文档编号：AN1055

上海东软载波微电子有限公司

应用笔记

ES32W3120 SDK User Guide

修订历史

版本	修订日期	修改概要
V1.0	2021-6-8	初版发布
V1.1	2022-1-6	1.增加绑定设备密钥管理 2.增加 iDesigner IDE 工程设置说明 3.增加 RTT Studio IDE 工程设置说明 4.增加低功耗管理、MAC 地址管理及 OTA 使用指南； 5. 更新公司地址。
V1.2	2022-7-20	1. 增加 LR PHY 配置； 2. 修正低功耗配置说明。
V1.3	2022-9-20	1.增加应用例程章节

地　址：中国上海市徐汇区古美路 1515 号凤凰园 12 号楼 3 楼
E-mail：support@essemi.com
电　话：+86-21-60910333
传　真：+86-21-60914991
网　址：http://www.essemi.com/

目 录

内容目录

图目录

表目录

1 前言

ES32W3120 SDK 是一款软件开发包，集成了 BLE5.0 协议栈、ES32W3120 芯片的外设驱动以及实时操作系统。本文档是 ES32W3120 SDK 的应用开发指南，可帮助用户使用 SDK 快速建立和开发工程项目。

2 概述

2.1 SDK架构

ES32W3120 SDK 架构如图 2-1 所示：

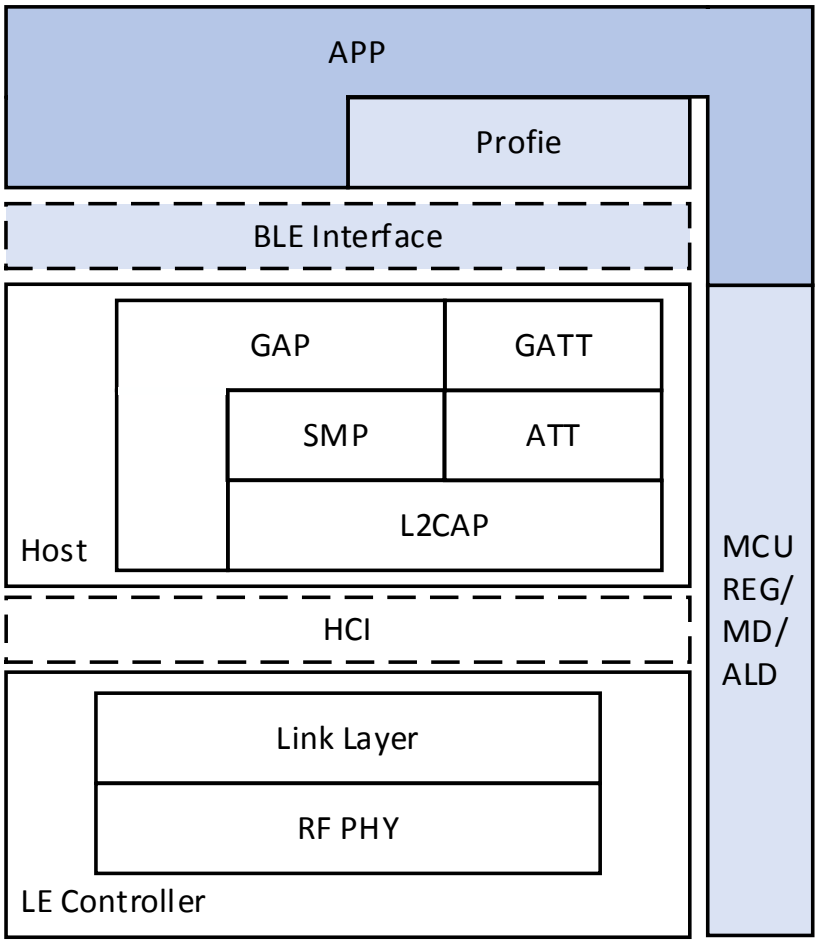


图 2-1 SDK 架构

ES32W3120 包含 BLE 5.0 完整 BLE 协议栈、Profile、应用示例。其中，Host、LEController、HCI 等协议栈部分以固件库方式提供，用户可使用提供的 BLE Interface 访问 BLE 协议栈；剩余部分以源码形式开放至用户，用户可按具体产品开发需求修改、编写应用程序，自定义 Profile。

2.2 空间分配

ES32W3120 协议栈空间分配如图 2-2 所示：

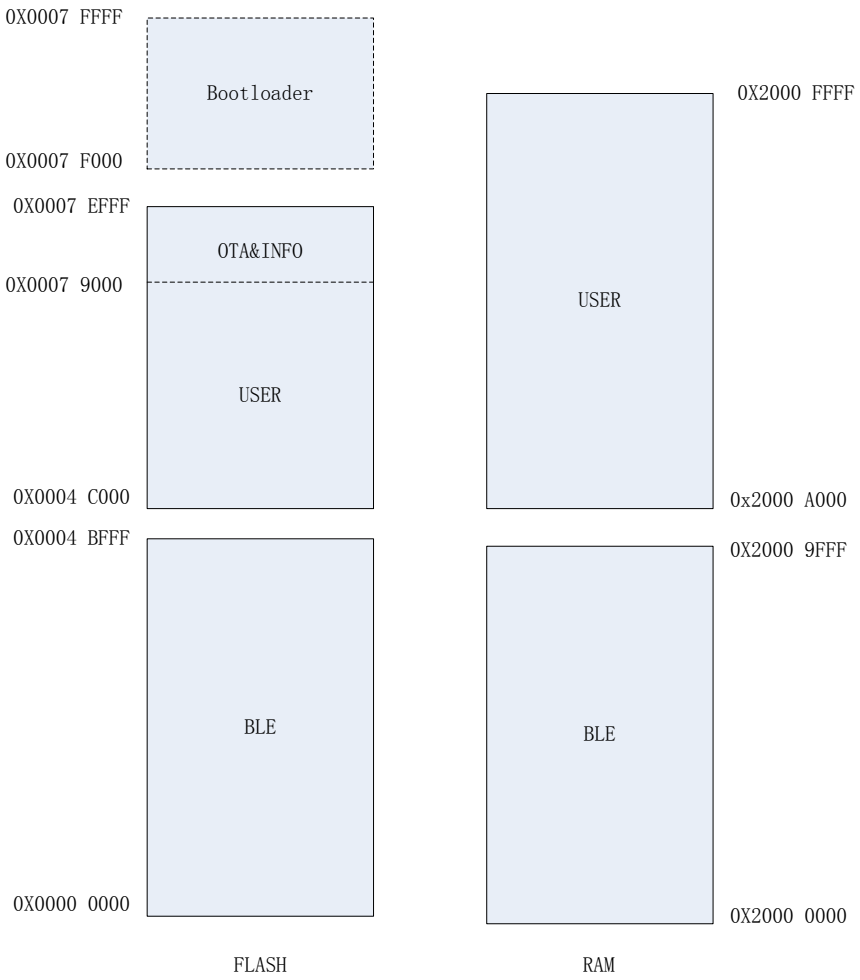


图 2-2 SDK 空间分配

BLE 协议栈代码与存储数据固定在 Flash 0x00000000 ~ 0x0004BFFF 位置，用户代码起始地址必须固定在 0x0004C000，用户通过分散加载文件配置用户固件空间，如图 2-3 所示。

```
;*************************************************************
;*** Scatter-Loading Description File generated by uVision ***
;*************************************************************

LR_IROM1 0x0004C000 0x0002D000  {    ; load region size_region
  ER_IROM1 0x0004C000 0x0002D000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x20009D80 0x0000A000  {  ; RW data
   .ANY (+RW +ZI)
  }
}
```

图 2-3 用户空间配置

启动代码与默认链接配置会将用户中断向量表起始地址放置在 0x0004C000。用户

可选择是否使用 Bootloader，Bootloader 具体配置请参考 ES32W3120 芯片的参考手册《ES32W3120_Reference_Manual》。

BLE 协议栈与用户代码共同使用 64K 的 SRAM，BLE 协议栈默认配置使用 SRAM 地址的 0x2000000~0x20009FFF 段，剩余 RAM 供用户使用。用户初始化 BLE 协议栈时，BLE 协议栈将对 RAM 空间进行检查，初始化用户 Stack 并将 SP 指针指向用户 Stack，同时通过用户选择的输出方式向用户提示当前用户配置使用的 RAM 起始地址是否合适，用户可根据提示信息修改 RAM 起始地址配置。其中用户 Stack 默认配置在启动代码中定义。用户通过工程 IRAM 配置用户 RAM 空间，如图 2-3 所示。

2.3 启动配置流程

无论用户是否选择通过 Bootloader 启动，芯片在运行用户代码前，都将首先运行 BLE 固件，完成基本时钟、RF 等配置后，再跳转至用户代码运行。用户代码中调用 ble_init 接口后，底层 BLE 固件将重新初始化用户 Stack 并修改 SP 指针。因此，用户应在代码中尽早调用 ble_init 接口。启动流程如图 2-4 所示：

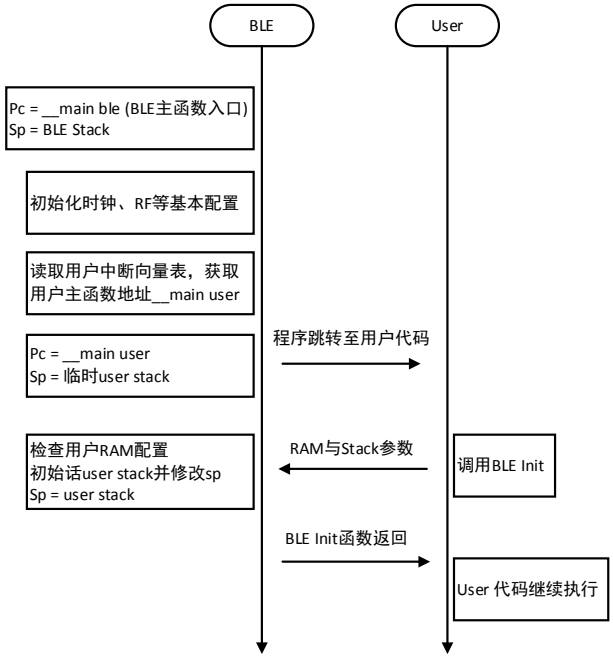


图 2-4 代码启动流程

目前，用户完成启动并执行成功 ble_init 后，在使用其他 BLE 接口前应首先完成协议栈配置。配置流程如图 2-5 所示：

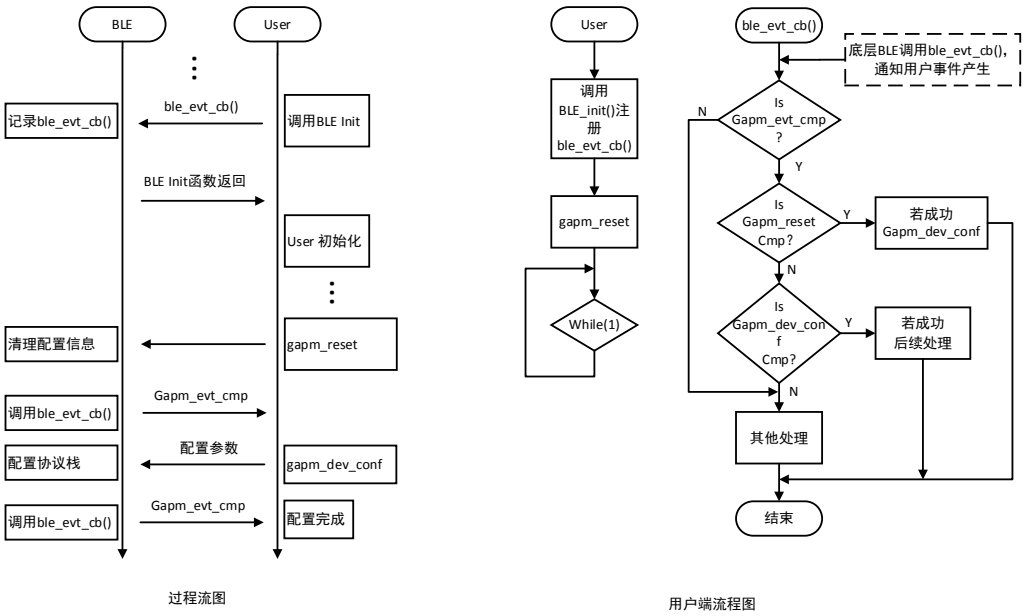


图 2-5 BLE 协议栈配置流程

2.4 用户工程编程说明

2.4.1 目录说明

SDK 包文件目录如表 2-1 所示：

目录	说明
➥Drivers	驱动
➥ALD	抽象层驱动
➥CMSIS	微控制器软件接口标准
➥MD	微驱动
➥Middlewares	中间层
➥EastSoft	东软载波提供的中间层
➥BLE5.0	BLE 5.0 相关中间层
➥Firmware	BLE 协议栈 HEX 文件
➥Interface	BLE 5.0 基本接口文件
➥Lib	进一步封装的 BLE5.0 接口文件
➥Log	ES32W3120 调试输出
➥Profile	BLE 5.0 Profile 示例

➥Projects	工程示例
➥ES32W3120	ES32W3120 相关示例工程
➥Application	应用示例
➥Examples_BLE_peripheral	BLE 外设示例
➥uart_boot	Bootloader 示例
➥Example_ALD	抽象层驱动示例
➥Example_MD	微驱动示例

表 2-1 SDK 包目录说明

2.4.2 BLE应用实例

以 LED 外设为例，工程路径：

\Projects\ES32W3120\Applications\Examples_BLE_peripheral\LED。其中，MDK-ARM 为工程配置；Src 为应用相关代码；Inc 为应用相关配置文件。

例程中，数据库定义如表 2-2 所示：

服务/特性/特性描述符	UUID	权限
LED 服务(自定义)	{0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, 0x11, 0x012 ,0x13, 0x14, 0x15, 0x16}(自定义)	可读
LED 特性(自定义)	{0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7, 0xA8, 0xA9, 0xA0, 0xA1, 0x0A2 ,0xA3, 0xA4, 0xA5, 0xA6}(自定义)	可读、可写
Button 特性(自定义)	{0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7, 0xB8, 0xB9, 0xB0, 0xB1, 0x0B2 ,0xB3, 0xB4, 0xB5, 0xB6}(自定义)	可通知
ButtonCCC 特性描述符	0x2902(标准)	可读、可写

