SYD8811 透传使用说明

2019 年 5 月 27 日

19:43

这里提供一个透传的通用 demo，主要功能是程序开机的时候正常广播，蓝牙连接上并且使能了 notify 功能后，SYD8811 一直打开串口，这时候主机端（手机 APP)发送的任何数据都会原封不动的通过串口 0（GPIO15,GPIO16 管脚）发送给主控制器或者 PC。本文章对应的程序在：“SYD8811_SDK\Source Code\SYD8811_ble_peripheral\1.SYD8811_BLE_UART_notifyen_open_power”

这里先介绍和蓝牙相关的内容，本程序中广播间隔是：

adv_params.type = ADV_IND;

adv_params.channel = 0x07; // advertising channel : 37 & 38 & 39

adv_params.interval = 1600; // advertising interval : 1000ms (1600 * 0.625ms)

adv_params.timeout = 0x1e; // timeout : 30s

整个程序运行阶段只有定时器作为时钟源：

```
    /*
    当POWERDOWN_WAKEUP时，
    PW_CTRL->DSLP_LPO_EN = true，这些中断源才能唤醒并复位运行，
    如果是false就只有pin能唤醒并复位运行
    */
    pw_cfg.wakeup_type = SLEEP_WAKEUP;
    pw_cfg.wdt_wakeup_en = (bool)false;
    pw_cfg.rtc_wakeup_en = (bool)false;
    pw_cfg.timer_wakeup_en = (bool)true;
    pw_cfg.gpi_wakeup_en = (bool)false;
    pw_cfg.gpi_wakeup_cfg = WAKEUP_PIN; //中断唤醒pin
    WakeupConfig(&pw_cfg);
```

在没有使能蓝牙 notify 的时候蓝牙将正常进入休眠，当使能了 notify 后程序将不会休眠：

```
45
46          if(start_tx & 0x01)     //connected
47          {
48              //UartEn(true); //不允许RF sleep时关闭XO
49              send_to_master();
50          }
51          else
52          {
53  //          #if defined(_DEBUG_) || defined(_SYD_RTT_DEBUG_)
54  //           DBGPRINTF(("goto SystemSleep\r\n"));
55  //          #endif
56              SystemSleep();              //系统睡眠
57          }
```

在使能或者失能 notify 的时候要特殊对 uarten 的操作，使能 notify 后外部 32MHZ 晶振不会被关闭，功耗比较高：

```
455     else if(p_evt->evt_code == GAP_EVT_ATT_HANDLE_CONFIGURE)
456     {
457         if(p_evt->evt.att_handle_config_evt.uuid == BLE_UART)
458         {
459             if(p_evt->evt.att_handle_config_evt.value == BLE_GATT_NOTIFICATION)
460             {
461                 start_tx |= 0x01;
462                 UartEn(true);   //不允许RF sleep时关闭XO，休眠的时候因为32Mhz晶振还在，所以功耗很高
463
464                 //clr uart rx fifo
465                 uart_rx_buf.header = uart_rx_buf.tail;
466
467                 #ifdef _GPIO_LED_CONTROL_
468                 GPIO_Pin_Set(U32BIT(GPIO_LED_NOTIFYEN));
469                 #endif
470
471                 DBGPRINTF(("UART notify Enabled!\r\n"));
472             }
473             else
474             {
475                 start_tx &= ~0x01;
476                 UartEn(false);  //允许硬件自由控制32Mhz晶振，休眠的时候功耗很低
477
478                 #ifdef _GPIO_LED_CONTROL_
479                 GPIO_Pin_Clear(U32BIT(GPIO_LED_NOTIFYEN));
480                 #endif
481                 DBGPRINTF(("UART notify Disabled!\r\n"));
482             }
483         }
484     }
```

