

# NF3303 蓝牙模块 **BLE** **Server** 功能使用说明

## About the Document

本文档适用于 MDM9628 和 MDM9X07 平台

## History

Revision	Date	Author	Description
1.0	2017-12-14	Quinn	Initial

## 目录

<b>NF3303 蓝牙模块 BLE Server 功能使用说明</b>	0
About the Document	1
1. 蓝牙模块硬件接口简述	3
2. 蓝牙模块 SDK 资源使用	4
2.1 UART driver	4
2.2 芯片固件、协议栈	8
2.3 编译执行	8
2.4 调试开发	8
3. 代码使用指导	10
3.1 修改复位引脚	10
3.2 修改广播信息	10
3.3 修改 Service、特性和描述符配置参数	10
3.4 Debug 功能使用	12
4. 测试方法	13

## 1. 蓝牙模块硬件接口简述

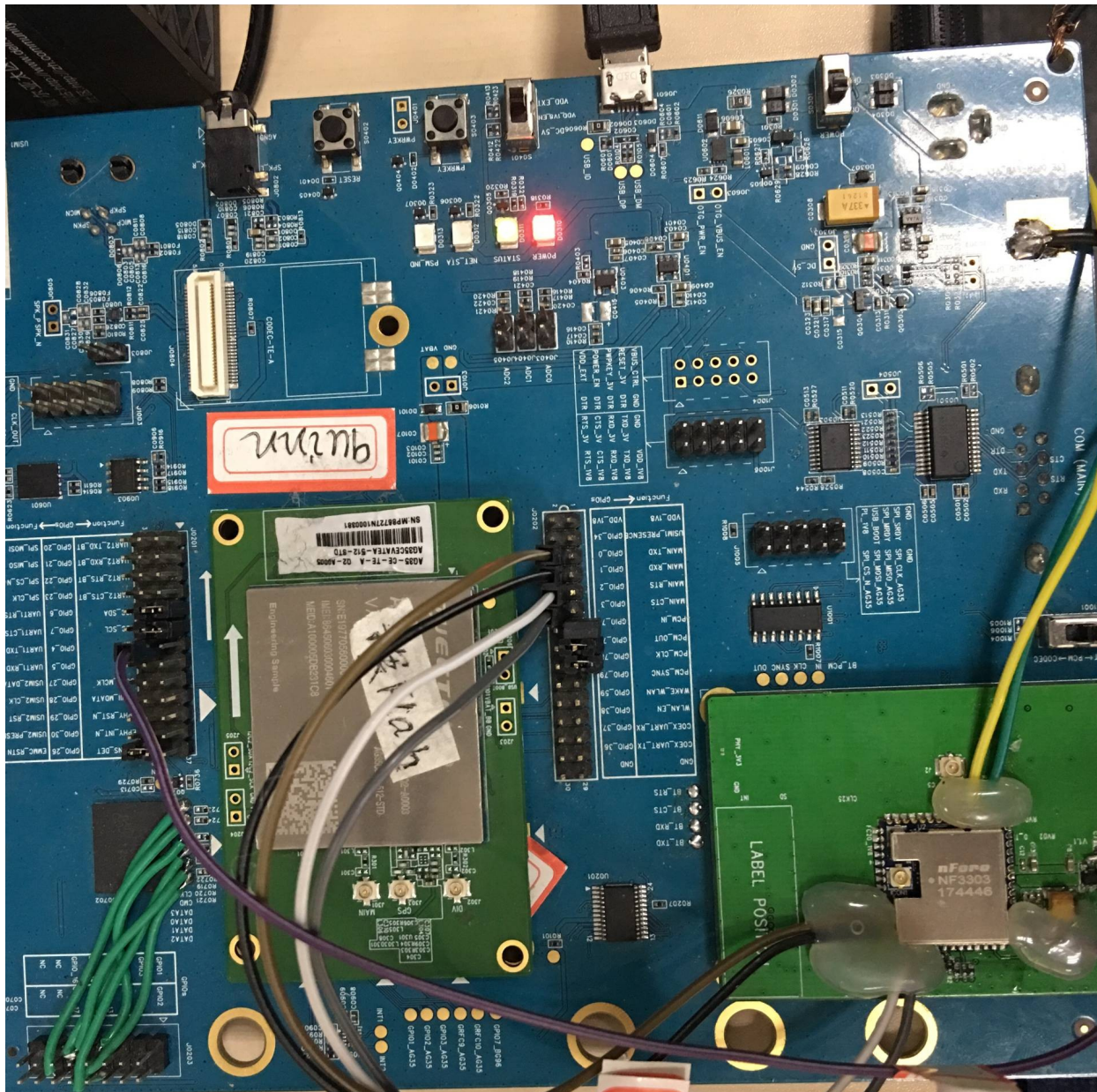
蓝牙模块与 EC20 模组硬件连接有如下需求：

- 1、蓝牙模块与 EC20 模组通过有流控功能的 UART 外设接口通信，传输 HCI Command 等。
- 2、EC20 模组使用 GPIO 来拉低、拉高蓝牙模块的复位引脚，复位模块。

