SDK 开发初级指南

V1.0.2

北京联盛德微电子有限责任公司（winner micro）

地址：北京市海淀区上园村 3 号交大知行大厦七层

电话：+86-10-62161900

公司网址：www.winnermicro.com

文档历史

版本	完成日期	修订记录	作者	审核	批准
V1.0.0	2015-7-15	创建	章鹏飞		
V1.0.1	2016-6-14	添加 APSTA 的应用和说明	章鹏飞		
V1.0.2	2016-7-20	添加 AT+FWUP 的 HTTP 升级以及代码	章鹏飞		

目录

1 概述

本文档主要是指导客户如何使用北京联盛德微电子有限责任公司(简称 winnermicro)的 SDK，添加客户定制的代码，实现一些客户定制化的功能。

2 从 main 开始

Winnermicro 官网下载 SDK,打开 winnermicro 的 SDK,整个工程有 400 多个文件，包含了 wifi 芯片的所有驱动和简单 DEMO 实现。Winnermicro 提供了大而全的 SDK，有 400 多个文件。在 C 语言中，一切程序从 main 函数开始。用 Source Insight 建立工程，搜索 main 函数可知，在文件..\WM_SDK\APP\main.c 中，如下：

```
/**************************************************************
* File Name : main.c
* Description: main
* Copyright (c) 2014 Winner Micro Electronic Design Co., Ltd.
* All rights reserved.
* Author : dave
* Date : 2014-6-14
**************************************************************/
#include "wm_include.h"
 extern tls_os_sem_t     *libc_sem;
int main(void)
{
#if TLS_OS_UCOS
    tls_main();
    OSStart();
#elif TLS_OS_FREERTOS
    tls_irq_init();
    tls_os_sem_create(&libc_sem, 1);
```

```
        tls_main();
        vTaskStartScheduler();
    #endif
        return 0;
    }
```

在 main 函数中，有宏定义#if…#elif…#endif。其中 TLS_OS_UCOS 则是针对 UCOS-II 而 TLS_OS_FREERTOS 则是针对 FREERTOS，从此处可知 winnermicro 的 SDK 支持两种操作系统，分别是 UCOS-II 和 FREERTOS。如果熟悉这两种操作系统，或者知道其中一种的，都知道除了 tls_main()函数，其他的都是操作系统启动函数。而 tls_main()函数正是整个 SDK 的主函数。

```
/***************tls_main 函数***************/
int tls_main(void)
{
#if TLS_OS_UCOS
    /* Initialize uC/OS-II    */
    OSInit();
    /* before use malloc() function, must create mutex used by c_lib */
    tls_os_sem_create(&libc_sem, 1);
#endif
    tls_os_task_create(NULL, NULL,
                       task_start,
                       (void *)0,
                       (void *)TaskStartStk,          /* 任务栈的起始地址 */
                       TASK_START_STK_SIZE * sizeof(u32), /* 任务栈的大小      */
                       1,
                       0);
    return 0;
```

```
}
```

在 tls_main 函数中对 UCOS-II 系统进行再次初始化之外，就是建立了一个任务 task_start。

此时用户可以从 tls_main 函数知道，winnermicro 的 SDK 是如何的建立任务的（注意优先级设置）。

```
/*****************************************************************************
*
 * Function Name         // task_start
 *Descriptor              // before create multi_task, we create a task_start task
// in this example, this task display the cpu usage
 * Input
 * Output
 * Return

*****************************************************************************/
void task_start (void *data)
{
    extern void CreateUserTask(void);
    extern void  RestoreParamToDefault(void);
   ….
    /* User start here */
    CreateUserTask();
    for (;;)
    {
#if 1
    tls_os_time_delay(0x10000000);
#else
    tls_os_time_delay(1000);
    disp_task_stat_info();
```

```
#endif
    }
}
```

在 task_start()函数中大致扫一遍，可以看出此函数主要是对 winnermicro 芯片的 WIFI 功能和所有接口初始化。最后进入了一个 for(;;)死循环。

在死循环前有函数 CreateUserTask();同时在这个函数上面还有一句注释/* User start here */。

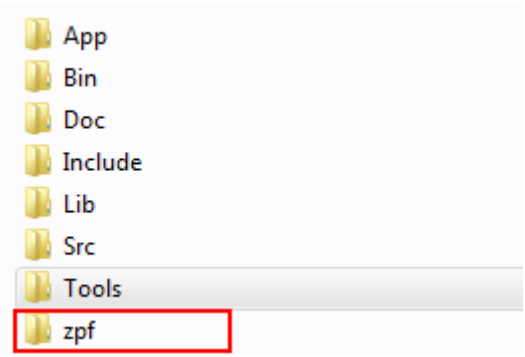
```
/*************** CreateUserTask 函数******************/
void CreateUserTask(void)
{
        printf("\n user task\n");
    //用户自己的 task
}
```

此时通过源码可知，这个函数接口是专门给二次开发用户使用的。那么用户可以 printf("\n user task\n");之后添加函数:UserMain();//当然可以自定义任意名称

那么如何定义 UserMain()，又在哪一个文件中进行定义呢？

3 User 文件和 User Main

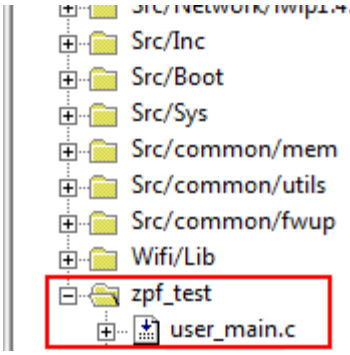
User 文件和 winnermicro 的 SDK 区分开来，最好新建一个文件夹存放【一般以工程的英文名称】，实例参考如下：



