

以太网芯片 CH563 TCP/IP 库说明

版本: 1E

<http://wch.cn>

1、概述

随着物联网的普及,越来越多的单片机系统需要用到网络通讯。

CH563 芯片自带以太网 MAC 和 PHY,支持 10/100M 全双工,半双工,自动协商,线路自动转换等功能,可以直接和网络终端如 PC,嵌入式设备进行数据交互。

CH563NET.LIB 提供了 TCP/IP 子程序库,集成了 TCP,UDP,ICMP,ARP,RARP,ICMP,IGMP 等以太网协议栈。可以同时支持 TCP,UDP 和 IPRAW 三种模式,最大支持 31 个 socket 同时通讯。

本文档适用 CH561,CH563 芯片。

2、全局变量

2.1. CH563NETConfig

CH563NET 库配置变量,其含义如下:

位 0-4 socket 的个数,最大值为 31,配置宏定义为 CH563NET_MAX_SOCKET_NUM;

位 5-8 MAC 接收描述符的个数,最大值为 15,配置宏定义为 RX_QUEUE_ENTRIES;

位 9-12 保留;

位 13 PING 使能,1 为开启 PING,0 为关闭 PING,默认为开启,配置宏定义为 CH563NET_PING_ENABLE ;

位 14-18 TCP 重试次数,配置宏定义为 TCP_RETRY_COUNT;

位 19-23 TCP 重试周期,单位为 50 毫秒,配置宏定义为 TCP_RETRY_PERIOD;

位 24 以太网中断类型配置,0 为 IRQ 中断,1 为 FIQ 中断;

位 25 发送重试配置,1 开启发送重试,0 关闭发送重试;

CH563NET_Config 是由上述配置宏定义在 CH563NET.H 中计算得出的,所以应用程序在包含 CH563NET.H 前应该根据实际情况进行配置,否则会采用默认的配置,关于默认配置请参考 CH563NET.H。

关于 CH563NET 的其他配置请参考 4.2 章节

2.2. CH563MACRxDesBuf

CH563MACRxDesBuf 为一维数组,MAC 接收描述符缓冲区,大小为 RX_QUEUE_ENTRIES * 16,此数组必须要 16 字节对齐,供 CH563MAC 使用。

2.3. CH563MACRxBuf

CH563MACRxBuf 为二维数组,MAC 接收缓冲区,定义如下:

UINT8 CH563MACRxBuf[RX_QUEUE_ENTRIES][MAX_PKT_SIZE]

地址 4 字节对齐,RX_QUEUE_ENTRIES 为接收描述个数,MAX_PKT_SIZE 为可以接收数据包的最大长度,在 CH563NET.H 中定义。

2.4. SocketInf

SocketInf 为一维数组,socket 信息列表,定义如下:

SOCK_INF SocketInf[CH563NET_MAX_SOCKET_NUM]

地址 4 字节对齐, CH563NET_MAX_SOCKET_NUM 为 socket 个数, SocketInf 保存了各个 socket 的信息, 其信息成员请参考 SOCK_INF 的定义。此变量由库内部进行读写操作, 如果没必要请勿在应用程序(是指调用库函数的用户程序, 本文称应用程序, 下同)中对其进行写操作。

2.5. CH563Inf

CH563Inf 此变量保存了库内部的全局信息变量, 其各成员请参考 CH563NET.H。应用程序可以通过此变量查询一些信息, 例如产生不可达中断后, 可以通过此变量查询不可达的一些信息。应用程序不可以对此变量进行写操作。

2.6. Memp_Memory

CH563NET 内部使用的池分配内存, 主要用于数据接收。其大小计算公式参考 CH563NET.H 关于 CH563NET_MEMP_SIZE 的宏定义。

2.7. Mem_Heap_Memory

CH563NET 内部使用的堆分配内存, 主要用于数据发送发数据。其大小计算公式参考 CH563NET.H 关于 CH563NET_RAM_HEAP_SIZE 的宏定义。

2.8. Mem_ArpTable

ARP 缓存表, 用于记录 IP 和 MAC 对。ARP 缓存表的大小可以配置。

2.9. MemNum、MemSize

MemNum和MemSize 是根据用户配置生成的数组, CH563NET用着两个数组来管理内存分配, 不可以修改。

3、子程序

3.1. 库子程序总表

分类	函数名	简要说明
基本函数	CH563NET_Init	库初始化
	CH563NET_GetVer	获取库版本号
	CH563NET_MainTask	库主任务函数，需要不断调用
	CH563NET_TimeIsr	定时器中断服务子函数
	CH563NET_ETHIsr	以太网中断服务子函数
	CH563NET_GetPHYStatus	获取 PHY 状态
	CH563NET_QueryGlobalInt	查询全局中断
	CH563NET_GetGlobalInt	查询全局中断并清零
	CH563NET_Aton	将 ASCII 码地址转换为网络地址
	CH563NET_Ntoa	将网络地址转换为 ASCII 码
	CH563NET_CloseMac	关闭 MAC
	CH563NET_OpenMac	打开 MAC
socket 函数	CH563NET_GetSocketInt	查询 socket 中断并清零
	CH563NET_SocketCreat	创建 socket
	CH563NET_SocketClose	关闭 socket
	CH563NET_SocketRecvLen	获取 socket 接收数据长度
	CH563NET_SocketRecv	socket 接收数据
	CH563NET_SocketSend	socket 发送数据
	CH563NET_SocketListen	TCP 监听
	CH563NET_SocketConnect	TCP 连接
	CH563NET_Mdi fyRecvBuf	修改 socket 接收缓冲区
	CH563NET_SocketUdpSendTo	向指定的 IP，端口发送 UDP 报文
	CH563NET_QueryUnack	查询是否有未 ACK 的报文
	CH563NET_RetrySendUnack	重发未 ACK 的报文
DHCP 函数	CH563NET_DHCPStart	启动 DHCP
	CH563NET_DHCPStop	关闭 DHCP
DNS 函数	CH563NET_InitDNS	初始化 DNS
	CH563NET_GetHostName	获取主机 IP 地址
KEEPLIVE 函数	CH563NET_ConfigKeepLive	配置 KEEP LIVE 参数
	CH563NET_SocketSetKeepLive	配置 socket 参数

