Emtronix

ETA6103 串口 Wifi 扩展模块应用手册

感谢您购买英创信息技术有限公司的产品：**ETA6103串口Wifi扩展模块。**

您可以访问英创公司网站或直接与英创公司联系以获得ETA6103的其他相关资料。

英创信息技术有限公司联系方式如下：

地址：成都市高新区高朋大道5号博士创业园B座407# 邮编：610041

联系电话：028-86180660 传真：028-85141028

网址：http://www.emtronix.com 电子邮件：support@emtronix.com

目　　录

1.ETA6103 V1.1简介

ETA6103 是基于亿伯特公司的 E103-W01 和沁恒公司的 CH340T 设计的串口 WIFI 转 miniPCIe 接口扩展电路板。该模块可以通过 MiniPCIe 转 USB 模块 ETA303 在英创公司的所有嵌入式主板（X86 系列及 ARM 系列）中使用。工作中被虚拟成/dev/ttyUSBx 设备，最高波特率 230400，最大支持 5 个网络连接（包括 TCP 和 UDP）。英创公司提供针对 ETA6103 的驱动及应用程序范例。本文将介绍 ETA6103 的使用、信号定义等。在英创公司网站搜索 ETA6103、串口 WIFI，可以得到更多扩展应用的信息。

2.硬件接口说明

ETA6103 的硬件部分主要由串口 wifi 模块 E103（U2）、USB 转串口芯片 CH340T（U1）和 miniPCIe 插头（J1）组成，如图 1 所示。用户可以通过 miniPCIe 转 USB 模块 ETA303 接入英创公司的嵌入式主板进行评估。

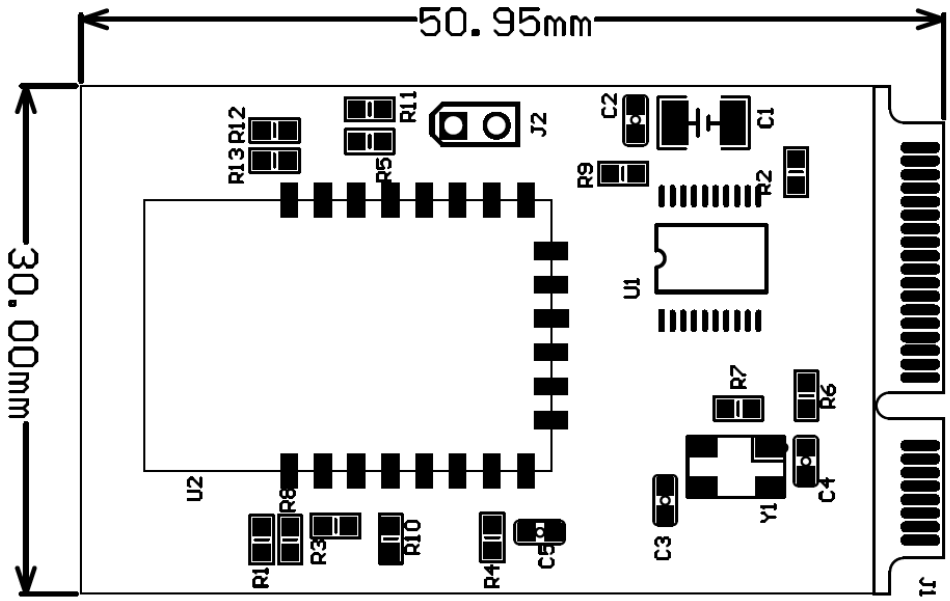

图 1：ETA6103 示意图

J1 为 miniPCIe 接口，正面为奇数引脚，背面为偶数引脚。E103 提供了一些额外功能（详见 E103 的使用手册），在 ETA6103 中默认只使用了串口 wifi 的功能（只引出了 36、38、42、44、46），如需使用其他功能，用户需要自己设计底板电路，并焊接 ETA6103 中对应的 0 欧电阻。此处的 GPIO 编号与主板的 GPIO 编号无关，是独立无相关的，即此处的 GPIO5/PWM3 与主板的 GPIO5 没有关系。