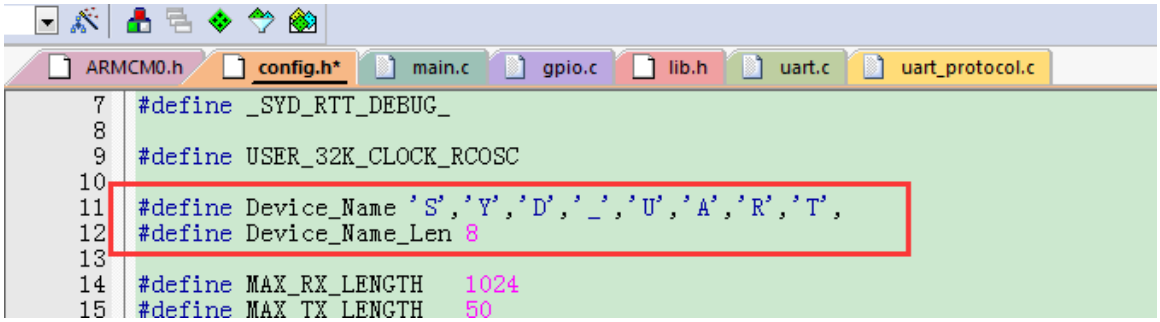
在断线的时候也会关闭外部 32MHZ 晶振，这里不管是失能 notify 或者断线都能够把功耗降低下来：

```
    else if(p_evt->evt_code == GAP_EVT_DISCONNECTED)
    {
        DBGPRINTF("Disconnected,reson:0x%02x\r\n", p_evt->evt.disconn_evt.reason);

        start_tx = 0;
        connect_flag=0;

        //clr uart rx fifo
        uart_rx_buf.header = uart_rx_buf.tail;
        Timer_Evt_Stop(EVT_1S_OTA);
        setup_adv_data();       //断开连接之后功耗大10uA
        StartAdv();

        #ifdef _GPIO_LED_CONTROL_
        GPIO_Pin_Clear(U32BIT(GPIO_LED_CONNECT));
        GPIO_Pin_Clear(U32BIT(GPIO_LED_NOTIFYEN));
        #endif

        UartEn(false);  //不允许RF sleep时关闭XO
        DBGPRINTF(("start adv @ disc!\r\n"));
    }
```

在连接上后将开启一个 2S 的定时器，2S 定时器计时 6S 后调用 BLSetConnectionUpdate 函数，这时候底层发出提高蓝牙连接间隔的请教来降低功耗，这里启用了智能连接参数管理的机制（smart_update_latency）用于管理连接参数，该机制在有数据参数的时候将关闭 latency 来提高传输速度，这里蓝牙连接间隔设置如下：

```
/* connection parameters */
smart_params.updateitv_target=0x0050;   //target connection interval (60 * 1.25ms = 75 ms)
smart_params.updatesvto=0x0258;   //supervisory timeout (400 * 10 ms = 4s)
smart_params.updatelatency=0x000A;
```

蓝牙名称定义如下：



下面介绍 PC 或者其他 MCU 发送数据到 SYD8811 的过程，程序上电运行的时候首先初始化 uart0 作为和 PC 端交互的串口，使用如下语句 uart_0_init，设置波特率为 115200，使能串口中断的功能，该函数具体内容如下：

```
//
void uart_0_init(void)
{
    PIN_CONFIG->PIN_15_SEL = PIN_SEL_UART_RXD0;
    PIN_CONFIG->PIN_16_SEL = PIN_SEL_UART_TXD0;

    //PM
    PIN_CONFIG->PAD_21_INPUT_PULL_UP = 0;
    PIN_CONFIG->PAD_22_INPUT_PULL_UP = 0;

    UART_0_CTRL->CLK_SEL = 0;        /* 1=32M, 0=16M */

    UART_0_CTRL->BAUDSEL = UART_BAUD_115200;
    UART_0_CTRL->FLOW_EN = UART_RTS_CTS_DISABLE;

    UART_0_CTRL->RX_INT_MASK = 0;    /* 1=MASK */
    UART_0_CTRL->TX_INT_MASK = 1;    /* 1=MASK */

    UART_0_CTRL->PAR_FAIL_INT_MASK = 1; /* 1=MASK */
    UART_0_CTRL->par_rx_even = 1;    /* 1=Even, 0=Odd */
    UART_0_CTRL->par_rx_en = 0;      /* 1=Rx Parity check enable */

    UART_0_CTRL->par_tx_even = 1;    /* 1=Even, 0=Odd */
    UART_0_CTRL->par_tx_en = 0;      /* 1=Tx Parity check enable */

    //clr Int Flag
    UART_0_CTRL->RI = 0;
    UART_0_CTRL->TI = 0;
    UART_0_CTRL->PAR_FAIL = 1;

    UART_0_CTRL->RX_FLUSH = 1;       /* clr rx fifo and set RX_FIFO_EMPTY */

    NVIC_EnableIRQ(UART0_IRQn);

    UART_0_CTRL->UART_EN = 1;
}
```