关于中断：

库的全局中断和 socket 中断，其实仅仅是变量的一个标志，并非 CH563 产生的硬件中断。

3.2. CH563NET_Init

函数原型	UINT8 CH563NET_Init(const UINT8* ip, const UINT8* gwip, const UINT8* mask, const UINT8* macaddr)
输入	ip: IP 地址指针 gwip: 网关地址指针 mask: 子网掩码指针 macaddr: MAC 地址指针
输出	无
返回	返回 0 表示成功，其他值错误
作用	① 将所有全局变量进行清零，并根据配置和传递的变量进行全局变量初始化； ② 初始化 MAC； ③ 初始化 TCP/IP 协议栈。

子网掩码指针，可以设置为 NULL，如果为 NULL 则库会使用 255.255.255.0 作为子网掩码。

CH563NET_Init 对 MAC 初始化为自动协商模式、接收广播和 MAC 地址匹配的数据包，如果需要进行特殊操作，例如 PHY 需要工作在 10M 半双工、进行 HASH 过滤等，可以在 CH563NET_Init 之后添加外部代码实现。

3.3. CH563NET_GetVer

函数原型	UINT8 CH563NET_GetVer(void)
输入	无
输出	无
返回	库的版本号
作用	获取库的版本号。

3.4. CH563NET_MainTask

函数原型	void CH563NET_MainTask(void)
输入	无
输出	无
返回	无
作用	库主任务函数，需要不断调用。

3.5. CH563NET_TimeIsr

函数原型	void CH563NET_TimeIsr(UINT16 timperiod)
输入	timperiod: 定时周期
输出	无

返回	无
作用	定时器中断服务函数。

此函数主要为 CH563NET 库提供时钟，需要周期性的定时调用，其调用周期为 `timperiod`，周期应该不大于重试次数，否则会导致重试间隔不准确。

3.6. CH563NET_ETHISR

函数原型	<code>void CH563NET_ETHISR(void)</code>
输入	无
输出	无
返回	无
作用	以太网中断服务函数，产生以太网中断后调用。

3.7. CH563NET_GetPHYStatus

函数原型	<code>UINT8 CH563NET_GetPHYStatus(void)</code>
输入	无
输出	无
返回	PHY 的状态
作用	获取 PHY 的当前状态，主要是以下状态： <code>PHY_DISCONN</code> 、 <code>PHY_10M_FLL</code> 、 <code>PHY_10M_HALF</code> 、 <code>PHY_100M_FLL</code> 、 <code>PHY_100M_HALF</code> 。

3.8. CH563NET_QueryGlobalInt

函数原型	<code>UINT8 CH563NET_QueryGlobalInt(void)</code>
输入	无
输出	无
返回	全局中断状态
作用	获取全局中断状态。

3.9. CH563NET_GetGlobalInt

函数原型	<code>UINT8 CH563NET_GetGlobalInt(void)</code>
输入	无
输出	无
返回	全局中断状态
作用	获取全局中断状态，并将全局状态清零。

3.10. CH563NET_Aton

函数原型	UINT8 CH563NET_Aton(const UINT8 *cp, UINT8 *addr)
输入	cp: ASCII 码形式的地址, 例如 “192.168.1.2”
输出	addr: 网络地址, 例如: 0xC0A80102
返回	0, 表示成功否则失败
作用	将 ASCII 码地址转换为网络地址。

3.11 CH563NET_Ntoa

函数原型	UINT8 *CH563NET_Ntoa(UINT8 *addr)
输入	addr: 网络地址, 例如: 0xC0A80102
输出	无
返回	ASCII 码地址指针
作用	将网络地址转换为 ASCII 码地址。

3.12 CH563NET_CloseMac

函数原型	void CH563NET_CloseMac(void)
输入	无
输出	无
返回	无
作用	关闭 MAC 的时钟和电源。

3.13 CH563NET_OpenMac

函数原型	void CH563NET_OpenMac(void)
输入	无
输出	无
返回	无
作用	打开 MAC。

CH563NET 提供 CH563NET_CloseMac 和 CH563NET_OpenMac 函数主要是基于功耗的考虑。例如在以太网未连接的情况下, 为了节省功耗, 可以周期性的打开和关闭 MAC, 直到检测到以太网连接。

应用层可以在不要网络连接时调用 CH563NET_CloseMac 来关闭 MAC 的时钟和电源。需要时再调用 CH563NET_OpenMac 来开启 MAC 的时钟和电源。注意, 在调用 CH563NET_OpenMac 后一定要重新打开以太网中断

3.14. CH563NET_GetSocketInt

函数原型	UINT8 CH563NET_GetSocketInt (void)
输入	无

输出	无
返回	返回 socket 中断
作用	获取 socket 中断，并将 socket 中断清零。

3.15. CH563NET_SocketCreat

函数原型	UINT8 CH563NET_SocketCreat (UINT8 *socketid, SOCK_INF *socinf)
输入	socinf: socket 信息列表
输出	socketid: socket 索引
返回	执行状态
作用	创建 socket。

socketinf 仅作为变量传递，CH563NET_SocketCreat 对列表信息进行分析，如果信息合法，则会从 SocketInf[CH563NET_MAX_SOCKET_NUM]中找到一个空闲的列表 n，将 socketinf 复制到 SocketInf[n]中，将 SocketInf[n]锁定并创建相应的 UDP、TCP 或者 IPRAW 连接。如果创建成功，将 n 写入到 socketid 中并返回成功。

在创建 UDP、TCP 客户端，IPRAW 时，应该在创建之前分配好接收缓冲区和接收缓冲区大小。TCP 服务器分配的方式则不同，应该在接收到连接成功中断后调用函数 CH563NET_ModifyRecvBuf 来分配接收缓冲区。

具体使用方法请参考相关例程。

3.16. CH563NET_SocketClose

函数原型	UINT8 CH563NET_SocketClose (UINT8 socketid, UINT8 flag)
输入	socketid: socket 索引 flag: 关闭标志
输出	无
返回	执行状态
作用	关闭 socket。

