安信可蓝牙Mesh开发专题

安信可开源团队

目　录

序言

更新时间：2022-01-14 16:00

安信可蓝牙Mesh开发资料

本文档基于安信可蓝牙Mesh系列模组的资料归档，包括AT指令使用、二次开发等资料，欢迎前来购买使用；

安信可科技蓝牙系列模组是针对物联网设计通用型的蓝牙模组，其功能强大、用途广泛。可以用于智能灯、智能插座、智能空调等其他智能家电。

同时符合SIG Mesh规范，可直接通过智能手机组建Mesh网络，可对接天猫精灵智能音箱。

支持自组网和

在售型号

型号	蓝牙	功耗	尺寸封装	供电	样品购买
PB-01	ble 5.0				淘宝链接
PB-02	ble 5.0				淘宝链接
PB-03	ble 5.2				
PB-03F	ble 5.2		24.0*16.0*3.1(±0.2) mm		
PB-03M	ble 5.2				
TB-01	ble 5.0				
TB-02	ble 5.0				
TB-03	ble 5.0				
TB-04	ble 5.0		12.2 * 13.0 * 2.3(±0.2) MM		

联系方式

公司官网：https://www.ai-thinker.com

开发资料：https://docs.ai-thinker.com

淘宝店铺：https://anxinke.taobao.com

天猫店铺：https://aithinker.tmall.com

商务合作：sales@aithinker.com

技术支持：support@aithinker.com

联系地址：深圳市宝安区西乡华丰智慧创新港C座403、408~410室

商务电话：0755-29162996

欢迎关注微信公众号 “安信可科技” ，干货实时推送！

快速使用

更新时间：2022-01-14 16:00

一、前言

本章节教会开发者快速使用 PB-01 模组与微信小程序进行蓝牙透传通讯。

硬件准备：

- 安信可PB-01模组或开发板 * 1PCS： 链接

软件准备：

- 微信搜索微信小程序 物联网常用工具 点击使用 安信可蓝牙透传 选项。
- 确保已烧录了最新的AT固件，可从安信可资料库网站获取。
- 如需PB-01模组烧录最新固件，见烧录教程的章节。
- 下载安信可模组调试助手：链接

二、操作步骤

联系方式

公司官网：https://www.ai-thinker.com

开发资料：https://docs.ai-thinker.com

淘宝店铺：https://anxinke.taobao.com

天猫店铺：https://aithinker.tmall.com

商务合作：sales@aithinker.com

技术支持：support@aithinker.com

联系地址：深圳市宝安区西乡华丰智慧创新港C座403、408~410室

商务电话：0755-29162996

欢迎关注微信公众号 “安信可科技” ，干货实时推送！

固件更新

TB系列模组更新固件教程

更新时间：2022-01-19 16:00

一、概述

安信可TB系列蓝牙模块采用的是 Telink 825X 系列芯片。该芯片内置 512K Flash，芯片硬件只支持官方专用烧录器烧录，不支持串口烧录。专用烧录器价格昂贵，每个开发者购买一个烧录器显然是不现实的。安信可作为物联网行业的推动者，在原来芯片的基础上开发了串口烧录的功能，可以极大地降低用户的开发成本，加速蓝牙技术的普及。

串口烧录工具采用 Python 语言编写，分为图形界面和命令行脚本两个版本，开发者可以非常方便地将自己编译的程序烧录到芯片中进行调试。为了解决之前版本（如 V1.5.0）的串口烧录软件无法烧录不带专有的 bootloader 固件的模组，以及因为 bootloader 固件占用 flash 空间而无法支持 OTA 升级的问题。

我们最新的串口烧录软件“安信可 TB 模块调试工具 V3.1.1”已解决上述问题，其原理是通过把 SWS数据转换成 UART 数据的协议实现擦除及烧录，然后在烧录工具里预先内置了一个bootloader 固件，只是将其烧录到芯片 ram 里面运行，用于加快烧录速度，但在硬件上需要把模组自身的 RXD 和 SWS 短接起来才能实现。

二、接线

图2.1 (TB-01烧录接线示意图）：需把 SWS 和 RXD 引脚短接。

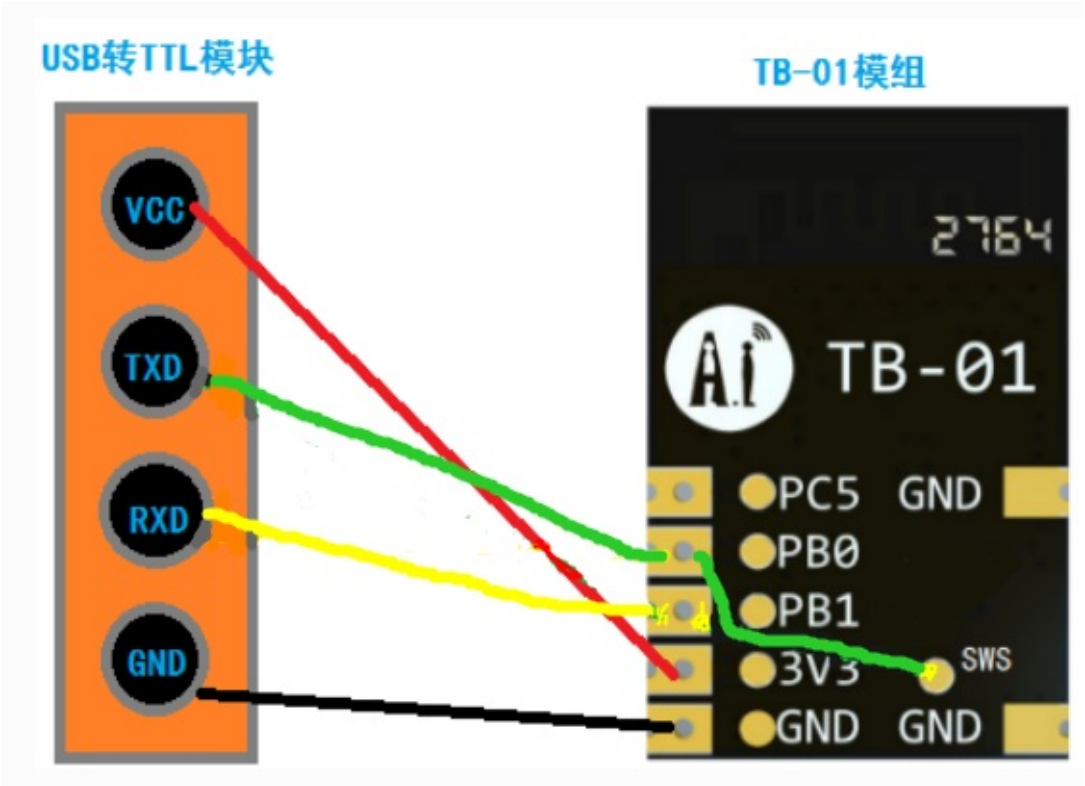


图2.2（TB-02烧录接线示意图）：需把 SWS 和 RXD 引脚短接。

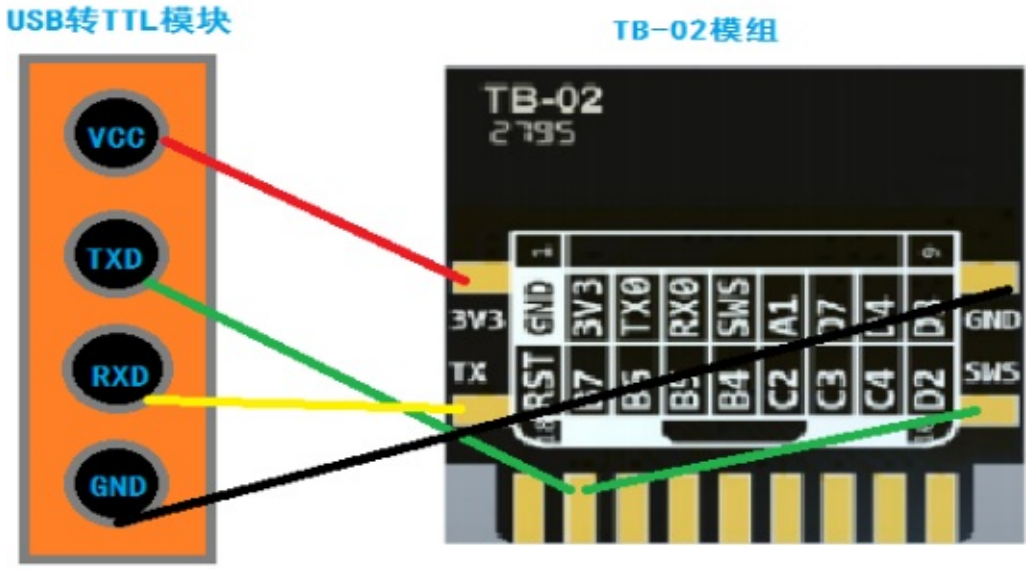


图2.3 (TB-03F烧录接线示意图 ）：需把 SWS 和 RXD 引脚短接。

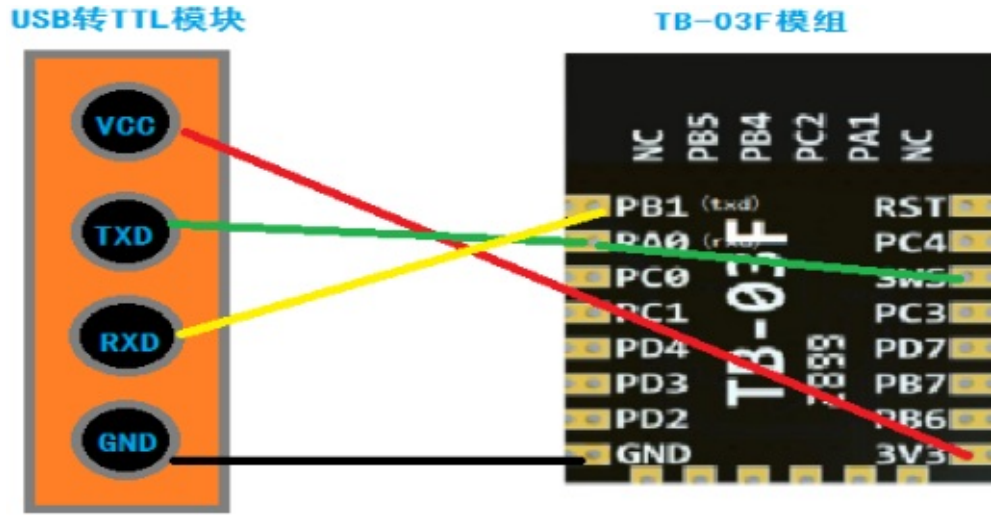


图2.4 (TB-04烧录接线示意图 ）：需把 SWS 和 RXD 引脚短接。

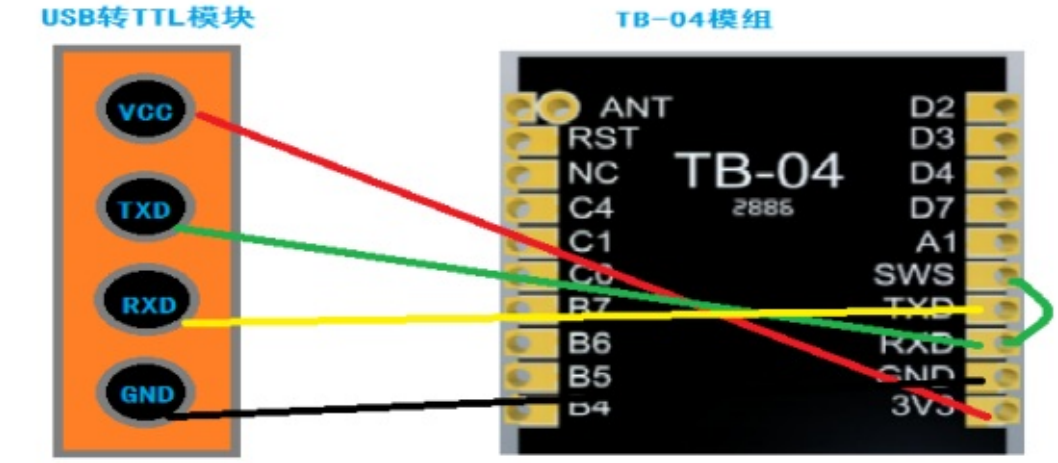


图2.5 TB-02开发板接线须知：需把 SWS 和 RXD 引脚短接。

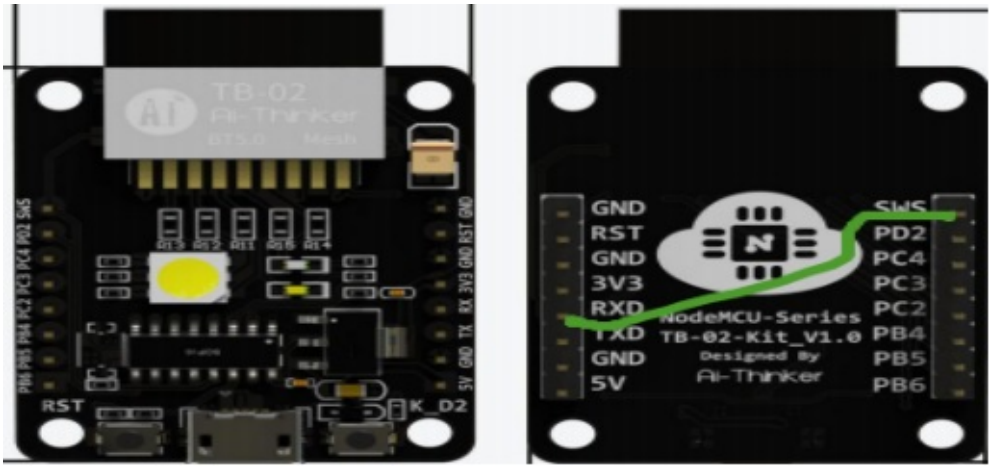


图2.6 TB-04开发板接线须知：需把 SWS 和 RXD 引脚短接。

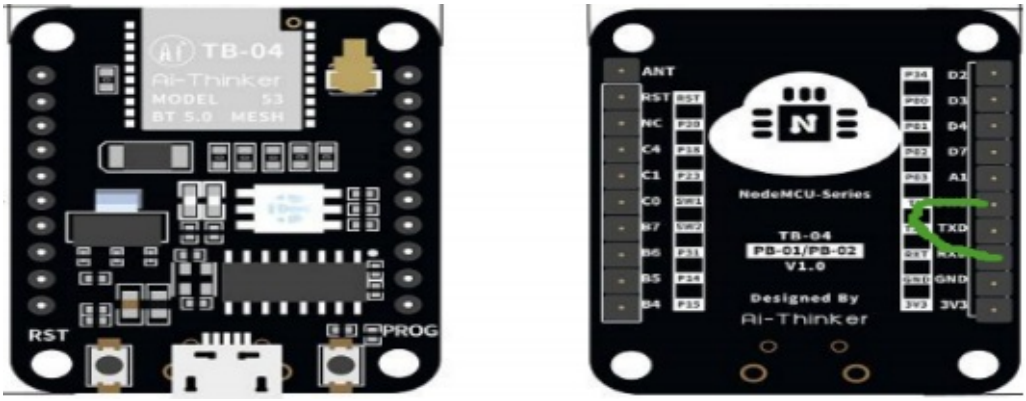


图2.7 TB-03F开发板接线须知：需把 SWS 和 RXD 引脚短接。

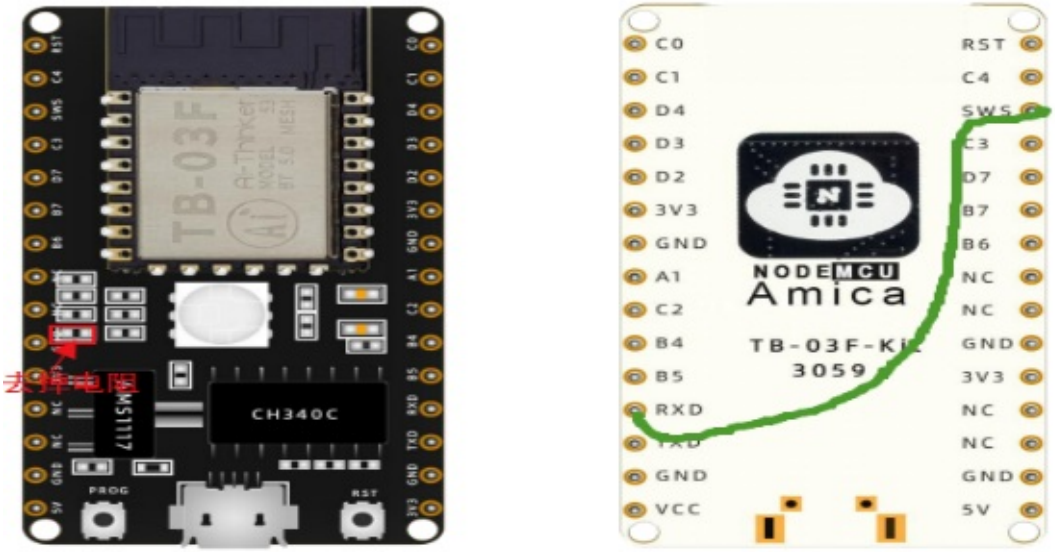


图2.8 TB-03F实物烧录接线：需把 SWS 和 RXD 引脚短接。

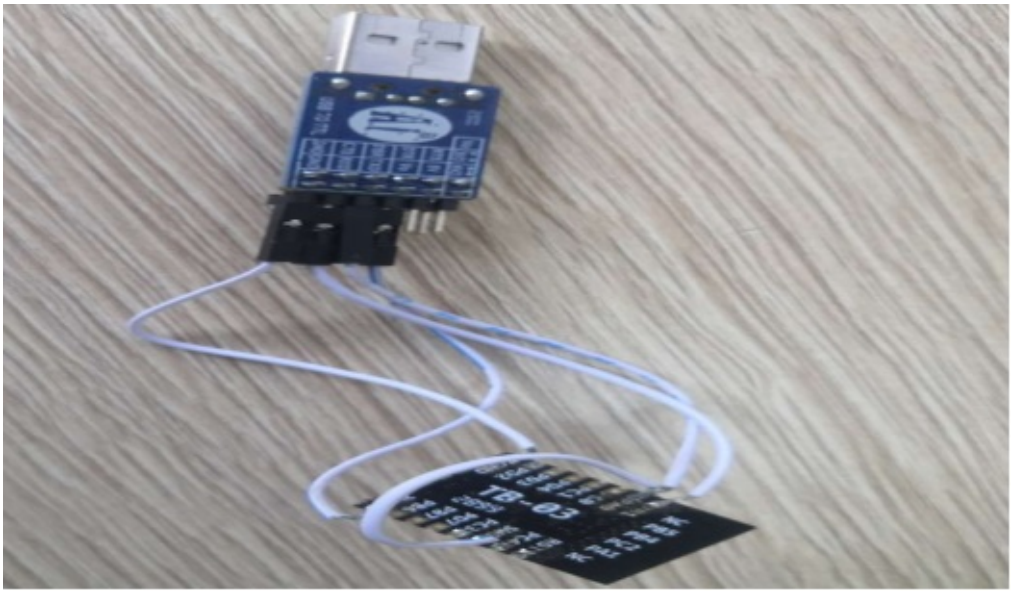


图2.9 TB-02开发板实物烧录接线：需把 SWS 和 RXD 引脚短接。

注意 Note ：

1. 请使用CP2102芯片或CH340芯片的usb转ttl模块，对应模组型号如上图2.1~2.4接线，在上电复位后进行烧录操作。
2. 如果是TB-02开发板和TB-04开发板，如图2.5~2.6只需用一根杜邦线将开发板上的SWS和RXD短接，用Mirco usb接线到PC烧录即可。
3. 如果是TB-03F开发板，如图2.7需要去掉与SWS连接的电阻,再与RXD短接用Mirco usb接线到PC烧录即可。实物接线可参考图2.8~2.9。

下载资源链接

1. TB系列模组固件烧录工具下载：https://docs.ai-thinker.com/_media/ai-thinker_tb_tools3.1.1.zip
2. USB转TTL电脑驱动（CP2102 和 CH340）：https://docs.ai-thinker.com/_media/tools/serial_driver_windos.7z

3. 烧录录工具及出厂最新AT固件获取：https://aithinker.readthedocs.io/zh_CN/latest/docs/taobao/ble/index.html#tb
4. 更多资源请访问：https://docs.ai-thinker.com/blue_tooth

三、烧录

首先点击 刷新串口 ，然后在串口选择框选择对应的COM口，然后点击 · · · 按钮选择要烧录的固件，建议每次烧录前先点击 整片擦除 擦除芯片flash，点击按钮 烧录固件 即可烧录，烧录成功后 Log 窗口将变成绿色，烧录失败 Log 窗口将变成红色。

烧录成功后，串口启动信息分别如图3.3和图3.4。

烧录界面按钮说明：

- 1、烧录三元组

  图形界面上的 擦除固件 ，分别对应三元组的ProductID,MAC,Secert， 在输入框中输入相应的数据并正确选择串口号，点击烧录三元组按钮即可烧录三元组。同样，烧录成功后Log窗口将变成绿色，烧录失败Log窗口将变成红色。（注意：该三元组是对接天猫精灵固件的三元组，一般固件不支持）
- 2、擦除固件

  点击按钮 擦除固件 ，将擦除模块中的固件,一般不用。
- 3、擦除 Mesh 数据

  点击 擦除Mesh Key 按钮，将擦除模块中的 Mesh 配网信息,包括Application Key 和 NetWork Key。
- 4、整片擦除

  点击 整片擦除 按钮，将擦除模块中所有的Flash 区域。
- 5、清空窗口

  点击 清空窗口 按钮，将清空烧录窗口打印的信息。

四、常见问题

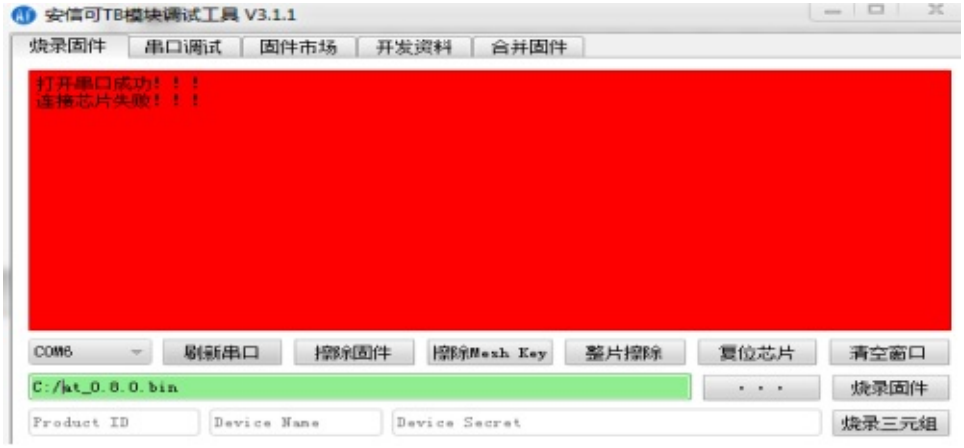
4.1、固件不存在

如果提示固件不存在，可能是固件路径位置不对，请重新选择正确固件位置即可。

4.2、串口打开失败

如果提示 打开串口 xxxx 失败....,可能是串口被其他软件占用，解除暂用后再试一次即可。

4.3、连接芯片失败



如果提示连接芯片失败,可能是接线错误请检查接线,注意模组自身的SWS跟RXD需要短接在一起才能进行串口烧录。

联系方式

公司官网：https://www.ai-thinker.com

开发资料：https://docs.ai-thinker.com

淘宝店铺：https://anxinke.taobao.com

天猫店铺：https://aithinker.tmall.com

商务合作：sales@aithinker.com

技术支持：support@aithinker.com

联系地址：深圳市宝安区西乡华丰智慧创新港C座403、408~410室

商务电话：0755-29162996

欢迎关注微信公众号 “安信可科技” ，干货实时推送！

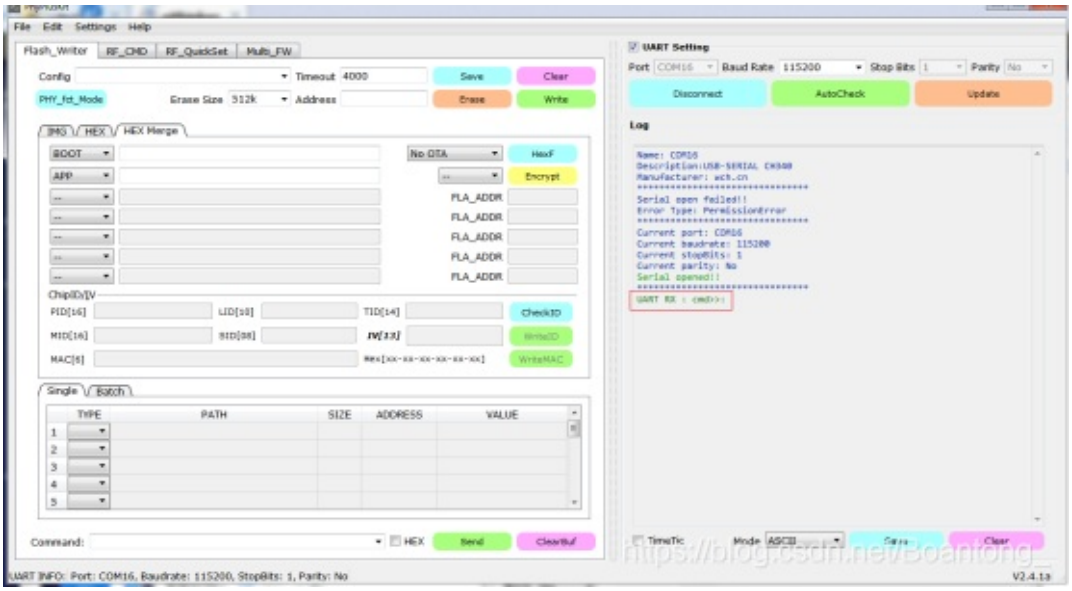
PB系列模组更新固件教程(PHY6212)

更新时间：2022-01-19 16:00

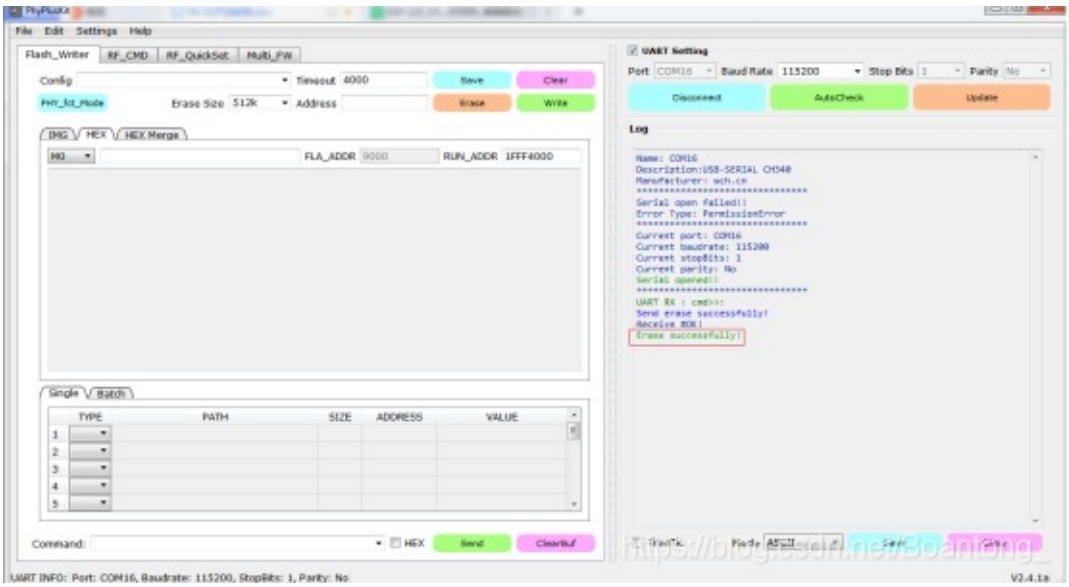
固件烧录

相关资料的获取：固件，烧写工具

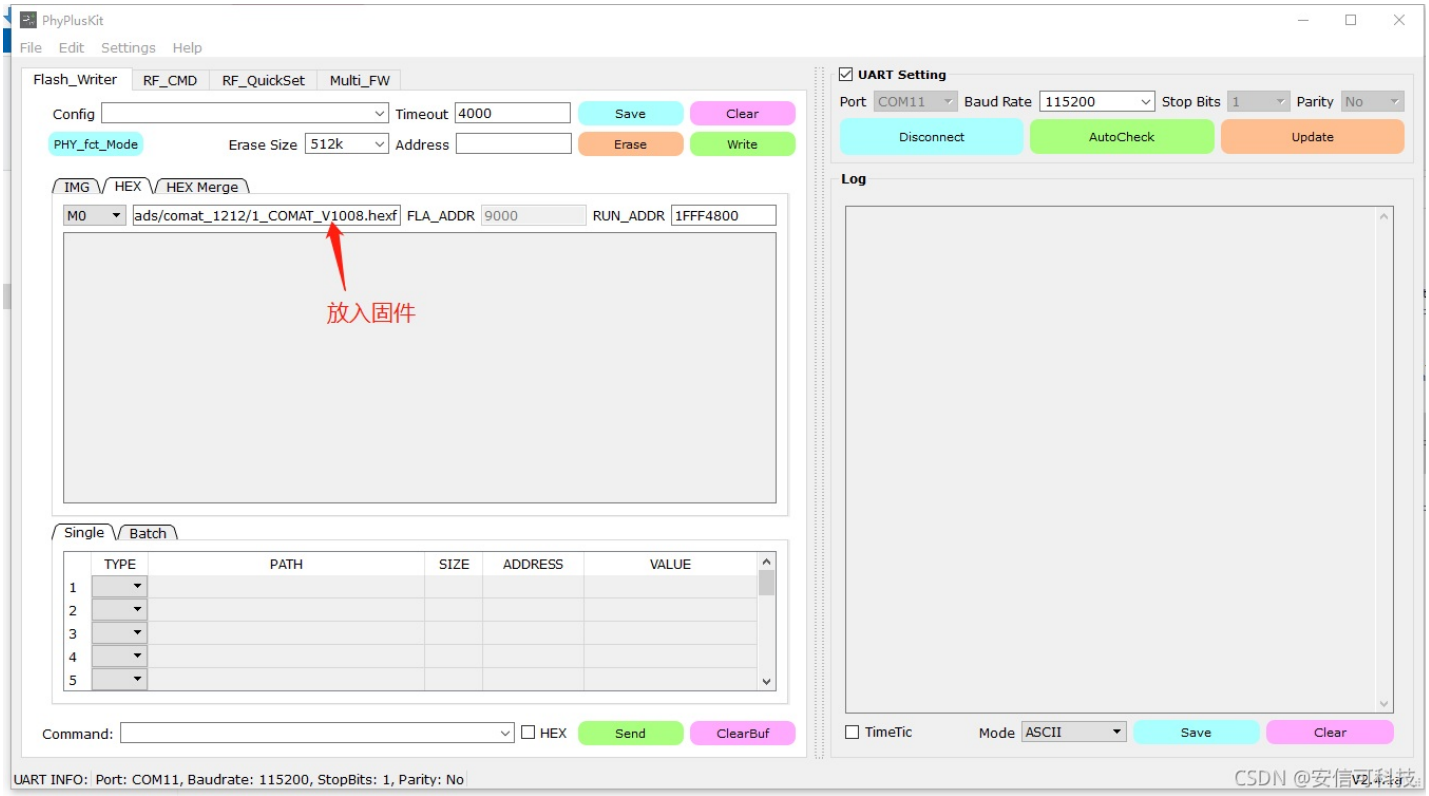
使用PlyPlusKit 工具擦除开发板或PB模组已经烧录的固件，RST和PROG按键同时按下：