J1 具体信号定义如下表（信号名称带#尾缀的，表示低电平有效）。

信号名称及简要描述	J1		信号名称及简要描述
	PIN#	PIN#	
GPIO16，输入，睡眠唤醒，需焊接 0 欧电阻 R10	1	2	VCC_3V3，电源 3.3V
未使用	3	4	GND，数字地
未使用	5	6	未使用
未使用	7	8	未使用
GND，数字地	9	10	未使用
未使用	11	12	未使用
未使用	13	14	未使用
GND，数字地	15	16	未使用
未使用	17	18	GND，数字地
未使用	19	20	WIFI_RST#，E103 复位输入，低有效
GND，数字地	21	22	未使用
未使用	23	24	未使用
未使用	25	26	GND，数字地
GND，数字地	27	28	未使用
GND，数字地	29	30	未使用
未使用	31	32	未使用
未使用	33	34	GND，数字地
GND，数字地	35	36	USB_D-，USB 数据差分信号-
ADC，需要焊接 0 欧电阻 R3	37	38	USB_D+，USB 数据差分信号+
GPIO13，需要焊接 0 欧电阻 R4	39	40	GND，数字地
GPIO5/PWM3，需要焊接 0 欧电阻 R5	41	42	GPIO12/PWM0
未使用	43	44	GPIO14/PWM1
未使用	45	46	GPIO4/PWM2
未使用	47	48	未使用
未使用	49	50	GND，数字地

未使用	51	52	VCC_3V3，电源 3.3V

3.应用说明

ETA6103 可以在英创的 WinCE 系统和 Linux 系统中使用。用户可以自己根据英创公司提供的相关芯片 AT 指令手册进行配置和 TCP/UDP 传输；同时，为方便客户使用，英创公司将 AT 指令操作封装成了 C++类，用户可以直接调用 C++类的方法，而不用关心底层的 AT 指令，下面就来详细讲述。

需要注意，ETA6103 不支持流控！最大支持 5 路连接！

3.1 ETA6103 在 WinCE 操作系统中的应用

ETA6103 可以直接到英创 WINCE 板卡上使用，如果提示未知的 USB 设备，请联系英创工程师更新最新的内核。

ETA6103 连接以后，可以通过 COM10 与模块进行通信。英创提供源代码形式的封装好的 CUartWifi 类，用户在工程中添加 CUartWifi 头文件，源文件，即可方便的通过 CUartWifi 类，查询 AP，连接 AP，及进行 TCP,UDP 的收发。用户也可以自己优化 CUartWifi 类的代码，以适应自己的应用需求。

3.1.1. 封装说明

一共需要添加 4 个文件加入用户工程，Serial.h，Serial..cpp，ETA6103.h，ETA6103.cpp。Serial 中是串口类 CSerial 的封装代码，ETA6103 中是 CUartWifi 类的封装代码，CUartWifi 类通过 CSerial 类进行串口通信。

使用时只需要引用 ETA6103 就行了

```
#include "ETA6103.h"
```

3.1.2. CUartWifi 基础功能函数

1.构造函数：

```
CUartWifi();
CUartWifi(int nPort, int baudrate = 115200);
//参数nPort串口号，baudrate波特率，如不制定，则默认使用COM10(ETA6103默认)，115200波特率
```

2.打开模块：

```
BOOL Open();
//打开模块，打开串口，打开串口接收线程，初始化事件参数
```

成功返回 TRUE，失败返回 FALSE

3.关闭模块：

```
BOOL Close();
//关闭模块，关闭串口，关闭串口接收线程，关闭事件
```

成功返回 TRUE，失败返回 FALSE

4.测试模块 AT 通信：

```
BOOL Test();
//测试与 WIFI 芯片 E103 AT 指令通信是否正常
```

成功返回 TRUE，失败返回 FALSE

5.重置模块：

```
BOOL Reset();
//重置模块，恢复所有默认配置
```

成功返回 TRUE，失败返回 FALSE

6.模块基础配置：

```
BOOL AutoCfg();
//使用之前的参数设置，用户可以自行调整。
//封装的代码为了方便指令解析，做了需要的设置，包括，1 关闭回显，2 设置模块工
```

