User mannual

M22A-F/N/U20 系列 ARM 嵌入式工业控制模块

以太网通信函数参考手册

UM05011001 V1.5 Date: 2008/04/26 产品用户手册

类别	内容
关键词	ZLG/IP、MiniARM、以太网函数库
摘　要	简要介绍 ZLG/IP 通讯函数库在 MiniARM® M22A 系列产品上的使用方法

修订历史

版本	日期	原因
V1.0	2006-08-26	创建文档
V1.1	2006-08-30	修改、校对文档
V1.2	2007-09-17	修改版面
V1.3	2007-12-17	修改销售服务网络联系方式
V1.4	2008-04-03	修改示例
V1.5	2008-04-26	增加 connectEx 函数介绍

销售与服务网络

广州致远电子有限公司

地址：广州市天河区车陂路黄洲工业区 3 栋 2 楼　　邮编：510660
电话：(020) 22644249　28872524　22644399　28872342
　　28872349　28872569　28872573
传真：**(020) 38601859**
网站：**www.embedtools.com　www.embedcontrol.com　www.ecardsys.com**

广州周立功单片机发展有限公司

地址：广州市天河北路 689 号光大银行大厦 12 楼 F4　邮编：510630
电话：(020) 38730916　38730917　38730972　38730976　38730977
传真：(020)38730925
网址：http://www.zlgmcu.com

广州专卖店
地址：广州市天河区新赛格电子城 203-204 室
电话：(020)87578634　87569917
传真：(020)87578842

南京周立功
地址：南京市珠江路 280 号珠江大厦 2006 室
电话：(025)83613221　83613271　83603500
传真：(025)83613271

北京周立功
地址：北京市海淀区知春路 113 号银网中心 712 室（中发电子市场斜对面）
电话：(010)62536178　62536179　82628073
传真：(010)82614433

重庆周立功
地址：重庆市石桥铺科园一路二号大西洋国际大厦（赛格电子市场）1611 室
电话：(023)68796438　68796439
传真：(023)68796439

杭州周立功
地址：杭州市登云路 428 号浙江时代电子市场 205 号
电话：(0571)88009205　88009932　88009933
传真：(0571)88009204

成都周立功
地址：成都市一环路南二段 1 号数码同人港 401 室（磨子桥立交西北角）
电话：(028) 85439836　85437446
传真：(028)85437896

深圳周立功
地址：深圳市深南中路 2070 号电子科技大厦 A 座 24 楼 2403 室
电话：(0755)83781788（5 线）
传真：(0755)83793285

武汉周立功
地址：武汉市洪山区广埠屯珞瑜路 158 号 12128 室（华中电脑数码市场）
电话：(027)87168497　87168297　87168397
传真：(027)87163755

上海周立功
地址：上海市北京东路 668 号科技京城东座 7E 室
电话：(021)53083452　53083453　53083496
传真：(021)53083491

西安办事处
地址：西安市长安北路 54 号太平洋大厦 1201 室
电话：(029)87881296　83063000　87881295
传真：(029)87880865

目录

1. Ethernet 通讯函数库简介

ZLG/IP 通讯函数库是广州致远电子有限公司自主开发的基于 TCP/IP 通讯协议的程序包，通过该程序包，用户可以在嵌入式系统中实现各种联网功能。

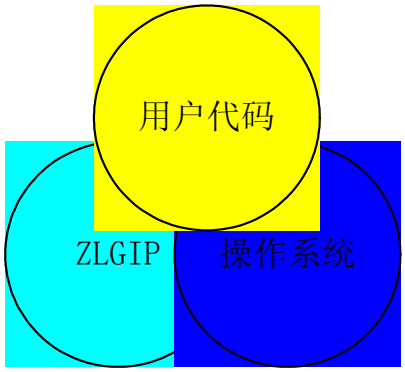

图 1.1 基于 Zlg/IP 的应用程序的结构图

使用 ZLG/IP 实现完整的联网功能的系统结构如图 1.1 所示，包含 3 个部分，ZLG/IP、操作系统、用户代码。在 MiniARM 系列产品中，ZLG/IP 和操作系统集成在核心板程序中，用户只需要编写用户代码就可以了。

这里将简单介绍 ZLG/IP 各个函数功能，以及如何使用 ZLG/IP 建立网络连接。

2. ZLG/IP 通讯函数说明

关于 ZLG/IP 的通讯函数我们分 3 部分来说明，第一部分是 ZLG/IP 的初始化函数，第二部分是套接字函数，第三部分是套接字数据库函数。

2.1 ZLG/IP 的初始化函数

在调用 ZLG/IP 通讯函数库之前，需进行函数库的加载和配置。首先要设置网络相关参数，需要设置的网络设置参数见程序清单 2.1

程序清单 2.1 配置网络参数

```
uint8    MCU_IP[4]      = {192, 168,   0, 253};
uint8    MCU_Mark[4]    = {255, 255, 255,   0};
uint8    MCU_GateWay[4] = {192, 168,   0, 254};
uint16   MCU_Port       = 1200;
uint8    PC_IP[4]       = {192, 168,   0, 131};
uint16   PC_Port        = 2200;
```

ZLG/IP 的初始化非常简单，直接调用 ZlgipInitial()函数（如表 2.1）就可以了，如程序清单 2.2 所示，要注意其输入参数。

程序清单 2.2 初始化函数的调用

```
void main(void)
{
    /*其他初始化代码*/
    ZlgipInitial(MCU_IP, MCU_GateWay, MCU_Mark);
    /* 添加自己的初始化代码 */
    /* Add your codes here */
}
```

表 2.1 Zlgip 初始化函数

函数名称	ZlgipInitial ()
函数原型	void ZlgipInitial(const uint8 *ip, const uint8 *Gateway, const uint8 *Mark)
功能描述	初始化 TCP/IP 协议栈。
函数参数	ip ：IP 地址； Gateway ：网关； Mark ：子网掩码。
函数返回值	无
特殊说明和注意点	用户如果想调用 ZLG/IP 函数库，必须先调用此初始化函数。

范例	uint8 MyIp[] = {192.168. 0.235}; //本低 IP 地址
	uint8 MyGateWay[] = {192.168. 0. 1}; //网关
	uint8 MyMark[] = {255,255,255, 0}; //子网掩码
	…
	ZlgipInitial(MyIp, MyGateWay, MyMark); //TCP/IP 初始化
	…

2.2 套接字（Socket）函数

套接字（Socket）是 TCP/IP 网络通讯的基本构建模块，它实际是一个可以关联名字的通信端点。不论是客户机工作方式还是服务器工作方式，在进行通讯前必须创建各自的套接字并建立连接。创建套接字由 socket()函数（见表 2.2）完成。

表 2.2 socket 函数

函数名称	socket()
函数原型	SOCKET socket(uint16 af, uint16 type, uint16 protocol)
功能描述	创建一个 SOCKET。
函数参数	af ：地址格式说明，在本协议中保留，常为 0。 type ：通讯类型，有 SOCK_STREAM 和 SOCK_DGRAM 两种类型，SOCK_STREAM 表示要创建的是流套接字，SOCK_DGRAM 表示要创建的是数据报套接字。 protocol：用于该套接字的一个特定通讯协议，可以是 TCP，也可以是 UDP。
函数返回值	如果函数成功执行，将返回一个新的套接字描述符。否则将返回 INVALID_SOCKET。
特殊说明和注意点	必须先建立一个 SOCKET 结构，该函数可用于 TCP 或 UDP 通讯任务。
范例	SOCKET s; … s = socket(0, SOCK_STREAM, TCP_PROTOCOL); …

将本地地址绑定到所创建的套接字上，以在网络上标识该套接字是通过 bind()函数来完成。函数见表 2.3。其中 bind()函数的第二个参数 name 是赋予套接字的地址指针，它由 struct sockaddr 结构表示，该结构的格式定义如程序清单 2.3.

