iEthernet W5200

数据手册

Version 1.21

北京博控自动化技术有限公司

www.bocon.com.cn

W5200

W5200是一款硬件TCPIP协议的网络控制器， 单片机通过SPI接口可以很简单地实现Internet网络连接。W5200非常适合那些通过单芯片实现TCPIP协议栈、10/100M以太网MAC和PHY。

W5200包含一个经过多年市场验证的硬件TCPIP协议栈、 以太网的MAC和PHY。硬件TCPIP协议支持TCP、UDP、IPv4、ICMP、ARP、IGMP和PPPoE， 这些协议在各种应用领域已经得到多年的验证。W5200内部有32K的存储器用于通信数据的存储。使用W5200 ， 通过简单的端口编程，用户可以实现他们想要的以太网通信的应用，而不必要处理复杂的以太网控制。

W5200只提供SPI接口与外部MCU连接。W5200的SPI接口可以支持高达80MHz的时钟。为了降低系统功耗，W5200支持WOL（网络唤醒功能）和低功耗模式。在网络唤醒期间，W5200将要接收一个魔数数据包（magic packet），这是一个以太网底层的数据包。

特征

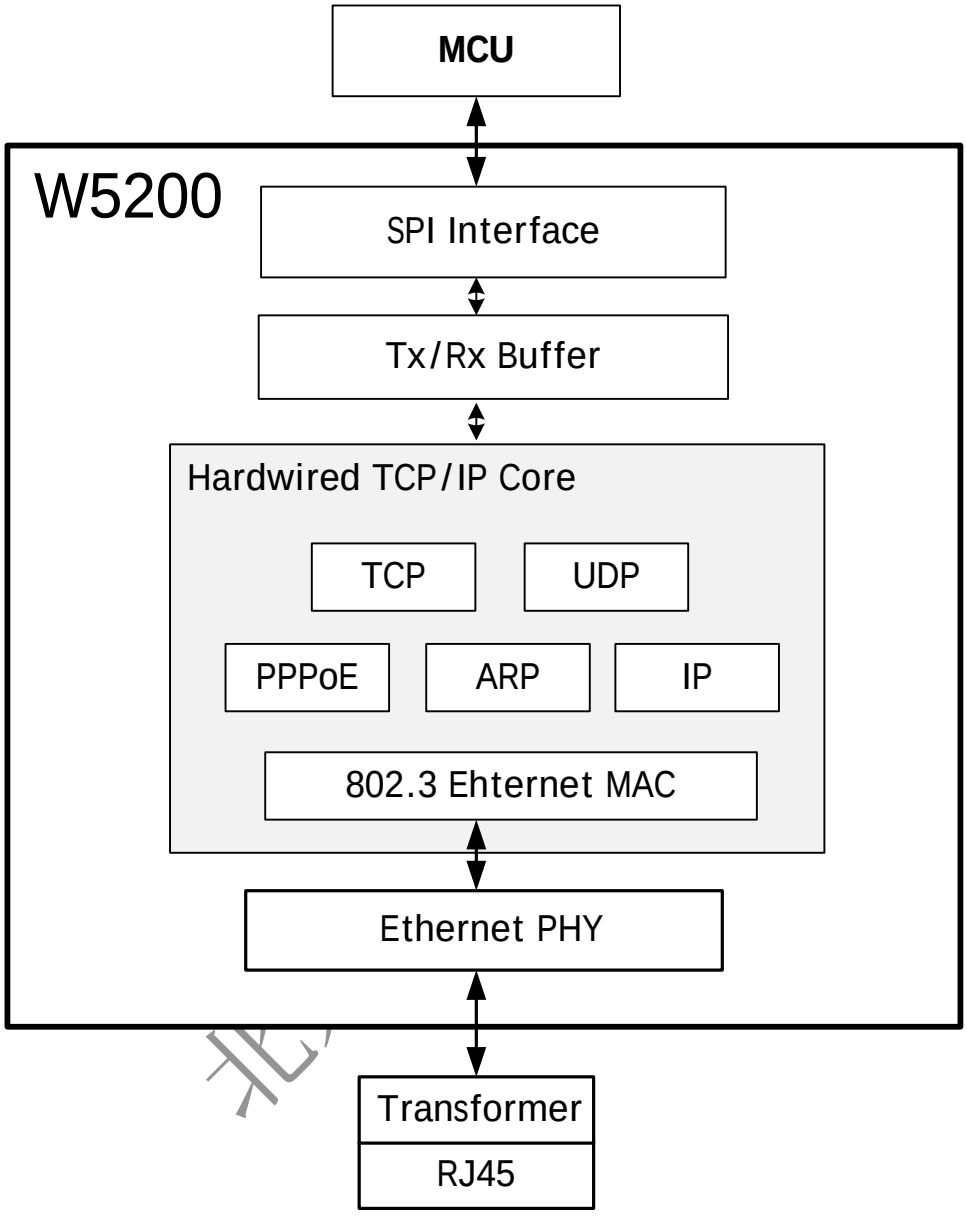
- 支持硬件TCPIP协议：TCP、UDP、ICMP、IPv4、ARP、IGMP、PPPoE和以太网
- 支持8个独立的端口（SOCKET）同时工作
- 支持低功耗模式
- 支持网络唤醒
- 支持高速SPI接口(SPI MODE 0, 3)
- 内部32K存储器用于Tx/Rx存储
- 内嵌10BaseT/100BaseTX以太网物理层（PHY）
- 支持自动握手（全双工/半双工，10/100M）
- 支持MDI/MDIX
- 支持ADSL连接（带PAP/CHAP认证模式的PPPoE协议）
- 不支持IP分片
- 3.3V工作电压，端口可承受5V电压
- 48脚QFN无铅环保封装
- 多功能LED指示输出（全双工/半双工，连接，速度等）

目标应用

W5200适用于以下的嵌入式系统：

- 家用网络设备：机顶盒、录放机、数字适配器
- 串口转以太网：存储控制器、LED显示器、无线应用中继等。
- 并口转以太网：POS机，微型打印机，复印机
- USB转以太网：存储设备，网络打印机
- GPIO转以太网：家用网络传感器
- 安保系统：数字录放设备，网络摄像机
- 工厂和建筑自动化
- 医疗监控设备
- 嵌入式服务器

功能框图


MCU
W5200
SPI Interface
Tx/Rx Buffer
Hardwired TCP/IP Core
TCP
UDP
PPPoE
ARP
IP
802.3 Ehternet MAC
Ethernet PHY
Transformer
RJ45

引脚布置

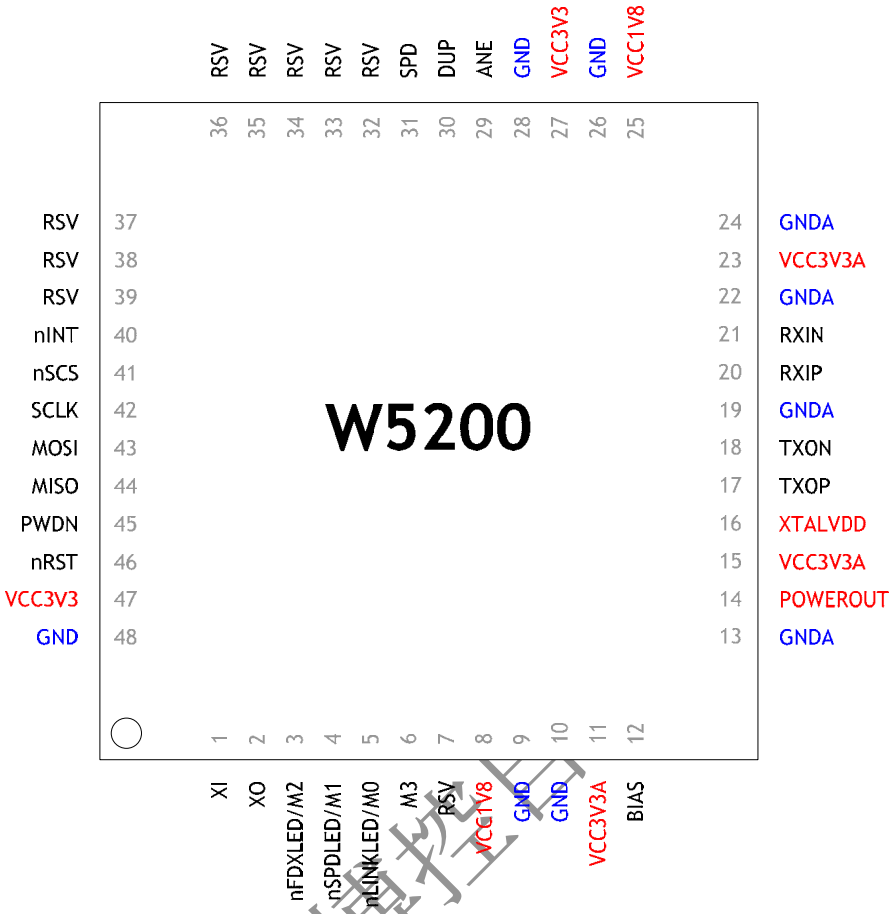


图1 W5200引脚布置

1.1 MCU接口信号

符号	类型	引脚号	描述
nRST	I	46	RESET（低电平有效） 该引脚输入低电平初始化或重新初始化W5200 RESET信号必须持续至少2us，恢复高电平至少150ms后，PLL逻辑电路稳定。RESET的时序请参考第7节“电气参数”
nSCS	I	41	SPI从模式片选（低电平有效） 该引脚输入低电平使SPI接口有效
nINT	O	40	中断（低电平有效） 在端口（SOCKET）产生连接、断开连接、数据收发、超时和网络唤醒（WOL）时，W5200通过该引脚通知MCU。通过回写IR或Sn_IR来清除中断。所有的中断都是可以屏蔽的。该引脚输出低电平有效

SCLK	I	42	SPI时钟 当使用SPI接口时，该引脚用于SPI时钟输入
MOSI	I	43	SPI主出从入 当使用SPI接口时，该引脚是SPI的MOSI（主出从入）
MISO	O	44	SPI主入从出 当使用SPI接口时，该引脚是SPI的MISO（主入从出）
PWDN	I	45	低功耗模式（高电平有效） 该引脚控制W5200进入低功耗模式 低电平：正常工作模式 高电平：低功耗模式

1.2 PHY信号

符号	类型	引脚号	描述
RXIP	I	20	RXIP/RXIN信号对 介质的差分数据信号由RXIP/RXIN端输入
RXIN	I	21	
TXOP	O	17	TXOP/TXON信号对 差分数据信号从TXOP/TXON端输出到外部介质
TXON	O	18	
BIAS	O	12	偏置电阻 通过一个28.7kΩ±1%电阻到地，请参考相关的应用设计电路
ANE	I	29	自动握手模式允许 该引脚允许/禁止自动握手模式 低电平：禁止自动握手 高电平：允许自动握手
DUP	I	30	允许全双工模式 该引脚允许/禁止全双工模式 低电平：半双工模式 高电平：全双工模式 该功能只有在复位期间有效
SPD	I	31	速度模式 该引脚选择100M/10M运行速度 低电平：10M速度 高电平：100M速度 该功能只有在复位期间有效

1.3 其它信号

符号	类型	引脚号	描述
nFDXLED/M2 nSPDLED/M1 nLINKLED/M0	I	3, 4, 5	W5200模式选择 普通运行模式：111 其它模式为芯片内部测试模式。 该功能只有在复位期间有效
M3	I	6	该引脚必须上拉
RSV	-	7 32, 33, 34, 35, 36, 37, 38, 39	保留引脚 引脚7需要上拉 除了引进7，其余的引脚可以上拉或接地

- 注意：上拉或接地电阻在40KΩ到100KΩ之间，典型值为75KΩ。

1.4 电源引脚

符号	类型	引脚号	描述
VCC3V3A	Power	11, 15, 23	3.3V模拟系统电源供电
VCC3V3	Power	27, 47	3.3V数字系统电源供电
VCC1V8	Power	8, 25	1.8V数字系统电源供电
GNDA	Ground	13, 19, 22,24	模拟地
GND	Ground	9, 10, 26, 28, 48	数字地
1V8O	O	14	1.8V电压调整输出 通过内部电源调整器输出1.8V/200mA给W5200的内核运行提供电源（VCC1V8） 在1V8O与GND之间连接一个3.3uF的钽电容以进行输出频率补偿，再连接一个0.1uF的去耦电容去出电源噪声 **注意：1V8O只给W5200内核提供电源，不能用于其它设备供电**
XTALVDD	I	16	通过一个10uF和0.1uF的电容到地

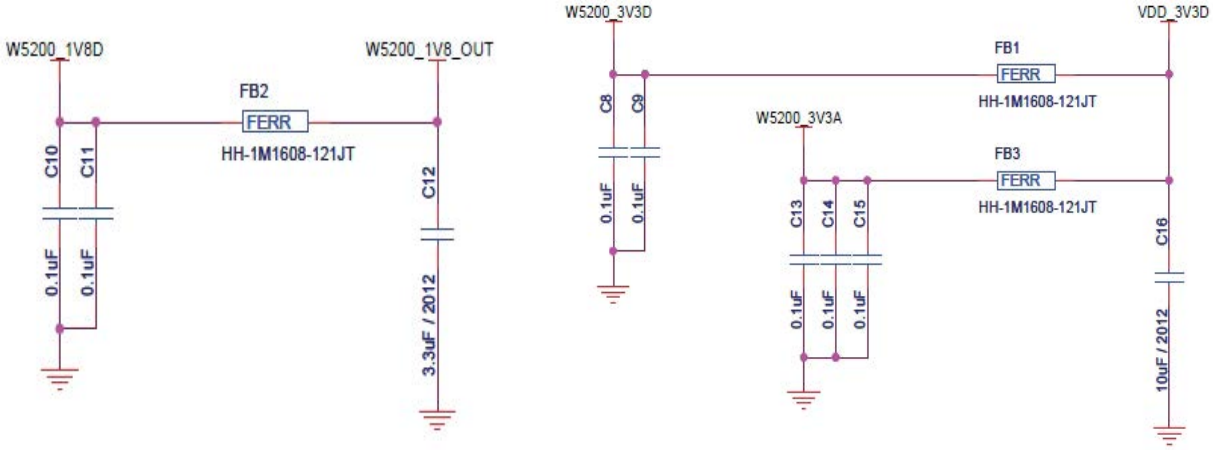


图1 电源设计

对电源设计的一点建议：

1. 本地的去耦电容应当尽量靠近W5200；
2. 地线应尽量布得很宽；
3. 如果地线空间足够的话，应尽量将模拟地和数字地分开；
4. 如果地线空间有限，那么最好就把数字地和模拟地合在一起，这样比分开布线效果更好。

1.5 时钟信号

符号	符号	引脚号	描述
XI	I	1	25MHz晶体振荡器输入/输出
XO	O	2	

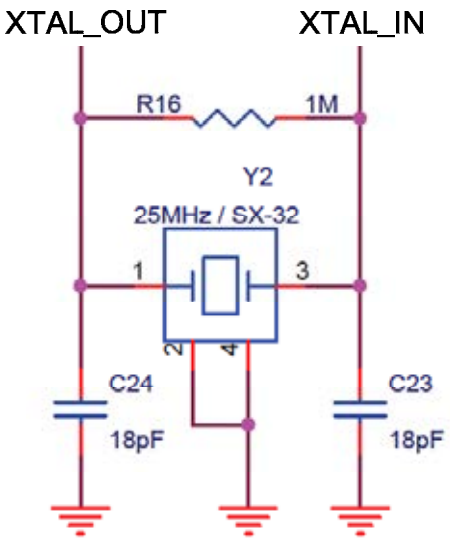


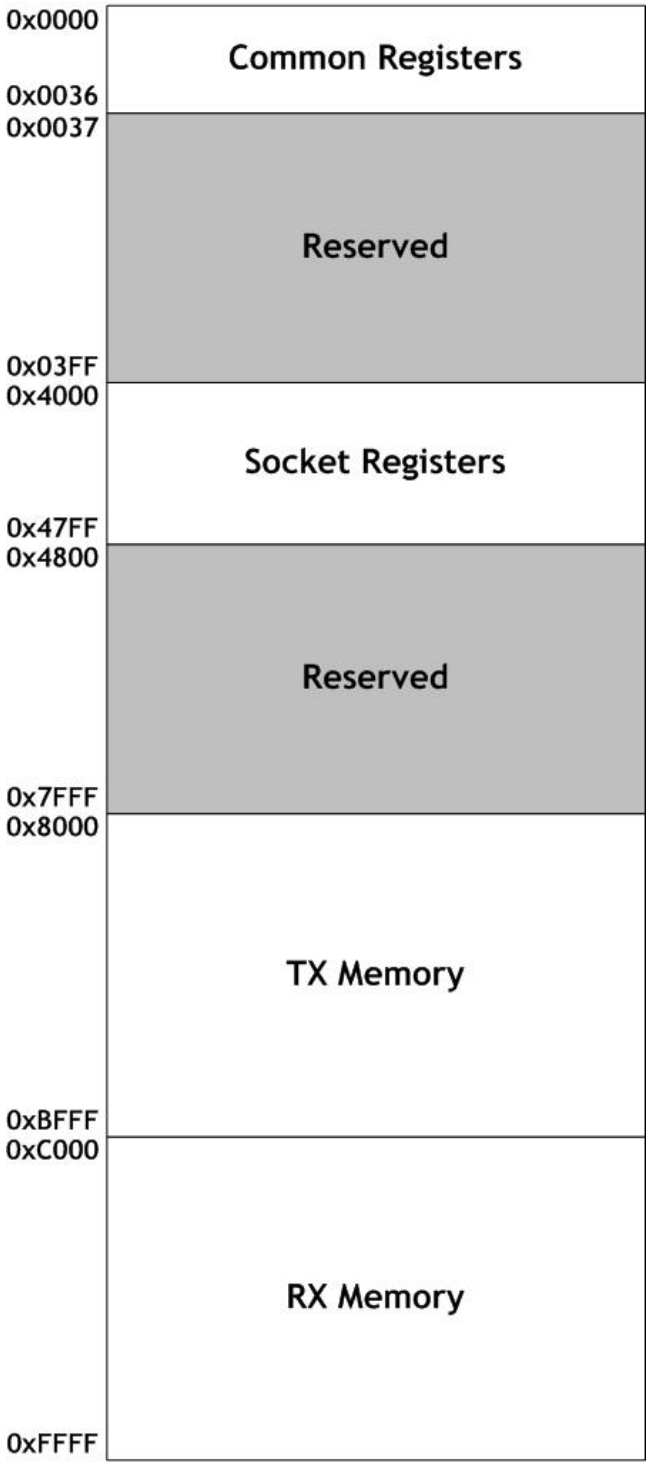
图2 晶体振荡器参考图

1.6 LED信号

符号	类型	引脚号	描述
nFDXLED/M2	O	3	**全双工/IP冲突LED指示** 低电平：全双工 高电平：半双工
nSPDLED/M1	O	4	**连接速度LED指示** 低电平：100Mbps 高电平：10Mbps
nLINKLED/M0	O	5	**连接LED指示** 低电平：10M/100M连接 高电平：未连接 闪烁：TX或RX状态

2 存储器映射

W5200的存储器由通用寄存器、端口（SOCKET）寄存器、TX存储器和RX存储器组成，如下图所示。



W5200存储器映射

3 W5200寄存器

3.1 通用寄存器

地址	寄存器
0x0000	模式寄存器(MR)
0x0001 0x0002 0x0003 0x0004	网关地址 (GAR0) (GAR1) (GAR2) (GAR3)
0x0005 0x0006 0x0007 0x0008	子网掩码 (SUBR0) (SUBR1) (SUBR2) (SUBR3)
0x0009 0x000A 0x000B 0x000C 0x000D 0x000E	本机物理地址 (SHAR0) (SHAR1) (SHAR2) (SHAR3) (SHAR4) (SHAR5)
0x000F 0x0010 0x0011 0x0012	本机IP地址 (SIPR0) (SIPR1) (SIPR2) (SIPR3)
0x0013 0x0014	保留
0x0015	中断 (IR)
0x0016	端口中断屏蔽寄存器(IMR2)
0x0017 0x0018	重复计时器 (RTR0，RTR1)
0x0019	重发计数器(RCR)
0x001A 0x001B	保留

地址	寄存器
0x001C 0x001D	PPPoE认证类型 (PATR0)，(PATR1)
0x001E	PPPoE认证算法 (PPPALGO)
0x001F	版本号(VERSIONR)
0x0020 ~ 0x0027	保留
0x0028	PPP LCP 请求时间 (PTIMER)
0x0029	PPP LCP魔数(PMAGIC)
0x002A ~ 0x002F	保留
0x0030 0x0031	中断信号低电平定时器 (INTLEVEL0) (INTLEVEL1)
0x0032 ~ 0x0033	保留
0x0034	端口中断 (IR2)
0x0035	物理层状态寄存器 (PSTATUS)
0x0036	中断屏蔽寄存器 (IMR)

