

WS73V100 Linux 平台驱动移植

用户指南

文档版本 08

发布日期 2024-10-21

前言

概述

本文档介绍了 WS73V100 SDK 开发环境及 Linux 平台的驱动移植指导，用于帮助用户在快速了解开发环境后进行二次开发。




读者对象



本文档主要适用于以下工程师：

- 软件开发工程师
- 软件测试工程师
- 技术支持工程师

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。

符号	说明
 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
08	2024-10-21	<ul style="list-style-type: none">更新“2.2 SDK 目录结构介绍”中目录结构。更新“3.5.2 自研协议栈驱动移植”中目录结构。更新“3.7.1 ko 编译”中目录结构。
07	2024-08-05	更新“3.5.1.2 工具移植步骤”中的操作步骤。
06	2024-06-07	<ul style="list-style-type: none">更新“2.2 SDK 目录结构介绍”中目录结构。删除“SDK 小型化使用说明”小节内容。
05	2024-04-17	更新“SDK 小型化使用说明”小节内容。
04	2024-04-08	<ul style="list-style-type: none">更新“3.5.1.2 工具移植步骤”中的操作步骤。新增“3.7 Linux 平台 UART 版本移植”小节内容。
03	2024-02-06	更新“3.5.1.2 工具移植步骤”中的操作步骤。
02	2024-01-08	<ul style="list-style-type: none">更新“2.1 SDK 模块框架”中系统框架说

文档版本	发布日期	修改说明
		<p>明。</p> <ul style="list-style-type: none">更新“3.5.1.2 工具移植步骤”中移植步骤。更新“3.6.3.2 加载驱动”中驱动操作步骤。
01	2023-12-11	<p>第一次正式版本发布。</p> <ul style="list-style-type: none">更新“2.2 SDK 目录结构介绍”中目录结构。更新“3.3 驱动加载步骤”小节内容。更新“3.4.2 工具移植步骤”中工具移植步骤。更新“3.6.3 星闪业务调试”小节内容。
00B03	2023-12-01	<ul style="list-style-type: none">更新“1.2 Python 环境安装”小节内容。更新“2.2 SDK 目录结构介绍”中目录结构。删除“Menuconfig 配置”小节内容。更新“2.3 编译 SDK”中编译说明及注意事项。删除“内核 PATCH 修改”小节内容。新增“3.3 驱动加载步骤”小节内容。更新“3.4 Linux 平台 WiFi 移植”中驱动与工具的移植步骤。更新“3.5 Linux 平台蓝牙移植”中驱动移植步骤。更新“3.6 Linux 平台星闪移植”中驱动移植步骤及工具获取方法。
00B02	2023-10-31	<ul style="list-style-type: none">更新“2.3.1 config 配置方法”的驱动编译参数。更新“Menuconfig 配置”的 Menuconfig 配置项说明。

文档版本	发布日期	修改说明
		<ul style="list-style-type: none">新增“3.2 单板配置文件配置”小节内容。更新“3.4.3 WiFi 业务调试”小节内容。更新“3.5.1.3 蓝牙业务调试”小节内容。
00B01	2023-08-01	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 Python 环境安装2

2 SDK 环境搭建4

2.1 SDK 模块框架4

2.2 SDK 目录结构介绍5

2.3 编译 SDK8

2.3.1 config 配置方法8

2.3.2 驱动代码编译11

2.3.3 注意事项11

3 Linux 移植说明12

3.1 内核参数配置12

3.1.1 配置 CFG8021112

3.1.2 配置 SDIO13

3.1.3 配置 USB14

3.1.4 配置 Netlink15

3.1.5 配置 NAT 转发 (可选)16

3.2 单板配置文件配置16

3.3 驱动加载步骤16

3.4 Linux 平台 WiFi 移植17

3.4.1 驱动移植步骤17

3.4.2 工具移植步骤18

3.4.3 WiFi 业务调试.....	19
3.5 Linux 平台蓝牙移植	19
3.5.1 bluez 协议栈驱动移植.....	19
3.5.1.1 驱动移植步骤.....	19
3.5.1.2 工具移植步骤.....	20
3.5.1.3 蓝牙业务调试.....	25
3.5.2 自研协议栈驱动移植	25
3.5.2.1 驱动移植步骤.....	25
3.5.2.2 蓝牙业务调试.....	26
3.5.2.2.1 加载驱动.....	26
3.5.2.2.2 载入工具.....	26
3.6 Linux 平台星闪移植	26
3.6.1 驱动移植步骤.....	26
3.6.2 星闪工具获取.....	27
3.6.3 星闪业务调试.....	27
3.6.3.1 星闪设备检测.....	27
3.6.3.2 加载驱动.....	28
3.6.3.3 载入工具.....	29
3.7 Linux 平台 UART 版本移植.....	29
3.7.1 ko 编译.....	29
3.7.2 ko 加载.....	34
4 常见问题.....	35
4.1 驱动加载出错.....	35
4.2 wpa_supplicant 启动失败.....	36

1 开发环境搭建

1.1 SDK 开发环境简介

1.2 Python 环境安装

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。

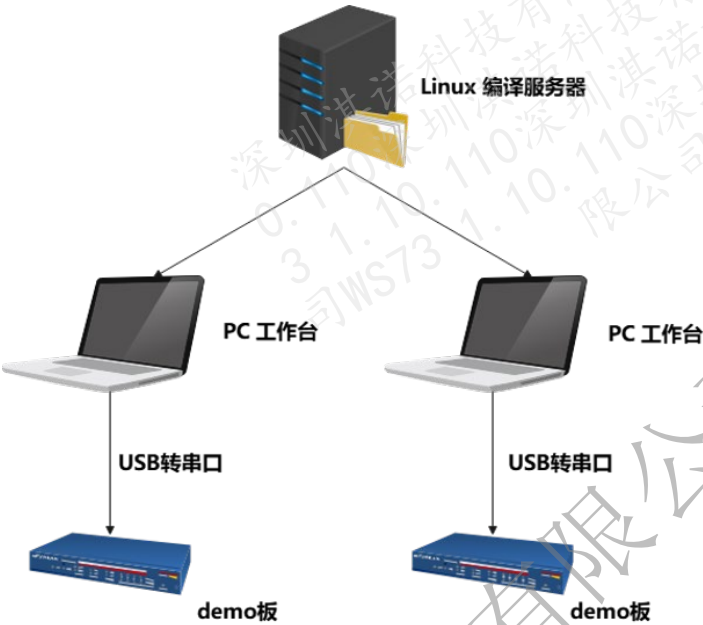
- PC 工作台

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 Python 环境安装

- 步骤 1 打开“Linux 编译服务器”，输入命令“python3 -V”，查看 Python 版本号，推荐 Python3.8 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 参考表 1-1，安装依赖库，配置 WS73 的编译环境。