程序清单 2.3 sockaddr 结构定义

```
struct  sockaddr{
            uint16      sin_family;
            uint8       sin_addr[4];
            uint16      sin_port;
    };
```

sin_family 必须设置为 0。sin_addr 用于把一个 IP 地址保存为一个 4 字节的值，可以表示一个本地的或远程的 IP 地址。sin_port 用于指定服务端口。

表 2.3 bind 函数

函数名称	bind()
函数原型	int bind(SOCKET s, struct sockaddr * name,uint16 namelen)

功能描述	对已创建但尚未连接的 SOCKET 绑定本地 IP 地址和服务端口。
函数参数	s ：已创建的 SOCKET； name ：保存 IP 地址和端口的结构； namelen ：name 的长度。
函数返回值	0 为成功，SOCKET_ERROR 为出错。
特殊说明和注意点	如果调用本函数但没有指定绑定的 IP 和服务端口，系统自动使用默认 IP 和一个空闲的端口，该函数可用于 TCP 或 UDP 通讯任务

范例	int iii;
	struct sockaddr clientaddr;
	…
	clientaddr.sin_family=0;
	getlocalip(clientaddr.sin_addr,0);
	clientaddr.sin_port=1025;
	iii=bind(s, (struct sockaddr*)&clientaddr,sizeof(clientaddr));
	…

listen()将套接字置入监听模式并准备接受连接请求，函数见表 2.4。

表 2.4 listen 函数

函数名称	listen()
函数原型	int listen(SOCKET s, uint16 backlog)
功能描述	将 TCP 服务器置入监听模式，并设定服务器需要监听的连接数。
函数参数	s ：已捆绑但并未连接的 SOCKET； backlog：需要监听的连接数，不大于 MAX_TCP_LINKS。
函数返回值	SOCKET_ERROR ：调用失败； 返回值等于 backlog 的值：表示设定成功。
特殊说明和注意点	只适用于 TCP 连接，backlog 不能大于 MAX_TCP_LINKS。
范例	int ei; … ei=listen(s, 2); …

进入监听状态后，通过调用 accept()函数使套接字做好接受客户连接的准备，函数见表 2.5。

表 2.5 accept 函数

函数名称	accept()
函数原型	uint8 accept(SOCKET s, struct sockaddr * addr, int *addrlen)
功能描述	用于 TCP 服务器确认客户机的连接。
函数参数	s ：已创建的 SOCKET； addr ：连接后保存对方的 IP 和端口； addrlen ：addr 的长度。
函数返回值	SOCKET_ERROR：没有客户机的连接； 返回值小于 MAX_TCP_LINKS：已经与客户机建立连接，返回值是“连接序号”。

特殊说明和注意点	只适用于 TCP 连接，本函数不阻塞，它属于超时退出。

范例	uint8 Temp；
	int addrlen;
	struct sockaddr cliaddr;
	…
	while (1)
	{
	Temp=accept(s, (struct sockaddr*)&cliaddr,&addrlen);
	if(Temp!= SOCKET_ERROR)
	{
	… //连接成功
	}
	…
	}

如果作为客户机想连接接服务器，可以通过 connect()函数实现。函数见**表 2.6**。

表 2.6 connect 函数

函数名称	connect()
函数原型	uint8 connect(SOCKET s, struct sockaddr * addr,uint16 addrlen)
功能描述	用于 TCP 主动连接（一般是 TCP 客户端）。
函数参数	s ：已创建的 SOCKET 指针； addr ：对方 IP 地址和端口； addrlen ：addr 的长度。
函数返回值	< MAX_TCP_LINKS：返回创建的“连接序号”； SOCKET_ERROR：连接失败返回 SOCKET_ERROR。
特殊说明和注意点	只适用于 TCP 客户机连接，本函数不阻塞，默认超时 500ms 退出。

范例	while (1)
	{
	Temp= connect (s, (struct sockaddr*)&cliaddr,&addrlen);
	if(Temp< MAX_TCP_LINKS)
	{
	
	}
	
	}

connectEx 函数是上述函数的升级版本，方便用户根据实际网络情况，自行设定超时退出的阻塞时间。见表 2.7

表 2.7 connectEx 函数

函数名称	connectEx ()
函数原型	uint8 connectEx(SOCKET s, struct sockaddr * addr, uint16 addrlen, INT32U ulTimeOut)
功能描述	用于 TCP 主动连接（一般是 TCP 客户端）。

函数参数	s ：已创建的 SOCKET 指针； addr ：对方 IP 地址和端口； addrlen ：addr 的长度。 ulTimeOut：超时时间(ms)
函数返回值	< MAX_TCP_LINKS：返回创建的“连接序号”； SOCKET_ERROR：连接失败返回 SOCKET_ERROR。
特殊说明和注意点	1、对于 M22A-N20 系列 MiniARM 不适用； 2、只适用于 M22A-FNU20 系列 MiniARM 的 TCP 客户机连接，本函数不阻塞，ulTimeOut 为“0”时，默认超时 500ms 退出。

范例	while (1)
	{
	Temp= connectEx (s, (struct sockaddr*)&cliaddr,&addrlen,200);
	if(Temp< MAX_TCP_LINKS)
	{
	
	}
	
	}

如果是 TCP 连接，可以通过 recv()函数接收指定连接的数据，函数见表 2.8。

表 2.8 recv 函数

函数名称	recv()
函数原型	uint16 recv(uint8 num, uint8 *buf, uint16 len, int flags)
功能描述	TCP 通讯连接建立后，读取所得到的数据。
函数参数	num ：从 accept()或 connect()函数获取的“连接序号”； buf ：接收缓存区起始地址； len ：接收的数据长度； flags ：保留。
函数返回值	读取的数据长度，0 表示没有数据，SOCKET_RCV_ERROR 表示连接出错。
特殊说明和注意点	只适用于 TCP 连接，本函数不阻塞，它属于超时退出。

范例	uint8 Temp,tempdata[1500];
	int addrlen;
	uint16 TempLength;
	struct sockaddr cliaddr;
	Temp=accept(s, (struct sockaddr*)&cliaddr,&addrlen);
	if(Temp!= SOCKET_ERROR)
	{
	TempLength=recv(Temp,tempdata ,1500, 0);
	if(TempLength == SOCKET_RCV_ERROR)
	{
	…//close the tcp link.
	}
	else if(TempLength>0)
	{
	…//Handle the revice data.
	}
	}

