ZS110A BLE 透传模组用户指南

发布 ***1.0.0***

2018 年 **12** 月 **22** 日

目录

CHAPTER 1

文档介绍

1.1 文档目的

提供透传模组的设计说明，供模组方案开发者开发参考。

1.2 术语说明

表 1.1: 术语说明

术语	说明
ZEPHYR	为所有资源受限设备，构建了针对低功耗、小型内存微处理器设备而进行优化的物联网嵌入式小型、可扩展的实时操作系统（RTOS）
BLE	Bluetooth Low Energy，低功耗蓝牙
设备 MCU	没有 BLE 的控制处理单元
智能设备	具有 BLE 功能的移动终端设备

1.3 参考文档

- http://docs.zephyrproject.org/

1.4 版本历史

表 1.2: 版本历史

日期	版本	注释	作者
2018-12-07	1.0	初始版本	ZS110A 项目组

CHAPTER 2

透传模组概述

2.1 概述

BLE 透传方案通过 BLE 技术实现了设备 MCU 与蓝牙 BLE 智能设备之间的透明连接。BLE 模组将从设备 MCU 收到的数据包转为 BLE 属性通知给智能设备。同时，将智能设备写入属性的数据通过 Uart 等数据接口发给设备 MCU。

设备 MCU 只需使用标准物理 UART 等接口连接 BLE 模组，使用标准 UART 协议发送和接收数据包。

通过 BLE 透传方案，设备开发商无需了解 BLE 技术细节，只需规定好设备与外界连接的接口应用通讯协议，即可建立与智能设备的无线连接，实现与智能设备的双向通信。在引入最新的蓝牙无线技术的同时，无需增加额外的学习工作，给设备开发带来极大的方便。

2.2 透传模组规格

- 支持 Uart 作为外部 MCU 和 ATB110X 交互的通讯接口。其中 Uart 可支持 576000bps、115200、1Mbps、4Mbps 等多种通讯速率。
- 模组支持命令工作模式和透传工作模式两种工作状态，命令工作模式实现对 ATB110X 模组相关配置功能，透传工作模式实现 MCU 和智能设备数据透明转发。
- 设备 MCU 支持 AT 指令对模组进行命令控制操作及配置。
- 暂时只支持用作 slave 模式

2.3 透传协议栈说明

2.3.1 AT 指令配置模式

模组在非连接状态接收的串口数据会被当成 AT 指令进行处理;

在连接状态，接收的串口数据会被当成转发数据通过 BLE 发送给主机。

即只能在模组非连接的状态通过发送 AT 指令配置模组。

2.3.2 休眠/复位

无论模组目前处于广播状态还是连接状态，EN 管脚输入上升沿，模组会立即进入深度休眠状态；再输入下降沿，模组唤醒开始广播（系统复位，但会保持之前所设定的参数）。

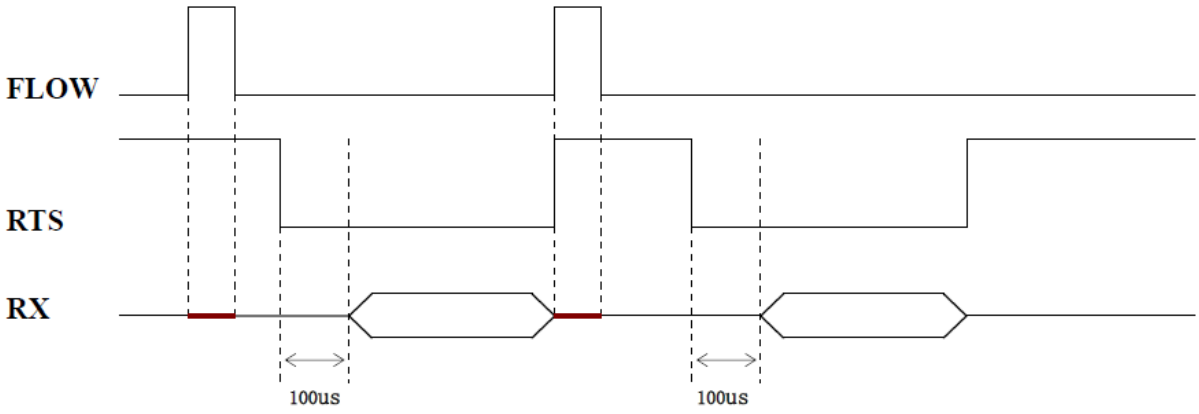
模组可以通过管脚或者 AT 指令两种方式对模组进行复位。具体的 AT 指令说明第一条 AT+STRS 指令。通过给 RSET 管脚一个上升沿也将致使管脚复位。通过这两种复位方式将会使数据恢复成默认的初始状态，并重新上电开始广播

2.3.3 主控 MCU 发送数据到透传模组

- 数据发送控制说明

  模组串口流控制管脚 FLOW 管脚指示模组当前的串口忙闲状态。FLOW 管脚为高电平：指示透传模组串口当前处于忙状态，主控 MCU 不可发送数据给透传模组；当管脚处于低电平时，指示当前处于闲状态，可以发送数据给透传模组

  如果有使能 RTS 检测管脚（默认使能），主控 MCU 如有数据发送到透传模组，需先给出下降沿（使能串口接收模式），发送数据完毕后，主控 MCU 应主动拉高 RTS，让模组退出串口接收模式，否则会增加透传模组功耗。拉高 RTS 之前请确保串口数据已发送完毕，否则会出现数据截尾现象



- 数据转发速率说明

  从主设备方（手机）向透传模组发送数据，需要自行分包发送 (1 — 20byte/packet)。透传模组收到数据之后，会将数据通过串口转发到主控 MCU。

  从主控 MCU 向透传模组发送数据，模组会根据数据包的大小量自动分包按序发送。当模组连接间隔为 20ms，串口波特率为 115200 时，模组具有最高理论转发能力。主控 MCU 可以从串口一次最多输入 200 字节数据包：假定每一个连接间隔主机能够接受 4 个包，并且每个包最大载荷为 20 字节，即每个连接间隔最多可传输 80 字节，所以在此条件下具有理论最高转发速率:

  V = 1000ms / 20ms * 20bytes * 4 = 4 KBytes/s

  在对数据进行透传时，需要注意一包数据长度（L，用户一次性串口输入的蓝牙透出模组的数据长度）与每包数据发送间隔（T，在输入蓝牙透传模组一包数据之后的间隔时间）之间的关系。当一包数据长度较大时，需要相对应的拉大发送间隔的长度。反之，如果一包数据长度较小，那可以适当调笑发送间隔的长度。每次发送的一次发送数据包长度与发送间隔以及 BLE 连接间隔（I）的关系如下:

  L/T <= 80/I

  注解:

