

SYD8821 透传使用说明

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，SYD8821 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（GPIO20,GPIO21 管脚）发送给主控制器或者 PC。本文章对应的程序在：“SYD8821_SDK\Source Code\SYD8821_ble_peripheral\1.SYD8821_BLE_UART”

关于 SYD8821 芯片的使用请看文章：《SYD8821 介绍》

关于蓝牙的一些基础只是请看文章：《SYD8811 透传使用说明》

下面介绍 PC 或者其他 MCU 发送数据到 SYD8821 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 ble_uart_init(UART_RTS_CTS_DISABLE, UART_BAUD_115200);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：

```
562 |
563 | void ble_uart_init(uint8_t flowctrl, uint8_t baud)
564 | {
565 |     pad_mux_write(20, 7);
566 |     pad_mux_write(21, 7);
567 |     if(flowctrl)
568 |     {
569 |         pad_mux_write(18, 7);
570 |         pad_mux_write(19, 7);
571 |     }
572 |
573 |     NVIC_DisableIRQ(UART0_IRQn);
574 |     UART_CTRL[0]->BAUD_SEL = baud;
575 |     UART_CTRL[0]->FLOWCTRL_EN = flowctrl;
576 |     UART_CTRL[0]->INT_RX_MASK = 0;
577 |     UART_CTRL[0]->UART_ENABLE = 1;
578 |
579 |     queue_init(&rx_queue[0], rx_buf[0], QUEUE_SIZE);
580 |
581 |     *(uint32_t *)0x20028024 |= U32BIT(UART0_IRQn);
582 |     *(uint32_t *)0x20028020 |= U32BIT(UART0_IRQn);
583 |     NVIC_EnableIRQ(UART0_IRQn);
584 | }
```

当 PC 端通过串口发送数据过来并且 APP 使能了 notify 的时候，SYD8821 的串口外设将能够正确进入 UART0_IRQHandler 函数，该函数调用了 enqueue 把串口上的数据保存到 rx_queue 的数据缓存区中，该函数具体内容如下：

```
34 |
35 | void UART0_IRQHandler(void)
36 | {
37 |     uint8_t int_st = UART_CTRL[0]->INT_STATUS;
38 |     uint8_t clear_int = int_st ^ UART_ALL_INT;
39 |
40 |     // Clear interrupt status
41 |     UART_CTRL[0]->INT_STATUS = clear_int;
42 |
43 |     if (int_st & UART_RX_INT) {
44 |         do {
45 |             enqueue(&rx_queue[0], UART_CTRL[0]->RX_DATA);
46 |             while (!UART_CTRL[0]->RXFF_EMPTY);
47 |         }
48 |     }
```

在 while(1)主循环体中，如果发现 app 已经使能了 notify 并且 rx_queue 的数据缓存区中有 PC 发过来的数据将调用 uart_to_ble_transfer 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：

```

471 void uart_to_ble_transfer(void) {
472     if (start_tx == 1) {
473         if (rx_queue[UART_TOBLE_QUEUE_ID].head != rx_queue[UART_TOBLE_QUEUE_ID].tail)
474         {
475             uint8_t i=0, num=rx_queue[UART_TOBLE_QUEUE_ID].tail-rx_queue[UART_TOBLE_QUEUE_ID].head/20+1, num_residue=(rx_queue[UART_TOBLE_QUEUE_ID].tail-rx_queue[UART_TOBLE_QUEUE_ID].head)%20;
476             for (i=0; i<num; i++) {
477                 if (i!=num-1) {
478                     if (BLE_SendData(&rx_queue[UART_TOBLE_QUEUE_ID].data[rx_queue[UART_TOBLE_QUEUE_ID].head], num_residue)) {
479                         num--;
480                         rx_queue[UART_TOBLE_QUEUE_ID].head+=num_residue;
481                     } else break;
482                 }
483             }
484             else {
485                 if (BLE_SendData(&rx_queue[UART_TOBLE_QUEUE_ID].data[rx_queue[UART_TOBLE_QUEUE_ID].head], 20)) {
486                     rx_queue[UART_TOBLE_QUEUE_ID].head+=20;
487                 } else break;
488             }
489         }
490     }
491 }
492 }
493 }
494 }
495 }
496 }