如果使用 UDP 连接，可以通过 recvfrom()函数接收数据，函数见表 2.9。

表 2.9 recvfrom 函数

函数名称	recvfrom()
函数原型	uint16 recvfrom(SOCKET s, uint8 *buf, uint16 len, int flags, struct sockaddr *from, uint16 *fromlen)
功能描述	用于 UDP 通讯时接收数据。
函数参数	s ：已创建的 SOCKET； buf ：接收缓存区起始地址； len ：接收的数据长度； flags ：保留； from ：保存发送方的 IP 地址和端口； fromlen ：from 的长度。
函数返回值	接收到的数据的长度。
特殊说明和注意点	仅适用与 UDP 通讯，使用前必须先建立 SOCKET。

范例	uint8 rec_buffer[200];
	SOCKET s;
	int rec_coute;
	uint16 iii;
	struct sockaddr servaddr,;
	…
	while (1)
	{
	rec_coute=recvfrom(s, rec_buffer, 200, 0, (struct sockaddr*)&servaddr, &iii);
	…
	}

如果是 TCP 连接，可以通过 send()函数发送数据到指定的连接，函数见**表 2.10**。

表 2.10 send 函数

函数名称	send()
函数原型	uint32 send(uint8 num, uint8 *buf, uint32 len, int flags)
功能描述	TCP 通讯连接建立后，发送数据。
函数参数	num ：从 accept()或 connect()函数获取的“连接序号”; buf ：发送数据起始地址; len ：发送的数据长度; flags ：保留。
函数返回值	发送的数据长度，0 表示发送失败, SOCKET_SEN_ERROR 表示连接出错。
特殊说明和注意点	只适用于 TCP 连接，本函数不阻塞，超时重发一次，再超时退出。

范例	uint8 Temp,tempdata[1000];
	int addrlen;
	uint16 TempLength;
	struct sockaddr cliaddr;
	…
	while (1)
	{
	Temp=accept(s, (struct sockaddr*)&cliaddr,&addrlen);
	if(Temp!= SOCKET_ERROR)
	{
	TempLength= send (Temp,tempdata ,1000, 0);
	if(TempLength＝＝1000)
	{
	…//send ok!
	}
	…
	}
	…
	}

如果是 UDP 连接，可以通过 sendto()函数发送数据，函数见表 2.11。

表 2.11 sendto 函数

函数名称	sendto()
函数原型	uint16 sendto(SOCKET s, uint8 *buf, uint16 len, int flags, struct sockaddr *to, uint16 tolen)
功能描述	UDP 通讯方式时发送数据。
函数参数	s ：已创建的 SOCKET; buf ：发送数据起始地址; len ：发送的数据长度（不大于以太网中的 UDP 最大数据 1478 个字节）; flags ：保留; to ：目标方的 IP 地址和端口; tolen ：to 的长度。

函数返回值	发送的数据的长度。
特殊说明和注意点	仅适用与 UDP 通讯，使用前必须先建立 SOCKET。

范例	uint8 send_buffer[200];
	SOCKET s;
	int send_coute;
	uint16 iii;
	struct sockaddr cliaddr,;
	…
	while (1)
	{
	send_coute=sendto(s, send_buffer, 200, 0, (struct sockaddr*)&cliaddr, &iii);
	…
	}

建立 TCP 连接后，可以通过 close()函数断开连接，函数见表 2.12。

表 2.12 close 函数

函数名称	close()
函数原型	uint8 close(uint8 num)
功能描述	关断 TCP 连接。
函数参数	num：从 accept()函数或 connect()函数获取的 “连接序号”。
函数返回值	num：返回连接号表示正常断开； SOCKET_ERROR：非正常断开。
特殊说明和注意点	只适用于 TCP 连接，本函数不阻塞，超时退出。

范例	Temp=accept(s, (struct sockaddr*)&cliaddr,&addrlen);
	if(Temp!= SOCKET_ERROR)
	{
	…
	close(Temp);
	}

当要关闭一个 socket 连接时，可以通过 close()函数实现，但调用此函数不能完全释放资源，由于 MiniARM 资源有限，需要将 socket 连接占用的资源完全释放，调用 TCP_Abort()函数可以实现此功能，函数见表 2. 13。

表 2.13 TCP_Abort 函数

函数名称	TCP_Abort ()
函数原型	uint8 TCP_Abort(uint8 num)
功能描述	断开 TCP 连接。
函数参数	num：网络端口号.
函数返回值	TCP 的连接状态。
特殊说明和注意点	无

范例	Temp=accept(s, (struct sockaddr*)&cliaddr,&addrlen);
	if(Temp!= SOCKET_ERROR)
	{
	…
	close(Temp);
	delay(50ms);
	TCP_Abrot(Temp);
	}

关闭建立的 socket 使用 closesocket()函数，函数见表 2.14。

表 2.14 closesocket 函数

函数名称	closesocket()
函数原型	Int closesocket(SOCKET s)
功能描述	删除已建立的 SOCKET。
函数参数	s：已创建的 SOCKET。
函数返回值	1：正确删除。
特殊说明和注意点	该函数可用于 TCP 或 UDP 通讯任务。
范例	… closesocket(s);

2.3 套接字数据库函数

获取本地的 IP 地址，可以通过函数 getlocalip()完成，函数见表 2.15。

表 2.15 getlocalip 函数

函数名称	getlocalip ()
函数原型	void getlocalip(uint8 * outptr,uint8 num)
功能描述	用于获取对应网络端口的 IP 地址。
函数参数	outptr：用于保存输出的 IP 地址值； num ：网络端口号，默认应该设为 0。
特殊说明和注意点	适用于用户的通讯任务，来获取本地 IP 地址。
范例	uint8 LocalIpAddr[4]; … getlocalip(LocalIpAddr,0);

对于某个 socket，可以通过 getpeername()或 getsocketname()来获取绑定在 socket 上的 IP 和端口，函数见表 2.16、表 2.17。

表 2.16 getpeername 函数

函数名称	getpeername ()
函数原型	int getpeername(SOCKET s, struct sockaddr *name,int *namelen)
功能描述	在 SCOKET 建立后，获取 SOCKET 的本地绑定 IP 和端口。

函数参数	s ：已创建的 SOCKET； name ：用于保存返回的本地绑定 IP 地址和端口； namelen ：name 的长度。
函数返回值	1：出错； 0：正确。
特殊说明和注意点	使用前必须先建立 SOCKET。
范例	SOCKET s; int iii,temp; struct sockaddr cliaddr,; … temp= getpeername (s, &cliaddr, &iii);

