第15章 基于NE64的嵌入式以太网

以太网(Ethernet) 最早由Xerox研发，经DEC公司和Intel公司联合扩展，于1982年公布的一种基带局域网规范。以太网技术IEEE802.3规范就是以这个规范为基础制定的。按ISO开放互联参考模型的分层结构，以太网规范只包括通信模型中的物理层和数据链路层。而大家所俗称的以太网技术不仅包括了上述以太网规范，而且包括了TCP/IP协议组。有时甚至把应用层的简单邮件传送协议SMTP、域名服务DNS、文件传输协议FTP等的应用协议都与以太网这个名词捆绑在一起。早期的以太网只有10Mbps的吞吐量，现在已经发展成为100Mbps、1000Mbps的以太网。目前以太网的传输介质主要有双绞线、同轴电缆和光纤，其拓扑结构可以分为三种：总线拓扑、星形拓扑和交换式拓扑。

本章节的内容采用自底向上的方法逐层介绍。其中链路层以下的内容与芯片相关（包括链路层），涉及到具体的寄存器的设置；网络层以上的为通用的与芯片无关部分，理论上可以运行在任何支持网络的体系结构上。

15.1 网络分层简介

15.1.1 协议模型

计算机网络是将在地理上分散的、具有独立功能的多台计算机通过通信媒体连接在一起，实现相互之间的通信和信息交换，并配以相应的网络软件，实现资源共享(包括硬件资源和软件资源)、数据传输，提高计算机的可靠性和可用性等为目的。为了方便和清晰地研究复杂的计算机网络，针对计算机网络所执行的各种功能，国际组织提出了“网络系统层次模型”的概念，下面将简要介绍基于该概念的两种网络参考模型。

1.OSI参考模型

在1983年国际标准化组织(International Standards Organization，ISO)颁布的“开放系统互连(Open Systems Interconnection，OSI)”参考模型中，将整个网络通信功能划分为七个层次，从上至下依次是应用层、表示层、会话层、传输层、网络层、数据链路层和物理层。参考模型结构如图15-1所示，各层简明功能如表15-1所示。

在国际电子电器工程师协会(The Institute of Electrical and Electronic Engineer，IEEE)定义的IEEE 802.3 协议中，将图15-1中的数据链路层划分为逻辑链路控制(Logic Link Control，LLC)和介质访问控制(Medium Access Control，MAC)。这两层的功能划分如下：

逻辑链路控制：负责上层的网络在使用网络服务时不必去考虑其下层是如何设置的，它会启动控制信号的交互、组合数据的流通、解释命令、发出响应，并执行错误控制及恢复的功能。

介质访问控制：负责定义出许多种不同的访问控制方法，以决定如何控制实际传送介质的使用。

2.TCP/IP协议模型

TCP/IP的层次比OSI参考模型的七层要少，由四个层组成：应用层、传输层、网络层及数据链路层。与OSI参考模型对照，它的应用层对应于OSI参考模型的会话层、表示层以及应用层；数据链路层对应于OSI的数据链路层和物理层；其他层为一一对应的关系，如图15-1所示。

表 15-1 网络模型的各层简明功能

OSI模型	TCP/IP模型	功能
应用层	应用层	实现网络虚拟终端的功能以及实现用户终端功能之间的映射，提供FTP、HTTP、SMTP等功能
表示层		用标准编码方式对数据进行编码，对该数据结构进行定义，并管理这些数据
会话层		把要求建立会话的用户地址，转换成相应的传送开始地址，以实现正确的传送连接
传输层	传输层	接收从会话层发出的数据，根据需要把数据划分为许多很小的单元（即报文），传送给网络层
网络层	网络层	(1) 处理来自传输层的分组发送请求，收到请求后，将分组装入IP数据报，填充报头，选择去往信宿机的路径，然后将数据报发往适当的网络接口； (2) 处理输入数据报，检查其合法性，然后再进行寻径；处理ICMP报文，处理路径、流控、拥塞问题
数据链路层	数据链路层	(1)将待发送的数据封装在多个数据帧里，并顺序地发送每一帧，同时处理接收方回送的确认帧； (2)该层通过在帧头和帧尾附加上特殊的二进制编码产生和识别帧界
物理层		按照传送介质的电气机械特性传送数据，传送单位主要以Bit为单位，并将信息按位逐一从一个系统经物理通道送往另一个系统

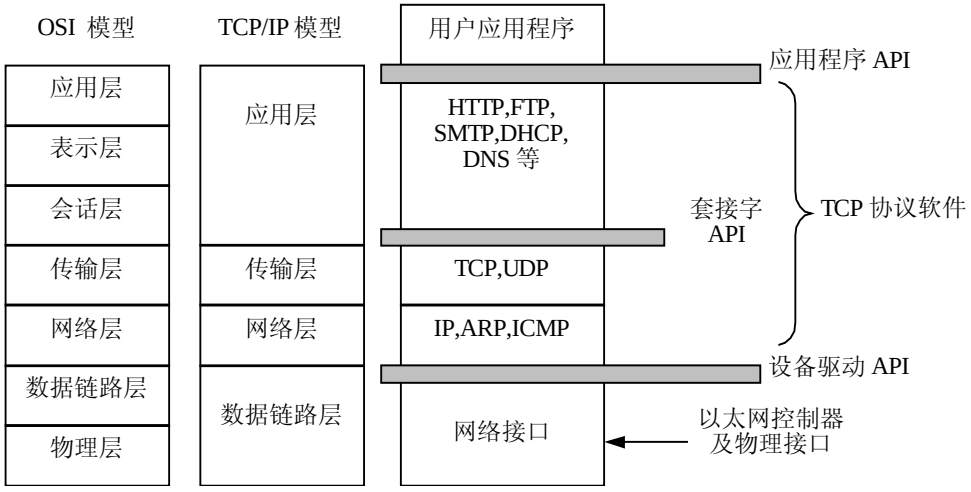

图15-1 网络参考模型及协议对应关系

15.1.2 协议封装

不同网络主机的通信实际上是各层对等实体之间在进行通信，除了最低层的对等实体之间进行的是实际通信外，其他对等实体之间是虚通信，即没有数据流从一台主机的某一层（除了最低层）直接流向另一台主机的相应层，他们之间的通信其实是通过下层提供的服务来完成的。图15-2举例说明了以太网上运行HTTP协议的两台主机通信的模型。实现表示实际通信，虚线表示虚通信。左边的是HTTP客户端，右边则是HTTP的服务器端。在模型的每一层中，双方使用特定的协议进行通信。比如在链路层中使用的是以太网协议，即使用以太帧的格式来收发数据；而在应用层则使用了HTTP协议。在某些层中可以使用的协议有多个，比如运输层可以使用UDP协议或者TCP协议，具体协议的内容在后续章节进行介绍。

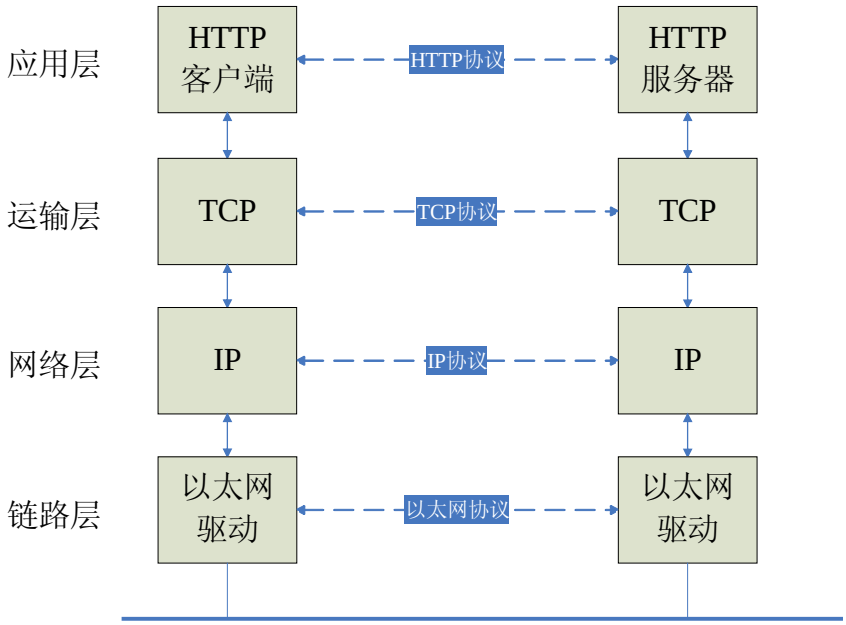

图15-2 以太网上运行HTTP的两台主机

当用户使用TCP/IP协议传输数据时，数据被送入协议栈中，然后逐个通过每一层直到被当作一串比特流送入网络。每层对收到的数据都要增加一些首部信息，如图15-3所示。其中TCP传给IP的数据单元称作TCP报文段(TCP segment)，IP传给网络接口的数据单元称作IP数据报(IP datagram)。通过以太网传输的比特称作帧(Frame)。

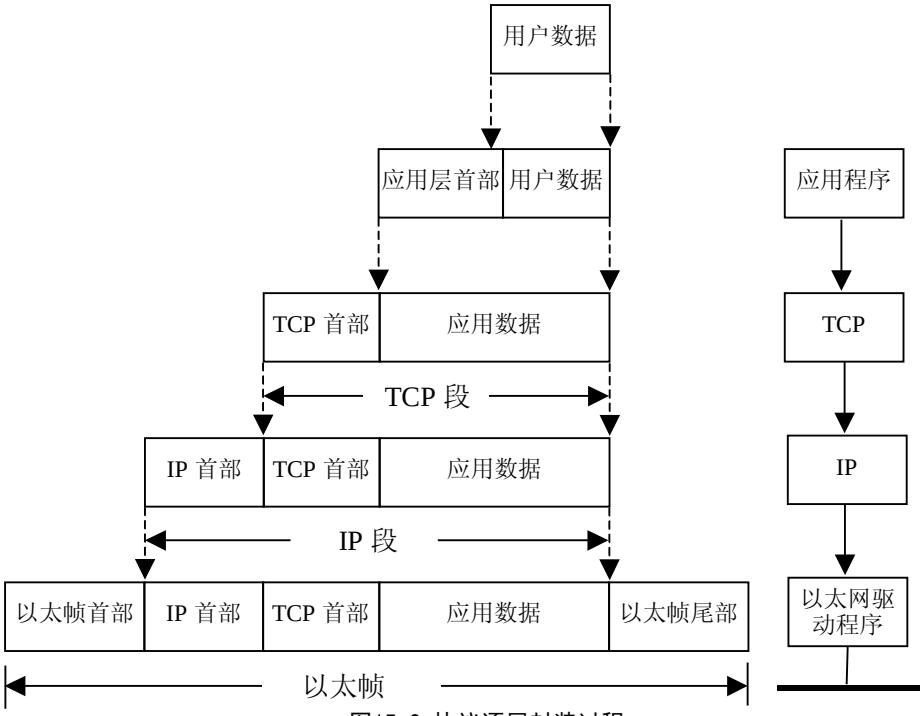

图15-3 协议逐层封装过程

15.2 NE64以太网模块编程基础

NE64的以太网模块实现了以太帧的收发功能。其中包括以太网介质访问控制模块(Ethernet Media Access Controller，EMAC)和以太网物理层驱动模块(Ethernet Physical Transceiver，EPHY)，这两个模块都遵循IEEE802.3标准。NE64的EMAC模块和EPHY模块的内部结构如图15-4所示。

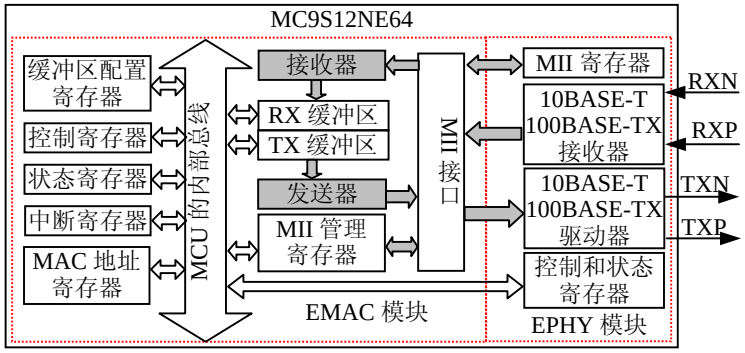

图15-4 NE64的以太网模块结构框图

图中NE64的EMAC模块和EPHY模块可以独立使用，也可以集成使用。如果分开使用则通过MII接口的18个引脚与外界EMAC模块或者EPHY模块通信；如果使用内部模

块，那么MII接口引脚对这两个模块均不可见，这样以太网模块对外通信的引脚只有4个：2差分个输入(RXN、RXP)和2个差分输出(TXN、TXP)。

15.2.1.EPHY模块

EPHY模块是物理接口收发器。支持IEEE-802.3定义的MII（Medium-independent interface）介质无关端口。MII层定义了MAC和各种物理层之间的标准电气和机械接口，它们使得EPHY与EMAC之间的数据通信、EPHY的配置及通信状态的判断得以实现。EPHY模块一端与EMAC通信，另一端与传输介质进行通信。EPHY模块提供4个存储映射寄存器和18个MII寄存器，通过设置这些寄存器，使EPHY模块正常工作。

1.EPHY模块操作模式

EPHY模块是遵循IEEE 802.3标准并兼容10/100Mbps的以太网收发器。可以配置成10BASE-T或者100BASE-TX模式。需要注意的是在外部总线模式下只能选择10BASE-T模式。

① 10BASE-T模式

10BASE-T模式中10表示传输速率为10Mb/S，BASE表示采用基带传输，T(twisted)为双绞线，每个网段最大传输距离为100M，实际传输距离受物理连线影响。使用曼彻斯特编码方式。可以采用3类UTP(Unshielded Twisted-Pair cable 非屏蔽双绞线)、集线器、RJ-45连接器连接。双绞线是两根绝缘导线互相绞结在一起的一种通用的传输介质，它可减少线间电磁干扰，适用于模拟、数据通信。

以太网通信是一种不可靠通信，实际它并不知道通信的对方有没有真正收到自己发出的数据。以太网中半双工模式下采用CSMA/CD（Carrier Sense Multiple Access/Collision Detect），即载波监听多路访问/冲突检测方法。CSMA/CD是一种分布式介质访问控制协议，网中的各个站（节点）都能独立地决定数据帧的发送与接收。每个站在发送数据帧之前，首先要进行载波监听，只有介质空闲时，才允许发送帧。这时，如果两个以上的站同时监听到介质空闲并发送帧，则会产生冲突现象，这使发送的帧都成为无效帧，发送随即宣告失败。每个站必须有能力随时检测冲突是否发生，一旦发生冲突，则应停止发送，以免介质带宽因传送无效帧而被白白浪费，然后随机延时一段时间后，再重新争用介质，重发送帧。

假设A检测到网络是空闲的，开始发数据包，尽力传输，当数据包还没有到达B时，B也监测到网络是空闲的，开始发数据包，这时就会发生碰撞，B首先发现发生碰撞，开始发送碰撞信号，所谓碰撞信号，就是连续的01010101或者10101010，十六进制就是55或AA。这个碰撞信号会返回到A，如果碰撞信号到达A时，A还没有发完这个数据包，A就知道这个数据包发生了错误，就会重传这个数据包。但如果碰撞信号到达A时，数据包已经发完，则A不会重传这个数据包。

② 100BASE-TX模式

快速以太网技术100Base-TX是由10Base-T标准以太网发展而来的，主要解决网络带宽在局域网络应用中的瓶颈问题。可支持100Mbps的数据传输速率，在交换式以太网

环境中可以实现全双工通信。在编码上，采用了效率更高的编码方式4B/5B编码。

对NE64 EPHY模块的实际通信速度可以采用一些方法进行测试。比如将NE64通过网线（5类UTP）接到交换机的端口上，指示灯闪烁表示连接有效，并以颜色表示数据速率：绿色－100Mbps，黄色－10Mbps。

2.EPHY模块的编程寄存器

EPHY支持MII接口和MII管理，它们使得EPHY与EMAC之间的数据通信、EPHY的配置及通信状态的判断得以实现。EPHY模块一端与EMAC通信，另一端与传输介质进行通信。EPHY模块提供4个存储映射寄存器和18个MII寄存器，通过设置这些寄存器，使EPHY模块正常工作。以下是EPHY中使用的所有可访问的寄存器。本部分可作为技术性文档，初学者可以先粗略阅读，编程时遇到相关寄存器设置时可以回头翻阅。

(1) 以太帧物理发送器控制寄存器0（Ethernet Physical Transceiver Control Register 0，EPHYCTL0）

该寄存器的地址为：$0120。该寄存器主要设置EPHY使能、EPHY层的自动协商、通信时钟使能、LED信号以及EPHY中断使能。

数据	D7	D6	D5	D4	D3	D2	D1	D0
定义	EPHYEN	ANDIS	DIS100	DIS10	LEDEN	EPHYWAI		EPHYIEN
复位	0	0	0	0	0	0	0	0

D7—EPHY使能位，可读/写。EPHYEN=1，使能EPHY；EPHYEN=0，禁止EPHY。

D6—自动协商使能位，可读/写。当EPHYEN位从0变为1时，该值被锁存到MII PHY控制寄存器的ANE位。ANDIS=1，启动之后禁止自动协商。工作模式需要手动设置；ANDIS=0，启动之后使能自动协商。工作模式自动实现。

D5—100BASE-TX PLL 使能位，可读/写。DIS100=1，关闭100BASE-TX PLL；DIS100=0，EPHY的操作模式决定100BASE-TX PLL的状态，即由实际使用的网络传输速度决定。

D4—10BASE-T PLL 使能位，可读/写。DIS10=1，关闭10BASE-T PLL；DIS10=0，EPHY的操作模式决定100BASE-T PLL的状态，即由实际使用的网络传输速度决定。

D3—LED驱动使能位，可读/写。LEDEN=1，启动EPHY来驱动LED信号；LEDEN=0，禁止EPHY来驱动LED信号，此时可以通过软件来驱动LED信号。

D2—WAIT模式下的EPHY模块状态位，可读/写。EPHYWAI=1，当MCU在WAIT模式下不能使用EPHY。EPHY中断也不能使得MCU从WAIT模式跳出；EPHYWAI=0，允许EPHY模块继续运行于WAIT模式。

D0—EPHY中断使能位，可读/写。EPHYIEN=1，使能EPHY模块中断；EPHYIEN=0，禁止EPHY模块中断。

(2) 以太帧物理发送器控制寄存器1（Ethernet Physical Transceiver Control Register 1，EPHYCTL1）

该寄存器的地址为：$0121。该寄存器设置MII请求的EPHY地址。PHYADD4是EPHY地址的最高位。当EPHYEN位从0变为1时，该地址值被锁存到MII PHY地址寄存器的4：

