11 基于 W5500 的以太网通信实验

11.1. 背景介绍

本章将介绍 AC208 开发板上的网口及其使用。在本章实验中将使用硬件 SPI 和 IO 模拟 SPI 两种方式驱动开发板上网口实现 TCP 服务器、TCP 客户端和 UDP 功能。

11.1.1. W5500 简介

W5500 是一款全硬件 TCP/IP 嵌入式以太网控制器，为嵌入式系统提供了更加简易的互联网连接方案。W5500 集成了 TCP/IP 协议栈，10/100M 以太网数据链路层（MAC）及物理层（PHY），使得用户使用单芯片就能够在他们的应用中拓展网络连接。

W5500 提供了 SPI（外设串行接口）从而能够更加容易与外设 MCU 整合。而且，W5500 的使用了新的高效 SPI 协议支持 80MHz 速率，从而能够更好的实现高速网络通讯。为了减少系统能耗，W5500 提供了网络唤醒模式（WOL）及掉电模式供客户选择使用。W5500 的特点如下所示：

(1) 支持硬件 TCP/IP 协议：TCP、UDP、ICMP、IPv4、ARP、IGMP、PPPoE；
(2) 支持 8 个独立端口（Socket）同时通讯；
(3) 支持掉电模式；
(4) 支持网络唤醒；
(5) 支持高速串行外设接口（SPI 模式 0、3）；
(6) 内部 32K 字节收发缓存；
(7) 内嵌 10BaseT/100BaseTX 以太网物理层（PHY）；
(8) 支持自动协商（10/100-Based 全双工/半双工）；
(9) 不支持 IP 分片；
(10) 3.3V 工作电压，I/O 信号口 5V 耐压；
(11) LED 状态显示（全双工/半双工，网络连接，网络速度，活动状态）；
(12) LQFP48 无铅封装（7x7mm，间距 0.5mm）。

W5500 的原理框图如下图 11. 1 所示。

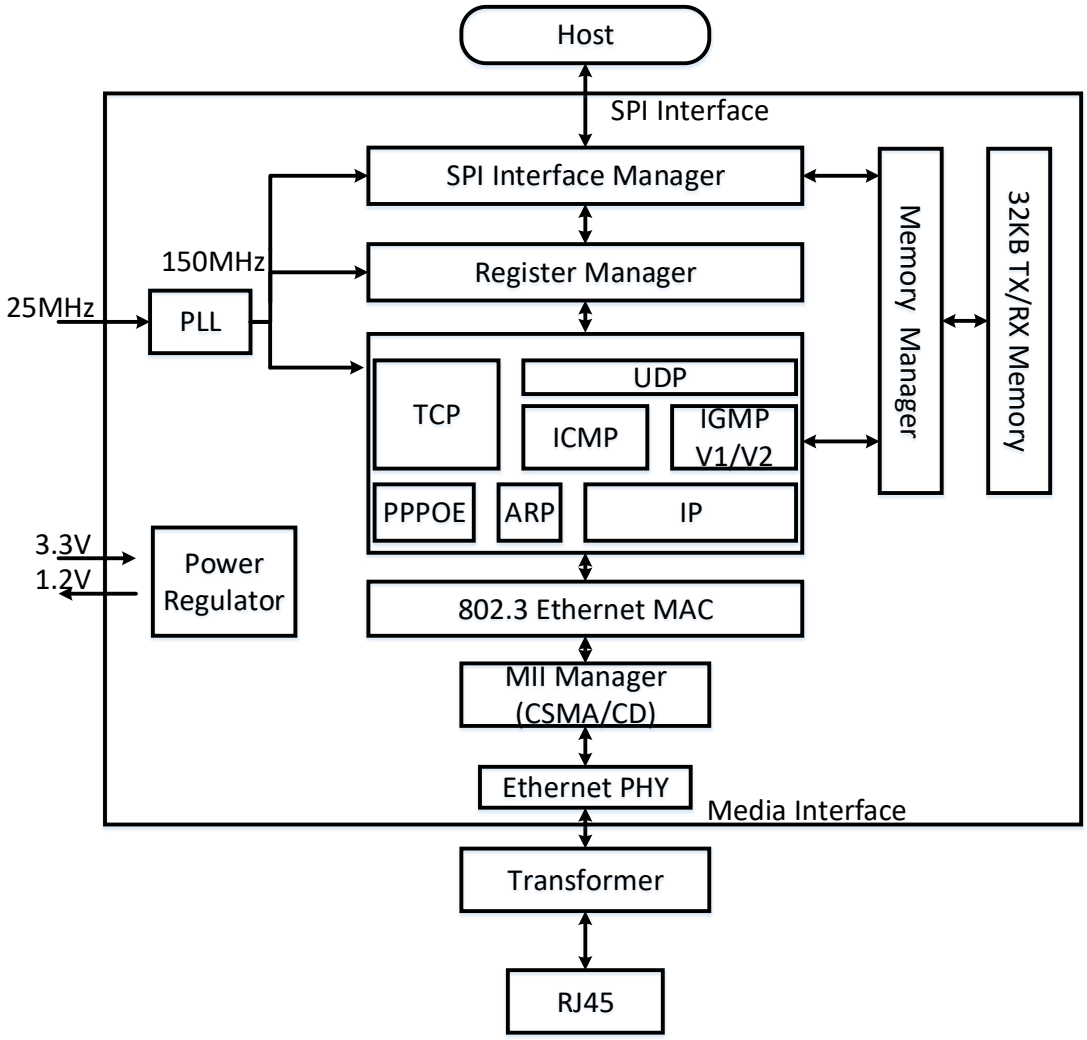


图 11. 1 W5500 原理框图

从上图可以看出，在芯片外围有时钟接口、电源接口、SPI 通信接口以及以太网数据收发接口。芯片内部主要分为 SPI 接口管理，芯片寄存器管理，收发缓存管理，传输层的各种通信协议，网络层的 IP 协议，链路层的 MAC，连接链路层和物理层的 MII 接口，以及最后的物理层。

11.1.2. W5500 主机接口介绍

W5500 提供了 SPI 作为外设主机接口，有 SCSn、SCLK、MOSI、MISO 共四路信号，且作为 SPI 从机工作。

W5500 与 MCU 的连接方式如下图 11. 2 和图 11. 3 所示。在可变数据模式中，W5500 可以与其他 SPI 设备共用 SPI 接口，但是一旦将 SPI 指定给 W5500 之后，则不能再与其它 SPI 设备共用；在固定数据长度模式中，其中的 SCSn 信号直接接地，将 SPI 指定给 W5500 之后，不能与其他 SPI 设备共享。

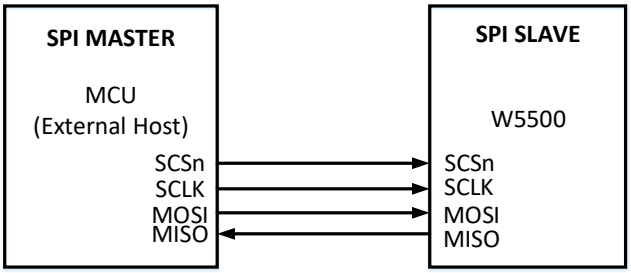


图 11. 2 VDM 可变数据长度模式（SCSn 受主机控制）

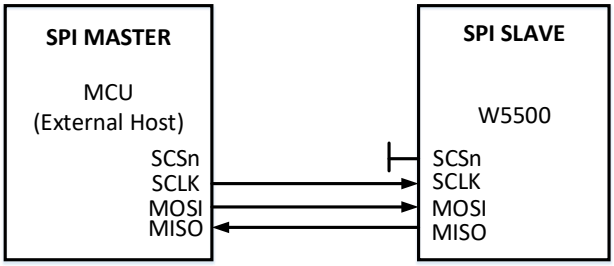


图 11. 3 FDM 固定数据长度模式（SCSn 保持接地）

SPI 一共有四种工作模式，每种模式的区别是根据 SCLK 的极性及相位不同定义的。W5500 支持模式 0 和模式 3，这两种模式下数据都是在 SCLK 的上升沿锁存，下降沿输出。MOSI 和 MISO 信号无论是接收或者发送，都遵从最高标志位（MSB）到最低标志位（LSB）的传输序列。

11.1.2.1. SPI 数据帧

W5500 的 SPI 数据帧包括了 16 位地址段的偏移地址，8 位控制段和 N 字节数据段，如下图 11. 4 所示。

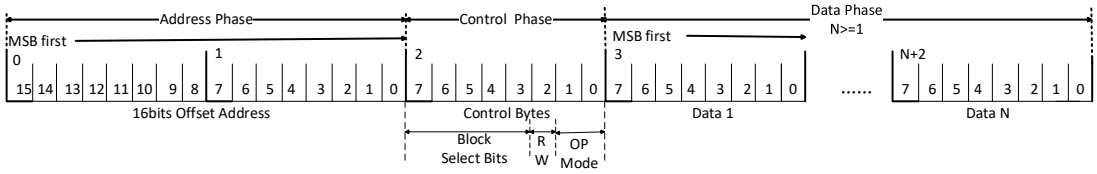


图 11. 4 SPI 数据帧格式

上图中的 8 位控制段可以通过修改区域选择位（BSB[4:0]），读/写访问模式位（RWB）以及 SPI 工作模式位（OM[1:0]）来重新定义。W5500 支持数据的连续读/写。其流程为数据从（2/4/N 字节连续数据的）偏移地址的基址开始传输，偏移地址会（自增寻址）加 1 传输接下来的数据。

