.mk	模块编译配置
build\make_scripts \config.mk	公共编译配置，主要包含： 1.编译模块配置 2.各模块路径配置 3.公共头文件配置 4.公共编译宏配置 5.公共编译选项 6.链接库 7.链接库路径 8.中间文件输出路径
build\make_scripts \usr.mk	公共编译配置，一般由用户配置： 1.app名称 2.芯片配置 3.编译器配置 4.编译结果输出路径 5.用户库 6.用户库路径
build\make_scripts \config_lib_factory.m k build\make_scripts \config_lib_non_fact ory.mk	产测和正常程序链接的库及库路径
build\make_scripts \lib.mk	目标文件、库文件的生成
build\make_scripts \module.mk	各个模块的Makefile的原始备份（新增模块时可直接拷贝至新模块下作为Makefile）

- 新增组件makeifile配置说明

以组件名为XXX为例进行说明，以下说明为了通用性，做了一些约束说明，以下方法若不适用，可通过调整组件结构适应该方法，也可自行修改makefile以适应组件编译方式：



- 修改build\make_scripts\config.mk，可参照其他组件。
- 增加变量说明新增组件相对路径：XXX_dir := 相对路径。
- 增加变量说明新增组件需要编译生成的库文件：XXX_libs := 库文件名1 库文件名2（允许生成1个及1个以上的库文件）。
- 拷贝build\make_scripts\module.mk至新组件根目录下，重命名为Makefile。
- 新组件根目录下新增文件module_config.mk，添加内容可参见其他组件，其主要修改内容如下：
 - 添加编译库文件所需要的源文件路径，若有多个库文件，则添加多个（注：此处只支持添加目录，不允许添加文件形式，所有被增加的目录下的源文件和汇编文件都会被编译）。
 - 库文件名_srcs := 组件所在目录的相对路径。
 - 添加组件编译需要的头文件路径、编译宏、编译选项等（根据需要进行添加，可参见其他组件）。

2.4 修改烧写镜像构成

实际应用场景中，可能有修改烧写镜像的需求，包括：

- 修改Hi3861_xxx_allinone.bin（以下简称allinone.bin）镜像的构成，比如在该镜像中，预置证书文件等用户自定义文件或替换默认的烧写程序文件；
- 修改Hi3861_xxx_burn.bin（以下简称burn.bin）镜像的构成，比如，从该镜像中去掉BOOT和NV文件，避免二次开发时重复烧写NV，导致FLASH上已有的NV数据被覆盖。如，产测校准采用的模式为校准参数写NV而不是写EFUSE，则参数已经写入NV后，不希望后续再重新烧写NV文件。

建议优先采用修改allinone.bin的方法实现修改烧写镜像的需求，因为可以通过替换allinone.bin中默认的burn.bin，达到修改burn.bin方法的目的。

2.4.1 修改 Hi3861_xxx_allinone.bin 构成

allinone.bin，默认包含如下组成元素：

- Hi3861_loader_signed.bin：用于烧写efuse或flash依赖的镜像，该镜像不烧写到flash中，仅仅加载到内存中，用于烧写其它镜像使用；
- efuse_cfg.bin：当sdk的build/basebin目录下，包括该文件时，会被打包到allinone.bin中，其为预置的efuse参数；
- Hi3861_xxx_burn.bin：默认包含boot、NV、可执行程序镜像；
- Hi3861_xxx_factory.bin：产测镜像，仅当menuconfig开启产测开关时，allinone.bin会包含该镜像，用于产线烧录；
- burn_for_erase_4k.bin：内容为全FF的镜像，用于不需要全片擦除时，可通过烧写该文件到某个地址，指定擦除某一段FLASH内容；
- Hi3861_boot_signed_B.bin：备份FLASHBOOT，用于当FLASH首地址的FLASHBOOT被异常破坏时，系统可以从备份FLASHBOOT启动；
- Hi3861_xxx_tee_cert_key.bin：TEE HUKS DEMO依赖的镜像，预置的12KB的证书和秘钥文件，存放在用户预留区；



