



# A76XX\_R3\_Series\_OpenSD K\_软件用户指南

LTE Module

## **SIMCom Wireless Solutions Limited**

Building B, SIM Technology Building, No.633, Jinzhong Road  
Changning District, Shanghai P.R. China

Tel: 86-21-31575100  
support@simcom.com  
www.simcom.com

文档名称:	A76xx_R3_Series_OpenSDK_软件用户指南
版本:	1.03
日期:	2023.12.05
状态:	已发布

## 版权声明

本手册包含芯讯通无线科技（上海）有限公司（简称：芯讯通）的技术信息。除非经芯讯通书面许可，任何单位和个人不得擅自摘抄、复制本手册内容的部分或全部，并不得以任何形式传播，违反者将被追究法律责任。对技术信息涉及的专利、实用新型或者外观设计等知识产权，芯讯通保留一切权利。芯讯通有权在不通知的情况下随时更新本手册的具体内容。

本手册版权属于芯讯通，任何人未经我公司书面同意进行复制、引用或者修改本手册都将承担法律责任。

### 芯讯通无线科技(上海)有限公司

上海市长宁区金钟路 633 号晨讯科技大楼 B 座 6 楼

电话：86-21-31575100

邮箱：simcom@simcom.com

官网：www.simcom.com

### 了解更多资料，请点击以下链接：

<http://cn.simcom.com/download/list-230-cn.html>

### 技术支持，请点击以下链接：

<http://cn.simcom.com/ask/index-cn.html> 或发送邮件至 [support@simcom.com](mailto:support@simcom.com)

版权所有 © 芯讯通无线科技(上海)有限公司 2023，保留一切权利。

## 前言

感谢使用 SIMCom 提供的 A76XX 模块。该文档适用于 SIMCom A76XX R3 Series, including A7670XX-XXXX, A7630 and A7680X。用于说明 SIMCOM SDK 的使用。

使用前请仔细阅读用户手册，您将领略其完善的功能和简洁的操作方法。

此模块主要用于语音或者数据通讯，本公司不承担由于用户不正常操作造成的财产损失或者人身伤害责任。请用户按照手册中的技术规格和参考设计开发相应的产品。同时注意使用移动产品应该关注的一般安全事项。在未声明之前，本公司有权根据技术发展的需要对本手册内容进行修改。

SIMCom  
Confidential

# 版本历史

版本	日期	章节	变更人	变更描述
V1.00.00	2022.6.13		陆国成	新版本
V1.00.01	2022.7.14		陆国成	修改一些描述错误
V1.00.02	2022.8.9		邱杰	编译架构调整
V1.00.03	2023.09.14		邱杰	补充完善编译说明
V1.01.00	2023.10.11		邱杰	1. 文档更名为软件用户指南 2. demo 和三方库编译控制方法变更
V1.02	2023.10.13	第 4 章	邱杰	修正客户配置 app 编译的方法
V1.03	2023.12.5	第 9 章	刘桂含	添加自定义代码封装指南

# 目录

目录.....	4
1 SDK 目录结构.....	5
2 编译环境配置.....	8
2.1 工具配置.....	8
2.2 make 工具选择.....	8
3 编译指令.....	9
3.1 编译生成.....	9
3.2 生成的所有目标文件.....	10
4 SIMCOM 预设模块编译控制.....	11
4.1 可视化定制编译配置（需安装工具）.....	11
4.1.1 menuconfig 配置.....	11
4.1.2 guiconfig 配置.....	12
4.2 直接修改配置文件（不需要安装工具）.....	13
4.3 恢复默认配置（需安装工具）.....	14
5 编译脚本中添加客户宏.....	15
6 新增代码模块进行编译.....	16
6.1 编写项目代码.....	16
6.2 为项目编写 makefile.....	16
6.3 将项目 makefile 添加到 SIMCOM 编译流程中.....	19
7 裁剪删除 SIMCOM demo.....	21
8 自定义 APP 入口函数.....	22
8.1 新建入口函数源文件.....	22
8.2 取消 simcom 原始入口源文件编译，并删除原始文件.....	22
8.3 把客户入口源文件添加到编译脚本.....	22
8.4 入口源文件编写规则.....	23
9 自定义代码封装成库.....	26
9.1 新建自定义源代码文件.....	26
9.2 修改根目录 CMakeLists.txt 配置文件.....	28
9.3 编译源文件生成 lib 库.....	29
9.4 删除源文件并引入编译好的库.....	30
9.5 编译测试.....	32

# 1 SDK 目录结构

B01V01A7690M6A_SDK_2... > 22104B01V01A7690M6A_SDK_230818		
名称	修改日期	类型
config	18/8/2023 星期五 10:...	文件夹
examples	18/8/2023 星期五 10:...	文件夹
kernel	18/8/2023 星期五 10:...	文件夹
out	14/9/2023 星期四 10:...	文件夹
public_libs	18/8/2023 星期五 10:...	文件夹
sc_demo	18/8/2023 星期五 10:...	文件夹
sc_lib	18/8/2023 星期五 10:...	文件夹
tools	18/8/2023 星期五 10:...	文件夹
app_build_doc.md	18/8/2023 星期五 10:...	Markdown 源文件
CMakeLists.txt	18/8/2023 星期五 10:...	文本文档
makeDepend.mak	18/8/2023 星期五 10:...	MAK 文件
Makefile	18/8/2023 星期五 10:...	文件
sc_application.c	18/8/2023 星期五 10:...	C 源文件

图 1 SDK 目录结构

工具	作用
config	每个模块关于 APP 编译的一些编译选项, app 链接脚本, cmake 编译 tool 配置。
examples	示例代码, 不参与编译。
kernel	每个模块 kernel 编译产生的文件(打包时需要依赖)。
out	编译生成的目标文件所在位置。
public_libs	公共库/开源库算法集成。
sc_lib	SIMCOM 提供的 API 静态库和头文件。
tools	linux/windows 编译和打包工具。
app_build_doc.md	app 编译简单介绍。
cmakelists.txt	代码编译脚本入口。
makefile	命令行编译指令入口。
makeDepend.mak	Makefile 的依赖关系文件。
sc_application.c	客户 APP 代码入口示例。

22104B01V01A7690M6A_SDK_230818 > config		在 co
名称	修改日期	类型
A7690C_LANS_1606_V102_OPENSdk_linkscript.ld	18/8/2023 星期五 10:...	LD 文件
buildOptions.cmake	18/8/2023 星期五 10:...	CMake 源文件
Config_APP.cmake	18/8/2023 星期五 10:...	CMake 源文件
make_image_16xx.mak	18/8/2023 星期五 10:...	MAK 文件
make_image_16xx_settings.mak	18/8/2023 星期五 10:...	MAK 文件
ToolChain.cmake	18/8/2023 星期五 10:...	CMake 源文件
userspaceConfig.h	18/8/2023 星期五 10:...	C Header 源文件
userspaceConfig.h.in	18/8/2023 星期五 10:...	IN 文件

工具	作用
xxxxxxx.ld	对应模块链接脚本。
buildOptions.cmake	SIMCOM 预设模块编译开关。与 userspaceConfig.h.in 配合使用。
Config_APP.cmake	SIMCOM 预设 APP 宏定义。
make_image_***.mak	烧录包生成脚本。
ToolChain.cmake	APP 编译脚本工具链及参数配置。
userspaceConfig.h	SIMCOM 预设模块编译开关自动生成头文件。编译时生成。代码使用预设开关，需要包含此头文件。
userspaceConfig.h.in	SIMCOM 预设模块编译开关定义。与 buildOptions.cmake 配合使用。

22104B01V01A7690M6A\_SDK\_230818 > tools

名称	修改日期	类型
linux	18/8/2023 星期五 10:...	文件夹
script	18/8/2023 星期五 10:...	文件夹
win32	18/8/2023 星期五 10:...	文件夹

工具	作用
linux	linux 系统编译工具。
script	编译过程使用脚本。
win32	windows 系统编译工具。



22104B01V01A7690M6A\_SDK\_230818 > tools > win32

名称	修改日期	类型
7z	18/8/2023 星期五 10:...	文件夹
aboot	18/8/2023 星期五 10:...	文件夹
cmake.zip	18/8/2023 星期五 10:...	WinRAR ZIP 压缩...
crc_set.exe	18/8/2023 星期五 10:...	应用程序
cross_tool.zip	18/8/2023 星期五 10:...	WinRAR ZIP 压缩...
GNUMake.exe	18/8/2023 星期五 10:...	应用程序
lzma.exe	18/8/2023 星期五 10:...	应用程序
msys64.zip	18/8/2023 星期五 10:...	WinRAR ZIP 压缩...
ninja.exe	18/8/2023 星期五 10:...	应用程序

