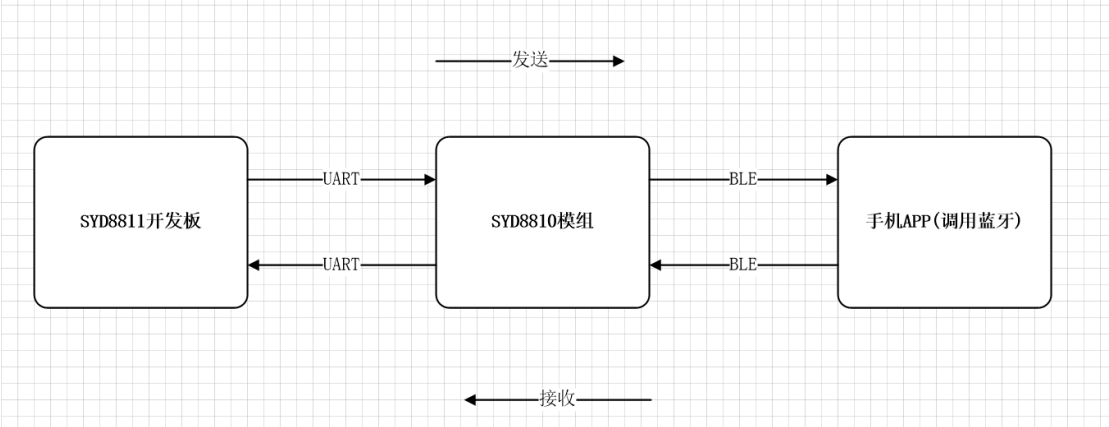
SYD8811 芯片透传的进阶之回环发送

为了更好的模拟现实的透传，这里做了如下测试环境：



首先是 SYD8811 开发板往 SYD8810 模组通过串口发送数据，然后 SYD8810 模组通过 BLE 往手机发送，手机收到数据后原路返回一样的数据到 SYD8811 开发板！

SYD8811 开发板这边的代码不单单是发送串口数据，还要进行数据的接收和对比，发送完数据后如果收到了 APP 发送过来的数据就会继续发送，如果 10s 都没有收到 APP 的数据，这里也会继续发送（因为 SYD8810 模组和手机 APP 都可能有延迟，所以这里设置了一个 10s 的超时时间），发送数据的处理如下：

```
759     while(1)
760     {
761         //ble_sched_execute();  //协议栈任务
762         //sendToUart(); //通过串口发送数据
763 
764         if(uart_rx_buf.header>=20)
765         {
766             Start_Send_to_Ble();
767             timer500ms_cnt=0;
768         }
769 
770         if(TIMER_EVT)
771         {
772             #ifdef  EVT_1S_UART
773             if(TIMER_EVT&EVT_1S_UART)
774             {
775                 timer500ms_cnt++;
776                 if(timer500ms_cnt>20)
777                 {
778                     Start_Send_to_Ble();
779                     timer500ms_cnt=0;
780                 }
781                 Timer_Evt_Clr(EVT_1S_UART);
782             }
783             #endif
784             #ifdef  EVT_2S
785             if(TIMER_EVT&EVT_2S)
786             {
787                 Connection_latency();   //连接参数相关设置
788                 Timer_Evt_Clr(EVT_2S);
789             }
790             #endif
791         }
792         if(RTC_EVT)
793         {
```

```
673 void Start_Send_to_Ble(void){
674     if(uart_rx_buf.header==0)
675     {
676         #if defined(_DEBUG_) || defined(_SYD_RTT_DEBUG_)
677         DBGPRINTF("No receive anything from BLE ");
678         #endif
679     }
680     else
681     {
682         #if defined(_DEBUG_) || defined(_SYD_RTT_DEBUG_)
683         DBGPRINTF("Receive %dByte from BLE ",uart_rx_buf.header);
684         #endif
685         if(memcmp(uart_rx_buf.data,data,20)==0)
686         {
687             ok_cnt++;
688             #if defined(_DEBUG_) || defined(_SYD_RTT_DEBUG_)
689             DBGPRINTF("Compare S/R OK!  ",uart_rx_buf.header);
690             #endif
691         }
692         else
693         {
694             faile_cnt++;
695             #if defined(_DEBUG_) || defined(_SYD_RTT_DEBUG_)
696             DBGPRINTF("Compare S/R Faile!\r\n",uart_rx_buf.header);
697             DBGHEXDUMP("RxData:\r\n",uart_rx_buf.data,20);
698             #endif
699         }
700     }
701     #if defined(_DEBUG_) || defined(_SYD_RTT_DEBUG_)
702     DBGPRINTF("Send 0X00-0X13(20Byte) num ok:%d faile:%d\r\n",ok_cnt,faile_cnt);
703     #endif
704     uart_rx_buf.header = 0;
705     uart_rx_buf.tail = 0;
706     memset(uart_rx_buf.data,0xff,20);
707     uart_cmd(data, 20);
708 }
```

当串口接收到数据的时候会填充 uart_rx_buf.data 数组，关键代码如下：

```
void uartRx_callback(void)
{
    uint8_t cnt,i,temp[4];
    uart_0_read(&cnt,temp);

    for(i=0;i<cnt;i++)
    {
        uart_rx_buf.data[uart_rx_buf.header]=temp[i];
        uart_rx_buf.header++;
        if(uart_rx_buf.header >= MAX_RX_LENGTH)
        {
            uart_rx_buf.header = 0;
        }
    }
}
```

