WS63V100 SDK 开发环境搭建

用户指南

文档版本　02

发布日期　2024-07-01

前言

概述

本文档介绍 WS63 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
WS63	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

符号	说明
危险	表示如不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。
注意	表示如不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	2024-07-01	• 更新“2.1 SDK 目录结构介绍”小节内容。 • 更新“2.2.4 添加 bin 文件编译”小节内容。 • 更新“2.2.6 Menuconfig 配置”小节内容。 • 新增“2.2.7 UART 配置方法”小节内容。 • 更新“3.1 建立源码目录”小节内容。
01	2024-04-10	第一次正式版本发布。 • 更新“2.2.2 编译参数详解”小节内容。 • 更新“2.2.4 添加 bin 文件编译”小节内容。 • 更新“2.2.5 Flash 分区表配置”小节内容。
00B03	2024-03-29	• 更新“2.2.2 编译参数详解”小节内容。 • 更新“2.2.4 添加 bin 文件编译”小节内容。

文档版本	发布日期	修改说明
00B02	2024-01-10	更新“1.2.3 安装 Python 环境”小节内容。
00B01	2023-11-27	第一次临时版本发布。

目 录

1 开发环境搭建

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。

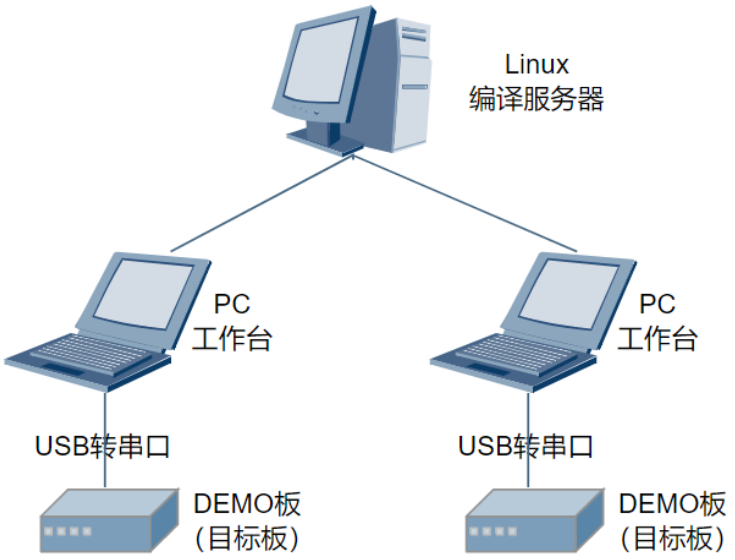
- 工作台

  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 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 20.04 及以上版本，Shell 使用 bash ，SDK 使用 Cmake 编译（3.14.1 以上），编译工具还包括 Python（3.8.0 以上）等。

1.2.1 配置 Shell

配置默认使用 bash。打开 Linux 终端，执行命令“sudo dpkg-reconfigure dash”，选择 no。

1.2.2 安装 Cmake

打开 Linux 终端，执行命令“sudo apt install cmake”，完成 Cmake 的安装。

1.2.3 安装 Python 环境

步骤 1 打开 Linux 终端，输入命令“python3 -V” ，查看 Python 版本号，推荐 python3.8.0 以上版本。

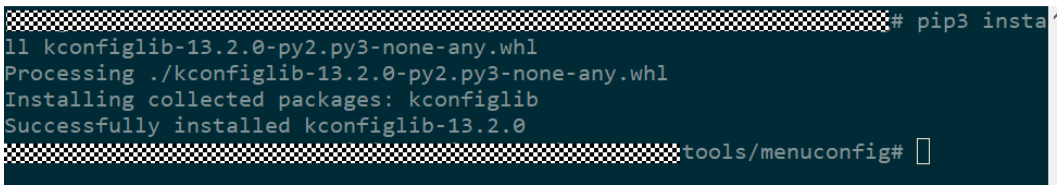
步骤 2 如果 Python 版本太低，请使用命令“sudo apt-get update”更新系统到最新，或通过命令“sudo apt-get install python3 -y”安装 Python3（需 root/sudo 权限安装），安装后再次确认 Python 版本。

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 14.1.0+，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-14.1.0-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 1-2 所示。

图1-2 安装 Kconfiglib 组件包完成示例



步骤 5 安装升级文件签名依赖的 Python 组件包。

安装 pycparser：

从“https://pypi.org/project/pycparser/”下载.whl 文件（例如：pycparser-2.21-py2.py3-none-any.whl）后，使用“pip3 install pycparser-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 pycparser-2.21”。

