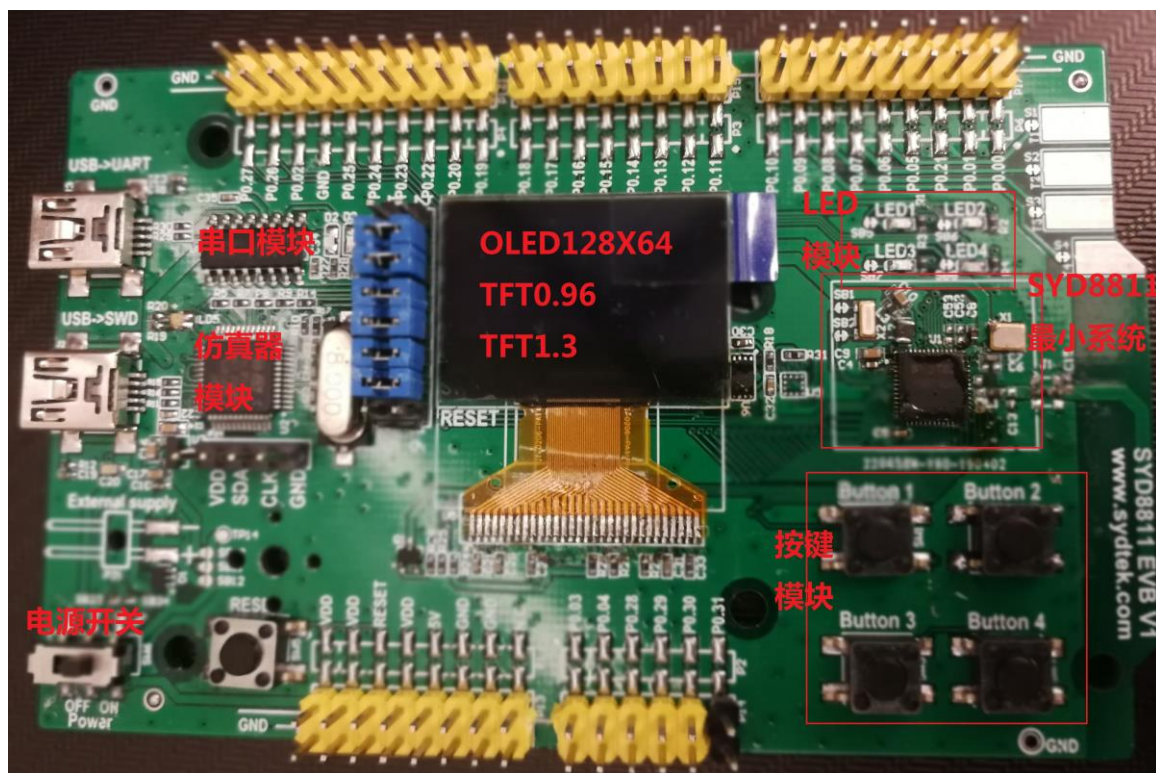


SYD8811 开发板使用说明

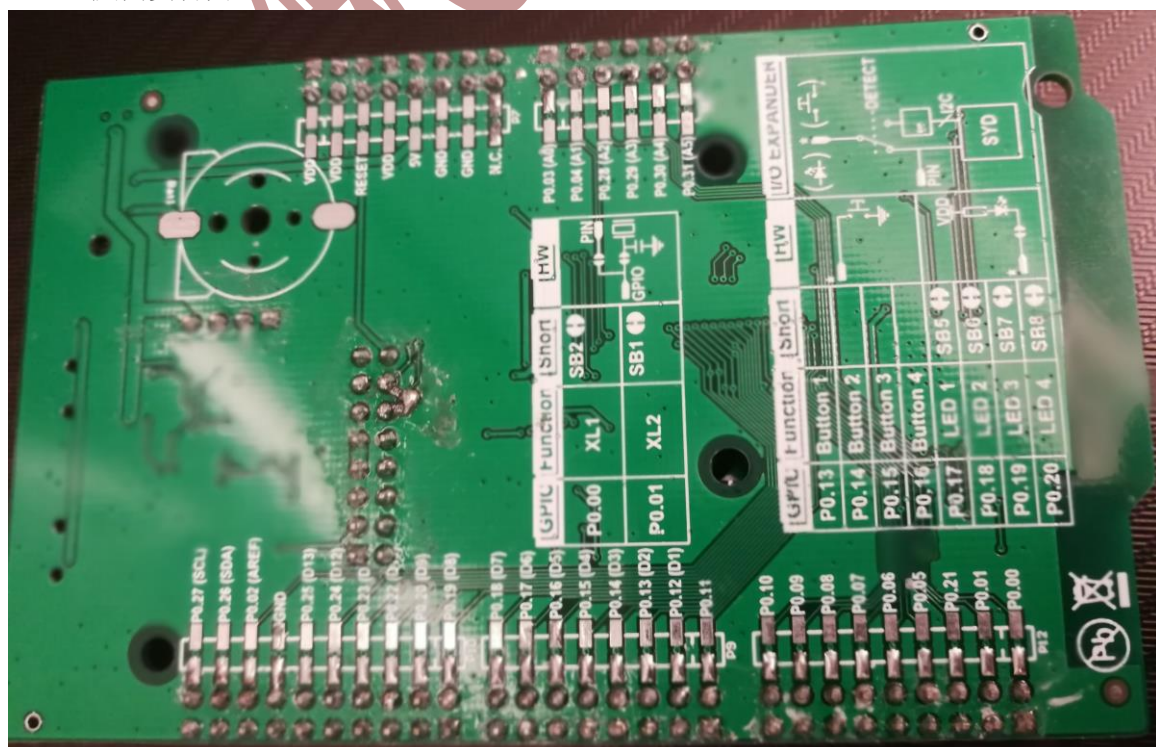
2019 年 6 月 4 日

8:47

SYD8811 开发板正面实物图如下：



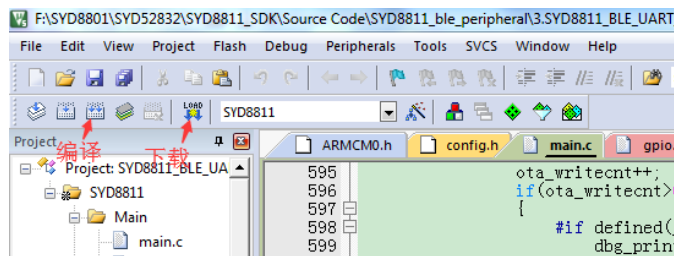
反面实物图：



注意：开发板默认焊接 OLED128X64 的屏幕，但是在 OLED 屏幕下方也留有 TFT0.96 和 TFT1.3 的接口！

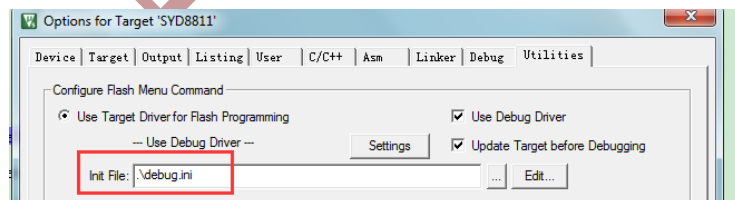
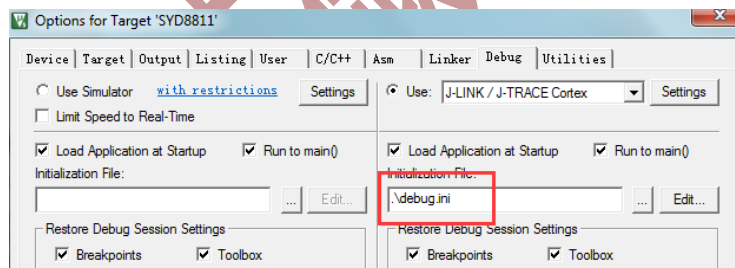
在使用开发板的时候先把电源开关往屏幕这个方向拨动，然后使用 jlink 连接按照《SYDTEK Studio》工具目录下的"SYDTEK Studio release\Documentation\SYD8811 固件烧录方法.pdf"文件烧录“4k_setting”文件（如果没有特定要求可以烧录《SYDTEK Studio》目录下存放的相应文件，比如"SYDTEK Studio release\SYD8811_4ksetting_10pf_LPO32K_2000PPM20190530.bin"）和"ble_service"文件（该文件存放在 KEIL 工程的“profile”文件夹下，比如"SYD8811_SDK\Source Code\SYD8811_ble_peripheral\3.SYD8811_BLE_UART_EVBOLED_notifyen_open_power\profile\SYD_ble_service_Flash.txt"）。

在烧录完“4k_setting”和“ble_service”后既可以在 tool 中直接烧录 KEIL 工程编译出来的 bin 或者 hex 档案（比如"SYD8811_SDK\Source Code\SYD8811_ble_peripheral\3.SYD8811_BLE_UART_EVBOLED_notifyen_open_power\Keil\output\Ble_Vendor_Service.bin"），也可以直接打开 keil 工程编译后下载：

