Tina Linux 蓝牙
开发指南

版本号: 2.5
发布日期: 2024.10.09

版本历史

版本号	日期	制/修订人	内容描述
1.0	2019.03.03	AWA1423	初始版本。
1.1	2022.05.12	AWA1423	完善整个开发文档。
1.2	2022.03.15	AWA1427	全文更新 API 到 btmanager4.0 版本。
2.0	2022.06.05	AWA1887	全文删除 API 接口具体介绍。
2.1	2023.04.20	AWA1887	增加 BLE5 拓展功能接口。
2.2	2023.10.25	AWA1887	修复文档里错误的内容描述与格式。
2.3	2023.11.27	AWA1887	增加文档适用 buildroot 构建方式。
2.4	2024.08.27	KPA0568	修复文档里错误的内容描述与格式。
2.5	2024.10.09	AWA1887	增加部分功能接口和整理文档目录结构。

目　录

1 前言 **1**
1.1 文档简介 . . . 1
1.2 目标读者 . . . 1
1.3 适用范围 . . . 1
1.4 文档约定 . . . 1
1.4.1 标志说明 . . . 1
1.4.2 地址与数据描述方法约定 . . . 2
1.5 相关术语介绍 . . . 2

2 模块介绍 **3**
2.1 Bluetooth 简介 . . . 3
2.1.1 Controller 介绍 . . . 4
2.1.1.1 BR/EDR Controller . . . 5
2.1.1.2 LE Controller . . . 5
2.1.2 HOST 介绍 . . . 13
2.1.2.1 经典蓝牙 Profile 介绍 . . . 14
2.1.2.2 低功耗蓝牙 Profile 介绍 . . . 19
2.2 软件框架 . . . 26
2.3 源码结构 . . . 28
2.3.1 HCI 驱动 . . . 28
2.3.2 内核蓝牙协议栈 . . . 28
2.3.3 BlueZ 协议栈 . . . 29
2.3.4 Btmanager 中间件 . . . 30

3 模块配置 **32**
3.1 配置说明 . . . 32
3.2 Device Tree 配置 . . . 33
3.3 menuconfig 配置 . . . 34
3.3.1 openwrt 构建方式 . . . 34
3.3.1.1 XR 系列配置 . . . 34
3.3.1.2 RTL87xx 系列配置 . . . 35
3.3.1.3 AIC 系列配置 . . . 36
3.3.2 buildroot 构建方式 . . . 37

4 模块接口说明 **39**
4.1 通用 API . . . 39
4.2 BlueZ Agent API . . . 40
4.3 A2DP Sink API . . . 41
4.4 AVRCP-sink API . . . 41

4.5 A2DP Source API . . . 41
4.6 AVRCP-source API . . . 42
4.7 HFP HF API . . . 42
4.8 HFP AG API . . . 42
4.9 SPP Client API . . . 43
4.10 SPP Server API . . . 43
4.11 GATT Server API . . . 43
4.12 GATT Client API . . . 44

5 功能开发 47
5.1 配置文件介绍 . . . 47
5.1.1 bt_init.sh . . . 47
5.1.2 bluetooth.json . . . 49
5.2 开发流程 . . . 51
5.2.1 A2DP Sink 功能调用 API 实现 . . . 51
5.2.2 A2DP Source 功能调用 API 实现 . . . 51
5.2.3 HFP HF 功能调用 API 实现 . . . 52
5.2.4 HFP AG 功能调用 API 实现 . . . 53
5.2.5 SPP Client 功能调用 API 实现 . . . 53
5.2.6 SPP Server 功能调用 API 实现 . . . 54
5.2.7 GATT Server 功能调用 API 实现 . . . 54
5.2.8 GATT Client 功能调用 API 实现 . . . 55

6 bt_test 工具介绍 57
6.1 bt_test 简介 . . . 57
6.2 使用说明 . . . 57
6.2.1 后台模式 . . . 58
6.2.2 交互模式 . . . 58
6.2.2.1 通用命令说明 . . . 59
6.2.2.2 经典蓝牙命令说明 . . . 59
6.2.2.3 BLE-GATT-Server 命令说明 . . . 61
6.2.2.4 BLE-GATT-Client 命令说明 . . . 61
6.3 功能验证 . . . 62
6.3.1 A2DP Sink 测试 . . . 62
6.3.2 AVRCP CT 测试 . . . 63
6.3.3 A2DP Souce 测试 . . . 63
6.3.4 AVRCP TG 测试 . . . 64
6.3.5 SPP Client 测试 . . . 64
6.3.6 SPP Server 测试 . . . 65
6.3.7 HFP HF 测试 . . . 66
6.3.8 HFP AG 测试 . . . 66
6.3.9 GATT Server 测试 . . . 67
6.3.10 GATT Client 测试 . . . 68

7 常见问题和调试指南 69

插　图

图 2-1　协议结构图 4
图 2-2　BR/EDR 链路状态 5
图 2-3　LE 链路状态 6
图 2-4　Format of static address 7
图 2-5　Format of non-resolvable private address 8
图 2-6　Format of resolvable private address 8
图 2-7　Mapping of PHY channel to physical channel index and channel type 10
图 2-8　Advertising physical channel PDU 11
图 2-9　Advertising and Scan Response data format 12
图 2-10　Permitted usages for data types 13
图 2-11　A2DP 传输结构 15
图 2-12　A2DP 传输例子 15
图 2-13　AVRCP 框架 16
图 2-14　AVRCP 示例 16
图 2-15　HFP 框架 18
图 2-16　SPP 示例 19
图 2-17　Attribute 20
图 2-18　GATT Profile attribute types 21
图 2-19　GATT 通信模型 22
图 2-20　GATT Server 模型 24
图 2-21　weight service 26
图 2-22　Tina 蓝牙协议栈结构图 27
图 3-1　主控与 BT 硬件连接简图 32

1 前言

1.1 文档简介

本文档介绍 Tina Linux 平台上 Bluetooth 开发，主要包括蓝牙协议栈介绍、蓝牙 API 接口介绍以及 demo 的使用方法。

1.2 目标读者

- 技术支持工程师
- 软件开发工程师

1.3 适用范围

Allwinner 软件平台 Tina v3.5 版本及以上，btmanager 版本在 4.0 版本以上。Allwinner 硬件平台 R 系列、V 系列、T 系列、F 系列、MR 系列等。

1.4 文档约定

1.4.1 标志说明

⚠ 注意

- **提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。**

说明

为准确理解文中指令、正确实施操作而提供的补充或强调信息。

技巧

一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

1.4.2 地址与数据描述方法约定

本文档在描述地址、数据时遵循如下约定：

表 1-1: 地址与数据描述方法约定

符号	例子	说明
0x	0x0200，0x79	地址或数据以 16 进制表示。
0b	0b010，0b00 000 111	数据采用二进制表示 (寄存器描述除外)。
X	00X，XX1	数据描述中，X 代表 0 或 1。 例如，00X 代表 000 或 001；XX1 代表 001，011，101 或 111。

1.5 相关术语介绍

术语	解释说明
BR/EDR	代表经典蓝牙。
LE	代表低功耗蓝牙。
Controller	蓝牙协议栈的底层部分，一般运行在无线模组端。
Host	蓝牙协议栈的底层部分，一般运行在主控端。
Bluez	Tina 平台所使用的一种开源的蓝牙协议栈。
bt_manager	AW 开发的蓝牙中间件，向下对接协议栈，向上提供 API 接口。
bt_test	AW 基于 btmanager API 接口，编写的实际测试 demo。

2 模块介绍

2.1 Bluetooth 简介

蓝牙技术发展至今已经迭代多个版本，截至目前 SIG Bluetooth 规范已经到 V5.2。蓝牙主要分为两种不同的技术: 经典蓝牙（Classic Bluetooth，简称 BT）和蓝牙低功耗（Bluetooth Low Energy，简称 BLE）。

蓝牙的工作频率范围是 2400MHz~2483.5MHz，在经典蓝牙中，将其分为 79 个频道（每个频道 1MHz），而在蓝牙低功耗中，分为 40 个频道（每个频道 2MHz）。

蓝牙协议从结构上可以分为控制器 (Controller) 和主机 (Host) 两大部分，如下图：

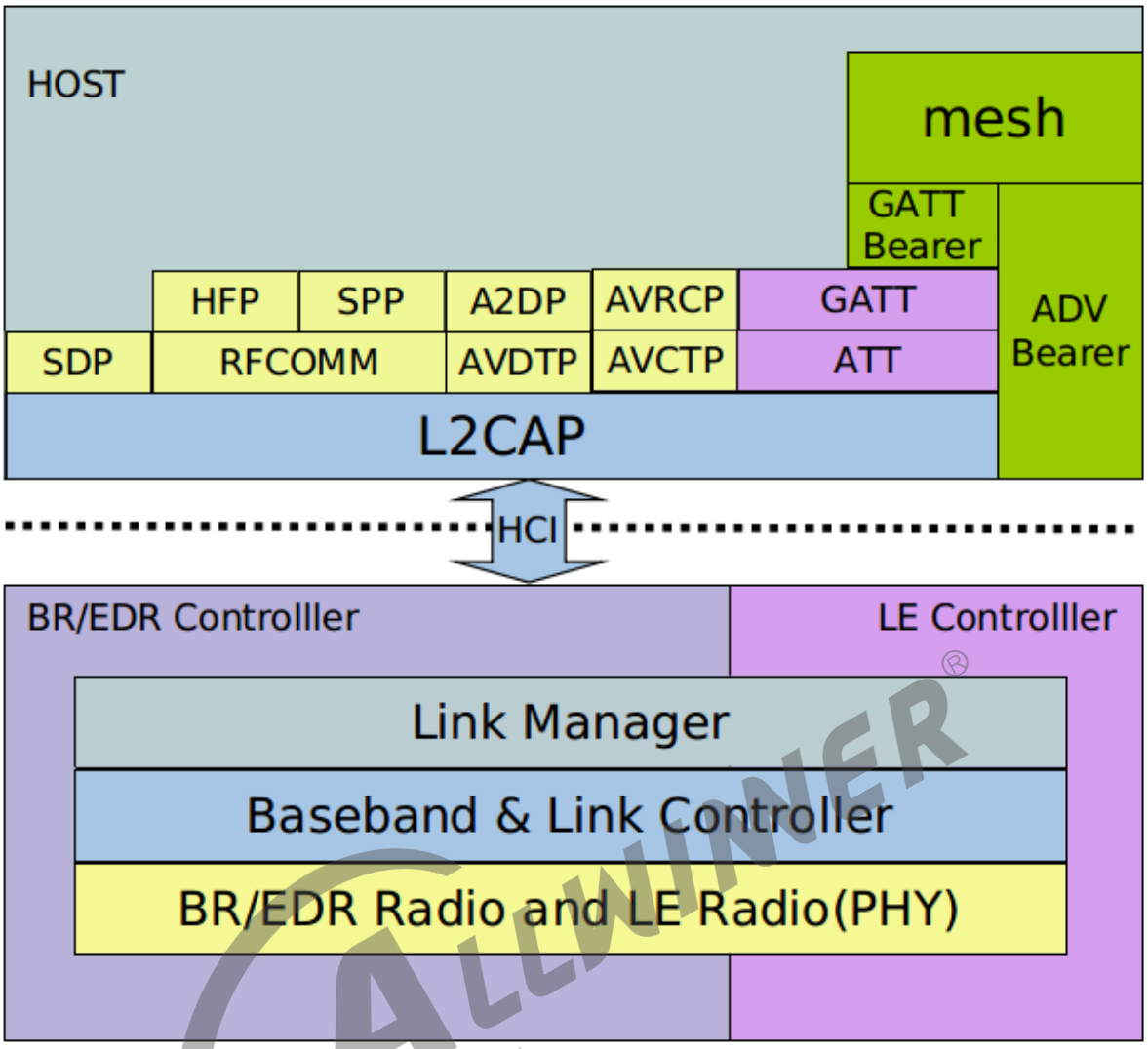

图 2-1: 协议结构图

Controller 和 Host 大部分情况下是运行在两个不同的芯片上，比如 Controller 运行在 XR829，而 Host 运行在 R328，两个芯片通过硬件通信接口（如 UART、USB、SDIO）进行连接和通信，双方的通信协议称为 HCI 协议。

2.1.1 Controller 介绍

Controller 分为 BR/EDR Controller 和 LE controller，两者的在 PHY 层的信道划分是不一样的，两部分可以认为是独立的。

2.1.1.1 BR/EDR Controller

BR/EDR 采用跳频技术，数据传输时，并不是固定的占用 79 个信道中的某一个，而是一定规律的跳动，这个跟 Wi-Fi 的固定信道传输不同。链路层包括下图几个状态：

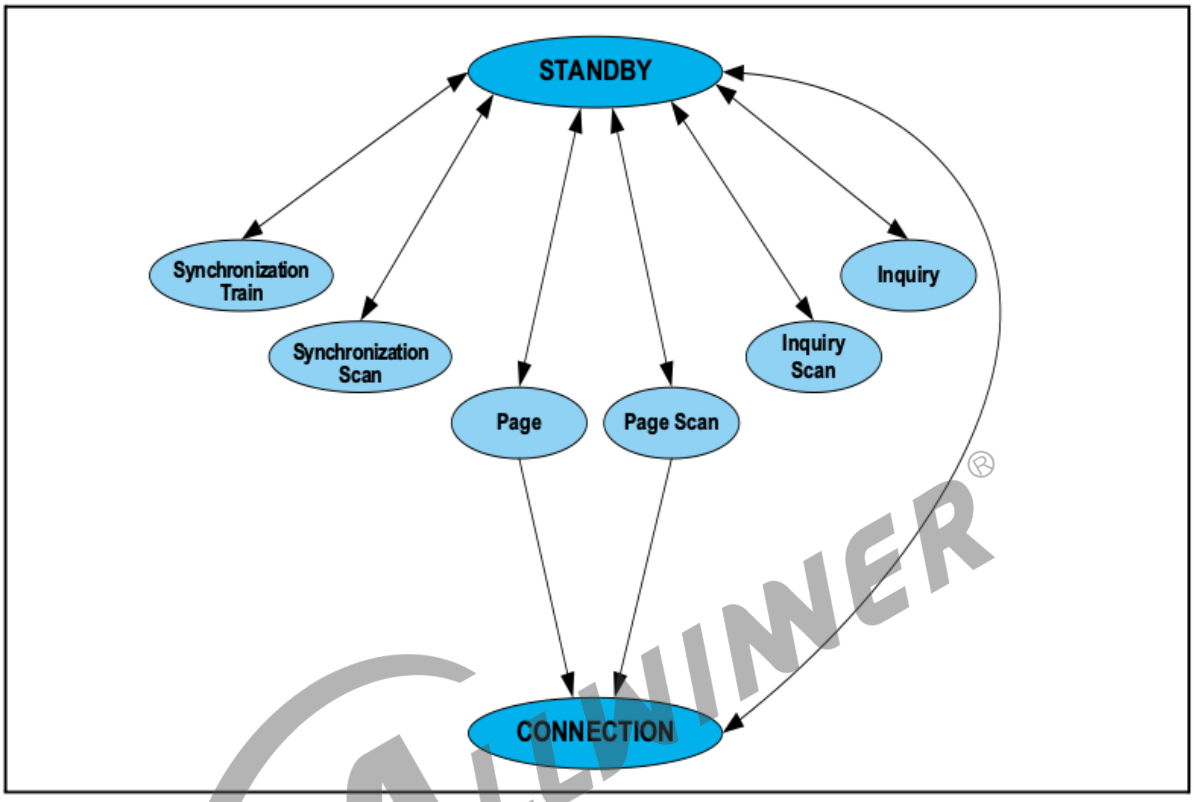

图 2-2: BR/EDR 链路状态

其中 synchronization train、synchronization scan 基本不用。

- STANDBY：一个设备的默认状态, 可以认为是初始的状态。
- CONNECTION：处于连接的状态, 可以进行数据的交互。它代表正常工作的状态。
- Page：这个子状态代表主动去连接、激活对应的 Slave 的状态。
- Page Scan：这个子状态是和 Page 对应的，它就是 Slave 等待被 Page 的状态。
- Inquiry：扫描状态，这个状态的设备就是去扫描周围的设备。
- Inquiry Scan：可被周围设备发现的状态。

2.1.1.2 LE Controller

LE 在 40 个信道中又可以分为两种信道：连接信道和广播信道。

连接信道用于处于连接状态的蓝牙设备直接通信，与 BR/EDR 一样，都是采用跳频，只不过是在 37 个信道上跳频。

广播信道是用于设备之间进行无连接的广播通信，这些广播通信可以用于蓝牙设备的发现、连接等操作。

LE 状态可以分为五种，如下图。

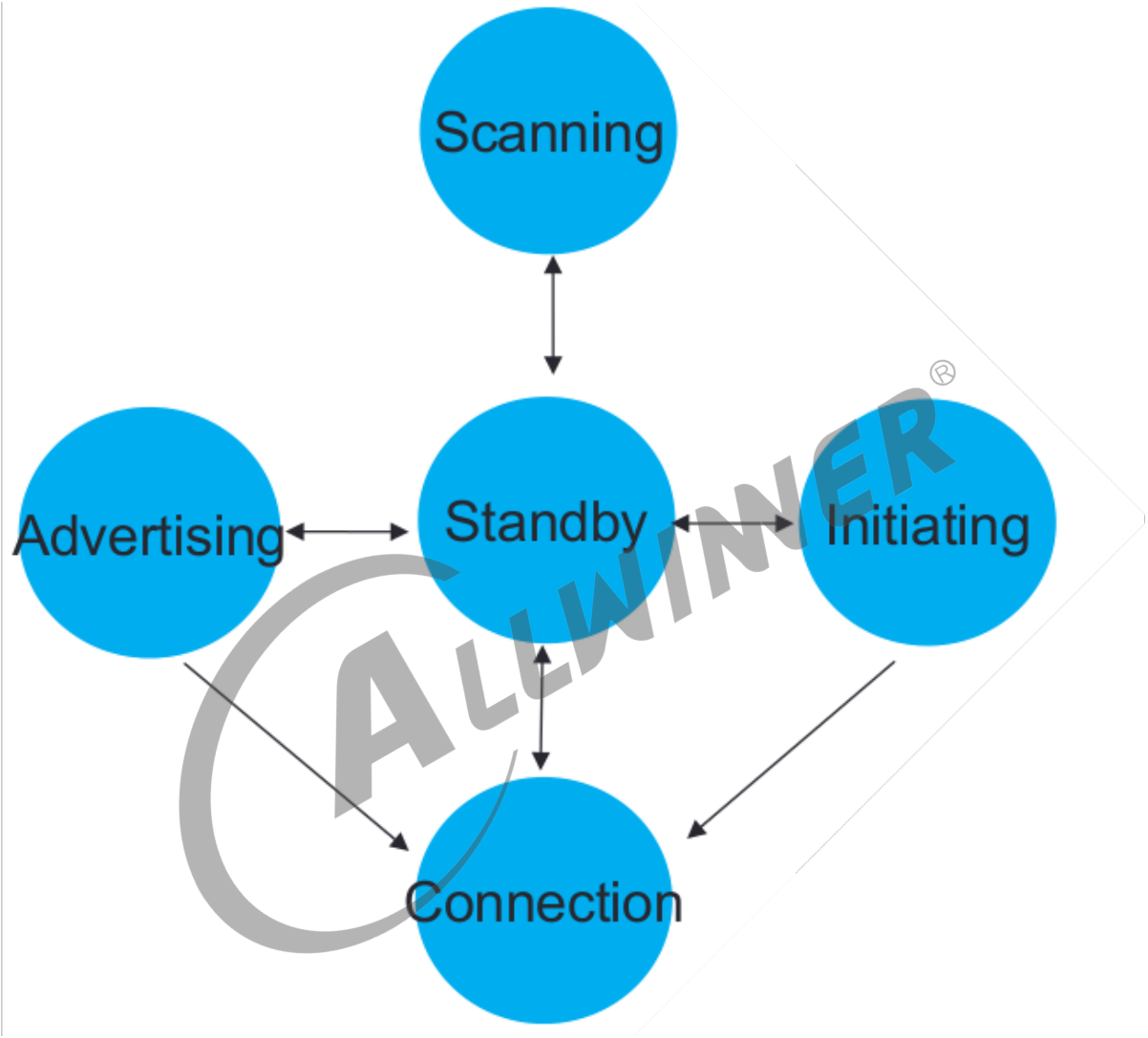

图 2-3: LE 链路状态

- Standby：链路层不收发报文。
- Advertising：链路层发送广播信道报文，并可能监听以及响应有这些广播信道报文触发的回应报文。
- Scanning：链路层监听广播者发送的广播信道报文。
- Initiating：链路层监听并响应从特定设备发起的广播信道报文。
- Connection：分为主从设备，有发起态进入连接态的设备为主设备，由广播态进入连接态的为从设备。

