hIP

中文说明文档

作者：黄志坚

制作日期：2011 年 9 月 23 日

版本：v0.1

第一章 hIP 介绍

hIP 是一个高度可定制,可移植的 TCP/IP 协议栈.

高度可定制性:

可以在编译的时候选择要定制的协议,最低可以定制到只剩下 arp 协议和 icmp 协议.仅仅作为调试硬件网络使用,这时候占用的内存是最低的.最少可以在 100B 以下.

通过定制,被精简掉的部分是在预编译的时候对预编译选项进行判断的,所以,被精简掉的部分并不会被随着编译而进入协议栈.定制不仅可以节约内存,同样可以节约闪存.

极佳的移植性能:

hIP 是本着为了可以在不同平台上实现兼容性而生的 TCP/IP 协议栈,可以兼容 8 位,16 位,和 32 位的中低端嵌入式主控.可以在任何支持 ANSI C 的平台上编译通过,当然,前提你得把移植接口拼接完整后才行,整个 hIP 协议栈的接口函数都是独立于核心之外的,所以移植过程会变得简单无比.移植过程请详见<<hIP 移植>>篇章.

可调整的内存占用空间:

hIP 协议栈在整个运行过程中只占用一个字长八位的字符串作为发送和接收缓冲区,当前协议栈为了节省空间,把发送和接收的缓冲区共用,对于内存紧张的设备效果还是非常明显的,如果使用的协议栈是最精简的版本的,或者发送的数据的最大大小是可以预算出来的,那么可以在编译的时候设定缓冲区的大小来决定,一般来说,缓冲区设在大小为 200 实现一些最简单的功能还是绰绰有余的.不过当你要发送的数据量大的时候,最好的建议还是开辟一个更大的缓冲区,这样子数据不会溢出(当前版本没有加入对于数据溢出控制,下一个版本会添加.).

支持的协议分类:

1.ARP 地址解析协议

2.IP 网络互联协议

3.ICMP internet 控制报文协议

4.DNS 动态域名解析协议(下一个版本添加)

5.DHCP 动态主机设置协议

6.UDP 用户数据报文协议

7.TCP 传输控制协议

8.SMTP 邮件传送协议(下一个版本添加)

第二章 hIP 文件系统介绍

hIP_drive.c/h

该文件包含了网卡驱动,以及最基本的数据包发送和接收函数.

hIP_arp.c/h

该文件为包含了 ARP 协议方面的处理函数.

hIP_core_process.c/h

该文件为核心文件,一些与协议无关的处理函数都放置与其中.

hIP_dhcp.c/h

该文件为包含了 DHCP 协议方面的处理函数

hIP_eth.c/h

该函数为包含了 MAC 层方面的处理函数

hIP_eth_interface.c/h

该函数为发送函数和接收函数的接口层函数设置文件.

hIP_http.c/h

该文件是存放超文本传输协议层的一些函数.

hIP_icmp.c/h

该文件是存放 internet 控制报文协议层的一些函数.

hIP_ip.c/h

该文件时存放 ip 层的一些函数和声明.

hIP_tcp.c/h

该文件时存放 tcp 协议层的一些函数和声明

hIP_udp.c/h

该文件是存放 udp 协议层的一些函数和声明

hIP_clock.c/h

该文件时是存放 hIP 延迟和时钟定时配置的一些函数和声明.

hIP_conf.c/h

该文件存放 hIP 的主要配置,包括一些协议裁剪,和硬件相关层的配置,存放着所有的共有的全局变量(当然,一些

私有的全局变量我们全部都给予封转进专门的函数层,这样子可以减少各个模块的相关性.).

hIP_debuger.c/h(后续版本添加)

该文件是关于后台调试的,请确保移植的主控有支持标准的输入输出设备才可以启用该选项,通过重定向输入输出函数来配合 hIP 协议栈的调试,可以起到事半功倍的效果.

第三章　hIP 基本函数介绍

前注:本章的分割以各个文件作为基础来介绍,先后顺序以从底层到高层的顺序之分

hIP_arp.c / h

==========

函数名:　hIP_arp_request_process

调用到的函数:　hIP_eth_updata_dst_mac(len,buf);

```
hIP_eth_make_eth_process(len,buf,hip_ETH_PROC_ARP_V);
eth_PacketSend(42,buf);
```

输入:　数据包长度和数据包头指针

输出:　空

功能:　收到 arp 请求包的时候调用该函数.该函数为 arp 回应包内填充函数,并且包含包的发送长度,发送包的长度为 42Byte,最后会由网卡自动填充字节为 64

注意事项:　该函数只支持 802.3 帧结构的数据包

==========

函数名:　hIP_arp_reply_process

调用到的函数:　无

输入:　数据包长度和数据包头指针

输出: 空

功能: 该函数会收到 arp 回应包的时候调时调用,会自动更新目标 mac 地址,这个函数的前提是要主动发送 arp 请求包,即函数"void hIP_make_arp_request(U8* buf,U8 *arp_dst_ip)",如果没有发送 arp 请求包,被动调用该函数可能会导致目标 mac 地址错乱.

注意事项: 该函数被动调用的前提是先发送 arp 请求包

==========

函数名: hIP_make_arp_request

调用到的函数: eth_PacketSend(42,buf);

```
hIP_eth_change_dst_mac_process(temp_arp_dst_mac);
hIP_eth_make_eth_process(len,buf,hip_ETH_PROC_ARP_V);
```

输入: 数据包长度 arp 目标 ip 地址

输出: 空

功能: 该函数为主动调用函数,在主动连接对方机的时候,必须调用该函数来更新目标 mac 地址,不然就会导致发送出去的包目标 mac 地址为空(如果 DHCP 没有裁剪的话,那么这时候保留的 dhcp 服务器的 mac 地址,或者是全 ff 的广播 mac 地址,这时候不能确定一定可以发到对方上面,因为有些网卡是会丢弃超过一定长度的广播包).