在 User 文件夹中新建源文件 user_main.c 和 user_main.h。然后在 user_main.c 中包含 user_main.h 文件，如下图：

```
user_main.c
1   #include "user_main.h"
```

那么 winnermicro 的 SDK 工程就转为 User 工程。



按照上述步骤去实现的好处是 winnermicro 的 SDK 升级或者变动，如果自己需要同步，那么只需要在 printf("\n user task\n");之后添加函数:UserMain();然后将 User 文件夹复制，添加到工程就 OK 了。而且这样操作也利于原厂技术支持对接，更快找到 user bug。因为 winnermicro 的 SDK 是没有任何变动的。仅仅是添加了一句话 UserMain();。

4 User 工程的框架

4.1 User 框架搭建

在 user_main.c 中定义 User Main 函数，在 UserMain 函数中创建 USER TASK。

```
/**************************************************************
* Description: user main
* Auth: zpf
*Date: 2015-7-15
**************************************************************/
void UserMain (void)
{
    …
    //加网（对于 Wi-Fi 模块来说，首先是加网处理，加网成功了，才能进行通信处理，当然前期必然是参数配置，所以加网前有三个点！省略用户具体参数配置）。
    CreateUserTask();// create user task
}

/**************************************************************
```

```
* Description: create user task
* Auth: zpf
*Date: 2015-7-15
*****************************************************************/
static int CreateUserTask(void)
/*static 限制了 CreateUserTask 范围在 user_main.c 中*/
{
    tls_os_queue_create(&gsUserTaskQueue,&UserTaskQueue,USER_QUEUE_SIZE,  0);/*
创建用户消息*/
    tls_os_task_create(NULL, NULL, UserTaskProc, NULL,
                       (void *)UserTaskStk,          /* 任务栈的起始地址*/
                       USER_TASK_SIZE * sizeof(u32),  /*任务栈的大小*/
                       50,
/*注意优先级，ucos-ii 64 个优先级，SDK 占用数十个任务，一般建议优先级从 45 之后，到 60。*/
                       0);
    return WM_SUCCESS;
}
/*****************************************************************
* Description: 用户任务函数
* Auth: zpf
*Date: 2015-7-15
*****************************************************************/
static void UserTaskProc(void)
{
    void *msg;
    …
/* 配置用户串口，将串口 1 设置波特率为 115200。*/
    tls_user_uart_set_baud_rate(115200);
    tls_uart_cfg_user_mode();
    tls_user_uart_rx_register(user_uart_rx);
    tls_netif_add_status_event(UserWlanStatusChangedEvent); /* 注册网络状态回
调函数，用来获取 Wi-Fi 模块的加网断网等情况。*/
```

```
for(;;)
{
    tls_os_queue_receive(gsUserTaskQueue, (void **)&msg, 0, 0);
    switch((u32)msg)
    {
        case MSG_NET_UP:                /* 加网成功*/
            …
            break;
        case MSG_NET_DOWN:                /*网络断开*/
            ….
            break;
        case MSG_ONESHOT:                /*启动一键配置*/
            printf("start oneshot…\r\n");
            tls_wifi_set_oneshot_flag(1);
            break;
        case MSG_SOCKET_ERR:            /*socket 断开*/
            ….
            break;
        case MSG_SOCKET_RECEIVE_DATA:    /*TCP client socket 收到数据*/
            …
            break;
        case MSG_SK_SERVER_RX_DATA: /*TCP server 接收的数据处理消息*/
            …
            break;
        case MSG_UDP_RECEIVE_DATA:/*处理从 UDP 发送过来的数据*/
            …
            break;
        case MSG_UART_RECEIVE_ALL_DATA:/*处理从串口接收的数据*/
            …
            break;
        case MSG_SK_SERVER_CONNECT:
            …
```

```
                break;
            case MSG_HTTP_FWUP:/*http server 升级固件*/
                ...
                break;
            default:
                break;
        }
    }
}

/*****************************************************************
* Description: 网络状态变化回调函数
* Auth: houxf
*Date: 2015-7-15
*****************************************************************/
static void UserWlanStatusChangedEvent(INT8U status)
{
    switch(status)
    {
        case NETIF_WIFI_JOIN_SUCCESS:
            printf("netif_wifi_join_success\r\n");
            break;
        case NETIF_WIFI_JOIN_FAILED:
            printf("netif_wifi_join_failed\r\n");
            break;
        case NETIF_WIFI_DISCONNECTED:
            printf("netif_wifi_disconnected\r\n");
            tls_os_queue_send(gsUserTaskQueue,(void *)MSG_NET_DOWN,0);
            break;
        case NETIF_IP_NET_UP:
            printf("netif_ip_net_up\r\n");
            tls_os_queue_send(gsUserTaskQueue, (void *)MSG_NET_UP, 0);
```

```
            break;
        default:
            break;
    }
}
```

在 UserTaskProc 函数中，通过消息机制对 TCP 通信，UDP 通信，串口通信，一键配置，网络升级分别进行处理。这样就基本上 winnermicro 的模块的框架基本搭建，也完成了 TLN13SP01 模块的 Wi-Fi 通信、UART 的应用。

UserWlanStatusChangedEvent 函数，是 WIFI 芯片特有的，通过扫描底层的寄存器，将模块网络状态通过消息机制发送给任务，调用对应的处理函数。