工具	作用
cross_tools.zip	交叉编译工具链，用于编译 app
crc_set	app 校验工具
7z	压缩工具
aboot	ASR 烧录镜像打包工具
cmake	window 的 cmake 组装工具
Ninja（可选/默认）	windows 的代码编译系统
GNUMake（默认）	Windows 下的 make 工具
msys64.zip	windows 下 linux 虚拟环境及常用 Linux 指令工具
python3（可选）	主要用于执行 menuconfig 和 guiconfig（SDK 使用版本为 3.8.5）

22104B01V01A7690M6A\_SDK\_230818 > tools > linux

名称	修改日期	类型
aboot	18/8/2023 星期五 10:...	文件夹
cmake.tar.bz2	18/8/2023 星期五 10:...	WinRAR 压缩文件
crc_set	18/8/2023 星期五 10:...	文件
cross_tool.tar.bz2	18/8/2023 星期五 10:...	WinRAR 压缩文件
make_image.sh	18/8/2023 星期五 10:...	Shell Script

工具	作用
cross_tools.tar.bz2	交叉编译工具链，用于编译 app
crc_set	app 校验工具
aboot	ASR 烧录镜像打包工具
cmake.tar.bz2	window 的 cmake 组装工具
make_image.sh	烧录包生成脚本



## 2 编译环境配置

### 2.1 工具配置

配置项	方法
kernel 编译环境配置	直接释放 bin 文件，不需要编译
app 编译环境配置（win32）	编译工具在 tools/win32 目录下，需要将此路径添加到系统环境 PATH 中，或者将 GNUmake.exe 放到 windows 目录下并重命名为 make.exe。若已有 make.exe 工具，则不需要重命名。 cmd 执行“make+回车”即可看到目标；若 GNUmake.exe 没有重命名，则执行 gnumake+回车。
app 编译环境配置（linux）	通常无需额外配置

### 2.2 make 工具选择

make 工具可以选择 gnumake 或者 ninja。默认情况下，Windows 使用 ninja，linux 使用 make。若需要调整，根据 windows 和 linux 的设置，用户可以自行更改，具体如下图：

```
tools > script > env.mak
24  RMDIRARG := /s /q
25  MOVE := ren
26  BUILD_TYPE := "Ninja"
27  BUILD := ninja
28  MAKE := gnumake
```

ninja 配置

```
tools > script > env.mak
53  RMDIRARG := -rf
54  MOVE := mv
55  BUILD_TYPE := "Unix Makefiles"
56  BUILD := make
57  MAKE := make
```

make 配置

## 3 编译指令

执行动作	指令
编译生成目标文件	根目录下执行 <code>make/gnumake A7670C_LANS</code>
清除某个模块的编译	根目录下执行 <code>make/gnumake clean_ A7670C_LANS</code>
清除所有模块的编译	根目录下执行 <code>make/gnumake clean</code>

### 3.1 编译生成

APP 的代码或配置做了改动，请执行编译操作。用户在 SDK 的根目录启动 cmd 终端，根据模组的型号，输入对应的型号进行编译。本次使用 A7670C\_LANS 作示例。

```
D:\Workspace\code\ASR16060PEN\cus_application\release\SIMCOM_SDK_SET>make A7670C_LANS
```

图 2 编译 app

编译成功的效果如下：

```
Calculating total progress weight ...  
Done.  
Generating download commands ...  
Done.  
Generating crane firmware image ...  
Done.  
Generating target release package ...  
Done.  
Release package generated successfully!  
gnumake[1]: Leaving directory `D:/Workspace/code/ASR16060PEN/cus_application/release/SIMCOM_SDK_SET/scrip  
----- build success!! -----
```

图 3 编译成功的提示

执行完成后，会在./out 下生成所有目标文件。

#### NOTE

路径请不要包含中文和空格。

### 3.2 生成的所有目标文件

查看 压缩的文件夹工具

\_application > release > SIMCOM\_SDK\_SET > out > A7670C\_LANS

名称	修改日期	类型	大小
CMakeFiles	2022/8/9 11:07	文件夹	
lib	2022/8/9 11:07	文件夹	
sc_demo	2022/8/9 11:07	文件夹	
A7670C_LANS.zip	2022/8/9 11:07	WinRAR ZIP 压缩...	11,603 KB
cmake_install.cmake	2022/8/9 11:07	CMake 源文件	2 KB
CMakeCache.txt	2022/8/9 11:07	文本文档	17 KB
customer_app.bin	2022/8/9 11:07	BIN 文件	131 KB
customer_app.elf	2022/8/9 11:07	ELF 文件	242 KB
customer_app.elf.map	2022/8/9 11:07	MAP 文件	401 KB
Makefile	2022/8/9 11:07	文件	7 KB

图 4 成功生成的固件包

## 4 SIMCOM 预设模块编译控制

客户配置编译遵循如下规则：

1. 客户关闭某模块编译一定会生效。
2. 客户打开某模块编译时，若 **kernel** 适配了该功能，则生效，若 **kernel** 没有适配该功能，则不生效。
3. 判断 **kernel** 是否有适配某个模块的方法为，在 **sc\_lib** 目录下查找是否有该模块的头文件。

### 4.1 可视化定制编译配置（需安装工具）

SDK 配置 Kconfig 可视化编译配置功能。该功能基于 Kconfig，Windows 下需要 python 支持，并安装必要的 python 插件。

SDK 使用 python 版本为 3.8.5。

python 必要插件安装如下：

- 1) pip install windows-curses
- 2) pip install kconfiglib 或者 pip install -i <https://pypi.tuna.tsinghua.edu.cn/simple/kconfiglib>

**重点：**可视化配置必须依赖 **python** 及其插件，在没有安装必要的工具之前，不要使用可视化配置指令，否则可能导致编译失败。

#### 4.1.1 menuconfig 配置

menuconfig 配置和 linux 内核的 menuconfig 使用方法一样。启动方法为：gnumake menuconfig。

该配置方法全程在命令行进行。不支持鼠标和上下左右箭头操作。

上下左右分别对应字母 jkhl，选中操作为空格键，ESC 退出。l 进入下一级菜单，h 进入上一级菜单。

每个选项前面有一对中括号，括号中有\*号表示该项参与编译，括号中为空表示该项不参与编译。

配置完成后使用字母 S 进行保存即可。

**BUILD SIMCOM DEMO:** 控制整个 demo 的编译，关闭后，所有 demo 都不进行编译，并且不可再单独配置单个 demo。

**Demo for driver:** 驱动的 demo 列表，进入后可对单个驱动 demo 进行编译控制。

**Demo for modem:** modem 的 demo 列表，进入后可对单个 modem 的 demo 进行编译控制。

**Demo for application:** app 的 demo 列表，进入后可对单个 app 的 demo 进行编译控制。

**Third lib configurations:** 三方库列表，进入后可对单个三方库进行编译控制。

示例如下：

```
(Top)
SIMCOM openSDK USER configurations
[*] BUILD SIMCOM DEMO
    Demo for driver --->
    Demo for modem --->
    Demo for application --->
    Third lib configurations --->

[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)
```