注意事项: 该函数被动调用的前提是先发送 arp 请求包

hIP_icmp.c / h

==========

函数名: hIP_icmp_request_process(U16 len,U8 *buf)

调用到的函数: hIP_eth_updata_dst_mac(len,buf);

```
hIP_ip_update_dst_ip(len,buf);
hIP_ip_fill_ip_process(len,buf,20+40,0x00,hIP_IP_PROTO_ICMP_V);
hIP_eth_make_eth_process(len,buf,hIP_ETH_PROC_IP_V);
eth_PacketSend(hIP_IP_HEADER_LEN + hIP_ETH_LEN + 40,buf);
hIP_udp_send_data_process(len,buf);
```

输入: 数据包长度和数据包头指针

输出: 空

功能: 收到 ICMP 请求包的时候调用该函数,该函数会自动返回包含 icmp 数据的数据包,但是不会超过所定义的最大数据包的大小,所以如果使用的是超大帧的分割数据包,那么它只能应答最先发过来的数据包,其他数据包由于不是标准格式的 icmp 包,所以会被丢弃掉.请注意

注意事项: 无

==========

函数名: hIP_icmp_reply_process(U16 len,U8 *buf)

调用到的函数: 函数为空,请自行添加

输入: 数据包长度和数据包头指针

输出: 空

功能: 该函数会收到 icmp 回应包的时候调用的例常函数,用于处理 icmp 回应包应该进行的操作,与之配合的是 icmp 请求发送包函数.

注意事项: 无

==========

函数名：hIP_icmp_make_request_process(U16 len,U8 *buf)

调用到的函数： 函数为空,请自行添加

输入：数据包长度和数据包头指针

输出：空

功能：该函数为主动发送 icmp 请求包用于征求对方机是否存活或者测试延迟使用,由于该协议栈基本处于被动连接状态,所以该函数使用极少,如果要使用,请搭配"hIP_icmp_reply_process"使用,以至其可以达到所谓测量延迟或者目标存活状态等目的.该函数为空,如要使用,请参照 icmp 标准协议进行代码重组.

注意事项： 无

hIP_eth.c / h

==========

函数名：hIP_eth_make_eth_process(U16 len,U8 *buf,U16 protocal)

调用到的函数： 空

输入：数据包长度 数据包头指针 协议类型

输出：空

功能：该函数为填充 mac 头的标准函数,其功能有填充目标 mac 地址和源 mac 地址,以

及填充协议类型,目前我们物理层支持的协议类型有 IP 和 ARP 两种类型.该类型值在 hIP_core_type.h 中有具体定义.

注意事项:#define hip_ETH_PROC_ARP_V 0x0806 arp 协议类型代码

#define hIP_ETH_PROC_IP_V 0x0800 ip 协议类型代码

==========

函数名: hIP_eth_change_dst_mac_process(U8 *dst_mac)

调用到的函数: 空

输入: 目标 mac 地址

输出: 空

功能: 该函数会更改全部变量定义的目标 mac 地址,请在收到被动连接的各种包的情况下调用该函数,这样可以免去发送 arp 请求包的时间代价.输入参数请给予一个 u8 类型的数组的首地址,数组必须包含六个字节的 mac 地址.

注意事项: 无

==========

函数名: hIP_eth_change_src_mac_process(U8 *src_mac)

调用到的函数: 空

输入: 本机 mac 地址

输出: 空

功能: 该函数会改变本地 mac 地址,但是一般这条命令是不用的,除非遇到特别情况,因为更改本机 mac 地址可能会导致严重的连接问题.输入参数请给予一个 u8 类型的数组的首地址,数组必须包含六个字节的 mac 地址.

注意事项: 无

==========

函数名: hIP_eth_updata_dst_mac(U16 len,U8 *buf)

调用到的函数： 空

输入： 数据包长度 数据包头指针

输出： 空

功能： 该函数会更新目标地址发过来的包的目标 mac, 并使其数据包中的源 mac 地址填充到全局变量中的目的 mac 地址.

注意事项： 无

hIP_core_process.c / h

==========

函数名: hIP_eth_updata_dst_mac(U16 len,U8 *buf)

调用到的函数: hIP_dhcp_tick_proces();

hIP_ip_change_src_ip_process(temp_src_ip);

输入: 空

输出: 空

功能: 该函数会视你的当前配置文件而进行编译,如果是启用了默认的 dhcp 协议栈支持,那么会自动启动地址申请,另外,如果关闭 dhcp 协议栈支持,那么会设定默认地址为"192.168.1.150",当然,如果你要更改该默认地址,可以直接修改该函数即可,若或,可以搜索"192,168,1,150",来替换成你所需要的本机 ip 地址.

注意事项: 请在初始化网卡之后初始化该函数,并且在初始化定时器之前初始化该函数,可以保证在打开 dhcp 支持的情况下得到最佳获取 ip 速度.

==========

函数名: checksum(U8 *buf, U16 len,U8 type)

调用到的函数: 无

输入: 要计算校验和的首地址 要计算校验和的数据长度 协议类型

输出: 十六字长的最终校验和

功能: 该函数适合于计算 icmp,tcp,udp 和 ip 校验和,ip 和 icmp 校验和不需要添加伪首部,并且 ip 校验和需要加上其头长度,icmp 只需校验其头和数据长度即可,tcp 和 udp 则需要加上其 12 字节的伪首部来填充校验和,伪首部并不能作为 tcp 或 udp 封装的一部分,它只是简单的作为校验和的.