3.2 Socket registers端口寄存器

注意：n是端口数（0, 1, 2, 3, 4, 5, 6, 7 ）

Address	Register
0x4n00	Socket n模式寄存器（Sn_MR）
0x4n01	Socket n命令寄存器（Sn_CR）
0x4n02	Socket n中断寄存器（Sn_IR）
0x4n03	Socket n状态寄存器（Sn_SR）
0x4n04 0x4n05	Socket n端口号 (SN_PORT0) (SN_PORT1)
0x4n06 0x4n07 0x4n08 0x4n09 0x4n0A 0x4n0B	Socket n目的物理地址寄存器 (Sn_DHAR0) (Sn_DHAR1) (Sn_DHAR2) (Sn_DHAR3) (Sn_DHAR4) (Sn_DHAR5)
0x4n0C 0x4n0D 0x4n0E 0x4n0F	Socket n目的IP地址寄存器 (Sn_DIPR0) (Sn_DIPR1) (Sn_DIPR2) (Sn_DIPR3)
0x4n10 0x4n11	Socket n目的端口号寄存器 (Sn_DPORT0) (Sn_DPORT1)
0x4n12 0x4n13	Socket n最大分片尺寸 (Sn_MSSR0) (Sn_MSSR1)
0x4n14	IPRAW模式Socket n协议类型 (Sn_PROTO)
0x4n15	Socket n IP TOS (Sn_TOS)
0x4n16	Socket n IP TTL (Sn_TTL)
0x4n17 ~ 0x4n1D	保留

Address	Register
0x4n1E	接收存储器大小寄存器 (Sn_RXMEM_SIZE)
0x4n1F	发送存储器大小寄存器 (Sn_TXMEM_SIZE)
0x4n20 0x4n21	Socket n发送剩余空间寄存器 (Sn_TX_FSR0) (Sn_TX_FSR1)
0x4n22 0x4n23	Socket n TX读指针 (Sn_TX_RD0) (Sn_TX_RD1)
0x4n24 0x4n25	Socket n TX写指针 (Sn_TX_WR0) (Sn_TX_WR1)
0x4n26 0x4n27	Socket n接收数据字节大小寄存器 (Sn_RX_RSR0) (Sn_RX_RSR1)
0x4n28 0x4n29	Socket n RX读指针 (Sn_RX_RD0) (Sn_RX_RD1)
x4n2A 0x4n2B	Socket n RX写指针 (Sn_RX_WR0) (Sn_RX_WR1)
0x4n2C	Socket n中断屏蔽寄存器 (Sn_IMR)
0x4n2D 0x4n2E	IP头中分片偏移 (Sn_FRAG0) (Sn_FRAG1)
0x4n30 ~ 0x4nFF	保留

4 寄存器描述

4.1 通用寄存器

MR (模式寄存器) [R/W] [0x0000] [0x00]

该寄存器实现软件复位，设置Ping阻止模式和设置PPPoE模式。

7	6	5	4	3	2	1	0
RST			PB	PPPoE			

位	符号	描述
7	RST	**软件复位** 该位置‘1’时内部寄存器将被初始化，复位后该位自动清‘0’
6	Reserved	保留
5	Reserved	保留
4	PB	**设置Ping阻止模式** 0：允许Ping功能 1：禁止Ping功能 如果设置该位为‘1’，将不响应Ping请求
3	PPPoE	**PPPoE模式** 0：禁止PPPoE模式 1：允许PPPoE模式 如果用户没有经过路由器而直接使用ADSL，该位需要置‘1’，与ADSL服务器连接。详细过程参考应用笔记“怎样与ADSL连接”
2	Reserved	保留
1	Reserved	保留
0	Reserved	保留

GAR (网关IP地址寄存器) [R/W] [0x0001 – 0x0004] [0x00]

该寄存器设置默认的网关IP地址。例：网关IP地址为：192.168.0.1

0x0001	0x0002	0x0003	0x0004
192 (0xC0)	168 (0xA8)	0 (0x00)	1 (0x01)

SUBR (子网掩码寄存器) [R/W] [0x0005 – 0x0008] [0x00]

该寄存器设置子网掩码。例：子网掩码值为：255.255.255.0

0x0005	0x0006	0x0007	0x0008
255 (0xFF)	255 (0xFF)	255 (0xFF)	0 (0x00)

SHAR (本机物理地址寄存器) [R/W] [0x0009 – 0x000E] [0x00]

该寄存器设置本机物理地址。例：本机物理地址为：00.08.DC.01.02.03

0x0009	0x000A	0x000B	0x000C	0x000D	0x000E
0x00	0x08	0xDC	0x01	0x02	0x03

SIPR (本机IP地址寄存器) [R/W] [0x000F – 0x0012] [0x00]

该寄存器设置本机IP地址。例：本机IP地址为：192.168.0.2

0x000F	0x0010	0x0011	0x0012
192 (0xC0)	168 (0xA8)	0 (0x00)	2 (0x02)

IR (中断寄存器) [R] [0x0015] [0x00]

主处理器通过访问该寄存器可以获得产生中断的类型。任何中断都可以通过中断屏蔽寄存器IMR进行屏蔽。一旦相应的中断屏蔽位被设置，且该寄存器的位置‘1’，INT引脚将保持低电平。直到该寄存器的位都被清‘0’。

7	6	5	4	3	2	1	0
CONFLICT	Reserved	PPPoE	Reserved	Reserved	Reserved	Reserved	Reserved

位	符号	描述
7	CONFLICT	IP地址冲突 当收到一个相同IP地址的ARP请求时，这一位置‘1’。向该位写‘1’则清0
6	Reserved	保留
5	PPPoE	PPPoE连接关闭 在PPPoE模式，如果PPPoE连接关闭，则该位置‘1’。向该位写‘1’则清0
4	Reserved	保留
3	Reserved	保留
2	Reserved	保留
1	Reserved	保留
0	Reserved	保留

IMR (中断屏蔽寄存器) [R/W] [0x0036] [0x00]

中断屏蔽寄存器用于屏蔽中断。每个中断屏蔽寄存器中的位对应中断寄存器（IR）中的位。如果中断屏蔽寄存器的位被置‘1’，相应的中断寄存器（IR）中的位被置‘1’时，将产生一个中断。如果中断屏蔽寄存器的位清‘0’，那么相对应的中断寄存器的位置‘1’时不会产生中断。

7	6	5	4	3	2	1	0
IM_IR7	Reserved	IM_IR5	Reserved	Reserved	Reserved	Reserved	Reserved

位	符号	描述
7	IM_IR7	IP地址冲突中断允许
6	Reserved	保留
5	IM_IR5	PPPoE关闭中断允许
4	Reserved	保留
3	Reserved	保留
2	Reserved	保留
1	Reserved	保留
0	Reserved	保留

RTR (重试时间值寄存器) [R/W] [0x0017 – 0x0018] [0x07D0]

它配置重新传送的超时时间周期。RTR的标准单位是100us。 RTR初始化后的设置值为2000（0x07D0），即200ms的超时周期。

例：当超时时间设置为400ms, RTR = (400ms / 1ms) X 10 = 4000(0x0FA0)

0x0017	0x0018
0x0F	0xA0

当执行CONNECT、DISCON、CLOSE、SEND、SEND_MAC和SEND_KEEP命令时，如果对端没有响应，或响应延迟，则会产生重新传送的过程。

RCR (重试计数寄存器) [R/W] [0x0019] [0x08]

该寄存器用于设置重新传送的次数。当重新传送发生的次数超过“ RCR+1 ”，则会产生超时中断。（中断寄存器（Sn_IR）的“ 中断 ”位设置为‘ 1 ’）。

在TCP通讯中，Sn_SR（Socket n的状态寄存器)的值将改变为“ SOCK_CLOSED ”和Sn_IR（Socket n的中断寄存器）（TIMEOUT）会改变为‘ 1 ’。如果不是TCP通讯，就只有Sn_IR（TIMEOUT）会变为‘ 1 ’。

例：RCR = 0x0007

0x0019
0x07

W5200的超时可以通过RTR和RCR进行配置。W5200超时有ARP超时和TCP重新传送超时。

在ARP（参考RFC 826, http://www.ietf.org/rfc.html）时重发超时，W5200自动发送ARP请求到IP地址的对端， 以获得对端的MAC地址信息 （ 用于IP ， UDP或TCP的通信 ） 。在等待对端的ARP响应时， 如果在RTR时间内没有得到响应， 那么将产生ARP超时并重新发送ARP请求， 直到重复“ RCR+1 ”次。

在ARP请求重复“ RCR+1 ”次时仍然没有得到ARP响应，最终则产生超时中断，且Sn_IR(TIMEOUT)置‘ 1 ’。

ARP请求的最终超时值(ARP_{TO})计算如下：

$$ARP_{TO} = (RTR \times 0.1ms) \times (RCR + 1)$$

在TCP数据包传输超时过程中，W5200将发送TCP数据包（SYN，FIN，RST，DATA数据包），并在RTR和RCR所设定的时间和次数范围内等待对端的响应(ACK)。如果对端没有响应(ACK)，将产生超时并重新传送TCP数据包，TCP重新发送“ RCR+1 ”次仍然没有得到对端的响应(ACK)，最终将产生超时中断，Sn_SR改变为“ SOCK_CLOSED ”，同样Sn_IR(TIMEOUT)置‘ 1 ’。

TCP数据包重新传送的最终超时(TCP_{TO})可以用以下公式计算：

$$TCP_{TO} = \left(\sum_{N=0}^{M} (RTR \times 2^N) + ((RCR-M) \times RTR_{MAX}) \right) \times 0.1ms$$

N：重发计数， 0<= N <= M
M：满足$RTR \times 2^{(M+1)} > 65535$ and 0 <= M <= RCR最小值
RTR_{MAX}: $RTR \times 2^M$

例：当RTR = 2000(0x07D0), RCR = 8(0x0008),

ARP_{TO} = 2000 X 0.1ms X 9 = 1800ms = 1.8s

TCP_{TO} = (0x07D0 + 0x0FA0 + 0x1F40 + 0x3E80 + 0x7D00 + 0xFA00 + 0xFA00 + 0xFA00 + 0xFA00) X 0.1ms

= (2000 + 4000 + 8000 + 16000 + 32000 + ((8 + 4) X 64000)) X 0.1ms

= 318000 X 0.1ms = 31.8s

PATR (PPPoE模式下的认证类型) [R] [0x001C-0x001D] [0x0000]

该寄存器通知认证的类型用与PPPoE服务器建立连接。W5200支持两种类型的认证方法 - PAP和CHAP。

值	认证类型
0xC023	PAP
0xC223	CHAP

PPPALGO(PPPoE模式下的认证算法)[R][0x001E][0x00]

在PPPoE模式下该寄存器通知认证算法。如需详细信息，请参阅PPPoE的应用说明

VERSIONR (W5200版本号)[R][0x001F][0x03]

该寄存器为W5200的版本号寄存器。

PTIMER (PPP连接控制协议请求定时器寄存器) [R/W] [0x0028]

该寄存器是指示发送LCP Echo请求的期限。PTIMER的值为1时，大约是25毫秒(ms)。

例：PTIMER为200,，200 * 25(ms) = 5000(ms) = 5 seconds

PMAGIC (PPP连接控制协议魔数寄存器) [R/W] [0x0029][0x00]

该寄存器用于选择LCP魔数(Magic number)。请参考应用笔记中的“ 如何连接ADSL ”。

INTLEVEL(中断信号低电平时间寄存器)[R/W][0x0030 – 0x0031][0x0000]

该寄存器用于设置中断有效等待的时间(IAWT)。它配置nINT低电平等待时间直到产生下一个中断。

I_{AWT} = (INTLEVEL0 + 1) * PLL_CLK (when INTLEVEL0 > 0)

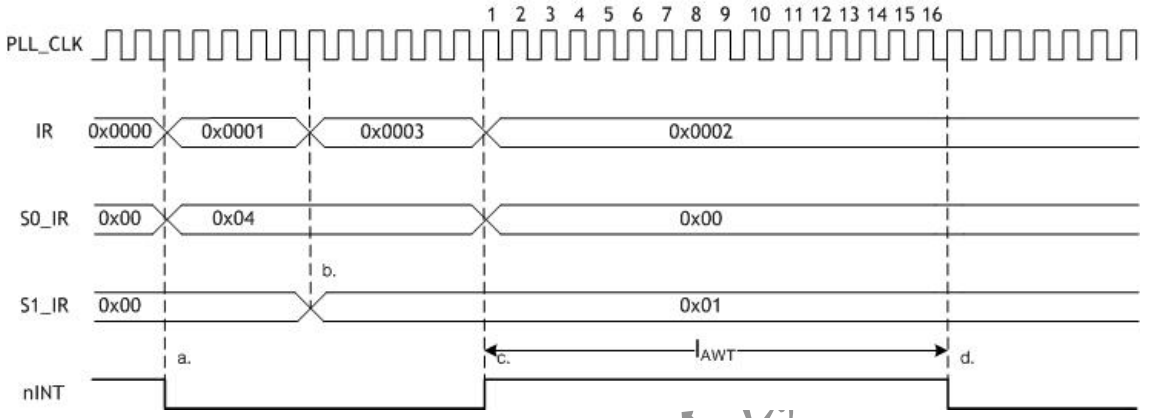


图3 中断低电平计时

a. 在Socket 0中，接收中断发生(S0_IR3=1)，相应的IR2位设置为‘ 1 ’(IR(S0_IR)=1)。那么nINT信号为低电平。

b. 在Socket 1中, 连接中断发生(S1_IR(0)=1)和相应的位设置为‘ 1 ’ (IR(S1_IR)=1)。

c. 主机清零S0_IR (S0_IR=0x00)，相应的IR位(bit)也会自动被清零(IR(S0_IR)= 0)。nINT信号转高。

d. S0_IR被清零了。IR2不是0x00， 那么在1个PLL_CLK后， nINT应为低电平。然而， 由于INTLEVEL0是0x000F，所以在IAWT(16 PLL_CLK)之后IR中断为有效。

IR2(W5200端口中断寄存器)[R/W][0x0034][0x00]

IR2寄存器用于通知主机W5200中的SOCKET产生中断。任何中断产生时，IR2相关的位会被置‘ 1 ’。当相关的屏蔽位被设置为‘ 1 ’时，nINT信号输出低电平。nINT会保持低电平直到所有Sn_IR的位被被清‘ 0 ’。如果有Sn_IR的所有位被清‘ 0 ’时，nINT会自动变为高电平。

7	6	5	4	3	2	1	0
S7_INT	S6_INT	S5_INT	S4_INT	S3_INT	S2_INT	S1_INT	S0_INT

位	符号	描述
7	S7_INT	当SOCKET7有一个中断发生，S7_INT就变成‘1’。该中断的信息在S7_IR中。当主机将S7_IR清0，该位(bit)会被自动清‘0’
6	S6_INT	当SOCKET6有一个中断发生，S6_INT就变成‘1’。该中断的信息在S6_IR中。当主机将S6_IR清0，该位(bit)会被自动清‘0’
5	S5_INT	当SOCKET5有一个中断发生，S5_INT就变成‘1’。该中断的信息在S5_IR中。当主机将S5_IR清0，该位(bit)会被自动清‘0’
4	S4_INT	当SOCKET4有一个中断发生，S4_INT就变成‘1’。该中断的信息在S4_IR中。当主机将S4_IR清0，该位(bit)会被自动清‘0’
3	S3_INT	当SOCKET3有一个中断发生，S3_INT就变成‘1’。该中断的信息在S3_IR中。当主机将S3_IR清0，该位(bit)会被自动清‘0’
2	S2_INT	当SOCKET2有一个中断发生，S2_INT就变成‘1’。该中断的信息在S2_IR中。当主机将S2_IR清0，该位(bit)会被自动清‘0’
1	S1_INT	当SOCKET1有一个中断发生，S1_INT就变成‘1’。该中断的信息在S1_IR中。当主机将S1_IR清0，该位(bit)会被自动清‘0’
0	S0_INT	当SOCKET0有一个中断发生，S0_INT就变成‘1’。该中断的信息在S0_IR中。当主机将S0_IR清0，该位(bit)会被自动清‘0’

PHYSTATUS(W5200 PHY状态寄存器)[R/W][0x17]

指示W5200的PHY状态。

位	符号	描述
7	Reserved	保留
6	Reserved	保留
5	LINK	**连接状态寄存器[只读]** 该寄存器指示W5200物理连接状态 0：连接断开 1：建立连接
4	Reserved	保留
3	POWERDOWN	**PHY的低功耗模式[只读]** 该寄存器指示W5200低功耗模式状态 0：低功耗模式禁止（正常运行模式） 1：低功耗模式开启
2	Reserved	保留
1	Reserved	保留
0	Reserved	保留

IMR2(中断屏蔽寄存器2)[R/W][0x0016][0x00]

该中断屏蔽寄存器是用来屏蔽中断。每个中断屏蔽位对应于中断寄存器2（IR2）的位。如果中断屏蔽的位设置为‘1’，当中断寄存器2（IR2）的位置‘1’时，中断将会产生。如果IMR的位置‘0’，那么该位将不会产生中断。

7	6	5	4	3	2	1	0
S7_INT	S6_INT	S5_INT	S4_INT	S3_INT	S2_INT	S1_INT	S0_INT

位	符号	描述
7	S7_INT	IR(S7_INT)中断屏蔽位
6	S6_INT	IR(S6_INT)中断屏蔽位
5	S5_INT	IR(S5_INT)中断屏蔽位
4	S4_INT	IR(S4_INT)中断屏蔽位
3	S3_INT	IR(S3_INT)中断屏蔽位
2	S2_INT	IR(S2_INT)中断屏蔽位
1	S1_INT	IR(S1_INT)中断屏蔽位
0	S0_INT	IR(S0_INT)中断屏蔽位

4.2 端口(SOCKET)寄存器

S*n*[1]_MR (Socket n的模式寄存器) [R/W] [0x4000+0x0n00] [0x00][2]

该寄存器设置每个端口（SOCKET）的功能选项和协议类型。

7	6	5	4	3	2	1	0
MULTI		ND / MC		P3	P2	P1	P0

位	符号	描述
7	MULTI	**多播，**只有在UDP模式下有效(P3-P0：0010) 0：禁止多播 1：允许多播 为了使用多播，在打开SOCKET之前，将多播分组的IP地址写到SOCKET n的目的IP寄存器，将多播分组的端口号写到SOCKET n的目的端口号寄存器
6	MF	MAC地址过滤，只用于MACRAW（P3~P0：0100）模式。 0：禁止MAC地址过滤 1：允许MAC地址过滤 当这一位置‘ 1 ’时，W5200只接收属于自己的数据包和广播的数据包。当这一位为‘ 0 ’时，W5200接收以太网上所有的数据包。当使用混合TCPIP协议栈时，建议将该位置‘ 1 ’，以免主机过渡接收数据包
5	ND/MC	**使用无延迟的应答（ACK）** 0：禁止无延迟ACK选项 1：允许无延迟ACK选项 只在TCP模式（P3-P0：0001）下有效。如果这一位设置为‘ 1 ’，在收到对端的数据包后马上响应一个ACK数据包。如果这一位清‘ 0 ’，ACK将根据内部时间溢出机制进行响应 **多播** 0：使用IGMP版本2 1：使用IGMP版本1 该位只有在MULTI置‘ 1 ’、且使用UDP模式（P3-P0：0010）时有效。另外多播也可以在诸如在线/离开/报告给多播分组等的IGMP信息中发送版本号
4	Reserved	保留

[1] *n* is Socket n的端口编号 (0, 1, 2, 3, 4, 5, 6, 7).