Winnermicro 的 SDK 从 V15 版本开始支持京东云、KII 云，以及后续会支持 QQ 物联，AIRKISS 等更多的云平台。具体这些平台的搭建，详见官网的各个云平台文档介绍。

4.2 模块设置为 AP

```
/*************************************************************
* Description:
* Auth:
*Date: 2015-7-15
*************************************************************/
void UserApMode(char *ssid)
{
    INT8S autoconnect;
    Unsigned char  temp;
    struct tls_param_ip param_ip;
    unsigned int params;
    struct tls_param_ssid params_ssid;
    struct tls_cmd_key_t params_key;

    tls_wifi_auto_connect_flag(WIFI_AUTO_CNT_FLAG_GET, &autoconnect);
    if(WIFI_AUTO_CNT_ON == autoconnect)
    {
        autoconnect = WIFI_AUTO_CNT_OFF;
        tls_wifi_auto_connect_flag(WIFI_AUTO_CNT_FLAG_SET, &autoconnect);
```

```
    }
//关闭自动联网

    temp = IEEE80211_MODE_AP;
    tls_param_set(TLS_PARAM_ID_WPROTOCOL, (void *)&temp, 0);
    params_ssid.ssid_len = strlen(ssid);
    memcpy(&params_ssid.ssid, ssid, params_ssid.ssid_len);
    tls_param_set(TLS_PARAM_ID_SSID, (void *)&params_ssid, 0);

    temp = IEEE80211_ENCRYT_NONE;
    tls_param_set(TLS_PARAM_ID_ENCRY, (void *)&temp, 0);

    params_key.format = 1;
    params_key.index = 0;
    params_key.key_len = 0;
    memset(&params_key.key, 0, 64);
    tls_cmd_set_key(&params_key, 0);

    param_ip.dhcp_enable = 0;
    string_to_ipaddr("192.168.1.1", (u8 *)&params);
    memcpy((char *)param_ip.dns1, (u8 *)&params, 4);
    memcpy((char *)param_ip.dns2, (u8 *)&params, 4);
    memcpy((char *)param_ip.gateway, (u8 *)&params, 4);
    memcpy((char *)param_ip.ip, (u8 *)&params, 4);
    string_to_ipaddr("255.255.255.0", (u8 *)&params);
    memcpy((char *)param_ip.netmask, (u8 *)&params, 4);
    tls_param_set(TLS_PARAM_ID_IP, (void *)&param_ip, 1);
    tls_param_to_flash(TLS_PARAM_ID_ALL);
    tls_sys_reset();//注意，是重启 API，调用此函数就是重启模式，
//如果通过 X DOWNLOAD 下载，则重启则固件消失。
}
/*
```

*此函数只是配置了模块的 AP 模式的参数，并未进行加网操作。

*请注意。此函数调用完成模块重启完成，跳过此配置函数【否则进入无限重启】，通过读取模块是 STA 还是 AP 进行加网操作完成加网

*/

4.3 模块配置为 STA

```
/*****************************************************************************
* Description:
* Auth: zpf
*Date: 2015-7-15
*****************************************************************************/
void UserStaMode(char *ssid, char *key)
{
    INT8S autoconnect;
    u8 temp;
    struct tls_param_ip param_ip;
    u32 params;
    struct tls_param_ssid params_ssid;
    struct tls_cmd_key_t params_key;

    tls_wifi_auto_connect_flag(WIFI_AUTO_CNT_FLAG_GET, &autoconnect);
    if(WIFI_AUTO_CNT_OFF == autoconnect)
    {
        autoconnect = WIFI_AUTO_CNT_ON;
        tls_wifi_auto_connect_flag(WIFI_AUTO_CNT_FLAG_SET, &autoconnect);
    }/*打开自动联网功能*/

    temp = IEEE80211_MODE_INFRA;
    tls_param_set(TLS_PARAM_ID_WPROTOCOL, (void *)&temp, 0);
/*设置模块的工作模式，类似 AT+WPRT=0*/

    params_ssid.ssid_len = strlen(ssid);
    memcpy(&params_ssid.ssid, ssid, params_ssid.ssid_len);
```

```
    tls_param_set(TLS_PARAM_ID_SSID, (void *)&params_ssid, 0);

    params_key.format = 1;
    params_key.index = 0;
    params_key.key_len = strlen(key);
    memcpy(&params_key.key, key, params_key.key_len);
    tls_cmd_set_key(&params_key, 0);/*设置模块的密码*/

    param_ip.dhcp_enable = 1; /*打开 DHCP*/
    string_to_ipaddr("192.168.1.1", (u8 *)&params);
    memcpy((char *)param_ip.dns1, (u8 *)&params, 4);
    memcpy((char *)param_ip.dns2, (u8 *)&params, 4);
    memcpy((char *)param_ip.gateway, (u8 *)&params, 4);
    memcpy((char *)param_ip.ip, (u8 *)&params, 4);
    string_to_ipaddr("255.255.255.0", (u8 *)&params);
    memcpy((char *)param_ip.netmask, (u8 *)&params, 4);
    tls_param_set(TLS_PARAM_ID_IP, (void *)&param_ip, 1);
    tls_param_to_flash(TLS_PARAM_ID_ALL);
    tls_sys_reset();/*模块重启函数，1 秒重启完成*/
}
/*
*此函数只是配置了模块的 STA 加网的参数，并未进行加网操作。
*请注意。此函数调用完成模块重启完成，跳过此配置函数【否则进入无限重启】，通过读取模块是 STA 还是 AP 进行加网操作完成加网
*/
```