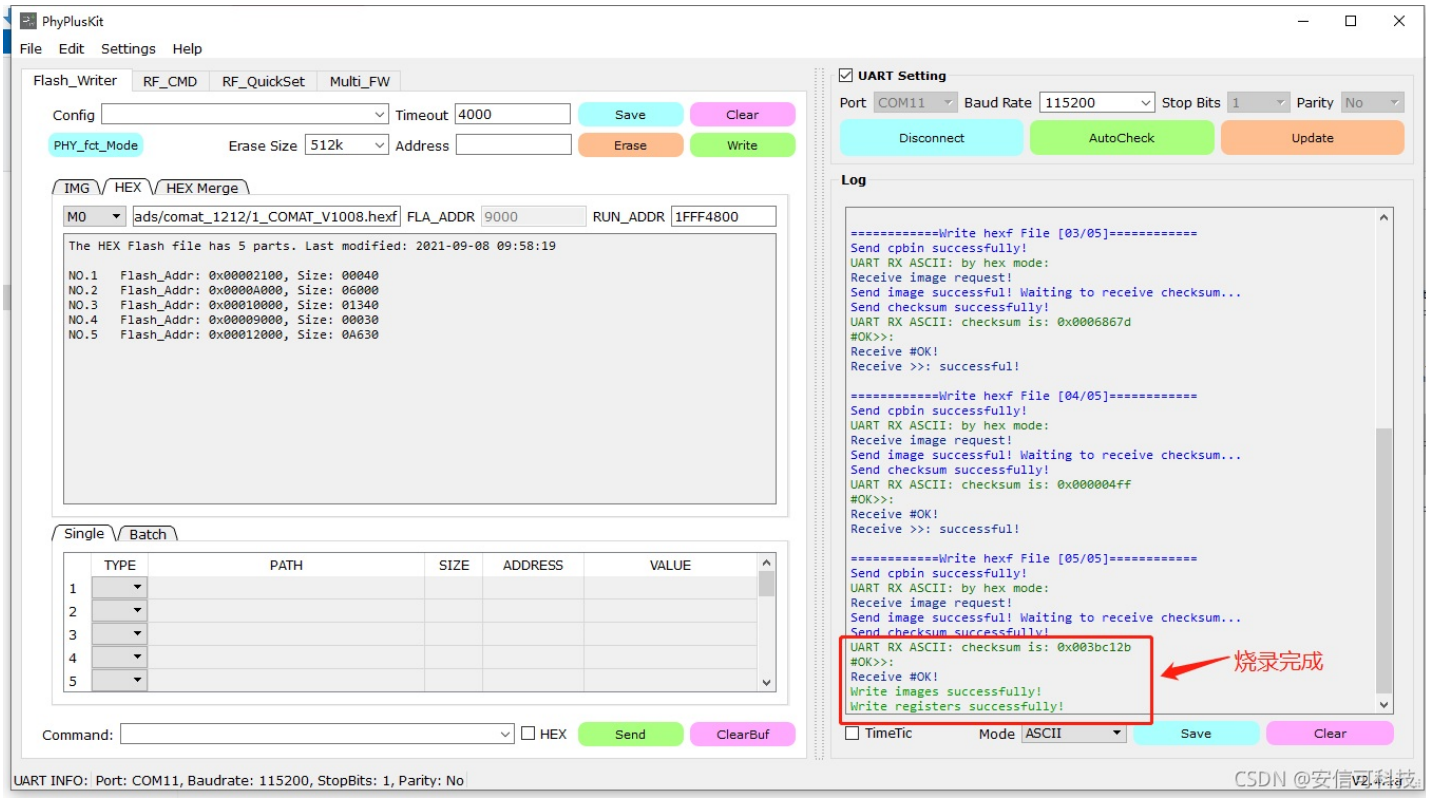


出现UART RX : cmd>>:信息，则表示进入了烧录模式，点击Erase,擦除成功如图：

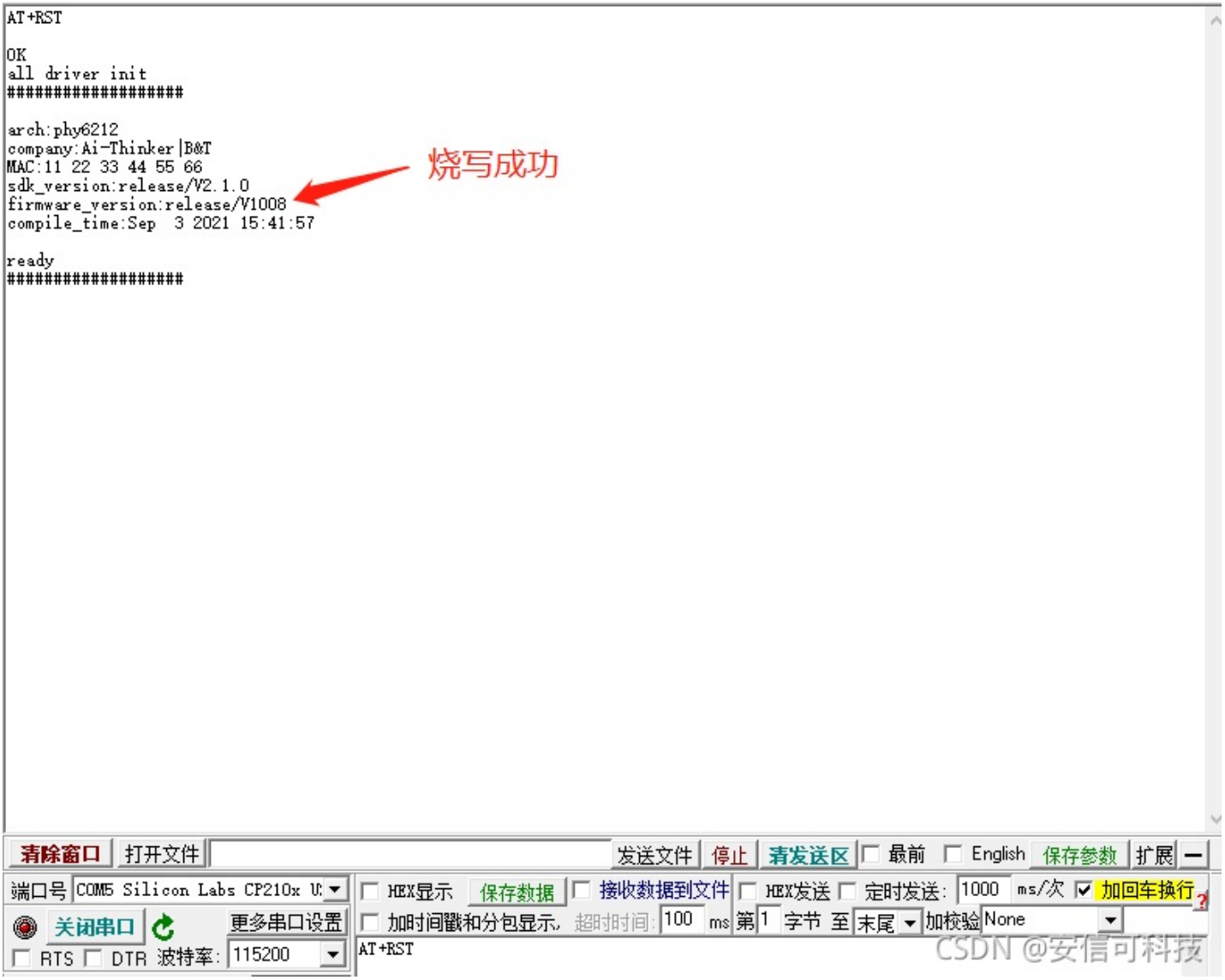


设置完成后直接点击Write 进行烧录，烧录完成后如下图





烧写完成后出现如下界面表示成功烧录



联系方式

公司官网：https://www.ai-thinker.com

开发资料：https://docs.ai-thinker.com

淘宝店铺：https://anxinke.taobao.com

天猫店铺：https://aithinker.tmall.com

商务合作：sales@aithinker.com

技术支持：support@aithinker.com

联系地址：深圳市宝安区西乡华丰智慧创新港C座403、408~410室

商务电话：0755-29162996

欢迎关注微信公众号 “安信可科技” ，干货实时推送！

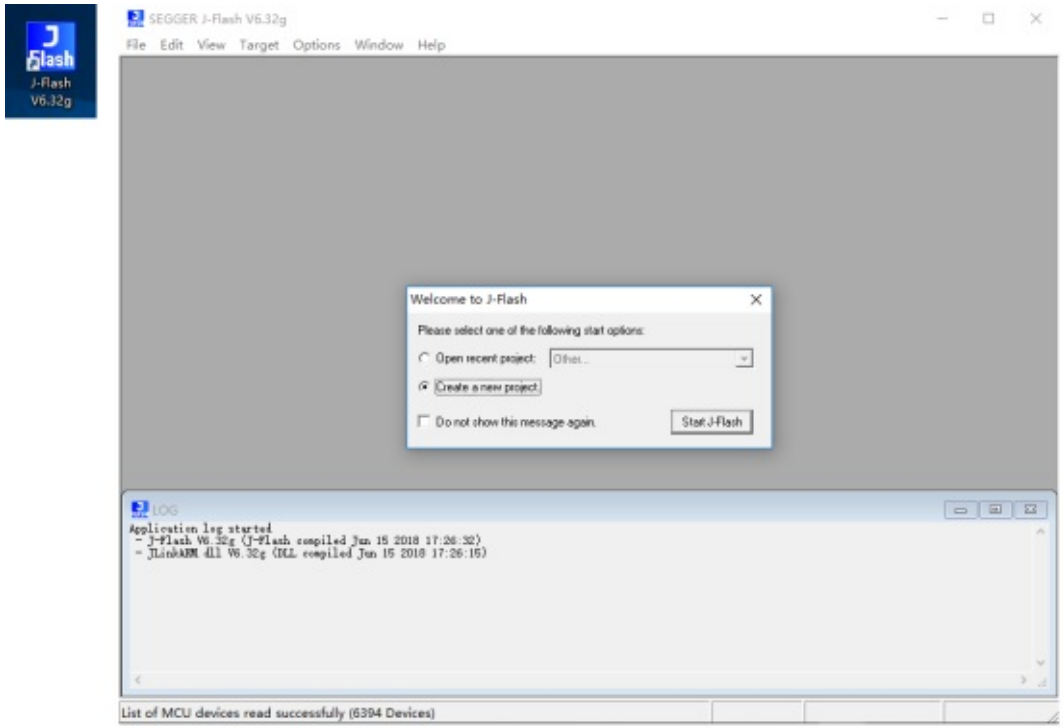
PB系列模组离线烧录器更新固件教程

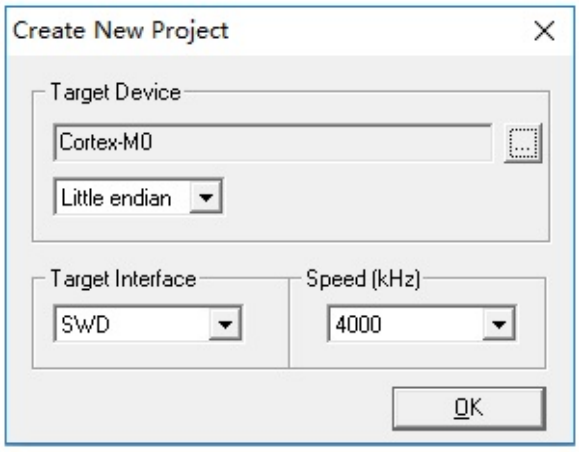
#更新离线烧录器底板的固件

使用JLINK将BootLoaderApplication.hex与MainApplication.hex文件烧录至烧录器,烧录工具：JLink_Windows_V632g。具体步骤如下：

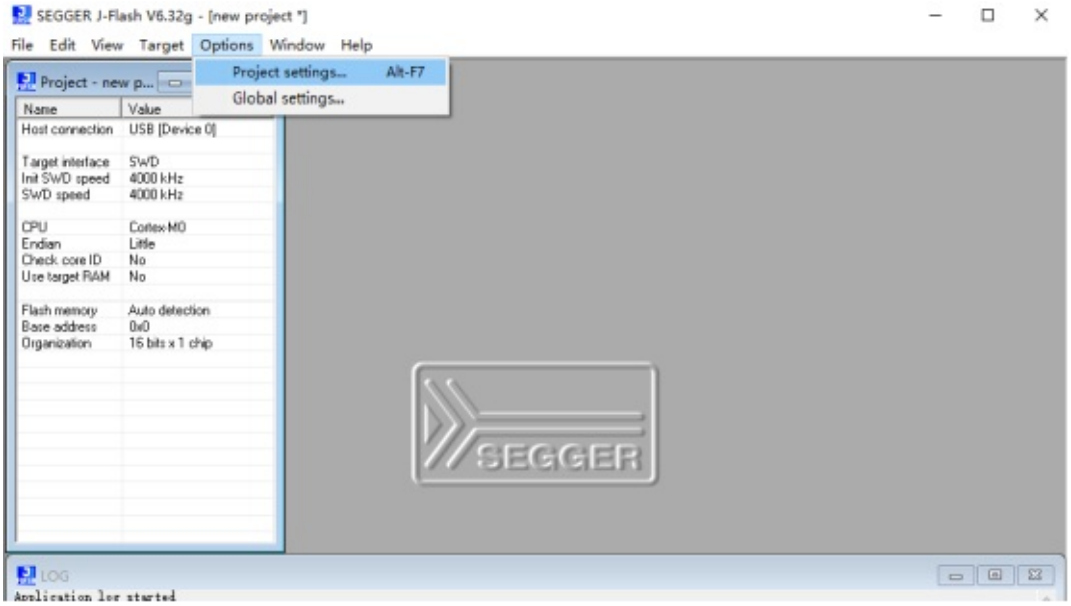
下面的a和b两个步骤可以省略，打开J-Flash 后，点击open recent project, 选中update_Offline_PhyWriter_Firmware.jflash即可(*.jflash文件已提供)

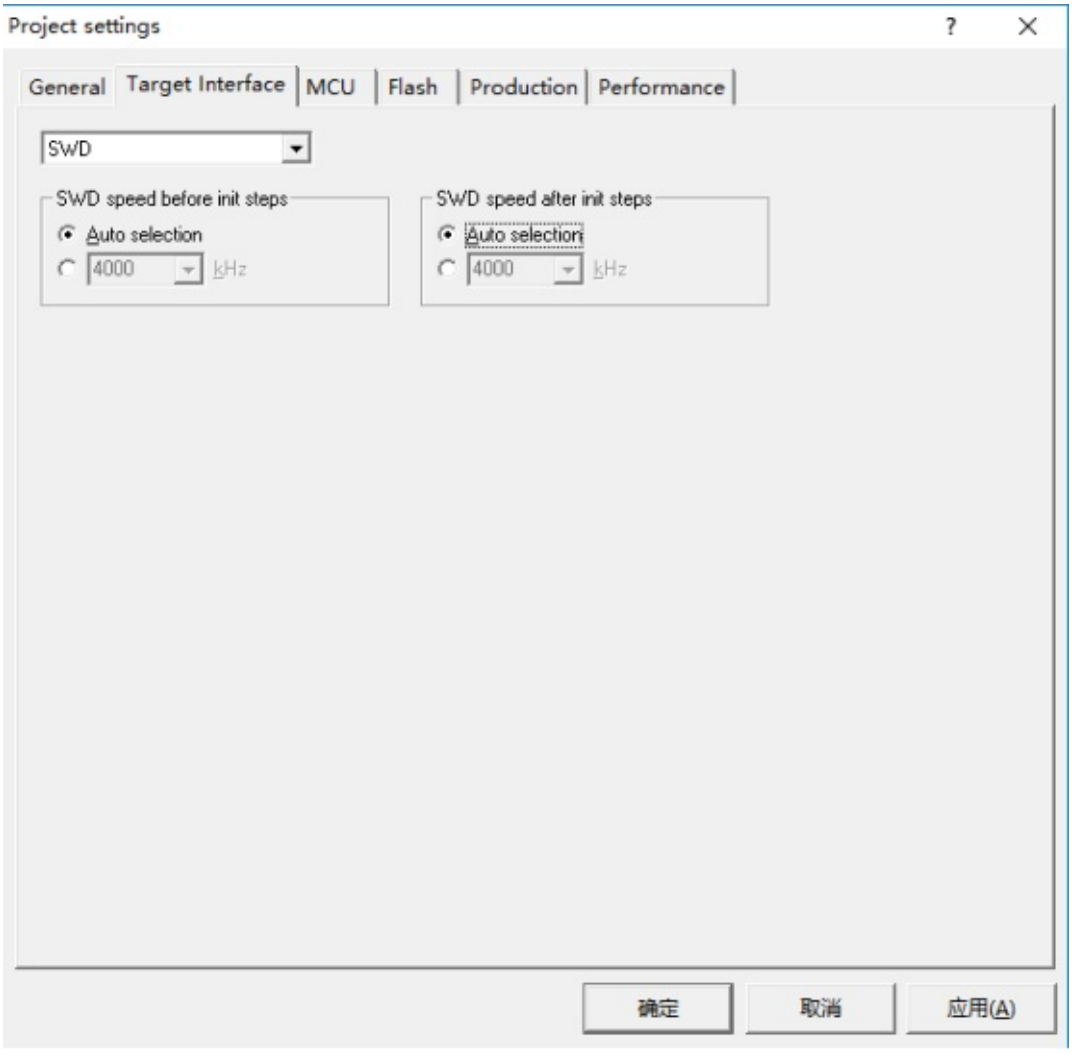
1.打开J-Flash V6.32g，新建工程

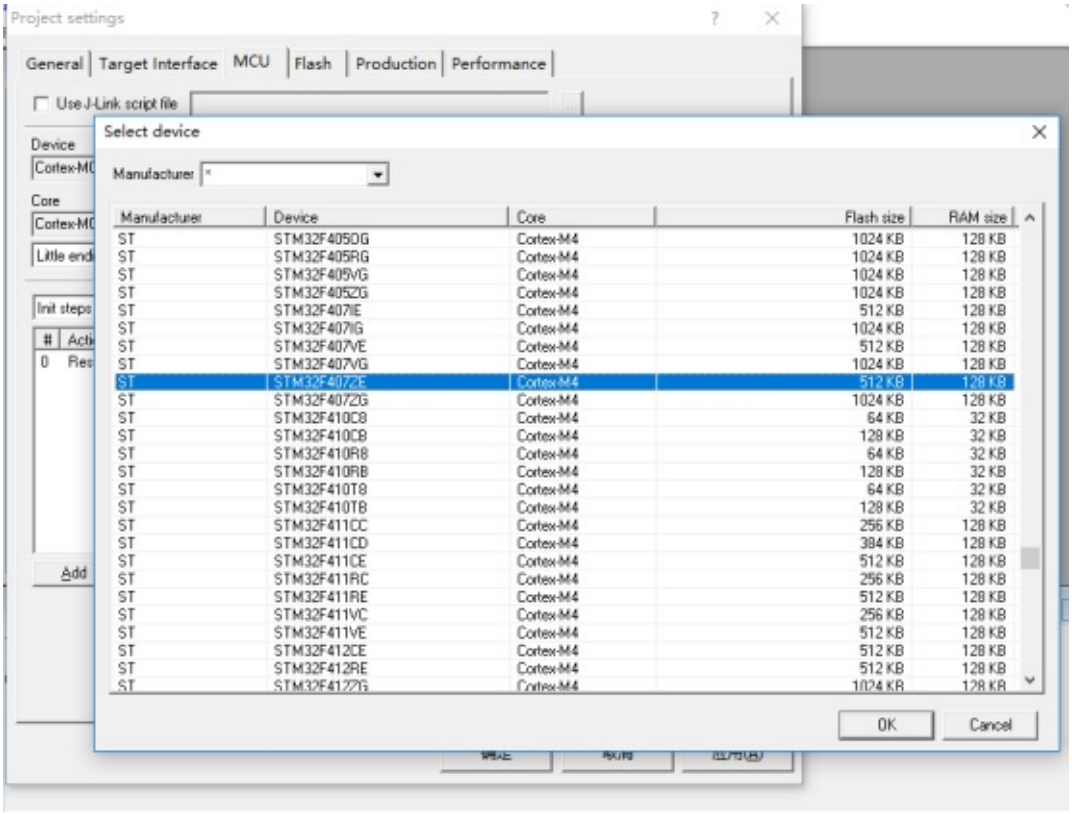




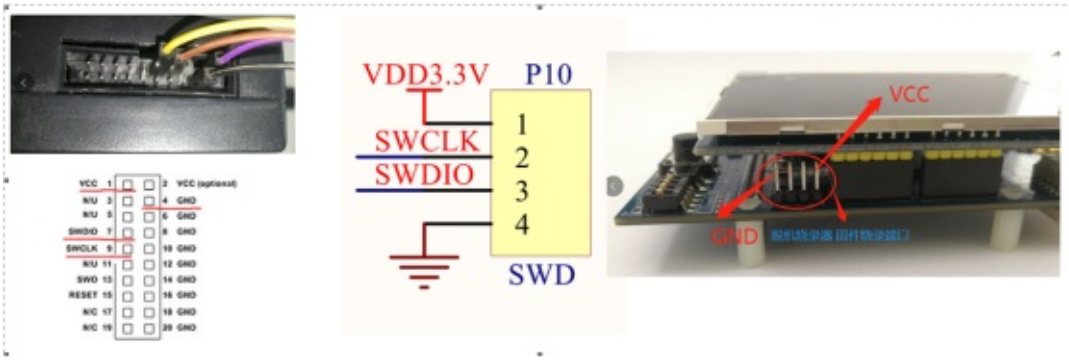
2.设置好JLINK


SEGGER J-Flash V6.32g - [new project *]
File Edit View Target Options Window Help
Project settings... Alt-F7
Global settings...
Project - new p...
Name
Value
Host connection
USB [Device 0]
Target interface
SWD
Init SWD speed
4000 kHz
SWD speed
4000 kHz
CPU
Cortex-M0
Endian
Little
Check core ID
No
Use target RAM
No
Flash memory
Auto detection
Base address
0x0
Organization
16 bits x 1 chip
SEGGER
LOG

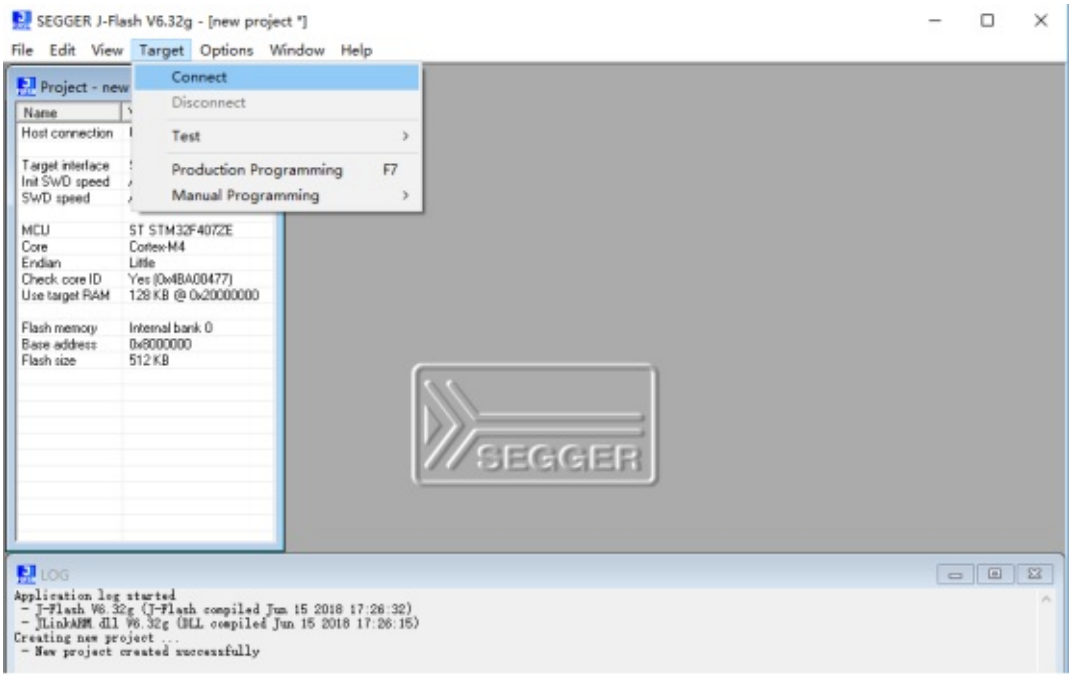

Project settings
General
Target Interface
MCU
Flash
Production
Performance
SWD
SWD speed before init steps
Auto selection
4000
kHz
SWD speed after init steps
Auto selection
4000
kHz
确定
取消
应用(A)

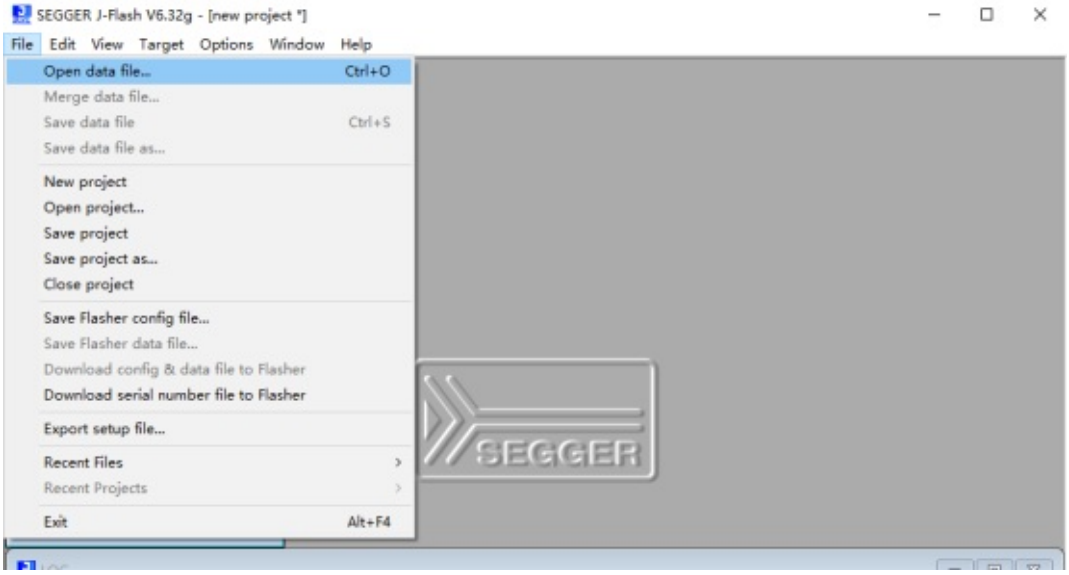


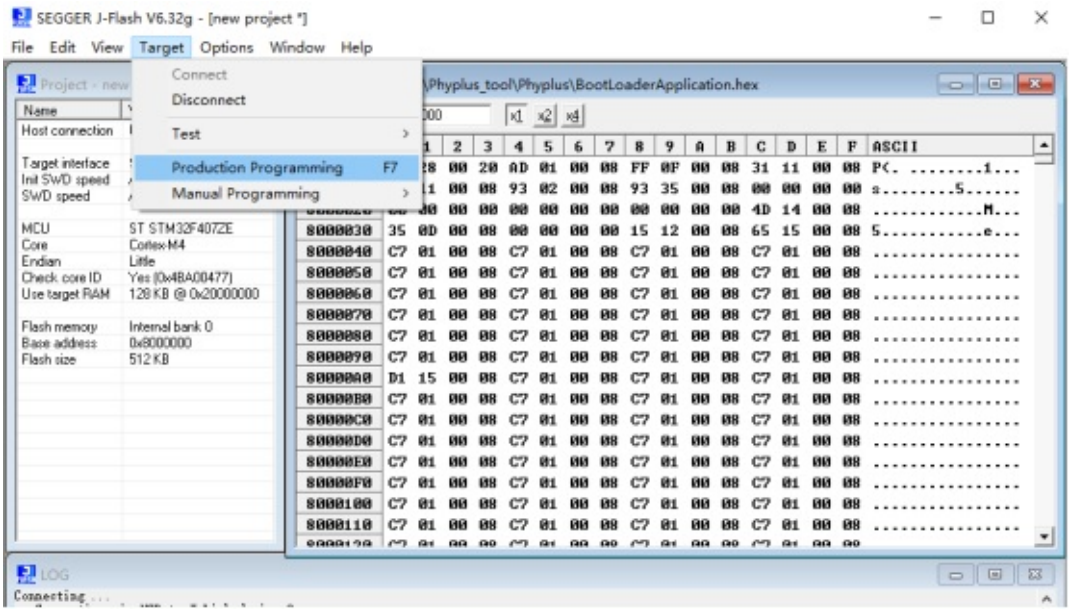
3.JLINK和烧录板的连接如下图所示：



4.Target->Connect，依次烧录BootLoaderApplication.hex与MainApplication.hex。每次烧录完成后都断开Miniusb和 Jlink一次。







