

WS73V100 星闪路由网关移植

用户指南

文档版本 02

发布日期 2024-10-21

前言

概述

本文档介绍使用 WS73E UART 模组硬件和 WS73V100 SDK 在星闪网关路由设备平台上的驱动移植指导，帮助用户在快速了解开发环境后进行开发移植。




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
02	2024-10-21	<ul style="list-style-type: none">更新“3.1.3 平台基础配置”章节内容。更新“3.2.1 工具移植步骤”章节内容。
01	2024-08-05	第一次正式版本发布。

目 录

前言i

1 硬件连接约束1

2 软件开发环境搭建3

2.1 SDK 开发环境简介3

2.2 Python 环境安装4

3 Linux 移植说明6

3.1 SDK 编译环境搭建6

3.1.1 SDK 模块框架6

3.1.2 SDK 目录结构介绍7

3.1.3 平台基础配置10

3.1.4 驱动代码编译15

3.1.5 驱动定制化配置文件配置15

3.1.6 ko 加载17

3.2 Linux 平台蓝牙移植17

3.2.1 工具移植步骤17

3.2.2 蓝牙业务调试22

3.3 Linux 平台星闪移植23

3.3.1 星闪工具获取23

3.3.2 星闪业务调试23

3.3.2.1 加载驱动23

3.3.2.2 载入工具23

3.3.3 星闪 AT 命令使用指南24

3.3.3.1 SLE AT 指令一览表24

3.3.3.2 SLE AT 指令描述	26
3.3.3.2.1 SLE 使能	26
3.3.3.2.2 SLE 去使能	27
3.3.3.2.3 设置 SLE 广播参数	27
3.3.3.2.4 设置 SLE 广播数据	28
3.3.3.2.5 起 SLE 广播	28
3.3.3.2.6 停 SLE 广播	29
3.3.3.2.7 设置扫描参数	29
3.3.3.2.8 使能扫描	30
3.3.3.2.9 关闭扫描	30
3.3.3.2.10 设置本端名称	30
3.3.3.2.11 获取本端名称	31
3.3.3.2.12 设置本端地址	31
3.3.3.2.13 获取本端地址	31
3.3.3.2.14 建立 SLE 连接	32
3.3.3.2.15 星闪逻辑链路更新参数	32
3.3.3.2.16 星闪读取远端 rssi	33
3.3.3.2.17 断开 SLE 连接	33
3.3.3.2.18 设置 SLE PHY	33
3.3.3.2.19 设置 SLE 默认连接参数	34
3.3.3.2.20 设置 SLE 连接传输单个数据包长度	35
3.3.3.2.21 设置 SLE 连接 mcs 传输特性	36
3.3.3.2.22 进行加密配对	36
3.3.3.2.23 移除加密配对	36
3.3.3.2.24 获取配对设备数目	37
3.3.3.2.25 获取配对设备	37
3.3.3.2.26 获取设备配对状态	37
3.3.3.2.27 获取绑定设备	38
3.3.3.2.28 注册服务端	38
3.3.3.2.29 添加服务	39
3.3.3.2.30 添加服务同步	39

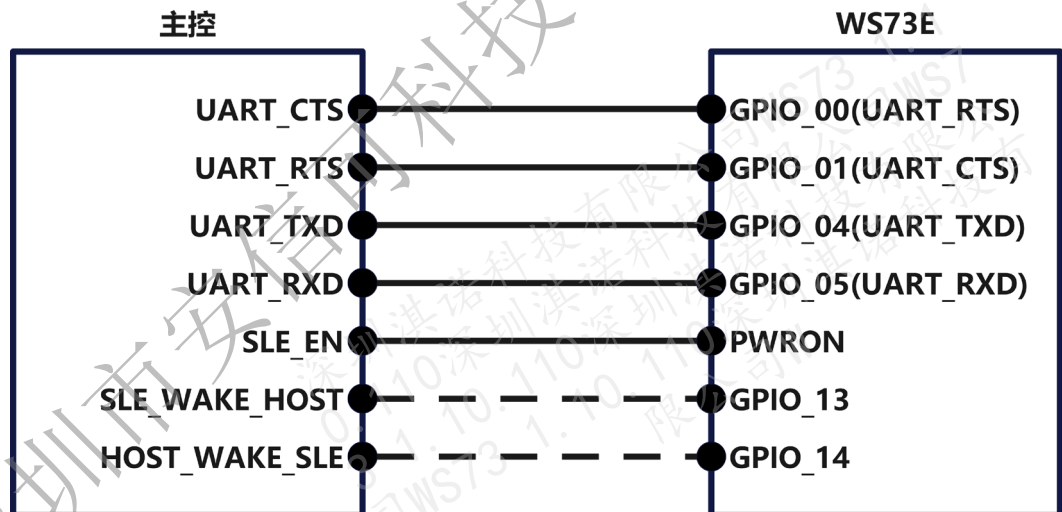
3.3.3.2.31 添加属性	39
3.3.3.2.32 添加属性同步	40
3.3.3.2.33 添加属性描述符	41
3.3.3.2.34 添加属性描述符同步	41
3.3.3.2.35 服务端向客户端发送通知	43
3.3.3.2.36 服务端向客户端通过 uuid 发送通知	43
3.3.3.2.37 服务端发送响应	44
3.3.3.2.38 服务端注册回调	46
3.3.3.2.39 start service	46
3.3.3.2.40 注册 SSAPC 回调函数	46
3.3.3.2.41 发现 service	47
3.3.3.2.42 客户端向服务端写入数据	47
3.3.3.2.43 客户端向服务端发送写请求	48
3.3.3.2.44 客户端发起信息交换	49
3.3.3.2.45 客户端通过 uuid 发送读请求	49
3.3.3.2.46 客户端读取服务端属性数据	50
3.3.3.2.47 SLE 断开所有连接	50
3.3.4 星闪软件开发指南	51
3.3.4.1 错误码	51
3.3.4.2 Device Discovery 接口	52
3.3.4.2.1 概述	52
3.3.4.2.2 开发流程	52
3.3.4.2.3 注意事项	55
3.3.4.3 Connection Manager 接口	55
3.3.4.3.1 概述	55
3.3.4.3.2 开发流程	55
3.3.4.4 SSAP Server 接口	58
3.3.4.4.1 概述	58
3.3.4.4.2 开发流程	58
3.3.4.5 SSAP client 接口	62
3.3.4.5.1 概述	62
3.3.4.5.2 开发流程	62

3.3.4.6 Sample 示例	65
3.3.4.6.1 sle_uuid sample 工程构建指导	65
4 BLE&SLE 配置说明	67
5 常见问题.....	74

1 硬件连接约束

在使用 WS73E UART 总线形态模组时，除满足供电规格外，WS73E 约束主控侧需要保证以下管脚的硬件连接以保证功能的正常驱动（如图 1-1 所示）。

图1-1 WS73E 管脚连接图



主控侧管脚：

- UART_CTS：必选管脚，作为 UART 传输控制信号线。
- UART_RTS：必选管脚，作为 UART 传输控制信号线。
- UART_TXD：必选管脚，作为 UART 数据信号线。
- UART_RXD：必选管脚，作为 UART RX 数据信号线。硬件约束主控侧可以修改该管脚复用为 GPIO 并控制高低电平，以支持 WS73E 模组初始化。
- SLE_EN：必选管脚，用于控制 WS73E 模组的开启与关闭。硬件约束主控提供该管脚给 WS73E 驱动进行控制。

- SLE_WAKE_HOST：可选管脚，低功耗功能相关，当存在 WS73E 模组唤醒主控场景时需要连接该管脚。
- HOST_WAKE_SLE：可选管脚，低功耗功能相关，当存在主控唤醒 WS73E 模组场景时需要连接该管脚。

2

软件开发环境搭建

2.1 SDK 开发环境简介

2.2 Python 环境安装

2.1 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。

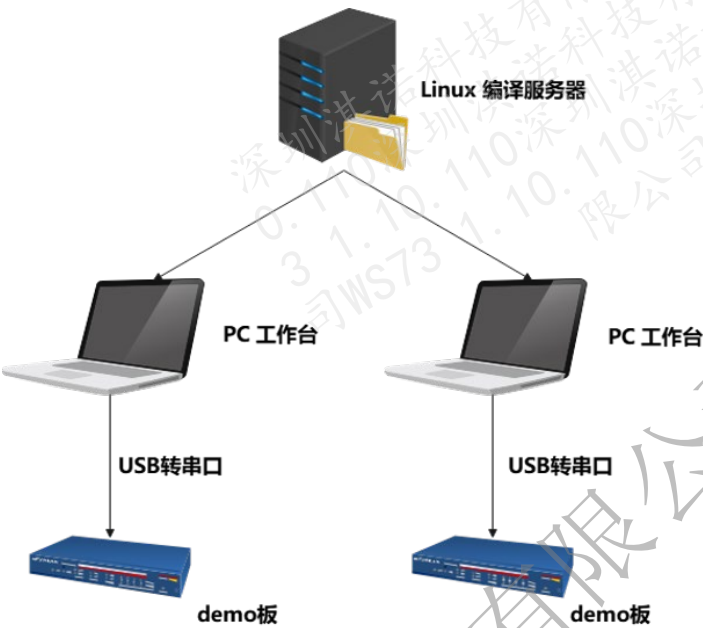
- PC 工作台

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 2-1 所示。

图2-1 SDK 开发环境



2.2 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 2-1，安装依赖库，配置 WS73 的编译环境。

表2-1 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

3 Linux 移植说明

- 3.1 SDK 编译环境搭建
- 3.2 Linux 平台蓝牙移植
- 3.3 Linux 平台星闪移植

3.1 SDK 编译环境搭建

3.1.1 SDK 模块框架

图3-1 WS73 SDK 系统框架

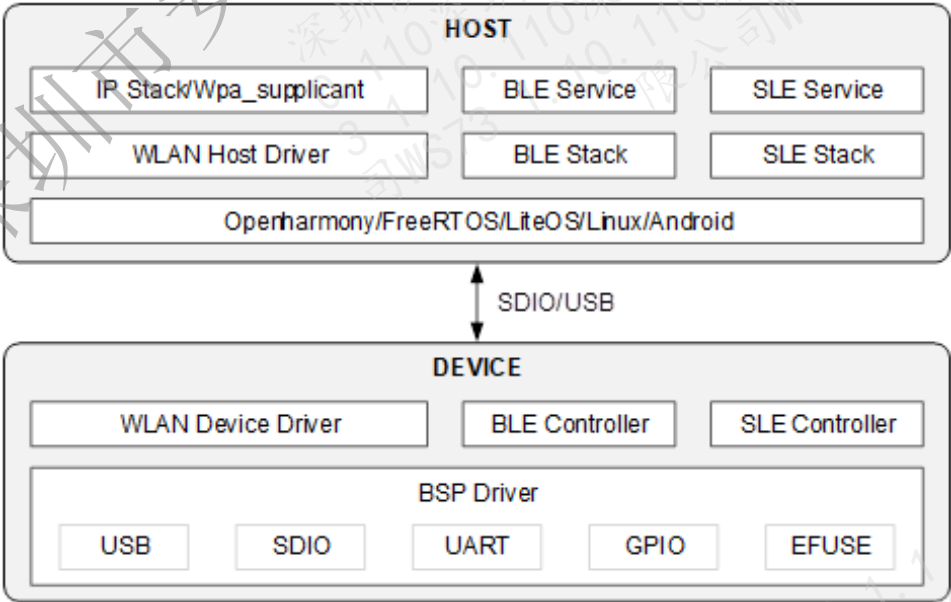


表3-1 WS73 SDK 系统框架说明

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。（WS73E UART 不支持）
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。（WS73E UART 不支持）
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。（WS73E UART 不支持）
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。