11.1.2.2. 可变数据长度模式（VDM）

VDM 模式下，SPI 的数据帧的长度被外设主机控制的 SCSn 所定义。这就意味着数据段长度根据 SCSn 的控制，可以是一个随机值，在该模式下，M[1:0]位

必须是“00”。下图 11.5 所示为 VDM 模式下的写访问时序图，此时 SPI 数据帧的控制段：读写控制位（RWB）为“1”，工作模式位为“00”；图 11.6 所示为 VDM 模式下读访问时序图，此时 SPI 的数据帧的控制段：读写控制位（RWB）为“0”，工作模式位为“00”。

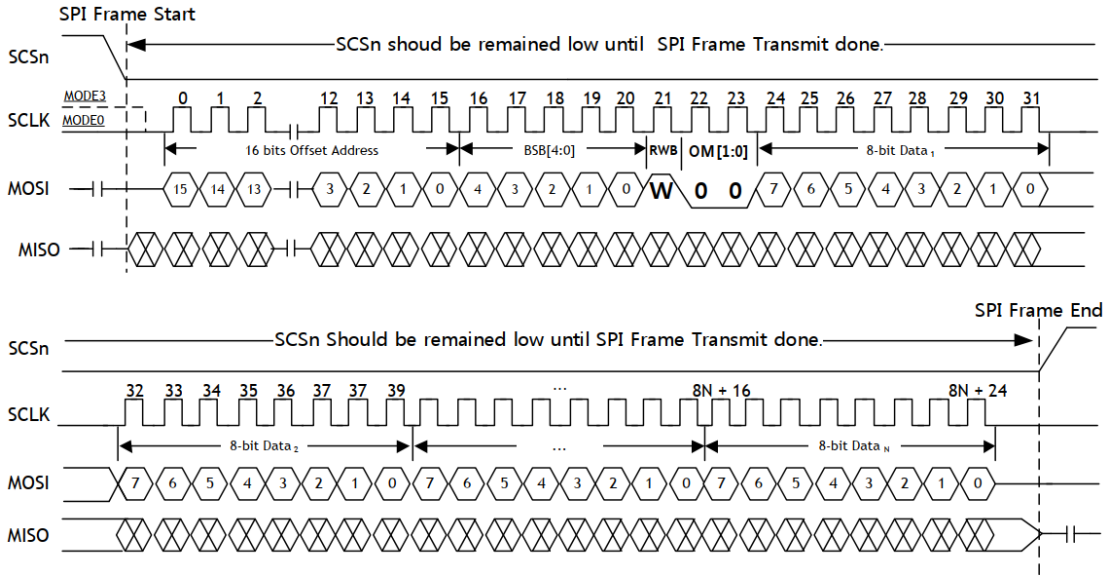


图 11.5 VDM 模式下写访问

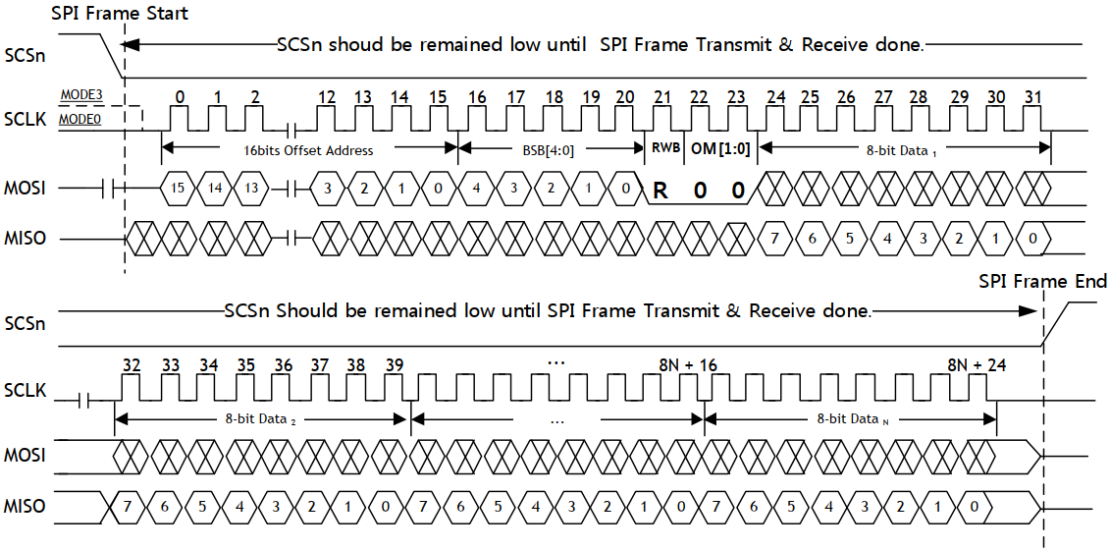


图 11.6 VDM 模式下读访问

11.1.2.3. 固定数据长度模式（FMD）

在 FMD 模式中，SCSn 必须连接到低电平（保持接地）。与此同时，SPI 接口不能与其他 SPI 设备共享。数据长度由 SPI 工作模式位的控制段的值控制（OM[1:0]），‘01’代表 1 字节、‘10’代表 2 字节、‘11’代表四字节，FMD 模式更详细的内容参考 W5500 的官方数据手册。

11.1.3. 寄存器及内存简介

W5500 有 1 个通用寄存器，8 个 Socket 寄存器区，以及对应每个 Socket 的收发缓存区。每个区域均通过 SPI 数据帧的区域选择位（BSB[4:0]）来选取。图 11. 7 显示了区域选择位（BSB[4:0]）选择的区域以及收/发缓存区的可用偏移地址范围。每一个 Socket 的发送缓存区都在一个 16KB 的物理发送内存中，初始化分配为 2KB。每一个 Socket 的接收缓存区都在一个 16KB 的物理接收内存中，初始化分配为 2KB。无论给每个 Socket 分配多大的收/发缓存，都必须在 16 位的偏移地址范围内（从 0x0000 到 0xFFFF）。

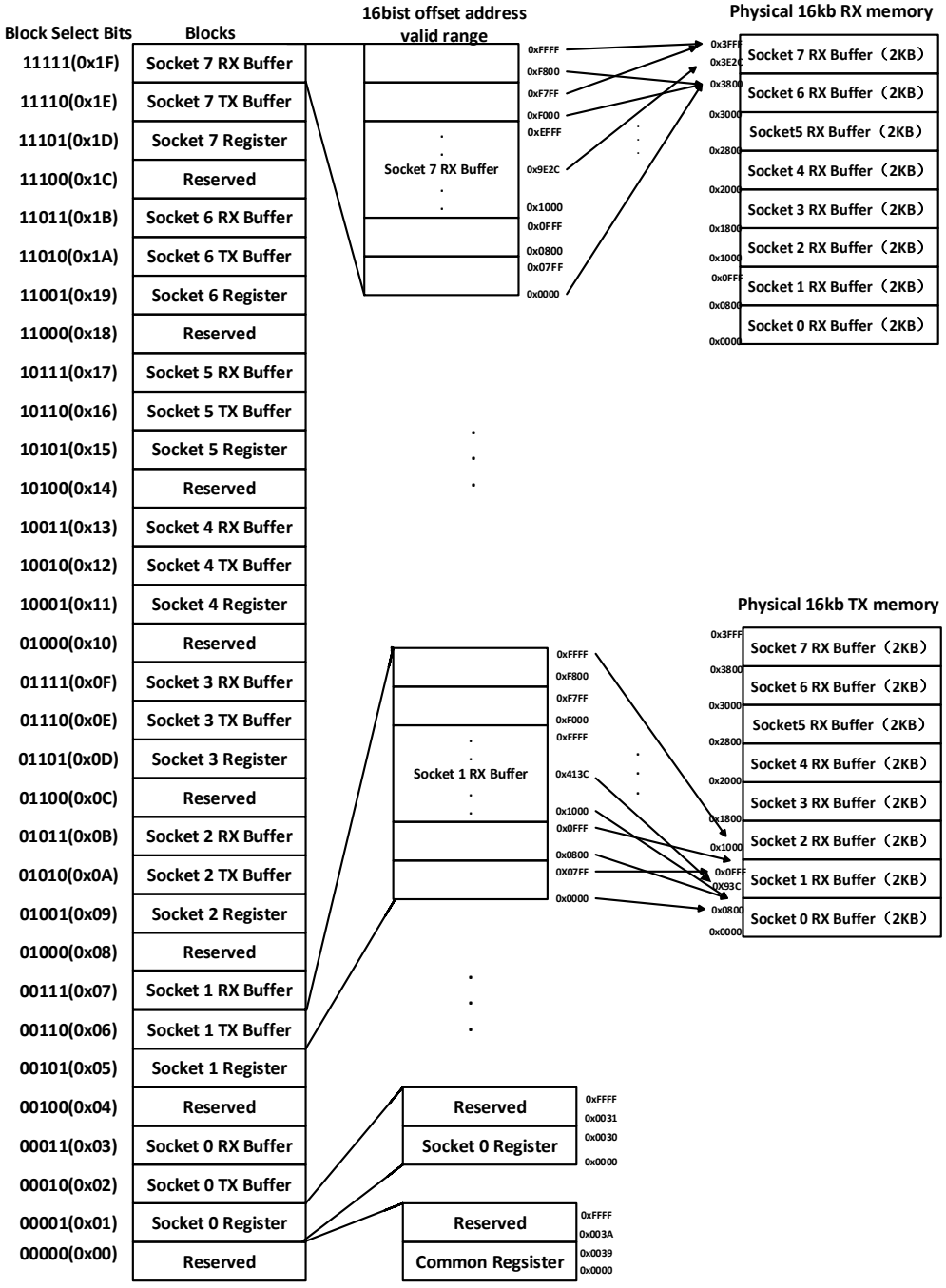


图 11. 7 寄存器及内存构成图

11.1.3.1. 通用寄存器区

通用寄存器区配置了 W5500 的基本信息，例如：IP 及 MAC 地址。该区域可以通过 SPI 数据帧的区域选择位（BSB[4:0]）的值选定。下表 11- 1 描述了通用寄存器的偏移地址，对于每个寄存器的详细信息请参考 W5500 的器件手册，这里就不进行讲述了。