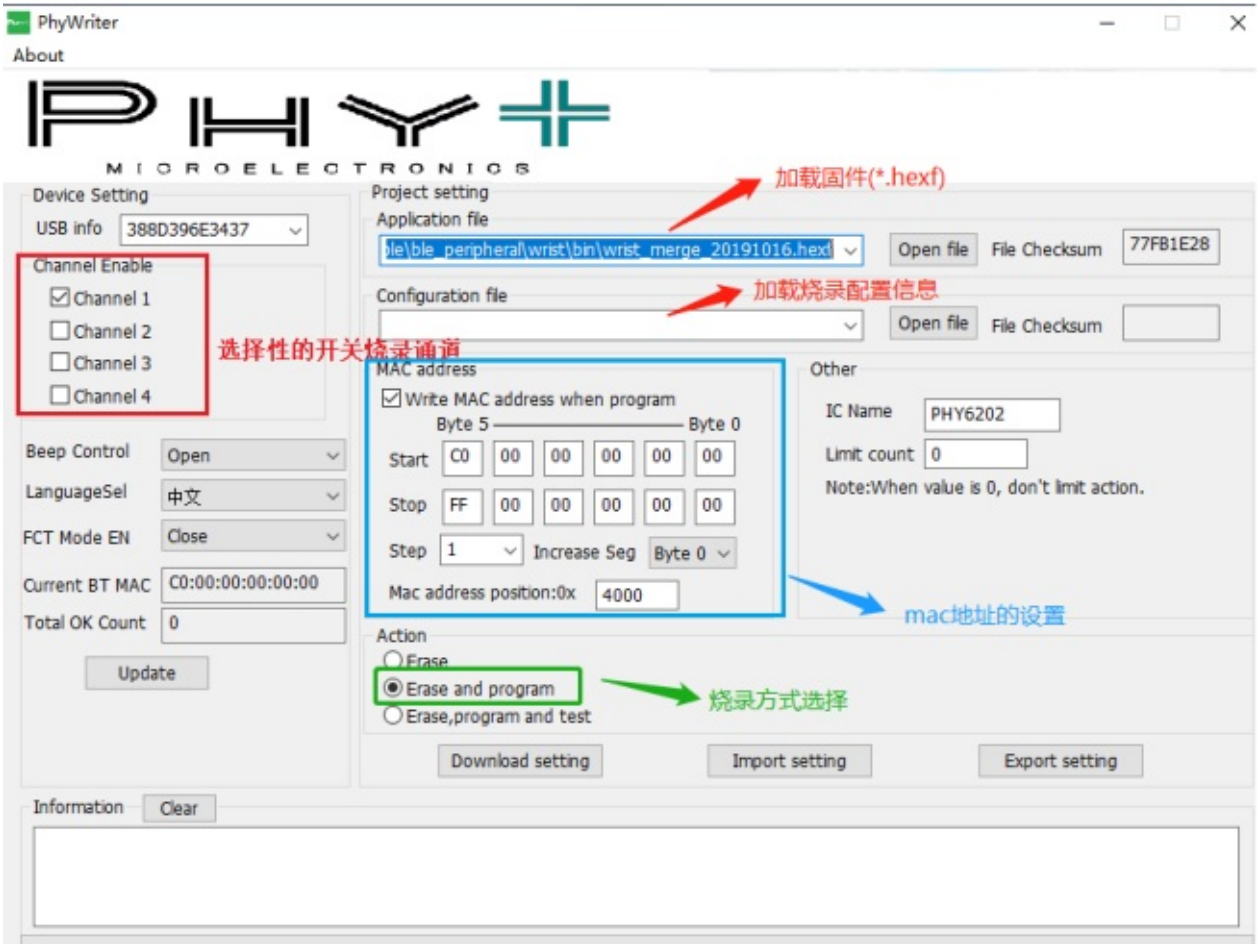
#烧录配置文件

打开PhyWriter.exe软件，下载烧录配置文件

1）、点击“open file”，选择烧录hexf文件

2）、配置MAC地址烧录

3）、点击“Download setting”，下载配置文件



#烧录操作

将PB模组的VDD、GND、TX、RX、TM分别接至烧录器中烧录接口的VDD、GND、RX、TX 、TM引脚上。目前脱机烧录板的线序是：

蓝线----EVB板的VDD

白线----EVB板的P9

黄线----EVB板的P10

红线----EVB板的GND

黑线----EVB板的TM

点击子板上的大圆形按键，或者点击母板的红色下载按键，即可烧录

联系方式

公司官网：https://www.ai-thinker.com

开发资料：https://docs.ai-thinker.com

淘宝店铺：https://anxinke.taobao.com

天猫店铺：https://aithinker.tmall.com

商务合作：sales@aithinker.com

技术支持：support@aithinker.com

联系地址：深圳市宝安区西乡华丰智慧创新港C座403、408~410室

商务电话：0755-29162996

欢迎关注微信公众号 “安信可科技” ，干货实时推送！

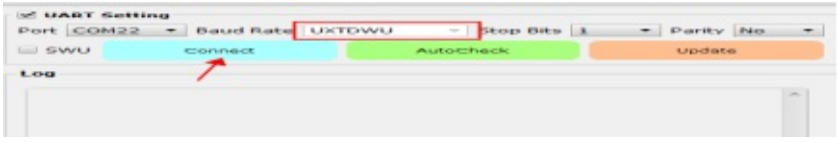
PB系列模组更新固件教程(PHY6252)

更新时间：2022-02-14 16:00

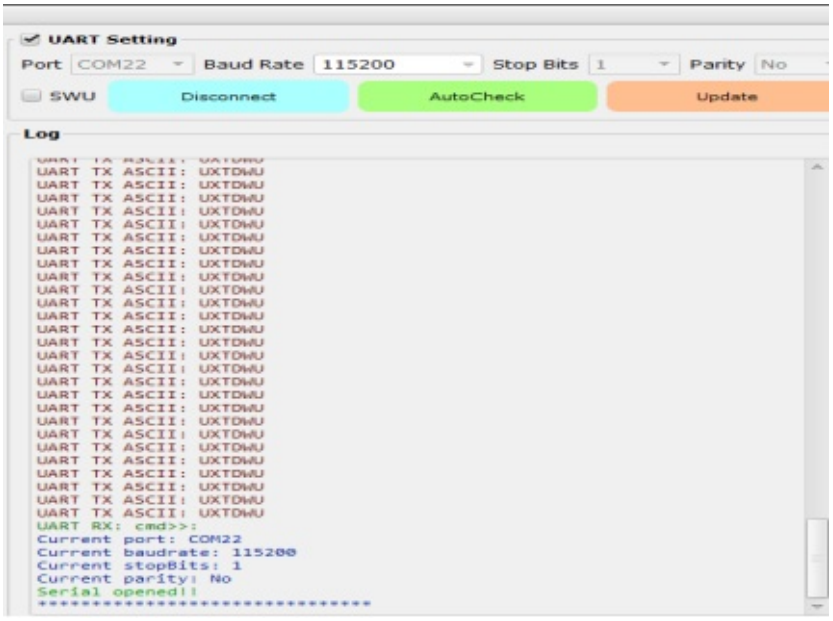
固件烧录

1、打开PhyPlusKit V2.4.5e.exe软件

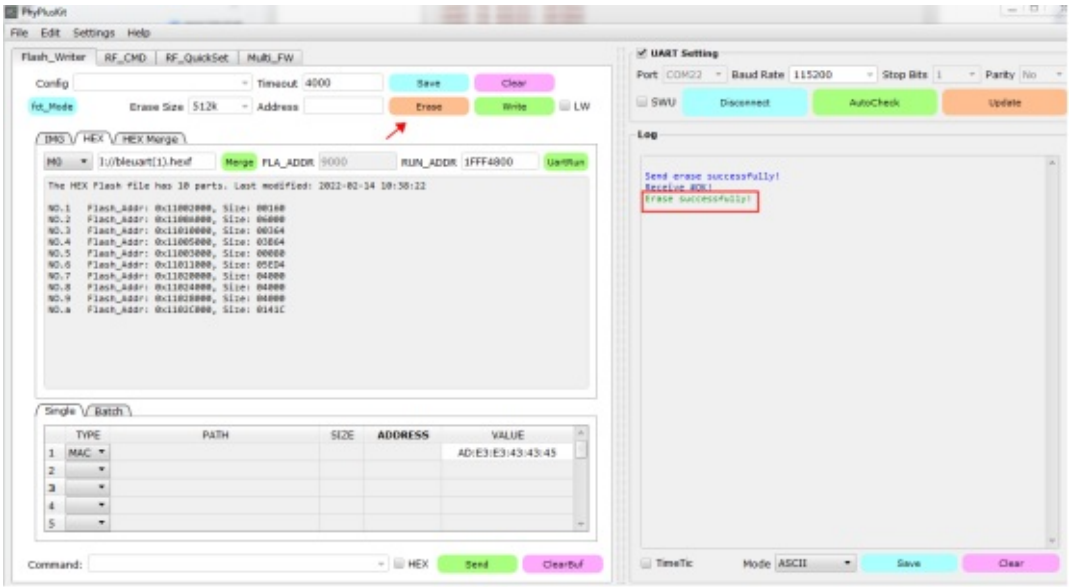
2、选择UXTDWU，点击Connet连接模块



3、开发板上按下RST按键或模组进行上电复位，等到串口出现“cmd>>”信息，即表示模组进入下载模式。



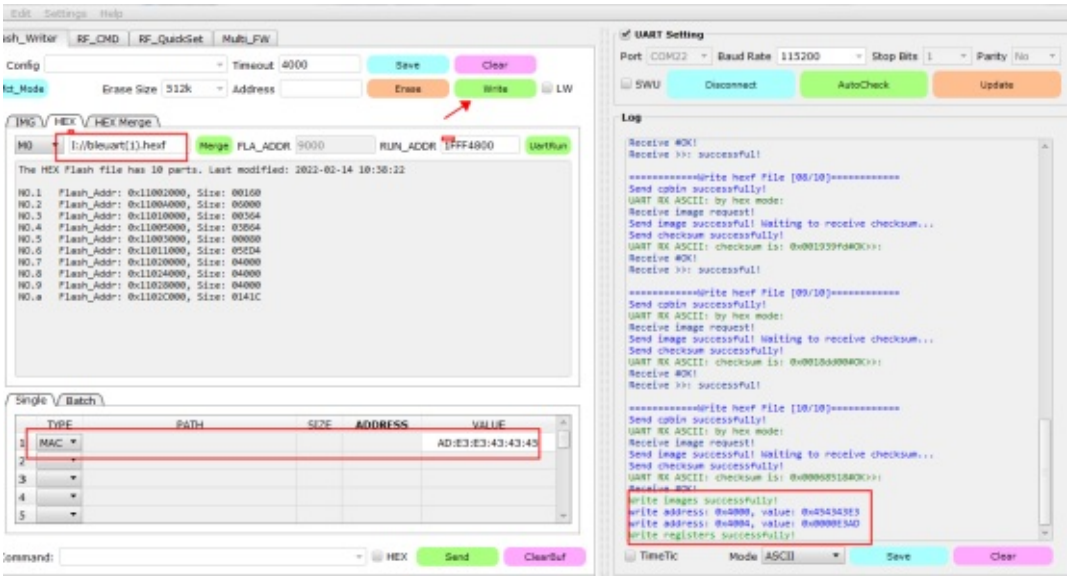
4、点击ERASE，进行擦除,串口出现“Erase successfully!”表示擦除成功！



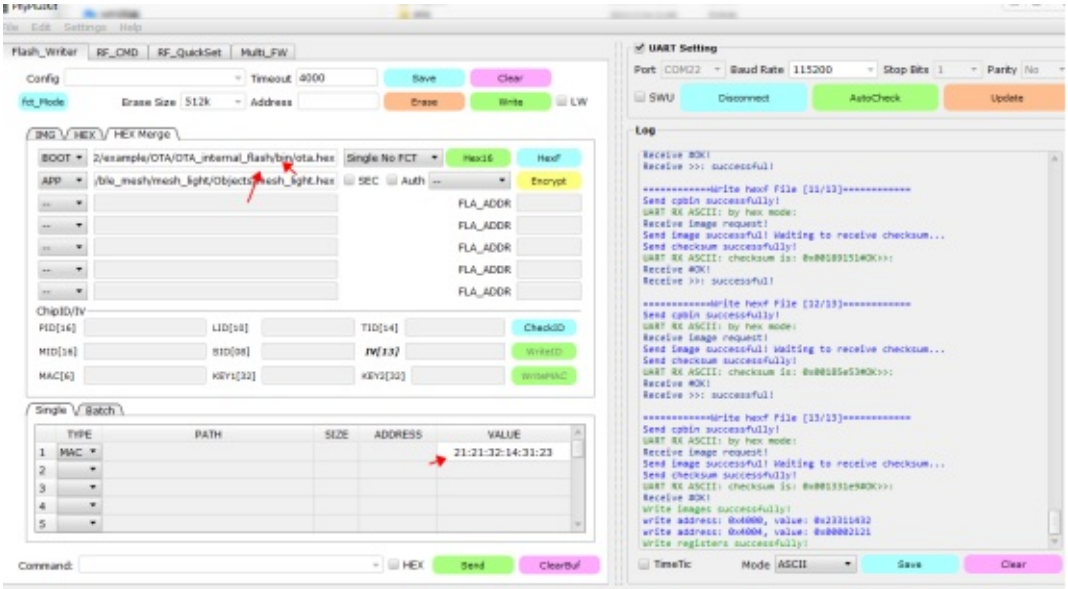
5、选择好所下载的HEX文件后，点击Write，串口出现“Write images successfully!”，“Write registers

successfully!”,即表示下载成功!

[已合并的固件]



[未合并的固件]



联系方式

公司官网：https://www.ai-thinker.com

开发资料：https://docs.ai-thinker.com

淘宝店铺：https://anxinke.taobao.com

天猫店铺：https://aithinker.tmall.com

商务合作：sales@aithinker.com

技术支持：support@aithinker.com

联系地址：深圳市宝安区西乡华丰智慧创新港C座403、408~410室

商务电话：0755-29162996

欢迎关注微信公众号 “安信可科技” ，干货实时推送！

【TB系列模组】二次开发

环境搭建

GPIO

概述

GPIO的驱动文件在drivers/8258/gpio_8258.h、gpio_8258.c,gpio_default_8258.h文件中

GPIO定义

825X系列芯片共有5组36个GPIO：

GPIO_PA0~GPIO_PA7

GPIO_PB0~GPIO_PB7

GPIO_PC0~GPIO_PC7

GPIO_PD0~GPIO_PD7

GPIO_PE0~GPIO_PE3

*具体对应到TB系列模组的IO个数，以各规格书说明的IO个数为准

PWM

中断

蓝牙Mesh基础入门

概述

SDK 给用户提供基于 SIG_mesh 协议应用开发的 demo code，用户可以在这些 demo code 基础上开发自己的应用程序

SDK 的文件架构

SDK 文件架构分为 app 应用层和 BLE&SIG_mesh 协议层。有几个主要的顶层文件夹:boot，common，drivers，homekit_src ，proj_lib，stack，vendor 。

boot:提供芯片的 bootloader，即 MCU 上电启动或 deepsleep 唤醒后的汇编处理过程，为后面 C 语言程序的运行搭建好环境。

drivers:提供与 MCU 紧密相关的硬件设置和外设驱动程序，如 clock、flash、i2c、usb、gpio、uart 等。

proj:提供 MCU 相关的外设驱动程序，如 flash，i2c，usb，gpio, uart 等。

proj_lib: 提供 MCU 运行所必需的库文件，包括 BLE 协议栈、RF 驱动、PM 驱动等，这部分是以库文件形式提供的，用户无法看到源文件，如 liblt_8258_mesh.a 为蓝牙协议栈的库文件，libsig_mesh.a 为 SIG_mesh 普通节点的库文件，libsig_mesh_LPN.a 为 SIG_mesh中的低功耗节点的库文件，libsig_mesh_prov.a为SIG_mesh中的provision节点的库文件。

stack:存放 BLE 协议栈相关的头文件。源文件被编译到库文件里面，对于用户是不可见的。

vendor:用于存放用户应用层代码。目前 vendor 目录下有:

——common:主要包含了 mesh/mesh_lpn/mesh_provision/mesh_switch 等共用的模块，例如 SIG mesh model 的处理，led 部分，出厂初始化 ，测试命令等模块。

——mesh/mesh_gw_node_homekit/mesh_lpn/mesh_provision/mesh_switch/

spirit_lpn 这几个文件夹的结构一样，每个文件夹对应一个应用类型，都包含了 app.c、app.h、app_att.c、app_config.h、main.c。app.c/app.h 主要是初始化和底层回调功能；app_att.c 是蓝牙 att 表的描述以及接口函数的说明；app_config.h 是定义工程中对应的宏和声明；main.c是主函数和中断函数的入口。

1.1.1 main.c

包括 main 函数入口，系统初始化的相关函数，以及无限循环 while(1)的写法，建议不要对此文件进行任何修改，直接使用固有写法。

```
int main (void)
{
FLASH_ADDRESS_CONFIG;
```

```
#if PINGPONG_OTA_DISABLE
ota_fw_check_over_write(); // 非 pingpong OTA 的 firmware copy
#endif
blc_pm_select_internal_32k_crystal(); //选择内部 32k rc 作为 32k counter 时钟源
cpu_wakeup_init();//MCU 最基本的硬件初始化
int deepRetWakeUp = pm_is_MCU_deepRetentionWakeup(); //判断是否从 deep retention 唤醒
rf_drv_init(RF_MODE_BLE_1M); //RF 初始化
gpio_init(!deepRetWakeUp); //gpio 初始化，user 在 app_config.h 中配置相关参数
clock_init(SYS_CLK_16M_Crystal);
if( deepRetWakeUp )
{
user_init_deepRetn ();//deep retention 醒来的快速初始化
}
else{
user_init_normal ();//ble 初始化，整个系统初始化，user 进行设定
}
irq_enable();
//开全局中断
while (1)
{
#if (MODULE_WATCHDOG_ENABLE)
wd_clear(); //clear watch dog
#endif
main_loop (); //包括 ble 收发处理、低功耗管理、mesh 和 user 的任务
}
}
```

1.1.2 app_config.h

用户配置文件，用于对整个系统的相关参数进行配置，包括 BLE 相关参数、GPIO 的配置、PM 低功耗管理的相关配置等。后面介绍各个模块时会对 app_config.h 中的各个参数的含义进行详细说明。

1.1.3 BLE stack entry

Telink BLE SDK 中 BLE stack 部分 code 的入口函数有两个:

1. main.c 文件 irq_handler 函数中 BLE 相关中断的处理入口 irq_blt_sdk_handler。

   ```
   _attribute_ram_code_ void irq_handler(void)
   ```

```
{
.....
irq_blt_sdk_handler ();
.....
}
```

2. application file main_loop 中 BLE 逻辑和数据处理的函数入口 blt_sdk_main_loop。

```
void main_loop (void)
{
mesh_loop_proc_prior();//优先处理,主要是跳过 mailoop 8269 的 10ms interval
////////////////////// BLE entry /////////////////////////////
blt_sdk_main_loop();
////////////////////// UI entry /////////////////////////////
factory_reset_cnt_check();//五次上电执行 factory reset 的实现
mesh_loop_process();//mesh 相关的 loop 函数
.....
////////////////////// PM configuration ////////////////////
.....
}
```

1.2 Demo Project

TeLink SIG MESH SDK 给用户提供了多个 BLE demo。

用户通过软硬件 demo 的运行，可以观察到直观的效果。用户也可以在 demo code 上进行修改，完成自己的应用开发。

mesh 的 demo 及区别如下表所示。

Demo	Vendor folder	Application	Mesh Feature
8258_mesh/ 8269_mesh	.\mesh	CT/HSL light 等	Relay, friend, proxy
8258_mesh_LPN\ 8269_mesh_LPN	.\mesh_lpn	LPN	LPN, proxy
8258_mesh_gw\ 8269_mesh_gw	.\mesh_provision	Gateway provisioner	adv provisioner, Relay, friend
8258_mesh _gw_node	.\ mesh_provision	Gateway+light node	adv provisioner, Relay, friend, proxy
8258_mesh_switch\ 8269_mesh_switch	.\mesh_switch	遥控器应用	proxy
8258_spirit_LPN	.\ spirit_lpn	天猫精灵自定义 LPN 模式	proxy
8258_gw_node_ho mekit	.\mesh_gw_node _homekit	Gateway+light node+homekit	adv provisioner, Relay, friend, proxy

“8258_mesh”:是普通 SIG MESH 节点的编译工程，可以被 provisioner 配置网络，有 relay，friend，proxy 功能，无 provision 功能。

“8258_mesh_LPN:是 LPN MESH 节点的编译工程，通过 friend ship 接收 message，无 relay，friend，proxy，provision 功能。

“8258_mesh_gw”:是 gateway provisioner 节点的编译工程，有 adv provisioner 功能，可以配置其他节点，同时也有 relay，friend 功能。

“8258_mesh_switch”:是普通遥控器(switch) MESH 节点的编译工程。该遥控器组网后，为了实现更低的功耗需求，基本上只发命令，不接收命令。无 relay，friend 功能，

“8258_gw_node_homekit”:具有 gateway adv provisioner + mesh node + homekit 这三种模式的功能。

“8258_spirit_LPN” 编译选项:是天猫精灵自定义 LPN 模式。

1.3 LIGHT_TYPE_SEL 介绍

该宏用于选择预先配置好的一些常用的产品类型。

```
#define LIGHT_TYPE_NONE 0
#define LIGHT_TYPE_CT 1
#define LIGHT_TYPE_HSL 2
```

#define LIGHT_TYPE_XYL 3

#define LIGHT_TYPE_POWER 4

#define LIGHT_TYPE_CT_HSL 5

#define LIGHT_TYPE_DIM 6 // only single PWM

#define LIGHT_TYPE_PANEL 7 // only ONOFF model

#define LIGHT_TYPE_LPN_ONOFF_LEVEL 8 // only ONOFF , LEVEL model

LIGHT_TYPE_CT

CT 是色温灯的简写，对应的产品类型是色温灯，包含色温相关的 model，比如 Light CTL Server，Light CTL Setup Server，Light CTL Temperature Server 以及对应的 extend model，比如:Generic OnOff Server，Generic Level Server，Light Lightness Server 等。

LIGHT_TYPE_HSL

对应的产品类型是彩色灯(HSL 灯)，包含 Light HSL Server，Light HSL Hue Server，Light HSLSaturation Server，Light HSL Setup Server，以及对应的 extend model，比如:Generic OnOff Server，Generic Level Server，Light Lightness Server 等。

LIGHT_TYPE_XYL

对应的产品类型是 XYL 灯，包含 Light xyL Server，Light xyL Setup Server，以及对应的 extend model，比如:Generic OnOff Server，Generic Level Server，Light Lightness Server 等。

LIGHT_TYPE_POWER

对应的产品类型是 power adapter 等，包含 Generic Power Level Server，Generic Power Level Setup Server，以及对应的 extend model，比如:Generic OnOff Server，Generic Level Server 等LIGHT_TYPE_CT_HSL对应的产品类型是 色温灯+彩色灯(HSL 灯)，包含色温以及 HSL 对应的 model，以及 GenericOnOff Server，Generic Level Server，Light Lightness Server 等，其中色温灯珠和 HSL 灯珠共用一个 lightness 和 onoff 值。另外，同一时间只有一种灯珠在点亮。

LIGHT_TYPE_DIM

对应的产品类型是调光灯，包含 Light Lightness Server，Light Lightness Setup Server 以及对应的 extend model，比如:Generic OnOff Server，Generic Level Server 等。

LIGHT_TYPE_PANEL

对应的产品类型是开关面板，该面板处于server角色，也就是被app等设备控制并执行onoff 切换。包含 Generic OnOff Server model。默认开关个数是 3 个（由 LIGHT_CNT 定义）。

LIGHT_TYPE_LPN_ONOFF_LEVEL

对应的产品类型是 LPN 设备，默认包含 Generic OnOff Server model 等，mesh OTA model 关闭。主要用于 demo LPN 的功能。

Note:开发 LPN 设备时，要注意 825x retention RAM 的使用不能超过 32K。

Note:Node 最终包含的所有 model，都会显示在 composition data 中(全局变

量:model_sig_cfg_s_cps)。

1.4 产品版本号(VID)产品类型(PID)设置

配置文件:\vendor\common\version.h，该文件在汇编代码中也会使用到。

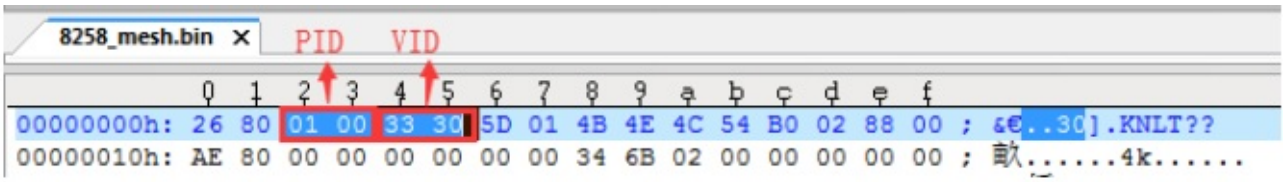
示例如下:

#define MESH_PID_SEL (LIGHT_TYPE_SEL)

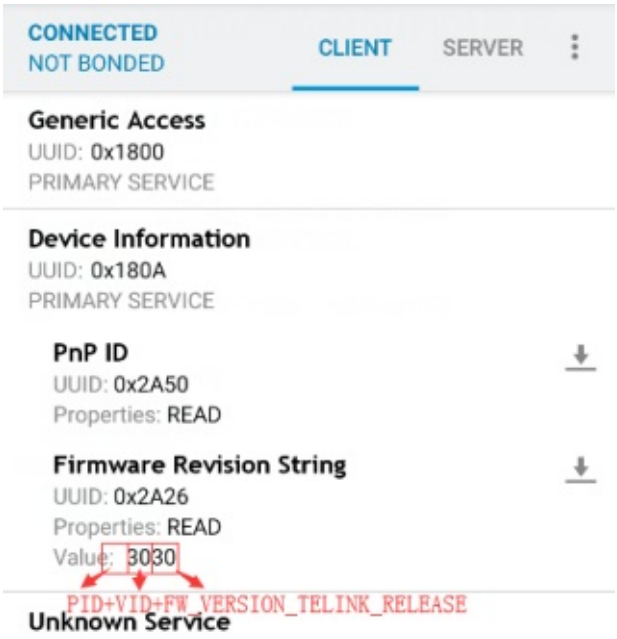
#define MESH_VID (VERSION_GET(0x33, 0x30))

#define FW_VERSION_TELINK_RELEASE (VERSION_GET(0x33, 0x30))

1. composition data 中的 PID 和 VID 分别取自这里的 MESH_PID_SEL，MESH_VID。
2. firmware 文件的第 3-6 字节分别表示这里的 PID 和 VID



3. 在通用 APP 的 ATT 页面的显示如下:



MESH_PID_SEL(PID): 因为通用 APP 上显示是以 ASCII 码解析，所以字符”0x00 0x01”这两个字符不可见。客户按实际需求修改为自己的 PID。

MESH_VID(VID):“0x33, 0x30”按 ASCII 显示为“30”。 客户按实际需求改为自己的 PID。

FW_VERSION_TELINK_RELEASE:“0x33, 0x30”按 ASCII 显示为“30”。这个是 telink release SDK 时的版本号，客户不应该修改

4. 开发者按实际需求，修改 MESH_PID_SEL 和 MESH_VID 的值即可。

1.5 Mesh 应用收发包处理

1.5.1 发包函数

节点之间发包

调用 mesh_tx_cmd2normal_primary()，该函数具体用法见后面介绍。此方式发送的数据遵循 SIG mesh 协议。access_cmd_onoff()等命令是对 mesh_tx_cmd2normal_primary()进行封装而来。

开发者可以启用 sim_tx_cmd_node2node()来进行发送命令的演示(该函数默认是屏蔽的)，

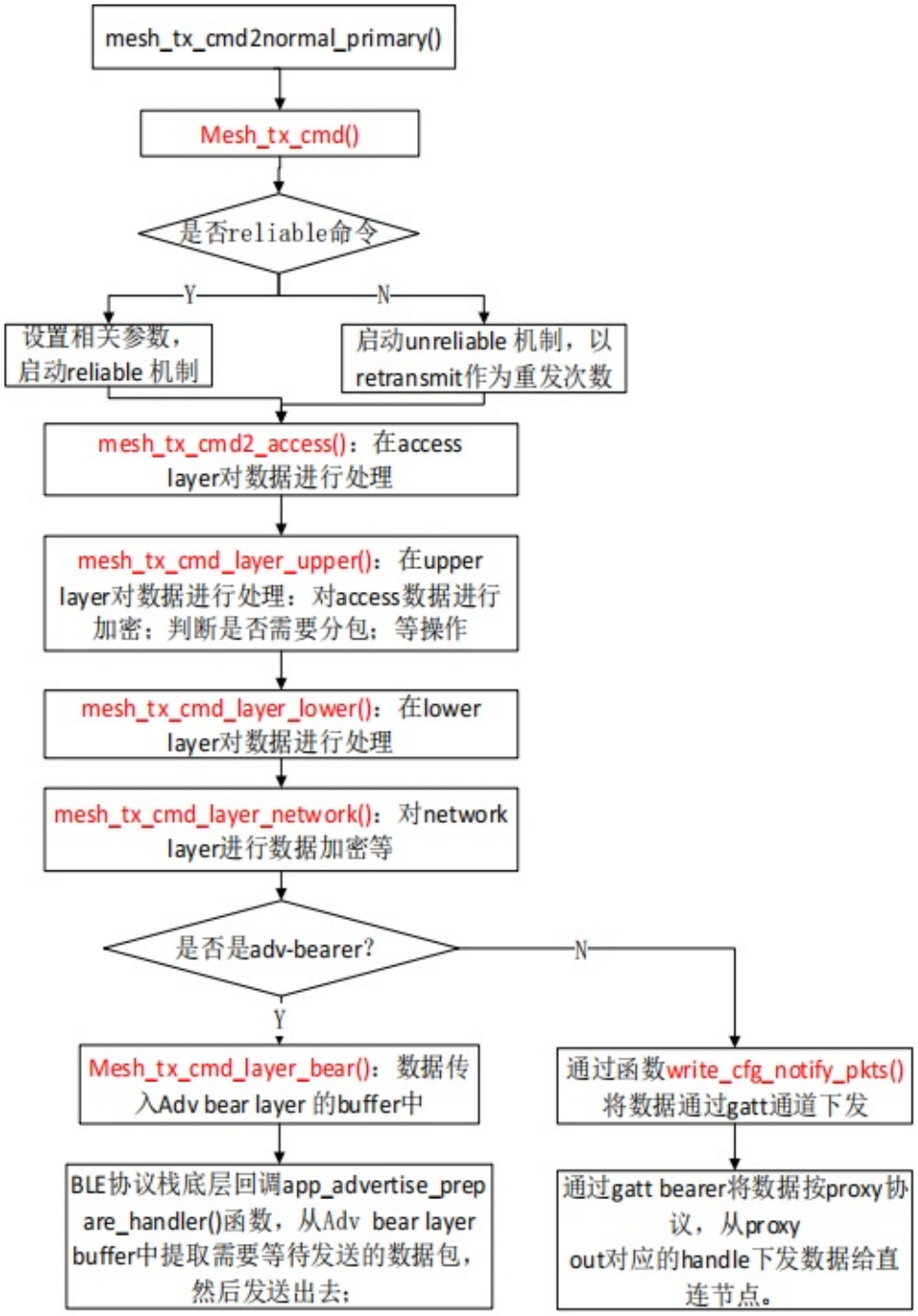
效果是:上电后，每隔 3 秒自动交替发送 ON/OFF 命令。详见后续对该函数的说明。

直连节点发给 master调用 bls_att_pushNotifyData()，具体用法请参考<AN_17092701_Telink 826x BLE SDK Developer Handbook> 3.4.3.10 节，此方式用于发送客户任意自定义格式的数据。但是没有 mesh 功能，一般不用。