2.1.1.2.1 LE Device address

LE Device address 可以分为两种类型：Public device address 和 Random device address。

（1）Public device address

在通信系统中，设备地址是用来唯一识别一个物理设备的，如 TCP/IP 网络中的 MAC 地址、传统蓝牙中的蓝牙地址等。对设备地址而言，一个重要的特性，就是唯一性（或者说一定范围内的唯一），否则很有可能造成很多问题。蓝牙通信系统也不例外。对经典蓝牙（BR/EDR）来说，其设备地址是一个 48bits 的数字，称作 “48-bit universal LAN MAC addresses(和电脑的 MAC 地址一样)“。正常情况下，该地址需要向 IEEE 申请（其实是购买）。当然，这种地址分配方式，在 BLE 中也保留下来了，就是 Public Device Address。Public Device Address 由 24-bit 的 company_id 和 24-bit 的 company_assigned 组成，具体可参考蓝牙 Spec 中相关的说明（Core_v5.2.pdf：[Vol 2] Part B,Section 1.2）。

（2）Random device address

Random device address 又分为 Static Device Address 和 Private Device Address 两类。在 BLE 时代，只有 Public Device Address 还不够，主要 3 个原因：首先 Public Device Address 需要向 IEEE 购买。虽然不贵，但在 BLE 时代，相比 BLE IC 的成本，还是不小的一笔开销；其次：Public Device Address 的申请与管理是相当繁琐、复杂的一件事情，再加上 BLE 设备的数量众多（和传统蓝牙设备不是一个数量级的），导致维护成本增大；最后，安全因素。BLE 很大一部分的应用场景是广播通信，这意味着只要知道设备的地址，就可以获取所有的信息，这是不安全的。因此固定的设备地址，加大了信息泄漏的风险。为了解决上述问题，BLE 协议新增了一种地址：Random Device Address，即设备地址不是固定分配的，而是在设备设备启动后随机生成的。根据不同的目的，Random Device Address 分为 Static Device Address 和 Private Device Address 两类。

（2-1）Static Device Address

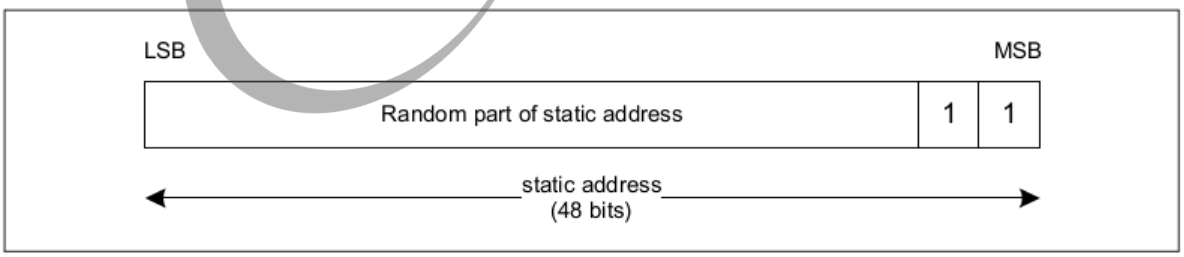

图 2-4: Format of static address

Static Device Address 是设备在上电时随机生成的地址，格式如上。46bits 的随机数，可以很好地解决 “设备地址唯一性” 的问题，因为两个地址相同的概率很小。地址随机生成，可以解决 Public Device Address 申请所带来的费用和维护问题。

特征可以总结为：

- 最高两个 bit 为 “11”。

- 剩余的 46bits 是一个随机数，不能全部为 0，也不能全部为 1。
- 在一个上电周期内保持不变。
- 下一次上电的时候可以改变，但不是强制的，因此也可以保持不变。如果改变，上次保存的连接等信息，将不再有效。

（2-2）Private Device Address

Static Device Address 通过地址随机生成的方式，解决了部分问题，Private Device Address 则更进一步，通过定时更新和地址加密两种方法，提高蓝牙地址的可靠性和安全性。根据地址是否加密，Private Device Address 又分为两类，Non-resolvable private address 和 Resolvable private address。下面我们分别描述：

（2-2-1）Non-resolvable private address

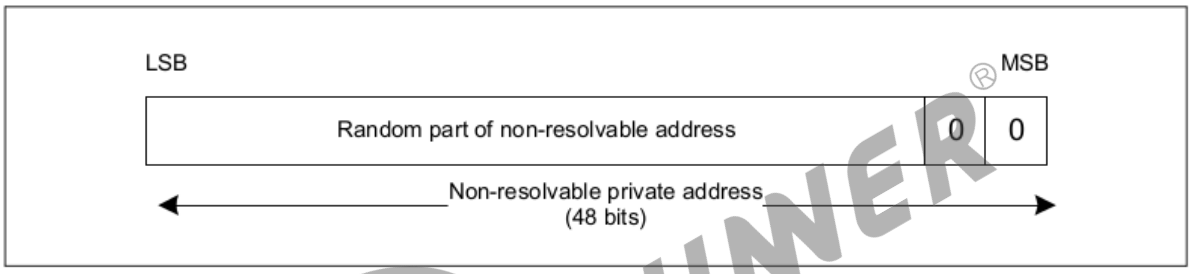

图 2-5: Format of non-resolvable private address

Non-resolvable private address 和 Static Device Address 类似。其格式如上，不同之处在于，Non-resolvable private address 会定时更新。更新的周期称是由 GAP 规定的，称作 T_GAP(private_addr_int)，建议值是 15 分钟。特征可以总结为：

- 最高两个 bit 为 “00”。
- 剩余的 46bits 是一个随机数，不能全部为 0，也不能全部为 1。
- 以 T_GAP(private_addr_int) 为周期，定时更新。

（2-2-2）Resolvable private address

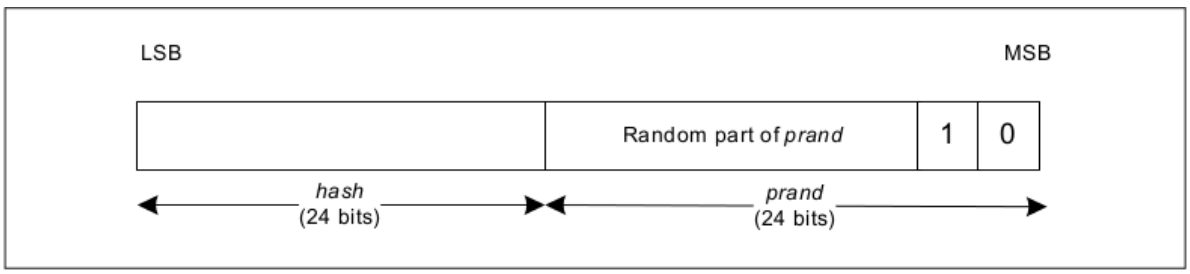

图 2-6: Format of resolvable private address

Resolvable private address 比较有用，格式如上，它通过一个随机数和一个称作 identity resolving key (IRK) 的密码生成，因此只能被拥有相同 IRK 的设备扫描到，可以防止被未知设备扫描和追踪。

- 由两部分组成：高位 24bits 是随机数部分，其中最高两个 bit 为 “10”，用于标识地址类型；低位 24bits 是随机数和 IRK 经过 hash 运算得到的 hash 值，运算的公式为 hash = ah(IRK, prand)。
- 当对端 BLE 设备扫描到该类型的蓝牙地址后，会使用保存在本机的 IRK，和该地址中的 prand，进行同样的 hash 运算，并将运算结果和地址中的 hash 字段比较，相同的时候，才进行后续的操作。这个过程称作 resolve（解析），这也是 Non-resolvable private address/Resolvable private address 命名的由来。
- 以 T_GAP(private_addr_int) 为周期，定时更新。哪怕在广播、扫描、已连接等过程中，也可能改变。
- Resolvable private address 不能单独使用，因此需要使用该类型的地址的话，设备要同时具备 Public Device Address 或者 Static Device Address 中的一种。

2.1.1.2.2 Physical channel

BLE 的信道划分为 0~39，其中 channel 37、38、39 为广播信道，其它为数据信道。

PHY Channel	RF Center Frequency	Channel Index	Physical Channel Type	
			Primary Advertising	All others
0	2402 MHz	37	●	
1	2404 MHz	0		●
2	2406 MHz	1		●
…	…	…	…	…
11	2424 MHz	10		●
12	2426 MHz	38	●	
13	2428 MHz	11		●
14	2430 MHz	12		●
…	…	…	…	…
38	2478 MHz	36		●
39	2480 MHz	39	●	

图 2-7: Mapping of PHY channel to physical channel index and channel type

2.1.1.2.3 LE 广播通信

LE 链路有 5 个状态，其中 Advertising 和 Scanning 是 LE 非常重要的两个状态，它对蓝牙在未建立连接之前至关重要。(下面描述针对 BLE4 传统广播，BLE5 拓展广播不作描述。)

广播通信的数据格式如下：

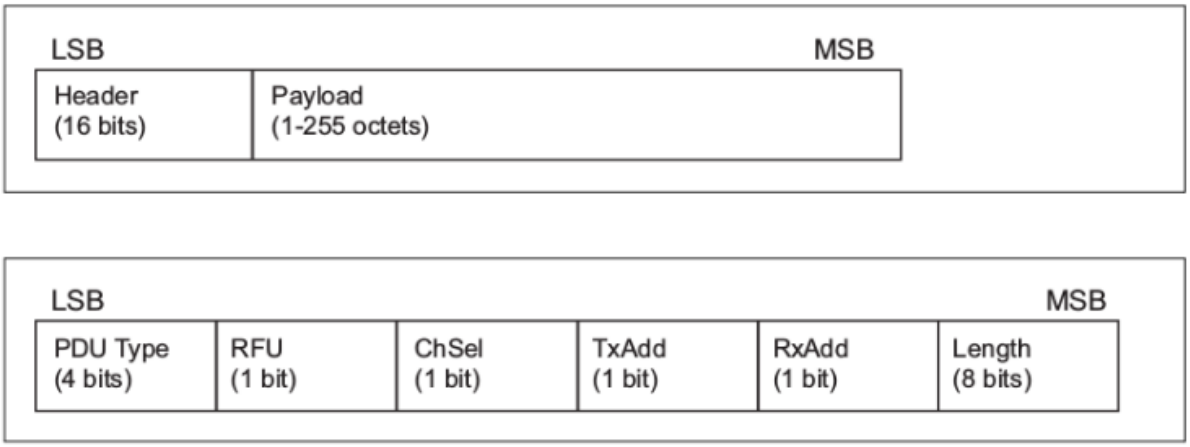

图 2-8: Advertising physical channel PDU

根据 PDU 中的 Type 字段，分为以下几个类型：

PDU 类型	PDU 格式	说明
ADV_IND	Adva(6 octets) AdvData(0~31 octets)	可被连接，可被扫描。
ADV_DRECT_IND	Adva(6 octets) ArgetA(6 octets)	可被指定的设备连接，不可被扫描。
ADV_NONCONN_IND	Adva(6 octets) AdvData(0~31 octets)	不可被连接，不可被扫描。
ADV_SCAN_IND	Adva(6 octets) AdvData(0~31 octets)	不可被连接，可被扫描。
SCAN_REQ	ScanA(6 octets) Adva(6 octets)	当接收到 ADV_IND 或 ADV_SCAN_IND 类型的广播包时，可以通过该 PDU，请求广播者广播更多的信息。
SCAN_RSP	Adva(6 octets) ScanData(0~31 octets)	广播者收到 SCAN_REQ 请求后，通过该 PDU 响应，把更多的数据传送给接收者。

上表中，重点关注 AdvA、AdvData、ScanRspData。

- AdvA 字段是广播者的地址，可以是 public address 也可以是 random address。
- AdvData 字段是广播者广播的数据内容，长度为 31 字节。
- ScanRspData 字段是广播者收到 SCAN_REQ 之后回复的广播数据内容。

2.1.1.2.4 AdvData 和 ScanRspData 格式

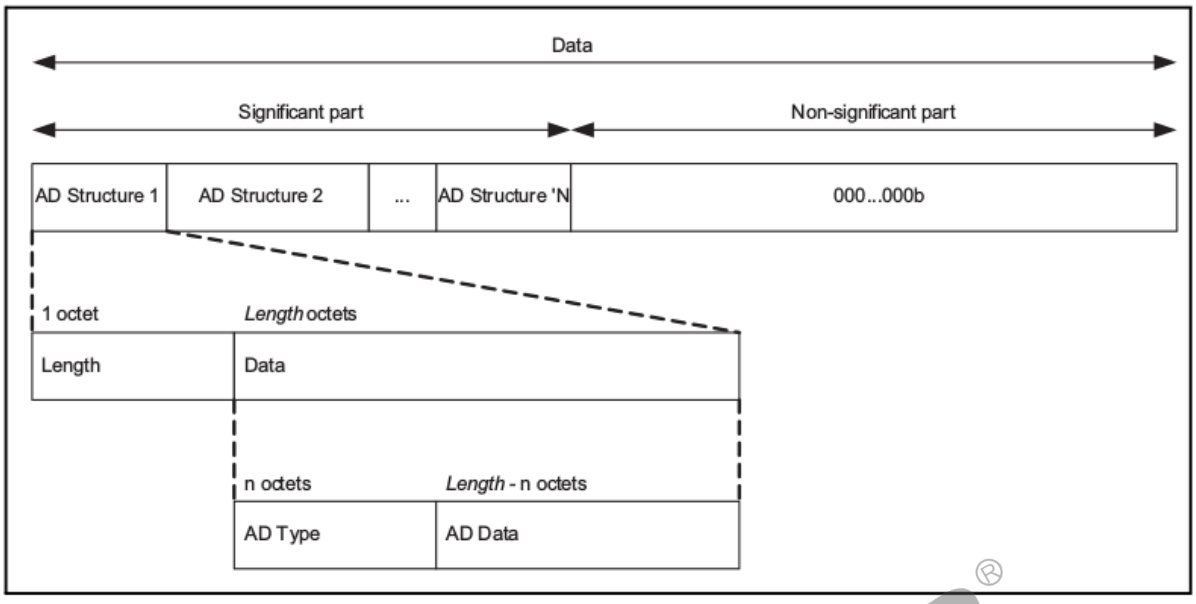

图 2-9: Advertising and Scan Response data format

如上图所示，AdvData 和 ScanRspData 格式内容由多个 AD Structure 组成，每个 AD structure 又细分为 length、AD Type 和 AD data。AD Type 和 AD data 有详细内容请参见 https://www.Bluetooth.com/specifications/assigned-numbers/generic-access-profile/ 或《蓝牙核心规范》。

	Context				
Data type	**EIR**	**AD**	**SRD**	**ACAD**	**OOB**
Service UUID	O	O	O	O	O
Local Name	C1	C1	C1	X	C1
Flags	C1	C1	X	X	C1
Manufacturer Specific Data	O	O	O	O	O
TX Power Level	O	O	O	X	O
Secure Simple Pairing OOB	X	X	X	X	O
Security Manager OOB	X	X	X	X	O
Security Manager TK Value	X	X	X	X	O
Slave Connection Interval Range	X	O	O	X	O
Service Solicitation	X	O	O	X	O
Service Data	X	O	O	O	O
Appearance	X	C2	C2	X	C1
Public Target Address	X	C2	C2	X	C1
Random Target Address	X	C2	C2	X	C1
Advertising Interval	X	C1	C1	X	C1
LE Bluetooth Device Address	X	X	X	X	C1
LE Role	X	X	X	X	C1
Uniform Resource Identifier	O	O	O	X	O
LE Supported Features	X	C1	C1	X	C1
Channel Map Update Indication	X	X	X	C1	X
BIGInfo	X	X	X	C1	X
Broadcast_Code	X	X	X	X	O

O: Optional in this context (may appear more than once in a block).

C1: Optional in this context; shall not appear more than once in a block.

C2: Optional in this context; shall not appear more than once in a block and shall not appear in both the AD and SRD of the same extended advertising interval.