表 2-2 LED 数据库说明

1. 下载协议栈 HEX 文件，文件路径为：

\Middlewares\EastSoft\BLE5.0\Firmware

可使用我司的 ESLINKII 下载，也可使用 JLINK 直接下载 HEX。

用我司 ESLINKII 下载，下载步骤如下：

a) 将 ESLinkII 连接开发板后插入 PC 端，打开 ESBurner-ESLinkII，如图 2-6 所示：

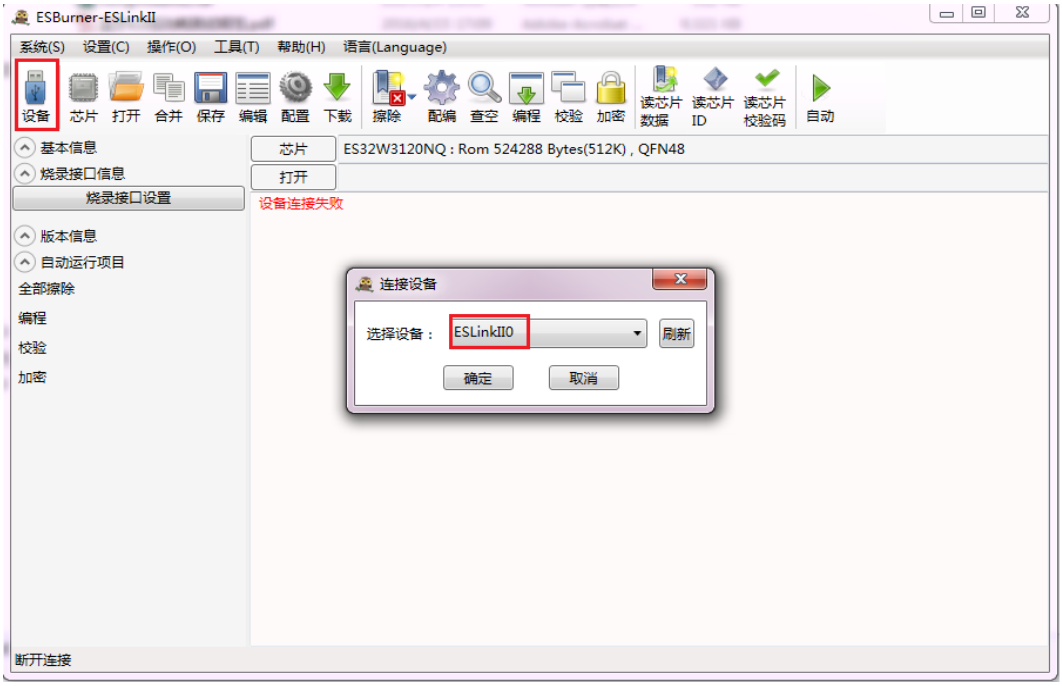


图 2-6 ESBurner-ESLinkII 界面

b) 选择使用 ES32W3120 芯片，如图 2-7 所示：

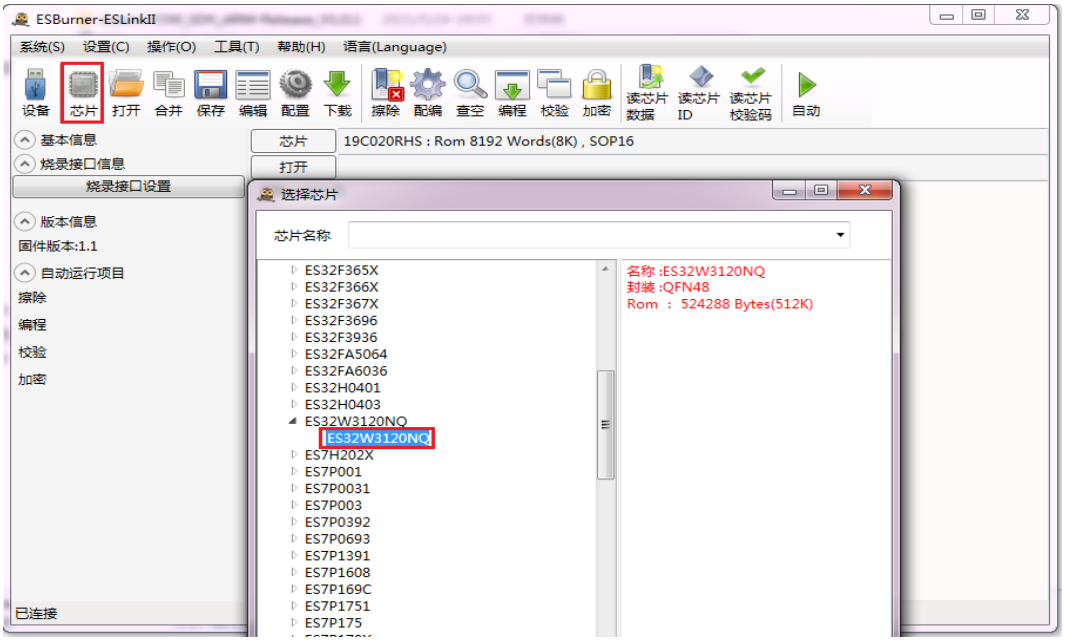


图 2-7 选择芯片型号

c) 打开需要下载的协议栈 HEX 文件，如图 2-8、图 2-9 所示：

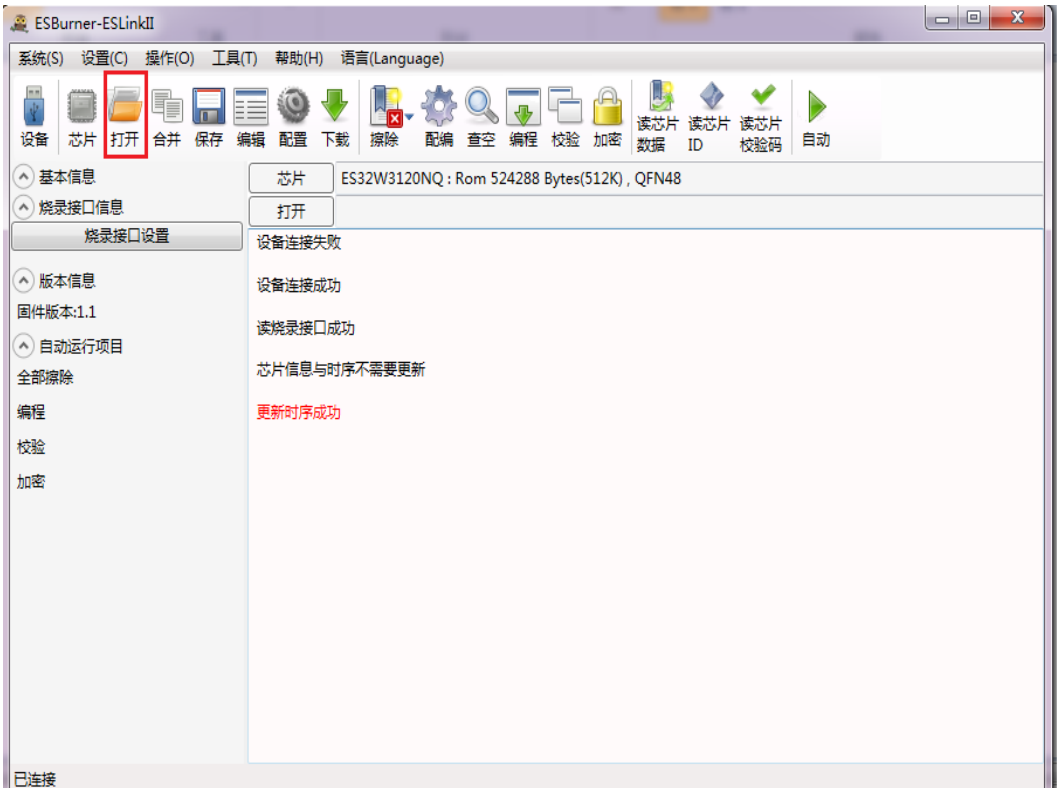


图 2-8 打开 HEX 文件

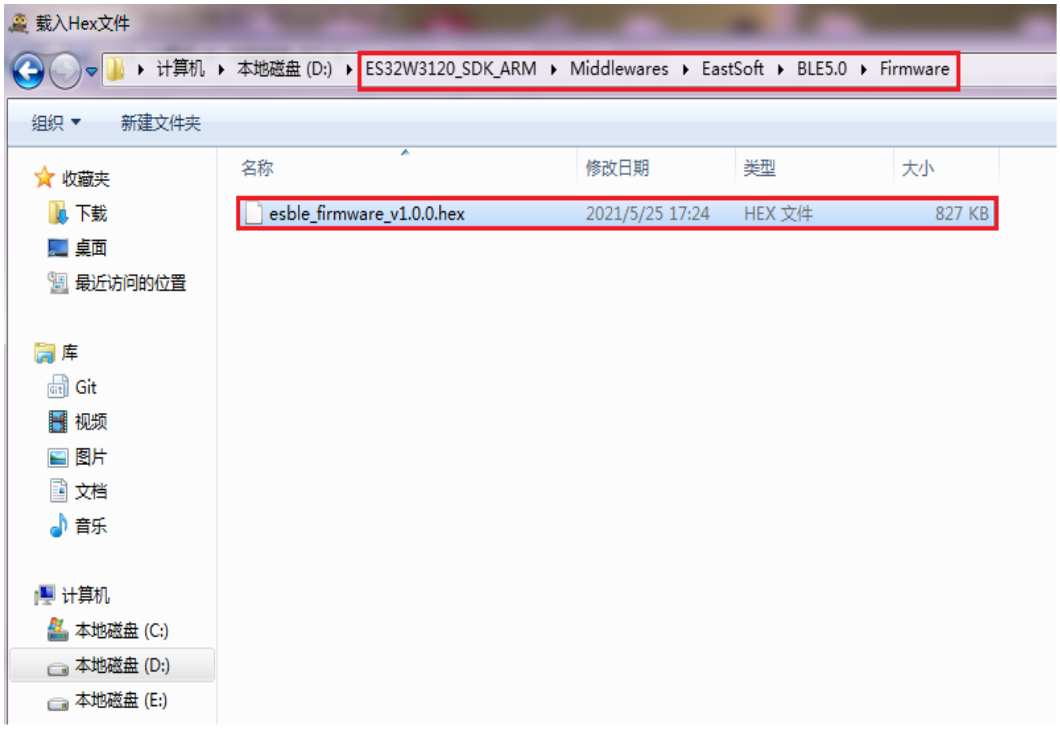


图 2-9 HEX 文件路径

d) 确认相关配置，如图 2-10、图 2-11、图 2-12 所示：

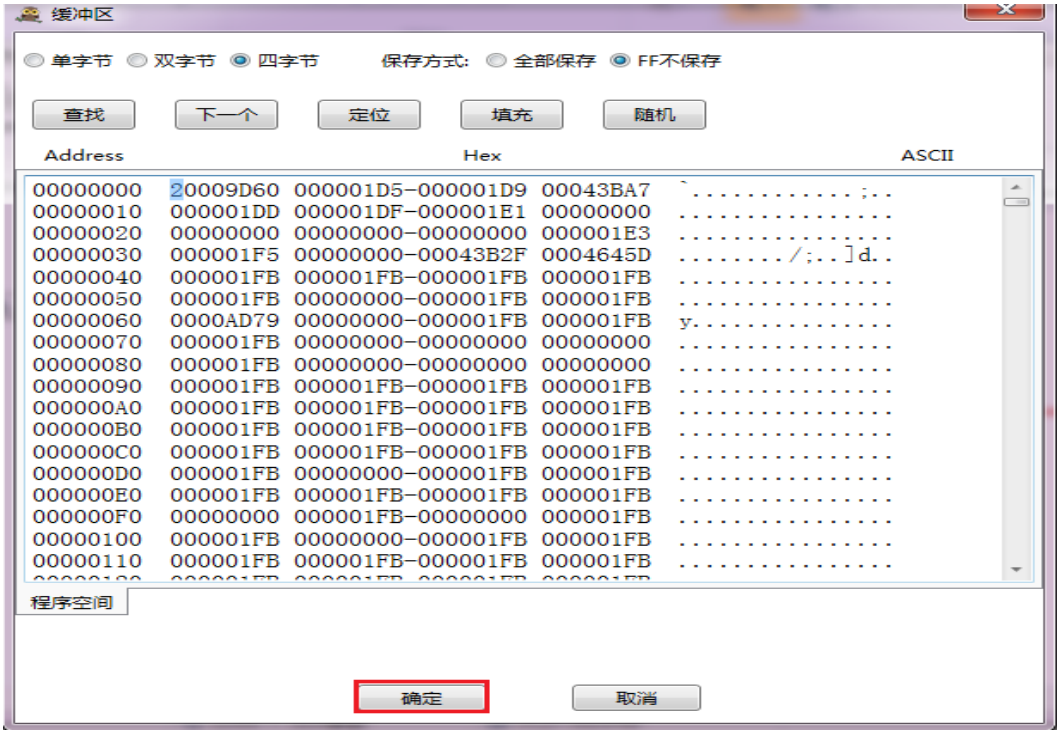


图 2-10 确认缓冲区

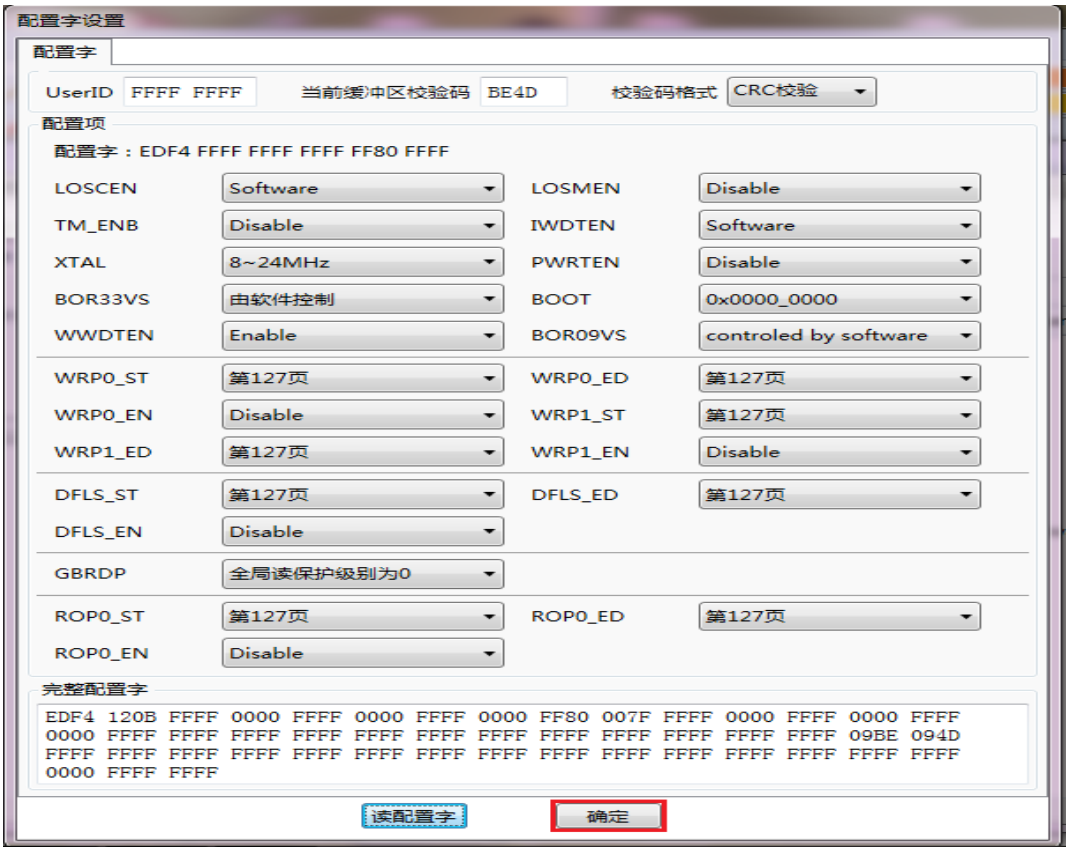


图 2-11 确认配置字

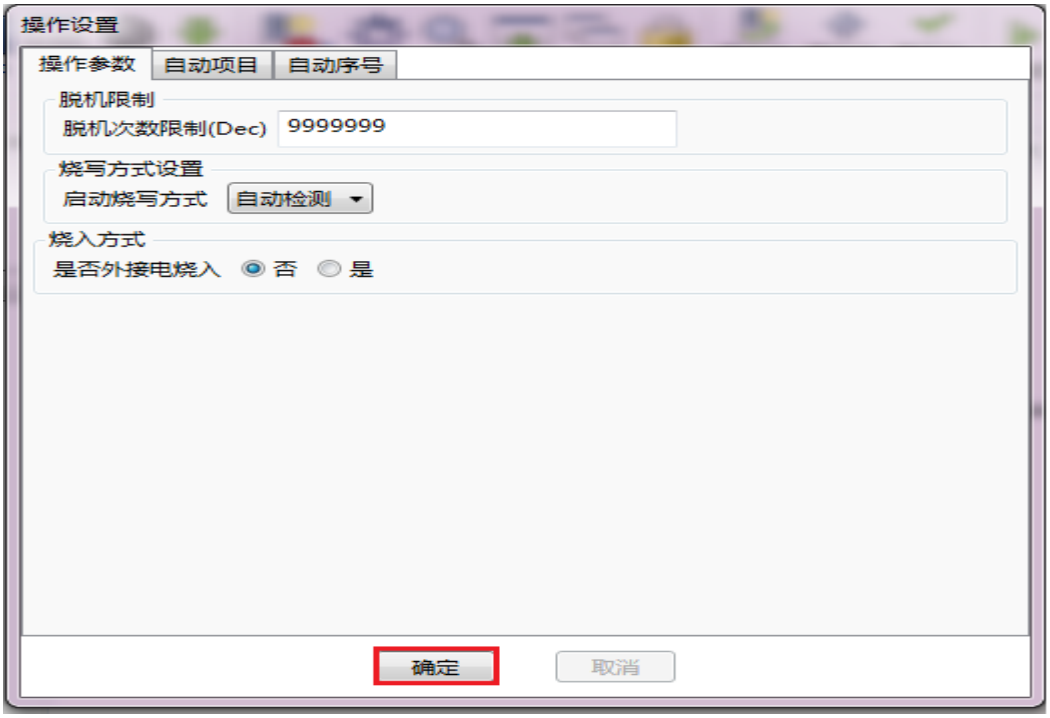


图 2-12 确认操作设置

e) 下载程序及配置字到编程器，如图 2-13 所示：

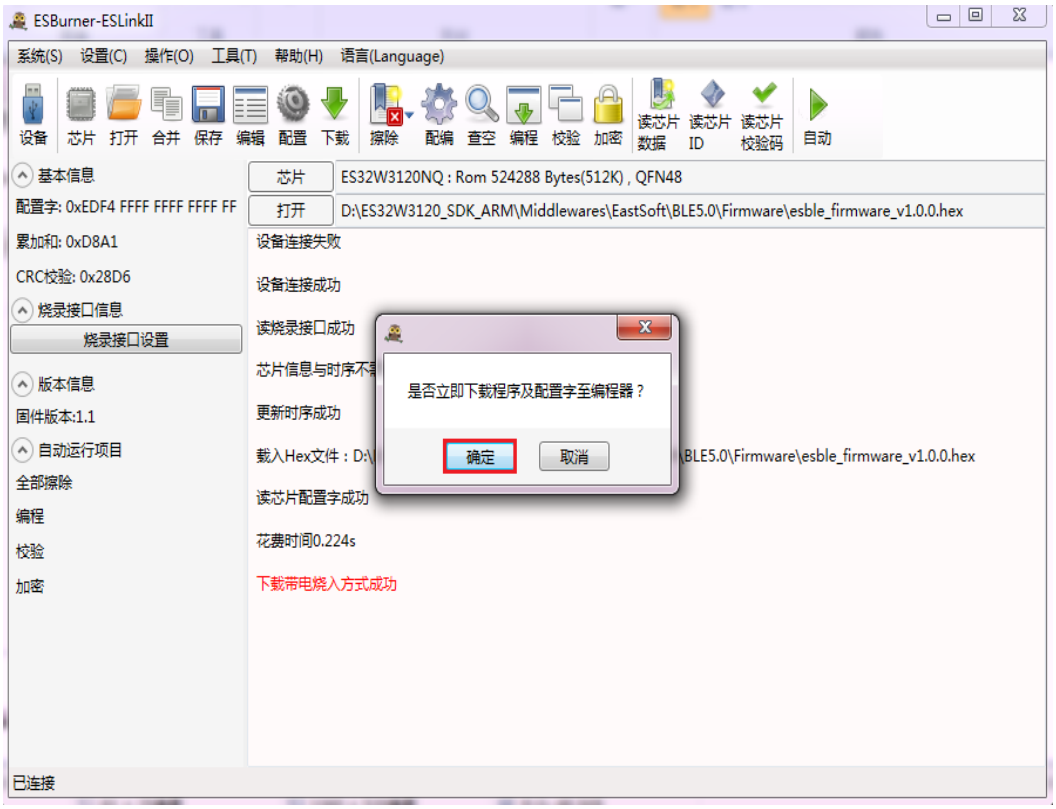


图 2-13 下载程序及配置字至编程器

f) 全部擦除及编程，如图 2-14、图 2-15 所示：

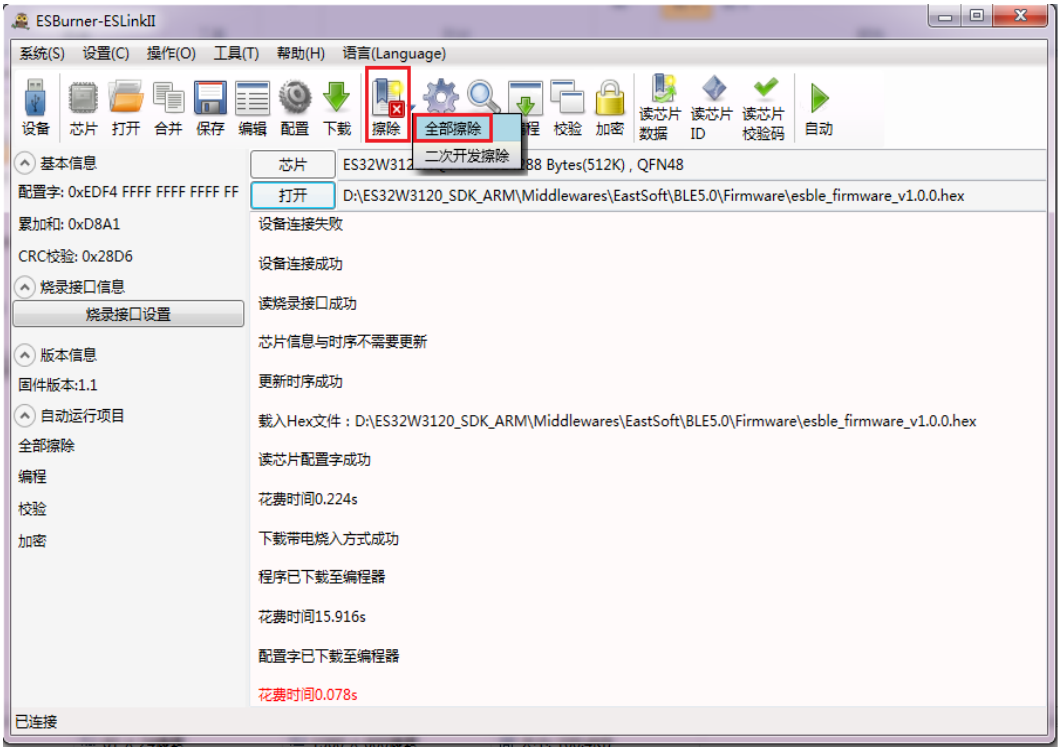


图 2-14 全部擦除

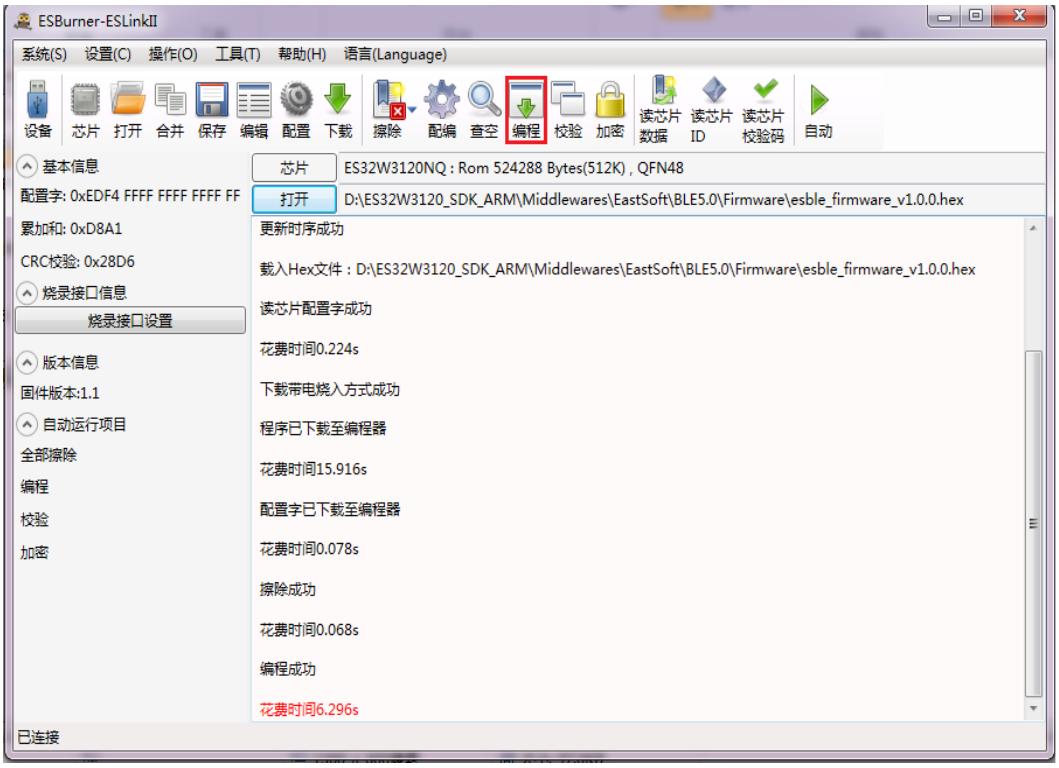


图 2-15 编程

使用 JLINK 下载 HEX 文件的步骤如下：

a) 打开 LED_Server 工程文件，并选择 ble_stack 工程如图 2-16 所示：

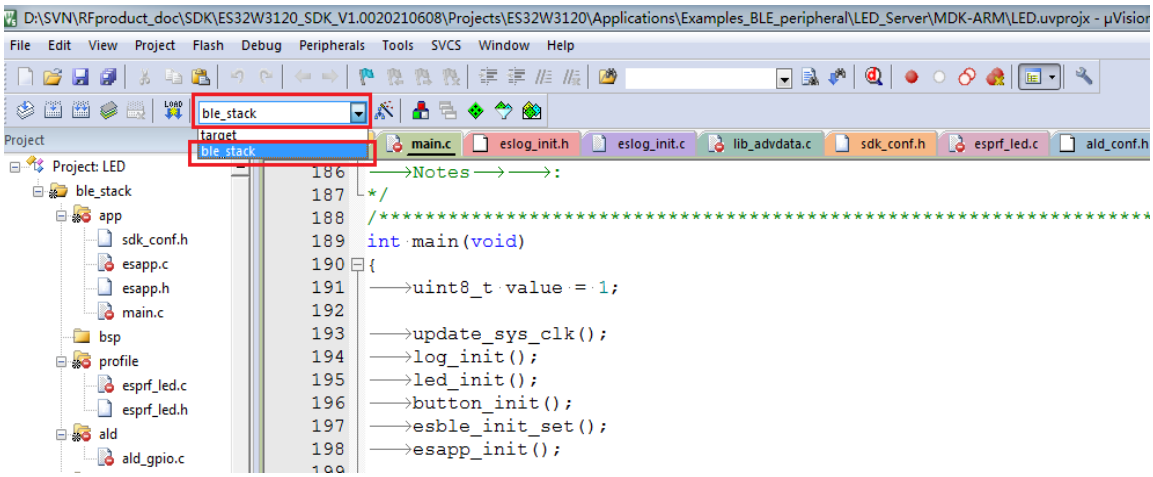


图 2-16 选择 ble_stack 工程

b) 点击下载，如图 2-17 所示：

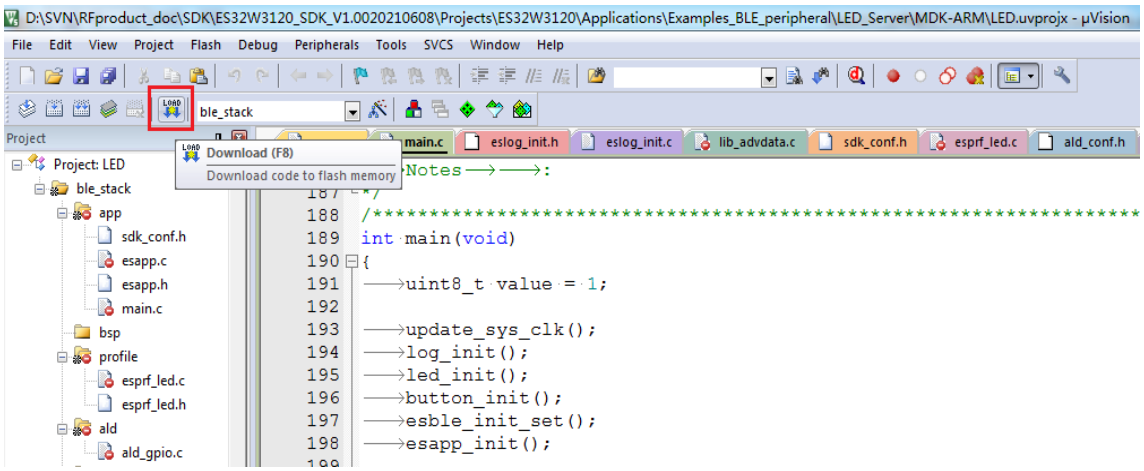


图 2-17 下载

2. 安装 ES32W3120 的 Keil 芯片支持包，可在我司官网中“其他 PC 软件"->"Keil5 芯片支持包"中下载使用。
3. 打开 LED 工程文件，路径为：
\Projects\ES32W3120\Applications\Examples_BLE_peripheral\LED_Server\MDK-ARM
4. 在 sdk_conf.h 中配置调试输出接口，SDK 提供三种调试输出接口：SEGGER RTT、UART 和 ESDAPViewer。可通过上述头文件中 BLE_LOG_RTT、BLE_LOG_UART、BLE_LOG_ESDAP 宏定义来配置接口，需注意三者只能选其一，若三个宏定义同时设置为 1 则默认使用 RTT 接口输出调试信息。若使用 UART 接口输出调试信息，用户

可设置串口相关配置，默认配置见图 2-18 所示：

```
static void uart_init_struct(md_uart_init_t *init)
{
    init->baud          = 115200;
    init->word_length   = MD_UART_WORD_LENGTH_8B;
    init->stop_bits     = MD_UART_STOP_BITS_1;
    init->parity        = MD_UART_PARITY_NONE;
    init->fctl          = MD_UART_FLOW_CTL_DISABLE;
    init->mode          = MD_UART_MODE;

    return;
}
```

图 2-18 串口默认配置

开发板默认使用串口 2 进行信息输出。

5. 编译并下载到开发板中，根据输出的调试信息确认配置的 RAM 起始地址是否合适，若需重新配置 RAM 起始地址，参考图 2-3 所示重新配置 RAM 参数，重复步骤 4。

使用 SEGGER RTT 查看调试输出的步骤如下：

a) 从开始菜单打开 SEGGER->J-Link RTT Viewer，设置配置，如图 2-19 所示：

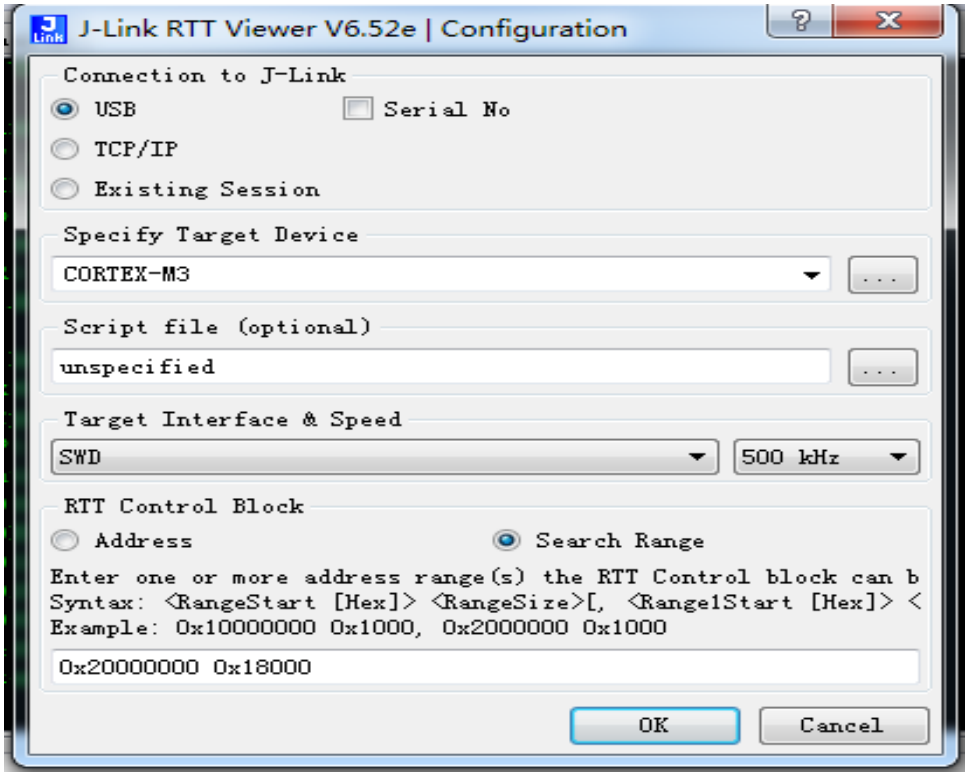


图 2-19 RTT Viewer 配置

b) 点击配置窗口的 OK 键后，即可在显示界面中看到输出的 LOG 信息，如图 2-20

所示：

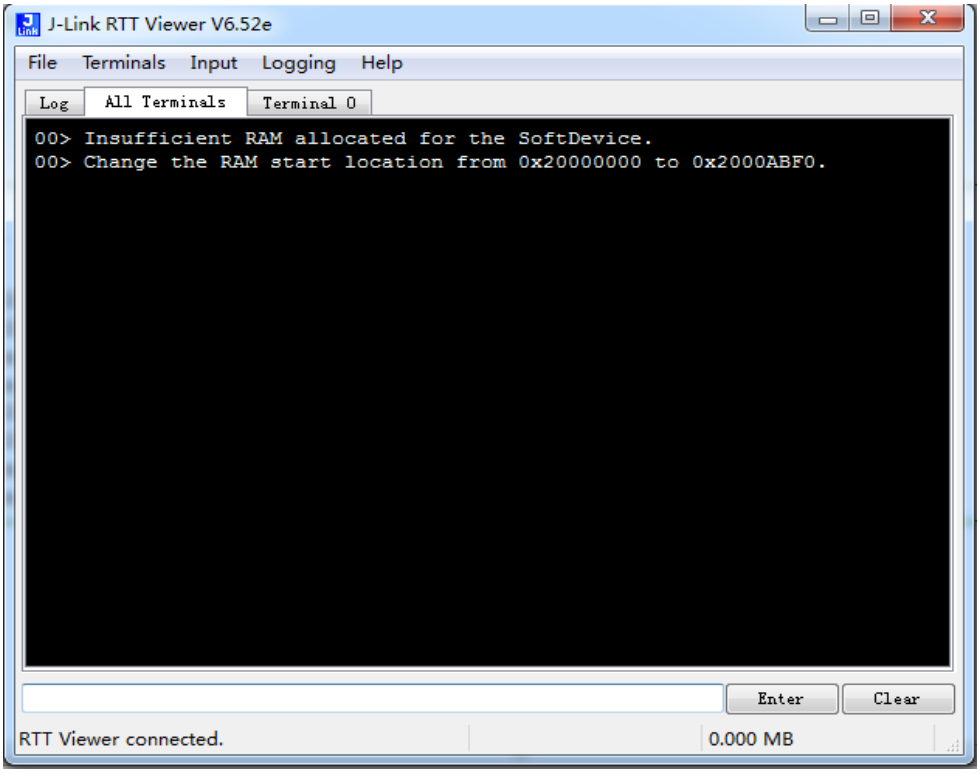


图 2-20 输出 LOG 信息

使用 UART 查看调试输出的步骤如下：

a) 使用 USB 将开发板连接到电脑上

b) 打开串口调试助手，根据工程中串口的配置信息设置配置：

图 2-21 串口默认配置

c) 连接后，即可在显示界面看到输出的 LOG 信息

共有三种 LOG 信息：

图 2-22 所示为用户配置的 SRAM 起始地址与建议的起始地址一致，此时用户无需更改 SRAM 起始地址配置。

图 2-23 所示表示用户设置的 SRAM 起始地址与 BLE 协议栈所使用的 SRAM 发生了冲突，用户必须重新更改 SRAM 起始地址。

图 2-24 所示表示用户设置的 SRAM 起始地址可以通过更改为推荐的起始地址，更改后可增大用户可使用的 SRAM 空间大小，若用户对 SRAM 使用空间评估可行，也可忽略 LOG 信息，不调整 SRAM 起始地址。

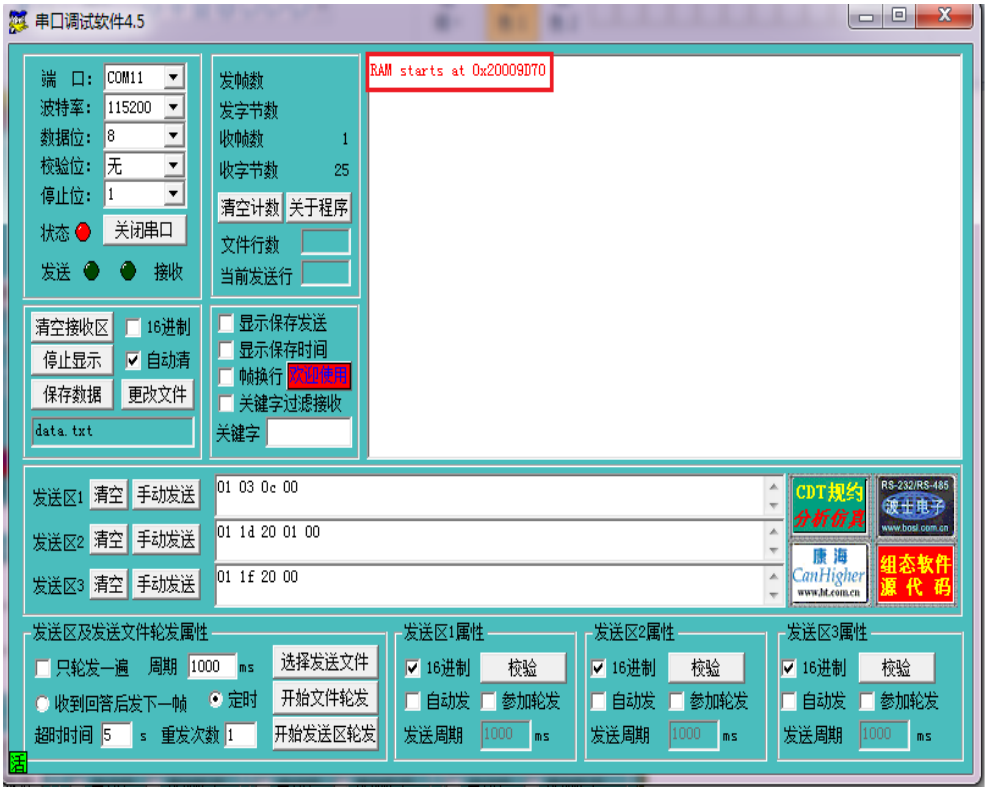


图 2-22 配置的 SRAM 起始地址与建议值一致

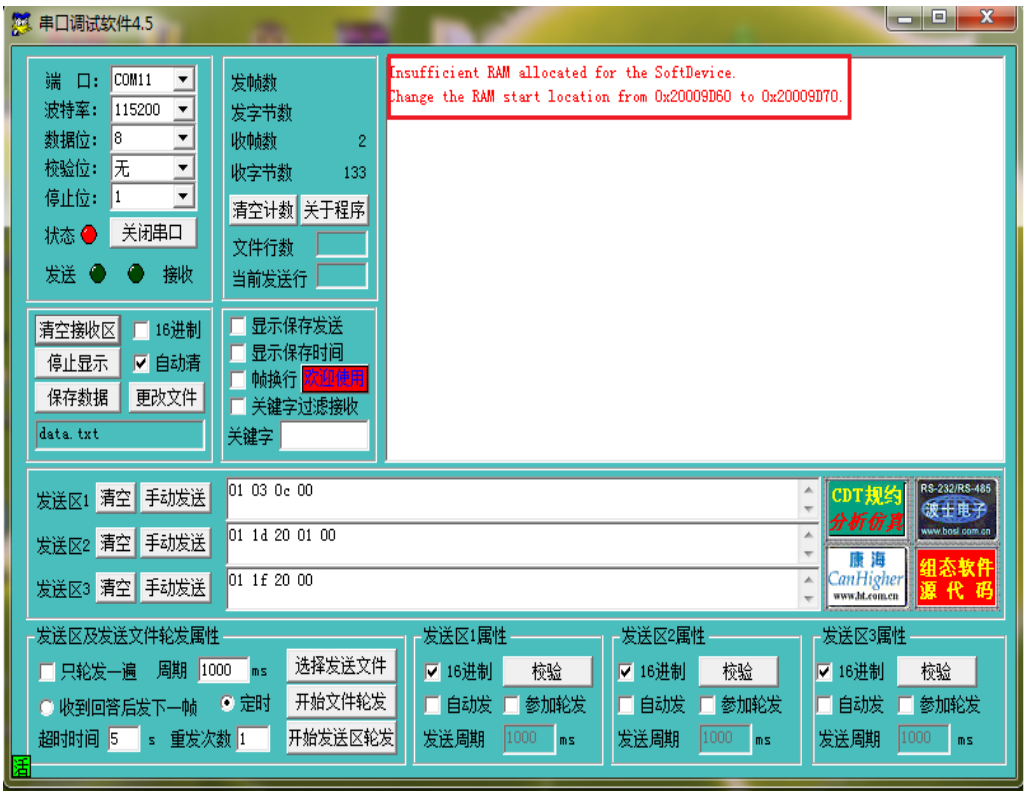


图 2-23 SRAM 起始地址必须重新设置

图 2-24 SRAM 起始地址建议重新设置

6. 复位开发板

7. 打开手机蓝牙，连接开发板，可对开发板进行控制：对应开发板，Button1 按下 LED1

亮，按键特性上报 0x1；Button1 松开 LED1 灭，按键特性上报 0x0；LED 特性写入 0x1，LED3 亮；LED 特性写入 0x0，LED3 灭；LED2 指示广播状态；LED4 指示连接状态。

2.4.3 低功耗管理

当设备在无事件处理时可进入低功耗模式，用户可在 sdk_config.h 头文件中配置 sleep mode，可选三种模式：ESBLE_SLEEP_NONE（不休眠）、ESBLE_SLEEP_STOP1（休眠进入 STOP1 模式）、ESBLE_SLEEP_STOP2（休眠进入 STOP2 模式）。在使用 STOP1 模式时，唤醒后无需对 RF、MODEM 和外设进行重新初始化，在使用 STOP2 模式时，唤醒后需要对 RF、MODEM 和外设重新进行初始化，GPIO 口仅 PA2~PA9 可保持，其余 GPIO 口会掉电。

note:当休眠时间小于 100ms 时为达到更好的低功耗效率建议使用 stop1 模式。

低功耗流程图如下图所示：

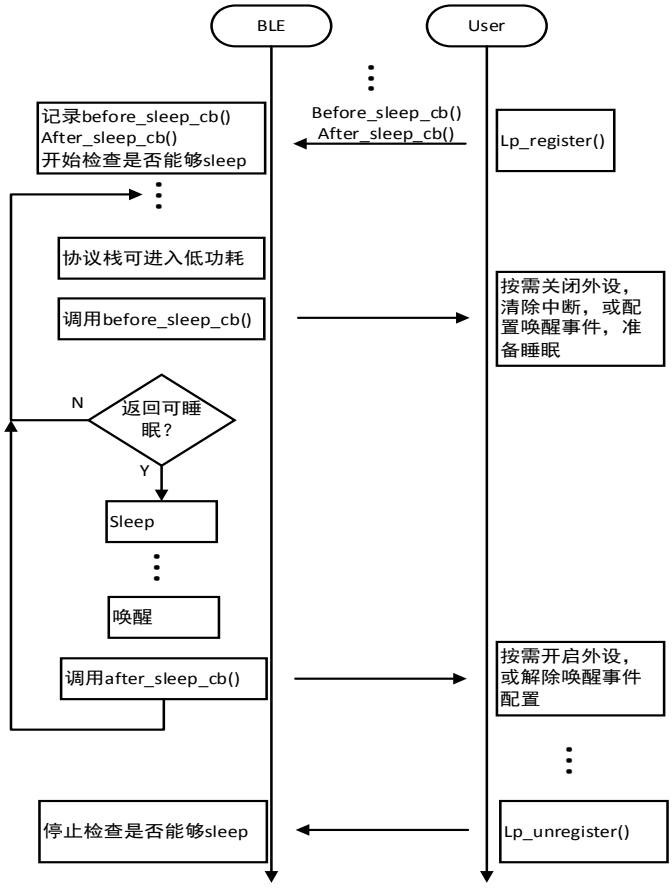


图 2-25 BLE 低功耗管理流程

note：使用低功耗功能时需要外接 32.768KHZ 的晶振

2.4.4 LR PHY配置

当使用 BLE5.0 的 LR 功能时，用户可在 sdk_config.h 头文件中配置所使用的 CODED PHY，

BLE_CODED_PHY_500_EN 配置为 1，CODED PHY 使用 500K，BLE_CODED_PHY_500_EN 配置为 0 时，CODED PHY 使用 125K。

2. 4. 5 MAC地址管理

设置设备 MAC 地址有三种方式：

1. 指定地址写入
2. sdk_conf.h 头文件配置
3. 默认设置

以上三种设置方式的优先级为 2>1>3，即若 sdk_conf.h 头文件中指定了有效地址，且在指定地址处也烧录了地址信息，则优先使用头文件中配置值。

设备的 MAC 地址可在量产烧录时使用 ESLINKII 的滚码烧录功能在指定地址：0X7E008 处烧录指定基地址的方式来保证量产芯片 MAC 地址的唯一性，在调试阶段可使用 sdk_conf.h 头文件中的#define BLE_BD_ADDR {{0x01,0x02,0x03,0x04,0x05,0x06}}来设置设备的 MAC 地址，若指定地址 0X7E008 及 sdk_conf.h 头文件中 BLE_BD_ADDR 均未设置有效值则设备使用默认 MAC 地址 {{0x01, 0x23, 0x45, 0x67, 0x89, 0xAB}}。