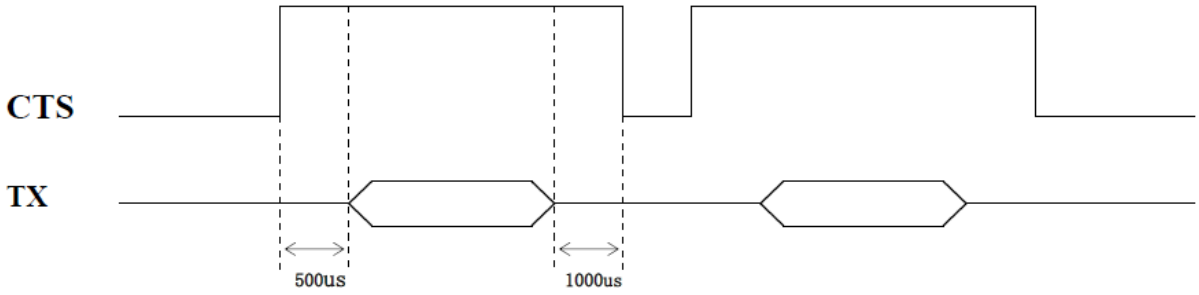
  – 每次连接间隔的最大包数量：对于不同的主机它的包数量是不同的，具体需要根据当前的蓝牙主机能力来进行选择

  – 蓝牙连接间隔：通过 AT 指令可选，最低选择 20ms 的连接间隔

  – 蓝牙连接间隔与一包数据长度一起决定了最大可接受的透传数据量大小

2.3.4 透传模组发送数据到主控 MCU

CTS 可用于唤醒主控 MCU，透传模组接收到蓝牙主机端发送过来的数据后，会通过串口将数据输出到主控 MCU。模组会先拉高 CTS，在 500us 之后开始发送，数据发送完毕，延时 1ms 后模组将会拉低 CTS. 参考时序图如图 3-4 所示。如果不需要这个功能的话，可以选择将此管脚悬空。



CHAPTER 3

AT 指令协议

3.1 指令说明

- 软件复位

指令	应答	参数
AT+STRS	OK+STRS	无

功能描述：使用此命令之后模组将会主动延时 100ms 并重启，然后将模组所有的参数恢复到出厂值

- 广播状态

指令	应答	参数
设置：AT+ADST+[Param]	AT+ADST+[Param]	0:停止广播 1：开始可连接广播 2：开始不可连接的广播 Default: 1
查询：AT+ADST-?	OK+ADST-[Param]	无

功能描述：更改指令参数修改模组广播方式。后带参数 0 将立即停止广播，后带参数 1 将开始可连接的广播，后带参数 2 将开始不可连接的广播

注解： 该参数掉电保存

- 广播间隔

指令	应答	参数
设置：AT+ADIT-[Param]	OK+ADIT-[Param]	0：20 (ms) 1：50 (ms) 2：100 (ms) 3：200 (ms) 4：300 (ms) 5：500 (ms) 6：1000 (ms) 7：2000 (ms) Default: 3
查询：AT+ADIT-?	OK+ADIT-[Param]	无

功能描述：更改指令参数修改模组广播间隔，后带参数对应相应的广播间隔

注解: 该参数掉电保存

- **广播名称**

指令	应答	参数
AT+NAME-[Index]	OK+NAME	无

功能描述：自定义广播名称，设定的最大字符数不能超过 20 个.

注解: 该参数掉电保存

- **波特率**

指令	应答	参数
设置：AT+BAUD-[Param]	OK+BAUD-[Param]	0：9600 (bps) 1：19200 (bps) 2：38400 (bps) 3：57600 (bps) 4: 115200 (bps) Default: 0
查询：AT+BAUD-?	OK+BAUD-[Param]	无

功能描述：该指令通过参数输入修改透传模组的波特率。主控芯片在接到透传模组的应答之后将改变自己的波特率为当前设定值。

注解: 该参数掉电保存

- **连接间隔**

指令	应答	参数
设置：AT+CNIT-[Param]	OK+CNIT-[Param]	0：20 (ms) 1：30 (ms) 2：50 (ms) 3：100 (ms) 4：200 (ms) 5：300 (ms) 6：500 (ms) 7：1000 (ms) 8：2000 (ms) Default：0
查询：AT+CNIT-?	OK+CNIT-[Param]	无

功能描述：改指令参数修改模组连接间隔，后带参数对应相应的连接间隔。

注解: 该参数掉电保存

- **发射功率**

指令	应答	参数
设置：AT+TXPW-[Param]	OK+TXPW-[Param]	0：+5 (db) 1：2 (db) 2：-1 (db) 3：-4 (db) 4：-7 (db) 5：-10 (db) 6：-13 (db) 7：-16 (db) Default：1
查询：AT+TXPW-?	OK+TXPW-[Param]	无

功能描述：该指令通过参数修改透传模组的发射功率，发射功率越好，发射距离及抗干扰性相对增强，但功耗也会相应提高。

注解:

- 该指令执行后，模组会重启才能生效
- 该参数掉电保存

- **MAC 地址**

指令	应答	参数
AT+GMAC	蓝牙地址	无

功能描述：输入该指令即蓝牙地址，返回的蓝牙地址是字符串，格式为××:××:××:××:××:××

- **自定义广播数据**

指令	应答	参数
设置：AT+MAFD-[Index]	OK+MAFD	无
查询：AT+MAFD-?	自定义数据	无

功能描述：自定义广播数据，设定的最大字节数不能超过 6 个字节，超过 6 个字节，如果自定义数据设定不超过 6 个字节，将对其进行自动补零操作。默认的广播数据是 6 个字节的蓝牙地址

ex：AT+MAFD-A1B2C3D4E5F6，广播中增加的 6 个字节为 0xA1,0xB2,0xC3,0xD4,0xE5,0xF6

注解： 该参数掉电保存

- **数据服务 UUID**

指令	应答	参数
设置：AT+SERN-[Index]	OK+SERN	无
查询：AT+SERN-?	OK+SERN-[Index]	无

功能描述：自定义数据服务的 UUID，数据长度为 2 个 byte

注解： 该参数掉电保存

- **驱动服务 UUID**

指令	应答	参数
设置：AT+SERD-[Index]	OK+SERD	无
查询：AT+SERD-?	OK+SERD-[Index]	无

功能描述：自定义驱动服务的 UUID，数据长度为 2 个 byte

注解： 该参数掉电保存

- **数据发送属性 UUID**

指令	应答	参数
设置：AT+CHAT-[Index]	OK+CHAT	无
查询：AT+CHAT-?	OK+CHAT-[Index]	无

功能描述：自定义数据发送属性的 UUID，数据长度为 2 个 byte

注解: 该参数掉电保存

- **数据接收属性 UUID**

指令	应答	参数
设置：AT+CHAR-[Index]	OK+CHAR	无
查询：AT+CHAR-?	OK+CHAR-[Index]	无

自定义数据接收属性的 UUID，数据长度为 2 个 byte

注解: 该参数掉电保存

- **数据 AD 采集属性 UUID**

指令	应答	参数
设置：AT+CHAD-[Index]	OK+CHAD	无
查询：AT+CHAD-?	OK+CHAD-[Index]	无

自定义数据 AD 采集属性的 UUID，数据长度为 2 个 byte

注解: 该参数掉电保存

- **数据 PWM 属性 UUID**

指令	应答	参数
设置：AT+CHAP-[Index]	OK+CHAP	无
查询：AT+CHAP-?	OK+CHAP-[Index]	无

功能描述：自定义数据 PWM 数据的 UUID，数据长度为 2 个 byte

注解: 该参数掉电保存

- **版本信息**

指令	应答	参数
AT+CODV	OK+CODV-[Index]	无

- **RSSI 值检测**

指令	应答	参数
设置：AT+RSSI-[Param]	OK+RSSI-[Param]	0:退出 RSSI 模式 1:进入 RSSI 模式 2：查询 RSSI 值
查询：AT+RSSI-[Param]	OK+RSSI-[Param]	无