表 2.17 getsockname 函数

函数名称	getsockname ()
函数原型	int getsockname (SOCKET s, struct sockaddr *name,int *namelen)
功能描述	功能与 getpeername()相同，在 SCOKET 建立后，获取 SOCKET 的本地绑定 IP 和端口。
函数参数	s ：已创建的 SOCKET； Name ：用于保存返回的本地绑定 IP 地址和端口； namelen：name 的长度。
函数返回值	1：出错； 0：正确。
特殊说明和注意点	使用前必须先建立 SOCKET。
范例	SOCKET s; int iii,temp; struct sockaddr cliaddr,; … temp= getsockname (s, &cliaddr, &iii);

如果想获取 socket 的连接状态，可以通过 gersocktcpsta()函数完成，函数见

表 2.18。

表 2.18 getsocktcpsta 函数

函数名称	getsocktcpsta ()
函数原型	uint8 getsocktcpsta(uint8 linknum)
功能描述	用于 TCP 通讯时获取连接状态。
函数参数	Linknum：TCP 的连接号，即调用 ACCEPT 或 CONNETC 函数的返回值。
函数返回值	TCP 的连接状态。 0：ERROR；1：关闭状态；2：监听状态；3：连接状态；4：断开状态。

特殊说明和注意点	仅适用与 TCP 通讯，使用前必须先建立 SOCKET，
范例	… iii= getsocktcpsta(TEMP);

对于一个已经建立连接的 socket，如果想获取连接此 socket 的 IP 地址和端口，可以通过调用 getsockcliaddr()函数实现，函数见表 2.19。

表 2.19 getsockcliaddr 函数

函数名称	getsockcliaddr ()
函数原型	int getsockcliaddr(uint8 linknum, struct sockaddr * addr,int * addrlen)
功能描述	用于 TCP 通讯时获取对方的 IP 地址和端口。
函数参数	linknum ：TCP 的连接号； addr ：用于保存对方的 IP 地址和端口； addrlen ：to 的长度。
函数返回值	操作结果。
特殊说明和注意点	仅适用与 TCP 通讯，使用前必须先建立 SOCKET。
范例	int iii； struct sockaddr cliaddr,; … getsockcliaddr (temp, &cliaddr, &iii);

不同的处理器有不同的字节存储顺序，网络默认的排列规则是“Big-Endian”，MiniARM 的存储排列规则是“Little-Endian”。ZLG/IP 函数对 IP 地址和端口号的引用和传递都是按照网络顺序组织的，因此在某些情况下，进行网络顺序与 CPU 顺序的转换无法避免。下面介绍这些转换字节顺序的函数。

表 2.20 htonl 函数

函数名称	htonl ()
函数原型	uint32 htonl(uint32 hostlong)
功能描述	改变长整形数据的排列顺序，把 CPU 顺序变成网络顺序
函数参数	hostlong：输入的数据
函数返回值	返回改变顺序的数据
特殊说明和注意点	无
范例	uint32 iii; iii= htonl (iii);

表 2.21 htons ()函数

函数名称	htons ()
函数原型	uint16 htons(uint16 hostshort)

功能描述	改变短整形数据的排列顺序，把 CPU 顺序变成网络顺序
函数参数	hostshort：输入的数据
函数返回值	返回改变顺序的数据
特殊说明和注意点	无
范例	uint16 iii; iii= htons (iii);

表 2.22 inet_addr 函数

函数名称	inet_addr ()
函数原型	uint32 inet_addr(char * inaddr)
功能描述	把 XXX.XXX.XXX.XXX 的 IP 地址字符转换为网络顺序的长整形
函数参数	* inaddr：输入的字符串
函数返回值	返回网络顺序的长整形
特殊说明和注意点	无
范例	uint32 iii; char temp[]={“192.168.0.22”} iii = inet_addr (temp);

表 2.23 inet_ntoa 函数

函数名称	inet_ntoa ()
函数原型	void inet_ntoa(uint8 *in , char * out)
功能描述	改变数组型的 IP 地址数据为字符型的 XXX.XXX.XXX.XXX 地址结构
函数参数	*in ：输入 IP 地址数据指针 *out ：输出的字符型 IP 地址结构
函数返回值	返回改变顺序的数据
特殊说明和注意点	无
范例	uint8 iii[4]={192,168,0,22}; char temp[16]… … inet_ntoa (iii, temp);

表 2.24 ntohl 函数

函数名称	ntohl ()
函数原型	uint32 ntohl(uint32 netlong)
功能描述	改变长整形数据的排列顺序，把网络顺序变成 CPU 顺序
函数参数	netlong：输入的数据

函数返回值	返回改变顺序的数据
特殊说明和注意点	无
范例	uint32 iii; … iii= ntohl (iii);

表 2.25 ntohs 函数

函数名称	ntohs ()
函数原型	uint16 ntohs(uint16 netlong)
功能描述	改变短整形数据的排列顺序，把网络顺序变成 CPU 顺序
函数参数	netlong：输入的数据
函数返回值	返回改变顺序的数据
特殊说明和注意点	无
范例	uint16 iii; … iii= ntohs (iii);

3. 示范例程

SOCKET API 函数从使用的方式来分有 3 种，一种是通用函数，就是 TCP 或 UDP 通信都使用的函数；一种是 TCP 专用函数，就是只在 TCP 通信中使用的函数；一种是 UDP 专用函数，就是只在 UDP 通信中使用的函数。下面先介绍 TCP 通信中 SOCKET API 函数的使用。

3.1 SOCKET API 函数在 TCP 通信中的使用

TCP 通信的任务分为服务器方式和客户机方式两种。服务器方式是需要监听连接，只有在与客户机建立连接后才能进行数据处理。客户机方式是主动连接服务器，它也是在连接成功后才能进行数据处理。图 3.1 就是 TCP 通讯时服务器端和客户机端通讯的函数应用过程图。