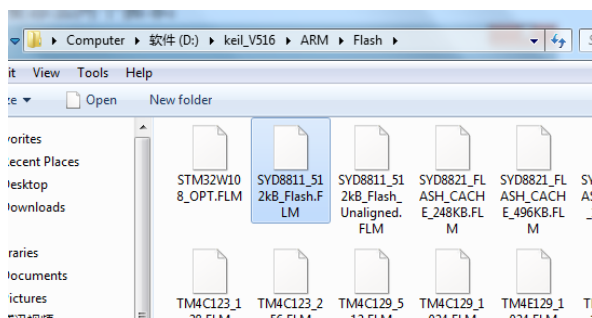


关于 keil 的设置这里有几点细节要说明：

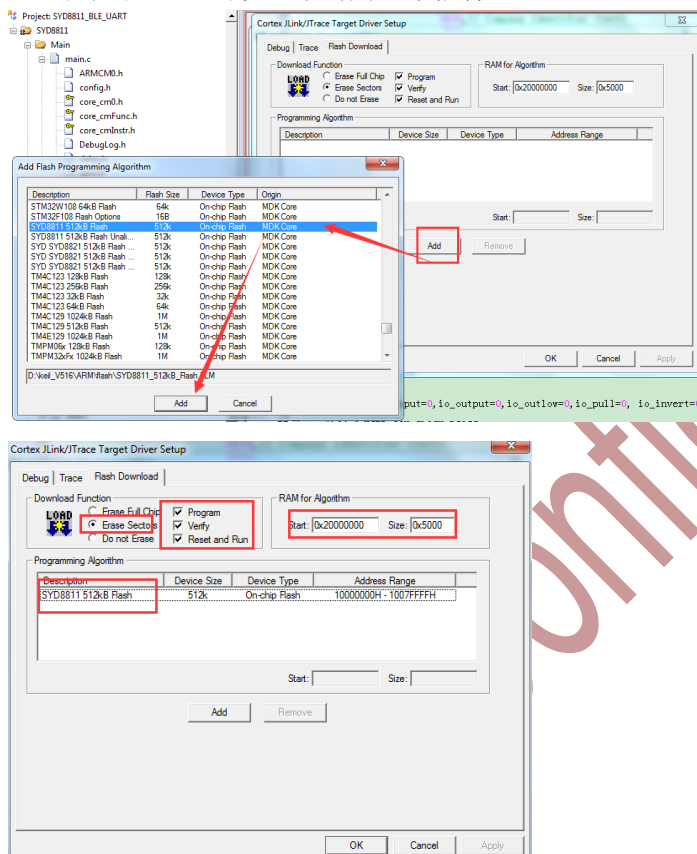
- 1.官方所有工程都是基于 KEIL516 进行编写调试的，所以建议使用 KEIL516 来编译工程
- 2.由于 SWD 复位不能够复位 RF，所以这里要添加两个脚本：



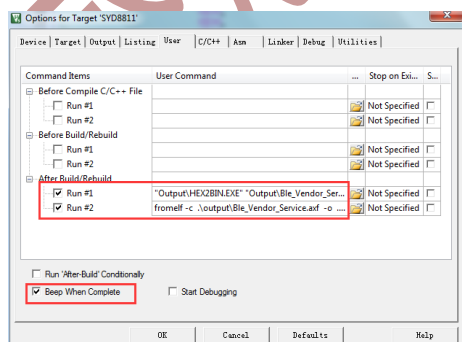
- 3.拷贝 SYDTEK 提供的官方插件（存放在“SYD8811_SDK\Documentation”文件夹下）复制到 KEIL 的 flash 目录：



然后在 KEIL 的设置中引用这个插件:



4. 在 user 中设置在编译完成是生成 bin 档案和反汇编文件:



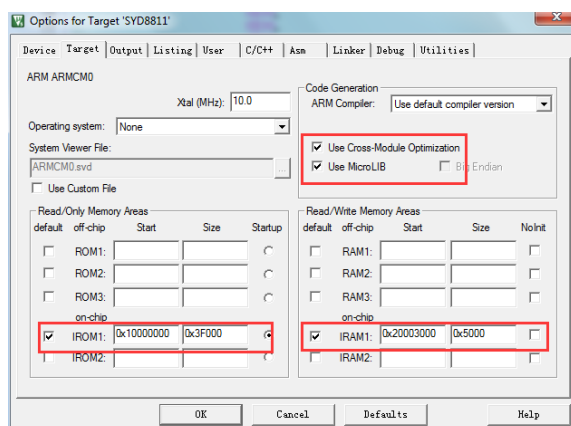
"Output\HEX2BIN.EXE" "Output\Ble_Vendor_Service.hex"

fromelf -c .\output\Ble_Vendor_Service.axf -o .\output\Ble_Vendor_Service_assembly.txt