功能描述：如果需要得到 RSSI 值，需要进入 RSSI 模式并与蓝牙模组连接并输入相应指令。需要注意的是，在进入 RSSI 模式之后，此时数据不能被透传，除非退出 RSSI 模式

注解: 该参数掉电保存

- **RTS 检测线使能**

指令	应答	参数
设置：AT+CRTS-[Param]	OK+CRTS-[Param]	0：取消 RTS 检测线 1：使能 RTS 检测线 Default: 0
查询：AT+CRTS-?	OK+CRTS-[Param]	无

功能描述：改指令参数修改模组 RTS 使能属性，默认情况下，RTS 数据线是被关闭的，主控 MCU 随时都可以发送数据至透传模组。如果修改参数设置为 1，主控 MCU 每次发送数据到透传模组之前都需要给 RTS 一个下降沿，透传模组才会接受主控 MCU 所传来的数据。注意，如果关闭 RTS 检测线透传模组的功耗会有所增加

注解: 该参数掉电保存

CHAPTER 4

BLE 透传 profile

角色	功能	UUID(默认)	属性
Service	数据传输服务	0001	/
Characteristic	RX	0002	Write
Characteristic	TX	0003	Notify
Service	模组驱动服务	0004	/
Characteristic	ADC	0005	Notify/Write
Characteristic	PWM	0006	Write

4.1 服务及特性的完整 UUID

数据传输服务 UUID：6940-0001-b5a3-f393-e0a9-e50e-24dc-ca99

写数据特性 UUID：6940-0002-b5a3-f393-e0a9-e50e-24dc-ca99

读数据特性 UUID：6940-0003-b5a3-f393-e0a9-e50e-24dc-ca99

模组驱动服务 UUID：7f51-0004-b5a3-f393-e0a9-e50e-24dc-ca9e

写数据特性 UUID：7f51-0005-b5a3-f393-e0a9-e50e-24dc-ca9e

读数据特性 UUID：7f51-0006-b5a3-f393-e0a9-e50e-24dc-ca9e

注解： 服务以及特性的 UUID 上电后默认如上所示，其部分 UUID 是可以通过串口输入 AT 指令进行更改

4.2 模组数据传送服务

4.2.1 数据发送通道

蓝牙输入转发到串口输出。APP 通过 BLE API 接口向此通道写操作后，数据将会从串口 TX 输出

4.2.2 数据接收通道

串口输入转发到蓝牙输出。如果打开了该属性的通知使能开关，MCU 通过串口向模组 RX 发送的合法数据后，将会在此通道产生一个 notify 通知事件，APP 可以直接在回调函数中进行处理和使用

4.3 模组驱动服务

4.3.1 ADC 采集

本模组提供一个通道的 8 位 ADC 采集和一路模组电池电量的采集。通过 BLE API 的接口向通道进行相应的写操作，来操作相应的 ADC 通道

输入的字符数目为 2 个字节，其中输入的第一个字节为校验字节 0xAD，输入的第二个字节为选通字节，其中 1-2 分别代表电池电量采集，ADC0 通道采集。输入命令后在模组会将采集的结果发送的主机端，即得到采集到的电压值。输入格式如下例所示

Ex:

- 采集 Vcc 电压：0xAD,0x01；返回的值为 0x03/0x0A，则表示返回值为 3.10V
- ADC 采集通道 0：0xAD,0x02；同上

4.3.2 PWM 波形输出

本模组提供两个通道的 PWM 输出波形。提供的最高频率为 250kHz，其中 20 级频率可调以及 100 级的占空比可调。通过 BLE API 的接口向通道进行相应的写操作来控制相应的 PWM 通道

如果两个 PWM 通道同时开启，必须保持相同的频率设置，如果在设置中给出的频率不一样，将会以 PWM0 通道的频率为标准。输入的字符数目为 6 个字节，其中输入的第一个字节为校验字节 0x97，第二个字节用以 PWM 通道的选择；其中全部开启为 0x00，全部关闭为 0x01，单独开启通道 0 为 0x02，单独开启通道 1 为 0x03；第三个与第四个字节分别控制 PWM0 通道的频率与占空比，第五个与第六个字节分别控制 PWM1 通道的频率与占空比，频率的选择见下表所示

Ex:

- PWM0 输出频率为 10kHz，占空比为 10%：0x97,0x02,0x0C,0x0A,0x00,0x00;
- PWM0/PWM1 输出频率为 100kHz，占空比分别为 10% 以及 40%:0x97,0x00,0x11,0x0A,0x11,0x28;
- 全部关闭：0x97,0x01,0x00,0x00,0x00,0x00;

频率	命令
50 Hz	1
100 Hz	2
200 Hz	3
400 Hz	4
500 Hz	5
800 Hz	6
1 KHz	7
2 KHz	8
4 KHz	9
5 KHz	10
8 KHz	11
10 KHz	12
20 KHz	13
25 KHz	14
40 KHz	15
50 KHz	16
100 KHz	17
125 KHz	18
200 KHz	19
250 KHz	20

注解: 为了实现方便，固定 PWM 源头频率为 1Mhz，也因为此原因，部分 PWM 的占空比无法调整到指定值，如 125Khz 只有 8 级占空比可调，250Khz 只有 4 级占空比可调

CHAPTER 5

BLE 模组应用工程简介

5.1 模组工程概述

- 工程路径在：\samples\ble_module_peripheral\
- 工程目录结构，如图所示

图 5.1: 工程目录结构图

1. keil 目录主要包含 3 个 keil 工程
2. outdir 目录主要包含工程产生的固件及 OTA 升级包
3. src 目录主要包含工程的源码
4. ble_module 目录下的脚本主要用于生成固件和 OTA 升级包

- 工程主要由 loader 工程、ble_module 工程和 rf_test 工程构成，如图所示:

图 5.2: 工程概况图

1. ble_module 是 BLE 模组应用工程，主要实现 AT 指令、UART 透传等核心功能
2. loader 是应用引导工程，根据不同的需求，引导应用工程或者 ble 频偏测试工程，默认情况，是引导应用工程，如需要测试频偏，可以在应用场景，通过一定方式进入频偏测试模式，进而引导 ble 频偏测试工程。
3. rf_test 是 ble 频偏测试工程。工厂生产组装好产品后需要对产品进行频偏确认时使用。