程序清单 3.1 是一个服务器任务的例子，用户可以参考该任务中 SOCKET API 函数的使用来编写自己的程序。客户机端任务如

程序清单 3.2，用户可以参考该任务中 SOCKET API 函数的使用来编写自己的客户端程序。

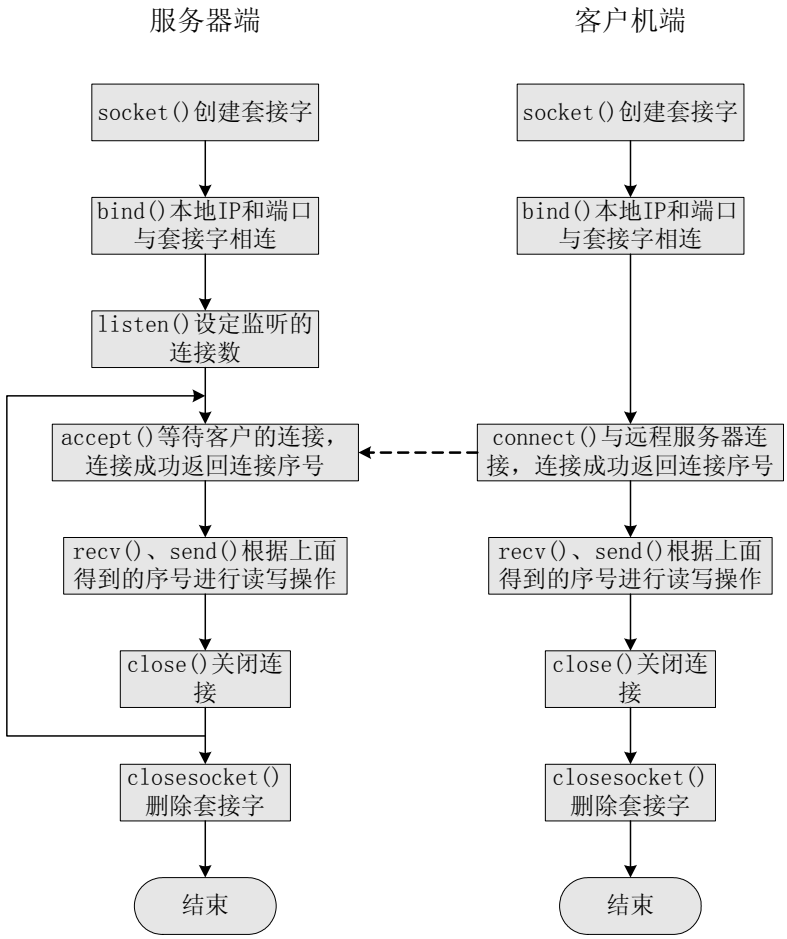

图 3.1 TCP 通讯时 SOCKET API 函数的应用

程序清单 3.1 TCP 服务器程序编写实例

```
#include "main.h"

#define TASK0_PRIO              17                          // 任务的优先级
#define TASK0_ID                TASK0_PRIO                  // 任务的 ID
#define TASK0_STACK_SIZE        1024                        // 定义用户堆栈长度

#define TASK1_PRIO              18                          // 任务的优先级
#define TASK1_ID                TASK1_PRIO                  // 任务的 ID
#define TASK1_STACK_SIZE        512                         // 定义用户堆栈长度

OS_STK    TASK0_STACK[TASK0_STACK_SIZE];
OS_STK    TASK1_STACK[TASK1_STACK_SIZE];

void TASK0(void *pdata);
void TASK1(void *pdata);

/*********************************************************************************************/
uint8     MCU_IP[]          = {192, 168,     0, 253};       // 设置 MCU 的 IP 地址等信息
uint8     MCU_Mark[]        = {255, 255, 255,     0};
uint8     MCU_GateWay[]     = {192, 168,     0, 254};
uint16    MCU_Port          = 1200;
uint16    UDP_Port          = 3200;
uint8     MCU_MAC[6];

uint8     PC_IP[]           = {192, 168,     0, 47};
uint16    PC_Port           = 2200;

OS_EVENT *TCP_CALL_Sem;

/*********************************************************************************************
** Function name: main
** Descriptions : 主函数
** Input        : 无
** Output       : 无
*********************************************************************************************/
int main (void)
{
#if OS_CRITICAL_METHOD == 3                               // Allocate storage for CPU status register
    OS_CPU_SR    cpu_sr;
#endif
```

```
/*----------------------   系统初始化代码   ------------------------------------------------*/
        TargetInit ();                                  //系统初始化,版本号验证,验证不通过函数不会返回!!!
        PinInit();                                                      //  驱动库初始化

        ZlgipInitial(MCU_IP, MCU_GateWay, MCU_Mark);            //  设置系统 IP 参数
        TCP_CALL_Sem = OSSemCreate(1);

       GpioSet(BUZZER);                                                 //  鸣叫 BEEP
       OSTimeDly(OS_TICKS_PER_SEC/10);
        GpioClr(BUZZER);
       OSTimeDly(OS_TICKS_PER_SEC/10);
       GpioSet(BUZZER);
       OSTimeDly(OS_TICKS_PER_SEC/10);
       GpioClr(BUZZER);

/*-------------------------------------------------------------------------------------------*/

       OSTaskCreateExt(TASK0,
                              (void *)0,
                              &TASK0_STACK[TASK0_STACK_SIZE-1],
                              TASK0_PRIO,
                              TASK0_ID,
                              &TASK0_STACK[0],
                              TASK0_STACK_SIZE,
                              (void *)0,
                              OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);

       OSTaskCreateExt(TASK1,
                              (void *)0,
                              &TASK1_STACK[TASK1_STACK_SIZE-1],
                              TASK1_PRIO,
                              TASK1_ID,
                              &TASK1_STACK[0],
                              TASK1_STACK_SIZE,
                              (void *)0,
                              OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);

        while (1) {
               OSTimeDly(OS_TICKS_PER_SEC);
       }
}

void timedly(int nms)
```

```
{
    OSTimeDly(nms);
}

#define TCP_SERNUM    1                                    //TCP 服务端连接个数

void ethernet_tcp_ser()
{
    uint8     Temp;
    uint16    recvnum;
    uint32    sendnum;
    SOCKET    s;
    int       ei, i;
    int       clilen[TCP_SERNUM];
    struct    sockaddr servaddr, cliaddr[TCP_SERNUM];

     uint8 TCPSerLinkFlag[TCP_SERNUM];                    // 服务器连接数目标置
     uint8 TCPSerNo[TCP_SERNUM];                          // 服务器连接序号
     uint8       linknum;                                 // 服务器连接个数
     uint8     err;

     uint8     datatemp[20], recvtemp[100];

     memset(datatemp, 0, 20);
     memset(recvtemp, 0, 100);