图 2-10: Permitted usages for data types

2.1.2 HOST 介绍

一般情况下蓝牙协议栈的 controller 运行在无线模组上，而 HOST 运行在主控芯片上，所以从用户的角度我们着重关注 HOST 端。在我们日常生活中，会碰到非常多的使用场景，比如蓝牙播放音乐，蓝牙鼠标，蓝牙传输文件，蓝牙语音通话，蓝牙 mesh 灯，通过蓝牙定位等等。根据这些不同的场景需求，SIG 定义了不同的规范（Profile）来支持这些场景下的需求。

根据不同的场景需求定义了不同用户规范（Profile），而 HOST 与 Controller 直接的传输是只有

一个接口线，同时对于 controller 只需要关心数据的收发，不需要关心用户的实际场景，所以在 HOST 端有了 L2CAP 规范，这样就能屏蔽上层不同用户的协议，达到协议复用的功能，类似 TCP/IP 协议中的传输层。

L2CAP 之上有很多 profile，profile 之间有些是相辅相成的，有些则是完全独立的。根据这些 profile，我们将其分为 3 大类（参考图 2-1 协议结构图）。

- 经典蓝牙部分（黄色部分）
- 蓝牙低功耗部分（紫色部分）
- mesh 部分（绿色部分）

下面我们对常用的 profile 进行简单的介绍，分为经典蓝牙和低功耗蓝牙两部分。

2.1.2.1 经典蓝牙 Profile 介绍

2.1.2.1.1 GAP

通用访问规范（Generic Access Profile, GAP）是一个基础的蓝牙 profile，用于提供蓝牙设备的通用访问功能，包括设备的发现、连接、鉴权、服务发现等等。

GAP 是所有其它应用模型的基础，它定义了在 Bluetooth 设备间建立基带链路的通用方法。还定义了一些通用的操作，这些操作可供引用 GAP 的应用模型以及实施多个应用模型的设备使用。GAP 确保了两个蓝牙设备（不管制造商和应用程序）可以通过 Bluetooth 技术交换信息，以发现彼此支持的应用程序。

2.1.2.1.2 A2DP

为了利用蓝牙异步无连接链路传输高质量的音频数据，蓝牙 SIG 发布了高级音频分发规范（Advanced Audio Distribution Profile, A2DP）。A2DP 典型的应用是音乐播放器将音频数据发送耳机或者音箱。当前 A2DP 仅仅定义了点对点的音频分发，没有定义广播式的音频分发。

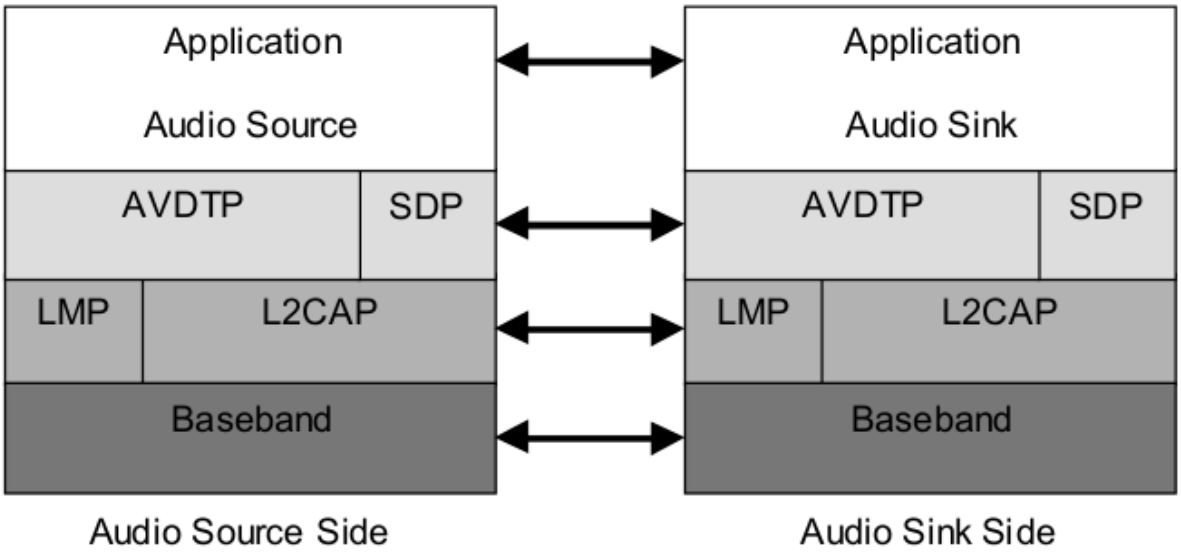

图 2-11: A2DP 传输结构

A2DP 可以分为 A2DP Source 和 A2DP Sink，发送音频数据那一端我们称为 Source 端（比如手机），接收音频的那一端我们称为 Sink 端（比如蓝牙音箱）。A2DP 是建立在 AVDTP 之上的，AVDTP 实现实体通过 L2CAP 分组进行 audio 数据流的传输和 audio 信令的交换，信令提供数据刘的发现、配置、建立和传输控制等功能，可以理解为 AVDTP 是 A2DP 更基础的协议。

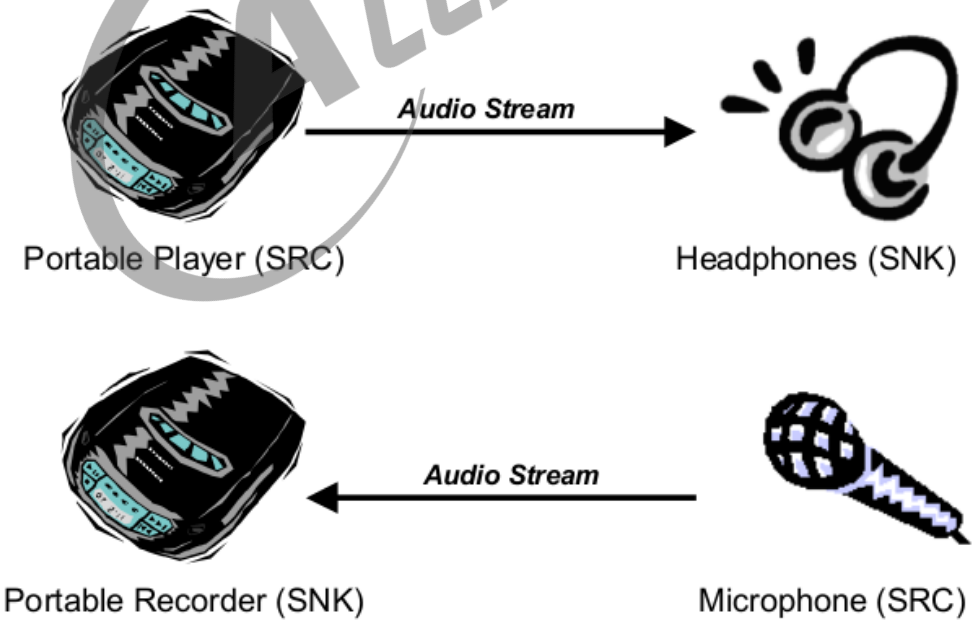

图 2-12: A2DP 传输例子

2.1.2.1.3 AVRCP

音视频远端控制协议（Audio/Video Remote Control Profile, AVRCP）是蓝牙音频实现蓝牙无线遥控功能的规范。

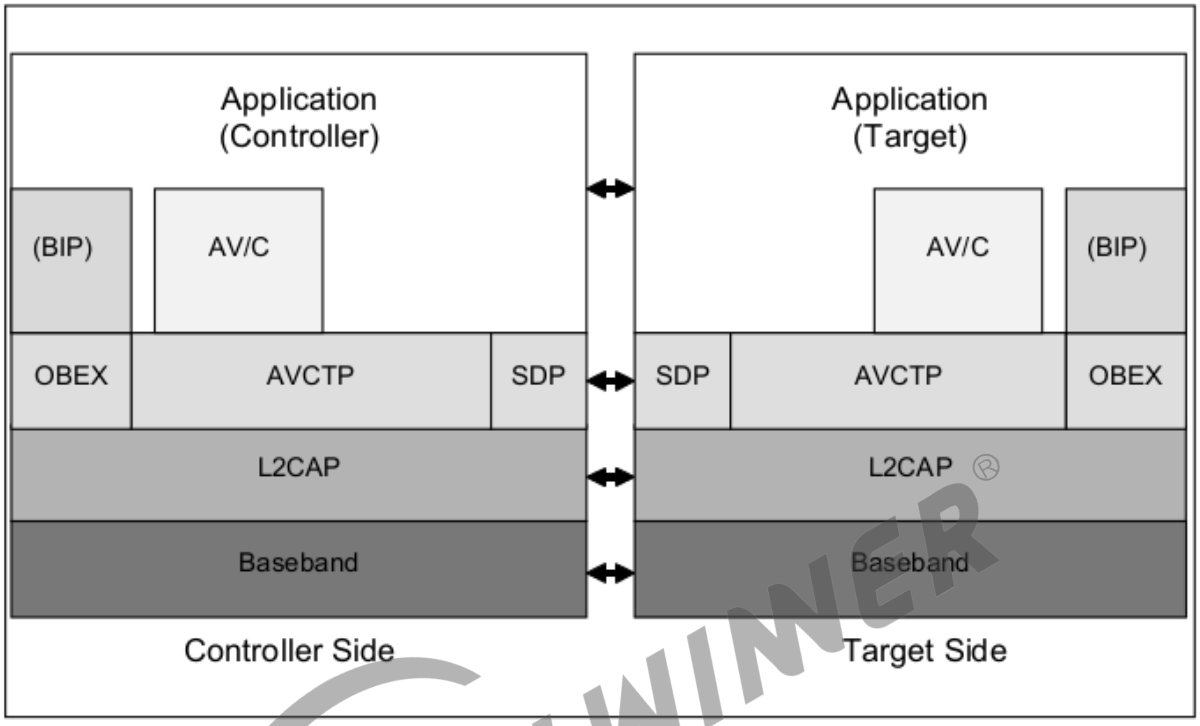

图 2-13: AVRCP 框架

AVRCP 中定义了两种设备角色：Controller（控制器，CT）、Target（目标机，TG）。CT 是发起命令传输给到 TG 的宿主，比如个人电脑、PDA、手机等。TG 是接收命令的宿主，比如蓝牙耳机、TV 等。

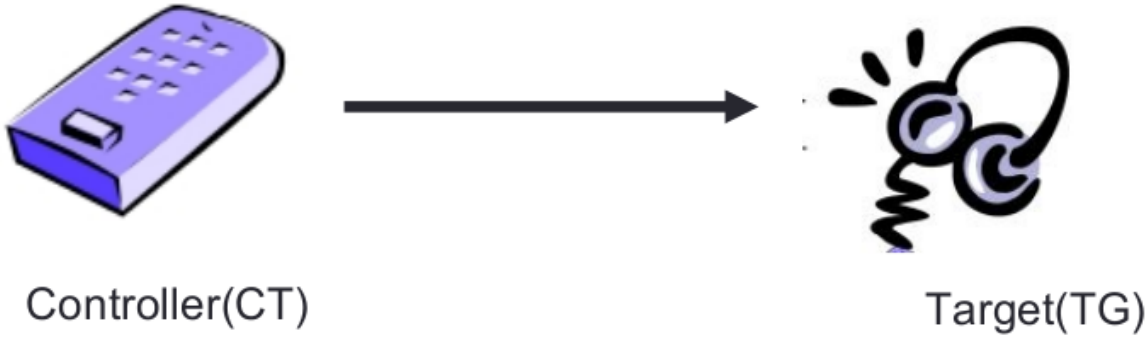

图 2-14: AVRCP 示例

AVRCP 中分为四种指令：

- Unit info：用来获取 AV/C 设备的整体信息。
- Subunit info：用来获取 AV/C 设备的子设备信息。
- Vendor Dependent：厂商自定义的 AV/C 指令。
- Pass Through：音频设备使用最多的命令，如播放、暂停、快进、快退、下一曲、上一曲。

2.1.2.1.4 HFP

HFP 全称 Hands-free Profile，可以用做蓝牙语音通话。蓝牙语音通话实际上我们只需要重点关注两个方面：一个是语音通话的音频数据通路（over pcm、over sco）；另一个是语音通话的指令，称其为 AT 指令（比如电话的接听、挂断、拨号、获取手机信息等等）。

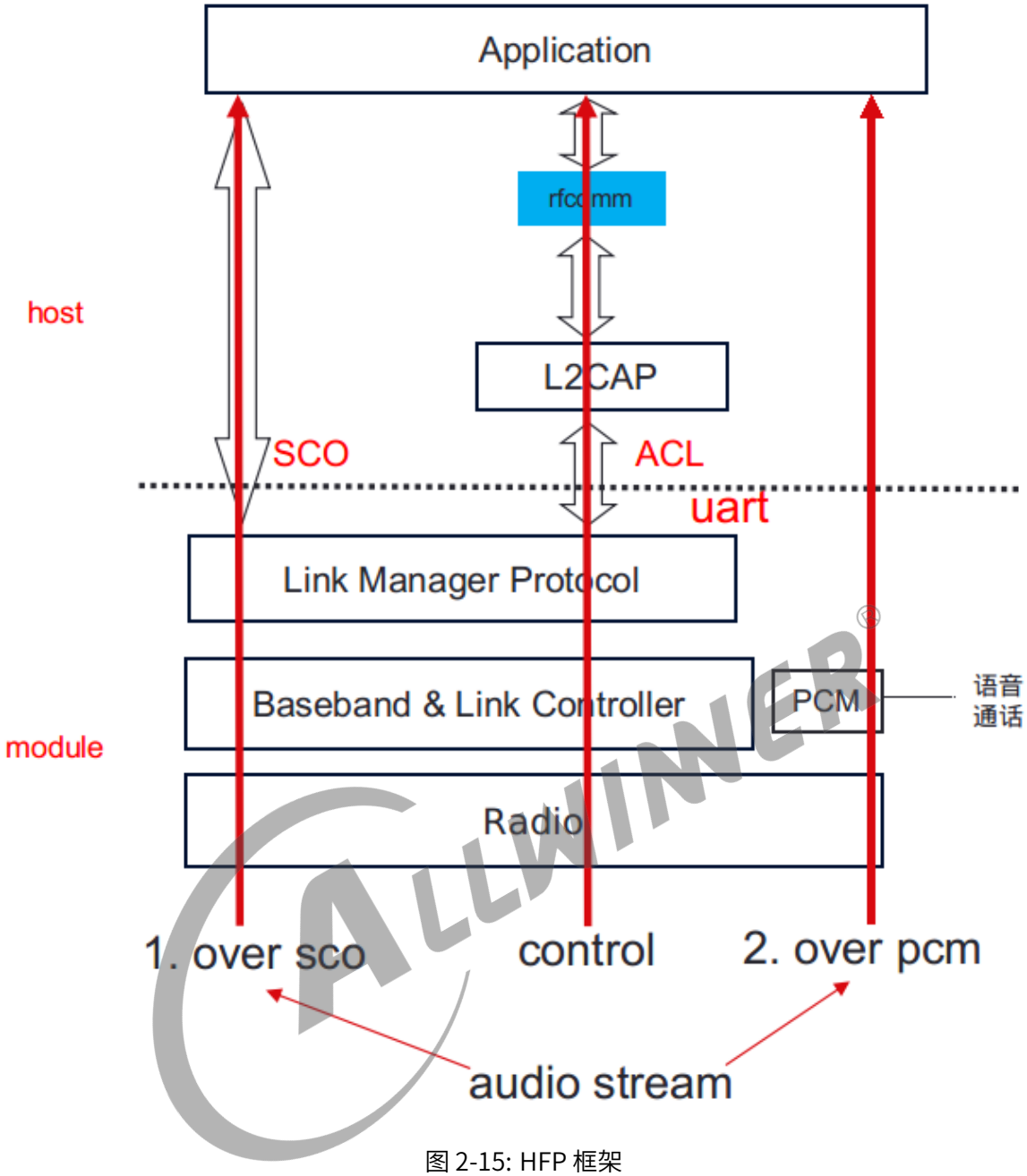

图 2-15: HFP 框架

（1）蓝牙语音通话音频数据

如上图所示，蓝牙语音通话的音频可以通过 HCI（UART），也可以直接通过 PCM（I2S）。

蓝牙语音通话数据走 HCI，数据流从模组端通过 UART 传输给主控，到 Host 端后通过蓝牙协议栈送到 bluealsa，再对接到 btmanager，类似 a2dp sink 通路。这样蓝牙语音通话将与其他 profile 同时占用 HCI，这样对多个 profile 同时存在时对 HCI 带宽有较高要求。

蓝牙语音通话走 PCM，在模组端有单独的 PCM 接口，可以通过 I2S 与主控直接进行连接，蓝牙语音通话数据就不需要再通过 HCI 占用带宽。当前大多数厂商此种方式。

2.1.2.1.5 SPP

蓝牙串口协议（Serial Port Profile, SPP）定义了使用蓝牙进行 RS232 （或类似）串行电缆仿真的设备应使用的协议和过程。简单来说就是在蓝牙设备之间建立虚拟的串口进行数据通信。SPP 底层基于 RFCOMM 协议，搭配 SDP 服务实现。

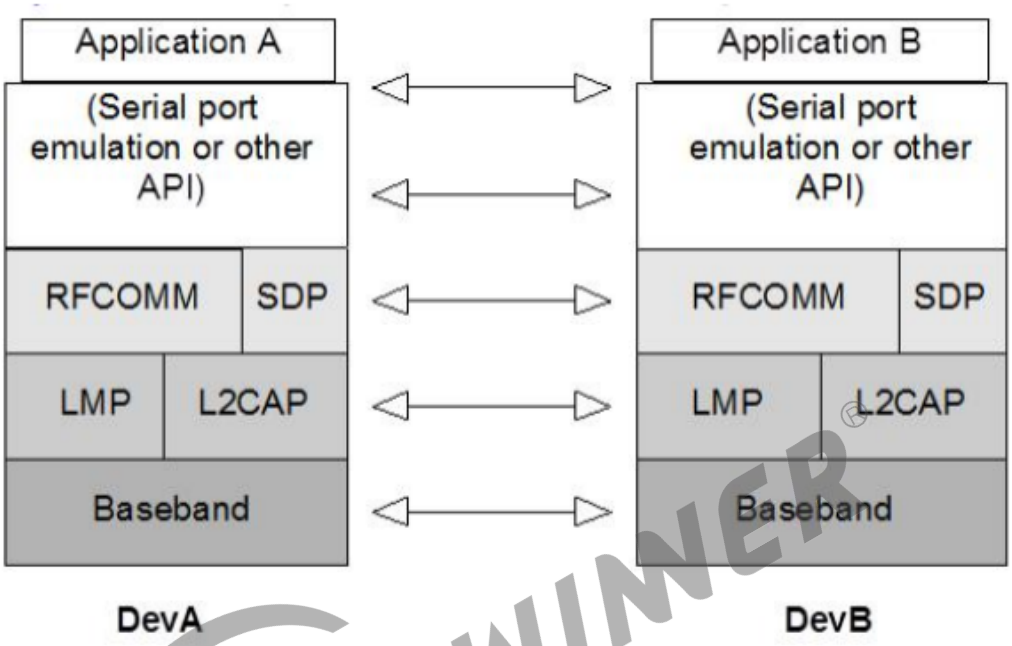

图 2-16: SPP 示例

2.1.2.2 低功耗蓝牙 Profile 介绍

蓝牙低功耗对应的 profile 是 GATT（Generic Attribute profile），从第 2 章节图 1 中我们知道 GATT 规范是基于 ATT 协议（Attribute Protocol）实现的。ATT 的通信模型遵循 C/S 模型，包括 Server 与 Client。