表 11- 1 通用寄存器的偏移地址

Offset	Register	Offset	Register	Offset	Register
0x0000	Mode (MR)	0x0013 0x0014	Interrupt Low Level Timer (INTLEVEL0) (INTLEVEL1)	0x0024 0x0025	PPP Session Identification (PSID0) (PSID1)
0x0001 0x0002 0x0003 0x0004	Gateway Address (GAR 0) (GAR 1) (GAR 2) (GAR 3)	0x0015	Interrupt (IR)	0x0026 0x0027	PPP Maximum Segment Size (PMRU0) (PMRU1)
		0x0016	Interrupt Mask (IMR)		
0x0005 0x0006 0x0007 0x0008	Subnet Mask Address (SUBR0) (SUBR1) (SUBR2) (SUBR3)	0x0017	Socket Interrupt (SIR)	0x0028 0x0029 0x002A 0x002B	Unreachable IP Address (UIPR0) (UIPR1) (UIPR2) (UIPR3)
		0x0018	Socket Interrupt Mask (SIMR)		
0x0009 0x000A 0x000B 0x000C 0x000D 0x000E	Source Hardware Address (SHAR0) (SHAR1) (SHAR2) (SHAR3) (SHAR4) (SHAR5)	0x0019 0x001A	Retry Time (RTR0) (RTR1)	0x002C 0x002D	Unreachable Port (UPORTR0) (UPORTR1)
		0x001B	Retry Count (RCR)	0x002E	PHY Configuration (PHYCFGR)
		0x001C	PPP LCP Request Timer (PTIMER)	0x002F — 0x0038	Reserved
		0x001D	PPP LCP Magic number (PMAGIC)		
0x000F 0x0010 0x0011 0x0012	Source IP Address (SIPR0) (SIPR1) (SIPR2) (SIPR3)	0x001E 0x001F 0x0020 0x0021 0x0020 0x0023	PPP Destination MAC Address (PHAR0) (PHAR1) (PHAR2) (PHAR3) (PHAR4) (PHAR5)	0x0039	Chip version (VERSIONR)
				0x003A~ 0xFFFF	Reserved

11.1.3.2. Socket 寄存器区

W5500 支持 8 个 Socket 作为通讯信道。每个 Socket 通过 Socket n 寄存器区控制。Socket n 寄存器可以通过 SPI 数据帧中的区域选择寄存器(BSB[4:0])来选定对应的寄存器 n。下表 11- 2 定义了 Socket n 寄存器区对应的 16 位偏移地址。

表 11- 2 Socket n 寄存器区对应的 16 位偏移地址

Offset	Register	Offset	Register	Offset	Register
0x0000	Socket n Mode (Sn_MR)	0x0010 0x0011	Socket n Destination Port (Sn_DPORT0) (Sn_DPORT1)	0x0024 0x0025	Socket n Tx Write Pointer (Sn_TX_WR0) (Sn_TX_WR1)
0x0001	Socket n Command (Sn_CR)	0x0012 0x0013	Socket n Maximum Segment Size (Sn_DPORT0) (Sn_DPORT1)	0x0026 0x0027	Socket n Rx Received Size (Sn_TX_PSR0) (Sn_TX_PSR1)

0x0002	Socket n Interrupt (Sn_IR)	0x0014	Reserved	0x0028 0x0029	Socket n Rx Read Pointer (Sn_TX_PSR0) (Sn_TX_PSR1)
0x0003	Socket n Status (Sn_SR)	0x0015	Socket n IP TOS (Sn_TOS)	0x002A 0x002B	Socket n Rx Read Pointer (Sn_TX_PSR0) (Sn_TX_PSR1)
0x0004 0x0005	Socket n Source Port (Sn_PORT0) (Sn_PORT1)	0x0016	Socket n IP TTL (Sn_TTL)	0x002C	Socket n Interrupt Mask (Sn_IMR)
0x0006 0x0007 0x0008 0x0009 0x000A 0x000B	Socket n Destination Hardware Address (Sn_DHAR0) (Sn_ DHAR1) (Sn_DHAR2) (Sn_DHAR3) (Sn_ DHAR4) (Sn_ DHAR5)	0x0017 — 0x001D	Reserved	0x002D 0x002E	Socket n Fragment Offset in IP header (Sn_TX_PSR0) (Sn_TX_PSR1)
		0x001E	Socket n Receive Buffer Size (Sn_RXBUF_Size)	0x002F 0x0030 — 0xFFFF	Keep alive timer (Sn_KPALVTR) Reserved
		0x001F	Socket n Transmit Buffer Size (Sn_TXBUF_Size)		
		0x0020 0x0021	Socket n TX Free Size (Sn_TX_FSR0) (Sn_TX_FSR1)		
0x000C 0x000D 0x000E 0x000F	Socket n Destination IP Address (Sn_DIPR0) (Sn_DIPR1) (Sn_DIPR2) (Sn_DIPR3)	0x0022 0x0023	Socket n TX Read Pointer (Sn_TX_RD0) (Sn_TX_RD1)		

11.1.3.3. 内存（Memory）

W5500 有一个 16KB 的发送内存用于 Socket n 的发送缓存区，以及一个 16KB 的接收内存用于 Socket n 的接收缓存区。

16KB 的发送内存初始化分配给每个 Socket 2KB 发送缓存区（2KB*8= 16KB）。初始化分配的 2KB Socket 发送缓存，可以通过使用 Socket 发送缓存大小寄存器（Sn_TXBUF_SIZE）重新分配。一旦所有的 Socket 发送缓存大小寄存器（Sn_TXBUF_SIZE）配置完成，16KB 的发送内存就会按照配置分配给每个 Socket 的发送缓存区，并按照从 Socket 0 到 7 顺序分配。16KB 物理内存的地址是可以自增的。但是，为了避免数据传输错误，需要避免发送缓存大小寄存器（Sn_TXBUF_SIZE）的和超过 16。

16KB 的接收内存的分派方式与 16KB 的发送内存一样。16KB 的接收内存初始化被分配为每个 Socket 2KB 接收缓存区（2KB*8=16KB）。初始化分配的 2 KB Socket 接收缓存，可以通过使用 Socket 接收缓存大小寄存器（Sn_RXBUF_ SIZE）重新分配。

11.1.4. TCP、UDP 简介

UDP 是基于 IP 的简单协议，是不可靠的协议，其优点是简单、轻量化和速度快，缺点是没有流控制，没有应答确认机制，不能解决丢包、重发和错序问题。UDP 不是面向连接的，是不可靠的传输。

TCP 是面向连接的协议，也就是说，在收发数据前，必须和对方建立一个可

靠的连接。一个 TCP 连接必须要经过三次握手才能建立，三次握手结束便建立起了连接，在三次握手过程中会发送数据包，并根据数据包判断二者的收发能力。

11.1.4.1. TCP 协议

TCP 把连接作为最基本的对象，每一条 TCP 连接都有两个端点，这种端点我们叫作套接字（socket），例如，当 IP 地址为 192.3.4.16，端口号为 80，那么得到的套接字为 192.3.4.16:80。IP 协议虽然能把数据报文送到目的主机，但是并没有交付给主机的具体应用进程。TCP 报文是 TCP 层传输的数据单元，也叫报文段。TCP 报文的格式如下图 11. 8 所示。

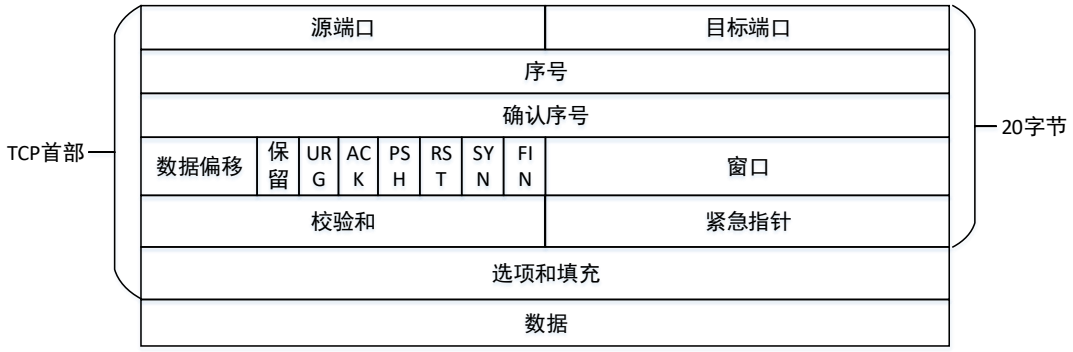


图 11. 8 TCP 报文格式图

从上图可以看出，TCP 的报文格式中能看到 6 个标志位：URG、ACK、PSH、RST、SYN、FIN，每一个标志位表示一个控制功能，对其说明如下表 11- 3 所示。TCP 的连接和断开过程与这几个标志位有关。