作模式为 station，即连接 WIFI 的单元，3 设置传输模式为普通传输，4 设置连接模式为多连接

成功返回 TRUE，失败返回 FALSE

3.1.3. Wifi 连接相关函数

结构体定义

```
struct APInfo{
    char    ssid[50];       //ssid
    char    mac[18];        //mac ca:d7:19:d8:a6:44
    int     channel;
    int     rssi;           //rssi 信号强度
    int     ecn;            //加密方式：OPEN，WEP，WPA_PSK，WPA2_PS，
WPA_WPA2_PSK
    int     freqoffset;     //频偏
    int     freqcalibration;    //频率校准
};
```

1.获取当前连接的 AP：

```
BOOL GetCurAP(char *buf, int buflen);
//参数为返回的信息buf及信息长度
```

成功返回 TRUE，失败或当前无 AP 返回 FALSE

```
BOOL GetCurAP(APInfo *pApInfo);
//参数为AP信息的结构体，结构体定义如上
```

成功返回 TRUE，失败或当前无 AP 返回 FALSE

2.获取默认连接的 AP：

```
BOOL GetDefAP (APInfo *pApInfo);
//默认AP即模块保证在flash中，重新上电后依然自动连接的AP
```

成功返回 TRUE，失败返回 FALSE

3.搜索附近 AP：

```
BOOL GetAPList(char *buf, int buflen);
//参数为返回的信息buf及信息长度
```

成功返回 TRUE，失败返回 FALSE

```
BOOL GetAPList (APInfo *pApInfo, int maxnum, int *pnum);
//参数为AP信息的结构体数组指针，数组最大大小，pnum为返回实际找到的AP数量，不超过maxnum
```

成功返回 TRUE，失败返回 FALSE

4.连接 AP：

```
BOOL ConnectAP(char *ssid, char *password);
//连接AP，参数ssid及密码
```

成功返回 TRUE，失败返回 FALSE

5.设置成默认连接 AP：

```
BOOL SetDefAP (char *ssid, char *password);
//保存到flash，重新上电后能自动连接
```

成功返回 TRUE，失败返回 FALSE

6.断开当前 AP：

```
BOOL DisconnectAP ();
//断开当前AP
```

成功返回 TRUE，失败返回 FALSE

3.1.4. 网络基础功能函数

结构体定义

```
struct IPInfo{
    char ip[16];
    char gw[16];
    char mask[16];
};
```

1.获取当前 IP 信息：

```
BOOL GetCurIP(char *buf, int buflen);
//参数为返回的信息buf及信息长度
```

成功返回 TRUE，失败返回 FALSE

```
BOOL GetCurIP(IPInfo *pIpInfo);
//参数为IP信息的结构体，结构体定义如上
```
成功返回 TRUE，失败返回 FALSE

未连接上 AP 时，IP 为 0.0.0.0

2.将网络地址解析为 IP 地址：

```
BOOL DomainIP(char *domain, char *buf, int buflen);
//参数为domain网络地址字符串，如”www.baidu.com”，返回的信息buf及信息长度
```
成功返回 TRUE，失败返回 FALSE

3.PING 一个 IP 地址：

```
BOOL Ping(char *ip, char *buf, int buflen);
//参数为IP地址字符串，返回的信息buf及信息长度
```
成功返回 TRUE，失败或无法 ping 通返回 FALSE

3.1.5. TCP/UDP 网络应用

与 SOCKET 编程不同，ETA6103 以连接号进行网络通信，最大 5 个链接。

TCP 客户端，UDP 连接端，连接到 TCP 服务端的远端都将占用一个链接，并有模块分配的一个链接号。所有通信都是基于这个链接号的。

1.连接 TCP 服务端（client 模式）：

```
BOOL ConnectTcpServer(int id, char *rip, int rport);
//参数为选择的链接号（必须未占用的），服务端ip字符串rip，服务端端口号rport
```
成功返回 TRUE，失败返回 FALSE

连接成功后，链接号 id 即作为与服务端通信的标记

2.打开 UDP 端口（同时设置接收和发送）：

```
BOOL ConnectUdp(int id, char *rip, int rport, int lport);
//参数为选择的链接号（必须未占用的），远端ip字符串rip，远端端口号rport，本
```