4.4 模块配置为 APSTA 模式【v25 以上的 SDK 版本】

在北京联盛德微电子有限责任公司的SDK的WM_SDK_G1.02.25【需要用户名密码，具体可联系地区办事处】中，添加了一项新的功能APSTA。也就说SDK必须是25版本以上，才会有此功能。

何谓APSTA，就是一块联盛德Wi-Fi模块，同时开启APSTA功能。即联盛德Wi-Fi模块即能连接路由，同时其他终端设备也可以连接此Wi-Fi模块。如果路由可以连接网络，那么连接联盛德Wi-Fi模块的终端设备，通过联盛德Wi-Fi模块中转，连接路由，也可以达到连

接网络的目的。

APSTA的优点：

1、增加路由连接终端设备的数量。

2、实现热点信号的延长。

APSTA如何开启：

1、替换WM_SDK_G1.02.25中的库WiFi.lib为STA_AP_APSTA文件夹下的WiFi.lib

2、Wm_config.h文件中将宏定义

#define TLS_CONFIG_APSTA (CFG_ON && TLS_CONFIG_AP) //打开APSTA功能

备注：在 wm_wifi_oneshot.h 中将宏 CONFIG_NORMAL_MODE_ONESHOT 定义为 0，然后编译出的就是带有 apsta 一键配置功能的版本。

APSTA 对应的使用 API 函数：

```
/*****************************************
* Description:
* Auth: zpf
*Date: 2016-6-14
*****************************************/
void UserApStaMode(char *ssid, char *key,char *ApStaSsid)
{
    INT8S autoconnect;
    u8 temp;
    u8 *ApStaName="ApSta_Scinan";
    struct tls_param_ip param_ip;
    u32 params;
    struct tls_param_ssid params_ssid;
    struct tls_param_ssid params_ssid2;
    struct tls_cmd_key_t params_key;

    tls_wifi_auto_connect_flag(WIFI_AUTO_CNT_FLAG_GET, &autoconnect);
    if(WIFI_AUTO_CNT_OFF == autoconnect)
    {
        autoconnect = WIFI_AUTO_CNT_ON;
        tls_wifi_auto_connect_flag(WIFI_AUTO_CNT_FLAG_SET, &autoconnect);
```

```
    }

    temp = IEEE80211_MODE_APSTA;
    tls_param_set(TLS_PARAM_ID_WPROTOCOL, (void *)&temp, 0);

    params_ssid.ssid_len = strlen(ssid);
    memcpy(&params_ssid.ssid, ssid, params_ssid.ssid_len);
    tls_param_set(TLS_PARAM_ID_SSID, (void *)&params_ssid, 0);

    params_ssid2.ssid_len = strlen(ApStaSsid);
    memcpy(&params_ssid2.ssid, ApStaSsid, params_ssid2.ssid_len);
    tls_param_set(TLS_PARAM_ID_APSTA_SSID, (void *)&params_ssid2, 0);

    params_key.format = 1;
    params_key.index = 0;
    params_key.key_len = strlen(key);
    memcpy(&params_key.key, key, params_key.key_len);
    tls_cmd_set_key(&params_key, 0);

    param_ip.dhcp_enable = 1;
    string_to_ipaddr("192.168.1.1", (u8 *)&params);
    memcpy((char *)param_ip.dns1, (u8 *)&params, 4);
    memcpy((char *)param_ip.dns2, (u8 *)&params, 4);
    memcpy((char *)param_ip.gateway, (u8 *)&params, 4);
    memcpy((char *)param_ip.ip, (u8 *)&params, 4);
    string_to_ipaddr("255.255.255.0", (u8 *)&params);
    memcpy((char *)param_ip.netmask, (u8 *)&params, 4);
    tls_param_set(TLS_PARAM_ID_IP, (void *)&param_ip, 1);
    tls_param_to_flash(TLS_PARAM_ID_ALL);
    tls_sys_reset();
}
/*
```

*此函数只是配置了模块的 APSTA 模式的参数，并未进行加网操作。
*请注意。此函数调用完成模块重启完成，跳过此配置函数【否则进入无限重启】，通过读取模块是 STA、AP 还是 APSTA 进行加网操作完成加网
*/

```
/*
*下面例子是一个对 AP,STA,APSTA 模式进行调用，进行参数赋值，并没有进行加网操作。
*调用了【static void UserTestMode(void)】必定导致模块重启，加网动作只能在重启后完成。
*建议通过网络状态函数【static void UserWlanStatusChangedEvent(INT8U status)】,
*查看模块是否在网，或者查询模块的在网的 SSID，或者查看模块在网的 AP 的 MAC 是否和
*自己匹配，或者通过【tls_param_get(TLS_PARAM_ID_WPROTOCOL, (void*)&mode, FALSE)】
*判断 Wi-Fi 模块的模式是否是所想要的模式，进而进行下一步操作。
*具体细节 user thinking
*/
/*****************************************************************
* Description:
* Auth: houxf
*Date: 2015-3-31
*****************************************************************/
static void UserTestMode(void)
{
    INT8U mode;
    u8 *TestSsid="scinan";
    u8 *TestKey="scinan88";
    u8 *ApStaName="ApStaTest";

    tls_param_get(TLS_PARAM_ID_WPROTOCOL, (void* )&mode, FALSE);
    if(IEEE80211_MODE_AP == mode)
    {
        UserStaMode("kevin", "12345678");
    }
```

```
    else if(IEEE80211_MODE_APSTA== mode)
    {
       UserApStaMode(TestSsid, TestKey, ApStaName);
    }
    else
    {
        UserApMode("kevin_ap");
    }
}
```