0位。

数据	D7	D6	D5	D4	D3	D2	D1	D0
定义				PHYADD4	PHYADD3	PHYADD2	PHYADD1	PHYADD0
复位	0	0	0	0	0	0	0	0

(3) 以太帧物理发送器状态寄存器（Ethernet Physical Transceiver Status Register，EPHYSR）

该寄存器的地址为：$0122。该寄存器主要反应EPHY中的状态，包括网络通信速度以及EPHY的中断情况。

数据位	D7	D6	D5	D4	D3	D2	D1	D0
定义			100DIS	10DIS				EPHYIF
复位	0	0	0	0	0	0	0	0

D5—EPHY速度状态位，只读。100DIS=1，禁止了100BASE-TX端口；100DIS=0，使能了100BASE-TX端口。

D4—EPHY速度状态位，只读。10DIS=1，禁止了10BASE-TX端口；10DIS=0，使能了10BASE-TX端口。

D0—EPHY中断标志位，可读/写。它表示是否发生了EPHY中断，向该位写1可以清除中断，写0无意义。具体发生何种中断需要通过MII管理接口读出中断控制寄存器。EPHYIF=1，发生EPHY中断；EPHYIF=0，未发生EPHY中断。

表 15-2 MII 寄存器

地址		寄存器名	访问权限
0	%00000	控制寄存器	读/写
1	%00001	状态寄存器	读/写
2	%00010	PHY识别寄存器1	读/写
3	%00011	PHY识别寄存器2	读/写
4	%00100	自动协商发布寄存器	读/写
5	%00101	自动协商连接能力寄存器	读/写
6	%00110	自动协商扩展寄存器	读/写
7	%00111	自动协商下页发送	读/写
8~15	%01000~%01111	保留	读/写
16	%10000	中断控制寄存器	读/写
17	%10001	MII管理专用状态寄存器	读/写
18	%10010	MII管理专用控制寄存器	读/写

通过MII管理接口可以访问的所有EPHY中的寄存器。这些寄存器不属于MCU的存储器映射空间，所以不能直接操作，只能通过MII接口进行配置和管理，表15-2给出了这些寄存器，下面详细介绍。

(4) EPHY控制寄存器（EPHY Control Register）

该MII寄存器的地址为0（%00000）。

数据	D15	D14	D13	D12	D11	D10	D9	D8
定义	RESET	LOOPBACK	DATARATE	ANE	PDWN	ISOL	RAN	DPLX
复位	0	0	0	0	0	0	0	0
数据	D7	D6	D5	D4	D3	D2	D1	D0
定义	COLTEST							

复位	0	0	0	0	0	0	0	0

D15—EPHY复位位。RESET=1，PHY复位端口状态，寄存器为复位值。复位之后，该位自动清0。复位时间大约需要1.3ms；RESET=0，无影响。

D14—数字循环模式位。LOOPBACK=1，使能数字循环模式。在该模式下，TXD数据直接发送到RXD，PHY与传输介质独立开来；LOOPBACK=0，禁止数字循环模式。

D13—速度选择位。DATARATE=1，如果禁止自动协商，则选择100Mbps的速度；DATARATE=0，则选择10Mbps的速度。

D12—自动协商使能位。ANE=1，使能自动协商；ANE=0，禁止自动协商，此时由DATARATE和DPLX位定义连接配置。

D11—低功耗位。PDWN=1，端口处于低功耗模式；PDWN=0，正常模式。

D10—隔离位。ISOL=1，MII与端口数据通道信号独立；ISOL=0，正常模式。

D9—自动协商重启位。RAN=1，自动协商使能时，重新启动自动协商；RAN=0，正常模式。

D8—双工模式位。当LOOPBACK=1时，该位对PHY无影响。DPLX=1，表示全双工模式；DPLX=0，表示半双工模式。

D7—冲突检测位。该功能在LOOPBACK=1时才能有效。COLTEST=1，强迫PHY当MII_TXEN有效时，在512位时间内使MII_COL信号有效；当MII_TXEN无效时，在4位时间内使MII_COL信号无效；COLTEST=0，正常模式。

(5) EPHY状态寄存器（EPHY Status Register）

该MII寄存器的地址为1（%00001）。该寄存器发布MII的端口能力。

数据位	D15	D14	D13	D12	D11	D10	D9	D8
定义	100T4	100XFD	100XHD	10TFD	10THD			
复位	0	0	0	0	0	0	0	0
数据位	D7	D6	D5	D4	D3	D2	D1	D0
定义		SUPPRE	ANCOMP	REMFLT	ANABL	LNKSTST	JABDT	EXCAP
复位	0	0	0	0	0	0	0	0

D15—PHY支持100BASE-T4发送。100T4=1，支持；100T4=0，不支持，EPHY模块未实现该功能。

D14—PHY是否支持100BASE-TX全双工模式。100XFD=1，支持；100XFD=0，不支持。

D13—PHY是否支持100BASE-TX半双工模式。100XHD=1，支持；100XHD=0，不支持。

D12—PHY是否支持10BASE-T全双工模式。10TFD=1，支持；10TFD=0，不支持。

D11—PHY是否支持10BASE-T半双工模式。10THD=0，支持；10THD=0，不支持。

D6—MF前导镇压位。SUPPRE=1，表明管理帧不需要前导位；SUPPRE=0，表明管理帧需要前导位。

D5—自动协商完成位。ANCOMP=1，表明自动协商处理完成了，此时MII寄存器4到7有效；ANCOMP=0，表明自动协商处理未完成。

D4—远程错误位。这些错误可能是：连接对象发送了一个RF=1位（MII5.13=1）；

连接对象协议不是00001（MII5.4:0）；连接对象发布能力只支持T4（MII5.9:5）；在PHY与连接对象之间无共同的工作模式。REMFLT=1，检测到远程错误，当读该寄存器或者PHY复位时，该位被清0；REMFLT=0，未检测到错误。

D3—自动协商能力位。ANABL=1，表示PHY有自动协商能力；否则表示无该能力。

D2—连接状态位。LNKSTST=1，表示有效连接建立；否则未建立连接。当连接失败时，LNKSTST被清0，一直保持到管理接口读该寄存器。

D1—模糊（Jabber）位。对于100BASE-TX模式，该位被清0。JABDT=1，检测到模糊条件位；反之未检测到。

D0—扩展能力位。EXCAP=1，PHY实现了扩展寄存器（寄存器2-31）；否则不支持扩展寄存器。

(6) 自动协商发布寄存器（Auto-Negotiate Advertisement Register）

自动协商处理需要4个寄存器来与连接对象进行连接信息交流，下面将分别介绍这4个寄存器。首先介绍的是自动协商发布寄存器，该MII寄存器的地址为4（%00100），为16位可读/写寄存器。上电之后，自动协商开始之前，该寄存器设置D4~D0位为00001，表示兼容IEEE 802.3标准。根据MII状态寄存器的D15~D11位设置该寄存器的D9~D5位。此外，MII可以设置D9~D5位用于二次自动协商。

数据	D15	D14	D13	D12	D11	D10	D9	D8
定义	NXTP		RFLT			FLCTL		TAF100FD
复位	0	0	0	0	0	0	0	0
数据	D7	D6	D5	D4	D3	D2	D1	D0
定义	TAF100HD	TAF10FD	TAF10HD	SELECTORFIELD[4：0]				
复位	0	0	0	0	0	0	0	0

D15—下一页位。NXTP=1，表示有能力发送下一页；否则无能力。

D13—远程错误位。RFLT=1，表示有远程错误；否则无远程错误。

D10—流量控制位。FLCTL=1，具有流量控制功能，如果FLP流中相应的位未设置，那么设置该位无意义；FLCTL=0，未实现流量控制。

TAF100FD、TAF100HD、TAF10FD、TAF10HD：分别表示100BASE-TX全双工、100BASE-TX半双工、10BASE-T全双工、10BASE-T半双工；当这些值为1时表示具有该能力，否则不具有该能力。

(7) 自动协商连接对象能力寄存器（Auto-Negotiate Link Partner Ability）

该MII寄存器的地址为5（%00101），为16位只读寄存器，它反应的仍然是对方的连接能力。

数据位	D15	D14	D13	D12	D11	D10	D9	D8
定义	NXTP	ACK	MSGP	ACK2	TGL	Message/Unformatted Code		
复位	0	0	0	0	0	0	0	0
数据位	D7	D6	D5	D4	D3	D2	D1	D0
定义	Message/Unformatted Code Field							
复位	0	0	0	0	0	0	0	0

D15—下一页位。NXTP=1，表示接下来还有一页；NXTP=0，表示这是最后一页。

D14—响应位。ACK=1，连接对象接收到了连接码字；ACK=0，未收到连接码字。

D13—信息页位。MSGP=1，表示该页是信息页，否则表示该页无意义。

D12—响应位2。ACK2=1，表示接收器有能力处理信息中定义的任务，否则表示无能力。

D11—跟踪位。TGL=1，传输的连接码字之前的值为0；TGL=0，传输的连接码字之前的值为1。

D10～D0—信息编码位。如果该值在IEEE802.3u-1995 附录28C中定义过，那么该值有意义，否则这11位值为任意值。

(8) 自动协商扩展寄存器（Auto-Negotiate Expansion Register）

该MII寄存器的地址为6（%00110），为16位只读寄存器，它反映的是连接对方的自动协商能力以及并行检测机制的状态信息。

数据位	D15	D14	D13	D12	D11	D10	D9	D8
定义								
复位	0	0	0	0	0	0	0	0
数据位	D7	D6	D5	D4	D3	D2	D1	D0
定义				PDFLT	LPNPA	NXTPA	PRCVD	LPANA
复位	0	0	0	0	0	0	0	0

D4—并行检测错误位，用于检测100BASE-TX的连接完整性功能。PDFLT=1，并行检测错误，即无完整性连接；PDFLT=0，连接正常。

D3—连接对象的下一页能力位。该位告诉MI和连接对象该端口是否有传输下一页能力。LPNPA=1，该端口具有传输下一页能力；否则不具有传输下一页能力。

D2—下一页能力位。这个位表示端口是否有下一页的能力。NXTPA=1，有；NXTPA=0，没有。

D1—页接收位。PRCVD=1，接收到了3个连续的相同的连接码字，并且已经存放在A/N连接对象连接能力寄存器中；否则表示未收到3个连续的相同的连接码字。

D0—连接对象的A/N能力位。LPANA=1，连接对象具有A/N能力；否则无A/N能力。

(9) 自动协商协商下一页发送寄存器（Auto-Negotiate Next Page Transmit）

该MII寄存器的地址为7（%00111），为16位可读/写寄存器，当需要交换与连接对象的信息时，可以通过MI写该寄存器。默认情况下，PHY只向连接对象发送NULL信息，除非STA将该寄存器器的值全部覆盖。只有当连接对象没有页需要发送并且D15位被STA清0时，才能发送下一页。该寄存器中各位值的含义请参考自动协商连接对象能力寄存器。

数据位	D15	D14	D13	D12	D11	D10	D9	D8
定义	NXTP		MSGP	ACK2	TGL	Message/Unformatted Code		
复位	0	0	0	0	0	0	0	0
数据位	D7	D6	D5	D4	D3	D2	D1	D0
定义	Message/Unformatted Code							
复位	0	0	0	0	0	0	0	0

接下来将讲述PHY中的设置其工作模式的MII寄存器。这些寄存器的功能包括：设置通信速度，测试模式，电路滤波。中断等。

(10) 中断控制寄存器（Interrupt Control Register）

该MII寄存器的地址为16（%10000），为16位可读/写寄存器，用于设置PHY的中断使能，同时判断中断发生情况。该寄存器的D15和D7~D0位为只读位，其他位可读/写。

数据位	D15	D14	D13	D12	D11	D10	D9	D8
定义		ACKIE	PRIE	LCIE	ANIE	PDFIE	RFIE	JABIE
复位	0	0	0	0	0	0	0	0
数据位	D7	D6	D5	D4	D3	D2	D1	D0
定义		ACKR	PGR	LKC	ANC	PDF	RMTF	JABI
复位	0	0	0	0	0	0	0	0

D14—（从连接对象）接收到响应位中断使能位。ACKIE=1，使能中断。

D13—接收到新页中断使能位。PRIW=1，使能中断。

D12—连接变化中断使能位。LCIE=1，使能中断。

D11—自动协商变化中断使能位，表示自从最后一次访问该寄存器开始，自动协商的状态发生了变化，将产生中断。ANIE=1，使能中断。

D10—发生并行检测错误中断使能位。PDFIE =1，使能中断。

D9—远程错误中断使能位。RFIE =1，使能中断。

D8—检测到模糊条件（Jabber）中断使能位。JABIE=1，使能中断。

D6~D0位分别对应上述中断使能的中断标志位，这些位为1时，表示发生中断。

(11) MII管理专用状态寄存器（Proprietary Status Register）

该MII寄存器的地址为17（%10001），为16位只读寄存器，反映了PHY的速度、双工模式等属性。

数据位	D15	D14	D13	D12	D11	D10	D9	D8
定义		LNK	DPMD	SPD		ANNC	PRCVD	ANCMODE
复位	0	0	0	0	0	0	0	0
数据位	D7	D6	D5	D4	D3	D2	D1	D0
定义			PLR					
复位	0	0	0	0	0	0	0	0

D14—连接状态位，为MII状态寄存器的LNKSTAT位的复制位。LNK=1，连接未建立；LNK=0，连接建立。

D13—双工模式位。DPMD=1，全双工模式DPMD=0，半双工模式。

D12—速度位。SPD=1，速度为100Mbps；SPD=0，速度为10Mbps。

D10—自动协商完成位，为MII状态寄存器的ANCOMP位的复制位。ANNC=1，自动协商完成；反之未完成。

D9—接收到页位。PRCVD=1，接收到3个相同的联系的连接码字；反之，未接收到。

D8—自动协商为共同工作模式位，只有当ANNC=1时该位才有效。ANCMODE=1，无共同工作模式；反之自动协商到共同工作模式。

D5—反极性位（10BASE-T）。PLR=1，10BASE-T的接收极性为反极性；反之为正常极性。

(12) MII管理专用控制寄存器（Proprietary Control Register）

该MII寄存器的地址为18（%10010），为16位可读/写寄存器，反映了PHY的速度、双工模式等属性。

数据	D15	D14	D13	D12	D11	D10	D9	D8
定义		FEFLT	MIILBD			JBDE	LNKTS	POLCO
复位	0	0	0	0	0	0	0	0
数据	D7	D6	D5	D4	D3	D2	D1	D0
定义	ALGD	ENCBY	SCRBYP	TRDAN	TRTST			
复位	0	0	0	0	0	0	0	0

D14—远程错误检测使能位。FEFLTD =1，禁止远程错误检测；FEFLTD =0，一旦发送和接收使能，远程错误检测就开始。只有在禁止自动协商时才适用。

D13—MII循环使能位。MIILBO=1，禁止MII循环；MIILBO=0，MII发送的数据直接送至MII接收引脚。

D10—模糊检测（10BASE-T）使能位。JBDE=1，使能模糊检测；反之禁止。

D9—连接测试（10BASE-T）使能位。LNKTSTD=1，禁止10BASE-T连接完整性测试；反之使能。

D8—极性纠错（10BASE-T）使能位。POLCORD=1，禁止10BASE-T接收极性纠错；POLCORD=0，自动极性纠错。

D7—对齐使能位。ALGD=1，非对齐模式，只有在字符模式中可用；ALGD=0，对齐模式。

D6—编码滤波位。ENCBYP=1，字符模式，忽略4B/5B编码和解码；ENCBYP=0，正常模式。

D5—编码（Scrambler）忽略模式位。SCRBYP=1，忽略编码（Scrambler）和译码；反之，正常工作。

D4—发送和接收断开和模拟循环位。TRDANALB=1，高阻双绞线发送器，如果是模拟循环模式，则该位强制为1；反之正常工作。

D3—（100BASE-TX）发送和接收测试位。TRTST=1，无论连接状态是什么，总是发送和接收数据；反之正常工作。

15.2.2.EMAC模块

MC9S12NE64的EMAC实现了数据链路层的功能，提供将一台设备的网络层数据传输至另一台设备的机制。EMAC主要负责数据的传输，具有10Mbps/100Mbps的传输能力。在数据传输之前，EMAC自动对以太帧进行封装：添加帧的前导位、帧起始位以及CRC校验码。此外，EMAC具有MAC地址识别和滤波功能，并能进行错误检测。在半双工模式下，EMAC采用载波多路访问和冲突检测(CSMA/CD)机制；在全双工模式下，可采用流量控制机制。因此，EMAC模块的初始化应该是包括这些功能在内的初始化，它可以通过读写EMAC模块提供的21个寄存器来实现。

1.EMAC模块的编程寄存器

MC9S12NE64的EMAC模块一共包括21个寄存器，占用48个字节。寄存器地址由基址和偏移量构成，基址由MCU决定NE64芯片中此基地址为$0140，偏移量在$00~$2F之间，地址=基地址+偏移量。EMAC模块存储器映射如表15-3所示。

表 15-3 EMAC 模块存储器映射

地址偏移	寄存器名	访问权限
$_00	网络控制寄存器（NETCTL）	读/写
$_01、$_02	保留	
$_03	接收控制和状态寄存器（RXCTS）	读/写
$_04	发送控制和状态寄存器（TXCTS）	读/写
$_05	以太帧类型控制寄存器（ETCTL）	读/写
$_06、$_07	可编程以太帧类型寄存器（ETYPE）	读/写
$_08、$_09	PAUSE帧定时和计数寄存器（PTIME）	读/写
$_0A、$_0B	中断事件寄存器（IEVENT）	读/写
$_0C、$_0D	中断屏蔽寄存器（IMASK）	读/写
$_0E	软件复位寄存器（SWRST）	读/写
$_0F	保留	
$_10	MII管理PHY地址寄存器（MPADR）	读/写
$_11	MII管理寄存器地址寄存器（MRADR）	读/写
$_12、$_13	MII管理写数据寄存器（MWDATA）	读/写
$_14、$_15	MII管理读数据寄存器（MRDATA）	读
$_16	MII管理命令和状态寄存器（MCMST）	读/写
$_17	保留	
$_18、$_19	以太帧缓冲区配置寄存器（BUFCFG）	读/写
$_1A、$_1B	A接收缓冲区帧结束指针寄存器（RXAEFP）	读
$_1C、$_1D	B接收缓冲区帧结束指针寄存器（RXBEFP）	读
$_1E、$_1F	发送缓冲区帧结束指针寄存器（TXEFP）	读
$_20~$_27	组播帧HASH表寄存器（MCHASH）	读/写
$_28~$_2D	MAC地址寄存器（MACAD）	读/写
$_2E、$_2F	杂项寄存器（EMISC）	读/写

以下是EMAC模块中各个寄存器以及寄存器中每位的含义。本部分可作为技术性文档，初学者可以先粗略阅读，编程时遇到相关寄存器设置时可以回头翻阅。

(1) 网络控制寄存器（Network Control，NETCTL）

该寄存器的地址为：$0140。通过该寄存器可以使能EMAC、外部PHY以及双工模式等。需要注意的是：必须在设置EMACE之前设置MLB或EXPHY，即至少分两次写该寄存器才能完成。当设置MLB或EXPHY时，必须断开内部或外部PHY连接，避免在端口配置逻辑时MII上产生干扰。

数据	D7	D6	D5	D4	D3	D2	D1	D0
定义	EMACE			ESWAI	EXTPHY	MLB	FXD	
复位	0	0	0	0	0	0	0	0

D7—EMAC使能位，可读/写。当数据正在传输时（TXACT=1），不能修改该位。=1，允许接收和发送数据；一旦置该位=0，则EMAC接收器和发送器立即停止。任何