地打开的UDP端口（可以不设置，即不接收UDP消息）
成功返回 TRUE，失败返回 FALSE

连接成功后，链接号 id 即作为与远端通信的标记

3.创建 TCP 服务端（server 模式）：

```
BOOL CreateTcpServer (int lport);
//开启TCP服务器，端口号lport
```
成功返回 TRUE，失败返回 FALSE

创建 server 并不占用链接 ID 号，但是每个连接进来的客户端都会占用一个链接号

4.关闭 TCP 服务端（server 模式）：

```
BOOL CloseTcpServer ();
//关闭TCP服务器
```
成功返回 TRUE，失败返回 FALSE

5.发送数据：

```
BOOL SendMsg(int id, char *buf, int buflen = -1);
//根据链接号id，发送数据，当buflen不设置，为-1时，默认发送buf里的字符串，长度为字符串长度。
```
成功返回 TRUE，失败返回 FALSE

该方法 TCP 客户端服务端及 UDP 通用

6.等待接收消息：

```
DWORD WaitMsg(DWORD dwMilliseconds = 2000);
//等待接收消息，默认延迟dwMilliseconds为2000毫秒
```
等待到消息返回 0，未等待到消息返回 258

通过返回值可以知道是否有数据接收，或者是 TCP 服务端有客户端连入。接收到事件后，需要查询 CUartWifi 类的 LinkInfo 成员的 eventFlag，来判定是什么事件，事件定义如

下

```
#define EVT_STAT        0x01            //连接或断开
#define EVT_RECV        0x02            //接收
#define EVT_ERR         0x04            //错误（BUFFER 满）
```

7.接收数据：

BOOL ReadMsg(int id, char *buf, int buflen, DWORD *dwlen);

//接收数据，以链接号id为标记，buf为要接收的BUFFER指针，buflen为接收BUFFER的大小，dwlen为实际接收的长度值

成功返回 TRUE，失败返回 FALSE

不同链接号接收的数据存在各自的 BUFFER 中，所以需要通过指定链接号来判断需要读取哪个链接号里的数据。

8.断开网络链接（TCP UDP 共用）：

BOOL CloseConn(int id);

//断开链接号为id的链接，当id为5，则断开所有链接

成功返回 TRUE，失败返回 FALSE

3.2 ETA6103 在 Linux 操作系统中的应用

ETA6103 在 Linux 平台中使用，当连接好硬件后，客户需要加载驱动，让系统识别出扩展的串口。驱动程序已经编译成两个 ko 文件，其中，usbserial.ko 以模块的形式放在文件系统中，ch340.ko 需要联系我们获取。用户需要拷贝 ch340.ko 到 usbserial.ko 所在目录，然后执行 insmod 命令加载 ko 文件即可：

insmod /lib/modules/(Linux versions)/usbserial.ko

insmod /lib/modules/(Linux versions)/ch340.ko

加载完成后，系统会将 ETA6103 识别为一个串口设备/dev/ttyUSBx，操作方法和标准的串口相同。调用 open( )打开设备文件，再调用 read( )、write( )对串口进行数据读写操作（发送 AT 指令），串口的具体操作可以参看 Step2_SerialTest。为方便用户更高效的开

发 ETA6103 的应用程序，我们封装了一个类 CUartWifi，直接调用 CUARTWifi 的方法可以实现 ETA6103 的配置以及 TCP/UDP 网络通信。下面介绍 CUartWifi 类的使用。

3.2.1. CUartWifi 类介绍

CUartWifi 类提供了 13 个公共方法，涵盖了模块的常用配置和网络传输：

1.配置方法：

```
// AT+RST 重启模块，成功返回0，失败返回-1
int Reset();
//连接到AP，如果ssid中有特殊字符，需要转义，如"Emtronix\\.20"表示SSID为"Emtronix.20"的无线网络
//0 -- 成功连接 ;-1 -- 连接超时; -2 -- 密码错误; -3 --找不到目标AP; -4 --连接失败
int ConnectToAP(char*ssid, char* password);
//查询ip地址、网关、子网掩码，结果写入addr,gateway,netmask所在地址，需要保证16字节的空间，成功返回0，失败返回-1
int GetIPInfo(char* addr, char* gateway, char* netmask);
//设置ip地址、网关、子网掩码，从addr,gateway,netmask中读取设置值，成功返回0，失败返回-1
int SetIPInfo(char* addr, char* gateway, char* netmask);
//设置dhcp,默认dhcp打开，成功返回0，失败返回-1
int SetDHCP(int en);
//断开与外部AP的连接，成功返回0，失败返回-1
int DisconnectToAP();
//一次ping功能，ipaddr可以为地址或者ip不能指定包大小
//成功返回响应时间ms;失败返回-1
int Ping(char *ipaddr);
```