- Hi3861_xxx_vercfg.bin: 版本号配置文件, 仅当开启安全启动时, allinone.bin会包含该镜像, 用于配置efuse中的boot和kernel版本号。

all_inone.bin的构成, 通过一个字符串列表进行拼接, 单个元素的格式为: “%s|burn_start_addr|burn_erase_size|type”。其中:

第一个参数表示该构成元素的路径名称, 例如, efuse_cfg.bin对应的路径名称为:
efuse_bin = os.path.join(root_path, 'build', 'basebin', 'efuse_cfg.bin');

第二个参数表示该构成元素的烧写起始地址, 仅仅对需要烧写到FLASH上的元素有效, 其它元素填写0, 例如, efuse_cfg.bin和Hi3861_xxx_ver_cfg.bin不需要烧写到FLASH, 则填写0即可, Hi3861_xxx_burn.bin从FLASH的0地址开始烧写, 填写0;

第三个参数表示该构成元素烧写时擦除的大小, 仅仅对需要烧写到FLASH上的元素有效, 其它元素填写0, 例如, efuse_cfg.bin和Hi3861_xxx_ver_cfg.bin不需要烧写到FLASH, 则填写0即可, Hi3861_xxx_burn.bin烧写时, 需要全片擦除, 则填写0x200000, 对应2M FLASH大小;

第四个参数表示该构成元素的类型: 0表示Hi3861_loader_signed.bin, 3表示efuse_cfg.bin, 1表示需要烧写到FLASH上应用程序或数据, 6表示产测镜像, 7表示版本号配置文件。

修改字符串列表:

- 采用SCONS编译时, 需要修改SDK根目录下的Sconstruct文件中的burn_bin_builder函数。
- 采用Makefile编译时, 需要修改SDK中build/scripts/pkt_builder.py文件中的__main__函数, 函数内对应的分支为“elif type == 'burn_bin'”。为了确保修改生效, 需要指定build/make_scripts/config.mk中“PYTHON_SCRIPTS = y”。

不管是SCONS编译还是Makefile编译, 修改方法同理, 均是修改字符串列表的构成, 以及单个字符串列表元素的各个参数。以通过SCONS脚本, 修改allinone.bin构成, 使得烧写的镜像不包括BOOT和NV文件为例。

替换burn.bin为不包含boot和nv的镜像, 仅仅包含可执行程序, 该程序镜像位于build\build_tmp\cache\Hi3861_xxx_ota_temp.bin。(注意: 不能选择Hi3861_xxx_ota_kernel.bin, 其未经过报文头封装, 无法校验通过)

```
def burn_bin_builder(target, source, env):  
    """Build binary  
    """  
    get_ota_object().set_build_temp_path(build_temp_path = env_cfg.cache_path)  
    #burn_bin = get_ota_object().BuildHiBurnBin(str(target[0]), str(source[0]))  
    burn_bin = os.path.join('build', 'build_tmp', 'cache', '%s_ota_temp.bin'%env_cfg.target_name)  
    loader_bin = str(source[1])  
    efuse_bin = str(source[2]) if len(source) == 3 else None  
    boot_b = os.path.join("output", "bin", "%s_boot_signed_B.bin"%(env.get("CHIP_TYPE")))  
    boot_b_size = os.path.getsize(boot_b)  
    factory_bin_path = os.path.join('build', 'libs', 'factory_bin')  
    factory_bin = os.path.join(factory_bin_path, '%s_factory.bin'%env_cfg.target_name)
```