3.1.2 SDK 目录结构介绍

说明

SDK 中 firmware 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。

SDK 目录结构如下：

```
.
├── application          # 用户态二进制文件和示例代码
├── └── 厂商 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		
		sle_android	# 星闪Android协议实现，按支持的Android版本发布		
		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		
				libc_sec	# 安全库函数
				hcc	
				octty	# 配合主控与3873通信的传输的用户态进程源码（linux内核版本小于4.14时使用）
				octty.c	# octty源文件
				octty.h	# octty头文件
				Makefile	
		wifi	# WS73 WiFi驱动文件		
		firmware	# WS73 驱动依赖固件		
		e	# 73E版本固件		
		us	# 73U/73S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3-2 WS73 开源代码目录说明

序号	模块名	文件名	功能介绍
----	-----	-----	------

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 rd_server	ble_hid_keyboard _server.c	ble 键盘 hid 服务 sample 代码，主 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_s erver.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_ser ver	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_ser ver	sle_mouse_dis_s erver.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_ser ver	sle_mouse_hid_s erver.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

序号	模块名	文件名	功能介绍
			码，主要实现服务端服务注册和数传等功能。

3.1.3 平台基础配置

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，命令如下：

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

步骤 2 修改编译配置文件 build/config/ws73_default.config 配置平台基础配置项

1. 编译相关配置。

参数	说明	示例
WSCFG_USING_GCC	配置编译器类型为 GCC，需要与 WSCFG_USING_LLVM_CLANG 互斥	WSCFG_USING_GCC=y
WSCFG_USING_LLVM_CLANG	配置编译器类型为 clang，需要与 WSCFG_USING_GCC 配置互斥	# WSCFG_USING_LLVM_CLANG is not set
WSCFG_CROSS_COMPILE	配置交叉编译工具链存放路径	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_ARCH_CUSTOM	是否配置为其他 CPU 架构类型，需要与 WSCFG_ARCH_ARM 互斥	# WSCFG_ARCH_CUSTOM is not set
WSCFG_ARCH_NAME	配置 CPU 架构名称	WSCFG_ARCH_NAME="arm"

参数\$1 和\$2 需要填具体路径，主控不同路径不一样，需要读者按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"。

2. 配置模组总线形态，对于 WS73E UART 总线模组，选择总线类型为 UART。

参数	说明	示例
WSCFG_BUS_SDIO	配置总线类型是否为 SDIO，在 WS73E UART 模组上配置为不使能	# WSCFG_BUS_SDIO is not set
WSCFG_BUS_USB	配置总线类型是否为 USB，在 WS73E UART 模组上配置为不使能	# WSCFG_BUS_USB is not set
WSCFG_BUS_UART	配置通信总线类型为 UART	WSCFG_BUS_UART=y

3. 配置主控侧 SLE_EN 管脚号以及 UART_RXD（对接 WS73 UART_TXD）管脚号。

参数	说明	示例
CONFIG_INI_HOST_GPIO	配置主控侧 SLE_EN 管脚号	CONFIG_INI_HOST_GPIO=\$1
CONFIG_INI_UARTCFG_GPIO	配置主控侧 UART_RXD 管脚号	CONFIG_INI_UARTCFG_GPIO=\$2
CONFIG_GPIO_ADAPTIVE_POWER_ON_LEVEL	配置主控侧 SLE_EN 使能 WS73 工作的工作电平是否自适应读取（需要主控保证加载平台驱动前已将 SLE_EN 电平置为工作电平）	CONFIG_GPIO_ADAPTIVE_POWER_ON_LEVEL=n

参数\$1 和\$2 需要主控侧 bsp 或者硬件确认具体管脚号是多少。

4. 其他基础配置项配置。

参数	说明	示例
CONFIG_DIAG_SUPPORT_UART	配置是否通过 UART 输出日志，在 WS73E UART 模组上配置为不使能	# CONFIG_DIAG_SUPPORT_UART is not set
CONFIG_DEVICE_INIT_STAND	配置是否初始化后关闭 WS73，	CONFIG_DEVICE_INIT_ST

参数	说明	示例
BY	在 WS73E UART 模组上配置为使能 (该配置项在 1.10.108 及以上版本中已删除, 无需配置)	ANDBY=y

步骤 3 配置 bin 和 ini 文件路径, 这些配置决定在单板设备中进行驱动加载时, 寻找 bin 文件和 ini 文件的路径。

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_BSLCALI_PATH	配置 btc_cali.bin 文件路径	CONFIG_FIRMWARE_BSLCALI_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"
CONFIG_PLAT_DFR_OUTPUT_PATH	配置 ws73 panic 文件存在路径	CONFIG_PLAT_DFR_OUTPUT_PATH="/etc/ws73"

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), WS73 UART TX 管脚需要是低电平。
2. 适配说明: 根据 WS73E UART 芯片上电要求, 主控侧在拉高 power on 管脚之前, 需要将 UART RX (对接 WS73 UART TX) 管脚拉低, 而 UART RX 管脚复用关系默认是 UART 模式, 需要通过改复用关系把 UART RX 管脚改为 GPIO 模式, 这样就可通过 GPIO 接口去拉低 WS73 UART TX 管脚。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，每个主控复用关系寄存器配置不一样，需主控侧 BSP 进行确认。

```
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 0xffffffff
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 0xffffffff
345 #define CFG_GPIO_STATE 0x4
346 #define CFG_UART_STATE 0x7
347 #endif
```

方案 2：添加 dts 节点，通过操控 dts 节点修改复用关系（1155 和 1156 方案）

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。

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。

```

#ifdef 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

3.1.4 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。编译生成的 ko、ini 配置文件在 out/bin 路径下，bin 文件在 firmware/e 路径下。

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

3.1.5 驱动定制化配置文件配置

编译生成的 ws73_cfg.ini 为驱动定制化配置文件，在该文件中进行部分 GPIO 相关定制化配置修改：

配置项	配置说明	样例	默认值
wkup_gpio_idx_device	Device 唤醒 Host 使用的 GPIO 管脚号 (Device 侧)。	wkup_gpio_idx_device=10 表示 Device 侧唤醒 Host 侧时，Device 侧使用 0 号引脚	10

配置项	配置说明	样例	默认值
		唤醒。	
wkup_gpio_level	Device 唤醒 Host 使用的 GPIO 电平： 0：低电平唤醒； 1：高电平唤醒。	wkup_gpio_level=1 表示 Device 侧使用高电平唤醒 Host 侧。	1
power_gpio_idx	Host 侧 GPIO 电源管脚号，用于上电/复位。 如果模组的 power_on 引脚没有和 host 的 GPIO 相连，则 reboot 复位功能会报错，需要将值改为-1，通过软件复位恢复 reboot 功能。	power_gpio_idx=40 表示 Host 侧使用 40 号 GPIO 引脚用于上电/复位。	40
wkup_gpio_idx	进行低功耗唤醒时，Host 侧唤醒 device 的 GPIO 管脚号。 未使用相关功能时请将值配为-1，以表示不支持该功能。	wkup_gpio_idx=70 表示 Host 侧使用 70 号 GPIO 引脚用于唤醒 Device 侧。	70
uart_cfg_gpio_idx	WS73E 版本与主控用 uart 传输时需要配置此管脚。 此管脚是主控 uart rx 管脚（对接 73E uart tx 管脚）对应的 gpio 管脚号，每个主控的值不太一样，可通过 config 文件 CONFIG_INI_UARTCFG_GPIO 宏配置。	config 文件中配置 CONFIG_INI_UARTCFG_GPIO=27 则编译生成的 ini 文件中 uart_cfg_gpio_idx=27	27

3.1.6 ko 加载

步骤 1 将 ko、ini 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，bin 和 ini 文件存放路径参考 3.1.3 节的步骤 3 进行配置。

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 加载平台和星闪 ko

```
insmod plat_soc.ko
```

```
insmod sle_soc.ko
```

----结束

3.2 Linux 平台蓝牙移植

3.2.1 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_getpwuid_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 3-3 所示。

表3-3 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 libreadline.so.8
/vendor/bin	dbus 及 bluez 相关工具可执行文件： bluetoothctl dbus-daemon hciconfig hcidtool
/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

3.2.2 蓝牙业务调试

参考《WS73V100 Linux Wi-Fi、BLE 软件开发指南》。

3.3 Linux 平台星闪移植

3.3.1 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

3.3.2 星闪业务调试

3.3.2.1 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（以 Host 侧默认配置路径为例，也可以根据 3.1.3 节中配置的 bin 和 ini 文件实际路径进行存放）。

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 3-2 所示，则说明星闪驱动初始化成功。

图3-2 星闪驱动加载成功

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

----结束

3.3.2.2 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

3.3.3 星闪 AT 命令使用指南

3.3.3.1 SLE AT 指令一览表

指令	描述
AT+SLEENABLE	SLE 使能
AT+SLEDISABLE	SLE 去使能
AT+SLESETADVPAR	设置 SLE 广播参数
AT+SLERMVADV	删除广播参数
AT+SLESETADVDATA	设置指令
AT+SLESTARTADV	起 SLE 广播
AT+SLESTOPADV	停 SLE 广播
AT+SLESTARTSCAN	启动扫描
AT+SLESTOPSCAN	关闭扫描
AT+SLESETNAME	设置本端名称
AT+SLEGETNAME	获取本端名称
AT+SLESETADDR	设置本端地址
AT+SLEGETADDR	获取本端地址
AT+SLECONN	建立 SLE 连接
AT+SLEDISCONN	断开 SLE 连接
AT+SLESETPHY	设置 SLE PHY
AT+SLESETDEFAULTCONNP	设置 SLE 默认连接参数
AT+SLESETDATALEN	设置 SLE 连接传输单个数据包长度
AT+SLESETMCS	设置 SLE 连接 mcs 传输特性
AT+SLEPAIR	进行加密配对