当前硬件连接 PIN 脚对应表格如下：

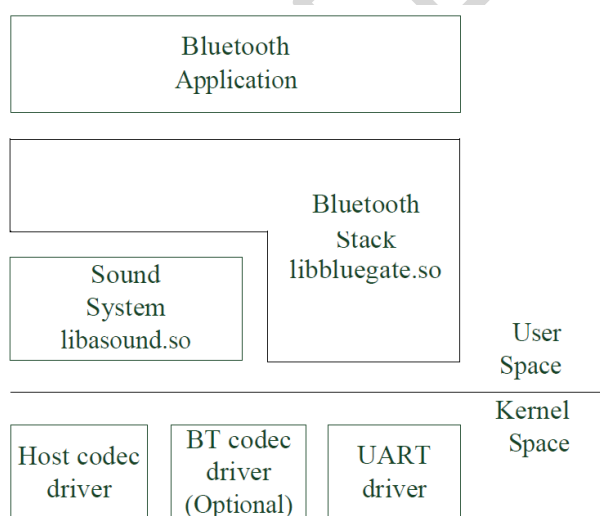
EC20 模组 PIN 脚	EC20 模组使用芯片 GPIO	蓝牙模块
37	UART TXD BLSP6/GPIO 20	HCI RX BT
38	UART RXD BLSP6/GPIO 21	HCI TX BT
39	UART RTS BLSP6/GPIO 22	HCI RTS BT
40	UART CTS BLSP6/GPIO 23	HCI CTS BT
139	GPIO 2	BT EN

- 3、当前使用飞线的方法连接 NF3303 模块和 OpenEVB 开发板，连接图如下：



## 2. 蓝牙模块 SDK 资源使用

蓝牙模块 SDK 软件架构图如下：



### 2.1 UART driver

通过架构图可知，蓝牙模块与 EC20 模组通过 UART 外设接口通信。因此，模组需要确保 UART 驱动工作正常，且流控功能正常。

1、修改设备树文件，添加对 EC20 模组与蓝牙模块通信使用的 UART 的支持

a) 修改路径为 “apps\_proc/kernel/arch/arm/boot/dts/qcom” 的设备树文件 “mdm9607-pinctrl.dtsi”，修改下图中的设备树信息：

```

blsp1_uart2_active: blsp1_uart2_active {
    mux {
        pins = "gpio4", "gpio5";
        function = "blsp_uart2";
    };
    config {
        pins = "gpio4", "gpio5";
        drive-strength = <2>;
        bias-disable;
    };
};

blsp1_uart6_sleep: blsp1_uart6_sleep {
    mux {
        pins = "gpio20", "gpio21";
        function = "blsp_uart6";
    };
    config {
        pins = "gpio20", "gpio21";
        drive-strength = <2>;
        bias-pull-down;
    };
};

blsp1_uart3_active: blsp1_uart3_active {
    mux {
        pins = "gpio0", "gpio1", "gpio2", "gpio3";
    };
};

```

修改前

修改后的设备树信息如下图：

```

blsp1_uart6_active: blsp1_uart6_active {
    mux {
        pins = "gpio20", "gpio21", "gpio22", "gpio23";
        function = "blsp_uart6";
    };
    config {
        pins = "gpio20", "gpio21", "gpio22", "gpio23";
        drive-strength = <2>;
        bias-disable;
    };
};

blsp1_uart6_sleep: blsp1_uart6_sleep {
    mux {
        pins = "gpio20", "gpio21", "gpio22", "gpio23";
        function = "gpio";
    };
    config {
        pins = "gpio20", "gpio21", "gpio22", "gpio23";
        drive-strength = <2>;
        bias-disable;
        output-low; //output low when sleep
    };
};

```

修改后

修改后的设备树文本信息如下：

```

blsp1_uart6_active: blsp1_uart6_active {
    mux {
        pins = "gpio20", "gpio21", "gpio22", "gpio23";
        function = "blsp_uart6";
    };

    config {
        pins = "gpio20", "gpio21", "gpio22", "gpio23";
        drive-strength = <2>;
        bias-disable;
    };
};

blsp1_uart6_sleep: blsp1_uart6_sleep {
    mux {
        pins = "gpio20", "gpio21", "gpio22", "gpio23";
    };
};