修改burn.bin的烧写起始地址和大小, 由于原始burn.bin默认包含了BOOT和NV, 所以从0地址开始烧写; 去掉BOOT和NV的bin文件后, 烧写起始地址需要对应后移52KB, 擦除的大小也需要比原始擦除大小减小52KB (根据分区表, BOOT占用32KB, NV占用20KB)。

```
burn_bin_ease_size = 0x200000;
#根据分区表适配写地址和大小
if (get_hilink_enable() == True):
    burn_bin_ease_size = 0x200000 - 0x8000 - 0x1000 - 0x2000
if (get_tee_huks_demo_enable() == True):
    burn_bin_ease_size = 0x200000 - 0x8000 - 0x1000 - 0x2000 - 0x5000
burn_bin_ease_size = burn_bin_ease_size - 0xD000

if os.path.exists(factory_bin):
    cmd_list = ['%s|0|0|0|loader_bin, "%s|0|0|3|fuse_bin, "%s|d|d|1|"%s(burn_bin, 0xD000, burn_bin_ease_size), "%s|d|d|d|6|"%s(factory_bin, 0x14D000, 0x960000) if %s|d|d|d|1|"%s(burn_bin, 0xD000, burn_bin_ease_size), "%s|d|d|d|6|"%s(factory_bin, 0x14D000, 0x960000).]
    shutil.copytree(factory_bin_path, os.path.join(env_cfg_bin_path, 'factory_bin'))
else:
    cmd_list = ['%s|0|0|0|0|loader_bin, "%s|0|0|3|fuse_bin, "%s|d|d|d|1|"%s(burn_bin, 0xD000, burn_bin_ease_size) if fuse_bin!=None else ["%s|0|0|0|0|loader_bin, "%s|d|d|d|1|"%s(burn_bin, 0xD000, burn_bin_ease_size)]

if (get_hilink_enable() == True) or (get_tee_huks_demo_enable() == True):
    cmd_list.append('%s|d|d|d|1|"%s(burn_for_erase_bin, 0x200000 - 0x8000 - 0x1000, 0x10000)'
```

如上图所示，可以通过修改cmd_list的构成，可以指定all_in_one.bin中包含任意的文件。编译生成的各目标文件，在SDK目录中均可获取，例如：

不包含BOOT和NV, 仅包含可执行程序的镜像, 位于build/build_tmp/cache目录下:
Hi3861 xxx ota temp.bin;

工厂区和非工厂区NV文件，位于build/build_tmp/nv目录下：Hi3861_xxx_factory.hnv
和Hi3861_ xxx_normal.hnv;

BOOT和备份BOOT文件，位于output/bin目录下：Hi3861_boot_signed.bin和Hi3861_boot_signed_B.bin。

2.4.2 修改 Hi3861 xxx burn.bin 构成

burn.bin，按照先后顺序默认包含如下组成元素。

- BOOT：占据32KB。
- 工厂区NV：占据8KB。
- 非工厂区NV：占据8KB。
- 非工厂区NV原始备份：占据4KB。
- 可执行程序镜像：占据实际镜像大小。

所有组成元素通过数组区间存放。

修改burn.bin的构成，需要修改build/scripts/make_upg_file.py中make_hbin函数。以修改burn.bin，使其不包含BOOT和NV为例。

```
kernel_st_addr = 0x0
bin_total_size = len(upg_file_bin)

boot_nv_kernel_bin = bytearray(bin_total_size)
boot_nv_kernel_bin[kernel_st_addr:kernel_st_addr + len(upg_file_bin)] = upg_file_bin
```

由于修改了burn.bin的构成，其在all_in_one.bin中指定的烧写地址也需要变化，原默认从0地址开始烧写，现需要从0xD000开始烧写，参见[2.4.1 修改Hi3861_xxx_allinone.bin构成](#)章节中，关于修改修改burn.bin的烧写起始地址和大小相关描述。

采用Makefile方式编译时，为了确保修改生效，需要指定build/make_scripts/config.mk中“PYTHON_SCRIPTS = y”。

2.4.3 注意事项

- 以Makefile方式编译时，如果修改了PYTHON脚本，均需要指定build/make_scripts/config.mk中“PYTHON_SCRIPTS = y”。如果保留“PYTHON_SCRIPTS=n”，则需要更新可执行文件build/scripts/ota_builder，确保修改生效。ota_builder为原始python脚本，通过PyInstaller和StaticX转换而成。