表 11- 3 TCP 报文标志位说明表

标 志 位	功能
URG	紧急指针标志，为 1 时表示紧急指针有效，为 0 则忽略紧急指针。
ACK	确认序号标志，为 1 时表示确认号有效，为 0 表示报文中不含确认信息，忽略确认号字段。
PSH	push 标志，为 1 表示是带有 push 标志的数据，指示接收方在接收到该报文段以后，应尽快将这个报文段交给应用程序，而不是在缓冲区排队。
RST	重置连接标志，用于重置由于主机崩溃或其他原因而出现错误的连接。或者用于拒绝非法的报文段和拒绝连接请求。
SYN	同步序号，用于建立连接过程，在连接请求中，SYN=1 和 ACK=0 表示该数据段没有使用捎带的确认域，而连接应答捎带一个确认，即 SYN=1 和 ACK=1。
FIN	finish 标志，用于释放连接，为 1 时表示发送方已经没有数据发送了，即关闭本方数据流。

1. 建立连接

TCP 的三次握手，意思就是建立连接的时候客户端和服务器之间需要三次数据包的交流：

1) 客户端发送给服务器一个请求连接数据包，即发送一个指向服务器目标端口的一个 SYN 位为 1 的 TCP 报文。

2) 服务器接收到客户端请求之后，会回应一个 SYN 位为 1 的 TCP 报文，表示同意连接，并且，会把 ACK 位也置 1 表示确认收到上次消息。

3) 客户端接收到服务器的同意连接的数据包之后，还要回复一个 ACK 为 1 的 TCP 报文，表示确认收到。

如下图 11. 9 所示，就是 TCP 的三次握手。

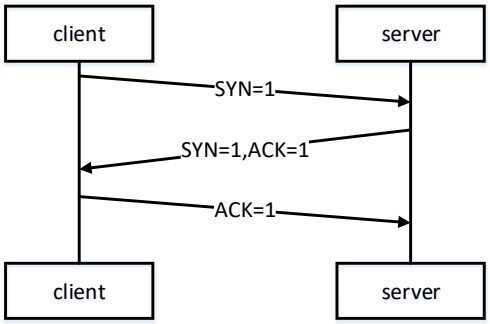


图 11. 9 TCP 三次握手示意图

2. 数据传输

TCP 以段为单位发送数据，在建立 TCP 连接的同时，也可以确定发送数据包的单位，也可以称其为“最大消息长度”（MSS:Maximum Segment Size）。最理想的情况是，最大消息长度正好是 IP 中不会被分片处理的最大数据长度。

TCP 在传输大量数据时，是以 MSS 的大小将数据进行分割发送的。进行重发时也是以 MSS 为单位。MSS 是在三次握手的时候，在两端主机之间被计算得出。两端的主机在发出建立连接的请求时，会在 TCP 首部中写入 MSS 选项，告诉对方自己的接口能够适应的 MSS 的大小（为附加 MSS 选项，TCP 首部将不再是 20 字节，而是 4 字节的整数倍）。然后会在两者之间选择一个较小的值投入使用（在建立连接时，如果某一方的 MSS 选项被省略，可以选 IP 包的长度不超过 576 字节的值（IP 首部 20 字节，TCP 首部 20 字节，MSS 536 字节））。例如客户端发送一段 3500 长度的数据到服务端，假设确定的 MSS 长度 1000，数据传输的过程如下图 11. 10 所示。

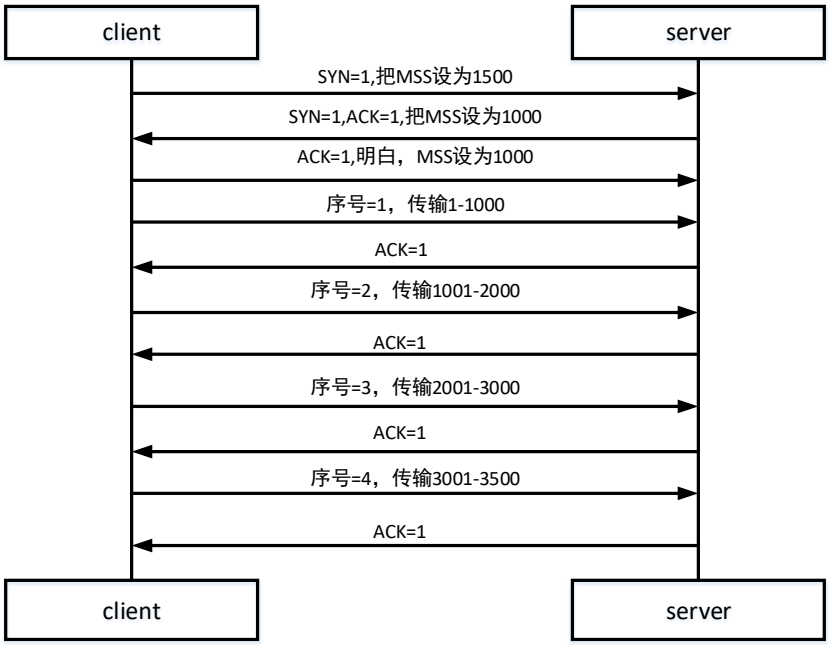


图 11. 10 数据传输示意图

1. 断开连接

TCP 的四次挥手，意思就是释放连接的时候客户端与服务器之间需要四次数据包的交流：

1) 客户端发送给服务器一个请求释放连接的数据包，即发送了一个指向服务器目标端口的一个 FIN 位为 1 的 TCP 报文，表示客户端没有数据要发送了，但是仍然可以接收数据；并且 ACK 位也为 1，表示对上次传输数据结果的确认。并且之后处去等待状态，等待服务器的两次回应。
2) 服务器接收到客户端的释放连接请求之后，会先回应一个 ACK 位为 1 的报文，表示确认收到。但是，这时服务器可能还有数据没有发送完成，继续发送数据。
3) 服务器发送完数据之后，发送一个 FIN 为 1 的 TCP 报文，表示我也没有要发送的数据了，你可以释放连接了。当然 ACK 位仍然为 1 。
4) 客户端接收到服务器的同意释放连接的数据包之后，回复一个 ACK 为 1 的 TCP 报文，表示确认收到。

如下图 11. 11 所示，这就是 TCP 的四次挥手。

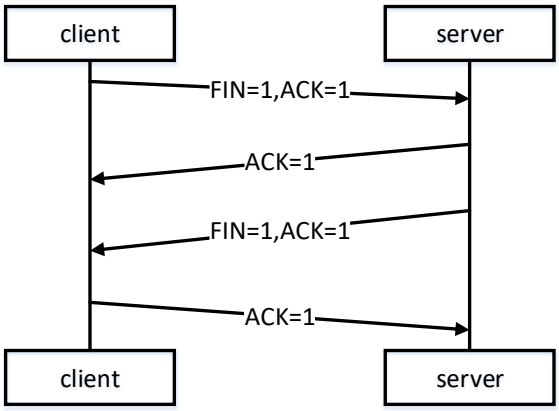


图 11. 11 TCP 的四次挥手示意图

11.1.4.2. UDP 协议

UDP 是 User Datagram Protocol 的简称，中文名用户数据报协议，是 OSI（Open System Interconnection，开放式系统互联）参考模型中一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务，UDP 在 IP 报文的协议号是 17。与 TCP（传输控制协议）协议一样，UDP 协议直接位于 IP（网际协议）协议的顶层。根据 TCP/IP 参考模型，UDP 和 TCP 都属于传输层协议。

UDP 协议的主要作用是将数据压缩成数据包的形式。一个典型的数据包就是一个二进制数据的传输单位。每一个数据包的前 8 个字节用来包含报头信息，剩余字节则用来包含具体的传输数据。

UDP 报文的具体格式如下图 11. 12 所示：

16位源端口	16位目标端口
16位长度	16位校验和
数据	

图 11. 12 UDP 报文格式图

UDP 通信过程：UDP 协议的通信较 TCP 简单了很多，减少了 TCP 的握手、确认、窗口、重传、拥塞控制等机制，UDP 是一个无状态的传输协议。UDP 客户端在发送数据时并不判断主机是否可达，服务器是否开启等问题，同样它不能确定数据是否成功送达服务器。它只是将数据简单的封了一个包，之后就丢出去了。

11.2. 实验介绍

本次实验将测试 W5500 的 UDP、TCP 客户端和 TCP 服务器模式的正常通信功能。

11.3. 建立 HQFPGA 工程

将实验五的工程复制至存放本次实验的工程文件夹下，然后参考实验二中 2.3 节的内容进行修改。

11.4. 物理管脚约束

在本次实验中将 SPI1 分配给 W5500 的 SPI 接口、GPIO[25]分配给 W5500 的复位引脚，引脚分配如下所示：

```
#----SPI1 接 w5500-----------
#SPI1_SCLK
phycst.pin.set {GPIO[11]} L15
#SPI1_CS
phycst.pin.set {GPIO[12]} K9
#SPI1_D0/SPI1_MOSI
phycst.pin.set {GPIO[13]} M15
#SPI1_D1/SPI1_MISO
phycst.pin.set {GPIO[14]} L14
#W5500_复位脚 RSTN
phycst.pin.set {GPIO[25]} M14
```