5.2 loader 工程简介

引导和加载应用工程。

更多与 loader 相关的信息，请参考 <<ZS110A-loader 示例 >> <<ZS110A-SDK 架构介绍 >>

5.3 ble_module 工程简介

5.3.1 ble_module project 目录介绍

- ble_module project 目录结构，如图所示

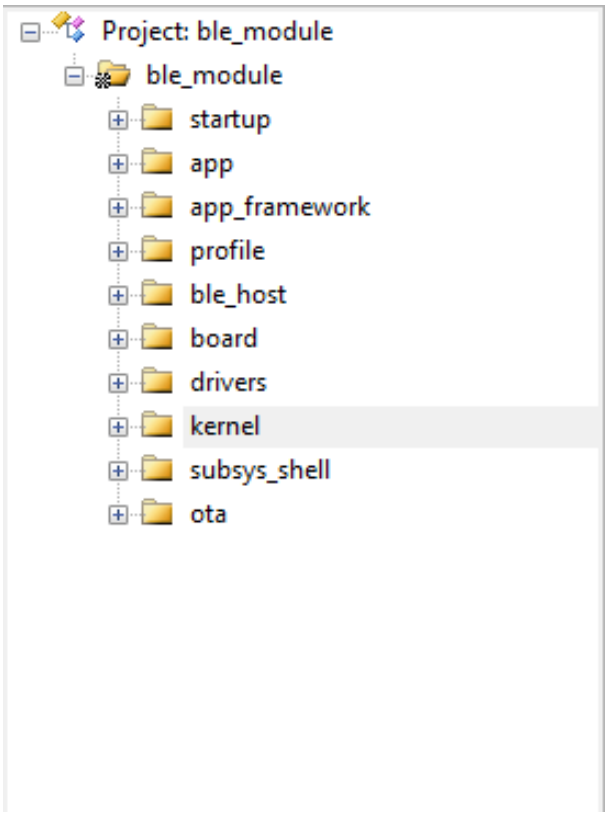


图 5.3: ble_module project 目录结构图

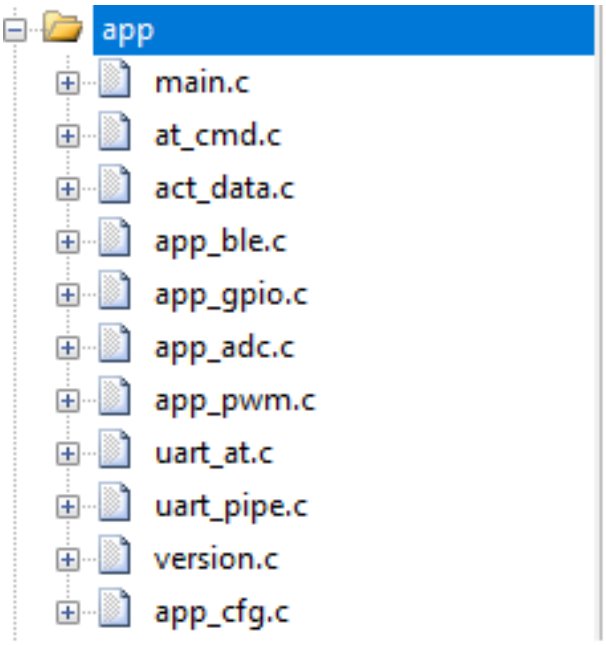
- startup 系统启动相关功能代码
- app 应用层代码
- app_framework 应用层框架代码
- profile Gatt profile 的代码
- bt_host bt host 层代码
- board 板级相关初始化代码
- drivers 驱动相关代码
- kernel Zephyr 内核相关代码
- ota 相关代码
- subsys_shell shell 相关代码

5.3.2 startup 目录代码介绍

关于 startup 启动相关代码，详细请参考 <<ZS110A-SDK 架构介绍 >>

5.3.3 app 目录代码介绍

- 应用层结构如图所示



main.c 介绍

- 应用层相关功能模组初始化

```
/**
 * @brief main module initialization of the application.
 *
 * @details Initialize msg/led/input/sm etc.
 *
 * @return No return value.
 */
__init_once_text void system_app_init(void)
{
        /* init module gpio pin */
        app_gpio_init();

        /* init app pwm */
        app_pwm_init();

        /* init message manager */
        msg_manager_init();

        /* init at command */
        at_cmd_init();

        /* init act data */
        act_data_init();
}
```

- 进入等待消息处理循环

```
/**
 * @brief the entry function of the application.
 *
 * @details Initialize app module and enter into msg mainloop.
 *
 * @return No return value.
 */
void app_main(void)
{
        int err;
        struct app_msg msg = {0};
        int result = 0;

        /* main module initialization */
        system_app_init();

        /* Initialize the Bluetooth Subsystem */
        app_get_wake_lock();
        err = bt_enable(bt_ready);
        if (err) {
                app_release_wake_lock();
                printk("Bluetooth init failed (err %d)\n", err);
                return;
        }

        while (1) {
                if (receive_msg(&msg, K_FOREVER)) {
                        switch (msg.type) {
```

at_cmd.c 介绍

- 初始化相关函数

```
void at_cmd_init(void)
void enter_into_at_cmd_mode(void)
void exit_out_at_cmd_mode(void)
```

- AT 命令处理线程

  1. k_fifo_get 从命令队列取 AT 命令
  2. at_cmd_exec 解析 AT 指令并分发给不同的处理函数
  3. k_fifo_put 命令处理完后，将 buffer 放回 uart 接收队列

```
void at_cmd_handler(void *p1, void *p2, void *p3)
{
        ARG_UNUSED(p1);
```

(continues on next page)

(续上页)

```
        ARG_UNUSED(p2);
        ARG_UNUSED(p3);

        while (1) {
                struct at_cmd_input *cmd;

                cmd = k_fifo_get(p_at_cmds_queue, K_FOREVER);

                at_cmd_exec(cmd->line);

                k_fifo_put(p_at_avail_queue, cmd);
        }
}
```

- AT 应答的发送

  如果上位机处于休眠状态，发送 AT 应答时，需要先唤醒上位机

```
/**
  * @brief  print string to at cmd port
  * @param  string
  * @retval None
  */
void at_cmd_port_print(const char *str)
{
        int i;
        int len = strlen(str);
#ifdef MODULE_CTS_PIN
        gpio_output_wakeup_host(true);
#endif
        for(i = 0; i< len; i++) {
                at_out(str[i]);
        }
#ifdef MODULE_CTS_PIN
        gpio_output_wakeup_host(false);
#endif
}
```

uart_at.c 介绍

- uart 中断回调函数接收命令

  1. 接收 AT 命令
  2. 接收到特殊标记字符后，认为已经接收到一条命令
  3. 将命令放到命令队列，供 at_cmd_handler 线程获取并分析处理

```
void uart_at_isr(struct device *unused)
```

act_data.c 介绍

- 初始化相关函数

```
void act_data_init(void)
void enter_into_data_mode(void)
void exit_out_data_mode(void)
```