注意事项: #define hIP_IP_PROTO_ICMP_V 0x01 icmp 校验和协议值

#define hIP_IP_PROTO_TCP_V 0x06 tcp 校验和协议值

#define hIP_IP_PROTO_UDP_V 17 udp 校验和协议值

```
// type     0=ip
//          1=udp
//          2=tcp
```

==========

函数名: hIP_systick_process(void)

调用到的函数: 无

输入: 无

输出: 无

功能: 该函数会决定于是是否启用 arp 和 dhcp 而在编译的时候产生不同的代码,如果启用 arp,这里会默认留下 arp 协议老化机制函数的地方,如果该协议栈是用在主动发送

比较多的场合,那么还是建议这里应该填上 arp 老化检查函数,对于长时间未使用的目标地址,更新器 mac 地址.

注意事项: arp 默认老化时间为两分钟,如果两分钟内目标地址有数据通讯,最高可以达到十分钟.另外,每次进行主动通讯之前,都必须确定目标 mac 地址是否存在,如果不存在.那么就应主动发送 arp 请求包.

hIP_core_type.c / h

==========

函数名: eth_type_core(U16 len,U8 *buf)

调用到的函数: hIP_ip_update_dst_ip(len,buf);

eth_ip_type(len,buf);

eth_arp_type(len,buf);

输入: 数据包长度 数据包头指针

输出: 无

功能: 该函数会判断收到的包相应的是什么具体类型的包,并调用具体的处理函数.

注意事项: 该函数只有在正确的设置了本机的 ip 地址或者正确的获取了本机的 ip 地址后才能正常工作.不然收到的一切包都会被抛弃掉..

==========

函数名: eth_ip_type(U16 len,U8 *buf)

调用到的函数: hIP_icmp_type(len,buf);

hIP_tcp_type(len,buf);

hIP_udp_type(len,buf);

输入: 数据包长度 数据包头指针

输出: 无

功能: 该函数会判断收到的包相应的是什么具体类型的包,并调用具体的处理函数.

注意事项: 无

==========

函数名: eth_arp_type(U16 len,U8 *buf)

调用到的函数: hIP_arp_request_process(len,buf);

hIP_arp_reply_process(len,buf);

输入: 数据包长度 数据包头指针

输出: 无

功能: 该函数会判断收到的包相应的是什么具体类型的包,并调用具体的处理函数.

注意事项: 无

==========

函数名: hIP_icmp_type(U16 len,U8 *buf)

调用到的函数: hIP_icmp_reply_process(len,buf);

hIP_icmp_request_process(len,buf);

输入: 数据包长度 数据包头指针

输出: 无

功能: 该函数会判断收到的包相应的是什么具体类型的包,并调用具体的处理函数.

注意事项: 无

==========

函数名: hIP_udp_type(U16 len,U8 *buf)

调用到的函数: hIP_dhcp_core_process(len,buf);

hIP_dns_core_process(len,buf);

hIP_udp_rec_data_process(len,buf);

输入: 数据包长度 数据包头指针

输出: 无

功能: 该函数会判断收到的包相应的是什么具体类型的包,并调用具体的处理函数.

注意事项: 无

==========

函数名: hIP_udp_type(U16 len,U8 *buf)

调用到的函数: hIP_tcp_core_process(len,buf);

输入: 数据包长度 数据包头指针

输出: 无

功能: 该函数会判断收到的包相应的是什么具体类型的包,并调用具体的处理函数.

注意事项: 无

hIP_dns.c / h

==========

函数名: 空

调用到的函数: 无

输入: 无

输出: 无

功能: 无

注意事项: 无

hIP_eth_interface.c / h

==========

函数名: U16 eth_PacketReceive(U16 maxlen, U8* packet)

调用到的函数: 视驱动情况而定

输入: 最大可接受包长度 要存放的数据包缓存头指针

输出: 接收到的包的长度

功能: 这个函数是接收包的接口函数,也是整个接口层最重要的函数之一,在调用这个函数之前,请确保网卡返回的数据必须为八位长度,若网卡工作于八位状态,请在驱动中转换成八位再返回给该函数.

注意事项: 无

==========

函数名: void eth_PacketSend(U16 len, U8* packet)

调用到的函数: 视驱动情况而定

输入: 数据包长度 数据包头指针

输出: 无

功能: 这个函数是发送包的接口函数,也是整个接口层最重要的函数之一,在调用这个函数之前,请确保网卡能够正常发送八位数据包,若网卡工作在十六位模式,请自行在驱动中更改成十六位数据发送模式.

注意事项: 无

hIP_ip.c / h

==========

函数名: hIP_ip_change_dst_ip_process(U8 *dst_ip)

调用到的函数: 空

输入: 目标 ip 地址

输出: 无

功能: 由于更换目标 ip 地址函数使用较少,一般用于主动发送函数包使用,如果有用到该功能,请自行添加.

注意事项: 涉及到的全局变量表: extern U8 dstip[4];

U8 dstip[4];

==========

函数名: hIP_ip_change_src_ip_process(U8 *src_ip)

调用到的函数: 空

输入: 本机 ip 地址

输出: 无

功能: 本函数使用到的地方一般只有在 hIP 协议栈初始化的时候有用到,包括固定 ip 初始化和 dhcp 初始化,其他时候请勿随意使用该函数,不然容易导致系统 ip 丢失或者不可估量的错误.