- 修改各元素的烧写地址和擦除大小时，务必和分区表配置相结合，分区表默认配置与修改方法，参见《Hi3861 / Hi3861LV100 SDK 开发指南》中FLASH分区与FLASH保护章节。



3 新建 APP

- 3.1 建立目录结构
- 3.2 开发代码
- 3.3 app_main接口
- 3.4 编译配置

3.1 建立目录结构

说明

用户可在demo同级目录下参考demo建立自己的app，以下均以建立my_demo为例。

建立目录结构的步骤如下：

- 步骤1** 拷贝工程根目录下的“app/demo/SConscript”到“app/my_demo/ SConscript”。
- 步骤2** 建立my_demo开发的目录结构（例如：src、include、init目录等），可参考“app/demo”进行建立，拷贝“app/demo/src/SConscript”到my_demo下新建的源码目录。

----结束

3.2 开发代码

目录结构建立完成后开始启动开发代码（用户可参考“app/demo”进行移植），代码开发完成后即可使用“sh ./build.sh my_demo”编译my_demo进行代码调试。

3.3 app_main 接口

demo中app_main接口在“app/demo/src/app_main.c”中实现（此为整个app程序的入口）。demo用例中app_main接口主要调用了一些程序初始化接口对芯片进行初始化，包括一些外设驱动的初始化（包括Flash、UART、WatchDog、IO、DMA、I2C、I2S、SPI等）和一些功能模块初始化（包括AT、DIAG、shell指令、WiFi、UPG等）。demo中实现了一些基本功能示例，用户可以参考demo在app_main中新建自己的app任务。



3.4 编译配置

3.4.1 编译基础脚本文件说明

编译基础脚本文件说明如表3-1所示（路径均为工程所在路径的相对路径）。

表 3-1 编译基础脚本文件说明

文件名	说明
SConstruct	SDK的编译脚本入口。
build/scripts/link.ld.S build/scripts/system_config.ld.S	SDK的链接脚本。
build/scripts/common_env.py	编译公共以及系统配置。
app/my_demo/SConscript	my_demo模块的编译脚本（顶层SConscript）。
app/my_demo/src/SConscript	my_demo模块的源码编译脚本（次级SConscript）。
app/my_demo/app.json	my_demo模块编译配置文件，内容需遵循json格式。

3.4.2 编写编译配置文件

步骤1 在my_demo下新建app.json，或将demo下的“app.json”拷贝至my_demo下。

步骤2 按照表3-2内容完成编译配置app.json，保存内容并退出。此文件是json文件，需遵循json格式要求。

表 3-2 app.json 编译选项配置

选项内容	选项说明	配置示例
TARGET_LIB	生成"TARGET_LIB"的.a库文件。	"TARGET_LIB": "my_demo"
APP_SRCS	包含SConscripts的源代码目录。需填写相对于此目录的相对路径。 例如：my_demo下拥有“init”与“src”目录，它们目录下已存在SConscripts文件，添加“init”和“src”到“APP_SRCS”即可。	"APP_SRCS": ["init","src"]