当 PC 端通过串口发送数据过来并且 APP 使能了 notify 的时候，SYD8811 的串口外设将能够正确进入 UART0_IRQHandler 函数，该函数调用了 uartRx_callback 来处理串口上的数据，uartRx_callback 函数把串口上的数据保存到 uart_rx_buf 的数据缓存区中，该函数具体内容如下：

```
5  void uartRx_callback(void)
6  {
7      uint8_t cnt,i,temp[4];
8      uart_0_read(&cnt,temp);
9
10     for(i=0;i<cnt;i++)
11     {
12         uart_rx_buf.data[uart_rx_buf.header]=temp[i];
13         uart_rx_buf.header++;
14         if(uart_rx_buf.header >= MAX_RX_LENGTH)
15         {
16             uart_rx_buf.header = 0;
17         }
18     }
19 }
```

在 while(1)主循环体中，如果发现 app 已经使能了 notify 并且 uart_rx_buf 的数据缓存区中有 PC 发过来的数据将调用 send_to_master 函数把串口接收到的数据发送给 APP，该函数把 uart_rx_buf 中的数据分为一个一个 20Byte 的数据包一次发送给 APP，该函数具体的内容如下：

```
310 
311 static void send_to_master(void)
312 {
313     uint32_t len, header, tail;
314 
315     header = uart_rx_buf.header;
316     tail   = uart_rx_buf.tail;
317 
318     if(header != tail)
319     {
320         #if DEFAULT_DESIRED_SLAVE_LATENCY
321         if(latency_state==0)
322         {
323             GAPConnectionLatencyMode(false);
324             latency_state=1;
325             //DBGPRINTF(("Latency Disabled!\r\n"));
326             uart_delay_cnt = 0;
327             timer_0_enable(TIMER_0_INTERVAL, timer_0_callback);
328         }
329         #endif
330     }
331     else
332     {
333         return;
334     }
335 
336     //caculate len
337     if(header >= tail)
338     {
339         len = header - tail;
340     }
341     else
342     {
343         len = header + MAX_RX_LENGTH - tail;
344     }
345 
346     if(len>=20)
347     {
348         for(; len>=20; len-=20)
349         {
350             uint8_t send_data[20], i;
351 
352             uart_delay_cnt = 0;
353 
354             //memcpy 20 byte
355             for(i=0; i<20; i++)
356             {
357                 send_data[i] = uart_rx_buf.data[tail++];
358 
359                 if(tail >= MAX_RX_LENGTH)
360                 {
361                     tail = 0;
362                 }
363             }
364             //DBGPRINTF(("BLE_Send 20\r\n"));
365             //send
366             if(BLE_SendUARTData(send_data, 20))
367             {
368                 uart_rx_buf.tail = tail;
369             }
370             else
371             {
372                 break;
373             }
374         }
375     }
376     else if(len >= 1)
377     {
378         if(uart_delay_cnt >= 3)
379         {
380             uint8_t send_data[20],i;
381             //memcpy len byte
382             for(i=0; i<len; i++)
383             {
384                 send_data[i] = uart_rx_buf.data[tail++];
385 
386                 if(tail >= MAX_RX_LENGTH)
387                 {
388                     tail = 0;
389                 }
390             }
391             //DBGPRINTF(("BLE_Send %x %x\r\n",len,uart_delay_cr
392             //send
```

下面介绍 APP 发送数据到 SYD8811，然后 SYD8811 通过串口转发给 PC 端或者其他 MCU 的流程，当 APP 对 SYD8811 进行写操作的时候，SYD8811 蓝牙底层将调用 ble_init 注册的 ble_evt_callback 钩子函数，并且进入 if(p_evt->evt_code == GAP_EVT_ATT_WRITE)分支，这里调用 ble_gatt_write 函数对该蓝牙行为进行处理：

```
void ble_evt_callback(struct gap_ble_evt *p_evt)
{
    if(p_evt->evt_code == GAP_EVT_ATT_WRITE)
    {
        #ifdef _GPIO_LED_CONTROL_
        GPIO_Pin_Turn(U32BIT(GPIO_LED_WRITEING));
        #endif

        ble_gatt_write(p_evt->evt.att_write_evt);
    }
    else if(p_evt->evt_code == GAP_EVT_ATT_READ)
    {
        #ifdef _GPIO_LED_CONTROL_
        GPIO_Pin_Turn(U32BIT(GPIO_LED_READING));
        #endif

        ble_gatt_read(p_evt->evt.att_read_evt);
    }
    else if(p_evt->evt_code == GAP_EVT_CONNECTED)
```

