使用说明

mxchipWNet Library 基础版

使用 mxchipWNet 基础版软件库开发嵌入式 Wi-Fi 应用

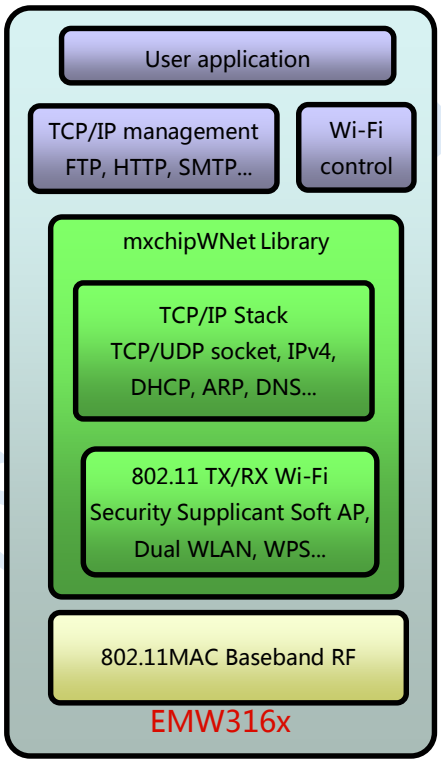
316x0001.014 **Date：2014-1-10** **Reference manual**

概述

mxchipWNet™ library 是运行在 MXCHIP 公司开发的 EMW316x 系列嵌入式 Wi-Fi 模块上的 TCP/IP 协议栈、Wi-Fi 射频驱动、运行环境以及相关示例程序，是构建嵌入式 Wi-Fi 应用的完整解决方案。

该软件库专门为嵌入式 Wi-Fi 应用设计，拥有快速的联网速度，高实时性的事件处理机制，低消耗、可配置的硬件资源管理等特性，并且在 EMW316x 模块上可以实现最高 20Mbps 的超高速传输速率。

同时该软件库采用标准的 BSD socket API 为基础，配合简单易用的无线网络配置 API，开发者可以很方便地在 EMW316x 模块上开发各种嵌入式 Wi-Fi 应用程序。该软件库的易用性能大大缩短开发周期，提高产品上市进度。



主要功能:

- 包含引导程序，应用程序和驱动程序
- 提供 73k 字节的可以自由分配的内存空间
- 提供多达 720k 字节的可用 Flash 空间
- 支持 OTA 无线升级的系统架构
- 内置的功能测试模式
- 提供多种支持工具

Wi-Fi features:

- 支持 IEEE 802.11 b/g/n@2.4GHz
- 支持 Station 和 Soft AP 模式
- 支持 WEP，WPA/WPA2 加密方式
- 支持 IEEE 802.11 power save 模式
- 内存管理和调试功能

TCP/IP features:

- 符合 BSD socket 接口定义
- DHCP 客户端和服务器
- DNS, mDNS (bonjour)服务
- 提供阻塞或非阻塞式的调用方式
- 支持 12 个 TCP socket 和 2 个 UDP socket
- 支持 SSL 加密的 socket 连接
- HTTP、FTP、SMTP 等应用范例

上海庆科信息技术有限公司

无线网络开发部

目录

1 mxchipWNet 软件库简介

1.1 主要功能

mxchipWNet 软件库（下称“软件库”）是运行在 MXCHIP 公司开发的 EMW316x 系列嵌入式 Wi-Fi 模块上的 TCP/IP 协议栈、Wi-Fi 射频驱动、运行环境以及相关示例程序，是构建嵌入式 Wi-Fi 应用的完整解决方案。

该软件库专门为嵌入式 Wi-Fi 应用设计，拥有快速的联网速度，高实时性的事件处理机制，低消耗、可配置的硬件资源管理等特性，并且在 EMW316x 模块上可以实现最高 20Mbps 的超高速传输速率。

同时该软件库采用标准的 BSD socket API 为基础，配合简单易用的无线网络配置 API，开发者可以很方便地在 EMW316x 模块上开发各种嵌入式 Wi-Fi 应用程序。该软件库的易用性能大大缩短开发周期，提高产品上市进度。

1.2 运行环境

1.2.1 硬件平台

基于软件库的应用可以运行在 EMW316x 嵌入式 Wi-Fi 模块上的 STM32 微控制器上。该微控制器主频 120MHz，包含 1M 字节 flash 空间和 128k 字节的 SRAM。

可以从以下网站获得相关的模块信息：

EMW3161：http://www.mxchip.com/product.php?class_id=15&id=2

EMW3162：http://www.mxchip.com/product.php?class_id=15&id=41

1.2.2 FLASH 和 SRAM 空间的使用

Flash 的使用

STM32 的 1M 字节 Flash 空间按照如下列表分配给各个功能块：

表 1.1 FLASH 的分区

起始地址	结束地址	类型	大小（字节）	内容
0x08000000	0x08003FFF	B	16k	引导程序
0x08004000	0x0800BFFF	P	32k	OTA信息，用户参数
0x0800C000	0x08060000	A	336k	用户应用程序
0x08060000	0x080C0000	—	384k	OTA升级数据的暂存空间
0x080C0000	0x080FFFFF	D	256k	射频芯片驱动程序

各个功能块的主要功能描述：

1. 引导程序：

STM32 上电或者复位后会自动运行这部分代码，程序根据 BOOT 引脚的状态决定是否引导应用程序还是进入引导程序的主菜单，用户可以在引导程序的主菜单下通过不同的命令来更新 Flash 中各个分区的内容。同时，引导程序也会根据 Flash 中区域的 OTA 信息决定是否使用 OTA 升级数据的暂存空间中的数据来替换其他分区的数据，以便实现 Flash 中数据的自动升级。

BOOT 引脚定义：EMW3162 PIN16（PB1）EMW3161 PIN36（PE6）

表 1.2 BOOT 引脚的功能

BOOT 引脚	引导状况
0（低电平）	进入引导程序主菜单
1（高电平）	正常引导应用程序

使用引导程序进行 Flash 内容更新的功能请参考应用笔记：

http://www.mxchip.com/uploadfiles/soft/EMW/AN0002E_FirmwareUpdate_V3.pdf

同时，引导程序在运行时，会检查“OTA 升级数据的暂存空间”中的数据状况，当不需要执行 OTA 升级时，或者 OTA 升级已经完成的情况下，引导程序会自动擦除这块区域的所有数据，这个功能的主要目的是：应用程序在接收到 OTA 数据时能够尽快地写入“OTA 升级数据的暂存空间”，而不需要执行擦除操作，提高了 OTA 操作的执行效率。

注意：以上功能用户都可以通过修改引导程序的源代码进行修改。

2. 启动信息，用户参数

这部分 Flash 空间可以用来存储系统运行的参数，比如模块启动时自动连接的无线网网络名称、密码等，除了一部分软件库明确定义的参数结构以外，所有的 Flash 空间都可以由用户应用程序自由组织，擦除和编程。这部分已经定义的参数结构称为 OTA 信息，用户在操作这部分存储空间时，应保证这部分区域不被破坏。

OTA 信息的存储地址：0x08004000，也就是这个存储区域的头部，可以用一个结构体描述这个 OTA 信息块，该结构体定已在 mxchipWNET.h 头文件中的 boot_table_t 结构体中。OTA 信息的结构如下：

表 1.3 OTA 信息的结构：

名称	类型	长度	内容
START ADDRESS	32位数据	1	OTA数据的存储地址（当前固定为0x08060000）
LENGTH	32位数据	1	OTA数据的长度
VERSION	字符	8	版本信息（当前未提供实际功能）
TYPE	字符	1	OTA数据的类型（‘B’,’P’,’A’,’D’）
UPDATE	字符	1	升级标志(‘U’)
REVERSED	8位数据	6	保留

OTA 数据的类型表明了 OTA 暂存数据的数据类型，用于表明引导程序用 OTA 数据刷新目标区块的位置。这个类型和表 1.1 中的“类型”是匹配的，如写入字符 A 时，引导程序就会使用 OTA 数据来刷写 A 区域的数据，也就是用户的应用程序。

UPDATE 字符用于声明是否需要使用引导程序执行刷新的操作，当 OTA 数据已经准备好，应用程序就可以在这个位置写入字符 U，并且软件重启模块，由引导程序使用 OTA 数据刷新目标存储区域。

所以，应用程序实现 OTA 功能的主要流程是：

- 将接收到的升级数据写入地址 0x08060000 开始的 OTA 数据暂存空间。
- 写入正确的 OTA 信息。
- 软件重新启动模块，由引导程序完成刷新操作。
- 引导程序完成 Flash 内存的更新后，将 OTA 信息恢复到初始状态，并擦除 OTA 数据暂存区域的数据
- 正常引导用户应用程序

3. 用户应用程序

用于实现模块功能的可执行程序的存储区域，由于这个区域的起始地址不是 STM32 的 Flash 起始地址，所以用户在编译链接时，应该设置程序的链接地址为 0x0800C000，同时将 STM32 的中断向量表重映射到这个地址。

4. OTA 升级数据的暂存空间

临时保存的 OTA 数据，如果用户在引导程序中关闭 OTA 升级功能，或者采用外部存储保存 OTA 数据，这部分存储空间可以合并到用户应用程序块中，这样就可以存放更庞大的用户应用程序。