----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。

2 编译 SDK

2.1 SDK 目录结构介绍

SDK 根目录结构如表 2-1 所示。

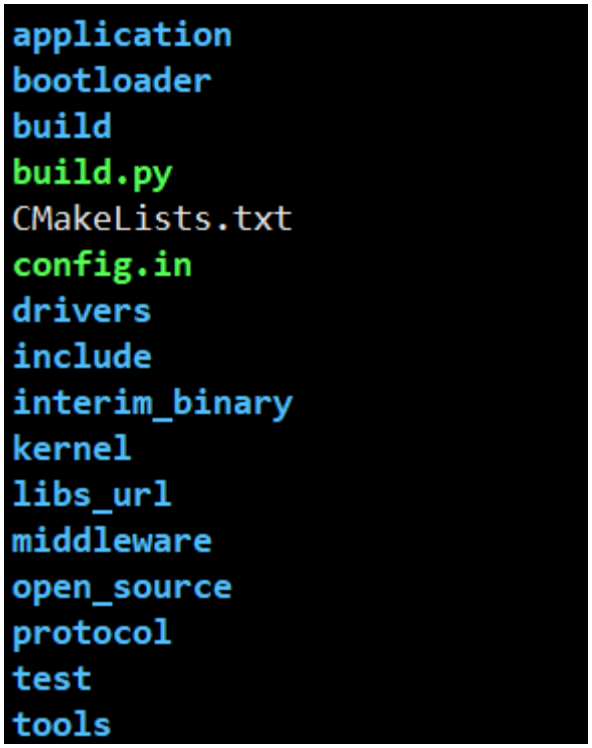
表2-1 SDK 根目录

目录	说明
application	应用层代码（其中包含 demo 程序为参考示例）。
bootloader	boot(Flashboot/SSB)代码。
build	SDK 构建所需的脚本、配置文件。
build.py	编译入口脚本。
CMakeLists.txt	Cmake 工程顶层“CMakeLists.txt”文件。
config.in	Kconfig 配置文件。
drivers	驱动代码。
include	API 头文件存放目录。
interim_binary	库存放目录。
kernel	内核代码和 OS 接口适配层代码。

目录	说明
libs_url	库文件。
middleware	中间件代码。
open_source	开源代码。
protocol	WiFi、BT、Radar 等组件代码。
test	testsuite 代码。
tools	包含编译工具链（包括 linux 和 windows）、镜像打包脚本、NV 制作工具和签名脚本等。
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）。

解压缩 SDK 后的根目录，如图 2-1 所示。注：上表所述的 output 目录是编译后生成的。

图2-1 解压缩 SDK 示例



2.2 编译 SDK（Cmake）

2.2.1 编译方法

根目录下执行“python3 build.py”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	python3 build.py ws63-liteos-app	启动 ws63-liteos-app 目标的增量编译。
-c	python3 build.py -c ws63-liteos-app	启动 ws63-liteos-app 目标的全量编译。
menuconfig	python3 build.py ws63-liteos-app menuconfig	启动 ws63-liteos-app 目标的 menuconfig 图形配置界面。

表2-3 编译目标介绍

编译目标	说明
python3 build.py -c ws63-liteos-app	app 版本编译目标（自动包含 flashboot 编译）
python3 build.py -c ws63-flashboot	flashboot 镜像编译目标
python3 build.py -c ws63-liteos-xts	openharmony xts 认证版本编译目标（详情请参考“鸿蒙 XTS 认证指导书”）
python3 build.py -c ws63-liteos-app-iot	Harmony connect 版本编译目标（详情请参考“HiLink 编译使用指南”）
python3 build.py -c ws63-liteos-hilink	Harmony connect 独立升级版本编译目标（详情请参考“HiLink 编译使用指南”）

编译得到的烧录镜像在“output/ws63/fwpkg/ws63-liteos-app”目录下（如表 2-4 所示）。

表2-4 烧录镜像

文件名	说明
ws63-liteos-app_all.fwpkg	空片烧录时，需要烧录此文件。包含了所有的需要升级的内容。包含：root_loaderboot_sign.bin、root_params.bin、flashboot_sign.bin、ws63_all_nv.bin、ws63-liteos-app-sign.bin。 各文件介绍如下： root_loaderboot_sign.bin：loaderboot 的镜像文件。升级开始时，芯片中固化的 romboot 会接收此镜像文件，加载到内存并运行，loadboot 负责接收后续的镜像文件。注：此镜像只在升级阶段放在 RAM 中运行，并不存放在 flash 中。 root_params.bin：flash 分区信息的镜像文件。分区信息供 romboot、loaderboot 和 flashboot 使用。 flashboot_sign.bin：flashboot 的镜像文件。 ws63_all_nv.bin：参数区的镜像文件。 ws63-liteos-app-sign.bin：版本镜像文件。
ws63-liteos-app_load_only.fwpkg	版本升级打包文件，包含：root_loaderboot_sign.bin、ws63-liteos-app-sign.bin。不包含 flashboot 相关内容。 当芯片烧录过“ws63-liteos-app_all.fwpkg”镜像后，如果后续修改不涉及 root_params、flash_boot、nv 的修改，则可以用此文件升级。