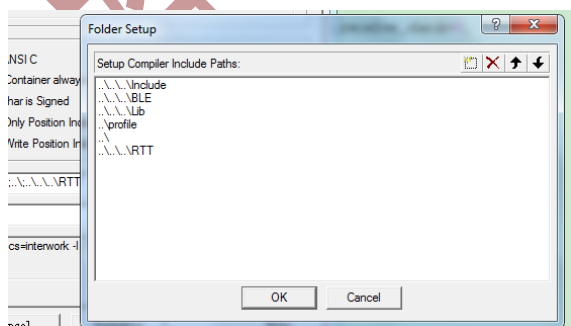
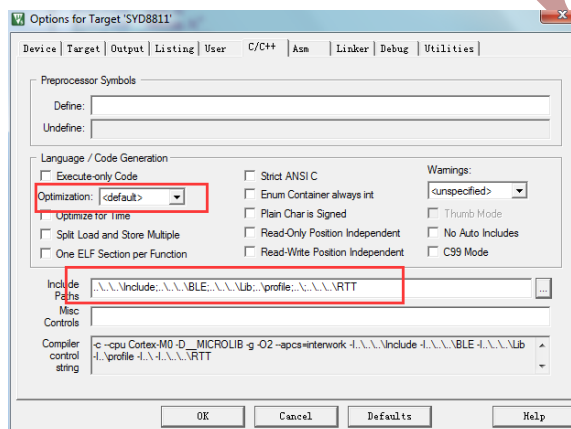
5. 设置 SYD8811 的代码地址 (IROM1) 和内存地址 (IRAM1), 在默认的情况下 SYD8811 的代码空间大小为 252KB, 也就是 0x3F000, cache 映射到 0x10000000, SYD8811 一共有 32KB 的内存,

其中 12KB (0x20000000-0x20003000) 被协议栈占用, 剩下 20KB 给用户代码使用, 但是注意 0x2000074a 开始的 4K 是芯片内部储存当前操作 flash 扇区的缓冲 (写 flash 的操作是先读回整个扇区的数据, 大小为 4K, 然后擦除扇区, 最后修改数据并重新写入, 所以这里要有一个临时数据缓冲区, 在不操作 flash 的时候该缓冲区不会被使用, 所以就可以流出来给用户使用), 该缓冲区可以作为一个特定的内存, 在特定的情况下使用。

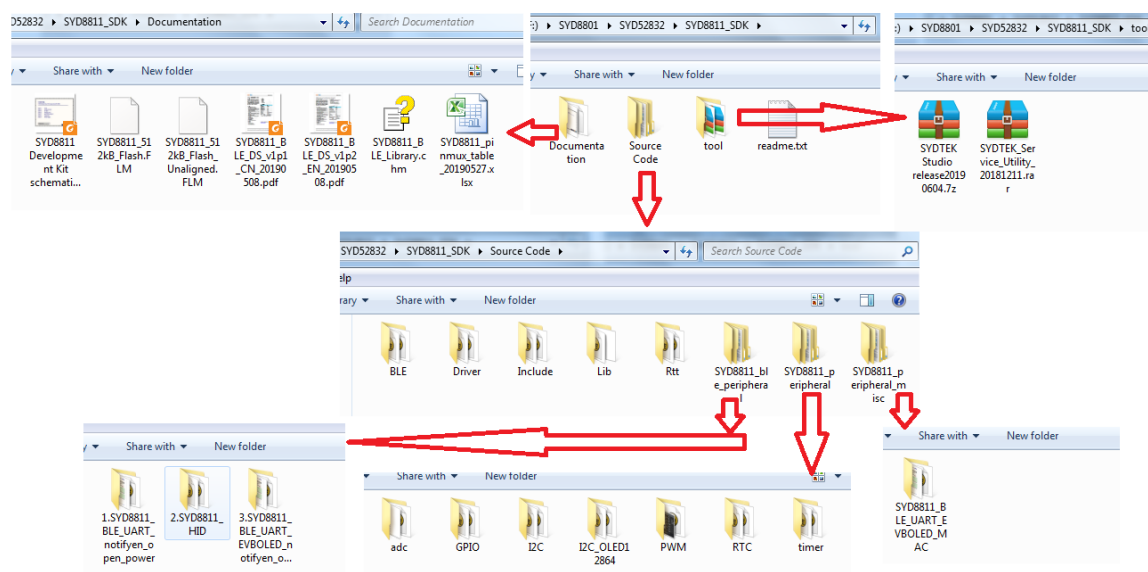
这里勾选上 “Use Cross-Module optimization” 虽然每次编译代码都会进行上次编译, 但是对于有 lib 库的工程而言能够做到最大程度的优化 (没有使用的函数和变量都会被去掉), 同时这里勾选 “Use Microlib”, 这个选项也会缩小代码。