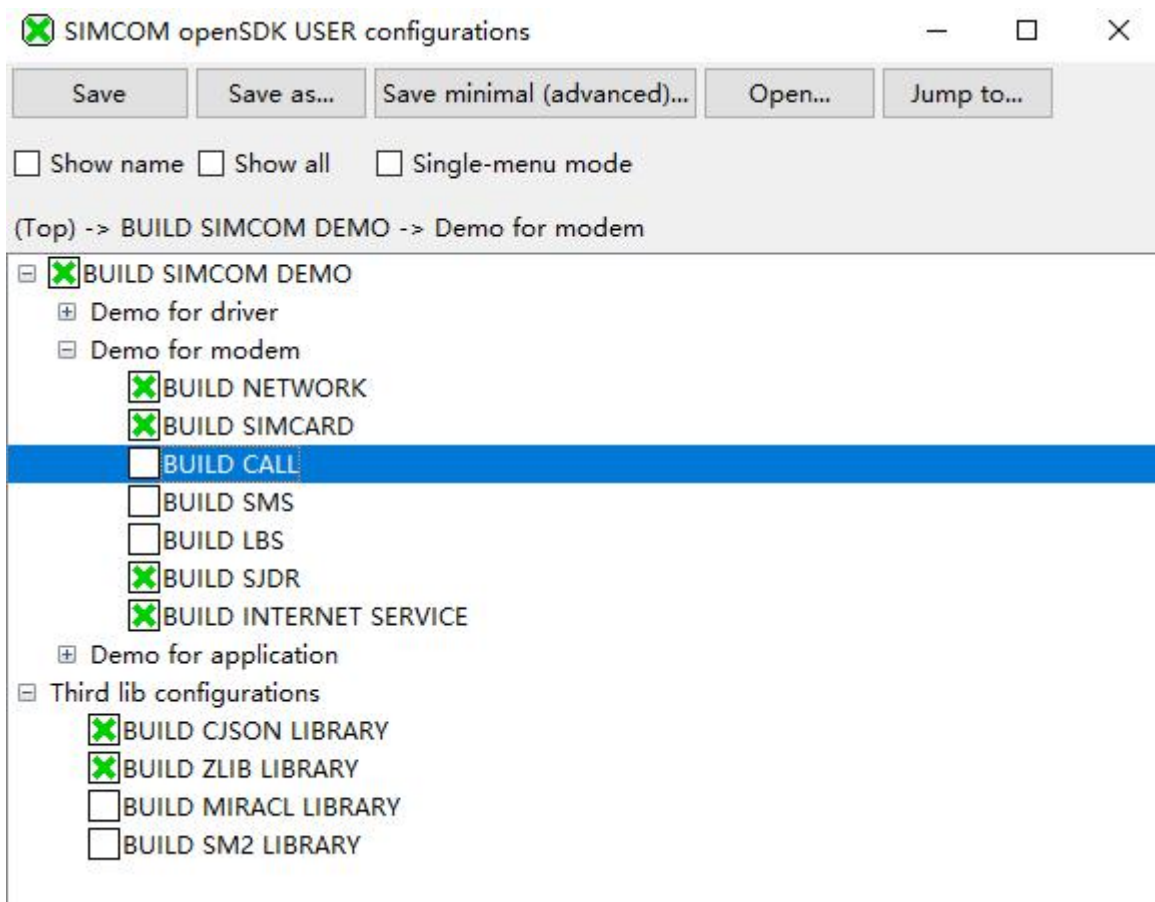
```
(Top) → BUILD SIMCOM DEMO → Demo for modem
SIMCOM openSDK USER configurations
[*] BUILD NETWORK
[*] BUILD SIMCARD
[ ] BUILD CALL
[ ] BUILD SMS
[*] BUILD LBS
[*] BUILD SJDR
[*] BUILD INTERNET SERVICE

[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)
```

### 4.1.2 guiconfig 配置

guiconfig 和 menuconfig 类似，只是 guiconfig 不在命令行进行，而是会弹出一个窗口，使用鼠标进行操作。其基本方法和功能和 menuconfig 相同。

示例如下：



## 4.2 直接修改配置文件（不需要安装工具）

不安装 python 及其插件则不可使用可视化编译配置，此时可通过直接修改配置文件进行配置。

用户需要修改的配置文件为：[./config/.config]

选项赋值为 y（必须小写）表示参与编译。

选项赋值为 n（必须小写）或者注释掉该行表示不参与编译。

选项赋值为其他值都将无效，编译时等效于不参与编译。但此时若再使用可视化编译，则会认为是一个新的配置像，配置工具会将其设置为默认值，默认值可能与客户实际想象的不一样，因此客户需要避免该情况的出现。

配置示例如下：



```
userspace > config > <> .config
1  CONFIG_HAS_DEMO=y
2
3  #
4  # Demo for driver
5  #
6  CONFIG_HAS_DEMO_WTD=y
7  CONFIG_HAS_DEMO_PMU=y
8  CONFIG_HAS_DEMO_GPIO=n
9  CONFIG_HAS_DEMO_PWM=n
10 CONFIG_HAS_DEMO_UART=y
11 CONFIG_HAS_DEMO_USB=y
12 CONFIG_HAS_DEMO_I2C=n
13 CONFIG_HAS_DEMO_SPI=y
14 CONFIG_HAS_DEMO_FLASH=y
15 # CONFIG_HAS_DEMO_OTA=y
16 # CONFIG_HAS_DEMO_GNSS=y
17 # CONFIG_HAS_DEMO_LCD=y
18 CONFIG_HAS_DEMO_CAM=y
19 CONFIG_HAS_DEMO_SYS=y
20 # end of Demo for driver
21
```

### 4.3 恢复默认配置（需安装工具）

恢复默认配置的方法是 clean 掉 config, 此时编译会自动按照默认配置进行编译。clean 方法为: gnumake clean\_config。


注意：恢复默认配置需要调用工具重新生成默认值。如果没有安装工具时调用该指令，会导致文件缺失而无法继续编译。



## 5 编译脚本中添加客户宏

客户添加宏定义的方法有两种：

- 1) 自己定义头文件，并在所有使用的源文件中 include 该头文件。
- 2) 在 config/Config\_APP.make 中添加宏定义，此方法定义的宏全局生效，如下图：



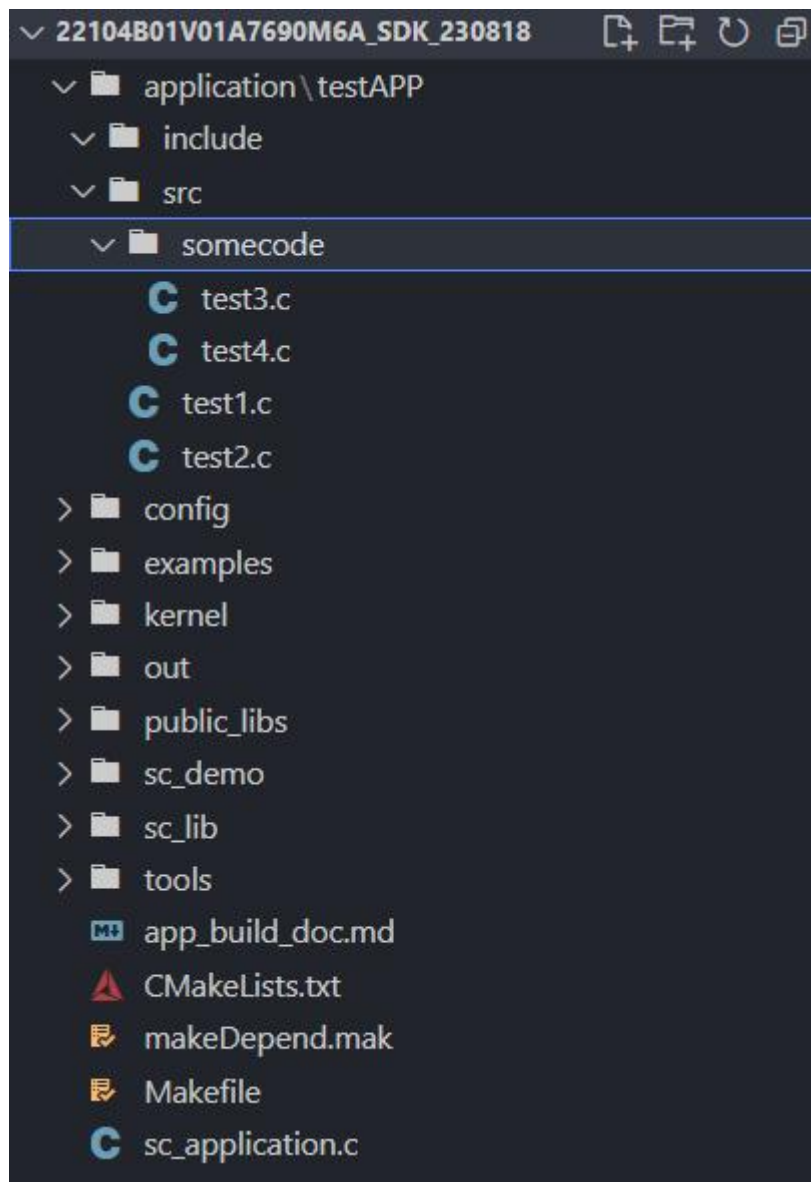
```

userspace > config > Config_APP.cmake > ...
You, 11个月前 | 1 author (You)
1  #设置私有编译宏
2  SET(SDK_ADD_PRIV_PREMACRO
3      -D__TM_ZONE=tm_zone
4      -D__TM_GMTOFF=tm_gmtoff
5      -D_REENT_SMALL
6  )
  