在处理过程中的接收操作均被丢弃，任何PAUSE定时器均被清空。EMACE与MII管理功能无关。

D4—WAIT模式下EMAC使能位，可读/写。当数据正在传输时（TXACT=1）或MII忙（BUSY=1）时，不能通过置该位进入WAIT模式。ESWA=1，在WAIT模式下禁止EMAC；ESWA=0，在WAIT模式下EMAC正常工作。

D3—外部PHY设置位，该位在硬件或软件复位之后，只能写一次，但是用户不能当EMACE=1或BUSY=1时修改该位。EXTPHY=1，使用外部PHY，此时所有外部的EMAC MII I/O可使用；EXTPHY=0，使用内部PHY，此时所有内部的MII接口可使用。

D2—MAC循环位，该位在硬件或软件复位之后，只能写一次，但是用户不能当EMACE=1或BUSY=1时修改该位。当处于循环模式时，接收器的帧识别算法仍然有效，那些不满足算法的发送帧会被接收器丢弃。MLB=1，EMAC进入循环模式，发送数据直接进入内部接收器，禁止MII；MLB=0，正常操作。

D1—双工设置位，可读/写。当数据正在传输时（TXACT=1），不能修改该位。FXD=1，EMAC为全双工模式，帧的传送和接收是相互独立的，该模式下可以启用流量控制；FXD=0，EMAC为半双工模式，帧的发送和接收不能同时进行，该模式下可以使用CSMA/CD协议进行冲突和网络访问的管理。

(2) 接收控制和状态寄存器（Receive Control and Status，RXCTS）

该寄存器的地址为：$0143。通过该寄存器可以判断接收器的状态，设置是否进行流量控制以及设置EMAC可以接收何种地址的帧。

数据	D7	D6	D5	D4	D3	D2	D1	D0
定义	RXACT			RFCE		PROM	CONMC	BCREJ
复位	0	0	0	0	0	0	0	0

D7—接收器状态位，只读。RXACT=1，接收器正忙，当MII_RXDV有效时，该位置1；RXACT=0，接收器空闲，当MII_RXDV无效后且EMAC已经处理完接收帧，该位置0。

D4—流量控制使能位，可读/写。当EMAC=1时，不能修改该位。RFCE=1，接收器检测到了PAUSE帧（只适用于全双工模式）。一旦接收到PAUSE帧，发送器将停止发送数据帧，停止的时间在接收到的帧内给出。每接收到有效的PAUSE帧，PAUSE定时计数器就会更新数值。RFCE=0，忽略接收到的PAUSE控制帧。

D2—帧地址混合模式位，可读/写。当EMAC=1时，不能修改该位。当接收器忙时修改该位，可能会影响滤波结果。PROM=1，地址识别滤波不起作用，无论帧的目的地址是什么，所有帧都被接收；PROM=0，EMAC将按照CONMC位和BCREJ位的设置接收相应的地址帧。

D1—条件组播地址接收设置位，可读/写。当EMAC=1时，不能修改该位。当接收器忙时修改该位，可能会影响滤波结果。CONMC=1，用组播hash表检查所有接收到的组播地址帧；CONMC=0，接收所有组播地址帧。

D0—广播帧地址接收设置位，可读/写。当EMAC=1时，不能修改该位。BCREJ=1，拒绝所有广播地址帧，只接收唯一帧；BCREJ=0，接收所有广播地址帧。

(3) 发送控制和状态寄存器（Transmit Control and Status，TXCTS）

该寄存器的地址为：$0144。通过该寄存器可以判断发送器的状态，设置流量控制功能，设置以太帧操作命令等。

数据	D7	D6	D5	D4	D3	D2	D1	D0
定义	TXACT		CSLF	PTRC	SSB		TCMD	TCMD
复位	0	0	0	0	0	0	0	0

D7—发送器状态位，只读。该状态位反应EMAC发送器是否忙。TXACT=1，发送器忙。当TCMD中写入有效命令时，TXACT=1；TXACT=0，发送器空闲。当EMAC完成了帧的发送，则TXACT=0。

D5—载波监听丢失标志位，可读/写。该位保证了帧正常发送，并且无需软件重试。向该位写1则清除该位，写0无影响。CSLF=1，载波监听丢失（半双工模式）或在无冲突的传输过程（不包括帧前导位）中没有发生载波监听；CSLF=0，没有检测到载波监听丢失。

D4—PAUSE帧定时器寄存器控制位，可读/写。PTRC =1，可以对PTIME寄存器进行写操作，从而更新PAUSE的持续时间，该时间单位为512位时间；PTRC =0，可以对PTIME寄存器进行读操作，获取接收器接收到PAUSE帧后，还余留持续时间，单位为512位时间。此时对PTIME写操作对其无影响。

D3—时间片设置位，可读/写。SSB=1，单个以太网时间片；SSB =0，随机backoff算法时间。

D1、D0—发送命令位，只写。通过这两位可以发送3个不同的命令：START，PAUSE和ABORT。START表示开始发送在发送缓冲区的帧；PAUSE表示发送一个由硬件产生的PAUSE帧；如果错误的CRC校验码添加到帧尾以及MII_TXER=1，ABORT命令可以中止任何帧发送。如果TXACT=1，START和PAUSE命令被忽略；如果接收到PAUSE命令，则START命令帧被悬挂直到PAUSE帧持续时间结束；在PAUSE帧持续时间，如果没有START命令被悬挂，EMAC帧可以发送一个控制PAUSE帧。

TCMD=0，保留，忽略；

TCMD=1，START，启动发送缓冲区帧；

TCMD=2，PAUSE，启动pause帧（只适用全双工）；

TCMD=3，ABORT，取消发送。

(4) 以太帧类型控制寄存器（Ethertype Control，ETCTL）

该寄存器的地址为：$0145。通过该寄存器可以设置EMAC可以接收哪些类型的帧。当EMAC=1时，不能修改该寄存器值。当接收器忙时修改该寄存器，可能会影响滤波结果。

如果ETCTL寄存器中的每位均为1，则可以接收所有类型的帧。相反，该寄存器中表示某种类型的位为0，则该类型的帧被滤波。

数据	D7	D6	D5	D4	D3	D2	D1	D0
定义	FPET			FEMW	FIPV6	FARP	FIPV4	FIEEE
复位	0	0	0	0	0	0	0	0

D7—可编程以太帧类型，FPET=1，接收ETYPE寄存器中设置的类型帧；FPET=0，

忽略ETYPE寄存器中设置的类型帧。

D4—Emware类型帧（0x8876），FEMW =1，接收Emware类型帧；FEMW =0，忽略Emware类型帧。

D3—网际协议6类型帧（0x86DD），FIPV6=1，接收IPv6类型帧；FIPV6 =0，忽略IPv6类型帧。

D2—地址解析协议类型帧（0x0806），FARP =1，接收ARP类型帧；FARP=0，忽略ARP类型帧。

D1—网际协议4类型帧（0x8600），FIPV4=1，接收IPv4类型帧；FIPV4 =0，忽略IPv4类型帧。

D0—IEEE802.3长度域以太帧（长度在0x0000～0x05DC），FIEEE=1，接收该长度范围的帧；FIEEE=0，忽略该长度范围的帧。

(5) 可编程以太帧类型寄存器（Programmable Ethertype，ETYPE）

该寄存器的地址为：$0146。它是16位寄存器，与ETCTL寄存器一起用于以太帧类型滤波。

(6) PAUSE帧定时和计数寄存器（PAUSE Timer Value and Counter，PTIME）

该寄存器的地址为：$0148。它是16位寄存器，控制PAUSE帧的持续时间，详细解释见TXCTS寄存器中PTRC位的解释。此外，无论PTRC为何值，接收到一个有效的PAUSE控制帧，将更新PAUSE定时计数器的值。

(7) 中断事件寄存器（Interrupt Event，IEVENT）

该寄存器的地址为：$014A。它是16位寄存器，包含了EMAC中的所有中断事件标志位，通过该标志可以判断发生了何种中断或者直接进入中断事件。该寄存器可读/写，向各个中断事件位写1将清除中断标志，写0无影响。

数据	D15	D14	D13	D12	D11	D10	D9	D8
定义	RFCIF		BREIF	RXEIF	RXAOIF	RXBOIF	RXACIF	RXBCIF
复位	0	0	0	0	0	0	0	0
数据	D7	D6	D5	D4	D3	D2	D1	D0
定义	MMCIF		LCIF	ECIF			TXCIF	
复位	0	0	0	0	0	0	0	0

D15—接收流量控制中断标志位。RFCIF=1，表示在全双工模式下，接收到了流量控制帧，发送器停止发送数据；RFCIF=0，正常发送数据。

D13—接收帧过长错误标志位。BREIF=1，表示接收到的帧长度大于MAXFL的值，产生出错中断；否则正常通信。

D12—接收帧错误标志位。RXEIF=1，表示接收帧的长度不匹配、队列出错或者CRC出错；否则数据正确。

D11—接收缓冲区A溢出中断标志位。RXAOIF=1，表示接收缓冲区A发生溢出；否则正常。

D10—接收缓冲区B溢出中断标志位。RXBOIF=1，表示接收缓冲区B发生溢出；否则正常。

D9—A缓冲区接收完成标志位。RXACIF=1，表示接收缓冲区A接收数据完成；否则表示接收帧还未被确认有效。

D8—B缓冲区接收完成标志位。RXBCIF=1，表示接收缓冲区B接收数据完成；否则表示接收帧还未被确认有效。

D7—MII管理传输完成中断标志位。MMCIF=1，表示MII管理传输完成，否则表示MII管理传输正在进行或者无请求。

D5—滞后冲突中断标志位。LCIF=1，表示在半双工模式下，在512位时间的冲突窗口之后发生了中断，否则表示没有发生该中断。

D4—过量冲突中断标志位。ECIF=1，表示在半双工模式下，冲突次数超过了最大重发次数15，此时该帧被丢弃，必须启动START命令重新发送。ECIF=0，表示冲突数量未超过15次。

D1—帧发送完成中断标志位。TXCIF=1，表示帧发送完成，否则表示发送还未确认。

(8) 中断屏蔽寄存器（Interrupt Mask，IMASK）

该寄存器的地址为：$014C。它是16位寄存器，包含了IEVENT中的所有中断事件的使能位，它们的每位含义相同。该寄存器可读/写，位值=1，表示允许中断，位值=0，表示禁止中断。

数据	D15	D14	D13	D12	D11	D10	D9	D8
定义	RFCIF		BREIE	RXEIE	RXAOIE	RXBOIE	RXACIE	RXBCIE
复位	0	0	0	0	0	0	0	0
数据	D7	D6	D5	D4	D3	D2	D1	D0
定义	MMCIE		LCIE	ECIE			TXCIE	
复位	0	0	0	0	0	0	0	0

(9) 软复位寄存器（Software Reset，SWRST）

该寄存器的地址为：$014E。它主要实现EMAC软复位功能，当BUSY=1时，不能修改该位。

数据	D7	D6	D5	D4	D3	D2	D1	D0
定义	MACRST							
复位	0	0	0	0	0	0	0	0

D7—EMAC软复位，有关EMAC的所有逻辑被初始化，所有EMAC寄存器被复位。任何正在进行的发送和接收操作全部结束。MACRST=1，复位EMAC；MACRST=0，正常操作。

(10) MII管理PHY地址寄存器（MII Management PHY Address，MPADR）

该寄存器的地址为：$0150。该寄存器的D4~D0位可以对32个与其连接的PHY设备进行编号。复位值为0，表示内部PHY的编号，当然也可以通过此寄存器改变该值。

数据位	D7	D6	D5	D4	D3	D2	D1	D0
定义				PADDR				
复位	0	0	0	0	0	0	0	0

(11) MII管理寄存器地址寄存器（MII Management Register Address，MRADR）

该寄存器的地址为：$0151。该寄存器的D4~D0位可以对32个MII管理寄存器设置地址。

数据位	D7	D6	D5	D4	D3	D2	D1	D0
定义				RADDR				
复位	0	0	0	0	0	0	0	0

(12) MII管理写数据寄存器（MII Management Write Data，MWDATA）

该寄存器的地址为：$0152。它是16位寄存器，可以通过该寄存器将数据写入MII寄存器。

数据位	D15	D14	D13	D12	D11	D10	D9	D8
定义	WDATA							
复位	0	0	0	0	0	0	0	0
数据位	D7	D6	D5	D4	D3	D2	D1	D0
定义	WDATA							
复位	0	0	0	0	0	0	0	0

(13) MII管理读数据寄存器（MII Management Read Data，MRDATA）

该寄存器的地址为：$0154。它是16位寄存器，只读，可以通过该寄存器读出MII寄存器数据。读操作必须在MMCIF=1时数据才有效。

数据位	D15	D14	D13	D12	D11	D10	D9	D8
定义	RDATA							
复位	0	0	0	0	0	0	0	0
数据位	D7	D6	D5	D4	D3	D2	D1	D0
定义	RDATA							
复位	0	0	0	0	0	0	0	0

(14) MII管理命令和状态寄存器（MII Management Command and Status，MCMST）

该寄存器的地址为：$0156。

数据位	D7	D6	D5	D4	D3	D2	D1	D0
定义	OP		BUSY	NOPRE	MDCSEL			
复位	0	0	0	0	0	0	0	0

D7、D6—操作码位，可读/写，2位。OP=10，将产生读帧操作；OP=01，将产生写帧操作；如果为其他组合的值均无意义。需要注意的是，这些操作必须在BUSY=0时才能进行。如果MDCSEL=0，OP写被忽略，这2位读出值始终为00。

D5—操作状态位，只读。BUSY=1，MII忙，当OP被写时该位置1；BUSY=0，MII空闲，当MMCIF=1时该位置0。

D4—前导设置位。根据IEEE802.3标准，如果PHY层不需要前导位，那么MII管理接口可以不发前导位。NOPPRE=1，无前导位；NOPPRE=0，32位前导位。

D3～D0—管理时钟频率设置位，可读/写，4位。该时钟用于MII管理传输，当MDC空闲时保持低电平。MDC频率的计算公式如下：

MDC频率=总线时钟/（2* MDCSEL）

根据IEEE的 MII 规范，MDCSEL的设置值必须使得MDC的时钟频率小于或等于

2.5MHz，且MDCSEL不能为0。下面举例计算MDCSEL值。假设总线频率为25 MHz，需要MDC的频率为2.5MHz，则MDCSEL=总线时钟/（MCD频率*2）=25 MHz/（2.5MHz *2）=5。

(15) 以太网缓冲区配置寄存器（Ethernet Buffer Configuration，BUFCFG）

该寄存器的地址为：$0158。它是16位寄存器，主要设置以太帧发送接收缓冲区的大小和存储分配，复位值不全为0。

数据位	D15	D14	D13	D12	D11	D10	D9	D8
定义		BUFMAP				MAXFL		
复位	0	0	0	0	0	0	0	0
数据位	D7	D6	D5	D4	D3	D2	D1	D0
定义	MAXFL							
复位	0	0	0	0	0	0	0	0

D14～D12—缓冲区大小和起始地址映射位，3位。复位之后且EMACE=0，该值才能且只能写一次。表15-4是缓冲区映射配置。

表 15-4 缓冲区配置

缓冲区映射	RAM启始地址	接收缓冲区A大小（字节）	接收缓冲区A地址空间	接收缓冲区B大小（字节）	接收缓冲区B地址空间	发送缓冲区启始地址
0	0x0000	128	0x0000 - 0x007F	128	0x0080 - 0x00FF	0x0100
1	0x0000	256	0x0000 - 0x00FF	256	0x0100 - 0x01FF	0x0200
2	0x0000	512	0x0000 - 0x01FF	512	0x0200 - 0x03FF	0x0400
3	0x0000	1K	0x0000 - 0x03FF	1K	0x0400 - 0x07FF	0x0800
4	0x0000	1.5K	0x0000 - 0x05FF	1.5K	0x0600 - 0x0BFF	0x0C00

D10～D0—允许接收最大帧长度位，11位。当EMACE=1时，不能改变这些位的值。MAXFL设置值的单位为字节，如果接收帧长度超过MAXFL值会发生帧过长事件。如果写入的数据小于64则被置64，如果写入的数据大于1518则被置1518。

(16) 接收缓冲区A帧结束指针寄存器（Receive A End-of-Frame Pointer，RXAEFP）

该寄存器的地址为：$015A。它是16位只读寄存器，高位D15~D11无意义。RXAEFP位为D10~D0，指出写到接收缓冲区A的最后一个字节的地址偏移量。当RXACIF=1时，RXAEFP的值才有效。

(17) 接收缓冲区B帧结束指针寄存器（Receive B End-of-Frame Pointer，RXBEFP）

该寄存器的地址为：$015C。它是16位只读寄存器，高位D15~D11无意义。RXBEFP位为D10~D0，指出写到接收缓冲区B的最后一个字节的地址偏移量。当RXBCIF=1时，

RXBEFP的值才有效。

(18) 发送缓冲区帧结束指针寄存器（Transmit End-of-Frame Pointer，TXEFP）

该寄存器的地址为：$015E。它是16位寄存器，高位D15~D11只读无意义。TXEFP位为D10~D0，可读/写，指出写到发送缓冲区的最后一个字节的地址偏移量。当TXACT=1时，不能TXEFP的值。

(19) 组播HASH表寄存器（Multicast Hash Table，MCHASH）

该组寄存器的地址为：$0160~$0166。每个寄存器为16位，对接收到的组播帧进行地址识别处理。Hash表算法计算如下：48位目的MAC地址映射到64位中的1位上。将48位地址进行32位CRC计算，取出最高6位（该值在0～63之间）。当接收到的组播帧按照该算法计算的结果在MCHASH对应的位被置1，则接受该帧。

(20) MAC地址寄存器（MAC Unicast Address，MACAD）

该组寄存器的地址为：$0168~$016C。每个寄存器为16位，复位后只能写一次，EMAC地址必须是唯一的。EMAC地址通常固化在非挥发性内存中，用户软件初始化时被拷贝到MAC地址寄存器。

(21) 杂项寄存器（Miscellaneous，EMISC）

该寄存器的地址为：$016E。它是16位寄存器，包括2个字段域：INDEX（D15~D13）和MISC（D10~D0）。该寄存器主要给出EMAC使用的内部计数器个数。INDEX域可读/写，MISC只读。

INDEX：MISC的索引位，用于选择何种计数器将在MISC位上读出。

2.EMAC模块收发器

NE64有2个接收缓冲区和一个发送缓冲区，在初始化的时候定位在RAM空间中，起始地址和缓冲区大小由BUFCFG寄存器的BUFMAP段定义。接收缓冲区中存放以太帧的目的MAC地址、源MAC地址、长度/类型、数据域和CRC校验段。而发送缓冲区与接收缓冲区相比不包含CRC校验段，这个段在发送时由发送器自动添加。

接收以太帧首先要使能EMAC模块，这样一旦缓冲区A和缓冲区B中的一个完成中断标志被清零就开始接收以太帧。如果两个缓冲区都为空则以太帧优先放在缓冲区A中。当缓冲区完成中断标志都被置位时，即无空闲的接收缓冲区，这时不接收以太网数据。