     for (i=0; i<TCP_SERNUM; i++) {
          TCPSerLinkFlag[i] = 0;
          TCPSerNo[i] = 0;
     }

    linknum = 0;                                          // 设置连接个数初值
     datatemp[0]      ='M';
     datatemp[1]      ='i';
     datatemp[2]      ='n';
     datatemp[3] ='i';
     datatemp[4] ='A';
     datatemp[5]      ='R';
     datatemp[6]      ='M';
     datatemp[7]      ='-';
     datatemp[8]      ='-';
     datatemp[9]      ='T';
     datatemp[10] ='C';
     datatemp[11] ='P';
```

```
servaddr.sin_family = 0;                                    // 设置 MCU(服务端)的 IP 地址和端口
getlocalip(servaddr.sin_addr,0);
servaddr.sin_port = PC_Port;

OSSemPend(TCP_CALL_Sem, 0, &err);
s = socket(AF_INET, SOCK_STREAM, TCP_PROTOCOL);             // 绑定 MCU(客户端)端口
if (s != INVALID_SOCKET) {                                  // 成功
    ei = bind(s, (struct sockaddr*)&servaddr, sizeof(servaddr));
    while (ei == SOCKET_ERROR) {                            // 失败
        ei = bind(s, (struct sockaddr*)&servaddr, sizeof(servaddr));
    }
}
ei = listen(s, TCP_SERNUM);                                 // 开始侦听
OSSemPost(TCP_CALL_Sem);

if (ei != SOCKET_ERROR) {                                   // 侦听正确
    while (1) {
        if (linknum < TCP_SERNUM) {
            for (i=0; i<TCP_SERNUM; i++) {
                if (TCPSerLinkFlag[i] == 0) {
                    break;
                }
            }

            OSSemPend(TCP_CALL_Sem, 0, &err);
            Temp = accept(s,&cliaddr[i], clilen+i);
            OSSemPost(TCP_CALL_Sem);

            if(Temp != SOCKET_ERROR) {                      // 连接成功
                TCPSerLinkFlag[i] = 1;                      // 设置成功连接标准
                TCPSerNo[i]=Temp;
                linknum++;                                  // 连接数目+1
            }
        }
        timedly(10);

        for (i=0; i<TCP_SERNUM; i++) {
            if (TCPSerLinkFlag[i] == 1) {                   // 连接
                OSSemPend(TCP_CALL_Sem, 0, &err);
                if (3 == getsocktcpsta(TCPSerNo[i])) {      // 确认网络是否出于连接状态
                    recvnum = recv(TCPSerNo[i], recvtemp, 50, 0 );
                    if (recvnum == SOCKET_RCV_ERROR) {      // 连接失败
                        close(TCPSerNo[i]);
                        timedly(50);
```

```
                                    TCP_Abort(TCPSerNo[i]);
                                    TCPSerLinkFlag[i] = 0;
                                    TCPSerNo[i] = 0;
                                    linknum--;
                                } else if(recvnum > 0) {
                                    datatemp[12] = i|0x30;
                                    datatemp[13] = ' ';
                                    datatemp[14] = ' ';
                                    sendnum = send( TCPSerNo[i], datatemp, 15, 0 );
                                    recvtemp[recvnum++] = ' ';
                                    recvtemp[recvnum++] = ' ';
                                    send(TCPSerNo[i], recvtemp, recvnum, 0 );
                                    memset(recvtemp, 0, recvnum);
                                }
                            } else {                                    // 连接断开
                                close(TCPSerNo[i]);                     // 关闭连接
                                timedly(50);
                                TCP_Abort(TCPSerNo[i]);
                                TCPSerLinkFlag[i] = 0;
                                TCPSerNo[i] = 0;
                                linknum--;
                            }
                                OSSemPost(TCP_CALL_Sem);
                            }
                        }
                        timedly(50);
                    }
                }
            }
}

/*********************************************************************************************
** Function name: TASK0
** Descriptions :
** Input          : 无
** Output         : 无
*********************************************************************************************/
void TASK0(void *pdata)
{
    pdata = pdata;

    while (1) {
        ethernet_tcp_ser();
    }
}
```