2. 4. 6 绑定设备密钥管理

若设备需要绑定操作，可参考工程：
\ESW_SDK_ES32W3120_V2.0.1\Projects\ES32W3120\Applications\Examples_BLE_peripheral\LED_Server_Bonding

需注意以下设置：

1. 设置配对模式

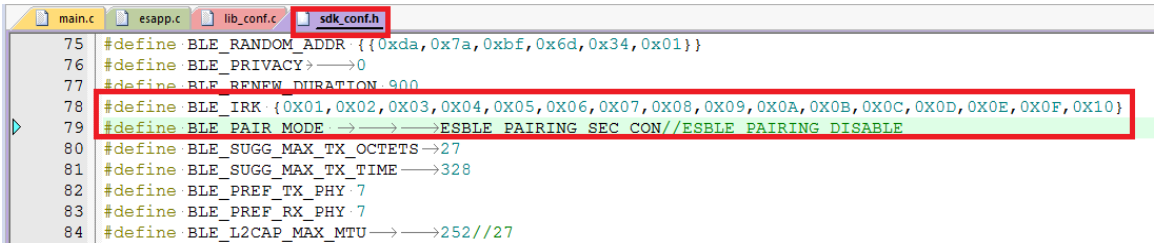


图 2-26 设置配对模式

2. 绑定时设备双方中间值确认

在两个设备执行绑定流程中，双方设备会生成中间密钥，并通过 UART 调试接口显示出来，中间密钥值为 6 位十进制数字，双方设备需要确认显示的中间密钥值是否一致，若一致则先按 server 端设备的 K1 按键确认，然后按 client 端设备的 K1 按键确认。双方设备确认中间密钥后会自动生成 LTK 并保存在指定信息区用于后续的重新连接时自动加密。

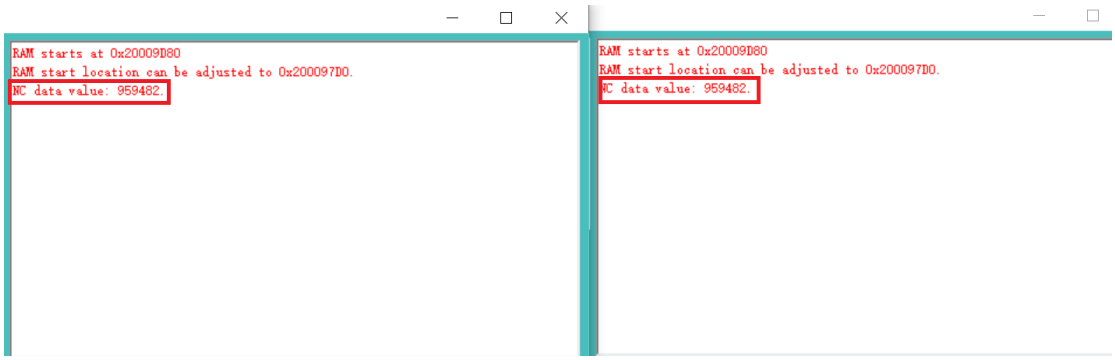


图 2-27 中间密钥显示

3. 绑定设备存储的密钥丢失时的处理

若之前已绑定的设备中的一方丢失了存储的密钥时，可分两种情况：

- Client 端丢失密钥

  Client 端会自动重新发起绑定流程。

- Server 端丢失密钥

  Client 端会在串口调试信息中显示“Peer LTK missing”信息，此时 Client 端会自动删除原存储的密钥，下次重新连接时 Client 端会重新发起绑定流程。

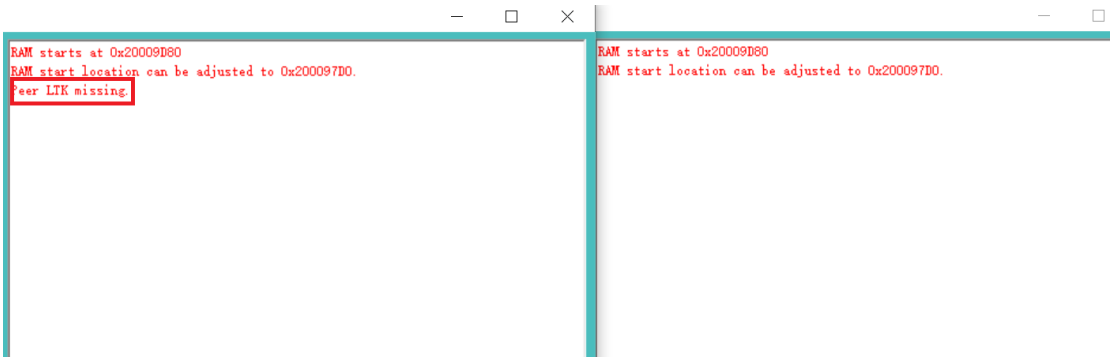


图 2-28 密钥丢失提示信息

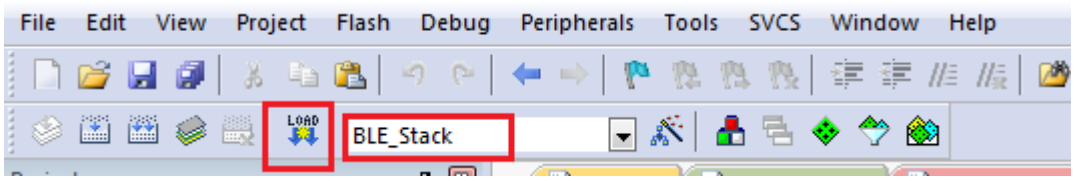
2. 4. 7 OTA使用指南

使用 OTA 无线升级功能时需要配置工程文件并通过手机 APK 实现。

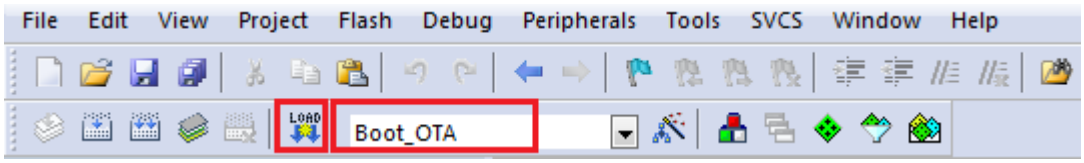
note：使用 OTA 功能时需要设置 sdk_conf.h 头文件：

#define BLE_BD_ADDR {{0x00,0x00,0x00,0x00,0x00,0x00}}

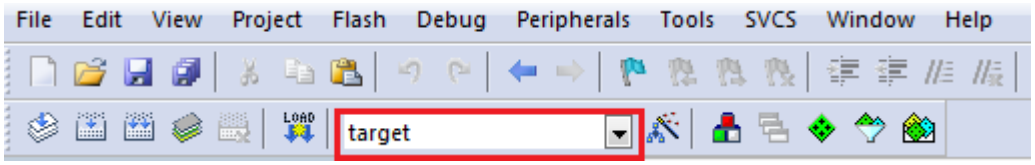
1. 打开工程文件 \ES32W3120_SDK_ARM\Projects\ES32W3120\Applications\Examples_BLE_peripheral\OTA_EN\MDK-ARM，选择 BLE_Stack，并点击 Load 下载协议栈



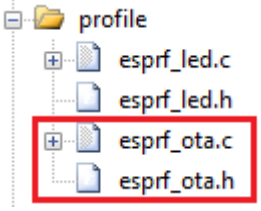
2. 选择 Boot_OTA，并点击 Load 下载协议栈



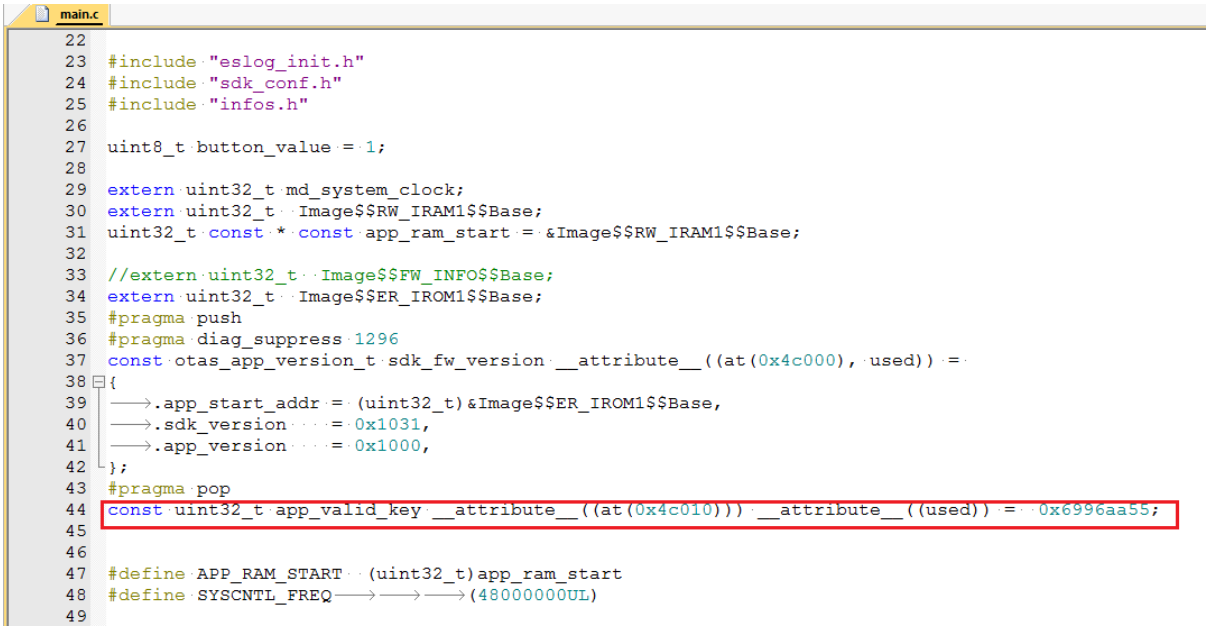
3. 选择待添加工程



4. 确认添加了 OTA_EN 的 profile



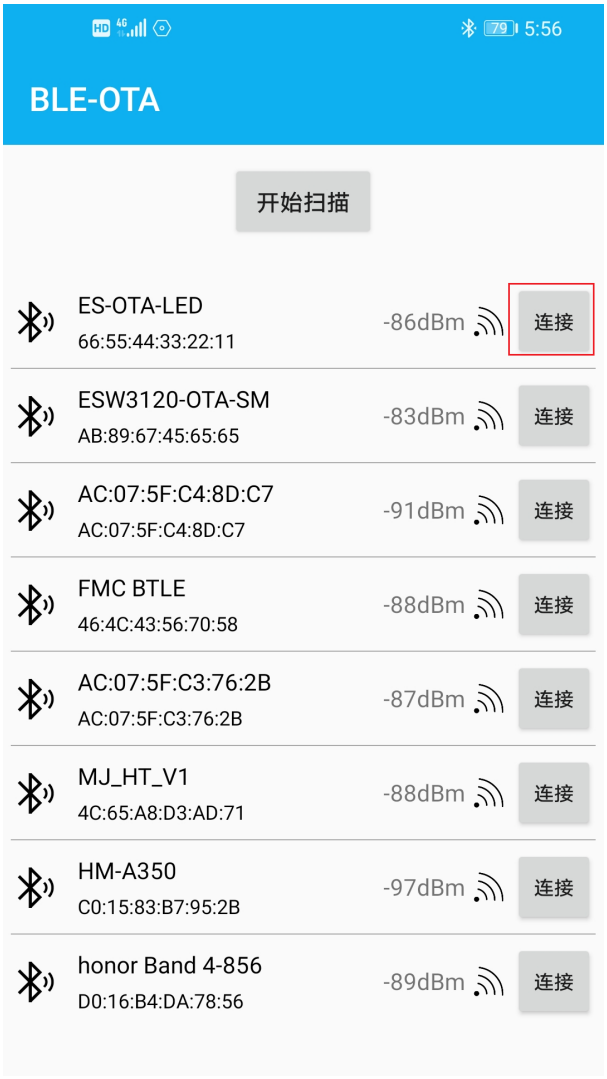
5. 若使用编译器下载的方式下载工程，需确认 KEY 数据包含编译



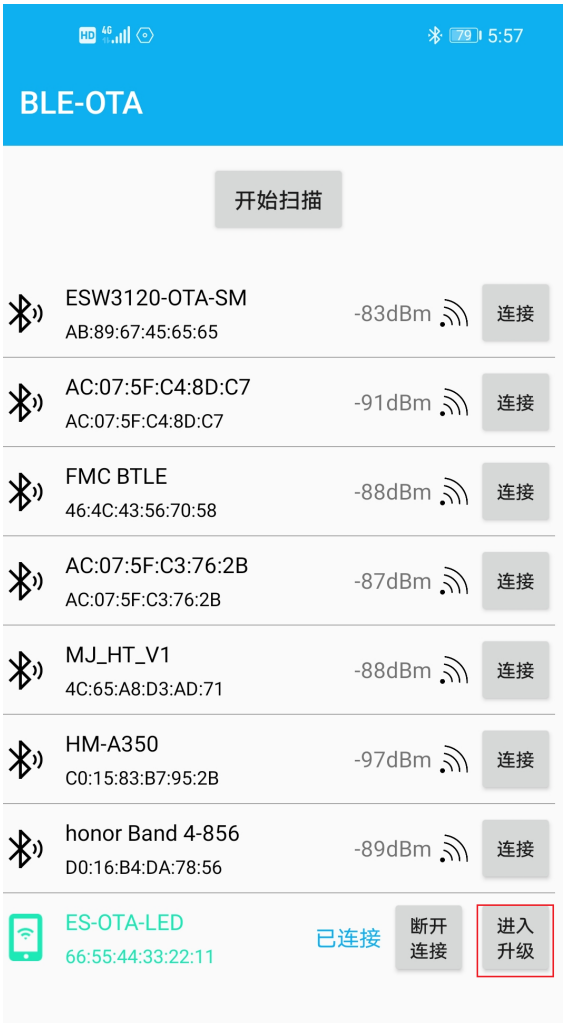
编译工程并下载

6. 安装\ES32W3120_SDK_ARM\Middlewares\EastSoft\BLE5.0 路径中的 OTA-BLE.apk 到安卓手机

7. 打开 APK 工具开始扫描，点击连接按钮连接到目标设备

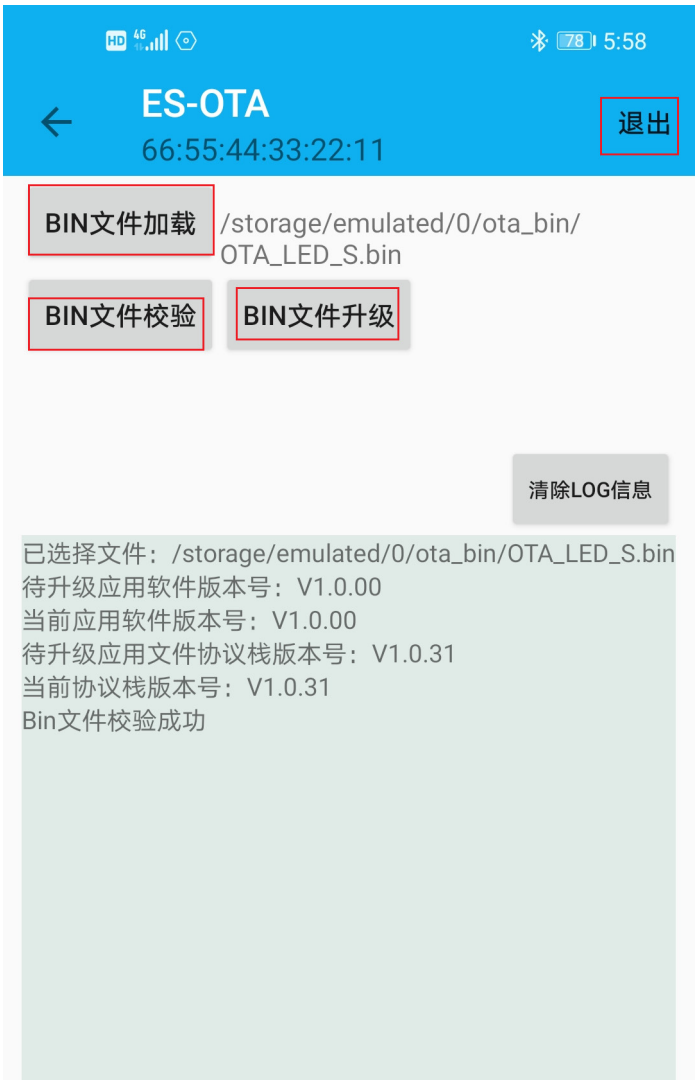


8. 连接成功后，点击"进入升级"按钮进入升级界面



9. 设备就绪后开始下一步操作

10. 依次点击"BIN 文件加载"加载待升级 BIN 文件，点击"BIN 文件校验"校验待升级 BIN 文件，点击"BIN 文件升级"执行 BIN 文件升级，升级成功后点击"退出"可退出 OTA 升级并进入升级后的 application code 区域执行



2.4.8 用户开发指南

1. 用户应按照图 2-5 和 2.4.2 节所示流程，初始化并配置协议栈后，才能调用对应接口进行用户自主开发。
2. 调用 BLE 接口基本遵循 cmd<-->event 结构，即用户调用命令后，协议栈经处理通过回调函数返回对应事件结果；BLE 上报接口基本遵循 Ind<--> Cfm 结构，即协议栈通过回调函数上报对应事件指示，用户按接口说明调用对应 cfm。详细说明见 API 接口说明。
3. 用户在将应用工程下载到芯片中后可通过手机 APP 进行连接测试。
4. 以 LED 工程为例，从机开发流程如下：
   a) Profile，用户可按照 esprf_led.c 中例程建立数据库表格，并在协议栈初始化完

成后调用服务添加接口添加数据库；用户可按照例程实现数据通用处理，如对端对数据库的读、写、指示、通知操作。当涉及硬件相关动作时，建议提交 App 进行处理。

b) App。外设实现时，App 在协议栈初始化完成后，可按产品设计要求，添加数据库、开启广播，并处理连接完成事件、参数更新、绑定加密等流程。并处理 profile 上报数据库读写操作；中心节点实现时，App 在协议栈初始化完成后，可按产品设计要求，开启扫描，处理扫描上报信息，按需建立连接、更新连接参数、绑定加密、发现对端数据库、设置特性通知指示、读/写特性值、处理特性值通知指示。

2. 4. 9 iDesigner IDE工程设置说明

SDK 包中提供 iDesigner IDE 的工程示例，其存放路径为：

\ESW_SDK_ES32W3120_V2.0.1\Projects\ES32W3120\Applications\Examples_BLE_peripheral\LED_Server\iDesigner

➢ 在使用 iDesigner IDE 的工程进行开发时需注意设置链接文件的路径:

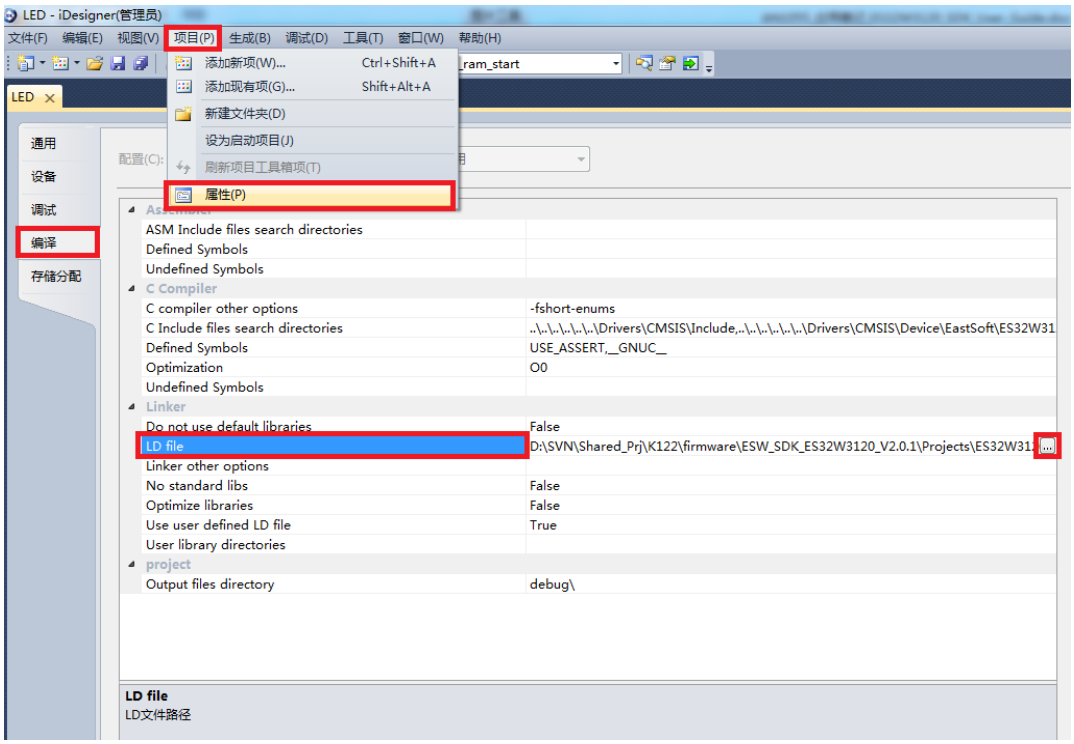


图 2-29 链接文件路径设置

将其设置为本工程存储的绝对路径，如：
\ESW_SDK_ES32W3120_V2.0.1\Projects\ES32W3120\Applications\Examples_BLE_peripheral\LED_Server\iDesigner\ led_ld.ld

➢ 下载调试工具选择

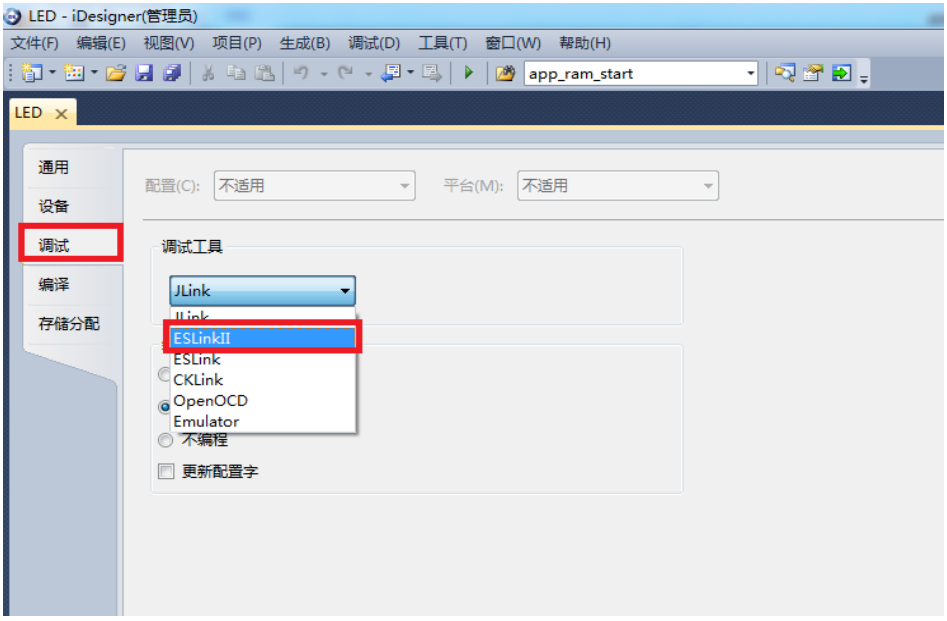


图 2-30 下载调试工具设置

➢ 例程调试

首先按照 2. 4. 2 章节中下载 HEX 的步骤将 stack hex 文件下载到开发板中，然后将应用工程代码下载到开发板中即可开始调试。

2. 4. 10 RTT Studio IDE工程设置说明

SDK 包中提供 RTT Studio IDE 的工程示例，其存放路径为：

\ESW_SDK_ES32W3120_V2.0.1\Projects\ES32W3120\Applications\Examples_BLE_peripheral\LED_Server\ RTT-Studio

➢ 添加头文件包含路径

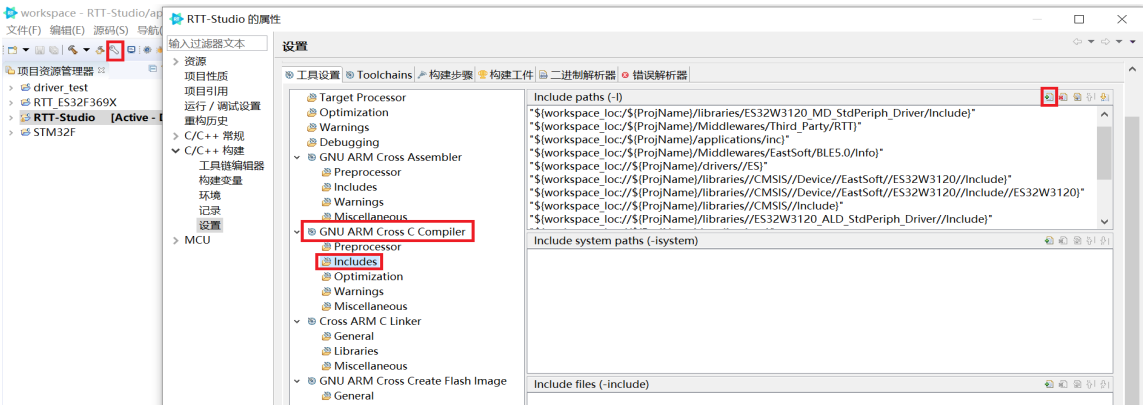


图 2-31 添加头文件包含路径

➢ 设置下载文件格式

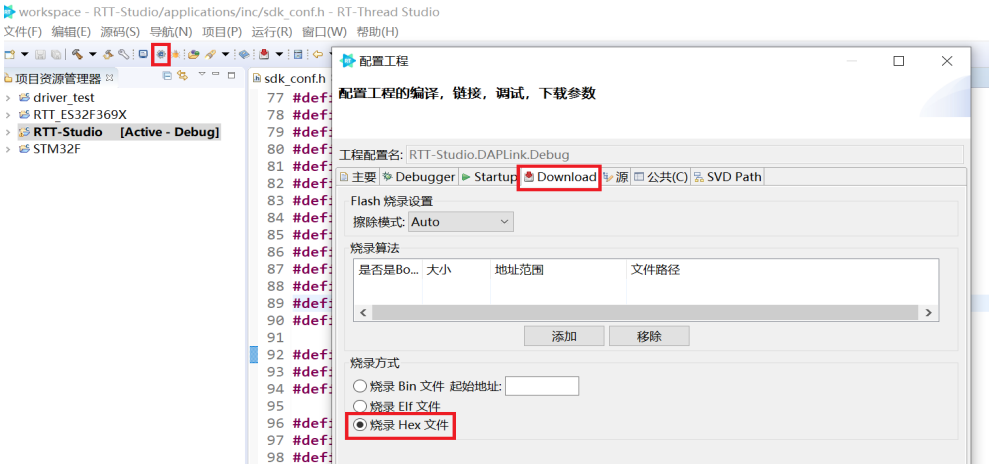


图 2-32 设置下载文件格式

➢ 设置下载调试工具

若使用 ESLINKII 进行烧录调试则选择下载调试工具为 DAP-LINK：

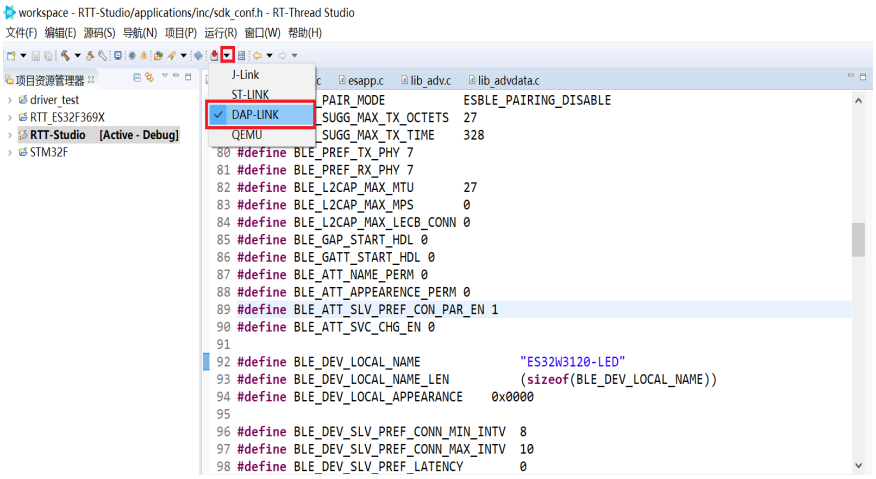


图 2-33 下载调试工具设置

3 LIB接口说明

为节省用户的开发时间，在基础 API 接口的基础上进一步进行封装形成 LIB 接口。

3.1 接口函数说明

3.1.1 设备配置

void lib_conf_init(esble_gapm_set_dev_config_cmd_t *conf, lib_conf_finish_t finish)

参数说明：

conf:设备配置结构体指针，指向用户配置的结构体参数地址，若此值为 NULL，则使用系统默认设备配置

finish:回调函数，当完成设备配置命令后调用此回调函数

函数功能：用于完成设备参数配置

3.1.2 广播参数配置

esble_err lib_adv_init(lib_adv_t *lib_adv, lib_adv_init_t *init)

参数说明：

lib_adv:全局参数，LIB 库广播参数结构体指针

init:广播参数结构体指针，指向用户配置的广播参数结构体

函数功能：用于完成广播参数配置

3.1.3 广播数据格式化转化

esble_err lib_advdata_encode(lib_advdata_t const * const p_advdata,
uint8_t * const p_encoded_data,
uint16_t * const p_len)

参数说明：

p_advdata:广播数据结构体指针，指向用户设置的广播数据结构体

p_encoded_data:字符指针，指向存放格式化广播数据的数组首地址

p_len:可转化的数据长度

函数功能：用于将用户设置的广播数据转化为标准的 BLE 广播数据

3.1.4 扫描参数配置

esble_err lib_scan_init(lib_scan_t *lib_scan, lib_scan_init_t *init)

参数说明：

lib_scan:全局参数，LIB 库扫描参数结构体指针

init:扫描参数结构体指针，指向用户配置的扫描参数结构体

函数功能：用于完成扫描参数配置

3.1.5 发起态参数配置

esble_err lib_init_init(lib_init_t *lib_init, lib_init_init_t *init, bool creat)