这部分接口主要用于ETA6103的wifi配置，使用ConnectToAP可以连接wifi，使用SetIPInfo可以设置ETA6103的IP地址等，使用Ping可以检查与目标地址是否连接上。

2.TCP/UDP 传输方法

```
/*-------------------------------------------------------------
    【函数介绍】:  此函数用于启动TCP服务器或者建立TCP/UDP客户端连接
    【入口参数】:
            ip  -- "*", 建立TCP服务器，等待连接到来
                -- “192.168.201.119”，建立客户端连接时远端IP地址
            port -- 端口号，做服务器是是监听的本地端口号1-65535；做
```

```
                客户端时为要连接的远端端口号
            transType -- 传输协议，做服务器时固定为TCP；做客户端时
                TCP/UDP可选
            keepalive -- 做TCP服务器时为服务器超时时间；做TCP客户
                端时，为TCP超时时间，UDP客户端时指定本地的端口
    【出口参数】：  无
    【返回  值】：
            client：    成功 -- 连接号
                        失败 -- -1
            server：    成功 -- 0
                        失败 -- -1
   ------------------------------------------------------*/
    int Open(char* ip,int port,int transType,int keepalive);

/*-------------------------------------------------------------
    【函数介绍】：  做TCP服务器时，在timeout ms内等待连接到来，有连接
到来返回连接号，无连接到来返回-1
    【入口参数】：  timeout ：超时时间ms，0为不超时
    【出口参数】：  无
    【返回  值】：   ≥ 0：有连接到来，返回连接号
                    -1：等待连接超时
  -------------------------------------------------------------*/
    int Accept(int timeout);

 /*-------------------------------------------------------------
    【函数介绍】：  等待连接conno远端的事件，返回事件号，表明数据、连接
终止
    【入口参数】：  conno  ：   连接号
            timeout ：超时时间ms，0为不超时
    【出口参数】：  无
    【返回  值】：   事件号
-------------------------------------------------------------*/
    int WaitEvent(int conno, int timeout_ms);

 /*-------------------------------------------------------------
    【函数介绍】：  从TCP/UDP传输数据缓存区读取最大maxSize个字节到
buf，返回读取的字节数
    【入口参数】：   conno：连接号
                    buf  ：读数据地址
                    maxSize：最大读字节数
    【出口参数】：  无
    【返回  值】：   >0：读取的实际字节数
                    0：失败
 -------------------------------------------------------------*/
```

```
    int Recv(int conno, char*buf, int maxSize);

/*------------------------------------------------------------
    【函数介绍】:  通过TCP/UDP发送len字节数据
    【入口参数】:    conno：连接号
                     msg：发数据地址
                     len  ：发送字节数
    【出口参数】:  无
    【返回  值】:    >0：发送的实际字节数
                     0：失败
  ------------------------------------------------------------*/
    int Send(int conno,char* msg, int len);
/*------------------------------------------------------------
    【函数介绍】:  关闭连接，如果有连接存在，关闭对应conno连接；如果没有连接存在，server开启状态，则关闭server
    【入口参数】:  conno：连接号
    【出口参数】:  无
    【返回  值】:   0：成功
                  -1：失败
------------------------------------------------------------*/
    int Close(int conno);
```

3.2.2. CUartWifi 类的调用

在 CUARTWifi 的构造函数中，对 ETA6103 的工作模式等做了设置，用户可以不用关心。与使用其他无线设备一样，首先，需要用户提供无线网络的 ssid 和密码，连接到无线网络，然后进行网络通信，提供服务端或者客户端功能，最后断开网络。