6.C/C++设置项中的优化选项设置为 default, 使用第二级优化, 当然可以尝试使用第 0 级优化, 能够最大减少对代码和内存的使用:



SDK 的框架介绍, SDK 文件框架如下:



所有的驱动 C 文件都放在了“\SYD8811_SDK\Source Code\Source”目录下

所有驱动已经工程需要的头文件都放在了“\SYD8811_SDK\Source Code\Include”目录下

协议栈库文件放在了“\SYD8811_SDK\Source Code\Lib”目录下

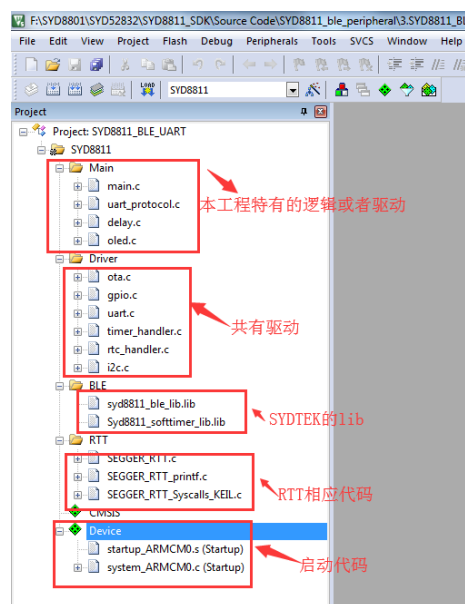
每个驱动工程都会对应的在“\SYD8811_SDK\Source Code\SYD8811_peripheral”目录下拥有一个工程文件夹，打开该文件夹即可看到具体工程的内容，并且该目录下还存放在该工程的 main.c 文件，也就是说所有工程的驱动的 c 文件和头文件都是一样的，不一样的只有工程的设置文件（包括工程文件）和 main.c 文件还有一个 config.h 头文件。

相应的所有的带 BLE 的工程代码在“\SYD8811_SDK\Source Code\SYD8811_ble_peripheral”目录下，每个工程对应一个文件。

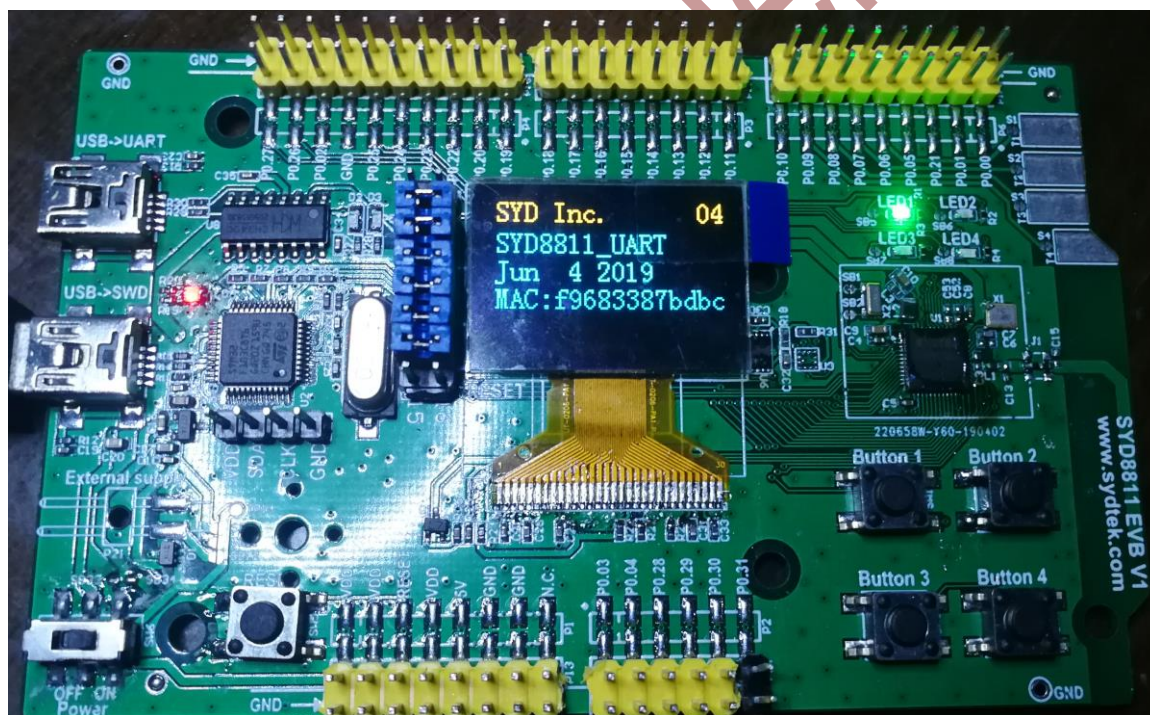
这里打开一个实例工程，比如“SYD8811_SDK\Source Code\SYD8811_ble_peripheral\3.SYD8811_BLE_UART_EVBOLED_notifyen_open_power”，可以看到他的工程结构如下：