[2] [Read / Write] [address of socket 0, address of socket 1, address of socket 2, address of socket 3,address of socket 4, address of socket 5, address of socket 6, address of socket 7] [Reset value]

<table>
<tr><td>3</td><td>P3</td><td rowspan="4">协议
设置端口的工作模式：TCP，UDP，或IP RAW等
<table><tr><th>P3</th><th>P2</th><th>P1</th><th>P0</th><th>意义</th></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>端口关闭</td></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>TCP</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>UDP</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>IPRAW</td></tr></table>
* SOCKET0可工作在MACRAW模式和PPPoE模式
<table><tr><th>P3</th><th>P2</th><th>P1</th><th>P0</th><th>Meaning意义</th></tr><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>MACRAW</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>PPPoE</td></tr></table>
S0_MR_MACRAW和S0_MR_PPPoE只能在SOCKET0工作
S0_MR_PPPoE是临时用于与PPPoE服务器的连接和中断，连接建立以后，该端口可以用于其它协议</td></tr>
<tr><td>2</td><td>P2</td></tr>
<tr><td>1</td><td>P1</td></tr>
<tr><td>0</td><td>P0</td></tr>
</table>

S*n*_CR (Socket n 命令寄存器) [R/W] [0x4001+0x0n00] [0x00]

该寄存器用于设置SOCKET n的命令，诸如：打开、关闭、连接、侦听、发送、接收等动作。W5200确认该命令以后，Sn_CR寄存器自动清零(0x00)。即使Sn_CR清零，指令仍然在处理中。为了验证命令是否执行完成，可以检查Sn_IR寄存器或Sn_SR寄存器。

<table>
<tr><th>值</th><th>符号</th><th>描述</th></tr>
<tr><td>0x01</td><td>OPEN</td><td>根据Sn_MR(P3:P0)所选择的协议初始化并打开SOCKET n。下表显示对应Sn_MR的Sn_SR的值
<table><tr><th>Sn_MR(P3:P0)</th><th>Sn_SR</th></tr><tr><td>Sn_MR_CLOSE (0x00)</td><td>-</td></tr><tr><td>Sn_MR_TCP (0x01)</td><td>SOCK_INIT (0x13)</td></tr><tr><td>Sn_MR_UDP (0x02)</td><td>SOCK_UDP (0x22)</td></tr><tr><td>Sn_MR_IPRAW (0x03)</td><td>SOCK_IPRAW (0x32)</td></tr><tr><td>S0_MR_MACRAW (0x04)</td><td>SOCK_MACRAW (0x42)</td></tr><tr><td>S0_MR_PPPoE (0x05)</td><td>SOCK_PPPoE (0x5F)</td></tr></table></td></tr>
<tr><td>0x02</td><td>LISTEN</td><td>该命令只有在TCP模式(Sn_MR(P3:P0)=Sn_MR_TCP)下有效。在这种模式下，SOCKET n配置为TCP服务器，等待其它TCP客户端的连接请求(SYN数据包)。Sn_SR寄存器从SOCK_INIT状态改变为SOCK_LISTEN状态

当客户端的连接请求成功建立，Sn_SR寄存器的状态将从SOCK_LISTEN变为SOCK_ESTABLISHED，且Sn_SR(0)的值置‘1’。如果连接失败(SYN/ACK数据包发送失败)，Sn_SR(3)置‘1’且Sn_SR变为SOCK_CLOSED</td></tr>
</table>

		例如，如果在连接请求过程中TCP客户端的目标端口号不存在，W5200将发送一个RST数据包，且Sn_SR的状态不变
0x04	CONNECT	该命令只有在TCP模式下有效， 且SOCKET n设置为TCP客户端。端口向TCP服务器发出连接请求(SYN数据包)。TCP服务器的IP地址和端口号由目的IP地址寄存器(Sn_DIPR0)和目的端口号寄存器(Sn_DPORT0)确定 当客户端的连接请求成功建立， Sn_SR将改变为SOCK_ESTABLISHED，且Sn_IR(0)改变为‘1’。在下面几种情况下，连接请求失败： ● 当一个ARP超时异常发生(Sn_IR(s)=1)，表明没有能够通过ARP过程获得目的硬件地址 ● 如果没有接收到SYN/ACK数据包，且产生TCP超时异常(Sn_IR(3)=1) ● 如果接收到RST数据包而不是SYN/ACK数据包 在以上这些情况下，Sn_SR状态改变为SOCK_CLOSED
0x08	DISCON	只有在TCP模式下有效 不管是在TCP服务器还是TCP客户端，该命令作断开连接的处理 主动关闭：它发送一个断开连接的请求（FIN数据包）到连接的对端 被动关闭：当收到对端的FIN数据包时，回复一个FIN数据包到对端 当收到FIN/ACK数据包时，Sn_SR的状态改变为SOCK_CLOSED 如果没有收到断开连接的请求，将产生TCP超时(Sn_IR(3)=1)，且Sn_SR的状态改变为SOCK_CLOSED 例如：如果用CLOSE命令代替DISCON，Sn_SR的状态将变成CLOSED，但不会有断开连接的过程（不产生断开连接的请求）。如果在通信过程中接收到RST数据包，Sn_SR无条件地改变为SOCK_CLOSED
0x10	CLOSE	关闭SOCKET Sn_SR的状态改变为SOCK_CLOSED
0x20	SEND	SEND命令将TX数据存储器中的所有数据都发送出去。详细资料请查看SOCKET n的TX剩余空间大小寄存器(Sn_TX_FSR)，SOCKET n的TX写指针寄存器(Sn_TX_WR)和SOCKET n的TX读指针寄存器(Sn_TX_RD)
0x21	SEND_MAC	只有在UDP模式有效 其基本操作与SEND相同。但SEND操作需要通过ARP（地址解析协议）过程获得目的硬件物理地址。SEND_MAC不需要通过ARP获得目的硬件物理地址，由用户设置SOCKET n的目的硬件物理地址
0x22	SEND_KEEP	只有在TCP模式下有效 通过发送一个字节的数据检查连接的状态。如果没有对端的响应或中断，将产生超时中断
0x40	RECV	RECV命令通过RX读指针寄存器(Sn_RX_RD)处理数据的接收。 详细的信息请参考5.2.1.1服务器模式接收处理，SOCKET n的RX接收数据长度寄存器(Sn_RX_RSR)，SOCKET n的RX写指针寄存器(Sn_RX_WR)，SOCKET n的RX读指针寄存器(Sn_RX_RD)

下面的命令只用于SOCKET 0和S0_MR(P3:P0)=S0_MR_PPPoE。*详细信息请参考“How to use ADSL”*

值	符号	描述Description
0x23	PCON	发送PPPoE搜寻数据包，开始ADSL连接
0x24	PDISCON	关闭PPPoE连接
0x25	PCR	在每一个阶段发送REQ信息
0x26	PCN	在每一个阶段发送NAK信息
0x27	PCJ	在每一个阶段发送REJECT信息

S*n*_IR (Socket*n* 中断寄存器) [R] [0x4002+0x0n00] [0x00]

Sn_IR寄存器提供Socket n的中断类型信息，如建立连接、断开连接、接收数据和超时。当SCOCKET产生中断且Sn_IMR相对应的屏蔽位为‘1’时，Sn_IR的中断位将会变成‘1’。

为了清除Sn_IR位，主机应该相应的位写‘1’。当所有Sn_IR的位被清零后，IR(n)将会自动清零。

7	6	5	4	3	2	1	0
PRECV	PFAIL	PNEXT	SEND_OK	TIMEOUT	RECV	DISCON	CON

位	符号	描述
7	PRECV	**Sn_IR(PRECV)中断** 只有在“SOCKET=0”且“S0_MR(P3:P0) = S0_MR_PPPoE”时有效。 当接收到一个不支持的选项时，产生PPP接收中断
6	PFAIL	**Sn_IR(PFAIL)中断** 只有在“SOCKET=0”且“S0_MR(P3:P0) = S0_MR_PPPoE”时有效。 当PAP认证失败时产生PPP失败中断
5	PNEXT	**Sn_IR(PNEXT)中断** 只有在“SOCKET=0”且“S0_MR(P3:P0) = S0_MR_PPPoE”时有效。 在ADSL连接过程中，当状态发生改变时产生下一个状态的中断
4	SEND_OK	**Sn_IR(SENDOK)中断** 当数据包发送完成时产生SEND OK中断
3	TIMEOUT	**Sn_IR(TIMEOUT)断** 当发生ARP超时或TCP超时，产生TIMEOUT中断
2	RECV	**Sn_IR(RECV)中断** 当接收到对端的数据时，产生接收数据中断
1	DISCON	**Sn_IR(DISCON) Interrupt中断** 当从对端接收到FIN/ACK数据包的FIN时，产生断开连接中断
0	CON	**Sn_IR(CON) Interrupt中断** 当连接建立时产生连接中断

S*n*_SR (Socket n状态寄存器) [R] [0x4003+0x0n00] [0x00]

该寄存器提供Socket n的状态。 当使用Sn_CR寄存器，或在传输/接收数据包时，Socket的状态将会更改。下表描述了不同Socket n的状态。

值	符号	描述
0x00	SOCK_CLOSED	这是SOCKET n的资源被释放的状态。不管以前的状态值是什么，当执行DISCON或CLOSE的命令，或产生ARP超时或TCP超时，它的数值都将会更改为 SOCK_CLOSED
0x13	SOCK_INIT	如果Sn_MR设置为TCP，且给Sn_CR写入OPEN命令，它将会改变为SOCK_INIT状态。这是建立TCP连接的第一步 这时，可以运行LISTEN命令将端口设置为TCP服务器模式，或运行CONNECT命令进入TCP客户端模式 这是 SOCKET n在TCP服务器模式下等待TCP客户端的连接请求的状态
0x14	SOCK_LISTEN	SOCKET n运行在TCP服务器模式下，等待从TCP客户端的连接请求 当执行了LISTEN命令，状态则改变为SOCK_LISTEN 当成功建立连接后，SOCKET n的状态将从 SOCK_LISTEN 改变为SOCK_ESTABLISHED。但是，如果连接失败，产生TCP超时(Sn_IR(TIME_OUT)=1)，且状态将转换为SOCK_CLOSED
0x17	SOCK_ESTABLISHED	它指示TCP的连接成功建立。在SOCK_LISTEN状态下，当来自TCP客户端的SYN数据包成功处理，或者是成功执行CONNECTS命令，它的状态都将会转变成SOCK_ESTABLISHED。在该状态下可以进行数据传送，也就是说，可以使用SEND或RECV命令
0x1C	SOCK_CLOSE_WAIT	这是当端口收到来自对端的断开连接请求的一个状态，TCP连接是半关闭的，它仍然可以传送数据。所以，如想完成整个TCP的断开连接，必须执行DISCON命令 没有经过断开连接的过程而关闭SOCKET n，那么应当执行CLOSE命令
0x22	SOCK_UDP	这是打开SOCKET n并设置为UDP模式的一个状态。当Sn_MR(P3:P0)设置为Sn_MR_UDP，且执行了OPEN命令时，端口的状态改变为SOCK_UDP。不同于TCP模式，在UDP模式下，不需要进行连接就可以进行数据传送
0x32	SOCK_IPRAW	端口工作在IPRAW模式下。当 Sn_MR(P3:P0)置Sn_MR_IPRAW且执行OPEN命令时，它的状态将改变为SOCK_IPRAW。与UDP模式相似，IP数据包可以在无连接的情况下传送
0x42	SOCK_MACRAW	如果S0_CR= OPEN 且S0_MR(P3:P0) = S0_MR_MACRAW，端口的状态将改变为 SOCK_MACRAW。 MACRAW数据包(以太网帧)的传送与UDP模式相似

0x5F	SOCK_PPPOE	这是SOCKET 0以PPPoE模式打开的一种状态。如果S0_CR=OPEN，且S0_MR (P3:P0) = S0_MR_PPPoE，状态将改变为 SOCK_PPPoE。这是一种暂时的状态，用于PPPoE的连接

以下是状态转换过程中的状态。

值	符号	描述
0x15	SOCK_SYNSENT	该状态表示一个连接请求(SYN数据包)发送到TCP服务器。它显示因执行CONNECT命令引起的状态由SOCK_INIT到SOCK_ESTABLISHED的一个转换过程。在此状态下，如果收到TCP服务器的连接授权(SYN/ACK数据包)，它会自动更改为 SOCK_ ESTBLISHED。如果在TCP超时(Sn_IR(超时)=1)出现之前，还没有收到TCP服务器发出的SYN/ACK数据包，它便会更改为 SOCK_CLOSED
0x16	SOCK_SYNRECV	此状态表示已收到一个从TCP客户端发出的连接请求(SYN数据包)。当W5200成功发出连接授权(SYN/ACK数据包) 到TCP客户端时，它就会自动更改为SOCK_ESTABLISHED。如果失败则产生TCP超时，并更改为SOCK_CLOSED
0x18	SOCK_FIN_WAIT	这些状况表示SOCKET n已关闭。它们出现在断连接过程中的“主动关闭”或“被动关闭”时期。当断开连接过程成功完成，或产生TCP超时(Sn_IR(TIMEOUT)=1)时，它便会改变为SOCK_CLOSED
0x1A	SOCK_CLOSING	
0X1B	SOCK_TIME_WAIT	
0X1D	SOCK_LAST_ACK	
0x01	SOCK_ARP	此状态表示已经发送ARP请求，以获取目的硬件地址。在SOCK_UDP或SOCK_IPRAW模式下执行SEND命令，或在SOCK_INIT状态时执行CONNECT命令时 将显现这种状态 如果成功获取目的硬件地址(收到ARP响应)，它便会更改为SOCK_UDP、SOCK_IPRAW，或SOCK_SYNSENT。如果失败，将产生ARP超时。如发生在 UDP或IPRAW模式，它便会返回到以前的状态(SOCK_UDP或SOCK_IPRAW)。如发生在TCP模式，它便会转到 SOCK_CLOSED状态 对比 - 在SOCK_UDP或SOCK_IPRAW模式下，当先前Sn_DIPR和当前的Sn_DIPR的值不同时，便会启动ARP过程。如果先前的和当前的Sn_DIPR值相同，将不回启动ARP，因为目标硬件的地址已经获得了

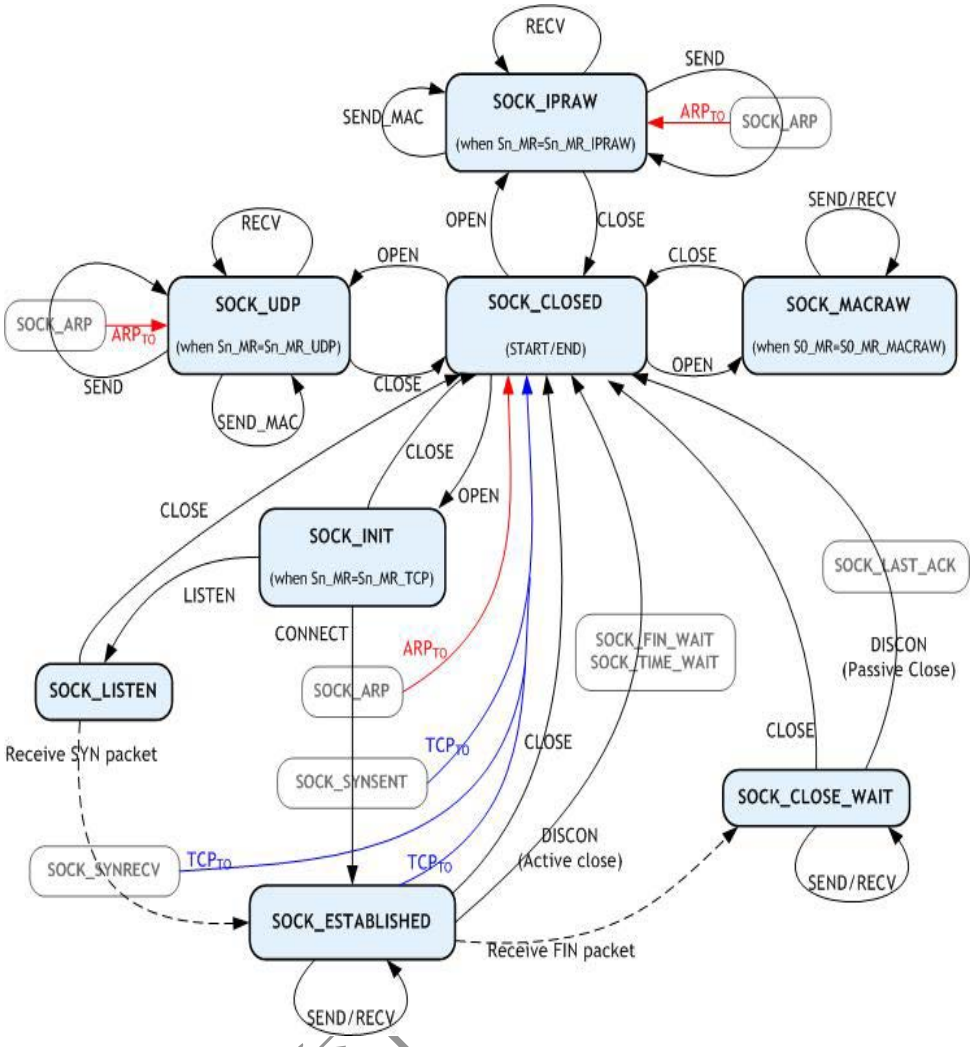


图4 Socket状态改变

S*n*_PORT (Socket n本机端口号寄存器) [R/W] [0x4004+0x0n00-0x4005+0x0n00] [0x0000]

在TCP和UDP模式下，该寄存器用于设置源端口的端口号，必须在OPEN命令之前设置好该寄存器。

例：Socket 0的端口号= 5000(0x1388)，配置如下：

0x4004	0x4005
0x13	0x88

Sn_DHAR (Socket n目的硬件地址寄存器) [R/W] [0x4006+0x0n00-0x400B+0x0n00] [0xFFFFFFFFFFFF]

该寄存器为SOCKET n设置目的硬件地址。如果SOCKET0用于PPPoE连接，S0_DHAR设置为一个已知的PPoE服务器的硬件地址。

在UDP和IPRAW模式下使用SEND_MAC命令时，它为Socket n设定目的硬件地址。而在TCP、UDP和IPRAW模式时，Sn_DHAR会根据定从CONNECT或SEND命令中的ARP过程获取的目的硬件地址设定。当成功执行CONNECT或SEND命令后，主机可以通过Sn_DHAR取得目的硬件地址。

当使用W5200的PPPoE过程时，PPPoE服务器的硬件地址不需要进行设置。然而，即使

W5200不需要使用PPPoE过程，仍然需要自己实现MACRAW模式。如想要发送或接收PPPoE的数据包，那PPPoE服务器硬件地址(从PPPoE程序中取得)、PPPoE服务器的IP地址、 和PPP session ID就必须设置。此外，MR(PPPoE) 也应设置为‘1’。

S0_DHAR在OPEN命令之前设置 PPPoE服务器硬件地址。 PPPoE服务器硬件地址是由S0_DHAR执行OPEN命令后，在 PDHAR里应用。而已经配置的PPPoE信息，即使是在CLOSE命令后，对于内部运作依然是有效的。

例：Socket 0 目的硬件地址= 08.DC.00.01.02.10，配置如下：

0x4006	0x4007	0x4008	0x4009	0x400A	0x400B
0x08	0xDC	0x00	0x01	0x02	0x0A

S*n*_DIPR (Socket n目的IP地址寄存器) R/W] [0x400C+0x0n000x400F+0x0n00] [0x00000000]

它设置SOCKET n的目的IP地址。如果Socket0用于PPPoE模式，那么S0_DIPR会设置为 已知的PPPoE服务器的IP地址。它只有在TCP、UDP、IPRAW或PPPoE模式下才有效，在MACRAW模式下被忽略。在TCP模式下，当作为TCP客户端的时候，在执行CONNECT命令之前，将该寄存器设置为TCP服务器的IP地址。而当作为TCP服务器的时候，它将在成功建立连接之后，该寄存器会在内部自动设置为TCP客户端的IP地址。

在UDP或IPRAW模式下，Sn_DIPR会设置为目标IP地址，它会在执行SEND或SEND_MAC命令之前，用作传输UDP或IPRAW数据包。

例：Socket 0的目的IP地址= 192.168.0.11，配置如下：

0x400C	0x040D	0x400E	0x040F
192 (0xC0)	168 (0xA8)	0 (0x00)	11 (0x0B)

S*n*_DPORT (Socket n目的端口号寄存器)[R/W][0x4010+0x0n00-0x4011+0x0n00] [0x00]

Socket n的目标端口号在Sn_DPORT设置。如果Socket0已用于PPPoE模式，那S0_DPORT0会为 PPP session ID设置一个已知的号码。它只有在TCP、UDP、或PPPoE模式下才有效，而在其他模式下会被忽略。

在TCP模式下，当作为TCP客户端时，它在运行CONNECT命令之前侦听TCP服务器的端口号。

在UDP模式下，在执行SEND或SEND_MAC命令之前， 在Sn_DPORT中设置目标端口号，用作于传输 UDP数据包。

在PPPoE模式，S0_DPORT设置为已知的PPP session ID。运行OPEN命令之后，PPP session ID(由S0_DPORT0设置)在PSIDR里应用。

例：Socket 0目的端口号为= 5000(0x1388)，配置如下：

0x4010	0x4011
0x13	0x88

S*n*_MSS (Socket n最大分片大小寄存器) [R/W][0x4012+0x0n00-0x4013+0x0n00] [0x0000]

该寄存器用于TCP的MSS (Maximum Segment Size) 。当TCP是在被动模式下启动，该寄存器会显示MSS的设置。它只是支持TCP或UDP模式。当使用PPPoE(MR(PPPoE)=1)的时候，TCP或UDP模式的MTU将被分配在PPPoE的MTU范围。

模式	**通常** (MR(PPPoE)=0)		**PPPoE** (MR(PPPoE)=1)	
	默认MTU	范围	默认MTU	范围
TCP	1460	1 ~ 1460	1452	1 ~ 1452
UDP	1472	1 ~ 1472	1464	1 ~ 1464
IPRAW	1480		1472	
MACRAW	1514			

在IPRAW或MACRAW模式下，MTU不在内部处理的，但会使用默认的MTU值。因此，当传输的数据长度大于默认的MTU时，主机应该以手动方式将数据长度划分成默认MTU单位

在TCP或UDP模式下，如果传输的数据长度大于MTU时，W5200会自动将数据划分成MTU的单位。在TCP模式下，MTU被称之为MSS。在主机写入的数值和对端的MSS数值当中，MSS会在TCP的连接过程中选择较小的值。

例：Socket 0的MSS = 1460(0x05B4)，配置如下：

0x4012	0x4013
0x05	0xB4

S*n*_PROTO (Socket n的IP协议寄存器) [R/W] [0x4014+0x0n00] [0x00]

这是一个单字节的寄存器。它用于设置IP层里IP字头中的协议号字段。它只有在 IPRAW模式下有效， 在其他模式下会被忽略。Sn_PROTO在执行OPEN命令之前设置。当Socket n在IPRAW模式下打开， 它会发送和接收在Sn_PROTO中设置的协议号数据。Sn_PROTO可以在0x00~0xFF的范围内分配，但W5200不支持TCP(0x06)和UDP(0x11)的协议号。