参数说明：

lib_init:全局参数，LIB 库发起态参数结构体指针

init:发起态参数结构体指针，指向用户配置的发起态参数结构体

creat:用于控制参数配置后是否开始创建发起事件，若此值设置为 0，则仅设置发起态参数并不执行创建事件。

函数功能：用于完成发起态参数配置

3.1.6 发现服务配置

esble_err lib_db_discovery_init(lib_db_discovery_t *lib_db_discovery, lib_db_discovery_init_t *init)

参数说明：

lib_db_discovery:全局参数，LIB 库发现服务参数结构体指针

init:发现服务参数结构体指针，指向用户配置的发现服务参数结构体

函数功能：用于完成发现服务参数配置

4 API接口说明

4.1 GAP

GAPM API 命令用于管理非连接态的事件，如设置设备参数、设置 ADV 参数及数据、设置 SCAN、INIT 参数及使能事件以及读取本机相关信息数据等。

GAPC API 命令用于管理与对端设备的连接，如检索对端设备信息，开始配对，加密连接，断开连接，基于 L2CAP 连接的交换 LE 信令等。

GAP 与 Application/Profile 和 BLE STACK 交互，有三种信息格式：命令、事件完成、指示。

命令是用户使用发给 BLE STACK 的下行命令；每个命令都会返回一个事件完成，用于指示命令是否正确发送；指示用于 BLE STACK 返回给用户，其中包含用户需要的数据信息。

在使用 API 接口函数进行应用开发时，需要首先依次调用 esble_gapm_reset()和 esble_gapm_set_dev_conf()函数来完成芯片的复位和配置，然后才能够执行后续的广播/扫描/发起连接操作。

4.1.1 数据定义

4.1.1.1 下行命令定义

```
typedef enum _esble_gapm_cmd_enum
{
ESBLE_GAPM_RESET                = ESBLE_GAPM_BASE,
ESBLE_GAPM_SET_DEV_CONF,
ESBLE_GAPM_CREATE_ADV,
ESBLE_GAPM_SET_ADV_DATA,
ESBLE_GAPM_SET_SCAN_RSP_DATA,
ESBLE_GAPM_ADV_START,
ESBLE_GAPM_ADV_STOP,
ESBLE_GAPM_CREATE_SCAN,
ESBLE_GAPM_SCAN_START,
ESBLE_GAPM_SCAN_STOP,
ESBLE_GAPM_CREATE_INIT,
ESBLE_GAPM_INIT_START,
ESBLE_GAPM_INIT_STOP,
ESBLE_GAPM_CREATE_PERIOD_SYNC,
ESBLE_GAPM_SET_PERIOD_ADV_DATA,
ESBLE_GAPM_PERIOD_SYNC_START,
```

ESBLE_GAPM_PERIOD_SYNC_STOP,

ESBLE_GAPM_STOP_ALL_ACTIVITIES,

ESBLE_GAPM_DELETE_ACTIVITY,

ESBLE_GAPM_DELETE_ALL_ACTIVITY,

ESBLE_GAPM_GET_NB_ADV_SETS,

ESBLE_GAPM_GET_DEV_ADV_TX_POWER,

ESBLE_GAPM_GET_MAX_LE_ADV_DATA_LEN,

ESBLE_GAPM_GET_PAL_SIZE,

ESBLE_GAPM_SET_PAL,

ESBLE_GAPM_USE_ENC_BLOCK,

ESBLE_GAPM_GEN_RAND_NB,

ESBLE_GAPM_SET_IRK,

ESBLE_GAPM_GEN_DH_KEY,

ESBLE_GAPM_GET_PUB_KEY,

ESBLE_GAPM_GET_RAL_LOC_ADDR,

ESBLE_GAPM_GET_RAL_PEER_ADDR,

ESBLE_GAPM_RESOLV_ADDR,

ESBLE_GAPM_GET_DEV_BDADDR,

ESBLE_GAPM_GET_RAL_SIZE,

ESBLE_GAPM_GEN_RAND_ADDR,

ESBLE_GAPM_SET_RAL,

ESBLE_GAPM_SET_CHANNEL_MAP,

ESBLE_GAPM_GET_DEV_VERSION,

ESBLE_GAPM_GET_DEV_TX_PWR,

ESBLE_GAPM_GET_SUGGESTED_DFLT_LE_DATA_LEN,

ESBLE_GAPM_GET_DEV_RF_PATH_COMP,

ESBLE_GAPM_GET_MAX_LE_DATA_LEN,

ESBLE_GAPM_SET_WL,

ESBLE_GAPM_GET_WLIST_SIZE,

ESBLE_GAPM_UNKNOWN_TASK_MSG,

ESBLE_GAPM_RENEW_ADDR,

ESBLE_GAPM_LE_TEST_RX_START,

```
    ESBLE_GAPM_LE_TEST_TX_START,
    ESBLE_GAPM_LE_TEST_STOP,
    ESBLE_GAPM_PROFILE_TASK_ADD,
    ESBLE_GAPM_PLF_RESET,
    ESBLE_GAPM_DBG_GET_MEM_INFO,
    ESBLE_GAPM_LEPSM_REG,
    ESBLE_GAPM_LEPSM_UNREG,
} esble_gapm_cmd_enum;
typedef enum _esble_gapc_cmd_enum
{
    ESBLE_GAPC_DEV_INFO_CFM          = ESBLE_GAPC_BASE,
    ESBLE_GAPC_CONN_CFM,
    ESBLE_GAPC_DISCONN,
    ESBLE_GAPC_PARAM_UPD,
    ESBLE_GAPC_SET_PREF_SLV_LATENCY,
    ESBLE_GAPC_GET_CON_RSSI,
    ESBLE_GAPC_GET_CON_CNL_MAP,
    ESBLE_GAPC_ENCRYPT,
    ESBLE_GAPC_ENCRYPT_CFM,
    ESBLE_GAPC_SEC,
    ESBLE_GAPC_KEY_PRESS_NOTIF,
    ESBLE_GAPC_BOND,
    ESBLE_GAPC_BOND_CFM,
    ESBLE_GAPC_GET_ADDR_RESOL_SUPP,
    ESBLE_GAPC_GET_PEER_NAME,
    ESBLE_GAPC_GET_PEER_VERSION,
    ESBLE_GAPC_GET_PEER_FEATURES,
    ESBLE_GAPC_GET_PEER_APPEARANCE,
    ESBLE_GAPC_GET_PEER_SLV_PREF_PARAMS,
    ESBLE_GAPC_GET_PHY,
    ESBLE_GAPC_GET_CHAN_SEL_ALGO,
    ESBLE_GAPC_SET_DEV_INFO_CFM,
```

```
ESBLE_GAPC_SET_LE_PING_TO,
ESBLE_GAPC_GET_LE_PING_TO,
ESBLE_GAPC_SIGN_PACKET,
ESBLE_GAPC_SIGN_CHECK,
ESBLE_GAPC_SET_PHY,
ESBLE_GAPC_SET_LE_PKT_SIZE,
} esble_gapc_cmd_enum;
```

4.1.1.2 私有配置

```
enum gapm_priv_cfg
{
GAPM_PRIV_CFG_PRIV_ADDR_BIT = (1 << 0),    /*0:public addr;1:random addr*/
GAPM_PRIV_CFG_RSVD          = (1 << 1),    /*保留，未使用*/
GAPM_PRIV_CFG_PRIV_EN_BIT   = (1 << 2),    /*指示控制器私密功能是否使能*/
};
```

4.1.1.3 配对模式配置

```
typedef enum _esble_pairing_mode_enum
{
ESBLE_PAIRING_DISABLE= 0,              /*不使能配对*/
ESBLE_PAIRING_LEGACY = (1 << 0),       /*传统配对*/
ESBLE_PAIRING_SEC_CON= (1 << 1),       /*安全连接配对*/
ESBLE_GAPM_PAIRING_FORCE_P256_KEY_GEN= (1<<7),/*强制重新生成公有及私有密
                                                */钥
} esble_pairing_mode_enum;
```

4.1.1.4 属性数据库配置

```
enum gapm_write_att_perm
{
GAPM_WRITE_DISABLE      = 0,   /*禁止写操作*/
GAPM_WRITE_NO_AUTH      = 1,   /*使能写操作，无认证需求*/
GAPM_WRITE_UNAUTH       = 2,   /*需要在非认证连接下写操作*/
GAPM_WRITE_AUTH         = 3,   /*需要在认证连接下写操作*/
GAPM_WRITE_SEC_CON      = 4    /*需要在安全连接下写操作*/
};
```

gapm_att_cfg_flag:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DBG	DBGT	RFU						Service Change	Pref Con Par	Appearance Permission			Name Permission		

Name Permission:对端设备对 Device Name 写操作的权限需求

Appearance Permission:对端设备对 Device Appearance 写操作的权限需求

Pref Con Par：GAP 属性数据库中存在 Slave Preferred Connection Parameters

Service Change：GATT 数据库中有 Service Change feature

4. 1. 1. 5 优先使用的LE PHY配置

```
enum gap_phy
{
GAP_PHY_ANY                 = 0x00,     /*无优先 PHY 设置*/
GAP_PHY_LE_1MBPS            = (1 << 0), /*活动连接优先使用 LE 1M PHY*/
GAP_PHY_LE_2MBPS            = (1 << 1), /*活动连接优先使用 LE 2M PHY*/
GAP_PHY_LE_CODED            = (1 << 2), /*活动连接优先使用 LE CODED PHY*/
};
```

4. 1. 1. 6 PAL配置

```
typedef struct _esble_gapm_period_adv_addr_cfg_t
{
esble_gap_bdaddr_t      addr;       /*广播设备的地址信息*/
uint8_t                 adv_sid;    /*广播设备的 SID*/
} esble_gapm_period_adv_addr_cfg_t;
```

4. 1. 1. 7 RAL配置

```
typedef struct _esble_gap_ral_dev_info_t
{
esble_gap_bdaddr_t      addr;                       /*设备地址*/
uint8_t                 priv_mode;                  /*私有模式*/
uint8_t                 peer_irk[ESBLE_GAP_KEY_LEN];    /*对端 IRK*/
uint8_t                 local_irk[ESBLE_GAP_KEY_LEN];   /*本机 IRK*/
}esble_gap_ral_dev_info_t;
```

4.1.1.8 随机地址类型

```
typedef enum _esble_gap_rnd_addr_type_enum
{
ESBLE_GAP_STATIC_ADDR        = 0xC0,    /*静态随机地址*/
ESBLE_GAP_NON_RSLV_ADDR      = 0x00,    /*不可解析随机地址*/
ESBLE_GAP_RSLV_ADDR          = 0x40,    /*可解析随机地址*/
} esble_gap_rnd_addr_type_enum;
```

4.1.1.9 本机地址类型

```
typedef enum _esble_own_addr_enum
{
ESBLE_STATIC_ADDR,           /*Public 或 Private 静态地址*/
ESBLE_GEN_RSLV_ADDR,         /*可解析随机地址*/
ESBLE_GEN_NON_RSLV_ADDR,     /*不可解析随机地址*/
} esble_own_addr_enum;
```

4.1.1.10 广播参数

```
typedef enum _esble_gapm_adv_type_enum{
ESBLE_GAPM_ADV_TYPE_LEGACY     = 0,    /* Legacy advertising*/.
ESBLE_GAPM_ADV_TYPE_EXTENDED,          /* Extended advertising*/
ESBLE_GAPM_ADV_TYPE_PERIODIC,          /* Periodic advertising*/.
} esble_gapm_adv_type_enum;
typedef enum _esble_gapm_adv_disc_mode_enum
{
ESBLE_GAPM_ADV_MODE_NON_DISC = 0,      /*不可发现模式*/
ESBLE_GAPM_ADV_MODE_GEN_DISC,          /*通用可发现模式*/
ESBLE_GAPM_ADV_MODE_LIM_DISC,          /*限时可发现模式*/
ESBLE_GAPM_ADV_MODE_BEACON,            /*广播数据中不含 AD_TYPE_FLAG 的广
                                       播模式*/
ESBLE_GAPM_ADV_MODE_MAX,
} esble_gapm_adv_disc_mode_enum;
typedef enum _esble_gapm_adv_prop_pos_enum
{
ESBLE_GAPM_ADV_PROP_CONNECTABLE_POS    = 0, /*可连接广播，不适用于 PER*/
```

```
    ESBLE_GAPM_ADV_PROP_SCANNABLE_POS,          /*可扫描广播*/
    ESBLE_GAPM_ADV_PROP_DIRECTED_POS,           /*定向广播仅用于可连接的
                legacy 和 extended 广播模式，或不可连接不可发现的 extended 广播模式*/
    ESBLE_GAPM_ADV_PROP_HDC_POS,                /*High duty cycle 广播*/
    ESBLE_GAPM_ADV_PROP_RESERVED_4_POS,         /*预留*/
    ESBLE_GAPM_ADV_PROP_ANONYMOUS_POS,          /*匿名广播，仅用于 extended
                                                广播模式*/
    ESBLE_GAPM_ADV_PROP_TX_PWR_POS,             /*广播数据中包含发送功率数
                                            据，仅用于 extended 广播模式*/
    ESBLE_GAPM_ADV_PROP_PER_TX_PWR_POS,         /*PER 广播中包含发送功率数
                                                据，仅用于 PER*/
    ESBLE_GAPM_ADV_PROP_SCAN_REQ_NTF_EN_POS,    /*接收到扫描请求时是否通知
                                                    application*/
} esble_gapm_adv_prop_pos_enum;
typedef enum _esble_gapm_adv_filter_policy_enum
{
    ESBLE_GAPM_ADV_ALLOW_SCAN_ANY_CON_ANY  = 0x00,  /*允许接收所有设备发送
                                                    的扫描请求和连接请求*/
    ESBLE_GAPM_ADV_ALLOW_SCAN_WLST_CON_ANY,     /*允许接收白名单中的设备发
                                            送的扫描请求和所有设备发送的连接请求*/
    ESBLE_GAPM_ADV_ALLOW_SCAN_ANY_CON_WLST,     /*允许接收白名单中的设备发
                                            送的连接请求和所有设备发送的扫描请求*/
    ESBLE_GAPM_ADV_ALLOW_SCAN_WLST_CON_WLST,    /*仅允许接收白名单中的设备
                                                发送的扫描请求和连接请求*/
} esble_gapm_adv_filter_policy_enum;
typedef enum _esble_gapm_phy_type_enum
{
    ESBLE_GAPM_PHY_TYPE_LE_1M  = 1,         /*LE 1M.*/
    ESBLE_GAPM_PHY_TYPE_LE_2M,              /*LE 2M.*/
    ESBLE_GAPM_PHY_TYPE_LE_CODED,           /* LE Coded.*/
} esble_gapm_phy_type_enum;
typedef struct _esble_gapm_adv_prim_cfg_t
{
    uint32_t  adv_intv_min;     /*广播事件的最小间隔，单位为 625us，需大于 20ms*/
    uint32_t  adv_intv_max;     /*广播事件的最小间隔，单位为 625us，需大于 20ms*/
```

```
uint8_t         chnl_map;     /*使用的信道位设置*/
uint8_t         phy;          /*primary 广播所使用 PHY 设置，不允许使用 2M*/
} esble_gapm_adv_prim_cfg_t;
typedef struct _esble_gapm_adv_second_cfg_t
{
uint8_t   max_skip;     /*在发送 AUX_ADV_IND 数据包前可跳过的最大广播事件数*/
uint8_t   phy;          /*secondary 广播所使用的 PHY 设置*/
uint8_t   adv_sid;      /*广播 SID*/
} esble_gapm_adv_second_cfg_t;
typedef struct _esble_gapm_adv_period_cfg_t
{
uint16_t  adv_intv_min;     /*最小广播间隔，单位为 1.25ms，需大于 20ms*/
uint16_t  adv_intv_max;     /*最大广播间隔，单位为 1.25ms，需大于 20ms*/
} esble_gapm_adv_period_cfg_t;
typedef struct _esble_gapm_create_adv_param_t
{
uint8_t                         type;       /*广播类型*/
uint8_t                         disc_mode;  /*发现模式*/
uint16_t                        prop;       /*特性设置*/
int8_t                          max_tx_pwr; /*广播包的最大发送功率等级，取值范围
                                              为[-127,126]，单位为 dBm*/
uint8_t                         filter_pol; /*过滤机制*/
esble_gap_bdaddr_t              peer_addr;  /*对端设备地址（仅用于定向广播）*/
esble_gapm_adv_prim_cfg_t       prim_cfg;   /*设置 primary advertising*/
esble_gapm_adv_second_cfg_t     second_cfg; /*设置 secondary advertising，仅在广播
                                              类型为 GAPM_ADV_TYPE_EXTENDED 或
                                              GAPM_ADV_TYPE_PERIODIC 有效.*/
esble_gapm_adv_period_cfg_t     period_cfg; /*设置周期性广播配置，仅在广播类型为
                                              GAPM_ADV_TYPE_PERIODIC 时有效)*/
} esble_gapm_create_adv_param_t;
```

4. 1. 1. 11 扫描参数

```
typedef enum _esble_gapm_scan_type
{
```

```
ESBLE_GAPM_SCAN_TYPE_GEN_DISC = 0,       /*通用发现*/
ESBLE_GAPM_SCAN_TYPE_LIM_DISC,           /*限时发现*/
ESBLE_GAPM_SCAN_TYPE_OBSERVER,           /*Observer*/
ESBLE_GAPM_SCAN_TYPE_SEL_OBSERVER,       /*使用白名单的 observer*/
ESBLE_GAPM_SCAN_TYPE_CONN_DISC,          /*仅接受可连接的广播*/
ESBLE_GAPM_SCAN_TYPE_SEL_CONN_DISC,      /*仅接受白名单内可连接的广播*/
}esble_gapm_scan_type;
typedef enum _esble_gapm_scan_prop
{
ESBLE_GAPM_SCAN_PROP_PHY_1M_BIT          = (1 << 0),/*在 LE 1M PHY 扫描*/
ESBLE_GAPM_SCAN_PROP_PHY_CODED_BIT   = (1 << 1),/*在 LE Coded PHY 扫描*/
ESBLE_GAPM_SCAN_PROP_ACTIVE_1M_BIT   = (1 << 2), /*在 LE 1M PHY 主动扫描*/
ESBLE_GAPM_SCAN_PROP_ACTIVE_CODED_BIT = (1 << 3), /*在 LE Coded PHY 主
                                                    动扫描*/
ESBLE_GAPM_SCAN_PROP_ACCEPT_RPA_BIT    = (1 << 4),/*当使用 RPA 时接受不
                                                    能解析目标地址的定向广播数据*/
ESBLE_GAPM_SCAN_PROP_FILT_TRUNC_BIT    = (1 << 5),/*设置是否过滤不完整的
                                                    广播或扫描响应数据包*/
}esble_gapm_scan_prop;
typedef struct _esble_gapm_scan_wd_op_param_t
{
uint16_t      scan_intv;          /*扫描间隔*/
uint16_t      scan_wd;            /*扫描窗口*/
}esble_gapm_scan_wd_op_param_t;
typedef struct _esble_gapm_scan_param_t
{
uint8_t                   type;          /*扫描类型*/
uint8_t                   prop;          /*扫描特性*/
uint8_t                   dup_filt_pol;  /*重复包过滤*/
uint8_t                   rsvd;          /*保留*/
esble_gapm_scan_wd_op_param_t  scan_param_1m;       /*LE 1M PHY 扫描窗口参数*/
esble_gapm_scan_wd_op_param_t  scan_param_coded;/*LE Coded PHY 扫描窗口参数*/
uint16_t                  duration; /*扫描持续事件，单位为 10ms，若为 0 表示无
                                      时间限制，会持续扫描直到接收到停止扫描命令*/
```

```
uint16_t                period;   /*扫描周期，单位为 1.28s，若为 0 表示扫描过程不
                                  是周期性事件*/
}esble_gapm_scan_param_t;
```

4. 1. 1. 12 发起连接参数

```
typedef enum _esble_gapm_init_type
{
ESBLE_GAPM_INIT_TYPE_DIRECT_CONN_EST = 0,  /*对指定设备地址建立连接*/
ESBLE_GAPM_INIT_TYPE_AUTO_CONN_EST,    /*对白名单中的设备自动建立连接*/
ESBLE_GAPM_INIT_TYPE_NAME_DISC,/*与指定的设备建立连接并读取对端设备名称*/
}esble_gapm_init_type;
typedef enum _esble_gapm_init_prop
{
ESBLE_GAPM_INIT_PROP_1M_BIT       = (1 << 0),/*在 LE 1M PHY 扫描可连接广播*/
ESBLE_GAPM_INIT_PROP_2M_BIT       = (1 << 1), /*在 LE 2M PHY 扫描可连接广播*/
ESBLE_GAPM_INIT_PROP_CODED_BIT= (1 << 2),/*在 LE CODED PHY 扫描可连接广播*/
}esble_gapm_init_prop;
typedef struct _esble_gapm_conn_param_t
{
uint16_t    conn_intv_min;   /*最小连接间隔，单位是 1.25ms，取值范围为[7.5ms,4s]*/
uint16_t    conn_intv_max;   /*最小连接间隔，单位是 1.25ms，取值范围为[7.5ms,4s]*/
uint16_t    conn_latency;    /*从机延迟，设置从机可以忽略的事件个数*/
uint16_t    supervision_to;  /*连接超时时间，单位是 10ms，取值范围为[100ms,32s]*/
uint16_t    ce_len_min;      /*推荐的连接事件最小持续时间，单位为 625us*/
uint16_t    ce_len_max;      /*推荐的连接事件最大持续时间，单位为 625us*/
}esble_gapm_conn_param_t;
typedef struct _esble_gapm_init_param_t
{
uint8_t                 type;        /*发起连接类型*/
uint8_t                 prop;        /*发起连接特性*/
uint16_t                conn_to;     /*设置自动连接建立的超时时间，单位是
                                     10ms，若在设置的时间内指定的设备没有全部建立连接
                                     在取消连接规程，若设置为 0 则表示无超时限制*/
esble_gapm_scan_wd_op_param_t   scan_param_1m;  /*LE 1M PHY 扫描窗口参数*/
```

```
esble_gapm_scan_wd_op_param_t   scan_param_coded;/*LE CODED PHY 扫描窗口参数*/
esble_gapm_conn_param_t         conn_param_1m;      /*LE 1M PHY 连接参数*/
esble_gapm_conn_param_t         conn_param_2m;      /*LE 2M PHY 连接参数*/
esble_gapm_conn_param_t         conn_param_coded;   /*LE CODED PHY 连接参数*/
esble_gap_bdaddr_t              peer_addr;/*若未使用白名单则使用此指定对端地址*/
}esble_gapm_init_param_t;
```

4. 1. 1. 13 周期同步参数

```
typedef enum _esble_gapm_per_sync_type_enum
{
ESBLE_GAPM_PER_SYNC_TYPE_GENERAL     = 0, /*不使用 PAL 中的地址信息同步，而
                                           使用开始命令中的地址信息*/
ESBLE_GAPM_PER_SYNC_TYPE_SELECTIVE,  /*使用 PAL 中的地址信息来进行同步*/
} esble_gapm_per_sync_type_enum;
```

4. 1. 1. 14 信道表

```
#define ESBLE_GAP_LE_CHNL_MAP_LEN    (0x05)        /*信道表长度*/
typedef struct _esble_le_chnl_map_t
{
uint8_t   map[ESBLE_GAP_LE_CHNL_MAP_LEN];     /*信道表*/
} esble_le_chnl_map_t;
```

4. 1. 1. 15 设备地址

```
#define ESBLE_GAP_BD_ADDR_LEN        (6)        /*地址长度*/
typedef struct _esble_bd_addr_t
{
uint8_t   addr[ESBLE_GAP_BD_ADDR_LEN];     /*6 字节地址*/
} esble_bd_addr_t;
typedef struct _esble_gap_bdaddr_t
{
esble_bd_addr_t      addr;          /*设备地址*/
uint8_t              addr_type;     /*地址类型，0=public/1=private random.*/
} esble_gap_bdaddr_t;
```

4. 1. 1. 16 测试模式数据包类型

```
typedef enum _esble_gap_pkt_pld_type_enum
```

```
{
ESBLE_GAP_PKT_PLD_PRBS9,                    /* PRBS9 序列*/
ESBLE_GAP_PKT_PLD_REPEATED_11110000,        /*11110000 序列*/
ESBLE_GAP_PKT_PLD_REPEATED_10101010,        /*10101010 序列*/
ESBLE_GAP_PKT_PLD_PRBS15,                   /* PRBS15 序列*/
ESBLE_GAP_PKT_PLD_REPEATED_11111111,        /*11111111 序列*/
ESBLE_GAP_PKT_PLD_REPEATED_00000000,        /*00000000 序列*/
ESBLE_GAP_PKT_PLD_REPEATED_00001111,        /*00001111 序列*/
ESBLE_GAP_PKT_PLD_REPEATED_01010101,        /*01010101 序列*/
}esble_gap_pkt_pld_type_enum;
```

4. 1. 1. 17 PHY

```
typedef enum _esble_gap_test_phy_enum
{
ESBLE_GAP_TEST_PHY_1MBPS        = 1,        /* LE 1M PHY (TX or RX)*/
ESBLE_GAP_TEST_PHY_2MBPS        = 2,        /* LE 2M PHY (TX or RX)*/
ESBLE_GAP_TEST_PHY_CODED        = 3,        /*编码 PHY (RX Only)*/
ESBLE_GAP_TEST_PHY_125KBPS      = 3,        /*125k*/
ESBLE_GAP_TEST_PHY_500KBPS      = 4,        /*500k*/
}esble_gap_test_phy_enum;
typedef enum _esble_gap_phy_enum
{
ESBLE_GAP_PHY_ANY          = 0x00,      /*无首选 PHY*/
ESBLE_GAP_PHY_LE_1MBPS  = (1 << 0),     /*首选 LE 1M PHY*/
ESBLE_GAP_PHY_LE_2MBPS  = (1 << 1),     /*首选 LE 2M PHY*/
ESBLE_GAP_PHY_LE_CODED = (1 << 2),      /*首选 LE Coded PHY*/
} esble_gap_phy_enum;
typedef enum _esble_gapc_phy_option
{
ESBLE_GAPC_PHY_OPT_LE_CODED_ALL_RATES = (1 << 0),  /*无首选空中速率*/
ESBLE_GAPC_PHY_OPT_LE_CODED_500K_RATE = (1 << 1),  /*首选 500kbps*/
ESBLE_GAPC_PHY_OPT_LE_CODED_125K_RATE = (1 << 2),  /*首选 125kbps*/
}esble_gapc_phy_option;
```

4. 1. 1. 18 调制类型

```
typedef enum _esble_gap_modulation_idx_enum
{
ESBLE_GAP_MODULATION_STANDARD,          /*标准调制*/
ESBLE_GAP_MODULATION_STABLE,            /*stable 调制*/
}esble_gap_modulation_idx_enum;
```

4. 1. 1. 19 设备名称

```
typedef struct _esble_gapc_dev_name_t
{
uint16_t    length;          /*设备名称长度*/
uint8_t     *name;           /*设备名称*/
} esble_gapc_dev_name_t;
```

4. 1. 1. 20 从设备首选参数

```
typedef struct _esble_gapc_slv_pref_t
{
uint16_t    con_intv_min;        /*最小连接间隔*/
uint16_t    con_intv_max;        /*最大连接间隔*/
uint16_t    slave_latency;       /*从设备握手延迟*/
uint16_t    conn_timeout;        /*连接超时*/
} esble_gapc_slv_pref_t;
```

4. 1. 1. 21 安全秘钥

```
#define ESBLE_GAP_KEY_LEN    (16)    /*秘钥长度*/
typedef struct _esble_gap_sec_key_t
{
uint8_t key[ESBLE_GAP_KEY_LEN];      /*秘钥：MSB -> LSB*/
} esble_gap_sec_key_t;
```

4. 1. 1. 22 身份验证

```
typedef enum _esble_gap_auth_mask_enum
{
ESBLE_GAP_AUTH_NONE = 0,             /*无标志设置*/
```

```
ESBLE_GAP_AUTH_BON = (1 << 0),          /*绑定*/
ESBLE_GAP_AUTH_MITM = (1 << 2),         /*MITM*/
ESBLE_GAP_AUTH_SEC_CON = (1 << 3),      /*安全连接*/
ESBLE_GAP_AUTH_KEY_NOTIF = (1 << 4),    /*秘钥通知*/
} esble_gap_auth_mask_enum;

typedef enum _esble_gap_auth_enum
{
ESBLE_GAP_AUTH_REQ_NO_MITM_NO_BOND = (ESBLE_GAP_AUTH_NONE),
                                        /*无 MITM 无绑定*/
ESBLE_GAP_AUTH_REQ_NO_MITM_BOND = (ESBLE_GAP_AUTH_BOND),
                                        /*无 MITM 有绑定*/
ESBLE_GAP_AUTH_REQ_MITM_NO_BOND = (ESBLE_GAP_AUTH_MITM),
                                        /*有 MITM 无绑定*/
ESBLE_GAP_AUTH_REQ_MITM_BOND = (ESBLE_GAP_AUTH_MITM |
                                ESBLE_GAP_AUTH_BOND),
                                        /*有 MITM 有绑定*/
ESBLE_GAP_AUTH_REQ_SEC_CON_NO_BOND =
                                        (ESBLE_GAP_AUTH_SEC_CON),
                                        /*安全连接无绑定*/
ESBLE_GAP_AUTH_REQ_SEC_CON_BOND = (ESBLE_GAP_AUTH_SEC_CON |
                                   ESBLE_GAP_AUTH_BOND),
                                        /*安全连接有绑定*/
ESBLE_GAP_AUTH_REQ_LAST,
ESBLE_GAP_AUTH_REQ_MASK = 0x1F,
} esble_gap_auth_enum;
```

4. 1. 1. 23 LTK

```
typedef struct _esble_gapc_ltk_t
{
esble_gap_sec_key_t  ltk;           /*LTK*/
uint16_t             ediv;          /*加密转换器*/
esble_rand_nb_t      randnb;        /*随机数*/
uint8_t              key_size;      /*加密秘钥长度(7 to 16)*/
} esble_gapc_ltk_t;
```

4. 1. 1. 24 随机数

```
#define ESBLE_GAP_RAND_NB_LEN          (0x08)
typedef struct _esble_rand_nb_t
{
uint8_t        nb[ESBLE_GAP_RAND_NB_LEN];        /*8 字节随机数数组*/
} esble_rand_nb_t;
```