ble_gatt_write 函数把蓝牙上的数据填充到 uart_tx_buf.data 缓存区的数据区中：

```
static void ble_gatt_write(struct gap_att_write_evt evt)
{
    if(evt.uuid == BLE_UART_Write_UUID)
    {
        // rx data

        uint32_t i;
        uart_tx_buf.data[uart_tx_buf.header][0] = evt.sz;
        for(i=1; i<=evt.sz; i++)
        {
            uart_tx_buf.data[uart_tx_buf.header][i] = evt.data[i-1];
        }
        uart_tx_buf.header++;
        if(uart_tx_buf.header >= MAX_TX_LENGTH)
        {
            uart_tx_buf.header = 0;
        }
    }
    else if(evt.uuid== BLE_OTA_Read_Write_UUID)
    {
        ota_cmd(evt.data, evt.sz);
    }
}
```

在 while(1)主循环体中，如果发现 art_tx_buf 缓存区中有 app 发送过来的数据将调用 sendToUart 函数把 APP 发送过来的数据发送给 PC 端，sendToUart 函数最终调用 uart_cmd 函数把数据往串口上写，sendToUart 函数具体内容如下：

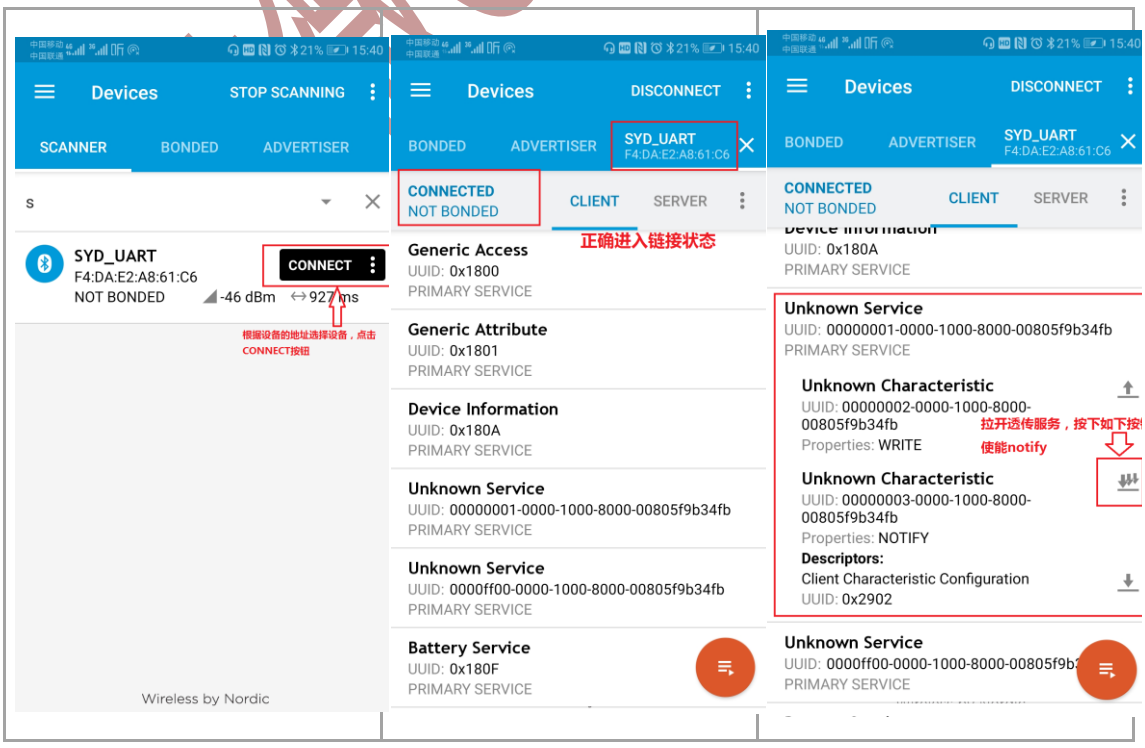
```c
//把蓝牙收到的数据一次从uart全发出去
static void sendToUart(void)
{
    uint32_t header= uart_tx_buf.header;

    if(header != uart_tx_buf.tail)
    {
        while(header != uart_tx_buf.tail)
        {
            uart_cmd(&uart_tx_buf.data[uart_tx_buf.tail][1], uart_tx_buf.data[uart_tx_buf.tail][0]);

            uart_tx_buf.tail++;
            if(uart_tx_buf.tail >= MAX_TX_LENGTH)
            {
                uart_tx_buf.tail = 0;
            }
        }
    }
}
```