其次，接收以太帧首先需要通过地址匹配。如果接收模式设为混杂模式，将接收所有以太帧。本书将接收模式设置为非混杂模式接收所有广播地址帧，组播地址帧以及唯一地址帧。

单播（Unicast）传输在发送者和每一接收者之间实现点对点网络连接。如果一台发送者同时给多个的接收者传输相同的数据，也必须相应的复制多份的相同数据包。如果有大量主机希望获得数据包的同一份拷贝时，将导致发送者负担沉重、延迟长、网络拥塞；为保证一定的服务质量需增加硬件和带宽。

组播（Multicast）传输在发送者和每一接收者之间实现点对多点网络连接。如果一

台发送者同时给多个的接收者传输相同的数据，也只需复制一份的相同数据包。它提高了数据传送效率。减少了骨干网络出现拥塞的可能性。

广播（Broadcast）传输是指在IP子网内广播数据包，所有在子网内部的主机都将收到这些数据包。广播意味着网络向子网每一个主机都投递一份数据包，不论这些主机是否乐于接收该数据包。所以广播的使用范围非常小，只在本地子网内有效，通过路由器和交换机网络设备控制广播传输。

最后，要进行长度/类型位的匹配，满足了设定的条件时才将以太帧放入接收缓冲区，包括是否接收ARP帧、Ipv4、Ipv6和长度信息等。

这些功能在EMAC正确初始化设置后由模块自动完成。下列情况会产生EAMC模块接收错误，当接收错误中断允许时置接收错误中断标志位。

① 接收的以太帧长度超出1518个字节（包括以太帧首部及校验）；

② MII接口收到PHY模块发出的传输媒介错误；

③ 当长度/类型位的值等于46，而数据域的长度小于46个字节；

④ 以太帧CRC校验出错；

15.3 链路层

在TCP/IP协议族中，链路层主要有三个目的：为IP模块发送和接收IP数据报；为ARP模块发送ARP请求和接收ARP应答；为RARP发送RARP请求和接收RARP应答。

15.3.1 链路层收发数据

TCP/IP支持多种不同的链路层协议，这取决于网络所使用的硬件，如以太网、令牌网、X.25、帧中继、ATM等，在本书中介绍的是以太网。以太网这个术语一般是指数字设备公司（Digital Equipment Corp.）、英特尔公司（IntelCorp）和Xerox公司在1982年联合公布的一个标准。它是当今TCP/IP采用的主要的局域网技术。它采用一种称作CSMA/CD的媒体接入方法，其意思是带冲突检测的载波侦听多路接入（Carrier Sense, Multiple Access with Collision Detection）。

1.以太帧格式

在以太网上传输的数据称为以太帧，其格式如表15-5所示。它的传输必须经过以太帧头的封装/解封装，内容包括目的MAC地址、源MAC地址和长度/类型。其中，前导位、帧起始位和校验码由硬件自动添加/删除，与上层协议无关，因此以太网驱动程序只需要处理其他字段。

表 15-5 以太帧结构

7字节	1字节	6字节	6字节	2字节	46~1500字节	4字节
前导位	帧起始位	目的MAC地址	源MAC地址	长度/类型	数据域	CRC校验

MAC（Media Access Control, 介质访问控制）地址是识别LAN（局域网）节点的

标识。也就是说，在网络底层的物理传输过程中，是通过物理地址来识别主机的，它一般也是全球唯一的。在以太网中，物理地址是48bit（比特位）的整数，如：F0-4E-77-8A-35-1D，前24位是由生产网卡的厂商向IEEE申请的厂商地址，后24位由厂商自行分配，这样的分配使得世界上任意一个拥有48位MAC地址的网卡都有唯一的标识。MAC广播地址是全1，即FF-FF-FF-FF-FF-FF。MAC地址中，第一字节最低位为1是多播地址，为0表示唯一地址；次低位为1表示局部唯一地址，为0表示全球唯一地址。在表15-5中，目的MAC地址表明了以太帧要发送的网络节点，而紧接其后的源MAC地址表明了以太帧是从哪个网络节点发出来的。

长度/类型位的值小于等于1500字节时，表示的是数据域里面的实际数据字节数。当接收以太帧时，将实际的数据域的数据字节数和长度/类型位的值进行比较，如果不匹配则报告错误。当长度/类型位的值大于等于1536字节时，这个域的值表示的是数据域内数据的类型，比如0x0800，表示数据域内的是IP数据报。如果长度/类型位的值大于1500而小于1536，则此以太帧无效。

数据域这里是指IP数据报。以太网的最大传输单元(MTU)是1500字节，如果某个IP数据报超出这个界限，则需要将数据报进行分段，在嵌入式解决方案中，为了简化协议我们不使用超过1500字节的数据报，以避免分组。数据域的最小长度是46字节，当IP数据报小于这个值，则需要填充到46字节。通信时IP数据报可以根据其首部的长度字段来判断那些是需要的数据，那些是填充的数据。

2.EPHY模块的初始化

EPHY模块的初始化步骤：

① 配置PLL，根据EPHY模块的运行要求，将芯片工作频率设为25MHz；

② 关闭EPHY的PHY时钟，直到EPHY和EMAC完全初始化再打开；

③ 配置EPHY地址，使用内部EPHY；

④ 禁止自动协商；

⑤ 启动EPHY模块，只有使能EPHY，才能实现EMAC和EPHY之间的MII操作；

⑥ 开放EPHY中断；

⑦ 配置EPHY的中断、传输速度、双工模式、流控、EPHY自动协商：通过EMAC模块的MII管理接口配置EPHY的MII寄存器；

⑧ 启动EPHY时钟，EPHY模块开始工作。

需要说明的是，EMAC模块初始化必须在第⑤和第⑥步之间实现。

3.EMAC模块初始化

EMAC模块初始化步骤：

① 复位EMAC模块；

② 设置MDC时钟频率，在总线频率为25MHz时可得2.5MHz的MDC时钟频率；

③ 配置以太帧的收发缓冲区及最大接收帧长度，此时以太帧接收缓冲区的大小为1.5K字节，最大接收帧长度为1536字节；

④ 设置MAC地址。因为该寄存器只能在NE64复位后写入一次，所以为了能实现

在线修改MAC地址，需要在网络初始化前将设定的MAC地址写入FLASH空间，然后在MAC初始化时将FALSH空间的对应值写入到MAC地址寄存器中。6字节的MAC地址根据最高字节的最低位分为唯一地址(最低位为0)和组播地址(最低位1)，本书设为唯一地址；

⑤ 设置接收的信包类型，接收所有类型的帧，否则如果类型的相应位为1表示接收该类型的帧，为0表示不接收该类型的帧；

⑥ 设置接收帧的MAC地址类型，接收所有广播地址帧，组播地址帧以及唯一地址帧，未使用流控帧；

⑦ 使能EMAC模块，手工设置传输速度、半双工模式、10BASE-T，启动EMAC；

⑧ 启动EMAC中断，允许发送完成、过量冲突溢出、延迟中断；允许缓冲区A/B接收完成中断；允许接收到流量控制帧中断、帧接收错误中断以及缓冲区A/B接收溢出中断。

4.EMAC的MII接口子程序设计

EPHY子模块的MII寄存器不占用存储器空间，因此不能直接访问。在EMAC子模块中提供了MII管理寄存器专门用于MII寄存器的访问。对某个MII寄存器进行写操作可以通过如下过程完成：首先，通过MII的操作完成标志位判断MII是否忙，如果忙，则等待其变成空闲状态；其次，MII空闲时，设置待写入的EPHY地址以及MII寄存器地址，接着将数据存放至MII数据写入寄存器；第三，执行MII写操作指令，等待执行完成；最后，清除MII操作完成标志位。一般地，写完之后需要读出MII寄存器的数据，确保写入成功。读某个MII寄存器的过程与写过程的不同之处在第二步和第三步：第二步，当MII空闲时，设置待读的EPHY地址以及MII寄存器地址；第三步，执行MII读操作指令，等待执行完成，然后将MII数据读出寄存器中的数据存放至内存空间。下面是MII读写子程序。

```
//-----------------------------------------------------------------------*
//MIIwrite:对MII写数据程序                                               *
//功  能:中断方式从缓冲区A或B接收数据并复制到输出队列                    *
//入口: _mpadr,EPHY设备编号; _mradr,MII寄存器地址                        *
//出口: * mwdata需写入的数据                                             *
//返  回: 操作成功返回0xff，操作失败返回0                                *
//说  明:                                                                *
//-----------------------------------------------------------------------*
INT8U MIIwrite(INT8U _mpadr, INT8U _mradr, INT16U _mwdata)
{
    INT16U temprdata;
    //1.读寄存器判断是否MII忙，若忙则返回不做这次写操作
    if ((MCMST & 0x20) != 0x00)  //MCMST.BUSY=1
    {
        return 0;                //MII忙,返回0
    }
    //2.MII空闲，则开始写操作
    MPADR = _mpadr;              //设置地址寄存器MPADR
```

```
    MRADR = _mradr;              //设置管理寄存器MRADR
    MWDATA = _mwdata;            //设置数据寄存器MWDATA
    //执行MII写操作指令MCMST.OP=10读;=01写
    MCMST &= 0x3f;
    MCMST |= 0x40;
    //3.等待MII写完成
    while ((IEVENT&0x80) == 0x00);
    //等待直到MII传送完成IEVENT.MMCIF=1;
    IEVENT |= 0x80; //写1清0,清除MII完成标志位
//等待读取操作结束，temprdata变量返回读取的数据内容
    //0x10=PHY中断控制寄存器
    //PHY_ADDRESS=0在ne64config.h中定义
    while ( ! (MIIread(PHY_ADDRESS, 0x10, &temprdata)));
    return 0xFF;          //操作成功
}

//MIIread:读MII数据程序-------------------------------------------------*
//功 能:中断方式从缓冲区A或B接收数据并复制到输出队列                    *
//入 口:mpadr起始基地址 _mradr寄存器对象                                 *
//出 口: *mrdata读出的数据                                               *
//返 回:操作成功返回0xff，操作失败返回0                                  *
//说 明:                                                                 *
//-----------------------------------------------------------------------*
INT8U MIIread(INT8U _mpadr, INT8U _mradr, INT16U * _mrdata)
{
    //1.读寄存器判断是否MII忙，若忙则返回不做这次写操作
    if ((MCMST & 0x20) != 0x00)  //MCMST.BUSY=1
    {
        return 0;                //MII忙,返回0
    }
    //2.MII空闲，则开始读操作
    MPADR = _mpadr;              //设置地址寄存器MPADR
    MRADR = _mradr;              //设置管理寄存器MRADR
    //执行MII读操作指令MCMST.OP=10读;=01写
    MCMST &= 0x3F;
    MCMST |= 0x80;

    //3.等待MII读完成
    //等待直到MII传送完成IEVENT.MMCIF=1;
    while ((IEVENT & 0x80) == 0x00);
    //将读出的数据接收存放
    *_mrdata = MRDATA;
    IEVENT |= 0x80; //写1清0,清除MII完成标志位
    return 0xff;    //操作成功
}
```

5.第一个测试实例：网络连接指示

在WINDOWS系统中，可以通过网卡图标的状态来判断本机是否连接到一个网路

中。如果连接成功，则网卡显示已连接；当连接不成功时，在网卡的图标上会出现一个红色的叉号，提示网络电缆没有插好。这样用户和应用程序可以明确的知道当前本机的网络连接情况。

在嵌入式网络设备中，上层程序也需要了解当前的设备当前的网络连接状况。在NE64的EPHY模块中也可以提供相应功能，首先设置MII寄存器中的ICR(Interrupt Control Register)寄存器LCIE位为1，允许连接状态变化产生EPHY中断。这样在连接状态发生变化时，比如网卡的插拔，将产生EPHY中断，用户程序可以通过判断MII寄存器中的PSR(Proprietary Status Register)寄存器的LNK位状态可以获得当前的网络连接状况。

网络物理连接测试工程文件列表见图15-5。

图15-5 网络物理连接测试工程文件

主函数如下表所示。使用交叉网线将NE64设备和PC相连，如果连接成功则PC网卡显示连接成功；NE64设备通过串口发送“V”，否则发送“F”。

```
//-------------------------------------------------------------------------*
//工 程 名:EPHY                                                            *
//硬件连接:与PC交叉网线直连,并接上串口SCI1                                 *
//程序描述:以太网模块初始化。当PC网卡显示“已连接上”表示成功               *
//         NE64如果网连接成功则发送“V”；否则发送“F”。                     *
//目    的:freescale 9s12 NE64系列程序                                     *
//说    明:                                                               *
//注    意:  具体内容见光盘代码                                           *
//-------《嵌入式系统－使用HCS12微控制器的设计与应用》教学实例-----------*
 . . . . . . //限于篇幅问题，只给出主要的部分，详细代码见光盘
//[主函数]
int  main()
{
    if(*(INT8U *)pData == 0xFF)
    {
        Flash_Write_Nword(0, pData, (INT16U)(ip_address), 2);    //本地IP地址
        Flash_Write_Nword(0 ,pData + 4 ,(INT16U)(ip_gateway) ,2);//网关地址
        Flash_Write_Nword(0 ,pData + 8 ,(INT16U)(ip_netmask) ,2);//子网掩码
        Flash_Write_Nword(0 ,pData + 12 ,(INT16U)(hard_addr) ,3);//MAC地址
    }
    //1.设置全局地址变量
    for(i = 0; i < 4; i++)
    {
        localmachine.localip[i] = *((INT8U *)(pData + i));     //本地IP地址
        localmachine.serverip[i] = *((INT8U *)(pData + 4 + i));//服务器IP地址
        localmachine.defgw[i]  = *((INT8U *)(pData + 4 + i));  //网关地址
        localmachine.netmask[i] = *((INT8U *)(pData + 8 + i)); //子网掩码
    }
    for(i = 0; i < 6; i++)
        localmachine.localHW[i] = *((INT8U *)(pData + 12 + i)); //MAC地址
    //2.网络模块初始化
    EtherInit(0,0,0,0,0,1,0);    //3.2初始化所有网络层,在netinit.c中
    ENABLE_INTERRUPTS;           //开中断
    //主循环
    while(1)
    {
       //即时更新除MAC地址外的网络参数
       INT16S pData;                  //网络参数FLASH存放地址
       pData = Flash_Data;
       for(i = 0; i < 4; i++)
       {
         localmachine.localip[i] = *((INT8U *)(pData + i));    //本地IP地址
         localmachine.serverip[i] = *((INT8U *)(pData+4+i));   //服务器IP地址
         localmachine.defgw[i]  = *((INT8U *)(pData + 4 + i)); //网关地址
         localmachine.netmask[i] = *((INT8U *)(pData + 8 + i));//子网掩码
       }
       //gotlink=1连接成功
       if (gotlink)
       {
```

```
            SCISend1('V');
        }
        else
        {
            SCISend1('F');
        }
        DelaySub(0x00FF);
    }//while(1)
}
//Phy_link()-----------------------------------------------------------
//功  能:Phy层中断处理
//参  数:无
//返回值:影响全局变量gotlink,gotlink=1:物理连接成功,否则没有链接
//---------------------------------------------------------------------
void  __attribute__ ((interrupt)) Phy_link(void)
{
  unsigned int mymrdata, isr_read, temp, temp2;
  unsigned int none_of_these = 0;
  while ( !(MIIread(0x00, 0x10, &isr_read)) );
  //连接变化
  if ((isr_read & 0x0010) == 0x0010)
  {
     none_of_these = none_of_these + 1;
     while ( !(MIIread(0x00, 0x11, &mymrdata)) );
     if ((mymrdata &  0x4000) ==  0x4000)
     {
         gotlink = 0;
             PTL|=0x01;   //turn LED off
         PTL|=(1<<1); //turn LED off
         PTL|=(1<<2); //turn LED off
         PTL|=(1<<3); //turn LED off
         PTL|=(1<<4); //turn LED off
     }
     else
     {
         gotlink = 1;
         PTL&=0xfd;
         while ( !(MIIread(0x00, 0x11, &mymrdata)));
     }
  }
  EPHYSR|=0x01;  //清Ephy 中断标志位
}
```

6.以太帧的发送和接收

在NE64的EPHY和EMAC模块的正确初始化后，就可以编写接收和发送以太帧的程序。这样已经完成了以太网的基本通信功能，即实现以太网节点间的数据交换，如果要完成更多的功能则需要在上层添加通信协议。

(1) 以太帧的发送

在NE64中发送一个以太帧，必须将该帧内容写入至EMAC模块的发送缓冲区(TX缓冲区)，然后再通过发送命令将其发送出去，接下来的工作由下层硬件完成。

与以太帧的发送相关的寄存器包括发送缓冲区帧结束指针寄存器(TXEFP)、发送控制和状态寄存器(TXCTS)。其中发送缓冲区帧结束指针寄存器标明了发送帧的最后一个字节的偏移量；而发送控制和状态寄存器的D1和D0位(TCMD位)为发送命令位，通过这两位可以发送3个不同的命令：START、PAUSE和ABORT。START表示开始发送在发送缓冲区的帧；PAUSE表示发送一个由硬件产生的PAUSE帧；ABORT命令可以中止任何帧的发送。TCMD=0，保留，忽略；TCMD=1，START，启动发送缓冲区帧；TCMD=2，PAUSE，启动PAUSE帧（只适用全双工）；TCMD=3，ABORT，取消发送。下面是发送一个以太帧的子程序。

```
//-----------------------------------------------------------------------*
//程序名:EtherStartFrameTransmission                                     *
//功  能:将缓冲区中准备好的数据发送出去                                  *
//入口:datalen 数据长度                                                  *
//出口:无                                                                *
//调用:无                                                                *
//返回:无                                                                *
//说明:发送前先规范数据帧长度,然后定义发送队列中最后一帧数据所在的地址偏移*
//       最终启动寄存器直接将数据缓冲区的数据发送出去.                  *
//-----------------------------------------------------------------------*
void EtherStartFrameTransmission(INT16U datalen)
{
    void *test;
    int i;
    datalen = datalen +14;
    //1.检查数据帧长度
    #if EMAC_TX_SZ > 1514
        //如果超出长度限制
        if (datalen > 1514) datalen = 1514;
    #else
        //如果超出发送缓冲区长度限制
        if (datalen > EMAC_TX_SZ) datalen = EMAC_TX_SZ;
    #endif
    //如果长度过短
    if (datalen < 60) datalen = 60;
    //2.设置发送队列中最后一帧数据所在的地址偏移
    TXEFP = datalen - 1;
    //3.等待发送队列空闲
    while ((TXCTS & 0x80) != 0x00);
    //4.启动发送数据指令
    TXCTS &= 0xFC;              //0b11111100 保护高六位,清低二位为全0
    TXCTS |= 0x01;              //设置为0x01即1
}
```