1、连接无线网络

```
CUartWifi uartWifi(devname);
ret = uartWifi.SetDHCP(1);
if(ret == 0)
{
   printf("******Set DHCP success\n");
}
else
{
   printf("******Set DHCP failed\n");
}
printf("******CONNECT TO AP\n");
ret = uartWifi.ConnectToAP("Emtronix\\.20","0987654321");//
```

需要对特殊字符进行转义

```
    if(ret < 0)
    {
        printf("*******CONNECT FAILED: %d\n",ret);
        exit(0);
    }
    printf("*******CONNECT SUCCESS\n");
    char ip[16];
    char gw[16];
    char netMask[16];
    ret = uartWifi.GetIPInfo(ip,gw,netMask);
    if(ret == 0)
    {
        printf("******GetIPInfo Success\n");
        printf("IP:%s\nGW:%s\nNetMask:%s\n",ip,gw,netMask);
    }
    ret = uartWifi.SetIPInfo("192.168.201.93","192.168.201.20",
"255.255.255.0");
    if(ret == 0)
    {
        ret = uartWifi.GetIPInfo(ip,gw,netMask);
        if(ret == 0)
        {
            printf("******SetIPInfo Success\n");
            printf("IP:%s\nGW:%s\nNetMask:%s\n",ip,gw,netMask);
        }
    }
    else
    {
        printf("******SetIPInfo Failed\n");
        exit(0);
    }
```

需要注意的是 ETA6103 的 AT 指令中 ssid 需要对特殊字符做转义，假如无线网络账号是"Emtronix.20"，ETA6103 的 AT 指令中 ssid 应该输入"Emtronix\.20"，而通过 CUARTWifi 类发送的 ssid 参数就应该是"Emtronix\\.20"。连接上无线网络后，可以调用 `SetIPInfo` 设置 IP 地址、网关、子网掩码，还可以使用 `Ping` 工具检查连接状况，详见例程。

2、建立 TCP 服务端

```
    char server[] = "*";
    ret = uartWifi.Open(server,6002,TCP_TRANS_TYPE,0);
    if(ret != 0)
```

```
    {
        printf("******TCP Server Start Failed\n");
        return 0;
    }
    printf("******TCP Server Start\n");

    int totalConno = 0;
    pthread_mutex_t totalConno_mutex;
    pthread_mutex_init(&totalConno_mutex,NULL);
    while(1)
    {
        ret = uartWifi.Accept(0);
        if(ret < 0)
        {
            printf("Accept timeout\n");
            continue;
        }
        pthread_mutex_lock(&totalConno_mutex);
        totalConno++;
        pthread_mutex_unlock(&totalConno_mutex);
        printf("Server:conno %d is comming\n",ret);
        threadFuncParam.pUartWifi = &uartWifi;
        threadFuncParam.conno = ret;
        threadFuncParam.pTotalConno = &totalConno;
        threadFuncParam.pTotalConnoMutex = &totalConno_mutex;
        if(ConnThreadCreate(&connThread[ret],(void * (*)(void
*))&ConnThreadFunc1,&threadFuncParam) == 0)
        {
            printf("Create thread for conno %d,
threadID %d\n",ret,(int)connThread[ret]);
        }
        printf("totalConno = %d\n",totalConno);
    }
```

ETA6103支持的最大网络连接数是5个，包括所有的客户端连接和服务端连接。在建立服务端连接时，首先使用Open函数开启服务端功能，然后马上进入Accept状态，每收到一个连接，就为这个新连接开启一个线程。建立新线程时传递参数要用到下面的结构体：

```
struct ThreadFuncParam
{
    CUartWifi * pUartWifi;
    int conno;
    int *pTotalConno;
    pthread_mutex_t *pTotalConnoMutex;
};
```