程序清单 3.2 TCP 客户机程序编写实例

```
#include "main.h"
#define TASK0_PRIO              17                          // 任务的优先级
#define TASK0_ID                TASK0_PRIO                  // 任务的 ID
#define TASK0_STACK_SIZE        1024                        // 定义用户堆栈长度

#define TASK1_PRIO              18                          // 任务的优先级
#define TASK1_ID                TASK1_PRIO                  // 任务的 ID
#define TASK1_STACK_SIZE        512                         // 定义用户堆栈长度

OS_STK    TASK0_STACK[TASK0_STACK_SIZE];
OS_STK    TASK1_STACK[TASK1_STACK_SIZE];

void TASK0(void *pdata);
void TASK1(void *pdata);

/*********************************************************************************************/
uint8     MCU_IP[]          = {192, 168,     0, 253};       // 设置 MCU 的 IP 地址等信息
uint8     MCU_Mark[]        = {255, 255, 255,     0};
uint8     MCU_GateWay[]     = {192, 168,     0, 254};
uint16    MCU_Port          = 1200;
uint8     MCU_MAC[6];

uint8     PC_IP[]           = {192, 168,     0, 47};
uint16    PC_Port           = 2200;

OS_EVENT *TCP_CALL_Sem;
#define TCP_CLINUM                  2                     // T CP 客户端连接个数

/*********************************************************************************************
** Function name: main
** Descriptions : 主函数
** Input        : 无
** Output       : 无
*********************************************************************************************/
int main (void)
{
#if OS_CRITICAL_METHOD == 3                               // Allocate storage for CPU status register
    OS_CPU_SR    cpu_sr;
#endif
```

```
/*----------------------    系统初始化代码    ----------------------------------------------*/
       TargetInit ();                                          //  系统初始化,版本号验证
       PinInit();                                              //  驱动库初始化

      ZlgipInitial(MCU_IP, MCU_GateWay, MCU_Mark);             //  设置系统 IP 参数
       TCP_CALL_Sem = OSSemCreate(1);

/*-------------------------------------------------------------------------------------------*/

      GpioSet(BUZZER);                                         //  鸣叫 BEEP
      OSTimeDly(OS_TICKS_PER_SEC/10);
      GpioClr(BUZZER);
      OSTimeDly(OS_TICKS_PER_SEC/10);
      GpioSet(BUZZER);
      OSTimeDly(OS_TICKS_PER_SEC/10);
      GpioClr(BUZZER);

/*-------------------------------------------------------------------------------------------*/

      OSTaskCreateExt(TASK0,
                      (void *)0,
                      &TASK0_STACK[TASK0_STACK_SIZE-1],
                      TASK0_PRIO,
                      TASK0_ID,
                      &TASK0_STACK[0],
                      TASK0_STACK_SIZE,
                      (void *)0,
                      OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);

      OSTaskCreateExt(TASK1,
                      (void *)0,
                      &TASK1_STACK[TASK1_STACK_SIZE-1],
                      TASK1_PRIO,
                      TASK1_ID,
                      &TASK1_STACK[0],
                      TASK1_STACK_SIZE,
                      (void *)0,
                      OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);

       while (1) {
            OSTimeDly(OS_TICKS_PER_SEC);
      }
}
```

```
void timedly(int nms)
{
   OSTimeDly(nms);
}
void closetcp(uint16 tcpno)
{
        uint8      result;

        result = close(tcpno);
        while (result != tcpno) {                                    // 断开失败
                result = close(tcpno);
        }
        timedly(50);
        TCP_Abort(tcpno);
}

void ethernet_tcp_cli()
{
       uint16    Temp;
       int         ei,i;
       uint16 recvnum;
       uint32    sendnum;
       SOCKET    s[TCP_CLINUM];
       struct    sockaddr        servaddr, cliaddr;

       uint8    TCPLinkFlag[TCP_CLINUM];
       uint16 TCPNo[TCP_CLINUM];

        uint8    datatemp[20],recvtemp[100];
        uint8    err;

        memset(datatemp,0,20);
        memset(recvtemp,0,100);
        for (i=0; i<TCP_CLINUM; i++) {
                TCPLinkFlag[i] = 0;
                TCPNo[i] = 0;
                s[i] = INVALID_SOCKET;
        }

        cliaddr.sin_family = 0;                                      // 设置 MCU(客户端)的 IP 地址和端口
        getlocalip(cliaddr.sin_addr,0);
        cliaddr.sin_port = MCU_Port;

        servaddr.sin_family = 0;                                     // 设置 PC(服务端)的 IP 地址和端口
```

```
servaddr.sin_addr[0] = PC_IP[0];
servaddr.sin_addr[1] = PC_IP[1];
servaddr.sin_addr[2] = PC_IP[2];
servaddr.sin_addr[3] = PC_IP[3];
servaddr.sin_port = PC_Port;

for (i=0; i<TCP_CLINUM; i++) {
        OSSemPend(TCP_CALL_Sem, 0, &err);
        s[i] = socket( AF_INET, SOCK_STREAM, TCP_PROTOCOL);
        if (s[i]!= INVALID_SOCKET) {                                    // 成功
                ei = bind(s[i], (struct sockaddr*)&cliaddr, sizeof(cliaddr));   // 绑定 MCU(客户端)端口
                while (ei == SOCKET_ERROR) {                            // 失败
                        closesocket(s[i]);                              // 关闭 socket
                        s[i] = socket( AF_INET, SOCK_STREAM, TCP_PROTOCOL);
                        ei = bind(s[i], (struct sockaddr*)&cliaddr, sizeof(cliaddr));
                }
        }
        cliaddr.sin_port++;
        if(cliaddr.sin_port > 1300) {
                cliaddr.sin_port = 1200;
        }
        Temp = connect(s[i], (struct sockaddr*)&servaddr, sizeof(servaddr));
        if (Temp != SOCKET_ERROR) {
                TCPLinkFlag[i] = 1;
                TCPNo[i]            = Temp;
        }
        OSSemPost(TCP_CALL_Sem);
        timedly(200);
}

datatemp[0] = 'M';
datatemp[1] = 'i';
datatemp[2] = 'n';
datatemp[3] = 'i';
datatemp[4] = 'A';
datatemp[5] = 'R';
datatemp[6] = 'M';
datatemp[7] = '-';
datatemp[8] = '-';
datatemp[9] = 'T';
datatemp[10] = 'C';
datatemp[11] = 'P';

while (1) {
```

```
for (i=0; i<TCP_CLINUM; i++) {
    if (TCPLinkFlag[i]) {
        OSSemPend(TCP_CALL_Sem, 0, &err);
        if (3 == getsocktcpsta(TCPNo[i])) {                    // 确认网络是否出于连接状态
            datatemp[12] = i|0x30;
            datatemp[13] = ' ';
            datatemp[14] = ' ';
            sendnum = send( TCPNo[i], datatemp, 15, 0);
            if (sendnum == SOCKET_SEN_ERROR) {    // 发送连接失败
                closetcp(TCPNo[i]);
                TCPLinkFlag[i] = 0;
            }
            recvnum = recv(TCPNo[i], recvtemp, 50, 0);
            if ((recvnum>0)&&(recvnum!=SOCKET_SEN_ERROR)) {
                send(TCPNo[i], recvtemp, recvnum, 0);
                memset(recvtemp, 0, recvnum);
            }
        } else {                                               // 连接断开
            closetcp(TCPNo[i]);                                // 关闭连接
            TCPLinkFlag[i]=0;
        }
        OSSemPost(TCP_CALL_Sem);
    } else {                                                   // 重新连接
        cliaddr.sin_port++;
        if(cliaddr.sin_port > 1300) {
            cliaddr.sin_port = 1200;
        }
        OSSemPend(TCP_CALL_Sem, 0, &err);
        ei = bind(s[i], (struct sockaddr*)&cliaddr, sizeof(cliaddr));
        while (ei == SOCKET_ERROR) {                           // 失败
            closesocket(s[i]);                                 // 关闭 socket
            s[i] = socket( AF_INET, SOCK_STREAM, TCP_PROTOCOL);
            ei = bind(s[i], (struct sockaddr*)&cliaddr, sizeof(cliaddr));
        }
        Temp = connect(s[i], (struct sockaddr*)&servaddr, sizeof(servaddr));
        OSSemPost(TCP_CALL_Sem);
        if (Temp != SOCKET_ERROR) {
            TCPLinkFlag[i] = 1;
            TCPNo[i] = Temp;
        }
        timedly(200);
    }
}
timedly(1000);
```

```
    }
}
/*********************************************************************************************
** Function name: TASK0
** Descriptions :
** Input          : 无
** Output         : 无
*********************************************************************************************/
void TASK0(void *pdata)
{
    pdata = pdata;

    while (1) {
        ethernet_tcp_cli();
    }
}
```