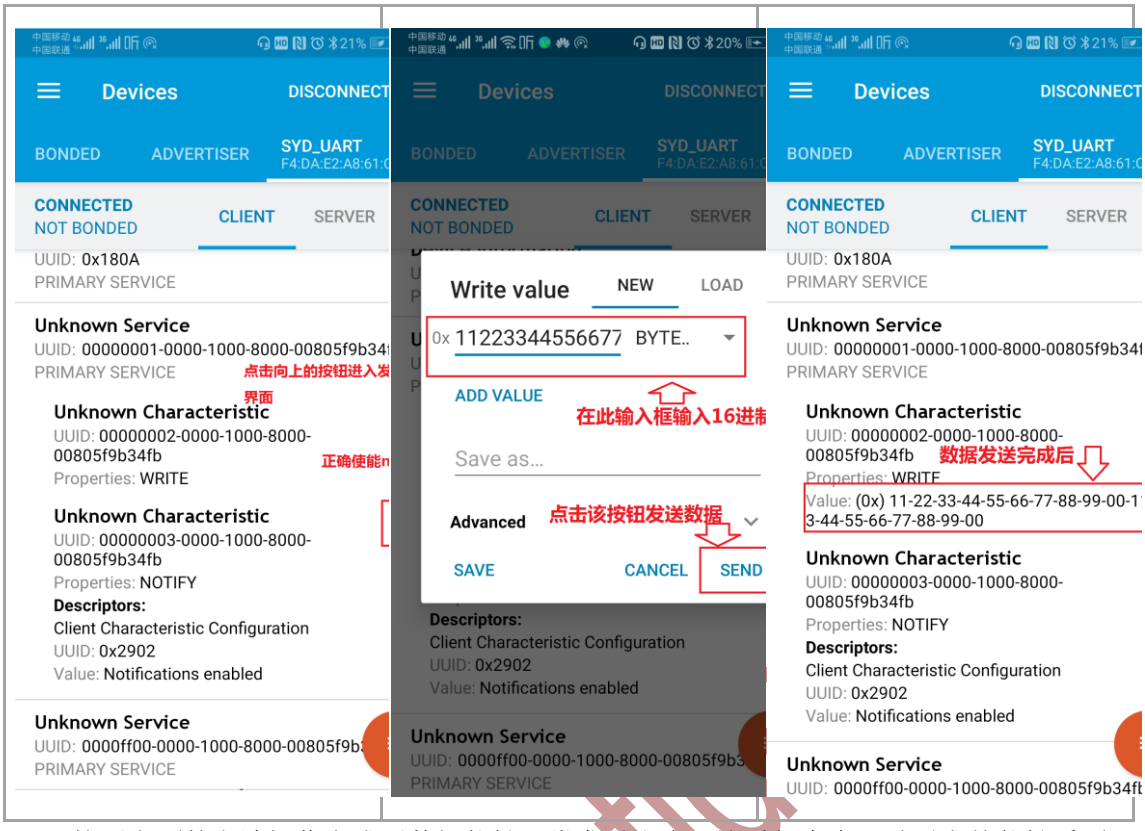
指示灯的控制，为了能够让与 SYD8811 连接的 MCU 更好的掌握当前蓝牙的状态，这里在关键的蓝牙状态发生变换的时候将会控制 GPIO 管脚以指示外部 MCU 蓝牙状态，该程序中一共控制的有 4 个 GPIO，他们的定义如下：

```c
#define MAX_RX_LENGTH    1024
#define MAX_TX_LENGTH    50

#define GPIO_UART_TX     GPIO_16
#define GPIO_UART_RX     GPIO_15

#define WAKEUP_PIN       BIT10

#define _GPIO_LED_CONTROL_    //蓝牙状态指示灯是否打开的宏，注释本行SYD8811将不控制蓝牙状态管脚

#define GPIO_LED_CONNECT     GPIO_9    //蓝牙连接状态指示灯，0：当前蓝牙处于断线状态 1：当前蓝牙处于连接状态
#define GPIO_LED_NOTIFYEN    GPIO_8    //蓝牙notify开关的状态指示灯，0：当前未使能notify 1：当前使能了notify
#define GPIO_LED_WRITEING    GPIO_7    //蓝牙发生了写操作的状态指示灯，APP对SYD8811进行一次写操作，SYD8811将翻转该管脚一次
#define GPIO_LED_READING     GPIO_6    //蓝牙发生了读操作的状态指示灯，APP对SYD8811进行一次读操作，SYD8811将翻转该管脚一次
```

