

# SYDTEK OTA 设备端使用说明

SYDTEK的OTA协议适用于SYDTEK公司旗下所有系列的芯片，这里为了方便说明，就用SYD8801来作为例子。

## 一、简介

SYD8801 设备端使用 A、B 区的方式储存代码，即当前程序是在存储在 A 区，OTA 将新程序写入 B 区，然后重启系统，程序从 B 区开始执行，故中途断开连接或者中断 OTA 不会造成设备“变砖”。A、B 区随着 OTA 的次数相互切换。

设备端大部分功能已经在 ota.h、ota.c 实现，只需实现相应的服务特性读写，以及调用相关 API 即可。

## 二、OTA 升级，设备端需要条件

1、实现服务以及对应的特性读写

<1>服务 UUID 为 FF00

<2>特性 UUID 为 FF01 权限可读、可写



2、keil 工程包含 ota.c，ota.h，并调用相关函数

3、OTA 之前，最好能将蓝牙的连接间隔变小、以便提升 OTA 的速度。

## 三、OTA 实现

第一、实现设备端与 APP 数据接收与返回

```
main.c lib.h ota.c ota.h
986 void ble_evt_callback(struct gap_ble_evt *p_evt)
987 {
988     if(p_evt->evt_code == GAP_EVT_ADV_END)
989     {
990     }
991     else if(p_evt->evt_code == GAP_EVT_CONNECTED) //连接事件
992     {
993     }
994     else if(p_evt->evt_code == GAP_EVT_DISCONNECTED) //断连事件
995     {
996     }
997     else if(p_evt->evt_code == GAP_EVT_ATT_HANDLE_CONFIGURE)
998     {
999     }
1000     else if(p_evt->evt_code == GAP_EVT_ATT_WRITE)
1001     {
1002     #ifdef OTA
1003     if(p_evt->evt.att_write_evt.uuid == BLE_SERVICE_UUID_OTA_READ_WRITE)
1004     {
1005         update_latency_mode=0;
1006         ota_cmd(p_evt->evt.att_write_evt.data, p_evt->evt.att_write_evt.sz);
1007     } else
1008     {
1009         ble_gatt_write(p_evt->evt.att_write_evt);
1010     }
1011     }
1012     else if(p_evt->evt_code == GAP_EVT_ATT_READ)
1013     {
1014     #ifdef OTA
1015     if(p_evt->evt.att_read_evt.uuid == BLE_SERVICE_UUID_OTA_READ_WRITE)
1016     {
1017         uint8_t sz=0;
1018         uint8_t rsp[sizeof(struct gap_ble_evt)]=0;
1019         ota_rsp(rsp, &sz);
1020         SetGATTReadResp(sz, rsp);
1021     } else
1022     {
1023         ble_gatt_read(p_evt->evt.att_read_evt);
1024     }
1025     }
1026     #endif CONFIG_MARCHE_STATE
1027     // march state.state =MARCH_STATE_ATTREAD | MARCHE_STATE_NOTIFY;
1028 }
```

蓝牙协议栈回调

接收APP发来的OTA数据

在OTA.C

APP的OTA读请求返回处理

## 第二、main.c 实现 ota 状态管理函数——ota\_manage

```
main.c lib.h ota.c ota.h
1677 }
1678 #endif
1679 //oled_close_down_init(0);
1680 }
1681 }
1682 }
1683 }
1684 void ota_manage(void){
1685 #ifdef OTA
1686 if(ota_state){
1687     uint8_t ota_callback[2]={0};
1688     switch(ota_state){
1689     case 1 :
1690     #ifdef _DEBUG
1691     dbg_printf("OTA start\r\n");
1692     #endif
1693     break;
1694     case 2 :
1695     #ifdef _DEBUG
1696     dbg_printf("OTA ing\r\n");
1697     #endif //oled_close_down_init(255);
1698     break;
1699     case 3 :
1700     ota_state=0;
1701     #ifdef _DEBUG
1702     dbg_printf("OTA finish\r\n");
1703     #endif
1704     SystemReset();
1705     delay_ms(400);
1706     #ifdef _DEBUG
1707     dbg_printf("Reset failed \r\n");
1708     #endif
1709     break;
1710     default :
1711     break;
1712     }
1713 }
1714 #endif
1715 }
1716 }
```