- 数据线程的转发

1. k_fifo_get 从数据队列获取数据
2. ble_send_data 将数据经过 BLE 接口转发给移动设备终端
3. k_fifo_put 数据发送到协议栈后，将 buffer 放回 uart 接收队列

```
static void data_handler(void *p1, void *p2, void *p3)
{
        while (1) {
                int err ;
                struct act_data_buf *data;
                data = k_fifo_get(&datas_queue, K_FOREVER);

                /* TODO
                 * verify if service is registered before␣
↪calling handler
                 */
                err = ble_send_data(data->data, data->data_
↪len);
```

- 数据从 BLE 发送到 Uart

```
int act_data_send_data(const u8_t *data, int len)
{
#ifdef MODULE_CTS_PIN
        gpio_output_wakeup_host(true);
#endif
        uart_pipe_send(data, len);
#ifdef MODULE_CTS_PIN
        gpio_output_wakeup_host(false);
#endif
        return 0;
}
```

uart_pipe.c 介绍

- uart 中断回调函数接收命令

1. 接收数据，并设置一个定时器
2. 当接收数据达到预设置大小或者定时器超时，认为接收一条有效数据
3. 将数据 buffer 放到数据队列，供 data_handler 线程获取并转发

```
void uart_pipe_isr(struct device *dev)
```

- uart 数据发送

```
int uart_pipe_send(const u8_t *data, int len)
{
        while (len--)  {
                uart_poll_out(uart_pipe_dev, *data++);
        }

        return 0;
}
```

system_app_ble.c 介绍

- ble 相关配置
  - 连接参数配置

```
struct bt_le_conn_param app_conn_update_cfg = {
        .interval_min = (BLE_CONN_INTERVAL_DEF),
        .interval_max = (BLE_CONN_INTERVAL_DEF),
        .latency = (0),
        .timeout = (2000),
};
```

  - 广播参数配置

```
struct bt_le_adv_param app_adv_cfg = {
        .options = BT_LE_ADV_OPT_CONNECTABLE | BT_LE_ADV_OPT_
↪ONE_TIME,
        .interval_min = BLE_ADV_INTERVAL_DEF,
        .interval_max = BLE_ADV_INTERVAL_DEF,
};
```

  - 广播内容配置

```
u8_t adv_user_data[BLE_ADV_USER_DATA_LEN] = {BLE_ADV_USER_
↪DATA_DEF};
struct bt_data app_adv_data[] = {
        BT_DATA_BYTES(BT_DATA_FLAGS, (BT_LE_AD_GENERAL | BT_
↪LE_AD_NO_BREDR)),
        BT_DATA(BT_DATA_MANUFACTURER_DATA, adv_user_data,␣
↪sizeof(adv_user_data)),
};
```

  - 广播应答包内容配置

```
u8_t adv_name_data[BLE_ADV_NAME_DATA_MAX+1] = {BLE_ADV_NAME_
↪DATA_DEF};
struct bt_data app_scan_data[] = {
```

(continues on next page)

(续上页)

```
        BT_DATA(BT_DATA_NAME_COMPLETE, adv_name_data, BLE_
↪ADV_NAME_LEN_DEF),
};
```

- ble 回调函数处理

```
static struct bt_conn_cb conn_callbacks = {
.connected = connected,
.disconnected = disconnected,
.identity_resolved = le_identity_resolved,
.security_changed = security_changed,
.le_param_updated = le_param_updated,
};
```

  - connected 蓝牙连接事件的回调
  - disconnected 蓝牙断开事件的回调
  - le_identity_resolved 蓝牙加密过程中私有地址解析的回调
  - security_changed 蓝牙加密后的回调

- ble 事件处理
  - 通过 BLE 发送数据到数据服务

```
int ble_send_data(u8_t *data, u16_t len);
```

  - 通过 BLE 发送数据到驱动服务

```
void ble_send_drv_data(u8_t *data, u16_t len);
```

app_gpio.c 介绍

主要实现 gpio 相关功能接口

app_adc.c 介绍

一个通道的 8 位 ADC 采集和一路模组电池电量的采集

- 接收并解析 adc 命令

```
void adc_event_handle(u32_t event)
{
        u32_t val;
        u8_t tmp_buf[2];
        switch (event) {
        case 1:
```

- 获取 adc 值

```
val = adc_get(ADC_ID_BATV);
```

- 将结果返回给移动设备端

```
ble_send_drv_data(tmp_buf,sizeof(tmp_buf));
```

app_pwm.c 介绍

实现两个通道的 PWM 输出波形

- 接收并解析命令

```
void pwm_cmd_handle(u8_t *cmd_buf)
{
        u8_t channel;
        u32_t duty_max_0, duty_0, duty_max_1, duty_1;

        /* check cmd paramter */
        if ((pwm_dev == NULL) ||
                        (cmd_buf[0] != 0x97) ||
                        (cmd_buf[1] > 3) ||
                        (cmd_buf[2] > 20) ||
                        (cmd_buf[3] > 100) ||
                        (cmd_buf[4] > 20) ||
                        (cmd_buf[5] > 100)
                        ) return ;

        if (cmd_buf[2] != 0)
                duty_max_0 = pwm_duty_max_tbl[cmd_buf[2] -␣
↪1];

        if (cmd_buf[3] != 0)
                duty_0 = duty_max_0 * cmd_buf[3] / 100;

        if (cmd_buf[4] != 0)
                duty_max_1 = pwm_duty_max_tbl[cmd_buf[4] -␣
↪1];

        if (cmd_buf[5] != 0)
                duty_1 = duty_max_1 * cmd_buf[5] / 100;
```

- 根据配置调用 pwm 驱动接口实现频率输出

```
/* pwm base clk source is 1Mhz */
static const u32_t pwm_duty_max_tbl[20] = {
20000, /* 50   hz */
10000, /* 100  hz */
5000,  /* 200  hz */
2500,  /* 400  hz */
2000,  /* 500  hz */
1250,  /* 800  hz */
1000,  /* 1     Khz */
500,   /* 2     Khz */
250,   /* 4     Khz */
200,   /* 5     Khz */
```

(continues on next page)

(续上页)

```
125,    /* 8     Khz */
100,    /* 10 Khz */
50,     /* 20 Khz */
40,     /* 25  Khz */
25,     /* 40  Khz */
20,     /* 50  Khz */
10,     /* 100 Khz */
8,      /* 125 Khz */
5,      /* 200 Khz */
4,      /* 250 Khz */
};

if (cmd_buf[0] == 0x97) {
channel = cmd_buf[1];
switch (channel) {
case 0:  /* open pwm channel 0 & 1 */
        pwm_pin_set_cycles(pwm_dev, 0, duty_max_0, duty_0);
        pwm_pin_set_cycles(pwm_dev, 1, duty_max_1, duty_1);
```

app_cfg.c 介绍

实现具体的 AT 指令功能

下面以软件复位为例

- 软件复位

```
/* 1. soft reset */
int at_cmd_reset(void)
{
        set_app_cfg_to_default();
        at_cmd_response("OK+STRS\r\n");
        printk("reset\n");
        k_sleep(1000);
        sys_pm_reboot(REBOOT_TYPE_GOTO_APP);
        return 0;
}
```