2.1.2.2.1 Attribute

一台设备如果作为 gatt server 端，在 server 端可以有很多服务，比如心率服务、血压服务、电量服务等等，而服务的基本组成单元是 Attribute（属性）。

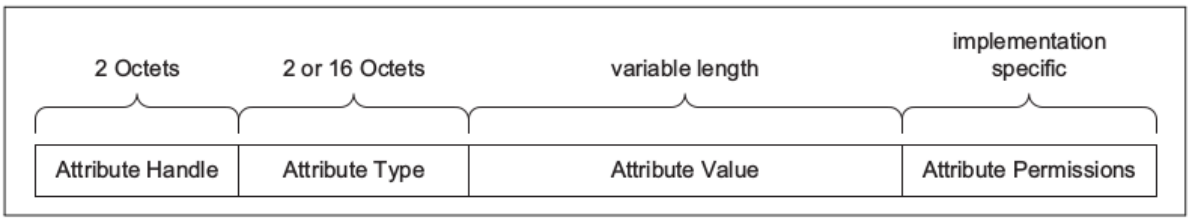

图 2-17: Attribute

属性（Attribute）是服务的基石，Attribute 的数据包类型如上图，包含了四种元素：

- Attribute Handle
- Attribute Type
- Attribute Value
- Attribute Permissions

（1）Attribute Type

Attribute Type 由 UUID 唯一标识，SIG 蓝牙联盟规定一些 UUID 代表特定的类型，比如 0x180D 代表 Heart Rate、0x1810 代表 Blood Pressure 等等（可参考：https://www.Bluetooth.com/specifications/gatt/services/）。

128 位的 UUID 相当长，设备间为了识别数据类型需要发送长达 16 字节的数据，为了提高效率，SIG 定义了 “蓝牙 UUID 基数” 的 128 位通用唯一标示码，结合一个较短的 16 位数使用，因此在实际传输的时候是 16 位的 uuid，在收发后补上蓝牙 UUID 基数即可。

蓝牙 UUID 基数如下：

```
00000000-0000-1000-8000-00805F9B34FB
```

需要发送的 16 位识别码为 0x2A01，完整的 128 位 UUID 便是：

```
00002A01-0000-1000-8000-00805F9B34FB
```

UUID 可以分为以下几组：

- 0x1800 ~ 0x26FF 用作服务类型通用唯一识别码
- 0x2700～0x27FF 用作标示计量单位
- 0x2800 ~ 0x28FF 用于区分属性类型
- 0x2900～0x29FF 用作特性描述
- 0x2A00～0X7FFF 用于区分特性类型

（2）Attribute Handle

设备中有许多服务，而服务有许多属性组成，比如温度传感器服务包含温度属性、设备名称属性、电池电量属性等等，这些属性似乎可以通过 Attribute Type 来作于区分，但是如果温度属性

有分为室内温度属性和室外温度属性，这样就没法通过 Attribute Type 来进行区分了，为了解决这个问题引入了 Attribute Handle，属性句柄。有效的属性句柄取值范围 0x0001~0xFFFF。

（3）Attribute Value

Attribute Value 是实际属性的值，比如温度传感器服务中温度属性温度是多少度。

（4）Attribute Permissions

Attribute 具有一组与之关联的权限值。权限值指定了关联属性是否具备读写、安全权限。一般有以下几种类型：

- Readable
- Writeable
- Readable and writable
- Encryption required
- No encryption required
- Authentication Required
- No Authentication Required

以上主要是关于属性四种元素的介绍，总结下 GATT profile 常见的属性定义，如下：

Attribute Type	UUID	Description
«Primary Service»	0x2800	Primary Service Declaration
«Secondary Service»	0x2801	Secondary Service Declaration
«Include»	0x2802	Include Declaration
«Characteristic»	0x2803	Characteristic Declaration
«Characteristic Extended Properties»	0x2900	Characteristic Extended Properties
«Characteristic User Description»	0x2901	Characteristic User Description Descriptor
«Client Characteristic Configuration»	0x2902	Client Characteristic Configuration Descriptor
«Server Characteristic Configuration»	0x2903	Server Characteristic Configuration Descriptor
«Characteristic Presentation Format»	0x2904	Characteristic Presentation Format Descriptor
«Characteristic Aggregate Format»	0x2905	Characteristic Aggregate Format Descriptor

图 2-18: GATT Profile attribute types

2.1.2.2.2 GATT

GATT 是基于 ATT 协议规范，所以 GATT 遵循 C/S 通信模型，包括 GATT server 和 GATT client。双方数据的传输方式分为以下 4 类：

- Client Request read
- Client Request write
- Server Notify
- Server Indication

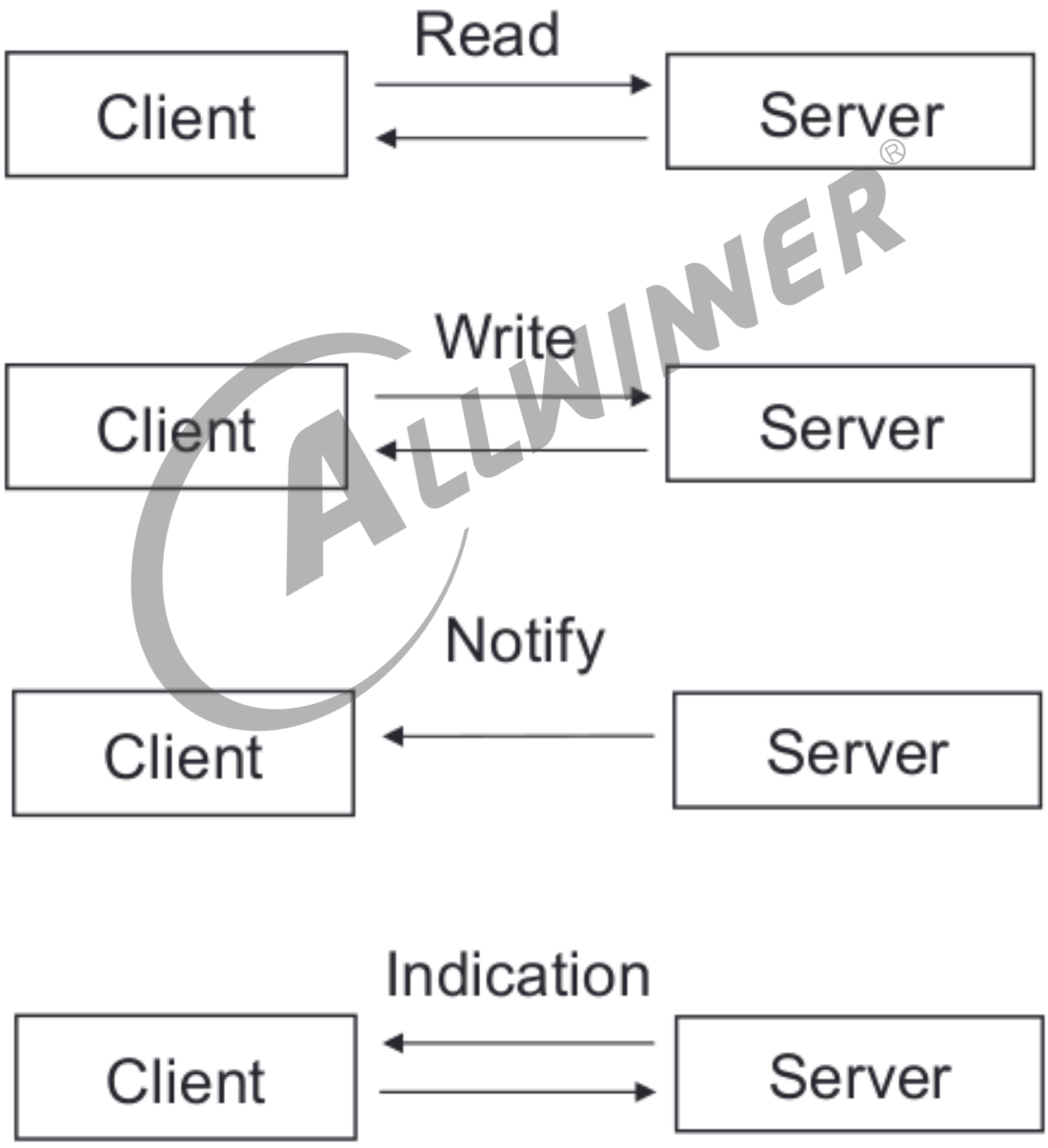

图 2-19: GATT 通信模型

其中 Server Notify 和 Server Indication 的区别：

- Notify：server 发送数据给 client 端不需要 client 回复；
- Indication ： server 发送数据给 client 端需要 client 回复。

2.1.2.2.3 GATT Server

前面说了，一个设备中可能有很多个服务，而服务一般具备一定的格式，服务的内容由属性（Attribute）构成。

一个设备的服务结构组成如下图：

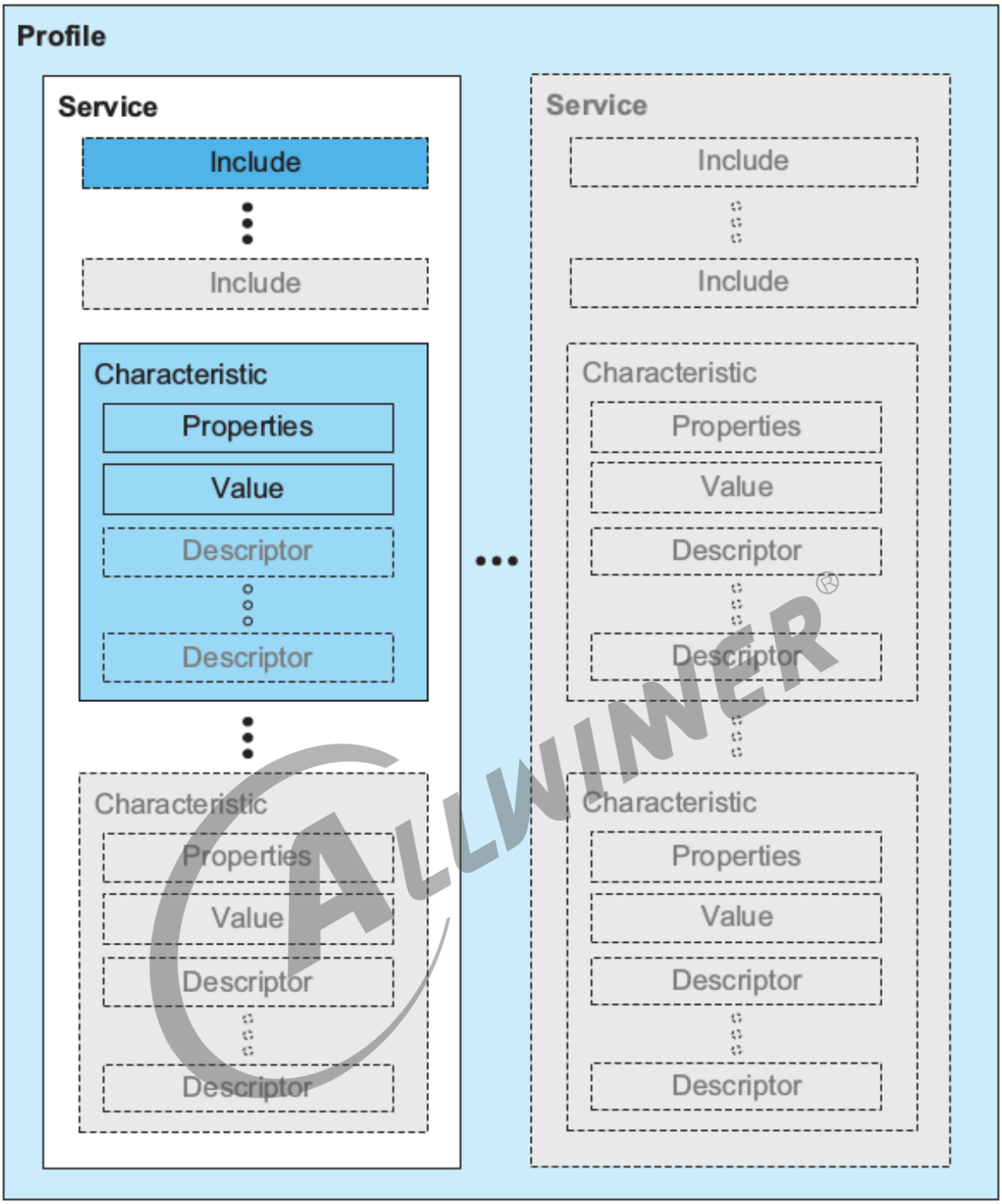

图 2-20: GATT Server 模型

GATT profile 的层次结构依次为 Profile->Service->Characteristic，“profile” 是基于 GATT 所派生的真正 profile, 位于 GATT profile hierarchy 最顶层, 有一个或者多个和某一应用的场景有关的 service 组成。

GATT server 是一系列数据和相关行为组成的集合，为了完成某个功能或特性。一个 service 包含一个或者多个 Characteristic, 也可以通过 include 的方式, 包含其他 service. 所有一个 server 的属性类型可以分为以下几类：

- Primary Service
- Secondary Service
- Include
- Characteristic

大部分情况下，我们只会用到 Primary Service 和 Characteristic。Primary service 是用于区分不同的 service，比如上图中有两个 service。一个 service 开头的 uuid 一般固定为 0x2800，其 value 值将用于表征这是那一类 service，同时将结束于下一个 0x2800。

Characteristic 则是 GATT profile 最基本的数据单位, 由一个 properties、一个 value、一个 Description 组成。

- Characteristic Properties 定义了 Characteristic 的 value 如何被使用, 以及 Characteristic 的 descriptor 如何被访问。
- Characteristic value 是特征的实际值, 例如一个温度特征, 就是温度值。
- Characteristic descriptor 则保存了一些和 Characteristic value 相关的信息。比如温度的单位是什么表征的。

注意：server 中的每一个定义，service、Characteristic、Characteristic Properties、Characteristic value、Characteristic descriptor 等等，都是通过 Attribute 来进行表征的。

下图是实际一个 service 的例子：

Handle	Attribute Type	Attribute Value
0x0001	«Primary Service»	«GAP Service»
0x0004	«Characteristic»	{0x02, 0x0006, «Device Name»}
0x0006	«Device Name»	“Example Device”
0x0010	«Primary Service»	«GATT Service»
0x0011	«Characteristic»	{0x26, 0x0012, «Service Changed»}
0x0012	«Service Changed»	0x0000, 0x0000
0x0100	«Primary Service»	«Battery State Service»
0x0106	«Characteristic»	{0x02, 0x0110, «Battery State»}
0x0110	«Battery State»	0x04
0x0200	«Primary Service»	«Thermometer Humidity Service»
0x0201	«Include»	{0x0500, 0x0504, «Manufacturer Service»}
0x0202	«Include»	{0x0550,0x0568}
0x0203	«Characteristic»	{0x02, 0x0204, «Temperature»}
0x0204	«Temperature»	0x028A
0x0205	«Characteristic Presentation Format»	{0x0E, 0xFE, «degrees Celsius», 0x01, «Outside»}
0x0206	«Characteristic User Description»	“Outside Temperature”
0x0210	«Characteristic»	{0x02, 0x0212, «Relative Humidity»}
0x0212	«Relative Humidity»	0x27
0x0213	«Characteristic Presentation Format»	{0x04, 0x00, «Percent», «Bluetooth SIG», «Outside»}
0x0214	«Characteristic User Description»	“Outside Relative Humidity”
0x0280	«Primary Service»	«Weight Service»
0x0281	«Include»	0x0505, 0x0509, «Manufacturer Service»}
0x0282	«Characteristic»	{0x02, 0x0283, «Weight kg»}

图 2-21: weight service

2.2 软件框架

Tina 系统当前使用的是开源的 BlueZ 协议栈，整个蓝牙协议栈的软件框架图如下：

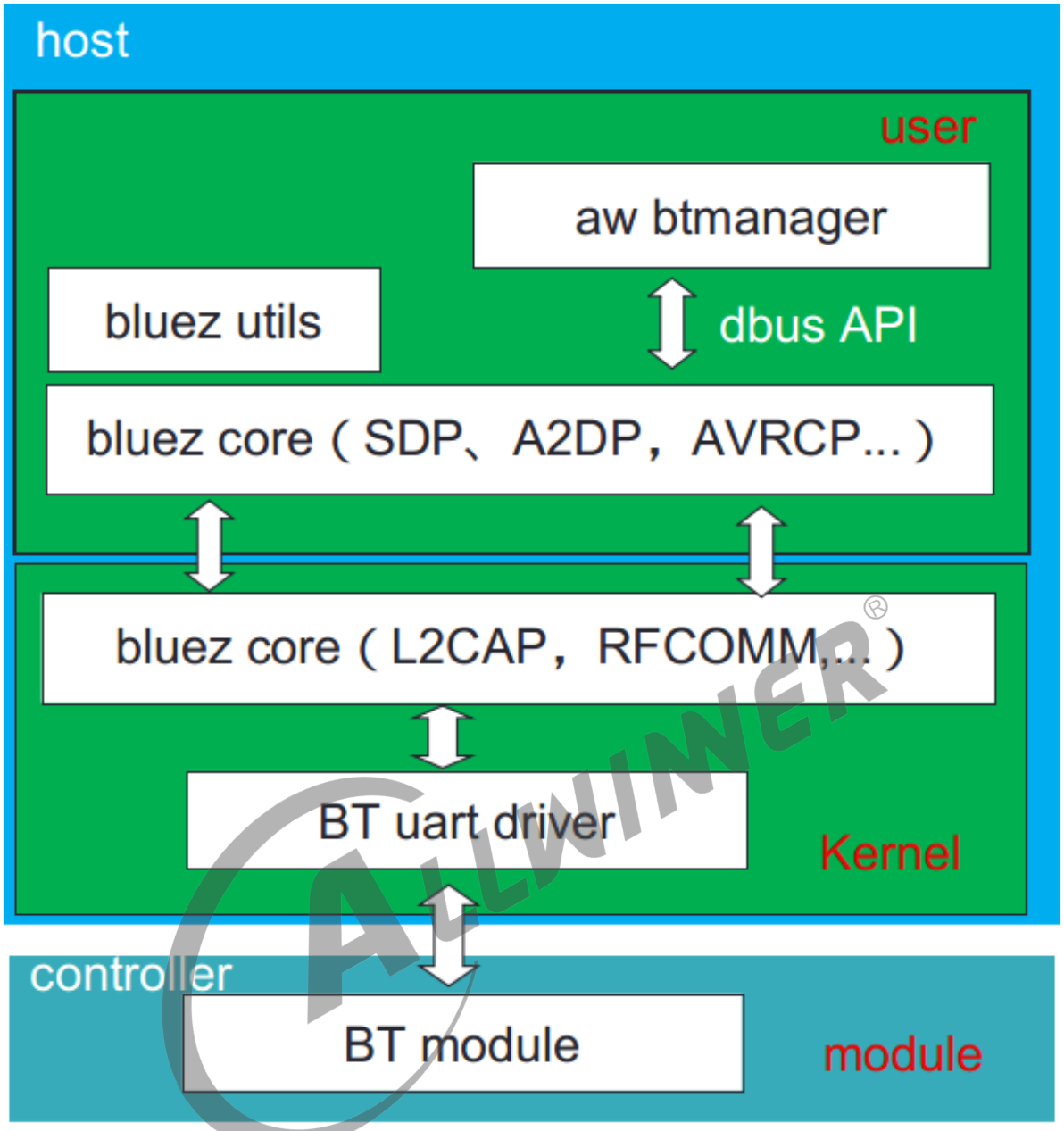

图 2-22: Tina 蓝牙协议栈结构图

如上图所示，蓝牙规范的 controller 部分是在模组端实现，host 部分是在主控端实现，模组与主控通过 UART 进行连接通信。主控芯片（如 R328）主要实现包括 hci uart 驱动、L2CAP，以及 L2CAP 之上的各种 profile，其中 hci uart 驱动、L2CAP、rfcomm 等基本核心协议是在内核空间实现，其他基本的 profile 是在用户空间 bluez 里实现。开源的 BlueZ 协议栈主要是实现了基本的 profile，提供了一些基于 dbus 通信的接口，但是让客户二次开发，有一定难度。如开发蓝牙音乐，BlueZ 仅仅实现了 profile 部分，没有实现音频播放部分。由此 Allwinner 为了客户方便，开发了 btmanager 中间件，集成蓝牙绝大多数功能，向下对接 bluez，向上提供客户 API。