5. 射频芯片驱动程序

用于存放模块上的射频芯片的固件，这个固件是固定的，不能修改的，任何对这个区域的修改都会导致系统不能正常运行。一旦这个区域的数据被破坏，就只能通过仿真工具重新写入数据，或者在引导程序的主菜单下，通过命令“driverupdate”命令来重新写入数据。

SRAM 的使用

STM32F2 提供 128k 字节的内存空间，一旦软件库初始化完成，软件库中的 Wi-Fi 驱动和网络协议栈会占用一部分内存空间，并且为 UDP socket 分配 8k 字节静态内存，剩下的可用内存约为 73kbytes 字节。这些剩余的内存空间可以用于用户的应用程序，并且为每一个 socket 分配发送缓冲区(WRBUF)和接收缓冲区(RDBUF)的大小，需要注意的是 socket 的发送缓冲区和接收缓冲区的大小和网络传输速度息息相关，比较大的缓冲区能够提供更加高速的传输速度，但是总内存大小的限制又使得缓冲区的大小不能无限增加。所以在编程时，应该将传输速度，连接数量，应用的内存占用综合起来考虑。

软件库中，UDP socket 连接的内存分配采用静态方式，协议栈最多允许创建 2 个 UDP 连接，每一个 UDP 占用 4k 字节，

一共占用 8k 字节内存，此外，用于监听的 TCP socket 仅占用 256 字节的动态内存。

在软件库中，提供了两种方式来管理 TCP socket 连接的内存占用。在协议栈初始化之前因对这两种管理方式进行选择。

1. 静态分配（默认）

协议栈完成初始化后，立即分配一段静态的 48k 字节的内存空间，用于 socket 的发送缓冲和接收缓冲，每一个 socket 的发送缓冲和接收缓冲的大小都固定为 2k 字节，所以一共可以建立 12（48/4）个 socket 连接。除此之外，每一个 socket 连接建立时仍然会动态分配 256 字节的内存。

使用静态分配之后，用于应用的可用空间约 23k 字节左右。在该情况下，用户仅需要管理好用户应用中的内存消耗即可。

在静态分配的情况下，软件库可建立的最大的 TCP 连接数和每一个 TCP 连接的缓冲区大小可以在软件库初始化之前进行设置，以改变默认值。

2. 动态分配

协议栈完成初始化后，不会再分配额外的内存空间，协议栈在创建每一个 socket 连接时，根据用户设定的参数动态的分配内存，并且在 socket 关闭时释放。这些参数参数可以使用 setsockopt 函数来进行设置。由于用户自行设定 socket 的缓冲区大小，所以协议栈总共可以建立 socket 的连接数量不是固定的。如果剩余的内存空间不能满足新增的一个 socket 连接时，

- 如果调用 connect 函数来创建一个 socket，connect 函数会返回错误，
- 当用户使用一个 socket 监听 TCP 客户端的连接时，协议栈会直接拒绝新的客户端的连接请求，不会将客户端的连接事件提交给这个监听的 socket。

使用动态分配，用户要管理好 socket 连接的内存分配和用户应用程序的内存分配，并且防止内存碎片的产生，比静态分配的方式要复杂一些。为此，软件库提供了一些内存分析和管理的 API 函数，详情请参阅章节：2.7.1 内存调试

1.2.3 已使用的处理器资源

软件库在运行时会使用 STM32 的一些外设资源，现列出如下，在开发应用程序中引避免使用这些外设，或者使用软件库提供的 API 接口来操作这些外设。

外设	已使用的功能	模块	注释
SystemTick及其中断	为协议栈运行产生时基，设置的时钟周期是: 1ms	所有	请勿修改该外设参数，但仍可在其中断处理函数中增加简单功能。
SDIO	用于和Wi-Fi射频芯片通讯	所有	
DMA2 Stream 3	用于和Wi-Fi射频芯片通讯	所有	
EXTI Line13	用于和Wi-Fi射频芯片通讯	EMW3162	
PC2	看门狗信号输出	EMW3162	须通过lib_config函数开启看门狗输出
EXTI Line0	用于和Wi-Fi射频芯片通讯	EMW3161	
PB14	看门狗信号输出	EMW3161	须通过lib_config函数开启看门狗输出

当使用软件库中提供的 UART 相关函数时，如 SetUartPara, OpenUART 等。

外设	已使用的功能	模块	注释
UART1	模块的UART功能	EMW3162	仅在使用UART相关API时启用,
PA8, PA9, PA10, PA11	UART功能使用的引脚	EMW3162	使用模块引脚PIN20, 21, 22, 23
DMA2 Stream 2	模块的UART接收DMA功能	EMW3162	仅在使用UART相关API时启用
UART2	模块的UART功能	EMW3161	仅在使用UART相关API时启用
PA0, PA1, PA2, PA3	UART功能使用的引脚	EMW3161	使用模块引脚PIN45, 46, 47, 48
DMA1 Stream 5	模块的UART接收DMA功能	EMW3161	仅在使用UART相关API时启用

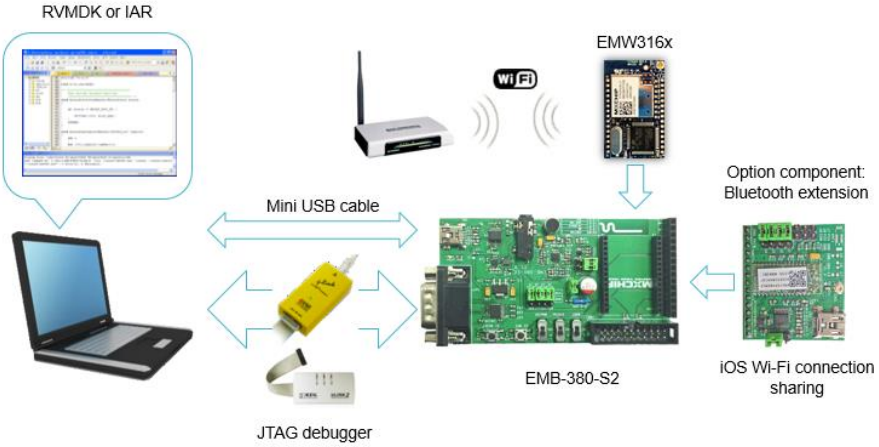
当使用软件库中提供的 NFC 相关函数时，如 OpenConfigmodeNFC 等。

外设	已使用的功能	模块	注释
IIC1	用于和NFC模块EMF2104通讯	所有	仅在使用NFC功能时启用
PB6, PB7, PB14	用于和NFC模块EMF2104通讯	EMW3162	使用模块引脚PIN1,2,13
EXTI Line14	用于和NFC模块EMF2104通讯	EMW3162	仅在使用NFC功能时启用

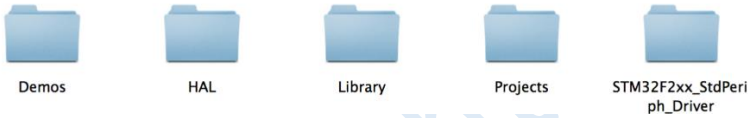
PB6, PB7, PB15	用于和NFC模块EMF2104通讯	EMW3161	使用模块引脚PIN6,28,29
EXTI Line15	用于和NFC模块EMF2104通讯	EMW3161	仅在使用NFC功能时启用

1.2.4 软件库内容和开发环境

使用 mxchipWNet 软件库进行开发时，硬件的连接方法如下：



MXCHIP 提供的 mxchipWNet 软件包中包含如下的文件夹



Demos 文件夹中包含了基于 mxchipWNet 软件库开发的多个示例程序，用于演示软件库的各种功能，为用户提供参考。

HAL 目录中包含了独立于 mxchipWNet 软件库功能以外的与平台相关的硬件配置，常用功能实现。

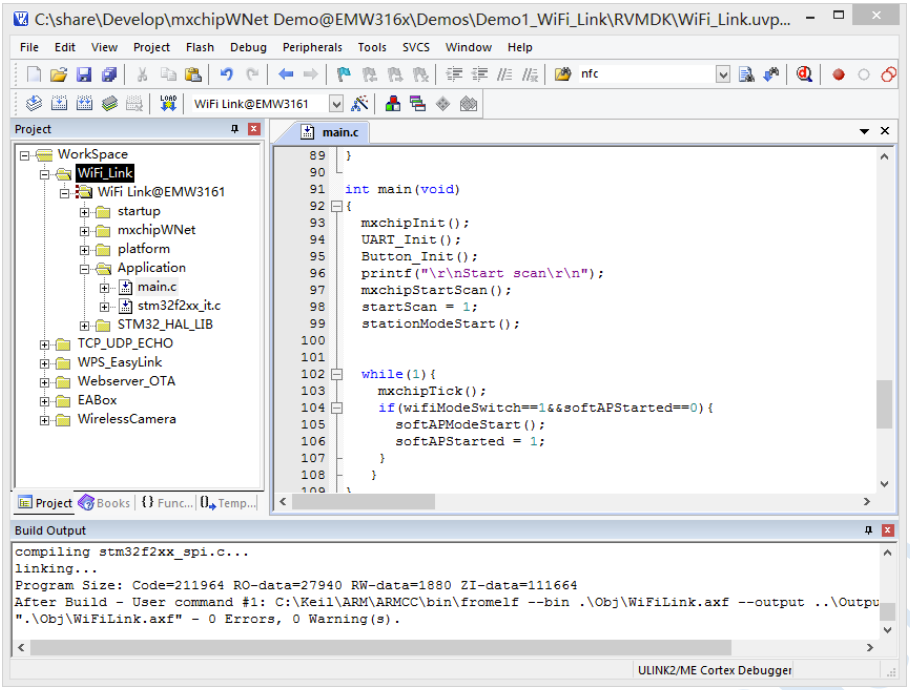
Library 目录中包含了可以在各个开发环境中使用的 mxchipWNet 软件库，及其头文件。