协议号由IANA(Internet assigned numbers authority)定义，详细信息请参考：

http://www.iana.org/assignments/protocol-numbersU

例：因特网控制信息协议(ICMP) = 0x01，因特网分组管理协议(IGMP)= 0x02

S*n*_TOS (Socket n的IP服务类型寄存器) [R/W] [0x4015+0x0n00] [0x00]

该寄存器设置IP层的IP头中的TOS(服务类型) 字段。它必须在执行OPEN命令之前设置。请参考：

http://www.iana.org/assignments/ip-parametersU

S*n*_TTL (Socket n的IP生存时间寄存器) [R/W] [0x4016+0x0n00] [0x80]

该寄存器设置在IP层的IP头中的TTL(生存时间) 字段。它必须在执行OPEN命令之前设置。请参考：

http://www.iana.org/assignments/ip-parametersU

S*n*_RXMEM_SIZE (Socket n 接收数据存储器大小寄存器) [R/W] [0x401E+0x0n00] [0x02]

该寄存器为每个SOCKET配置接收数据存储器的大小。每个SOCKET的接收数据存储器大小是可以配置为 1、2、4、8和16K字节。 系统复位(Reset)后，默认分配为2K字节。 每个SOCKET的Sn_RXMEM_SIZE的总和(Sn_RXMEM_SIZESUM) 应是16K字节。

值	0x01	0x02	0x04	0x08	0x0F
存储器大小	1KB	2KB	4KB	8KB	16KB

例1) SOCKET 0 : 8KB, SOCKET 1 : 2KB

0xFE401E	0xFE411E
0x08	0x02

例2) SOCKET 2 : 1KB, SOCKET 3 : 1KB

0xFE421E	0xFE431E
0x01	0x01

例3) SOCKET 4 : 1KB, SOCKET 5 : 1KB

0xFE441E	0xFE451E
0x01	0x01

例4) SOCKET 6 : 1KB, SOCKET 7 : 1KB

0xFE461E	0xFE471E
0x01	0x01

S*n*_TXMEM_SIZE (Socket n发送数据存储器大小寄存器) [R/W] [0x401E+0x0n00] [0x02]

该寄存器为每个SOCKET配置发送数据存储器的大小。每个SOCKET的发送数据存储器的大小可以配置为1、2、4、8和16K字节。 系统复位(Reset)后 ， 默认分配为2K字节。 每个SOCKET的Sn_TXMEM_SIZE的总和(Sn_TXMEM_SIZESUM) 应是16K字节。

例1) SOCKET 0 : 4KB, SOCKET 1 : 1KB

0xFE401F	0xFE411F
0x04	0x01

例2) SOCKET 2 : 2KB, SOCKET 3 : 1KB

0xFE421F	0xFE431F
0x02	0x01

例3) SOCKET 4 : 2KB, SOCKET 5 : 2KB

0xFE441F	0xFE451F
0x02	0x02

例4) SOCKET 6 : 2KB, SOCKET 7 : 2KB

0xFE461F	0xFE471F
0x02	0x02

S*n*_TX_FSR (Socket n的TX存储器剩余空间大小寄存器) [R] [0x4020+0x0n00-0x4021+0x0n00] [0x0800]

该寄存器提供Socket n发送数据存储器的可用空间大小(可供发送数据的字节数)。请注意，主机不能写入大于Sn_TX_FSR的数据。因此，在发送数据之前，一定检查 Sn_TX_FSR。如果你发送的数据长度小于或等于 Sn_TX_FSR时，在写入数据后，可执行SEND或SEND_MAC命令来发送数据。

在TCP模式下，当对端检查已发送的数据包(如果收到对端的DATA/ ACK数据包)时，Sn_TX_FSR会自动增加已发送的数据包大小。在其它模式下，当Sn_IR(SENDOK)为‘1’，Sn_TX_FSR也会自动增加已发送的数据包大小。当检查该寄存器时，用户应先读取高字节(0x4020，0x4120，0x4220，0x4320，0x4420，0x4520，0x4620，0x4720) ，然后读取低字节(0x4021，0x4121，0x4221，0x4321，0x4421，0x4521，0x4621，0x4721)，这样 才会得到正确的数值。

例： S0_TX_FSR=2048(0x0800)，

0x4020	0x4021
0x08	0x00

Sn_TX_RD (Socket n的TX存储器读指针寄存器) [R] [0x4022+0x0n00-0x4023+0x0n00] [0x0000]

该寄存器显示发送数据存储器最后一个传输的地址值。当Socket n的命令寄存器执行SEND命令时，把从当前Sn_TX_RD指针地址到Sn_TX_WR指针地址的数据发送出去，并在发送完成后自动更新Sn_TX_RD的值。因此，在发送完成后，Sn_TX_RD和Sn_TX_WR的值相同。

当读取该寄存器时，用户应先读取高字节 (0x4022, 0x4122, 0x4222, 0x4322, 0x4422, 0x4522,0x4622, 0x4722)，然后读取低字节 (0x4023, 0x4123, 0x4223, 0x4323, 0x4423, 0x4523, 0x4623, 0x4723) ，这样才会得到正确的值。

S*n*_TX_WR (Socket n的TX存储器写指针寄存器) [R/W] [0x4024+0x0n00-0x4025+0x0n00] [0x0000]

该寄存器提供地址指针信息，指示当前写入要发送的数据的位置。当读取该寄存器时，用户应先读取高字节 (0x4024,0x4124, 0x4224, 0x4324, 0x4424, 0x4524, 0x4624, 0x4724)，然后读取低字节 (0x4025, 0x4125, 0x4225, 0x4325, 0x4425, 0x4525, 0x4625, 0x4725) ，这样才会得到正确的数值。

注意：此寄存器的值会在Sn_CR寄存器成功执行SEND命令后被改变。

例：S0_TX_WR=2048(0x0800)，

0x4024	0x4025
0x08	0x00

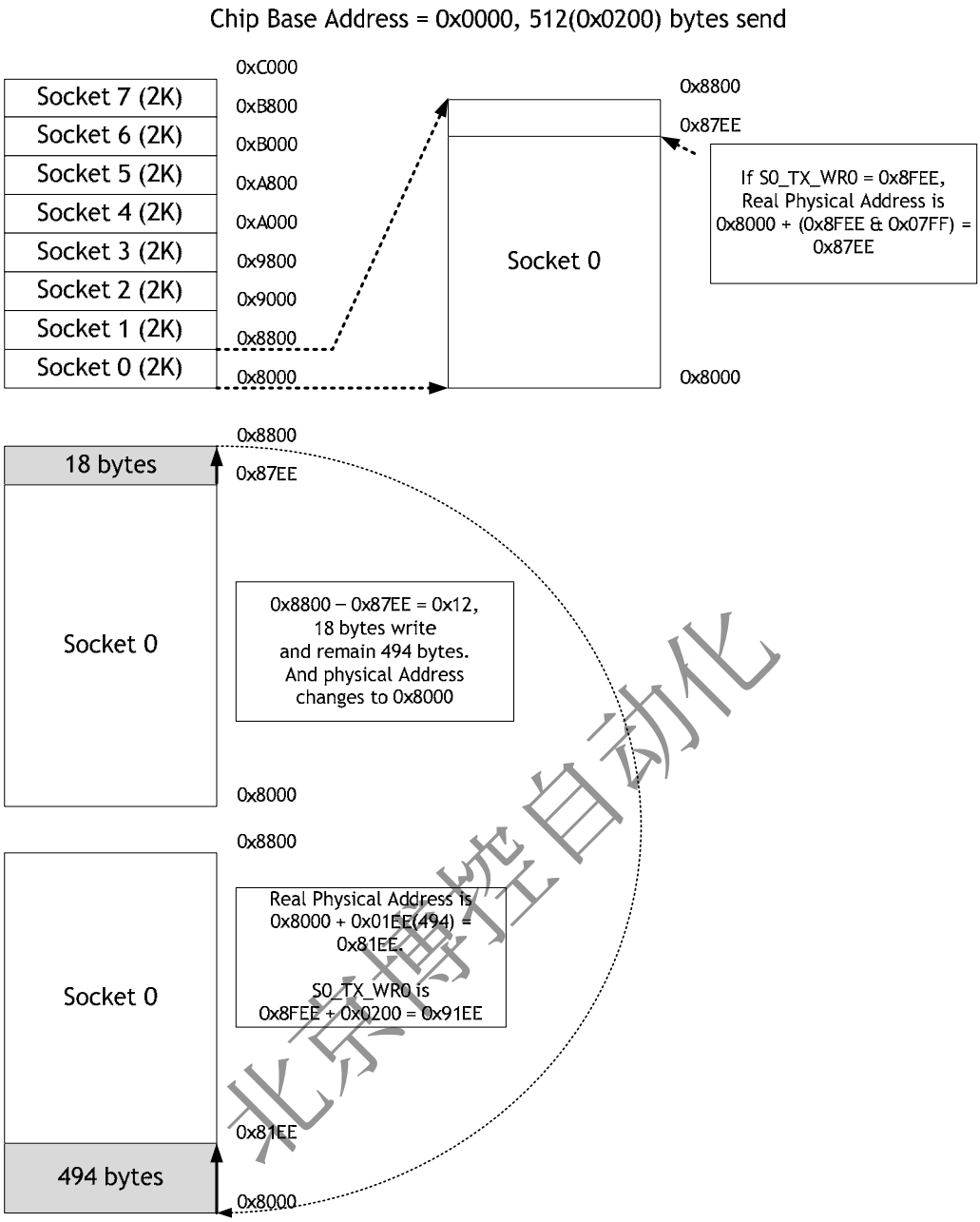


图5 实际物理地址的计算

但这个数值并不是实际的物理地址，实际的物理地址需要经过以下的计算：

1. Socket n发送存储器的基地址（这里称之为gSn_TX_BASE）和掩码地址值（这里称之为gSN_TX_MASK）可以通过TMSR的值计算获得。
2. 将Sn_TX_WR和个Sn_TX_MASK这两个值“位与”，得到该端口TX存储器区域内的偏移地址（这里称之为get_offset）。
3. 将get_offset和gSn_TX_BASE相加，得到实际访问的物理地址（这里称之为get_start_address）。

现在，可以将需要发送的数据写入到get_start_address地址。(有一种情况，如在写入数据时地址超过了Socket发送存储器的上界，这时，先把数据写入到上界地址，然后把物理地址转到

gSn_TX_BASE，然后再写入其余的数据)。 之后，切记要把Sn_TX_WR的值增加到与写入数据的大小相同。最后，给Sn_CR (Socket n命令寄存器)一个SEND命令。

如果有需要，请参考TCP服务器模式下传输部分的伪代码

Sn_RX_RSR (RX接收数据的字节长度寄存器) [R] [0x4026+0x0n00-0x4027+0x0n00] [0x0000]

该寄存器向用户提供在Socket n-接收存储器中已接收数据的字节大小。由于这个值是通过Sn_RX_RD和Sn_RX_WR内部计算获得的，它会因Sn_CR (Socket n命令寄存器)中的RECV命令和从远程节点接收数据而自动改变。

当读取该寄存器时，用户应先读取高字节*(0x4026, 0x4126, 0x4226, 0x4326, 0x4426, 0x4526, 0x4626, 0x4726)*，然后读取低字节*(0x4027, 0x4127, 0x4227, 0x4327, 0x4427, 0x4527, 0x4627, 0x4727)，这样才*会得到正确的数值。

例：S0_RX_RSR,中的值为2048（0x0800），

0x4026	0x04027
0x08	0x00

根据RX存储器的大小决定了该寄存器的总的长度值。

S*n*_RX_RD (Socket n的RX存储器读指针寄存器) [R/W] [0x4028+0x0n00-0x4028+0x0n00] [0x0000]

该寄存器提供读取接收数据的位置信息。当读取该寄存器时，用户应先读取高字节 *(0x4028, 0x4128, 0x4228, 0x4328, 0x4428, 0x4528, 0x4628, 0x4728)*，然后再读取低字节*(0x4029, 0x4129, 0x4229, 0x4329, 0x4429, 0x4529, 0x4629, 0x4729) ，这样才*会得到正确的数值。

注意：Sn_CR执行了SEND命令后，该寄存器的值将改变。

例：S0_RX_RD的值为2048(0x0800)

0x4028	0x4029
0x08	0x00

但该值本身并不是要读取数据的实际物理地址。实际的物理地址计算如下：

1. Socket n接收存储器(RX)的基地址(此后称之为gSn_RX_BASE) 和Socket n的RX屏蔽地址(此后称之为gSn_RX_MASK)将会以RMSR数值计算。请参考的5.1初始化的psedo代码。
2. 将Sn_RX_RD和gSn_RX_MASK进行“位与”运算(bitwise-AND operation)，会得到Socket接收存储器地址范围的偏移地址(此后会称为 get_offset)。
3. 将get_offset和gSn_RX_BASE的数值相加，得到实际访问的物理地址(会之称为get_start_address)。

Sn_RX_WR (Socket n的RX存储器写指针寄存器)[R/W][(0xFE402A + 0xn00) – (0xFE402B + 0xn00)][0x0000]

该寄存器提供内部存储器中写入接收数据的位置信息。当读取该寄存器时，用户应先读取高字节*(0x402A, 0x412A, 0x422A, 0x432A, 0x442A, 0x452A, 0x462A, 0x472A)*，然后读取低字节*(0x402B, 0x412B,0x422B, 0x432B, 0x442B, 0x452B, 0x462B, 0x472B)，这样才*会得到正确的数

值。

例：S0_RX_WR,中的值为2048(0x0800)

0x402A	0x402B
0x08	0x00

Sn_IMR (Socket n中断屏蔽寄存器)[R/W][0x402C+0x0n00][0xFF]

该寄存器配置Socket n的中断，并通知到主机。Sn_IMR的中断屏蔽位跟Sn_IR的中断位是对应的。如果中断发生在任何Socket时，且该位设置为‘1’，那么与之对应的Sn_IR位也会设置为‘1’。当Sn_IMR和Sn_IR的位同时为‘1’时，IR(n)将成为‘1’。这时，如果IMR(n) 是‘1’，W5200将向主机发出中断请求 (nINT引脚信号为低电平)。

7	6	5	4	3	2	1	0
PRECV	PFAIL	PNEXT	SEND_OK	TIMEOUT	RECV	DISCON	CON

Bit	Symbol	Description
7	PRECV	Sn_IR(PRECV)中断屏蔽 只有在“SOCKET = 0”和“S0_MR(P3:P0) = S0_MR_PPPoE”时才有效
6	PFAIL	Sn_IR(PFAIL)中断屏蔽 只有在“SOCKET = 0”和“S0_MR(P3:P0) = S0_MR_PPPoE”才有效
5	PNEXT	Sn_IR(PNEXT)中断屏蔽 只有在“SOCKET = 0”和“S0_MR(P3:P0) = S0_MR_PPPoE”才有效
4	SENDOK	Sn_IR(SENDOK)中断屏蔽
3	TIMEOUT	Sn_IR(TIMEOUT)中断屏蔽
2	RECV	Sn_IR(RECV)中断屏蔽
1	DISCON	Sn_IR(DISCON)中断屏蔽
0	CON	Sn_IR(CON)中断屏蔽

Sn_FRAG(Socket n分片寄存器)[R/W][0x402D+0x0n00-0x402E+ 0x0n100][0x4000]

该寄存器设置在IP层上的IP头的分片字段的值。W5200并不支持在IP层的分片数据包。尽管Sn_FRAG已配置，但IP数据并不会分片，同时也不建议。该寄存器它应该在执行OPEN命令之前设置。

例：Sn_FRAG0 = 0x4000（不分片）

0x402D	0x402E
0x40	0x00

5 功能描述

通过设置W5200的一些寄存器可以建立Internet的连接。本章将讲述这些操作。

5.1 初始化

基本设置

正确选择和设置下面的寄存器操作W5200

1. 模式寄存器 (MR)
2. 中断屏蔽寄存器(IMR)
3. 重发时间寄存器(RTR)
4. 重发次数寄存器(RCR)

关于上面寄存器的详细信息，请参考“ 寄存器描述 ”章节。

设置网络信息

为了建立通信，需要设置基本网络信息。

① 本机硬件地址寄存器

由SHAR所设置的本机硬件地址在以太网 MAC层里被规定，会使用唯一的硬件地址 (以太网MAC地址)。IEEE负责管理MAC地址的分配。而生产网络设备的制造商会替其产品分配MAC地址

关于MAC地址分配的详细请参考下面的网页。

http://www.ieee.org/, http://standards.ieee.org/regauth/oui/index.shtml

② GAR(网关地址寄存器)

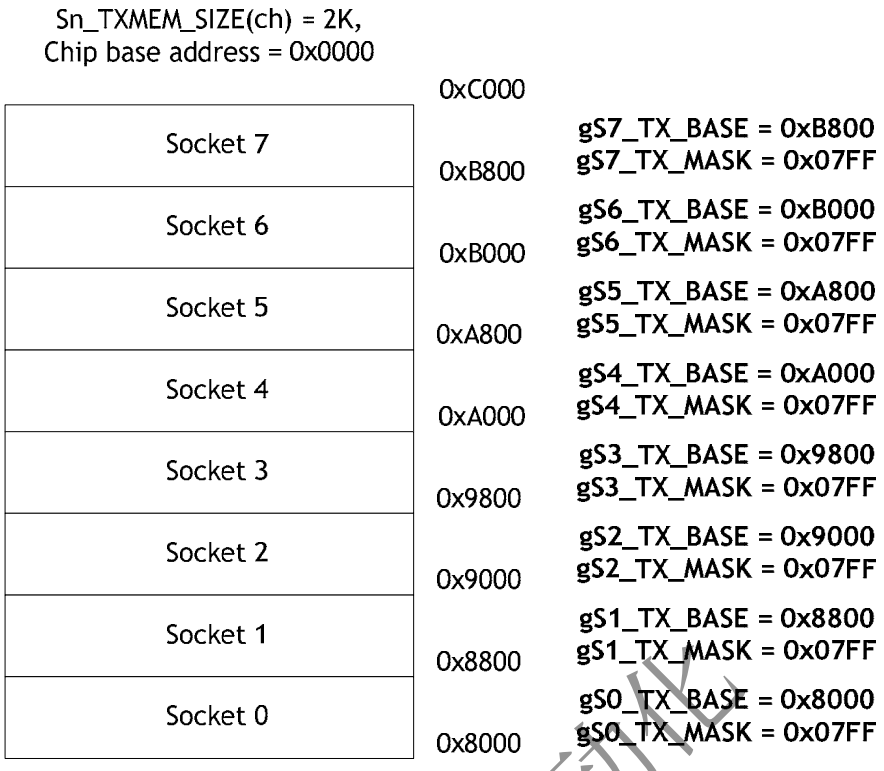
③ SUBR(子网掩码寄存器)

④ SIPR(本机IP地址寄存器)

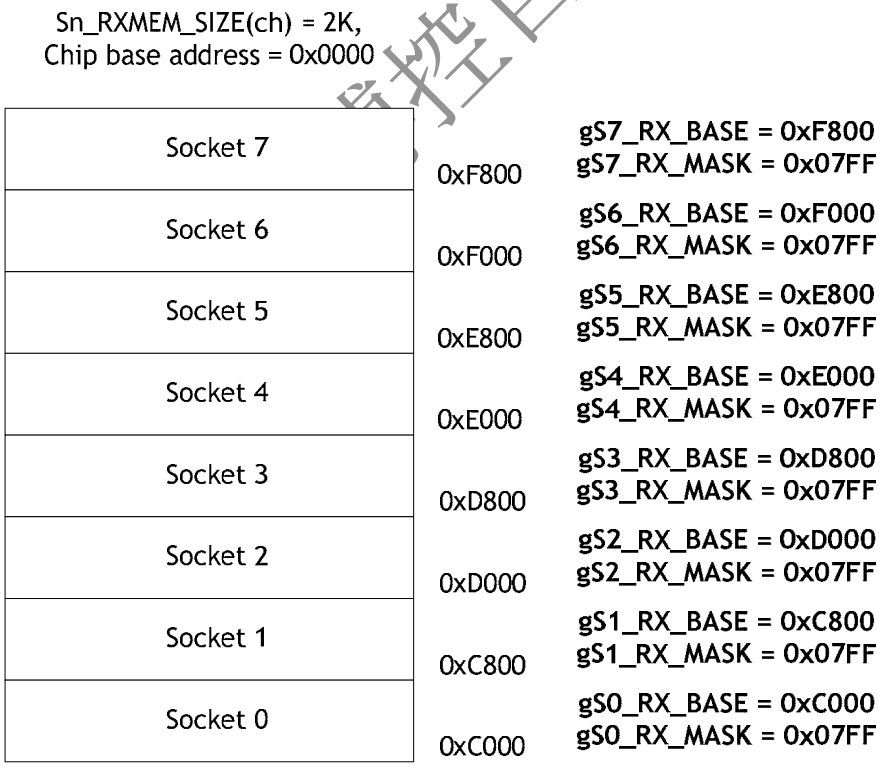
设置SOCKET存储器信息

这一步设置端口TX/RX存储器的信息。每个端口的基地址和地址掩码都在这一步确定并保存。

```
In case of, assign 2KB rx, tx memory per SOCKET
{
gS0_RX_BASE = 0x0000(Chip base address) + 0xC000(Internal RX buffer address); // Set base address of RX memory for SOCKET 0
Sn_RXMEM_SIZE(ch) = (uint8 *) 2;    // Assign 2K rx memory per SOCKET
gS0_RX_MASK = 2K – 1; // 0x07FF, for getting offset address within assigned SOCKET 0 RX memory
gS1_RX_BASE = gS0_RX_BASE + (gS0_RX_MASK + 1);
gS1_RX_MASK = 2K – 1;
gS2_RX_BASE = gS1_RX_BASE + (gS1_RX_MASK + 1);
gS2_RX_MASK = 2K – 1;
gS3_RX_BASE = gS2_RX_BASE + (gS2_RX_MASK + 1);
gS3_RX_MASK = 2K – 1;
gS4_RX_BASE = gS3_RX_BASE + (gS3_RX_MASK + 1);
gS4_RX_MASK = 2K – 1;
gS5_RX_BASE = gS4_RX_BASE + (gS4_RX_MASK + 1);
gS5_RX_MASK = 2K – 1;
gS6_RX_BASE = gS5_RX_BASE + (gS5_RX_MASK + 1);
gS6_RX_MASK = 2K – 1;
gS7_RX_BASE = gS6_RX_BASE + (gS6_RX_MASK + 1);
gS7_RX_MASK = 2K – 1;
gS0_TX_BASE = 0x0000(Chip base address) + 0x8000(InternalTX buffer address); // Set base address of TX memory for SOCKET 0
Sn_TXMEM_SIZE(ch) = (uint8 *) 2;    // Assign 2K rx memory per SOCKET
gS0_TX_MASK = 2K – 1;
/* Same method, set gS1_TX_BASE, gS1_TX_MASK, gS2_TX_BASE, gS2_TX_MASK, gS3_TX_BASE, gS3_TX_MASK, gS4_TX_BASE, gS4_TX_MASK, gS5_TX_BASE, gS5_TX_MASK, gS6_TX_BASE, gS6_tx_MASK, gS7_TX_BASE, gS7_TX_MASK */
}
```