目前 SDK 已经完全适配 RTL 系列、XR 系列、AIC 系列的模组，如用户需要再使用其他模组，可以重新适配模组相关的硬件驱动即可，如 hci uart 驱动。部分模组厂商提供有自己私有协议栈的另说。

2.3 源码结构

蓝牙代码主要分为以下几个部分，从底层到上层依次展开介绍：

2.3.1 HCI 驱动

```
路径：linux-xxx/drivers/
├──── bluetooth
   ├──── btsdio.c
   ├──── btusb.c  //基于usb接口的hci驱动
   ├──── h4_recv.h
   ├──── hci_h4.c //基于uart接口的hci驱动，最常用
   ├──── hci_h5.c //基于uart接口的hci驱动
   ├──── hci_intel.c
   ├──── hci_ldisc.c //基于uart接口的hci驱动，配合hci_h4 hci_h5使用
   ├──── hci_ll.c
   ├──── hci_serdev.c
   ├──── hci_uart.h
   ├──── hci_vhci.c
   ├──── Kconfig
   └──── Makefile
```

2.3.2 内核蓝牙协议栈

```
路径：linux-xxx/net/
├──── bluetooth
   ├──── 6lowpan.c
   ├──── a2mp.c
   ├──── a2mp.h
   ├──── af_bluetooth.c
   ├──── amp.c
   ├──── amp.h
   ├──── bnep
   ├──── cmtp
   ├──── ecdh_helper.c
   ├──── ecdh_helper.h
   ├──── hci_conn.c    //控制连接相关
   ├──── hci_core.c    //hci core文件，初始化和对接hci driver
   ├──── hci_debugfs.c  //调试节点
   ├──── hci_debugfs.h
   ├──── hci_event.c    //解析hci event
   ├──── hci_request.c  //发送hci cmd相关
   ├──── hci_request.h
   ├──── hci_sock.c    //hci socket服务端
   ├──── hci_sysfs.c
   ├──── hidp
   ├──── Kconfig
   ├──── l2cap_core.c   //l2cap协议核心
   ├──── l2cap_sock.c   //l2cap socket服务端
   ├──── leds.c
   ├──── leds.h
   ├──── lib.c
```

```
├──── Makefile
├──── mgmt.c      //mgmt组件，可与bluez通信
├──── mgmt_util.c
├──── mgmt_util.h
├──── rfcomm      //rfcomm协议核心
├──── sco.c
├──── selftest.c
├──── selftest.h
├──── smp.c
└──── smp.h       //smp协议核心
```

2.3.3 BlueZ 协议栈

```
编译解压后路径：out/平台/板级/openwrt/build_dir/target/bluez-5.54/
├──── acinclude.m4
├──── aclocal.m4
├──── android  //旧版用于安卓平台的代码，已弃用
├──── attrib
├──── AUTHORS
├──── btio
├──── ChangeLog
├──── client   //bluetoothctl工具的源码
├──── doc
├──── ell
├──── emulator
├──── gdbus
├──── gobex
├──── lib     //编译生成libbluetooth.so库
├──── libtool
├──── ltmain.sh
├──── mesh
├──── missing
├──── monitor  //hcidump、btmon工具的源码
├──── NEWS
├──── obexd
├──── peripheral
├──── plugins
├──── profiles  //里面处理蓝牙具体的profiles
  ├──── audio  //里面处理a2dp avrcp等协议，最常用
  ├──── battery
  ├──── cups
  ├──── deviceinfo
  ├──── gap
  ├──── health
  ├──── iap
  ├──── input  //hid hog协议
  ├──── midi
├──── README
├──── src
  ├──── adapter.c  //控制着本地蓝牙的状态
  ├──── adapter.h
  ├──── advertising.c
  ├──── advertising.h
  ├──── agent.c   //处理配对时的认证
  ├──── agent.h
  ├──── dbus-common.c
  ├──── dbus-common.h
```

```
    ├── device.c   //维护已扫描、已连接的对端设备状态
    ├── device.h
    ├── gatt-client.c
    ├── gatt-client.h
    ├── gatt-database.c  //gatt database服务器的实现
    ├── gatt-database.h
    ├── log.c
    ├── log.h
    ├── main.c     //bluetoothd的主函数，协议栈的初始化和协议注册
    ├── main.conf
    ├── plugin.c
    ├── plugin.h
    ├── profile.c
    ├── profile.h
    ├── rfkill.c
    ├── sdp-client.c   //作为sdp客户端用到的接口
    ├── sdp-client.h
    ├── sdpd-database.c
    ├── sdpd.h
    ├── sdpd-request.c
    ├── sdpd-server.c
    ├── sdpd-service.c   //维护着本地sdp服务端
    ├── sdp-xml.c
    ├── sdp-xml.h
    ├── service.c
    ├── service.h
    ├── shared
  ├── tools
    ├── btmgmt.c
    ├── hciattach_aic.c  //aic模组的启动逻辑
    ├── hciattach.c     //hciattach工具的主函数，提供接口给第三方模组注册
    ├── hciattach.h
    ├── hciattach_intel.c
    ├── hciattach_qualcomm.c
    ├── hciattach_xradio.c  //XR模组的启动逻辑，涉及下载fimrware等
    ├── hciconfig.c       //hciconfig工具的源码
    ├── hcidump.c         //hcidump工具的源码
    ├── hcitool.c         //hcitool工具的源码
```

2.3.4 Btmanager 中间件

Tina4.0 的 SDK，代码路径：

```
btmanager3.0：
package/allwinner/btmanager
```

```
btmanager4.0：
package/allwinner/wireless/btmanager4.0
```

Tina5.0 的 SDK，代码路径：

```
btmanager4.0：
openwrt/package/allwinner/wireless/btmanager
platform/allwinner/wireless/btmanager
```

源码结构：

```
├── config
│   ├── bluetooth.json      //配置不指定协议时，默认打开蓝牙使用的协议和其他协议的参数。
│   └── bt_init.sh          //蓝牙的启动脚本，无需客户调用，btmanager里会调用。
├── demo                    //bt_test工具，基于btmanger API接口实现的测试demo，供客户参考。
│   ├── bt_classic_demo.c   //经典蓝牙部分的源码
│   ├── bt_gatt_client_demo.c  //做ble client 的源码
│   ├── bt_gatt_server_demo.c  //做ble server 的源码
│   ├── bt_test.c           //入口main函数，回调注册。
│   ├── bt_test.h
│   ├── Makefile
│   └── mesh
└── src                     //btmanager各个协议的源码
    ├── a2dp
    ├── avrcp
    ├── bt_manager.c        //将各个协议的接口统一封装成API接口
    ├── common
    ├── device
    ├── gatt
    ├── hfp
    ├── include             //头文件汇总
    ├── log
    ├── Makefile
    ├── platform
    ├── shared
    └── spp
```

3 模块配置

3.1 配置说明

蓝牙运行需要有以下几个工作条件：

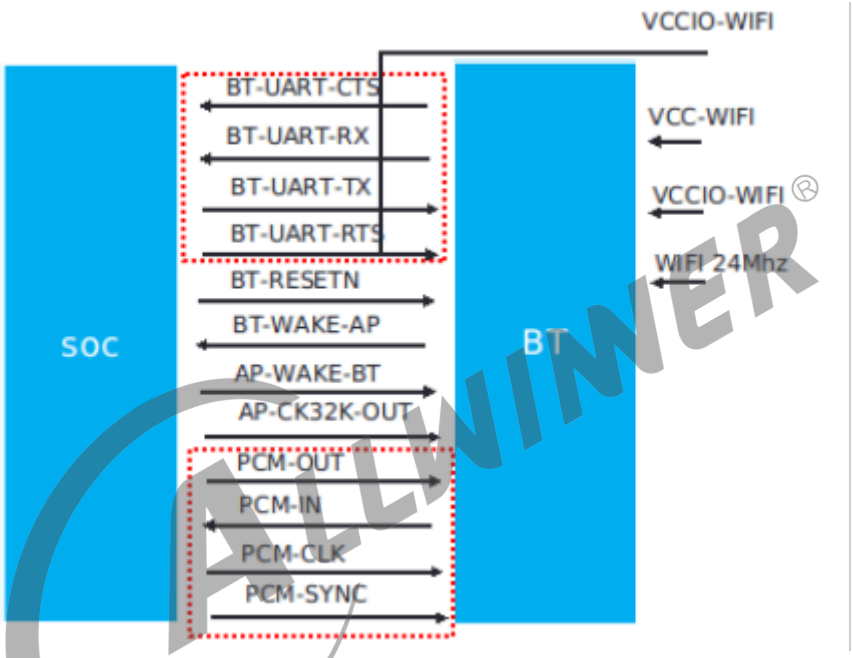

图 3-1: 主控与 BT 硬件连接简图

- 供电：蓝牙供电一般需要两路电源，一路为主电源，另一路用于 IO 上拉电源。
- 复位：需要对 BT-RESETN 进行复位操作。aic 芯片系列不需要。
- AP-WAKE-BT：主要用于使 BT 进行休眠，当 BT 正常工作时，需要输出高电平。
- 接口：主控与 BT 大部分数据通信都是通过 UART 接口，而部分模组蓝牙语音通话走 PCM 接口。
- 24/40MHz 时钟信号。
- 32.768KHz 信号：根据模组而定，有些模组内部通过输入的 CLK 进行分频得到，有些需要外部单独输入该信号。

软件上，Bluetooth 需要配置的是供电，AP-WAKE-BT 拉高，BT-RESETN 可进行复位，输出 32khz 信号。关于

供电部分，大部分的模组都是 Wi-Fi，BT 一体，所以大部分操作同 Wi-Fi 一致。

3.2 Device Tree 配置

以下是 board.dts Bluetooth 相关的配置：

```
uart1_pins_default: uart1_pins@0 {
	pins = "PD7", "PD8", "PD9", "PD10";
	function = "uart1";
};

uart1_pins_sleep: uart1_pins@1 {
	pins = "PD7", "PD8", "PD9", "PD10";
	function = "io_disabled";
};

&uart1 {
	pinctrl-names = "default", "sleep";
	pinctrl-0 = <&uart1_pins_default>;
	pinctrl-1 = <&uart1_pins_sleep>;
	status = "okay";
};

&uart1 {
  pinctrl-names = "default", "sleep";
  pinctrl-0 = <&uart1_pins_active>;
  pinctrl-1 = <&uart2_pins_sleep>;
  uart-supply = <&reg_dcdc1>;
  status = "okay";
};

bt: bt@0 {
	compatible   = "allwinner,sunxi-bt";
	/*clock-names = "32k-fanout1";*/
	/*clocks = <&ccu CLK_FANOUT1_OUT>;*/
	/*bt_power_num = <0x01>;*/
	/*bt_power     = "axp803-dldo1";*/
	/*bt_io_regulator = "axp803-dldo1";*/
	/*bt_io_vol = <3300000>;*/
	/*bt_power_vol = <330000>;*/
	bt_rst_n     = <&pio PD 23 GPIO_ACTIVE_LOW>;
	status       = "okay";
};

btlpm: btlpm@0 {
	compatible  = "allwinner,sunxi-btlpm";
	uart_index  = <0x1>;
	bt_wake     = <&pio PD 22 GPIO_ACTIVE_HIGH>;
	bt_hostwake = <&pio PD 16 GPIO_ACTIVE_HIGH>;
	status      = "okay";
};
```

uart字段：主要配置UART的TX、RX、CTS、RTS所使用的GPIO PIN，必须四线。
bt字段：主要是配置时钟、电源域、电压值、复位所使用的GPIO PIN。
btlpm字段：主要是配置休眠唤醒用的PIN脚，bt_wake是AP_WAKE_BT,bt_hostwake是BT_WAKE_AP。

3.3 menuconfig 配置

下面将 openwrt 构建和 buildroot 构建分开，展示三款常用的模组配置，XR 系列，RTL 系列、AIC 系列。

3.3.1 openwrt 构建方式

以下列出各个模组 kernel_menuconfig 以及 menuconfig 的选项。

3.3.1.1 XR 系列配置

内核部分：make kernel_menuconfig

```
Networking support  --->
 <*>  Bluetooth subsystem support  --->
  [*]  Bluetooth Classic (BR/EDR) features
  <*>    RFCOMM protocol support     //如果需要支持hfp，需要选上RFCOMM两个选项
  [*]    RFCOMM TTY support
  [*]  Bluetooth Low Energy (LE) features
  [ ]  Bluetooth self testing support
  [*]  Export Bluetooth internals in debugfs
       Bluetooth device drivers  --->

> Networking support > Bluetooth subsystem support > Bluetooth device drivers
  < > HCI USB driver
  < > HCI SDIO driver
  <*> HCI UART driver
  [*]  UART (H4) protocol support
  [ ]  BCSP protocol support
  [ ]  Atheros AR300x serial support

> Networking support
   <*>  RF switch subsystem support  --->
     --- RF switch subsystem support
     [ ]  RF switch input support
     <*>  GPIO RFKILL driver

> Allwinner BSP > Device Drivers > Network Device Drivers > Wireless LAN
  <M> XR819S WLAN support //根据实际模组修改

> Allwinner BSP > Device Drivers > Misc Devices Drivers
  <*> Allwinner rfkill driver
  <*> Allwinner Network MAC Addess Manager
  < > Enable aw bootevent debugger tool

 > Allwinner BSP > Device Drivers > Networking Drivers > Bluetooth Subsystem Support > Bluetooth Device Drivers
  < > Realtek Bluesleep driver support
  <*> Xradio Bluetooth sleep driver support //如果使用的tina4.0，这个选项位置在hci-h4驱动那栏
```

用户空间部分：make menuconfig 配置

```
> Allwinner > Wireless >
  <*> btmanager-v4.0..... bluetooth manager core
  [*]  Enable btmanager demo support
  [ ]  Enable xradio mesh support
    select serial for Bluetooth (ttyS1)  --->
  -*- wirelesscommon..... Allwinner Wi-Fi/BT Public lib

> Utilities
  -*- bluez-daemon..... Bluetooth daemon
  [ ] Enable xr829 extra config
  -*- bluez-utils..... Bluetooth utilities
  -*- bluez-utils-extra..... Bluetooth additional utilities

> Multimedia
  -*- bluez-alsa..... Bluetooth Audio ALSA backend

> Kernel modules > Wireless Drivers
  <*> kmod-net-xr819s..... xr819s support (staging)
  //如果原配置是其他模组，请先取消勾选

> Firmware
  (/lib/firmware/) Firmware's directory
  [ ] xr829 with 40M sdd
  [*] xr819s with 40M sdd
  -*- xr819s-firmware..... Xradio xr819s firmware
  //如果原配置是其他模组，请先取消勾选

> Utilities
  rf test tool  --->
    -*- xr819s-rftest..... xr819s rf test tools
```

3.3.1.2 RTL87xx 系列配置

内核部分配置：make kernel_menuconfig

```
Networking support  --->
 <*>  Bluetooth subsystem support  --->
  [*]  Bluetooth Classic (BR/EDR) features
  <*>    RFCOMM protocol support     //如果需要支持hfp，需要选上RFCOMM两个选项
  [*]     RFCOMM TTY support
  [*]  Bluetooth Low Energy (LE) features
  [ ]  Bluetooth self testing support
  [*]  Export Bluetooth internals in debugfs
      Bluetooth device drivers  --->

Networking support  --->
 <*>  Bluetooth subsystem support  --->
  Bluetooth device drivers  --->
   [*] Realtek Three-wire UART (H5) protocol support  //rtl不是用h4驱动，用自己的h5驱动
```

用户空间配置：make menuconfig

```
> Allwinner > Wireless >
  <*> btmanager-v4.0..... bluetooth manager core
  [*]  Enable btmanager demo support
  [ ]  Enable xradio mesh support
    select serial for Bluetooth (ttyS1)  --->
```

```
   -*- wirelesscommon..... Allwinner Wi-Fi/BT Public lib

> Utilities
   -*- bluez-daemon..... Bluetooth daemon
   -*- bluez-utils..... Bluetooth utilities
   -*- bluez-utils-extra..... Bluetooth additional utilities

Kernel modules--->
 Wireless Drivers--->
   <*> kmod-net-rtl8723ds..... RTL8723DS support (staging)

Firmware--->
 <*> r8723ds-firmware..... RealTek RTL8723DS firmware

Utilities  --->
 rtk_hciattach  --->
   <*> rtk_hciattach..... Realtek BT HCI UART initialization tools
```

3.3.1.3 AIC 系列配置

内核部分配置：make kernel_menuconfig

```
> Networking support > Bluetooth subsystem support
   --- Bluetooth subsystem support
   [*]   Bluetooth Classic (BR/EDR) features
   <*>     RFCOMM protocol support
   [*]       RFCOMM TTY support
   < >     BNEP protocol support
   < >     HIDP protocol support
   [*]     Bluetooth High Speed (HS) features
   [*]   Bluetooth Low Energy (LE) features
   [ ]   Bluetooth self testing support
   [*]   Export Bluetooth internals in debugfs
         Bluetooth device drivers  --->

> Networking support > Bluetooth subsystem support > Bluetooth device drivers
   < > HCI USB driver
   < > HCI SDIO driver
   <*> HCI UART driver
   [*]   UART (H4) protocol support
   [ ]   BCSP protocol support
   [ ]   Atheros AR300x serial support

> Networking support
    <*>   RF switch subsystem support  --->
      --- RF switch subsystem support
      [ ]   RF switch input support
      <*>   GPIO RFKILL driver

> Allwinner BSP > Device Drivers > Network Device Drivers > Wireless LAN
   [*] AIC wireless Support
         Enable Chip Interface (SDIO interface support)  --->
       Choice host wake IRQ type (UNSET IRQ type use default config)  --->
   <M> AIC8800 wlan Support
   <M> AIC8800 bluetooth Support (UART)

> Allwinner BSP > Device Drivers > Misc Devices Drivers
```

```
  <*> Allwinner rfkill driver
  <*> Allwinner Network MAC Addess Manager
  < > Enable aw bootevent debugger tool
```