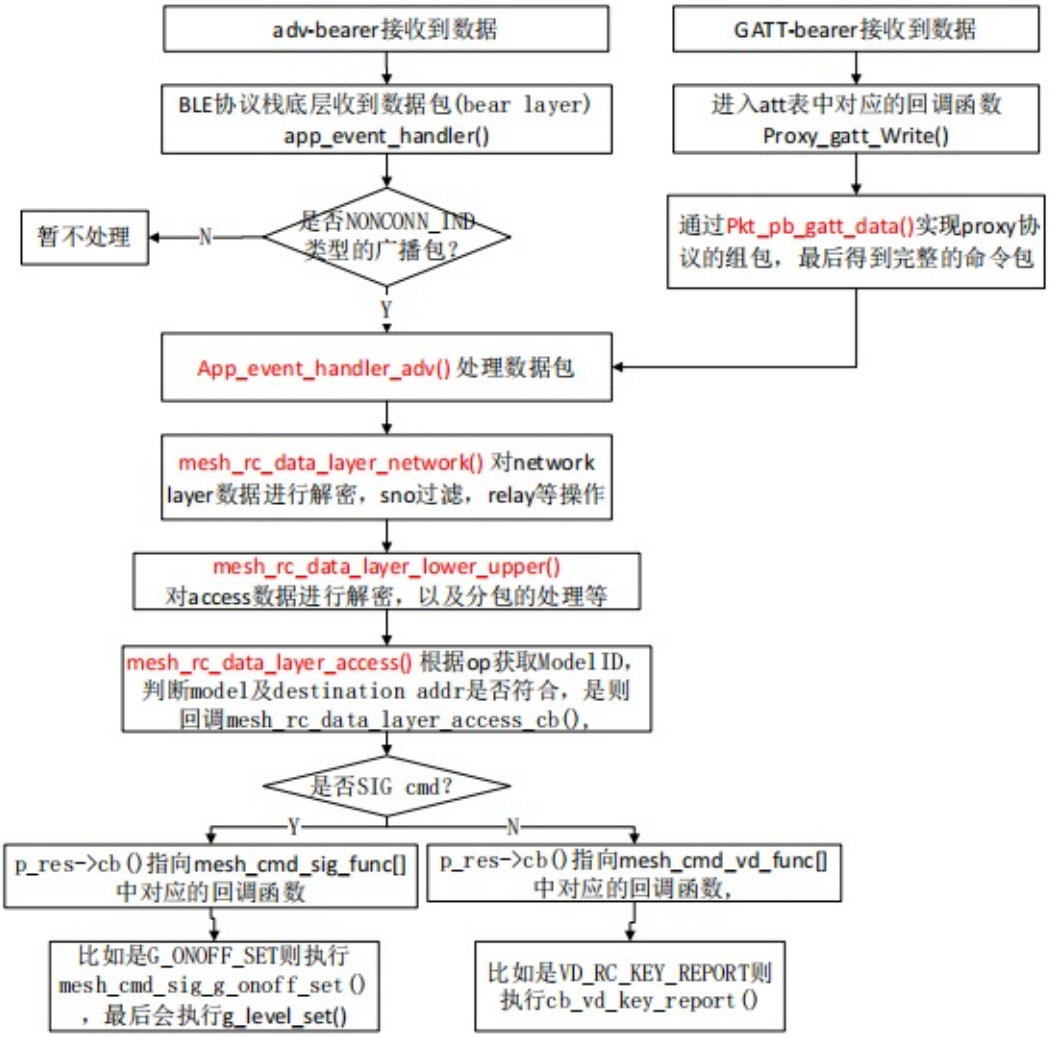
注意:需要新增 UUID，然后才能用该方式，否则可能会和当前 UUID 的协议冲突。新增 UUID的方法，可以在 my_Attributes_provision[]/my_Attributes_proxy[]/my_Attributes[]定义，自己订制 BLE service

1.5.2 发包流程图简介

注:红色字体为 library 函数。


mesh_tx_cmd2normal_primary()
Mesh_tx_cmd()
是否reliable命令
Y
N
设置相关参数，启动reliable 机制
启动unreliable 机制，以retransmit作为重发次数
mesh_tx_cmd2_access()：在access layer对数据进行处理
mesh_tx_cmd_layer_upper()：在upper layer对数据进行处理：对access数据进行加密；判断是否需要分包；等操作
mesh_tx_cmd_layer_lower()：在lower layer对数据进行处理
mesh_tx_cmd_layer_network()：对network layer进行数据加密等
是否是adv-bearer?
N
Y
Mesh_tx_cmd_layer_bear()：数据传入Adv bear layer 的buffer中
通过函数write_cfg_notify_pkts()将数据通过gatt通道下发
BLE协议栈底层回调app_advertise_prepare_handler()函数，从Adv bear layer buffer中提取需要等待发送的数据包，然后发送出去；
通过gatt bearer将数据按proxy协议，从proxy out对应的handle下发数据给直连节点。

1.5.3 收包流程图简介



1.5.4 收包回调函数的介绍

generic model

generic model 的接口在文件 vendor/common/generic_model.c，回调函数见结构体 mesh_cmd_sig_func[]。比如 generic on message，收到这个命令后，按上面介绍的“收包流程图”，走到 mesh_cmd_sig_g_onoff_set()这个函数，用户在这个函数里面实现开关灯或设置渐变参数,渐变效果在 light_transition_proc 处理，处理间隔为 LIGHT_ADJUST_INTERVAL(20ms)。

vendor model

vendor model的接口在文件vendor/common/vendor_model.c。参考mesh_cmd_vd_func[]，用户可以自行添加需要的 vendor command，收到这样的命令，按 SIG mesh spec 流程，会走到对应的 callback 函数，比如 cb_vd_key_report()。

1.5.5 SIG_mesh 信道

SIG_mesh 的通信信道分为两种:

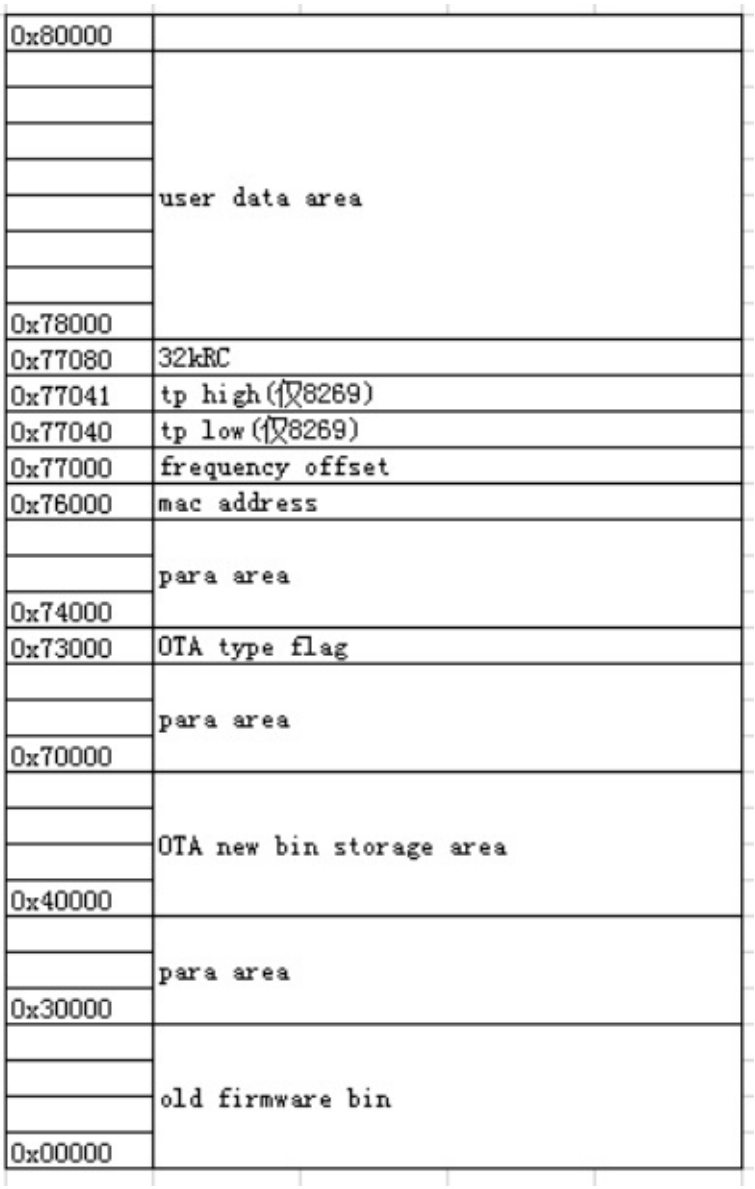
一种是 adv-bearer，是通过 adv 机制实现相互通信的，通信双方不需要建立 BLE 的连接，

通讯 channel 是标准的 37/38/39；

另外一种是通过 gatt-bearer，通过建立 BLE 连接实现通信, 通讯 channel 是 1-36。

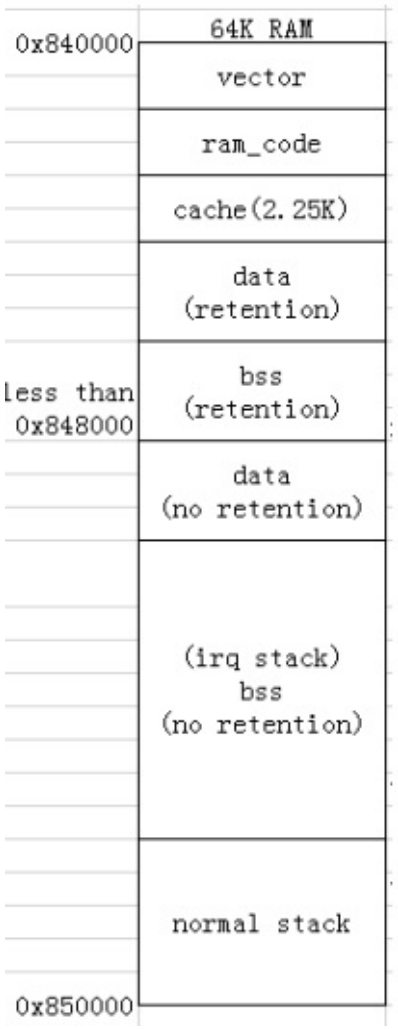
2. MCU 基础模块

2.1.1 Flash map 介绍(以 512K flash 为例)



2.1.2 RAM map(以 825x 64K 为例)

对应的配置文档，详见 ./boot.link



data(retention):包含有_attribute_data_retention_前缀的变量，以及所有初始化值不等于 0 的全局变量,默认为 retention data 属性；

bss(retention):包含有_attribute_bss_retention_前缀的变量，以及所有初始化值 等于 0 的全局变量,默认为 retention bss 属性；

data(no retention):有 _attribute_no_retention_data_ 前缀的全局变量

bss(no retention):有 _attribute_no_retention_bss_ 前缀的全局变量。并且包含 irq_stack，其 size 由 IRQ_STK_SIZE 定义，默认 0x300，demo SDK 使用量小于 0x200，

normal stack:bss 之后，64KRAM 之前都属于 normal stack。初始化值为 0xffffffff，目前为了加快 RAM 初始化速度，只初始化 3K 的 RAM。

注意:_attribute_bss_retention_和_attribute_no_retention_bss_这两个前缀，对于初始化不为0 的变量不能使用。否则初始化值无效，默认为 0。

2.1.3 堆栈(stack) 溢出的判断

首先，要确保，bss(no retention)的末尾地址要小于 RAM 的末尾地址，也就是要留有空间给 normal stack。End of bss(no retention) 可以通过 tdebug Tool 对变量按地址进行排序，最后一个变量之后的地址，就是末尾地址。

另外也可以通过 编译生成的 *.lst 文档计算:

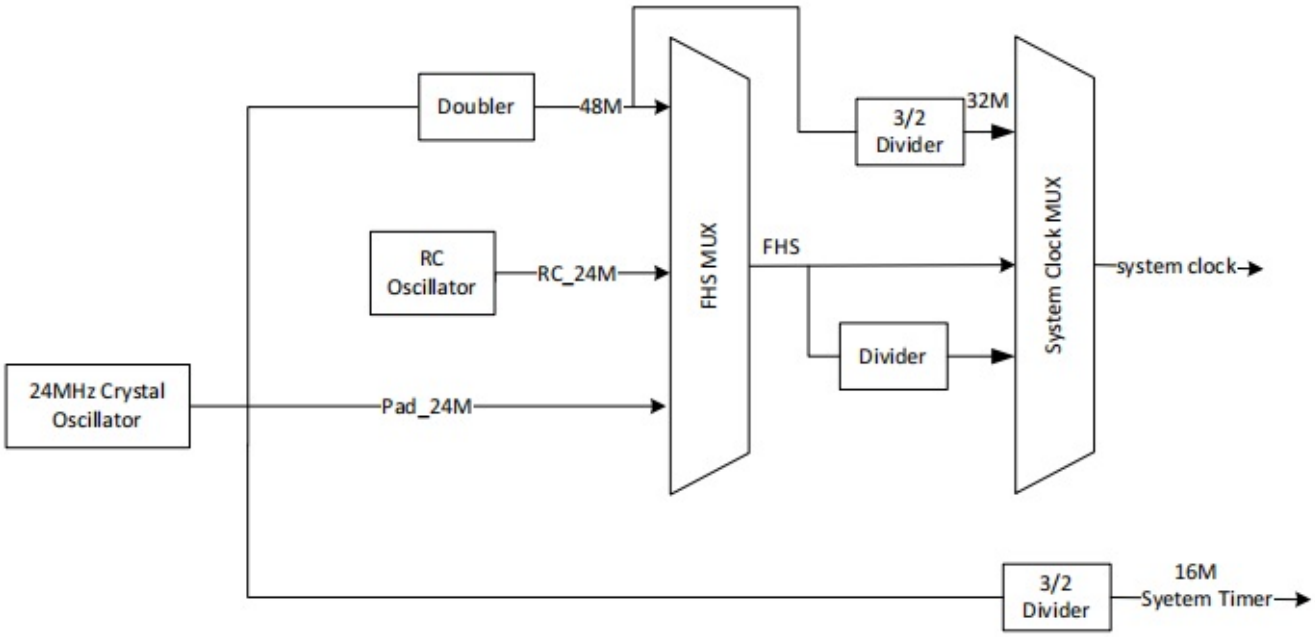
然后，把所有的功能都测试一遍，然后查看 normal stack 和 irq_stack 是否有溢出。
Normal stack:初始化值为 0xffffffff。demo SDK 需要 stack 2K 左右，因为目前使用的 stack 都小于 3K，所以为了加快 RAM 初始化速度，只初始化 3K 的 RAM。通过读取 RAM 查看，如果发现 61K 位置即 0x84F400--0x84F403 这 4 个 byte 不是 0xFFFFFFFF，则表示堆栈使用已经超过 3K，如果超过，请联系我们做进一步的分析确认。irq_stack:初始化值为 0x00000000，通过 tdebug 查看 irq_stk[]变量，观察使用情况；
如果发现 irq_stk[0--3] 这 4 个 byte 非 0，则表示已经溢出。

2.2 时钟模块

MCU 的 clock 由 CLOCK_SYS_CLOCK_HZ 定义，825x 默认是 16MHz.

2.2.1 System clock & System Timer

系统时钟（system clock）是 MCU 执行程序的时钟。
系统定时器（System Timer）是一个只读的定时器，为 BLE 的时序控制提供时间基准，同时也可以提供给用户使用
在 Telink 上一代 IC（826x 系列）上，System Timer 的时钟来源于 system clock，而 8x5x IC 上，System Timer 和 system clock 是独立分开的。如下图所示，System Timer 是由外部 24M Crystal Oscillator 经 3/2 分频得到的 16M。



图上可以看到，system clock 可以由外部 24M 晶体振荡器经"doubler”电路倍频到 48M 后再分频得到 16M、24M、32M、48M 等，这一类 clock 我们称为 crystal clock（如 16M crystal system clock、24M crystal system clock）；也可以由 IC 内部 24M RC Oscillitor 处理后得到 24M RC clock、32M RC clock、48M RC clock 等。这一类我们称为 RC clock（BLE SDK 不支持 RC clock）。
在 BLE SDK 中，我们推荐使用 crystal clock。

初始化时调用下面的 API 配置 system clock，在枚举变量 SYS_CLK_TYPEDEF 定义中选择时钟对应的 clock 即可。

void clock_init(SYS_CLK_TYPEDEF SYS_CLK)

由于 8x5x System Timer 与 system clock 不一样，用户需要了解 MCU 上各硬件模块的 clock是来源于 system clock 还是 System Timer。我们以 system clock 为 24M crystal 的情况来进行说明，此时 system clock 为 24M，而 System Timer 是 16M。

在文件 app_config.h 中 system clock 以及对应的 S、mS、uS 的定义如下:

```
#define CLOCK_SYS_CLOCK_HZ 24000000
enum{
CLOCK_SYS_CLOCK_1S = CLOCK_SYS_CLOCK_HZ,
CLOCK_SYS_CLOCK_1MS = (CLOCK_SYS_CLOCK_1S / 1000),
CLOCK_SYS_CLOCK_1US = (CLOCK_SYS_CLOCK_1S / 1000000),
};
```

所有时钟源为 system clock 的硬件模块，在设置模块的 clock 时，只能使用上面 CLOCK_SYS_CLOCK_HZ、CLOCK_SYS_CLOCK_1S 等；换言之，如果用户看到模块中 clock 的设置使用的是以上几个定义，说明该模块的时钟源为 system clock。

如 PWM 驱动中 PWM 周期和占空的设置如下，说明 PWM 的时钟源是 system clock。

```
pwm_set_cycle_and_duty(PWM0_ID, (u16) (1000 * CLOCK_SYS_CLOCK_1US), (u16) (500
*CLOCK_SYS_CLOCK_1US) );
```

System Timer 是固定的 16M，所以对于这个 timer，SDK code 中使用如下的数值来表示 S、mS 和 uS。

```
//system timer clock source is constant 16M, never change
enum{
CLOCK_16M_SYS_TIMER_CLK_1S = 16000000,
CLOCK_16M_SYS_TIMER_CLK_1MS = 16000,
CLOCK_16M_SYS_TIMER_CLK_1US = 16,
};
```

SDK 中以下几个 API 都是跟 System Timer 相关的一些操作，所以涉及到这些 API 操作时，都使用上面类似”CLOCK_16M_SYS_TIMER_CLK_xxx”的方式来表示时间。

```
void sleep_us (unsigned long microsec);
unsigned int clock_time(void);
int clock_time_exceed(unsigned int ref, unsigned int span_us);
#define ClockTime clock_time
#define WaitUs sleep_us
#define WaitMs(t) sleep_us((t)*1000)
```

由于 System Timer 是 BLE 计时的基准，SDK 中所有 BLE 时间相关的参数和变量，在涉及

到时间的表达时，都是用”CLOCK_16M_SYS_TIMER_CLK_xxx”的方式。

2.2.2 System Timer 的使用

Main 函数中 cpu_wakeup_init 初始化完成后后，System Timer 就开始工作，用户可以读取 System Timer 计数器的值（简称 System Timer tick）。

System Timer tick 每一个时钟周期加一，其长度为 32bit，即每 1/16 us 加 1，最小值 0x00000000，最大值 0xffffffff。System Timer 刚启动的时候，tick 值为 0，到最大值 0xffffffff 需要的时间为:(1/16) us * (2^32) 约等于 268 S，每过 268 S System Timer tick 转一圈。

MCU 在运行程序过程中 system tick 不会停止。

System Timer tick 的读取可以通过 clock_time()函数获得:

u32 current_tick = clock_time()

该 BLE SDK 整个 BLE 时序都是基于 System Timer tick 设计的，程序中也大量使用这个 System Timer tick 来完成各种计时和超时判断，强烈推荐 user 使用这个 System Timer tick 来实现一些简单的定时和超时判断。

比如要实现一个简单的软件定时。软件定时器的实现基于查询机制，由于是通过查询来实现，不能保证非常好的实时性和准确性，一般用于对误差要求不是特别苛刻的应用。实现方法:

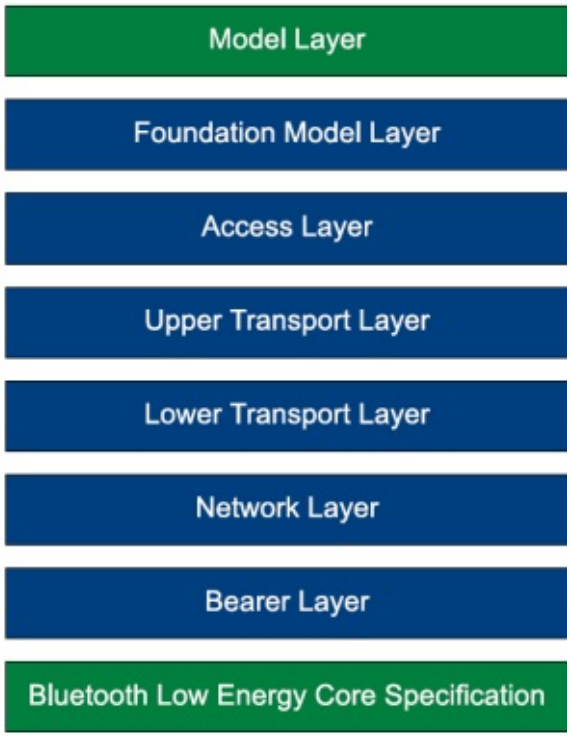
1. 启动计时:设置一个 u32 的变量，读取并记录当前 System Timer tick。
   u32 start_tick = clock_time(); // clock_time()返回 System Timer tick 值。
2. 在程序的某处不断查询当前 System Timer tick 和 start_tick 的差值是否超过需要定时的时间值。若超过，认为定时器触发，执行相应的操作，并根据实际的需求清除计时器或启动新一轮的定时。
   假设需要定时的时间为 100 ms，查询计时是否到达的写法为:
   if((u32) (clock_time() - start_tick) > 100 * CLOCK_16M_SYS_TIMER_CLK_1MS)
   由于将差值转化为 u32 型，解决了 system clock tick 从 0xffffffff 到 0 这个极限情况。
   实际上 SDK 中为了解决系统时钟不同导致和 u32 转换的问题，提供了统一的调用函数，不管系统时钟多少，都可以下面函数进行查询判断:
   if(clock_time_exceed(start_tick, 100 * 1000)) //第二个参数单位为 us
   需要注意的是:由于 16M 时钟转一圈为 268 S，这个查询函数只适用于 268 S 以内的定时。如果超过 268 S，需要在软件上加计数器累计实现（这里不介绍）。
   应用举例:A 条件触发（只会触发一次）的 2 S 后，程序进行 B()操作。

```
u32 a_trig_tick;
int a_trig_flg = 0;
while(1)
{
if(A)
{
```

```
a_trig_tick = clock_time();
a_trig_flg = 1;
}
if(a_trig_flg &&clock_time_exceed(a_trig_tick,2 *1000 * 1000))
{
a_trig_flg = 0;
B();
}
}
```

3. MESH spec 中的常用概念介绍

3.1 Layered architecture



3.1.1 Model layer

model 定义了一个节点支持的功能，每一个 model 都定义了自己的 op code，以及 status。

比如 generic onoff model，定义了 Generic ON/OFF/GET/STATUS。

Provisioner 在组网的时候，会通过 get composition data 命令去获取节点支持的所有 model id，然后 provisioner 就能知道节点具体支持什么功能了。只有这个节点支持了对应的 model 之后，才应该给它发送该 model 定义的 op code。

Model 又分为 server model 和 client model。server model:简单的说，他是一个被控制的角色，有自己的状态，可以被别的节点改变和获取，比如 onoff server model，可以接收 onoff set/ get 命令，可以回复 onoff status 命令，但是不能发送 onoff set/get 命令，也不会处理onoff status 命令。

client model:是一个控制 server 节点的角色，没有自己的状态。比如 onoff client model，可以发送 onoff set/ get 命令，可以处理收到的 onoff status 命令，但是不能发送 onoff status命令，也不会处理 onoff set/get 命令。

3.1.2 Foundation Model layer

Foundation Model 的模式和 model 基本一样，是基础 model，包含 Configuration Server model，Configuration Client model，Health Server model，Health Client model。
对于被配网节点都必须包含 Configuration Server model， provisioner 节点必须包含 Configuration Client model。这两个 model 包含的常用 op code 是 subscription add/delete(即组号添加/删除)等，并且这两个 model 的 access layer 层的加密都使用 device key，所以一般来说只有 provisioner 节点才能发送 configuration model 的 set/get 命令。

3.1.3 access layer

把 op code 和 parameter 按规定的格式组合在一起。

3.1.4 transport layer

使用 app key 或者 device key(configuration model 使用)进行加解密。判断并确认是否需要执行 分包和组包协议。目前为了兼容 BLE4.2 等不支持长广播包的设备，所以都统一设定 adv 的最大 payload 为31byte。

3.1.5 Network layer

对于发送流程:主要包含，对数据包添加 sequence number，等，并使用 network key，iv index 对数据进行加密。发送完成后 sequence number 会执行“加 1”的操作。
对于接收流程:主要包含，使用 network key，iv index 对数据进行解密，解密后判断 sequence number 是否有效(即是否大于已经接收过的值)，如果无效则直接丢弃。

3.1.6 Bearer layer

把已经执行过加密的数据包通过type为 LL_TYPE_ADV_NONCONN_IND(0x02) 的广播包发送到 mesh 网络中。

3.2 Architectural concepts

3.2.1 States

一个节点的状态，比如 onoff States，lightness States

3.2.2 Bound states

两个相互关联的 states，比如 onoff 和 lightness。比如，当 onoff 的值由 1 变为 0 的时候，

lightness 的值也会变成 0；当 onoff 的值由 0 变为 1 的时候，lightness 的值也会由 0 变为关灯之前的 lightness 的值.同理，当 lightness 的值在 0 和非 0 之间变化时，onoff 的值也会由 0 变为 1。

3.2.3 Messages

就是把加密完成后，发送到 mesh 网络中的 packet，我们也叫做 mesh packet、mesh command。

3.2.4 Node & Elements

Node 表示一个完整的节点或者蓝牙模块，Element 表示 Node 里面的某个操作元素。
Node address 有且只有一个，element address 可以是一个或者多个，当 element address 是多个的情况，这些 address 都会是连续的。第一个 element address 又叫做 primary address，Node address 的值和 primary address 相同。
有多个 element 的原因是，当一个 node 有多个相同的 states 的时候，就需要用到了。比如一个插座产品，插座上有 3 个插孔，要使用 Generic ONOFF 命令控制插孔的 onoff 状态，如果只有一个 address 的话，当节点收到命令后，是没办法确定要控制哪一个插孔的，所以为了能指定控制某一个插孔，就需要使用多个 element address。
另外，色温灯(CT Light)的 element 个数是 2 个，虽然色温灯只需要一个 onoff states，但是他需要两个 generic level model，一个是和 lightness 对应，另一个是和 Temp 对应，所以就需要 2 个 element。
同理，HSL 灯(RGB 灯)的 element 个数是 3，因为 HSL 灯需要 3 个 generic level model，分别和 lightness、Hue 以及 Sat 对应。
在组网的时候，Node 会在 provision flow 的交互信息里面上报 element 的个数，比如 2，provisioner 会分配一个地址给 Node，比如是 0x0002。Node 收到后，会按顺序分配 0x0002 给 element 1, 0x0003 给 element 2。Provisioner 在对下一个节点进行组网的时候，会从 0x0004 开始分配。

3.2.5 Models

参考 3.3.1 “Model layer”的介绍。

3.2.6 Publish & subscribe

Publish:publish 就是 Element 主动发送 status 的过程，可以通过 Config Model Publication Set 命令配置 publish address，以及设置周期 publish 参数。当配置了 publish address 后，只要状态发生变化，Node 都会自动执行 publish status 的动作。是否需要周期发送，就要看周期 publish 的参数。

Subscribe:Subscribe (即订阅)说的是:节点接收到别的节点 publish 出来的 status message(比如 generic onoff status)，或者 control message(比如 generic onoff)后，根

据 model 的 Subscribe list[]来判断是否处理该 message。Subscribe list[]里面是 group address 或者 virtual address，不能是 unicast address，也不能是 0xffff。可以通过 Config Model Subscription Add，Config Model Subscription Virtual Address Add 等命令添加。判断是否接收并处理的依据是:

1. 当收到 message 的 destination address 为非 unicast address，再判断是否能在对应 model 的 Subscribe list 的 address 里面匹配到。
2. 当 destination address 为 unicast address，则直接判断和自身的 element address 是否匹配。
3. 当 destination address 为 0xffff，则表示符合判断条件。

3.2.7 Security

加解密时需要使用的要素包含:Network key，IV index，App key 或者 device key。
一个 message 需要进行两层加密，分别是使用 app key 或者 device key 对整个 access layer(包含 op code，parameters)进行加密，以及使用 network key + iv index 对 network layer 进行加密，network layer 就是发送到 mesh 网络中的 packet，包含 source address、destination address 和 sequence number 等。
对 access layer 加密的时候，如果 op code 对应的 model 是 config model，则使用 device key，否则使用 app key。
对于需要分包的 message，因为 access layer 的 payload 在加密的时候是整个 payload 进行加密的，所以需要接收到所有的 packet 后，才能完成解密和校验。
我们的SDK 默认支持 2 个network key (NET_KEY_MAX)和两个 app key(APP_KEY_MAX)。
多个 network key 可用于管理多个 network。
多个 app key 可以用于管理不同安全级别的产品。比如 mesh 网络里面既有灯，也有门锁，如果需要门锁的安全级别更高的话，只有某些人能控制，此时可以通过单独给门锁分配一个独立的 app key，并且保证这个 app key 只有某些手机 app(provisioner)知道，在进行网络分享的时候，也不会分享出去。这样就可以提高安全级别。

3.2.8 存储 Sequence Number 的处理

前面”3.1.5 Network layer”章节有提到，mesh message 的 Sequence Number(SNO)每发完一次命令都会进行累加，接收端在进行 SNO 判断的时候，要求大于已经收到过的值，如果小于等于则认为是无效的 message。这个就要求发送端每发送一个命令后，都要把 SNO 存储到flash 中，这个存储频率太高了，特别是在需要分包发送的时候。所以我们做了一个处理:SNO每增加 MESH_CMD_SNO_SAVE_DELTA(默认 0x80)，才存储一次，此时，为了能保证在断电并重新上电后的 SNO 大于已经使用过的值，在上电初始化时，把读取到的 SNO 加上MESH_CMD_SNO_SAVE_DELTA。具体实现部分，请参考 mesh_flash_save_check()和 mesh_misc_retrieve()关于 MESH_CMD_SNO_SAVE_DELTA 的处理。