```



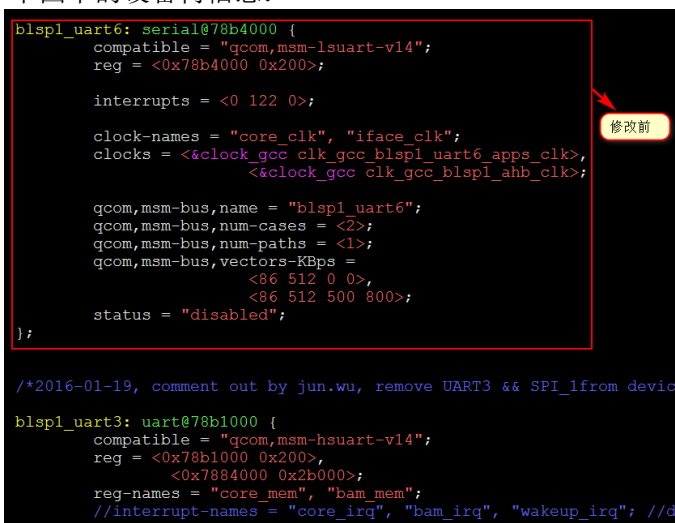
```

        function = "gpio";
    };

    config {
        pins = "gpio20", "gpio21", "gpio22", "gpio23";
        drive-strength = <2>;
        bias-disable;
        output-low;    //output low when sleep
    };
};

```

b) 修改路径为 “apps\_proc/kernel/arch/arm/boot/dts/qcom”的设备树文件 “mdm9607.dtsi”，修改下图中的设备树信息：



```

blsp1_uart6: serial@78b4000 {
    compatible = "qcom,msm-lsuart-v14";
    reg = <0x78b4000 0x200>;

    interrupts = <0 122 0>;

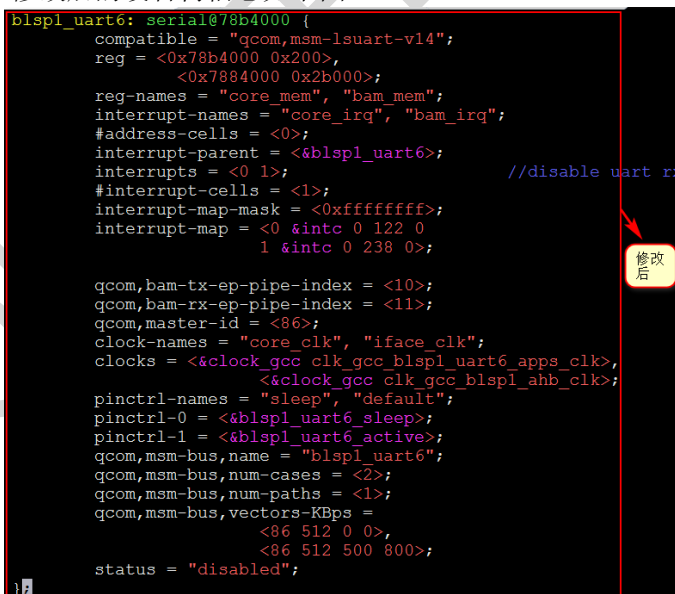
    clock-names = "core_clk", "iface_clk";
    clocks = <&clock_gcc clk_gcc_blsp1_uart6_apps_clk>,
            <&clock_gcc clk_gcc_blsp1_ahb_clk>;

    qcom,msm-bus,name = "blsp1_uart6";
    qcom,msm-bus,num-cases = <2>;
    qcom,msm-bus,num-paths = <1>;
    qcom,msm-bus,vectors-KBps =
        <86 512 0 0>,
        <86 512 500 800>;
    status = "disabled";
};

/*2016-01-19, comment out by jun.wu, remove UART3 && SPI_1from device
blsp1_uart3: uart@78b1000 {
    compatible = "qcom,msm-hsuart-v14";
    reg = <0x78b1000 0x200>;
        <0x7884000 0x2b000>;
    reg-names = "core_mem", "bam_mem";
    //interrupt-names = "core_irq", "bam_irq", "wakeup_irq"; //d
    interrupt-names = "core_irq", "bam_irq", "wakeup_irq";
}

```

修改后的设备树信息如下图：



```

blsp1_uart6: serial@78b4000 {
    compatible = "qcom,msm-lsuart-v14";
    reg = <0x78b4000 0x200>,
        <0x7884000 0x2b000>;
    reg-names = "core_mem", "bam_mem";
    interrupt-names = "core_irq", "bam_irq";
    #address-cells = <0>;
    interrupt-parent = <&blsp1_uart6>;
    interrupts = <0 1>;    //disable uart rx
    #interrupt-cells = <1>;
    interrupt-map-mask = <0xffffffff>;
    interrupt-map = <0 &intc 0 122 0
        1 &intc 0 238 0>;

    qcom,bam-tx-ep-pipe-index = <10>;
    qcom,bam-rx-ep-pipe-index = <11>;
    qcom,master-id = <86>;
    clock-names = "core_clk", "iface_clk";
    clocks = <&clock_gcc clk_gcc_blsp1_uart6_apps_clk>,
            <&clock_gcc clk_gcc_blsp1_ahb_clk>;
    pinctrl-names = "sleep", "default";
    pinctrl-0 = <&blsp1_uart6_sleep>;
    pinctrl-1 = <&blsp1_uart6_active>;
    qcom,msm-bus,name = "blsp1_uart6";
    qcom,msm-bus,num-cases = <2>;
    qcom,msm-bus,num-paths = <1>;
    qcom,msm-bus,vectors-KBps =
        <86 512 0 0>,
        <86 512 500 800>;
    status = "disabled";
};