用户空间配置：make menuconfig

```
> Allwinner > Wireless >
  <*> btmanager-v4.0..... bluetooth manager core
  [*]   Enable btmanager demo support
  [ ]   Enable xradio mesh support
    select serial for Bluetooth (ttyS1)  --->
  -*- wirelesscommon..... Allwinner Wi-Fi/BT Public lib

> Utilities
  -*- bluez-daemon..... Bluetooth daemon
  [ ] Enable xr829 extra config
  -*- bluez-utils..... Bluetooth utilities
  -*- bluez-utils-extra..... Bluetooth additional utilities

> Multimedia
  -*- bluez-alsa..... Bluetooth Audio ALSA backend

> Kernel modules > Wireless Drivers
  <*> kmod-net-aic8800..... aic8800 support (staging)
  //如果原配置是其他模组，请先取消勾选

> Firmware
  -*- aic8800-firmware..... AIC aic8800 firmware
  //如果原配置是其他模组，请先取消勾选

> Utilities
  rf test tool  --->
    -*- aic8800-rftest..... aic8800 rf test tools
```

3.3.2 buildroot 构建方式

1. 内核部分的配置项，与上面的 openwrt 方式配置项一样，执行：

```
./build.sh menuconfig
```

配置内容：

```
> Networking support
  <*>  Bluetooth subsystem support  --->
    [*]  Bluetooth Classic (BR/EDR) features
    <*>    RFCOMM protocol support
    [*]    RFCOMM TTY support
    < >    BNEP protocol support
    < >    HIDP protocol support
    [*]    Bluetooth High Speed (HS) features
    [*]  Bluetooth Low Energy (LE) features
    [ ]  Bluetooth self testing support
    [*]  Export Bluetooth internals in debugfs
      Bluetooth device drivers  --->
        <*> HCI UART driver
        [*]  UART (H4) protocol support  //若使用rtk模组，这里改为h5驱动
        <*> Xradio Bluetooth sleep driver support
```

```
<*>  RF switch subsystem support  --->
  [ ]  RF switch input support
  <*>  GPIO RFKILL driver
```

2. 用户空间的配置项，与上述的 openwrt 构建方式界面有差异，执行：

```
./build.sh buildroot_menuconfig
```

配置内容：

```
Target packages  --->
  allwinner platform private package select  --->
    wireless  --->
      -*- wireless_common
      [*] btmanager-core
      [*]  Enable btmanager demo support
        firmware  --->
          [*] wifi bt firmware
           [*]  xr829-firmware //根据实际使用的模组修改
           [*]   xr829_40M
  Audio and video applications  --->
    -*- bluez-alsa
  Networking applications  --->
    [ ] bluez-tools
    [ ] bluez-utils
    -*- bluez-utils
      [ ]  build OBEX support
      [*]  build CLI client
      [*]  build monitor utility
      [*]  build tools
      [*]   install deprecated tools
      [ ]  build experimental tools
      -*-  build audio plugins (a2dp and avrcp)
```

4 模块接口说明

4.1 通用 API

API 接口	说明
bt_manager_set_loglevel	设置 bt_manager 内部打印等级，默认打印等级是 BTMG_LOG_LEVEL_INFO。
bt_manager_get_loglevel	获取 bt_manager 内部当前使用的打印等级。
bt_manager_set_ex_debug_mask	设置拓展调试标志位，用于打开特殊打印输出。
bt_manager_get_ex_debug_mask	获取当前设置拓展调试标志位的值。
bt_manager_get_error_info	通过错误码获取对应的错误提示信息。
bt_manager_preinit	用于对用户定义的回调函数结构体指针 btmg_callback_t * 进行分配内存初始化，用户也可自行显示地对指针进行初始化。初始化的指针在用户程序 exit 之前必须调用。
bt_manager_init	bt_manager 的初始化，为内部变量分配空间。
bt_manager_deinit	bt_manager 反初始化。在 bt_manager 蓝牙应用程序退出前必要进行的调用。
bt_manager_enable_profile	设置蓝牙的 profile 功能，在 bt_manager_enable() 之前设置有效，后续在用户调用 bt_manager 的进程未退出的情况下该设置一直有效。
bt_manager_set_enable_default	是否使用 bluetooth.json 配置文件的默认配置。在 bt_manager_enable() 之前设置有效，后续在用户调用 bt_manager 的进程未退出的情况下该设置一直有效。
bt_manager_enable	开关蓝牙的作用, 涉及 profile 配置文件、btmanager 内部线程、hci 设备、蓝牙协议栈等操作。
bt_manager_set_scan_mode	设置本地 BT 设备扫描模式（PSCAN,ISCAN），决定设备是否可连接，可发现。
bt_manager_scan_filter	扫描过滤设置，如指定经典蓝牙扫描，指定信号强度扫描。
bt_manager_start_scan	开始扫描周围的蓝牙设备。
bt_manager_stop_scan	停止 BT 扫描设备的动作。
bt_manager_is_scanning	获得当前 BT 扫描状态。

API 接口	说明
bt_manager_pair	配对蓝牙设备，配对状态请根据 gap 回调函数 gap_bond_state_cb() 判断。
bt_manager_unpair	取消配对，如果设备处于连接状态，会先断开后再取消配对；配对状态请根据 gap 回调函数 gap_bond_state_cb() 判断。
bt_manager_get_paired_devices	获取已配对列表，获取完后，务必使用 bt_manager_free_paired_devices(btmg_bt_device_t *dev_list, int count) 释放 dev_list，避免内存泄漏。
bt_manager_free_paired_devices	用于释放 bt_manager_get_paired_devices() API 获取已配对设备时已申请到的内存，避免内存泄漏。
bt_manager_get_adapter_state	获取蓝牙开关状态。
bt_manager_get_adapter_name	获取本地蓝牙设备名称。
bt_manager_set_adapter_name	设置本地蓝牙设备名称。
bt_manager_get_adapter_address	获取本地蓝牙设备地址。
bt_manager_get_device_name	获取指定蓝牙设备的名称。
bt_manager_connect	蓝牙通用连接。
bt_manager_disconnect	蓝牙通用断开连接。
bt_manager_device_is_connected	判断对端设备是否跟本地设备存在连接关系。
bt_manager_remove_device	移除指定缓存的蓝牙设备，移除扫描列表缓存。
bt_manager_set_page_timeout	设置 page timeout，即蓝牙链路建立的最大超时时间。
bt_manager_set_link_supervision_timeout	设置 Supervision 超时时间；Supervision Timeout 作用: 判断设备断开的超时时间，ACL 连接双方约定多长时间没有联系上算断开；如果要设置，一般在设备刚连接上的时候设置，断开后会失效，因此每次连接上都需要设置一次；例如：设备已经连上一个蓝牙音箱，此时主动将蓝牙音箱的电关闭，应用层可能会过一段时间才能收到断开的事件。如果设置了 Supervision Timeout 为 1000ms，那么蓝牙音箱断电 1000ms 后，应用层就可以收到断开的事件了。

4.2 BlueZ Agent API

BlueZ 协议栈使用 agent(代理) 来实现配对，配对过程中的交互都由 agent 来完成，代理根据不同的配对方式有不同的行为。

API 接口	说明
bt_manager_agent_set_io_capability	设置本地蓝牙设备的输入、输出能力。
bt_manager_agent_send_pincode	回复 Pincode 给对端设备。
bt_manager_agent_send_passkey	回复 passkey 给对端设备。
bt_manager_agent_pair_send_empty_response	发送一个空消息给对端设备。
bt_manager_agent_send_pair_error	回复配对错误给对端设备。

4.3 A2DP Sink API

无

4.4 AVRCP-sink API

该部分 AVRCP 接口，是板子作为 A2DP Sink 时可使用。

API 接口	说明
bt_manager_avrcp_command	发送命令控制 A2DP Source 设备的播放、暂停、上下一曲。
bt_manager_a2dp_set_vol	调节音量 [0~100]。
bt_manager_a2dp_get_vol	获取当前音量值。

4.5 A2DP Source API

API 接口	说明
bt_manager_a2dp_src_init	A2DP Source 初始化，根据传入的通道、采样率和位深进行初始化；初始化务必在 “启动播放线程” 和 “发送音频数据” 之前完成；后面发送的音频参数务必和初始化的参数保持一致，否则需要反初始化后再重新初始化相应的音频参数。
bt_manager_a2dp_src_deinit	A2DP Source 反初始化。
bt_manager_a2dp_src_stream_start	启动内部播放线程。
bt_manager_a2dp_src_stream_send	应用发送音频数据给蓝牙协议栈。
bt_manager_a2dp_src_is_stream_start	判读当前是否处于发送状态。
bt_manager_a2dp_src_stream_stop	停止内部播放线程，停止播放。

4.6 AVRCP-source API

该部分 AVRCP 接口，是板子作为 A2DP Source 时搭配使用。

API 接口	说明
bt_manager_a2dp_set_vol	设置蓝牙音乐的音量大小，此种方式 bluealsa 直接对 pcm 数据调增益。
bt_manager_a2dp_get_vol	获取当前蓝牙音乐的音量大小。
bt_manager_avrcp_set_abs_volume	设置蓝牙绝对音量的大小，此种方式是通知对方调节音量。
bt_manager_avrcp_get_abs_volume	获取当前蓝牙绝对音量的大小。
bt_manager_set_avrcp_status	设置 A2DP 蓝牙音乐的播放、暂停，上下曲。
bt_manager_get_avrcp_status	获取 A2DP 蓝牙音乐的状态。

4.7 HFP HF API

API 接口	说明
bt_manager_hfp_hf_send_at_ata	主动接听电话。
bt_manager_hfp_hf_send_at_chup	主动拒接或挂断电话。
bt_manager_hfp_hf_send_at_atd	指定电话号码拨号。
bt_manager_hfp_hf_send_at_bldn	回拨最后一个通话的号码。
bt_manager_hfp_hf_send_at_btrh	查询与报告通话的状态。
bt_manager_hfp_hf_send_at_vts	拨打分机，发送 AT+VTS=“分机号码”。
bt_manager_hfp_hf_send_at_bcc	发送 AT 命令 AT+BCC 给 AG，发起音频编解码连接。
bt_manager_hfp_hf_send_at_cnum	获取本机号码，本机号码将通过回调函数返回。
bt_manager_hfp_hf_send_at_vgs	调节手机端扬声器音量大小。
bt_manager_hfp_hf_send_at_vgm	调节手机端麦克风音量大小。
bt_manager_hfp_hf_send_at_cmd	用户可以根据实际情况构建自己需要的 AT 命令，命令格式必须是 “AT+XXX” 的格式。

4.8 HFP AG API

该功能需 Btmanager Version:4.0.4.20231120 后的版本才支持。

API 接口	说明
bt_manager_hfp_ag_sco_conn_cmd	发起 sco 连接。

API 接口	说明
bt_manager_hfp_ag_sco_disconn_cmd	断开 sco 链路。

4.9 SPP Client API

API 接口	说明
bt_manager_spp_client_connect	连接 SPP Server 设备，注意，如果目标设备是第一次连接，需要先通过 bt_manager_pair() 配对后再调用此接口连接。
bt_manager_spp_client_send	发送数据给对端设备。
bt_manager_spp_client_disconnect	断开与对端设备连接。

4.10 SPP Server API

API 接口	说明
bt_manager_spp_service_accept	初始化 SPP Server 并开设监听 Client 设备。
bt_manager_spp_service_send	发送数据到对端设备。
bt_manager_spp_service_disconnect	断开与对端设备连接。

4.11 GATT Server API

API 接口	说明
bt_manager_le_set_random_address	设置 BLE 广播使用的随机地址。
bt_manager_le_set_adv_param	设置广播的参数，如广播的间隔、广播的信道，广播的类型等。
bt_manager_le_set_adv_data	设置广播携带的数据内容，如 BLE 设备名称、UUID 等。
bt_manager_le_enable_adv	打开或者关闭广播；打开广播之前先设置广播的参数和数据内容。
bt_manager_le_set_scan_rsp_data	设置扫描响应数据。
bt_manager_le_disconnect_all	断开当前所有的 BLE 连接。
bt_manager_le_disconnect	断开指定 handle 的 BLE 连接。
bt_manager_le5_ext_adv_set_rand_addr	设置拓展广播使用的随机地址；多个广播用 handle 区分，需分别设置。

API 接口	说明
bt_manager_le5_ext_adv_set_params	设置拓展广播的广播参数；多个广播用 handle 区分，需分别设置。
bt_manager_le5_ext_adv_data_raw	设置拓展广播携带的数据；多个广播用 handle 区分，需分别设置；只有可连接类型的广播需设置。
bt_manager_le5_ext_scan_rsp_data_raw	设置拓展广播的扫描回复的数据；多个广播用 handle 区分，需分别设置；只有可扫描类型的广播需设置。
bt_manager_le5_ext_adv_enable	打开或者关闭拓展广播。
bt_manager_le5_ext_adv_set_remove	清除指定 handle 的广播信息。
bt_manager_le5_ext_adv_set_clear_all	清除所有 handle 的广播信息。
bt_manager_le5_periodic_adv_set_params	设置周期广播的广播参数。
bt_manager_le5_periodic_adv_data_raw	设置周期广播携带的数据内容。
bt_manager_le5_periodic_adv_enbale	设置周期广播的扫描回复的数据内容。
bt_manager_le5_set_phy	设置 BLE 要使用的 phy 类型。
bt_manager_gatt_server_init	gatt server 初始化。
bt_manager_gatt_server_deinit	gatt server 反初始化。
bt_manager_gatt_server_create_service	创建一个 service。
bt_manager_gatt_server_add_characteristic	指定服务中添加 characteristic。
bt_manager_gatt_server_add_descriptor	指定服务中添加 descriptor。
bt_manager_gatt_server_start_service	启动指定的 service；创建 service 后不需要调用该 API，一般用于启动已经停止的 service。
bt_manager_gatt_server_stop_service	停止指定的 service。
bt_manager_gatt_server_remove_service	删除一个 gatt service，删除后如果还要使用，需要重新创建。
bt_manager_gatt_server_send_read_response	Client 端读取 server 属性的时候，会激活对应的回调函数，server 通过该函数回复读请求，以通知 client 读是否成功。
bt_manager_gatt_server_send_write_response	Client 端写 server 属性的时候，会激活对应的回调函数，server 通过该函数回复写请求，以通知 client 写是否成功。
bt_manager_gatt_server_send_notify	Server 通过该函数通知 client，client 不需要回复。
bt_manager_gatt_server_send_indication	Server 通过该函数指示 client，client 需要回复。
bt_manager_gatt_server_get_mtu	获取当前 mtu。

4.12 GATT Client API

API 接口	说明
bt_manager_gatt_client_init	初始化 GATT Client 功能。
bt_manager_gatt_client_deinit	反初始化 GATT Client 功能。
bt_manager_gatt_client_ecode_to_string	通过错误码获取相应的错误提示信息。
bt_manager_uuid_to_uuid128	把 16 位的 UUID 转换为 128 位的 UUID。
bt_manager_uuid_to_string	把 UUID 转换为字符串。
bt_manager_le_set_scan_parameters	设置 BLE 扫描参数。
bt_manager_le_scan_start	开始扫描 BLE 设备，扫描结果通过 gap_le_scan_report_cb 回调函数返回。
bt_manager_le_scan_stop	停止扫描 BLE 设备。
bt_manager_le_update_conn_params	更新 BLE 连接参数。
bt_manager_le_add_white_list	添加指定设备到白名单。
bt_manager_le_rm_white_list	移除指定设备的白名单。
bt_manager_le_read_white_list_size	读取白名单的最大存储数。
bt_manager_le_clear_white_list	清空白名单。
bt_manager_le5_set_ext_scan_params	设置 BLE 拓展扫描参数。
bt_manager_le5_start_ext_scan	开始拓展扫描 BLE 设备，扫描结果通过 gap_le5_ext_scan_report_cb 回调函数返回。
bt_manager_le5_stop_ext_scan	停止拓展扫描 BLE 设备。
bt_manager_gatt_client_connect	连接 GATT Server 设备，设备最终连接状态请通过回调函数 gattc_conn_cb 判断，包含连接标识符。
bt_manager_gatt_client_disconnect	断开与 GATT Server 设备的连接，设备最终连接状态请通过回调函数 gattc_conn_cb 判断。
bt_manager_gatt_client_get_conn_list	获取已连接设备的列表，结果通过 gattc_connected_list_cb 返回，目前是每个设备会触发一次回调。
bt_manager_gatt_client_get_mtu	获取当前连接协商后使用的 MTU 大小。
bt_manager_gatt_client_set_security	设置 GATT 安全级别。
bt_manager_gatt_client_get_security	获取 GATT 安全级别。
bt_manager_gatt_client_register_notify	注册 notify 或 indicate，协议栈底层已经包含写 CCC 的操作，因此无需再写 “Client Characteristic Configuration”，注册完成后，Server 端才可以对 Client 通知或者指示。
bt_manager_gatt_client_unregister_notify_indicate	注销 notify 或 indicate。
bt_manager_gatt_client_write_request	往 GATT Server 设备写数据，Server 需要 response，最终的写入结果需要通过 gattc_write_cb 回调函数获取。

API 接口	说明
bt_manager_gatt_client_write_long_request	往 GATT Server 设备写长数据，Server 需要 response；最终的写入结果需要通过 gattc_write_long_cb 回调函数获取。
bt_manager_gatt_client_write_command	往 GATT Server 设备写命令，Server 不需要 response。
bt_manager_gatt_client_read_request	读取 GATT Server 设备的数据；最终的读取结果需要通过 gattc_read_cb 回调函数获取。
bt_manager_gatt_client_read_long_request	读取 GATT Server 设备的数据；最终的读取结果需要通过 gattc_read_cb 回调函数获取。
bt_manager_gatt_client_discover_all_services	从指定 handle 范围中发现所有的服务；发现的结果通过 gattc_dis_service_cb 回调函数返回。
bt_manager_gatt_client_discover_services_by_uuid	指定 uuid 从指定 handle 范围中发现所有符合条件的服务；发现的结果通过 gattc_dis_service_cb 回调函数返回。
bt_manager_gatt_client_discover_service_all_char	发现指定 Service 里面的所有 Characteristic；发现的结果通过 gattc_dis_char_cb 回调函数返回。
bt_manager_gatt_client_discover_char_all_descriptor	发现指定 Characteristic 里面的所有 descriptor；发现的结果通过 gattc_dis_desc_cb 回调函数返回。

