AN833

Microchip TCP/IP 协议栈

作者： *Nilesh Rajbharti*
Microchip Technology Inc.

简介

在 Microchip 单片机上实现 TCP/IP（传输控制协议 / 网际协议）不需要任何创新之举。感兴趣的开发人员可以很容易找到许多 Microchip 产品的商业和非商业的 TCP/IP 实现方案。本应用笔记详细说明了 Microchip 公司自己免费提供的 TCP/IP 协议栈。

Microchip TCP/IP 协议栈是一套程序，它服务于标准的、基于 TCP/IP 的应用程序（HTTP 服务器或邮件客户机等），或者使用在定制的、基于 TCP/IP 的应用程序中。为了更好地说明这一点，在本文档的末尾描述了一个完整的 HTTP 服务器应用程序，同时给出了协议栈的源代码。

Microchip TCP/IP 协议栈是按照模块化方式实现的，它所有的服务创建了高度抽象的协议层。潜在用户使用时不需要知道 TCP/IP 规范的所有复杂细节。实际上，只对实现 HTTP 服务器应用程序感兴趣的用户并不需要知晓任何有关 TCP/IP 的具体知识。（关于 HTTP 服务器的具体信息请参见从第 77 页开始的部分。）

本应用笔记并没有深入讨论 TCP/IP 协议。建议对该协议细节感兴趣的用户阅读 RFC（Request For Comment）文档。在本文档的末尾可以找到一部分主要 RFC 编号列表。

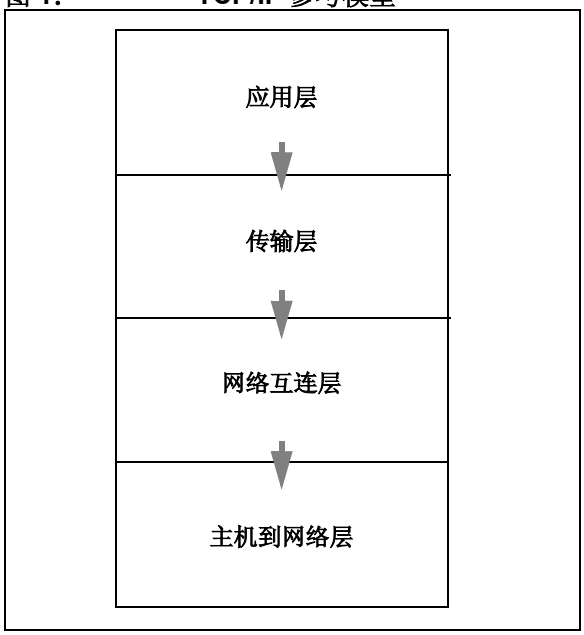
协议栈架构

许多 TCP/IP 的实现方案都遵循了称为“TCP/IP 参考模型”的软件架构。基于此模型的软件被分成多层，一层一层地堆叠（故称为“TCP/IP 协议栈”），并且每层接受来自该层下面的一层或多层的服务。图 1 中显示了 TCP/IP 协议栈模型的一个简化版本。

根据规范，许多 TCP/IP 层都是“活动的”，这意味着不仅在被请求服务时，而且在像超时或新包到达这样的事件发生时，它们都会作出反应。带有大量数据存储器和程序存储器的系统可以十分容易地满足这些要求。多任务操作系统可以提供额外工具，帮助程序实现模块化。但是当系统只使用 8 位单片机以及几百字节的 RAM 和有限的程序存储器时，该任务变得十分困难。此外，如果不能访问一个多任务操作系统，用户必须特别注意要保证协议栈独立于主应用程序。集成在主应用程序中的 TCP/IP 协议栈实现起来相对容易些，并且节省存储空间。但是当集成越来越多的新应用程序时，此专用协议栈可能会产生特殊问题。

此协议栈用 C 语言编写，可使用 Microchip C18 和 Hi-Tech PICC 18 编译器。根据所使用的编译器，源文件会自动进行必要的更改。Microchip TCP/IP 协议栈被设计为只在 Microchip 的 PIC18 系列单片机上运行。此外，目前该协议栈专用于在 Microchip 的 PICDEM.net™ 因特网 / 以太网演示板上运行。但是，可以十分容易地修改使之运行在装有 PIC18 单片机的任何硬件上。

图 1： TCP/IP 参考模型



协议栈层

类似于 TCP/IP 参考模型， Microchip TCP/IP 协议栈将 TCP/IP 协议栈分为多层（图 2）。每层的实现代码驻留在一个独立的源文件中，而服务和应用程序编程接口（Application Programming Interfaces， API）是通过头文件或包含文件定义的。与 TCP/IP 参考模型不同的是，Microchip TCP/IP 协议栈中的许多层可以直接访问不正好在它下面的一层或多层。关于一个层是否绕过相邻模块来获得所需的服务，主要根据开销的大小以及服务是否需要智能处理后才能传递到下一层来决定。

与传统 TCP/IP 协议栈实现方法不同的另外一个地方是添加了两个新模块：“StackTask”和 “ARPTask” 。“StackTask”管理协议栈及其所有模块的操作，而“ARPTask”管理地址解析协议 （Address Resolution Protocol， ARP）层的服务。

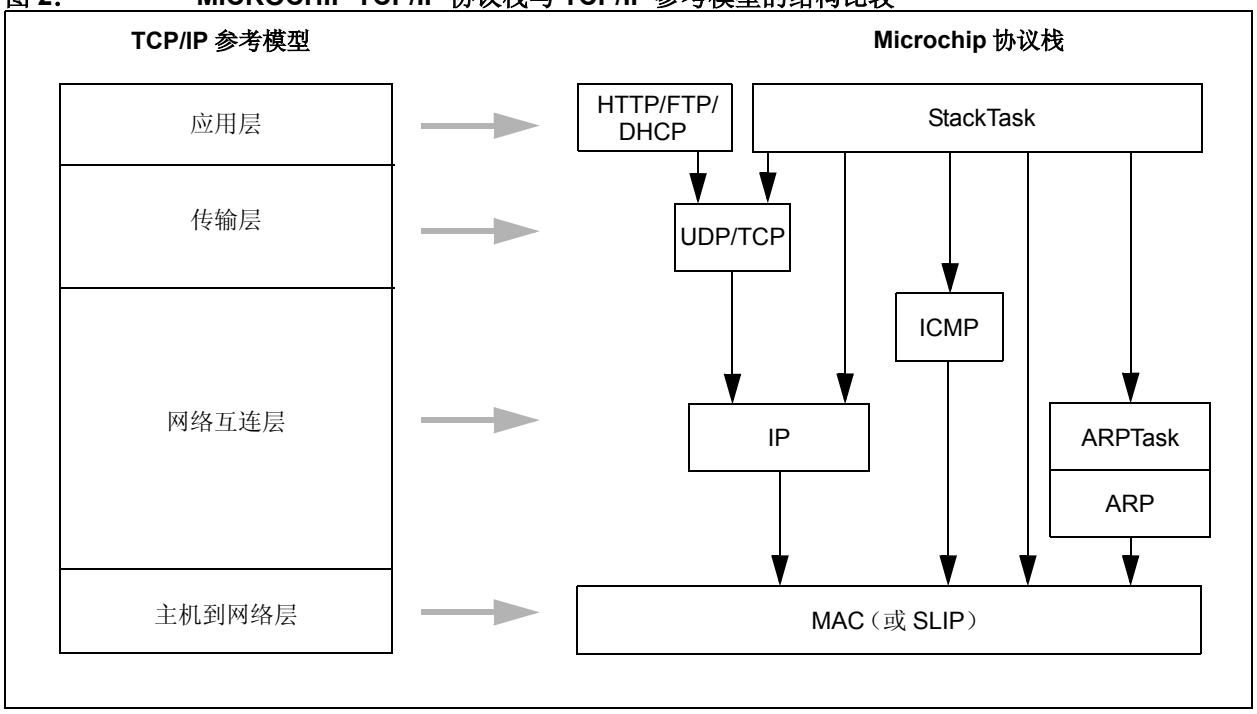
如前面所提到的，TCP/IP 协议栈是“活动的”，它的某些层必须能异步执行某些定时操作。为了能满足这个要求，并与使用其服务的主应用程序保持相对独立，Microchip TCP/IP 协议栈使用了一种众所周知的技术，称为*协同式多任务处理（Cooperative Multitasking）技术*。在协同式多任务处理系统中，同时存在多个任务，每个任务执行自己的作业然后交回控制权，这样下个任务才能够执行作业。就此而论，“StackTask”和“ARPTask”都是协同式任务。

通常，协同式多任务处理（或任何其他类型的多任务处理）是由操作系统或主应用程序自身来实现的。Microchip TCP/IP 协议栈被设计为独立于任何操作系统，可实现它自己的协同式多任务处理系统。因此，它可以被应用在任何系统中，而不论该系统是否为多任务操作系统。但是，使用 Microchip TCP/IP 协议栈的应用程序自身必须使用协同式多任务处理方法。可以通过将其作业分为多个任务，或将主作业组织为有限状态机（Finite State Machine， FSM）并将长作业分为多个较小的作业来实现。本文档中稍后讨论的 HTTP 服务器就遵循了后一种方法，并具体解释了如何实现协同式应用程序。

请注意 Microchip TCP/IP 协议栈并没有实现 TCP/IP 协议栈中通常有的所有模块。尽管不提供，但是在需要时可以将它们作为单独的任务或模块来实现。

Microchip 将在该协议栈的基础上实现其他协议。

图 2： MICROCHIP TCP/IP 协议栈与 TCP/IP 参考模型的结构比较



协议栈配置

协同式多任务处理技术允许用户的主应用程序执行自己的任务而不必同时管理 TCP/IP 协议栈。如前所述，要实现这个功能意味着使用 Microchip TCP/IP 协议栈的所有应用程序也必须用协同式多任务处理方式来编写。除了协同式多任务处理设计以外，用户必须首先了解一些基本的配置细节。

为了简化配置过程，协议栈使用 C 编译器的“定义项（define）”来使能、禁止或设置某个特定参数，用户需要更改一个或多个“定义项”。这些“定义项”中的大多数是在头文件“StackTsk.h”中定义的。在其他文件中定义的“定义项”会显示相应的文件名。一旦修改了这些文件，用户必须重新编译应用程序项目来包含这些更改。表 1 中列出了这些“定义项”。

表 1： 协议栈配置定义

定义项	数值	使用程序	用途
CLOCK_FREQ（compiler.h）	振荡器频率（Hz）	Tick.c	定义系统振荡器频率，以确定“滴答”计数器（Tick Counter）值
TICKS_PER_SECONDS	10 - 255	Tick.c	计算一秒
TICK_PRESCALE_VALUE	2, 4, 8, 16, 32, 64, 128, 256	Tick.c	确定“滴答”计数器值
MPFS_USE_PGRM	N/A	MP 文件系统（MPFS.c）	如果将程序存储器用于 MPFS 存储，则取消该项的注释符号
MPFS_USE_EEPROM	N/A	MPFS.c	如果将外部串行 EEPROM 用于 MPFS 存储，则取消该项的注释符号
MPFS_RESERVE_BLOCK	0 - 255	MPFS.c	启动 MPFS 存储之前要保留的字节数
EEPROM_CONTROL	外部数据 EEPROM 控制代码	MPFS.c	寻址外部数据 EEPROM
STACK_USE_ICMP	N/A	StackTsk.c	如果不需要 ICMP，则对该项加注释符号
STACK_USE_SLIP	N/A	SLIP.c	如果不需要 SLIP，则对该项加注释符号
STACK_USE_IP_GLEANING	N/A	StackTsk.c	如果不需要 IP Gleaning，则对该项加注释符号
STACK_USE_DHCP	N/A	DHCP.c 和 StackTsk.c	如果不需要 DHCP，则对该项加注释符号
STACK_USE_FTP_SERVER	N/A	FTP.c	如果不需要 FTP 服务器，则对该项加注释符号
STACK_USE_TCP	N/A	TCP.c 和 StackTsk.c	如果不需要 TCP 模块，则对该项加注释符号。如果至少有一个高级模块需要 TCP，则自动使能此模块。
STACK_USE_UDP	N/A	UDP.c 和 StackTsk.c	如果不需要 UDP 模块，则对该项加注释符号。如果至少有一个高级模块需要 UDP，则自动使能此模块。
STACK_CLIENT_MODE	N/A	ARP.c 和 TCP.c	将使能与客户机相关的代码
TCP_NO_WAIT_FOR_ACK	N/A	TCP.c	在发送下个包之前 TCP 将等待 ACK
MY_DEFAULT_IP_ADDR_BYTE? MY_DEFAULT_MASK_BYTE? MY_DEFAULT_GATE_BYTE? MY_DEFAULT_MAC_BYTE?	0 - 255	用户应用程序	定义缺省的 IP、 MAC、网关和子网掩码值。 缺省值是： 10.10.5.15（IP 地址） 00:04:163:00:00:00（MAC） 10.10.5.15（网关） 255.255.255.0（子网掩码）

表 1： 协议栈配置定义 （续）

定义项	数值	使用程序	用途
MY_IP_BYTE? MY_MASK_BYTE? MY_GATE_BYTE? MY_MAC_BYTE?	0 - 255	MAC.c、 ARP.c、DHCP.c 和用户应用程序	由应用程序保存 / 定义的实际 IP、MAC、网关和子网掩码值。如果使能了 DHCP，则这些值反映了为当前 DHCP 服务器分配的配置。
MAX_SOCKETS	1 - 253	TCP.c	定义受支持的套接字总数 （受可用 RAM 限制）。进行编译时要对此进行检查，确保为选定的 TCP 应用程序提供了足够的套接字。
MAX_UDP_SOCKETS	1 - 254	UDP.c	定义受支持的套接字总数 （受可用 RAM 限制）。进行编译时要对此进行检查，确保为选定的 UDP 应用程序提供了足够的套接字。
MAC_TX_BUFFER_SIZE	201 - 1500	TCP.c 和 MAC.c	定义单个发送缓冲区大小
MAX_TX_BUFFER_COUNT	1 - 255	MAC.c	定义发送缓冲区的总数。该数目受可用 MAC 缓冲区大小限制。
MAX_HTTP_CONNECTIONS	1 - 255	HTTP.c	定义任意时间允许的 HTTP 连接的最大数目
MPFS_WRITE_PAGE_SIZE （MPFS.h）	1 - 255	MPFS.c	定义当前 MPFS 存储介质的可写页面大小
FTP_USER_NAME_LEN （FTP.h）	1 - 31	FTP.c	定义 FTP 用户名字符串的最大长度
MAX_HTTP_ARGS （HTTP.c）	1 - 31	HTTP.c	定义包括 HTML 表单名称在内的 HTML 表单字段的最大数目
MAX_HTML_CMD_LEN （HTTP.c）	1 - 128	HTTP.c	定义 HTML 表单 URL 字符串的最大长度

使用协议栈

本应用笔记附带的文件包含 Microchip TCP/IP 协议栈、HTTP 和 FTP 服务器以及 DHCP 和 IP Gleaning 模块的完整源代码。还包含了一个使用此协议栈的应用程序示例。

有多个MPLAB®项目文件用来说明协议栈可以采用的所有配置。表 2 中描述了这些项目文件。

由于组成协议栈的每个模块都驻留在各自的文件中，所以用户必须确保在项目中包含了所有所需的文件以进行正确的编译。表 3（下一页）中显示了模块和所需文件的完整列表。

表 2： 协议栈项目文件

项目名称	用途	使用的 “定义项”
HtNICEE.pjt	使用网络接口控制器 （Network Interface Controller， NIC）和外部串行 EEPROM 的 Microchip TCP/IP 协议栈——Hi-Tech 编译器。 使用 IP Gleaning、 DHCP、 FTP 服务器、 ICMP 和 HTTP 服务器。	MPFS_USE_EEPROM, STACK_USE_IP_GLEANING, STACK_USE_DHCP, STACK_USE_FTP_SERVER, STACK_USE_ICMP, STACK_USE_HTTP_SERVER
HtNICPG.pjt	使用 NIC 和内部程序存储器的 Microchip TCP/IP 协议栈——Hi-Tech 编译器。 使用 IP Gleaning、 DHCP、 ICMP 和 HTTP 服务器。	MPFS_USE_PGRM, STACK_USE_IP_GLEANING, STACK_USE_DHCP, STACK_USE_ICMP, STACK_USE_HTTP_SERVER
HtSlEE.pjt	使用 SLIP 和外部串行 EEPROM 的 Microchip TCP/IP 协议栈——Hi-Tech 编译器。 使用 FTP 服务器、 ICMP 和 HTTP 服务器。	STACK_USE_SLIP, MPFS_USE_EEPROM, STACK_USE_FTP_SERVER, STACK_USE_ICMP, STACK_USE_HTTP_SERVER
HtSlPG.pjt	使用 SLIP 和内部程序存储器的 Microchip TCP/IP 协议栈——Hi-Tech 编译器。 使用 ICMP 和 HTTP 服务器。	STACK_USE_SLIP, MPFS_USE_PGRM, STACK_USE_ICMP, STACK_USE_HTTP_SERVER
MPNICEE.pjt	使用 NIC 和外部串行 EEPROM 的 Microchip TCP/IP 协议栈——Microchip C18 编译器。 使用 ICMP 和 HTTP 服务器。	MPFS_USE_EEPROM, STACK_USE_ICMP, STACK_USE_HTTP_SERVER
MPNICPG.pjt	使用 NIC 和内部程序存储器的 Microchip TCP/IP 协议栈——Microchip C18 编译器。 使用 ICMP 和 HTTP 服务器。	MPFS_USE_PGRM, STACK_USE_ICMP, STACK_USE_HTTP_SERVER
MPSlEE.pjt	使用 SLIP 和外部串行 EEPROM 的 Microchip TCP/IP 协议栈——Microchip C18 编译器。 使用 ICMP 和 HTTP 服务器。	STACK_USE_SLIP, MPFS_USE_EEPROM, STACK_USE_ICMP, STACK_USE_HTTP_SERVER
MPSlPG.pjt	使用 SLIP 和内部程序存储器的 Microchip TCP/IP 协议栈——Microchip C18 编译器。 使用 ICMP 和 HTTP 服务器。	STACK_USE_SLIP, MPFS_USE_PGRM, STACK_USE_ICMP, STACK_USE_HTTP_SERVER