双击“\Keil\SYD8811_BLE_UART.uvprojx”可以在 KEIL 中看到如下的工程页面：



关于 KEIL 的下载和仿真这里不再累述，具体可以自行查看相应手册。
编译下载程序运行后开发板的状态是这样的：



SYD8811 程序框架的简单说明：

程序下载完成后通过一系列的初始化操作将会运行到用户代码的 main 函数中，该函数会先进行一系列的初始化流程，然后进入 while (1) 主循环，初始化操作如下：

```

627 int main(void)
628 {
629     uint8_t while_cnt=0;
630     __disable_irq(); //关闭总中断
631
632     ble_init(); //蓝牙初始化，系统主时钟初始化64M, 32K时钟初始化为LPO
633
634     //根据需要重新设置时钟为64M并校准
635     MCUClockSwitch(SYSTEM_CLOCK_64M_RCOSC);
636     RCOSCCalibration(); //对内部64MRCOSC极性校准
637     #ifdef USER_32K_CLOCK_RCOSC
638     ClockSwitch(SYSTEM_32K_CLOCK_RCOSC); //设置内部32.768KHz时钟作为低速定时器看门狗等模块的时钟源
639     CAPEDelayMS(500); //这是内部32KHz晶振的校准函数 经过该函数后定时器能够得到一个比较准确的值
640     LP0Calibration(); //设置外部32.768KHz时钟晶振作为低速定时器看门狗等模块的时钟源
641     #else
642     ClockSwitch(SYSTEM_32K_CLOCK_XOSC); //设置外部32.768KHz时钟晶振作为低速定时器看门狗等模块的时钟源
643     #endif
644
645     #ifdef _SYD_RTT_DEBUG_
646     DebugLogInit(); //初始化RTT
647     #endif
648
649     dbg_printf("Syd8811_UART %s:%s\r\n", __DATE__, __TIME__); //使用RTT进行打印
650
651     //software:tiar0 blelib timer3
652     SYD_Timer_Init(EVT_NUM, syd_sysTimer); //初始化软件定时器，软件定时器使用TIMER0
653     Timer_Evt_List(); //创建软件定时器
654
655     SYD_RTC_Init(RTCEVT_NUM, syd_rtc); //初始化RTC定时器，因为RTC专门是用来做时钟的，定时器的功能是额外的，所以建议间隔很长的定时器采用RTC定时器，比如10 60S 6
656     RTC_Evt_List(); //创建RTC定时器
657
658     gpio_init(); //低功耗时GPIO的初始化设置
659     uart_0_init(); //串口透传功能初始化
660
661     uart_tx_buf.header = 0;
662     uart_tx_buf.tail = 0;
663
664     oled_init(); //OLED初始化
665
666     oled_printf(0,0,"SYD Inc."); //打印
667     oled_printf(0,2,"SYD8811_UART");
668     oled_printf(0,4,"%s",__DATE__);
669     {
670         struct gap_ble_addr dev_addr;
671         /*get bluetooth address */
672         GetDevAddr(&dev_addr);
673         oled_printf(0,6,"MAC:",__DATE__); //打印设备地址
674         oled_printf(0,8,"%02x",dev_addr.addr[0]);
675         oled_printf(0,9,"%02x",dev_addr.addr[1]);
676         oled_printf(0,10,"%02x",dev_addr.addr[2]);
677         oled_printf(0,11,"%02x",dev_addr.addr[3]);
678         oled_printf(0,12,"%02x",dev_addr.addr[4]);
679         oled_printf(0,13,"%02x",dev_addr.addr[5]);
680     }
681
682     GPIO_Set_Output(U32BIT(LED1)); //初始化LED1
683     GPIO_Pin_Set(U32BIT(LED1));
684
685     StartAdv(); //开始广播
686
687     __enable_irq(); //开启总中断
688
689     while(1)
690     {
691         ble_sched_execute(); //协议栈任务
692
693         GPIO_Pin_Turn(U32BIT(LED1)); //翻转LED1
694         oled_printf(14,8,0,"%02x",while_cnt++);
695
696         sendToUart(); //通过串口发送数据
697     }
698 }