注意事项: 涉及到的全局变量表: extern U8 srcip[4];

U8 srcip[4];

==========

函数名: hIP_ip_fill_ip_process(U16 len,

U8 *buf,

U16 total_len,

U8 fra_flag,

U8 protocal)

调用到的函数: 空

输入: 包长 包头指针 ip 层封装数据总长度 ip 分段标志 ip 上层封装的协议

输出: 无

功能: 该函数为 ip 包头封装函数,基于标准 ip 封包标准,其中该函数是不支持设置 ip 包生存时间(TTL),默认是 64s,另外,不支持超大基于 ip 的 ip 分片包.该函数校验和是整个 ip 头部校验和,不包括数据部分的校验和.另外,ip 头部长度也固定为 20 个字节,没有其他额外的选项来增加整个 ip 头长度.

注意事项: 涉及到的数据 #define hIP_IP_PROTO_ICMP_V 0x01

#define hIP_IP_PROTO_TCP_V 0x06

#define hIP_IP_PROTO_UDP_V 17

目前 ip 层支持的上层协议有 ICMP TCP UDP其他的可以自行添加.

==========

函数名: hIP_ip_update_dst_ip(U16 len,U8 *buf)

调用到的函数: 空

输入: 包长 包头指针

输出: 无

功能: 该数据包为更新目标 ip 地址函数,一般使用于当收到对方的数据包并且需要更新的时候.使用该函数请视通讯情况而定.

注意事项: 无

hIP_udp.c / h

==========

函数名: hIP_udp_rec_data_process(U16 len,U8 *buf)

调用到的函数: 空

输入: 包长 包头指针

输出: 无

功能: 该函数为 udp 数据包例常处理函数,只有当所定义的接口是私有接口(如不属于 67 68 等 dhcp 端口)时候,才会在接收到 udp 包之后调用此函数,该函数为空,使用情况也视通讯需求而定,因为每个通讯需求对于有需要用到 udp 数据包的地方都会进行不同的处理,所以请开发者自行填写.

注意事项: 无

==========

函数名: hIP_udp_make_header(U16 len,

U8 *buf,

U16 dst_port,

U16 src_port,

U16 data_len)

调用到的函数: 空

输入: 包长 包头指针 目标端口 源端口 udp 封装的数据长度

输出: 无

功能: 该函数为 udp 协议层最重要的 udp 头构造函数.请在封装好基于 udp 数据的情况下,再调用该函数,需要正确对待的是数据长度,因为如果数据长度出错,那么 udp 校验和也会相对出错,并且整个数据长度必须与 ip 层的数据长度相对应.两个其中一个错了都会导致包被丢弃.

注意事项: 无

==========

函数名: hIP_udp_fill_data_out(U16 len,U8 *buf,U16 data_len,U8 *data)

调用到的函数: 空

输入: 包长 包头指针 数据长度 数据缓冲指针

输出: 无

功能: 该函数为外部 udp 数据包外部填充函数,在调用这个函数之前,必须先建一个数据缓冲区,并且该数据缓冲区不能是与发送接收缓冲区有交集的,不然会发生数据冲突,这样子填充出来的数据并不是你想要的数据.该函数会把外部定义的数据临时缓冲区的数据搬运到发送缓冲区中.

注意事项: 无

==========

函数名: hIP_udp_fill_data_in(U16 len,U8 *buf)

调用到的函数: 空

输入: 包长 包头指针

输出: 数据长度

功能: 该函数为内部 udp 数据填充函数,所有的数据必须直接在这里面生成,并且该函数是直接对接收发送缓冲区进行操作的,所以如果是产生的数据是视接收发送缓冲区的数据而定的话,那么请提前做好数据备份,如用临时变量或者临时数组用来备份.

注意事项: 无

==========

函数名: hIP_udp_send_data_process(U16 len,U8 *buf)

调用到的函数: 空

输入: 包长 包头指针

输出: 无

功能: 该函数为发送 udp 数据包的 demo 函数,如果要发送数据包,可以参照该函数作为模板来发送数据包,如果是要测试 udp 数据包的话,那么可以在主循环定时发送 udp 数据包,但是前提请记得更新目标 ip 和目标 mac,另外,远程请注意开启相应的 udp 端口(默认是 9506)和十六进制接收模式.

注意事项: 无

hIP_tcp.c / h

==========

函数名: hIP_tcp_core_process(U16 len,U8 *buf)

调用到的函数: 空

输入: 包长 包头指针

输出: 无

功能: 该函数为 tcp 核心处理函数,如果网卡轮询接收函数判断收到的为 tcp 数据包的时候,那么会主动调用该函数,该 tcp 例常处理函数支持所有基于 tcp 的协议,像 ftp,smtp,telnet,http 等协议,要使用该函数同时也必须监听相应的端口,不然该函数判定完所有端口之后,如果所有的端口都没有开启的话,那么会跳出函数,这样对于一些实时性要求高的可以节省更多的时间来进行其他操作.该函数同时也是处理连接和断开函数的主要组成部分,几乎所有的操作都集中在该函数中,同样的,该函数也只支持被动连接,所以连接第二步包暂时是没有编译进去的,只有一三步有调试进去.另外该 tcp 不支持单向断开连接,如果接收断开连接包,或者主动断开连接,都是要求双方都确认为断开连接,并且同时重新初始化 tcp 连接的所有参数,回复到刚启动的状态.另外该函数还同时支持 tcp 重置包,当收到 tcp 重置包的时候,不会发送任何数据包进行回应,因为重置包是发生了不可恢复的错误而不得不发送的一个包,当对方发送重置包的时候是单方面断开连接,所以我们这里也要断开连接,但是不用发送包就对了.只要重新初始化即可.

注意事项: 如要详细了解该 tcp 核心函数使用,请详见该函数内置 demo.