11.5. 编译设计

保存完修改后的顶层文件和物理约束文件之后，点击全部运行，生成本次实验需要的 FPGA 侧的 bin 文件，操作如图 11. 13 所示。

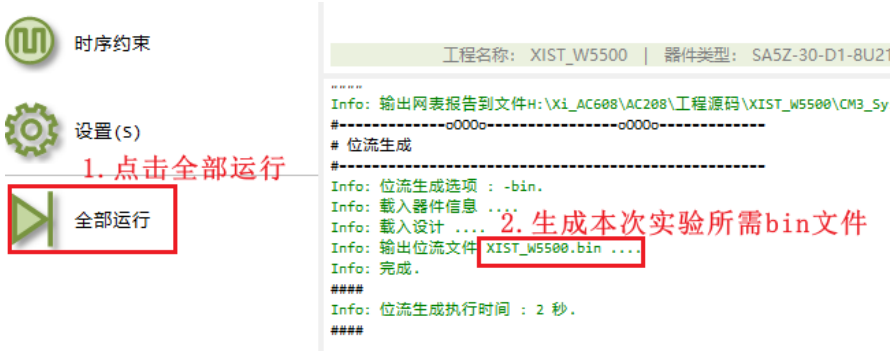


图 11. 13 编译设计

11.6. 建立 MDK 工程

找到已有的 MDK 工程，复制至本次工程的 CM3_Softerware 文件夹下，对其进行修改方式参考实验二中 2.8 节中的内容。

11.7. 软件设计

本次实验使用到的主要库文件为“W5500”，用户可以在我们提供例程的 HARDWARE 文件夹下进行复制，添加文件的步骤可以参考实验三中 3.6.1 节的内容。

11.7.1. W5500 库

打开“W5500.c”这个文件，可以看到在代码的最开始是两句条件编译语句，如下所示：

```
#define W5500_HAL 1 //选择使用硬件 SPI
#define W5500_IO 0 //选择 IO 模拟 SPI
```

上述所示的条件编译语句是关于 SPI 通信实现的选择：一种是通过 IO 模拟来实现；另外一种是通过硬件 SPI，默认使用的硬件 SPI 的方式。

11.7.1.1. W5500_SPI_Init

SPI 初始化函数，函数中包含 IO 模拟 SPI 初始化以及硬件 SPI 初始化两种方式。

首先是 IO 模拟 SPI 初始化函数 SPI_GPIO_Init()，该函数的具体内容位于“io_spi.c”，主要就是将 SPI1 的 SCLK、CS、MOSI 引脚设置为输出，MISO 引脚设置为输入，关闭其复用和中断的功能。SPI1 的引脚定义在“io_spi.h”文件中，如果想要使用 SPI0 的话修改以下引脚定义就可以了：

```
#define SPI_SCLK GPIO_Pin_11
#define SPI_CS   GPIO_Pin_12
#define SPI_MOSI GPIO_Pin_13
#define SPI_MISO GPIO_Pin_14
```

硬件 SPI 初始化，首先就是将 SPI 使用到的 CLK、MOSI、MISO 引脚进行复用，CS 信号由我们手动控制。然后就是设置 SPI 的时钟分频系数、数据位数、SPI 工作模式、传输数据的大小、传输方式、为回环模式或正常传输，函数的具体实现代码请自行查看对应的源工程代码。

在使用到 W5500 的时候，必须首先对驱动 W5500 的 SPI 进行初始化，否则将不能正常的进行数据传输，函数使用举例如下所示：

```
W5500_SPI_Init();
```

11.7.1.2. Load_Net_Parameters

装载网络参数。函数主要内容：设置端口 0 的工作模式，加载设备的网关、掩码、物理地址、本机 IP 地址、端口号、目的 IP 地址、目的端口号、端口工作模式。在该函数中并未对寄存器进行任何操作，只是对相应参数进行赋值。函数

的输入参数是一个结构体变量 Typedef_Internet * Intnet_set，结构体_Typedef_Internet 的定义位于“W5500.h”文件中，在使用到该函数的时候，需要对结构体中的相关参数进行设置，比如在本次实验中，在 main 函数定义如下：

```
_Typedef_Internet Intnet_set = {
    TCP_CLIENT,                          //端口 0 的工作模式
    192,168,90,1,                        //设备网关参数
    255,255,255,0,                       //加载设备子网掩码
    0x0c,0x29,0xab,0x7c,0x00,0x01,       //加载设备物理地址
    192,168,90,199,                      //加载设备 IP 地址
    0x13,0x88,                           //设备 0 的端口号 5000
    192,168,90,188,                      //客户端模式下目的(如 PC 机)IP 地址
    0x17,0XD6,                           //客户端模式下目的端口号 6102
    192,168,90,188,                      //UDP 模式下的目的 IP 地址
    0x17,0XD6,                           //UDP 模式下目的端口号 6102
};
```

定义上述结构体之后，在使用 Load_Net_Parameters 函数的时候调用该结构体就完成了对相关参数的设置，使用举例如下所示：

```
Load_Net_Parameters(&Intnet_set); //装载网络参数
```

11.7.1.3. W5500_Hardware_Reset

硬件复位 W5500。首先拉低 W5500 的复位引脚，然后拉高复位引脚，最后通过 Read_W5500_1Byte 函数读取通用寄存器 0x002e 的地址以确保以太网连接完成（如果读到 1，表示以太网连接完成）。

Read_W5500_1Byt 函数：读 W5500 指定寄存器的 1 个字节数据。函数内容如下所示：

1. 拉低片选，选中器件；
2. 使用硬件 SPI 时，首先就是需要设置需要读取的字节数，这里设置为 1；
3. 通过 SPI 写 16 位的寄存器地址；
4. 通过 SPI 写控制字节，在 11.1.2.1 一节中曾介绍过 W5500 的数据帧构成，其控制字段主要由区域选择位（BSB[4:0]），读/写访问模式位（RWB）以及 SPI 工作模式位（OM[1:0]）构成，这里采用 FDM 模式：1 字节访问时，OM=0x01；读访问模式，RWB=0；区域选择位未被定义，这里默认设置为 0。即此时的 SPI 控制字节为 0x01；
5. 使用硬件 SPI 时，此时使能 SPI，传输结束；
6. SPI_Read_Byte 函数读取数据；
7. 拉高片选，传输结束；
8. 返回读取到的数据。

Read_W5500_1Byte 函数的实现代码如下所示：

```
unsigned char Read_W5500_1Byte(unsigned short reg) {
    unsigned char i;
    W5500_CS(0);

    #if W5500_HAL
    SSP_ReceiveDataNum(CM3DS_MPS2_SSP1, 1);
    #endif

    SPI_Send_Short(reg); //通过 SPI 写 16 位寄存器地址
    SPI_Send_Byte(FDM1 | RWB_READ | COMMON_R); //通过 SPI 写控制字节

    #if W5500_HAL
    SSP_Cmd(CM3DS_MPS2_SSP1,ENABLE);//SPI 使能
    SSP_SendFinish(CM3DS_MPS2_SSP1);//传输结束
    #endif

    i = SPI_Read_Byte();
    W5500_CS(1);
    return i; //返回读取到的寄存器数据
}
```

W5500_Hardware_Reset 函数的使用举例如下（使用 W5500 之前必须先进行硬件复位）：

```
W5500_Hardware_Reset(); //硬件复位 W5500
```

11.7.1.4. W5500_Init

初始化 W5500 寄存器函数。函数内容：通过 Write_W5500_1Byte 函数向模式寄存器 MR 的第[7]位 RST 写入 1，将该位置‘1’之后，内部寄存器将被初始化，它会在复位之后自动清零；通过 Write_W5500_nByte 函数将 Load_Net_Parameters 函数定义的相关参数写入到寄存器中，关于寄存器的定义可以查看 11.1.3.1 中表 11- 1 的内容；通过 Write_W5500_SOCK_1Byte 函数设置发送缓冲区和接收缓冲区的大小，Sn_RXBUF_SIZE 和 Sn_TXBUF_SIZE 配置了接收缓存和发送缓存的大小，可以设置位 1，2，4，8，16Kb，如果配置为其他大小，则 W5500 不能正常的从对方主机接收数据，这里设置为 2Kb；Write_W5500_2Byte 函数将重试时间值寄存器（RTR）值设为 2000（0x07D0），即相当于 200 毫秒（100us*2000）；Write_W5500_1Byte 函数设置重试次数，默认为 8 次如果重发的次数超过设定值，则产生超时中断(相关的端口中断寄存器中的 Sn_IR 超时位(TIMEOUT)置“1”)。

Write_W5500_1Byte、Write_W5500_2Byte、Write_W5500_nByte 函数分别是

通过 SPI 向指定地址寄存器写 1 个字节数据、2 字节的数据和 n 字节的数据。这里以写 1 字节的数据为例，函数内容描述如下所示：

1. 拉低片选，选中器件；
2. 通过 SPI 写 16 位的寄存器地址；
3. 通过 SPI 写控制字节。写 1 字节和 2 字节数据时采用 FDM 模式，其对应的 SPI 工作模式位（OM[1:0]）分别为 0x01、0x02，读访问模式(第[2]位)，RWB=1(0x04)，区域选择位未被定义，这里默认设置为 0。写 n 字节的数据时采用 VDM 模式，选择通用寄存器，此时 SPI 工作模式位（OM[1:0]）为 0x00，读访问模式，RWB=1，区域选择位为 0x00。
4. 通过 SPI 写数据；
5. 使用硬件 SPI 时，首先使能 SPI，SPI 传输结束，最后失能 SPI；
6. 拉高片选，传输结束。

Write_W5500_1Byte 函数的内容如下所示：

```
void Write_W5500_1Byte(unsigned short reg, unsigned char dat) {
    W5500_CS(0);//拉低片选，开始传输数据
    SPI_Send_Short(reg); //通过 SPI 写 16 位寄存器地址
    SPI_Send_Byte(FDM1 | RWB_WRITE | COMMON_R); //通过 SPI 写控制字节
    SPI_Send_Byte(dat); //写 1 个字节数据

    #if W5500_HAL
    SSP_Cmd(CM3DS_MPS2_SSP1,ENABLE);//SPI 使能
    SSP_SendFinish(CM3DS_MPS2_SSP1);//传输结束
    SSP_Cmd(CM3DS_MPS2_SSP1,DISABLE);
    #endif

    W5500_CS(1);
}
```