注：编译得到的中间文件在“output/ws63/acore/ws63-liteos-app”目录下。

2.2.2 编译参数详解

编译命令接收参数及解释如表 2-5 所示。

表2-5 编译参数信息表

参数	参数信息
-c	clean 后编译
-j	-j<num>，以 num 线程数执行编译，如-j16，-j8

参数	参数信息
	默认最大线程
-def=	-def=XXX,YYY,ZZZ=x,... 向本次编译 target 中添加 XXX、YYY、ZZZ=x 编译宏 可使用-def=-:XXX 来屏蔽 XXX 宏 可使用-def=-:ZZZ=x 来添加或者修改 ZZZ 宏
-component=	-component=XXX,YYY,... 仅编译 XXX,YYY 组件
-ninja	使用 ninja 生成中间文件，默认使用 Unix makefile
-[release / debug]	release: 在生成反汇编文件时节省时间 debug: 在生成反汇编文件时信息更加全面但也更耗时 默认为 debug
-dump	编译时在终端输出 target 的所有参数列表 -- 包括编译宏、组件、编译选项等
-nhso	不更新 HSO 数据库
-out_libs	-out_libs=file_path，不再链接成 elf，转而将所有.a 打包成一个大的.a
others	作为匹配编译 target_names 的关键字

2.2.3 编译选项详解

WS63 在不同目录下的.py 文件下配置编译选项，如表 2-6 所示。

表2-6 WS63 通用组件编译选项

编译选项类型	说明	内容	对应文件控制路径
common_cflags	基础编译选项	-std=gnu99 -Wall -Werror -Wextra -Winit-self -Wpointer-arith -Wstrict-prototypes -Wno-type-limits -fno-strict-aliasing -Os -fno-unwind-tables	\sdk\build\config\target_config\common_config.py
riscv31	芯片类型编	-ffreestanding -fdata-sections -Wno-implicit-fallthrough -	\sdk\build\config\target_config\comm

编译选项类型	说明	内容	对应文件控制路径
	译选项	ffunction-sections -nostdlib -pipe -fno-tree-scev-cprop -fno-common -mpush-pop -msmall-data-limit=0 -fno-ipa-ra -Wtrampolines -Wlogical-op -Wjump-misses-init -Wa,-enable-c-lbu-sb -Wa,-enable-c-lhu-sh -fimm-compare -femit-muliadd -fmerge-immshf -femit-uxtb-uxth -femit-lli -femit-clz -fldm-stm-optimize -g	on_config.py
fp_flags	硬浮点编译选项	-march=rv32imfc -mabi=ilp32f	\sdk\build\config\target_config\ws63\target_config.py
codesize_flags	codesize 优化选项	--short-enums -madjust-regorder -madjust-const-cost -freorder-commu-args -fimm-compare-expand -frmv-str-zero -mfp-const-opt -mswitch-jump-table -frtl-sequence-abstract -frtl-hoist-sink -fsafe-alias-multipointer -finline-optimize-size -fmuliadd-expand -mlli-expand -Wa,-mcjal-expand -foptimize-reg-alloc -fsplit-multi-zero-assignments -floop-optimize-size -Wa,-mlli-relax -mpattern-abstract -foptimize-pro-and-epilogue	\sdk\build\config\target_config\ws63\target_config.py

其中，编译选项的详细说明如表 2-7 所示。

表2-7 编译选项详细说明

选项	说明
-std=gnu99	使用 ISO C99 标准再加上 GNU 的扩展
-Wall	选项意思是编译后显示所有警告
-Werror	用于将所有警告升级成错误
-Wextra	用于开启额外的警告信息（-Wall 的补充）
-Winit-self	警告使用自己初始化的未初始化变量
-Wpointer-arith	警告任何取决于“功能类型”或“功能类型”的大小 void