表 3： MICROCHIP TCP/IP 协议栈模块和文件要求

模块	所需的文件	用途
MAC	MAC.c Delay.c	介质访问层 （Media Access Layer）
SLIP	SLIP.c	用于 SLIP 的介质访问层
ARP	ARP.c ARPTsk.c MAC.c 或 SLIP.c Helpers.c	地址解析协议 （Address Resolution Protocol）
IP	IP.c MAC.c 或 SLIP.c Helpers.c	网际协议 （Internet Protocol）
ICMP	ICMP.c StackTsk.c IP.c MAC.c 或 SLIP.c Helpers.c	网际控制报文协议 （Internet Control Message Protocol）
TCP	StackTsk.c TCP.c IP.c MAC.c 或 SLIP.c Helpers.c Tick.c	传输控制协议 （Transmission Control Protocol）
UDP	StackTsk.c UDP.c IP.c MAC.c 或 SLIP.c Helpers.c	用户数据报文协议 （User Datagram Protocol）
Stack Manager	StackTsk.c TCP.c IP.c ICMP.c ARPTsk.c ARP.c MAC.c 或 SLIP.c Tick.c Helpers.c	协议栈管理器 （“StackTask”），协调其他 Microchip TCP/IP 协议栈模块
HTTP Server	HTTP.c TCP.c IP.c MAC.c 或 SLIP.c Helpers.c Tick.c MPFS.c XEEPROM.c(1)	超文本传输协议服务器 （HyperText Transfer Protocol Server）
DHCP Client	DHCP.c UDP.c IP.c MAC.c Helpers.c Tick.c	动态主机配置协议 （Dynamic Host Configuration Protocol）

注 1： 只有在 MPFS 存储选项中使能外部串行 EEPROM （MPFS_USE_EEPROM 定义）后才需要。如果选择此选项，则必须包含相应的 MPFS 映像文件。（详情请参见第 84 页的 “MPFS Image Builder”。）

表 3： MICROCHIP TCP/IP 协议栈模块和文件要求 （续）

模块	所需的文件	用途
IP Gleaning	StackTsk.c ARP.c ARPTsk.c ICMP.c MAC.c 或 SLIP.c	只配置节点 IP 地址。
FTP Server	FTP.c TCP.c IP.c MAC.c 或 SLIP.c	文件传输协议服务器 （File Transfer Protocol Server）。

注 1： 只有在 MPFS 存储选项中使能外部串行 EEPROM （MPFS_USE_EEPROM 定义）后才需要。如果选择此选项，则必须包含相应的 MPFS 映像文件。（详情请参见第 84 页的 “MPFS Image Builder”。）

一旦包含了相应的文件设置项目后，就必须修改主应用程序源文件来包含例 1 中所示的编程语句。如需完整的工作示例，请参见 “Websrvr.c” 的源代码。该源文件与其他协议栈文件一起实现了 HTTP 服务器。

要了解协同式多任务应用程序是如何执行的，请参见 HTTP.c 的源代码（HTTP 服务器任务）。如果不使用协同式多任务工作模式，使用协议栈的应用程序需要将它们的作业划分为多个较小的任务，每个任务都被限制为不能独占 CPU 太长的时间。“HTTP.c” 的代码说明了状态机方法，即将长应用程序分为较小的状态机状态，在作业等待特定事件期间需将控制权归还给主应用程序。

例 1：　　补充到主应用程序的代码

```
// 声明该文件为主应用程序
#define THIS_IS_STACK_APPLICATION

#include “StackTsk.h”
#include “Tick.h”
#include “dhcp.h” // 只有在使用 DHCP 时
#include “http.h” // 只有在使用 HTTP 时
#include “ftp.h”  // 只有在使用 FTP 时
// 其他应用程序特定的包含文件
...

// 主程序入口
void main(void)
{
        // 执行应用程序的初始化
        ...

        // 初始化协议栈各组成部分
        // 如果使用 StackApplication，同样要对它进行初始化
        TickInit();
        StackInit();
        HTTPInit(); // 只有在使用 HTTP 时
        FTPInit();  // 只有在使用 FTP 时

        // 进入无限程序循环
        while(1)
        {
                // 更新 “滴答”计数器。可通过中断完成
                TickUpdate();

                // 协议栈管理器执行其任务
                StackTask();

                // 协议栈应用程序执行其任务
                HTTPServer();   // 只有在使用 HTTP 时
                FTPServer();    // 只有在使用 FTP 时

                // 应用程序逻辑驻留于此
                DoAppSpecificTask();
        }
}
```

协议栈模块和 API

Microchip TCP/IP 协议栈由许多模块组成。在使用模块之前，用户必须查阅并理解它的用途和 API。以下各节中提供每个模块的概述以及对其 API 的说明。

介质访问控制层 （MAC）

本应用笔记中的 Microchip TCP/IP 协议栈版本是特别针对 Realtek RTL8019AS NIC 而编写的。RTL8019AS 是与 NE2000 兼容的 NIC，实现了以太网物理 （Ethernet Physical，PHY）层和 MAC 层。如果使用不同的 NIC，则用户需要修改或创建新的 MAC.c 文件来实现访问。只要 MAC.c 提供的服务没有更改，所有其他模块将保持不变。

协议栈使用 NIC 上提供的片上 SRAM 作为保持缓冲区，直到更高级别的模块读取它为止。该 SRAM 缓冲区还用于执行必需的 IP 校验和计算。除了接收 FIFO 缓冲区由 NIC 自己管理外，MAC 层管理它自己的发送队列。使用这个队列，呼叫方可以发送一条消息并请求 MAC 来保留它，这样在需要时就可以再次发送。用户可以使用 C “定义项”来指定发送缓冲区、发送队列和接收队列的大小 （详情请参见表 1 “协议栈配置定义”）。

串行线路网际协议 （SLIP）

SLIP 层使用串行电缆而不是以太网电缆作为主要通讯介质。SLIP 不需要 NIC，因此只能提供非常简单和廉价的 IP 连接服务。 SLIP 通常是一对一连接，其中一台主机充当客户机。 SLIP 模块被设计为工作在基于 Windows® 的计算机上，但是只需经过非常少的更改就可以运行在其他操作系统上。由于协议栈的模块化设计，使用时只需要链接 SLIP 模块 （SLIP.c）即可。SLIP 模块提供的 API 与 MAC 使用的 API 基本上是一样的 （详情请参见以下几页中对 MAC 的 API 描述）。

SLIP 使用中断驱动的串行数据发送，与 NIC MAC 使用的轮询方法不同。主应用程序必须声明一个中断处理程序，并调用 SLIP 中断处理程序 MACISR。有关更多详细信息，请参见可下载的 Zip 文件中包含的 Web 服务器示例程序，即 “Websrvr.c”的源代码。

为了使用 SLIP 模块连接到 Microchip TCP/IP 协议栈，必须将主机系统配置为使用 SLIP 连接。请参见第 87 页的 “演示应用程序”以获取详细信息。

MACInit

此函数初始化 MAC 层。它初始化内部缓冲区并将 NIC 复位到一个已知状态。

语法

```
void MACInit()
```

参数

无

返回值

无

前提条件

无

副作用

所有等待的发送和接收数据都被丢弃。

示例

```
// 初始化 MAC 模块
MACInit();
```

MACIsTxReady

此函数指示是否至少有一个 MAC 发送缓冲区为空。

语法

```
BOOL MACIsTxReady()
```

参数

无

返回值

TRUE： 如果至少有一个 MAC 发送缓冲区为空。

FALSE：如果所有 MAC 发送缓冲区都是满的。

前提条件

无

副作用

无

备注

无

示例

```
// 检查 MAC 发送是否就绪 ...
if ( MACIsTxReady() )
{
       // 如果发送缓冲区为空，则发送一条消息
...
```

MACGetHeader

此函数检查 MAC 接收缓冲区，如果找到数据包，则返回远程主机和数据包的信息。

语法

```
BOOL MACGetHeader(MAC_ADDR *remote, BYTE *type)
```

参数

Remote [out]

远程 MAC 地址

type [out]

数据包类型

此参数的值可能为：

值	含义
MAC_IP	接收到 IP 数据包
MAC_ARP	接收到 ARP 数据包
MAC_UNKNOWN	接收到未知或不支持的数据包

返回值

TRUE： 如果接收到数据包并且该数据包是有效的。填充所有参数。

FALSE： 如果没有接收到数据包或发现数据包是无效的。

前提条件

无

副作用

无

备注

一旦通过调用 MACGetHeader 获取了一个数据包，就必须获取（使用 MACGet）并丢弃（使用 MACDiscardRx）整个数据包。用户不能多次调用 MACGetHeader 来接收多个包并获取数据。

示例

```
// 获取可能的数据包信息
if ( MACGetHeader(&RemoteNodeMAC, &PacketType) )
{
        // 接收到一个包后对其进行处理
        ...

        // 处理完成后即丢弃它
       MACDiscardRx();
...
```

MACGet

此函数返回来自活动的发送或接收缓冲区的下个字节。

语法

```
BYTE MACGet()
```

参数

无

返回值

数据字节

前提条件

必须已经调用 MACGetHeader、 MACPutHeader、 MACSetRxBuffer 或 MACSetTxBuffer。

副作用

无

备注

活动的缓冲区 （无论是发送还是接收）由先前调用的是 MACGetHeader、 MACPutHeader、 MACSetRxBuffer 还是 MACSetTxBuffer 函数确定。例如，如果先调用 MACGetHeader 然后调用 MACGet，则接收缓冲区变成活动的缓冲区。但是如果先调用 MACPutHeader 然后调用 MACGet，则发送缓冲区变成活动的缓冲区。

示例 1

```
// 获取可能的数据包信息
if ( MACGetHeader(&RemoteNode, &PacketType) )
{
        // 接收到一个包后对其进行处理
        data = MACGet();
        ...

        // 处理完成后即丢弃它
        MACDiscardRx();
...
```

示例 2

```
// 装入待发送的包
if ( MACIsTxReady() )
{
        // 装入 MAC 头
        MACPutHeader(&RemoteNode, MAC_ARP);

        // 获取活动的发送缓冲区
        Buffer = MACGetTxBuffer();

        // 装入所有数据
        ...

        // 希望计算整个包的校验和
        // 将发送缓冲区存取指针指向包的开头位置
        MACSetTxBuffer(buffer, 0);

        // 读取发送缓冲区内容来计算校验和
        checksum += MACGet();
        ...
...
```

MACGetArray

此函数从活动的发送或接收缓冲区获取字节数组。

语法

```
WORD MACGetArray(BYTE *val, WORD len)
```

参数

val [out]

指向字节数组的指针

len [in]

要获取的字节数

返回值

已获取的字节总数。

前提条件

必须已经调用 MACGetHeader、 MACPutHeader、 MACSetRxBuffer 或 MACSetTxBuffer。

副作用

无

备注

调用者必须确保欲读取的数据字节总数不会超过当前缓冲区中可用的数据总数。此函数不检查缓冲区下溢条件。

示例

```
// 获取可能的数据包信息
if ( MACGetHeader(&RemoteNode, &PacketType) )
{
        // 接收到一个包后对其进行处理
        actualCount = MACGetArray(data, count);
...
```

MACDiscardRx

此函数丢弃活动的接收缓冲区中的数据，并将该缓冲区标记为可用。

语法

```
void MACDiscardRx()
```

参数

无

返回值

无

前提条件

必须已经调用 MACGetHeader 并返回 TRUE 值。

副作用

无

备注

调用者必须确保在调用此函数之前至少已经接收了一个数据包。应该使用 MACGetHeader 来确定是否应丢弃接收缓冲区。

示例

```
// 获取可能的数据包信息
if ( MACGetHeader(&RemoteNode, &PacketType) )
{
        // 接收到一个包后对其进行处理
        actualCount = MACGetArray(data, count);
        ...

        // 处理完成后即丢弃它
        MACDiscardRx();
        ...
```

MACPutHeader

此函数组装 MAC 包头并将其装入活动的发送缓冲区中。

语法

```
void MACPutHeader(MAC_ADDR *remote, BYTE type, WORD dataLen)
```

参数

remote [in]

远程节点 MAC 地址

type [in]

待发送数据包类型

此参数的值可能为：

值	含义
MAC_IP	要发送的是 IP 数据包
MAC_ARP	要发送的是 ARP 数据包

data [in]

此包的字节数（包括 IP 包头在内）

返回值

无

前提条件

必须已经调用 MACIsTxReady 并返回 TRUE 值。

副作用

无

备注

此函数仅将 MAC 包头装入活动的发送缓冲区中。调用者仍然需要装入更多数据和 / 或清空缓冲区用于发送。调用 MACFlush 来启动对当前包的发送。

示例

```
// 检查是否至少有一个发送缓冲区为空
if ( MACIsTxReady() )
{
        // 组装 IP 包，整个 IP 包大小为 100 字节
        // 包含 IP 包头
        MACPutHeader(&RemoteNodeMAC, MAC_IP, 100);
        ...
```

MACPut

此函数将给定数据字节装入活动的发送缓冲区或接收缓冲区中。

语法

```
void MACPut(BYTE val)
```

参数

val [in]

要写入的数据字节

返回值

无

前提条件

必须已经调用 MACGetHeader、 MACPutHeader、 MACSetRxBuffer 或 MACSetTxBuffer。

副作用

无

备注

此函数可用于修改当前活动的发送缓冲区或接收缓冲区。

示例

```
// 检查是否至少有一个发送缓冲区为空
if ( MACIsTxReady() )
{
    // 组装 IP 包，整个 IP 包大小为 100 字节
    // 包含 IP 包头
    MACPutHeader(&RemoteNodeMAC, MAC_IP, 100);

    // 现在装入实际的 IP 数据字节
    MACPut(0x55);
    ...
```

MACPutArray

此函数将数据字节数组写入活动的发送缓冲区或接收缓冲区中。

语法

```
void MACPutArray(BYTE *val, WORD len)
```

参数

val [in]

要写入的数据字节

len [in]

要写入的字节总数

返回值

无

前提条件

必须已经调用 MACGetHeader、 MACPutHeader、 MACSetTxBuffer 或 MACSetRxBuffer。

副作用

无

备注

无

示例

```
// 检查是否至少有一个发送缓冲区为空
if ( MACIsTxReady() )
{
    // 组装 IP 包，整个 IP 包大小为 100 字节
    // 包含 IP 包头
    MACPutHeader(&RemoteNodeMAC, MAC_IP, 100);

    // 现在装入实际的 IP 数据字节
    MACPut(0x55);
    MACPutArray(data, count);
    ...
```

MACFlush

此函数将活动的发送缓冲区标记为已准备好发送。

语法

```
void MACFlush()
```

参数

无

返回值

无

前提条件

必须已经调用 MACPutHeader 或 MACSetTxBuffer。

副作用

无

备注

无

示例

```
// 检查是否至少有一个发送缓冲区为空
if ( MACIsTxReady() )
{
    // 组装 IP 包，整个 IP 包大小为 100 字节
    // 包含 IP 包头
    MACPutHeader(&RemoteNodeMAC, MAC_IP, 100);

    // 装入实际的 IP 数据字节
    MACPut(0x55);
    MACPutArray(data, count);
    ...

    // 现在进行发送
    MACFlush();
...
```

MACDiscardTx

此函数丢弃指定发送缓冲区内容并将其标记为可用。

语法

```
void MACDiscardTx(BUFFER buffer)
```

参数

buffer [in]

要丢弃的缓冲区

返回值

无

前提条件

无

副作用

无

备注

必须使用通过调用 MACGetTxBuffer 而收到的发送缓冲区标识符。

示例

```
// 检查是否至少有一个发送缓冲区为空
if ( MACIsTxReady() )
{
        // 组装 IP 包，整个 IP 包大小为 100 字节
        // 包含 IP 包头
        MACPutHeader(&RemoteNodeMAC, MAC_IP, 100);

        // 获取当前的发送缓冲区
        buffer = MACGetTxBuffer();

        // 保留它
        MACReserveTxBuffer (Buffer);

        // 装入实际的 IP 数据字节
        ...

        // 现在进行发送
        MACFlush();

        // 不再需要该缓冲区
        MACDiscardTx(buffer);

        ...
```

MACSetRxBuffer

此函数设置活动的接收缓冲区的存取地址。

语法

```
void MACSetRxBuffer(WORD offset)
```

参数

offset [in]

下次存取发生的地址 （相对于缓冲区起始位置）

返回值

无

前提条件

无

副作用

无

备注

用户必须确保所提供的 ***offset*** 不会超出当前接收缓冲区内的数据范围。如果 ***offset*** 超出当前接收缓冲区的有效范围，所有之后对该缓冲区的存取将产生无效数据。

示例

```
// 获取可能的数据包信息
if ( MACGetHeader(&RemoteNode, &PacketType) )
{
        // 接收到一个包后对其进行处理
        actualCount = MACGetArray(data, count);
        ...

        // 从相对于缓冲区开头偏移量为 20 的地址处取数据
        MACSetRxBuffer(20);
        data = MACGet();
...
```

MACSetTxBuffer

此函数设置指定发送缓冲区的存取地址，并将发送缓冲区标记为活动的。

语法

```
void MACSetTxBuffer(BUFFER buffer, WORD offset)
```

参数

buffer [in]

要应用此存取偏移量的发送缓冲区

offset [in]

下次存取发生的位置 （相对于缓冲区开头的位置）

返回值

无

前提条件

无

副作用

无

备注

用户必须确保所提供的 ***offset*** 不会超出当前发送缓冲区内的数据范围。如果 ***offset*** 超出当前发送缓冲区的有效范围，所有之后对该缓冲区的存取将产生无效数据。

示例

```
// 检查是否至少有一个发送缓冲区为空
if ( MACIsTxReady() )
{
        // 组装 IP 包，整个 IP 包大小为 100 字节
        // 包含 IP 包头
        MACPutHeader(&RemoteNodeMAC, MAC_IP, 100);

        // 获取当前的发送缓冲区
        buffer = MACGetTxBuffer();

        // 装入实际的 IP 数据字节
        ...

        // 计算正在发送的数据包的校验和
        ...

        // 现在更新该包内的校验和
        // 若要更新校验和，将发送缓冲区的访问地址设置到校验和的位置
        MACSetTxBuffer(buffer, checksumLocation);
        ...

        // 现在进行发送
        MACFlush();

...
```

MACReserveTxBuffer

此函数保留指定的发送缓冲区，并将其标记为不可用。此函数对于 TCP 层十分有用，在这里消息将会排队直到被远程主机正确应答为止。

语法

```
void MACReserveTxBuffer(BUFFER buffer)
```

参数

buffer [in]