5 模块加网

5.1 加载配置参数之后-加网

int tls_cmd_create_net(void);

//只适用于 AP 配置完成之后的加网，如 UserApMode(Ssid)之后，此函数加网原理则是读取模块的 FLASH 中的参数进行加网操作【所以加网参数必须齐全】，具体可参见 wm_cmdp.c 中的源码。

static int tls_cmd_create_apsta_net(void)

////只适用于 APSTA 配置完成之后的加网，此函数加网原理则是读取模块的 FLASH 中的参数进行加网操作【所以加网参数必须齐全】。如 UserApStaMode(Ssid, Key, ApStaSsid)之后;，具体可参见 wm_cmdp.c 中的源码。

int tls_cmd_join(enum tls_cmd_mode mode, struct tls_cmd_connect_t *conn);

//适用于工作模式，加网参数配置等参数都配置的加网，适用 AP 创建、STA 加网、APSTA 加网。具体可参见 wm_cmdp.c 中的源码。

5.2 自动联网

当模块完成了 STA 加网的配置，或者 AP 创网的配置之后，可在 void UserMain (void)函数中，直接打开自动加网功能。模块执行 void UserMain (void)函数时，检测自动联网标志被置 1 了，会直接启动联网功能，加载已经配置完成的联网参数，该函数主要特点是即使联网失败了，也会再次进行联网。具体可参看 wm_wifi.h 中对 int tls_wifi_auto_connect_flag(u8 opt, u8* mode)函数的描述。

/***

```
* Description: UserMain//用户二次开发的起始函数
* Auth: zpf
*Date: 2015-7-15
*********************************************************************************/
void UserMain (void)
{
    …
    INT8S autoconnect;
// check autoconnect
    tls_wifi_auto_connect_flag(WIFI_AUTO_CNT_FLAG_GET, &autoconnect);
    if(WIFI_AUTO_CNT_OFF == autoconnect)
    {
        autoconnect = WIFI_AUTO_CNT_ON;
        tls_wifi_auto_connect_flag(WIFI_AUTO_CNT_FLAG_SET, &autoconnect);
    }
    …
    CreateUserTask();// create user task

}
```

5.3 三种模式的联网 API【APSTA 加网、AP 创建、STA 加网】

在 wm_wifi.h 文件中，有 API 函数：

```
int tls_wifi_connect(u8 *ssid, u8 ssid_len, u8 *pwd, u8 pwd_len);
```

此函数的可以通过设置 SSID 和 PWD，进行联网。

For example:

```
unsigned char user_ssid[32] = “lsd”;
Unsigned char user_pwd[64] = “winnermicro”;

Int ssid_len = strlen(“lsd”);
Int pwd_len = strlen(“winnermicro”);

tls_wifi_connect(user_ssid, ssid_len, user_pwd,  pwd_len);
```

```
/*
*说明：此函数就是一个 STA 联网动作。为保证此函数执行顺利的前提，必须先检查模块*
的模式是否 STA【如不是，重新配置】，同时检查 DHCP 是关闭状态【否则容易出现路由分*
配 IP 冲突，具体可以参考终端设置静态 IP，必须是热点同一局域网内，】。
*/
```

APSTA 模式的加网：

```
unsigned char user_ssid[32] = “lsd”;
Unsigned char user_pwd[64] = “winnermicro”;
unsigned char user_apsta_ssid[32] =” lsd_apsta”

int ssid_len = strlen(“lsd”);
int pwd_len = strlen(“winnermicro”);
int apsta_ssid_len =strlen(“lsd_apsta”);

tls_wifi_apsta_start(user_ssid,strlen(user_ssid),user_pwd,strlen(user_pwd),us
er_apsta_ssid,strlen(apsta_ssid_len));

/*
说明：此函数是 APSTA 模式下的联网操作，类似 STA 联网动作。为保证此函数执行顺利
的前提，必须先检查模块的模式是否 APSTA【如不是，重新配置】，同时检查 DHCP 是关闭
状态【否则容易出现路由分配 IP 冲突，具体可以参考终端设置静态 IP，必须是热点同一
局域网内，同时不能和热点下的其他终端有冲突】。
*/
```

6 用户参数存储

在 wm_config.h 中有一行如下描述：

```
#define TLS_FLASH_PARAM_DEFAULT  (0x00099000)
//(0x00002000),99000 这个区间没有使用，存放用户参数
```

由于模块是外挂 FLASH 共 1MB，于是可在 user_main.c 中，用户可以宏定义

```
#define FLASH _ADDR_UAER_START      (0x00099000+1)
#define FLASH _ADDR_UAER_END        (0x00100000)
```

FLASH 读写函数可参见 wm_flash.h 中的

```
int tls_fls_read(u32 addr, u8 *buf, u32 len);
int tls_fls_write(u32 addr, u8 *buf, u32 len);
```

FOR EXAMPLE：

```
u8 set_uart_one_init = 0;
tls_fls_read(FLASH _ADDR_UAER_END -1, &set_uart_one_init, 1);
tls_fls_write(FLASH _ADDR_UAER_END -1, &set_uart_one_init, 1);
```

说明：为什么例子会从 FLASH _ADDR_UAER_END 开始呢，逐步减呢？因为我司的 SDK 会不断更新，FLASH 使用会不断的增加。如果直接从 FLASH _ADDR_UAER_START 开始，那么 SDK 一旦增大，则用户使用 FLASH 的地址就会和 SDK 产生冲突。从 FLASH 的最后往前，则用户使用 FLASH 的地址就会和 SDK 产生冲突大大减小，除非两者使用的 FLASH 大于 1MB。