指令	描述
AT+SLEUNPAIR	移除加密配对
AT+SLEGETPAIREDNUM	获取配对设备数目
AT+SLEGETPAIRDEV	获取配对设备
AT+SLEGETPAIRSTA	获取配对状态
AT+SLEGETBONDDEV	获取绑定设备状态
AT+SLECONNPARDUPD	星闪逻辑链路更新参数
AT+SLEReadPEERRSSI	读取对端 rssi
AT+SSAPSADDSRV	注册服务端
AT+SSAPSDELALLSRV	去注册服务端
AT+SSAPSADDSERV	添加服务
AT+SSAPSSYNCADDSERV	添加服务同步
AT+SSAPSADDPROPERTY	添加属性
AT+SSAPSSYNCADDPROPERTY	添加属性同步
AT+SSAPSADDDESCR	添加属性描述符
AT+SSAPSSYNCADDDESCR	添加属性描述符同步
AT+SSAPSSTARTSERV	start service
AT+SSAPSSNDNTFY	服务端向客户端发送通知
AT+SSAPSNTFYBYUUID	服务端向客户端通过 uuid 发送通知
AT+SSAPSSNDRESP	服务端向客户端发送响应
AT+SSAPSREGCBK	服务端注册回调函数
AT+SSAPCREGCBK	注册 SSAPC 回调函数
AT+SSAPCFNDSTRU	发现 service
AT+SSAPCWRITECMD	客户端向服务端写入数据
AT+SSAPCWITEREQ	客户端向服务端发送写请求

指令	描述
AT+SSAPCEXCHINFO	客户端发起信息交换
AT+SSAPCREADBYUUID	客户端通过 uuid 发送读请求
AT+SSAPCREADREQ	客户端读取服务端属性数据
AT+SLESETSCANPAR	设置扫描参数
AT+SLEDISCONNALL	SLE 断开所有连接

3.3.3.2 SLE AT 指令描述

使用命令之前的配置：

- 加载平台和星闪驱动，komod 为驱动文件所在路径

```
insmod /komod/plat_soc.ko
insmod /komod/sle_soc.ko
```

- 可执行程序增加执行权限，/bin 为可执行文件所在路径

```
cd /bin
chmod +x sparklinkd
chmod +x sparklinkctrl
```

- 进入视图

```
cd /bin
sparklinkd&
sparklinkctrl
```

3.3.3.2.1 SLE 使能

设置指令	AT+SLEENABLE
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	-
示例	AT+SLEENABLE
注意	-

3.3.3.2.2 SLE 去使能

设置指令	AT+SLEDISABLE
响应	<ul style="list-style-type: none">成功: OK失败: ERROR
参数说明	-
示例	AT+SLEDISABLE
注意	-

3.3.3.2.3 设置 SLE 广播参数

设置指令	AT+SLESETADVPAR=<announce_handle>,<announce_mode>,<announce_interval_min>,<announce_interval_max>,<own_addr_type>,<own_addr_addr>,<peer_addr_type>,<peer_addr_addr>,[tx_power]
响应	<ul style="list-style-type: none">成功: OK失败: ERROR
参数说明	<ul style="list-style-type: none"><announce_handle>: 设备公开句柄, 取值范围[0, 0x10]<announce_mode>: 设备公开类型<ul style="list-style-type: none">0: 不可连接不可扫描1: 可连接不可扫描2: 不可连接可扫描3: 不可连接可扫描<announce_interval_min>: 最小设备公开周期, 0x000020~0xffff, 单位 125μs<announce_interval_max>: 最大设备公开周期, 0x000020~0xffff, 单位 125μs<own_addr_type>: SLE 本端地址类型<ul style="list-style-type: none">0: 公有地址,6: 随机地址<own_addr_addr>: SLE 本端设备地址

	<ul style="list-style-type: none"> • <peer_addr_type>: SLE 对端设备地址类型 0: 公有地址, 6 : 随机地址 • <peer_addr_addr>: SLE 对端设备地址 • [tx_power]: 功率, 取值范围: -127~20, 或者 127, (缺省参数, 默认值为 0)
示例	AT+SLESETADVPAR=1,3,200,200,0,000000000000,0,000000000000
注意	此命令需在 SLE 使能 AT+SLEENABLE 后下发。

3.3.3.2.4 设置 SLE 广播数据

设置指令	AT+SLESETADVDATA=<adv_handle>,<announce_data_len>,<seek_rsp_data_len>,<announce_data>,<seek_rsp_data>
响应	<ul style="list-style-type: none"> • 成功: OK • 失败: ERROR
参数说明	<ul style="list-style-type: none"> • <adv_handle>: 广播 handle, 取值范围[0, 0x10] • <announce_data_len>: 设备公开数据长度。 • <seek_rsp_data_len>: 扫描响应数据长度。 • <announce_data>: 设备公开数据 (hex 类型字符串, 最大长度 521 个字符)。 • <seek_rsp_data>: 扫描响应数据 (hex 类型字符串, 最大长度 521 个字符)。
示例	AT+SLESETADVDATA=1,10,4,aabbccddeeff11223344,11224455
注意	此命令需在 SLE 使能 AT+SLEENABLE 后下发。

3.3.3.2.5 起 SLE 广播

设置指令	AT+SLESTARTADV=<adv_enable>
响应	<ul style="list-style-type: none"> • 成功: OK • 失败: ERROR

参数说明	<adv_handle>: adv handle, 取值范围[0, 0x10]
示例	AT+SLESTARTADV=1
注意	此命令需在 SLE 使能 AT+SLEENABLE 后下发。

3.3.3.2.6 停 SLE 广播

设置指令	AT+SLESTOPADV=<adv_handle>
响应	<ul style="list-style-type: none">成功: OK失败: ERROR
参数说明	<adv_handle>: adv handle
示例	AT+SLESTOPADV=1
注意	此命令需在 SLE 起广播 AT+SLESTARTADV=1 后下发。

3.3.3.2.7 设置扫描参数

设置指令	AT+SLESETSCANPAR=<scan_type>,<scan_interval>,<scan_window>
响应	<ul style="list-style-type: none">成功: OK失败: ERROR
参数说明	<ul style="list-style-type: none"><scan_type>: 扫描类型 0: 被动扫描 1: 主动扫描<scan_interval>: 扫描间隔, 取值范围[0x14, 0xFFFF], 单位 125μs<scan_window>: 扫描窗口, 取值范围[0x14, 0xFFFF], 单位 125μs
示例	AT+SLESETSCANPAR=0,0x48,0x48
注意	此命令需在 SLE 起扫描 AT+SLESTARTSCAN 前下发

3.3.3.2.8 使能扫描

设置指令	AT+SLESTARTSCAN
响应	<ul style="list-style-type: none">成功: OK失败: ERROR
参数说明	-
示例	AT+SLESTARTSCAN
注意	-

3.3.3.2.9 关闭扫描

设置指令	AT+SLESTOPSCAN
响应	<ul style="list-style-type: none">成功: OK失败: ERROR
参数说明	-
示例	AT+SLESTOPSCAN
注意	-

3.3.3.2.10 设置本端名称

设置指令	AT+SLESETNAME
响应	<ul style="list-style-type: none">成功: OK失败: ERROR
参数说明	<ul style="list-style-type: none"><len>: name 长度。<name>: 名字。
示例	AT+SLESETNAME=7,SDKTEST
注意	-

3.3.3.2.11 获取本端名称

设置指令	AT+SLEGETNAME
响应	<ul style="list-style-type: none">成功: OK失败: ERROR
参数说明	-
示例	AT+SLEGETNAME
注意	-

3.3.3.2.12 设置本端地址

设置指令	AT+SLESETADDR
响应	<ul style="list-style-type: none">成功: OK失败: ERROR
参数说明	<addr_type>: 地址类型, 当前仅支持联盟分配地址标识-0 <addr>: 地址
示例	AT+SLESETADDR=0,0x0000000000001
注意	-

3.3.3.2.13 获取本端地址

设置指令	AT+SLEGETADDR
响应	<ul style="list-style-type: none">成功: OK失败: ERROR
参数说明	-
示例	AT+SLEGETADDR
注意	-

3.3.3.2.14 建立 SLE 连接

设置指令	AT+SLECONN=<sle_addr_type>,<sle_addr>
响应	<ul style="list-style-type: none">成功：OK失败：ERROR 连接成功后会打印[connected]字样以及对端设备地址与 handle 值。
参数说明	<ul style="list-style-type: none"><sle_addr_type>：SLE 设备地址类型。 取值：<ul style="list-style-type: none">0：公有地址。6：随机地址。<sle_addr>：SLE 设备地址。
示例	AT+SLECONN=0,0x00000000000000
注意	-

3.3.3.2.15 星闪逻辑链路更新参数

设置指令	AT+SLECONNPARUPD=<conn_id>,<interval_min>,<interval_max>,<max_latency>,<supervision_timeout>
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	<ul style="list-style-type: none"><conn_id>：连接 ID<interval_min>：链路调度最小间隔，取值范围[0x0020, 0x32000]，单位 125μs<interval_max>：链路调度最大间隔，取值范围[0x0020, 0x32000]，单位 125μs<max_latency>：延迟周期，单位 slot（该值表示在设置值的周期内可以不回复，为 0 时表示每包都需回复）<supervision_timeout>：超时时间，单位 10ms
示例	AT+SLECONNPARUPD=0,20,20,0,500
注意	-

3.3.3.2.16 星闪读取远端 rssi

该命令的作用：读取当前 SLE ACB 链路的 RSSI 信号强度，RSSI (Received Signal Strength Indicator) 是接收信号的强度指示。

设置指令	AT+SLEREADPEERRSSI=<conn_id>
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	<conn_id>：连接 ID。
示例	AT+SLEREADPEERRSSI=0
注意	-

3.3.3.2.17 断开 SLE 连接

设置指令	AT+SLEDISCONN=<sle_addr_type>,<sle_addr>
响应	<ul style="list-style-type: none">成功：OK失败：ERROR 连接成功后会打印[disconnected]字样以及对端设备地址与 handle 值。
参数说明	<ul style="list-style-type: none"><sle_addr_type>：SLE 设备地址类型。 取值：<ul style="list-style-type: none">0：公有地址。6：随机地址。<sle_addr>：SLE 设备地址。
示例	AT+SLEDISCONN=0,000000000000
注意	-

3.3.3.2.18 设置 SLE PHY

默认 1M 4M 暂时不支持

该命令作用：某些场景下需要高的传输速率，此时就通过设置 tx_phy、rx_phy 参数为 2M。如果两端都支持 2M,才能设置成功。

设置指令	AT+SLESETPHY=<conn_id>,<tx_phy>,<rx_phy>
响应	<ul style="list-style-type: none"> 成功：OK 失败：ERROR
参数说明	<ul style="list-style-type: none"> <conn_id>：连接 id <tx_phy>：tx phy 值 <ul style="list-style-type: none"> 0：1M PHY 1：2M PHY 2：4M PHY <rx_phy>：tx phy 值 <ul style="list-style-type: none"> 0：1M PHY 1：2M PHY 2：4M PHY
示例	AT+SLESETPHY=0,1,1
注意	-