```

## 6 新增代码模块进行编译

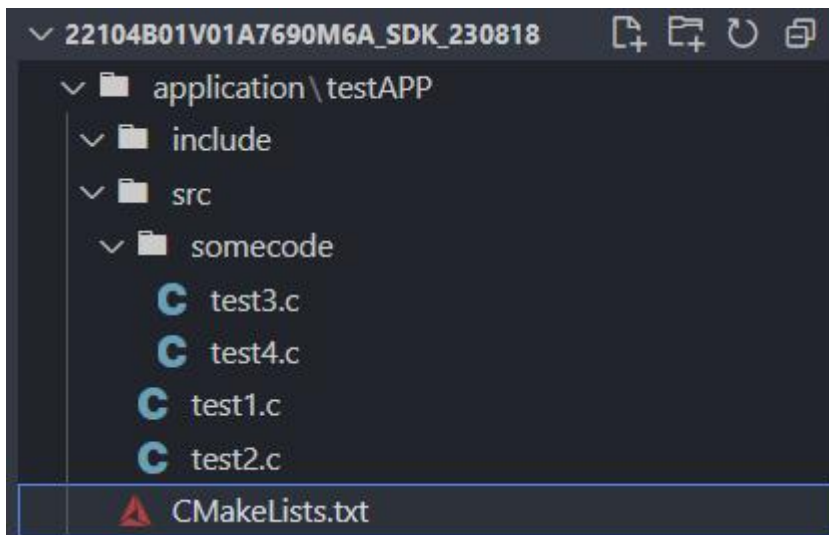
### 6.1 编写项目代码

示例项目名为 testAPP，代码目录结构如下：



### 6.2 为项目编写 makefile

1) 在 testAPP 根目录下新建 CMakeLists.txt，放入 testAPP 根目录：



- 2) 声明 cmake 版本:

```
application > testAPP > CMakeLists.txt > ...
1  cmake_minimum_required(VERSION 3.10)
2
```

- 3) 将 src 目录下所有代码文件加入编译列表:

```
application > testAPP > CMakeLists.txt > ...
1  cmake_minimum_required(VERSION 3.10)
2
3  AUX_SOURCE_DIRECTORY(./src testAPP_src)
```

- 4) 将 src/somecode 目录下 test3.c 和 test4.c 加入编译列表:

```
application > testAPP > CMakeLists.txt > ...
1  cmake_minimum_required(VERSION 3.10)
2
3  AUX_SOURCE_DIRECTORY(./src testAPP_src)
4  list(APPEND testAPP_src ./src/somecode/test3.c ./src/somecode/test4.c)
```

- 5) 将编译列表中的所有文件进行编译, 并生成为一个 lib 文件:

```
application > testAPP > CMakeLists.txt > ...
1  cmake_minimum_required(VERSION 3.10)
2
3  AUX_SOURCE_DIRECTORY(./src testAPP_src)
4  list(APPEND testAPP_src ./src/somecode/test3.c ./src/somecode/test4.c)
5
6  # Add the static library
7  add_library(testAPP STATIC ${testAPP_src})
```

- 6) 设置编译等级 (若要降低编译等级, 则不设置该项, 不建议降低等级):

```
application > testAPP > CMakeLists.txt > ...
1  cmake_minimum_required(VERSION 3.10)
2
3  AUX_SOURCE_DIRECTORY(./src testAPP_src)
4  list(APPEND testAPP_src ./src/somecode/test3.c ./src/somecode/test4.c)
5
6  #Add the static library
7  add_library(testAPP STATIC ${testAPP_src})
8  target_compile_options(testAPP PUBLIC -Werror)
```

7) 为 testAPP 项目指定头文件搜索路径:

```
application > testAPP > CMakeLists.txt > ...
1  cmake_minimum_required(VERSION 3.10)
2
3  AUX_SOURCE_DIRECTORY(./src testAPP_src)
4  list(APPEND testAPP_src ./src/somecode/test3.c ./src/somecode/test4.c)
5
6  #Add the static library
7  add_library(testAPP STATIC ${testAPP_src})
8  target_compile_options(testAPP PUBLIC -Werror)
9  target_include_directories(testAPP PUBLIC
10     ./include
11     ${CMAKE_SOURCE_DIR}/config
12     ${CMAKE_SOURCE_DIR}/sc_lib/inc
13     ${CMAKE_SOURCE_DIR}/sc_lib/inc/GPIO
14     ${CMAKE_SOURCE_DIR}/sc_lib/inc/token
15     ${CMAKE_SOURCE_DIR}/sc_lib/inc/utils
16     ${CMAKE_SOURCE_DIR}/sc_lib/${SCMODULE}/inc
17 )
```

8) 为 testAPP 编译生成的 lib 文件指定存放路径:

```
application > testAPP > CMakeLists.txt > ...
1  cmake_minimum_required(VERSION 3.10)
2
3  AUX_SOURCE_DIRECTORY(./src testAPP_src)
4  list(APPEND testAPP_src ./src/somecode/test3.c ./src/somecode/test4.c)
5
6  #Add the static library
7  add_library(testAPP STATIC ${testAPP_src})
8  target_compile_options(testAPP PUBLIC -Werror)
9  target_include_directories(testAPP PUBLIC
10     ./include
11     ${CMAKE_SOURCE_DIR}/config
12     ${CMAKE_SOURCE_DIR}/sc_lib/inc
13     ${CMAKE_SOURCE_DIR}/sc_lib/inc/GPIO
14     ${CMAKE_SOURCE_DIR}/sc_lib/inc/token
15     ${CMAKE_SOURCE_DIR}/sc_lib/inc/utils
16     ${CMAKE_SOURCE_DIR}/sc_lib/${SCMODULE}/inc
17 )
18 SET(LIBRARY_OUTPUT_PATH ${PROJECT_BINARY_DIR}/lib)
```

### 6.3 将项目 makefile 添加到 SIMCOM 编译流程中

该过程操作文件为 SDK 根目录下 CMakeLists.txt

- 1) 指定项目编译，生成 lib 文件（注意添加位置，不要放到不必要的 if 条件中）：



```

CMakeLists.txt > ...
77     ...     endif()
78     ...     add_subdirectory(../sc_demo sc_demo)
79     ...     endif()
80     ... endif(HAS_DEMO)
81
82     ... if(PROJECT_YSZNBG)
83     ...     add_subdirectory(../application/ysznbg ysznbg)
84     ... endif(PROJECT_YSZNBG)
85
86     ... if(PROJECT_TBDTU)
87     ...     add_subdirectory(../application/topband topband)
88     ... endif(PROJECT_TBDTU)
89
90     ... add_subdirectory(../application/testAPP testAPP)
91
92     ... # 外部库列表
93     ... add_library(third_libs OBJECT IMPORTED GLOBAL)
94
95     ... if(DEFINED THIRD_LIB)
96     ...     set_property(TARGET third_libs APPEND PROPERTY IMPORTED_OBJECTS
97     ...         ${THIRD_LIB})
98     ... )
99     ... endif()
100

```

- 2) 将生成的 lib 文件链接到最终的 elf 文件和 bin 文件中（注意添加位置，不要放到不必要的 if 条件中）：

```

CMakeLists.txt > ...
211     ...     target_link_libraries(userspace PRIVATE sm2)
212     ...     endif(HAS_SM2)
213     ...     if(HAS_CJSON)
214     ...         target_link_libraries(userspace PRIVATE cJSON)
215     ...     endif(HAS_CJSON)
216     ... endif(NOT DEFINED SIMCOM_SDK)
217
218     ... if(PROJECT_YSZNBG)
219     ...     target_link_libraries(userspace PRIVATE ysznbg)
220     ... endif(PROJECT_YSZNBG)
221
222     ... if(PROJECT_TBDTU)
223     ...     target_link_libraries(userspace PRIVATE topband)
224     ... endif(PROJECT_TBDTU)
225
226     ... target_link_libraries(userspace PRIVATE testAPP)
227
228     ... # 设置连接脚本
229     ... SET(LINK_SCRIP "${CMAKE_CURRENT_SOURCE_DIR}/config/${SCMODULE}_linkscript.ld")
230
231     ... TARGET_LINK_OPTIONS(userspace PRIVATE
232     ...     -gc-sections -T${LINK_SCRIP}
233     ...     -NOSTDLIB ${SDK_ADD_PRIV_WRAPFLAGS}
234     ... )