```

修改后的设备树文本信息如下：

```
blsp1_uart6: serial@78b4000 {
```



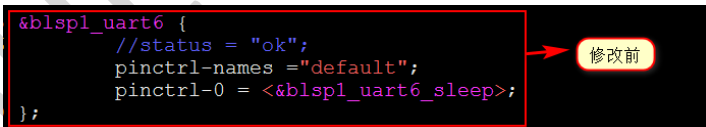
```

compatible = "qcom,msm-lsuart-v14";
reg = <0x78b4000 0x200>,
      <0x7884000 0x2b000>;
reg-names = "core_mem", "bam_mem";
interrupt-names = "core_irq", "bam_irq";
#address-cells = <0>;
interrupt-parent = <&blsp1_uart6>;
interrupts = <0 1>;          //disable uart rx wakeup source
#interrupt-cells = <1>;
interrupt-map-mask = <0xffffffff>;
interrupt-map = <0 &intc 0 122 0
                1 &intc 0 238 0>;

qcom,bam-tx-ep-pipe-index = <10>;
qcom,bam-rx-ep-pipe-index = <11>;
qcom,master-id = <86>;
clock-names = "core_clk", "iface_clk";
clocks = <&clock_gcc clk_gcc_blsp1_uart6_apps_clk>,
        <&clock_gcc clk_gcc_blsp1_ahb_clk>;
pinctrl-names = "sleep", "default";
pinctrl-0 = <&blsp1_uart6_sleep>;
pinctrl-1 = <&blsp1_uart6_active>;
qcom,msm-bus,name = "blsp1_uart6";
qcom,msm-bus,num-cases = <2>;
qcom,msm-bus,num-paths = <1>;
qcom,msm-bus,vectors-KBps =
    <86 512 0 0>,
    <86 512 500 800>;
status = "disabled";
};

```

c) 修改路径为 “apps\_proc/kernel/arch/arm/boot/dts/qcom” 的设备树文件 “mdm9607-mtp.dtsi”，修改前设备树信息如下图：



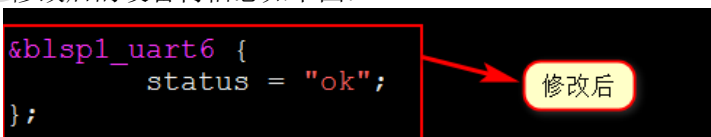
```

&blsp1_uart6 {
    //status = "ok";
    pinctrl-names = "default";
    pinctrl-0 = <&blsp1_uart6_sleep>;
};

```

修改前

修改后的设备树信息如下图：



```

&blsp1_uart6 {
    status = "ok";
};

```

修改后

修改后的设备树文本信息如下：

```

&blsp1_uart6 {
    status = "ok";
};

```

```
};
```

## 2、添加串口配置文件

a)在路径 “/etc/bluetooth/”下，创建文件 “bluegate\_hw.conf”。参考命令如下：

```
$ mkdir /etc/bluetooth /*若 etc 目录下不存在 bluetooth 目录，则新建*/
```

```
$ touch /etc/bluetooth/bluegate_hw.conf /*创建配置文件*/
```

b)在新建的 bluegate\_hw.conf 文件中添加如下图所示信息：

```
# UART device port where Bluetooth controller is attached
UartPort = /dev/ttyHSL2

# Firmware patch file location
FwPatchFilePath = /etc/bluetooth/
FwPatchFileName = BCM4339_003.001.009.0119.0000.hcd
```

修改后的配置文件文本信息如下：

```
# UART device port where Bluetooth controller is attached
```

```
UartPort = /dev/ttyHSL2
```

```
# Firmware patch file location
```

```
FwPatchFilePath = /etc/bluetooth/
```

```
FwPatchFileName = BCM4339_003.001.009.0119.0000.hcd
```

注意：UartPort 要与实际所用串口的设备信息一致。

## 2.2 芯片固件、协议栈

芯片固件和协议栈库文件将随 SDK 一块发布，路径为 “ql-ol-sdk/ql-ol-extsdk/lib”。

### 1、芯片固件

在模组与蓝牙模块的串口通信建立成功后，会通过串口将芯片固件(\*.hcd 文件)下载至蓝牙芯片。因此需要将芯片固件预先放置在文件系统里，可以使用 adb 工具推送到 EC20 模组文件系统里，参考命令如下：

```
$ adb push BCM4339_003.001.009.0119.0000.hcd /etc/bluetooth/
```