其中，

set_app_cfg_to_default() 将模组的配置参数恢复成默认值

at_cmd_response(“OK+STRSrn”); 应答 MCU AT 指令

sys_pm_reboot(REBOOT_TYPE_GOTO_APP); 系统重启

version.c 介绍

应用版本维护

```
char sys_version[] = "2.0.00.18101609";
```

5.3.4 app_framework 目录代码介绍

- msg_manager 消息的发送和接收
  - 同步消息发送
  - 异常消息发送
  - 消息 buffer 的配置

```
NET_BUF_POOL_DEFINE(msg_buffer_pool_async, 8, sizeof(struct␣
↪app_msg), 4, NULL);
NET_BUF_POOL_DEFINE(msg_buffer_pool_sync, 8, sizeof(struct␣
↪app_msg), 4, NULL);
```

注解:

- 同步消息个数，建议根据应用需求配置足够多，但同时注意避免浪费
- 异步消息个数，建议保证应用的正常需求个数，如按键消息，如果主线程处理不过来，从设计上允许丢按键，保证系统不卡住。

5.3.5 profile 目录代码介绍

WP profile 实现了数据透传服务和数据驱动服务，用于跟移动设备终端交互

5.3.6 bt_host 目录代码介绍

zephyr bt_host 相关配置及 patch 文件详细的配置，请参考 <<ZS110A-方案配置指南 >>

5.3.7 drivers 目录代码介绍

驱动实现相关源码，详情请参考 <<ZS110A 驱动用户指南 >>

5.3.8 kernel 目录代码介绍

kernel 部分启动代码及 kernel 配置文件详细的配置，请参考 <<ZS110A-方案配置指南 >>

5.3.9 ota 工程目录介绍

使用 OTA 进行升级的源码，详情请参考 <<ZS110A-OTA 指南 >>

5.4 rf_test 工程简介

5.4.1 频偏测试

产品进入频偏测试模式后自动在指定频点 (默认为 19 通道，频点为 2440MHz) 上发送 single tone。

将产品靠近频谱仪的接收天线，从频谱仪上得到实际频率和频偏值后可以判断产品的频偏是否在合理范围内。

5.4.2 如何修改频点

下面代码中 0x13(十六进制) 表示 19(十进制) 通道。修改该值就可以改变信号频点。

```
u8_t BUF_CONTINUOUS_TX[] = {0x0F, 0xFE, 0x03, 0x13, 0x0, 0x00};
```

CHAPTER 6

BLE 透传模组功能测试简介

6.1 基本功能测试

- AT 指令测试
- 移动端 ADC、PWM 操作指令测试
- 透传功能测试

6.2 软硬件资源准备

6.2.1 硬件资源准备

- PC
- ATB110X EVB 开发板
- android 设备

6.2.2 软件资源准备

- 串口助手
- ble_module firmware
- Actions 透传 APK

6.2.3 工具安装

- 在 pc 上安装串口助手
- EVB 更新固件

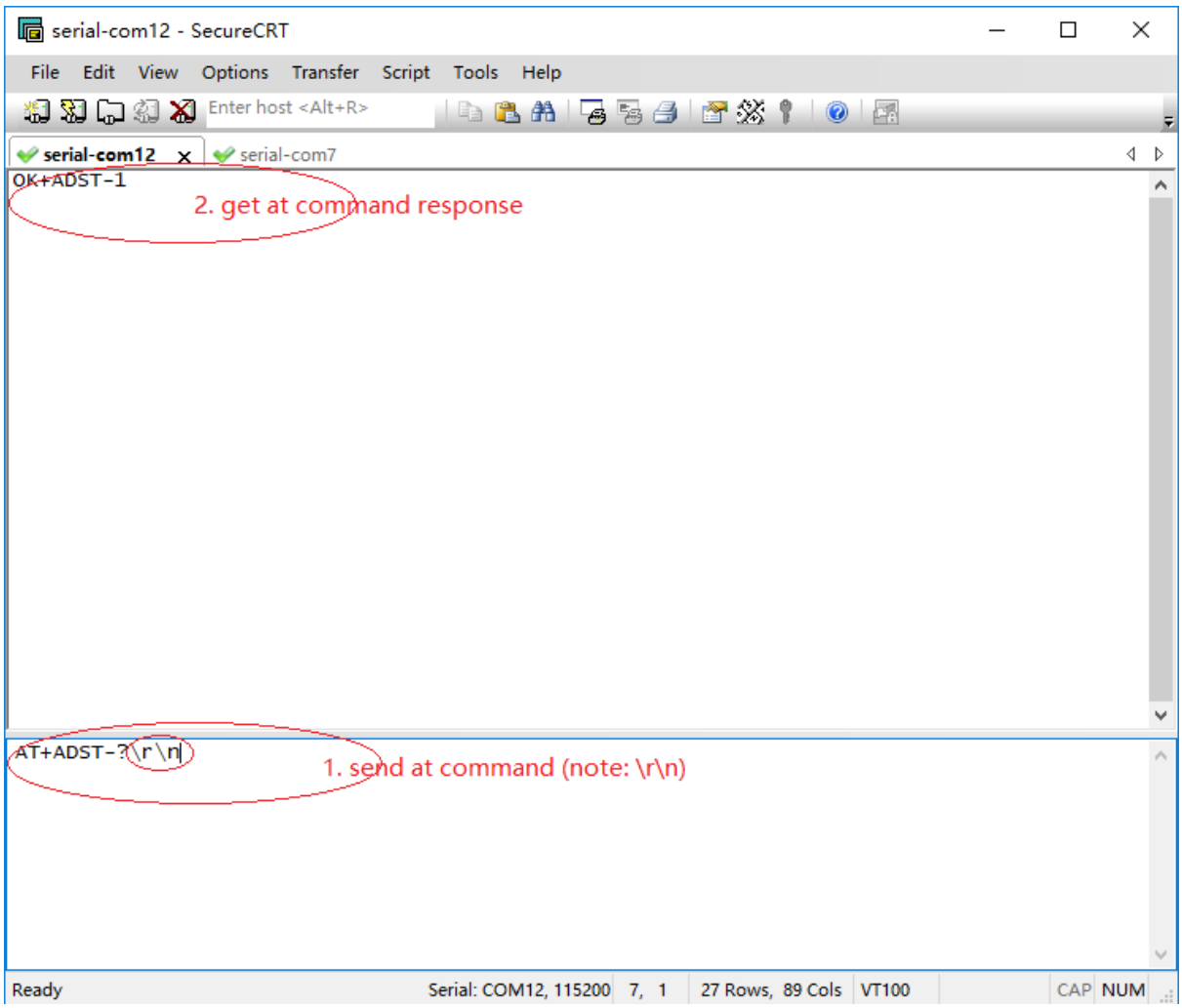
> **注解:** 为了方便 PC 串口演示功能，建议固件关掉 EVB 固件的休眠