7 GPIO 口

对 GPIO 口的操作，参见 wm_gpio.c 源文件。

由于我司的 GPIO 口和其他管脚存在复用现象，那么在对 GPIO 口进行拉高拉低等操作之前，需要先进行设置。

FOR EXAMPLE:

拉高拉低 GPIO 口：

```
u16 gpio_pin = 11;//具体的管脚需要参考对应的模块管脚列表
u16 ret;
tls_gpio_cfg(gpio_pin, TLS_GPIO_DIR_INPUT, TLS_GPIO_ATTR_FLOATING);
ret = tls_gpio_read(gpio_pin); /*先读默认状态*/
printf("\ngpio[%d] default value==[%d]\n",gpio_pin,ret);

tls_gpio_cfg(gpio_pin, TLS_GPIO_DIR_OUTPUT, TLS_GPIO_ATTR_FLOATING);
tls_gpio_write(gpio_pin,1);          /*写高*/
ret = tls_gpio_read(gpio_pin);
printf("\ngpio[%d] floating high value==[%d]\n",gpio_pin,ret);

tls_gpio_cfg(gpio_pin, TLS_GPIO_DIR_OUTPUT, TLS_GPIO_ATTR_FLOATING);
tls_gpio_write(gpio_pin,0);          /*写低*/
ret = tls_gpio_read(gpio_pin);
printf("\ngpio[%d] floating low value==[%d]\n",gpio_pin,ret);
```

另外 GPIO 口中断脚的操作，请参见 wm_gpio_demo.c 中的 int gpio_isr_test(char *buf)函数。

8 ADC 口简略

芯片的 ADC 口一般情况下为 4 路，具体参见对应模块的说明。

ADC 的实例请参见 wm_adc_demo.c

ADC 检测电压范围为 0~3.3V，对应数据为 0~0x800

9 I2C 简略

I2C 实例参见 wm_i2c_demo.c

注意一般情况下，模块的 I2C 接口为主机（暂无从机驱动）

10 PWM 简略

SDK 中提供 PWM 驱动，是通过 GPIO 口模拟的。

在 wm_pwm.c 中是以 GPIO11，12，13，0 四口进行模拟的。用户可自行设置自己的 GPIO 口实现 PWM。

在 wm_pwm.h 中可对 PWM 进行设置，具体参数设置如下：

```
#define PWM_CHANNEL_MAX_NUM 4
#define PWM_MAX_FREQ      400
//单路 pwm 最大频率，如果 DEPTH 调低，该频率可适当调高
#define PWM_DEPTH 250      //最大可调级别，可根据需要调整
#define PWM_1S 1000000
#define PWM_BASE_TIME_COUNT (PWM_1S/PWM_MAX_FREQ/PWM_DEPTH)
```

源文件 wm_pwm_demo.c 是对 PWM 的一个简单实例，用户可参考。

11 定时

系统定时：

通过 SDK 中的操作系统中的定时函数进行定时（10ms 单位），具体可参见此函数

```
static __inline tls_os_status_t
       tls_os_timer_create(
                           tls_os_timer_t **timer,
                           TLS_OS_TIMER_CALLBACK callback,
                           void *callback_arg,u32 period,
```

```
                        bool repeat,
                        u8 *name
                        )
```

硬定时(定时单位为 1us，高精度定时使用):

```
static int temp_i =0;
static void UserIRQOneTimeCallback(void)
{
        tls_timer_clear_irq();
        if(100 == temp_i)
        {
            printf("test!\r\n");
            temp_i = 0;
        }
        temp_i ++;
}
void UserHardwareIRQOne (void)
{
        tls_timer_irq_register(UserIRQOneTimeCallback);
        tls_timer_start(1000); //1000*1us
}
```

12 SOCKET 通信

SDK 中存在两种封装的 SOCKET，一种是标准 SOCKET 的 API，一种是我司对标准 SOCKET 的 API 再封装的 API。默认情况下，两种接口都是打开的，具体参见 wm_config.h

标准 SOCKET 的 API 列表见 wm_sockets.h

Winnermicro 封装的 SOCKET 的 API 列表见 wm_socket.h

至于标准的 SOCKET 通信的 API 则可自行网络学习和使用。

Winnermicro 封装的 SOCKET 的 API 实例见：

Wm_socket_client_demo.c

Wm_socket_server_demo.c

13 AT 指令添加

13.1 实例：HTTP 指令远程升级

实现用户自定义的 AT 指令，达到模块获取远程服务器的升级固件。

用户可以通过 source insight 搜索“Z”(必须加双引号)，找到文件 wm_cmdp_at.c 的 4536 行（具体以搜索结果为准）

在结构体数组 static struct tls_atcmd_t atcmd_tbl[]中添加一行：

{"FWUP", 0, atcmd_fwup_proc},

其中 FWUP 是 AT 指令，

atcmd_fwup_proc 则是该指令对应的函数

如下图红色方框标记：

```
04948: #endif
04949: #if TLS_CONFIG_WIFI_PING_TEST
04950:     { "PING", 0, atcmd_ping_proc},
04951: #endif
04952: #if TLS_CONFIG_WPS
04953:   { "WWPS", 0, atcmd_wps_proc },
04954: #endif
04955: #if USER_AT_COMMAND
04956:     {"FWUP", 0, atcmd_fwup_proc},
04957: #endif
04958:     { "CUSTDATA", 0, atcmd_custdata_proc},
04959:   { NULL, 0, NULL },
04960: };
```