表1-1 Python 依赖库

依赖库	安装命令
pyparser	pip3 install pyparser>=2.21

依赖库	安装命令
2.21+	

---结束

2 SDK 环境搭建

- 2.1 SDK 模块框架
- 2.2 SDK 目录结构介绍
- 2.3 编译 SDK

2.1 SDK 模块框架

图2-1 WS73 SDK 系统框架

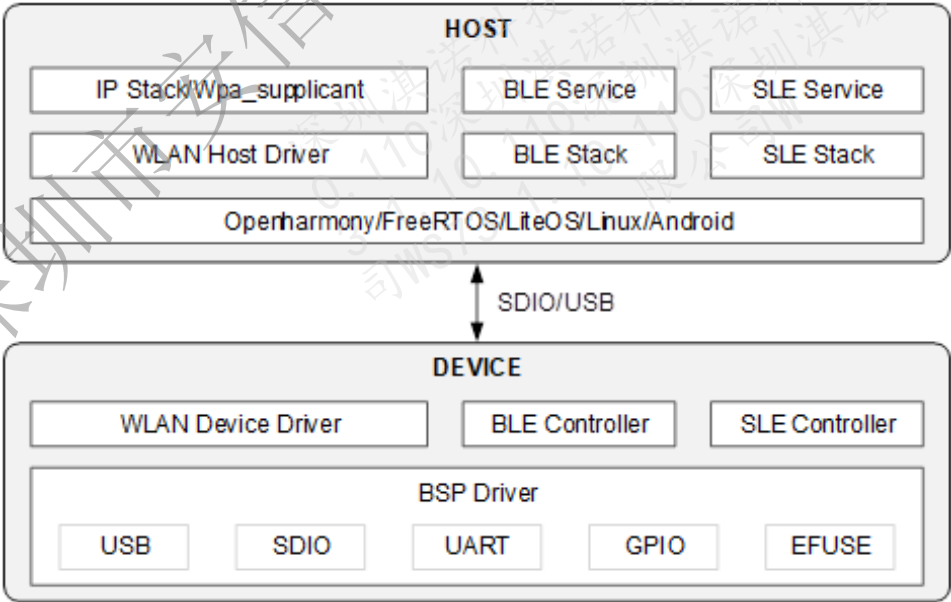


表2-1 WS73 SDK 系统框架说明

序号	模块名	功能介绍
1	IP Stack	提供网络通信服务，由主控侧提供，WS73 提供功能适配。
2	Wpa_supplicant	提供 Wi-Fi 通信服务，由开源代码仓提供，WS73 提供功能适配。
3	BLE Service	提供 BLE 通信服务，由开源代码仓提供，WS73 提供功能适配。
4	SLE Service	提供星闪通信服务，由 WS73 提供。
5	WLAN Host Driver	支持 Wi-Fi 服务的驱动，由 WS73 提供。
6	BLE Stack	支持 BLE 服务的驱动，由 WS73 提供。
7	SLE Stack	支持 SLE 服务的驱动，由 WS73 提供。
8	WLAN Device Driver	支持 Wi-Fi 服务的固件，由 WS73 提供。
9	BLE Controller	支持 BLE 服务的固件，由 WS73 提供。
10	SLE Controller	支持 SLE 服务的固件，由 WS73 提供。

2.2 SDK 目录结构介绍

说明

SDK 中 firmware 具体差异可参考《Q35343 系列 Wi-Fi、BLE 和 SLE Combo 芯片 用户指南》产品概述小节。

SDK 目录结构如下：

└─ application	# 用户态二进制文件和示例代码
├─ bin	# 星闪可执行程序
├─ lib	# 星闪可执行程序依赖库
├─ sample	# WS73 BLE/SLE示例代码
└─ build	# SDK构建所需配置文件和编译脚本

		config	# WS73 默认配置文件
		driver	# WS73的驱动文件
		android_autoconfig.h	# Android配置文件
		android.config	# Android配置文件
		bsle	# WS73 BLE/SLE驱动文件
		Makefile	# WS73 驱动编译入口
		platform	# WS73 平台文件
		wifi	# WS73 WiFi驱动文件
		firmware	# WS73 驱动依赖固件
		e	# WS73E版本固件
		us	# WS73U/S版本固件
		include	# API头文件存放目录
		Makefile	# Makefile编译总入口
		open_source	# 存放开源组件及相关patch
		output	# 编译时生成的目标文件和中间文件

SDK 开源目录代码说明:

表2-2 WS73 开源代码目录说明

序号	模块名	文件名	功能介绍
1	ble_gatt_client	ble_gatt_client.c	ble gatt 协议客户端 sample 代码， 主要实现客户端发现所有服务端注册 服务。
2	ble_uuid_server	ble_server_adv.c	ble gatt 协议服务端 sample 代码， 主要实现服务端设置广播数据和参 数，启动广播等功能。
3	ble_uuid_server	ble_uuid_server.c	ble gatt 协议服务端 sample 代码， 主要实现注册 uuid 服务和数传等功 能。
4	ble_hid_keyboa	ble_hid_keyboard	ble 键盘 hid 服务 sample 代码，主

序号	模块名	文件名	功能介绍
	rd_server	_server.c	要实现键盘 hid 服务注册，广播和数传等功能。
5	ble_hid_mouse_server	ble_hid_mouse_server.c	ble 鼠标 hid 服务 sample 代码，主要实现实现鼠标 hid 服务注册，广播和数传等功能。
6	sle_mouse_server	sle_mouse_adv.c	星闪 sle 鼠标广播 sample 代码，主要实现鼠标广播参数和数据设置，启动广播等功能。
7	sle_mouse_server	sle_mouse_dis_server.c	星闪 sle 鼠标服务注册 sample 代码，主要实现鼠标服务注册等功能。
8	sle_mouse_server	sle_mouse_hid_server.c	星闪 sle 鼠标 hid 服务注册 sample 代码，主要实现鼠标 hid 服务注册等功能。
9	sle_uuid_client	sle_uuid_client.c	星闪 ssap 协议客户端 sample 代码，主要实现 ssap 协议客户端初始化和发现设备等功能。
10	sle_uuid_server	sle_server_adv.c	星闪 ssap 协议服务端广播 sample 代码，主要实现服务端广播参数设置和广播启动等功能。
11	sle_uuid_server	sle_uuid_server.c	星闪 ssap 协议服务端 sample 代码，主要实现服务端服务注册和数传等功能。

2.3 编译 SDK

2.3.1 config 配置方法

首次运行

用户首次运行时，SDK 根目录下执行 “vi build/config/ws73_default.config” 命令，配置相关参数如下。

编译参数

WS73 编译相关参数如表 2-3 所示。

表2-3 编译参数配置说明

参数	说明	示例
WSCFG_USING_GCC	是否配置编译器类型为 GCC	WSCFG_USING_GCC=y
WSCFG_CROSS_COMPILE	配置编译器存放路径	WSCFG_CROSS_COMPILE=/opt/toolchains/bin/arm-himix100-linux-
WSCFG_KERNEL_DIR	配置内核存放路径	WSCFG_KERNEL_DIR=/home/share/kernel/linux-4.9.y
WSCFG_ARCH_ARM	是否配置 CPU 架构类型为 ARM	WSCFG_ARCH_ARM=y
WSCFG_BUS_USB	是否配置通信总线类型为 USB	WSCFG_BUS_USB=y

图2-2 编译参数配置示例

```
#
# Configure the compilation environment
#
WSCFG_USING_GCC=y
# WSCFG_USING_LLVM_CLANG is not set
WSCFG_CROSS_COMPILE="/opt/toolchains/arm-bi-110/bin/arm-bi-110-linux-"
WSCFG_CLANG_EXTRA_FLAGS=""
WSCFG_KERNEL_DIR="/home/share/110/kernel/linux-4.9.y"
WSCFG_ARCH_ARM=y
# WSCFG_ARCH_CUSTOM is not set
WSCFG_ARCH_NAME="arm"
BOARD_ASIC=y
BOARD_ASIC_WIFI=y
# WSCFG_BUS_SDIO
WSCFG_BUS_USB=y
# WSCFG_BUS_UART is not set
WSCFG_LINUX=y
PRE_OS_VERSION_LINUX=0
PRE_OS_VERSION_WIN32=1
PRE_OS_VERSION_WINDOWS=2
PRE_OS_VERSION_RAW=3
PRE_OS_VERSION_HIRTOS=4
PRE_OS_VERSION_WIN32_RAW=5
PRE_OS_VERSION_LITEOS=6
PRE_OS_VERSION_ANDROID=7
PRE_OS_VERSION=0
# end of build
```

