iEthernet W5200数据手册

版本1.2.7

☞更多信息，请访问我们的官方网站 http://www.wiznet.co.kr

http://www.iwiznet.cn/

W5200

W5200芯片是一种采用全硬件TCP/ IP协议栈的嵌入式以太网控制器，它能使嵌入式系统通过SPI(串行外设接口)接口轻松地连接到网络。W5200特别适合那些需要使用单片机来实现互联网功能的客户，而这就需要单片机系统具有完整的TCP/ IP协议栈和10/100Mbps以太网网络层（MAC）和物理层（PHY）。

W5200是由已经通过市场考验的全硬件TCP/ IP协议栈、及以太网网络层和物理层的整合而成。其全硬件的TCP/ IP协议栈全程支持TCP、UDP、IPv4、ICMP、ARP、IGMP和PPPoE协议，而且已经连续多年在各种实际应用中得以证明。W5200使用32KB缓存作为其数据通信内存。通过使用W5200，用户只需通过使用一个简单的socket程序就能实现以太网的应用，而不再需要处理一个复杂的以太网控制器了。

SPI(串行外设接口) 提供了轻松与外部MCU连接的接口。W5200支持高达80MHZ的SPI接口间通信。为了降低系统功率的消耗，W5200提供了网络唤醒和休眠模式。W5200收到原始以太网数据包形式的magic packet时将被唤醒。

特点

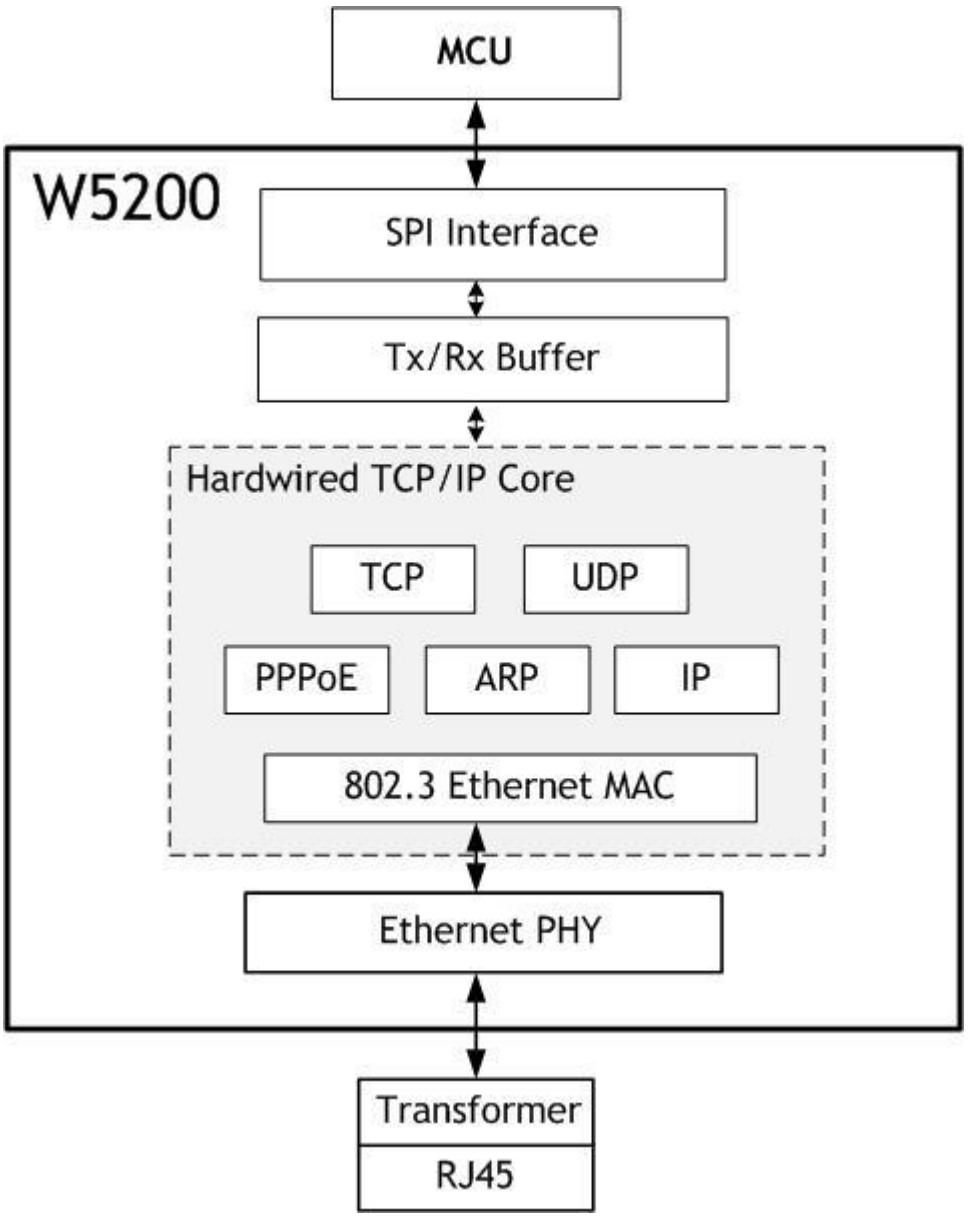
- 支持硬件TCP/IP协议：TCP、UDP、ICMP、IGMP、IPv4、ARP、IGMP、PPPoE和以太网
- 支持8个独立的端口(sockets)
- 极小巧的48 Pin QFN无铅封装
- 提供休眠模式
- 提供网络唤醒
- 支持高速SPI总线(SPI模式0,3)
- 内部32K字节存储器作TX/RX缓存
- 内嵌10/100Mbps以太网物理层
- 支持自动应答 (全双工/半双工模式、10BASET/100BASET)
- 支持自动极性变换(MDI/MDIX)
- 支持ADSL连接(与PAP/CHAP验证模式下，支持PPPOE协议)
- 不支持IP分段
- 3.3V工作电压，I/O口可承受5V电压
- 多种指示灯信号输出(全双工/半双工模式、网络连接和网络速度)