选项内容	选项说明	配置示例
INCLUDE	包含所需头文件的目录。需填写相对于SDK根目录的相对路径。如果无需另外的头文件目录，此项留空即可。 例如：my_demo需要引用存在于my_demo/include内的头文件，即将“app/my_demo/include”添加进“INCLUDE”。	"INCLUDE": ["app/my_demo/include", "config/app"]
CC_FLAGS	编译器的选项配置。类似于gcc的编译选项。如果无需使用，此项留空即可。	"CC_FLAGS": ["-Werror"]
DEFINES	gcc编译宏，配置内容即为gcc的-D选项，defines列表中每一项元素在编译时，会被当做gcc的“-D”选项。如果无需使用，此项留空即可。	"DEFINES": ["_PRE_WLAN_FEATURE_CSI", "_PRE_WLAN_FEATURE_P2P", "LWIP_ENABLE_DIAG_CMD=0"]
AR_FLAGS	静态库打包命令的编译选项。如果无需使用，此项留空即可。	"AR_FLAGS": ["-X32_64"]
LD_FLAGS	链接选项。如果无需使用，此项留空即可。	"LD_FLAGS": ["-A"]
AS_FLAGS	编译汇编文件选项。如果无需使用，此项留空即可。	"AS_FLAGS": ["--warn"]
LD_DIRS	链接目录选项，将所需的.a库文件放置在此目录下，链接时将自动添加进链接选项。如果无需使用，此项留空即可。	"LD_DIRS":["app/my_demo/libs"]
CLEAN	在执行“scons -c”清理工程时，自动删除的文件。一般情况下留空即可。如果有需要特殊处理的文件，例如生成的临时文件或中间文件等，可配置此项。此项需填写相对于SDK根目录的相对路径。	"CLEAN": ["app/my_demo/test.txt"]

步骤3 进入SDK根目录，运行“sh build.sh my_demo”命令开始编译。也可使用命令“scons app=my_demo”启动编译。如果需要修改默认编译的app工程，可修改build.sh文件，将启动编译的“scons”命令加上参数“scons app=my_demo”。

----结束



4 客制化 SDK 组件

说明

本章节介绍SDK构建系统的相关信息。一般情况下，用户仅需关注APP的开发即可完成项目的开发。如果用户通过APP的开发无法满足特定的需求，可通过新增、修改、删除SDK内组件，以完成项目开发。

[4.1 构建系统概述](#)

[4.2 新建组件](#)

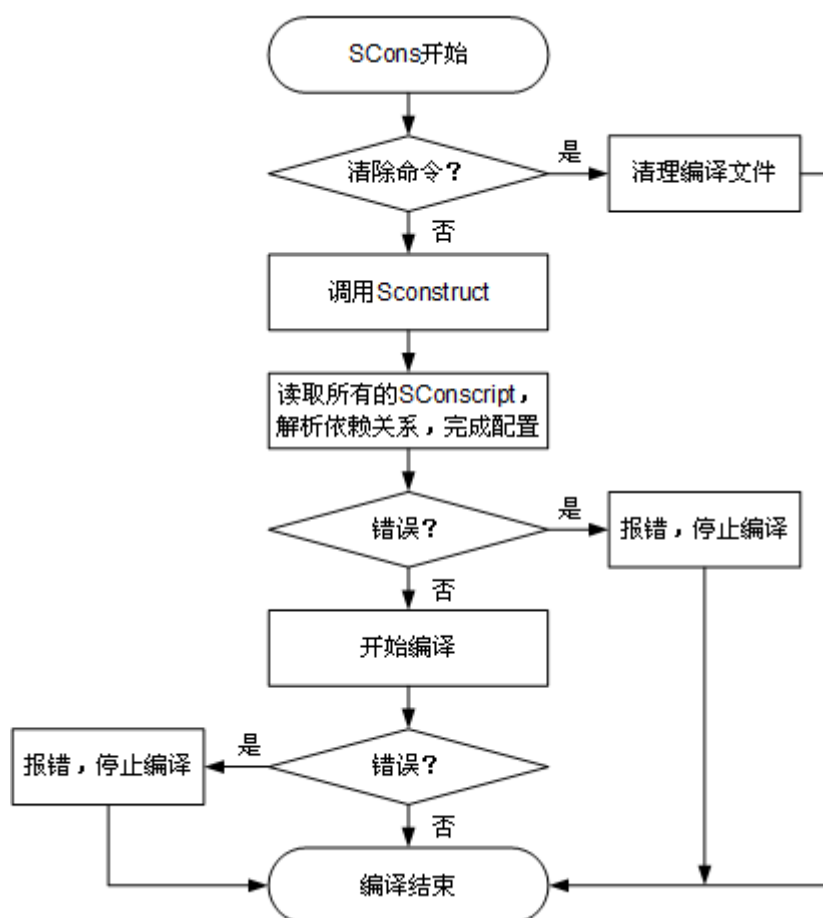
[4.3 客制化Menuconfig](#)