3.2 SOCKET API 函数在 UDP 通信中的使用

编写 UDP 通信的任务时也分为服务器方式和客户机方式两种。服务器方式是先接收到数据在进行处理，而客户机则是先发送数据然后等待回应处理，它们所用到的 SOCKET API 函数都是相同的。如图 3.2 就是 UDP 通讯时服务器端和客户机端通讯的函数应用过程图。

程序清单 3.3 是一个服务器任务的例子，用户可以参考该任务中 SOCKET API 函数的使用来编写自己的程序。

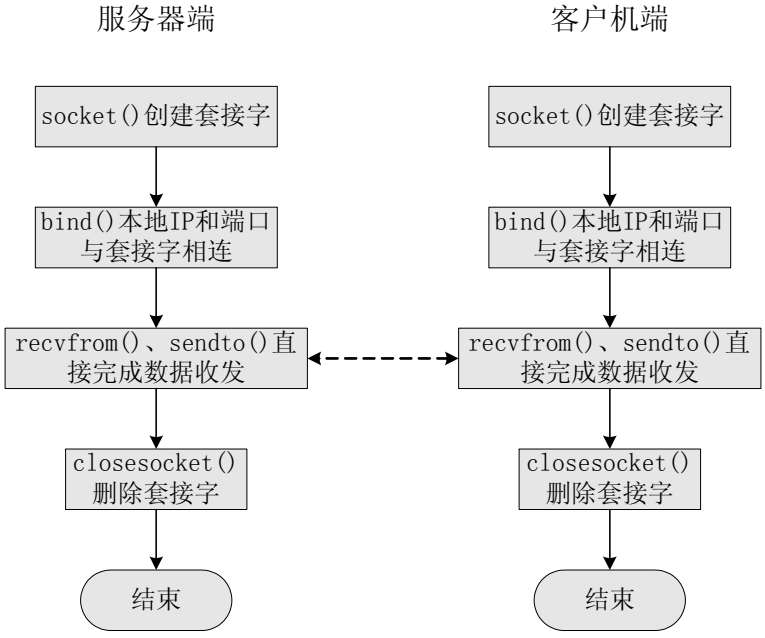

图 3.2 UDP 通讯时 SOCKET API 函数的应用

程序清单 3.3 UDP 服务器程序编写实例

```
#include "main.h"

#define TASK0_PRIO              17                          // 任务的优先级
#define TASK0_ID                TASK0_PRIO                      // 任务的 ID
#define TASK0_STACK_SIZE    1024                        // 定义用户堆栈长度

#define TASK1_PRIO              18                          // 任务的优先级
#define TASK1_ID                TASK1_PRIO                  // 任务的 ID
#define TASK1_STACK_SIZE    512                         // 定义用户堆栈长度

OS_STK    TASK0_STACK[TASK0_STACK_SIZE];
OS_STK    TASK1_STACK[TASK1_STACK_SIZE];

void TASK0(void *pdata);
void TASK1(void *pdata);

/*********************************************************************************************/
uint8    MCU_IP[] = {192, 168,     0, 253};              // 设置 MCU 的 IP 地址等信息
uint8    MCU_Mark[] = {255, 255, 255,     0};
uint8    MCU_GateWay[] = {192, 168,     0, 254};
uint16 UDP_Port  = 3200;

uint8    PC_IP[]  = {192, 168,     0, 47};
uint16 PC_Port   = 2200;

OS_EVENT *TCP_CALL_Sem;

/*********************************************************************************************
** Function name: main
** Descriptions : 主函数
** Input              : 无
** Output             : 无
*********************************************************************************************/
int main (void)
{
#if OS_CRITICAL_METHOD == 3                             // Allocate storage for CPU status register
    OS_CPU_SR    cpu_sr;
#endif

/*---------------------    系统初始化代码    ----------------------------------------------*/
    TargetInit ();                                      // 系统初始化,版本号验证
    PinInit();                                          // 驱动库初始化
```

```
    ZlgipInitial(MCU_IP, MCU_GateWay, MCU_Mark);            // 设置系统 IP 参数
     TCP_CALL_Sem = OSSemCreate(1);

     GpioSet(BUZZER);
     OSTimeDly(OS_TICKS_PER_SEC/10);
     GpioClr(BUZZER);
     OSTimeDly(OS_TICKS_PER_SEC/10);
     GpioSet(BUZZER);
     OSTimeDly(OS_TICKS_PER_SEC/10);
     GpioClr(BUZZER);
/*-------------------------------------------------------------------------------------*/

    OSTaskCreateExt(TASK0,
                    (void *)0,
                    &TASK0_STACK[TASK0_STACK_SIZE-1],
                    TASK0_PRIO,
                    TASK0_ID,
                    &TASK0_STACK[0],
                    TASK0_STACK_SIZE,
                    (void *)0,
                    OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);

    OSTaskCreateExt(TASK1,
                    (void *)0,
                    &TASK1_STACK[TASK1_STACK_SIZE-1],
                    TASK1_PRIO,
                    TASK1_ID,
                    &TASK1_STACK[0],
                    TASK1_STACK_SIZE,
                    (void *)0,
                    OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);

     while (1) {
          OSTimeDly(OS_TICKS_PER_SEC);
    }
}


void timedly(int nms)
{
    OSTimeDly(nms);
}

void ethernet_udp()
```

```
{
    int         ei;
    uint16    recvnum;
    uint16    sendnum;
    SOCKET    s;
    struct    sockaddr servaddr, cliaddr;
    uint16    clilen;
     uint8      err;
     uint8      datatemp[20],recvtemp[100];

     memset(datatemp,0,20);
     memset(recvtemp,0,100);

     datatemp[0] = 'M';
     datatemp[1] = 'i';
     datatemp[2] = 'n';
     datatemp[3] = 'i';
     datatemp[4] = 'A';
     datatemp[5] = 'R';
     datatemp[6] = 'M';
     datatemp[7] = '-';
     datatemp[8] = '-';
     datatemp[9] = 'U';
     datatemp[10] = 'D';
     datatemp[11] = 'P';

     servaddr.sin_family = 0;                                    // 设置 MCU(服务端)的 IP 地址和端口
     getlocalip(servaddr.sin_addr,0);
     servaddr.sin_port = UDP_Port;

     cliaddr.sin_family = 0;                                     // 设置 PC(服务端)的 IP 地址和端口

     OSSemPend(TCP_CALL_Sem, 0, &err);
     s = socket( AF_INET, SOCK_DGRAM, UDP_PROTOCOL);   // 绑定 MCU(客户端)端口
     if (s != INVALID_SOCKET) {                                  // 成功
          ei = bind(s, (struct sockaddr*)&servaddr, sizeof(servaddr));
          while (ei == SOCKET_ERROR) {                           // 失败
               ei = bind(s, (struct sockaddr*)&servaddr, sizeof(servaddr));
          }
     }

     OSSemPost(TCP_CALL_Sem);
     if (ei != SOCKET_ERROR) {
          while (1) {
```

```
                OSSemPend(TCP_CALL_Sem, 0, &err);
                recvnum = recvfrom(s, recvtemp, 50, 0, &cliaddr, &clilen);
                if ((recvnum > 0) && (recvnum < 50)) {
                        sendnum = sendto(s, datatemp, 15, 0, &cliaddr, clilen);
                        recvtemp[recvnum++] = ' ';
                        recvtemp[recvnum++] = ' ';
                        sendnum = sendto(s, recvtemp, recvnum, 0, &cliaddr, clilen);
                }
                OSSemPost(TCP_CALL_Sem);
                timedly(50);
            }
        }
}

/*********************************************************************************************
** Function name: TASK0
** Descriptions :
** Input            : 无
** Output           : 无
*********************************************************************************************/
void TASK0(void *pdata)
{
       pdata = pdata;

      while (1) {
            ethernet_udp();
      }
}
```

注意：因为 UDP 发送不基于可靠的连接，所以在使用 sendto()函数是应该判断返回值是否大于 0，只有在大于 0 的情况下才算发送成功。（以上的例子使用得不算规范）

4. 声明

开发预备知识

MiniARM® M22A 系列产品将提供尽可能全面的开发模板、驱动程序及其应用说明文档以方便用户使用，但 MiniARM® M22A 系列产品不是教学开发平台。对于需要熟悉 ARM7 体系结构，LPC2200 系列微控制器特性及其 ADS 开发环境的用户，建议同时购买我公司 SmartARM2200 或 EasyARM2200 教学开发平台。

LPC2000 系列微控制器

建议用户开发在飞利浦半导体主页（http://www.semiconductors.philips.com）上获取最新勘误表并仔细阅读。广州致远电子有限公司对 LPC2200 系列微控制器无论是已知的还是潜在的设计缺陷不负任何责任。

修改文档的权利

广州致远电子有限公司保留任何时候在不事先声明的情况下对 MiniARM® M22A 系列产品相关文档的修改的权力。

ESD 静电放电保护

MiniARM® M22A 系列产品部分元器件内置 ESD 保护电路，但依然建议用户在设计底板时提供 ESD 保护措施，特别是电源与 I/O 设计，以保证产品的稳定运行。安装 MiniARM® M22A 系列产品时，请先将积累在身体上的静电释放，例如佩戴可靠接地的静电环，触摸接入大地的自来水管等。

公　　司：广州致远电子有限公司 嵌入式系统事业部
地　　址：广州市天河区车陂路黄洲工业区二栋四楼（研发部）
邮　　编：510660
网　　址：www.embedtools.com
销售电话：+86 (020) 2264-4249
技术支持：+86 (020) 2887-2684
传　　真：+86 (020) 3860-1859
E–mail：　miniarm.sales@embedtools.com（销售）
　　miniarm.support@ embedtools.com（技术支持）