3.3.3.2.19 设置 SLE 默认连接参数

设置指令	AT+ SLESETDEFAULTCONNP =<enable_filter_policy>,<initiate_phys>,<gt_negotiate>,<scan_interval>,<scan_window>,<max_interval>,<min_interval>,<timeout>
响应	<ul style="list-style-type: none"> 成功：OK 失败：ERROR
参数说明	<ul style="list-style-type: none"> <enable_filter_policy>：是否打开链路过滤 <initiate_phys>：链路扫描带宽 <ul style="list-style-type: none"> 1：1M PHY 2：2M PHY <gt_negotiate>：链路建立时是否进行 G 和 T 交互 <scan_interval>：扫描对端设备的 interval

	<p>最小值：0x14</p> <p>最大值：无上限</p> <ul style="list-style-type: none">• <scan_window>：扫描对端设备的 windows <p>最小值：0x14</p> <p>最大值：传入的<scan_interval></p> <ul style="list-style-type: none">• <max_interval>：链路最大调度 interval <p>最大值：32000</p> <ul style="list-style-type: none">• <min_interval>：链路最小调度 interval <p>最小值：10</p> <ul style="list-style-type: none">• <timeout>：链路超时时间(N×10ms) <p>最小值：10</p> <p>最大值：3200</p>
示例	AT+SLESETDEFAULTCONNP=0,1,0x1,0x20,0x20,0x64,0x64,0x1FC
注意	-

3.3.3.2.20 设置 SLE 连接传输单个数据包长度

设置指令	AT+SLESETDATALEN=<conn_id>,<tx_octets>
响应	<ul style="list-style-type: none">• 成功：OK• 失败：ERROR
参数说明	<ul style="list-style-type: none">• <conn_id>：SLE 连接 id• <tx_octets>：传输单个数据包最大长度 <p>WS73E 最大值：1524</p> <p>WS73US 最大值：254</p>
示例	AT+SLESETDATALEN=0,251
注意	传输单个数据包最大长度的最终值为主机、从机中的较小值

3.3.3.2.21 设置 SLE 连接 mcs 传输特性

设置指令	AT+SLESETMCS=<conn_id>,<mcs>
响应	<ul style="list-style-type: none">成功: OK失败: ERROR
参数说明	<ul style="list-style-type: none"><conn_id>: SLE 连接 id<mcs>: conn_id 对应的连接的传输方式
示例	AT+SLESETMCS=0,14
注意	-

3.3.3.2.22 进行加密配对

设置指令	AT+SLEPAIR=<sle_addr_type>,<sle_addr>
响应	<ul style="list-style-type: none">成功: OK失败: ERROR
参数说明	<ul style="list-style-type: none"><sle_addr_type>: SLE 设备地址类型 取值:<ul style="list-style-type: none">0: 公有地址6: 随机地址<sle_addr>: SLE 设备地址
示例	AT+SLEPAIR=0,000000000000
注意	需在 SLE 建立连接以后, 和对端启动加密配对

3.3.3.2.23 移除加密配对

设置指令	AT+SLEUNPAIR=<sle_addr_type>,<sle_addr>
响应	<ul style="list-style-type: none">成功: OK失败: ERROR
参数说明	<ul style="list-style-type: none"><sle_addr_type>: SLE 设备地址类型。

	取值： <ul style="list-style-type: none"> - 0：公有地址。 - 6：随机地址。 • <sle_addr>：SLE 设备地址。
示例	AT+SLEUNPAIR=0,0000000000000
注意	-

3.3.3.2.24 获取配对设备数目

设置指令	AT+SLEGETPAIREDNUM
响应	<ul style="list-style-type: none"> • 成功：OK • 失败：ERROR
参数说明	-
示例	AT+SLEGETPAIREDNUM
注意	-

3.3.3.2.25 获取配对设备

设置指令	AT+SLEGETPAIRDEV
响应	<ul style="list-style-type: none"> • 成功：OK • 失败：ERROR
参数说明	-
示例	AT+SLEGETPAIRDEV
注意	--

3.3.3.2.26 获取设备配对状态

设置指令	AT+SLEGETPAIRSTA=<sle_addr_type>,<sle_addr>
------	---

响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	<ul style="list-style-type: none">< sle_addr_type >: SLE 设备地址类型 0: 公有地址, 6: 随机地址<sle_addr>: SLE 设备地址
示例	AT+SLEUNPAIR=0,000000000000
注意	-

3.3.3.2.27 获取绑定设备

设置指令	AT+SLEGETBONDDEV
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	-
示例	AT+SLEGETBONDDEV
注意	-

3.3.3.2.28 注册服务端

设置指令	AT+SSAPSADDSRV=<uuid>
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	-
示例	AT+SSAPSADDSRV=0x1234
注意	-

3.3.3.2.29 添加服务

注册成“次要服务”无主要影响。现在都是主要服务，次要服务的概念弱化，因此通常情况下注册成主要服务。

设置指令	AT+SSAPSADDSERV=<uuid>,<is_primary>
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	<ul style="list-style-type: none"><uuid>：SSAP 服务 UUID。<is_primary>：是否是首要服务。
示例	AT+SSAPSADDSERV=0x2222,1
注意	功能使用 AT+SSAPSSYNCADDSERV 替代，推荐使用 AT+SSAPSSYNCADDSERV 添加服务。

3.3.3.2.30 添加服务同步

设置指令	AT+SSAPSSYNCADDSERV=<uuid>,<is_primary>
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	<ul style="list-style-type: none"><uuid>：SSAP 服务 UUID。<is_primary>：是否是首要服务。
示例	AT+SSAPSSYNCADDSERV=0x2222,1
注意	-

3.3.3.2.31 添加属性

设置指令	AT+SSAPSADDPROPERTY=<service_handle>,<uuid>,<permissions>,<operate_indication>,<value_len>,<value>
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	<ul style="list-style-type: none"><service_handle>：服务的 handle。

	<ul style="list-style-type: none">• <uuid>: SSAP 特征 UUID。• <permissions>: 特征权限。• <operate_indication>: 操作指示。• <value_len>: 响应的数据长度。• <value>: 响应的数据。
示例	AT+SSAPSADDPROPERTY=1,0x2323,5,5,2,0x1234
注意	功能使用 AT+SSAPSSYNCADDPROPERTY 替代, 推荐使用 AT+SSAPSSYNCADDPROPERTY 添加属性

3.3.3.2.32 添加属性同步

设置指令	AT+SSAPSSYNCADDPROPERTY=<service_handle>,<uuid>,<permissions>,<operate_indication>,<value_len>,<value>
响应	<ul style="list-style-type: none">• 成功: OK• 失败: ERROR
参数说明	<ul style="list-style-type: none">• <service_handle>: 服务的 handle。• <uuid>: SSAP 特征 UUID。• <permissions>: 特征权限。<ul style="list-style-type: none">0x01: 可读。0x02: 可写。0x04: 需要加密。0x08: 需要认证。0x10: 需要授权。• <operate_indication>: 操作指示。<ul style="list-style-type: none">0x01: 数据值可被读取。0x02: 数据值可被写入, 写入后无反应。0x04: 数据值可被写入, 写入后产生反馈给客户端。0x08: 数据值通过通知的方式传递给客户端。0x10: 数据值通过指示的方式传递给客户端。0x20: 数据值可携带在广播中。

	0x100: 数据值说明描述符可被写入。 <ul style="list-style-type: none">• <value_len>: 响应的数据长度。• <value>: 响应的数据。
示例	AT+SSAPSSYNCADDPROPERTY=1,0x2323,5,5,2,0x1234
注意	-

3.3.3.2.33 添加属性描述符

设置指令	AT+SSAPSADDDESCR=<service_handle>,<property_handle>,<u uid>,<permissions>,<operate_indication>,<type>,<value_len>,<va lue>
响应	<ul style="list-style-type: none">• 成功: OK• 失败: ERROR
参数说明	<ul style="list-style-type: none">• <service_handle>: 服务 handle。• <property_handle>: 属性 handle。• <uuid>: SSAP 描述符 UUID。• <permissions>: 特征权限。• <operate_indication>: 操作指示。• <type>: 描述符类型。• <value_len>: 数据长度。• <value>: 数据。
示例	AT+SSAPSADDDESCR=1,2,0x3333,5,5,2,2,0x0200
注意	功能使用 AT+SSAPSSYNCADDDESCR 替代, 推荐使用 AT+SSAPSSYNCADDDESCR 添加属性描述符。

3.3.3.2.34 添加属性描述符同步

设置指令	AT+SSAPSSYNCADDDESCR=<service_handle>,<property_handle >,<uuid>,<permissions>,<operate_indication>,<type>,<value_len>, <value>
响应	<ul style="list-style-type: none">• 成功: OK

	<ul style="list-style-type: none">失败：ERROR
参数说明	<ul style="list-style-type: none"><service_handle>：服务 handle。<property_handle>：属性 handle。<uuid>：SSAP 描述符 UUID。<permissions>：特征权限。 0x01：可读。 0x02：可写。 0x04：需要加密。 0x08：需要认证。 0x10：需要授权。<operate_indication>：操作指示。 0x01：数据值可被读取。 0x02：数据值可被写入，写入后无反应。 0x04：数据值可被写入，写入后产生反馈给客户端。 0x08：数据值通过通知的方式传递给客户端。 0x10：数据值通过指示的方式传递给客户端。 0x20：数据值可携带在广播中。 0x100：数据值说明描述符可被写入。<type>：描述符类型。 0：特征值。 1：属性说明描述符。 2：客户端配置描述符。 3：服务端配置描述符。 4：格式描述符。 5：服务管理保留描述符，0x05–0x1F。 0xFF：厂商自定义描述符。<value_len>：数据长度。<value>：数据。
示例	AT+SSAPSSYNCADDDESCR=1,2,0x3333,5,5,2,2,0x0200

注意	-
----	---

3.3.3.2.35 服务端向客户端发送通知

设置指令	AT+SSAPSSNDNTFY=<conn_id>,<handle>,<type>,<value_len>,<value>
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	<ul style="list-style-type: none"><conn_id>：服务 handle。<handle>：属性 handle。<type>：SSAP 特征类型。<ul style="list-style-type: none">0：特征值。1：属性说明描述符。2：客户端配置描述符。3：服务端配置描述符。4：格式描述符。5：服务管理保留描述符，0x05-0x1F。0xFF：厂商自定义描述符。<value_len>：数据长度。<value>：数据。
示例	AT+SSAPSSNDNTFY=0,2,0,2,0x0200
注意	-

3.3.3.2.36 服务端向客户端通过 uuid 发送通知

设置指令	AT+SSAPSNTFYBYUUID=<conn_id>,<uuid>,<start_hdl>,<end_hdl>,<type>,<value_len>,<value>
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	<ul style="list-style-type: none"><conn_id>：服务 handle。

	<ul style="list-style-type: none">• <uuid>: 属性 uuid。• <start_hdl>: 开始句柄。• <end_hdl>: 结束句柄。• <type>: SSAP 特征类型。 0: 特征值。 1: 属性说明描述符。 2: 客户端配置描述符。 3: 服务端配置描述符。 4: 格式描述符。 5: 服务管理保留描述符, 0x05–0x1F。 0xFF: 厂商自定义描述符。• <value_len>: 数据长度。• <value>: 数据。
示例	AT+SSAPSNTRYBYUUID=0,0x1234,0,0xFFFF,0,2,0x0200
注意	-