在 UDP 和 IPRAW 模式下，flag 无效，调用此函数可以立即关闭 socket。

在 TCP 模式下，flag 可以为：

TCP_CLOSE_NORMAL 表示正常关闭，4 次握手后关闭，关闭速度较慢；

TCP_CLOSE_RST 表示复位连接，CH563NET 会向目的端发送 RST 进行复位，关闭速度较快；

TCP_CLOSE_ABANDON 表示直接丢弃，不会向目的端发送任何信息，关闭 socket，关闭速度最快。

调用此函数，一般可能需要等待一定的时间才可以关闭，这个时间主要是因为库需要一定的时间去中止 TCP 连接，只要产生 SINT_STAT_TIM_OUT 或 SINT_STAT_DISCONNECT 中断，则此 socket 一定是关闭状态。

3.17. CH563NET_SocketRecvLen

函数原型	UINT32 CH563NET_SocketRecvLen(UINT8 socketid, UINT32 *bufaddr)
输入	socketid: socket 索引
输出	bufaddr: socket 接收缓冲区地址
返回	接收数据长度
作用	获取 socket 接收数据长度。

此函数主要用于获取 socket 接收数据长度和接收缓冲区地址,应用程序可以直接使用此函数输出的地址,不需要复制即可使用内部接收缓冲区的数据,可以在一定程度上节约 RAM。如果 bufaddr 置为 NULL,则此函数仅返回 socket 接收数据的长度。

3.18. CH563NET_SocketRecv

函数原型	UINT8 CH563NET_SocketRecv(UINT8 socketid, UINT8 *buf, UINT32 *len)
输入	socketid: socket 索引 buf: 应用层缓冲区地址 len: 接收长度
输出	无
返回	执行状态
作用	读取 socket 缓冲区的数据。

此函数将 socket 接收缓冲区的数据复制到 buf 中,实际复制的数据长度会写入到 len 中。

CH563NET 提供了两种接收数据的方式,第一种为中断方式,另一种为回调模式。

中断方式是指 CH563NET 在接收到数据后,产生中断,用户可以通过函数 CH563NET_SocketRecvLen 和 CH563NET_SocketRecv 来读取接收到的数据。IPRAW、UDP 和 TCP 均可以采用这种方式接收数据。如果参数 buf 不为 NULL,CH563NET_SocketRecv 将内部缓存区的数据复制到 buf 中。如果 buf 为 NULL,则*len 不能小于实际的长度,表示应用层已经将所有的数据处理完毕。

回调模式仅在 UDP 模式下有效,CH563NET 在接收到数据后通过回调 SocketInf 结构中的 AppCallBack 函数来通知应用层接收数据。AppCallBack 由应用层来实现,应用层必须在此函数中将所有数据读完,否则 CH563NET 会强行清除。如果不需要回调模式,务必在创建 socket 时将 AppCallBack 清除为 0。回调函数的原型如下:

函数原型	void (*AppCallBack)(struct _SCOK_INF *socinf, UINT32 ipaddr, UINT16 port, UINT8 *buf, UINT32 len)
输入	socinf: CH563NET 将 socket 信息列表通过此形参传递给应用层,应用层通过此参数可以知道 socket 信息。 ipaddr: 数据报文的源 IP 地址 port: 数据报文的源端口 buf: 缓冲区地址 len: 数据长度
输出	无

返回	无
作用	UDP 模式下接收回调函数。

3.19. CH563NET_SocketSend

函数原型	UINT8 CH563NET_SocketSend(UINT8 socketid, UINT8 *buf, UINT32 *len)
输入	socketid: socket 索引 buf: 应用层缓冲区地址 len: 发送长度
输出	无
返回	执行状态
作用	socket 发送数据。

该函数将 buf 中的数据复制到内部协议栈发送缓冲区中，将数据发送，并将实际发送的长度通过 len 输出，应用层在实际处理的时候需要检查 len，以便确定实际发送的数据长度。如果发送的数据过多，此函数会自动重试多次进行发送，此函数返回 0(成功)并不表示将所有的数据发送完毕。

3.20 CH563NET_SocketUdpSendTo

函数原型	UINT8 CH563NET_SocketUdpSendTo(UINT8 socketid, UINT8 *buf, UINT32 *len, UINT8 *ip, UINT16 port)
输入	socketid: socket 索引 buf: 缓冲区起始地址 len: 发送数据长度 ip: 目的 IP 地址 port: 目的端口
输出	len 实际发送的长度
返回	执行状态
作用	向指定的 IP 和端口发送 UDP 数据报文。

在 UDP 模式下 CH563NET_SocketSend 和 CH563NET_SocketUdpSendTo 的区别在于，前者只能向创建 socket 时指定的目标 IP 和端口发送数据，后者可以向任意的 IP 和端口发送数据。CH563NET_SocketUdpSendTo 一般用在 UDP 服务器模式。

3.21. CH563NET_SocketListen

函数原型	UINT8 CH563NET_SocketListen(UINT8 socketid)
输入	socketid: socket 索引
输出	无
返回	执行状态
作用	TCP 监听，在 TCP SERVER 模式下使用。

如果应用层需要建立一个 TCP SERVER, 首先使用 CH563NET_SocketCreat 创建一个 TCP, 然后调用该函数使 TCP 进入监听模式。在监听模式的 TCP 是不进行数据收发的, 仅仅是监听 TCP 连接, 一旦有客户端向此服务器连接, 库会自动分配一个 socket 并产生连接中断 SINT_STAT_CONNECT。所以监听的 TCP 并不需要分配接收缓冲区。

3. 22. CH563NET_SocketConnect

函数原型	UINT8 CH563NET_SocketConnect (UINT8 socketid)
输入	socketid: socket 索引
输出	无
返回	执行状态
作用	TCP 连接, 在 TCP Client 模式下使用。