(a) TX memory



(b) RX memory

图6 Socket n的内部TX/RX存储器定位

5.2 数据通信

初始化过程结束之后，W5200可以以TCP、UDP、IPRAW和MACRAW的模式打开SOCKET，进行 数据发送和接收。W5200支持8个端口独立且同时工作。在本节将介绍每一种模式的通信方法。

5.2.1 TCP

TCP是一种面向连接的通信协议。TCP使用本机IP地址、端口号和目的IP地址、目的端口号等建立端口连接。使用该端口发送和接收数据。

建立端口连接的方法有“ TCP服务器 ”和“ TCP客户端 ”， 它们的区别是发送连接请求（SYN数据包）。

“ TCP服务器 ”侦听来自“ TCP客户端 ”的连接请求，接收发送来的连接请求并建立端口连接（被动打开连接）。

“ TCP客户端 ”发送连接请求到“ TCP服务器 ”以建立连接（主动打开连接）。

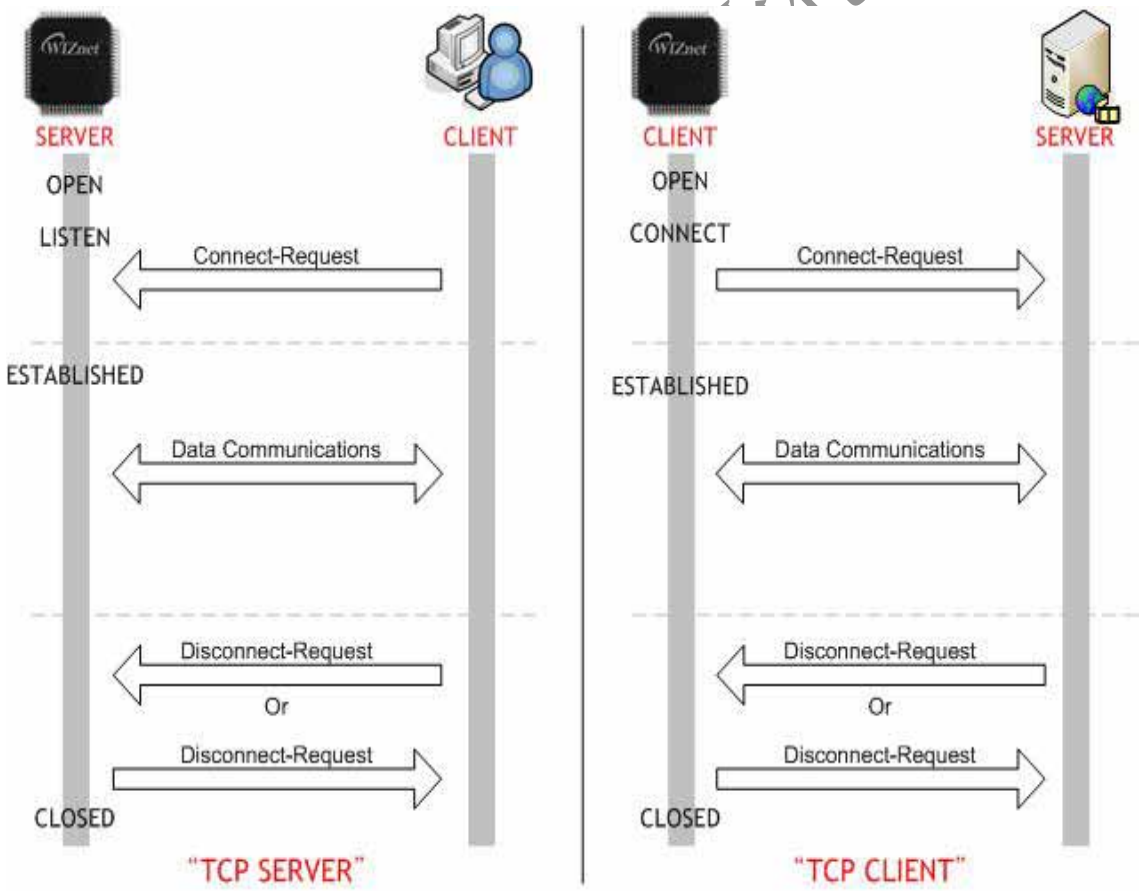


图7 TCP服务器和TCP客户端

5.2.1.1 TCP服务器

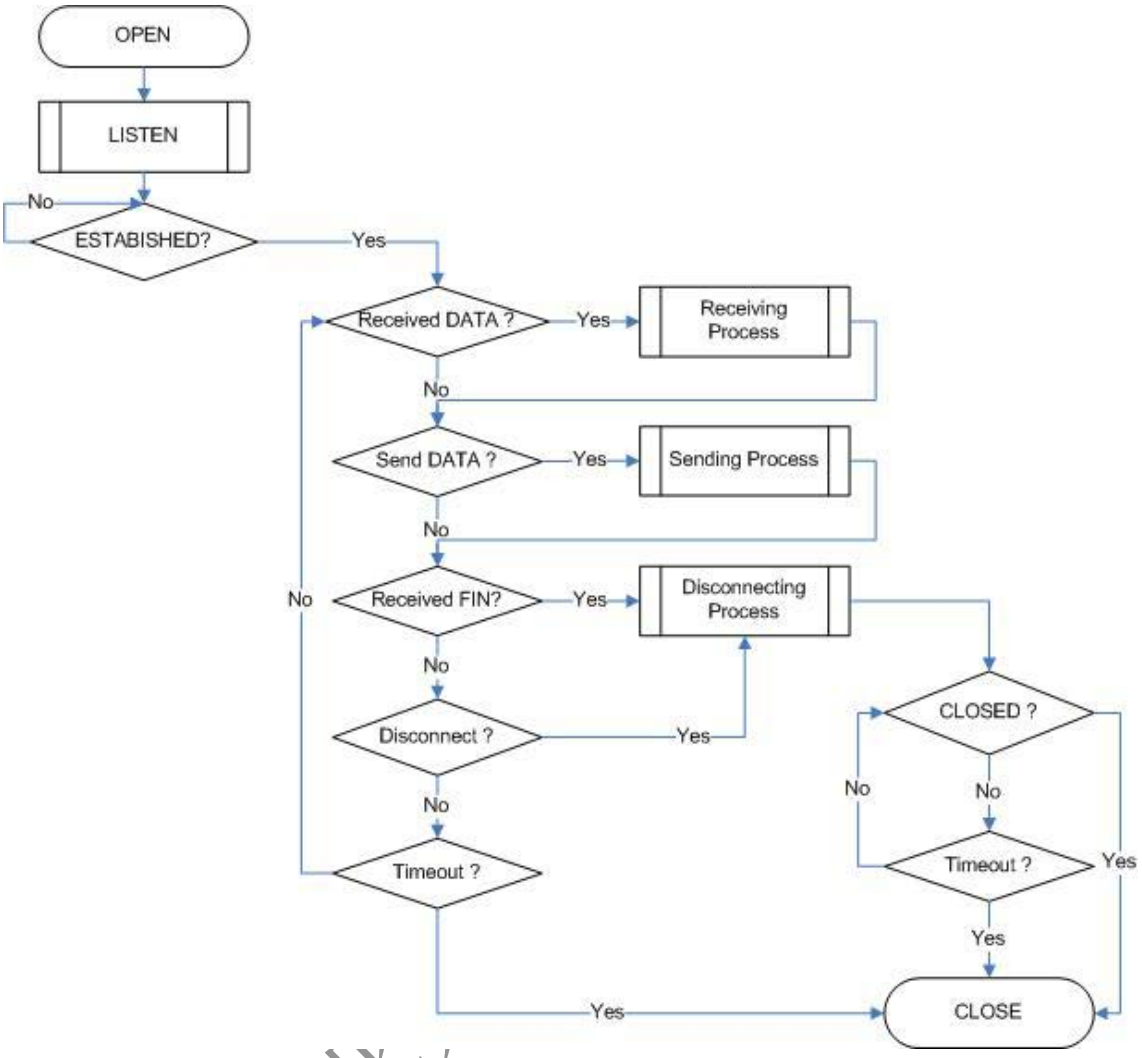


图8 TCP服务器操作流程

端口初始化

TCP数据通信必需要先进行的端口初始化。初始化就是打开端口。端口初始化的过程是：选择W5200的8个端口中的一个端口，并设置该端口的协议模式(Sn_MR)和本机端口号(Sn_PORT – 在“ TCP服务器 ”中的侦听端口号)，然后执行OPEN命令。执行命令之后，如果Sn_SR的状态更改为 SOCK_INIT，表示端口初始化过程已完成。

初始化端口的过程在“ TCP服务器 ”和“ TCP客户端 ”模式下都是相同的。初始化端口的过程如下：

```
{
START:
Sn_MR = 0x01;              // sets TCP mode
Sn_PORT0 = source_port;          // sets source port number
Sn_CR = OPEN;                          // sets OPEN command
/* wait until Sn_SR is changed to SOCK_INIT */
if (Sn_SR != SOCK_INIT) Sn_CR = CLOSE; goto START;
}
```

侦听

执行LISTEN命令进入TCP服务器模式

```
{
/* listen SOCKET */
Sn_CR = LISTEN;
/* wait until   Sn_SR is changed to SOCK_LISTEN */
 if (Sn_SR != SOCK_LISTEN) Sn_CR = CLOSE; goto START;
}
```

建立连接

当端口的状态寄存器Sn_SR为SOCK_LISTEN时，如果收到SYN数据包，Sn_SR的将改变为SOCK_SYNRECV状态，并发送SYN/ACK数据包。至此，端口建立起连接。端口建立连接后才允许进行数据通信。有两种方法确认端口的连接。

```
方法一：
{
 if (Sn_IR(CON) == '1')   Sn_IR(CON) = '1'; goto ESTABLISHED stage;
 /* In this case, if the interrupt of Socket n-th is activated, interrupt occurs. Refer to IR, IMR Sn_IMR
    and Sn_IR. */
}
```

```
方法二：
{
 if (Sn_SR == SOCK_ESTABLISHED) goto ESTABLISHED stage;
}
```

建立连接：检查接收的数据

确定接收到TCP数据。

```
方法一：
{
 if (Sn_IR(RECV) == ‘1’) Sn_IR(RECV) = ‘1’; goto Receiving Process stage;
 /* In this case, if the interrupt of Socket n-th is activated, interrupt occurs. Refer to IR, IMR
    Sn_IMR and Sn_IR. */
}
```

```
方法二：
{
    if (Sn_RX_RSR0 != 0x0000) goto Receiving Process stage;
}
```

方法一：每当收到一个数据包， 把Sn_IR(RECV)设置为‘1’。如果主机没有把Sn_IR(RECV)设置为‘1’（清除Sn_IR(RECV)）就接收到下一个数据包时，它便不能识别下一个数据包的Sn_IR(RECV)。这是由于前一个的Sn_IR(RECV与下一个的Sn_IR(RECV)重迭。如果主机不能完全处理每个Sn_IR(RECV)的数据包，便不建议使用这个方法。

建立连接：接收处理

在这个过程中， 它处理已接收在内部接收存储器中的TCP数据。在TCP模式下， 如接收数据的长度大于端口的接收内存空间剩余大小，W5200便无法接收数据。如果发生了前面所说的情况，W5200便会暂停连接，并等待直到接收存储器空间大小大于接收数据的大小。

```
{
/* first, get the received size */
 len = Sn_RX_RSR;       // len is received size
 /* calculate offset address */
 src_mask = Sn_RX_RD&gSn_RX_MASK;       // src_mask is offset address
 /* calculate start address(physical address) */
 src_ptr = gSn_RX_BASE + src_mask;       // src_ptr is physical start address

 /* if overflow SOCKET RX memory */
 If((src_mask + len) > (gSn_RX_MASK + 1))
 {
    /* copy upper_size bytes of source_ptr to destination_address */
    upper_size = (gSn_RX_MASK + 1) – src_mask;
    memcpy(src_ptr,dst_ptr, upper_size);
    /* update destination_ptr */
    dst_address += upper_size;
    /* copy left_size bytes of gSn_RX_BASE to destination_address */
```

```
    left_size = len – upper_size;
    memcpy(gSn_RX_BASE,dst_address, left_size);
  }
  else
  {
    copy len bytes of source_ptr to destination_address*/
    memcpy(src_ptr, dst_ptr, len);
  }
  /* increase Sn_RX_RD as length of len */
  Sn_RX_RD += len;
  /* set RECV command */
  Sn_CR = RECV;
}
```

建立连接：检查发送的数据/发送处理

发送数据的大小不能大于指定的SOCKET n的发送存储器的大小。如果发送数据的大小大于设置的MSS， 他将拆分成MSS大小的长度然后再发送。要发送下一包数据，用户必须检查上一包数据的SEND命令执行完成。如果前面一个SEND命令没有完成而又执行下一个数据包的SEND命令，将产生错误的结果。发送的数据包越大，完成SEND命令所需要的时间就越长。因此用户可以适当地将数据包分片，然后再发送。

为了检查SEND命令执行完成，需要检查发送数据的长度是否与实际发送的数据长度相等。实际发送数据的长度是Sn_TX_TD寄存器在SEND命令执行前和执行后的差计算所得的值。如果实际发送数据的长度小于要发送数据的长度，SEND命令将试着发送剩余的数据。当实际发送的数据长度等于发送数据的长度时，发送的过程才将SEND命令执行完毕。一个简单的发送处理过程如下：

例：发送数据长度=10

- 带有发送数据长度信息，执行SEND命令，
- 计算实际发送数据的长度，
- 如果实际发送的长度为7（SEND命令前的Sn_TX_RD减去SEND命令后的Sn_TX_RD），还余3，
- 再执行SEND命令直到实际发送数据的长度与发送数据长度相同。

注意：在数据没有发送完毕时，不要向发送数据缓冲区写入数据。

```
{
  /* first, get the free TX memory size */
FREESIZE:
  freesize = Sn_TX_FSR;
  if (freesize<len) goto FREESIZE;     // len is send size

  /* calculate offset address */
```

```
dst_mask= Sn_TX_WR0 &gSn_TX_MASK;       // dst_mask is offset address
/* calculate start address(physical address) */
dst_ptr = gSn_TX_BASE + dst_mask;       // destination_address is physical start address
/* if overflow SOCKETTX memory */
if ( (dst_mask + len) > (gSn_TX_MASK + 1) )
{
    /* copy upper_size bytes of source_addr to destination_address*/
    upper_size = (gSn_TX_MASK + 1) – dst_mask;
    memcpy(src_addr, dst_ptr, upper_size);
    /* update source_addr*/
    source_addr += upper_size;
    /* copy left_size bytes of source_addr to gSn_TX_BASE */
    left_size = len – upper_size;
    mememcpy(source_addr, gSn_TX_BASE, left_size);
}
else
{
    /* copy len bytes of source_addr to destination_address */
    memcpy(source_addr, dst_ptr, len);
}
/* increase Sn_TX_WR as length of len */
 Sn_TX_WR0 += send_size;
/* set SEND command */
Sn_CR = SEND;
/* return real packet size */
return ( read_ptr_after_send - read_ptr_befor_send )
/* if return value is not equal len (len is send size),
 retry send left data without copying data*/
}
```

建立连接：检查断开连接的请求

检查是否收到断开连接的请求（FIN数据包）。用户可以如下确认FIN数据包。

```
方法一：
{
 if (Sn_IR(DISCON) == ‘1’) Sn_IR(DISCON)=‘1’; goto CLOSED stage;
 /* In this case, if the interrupt of Socket n-th is activated, interrupt occurs. Refer to IR, IMR
    Sn_IMR and Sn_IR. */
}
```

```
方法二：
{
 if (Sn_SR == SOCK_CLOSE_WAIT) goto CLOSED stage;
}
```

建立连接：检查断开连接/断开连接处理

当用户不想与其它节点进行数据通信、或接收到一个FIN数据包，则断开当前的连接。

```
{
/* set DISCON command */
Sn_CR = DISCON;
}
```

建立连接：检查关闭

确认端口是由DISCON断开连接或由CLOSE命令关闭。

```
方法一：
{
 if (Sn_IR(DISCON) == '1') goto CLOSED stage;
 /* In this case, if the interrupt of Socket n-th is activated, interrupt occurs. Refer to IR, IMR
    Sn_IMR and Sn_IR. */
}
```

```
方法二：
{
 if (Sn_SR == SOCK_CLOSED) goto CLOSED stage;
}
```

建立连接：超时

超时会在以下情况下发生: 连接请求(SYN数据包)或它的响应(SYN/ACK数据包)、数据包或它的响应(DATA/ACK数据包)、断开连接请求(FIN数据包)或它的响应(FIN/ACK数据包)、发送所有的TCP数据包。如果这些数据包不能在设定的“ 超时 ”(在RTR和RCR中设置)之内完发送，那TCP的最终超时(TCP超时)将会产生，Sn_SR的状态会设置为 SOCK_CLOSED。TCP超时的确认方法如下。

```
方法一：
{
 if (Sn_IR(TIMEOUT bit) == '1') Sn_IR(TIMEOUT)='1'; goto CLOSED stage;
 /* In this case, if the interrupt of Socket n-th is activated, interrupt occurs. Refer to IR, IMR
    Sn_IMR and Sn_IR. */
}
```

```
方法二：
{
  if (Sn_SR == SOCK_CLOSED) goto CLOSED stage;
}
```

端口关闭

用于关闭SOCKET。其中可利用断开连接处理来断开连接、使用TCP超时来关闭端口、或利用主机在没有断开连接就直接关闭端口。

```
{
  /* clear the remained interrupts of Socket n-th*/
Sn_IR = 0xFF;
    IR(n) = '1';
  /* set CLOSE command */
Sn_CR = CLOSE;
}
```