3.3.3.2.37 服务端发送响应

设置指令	AT+SSAPSSNDRESP=<conn_id>,<request_id>,<status>,<value_len>,<value>
响应	<ul style="list-style-type: none">• 成功: OK• 失败: ERROR
参数说明	<ul style="list-style-type: none">• <conn_id>: 服务 handle。• <request_id>: 请求 id。• <status>: 发送响应原因。• <value_len>: 数据长度。• <value>: 数据。
示例	AT+SSAPSSNDRESP=0,0,1,2,0x0200
注意	-

发送响应原因说明:

```
typedef enum {  
    ERRCODE_SSAP_INVALID_PDU = ERRCODE_SLE_SSAP_BASE + 0x01,          /*  
    服务端接收的 PDU 无效*/  
    ERRCODE_SSAP_PDU_NOT_SUPPORT = ERRCODE_SLE_SSAP_BASE + 0x02,  
    /*服务端不支持处理接收的PDU*/  
    ERRCODE_SSAP_UNKNOW = ERRCODE_SLE_SSAP_BASE + 0x03,  
    /*服务端执行请求时发生未知错误*/  
    ERRCODE_SSAP_INVALID_HANDLE = ERRCODE_SLE_SSAP_BASE + 0x04,  
    /*请求中的句柄无效*/  
    ERRCODE_SSAP_INSUFFICIENT_RESOURCES = ERRCODE_SLE_SSAP_BASE + 0x05,  
    /*服务端没有足够资源完成请求*/  
    ERRCODE_SSAP_PROHIBIT_READING = ERRCODE_SLE_SSAP_BASE + 0x06,  
    /*服务端禁止客户端读取值*/  
    ERRCODE_SSAP_PROHIBIT_WRITE = ERRCODE_SLE_SSAP_BASE + 0x07,  
    /*服务端禁止客户端写入值*/  
    ERRCODE_SSAP_CLIENT_NOT_AUTHENTICATED = ERRCODE_SLE_SSAP_BASE + 0x08,  
    /*客户端未经过认证*/  
    ERRCODE_SSAP_CLIENT_NOT_AUTHORIZATION = ERRCODE_SLE_SSAP_BASE + 0x09,  
    /*客户端未被授权*/  
    ERRCODE_SSAP_BEARER_NOT_ENCRYPTED = ERRCODE_SLE_SSAP_BASE + 0x0A,  
    /*传输 PDU 的承载未加密*/  
    ERRCODE_SSAP_ENTRIES_NOT_FOUND = ERRCODE_SLE_SSAP_BASE + 0x0B,  
    /*服务端未找到对应条目*/  
    ERRCODE_SSAP_DATA_NOT_FOUND = ERRCODE_SLE_SSAP_BASE + 0x0C,  
    /*服务端未找到对应类型数据*/  
    ERRCODE_SSAP_INCORRECT_DATA_TYPE = ERRCODE_SLE_SSAP_BASE + 0x0D,  
    /*客户端发送写入数据类型不符的错误*/  
    ERRCODE_SSAP_INCORRECT_DATA_VALUE = ERRCODE_SLE_SSAP_BASE + 0x0E,  
    /*客户端发送写入值不符的错误*/  
    ERRCODE_SSAP_VALUE_OUT_OF_RANGE = ERRCODE_SLE_SSAP_BASE + 0x0F,  
    /*客户端写入的值超出范围*/  
    ERRCODE_SSAP_UPPERLAYER_APPLICATION_ERROR_MIN =  
    ERRCODE_SLE_SSAP_BASE + 0xAF,          /*预留给上层协议定义应用错误*/  
    ERRCODE_SSAP_UPPERLAYER_APPLICATION_ERROR_MAX =
```

```
ERRCODE_SLE_SSAP_BASE + 0xFF, /*预留给上层协议定义应用错误*/  
} errcode_sle_ssap_t;
```

3.3.3.2.38 服务端注册回调

设置指令	AT+SSAPSREGCBK
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	-
示例	AT+SSAPSREGCBK
注意	-

3.3.3.2.39 start service

设置指令	AT+SSAPSSTARTSERV=<service_handle>
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	<service_handle>：服务 handle。
示例	AT+SSAPSSTARTSERV=1
注意	-

3.3.3.2.40 注册 SSAPC 回调函数

设置指令	AT+SSAPCREGCBK
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	-
示例	AT+SSAPCREGCBK
注意	-

3.3.3.2.41 发现 service

设置指令	AT+SSAPCFNDSTRU=<client_id>,<conn_id>,<type>,<uuid>,<start_hdl>,<end_hdl>
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	<ul style="list-style-type: none"><client_id>：客户端 id<conn_id>：连接 id<type>：查找类型，取值如下： 0：服务结构。 1：首要服务。 3：属性。
示例	AT+SSAPCFNDSTRU=0,0,1,0x1234,0,0xff
注意	-

查找类型字段说明：

```
typedef enum {
    SSAP_FIND_TYPE_SERVICE_STRUCTURE = 0x00,    /*服务结构*/
    SSAP_FIND_TYPE_PRIMARY_SERVICE    = 0x01,    /*首要服务*/
    SSAP_FIND_TYPE_REFERENCE_SERVICE = 0x02,    /*引用服务*/
    SSAP_FIND_TYPE_PROPERTY           = 0x03,    /*属性*/
    SSAP_FIND_TYPE_METHOD              = 0x04,    /*方法*/
    SSAP_FIND_TYPE_EVENT               = 0x05,    /*事件*/
} ssap_find_type_t;
```

3.3.3.2.42 客户端向服务端写入数据

设置指令	AT+SSAPCWRITECMD=<client_id>,<conn_id>,<handle>,<type>,<len>,<write_data>
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	<ul style="list-style-type: none"><client_id>：客户端 id。

	<ul style="list-style-type: none">• <conn_id>: 连接 id。• <handle>: 连接 handle。• <type>: 客户端类型, 取值: 0/1/3。 0: 特征值。 1: 属性说明描述符。 3: 服务端配置描述符。• <len>: 写入数据长度。• <write_data>: 写入数据段。
示例	AT+SSAPCWRITECMD=0,0,2,0,2,0x8899
注意	-

3.3.3.2.43 客户端向服务端发送写请求

设置指令	AT+SSAPCWRIREQ=<client_id>,<conn_id>,<handle>,<type>,<len>,<write_data>
响应	<ul style="list-style-type: none">• 成功: OK• 失败: ERROR
参数说明	<ul style="list-style-type: none">• <client_id>: 客户端 id。• <conn_id>: 连接 id。• <handle>: 连接 handle。• <type>: 客户端类型, 取值: 0/1/3。 0: 特征值。 1: 属性说明描述符。 3: 服务端配置描述符。• <len>: 写入数据长度。• <write_data>: 写入数据段。
示例	AT+SSAPCWRIREQ=0,0,2,0,2,0x8899
注意	-

3.3.3.2.44 客户端发起信息交换

设置指令	AT+SSAPCEXCHINFO=<client_id>,<conn_id>,<mtu_size>,<version>
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	<ul style="list-style-type: none"><client_id>：客户端 id。<conn_id>：连接 id。<mtu_size>：ssap 通道 mtu。 最大值：1500<version>：版本号。
示例	AT+SSAPCEXCHINFO=0,0,251,1
注意	-

3.3.3.2.45 客户端通过 uuid 发送读请求

设置指令	AT+SSAPCREADBYUUID=<client_id>,<conn_id>,<uuid>,<type>,<start_hdl>,<end_hdl>
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	<ul style="list-style-type: none"><client_id>：客户端 id。<conn_id>：连接 id。<handle>：连接 handle。<type>：客户端类型，取值：0/1/3。 0：特征值。 1：属性说明描述符。 3：服务端配置描述符。<start_hdl>：开始 handle。<end_hdl>：结束 handle。
示例	AT+SSAPCREADBYUUID=0,0,0x1234,0,0,0xFFFF

注意	-
----	---

3.3.3.2.46 客户端读取服务端属性数据

设置指令	AT+SSAPCREADREQ=<client_id>,<conn_id>,<handle>,<type>
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	<ul style="list-style-type: none"><client_id>：客户端 id（预留参数）。<conn_id>：连接 id。<handle>：连接 handle（连接成功后的回调里会打印）。<type>：客户端类型，取值：0/1/3。<ul style="list-style-type: none">0：特征值。1：属性说明描述符。3：服务端配置描述符。
示例	AT+SSAPCREADREQ=0,0,2,0
注意	读数据时的 handle 需与写入数据时的 handle 一致

3.3.3.2.47 SLE 断开所有连接

设置指令	AT+SLEDISCONNALL
响应	<ul style="list-style-type: none">成功：OK失败：ERROR
参数说明	
示例	AT+SLEDISCONNALL
注意	-

3.3.4 星闪软件开发指南

3.3.4.1 错误码

SLE SDK 使用错误码指示用户当前任务执行结果，如表 3-4 所示。

表3-4 错误码

序号	定义	实际数值	描述
1	ERRCODE_SLE_SUCCESS	0	执行成功错误码。
2	ERRCODE_SLE_CONTINUE	0x80006000	继续执行错误码。
3	ERRCODE_SLE_DIRECT_RETURN	0x80006001	直接返回错误码。
5	ERRCODE_SLE_PARAM_ERR	0x80006002	参数错误错误码。
6	ERRCODE_SLE_FAIL	0x80006003	执行失败错误码。
7	ERRCODE_SLE_TIMEOUT	0x80006004	执行超时错误码。
8	ERRCODE_SLE_UNSUPPORTED	0x80006005	参数不支持错误码。
9	ERRCODE_SLE_GETRECORD_FAIL	0x80006006	获取当前记录失败错误码。
10	ERRCODE_SLE_POINTER_NULL	0x80006007	指针为空错误码。
11	ERRCODE_SLE_NO_RECORD	0x80006008	无记录返回错误码。
12	ERRCODE_SLE_STATUS_ERROR	0x80006009	状态错误错误码。
13	ERRCODE_SLE_NOMEM	0x8000600a	内存不足错误码。
14	ERRCODE_SLE_AUTH_FAIL	0x8000600b	认证失败错误码。
15	ERRCODE_SLE_AUTH_PKEY_MISS	0x8000600c	PIN 码或密钥丢失致认证失败错误码。
16	ERRCODE_SLE_RMT_DEV_DOWN	0x8000600d	对端设备关闭错误码。
17	ERRCODE_SLE_PAIRING_REJECT	0x8000600e	配对拒绝错误码。