Projects 目录中包含了在常用的开发环境中可以直接打开的，配置好的示例工程。

STM32……目录中包含了 ST 提供的用于操作 STM32 外设的 API 接口函数及其源代码，mxchipWNet 软件库会调用某些接口函数。

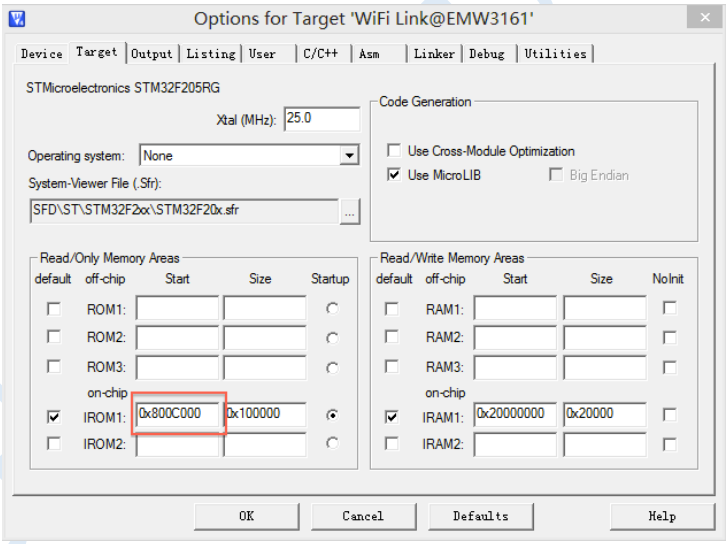
可以使用 KEIL（RVMDK）或者 IAR 开发基于该软件库的应用程序。用户可以打开/Projects 文件夹，打开相应的工程文件。

以 KEIL 为例：

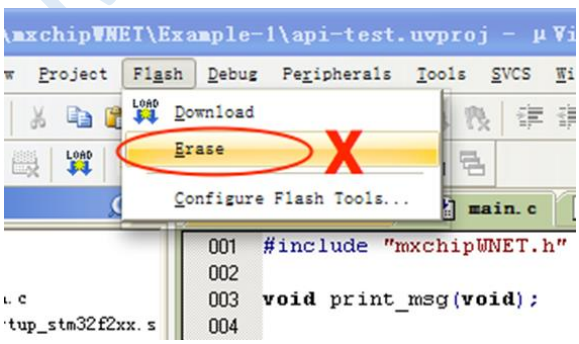


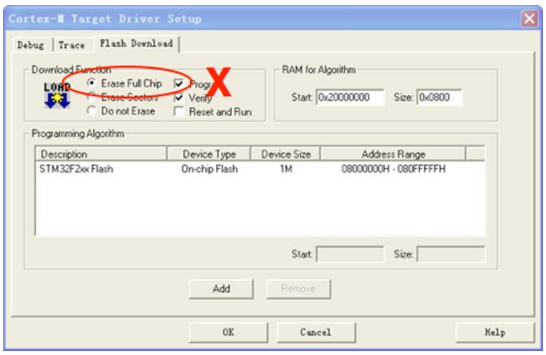
用户可以在 Project 窗口中选择当前使用的 Demo 工程， 然后进行编译和下载。在开发中有两点和普通的 STM32 单片机开发不同，并且需要特别注意：

1. 由于应用程序的起始地址不是 Flash 的起始地址，所以和普通的 STM32 程序的链接地址不同。以 KEIL 为例，应保证在 Project->Options for...->Target->IROM1 中的值由默认的 0x08000000 修改为 0x0800c000

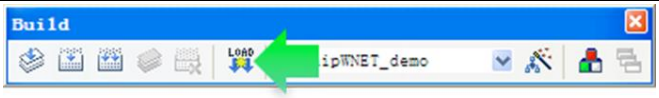


2. 由于模块中的 STM32 的 Flash 中包含多个分区，而这些分区中包含多种关键信息，如 Wi-Fi 射频驱动等。所以在将应用程序通过仿真器下载到模块中时要注意不要破坏其他分区中的信息。所以，在开发环境中对 Flash 的整片擦除操作是禁止的。

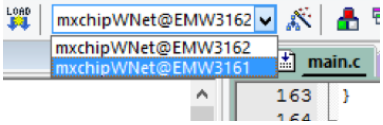




用户应通过下面的方法下载生成的模块应用程序：



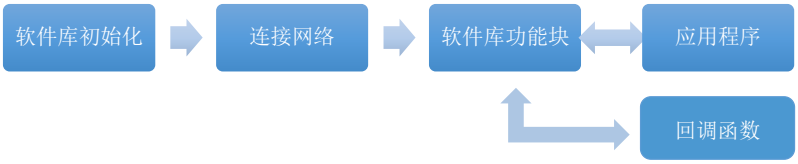
另外，请在一些 demo 可以同时在多种模块平台下使用，请在下图的位置选择正确的硬件平台：



1.3 基本运行模型

mxchipWNet 软件库不需要使用操作系统即可运行 ,但是软件库的功能块必须被周期地调用才能保证网络通讯和 Wi-Fi 网络能正常工作。软件库的功能块和用户的应用程序在一个循环中先后被调用。同时，在软件库的运行中会产生一些事件，如联网成功，网络断开，扫描完成等，这些事件都由都由软件库通过回调函数通知用户的应用程序。回调函数中的功能由用户实现。

软件库的工作流程如下：



通过调用 mxchipTick 函数来执行软件库的主要功能块。

如果用户的应用需要软件延时操作 ,不能马上调用软件库的主要功能块。用户应该选用软件库提供的延时函数 sleep ,msleep 函数，这些函数在完成延时功能同时也会执行软件库的主要功能块。

下面是一个简单的应用程序示例：

```
int main(void)
{
  ……初始化各个变量……
  mxchipInit();                                          // 初始化软件库

  memset(&wNetConfig, 0x0, sizeof(network_InitTypeDef_st));
  wNetConfig.wifi_mode = Station;
  strcpy((char*)wNetConfig.wifi_ssid,  "MXCHIP_RD" );    // 连接无线网络
  strcpy((char*)wNetConfig.wifi_key,  "stm32f215" );
  wNetConfig.dhcpMode = DHCP_Client;
  StartNetwork(&wNetConfig);

  while(1) {
    mxchipTick();                                        // 软件库功能块
    if (fd_udp==-1) {                                    // 应用程序示例，UDP 收发
      fd_udp = socket(AF_INET, SOCK_DGRM, IPPROTO_UDP);
      addr.s_port = 8090;
      bind(fd_udp, &addr, sizeof(addr));
    }
    FD_ZERO(&readfds);
    FD_SET(fd_udp, &readfds);
    select(1, &readfds, NULL, &exceptfds, &t);

    /*Read data from udp and send data back */
    if (FD_ISSET(fd_udp, &readfds)) {
      con = recvfrom(fd_udp, buf, 3*1024, 0, &addr, &addrLen);
      sendto(fd_udp, buf, con, 0, &addr, sizeof(struct sockaddr_t));
    }
  }
}
```

```
void WifiStatusHandler(int event)
{
  switch (event) {
    case MXCHIP_WIFI_UP:
      wifi_up = 1;
      break;
    case MXCHIP_WIFI_DOWN:
      break;
    default:
      break;
  }
  return;
}
```

Wi-Fi 连接状态回调函数，当 Wi-Fi 的连接状态改变时，该回调函数会自动执行

```
void NetCallback(net_para_st *pnet)
{
    printf("IP address: %s \r\n", pnet->ip);
    printf("NetMask address: %s \r\n", pnet->mask);
    printf("Gateway address: %s \r\n", pnet->gate);
    printf("DNS server address: %s \r\n", pnet->dns);
   printf("MAC address: %s \r\n", pnet->mac);
}
```

网络参数回调函数，当 Wi-Fi 连接成功后，该回调函数自动执行，向应用程序报告当前的 IP 地址等信息。

使用软件库时，有以下几点需要注意：

- 不要在回调函数中执行复杂的操作。
- 不要在硬件中断处理函数中调用软件库提供的 API 函数，尽量采用精简的中断处理函数。
- 软件库功能块（mxchipTick）调用时间间隔不要超过 500ms，使用 sleep 或者 msleep 函数实现应用程序的延时。
- 在执行 send，sendto 和 select 函数时也会自动执行软件库的功能块，所以在使用这些函数的时候也可以步调用 mxchipTick 函数来执行软件库功能块。

2 功能描述和 API 详解

2.1 数据类型

名称	数据长度	数据类型
u8	8bit	unsigned char
u16	16bit	unsigned short
u32	32bit	unsigned int
socklen_t	32bit	int
ssize_t	32bit	unsigned int