```

## 7 裁剪删除 SIMCOM demo

- 1) 关闭 demo 编译开关，确保 buildOptions.cmake 中 HAS\_DEMO 的配置为 OFF:
- 2) 删除 sc\_demo 文件夹

SIMCom  
Confidential



## 8 自定义 APP 入口函数

### 8.1 新建入口函数源文件

SDK 根目录下新建源文件，教材示例名字为：test\_cus\_main.c

源文件也可存放在其他路径下，在添加编译脚本是带上完整路径名即可，教材示例中不做该展示。

### 8.2 取消 simcom 原始入口源文件编译，并删除原始文件

修改文件为根目录下 CmakeLists.txt

1) 删除或注释下图中高亮的代码：



```

156  ... ./ql_lib/inc/lwip/lwip
157  ... ./ql_lib/inc/lwip/netif
158  ... )
159  ... endif()
160  ... else()
161  ... SET(app_src sc_application.c)
162  ... INCLUDE_DIRECTORIES(
163  ...     ./config
164  ...     ./sc_lib/inc
165  ...     ./sc_lib/${SCMODULE}/inc
166  ... )
    
```

2) 删除 SDK 根目录下的 sc\_application.c

### 8.3 把客户入口源文件添加到编译脚本

修改文件为根目录下 CmakeLists.txt

1) 把客户的源文件加入到下图高亮位置：

```

CMakeLists.txt > ...
156     ../ql_lib/inc/lwip/lwip
157     ../ql_lib/inc/lwip/netif
158 )
159 endif()
160 else()
161     # SET(app_src sc_application.c)
162     SET(app_src ./test_cus_main.c)
163     INCLUDE_DIRECTORIES(
164         ../config
165         ../sc_lib/inc
166         ../sc_lib/${SCMODULE}/inc
167     )

```

## 8.4 入口源文件编写规则

- 1) 添加必要头文件:

```

C test_cus_main.c > ...
1  #include "userspaceConfig.h"
2  #include "api_map.h"
3

```

- 2) 编写 main 函数, 名字由客户自行定义, 教程示例使用 test\_cus\_main; 在 main 函数中添加初始化代码调用 ApiMapInit(arg), 该调用必须放在 main 函数中最开始的位置, 客户自己的代码必须添加在 ApiMapInit 之后:

```

C test_cus_main.c > ...
1  #include "userspaceConfig.h"
2  #include "api_map.h"
3
4  void test_cus_main(void *arg)
5  {
6      ApiMapInit(arg);
7
8      // start add cus code there.
9  }

```

- 3) 添加 abort 函数, 该函数主要会被部分 C/C++ 库调用, 在 C/C++ 库函数中产生引起系统挂起问题时, 会调用该函数:

```

C test_cus_main.c > ...
1  #include "userspaceConfig.h"
2  #include "api_map.h"
3
4  void test_cus_main(void *arg)
5  {
6      ... ApiMapInit(arg);
7
8      ... // start add cus code there.
9  }
10
11 void abort(void)
12 {
13     ... printf("abort!!!");
14     ... while (1);
15 }

```

4) 将 main 函数注册到 kernel 中，kernel 会启动一个 task 来调度 main 函数：

```

C test_cus_main.c > ...
1  #include "userspaceConfig.h"
2  #include "api_map.h"
3
4  void test_cus_main(void *arg)
5  {
6      ... ApiMapInit(arg);
7
8      ... // start add cus code there.
9  }
10
11 void abort(void)
12 {
13     ... printf("abort!!!");
14     ... while (1);
15 }
16
17 userSpaceEntry_t userSpaceEntry __attribute__((unused, section(".userSpaceRegTable"))) =
18 {NULL, NULL, (1024 * 10), 30, "userSpaceMain", test_cus_main, NULL };
19

```

5) main 函数注册参数说明：

参数一：通常为 NULL。

参数二：指定 main 函数运行的 task 的栈，可以传入一个 buffer 的指针作为栈，传入 NULL 表示系统自动 malloc 一个空间作为栈使用。

参数三：执行 main 函数的 task 运行时使用的栈的大小，单位为字节。

参数四：main 函数所在 task 的优先级，由于历史原始，该处固定为 30，不要随意更改，实际运行优先级为 150。

参数五：main 函数 task 的名字，字符串常量。

参数六：main 函数。

参数七： 传递一个参数给 `kernel`，通常为定制功能，默认情况下可传入 `NULL`。

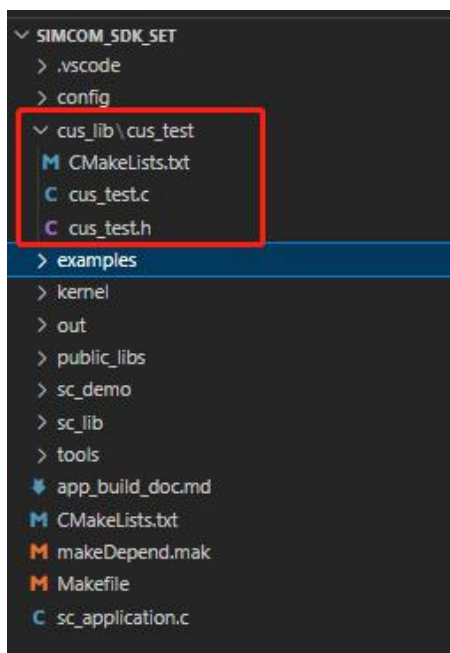
SIMCom  
Confidential

## 9 自定义代码封装成库

### 9.1 新建自定义源代码文件

#### 1). 新建源文件

在 SDK 中创建自定义源代码文件夹及源文件，并在自定义源文件目录下新建子配置文件 CMakeLists.txt，如下图所示中红框所示



#### 2). 写入源码

在对应文件中写入自定义的源码（此处代码仅做示例）

相对路径：cus\_lib\cus\_test\cus\_test.c

```
cus_lib > cus_test > C cus_test.c > print_test()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include "cus_test.h"
5  #include "simcom_debug.h"
6
7  int print_test()
8  {
9      char buf[256] = {0};
10     snprintf(buf, sizeof(buf), "Source file for client code");
11     sAPI_Debug(buf);
12     return 0;
13 }
```

相对路径：cus\_lib\cus\_test\cus\_test.h

```
cus_lib > cus_test > C cus_test.h > ...
1  #ifndef __CUS_TEST_H__
2  #define __CUS_TEST_H__
3
4  int print_test();
5
6  #endif
```

### 3).编写子配置文件 CMakeLists.txt

相对路径: cus\_lib\cus\_test\CMakeLists.txt

```
cus_lib > cus_test > M CMakeLists.txt
1  cmake_minimum_required(VERSION 3.10)
2
3  AUX_SOURCE_DIRECTORY(./ cus_lib_src)
4
5  # Add the static library
6  add_library(cus_test STATIC ${cus_lib_src})
7  target_compile_options(cus_test PUBLIC -Wall)
8  target_include_directories(cus_test PUBLIC
9  .... ./
10  ... ${CMAKE_SOURCE_DIR}/sc_lib/inc
11  ... ${CMAKE_SOURCE_DIR}/sc_lib/${SCMODULE}/inc
12  )
13
14  # SET(LIBRARY_OUTPUT_PATH ${CMAKE_CURRENT_SOURCE_DIR}/../lib)
15  SET(LIBRARY_OUTPUT_PATH ${CMAKE_SOURCE_DIR}/cus_lib/lib)
16
```

如上图所示, 这段代码 CMakeLists.txt 文件的示例, 它描述了一个使用 CMake 构建的项目的配置和构建过程。下面将解释每个部分的含义:

#### 1.cmake\_minimum\_required(VERSION 3.10)

这一行指定了项目所需的最低 CMake 版本。在这里, 指定的最低版本为 3.10。这意味着如果使用低于 3.10 版本的 CMake 来构建该项目, 将会收到版本不兼容的警告或错误。