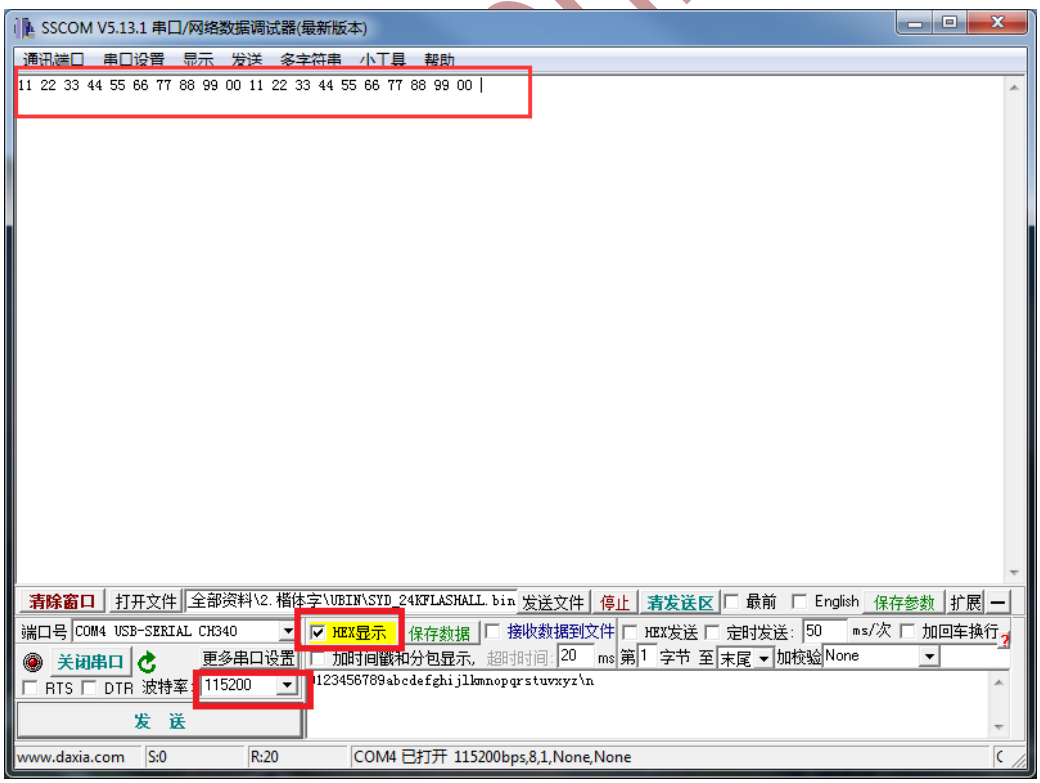
透传程序测试流程：

编译下载程序后使用使用 NRF connect 扫描蓝牙设备然后连接，进行 APP 发送数据给 PC 串口助手的测试：

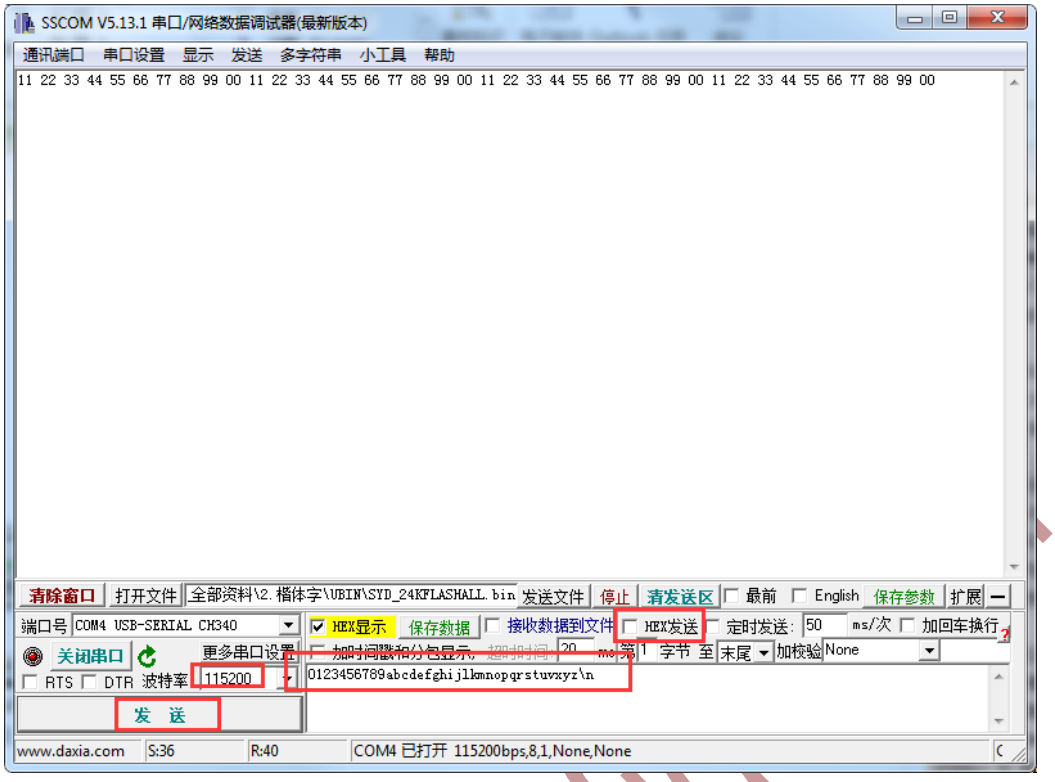




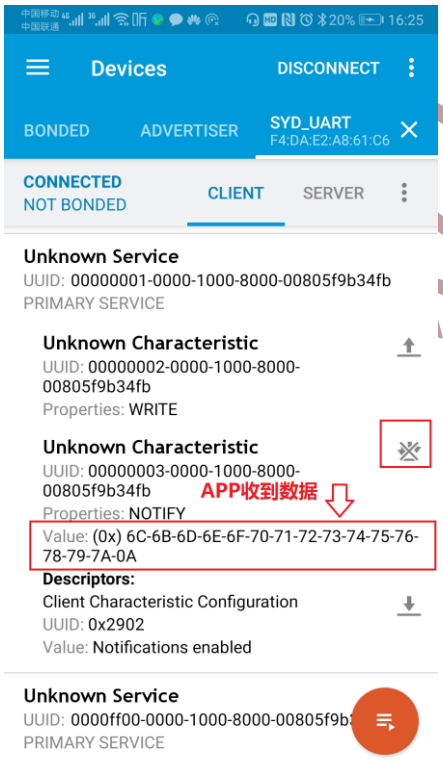
