

WS63V100 第三方软件

移植指南

文档版本 02

发布日期 2024-06-27

前言

概述

本文档详细的描述了 WS63V100 移植第三方软件到 SDK 中的构建操作指导，同时提供了常见的问题解答及故障处理方法。





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。
 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。

符号	说明
	“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	2024-06-27	<ul style="list-style-type: none">更新 “1.2 CMake 构建” 章节内容。更新 “2.1 CMake 基本语法” 章节内容。
01	2024-04-10	第一次正式版本发布。
00B01	2024-02-22	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 CMake 构建1

2 常见问题5

2.1 CMake 基本语法5

2.2 头文件引用问题5

1 移植指引

1.1 概述

1.2 CMake 构建

1.1 概述

WS63V100 的 SDK 使用 CMake 作为构建工具，因此建议使用 CMake 进行第三方库的移植，从而保证编译的完整性和连贯性。其主要文件编译依赖 CMakeLists.txt 文件，当需要新增并编译第三方组件时，需要对 CMake 框架进行修改新增，即修改 CMakeLists.txt。

1.2 CMake 构建

以移植 cJSON 为例（SDK 已集成该组件，可以参考对应文件的修改），移植的步骤如下：

步骤 1 将第三方组件放置于 “opensource” 目录下（例如：opensource/cjson/cjson，新增一层路径便于对文件路径处理）。

步骤 2 在 “opensource” 目录下，找到本层级的 CMakeLists.txt，在该文件内新增一行

```
add_subdirectory_if_exist(cjson)
```

即可将对应的 “opensource/cjson” 路径新增到编译框架中。

open_source				
名称	修改日期	类型	大小	
7-zip-lzma-sdk	2024/6/25 10:27	文件夹		
cjson	2024/6/25 10:27	文件夹		
GmSSL3.0	2024/6/25 10:27	文件夹		
libboundscheck	2024/6/25 10:27	文件夹		
libcoap	2024/6/25 10:27	文件夹		
lwip	2024/6/25 10:27	文件夹		
mbdttls	2024/6/25 10:27	文件夹		
mqtt	2024/6/25 10:27	文件夹		
wpa_supplicant	2024/6/25 10:27	文件夹		
CMakeLists.txt	2024/6/25 4:35	文本文档	3 KB	

步骤 3 对新增组件内部 opensource/cjson 路径的 CMakeList.txt 进行修改（可参考已有的第 三方组件）。

设置组件名称：

```
set(COMPONENT_NAME "cjson")
```

将 xxx.c 加入到编译：

```
set(SOURCES xxx.c)
```

私有头文件引用路径：

```
set(PRIVATE_HEADER yyy)
```

私有编译参数：

```
set(COMPONENT_CCFLAGS zzz)
```

设置输出路径：

```
set(LIB_OUT_PATH ...)
```

open_source > cjson				
名称	修改日期	类型	大小	
cjson	2024/6/27 11:58	文件夹		
CMakeLists.txt	2024/6/25 4:35	文本文档	2 KB	

```

open_source > cJSON > M CMakeLists.txt
1  #=====
2  # @brief      cmake file
3  # Copyright (c) @CompanyNameMagicTag 2023-2023. All rights reserved.
4  #=====
5  set(COMPONENT_NAME "cjson")
6
7  set(CMAKE_CJSON_SOURCE_DIR
8      ${CMAKE_CURRENT_SOURCE_DIR})
9
10 set(SOURCES
11     ${CMAKE_CJSON_SOURCE_DIR}/cjson/cJSON.c
12     ${CMAKE_CJSON_SOURCE_DIR}/cjson/cJSON_Utils.c
13 )
14
15 set(PUBLIC_HEADER
16     ${CMAKE_CJSON_SOURCE_DIR}/cjson
17 )
18
19 set(PRIVATE_HEADER
20     ${CMAKE_CJSON_SOURCE_DIR}/cjson
21 )
22
23 # use this when you want to add ccflags like -include xxx
24 set(COMPONENT_PUBLIC_CCFLAGS
25 )
26
27 set(COMPONENT_CCFLAGS
28     -Wno-error=logical-op
29     -Wno-error=sign-compare
30     -Wno-error=jump-misses-init
31     -Wno-sign-compare
32     -Wno-jump-misses-init
33     -Wno-error=unused-parameter
34     -Wno-unused-parameter
35     -Wno-unused-but-set-variable
36     -Wno-error=unused-variable
37 )
38
39 set(WHOLE_LINK
40     true
41 )
42
43 set(MAIN_COMPONENT
44     false
45 )
46
47 set(LIB_OUT_PATH ${BIN_DIR}/${CHIP}/libs/wifi/${TARGET_COMMAND})
48
49 build_component()