序号	定义	实际数值	描述
18	ERRCODE_SLE_BUSY	0x8000600f	系统繁忙错误码。
19	ERRCODE_SLE_NOT_READY	0x80006010	系统未准备好错误码。
20	ERRCODE_SLE_CONN_FAIL	0x80006011	连接失败错误码。
21	ERRCODE_SLE_OUT_OF_RANGE	0x80006012	越界错误码。
22	ERRCODE_SLE_MEMCPY_FAIL	0x80006013	拷贝失败错误码。
23	ERRCODE_SLE_MALLOC_FAIL	0x80006014	内存申请失败错误码。

3.3.4.2 Device Discovery 接口

3.3.4.2.1 概述

Device Discovery 接口是星闪设备发现协议的软件实现，主要功能有 SLE 设备开关、设备管理、设备公开和设备发现。

3.3.4.2.2 开发流程

使用场景

打开 SLE 设备开关是使用 SLE 功能的首要条件，SLE 启动后可进行设备信息管理，包括获取与设置本地设备名称、获取与设置本地设备地址和设置本地设备外观。

- 当 SLE 设备需要进行设备公开时，可先设置设备公开参数、设备公开数据，然后使能设备公开。
- 当 SLE 设备需要进行设备发现时，可先设置设备发现参数，然后使能设备发现，并通过回调函数观察发现到的设备公开数据包。

功能

Device Discovery 提供的接口如表 3-5 所示。

表3-5 Device Discovery 接口描述

接口名称	描述	参数说明	返回信息说明
------	----	------	--------

接口名称	描述	参数说明	返回信息说明
enable_sle	使能 SLE。	-	接口返回值：错误码。
disable_sle	去使能 SLE。	-	接口返回值：错误码。
sle_set_local_addr	设置本地设备地址。	addr：本地设备地址。	接口返回值：错误码。
sle_get_local_addr	获取本地设备地址。	addr：[out]本地设备地址。	接口返回值：错误码。
sle_set_local_name	设置本地设备名称。	name：本地设备名称； len：本地设备名称长度。	接口返回值：错误码。
sle_get_local_name	获取本地设备名称。	name：[out]本地设备名称； len：[inout]入参时为用户预留内存大小，出参时为本地设备名称长度。	接口返回值：错误码。
sle_set_announce_data	设置设备公开数据。	announce_id：设备公开 ID； data：设备公开数据。	接口返回值：错误码。
sle_set_announce_param	设置设备公开参数。	announce_id：设备公开 ID； data：设备公开参数。	接口返回值：错误码。
sle_start_annou	开始设备公	announce_id	接口返回值：错误码。

接口名称	描述	参数说明	返回信息说明
nce	开。	: 设备公开 ID。	
sle_stop_announce	停止设备公开。	announce_id : 设备公开 ID。	接口返回值：错误码。
sle_set_seek_param	设置设备发现参数。	param: 设备发现参数。	接口返回值：错误码。
sle_start_seek	开始设备发现。	-	接口返回值：错误码。
sle_stop_seek	停止设备发现。	-	接口返回值：错误码。
sle_announce_seek_register_callbacks	注册设备公开和设备发现回调函数。	func: 用户回调函数。	接口返回值：错误码。

开发流程

Device Discovery 开发的典型流程如下，具体编程实例可参考“application/samples/bt”。

Terminal Node:

- 步骤 1 调用 enable_sle，打开 SLE 开关。
- 步骤 2 调用 sle_announce_seek_register_callbacks，注册设备公开和设备发现回调函数。
- 步骤 3 调用 sle_set_local_addr，设置本地设备地址。
- 步骤 4 调用 sle_set_local_name，设置本地设备名称。
- 步骤 5 调用 sle_set_announce_param，设置设备公开参数
- 步骤 6 调用 sle_set_announce_data，设置设备公开数据
- 步骤 7 调用 sle_start_announce，启动设备公开。

----结束

Grant Node:

步骤 1 调用 enable_sle，打开 SLE 开关。

步骤 2 调用 sle_announce_seek_register_callbacks，注册设备公开和设备发现回调函数。

步骤 3 调用 sle_set_local_addr，设置本地设备地址。

步骤 4 调用 sle_set_local_name，设置本地设备名称。

步骤 5 调用 sle_set_seek_param，设置设备发现参数。

步骤 6 调用 sle_start_seek，启动设备发现，并在回调函数中获得正在进行设备公开的设备信息。

----结束

3.3.4.2.3 注意事项

若扫描不到设备，请先检查设备是否已在配对设备列表中，或者设备是否已与其他设备配对（此情况下需要先清除设备端配对信息）

3.3.4.3 Connection Manager 接口

3.3.4.3.1 概述

Connection Manager 接口是星闪连接管理协议的软件实现，主要功能有连接、配对和读远端设备 RSSI 值。

3.3.4.3.2 开发流程

使用场景

当设备需要与对端设备建立连接时，可向对端设备发起连接请求。在连接过程中，设备可读取远端设备 RSSI 值。当设备需要更新连接参数时，可向对端设备发起连接参数更新请求。

当设备需要与对端设备配对时，可向对端设备发起配对请求。在配对过程中，可获取当前本端设备与指定对端设备的配对状态。设备可获取当前配对设备数量以及当前配对设备信息链表。

功能

Connection Manager 提供的接口如表 3-6 所示。

表3-6 Connection Manager 接口描述

接口名称	描述	参数说明	返回信息说明
sle_connect_remote_device	向对端设备发起连接请求。	addr: 对端设备地址。	接口返回值: 错误码。
sle_disconnect_remote_device	向对端设备发起断连请求。	addr: 对端设备地址。	接口返回值: 错误码。
sle_update_connect_param	连接参数更新。	params: 连接参数	接口返回值: 错误码。
sle_pair_remote_device	向对端设备发起配对请求 (目前星闪鉴权流程仅支持免输入模式)。	addr: 对端设备地址。	接口返回值: 错误码。
sle_remove_paired_remote_device	与对端设备取消配对。	addr: 对端设备地址。	接口返回值: 错误码。
sle_remove_all_pairs	取消与所有对端设备的配对。	-	接口返回值: 错误码。
sle_get_paired_devices_num	获取配对设备数量。	number: [out]配对设备数量。	接口返回值: 错误码。
sle_get_paired_devices	获取配对设备信息。	addr: [out]设备地址链表; number: [inout]入参时为用户预留内存大小, 出参时为设备数量。	接口返回值: 错误码。
sle_get_pair_state	获取配对状态。	addr: 设备地址; state: [out]配对状态。	接口返回值: 错误码。

接口名称	描述	参数说明	返回信息说明
sle_read_remote_device_rssi	读对端设备 RSSI 值。	conn_id: 连接 id	接口返回值: 错误码。
sle_connection_register_callbacks	注册连接管理回调函数。	func: 用户回调函数。	接口返回值: 错误码。

开发流程

Connection Manager 开发的典型流程如下，具体编程实例可参考 application/samples/bt。

Terminal Node:

- 步骤 1 调用 enable_sle，打开 SLE 开关。
- 步骤 2 调用 sle_announce_seek_register_callbacks，注册设备公开和设备发现回调函数。
- 步骤 3 调用 sle_connection_register_callbacks，注册连接管理回调函数。
- 步骤 4 调用 sle_set_local_addr，设置本地设备地址。
- 步骤 5 调用 sle_set_local_name，设置本地设备名称。
- 步骤 6 调用 sle_set_announce_param，设置设备公开参数
- 步骤 7 调用 sle_set_announce_data，设置设备公开数据
- 步骤 8 调用 sle_start_announce，启动设备公开。

----结束

Grant Node:

- 步骤 1 调用 enable_sle，打开 SLE 开关。
- 步骤 2 调用 sle_announce_seek_register_callbacks，注册设备公开和设备发现回调函数。
- 步骤 3 调用 sle_connection_register_callbacks，注册连接管理回调函数。
- 步骤 4 调用 sle_set_local_addr，设置本地设备地址。
- 步骤 5 调用 sle_set_local_name，设置本地设备名称。

- 步骤 6 调用 sle_set_seek_param，设置设备发现参数。
- 步骤 7 调用 sle_start_seek，启动设备发现，并在回调函数中获得正在进行设备公开的设备信息。
- 步骤 8 调用 sle_connect_remote_device，向对端设备发起连接请求。
- 步骤 9 调用 sle_pair_remote_device，向对端设备发起配对请求。
- 步骤 10 调用 sle_get_paired_devices_num，获取当前配对设备数量
- 步骤 11 调用 sle_get_paired_devices，获取当前配对设备信息。
- 步骤 12 调用 sle_get_pair_state，获取配对状态。

----结束

3.3.4.4 SSAP Server 接口

3.3.4.4.1 概述

SSAP 是 SLE 发送和接收数据的通用规范，支持在两个 SLE 设备间进行数据传输。

3.3.4.4.2 开发流程

使用场景

SSAP Server 主要接收对端的请求和命令，向对端发送响应、通知和指示。

功能

SSAP Server 提供的接口如表 3-7 所示。

表3-7 SSAP Server 接口描述

接口名称	描述	参数说明	返回信息说明
ssaps_register_server	注册 SSAP server。 注：目前只支持注册一个 SSAP server。	app_uuid：应用 UUID 指针； server_id：[out] server id 指针。	接口返回值：错误码。
ssaps_unregister_server	注销 SSAP server。	server_id：server id。	接口返回值：错误码。

接口名称	描述	参数说明	返回信息说明
ssaps_add_service	添加服务。	server_id: server id; service_uuid: 服务UUID; is_primary: 是否是首要服务。	接口返回值: 错误码。
ssaps_add_property	添加特征。	server_id: server id; service_handle: 服务句柄; property: 特征信息。	接口返回值: 错误码。
ssaps_add_descriptor	添加特征描述符。	server_id: server id; service_handle: 服务句柄; property_handle: 特征句柄; descriptor: 描述符信息。	接口返回值: 错误码。
ssaps_add_service_sync	添加服务同步接口, 服务句柄同步返回。	server_id: server id; service_uuid: 服务UUID; is_primary: 是否是首要服务; handle: [out]服务句柄指针。	接口返回值: 错误码。
ssaps_add_property_sync	添加特征同步接口, 特征句柄同步返回。	server_id: server id; service_handle: 服	接口返回值: 错误码。

接口名称	描述	参数说明	返回信息说明
		务句柄; property: 特征; handle: [out]特征句柄指针。	
ssaps_add_descriptor_sync	添加特征描述符同步接口, 特征描述符句柄同步返回。	server_id: server id; service_handle: 服务句柄; property_handle: 特征句柄; descriptor: 特征描述符; handle: [out]特征描述符句柄指针。	接口返回值: 错误码。
ssaps_start_service	启动服务。	server_id: server id; service_handle: 服务句柄。	接口返回值: 错误码。
ssaps_delete_all_services	删除所有服务。	server_id: server id。	接口返回值: 错误码。
ssaps_send_response	发送响应。	server_id: server id; conn_id: 连接 ID; param: 响应参数。	接口返回值: 错误码。
ssaps_notify_indicate	给对端发送通知或指示。明确下发数据的速率要大于连接间隔 30%。	server_id: server id; conn_id: 连接 ID; param: 通知或指示参数。	接口返回值: 错误码。
ssaps_notify_indicate	按照 uuid 给对端发送通知	server_id: server	接口返回值: 错