3.2.9 Friendship

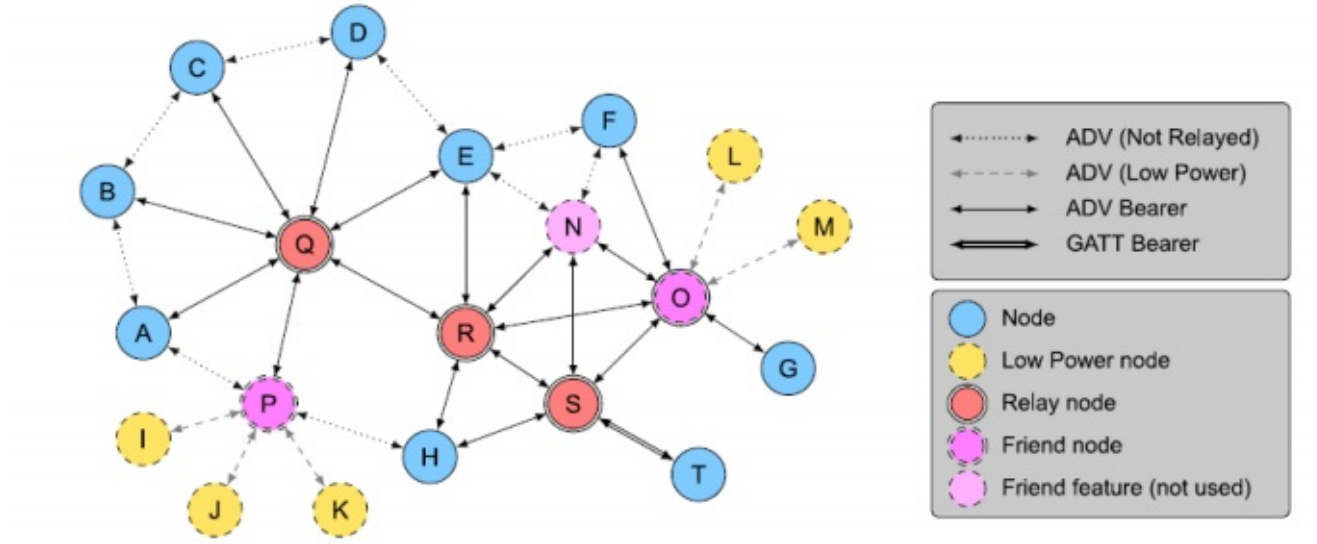
Friend ship 是指 Friend Node(FN)和 Low Power Node(LPN)之间通过规定的 establish friend ship flow 建立的关系。在 friend ship 建立成功后，LPN 在 sleep 期间，当有节点发命令给 LPN 时，FN 会先把这个 message 先保存下来。当 LPN 唤醒后，会发送 POLL 的查询命令给 FN，此时 FN 就会把保存的 message 发送给 LPN。这样就能达到较低的功耗，缺点是需要网络中有 friend node 的存在，以及命令的接收和响应有一定的延迟。
更详细的部分请参考 spec 以及后续 LPN 的章节。

3.2.10 Features

Mesh features 主要有:

Relay feature ---- 节点收到有效的 network message 后，会把 message 的 TTL 值减 1，然后 relay 发送出去，如果收到的 TTL 的值小于等于 1，则不进行 relay。这样可以使 mesh 网络传输距离更远。引入 TTL，主要是控制最后收到该 message 的节点的 delay 时间在可控范围内。我们 SDK 的TTL 的值默认是TTL_DEFAULT(0x0A)，可以修改这个宏，provisioner 也可以通过 Config Default TTL Set 配置，最大值是 127。
Proxy feature ---- proxy 是用于手机 APP 接入 mesh 网络的协议。在 mesh 网络中，APP 也是一个独立的节点，有自己的 Node address。引入 proxy，是因为目前大部分手机不能完全自定义发送广播包，以及不能一直处于监听 mesh 网络的状态(中间要切换到 WiFi 等)，所以手机 APP 需要通过 BLE GATT 连接一个节点，直连节点收到 APP 发过来的数据后会转发出去，当直连节点收到 mesh 网络中回复给 app 的 message 后，会先通过 GATT 按 proxy 协议回复给手机 APP。
APP 下发 message 给直连节点用的是 ATT_OP_WRITE_CMD(0x52)，直连节点回复 message 给 APP 用的是 notify，即 ATT_OP_HANDLE_VALUE_NOTI(0x1B)。
Low Power feature ---- 参考 3.2.9 的“Friendship”的介绍。
Friend feature ----参考 3.2.9 的“Friendship”的介绍。

3.2.11 Mesh Topology



我们 SDK，默认常供电节点都支持 Relay，Friend. 所有节点都支持通过 GATT proxy 和 ADV组网。

3.3 Mesh networking

3.3.1 Network layer

Address

Values	Address Type
0b0000000000000000	Unassigned Address
0b0xxxxxxxxxxxxxx (excluding 0b0000000000000000)	Unicast Address
0b10xxxxxxxxxxxxx	Virtual Address
0b11xxxxxxxxxxxxx	Group Address

Unassigned address:0 表示 Unassigned address

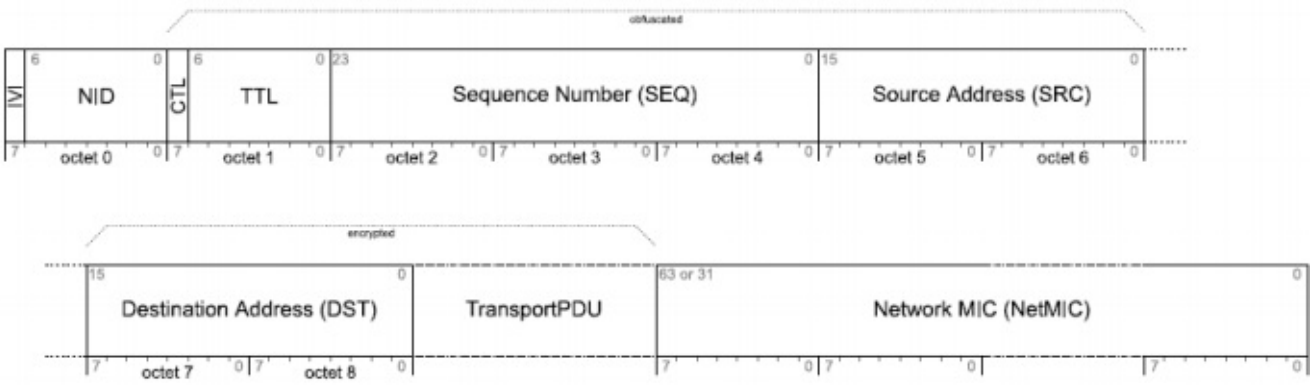
Unicast address:用于表示 element address

Group address:组号地址，用于组控以及 publish----subscribe 机制。

Virtual address:需要结合 16BYTE 的 lable UUID 使用，Virtual address 是这个 UUID 经过 hash 算法后生成的值。当 group address(共 16384 个)不够使用的时候，可以进行扩展。目前的应用暂时用不到。

Network PDU

图 3-3 Network PDU 格式



表格 3-2 Network PDU Field Definitions

Field Name	Bits	Notes
IVI	1	Least significant bit of IV Index
NID	7	Value derived from the NetKey used to identify the Encryption Key and Privacy Key used to secure this PDU
CTL	1	Network Control
TTL	7	Time To Live
SEQ	24	Sequence Number
SRC	16	Source Address
DST	16	Destination Address
TransportPDU	8 to 128	Transport Protocol Data Unit
NetMIC	32 or 64	Message Integrity Check for Network

IVI: iv index(即 SDK 的 iv_idx_st.tx[3]这个 byte 的最低 bit, 目前 SDK 的 iv index 按 big endian 存储)。

NID:和 network key 相关

CTL:标记是否是 control message。

Network transmit count/interval (重传次数和重传间隔的定义)

network transmit count 是指发送一个命令，需要重传的次数，这些重传的 rf packet 是一模一样的，包括 sno 等。重传的目的是为了提升接收成功率。假如当网络中只有两个 mesh 节点，只发送一次的成功率是 80%，即丢包率是 20%，那么，理论上，发送 6 次的丢包率是 20%的 6 次方，即 0.0064%，即收包成功率是 99.993%，当然这个是理论分析。和 RF 环境等还有一定的关系。

我们的 SDK 协议栈默认重发 5 次，即 TRANSMIT_CNT_DEF(5)，总共发发送次数是 n+1 = 6 个。

network transmit interval 是指重传的两个包之间的发送间隔，我们 SDK 默认在 30-40ms 之间，由 TRANSMIT_INVL_STEPS_DEF(2) 决定，计算方式是 ((TRANSMIT_INVL_STEPS_DEF + 1)*10 + (0----10))ms。”

network transmit count、transmit interval 还可以通过 SIG 定义的标准 config 命令

CFG_NW_TRANSMIT_SET 来配置。

综上所述，我们 SDK 默认发送一个 network packet，比如 generic ONOFF no ack 命令(该命令不需要分包)，需要的时间大概是 40 * 6 = 240ms。

Reliable retry(发包重试次数)Reliable retry 是指应用层的重试，用于有 status 回复的命令，比如 generic ONOFF。当发送一个 network packet 后(包含 network transmit)，会判断是否收到 status，如果没有收到，我们会进行 retry，retry 的时候，network packet 里面的 sequence number 会变化。我们默认最多重试两次。

3.3.2 Access layer

表格 3-3 Access Payload Field

Field Name	Size (octets)	Notes
Opcode	1, 2, or 3	Operation Code
Parameters	0 to 379	Application Parameters

表格 3-4 Opcode Format

Opcode Format	Notes
0xxxxxxx (excluding 01111111)	1-octet Opcodes
01111111	Reserved for Future Use
10xxxxxx xxxxxxxx	2-octet Opcodes
11xxxxxx zzzzzzzz	3-octet Opcodes

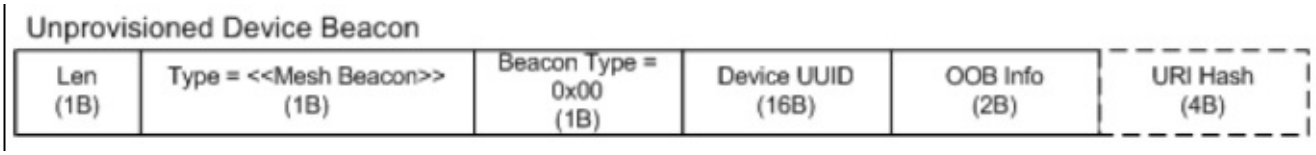
op code 有三种类型，1byte，2byte 和 3byte。1byte，2byte 是 SIG 定义的命令。3byte 是 vendor 自定义的命令，其中 2 个 byte 是 vendor ID（CID）,整个 mesh 网络中，一个 vendor id 最多支持 64 个 vendor opcode。Access layer 包含 op code 和 parameter，最大支持 380 byte。

3.3.3 transport layer

目前为了兼容 BLE4.2 等不支持长广播包的设备，所以都统一设定 adv 的最大 payload 为 31byte。去掉一些数据包的通讯协议需要占用的部分，单包的有效 payload 是 11byte，所以当 Access layer 超过 11byte 后，就需要分包，所以对于 vendor op code 来说，当 parameters 大于 8 个 byte(8=11-3)，mesh 协议栈就会自动执行分包发送。用户不需要介入。

3.3.4 mesh beacon

下图是 unprovisioned device beacon 的 PDU。



Device UUID 可以唯一识别 node。因为有些手机，比如 IOS 不能获取 mac，以及在未来的 remote provision 中无法获得 mac，所以在 SIG mesh 中是通过 Device UUID 来唯一识别 node，而不是通过 mac。unprovisioned beacon 通过 non-connectable ADV 的 packet 来发送，用于 PB-ADV provision 模式。

Oob info 及 URI Hash 请参考 spec 《3.9.2 Unprovisioned Device beacon》

在未组网时，会发送 unprovision beacon，通过 unprov_beacon_send()来发送。发送周期由”beacon_send.inter = MAX_BEACON_SEND_INTERVAL”来定义，默认是 2 秒。

在组网后，会发送 security beacon，通过 mesh_tx_sec_nw_beacon()来发送。另外还可以通过 CFG_BEACON_SET 命令打开或者关闭这个发送使能开关。请参考本文章节《4.4 控制对应的节点》“”SecNwBc”按钮”的操作。发送周期由 SEC_NW_BC_INV_DEF_100MS 来定义，默认是 10 秒。

3.3.5 iv update folw

即 iv index 的update flow。network layer和 access layer 的加解密过程都需要用到 iv index。

前面提到，mesh网络要求 network PDU的 sequence number 要一直累加，而 sequence number 是 3byte 表示。当使用很长一段时间后，sequence number 接近最大值的时候，就需要考虑更新 iv index，否则 sequence number 就会归 0，导致接收端认为是一个无效的 message。从某种程度上，可以理解 iv index 为 sequence number 的扩展位。

iv index update flow 是节点自行发起，自行 update 的过程。目前 SDK，当节点检测自己的 sequence number 超过了 IV_UPDATE_START_SNO (0xC00000) 后，会主动发起 iv update folw。每次 Iv update 后，iv index 会加 1.

详细的 iv update flow 请查阅 spe 相关章节: SPEC V1.0 《3.10.5 IV Update procedure》。

3.3.6 heartbeat

mesh 的心跳包，可以配置周期发送，所以可以用于做在线离线检测(周期 publish 机制也可以做在线离线检测)，以及 hops 的计算，即计算 heartbeat message 经过多少跳之后，才被接收到。经过统计一定时间内收到的 heartbeat 的个数(count)，并计算得到每个 heartbeat 的 hops 的 值，得到 min hops 和 max hops，进而了解整个网络的布局，以及每一个节点的 message 传输的可靠程度。不过每次配置 heartbeat subscription 只能监听和统计一个节点的状态。

hops 计算方式是 hops = InitTTL - RxTTL +1。

InitTTL:是 heartbeat publish set 里面的 TTL 参数。

RxTTL:是收到的 message 的 network PDU 里面的 TTL。

节点默认不发送 heartbeat，具体配置方式，详见“5.6 Heartbeat 的演示”章节

3.3.7 Health

Health model 相关的 message 是用来反映节点的 warning 或者 error 状态。比如电量的 warning 和 error 指示等.

天猫精灵控制

阿里天猫精灵平台

配置方式

```
//------------ mesh config (user can config)-------------
#define MESH_NORMAL_MODE        0
#define MESH_CLOUD_ENABLE       1
#define MESH_SPIRIT_ENABLE      2// use this mode should burn in the pa
#define MESH_AES_ENABLE         3
#define MESH_GN_ENABLE          4
#define MESH_MI_ENABLE          5
#define MESH_MI_SPIRIT_ENABLE   6    // dual vendor
#if __PROJECT_MESH_PRO__
#define MESH_USER_DEFINE_MODE   MESH_NORMAL_MODE // must normal
#elif __PROJECT_SPIRIT_LPN__
#define MESH_USER_DEFINE_MODE   MESH_SPIRIT_ENABLE // must sprit
#else
#define MESH_USER_DEFINE_MODE   MESH_SPIRIT_ENABLE
#endif
```

Provision 采用 MESH_STATIC_OOB 模式。VENDOR_ID 是 0x01A8

向阿里申请三元组

默认的三元组信息在是空的(con_sec_data[16]为空)，具体代码在 user_ali.c 如下图所示:

```
#if(AIS_ENABLE)
STATIC_ASSERT(sizeof(sha256_dev_uuid_str) <= 16);   // because sizeof dev uuid is 16

// sha256 init for three parameters
#if ALI_NEW_PROTO_EN
    u32 con_product_id = 9873;
    //u8 con_mac_address[6]={0xf8,0xa7,0x63,0x6e,0x46,0x49};// in the flash ,is big endiness
    u8 con_mac_address[6]={0x49,0x46,0x6e,0x63,0xa7,0xf8};// in the demo ,switch to small endiness .
    u8 con_sec_data[16]={0xc5,0xa2,0x78,0xcf, 0x21,0x4b,0x84,0x63, 0xc7,0xb4,0x70,0xe2, 0x83,0x7d,0x55,0xb5};
    #define SIZE_CON_SEC_DATA   (sizeof(con_sec_data))
    #else
#if(MESH_USER_DEFINE_MODE == MESH_CLOUD_ENABLE)
u32 con_product_id=192941;// little endiness
    #if(MESH_USER_DEFINE_MODE != MESH_MI_SPIRIT_ENABLE)
const
    #endif
u8 con_mac_address[6]={0xee,0x11,0x33,0x55,0x77,0x03};//small endiness
```

所以客户需要从阿里获取三元组:

(共 24byte: PID(4byte,小端) + MAC(6byte,大端) + secret data(16 byte))，

然后烧录到 FLASH_ADR_THREE_PARA_ADR(0x78000)，代码会自动读取 0x78000 的参数。

通过天猫精灵组网

直接通过天猫精灵的语音命令组网就可以了。

通过上位机组网

因为天猫精灵模式默认只支持 static oob 的模式， oob 和三元组是有绑定关系的，所以上位机需要知道三元组的信息才能组网。所以需要把三元组添加到上位机的这个文件:

SIG_MESH_Release_Vxxx\tools\telink-ble-phone\three_para.txt

SDK 默认的三元组信息已经添加在这个文件里面了。添加新的三元组信息的时候，参考这个格式添加即可。

然后直接按“4.调试工具操作说明”章节里面的“组网部分”进行组网就可以。

同时支持 static oob 和 no oob 模式

天猫精灵模式，节点端默认只响应 static oob 的组网模式，如果需要同时支持 no oob 模式，把 ENABLE_NO_OOB_IN_STATIC_OOB 由 0 改为 1，即可。

【PB系列模组】二次开发（PHY6212）

环境搭建

GPIO

PHY6212 GPIO 最多支持 35 个 IO,不同 IO 上电后默认属性不同，除了做 GPIO 模式，IO 还可以配置为功能模式，即复用为外设模块的驱动引脚，比如 I2C、UART 等。实际使用时一定要确保 IO 属性是配置正确的。表中 phyplus_spi* 是 phyplus 私有协议的 SPI 不是通用 spi。可以通过 IOMUX 功能配置成别的功能 IO。
35 个 IO 都可以作为数字端口或模拟端口。
35 个 IO 作为数字端口使用时，都可以配置端口方向为输入或输出。
每个 IO 都可以配置上下拉电阻，用来设置引脚的默认状态。

浮空:未知态。
弱上拉:上拉到 AVDD33，高电平，驱动电流小。上拉电阻 150K 欧姆
强上拉:上拉到 AVDD33，高电平，驱动电流大。上拉电阻 10K 欧姆
下拉:下拉到地，低电平，下拉电阻 100K 欧姆。
P00~P17，这 18 个 GPIO 支持中断和唤醒。
P18~P34，这 17 个 GPIO 支持唤醒，不支持中断。
提供 GPIO 相关的操作，包括 GPIO 配置，上下拉设置，GPIO 输入模式的事件驱动模型，GPIO 唤醒操作等。
其中 GPIO_DUMMY 定义为虚拟 pin，一般作为无效 pin 使用。

```
typedef enum{
  GPIO_P00   =   0,    P0  =  0,
  GPIO_P01   =   1,    P1  =  1,
  GPIO_P02   =   2,    P2  =  2,
  GPIO_P03   =   3,    P3  =  3,
  GPIO_P04   =   4,    P4  =  4,
  GPIO_P05   =   5,    P5  =  5,
  GPIO_P06   =   6,    P6  =  6,
  GPIO_P07   =   7,    P7  =  7,
  TEST_MODE  =   8,    P8  =  8,
  GPIO_P09   =   9,    P9  =  9,
  GPIO_P10   =   10,   P10  =  10,
  GPIO_P11   =   11,   P11  =  11,   Analog_IO_0 = 11,
  GPIO_P12   =   12,   P12  =  12,   Analog_IO_1 = 12,
  GPIO_P13   =   13,   P13  =  13,   Analog_IO_2 = 13,
  GPIO_P14   =   14,   P14  =  14,   Analog_IO_3 = 14,
  GPIO_P15   =   15,   P15  =  15,   Analog_IO_4 = 15,
  GPIO_P16   =   16,   P16  =  16,   XTALI = 16,
  GPIO_P17   =   17,   P17  =  17,   XTALO = 17,
  GPIO_P18   =   18,   P18  =  18,   Analog_IO_7 = 18,
  GPIO_P19   =   19,   P19  =  19,   Analog_IO_8 = 19,
  GPIO_P20   =   20,   P20  =  20,   Analog_IO_9 = 20,
  GPIO_P21   =   21,   P21  =  21,
  GPIO_P22   =   22,   P22  =  22,
  GPIO_P23   =   23,   P23  =  23,
  GPIO_P24   =   24,   P24  =  24,
  GPIO_P25   =   25,   P25  =  25,
  GPIO_P26   =   26,   P26  =  26,
  GPIO_P27   =   27,   P27  =  27,
  GPIO_P28   =   28,   P28  =  28,
  GPIO_P29   =   29,   P29  =  29,
  GPIO_P30   =   30,   P30  =  30,
  GPIO_P31   =   31,   P31  =  31,
  GPIO_P32   =   32,   P32  =  32,
  GPIO_P33   =   33,   P33  =  33,
  GPIO_P34   =   34,   P34  =  34,
  GPIO_DUMMY  =  0xff,
}GPIO_Pin_e;
```

GPIO 输入或者输出配置

GPIO_ioe

IE	配置为输入
OEN	配置为输出

IO pin 配置为 GPIO 模式或者配置为功能 pin

BitAction_e

Bit_DISABLE	配置为GPIO模式
Bit_ENABLE	配置为功能PIN

配置 pin 的上下拉模式

IO_Pull_Type_e

FLOATING	配置为无上下拉，pin悬空
WEAK_PULL_UP	配置为弱上拉

STRONG_PULL_UP	配置为强上拉
PULL_DOWN	配置为下拉

配置 pin 的唤醒极性，上升沿唤醒或者下降沿唤醒

IO_Wakeup_Pol_e

POSEDGE	配置为上升沿唤醒
NEGEDGE	配置为下降沿唤醒

APIs

int hal_gpio_init(void)

GPIO 模块初始化：初始化硬件，使能中断，配置中断优先级等等。该函数需要在系统初始化时候设置，一般是在 hal_init()函数中调用，具体请参考例程。

参数

无。

返回

PPlus_SUCCESS	初始化成功
其他数值	参考<error.h>

void hal_gpio_wakeup_set (GPIO_Pin_e pin,IO_Wakeup_Pol_e type)

配置 GPIO 唤醒模式：上升沿唤醒或者下降沿唤醒。

参数

类型	参数名	说明
GPIO_Pin_e	pin	GPIO pin
IO_Wakeup_Pol_e	type	唤醒模式

返回值

无。

void hal_gpio_pin_init(GPIO_Pin_e pin,GPIO_ioe type)

配置 GPIO 模式，输入或者输出。

参数

类型	参数名	说明
GPIO_Pin_e	pin	GPIO pin
GPIO_ioe	type	配置GPIO模式输入或者输出

返回值

无。

void hal_gpio_write(GPIO_Pin_e pin, uint8_t en)

向某一个 GPIO 写 1 或者 0。

参数

类型	参数名	说明
GPIO_Pin_e	Pin	GPIO pin
uint8_t	en	0:写0，其他值:写1

返回值

无。

uint32_t hal_gpio_read(GPIO_Pin_e pin)

读取某一个 GPIO 的值。

参数

类型	参数名	说明
GPIO_Pin_e	Pin	GPIO pin

返回值

0：低电平， 1：高电平

int hal_gpio_cfg_analog_io(GPIO_Pin_e pin, BitAction_e value)

配置 IO 为 analog 模式。

参数

类型	参数名	说明
GPIO_Pin_e	Pin	GPIO pin
BitAction_e	value	使能或者关闭IO的analog模式

返回值

PPlus_SUCCESS	成功
其他数值	参考<error.h>

int hal_gpio_pull_set(GPIO_Pin_e pin,IO_Pull_Type_e type);

设置 IO 的上下拉。

参数

类型	参数名	说明
GPIO_Pin_e	Pin	GPIO pin
Pull_Type_e	type	IO上下拉设置

返回值

PPlus_SUCCESS	成功
其他数值	参考<error.h>

int hal_gpio_fmux_set(GPIO_Pin_e pin,Fmux_Type_e type)

配置 IO 的功能模式。

参数

类型	参数名	说明
GPIO_Pin_e	Pin	GPIO pin
Fmux_Type_e	type	IO的功能模式

返回值

PPlus_SUCCESS	成功
其他数值	参考<error.h>

int hal_gpioin_register(GPIO_Pin_e pin, gpioin_Hdl_t posedgeHdl, gpioin_Hdl_t negedgeHdl)

注册 GPIO 的输入模式，该模式下支持中断和唤醒回调，包括上升沿回调和下降沿回调函数。

参数

类型	参数名	说明
GPIO_Pin_e	Pin	GPIO pin
gpioin_Hdl_t	posedgeHdl	上升沿的回调函数，可以为NULL
gpioin_Hdl_t	negedgeHdl	下降沿的回调函数，可以为NULL

返回值

PPlus_SUCCESS	成功
其他数值	参考<error.h>

int hal_gpioin_unregister(GPIO_Pin_e pin)

注销 GPIO 的输入模式，该模式下支持中断和唤醒回调，包括上升沿回调和下降沿回调函数，注销之后这些功能无效。

参数

类型	参数名	说明
GPIO_Pin_e	Pin	GPIO pin

返回值

PPlus_SUCCESS	成功
其他数值	参考<error.h>

int hal_gpioin_enable(GPIO_Pin_e pin)

对于已经注册为 gpioin 的 pin，如果该 pin 已经停用（参考 hal_gpioin_disable），通过该函数可以启用 gpioin 功能。

参数

类型	参数名	说明
GPIO_Pin_e	Pin	GPIO pin

返回值

PPlus_SUCCESS	成功
其他数值	参考<error.h>

int hal_gpioin_disable(GPIO_Pin_e pin)

对于已经注册为 gpioin 的 pin，如果该 pin 已经使能，通过该函数可以停止 gpioin 功能。

参数

类型	参数名	说明
GPIO_Pin_e	Pin	GPIO pin

返回值

PPlus_SUCCESS	成功
其他数值	参考<error.h>

PWM

中断

蓝牙Mesh基础入门

天猫精灵控制

【PB系列模组】二次开发（PHY6252）

环境搭建

蓝牙Mesh基础入门

简介

本文档用于 PHY6252 Mesh 的介绍以及使用方法，它有助于您了解和理解本公司 Mesh提供的组件，样例的使用方法，并且帮助您如何从提供的样例开始进行 BLE Mesh 的开发。

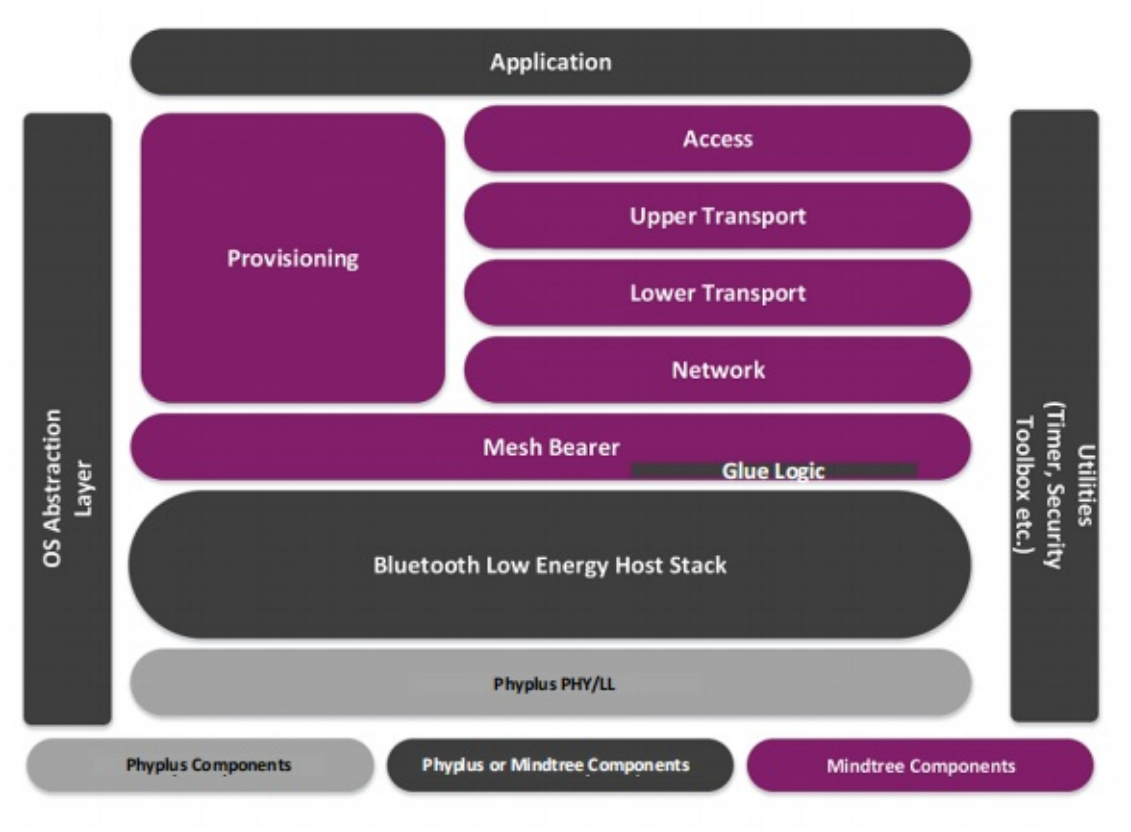


图 1：整体协议栈框架

1mesh 协议栈

这一协议栈建立在低功耗蓝牙技术之上。下图描绘了协议栈的层级。

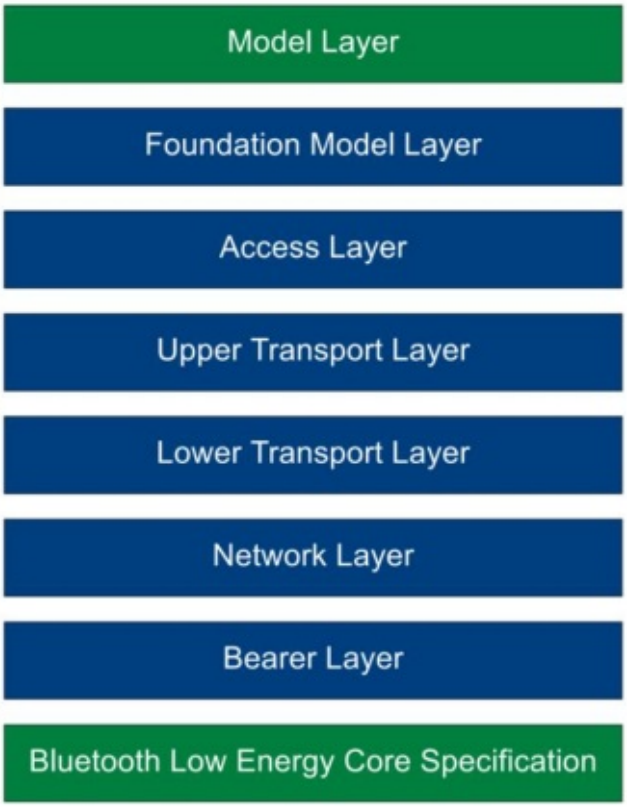


图 2：Mesh 协议栈框架

1.1 协议栈层

模型层（Model Layer）：模型层与模型等的实施、以及诸如行为、消息、状态等的实施有关。

基础模型层（Foundation Model Layer）：基础模型层负责实现与 mesh 网络配置和管理相关的模型。

访问层（Access Layer）：负责应用数据的格式、定义并控制上层传输层中执行的加密和解密过程，并在将数据转发到协议栈之前，验证接收到的数据是否适用于正确的网络和应用

上层传输层（Upper Transport Layer）：负责对接入层进出的应用数据进行加密、解密和认证。它还负责称为“传输控制消息”（transport control messages）这一特殊的消息，包括与“friendship”相关的心跳和消息。

底层传输层（Lower Transport Layer）：在需要之时，底层传输层能够处理 PDU 的分段和重组。

网络层（Network Layer）：网络层定义了各种消息地址类型和网络消息格式。中继和代理行为通过网络层实施。

承载层（Bearer Layer）：承载层定义了如何使用底层低功耗堆栈传输 PDU。目前定义了两个承载层：广播承载层（Advertising Bearer）和 GATT 承载层。