#### 2.AUX\_SOURCE\_DIRECTORY(./ cus\_lib\_src)

这一行指示 CMake 在当前目录下查找所有源文件, 并将它们存储到 cus\_lib\_src 变量中。这个命令通常用于自动发现当前目录下的所有源文件, 以便将它们添加到项目中。

#### 3.add\_library(cus\_test STATIC \${cus\_lib\_src})

这一行定义了一个静态库 cus\_test, 并将 cus\_lib\_src 中列出的源文件编译成这个静态库。这个静态库将用于链接到其他可执行文件或库中。

#### 4.target\_compile\_options(cus\_test PUBLIC -Wall)

这一行指定了编译 cus\_test 库时的选项, 其中 -Wall 表示开启所有警告。这样可以让编译器在编译代码时输出更多的警告信息, 帮助开发者尽早发现潜在的问题。

#### 5.target\_include\_directories(cus\_test PUBLIC ...)

这部分指定了 cus\_test 库的公共包含目录, 使得在编译时可以找到所需的头文件。

#### 6.SET(LIBRARY\_OUTPUT\_PATH \${CMAKE\_SOURCE\_DIR}/cus\_lib/lib)

这一行设置了生成的静态库 cus\_test 的输出路径。在这里, 将生成的库文件放置在 cmake 源目录下的 cus\_lib/lib 目录中。

总的来说, 这个 CMakeLists.txt 文件配置了一个名为 cus\_test 的静态库项目, 并指定了它的源文件、编译选项、包含目录和输出路径。这些配置将由 CMake 工具自动解析并生成适用于具体平台和编译器的构建文件, 下附源码。



```
cmake_minimum_required(VERSION 3.10)

AUX_SOURCE_DIRECTORY(./ cus_lib_src)

# Add the static library
add_library(cus_test STATIC ${cus_lib_src})
target_compile_options(cus_test PUBLIC -Wall)
target_include_directories(cus_test PUBLIC
    ./
    ${CMAKE_SOURCE_DIR}/sc_lib/inc
    ${CMAKE_SOURCE_DIR}/sc_lib/${SCMODULE}/inc
)

# SET(LIBRARY_OUTPUT_PATH ${CMAKE_CURRENT_SOURCE_DIR}/../lib)
SET(LIBRARY_OUTPUT_PATH ${CMAKE_SOURCE_DIR}/cus_lib/lib)
```

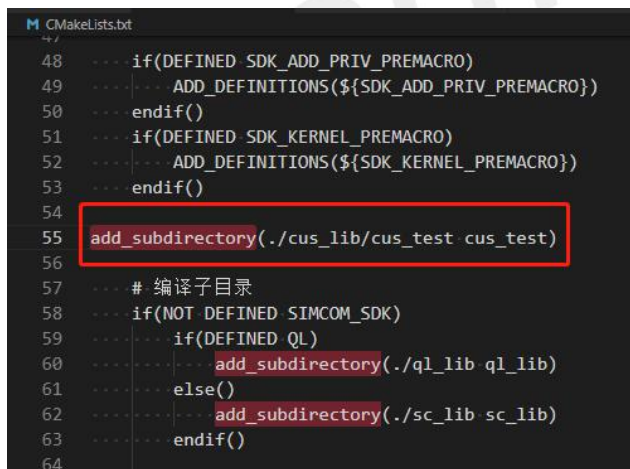
## 9.2 修改根目录 CMakeLists.txt 配置文件

该过程操作文件为 SDK 根目录下 CMakeLists.txt

### 1).编译子目录

这行命令使用 `add_subdirectory` 将位于 `./cus_lib/cus_test` 目录下的子目录 `cus_test` 添加到当前的 `CMakeLists.txt` 文件中。

这意味着在构建过程中，CMake 将进入该子目录，并在那里执行另一个 `CMakeLists.txt` 文件（9.1 中编写的子配置文件 `CMakeLists.txt`），以构建名为 `cus_test` 的目标。



```
M CMakeLists.txt
48 ---if(DEFINED SDK_ADD_PRIV_PREMACRO)
49 ---... ADD_DEFINITIONS(${SDK_ADD_PRIV_PREMACRO})
50 ---endif()
51 ---if(DEFINED SDK_KERNEL_PREMACRO)
52 ---... ADD_DEFINITIONS(${SDK_KERNEL_PREMACRO})
53 ---endif()
54
55 add_subdirectory(./cus_lib/cus_test cus_test)
56
57 ---# 编译子目录
58 ---if(NOT DEFINED SIMCOM_SDK)
59 ---...if(DEFINED QL)
60 ---...add_subdirectory(./ql_lib ql_lib)
61 ---...else()
62 ---...add_subdirectory(./sc_lib sc_lib)
63 ---...endif()
64
```



## 2).链接库

这行命令使用 `target_link_libraries` 将名为 `cus_test` 的库链接到名为 `userspace` 的目标中。

具体来说,它将 `cus_test` 库添加为 `userspace` 目标的依赖项,以便在构建 `userspace` 时,链接器可以将 `cus_test` 库的符号信息合并到最终的可执行文件或库文件中。

请注意,这里使用了 `PRIVATE` 关键字,这意味着 `cus_test` 库是 `userspace` 目标的私有依赖项,只会在链接 `userspace` 时使用,而不会被其他使用了 `userspace` 的目标所链接。如果你想将 `cus_test` 库作为公共依赖项添加到多个目标中,可以使用 `PUBLIC` 关键字。例如:

```
target_link_libraries(target1 PUBLIC cus_test)
```

```
target_link_libraries(target2 PUBLIC cus_test)
```

这会将 `cus_test` 库作为名为 `target1` 和 `target2` 的两个目标的公共依赖项添加到项目中。

```

194 ... add_executable(userspace ${app_src} ${TARGET_OBJECTS:third_libs})
195 ... target_compile_options(userspace PUBLIC -Wall)
196 ... set_target_properties(userspace PROPERTIES SUFFIX ".elf")
197 ... set_target_properties(userspace PROPERTIES OUTPUT_NAME ${APP_NAME})
198
199 target_link_libraries(userspace PRIVATE cus_test)
200
201 ... # 连接目标配置
202 ... if(DEFINED QL)
203 ...     if(NOT DEFINED SIMCOM_SDK)
204 ...         target_link_libraries(userspace PRIVATE ql_lib)
205 ...     endif()
206 ... else(DEFINED QL)
207 ...     if(NOT DEFINED SIMCOM_SDK)

```

## 9.3 编译源文件生成 lib 库

### 1).使用 make 指令编译生成 lib 库

如下图使用 `gnumake A7670C_FANS_1606_V603_OPENSdk` 指令编译代码,若上述配置无误,我们会看到编译成功,反之若编译器报错,应根据实际情况处理配置问题。

```

问题 2 输出 调试控制台 终端 注释

Done.
Done.
Calculating total progress weight ...
Done.
Generating download commands ...
Done.
Generating crane firmware image ...
Done.
Generating target release package ...
Done.
Release package generated successfully!
gnumake[1]: Leaving directory `E:/SDK/SIMCOM_SDK_SET/config'

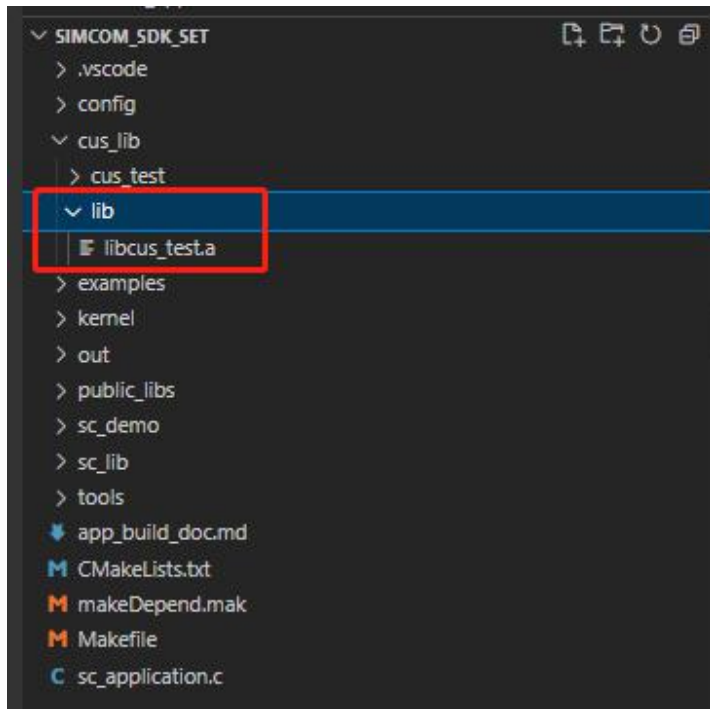
E:\SDK\SIMCOM_SDK_SET>gnumake A7670C_FANS_1606_V603_OPENSdk