要保留的发送缓冲区

该值必须是由 MACGetTxBuffer 函数返回的有效发送缓冲区标识符

返回值

无

前提条件

无

副作用

无

备注

无

示例

```
// 检查是否至少有一个发送缓冲区为空
if ( MACIsTxReady() )
{
        // 发送 IP 包，整个 IP 包大小为 100 字节
        // 包含 IP 包头
        MACPutHeader(&RemoteNodeMAC, MAC_IP, 100);

        // 获取当前的发送缓冲区
        buffer = MACGetTxBuffer();

        // 保留它，直到接收到 ACK 后丢弃
        MACReserveTxBuffer(buffer);

        // 装入实际的 IP 数据字节
        ...

        // 计算正在发送的数据包的校验和
        ...

        // 现在更新该包内的校验和
        // 若要更新校验和，将发送缓冲区的访问地址设置到校验和的位置
        MACSetTxBuffer(buffer, checksumLocation);
        ...

        // 现在进行发送
        MACFlush();

        ...
```

MACGetFreeRxSize

此函数返回未来可用的接收缓冲区总的大小。

语法

```
WORD MACGetFreeRxSize()
```

参数

无

返回值

未来数据包可用的总字节数。

前提条件

无

副作用

无

备注

无

示例

```
// 获取可用的接收缓冲区大小
freeRxBuffer = MACGetFreeRxSize();
```

MACGetRxBuffer

此宏返回当前接收缓冲区标识符。

语法

```
BUFFER MACGetRxBuffer()
```

参数

无

返回值

活动的接收缓冲区标识符。

前提条件

无

副作用

无

备注

无

示例

```
// 获取可能的数据包信息
if ( MACGetHeader(&RemoteNode, &PacketType) )
{
        // 获取活动的接收缓冲区标识符
        buffer = MACGetRxBuffer();

        // 直接读取校验和，无需任何其他数据
        MACSetRxBuffer(checkLocation);
        checksum = MACGet();

...
```

MACGetTxBuffer

此宏返回当前发送缓冲区标识符。

语法

```
BUFFER MACGetTxBuffer()
```

参数

无

返回值

活动的发送缓冲区标识符。

前提条件

无

副作用

无

备注

无

示例

```
// 如果有空间，装入新消息
if ( MACIsTxReady() )
{
        // 装入 MAC 包头
        MACPutHeader(&RemoteNode, MAC_ARP, 100);

        // 获取活动的发送缓冲区标识符
        buffer = MACGetTxBuffer();

        // 修改发送缓冲区中的数据字节 #20
        MACSetTxBuffer(buffer, 20);
        MACPut(0x55);
...
```

地址解析协议 （ARP 和 ARPTask）

Microchip TCP/IP 协议栈的 ARP 层实际上由两个模块来实现：ARP 和 ARPTask。ARP（由文件“ARP.c”实现）创建 ARP 原始函数。ARPTask（由文件“ARPTsk.c”实现）使用这些原始函数并提供完整的 ARP 服务。

ARPTask 是以协同式状态机实现的，对来自远程主机的 ARP 请求作出响应。它还保留一个单级缓冲区来保存 ARP 回复，以便在相应的函数调用时，能将其传递到更高协议层。ARPTask 没有实现重试机制，因此上级模块或应用程序必须要检测超时条件并作出相应的响应。

ARPTask 以两种模式运行：*服务器模式*和*服务器 / 客户机模式*。在服务器 / 客户机模式下，部分代码被使能并编译，由本地主机自身生成 ARP 请求。在服务器模式下，不编译 ARP 请求代码。通常，如果协议栈与服务器应用程序（例如 HTTP 服务器或 FTP 服务器）一起使用，则必须在服务器模式下编译 ARPTask 来减少代码长度。

编译器“定义项”STACK_CLIENT_MODE 包含代码的客户机部分。在服务器 / 客户机模式下，ARPTask 保留一个单级缓冲区来存储来自远程主机的 ARP 回复。如果不使能服务器 / 客户机模式，则不需要定义缓冲区并且不使用对应的 RAM 和程序存储器。

ARP 函数

ARPIsTxReady

这是一个代替调用 MACIsTxReady 的宏。

语法

```
BOOL ARPIsTxReady()
```

参数

无

返回值

TRUE： 如果至少有一个发送缓冲区为空。

FALSE：如果没有空的发送缓冲区。

前提条件

无

副作用

无

备注

此宏只是一个抽象函数。使用 ARP 服务的上层应该调用此宏而不是直接调用 MACIsTxReady。

示例

```
// 如果 ARP 发送缓冲区准备就绪，则发送 ARP 包
if ( ARPIsTxReady() )
{
        // 发送 ARP 包
        ARPPut(&RemoteNode, ARP_REPLY);

...
```

ARPGet

此函数用于获取完整的 ARP 包，并返回必需的信息。

语法

```
BOOL ARPGet(NODE_INFO *remote, BYTE *opCode)
```

参数

remote [out]

远程节点信息，例如 MAC 和 IP 地址

opCode [out]

ARP 代码

此参数的值可能为：

值	含义
ARP_REPLY	接收到 “ARP 回复”包
ARP_REQUEST	接收到 “ARP 请求”包
ARP_UNKNOWN	接收到未知的 ARP 包

返回值

TRUE： 如果获取的是发到本地主机的有效 ARP 包； remote 和 opCode 包含有效值。

FALSE： 获取了未知的 ARP 代码或此包不是发到本地主机的。

前提条件

MACGetHeader 已被调用，并且

Received MAC packet type == MAC_ARP

副作用

无

备注

此函数假设活动的接收缓冲区包含 1 个 ARP 包，并且此缓冲区的存取指针指向 ARP 包的开头。通常较高级别的层将检查 MAC 缓冲区，并且仅在检测到 ARP 包时调用此函数。此函数获取完整的 ARP 包并释放活动的接收缓冲区。

示例

```
// 如果接收到 MAC 包 ...
if ( MACGetHeader(&RemoteNode, &PacketType) )
{
   // 如果这是一个 ARP 包，则取出它
   If ( PacketType == MAC_ARP )
   {
      // 这是一个 ARP 包
      ARPGet(&RemoteNode, &ARPCode);

...
```

ARPPut

此函数将有效的 ARP 包装入 MAC 缓冲区中。

语法

```
void ARPPut(NODE_INFO *remote, BYTE opCode)
```

参数

remote [in]

远程节点信息，例如 MAC 和 IP 地址

opCode [in]

ARP 代码

此参数的值可能为：

值	含义
ARP_REPLY	将此包作为 “ARP 回复”发送
ARP_REQUEST	将此包作为 “ARP 请求”发送

返回值

无

前提条件

```
ARPIsTxReady == TRUE
```

副作用

无

备注

此函数组装完整的 ARP 包并发送它。

示例

```
// 检查发送缓冲区是否可用
if ( ARPIsTxReady() )
{
        // 进行发送
        ARPPut(&RemoteNode, ARP_REQUEST);

...
```

ARPTask 函数

ARPInit

此函数初始化 ARPTask 状态机，并让它准备好处理 ARP 请求和回复。

语法

```
void ARPInit()
```

参数

无

返回值

无

前提条件

无

副作用

无

备注

在服务器 / 客户机模式下，此函数还初始化内部的单级缓冲区。

示例

```
// 初始化 ARPTask
ARPInit();
...
```

ARPResolve

此函数将 ARP 请求发送到远程主机。

语法

```
void ARPResolve(IP_ADDR *IPAddr)
```

参数

IPAddr [in]

要解析的远程主机的 IP 地址

返回值

无

前提条件

ARPIsTxReady == TRUE

副作用

无

备注

此函数只有在定义了 STACK_CLIENT_MODE 时才可用。

示例

```
// 检查发送缓冲区是否可用
if ( ARPIsTxReady() )
{
        // 发送 ARP 请求
        ARPResolve(&RemoteNodeIP);

...
```

ARPIsResolved

此函数检查内部缓冲区并返回匹配的主机地址信息。

语法

```
BOOL ARPIsResolved(IP_ADDR *IPAddr, MAC_ADDR *MACAddr)
```

参数

IPAddr [in]

要解析的远程主机的 IP 地址

MACAddr [out]

用于保存要解析的远程主机的 MAC 地址的缓冲区

返回值

TRUE： 如果在内部缓冲区中找到匹配的 IP 地址，则将相应的 MAC 地址复制到 MACAddr 中。

FALSE： 如果在内部缓冲区中没有匹配的 IP 地址；则不填充 MACAddr。

前提条件

无

副作用

与内部缓冲区匹配的条目将被从缓冲区中删除，并将其声明为已解析的。

备注

由于 ARPTask 只保留一级缓冲区，所以更高级别的层必须确保先前的请求被解析后才能发送下一个 ARP 请求。此函数只有在定义了 STACK_CLIENT_MODE 时才可用。

示例

```
// 检查发送缓冲区是否可用
if ( ARPIsResolved(&RemoteIPAddr, &RemoteMACAddr) )
{
        // ARP 已被解析。继续 ...
        ...
}
else
{
        // 未被解析。等待 ...
        ...
}
```

网际协议 （IP）

Microchip TCP/IP 协议栈的 IP 层由文件“IP.c”实现。头文件 “IP.h”定义该层提供的服务。

在这个架构中， IP 层是被动的；它不会响应任何 IP 数据包。相反，更高级别的层会使用 IP 原始函数并获取 IP 包，解析它并采取相应的操作。

IPIsTxReady

这是一个代替调用 MACIsTxReady 的宏。

语法

```
BOOL IPIsTxReady()
```

参数

无

返回值

TRUE： 如果至少有一个发送缓冲区为空。

FALSE：如果没有空的发送缓冲区。

前提条件

无

副作用

无

备注

此宏只是一个抽象函数。使用 IP 服务的上层应该调用此宏，而不是直接调用 MACIsTxReady。

示例

```
// 如果 IP 发送缓冲区准备就绪，则发送 IP 包
if ( IPIsTxReady() )
{
        // 组装 IP 包
        IPPutHeader(&Remote, MAC_TCP, IPPacketLen);

...
```

IP规范要求本地主机为它发送的每个包生成唯一的包标识符。这个标识符使得远程主机可以识别重复的包并丢弃它们。 Microchip TCP/IP 协议栈的 IP 层有一个专用 16 位变量来跟踪包标识符。

IPSetTxBuffer

这是一个宏，允许更高级别的层设置发送缓冲区的存取指针。在调用MACSetTxBuffer之前它使用的是IP包头的数据。

语法

```
void IPSetTxBuffer(BUFFER buffer, WORD offset)
```

参数

buffer [in]

被设置存取指针的发送缓冲区的标识符

offset [in]

与 IP 数据有关的偏移量

返回值

无

前提条件

无

副作用

无

备注

使用 IP 服务的各层必须调用此宏来为指定发送缓冲区设置存取指针。此宏根据 IP 包头的长度调整偏移量数值。

示例

```
// 如果 IP 发送缓冲区准备就绪，则发送 IP 包
if ( IPIsTxReady() )
{
        // 组装 IP 包
        IPPutHeader(&Remote, MAC_TCP, IPPacketLen);

        // 获取当前的发送缓冲区标识符
        buffer = MACGetTxBuffer();

        // 装入发送数据
        ...

        // 计算校验和 checkHi:checkLo
        ...

        // 更新校验和
        IPSetTxBuffer(buffer, checkLocation);
        MACPut (checkHi);
        MACPut (checkLo);
        ...
```

IPPutHeader

此函数用于组装一个有效的 IP 包头，并将其装入活动的发送缓冲区中。

语法

```
WORD IPPutHeader(NODE_INFO *remote, BYTE protocol, WORD len)
```

参数

remote [in]

远程节点信息，例如 MAC 和 IP 地址

protocol [in]

此数据包所用的协议

此参数的值可能为：

值	含义
IP_PROT_ICMP	将此包组装为 ICMP
IP_PROT_TCP	将此包组装为 TCP 段
IP_PROT_UDP	将此包组装为 UDP 段

len [in]

IP 数据字节的总长度， IP 包头除外

返回值

无

前提条件

```
IPIsTxReady == TRUE
```

副作用

无

备注

此函数使用头校验和以及正确的网络字节顺序组装 IP 包。在此函数之后，活动的发送缓冲区存取指针指向 IP 数据区域的开头。

此函数不会启动 IP 包的发送。调用者必须通过调用 MACPut 函数装入 IP 数据，和 / 或调用 MACFlush 将缓冲区标记为准备好发送。

示例

```
// 检查发送缓冲区是否可用
if ( IPIsTxReady() )
{
        // 装入头
        IPPutHeader(&RemoteNode, IP_PROT_ICMP, ICMP_HEADER_SIZE+dataLen);

        // 装入 ICMP 数据
        IPPutArray(ICMPData, dataLen);

        // 将其标记为准备好进行发送
        MACFlush();
...
```

IPPutArray

此宏将字节数组装入活动的发送缓冲区中。

语法

```
void IPPutArray(BYTE *buffer, WORD len)
```

参数

buffer [in]

要装入的数据数组

len [in]

数据数组中数据项的总数

返回值

无

前提条件

```
IPIsTxReady == TRUE
```

副作用

无

备注

此宏调用 MACPutArray 函数。它只是一个抽象函数。

示例

```
// 检查发送缓冲区是否可用
if ( IPIsTxReady() )
{
        // 装入头
        IPPutHeader(&RemoteNode, IP_PROT_ICMP, ICMP_HEADER_SIZE+dataLen);

        // 装入 ICMP 数据
        IPPutArray(ICMPData, dataLen);

        // 将其标记为准备好进行发送
        MACFlush();
...
```

IPGetHeader

此函数从活动的发送缓冲区中获取 IP 包头并验证它。

语法

```
BOOL IPGetHeader(IP_ADDR *localIP, NODE_INFO *remote, BYTE *protocol, WORD *len)
```

参数

localIP [out]

本地节点信息，例如 MAC 和 IP 地址

remote [out]

远程节点信息，例如 MAC 和 IP 地址

protocol [out]

与此 IP 包关联的协议

此参数的值可能为：

值	含义
IP_PROT_ICMP	这是一个 ICMP 包
IP_PROT_TCP	这是一个 TCP 包
IP_PROT_UDP	这是一个 UDP 包
所有其他	未知协议

len [out]

此包中 IP 数据的总长度

返回值

TRUE： 接收到有效的 IP 包。会填充远程 IP 地址、包协议和包长度参数。

FALSE：接收到无效的 IP 包。不填充参数。

前提条件

```
MACGetHeader == TRUE
```

副作用

无

备注

此函数假设活动的接收缓冲区存取指针定位在 MAC 数据区域的开头。为了满足这个条件，更高级别的层在调用此函数之前必须执行以下检查：

```
If MACGetHeader == TRUE and PacketType == MAC_IP, call IPGetHeader
Else do not call IPGetHeader
```

一旦处理了 IP 包并且不再需要它后，调用者必须通过调用 MACDiscardRx 函数从 MAC 缓冲区清除它。请参见 Stack Task Manager（StackTsk.c）的源代码以获取详细信息。

IPGetHeader（续）

示例

```
// 检查是否有包准备就绪
if ( MACGetHeader(&RemoteMACAddr, &PacketType) )
{
        // 检查它使用的是何种协议
        if ( PacketType == MAC_IP )
        {
                // 这是 IP 包。取出它
                IPGetHeader(&Local, &Remote, &IPProtocol, &IPLen);

                // 处理该 IP 包
                ...

                // 处理完后即丢弃该包
                MACDiscardRx();
        }
        else
        {
                // 这不是 IP 包。处理它
...
```

IPGetArray

此宏从活动的发送缓冲区或接收缓冲区获取字节数组。

语法

```
WORD IPGetArray(BYTE *val, WORD len)
```

参数

val [out]

指向缓冲区中的字节数组的指针

len [in]

要获取的字节数

返回值

已获取的字节总数

前提条件

必须已经调用 IPGetHeader、 IPPutHeader、 IPSetRxBuffer 或 IPSetTxBuffer。

副作用

无

备注

调用者必须确保要获取的数据字节数不会超过活动缓冲区中可用的数据总数。此宏不检查缓冲区下溢条件。

示例

```
// 检查是否有包准备就绪
if ( MACGetHeader(&RemoteMACAddr, &PacketType) )
{
        // 检查它使用的是何种协议
        if ( PacketType == MAC_IP )
        {
                // 这是 IP 包。取出它
                IPGetHeader(&Remote, &IPProtocol, &IPLen);

                // 获取 20 个字节的数据
                IPGetArray(IPData, 20);
                ...

                // 处理完后即丢弃该包
                MACDiscardRx();
        }
        else
        {
                // 这不是 IP 包。处理它
...
```

IPSetRxBuffer

此宏允许更高级别的层设置接收缓冲区存取指针。在调用 MACSetRxBuffer 之前它使用的是 IP 包头的数据。

语法

```
void IPSetRxBuffer(WORD offset)
```

参数

offset [in]

与 IP 数据有关的偏移量

返回值

无

前提条件

无

副作用

无

备注

使用 IP 服务的各层必须调用此宏为当前缓冲区设置存取指针。此宏根据 IP 包头的长度调整偏移量数值。

示例

```
// 检查是否有包准备就绪
if ( MACGetHeader(&RemoteMACAddr, &PacketType) )
{
        // 检查它使用的是何种协议
        if ( PacketType == MAC_IP )
        {
                // 这是 IP 包。取出它
                IPGetHeader(&Remote, &IPProtocol, &IPLen);

                // 取出 IP 数据内的第 20 个字节
                IPSetRxBuffer(20);
                data = MACGet();
                ...

                // 处理完后即丢弃该包
                MACDiscardRx();
        }
        else
        {
                // 这不是 IP 包。处理它
...
```

网际控制消息协议 （ICMP）

ICMP 层由文件“ICMP.c”实现。由头文件“ICMP.h”定义该层提供的服务。

在这个架构中， ICMP 层是被动层；它不会响应 ICMP 数据包。相反，更高级别的层使用 ICMP 原始函数获取 ICMP 包，解析它并采取相应的操作。

通常， ICMP 用于发送和接收 IP 错误或诊断消息。在 Microchip TCP/IP 协议栈中， ICMP 实现了可用来生成任何 ICMP 消息的 ICMP 原始函数。在嵌入式应用程序中， ICMP 在诊断方面很有用处。使能 ICMP 后，它可以响应 “ping”包，从而使远程主机可以确定本地主机是否存在。

Microchip ICMP层只响应32个字节以下大小的ping数据包；超出 32 个字节的包将被忽略。如果需要处理更大的包，请修改编译器“定义项”MAX_ICMP_DATA_LEN（在头文件 “StackTsk.h”中）。