5.2.1.2 TCP客户端

除了“ CONNECT ”状态以外，其它过程与“ TCP服务器 ”相同。用户可参考“ 5.2.1.1 TCP服务器 ”。

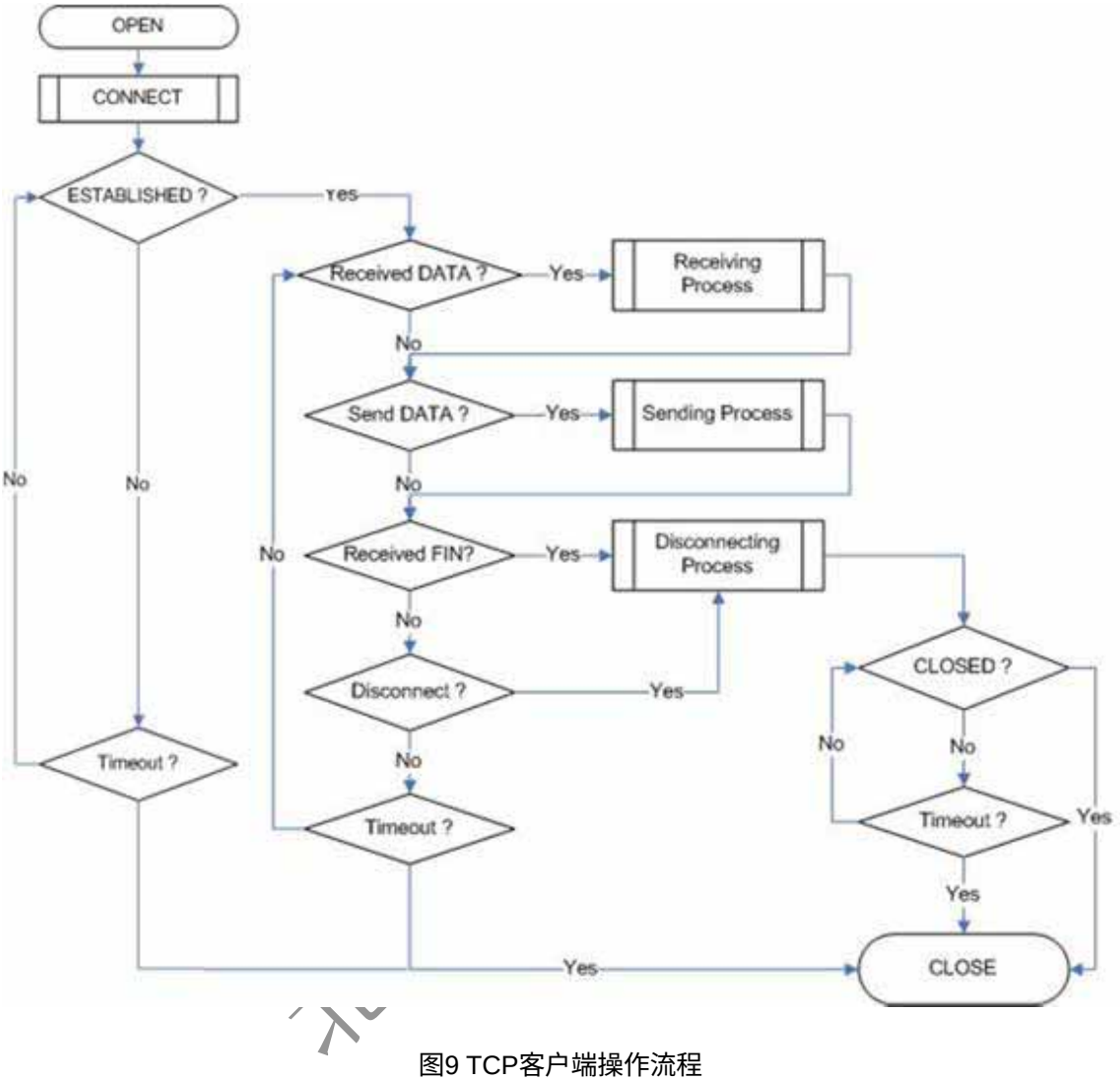


图9 TCP客户端操作流程

连接

发送连接请求（SYN数据包）到“ TCP服务器 ”。在与“ TCP服务器 ”建立连接的过程中可能会产生ARP超时、TCP超时。

```
{
Sn_DIPR0 = server_ip;          /* set TCP SERVER IP address*/
Sn_DPORT0 = server_port;       /* set TCP SERVER listen port number*/
Sn_CR = CONNECT;               /* set CONNECT command */
}
```

5.2.2 UDP

UDP是一个非连接的协议。它的通信不用“ 连接端口 ”。TCP协议保证了可靠的数据通信，但基于UDP协议的数据报不能保证传输数据的可靠性。因为UDP不使用“ 连接端口 ”， 因此它可以通过已知的主机IP地址和端口号与其他很多设备进行通信。这是一个很大的优势；但也有缺陷。只用一个端口与其他设备进行通信 ， 会产生数据丢失、或从其他设备收到意外的数据等问题。为了避免这些问题， 保证可靠性 ， 主机采取重新传输损坏的数据或忽略从其他设备发来的意外接收数据。UDP协议支持单播、广播、多播的通信。它遵循下面的通信流程：

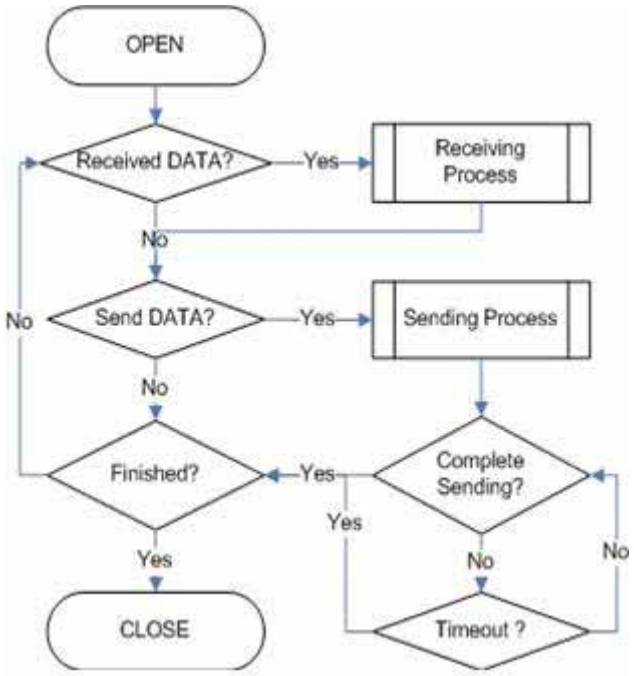


图10 UDP操作流程

5.2.2.1 单播和广播

单播是UDP的一种通信方式。它一次只能将数据传输给一个目的站点。而广播通信则使用广播地址（255.255.255.255），将数据发送给所有的可接收的目的站点。例如，用户将数据传输给目的站点A、B和C。单播是每一次将数据分别传输给A、B或C。在这种情况下，在获得目的站点A、B或C的物理地址时可能产生ARP超时。在产生ARP超时时是不能够将数据传输到目的地的。

广播则使用广播地址（255.255.255.255），可以一次同时将数据发送到目的站点A、B和C。这时不需要得到目的站点A、B或C的物理地址，因此不会产生ARP超时。

注意：广播IP地址可以通过本机IP地址与子网掩码的反码相“ 位或 ”运算而得。

例：本机IP地址为：222.98.173.123，子网掩码为：255.255.255.0，则广播的IP地址则为：222.98.173.255

描述	十进制码	二进制码
主机IP地址	222.098.173.123	11011110.01100010.10101101.01111011
子网掩码的反码	000.000.000.255	00000000.00000000.00000000.11111111
位或	-	-
IP广播IP地址	222.098.173.255	11011110.01100010.10101101.11111111

端口初始化

要实现UDP通信，端口必须进行初始化设置。打开端口的操作过程如下：首先在W5200的8个端口中选择一个为UDP的工作端口，并设置为UDP模式(Sn_MR(P3:P0)=UDP)，然后设置本机端口号(Sn_PORT)，最后运行OPEN命令。执行OPEN命令后，如果端口的Sn_SR状态改变为SOCK_UDP，则完成了端口的初始化设置。

```
{
START:
Sn_MR = 0x02;                    /* sets UDP mode */
Sn_PORT0 = source_port;          /* sets source port number */
Sn_CR = OPEN;                              /* sets OPEN command */
/* wait until Sn_SR is changed to SOCK_UDP */
if (Sn_SR != SOCK_UDP) Sn_CR = CLOSE; goto START;
}
```

检查接收的数据

检查来自目的地的UDP数据，检查的方法与TCP相同，强烈建议采用第二种方法，其原因与TCP的相同。参看“ 5.2.1.1 TCP服务器 ”中的介绍。

```
方法一：
{
 if (Sn_IR(RECV) == '1') Sn_IR(RECV) = '1'; goto Receiving Process stage;
 /* In this case, if the interrupt of Socket n-th is activated, interrupt occurs. Refer to IR, IMR
    Sn_IMR and Sn_IR. */
}
```

```
方法二：
{
    if (Sn_RX_RSR0 != 0x0000) goto Receiving Process stage;
}
```

接收处理

处理内部RX存储器中接收到的UDP数据。UDP数据结构如下：

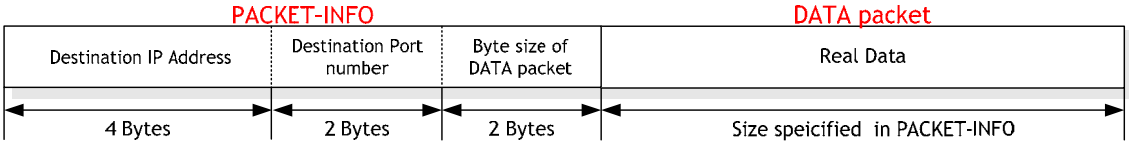


图11 接收的UDP数据结构

接收的UDP数据包含8个字节的数据包信息（PACKET-INFO）和有效数据。数据包信息包括两个部分：发送者的信息（IP地址和端口号）和数据包的长度。UDP可以接收其它的很多UDP数

据，用户可以通过发送者的信息区分UDP数据来源。它也接收以“ 255.255.255.255 ”的广播地址发送的信息。因此主机可以通过分析发送者的信息，丢掉那些不需要的数据。

如果要接收的数据长度大于端口RX存储器的剩余空间， 用户将无法接收到数据， 也不能够接收分片的数据。

```
{
  /* calculate offset address */
  src_mask = Sn_RX_RD&gSn_RX_MASK;       // src_mask is offset address
  /* calculate start address(physical address) */
  src_ptr = gSn_RX_BASE + src_mask;       // src_ptr is physical start address

  /* read head information (8 bytes) */
  header_size = 8;
  /* if overflow SOCKET RX memory */
  if ( (src_mask + header_size) > (gSn_RX_MASK + 1) )
  {
      /* copy upper_size bytes of src_ptr to header_addr*/
      upper_size = (gSn_RX_MASK + 1) – src_mask;
      memcpy(src_ptr, header, upper_size);
      /* update header_addr*/
      header_addr += upper_size;
      /* copy left_size bytes of gSn_RX_BASE toheader_address */
      left_size = header_size – upper_size;
      memcpy(gSn_RX_BASE, header, left_size);
      /* update src_mask */
      src_mask = left_size;
  }
  else
  {
      /* copy header_size bytes of get_start_address toheader_address*/
      memcpy(src_ptr, header, header_size);
      /* update src_mask */
      src_mask += header_size;
  }
  /* update src_ptr */
  src_ptr = gSn_RX_BASE + src_mask;

  /* save remote peer information & received data size */
  peer_ip = header[0 to 3];
```

```
peer_port = header[4 to 5];
get_size = header[6 to 7];

/* if overflow SOCKET RX memory */
if ( (src_mask + get_size) > (gSn_RX_MASK + 1) )
{
    /* copy upper_size bytes of src_ptr to destination_address*/
    upper_size = (gSn_RX_MASK + 1) – src_mask;
    memcpy(src_ptr, destination_addr, upper_size);
    /* update destination_addr*/
    destination_addr += upper_size;
    /* copy left_size bytes of gSn_RX_BASE todestination_address */
    left_size = get_size – upper_size;
    memcpy(gSn_RX_BASE, destination_addr, left_size);
}
else
{
    /* copy len bytes of src_ptr todestination_address */
    memcpy(src_ptr, destination_addr, get_size);
}
/* increase Sn_RX_RD as length of len+ header_size */
 Sn_RX_RD = Sn_RX_RD + header_size + get_size;
/* set RECV command */
Sn_CR = RECV;
}
```

检查发送的数据/发送过程

要发送的数据不能比端口内部TX存储器定义的空间大。但如果比MTU大，它将根据MTU自动进行分片，然后再发送。如果要使用广播发送，Sn_DIPR设置为“ 255.255.255.255 ”。

```
{
 /* first, get the free TX memory size */
FREESIZE:
 freesize = Sn_TX_FSR0;
 if (freesize<len) goto FREESIZE;       // len is send size

 /* Write the value of remote_ip, remote_port to the Socket n-th Destination IP Address
    Register(Sn_DIPR), Socket n-th Destination Port Register(Sn_DPORT). */
    Sn_DIPR0 = remote_ip;
```

```
    Sn_DPORT0 = remote_port;

  /* calculate offset address */
  dst_mask = Sn_TX_WR0 &gSn_TX_MASK;       // dst_mask is offset address
  /* calculate start address(physical address) */
  dst_ptr = gSn_TX_BASE + dst_mask;       // dst_ptr is physical start address

  /* if overflow SOCKETTX memory */
  if ( (dst_mask + len) > (gSn_TX_MASK + 1) )
  {
      /* copy upper_size bytes of source_address to dst_ptr */
      upper_size = (gSn_TX_MASK + 1) – dst_mask;
      memcpy(src_ptr, destination_addr, upper_size);

      /* update source_address*/
      source_address += upper_size;
      /* copy left_size bytes of source_address to gSn_TX_BASE */
      left_size = send_size – upper_size;
      memcpy(src_ptr, destination_addr, left_size);
  else
  {
      /* copy len bytes of source_address to dst_ptr */
      memcpy(src_ptr, destination_addr, len);
  }
 /* increase Sn_TX_WR0 as length of len */
  Sn_TX_WR0 += len;
  /* set SEND command */
 Sn_CR = SEND;
 }
```

检查发送完成/超时

为了下一次的数据发送，用户必须检查前一次的SEND命令是否完成。数据量越大，所需要的发送时间就越长。因此用户可以将数据适当分片然后再进行发送。在数据UDP发送时可能会产生ARP超时。如果产生ARP超时，则UDP发送数据失败。

```
  方法一：
  {
 /* check SEND command completion */
 while(Sn_IR(SENDOK)==‘0’)    /* wait interrupt of SEND completion */
```

```
{
  /* check ARP_TO */
  if (Sn_IR(TIMEOUT)=='1') Sn_IR(TIMEOUT)='1'; goto Next stage;
}
Sn_IR(SENDOK) = '1'; /* clear previous interrupt of SEND completion */
}
```

方法二：

```
{
If (Sn_CR == 0x00) transmission is completed.
If (Sn_IR(TIMEOUT bit) == '1') goto next stage;
/* In this case, if the interrupt of Socket n-this activated, interrupt occurs. Refer to Interrupt Register(IR), Interrupt Mask Register (IMR) and Socket n-thInterrupt Register (Sn_IR). */
}
```

检查通信结束/关闭端口

如果用户不再需要通信，可以关闭端口。

```
{
 /* clear remained interrupts */
Sn_IR = 0x00FF;
    IR(n) = '1';
 /* set CLOSE command */
Sn_CR = CLOSE;
}
```

5.2.2.2 多播

广播是与所有的、不确定的目的站点进行通信。但多播是与多个、但在多播组注册的目的站点进行通信。假如A、B和C是在一个特定的多播组里注册的站点。如果用户将数据传送到多播组(站点A)，站点B和C也能够从站点A得到数据。为了使用多播通信，使用IGMP协议将目的站点列表注册到多播组。多播组包括：分组硬件地址、分组IP地址和组端口号。用户不能够更改“分组硬件地址”和“分组IP地址”。但用户可以更改“分组端口号”。

分组硬件地址的选择范围在“01:00:5e:00:00:00”到“01:00:5e:7f:ff:ff”之间，而分组IP地址则使用D类地址，范围从“224.0.0.0”到“239.255.255.255”。详细内容请参考官方网站：http://www.iana.org/assignments/multicast-addressesU).

在选择时，6个字节的高23位硬件地址和4个字节的组IP地址必须相同。例如，如果用户选择的组IP地址为“244.1.1.11”，那么组硬件地址为“01:00:5e:01:01:0b”。详细信息请参考RFC1112：http://www.ietf.org/rfc.html。

在W5200内部，IGMP处理多播注册是由内部自动完成的。当用户以多播的模式打开端口时，“JOIN”信息将在内部自动传送。如果用户关闭端口，“LEAVE”信息将在内部自动传送。端口

打开以后，“ REPORT ”信息将在数据传输过程中每隔一定的时间传送。

W5200支持IGMP v1和v2版本。如果用户想使用一个升级的版本，主机可以使用IPRAW模式直接处理IGMP。

端口初始化

在W5200的8个端口中选择一个端口作为多播通信的工作端口。设置Sn_DHAR0为多播组硬件地址，设置Sn_DIPR0为多播组IP地址。然后设置Sn_PORT0和Sn_DPORT0为多播组端口号，将Sn_MR（P3:P0）设置为UDP，且将Sn_MR（MULTI）设置为‘ 1 ’。最后运行OPEN命令。如果Sn_SR的状态在执行完OPEN命令后改变为SOCK_UDP，端口的初始化即完成。

```
{
START:
 /* set Multicast-Group information */
 Sn_DHAR0 = 0x01;      /* set Multicast-Group H/W address(01:00:5e:01:01:0b) */
 Sn_DHAR1 = 0x00;
 Sn_DHAR2 = 0x5E;
 Sn_DHAR3 = 0x01;
 Sn_DHAR4 = 0x01;
 Sn_DHAR5 = 0x0B;
 Sn_DIPR0 = 211;         /* set Multicast-Group IP address(211.1.1.11) */
 Sn_DIPR1 = 1;
 Sn_DIPR2 = 1;
 Sn_DIRP3 = 11;
 Sn_DPORT0 = 0x0BB8;           /* set Multicast-GroupPort number(3000) */
 Sn_PORT0 = 0x0BB8;  /* set SourcePort number(3000) */
 Sn_MR = 0x02 | 0x80; /* set UDP mode & Multicast on Socket n-th Mode Register */

 Sn_CR = OPEN;         /* set OPEN command */

 /* wait until Sn_SR is changed to SOCK_UDP */
 if (Sn_SR != SOCK_UDP) Sn_CR = CLOSE; goto START;
}
```

检查接收的数据

参考“ 5.2.2.1 单播和广播 ”。

接收处理

参考“ 5.2.2.1 单播和广播 ”。

检查发送的数据/发送处理

因为在端口初始化时用户设置了组播信息，用户不需要再设置目的IP和目的端口号。因此，将数据拷贝到内部TX存储器然后执行SEND命令。

```
{
  /* first, get the free TX memory size */
 FREESIZE:
  freesize = Sn_TX_FSR;
  if (freesize<len) goto FREESIZE;       // len is send size

  /* calculate offset address */
  dst_mask = Sn_TX_WR0 &gSn_TX_MASK;       // dst_mask is offset address
  /* calculate start address(physical address) */
  dst_ptr = gSn_TX_BASE + dst_mask;       // dst_ptr is physical start address
  /* if overflow SOCKETTX memory */
  if ( (dst_mask + len) > (gSn_TX_MASK + 1) )
  {
      /* copy upper_size bytes of source_addr to destination_address*/
      upper_size = (gSn_TX_MASK + 1) – dst_mask;
      wizmemcpy((0x000000 + source_addr), (0xFE0000 + dst_ptr), upper_size);
      /* update source_addr*/
      source_addr += upper_size;
      /* copy left_size bytes of source_addr to gSn_TX_BASE */
      left_size = len – upper_size;
      wizmemcpy(source_addr, gSn_TX_BASE, left_size);
  }
  else
  {
      /* copy len bytes of source_addr to dst_ptr */
      wizmemcpy(source_addr, dst_ptr, len);
  }
 /* increase Sn_TX_WR as length of len */
  Sn_TX_WR0 += send_size;
 /* set SEND command */
 Sn_CR = SEND;
}
```

检查发送完成/超时

因为主机管理所有的数据通讯协议的处理过程，因此不会出现超时的现象。

```
{
/* check SEND command completion */
while(S0_IR(SENDOK)==‘0’);    /* wait interrupt of SEND completion */
S0_IR(SENDOK) = ‘1’;          /* clear previous interrupt of SEND completion */
}
```

完成操作/关闭端口

参考“ 5.2.2.1 单播和广播 ”。

5.2.3 IPRAW

IPRAW属于IP层的数据通信 ， 它是比TCP、UDP低一层协议。IPRAW支持IP层的协议 ， 如ICMP（0x01）和IGMP（0x02），由协议号决定。ICMP的‘ PING ’功能和IGMP v1/v2已经在W7100A中由硬件实现。如果用户需要， 主机可以将SOCKET n以IPRAW的模式打开， 直接处理IPRAW的数据。在使用IPRAW模式时， 用户必须设置IP头中用户所使用的协议号。协议号由IANA定义，请参考官方网站：

http://www.iana.org/assignments/protocol-numbers。

在打开端口前，必须先由Sn_PROTO定义协议号。W5200在IPRAW模式下不支持TCP（0x06）和UDP（0x11）。IPRAW模式的端口通信只支持设定的协议号通信。ICMP端口不能接收非设定的协议数据，如IGMP。