==========

函数名: void hIP_tcp_fill_header(U16 len,

U8 *buf,

U16 data_len,

U16 src_port,

U16 dst_port,

U32 seq_num,

U32 ack_num,

U8 flags,

U16 wind,

U8 opt)

调用到的函数: checksum(&buf[hIP_IP_SRC_P],

(buf[hIP_TCP_OFFSET]>>4)*4 +8 + data_len,2);

输入: 包长 包头指针 数据长度 源端口 目的端口 确认号 序列号 标志 窗口大小 选项

输出: 无

功能: 该函数为核心 tcp 报头填充函数,该版本中,默认是支持无选项的 tcp 连接,当然,如果是特殊情况,请另外添加该函数的选项 case,并修改 tcp 报头长度(tcp 偏移量).该函数校验和为校验 tcp 头以及数据长度.在这里,校验和会自动去匹配 tcp 头长度,所以当你有选项的时候他也不会校验出错.其他填充需求请详见 tcp 协议规范.

注意事项: 如要详细了解该函数使用,请详见该函数内置 demo.

==========

函数名: hIP_tcp_get_lpc_connecting(U8 hIP_tcp_status)

hIP_tcp_get_lpc_conneted(U8 hIP_tcp_status)

hIP_tcp_get_lpc_disconnecting(U8 hIP_tcp_status)

hIP_tcp_get_upc_connecting(U8 hIP_tcp_status)

hIP_tcp_get_upc_conneted(U8 hIP_tcp_status)

hIP_tcp_get_upc_disconnecting(U8 hIP_tcp_status)

调用到的函数: 无

输入: tcp 指定上层协议连接状态字节

输出: Bool True

功能: 该些函数为返回指定 tcp 指定端口的连接状态,他是由一个字节的状态位构成,相应的,对于每个端口(每个应用),都要有指定的单独的状态字节,不然会导致连接出错,因为每次连接和断开连接都是决定于该独立的状态字节.另外,要查看该状态字节,唯一的方法是调用如上函数,可以保证状态字节完整.

注意事项: /*

** 位 0: 下位机主动申请连接中 1:正在连接 0:没有处于正在连接状态

** 位 1: 下位机到上位机连接状态 1:连接 0:断开

** 位 2: 下位机主动申请断开中 1:正在断开中 0:没有处于这个状态

** 位 3: 保留

** 位 4: 上位机主动申请连接中 1:正在连接 0:没有处于正在连接状态

** 位 5: 上位机到下位机连接状态 1:连接 0:断开

** 位 6: 上位机主动申请断开中 1:正在断开中 0:没有处于这个状态

** 位 7: 保留

*/

==========

函数名: hIP_tcp_set_lpc_connecting(U8 statu,U8 *hIP_tcp_status)

hIP_tcp_set_lpc_conneted(U8 statu,U8 *hIP_tcp_status)

hIP_tcp_set_lpc_disconnecting(U8 statu,U8 *hIP_tcp_status)

hIP_tcp_set_upc_connecting(U8 statu,U8 *hIP_tcp_status)

hIP_tcp_set_upc_conneted(U8 statu,U8 *hIP_tcp_status)

hIP_tcp_set_upc_disconnecting(U8 statu,U8 *hIP_tcp_status)

调用到的函数: 无

输入: tcp 指定上层协议连接状态字节 要设定的状态

输出: void

功能: 该些函数为设定指定 tcp 指定端口的连接状态,他是由一个字节的状态位构成,相应的,对于每个端口(每个应用),都要有指定的单独的状态字节,不然会导致连接出错,因为每次连接和断开连接都是决定于该独立的状态字节.另外,要修改该状态字节,唯一的方法是调用如上函数,可以保证状态字节完整.

注意事项: /*

** 位 0: 下位机主动申请连接中 1:正在连接 0:没有处于正在连接状态

** 位 1: 下位机到上位机连接状态 1:连接 0:断开

** 位 2: 下位机主动申请断开中 1:正在断开中 0:没有处于这个状态

** 位 3: 保留

** 位 4: 上位机主动申请连接中 1:正在连接 0:没有处于正在连接状态

** 位 5: 上位机到下位机连接状态 1:连接 0:断开

** 位 6: 上位机主动申请断开中 1:正在断开中 0:没有处于这个状态

** 位 7: 保留

*/

==========

函数名: hIP_tcp_get_buf_seq(U16 len,U8 *buf)

调用到的函数: 空

输入: 包长 包头指针

输出: 无

功能: 该函数为返回 tcp 数据包中的序列号,调用这个函数的前提是收到的且未修改过的包,如果修改过的 tcp 包,那么返回的就是要发送的序列号.

注意事项: 无

==========

函数名: hIP_tcp_get_buf_seq(U16 len,U8 *buf)

调用到的函数: 空

输入: 包长 包头指针

输出: 无

功能: 该函数为返回 tcp 数据包中的确认号,调用这个函数的前提是收到的且未修改过的包,如果修改过的 tcp 包,那么返回的就是要发送的确认号.

注意事项: 无

==========

函数名: hIP_tcp_get_buf_flags(U16 len,U8 *buf)

hIP_tcp_get_flags_syn(U16 len,U8 *buf)

hIP_tcp_get_flags_reset(U16 len,U8 *buf)

hIP_tcp_get_flags_ack(U16 len,U8 *buf)

hIP_tcp_get_flags_end(U16 len,U8 *buf)

调用到的函数: 空

输入: 包长 包头指针

输出: Bool True

功能: 该些函数为返回 tcp 接收的的包的连接标志位,第一个函数返回值需要自己处理,另外几个函数(查看是否是同步包,重置包,确认包,断开包),都只返回是或否.

注意事项: 要详细了解该些函数使用,请详见内置 demo.