目标应用

W5200非常适合许多嵌入式应用，包括：

- 家庭网络设备：机顶盒、个人录像机、数码媒体适配器
- 串行转以太网:门禁控制、LED显示屏、无线AP继电器等
- 并行转以太网：POS/微型打印机、复印机
- USB转以太网：存储设备、网络打印
- GPIO转以太网：家庭网络传感器
- 安全系统：数字录像机、网络摄像机、信息亭
- 工厂和楼宇自动化控制系统
- 医疗监测设备
- 嵌入式服务器

结构图



目录

插图清单

1.引脚分配

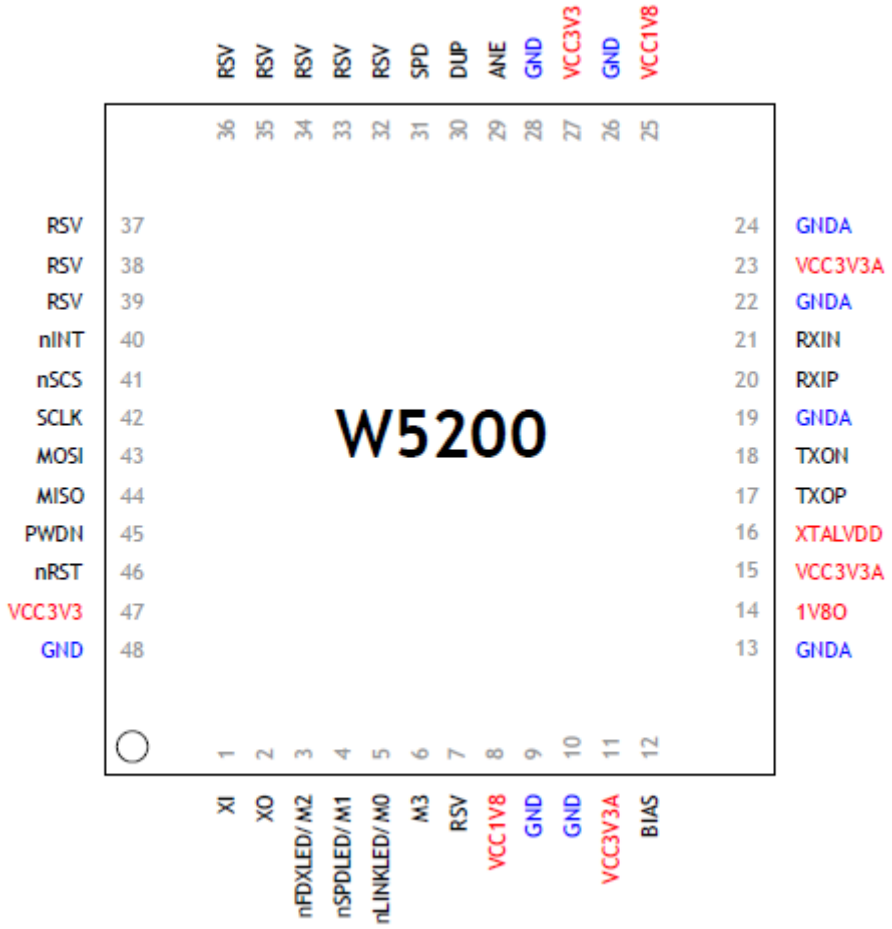


图 1

1.1 微控制器(MCU)接口信号

代号 (Symbol)	类型 (Type)	引脚号码	说明
nRST	I	46	复位(低电平有效) 此引脚为输入低电平，从而使 W5200 初始化或返回初始化。 为了有效复位，低电平时间最少要保持 2us。芯片复位时锁相环上锁时间至少为 150 毫秒。所以，当用户由低置高 nRST 后，勿进行其他操作，静等 150 毫秒为宜。 以便锁相环(PLL)逻辑的失效高电平变得稳定。请参考“7 电子规格”的复位时序
nSCS	I	41	SPI SLAVE 选择(低电平有效) 当使用SPI接口时，此引脚用于SPI被动信号选择引脚

nINT	O	40	中断(低电平有效) 此引脚是W5200与MCU进行socket连接/断开,数据接收超时和WOL(网络唤醒)后的需求指令。对中断寄存器(IR)或Sockte n-th中断寄存器(Sn_IR)的相应位，进行置“1”的写操作可使相应中断标志位清零。该引脚为低电平有效。
SCLK	I	42	SPI时钟(Clock) 当使用SPI接口时，此引脚用于SPI时钟(Clock)的信号引脚
MOSI	I	43	SPI 主输出被动输入 当使用SPI接口时，此引脚用于SPI MISO的信号引脚
MISO	O	44	SPI 主输入被动输出 此引脚用于SPI MISO的信号引脚。
PWDN	I	45	掉电 (高电平有效) (此引脚用于控制w5200处于普通模式还是掉电模式) 低：启用普通模式 高：启用掉电模式