```

## 2).生成 lib 库

编译完成后，编译器会自动生成 lib 文件夹，并在 lib 文件夹中自动创建以“lib”开头的静态库文件，如下图这个 lib 文件夹创建的路径由上面 9.1 章节 3 小节中第 6 点描述的

SET(LIBRARY\_OUTPUT\_PATH \${CMAKE\_SOURCE\_DIR}/cus\_lib/lib)指令配置

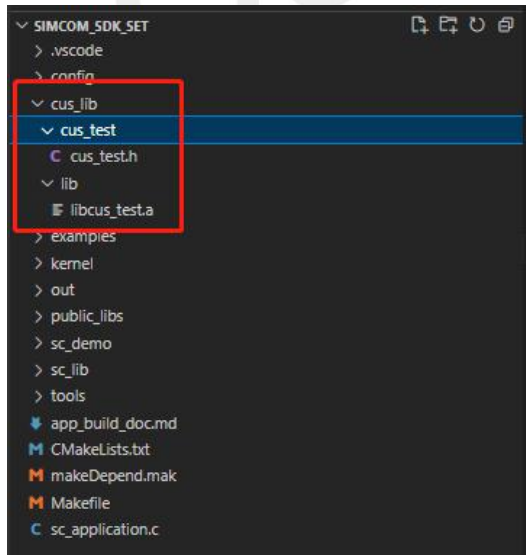


## 9.4 删除源文件并引入编译好的库

### 1).删除源文件

成功生成静态库文件后，即可删除 C 源文件、子目录 CMakeLists.txt 配置文件，以及根目录 CMakeLists.txt 中添加的配置指令，仅保留头文件与生成的 lib 库文件。（删除的文件具体为：上述 9.1 章节中创建的 cus\_test.c 和子配置文件 CMakeLists.txt，以及 9.2 章节中修改的根目录 CMakeLists.txt 配置文件中添加的两条指令）

删除完成后的客户文件夹如下图所示：



## 2).引入已经编好的库

该过程操作文件为 SDK 根目录下 CMakeLists.txt:

在根目录下 CMakeLists.txt 中使用如下指令引入 9.2 章节中编好的 lib 库

```
set_property(TARGET third_libs APPEND PROPERTY IMPORTED_OBJECTS
"${CMAKE_SOURCE_DIR}/cus_lib/lib/libcus_test.a"
)
```

这行命令使用 set\_property 将一个导入的目标对象文件路径添加到名为"third\_libs"的目标中。

具体来说,它将路径\${CMAKE\_SOURCE\_DIR}/cus\_lib/lib/libcus\_test.a的对象文件添加到"third\_libs"目标中。这意味着在构建过程中,"third\_libs"目标将包含来自该对象文件的符号信息,并且将在链接过程中使用。该命令用于引入已经编译好的第三方库,并将其与项目一起链接。但需要确保目标文件和库文件的路径是正确的。

```
102
103 ... # 外部库列表
104 ... add_library(third_libs OBJECT IMPORTED GLOBAL)
105
106 ... if(DEFINED THIRD_LIB)
107 ... set_property(TARGET third_libs APPEND PROPERTY IMPORTED_OBJECTS
108 ... "${THIRD_LIB}")
109 ... )
110 ... endif()
111
112 ... # 引入已经编译好的库
113 ... set_property(TARGET third_libs APPEND PROPERTY IMPORTED_OBJECTS
114 ... "${CMAKE_SOURCE_DIR}/cus_lib/lib/libcus_test.a")
115 ... )
116
117 ... if(DEFINED SIMCOM_SDK)
118 ... if(DEFINED QL)
119 ... set_property(TARGET third_libs APPEND PROPERTY IMPORTED_OBJECTS
120 ... "${CMAKE_SOURCE_DIR}/ql_lib/${SCMODULE}/lib/libql_lib.a")
121 ... )
122 ... else(DEFINED QL)
123 ... set_property(TARGET third_libs APPEND PROPERTY IMPORTED_OBJECTS
124 ... "${CMAKE_SOURCE_DIR}/sc_lib/${SCMODULE}/lib/libsc_lib.a")
```

## 4).再次编译代码

再次编译代码,若上述配置均无误,我们应当看到编译成功,到这一步为止,我们就完成了自定义代码封装成库了。

```
问题 2 输出 调试控制台 终端 注释
Done.
Done.
Done.
Calculating total progress weight ...
Done.
Generating download commands ...
Done.
Generating crane firmware image ...
Done.
Generating target release package ...
Done.
Release package generated successfully!
gnumake[1]: Leaving directory `E:/SDK/SIMCOM_SDK_SET/config'
E:\SDK\SIMCOM_SDK_SET>gnumake A7670C_FANS_1606_V603_OPENSdk
```

## 9.5 编译测试

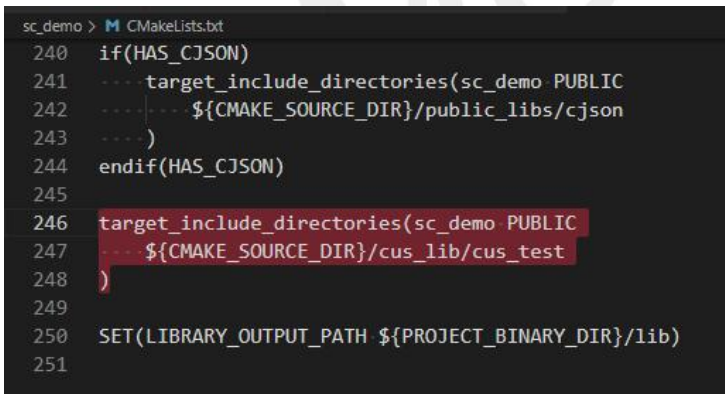
### 1).添加头文件目录

该过程操作文件为 `sc_demo` 目录下 `CMakeLists.txt`:

```
target_include_directories(sc_demo PUBLIC
    ${CMAKE_SOURCE_DIR}/cus_lib/cus_test
)
```

这行命令使用 `target_include_directories` 将指定目录添加到名为 `sc_demo` 的目标中的头文件搜索路径中。具体来说，它将 `${CMAKE_SOURCE_DIR}/cus_lib/cus_test` 目录添加到 `sc_demo` 目标的头文件搜索路径中。这意味着，在编译 `sc_demo` 目标时，编译器将在该目录中搜索所需的头文件。使用 `PUBLIC` 关键字表示这个目录是 `sc_demo` 目标的公共头文件搜索路径，其他需要链接 `sc_demo` 的目标也可以使用这个路径。

注意：这里仅示例将 `cus_test` 目录添加到 `sc_demo` 的目标中的头文件搜索路径中，具体使用应根据实际需求来包含头文件搜索路径。

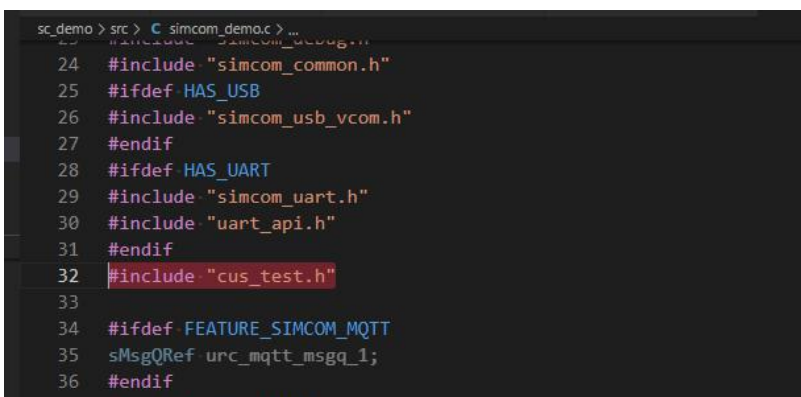


```
sc_demo > CMakeLists.txt
240 if(HAS_CJSON)
241     target_include_directories(sc_demo PUBLIC
242         ${CMAKE_SOURCE_DIR}/public_libs/cjson
243     )
244 endif(HAS_CJSON)
245
246 target_include_directories(sc_demo PUBLIC
247     ${CMAKE_SOURCE_DIR}/cus_lib/cus_test
248 )
249
250 SET(LIBRARY_OUTPUT_PATH ${PROJECT_BINARY_DIR}/lib)
251
```