接口名称	描述	参数说明	返回信息说明
_by_uuid	或指示。明确下发数据的速率要大于连接间隔30%。	id; conn_id: 连接 ID; param: 通知或指示参数。	误码。
ssaps_set_info	在连接之前设置 server 信息。	server_id: server id; info: server 信息。	接口返回值: 错误码。
ssaps_register_callbacks	注册 SSAP server 回调函数。	func: 用户回调函数。	接口返回值: 错误码。

开发流程

SSAP server 开发的典型流程：注册 SSAP server，注册本端属性数据库，接收对端的请求和命令，向对端发送通知和指示，具体编程实例可参考“application/samples/bt”。

- 步骤 1 调用 enable_sle，打开 SLE 开关。
- 步骤 2 调用 ssaps_register_callbacks，注册 SSAP server 回调。
- 步骤 3 调用 sle_announce_seek_register_callbacks，注册设备公开和设备发现回调函数。
- 步骤 4 调用 ssaps_register_server，创建一个 server 实体。
- 步骤 5 调用 ssaps_add_service_sync、ssaps_add_property_sync、ssaps_add_descriptor_sync 和 ssaps_start_service 注册本端属性数据库，每一个服务及其内容添加完成后调用 ssaps_start_service 启动服务。
- 步骤 6 调用 sle_set_local_addr，设置本地设备地址。
- 步骤 7 调用 sle_set_local_name，设置本地设备名称。
- 步骤 8 调用 sle_set_announce_param，设置设备公开参数。
- 步骤 9 调用 sle_set_announce_data，设置设备公开数据。
- 步骤 10 调用 sle_start_announce，启动设备公开。
- 步骤 11 连接建立。

- 步骤 12 接收对端设备的读写请求，当对端设备读写需要授权的特征或描述符时，调用 ssaps_send_response 向对端发送响应并修改本端特征值。
- 步骤 13 当某个特征的客户端特征配置描述符为 0x0001 时，在特征值变化时向对端设备发送通知，当某个特征的客户端特征配置描述符为 0x0002 时，在特征值变化时向对端设备发送指示。

----结束

3.3.4.5 SSAP client 接口

3.3.4.5.1 概述

SSAP 是 SLE 发送和接收数据的通用规范，支持在两个 SLE 设备间进行数据传输。

3.3.4.5.2 开发流程

使用场景

SSAP Client 主要向对端发送请求和命令，接收对端的响应、通知和指示。

功能

SSAP Client 提供的接口如表 3-8 所示。

表3-8 SSAP Client 接口描述

接口名称	描述	参数说明	返回信息说明
ssapc_register_client	注册 SSAP client。 注：目前只支持注册一个 SSAP client。	app_uuid：应用 UUID 指针； client_id：[out] client id 指针。	接口返回值：错误码。
ssapc_unregister_client	注销 SSAP client。	client_id：client id。	接口返回值：错误码。
ssapc_find_structure	查找对端服务、特征和描述符。	client_id：client id； conn_id：连接 ID； param：查找参数。	接口返回值：错误码。
ssapc_read	向对端发送按照 uuid 读取	client_id：client id；	接口返回值：错

接口名称	描述	参数说明	返回信息说明
d_req_by_uuid	请求。	conn_id: 连接 ID; param: 读取参数。	误码。
ssapc_read_req	向对端发送读取请求。	client_id: client id; conn_id: 连接 ID; handle: 句柄; type: 类型。	接口返回值: 错误码。
ssapc_write_req	向对端发送写请求。明确下发数据的速率要大于连接间隔 30%。	client_id: client id; conn_id: 连接 ID; param: 写参数。	接口返回值: 错误码。
ssapc_write_cmd	向对端发送写命令。明确下发数据的速率要大于连接间隔 30%。	client_id: client id; conn_id: 连接 ID; param: 写参数。	接口返回值: 错误码。
ssapc_exchange_info_req	向对端发送交换信息请求。	client_id: client id; conn_id: 连接 ID; param: 交换信息参数。	接口返回值: 错误码。
ssapc_register_callbacks	注册 SSAP client 回调函数。	func: 用户回调函数。	接口返回值: 错误码。

开发流程

SSAP Client 开发的典型流程：注册 SSAP Client，查找对端属性数据库，向对端发送请求和命令，接收对端的通知和指示，具体编程实例可参考 application/samples/bt。

SSAP Server:

- 步骤 1 调用 enable_sle，打开 SLE 开关。
- 步骤 2 调用 ssaps_register_callbacks，注册 SSAP server 回调。
- 步骤 3 调用 sle_announce_seek_register_callbacks，注册设备公开和设备发现回调函数。
- 步骤 4 调用 ssaps_register_server，创建一个 server 实体。

- 步骤 5 调用 `ssaps_add_service_sync`、`ssaps_add_property_sync`、`ssaps_add_descriptor_sync` 和 `ssaps_start_service` 注册本端属性数据库，每一个服务及其内容添加完成后调用 `ssaps_start_service` 启动服务。
- 步骤 6 调用 `sle_set_local_addr`，设置本地设备地址。
- 步骤 7 调用 `sle_set_local_name`，设置本地设备名称。
- 步骤 8 调用 `sle_set_announce_param`，设置设备公开参数。
- 步骤 9 调用 `sle_set_announce_data`，设置设备公开数据。
- 步骤 10 调用 `sle_start_announce`，启动设备公开。
- 步骤 11 连接建立。
- 步骤 12 接收对端设备的读写请求，当对端设备读写需要授权的特征或描述符时，调用 `ssaps_send_response` 向对端发送响应并修改本端特征值。
- 步骤 13 当某个特征的客户端特征配置描述符为 `0x0001` 时，在特征值变化时向对端设备发送通知，当某个特征的客户端特征配置描述符为 `0x0002` 时，在特征值变化时向对端设备发送指示。

----结束

SSAP Client:

- 步骤 1 调用 `enable_sle`，打开 SLE 开关。
- 步骤 2 调用 `ssapc_register_callbacks`，注册 SSAP client 回调。
- 步骤 3 调用 `sle_announce_seek_register_callbacks`，注册设备公开和设备发现回调函数。
- 步骤 4 调用 `ssapc_register_client`，创建一个 client 实体。
- 步骤 5 递归调用 `ssapc_find_structure` 查找对端属性数据库。
- 步骤 6 如果关注对端某个特征，可调用 `ssapc_write_req` 或 `ssapc_write_cmd` 将该特征的客户端特征配置描述符写为 `0x0001` 或 `0x0002`，前者可使能对端特征通知，后者可使能对端特征指示。
- 步骤 7 调用读写接口操作对端属性数据库。

----结束

3.3.4.6 Sample 示例

3.3.4.6.1 sle_uuid sample 工程构建指导

该 sample 用于验证星闪的 client 能力，包括打开星闪、client 注册、扫描、连接、数传和关闭星闪功能。其中设置的数据 length 和 phy 默认为 WS73U 参数，若使用 WS73E，请根据要求进行调整（WS73E 的 data size 最大 1450，data length 最大 1530，MTU 最大 1500，phy 支持 1M/2M/4M）。

```

27 #define SLE_DATA_SIZE_DEFAULT      200 // 73E可使用1450
28 #define SLE_SEEK_INTERVAL_DEFAULT  160 // 160
29 #define SLE_SEEK_WINDOW_DEFAULT    40 // 40
30 #define UUID_16BIT_LEN              2
31 #define UUID_128BIT_LEN             16
32 #define SPEED_DEFAULT_CONN_INTERVAL 0x10
33 #define SPEED_DEFAULT_TIMEOUT_MULTPLIER 0x1f4
34 #define DATA_LEN                    251 // 设置datalen, 73e用1530, 73u用251
35 #define DEFAULT_SLE_SPEED_MCS        10
36 #define DEFAULT_SLE_SPEED_MTU_SIZE   1500
37
38 #define MS_100                        100000
39 #define THOUSAND                       1000
40 #define UUID_14_BYTE                   14
41 #define UUID_15_BYTE                   15

```

```

121 void sle_sample_pair_complete_cbk(uint16_t conn_id, const sle_addr_t *addr, errcode_t status)
122 {
123     printf("sle_sample_pair_complete_cbk status: %d\n", status);
124     sle_set_mcs(conn_id, DEFAULT_SLE_SPEED_MCS);
125
126     sle_set_phy_t phy_parm = {
127         .tx_format = SLE_RADIO_FRAME_2,
128         .rx_format = SLE_RADIO_FRAME_2,
129         .tx_phy = SLE_PHY_2M, // 73E支持SLE_PHY_4M
130         .rx_phy = SLE_PHY_2M, // 73E支持SLE_PHY_4M
131         .tx_pilot_density = SLE_PHY_PILOT_DENSITY_16_TO_1,
132         .rx_pilot_density = SLE_PHY_PILOT_DENSITY_16_TO_1,
133         .g_feedback = 0,
134         .t_feedback = 0,
135     };
136     errcode_t ret2 = sle_set_phy_param(conn_id, &phy_parm);
137     printf("[ssap client] set phy, ret: %d\n", ret2);
138 }

```

步骤 1 更改 Makefile 中的 CROSS 宏为主控使用的工具链。

```

1 # CROSS 为交叉编译工具链
2 CROSS = arm-himix100-linux-
3 CC = @echo " GCC  $@"; $(CROSS)gcc
4 LD = @echo " LD   $@"; $(CROSS)ld
5 AR = @echo " AR    $@"; $(CROSS)ar
6 RM = @echo " RM    $@"; rm -f
7 STRIP = @echo "STRIP $@"; $(CROSS)strip

```


- 步骤 2 在 sample 的 lib 文件夹中放置 BTH 协议栈静态库文件（若 WS73 SDK 中 "application/lib" 文件夹下有对应主控文件夹，可使用对应主控中的协议栈；否则需要提供主控对应的交叉编译工具链给原厂，原厂编译协议栈后提供给客户）。
- 步骤 3 在 WS73 SDK 的 "application/sample/sle/sle_uuid/sle_uuid_client" 路径下执行 "make clean;make" 即可编译出 sle_client_sample 应用（不要更改 sample 的路径，Makefile 中定义了 sample 需要的头文件的相对路径）；sle_server_sample 同理，存放在 "application/sample/sle/sle_uuid/sle_uuid_server" 路径下，在此路径下执行 "make clean;make"。
- 步骤 4 在主控上加载 plat_soc.ko 和 sle_soc.ko，并确认正常运行后，在 client 端运行 sle_client_sample，在 server 端运行 sle_server_sample。两块主控会自动连接并开始数传。