==========

函数名: hIP_tcp_snd_update_struct(

struct hIP_TCP_send_windows *temp_snd_win,

struct hIP_TCP_receive_windows *temp_rec_win,

U16 len,

U8 *buf,

U16 data_len))

调用到的函数: 空

输入: 包长 包头指针

输出: Bool True

功能: 该函数为当发送 tcp 包的时候必须调用的函数,因为该函数会自动更新发送结构体和接收结构体.调用这函数的时机非常关键,一般是在构造好 tcp 上层数据之后,要去调用该函数,并且给予指定端口 tcp 的层发送和接收结构体,最后给予其长度,这样子就会正常更新了,至于其中的,可以参考 demo 模板应用.

注意事项: struct hIP_TCP_send_windows

```
{
    /* 总的窗口大小 */
    U16 TCP_snd_win_size;

    /* 剩余窗口大小 */
    U16 TCP_snd_win_remain;

    /* 已经发送未被确认的位置 */
    U32 TCP_snd_win_una;

    /* 下一个要发送的位置 */
    U32 TCP_snd_win_next;
};
```

```
struct hIP_TCP_receive_windows
{
    /* 总的窗口大小 */
    U16 TCP_rec_win_size;

    /* 已经被接收且确认的码 */
    U32 TCP_rec_win_next;

    /* 已经接收未发送确认包的码 */
```

```
    U32 TCP_rec_win_una;

    /* 还可以使用的字节 */
    U16 TCP_rec_win_remain;
};
```

==========

函数名: hIP_tcp_rec_update_struct(

struct hIP_TCP_receive_windows *temp_rec_win,

struct hIP_TCP_send_windows *temp_snd_win,

U16 len,

U8 *buf)

调用到的函数: 空

输入: 包长 包头指针

输出: Bool True

功能: 该函数为当接收 tcp 包的时候必须调用的函数,因为该函数会自动更新发送结构体和接收结构体.调用这函数的时机非常关键,一般是在接收好并确定是 tcp 上层数据之后,要去调用该函数,并且给予指定端口 tcp 的层发送和接收结构体,最后给予其长度,这样子就会正常更新了,至于其中的,可以参考 demo 模板应用.

注意事项: struct hIP_TCP_send_windows

```
{
    /* 总的窗口大小 */
    U16 TCP_snd_win_size;

    /* 剩余窗口大小 */
    U16 TCP_snd_win_remain;

    /* 已经发送未被确认的位置 */
    U32 TCP_snd_win_una;
```

```
    /* 下一个要发送的位置 */
    U32 TCP_snd_win_next;
};

struct hIP_TCP_receive_windows
{
    /* 总的窗口大小 */
    U16 TCP_rec_win_size;

    /* 已经被接收且确认的码 */
    U32 TCP_rec_win_next;

    /* 已经接收未发送确认包的码 */
    U32 TCP_rec_win_una;

    /* 还可以使用的字节 */
    U16 TCP_rec_win_remain;
};
```

==========

函数名: hIP_tcp_caculate_datalen(U16 len,U8 *buf)

调用到的函数: 空

输入: 包长 包头指针

输出: 数据长度

功能: 该函数一般是使用在计算收到包的情况下数据长度有多少,数据长度就是该包的返回值,当然,你也可以在已经封装好到 ip 层的数据包调用该函数,同样也可以得出数据包中 tcp 的数据量,但是一般不会这么做,因为 tcp 数据生成函数会自身返回数据长度,并不需要对其进行如此繁杂的操作.

注意事项: 无

==========

函数名: hIP_tcp_make_ack_for_any(U16 len,U8 *buf)

调用到的函数: 空

输入: 包长 包头指针

输出: 数据长度

功能: 该函数默认为发送空的 tcp 回应包,就是不带有任何数据的 tcp 包,所以,同理,我们每个创建 tcp 上层数据的函数,都是要求要返回长度的,这里我们把这个函数返回的数据长度定为 0.

注意事项: 要想详细了解该函数使用,请详见 demo 模板函数.

==========

函数名: hIP_tcp_make_ack_for_any(U16 len,U8 *buf)

调用到的函数: 空

输入: 包长 包头指针

输出: 数据长度

功能: 该函数默认为发送空的 tcp 回应包,就是不带有任何数据的 tcp 包,所以,同理,我们每个创建 tcp 上层数据的函数,都是要求要返回长度的,这里我们把这个函数返回的数据长度定为 0.

注意事项: 要想详细了解该函数使用,请详见 demo 模板函数.

hIP_clock.c / h

==========

函数名: hIP_clock_delay_ms(U16 clock)

调用到的函数: 空

输入: 要延迟的毫秒数

输出: 无

功能: 该函数为时钟接口函数,最高能实现 2^16 个 ms 延迟,调用该函数之前请确保该接口函数的函数驱动是完整的,因为对于每个不同的主控,都会有不同的延迟方案,请自行添加驱动.

注意事项: 无

==========

函数名: hIP_clock_delay_sec(U16 clock)

调用到的函数: 空

输入: 要延迟的秒数

输出: 无

功能: 该函数为时钟接口函数,最高能实现 2^16 个 s 延迟,调用该函数之前请确保该接口函数的函数驱动是完整的,因为对于每个不同的主控,都会有不同的延迟方案,请自行添加驱动.