atcmd_fwup_proc 函数源码如下：【此函数代码仅供参考】

使用说明：

1：可以传入 1 个参数，升级固件的完成路径，如 1：

操作一：串口输入：at+fwup=http://192.168.0.102:8080/TestWeb/WM_SDK.img

2：可以传入 3 个参数，服务器 IP 或域名(必须可用)，固件路径，端口号，如 2：

操作二：串口输入：AT+FWUP=192.168.0.102, /TestWeb/WM_SDK.img,8080

串口输入：AT+FWUP=www.winnermicro.com, /TestWeb/WM_SDK.img,8080

【Http 服务器软件，推荐搜索关键字：网络服务器 hfs】

/*因文档原因，如直接复制，请自行排版格式*/

```
#if USER_AT_COMMAND
static int atcmd_fwup_proc( struct tls_atcmd_token_t *tok, char *res_resp, u32
```

```
*res_len)
    {
        u8 *FwupIp = NULL;
        u8 *FwupName =NULL;
        uint32 nRetCode = 0;
        u8 lks_status =0;
        struct hostent* HostEntry;
        int err = 0;
        int ret;
        u32    params;
        struct tls_ip_info_t* ipinfo;
        HTTPParameters httpParams;

        memset(&httpParams, 0, sizeof(HTTPParameters));
        httpParams.Uri = (CHAR*)tls_mem_alloc(128);
        if(httpParams.Uri == NULL)
        {
            printf("malloc error.\n");
            return WM_FAILED;
        }

        if (tls_cmd_get_net_up()) //http 升级前，必须检测模块是否在网
        {
            lks_status = 1;
        }else
        {
             lks_status = 0;
            printf("net is fail\r\n");
            return WM_FAILED;
        }
        ipinfo = tls_mem_alloc(sizeof(struct tls_ip_info_t));
        if(ipinfo == NULL){
```

```
            printf("malloc error.\n");
            return WM_FAILED;
        }
        FwupIp = (u8*)tls_mem_alloc((56+8)*sizeof(u8));//www.[0~63].com
        if(FwupIp == NULL){
            printf("malloc error.\n");
            return WM_FAILED;
        }
        FwupName=  (u8*)tls_mem_alloc(63*sizeof(u8));// .../.../.../...[length is 63]
        if(FwupName == NULL){
            printf("malloc error.\n");
            return WM_FAILED;
        }
        if (!tok->arg_found && ((tok->op == ATCMD_OP_NULL) || (tok->op == ATCMD_OP_QU))) {
        res_resp=sprintf(res_resp,"err,example:at+fwup=192.168.1.100/fwup/wm_sdk.img");
        }
        else if(tok->arg_found == 1 &&((tok->op == ATCMD_OP_EP)||(tok->op==ATCMD_OP_EQ))){
            ret = atcmd_filter_quotation(&FwupName,(u8 *)tok->arg[0]);
            if (ret){
                err = 1;
                return err;
            }//judge args is ok or err;
                memset(httpParams.Uri, 0, 128);
                sprintf(httpParams.Uri,FwupName);
               printf("Location: %s\n",httpParams.Uri);
               httpParams.Verbose = TRUE;
               user_http_fwup(httpParams); //调用 HTTP API
               tls_mem_free(httpParams.Uri);
```

```
        }
        else if (tok->arg_found == 3 &&((tok->op == ATCMD_OP_EP)||(tok->op==
ATCMD_OP_EQ))){
            ret = string_to_ipaddr(tok->arg[0], (u8 *)&params);//args changge
ip_addr
            if (!ret){
                MEMCPY(ipinfo->ip_addr, (u8 *)&params, 4);
printf("ipis  %d.%d.%d.%d",ipinfo->ip_addr[0],ipinfo->ip_addr[1],ipinfo->ip_ad
dr[2],ipinfo->ip_addr[3]);
sprintf(FwupIp,"%d.%d.%d.%d",ipinfo->ip_addr[0],ipinfo->ip_addr[1],ipinfo->ip
_addr[2],ipinfo->ip_addr[3]);
            }else
            {
                atcmd_filter_quotation(&FwupIp, (u8 *)tok->arg[0]);
                HostEntry = gethostbyname((char *)FwupIp);
                if(HostEntry){
                    MEMCPY(ipinfo->ip_addr, HostEntry->h_addr_list[0], 4);
        //printf("hostip:  %d.%d.%d.%d",ipinfo->ip_addr[0],ipinfo->ip_addr[1],i
pinfo->ip_addr[2],ipinfo->ip_addr[3]);
                    sprintf(FwupIp,
"%d.%d.%d.%d",ipinfo->ip_addr[0],ipinfo->ip_addr[1],ipinfo->ip_addr[2],ipinfo
->ip_addr[3]);
                } else
                    {
                        err = 1;
                        return err;
                    }
            }
            ret = atcmd_filter_quotation(&FwupName,(u8 *)tok->arg[1]);
            //printf("the fwupname is%s\r\n",FwupName);
            if (ret){
                err = 1;
```

```
            return err;
        }
        ret = string_to_uint(tok->arg[2], &params);
        printf ("port is %d\r\n",params);
            if (ret || params > 65535||params < 1) {
                err = 1;
                return err;
            }
        memset(httpParams.Uri, 0, 128);
        if(strncmp(FwupName , "/" , 1 ) == 0){
        sprintf(httpParams.Uri,"http://%s:%d%s",FwupIp,params,FwupName);
        }else{
        sprintf(httpParams.Uri,"http://%s:%d/%s",FwupIp,params,FwupName);
        }
        printf("Location: %s\n",httpParams.Uri);
        httpParams.Verbose = TRUE;
        user_http_fwup(httpParams);
        tls_mem_free(httpParams.Uri);
    }else//input args number is too many
    {
        *res_len = atcmd_err_resp(res_resp, CMD_ERR_OPS);
    }
    return 0;
}
#endif
```