说明

- WSCFG_CROSS_COMPILE 和 WSCFG_KERNEL_DIR 需读者依照实际情况进行配置。
- 若非 ARM 架构，可关闭 WSCFG_ARCH_ARM，打开 WSCFG_ARCH_CUSTOM 宏，并修改 WSCFG_ARCH_NAME，如 WSCFG_ARCH_NAME="mips"。
- 若非 SDIO 总线，可关闭 WSCFG_BUS_SDIO，打开对应总线宏，如 WSCFG_BUS_USB。

平台参数

WS73 平台相关参数如表 2-4 所示。

表2-4 平台参数配置说明

参数	说明	示例
CONFIG_FIRMWARE_BIN_PATH	配置 ws73.bin 文件路径	CONFIG_FIRMWARE_BIN_PATH="/etc/ws73/ws73.bin"
CONFIG_FIRMWARE_WIFICALI_PATH	配置 wifi_cali.bin 文件路径	CONFIG_FIRMWARE_WIFICALI_PATH="/etc/ws73/wifi_cali.bin"
CONFIG_FIRMWARE_BSLECALI_PATH	配置 btc_cali.bin 文件路径	CONFIG_FIRMWARE_BSLECALI_PATH="/etc/ws73/btc_cali.bin"
CONFIG_FIRMWARE_WOW_PATH	配置 wow.bin 文件路径	CONFIG_FIRMWARE_WOW_PATH="/etc/ws73/wow.bin"
CONFIG_INI_FILE_PATH	配置 ws73_cfg.ini 文件路径	CONFIG_INI_FILE_PATH="/etc/ws73_cfg.ini"

参数	说明	示例
CONFIG_PLAT_DFR_OUTPUT_PATH	配置 ws73 panic 文件存在路径	CONFIG_PLAT_DFR_OUTPUT_PATH="/etc/ws73"

图2-3 平台参数配置示例

```
#
# Platform
#
_PRE_PLAT_SHA256SUM_CHECK=y
WSCFG_ONEIMAGE=y
WSCFG_PLAT_VARIABLE_FEATRUE=y
WSCFG_PLAT_DIAG_LOG_OUT=y
# CONFIG_DIAG_SUPPORT_SOCKET is not set
CONFIG_DIAG_SUPPORT_UART=y
CONFIG_DIAG_SUPPORT_LOCAL_LOG=y
CONFIG_PLAT_SUPPORT_DFR=y
CONFIG_PLAT_DFR_OUTPUT_PATH="/etc/ws73"
CONFIG_HCC_SUPPORT_TEST=y
CONFIG_HCC_ERROR_PRINT=y
_PRE_PLAT_HCC_SDI0=y
BT_EM_BUFFER_CALI_SUPPORT=y
# _PRE_PLAT_SLP_UART_FORWARD is not set
CONFIG_DFX_SUPPORT_SYSFS=y
# CONFIG_SDI0_RESCAN is not set

#
# Configure the loading path of files such as firmware
#
CONFIG_FIRMWARE_BIN_PATH="/etc/ws73/ws73.bin"
CONFIG_FIRMWARE_WIFICALI_PATH="/etc/ws73/wifi_cali.bin"
CONFIG_FIRMWARE_BSLECALI_PATH="/etc/ws73/btc_cali.bin"
CONFIG_FIRMWARE_WOW_PATH="/etc/ws73/wow.bin"
CONFIG_INI_FILE_PATH="/etc/ws73_cfg.ini"
# end of Configure the loading path of files such as firmware
```

说明

- 配置 CONFIG_FIRMWARE_BIN_PATH、CONFIG_FIRMWARE_WIFICALI_PATH、CONFIG_FIRMWARE_BSLECALI_PATH、CONFIG_FIRMWARE_WOW_PATH等参数后，在加载 ko 前，需要保证单板对应目录下，有对应的二进制文件。二进制文件在 SDK 中的 firmware 目录下，有 us/e 两个版本，根据实际情况选用对应版本。两个版本的具体差异可参考《NW117V100 系列 Wi-Fi、BLE 和 SLE Combo 芯片 用户指南》产品概述小节。
- 配置 CONFIG_INI_FILE_PATH 后，在加载 ko 前，需要保证单板对应目录下，有对应的 ini 文件。
- 配置 CONFIG_PLAT_DFR_OUTPUT_PATH 需要保证该路径随时可写，以便出现系统 panic 时，能正常保存日志

重复运行

用户执行 “make” 完成编译后，如需重新配置驱动参数，可以在根目录下执行 “vi build/config/ws73_default.config” 进行配置。

2.3.2 驱动代码编译

SDK 根目录下执行 “make” 指令运行脚本编译，即可编译出对应的 SDK 程序。编译命令列如表 2-5 所示。

表2-5 make 参数列表

参数	示例	说明
无	make	默认执行全量编译。
all	make all	执行全量编译。
工程名称	make 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。
light	make light	执行小型化版本编译。
clean	make clean	清理编译过程中生成的中间文件和烧写文件。
hso	make hso	生成 DebugKit 应用数据库，用于驱动维测功能

2.3.3 注意事项

- 编译过程中，报错找不到某个包，请检查环境中的 Python 版本是否满足要求，相关组件是否已安装。
- 参见《WS73V100 常见问题 FAQ》中 “SDK 开发环境和 SDK 使用类” 章节。

3 Linux 移植说明

说明

WS73 支持 USB、SDIO、USB/URAT 三种总线模式，用户移植前，先确认目标开发板的 WS73 模组使用的总线类型。（不同总线类型差异可参考《NW117V100 系列 Wi-Fi、BLE 和 SLE Combo 芯片 用户指南》1.1 概述）

3.1 内核参数配置

3.2 单板配置文件配置

3.3 驱动加载步骤

3.4 Linux 平台 WiFi 移植

3.5 Linux 平台蓝牙移植

3.6 Linux 平台星闪移植

3.7 Linux 平台 UART 版本移植

3.1 内核参数配置

3.1.1 配置 CFG80211

CFG80211 也是内核中 Wi-Fi 驱动和用户态进程的标准接口。

步骤 1 进入内核源码目录，执行 make menuconfig 命令，进入 Networking support → Wireless，设置 cfg80211 和 mac80211。

----结束

图3-1 配置 CFG80211

```
--- Wireless
[*]> cfg80211 - wireless configuration API
[ ] nl80211 testmode command
[ ] enable developer warnings
[*] enable powersave by default
[ ] cfg80211 wireless extensions compatibility
[ ] lib80211 debugging messages
<M> Generic IEEE 802.11 Networking Stack (mac80211)
Default rate control algorithm (Minstrel) --->
[ ] Enable mac80211 mesh networking (pre-802.11s) support
[ ] Trace all mac80211 debug messages
[ ] Select mac80211 debugging features ----
```

说明

如果内核中的 CFG80211 设置为'M', cfg80211 会编译成 cfg80211.ko 文件; 如果 CFG80211 设置为 '*', cfg80211 会被集成到内核中。

3.1.2 配置 SDIO

当使用 SDIO 模组设备时, 需要开启 MMC/SD/SDIO card support.