ICMPIsTxReady

此宏确定是否至少有一个发送缓冲区为空。

语法

```
BOOL ICMPIsTxReady()
```

参数

无

返回值

TRUE： 如果至少有一个发送缓冲区为空。

FALSE：如果没有空的发送缓冲区。

前提条件

无

副作用

无

备注

此宏只是一个抽象函数。使用 IP 服务的上层应该调用此宏，而不是直接调用 MACIsTxReady。

示例

```
// 如果 IP 发送缓冲区准备就绪，则发送 IP 包
if ( ICMPIsTxReady() )
{
        // 发送 ICMP 包
...
```

ICMPPut

此函数组装有效的 ICMP 包并发送它。

语法

```
void ICMPPut( NODE_INFO *remote,
              ICMP_CODE code,
              BYTE *data,
              BYTE len,
              WORD id,
              WORD seq)
```

参数

remote [in]

远程节点信息，例如 MAC 和 IP 地址

code [in]

供此 ICMP 包使用的 ICMP 代码

此参数的值可能为：

值	含义
ICMP_ECHO_REPLY	这是一个 ICMP Echo 回复包
ICMP_ECHO_REQUEST	这是一个 ICMP Echo 请求包

data [in]

ICMP 数据

len [in]

ICMP 数据长度

id [in]

ICMP 包标识符

seq [in]

ICMP 包序号

返回值

无

前提条件

IPIsTxReady == TRUE

副作用

无

备注

此函数要求 IP 层组装 IP 包头以及 ICMP 包，该包含有头校验和以及正确的网络字节顺序。此函数和其他层的 “Put” 函数之间的一个主要区别是，此函数在同一个函数内组装并发送包。调用者提供完整的数据缓冲区，但不需要像单独的函数调用那样为其提供数据。

示例

```
// 检查发送缓冲区是否可用
if ( ICMPIsTxReady() )
{
    // 发送 ICMP 包
    ICMPPut(&RemoteNode, ICMP_ECHO_REPLY, data, datalen, id, seq);

    // 完成。 ICMP 被放入发送队列
...
```

ICMPGet

此函数从活动的发送缓冲区获取 ICMP 包头并验证它。

语法

```
void ICMPGet( NODE_INFO *remote,
              ICMP_CODE *code,
              BYTE *data,
              BYTE *len,
              WORD *id,
              WORD *seq)
```

参数

remote [out]

远程节点信息，例如 MAC 和 IP 地址

code [out]

供接收到的 ICMP 包使用的 ICMP 代码

此参数的值可能为：

值	含义
ICMP_ECHO_REPLY	接收到 ICMP Echo 回复包
ICMP_ECHO_REQUEST	接收到 ICMP Echo 请求包
所有其他	接收到未知 / 不受支持的包

data [out]

ICMP 数据

len [out]

ICMP 数据长度

id [out]

ICMP 包标识符

seq [out]

ICMP 包序号

返回值

TRUE： 接收到有效的 ICMP 包。填充所有参数。

FALSE：接收到无效的 ICMP 包。不填充参数。

前提条件

IPGetHeader == TRUE 和 PacketType == IP_PROT_ICMP

副作用

无

备注

此函数假设活动的接收缓冲区存取指针定位在 IP 数据区域的开头。为了满足这个条件，更高级别的层在调用此函数之前必须执行以下检查：

```
If IPGetHeader == TRUE and PacketType == IP_PROT_ICMP, call ICMPGet
Else
       Do not call ICMPGet
```

一旦处理了 IP 包并且不再需要它后，调用者必须通过调用 MACDiscardRx 函数将其从 MAC 缓冲区清除。

ICMPGet（续）

示例

```
// 检查是否有包准备就绪
if ( IPGetHeader(&Remote, &IPProtocol, &IPLen) )
{
        // 检查它使用的是何种协议
        if ( IPProtocol == IP_PROT_ICMP )
        {
                // 这是 ICMP 包。取出它
                ICMPGet(&ICMPCode, data, &dataLen, &id, &seq);

                // 处理该 ICMP 包
                ...

                // 处理完后即丢弃该包
                MACDiscardRx();
        }
        else
        {
                // 这不是 ICMP 包。处理它
...
```

传输控制协议 （TCP）

Microchip TCP/IP 协议栈的 TCP 层由文件 “TCP.c” 实现。由头文件 “TCP.h” 定义该层提供的服务。在该协议栈架构中， TCP 是活动的层。它获取 TCP 包并根据 TCP 状态机对远程主机进行响应。 TCP 模块也是以协同式任务处理方式实现的，不需要应答主应用程序就可以执行自动操作。

“TCP.h” 提供 TCP 套接字服务，对调用者而言所有 TCP 包处理过程都是不可见的。此层允许有 2 到 253 个 TCP 套接字，具体数目受可用内存和所用编译器的限制。使用多个套接字，更高级别的应用程序可以同时建立多个 TCP 连接，并且可以允许多个应用程序使用该层。使用 HTTP 服务器时此工具十分有用。每个套接字占用大约 36 个字节 （具体数目请参见源文件）并会增加总体 TCP 的处理时间，了解这一点十分重要。

与其他 TCP/IP 实现方式不同， Microchip TCP/IP 协议栈中的所有套接字共享一个或多个公共发送缓冲区。这个方法减少了总体 RAM 需求，但是可能会带来一个潜在问题，即少数套接字保留了所有可用发送缓冲区，且不能及时释放它们供其他套接字使用。在这种情况下，远程主机和 / 或本地应用程序将无法使用协议栈。为了避免这一点，用户必须确保所有套接字都有足够的发送缓冲区。

在接收方，只有一个接收缓冲区。如果套接字接收到它的数据，该套接字的所有者必须在一个任务时间内获取数据并释放接收缓冲区，以使其他套接字可以接收它们的数据。这个设计强制任务一旦检测到它感兴趣的包，必须在一个任务时间内获取整个包，而不能在一个任务时间内获取包的一部分，并期望稍后获取剩余部分。

根据 TCP 规范的要求，每个 TCP 段必须包含 1 个覆盖整个 TCP 包 （包括数据区域）的校验和。为了减少对 RAM 的要求， TCP 层使用 NIC 中的 MAC 缓冲区作为存储器，并在 MAC 缓冲区中执行校验和计算。如果 NIC 用作 MAC，则 NIC SRAM 用作缓冲区空间。但是如果将 SLIP 用作 MAC，则使用单片机的内部数据 RAM。

Microchip TCP/IP 协议栈的 TCP 层实现了 RFC793 建议的大多数 TCP 状态机状态。它还实现了自动重试和定时操作，用户可以通过编译时间 “定义项” TCP_NO_WAIT_FOR_ACK 来使能或禁止这些操作。自动重试被使能后，每个套接字发送缓冲区将被保留，直到接收到来自远程主机的应答为止。这个设计可以有效地为每个 TCP 段建立一个发送时间窗口。因而，数据吞吐量将显著低于 “无重试” 模式下的数据吞吐量。如果只使用 HTTP 服务器应用程序，则用户可以禁止自动重试从而有效地增加吞吐量。如果主应用程序的逻辑要求在发送新包前主机需要对前一个包作出应答，则用户必须使能 “自动重试” 模式。使能 “自动重试” 后，某些打开的连接可能无法及时获得服务，远程主机可能发生超时或复位错误。

TCPInit

此函数初始化 TCP 状态机并将其准备好进行多个 TCP 连接。

语法

```
void TCPInit()
```

参数

无

返回值

无

前提条件

无

副作用

无

备注

此函数只在应用程序启动时调用一次。

示例

```
// 初始化 TCP
TCPInit();
```

TCPListen

此函数指定某个可用的套接字用于侦听给定的 TCP 端口。

语法

TCP_SOCKET TCPListen(TCP_PORT *port*)

参数

port [in]

要侦听的 TCP 端口号

返回值

有效的套接字标识符：如果至少有一个套接字可用。

INVALID_SOCKET：如果没有可用的套接字。

前提条件

无

副作用

无

备注

无

示例

```
...

switch(smState)
{
case SM_LISTEN:
        // 侦听 HTTP 请求
        httpSocket = TCPListen(80);
        If ( httpSocket == INVALID_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
                ...
        }
        else
                smState = SM_LISTEN_WAIT;
        return;
case SM_LISTEN_WAIT:
        // 等待连接 ...
...
```

TCPConnect

此函数在给定远程端口上启动对远程主机的连接请求。

语法

TCP_SOCKET TCPConnect(NODE_INFO *remote, TCP_PORT port)

参数

remote [in]

欲连接的远程主机

port [in]

欲连接的远程主机上的 TCP 端口号

返回值

有效的套接字标识符：如果至少有一个套接字可用，并且连接请求已发送。

INVALID_SOCKET：如果没有可用的套接字。

前提条件

无

副作用

无

备注

此函数仅在定义了 STACK_CLIENT_MODE 后才可用 （见第 3 页的 “协议栈配置”）。

示例

```
...

switch(smState)
{
case SM_CONNECT:
        // 连接到远程 FTP 服务器
        ftpSocket = TCPConnect(&RemoteNode, 21);
        If ( ftpSocket == INVALID_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
        }
        else
                smState = SM_CONNECT_WAIT;
        return;
case SM_CONNECT_WAIT:
        // 等待连接 ...
...
```

TCPIsConnected

此函数用于确定给定套接字是否已连接到远程主机。

语法

BOOL TCPIsConnected(TCP_SOCKET *socket*)

参数

socket [in]

检查其连接的套接字标识符

返回值

TRUE: 如果给定套接字已连接到远程主机。

FALSE: 如果给定套接字未连接到远程主机。

前提条件

无

副作用

无

备注

无

示例

```
...

switch(smState)
{
case SM_CONNECT:
        // 连接到远程 FTP 服务器
        ftpSocket = TCPConnect(&RemoteNode, 21);
        If ( ftpSocket == INVALID_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
        }
        else
                smState = SM_CONNECT_WAIT;
        return;
case SM_CONNECT_WAIT:
        // 等待连接 ...
        if ( TCPIsConnected(ftpSocket) )
                smState = SM_CONNECTED;
        return;
...
```

TCPDisconnect

此函数请求远程主机断开连接。

语法

```
void TCPDisconnect(TCP_SOCKET socket)
```

参数

socket [in]

需要断开连接的套接字标识符

返回值

无

前提条件

无

副作用

无

备注

无

示例

```
...

switch(smState)
{
case SM_CONNECT:
        // 连接到远程 FTP 服务器
        ftpSocket = TCPConnect(&RemoteNode, 21);
        If ( ftpSocket == INVALID_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
        }
        else
                smState = SM_CONNECT_WAIT;
        return;
case SM_CONNECT_WAIT:
        // 等待连接 ...
        if ( TCPIsConnected(ftpSocket) )
                smState = SM_CONNECTED;
        return;
case SM_CONNECTED:
        // 装入数据
        ...
        // 断开连接
        TCPDisconnect(ftpSocket);
...
```

TCPIsPutReady

此函数用于确定套接字是否已准备好发送。当套接字已经连接到远程主机并且它的发送缓冲区为空时，该套接字已准备好发送。

语法

BOOL TCPIsPutReady(TCP_SOCKET *socket*)

参数

socket [in]

需要检查的套接字标识符

返回值

TRUE: 如果给定套接字已经准备好发送。

FALSE: 如果给定套接字没有建立连接或没有可用的发送缓冲区。

前提条件

无

副作用

无

备注

无

示例

```
...

switch(smState)
{
case SM_CONNECT:
        // 连接到远程 FTP 服务器
        ftpSocket = TCPConnect(&RemoteNode, 21);
        If ( ftpSocket == INVALID_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
        }
        else
                smState = SM_CONNECT_WAIT;
        return;
case SM_CONNECT_WAIT:
        // 等待连接 ...
        if ( TCPIsConnected(ftpSocket) )
                smState = SM_CONNECTED;
        return;
case SM_CONNECTED:
        // 装入数据
        if ( TCPIsPutReady(ftpSocket) )
        {
                // 装入数据
...
```

TCPPut

此函数将数据字节装入给定套接字的发送缓冲区中。

语法

```
BOOL TCPPut(TCP_SOCKET socket, BYTE byte)
```

参数

socket [in]

需要检查的套接字标识符

byte [in]

要装入的数据字节

返回值

TRUE： 如果给定数据字节已经成功装入发送缓冲区，并且还有空间可以容纳更多的数据。

FALSE：如果给定数据字节已经成功装入发送缓冲区，但没有空间可以容纳更多的数据。

前提条件

```
TCPIsPutReady == TRUE
```

副作用

无

备注

一旦发现套接字已准备好进行发送，用户必须装入所有数据，或者装入尽可能多的数据直到套接字缓冲区满了为止。用户不能将数据的一部分装入一个套接字缓冲区，而将另一部分装入另一个套接字缓冲区中。

使用者必须注意，使用此函数装入套接字数据时，发送能否成功进行取决于已装入的数据字节总数。如果装入的字节数小于可用的套接字缓冲区，用户必须使用 TCPFlush 函数清空发送缓冲区（启动发送）。如果用户尝试装入多于可用缓冲区大小的字节数，此函数将自动开始发送并返回 FALSE，这样用户就可以重新尝试发送。通常，不管缓冲区是否已满，在所有已知数据装入缓冲区后清空套接字缓冲区是一个很好的习惯。

TCPPut（续）

示例

```
...

switch(smState)
{
case SM_CONNECT:
        // 连接到远程 FTP 服务器
        ftpSocket = TCPConnect(&RemoteNode, 21);
        If ( ftpSocket == INVALID_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
        }
        else
                smState = SM_CONNECT_WAIT;
        return;
case SM_CONNECT_WAIT:
        // 等待连接 ...
        if ( TCPIsConnected(ftpSocket) )
                smState = SM_CONNECTED;
        return;
case SM_CONNECTED:
        // 装入数据
        if ( TCPIsPutReady(ftpSocket) )
        {
                // 装入数据
                TCPPut(ftpSocket, dataByte);
...
```

TCPFlush

此函数将给定套接字发送缓冲区标记为准备好可以发送。

语法

```
void TCPFlush(TCP_SOCKET socket)
```

参数

socket [in]

需要发送的套接字标识符

返回值

无

前提条件

无

副作用

无

备注

此函数将当前发送缓冲区标记为准备好发送，而实际发送不一定立即开始。用户不需要检查发送的状态。 NIC 最多将重试发送消息 15 次（适用于 RTL8019AS。如果未使用 RTL8019AS，该数值需参考特定 NIC 的文档）。如果套接字已被清空，则随后的清空操作会被忽略。

示例

```
...

switch(smState)
{
case SM_CONNECT:
        // 连接到远程 FTP 服务器
        ftpSocket = TCPConnect(&RemoteNode, 21);
        If ( ftpSocket == INVALID_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
        }
        else
                smState = SM_CONNECT_WAIT;
        return;
case SM_CONNECT_WAIT:
        // 等待连接 ...
        if ( TCPIsConnected(ftpSocket) )
                smState = SM_CONNECTED;
        return;
case SM_CONNECTED:
        // 装入数据
        if ( TCPIsPutReady(ftpSocket) )
        {
                // 装入数据
                TCPPut(ftpSocket, dataByte);
                ...

                // 现在进行发送
                TCPFlush(ftpSocket);
...
```

TCPIsGetReady

此函数用于确定给定套接字是否包含接收数据。

语法

BOOL TCPIsGetReady(TCP_SOCKET *socket*)

参数

socket [in]

需要发送的套接字标识符

返回值

TRUE: 如果给定套接字包含接收数据。

FALSE: 如果给定套接字不包含任何数据。

前提条件

无

副作用

无

备注

无

示例

```
...

switch(smState)
{
case SM_LISTEN:
        // 侦听 HTTP 套接字
        httpSocket = TCPListen(&RemoteNode, 80);
        If ( httpSocket == INVALID_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
        }
        else
                smState = SM_LISTEN_WAIT;
        return;
case SM_LISTEN_WAIT:
        // 等待连接 ...
        if ( TCPIsConnected(httpSocket) )
                smState = SM_CONNECTED;
        return;
case SM_CONNECTED:
        // 取数据
        if ( TCPIsGetReady(httpSocket) )
        {
                // 取数据
...
```

TCPGet

此函数从给定套接字接收缓冲区获取一个数据字节。

语法

BOOL TCPGet(TCP_SOCKET *socket*, BYTE **byte*)

参数

socket [in]

需要获取的套接字标识符

byte [out]

已经读取的数据字节

返回值

TRUE： 如果已经读取一个字节。

FALSE：如果没有读取任何字节。

前提条件

TCPIsGetReady == TRUE

副作用

无

备注

如果发现套接字包含接收数据，用户必须在一个任务时间内获取一个或更多数据字节 （如果需要），然后清空该套接字缓冲区。直到第一个套接字的缓冲区内容被清空后，才能从另一个套接字获取数据。

示例

```
...
switch(smState)
{
case SM_LISTEN:
        // 侦听 HTTP 套接字
        httpSocket = TCPListen(&RemoteNode, 80);
        If ( httpSocket == INVALID_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
        }
        else
                smState = SM_LISTEN_WAIT;
        return;
case SM_LISTEN_WAIT:
        // 等待连接 ...
        if ( TCPIsConnected(httpSocket) )
                smState = SM_CONNECTED;
        return;
case SM_CONNECTED:
        // 取数据
        if ( TCPIsGetReady(httpSocket) )
        {
                // 取数据
                TCPGet(httpSocket, &dataByte);
...
```

TCPGetArray

此函数从给定套接字接收缓冲区获取一个数据数组。

语法

```
WORD TCPGetArray(TCP_SOCKET socket,
                 BYTE *byte,
                 WORD count)
```

参数

socket [in]

需要获取的套接字标识符

byte [out]

已经读取的数据数组

count [out]

要读取的字节总数

返回值

已读取的数据字节总数。

前提条件

`TCPIsGetReady == TRUE`

副作用

无

备注

如果发现套接字包含接收数据，则用户必须获取一个或更多数据字节 （如果需要），然后清空该套接字缓冲区。直到第一个套接字的缓冲区内容被清空后，才能从另一个套接字获取数据。

这个函数不会检查申请读取的字节数是否超出接收缓冲区中提供的数据字节数。用户必须确定当前接收缓冲区不会发生下溢。

TCPGetArray（续）

示例

```
...

switch(smState)
{
case SM_LISTEN:
        // 侦听 HTTP 套接字
        httpSocket = TCPListen(&RemoteNode, 80);
        If ( httpSocket == INVALID_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
        }
        else
                smState = SM_LISTEN_WAIT;
        return;
case SM_LISTEN_WAIT:
        // 等待连接 ...
        if ( TCPIsConnected(httpSocket) )
                smState = SM_CONNECTED;
        return;
case SM_CONNECTED:
        // 取数据
        if ( TCPIsGetReady(httpSocket) )
        {
                // 取 20 个字节的数据
                TCPGetArray(httpSocket, buffer, 20);
...
```