(2) 以太帧的接收

判断以太帧的接收有两种方法：查询法和中断法。由于中断法有更好的执行效率，本书使用了中断法接收以太帧。由于NE64有两个接收缓冲区A和B，因此到达的帧可能存储在A缓冲区也可能存储在B缓冲区，所以中断矢量也有两个：A缓冲区接收完成中断和B缓冲区接收完成中断，其矢量地址分别是$FFB2和$FFB4。中断服务程序如下。

```
//-------------------------------------------------------------------------*
///emac_rx_b_a_c_isr -EMAC数据缓冲区A接收满中断                              *
//功  能: EMAC数据缓冲区A接收满中断                                         *
//说  明:                                                                  *
//-------------------------------------------------------------------------*
void  __attribute__ ((interrupt)) emacrxbac_isr(void)
{
    //SCISend1(0x07);    //1个信号
    //RX_POLL_MODE=0,  1=查询方式;0=使用中断方式
    #if RX_POLL_MODE
    #else
        //中断方式接收缓冲区中的数据 并进行处理
        //emacFIFOa在ne64drive.h中定义
        NE64Receive (emacFIFOa, RXAEFP, IEVENT & (0x200));
        #if (USE_SWLED && ACTLED)
            //点亮ACTLED灯
            PTL &= 0xFE;
        #endif
    #endif//RX_POLL_MODE
}

//-------------------------------------------------------------------------*
///emac_rx_b_b_c_isr -EMAC数据缓冲区B接收满中断                              *
//功  能: EMAC数据缓冲区B接收满中断                                         *
//说  明:                                                                  *
//-------------------------------------------------------------------------*
void  __attribute__ ((interrupt)) emacrxbbc_isr(void)
{
    //SCISend1(0x08);    //1个信号
    //RX_POLL_MODE=0,  1=查询方式;0=使用中断方式
    #if RX_POLL_MODE
        //Polled mode used
    #else
        //中断方式接收缓冲区中的数据 并进行处理
        //emacFIFOa在ne64drive.h中定义
        NE64Receive (emacFIFOb, RXBEFP, IEVENT & (0x100));
        #if (USE_SWLED && ACTLED)
            //点亮ACTLED灯
            PTL &= 0xFE;
        #endif
    #endif//RX_POLL_MODE
}
```

无论是A缓冲区还是B缓冲区接收到数据，处理方法是一样的，都是将接收到的数据帧读出来，再进行相应的处理。在数据读取过程中使用到接收缓冲区帧结束指针寄存器(RXAEFP和 RXBEFP)，它们指出接收到以太帧的最后一个字节的偏移量。在完成以太帧接收操作后，需要清除中断标志位。NE64中清除标志位的方法是通过向IMASK寄存器的相应位写0。下面是接收一帧的子程序。

```
//中断方式接收数据处理子程序
//-------------------------------------------------------------------------*
//程序名:NE64Receive                                                       *
//功  能:中断方式接收接收缓冲区中的数据只处理以太网帧头                    *
//入口: *PktBuffer缓冲区指针  len缓冲区数据长度 flags状态标志位            *
//出口: 无                                                                 *
//内部调用:memcopy(INT8U * buf , INT8U *scr,INT16U len)                    *
//         mENQUEUE()                                                      *
//返  回:操作结束返回0                                                     *
//说  明:只处理以太网帧头取出协议类型和MAC地址                             *
//-------------------------------------------------------------------------*
INT16U NE64Receive (void *PktBuffer, INT16U len, INT16U flags)
{
    MBUF mp;
    mp.data        = (INT8U *)PktBuffer;    //指针位置
    mp.working_ptr = mp.data;               //数据位置
    mp.len         = len;
    mp.status      = (MBUF_NOTEMPTY | flags);
    //关闭相应接收缓冲区中断使能，防止数据被随后的帧覆盖
    if (mp.status & 0x200)  //0x200=IEVENT_RXACIF_MASK
    {
        IMASK &= 0xFDFF;   //缓冲区A接收中断禁止
    }
    else
    {
        IMASK &= 0xFEFF;   //缓冲区B接收中断禁止
    }
    //ETH_ADDRS_LEN=6 协议静态变量
    //取出目标MAC地址和源MAC地址
    (void)memcopy (received_frame.destination, mp.data, 2*ETH_ADDRS_LEN);
    //实际接收的以太帧长度
    received_frame.frame_size = mp.len;
    //以太帧中的长度/类型字段
    received_frame.protocol   = (*((INT16U *)&mp.data[12]));
    //ETH_HDR_LEN=14 以太帧首部大小
    received_frame.buf_index  = ETH_HDR_LEN;
    //读缓冲数组的值
    (void)mENQUEUE (&mp);
return 0;}
```

经过这样的处理后，接收到的以太帧的首部信息就保存在received_frame结构体中。上层的协议可以通过判断received_frame的内容再做相应的处理。比如

received_frame.protocol中的值为0x0806则表示为ARP分组，0x0800表示IP数据报。

7.第二个测试实例：以太帧的发送和接收

测试以太帧接收的工程名为FRAME，工程列表同表15-6。main.c文件的主循环while()内容修改如下。

当gotlink=1连接成功后，通过NETWORK_CHECK_IF_RECEIVED()以太帧处理函数来判断是否接受到以太帧。如果是则通过串口发送“V”，并将以太帧首部的类型字段received_frame.protocol也通过串口发送给PC；连接成功而未收到以太帧则发送“N”；网络不通发送“F”。

将NE64设备B(192.168.149.137)与主机A(192.168.149.153)用交叉网线直连，在windows下点击运行，输入“CMD”进入DOS界面，再运行“PING 192.168.149.137”，则会发现NE64设备通过串口发送“V”和类型字段“08 06”，表明其收到以太帧，且为ARP请求分组。ARP请求的含义将再在下一节具体说明。

```
//-------------------------------------------------------------------------*
//工 程 名:FRAME                                                            *
//硬件连接:与PC交叉网线直连,并接上串口SCI1                                  *
//程序描述:接收以太帧，NE64如果网连接成功则发送“V”并回送以太帧首部类型；*
//         连接成功而未收到以太帧则发送“N”；网络不通发送“F”。             *
//目    的:freescale 9s12 NE64系列程序                                      *
//-------《嵌入式系统－使用HCS12微控制器的设计与应用》教学实例------------*
While(1)
{
. . . . . //限于篇幅问题，只给出主要的调用部分，详细代码见光盘
        if (gotlink)
        {
            //处理接收到的以太网帧
            if( NETWORK_CHECK_IF_RECEIVED() == 1 )
            {
                SCISend1(0x56);//发送“V”
                SCISend2(received_frame.protocol);//发送以太帧首部的类型
                NETWORK_RECEIVE_END();//清以太帧缓冲区
            }
            else
                SCISend1(0x4E);//发送“N”
        }
        else
            SCISend1(0x46);   //发送"F"
        DelaySub(0x00FF);
    }//while(1)
```

15.3.2 ARP协议

1.ARP帧格式

在TCP/IP协议栈的网络层中以IP地址标示不同的主机。以太网中，ARP的功能是在

32位的IP地址和MAC地址之间提供动态映射。

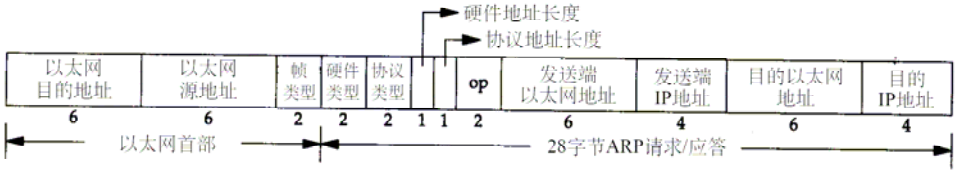

图15-6 用于以太网的ARP请求或应答分组格式

（1）硬件类型：表示网络上硬件的类型，以太网表示为1。

（2）上层协议类型：表示网络层协议的类型，如果是IP协议，为800。

（3）硬件地址长度：MAC地址的长度，如果是以太网，为6。

（4）协议地址长度：上层协议的长度，如果是IPV4时为4。

（5）操作号：表示操作内容的数值。1为ARP请求；2为ARP响应；3为RARP请求；4为RARP响应。

（6）ARP包中的目的MAC地址和目的IP地址分别为查询对象系统的MAC地址和IP地址。这里要用全0来填充MAC地址部分。

2.第三个测试实例：ARP请求与应答

图15-7 ARP协议实现工程文件

ARP协议实现的工程列表如图15-7所示。主函数中添加ARP相关部分，在主循环的协议类型判断中，添加对ARP请求的处理。

```
//-------------------------------------------------------------------------*
//工 程 名:ARP                                                             *
//硬件连接:与PC交叉网线直连                                                *
//程序描述:ARP响应，PC方先通过“arp -d”删除ARP表，再PING 192.168.149.137 *
//         然后检查“arp -a”,能看到多了一条192.168.149.137的记录，其MAC  *
//         地址为“f0 4e 77 8a 35 1d”                                       *
//目    的:freescale 9s12 NE64系列程序                                     *
//说    明:                                                               *
//注    意:  具体内容见光盘代码                                           *
//-------《嵌入式系统－使用HCS12微控制器的设计与应用》教学实例------------*
    while(1)
```

```
{
 . . . //限于篇幅问题，只给出主要的调用部分，详细代码见光盘
 if (gotlink)
    {
       //处理接收到的以太网帧
       if( NETWORK_CHECK_IF_RECEIVED() == 1 )
       {
         switch( received_frame.protocol)//received_frame在ethernet.h中定义其结构
         {
            case PROTOCOL_ARP:
            //1.若为ARP请求则作出ARP响应
                process_arp (&received_frame);//在arp.c中
                break;
            default:break;
          }//switch
        //3.丢弃剩余帧
        NETWORK_RECEIVE_END();
        }//NETWORK_CHECK_IF_RECEIVED
      }//gotlink
    }//while(1)
}
```

当一台PC主机A其IP地址为192.168.149.153，与一台NE64设备B其IP地址为192.168.149.137通过交叉线连接后，主机A在向设备B第一次发送PING命令前需要知道B设备的MAC地址，于是发送ARP请求。当主机A已经知道设备B的MAC地址后则不会发送ARP请求，这时需要清空主机A内存中的ARP表，才能再次发送ARP请求，具体操作见下一节。

ethereal是一个可以用于windows下的一个网络协议分析器，而且是免费的。Ethereal本身并不能抓包，它只能用来解析数据包；要抓取数据包，它需要借助于PCap。Pcap在windows下面的实现称作Winpcap。Ethereal的安装文件见随书光盘。

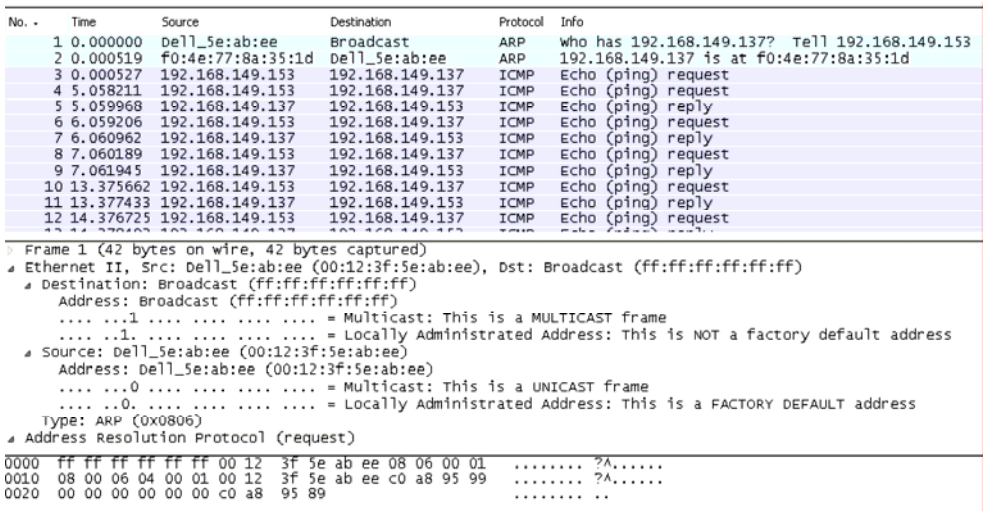

图15-8 使用ethereal捕捉的ARP请求以太帧

图15-8是使用ethereal的分析的结果。在帧的开头是“ff ff ff ff ff ff”表示这是一个以太网的广播帧；随后的“00 12 3f 5e ab ee”指示的是主机A的MAC地址；“08 06”说明这是一个ARP请求；“00 01”表明是通过以太网传输；“08 00”是指使用IP协议；“06 04”对应的是MAC和IP地址的字节数；“00 01”表示是ARP请求，在图15-9中的ARP应答使用的是“00 02”；“00 12 3f 5e ab ee”和“c0 a8 95 99”是主机的MAC地址和十六进制表示的IP地址；注意后面的MAC地址为空“00 00 00 00 00 00”最后是设备B的IP地址“c0 a8 95 89”。主机A通过ARP请求帧向网内的节点询问“我是主机192.168.149.153，拥有IP地址192.168.149.137的主机，请告诉我你的MAC地址。”

当设备B接受到主机A发送的以太网广播帧后，通过以太帧首部判断这是一个ARP请求报文，并且目的IP地址就是自己的MAC地址，于是就向主机A发送了一个ARP应答报文如图15-9，表明自己的MAC地址是“f0 4e 77 8a 35 1d”。注意这时的源是主机是设备B而和目的主机为A。这样主机A就得到了设备B的MAC地址，并能够在以太网上向其发送其他协议的数据帧。

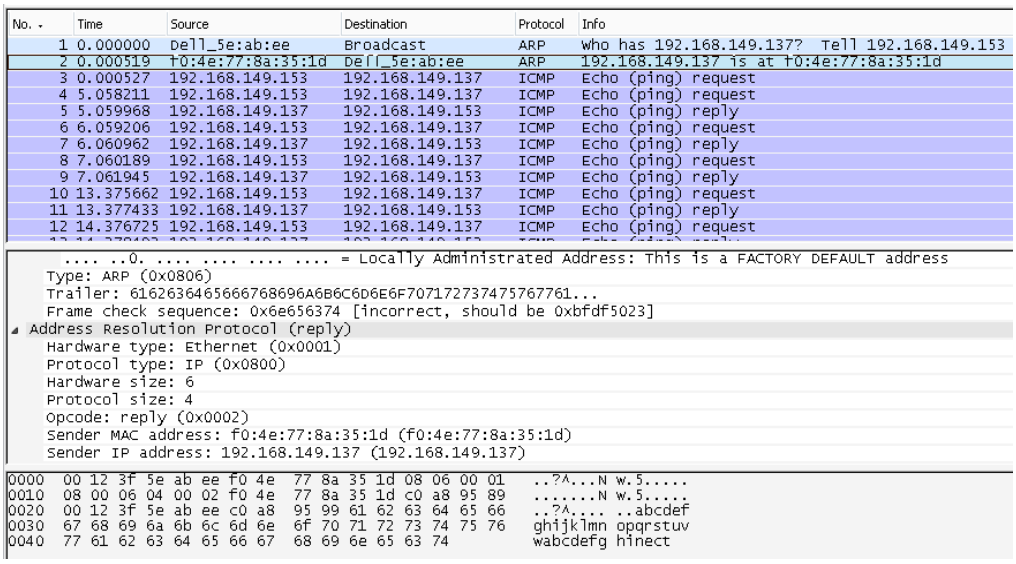

图15-9 使用ethereal捕捉的ARP应答以太帧

3.ARP缓存表

ARP协议实现了IP地址与以太网MAC地址的转换。计算机在内存中建立一个称为ARP缓冲区的表格来存放IP地址和其对应的MAC地址，在Windows xp中可以通过“arp -a”来显示当前的ARP表。如图15-10所示，首先执行“arp -d”删除缓冲区的内容，然后检查发现只有网关的选项，且MAC地址全0，表示不可达，接着PING设备B，由于此时主机A不知道设备B的MAC地址，且判断设备B与其在一个子网内，所以会以广播方式在该子网内发送ARP请求，再检查ARP缓冲区的时候发现已经多了一条“192.168.149.137 f0-4e-77-8a-35-1d”，这就是设备B的IP地址和MAC地址。其他网络协议则可以通过查找ARP缓冲区来确定目的主机的MAC地址。注意图15-10所示实例中，设备B已经包含IMCP处理函数，所以显示连通；如果不包含IMCP，则会显示PING不通，但是不影响ARP请求的结果。

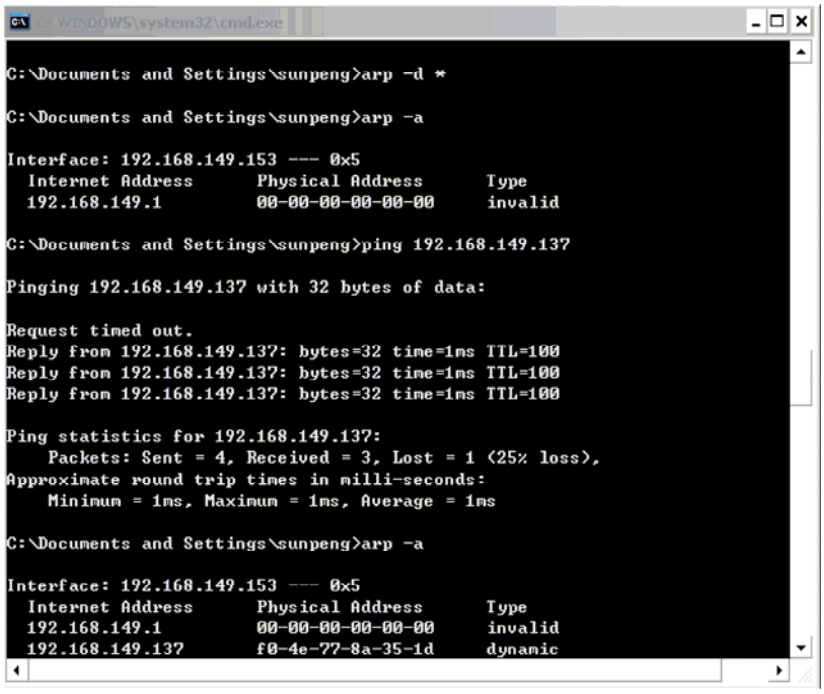

图15-10 ARP缓冲区

在嵌入式网络设备中也需要建立这样的ARP缓冲区，本书实例中在arp.h文件中定义了相关的数据结构。可以看到在 tARP_CACHE 结构体中，hwaddr存储MAC地址，ipaddr存放IP地址，最后的hitcounter表示碰撞计数。ARP_CACHE_SZ定义了在内存中维护3组对应关系。

```
//ARP报文头中常量定义
#define  MAXHWALEN  6     //最大的硬件地址长度(6=Eth)
#define  MAXPRALEN  4      //最大的协议地址长度(4=IPv4)
#define AR_HARDWARE 0x0001 // 以太网硬件类型
#define ARP_ETHCODE 0x0806 // ARP类型
#define  ARP_REQUEST  1    // ARP请求
#define  ARP_REPLY    2    // ARP响应
//ARP cache的目录数量设置
#define ARP_CACHE_SZ 3
//ARP缓冲区结构
typedef struct
  {
   tU08 hwaddr[MAXHWALEN];  //MAC地址
   tU08 ipaddr[MAXPRALEN];  //IP地址
   tU08 hitcounter;         //碰撞计数
  }tARP_CACHE;
```