步骤 1 进入内核源码目录, 执行 make menuconfig 命令, 进入 Device Drivers → MMC/SD/SDIO card support, 按图 3-2 进行配置。

----结束

图3-2 配置 SDIO

```
-- MMC/SD/SDIO card support
[ ] MMC debugging
<*> HW reset support for eMMC
<*> Simple HW reset support for MMC
*** MMC/SD/SDIO Card Drivers ***
<*> MMC block device driver
(8) Number of minors per block device
[*] Use bounce buffer for simple hosts
< > SDIO UART/GPS class support
< > MMC host test driver
*** MMC/SD/SDIO Host Controller Drivers ***
<*> Secure Digital Host Controller Interface support
<*> SDHCI platform and OF driver helper
< > SDHCI OF support for the Atmel SDMMC controller
< > SDHCI support for Fujitsu Semiconductor F SDH30
< > Winbond W83L51xD SD/MMC Card Interface support (NEW)
< > MMC/SD/SDIO over SPI
< > VUB300 USB to SDIO/SD/MMC Host Controller support
< > USB SD Host Controller (USHC) support
< > Renesas USDHI6R0L0 SD/SDIO Host Controller support
< > MediaTek SD/MMC Card Interface support
< > Command Queue Support
```

3.1.3 配置 USB

当使用 USB 模组设备时，需要开启 USB Support。

步骤 1 进入内核源码目录，执行 make menuconfig 命令，进入 Device Drivers → USB support，设置 “xHCI HCD (USB 2.0) support” 为 “*”。

---结束

图3-3 配置 USB

```

--- USB support
<*> Support for Host-side USB
[ ] USB announce new devices
    *** Miscellaneous USB options ***
[*] Enable USB persist by default
[ ] Dynamic USB minor allocation
[ ] OTG support
[ ] Rely on OTG and EH Targeted Peripherals List
< > USB Monitor
< > Support WUSB Cable Based Association (CBA)
    *** USB Host Controller Drivers ***
< > Cypress C67x00 HCD support
<*> xHCI HCD (USB 3.0) support
- *- Generic xHCI driver for a platform device
<*> EHCI HCD (USB 2.0) support
[*] Root Hub Transaction Translators
[*] Improved Transaction Translator scheduling
<*> Generic EHCI driver for a platform device

```

3.1.4 配置 Netlink

由于 wpa_supplicant、hostapd 应用采用 Netlink 技术与 Linux 内核通信，需要配置 Netlink。

- 步骤 1 进入内核源码目录，执行 make menuconfig 命令。选择 Networking support → Networking options，设置 “Network packet filtering framework (Netfilter)” 为 “y”（选择为 “*”）。
- 步骤 2 进入 Network packet filtering framework (Netfilter)，设置 “Advanced netfilter Configuration” 为 “y”（选择为 “*”）。

-----结束

图3-4 配置 Netlink

```

--- Network packet filtering framework (Netfilter)
[ ] Network packet filtering debugging (NEW)
[*] Advanced netfilter configuration (NEW)
<M> Bridged IP/ARP packets filtering (NEW)
Core Netfilter Configuration --->
< > IP set support (NEW) ----
< > IP virtual server support (NEW) ----
IP: Netfilter Configuration --->
IPv6: Netfilter Configuration --->

```


3.1.5 配置 NAT 转发（可选）

如果需要使用 SoftAP 网络共享功能，需要配置 NAT 转发功能。

- 步骤 1 进入内核源码目录，执行 make menuconfig 命令。选择 Networking support → Networking options → Network packet filtering framework (Netfilter) → Core Netfilter Configuration，设置 “Netfilter connection tracking support” 为 “y”（选择为 “*”）。
- 步骤 2 再进入 Networking support → Networking options → Network packet filtering framework (Netfilter)，根据使用需求配置 NAT 转发功能。

----结束

图3-5 配置 NAT 转发

```
<*> Netfilter IPv4 packet duplication to alternate destination
<*> ARP packet logging
<*> IPv4 packet logging
<*> IPv4 packet rejection
<*> IP tables support (required for filtering/masq/NAT)
<> "ah" match support
<> "ecn" match support (NEW)
<> "ttl" match support (NEW)
<> Packet filtering (NEW)
<> Packet mangling (NEW)
<> raw table support (required for NOTRACK/TRACE) (NEW)
<*> ARP tables support
<> ARP packet filtering (NEW)
<> ARP payload mangling (NEW)
```

3.2 单板配置文件配置

具体内容请参见《WS73V100 单板配置文件说明书》中“GPIO 配置”小节。

3.3 驱动加载步骤

- 步骤 1 默认加载 cfg80211.ko
- 步骤 2 确认采用的驱动加载模式，若通过配置启动脚本来加载驱动，可直接跳到步骤 4；若通过适配驱动模组设备号，再加载驱动，可继续执行步骤 3。
- 步骤 3 针对 USB 模组，可执行 “lsusb” 查看模组的 VID 和 PID 号，其中 WS73 模组默认值为 ffff:3733。

```
/komod # lsusb
Bus 001 Device 001: ID 1d6b:0002
Bus 001 Device 002: ID ffff:3733
Bus 002 Device 001: ID 1d6b:0003
```

针对 SDIO 模组，可查看内核实际注册的设备节点来获取模组的 VID 和 PID 号，其中 WS73 模组默认值为 ffff:3733。

```
/ # cat /sys/bus/sdio/devices/mmc1\:0001\:1/uevent
SDIO_CLASS=07
SDIO_ID=FFFF:3733
MODALIAS=sdio:c07vFFFFd3733
/ #
```

步骤 4 先加载 plat_soc.ko，再根据需要加载 wifi_soc.ko、sle_soc.ko、ble_soc.ko（wifi、sle、ble 间无依赖关系，只要在 plat ko 后加载即可）

----结束

3.4 Linux 平台 WiFi 移植

3.4.1 驱动移植步骤

步骤 1 解压驱动源码包。

将驱动源码包 WS73V100R001C00SPCXXX.tar.gz 放置于服务器上，并进行解压，参考“2.3 编译 SDK”配置编译工具链、内核路径、CPU 架构等，命令如下：

```
$ tar -xzf WS73V100R001C00SPCXXX.tar.gz
$ cd WS73V100R001C00SPCXXX
```

步骤 2 完成配置后，执行 make all 命令编译驱动文件，在 output 生成目标文件：

```
$ make all
```

编译结果输出到"output"目录下，如表 3-1 所示。

表3-1 全量编译结果

文件名	说明
plat_soc.ko	WS73 平台驱动模块。
wifi_soc.ko	WS73WiFi 驱动模块。
ble_soc.ko	WS73 蓝牙驱动模块。

文件名	说明
sle_soc.ko	WS73 星闪驱动模块。
ws73_cfg.ini	WS73 客制化的配置文件。

----结束

📖 说明

- 编译工具链接需要提前配置环境变量，例如 `export PATH=$PATH:/home/wifi/share/arm-linux-gnueabi/bin`，路径根据实际情况配置。
- 使用不同工具链时，可能会导致编译过程中出现告警导致编译失败，可在相关驱动组件的 Makefile 文件中修改编译选项。

3.4.2 工具移植步骤

步骤 1 用户选择所需的第三方组件进行下载，相关组件下载链接如下：

- libnl-3.5.0 下载链接：
<https://www.linuxfromscratch.org/blfs/view/9.1/basicnet/libnl.html>
- wpa_supplicant-2.10、hostapd-2.10 下载链接：<http://w1.fi/releases>
- openssl-1.1.1n 下载链接：<https://www.openssl.org/source/old/1.1.1>