4. 1. 1. 25 配对

```
typedef enum _esble_gap_io_cap_enum
{
ESBLE_GAP_IO_CAP_DISPLAY_ONLY   = 0x00,          /*只显示*/
ESBLE_GAP_IO_CAP_DISPLAY_YES_NO,                 /*是否显示*/
ESBLE_GAP_IO_CAP_KB_ONLY,                        /*只有键盘*/
ESBLE_GAP_IO_CAP_NO_INPUT_NO_OUTPUT,             /*无输入无输出*/
ESBLE_GAP_IO_CAP_KB_DISPLAY,                     /*键盘和显示*/
ESBLE_GAP_IO_CAP_LAST
} esble_gap_io_cap_enum;

typedef enum _esble_gap_oob_enum
{
ESBLE_GAP_OOB_AUTH_DATA_NOT_PRESENT = 0x00,      /*无 OOB*/
ESBLE_GAP_OOB_AUTH_DATA_PRESENT,                 /*有 OOB*/
ESBLE_GAP_OOB_AUTH_DATA_LAST
} esble_gap_oob_enum;

typedef enum _esble_gap_kdist_enum
{
ESBLE_GAP_KDIST_NONE        = 0x00,    /*不分配秘钥*/
ESBLE_GAP_KDIST_ENCKEY      = (1 << 0),  /*分配加密秘钥*/
ESBLE_GAP_KDIST_IDKEY       = (1 << 1),  /*分配 IRK*/
ESBLE_GAP_KDIST_SIGNKEY     = (1 << 2),  /*分配 CSRK*/
ESBLE_GAP_KDIST_LINKKEY     = (1 << 3),  /*分配 LTK*/
```

```
ESBLE_GAP_KDIST_LAST      = (1 << 4)
} esble_gap_kdist_enum;

typedef enum _esble_gap_sec_req_enum
{
ESBLE_GAP_NO_SEC = 0x00,              /*无认证和加密*/
ESBLE_GAP_SEC1_NOAUTH_PAIR_ENC,    /*无认证的加密配对*/
ESBLE_GAP_SEC1_AUTH_PAIR_ENC,      /*有认证的加密配对*/
ESBLE_GAP_SEC2_NOAUTH_DATA_SGN,    /*无认证的数字签名配对*/
ESBLE_GAP_SEC2_AUTH_DATA_SGN,      /*有认证的数字签名配对*/
ESBLE_GAP_SEC1_SEC_CON_PAIR_ENC,   /*加密的安全连接配对*/
} esble_gap_sec_req_enum;

typedef struct _esble_gapc_pairing_t
{
uint8_t   iocap;              /*IO 功能*/
uint8_t   oob;                /*OOB 信息*/
uint8_t   auth;               /*身份验证*/
uint8_t   key_size;           /*加密秘钥长度 7 to 16)*/
uint8_t   ikey_dist;          /*发起者秘钥分配*/
uint8_t   rkey_dist;          /*响应者分配秘钥*/
uint8_t   sec_req;            /*最低安全等级*/
} esble_gapc_pairing_t;
```

4. 1. 1. 26 绑定

```
typedef enum _esble_gapc_bond_enum
{
ESBLE_GAPC_PAIRING_REQ,               /*配对请求*/
ESBLE_GAPC_PAIRING_RSP,               /*配对请求响应*/
ESBLE_GAPC_PAIRING_SUCCEED,           /*配对完成*/
ESBLE_GAPC_PAIRING_FAILED,            /*配对失败*/
ESBLE_GAPC_TK_EXCH,                   /*检索配对临时秘钥*/
ESBLE_GAPC_IRK_EXCH,                  /*IRK 交换*/
```

```
    ESBLE_GAPC_CSRK_EXCH,                /*CSRK 交换*/
    ESBLE_GAPC_LTK_EXCH,                 /*LTK 交换*/
    ESBLE_GAPC_REPEATED_ATTEMPT,         /*重复配对请求问题*/
    ESBLE_GAPC_OOB_EXCH,                 /*OOB 交换*/
    ESBLE_GAPC_NC_EXCH                   /*NC 交换*/
}esble_gapc_bond_enum;
typedef union _esble_gapc_bond_cfm_data_t
{
 esble_gapc_pairing_t   pairing_feat;  /*配对功能*/
 esble_gapc_ltk_t       ltk;           /*LTK*/
 esble_gap_sec_key_t    csrk;          /*CSRK*/
 esble_gap_sec_key_t    tk;            /*TK*/
 esble_gapc_irk_t       irk;           /*IRK*/
 esble_gapc_oob_t       oob;           /*来自对端设备的 OOB 确认和随机数*/
} esble_gapc_bond_cfm_data_t;
typedef struct _esble_gapc_bond_auth_t
{
 uint8_t        info;                  /*认证消息*/
 bool           ltk_present;           /*配对时 LTK 交换*/
}esble_gapc_bond_auth_t;
typedef struct _esble_gapc_nc_t
{
 uint8_t        value[4];
}esble_gapc_nc_t;
typedef union _esble_gapc_bond_data_t
{
 esble_gapc_bond_auth_t     auth;       /*认证信息*/
 uint8_t                    reason;     /*配对失败原因*/
 esble_gapc_ltk_t           ltk;        /*LTK*/
 esble_gapc_irk_t           irk;        /*IRK*/
 esble_gap_sec_key_t        csrk;       /*CSRK*/
}esble_gapc_bond_data_t;
```

```
typedef union _esble_gapc_bond_req_data_t
{
uint8_t             auth_req;       /*认证等级*/
uint8_t             key_size;       /* LTK 大小*/
uint8_t             tk_type;        /*TK*/
esble_gapc_oob_t    oob_data;       /*OOB 数据*/
esble_gapc_nc_t     nc_data;        /*NC 数据*/
}esble_gapc_bond_req_data_t;
```

OOB 数据详见 4. 1. 1. 29

4. 1. 1. 27 IRK

```
typedef struct _esble_gapc_irk_t
{
esble_gap_sec_key_t     irk;    /*IRK*/
esble_gap_bdaddr_t      addr;   /*设备地址*/
}esble_gapc_irk_t;
```

4. 1. 1. 28 设备地址

```
#define ESBLE_GAP_BD_ADDR_LEN       (6)     /* BD address length*/
typedef struct _esble_bd_addr_t
{
uint8_t   addr[ESBLE_GAP_BD_ADDR_LEN];      /*6 字节的设备地址*/
} esble_bd_addr_t;
typedef struct _esble_gap_bdaddr_t
{
esble_bd_addr_t   addr;         /*设备地址*/
uint8_t           addr_type;    /*地址类型，0=public/1=private random*/
} esble_gap_bdaddr_t;
```

4. 1. 1. 29 OOB

```
#define ESBLE_GAP_KEY_LEN           (16)                /* Key length*/
typedef struct _esble_gapc_oob_t
{
uint8_t       conf[ESBLE_GAP_KEY_LEN];          /*确认值*/
```

```
uint8_t         rand[ESBLE_GAP_KEY_LEN];            /*随机数*/
}esble_gapc_oob_t;
```

4.1.2 GAP API命令

4.1.2.1 复位

esble_gapm_reset()

参数说明：

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

函数描述：对设备进行复位操作。

4.1.2.2 设备配置

esble_gapm_set_dev_conf(esble_gapm_set_dev_config_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gapm_set_dev_config_cmd_t
{
uint8_t         role;       /*设备角色：Central、Peripheral、Observer、Broadcaster*/
uint16_t        renew_dur; /*重新生成私密地址的周期，单位为 s，取值范围为[150,41400]*/
esble_bd_addr_t   addr; /*  当使能 privacy 时提供本机的静态私有随机地址*/
esble_gap_sec_key_t   irk; /*  设置本机的 IRK*/
uint8_t         privacy_cfg; /*私有配置设置*/
uint8_t         pairing_mode; /*配对模式设置*/
uint16_t        gap_start_hdl; /* GAP 服务起始句柄*/
uint16_t        gatt_start_hdl; /* GATT 服务起始句柄*/
uint16_t        att_cfg;    /*属性数据库配置*/
uint16_t        sugg_max_tx_octets; /*控制器发送载荷最大字节数推荐值*/
uint16_t        sugg_max_tx_time; /*控制器发送一帧数据最大时间推荐值(ms)*/
uint16_t        max_mtu;      /*设备支持的最大 MTU*/
uint16_t        max_mps;      /*设备支持的最大 MPS*/
uint8_t         max_nb_lecb;/*最大可建立的 LECB 连接个数*/
uint16_t        audio_cfg;      /*不使用*/
uint8_t         tx_pref_phy;   /*发送数据时优先使用的 LE PHY*/
uint8_t         rx_pref_phy;   /*接收数据时优先使用的 LE PHY*/
```

uint16_t tx_path_comp; /*RF TX 路径损耗补偿，单位是 0.1dB，取值范围为[0,1280]*/

uint16_t rx_path_comp; /*RF TX 路径损耗补偿，单位是 0.1dB，取值范围为[0,1280]*/

} esble_gapm_set_dev_config_cmd_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

函数描述：

用于设置设备的配置信息如设备角色等

设备角色

管理设备地址类型：公开、私有

设置用于生成 RAL 地址的 IRK

设置本机内部 GAP/GATT 服务的起始句柄

设置本机内部 GAP 数据库中 Name 和 Appearance 指定的写操作权限

管理是否存在某些属性

配置数据长度的扩展特性

note:

在执行完软件复位命令后，必须执行一次此命令进行设备配置

由于系统不支持动态角色切换，因此此命令仅能在无连接时调用。

4. 1. 2. 3 设置PAL

esble_gapm_set_pal(esble_gapm_set_pal_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_set_pal_cmd_t

{

uint8_t size; /*添加到 PAL 的周期性广播设备信息的个数，若为 0 则将 PAL 清除*/

esble_gapm_period_adv_addr_cfg_t *pal_info; /*待添加的周期性广播设备信息*/

} esble_gapm_set_pal_cmd_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

函数描述：设置 PAL 的信息，当前 PAL 中的信息会被清除并使用新的设置值。

Note：若添加的设备信息个数大于 PAL 的列表大小，则 esapp_gapm_cmp 返回 GAP_ERR_INSUFF_RESOURCES(0X4B)错误码；

每个设备信息仅能在列表中出现一次，若重复设置则 esapp_gapm_cmp 返回

GAP_ERR_INVALID_PARAM(0X40)错误码。

4. 1. 2. 4 设置RAL

esble_gapm_set_ral(esble_gapm_set_ral_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_set_ral_cmd_t

{

uint8_t size; /*添加到 RAL 的可解析地址设备信息的个数，若为 0 则将 RAL 清除*/

esble_gap_ral_dev_info_t *ral_info; /*待添加的可解析地址设备信息*/

} esble_gapm_set_ral_cmd_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

函数描述：设置 RAL 的信息，当前 RAL 中的信息会被清除并使用新的设置值。

Note：若添加的设备信息个数大于 RAL 的列表大小，则 esapp_gapm_cmp 返回 GAP_ERR_INSUFF_RESOURCES(0X4B)错误码；

每个设备信息仅能在列表中出现一次，若重复设置则 esapp_gapm_cmp 返回 GAP_ERR_INVALID_PARAM(0X40)错误码。

4. 1. 2. 5 读取本机地址

esble_gapm_get_ral_loc_addr(esble_gapm_get_ral_loc_addr_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_get_ral_loc_addr_cmd_t

{

esble_gap_bdaddr_t loc_identity; /*本机设备地址信息*/

} esble_gapm_get_ral_loc_addr_cmd_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_ral_loc_addr_ind：返回读取到的地址信息

函数描述：读取指定地址在 RAL 本机列表中的数据信息。

4. 1. 2. 6 读取对端设备地址

esble_gapm_get_ral_peer_addr(esble_gapm_get_ral_peer_addr_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_get_ral_peer_addr_cmd_t

{

esble_gap_bdaddr_t peer_identity; /* 对端设备地址*/

} esble_gapm_get_ral_peer_addr_cmd_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_ral_peer_addr_ind：返回读取到的地址信息

函数描述：读取指定地址在 RAL 对端设备列表中的数据信息。

4. 1. 2. 7 读取设置的广播个数

esble_gapm_get_nb_adv_sets(void)

参数说明：无

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_nb_adv_sets_ind：返回读取到的设置的广播个数

函数描述：读取本机设置的广播的个数信息。

4. 1. 2. 8 读取设置的广播发送功率

esble_gapm_get_dev_adv_tx_power_sets(void)

参数说明：

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_dev_adv_tx_power_ind：返回读取到的广播发送功率值

函数描述：读取本机设置的广播发送功率值。

4. 1. 2. 9 读取控制器支持的最大广播数据长度

esble_gapm_get_max_le_adv_data_len(void)

参数说明：

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_max_le_adv_data_len_ind：返回读取到控制器所支持的最大广播数据长度

函数描述：读取本机控制器支持的最大广播数据长度。

4. 1. 2. 10 读取PAL列表大小

esble_gapm_get_pal_size(void)

参数说明：

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_list_size_ind：返回读取到 PAL 列表大小

函数描述：读取本机 PAL 列表的大小。

4. 1. 2. 11 读取设备地址

esble_gapm_get_dev_bdaddr(void)

参数说明：

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_dev_bdaddr_ind：返回读取到设备地址信息

函数描述：读取本机设备地址信息。

4. 1. 2. 12 读取RAL列表大小

esble_gapm_get_ral_size(void)

参数说明：

无

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_list_size_ind：返回读取到 RAL 列表大小

函数描述：读取本机 RAL 列表大小。

4. 1. 2. 13 地址解析

esble_gapm_resolv_addr(esble_gapm_resolv_addr_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gapm_resolv_addr_cmd_t
{
uint8_t                 nb_key;     /*提供的 IRK 数量（需大于 0）*/
esble_bd_addr_t         addr;       /*待解析的可解析随机地址*/
esble_gap_sec_key_t     *irk;       /*用于地址解析的 IRK 序列*/
} esble_gapm_resolv_addr_cmd_t;
```

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_addr_solved_ind：返回解析地址所使用的 IRK 信息

函数描述：发送解析地址命令。

4. 1. 2. 14 生成随机地址

esble_gapm_gen_rand_addr(esble_gapm_gen_rand_addr_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gapm_gen_rand_addr_cmd_t
{
uint8_t        prand[ESBLE_GAP_ADDR_PRAND_LEN];     /*存储地址的随机值部分*/
uint8_t        rnd_type;                            /*随机地址类型*/
} esble_gapm_gen_rand_addr_cmd_t;
```

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_dev_bdaddr_ind：返回本机设备地址

函数描述：生成随机地址。

4. 1. 2. 15 生成随机数

esble_gapm_gen_rand_nb(void)

参数说明：

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_gen_rand_nb_ind：返回生成的 8 字节随机值

函数描述：生成随机数。

4. 1. 2. 16 数据加密

esble_gapm_use_enc_block(esble_gapm_use_enc_block_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gapm_use_enc_block_cmd_t
{
uint8_t        operand_1[ESBLE_GAP_KEY_LEN];        /*16 字节操作数 1*/
uint8_t        operand_2[ESBLE_GAP_KEY_LEN];        /*16 字节操作数 2*/
} esble_gapm_use_enc_block_cmd_t;
```

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_use_enc_block_ind：返回生成的 16 字节加密结果

函数描述：使用加密模块进行数据加密。

4. 1. 2. 17 设置IRK

esble_gapm_set_irk(esble_gapm_set_irk_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_set_irk_cmd_t

{

uint8_t key[ESBLE_GAP_KEY_LEN]; /*16 字节用于生成可解析随机地址的 IRK*/

} esble_gapm_set_irk_cmd_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

函数描述：使用新的 IRK 数据代替当前 IRK 数据用于生成可解析随机地址。

4. 1. 2. 18 生成DHKEY

esble_gapm_gen_dh_key(esble_gapm_gen_dh_key_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_gen_dh_key_cmd_t

{

uint8_t operand_1[ESBLE_GAP_P256_KEY_LEN]; /*32 字节操作数 1*/

uint8_t operand_2[ESBLE_GAP_P256_KEY_LEN]; /*32 字节操作数 2*/

} esble_gapm_gen_dh_key_cmd_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_gen_dh_key_ind：返回生成的 32 字节 DHKEY

函数描述：使用给定的输入参数值生成 DHKEY。

4. 1. 2. 19 读取PUBKEY

esble_gapm_get_pub_key(esble_gapm_get_pub_key_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_get_pub_key_cmd_t

{

uint8_t renew; /*1：重新生成 PUBKEY 0：读取当前 PUBKEY*/

} esble_gapm_get_pub_key_cmd_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_pub_key_ind：返回读取的 PUBKEY

函数描述：读取 PUBKEY。

4. 1. 2. 20 创建广播

esble_gapm_create_adv(esble_gapm_create_adv_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gapm_create_adv_cmd_t
{
uint8_t                          own_addr_type;    /*本机设备地址类型*/
esble_gapm_create_adv_param_t    adv_param;        /*广播参数*/
} esble_gapm_create_adv_cmd_t;
```

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gamp_create_adv_ind：返回创建的广播事件的索引值和广播事件选定的发送功率值

函数描述：创建广播事件。

Note：可创建的事件的总数有限制，若超过限制值则 esapp_gapm_cmp()事件会返回错误码 GAP_ERR_INSUFF_RESOURCES(0X4B)。

4. 1. 2. 21 设置广播数据

esble_gapm_set_adv_data(esble_gapm_adv_data_t *cmd)

参数说明：

```
typedef struct _esble_gapm_adv_data_t
{
uint8_t        adv_idx;          /*广播事件索引值*/
uint16_t  length;                /*广播数据长度*/
uint8_t        *data;            /*广播数据*/
} esble_gapm_adv_data_t;
```

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成。

函数描述：设置广播数据。

4. 1. 2. 22 设置周期性广播数据

esble_gapm_set_period_adv_data(esble_gapm_set_period_adv_data_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_set_period_adv_data_cmd_t

{

uint8_t adv_idx; /*广播索引*/

uint16_t length; /*数据长度*/

uint8_t *data; /*数据*/

} esble_gapm_set_period_adv_data_cmd_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成。

函数描述：设置周期性广播数据。

4. 1. 2. 23 设置扫描响应数据

esble_gapm_set_scan_rsp_data(esble_gapm_adv_data_t *cmd)

参数说明：

typedef struct _esble_gapm_adv_data_t

{

uint8_t adv_idx; /*广播事件索引值*/

uint16_t length; /*广播数据长度*/

uint8_t *data; /*广播数据*/

} esble_gapm_adv_data_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

函数描述：设置扫描响应数据。

4. 1. 2. 24 启动开始广播

esble_gapm_adv_start(esble_gapm_adv_start_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_adv_start_cmd_t

{

uint8_t adv_idx; /*广播事件索引*/

uint16_t duration; /*广播事件持续时间，单位是 10ms，0 表示无时间限制*/

uint8_t max_adv_evt; /*extended 广播持续的最大事件数量*/

} esble_gapm_adv_start_cmd_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_start_adv_ind：返回开始的广播索引值和状态

esapp_gapm_scan_request_ind：若使能扫描请求通知功能，则返回接收到的扫描请求给应用

函数描述：设置开始广播

Note：多个广播事件可并行同时开始。

4. 1. 2. 25 停止广播

esble_gapm_adv_stop(esble_gapm_adv_stop_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_adv_stop_cmd_t

{

uint8_t adv_idx; /*广播事件索引*/

} esble_gapm_adv_stop_cmd_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_stop_adv_ind：返回停止的广播事件的索引、停止的原因，若是 PER 则同时返回 PER ADV 是否停止。

函数描述：设置停止广播。

Note：若请求的事件索引不存在，则 esapp_gapm_cmp()事件返回 GAP_ERR_INVALID_PARAM (0X40)错误码，若事件索引存在但未开始，则 esapp_gapm_cmp()事件返回 GAP_ERR_COMMAND_DISALLOWED (0X43)错误码，且不返回 esapp_gapm_stop_adv_ind 事件。

4. 1. 2. 26 创建扫描

esble_gapm_create_scan(esble_gapm_create_scan_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_create_scan_cmd_t

{

uint8_t own_addr_type; /*本机设备地址类型*/

} esble_gapm_create_scan_cmd_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gamp_create_scan_ind：返回创建的扫描事件的索引

函数描述：设置创建扫描事件

Note：可创建的事件的总数有限制，若超过限制值则 esapp_gapm_cmp()事件会返回错误码 GAP_ERR_INSUFF_RESOURCES(0X4B)。

4. 1. 2. 27 开始扫描

esble_gapm_scan_start(esble_gapm_scan_start_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gapm_scan_start_cmd_t
{
uint8_t                 scan_idx;       /*扫描事件索引*/
esble_gapm_scan_param_t scan_param;     /*扫描参数*/
} esble_gapm_scan_start_cmd_t;
```

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_start_scan_ind：返回扫描事件的索引值和状态函数描述：设置开始扫描事件。

4. 1. 2. 28 停止扫描

esble_gapm_scan_stop(esble_gapm_scan_stop_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gapm_scan_stop_cmd_t
{
uint8_t     scan_idx;           /*扫描事件索引*/
} esble_gapm_scan_stop_cmd_t;
```

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_stop_scan_ind：返回停止的广播事件的索引、停止的原因

函数描述：设置停止扫描事件

Note：若请求的事件索引不存在，则 esapp_gapm_cmp()事件返回 GAP_ERR_INVALID_PARAM (0X40)错误码，若事件索引存在但未开始，则 esapp_gapm_cmp()事件返回 GAP_ERR_COMMAND_DISALLOWED (0X43)错误码，且不返回 esapp_gapm_stop_adv_ind 事件。

4. 1. 2. 29 创建发起连接

esble_gapm_create_init(esble_gapm_create_init_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_create_init_cmd_t

{

uint8_t own_addr_type; /*本机设备地址类型*/

} esble_gapm_create_init_cmd_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gamp_create_init_ind：返回创建的发起连接事件的索引

函数描述：设置创建发起连接事件

Note：可创建的事件的总数有限制，若超过限制值则 esapp_gapm_cmp()事件会返回错误码 GAP_ERR_INSUFF_RESOURCES(0X4B)。

4. 1. 2. 30 开始发起连接

esble_gapm_init_start(esble_gapm_init_start_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_init_start_cmd_t

{

uint8_t own_addr_type; /*本机设备地址类型*/

esble_gapm_init_param_t init_param; /*发起连接参数*/

} esble_gapm_init_start_cmd_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_start_init_ind：返回发起连接事件的索引值和状态

函数描述：设置开始发起连接事件。

4. 1. 2. 31 停止发起连接

esble_gapm_init_stop(esble_gapm_init_stop_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_init_stop_cmd_t

{

uint8_t init_idx; /*发起连接事件索引*/

} esble_gapm_init_stop_cmd_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_stop_init_ind：返回停止的发起连接事件的索引、停止的原因

函数描述：设置停止发起连接事件

Note：若请求的事件索引不存在，则 esapp_gapm_cmp()事件返回 GAP_ERR_INVALID_PARAM (0X40)错误码，若事件索引存在但未开始，则 esapp_gapm_cmp()事件返回 GAP_ERR_COMMAND_DISALLOWED (0X43)错误码，且不返回 esapp_gapm_stop_adv_ind 事件。

4. 1. 2. 32 创建周期同步

esble_gapm_create_period_sync(esble_gapm_create_period_sync_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_create_period_sync_cmd_t

{

uint8_t own_addr_type; /*本机设备地址类型*/

} esble_gapm_create_period_sync_cmd_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gamp_create_per_sync_ind：返回创建的周期同步事件的索引

函数描述：设置创建周期同步事件

Note：可创建的事件的总数有限制，若超过限制值则 esapp_gapm_cmp()事件会返回错误码 GAP_ERR_INSUFF_RESOURCES(0X4B)。

4. 1. 2. 33 开始周期同步

esble_gapm_period_sync_start(esble_gapm_period_sync_start_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_period_sync_start_cmd_t

{

uint8_t per_sync_idx; /*周期同步事件索引*/

uint16_t skip;/*在成功接收到广播数据后可忽略的最大广播事件数，最大值为 499*/
uint16_t sync_to; /*对周期广播同步的超时时间，单位是 10ms，取值范围为[100ms,163.84s]*/
uint8_t type; /*周期同步事件类型*/

uint8_t rsvd; /*保留*/

esble_gap_bdaddr_t addr; /*广播地址信息*/

uint8_t adv_sid; /*广播 SID*/

} esble_gapm_period_sync_start_cmd_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_start_per_sync_ind：返回周期同步事件的索引值和状态

esapp_gapm_sync_established_ind：接收到周期性广播数据后返回同步建立的状态

函数描述：设置开始周期同步事件

Note：周期同步事件必须与扫描时间同时开始。

4. 1. 2. 34 停止周期同步

esble_gapm_period_sync_stop(esble_gapm_period_sync_stop_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_period_sync_stop_cmd_t

{

uint8_t per_sync_idx; /*同步事件索引*/

} esble_gapm_period_sync_stop_cmd_t;

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_stop_per_sync_ind：返回停止的发起连接事件的索引、停止的原因

函数描述：设置停止周期同步事件

Note：若请求的事件索引不存在，则 esapp_gapm_cmp()事件返回 GAP_ERR_INVALID_PARAM (0X40)错误码，若事件索引存在但未开始，则 esapp_gapm_cmp()事件返回 GAP_ERR_COMMAND_DISALLOWED (0X43)错误码，且不返回 esapp_gapm_stop_adv_ind 事件。

4. 1. 2. 35 停止所有事件

esble_gapm_stop_all_acts(void)

参数说明：

函数响应：

esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

esapp_gapm_stop_all_ind：返回已停止所有事件指示给 application

函数描述：停止所有事件。

4. 1. 2. 36 删除事件

esble_gapm_delete_actv(esble_gapm_delet_actv_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_delet_actv_cmd_t

{

uint8_t actv_idx; /*事件索引*/

} esble_gapm_delet_actv_cmd_t;

函数响应：esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

函数描述：删除指定事件

Note：若指定的事件没有停止则 esapp_gapm_cmp()事件返回 GAP_ERR_COMMAND_DISALLOWED(0X43)错误码。

4. 1. 2. 37 删除所有事件

esble_gapm_delete_all_actv(void)

参数说明：无

函数响应：esapp_gapm_cmp()：命令执行完成后 BLE STACK 返回命令完成

函数描述：删除所有当前事件

Note：若当前事件至少有一个没有停止则 esapp_gapm_cmp()事件返回 GAP_ERR_COMMAND_DISALLOWED(0X43)错误码。

4. 1. 2. 38 设置信道表

esble_gapm_set_channel_map(esble_gapm_set_channel_map_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_set_channel_map_cmd_t

{

esble_le_chnl_map_t chmap; /*信道表数据*/

}esble_gapm_set_channel_map_cmd_t;

函数响应：

函数描述：设置信道表，只有中央设备才能更改信道表。

4. 1. 2. 39 获取设备版本信息

esble_gapm_get_dev_version(void)

参数说明：无

函数响应：

函数描述：获取本地设备的版本信息。

4. 1. 2. 40 获取设备发送功率

esble_gapm_get_dev_tx_pwr(void)

参数说明：无

函数响应：

函数描述：获取设备的发射功率大小。

4. 1. 2. 41 获取建议的默认LE数据长度

esble_gapm_get_suggested_dflt_le_data_len(void)

参数说明：无

函数响应：

函数描述：获取建议的默认 LE 数据长度。

4. 1. 2. 42 获取设备RF路径补偿

esble_gapm_get_dev_rf_path_comp(void)

参数说明：无

函数响应：

函数描述：获取 RF 路径补偿。

4. 1. 2. 43 获取最大LE数据长度

esble_gapm_get_max_le_data_len(void)

参数说明：无

函数响应：

函数描述：获取最大的 LE 数据长度。

4. 1. 2. 44 获取内存信息（仅调试状态下可用）

esble_gapm_dbg_get_mem_info(void)

参数说明：无

函数响应：

函数描述：获取内存信息（仅调试使用）。

4. 1. 2. 45 设置白名单

esble_gapm_set_wl(esble_gapm_list_set_wl_cmd_t *cmd)

参数说明：

typedef struct _esble_gapm_list_set_wl_cmd_t

{

```
uint8_t                 size;          /*列表数目*/
esble_gap_bdaddr_t      *wl_info;      /*白名单数据信息*/
}esble_gapm_list_set_wl_cmd_t;
```

函数响应：

函数描述：请求设置白名单。

4. 1. 2. 46 获取白名单大小

esble_gapm_get_wlist_size(void)

参数说明：无

函数响应：

函数描述：获取白名单大小。

4. 1. 2. 47 未知task消息

esble_gapm_unknown_task_msg(void)

参数说明：无

函数响应：

函数描述：未知 task 的消息。

4. 1. 2. 48 更新随机地址

esble_gapm_renew_addr(esble_gapm_renew_addr_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gapm_renew_addr_cmd_t
{
uint8_t             actv_idx;              /*Activity  索引*/
uint8_t             init_scan_actv_idx;    /*首次被扫描到的地址更新的索引*/
}esble_gapm_renew_addr_cmd_t;
```

函数响应：

函数描述：更新当前所有的随机私有地址。

4. 1. 2. 49 开始测试模式接收

esble_gapm_le_test_rx_start(esble_gapm_le_test_mode_ctrl_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gapm_le_test_mode_ctrl_cmd_t
{
```

```
uint8_t         channel;          /*信道号*/
uint8_t         tx_data_length;   /*数据长度，范围：0x00-0xFF*/
uint8_t         tx_pkt_payload;   /*包载荷*/
uint8_t         phy;              /*PHY 设置*/
uint8_t         modulation_idx;   /*调制方式*/
}esble_gapm_le_test_mode_ctrl_cmd_t;
```

tx_pkt_payload 详见 4. 1. 1. 16，phy 详见 4. 1. 1. 17，modulation_idx 详见 4. 1. 1. 18

函数响应：

函数描述：开始 RX 测试模式。

4. 1. 2. 50 开始测试模式发送

esble_gapm_le_test_tx_start(esble_gapm_le_test_mode_ctrl_cmd_t *cmd)

参数说明：详见 4. 1. 2. 49

函数响应：

函数描述：开始 TX 测试模式。

4. 1. 2. 51 停止测试模式

esble_gapm_le_test_stop(esble_gapm_le_test_mode_ctrl_cmd_t *cmd)

参数说明：详见 4. 1. 2. 49

函数响应：

函数描述：停止测试模式。

4. 1. 2. 52 添加profile任务

esble_gapm_profile_task_add(esble_gapm_profile_task_add_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gapm_profile_task_add_cmd_t
{
uint8_t         sec_lvl;          /*安全等级*/
uint16_t        prf_task_id;      /* Profile 标识符*/
uint16_t        app_task;         /*应用任务*/
uint16_t        start_hdl;        /*服务启动句柄*/
uint32_t        *param;           /*初始化 profile 的 32 位的数据*/
} esble_gapm_profile_task_add_cmd_t;
```