```

其中 Timer_Evt_List 和 RTC_Evt_List 函数定义如下：

```

23 void Timer_Evt_List(void)
24 {
25     #ifdef EVT_2S
26     Timer_Evt_Creat(EVT_2S, 2000, Timer_Evt_2s, EVT_DISABLE_MODE); //生成触发事件
27     #endif
28
29     #ifdef EVT_1S_OTA
30     Timer_Evt_Creat(EVT_1S_OTA, 2000, Timer_EVT_1S_OTA, EVT_DISABLE_MODE); //生成触发事件
31     #endif
32 }
33
34 void RTC_Evt_List(void)
35 {
36     #ifdef RTCEVT_whole_minute
37     //该定时器事件比较特殊，是一个整分钟的定时器事件，用于处理整分钟的任务，比如闹钟等，该事件默认设置的长度是60（1分钟）但是当有调整的事件发生的时候，
38     //比如APP更新了本地的事件，那么该定时器由RTC时钟系统把该事件自动调整回整分钟对齐，但是不可避免的造成当前分钟该事件的长度并不是1分钟了，所以该事件并不
39     //适用于1分钟的准时定时器！
40     RTC_EVT_Creat(RTCEVT_whole_minute, 60, RTC_Evt_whole_minute, RTCEVT_DISABLE_MODE);
41     RTC_EVT_whole_minute_setid(RTCEVT_whole_minute);
42     #endif
43
44     #ifdef RTCEVT_10S
45     RTC_EVT_Creat(RTCEVT_10S, 10, RTC_Evt_10S, RTCEVT_DISABLE_MODE); //生成触发事件
46     #endif
47
48     #ifdef RTCEVT_185S
49     RTC_EVT_Creat(RTCEVT_185S, 185, RTC_Evt_185s, RTCEVT_DISABLE_MODE); //生成触发事件
50     #endif
51
52     #ifdef RTCEVT_whole_minute
53     RTC_EVT_Start(RTCEVT_whole_minute);
54     #endif
55
56     #ifdef RTCEVT_10S
57     RTC_EVT_Start(RTCEVT_10S);
58     #endif
59
60     #ifdef RTCEVT_185S
61     RTC_EVT_Start(RTCEVT_185S);
62     #endif
63 }

```

其中 ble_init 函数内容如下：

```

539 static void ble_init()
540 {
541     struct gap_evt_callback evt;
542     struct smp_pairing_req sec_params;
543     struct gap_wakeup_config pw_cfg;
544
545     BleInit(); //调用底层的初始化函数
546     SetWinWideMinusCnt(1); //设置底层窗口参数
547
548     GetGATTReportHandle(&g_report); //获取报告头
549
550     /* security parameters */
551     sec_params.io = IO_NO_INPUT_OUTPUT; //没有输入也没有输出, 将使用passkey的形式配对
552     sec_params.oob = OOB_AUTH_NOT_PRESENT;
553     sec_params.flags = AUTHREQ_BONDING;
554     sec_params.mitm = 0;
555     sec_params.max_enc_sz = 16;
556     sec_params.init_key = 0;
557     sec_params.resp_key = (SMP_KEY_MASTER_IDEN | SMP_KEY_ADDR_INFO);
558     SetSecParams(&sec_params); //设置安全参数
559
560     evt.evt_mask = (GAP_EVT_CONNECTION_SLEEP | GAP_EVT_CONNECTION_INTERVAL); //设置蓝牙事件屏蔽
561     evt.p_callback = ble_evt_callback; //设置蓝牙钩子函数, 当蓝牙底层状态发生改变(比如断线、连线等) 蓝牙协议栈将调用该钩子函数告诉应用程序蓝牙事件变化了
562     SetEvtCallback(&evt); //把钩子的函数和蓝牙事件屏蔽设置给蓝牙协议栈
563
564     /* Bond Manager (MAX:10) */
565     SetBondManagerIndex(0x00);
566
567     setup_adv_data(); //设置广播数据
568
569     /*
570     当POWERDOWN_WAKEUP时,
571     PW_CTRL->DSLP_LPO_EN = true, 这些中断源才能唤醒并复位运行,
572     如果是false就只有pin能唤醒并复位运行
573     */
574     pw_cfg.wakeup_type = SLEEP_WAKEUP;
575     pw_cfg.wdt_wakeup_en = (bool)false;
576     pw_cfg.rtc_wakeup_en = (bool)false;
577     pw_cfg.timer_wakeup_en = (bool)true;
578     pw_cfg.gpi_wakeup_en = (bool)false;
579     pw_cfg.gpi_wakeup_cfg = WAKEUP_PIN; //中断唤醒pin
580     WakeupConfig(&pw_cfg);
581 }