步骤 2 将下载压缩包放置 SDK 相应目录，并执行下列命令：

- 解压 libnl-3.5.0 至 SDK 的 `open_source/libnl` 目录下。
- 解压 openssl-1.1.1 至 SDK 的 `open_source/openssl` 目录下。
- 解压 wpa_supplicant-2.10 至 SDK 的 `open_source/wpa_supplicant` 目录下。
- 解压 hostapd-2.10 至 SDK 的 `open_source/hostapd` 目录下。

```
$ mkdir -p open_source/libnl && tar -xzf libnl-3.5.0.tar.gz -C open_source/libnl/ --strip-component=1
$ mkdir -p open_source/openssl && tar -xzf openssl-1.1.1n.tar.gz -C open_source/openssl/ --strip-component=1
$ mkdir -p open_source/wpa_supplicant && tar -xzf wpa_supplicant-2.10.tar.gz -C open_source/wpa_supplicant/ --strip-component=1
$ mkdir -p open_source/hostapd && tar -xzf hostapd-2.10.tar.gz -C open_source/hostapd/ --strip-component=1
```

步骤 3 给 wpa_supplicant 组件补丁，相关命令如下：

```
$ cd open_source/wpa_supplicant (以用户实际路径为准)
```

```
$ patch -p1 < wpa_supplicant_2_10_linux.patch
```

步骤 4 完成对相关组件的补丁后，在 SDK 根目录下执行“make tools”命令，在 output 目录下生成目标文件如下：

```
$ ls output/bin/open_source/
hostapd      hostapd.conf  libnl-3.so.200.26.0  libnl-route-3.so.200  wpa_cli      wpa_supplicant.conf
hostapd_cli  libcrypto.so.1.1  libnl-genl-3.so.200.26.0  libssl.so.1.1        wpa_supplicant
```

----结束

📖 说明

提供的 wpa_supplicant.patch 是基于 wpa_supplicant 2.10 生成的补丁文件，若用户使用其他版本的开源组件，可能会产生兼容性问题。

3.4.3 WiFi 业务调试

参考《WS73V100 Linux Wi-Fi、BLE 软件开发指南》。

3.5 Linux 平台蓝牙移植

3.5.1 bluez 协议栈驱动移植

3.5.1.1 驱动移植步骤

步骤 1 解压驱动源码包。

将驱动源码包 WS73V100R001C00SPCXXX.tar.gz 放置于服务器上，并进行解压，参考“2.3 编译 SDK”配置编译工具链、内核路径、CPU 架构等，命令如下：

```
$ tar -xzf WS73V100R001C00SPCXXX.tar.gz
$ cd WS73V100R001C00SPCXXX
```

步骤 2 完成配置后，执行 make all 命令编译驱动文件，在 output 生成目标文件：

```
$ make all
```

----结束

📖 说明

- 编译工具链需要提前配置环境变量，例如 export PATH=\$PATH:/home/AIoT/share/arm-himix100-linux/bin，路径根据实际情况配置。
- prefix=/vendor 目录是对应工具库所安装的位置，dbus 运行依赖/vendor/share/dbus-1 该位置下的配置文件，可根据系统版本情况配置。

3.5.1.2 工具移植步骤

步骤 1 添加系统依赖工具：

```
apt-get install gettext
apt-get install libglib2.0-dev
apt install python-docutils
apt-get install libtool
apt-get install autoconf
apt install pkg-config
```

📖 说明

以上命令需要使用 Root 用户执行。

步骤 2 下载第三方工具 bluez-5.64.tar.xz 及其依赖库，需要下载对应版本的源码压缩文件，下载链接如下：

```
bluez-5.64.tar.xz--https://mirrors.edge.kernel.org/pub/linux/bluetooth/bluez-5.64.tar.xz
dbus-1.12.20.tar.gz--https://dbus.freedesktop.org/releases/dbus/dbus-1.12.20.tar.gz
expat-2.4.6.tar.gz--
https://github.com/libexpat/libexpat/releases/download/R_2_4_6/expat-2.4.6.tar.gz
gettext-0.21.tar.gz--https://ftp.gnu.org/gnu/gettext/gettext-0.21.tar.gz
json-c-0.13.tar.gz--http://sources.buildroot.net/json-c/json-c-0.13.tar.gz
libffi-3.3.tar.gz--https://gcc.gnu.org/pub/libffi/libffi-3.3.tar.gz
libical-1.0.tar.gz--https://src.fedoraproject.org/repo/pkgs/libical/libical-1.0.tar.gz
ncurses-6.3.tar.gz--https://ftp.gnu.org/gnu/ncurses/ncurses-6.3.tar.gz
pcre-8.45.tar.gz--https://ftp.exim.org/pub/pcre/pcre-8.45.tar.gz
readline-8.1.tar.gz--https://ftp.gnu.org/gnu/readline/readline-8.1.tar.gz
zlib-1.2.11.tar.gz--https://src.fedoraproject.org/repo/pkgs/R/zlib-1.2.11.tar.gz
glib-2.40.0.tar.xz--https://src.fedoraproject.org/repo/pkgs/glib2/glib-2.40.0.tar.xz
```

步骤 3 下载压缩文件并放至 SDK 相应目录下。

- 解压 bluez-5.64.tar.xz 至 SDK 的 open_source 目录下。

```
tar -xvf bluez-5.64.tar.xz
```

- 解压依赖库至 SDK 的 open_source 目录下。

```
tar zxvf expat-2.4.6.tar.gz
tar zxvf libical-1.0.tar.gz
tar zxvf dbus-1.12.20.tar.gz
tar zxvf zlib-1.2.11.tar.gz
tar zxvf libffi-3.3.tar.gz
```

```
tar zxvf ncurses-6.3.tar.gz
tar zxvf readline-8.1.tar.gz
tar zxvf pcre-8.45.tar.gz
tar zxvf gettext-0.21.tar.gz
tar -xvf glib-2.40.0.tar.xz
```

步骤 4 添加对 glib 的修改，步骤如下：

- 打开 glib-2.40.0/glib/gdate.c 文件。
- 找到 g_date_strftime 方法定义。
- 在方法前增加如下代码：

```
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
```

修改后如下图：

```
2442 #pragma GCC diagnostic push
2443 #pragma GCC diagnostic ignored "-Wformat-nonliteral"
2444 gsize .....
2445 g_date_strftime (gchar *s,
2446                 gsize slen,
2447                 const gchar *format,
2448                 const GDate *d)
2449
2450 struct tm tm;
```

- 在方法后增加如下代码：

```
#pragma GCC diagnostic pop
```

修改后如下图：

```
2550
2551 return retval;
2552 #endif
2553 }
2554 #pragma GCC diagnostic pop
```

步骤 5 添加对 bluez 的修改，相关步骤如下：

- 打开 bluez-5.64/tools/test-runner.c 文件。
- 在文件中增加如下代码：

```
#ifndef MS_STRICTATIME
#define MS_STRICTATIME (1 << 24)
#endif
```

修改后如下图：

```

36  #include "tools/hciattach.h"
37
38  #ifndef MS_STRICTATIME
39  #define MS_STRICTATIME (1 << 24)
40  #endif
41
42  #ifndef WAIT_ANY

```

步骤 6 编译 bluez 依赖库，相关指令如下：

- 配置环境变量

```

export PKG_CONFIG_LIBDIR=/vendor/lib/pkgconfig/
export PKG_CONFIG_LIBDIR=$PKG_CONFIG_LIBDIR:/usr/include/

```

- 编译 expat

```

cd expat-2.4.6/
./configure --prefix=/vendor --host=arm-himix100-linux CC=arm-himix100-linux-gcc
CFLAGS="-fPIC -fstack-protector-all" LDFLAGS="-Wl,-z,now -Wl,-z,relro -Wl,-z,noexecstack"
make
make install

```

- 编译 libical

```

cd libical-1.0/
export CC=arm-himix100-linux-gcc
cmake -DCMAKE_INSTALL_PREFIX=/vendor
make -j8
make install