选项	说明
-Wstrict-prototypes	警告如果一个函数被声明或定义而没有指定参数类型
-Wno-type-limits	屏蔽由于数据类型范围有限而导致比较始终为真或始终为 false 的告警
-fno-strict-aliasing	禁用 strict-aliasing 优化规则：不同类型的指针绝对不会指向同一块内存区域
-Os	专门优化目标文件大小，执行所有的不增加目标文件大小的-O2 优化选项，同时-Os 还会执行更加优化程序的选项
-fno-unwind-tables	删除 unwind 调试信息
-ffreestanding	断言编译发生在独立环境中
-fdata-sections	将每个数据放入自己的部分（仅限 ELF）
-Wno-implicit-fallthrough	可忽略编译时 switch-case 中缺少 break 的错误
-ffunction-sections	将每个函数放在自己的节中（仅限 ELF）
-nostdlib	关闭默认头文件与库文件搜索目录
-pipe	编译过程中使用管道，借助 GCC 的管道功能来提高编译速度
-fno-tree-scev-cprop	禁用标量演化信息进行复写传递，代码空间优化相关
-fno-common	可以把静态库中的没有初始化的全局变量从弱符号变成强符号，当所有静态库链接成可执行文件，如果同时有两个以上“重名强符号”，链接器会报错。
-mpush-pop	CodeSize 优化，改编译选项需要 CPU 版本支持 push/pop/popret/lwm/swm 等指令
-msmall-data-limit=0	CodeSize 优化，改编译选项需要 CPU 版本支持 push/pop/popret/lwm/swm 等指令
-fno-ipa-ra	禁用编译器针对叶子函数的编译优化（编译选项中添加了-O2 优化选项时-fipa-ra 参数导致）
-Wtrampolines	该选项用于检查代码中是否包含内嵌函数,gcc 对内嵌函数有个

选项	说明
	专门的称呼:trampoline
-Wlogical-op	当逻辑操作结果似乎总为真或假时给出警告
-Wjump-misses-init	switch 或者 goto 语句后声明并且初始化变量，进行告警。
-Wa,-enable-c-lbu-sb	汇编器优化，默认禁用。如果启用此优化，汇编器将使用压缩的 lbu & sb 来替换 lbu & sb。
-Wa,-enable-c-lhu-sh	汇编器优化，默认禁用。如果启用此优化，汇编器将使用压缩的 lhu & sh 来替换 lhu & sh
-fimm-compare	代码大小的优化，可以将非零立即比较的两条指令（li，bxx）合并到一条指令（bxxi）
-femit-muliadd	CodeSize 优化，可以将多条加树指令合并为一条指令
-fmerge-immshf	CodeSize 优化，可以将立即移位合并为一条指令。组合仅在 -O1 以上的选项中生效
-femit-uxtb-uxth	CodeSize 优化，将无符号扩展字节和无符号扩展半字优化为 uxtb、uxth(16 字节)。组合仅在 -O1 以上的选项中
-femit-lli	使用 48 位 l.li 指令代替 64 位指令 lui + addi 进行 32 位长立即加载。此优化与 insn 组合结合使用
-femit-clz	支持 CLZ 指令，所有__builtin_clz 函数的调用都会优化为 CLZ 指令。
-fldm-stm-optimize	启用用 ldmia/stmia 替换连续 WORD 加载/存储的优化，默认禁用。
-g	调试编译选项，对于可执行二进制文件可通过以下方法确定是否包含调试信息
-mabi=ilp32f	支持硬浮点（指定整数和浮点调用约定）
-march=rv32imfc	支持硬浮点（为给定的 RISC-V ISA 生成代码）
--short-enums	CodeSize 优化，enum 类型等于大小足够的最小整数类型
-madjust-regorder	寄存器分配优化-寄存器分配顺序调整优化

选项	说明
-madjust-const-cost	立即数重复加载优化
-freorder-commu-args	浮点运算可交换操作数优化
-fimm-compare-expand	扩展指令常量比较指令优化
-frmv-str-zero	rodata 段常量字符串对齐优化
-mfp-const-opt	浮点常量加载优化
-mswitch-jump-table	switch case 跳转表优化
-frtl-sequence-abstract	函数内过程优化
-frtl-hoist-sink	代码移动优化
-fsafe-alias-multipointer	多级指针重复加载优化
-finline-optimize-size	inline 内联代价模型优化
-fmuliadd-expand	扩展指令乘加指令优化（muliadd 优化）
-mlli-expand	扩展指令 l.li 指令优化
-Wa,-mcjal-expand	汇编器上 jal 压缩指令优化
-foptimize-reg-alloc	寄存器分配优化-寄存器分配优先级调整优化
-fsplit-multi-zero-assignments	连续赋零值优化
-floop-optimize-size	循环结构优化
-Wa,-mlli-relax	高频立即数加载优化（汇编器和链接器协同优化）
-mpattern-abstract	过程间抽象优化(根据已知 pattern 抽象优化)
-foptimize-pro-and-epilogue	函数 prologue 和 epilogue 优化