这里循环发送数据，完整的流程从调用 uart_cmd 函数进行发送开始，到接收到数据并且填充到 uart_rx_buf.data 数组结束，循环往复的进行！

这里有一个比较特殊的处理就是，当 SYD8811 开发板发送完数据后在很长的时间内（2.5S）收不到手机原路返回的数据，这里就会重新进入发送流程，而打印出“No receive anything from BLE”的相关打印！如果正确的接收到数据则会增加 OK 的计数，反之增加 faile 的计数！

注意：当该开始发送数据或者蓝牙刚开始连接的时候有可能会有数据发送失败，这算是正常的，因为 SYD8810 并没有准备好接收数据！

如果数据接收成功的打印一般如下：

```
20:02:36:429  Receive 20Byte from BLE Compare S/R OK!  Send 0X00-0X13(20Byte) num ok:2671 faile:4
20:02:38:137  Receive 20Byte from BLE Compare S/R OK!  Send 0X00-0X13(20Byte) num ok:2672 faile:4
20:02:39:834  Receive 20Byte from BLE Compare S/R OK!  Send 0X00-0X13(20Byte) num ok:2673 faile:4
20:02:41:542  Receive 20Byte from BLE Compare S/R OK!  Send 0X00-0X13(20Byte) num ok:2674 faile:4
20:02:43:262  Receive 20Byte from BLE Compare S/R OK!  Send 0X00-0X13(20Byte) num ok:2675 faile:4
20:02:45:012  Receive 20Byte from BLE Compare S/R OK!  Send 0X00-0X13(20Byte) num ok:2676 faile:4
20:02:46:711  Receive 20Byte from BLE Compare S/R OK!  Send 0X00-0X13(20Byte) num ok:2677 faile:4
```

因为在透传过程中串口的数据是非常关键的，所以这里要把串口的优先级提升到最高（在 SYD8810 模组端）：

```
void nvic_priority(void){
    NVIC_SetPriority(LLC_IRQn   , 2);
    NVIC_SetPriority(RTC_IRQn   , 2);
    NVIC_SetPriority(SW_IRQn    , 2);
    NVIC_SetPriority(I2C0_IRQn , 2);
    NVIC_SetPriority(I2C1_IRQn , 2);
    NVIC_SetPriority(UART0_IRQn, 0);
    NVIC_SetPriority(UART1_IRQn, 2);
    NVIC_SetPriority(TIMER0_IRQn, 2);
    NVIC_SetPriority(TIMER1_IRQn, 2);
    NVIC_SetPriority(TIMER2_IRQn  , 2);
    NVIC_SetPriority(TIMER3_IRQn  , 2);
    NVIC_SetPriority(GPIO_IRQn    , 2);
    NVIC_SetPriority(HID_IRQn   , 2);
    NVIC_SetPriority(SPIM_IRQn    , 2);
    NVIC_SetPriority(CAP_IRQn , 0);
    NVIC_SetPriority(GPADC_IRQn , 2);
    NVIC_SetPriority(LLC2_IRQn, 2);
    NVIC_SetPriority(ISO7816_IRQn, 2);
    NVIC_SetPriority(IR_Tx_IRQn, 2);
    NVIC_SetPriority(TOUCH_IRQn, 2);
    NVIC_SetPriority(HPWM_IRQn   , 2);
    NVIC_SetPriority(HTIMER_IRQn  , 2);
    NVIC_SetPriority(IR_Rx_IRQn  , 2);
}


int main(void)
{
    __disable_irq();

    ble_init();  //蓝牙初始化，系统主时钟初始化64M,32K时钟初始化为LPO
    nvic_priority();

    //根据需要重新设置时钟为4M并校准
    MCUClockSwitch(SYSTEM_CLOCK_64M_RCOSC);
    RCOSCCalibration();
```

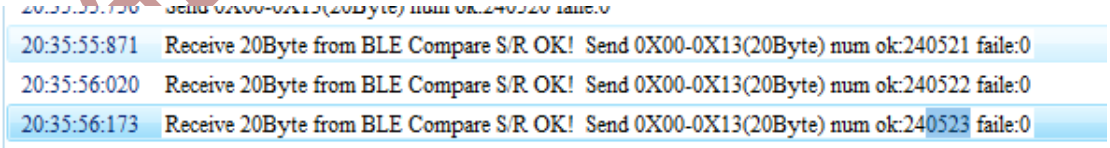
另外注意：在一般的蓝牙协议栈中校准的时候是要关闭总中断的，但是在串口透传中关闭总中断就有可能丢数据，所以这里建议使用下面这个 lib，或者使用外部 32.768Khz 的晶振，这样就没有校准的事情了：

syd8811_ble_lib2 0200310 194839.l

在测试的时候 APP 要选择上回环发送和自动清空的功能！

SYDTEK BLE.apk

经过一天的测试，最终蓝牙一共回环了 240523 次数据，没有一次错误！



Debug_log2020-3-12 203550.txt

本文所用到的源代码如下：

Source Code20200313 07

整体的透传测试结束！