```

- 编译 dbus

```

cd dbus-1.12.20/
./configure --prefix=/vendor --host=arm-himix100-linux --with-x=no --enable-abstract-sockets
CC=arm-himix100-linux-gcc CFLAGS="-I/vendor/include -fstack-protector-all -fPIC -fPIE -pie"
LDFLAGS="-L/vendor/lib -Wl,-z,now -Wl,-z,relro -Wl,-z,noexecstack -fstack-protector-all"
ac_cv_search_pthread_mutexattr_init= ac_cv_search_pthread_cond_timedwait=
ac_cv_search_pthread_mutexattr_settype=
make
make install

```

编译安装dbus的时候，环境上最好不要有后面依赖库的残留，否则在编译的时候会报错

- 编译 zlib

```

cd zlib-1.2.11/
CC=arm-himix100-linux-gcc prefix=/vendor CFLAGS="-fstack-protector-all -fPIC"
LDFLAGS="-L/vendor/lib -Wl,-z,now -Wl,-z,relro -Wl,-z,noexecstack" ./configure
make
make install

```

- 编译 libffi

```
cd libffi-3.3
./configure --prefix=/vendor CC=arm-himix100-linux-gcc --host=arm-himix100-linux
CFLAGS="-fPIC -fstack-protector-all" LDFLAGS="-Wl,-z,now -Wl,-z,relro -Wl,-z,noexecstack"
make
make install
```

- 编译 ncurses

```
cd ncurses-6.3
./configure --disable-stripping --host=arm-himix100-linux --prefix=/vendor CC=arm-himix100-
linux-gcc CFLAGS="-fPIC -fstack-protector-all" LDFLAGS="-Wl,-z,now -Wl,-z,relro -Wl,-
z,noexecstack" ac_cv_func_gettimeofday=yes
make
make install
```

- 编译 readline

```
export PKG_CONFIG_LIBDIR=$PKG_CONFIG_LIBDIR:/usr/lib/x86_64-linux-gnu/pkgconfig/
export PKG_CONFIG_LIBDIR=$PKG_CONFIG_LIBDIR:/usr/share/pkgconfig/
cd readline-8.1
./configure --prefix=/vendor --host=arm-himix100-linux CC=arm-himix100-linux-gcc --enable-
shared --enable-static bash_cv_wcwidth_broken=yes CFLAGS="-I/vendor/include -fstack-
protector-all -fPIC" LDFLAGS="-L/vendor/lib -Wl,-z,now -Wl,-z,relro -Wl,-z,noexecstack"
make -j4 SHLIB_LIBS=-lncurses
make install
```

- 编译 pcre

```
cd pcre-8.45
./configure --prefix=/vendor --host=arm-himix100-linux CC=arm-himix100-linux-gcc CXX=arm-
himix100-linux-g++ CFLAGS="-I/vendor/include -fstack-protector-all -fPIC" LDFLAGS="-
L/vendor/lib -Wl,-z,now -Wl,-z,relro -Wl,-z,noexecstack"
make
make install
```

- 编译 gettext

```
cd gettext-0.21
./configure --prefix=/vendor --host=arm-himix100-linux CC=arm-himix100-linux-gcc CXX=arm-
himix100-linux-g++ CFLAGS="-fPIC -fstack-protector-all" LDFLAGS="-L/vendor/lib -Wl,-z,now
-Wl,-z,relro -Wl,-z,noexecstack" ac_cv_func_gettimeofday=yes
make -j4
make install
```

- 编译 glib

```
cd glib-2.40.0
./configure --prefix=/vendor --host=arm-himix100-linux CC=arm-himix100-linux-gcc
```



```
CFLAGS="-I/vendor/include -fPIC -fstack-protector-all" CPPFLAGS=-I/vendor/include
LDFLAGS="-L/vendor/lib -Wl,-z,now -Wl,-z,relro -Wl,-z,noexecstack" glib_cv_stack_grows=no
glib_cv_uscore=yes ac_cv_func_posix_getpuid_r=yes ac_cv_func_posix_getgrgid_r=yes
make
make install
```

步骤 7 编译 bluez 协议栈及用户态工具，相关指令如下：

- 配置环境变量

```
export PKG_CONFIG_LIBDIR=$PKG_CONFIG_LIBDIR:/usr/lib/x86_64-linux-
gnu/pkgconfig:/usr/share/pkgconfig/
```

- 编译 bluez

```
cd bluez-5.64
autoreconf
./configure --prefix=/vendor --host=arm-himix100-linux CC=arm-himix100-linux-gcc --enable-
deprecated --enable-tools --enable-cups --enable-test CFLAGS="-I/vendor/include -fstack-
protector-all -fPIC -fPIE -pie" LDFLAGS="-fstack-protector-all -L/vendor/lib -Wl,-z,now -Wl,-
z,relro -Wl,-z,noexecstack -lpcre" --enable-experimental -enable-maintainer-mode --disable-
udev --enable-pie --enable-shared --enable-static ac_cv_lib_rt_clock_gettime=yes &&sed -i
's/^MISC_LDFLAGS = -pie -Wl,-z,now/MISC_LDFLAGS = -pie -Wl,-z,now/g' Makefile&&
sed -i s/-Werror//g ./Makefile
make -j4
make install
```

- 交叉编译环境没有 kernel 头文件，make 会报如下的错误

```
linux/uhid.h: No such file or directory
```

可将系统 kernel 头文件拷贝到 vendor 目录供编译引用，执行如下命令，重新编译

```
cp -r /usr/include/linux /vendor/include
```

步骤 8 生成目标文件在 vendor 目录，如表 3-2 所示。

表3-2 生成目标文件目录

目录	说明
/vendor/lib	依赖库动态库文件： libglib-2.0.so.0 libexpat.so.1 libpcre.so.1 libdbus-1.so.3 libintl.so.8

目录	说明
	libreadline.so.8
/vendor/bin	dbus 及 bluez 相关工具可执行文件： bluetoothctl dbus-daemon hciconfig hcitool
/vendor/share/dbus-1	dbus 配置文件： session.conf system.conf
/vendor/libexec/bluetooth	bluez 协议栈可执行文件： bluetoothd

----结束

3.5.1.3 蓝牙业务调试

参考《WS73V100 Linux Wi-Fi、BLE 软件开发指南》中的“Bluez 蓝牙业务基础操作示例”章节。

3.5.2 自研协议栈驱动移植

3.5.2.1 驱动移植步骤

步骤 1 解压驱动源码包。

将驱动源码包 WS73V100R001C00SPCXXX.tar.gz 放置于服务器上，并进行解压，参考“2.3 编译 SDK”配置编译工具链、内核路径、CPU 架构等，命令如下：

```
$ tar -xzf WS73V100R001C00SPCXXX.tar.gz
$ cd WS73V100R001C00SPCXXX
```

步骤 2 完成配置后，执行 make ble_android 命令编译蓝牙驱动文件，在 output 生成目标文件：

```
$ make ble_android
```

----结束

3.5.2.2 蓝牙业务调试

参考《WS73V100 BLE 和 Wi-Fi 小型化开发指南》中的"BLE 小型化使用和开发指导"章节。

3.5.2.2.1 加载驱动

步骤 1 将驱动依赖的 bin 文件和 ini 文件分别拷贝至/etc/ws73 目录和/etc 目录（Host 侧默认配置路径）。

步骤 2 将编译出的驱动拷贝至单板，依次加载 plat_soc.ko、ble_soc.ko。

```
$ insmod plat_soc.ko
$ insmod ble_soc.ko
```

----结束

3.5.2.2.2 载入工具

步骤 1 将星闪执行需要的工具 "application/bin/bluetoothd" "application/bin/bluetoothctrl" 复制到/bin 目录下；