```

步骤 4 现在，已经成功将一个名为“cjson”（COMPONENT_NAME）的组件新增到框架中了，最后应开启对该组件的编译。通过修改“build/config/target_config/ws63/config.py”中对应的 target 的“ram_component”，将需要编译的组件加入到编译流程中。例如：

想要编译的 target 名称为 “ws63-liteos-app”，则找到 “ws63-liteos-app” 字典下的 ram_component，在该数组中新增值 “cjson”，当启动 “ws63-liteos-app” 的编译时（使用 IDE 启动构建，或使用 python build.py 编译），CMake 就会尝试编译 “cjson”（SOURCES）。

```
build > config > target_config > ws63 > config.py
46 MBEDTLS_CONFIG_FILE=\ config-ws-iot.h\"
47 ],
48 'ram_component': [
49     'ws63_liteos_app',
50     'ws63_liteos_app_lds',
51     'liteos_port',
52     'irmalloc',
53     'non_os',
54     ':-rtc',
55     'arch_port',
56     'board_config', # 被 pinctrl 组件依赖
57     'lpm', # 被 pwm 组件依赖
58     'chip_ws63', 'pmp_cfg_ws63',
59     'reboot', 'hal_reboot', 'reboot_port', 'cpu_utils', 'hal_cpu_core',
60     'testsuite',
61     'gpio', 'hal_gpio_v150', 'gpio_port',
62     'dfx_port_ws63', 'algorithm', 'cmn_header', 'lwip', 'lwip_tcm', 'wifi_
63     'at', 'wifi_driver_hmac', 'wifi_driver_dmac', 'wifi_driver_tcm', 'wifi
64     'wifi_auto_adjust_freq', 'wifi_alg_anti_interference', 'wifi_alg_edca
65     'wifi_btcoex', 'wifi_uapsd_ap', 'sio_port', 'i2s', 'hal_sio',
66     'liteos_208_5_0', 'rtc_unified', 'hal_rtc_unified', 'rtc_unified_port'
67     'nv', 'nv_ws63', 'nv_zdiag_ws63', 'plt_at', 'dfx_printer',
68     'update_common', 'update_common_ws63', 'update_ab_ws63', 'factory_ws63
69     'pm_port_ws63',
70     'gmssl_hmac_sm3',
71     'bt_at',
72     'bt_host',
73     'bg_common',
74     'bth_gle',
75     'bth_sdk',
76     'samples',
77     'bts_header',
78     'bt_app',
79     'mips',
80     'hal_mips',
81     'bgtp',
82     'soc_port',
83     'radar_sensing',
84     'radar_at',
85     'radar_ai',
86     'cjson',
87     'xo_trim_port',
88     'mqtt',
89     'coap'
90 ],
```

----结束

2 常见问题

本章节对用户常见的编译问题进行收集整理，并为用户提供基本的参考。

2.1 CMake 基本语法

2.2 头文件引用问题

2.1 CMake 基本语法

除了上文提到的“COMPONENT_NAME”、“SOURCES”、“PRIVATE_HEADER”、“COMPONENT_CCFLAGS”、“LIB_OUT_PATH”之外，还可以定义公共头文件、公共编译参数等。

用户可参考标准 CMakeLists.txt 的开发流程，对 SDK 进行定制化修改。

若在三方组件中已有 CMakeLists.txt 可进行调用，也可以 opensource 路径下的 CMakeLists.txt 将文件夹直接 add_subdirectory_if_exists(xxx)，来进行编译

2.2 头文件引用问题

当用户在开发过程中需要引用新增第三方组件的头文件时，可以在第三方组件的 CMakeLists.txt 中指定公共的头文件。

例如，需要引用 cJSON 组件下的“cJSON.h”（实际上已全文件引用，仅作参考），则需要在 opensource/cjson 中的 CMakeLists.txt 增加：

```
set (PUBLIC_HEADER ${CMAKE_CURRENT_SOURCE_DIR}/cjson/cJSON.h)
```