5 功能开发

5.1 配置文件介绍

btmanager 依赖的配置文件有 bt_init.sh 和 bluetooth.json 两个文件。

小机端路径：

```
/etc/bluetooth/
```

SDK 源码路径：

```
tina4.0：
tina/package/allwinner/wireless/btmanager4.0/config/
tina5.0：
tina/platform/allwinner/wireless/btmanager/config/
```

5.1.1 bt_init.sh

作用：开启或关闭蓝牙的操作集合，包括控制节点复位、执行 hciattach 模组初始化、加载协议栈 bluetoothd 进程等。

执行时机：bt_manager_enable() 接口里面会执行此脚本，用户不需要再单独执行。当然用户可以在其他位置调用，可重复执行。

需要注意，SDK 源码路径下的 bt_init.sh 不一定和小机端的内容相同。因为各模组的 hciattach 有差异，使用的 UART 口差异。所以在 btmanager 的 Makefile 里会根据配置的 UART 和模组使能情况，通过 sed 修改 bt_init.sh，因此最终的 bt_init.sh 脚本需看小机端/etc/bluetooth/bt_init.sh 文件。

如搭配 XR829 模组，最终内容如下：

```
#!/bin/sh
bt_hciattach="hciattach"

//下电再上电，达到复位效果，需提前配置bt的pin脚。部分模组不需要。
reset_bluetooth_power()
{
   echo 0 > /sys/class/rfkill/rfkill0/state;
   sleep 1
   echo 1 > /sys/class/rfkill/rfkill0/state;
   sleep 1
}
```

```
//指定参数执行hciattach，生成hci0节点。
start_hci_attach()
{
  h=`ps | grep "$bt_hciattach" | grep -v grep`
  [ -n "$h" ] && {
    killall "$bt_hciattach"
  }

  # reset_bluetooth_power

  //下载firmware，模组初始化
  "$bt_hciattach" -n ttyS1 xradio >/dev/null 2>&1 &

  wait_hci0_count=0
  while true
  do
    [ -d /sys/class/bluetooth/hci0 ] && break
    usleep 100000
    let wait_hci0_count++
    [ $wait_hci0_count -eq 70 ] && {
      echo "bring up hci0 failed"
      exit 1
    }
  done
}

start() {

  if [ -d "/sys/class/bluetooth/hci0" ];then
    echo "Bluetooth init has been completed!!"
  else
    start_hci_attach
  fi

  d=`ps | grep bluetoothd | grep -v grep`
  [ -z "$d" ] && {
    # bluetoothd -n &

    //启动BlueZ协议栈
    /etc/bluetooth/bluetoothd start
    sleep 1
  }
}

ble_start() {
  if [ -d "/sys/class/bluetooth/hci0" ];then
    echo "Bluetooth init has been completed!!"
  else
    start_hci_attach
  fi

  hci_is_up=`hciconfig hci0 | grep UP`
  [ -z "$hci_is_up" ] && {
    hciconfig hci0 up
  }
}

stop() {
  d=`ps | grep bluetoothd | grep -v grep`
  [ -n "$d" ] && {
```

```
    killall bluetoothd
    sleep 1
  }

  h=`ps | grep "$bt_hciattach" | grep -v grep`
  [ -n "$h" ] && {
    killall "$bt_hciattach"
    sleep 1
  }
  echo 0 > /sys/class/rfkill/rfkill0/state;
  sleep 1
  echo "stop bluetoothd and hciattach"
}

case "$1" in
 start|"")
    start
    ;;
 stop)
    stop
    ;;
 ble_start)
    ble_start
    ;;
 *)
    echo "Usage: $0 {start|stop}"
    exit 1
esac
```

- reset_bluetooth_power: 对蓝牙模组的 reset 引脚上下电，达到复位目的，部分模组才需要执行；
- start_hci_attach: 调用 hciattach，实现和蓝牙模组 uart 波特率同步，下载 firmware，生成 hci0 节点；
- start: 启动蓝牙，包括执行 hciattach 和 bluetoothd，其被 bt_manager_enable 函数调用；
- ble_start: 仅启动 BLE，通过此方式不需要启动 bluetoothd，适合只用 ble 场景，启动快；
- stop: 关闭蓝牙，kill 掉蓝牙相关进程，其被 bt_manager_enable 函数调用。

5.1.2 bluetooth.json

作用：配置没指定协议时默认使能的协议，以及配置 a2dp 等协议的参数。

使用时机：在开启蓝牙时，被 bt_manager_enable 函数解析。

内容如下：

```
{
    "profile":{
        "a2dp_sink":1,
        "a2dp_source":0,
        "avrcp":1,
        "hfp_hf":0,
        "hfp_ag":0,
```

```
        "spp":0,
        "gatt_client":0,
        "gatt_server":0
        "smp":0
    },
    "a2dp_sink":{
        "device":"default",
        "buffer_time":400000,
        "period_time":100000,
        "cache_time_ms":200
    },
    "a2dp_source":{
        "hci_index":0,
        "DEV":"00:00:00:00:00:00",
        "DELAY":20000
    },
    "hfp_pcm":{
        "interface_type":"uart",
        "rate":8000,
        "phone_to_dev_cap":"hw:snddaudio1",
        "phone_to_dev_play":"default",
        "dev_to_phone_cap":"CaptureMic",
        "dev_to_phone_play":"hw:snddaudio1"
    }
}
```

- profile 指默认使能的 profile；

如果用户没有主动调用 bt_manager_set_default_profile() 使能指定的 profile 时，将默认从这个条目读取配置作为默认使能的 profile。

- a2dp_sink 指 a2dp sink 播放音频相关的配置；

device：表示使用的硬件声卡；

buffer_size：alsa 的 buffer size 参数；

period_size：alsa 的 period size 参数。

cache_time_ms：设置缓存 buff 的大小。

- a2dp_source：指 a2dp_source 的配置参数，当前尚未使用；
- hfp_pcm ：指 hfp over pcm 的配置参数。

interface_type：接口类型，如 XR829 音频数据走 PCM，AIC 系列走 UART。

rate：蓝牙 pcm 使用的采样率，跟具体蓝牙模组有关；

phone_to_dev_cap：主控端从蓝牙模组获取蓝牙通话音频的声卡（手机先传给蓝牙模组，蓝牙模组再通过 i2s 传给主控端，也就是对端手机讲话的声音);

phone_to_dev_play：对端手机讲话的声音在主控端进行播放的声卡；

dev_to_phone_cap：表示录制我方讲话声音的声卡；

dev_to_phone_play：表示我方声音写入蓝牙模组的声卡（传输到对端手机中)。

5.2 开发流程

5.2.1 A2DP Sink 功能调用 API 实现

关于 A2DP sink 功能的实现，已编写了使用示例，供用户参考，代码路径如下：

```
btmanager/demo/bt_test.c
btmanager/demo/bt_classic_demo.c
```

打开蓝牙 sink 功能，手机连接，控制播放暂停，其整个过程 api 调用步骤：

1. 调用 _bt_init(BTMG_A2DP_SINK_ENABLE)，初始化回调、打开蓝牙和添加 cmd table。
2. 打开蓝牙成功会有回调到 bt_test_adapter_status_cb。
3. 调用 bt_manager_set_adapter_name，设置蓝牙名称。
4. 调用 bt_manager_agent_set_io_capability，设置 io_capability 能力。
5. 调用 bt_manager_set_scan_mode，设置发现模式。
6. 此时蓝牙协议栈都已经准备好，手机端可以发起连接，连接成功的回调是 bt_test_a2dp_sink_connection_state_cb。
7. 获取播放状态、歌曲进度、歌曲信息通过 bt_callback->btmg_avrcp_cb 回调得到。
8. 主动控制播放、暂停、上下一曲，通过 avrcp 实现，api 调用参考 cmd_bt_avrcp。
9. 获取、设置音量，api 调用参考 cmd_bt_a2dp_set_vol/cmd_bt_a2dp_get_vol。
10. 主动断开连接，api 调用参考 cmd_bt_device_disconnect。
11. 关闭蓝牙，api 调用参考 cmd_bt_enable。

5.2.2 A2DP Source 功能调用 API 实现

关于 A2DP source 功能的实现，已编写了使用示例，供用户参考，代码路径如下：

```
btmanager/demo/bt_test.c
btmanager/demo/bt_classic_demo.c
```

打开蓝牙 source 功能，连接上音箱播放音乐，其整个过程 api 调用步骤：

1. 调用 _bt_init(BTMG_A2DP_SOUCE_ENABLE)，初始化回调、打开蓝牙和添加 cmd table。

2. 打开蓝牙成功会有回调到 bt_test_adapter_status_cb。
3. 调用 bt_manager_set_adapter_name，设置蓝牙名称。
4. 调用 bt_manager_agent_set_io_capability，设置 io_capability 能力。
5. 调用 bt_manager_set_scan_mode，设置发现模式。
6. 此时蓝牙协议栈都已经准备好，可以扫描周围设备，发起连接，播放音频。
7. 扫描周围设备，api 调用参考 cmd_bt_scan。
8. 查看扫描到的设备列表，api 调用参考 cmd_bt_inquiry_list。
9. 发起连接音箱等设备，api 调用参考 cmd_bt_device_connect。连接成功的回调是 bt_test_a2dp_source_connection_state_cb。
10. 播放音乐，api 调用参考 cmd_bt_a2dp_src_run。其具体实现如下：
11. 先读取指定音频头部 (wav 格式)，获取该音频的采样率和通道数，然后调用 bt_manager_a2dp_src_init 进行 source 初始化。注意如果切换采样率，需要重新调用这个接口设置采样率。
12. 开启 source stream 传输工作，调用 bt_manager_a2dp_src_stream_start(A2DP_SRC_BUFF_SIZE)。
13. 完成上面的工作，就可以一直发送音频数据，通过接口 bt_manager_a2dp_src_stream_send，该接口会将数据送到蓝牙虚拟声卡 “bluealsa:DEV=mac”。
14. 停止播放调用 bt_manager_a2dp_src_stream_stop，这个接口会有短暂的阻塞时间，因为将缓存 buff 数据播放完。
15. 获取、设置音量，api 调用参考 cmd_bt_a2dp_set_vol/cmd_bt_a2dp_get_vol。
16. 主动断开连接，api 调用参考 cmd_bt_device_disconnect。
17. 关闭蓝牙，api 调用参考 cmd_bt_enable。

5.2.3 HFP HF 功能调用 API 实现

关于 HFP HF 功能的实现，已编写了使用示例，供用户参考，代码路径如下：

```
btmanager/demo/bt_test.c
btmanager/demo/bt_classic_demo.c
```

打开蓝牙 HFP HF 功能，手机连接，拨打/接通电话，其整个过程 api 调用步骤：

1. 调用 _bt_init(BTMG_HFP_HF_ENABLE | BTMG_A2DP_SINK_ENABLE)，初始化回调、打开蓝牙和添加 cmd table。
2. 打开蓝牙成功会有回调到 bt_test_adapter_status_cb。
3. 调用 bt_manager_set_adapter_name，设置蓝牙名称。
4. 调用 bt_manager_agent_set_io_capability，设置 io_capability 能力。
5. 调用 bt_manager_set_scan_mode，设置发现模式。

6. 此时蓝牙协议栈都已经准备好，手机端可以发起连接，连接成功 hfp 的回调是 bt_test_hfp_hf_connection_state_cb。
7. 接收 at 命令，通过回调 bt_test_hfp_event_cb。
8. 主动发送 at 命令，如接听电话、拨打电话、查询电话号码等，api 调用参考 cmd_bt_hfp_answer_call、cmd_bt_hfp_hangup 等。
9. 主动断开连接，api 调用参考 cmd_bt_device_disconnect。
10. 关闭蓝牙，api 调用参考 cmd_bt_enable。

5.2.4 HFP AG 功能调用 API 实现

关于 HFP AG 功能的实现，已编写了使用示例，供用户参考，代码路径如下：

```
btmanager/demo/bt_test.c
btmanager/demo/bt_classic_demo.c
```

打开 HFP AG 功能，连接 hfp-hf 端通讯，其整个过程 api 调用步骤：

1. 调用 _bt_init(BTMG_HFP_AG_ENABLE)，初始化回调、打开蓝牙和添加 cmd table。
2. 打开蓝牙成功会有回调到 bt_test_adapter_status_cb。
3. 调用 bt_manager_set_adapter_name，设置蓝牙名称。
4. 调用 bt_manager_agent_set_io_capability，设置 io_capability 能力。
5. 调用 bt_manager_set_scan_mode，设置发现模式。
6. 此时蓝牙协议栈都已经准备好，扫描后发起连接 bt_manager_connect，再发起 bt_manager_hfp_ag_sco_conn_cmd 创建 sco 链路。
7. 连接成功的回调是 hfp_ag_connection_state_cb。
8. 主动断开连接，api 调用参考 bt_manager_hfp_ag_sco_disconn_cmd。
9. 关闭蓝牙，api 调用参考 cmd_bt_enable。

5.2.5 SPP Client 功能调用 API 实现

关于 SPP Client 功能的实现，已编写了使用示例，供用户参考，代码路径如下：

```
btmanager/demo/bt_test.c
btmanager/demo/bt_classic_demo.c
```

打开 SPP Client 功能，连接 server 端通讯，其整个过程 api 调用步骤：

1. 调用 _bt_init(BTMG_SPP_ENABLE)，初始化回调、打开蓝牙和添加 cmd table。
2. 打开蓝牙成功会有回调到 bt_test_adapter_status_cb。
3. 调用 bt_manager_set_adapter_name，设置蓝牙名称。
4. 调用 bt_manager_agent_set_io_capability，设置 io_capability 能力。

5. 调用 bt_manager_set_scan_mode，设置发现模式。
6. 此时蓝牙协议栈都已经准备好，扫描后发起配对 bt_manager_pair，再发起 bt_manager_spp_client_connect 连接 spp。
7. 连接成功的回调是 bt_test_spp_client_connection_state_cb。
8. 发送数据，api 调用参考 cmd_bt_spp_client_send。
9. 接收数据，通过回调 bt_test_spp_client_recvdata_cb。
10. 主动断开连接，api 调用参考 cmd_bt_spp_client_disconnect。
11. 关闭蓝牙，api 调用参考 cmd_bt_enable。

5.2.6 SPP Server 功能调用 API 实现

关于 SPP Server 功能的实现，已编写了使用示例，供用户参考，代码路径如下：

```
btmanager/demo/bt_test.c
btmanager/demo/bt_classic_demo.c
```

打开 SPP Server 功能，和 client 端建立连接进行通讯，其整个过程 api 调用步骤：

1. 调用 _bt_init(BTMG_SPP_ENABLE)，初始化回调、打开蓝牙和添加 cmd table。
2. 打开蓝牙成功会有回调到 bt_test_adapter_status_cb。
3. 调用 bt_manager_set_adapter_name，设置蓝牙名称。
4. 调用 bt_manager_agent_set_io_capability，设置 io_capability 能力。
5. 调用 bt_manager_set_scan_mode，设置发现模式。
6. 监控客户端发起的连接，api 调用参考 cmd_bt_spp_service_accept。
7. 此时蓝牙协议栈都已经准备好，可被扫描被连接。
8. 连接成功的回调是 bt_test_spp_service_connection_state_cb。
9. 发送数据，api 调用参考 cmd_bt_spp_service_send。
10. 接收数据，通过回调 bt_test_spp_service_accept_cb。
11. 主动断开连接，api 调用参考 cmd_bt_spp_service_disconnect。
12. 关闭蓝牙，api 调用参考 cmd_bt_enable。

5.2.7 GATT Server 功能调用 API 实现

关于 GATT Server 功能的实现，已编写了使用示例，供用户参考，代码路径如下：

```
btmanager/demo/bt_test.c
btmanager/demo/bt_gatt_server_demo.c
```

打开 GATT Server 功能，和 client 端建立连接进行通讯，其整个过程 api 调用步骤：

1. 调用 bt_manager_preinit，预初始化，运行期间只需要调用一次。
2. 调用 bt_manager_enable_profile(BTMG_GATT_SERVER_ENABLE)，设置本次要使用的 profile。
3. 填充 btmg_callback_t 结构体，即回调函数。
4. 调用 bt_manager_init，蓝牙初始化。
5. 调用 system(“/etc/bluetooth/bt_init.sh ble_start”)，打开 BLE。
6. 调用 add_cmd_table，如果使用 cmd 命令需添加 gatt server 的命令列表。
7. 调用 bt_manager_gatt_server_init，初始化 gatt server，其功能主要是建立 L2CAP socket。
8. 构建一个 server，主要包括创建一个 service(bt_manager_gatt_server_create_service)，填充 service 中的内容特性内容和描述信息。可参考 bt_gatt_server_register_bt_test_svc。
9. 调用 bt_manager_le_set_adv_param，设置广播参数。可参考 set_adv_param。
10. 调用 bt_manager_le_set_adv_data，设置广播数据。可参考 le_set_adv_data。
11. 调用 bt_manager_le_set_scan_rsp_data，设置扫描回复数据。可参考 le_set_adv_data。
12. 调用 bt_manager_le_enable_adv，使能广播。
13. 手机 nrf 扫描建立连接后，调用 bt_manager_gatt_server_send_indication，对指定的 handle 发指示。
14. 调用 bt_manager_gatt_server_send_notify，对指定的 handle 发通知。与指示的区别在于通知不需对端应答。
15. 对方发出读请求、写请求，会通过 gatts_char_read_req_cb、gatts_char_write_req_cb 回调通知应用层。

5.2.8 GATT Client 功能调用 API 实现

关于 GATT Client 功能的实现，已编写了使用示例，供用户参考，代码路径如下：

```
btmanager/demo/bt_test.c
btmanager/demo/bt_gatt_client_demo.c
```

打开 GATT Client 功能，和 server 端建立连接进行通讯，其整个过程 api 调用步骤：

1. 调用 bt_manager_preinit，预初始化，运行期间只需要调用一次。
2. 调用 bt_manager_enable_profile(BTMG_GATT_CLIENT_ENABLE)，设置本次要使用的 profile。
3. 填充 btmg_callback_t 结构体，即回调函数。
4. 调用 bt_manager_init，蓝牙初始化。
5. 调用 system(“/etc/bluetooth/bt_init.sh ble_start”);，打开 BLE。
6. 调用 add_cmd_table，如果使用 cmd 命令需添加 gatt client 的命令列表。