### 2、协议栈

协议栈（libbluegate.so）是以动态库的形式提供。因此，需要将该文件复制到 “/lib”目录下。参考命令如下：

```
$ adb push libbluegate.so /lib/
```

## 2.3 编译执行

解压缩 SDK 压缩包，进入 ql-ol-sdk 文件夹下，编译例程。参考命令如下：

```
$ source ql-ol-crosstool/ql-ol-crosstool-env-init
```

```
$ cd ql-ol-extsdk/example/bt/
```

```
$ make clean
```

```
$ make
```

在 SDK 中路径为 “ql-ol-sdk/ql-ol-extsdk/example/bt” 文件夹下生成名为 “example\_nf3303\_ble\_server” 的可执行程序。生成的可执行程序可以通过 adb 推送到 EC20 模组文件系统。ADB 方式的参考命令如下：

```
$ adb push example_nf3303_ble_server /data/
```

```
$ adb shell chmod a+x /data/example_nf3303_ble_server
```

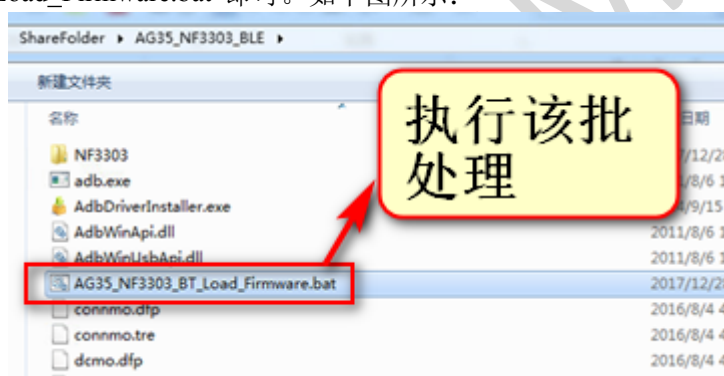
进入 Linux Shell 终端，输入以下命令，启动 BLE Server：

```
$ cd /data/
```

```
$ ./example_nf3303_ble_server
```

## 2.4 调试开发

为了便于客户调试开发，以免手动去推送芯片固件、协议栈等至模组的文件系统。提供了一个集成了配置文件、芯片固件和协议栈库文件的命名为 “AG35\_NF3303\_BLE.zip” 压缩包。客户修改、编译后，只需将生成的可执行文件 “example\_nf3303\_ble\_server” 放入路径 “AG35\_NF3303\_BLE/NF3303/data” 的目录下。然后，保证 ADB 通讯正常，到路径 “AG35\_NF3303\_BLE/”，执行批处理文件 “AG35\_NF3303\_BT\_Load\_Firmware.bat” 即可。如下图所示：



## 3. 代码使用指导

### 3.1 修改复位引脚

在调用 NFBT\_NFBT\_Enabl 之前，需要将 BT\_EN 引脚拉低 100ms 后再拉高，对模块进行复位重置。如果平台不同，需要根据实际修改复位引脚。请在路径为 “ql-ol-sdk/ql-ol-extsdk/example/bt” 的 C 文件 “ql\_nf3303\_ble\_server.c” 的 main 函数的起始处加上复位代码。代码如下面截图所示：

```
/*Reset BT Module*/  
if(RES_OK != (Ql_GPIO_Init(m_GpioPin, PINDIRECTION_OUT, PINLEVEL_HIGH,  
    PINPULLSEL_DISABLE)))  
    ERR("GPIO init Failed\n");  
Ql_GPIO_SetLevel(m_GpioPin, PINLEVEL_LOW);  
usleep(200000);  
Ql_GPIO_SetLevel(m_GpioPin, PINLEVEL_HIGH);
```

### 3.2 修改广播信息

广播信息通过修改路径为 “ql-ol-sdk/ql-ol-extsdk/example/bt” 的 C 文件 “ql\_nf3303\_ble\_server.c” 中函数 “NFBT\_GATT\_SetAdvertisingData” 来修改。可修改参数有广播设备名、发射功率、服务 UUID 和广播间隔。下面介绍各修改项修改方法：

#### 1、修改广播信息中的设备名

通过修改路径为 “ql-ol-sdk/ql-ol-extsdk/example/bt” 的 C 文件 “ql\_nf3303\_ble\_server.c” 中宏定义 “LOCAL\_NAME” 改变广播信息中的设备名。

#### 2、修改广播信息中的发射功率

目前模块厂商 SDK 不支持修改发射功率。

#### 3、修改广播信息中的 Service UUID

通过修改路径为“ql-ol-sdk/ql-ol-extsdk/example/bt”的C文件“ql\_nf3303\_ble\_server.c”中main函数中的局部变量“gatt\_base\_uuid”改变广播信息中的服务信息。目前模块厂商SDK仅支持添加一个Service UUID。