```

同时这里还会设立一个 10MS 的定时器，20MS 后发现出口再也没有发送数据过来，这里就把串口发过来的零散数据发送出去：

```

631 #endif
632
633 while(1)
634 {
635     ble_sched_execute();
636     //gpo_toggle(LED2_Pin);
637
638     if (start_tx==1) {
639         if (!is_queue_empty(&rx_queue[UART_TOBLE_QUEUE_ID])) {
640             if (timer_state_get(TIMER_WAIT_TIMER_ID) == false)
641             {
642                 Timer_Module_Init();
643             }
644             if (queue_size(&rx_queue[UART_TOBLE_QUEUE_ID]) >= UART_TOBLE_THU)
645             {
646                 timer_wait_cnt=0;
647                 uart_to_ble_transfer();
648             }
649         }
650     }

```

```

506 void timer_uart_wait(void)
507 {
508     timer_wait_cnt++;
509     if (timer_wait_cnt >= MAX_UART_WAIT)
510     {
511         timer_wait_cnt=0;
512         timer_disable(UART_WAIT_TIMER_ID);
513
514         if (!is_queue_empty(&rx_queue[UART_TOBLE_QUEUE_ID]))
515         {
516             uart_to_ble_transfer();
517         }
518     }
519 }
520
521 void Timer_Module_Init(void)
522 {
523     timer_disable(TIMER_1);
524     timer_enable(TIMER_1, timer_uart_wait, 32768/100, 1); //32768 = 1S 16384 = 500ms
525     NVIC_EnableIRQ(TIMER1_IRQn);
526 }
527 }

```

下面介绍 APP 发送数据到 SYD8821，然后 SYD8821 通过串口转发给 PC 端或者其他 MCU 的流程，当 APP 对 SYD8821 进行写操作的时候，SYD8821 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，ble_gatt_write 函数把蓝牙上的数据填充到 rx_queue 缓存区的数据区中：

```

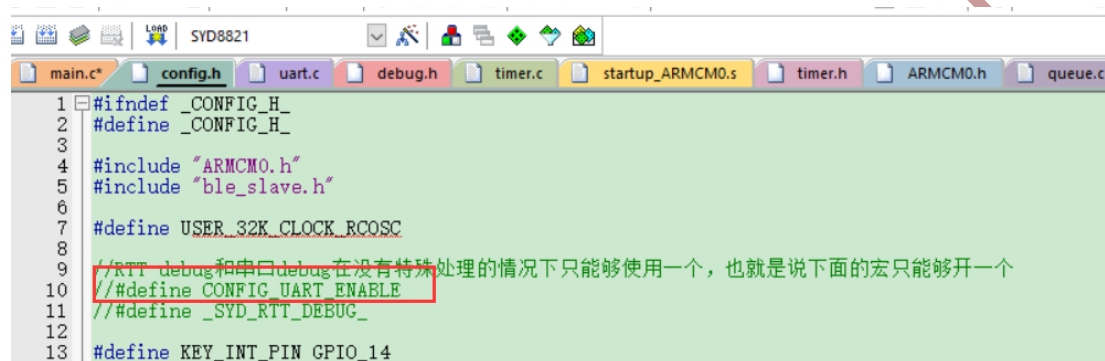
241 //接收函数
242 static void ble_gatt_write(struct gap_att_write_evt evt)
243 {
244     // rx data
245     //evt.data是收取到的数据buf
246     if (evt.uuid == BLE_UART_Write_UUID)
247     {
248         enqueue_all(&rx_queue[BLE_TOUART_QUEUE_ID], evt.data, evt.sz);
249     }
250 #ifdef _OTA_
251 else if (evt.uuid == BLE_OTA_Read_Write_UUID)
252 {
253     update_latency_mode=0;
254     ota_cmd(evt.data, evt.sz);
255 }
256 #endif
257 }