7. 调用 bt_manager_gatt_client_init，初始化 gatt client ，其功能主要是建立 L2CAP socket，注册 EVENT 事件回调。
8. 调用 bt_manager_le_scan_start，开启扫描。可参考 cmd_ble_scan。
9. 调用 bt_manager_gatt_client_connect，发起连接。可参考 cmd_gatt_client_connect。
10. 连接成功后，调用 bt_manager_gatt_client_discover_all_services，发现对端服务列表。
11. 调用 bt_manager_gatt_client_read_request，对指定的 handle 读操作。
12. 调用 bt_manager_gatt_client_write_request，对指定的 handle 写请求。
13. 调用 bt_manager_gatt_client_write_command，对指定的 handle 写命令。与写请求的区别是无需对方应答。
14. 调用 bt_manager_gatt_client_register_notify_indicate，对指定的 handle 注册接收通知或指示。

6 bt_test 工具介绍

6.1 bt_test 简介

bt_test，是全志基于 btmanager 接口编写的蓝牙 demo，编译生成的可执行程序。其源码路径：

```
tina4.0：
tina/package/allwinner/wireless/btmanager4.0/demo/
tina5.0：
tina/platform/allwinner/wireless/btmanager/demo/
```

该程序可以在前台或者后台运行。建议开发者先熟悉一下这套工具的使用，再结合 demo 源码开发自己的应用。

6.2 使用说明

查看运行帮助：

```
# bt_test -h
Usage:
 [OPTION]...

Options:
 -h, --help          print this help and exit
 -v, --version       print btmanager version
 -d, --debug    open debug :-d [0~5]
 -s, --stop     stop bt_test
 -i, --interaction      interaction
 -p, --profile=NAME    enable BT profile
 -e, --extend   support extend advertise
 [supported profile name]:
 - a2dp-source Advanced Audio Source
 - a2dp-sink   Advanced Audio Sink
 - hfp-hf      Hands-Free
 - spp Serial Port Profile
 - gatt-server BLE server
 - gatt-client BLE client
```

参数说明：

- -h：查看帮助；
- -v：查看 btmanger 的当前版本；
- -d：打印等级，如果不指定，打印等级默认为 3(WARNG)，使用时加上打印等级，例如-d4；

- -s：关闭 bt_test，一般在后台模式使用；
- -i：表示是否运行在交互模式；
- -e：表示使用 BLE5 extend 拓展特性；
- -p：指定 profile，当前支持 a2dp-source/a2dp-sink/hfp-hf/spp/gatt-server/gatt-client。如果需要指定两个 profile，通过-p profile1 -p profile2 这种使用方法。

例子：运行 bt _test 在交互模式，同时指定 a2dp-sink 和 gatt-server 两个 profile，打印等级设置为 4(DEBUG):

```
bt_test -i -p a2dp-sink -p gatt-server -d4
```

默认 profile 说明

如果运行 bt_test 不加-p 指定 profile，默认选择 a2dp-sink 和 hfp-hf 这两个 profile。

可以通过修改 bluetooth.json 文件配置默认 profile。

6.2.1 后台模式

后台模式指的是运行 bt_test 程序时不加上-i 参数，例如：

```
bt_test -p a2dp-sink
```

后台模式运行之后没有程序控制台，为了方便调试与测试，我们支持在后台模式下通过写节点的方式发送控制命令，控制节点为/tmp/bt_io，例如：

```
echo get_adapter_name >/tmp/bt_io
```

6.2.2 交互模式

交互模式指指的是运行 bt_test 程序时加上了-i 参数，运行后会有控制终端，可以输入命令，例如：

```
bt_test -p a2dp-sink -i
```

在交互模式下同样可以通过控制节点/tmp/bt_io 进行控制。help 命令可以列出所有命令，如下：

```
[BT]:help
Available commands:
    enable                  enable [0/1]: open bt or not
    scan                    scan [0/1]: scan for devices
    scan_list               scan_list: list available devices
    pair                    pair [mac]: pair with devices
    unpair                  unpair [mac]: unpair with devices
    paired_list             paired_list: list paired devices
    get_adapter_state       get_adapter_state: get bt adapter state
    get_adapter_name        get_adapter_name: get bt adapter name
    set_adapter_name        set_adapter_name [name]: set bt adapter name
```

```
get_adapter_addr          get_adapter_addr: get bt adapter address
get_device_name           get_device_name[mac]: get remote device name
set_scan_mode             set_scan_mode [0~2]:0-NONE,1-page scan,2-inquiry scan&page scan
set_page_to               real timeout = slots * 0.625ms
set_io_cap                set_io_cap [0~4]:0-keyboarddisplay,1-displayonly,2-displayyesno,3-keyboardonly,4-
  noinputnooutput
avrcp                   avrcp [play/pause/stop/fastforward/rewind/forward/backward]: avrcp control
connect                 connect [mac]:generic method to connect
disconnect              disconnect [mac]:generic method to disconnect
remove                  remove [mac]:removes the remote device
a2dp_src_run            a2dp_src_run -p [folderpath]  or  a2dp_src_run -f [firepath]
a2dp_src_status          a2dp_src_status [status]:play pause forward backward
a2dp_src_stop           a2dp_src_stop:stop a2dp source playing
a2dp_set_vol            a2dp_set_vol: set a2dp device volme
a2dp_get_vol            a2dp_src_get_vol: get a2dp device volme
hfp_answer              hfp_answer: answer the phone
hfp_hangup              hfp_hangup: hangup the phone
hfp_dial                hfp_dial [num]: call to a phone number
hfp_cnum                hfp_cum: Subscriber Number Information
hfp_last_num            hfp_last_num: calling the last phone number dialed
hfp_vol                 hfp_vol [0~15]: update phone's volume.
sppc_connect            sppc_connect[mac]:connect to spp server
sppc_send               sppc_send xxx: send data
sppc_disconnect          sppc_disconnect[mac]:disconnect spp server
spps_accept             spps_accept the client
spps_send               spps_send data
spps_disconnect          spps_disconnect dst
get_version             get_version: get btmanager version
debug                   debug [0~5]: set debug level
ex_dbg                  ex_dbg [mask]: set ex debug mask
```

6.2.2.1 通用命令说明

API 接口	说明
help	打印目前模式下所有支持的命令和使用方法。
quit	退出 bt_test 程序。
get_version	获取 btmanager 版本信息。
debug	设置打印等级。取值范围【0-5】，对应 btmg_log_level_t。

6.2.2.2 经典蓝牙命令说明

API 接口	说明
enable	打开或关闭蓝牙。
scan	打开或关闭 BT 扫描。
scan_list	获取扫描设备缓存列表。
pair	配对指定 Mac 的设备。
unpair	取消与指定 Mac 设备的配对。

API 接口	说明
paired_list	获取已配对设备缓存列表。
get_adapter_state	获取 BT 状态。
get_adapter_name	获取本地 BT 名称。
set_adapter_name	设置本地 BT 名称。
get_adapter_addr	获取本地 BT Mac 地址。
get_device_name	根据 Mac 获取对端 BT 设备名称。
set_scan_mode	设置是否可发现、是否可连接状态。
set_page_to	设置 page timeout。单位：0.625ms。
set_io_cap	设置设备输入输出能力。
avrcp	AVRCP CT 发送音频控制命令。
avrcp_set_abs_volume	设置绝对音量。
avrcp_get_abs_volume	获取绝对音量。
connect	连接指定设备，连接的设备务必是扫描列表或者已配对列表里面的。
disconnect	断开与指定设备的连接。
remove	移除扫描缓存中的指定设备。
a2dp_src_run	初始化 a2dp_src 并开始播放音频，当前仅支持播放 wav 格式。
a2dp_src_set_status	用于控制播放、暂停以及文件夹播放时上一曲和下一曲操作。
a2dp_src_stop	停止 a2dp_src 播放。
a2dp_set_vol	设置 A2DP 设备播放的音量。设置作为 A2DP Sink 时，手机如果支持绝对音量，可以通过该命令调节手机的音量。设备作为 A2DP Source 时，通过该接口调节播放音量大小。
a2dp_get_vol	与 a2dp_set_vol 相反，获取 A2DP 设备的音量。
hfp_answer	接听电话。
hfp_hangup	拒接或挂断电话。
hfp_dial	指定号码拨号。
hfp_cnum	获取本机号码。
hfp_last_num	回拨最后一个通话。
hfp_vol	设置通话音量。
hfp_at_cmd	发送 at 指令。
hfp_ag_call	hfp ag 建立 sco 链路。
hfp_ag_hangup	hfp ag 断开 sco 链路。
sppc_connect	连接 spp server 设备，连接之前请确认设备已经使用 pair 命令配对过了。
sppc_send	发送数据给 spp server。
sppc_disconnect	断开与 spp server 设备连接。
spps_accept	启动 spp server 并开始监听。
spps_send	发送数据给 spp client。

API 接口	说明
spps_disconnect	断开与 spp client 设备连接。
ex_dbg	设置拓展调试标志位，用于打开特殊打印信息。

6.2.2.3 BLE-GATT-Server 命令说明

API 接口	说明
gatts_init	初始化 GATT server，注意运行 bt_test -i -p gatt-server 之后不需要单独执行 gatts_init 了，除非执行了 gatts_deinit 又需要重新初始化。
gatts_deinit	反初始化 GATT server。
ble_name	设置和获取 BLE 名字。
ble_advertise	打开或关闭 BLE 广播。
set_adv_int	调整广播间隔。
ble_disconnect_all	断开所有连接。
gatt_indicate	指定 handle 发出指示。
gatt_notify	指定 handle 发出通知。
set_phy	设置 BLE 使用的 PHY

6.2.2.4 BLE-GATT-Client 命令说明

API 接口	说明
gattc_init	初始化 GATT Client，注意运行 bt_test -i -p gatt-client 之后不需要单独执行 gattc_init 了，除非执行了 gattc_deinit 又需要重新初始化。
gattc_deinit	反初始化 GATT Client。
get_public_addr	获取当前蓝牙的公共地址，即 public 地址。
add_white_list	将指定 mac 地址添加到白名单。例如：add_white_list 48:45:20:FE:F0:B9 random；public 与 random 表示对端设备的 mac 地址类型。
ble_scan	打开或关闭 BLE 扫描。on 和 off 是必选项，例如 ble on；其他参数可选，参数含义如下：passive：被动扫描；int：扫描间隔；win：扫描窗口；wl：扫描过滤策略，对应 filter_policy 参数；dups/nodups：是否打开重复过滤；
ble_connect	GATT Client 设备连接 GATT Server 设备。
ble_disconnect	GATT Client 设备断开与 GATT Server 设备的连接。
ble_conn_update	更新连接参数。

API 接口	说明
ble_connections	获取当前连接列表，结果通过回调函数打印出来。
ble_select	在连接列表中选择要操作的设备。
gatt_write	写 Gatt Server 端特征值，Server 需要 response。handle：特征值 handle；offset：固定位为 0；data：数据内容，例如:123456aabb，如果后一个参数是 string，可以任意内容；[strings]：可选项，如果有 string，会把解析成字符串，否则是 16 进制数据。
gatt_write_cmd	写 Gatt Server 端特征值，Server 不需要 response。handle：特征值 handle；sign：是否需要添加认证签名；data：数据内容，例如:123456aabb，如果后一个参数是 string，可以任意内容；[strings]：可选项，如果有 string，会把解析成字符串，否则是 16 进制数据；repeat：重复写的次数。
gatt_read	读 Gatt Server 端特征值内容。
gatt_read_long	以数据方式读 Gatt Server 端特征值。
gatt_discover	发现当前连接的服务列表，通过回调函数打印出来。
gatt_discover_uuid	指定 uuid 发现服务。
get_mtu	获取当前 BLE 连接的 MTU 大小。
register_notify_indicate	注册 notify 或者 indicate，同时打开对方的 notify 或 indicate 开关，执行该命令之后，对端才能 notify 或者 indicate 给本机。
unregister_notify_indicate	注销 notify 或者 indicate。
test_write_rate	用来 gatt 两板对测 ble 传输速率，连续发送 gatt write command。

6.3 功能验证

6.3.1 A2DP Sink 测试

1. 指定 a2dp-sink 运行 bt_test：

```
bt_test -p a2dp-sink
```

或者进入交互模式（建议）：

```
bt_test -p a2dp-sink -i
```

2. 手机打开蓝牙，搜索 “aw-bt-test-xxxx” 的蓝牙设备，并发起连接；

3. 连上之后，手机打开音乐播放器 APP，播放音乐，设备端将同步输出声音。

6.3.2 AVRCP CT 测试

通过 avrcp play/pause/stop/fastforward/rewind/forward/backward 命令可进行音乐播放，暂停，快进，快退，上下曲等操作，例如下一曲：

```
avrcp forward
```

6.3.3 A2DP Souce 测试

1. 确认音频文件

A2DP Souce 测试需要播放音频，需要保证设备中有 wav 音频文件，否则请通过 adb push 进去，路径可以根据实际情况选择，一般建议/mnt 或者/tmp 路径，音频文件可以在 SDK 中获取，例如：

```
Tina/package/testtools/testdata/audio_wav/common/44100-stereo-s16_le-10s.wav
```

2. 指定 a2dp-source 运行 bt_test

建议运行为交互模式

```
bt_test -i -p a2dp-source
```

3. 扫描蓝牙音箱/耳机设备

如果需要连接的设备尚未配对（第一次连接或者已经取消配对），务必先扫描设备：

```
scan 1
```

通过观察打印或者通过命令查看是否扫描到：

```
scan_list
```

如果扫描到了，请停止扫描：

```
scan 0
```

4. 连接蓝牙音箱/耳机设备

连接的蓝牙设备务必是扫描缓存列表或者已配对列表中的，否则连接失败：

```
connect [mac]，例如：connect 40:EF:4C:7B:77:ED
```

连接成功会有如下打印：

```
BTMG[bt_test_a2dp_source_connection_state_cb:228]: A2DP source connected with device:....;
```

连接的过程中包含了配对的过程。

5. 开始播放

指定文件夹播放：

```
a2dp_src_run -p /mnt
```

或者指定文件播放：

```
a2dp_src_run -p /mnt/bt_test.wav
```

可以通过 a2dp_src_set_status [status] 进行播放控制，播放状态有 play、pause、forward、backward；

forward、backward 在指定文件夹播放时才有用。例如播放暂停：

```
a2dp_src_set_status pause
```

6. 音量大小的设置和获取可以通过如下命令：

```
a2dp_set_vol [0~100]
a2dp_get_vol
```

7. 结束播放

停止播放的线程，结束播放：

```
a2dp_src_stop
```

6.3.4 AVRCP TG 测试

连接上蓝牙音箱/耳机，点击蓝牙音箱/耳机的播放暂停、上下曲测试，设备端会收到对应的按键事件，并通过回调函数上报，同时会有如下打印：“BT palying music playing with device:xxx”。

6.3.5 SPP Client 测试

1. 指定 spp 运行 bt_test：

```
bt_test -i -p spp
```

2. 扫描设备

如果需要连接的设备尚未配对（第一次连接或者已经取消配对），务必先扫描设备：

```
scan 1
```

扫描设备之后停止扫描：

```
scan 0
```

3. 配对设备

如果设备尚未配对过或者已经取消配对，在连接之前务必先配对：

```
pair [mac]
```

4. 连接设备

连接已经配对的设备：

```
sppc_connect [mac]
```

5. 发送数据：

```
sppc_send xxxxx
```

6. 断开连接：

```
sppc_disconnect [mac]
```

6.3.6 SPP Server 测试

1. 指定 spp 运行 bt_test：

```
bt_test -i -p spp
```

2. 初始化 spp server 并开启线程监听 client 设备：

```
spps_accept
```

3. 连接成功后，发送数据：

```
spps_send xxxx
```

4. 断开连接

主动断开对端设备：

```
spps_disconnect [mac]
```

6.3.7 HFP HF 测试

1. 指定 hfp-hf 运行 bt_test：

```
bt_test -i -p hfp-hf
```

2. 手机 A 搜索连接上样机，名字为 “aw-bt-test-xxxx”
3. 手机 B 播打手机 A，手机 A 接听电话：

```
hfp_answer
```

4. 手机 B 播打手机 A，手机 A 拒接电话：

```
hfp_hangup
```

5. 样机拨打电话：

```
hfp_dial 10086
```

6. 回拨最后一个通话：

```
hfp_last_num
```

7. 获取手机号码：

```
hfp_cnum
```

8. 设置通话的音量：

```
hfp_vol [0~15]
```

6.3.8 HFP AG 测试

1. 指定 hfp-ag 运行 bt_test：

```
bt_test -i -p hfp-ag
```

2. 扫描 hfp-hf 设备：

```
scan 1
scan 0
```

3. 发起连接：

```
connect xxx(mac地址)
```

4. 创建 sco 链路：

```
hfp_ag_call xxx(mac地址)
```

此时会打开已配置好的声卡，能进行对话。

6.3.9 GATT Server 测试

1. 指定 gatt-server 运行 bt_test，使用 BLE5 的模组需要加-e：

```
bt_test -i -p gatt-server
```

2. 手机搜索连接

手机需要安装 “nrf connect” APP，打开 APP 后扫描搜索到 “aw-ble-test-007” 名称的 BLE 设备，并连接上；

手机 APP 可以发现样机上的服务信息，目前 demo 注册了两个 Service：

- test_svc：UUID 为 “1112”；
- uart_svc: UUID 为 “6e400001-b5a3-f393-e0a9-e50e24dcca9e”。

3. 读写数据

对 uuid 为 “3344” 的 characteristic 分别进行 read 和 write 操作：

- read 操作时手机 APP 会收到数值的累计增加；
- write 操作时手机 APP 发送的字符会打印在样机的终端上。

4. notify 通知测试

在 nrf connect 里面设置 CCC 为通知，点击选中【三个下】箭头，然后样机端输入命令通知手机端：

```
gatt_notify [char_handle] [value]
```

5. indicate 指示测试

在 nrf connect 里面设置 CCC 为指示，点击选中【上下】箭头，然后样机端输入命令指示手机端：

```
gatt_indicate [char_handle] [value]
```

6.3.10 GATT Client 测试

1. 指定 gatt-client 运行 bt_test，使用 BLE5 的模组需要加-e：

```
bt_test -i -p gatt-client
```

2. 手机创建 server 端

手机需要安装 “nrf connect” APP，打开 APP 后创建一个 device, 添加 server，并打开广播。

3. 板子搜索连接：

```
ble_scan 1
```

4. 扫描到设备后停止扫描：

```
ble_scan 0
```

5. 连接设备：

```
ble_connect [mac]
```

6. 发现所有服务：

```
gatt_discover
```

7. read 测试

参数 handle 根据发现服务列表中获取：

```
gatt_read <handle>
```

读取到的值会显示在样机终端上。暂时不支持需配对才能读取的 handle。

7 常见问题和调试指南

蓝牙开发使用中遇到问题请参考《Tina_Linux_ 蓝牙 _ 常见问题与调试指南》文档或者参考全志客户服务平台的常见问题。

著作权声明

商标声明

免责声明