按照上面的方法操作完成后数据能够正常发送出去，这时候在串口助手上就能够看到 app 发送过来的数据：



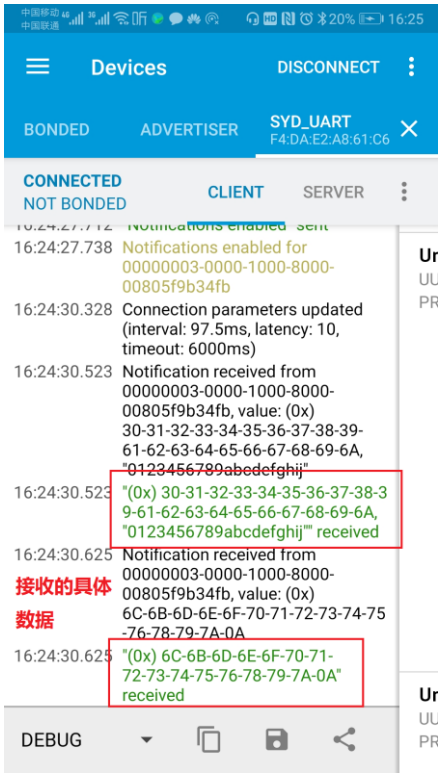
如果要测试 PC 端串口助手发送给 APP，可以这样操作，首先在 APP 上连接蓝牙并且使能 notify，然后在串口转助手中输入正确的数据，然后点击“发送”按钮，这时候将能够在 APP 看到 PC 端输入的数据：