2.2.4 添加 bin 文件编译

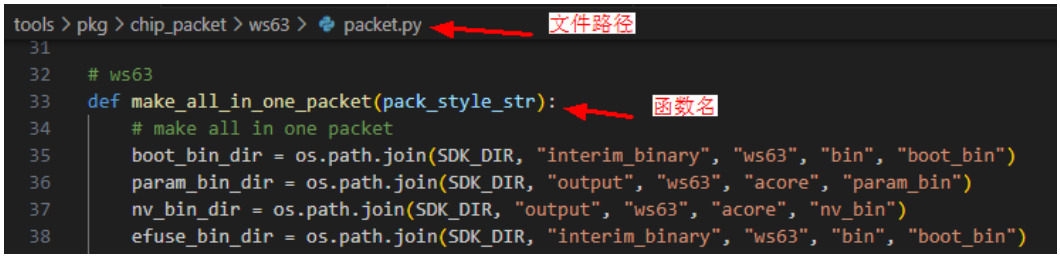
当前 ws63 编译 ws63-liteos-app_all.fwpkg 时，默认编译 root_loaderboot_sign.bin、root_params.bin、flashboot_sign.bin、ws63_all_nv.bin、ws63-liteos-app-sign.bin，文件介绍如表 2-4 所示。

若需编译其他 bin 文件，可按如下步骤添加：

步骤 1 在根路径下打开/tools/pkg/chip_packet/ws63/packet.py 文件

步骤 2 在 make_all_in_one_packet 函数中添加代码，如图 2-2 所示

图2-2 文件路径和函数

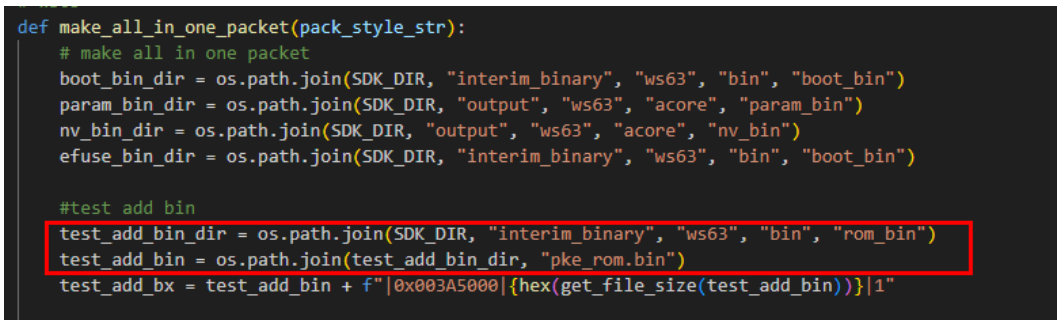


步骤 3 在函数中添加 bin 文件路径

其中每个拼接字符串代表一层目录（文件夹名和文件名）

最终拼接完成的 test_add_bin 实际值：
sdk\interim_binary\ws63\bin\rom_bin\pke_rom.bin

图2-3 bin 文件路径图



步骤 4 设置编译参数，参数之间用"|"分割

图2-4 bin 文件编译参数

```
def make_all_in_one_packet(pack_style_str):
    # make all in one packet
    boot_bin_dir = os.path.join(SDK_DIR, "interim_binary", "ws63", "bin", "boot_bin")
    param_bin_dir = os.path.join(SDK_DIR, "output", "ws63", "acore", "param_bin")
    nv_bin_dir = os.path.join(SDK_DIR, "output", "ws63", "acore", "nv_bin")
    efuse_bin_dir = os.path.join(SDK_DIR, "interim_binary", "ws63", "bin", "boot_bin")

    #test add bin
    test_add_bin_dir = os.path.join(SDK_DIR, "interim_binary", "ws63", "bin", "rom_bin")
    test_add_bin = os.path.join(test_add_bin_dir, "pke_rom.bin")
    test_add_bx = test_add_bin + f"|0x003A5000|{hex(get_file_size(test_add_bin))}|1"
                                  ①          ②                                ③
```

1. 烧录位置，单板剩余地址可在 sdk\build\config\target_config\ws63\param_sector\param_sector.json 文件中查看