```

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

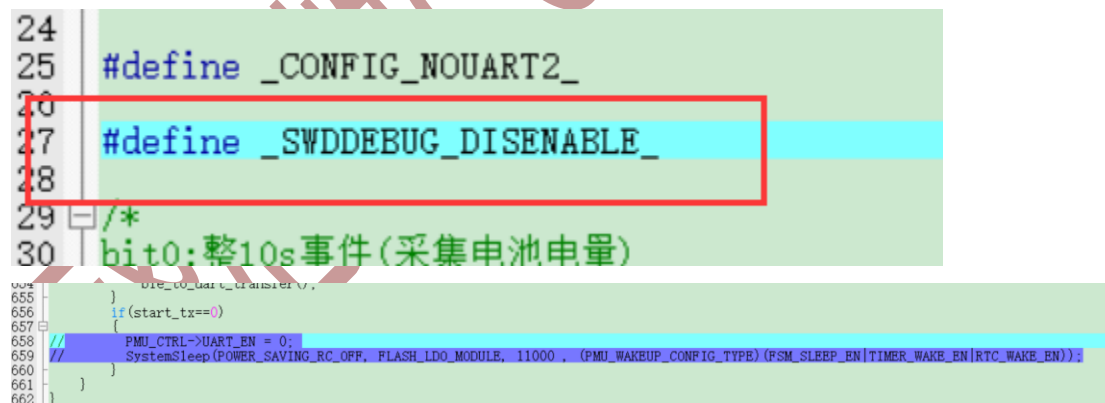
```
197
198 void ble_to_uart_transfer(void) {
199     uint8_t buf=0;
200     while(dequeue(&rx_queue[BLE_TOUART_QUEUE_ID], &buf))
201     {
202         uart_write(0, &buf, 1);
203     }
204 }
```

LOG 的查看，为了能够让 SYD8821 显示 log，这里打开宏：



```
main.c* config.h uart.c debug.h timer.c startup_ARMCM0.s timer.h ARMCM0.h queue.c
1 #ifndef _CONFIG_H_
2 #define _CONFIG_H_
3
4 #include "ARMCM0.h"
5 #include "ble_slave.h"
6
7 #define USER_32K_CLOCK_RCOSC
8
9 //RTT debug和串口debug在没有特殊处理的情况下只能使用一个，也就是说下面的宏只能开一个
10 #define CONFIG_UART_ENABLE
11 //define _SYD_RTT_DEBUG_
12
13 #define KEY_INT_PIN GPIO_14
```