W5500_Init 函数的使用举例如下所示：

```
W5500_Init(); //初始化 W5500 寄存器函数
```

11.7.1.5. Detect_Gateway

检查网关服务器函数。主要的功能就是检查网关的服务器，主要进行的操作就是检查网关及获取网关的物理地址，最后检查完成之后成功的话返回 TRUE，失败的话返回 FALSE。可以看到的是在检查网关的时候将 IP 地址每位都加上了一个‘1’，这是因为在正确设置网关的情况下，去连接一个不在同一子网（外网）的目的 IP，就可以找到网关，将目的 IP 每位加‘1’，正好造就一个不在同一子网的目的 IP，不一定要加‘1’，加 2 或者加 3 都是可以的。

该函数的实现主要调用了 Write_W5500_SOCK_1Byte 函数和 Write_W5500_

SOCK_4Byte 函数，分别代表了向指定端口寄存器写入 1 字节和 4 字节的数据。时具体的实现方式参考 11.7.1.4.中 Write_W5500_1Byte 函数的描述，需要注意的就是关于控制字节的设定，这里新增加了端口的设定，可以通过 SPI 数据帧中的区域选择寄存器(BSB[4:0]，对应控制字节[3:7])来选定对应的端口寄存器 n，对于区域选择位的说明如下表 11- 4 所示：

表 11- 4 区域选择位说明表

BSB[4:0]	含义 Meaning	BSB[4:0]	含义 Meaning
00000	选择通用寄存器	10001	选择 Socket 4 寄存器
00001	选择 Socket 0 寄存器	10010	选择 Socket 4 发送缓存
00010	选择 Socket 0 发送缓存	10011	选择 Socket 4 接收缓存
00011	选择 Socket 0 接收缓存	10100	保留位
00100	保留位	10101	选择 Socket 5 寄存器
00101	选择 Socket 1 寄存器	10110	选择 Socket 5 发送缓存
00110	选择 Socket 1 发送缓存	10111	选择 Socket 5 接收缓存
00111	选择 Socket 1 接收缓存	11000	保留位
01000	保留位	11001	选择 Socket 6 寄存器
01001	选择 Socket 2 寄存器	11010	选择 Socket 6 发送缓存
01010	选择 Socket 2 发送缓存	11011	选择 Socket 6 接收缓存
01011	选择 Socket 2 接收缓存	11100	保留位
01100	保留位	11101	选择 Socket 7 寄存器
01101	选择 Socket 3 寄存器	11110	选择 Socket 7 发送缓存
01110	选择 Socket 3 发送缓存	11111	选择 Socket 7 接收缓存
01111	选择 Socket 3 接收缓存		
10000	保留位		

函数的实现代码如下所示：

```
unsigned char Detect_Gateway(void) {
    unsigned char ip_adde[4];
    ip_adde[0] = IP_Addr[0] + 1;
    ip_adde[1] = IP_Addr[1] + 1;
    ip_adde[2] = IP_Addr[2] + 1;
    ip_adde[3] = IP_Addr[3] + 1;

    //检查网关及获取网关的物理地址
    //向目的地址寄存器写入与本机 IP 不同的 IP 值
    Write_W5500_SOCK_4Byte(0, Sn_DIPR, ip_adde);
    Write_W5500_SOCK_1Byte(0, Sn_MR, MR_TCP); //设置 socket 为 TCP 模式
    Write_W5500_SOCK_1Byte(0, Sn_CR, OPEN); //打开 Socket
    Delay(5); //延时 5ms

    if (Read_W5500_SOCK_1Byte(0, Sn_SR) != SOCK_INIT) //如果 socket 打开失败
    {
```

```
        Write_W5500_SOCK_1Byte(0, Sn_CR, CLOSE); //打开不成功,关闭 Socket
        return FALSE; //返回 FALSE(0x00)
    }

    Write_W5500_SOCK_1Byte(0, Sn_CR, CONNECT); //设置 Socket 为 Connect 模式

    do {
        unsigned char j = 0;
        j = Read_W5500_SOCK_1Byte(0, Sn_IR); //读取 Socket0 中断标志寄存器
        if (j != 0)
            Write_W5500_SOCK_1Byte(0, Sn_IR, j);
        Delay(5); //延时 5ms
        if ((j & IR_TIMEOUT) == IR_TIMEOUT) {
            return FALSE;
        } else if (Read_W5500_SOCK_1Byte(0, Sn_DHAR) != 0xff) {
            Write_W5500_SOCK_1Byte(0, Sn_CR, CLOSE); //关闭 Socket
            return TRUE;
        }
    } while (1);
    return 0;
}
```

11.7.1.6. Socket_Init

端口初始化函数。在这里我们使用的是端口 0，对端口 0 进行了初始化操作。在该函数中首先进行的设置就是 Socket 的最大传输单元 MTU(Maximum Transfer Unit)。该寄存器的地址为 0x0012~0x0013，所以我们在这里使用的是 Write_W5500_SOCK_2Byte()函数，实现向指定端口寄存器写 2 个字节数据的功能。根据 W5500 数据手册的定义，根据不同模式的选择，最大的传输单元的大小是不一样的，具体的如下表 11- 5 所示。

表 11- 5 W5500 最大传输定义表

Mode	Normal(MR(PPPoE)=‘0’)		PPPoE(MR(PPPoE)=‘1’)	
	Default MTU	Range	Default	Range
TCP	1460	1~1460	1452	1~1452
UDP	1472	1~1472	1464	1~1464
MACRAW	1514			

我们这里主要使用的是 TCP/UDP 模式，当 Socket n 处于 TCP/UDP 模式，当传输数据比 MTU 大时，数据将会被自动的划分成默认 MTU 单元大小传输。这里采用的 Normal 模式，MTU 定义为 1460。

最大传输单元定义完成之后，设置端口 0 的端口号，其源端口寄存器的地址为 0x0004~0x0005，需要传输两个字节的数据，我们在 Load_Net_Parameters 函

数中将其定义成了 5000，也就是对应的向 0x0004 中写入 0x13，0x0005 中写入 0x88，采用 Write_W5500_SOCK_2Byte()函数，需要写入的值为 0x13*256+0x88。

设置端口 0 的目的（远程）端口号，对应也就是我们电脑的端口号，我们将其设置为 6102。也就是向对应的寄存器 0x0010~0x0011 分别写入 0x17 和 0xD6。

设置端口 0 的目的（远程）IP 地址，在 Load_Net_Parameters 函数中将其定义为了“162.168.90.188”，也就是对寄存器 0x000C~0x000F 配置如下表 11- 6 所示

表 11- 6 目标 IP 对应寄存器的配置表

0x000C	0x000D	0x000E	0x000F
192(0xD0)	168(0xA8)	90(0x5A)	188(0xBC)

上述的配置通过 Write_W5500_SOCK_4Byt()函数，实现向对应的 4 个寄存器中写入对应的值的功能。函数实现代码如下所示：

```
void Socket_Init(SOCKET s) {
    //设置分片长度，参考 W5500 数据手册，该值可以不修改
    Write_W5500_SOCK_2Byte(0, Sn_MSSR, 1460); //最大分片字节数=1460
    //设置指定端口
    switch (s) {
    case 0:
        //设置端口 0 的端口号
        Write_W5500_SOCK_2Byte(0, Sn_PORT, S0_Port[0] * 256 + S0_Port[1]);
        //设置端口 0 目的(远程)端口号
        Write_W5500_SOCK_2Byte(0, Sn_DPORTR, S0_DPort[0] * 256 + S0_DPort[1]);
        //设置端口 0 目的(远程)IP 地址
        Write_W5500_SOCK_4Byte(0, Sn_DIPR, S0_DIP);
        break;
    case 1:
        break;
    case 2:
        break;
    case 3:
        break;
    case 4:
        break;
    case 5:
        break;
    case 6:
        break;
    case 7:
        break;
    default:
```

```
        break;
    }
}
```

11.7.1.7. W5500_Initialization

W5500 初始化配置。该函数主要调用了三个函数：初始化 W5500 寄存器函数 W5500_Init；检查网关服务器函数 Detect_Gateway；指定 Socket(0~7)初始化函数 Socket_Init。函数的实现代码如下所示：

```
void W5500_Initialization(void) {
    W5500_Init(); //初始化 W5500 寄存器函数
    Detect_Gateway(); //检查网关服务器
    Socket_Init(0); //指定 Socket(0~7)初始化,初始化端口 0
}
```

在使用 W5500 的时候，只需要执行该函数，便可以实现对 W5500 的所有初始化操作。

11.7.1.8. W5500_Socket_Set

W5500 端口初始化配置。根据端口的工作模式，将端口置于 TCP 服务器、TCP 客户端或者 UDP 模式，从端口的状态字节判断端口的工作情况。对于端口的运行状态定义两种模式，一种是 S_INIT，代表端口完成初始化，另外一种是 S_CONN，表示端口完成连接，可以正常传输数据。最后将端口的状态返回，提供给其它函数使用，函数的代码实现如下所示：

```
void W5500_Socket_Set(void) {
    if (S0_State == 0) //端口 0 初始化配置
    {
        if (S0_Mode == TCP_SERVER) //TCP 服务器模式
        {
            if (Socket_Listen(0) == TRUE)
                S0_State = S_INIT;
            else
                S0_State = 0;
        } else if (S0_Mode == TCP_CLIENT) //TCP 客户端模式
        {
            if (Socket_Connect(0) == TRUE)
                S0_State = S_INIT;
            else
                S0_State = 0;
        } else //UDP 模式
        {
            if (Socket_UDP(0) == TRUE)
```

```
                S0_State = S_INIT | S_CONN;
            else
                S0_State = 0;
        }
    }
}
```