2. 占用空间大小

3. 文件类型，0 代表 loader，1 代表普通烧写文件，3 是 efuse，4 是 otp

图2-5 单板剩余地址

```
31          "param_info" : [
32              ["0x00", "0x00002000", "0x00008000"],
33              ["0x01", "0x0000A000", "0x00010300"],
34              ["0x02", "0x0001B000", "0x00010300"],
35              ["0x03", "0x00000000", "0x00000080"],
36              ["0x04", "0x00000080", "0x00000700"],
37              ["0x05", "0x00000780", "0x00000080"],
38              ["0x06", "0x00000800", "0x00000700"],
39              ["0x07", "0x00001000", "0x00000800"],
40              ["0x10", "0x0002C000", "0x00008000"],
41              ["0x11", "0x00034000", "0x00001000"],
42              ["0x20", "0x00035000", "0x00200000"],
43              ["0x21", "0x00235000", "0x00170000"],
44
45              ["0x31", "0x003A5000", "0x00000000"],
46              ["0x32", "0x003A5000", "0x00000000"],
47              ["0x33", "0x003A5000", "0x00000000"],
48              ["0x34", "0x003A5000", "0x0005B000"]
49          ]
50      },
51      "Output_file_prefix":"AIoT"
52  }
```

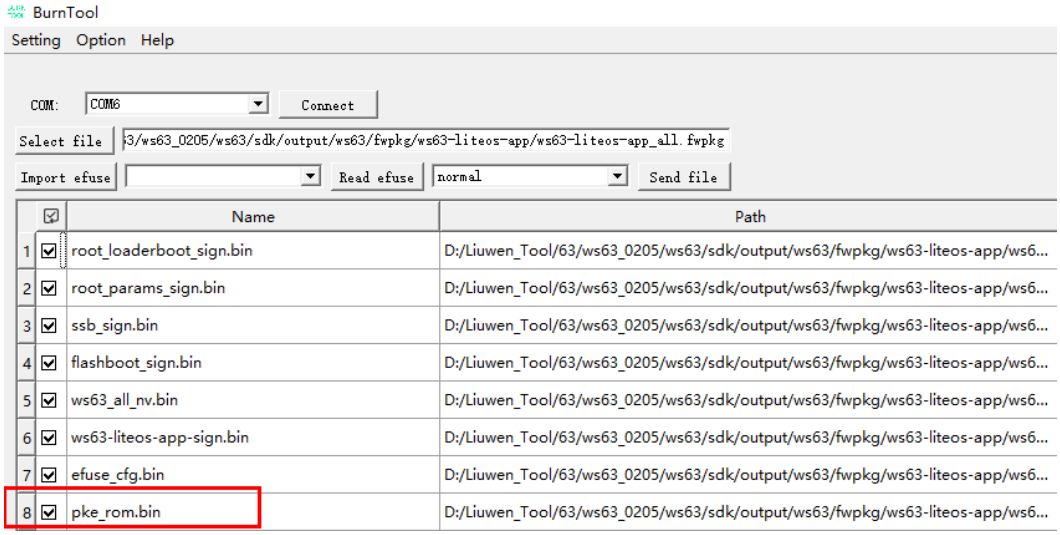
步骤 5 在函数末尾，将设置好编译参数和路径的变量添加到编译列表

图2-6 添加路径到编译列表

```
201     packet_post_agvs = list()
202     packet_post_agvs.append(loadboot_bx)
203     packet_post_agvs.append(params_bx)
204     packet_post_agvs.append(ssb_bx)
205     packet_post_agvs.append(flashboot_bx)
206     packet_post_agvs.append(nv_bx)
207     packet_post_agvs.append(app_bx)
208     packet_post_agvs.append(efuse_bx)
209     packet_post_agvs.append(test_add_bx)
210     fpga_fwpkg = os.path.join(fwpkg_outdir, f"{pack_style_str}_all.fwpkg")
211     packet_bin(fpga_fwpkg, packet_post_agvs)
```

步骤 6 编译结果展示

图2-7 编译结果



----结束

说明

如果新增的 bin 文件有通过 OTA 升级的需求，请参见《WS63V100 FOTA 开发指南》中对应内容，适配新增 bin 文件的 OTA 升级支持。

2.2.5 Flash 分区表配置

分区表配置文件路径
sdk\build\config\target_config\ws63\param_sector\param_sector.json

表2-8 分区表说明

分区表 ID	起始地址	分区长度	描述
0x00	0x00002000	0x00008000	ssb(全称：secure stage boot)：一级 boot
0x01	0x0000A000	0x00010300	FlashBoot：二级 boot