1.2 消息处理流程

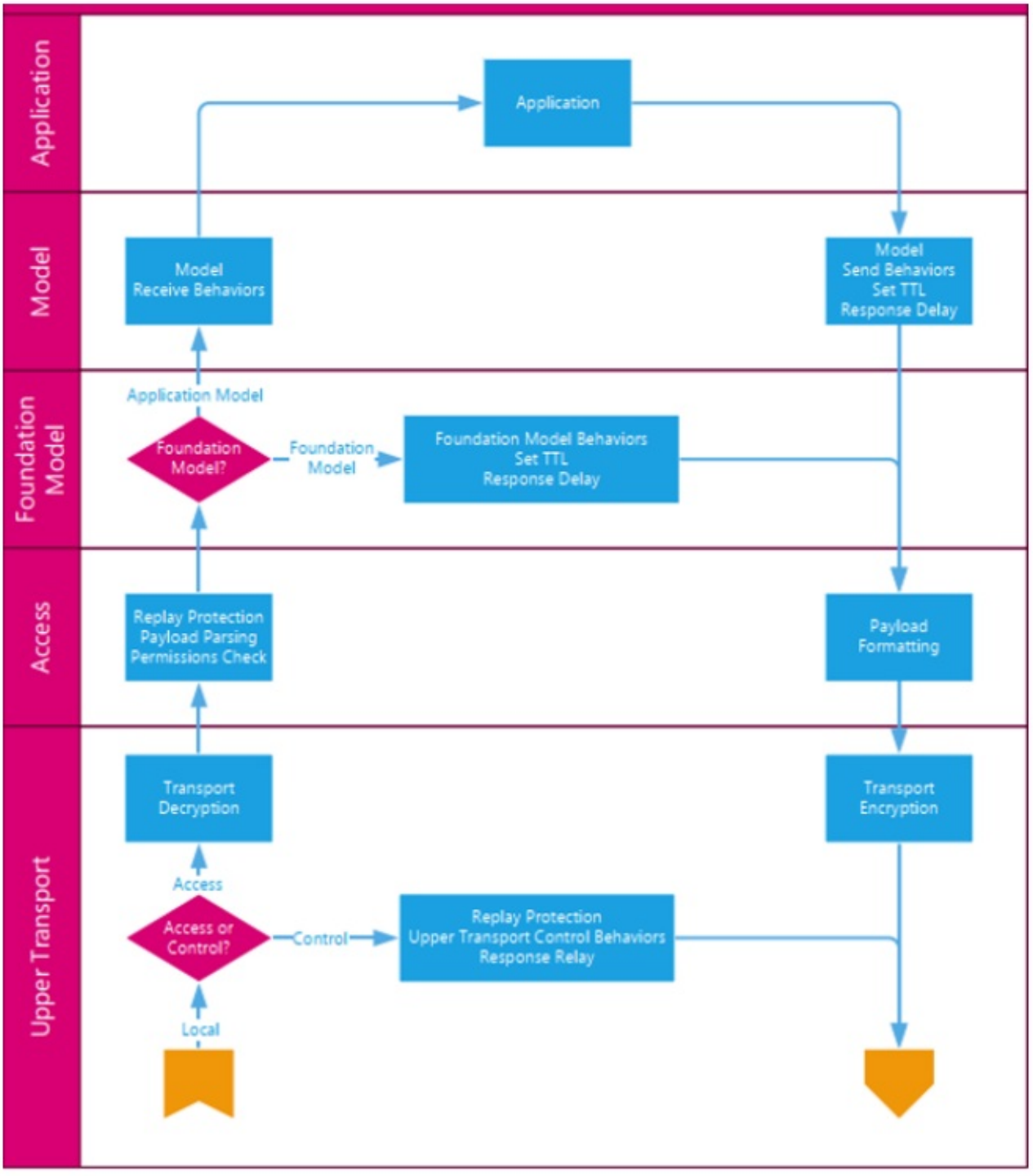


图 3：Message Flow Diagram – Upper Layers

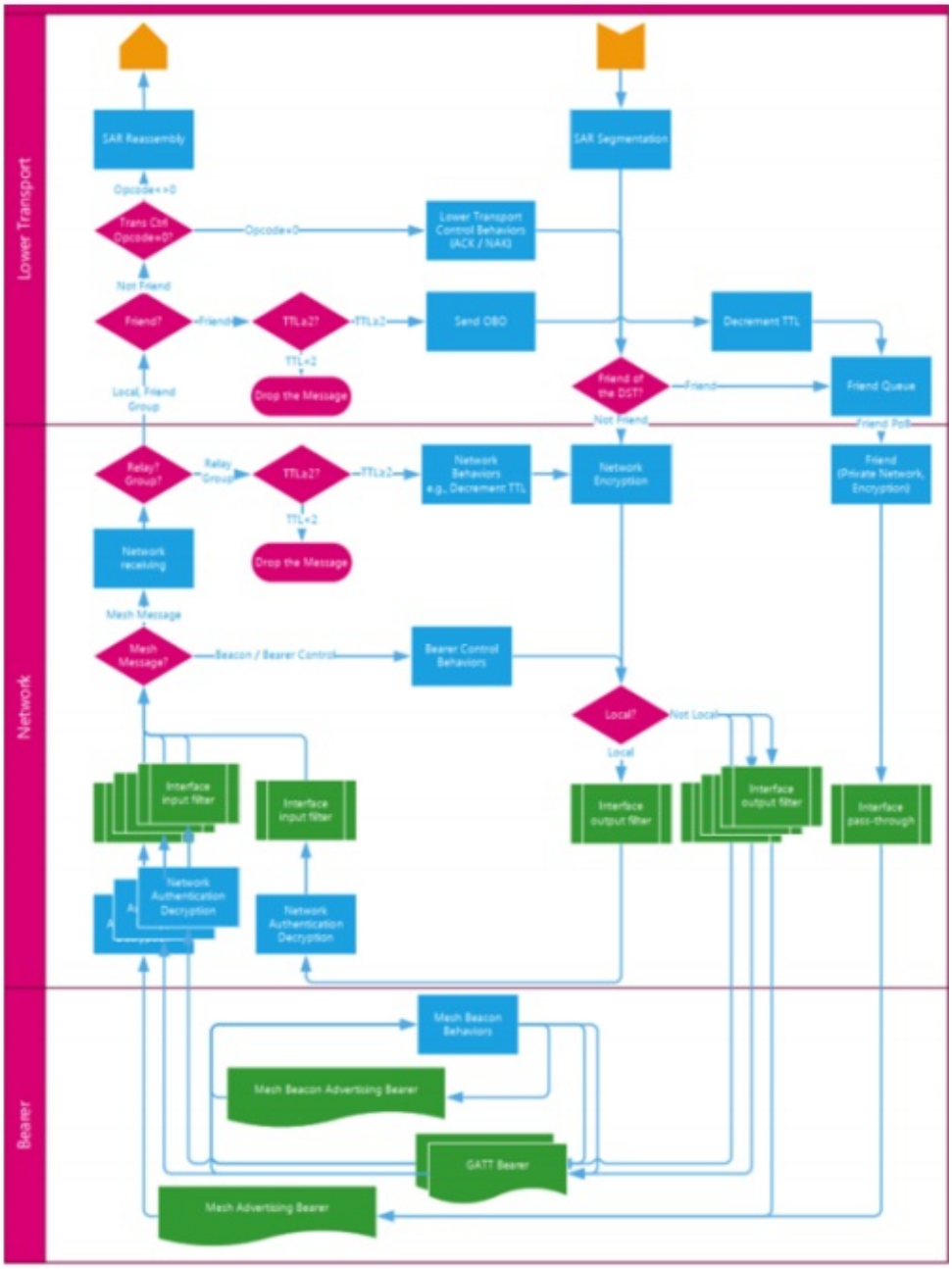


图 4：Message Flow Diagram – Lower Layers

Mesh 消息传输流程如上两图所示，mesh 消息从 bear 层通过 ADV/GATT 进入之后，在网络层解码通过接口输入过滤器之后，如果是中继或者代理消息则在网络层实现，非中继消息会进入底层传输层进行拆分和重组之后进入上层传输层进行传输解密，在访问层经过合法性检查发送给基础模型层，最后通过模型层的实施来实现。

发送消息时则通过模型层实例发送，经过访问层定义数据格式，在上层传输层加密，传入底层传输层进行拆包与组包，在网络层进行加密，通过接口输出过滤器进入承载层之后输出。

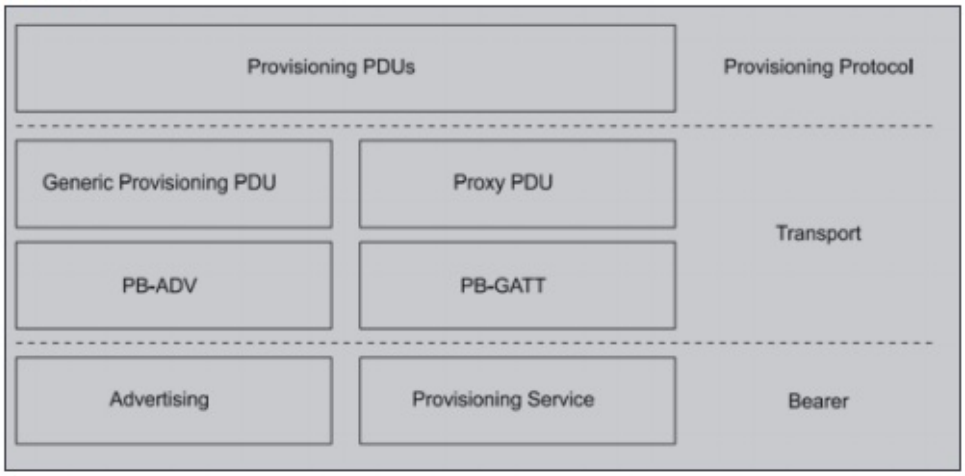


图 5：配置协议框架

设备的配置是使用发送配置 pdu 的配置协议完成的。配置 pdu 通过通用配置层传输到未配置的设备。这一层定义了如何将配置 pdu 作为可以分割和重新组装的事务进行处理。

这些处理是通过一个配置承载层发送的。配置承载层定义如何建立会话，以便将来自通用配置层的事务交付到单个设备。最后，配置体系结构的底部是承载层。

2 工程及 api 介绍

2.1 工程介绍

图 6：工程目录内容

2.2 公共模块定义

一些公共的模块定义与其他 sdk demo 类似，以下是 mesh 方面的定义。

名称	值	说明
OSAL_CBTIMER_NUM_TASKS	1	使用内部 call back timer 的个数，目前只支持 1 个，且不能更改，mesh 里面配置使用的 timer 都是调用内部 callback timer
CFG_HEARTBEAT_MODE	0	关闭 heartbeat 功能
CFG_HEARTBEAT_MODE	1	打开 heartbeat 功能

2.2.1 Mesh Sample 介绍

PHY62XX Mesh 内部接口都在 lib 库中，常用的 lib 为 libethermind_ecdh.lib、libethermind_mesh_core.lib、libethermind_mesh_models.lib 和 libethermind_utils.lib；功能如下：

1. libethermind_ecdh.lib：跟 ecdh 相关，目前 sdk 没有使用
2. libethermind_mesh_core.lib：mesh 协议栈相关；provision、config 和 message 处理都在这里面进行
3. libethermind_mesh_models.lib：mesh model 相关；目前使用的 on/off 等 model 的实现都在这里面处理
4. libethermind_utils.lib：mesh 存储相关

除了 lib 之外，用户接触和更改的文件一般在 appl_sample_mesh_XXX.c，本章节重点介绍 sample 相关的接口和定义（以常用的 mesh_light 为例）。

2.2.2 定义 model

USE_HEALTH	#undef：关闭 health model #define：使能 health model
USE_HSL	#undef：关闭 Light HSL model #define：使能 Light HSL model
USE_LIGHTNESS	#undef：关闭 Light Lightness model #define：使能 Light Lightness model
USE_CTL	#undef：关闭 Light CTL model #define：使能 Light CTL model
USE_SCENE	#undef：关闭 Light Scene model #define：使能 Light Scene model
USE_VENDORMODEL	#undef：关闭 Light vendormodel model #define：使能 Light vendormodel model

Vendormodel 如果使能的话，会自动使能 easy bonding（目前我们使用的 sdk 是这样考虑的，配合 Phy mesh app 使用）。

2.3 api 介绍

2.3.1 UI_health_server_cb

Health server 的回调函数

类型	参数名	说明
MS_ACCESS_MODEL_HANDLE*	handle	model handle
UINT8	event_type	event 的类型
UINT8*	event_param	event的参数内容
UINT16	param_len	参数长度

返回值

API_SUCCESS	成功
API_FAILER	失败

低功耗管理

1 概述

PHY6252低功耗相关的功能在 PWR_MGR 模块实现，对应的 API 代码存放在 SDK 的 components\driver\pwrmgr 目录下的 pwrmgr.c 和 pwrmgr.h 中。

PHY6252 有四类功耗模式：

- 普通模式：CPU 和外设全速运行，不休眠
- CPU 休眠模式：只有 CPU 会进入休眠，可由中断或事件唤醒。该模式由 OS 自行控制，应用程序无需干预
- 深度休眠模式：CPU 和大部分外设都会进入休眠。应用程序应根据需要设置休眠唤醒源（GPIO pin 和触发方式）和内存保持(memory retention,以保持运行时上下文)
- standby 模式：除了 AON 和一块 RAM 区域内存保持外，CPU 和其它外设都进入休眠。仅维持 RAM0 区域内容。应用程序应根据需要设置唤醒源（GPIO pin 和触发方式）
- 关机模式：除了 AON 外，CPU 和其它外设都进入休眠。唤醒后相当于系统重启，不能维持运行时上下文。应用程序应根据需要设置唤醒源（GPIO pin 和触发方式）

1.1 PWR_MGR 原理框图

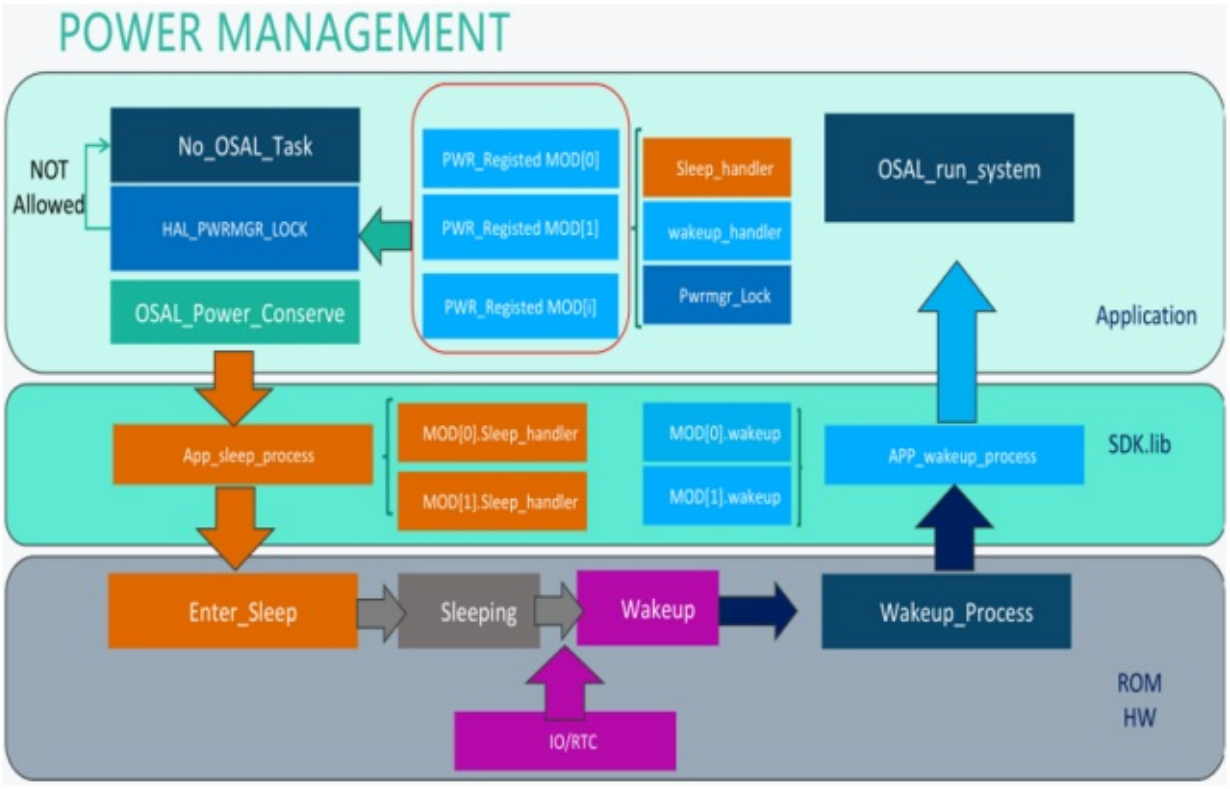
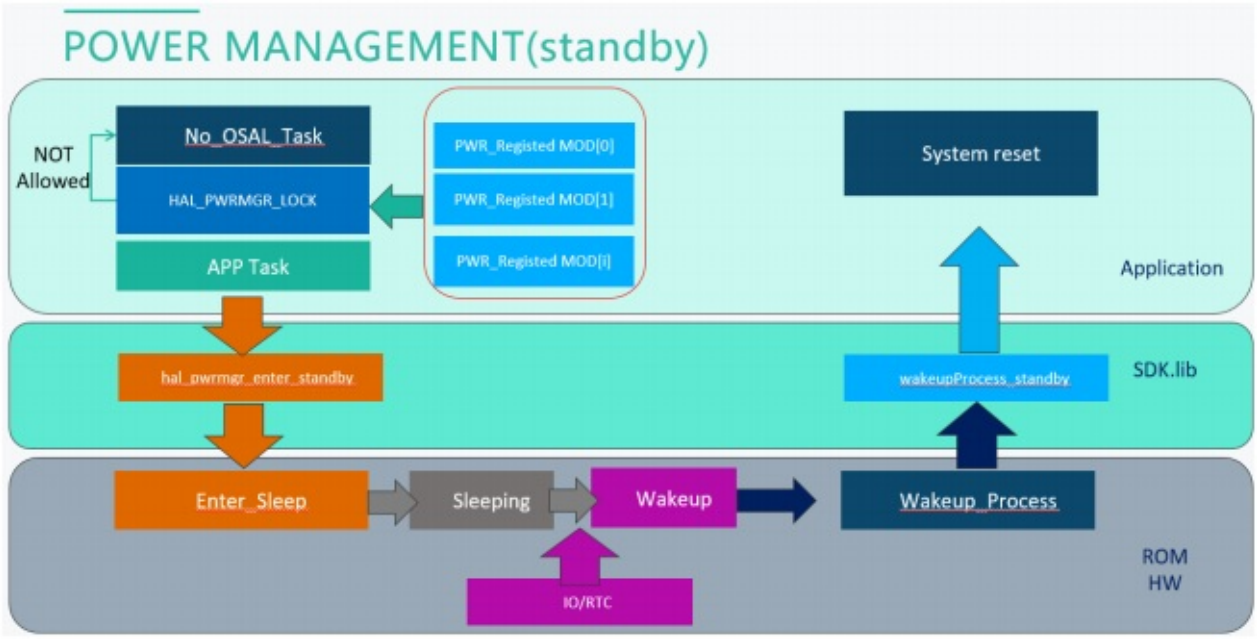


Figure 1：深度休眠模式原理框图

在 CPU/深度休眠模式下，当系统处于 IDLE 状态时，调用 sleep_process()尝试进入休眠模

式。若符合 CPU 休眠条件，则调用__WFI()等待中断唤醒后返回；否则的话，系统将进入深度休眠，在此之前，应用程序通过 hal_pwrmgr_register() API 注册的 sleep_handler()函数都会被回调，然后系统进入休眠；而当有 IO/RTC 触发唤醒时，系统会被唤醒并做相应的初始化。在系统唤醒过程中，应用程序通过 hal_pwrmgr_register() API 注册的所有 wakeup_handler()函数也都会被回调，然后由 OSAL 调度 task。



在 standby 模式下，APP Task 调用 hal_pwrmgr_enter_standby()进入 standby 模式；而当有 IO 触发唤醒时，系统会被唤醒并调用 wakeupProcess_standby(), 若系统满足唤醒的条件，则触发系统 reset。

2 PWR_MGR 模块 API

2.1 数据结构和类型

2.1.1 MODULE_e 类型

在 mcu_phy_bumbee.h 文件中定义了下列 module ID.

typedef enum {

MOD_NONE = 0, MOD_CK802_CPU = 0,

MOD_DMA = 3,

MOD_AES = 4,

MOD_IOMUX = 7,

MOD_UART0 = 8,

MOD_I2C0 = 9,

MOD_I2C1 = 10,

MOD_SPI0 = 11,

```
MOD_SPI1 = 12,
MOD_GPIO = 13,
MOD_QDEC = 15,
MOD_ADCC = 17,
MOD_PWM = 18,
MOD_SPIF = 19,
MOD_VOC = 20,
MOD_TIMER5 = 21,
MOD_TIMER6 = 22,
MOD_UART1 = 25,
MOD_CP_CPU = 0 + 32,
MOD_BB = MOD_CP_CPU + 3,
MOD_TIMER = MOD_CP_CPU + 4,
MOD_WDT = MOD_CP_CPU + 5,
MOD_COM = MOD_CP_CPU + 6,
MOD_KSCAN = MOD_CP_CPU + 7,
MOD_BBREG = MOD_CP_CPU + 9,
BBLL_RST = MOD_CP_CPU + 10, //can reset,but not gate in here
BBTX_RST = MOD_CP_CPU + 11, //can reset,but not gate in here
BBRX_RST = MOD_CP_CPU + 12, //can reset,but not gate in here
BBMIX_RST = MOD_CP_CPU + 13, //can reset,but not gate in here
MOD_TIMER1 = MOD_CP_CPU + 21,
MOD_TIMER2 = MOD_CP_CPU + 22,
MOD_TIMER3 = MOD_CP_CPU + 23,
MOD_TIMER4 = MOD_CP_CPU + 24,
MOD_PCLK_CACHE = 0 + 64,
MOD_HCLK_CACHE = MOD_PCLK_CACHE + 1,
MOD_USR0 = 0 + 96,
MOD_USR1 = MOD_USR0 + 1,
MOD_USR2 = MOD_USR0 + 2,
MOD_USR3 = MOD_USR0 + 3,
MOD_USR4 = MOD_USR0 + 4,
MOD_USR5 = MOD_USR0 + 5,
MOD_USR6 = MOD_USR0 + 6,
MOD_USR7 = MOD_USR0 + 7,
```

MOD_USR8 = MOD_USR0 + 8,
} MODULE_e;

2.1.2 pwrmgr_Ctx_t

PWR_MGR 模块为每个注册的模块(与 MODULE_e 对应)维护一个该结构类型的变量。最多 10 个。

```
typedef struct _pwrmgr_Context_t {
MODULE_e moudle_id;
bool lock; // 为 TRUE 时表示禁止休眠；反之，允许休眠
pwrmgr_Hdl_t sleep_handler; // 该模块对应的进入休眠之前会被调用的回调函数
pwrmgr_Hdl_t wakeup_handler; // 该模块对应的唤醒之前会被调用的回调函数
} pwrmgr_Ctx_t;
```

2.1.3 pwroff_cfg_t

在系统调用 hal_pwrmgr_poweroff() API 进入关机模式之前，需要设置的唤醒源(GPIO pin)和触发方式保存在该类型的变量中。

```
typedef struct {
gpio_pin_e pin;
gpio_polarity_e type; // POL_FALLING or POL_RISING
} pwroff_cfg_t;
```

2.2 APIs

PWR_MGR 模块的接口函数如下：

1. int hal_pwrmgr_init(void);
   模块初始化函数
2. bool hal_pwrmgr_is_lock(MODULE_e mod);
   查询模块 mod 的 lock 状态。TRUE:禁止休眠；FALSE:使能休眠
3. int hal_pwrmgr_lock(MODULE_e mod);
   设置模块 mod 的 lock 为 TRUE,并禁止休眠
4. int hal_pwrmgr_unlock(MODULE_e mod);
   设置模块 mod 的 lock 为 FALSE,并使能休眠
5. int hal_pwrmgr_register(MODULE_e mod, pwrmgr_Hdl_t sleepHandle, pwrmgr_Hdl_t wakeupHandle);
   注册模块 mod,并提供相应的休眠/唤醒回调函数
6. int hal_pwrmgr_unregister(MODULE_e mod);

取消注册模块 mod

7. int hal_pwrmgr_wakeup_process(void) attribute((weak));
8. int hal_pwrmgr_sleep_process(void) attribute((weak));

休眠/唤醒过程中 PWR_MGR 模块定义的处理函数，应用程序不需要也不应该调用它们

9. int hal_pwrmgr_RAM_retention(uint32_t sram);

配置需要保持的 RAM 区域，可选的有 RET_SRAM0 | RET_SRAM1 | RET_SRAM2

10. int hal_pwrmgr_clk_gate_config(MODULE_e module);

配置在唤醒时需要使能的时钟源。

11. int hal_pwrmgr_RAM_retention_clr(void);
12. int hal_pwrmgr_RAM_retention_set(void);

使能/清除对配置的 RAM 区域的保持(retention)功能

13. int hal_pwrmgr_LowCurrentLdo_enable(void);
14. int hal_pwrmgr_LowCurrentLdo_disable(void);

使能/禁用调节 LowCurrentLDO 的输出电压。

15. int hal_pwrmgr_poweroff(pwroff_cfg_t *pcfg, uint8_t wakeup_pin_num);

配置唤醒源后系统进入关机模式

16. void wakeupProcess_standby(void);

系统在 standby 模式下的唤醒函数。应用程序不需要也不应该调用它。

17. void hal_pwrmgr_enter_standby(pwroff_cfg_t * pcfg,uint8_t wakeup_pin_num);

让系统进入 standby 模式的 API 函数。应用程序需要在合适的时机调用它进入 standby

3 低功耗模式使用注意事项

为使用低功耗模式，编程时需要注意以下几个方面：

- 配置 CFG_SLEEP_MODE 宏：

在工程中需要设置 CFG_SLEEP_MODE=PWR_MODE_SLEEP 以使能休眠模式，而在其它模式下系统将不会进入休眠

- 初始化 pwrmgr 模块：

若要使用低功耗模式，在系统初始化时须调用 hal_pwrmgr_init()以初始化 pwrmgr 模块

- 配置不同 RAM 的 retention 属性：

若要使用低功耗模式，在系统初始化时需要调用 hal_pwrmgr_RAM_retention()以在系统休眠后保留对应 memory 区域的内容。可选的 RAM 区域有 RET_SRAM0，RET_SRAM1，和 RET_SRAM2，用户根据需要自行指定要保留的 RAM 区域

- 选择模块 ID 并注册休眠或唤醒回调函数：

PHY62 系列 SDK 在 mcu_phy_bumbee.h 文件中定义了一些 MODULE_e 类型的模块名/ID，在 pwrmgr 模块中作为模块 ID 使用。APP task 可以在 MOD_USR1 和 MOD_USR8 之间任选一个

作为模块 ID 使用。

若要使用低功耗模式，APP task 在初始化时需要调用以下函数进行注册：

hal_pwrmgr_register(MODULE_e mod, pwrmgr_Hdl_t sleepHandle,

pwrmgr_Hdl_t wakeupHandle);

其中 mod 就是模块 ID，是必须项，在 MOD_USR1 和 MOD_USR8 之间任选一个即可；

sleepHandle()和 wakeupHandle ()分别对应的是可选的休眠和唤醒的回调函数。

一个常用的做法是：用户可以在 sleepHandle()函数中做一些休眠唤醒使用的 pin 和相应属性的设置；而在 wakeupHandle()函数中做唤醒源的判定和初始化，以便系统唤醒后能恢复回到休眠前的状态

- 控制是否允许 APP task 进入休眠：

在使用低功耗模式时，APPtask 可以调用 pwrmgr 模块提供的下列接口来查询或控制是否允许进入休眠：

hal_pwrmgr_is_lock(MODULE_e mod)： 查询是否允许模块进入休眠；

hal_pwrmgr_lock(MODULE_e mod)： 禁止模块进入休眠；

hal_pwrmgr_unlock(MODULE_e mod)： 允许模块进入休眠

4 功耗评估及实测数据

4.1 功耗评估模型

为了方便评估功耗，我们引入下图中的一个休眠唤醒周期作为功耗模型来估算平均功耗。

在此模型中，一个休眠唤醒周期由休眠时间(X1), 唤醒时间(X2), 工作时间(X3), 射频发送时间(X4, Tx RF), 和射频接收时间(X5, Rx RF)五部分组成。而 Y1,Y2,Y3,Y4,和 Y5 则表示其对应的各个部分的功耗。

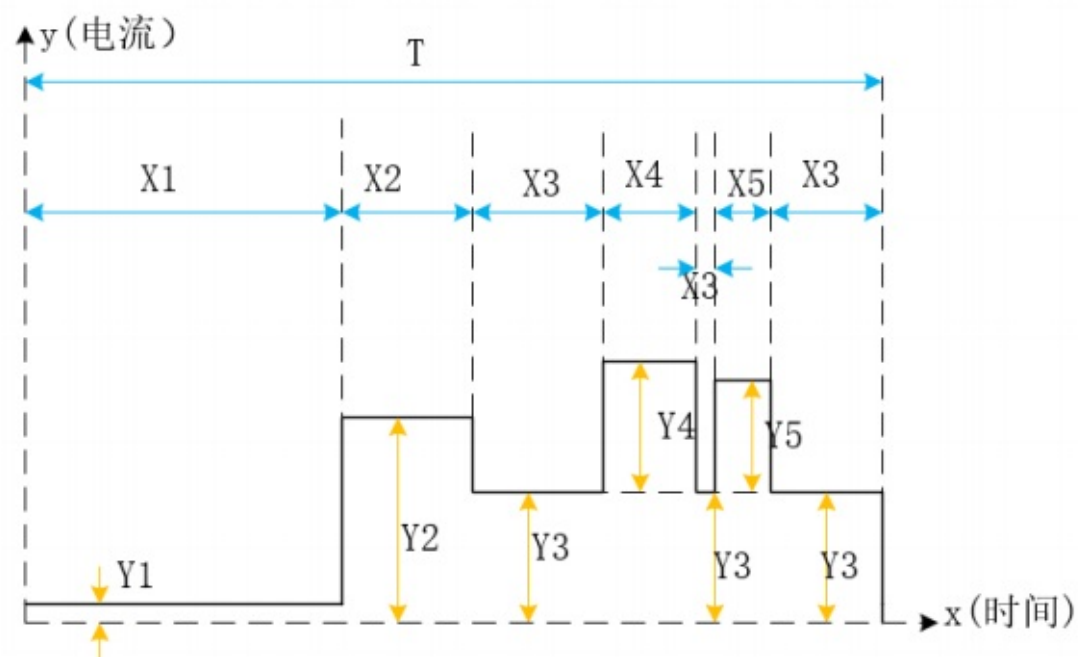


Figure 3：功耗评估模型

其中：

X1: 休眠时间，即系统处于休眠阶段的时间。这阶段对应的功耗为 Y1,影响较大的因素为 RAM retention 的数量，需要 retention 的 RAM 越多，Y1 越大；