在此线程中调用 WaitEvent 等待此连接的数据或连接中断的事件，并做对应的处理。这里对接收到的数据进行回发，使用 do{}while 循环是保证对应 conn 的接收数据缓存中的数据能全部读出。在退出此线程时，表示 conn 连接关闭，使用指针对主程序中的 TotalConno 做了“减一”处理。

```
int ConnThreadFunc1( void* lparam )
{
    CUartWifi * pUartWifi = ((struct ThreadFuncParam
* )lparam)->pUartWifi;
    int conn = ((struct ThreadFuncParam * )lparam)->conno;
    int* pTotalConno = ((struct ThreadFuncParam
* )lparam)->pTotalConno;
    int connClose = 0;
    int event;
    int readNum = 0;
    int i = 0;
    char buf[1024] = {'0'};
    printf("conno %d waiting event\n",conn);
    while(connClose != 1)
    {
        event = pUartWifi->WaitEvent(conn,2000);
        switch(event)
        {
        case CONN_EVENT_TIMEOUT:
            printf("Conn %d, WaitEvent Timeout\n",conn);
            break;
        case CONN_EVENT_DATAIN:
            printf("Conn %d, WaitEvent Data In\n",conn);
            do{
                i = 0;
                readNum = pUartWifi->Recv(conn,buf,1024);
                while(readNum > i)
                {
                    //处理接收数据
                    printf("%c ",buf[i]);
                    if(i % 10 == 9)
                    {
                        printf("\n");
                    }
                    i++;
                }
                printf("\n");
                pUartWifi->Send(conn,buf,readNum);
```

```
            }while(readNum == 1024);//不一定是1024，
            break;
        case CONN_EVENT_CLOSE:
            printf("Conn %d, WaitEvent Close\n",conn);
            connClose = 1;
            break;
        case CONN_EVENT_ERROR:
            printf("Conn %d error\n",conn);
            connClose = 1;
            break;
        default:
            printf("Conn %d, WaitEvent Default\n",conn);
            break;
        }
    }
    pthread_mutex_lock(((struct ThreadFuncParam
* )lparam)->pTotalConnoMutex);
    if(*pTotalConno > 0)
        *pTotalConno = *pTotalConno - 1;
    pthread_mutex_unlock(((struct ThreadFuncParam
* )lparam)->pTotalConnoMutex);

    printf("conno %d thread finish\n",conn);
    return 0;
}
```

3、建立 TCP 客户端连接

客户端也使用 Open 建立，得到新连接后使用 WaitEvent 监听处理此连接的事件。

```
    ret = uartWifi.Open(remoteIP,remotePort,TCP_TRANS_TYPE,0);
    if(ret >= 0 && ret < 5)
    {
        printf("******Client TCP Connection %d Established\n",
ret);
        int connClose = 0;
        int conn = ret;
        int event;
        int readNum = 0;
        int i = 0;
        int timeoutCount = 0;
        char sendBuf[1024] = {0};
```

```
        char buf[1024] = {0};
        printf("conno %d waiting event\n",conn);
        while(connClose != 1)
        {
            event = uartWifi.WaitEvent(conn,2000);
            switch(event)
            {
            case CONN_EVENT_TIMEOUT:
                printf("Conn %d, WaitEvent Timeout\n",conn);
                memset(sendBuf,0,1024);
                sprintf(sendBuf,"Conn %d, WaitEvent
Timeout %d",conn,++timeoutCount);
                uartWifi.Send(conn,sendBuf,strlen(sendBuf));
                break;
            case CONN_EVENT_DATAIN:
                printf("Conn %d, WaitEvent Data In\n",conn);
                do{
                    i = 0;
                    readNum = uartWifi.Recv(conn,buf,1024);
                    while(readNum > i)
                    {
                        //处理接收数据
                        printf("%c ",buf[i]);
                        if(i % 10 == 9)
                        {
                            printf("\n");
                        }
                        i++;
                    }
                    printf("\n");
                }while(readNum == 1024);//不一定是1024，
                printf("Conn %d, WaitEvent Data In
Finished\n",conn);
                break;
            case CONN_EVENT_CLOSE:
                printf("Conn %d, WaitEvent Close\n",conn);
                connClose = 1;
                break;
            case CONN_EVENT_ERROR:
                printf("Conn %d error\n",conn);
                connClose = 1;
                break;
            default:
                printf("Conn %d, WaitEvent Default\n",conn);
```

```
                break;
            }
        }

    }
```

4、关闭连接，断开网络

```
for(i1=0;i1<MAX_LINK_NUM;i1++)
{
    uartWifi.Close(i1);
}

uartWifi.DisconnectToAP();
```

至此，完成了整个通信过程，具体的使用请参考例程。