如果应用层需要建立一个 TCP Client, 首先使用 CH563NET_SocketCreat 创建一个 TCP, 然后调用该函数进入连接。连接成功后会产生连接中断 SINT_STAT_CONNECT。如果远端不在线或端口未打开, 库会自动重试一定次数, 仍然不成功会产生超时中断 SINT_STAT_TIM_OUT。

3. 23. CH563NET_Mdi fyRecvBuf

函数原型	void CH563NET_Mdi fyRecvBuf (UINT8 sockeid, UINT32 bufaddr, UINT32 bufsize)
输入	socketid: socket 索引 bufaddr: 缓冲区起始地址 bufsize: 缓冲区大小
输出	无
返回	无
作用	修改 socket 接收缓冲区。

为了使应用层方便灵活的处理数据, 库允许动态修改修改 socket 接收缓冲区的地址和大小, 在修改接收缓冲区前最好调用 CH563NET_SocketRecvLen 来检查缓冲区中是否有剩余数据, 一旦调用 CH563NET_Mdi fyRecvBuf, 原缓冲区的数据将会被清除。在 TCP 模式下, 如果连接已经建立, 调用 CH563NET_Mdi fyRecvBuf, 库会向远端通告当前窗口大小。

3. 24. CH563NET_SetSocketTTL

函数原型	UINT8 CH563NET_SetSocketTTL(UINT8 socketid, UINT8 ttl)
输入	socketid: socket 索引 ttl: TTL 值
输出	无
返回	执行状态
作用	修改 socket 的 TTL。

注意: TTL 不可以为 0, 默认为 128。

3.25. CH563NET_QueryUnack

函数原型	UINT8 CH563NET_QueryUnack(SOCK_INF *sockinf, UINT32 *addrlist, UINT16 lisen)
输入	socketid: socket 索引 addrlist: 未发送成功数据缓存地址列表 lisen: addrlist 长度
输出	无
返回	未发送成功报文的个数
作用	查询 TCP 未发送成功的报文信息。

Unack Segment 指未发送成功的 TCP 报文。

CH563NET_QueryUnack 用于查询 socket 未发送成功报文的个数以及报文的地址。有两种使用法:

1: 查询 Unack Segment 个数, 参数 addrlist 为 NULL, CH563NET_QueryUnack 会返回此 socket Unack Segment 的个数。

2: 查询 Unack Segment 的信息, CH563NET_QueryUnack 会向 addrlist 写入这些数据报文的地址。

应用层也可以通过循环 CH563NET_QueryUnack(sockinf, NULL, 0) 来查询是否有 Unack Segment。如果有再次调用 CH563NET_QueryUnack 来获取信息:

```
While(1)
{
    If(CH563NET_QueryUnack(sockinf, NULL, 0))
    {
        CH563NET_QueryUnack(sockinf, addrlist, sizeof(addrlist));
    }
    /*其他任务*/
}
```

3.26. CH563NET_RetrySendUnack

函数原型	void CH563NET_RetrySendUnack (UINT8 socketid)
输入	socketid: socket 索引
输出	无
返回	无
作用	重发未发送成功的报文。

CH563NET_RetrySendUnack 仅在 TCP 模式下有效, 用于重发未发送成功的报文。应用程序可以通过 CH563NET_QueryUnack 检查 socket 的数据是否已经全部成功发送, 如果需要可以调用 CH563NET_RetrySendUnack 立即将数据报文重新发送。一般情况下不需要应用层来重新发送, CH563NET 会自动重试。

3.27. CH563NET_DHCPStart

函数原型	UINT8 CH563NET_DHCPStart(u8(* usercall)(UINT8, void*))
输入	usercall: 应用层回调函数
输出	无
返回	状态。0 表示成功, 其他值为失败
作用	启动 DHCP。

如果使用 DHCP 功能, 需要在工程中添加 CH563NET_DHCP.0 文件。当 DHCP 成功或者失败时, 库会调用 usercall 函数, 通知应用层 DHCP 的状态, CH563NET 向本函数传递两个参数, 第一个参数为 DHCP 状态, 0 为成功, 其他值失败, 当 DHCP 成功时, 用户可以通过第二个参数获取到一个指针, 该指针指向的地址依次保存了 IP 地址, 网关地址, 子网掩码, 主 DNS 和次 DNS, 一共 20 个字节。注意该指针为临时变量 usercall 返回后, 该指针失效。

如果当前网络内没有 DHCP Server, 会产生超时时间, 超时时间约为 10 秒。超时后调用 usercall 函数通知应用层, 此时 DHCP 并不会停止, 会一直查找 DHCP Server。用户可以调用 CH563NET_DHCPStop 来停止 DHCP。

使用时注意以下两点:

- 1: 必须在 CH563NET_Init 成功之后启动 DHCP (必须)。
- 2: 在 DHCP 成功之后, 再创建 socket (推荐)。

如果 DHCP 失败, 则可以用 CH563NET_Init 时使用的 IP 地址进行通讯。

3.28. CH563NET_DHCPStop

函数原型	UINT8 CH563NET_DHCPStop(void)
输入	无
输出	无
返回	返回成功
作用	停止 DHCP。

3.29. CH563NET_InitDNS

函数原型	void CH563NET_InitDNS(UINT8 *dnsip, UINT16 port)
输入	dnsip: DNS 服务器的 IP 地址 port: DNS 服务器端口
输出	无
返回	无
作用	初始化 DNS

如果使用 DNS 功能, 在 CH563NET_Init 之前调用此函数, 来启动 DNS 功能。

注意: 如果使用 DNS 功能, 需要在工程文件中添加 ch563net_dns.o。

3.30. CH563NET_GetHostName

函数原型	UINT8 CH563NET_GetHostName(const char *hostname, UINT8
------	--

	*addr, dns_callback found, void *arg)
输入	hostname: 主机的域名 found: 回调函数 arg: found 的参数
输出	addr: 输出主机的 IP 地址, 仅在函数返回 0 时有效, 必须四字节对齐。
返回	执行结果
作用	获取主机 IP 地址

found 为应用层的函数指针。其基本原型如下:

```
typedef void (*dns_callback)(const char *name, u8 *ipaddr, void *callback_arg);
```