步骤 2 在 bin 目录下修改工具的可执行权限：

```
chmod a+x bluetoothd
chmod a+x bluetoothctrl
```

说明

SDK 中发布了部分主控的 bluetoothd 和 bluetoothctrl 工具，若 SDK 中无对应主控所需版本，则需提供编译工具链，并联系相关人员进行编译。

----结束

3.6 Linux 平台星闪移植

3.6.1 驱动移植步骤

步骤 1 解压驱动源码包。

将驱动源码包 WS73V100R001C00SPCXXX.tar.gz 放置于服务器上，并进行解压，参考 "2.3 编译 SDK" 中配置编译工具链、内核路径、CPU 架构等，命令如下：

```
$ tar -xzf WS73V100XXX.tar.gz
$ cd WS73V100XXX
```

步骤 2 完成配置后，执行 make all 命令编译驱动文件，在 output 生成目标文件：

```
$ make all
```

----结束

📖 说明

- 编译工具链接需要提前配置环境变量，例如 export PATH=\$PATH:/home/AIoT/share/arm-himix100-linux/bin，路径根据实际情况配置。
- 使用不同工具链时，可能会导致编译过程中出现告警导致编译失败，可在 Makefile 文件中修改编译选项。

3.6.2 星闪工具获取

步骤 1 进入 sdk 的 application/bin 目录。

步骤 2 根据 host 版本选择对应版本。

----结束

3.6.3 星闪业务调试

3.6.3.1 星闪设备检测

SDIO 星闪设备检测

SDIO 星闪在使用前 Host 端需要先检测到 SDIO 设备：

- 如果星闪芯片是默认上电的，则 Linux 启动时会检测到 SDIO 设备，显示 “mmc1: new SDIO card at address 0001” 打印信息，说明 SDIO 检测成功。
- 如果星闪芯片是默认不上电的，则需要在星闪驱动中添加上电 SDIO 检测相关流程，再继续执行。

平台可以通过执行 “cat /proc/mci/mci_info” 命令查看是否检测到 SDIO 设备，如图 3-6 所示。

图3-6 SDIO 设备示例

```
~ # cat /proc/mci/mci_info
MCI0: unplugged_disconnected
MCI1: plugged_connected
      Type: SDIO card Mode: HS
      Speed Class: Class 0
      Uhs Speed Grade: Less than 10MB/sec(0h)
      Host work clock: 25MHz
      Card support clock: 25MHz
      Card work clock: 25MHz
      Card error count: 0
MCI2: invalid
.. ..
```

USB 星闪设备检测

USB 星闪在使用前 Host 端需要先检测到 USB 设备：

- 如果主控平台支持 USB 功能且是 build-in 模式，则 Linux 启动时会检测到 USB 设备，显示 “xHCI Host Controller” 打印信息，说明探测到 USB 设备。
- 如果主控平台支持 USB 功能且需要加载驱动，则需要加载 usb 驱动后，再加载 WS73 驱动。

平台可以通过执行 “lsusb” 命令查看是否检测到 USB 设备，如图 3-7 所示。

图3-7 USB 设备示例

```
~ # lsusb
Bus 001 Device 001: ID 1d6b:0002
Bus 001 Device 006: ID ffff:3733 ← USB SLE
Bus 002 Device 001: ID 1d6b:0003
~ "
```

3.6.3.2 加载驱动

步骤 1 将驱动依赖的 bin 文件和 ini 文件分别拷贝至/etc/ws73 目录和/etc 目录（Host 侧默认配置路径）。

步骤 2 将编译出的驱动拷贝至单板，依次加载 plat_soc.ko、sle_soc.ko。

```
$ insmod plat_soc.ko
$ insmod sle_soc.ko
```

步骤 3 串口打印如图 3-8 所示，则说明星闪驱动初始化成功。

图3-8 星闪驱动加载成功

```
[HCC] sle host init finished
```

----结束

3.6.3.3 载入工具

步骤 1 将星闪执行需要的工具 “application/bin/sparklinkd” “application/bin/sparklinkctrl” 复制到/bin 目录下；

步骤 2 在 bin 目录下修改工具的可执行权限：

```
chmod a+x sparklinkd
chmod a+x sparklinkctrl
```

说明

SDK 中发布了部分主控的 sparklinkd 和 sparklinkctrl 工具，若 SDK 中无对应主控所需版本，则需提供编译工具链，并联系相关人员进行编译。

----结束

3.7 Linux 平台 UART 版本移植

3.7.1 ko 编译

步骤 1 解压 WS73 SDK。

将驱动源码包 WS73V100R001C00SPCXXX.tar.gz 放置于服务器上，并进行解压，参考 “2.3 编译 SDK” 中配置编译工具链、内核路径、CPU 架构等，命令如下：

```
$ tar -xzf WS73V100XXX.tar.gz
$ cd WS73V100XXX
```

步骤 2 修改 build/config/ws73_default.config 配置相关参数

1. 编译器是 gcc。

参数	说明	示例
WSCFG_USING_GCC	配置编译器类型为 GCC	WSCFG_USING_GCC=y
WSCFG_CROSS_	配置交叉编译工具链存	WSCFG_CROSS_COMPILE=\$1

参数	说明	示例
COMPILE	放路径	
WSCFG_KERNEL_DIR	配置内核存放路径	WSCFG_KERNEL_DIR=\$2
WSCFG_ARCH_ARM	是否配置 CPU 架构类型为 ARM	WSCFG_ARCH_ARM=y
WSCFG_BUS_UART	配置通信总线类型为 UART	WSCFG_BUS_UART=y

参数\$1 和\$2 需要填具体路径，主控不同路径不一样

2. 编译器是 clang。

参数	说明	示例
WSCFG_USING_LLVM_CLANG	配置编译器类型为 clang	WSCFG_USING_LLVM_CLANG=y
WSCFG_CROSS_COMPILE	配置编译器存放路径	WSCFG_CROSS_COMPILE=\$1
WSCFG_KERNEL_DIR	配置内核存放路径	WSCFG_KERNEL_DIR=\$2
WSCFG_ARCH_ARM	是否配置 CPU 架构类型为 ARM	WSCFG_ARCH_ARM=y
WSCFG_BUS_UART	配置通信总线类型为 UART	WSCFG_BUS_UART=y

参数\$1 和\$2 需要填具体路径，主控不同路径不一样。

3. 配置 WS73 power on 管脚号以及 UART RX （对接 WS73 UART TX）管脚号。

参数	说明	示例
CONFIG_INI_HOST_GPIO	配置 power on 管脚号	CONFIG_INI_HOST_GPIO=\$1
CONFIG_INI_UARTCFG_GPIO	配置 UART RX 管脚号	CONFIG_INI_UARTCFG_GPIO=\$2

参数\$1 和\$2 需要主控侧 bsp 或者硬件确认具体值是多少。

步骤 3 配置 bin 和 ini 文件路径，具体如表 2-4 所示。

步骤 4 WS73E UART 板芯片上电时序适配。

1. 上电要求: WS73E UART 芯片在上电时 (power on 拉高)，如果晶体选择 40M，WS73 UART TX 管脚需要是低电平；晶体选择 24M，要求管脚为高电平。
2. 适配说明: 通常主控上电启动代码初始化时已经把管脚配置为 UART 模式，根据 WS73E UART 芯片上电要求，WS73E 驱动在下载固件前，要求主控的 UART RX PIN 先配置到 GPIO 模式，根据 WS73E 的晶体类型输出对应电平，然后操作 WS73E 的 power on 管脚使 WS73E 重新上电，主控再配置回 UART 模式开始通讯，才能完成正常的固件下载。。WS73E 修改 UART RX 管脚复用关系目前有两种方案：