TCPDiscard

此函数释放与给定套接字关联的接收缓冲区。

语法

BOOL TCPDiscard(TCP_SOCKET *socket*)

参数

socket [in]

需要发送的套接字标识符

返回值

TRUE: 如果成功释放了指定的接收缓冲区。

FALSE: 如果指定的接收缓冲区已经被清空。

前提条件

无

副作用

无

备注

无

示例

```
...

switch(smState)
{
case SM_LISTEN:
        // 侦听 HTTP 套接字
        httpSocket = TCPListen(&RemoteNode, 80);
        If ( httpSocket == INVALID_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
        }
        else
                smState = SM_LISTEN_WAIT;
        return;
case SM_LISTEN_WAIT:
        // 等待连接 ...
        if ( TCPIsConnected(httpSocket) )
                smState = SM_CONNECTED;
        return;
case SM_CONNECTED:
        // 取数据
        if ( TCPIsGetReady(httpSocket) )
        {
                // 取 20 个字节的数据
                TCPGetArray(httpSocket, buffer, 20);

                // 处理数据
                ...

                // 释放缓冲区
                TCPDiscard(httpSocket);
...
```

TCPProcess

此函数充当 “TCPTask”。它获取已接收到的 TCP 包，并对匹配的套接字执行 TCP 状态机。仅当接收到 TCP 包后才可以调用这个函数。

语法

```
BOOL TCPProcess(NODE_INFO *remote, WORD len)
```

参数

remote [in]

发出当前 TCP 包的远程节点

len [out]

TCP 包的总长度 （包括 TCP 包头）

返回值

TRUE： 如果此函数 （任务）已完全处理了当前包。

FALSE：如果此函数 （任务）已部分处理了当前包。

前提条件

IPGetHeader == TRUE 和 IPProtocol = IP_PRO_TCP

副作用

无

备注

如上所述，此函数实现了 TCP 状态机。接收到有效的 TCP 数据包时，用户必须调用这个函数。一旦检测到包，此函数将获取并处理这个包。此函数的返回值指示调用者是否需要更改 StackTask 状态机状态。

在它的当前实现方案中，此函数总是返回 TRUE。请参见 Stack Manager 源代码 （StackTsk.c）以获取详细信息。

示例

```
...

switch(smState)
{
case SM_STACK_IDLE:
        if ( MACGetHeader(&RemoveMAC, &MACFrameType) )
        {
                if ( MACFrameType == MAC_IP )
                        smState = SM_STACK_IP;
        ...
        return;
case SM_STACK_IP:
        if ( IPGetHeader(&RemoteNode, &IPFrameType, &IPDataCount) )
        {
                if ( IPFrameType == IP_PROT_TCP )
                        smState = SM_STACK_TCP;
        ...
        return;
case SM_STACK_TCP:
        if ( TCPProcess(&RemoteNode, IPDataCount) )
                smState = SM_STACK_IDLE;
        return;
...
```

TCPTick

此函数充当除了 TCPProcess 以外的另一个 “TCPTask”。它检查所有套接字的超时条件并尝试恢复。

语法

```
void TCPTick()
```

参数

无

返回值

无

前提条件

无

副作用

无

备注

此函数实现了超时处理。用户必须定期调用这个函数以确保能及时处理超时条件。

示例

```
TCPTick();
```

用户数据报协议 （UDP）

Microchip TCP/IP 协议栈的 UDP 层由文件 “UDP.c” 实现。头文件 “UDP.h” 定义该层提供的服务。在这个协议栈架构中，UDP 是活动的层。它获取 UDP 包并通知相应的 UDP 套接字有关数据的到达或发送。UDP 模块以协同式任务处理方式实现，不需要应答主应用程序就可以执行自动操作。

“UDP.h” 提供 UDP 套接字服务，对调用者而言所有 UDP 包处理过程都是不可见的。此层允许最多有 254 个 UDP 套接字，具体数值受可用内存和所用编译器的限制。使用多个套接字，更高级别的应用程序可以同时建立多个 UDP 连接，并且可以允许多个应用程序使用该层。每个套接字占用大约 19 个字节 （具体数值请参见 “UDP.h” 文件）并增加了总体 UDP 的处理时间，了解这一点十分重要。

与其他套接字的实现方式不同，Microchip TCP/IP 协议栈中的所有套接字都共享一个或多个公共发送缓冲区。这个方法减少了对总体 RAM 的需求，但是可能会带来一个潜在问题，即少数套接字保留所有可用发送缓冲区，且不能及时释放它们供其他套接字使用。在这种情况下，远程主机和/或本地应用程序将无法使用协议栈。为了避免这一点，用户必须确保所有套接字都有足够的发送缓冲区。

在接收方，只有一个接收缓冲区。如果套接字接收到数据，该套接字的所有者必须在一个任务时间内获取数据并释放接收缓冲区，以使其他套接字可以接收它们的数据。这个设计强制任务一旦检测到它感兴趣的包，必须在一个任务时间内获取整个包，而不能在一个任务时间内获取包的一部分，并期望稍后获取剩余部分。

UDP 规范并没有强制对 UDP 包执行校验和计算。为了减少对程序和数据存储器的要求，Microchip TCP/IP 协议栈没有实现 UDP 校验和计算；而是将校验和字段设置为零来表明不执行校验和计算。因此需要使用UDP模块的所有模块必须确保它们的数据完整性。

UDPInit

此函数初始化 UDP 模块并将其准备好进行多 UDP 连接。

语法

```
void UDPInit()
```

参数

无

返回值

无

前提条件

无

副作用

无

备注

此函数只在应用程序启动时调用一次。

示例

```
// 初始化 UDP
UDPInit();
```

UDPOpen

此函数在给定端口上准备下个可用的 UDP 套接字，以进行可能的数据发送。本地或远程节点都可以启动数据发送。

语法

UDP_SOCKET UDPOpen(UDP_PORT *localPort*, NODE_INFO **remoteNode*, UDP_PORT *remotePort*)

参数

localPort [in]

发送数据的本地 UDP 端口号

remoteNode [in]

包含 *remotePort* 的远程主机

remotePort [in]

远程主机上用于来回发送数据的 UDP 端口号

返回值

有效的套接字标识符：如果至少有一个套接字可用。

INVALID_UDP_SOCKET：如果没有可用的套接字。

前提条件

无

副作用

无

备注

此函数既可用于本地主机启动的数据发送，也可用于远程主机启动的数据发送。UDP 不需要明确的连接步骤。一旦 UDP 套接字被打开后，它就准备好发送或接收数据。所谓的套接字被打开，指的是已设置好相应的套接字来接收或发送一个或多个数据包，直到该套接字关闭为止。

示例

```
...

switch(smState)
{
case SM_OPEN:
        // 与远程 DHCP 服务器进行对话
        DHCPSocket = UDPOpen(68, &DHCPServerNode, 67);
        If ( DHCPSocket == INVALID_UDP_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
        }
        else
                // 发送 DHCP 广播消息
        break;
...
```

UDPClose

此函数关闭给定的 UDP 套接字并将其声明为可用套接字。

语法

```
void UDPDisconnect(UDP_SOCKET socket)
```

参数

socket [in]

需要关闭的套接字的标识符

返回值

无

前提条件

无

副作用

无

备注

无

示例

```
...

switch(smState)
{
case SM_OPEN:
        // 与远程 DHCP 服务器进行对话
        DHCPSocket = UDPOpen(68, &DHCPServerNode, 67);
        If ( DHCPSocket == INVALID_UDP_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
        }
        else
                // 发送 DHCP 请求 ...
                ...
                // 关闭套接字
                UDPClose(DHCPSocket);
        break;
...
```

UDPIsPutReady

此宏用于确定给定套接字是否准备好发送。当至少有一个 MAC 发送缓冲区为空时，套接字即准备好发送。它还将给定套接字设置为活动的 UDP 套接字。

语法

BOOL UDPIsPutReady(UDP_SOCKET *socket*)

参数

socket [in]

需要进行检查并设置为活动的套接字的标识符

返回值

TRUE： 如果给定套接字已准备好发送。

FALSE： 如果发送缓冲区尚未准备就绪。

前提条件

无

副作用

无

备注

由于 UDP 是一个无连接协议，所以只要至少有一个 MAC 发送缓冲区为空，套接字就可以准备进行发送。除了检查发送器是否就绪外， UPDIsPutReady 还设置活动的 UDP 套接字。一旦将套接字设置为活动的，之后对其他 UDP 函数的调用（例如 UDPGet、UDPPut、UDPFlush 和 UDPDiscard）都将活动的套接字用作当前套接字。因此，用户不需要在每次调用时传递套接字。

请参见 UDPGet、 UDPPut、 UDPFlush 和 UDPDiscard 函数以了解更多信息。

示例

```
...

switch(smState)
{
case SM_OPEN:
        // 与远程 DHCP 服务器进行对话
        DHCPSocket = UDPOpen(68, &DHCPServerNode, 67);
        If ( DHCPSocket == INVALID_UDP_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
        }
        else
                // 发送 DHCP 广播消息
                smState = SM_BROADCAST;
        break;
case SM_BROADCAST:
        if ( UDPIsPutReady(DHCPSocket) )
        {
                // 套接字准备好进行发送。发送数据 ...
                ...
        }
        break;
...
```

UDPPut

此函数将一个数据字节装入活动的套接字的发送缓冲区中。

语法

```
BOOL UDPPut(BYTE byte)
```

参数

byte [in]

要装入的数据字节

返回值

TRUE： 如果给定数据字节已经成功装入发送缓冲区，并且还有空间可以容纳更多的数据。

FALSE：如果给定数据字节已经成功装入发送缓冲区，但没有空间可以容纳更多的数据。

前提条件

```
UDPIsPutReady == TRUE
```

副作用

无

备注

一旦发现套接字已准备好进行发送，用户必须装入所有数据，或者装入尽可能多的数据直到套接字缓冲区满了为止。用户不能将数据的一部分装入一个套接字缓冲区，而将另一部分装入另一个套接字缓冲区中。

使用者必须注意，使用此函数装入套接字数据时，发送能否成功进行取决于已装入的数据字节总数。如果装入的字节未装满可用的套接字缓冲区，用户必须使用 UDPFlush 函数清空发送缓冲区（启动发送）。如果用户尝试装入的字节数超过可用缓冲区的大小，此函数将自动开始发送并返回 FALSE，这样用户就可以重新尝试发送。通常，不管缓冲区是否已满，在所有已知数据装入缓冲区后清空套接字缓冲区是一个很好的习惯。

UDPPut（续）

示例

```
...

switch(smState)
{
case SM_OPEN:
        // 与远程 DHCP 服务器进行对话
        DHCPSocket = UDPOpen(68, &DHCPServerNode, 67);
        If ( DHCPSocket == INVALID_UDP_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
        }
        else
                // 发送 DHCP 广播消息
                smState = SM_BROADCAST;
        break;
case SM_BROADCAST:
        if ( UDPIsPutReady(DHCPSocket) )
        {
                // 套接字准备好进行发送。发送数据 ...
                // 注意 UDPPut 中有一个 DHCPSocket 参数
                // 该 UDPPut 调用将使用活动的套接字
                // 由 UDPIsPutReady() 设置 （DHCPSocket）
                UDPPut(0x55);
                ...
        }
        break;
...
```

UDPFlush

此函数将活动的套接字发送缓冲区标记为准备好可以发送。

语法

```
void UDPFlush()
```

参数

返回值

无

前提条件

已经调用了 UDPPut()，并且通过调用 UDPIsPutReady() 将所需的 UDP 套接字设置为活动的套接字。

副作用

无

备注

此函数将当前发送缓冲区标记为准备好可以发送，而实际发送不一定立即开始。用户不需要检查发送的状态。 NIC 最多将重试发送 15 次（适用于 RTL8019AS。如果未使用 RTL8019AS，具体值请参考特定 NIC 的文档）。如果套接字已被清空，则后续的清空操作将被忽略。

示例

```
...

switch(smState)
{
case SM_OPEN:
        // 与远程 DHCP 服务器进行对话
        DHCPSocket = UDPOpen(68, &DHCPServerNode, 67);
        If ( DHCPSocket == INVALID_UDP_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
        }
        else
                // 发送 DHCP 广播消息
                smState = SM_BROADCAST;
        break;
case SM_BROADCAST:
        if ( UDPIsPutReady(DHCPSocket) )
        {
                // 套接字准备好进行发送。发送数据 ...
                // 注意 UDPPut 中有一个 DHCPSocket 参数
                // 该 UDPPut 调用将使用活动的套接字
                // 由 UDPIsPutReady() 设置 （DHCPSocket）
                UDPPut(0x55);
                ...
                // 现在进行发送
                UDPFlush();
        }
        break;
...
```

UDPIsGetReady

此函数用于确定给定套接字是否包含接收数据。它还将给定套接字设置为活动的套接字。

语法

BOOL UDPIsGetReady(UDP_SOCKET *socket*)

参数

socket [in]

被发送并设置为活动的套接字的标识符

返回值

TRUE: 如果给定套接字包含接收数据。

FALSE: 如果给定套接字不包含任何数据。

前提条件

已经调用 UDPOpen()。 *socket* 的值必须是 UDPOpen() 调用返回的值。

副作用

无

备注

无

示例

```
...

switch(smState)
{
case SM_OPEN:
        // 与远程 DHCP 服务器进行对话
        DHCPSocket = UDPOpen(68, &DHCPServerNode, 67);
        If ( DHCPSocket == INVALID_UDP_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
        }
        else
                // 等待来自 DHCP 服务器的响应
                smState = SM_WAIT_FOR_DATA;
        break;
case SM_WAIT_FOR_DATA:
        if ( UDPIsGetReady(DHCPSocket) )
        {
                // 套接字包含某些数据。取出并进行处理
                ...
        }
        break;
...
```

UDPGet

此函数从活动的套接字接收缓冲区获取一个数据字节。

语法

BOOL UDPGet(BYTE *byte)

参数

byte [out]

已经读取的数据字节

返回值

TRUE： 如果已经读取一个字节。

FALSE：如果没有读取任何字节。

前提条件

UDPIsGetReady == TRUE

副作用

无

备注

如果发现套接字包含接收数据，用户必须在一个任务时间内获取一个或更多数据字节 （如果需要），然后清空该套接字缓冲区。直到清空了第一个套接字缓冲区的内容后，才能从另一个套接字获取数据。

示例

```
...

switch(smState)
{
case SM_OPEN:
        // 与远程 DHCP 服务器进行对话
        DHCPSocket = UDPOpen(68, &DHCPServerNode, 67);
        If ( DHCPSocket == INVALID_UDP_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
        }
        else
                // 等待来自 DHCP 服务器的响应
                smState = SM_WAIT_FOR_DATA;
        break;
case SM_WAIT_FOR_DATA:
        if ( UDPIsGetReady(DHCPSocket) )
        {
                // 套接字包含某些数据。全部取出
                // 缓冲区是 BYTE 的指针
                while( UDPGet(buffer) )
                        buffer++;
                // 处理它
                ...
                // 清空套接字缓冲区
                ...
        }
        break;
...
```

UDPDiscard

此函数释放与活动的套接字关联的接收缓冲区。

语法

```
BOOL UDPDiscard()
```

参数

无

返回值

TRUE: 如果成功清空了接收缓冲区。

FALSE: 如果接收缓冲区已经被清空。

前提条件

无

副作用

无

备注

无

示例

```
...

switch(smState)
{
case SM_OPEN:
        // 与远程 DHCP 服务器进行对话
        DHCPSocket = UDPOpen(68, &DHCPServerNode, 67);
        If ( DHCPSocket == INVALID_UDP_SOCKET )
        {
                // 套接字不可用
                // 返回出错信息
        }
        else
                // 等待来自 DHCP 服务器的响应
                smState = SM_WAIT_FOR_DATA;
        break;
case SM_WAIT_FOR_DATA:
        if ( UDPIsGetReady(DHCPSocket) )
        {
                // 套接字包含某些数据。全部取出
                // 缓冲区是 BYTE 的指针
                while( UDPGet(buffer) )
                        buffer++;
                // 处理它 ..
                ...
                // 清空套接字缓冲区
                UDPDiscard();
        }
        break;
...
```

UDPProcess

此函数充当 “UDPTask”。它获取已接收到的 UDP 包，并将其分配给匹配的 UDP 套接字。仅当接收到 UDP 包后才可以调用这个函数。

语法

BOOL UDPProcess(NODE_INFO *remote, WORD len)

参数

remote [in]

发出当前 UDP 包的远程节点

len [in]

UDP 包的总长度 （包括 UDP 包头）

返回值

TRUE： 如果此函数 （任务）已全部处理了当前包。

FALSE：如果此函数 （任务）已部分处理了当前包。

前提条件

IPGetHeader == TRUE 和 IPProtocol = IP_PRO_UDP

副作用

无

备注

如上所述，此函数实现了 UDP 任务。接收到有效的 UDP 数据包时，用户必须调用这个函数。一旦检测到包，此函数将获取并处理这个包。此函数的返回值指示调用者是否需要更改 StackTask 状态机状态。在它的当前实现方案中，这个函数总是返回 TRUE。

请参见 Stack Manager 源代码 （StackTsk.c）以获取详细信息。

示例

```
...

switch(smState)
{
case SM_STACK_IDLE:
        if ( MACGetHeader(&RemoveMAC, &MACFrameType) )
        {
                if ( MACFrameType == MAC_IP )
                        smState = SM_STACK_IP;
        ...
        return;
case SM_STACK_IP:
        if ( IPGetHeader(&RemoteNode, &IPFrameType, &IPDataCount) )
        {
                if ( IPFrameType == IP_PROT_UDP )
                        smState = SM_STACK_UDP;
        ...
        return;
case SM_STACK_UDP:
        if ( UDPProcess(&RemoteNode, IPDataCount) )
                smState = SM_STACK_IDLE;
        return;
...
```

动态主机配置协议 （DHCP）

Microchip TCP/IP 协议栈的 DHCP 层由文件“dhcp.c”实现。头文件“dhcp.h”定义该层提供的服务。DHCP 是一个活动的层，它广播 DHCP 请求并自动接收和解码 DHCP 响应。它的主要功能包括：