2.2 软件库初始化、功能块

2.2.1 常量

MxchipStatus

初始化软件库的结果

```
typedef enum {
MXCHIP_SUCCESS,
MXCHIP_FAILED,
MXCHIP_RF_INIT_FAILED,
MXCHIP_SYS_ILLEGAL,
} MxchipStatus;
```

Fields

MXCHIP_SUCCESS

软件库初始化成功

MXCHIP_FAILED

软件库初始化成功

MXCHIP_RF_INIT_FAILED,

射频初始化失败

MXCHIP_SYS_ILLEGAL

非法的硬件平台

2.2.2 数据类型

lib_config_t

软件库运行参数

```
typedef struct _lib_config_t {
   int tcp_buf_dynamic;
   int tcp_max_connection_num; /.
   int tcp_rx_size; // the default TCP read buffer size.
   int tcp_tx_size; // the default TCP write buffer size.
   int hw_watchdog;
   int wifi_channel;
}   lib_config_t;
```

Fields

tcp_buf_dynamic

0 表示使用静态内存分配，1 表示使用动态内存分配 （默认为 0 ）

tcp_max_connection_num

在静态分配内存的配置方式下，软件库允许的最大 TCP 连接数量 （默认为 12 ）

tcp_rx_size

在静态分配内存的配置方式下，每一个 TCP 连接接收缓冲区的大小（默认为 2048 ）

tcp_tx_size

在静态分配内存的配置方式下，每一个 TCP 连接发送缓冲区的大小（默认为 2048 ）

hw_watchdog

0 表示不是用看门狗信号输出，1 表示使用看门狗信号输出（默认为 1 ）

wifi_channel

0 支持通道 1-11（USA ），1 支持通道 1-13（CHN ），2 支持通道 1-14（JPN ）

2.2.3 函数

system_lib_version

功能：读取当前运行的软件库的版本

```
char* system_lib_version(void);
```

Return values：

指向保存软件库版本的字符串内存的指针

system_version(char *str, int len)

功能：回调函数，向软件库报告当前应用程序的版本

```
void system_version(char *str, int len);
```

Parameters：

str

指针，指向用于存放应用程序的版本字符串的内存的地址

len

指向用于存放应用程序的版本字符串的内存的长度

Discussion：

当软件库进入产测模式时（STATUS 引脚接地），软件库会通过这个回掉函数读取应用程序的版本号，并且通过串口输出。请参阅 mxchipInit 函数的说明。

lib_config

功能：设置软件库运行的参数

```
void lib_config(lib_config_t* conf)
```

Parameters：

conf

存放参数结构体的内存地址，请参阅数据类型：lib_config_t

Discussion：

该函数应在软件库初始化函数运行之前被调用，如不调用该函数，则默认采用静态分配内存的方式初始化软件库。

mxchipInit

功能：设置软件库运行的参数

```
MxchipStatus mxchipInit(void);
```

Return values：

软件库初始化的结果

Discussion：

在使用任何软件库提供的功能之前必须首先调用该函数完成软件库的初始化。如果在初始化过程中，如果检测到 STATUS 引脚是低电平，模块会进入产测模式，在串口上输出相应的测试信息。

STATUS 引脚定义：EMW3162 PIN30（PB9）EMW3161 PIN39（PF1）

STATUS 引脚的功能

STATUS 引脚	引导状况
0（低电平）	在初始化软件库时，进入产测模式，该函数不会返回
1（高电平）	该函数可以正常返回，使 CPU 执行之后的应用程序

mxchipTick

功能：执行软件库的功能块

```
void mxchipTick(void);
```

Discussion：

启动网络连接之后，如果没有网络数据的收发，则应该周期性的调用这个函数，以便维持网络连接，并且能够接收到无线网络上收到的数据。每次调用的时间间隔不应超过 500ms。此外如果有使用以下命令，则可以不调用该函数，因为这些函数在执行中也实现了软件库的功能块。

1、 sleep，msleep 函数用于延时

2、 send，sendto 函数用于发送网络数据

3、 select 函数用于查询 socket 的状态

2.3 Wi-Fi 无线网络的扫描，控制，连接和状态返回

2.3.1 数据类型

UwtPara_str

无线网络信息搜索结果

```
typedef  struct  _UwtPara_str  {
   char ApNum;
   ApList_str * ApList;
} UwtPara_str;
```

Fields

ApNum

搜索到的无线网络的数量

ApList

存放各个无线网络名称和信号强度的内存地址，请参阅数据类型 ApList_str。

ApList_str

无线网络信息搜索结果

```
typedef  struct  _ApList_str  {
      char ssid[32];
      char ApPower;
}ApList_str;
```

Fields

ssid

无线网络的名称

ApPower

无线网络的信号强度，以数值表示：0-100，数字越高信号强度越高。

network_InitTypeDef_st

无线网络运行参数

```
typedef struct _network_InitTypeDef_st
{
      char wifi_mode;
      char wifi_ssid[32];
      char wifi_key[32];
      char local_ip_addr[16];
      char net_mask[16];
      char gateway_ip_addr[16];
      char dnsServer_ip_addr[16];
      char dhcpMode;
      char address_pool_start[16];
      char address_pool_end[16];
      int wifi_retry_interval;
} network_InitTypeDef_st;
```

Fields

wifi_mode

无线网络的模式，0: Soft AP 模式， 1: Station 模式

wifi_ssid

无线网络的名称，在 Soft AP 模式，软件库会建立一个以该名称命名的无线网络，在 Station 模式，软件库会尝试加入以该名称命令的无线网络。

wifi_key

无线网络的密码，在 Soft AP 模式，软件库建立一个 WPA2 模式加密的无线网络，在 Station 模式，软件库会以自动匹配目标网络的加密方式。如果密码为空，则不进行加密。

local_ip_addr

模块的静态 IP 地址，以字符串格式输入，如："192.168.2.120"，如果开启 DHCP 客户端，则该参数可以为空。

net_mask

模块的子网掩码，以字符串格式输入，如："255.255.255.0"，如果开启 DHCP 客户端，则该参数可以为空。

gateway_ip_addr

模块的静态网关地址，以字符串格式输入，如："192.168.2.1"，如果开启 DHCP 客户端，则该参数可以为空。

dnsServer_ip_addr

DNS 服务器地址，以字符串格式输入，如："192.168.2.1"，如果开启 DHCP 客户端，则该参数可以为空。

dhcpMode

DHCP 模式，0: 表示关闭 DHCP， 1: 表示开启 DHCP 客户端， 2: 表示模块创建一个 DHCP 服务器。

address_pool_start

DHCP 地址池的起始地址，以字符串格式输入，如："192.168.2.100"，如果不使用 DHCP 服务器，则该参数可以为空。

address_pool_end

DHCP 地址池的结束地址，以字符串格式输入，如："192.168.2.200"，如果不使用 DHCP 服务器，则该参数可以为空。

wifi_retry_interval

station 模式下无法建立于 AP 的连接时，重新连接的时间间隔，如果设置为 0，则采用默认值 100ms。增加这个值可以降低连接频率，从而降低功耗，但会导致在网络恢复时，不能及时恢复网络连接。

sta_ap_state_t

无线网络信息搜索结果

```
typedef struct _sta_ap_state{
    int is_connected;
    int wifi_strength;
    u8   ssid[32];
    u8   bssid[6];
}sta_ap_state_t;
```

Fields

is_connected

当前模块和 AP 的连接状态, 0: 未连接， 1: 已连接

wifi_strength

当前无线网络连接的信号强度

ssid

当前无线网络连接的无线网络名称

bssid

当前无线网络连接的 BSSID

2.3.2 函数

mxchipStartScan

功能：使射频开始扫瞄所有频段上的无线网络

```
void mxchipStartScan(void)
```

Discussion：

由于搜索需要一段时间，所以软件库通过回调函数 ApListCallback 返回搜索的结果。

ApListCallback

功能：这是一个回调函数，用于返回模块搜索附近无线网络的结果

```
void ApListCallback(UwtPara_str *pApList)
```

Parameters：

pApList

存放无线网络信息的内存地址，请参阅数据类型 UwtPara_str。

Discussion：

该函数是回调函数，用户应在函数体中根据返回的无线网络信息实现用户自定义的功能。

StartNetwork

功能：启动无线网络连接

```
int StartNetwork(network_InitTypeDef_st* pNetworkInitPara)
```

Parameters：

pNetworkInitPara

存放待启动的无线网络的各项网络参数的内存地址，请参阅数据类型：network_InitTypeDef_st

Discussion：

该函数用于启动一个无线网络的连接，并且立即返回。无线网络的连接状态通过回调函数 WifiStatusHandler 中的参数表示。一旦一个无线网络已经启动，则由软件库自动管理这个无线网络，并且自动保持该无线网络的联通性，一旦网络断开，软件库会自动尝试重新连接，并且将无线网络的当前状态通过 WifiStatusHandler 提交给应用程序。