方案 1: 直接读写寄存器方式修改复用关系。

1. 在 ws73_default.config 中配置
CONFIG_SUPPORT_CHANGE_UART_MUX_BY_REG=y
2. driver/platform/pm/plat_pm_board.c 里修改如下宏的值(参照 Hi3518v300 或者 Hi3798MV320)，每个主控复用关系寄存器配置不一样，需主控侧 BSP 确认

```
334
335 #if defined(CONFIG_SUPPORT_HI3518V300)
336 #define UART_MUX_REG 0x120c0010
337 #define UART_MUX_REG_LEN 0x100
338 #define UART_REG_MUX_MASK 0xffffffff0
339 #define CFG_GPIO_STATE 0x2
340 #define CFG_UART_STATE 0x4
341 #elif defined(CONFIG_SUPPORT_HI3798MV320)
342 #define UART_MUX_REG 0xf8a2106c
343 #define UART_MUX_REG_LEN 0x100
344 #define UART_REG_MUX_MASK 0xffffffff0
345 #define CFG_GPIO_STATE 0x4
346 #define CFG_UART_STATE 0x7
347 #endif
```

软件宏说明：

宏 UART_MUX_REG：请参考主控芯片指导文档，获得 UART RX 管脚的模式配置寄存器。

宏 UART_REG_MUX_MASK：主控一个寄存器可能包含多个管脚的复用关系配置，该宏代表 UART RX 管脚对应的 bit (如上面 0xffffffff0 表示低 4 位为 UART 模式配置)。

宏 CFG_GPIO_STATE: UART_MUX_REG 代表寄存器配置 GPIO 模式需要配置的值。

宏 CFG_UART_STATE: UART_MUX_REG 代表寄存器配置为 UART 模式需要配置的值。

方案 2: 有些主控方案对管脚复用关系添加了 dts 节点, WS73E 驱动软件可以通过操控主控提供的 dts 节点修改复用关系 (比如主控新 115x 芯片)

1. 在 ws73_default.config 中配置 #
CONFIG_SUPPORT_CHANGE_UART_MUX_BY_REG is not set
2. 主控侧 dts 需加上切 UART RX 复用的节点, 同时主控 pinmux 驱动需支持使用 dts 节点切 UART RX 复用关系, 具体参考如下。

WS73 适配时, 需要在对应 host 芯片的 dts 增加其管脚复用的配置, 具体包含如下:

- a. 将 host 芯片的 POWER_ON_SLE 复用为 gpio 模式供 WS73 驱动调用, 使 WS73 初始化上电
- b. 将 host 芯片的 UART_SIN 所在管脚先复用为 gpio 模式供 WS73 驱动调用, 配置 WS73 硬件配置字。
- c. 将 host 芯片的 UART_SIN、UART_SOUT、UART_CTS、UART_RTS 所在管脚复用为 UART 模式, 供 host 芯片与 WS73 通信使用。

以 tiangong1 为例:

在 "chip/tiangong1/dts/xxx_overlay.dts" 中新增两个节点:

```
fragment@x0 {
    target = <&pinctrl_hgpio>;
    __overlay__ {
        pinctrl-names = "default"; /* 固定配置, 不需要修改 */
        pinctrl-0 = <&gpio36_default_state>; /* WS73 上电管脚复用成 gpio 模式 */
    };
};

fragment@x1 {
    target = <&ws73>;
    __overlay__ {
        pinctrl-names = "uart0_pad0", "uart0_pad1"; /* 固定配置, 不需要修改 */
        pinctrl-0 = <&gpio38_default_state>; /* WS73 硬件配置字管脚复用成 gpio 模式 */
        pinctrl-1 = <&uart0_1_default_state>;
    };
};
```



```
};
};
```

说明

xxx_overlay.dts 为客户单板所对应的 overlaydts, fragment@x0 和 fragment@x1 需要按实际的编号来填写。

下载固件前对单板需要进行复位操作, 该功能使用宏

CONFIG_SUPPORT_RESET_DEVICE_IN_INSMOD 隔离, 产品需要配置该宏。

步骤 5 修改 UART 波特率, 每个主控支持的最高 UART 波特率可能不一样, 在 driver/platform/hcc/host/hcc_uart_host.h 中修改 UART 支持的波特率, 修改 HCC_UART_FW_BAUDRATE 宏的值。

```
#if defined(CONFIG_SUPPORT_HI3798MV320)
#define HCC_UART_FW_BAUDRATE UART_BAUD_RATE_4M
#elif defined(CONFIG_SUPPORT_HI3518V300)
#define HCC_UART_FW_BAUDRATE UART_BAUD_RATE_1M
#else
#define HCC_UART_FW_BAUDRATE UART_BAUD_RATE_10M
#endif
```

步骤 6 编译 octty (用户态程序, 用于操控 tty 节点, 内核版本大于等于 4.14 可跳过此步)。

1. Linux 平台: driver/platform/hcc 路径下执行 make, 生成 octty 二进制程序在 out/bin 路径下。
2. android 平台: android 平台下 octty 的编译需要主机侧适配。

步骤 7 修改根目录 Makefile, 不编译 wifi。

将 ALL_CBB_BUILD_TARGETS 和 ALL_CBB_BUILD_TARGETS_ANDROID 改成如下形式

```
43
44 ALL_CBB_BUILD_TARGETS := platform sle
45 LIGHT_CBB_BUILD_TARGETS := platform_light wifi_light
46 ALL_CBB_CLEAN_TARGETS := platform_clean wifi_clean
47 ALL_CBB_BUILD_TARGETS_ANDROID := platform sle
```

步骤 8 在 sdk 根路径下执行 make, 编译生成的 ko 在 out/bin 路径下, bin 文件在 firmware/e 路径下。

----结束

3.7.2 ko 加载

步骤 1 将 ko 和 bin 文件打包到主控的文件系统，ko 和 octty 存放路径 可由主控自定义一般存放在/komod 路径下，bin 和 ini 文件存放路径参考 2.3.1 节的表 2；也可将 ko 和 bin 通过 tftp 方式传进主控板。

步骤 2 升级镜像重启板子后，加载 octty（内核版本大于等于 4.14 不需要）

```
./octty &
```

步骤 3 加载平台星闪 ko

```
insmod plat_soc.ko
```

```
insmod sle_soc.ko
```

----结束

4 常见问题

4.1 驱动加载出错

4.2 wpa_supplicant 启动失败

4.1 驱动加载出错

问题描述

在加载驱动时，出现如下错误：

```
Unknown symbol cfg80211_inform_bss_frame_data (err 0)
Unknown symbol cfg80211_sched_scan_results (err 0)
Unknown symbol cfg80211_scan_done (err 0)
Unknown symbol cfg80211_sched_scan_stopped (err 0)
```

问题原因

加载 wifi_soc.ko 时，找不到 Linux 内核 CFG80211 相关的符号引用。

解决方案

1. 如果 CFG80211 配置没有打开，可参考“3.1.1 配置 CFG80211”配置内核选项，并将重新编译后的内核烧录进目标开发板。
2. 如果 CFG80211 配置为“M”，则需要将主控 Linux 内核目录下的 cfg80211.ko 拷贝至主控板，加载 cfg80211.ko 后，重新加载驱动。

4.2 wpa_supplicant 启动失败

问题描述

加载驱动完成后，启动 wpa_supplicant，但是启动失败。

解决办法

1. 为 wpa_supplicant 添加可执行权限。
2. 执行 wpa_supplicant 命令时，确定指定路径下存在 wpa_supplicant.conf 配置文件，且配置文件正确。
3. wpa_supplicant.conf 文件中会记录 wlan0 存放位置，确保该目录下具有读写权限。