14 SPI 指令添加

在 wm_cmdp_hostif.h 中添加一个宏定义，一定不要和其他存在 RI 指令数字存在冲突，同时也不要定义 0xE0~0xEF(这是 RI 指令系统事件用的)。

如下图，定义一个宏 HOSTIF_CMD_TEST

```
00090: #define HOSTIF_CMD_AOLM          0x63
00091: #define HOSTIF_CMD_PORTM          0x64
00092: #define HOSTIF_CMD_UART          0x65
00093: #define HOSTIF_CMD_ATLT          0x66
00094: #define HOSTIF_CMD_DNS           0x67
00095: #define HOSTIF_CMD_DDNS          0x68
00096: #define HOSTIF_CMD_UPNP          0x69
00097: #define HOSTIF_CMD_DNAME          0x6A
00098:
00099: #define HOSTIF_CMD_TEST          0xDF
00100:
00101: #define HOSTIF_CMD_DBG           0xF0
00102: #define HOSTIF_CMD_REGR          0xF1
00103: #define HOSTIF_CMD_REGW          0xF2
00104: #define HOSTIF_CMD_RFR           0xF3
```

接着在 wm_cmdp_ri.c 中的结构数组 static struct tls_ricmd_t ri_cmd_tbl[]添加一行：

{HOSTIF_CMD_TEST, ricmd_test_proc },

其中 HOSTIF_CMD_TEST 数值对应 RI 指令的 HEX 值，而 ricmd_test_pro 则是其对应的函数。下图方框所示：

```
:
: static struct tls_ricmd_t ri_cmd_tbl[]={
: #if TLS_CONFIG_RI_CMD
: {HOSTIF_CMD_NOP,        ricmd_nop_proc        },
: {HOSTIF_CMD_TEST,       ricmd_test_proc       }, //ADD BY ZPF
: {HOSTIF_CMD_RESET,      ricmd_reset_proc      },
: {HOSTIF_CMD_PS,         ricmd_ps_proc         },
: {HOSTIF_CMD_RESET_FLASH, ricmd_reset_flash_proc },
: {HOSTIF_CMD_PMTF,       ricmd_pmtf_proc       },
: {HOSTIF_CMD_GPIO,       ricmd_gpio_proc       },
: {HOSTIF_CMD_MAC,        ricmd_get_mac_proc    },
: {HOSTIF_CMD_VER,        ricmd_ver_proc        },
: {HOSTIF_CMD_WJOIN,      ricmd_join_proc       },
: {HOSTIF_CMD_WLEAVE,     ricmd_disconnect_proc },
: {HOSTIF_CMD_WSCAN,      ricmd_scan_proc       },
: {HOSTIF_CMD_LINK_STATUS, ricmd_link_status_proc },
```

在 wm_cmdp_ri.c 中实现 ricmd_test_proc 函数，该函数实现了打印一句话：this is test by zpf

```
int ricmd_test_proc(char *buf, u32 length, char *cmdrsp_buf, u32 *cmdrsp_size)
{
    u8 err = 0;
    struct tls_hostif_cmdrsp *cmdrsp = (struct tls_hostif_cmdrsp *)cmdrsp_buf;
    if (length != sizeof(struct tls_hostif_cmd_hdr))
        err = CMD_ERR_INV_PARAMS;
    tls_hostif_fill_cmdrsp_hdr(cmdrsp, HOSTIF_CMD_RESET, err, 0);
    *cmdrsp_size = sizeof(struct tls_hostif_cmd_hdr);
    printf("this is test by zpf\r\n");
```

```
    return 0;
}
```

15 网络升级

```
/*****************************************************************
* Description: http fwup function is for http fwup;
* please set HttpRemoteIp value and path is":8080/TestWeb/fwup.img"
* Auth: zpf
*Date: 2015-6-10
*****************************************************************/
int http_fwup_user(void)
{
    char *user_FwupLocation ="http://180.178.37.99/arc/code/fwup.img";
/*定义升级的固件所在的服务器位置。如果使用域名，则需要将 IP 地址换成域名即可*/
    HTTPParameters httpParams;
    memset(&httpParams, 0, sizeof(HTTPParameters));
    httpParams.Uri = (CHAR*)tls_mem_alloc(128);
    if(httpParams.Uri == NULL)
    {
        printf("malloc error.\n");
        return WM_FAILED;
    }
    memset(httpParams.Uri, 0, 128);
    //sprintf(httpParams.Uri,"http://%d.%d.%d.%d:8080/TestWeb/fwup.img",
HttpRemoteIp[0],HttpRemoteIp[1],HttpRemoteIp[2],HttpRemoteIp[3]);
    //printf("Location: %s\n",httpParams.Uri);
    sprintf(httpParams.Uri, user_FwupLocation);
    printf("Location: %s\n",httpParams.Uri);
    httpParams.Verbose = TRUE;
    http_fwup(httpParams);
    tls_mem_free(httpParams.Uri);
    return WM_SUCCESS;
```

```
}
//说明：可在主任务中，通过消息机制，一旦触发联网消息，
//便调用此函数，即可实现联网升级功能。
```