此函数用于获取主机的 IP 地址。hostname 为主机域名, addr 为 IP 地址指针, 如果 hostname 对应的 IP 地址已经在 DNS 的缓存中, CH563NET_GetHostName 会直接返回成功, 并且把主机所对应的 IP 地址输出到 addr 中。如果不在缓冲存中, 则 DNS 模块会发起 DNS 事务, 向 DNS 服务器询问, 失败或者成功后会调用 found 函数, 失败参数 ipaddr 为 NULL。

arg 为 found 所需要的参数, 可以为 NULL。但是 found 不可以为 NULL。

CH563NET_GetHostName 函数有以下几点需要注意:

- 1: addr 必须 4 字节对齐, 否则会返回对齐错误。
- 2: hostname 字符串最大长度不得大于 256。

参考例程:

```
i = CH563NET_GetHostName("www.wch.cn", p, CH563NET_DNSCallback, NULL);
if(i == 0)
{
    printf("CH563NET_GetHostName Success\n");
    printf("Host Name = %s\n", name);
    printf("IP = %d.%d.%d.%d\n", p[0], p[1], p[2], p[3]);
}
```

```
void CH563NET_DNSCallback(const char *name, UINT8 *ipaddr, void *callback_arg)
{
    if(ipaddr == NULL)
    {
        printf("DNS Fail\n");
        return;
    }
    printf("Host Name = %s\n", name);
    printf("IP = %d.%d.%d.%d\n", ipaddr[0], ipaddr[1], ipaddr[2], ipaddr[3]);
    if(callback_arg != NULL)
    {
        printf("callback_arg = %02x\n", (*(u8 *)callback_arg));
    }
}
```

3.31. CH563NET_ConfigKeepLive

函数原型	void CH563NET_ConfigKeepLive(struct _KEEP_CFG *cfg)
输入	cfg: KEEPLIVE 配置参数
输出	无
返回	无
作用	配置库的 KEEPLIVE 全局参数

此函数用于配置 CH563NET 中 KEEPLIVE 的参数。struct _KEEP_CFG 结构在 CH563NET.H 中定义。

```
struct _KEEP_CFG
{
    UINT32 KLIIdle;
    UINT32 KLIIntvl;
    UINT32 KLCCount;
};
```

KLIIdle: 空闲时间, 是指 TCP 连接空闲一定的时间后启动 KEEPLIVE 探测, 单位为 MS, 默认值为 20000。

KLIIntvl: 间隔, 是指 TCPKVIIE 探测超时间隔时间, 单位为 MS, 默认值为 15000。

KLCCount: 次数, 是指 KEEPLIVE 探测的次数, 默认值为 9。

此设置是对库做全局设置, 对启动 KEEPLIVE 功能的 TCP 连接有效。当 KEEPLIVE 发送 KLCCount 次探测对方均无任何响应, 则认为此连接无效。会断开连接。此函数应该在初始化库后调用。

3.32. CH563NET_SocketSetKeepLive

函数原型	UINT8 CH563NET_SocketSetKeepLive(UINT8 socketid, UINT8 cfg);
输入	socketid: socket 索引 cfg: 配置值
输出	无
返回	0 为成功, 其他为失败
作用	启动/关闭 socket

此函数用于开启或者关闭 socket 的 KEEPLIVE 功能, cfg 为 1 表示开启 KEEPLIVE, 为 0 表示关闭。在创建 scoekt 后, KEEPLIVE 默认为关闭。TCP 客户端, 在打开 socket 后, 调用此函数开启 KEEPLIVE 功能; TCP 服务器, 在产生 SINT_STAT_CONNECT 后调用此函数开启 KEEPLIVE 功能。

4、使用指南

4.1. 初始化

CH563NET_Init 为库初始化函数。关于 CH563NET_Init 函数使用方法请参考 3.2。对 CH563NET 初始化后, 应用层需要开启以太网中断和全局中断, 并在相应的中断函数中调用

中断服务函数 CH563NET_ETHIsr, 另外库函数需要外部提供时钟, 用于和时间相关的任务, 例如刷新 ARP 列表, TCP 超时等, 所以还需要应用程序初始化一个定时器, 在定时器溢出中断中调用定时器中断服务函数 CH563NET_TimeIsr, 定时周期应用层自行定义, 但是不得超过 TCP 重试周期。

综上, 在调用 CH563NET_Init 后应用层需要开启以太网中断, 初始化定时器, 开启定时器中断, 开启总中断。

4.2. 关于配置

CH563NETConfig 是库中定义的一个 32 位的全局变量, 详细信息请参考 2.1., 在调用 CH563NET_Init 之前通过 LIB_CFG_VALUE 将配置信息传递给 CH563NETConfig。请参考例程。

由于 LIB_CFG_VALUE 的值是在 CH563NET.H 中进行计算的, 所以应用程序应该在 include "CH563NET.H" 之前对库进行配置, 否则将采用默认配置, 默认配置请参考 CH563NET.H。

V4 版本的库增加了对 IPRAW, UDP, TCP, 内存分配等配置, 本节主要介绍配置的含义以及方法。

(1) CH563NET_MEM_ALIGNMENT

字节对齐, 必须配置为 4 的整数倍。

(2) CH563NET_NUM_IPRAW

用于配置 IPRAW (IP 原始套接字) 连接的个数, 最小值为 1。用于 IPRAW 通讯。

(3) CH563NET_NUM_UDP

用于配置 UDP 连接的个数, 最小值为 1。用于 UDP 通讯。

(4) CH563NET_NUM_TCP

用于配置 TCP 连接的个数, 最小值为 1。用于 TCP 通讯。

(5) CH563NET_NUM_TCP_LISTEN

用于配置 TCP 监听的个数, 最小值为 1。TCP 监听的 socket 仅仅用于监听, 一旦监听到 TCP 连接, 会立即分配一个 TCP 连接, 占用 CH563NET_NUM_TCP 的个数。

(6) CH563NET_NUM_PBUF

用于配置 PBUF 结构的个数, PBUF 结构主要用于管理内存分配, 包括申请 UDP, TCP, IPRAW 内存以及收发内存, 如果应用程序需要较多的 socket 连接和有大批量的数据收发, 此值要设置大些。