----结束

4 BLE&SLE 配置说明

表4-1 BLE&SLE 说明

配置项	配置说明	样例	默认值
bt_coex_mode=0	配置蓝牙共存模式配置： 0：片外共存； 1：片内共存。	bt_coex_mode=0 表示配置为片外共存。 说明：没有片外 WiFi 的场景，或者片外 WiFi 与 WS73 之间无 PTA 仲裁的场景，需配置为片内共存模式： bt_coex_mode=1	0
ble_disable_ll_privacy=0	配置 ble 是否关闭可解析特性： 0：开启可解析特性； 1：关闭可解析特性。	ble_disable_ll_privacy=1 表示关闭可解析特性。 遥控器场景关闭可解析特性。	0
bsle_suspend_mode=0	配置待机唤醒模式： 0：ble 待机扫描； 1：sle 待机扫描。	bsle_suspend_mode=0 表示使用 ble 待机扫描，扫描并匹配 ble 唤醒广播报文。	0
bsle_suspend_scan_interval=300	配置待机唤醒的扫描 interval： 取值范围 3-10240	bsle_suspend_scan_interval=300 表示待机唤醒的扫描 interval 为 300ms	300
bsle_suspend_scan_window=30	配置待机唤醒的扫描	bsle_suspend_scan_window=30	30

配置项	配置说明	样例	默认值
	window： 取值范围 3-10240	表示待机唤醒的扫描 window 为 30ms	
bsle_use_flash=0	配置 bsle 的产线校准值和 mac 地址来自 flash 或 efuse： 0：来自 efuse； 1：来自 flash。	bsle_use_flash=1 表示 bsle 的产线校准值和 mac 地址来自 flash。 说明：若有任意 flash 项被判定无效，则拒绝所有的 flash 值，而采用 efuse 优先。	0
bt_maxpower=7	最大功率档位：范围 0-7 7：BLE 最大 8 个功率档位。	bt_maxpower=7，表示 BLE 的功率档位数。 <ul style="list-style-type: none">若设置为 7，可以使用的档位是 0~7，每档对应 -6, -2, 2, 6, 10, 14, 16, 20；若设置为 5，可以使用的档位是 0~5，每档对应-6, -2, 2, 6, 10, 14。	7

表4-2 无委认证配置说明

配置项	配置说明	样例	默认值
bt_srrc_switch=1	配置蓝牙无委认证开关： 0：关闭无委认证降功率； 1：打开无委认证降功率。	bt_coex_mode=1 打开无委认证降功率。	1
bt_srrc_pa_ref_val1	配置无委认证降功率值，单位 dB，范围 0~14dB	bt_srrc_pa_ref_val1 = 10 表示每个功率档位降 10dB	10

配置项	配置说明	样例	默认值
bt_srrc_pa_ref_val2	配置无委认证降功率值, 单位 dB, 范围 0~14dB	bt_srrc_pa_ref_val2 = 6 表示每个功率档位降 6dB	6
bt_srrc_pa_ref_val3	配置无委认证降功率值, 单位 dB, 范围 0~14dB	bt_srrc_pa_ref_val3 = 6 表示每个功率档位降 6dB	6
bt_srrc_pa_ref_val4	配置无委认证降功率值, 单位 dB, 范围 0~14dB	bt_srrc_pa_ref_val4 = 6 表示每个功率档位降 6dB	6
bt_srrc_pa_ref_val5	配置无委认证降功率值, 单位 dB, 范围 0~14dB	bt_srrc_pa_ref_val5 = 14 表示每个功率档位降 14dB	14
bt_srrc_pa_ref_val6	配置无委认证降功率值, 单位 dB, 范围 0~14dB	bt_srrc_pa_ref_val6 = 255 填写非法值, 表示该字段不被使用	255
bt_srrc_pa_ref_val7	配置无委认证降功率值, 单位 dB, 范围 0~14dB	bt_srrc_pa_ref_val7 = 255 填写非法值, 表示该字段不被使用	255
bt_srrc_pa_ref_val8	配置无委认证降功率值, 单位 dB, 范围 0~14dB	bt_srrc_pa_ref_val8 = 255 填写非法值, 表示该字段不被使用	255
bt_srrc_pa_fre1	配置无委认证降功率的频点, 范围 0~78, 与降功率值一一对应	bt_srrc_pa_fre1=0 表示 RF 第 0 信道降功率	0
bt_srrc_pa_fre2	配置无委认证降功率的频点, 范围 0-78, 与降功率值一一对应	bt_srrc_pa_fre2=75 表示 RF 第 75 信道降功率	75
bt_srrc_pa_fre3	配置无委认证降功率的频点, 范围 0~78, 与降功率值一一对应	bt_srrc_pa_fre3=76 表示 RF 第 76 信道降功率	76
bt_srrc_pa_fre4	配置无委认证降功率的频点, 范围 0~78, 与	bt_srrc_pa_fre4=77 表示 RF 第 77 信道降功率	77

配置项	配置说明	样例	默认值
	降功率值——对应		
bt_srrc_pa_fre5	配置无委认证降功率的频点，范围 0~78，与降功率值——对应	bt_srrc_pa_fre5=78 表示 RF 第 78 信道降功率	78
bt_srrc_pa_fre6	配置无委认证降功率的频点，范围 0~78，与降功率值——对应	bt_srrc_pa_fre6=255 填写非法值，表示该字段不被使用	255
bt_srrc_pa_fre7	配置无委认证降功率的频点，范围 0~78，与降功率值——对应	bt_srrc_pa_fre7=255 填写非法值，表示该字段不被使用	255
bt_srrc_pa_fre8	配置无委认证降功率的频点，范围 0~78，与降功率值——对应	bt_srrc_pa_fre8=255 填写非法值，表示该字段不被使用	255

表4-3 蓝牙发射功率校准配置项说明

配置项	配置说明	样例	默认值
bt_cali_txpwr_pa_fre1	配置 BAND1 发射功率校准频率值（与 bt_cali_txpwr_pa_ref_band1 对应）。	bt_cali_txpwr_ppa_fre1=6，表示 BAND1 频率点为（2402+6）MHz	6
bt_cali_txpwr_pa_fre2	配置 BAND2 发射功率校准频率值（与 bt_cali_txpwr_pa_ref_band2 对应）。	bt_cali_txpwr_ppa_fre2=15，表示 BAND2 频率点为（2402+15）MHz	15
bt_cali_txpwr_pa_fre3	配置 BAND3 发射功率校准频率值（与 bt_cali_txpwr_pa_ref_band3 对应）。	bt_cali_txpwr_ppa_fre3=26，表示 BAND3 频率点为（2402+26）MHz	26
bt_cali_txpwr_pa_fre4	配置 BAND4 发射功率校准频率值（与 bt_cali_txpwr_pa_ref_band4 对应）。	bt_cali_txpwr_ppa_fre4=39，表示 BAND4 频率点为（2402+39）MHz	39

配置项	配置说明	样例	默认值
	d4 对应)。		
bt_cali_txpwr_pa_fre5	配置 BAND5 发射功率校准频率值 (与 bt_cali_txpwr_pa_ref_band5 对应)。	bt_cali_txpwr_ppa_fre5=60, 表示 BAND5 频率点为 (2402+60) MHz	60
bt_cali_txpwr_pa_fre6	配置 BAND6 发射功率校准频率值 (与 bt_cali_txpwr_pa_ref_band6 对应)。	bt_cali_txpwr_ppa_fre6=68, 表示 BAND6 频率点为 (2402+68) MHz	68
bt_cali_txpwr_pa_fre7	配置 BAND7 发射功率校准频率值 (与 bt_cali_txpwr_pa_ref_band7 对应)。	bt_cali_txpwr_ppa_fre7=72, 表示 BAND7 频率点为 (2402+72) MHz	72
bt_cali_txpwr_pa_fre8	配置 BAND8 发射功率校准频率值 (与 bt_cali_txpwr_pa_ref_band8 对应)。	bt_cali_txpwr_ppa_fre8=76, 表示 BAND8 频率点为 (2402+76) MHz	76
bt_cali_txpwr_pa_ref_band1	配置 BAND1 发射功率, 单位 dBm, 范围 18~22。	20, 代表设置发射功率为 20dBm	20
bt_cali_txpwr_pa_ref_band2	配置 BAND2 发射功率, 单位 dBm, 范围 18~22。	20, 代表设置发射功率为 20dBm	20
bt_cali_txpwr_pa_ref_band3	配置 BAND3 发射功率, 单位 dBm, 范围 18~22。	20, 代表设置发射功率为 20dBm	20
bt_cali_txpwr_pa_ref_band4	配置 BAND4 发射功率, 单位 dBm, 范围 18~22。	20, 代表设置发射功率为 20dBm	20
bt_cali_txpwr_pa_ref_band5	配置 BAND5 发射功率, 单位 dBm, 范围 18~22。	20, 代表设置发射功率为 20dBm	20
bt_cali_txpwr_pa_ref_band6	配置 BAND6 发射功率, 单位 dBm, 范围 18~22。	20, 代表设置发射功率为 20dBm	20
bt_cali_txpwr_pa_ref_band7	配置 BAND7 发射功率, 单位 dBm, 范围 18~22。	20, 代表设置发射功率为 20dBm	20

配置项	配置说明	样例	默认值
bt_cali_txpwr_pa_ref_band8	配置 BAND8 发射功率, 单位 dBm, 范围 18~22。	20, 代表设置发射功率为 20dBm	20
bt_cali_txpwr_pa_fre_block1	配置频点应用功率范围。	bt_cali_txpwr_pa_fre_block1=10, 表示将 BAND1 功率校准值应用至 (2402+0)MHz~(2402+10)MHz	10
bt_cali_txpwr_pa_fre_block2	配置频点应用功率范围。	bt_cali_txpwr_pa_fre_block2=20, 表示将 BAND2 功率校准值应用至 (2402+10)MHz~(2402+20)MHz	20
bt_cali_txpwr_pa_fre_block3	配置频点应用功率范围。	bt_cali_txpwr_pa_fre_block3=32, 表示将 BAND3 功率校准值应用至 (2402+20)MHz~(2402+32)MHz	32
bt_cali_txpwr_pa_fre_block4	配置频点应用功率范围。	bt_cali_txpwr_pa_fre_block4=50, 表示将 BAND4 功率校准值应用至 (2402+32)MHz~(2402+50)MHz	50
bt_cali_txpwr_pa_fre_block5	配置频点应用功率范围。	bt_cali_txpwr_pa_fre_block5=64, 表示将 BAND5 功率校准值应用至 (2402+50)MHz~(2402+64)MHz	64
bt_cali_txpwr_pa_fre_block6	配置频点应用功率范围。	bt_cali_txpwr_pa_fre_block6=70, 表示将 BAND6 功率校准值应用至 (2402+64)MHz~(2402+70)MHz	70

配置项	配置说明	样例	默认值
bt_cali_txpwr_pa_fre_block7	配置频点应用功率范围。	bt_cali_txpwr_pa_fre_block7=74, 表示将 BAND7 功率校准值应用至 (2402+70)MHz~(2402+74)MHz, 也表示将 BAND8 功率校准值应用至 (2402+74)MHz~(2402+78)MHz	74

5 常见问题

暂无。