函数响应：

函数描述：此命令用于为 profile 分配任务。

4. 1. 2. 53 平台复位

esble_gapm_plf_reset(void)

参数说明：无

函数响应：

函数描述：gapm 的 platform 硬件复位。

4. 1. 2. 54 注册LEPSM

esble_gapm_lepsm_reg(esble_gapm_lepsm_reg_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gapm_lepsm_reg_cmd_t
{
 uint16_t  le_psm;                /*LE Protocol/Service Multiplexer*/
 uint16_t  app_task;              /*APP  任务号*/
 uint8_t          sec_lvl;        /*安全等级*/
} esble_gapm_lepsm_reg_cmd_t;
```

函数响应：

函数描述：此命令用于注册一个协议/服务复用器标识符，允许对端设备以此创建 LE 信令连接。

4. 1. 2. 55 取消注册LEPSM

esble_gapm_lepsm_unreg(esble_gapm_lepsm_unreg_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gapm_lepsm_unreg_cmd_t
{
 uint16_t  le_psm;              /*LE Protocol/Service Multiplexer*/
} esble_gapm_lepsm_unreg_cmd_t;
```

函数响应：

函数描述：此命令用于注销设备中的 LE 协议/服务复用器标识符。

4. 1. 2. 56 连接确认

esble_gapc_conn_cfm(esble_gapc_conn_cfm_t *cfm)

参数说明：

```
typedef struct _esble_gapc_conn_cfm_t
```

```
{
uint8_t                 conidx;                 /*连接索引*/
esble_gap_sec_key_t     lcsrk;                  /*本地 CSRK 的值*/
uint32_t                lsign_counter;          /*本地签名计数器值*/
esble_gap_sec_key_t     rcsrk;                  /*远程 CSRK 的值*/
uint32_t                rsign_counter;          /*远程签名计数器值*/
uint8_t                 auth;                   /*身份验证*/
uint8_t                 svc_changed_ind_enable; /*启用服务更改指示*/
bool                    ltk_present;            /*配对时交换 LTK*/
} esble_gapc_conn_cfm_t;
```

函数响应：无

函数描述：设置链路安全配置和绑定数据。

4. 1. 2. 57 断开连接

esble_gapc_disconn(esble_gapc_disconn_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gapc_disconn_cmd_t
{
uint8_t con_idx;                /*连接索引号*/
uint8_t reason;                 /*断开连接原因*/
} esble_gapc_disconn_cmd_t;
```

函数响应：

函数描述：请求断开指定索引号的连接，可以由主或从发出请求，断开原因应是有效的原因。

4. 1. 2. 58 从设备延迟首选参数设置

esble_gapc_set_pref_slv_latency(esble_gapc_set_pref_slv_latency_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gapc_set_pref_slv_latency_cmd_t
{
uint8_t         conidx;         /*连接索引*/
uint16_t        latency;        /*从设备连接延迟*/
} esble_gapc_set_pref_slv_latency_cmd_t;
```

函数响应：

函数描述：在主机允许的范围内，设置从设备的连接延迟。

4. 1. 2. 59 更新连接参数

esble_gapc_param_upd(esble_gapc_param_upd_cmd_t *cmd)

参数说明：

typedef struct _esble_gapc_param_update_cmd_t

{

uint8_t conidx; /*连接索引*/

uint8_t pkt_id; /*内部参数用来管理内部 l2cap 信号的 packet id*/

uint16_t intv_min; /*最小连接间隔 N*1.25ms*/

uint16_t intv_max; /*最大连接间隔 N*1.25ms*/

uint16_t latency; /*握手延迟*/

uint16_t time_out; /*链路超时参数 N*10ms*/

uint16_t ce_len_min; /*最小连接事件时长 N*0.625ms*/

uint16_t ce_len_max; /*最小连接事件时长 N*0.625ms*/

} esble_gapc_param_upd_cmd_t;;

函数响应：

函数描述：连接参数更新，可以由主或从发出请求，对于主机，将立即应用新的连接参数，对于从机，连接更新请求将通过 l2cap 发送给主机，主机决定接受或拒绝，如果主机端接受新参数，新参数将被使用。从机发送请求时，会启动一个 30s 的计时器，如果超过 30s，还没有收到主机的响应，将自动断开连接。

4. 1. 2. 60 参数更新确认

esble_gapc_param_upd_cfm(esble_gapc_param_upd_cfm_t *cfm)

参数说明：

typedef struct _esble_gapc_param_update_cfm_t

{

uint8_t conidx; /*连接索引*/

bool accept; /*1：接受从机的连接参数 0：拒绝从机的连接参数*/

uint16_t ce_len_min; /*连接事件持续时间最小值*/

uint16_t ce_len_max; /*连接事件持续时间最大值*/

} esble_gapc_param_upd_cfm_t;

函数响应：

函数描述：主机应用端确认是否接受从机发送的连接参数更新

4. 1. 2. 61 设备信息确认

esble_gapc_dev_info_cfm(esble_gapc_dev_info_cfm_t *cfm)

参数说明：

```
typedef struct _esble_gapc_dev_info_cfm_t
{
uint8_t         conidx;                 /*连接索引*/
uint8_t         req;                    /*请求信息*/
union
{
    esble_gapc_dev_name_t    name;              /*设备名称*/
    uint16_t                 appearance;        /*设备外观*/
    esble_gapc_slv_pref_t    slv_pref_params;   /*从设备首选参数*/
    uint8_t                  ctl_addr_resol;    /*地址解析*/
} info;
} esble_gapc_dev_info_cfm_t;
```

函数响应：

函数描述：将请求信息发送到对端设备。

4. 1. 2. 62 读取RSSI

esble_gapc_get_con_rssi(void)

参数说明：无

函数响应：

函数描述：获取当前连接的 RSSI 值。

4. 1. 2. 63 读取信道图

esble_gapc_get_con_cnl_map(void)

参数说明：无

函数响应：

函数描述：检索连接通道。

4. 1. 2. 64 读取是否支持地址解析

esble_gapc_get_addr_resol_supp(void)

参数说明：无

函数响应：

函数描述：检查是否支持 central 地址解析

4. 1. 2. 65 读取对端设备名称

esble_gapc_get_peer_name(void)

参数说明：无

函数响应：

函数描述：检索对端设备名称

4. 1. 2. 66 读取对端设备版本信息

esble_gapc_get_peer_version(void)

参数说明：无

函数响应：

函数描述：检索对端设备版本信息

4. 1. 2. 67 读取对端设备特性

esble_gapc_get_peer_features(void)

参数说明：无

函数响应：

函数描述：检索对端设备特性

4. 1. 2. 68 读取对端设备外观信息

esble_gapc_get_peer_appearance(void)

参数说明：无

函数响应：

函数描述：获取对端设备外观信息

4. 1. 2. 69 读取对端设备首选参数

esble_gapc_get_peer_slv_pref_params(void)

参数说明：无

函数响应：

函数描述：获取对端从设备首选参数

4. 1. 2. 70 读取当前连接所使用的PHY

esble_gapc_get_phy(void)

参数说明：无

函数响应：

函数描述：获取当前连接的物理层信息

4. 1. 2. 71 读取信道选择算法

esble_gapc_get_chan_sel_algo(void)

参数说明：无

函数响应：

函数描述：获取链路当前使用的信道选择算法

4. 1. 2. 72 读取PING超时时间

esble_gapc_get_le_ping_to(void)

参数说明：无

函数响应：

函数描述：获取 LE ping 超时信息

4. 1. 2. 73 设置设备信息确认

esble_gapc_set_dev_info_cfm(esble_gapc_set_dev_info_cfm_t *cfm)

参数说明：

typedef struct _esble_gapc_set_dev_info_cfm_t

{

uint8_t	conidx;	/*连接索引*/
uint8_t	req;	/*被请求的信息，设备名字，外观*/
uint8_t	status;	/*请求是否被接受*/

} esble_gapc_set_dev_info_cfm_t;

函数响应：

函数描述：本地设备是否接受设备信息更改的确认。

4. 1. 2. 74 设置PING的超时时间

esble_gapc_set_le_ping_to(esble_gapc_set_le_ping_to_cmd_t *cmd)

参数说明：

typedef struct _esble_gapc_set_le_ping_to_cmd_t

{

uint8_t	conidx;	/*连接索引*/
uint16_t	timeout;	/*验证 payload 超时 N*10ms*/

} esble_gapc_set_le_ping_to_cmd_t;

函数响应：

函数描述：更改当前连接的底层 LE Ping 认证 payload 超时时间。

4. 1. 2. 75 设置PHY

esble_gapc_set_phy(esble_gapc_set_phy_cmd_t *cmd)

参数说明：

typedef struct _esble_gapc_set_phy_cmd_t

{

uint8_t	conidx;	/*连接索引*/
uint8_t	tx_phy;	/*首选发射 phy*/
uint8_t	rx_phy;	/*首选接收 phy*/
uint8_t	phy_opt;	/*编码 PHY 的首选方案*/

} esble_gapc_set_phy_cmd_t;

函数响应：

函数描述：设置当前活动链接的首选 PHY，它还可用于与对端设备的协商更新 PHY。

4. 1. 2. 76 设置载荷大小参数

esble_gapc_set_le_pkt_size(esble_gapc_set_le_pkt_size_cmd_t *cmd)

参数说明：

typedef struct _esble_gapc_set_le_pkt_size_cmd_t

{

uint8_t	conidx;	/*连接索引*/
uint16_t	tx_octets;	/*最大有效载荷*/
uint16_t	tx_time;	/*最大传输 PDU 时间*/

} esble_gapc_set_le_pkt_size_cmd_t;

函数响应：

函数描述：更改控制器中数据长度扩展的命令。

4. 1. 2. 77 连接加密

esble_gapc_encrypt(esble_gapc_encrypt_cmd_t *cmd)

参数说明：

typedef struct _esble_gapc_encrypt_cmd_t

{

```
uint8_t          conidx;      /*连接索引*/
esble_gapc_ltk_t ltk;         /*LTK 信息*/
} esble_gapc_encrypt_cmd_t;
```

函数响应：

函数描述：由连接的主设备发起加密。

4. 1. 2. 78 加密确认

esble_gapc_encrypt_cfm(esble_gapc_encrypt_cfm_t *cfm)

参数说明：

```
typedef struct _esble_gapc_encrypt_cfm_t
{
uint8_t          conidx;      /*连接索引*/
uint8_t          found;       /*指示是否找到了对端设备的 LTK*/
esble_gap_sec_key_t ltk;      /*LTK*/
uint8_t          key_size;    /*LTK 秘钥大小*/
} esble_gapc_encrypt_cfm_t;
```

函数响应：

函数描述：在收到加密请求指示后发送此确认消息，此消息能用来是否已找到加密秘钥，如果是，则应提供 LTK 及其长度大小。

4. 1. 2. 79 安全请求

esble_gapc_sec(esble_gapc_sec_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gapc_sec_cmd_t
{
uint8_t          conidx;      /*连接索引*/
uint8_t          auth;        /*身份验证级别*/
} esble_gapc_sec_cmd_t;
```

函数响应：

函数描述：只能由连接的从机发出启动安全请求过程，包含当前设备请求的身份验证级别。

4. 1. 2. 80 按键通知

esble_gapc_key_press_notif(esble_gapc_key_press_notif_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gapc_key_press_notif_cmd_t
{
uint8_t             conidx;                 /*连接索引*/
uint8_t             notification_type;      /*通知类型*/
} esble_gapc_key_press_notif_cmd_t;
```

函数响应：

函数描述：当输入或删除数字时，防止超时，向对端设备发送一个 keypress 通知。

4. 1. 2. 81 绑定

```
esble_gapc_bond(esble_gapc_bond_cmd_t *cmd)
```

参数说明：

```
typedef struct _esble_gapc_bond_cmd_t
{
uint8_t                 conidx;     /*连接索引*/
esble_gapc_pairing_t    pairing;    /*配对信息*/
} esble_gapc_bond_cmd_t;
```

函数响应：

函数描述：连接的主设备可以请求绑定规程，包含发起者的配对请求。

4. 1. 2. 82 绑定确认

```
esble_gapc_bond_cfm(esble_gapc_bond_cfm_t *cfm)
```

参数说明：

```
typedef struct _esble_gapc_bond_cfm_t
{
uint8_t                         conidx;     /*连接索引*/
uint8_t                         request;    /*绑定请求类型*/
uint8_t                         accept;     /*0x01 接受请求，0x00 拒绝请求*/
esble_gapc_bond_cfm_data_t      data;       /*绑定规程信息数据*/
} esble_gapc_bond_cfm_t;
```

函数响应：

函数描述：收到绑定请求指示消息后发送的确认消息，包含从设备配对信息，配对的临时秘钥 TK，秘钥交换时提供秘钥给对端设备。

4. 1. 2. 83 数据签名

esble_gapc_sign_packet(esble_gapc_sign_cmd_t *cmd)

参数说明：

typedef struct _esble_gapc_sign_cmd_t

{

uint8_t conidx; /*连接索引*/

uint16_t byte_len; /*Data PDU 长度（字节）*/

uint8_t *msg; /*Data PDU + SignCounter*/

} esble_gapc_sign_cmd_t;

函数响应：

函数描述：属性 packet 的签名

4. 1. 2. 84 签名验证

esble_gapc_sign_check(esble_gapc_sign_cmd_t *cmd)

参数说明：

typedef struct _esble_gapc_sign_cmd_t

{

uint8_t conidx; /*连接索引*/

uint16_t byte_len; /*Data PDU 长度（字节）*/

uint8_t *msg; /*Data PDU + SignCounter + MAC*/

} esble_gapc_sign_cmd_t;

函数响应：

函数描述：签名验证。

4. 1. 3 GAP 上行事件

4. 1. 3. 1 GAP事件完成

esapp_gapm_cmp(const esble_gapm_cmp_t *evt)

参数说明：

typedef struct _esble_gapm_cmp_t

{

esble_gapm_cmd_enum cmd; /*下行命令 opcode*/

uint8_t status; /*命令执行状态*/

} esble_gapm_cmp_t;

函数响应：无

函数描述：所有的 GAPM 下行命令完成后都会返回此事件。

4. 1. 3. 2 读取本机可解析地址指示

esapp_gapm_ral_loc_addr_ind(const esble_gapm_ral_addr_ind_t *ind)

参数说明：

```
typedef struct _esble_gapm_ral_addr_ind_t{
esble_gap_bdaddr_t        addr;                /*读取的 RAL 中本机可解析地址信息*/
} esble_gapm_ral_addr_ind_t;
```

函数响应：无

函数描述：esble_gapm_get_ral_loc_addr ()下行命令触发 BLE STACK 返回此事件，将读取的 RAL 中本机可解析地址信息返回给 Application。

4. 1. 3. 3 读取对端设备可解析地址指示

esapp_gapm_ral_peer_addr_ind(const esble_gapm_ral_addr_ind_t *ind)

参数说明：

```
typedef struct _esble_gapm_ral_addr_ind_t{
esble_gap_bdaddr_t        addr;                /*读取的 RAL 中对端设备可解析地址信息*/
} esble_gapm_ral_addr_ind_t;
```

函数响应：无

函数描述：esble_gapm_get_ral_peer_addr ()下行命令触发 BLE STACK 返回此事件，将读取的 RAL 中对端设备可解析地址信息返回给 Application。

4. 1. 3. 4 读取广播设置个数指示

esapp_gapm_nb_adv_sets_ind(const esble_gapm_nb_adv_sets_ind_t *ind)

参数说明：

```
typedef struct _esble_gapm_nb_adv_sets_ind_t
{
uint8_t                nb_adv_sets;              /*有效的广播设置个数*/
} esble_gapm_nb_adv_sets_ind_t;
```

函数响应：无

函数描述：esble_gapm_get_nb_adv_sets()下行命令触发 BLE STACK 返回此事件，将读取的数量信息返回给 Application。

4.1.3.5 设备广播发送功率指示

esapp_gapm_dev_adv_tx_power_ind(const esble_gapm_dev_adv_tx_power_ind_t *ind)

参数说明：

typedef struct _esble_gapm_dev_adv_tx_power_ind_t

{

int8_t power_lvl; /*广播通道的发送功率等级值*/

} esble_gapm_dev_adv_tx_power_ind_t;

函数响应：无

函数描述：esble_gapm_get_dev_adv_tx_power_sets()下行命令触发 BLE STACK 返回此事件，将读取的广播通道的发送功率等级值返回给 Application。

4.1.3.6 最大广播数据长度指示

esapp_gapm_max_le_adv_data_len_ind(const esble_gapm_max_le_adv_data_len_ind_t *ind)

参数说明：

typedef struct _esble_gapm_max_le_adv_data_len_ind_t

{

uint16_t length; /*控制器支持的最大广播数据长度*/

} esble_gapm_max_le_adv_data_len_ind_t;

函数响应：无

函数描述：esble_gapm_get_max_le_adv_data_len()下行命令触发 BLE STACK 返回此事件，将读取的控制器支持的最大广播数据长度返回给 Application。

4.1.3.7 列表大小指示

esapp_gapm_list_size_ind(const esble_gapm_list_size_ind_t *ind)

参数说明：

typedef struct _esble_gapm_list_size_ind_t

{

uint8_t size; /*列表大小*/

} esble_gapm_list_size_ind_t;

函数响应：无

函数描述：esble_gapm_get_pal_size()下行命令触发 BLE STACK 返回此事件，将读取的 PAL 列表大小返回给 Application。

4. 1. 3. 8 设备地址指示

esapp_gapm_dev_bdaddr_ind(const esble_gapm_dev_bdaddr_ind_t *ind)

参数说明：

typedef struct _esble_gapm_dev_bdaddr_ind_t

{

esble_gap_bdaddr_t addr; /*本机设备地址信息*/

uint8_t actv_idx; /*返回值无效*/

} esble_gapm_dev_bdaddr_ind_t;

函数响应：无

函数描述：esble_gapm_get_dev_bdaddr()下行命令触发 BLE STACK 返回此事件，将读取的本机设备地址返回给 Application。

4. 1. 3. 9 地址解析指示

esapp_gapm_addr_solved_ind(const esble_gapm_addr_solved_ind_t *ind)

参数说明：

typedef struct _esble_gapm_addr_solved_ind_t

{

esble_bd_addr_t addr; /*解析完成的可解析随机地址*/

esble_gap_sec_key_t irk; /*解析地址所使用的 IRK 信息*/

} esble_gapm_addr_solved_ind_t;

函数响应：无

函数描述：esble_gapm_resolv_addr()下行命令触发 BLE STACK 返回此事件，将解析的地址和解析地址所使用的 IRK 返回给 Application。

4. 1. 3. 10 生成随机值指示

esapp_gapm_gen_rand_nb_ind(const esble_gapm_gen_rand_nb_ind_t *ind)

参数说明：

typedef struct _esble_gapm_gen_rand_nb_ind_t

{

uint8_t nb[ESBLE_GAP_RAND_NB_LEN]; /*8 字节随机值*/

} esble_gapm_gen_rand_nb_ind_t;

函数响应：无

函数描述：esble_gapm_gen_rand_nb()下行命令触发 BLE STACK 返回此事件，将生成的 8 字节随机值返回给 Application。

4. 1. 3. 11 加密数据指示

esapp_gapm_use_enc_block_ind(const esble_gapm_use_enc_block_ind_t *ind)

参数说明：

typedef struct _esble_gapm_use_enc_block_ind_t

{

uint8_t result[ESBLE_GAP_KEY_LEN]; /*16 字节加密结果*/

} esble_gapm_use_enc_block_ind_t;

函数响应：无

函数描述：esble_gapm_use_enc_block()下行命令触发 BLE STACK 返回此事件，将生成的 16 字节加密结果返回给 Application。

4. 1. 3. 12 生成DHKEY指示

esapp_gapm_gen_dh_key_ind(const esble_gapm_gen_dh_key_ind_t *ind)

参数说明：

typedef struct _esble_gapm_gen_dh_key_ind_t

{

uint8_t result[ESBLE_GAP_P256_KEY_LEN]; /*32 字节 DHKEY*/

} esble_gapm_gen_dh_key_ind_t;

函数响应：无

函数描述：esble_gapm_gen_dh_key()下行命令触发 BLE STACK 返回此事件，将生成的 32 字节 DHKEY 结果返回给 Application。

4. 1. 3. 13 公用密钥指示

esapp_gapm_pub_key_ind(const esble_gapm_pub_key_ind_t *ind)

参数说明：

typedef struct _esble_gapm_pub_key_ind_t

{

uint8_t pub_key_x[ESBLE_GAP_P256_KEY_LEN]; /*X 坐标值*/

uint8_t pub_key_y[ESBLE_GAP_P256_KEY_LEN]; /*Y 坐标值*/

} esble_gapm_pub_key_ind_t;

函数响应：无

函数描述：esble_gapm_get_pub_key()下行命令触发 BLE STACK 返回此事件，将读到的 PUBKEY 结果返回给 Application。

4. 1. 3. 14 创建广播指示

esapp_gamp_create_adv_ind(const esble_gapm_create_actv_ind_t *ind)

参数说明：

typedef struct _esble_gapm_create_actv_ind_t

{

uint8_t	actv_idx;	/*广播事件的索引*/
uint8_t	tx_pwr;	/*广播事件的发送功率*/

} esble_gapm_create_actv_ind_t;

函数响应：无

函数描述：esble_gapm_create_adv()下行命令触发 BLE STACK 返回此事件，将创建的广播事件的索引和发送功率值返回给 Application。

4. 1. 3. 15 开始广播指示

esapp_gapm_start_adv_ind(const esble_gapm_start_actv_ind_t *ind)

参数说明：

typedef struct _esble_gapm_start_actv_ind_t

{

uint8_t	actv_idx;	/*事件索引值*/
uint8_t	status;	/*状态*/

} esble_gapm_start_actv_ind_t;

函数响应：无

函数描述：esble_gapm_adv_start ()下行命令触发 BLE STACK 返回此事件，将开始的广播事件的索引和状态信息返回给 Application。

4. 1. 3. 16 停止广播指示

esapp_gapm_stop_adv_ind(const esble_gapm_stop_actv_ind_t *ind)

参数说明：

typedef struct _esble_gapm_stop_actv_ind_t

{

uint8_t	actv_idx;	/*事件索引*/
uint8_t	reason;	/*事件停止原因*/
uint8_t	per_adv_stop;	/*若是 PER ADV，指示 PER 是否停止*/

} esble_gapm_stop_actv_ind_t;

函数响应：无

函数描述：esble_gapm_adv_stop()下行命令触发 BLE STACK 返回此事件，将停止的广播事件的索引和停止原因等信息返回给 Application。

4. 1. 3. 17 扫描请求指示

esapp_gapm_scan_request_ind(const esble_gapm_scan_request_ind_t *ind)

参数说明：

```
typedef struct _esble_gapm_scan_request_ind_t
{
uint8_t actv_idx;                    /*事件的索引*/
esble_gap_bdaddr_t _trans_addr;      /*发送扫描请求设备的地址信息*/
} esble_gapm_scan_request_ind_t;
```

函数响应：无

函数描述：广播设备使能扫描请求通知功能时，若接收到扫描请求则会触发此事件返回给应用

4. 1. 3. 18 创建扫描指示

esapp_gamp_create_scan_ind(const esble_gapm_create_actv_ind_t *ind)

参数说明：

```
typedef struct _esble_gapm_create_actv_ind_t
{
uint8_t          actv_idx;          /*创建的事件索引*/
uint8_t          tx_pwr;            /*无用*/
} esble_gapm_create_actv_ind_t;
```

函数响应：无

函数描述：esble_gapm_create_scan()下行命令触发 BLE STACK 返回此事件，将创建的扫描事件的索引返回给 Application。

4. 1. 3. 19 开始扫描指示

esapp_gapm_start_scan_ind(const esble_gapm_start_actv_ind_t *ind)

参数说明：

```
typedef struct _esble_gapm_start_actv_ind_t
{
uint8_t          actv_idx;          /*事件索引值*/
uint8_t          status;            /*状态*/
```

} esble_gapm_start_actv_ind_t;

函数响应：无

函数描述：esble_gapm_scan_start ()下行命令触发 BLE STACK 返回此事件，将开始的扫描事件的索引和状态信息返回给 Application。

4. 1. 3. 20 停止扫描指示

esapp_gapm_stop_scan_ind(const esble_gapm_stop_actv_ind_t *ind)

参数说明：

typedef struct _esble_gapm_stop_actv_ind_t

{

uint8_t actv_idx; /*事件索引*/

uint8_t reason; /*事件停止原因*/

uint8_t per_adv_stop; /*若是 PER ADV，指示 PER 是否停止*/

} esble_gapm_stop_actv_ind_t;

函数响应：无

函数描述：esble_gapm_scan_stop()下行命令触发 BLE STACK 返回此事件，将停止的扫描事件的索引和停止原因等信息返回给 Application。

4. 1. 3. 21 创建发起连接指示

esapp_gamp_create_init_ind(const esble_gapm_create_actv_ind_t *ind)

参数说明：

typedef struct _esble_gapm_create_actv_ind_t

{

uint8_t actv_idx; /*创建的事件索引*/

uint8_t tx_pwr; /*无用*/

} esble_gapm_create_actv_ind_t;

函数响应：无

函数描述：esble_gapm_create_init()下行命令触发 BLE STACK 返回此事件，将创建的发起连接事件的索引返回给 Application。

4. 1. 3. 22 开始发起连接指示

esapp_gapm_start_init_ind(const esble_gapm_start_actv_ind_t *ind)

参数说明：

typedef struct _esble_gapm_start_actv_ind_t

{

```
uint8_t          actv_idx;        /*事件索引值*/
uint8_t          status;          /*状态*/
} esble_gapm_start_actv_ind_t;
```

函数响应：无

函数描述：esble_gapm_init_start ()下行命令触发 BLE STACK 返回此事件，将开始的发起连接事件的索引和状态信息返回给 Application。

4. 1. 3. 23 停止发起连接指示

esapp_gapm_stop_init_ind(const esble_gapm_stop_actv_ind_t *ind)

参数说明：

```
typedef struct _esble_gapm_stop_actv_ind_t
{
uint8_t      actv_idx;        /*事件索引*/
uint8_t      reason;          /*事件停止原因*/
uint8_t      per_adv_stop;    /*无用*/
} esble_gapm_stop_actv_ind_t;
```

函数响应：无

函数描述：esble_gapm_init_stop()下行命令触发 BLE STACK 返回此事件，将停止的发起连接事件的索引和停止原因等信息返回给 Application。

4. 1. 3. 24 创建周期同步指示

esapp_gamp_create_per_sync_ind(const esble_gapm_create_actv_ind_t *ind)

参数说明：

```
typedef struct _esble_gapm_create_actv_ind_t
{
uint8_t          actv_idx;        /*创建的事件索引*/
uint8_t          tx_pwr;          /*无用*/
} esble_gapm_create_actv_ind_t;
```

函数响应：无

函数描述：esble_gapm_create_period_sync()下行命令触发 BLE STACK 返回此事件，将创建的周期同步事件的索引返回给 Application。

4. 1. 3. 25 开始周期同步指示

esapp_gapm_start_per_sync_ind(const esble_gapm_start_actv_ind_t *ind)

参数说明：

typedef struct _esble_gapm_start_actv_ind_t

{

uint8_t actv_idx; /*事件索引值*/

uint8_t status; /*状态*/

} esble_gapm_start_actv_ind_t;

函数响应：无

函数描述：esble_gapm_init_period_sync()下行命令触发 BLE STACK 返回此事件，将开始的周期同步事件的索引和状态信息返回给 Application。

4. 1. 3. 26 周期同步建立指示

esapp_gapm_sync_established_ind(const esble_gapm_sync_established_ind_t *ind)

参数说明：

typedef struct _esble_gapm_sync_established_ind_t

{

uint8_t actv_idx; /*事件索引*/

uint8_t phy; /*同步建立所使用的 PHY*/

uint16_t intv; /*周期广播的事件间隔*/

uint8_t adv_sid; /*广播的 SID*/

uint8_t clk_acc; /*广播的时钟精度*/

esble_gap_bdaddr_t addr; /*广播的设备地址*/

} esble_gapm_sync_established_ind_t;

函数响应：无

函数描述：当周期同步接收到第一包周期广播数据时触发此事件上报同步建立成功指示。

4. 1. 3. 27 停止周期同步指示

esapp_gapm_stop_per_sync_ind(const esble_gapm_stop_actv_ind_t *ind)

参数说明：

typedef struct _esble_gapm_stop_actv_ind_t

{

uint8_t actv_idx; /*事件索引*/

uint8_t reason; /*事件停止原因*/

uint8_t per_adv_stop; /*无用*/

} esble_gapm_stop_actv_ind_t;

函数响应：无

函数描述：esble_gapm_period_sync_stop()下行命令触发 BLE STACK 返回此事件，将停止的周期同步事件的索引和停止原因等信息返回给 Application。

4. 1. 3. 28 全部停止指示

esapp_gapm_stop_all_ind(void)

参数说明：无

函数响应：无

函数描述：esble_gapm_stop_all_acts()下行命令触发 BLE STACK 返回此事件，返回给 Application。

4. 1. 3. 29 设备版本信息指示

esapp_gapm_dev_version_ind(const esble_gapm_dev_version_ind_t *ind)

参数说明：

```
typedef struct _esble_gapm_dev_version_ind_t
{
uint8_t     hci_ver;      /*HCI 版本*/
uint8_t     lmp_ver;      /*LMP 版本*/
uint8_t     host_ver;     /*Hos 版本*/
uint16_t    hci_subver;   /*HCI 修订版本*/
uint16_t    lmp_subver;   /*LMP 修订版本*/
uint16_t    host_subver;  /*Host 修订版本*/
uint16_t    manuf_name;   /*制造商名称*/
} esble_gapm_dev_version_ind_t;
```

函数响应：

函数描述：本地设备版本事件。

4. 1. 3. 30 设备发送功率指示

esapp_gapm_dev_tx_pwr_ind(const esble_gapm_dev_tx_pwr_ind_t *ind)

参数说明：

```
typedef struct _esble_gapm_dev_tx_pwr_ind_t
{
int8_t      min_tx_pwr;     /*最小发射功率*/
int8_t      max_tx_pwr;     /*最大发射功率*/
} esble_gapm_dev_tx_pwr_ind_t;
```

函数响应：

函数描述：控制器支持的传输功率。

4. 1. 3. 31 推荐默认数据长度指示

esapp_gapm_sugg_dflt_data_len_ind(const esble_gapm_sugg_dflt_data_len_ind_t *ind)

参数说明：

```
typedef struct _esble_gapm_sugg_dflt_data_len_ind_t
{
uint16_t    suggted_max_tx_octets;      /*主机对控制器最大传输荷载数建议值*/
uint16_t    suggted_max_tx_time;        /*主机对控制器最大包传输时间建议值*/
} esble_gapm_sugg_dflt_data_len_ind_t;
```