WifiStatusHandler

功能：这是一个回调函数，用于返回模块当前的无线网络连接的状态

```
void WifiStatusHandler(int status)
```

Parameters：

status

无线网络状态，0: 表示无线网络连接成功， - 1：表示无线网络连接断开

Discussion：

软件库在监测到无线网络的连接状态改变时调用这个函数，向应用程序报告当前的无线网络连接状态。应用程序通过该回调函数中的状态判断网络的状态，并执行相对应的操作。

wlan_disconnect

功能：断开无线网络连接

```
int wlan_disconnect(void);
```

Discussion：

断开无线网络时，用户也应该释放之前所有的网络 socket 连接以及相应的资源。软件库也不再会尝试重新连接。

wifi_power_down

功能：关闭 Wi-Fi 无线射频的电源

```
int wifi_power_down (void);
```

Discussion：

关闭无线网络时，用户也应该释放所有的网络 socket 连接以及相应的资源。射频的功耗

wifi_power_up

功能：开启 Wi-Fi 无线射频的电源

```
int wifi_power_up (void);
```

Discussion：

打开射频处理器的电源，并且执行初始化操作。

CheckNetLink

功能：查询当前 Wi-Fi 连接的状态

```
int CheckNetLink(sta_ap_state_t *ap_state)
```

Parameters：

ap_state

当前无线网络连接的网络状态信息

wifi_roam_trigger

功能：设置 Wi-Fi 漫游的参数

```
int wifi_roam_trigger(int dbm)
```

Parameters：

dbm

模块尝试切换 AP 时的信号强度

Discussion：

模块在连接上无线热点之后，会在后台不断检测当前的无线热点连接的信号的强度，当信号强度低于设置阈值，并且当前环境中有信号更好，并且使用同样的无线网络名称的无线热点，则模块会自动切换到这个新的无线热点上。

ps_enable

功能：使能射频的 IEEE power save 模式

```
void ps_enable(void)
```

Discussion：

打开该功能之后，软件库会在 station 模式并且成功地连接到 AP 的情况下，自动打开射频的 IEEE power save 模式以降低功耗。

ps_disable

功能：关闭射频的 IEEE power save 模式

```
void ps_disable (void);
```

OpenConfigmodeWPS

功能：开启 WPS（Wi-Fi protected Setup）配置功能

```
int OpenConfigmodeWPS(int timeout)
```

Parameters：

timeout

WPS 配置超时时间，单位：秒。如果在这个时间段内没有完成 WPS 配置功能，软件库会通过回调函数返回一个错误的状态。并且停止当前的 WPS 配置进程。

Discussion：

该函数用于启动 WPS 配置功能 ,中止当前所有网络通讯 ,并且立即返回。WPS 的配置结果通过回调函数 RptConfigmodeRslt 返回给应用程序。

CloseConfigmodeWPS

功能：立即关闭当前的 WPS（Wi-Fi protected Setup）配置功能

```
int CloseConfigmodeWPS(void)
```

OpenEasylink，OpenEasylink2

功能：开启 EasyLink 配置功能

```
int OpenEasylink(int timeout)
int OpenEasylink2(int timeout)
```

Parameters：

timeout

EasyLink 配置超时时间，单位：秒。如果在这个时间段内没有完成 EasyLink 配置功能，软件库会通过回调函数返回一个错误的状态。并且停止当前的 EasyLink 配置进程。

Discussion：

该函数用于启动 Easylink 配置功能，中止当前所有网络通讯，并且立即返回。Easylink 的配置结果通过回调函数 RptConfigmodeRslt 返回给应用程序。

CloseEasylink, CloseEasylink2

功能：立即关闭当前的 EasyLink 配置功能

```
int CloseEasylink(void)
int CloseEasylink2(void)
```

RptConfigmodeRslt

功能：这是一个回调函数，用于返回模块启动各种配置功能后的配置结果

```
void RptConfigmodeRslt(network_InitTypeDef_st *nwkpara);
```

Parameters：

nwkpara

用于存放配置结果的内存的地址。仅使用 network_InitTypeDef_st 结构体中的 wifi_ssid 和 wifi_key 表示配置的无线网络名称和密码。结构体中的其他参数在本回调函数中没有意义。如果等于 0，表示当前的配置操作超时了，或者由于在配置过程中产生错误导致了失败。

Discussion：

WPS，EasyLink，Bluetooth 以及 NFC 配置功能均通过这个回调函数向用户的应用程序报告配置的结果。

2.4 基本网络服务

2.4.1 常量

WiFi_Interface

初始化软件库的结果

```
typedef enum {
  Soft_AP,
  Station
} WiFi_Interface;
```

Fields

Soft_AP

Soft AP 端的网络接口

Station

Station 端的网络接口

2.4.2 数据结构

net_para_st

模块的网络参数

```
typedef struct _net_para {
  char dhcpMode;
  char local_ip_addr[16];
  char gateway_ip_addr [16];
  char net_mask[16];
  char dnsServer_ip_addr[16];
  char mac_addr [16];
  char broadcast_ip_addr [16];
} net_para_st;
```

Fields

dhcpMode

DHCP 模式，0: 表示关闭 DHCP， 1: 表示开启 DHCP 客户端， 2: 表示模块创建一个 DHCP 服务器

local_ip_addr

模块当前的 IP 地址，字符串格式，如："192.168.2.120"。

gateway_ip_addr

模块当前的网关地址，字符串格式，如："192.168.2.1"。

net_mask

模块当前的子网掩码，字符串格式，如："255.255.255.0"。

dnsServer_ip_addr

模块当前的 DNS 服务器地址，字符串格式，如："192.168.2.1"。

mac_addr

模块的 MAC 地址，字符串格式，如："7E0000001111"。

broadcast_ip_addr

模块当前的广播地址，暂无功能，保留。

Discussion：

模块当前的网络状态可以通过可以通过 getNetPara 函数查询，也可以通过回调函数 NetCallback 获得，该回调函数在模块成功连接到网络后会被软件库被自动调用。

2.4.3 函数

getNetPara

功能：查询模块当前的网络参数

```
int getNetPara(net_para_st * pnetpara， WiFi_Interface iface);
```

Parameters：

pnetpara

指向用于存放获得的网络参数结构体的指针。

iface

指定从那一个网络接口获取网络参数

Discussion：

该函数会立即返回，不需要通过回调函数 **NetCallback** 来获得相关的参数。

NetCallback

功能：DHCP 成功时，返回当前模块网络参数的回调函数

```
void NetCallback(net_para_st *pnet)
```

Parameters：

Pnetpara

指向用于存放获得的网络参数结构体的指针

Discussion：

在开启 DHCP 客户端的模式下，模块成功连接到无线网络后会启动 DHCP 协商，此外如果调用 ReallocIP 函数也会重新启动 DHCP 协商，DHCP 协商成功后软件库会产生这个回调，用户可以通过访问该函数的参数可以立即获得模块当前的网络参数。

gethostbyname

功能：以阻塞的方式，通过 DNS 服务器获取域名对应的 IP 地址

```
int gethostbyname(const u8 * hostname, u8 * ip_addr, u8 ipLength);
```

Parameters：

hostname

指向存放域名的字符串的指针

ip_addr

指向存放 IP 地址字符串的指针。IP 地址的格式类似于：“202.120.23.120”

ipLength

用于存放 IP 地址的字符串内存块的长度

Return values：

该函数运行的结果： - 1：表示没有成功获取 IP 地址， 0：标示获取 IP 地址成功

Discussion：

该函数是阻塞运行的，超时时间是 5s。如果在这段时间内成功获取了 IP 地址，则将这个 IP 地址存放在地址是 ip_addr，长度 ipLength 的内存块中。

dns_request

功能：以以非阻塞的方式，启动软件库的 DNS 解析服务

```
u32 dns_request(char *hostname);
```

Parameters：

hostname

指向存放域名的字符串的指针

Return values：

该函数运行的结果：-1：启动 DNS 解析服务失败， 0：启动 DNS 解析服务器成功，等待 DNS 回调函数的返回， >0：直接获得了域名对应的 IP 地址（以一个 32 位数表示的 IP 地址），这种情况可能发生的情况是：

- DNS 结果已经保存在了软件库中
- 输入的域名字符串实际上就是一个 IP 地址

Discussion：

启动 DNS 解析服务之后，需要通过回调函数 dns_ip_set 来获得对应的 IP 地址。在软件库的处理过程中，可以执行其他的代码。

dns_ip_set

功能：用于返回 DNS 解析结果的回调函数

```
void dns_ip_set(u8 *hostname, u32 ip);
```

Parameters：

hostname

指向存放域名的字符串的指针，表示 DNS 结果对应的域名

ip

DNS 解析的结果 IP 地址，如果 IP = -1，说明 DNS 解析失败

ReallocIP

功能：通过 DHCP 协议重新获取模块的 IP 地址等参数

```
int gethostbyname(const u8 * name, u8 * ip_addr, u8 ipLength);
```

Discussion：

请在 DHCP 客户端开启的情况下使用这个功能。

2.5 类 BSD Socket 网络数据传输

2.5.1 常量

名称	值	含义
AF_INET	2	BSD socket协议簇
SOCK_STREAM	1	是有保障的面向连接的套接字
SOCK_DGRM	2	是无保障的面向消息的套接字
IPPROTO_TCP	6	TCP协议
IPPROTO_UDP	17	UDP协议
INADDR_ANY	0	表示任意的一个IP地址
INADDR_BROADCAST	0xFFFFFFFF	局域网中的广播IP地址
SOL_SOCKET	1	软件库仅支持在套接字级别上设置选项