(7) CH563NET_NUM_POOL_BUF

POOL BUF 的个数, POOL BUF 主要是用来接收数据时使用, 如果接收大批数据, 此值要设置大些。

(8) CH563NET_NUM_TCP_SEG

TCP Segments 的个数, CH563NET 每次发送一个 TCP 数据包时, 都要先申请一个 TCP Segments。如果 TCP 连接数较多时, 收发数据量较大时, 此值应设置大些。例如当前 TCP 连接有 4 个, 每个接收缓冲区设置为 2 个 TCP_MSS, 假设每次收到一包数据后都会进行一次 ACK, 则 CH563NET_NUM_TCP_SEG 应该配置大于 (4*2), 这只是最严重的情况, 实际上每次 ACK (或者发送的数据) 被收到后, 都会释放此数据的 Segments。

(9) CH563NET_NUM_IP_REASSDATA

IP 分片的包个数。每个包的大小为 CH563NET_SIZE_POOL_BUF, 此值最小可以设置为 1。

(10) CH563NET_TCP_MSS

TCP 最大报文段的长度, 此值最大为 1460, 最小为 60, 报文段的长度 TCP 报文最大数据块长度。综合传输和资源考虑, 建议此值不要小于 536 字节。

(11) CH563_MEM_HEAP_SIZE

堆分配内存大小，堆分配主要用于一些不定长度的内存分配，例如发送数据。如果 TCP 有大量数据收发，则此值应该设置大些。V4 版本的库对发送数据进行了改进，在发送数据时可以使用应用层内存，参考本节 MISCE_CFGO_TCP_SEND_COPY。

(12) CH563NET_NUM_ARP_TABLE

ARP 缓存，存放 IP 和 MAC，此值最小可以设置为 1，最大为 0X7F。如果 CH563NET 需要和 4 台 PC 进行网络通讯，其中两台会大批量收发数据，则建议设置为 4。如果小于 2，则会严重影响通讯效率。

(13) MISCE_CFGO_TCP_SEND_COPY

该配置仅在 TCP 通讯时有效。

MISCE_CFGO_TCP_SEND_COPY 为 1 时表示开启发送复制功能，CH563NET 将应用层数据复制到内部的堆内存中，然后进行打包发送。

MISCE_CFGO_TCP_SEND_COPY 为 0 时表示关闭发送复制功能，详细的使用方法请参考 3.25 章节。

(14) MISCE_CFGO_TCP_RECV_COPY

调试使用，默认为开启，此值为 1 速度会加快。

(15) MISCE_CFGO_TCP_OLD_EDLETE

MISCE_CFGO_TCP_OLD_EDLETE 为 1 时，CH563NET 申请不到 TCP 连接时，会删除较老的 TCP 连接，默认关闭此功能。

4.3. 关于中断

中断分为全局中断和 socket 中断。全局中断中状态定义如下表。

位	名称	描述
[5:7]	-	保留
4	GINT_STAT_SOCKET	socket 中断
3	-	保留
2	GINT_STAT_PHY_CHANGE	PHY 状态改变中断
1	-	保留
0	GINT_STAT_UNREACH	不可达中断

① GINT_STAT_UNREACH，不可达中断。当库收到 ICMP 不可达中断报文后，将不可达 IP 数据包的 IP 地址，端口，协议类型保存到不可达信息表中，然后产生此中断，应用程序可以通过查询 CH563Inf 结构中的 UnreachCode，UnreachProto 和 UnreachPort 来获取不可达的相关信息。

② GINT_STAT_PHY_CHANGE，PHY 变化中断。当 CH563 的 PHY 连接有变化时产生此中断，例如 PHY 状态由连接状态变化为断开状态或者由断开状态变化为连接状态。应用程序可以通过 CH563NET_GetPHYStatus 来获取当前的 PHY 状态

③ GINT_STAT SOCK，socket 中断。当 socket 有中断事件时库会产生此中断，应用程序可以通过 CH563NET_GetSocketInt 来获取 socket 的中断状态。

socket 的中断状态定义如下表：

位	名称	描述
7	-	保留

6	SINT_STAT_TIM_OUT	超时
5	-	保留
4	SINT_STAT_DISCONNECT	TCP 断开
3	SINT_STAT_CONNECT	TCP 连接
2	SINT_STAT_RECV	接收缓冲区非空
[0:1]	-	保留

① SINT_STAT_RECV, 接收缓冲区非空中断, 当 socket 收到数据后, 会产生此中断, 应用层收到此中断后, 应该使用 CH563NET_SocketRecvLen 来获取接收数据长度, 根据长度使用 CH563NET_SocketRecv 来读取接收缓冲区的数据。

② SINT_STAT_CONNECT, TCP 连接中断, 仅在在 TCP 模式下有效 0, 表明 TCP 连接成功, 应用层必须在产生此中断后, 才可以进行数据传输。

③ SINT_STAT_DISCONNECT, TCP 连接断开中断, 仅在 TCP 模式下有效, 表明 TCP 连接断开, 连接断开后, 应用层不得再进行数据传输。

④ SINT_STAT_TIM_OUT, TCP 模式下, TCP 连接、断开、发送数据等过程中出现超时, 会产生此中断。如果发送某些异常情况, 库内部会关闭此连接时, 也会产生该中断。TCP 模式下一旦产生此中断, socket 将会被关闭, 且 socket 的相关配置被清除, 所以应用层如果需要再次使用此 socket 必须重新初始化并连接或者监听。IPRAW、UDP 模式下, 发送数据失败也会产生此中断, 但 socket 不会被关闭。

注: 本章节所表述的中断是指变量的标志, 为了表述方便和易于理解, 在库和本文档中均用中断进行说明, 并非 MCU 的硬件中断。

4.4. 关于 socket

库支持的 socket 种类及个数分配如下表:

种类	个数
IPRAW	1-31
UDP	1-31
TCP 连接	1-31
TCP 监听	1-31

CH563NET 最多支持 31 个 socket。

4.5. IPRAW

创建 IPRAW socket 的步骤:

- ① 设置协议字段, 在 IPRAW 模式下, 即 SourPort;
- ② 设置目标 IP 地址;
- ③ 设置接收缓冲区起始地址和长度;
- ④ 设置协议类型为 PROTO_TYPE_IP_RAW;
- ⑤ 调用 CH563NET_SocketCreat 函数, 将上述设置传递给本函数即可。

CH563NET_SocketCreat 会在 socket 信息列表中找到一个空闲的信息表, 将上述配置复制到此空闲列表中。如果没找到空闲列表将返回错误, 创建成功后返回, 并将空闲信息表输出给应用层。

IP 报文结构:

目的 MAC	源 MAC	类型	IP 首部	IPRAW数据	CRC32
6 Bytes	6 Bytes	2 Bytes	20 Bytes	最大 1480 Bytes	4 Bytes

应用层可以调用 CH563NET_SocketSend 发送数据，发送的数据长度不做限制，库内部会自动循环将数据依次进行发送，IPRAW一包允许发送的最大长度为 1480 字节，如果应用层写入的数据流长度大于 1480 字节，库会将数据流封装成若干个 IP 包进行发送。如果发送失败会立即返回。

IPRAW模式下发送数据可能会产生 SINT_STAT_TIM_OUT 中断，表示数据发送失败，导致发送数据失败一般有以下两个原因：

- ① 如果目的 IP 地址和 CH563 在同一个子网，则可能目的 IP 地址的网络设备不在线。
- ② 如果目的 IP 地址和 CH563 不在同一个子网，则可能 CH563 的网关不在线。

当库收到 IP 数据包后，首先检测协议字段和 socket 设置的协议字段是否相同，如果相同则将 IPRAW数据包复制到接收缓冲区中并产生 SINT_STAT_RECV 中断，应用层收到此中断后，可以调用 CH563NET_SocketRecvLen 获取当前 socket 缓冲区的有效长度，根据长度应用层调用 CH563NET_SocketRecv 读取 socket 接收缓冲区的数据，应用程序可以一次将所有数据读出，也可以分多次读出。由于 IPRAW模式下无法进行流控，建议应用层查询到接收数据中断口后应立即将所有数据读出，以免被后续的数据覆盖。

关于协议字段设置的注意事项

库处理 IPRAW的优先级高于 UDP 和 TCP，如果 IP 协议字段设置为 17(UDP)或者 6(TCP)，则可能存在和其他 socket 冲突的可能性，在使用时应当注意避免，下面列举两种情况进行说明：

① socket0 设置为 IPRAW模式，IP 协议字段为 17，socket1 为 UDP 模式。在 UDP 模式下，IP 包的协议字段也是 17，这样就会导致 socket1 通讯的数据会被 socket0 拦截，无法接收到数据。

② socket0 设置为 IPRAW模式，IP 协议字段为 6，socket1 为 TCP 模式。在 TCP 模式下，IP 包的协议字段也是 6，这样就会导致 socket1 通讯的数据会被 socket0 拦截，无法接收到数据。

库支持 IP 分片功能，但是最大分片的数据包长度不得大于接收缓冲区的长度。

4.6. UDP 客户端

创建 UDP socket 的步骤:

- ① 设置源端口；
- ② 设置目的端口；
- ③ 设置目标 IP 地址；
- ④ 设置接收缓冲区起始地址和长度；
- ⑤ 设置协议类型为 PROTO_TYPE_UDP；
- ⑥ 调用 CH563NET_SocketCreat 函数，将上述设置传递给本函数即可。

CH563NET_SocketCreat 会在 socket 信息列表中找到一个空闲的信息表，将上述配置复制到此空闲列表中。如果没找到空闲列表将返回错误，创建成功后返回，并将空闲信息表输出给应用层。

UDP 报文结构:

目的 MAC	源 MAC	类型	IP 首部	UDP 首部	UDP 数据	CRC32
6 Bytes	6 Bytes	2 Bytes	20 Bytes	8 Bytes	最大 1472 Bytes	4 Bytes

UDP 是一个简单的，不可靠的，面向数据报文的运输层协议，传输速度较快，不能保证数据能达到目的地，必须由应用层来保证传输的可靠稳定。

应用层可以调用 CH563NET_SocketSend 发送数据，发送的数据长度不做限制，库内部会自动循环将数据依次进行发送，UDP 一包允许发送的最大长度为 1472 字节，如果应用层写入的数据流长度大于 1472 字节，库会将数据流封装成若干个 UDP 包进行发送。如果发送失败会立即返回。

UDP 模式下发送数据可能会产生 SINT_STAT_TIM_OUT 中断，表示数据发送失败，在 UDP 模式下产生此中断后，socket 不会被关闭，这点和 TCP 处理有所不同。导致发送数据失败一般有以下两个原因：

- ① 如果目的 IP 地址和 CH563 在同一个子网，则可能目的 IP 地址的网络设备不在线。
- ② 如果目的 IP 地址和 CH563 不在同一个子网，则可能 CH563 的网关不在线。

当库收到 UDP 数据包后，将 UDP 数据包复制到接收缓冲区中并产生 SINT_STAT_RECV 中断，应用层收到此中断后，可以调用 CH563NET_SocketRecvLen 获取当前 socket 缓冲区的有效长度，根据长度应用层调用 CH563NET_SocketRecv 读取 socket 接收缓冲区的数据，应用程序可以一次将所有的数据读出，也可以分多次读出。由于 UDP 模式下无法进行流控，建议应用层查询到接收数据中断口后应立即将所有数据读出，以免被后续的数据覆盖。

4.7. UDP 服务器

UDP 服务器，可以接收任意 IP 地址发送给本机端口的地址。

创建 UDP socket 的步骤：

- ① 设置源端口；
- ② 设置目标 IP 地址，目的地址为 255.255.255.255；
- ③ 设置接收缓冲区起始地址和长度；
- ④ 设置协议类型为 PROTO_TYPE_UDP；
- ⑤ 设置接收回调函数的入口地址；
- ⑥ 调用 CH563NET_SocketCreat 函数，将上述设置传递给本函数即可。