这时候在 App 上可以看到有数据发送过来了：

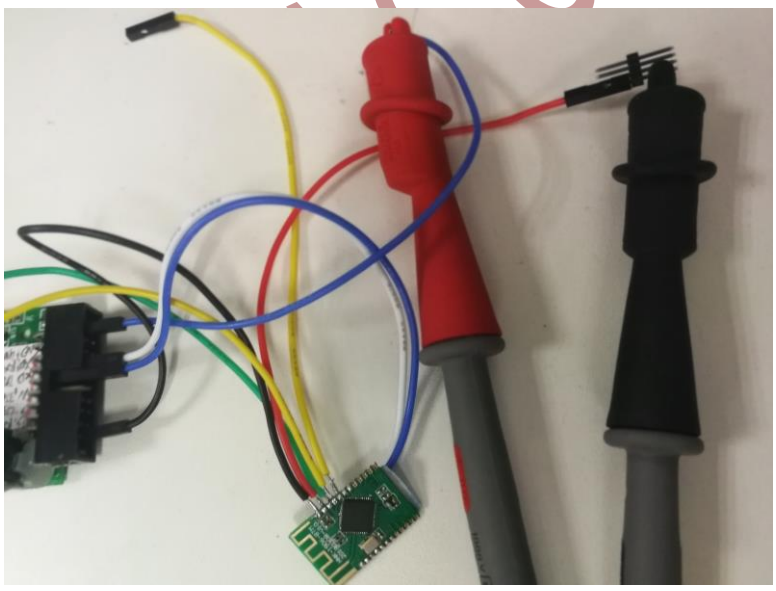


这时候在这个界面向右滑动即可看到具体的数据的 log：

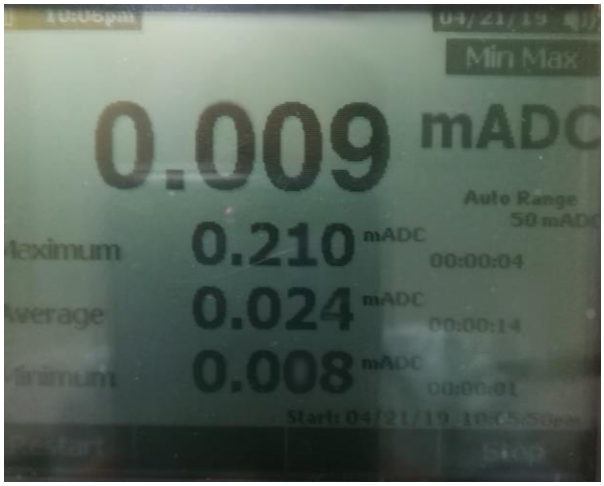


SYD8811 透传模块实物和功耗的测试：

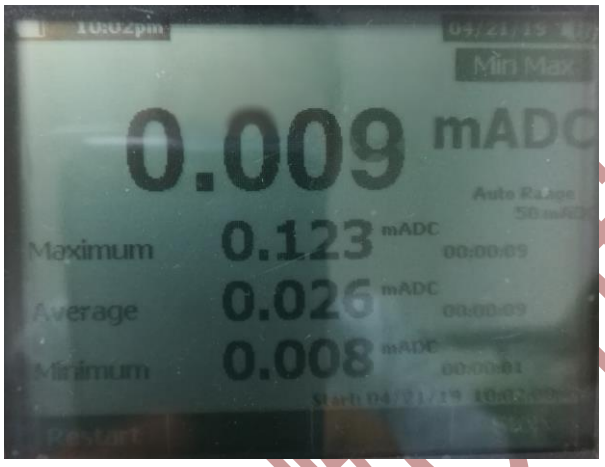
这里透传模块 VBAT 和 VDDIO 同时连接到了 3.3V，测量 VBAT 和 VDDIO 合在一起的电流，经过测试发现带着串口线和 SWD 总线对功耗没有影响，所以这里没有摘下这两个总线，透传模块实物图如下：



在广播的时候功耗如下（广播间隔为 1S，三个通道同时广播）：

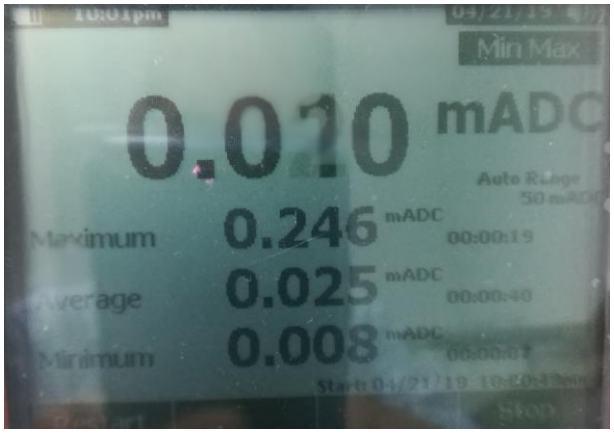


连接上后有一段时间（6S)比较高的电流（还没有使用低功耗的连接参数的时候），然后电流恢复稳定，电流值如下：



在这个程序中使能 notify 后程序不再休眠（客户可以根据自己的场合调整，比如下一个程序就通过 IO 口来控制数据的发送）,不休眠的电流如下：

断线后电流将恢复到广播阶段的水平，电流如下：



<<SYD8811_SDK_20190522.7z>>