```

当蓝牙事件发生变化的时候蓝牙协议栈将调用 `ble_evt_callback` 函数, 该函数根据蓝牙事件的类型进行不同的处理:

```

401 void ble_evt_callback(struct gap_ble_evt *p_evt)
402 {
403     if(p_evt->evt_code == GAP_EVT_ATT_WRITE)
404     {
405         #ifdef GPIO_LED_CONTROL
406         GPIO_Pin_Turn(U32BIT(GPIO_LED_WRITEING));
407         #endif
408
409         ble_gatt_write(p_evt->evt.att_write_evt);
410     }
411     else if(p_evt->evt_code == GAP_EVT_ATT_READ)
412     {
413         #ifdef GPIO_LED_CONTROL
414         GPIO_Pin_Turn(U32BIT(GPIO_LED_READING));
415         #endif
416
417         ble_gatt_read(p_evt->evt.att_read_evt);
418     }
419     else if(p_evt->evt_code == GAP_EVT_CONNECTED)
420     {
421         connect_flag=1;
422         update_latency_mode=0; //连接状态
423
424         Timer_Evt_Start(EVT_2S);
425
426         #ifdef GPIO_LED_CONTROL
427         GPIO_Pin_Set(U32BIT(GPIO_LED_CONNECT));
428         #endif
429
430         DBGHEXDUMP("Connected addr:", p_evt->evt.bond_dev_evt.addr, sizeof(p_evt->evt.bond_dev_evt.addr));
431     }
432     else if(p_evt->evt_code == GAP_EVT_DISCONNECTED)
433     {
434         DBGPRINTF("Disconnected, reason: 0x%02x\r\n", p_evt->evt.disconnect_evt.reason);
435
436         start_tx = 0;
437         connect_flag=0;
438
439         //clr uart rx fifo
440         uart_rx_buf.header = uart_rx_buf.tail;
441         Timer_Evt_Stop(EVT_1S_OTA);
442         setup_adv_data(); //断开连接之后功耗大10uA
443         StartAdv();
444
445         #ifdef GPIO_LED_CONTROL
446         GPIO_Pin_Clear(U32BIT(GPIO_LED_CONNECT));
447         GPIO_Pin_Clear(U32BIT(GPIO_LED_NOTIFYEN));
448         #endif
449
450         UartEn(false); //不允许RF sleep时关闭IO
451         DBGPRINTF(("start adv @ disconnect\r\n"));
452     }
453     else if(p_evt->evt_code == GAP_EVT_ATT_HANDLE_CONFIGURE)
454     {
455         if(p_evt->evt.att_handle_config_evt.uid == BLE_UART)
456         {
457             if(p_evt->evt.att_handle_config_evt.value == BLE_GATT_NOTIFICATION)

```

到此开发板的说明结束!

关键的提示:

到目前为止SYDTEK已经有3颗芯片在大批量量产, 分别是SYD8801, SYD8821, SYD8811, 所以在芯片和软件以及硬件的说明上, SYD8801是最完善的, SYD8821次之, 但是因为这三颗芯片都是同一个团队开发的, 所以他们在软件框架, 使用方式上基本都一样。

当我们困惑与 SYD8811 的某个技术点想不通的时候可以参考 SYD8801 或者 SYD8821 的资料，这样或许你能够快速的得到你想要的知识，相对于不同的芯片，这里变化的只是函数的名称而已。比如 ble_init 函数的详细说明就可以在这篇博文中得到很理想的解析：

<https://blog.csdn.net/chengdong1314/article/details/76169279>； 另外 ble_evt_callback 函数可以在这篇博文中得到最佳的解析：

<https://blog.csdn.net/chengdong1314/article/details/73929998>

SYDTEK 技术方面的资料可以从微信公众号“SYDTEK 技术交流讨论”开始寻找了解，也可以直接扫描如下二维码关注：