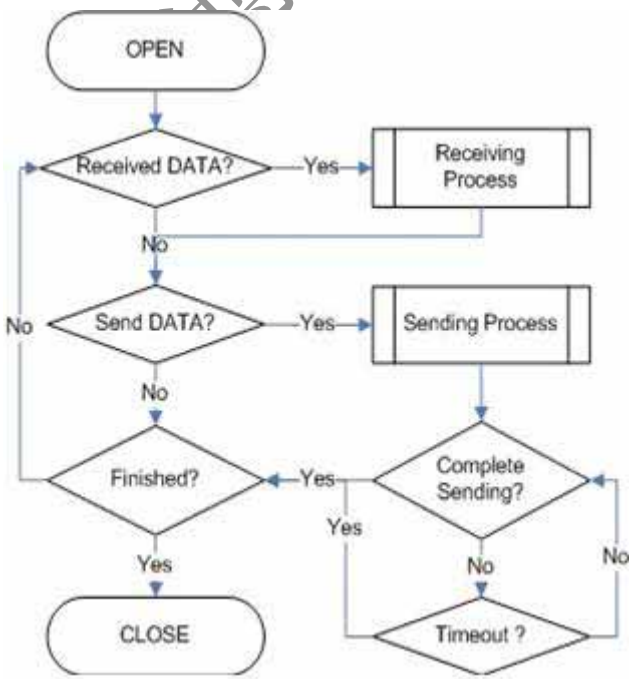


图12 IPRAW操作流程

端口初始化

选择一个端口设置协议号。设置Sn_MR（P3:P0）为IPRAW模式并运行OPEN命令。如果执行OPEN命令后的Sn_SR的状态改变为SOCK_IPRAW，端口初始化则完成。

```
{
START:
/* sets Protocol number */
  /* The protocol number is used in Protocol Field of IP Header. */
  Sn_PROTO = protocol_num;
  /* sets IP raw mode */
  Sn_MR = 0x03;
/* sets OPEN command */
  Sn_CR = OPEN;
 /* wait until Sn_SR is changed to SOCK_IPRAW */
  if (Sn_SR != SOCK_IPRAW) Sn_CR = CLOSE; goto START;
}
```

检查接收的数据

参考“ 5.2.2.1 单播和广播 ”。

接收处理

处理内部RX存储器的IPRAW数据。IPRAW的数据结构如下所示。

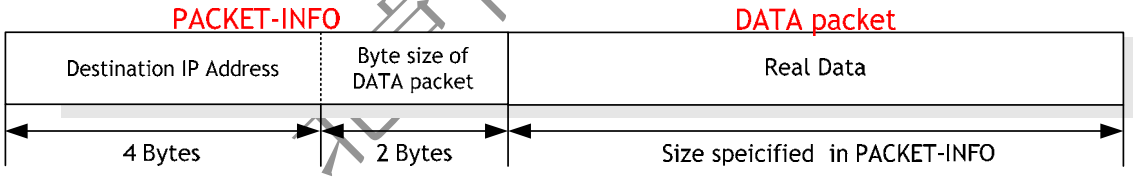


图13 接收的IPRAW数据格式

IPRAW的数据包括6个字节的数据包信息和数据。数据包信息包括：发送者信息（IP地址）和数据长度信息。IPRAW的数据包与UDP数据包结构相同，只是IPRAW数据包中没有端口号的信息。请参考“ 5.2.2.1 单播和广播 ”。

如果传输数据的长度大于RX存储器的剩余空间，用户不能接收数据，也不能接收分片的数据。

检查发送的数据/发送处理

用户要发送的数据长度不能超过内部TX存储器的大小，也不能超过默认的MTU。IPRAW数据传输与UDP的数据传输相同，只是没有目标端口号。参考“ 5.2.2.1 单播和广播 ”。

完成发送/超时

与UDP相同，参考“ 5.2.2 UDP ”。

检查完成通信/端口关闭

与UDP相同，参考“ 5.2.2 UDP ”。

5.2.4 MACRAW

MACRAW通信是基于以太网MAC层的通信，它可以让用户更灵活地使用上层协议。

MACRAW模式只能使用一个端口。如果用户使用端口0为MACRAW模式 ， 端口1-7不仅可以作“ 硬件TCPIP协议栈 ”用，而且该端口还可以作网络接口控制用（NIC），因此任何端口1-7都可以用于“ 软件TCPIP协议栈 ”。因为W7100A可以支持“ 硬件TCPIP协议栈 ”和“ 软件TCPIP协议栈 ”，因此称它为“ 混合TCPIP协议栈 ”。如果用户想要多于8个端口，那么需要高性能的应用则使用“ 硬件TCPIP协议栈 ”，而其它的则使用MACRAW模式，由“ 软件TCPIP协议栈 ”完成。这样就可以突破8个端口的限制。端口的MACRAW模式可以处理所有的协议（端口1-7除外）。因为MACRAW通信纯粹是以太网数据包通信（没有其它的处理），MACRAW设计者需要使用“ 软件TCPIP协议栈 ”进行协议处理。MACRAW数据包需要一些基本信息：6个字节的本机物理地址，6个字节的目的物理地址和2个字节的以太网类型，这些都是基于以太网MAC层的信息。

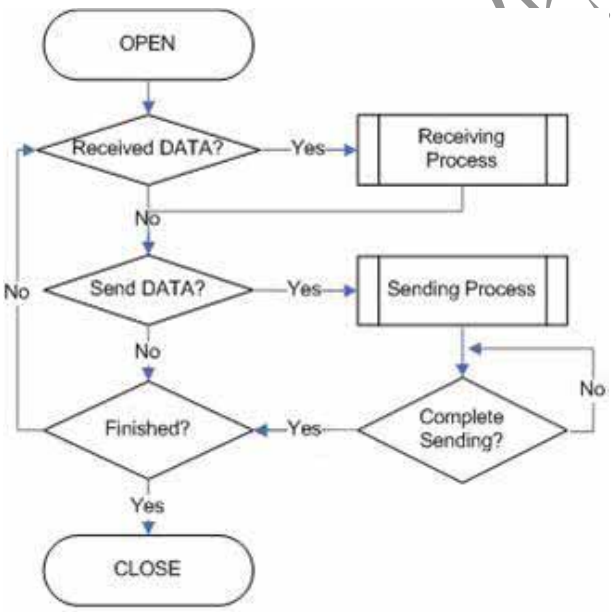


图14 MACRAW操作流程

端口初始化

选择SOCKET0，设置S0_MR（P3:P0）为MACRAW模式，然后运行“ OPEN ”命令。执行“ OPEN ”命令后，Sn_SR的状态改变为“ SOCK_MACRAW ”，则完成端口初始化。因为所有的通信信息（本机物理地址、IP地址和端口号，目的物理地址、IP地址和端口号）都是“ MACRAW ”数据，因此不需要设置更多的寄存器。

```
{
START:
 /* sets MAC raw mode */
 S0_MR = 0x04;
```

```
  /* sets OPEN command */
   S0_CR = OPEN;
   /* wait until Sn_SR is changed to SOCK_MACRAW */
   if (Sn_SR != SOCK_MACRAW) S0_CR = CLOSE; goto START;
  }
```

检查接收的数据

参考“ 5.2.2.1 单播和广播 ”。

接收处理

处理端口内部RX存储器接收到的MACRAW的数据，MACRAW数据结构如下：

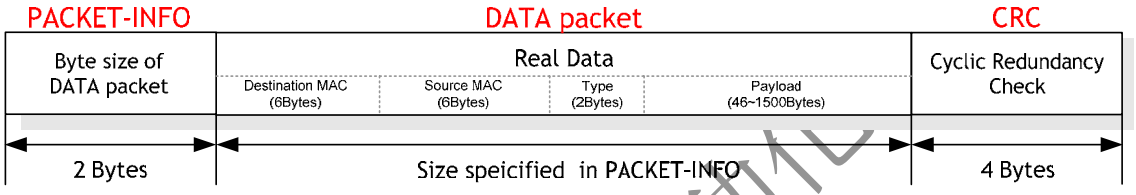


图15 接收的IPRAW数据结构

MACRAW数据包有以下信息 ： 数据包信息、数据和4个字节的CRC。数据包信息就是这个数据包的长度。数据有 ：6个字节的目的MAC地址，6个字节的本机MAC地址和2个字节的“ 类型 ”，46~1500字节的有效载荷。根据其“ 类型 ”，数据包的有效载荷包含网络协议，如ARP、IP等。有关“ 类型 ”的详细内容请参看网站：

http://www.iana.org/assignments/ethernet-numbers

```
{
 /* calculate offset address */
 src_mask = Sn_RX_RD&gSn_RX_MASK;       // src_mask is offset address
 /* calculate start address(physical address) */
 src_ptr = gSn_RX_BASE + src_mask;       // src_ptr is physical start address
 /* get the received size */
 len = get_Byte_Size_Of_Data_packet // get Byte size of DATA packet from Packet-INFO
 /* if overflow SOCKET RX memory */
 If((src_mask + len) > (gSn_RX_MASK + 1))
 {
 /* copy upper_size bytes of get_start_address to destination_address */
 upper_size = (gSn_RX_MASK + 1) – src_mask;
    memcpy(src_ptr, dst_addr, upper_size);
 /* update destination_address */
 dst_addr += upper_size;
 /* copy left_size bytes of gSn_RX_BASE to destination_address */
```

```
    left_size = len – upper_size;
     memcpy(src_ptr, dst_addr, left_size);
    }
    else
    {
    /* copy len bytes of src_ptr to destination_address */
    memcpy(src_ptr, dst_addr, len);
    }
    /*   increase Sn_RX_RD as length of len */
    Sn_RX_RD += len;
    /* extract 4 bytes CRC from internal RX memory and then ignore it */
    memcpy(src_ptr, dst_addr, len);
    /* set RECV command */
    Sn_CR = RECV;
  }
```

注意

如果RX存储器的剩余空间比MACRAW的数据小，存储在RX存储器的一些信息，如数据包信息和数据，可能会产生一些错误。数据包信息的错误将会引起MACRAW数据处理的错误。RX存储器中的数据越满，出现错误的可能性就越大。如果允许丢掉一部分MACRAW的数据，这个问题可以作如下解决：

- 尽最大可能提高处理RX存储器的速度，以防止填满数据。
- 只接收其自身的数据，以减少接收数据的负荷。

在下面的端口初始化的例程中设置S0_MR的MF为‘1’，

```
{
START:
/* sets MAC raw mode with enabling MAC filter */
S0_MR = 0x44;
/* sets OPEN command */
S0_CR = OPEN;
/* wait until Sn_SR is changed to SOCK_MACRAW */
if (Sn_SR != SOCK_MACRAW) S0_CR = CLOSE; goto START;
}
```

如果端口内部的RX存储器的剩余空间小于1528个字节 - 默认的“INFO(2)+ DATA packet(8) + CRC(4)”，关闭端口然后处理接收的数据，然后再打开端口。端口关闭以后，在端口关闭期间来的MACRAW数据将丢失。

```
{
/* check the free size of internal RX memory */
if((Sn_RXMEM_SIZE(0) * 1024) - Sn_RX_RSR0(0) < 1528)
   {
     recved_size = Sn_RX_RSR0(0);          /* backup Sn_RX_RSR */
        Sn_CR0 = CLOSE;                    /* SOCKET Closed */
     while(Sn_SR != SOCK_CLOSED);          /* wait until SOCKET is closed */
     /* process all data remained in internal RX memory */
   while(recved_size> 0)
       {/* calculate offset address */
          src_mask = Sn_RX_RD&gSn_RX_MASK;     // src_mask is offset address
          /* calculate start address(physical address) */
          src_ptr = gSn_RX_BASE + src_mask;    // src_ptr is physical start address
          /* if overflow SOCKET RX memory */
          If((src_mask + len) > (gSn_RX_MASK + 1))
          {
          /* copy upper_size bytes of get_start_address to destination_address */
             upper_size = (gSn_RX_MASK + 1) – src_mask;
             memcpy(src_ptr, dst_addr, upper_size);
             /* update destination_address */
             dst_address += upper_size;
             /* copy left_size bytes of gSn_RX_BASE to destination_address */
             left_size = len – upper_size;
             memcpy(src_ptr, dst_addr, left_size);
          }
          else
          {   /* copy len bytes of src_ptr to destination_address */
             memcpy(src_ptr, dst_addr, len);

          }
          /* increase Sn_RX_RD as length of len */
          Sn_RX_RD += len;
          /* extract 4 bytes CRC from internal RX memory and then ignore it */
          memcpy(src_ptr, dst_addr, len);
          /* calculate the size of remained data in internal RX memory*/
recved_size = recved_size – 2 – len – 4;
       }
   /* Reopen the SOCKET */
```

```
  /* sets MAC raw mode with enabling MAC filter */
     S0_MR = 0x44; /* or S0_MR = 0x04 */
/* sets OPEN command */
     S0_CR = OPEN;
     /* wait until Sn_SR is changed to SOCK_MACRAW */
     while (Sn_SR != SOCK_MACRAW);
   }
else /* process normally the DATA packet from internal RX memory */
   {/* This block is same as the code of “Receiving process” stage*/
   }
}
```

检查发送数据/发送处理

用户要发送的数据不能比内部TX存储器的空间大，也不能大于默认的MTU。主机要产生与接收的数据格式相同的MACRAW数据，并发送。如果数据小于60个字节，内部将填充‘0’到发送的以太网数据包，然后再发送。

```
{
  /* first, get the free TX memory size */
FREESIZE:
  freesize = S0_TX_FSR;
  if (freesize<send_size) goto FREESIZE;
  /* calculate offset address */
  dst_mask = Sn_TX_WR0 &gSn_TX_MASK;       // dst_mask is offset address
  /* calculate start address(physical address) */
  dst_ptr = gSn_TX_BASE + dst_mask;       // dst_ptr is physical start address

  /* if overflow SOCKETTX memory */
  if ( (dst_mask + len) > (gSn_TX_MASK + 1) )
  {/* copy upper_size bytes of source_addr to destination_address*/
       upper_size = (gSn_TX_MASK + 1) – dst_mask;
       memcpy(src_ptr, dst_addr,upper_size);
       /* update source_addr*/
       source_addr += upper_size;
       /* copy left_size bytes of source_addr to gSn_TX_BASE */
       left_size = len – upper_size;
       memcpy(src_ptr, dst_addr,left_size);
  }
  else
```

```
  {/* copy len bytes of source_addr to destination_address*/
      memcpy(src_ptr, dst_addr, len);
  }
 /* increase Sn_TX_WR as length of len */
  Sn_TX_WR += send_size;

 /* set SEND command */
    S0_CR = SEND;
}
```

检查发送完成

因为主机管理所有协议处理进行数据通信，因此不会发生超时的现象。

```
{
/* check SEND command completion */
while(S0_IR(SENDOK)==‘0’);    /* wait interrupt of SEND completion */
S0_IR(SENDOK) = ‘1’;          /* clear previous interrupt of SEND completion */
}
```

检查操作是否结束/关闭端口

参考“ 5.2.2.1 单播和广播 ”。

6 外部接口

W5200提供SPI接口与MCU通信。

6.1 SPI模式

SPI接口只使用4根引脚用于数据通信，对应的引脚为nSCS, SCLK, MOSI和MISO。

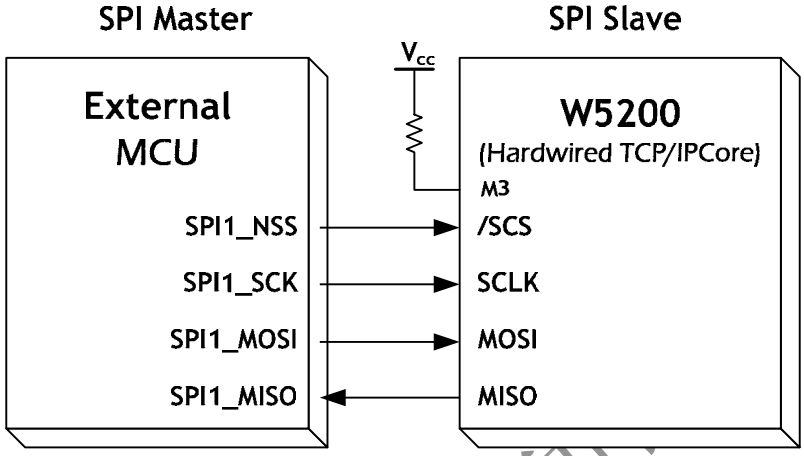


图16 W5200的SPI接口

6.2 器件操作

外部主机（通常被称为SPI主机）通过一系列的指令对W5200进行控制。SPI主机通过SPI总线的4个信号线与W5200通信，这些信号是片选信号（nSCS），时钟信号（SCLK），MOSI（主出从入）和MISO（主入从出）。

SPI协议定义了4种模式，每种模式的差异在于SCLK的极性和相位，即SCLK的极性和相位如何控制SPI数据流。W5200使用SPI模式中最常用的两种，模式0和模式3。

模式0和模式3的区别在于非激活状态SCLK的极性。通过SPI模式0和模式3，数据在SCLK的上升沿锁定，而在下降沿输出。

6.3 使用通用的SPI主机处理过程

- 在主机SPI设备上定义输入/输出的方向；
- 在非激活状态nSCS设置为高电平；
- 写入要发送的目的地址到SPDR寄存器；
- 写入要发送的操作代码和数据长度到SPDR寄存器；
- 写入要发送的数据到SPDR寄存器；
- 设置nSCS为低电平（启动数据传输）；
- 等待接收完成；
- 所有数据发送结束，设置nSCS为高电平。

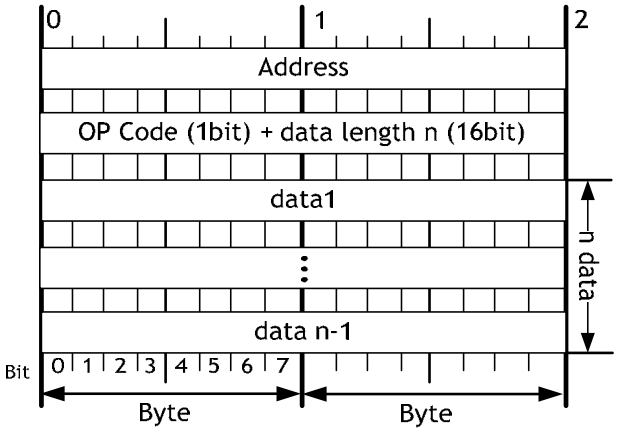


图17 W5200 SPI数据帧的格式

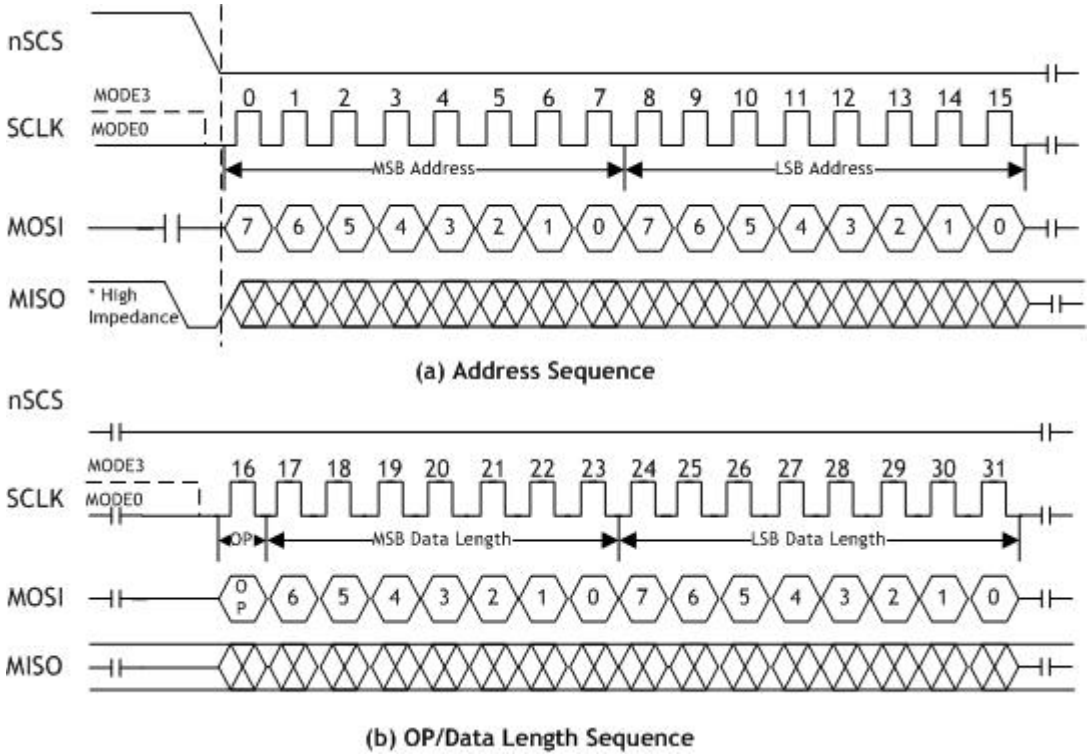


图18 地址和操作码/数据长度时序图

读操作

‘读’过程的时序图如图20所示。‘读’过程是先将nSCS引脚置低电平，然后在MOSI引脚输出地址、操作码、数据长度，数据字节在W5200的MISO引脚输出。地址、操作码/数据长度和数据在图19所示。操作码定义“读操作”和“写操作”，OP=0是选择读操作，OP=1则是选择写操作。

在W5200的SPI模式，可支持单字节‘读’和数据块多字节‘读’操作。单字节‘读’需要4个部分的指令，16位地址，一个操作码（0x0）和15位数据长度，然后才是数据。数据块‘读’在发送了‘读’操作指令之后只需要连续进行‘读’操作。为了区分单字节‘读’和数据块‘读’，所以需要数据长度信息。如果数据长度为1，则是单字节‘读’，否则为多字节‘读’。在nSCS为低电平后，驱动MISO为低电平，以选择MISO引脚。