注意事项: 无

hIP_dhcp.c / h

==========

函数名: hIP_dhcp_send_discover(U8* buf);

调用到的函数:

```
hIP_udp_make_header(len,buf,67,68,254);
hIP_ip_change_dst_ip_process(temp_dhcp_send_dst_ip);
hIP_ip_change_src_ip_process(temp_dhcp_send_src_ip);
hIP_ip_fill_ip_process(  len,
                         buf,
                         hIP_IP_HEADER_LEN +
                         hIP_UDP_HEADER_LEN + 254,
                         0x00,
                         hIP_IP_PROTO_UDP_V);
hIP_eth_change_dst_mac_process(temp_dhcp_dst_mac);
hIP_eth_make_eth_process(len,buf,hIP_ETH_PROC_IP_V);
eth_PacketSend(  hIP_UDP_HEADER_LEN +
                 hIP_IP_HEADER_LEN +
                 hIP_ETH_LEN +
                 254,
                 buf);
hIP_dhcp_statu_set_discover(1);
```

输入: 输入输出缓冲区首地址

输出: 无

功能: dhcp 动态主机配置协议是通常要搭配 dhcp 服务器才能使用,不过可以通过程序进行修改,当无法发现 dhcp 服务器的时候,就自动进行自我 ip 地址配置,而不是一直等待 dhcp 服务器进行回应.动态分配 ip 地址一共有分为四个步骤,该函数则是第一步骤,发送 dhcp 发现包,当该网络上有 dhcp 服务器的时候(有且只有一个),那么 dhcp 服务器

会对其进行回应.

注意事项: 无

==========

函数名: hIP_dhcp_send_request(U8* buf)

调用到的函数:

```
hIP_udp_make_header(len,buf,67,68,280);
hIP_ip_change_dst_ip_process(temp_dhcp_send_dst_ip);
hIP_ip_change_src_ip_process(temp_dhcp_send_src_ip);
hIP_ip_fill_ip_process(len,buf,
                       hIP_IP_HEADER_LEN +
                       hIP_UDP_HEADER_LEN + 280,
                       0x00,
                       hIP_IP_PROTO_UDP_V);
hIP_eth_change_dst_mac_process(temp_dhcp_dst_mac);
hIP_eth_make_eth_process(len,buf,hIP_ETH_PROC_IP_V);
eth_PacketSend(hIP_UDP_HEADER_LEN +
               hIP_IP_HEADER_LEN +
               hIP_ETH_LEN +
               280,
               buf);
hIP_dhcp_statu_set_request(1);
```

输入: 要延迟的秒数

输出: 无

功能: 该函数如上个函数所示,会在相应的时候被调用用于续约 ip 地址或者租约请求.

注意事项: 无

==========

函数名: hIP_dhcp_offer_process(U8* buf,U16 len)

调用到的函数: hIP_dhcp_statu_set_offset(1);

输入: 输入输出缓冲区首地址　　缓冲区长度

输出: 无

功能: 该函数为 dhcp 提供包处理函数,当发送 dhcp_discover 包之后,如果该网络上有存在 dhcp 服务器的时候,会自动回应以该包,该函数就是相应的处理该包来获取设置 ip 地址所需要的信息.

注意事项: 无

==========

函数名: hIP_dhcp_ack_process(U8* buf,U16 len)

调用到的函数: hIP_dhcp_statu_set_offset(1);

输入: 输入输出缓冲区首地址　　缓冲区长度

输出: 无

功能: 该函数为 dhcp 确认包处理函数,当发送 dhcp_request 包之后,如果该网络上有存在 dhcp 服务器的时候,会自动回应以该包,该函数就是相应的处理该包来获取设置 ip 地址所需要的信息.并且该包会同样启动租约倒计时计时器,和最后 ip 地址,网关地址,服务器地址,dns 服务器地址的确认.

注意事项: 无

==========

函数名: void hIP_dhcp_statu_set_init(U8 tmp_sta);

void hIP_dhcp_statu_set_discover(U8 tmp_sta);

void hIP_dhcp_statu_set_offset(U8 tmp_sta);

void hIP_dhcp_statu_set_request(U8 tmp_sta);

void hIP_dhcp_statu_set_ack(U8 tmp_sta);

void hIP_dhcp_statu_set_ok(U8 tmp_sta);

void hIP_dhcp_statu_set_error(U8 tmp_sta);

void hIP_dhcp_statu_set_timeout(U8 tmp_sta);

调用到的函数: 无

输入: dhcp 临时状态标志

输出: 无

功能: 该些函数为设置 dhcp 临时状态的函数,下面注意事项也列出了该些函数可以用来设置的状态,包含了 dhcp 所有可能处于的状态.

注意事项: /*

```
 * dhcp 状态位定义
 * 位 0:dhcp 是否初始化
 * 位 1:是否发送 dhcpdiscover 包
 * 位 2:是否收到 dhcpoffer 包
 * 位 3:是否发送 dhcprequest 包
 * 位 4:是否收到 dhcpack 包
 * 位 5:所有 ip 时候设置完毕
 * 位 6:dhcp 出错,需要终止
 * 位 7:dhcp 重新申请时间到
*/
```

==========

函数名: U8 hIP_dhcp_statu_get_init(void);

U8 hIP_dhcp_statu_get_discover(void);

U8 hIP_dhcp_statu_get_offset(void);

U8 hIP_dhcp_statu_get_request(void);

U8 hIP_dhcp_statu_get_ack(void);

U8 hIP_dhcp_statu_get_ok(void);

U8 hIP_dhcp_statu_get_error(void);

U8 hIP_dhcp_statu_get_timeout(void);

调用到的函数: 无

输入: dhcp 临时状态标志

输出: 无

功能: 该些函数为获取 dhcp 临时状态的函数,下面注意事项也列出了该些函数可以用来设置的状态,包含了 dhcp 所有可能处于的状态.

注意事项: /*

```
 * dhcp 状态位定义
 * 位 0:dhcp 是否初始化
 * 位 1:是否发送 dhcpdiscover 包
 * 位 2:是否收到 dhcpoffer 包
 * 位 3:是否发送 dhcprequest 包
 * 位 4:是否收到 dhcpack 包
 * 位 5:所有 ip 时候设置完毕
 * 位 6:dhcp 出错,需要终止
 * 位 7:dhcp 重新申请时间到
*/
```

==========

函数名: hIP_dhcp_init(void)

调用到的函数: hIP_dhcp_statu_set_init(1);

hIP_dhcp_send_discover(eth_buf);

hIP_dhcp_statu_set_discover(1);

输入: 无

输出: 无

功能: 该函数为 dhcp 动态获取地址初始化函数,包括了发送 discover 包和初始化所有有关于 dhcp 的计时全局函数和初始化 dhcp 临时状态.