分区表 ID	起始地址	分区长度	描述
0x02	0x0001B000	0x00010300	FlashBoot backup：二级 boot 备份区
0x03	0x00000000	0x00000080	root public key：根公钥
0x04	0x00000080	0x00000700	param area：参数区
0x05	0x00000780	0x00000080	root public key backup：根公钥备份区
0x06	0x00000800	0x00000700	param area back up：参数备份区
0x07	0x00001000	0x00000800	CFCT：芯片特性分区
0x10	0x0002C000	0x00008000	NV DATA：NV 数据存放区
0x11	0x00034000	0x00001000	crash info：异常信息存储区
0x20	0x00035000	0x00200000	imageA：app 镜像区
0x21	0x00235000	0x00170000	imageB：压缩分区/OTA 升级分区/产测镜像分区/b 面分区
0x30	0x003A5000	0x00000000	rsv1:预留分区 1
0x31	0x003A5000	0x00001000	rsv2:预留分区 2
0x32	0x003A6000	0x00008000	rsv3:预留分区 3
0x33	0x003A6000	0x0005A000	rsv4:预留分区 4

说明

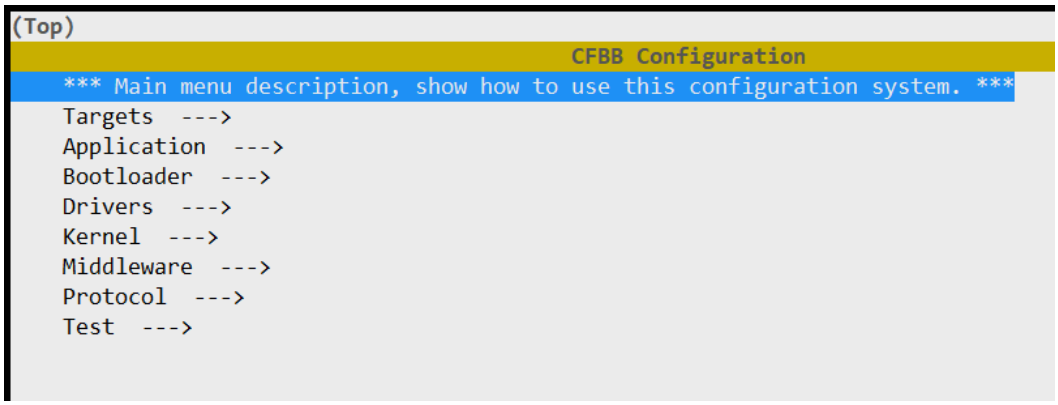
分区表 Id 限制 16 个分区数量，默认 Flash 共 4M 大小，预留 4 个分区共 364K，可通过 uapi_partition_get_info 接口传入分区 Id 获取对应地址和长度

2.2.6 Menuconfig 配置

运行“python3 build.py -c ws63-liteos-app 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，如图 2-8 所示。

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“python3 build.py -c ws63-liteos-app menuconfig”更改配置。

图2-8 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 2-9 所示，在 Menuconfig 界面中可输入快捷键进行配置。

表2-9 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 2-9 所示。

图2-9 Menuconfig 命令帮助栏

表2-10 Menuconfig 菜单项说明

菜单	说明
Targets	编译 target 相关配置。
Application	应用相关配置（主要是 sample 相关）。
Bootloader	boot 相关配置。
Drivers	外设驱动相关配置和板级相关配置。
Kernel	内核相关配置。
Middleware	中间件（NV、FOTA、AT、DFX、PM 等）相关配置。
Protocol	WiFi、蓝牙相关配置。
Test	testsuite 相关配置。