- 配置 IP 地址、网关地址和子网掩码
- 自动 DHCP 租用时间和租用延期管理
- 不需要用户应用程序干预的完全自动化操作

DHCP 模块以协同式任务处理方式实现，不需要应答主应用程序就可以执行自动操作。实际上 DHCP 集成和控制是 Stack Manager 完成的；它将所有必需的操作作为它的标准任务的一部分来处理，并使用 DHCP API 来控制模块的行为。用户在使用 DHCP 时并不需要知晓任何细节。

用户应用程序还可以选择调用某些 API 来直接控制 DHCP 操作，例如是否配置 DHCP 以及是否永久停止 DHCP。通常，用户的应用程序根本不需要直接与 DHCP 交互。

要使用 DHCP 模块，必须对用户项目文件进行如下修改：

1. 在头文件“StackTsk.h”中取消对 STACK_USE_DHCP 的注释。
2. 将“dhcp.c”和“udp.c”文件包含在项目中。
3. 将 MAX_UDP_SOCKETS 加 1（对于 DHCP，必须至少有一个 UDP 套接字；根据 UDP 和 DHCP 的需要调整套接字的数目）。

实现 DHCP 时，在正确配置 DHCP 之前，用户应用程序不得尝试进行网络通讯。通常，如果用户应用程序包含一个或多个在上电或复位时需要通讯的客户机应用程序，则应用程序在使用低层模块发送任何数据之前必须检查是否配置了 DHCP。可以使用函数 DHCPIsBound 来实现 （第 75 页）。

官方 DHCP 规范（RFC1541）要求 DHCP 客户机在其租用时间到期之前更新 IP 配置。为跟踪租用时间，用户应用程序必须确保按需要调用了 TickUpdate()，并使用了适当的时间精度（实际示例见源代码文件“websrvr.c”）。要求的时间精度是 15 分钟（正或负），表示 TickUpdate() 调用的优先级非常低。

为了使 DHCP 模块自动配置地址和子网掩码，在网络中必须至少有一个 DHCP 服务器。用户必须负责采取一些措施将配置“发布”给潜在用户。可选择的方法有手动从每个节点上的显示器读取信息，或者将信息存储在中央服务器上。作为自动配置的结果，DHCP 会更新 MY_IP_BYTE?、MY_GATE_BYTE?和 MY_MASK_BYTE? 的值。

DHCPTask

这是一个执行必要协议操作的 DHCP 任务函数。

语法

```
void DHCPTask()
```

参数

无

返回值

无

前提条件

无

副作用

无

备注

必须定期调用此函数。

示例

```
// 调用 DHCP 任务
DHCPTask();
```

DHCPDisable

此宏禁止 DHCP 模块，即使是在调用它的任务以前。

语法

```
void DHCPDisable()
```

参数

无

返回值

无

前提条件

必须在调用 DHCPTask 之前调用它。

副作用

无

备注

如果应用程序需要用户选择将禁止 DHCP 模式作为其配置的一部分，则这个函数十分有用。通常，如果不需要 DHCP，则不应将其包含在项目中。但是一旦应用程序需要在运行时选择 DHCP，这个函数可用来禁止 DHCP。Microchip TCP/IP 协议栈演示应用程序使用此函数来说明如何在运行时禁止 DHCP 模块。

必须在调用 DHCPTask 函数之前调用此函数。一旦已经调用了 DHCPTask，则必须使用 DHCPAbort （第 74 页）来禁止 DHCP。如果 DHCPDisable 是在执行了 DHCPTask 后调用的，则 DHCP UDP 套接字将变成孤儿套接字。这个节点接收到的任何新的 DHCP 响应将存储在 DHCP UDP 套接字中，但是不会被任何应用程序获取；最终，所有的应用程序或层都无法再接收到数据。

示例

```
void main()
{
        // 初始化应用程序和协议栈
        ...
        StackInit();

        // 如果需要，在此禁止 DHCP
        if ( DHCPIsDisabled )
                DHCPDisable();

        // 现在进入主循环
        while(1)
        {
                // 执行必要的步骤
                ...
        }
}
```

DHCPAbort

此函数中止一个已经激活的 DHCP 任务并在应用程序的有效期内禁止该任务。

语法

```
void DHCPAbort()
```

参数

无

返回值

无

前提条件

必须至少在调用了 DHCPTask 一次以后才能调用它。

副作用

无

备注

此函数可用于在调用 DHCPTask 后禁止或中止 DHCP。请注意，一旦 DHCPTask 开始后，节点可能已经获得了一个 DHCP 配置。如果在获得一个 IP 配置后中止 DHCP，则不会自动更新这个配置。无法更新配置会导致该节点在网络中无法通讯。它还可能使得同一 IP 地址被分配给另一个节点，从而导致不可靠的通讯。

Microchip 协议栈不提供任何对 IP 配置进行验证的反馈信息。

示例

```
void main()
{
        // 初始化应用程序和协议栈
        ...
        StackInit();

        // 现在进入主循环
        while(1)
        {
                // 执行必要的步骤
                ...
                // 经过一些迭代后，应用程序需要中止 DHCP
                DHCPAbort();
        }
}
```

DHCPIsBound

此宏检查 DHCP 模块是否获得了 IP 配置 （“绑定”）。

语法

```
BOOL DHCPIsBound()
```

参数

无

返回值

TRUE： 如果 DHCP 绑定到一个 IP 配置。

FALSE：如果 DHCP 还未绑定。

前提条件

无

副作用

无

备注

使用此函数来确定 DHCP 是否绑定。当用户应用程序包括一个或多个需要在上电或复位时进行通讯的客户机应用程序时，这个函数十分有用。尝试进行通讯之前，用户应用程序必须等到 DHCP 绑定到 IP 配置。

当 DHCP 绑定到 IP 配置时，会更新 MY_IP_BYTE?、 MY_GATE_BYTE? 和 MY_MASK_BYTE?。

示例

```
void main()
{
        // 初始化应用程序和协议栈
        ...
        StackInit();

        // 现在进入主循环
        while(1)
        {
                // 执行必要的步骤
                ...
                // 检查 DHCP 是否已绑定。如果是，则显示 IP 地址
                if ( DHCPIsBound() )
                {
                        // 显示 MY_IP_BYTE? 的值
                        DisplayIPAddress();
                }
        }
}
```

用于 IP 地址配置的 IP Gleaning

作为 DHCP 的简单替代品，Microchip TCP/IP 协议栈还使用了一种称为 *IP Gleaning* 的简单方法，来远程设置 Microchip 协议栈节点的 IP 地址。这种方法*不是*网际协议标准，没有相应的 RFC 文档。IP Gleaning 只允许设置 IP 地址。要进行完整的 IP 配置，则必须使用 DHCP。

IP Gleaning不需要任何额外的软件模块。而是使用ARP 和 ICMP 模块。要使用它，必须将文件 “icmp.c” 包含在项目中，并且必须在StackTsh.h头文件中取消对编译器“定义项”STACK_USE_IP_GLEANING的注释。

当 IP Gleaning 被使能后，Microchip 协议栈在复位时进入到特殊的“IP 配置”模式。在这种模式下，它接受任何以该节点为目标的 ICMP （ping）消息，并将该节点的 IP 地址设置为 ping 消息的目标 IP 地址。一旦接收到有效的 ping 包，协议栈将从 “配置”中退出并进入到常规模式中。

在配置期间，用户应用程序不得尝试通讯。可以调用函数 StackIsInConfigMode() 来确定协议栈是否处于“配置”模式下。StackIsInConfigMode() == TRUE 表示协议栈处于 “配置”模式下。

要远程设置协议栈节点的IP地址，必须从远程机器发出一系列命令。对于这个示例，假定运行 Microchip 协议栈的节点需要分配 10.10.5.106 的 IP 地址。这个节点的 MAC 地址（十六进制）是 0a-0a-0a-0a-0a-0a。重新设置这个节点后，网络中的另一个系统（假设是运行 Microsoft Windows 的计算机）发出以下命令：

```
> arp -s 10.10.5.106 0a-0a-0a-0a-0a-0a
> ping 10.10.5.106
```

就会生成来自 10.10.5.106 的标准 ping 响应序列，表示已为该节点成功分配了该 IP 地址。

Stack Manager

如前面所述，Microchip TCP/IP 协议栈是不同模块的集合。接收到某种类型的包时必须调用相应的模块（例如 IP、TCP、UDP 和 ICMP）。使用 Microchip TCP/IP 协议栈的任何应用程序都必须执行一定的步骤，以确保在适当的时间调用这些模块。无论主应用程序逻辑如何，管理协议栈模块的任务都始终相同。

为了减轻主应用程序管理各个模块的负担， Microchip TCP/IP协议栈使用了一个特殊的应用程序层模块，称为“StackTask”或 Stack Manager。这个模块由源文件“StackTsk.c”来实现。“StackTask”以协同式任务处理方式来实现；被赋予处理时间后，它将轮询 MAC 层是否存在有效的数据包。如果接收到一个包，它将对其进行解码并路由到相应的模块进行下一步的处理。

请注意 Stack Manager *不是* Microchip TCP/IP 协议栈的一部分。它与协议栈一起提供给用户，这样主应用程序除了自己的工作以外，不需要管理协议栈模块。执行 Stack Manager任务之前，必须通过调用StackInit() 函数来初始化它。这个函数按照正确的顺序初始化 Stack Manager 变量和各个模块。一旦调用了 StackInit()，主程序就必须定期调用StackTask() 函数，以确保及时处理所有进入的包，以及所有的超时和错误条件。

StackTask 使用的具体步骤如例 2 中的算法所示。

例 2： STACK MANAGER 算法

```
If a data packet received then
   Get data packet protocol type
      If packet type is IP then
         Fetch IP header of packet
         Get IP packet type
         if IP packet type is ICMP then
            Call ICMP module
         else if IP packet type is TCP then
            Call TCP module
         else if IP packet type is UDP then
            Call UDP module
         else
            Handle not supported protocol
         End If
      End If
   Else if packet type is ARP then
      Call ARP module
   End If
End If
```

MICROCHIP HTTP 服务器

本应用笔记中包含的 HTTP 服务器以协同式任务处理方式实现，它与 Microchip TCP/IP 协议栈以及用户的主应用程序共存。这个服务器自身在源文件 “HTTP.c” 中实现，使用一个用户应用程序实现两个回调函数。演示应用程序源文件 “Websrvr.c” 用作模板应用程序来创建必需的接口。

这里提供的 HTTP 服务器不会实现所有的 HTTP 功能，它是面向嵌入式系统的最小型服务器。用户可以按需要方便地添加新功能。

HTTP 服务器包含以下主要功能：

- 支持多 HTTP 连接
- 包含简单文件系统 （MPFS）
- 支持位于内部程序存储器或外部串行 EEPROM 中的网页
- 包含基于 PC 的程序以从给定目录创建 MPFS 映象
- 支持 HTTP 方法 “GET”（可以很容易地添加其他方法）
- 支持改进的公共网关接口 （Common Gateway Interface， CGI）以从远程浏览器内部调用预定义的函数
- 支持动态网页内容生成

服务器包含以下主要组件：

- MPFS Image Builder
- MPFS Access Library
- MPFS Download Routine （由主应用程序实现）
- HTTP Server Task

为了将 HTTP 服务器集成到用户应用程序中，请执行以下操作：

1. 在头文件 “StackTsk.h” 中取消对 STACK_USE_HTTP_SERVER 的注释，从而使能与 HTTP 服务器相关的代码。
2. 在头文件 “StackTsk.h” 中设置所需的 MAX_HTTP_CONNECTIONS 值。
3. 将文件 “http.c” 和 “mpfs.c” 包含在项目中。
4. 根据网页存储的位置，取消对 MPFS_USE_PGRM 或 MPFS_USE_EEPROM 的注释。如果外部数据 EEPROM 用作存储介质，则还要包含文件 “xeeprom.c”。
5. 修改应用程序的 main() 函数以包含 HTTP 服务器 （见第 8 页的例 1 中的代码示例）。

还可能需要提前生成网页并将它们转换为兼容的格式进行存储。对于 Microchip 协议栈及其 HTTP 服务器，特定格式是 MPFS （有关详细信息，请参见第 83 页的 “Microchip 文件系统 （MPFS）” 一节）。

如果 MPFS 映像要存储在外部数据 EEPROM 中，则可能需要在应用程序中包含编程方法，特别是在内容会更改的情况下。有三个主要选项用于编程外部 EEPROM：

1. 使用编程器应用程序或数据 EEPROM 供应商提供的器件对 MPFS 映像进行编程。必须确保 MPFS 映像在 “保留块” 之后开始。
2. 在主应用程序中包含一个下载的程序，它可以从外部数据源 （例如，通过串行口连接的计算机） 接收数据并将其编程到 EEPROM 中。Microchip 协议栈源代码提供了一个示例程序 （见第 88 页的 “MPFS 下载演示例程”）。
3. 将 FTP 服务器模块包含到项目中，并使用 FTP 跨网络对 MPFS 映像进行远程编程。更多信息，请参见第 85 页的 “使用 FTP 客户机上传 MPFS 映像”。

HTTP 服务器使用文件 “index.htm” 作为它的缺省网页。如果远程客户机（浏览器）通过 HTTP 服务器的 IP 地址或域名来访问它，则 “index.htm” 是其缺省页面。这要求所有应用程序都将名为 “index.htm” 的文件作为它们 MPFS 映像的一部分。如果需要，通过更改 “http.c” 文件中编译器 “定义项” HTTP_DEFAULT_FILE_STRING 来修改这个缺省文件的名称。

要确保网页文件名称不要包含以下非字母数字字符：

- 单引号或双引号 （' 和 "）
- 左尖括号和右尖括号 （< 和 >）
- 井号 （#）
- 百分号 （%）
- 左右方括号或左右花括号 （[、{、] 和 }）
- “管道” 符号 （|）
- 反斜杠 （\）
- 插入标记 （^）
- 颚化符号 （~）

如果文件名包含这些字符中的任何一个，则相应的网页将不能被访问，且不会给出事先警告。

HTTP 服务器还有一个支持的文件类型列表。它使用这个信息来建议远程浏览器如何根据文件三个字母的扩展名来解析特定文件。缺省情况下，Microchip HTTP 服务器支持 “.txt”、“.htm”、“.gif”、“.cgi”、“.jpg”、“.cla” 和 “.wav” 文件。如果应用程序使用这个列表中不包含的文件类型，用户可以修改表 “httpFiles”，以及文件 “http.c” 中 “httpContents” 的内容。

动态 HTTP 页面生成

HTTP 服务器可以动态更改页面来发布实时信息，例如输入 / 输出状态。为包含实时信息，相应的 CGI 文件（*.cgi）必须包含文本字符串“%xx”，其中“%”字符是控制代码，“xx”代表两位变量标识符。变量值的范围是00-99。当HTTP服务器遇到这个文本字符串时，它将除去“%”字符并调用 HTTPGetVar 函数。如果页面需要“%”作为显示字符，则该字符前应该加上另一个“%”字符。例如，为了在页面上显示“23%”，则需输入“23%%”。

HTTPGetVar

这个函数是 HTTP 的回调函数。当 HTTP 服务器在它运行的 CGI 页面中遇到字符串“%xx”时，将调用此函数。此函数由主应用程序实现，用于将应用程序专用变量状态发送到 HTTP。

语法

```
WORD HTTPGetVar(BYTE var, WORD ref, BYTE *val)
```

参数

var [in]

要返回其状态的变量标识符

ref [in]

调用参考。这个参考值指示这是不是第一个调用。第一次调用后，这个值只限于由主应用程序维护。 HTTP 使用这个函数的返回值来确定是否再次调用此函数以获取更多数据。假设使用这个回调函数每次只传输一个字节，这个参考值允许主应用程序跟踪它的数据传输。如果变量状态要求更多字节，主应用程序可以将 *ref* 用作要返回的数据数组的索引。每次发送一个字节后，*ref* 的更新值作为返回值返回；同一个值被传递到下个回调。最后，当最后一个字节被发送后，应用程序必须返回 HTTP_END_OF_VAR 作为返回值。 HTTP 将一直调用这个函数，直到它接收到 HTTP_END_OF_VAR 作为返回值为止。

此参数的值可能为：

值	含义
HTTP_START_OF_VAR	这是当前实例的给定变量的第一个回调。如果需要多字节数据传输，则应该使用这个值有条件地初始化多字节数组的索引，这个数组的值将被发送到当前变量。
所有其他	主应用程序的特定值。

val [out]

要发送的单个字节数据

返回值

主应用程序确定的新参考值。如果这个值不是 HTTP_END_OF_VAR， HTTP 将使用上次调用的返回值再次调用此函数。

如果返回 HTTP_END_OF_VAR， HTTP 将不会调用此函数并假设变量值发送已经结束。

此参数的值可能为：

值	含义
HTTP_END_OF_VAR	这是给定变量的最后一个数据字节。 HTTP 不会调用此函数直到需要另一个变量值为止。
所有其他	主应用程序的特定值。

前提条件

无

HTTPGetVar（续）

副作用

无

备注

尽管这个函数从主应用程序请求一个变量值，但是主应用程序不一定非要返回值。实际变量值可以是字节数组，它可以是也可以不是变量值。要返回的信息完全取决于主应用程序以及关联的网页。例如，变量“50”可以表示 120 x 120 像素的 JPEG 帧。在这种情况下，主应用程序可以将参考用作对 JPEG 帧的索引，并每次返回一个字节给 HTTP。 HTTP 将继续调用这个函数，直到它接收到这个函数的返回值 HTTP_END_OF_VAR 为止。

假设这个函数的返回值为 16 位，则每个变量值最多可以传递 64KB 的数据。如果需要更多的数据，则可以将两个或更多变量并列放置来创建更大的数据传输数组。

示例 1

请看 HTTP 运行的页面“status.cgi”。

“status.cgi”包含以下 HTML 行：

```
…
<td>S3=%04</td><td>D6=%01</td><td>D5=%00</td>
…
```

处理这个文件时， HTTP 遇到“%04”字符串。对它进行解析后， HTTP 回调 HTTPGetVar(4, HTTP_START_OF_VAR, &value)。主应用程序按如下方式实现 HTTPGetVar：

```
WORD HTTPGetVar(BYTE var, WORD ref, BYTE *val)
{
        // 识别变量
        // 是 RB5 吗?
        if ( var == 4 )
        {
                // 如果 RB5 为高电平则返回 1
                // 如果 RB5 为低电平则返回 0
                if ( PORTBbits.RB5 )
                        *val = ‘1’;
                else
                        *val = ‘0;
                // 告诉 HTTP 这是变量值的最后一个字节
                return HTTP_END_OF_VAR;
        }
        else
        // 检查其他变量 ...
        ...
}
```