注意事项: 无

==========

函数名: hIP_dhcp_core_process(U16 len,U8 *buf)

调用到的函数: hIP_dhcp_offer_process(buf,len);

hIP_dhcp_ack_process(buf,len);

输入: 输出输出缓冲区首地址 缓冲区长度

输出: 无

功能: 该函数为 dhcp 包处理函数,当收到 dhcp 包之后,会主动调用该函数,同理该函数会对于不同类型的 dhcp 数据包而调用不同的函数进行处理.

注意事项: 无

==========

函数名: hIP_dhcp_tick_proces(void)

调用到的函数: 函数很多,不一一列举

输入: 空

输出: 无

功能: 该函数为 dhcp 定时器例常处理函数,会根据 dhcp 的临时状态数据和租约倒计时来判断相应应该进行的操作.如果定时器没有溢出,会自动对定时器进行递减,知道触发

定时器溢出处理.

注意事项: 无

第四章　　hIP 移植相关

如上文所诉,hIP 是为了定制而生的协议,所以,移植性对于 hIP 来说是至关重要的一个环节,下面我们详细讲述下 hip 移植需要

注意的事项:

1.移植需要更改的文件:

hIP_conf.c / h

hIP_eth_interface.c / h

hIP_driver.c / h

2.hIP_conf.c

```
U8 eth_buf[ETH_MAXLEN];
```

该数组为网络输入输出缓冲区

```
U8 srcmac[6] = {0x32,0x12,0x35,0x11,0x01,0x51};
```

该数组为源 mac 地址定义.修改这个可以更改本机 mac

```
U8 dstmac[6] = {0x00};
```

该数组为目标 mac,无需更改.

```
U8 srcip[4] = {0x00};
```

该数组为源 ip,在初始化的时候更改.

```
U8 dstip[4] = {0x00};
```

该数组为目的 mac,在初始化的时候更改.

```
U8 submark[4] = {0x00};
U8 dhcpserverip[4] = {0x00};
U8 gtip[4] = {0x00};
U8 dnsserver[4] = {0x00};
U32 dhcpfreetime;
U32 dhcpfreecountdown;
```

U8 dhcp_statu;

该些函数在接收 dhcp 数据包的时候会自动更改.

3.hIP_conf.h

```
#define ETH_MAXLEN 1000
#define HTTP_MAXBUF_LEN 2000
#define hIP_conf_ARP
#define hIP_conf_IP
#define hIP_conf_UDP
#define hIP_conf_TCP
#define hIP_conf_ICMP
#define hIP_conf_DHCP
#define hIP_conf_DNS
#define hIP_conf_HTTP
#define hIP_conf_DEBUG
```

该些函数为预编译选项,请把不需要定制的协议给注释掉,这样子就可以完成最基本的定制了.

```
typedef   unsigned char        U8;
typedef   unsigned short int   U16;
typedef   unsigned int         U32;
```

请在红色的地方换上相应的类型声明,分别为 8 位无符号字符型,16 无符号整形,32 位无符号整形,这些类型声明完全取决于编译目标系统.所以请在移植的时候能够明白相应的处理器的类型相关事项.

4.hIP_eth_interface.c

```
U16 eth_PacketReceive(U16 maxlen, U8* packet)
{
    // TODO:请在该函数里填充上相应的网卡收包函数
}

void eth_PacketSend(U16 len, U8* packet)
{
```

```
        // TODO:请在该函数里填充上相应的网卡发包函数
    }
```

5.hIP_drive.c / h

请在该文件里面把所有的网卡驱动填写完整,包括最基本的收包和发包函数,和网卡初始化函数(非软件层,硬件初始化).

6.DEMO 实例

```
void InitNet(void)
{
    unsigned char ch1,a,b;
    unsigned int i;

    etherdev_init();
    hIP_init();

    /* 配置 systic 作为 1ms 中断,这个函数在 */
    SysTick_Config(SystemFrequency / 1000);

    hIP_http_init();

    for(i = 0; i< 99999; i++);

    a = 0xff;
    b = 0xff;
    while(1)
    {
        unsigned int plen = 0;

        //判断是否有接收到有效的包
        plen = eth_PacketReceive(ETH_MAXLEN, eth_buf);
        //如果收到有效的包，plen 将为非 0 值。
        if(plen==0)
        {
```

```
            ; //没有收到有效的包就退出重新检测
        }
        else
        {
            eth_type_core(plen,eth_buf);
            continue;
        }
        ch1 = hDBG_interface_rec_char();
        if(ch1 != 0xff)
        {
            if (a == 0xff)
            {
                a = ch1;
                continue ;
            }
            if (b == 0xff)
            {
                b = ch1;
                continue;
            }
            if ((a != 0xff) && (b != 0xff))
            {
                printf("%X  %X",a,b);
                a = 0xff;b= 0xff;
                continue;
            }
        }
    }
}
```

第五章 hIP 调试相关

hIP 调试工具介绍:

我们在 hIP 这个协议栈里面定制了一个强大的调试工具:hDBG,这个工具可以让你在仿真资源有限的情况下,可以利用串口调试或者其他方法,只要是可以进行交互的方式,都可以使用上去,甚至是直接移植到网页上面,当然,这其中可能会需要更多的交互转换,会麻烦一点.