SOCK_OPT_VAL

socket 的参数

```
typedef enum {
        SO_REUSEADDR = 0x0002,
        SO_BROADCAST = 0x0006,
        IP_ADD_MEMBERSHIP = 0x0003,
        IP_DROP_MEMBERSHIP = 0x0004,
        SO_BLOCKMODE = 0x1000,
        SO_SNDTIMEO = 0x1005,
        SO_RCVTIMEO =0x1006,
        SO_CONTIMEO =0x1007,
        SO_RDBUFLEN = 0x1008,
        SO_WRBUFLEN = 0x1009,
} SOCK_OPT_VAL;
```

Fields

SO_REUSEADDR

设置允许地址重用，不管设置与否，软件库始终允许

SO_BROADCAST

设置是否允许广播，不管设置与否，软件库始终允许

IP_ADD_MEMBERSHIP

加入组播组

IP_DROP_MEMBERSHIP

退出组播组

SO_BLOCKMODE

设置 socket 的阻塞 / 非阻塞模式，在默认情况下，软件库使用阻塞模式

SO_CONTIMEO

在阻塞模式下，连接到 TCP 服务器的等待超时时间，默认时间：10 秒

SO_SNDTIMEO

在阻塞模式下，socket 发送数据的等待超时时间

SO_RCVTIMEO

在阻塞模式下，socket 接收数据的等待超时时间

SO_RDBUFLEN

socket 的接收缓冲区长度，单位：字节，默认长度是 2048 字节

SO_WRBUFLEN

socket 的发送缓冲区长度，单位：字节，默认长度是 2048 字节

2.5.2 数据结构

sockaddr_t

在 socket 编程中用于描述远程或者本地的端点地址

```
struct sockaddr_t {
    u16   s_type;
    u16   s_port;
    u32   s_ip;
    u16   s_spares[6];
} ;
```

Fields

s_type

地址的类型，只能设定为 AF_INET

s_port

socket 通讯的端口。

s_ip

以一个 int 类型表示的 IP 地址。可以用 inet_addr 函数将字符型的 IP 地址转换得到。

s_spares

没有意义，仅用于占位。

timeval_t

用于表示一段时间间隔

```
struct timeval_t {
        unsigned long    tv_sec;
        unsigned long    tv_usec;
};
```

Fields

tv_sec

时间，单位：秒

tv_usec

时间，单位：微秒

Discussion：

由于软件库能够处理的最小时间单位是毫秒，所以，尽管在结构体中可以设置微秒，但是软件库是将设定的时间做进位处理，换算成毫秒。此外，如果结构体中的两个数值都是 0，则表示没有时间间隔。

fd_set

套接字集合

```
typedef struct fd_set {
   fd_mask fds_bits[howmany(FD_SETSIZE, NFDBITS)];
} fd_set;
```

Fields

fds_bits

套接字集合数组

Discussion：

可以用 select 函数查询套接字集合中的各个套接字的状态

2.5.3 函数

socket

功能：为通讯创建一个端点，为套接字 socket 返回一个文件描述符。

```
int socket(int domain, int type, int protocol);
```

Parameters：

domain

该软件库只能使用 AF_INET 参数，表示使用 IPV4 协议

type

套接字的类型：SOCK_STREAM：是有保障的面向连接的套接字，SOCK_DGRAM：是无保障的面向消息的套接字

protocol

指定实际使用的传输协议： IPPROTO_TCP：TCP 协议，IPPROTO_UDP：UDP 协议

Return values：

创建的套接字的文件描述符。之后对该套接字的所有操作都使用这个文件描述符来表示这个套接字。如果返回 0，则表示无法建立该套接字。

Discussion：

套接字类型和 type 和传输协议 protocol 是对应的，SOCK_STREAM 必须和 IPPROTO_TCP 配对使用，SOCK_DGRAM 和 IPPROTO_UDP 配对使用。此外，调用该函数时，软件库会为这个套接字开辟一块长度是 256 字节的内存区域用于存放套接字的状态信息。

setsockopt

功能：设置套接字的运行参数。

```
int setsockopt(int sockfd, int level, int optname,const void *optval, socklen_t optlen)
```

Parameters：

sockfd

套接字的文件描述符

level

设置参数的级别

optname

设置的参数名称，参阅常量 SOCK_OPT_VAL

optval

指向存放参数的指针

optlen

参数的数据长度，单位：字节

Return values：

0：表示连接成功， - 1：表示连接失败

bind

功能：将套接字绑定到网络地址上。

```
int bind(int sockfd, const struct sockaddr_t *addr, socklen_t addrlen)
```

Parameters：

sockfd

套接字的文件描述符

addr

指向存放网络地址的指针

addrlen

地址参数的长度，单位是字节

Return values：

0：表示连接成功， - 1：表示连接失败

Discussion：

在实际的函数调用中，本函数主要用于设定本地的端口，如 UDP 通讯的本机端口，TCP 服务器的监听端口。

connect

功能：建立与远程设备的套接字连接

```
int connect(int sockfd, const struct sockaddr_t *addr, socklen_t addrlen)
```

Parameters：

sockfd

套接字的文件描述符

addr

指向存放远程设备的网络地址的指针

addrlen

远程设备的地址的数据长度，单位是字节

Return values：

0：表示连接成功， - 1：表示连接失败

Discussion：

即使远程设备不可达，UDP 协议下本函数仍然能正确得返回。

在阻塞模式下使用 TCP 协议连接 TCP 服务器，软件库会在该函数执行时尝试与 TCP 服务器建立连接，并且有 10 秒（可以通过 setsockopt 修改）的超时时间。但是在非阻塞模式下，该函数会立即返回，但并不表示已经建立了连接。当连接成功后，软件库会通过回调函数 socket_connected 通知用户的应用程序。

在动态分配内存的情况下调用该函数时，软件库会按照设置的值自动分配发送缓冲区和接收缓冲区。如果没有足够的内存空间，本函数会返回失败。

socket_connected

功能：当一个 socket 连接成功后，产生的回调函数

```
void socket_connected(int fd);
```

Parameters：

fd

连接成功的套接字的文件描述符

Discussion：

当一个非阻塞的套接字调用 connect 函数尝试 TCP 服务器建立连接时，软件库通过这个回调函数向应用程序反馈该套接字是否连接成功。

listen

功能：在套接字上进行监听，即创建一个 TCP 监听服务器

```
int listen(int sockfd, int backlog)
```

Parameters：

sockfd

套接字的文件描述符

backlog

在本软件库中，该参数没有意义，仅为了和标准 socket 接口函数保持一致，写入 0 即可

Return values：

0：表示监听成功， - 1：表示监听失败

Discussion：

当一个套接字用于监听时，软件库不会为该 socket 分配发送和接收缓冲区。但是通过这个监听套接字使用 accept 函数创建的各个套接字（TCP 客户端链接）的发送和接收缓冲区的大小等于监听套接字设定的参数。

accept

功能：接收远程设备的连接请求，创建链接并返回该套接字的文件描述符

```
int accept(int sockfd, struct sockaddr_t *addr, socklen_t *addrlen)
```

Parameters：

sockfd

套接字的文件描述符

addr

指向用于存放远程设备网络地址的内存块的指针，本函数会在该地址上写入远程设备的网络地址数据

addrlen

指向用于存放远程设备网络地址的数据长度的内存指针，本函数会在该地址上写入远程设备的网络地址数据的实际的数据长度

Return values：

创建的套接字的文件描述符。之后对该套接字的所有操作都使用这个文件描述符来表示这个套接字。如果返回 0，则表示无法建立该套接字。

Discussion：

与标准 socket 编程不同的是，接受远程设备的连接请求是由软件库自动完成的。只要有足够的内存空间，就能允许远程设备的连接，并且创建这个链接。使用 accept 函数仅仅是为了获得这个套接字的文件描述符以及远程设备的网络地址。如果用户不希望建立这个连接，只能在 accept 调用之后，再使用 close 函数将这个套接字关闭掉。

select

功能：获取套接字的状态

```
int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval_t *timeout)
```

Parameters：

sockfd

套接字的文件描述符

readfds

指向用于检测是否可以读取数据的套接字集合的指针（可选）

writefds

指向用于检测是否可以写入数据的套接字集合的指针（可选）

exceptfds

指向用于检测是出现异常的套接字集合的指针（可选）

timeout

在阻塞模式下，该函数执行的等待超时时间，如设为 0，表示立即返回，不等待

Return values：

0：表示执行成功，-1：表示执行失败

Discussion：

在阻塞模式下，如果在接收和发送数据之前能够首先判断套接字的状态，根据套接字的状态执行数据的收发，那么可以避免在执行数据收发的相关函数时产生等待。在使用 select 函数之前，应该先将需要查询的套接字存放在对应的套接字集合中，在 select 执行之后，通过检测套接字集合中的对应套接字是否被设置，来判断当前的状态，以便执行对应的操作。

如：readfds 集合中的套接字被置位，说明该套接字接收到了数据，或者需要关闭，所以需要通过 recv 等用于接收数据的函数来接收相关的数据。

tx_buf_size

功能：获取指定套接字的可用的发送缓冲大小

```
int tx_buf_size(int sockfd)
```

Parameters：

sockfd

套接字的文件描述符

Return values：

可用的发送缓冲大小

Discussion：

如果套接字的发送缓冲区大小少于需要发送的数据长度，则有可能在发送函数时，无法将需要发送的所有数据一次性写入。在这种情况下，用户需要多次调用发送函数，才能将所有数据发送出去。所以发送数据前可以调用这个函数来判断对应的套接字是否有足够的发送缓冲区，如果缓冲区足够大，就可以通过发送函数一次性将数据写入对应的发送缓冲。

send，write, sendto

功能：通过套接字发送数据（写入套接字的发送缓冲区）

```
ssize_t send(int sockfd, const void *buf, size_t len, int flags)
int write(int sockfd, void *buf, size_t len)
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,const struct sockaddr_t *dest_addr, socklen_t addrlen)
```