更多详细信息，请参见“Webpages*.cgi”文件以及“Websrvr.c”源文件中相应的回调部分。

HTTPGetVar（续）

示例 2

假设页面“status.cgi”需要显示 HTTP Web 服务器设备的序列号。

HTTP 运行的页面“status.cgi”包含以下 HTML 行：

…
<td>Serial Number=**%05**</td>
…

处理这个文件时，HTTP 遇到“%05”字符串。对它进行解析后，HTTP 回调 HTTPGetVar(4, HTTP_START_OF_VAR, &value)。主应用程序按如下方式实现 HTTPGetVar：

```
WORD HTTPGetVar(BYTE var, WORD ref, BYTE *val)
{
        // 识别变量
        // 是 RB5 吗？
        // 如果是，则处理 RB5 值（与示例 1 类似）
        // 它是“SerialNumber”变量吗？
         if ( var == 5 )
        {
                // 序列号是一个以 NULL 终止的字符串
                // 首先要确定这是否是第一个调用
                if ( ref == HTTP_START_OF_VAR )
                {
                        // 这是第一个调用。初始化到 SerialNumber 字符串的索引
                        // 我们将 ref 用作索引
                        ref = (BYTE)0;
                }
                // 现在访问当前索引处的字节并保存在缓冲区中
                *val = SerialNumberStr[(BYTE)ref];
                // 到达字符串结尾了吗？
                if ( *val == '\0' )
                {
                        // 是的，已经完成字符串传输
                        // 返回 HTTP_END_OF_VAR 通知 HTTP 服务器传输已完成
                        return HTTP_END_OF_VAR;
                }
                // 否则，阵列索引加 1，并将其返回给 HTTP 服务器
                (BYTE)ref++;
                // 由于 ref 的值不是 HTTP_END_OF_VAR，HTTP 服务器将再次调用以获取值的剩余部分
                return ref;
        else
        // 检查其他变量 ......
        ...
}
```

更多详细信息，请参见“Webpages*.cgi”文件以及“Websrvr.c”源文件中相应的回调部分。

HTTP CGI

HTTP 服务器实现了 CGI 的一个修订版本。使用这个接口，HTTP 客户机可以在 HTTP 内部调用函数并以网页的形式接收结果。远程客户机使用 HTML GET 方法以及多个参数来调用函数。更多信息，请参见 RFC1866（HTML 2.0 语言规范）。

当远程浏览器执行带有多个参数的 GET 方法时，HTTP 服务器对其进行解析并使用实际方法代码和它的参数来调用主应用程序。为了处理这个方法，主应用程序必须使用相应的代码来实现回调函数。

Microchip HTTP 服务器不执行 “URL 解码”。这意味着如果表格字段文本包含特定的非字母数字字符（例如 <、>、”、# 和 % 等），则实际参数值将包含 “%xx”（“xx”是 ASCII 字符的两位十六进值）而不是实际字符。例如，“<Name>”条目将返回 “%3CName%3C”。请参见第 77 页的 “Microchip HTTP 服务器”以获取字符的完整列表。

包含 HTML 表单的文件必须具有 “.cgi”的文件扩展名。

HTTPExecCmd

这个函数是 HTTP 的回调函数。当 HTTP 服务器接收到具有多个参数的 GET 方法时，它将调用该函数。这个回调函数由主应用程序实现。这个函数必须对给定的方法代码进行解码并采取相应的操作。这些操作包括提供要返回的新网页名称和 / 或执行 I/O 任务。

语法

```
void HTTPExecCmd(BYTE **argv, BYTE argc)
```

参数

argv [in]

命令字符串参数列表。第一个字符串 （*argv*[0]）代表表单操作，而剩下的 （*argv*[1..n]）是命令参数。

argc [in]

参数总数，包括表单操作。

返回值

主应用程序可能需要使用有效的网页名称来修改 argv[0] 并用作命令结果。

前提条件

无

副作用

无

备注

这是作为远程调用的结果， HTTP 对主应用程序的回调。可以对给定方法进行同时 （逐个）调用。主应用程序必须同时解析这些调用并进行相应的操作。

缺省情况下，参数数目 （或表单字段）以及参数字符串总长度 （或表单 URL 字符串）分别限制为 5 和 80。表单字段限制中包含表单操作名。如果应用程序需要多于四个字段的表单和 / 或 URL 字符串总长超过 80 个字符，则必须增加 MAX_HTTP_ARGS 和 MAX_HTML_CMD_LEN 的数值 （在 “http.c”中定义）。

HTTPExecCmd（续）

示例

请看 HTML 页面 “Power.cgi”，在远程浏览器上显示为：

```
<html>
<body><center>
<FORM METHOD=GET action=Power.cgi>
<table>
<tr><td>Power Level:</td>
<td><input type=text size=2 maxlength=1 name=P value=%07></td></tr>
<tr><td>Low Power Setting:</td>
<td><input type=text size=2 maxlength=1 name=L value=%08></td></tr>
<tr><td>High Power Setting:</td>
<td><input type=text size=2 maxlength=1 name=H value=%09></td></tr>
<tr><td><input type=submit name=B value=Apply></td></tr>
</table>
</form>
</body></html>
```

这个页面显示了一个表，它的标签显示在第一列中，文本框值显示在第二列中。第一行第一列的单元格包含字符串“Power Level”，第二列是用来显示并修改“Power Level”值的文本框。最后一行包含标有“Apply”的按钮。查看这个页面的用户可以修改“Power Level”文本框中的值，并单击“Apply”按钮将新值提交给 Microchip 协议栈。

假设用户在“Power Level”、“Low Power Setting”和“High Power Setting”文本框中分别输入“5”、“1”和“9”，接着单击“Apply”按钮。浏览器将创建 HTTP 请求字符串“Power.cgi?P=5&L=1&H=9”并将其发送到 HTTP 服务器。接着服务器使用以下参数调用 HTTPExecCmd：

argv[0] = “Power.cgi”, *argv[1]* = “P”, *argv[2]* = “5”, *argv[3]*=“L”, *argv[4]*=“1”, *argv[5]*=“H”, *argv[6]*=“9”
argc = 7

主应用程序按如下方式实现 HTTPExecCmd：

```
void HTTPExecCmd(BYTE *argv, BYTE argc)
{
        BYTE i;
        // 检查用于当前表单命令的每个参数
        // 我们跳过表单操作名，所以 i 从 1 开始 ...
        for (i = 1; i < argc; i++ )
        {
                // 识别参数
                if ( argv[i][0] == ‘P’ )        // 这是 “power level” 吗?
                {
                        // 下一个参数是 “Power level” 值
                        PowerLevel = atoi(argv[++i]);
                }
                else if ( argv[i][0] == ‘L’ )   // 这是 “Low Power Setting” 吗?
                        LowPowerSetting = atoi(argv[++i]);
                else if ( argv[i][0] == ‘H’ )   // 这是 “High Power Setting” 吗?
                        HighPowerSetting = atoi(argv[++i]);
        }
        // 如果有其他页要显示为该命令的结果，复制其大写名称至 argv[0]
        // strcpy(argv[0], “RESULTS.CGI”);
}
```

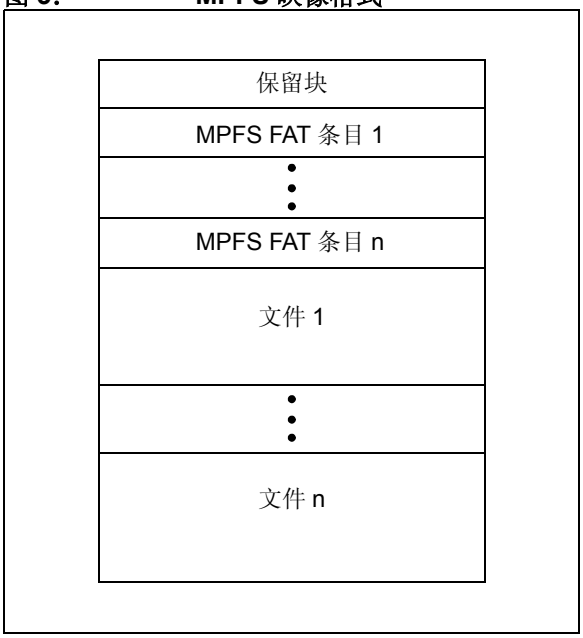
注： 对于这个示例，参数数目超过了缺省的上限值 5。为了让这个示例能正确运行， MAX_HTTP_ARGS （位于“http.c”中）的值必须至少置为 7。

更多详细信息，请参见“Webpages*.cgi”文件以及“Websrvr.c”源文件中相应的回调部分。

MICROCHIP 文件系统 （MPFS）

Microchip HTTP 服务器使用简单文件系统 （Microchip 文件系统或 “MPFS”）来存储网页。 MPFS 映像可以存储在片上程序存储器或外部串行EEPROM中。MPFS 采用特殊的格式在给定的存储介质上存储多个文件，如图 3 所示。

图 3： MPFS 映像格式



“保留块”的长度由 MPFS_RESERVE_BLOCK 定义。主应用程序可以使用保留块来存储简单配置值。MPFS 以一个或多个 MPFS FAT（文件分配表）条目开始，后跟一个或多个文件数据。FAT 条目描述文件的名称、位置以及状态。图 4 中显示了 FAT 条目的格式。

图 4： MPFS FAT 条目格式

标志 （8 位）	地址 （16 或 24 位）	文件名 （8 字节 + 3 字节格式）

*标志*表示当前条目正在使用、已被删除或在 FAT 的结尾。

每个 FAT 条目都包含一个 16 位或 24 位地址值。地址长度由所使用的存储器类型以及存储器容量模型确定。如果使用内部程序存储器，并且使用小型存储器模型来编译 Microchip TCP/IP 协议栈项目，则使用 16 位地址。如果使用内部程序存储器，并且选择大型存储器模型，则使用 24 位地址。不管使用哪种存储器模型，总是将 16 位寻址机制用于外部 EEPROM 器件。所以对于这些器件来讲， MPFS 映像最大为 64KB。

MPFS 使用 “8 + 3” 格式（8 个字节用于实际文件名，3 个字节用于扩展名，或 NNNNNNNN.EEE）的 “短” 文件名。16 位地址给出第一个文件数据块的开始地址。所有文件名都用大写字母存储，以方便文件名的比较。

每个 FAT 条目中的地址指向包含实际文件数据的数据块。图 5 中显示了数据块格式。数据块以名为 EOF（End of File）的特殊 8 位标志结束，后跟 FFFFh （用于 16 位寻址）或 FFFFFFh （24 位寻址）。如果块的数据部分包含 EOF 字符，则用特殊转义字符 DLE （Data Link Escape）来填充。任何 DLE 字符也必须用 DLE 来填充。

图 5： MPFS 数据块格式

数据 （可变长度）	EOF （8 位）	FFFFh 或 FFFFFFh （16 或 24 位）

MPFS Image Builder

本应用笔记包含一个特殊的、基于 PC 的程序（MPFS.exe），使用该程序可以利用一组文件构建 MPFS 映像。取决于 MPFS 最终存储在哪里，用户可以选择生成“C”数据文件或二进制文件来表示 MPFS 映像。

MPFS.exe 的完整命令行语法是：

```
mpfs [/?] [/c] [/b] [/r<Block>]
<InputDir> <OutputFile>
```

其中

/? 显示命令行帮助

/c 生成“C”数据文件

/b 生成二进制数据文件（缺省输出）

/r 在文件的开始保留一块内存区域（只有在“二进制输出”模式下才有效，缺省值为 32 个字节）

<InputDir> 是包含用于创建映像的文件的目录

<OutputFile> 是输出文件名

例如，命令

```
mpfs /c <Your WebPage Dir> mypages.c
```

利用目录“Your Web Pages”的内容生成的 MPFS 映像为“C”数据文件 mypages.c。相比之下，命令

```
mpfs <Your WebPage Dir> mypages.bin
```

生成的是带有 32 个字节保留块的二进制映像文件（二进制格式和 32 个字节保留块都是缺省的），而

```
mpfs /r128 <Your WebPage Dir> mypages.bin
```

生成同样的二进制文件，只是保留块大小为 128 个字节。

注：	使用不是 32 个字节的缺省保留块大小需要对头文件“StackTsk.h”中的编译器 define MPFS_RESERVE_BLOCK 进行更改。

在创建 MPFS 映像之前，用户必须先创建所有的网页和相关文件，并保存在同一个目录中。如果文件扩展名是“htm”，则 Image Builder 将去掉所有回车和换行字符来减少 MPFS 映像的大小。

如果 MPFS 映像要存储在内部程序存储器中，则生成的“C”数据文件必须与“Websrvr”项目链接在一起。如果映像要存储在外部串行 EEPROM 中，则必须把二进制文件装载到串行 EEPROM 中。更多信息，请参见第 85 页的“Microchip FTP 服务器”。

MPFS Image Builder 不会检查文件大小。如果选择二进制数据格式，需验证映像大小不会超过可用的 MPFS 存储空间。

MPFS Access Library

源文件“MPFS.c”实现了访问 MPFS 映像文件所需的例程。用户使用 HTTP 时不需要了解 MPFS 的细节。对 MPFS 的所有访问由 HTTP 执行，不需要主应用程序的参与。

MPFS 库的当前版本不允许对已有的映像文件集合添加或删除文件，即组成单个 MPFS 映像的所有文件必须一次添加完成。任何更改都需要重新创建新的映像文件。

如果使用内部程序存储器存储 MPFS，则必须定义 MPFS_USE_PGRM。同样，如果外部数据 EEPROM 用于 MPFS 存储，则必须定义 MPFS_USE_EEPROM。这些定义只有一个可以出现在“StackTsk.h”中，编译时检查确保二者只选其一。

根据使用的存储器件类型，它的页面缓冲区大小会不一样。页面缓冲区大小（由头文件“MPFS.h”中的 MPFS_WRITE_PAGE_SIZE 定义）的缺省设置是 64 个字节。如果需要不同的缓冲区大小，则相应地更改 MPFS_WRITE_PAGE_SIZE 的值。

注：	这个版本的 MPFS Access Library 使用文件“xeeprom.c”对外部数据 EEPROM 进行访问。对文件进行读或写时，MPFS 独占 I^2C™ 总线，不允许其他任何 I^2C 从动器件或主控器件进行通信。使用多个 I^2C 器件的用户需要记住这一点。

MICROCHIP FTP 服务器

本应用笔记中包含的FTP服务器以协同式任务处理方式实现，它与 Microchip TCP/IP 协议栈以及用户的主应用程序共存。服务器自身在源文件 “FTP.c”中实现。

这里提供的 FTP 服务器没有实现 RFC 959 中指定的所有功能；它是面向嵌入式系统的最小型服务器。它包括下列主要功能：

- 一个 FTP 连接，由用户应用程序认证
- 与文件系统 （MPFS）的自动交互
- 使用一个 “put”命令对整个 MPFS 映像进行远程编程
- 不支持单个文件的上传或检索

用户可以按需要添加新功能。

服务器实际上由两个组件组成：FTP 服务器自身以及用户应用程序实现的 FTP 认证回调。用户必须用 FTPVerify 函数实现回调，该函数验证 FTP 用户名和密码。（更多详细信息，请参见下一页的 “FTPVerify”函数。）演示应用程序源文件 “Websrvr.c”应该用作模板应用程序，用来创建必需的接口。有关如何将 FTP 服务器集成到用户应用程序中的实例，请参见 “演示应用程序”。

为了将 FTP 服务器集成到用户应用程序中，请执行以下操作：

1. 在 “StackTsk.h”头文件中取消对 STACK_USE_FTP_SERVER 的注释。
2. 将已定义的套接字数目加 2（具体值取决于已有的数目）。
3. 将文件 “FTP.c”和 “mpfs.c”包含在项目中。
4. 根据网页存储的位置，取消对 MPFS_USE_PGRM 或 MPFS_USE_EEPROM 的注释。如果外部数据 EEPROM 用作存储介质，则将 “xeeprom.c”包含在项目中。
5. 修改应用程序的 main() 函数以包含 FTP 服务器（见第 8 页的例 1 中的代码示例）。

Microchip FTP 服务器一次只允许一个 FTP 连接。每个 FTP 连接需要两个 TCP 套接字。

FTP 服务器的上传和下载的缺省超时时间限制为约 180 秒。如果远程 FTP 连接保持空闲状态超过 180 秒，它将自动断开连接。这保证孤儿连接或文件上传期间发生的问题不会一直占用 FTP 服务器。

使用 FTP 客户机上传 MPFS 映像

Microchip 协议栈中的 FTP 服务器的主要用途是远程升级 MPFS 映像。当前版本只能使用外部数据 EEPROM 存储 MPFS，如果选择 MPFS_USE_PGRM 选项，则该版本无法正常使用。

用户和运行带有 FTP 服务器的 Microchip 协议栈的节点之间进行数据交换的典型示例如例 3 所示。在这个实例中，一个 MPFS 映像从计算机上传到了节点。为了方便举例说明，这是用户从运行 Microsoft Windows 的计算机命令窗口看到的内容；其他系统和终端的显示可能会稍微不同。具体的 FTP 用户名和密码取决于用户应用程序； Web Server 演示应用程序 （第 87 页）使用的是例子中的值。FTP 客户机操作（即：来自用户的手动输入）用**粗体**显示。系统提示和 FTP 服务器响应用常规字体显示。

例 3： 使用 FTP 上传 MPFS 映像

```
c:\ ftp 10.10.5.15
220 ready
User (10.10.5.15:(none)):ftp
331 Password required
Password:microchip
230 Logged in
ftp> put mpfsimg.bin
200 ok
150 Transferring data...
226 Transfer Complete
ftp> 16212 bytes transferred in
0.01Seconds 16212000.00Kbytes/sec.
ftp> quit
221 Bye
```

注： 用户输入密码时 FTP 服务器不会回显。在以上示例中，显示密码是为了说明用户输入的内容。

FTPVerify

这个函数是 FTP 服务器的回调函数。当服务器接收到来自远程 FTP 客户机的连接请求时调用此函数。此函数由主用户应用程序实现，用于验证远程 FTP 用户。

语法

```
BOOL FTPVerify(char *login, char *password)
```

参数

login [in]

包含用户名的字符串

password [in]

包含密码的字符串

返回值

TRUE： 如果 *login* 和 *password* 与用户应用程序中定义的登录和密码变量值匹配

FALSE： 如果 *login* 或 *password* 不匹配

FTP 服务器使用此函数的返回值允许或不允许远程 FTP 用户的访问。

前提条件

无

副作用

无

备注

用户名长度可以从 0 到 9。如果需要更长的用户名，在头文件 “ftp.h” 中修改 FTP_USER_NAME_LEN 的值。

最长密码字符串长度和 FTP 命令总长度被预定义为 31。如果需要更长的密码和 / 或命令，则修改 “FTP.c” 中 MAX_FTP_CMD_STRING_LEN 的值。