#### 4、修改广播信息中的广播时间间隔

目前可修改时间间隔值如下，修改参数为对应的宏定义即可。参数如下：

```
/* GATT Advertising Data definition */
#define GATT_ADV_LOW      0 //1 s
#define GATT_ADV_NORMAL   1 //100 ms
#define GATT_ADV_HIGH     2 //30 ms
```

### 3.3 修改 Service、特性和描述符配置参数

#### 1、配置最大 Service 数目参数

最大 Service 数目参数通过修改路径为“ql-ol-sdk/ql-ol-extsdk/include”的头文件“ql\_nf3303\_bt\_callback.h”中宏定义“BLE\_TOTAL\_SERVICE\_NUM”来定义。可以根据需要增大、减小该宏定义，默认为8。

#### 2、配置 Service 允许最大 characteristic 和 Descriptor 数目参数

最大 characteristic 和 Descriptor 总和数目参数通过修改路径为“ql-ol-sdk/ql-ol-extsdk/include”的头文件“ql\_nf3303\_bt\_callback.h”中宏定义“BT\_MAX\_ATTR\_NUM”来定义。假设有多个 Service，其中一个 Service 的 characteristic 和 Descriptor 总和为8，其他 Service characteristic 和 Descriptor 总和小于8，该值应该配为8。

#### 3、修改 Service、特性和描述符配置参数

以下代码在路径为“ql-ol-sdk/ql-ol-extsdk/example/bt”的C文件“ql\_nf3303\_ble\_server.c”中。以 Service 0 的 characteristic 0 为例说明如何修改。请根据下面中的中文解释修改，未中文解释的参数不用修改，需要修改的地方已用红色字体标识。

```
/*Service 0 Definition*/
/*请在下面代码行填写 Service UUID */
ql_gatt_demo.ql_gatt_service_s[0].service_uuid = QUECTEL_SERVICE0_UUID;
/*请在下面代码行填写该 Service 的 characteristic 和 Descriptor 总和数目*/
ql_gatt_demo.ql_gatt_service_s[0].total_attr_num= BLE_SERVICE0_ATTR_NUM;
ql_gatt_demo.ql_gatt_service_s[0].is_srv_valid = 0x1;
ql_gatt_demo.ql_gatt_service_s[0].status = SRV_REG;

/*Service 0 characteristic 0 Definition*/
/*请在下面代码行填写 ATTRTYPE_CHARACTERISTIC 或 ATTRTYPE_DESCRIPTOR */
ql_gatt_demo.ql_gatt_service_s[0].attributes[0].type = ATTRTYPE_CHARACTERISTIC;
/*请在下面代码行填写 characteristic 或 Descriptor 的 UUID */
ql_gatt_demo.ql_gatt_service_s[0].attributes[0].uuid = DEVINFO_MANUFACTURER_NAME_UUID;
/*请在下面代码行填写 characteristic 或 Descriptor 的 Permission */
ql_gatt_demo.ql_gatt_service_s[0].attributes[0].permission = GATT_PERMIT_READ;
/*请在下面代码行填写 characteristic 或 Descriptor 的 Property*/
ql_gatt_demo.ql_gatt_service_s[0].attributes[0].property = GATT_PROP_READ|GATT_PROP_NOTIFY;
```

```

ql_gatt_demo.ql_gatt_service_s[0].attributes[0].is_attr_valid = 0x1;
/*请在下面代码行填写 characteristic 或 Descriptor 的数据字节数目*/
ql_gatt_demo.ql_gatt_service_s[0].attributes[0].data_bytes = 100;
tmp_len = ql_gatt_demo.ql_gatt_service_s[0].attributes[0].data_bytes;
ql_gatt_demo.ql_gatt_service_s[0].attributes[0].data = (char *)calloc(tmp_len, sizeof(char));
if(NULL == ql_gatt_demo.ql_gatt_service_s[0].attributes[0].data)
ERR("Calloc Failed:\n");
/*请在下面代码行为 characteristic 或 Descriptor 填充初始数据*/
tmp_len = sprintf(ql_gatt_demo.ql_gatt_service_s[0].attributes[0].data, "%s", "Quectel NF3303 BT
Module");
if (tmp_len < 0) {
ERR("Sprintf Failed:\n");
}

```

### 3.4 Debug 功能使用

为了便于 Debug 版本和 Release 版本的区分，实现了一个小型的日志输出系统。日志输出分为三个优先级，定义为 ERR、INFO、DEBUG 三个日志级别。优先级为 ERR>INFO>DEBUG。ERR 级别只输出错误信息，INFO 级别输出一般信息，DEBUG 级别输出所有信息。

#### 1、修改输出级别

用户在每个源码 C 文件中，通过宏 “DEBUG\_SET\_LEVEL”来定义日志输出级别。在路径为 “ql-ol-sdk/ql-ol-extsdk/example/bt”的 C 文件 “ql\_nf3303\_ble\_server.c”中的应用示例如下：

```

/*
 * define the debug level of this file,
 * please see 'debug.h' for detail info
 */
DEBUG_SET_LEVEL(DEBUG_LEVEL_INFO);