Parameters：

sockfd

套接字的文件描述符

buf

指向待发送数据的内存指针

len

发送数据的长度

flags

在本函数库中没有功能，仅为了和标准 BSD socket 的相关 API 兼容，写入 0 即可

dest_addr

指向数据发送的目的地址的指针

addrlen

数据发送的目的地址的数据长度

Return values：

实际写入套接字的发送缓冲区的数据长度

Discussion：

当一个使用 UDP 协议的套接字在调用了 connect 函数之后才能使用 send 和 write 来发送数据，否则只能使用 sendto。使用 TCP 协议的套接字只能使用 send 和 write 来发送数据。

在 TCP 协议下在使用数据发送相关函数时，软件库会将这些数据保存在套接字的发送缓冲区中，并且通过协议栈和 Wi-Fi 发送这些数据，如果这些数据不能全部发送，则仍然将这些数据保存在发送缓冲区中，等下一次软件库的运行时再次尝试发送。所以这些函数的返回值实际上是将数据保存在发送缓冲区中的数据长度，而不是通过网络已经成功发送出去的数据的长度。

recv，read, recvfrom

功能：通过套接字发送数据

```
ssize_t recv(int sockfd, void *buf, size_t len, int flags)
int read(int sockfd, void *buf, size_t len)
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr_t *src_addr, socklen_t *addrlen)
```

Parameters：

sockfd

套接字的文件描述符

buf

指向存放接收到的数据的内存的指针

len

接收的数据的长度，该长度值应小于或等于存放接收数据的内存的大小

flags

在本函数库中没有功能，仅为了和标准 BSD socket 的相关 API 兼容，写入 0 即可

src_addr

指向接收数据的源地址的指针

addrlen

数据接收的源地址的数据长度

Return values：

接收到的数据长度。如果使用 select 确认套接字已经接收到数据，但是返回值是 0，说明当前套接字出现了异常，需要关闭

Discussion：

当一个使用 UDP 协议的套接字在使用 recvfrom 来接收数据时，可以获取发送方的 IP 地址和发送端口。而 TCP 协议下则不应使用 recv 和 read 来接收数据。

close

功能：关闭一个套接字

```
int close(int fd);
```

Parameters：

sockfd

套接字的文件描述符

Return values：

0：表示关闭成功， - 1：表示关闭失败

Discussion：

在本软件库中，本函数总能正确得关闭。在 TCP 协议下，如果远程设备没有及时地应答模块的关闭请求，则软件库会等待一个超时时间 2 秒，之后释放掉模块端的套接字资源。

FD_ZERO(p)

清空指针 p 指向的套接字集合

FD_SET(n, p)

在指针 p 指向的套接字集合中增加套接字 n

FD_CLR(n, p)

在指针 p 指向的套接字集合中删去套接字 n

FD_ISSET(n, p)

查看在指针 p 指向的套接字集合中的套接字 n 是否被置位，返回 1 表示置位，返回 0 表示未被置位

set_tcp_keepalive

功能：设置 TCP 通讯时，keepalive 收发的参数

```
void set_tcp_keepalive(int num, int seconds)
```

Parameters：

num

在断开 TCP 连接之前，没有正确收到 keepalive 包的数目

seconds

模块在 TCP 连接上没有收到数据时，周期性地发送 keepalive 数据包的时间间隔

Discussion：

TCP 连接在通讯时可能产生异常，如远程设备断网，断电等，而在 TCP 链路上没有正常的数据通讯。为了及时检测到 TCP 的异常状态，模块在 TCP 链路上没有正常的数据通讯时，会周期性自动发送 keepalive 数据包，远程设备应该应答。如果连续的几次 keepalive 数据包都没有正确应答，模块会自动释放当前 TCP 连接的资源，也会通过 recv（接收状态置位，但是 recv 等函数的返回值是 0）等相关函数通知用户的应用程序，使应用程序也做出对应的操作。

此外，如果 TCP 产生异常，而模块在该 TCP 上有数据需要发送。软件库会尝试发送这些数据，直到 2 分钟超时后，也会自动关闭这个 TCP 连接。

get_tcp_keepalive

功能：获取 TCP 通讯时，keepalive 收发的参数

```
void get_tcp_keepalive(int *num, int *seconds)
```

Parameters：

num

一个指针，指向的数据的含义是在断开 TCP 连接之前，没有正确收到 keepalive 包的数目

seconds

一个指针，指向的数据的含义是模块在 TCP 连接上没有收到数据时，周期性地发送 keepalive 数据包的时间间隔

Discussion：

参考函数：set_tcp_keepalive

setSslMaxlen

功能：设置用于保存待解密数据的内存空间大小

```
int setSslMaxlen(int len);
```

Parameters：

len

一块内存空间的大小

Return values：

0：表示设置成功，-1：表示设置失败

Discussion：

SSL 加密过的数据必须解密后才能够提交给上层处理，这些未解密的数据必须保存在一个独立的内存空间中，这个内存空间的大小必须大于在通讯中可能使用到的最大的数据块的长度。

getSslMaxlen

功能：获取用于保存待解密数据的内存空间大小

```
int getSslMaxlen(void)
```

Return values：

应用于保存待解密数据的内存的大小

Discussion：

参考函数：setSslMaxlen。

setSSLmode

功能：初始化 TCP 连接的 SSL 加密连接

```
int setSSLmode(int enable, int sockfd)
```

Parameters：

enable

一块内存空间的大小

sockfd

1: 开启 SSL， 0: 关闭 SSL

Return values：

0：SSL 通讯初始化完成， - 1：SSL 通讯初始化失败

Discussion：

在一个已经成功建立连接的 TCP 连接上启动 SSL 数据加密通讯。一旦初始化成功，通过这个 TCP 连接发送的数据都会进行 SSL 加密。同时，接收到的加密数据，协议栈会首先解密，用户再使用接收函数来获得解密后的数据。

2.6 STM32 的外部中断配置和使用

由于软件库运行时需要用到一部分 STM32 的外部中断线，为了使用户设置的外部中断线和软件库的相关部分能够共存，所以软件库提供了一套 API 函数 ,将软件库和用户端对外部中断的使用统一起来。具体的应用范例请参阅 Demo1。基本流程如下：

1. 实现中断处理函数，但不需要在终端处理函数中对中断相关硬件的设置，这部分设置由软件库实现
2. 使用 gpio_irq_enable 函数设置并使能外部中断，并将中断处理函数和终端源关联起来
3. 在 STM32 提供的终端处理函数，如 EXTI3_IRQHandler 中添加函数 gpio_irq()

2.6.1 常量

gpio_irq_trigger_t

外部中断的触发方式

```
typedef enum{
    IRQ_TRIGGER_RISING_EDGE  = 0x1,
    IRQ_TRIGGER_FALLING_EDGE = 0x2,
    IRQ_TRIGGER_BOTH_EDGES   = IRQ_TRIGGER_RISING_EDGE | IRQ_TRIGGER_FALLING_EDGE,
} gpio_irq_trigger_t;
```

Fields

IRQ_TRIGGER_RISING_EDGE

上升沿触发

IRQ_TRIGGER_FALLING_EDGE

下降沿触发

IRQ_TRIGGER_BOTH_EDGES

上升沿或下降沿触发

2.6.2 函数

gpio_irq_enable

功能：设置并使能外部中断

```
int gpio_irq_enable(GPIO_TypeDef* gpio_port, uint8_t gpio_pin_number, gpio_irq_trigger_t trigger,
gpio_irq_handler_t handler, void* arg )
```

Parameters：

gpio_port

用于外部中断的 STM32 的 GPIO 的端口，如 GPIOA，GPIOB 等

gpio_pin_number

用于外部中断的 STM32 的 GPIO 的序号，0-15

trigger

触发中断的电平变化方式

handler

指向中断处理函数的指针

arg

指向中断处理函数输入参数的指针

Return values：

0：表示执行成功， - 1：表示执行失败

Discussion：

使用该函数开启外部中断后，用户在自定义的中断函数 handler 中完成中断处理函数，在该用户完成的函数中，不需要对 STM32 的中断相关的硬件做任何操作。但是需要在 ST 软件库提供的底层中断处理函数中添加 gpio_irq 函数，以便让软件库处理对应的底层中断，并切换到用户自定义的中断处理函数中。

gpio_irq_disable

功能：关闭外部中断

```
int gpio_irq_disable( GPIO_TypeDef* gpio_port, uint8_t gpio_pin_number )
```