X2: 唤醒时间，即系统被唤醒到系统时钟(hclk)切换之前的这段时间。这阶段对应的功耗为 Y2，由于时钟源固定为 32m RC，Y2 的值相对稳定，不过用户可以通过适当地调整唤醒时的参数，缩小唤醒时间 X2，从而达到减小这部分功耗的目的；

X3: 工作时间，即系统切换好系统时钟且不处于射频发送/接收阶段的时间总和。这阶段对应的功耗为 Y3，主要的影响因素为：系统时钟（16/32/48/64M）和 LowCurrentLdo(工程里默认为 enable)。应用程序也需要尽量缩短这部分的时间以减小功耗；

X4: 射频发送时间，即系统处于射频发送阶段的时间。这阶段对应的功耗为 Y4.对于实际的应用而言，这个阶段是可选项，可以没有，有一个或多个。主要的影响因素为：PA 发射功率；

X5: 射频接收时间，即系统处于射频接收阶段的时间。这阶段对应的功耗为 Y5.对于实际的应用而言，这个阶段是可选项，可以没有，有一个或多个。接收功率相对稳定。

4.2 功耗估算

在 4.1 节描述的模型的基础上，我们可以估算功耗如下：

平均功耗 = 总功耗 / 总时间

总功耗 = 休眠时的电流 X 休眠时间 + 唤醒时的电流 X 唤醒时间 + 工作电流 X （工作时间 + 射频发送时间 + 射频接收时间） + 射频发送时的电流 X 射频发送时间 + 射频接收时的电流 X 射频接收时间

总时间 = 休眠时间 + 唤醒时间 + 工作时间 + 射频发送时间 + 射频接收时间

电池使用时间 = 电池容量 X 3600 / 平均功耗 (秒)

4.3 实际测算的功耗

这里以 simpleBlePeripheral 工程为例，描述功耗估测的大致过程：

HCLK	RF time	wakeup time	work time	sleep time	total time	wakeup curt	RF curt	work curt	sleep curt	averag power
64M	676X3us	854us	728x3us	500ms	505ms	3.07ma	3.65ma	2.55ma	5.8ua	0.046
48M	684X3us	886us	738x3us	500ms	505ms	3.03ma	3.60ma	2.26ma	5.6ua	0.044
16M	683X3us	964us	736x3us	500ms	505ms	3.03ma	3.60ma	1.65ma	5.6ua	0.039

Table 1 advertising 功耗测算值

由于 simpleBlePeripheral 工程在唤醒后有三组射频发送/接收的过程，实际测试的波形图如下：

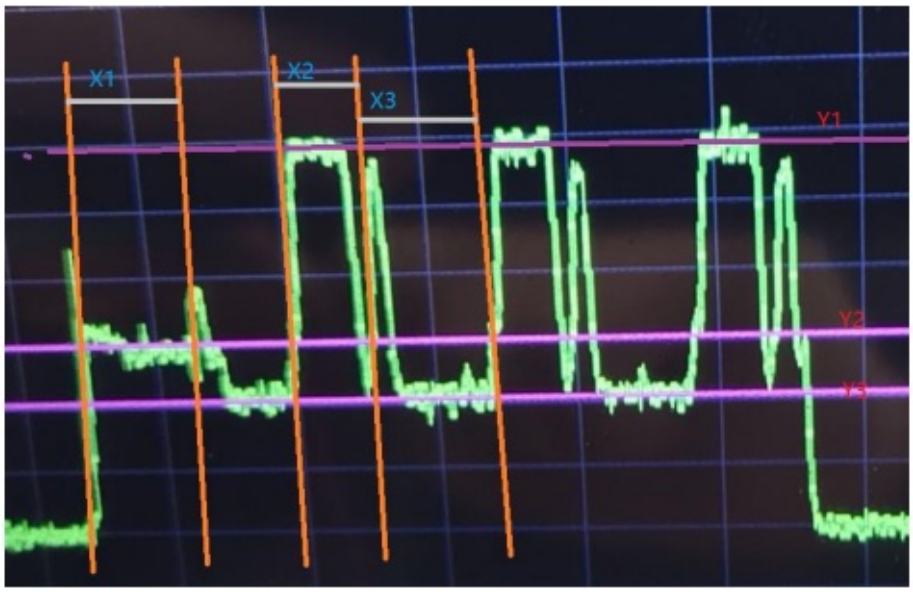


Figure 4：功耗实测图

其中：

X1: 系统唤醒时间，对应的功耗为 Y2;

X2: 一次射频收发的估计时间，对应的功耗为(Y1 – Y3),对于 simpleBlePeripheral 工程，每个休眠唤醒周期有三组射频发送/接收的过程；

X3: 系统唤醒且切换好时钟后，到再次进入休眠之前的这段时间减去射频收发的总时间，对应的功耗为 Y3；

系统在休眠状态的时间及其功耗可以在功率计上直接读取。

参考上一节的功耗模型，可得

总时间 = 休眠时间 + 唤醒时间 + 工作时间 + 射频收发时间(Tx/Rx RF 的总时间)

总功耗 = 休眠时的电流 X 休眠时间 + 唤醒时的电流 X 唤醒时间 + 工作电流 X （工作时间 + 射频收发时间） + 射频发送接收时的电流 X 射频收发时间

平均功耗 = 总功耗 / 总时间

电池使用时间 = 电池容量 X 3600 / 平均功耗 (秒)

例如：电池容量为 520 mAh, 平均功耗为 0.0398，则电池使用时间为

520 X 3600 / 0.0398 = 47035175 秒 = 13065 小时 = 544 天 = 1.5 年

4.4 系统上电启动时间

过程	时间	示例工程
冷启动	91ms	simpleBLEPeripheral
软启动	44ms	simpleBLEPeripheral
Wakeup	3.4ms	bleuart_at
Main 函数入口到 ble 初始化完成	31ms	simpleBLEPeripheral

Table 2 上电启动时间

系统上电可分成以下几种情况：

1. 冷启动， 第一次上电的启动模式，启动时间与需要初始化的代码有关系，这份过程由 rom code 完成。
2. 软启动，software reset。可以配置 AON 寄存器绕开 dwc（大概耗时 50ms）， 剩余时间是代码初始化时间。
3. Wakeup， 这份是从 sram 启动，系统回复速度最快，没有从 flash 搬运代码的时间和 dwc 的时间

AT固件使用

简介

安信可 Ai-Thinker-Combo-AT是安信可开发的可直接用于量产的物联网应用固件，旨在降低客户开发成本，快速形成产品。通过 Ai-Thinker-Combo-AT 指令，您可以快速加入无线网络、连接云平台、实现数据通信以及远程控制等功能，真正的通过无线通讯实现万物互联。

安信可 Ai-Thinker-Combo-AT 是安信可基于不同的芯片厂商软件适配实现的软件工程。它使安信可模组作为从机，MCU 作为主机。MCU 发送 AT 命令给安信可模组，以控制模组执行不同的操作，并接收安信可模组返回的 AT 响应。Ai-Thinker-Combo-AT 提供了大量功能不同的 AT 命令，如 Wi-Fi 命令、TCP/IP 命令、Bluetooth LE 命令、Bluetooth 命令、Bluetooth Mesh 命令、MQTT 命令、HTTP 命令、天猫精灵接入命令等。

AT 命令以 “AT” 开始，代表 Attention，以新的一行 (CR LF) 为结尾。输入的每条命令都会返回 `OK` 或 `ERROR` 的响应，表示当前命令的最终执行结果。注意，所有 AT 命令均为串行执行，每次只能执行一条命令。因此，在使用 AT 命令时，应等待上一条命令执行完毕后，再发送下一条命令。如果上一条命令未执行完毕，又发送了新的命令，则会返回 `busyp...` 提示。更多有关 AT 命令的信息可参见 AT 命令集。

目前 Ai-Thinker-Combo-AT 工程源码暂不开放。

联系方式

公司官网：https://www.ai-thinker.com

开发资料：https://docs.ai-thinker.com

淘宝店铺：https://anxinke.taobao.com

天猫店铺：https://aithinker.tmall.com

商务合作：sales@aithinker.com

技术支持：support@aithinker.com

联系地址：深圳市宝安区西乡华丰智慧创新港C座403、408~410室

商务电话：0755-29162996

欢迎关注微信公众号 “安信可科技” ，干货实时推送！

BLE MESH AT指令组网教程

TB系列模组MESH APP组网教程

更新时间：2022-01-22 09:00

一、前言

本文以TB-02开发板为例，介绍TB系列蓝牙模组通过安卓“TelinkSigMesh” app实现BLE Mesh组网

二、硬件

TB-02开发板，至少三块，购买链接

三、软件准备

支持TB系列模组实现BLE Mesh组网的资料包

其中包括支持TB系列模组实现BLE Mesh组网的AT指令固件以及相匹配的安卓版“TelinkSigMesh” app

固件烧录软件

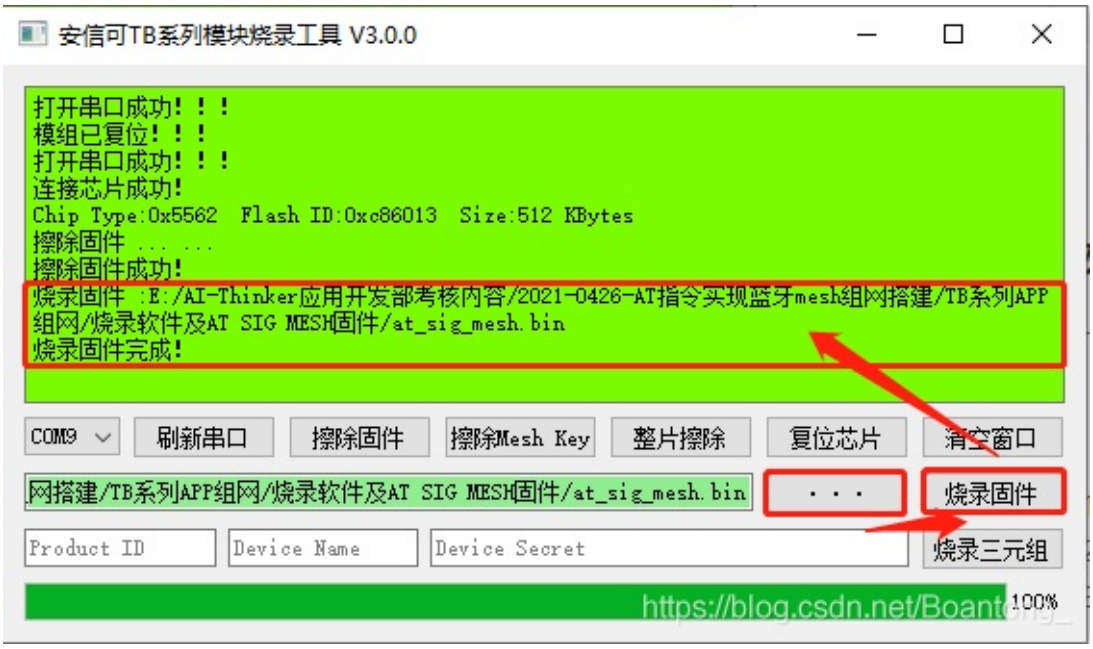
（注：资料包中的固件烧录软件已过时，推荐使用最新版本）

固件烧录步骤

接线：直接将开发板的MicroUSB与电脑通过数据线连接，同时将开发板中的SWS引脚与RXD引脚短接，使开发板进入烧录模式

打开固件烧录软件，选择烧录串口号，点击复位芯片，确保串口工作正常

点击“. . .”按钮，选择烧录的“at_sig_mesh.bin”文件，点击“烧录固件” ，log窗口显示“烧录固件完成”表示固件烧录成功



四、AT指令集

AT 指令	指令说明
AT+SETUP +BLE_CONNECTED +STATE:1 回复+STATE:1，即配网成功	使能节点设备指令 当节点处于 unProvisioning 状态，即未配网过，此时不发送广播，网关无法扫描到此设备并进行连接，如需连接需使用 AT+SETUP 指令使能节点，使得设备能被扫描和连接。 当设备处于 Provisioning 状态，即已经与网关配网过了，无需使能节点，节点自动接入已经配网的 mesh 网络中
AT+TEST=daddr,opcode,pram	daddr 为发送数据的目标地址。 如： 1CA8 为单节点地址（由配网者下发分配） C000 为群组地址（配网者设置） FFFF 为广播地址 opcode 为操作码。 操作码为 0282 pram 是数据部分。 由用户自定义，如灯的开关，亮度和颜色，并且最长 20 个字节，set，get 指令需要回复 ack，也可主动 ack 上报数据。（16 进制格式）
AT+RESTORE	恢复出厂设置
AT+ADDR?	查询网络地址,即源地址
AT+MAC?	查询 MAC 地址
AT+STATE	查询是否配网
{"mesh_data": {"daddr":1CA8,"saddr":1CA9,"opcode":8202,"data_len":4,"data":00040000}}	接收数据为 JSON 格式： daddr:目的地址 saddr:源地址 opcode:操作码 data_len:数据长度 data:用户数据

五、APP 组网示例

完成固件烧录之后，去掉SWS引脚与RXD之间的短接线，打开串口调试助手，配置串口波特率为115200，打开串口，依次输入 AT+SETUP 进入组网状态

（注：若打开串口，开发板复位之后串口调试助手没有任何反应，则切换串口波特率为9600，在复位，若出现乱码，再将串口波特率切换回115200）

在这里插入图片描述安装并且启动“TelinkSigMesh”ANDROID APP。启动后界面如下图一所示。底部导航栏分别为，设备（Device），分组（Group），和设置（Setting）选项页面。

点击网络页面右上的“+”号，进入 Device Scan 开始扫描未配置的节点（UNPROVISIONED NODE）。如下图二所示，附近的未配置的节点将会列于表上，点击希望配置的节点开始配置。成功配置后，APP 提示配置成功，如下图所示，点击“OK”继续。该成功配置的节点，如下图三所示，将会列于表上。

配置成功后，模组串口输出信息。

在这里插入图片描述可以点击 APP 的 ALL ON 和 ALL OFF 按钮进行所有 mesh 设备进行开关控制。模组接收到来自 APP 配网者的数据如下：

```
+BLE_CONNECTED

{"mesh_data":
{"daddr":FFFF,"saddr":1C06,"opcode":8202,"data_len":4,"data":01080000}}

{"mesh_data":
{"daddr":FFFF,"saddr":1C06,"opcode":8202,"data_len":4,"data":00090000}}

+BLE_DISCONNECTED
```

长按APP设备面板的某一设备，跳转到 GROUP，加入一个群组 Libving room 如图一，在 Group 栏即可对 Libving room 进行操作，如图二：

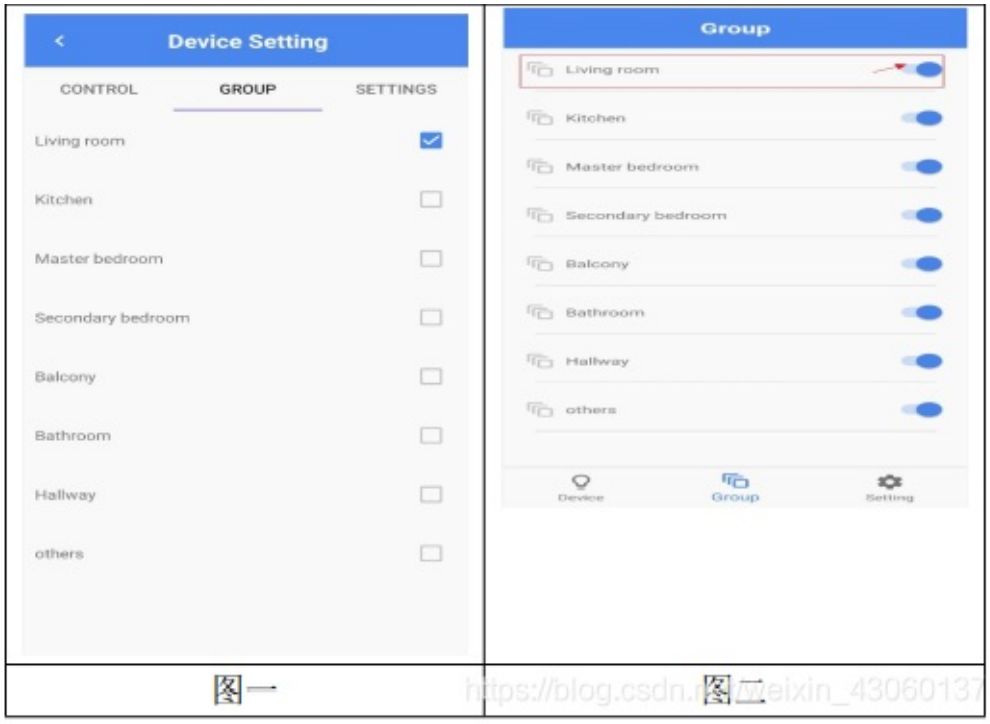
```
+BLE_CONNECTED

{"mesh_data":
{"daddr":FFFF,"saddr":1C06,"opcode":8202,"data_len":4,"data":01080000}}

{"mesh_data":
{"daddr":FFFF,"saddr":1C06,"opcode":8202,"data_len":4,"data":00090000}}

+BLE_DISCONNECTED
```

群组控制，模组返回群组控制的消息。



图一　　图二

六、数据通信

APP 给两个模组分配的地址若各为 0x1CA8,0x1CA9，可以把地址为 0x1CA9 的节点加入群组 Living Room,群组地址为 0xC000。

如下表：设备 1 可以通过 AT+TEST 指令分别向设备 2，群组 1，所有设备发送消息。APP 也可以通过群组控制下发信息。

发送方 / 接收方	设备 1（0x1CA8）	设备 2（0x1CA9）	群组 1（0xC000）	所有设备（0xFFFF）
设备 1（0x1CA8）	-	AT+TEST=1CA9,0282,0101	AT+TEST=C000,0282,0101	AT+TEST=FFFF,0282,0101
设备 2（0x1CA9）	AT+TEST=1CA8,0282,0101	-	AT+TEST=C000,0282,0101	AT+TEST=FFFF,0282,0101
APP 群组 1（0xC000）	APP 控制	APP 控制	APP 控制	APP 控制
APP 所有（0xFFFF）	APP 控制	APP 控制	APP 控制	APP 控制

该组网教程，需要烧录固件，相关配网APP及固件资源下载链接：https://axk.coding.net/s/967aab56-fe56-4d2c-9a25-7bc301d63292

联系方式

公司官网：https://www.ai-thinker.com

开发资料：https://docs.ai-thinker.com

淘宝店铺：https://anxinke.taobao.com

天猫店铺：https://aithinker.tmall.com

商务合作：sales@aithinker.com

技术支持：support@aithinker.com

联系地址：深圳市宝安区西乡华丰智慧创新港C座403、408~410室

商务电话：0755-29162996

欢迎关注微信公众号 “安信可科技”，干货实时推送！

PB系列模组MESH APP组网教程

更新时间：2022-01-22 09:00

一、前言

本文以PB-02开发板为例，介绍PB系列蓝牙模组通过安卓“PHY Mesh” app实现BLE Mesh组网

二、硬件准备

PB-02开发板，至少三块，购买链接

USB转TTL（模组烧录的时候要用到）

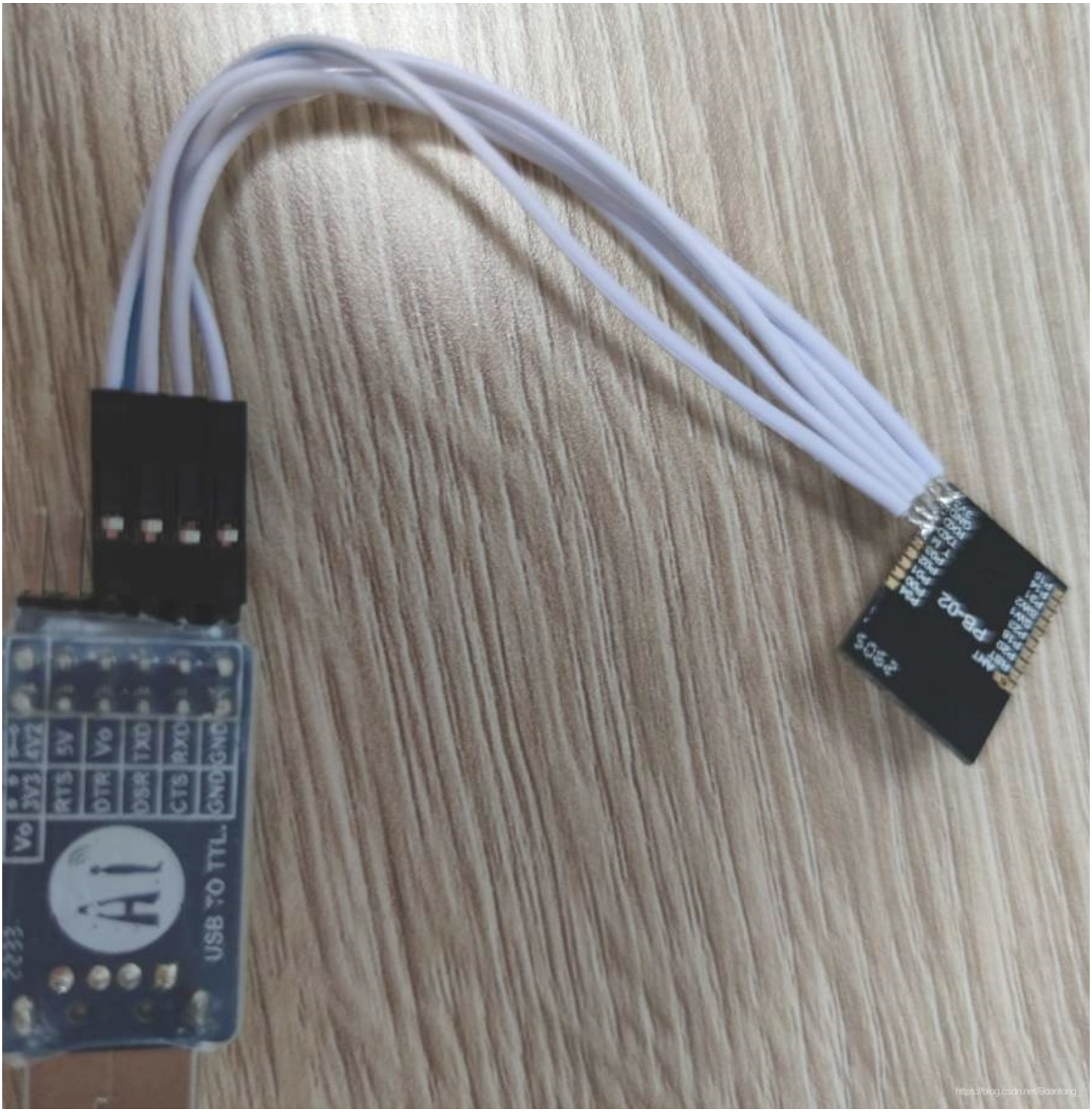
硬件接线：

PB-01/PB-02开发板烧录只需直接将MicroUSB口连接电脑即可

PB系列模组的接线如下表

PB系列模组	USB转TTL
GND	GND
3V3	Vo

Tx	Rx
Rx	Tx
TM	DTR



三、软件准备

支持PB系列模组实现BLE Mesh组网的资料包

其中包括支持PB系列模组实现BLE Mesh组网的AT指令固件以及相匹配的安卓版“PHY Mesh” APP

固件烧录软件

语言: zh en

Search

资源汇总

一、规格书

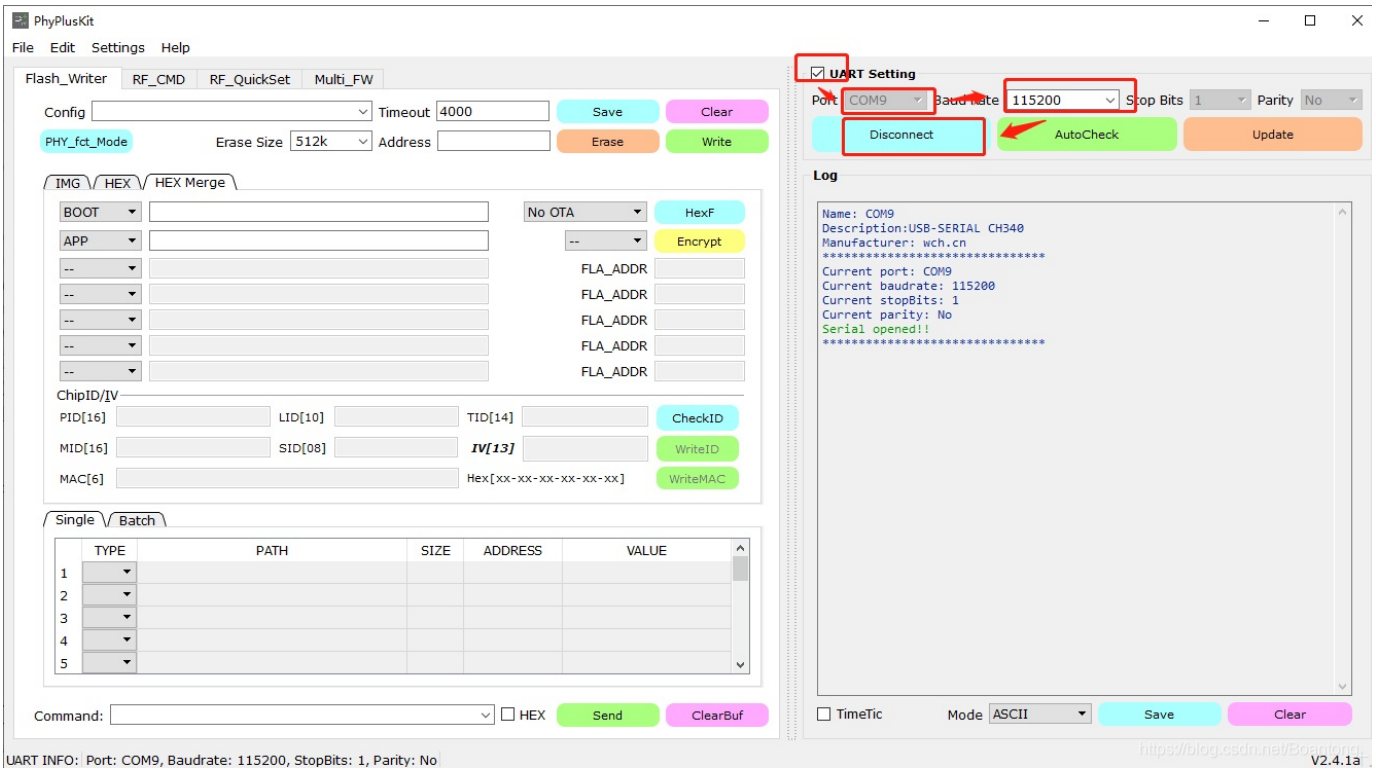
- PB-01规格书：PB-01规格书 PB-01_specification_en.pdf
- PB-02规格书：PB-02规格书 PB-02_specification_en.pdf
- PB-01/02-Kit 开发板 规格书：PB-01/02-Kit 开发板规格书
- PB-01/02 天猫精灵对接AT指令集: 中文
- PB-01/02 蓝牙ble透传(非mesh) AT指令集: 中文
- PB-01/02 模组Mesh组网资料：https://axk.coding.net/s/d7b73016-929a-49a5-8031-9a36c529eb45

二、应用资料

- 二次开发蓝牙SDK源码：https://github.com/Ai-Thinker-Open/AiThinker-Open-PB-BleMesh
- 二次开发烧录工具：pb-download.zip
- PB-01/02 天猫精灵对接AT专有固件: 固件
- 微信小程序控制PB系列蓝牙模块源码： https://github.com/Ai-Thinker-Open/AiPBxxForWeChat
- 【安信可PB-01/02模组专题①】PB-01/02模组开发板应用- BLE-UART固件的使用教程：https://aithinker.blog.csdn.net/article/details/109474628
- 【安信可PB-01/02模组专题②】PB-01/02模组天猫精灵冷暖七彩灯应用：https://aithinker.blog.csdn.net/article/details/109535060
- 【安信可PB-01/02模组专题③】PB-01/02模组开发板应用-快速入门SDK二次开发：https://aithinker.blog.csdn.net/article/details/109253577
- 【安信可PB-01/02模组专题④】ESP32-G蓝牙网关与PB02模组开发进行组网通讯：https://aithinker.blog.csdn.net/article/details/109839685

固件烧录步骤

1. 打开PhyPlusKit烧录软件，点击UART Setting，打开串口配置界面，选择进行烧录的端口号，串口波特率选择115200，点击“connect”按钮；



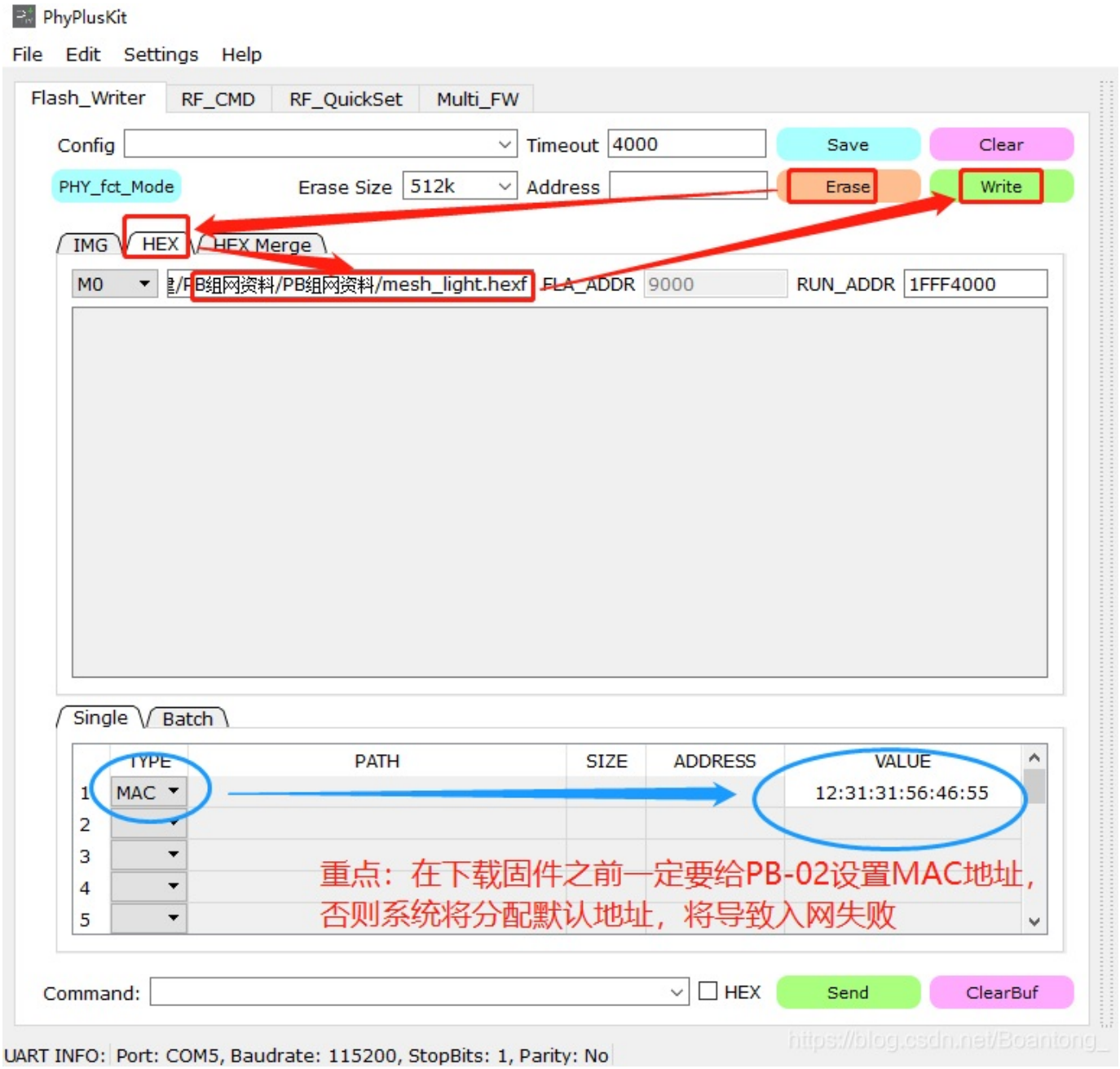
先后按下PB-02开发板的PROG以及RST按键（尽量快，几乎同步），当串口监视窗口（log窗口）中显示“UART RX : cmd>>:”表示开发板已进入烧录模式，点击Erase，擦除开发板中的源固件；