2.2.7 UART 配置方法

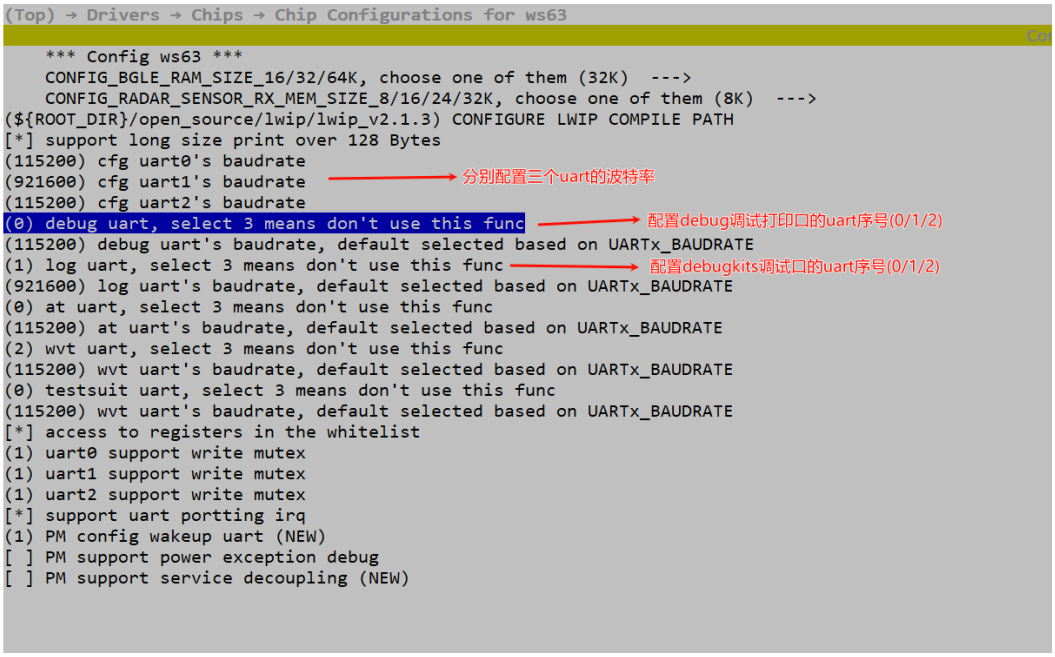
WS63 芯片总共有 3 个 UART，SDK 默认配置如下。

uart 序号	波特率	用途
0	115200	debug 调试打印/烧录/AT。
1	921600	debugkits 调试口。
2	115200	空闲，暂未使用。

烧录功能，固定用 uart0，不可更改。

debug 调试打印/AT/debugkits 调试口，可通过 menuconfig 进行配置，以便适应不同硬件板级 uart 连接，波特率也可以通过 menuconfig 进行定制。

menuconfig 的配置路径为 Drivers->Chips->Chip Configurations for ws63：



须知

1.debugkits 调试口不能与其他功能共用同一个 UART 口，必须独占。

2.UART 波特率建议配置典型值，如 115200/921600/1M 等，考虑到兼容性，不建议配置不常用的特殊值，比如 115623 此类波特率值。

3.修改 UART 序号请慎重，必须要与板级硬件工程师确认 uart 硬件连接，确保软件配置与硬件板级的实际电路连接是对得上的，否则无法正常工作。

2.2.8 注意事项

- 如果执行“./build.py”提示无权限，可执行命令“chmod +x build.py”添加执行权限或执行“python3./build.py”。
- 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。

3 新建 APP

3.1 建立源码目录

说明

用户可在“application/ws63”同级目录下参考“ws63_liteos_application”目录建立 app，以下均以建立“my_demo”为例。

步骤如下：

步骤 1 新建“application/ws63/my_demo”目录，用来存放“my_demo”的源文件。

步骤 2 复制“application/ws63/ws63_liteos_application/CMakeLists.txt”到“application/ws63/my_demo/CmakeLists.txt”，并将源文件放在“application/ws63/my_demo”目录下。

步骤 3 修改“application/ws63/my_demo/CmakeLists.txt”文件。其中各个变量的含义如表 3-1 所示。

表3-1 组件的 CmakeLists.txt 中的变量含义

变量名称	变量含义
COMPONENT_NAME	当前组件名称，如“my_demo”。
SOURCES	当前组件的 C 文件列表，其中

变量名称	变量含义
	CMAKE_CURRENT_SOURCE_DIR 变量标识当前 CMakeLists.txt 所在的路径。
PUBLIC_HEADER	当前组件需要对外提供的头文件的路径。
PRIVATE_HEADER	当前组件内部的头文件搜索路径。
PRIVATE_DEFINES	当前组件内部生效的宏定义。
PUBLIC_DEFINES	当前组件需要对外提供的宏定义。
COMPONENT_PUBLIC_CC FLAGS	当前组件需要对外提供的编译选项。
COMPONENT_CCFLAGS	当前组件内部生效的编译选项。

步骤 4 修改“application/ws63/CMakeLists.txt”，将 my_demo 目录加入编译。

步骤 5 修改“build/config/target_config/ws63/config.py”，在 ram_component 字段中加入‘my_demo’，向编译系统中注册 my_demo 组件。

----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“application/samples”进行移植），代码开发完成后即可使用“python3 build.py -c ws63-liteos-app -component=my_demo”编译 my_demo 进行代码编译调试。

3.3 镜像烧录

镜像烧录方法，请参见《WS63V100 BurnTool 工具 使用指南》中“操作指南”章节。