4.1 构建系统概述

Hi3861V100与Hi3861LV100的SDK使用SCons作为构建工具，因此，推荐使用SCons进行开发，从而保持编译的完整性和连贯性。SCons基于python开发，使用python语法。SCons的编译流程如[图4-1](#)所示。



图 4-1 SCons 编译流程



调试SCons构建系统的常见问题：由于SCons有预先读取脚本、动态生成构建依赖关系等编译信息的设定，因此它的编译开始是在这些操作完成之后，然而在预先读取脚本的过程中，将运行非SCons内容的python语句。例如：python语句的print函数将会在SCons预先读取脚本的过程中打印，真正开始编译的过程中，并没有print语句可执行。

SCons的详细特性请参见SCons官网“<http://www.scons.org/>”。

SDK构建系统常用文件以及描述如表4-1所示。

表 4-1 SDK 构建系统常用文件

文件名称以及路径	描述
build.sh	启动编译脚本，可编辑默认编译app工程。
SConstruct	SCons启动入口，文件名遵循业界规范。SDK编译流程主控实现文件，包括编译以及链接。SCons遵循此文件的实现，生成最终文件所依赖的库文件的依赖关系，以及生成方法。



文件名称以及路径	描述
SConscript	<p>从属SCons脚本，文件名称遵循业界规范。SDK中大致分成两类，结合以下示例说明：</p> <ul style="list-style-type: none">• app/demo/SConscript 配置生成库文件的脚本，SCons遵循此文件的实现，生成库文件所依赖的.o文件的依赖关系，以及生成方法（简称为1型SConscript脚本）。• app/demo/src/SConscript 配置生成.o文件的脚本，SCons遵循此文件的实现，生成.o文件所依赖的.c、.s文件的依赖关系，以及生成方法（简称为2型SConscript脚本）。 <p>一般情况下，1型SConscript和2型SConscript在物理位置上是树形结构，即1型属于根节点，2型属于叶子节点。</p>
build/scripts/common_env.py	<p>编译过程中，编译选项的配置文件，包括单个模块所依赖的头文件目录、单个模块的编译选项等编译相关的内容配置。</p>

4.2 新建组件

本章节以新建模块“my_module”为例，介绍如何修改构建系统、完成“my_module”模块的添加。“my_module”下包含头文件目录“include”、源码文件的“src”目录。整个模块放置在“components”下。

4.2.1 编译输出

SDK含有编译信息输出，如果需要打开详细信息，请修改“build/scripts/common_env.py”中“log_output_flag”的值。

- True：开启详细输出。
- False：关闭详细输出。

说明

详细输出可能与编译过程不同步，原因请参加“[4.1 构建系统概述](#)”。

4.2.2 编译配置

编译配置的步骤如下：

步骤1 将“my_module”目录放置于“components”目录下（例如：components/my_module）。

步骤2 在“my_module”目录下，添加第一层SConscript。

建议从SDK包含的APP目录中复制，例如：复制“app/demo/SConscript”至“components/my_module”下，此SConscript脚本即1型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“my_module/src”目录下添加次级SConscript。

建议从SDK包含的APP目录中复制次级目录中的SConscript，例如：复制“app/demo/src/SConscript”至“my_module/src/SConscript”下，此SConscript脚本即2型



SConscript脚本。当“my_module”有多个源码目录时，需要为每个源码目录复制一份2型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1型SConscript，只需要小幅度修改2型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2型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”配置库文件名、源码路径、模块编译配置等。