- 在 android 设备上安装 Actions 最新 bletest APK

> **注解:** Actions APK 路径: \scripts\support\actions\utils\android_apk\

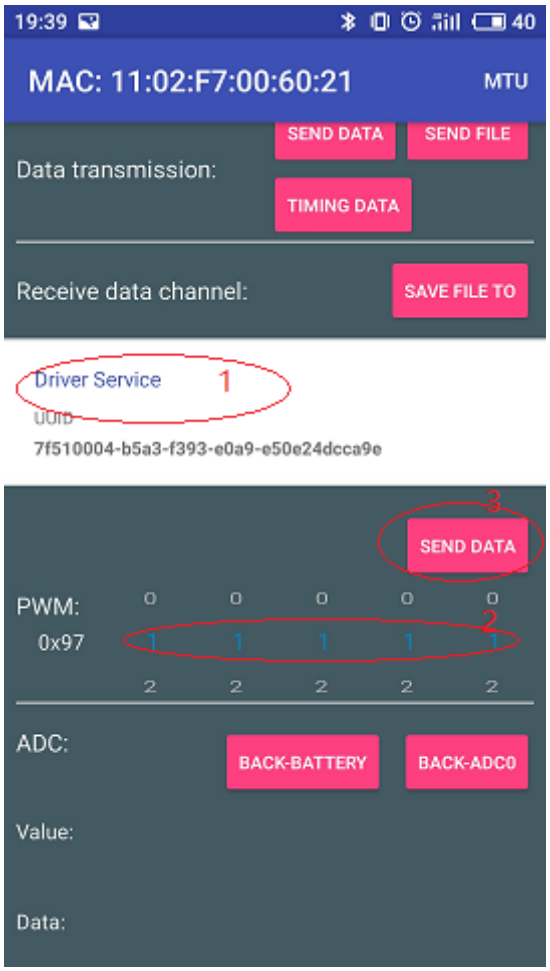
6.3 AT 指令测试

- 将 PC 串口和 evb 连接, 然后将 pc 串口的波特率跟 evb 配置成一致
- 在 pc 串口中输入 AT 指令，然后得到相应的指令应答

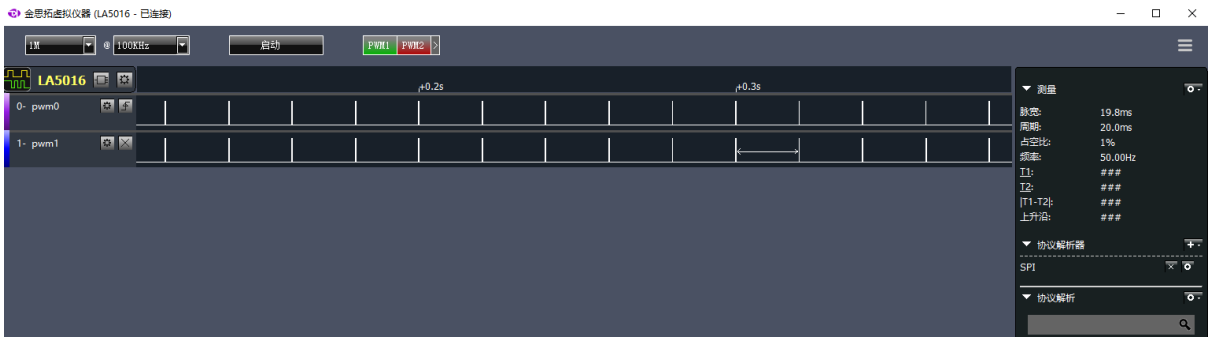


6.4 移动端 ADC、PWM 操作指令测试

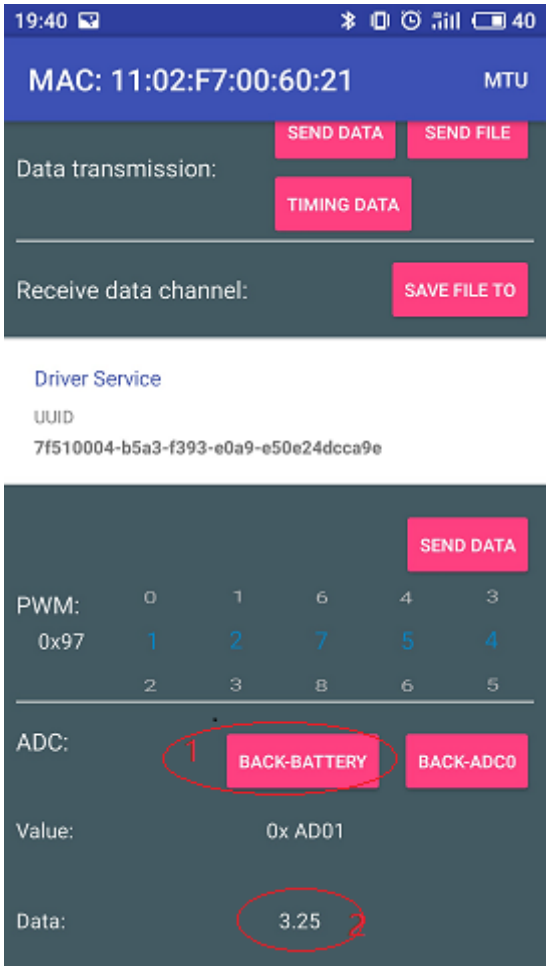
- evb 上电，模组上电即是广播状态
- 打开 actions bletest APK，然后连接模组
- 点击开 driver service，然后设置 pwm 命令，点击“SEND DATA”，发送 pwm 命令



- 可以使用 KingstVIS 抓取 evb pwm 输出 gpio 口的波形，如下图所示



- 点击开 driver service，然后点击“BACK-BATTERY”，就会返回电量值，如下图所示



6.5 透传功能测试

- 移动设备通过模组将数据透传到 PC 串口助手

- 在 Android 手机端安装上 Actions 的 bletest APK，启动 APK，然后点击“Start Scan”并连接上。
- 在 Actions Transmission Service 上，点击“SEND FILE”，然后选择一个文件，点击“START SENDING”
- 这样，数据就不停的从手机端通过 BLE 传送到设备端，并透传到 pc 串口助手。