### 2).编译测试代码

操作文件路径: `sc_demo\src\simcom_demo.c`

在我们的应用程序中添加我们的自定义函数接口（参考 9.1 章节写入的源文件，此处为示例代码），编译成功后，烧录新编好的版本至模块中。



```
sc_demo > src > C simcom_demo.c > ...
24 #include "simcom_common.h"
25 #ifdef HAS_USB
26 #include "simcom_usb_vcom.h"
27 #endif
28 #ifdef HAS_UART
29 #include "simcom_uart.h"
30 #include "uart_api.h"
31 #endif
32 #include "cus_test.h"
33
34 #ifdef FEATURE_SIMCOM_MQTT
35 sMsgQRef urc_mqtt_msgq_1;
36 #endif
37
```



```

sc_demo > src > C simcom_demo.c > sTask_SimcomUIProcessor(void *)
368 /**
369  * @brief SIMCom UI demo processor.
370  * @param arg
371  * @note Please select demo according to CLI.
372  * @retval void
373  */
374 void sTask_SimcomUIProcessor(void *arg)
375 {
376     UINT32 opt = 0;
377     char *note = "Please select an option to test from the items listed below.\n";
378
379     print_test();
380
381     while (1)
382     {

```

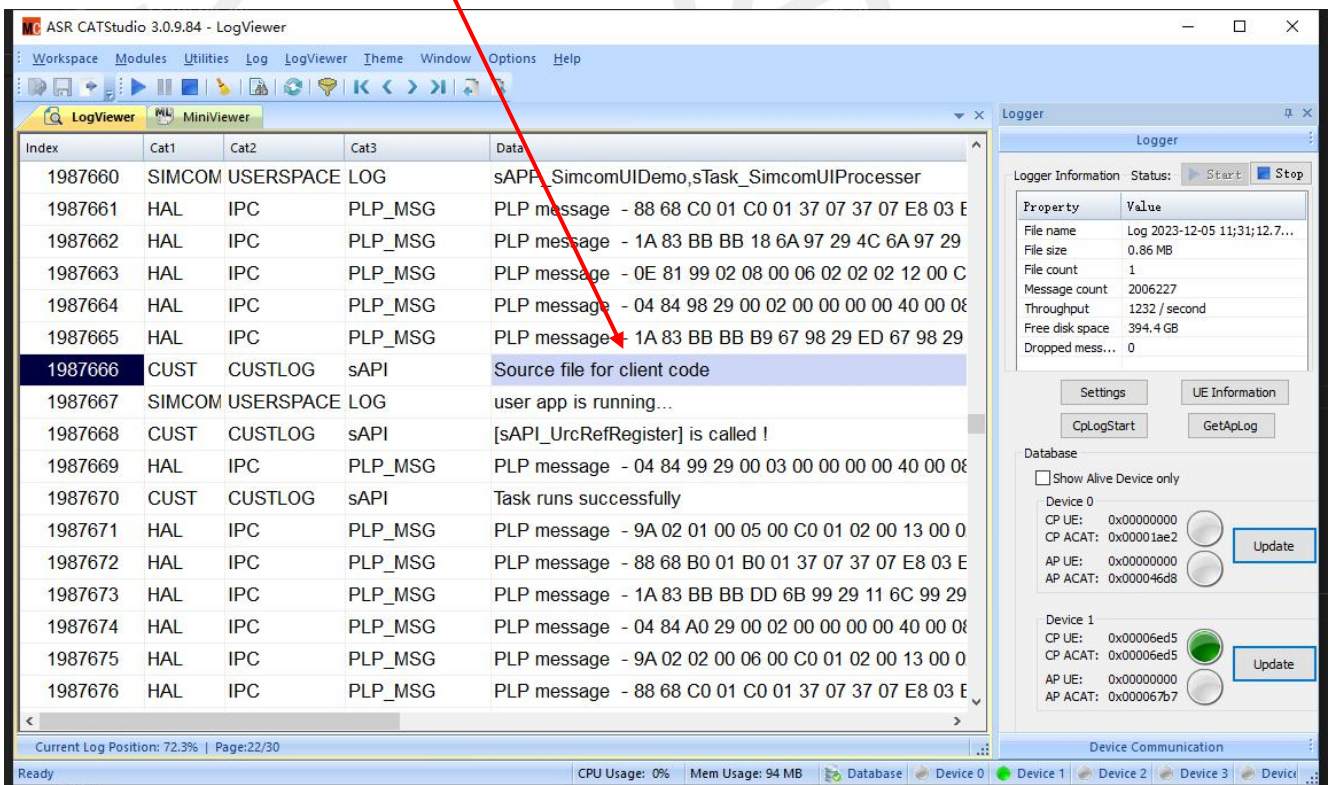
### 3).测试

烧录完成后，开机，可在 CATStudio 中看到我们编写的测试代码中的 log 已经成功被打印出来，说明我们的自定义代码已经可以被正常调用了。

```

cus_lib > cus_test > C cus_test.c > print_test()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include "cus_test.h"
5  #include "simcom_debug.h"
6
7  int print_test()
8  {
9      char buf[256] = {0};
10     snprintf(buf, sizeof(buf), "Source file for client code");
11     sAPI_Debug(buf);
12     return 0;
13 }

```



The screenshot shows the ASR CATStudio 3.0.9.84 - LogViewer interface. The main window displays a log table with columns: Index, Cat1, Cat2, Cat3, and Data. A red arrow points from the 'Source file for client code' log entry in the Data column to the corresponding line in the code snippet above.

Index	Cat1	Cat2	Cat3	Data
1987660	SIMCOM	USERSPACE	LOG	sAPP_SimcomUIDemo,sTask_SimcomUIProcessor
1987661	HAL	IPC	PLP_MSG	PLP message - 88 68 C0 01 C0 01 37 07 37 07 E8 03 E
1987662	HAL	IPC	PLP_MSG	PLP message - 1A 83 BB BB 18 6A 97 29 4C 6A 97 29
1987663	HAL	IPC	PLP_MSG	PLP message - 0E 81 99 02 08 00 06 02 02 02 12 00 C
1987664	HAL	IPC	PLP_MSG	PLP message - 04 84 98 29 00 02 00 00 00 00 40 00 0
1987665	HAL	IPC	PLP_MSG	PLP message - 1A 83 BB BB B9 67 98 29 ED 67 98 29
1987666	CUST	CUSTLOG	sAPI	Source file for client code
1987667	SIMCOM	USERSPACE	LOG	user app is running...
1987668	CUST	CUSTLOG	sAPI	[sAPI_UrcRefRegister] is called !
1987669	HAL	IPC	PLP_MSG	PLP message - 04 84 99 29 00 03 00 00 00 00 40 00 0
1987670	CUST	CUSTLOG	sAPI	Task runs successfully
1987671	HAL	IPC	PLP_MSG	PLP message - 9A 02 01 00 05 00 C0 01 02 00 13 00 0
1987672	HAL	IPC	PLP_MSG	PLP message - 88 68 B0 01 B0 01 37 07 37 07 E8 03 E
1987673	HAL	IPC	PLP_MSG	PLP message - 1A 83 BB BB DD 6B 99 29 11 6C 99 29
1987674	HAL	IPC	PLP_MSG	PLP message - 04 84 A0 29 00 02 00 00 00 00 40 00 0
1987675	HAL	IPC	PLP_MSG	PLP message - 9A 02 02 00 06 00 C0 01 02 00 13 00 0
1987676	HAL	IPC	PLP_MSG	PLP message - 88 68 C0 01 C0 01 37 07 37 07 E8 03 E

The right-hand pane shows the 'Logger' window with 'Logger Information' and 'Database' sections. The 'Logger Information' section includes fields for File name, File size, File count, Message count, Throughput, Free disk space, and Dropped messages. The 'Database' section shows a list of devices with their CP UE, CP ACAT, AP UE, and AP ACAT values, and buttons for 'Update' and 'GetApLog'.