```

#### 2、输出信息使用函数

提供了 ERR()、INFO()、DEBUG()三个函数分别用于 ERR、INFO 和 DEBUG 三个日志级别。在路径为 “ql-ol-sdk/ql-ol-extsdk/example/bt”的 C 文件 “ql\_nf3303\_ble\_server.c”中的应用示例如下：

```

while(retry_times--) {
    if(cmd_ret == NFBT_STATUS_SUCCESS) {
        INFO("Disable BT stack success\n");
        break;
    }
    else
        sleep(timeout);
    if(0 == retry_times) {
        ERR( "Disable BT stack fail\n" );
    }
}

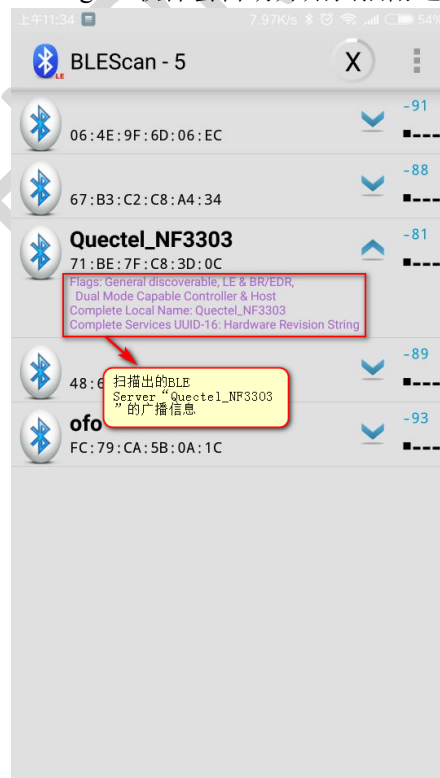
```

## 4. 测试方法

- 1、 安卓平台下，请到应用商店下载名为“BLE Deng”测试软件，图标如下：



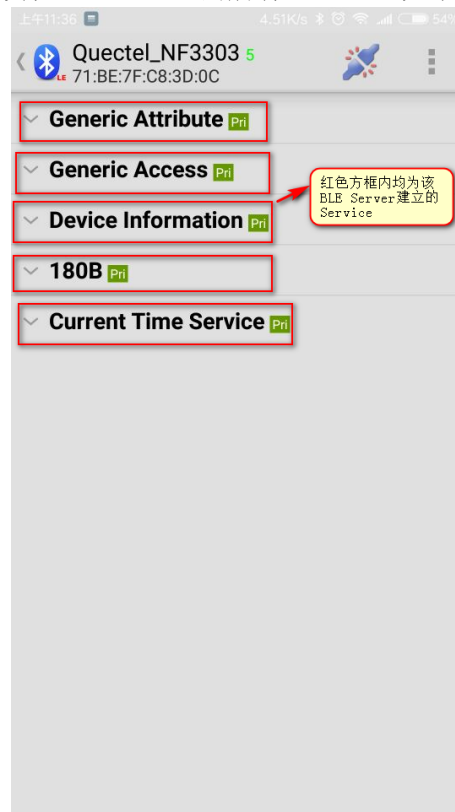
- 2、 开启手机蓝牙功能，打开“BLE Deng”，软件会自动开始扫描附近的 BLE 设备。如下图所示：



通过上图可知，手机已经扫描到 BLE Server(Quectel\_NF3303)，点击“Quectel\_NF3303”连接 BLE Server。



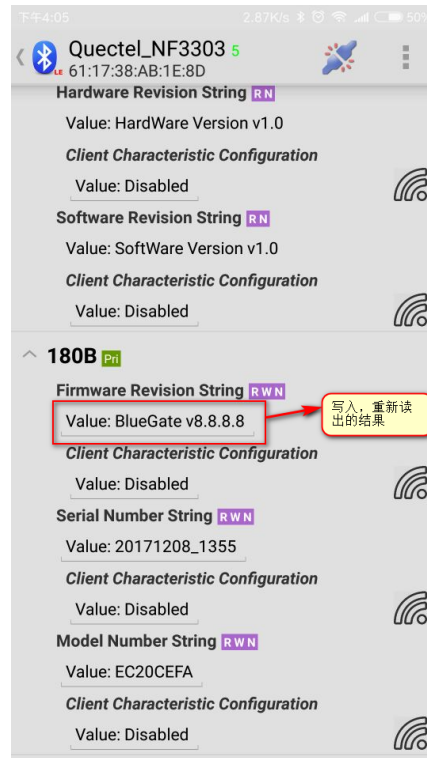
3、连接建立后，BLE Deng 会扫描发现 BLE Server 的所有 Service。如下图所示：