（注：PBxx系列模组在烧录的时候需要将TM引脚拉高，使模组进入烧录模式，待烧录结束之后，再将TM引

脚拉低，进入运行模式）

在烧录配置对话框中选择HEX文件烧录，双击M0后面的文本框，选择要烧录的hex文件，点击Write，开始烧录固件，当log窗口显示“Write images successfully”表示固件烧录成功；

（注：在烧录之前，先给模组设置MAC地址，否则将入网失败）


PhyPlusKit
File Edit Settings Help
Flash_Writer
RF_CMD
RF_QuickSet
Multi_FW
Config
Timeout
4000
Save
Clear
PHY_fct_Mode
Erase Size
512k
Address
Erase
Write
IMG
HEX
HEX Merge
M0
/PB组网资料/PB组网资料/mesh_light.hexf
FLA_ADDR
9000
RUN_ADDR
1FFF4000
Single
Batch
TYPE
PATH
SIZE
ADDRESS
VALUE
1
MAC
12:31:31:56:46:55
2
3
4
5
重点：在下载固件之前一定要给PB-02设置MAC地址，
否则系统将分配默认地址，将导致入网失败
Command:
HEX
Send
ClearBuf
UART INFO: Port: COM5, Baudrate: 115200, StopBits: 1, Parity: No
https://blog.csdn.net/Boantong_

四、AT指令集

AT 指令	指令说明
AT+SETUP 回复 connect ok，即配网成功	使能节点设备指令 当节点处于 unProvisioning 状态，即未配网过，此时不发送广播，网关无法扫描到此设备并进行连接，如需连接需使用 AT+SETUP 指令使能节点，使得设备能被扫描和连接。 当设备处于 Provisioning 状态，即已经与网关配网过了，无需使能节点，节点自动接入已经配网的 mesh 网络中
AT+TEST=daddr,opcode,pram	daddr 为发送数据的目标地址。 如： 0004 为单节点地址（由配网者下发分配） C000 为群组地址（配网者设置） FFFF 为广播地址 opcode 为操作码。 操作码为 d08888 时，为 get 指令。 操作码为 d18888 时，为 set 指令。 操作码为 d38888 时,为 ACK 指令。 pram 是数据部分。 由用户自定义，如灯的开关，亮度和颜色，并且最长 20 个字节，set，get 指令需要回复 ack，也可主动 ack 上报数据。（16 进制格式）
AT+RESTORE	恢复出厂设置
{"mesh_data vendor":{"daddr":3,"saddr":2,"opcode":d38888 ,"data_len":2,"data":0101(为 hex 字符串) ret:1}}	接收数据为 JSON 格式： daddr:目的地址 saddr:源地址 opcode:操作码 data_len:数据长度 data:用户数据

五、APP组网示例

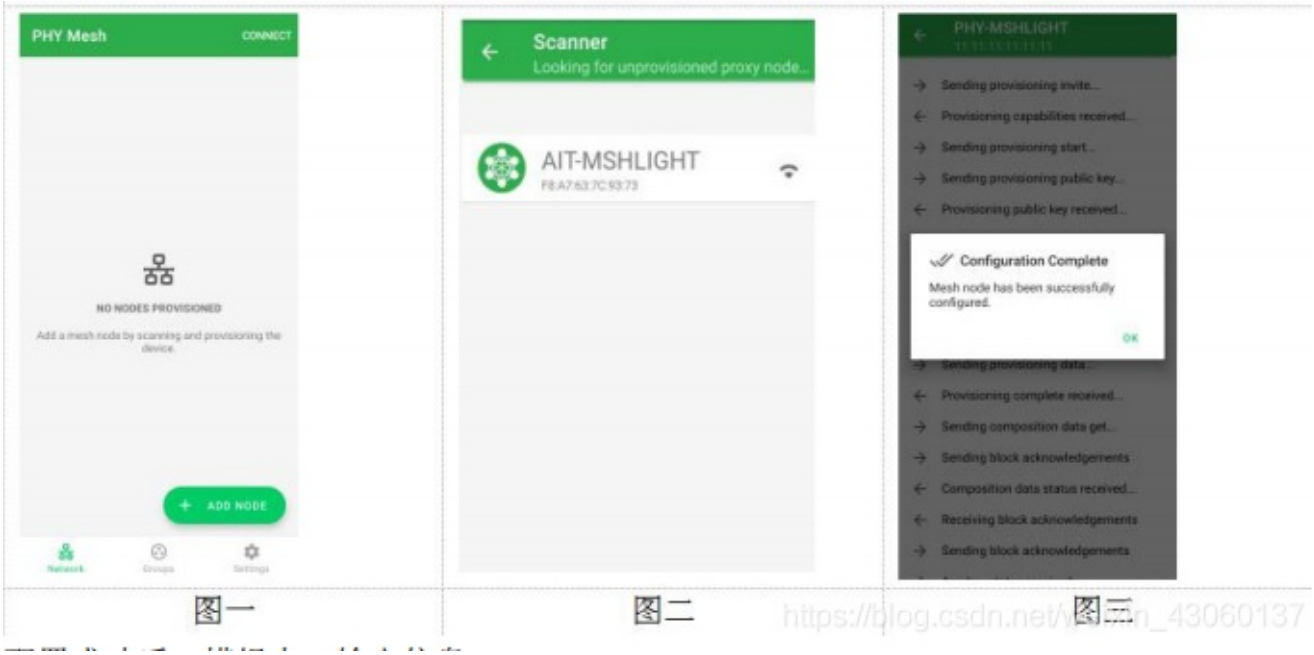
完成固件烧录之后，打开串口调试助手，配置串口波特率为115200，打开串口，依次输入 AT+SETUP 进入组网状态；

开发板或者模组的准备工作结束之后，手机打开安装好的“PHY Mesh” app，启动后界面如下图一所示。底部导航栏分别为，网络（NETWORK），分组（GROUPS），和设置（SETTINGS）选项页面。

点击网络页面的“ADD NODE”开始扫描未配置的节点UNPROVISIONED NODE）。如下图二所示，附近的未配置的节点将会列于表上，点击希望配置的节点开始配置。

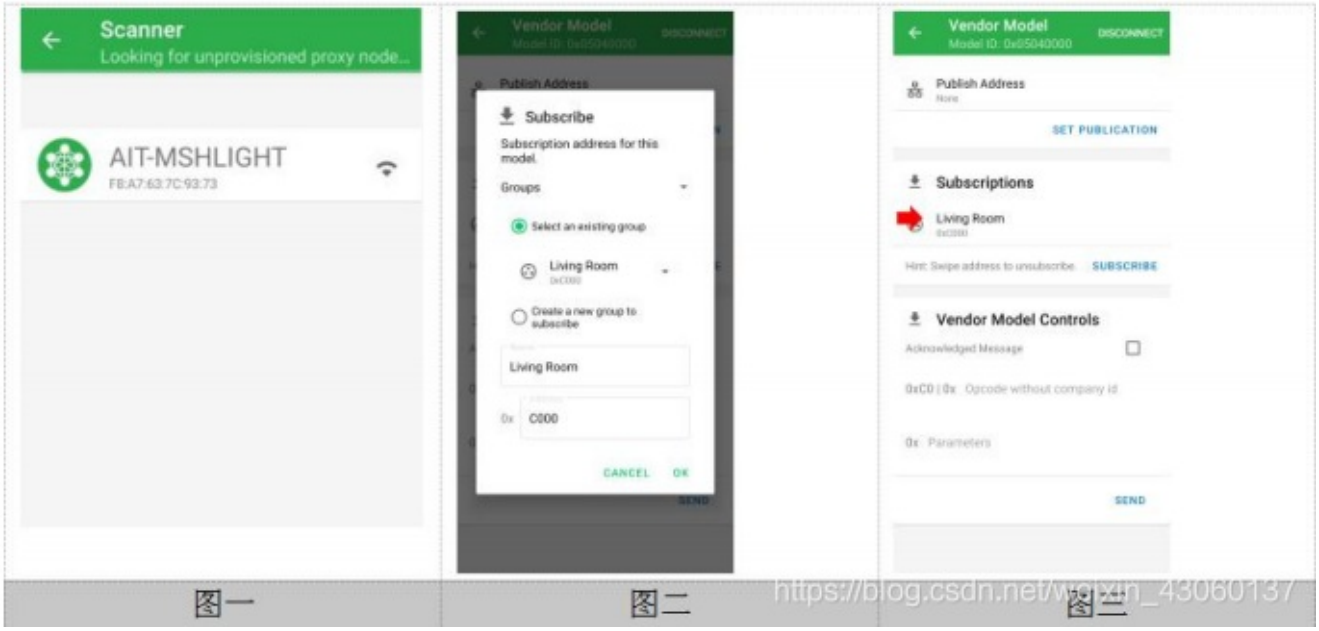
成功配置后，APP 提示配置成功，如下图所示，点击“OK”继续。该成功配置的节点，如下图三所示，将会列

于表上。



配置成功后，模组串口输出信息。

节点需要首先分组后才能控制操作，点击左上角第一个图标进行分组。灯具类节点需点击“SUBSCRIBE”订阅分组，如下图一所示。用户可以选择订阅已有分组，或者选择创建新分组，点击“OK”确认。

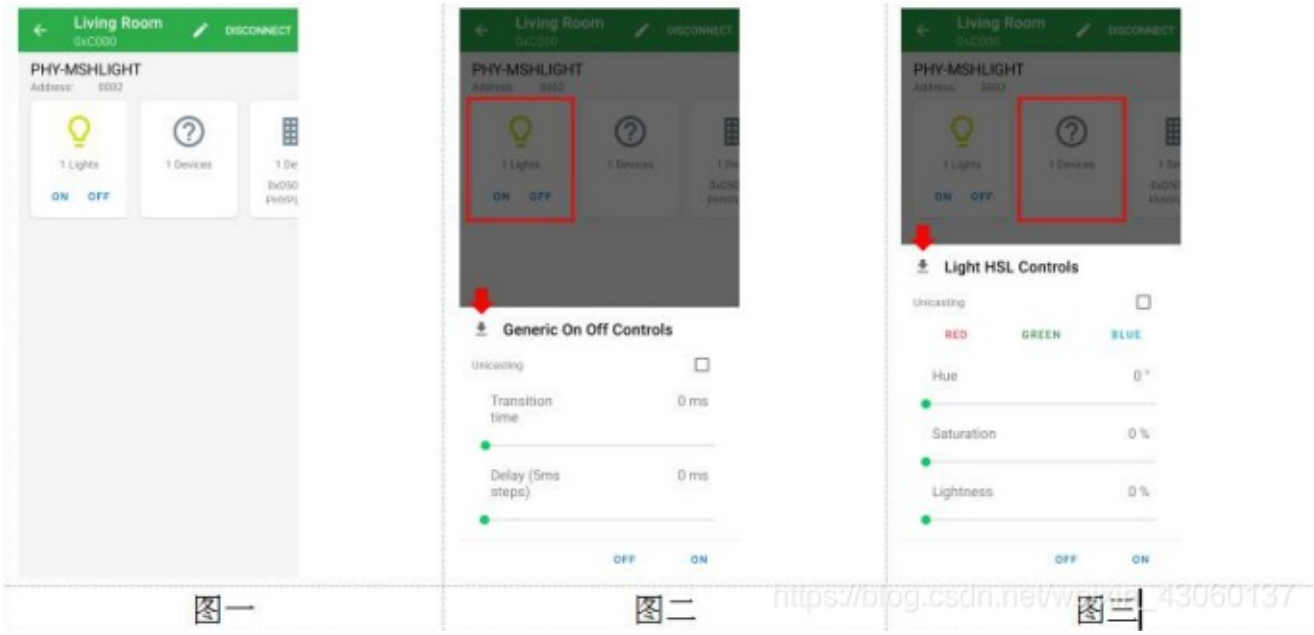


点击分组页面，切换至分组列表，如下图所示，列表列出所有的已创建的分组，以及各分组内所有的设备数量。

灯具类节点需点击分组项目，进入子页面控制开关，以及调色，如下图所示。

所有控制操作（开关，调色，及其他），都是组控制的。对某个节点的控制操作，同属于该节点所在组的其它节

点都有效。如果需要对某个节点单控操作，需要勾选“Unicasting”然后再控制操作。

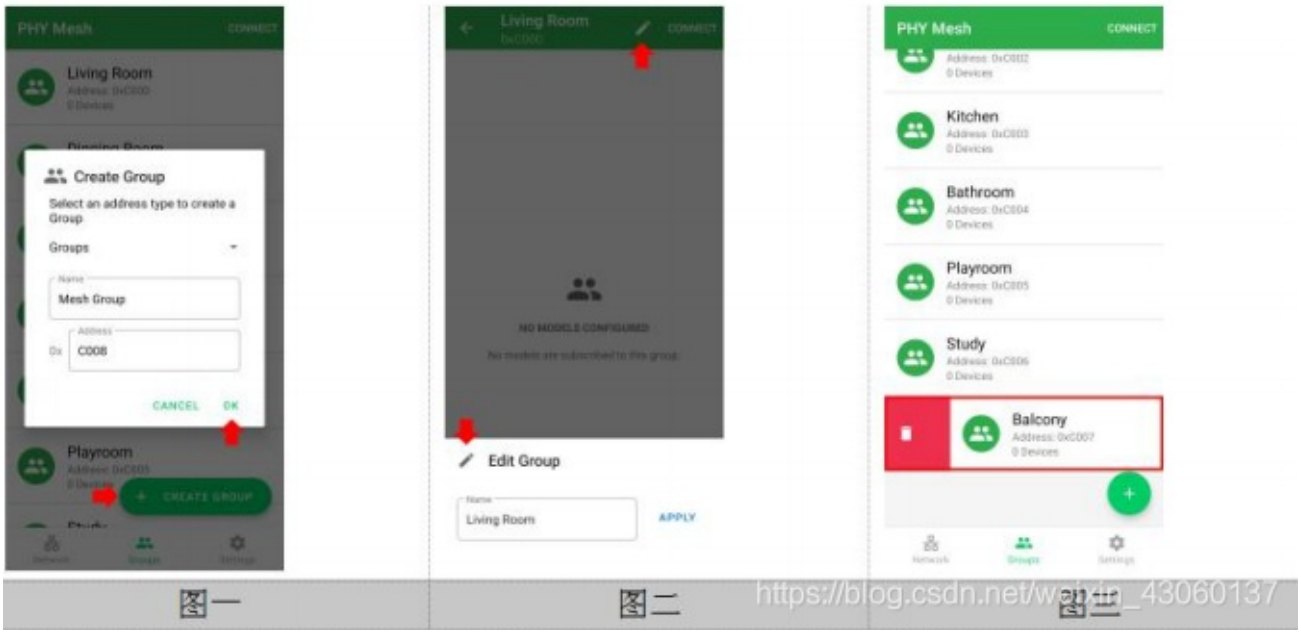


在 PHY Mesh APP 中，用户可以创建新分组，修改分组名，删除分组。

创建分组：点击分组页面，切换至分组列表，如下图所示，点击“CREATE GROUP”创建分组，输入希望的组名和地址，或者保留缺省设置，然后点击“OK”创建。新分组将会更新显示在列表中。

修改分组：点击分组页面，切换至分组列表，如下图所示，输入希望的新的组名，然后点击“APPLY”生效。

删除分组：点击分组页面，切换至分组列表，如下图所示，向左或者向右滑动需要删除组项，则可删除该分组。

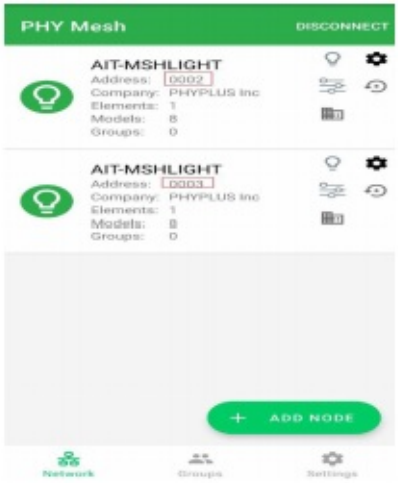


通过点击设置页面的下拉菜单，APP 提供导出与导入功能，方便用户更换手机或者其它的节点配置设备时，迅速恢复原网络关键配置（包括 NETWORK KEYS，APP KEYS，节点，分组及其他配置）。

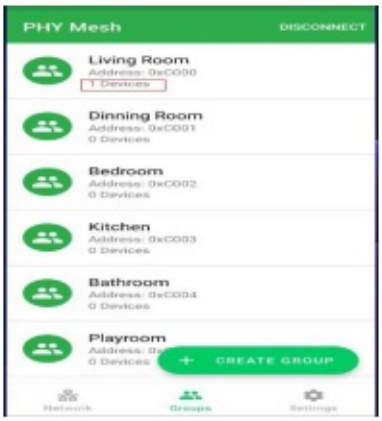
图一

六、数据通信

APP 给两个模组分配的地址各为 0x0002,0x0003，并把地址为 0x0002 的节点加入群组Living Room,群组地址为 0xC000



图一

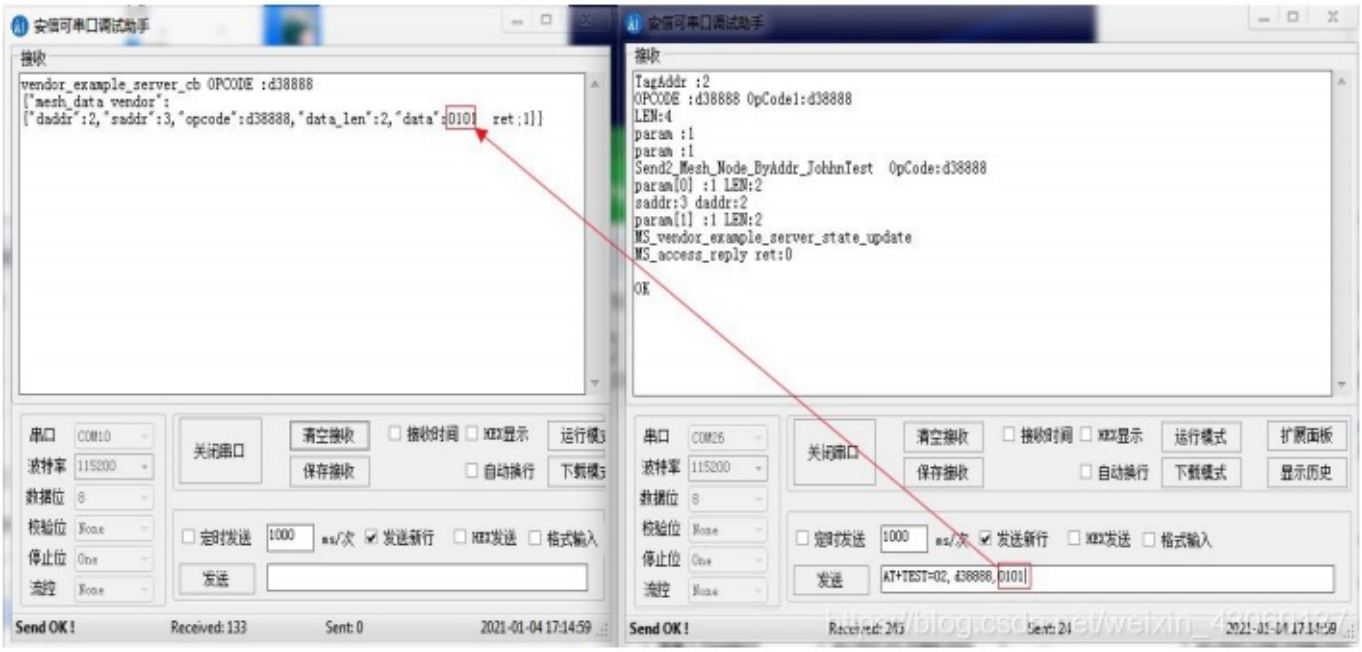


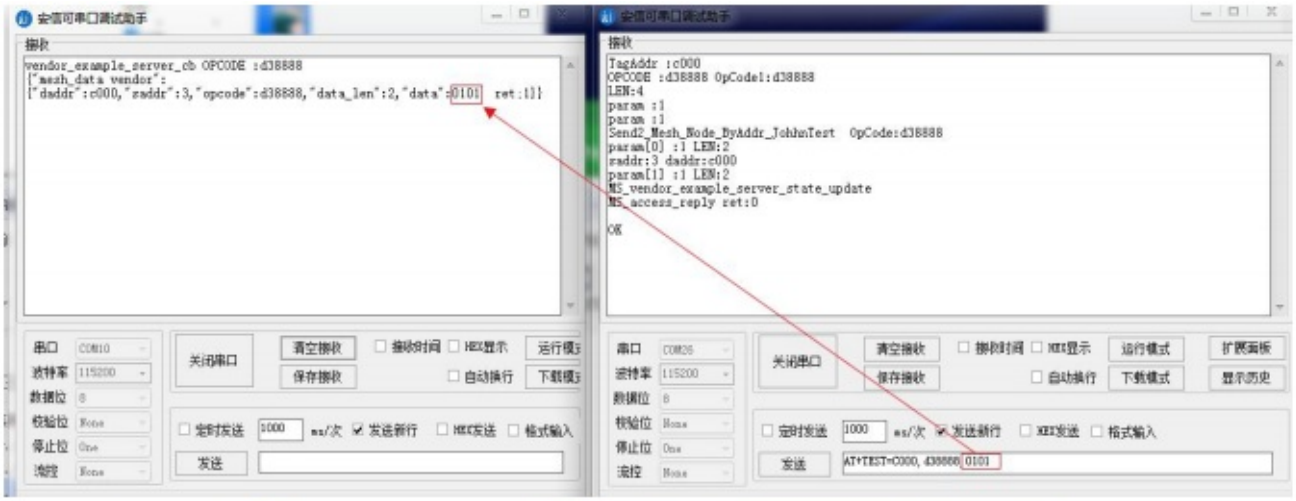
图二

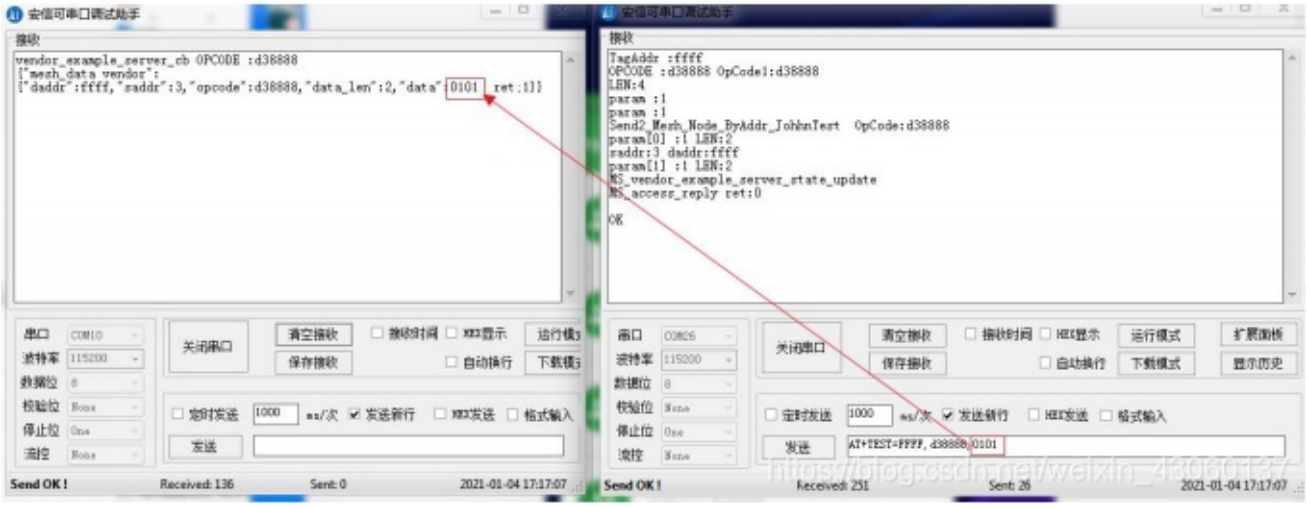
如下表：设备 1 可以通过 AT+TEST 指令分别向设备 1，群组 1，所有设备发送消息。APP 也可以通过群组控制下发信息。

发送方 \ 接收方	设备 1（0x0002）	设备 2（0x0003）	群组 1（0xC000）	所有设备（0xFFFF）
设备 1（0x0002）	-	AT+TEST=03,d3888,0101	AT+TEST=C000,d3888,0101	AT+TEST=FFFF,d3888,0101
设备 2（0x0003）	AT+TEST=02,d3888,0101	-	AT+TEST=C000,d3888,0101	AT+TEST=FFFF,d3888,0101
APP 群组 1（0xC000）	APP 控制	APP 控制	-	-
APP 所有（0xFFFF） 注：参加 APP 没有用该地址，可以把所有设备添加一个群组里作为所有群组使用。	APP 控制	APP 控制	APP 控制	APP 控制

附：AT 指令串口操作截图







该组网教程，需要烧录固件，相关配网APP及固件资源下载链接：https://axk.coding.net/s/d7b73016-929a-49a5-8031-9a36c529eb45

联系方式

公司官网：https://www.ai-thinker.com

开发资料：https://docs.ai-thinker.com

淘宝店铺：https://anxinke.taobao.com

天猫店铺：https://aithinker.tmall.com

商务合作：sales@aithinker.com

技术支持：support@aithinker.com

联系地址：深圳市宝安区西乡华丰智慧创新港C座403、408~410室

商务电话：0755-29162996

欢迎关注微信公众号 “安信可科技” ，干货实时推送！

TB系列模组自组网教程

更新时间：2022-02-16 11:00

##概述

TB系列的ble mesh自组网方式，是无需网关配网，仅通过设置相同的网络名称和密码即可完成自动组网，适合小规模无需网关的局域组网使用场景。

自组网AT mesh固件下载

BLE MESH自组网AT指令表

序号	指令	功能	备注
1	AT+GMR	查询固件版本	
2	AT+RST	重启模组	
3	AT+ADDR	查询或设置模块的地址	
4	AT+MESHNAME	查询或设置模块的网络名称	
5	AT+MESHPWD	查询或设置模块的网络密码	只能设置
6	AT+SEND	发送数据	
7	+DATA	收到数据	
8	AT+ RESTORE	恢复出厂设置	恢复后将重启
9	AT+TTL	查询或设置数据包重传次数	
10	AT+ADV	查询或设置广播功能	
11	AT+BUAD	查询或设置波特率	目前只支持9600，19200 ，38400 ，115200
12	AT+MAC	查询或设置 MAC	
13	AT+SLEEP	设置深度睡眠（串口唤醒）	

##应用示例

网络密钥

此 Mesh 网络采用 `MESHNAME` 和 `MESHPWD` 进行加密(可对应于 SigMesh 中的

`NetworkKey` 和 `ApplicationKey`)，只要两个设备的 `MESHNAME` 和 `MESHPWD` 相同，即可互发数据。

模块出厂默认的 `MESHNAME` 是 `at_mesh` ，

默认的 `MESHPWD` 是 `123456` ，用户可以在 `vendor/commom/user_config.h` 文件中修改这些默认值。

模块出厂后，亦可通过 AT 指令配置 `MESHNAME` 和 `MESHPWD` ，指令如下：

设置网络名称：

AT+MESHNAME=my_mesh123456

设置网络密码：

AT+MESHPWD=my_password

> 备注： `MESHNAME` 和 `MESHPWD` 最大长度为 16 个字节。

查询模块地址：

首次上电会生成一个随机地址

AT+ADDR?

+ADDR:00a8

设置模块地址:

AT+ADDR=89

模块间发送数据：

(向地址为 a8 的模块发送5个数据，内容为 12345)

AT+SEND=a8,5,12345

模块发广播包:

AT+SEND=ffff,5,12345

所有在线设备都可以收到

模块向手机发送数据:

手机的地址定义为：0xFFF0，只要手机连上 Mesh 网络中的任一设备，网内的其他设备都可以与手机通信(即使该设备没有直接与手机连接)

AT+SEND=FFF0,5,12345

手机向模块发送数据：

目前手机只能与与之相连的模块通信，暂不支持与网内其他模组通信。数据可以通过直连节点转发到其他节点

联系方式

公司官网：https://www.ai-thinker.com

开发资料：https://docs.ai-thinker.com

淘宝店铺：https://anxinke.taobao.com

天猫店铺：https://aithinker.tmall.com

商务合作：sales@aithinker.com

技术支持：support@aithinker.com

联系地址：深圳市宝安区西乡华丰智慧创新港C座403、408~410室

商务电话：0755-29162996

欢迎关注微信公众号 “安信可科技” ，干货实时推送！

AT 指令集汇总

AT指令0.8版本

AT指令0.9版本

SIG MESH AT版本指令

1.1AT+PROVISION蓝牙设置启动配网功能

AT+MESHSEND=addr,opcode,data	
描述	
参数	addr：目标的地址 opcode：操作码 data：数据

1.2AT+MESHSEND SIG-MESH发送数据

1.3AT+MESHADDR查询节点地址

1.4AT+MESHSTATE查询是否配网成功

FAQ

常规出厂AT固件的TB-02/03F/04模组有以下特殊指示引脚

蓝牙连接指示引脚 PC4

蓝牙浅休眠低功耗指示引脚 PC3

蓝牙透传/AT指令切换引脚 PB6

TB02开发板中 USB转串口的TX RX是连接到 TB02的 哪2个PIN ？ TB02开发板中 排针的TXD RXD分别是哪2个PIN？

TB02开发板中USB转串口的TXD RXD是连接到TB02的分别是PB1和 PB7

TB02开发板中 排针的TXD RXD分别是PB1，PA0.

请问使用TB04模块蓝牙连接天猫精灵音箱，TB04模块使用的是锂电池供电，待机功耗方面怎么处理才能最低？

第一种，采用Telink Kite BLE SDK Developer Handbook手册介绍的deep-sleep模式，去自定义实现定时唤醒/进入

第二种，采用Telink SIG Mesh SDK Developer Handbook介绍的8258_spirit_LPN工程，即实现对接天猫精灵的lpn节点