OTA完成必须软复位或手动复位

## 第三、在 main 函数的 while(1)调用 ota\_manage 函数

```
main.c lib.h ota.c ota.h
1943 wdt_enable(0x400); // 256 * 8 ms = 2*60ms = 2 s
1944 #endif
1945 __enable_irq();
1946 motor_shock_time=10;
1947 while(1)
1948 {
1949     ble_sched_execute();
1950     if(timer1s_inting){ //1s定时器
1951         ota_manage();
1952     }
1953     Timer_Evt_Handle_1s();
1954     #ifdef _WDT_
1955     wdt_clear();
1956     #endif
1957     #ifdef USER_32K_CLOCK_RCOSC
1958     if(SYD 1S EVT&SYD 1S 185S)
1959
```

调用这个OTA管理函数，不一定是1s调用一次，但是最好能在while（1）中

名称	日期	撰写人	版本
SYD8801 OTA 设备端使用说明.PDF	2018 年 3 月 20 日	Bihu	0.1

Version 2.0版本协议补充说明：

上文中所述都是针对Version 1.0版本的协议，为了弥补Version 1.0版本在空升代码量只能达到65536Byte的局限性，这里开出Version 2.0的版本。

不能够升级大于65536Byte的代码主要是因为CMD\_FW\_UPGRADE命令中的” sz” 时候“uint16\_t” 类型的，所以这里增加一个新的命令CMD\_FW\_UPGRADEV20替换原来的CMD\_FW\_UPGRADE命令，这里增加一个结构体：

```
struct cmd_fw_upgrade_V20 {  
    uint32_t    sz;  
    uint16_t    checksum;  
};
```

在参数函数里执行如下函数：

```
case CMD_FW_WRITE:  
    CmdFwWrite(&pcmd->cmdparm.CmdFwWrite);  
    if(!ota_state) ota_state=2;  
    ota_writecnt =0;  
    break;  
case CMD_FW_UPGRADE:  
    CmdFwUpgrade(&pcmd->cmdparm.CmdFwUpgrade);  
    ota_state=3;  
    break;  
case CMD_FW_UPGRADEV20:  
    CmdFwUpgradeV20(&pcmd->cmdparm.CmdFwUpgradeV20);  
    ota_state=3;  
    break;  
case CMD_FW_WRITE_START:  
    .  
    .  
    .  
static void CmdFwUpgradeV20(struct cmd_fw_upgrade_V20 *p_cmd)  
{  
    struct ret_fw_erase_cmd Ret;  
    uint8_t temp = 0;  
  
    if(ota_w.idx != 0)  
        CodeWrite(ota_w.cnt*32, ota_w.idx, ota_w.buf);  
  
    #if defined(_DEBUG_) || defined(_SYD_RTT_DEBUG_)  
    DBGPRINTF("ota sz:%x checksum:%x", p_cmd->sz, p_cmd->checksum);  
    #endif  
    temp = CodeUpdate(NULL, NULL, p_cmd->sz, p_cmd->checksum);  
    #if defined(_DEBUG_) || defined(_SYD_RTT_DEBUG_)  
    DBGPRINTF("CodeUpdatev20=%d\r\n", temp);  
    #endif  
  
    if(temp==0)  
        Ret.status = ERR_COMMAND_FAILED;  
    else  
        Ret.status = ERR_COMMAND_SUCCESS;  
  
    EvtCommandComplete(CMD_FW_UPGRADE, (uint8_t *)&Ret, sizeof (Ret));  
  
    *((uint8_t*)(0x50001000 + 0x24)) = 0x01;  
}
```

Version 3.0版本协议补充说明：

Version 2.0虽然解决了OTA空间大小的问题，但是因为他基于Version 1.0的，所以OTA的速度依旧达不到理想的效果，特别是在SYD8821和SYD8811中代码量至少达到了248KB以上，这时候如果还用原来协议OTA的速度的体验不是很好。