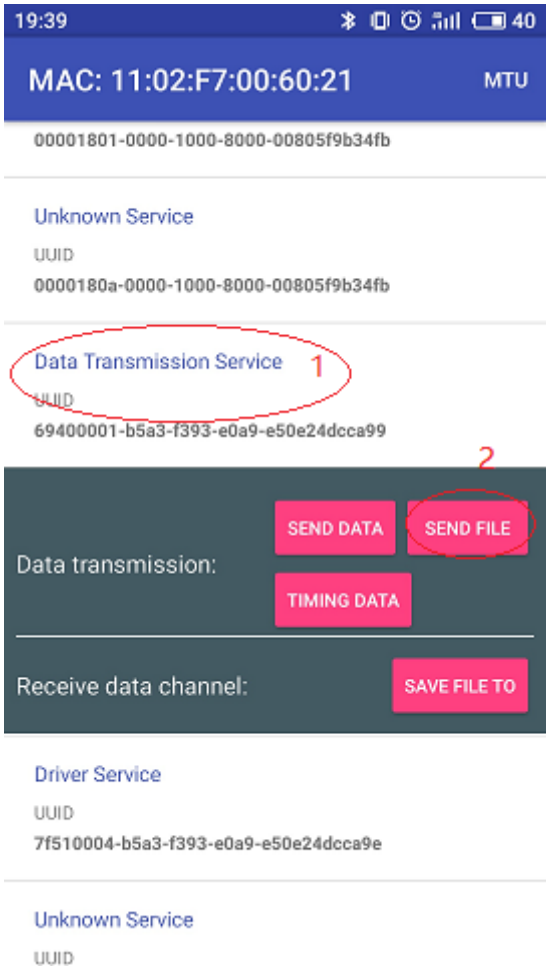


图 6.1: 数据发送操作示意图

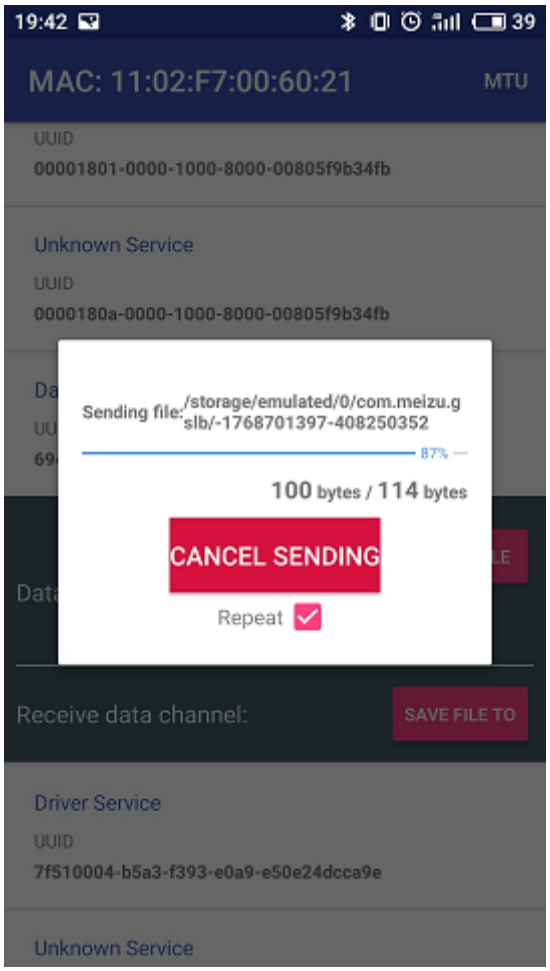


图 6.2: 数据发送示意图

- PC 串口助手通过模组将数据透传到移动设备
  - 在 Android 手机端安装上 Actions 的 bletest APK，启动 APK，然后点击“Start Scan”并连接上。
  - 在 Data Transmission Service 上，点击“SAVE FILE TO”，然后选择一个文件用于保存数据
  - 然后 PC 串口发送数据
  - 这样移动设备端就能收到 pc 串口助手发送的数据，并保存在文件中

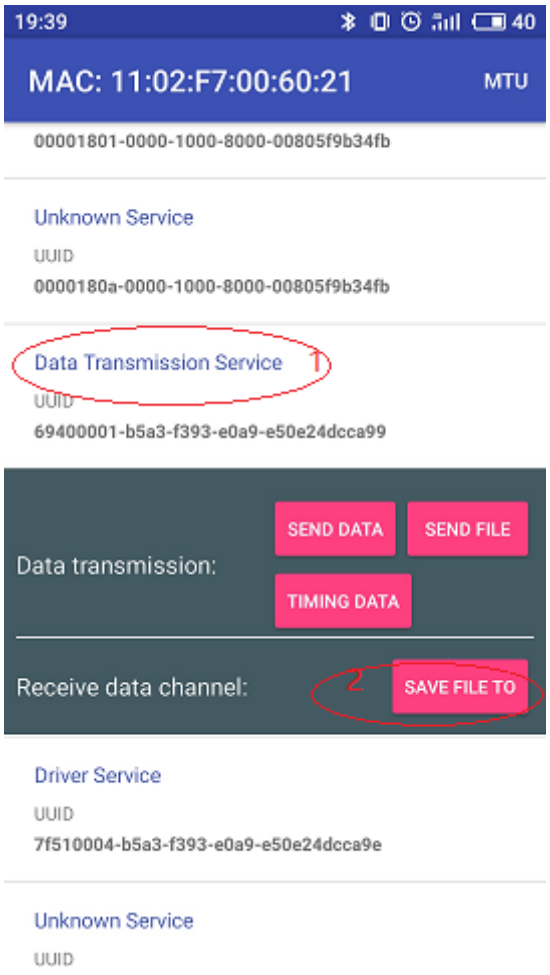


图 6.3: 数据接收操作示意图

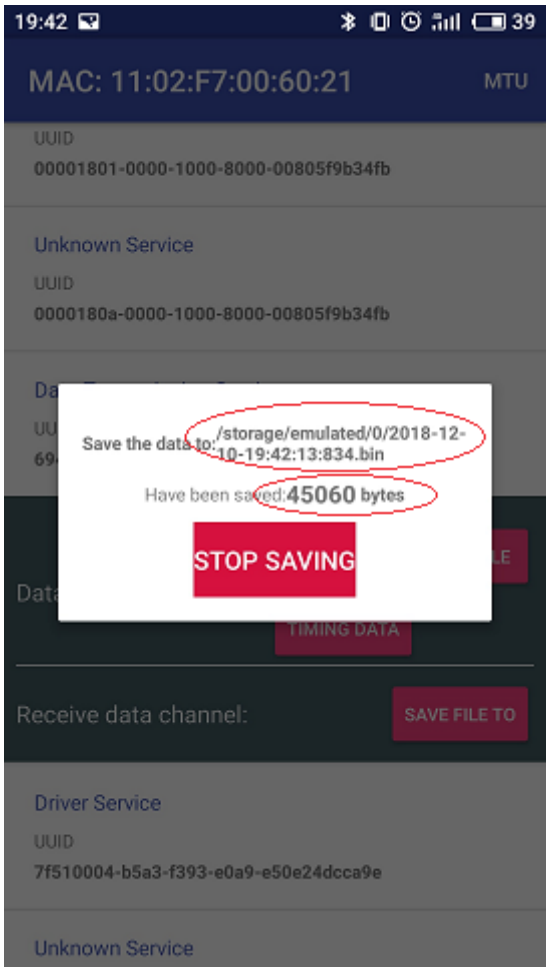

19:42
39
MAC: 11:02:F7:00:60:21
MTU
UUID
00001801-0000-1000-8000-00805f9b34fb
Unknown Service
UUID
0000180a-0000-1000-8000-00805f9b34fb
Save the data to:/storage/emulated/0/2018-12-10-19:42:13:834.bin
Have been saved:45060 bytes
STOP SAVING
TIMING DATA
Receive data channel:
SAVE FILE TO
Driver Service
UUID
7f510004-b5a3-f393-e0a9-e50e24dcca9e
Unknown Service

图 6.4: 数据接收示意图

List of Figures

List of Tables