对ARP缓冲区的操作有3中，查询、写入和初始化。初始化将表中各项清零，当成功的接受到ARP应答时将相应的表项写入到缓冲区中，高层程序需要进行通信时再调用查询函数检查目的IP地址是否再缓冲区，如果不在则发送ARP请求。由于只能有3组

表项，所以当需要写ARP缓冲区而缓冲区满时，则检测碰撞计数最少的表项，将其覆盖。

当嵌入式设备完全是被动通信，即只回送数据到提出请求的主机时，则可以进一步简化ARP协议。由于接收的以太帧中有发送端的MAC地址，因此可以只保留ARP应答，而不使用ARP请求；可以不使用ARP缓冲区，将以太帧中的源MAC地址作为全局变量，在发送时直接写入到目的MAC地址中。

值得注意的是，ARP协议是在一个子网内的机器间运行，不用子网的机器之间是不能运行ARP请求或应答的，如果要进行网络层以上的通信，要通过网关接口。

15.4 网络层

网络层是在TCP/IP协议族中最复杂的一层，主要解决主机到主机的通信问题。网间的数据报可以根据它携带的目的IP地址，通过路由器由一个网络传送到另一个网络。在嵌入式设计中，如果设备只是作为一个网络终端，而不需要具有网关或路由功能时，则只需要实现ICMP协议（检测网络）和IP协议（数据传送）。

15.4.1 ICMP协议和PING程序

1.ICMP协议简介

ICMP协议传递差错报文，并被IP层或更高层协议（TCP和UDP）使用。ICMP报文是在数据报内部被传输的，如图15-11所示。

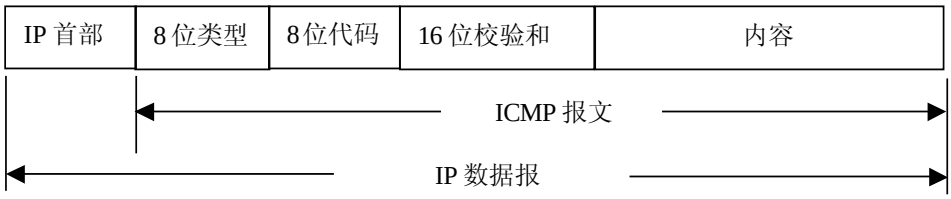

图15-11 封装在IP数据报内的ICMP报文

类型字段可以有15个不同的值，表示不同特定类型的ICMP报文。一些ICMP报文还使用代码字段的值来区分不用的报文。

PING程序通过发送一份ICMP回显请求报文到目的主机，并等待其ICMP回显应答，由此来测试目的主机是否可达。在没有访问控制或防火墙的情况下，可以确定如果不能PING到某台主机，那么就不能在高层的协议上访问该主机，比如TCP等。

当嵌入式设备和PC主机通过交叉线直连后，当PC的网卡显示已连通后，可以通过PING程序来测试是否可以与该设备通信。因为在网络通信中，嵌入式设备大多是被动接受通信请求，所以为了简化协议，本书实例只实现ICMP回显应答服务。

图15-12 ICMP回显请求和应答报文

当类型段为8，代码段为0时，表示是一个ICMP回显请求报文。如果类型段的值变成0则说明这是一个回显应答报文。

2.第四个测试实例：ping命令使用

ICMP实例的工程文件列表如图15-13所示。与PC交叉网线直连,然后PING 192.168.149.137。可以看看应答信息验证有没有ping通。方法与ping普通PC机一样。

图15-13 ICMP协议实现工程文件

主函数如下所示，增加对IP和ICMP的处理部分。

```
//-------------------------------------------------------------------------*
//工 程 名:ICMP                                                            *
//硬件连接:与PC交叉网线直连                                                *
//程序描述:PING 192.168.149.137                                            *
//目     的:freescale 9s12 NE64系列程序                                    *
//-------《嵌入式系统－使用HCS12微控制器的设计与应用》教学实例------------*
. . . . . //限于篇幅问题，只给出主要的调用部分，详细代码见光盘
    //主循环
    while(1)
    {
       . . . . . .
        if (gotlink)
        {
```

```
            //处理接收到的以太网帧
            if( NETWORK_CHECK_IF_RECEIVED() == 1 )
            {
            //received_frame在ethernet.h中定义其结构
                switch( received_frame.protocol)
                {
                case PROTOCOL_ARP:
                    //1.若为ARP请求则作出ARP响应
                    process_arp (&received_frame);//在arp.c中
                    break;
                case PROTOCOL_IP:
                    //2.若为IP信包则作下一步判断处理
                    len = process_ip_in(&received_frame);
                    if(len < 0) break;
                    switch (received_ip_packet.protocol)
                    {
                    case IP_ICMP:
                        //2.1.若为ICMP信包则返回ICMP报文,完成一次PING命令
                        process_icmp_in (&received_ip_packet, len);
                        break;
                    default:break;
                    }
                default:break;
                }//switch
                //3.丢弃剩余帧
                NETWORK_RECEIVE_END();
            }//NETWORK_CHECK_IF_RECEIVED
        }//gotlink
    }//while(1)
}
```

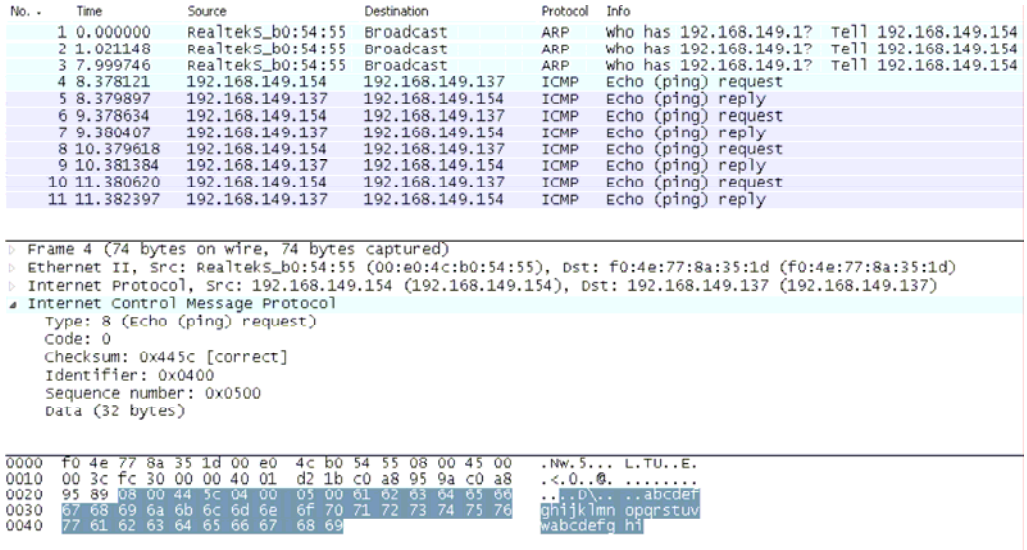

图15-14 Ethereal捕捉的ICMP回显请求报文

图15-10显示了主机C（IP地址192.168.149.154）PING设备B（IP地址192.168.149.137）的成功的显示结果。图15-14是用Ethereal捕捉的报文信息。可以看到在Internet Control Message Protocol（ICMP）中，Type值为8表示这是一个请求报文。当设备B接收到主机A发出的请求后，将Type值变成了0，重新计算校验和，标识符和序号不变，封装成ICMP回显应答报文发送给主机C，Ethereal捕捉的回显应答报文如图15-15所示。

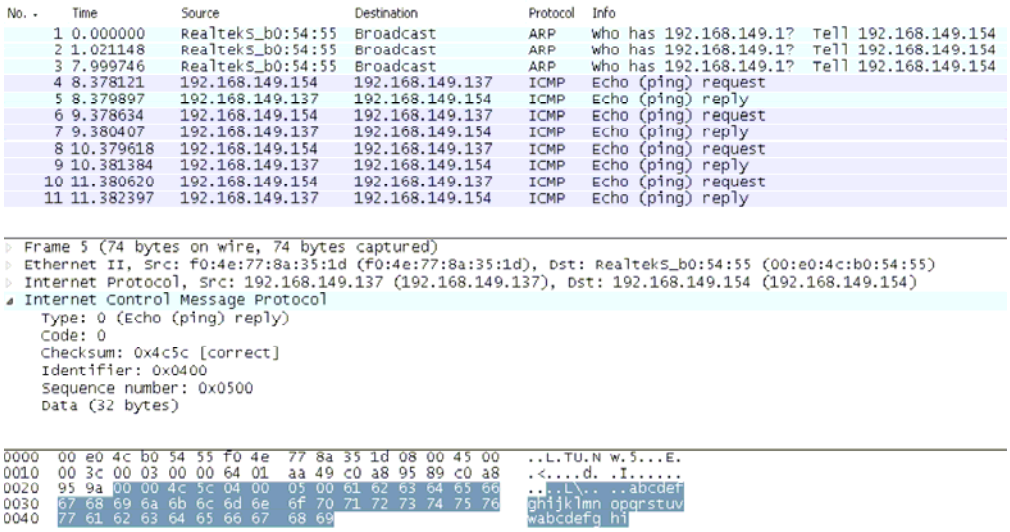

图15-15 Ethereal捕捉的ICMP回显应答报文

15.4.2 IP协议

IP协议的基本功能时提供无连接的数据报传送服务和数据报路由选择服务，但不保证服务的可靠性。IP协议(Internet Protocol，网际协议)是TCP/IP协议族中最核心的协议，

TCP、UDP和ICMP都以IP数据报的格式传输。

1.IP协议术语

(1) IP地址

Internet上的每台主机都有一个唯一的IP地址。IP协议就是使用这个地址在主机之间传递信息，这是Internet能够运行的基础。IP地址的长度为32位，分为4段，每段8位，用十进制数字表示，每段数字范围为1～254，段与段之间用句点隔开。例如192.168.149.1。IP地址有两部分组成，一部分为网络地址，另一部分为主机地址。IP地址分为A、B、C、D、E5类。常用的是B和C两类。IP地址就像是我们的家庭住址一样，如果你要写信给一个人，你就要知道他（她）的地址，这样邮递员才能把信送到，计算机发送信息是就好比是邮递员，它必须知道唯一的“家庭地址”才能不至于把信送错人家。只不过我们的地址使用文字来表示的，计算机的地址用十进制数字表示。

(2) 网段

IP地址由两部分组成，即网络号（Netgwork ID）和主机号（Host ID）。网络号标识的是Internet上的一个子网，而主机号标识的是子网中的某台主机。网际地址分解成两个域后，带来了一个重要的优点：IP数据包从网际上的一个网络到达另一个网络时，选择路径可以基于网络而不是主机。在大型的网际中，这一点优势特别明显，因为路由表中只存储网络信息而不是主机信息，这样可以大大简化路由表。IP地址根据网络号和主机号的数量而分为A、B、C三类：

A类IP地址：用7位（bit）来标识网络号，24位标识主机号，最前面一位为"0"，即A类地址的第一段取值介于1～126之间。A类地址通常为大型网络而提供，全世界总共只有126个只可能的A类网络，每个A类网络最多可以连接16777214台主机。

B类IP地址：用14位来标识网络号，16位标识主机号，前面两位是"10"。B类地址的第一段取值介于128～191之间，第一段和第二段合在 一起表示网络号。B类地址适用于中等规模的网络，全世界大约有16000个B类网络，每个B类网络最多可以连接65534台主机。

C类IP地址：用21位来标识网络号，8位标识主机号，前面三位是"110"。C类地址的第一段取值介于192～223之间，第一段、第二段、第三段合在一起表示网络号。最后一段标识网络上的主机号。C类地址适用于校园网等小型网络，每个C类网络最多可以有254台主机，0和255留作特殊用途。

从上面的介绍我们知道，IP地址是以网络号和主机号来标示网络上的主机的，只有在一个网络号下的计算机之间才能"直接"互通，不同网络号的计算机要通过网关（Gateway）才能互通。

(3) 子网掩码

子网掩码将某个IP地址划分成网络地址和主机地址两部分，以便于IP地址的寻址操作。子网掩码的设定必须遵循一定的规则。与IP地址相同，子网掩码的长度也是32位，左边是网络位，用二进制数字“1”表示；右边是主机位，用二进制数字“0”表示。比如IP

地址“192.168.1.1”和子网掩码“255.255.255.0”。其中，“1”有24个，代表与此相对应的IP地址左边24位是网络号；“0”有8个，代表与此相对应的IP地址右边8位是主机号。这样，子网掩码就确定了一个IP地址的32位二进制数字中哪些是网络号、哪些是主机号。这对于采用TCP/IP协议的网络来说非常重要，只有通过子网掩码，才能表明一台主机所在的子网与其他子网的关系，使网络正常工作。子网掩码有数百种，最常用的两种子网掩码是“255.255.255.0”和“255.255.0.0”。

(4) 网关

网关实质上是一个网络通向其他网络的IP地址。比如有网络A和网络B，网络A的IP地址范围为“192.168.1.1～192.168.1.254”，子网掩码为255.255.255.0；网络B的IP地址范围为“192.168.2.1~192.168.2.254”，子网掩码为255.255.255.0。在没有路由器的情况下，两个网络之间是不能进行TCP/IP通信的，即使是两个网络连接在同一台交换机(或集线器)上，TCP/IP协议也会根据子网掩码(255.255.255.0)判定两个网络中的主机处在不同的网络里。而要实现这两个网络之间的通信，则必须通过网关。如果网络A中的主机发现数据包的目的主机不在本地网络中，就把数据包转发给它自己的网关，再由网关转发给网络B的网关，网络B的网关再转发给网络B的某个主机。所以说，只有设置好网关的IP地址，TCP/IP协议才能实现不同网络之间的相互通信。那么这个IP地址是哪台机器的IP地址呢？网关的IP地址是具有路由功能的设备的IP地址，具有路由功能的设备有路由器、启用了路由协议的服务器(实质上相当于一台路由器)、代理服务器(也相当于一台路由器)。

2.IP首部

IP数据报的格式如图15-16所示。标准的IP首部为20个字节，本书的程序中不对选项段做处理，并且不支持IP分片。

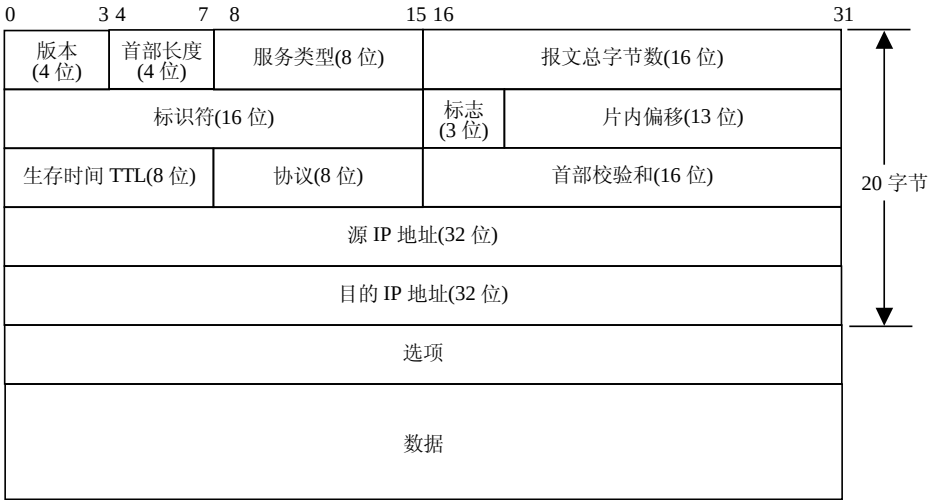

图15-16 IP首部

首先示IP协议的版本信息，为4表示Ipv4，为6则表示Ipv6，这里只使用4；首部长

度表示IP首部占32位的长度，将其乘以4得到首部字节数；服务类型不处理；报文总字节数表示整个IP数据报的长度，将其减去首部长度就可以得到数据长度；标识符唯一标识主机的每一份数据报；标志和片内偏移用于IP分片，这里不处理；生存周期设置了数据报可以经过的最多路由器数，每过一个路由减1，为0则丢弃；协议字段表示示哪个高层协议调用了IP协议；首部校验和不包含数据部分；每个IP数据报都包含源IP地址和目的IP地址；选项字段一般不使用。

3.IP协议接收处理

链路层的以太帧成功接收后，将其数据部分提供IP协议的接收数据报程序处理。IP接收处理程序提取首部的相关信息字段，在校验通过且相关设置符合时，比如IP不分片或数据报是发往本地的等等，则将其数据部分提供给更高层的协议处理，否则丢弃。由于嵌入式设备一般是作为终端，所以为了简化协议，不实现转发功能。

4.IP协议发送处理

IP协议在网络层，提供调用接口给其他协议，因此发送程序首先要判断调用方的协议类型，并做相应初始化。然后需要设定目的MAC地址，当目的主机与源主机在同一子网内，可以使用ARP协议直接匹配或发送ARP请求；但是如果目的主机在另一个子网中时，则将目的MAC地址设置成默认网关的MAC地址，让路由器去转发。最后要做的就是根据图15-13封装IP数据报，再封装成以太帧发送出去。

15.5 运输层

运输层是TCP/IP协议层的核心部分，提供端到端的数据传输服务。如果说网络层提供了主机之间的逻辑通信，那么运输层提供的是运行再不同主机上的进程间的逻辑通信。运输层传输的数据格式称为报文段(segment)。UDP和TCP是该层的主要协议。

15.5.1 UDP协议

1.UDP概述

UDP是一种基本的通信协议，只在发送的报文中增加了端口寻址和可选的差错检测功能。它不是一种握手信息协议，不能确认接收到的数据或交换其他流量控制信息。UDP是一种非连接协议，计算机在使用UDP发送报文之前，不要求远程已联网或指定的目的端口可用于通信。正因为如此，将UDP称为不可靠协议，即如果只使用UDP，则发送方不知道目的主机何时是否接收到报文。

发送主机将UDP数据报置于IP数据报的数据字段中。在以太网中，IP数据报存放在以太网帧的数据字段中。接收到以太网帧后，目的计算机网络栈将UDP数据报的数据部分传递给数据报报头指定的端口或进程。

UDP的大多数功能不如TCP，所以UDP的实现要简单些，更适合于特定的应用场合。但是UDP可将报文发送到多个目的主机，包括向局域网内所有的IP地址以广播方式发

送，或者向指定的IP地址以组播方式发送。对于TCP而言，广播和组播都不现实，因为源主机必须与所有目的主机握手。

2.UDP报头

UDP报头由4个字段组成，后面紧接着是要传输的数据，如图15-17所示。

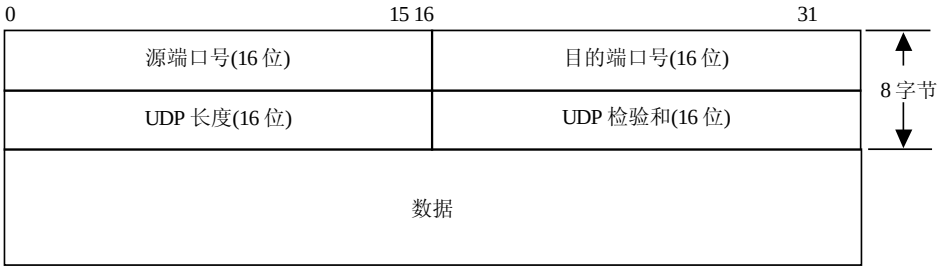

图15-17 UDP首部

①源端口号：源端口号标识发送报文的计算机端口或进程。长度2字节，可选，如果接收进程不需要知道发送数据报的进程，则该字段可置为0。

②目的端口号：目的端口号标识接收报文的目的主机端口或进程。2字节。

③数据报长度：指整个数据报的长度，以字节为单位，包括报头，最大值为65535。该字段为2字节。

④UDP检验和：是根据UDP数据报和伪报头计算得到的差错检测值，可选，该字段伪为2个字节。

伪报头包含源和目的IP地址，以及来自IP数据报报头的协议值。IP数据报在网络中传送时包含UDP数据报。伪报头并不会在网络中传送，校验和中所包含的伪报头内容可以避免目的端错误地接收错误路由的数据报。校验和值的计算方法和IP报头检验和的计算方法类似。