- （必选）添加新增的“my_module”名称到“compile_module”。
`compile_module = ['drv','sys','os','at','mqtt','mbedtls','sigma','my_module']`
- （必选）添加新增的“my_module”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1型SConscript。
`module_dir = {
 'drv': os.path.join('platform', 'drivers'),
 'my_module': os.path.join('components', 'my_module'), #填写相对于工程根目录的相对路径，此路径下包含1型SConscript
 ...
}`
- （必选）添加新增的“my_module”的库文件名称与源码文件目录到“proj_lib_cfg”。添加格式为‘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1型SConscript的相对路径。源码目录路径中包含2型SConscript。
`proj_lib_cfg = {
 'drv': {'adc': ['adc'],
 'flash': ['flash'],
 'spi': ['spi'],
 'uart': ['uart'],
 'pwm': ['pwm'],
 'i2c': ['i2c'],
 'my_module': {'my_module': ['src']}}, #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1型SConscript的相对路径。源码目录路径中，包含2型SConscript。
 ...
}`
- （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),`



```
        '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'
    ],
    'iperf':[], # 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“my_module”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“my_module”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'),
    ],
    'my_module':[os.path.join('#', 'components', 'my_module', 'include')],
    ...
}
```

----结束

“build/scripts/common_env.py”中承担了大部分的编译配置，常用的配置项如[表4-2](#)所示。

表 4-2 common_env.py 中的常用配置项

配置项	描述
compile_module	python list格式，内容为需要编译的子模块。
log_output_flag	详细编译内容输出使能开关。
os_lib_path	额外的链接目录。
module_dir	python dict类型，内容为子模块的配置。包括名称（需要和 compile_module中的一致）、模块路径等。模块路径即1型SConscript 路径。



配置项	描述
proj_lib_cfg	python dict类型，内容为子模块所包含的子目录的配置，子目录中含有.c、.s文件，即所有包含2型SConscript的目录。
proj_environment	编译的常用选项配置： <ul style="list-style-type: none">• ar_flags: 打包参数。• link_flags: 链接参数。• defines: 编译使用的宏定义，“common”为全局宏定义，可配置单模块的宏定义。• opts: 编译器支持的编译选项，“common”为全局选项，可单独配置子模块。• liteos_inc_path: os相关的头文件路径，“common”为全局添加的头文件目录，可单独配置子模块。内容填写相对于“platform/os/Huawei_LiteOS/”的相对路径。• common_inc_path: 除os相关的头文件路径，“common”为全局添加的头文件目录，可单独配置子模块。内容填写相对于SDK根目录的相对路径，以“#”开头。

4.3 客制化 Menuconfig

Menuconfig原为Linux Kernel的客制化工具，在SDK中用于完成配置功能开关、定义某配置的值等工作，同时支持用户自定义新内容。

以添加“my_module”编译开关为例，客制化Menuconfig的方法如下：

步骤1 编写kconfig文件：在“components/my_module/”中添加kconfig文件，并且完成配置内容编写。

kconfig的语法请参考Linux Kernel文档：<https://www.kernel.org/doc/html/latest/kbuild/kconfig.html>

步骤2 编辑“tools/menuconfig/Kconfig”文件：在文件最后新建一行，添加语句“source components/my_module/Kconfig”，保存并关闭。

步骤3 运行“sh build.sh menuconfig”，查看新配置是否显示在图形界面上。

步骤4 退出时选择“Save”选项，保存新配置。

步骤5 打开“build/config/usr_config.mk”文件，找到复制的配置项“CONFIG_ENABLE_MY_MODULE”。

步骤6 编辑“build/scripts/common_env.py”文件，找到注释“Configurations”，在此代码块中添加读取配置项的工作流，即当“CONFIG_ENABLE_MY_MODULE”设置为“y”时，系统编译“my_module”模块。完成后保存退出。

```
1 # Configurations
2 if scon_usr_bool_option('CONFIG_ENABLE_MY_MODULE') == 'y':
3     compile_module.append("my_module")
```

----结束