函数响应：

函数描述：当应用层请求建议数据长度时触发事件。

4. 1. 3. 32 设备RF路径补偿值指示

esapp_gapm_dev_rf_path_comp_ind(const esble_gapm_dev_rf_path_comp_ind_t *ind)

参数说明：

```
typedef struct _esble_gapm_dev_rf_path_comp_ind_t
{
uint16_t    tx_path_comp;           /*RF TX 路径补偿*/
uint16_t    rx_path_comp;           /*RF RX 路径补偿*/
} esble_gapm_dev_rf_path_comp_ind_t;
```

函数响应：

函数描述：RF 路径补偿值指示。

4. 1. 3. 33 最大数据长度指示

esapp_gapm_max_data_len_ind(const esble_gapm_max_data_len_ind_t *ind)

参数说明：

```
typedef struct _esble_gapm_max_data_len_ind_t
{
uint16_t  suppted_max_tx_octets;      /*本地支持的最大传输载荷（字节）*/
uint16_t  suppted_max_tx_time;        /*本地支持的最大传输时间（ms）*/
uint16_t  suppted_max_rx_octets;      /*本地支持的最大接收载荷（字节）*/
uint16_t  suppted_max_rx_time;        /*本地支持的最大接收时间（ms）*/
```

} esble_gapm_max_data_len_ind_t;

函数响应：

函数描述：当应用程序请求控制器支持的最大数据长度时触发此事件。

4. 1. 3. 34 对端设备名称指示

esapp_gapm_peer_name_ind(const esble_gapm_peer_name_ind_t *ind)

参数说明：

typedef struct _esble_gapm_peer_name_ind_t

{

esble_bd_addr_t	addr;	/*对端设备地址*/
uint8_t	addr_type;	/*对端设备地址类型*/
uint8_t	name_len;	/*对端设备名字长度*/
uint8_t	*name;	/*对端设备名字*/

} esble_gapm_peer_name_ind_t;

函数响应：

函数描述：当检索对端设备名称而获取到对端设备名称时，触发此事件。

4. 1. 3. 35 内存信息指示（仅调试状态可用）

esapp_gapm_dbg_mem_info_ind(const esble_gapm_dbg_mem_info_ind_t *ind)

参数说明：

typedef struct _esble_gapm_dbg_mem_info_ind_t

{

uint32_t	max_mem_used;	/*内存使用峰值*/
uint16_t	mem_used[ESBLE_KE_MEM_BLOCK_MAX];	

} esble_gapm_dbg_mem_info_ind_t;

函数响应：

函数描述：当应用程序请求当前使用的内存时，触发此事件。

4. 1. 3. 36 未知任务指示

esapp_gapm_unknown_task_ind(const esble_gapm_unknown_task_ind_t *ind)

参数说明：

typedef struct _esble_gapm_unknown_task_ind_t

{

uint16_t	msg_id;	/*消息标识符*/

uint16_t task_id; /*任务标识符*/

} esble_gapm_unknown_task_ind_t;

函数响应：

函数描述：未知任务指示。

4. 1. 3. 37 地址更新超时指示

esapp_gapm_addr_renew_to_ind(void const *ind)

参数说明：无

函数响应：

函数描述：可解析随机地址生成超时指示。

4. 1. 3. 38 自动连接超时指示

esapp_gapm_auto_conn_to_ind(void const *ind)

参数说明：无

函数响应：

函数描述：自动连接超时指示。

4. 1. 3. 39 测试模式结束指示

esapp_gapm_le_test_end_ind(const esble_gapm_le_test_end_ind_t *ind)

参数说明：

typedef struct _esble_gapm_le_test_end_ind_t

{

uint16_t nb_packet_received; /*接收包数*/

} esble_gapm_le_test_end_ind_t;

函数响应：

函数描述：如果接收包数大于 0，指示测试模式事件结束。

4. 1. 3. 40 添加Profile任务指示

esapp_gapm_profile_added_ind(const esble_gapm_profile_added_ind_t *ind)

参数说明：

typedef struct _esble_gapm_profile_added_ind_t

{

uint16_t prf_task_id; /*配置文件任务标识符*/

uint16_t prf_task_nb; /*配置文件任务编号*/

```
uint16_t        start_hdl;              /*服务启动句柄*/
} esble_gapm_profile_added_ind_t;
```

函数响应：

函数描述：添加 profile 任务时触发此事件。

4. 1. 3. 41 连接请求指示

esapp_gapc_conn_req_ind(const esble_gapc_conn_req_ind_t *ind)

参数说明：

```
typedef struct _esble_gapc_conn_req_ind_t
{
uint8_t         conidx;             /*连接索引*/
uint16_t        conhdl;             /*连接句柄*/
uint16_t        con_interval;       /*连接间隔  N*1.25ms*/
uint16_t        con_latency;        /*连接延迟*/
uint16_t        sup_to;             /*连接超时  N*10ms*/
uint8_t         clk_accuracy;       /*时钟精度*/
uint8_t         peer_addr_type;     /*对端设备地址类型*/
esble_bd_addr_t peer_addr;          /*对端设备 BLE 地址*/
} esble_gapc_conn_req_ind_t;
```

函数响应：

函数描述：通知已与对端设备建立连接，此消息是请求，因为它正在等待 esble_gapc_conn_cfm 消息。

4. 1. 3. 42 断开连接指示

esapp_gapc_disconn_ind(const esble_gapc_disconn_ind_t *ind)

参数说明：

```
typedef struct _esble_gapc_disconn_ind_t
{
uint8_t         conidx;             /*连接索引号*/
uint16_t        conhdl;             /*连接句柄*/
uint8_t         reason;             /*连接断开原因*/
} esble_gapc_disconn_ind_t;
```

函数响应：

函数描述：发送到应用程序以通知指定索引号的连接已断开。

4. 1. 3. 43 参数更新超时指示

esapp_gapc_param_upd_to_ind(void const *ind)

参数说明：无

函数响应：

函数描述：参数更新超时指示。

4. 1. 3. 44 参数更新指示

esapp_gapc_param_upd_ind(const esble_gapc_param_upd_ind_t *ind)

参数说明：

```
typedef struct _esble_gapc_param_upd_ind_t
{
uint8_t     conidx;          /*连接索引*/
uint16_t    con_interval;    /*连接间隔  N*1.25ms*/
uint16_t    con_latency;     /*连接延迟（事件数）*/
uint16_t    sup_to;          /*连接超时 N*10ms*/
} esble_gapc_param_upd_ind_t;
```

函数响应：

函数描述：连接参数被更新时触发此事件。

4. 1. 3. 45 参数更新请求指示

esapp_gapc_param_upd_req_ind(const esble_gapc_param_upd_req_ind_t *ind)

参数说明：

```
typedef struct _esble_gapc_param_upd_req_ind_t
{
uint8_t     conidx;      /*连接索引*/
uint16_t    intv_min;    /连接间隔最小值 N*1.25ms*/
uint16_t    intv_max;    /*连接间隔最大值 N*1.25ms*/
uint16_t    latency;     /*连接延迟（事件数）*/
uint16_t    time_out;    /*连接超时 N*10ms*/
} esble_gapc_param_upd_req_ind_t;
```

函数响应：

函数描述：当从机请求更新连接参数时，主机端触发此事件。

4. 1. 3. 46 设置设备信息请求指示

esapp_gapc_set_dev_info_req_ind(const esble_gapc_set_dev_info_req_ind_t *ind)

参数说明：

```
typedef struct _esble_gapc_set_dev_info_req_ind_t
{
uint8_t                 conidx;                 /*连接索引*/
uint8_t                 req;                    /*请求信息*/
union
{
    esble_gapc_dev_name_t   name;               /*设备名称*/
    uint16_t                appearance;         /*设备外观*/
} info;
} esble_gapc_set_dev_info_req_ind_t;
```

函数响应：

函数描述：对端设备请求改变本地设备的信息，如设备名称，外观等。

4. 1. 3. 47 读取设备信息请求指示

esapp_gapc_dev_info_req_ind(const esble_gapc_dev_info_req_ind_t *ind)

参数说明：

```
typedef struct _esble_gapc_dev_info_req_ind_t
{
uint8_t         conidx;         /*连接索引*/
uint8_t         req;            /*请求信息，设备名称，外观，从机首选参数*/
} esble_gapc_dev_info_req_ind_t;
```

函数响应：

函数描述：将请求的信息发送到对端设备。

4. 1. 3. 48 读取RSSI指示

esapp_gapc_con_rssi_ind(const esble_gapc_con_rssi_ind_t *ind)

参数说明：

```
typedef struct _esble_gapc_con_rssi_ind_t
{
uint8_t         conidx;             /*连接索引*/
```

int8_t rssi; /*RSSI*/

} esble_gapc_con_rssi_ind_t;

函数响应：

函数描述：请求获取连接的 RSSI 时，触发此事件。

4. 1. 3. 49 读取连接时的信道图指示

esapp_gapc_con_cnl_map_ind(const esble_gapc_con_cnl_map_ind_t *ind)

参数说明：

typedef struct _esble_gapc_con_cnl_map_ind_t

{

uint8_t conidx; /*连接索引*/

esble_le_chnl_map_t ch_map; /*信道表*/

} esble_gapc_con_cnl_map_ind_t;

函数响应：

函数描述：请求获取连接的信道映射表时，触发此事件。

4. 1. 3. 50 读取对端设备版本信息指示

esapp_gapc_peer_version_ind(const esble_gapc_peer_version_ind_t *ind)

参数说明：

typedef struct _esble_gapc_peer_version_ind_t

{

uint8_t conidx; /*连接索引*/

uint16_t compid; /*厂商名称*/

uint16_t lmp_subvers; /* LMP 版本*/

uint8_t lmp_vers; /*LMP 版本*/

} esble_gapc_peer_version_ind_t;

函数响应：

函数描述：请求获取对端设备版本信息时，触发此事件。

4. 1. 3. 51 读取对端设备特性指示

esapp_gapc_peer_features_ind(const esble_gapc_peer_features_ind_t *ind)

参数说明：

typedef struct _esble_gapc_peer_features_ind_t

{

```
uint8_t      conidx;                          /*连接索引*/
uint8_t      features[ESBLE_GAP_LE_FEATS_LEN];  /*8 字节的 LE 特性*/
} esble_gapc_peer_features_ind_t;
```

函数响应：

函数描述：请求获取对端设备 LE 特性时，触发此事件。

4. 1. 3. 52 读取对端设备属性数据指示

```
esapp_gapc_peer_att_info_ind(const esble_gapc_peer_att_info_ind_t *ind)
```

参数说明：

```
typedef struct _esble_gapc_peer_att_info_ind_t
{
uint8_t          conidx;          /*连接索引*/
uint8_t          req;             /*请求信息*/
uint16_t         handle;          /*属性句柄*/
union
{
    esble_gapc_dev_name_t   name;          /*设备名称*/
    uint16_t                appearance;    /*外观*/
    esble_gapc_slv_pref_t   slv_pref_params;  /*从设备首选参数*/
    uint8_t                 ctl_addr_resol;   /*中心地址解析*/
} info;
} esble_gapc_peer_att_info_ind_t;
```

函数响应：

函数描述：请求获取属性数据库信息时，触发此事件。

4. 1. 3. 53 读取PHY指示

```
esapp_gapc_le_phy_ind(const esble_gapc_le_phy_ind_t *ind)
```

参数说明：

```
typedef struct _esble_gapc_le_phy_ind_t
{
uint8_t      conidx;      /*连接索引*/
uint8_t      tx_phy;      /*发射 PHY*/
uint8_t      rx_phy;      /*接收 PHY*/
```

} esble_gapc_le_phy_ind_t;

函数响应：

函数描述：请求获取物理层信息时，触发此事件

4. 1. 3. 54 读取信道选择算法指示

esapp_gapc_chan_sel_algo_ind(const esble_gapc_chan_sel_algo_ind_t *ind)

参数说明：

typedef struct _esble_gapc_chan_sel_algo_ind_t

{

uint8_t conidx; /*连接索引*/

uint8_t chan_sel_algo; /*信道选择算法*/

} esble_gapc_chan_sel_algo_ind_t;

函数响应：

函数描述：请求获取信道选择算法时，触发此事件。

4. 1. 3. 55 PING超时指示

esapp_gapc_le_ping_to_ind(void)

参数说明：无

函数响应：

函数描述：LE Ping 超时指示。

4. 1. 3. 56 PING超时时间指示

esapp_gapc_le_ping_to_val_ind(const esble_gapc_le_ping_to_val_ind_t *ind)

参数说明：

typedef struct _esble_gapc_le_ping_to_val_ind_t

{

uint8_t conidx; /*连接索引*/

uint16_t timeout; /*验证的负载超时 N*10ms*/

} esble_gapc_le_ping_to_val_ind_t;

函数响应：

函数描述：LE Ping 超时指示。

4. 1. 3. 57 载荷大小指示

esapp_gapc_le_pkt_size_ind(const esble_gapc_le_pkt_size_ind_t *ind)

参数说明：

```
typedef struct _esble_gapc_le_pkt_size_ind_t
{
uint8_t     conidx;          /*连接索引*/
uint16_t    max_tx_octets;   /*最大 TX 有效载荷*/
uint16_t    max_tx_time;     /*本地控制器最大的 TX 时间*/
uint16_t    max_rx_octets;   /*最大 RX 有效载荷*/
uint16_t    max_rx_time;     /*本地控制器最大的 RX 时间*/
} esble_gapc_le_pkt_size_ind_t;
```

函数响应：

函数描述：修改本地数据长度扩展参数时，触发此事件。

4. 1. 3. 58 加密指示

esapp_gapc_encrypt_ind(const esble_gapc_encrypt_ind_t *ind)

参数说明：

```
typedef struct _esble_gapc_encrypt_ind_t
{
uint8_t     conidx;     /*连接索引*/
uint8_t     auth;       /*认证等级*/
} esble_gapc_encrypt_ind_t;
```

函数响应：

函数描述：加密成功时触发此事件。

4. 1. 3. 59 加密请求指示

esapp_gapc_encrypt_req_ind(const esble_gapc_encrypt_req_ind_t *ind)

参数说明：

```
typedef struct _esble_gapc_encrypt_req_ind_t
{
uint8_t         conidx;     /*连接索引*/
uint16_t        ediv;       /*加密转换器*/
esble_rand_nb_t rand_nb;    /*随机数*/
} esble_gapc_encrypt_req_ind_t;
```

函数响应：

函数描述：从设备加密时触发此事件，以便获取合适的 LTK。

4. 1. 3. 60 安全请求指示

esapp_gapc_sec_ind(const esble_gapc_sec_ind_t *ind)

参数说明：

```
typedef struct _esble_gapc_sec_ind_t
{
uint8_t     conidx;     /*连接索引*/
uint8_t     auth;       /*认证等级*/
} esble_gapc_sec_ind_t;
```

函数响应：

函数描述：从设备请求一定级别的身份验证时，由主设备触发此事件。

4. 1. 3. 61 按键通知指示

esapp_gapc_key_press_notif_ind(const esble_gapc_key_press_notif_ind_t *ind)

参数说明：

```
typedef struct _esble_gapc_key_press_notif_ind_t
{
uint8_t     conidx;             /*连接索引*/
uint8_t     notification_type;  /*通知类型*/
} esble_gapc_key_press_notif_ind_t;
```

函数响应：

函数描述：指示在对端设备上有按键按下。

4. 1. 3. 62 数据签名指示

esapp_gapc_sing_cnt_ind(const esble_gapc_sign_cnt_ind_t *ind)

参数说明：

```
typedef struct _esble_gapc_sign_cnt_ind_t
{
uint8_t     conidx;             /*连接索引*/
uint32_t    local_sign_counter; /*本地签名数据*/
uint32_t    peer_sign_counter;  /*对端设备签名数据*/
} esble_gapc_sign_cnt_ind_t;
```

函数响应：

函数描述：签名数据上传给应用层。

4. 1. 3. 63 多次配对超时指示

esapp_gapc_smp_rep_attempts_timer_ind(void const *ind)

参数说明：无

函数响应：

函数描述：重复多次配对超时指示。

4. 1. 3. 64 配对超时指示

esapp_gapc_smp_to_timer_ind(void const *ind)

参数说明：无

函数响应：

函数描述：配对超时指示。

4. 1. 3. 65 绑定指示

esapp_gapc_bond_ind(const esble_gapc_bond_ind_t *ind)

参数说明：

typedef struct _esble_gapc_bond_ind_t

{

uint8_t	conidx;	/*连接索引*/
uint8_t	info;	/*绑定信息类型*/
esble_gapc_bond_data_t	data;	/*绑定数据*/

} esble_gapc_bond_ind_t;

函数响应：

函数描述：当绑定信息有效时，触发此事件，比如配对的状态，与对端设备交换秘钥等。

4. 1. 3. 66 绑定请求指示

esapp_gapc_bond_req_ind(const esble_gapc_bond_req_ind_t *ind)

参数说明：

typedef struct _esble_gapc_bond_req_ind_t

{

uint8_t	conidx	/*连接索引*/
uint8_t	request;	/*绑定类型*/
esble_gapc_bond_req_data_t	data;	/*绑定请求数据*/

} esble_gapc_bond_req_ind_t;

函数响应：

函数描述：在绑定规程中触发此事件，以获取从机配对信息，配对临时秘钥，在秘钥交换时将秘钥提供给对端设备。

4. 1. 3. 67 签名结果指示

esapp_gapc_sign_ind(const esble_gapc_sign_ind_t *ind)

参数说明：

```
typedef struct _esble_gapc_sign_ind_t
{
uint8_t         conidx;
uint16_t        byte_len;       /*Data PDU 长度（字节）*/
uint8_t         *signed_msg;    /* Data PDU + SignCounter + MAC*/
} esble_gapc_sign_ind_t;
```

函数响应：

函数描述：签名结果。

4. 2 GATT

GAPTT 是标准的 BLE 协议，它是建立在 ATT 协议基础上的。ATT 协议描述了 BLE 的基本数据单位：属性的结构和存取方法。ATT 相当于基类，而 GATT 就是派生类，它重新组织了属性并构建了应用层存取数据的结构：服务/特征，它定义了发现、读取、写入、通知这些数据结构的方法和机制。

4. 2. 1 数据定义

4. 2. 1. 1 服务权限

```
enum esble_attm_svc_perm_mask
{
ESBLE_PERM_MASK_SVC_MI          = 0x01,/*多实例的管理服务任务*/
ESBLE_PERM_POS_SVC_MI           = 0,
ESBLE_PERM_MASK_SVC_EKS         = 0x02, /*检查服务访问的秘钥长度*/
ESBLE_PERM_POS_SVC_EKS          = 1,
ESBLE_PERM_MASK_SVC_AUTH        = 0x0C, /*服务权限认证*/
ESBLE_PERM_POS_SVC_AUTH         = 2,
ESBLE_PERM_MASK_SVC_DIS         = 0x10, /*禁用服务*/
ESBLE_PERM_POS_SVC_DIS          = 4,
```

```
ESBLE_PERM_MASK_SVC_UUID_LEN      = 0x60, /*服务 UUID 长度*/
ESBLE_PERM_POS_SVC_UUID_LEN       = 5,
ESBLE_PERM_MASK_SVC_SECONDARY     = 0x80, /*次级服务*/
ESBLE_PERM_POS_SVC_SECONDARY      = 7,
};
```

4. 2. 1. 2 ATTM权限

```
enum esble_attm_perm_mask
{
ESBLE_PERM_MASK_ALL                = 0x0000,    /*检索所有权限信息*/
ESBLE_PERM_MASK_RP                 = 0x0003,    /*读取权限掩码*/
ESBLE_PERM_POS_RP                  = 0,
ESBLE_PERM_MASK_WP                 = 0x000C,    /*写权限掩码*/
ESBLE_PERM_POS_WP                  = 2,
ESBLE_PERM_MASK_IP                 = 0x0030,    /*指示访问掩码*/
ESBLE_PERM_POS_IP                  = 4,
ESBLE_PERM_MASK_NP                 = 0x00C0,    /*通知访问掩码*/
ESBLE_PERM_POS_NP                  = 6,
ESBLE_PERM_MASK_BROADCAST          = 0x0100,    /*广播描述符*/
ESBLE_PERM_POS_BROADCAST           = 8,
ESBLE_PERM_MASK_RD                 = 0x0200,    /*读访问掩码*/
ESBLE_PERM_POS_RD                  = 9,
ESBLE_PERM_MASK_WRITE_COMMAND      = 0x0400,    /*使能写命令属性掩码*/
ESBLE_PERM_POS_WRITE_COMMAND       = 10,
ESBLE_PERM_MASK_WRITE_REQ          = 0x0800,    /*使能写请求属性掩码*/
ESBLE_PERM_POS_WRITE_REQ           = 11,
ESBLE_PERM_MASK_NTF                = 0x1000,    /*通知访问掩码*/
ESBLE_PERM_POS_NTF                 = 12,
ESBLE_PERM_MASK_IND                = 0x2000,    /*指示访问掩码*/
ESBLE_PERM_POS_IND                 = 13,
ESBLE_PERM_MASK_WRITE_SIGNED       = 0x4000,    /*使能写签名访问掩码*/
ESBLE_PERM_POS_WRITE_SIGNED        = 14,
ESBLE_PERM_MASK_EXT                = 0x8000,    /*扩展属性描述符*/
```

```
ESBLE_PERM_POS_EXT                  = 15,
ESBLE_PERM_MASK_PROP                = 0xFF00,    /*保留*/
ESBLE_PERM_POS_PROP                 = 8,
};
```

4. 2. 1. 3 权限值字段定义

```
enum esble_attm_value_perm_mask
{
ESBLE_PERM_MASK_MAX_LEN             = 0x0FFF,    /* 属性最大长度*/
ESBLE_PERM_POS_MAX_LEN              = 0,
ESBLE_PERM_MASK_VAL_OFFSET          = 0x0FFF,    /*属性值补偿*/
ESBLE_PERM_POS_VAL_OFFSET           = 0,
ESBLE_PERM_MASK_EKS                 = 0x1000,    /* 检查秘钥大小掩码*/
ESBLE_PERM_POS_EKS                  = 12,
ESBLE_PERM_MASK_UUID_LEN            = 0x6000,    /* UUID 长度*/
ESBLE_PERM_POS_UUID_LEN             = 13,
ESBLE_PERM_MASK_RI                  = 0x8000,    /* 读取触发指示*/
ESBLE_PERM_POS_RI                   = 15,
};
```

4. 2. 1. 4 属性描述结构体

```
typedef struct _esble_gattm_att_desc_t
{
uint16_t  suuid;                    /*16bits UUID (小端模式)*/
uint8_t   luuid[ESBLE_ATT_UUID_128_LEN];   /*128bits UUID (小端模式)*/
uint16_t  perm;                     /*属性权限*/
uint16_t  ext_perm;                 /*属性扩展权限*/
uint16_t  max_len;                  /* 最大属性长度*/
} esble_gattm_att_desc_t;
```

属性权限见 4. 2. 1. 2,属性扩展权限见 4. 2. 1. 3

4. 2. 1. 5 服务信息

```
typedef struct _esble_gattm_svc_info_t
```

```
{
uint16_t        start_hdl;              /* 服务起始句柄*/
uint16_t        end_hdl;                /* 服务结束句柄*/
uint16_t        task_id;                /* 任务 ID */
uint8_t         perm;                   /* 服务权限*/
} esble_gattm_svc_info_t;
```

4. 2. 1. 6 通用完成事件

```
typedef struct _esble_gattc_cmp_evt_t
{
uint8_t                 conidx;
esble_gattc_cmd_enum    cmd;        /* GATTC 请求类型*/
uint8_t                 status;     /* 操作状态*/
uint16_t                seq_num;    /* 操作序列号（操作启动时提供）*/
} esble_gattc_cmp_evt_t;
```

4. 2. 1. 7 发现服务指示

```
typedef struct _esble_gattc_disc_svc_ind_t
{
uint8_t         conidx;
uint16_t        start_hdl;          /* 起始句柄*/
uint16_t        end_hdl;            /* 结束句柄*/
uint8_t         uuid_len;           /* UUID 长度*/
uint8_t         *uuid;              /* 服务 UUID*/
} esble_gattc_disc_svc_ind_t;
```

4. 2. 1. 8 简单读服务

```
typedef struct _esble_gattc_read_simple_t
{
uint16_t        handle;             /* 属性句柄*/
uint16_t        offset;             /* 数据中的偏移*/
uint16_t        length;             /* 读取的数据长度 (0：全部读取)*/
} esble_gattc_read_simple_t;
```

4. 2. 1. 9 通过UUID读取服务

```
typedef struct _esble_gattc_read_by_uuid_t
{
uint16_t    start_hdl;          /* 起始句柄*/
uint16_t    end_hdl;            /* 结束句柄*/
uint8_t     uuid_len;           /* UUID 长度*/
uint8_t     *uuid;              /* UUID 值*/
} esble_gattc_read_by_uuid_t;
```

4. 2. 1. 10 读取多个服务

```
typedef struct _esble_gattc_read_multiple_t
{
uint16_t    handle;             /* 属性句柄*/
uint16_t    len;                /* 句柄长度 (不能为 0)*/
} esble_gattc_read_multiple_t;
```

4. 2. 1. 11 读取服务请求

```
typedef union _esble_gattc_read_req_t
{
esble_gattc_read_simple_t     simple;   /* 简单读取服务*/
esble_gattc_read_by_uuid_t    by_uuid;  /* 通过 UUID 读取服务*/
esble_gattc_read_multiple_t   multiple[1];/* 读取多个服务*/
} esble_gattc_read_req_t;
```

4. 2. 1. 12 特征描述符信息

```
typedef struct _esble_gattc_sdp_att_char_t
{
uint8_t     conidx;
uint8_t     att_type;                   /* 属性类型*/
uint8_t     prop;                       /* 特征值*/
uint16_t    handle;                     /* 句柄*/
} esble_gattc_sdp_att_char_t;
```

4. 2. 1. 13 包含服务信息

```
typedef struct _esble_gattc_sdp_include_svc_t
{
uint8_t         conidx;
uint8_t         att_type;                       /* 属性类型*/
uint8_t         uuid_len;                       /* 包含服务的 UUID 长度*/
uint8_t         uuid[ESBLE_ATT_UUID_128_LEN];   /* 包含服务的 UUID*/
uint16_t        start_hdl;                      /* 包含服务的起始句柄*/
uint16_t        end_hdl;                        /* 包含服务的结束句柄*/
} esble_gattc_sdp_include_svc_t;
```

4. 2. 1. 14 属性信息

```
typedef struct _esble_gattc_sdp_att_t
{
uint8_t         conidx;
uint8_t         att_type;                       /* 属性类型*/
uint8_t         uuid_len;                       /* UUID 长度*/
uint8_t         uuid[ESBLE_ATT_UUID_128_LEN];   /* 属性 UUID*/
} esble_gattc_sdp_att_t;
```

4. 2. 1. 15 扫描的服务信息

```
typedef union _esble_gattc_sdp_att_info_t
{
uint8_t         conidx;
uint8_t         att_type;                       /* 属性类型*/
esble_gattc_sdp_att_char_t      att_char;       /* 特征描述符信息*/
esble_gattc_sdp_include_svc_t   inc_svc;        /* 包含服务信息*/
esble_gattc_sdp_att_t           att;            /* 属性信息*/
} esble_gattc_sdp_att_info_t;
```

4. 2. 2 GATT 下行命令

4. 2. 2. 1 添加服务请求

```
esble_gattm_add_svc_req(esble_gattm_add_svc_req_t *req)
```

参数说明：

```
typedef struct _esble_gattm_add_svc_req_t
{
uint8_t                 idx;
uint8_t                 perm;       /* 服务权限*/
uint8_t                 nb_att;     /* 服务中属性个数*/
uint16_t                suuid;      /* 服务 UUID(16bits)*/
uint16_t                suuid;      /* 服务 UUID(16bits)*/
esble_gattm_att_desc_t  *atts;      /* 服务中的属性描述表*/
} esble_gattm_add_svc_req_t;
```

服务中的属性描述表见 4. 2. 1. 4

函数响应：esapp_gattm_add_svc_rsp

函数描述：此函数将服务添加到数据库中。

4. 2. 2. 2 读取服务权限请求

esble_gattm_svc_get_permission_req(esble_gattm_svc_get_permission_req_t *req))

参数说明：

```
typedef struct _esble_gattm_svc_get_permission_req_t
{
uint16_t    start_hdl;          /* 起始服务句柄*/
} esble_gattm_svc_get_permission_req_t;
```

函数响应：esapp_gattm_svc_get_permission_rsp

函数描述：获取服务权限请求。

4. 2. 2. 3 设置服务权限请求

esble_gattm_svc_set_permission_req(esble_gattm_svc_set_permission_req_t *req))

参数说明：

```
typedef struct _esble_gattm_svc_set_permission_req_t
{
uint16_t    start_hdl;          /* 起始服务句柄*/
uint8_t     perm;               /* 服务权限*/
} esble_gattm_svc_set_permission_req_t;
```

函数响应：esapp_gattm_svc_set_permission_rsp

函数描述：设置服务权限。

4. 2. 2. 4 读取属性权限请求

esble_gattm_att_get_permission_req(esble_gattm_att_get_permission_req_t *req))

参数说明：

```
typedef struct _esble_gattm_att_get_permission_req_t
{
uint16_t        handle;                 /* 属性句柄*/
} esble_gattm_att_get_permission_req_t;
```

函数响应：esapp_gattm_att_get_permission_rsp

函数描述：获取属性权限。

4. 2. 2. 5 设置属性权限请求

esble_gattm_att_set_permission_req(esble_gattm_att_set_permission_req_t *req))

参数说明：

```
typedef struct _esble_gattm_att_set_permission_req_t
{
uint16_t        handle;                 /* 属性句柄*/
uint16_t        perm;                   /* 属性权限*/
uint16_t        ext_perm;               /* 扩展属性权限*/
} esble_gattm_att_set_permission_req_t;
```

函数响应：esapp_gattm_att_set_permission_rsp

函数描述：设置属性权限。

4. 2. 2. 6 读取属性值请求

esble_gattm_att_get_value_req(esble_gattm_att_get_value_req_t *req))

参数说明：

```
typedef struct _esble_gattm_att_get_value_req_t
{
uint16_t        handle;                 /* 属性句柄*/
} esble_gattm_att_get_value_req_t;
```

函数响应：esapp_gattm_att_get_value_rsp

函数描述：获取属性值。

4. 2. 2. 7 设置属性值请求

esble_gattm_att_set_value_req(esble_gattm_att_set_value_req_t *req))