Parameters：

gpio_port

用于外部中断的 STM32 的 GPIO 的端口，如 GPIOA，GPIOB 等

gpio_pin_number

用于外部中断的 STM32 的 GPIO 的序号，0-15

Return values：

0：表示执行成功， - 1：表示执行失败

2.7 工具

2.7.1 数据格式转换

inet_ntoa

功能：将 int 型的 IP 地址转换成字符串型

```
char *inet_ntoa( char *s, u32 x )
```

Parameters：

s

指向存放输出的字符串型的 IP 地址的指针

x

int 型 IP 地址

Return values：

与 s 的值一样

Discussion：

该函数与 PC 上普遍使用的 inet_ntoa 函数稍有区别，原始的 inet_ntoa 函数将结果保存在套接字的静态内存中，所以没有参数 char *s，而在本软件库中的，存放结果的内存空间是由用户代码提供的。

inet_addr

功能：将 int 型的 IP 地址转换成字符串型

```
u32 inet_addr(char *s)
```

Parameters：

s

指向存放待转换的字符串型的 IP 地址的指针

Return values：

int 型 IP 地址

ntohs，ntohl

功能：将网络字节序转换成主机字节序

```
u16 ntohs(u16 n);
u32 ntohl(u32 n);
```

Parameters：

n

待转换的以网络字节序排列的数据

Return values：

输出以主机字节序排列的数据

htons，htonl

功能：将网络字节序转换成主机字节序

```
u16 htons(u16 n);
u32 htonl(u32 n);
```

Parameters：

n

待转换的以网络字节序排列的数据

Return values：

输出以主机字节序排列的数据

2.7.2 定时相关

MsTimer

这是一个软件库维护的系统时间，单位是毫秒，有软件库初始化时开始计时。通常可以用这个系统时间计算一个时间间隔。如果用户的应用程序需要读取这个参数，可以在代码中定义 extern int MsTimer，就可以读取这个变量中的数据了。

sleep，msleep

功能：延时等待一段时间

```
int sleep(int seconds);
int msleep(int mseconds);
```

Parameters：

seconds

等待的时间，单位为秒

mseconds

等待的时间，单位为毫秒

Discussion：

在等待过程中，仍然会执行软件库的主功能块，所以可以保持网络的连接和数据的通讯。

SetTimer

功能：延时执行一个函数

```
int SetTimer(unsigned long ms, void (*psysTimerHandler)(void))
```

Parameters：

ms

延时等待的时间

psysTimerHandler

指向待执行的函数的指针

Return values：

0：表示执行成功， - 1：表示执行失败

Discussion：

当到达设定的时间后，软件库会以回调函数的方式来调用用户定义的函数。如果在这个回调函数中再次使用 SetTimer 函数再次将当前的函数进行延时执行，则可以实现周期行地执行用户定义的函数。

WatchDog

功能：该回调函数会被软件库周期性地调用

```
void WatchDog(void)
```

Discussion：

在函数库运行时，软件库会以 100ms 的周期周期性的调用这个回调函数。所以用户可以在这个函数中对看门狗进行喂狗，也可以驱动一个外部的看门狗。但是如果软件库的主功能块的调用时间较长（参阅 mxchipTick 函数），会导致该回调函数的调用不够及时。

2.7.3 调试工具

get_tcp_clients

功能：查询当前的 TCP 客户端数量

```
int get_tcp_clients(void);
```

Return values：

当前 TCP 客户端的数量

Discussion：

由于用户可能在软件库释放 TCP 链接时，没有正确地释放对应的资源，如：文件描述符等。使用这个函数来查询软件库当前维护的 TCP 客户端数量，和当前用户应用中的数量进行比对，以便找出这些问题。

memory_status

功能：查询当前的空闲内存空间

```
void memory_status(int *total_free, int *max_len)
```

Parameters：

total_free

指向当前所有空闲内存空间大小的指针

max_len

指向当前最大的空闲内存块的长度

Discussion：

当到达设定的时间后，软件库会以回调函数的方式来调用用户定义的函数。如果在这个回调函数中再次使用 SetTimer 函数再次开启了一个延时执行的函数，则可以周期地执行用户定义的函数。

2.7.4 其他

md5_hex

功能：计算一段数据的 md5 校验码

```
void md5_hex(u8 *input, u32 len, u8 *output)
```

Parameters：

input

指向待计算数据的指针

len

待计算数据的长度

output

指向存放输出结果的指针

Discussion：

由于生成的 md5 码由 32 个 16 进制数组成，所以用户需要为输出结果提供 16 个字节的内存空间（output 参数指向的内存空间），否则会引起内存的错误。

3 功能实现和应用范例

3.1 Demo1: Wi-Fi 网络的搜索、连接、切换

3.2 Demo2: TCP UDP 网络数据传输

3.3 Demo3: WPS 和 EasyLink 配置

3.4 Demo4: 网页服务器和 OTA 远程升级

3.5 应用范例: mxchipWNet-HA 家庭智能设备通讯协议范例

3.6 应用范例: Wi-Fi 控制的电器插座

3.7 应用范例: 无线网络监控摄像头

4 升级历史

版本 1.0

- ✓ 初始版本

版本 1.10

- ✓ 开发环境升级到 RVMDK 5.0
- ✓ 修复了模块上的 led 灯显示不正常的问题
- ✓ 增加了对射频芯片状态异常的检测功能
- ✓ 提供了看门狗信号输出的功能，用户可以外接看门狗芯片，通过 lib_config 函数来进行设置
- ✓ 更新了 EMW3162 库中的 NVRAM 内容，提高模块的射频性能
- ✓ 增加了设置无线漫游阈值的函数：int wifi_roam_trigger(int dbm);
- ✓ 修复了软件库在判断 socket 的文件描述符是否可以使用时存在的 bug
- ✓ getNetPara 函数增加了用于选择 station 模式和 soft AP 模式接口的参数
- ✓ 修复了 CheckNetLink 函数的功能，现在可以正常显示当前连接的信号强度了
- ✓ 修正了一个可能导致 socket 文件描述符被错误地重用的问题
- ✓ 可以通过两次调用 StartNetwork 函数分别启动 station 模式和 Soft AP 模式，来使得两个模式共存了
- ✓ 在 Station 和 Soft AP 共存的模式下，Station 的重连间隔时间固定为 20 秒
- ✓ 调整了扫描参数，尽可能减少扫描对正常通讯的影响
- ✓ 在回调函数 WifiStatusHandler 中增加了判断 Soft AP 是否启动的状态： MXCHIP_UAP_UP 和 MXCHIP_UAP_DOWN
- ✓ 提供了以非阻塞方式实现 DNS 域名解析和连接 TCP server 功能
- ✓ 增加了用于返回当前 socket 的可用的发送缓冲区长度的函数：int tx_buf_size(int sockfd)
- ✓ 增加了用于实现 mxchipWNet - HA 协议的示例，该协议定义了串口，网络上的命令传输格式，运行该固件的模块通过串口与嵌入式设备进行交互，通过无线网络实现对设备的本地无线控制和云端远程控制。详情请参阅文档：RM0005-mxchipWNet_HA
- ✓ 在 Demo2 中增加了以非阻塞模式进行 DNS 解析和建立 TCP server 的示例
- ✓ 在 Wireless camera 示例中增加了用过网页设置无线网络的名称和密码的功能

版本 1.13

- ✓ 将工程文件从代码中移出
- ✓ 支持以组播方式发送和接收数据包
- ✓ 支持基于 TCP 的 SSL 加密数据传输
- ✓ 修正 gethostbyname 返回值的不正确的问题
- ✓ 支持在 WEP 加密方式下直接通过字符串输入 16 进制密码的功能
- ✓ 功能修改：在非阻塞模式下的 DNS 解析，一旦 DNS 超时，即通过回掉函数返回失败，而不是之前的不断重试
- ✓ 支持 EasyLink 2.0 配置方式，并增加了在 WPS_EasyLink demo 中的演示功能
- ✓ 增加了 CheckNetLink 返回的参数内容，修正 Checknetlink 函数无法在非 WPA 模式下获取 AP 信息的问题
- ✓ 修正 UDP Socket 无法关闭的问题
- ✓ 修正 TCP Socket 在主动关闭的时候，在某些情况下出现无法关闭的问题
- ✓ 增加了在静态分配内存的情况下，对最大连接数和 TCP 缓冲区长度的设置
- ✓ 增加了对不同地区支持的 Wi-Fi 频段的设置
- ✓ 增加了读取软件库版本号的 API，和向产测模式的终端输出应用程序信息的回调 API

- ✓ 增加了一个 demo，通过组播方式实现 mDNS 及 Bonjour 协议
- ✓ 增加了一个 demo，通过 HTTPS 协议实现对加密网页的读取

版本 1.13

- ✓ 优化了无线连接的断线重连机制，和 Station 和 Soft AP 共存模式的
- ✓ 修正了 DNS 解析时对于错误的 DNS 服务器地址：0.0.0.0 不能正确处理的问题

5 Sales Information

If you need to buy this product, please call MXCHIP during the working hours. (Monday ~ Friday A.M.9:00~12:00; P.M. 1:00~6:00)

Telephone: +86-21-52655026 / 52655025

Address: Room 811, Tongpu Building, No.1220 Tongpu Road, Shanghai

Post Code: 200333

Email: sales@mxchip.com

6 Technical Support

If you need to get the latest information on this product or our other product information, you can visit: http://www.mxchip.com/

If you need to get technical support, please call us during the working hours:

ST ARM technical support

+86 (021)52655026-822 Email: support@mxchip.com

Wireless network technical support

+86 (021)52655026-812 Email: support@mxchip.com

Development tools technical support

+86 (021)52655026-822 Email: support@mxchip.com