但是这样做后 LOG 和 UART 的数据都将从 UART 输出，所以这里可以使用 RTT 来查看 log，但是使用 RTT 来查看 log 就要 SWD 时刻维持着运行状态，也就是说这里不能够睡眠，然后 SWDSDA 和 SWDCLK 的管脚状态必须设置为默认的 SWD 模式并且把睡眠去掉，所以看 log 的时候就不能够测试功耗了：



```
24
25 #define _CONFIG_NOUART2_
26
27 #define _SWDDEBUG_DISABLE_
28
29 /*
30 bit0: 整10s事件(采集电池电量)
31
32 ble_to_uart_transfer()
33 {
34     if(start_tx==0)
35     {
36         PMU_CTRL->UART_EN = 0;
37         SystemSleep(POWER_SAVING_RC_OFF, FLASH_IDO_MODULE, 11000, (PMU_WAKEUP_CONFIG_TYPE)(HSM_SLEEP_EN|TIMER_WAKE_EN|RTC_WAKE_EN));
38     }
39 }
```

```
void gpio_init(void)
{
    uint8_t i;
    for(i=0;i<39;i++)
    {
        switch(i)
        {
            case GPIO_0:
                pad_mux_write(i, 0);
                gpi_config(i,PULL_DOWN);
                gpo_config(i,0);
                break;

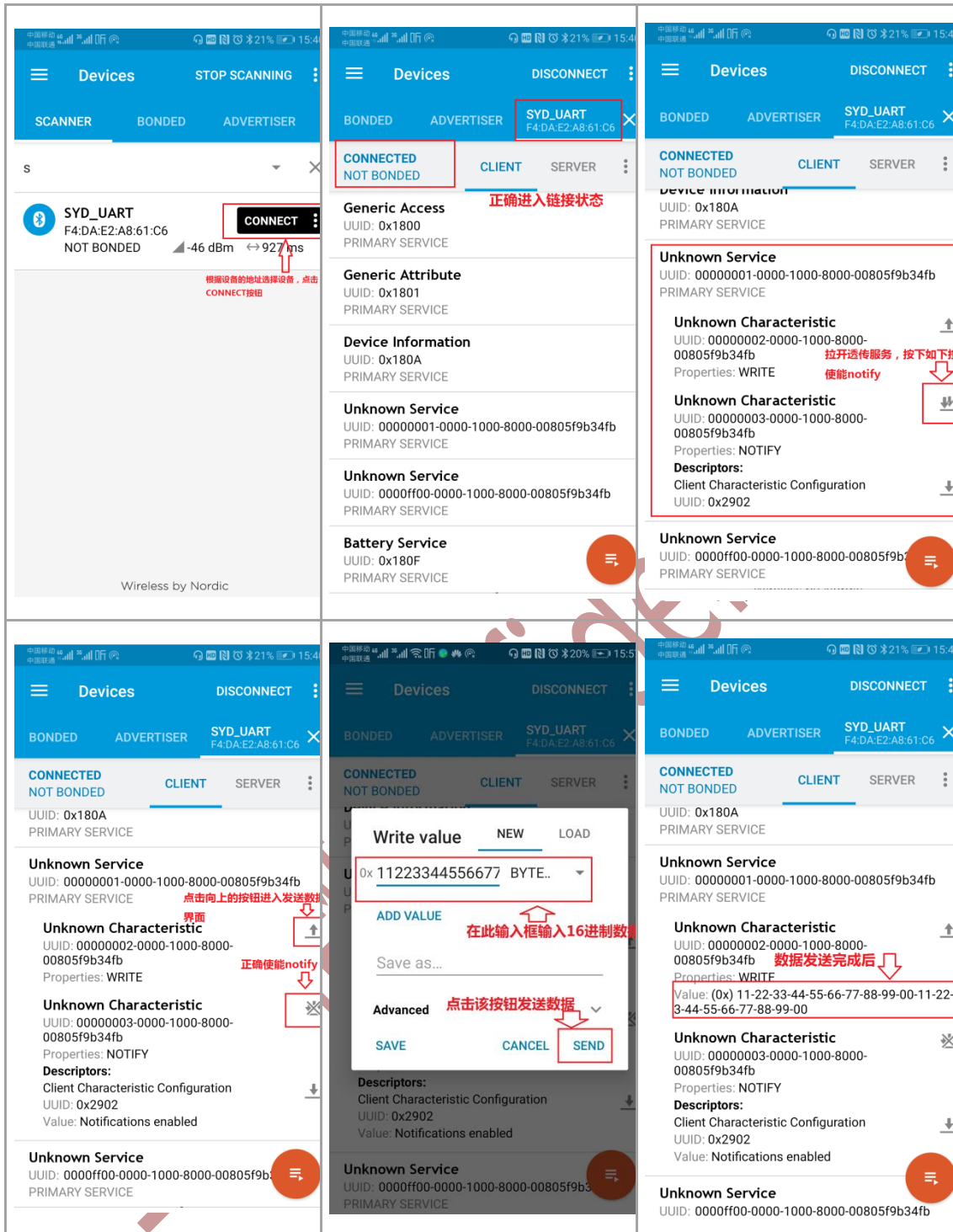
            case LED2_Pin:
            case LED1_Pin:
                pad_mux_write(i, 0);
                gpo_config(i,1);
                break;

            case GPIO_2:
            case GPIO_4:
            #ifdef _SWDDEBUG_DISABLE_
            case GPIO_31:
            #endif
                pad_mux_write(i, 0);
                gpo_config(i,0);
                break;

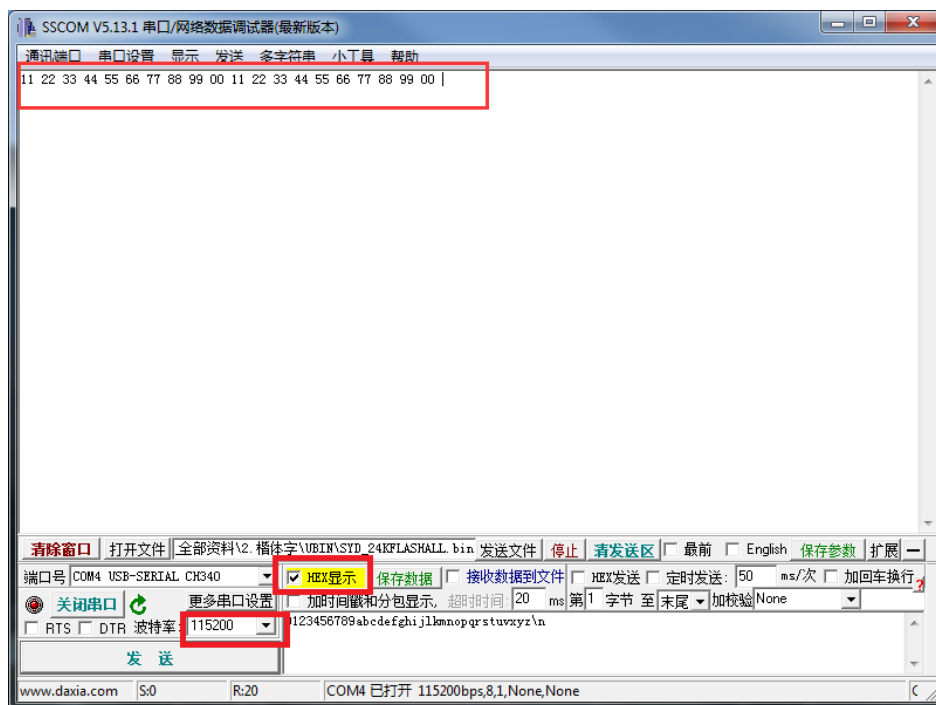
            case GPIO_24:
            case GPIO_30:
            #ifndef _SWDDEBUG_DISABLE_
            case GPIO_31:
            #endif
                break;
        }
    }
}
```