参数说明：

typedef struct _esble_gattm_att_set_value_req_t

{

uint16_t	handle;	/* 属性句柄*/
uint16_t	length;	/* 属性值长度*/
uint8_t	*value;	/* 属性值*/

} esble_gattm_att_set_value_req_t;

函数响应：esapp_gattm_att_set_value_rsp

函数描述：设置属性值。

4. 2. 2. 8 清除数据库请求

esble_gattm_destroy_db_req(esble_gattm_destroy_db_req_t *req)

参数说明：

typedef struct _esble_gattm_destroy_db_req_t

{

uint16_t	gap_hdl;	/* 新 GAP 起始句柄*/
uint16_t	gatt_hdl;	/* 新 GATT 起始句柄*/

} esble_gattm_destroy_db_req_t;

函数响应：

函数描述：设置 GAP 与 GATT 的句柄值，此函数会破坏数据库，仅用于调试的目的进行修改。

4. 2. 2. 9 读取服务列表请求

esble_gattm_svc_get_list_req(void *req))

参数说明：

函数响应：esapp_gattm_svc_get_list_rsp

函数描述：获取服务列表。

4. 2. 2. 10 读取属性信息请求

esble_gattm_att_get_info_req(esble_gattm_att_get_info_req_t *req))

参数说明：

typedef struct _esble_gattm_att_get_info_req_t

```
{
uint16_t        handle;                 /* Attribute Handle*/
} esble_gattm_att_get_info_req_t;
```

函数响应：esapp_gattm_att_get_info_rsp

函数描述：检索属性信息。

4. 2. 2. 11 交互MTU

esble_gattc_mtu_exch_cmd(esble_gattc_exc_mtu_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gattc_exc_mtu_cmd_t
{
uint8_t         conidx;
uint16_t        seq_num;                /*  操作序号  */
} esble_gattc_exc_mtu_cmd_t;
```

函数响应：esapp_gattc_cmp_evt、esapp_gattc_mtu_changed_ind

函数描述：交换 MTU。

4. 2. 2. 12 发现所有服务

esble_gattc_disc_all_svc_cmd(esble_gattc_disc_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gattc_disc_cmd_t
{
uint8_t         conidx;
uint8_t         uuid_len;               /* UUID 长度*/
uint16_t        seq_num;                /*  操作序号*/
uint16_t        start_hdl;              /*  起始句柄范围*/
uint16_t        end_hdl;                /*  结束句柄范围*/
uint8_t         *uuid;                  /* UUID*/
} esble_gattc_disc_cmd_t;
```

函数响应：esapp_gattc_cmp_evt、esapp_gattc_disc_svc_ind

函数描述：该函数应用于发现对端设备从 start_hdl 到 end_hdl 的所有服务，当搜索到 UUID 或者对端设备没有更多的服务时，此操作停止。要查找所有服务时，搜索的 UUID 设置为 0x0000。

4. 2. 2. 13 发现指定UUDI服务

esble_gattc_disc_by_uuid_svc_cmd(esble_gattc_disc_cmd_t *cmd);

参数说明：详见 4. 2. 2. 12

函数响应：esapp_gattc_cmp_evt、esapp_gattc_disc_svc_ind

函数描述：此函数应用于发现对端设备给定范围内搜索 UUID 对应的所有服务，当对端设备中没有更对的服务可用时，此操作停止。

4. 2. 2. 14 发现包含服务

esble_gattc_disc_included_svc_cmd(esble_gattc_disc_cmd_t *cmd)

参数说明：详见 4. 2. 2. 12

函数响应：esapp_gattc_cmp_evt、esapp_gattc_disc_svc_incl_ind

函数描述：当服务在句柄范围内创建时，触发 ESBLE_GATTC_DISC_SVC_INCL_IND 事件。参数中 UUID 设置为 0x0000。

4. 2. 2. 15 发现所有特征

esble_gattc_disc_all_char_cmd(esble_gattc_disc_cmd_t *cmd)

参数说明：详见 4. 2. 2. 12

函数响应：esapp_gattc_cmp_evt、esapp_gattc_disc_char_ind

函数描述：该函数应用于发现对端设备句柄范围内的所有特征，UUID 设置为 0x0000。

4. 2. 2. 16 发现指定UUID的特征

esble_gattc_disc_by_uuid_char_cmd(esble_gattc_disc_cmd_t *cmd)

参数说明：详见 4. 2. 2. 12

函数响应：esapp_gattc_cmp_evt、esapp_gattc_disc_char_ind

函数描述：此函数应用于发现对端设备句柄范围内，uuid 所描述的服务。

4. 2. 2. 17 发现特征描述符

esble_gattc_disc_desc_char_cmd(esble_gattc_disc_cmd_t *cmd)

参数说明：详见 4. 2. 2. 12

函数响应：esapp_gattc_cmp_evt、esapp_gattc_disc_char_desc_ind

函数描述：当特征描述符在句柄范围内时，触发 ESBLE_ GATTC_DISC_DESC_CHAR 事件。参数中 UUID 设置为 0x0000。

4. 2. 2. 18 读取服务

esble_gattc_read(esble_gattc_read_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gattc_read_cmd_t
{
uint8_t                 conidx;
uint8_t                 nb;         /* 读取服务数目 (仅用于读取多个服务)*/
uint16_t                seq_num;    /* 操作序号*/
esble_gattc_read_req_t  *req;       /* 读取类型*/
} esble_gattc_read_cmd_t;
```

函数响应：esapp_gattc_cmp_evt、esapp_gattc_read_ind、esapp_gattc_read_req_ind

函数描述：此函数只需要设置句柄，即可以从设置偏移地址中读取特定长度的数据，若读取全部属性，则需要把 length 设置为 0。

4. 2. 2. 19 读取服务中指定长度

esble_gattc_read_long(esble_gattc_read_cmd_t *cmd)

参数说明：详见 4. 2. 2. 18

函数响应：esapp_gattc_cmp_evt、esapp_gattc_read_ind、esapp_gattc_read_req_ind

函数描述：此函数只需要设置句柄，即可以从设置偏移地址中读取特定长度的数据，若读取全部属性，则需要把 length 设置为 0。

4. 2. 2. 20 读取指定UUID的服务

esble_gattc_read_by_uuid(esble_gattc_read_cmd_t *cmd)

参数说明：详见 4. 2. 2. 18

函数响应：esapp_gattc_cmp_evt、esapp_gattc_read_ind、esapp_gattc_read_req_ind

函数描述：此函数在句柄范围内，搜索对应的 UUID 找到的第一个参数。

4. 2. 2. 21 读取多个服务

esble_gattc_read_multiple(esble_gattc_read_cmd_t *cmd)

参数说明：详见 4. 2. 2. 18

函数响应：esapp_gattc_cmp_evt、esapp_gattc_read_ind、esapp_gattc_read_req_ind

函数描述：调用此函数能够同时读取多个句柄。调用此函数时，应知道对端设备的属性大小，否则esapp_gattc_cmp_evt 将返回状态错误。

4. 2. 2. 22 读取确认

esble_gattc_read_cfm(esble_gattc_read_cfm_t *cfm)

参数说明：

```
typedef struct _esble_gattc_read_cfm_t
{
uint8_t        conidx;
uint16_t       handle;          /* 读取的句柄*/
uint16_t       length;          /* 读取的数据长度*/
uint8_t        status;          /* 写命令执行的状态*/
uint8_t        *value;          /* 读取的属性值*/
} esble_gattc_read_cfm_t;
```

函数响应：无

函数描述：确认上层读取，将触发对端设备的读请求。

4. 2. 2. 23 属性信息确认

esble_gattc_att_info_cfm(esble_gattc_att_info_cfm_t *cmd)

参数说明：

```
typedef struct _esble_gattc_att_info_cfm_t
{
uint8_t        conidx;
uint16_t       handle;          /* 属性句柄*/
uint16_t       length;          /* 现有的属性长度*/
uint8_t        status;          /* 上层命令执行状态*/
} esble_gattc_att_info_cfm_t;
```

函数响应：无

函数描述：属性信息确认，用以通知对端设备是否被授权修改属性值，并获得当前属性长度。

4. 2. 2. 24 写特征值

esble_gattc_write(esble_gattc_write_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gattc_write_cmd_t
{
uint8_t        conidx;
bool           auto_execute;    /* 自动写*/
uint16_t       seq_num;         /* 操作序号*/
uint16_t       handle;          /* 属性句柄*/
```

```
uint16_t        offset;                 /* 写偏移地址*/
uint16_t        length;                 /* 写长度*/
uint16_t        cursor;                 /* 内部写出光标，必须初始化为 0*/
uint8_t         *value;                 /* 写入的值*/
} esble_gattc_write_cmd_t;
```

函数响应：esapp_gattc_cmp_evt、esapp_gattc_write_req_ind、esapp_gattc_att_info_req_ind

函数描述：此函数用于修改对端设备的特征描述符。使用此函数修改对端设备的描述符，需 auto_execute 设置为 1，offset 设置为 0。

4. 2. 2. 25 无响应写

esble_gattc_write_no_response(esble_gattc_write_cmd_t *cmd)

参数说明：详见 4. 2. 2. 24

函数响应：esapp_gattc_write_req_ind

函数描述：使用此函数修改对端设备的属性描述符时，一但发送完数据包，则触发 esapp_gattc_cmp_evt。

4. 2. 2. 26 带签名写

esble_gattc_write_signed(esble_gattc_write_cmd_t *cmd)

参数说明：详见 4. 2. 2. 24

函数响应：esapp_gattc_cmp_evt、esapp_gapc_sign_counter_ind

函数描述：修改签名值。一但发送完数据包，则触发 esapp_gattc_cmp_evt。

4. 2. 2. 27 执行写

esble_gattc_exec_write(esble_gattc_execute_write_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gattc_execute_write_cmd_t
{
uint8_t         conidx;
bool            execute;        /* 1：执行挂起的写操作；0：取消挂起的写操作*/
uint16_t        seq_num;        /* 操作序号*/
} esble_gattc_execute_write_cmd_t;
```

函数响应：esapp_gattc_cmp_evt、esapp_gattc_write_req_ind

函数描述：此函数用于执行/取消对对端设备挂起的准备写入操作。

4. 2. 2. 28 写确认

esble_gattc_write_cfm(esble_gattc_write_cfm_t *cfm)

参数说明：

typedef struct _esble_gattc_write_cfm_t

{

uint8_t conidx;

uint16_t handle; /* 写的属性句柄*/

uint8_t status; /* 写命令的执行状态*/

} esble_gattc_write_cfm_t;

函数响应：无

函数描述：确认上层写命令，将触发对端设备写请求

4. 2. 2. 29 注册属性

esble_gattc_register(esble_gattc_reg_to_peer_evt_cmd_t *cmd)

参数说明：

typedef struct _esble_gattc_reg_to_peer_evt_cmd_t

{

uint8_t conidx;

uint16_t seq_num; /* 操作序号*/

uint16_t start_hdl; /* 起始句柄*/

uint16_t end_hdl; /* 结束句柄*/

} esble_gattc_reg_to_peer_evt_cmd_t;

函数响应：esapp_gattc_cmp_evt

函数描述：在对端设备中注册新的属性表。

4. 2. 2. 30 取消注册属性

esble_gattc_unregister(esble_gattc_reg_to_peer_evt_cmd_t *cmd)

参数说明：详见 4. 2. 2. 29

函数响应：esapp_gattc_cmp_evt

函数描述：在对端设备中取消注册新的属性表。

4. 2. 2. 31 事件确认

esble_gattc_event_cfm(esble_gattc_event_cfm_t *cmd)

参数说明：

```
typedef struct _esble_gattc_event_cfm_t
{
uint8_t        conidx;
uint16_t       handle;                    /* 属性句柄*/
} esble_gattc_event_cfm_t;
```

函数响应：无

函数描述：此函数必须在收到 esapp_gattc_event_req_ind 后调用，以确认收到指示。

4. 2. 2. 32 发现指定服务（包含特征等）

esble_gattc_sdp_disc_svc_cmd(esble_gattc_sdp_svc_disc_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gattc_sdp_svc_disc_cmd_t
{
uint8_t        conidx;
uint8_t        uuid_len;                  /* UUID 长度*/
uint16_t       seq_num;                   /* 操作序号*/
uint16_t       start_hdl;                 /* 搜索的起始句柄*/
uint16_t       end_hdl;                   /* 搜索的结束句柄*/
uint8_t        uuid[ESBLE_ATT_UUID_128_LEN];   /* 服务 UUID*/
} esble_gattc_sdp_svc_disc_cmd_t;
```

函数响应：esapp_gattc_cmp_evt、esapp_gattc_disc_svc_ind

函数描述：此函数搜索特定服务，将自动搜索包含的服务、特征和描述符。

4. 2. 2. 33 发现所有服务（包含特征等）

esble_gattc_sdp_disc_svc_all_cmd(esble_gattc_sdp_svc_disc_cmd_t *cmd)

参数说明：详见 4. 2. 2. 32

函数响应：esapp_gattc_cmp_evt、esapp_gattc_disc_svc_ind

函数描述：此函数搜索全部服务，将自动搜索包含的服务、特征和描述符。

4. 2. 2. 34 取消发现服务

esble_gattc_sdp_disc_cancel_cmd(esble_gattc_sdp_svc_disc_cmd_t *cmd)

参数说明：详见 4. 2. 2. 32

函数响应：esapp_gattc_cmp_evt

函数描述：取消搜索服务操作。

4. 2. 2. 35 发送通知

esble_gattc_send_notf(esble_gattc_send_evt_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gattc_send_evt_cmd_t
{
uint8_t     conidx;
uint16_t    seq_num;        /* 操作序号*/
uint16_t    handle;         /* 特征值句柄*/
uint16_t    length;         /* 发送的数据长度*/
uint8_t     *value;         /* 数据*/
} esble_gattc_send_evt_cmd_t;
```

函数响应：esapp_gattc_cmp_evt、esapp_gattc_event_ind

函数描述：发送通知。

4. 2. 2. 36 发送指示

esble_gattc_send_ind(esble_gattc_send_evt_cmd_t *cmd)

参数说明：详见 4. 2. 2. 35

函数响应：esapp_gattc_cmp_evt、esapp_gattc_event_req_ind

函数描述：发送指示。

4. 2. 2. 37 更改服务

esble_gattc_svc_changed_cmd(esble_gattc_send_svc_changed_cmd_t *cmd)

参数说明：

```
typedef struct _esble_gattc_send_svc_changed_cmd_t
{
uint8_t     conidx;
uint16_t    seq_num;        /* 操作序号*/
uint16_t    svc_shdl;       /* 更改的起始句柄*/
uint16_t    svc_ehdl;       /* 更改的结束句柄*/
} esble_gattc_send_svc_changed_cmd_t;
```

函数响应：esapp_gattc_cmp_evt、esapp_gattc_event_req_ind

函数描述：此函数用来发送服务描述符已更改的指示。

4. 2. 3 GATT 上行事件

4. 2. 3. 1 添加服务响应

esapp_gattm_add_svc_rsp(const esble_gattm_add_svc_rsp_t *rsp)

参数说明：

```
typedef struct _esble_gattm_add_svc_rsp_t
{
uint16_t        start_hdl;              /* 数据库中分配服务的起始句柄*/
uint8_t         status;                 /* 数据库中分配服务的状态*/
} esble_gattm_add_svc_rsp_t;
```

函数响应：

函数描述：响应添加服务请求，如果添加成功，则 start_hdl 为第一个属性的起始句柄。

状态代码：

1. ATT_ERR_NO_ERROR：服务添加成功。
2. ATT_ERR_INVALID_HANDLE：添加的服务句柄覆盖了一些现有的服务句柄。
3. ATT_INSUFF_RESOURCE：没有足够的内存来分配服务缓冲区

4. 2. 3. 2 读取服务权限响应

esapp_gattm_svc_get_permission_rsp(const esble_gattm_svc_get_permission_rsp_t *ind)

参数说明：

```
typedef struct _esble_gattm_svc_get_permission_rsp_t
{
uint16_t        start_hdl;              /* 起始服务句柄*/
uint8_t         perm;                   /* 服务权限*/
uint8_t         status;                 /* 返回状态*/
} esble_gattm_svc_get_permission_rsp_t;
```

函数响应：

函数描述：响应获取服务权限请求。

状态代码：

1. ATT_ERR_NO_ERROR: 成功 。
2. ATT_ERR_INVALID_HANDLE: 数据库中没有此服务句柄。

4. 2. 3. 3 设置服务权限响应

esapp_gattm_svc_set_permission_rsp(const esble_gattm_svc_set_permission_rsp_t *ind)

参数说明：

```
typedef struct _esble_gattm_svc_set_permission_rsp_t
{
uint16_t        start_hdl;              /* 起始属性句柄*/
uint8_t         status;                 /* 返回状态*/
} esble_gattm_svc_set_permission_rsp_t;
```

函数响应：

函数描述：响应设置服务权限请求。

状态代码：

1. ATT_ERR_NO_ERROR: 成功
2. ATT_ERR_INVALID_HANDLE: 数据库中没有此服务句柄

4. 2. 3. 4 读取属性权限响应

```
esapp_gattm_att_get_permission_rsp(const esble_gattm_att_get_permission_rsp_t *ind)
```

参数说明：

```
typedef struct _esble_gattm_att_get_permission_rsp_t
{
uint16_t        handle;                 /* 属性句柄*/
uint16_t        perm;                   /* 属性权限*/
uint16_t        ext_perm;               /* 扩展属性权限*/
uint8_t         status;                 /* 返回状态*/
}esble_gattm_att_get_permission_rsp_t;
```

函数响应：

函数描述：响应获取属性权限请求。

状态代码：

1. ATT_ERR_NO_ERROR: 成功
2. ATT_ERR_INVALID_HANDLE: 数据库中没有此服务句柄

4. 2. 3. 5 设置属性权限响应

```
esapp_gattm_att_set_permission_rsp(const esble_gattm_att_set_permission_rsp_t *ind)
```

参数说明：

```
typedef struct _esble_gattm_att_set_permission_rsp_t
{
```

uint16_t handle; /* 属性句柄*/

uint8_t status; /* 返回状态*/

} esble_gattm_att_set_permission_rsp_t;

函数响应：

函数描述：响应设置属性权限请求。

状态代码：

1. ATT_ERR_NO_ERROR: 成功
2. ATT_ERR_INVALID_HANDLE: 数据库中没有此服务句柄

4. 2. 3. 6 读取属性值响应

esapp_gattm_att_get_value_rsp(const esble_gattm_att_get_value_rsp_t *ind)

参数说明：

typedef struct _esble_gattm_att_get_value_rsp_t

{

uint16_t handle; /* 属性句柄*/

uint16_t length; /* 属性值长度*/

uint8_t status; /* 返回状态*/

uint8_t *value; /* 属性值*/

} esble_gattm_att_get_value_rsp_t;

函数响应：

函数描述：响应获取属性值请求。

状态代码：

1. ATT_ERR_NO_ERROR: 成功
2. ATT_ERR_INVALID_HANDLE: 数据库中没有此服务句柄

4. 2. 3. 7 设置属性值响应

esapp_gattm_att_set_value_rsp(const esble_gattm_att_get_value_rsp_t *ind)

参数说明：

typedef struct _esble_gattm_att_set_value_rsp_t

{

uint16_t handle; /* 属性句柄*/

uint8_t status; /* 返回状态*/

} esble_gattm_att_set_value_rsp_t;

函数响应：

函数描述：响应设置属性值请求。

状态代码：

1. ATT_ERR_NO_ERROR: 成功。
2. ATT_ERR_INVALID_HANDLE: 数据库中没有此服务句柄。
3. ATT_ERR_INVALID_ATTRIBUTE_VAL_LEN: 参数长度超过最大属性值长度

4. 2. 3. 8 清除数据库响应

esapp_gattm_destory_db_rsp(const esble_gattm_destroy_db_rsp_t *ind)

参数说明：

```
typedef struct _esble_gattm_destroy_db_rsp_t
{
uint8_t        status;                    /* 返回状态*/
} esble_gattm_destroy_db_rsp_t;
```

函数响应：

函数描述：响应 esble_gattm_att_set_value_req，返回修改 GAP 与 GATTA 句柄的操作结果。

状态代码：

1. ATT_ERR_NO_ERROR: 成功。

4. 2. 3. 9 读取服务列表响应

esapp_gattm_svc_get_list_rsp(const esble_gattm_svc_get_list_rsp_t *ind)

参数说明：

```
typedef struct _esble_gattm_svc_get_list_rsp_t
{
uint8_t                    status;        /* 返回状态*/
uint8_t                    nb_svc;        /*服务数目*/
esble_gattm_svc_info_t     *svc;          /* 服务信息*/
} esble_gattm_svc_get_list_rsp_t;
```

服务信息详见 4. 2. 1. 5

函数响应：

函数描述：响应 esble_gattm_svc_get_list_req，返回服务列表信息。

状态代码：

1. ATT_ERR_NO_ERROR: 成功。
2. ATT_ERR_INVALID_HANDLE: 数据库中没有此服务句柄。

4. 2. 3. 10 读取属性信息响应

esapp_gattm_att_get_info_rsp(const esble_gattm_att_get_info_rsp_t *ind)

参数说明：

typedef struct _esble_gattm_att_get_info_rsp

{

uint8_t	status;	/* 返回状态*/
uint8_t	uuid_len;	/* UUID 长度*/
uint16_t	handle;	/* 属性句柄*/
uint16_t	perm;	/* 属性权限*/
uint16_t	ext_perm;	/* 扩展属性权限*/
uint8_t	uuid[ESBLE_ATT_UUID_128_LEN];	/* UUID value*/

} esble_gattm_att_get_info_rsp_t;

函数响应：

函数描述：响应 esble_gattm_att_get_info_req，返回属性信息

4. 2. 3. 11 GATTC事件完成

esapp_gattc_cmp_evt(const esble_gattc_cmp_evt_t *ind)

typedef struct _esble_gattc_cmp_evt_t

{

uint8_t	conidx;	
esble_gattc_cmd_enum	cmd;	/* GATT 请求类型*/
uint8_t	status;	/* 请求状态*/
uint16_t	seq_num;	/* 操作序列号（操作启动时提供）*/

} esble_gattc_cmp_evt_t;

函数响应：

函数描述：此函数为 GATTC 的通用完成时间。操作完成后，所有的操作都会触发此函数。

4. 2. 3. 12 MTU更改指示

esapp_gattc_mtu_changed_ind(const esble_gattc_mtu_changed_ind_t *ind)

参数说明：

typedef struct _esble_gattc_mtu_changed_ind_t

{

uint8_t	conidx;	

```
uint16_t        mtu;                    /* 交换后的 MTU 值*/
uint16_t        seq_num;                /* 操作序号*/
} esble_gattc_mtu_changed_ind_t;
```

函数响应：

函数描述：响应 esble_gattc_mtu_exch_cmd，返回交换后的 MTU 值。

4. 2. 3. 13 发现服务指示

esapp_gattc_disc_svc_ind(const esble_gattc_disc_svc_ind_t *ind)

参数说明：

```
typedef struct _esble_gattc_disc_svc_ind_t
{
uint8_t         conidx;
uint16_t        start_hdl;              /* 起始句柄*/
uint16_t        end_hdl;                /* 结束句柄*/
uint8_t         uuid_len;               /* UUID 长度*/
uint8_t         *uuid;                  /* UUID*/
} esble_gattc_disc_svc_ind_t;
```

函数响应：

函数描述：在调用 esble_gattc_disc_all_svc_cmd、esble_gattc_disc_by_uuid_svc_cmd 后，会返回此函数。

4. 2. 3. 14 发现包含服务指示

esapp_gattc_disc_svc_incl_ind(const esble_gattc_disc_svc_incl_ind_t *ind)

参数说明：

```
typedef struct _esble_gattc_disc_svc_incl_ind_t
{
uint8_t         conidx;
uint16_t        attr_hdl;               /* 元素句柄*/
uint16_t        start_hdl;              /* 起始句柄*/
uint16_t        end_hdl;                /* 结束句柄*/
uint8_t         uuid_len;               /* UUID 长度*/
uint8_t         *uuid;                  /* 包含的服务 UUID*/
} esble_gattc_disc_svc_incl_ind_t;
```

函数响应：

函数描述：在调用 esble_gattc_disc_included_svc_cmd 后，若发现包含服务时，触发此函数

4. 2. 3. 15 发现特征指示

esapp_gattc_disc_char_ind(const esble_gattc_disc_char_ind_t *ind)

参数说明：

```
typedef struct _esble_gattc_disc_char_ind_t
{
uint8_t     conidx;
int16_t     attr_hdl;          /* 元素句柄*/
uint16_t    pointer_hdl;       /* 指向 UUID 的指针属性句柄*/
uint8_t     prop;              /* 特征值*/
uint8_t     uuid_len;          /* UUID 长度*/
uint8_t     *uuid;             /* 特征描述符的 UUID*/
} esble_gattc_disc_char_ind_t;
```

函数响应：

函数描述：在调用 esble_gattc_disc_all_char_cmd、esble_gattc_disc_by_uuid_char_cmd 后，会返回此函数。

4. 2. 3. 16 发现特征描述符指示

esapp_gattc_disc_char_desc_ind(const esble_gattc_disc_char_desc_ind_t *ind)

参数说明：

```
typedef struct _esble_gattc_disc_char_desc_ind_t
{
uint8_t     conidx;
int16_t     attr_hdl;          /* 元素句柄*/
uint8_t     uuid_len;          /* UUID 长度*/
uint8_t     *uuid;             /* 特征描述符的 UUID*/
} esble_gattc_disc_char_desc_ind_t;
```

函数响应：

函数描述：在调用 esble_gattc_disc_desc_char_cmd 后，会返回此函数。

4. 2. 3. 17 读命令指示

esapp_gattc_read_ind(const esble_gattc_read_ind_t *ind)

参数说明：

```
typedef struct _esble_gattc_read_ind_t
{
uint8_t     conidx;
uint16_t    handle;          /* 属性句柄*/
uint16_t    offset;          /* 读物的偏移地址*/
uint16_t    length;          /* 读取长度*/
uint8_t     *value;          /* 读取的数据*/
} esble_gattc_read_ind_t;
```

函数响应：

函数描述：调用 esble_gattc_read、esble_gattc_read_long、esble_gattc_read_by_uuid、esble_gattc_read_multiple 时，触发此函数。

4. 2. 3. 18 写请求指示

```
esapp_gattc_write_req_ind(const esble_gattc_write_req_ind_t *ind)
```

参数说明：

```
typedef struct _esble_gattc_write_req_ind_t
{
uint8_t     conidx;
uint16_t    handle           /* 写入的属性句柄*/
uint16_t    offset;          /* 写入的偏移地址*/
uint16_t    length;          /* 写入的长度*/
uint8_t     *value;          /* 向属性库中写入的值*/
} esble_gattc_write_req_ind_t;
```

函数响应：

函数描述：接收对端设备写入请求。

4. 2. 3. 19 事件指示

```
esapp_gattc_event_ind(const esble_gattc_event_ind_t *ind)
```

参数说明：

```
typedef struct _esble_gattc_event_ind_t
{
uint8_t     conidx;
uint8_t     type;            /* 事件类型*/
uint16_t    length;          /* 数据长度*/
```

```
uint16_t        handle;                 /* 属性句柄*/
uint8_t         *value;                 /* 事件数据*/
} esble_gattc_event_ind_t;
```

函数响应：

函数描述：此函数在注册任务时被触发，此事件包含了新的属性句柄

4. 2. 3. 20 事件请求指示

esapp_gattc_event_req_ind(const esble_gattc_event_ind_t *ind)

参数说明：

详见 4. 2. 3. 19

函数响应：

函数描述：此函数被注册任务触发，此事件包含了新的对端属性句柄。此函数必须经过 GATTC_EVENT_CFM 消息确认

4. 2. 3. 21 发现服务（包含特征等）指示

esapp_gattc_sdp_svc_ind(const esble_gattc_sdp_svc_ind_t *ind)

参数说明：

```
typedef struct _esble_gattc_sdp_svc_ind_t
{
uint8_t         conidx;
uint8_t         uuid_len;
uint8_t         uuid[ESBLE_ATT_UUID_128_LEN];       /* 服务 UUID*/
uint16_t  start_hdl;                                /* 起始服务句柄*/
uint16_t  end_hdl;                                  /*结束服务句柄*/
esble_gattc_sdp_att_info_t  *info;                  /* 服务中的属性信息*/
} esble_gattc_sdp_svc_ind_t;
```

函数响应：

函数描述：当发现服务后，会触发此事件

4. 2. 3. 22 服务更改配置指示

esapp_gattc_svc_changed_cfg_ind(const esble_gattc_svc_changed_cfg_t *ind)

参数说明：

```
typedef struct _esble_gattc_svc_changed_cfg_t
{
```

```
uint8_t         conidx;
uint16_t        ind_cfg;                    /* 更改后的客户端属性描述符*/
} esble_gattc_svc_changed_cfg_t;
```

函数响应：

函数描述：每次由对端设备发送客户配置描述符时，都会触发应用程序发送此函数。

5 应用例程

SDK 中以源码的形式提供了几种 BLE 常用的应用例程，供用户参考和修改。应用例程的源码可在 SDK 的目录\Projects\ES32W3120\Applications 下找到。

5.1 一主多从

LED_multilink_central。主要用于演示作为 Master 与多个 LED Server 之间的连接与控制。

主机端所能连接的最大的从机个数可通过宏 CONN_COUNTS_MAX 设置，上电后主机自动进入扫描态，此时 PA7 对应的 LED 灯亮，当扫描到 CONN_COUNTS_MAX 个 Device name 为指定的"ES32W3120-LED1"的广播时（需注意各个从机的 BDADDR 应设置为不同的值，否则主机会当作同一个设备），开始进行发起态，此时 PA7 对应的 LED 灯灭，PA6 对应的 LED 灯亮，当建立连接成功后，PA6 对应的 LED 灯灭，PA8 对应的 LED 灯亮。

LED Server 端（即从机端）上电后自动进入广播状态，此时 PA6 对应的 LED 灯亮，当成功建立连接后，PA 对应的 LED 灯灭，PA11 对应的 LED 灯亮。

当主机与所有从机都建立连接后，主机可通过 PB12 对应的按键来控制所有从机设备的 PA15 对应的 LED 灯的亮与灭，每个从机也可通过 PB12 对应的按键来控制主机设备 PA11 对应的 LED 灯的亮与灭。

5.2 OTA

OTA_EN。主要用于演示 OTA 升级功能，需要配合对应的 APP：OTA-BLE 使用。支持应用代码的升级操作。

当需要对应用代码进行升级时，需要将待升级的应用程序的版本号设置为比原应用程序的版本号高，并生成用于 OTA 升级的 BIN 文件，具体的升级流程可参考 2.4.7。