这里改掉CMD\_FW\_WRITE的协议，原来的一个CMD\_FW\_WRITE命令只是发送了15个byte的数据，这里改成了一个命令发送20个byte，同时这里采用分段发送，每个段都会进行校验，如果检验不通过会重发当前段！具体修改如下：

1. 增加CMD\_FW\_WRITE\_START及其流程：

```
#define CMD_FW_WRITE_START      0x14
```

```
struct cmd_fw_write_start {
```

```
    uint32_t    offset;
```

```
    uint16_t    sz;
```

```
    uint16_t    checksum;
```

```
};
```

```
struct ret_fw_write_start_cmd {
```

```
    uint8_t     status;
```

```
    uint16_t    sz;
```

```
    uint16_t    checksum;
```

```
};
```

```
7     break;
8     case CMD_FW_UPGRADEV20:
9         CmdFwUpgradeV20(&pcmd->cmdparm.CmdFwUpgradeV20);
10        ota_state=3;
11        break;
12    case CMD_FW_WRITE_START:
13        ota_section_size=pcmd->cmdparm.CmdFwWriteStart.sz;
14        ota_section_check=pcmd->cmdparm.CmdFwWriteStart.checksum;
15        ota_section_offset=pcmd->cmdparm.CmdFwWriteStart.offset;
16        ota_receive_size=0;
17        ota_receive_check=0;
18        #if defined(DEBUG) || defined(SYD_RTT_DEBUG)
19            dbg_printf("CMD_FW_WRITE_START offset:%x,size:%x checksum:%x\r\n",pcmd->cmdparm.CmdFwWriteStart.
20            #endif
21            break;
```

2. 当该段数据发送完后将发起一次对设备端的读操作，读回的数据包含了设备端计算出来的校验和值。当该段落发送完成的时候则返回该段落的校验和以及大小：

```

174     else
175     {
176         uint16_t idx;
177         ota_writecnt = 0;
178
179         for(idx=0; idx < sz ; idx++)
180             ota_receive_check += p_cmd[idx];
181
182         if(((int)p_cmd % 4)!=0)
183         {
184             memcpy(ota_w_buf, p_cmd, sz);
185             CodeWrite(ota_section_offset+ota_receive_size, sz, ota_w_buf);
186             #if defined(_DEBUG_) || defined(_SYD_RTT_DEBUG_)
187             dbg_printf("CodeWrite noalign ");
188             #endif
189         }
190         else
191         {
192             CodeWrite(ota_section_offset+ota_receive_size, sz, p_cmd);
193             #if defined(_DEBUG_) || defined(_SYD_RTT_DEBUG_)
194             dbg_printf("CodeWrite align ");
195             #endif
196         }
197
198         #if defined(_DEBUG_) || defined(_SYD_RTT_DEBUG_)
199         dbg_printf("section_offset:%x receive_size:%x section_size:%x\r\n", ota_section_offset, ota_receive_size, ota_section_size);
200         #endif
201
202         ota_receive_size +=sz;
203         if(ota_receive_size>ota_section_size)
204         {
205             if(ota_receive_check==ota_section_check)
206             {
207                 CmdFwWriteStart(ERR_COMMAND_SUCCESS, ota_receive_size, ota_receive_check);
208                 #if defined(_DEBUG_) || defined(_SYD_RTT_DEBUG_)
209                 dbg_printf("section OK! ");
210                 #endif
211             }
212             else
213             {
214                 CmdFwWriteStart(ERR_COMMAND_FAILED, ota_receive_size, ota_receive_check);
215                 #if defined(_DEBUG_) || defined(_SYD_RTT_DEBUG_)
216                 dbg_printf("section fail! ");
217                 #endif
218             }
219             #if defined(_DEBUG_) || defined(_SYD_RTT_DEBUG_)
220             dbg_printf("ota_receive_check:%x ota_section_check:%x\r\n", ota_receive_check, ota_section_check);
221             #endif
222
223             ota_variable_clear(false);
224         }
225     }
226 }
227

```