在 UDP 服务器模式下，为了区分数据包的源 IP 和源端口，CH563NET 在 SocketInf 中增加了 AppCallBack 函数指针。当 UDP 接收到数据后，通过 AppCallBack 函数通告应用层此数据包的源 IP，源端口。在 AppCallBack 函数中，应用层应将所有数据全部读，CH563NET 在回调 AppCallBack 后会将接收缓冲区所有相关变量初始化。

如果不通过回调方式接收数据，请务必将 SocketInf 中的 AppCallBack 初始化为 0。

4.8. TCP 客户端

创建 TCP 客户端 socket 的步骤：

- ① 设置源端口；
- ② 设置目的端口；
- ③ 设置目标 IP 地址；
- ④ 设置接收缓冲区起始地址和长度；
- ⑤ 设置协议类型为 PROTO_TYPE_TCP；
- ⑥ 调用 CH563NET_SocketCreat 函数，将上述设置传递给本函数；
- ⑦ 调用 CH563NET_SocketConnect，TCP 将会发起连接。

CH563NET_SocketCreat 会在 socket 信息列表中找到一个空闲的信息表，将上述配置复制到此空闲列表中。如果没找到空闲列表将返回错误，创建成功后返回，并将空闲信息表输出给应用层。

调用 CH563NET_SocketConnect 后，库会主动向远端发起连接请求，连接成功后会产生连接中断 SINT_STAT_CONNECT，如果远端不在线或有其他异常，库会自动重试，重试次数和重试周期可以在应用层设置，如果超过应用层设置的重试次数后仍然不能连接成功，库会自动将 socket 关闭，并产生超时中断 SINT_STAT_TIM_OUT。只有产生连接中断后，应用层才可以用此 socket 进行数据收发。

TCP 报文结构：

目的 MAC	源 MAC	类型	IP 首部	TCP 首部	TCP 数据	CRC32
6 Bytes	6 Bytes	2 Bytes	20 Bytes	20 Bytes	最大 1460Bytes	4 Byte

TCP 提供面向连接的，可靠的字节流服务。

Unack Segment 指未成功发送的 TCP 报文。

CH563NET TCP 模式有两种发送方式：

1：复制方式，是指将用户的数据复制到 Mem_Heap_Memory 中发送，数据总长度不做限制，如果长度大于 CH563NET_TCP_MS，CH563NET 会将数据分成若干个大小为 CH563NET_TCP_MSS 的 TCP 包发送。复制方式一般用在 socket 数量较少，发送数据量相对少的情况下，应用层只需要调用 CH563NET_SocketSend 函数即可。

2：非复制方式，是指直接使用用户缓冲区进行发送。数据长度最大 CH563NET_TCP_MSS，非复制方式一般用在 socket 数量较多、发送数据量多和对 RAM 苛刻的情况下。在使用非复制方式需要注意：

调用 CH563NET_SocketSend(sockeid, tcpdata, &len) 发送，len 必须不能大于 TCP_MSS，tcpdata 不可以为局部或者在栈中分配的缓冲区，且在调用 CH563NET_SocketSend 后应用层不能再使用 tcpdata 缓冲区，直到 CH563NET 通知应用层此缓冲区的数据段被成功发送。

CH563NET 通过 AppCallBack 来通知应用层数据段成功发送，AppCallBack 的原型如下：

```
void (*AppCallBack)(struct _SCOK_INF *socinf, UINT32 ipaddr, UINT16 port, UINT8 *buf, UINT32 len);
```

在 TCP 模式下 AppCallBack 用于通知应用层此 socket Unack Segment 的个数，socinf 为 socket 信息，len 为 Unack Segment 个数，应用层获取到个数后可以调用 CH563NET_QueryUnack 来获取这些报文的信息，如果 tcpdata 还没发送成功，CH563NET 会将 tcpdata（缓冲区地址）写入到 addrlist 中。关于 CH563NET_QueryUnack 的用法可以参考 3.25。关于发送方式的配置请参考 4.2

在 TCP 模式下，如果数据发送失败会产生 SINT_STAT_TIM_OUT 中断，应用层应该关闭 socket。

当库收到 TCP 数据包后，将 TCP 数据包复制到接收缓冲区中并产生 SINT_STAT_RECV 中断，应用层收到此中断后，可以调用 CH563NET_SocketRecvLen 获取当前 socket 缓冲区的有效长度，根据长度应用层调用 CH563NET_SocketRecv 读取 socket 接收缓冲区的数据，应用程序可以一次将所有数据读出，也可以分多次读出。在 TCP 模式下应用层每次调用 CH563NET_SocketRecv，库将把接收数据复制给应用层的接收缓冲区，然后向远端通告当前的窗口大小。关于 CH563NET_SocketRecv 的用法请参考 3.18。

4.9. TCP 服务器

创建 TCP 服务器 socket 的步骤:

- ① 设置源端口;
- ② 设置协议类型为 `PROTO_TYPE_TCP`;
- ③ 调用 `CH563NET_SocketCreat` 函数, 将上述设置传递给本函数;
- ④ 调用 `CH563NET_SocketListen`, TCP 将会进入监听;

上述步骤会建立一个监听的 socket, 此 socket 仅仅是监听客户端连接, 本身不进行数据收发, 所以无需设置接收缓冲区。

如果有一个客户端连接成功后, 监听的 socket 将会从 socket 信息列表中找一个空闲的列表, 如果没找到空闲列表, 则会将此连接断开。如果找到, 则会对此列表初始化并将目的 IP, 源端口和目的端口等信息写入此列表中, 并产生连接中断 `SINT_STAT_CONNECT`, 应用层软件接收此中断后, 应立即调用 `CH563NET_ModifyRecvBuf` 为此连接分配一个接收缓冲区。如果应用软件建立多个服务器, 可以通过查询 socket 信息列表中的源端口来确定此连接是哪个服务器的连接。

关于数据结构, 发送数据和接收数据流程可以参考 TCP 客户端模式。