11.7.1.9. W5500_Interrupt_Process

W5500 中断处理程序框架，首先去读取 W5500 的 SIR（Socket 中断寄存器）的值，SIR 指明了 Socket 的中断状态，该寄存器的每一位直到被主机置‘1’前均为‘0’。INTn 只有在 SIR 为 0x00 的时候才能被拉低，根据 S0_INT 的值判断是否产生中断。当产生中断之后，对于端口 0 的事件进行处理，处理步骤如下所示：

1. 读取 Socket 0 中断标志寄存器（Sn_IR），Sn_IR 寄存器用于提供 Socket n 中断类型信息，如建立、终止、接收数据和超时。当触发一个中断即 Sn_IMR 的对应位是‘1’的时候，Sn_IR 的对应位也将会变成‘1’。如果想把 Sn_IR 位清零的话，主机应该将该位置‘1’。

2. TCP 模式下，读取到的 Socket 0 中断标志寄存器（Sn_IR）的第 0 位（CON）为 1 的时候，Socket 0 成功连接，此时网络连接状态为 0x02，端口完成连接，可以正常传输数据。

3. 当读取到的 Socket 0 中断标志寄存器（Sn_IR）的第 1 位（DISCON）为 1 的时候，Socket 断开连接，关闭端口，等待重新打开连接，也就是向 Socket 配置寄存器（Sn_CR）中写入 0x10。初始化端口 0，并将网络的连接状态定义为 0x00，表示端口连接失败。

4. 当读取到的 Socket 0 中断标志寄存器（Sn_IR）的第 4 位（SEND_OK）为 1 的时候，表示数据发送完成，可以再次启动 Write_SOCK_Data_Buffer()函数再次发送数据。并将端口状态设置为 S_TRANSMITOK，表示端口发送一个数据包完成。

5. 当读取到的 Socket 0 中断标志寄存器（Sn_IR）的第 2 位（RECV）位为 1 的时候，表示 Socket 接收到数据，可以启动 Read_SOCK_Data_Buffer()函数继续读取数据，此时将端口 0 的状态设置为 S_RECEIVE，表示端口接收到一个数据包成功。

6. 当读取到的 Socket 0 中断标志寄存器（Sn_IR）的第 3 位（TIMEOUT）位为 1 的时候，此时表明 Socket 连接或数据传输超时，关闭端口，等待重新打开，将网络连接状态定义为 0x00，表明端口连接失败。

7. 当读取到的 Socket 0 中断寄存器（SIR）中的值不等于 0 时候，执行上

诉的步骤的 1~6。

具体的函数实现代码请自行查看对应的库函数文件。

11.7.1.10. Process_Socket_Data

W5500 接收并发送接收到的数据，该函数的输入参数为 s，代表对应的端口号，该函数首先调用 Read_SOCK_Data_Buffer()函数从 W5500 的端口接收数据缓存区读取数据，然后将读取到的数据从 Rx_Buffer 拷贝到 Tx_Buffer 缓冲区，最后调用 Write_SOCK_Data_Buffer()函数发送数据。需要注意的是如果是 UDP 模式，则收到的数据包的前 8 个字节为远方主机的 IP 和端口号，非用户数据，首先从数据报中提取得到远方主机的 IP 和端口号然后通过 size-8 得到真正的用户数据长度，如果是 UDP 模式，用户数据从 Rx_Buffer+8 位置开始。函数的代码实现如下所示：

```
void Process_Socket_Data(SOCKET s) {
    unsigned short size;
    size = Read_SOCK_Data_Buffer(s, Rx_Buffer);
    //如果是 UDP 模式，从前 8 个字节中提取出远方 IP 和端口号
    if(S0_Mode == UDP_MODE){
        UDP_DIPR[0] = Rx_Buffer[0];
        UDP_DIPR[1] = Rx_Buffer[1];
        UDP_DIPR[2] = Rx_Buffer[2];
        UDP_DIPR[3] = Rx_Buffer[3];
        UDP_DPORT[0] = Rx_Buffer[4];
        UDP_DPORT[1] = Rx_Buffer[5];
    //有效数据长度为数据包总长度减去 8,（数据包前 8 个字节为远方 IP 和端口号）
        size = size - 8;
        memcpy(Tx_Buffer, Rx_Buffer+8, size);
    }
    else
        memcpy(Tx_Buffer, Rx_Buffer, size);
    //回传接受到的数据
    Write_SOCK_Data_Buffer(s, Tx_Buffer, size);
}
```

11.7.1.11. Write_SOCK_Data_Buffer

将数据写入 W5500 的数据发送缓冲区。函数的输入参数如下表 11- 7 所示：

表 11- 7 Write_SOCK_Data_Buffer 函数输入参数说明表

参数名称	参数意义
s	端口号
dat_ptr	数据保存缓冲区指针
size	待写入数据的长度

函数内容：如果是 UDP 模式，Socket 打开失败，置目的主机的 IP 和端口号，计算实际的物理地址，写 16 位地址，写控制字节。如果最大地址未超过 W5500 发送缓冲区寄存器的最大地址，循环写入 size 个字节数据；如果最大地址超过 W5500 发送缓冲区寄存器的最大地址，循环写入前 offset 个字节数据。然后写 16 位地址、控制字节，循环写入 size-offset 个字节数据。最后更新实际物理地址，即下次写待发送数据到发送数据缓冲区的起始地址，发送启动发送命令。函数代码请自行查看对应的库文件。

11.7.2. 添加用户代码

实验内容：测试 W5500 的 UDP、TCP 客户端和 TCP 服务器模式的正常通信功能。

代码编写：首先是定义一个结构体用来存放相关网络参数的设置，通过改变结构体中的第一个参数端口 0 的工作模式来测试 UDP、TCP 客户端和 TCP 服务器模式的正常通信功能，对于模式的定义在“W5500.c”文件中，如下所示：

```
#define TCP_SERVER      0x00    //TCP 服务器模式
#define TCP_CLIENT      0x01    //TCP 客户端模式
#define UDP_MODE        0x02    //UDP(广播)模式
```

接着装载网络参数，硬件复位 W5500，对 W5500 初始化配置。W5500 初始化完成之后，对端口进行初始化配置，添加 W5500 中断处理程序，如果 Socket 0 接收到数据的话，也就是端口的状态为 S_RECEIVE，那么就可以通过 Process_Socket_Data()函数实现接收并发送接收到数据，我们在这里定义了一个 W5500_Send_Delay_Counter 的值，当该值累加至 3000 的时候，通过 Write_SOCK_Data_Buffer 函数发送指定的数据，本次实验发送的数据为“hello w5500!!!”，用户在使用的可自行定义需要发送的数据，main 函数如下所示：

```
#include "CM3DS_rcc.h"
#include "CM3DS_gpio.h"
#include "CM3DS_spi.h"
#include "CM3DS_MPS2.h"
#include "W5500.h"
#include "io_spi.h"

_Typedef_Internet Intnet_set = {
    TCP_CLIENT,                                     //端口 0 的工作模式
    192,168,90,1,                                   //设备网关参数
    255,255,255,0,                                 //加载设备子网掩码
    0x0c,0x29,0xab,0x7c,0x00,0x01,                 //加载设备物理地址
    192,168,90,199,                                //加载设备 IP 地址
    0x13,0x88,                                      //设备 0 的端口号 5000
```

```
    192,168,90,188,                                    //客户端模式下目的(如 PC 机)IP 地址
    0x17,0x70,                                            //客户端模式下目的端口号 6000
    192,168,90,188,                                    //UDP 模式下的目的 IP 地址
    0x17,0x70,                                            //UDP 模式下目的端口号 6000
};

int main(void)
{
    unsigned int W5500_Send_Delay_Counter = 0;
    W5500_SPI_Init();//SPI 初始化
    Load_Net_Parameters(&Intnet_set); //装载网络参数
    W5500_Hardware_Reset(); //硬件复位 W5500
    W5500_Initialization(); //W5500 初始化配置
    while (1) {
        W5500_Socket_Set(); //W5500 端口初始化配置
        W5500_Interrupt_Process(); //W5500 中断处理程序框架
        if ((S0_Data & S_RECEIVE) == S_RECEIVE) //如果 Socket0 接收到数据
        {
            S0_Data &= ~S_RECEIVE;
            Process_Socket_Data(0); //W5500 接收并发送接收到的数据
        } else if (W5500_Send_Delay_Counter >= 30000) //定时发送字符串
            {
            if (S0_State == (S_INIT | S_CONN)) {
                S0_Data &= ~S_TRANSMITOK;
                memcpy(Tx_Buffer, "hello w5500!!!\n", sizeof("hello w5500!!!\n"));
                Write_SOCK_Data_Buffer(0, Tx_Buffer, sizeof("hello w5500!!!\n"));
            }
            W5500_Send_Delay_Counter = 0;
        }
        W5500_Send_Delay_Counter++;
    }
}
```

11.8. 板级验证

11.8.1. 实验所需硬件

(1) AC208 开发板
(2) FPGA 下载器：XIST USB Cable
(3) CM3 仿真器：DAP Link
(4) 电源线一根
(5) 网线一根