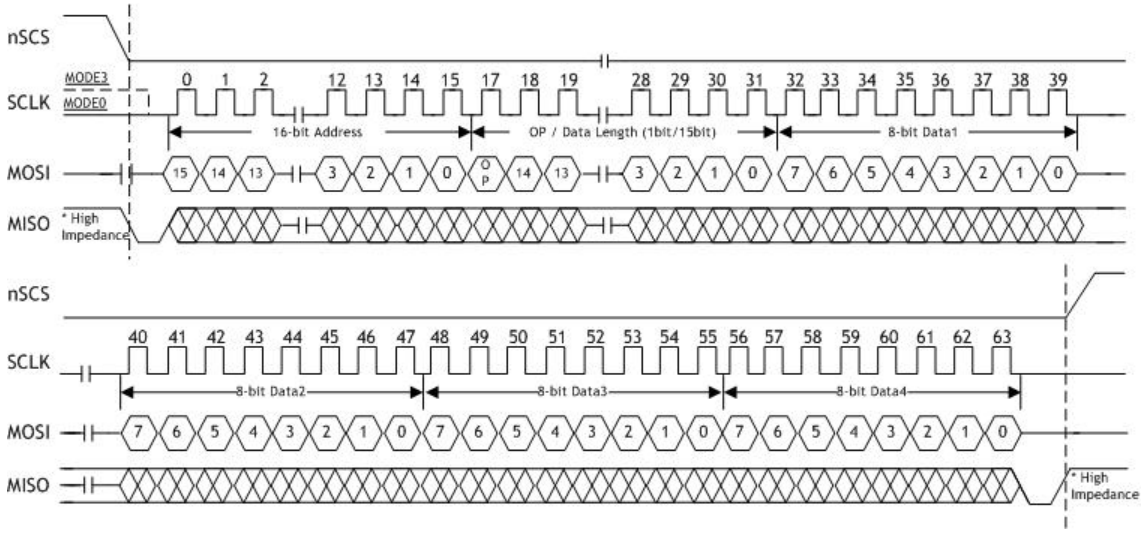


图19 读时序

```
/* Pseudo Code for Read data of 8bit per packet */
#define data_read_command0x00
uint16 addr;       // Address : 16bits
int16 data_len;     // Datalength :15bits
uint8 data_buf[];// Array for data
SpiSendData();     // Send data from MCU to W5200
SpiRecvData();     //Receive data from W5200 to MCU

{
ISR_DISABLE();// Interrupt Service Routine disable
CSoff();// CS=0, SPI start

//SpiSendData
SpiSendData(((addr+idx) & 0xFF00) >> 8);//Address byte 1
SpiSendData((addr+idx) & 0x00FF);       //Address byte 2

// Data write command + Data length upper 7bits
SpiSendData((data_read_command| ((data_len& 0x7F00) >> 8)));
// Data length bottom 8bits
SpiSendData((data_len& 0x00FF));

//Read data:On data_len> 1, Burst Read Processing Mode.
for(intidx = 0; idx<data_len; idx++)
{
    SpiSendData(0);// Dummy data
```

```
    data_buf[idx] = SpiRecvData(idx);//Read data
  }
  CSon();// CS=1, SPI end
  ISR_ENABLE();// Interrupt Service Routine disable
  }
```

写操作

‘写’过程的时序图如图21所示。‘写’过程是先将nSCS引脚置低电平，然后在MOSI输出地址、操作码、数据长度和数据字节。

在W5200的SPI模式，支持单字节‘写’操作和多字节数据块‘写’操作。单字节‘写’需要4个部分的指令，16位地址，一个操作码（0x0）和15位数据长度，然后是数据。数据块‘写’在发送了‘写’操作指令之后只需要连续进行‘写’操作。为了区分单字节‘写’和数据块‘写’，所以需要数据长度信息。如果数据长度为1，则是单字节‘写’，否则为多字节‘写’。在nSCS为低电平后，驱动MOSI为低电平，以选择MOSI引脚。

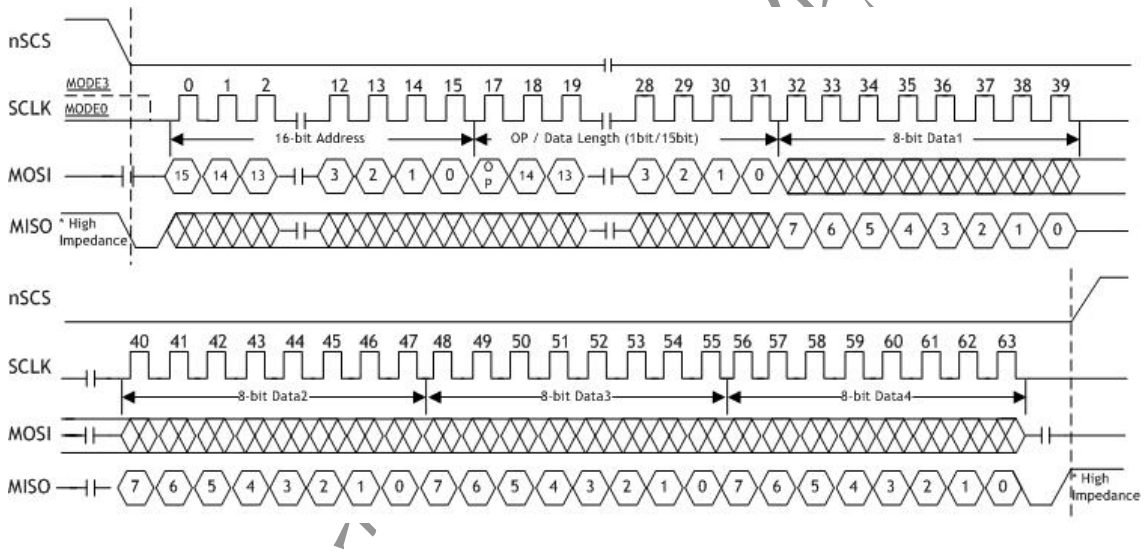


图20 写时序

```
/* Pseudo Code for Write data of 8bit per packet */
#define data_write_command0x80
uint16 addr;       // Address : 16bits
int16 data_len;       // Datalength :15bits
uint8 data_buf[];// Array for data

{
SpiSendData();     //Send data from MCU to W5200

ISR_DISABLE();// Interrupt Service Routine disable
```

```
CSoff();// CS=0, SPI start

SpiSendData(((addr+idx) & 0xFF00) >> 8);//Address byte 1
SpiSendData((addr+idx) & 0x00FF);//Address byte 2

// Data write command + Data length upper 7bits
SpiSendData((data_write_command | ((data_len& 0x7F00) >> 8)));
// Data length bottom 8bits
SpiSendData((data_len& 0x00FF));

//Write data: On data_len> 1, Burst Write Processing Mode.
for(intidx = 0; idx<data_len; idx++)
    SpiSendData(data_buf[idx]);

CSon();// CS=1, SPI end
IINCHIP_ISR_ENABLE();// Interrupt Service Routine disable
}
```

7 电气特性参数

7.1 极限值

符号	参数	参数值	单位
V_{DD}	直流电压	-0.5 to 3.63	V
V_{IN}	直流输入电压	-0.5 to 5.5 (5V tolerant)	V
I_{IN}	直流输入电流	±5	mA
T_{OP}	工作温度	-40 to 85	°C
T_{STG}	储藏温度	-55 to 125	°C

注意：器件在超过极限参数的条件下工作时，将可能造成永久性的损坏。

7.2 直流特性

符号	参数	测试条件	最小	典型	最大	单位
VDD	直流电源电压	节点温度从-55°C到125°C	2.97		3.63	V
VIH	高电平输入电压		2.0		5.5	V
VIL	低电平输入电压		- 0.3		0.8	V
VOH	高电平输出电压	IOH = 4 ~8 mA	2.4			V
VOL	低电平输出电压	IOL = 4 ~8mA			0.4	V
II	输入电流	VIN = VDD			±5	μA

7.3 功耗(Vcc= 3.3V，温度：25°C)

条件	最小	典型	最大	单位
100M以太网连接	-	160	175	mA
10M以太网连接	-	110	125	mA
以太网连接断开	-	125	140	mA
100M发送	-	160	175	mA
10M发送	-	110	125	mA
低功耗模式	-	2	4	mA

7.4 交流特性参数

7.4.1 复位时序

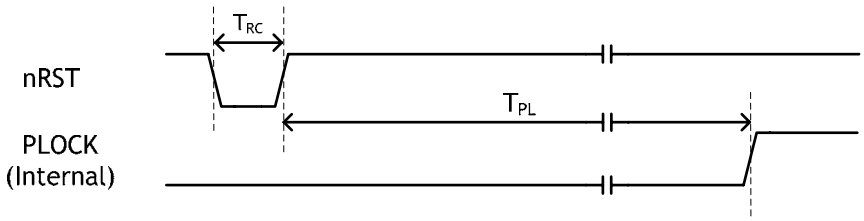


图21 复位时序

符号	描述	最小	Max
TRC	复位时间	2 us	-
TPL	nRST内部PLOCK时间	-	150 ms

7.4.2 晶体特性

参数	取值范围
频率	25 MHz
频偏 (25℃温度)	±30 ppm
偏差电容	7pF Max
驱动电平	59.12uW/MHz
负载电容	27pF
老化（25℃温度)	±3ppm / year Max

7.4.3 SPI时序

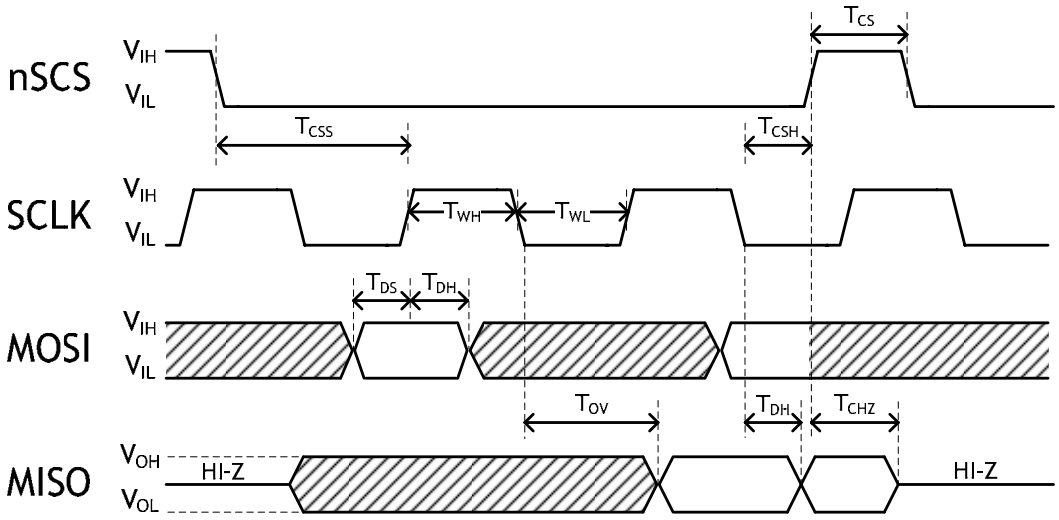


图22 SPI时序

符号	描述	最小	最大	单位
F_{SCK}	SCK Clock Frequency		80	MHz
T_{WH}	SCK High Time	6		ns
T_{WL}	SCK Low Time	6		ns
T_{CS}	nSCS High Time	5		ns
T_{CSS}	nSCS Hold Time	5	-	ns
T_{CSH}	nSCS Hold Time	5		ns
T_{DS}	Data In Setup Time	3		ns
T_{DH}	Data In Hold Time	3		ns
T_{OV}	Output Valid Time		5	ns
T_{OH}	Output Hold Time	0		ns
T_{CHZ}	nSCS High to Output Hi-Z		5	ns

7.4.4 网络变压器特性

Parameter参数	Transmit End发送端	Receive End接收端
Turn Ratio变比	1:1	1:1
Inductance电感	350 uH	350 uH

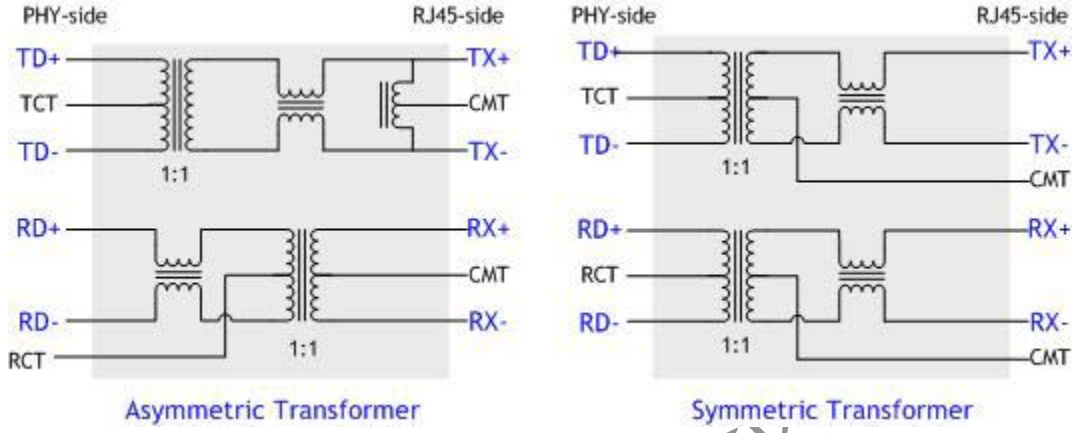


图23 网络变压器类型

使用内部PHY，为了支持自动MDI/MDIX（交叉），确保使用对称的网络变压器。

8 IR Reflow Temperature Profile (Lead-Free)

Moisture Sensitivity Level : 3

Dry Pack Required: Yes

Average Ramp-Up Rate (Ts_{max} to Tp)	3° C/second max.
Preheat – Temperature Min (Ts_{min}) – Temperature Max (Ts_{max}) – Time (ts_{min} to ts_{max})	 150 °C 200 °C 60-120 seconds
Time maintained above: – Temperature (TL) – Time (tL)	 217 °C 60-150 seconds
Peak/Classification Temperature (Tp)	265 + 0/-5°C
Time within 5 °C of actual Peak Temperature (tp)	30 seconds
Ramp-Down Rate	6 °C/second max.
Time 25 °C to Peak Temperature	8 minutes max.

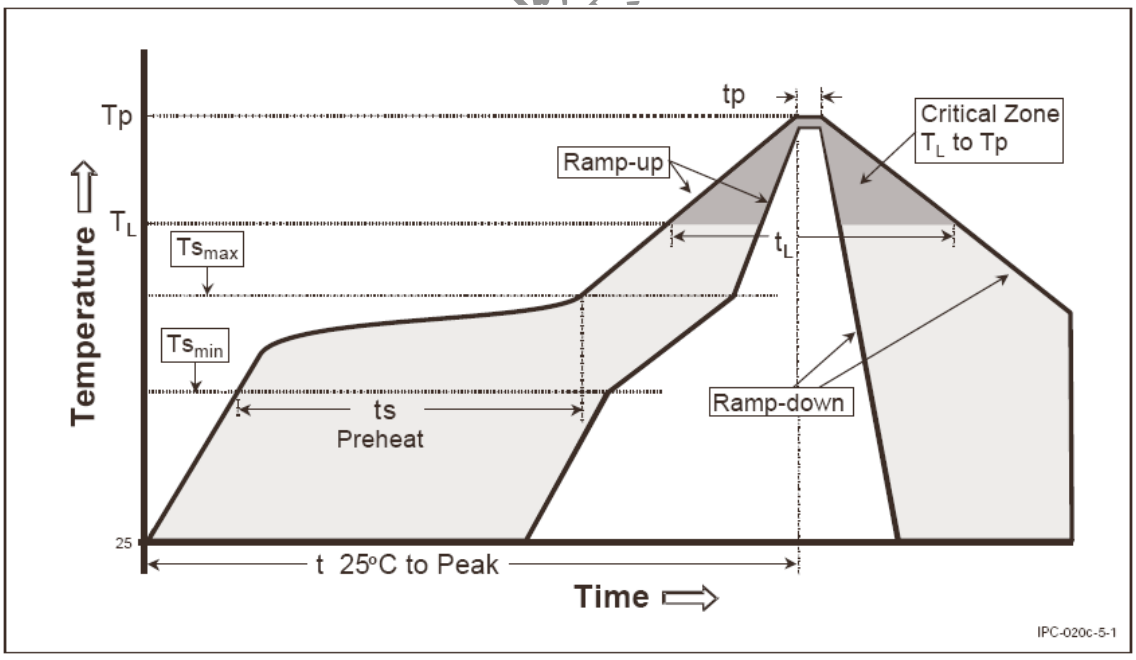
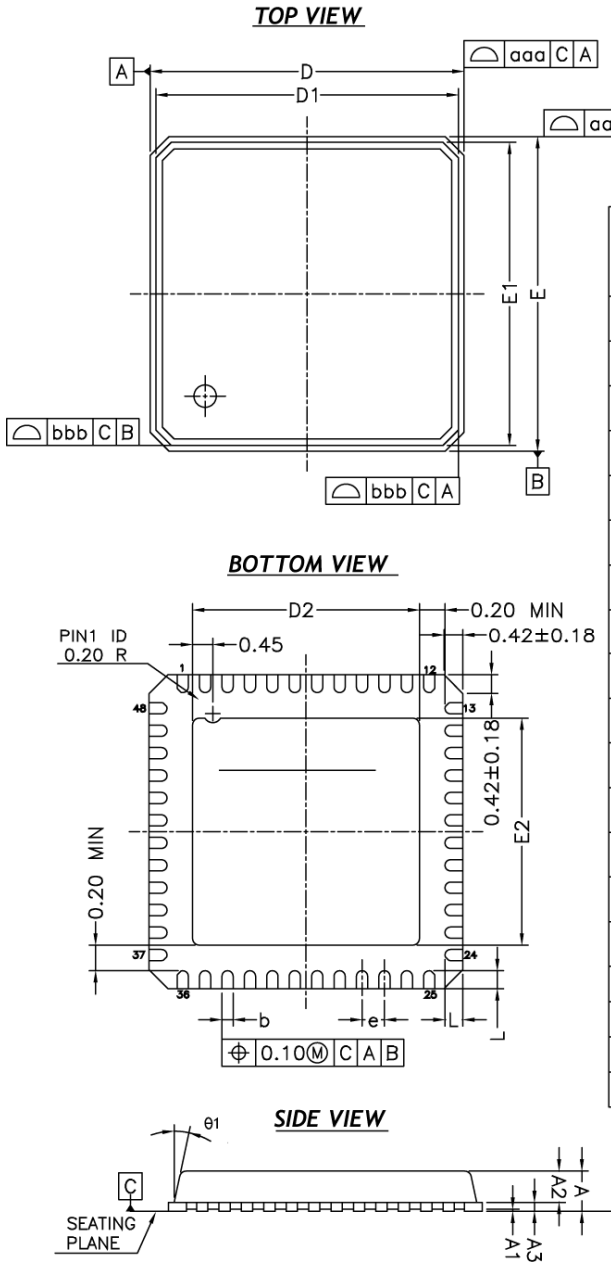


Figure 24 IR Reflow Temperature

9 Package Descriptions



SIMBOL	MILLIMETER			INCH		
	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.
A	-	-	0.90	-	-	0.035
A1	0.00	0.01	0.05	0.00	0.0004	0.02
A2	-	0.65	0.70	-	0.26	0.028
A3	0.20 REF.			0.008 REF.		
b	0.20	0.25	0.32	0.008	0.010	0.013
D	7.00 bsc			0.276 bsc		
D1	6.75 bsc			0.266 bsc		
D2	4.86	5.06	5.26	0.191	0.199	0.207
E	7.00 bsc			0.276 bsc		
E1	6.75 bsc			0.266 bsc		
E2	4.86	5.06	5.26	0.191	0.199	0.207
L	0.30	0.40	0.50	0.012	0.016	0.020
e	0.50 bsc			0.020bsc		
θ1	0°	-	12°	0°	-	12°
R	0.65	-	-	0.028	-	-
TOLERANCES OF FORM AND POSITION						
aaa	0.10			0.004		
bbb	0.10			0.004		
ccc	0.05			0.002		

Figure 25 Package Dimensions

Note:

5. All dimensions are in millimeters.
6. Die thickness allowable is 0.0304 mm MAXMUM (0.012 Inches MAXIMUM)
7. Dimension & tolerances conform to same Y14.5M. -1994.
8. Dimension applies to plated terminal and is measured between 0.20 and 0.25mm from terminal tip.
9. The pin #1 identifier must be placed on the top surface of the package by using indentation mark or other feature of package body.

10. Exact shape and size of this feature is optional.
11. Package warpage max 0.08 mm.
12. Applied for exposed pad and terminals. Exclude embedding part of exposed pad from measuring
13. Applied only to terminals
14. Package corners unless otherwise specified are R0.175+/- 0.025mm

文档版本信息

Version	Date	Descriptions
Ver.1.0	Mar2011	Released with W5200 Launching
Ver. 1.1	13MAR2011	Changed IMR address (0x16 to 0x36) (P.14, P.18) Changed IMR2 address (0x36 to 0x16) (P.14, P.22)
Ver.1.2	22APR2011	Fixed the description of RSV at 1.3 Miscellaneous Signals (P.10) Fixed the values of typical at 7.3 power dissipation (P.77) Added the values of maximum at 7.3 power dissipation (P.77) Fixed the description of RSV at 1.3 Miscellaneous Signals (removed PIN 31, P.10)
Ver. 1.21	2AUG2011	Fixed the description of READ processing at 6.3 Processing of using general SPI Master device (P.73)