透传程序测试流程:

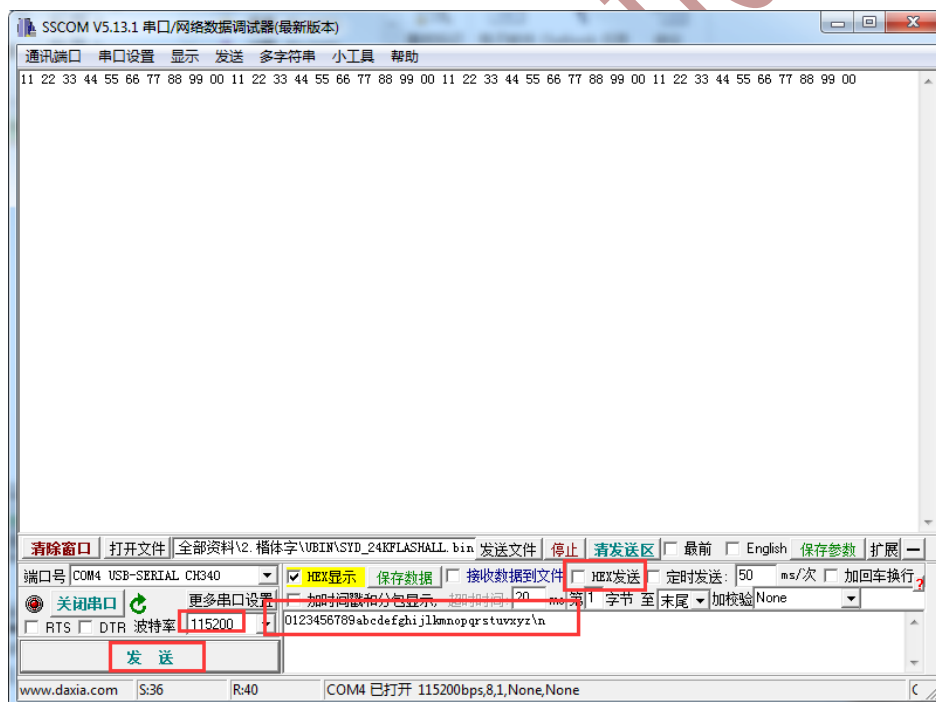
编译下载程序后使用使用 NRF connect 扫描蓝牙设备然后连接,进行 APP 发送数据给 PC 串口助手的测试:



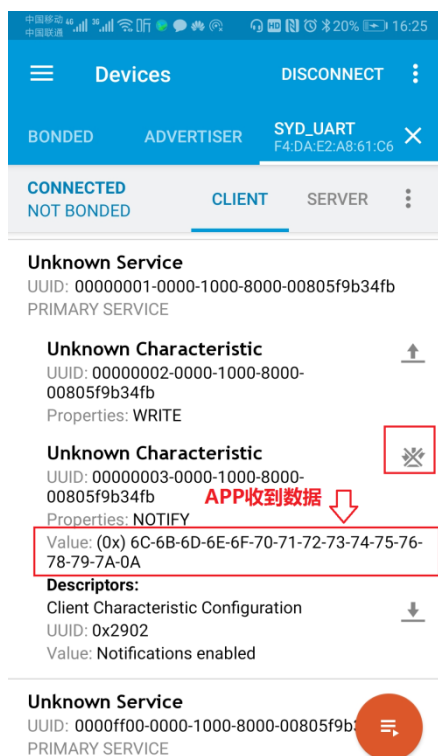
按照上面的方法操作完成后数据能够正常发送出去, 这时候在串口助手上就能够看到app 发送过来的数据:



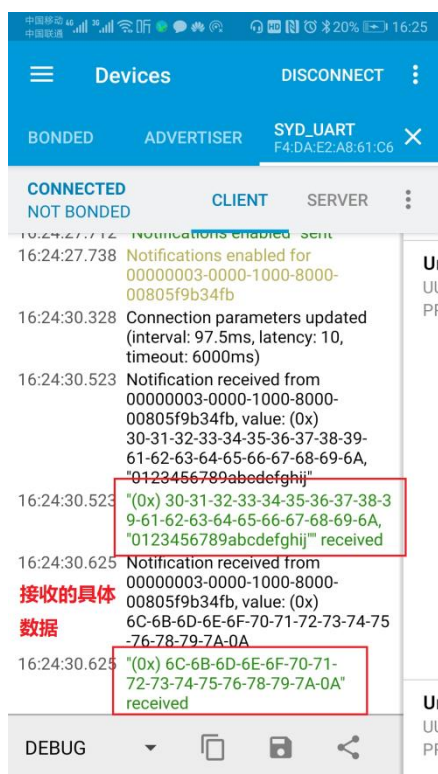
如果要测试 PC 端串口助手发送给 APP，可以这样操作，首先在 APP 上连接蓝牙并且能 notify，然后在串口助手输入正确的数据，然后点击“发送”按钮，这时候将能够在 APP 看到 PC 端输入的数据：



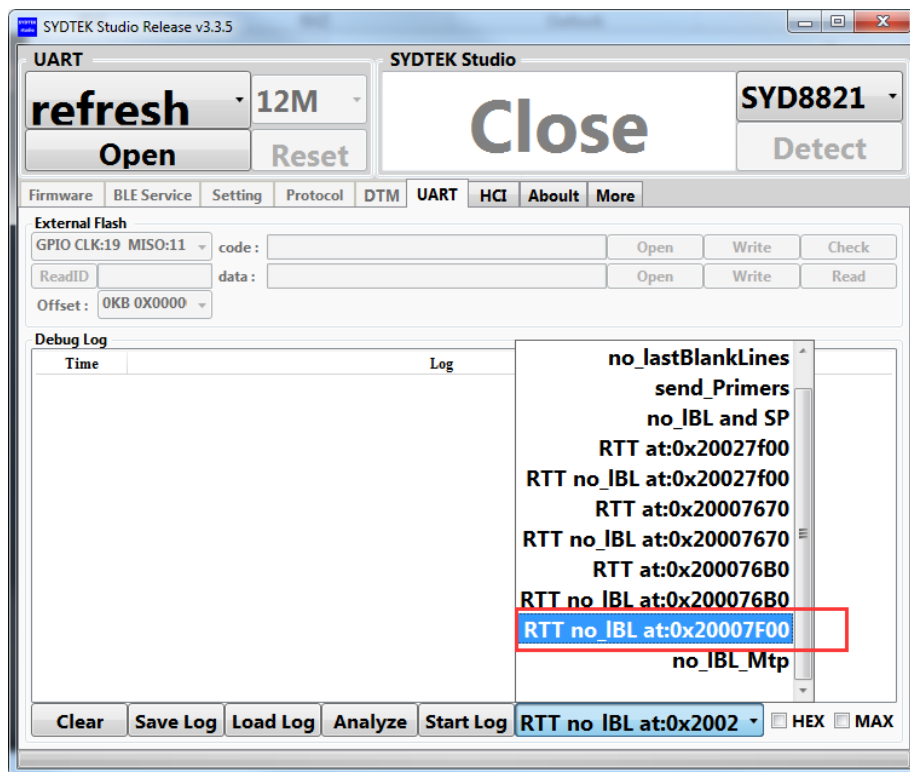
这时候在 App 上可以看到有数据发送过来了：



这时候在这个界面向右滑动即可看到具体的数据的 log:



使用《SYDTEK_Studio.exe》查看透传工程通过 RTT 输出的打印信息，关于 RTT 打印的功能请看《STDTEK RTT 打印的设置.pdf》，本工程 RTT 地址设置如下：



SYD8821 透传模块实物和功耗的测试:


Source
Code20191010 20