示例

此示例说明如何实现 FTP 回调。

```
ROM char FTPUserName[] = “ftp”;
#define FTP_USER_NAME_LEN   (sizeof(FTP_USER_NAME)-1)
ROM char FTPPassword[] = “microchip”;
#define FTP_USER_PASS_LEN   (sizeof(FTP_USER_PASS)-1)

BOOL FTPVerify(char *login, char *password)
{
        if ( !memcmppgm2ram(FTP_USER_NAME, login, FTP_USER_NAME_LEN) )
        {
                if ( !memcmppgm2ram(FTP_USER_PASS, password, FTP_USER_PASS_LEN) )
                        return TRUE;
        }
        return FALSE;
}
```

更多详细信息，请参见 “Webpages*.cgi” 文件以及 “Websrvr.c” 源文件中相应的回调部分。

演示应用程序

Microchip TCP/IP 协议栈附带了一个完整的应用程序实例，用来演示所有 TCP/IP 模块。这个应用程序（“Web Server”）被设计成在 Microchip 的 PICDEM.net™ 因特网/以太网演示板上运行，其主源文件是“websrvr.c”。用户应该参考这些源代码来创建自己的应用程序和学习了解 Microchip TCP/IP 协议栈。Microchip 协议栈包含的项目示例说明了该应用程序示例可以运行的不同配置。

配置 PICDEM.net 演示板和 Web Server

必须有 HEX 代码文件才能运行演示应用程序。一个选择是使用编译器和硬件配置创建一个项目示例；另一个选择是，为相应的项目使用已经创建的 HEX 文件（有关 Microchip 协议栈包含的项目列表，请参见第 5 页的表 2）。对单片机进行编程时请遵循器件编程器的标准过程。确保设置了以下配置选项：

- 振荡器：HS
- 看门狗定时器：禁止
- 低电压编程：禁止

将已编程的单片机安装在 PICDEM.net 演示板上并上电时，系统 LED 将闪烁表示应用程序正在运行。 LCD 显示屏将在第一行显示

MCHPStack v2.0

（取决于应用程序的发行级别，这个版本号可能不同），在第二行显示配置消息或 IP 地址。

编程以后，将演示应用程序用于实际网络之前仍然需要对其进行正确配置。以下操作专用于 Microsoft Windows 和 “超级终端”仿真器，如果使用不同的操作系统或终端软件，则过程可能不同。

1. 如上所述对 PIC18 单片机进行编程，并将其安装在 PICDEM.net 演示板上。
2. 使用标准 RS-232 电缆，将 PICDEM.net 演示板连接到计算机可用的串行口。
3. 启动 “超级终端”（*开始 > 程序 > 附件*）。
4. 在 “连接属性”对话框中，为连接输入任何方便的名称。单击 “确定”。
5. 在 “连接到”对话框，选择连接 PICDEM.net 演示板的 COM 端口。单击 “确定”。
6. 配置连接到 PICDEM.net 演示板的串行口：
   - 19200 bps
   - 8 个数据位、 1 个停止位且无奇偶校验
   - 无流量控制

   单击 “确定”启动连接。
7. 按住 S3 开关给演示板上电，或按下 RESET 和 S3 开关并保持，接着松开 RESET 开关。LCD 显示器显示如下消息

   MCHPStack v2.0

   Board Setup...

   （取决于应用程序的发行级别，这个版本号可能不同。）松开 S3。

   配置菜单出现在终端窗口中：

   ```
   MCHPStack Demo Application v1.0
   (Microchip TCP/IP Stack 2.0)
   1: Change board Serial number.
   2: Change default IP address.
   3: Change default gateway address.
   4: Change default subnet mask.
   5: Enable DHCP and IP Gleaning.
   6: Disable DHCP and IP Gleaning.
   7: Download MPFS image.
   8: Save & Quit.
   Enter a menu choice (1-8):
   ```
8. 选择每个要配置的项并输入新值 （通常是第 2、3 和 4 项）。选择第 8 项保存并退出配置，新地址已保存到数据 EEPROM 中。应用程序退出配置模式并运行 Web Server。

连接到以太网

运行 Web Server 演示应用程序时， PICDEM.net 演示板可以不经过任何修改直接连接到以太网网络。当然，IP 配置必须与该网络兼容。缺省情况下，Web Server 应用程序使用下列值进行配置：

- IP 地址：10.10.5.15
- 网关地址：10.10.5.15
- 子网掩码：255.255.255.0

即使 IP 地址是兼容的，网关和掩码也可能不兼容。如果需要进行更改，有许多方法可行。

使用 DHCP 自动配置

如果网络使用 DHCP 配置，则不需要其他的改动。当演示板连接到网络中并上电后，DHCP 服务器将分配给它一个 IP 配置。在这个过程中，LCD 显示屏显示以下消息

DHCP/Gleaning...

几秒钟后，显示屏显示分配的 IP 地址，例如：

100.100.100.1 1

实际 IP 地址放在演示板的指定地址中。最右边的数字指示 DHCP 租用更新的次数，供参考。

根据网络是如何配置的，PICDEM.net 演示板的 IP 地址可能在长时间掉电后发生更改（例如，演示板的 DHCP 租用已经到期，旧的 IP 地址可能被另一个设备占用）。总是使用当前显示的 IP 地址与演示板通信。

预配置的网络配置

某些网络可能是“硬配置”的，即每个设备的地址由网络管理员手动分配。在这种情况下， PICDEM.net 演示板在连接到网络之前必须使用管理员提供的IP配置手动进行配置。详细信息，请回溯到第 87 页的“配置 PICDEM.net 演示板和 Web Server”。

使用 IP GLEANING 设置 IP 地址

如果已经将演示板连接到网络，唯一需要做的是更改 IP 地址，则可以使用 IP Gleaning（第 76 页）。已经提到，这个方法可用于配置IP地址，但是不能配置网关或子网掩码。

为了使用 IP gleaning，必须知道设备的 MAC 地址。它通常是 6 个字节的十六进制数字，格式为“xx-xx-xx-xx-xx-xx”。对于PICDEM.net演示板，MAC 通常是 00-04-A3-00-nn-nn，其中“nn-nn”是十六进制的演示板的序列号。因此，序列号为 1234（或 04D2h）的演示板的 MAC 地址为 00-04-A3-00-04-D2。

一旦得到了设备的 MAC 地址和新 IP 地址，先复位设备，接着从远程终端发出 arp 和 ping 命令，就可以设置新的 IP 地址了。继续前面的示例，如果希望给前面提到的演示板分配 10.10.5.50 的新 IP 地址，我们将发出命令：

```
> arp -s 10.10.5.50 00-04-a3-00-04-d2
> ping 10.10.5.50
```

成功的 ping 响应表明 IP 地址已经发生更改。

MPFS 下载演示例程

如果要将 MPFS 映像存储在外部串行 EEPROM 中，则必须使用 MPFS 映像预编程或从另一个应用程序下载。Web Server 演示程序实现了一个简单的 MPFS 下载例程，它从 Xmodem 协议的终端仿真器接收 MPFS 二进制文件。

要下载 MPFS 格式的二进制文件：

1. 如果还未下载，则配置 PICDEM.net 演示板（请参见“配置 PICDEM.net 演示板和 Web Server”中的步骤 1 到 7）。
2. 在配置菜单，输入 “7”开始 MPFS 下载。应该看到 “Ready to download...”消息。此时，应该看到左边的 User LED （D6）大约每秒闪烁两次。
3. 从 “超级终端”的 “传送”菜单，选择 “发送文件”。在 “发送文件”对话框中，浏览至包含文件 “mpfsimg.bin”的目录，选中该文件。选择 “Xmodem”作为协议。
4. 单击“发送”。数据发送应自动启动。User LED 按照从计算机接收数据的速度闪烁。
5. 文件发送完成后，按 “8”退出配置模式。

Web Server 应用程序现正在运行，HTTP 服务器准备运行刚装入的页面。

演示 HTTP 服务器

正确配置并提供 MPFS 映像后，演示 HTTP 服务器（第 77 页）将运行网页。 Microchip 协议栈源文件包含的示例页面说明了 CGI 的修改方法 （远程方法调用）和动态页面生成方法。

演示 FTP 服务器

对这个演示应用程序的详细讨论从第 85 页开始。除此之外，它允许跨网络远程对 MPFS 映像进行更新。缺省配置使用 “ftp”作为 FTP 用户名，“microchip”作为密码。这些在 Web Server 演示应用程序中定义。

WINDOWS 95/98 系统的 SLIP 配置

任何运行 32 位版本的 Microsoft Windows（即 Windows 95 或更高版本）的个人计算机都可以配置为使用 SLIP 连接进行网络通讯服务。这个部分列出为桌面系统配置 SLIP 的步骤，并创建一个预配置的 SLIP 连接。

创建连接分两步完成：1）在可用 COM 端口上创建虚拟调制解调器，接着 2）为该设备定义拨号连接类型。

这里描述的步骤适用于 Windows 95 （除了最初发行版以外的所有版本）和 98 ；取决于操作系统和版本，某些步骤可能会不同。简单起见， Windows NT 和 Windows 2000 Professional Desktop 的配置过程被省略了。感兴趣的用户在这些操作系统中创建 SLIP 连接时可以参见Microsoft文档以获取更多详细信息，这些步骤可作为参考。

要创建虚拟调制解调器：

1. 在“控制面板”（*开始 > 设置 > 控制面板*）窗口中，双击“调制解调器”图标。
2. 如果系统中没有安装调制解调器，则出现提示对话框；选中相应的选项以防止 Windows 搜索该设备。单击“下一步”。
   如果已经安装了调制解调器，则出现“调制解调器属性”页。单击“添加”按钮。在后继对话框上，手动选择设备；不要让 Windows 搜索设备或自动选择驱动程序。
3. 在设备选择对话框上，从“制造商”下拉列表中选择“标准调制解调器类型”；从“型号”列表选择任何一种标准调制解调器（最好是 19200 bps 或更快）。单击“下一步”。
4. 在后续对话框中，选择相应的 COM 端口和速度（如果要求）。单击“下一步”。
5. 在安装提示信息和短暂的延迟后，将出现一条成功安装的消息。单击“完成”。

一旦在相应端口上创建了设备，就可以为它配置 SLIP。

1. 启动“拨号连接向导”（*开始 > 设置 > 拨号连接 > 添加连接*）。
2. 在第一个“创建新连接”对话框，选中新创建的标准调制解调器。单击“下一步”。
3. 在下个对话框中询问电话号码时，输入任意的电话号码（不会使用该号码）。使用缺省区域代码和位置。单击“下一步”。
4. 单击“完成”可创建下个连接。在“拨号网络”文件夹中将出现一个新图标。
5. 右键单击新连接图标并从菜单中选择“属性”。
6. 在“连接属性”页，选择“常规”选项卡。在“连接使用”下拉菜单中确认标准（虚拟）调制解调器已被选中。单击“配置”按钮。
7. 在配置属性页，将波特率设置为 38,400。选择“连接”选项卡，请确认通讯设置是 8 个数据位、1 个停止位并且没有奇偶校验。单击“高级”按钮。
8. 取消选择“用户流量控制”复选框。单击“确定”关闭对话框，接着单击“确定”关闭配置属性页（现在您已经回到“连接属性”页）。
9. 选择“服务器类型”选项卡。从“拨号服务器类型”下拉列表选择“SLIP——Unix 连接”。除了底部的复选框“TCP/IP”以外取消所有复选框选项。单击“TCP/IP 设置”按钮。
10. 输入“10.10.5.1”的 IP 地址。取消选中“IP 头压缩”以及“在远程网络上使用缺省网关”选项。单击“确定”返回“属性”页。

> **注** **1：**这个配置中使用的地址是用于主机桌面系统的；不是目标系统的地址。 IP 地址 10.10.5.1 是特别针对 PICDEM.net 演示板的。
>
> **2：**主机系统和目标必须在同一个子网中。

11. 选择“脚本”选项卡，接着单击“浏览”按钮找到文件“`empty.scp`”驻留的目录（包含在 Microchip TCP/IP 协议栈归档文件中）。选择该文件并单击“确定”。
12. 单击“确定”完成操作。

双击该图标即可使用 SLIP 连接。可使用 Web 浏览器和任意 IP 应用程序通过该连接进行通讯。

存储器使用

用于Microchip TCP/IP协议栈的存储器总使用量取决于所使用的编译器和模块。出版此文档时（2002年6月），使用Hi-Tech PICC 18™ V8.11PL1编译器时各种协议栈模块的典型存储器使用状况如表 4 所示。

表 4： 使用 HI-TECH PICC 18 编译器时各种协议栈模块的存储器使用状况

模块	程序存储器（字）	数据存储器（字节）
MAC （以太网）	906	5(1)
SLIP	780	12(2)
ARP	392	0
ARPTask	181	11
IP	396	2
ICMP	318	0
TCP	3323	42
HTTP	1441	10
FTP 服务器	1063	35
DHCP 客户机	1228	26
IP Gleaning	20	1
MPFS(3)	304	0
Stack Manager	334(4)	12+ICMP 缓冲区

注 1： 与 RTL8019AS NIC 一起实现。
2： 不包含发送和接收缓冲区的大小，它们是用户定义的。
3： 内部程序存储器存储。
4： 最大值。实际大小可能会不同。

结论

Microchip TCP/IP 协议栈不是第一个用于 Microchip PIC18 系列单片机的此类应用程序。它提供了一个节约空间且模块化的 TCP/IP 版本，在小引脚数器件中实现了协同式多任务处理 （不使用操作系统）。由于其极强的适应性和免费提供的优点，在开发需要网络连接的嵌入式控制应用时将极为有用。

附录 A： 源代码

Microchip TCP/IP 协议栈的完整源代码，包括演示应用程序以及必需的支持文件，是免费提供的。作为单个归档文件，可以从 Microchip 公司网站下载，该站点位于

www.microchip.com

下载文档后，查看文件 “version.log”以获取当前版本以及软件的更新历史。

附录 B： 部分 RFC 文档列表

RFC 文档	说明
RFC 826	以太网地址解析协议 （ARP）
RFC 791	网际协议 （IP）
RFC 792	网际控制消息协议 （ICMP）
RFC 793	传输控制协议 （TCP）
RFC 768	用户数据报协议 （UDP）
RFC 821	简单邮件传输协议 （Simple Mail Transfer Protocol， SMTP）
RFC 1055	串行线路网际协议 （SLIP）
RFC 1866	超文本标记语言 （Hypertext Markup Language） （HTML 2.0）
RFC 2616	超文本传输协议 （HTTP） 1.1
RFC 1541	动态主机配置协议 （DHCP）
RFC 1533	DHCP 选项
RFC 959	文件传输协议 （FTP）

因特网 RFC 以及相关文档的完整列表可以在许多因特网网站获得。感兴趣的读者可以参见 **www.faqs.org/rfcs** 和 **www.rfc-editor.org**。

注：

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术指标。
- Microchip 确信：在正常使用的情况下， Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿与那些注重代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是 “牢不可破”的。

代码保护功能处于持续发展中。 Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了《数字器件千年版权法案（Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Microchip 位于美国亚利桑那州 Chandler 和 Tempe、位于俄勒冈州 Gresham 及位于加利福尼亚州 Mountain View 的全球总部、设计中心和晶圆生产厂均于通过了 ISO/TS-16949:2002 认证。公司在 PICmicro® 8 位单片机、 KEELOQ® 跳码器件、串行 EEPROM、单片机外设、非易失性存储器和模拟产品方面的质量体系流程均符合 ISO/TS-16949:2002。此外， Microchip 在开发系统的设计和生产方面的质量体系也已通过了 ISO 9001:2000 认证。

全球销售及服务网点

美洲

公司总部 Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 1-480-792-7200
Fax: 1-480-792-7277
技术支持：
http://support.microchip.com
网址：www.microchip.com

亚太总部 Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Habour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

亚特兰大 Atlanta
Alpharetta, GA
Tel: 1-770-640-0034
Fax: 1-770-640-0307

波士顿 Boston
Westborough, MA
Tel: 1-774-760-0087
Fax: 1-774-760-0088

芝加哥 Chicago
Itasca, IL
Tel: 1-630-285-0071
Fax: 1-630-285-0075

达拉斯 Dallas
Addison, TX
Tel: 1-972-818-7423
Fax: 1-972-818-2924

底特律 Detroit
Farmington Hills, MI
Tel: 1-248-538-2250
Fax: 1-248-538-2260

科科莫 Kokomo
Kokomo, IN
Tel: 1-765-864-8360
Fax: 1-765-864-8387

洛杉矶 Los Angeles
Mission Viejo, CA
Tel: 1-949-462-9523
Fax: 1-949-462-9608

圣何塞 San Jose
Mountain View, CA
Tel: 1-650-215-1444
Fax: 1-650-961-0286

加拿大多伦多 Toronto
Mississauga, Ontario,
Canada
Tel: 1-905-673-0699
Fax: 1-905-673-6509

亚太地区

中国 – 北京
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

中国 – 成都
Tel: 86-28-8676-6200
Fax: 86-28-8676-6599

中国 – 福州
Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

中国 – 香港特别行政区
Tel: 852-2401-1200
Fax: 852-2401-3431

中国 – 青岛
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

中国 – 上海
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

中国 – 沈阳
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

中国 – 深圳
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

中国 – 顺德
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

中国 – 武汉
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

中国 – 西安
Tel: 86-29-8833-7250
Fax: 86-29-8833-7256

台湾地区 – 高雄
Tel: 886-7-536-4818
Fax: 886-7-536-4803

台湾地区 – 台北
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

台湾地区 – 新竹
Tel: 886-3-572-9526
Fax: 886-3-572-6459

亚太地区

澳大利亚 Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

印度 India - Bangalore
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

印度 India - New Delhi
Tel: 91-11-5160-8631
Fax: 91-11-5160-8632

印度 India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

日本 Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

韩国 Korea - Gumi
Tel: 82-54-473-4301
Fax: 82-54-473-4302

韩国 Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 或
82-2-558-5934

马来西亚 Malaysia - Penang
Tel: 60-4-646-8870
Fax: 60-4-646-5086

菲律宾 Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

新加坡 Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

泰国 Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

欧洲

奥地利 Austria - Wels
Tel: 43-7242-2244-3910
Fax: 43-7242-2244-393

丹麦 Denmark-Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

法国 France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

德国 Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

意大利 Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

荷兰 Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

西班牙 Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

英国 UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

06/08/06