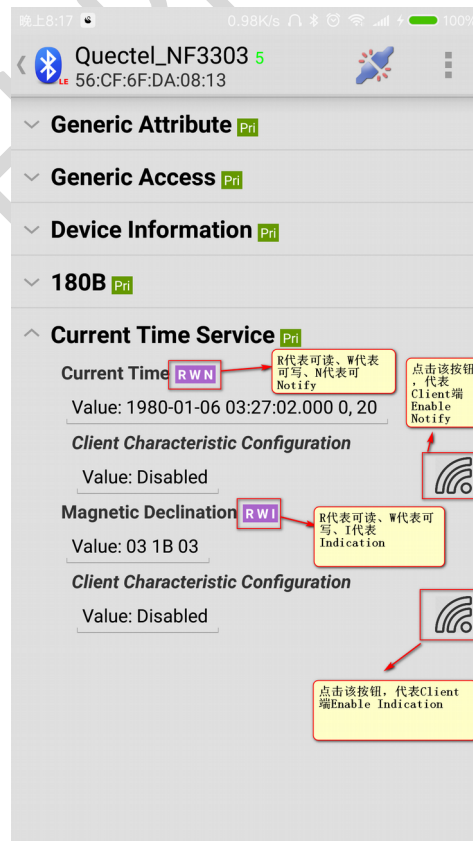
4、查看 Service 包含的 characteristic。选中一个 Service，点击查看该 Service 包含的 characteristic。此处选择 “Device Information”Service。如下图所示：



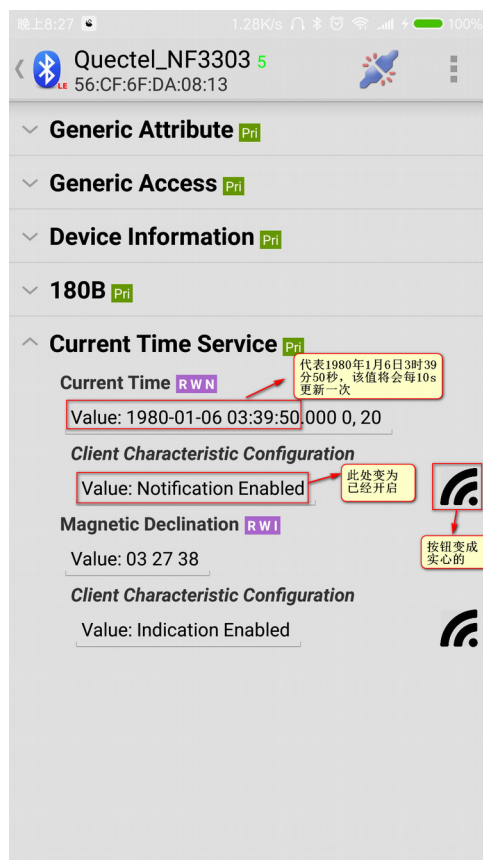
- 5、读写测试。如下图所示，Characteristic FirmwareRevisionString 能够被读写。将值修改为 BlueGate v8.8.8.8，效果如下图所示：



- 6、Notification 功能测试。选中一个“Current Time Service”，点击查看该 Service 包含的 characteristic。如下图所示：

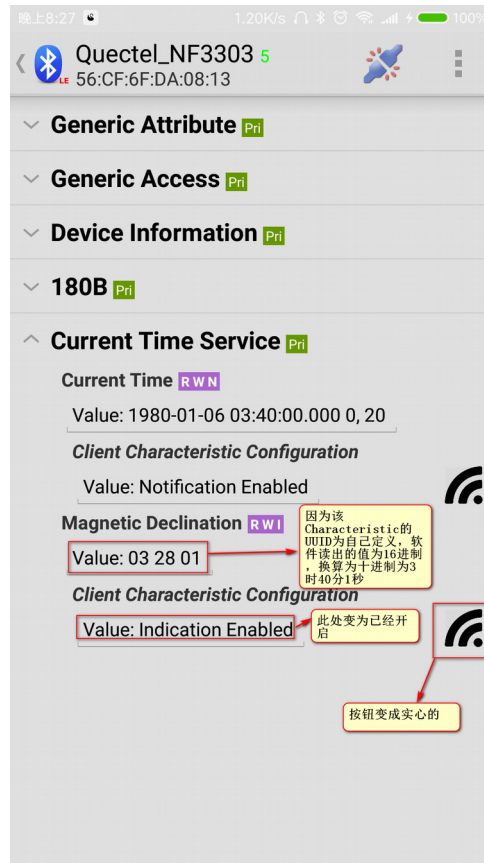


点击“Current Time”旁的 Enable Notification 按钮，按钮将会变成实心的。Value 代表的时间将会每 10s 更新一次。如下图所示：



点击“Magnetic Declination”旁的 Enable Notification 按钮，按钮将会变成实心的。Value 代表的时间将会每 5s 更新一次。如下图所示：

QUECTEL



7、断开连接。关闭开启的 Notification 和 Indication 功能。接着点击断连接按钮开连接。如下图所示：