11.8.2. 硬件连接

JTAG 和 DAP Link 的硬件连接参考实验一。

开发板的网口位于右上角（JTAG 接口旁边），在使用的用一个网口线，一边连接电脑的网口，一边连接开发板上的网口，连接成功之后，网口会亮灯。如下图 11. 14 所示：

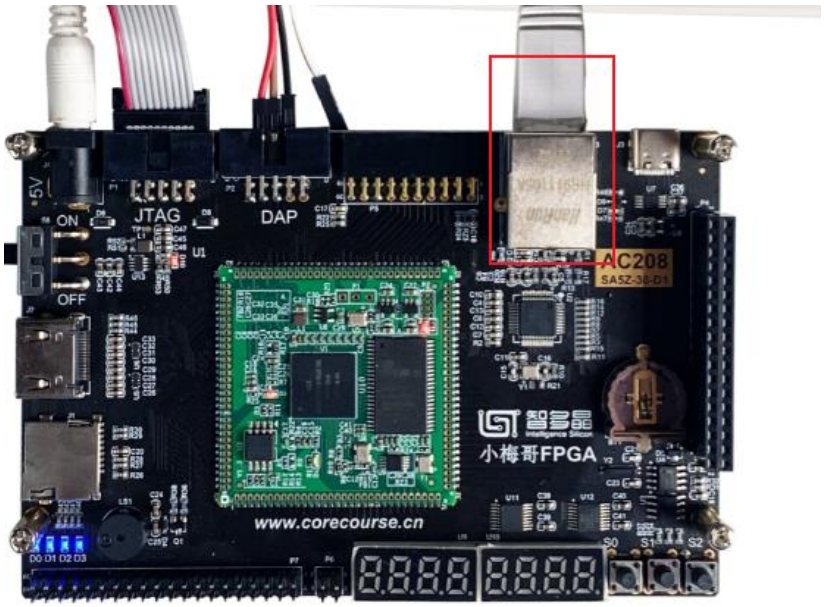


图 11. 14 开发板上网线接口

11.8.3. 下载文件至目标板

下载文件的方式参考实验一。

11.8.4. 功能演示

对于 W5500 不同的工作模式进行测试。

11.8.4.1. TCP_CLIENT（TCP 客户端模式）

由于此时 W5500 是作为客户端的，这时候测试的时候要将电脑作为服务器，这时候我们要将电脑的 IP 设置为“192.168.90.188”，这时我们打开网络调试助手，进行设置，如下图 11. 15 所示。

图 11. 15 TCP 客户端模式测试结果示意图

11.8.4.2.TCP_SERVER（TCP 服务器模式）

测试 W5500 的服务器模式，这时需要将电脑应该为客户端，代码中需要将结构体中的 TCP_CLIENT 修改为 TCP_SERVER，测试结果如图 11. 16 所示。

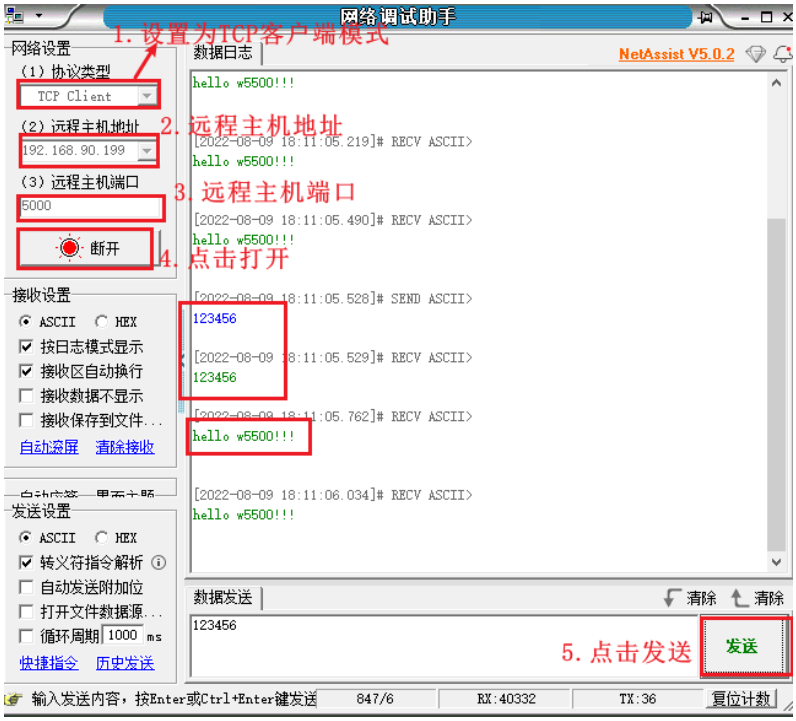


图 11. 16 TCP 服务器模式测试结果示意图

11.8.4.3.UDP_MODE（UDP 广播模式）

将结构体中的 TCP_CLIENT 修改为 UDP_MODE，将网络调试助手的协议类型修改为 UDP，测试结果如下图 11. 17 所示:

图 11. 17 UDP 模式测试结果示意图

11.9. 思考与总结

本次实验测试了 W5500 的三种工作模式：TCP 客户端模式；TCP 服务器模式；UDP 模式，这三种模式通信都测试通信成功了，实验中需要注意以下几点：

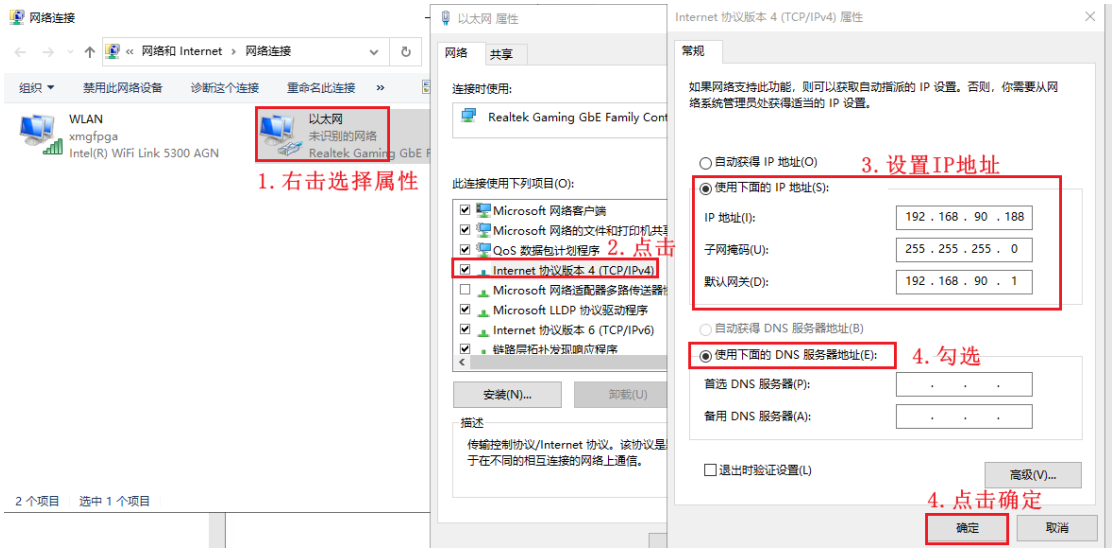
1. 当出现测试失败的时候，优先考虑电脑的防火墙是否关闭，关闭方式就是进入网络和 Internet 设置->以太网->windows 防火墙->公用网络->点击关闭。

2. 由于代码中定义了电脑的 IP 地址，与我们电脑的 IP 地址可能不一样，可以将电脑的 IP 地址修改的与我们代码一致，用户也可以根据需求修改结构体中的目的 IP 地址、子网掩码和默认网关。修改电脑 IP 地址的步骤如下所示。

(1) 找到电脑连接 WIFI 的界面，点击网络和 Internet 选项，如下所示：

(2) 进入设置界面之后，选择以太网，更改适配器选项如下所示。

(3) 进入网络连接设置界面之后，设置 IP 地址，步骤如下所示。


网络连接
网络和 Internet › 网络连接
组织
禁用此网络设备
诊断这个连接
重命名此连接
WLAN
xmgfpga
Intel(R) WiFi Link 5300 AGN
以太网
未识别的网络
Realtek Gaming GbE F
1. 右击选择属性
2 个项目　选中 1 个项目
以太网 属性
网络
共享
连接时使用:
Realtek Gaming GbE Family Cont
此连接使用下列项目(O):
Microsoft 网络客户端
Microsoft 网络的文件和打印机共
QoS 数据包计划程序
2. 点击
Internet 协议版本 4 (TCP/IPv4)
Microsoft 网络适配器多路传送器
Microsoft LLDP 协议驱动程序
Internet 协议版本 6 (TCP/IPv6)
链路层拓扑发现响应程序
安装(N)...
卸载(U)
描述
传输控制协议/Internet 协议。该协议是
于在不同的相互连接的网络上通信。
Internet 协议版本 4 (TCP/IPv4) 属性
常规
如果网络支持此功能，则可以获取自动指派的 IP 设置。否则，你需要从网
络系统管理员处获得适当的 IP 设置。
自动获得 IP 地址(O)
3. 设置IP地址
使用下面的 IP 地址(S):
IP 地址(I):
192 . 168 . 90 . 188
子网掩码(U):
255 . 255 . 255 . 0
默认网关(D):
192 . 168 . 90 . 1
自动获得 DNS 服务器地址(B)
使用下面的 DNS 服务器地址(E):
4. 勾选
首选 DNS 服务器(P):
备用 DNS 服务器(A):
退出时验证设置(L)
高级(V)...
4. 点击确定
确定
取消