仅在局域网内部传输报文不需要UDP校验和，因为以太网帧的校验和已经提供了差错控制。而对于那些需要通过不同的、也许未知网络传输的报文而言，校验和可以让目的主机能检测到错误数据。

3.UDP数据

一个UDP数据报最大可达到65535个字节，且报头为8字节，因此一个数据报最多可以携带65527个字节的数据。实际上，源计算机常将数据报限制在较短的长度内。使用较短的数据报的一个原因是，过大的数据报可能不适合目的主机的接收缓冲器大小；或者因为接收数据的应用程序可能需要特定长度的报文。较短数据报也许更高效。当大的数据报经过不同能力的网络时，网络协议可能会将数据报拆分为报片，要求目的主机将这些报片重组。所有数据都会到达目的主机，但一般来说，在源主机处拆分数据，然后在目的主机处重组，会比依赖IP进行这项工作更高效。

IP标准要求主机能接收最大长度为576字节的数据报。没有可选项的IP报头长度为20字节，UDP报头为8字节，因此一个最大长度为548字节且没有IP选项的UDP数据报不经过拆分便可到达目的主机。

4.第五个测试实例：UDP数据包的接收和发送

如果IP数据报接收成功，则从UDP报文段中剥离报头。如果使用了UDP校验和，则计算其值，并与接收到的值相比较。使用目的端口号决定将接收到的数据转发到何处。

发送UDP报文段时，在UDP报头的适当位置设置目的端口号和数据报长度。报头中的源端口号和检验和都是可选项。计算校验和需要知道源和目的IP地址。向报头添加待发送的数据。在IP数据报的数据部分放置UDP数据报。IP数据报需要源和目的IP地址，以及根据报头计算的校验和，将IP数据报传送给以太网控制器，以便向网络发送。

UDP协议实现工程列表如图15-18所示。

图15-18 UDP协议实现工程文件

主程序部分如下，增加对UDP的接收和发送。

```
//-------------------------------------------------------------------------*
//工 程 名:UDP                                                             *
//硬件连接:与PC交叉网线直连                                                *
//程序描述:PC使用VB程序通过UDP协议与NE64通信，NE64将接收到的字符串回送给PC*
//目    的:freescale 9s12 NE64系列程序                                     *
//说    明:                                                               *
//注    意:  具体内容见光盘代码                                           *
//-------《嵌入式系统－使用HCS12微控制器的设计与应用》教学实例------------*
 . . . . . . //限于篇幅问题，只给出主要的调用部分，详细代码见光盘.
    //主循环
    while(1)
    {
        //即时更新除MAC地址外的网络参数
        INT16S pData;                   //网络参数FLASH存放地址
        pData = Flash_Data;
        for(i = 0; i < 4; i++)
        {
          localmachine.localip[i] = *((INT8U *)(pData + i));    //本地IP地址
          localmachine.serverip[i] = *((INT8U *)(pData + 4 + i));//服务器IP地址
          localmachine.defgw[i]  = *((INT8U *)(pData + 4 + i)); //网关地址
          localmachine.netmask[i] = *((INT8U *)(pData + 8 + i));//子网掩码
        }
        //gotlink=1连接成功
        if (gotlink)
        {
            //处理接收到的以太网帧
            if( NETWORK_CHECK_IF_RECEIVED() == 1 )
            {
               //received_frame在ethernet.h中定义其结构
                switch( received_frame.protocol)
                {
                case PROTOCOL_ARP:
                    //1.若为ARP请求则作出ARP响应
                    process_arp (&received_frame);//在arp.c中
                    break;
                case PROTOCOL_IP:
                    //2.若为IP信包则作下一步判断处理
                    len = process_ip_in(&received_frame);
                    if(len < 0)  break;
                    switch (received_ip_packet.protocol)
                    {
                    case IP_ICMP:
                        //2.1.若为ICMP信包则返回ICMP报文,完成一次PING命令
                        process_icmp_in (&received_ip_packet, len);
                        break;
                     case IP_UDP://17
                        //2.2.接收到UDP包后保存数据部分到UDP_buf[]数组中
                        process_udp_in (&received_ip_packet,len);
                        //重新封包后发出,用于验证一次UDP包的收发过程
```

```
                        udp_demo_send (&received_ip_packet);
                        break;
                    default:   break;
                    }
                default:  break;
                }//switch
                //3.丢弃剩余帧
                NETWORK_RECEIVE_END();
            }//NETWORK_CHECK_IF_RECEIVED
        }//gotlink
    }//while(1)
}
```

图15-19是运行在主机C（192.168.149.154）上的UDP通信PC程序界面，可以看到目的主机地址和目的MAC地址都是指向设备B，同时也规定了通信双方的端口号，主机C定时向设备B发送字符串“123456”，设备B将接收到的字符串再封装成UDP报文段发送给主机C，主机C从接收的UDP报文中提取出数据放在程序界面的接收文本框中，以供比较。

图15-19 UDP通信PC程序界面

图15-20描述了使用Ethereal捕捉的UDP报文段的情况。这是主机C发送给设备B的报文，在User Datagram Protocol(UDP)中描述了UDP首部的情况。目的端口号和源端口号与图15-15中的设置情况一致。图15-20最下方的一组数据倒数第2行中的37是字符“1”的ASCII码的16进制表示，可以看出从“31”到“38”正是图15-15中发送的数据。

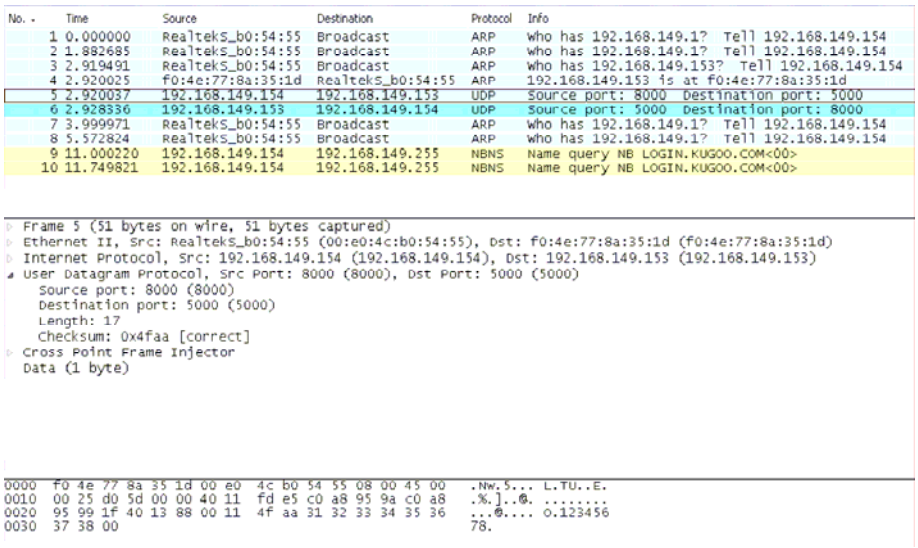

图15-20 Ethereal捕捉的UDP报文段

UDP协议已经可以满足嵌入式以太网设备的通信要求，即可以完成进程间的逻辑通信。用户可以在此基础上完成应用实例。

15.5.2 TCP协议

1.TCP协议概述

TCP称为面向连接的协议，因为进程在交换数据之前必须先彼此建立通信连接。TCP也被称为可靠的协议，因为握手、校验以及序列号和确认号等使源主机可验证数据已正确地到达目的主机。

TCP报文段由报头和可选的数据组成（也可能传输不含数据的报头，用于发送状态或控制信息）。术语"报文段"表示单个TCP报文段只是完整TCP数据传输的一部分。实际上，每次成功的数据传输至少需要2个报文段。源主机发送一个或多个包含数据的报文段，目的主机也发送一个或多个报文段确认已接收到的数据。一次确认可确认多个报文段。与此相反，每个UDP数据报都是独立的单元，不需要额外的通信。与UDP一样，TCP使用端口号表示源和目的主机的进程。

两个进程使用TCP发送和接收数据之前，首先要通过3此握手建立进程所在的计算机之间的连接。在完成握手时，每台计算机都要确认握手中指定的端口可用于接收来自另一台计算机指定端口的通信。然后，双方都可以使用连接向对方计算机发送TCP报文段。

在通过已建立的连接接收到数据时，目的主机响应，返回数据是否已正确到达、是否可以发送更多的数据信息。如果可以，还要返回目的主机能接收新数据的数量。

要关闭连接，双方都需要发送关闭连接的请求，并且等待对方对请求的确认。

2.TCP协议简化

(1)简化的TCP状态机

标准的TCP状态机很复杂，对于资源有限的16位MCU来说，要维护这样的状态机是非常困难的。然而，嵌入式系统的设计都是针对某个特定的应用进行的，其目的在于实现一个无冗余的尽可能简洁的系统。本书要实现的是一个基于嵌入式Web服务器的测控系统，经过仔细分析，本书得到如图15-21所示的简化的TCP状态机。其中连接的断开由服务器主动执行，通过多次实验总结出来该方式在本书系统中，比标准的TCP协议主动断开连接的状态机简单且稳定。

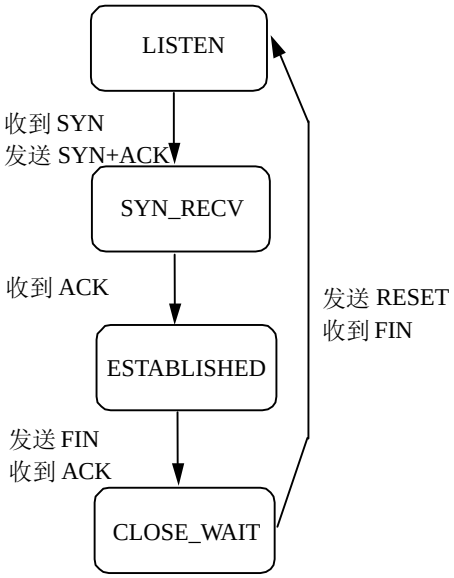

图15-21 服务端简化的TCP状态图

(2)减少同时支持的连接数目

一个TCP连接由源IP地址、源端口号、目的IP地址、目的端口号唯一确定。因此每次服务器在LISTEN状态时，当接收到一个TCP的连接请求，发送SYN+ACK后，要记录请求端的IP地址、使用的端口号等相关信息，以便对后面到达的信息进行检查，判断是否属于某个已建立的连接。如果支持多个连接，则要相应地记录各个与连接的相关信息，在每个数据报到达时都要判断该数据报所属连接，并在不同连接之间进行切换。

本书可以根据不同的应用要求调整TCP所支持的连接数量，但是通常在同一时刻仅支持单个TCP连接。同时为了避免因为数据报的丢失而造成状态机的死锁，本书使用简单定时机制，使TCP状态机在超时后复位。

(3)简化的确认机制

从TCP协议头部中的序列号和确认号字段可以完成TCP协议中的确认工作。由于标准的TCP协议使用慢启动的滑动窗口机制，对于使用了滑动窗口的TCP连接，其确认实

质上是一种批量确认，资源相对紧张的16位处理器要对多个数据报连续传输进行维护和处理，困难很大。考察滑动窗口协议可以发现，滑动窗口的一个极限情况，是只使用单个窗口，这样就变成一种简单确认的处理方法。使用该项方法后，所有的处理只是对单个数据报的发送和确认，节约了系统的资源，也使维护更加的方便。

出于兼容性考虑，需要通信的另一方也应该使用简单确认方法。因为如果使用较大的窗口，就可能造成处理器被淹没。这可以通过设置待发送数据报的TCP头部的Windows字段的大小来解决。Windows字段告诉对方本地的接收缓冲区的大小。通过合理设置本地待发送数据报的Windows字段，可以使参与通信的另外一方遵守简单确认方法。

3.TCP报头

TCP报文段的报头有10个必需的字段和1个可选字段。报头至少为20字节。报头后面的数据是可选项。如图15-22所示。

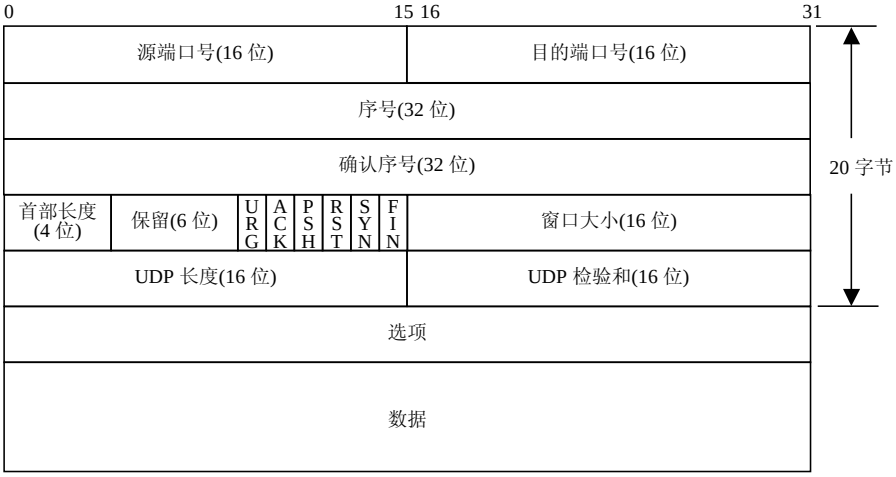

图15-22 TCP首部

源端口号、目的端口号标识发送报文的计算机端口或进程。一个TCP报文段必须包括源端口号，使目的主机知道应该向何处发送确认报文。

序号用于标识每个报文段，使目的主机可确认已收到指定报文段中的数据。当源主机用于多个报文段发送一个报文时，即使这些报文到达目的主机的顺序不一样，序列号也可以使目的主机按顺序排列它们。

在建立连接时发送的第一个报文段中，双方都提供一个初始序列号。TCP标准推荐使用以4ms间隔递增1的计数器值作为这个初始序列号的值。使用计数器可以防止连接关闭再重新连接时出现相同的序列号。

对于那些包含数据的报文段，报文段中第一个数据字节的数量就是初始序列号，其后数据字节按顺序编号。如果源主机使用同样的连接发送另一个报文段，那么这个报文段的序列号等于前一个报文段的序列号与前一个报文段中数据字节的数量之和。例如，假设源主机发送3个报文段，每个报文段有100字节的数据，且第一个报文段的

序列号是1000，那么第二个报文段的序列号就是1100（1000＋100），第三个报文段的序列号就是1200（1100＋100）。如果序列号增大至最大值将复位为0。

目的主机返回确认号，使源主机知道某个或几个报文段已被接收。如果ACK控制位被设置为1，则该字段有效。确认号等于顺序接收到的最后一个报文段的序号加1，这也是目的主机希望下次接收的报文段的序号值。返回确认号后，计算机认为已接收到小于该确认号的所有数据。

例如，序列号等于前一个报文段的序列号与前一个报文段中数据字节的数量之和。例如，假设源主机发送3个报文段，每个报文段有100字节的数据，且第一个报文段的序列号是1000，那么接收到第一个报文段后，目的主机返回含确认号1100的报头。接收到第二个报文段（其序号为1100）后，目的主机返回确认 号1200。接收到第三个报文段后，目的主机返回确认号1300。

目的主机不一定在每次接收到报文段后都返回确认号。在上面的例子中，目的主机可能等到所有3个报文段都收到后，再返回一个含确认号1300的报文段，表示已接收到全部1200字节的数据。但是如果目的主机再发回确认号之前等待时间过长，源主机会认为数据没有到达目的主机，并自动重发。

上面的例子中，如果目的主机接收到了报文段号为1000的第一个报文段以及报文段号为1200的最后一个报文段，则可返回确认号1100，但是再返回确认号1300之前，应该等待报文段号为1100的中间报文段。

由于TCP报头的长度随TCP选项字段内容的不同而变化，因此报头中包含一个指定报头字段的字段。该字段以32比特为单位，所以报头长度一定是32比特的整数倍，有时需要在报头末尾补0。如果报头没有TCP选项字段，则报头长度值为5，表示报头一个有160比特，即20字节。

保留位全部为0。

控制位（6位）

URG:报文段紧急。

ACK：确认号有效。

PSH：建议计算机立即将数据交给应用程序。

RST：复位连接。

SYN：进程同步。在握手完成后SYN为1，表示TCP建立已连接。此后的所有报文段中，SYN都被置0。

FIN：源主机不再有待发送的数据。如果源主机数据发送完毕，将把该连接下要发送的最后一个报文段的报头中的FIN位置1，或将该报文段后面发送的报头中该位置1。

接收计算机可接收的新数据字节的数量，根据接收缓冲区可用资源的大小，其值随计算机所发送的每个报文段而变化。源主机可以利用接收到的窗口值决定下一个报文段的大小。

源主机和目的主机根据TCP报文段以及伪报头的内容计算校验和。在伪报头中存放着来自IP报头以及TCP报文段长度信息。与UDP一样，伪报头并不在网络中传输，并且在校验和中包含伪报头的目的是为了防止目的主机错误地接收存在路由的错误数据

报。

如果URG为1，则紧急指针标志着紧急数据的结束。其值是紧急数据最后1字节的序号，表示报文段序号的偏移量。例如，如果报文段的序号是1000，前8个字节都是紧急数据，那么紧急指针就是8。紧急指针一般用途是使用户可中止进程。

完整的TCP报头必须是32比特的整数倍，为了达到这一要求，常在TCP选项字段的末尾补零。

报头后面是可选的报文段数据部分。IP协议标准要求能接收最长达576字节的数据报。无其他选项的IP报头是20字节，无其他选项的TCP报头也是20字节，所以IP选项和TCP选项且含有多达536字节数据的TCP报文段无须分片就可达到目的主机。

4.TCP协议连接的建立、数据传输及关闭

TCP协议连接建立的过程被称为“三次握手”。首先，客户端向服务端提出连接请求。此时客户端在TCP报头中插入自己的ISN，并置SYN标志为1，表示序列号字段合法，需要检查。其次，服务端收到该TCP分段后，以自己的ISN回应，同时确认收到客户端的TCP分段，置ACK标志为1。最后，客户端确认收到服务端的ISN，置ACK标志为1。至此完整的TCP连接建立，开始全双工模式的数据传输过程。

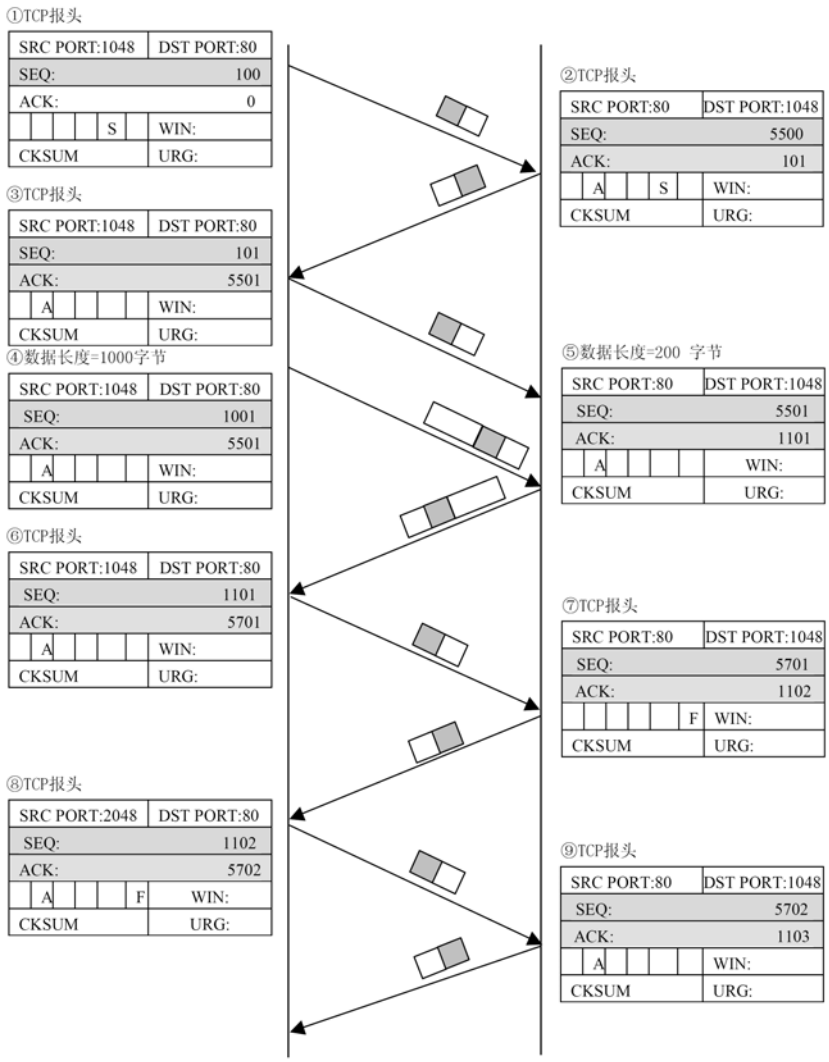

图15-23 TCP协议建立连接、传送数据和关闭过程图.

图15-23给出了本书应用的TCP连接建立和撤销过程。Web浏览器向Web服务器发送的报文初始序列号为100，服务器向浏览器发送的报文初始序列号为5500。在连接建立之后，则开始应用程序数据的发送与接收。在传输数据的时候，传输了TCP协议的各种不同的段。在④中，浏览器从序列号101发送了1000字节的数据。服务器在收到如④所示的一个段后，返回一个确认应答段，并在段中发送200 字节的数据。此时，确认应答号是所接收到的段的序列号加上所接收的数据长度，即如⑤所示的应答号为1101。浏览器在接收到如⑤所示的段后，将确认号作为序列号，并将序列号加上数据长度作为确认号，发送如⑥所示的段。重复该过程至数据发送完毕。

在不需要发送和接收数据时，则切断已经建立的连接。连接的切断需要传输如⑦⑧⑨所示的段。考虑到节省资源和安全性问题，本系统的服务器在发送完数据后立即

提出切断连接的请求。

5.TCP协议的发送和接收

TCP报文的接收较上述协议的接收复杂：首先需要判断数据包的protocol值是否为6，校验和是否正确，报头中是否存在选项字段值，如果存在，是否为MSS值，其具体处理流程如图15-24所示。其中TCP协议接收的关键是状态机的实现，其处理流程如图15-25所示。而TCP协议报文的发送与UDP协议类似，只需按照报文格式封装数据，再调用IP协议发送函数就可以将数据发送至网络上。

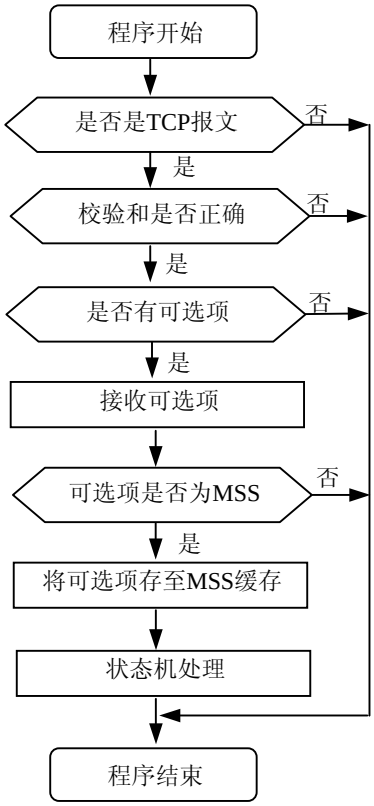

图15-24 TCP接收处理流程图

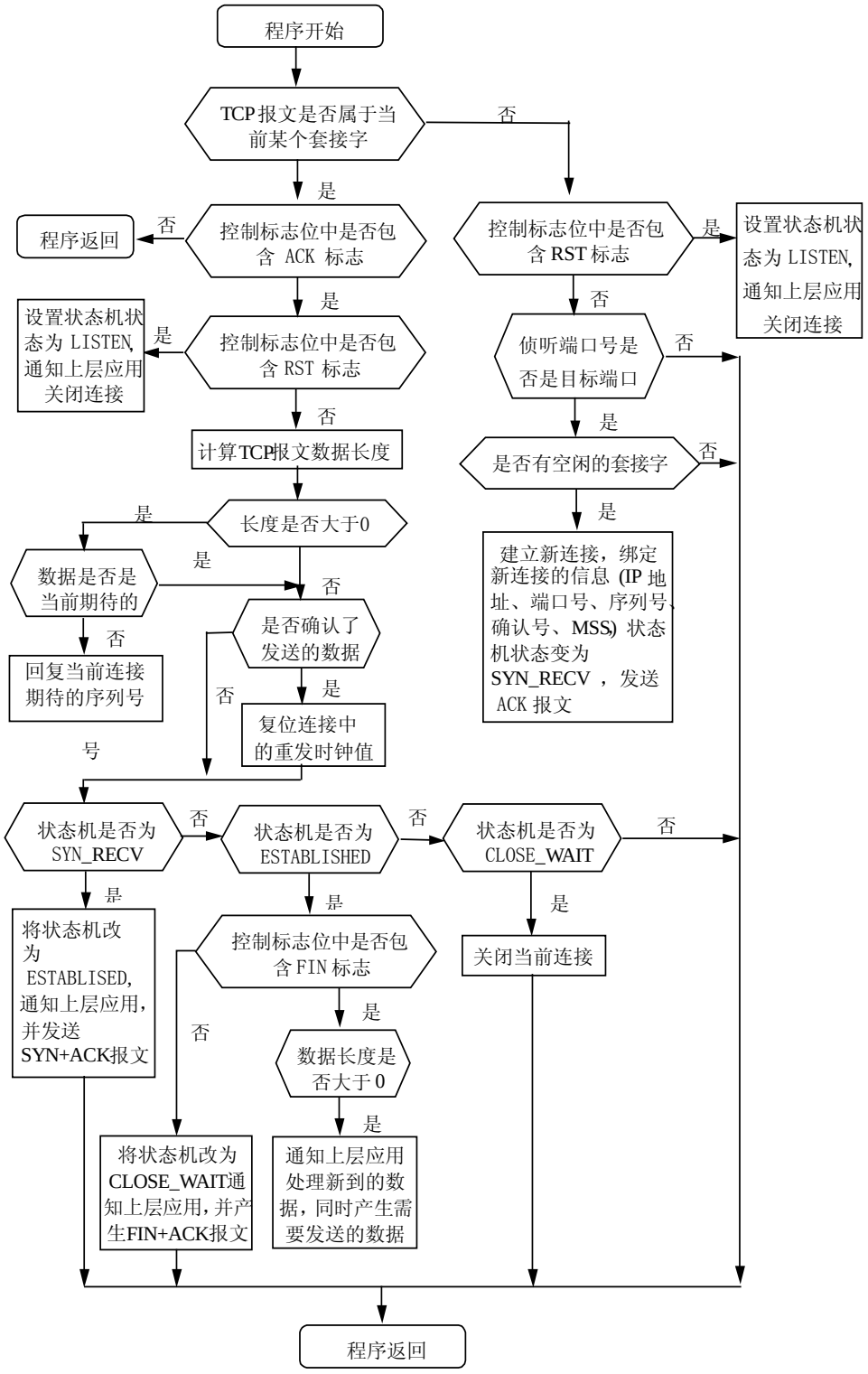

图15-25 TCP协议接收的状态处理流程图

15.6 应用层

网络应用是计算机网络存在的理由。应用层位于运输层之上，包含的高层协议包括超文本传输协议HTTP（Hyper Text Transport Protocl）、文件传输协议FTP(File Transport Protocol)和简单邮件传输协议SMTP(Simple Mail Transport Protocol)等。

15.6.1 Web应用

Web的全称是world wide web，我们常听说的WWW其实就是这个名称的缩写，中文中叫它万维网。Web非常流行的一个很重要的原因就在于它可以在一页上同时显示色彩丰富的图形和文本的性能。在Web之前Internet上的信息只有文本形式。Web可以提供将图形、音频、视频信息集合于一体的特性。同时，Web是非常易于导航的，只需要从一个连接跳到另一个连接，就可以在各页各站点之间进行浏览了。

无论你的系统平台是什么，你都可以通过Internet访问WWW。浏览WWW对你的系统平台没有什么限制。无论从Windows平台、UNIX平台、Macintosh还是别的什么平台我们都可以访问WWW。对WWW的访问是通过一种叫做浏览器（browser）的软件实现的。如 Netscape 的Navigator、NCSA的Mosaic、Microsoft的Explorer等。

Web应用程序采用客户机/服务器模式。总是打开的Web服务器服务于运行在客户机上的浏览器的请求，并向该客户机发送所请求的对象作为回应。客户机之间是不进行通信的，服务器需要有一个固定的IP地址。

15.6.2 HTTP协议

Web的应用层协议是HTTP协议，它是Web的核心。HTTP协议有两部分程序实现：一个是客户机程序，一个是服务器程序，它们运行于不同的端系统中，通过交换HTTP报文进行会话。

1. HTTP消息格式

HTTP消息分为请求消息和响应消息两类。要实现Web服务器，必须清楚这两部分的构成。请求消息和响应消息的格式分别如表15-6和15-7所示。

表 15-6 HTTP 请求消息格式

请求方法	空格符	URL	空格符	版本号	回车符	换行符
头部字段名:值						
……						
头部字段名:值						
空行						
实体						

在表15-6中，请求方法字段有若干个值可以选择，最常用的有GET、POST和HEAD。

HTTP请求消息绝大多数使用GET方法，这也是浏览器请求对象的方法。但是实体数据在GET方法中未使用，只在POST方法中使用。POST方法和GET方法都适用于需要用户填写表单的场合，只是GET方法是将数据附在URL后作为URL的一部分传递的，受URL的最大长度为1024字节的限制，而POST方法的数据量很大。经过仔细分析，GET方法足以满足本书表单提交的要求，因此本书只实现了GET方法，从而简化程序，提高系统工作效率。版本号有HTTP1.1和HTTP1.0，由于请求消息是浏览器发出的，可以通过浏览器的选项设置版本号。头部字段名部分为可选的，它表示请求主机、是否使用持久连接等信息。本书中这些信息都由浏览器生成，不牵涉到嵌入式Web服务器的设计，所以不作详细介绍。

表 15-7 HTTP 响应消息格式

版本号	空格符	状态码	空格符	原因短语	回车符	换行符
头部字段名:值						
……						
头部字段名:值						
空行						
实体						

要实现Web服务器必须实现响应消息，结合本书讲述各个字段的含义和具体用法。在表15-7中，版本号含义与请求消息的含义相同。但是，考虑到系统实现时嵌入式TCP协议同时支持的连接次数和安全性问题，本书采用HTTP1.0协议，Web服务器每次发送完响应就断开连接。状态码的含义很多，本书使用了两种：当请求网页成功时，返回状态码200，原因短语为OK；当所请求的网页不存在时，返回状态码404，原因短语为NOT FOUND。头部字段名也是可选部分，但是本书使用了其中一个选项Content-Length:指出所发送对象的字节数，以方便程序调试。实体部分就是响应的具体内容，譬如一个HTML网页或者一张图片等等。

2.第六个测试实例：HTTP协议静态页面的实现

本书HTTP协议静态页面的实现需要完成如下内容：首先获取URL中的文件名，接着根据该文件名调用https_calculatehash()函数获取文件句柄，即文件处理入口数据结构中的hash域值，根据该值查找文件的起始地址，然后将文件装入TCP套接字发送缓冲区。当所发送的文件过长而大于发送缓冲区的大小时则会发生缓冲区的溢出问题，本书的解决办法是：首先判断文件的长度，当文件过长时，将文件分割成多个不大于发送缓冲区大小的分段，然后循环发送出去。

HTTP协议实现工程列表如图15-26所示。

图15-26 HTTP协议实现工程文件

主函数如下所示，增加HTTP协议和TCP协议处理部分。

```
//-------------------------------------------------------------------------*
//工 程 名:HTTP.IDESln                                                      *
//硬件连接:与PC交叉网线直连,并接上串口SCI1                                  *
//程序描述:用浏览器访问http://192.168.149.137/index.htm                    *
```

```
//目    的:freescale 9s12 NE64系列程序                                    *
//说    明:                                                              *
//注    意:  具体内容见光盘代码                                          *
//-------《嵌入式系统－使用HCS12微控制器的设计与应用》教学实例------------*
    . . . . . . //限于篇幅问题，只给出主要的调用部分，详细代码见光盘
    //主循环
    while(1)
    {
        //即时更新除MAC地址外的网络参数
        INT16S pData;                  //网络参数FLASH存放地址
        pData = Flash_Data;
        for(i = 0; i < 4; i++)
        {
          localmachine.localip[i] = *((INT8U *)(pData + i));      //本地IP地址
          localmachine.serverip[i] = *((INT8U *)(pData + 4 + i));//服务器IP地址
          localmachine.defgw[i]  = *((INT8U *)(pData + 4 + i));   //网关地址
          localmachine.netmask[i] = *((INT8U *)(pData + 8 + i));  //子网掩码
        }
        //gotlink=1连接成功
        if (gotlink)
        {
          //处理接收到的以太网帧
          if( NETWORK_CHECK_IF_RECEIVED() == 1 )
          {
            switch( received_frame.protocol)//received_frame在ethernet.h中定义
            {
                case PROTOCOL_ARP:
                    //1.若为ARP请求则作出ARP响应
                    process_arp (&received_frame);//在arp.c中
                    break;
                case PROTOCOL_IP:
                    //2.若为IP信包则作下一步判断处理
                    len = process_ip_in(&received_frame);
                    if(len < 0)
                        break;
                    switch (received_ip_packet.protocol)
                    {
                    case IP_ICMP:
                        //若为ICMP信包则返回ICMP报文,完成一次PING命令
                        process_icmp_in (&received_ip_packet, len);
                        break;
                    case IP_TCP://6
                        //HTTP命令
                        process_tcp_in (&received_ip_packet, len);
                        break;
                    default:
                        break;
                    }
                default:
```

```
                    break;
                }//switch
                //3.丢弃剩余帧
                NETWORK_RECEIVE_END();
            }//NETWORK_CHECK_IF_RECEIVED
        }//gotlink
    }//while(1)
}
```

HTTP协议中静态页面处理的程序流程如图15-27所示。当主机A（192.168.149.153）通过IE浏览器向已经配置成WEB服务器的设备B（192.168.149.137）发送HTTP请求后，设备B将存在FLASH区的页面数据封装成HTTP报文回送给主机A。主机A上的页面显示如图15-28所示。该页面是采用HTML语言编写后转换成16进制的文件存储在设备B的FLASH中，由于NE64的FLASH空间有限，尽量不要使用图片等大容量的对象。

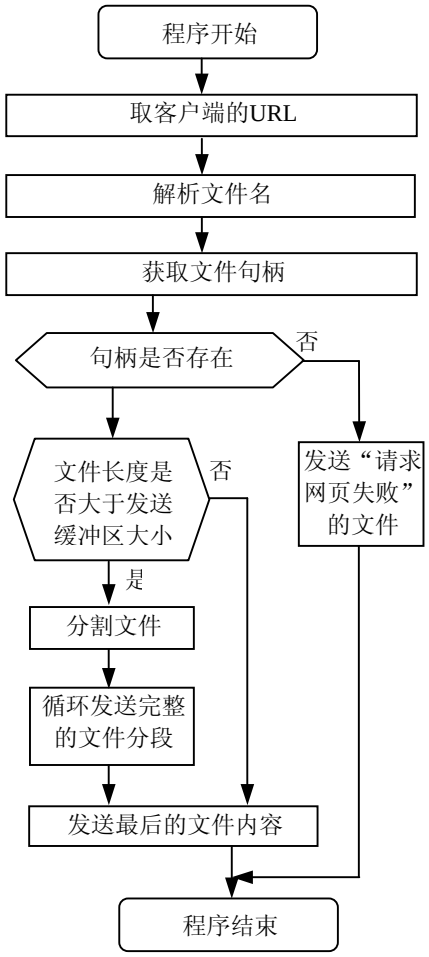

图15-27 HTTP静态页面处理流程图

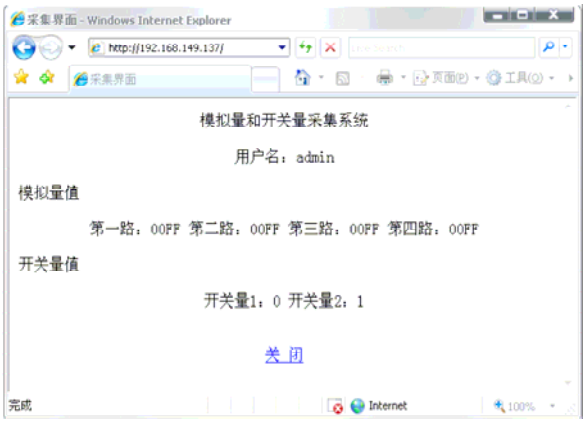

图15-28 主机A上浏览器打开的HTTP静态页面

练习题

【基础题】

1.简述嵌入式以太网的含义。

2.TCP/IP协议模型共分几层，每一层的功能是什么？

2.简述IP地址、网段和子网掩码的关系。

3.ARP协议的功能是什么？

4.UDP和TCP协议有那些异同点？各适用于什么样的应用？

5.简述WEB服务的概念。

【综合题】

1.以太网子网内主机之间互相通信只少要知道对方的什么地址？怎样获得？

2.不同子网的主机之间可以PING通吗？为什么？

3.画图描述TCP通信过程。