1.2 物理层(PHY)信号

代号 (Symbol)	类型 (Type)	引脚号码	说明
RXIP	I	20	RXIP/RXIN (差分信号) 接收数据时，物理媒介上的差分信号分别对应这里的 RXIP/RXIN。
RXIN	I	21	
TXOP	O	17	TXOP/TXON (差分信号) 发送数据时，物理媒介上的差分信号分别对应这里的 TXOP/TXON。
TXON	O	18	
BIAS	O	12	BIAS 寄存器 连接28.7kΩ±1％的电阻到地线。 请参考“参考原理图”。
ANE	I	29	全双工/半双工应答谈判模式 该引脚用来选择启用/关闭自动应答模式. 低：关闭自动应答谈判模式 高：启用自动应答谈判模式
DUP	I	30	启用全双工模式 该引脚用来选择启用/关闭全双工模式. 低 = 启用半双工模式

			高 = 启用全双工模式 只有在复位期间才能激活此功能。
SPD	I	31	速度模式 该引脚用来选择100M/10M的速度模式 低 = 10M速度模式 高 = 100M速度模式 只有在复位期间才能激活此功能。

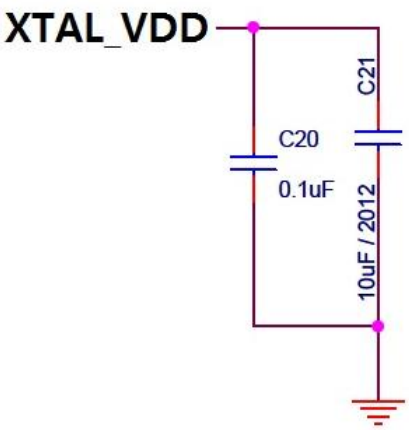
1.3 综合信号

代号 (Symbol)	类型 (Type)	引脚号码	说明
nFDXLED/M2 nSPDLED/M1 nLINKLED/M0	I	3, 4, 5	W5200 模式选择 普通模式: 111 其他测试模式是内部测试模式 只有在复位期间才能激活此功能。
M3	I	6	该引脚应该被上拉(pull-up)。
RSV	-	7,31,32,33,34, 35,36,37,38,39	保留引脚 引脚7应该被上拉(pull-up)。 其他保留引脚(除了引脚7外)应该被下拉(pull-down)或GND。

- 注意: 上拉/下拉电阻 = 40KΩ 到 100KΩ。平常的值为75KΩ.

1.4 电源信号

代号 (Symbol)	类型 (Type)	引脚号码	说明
VCC3V3A	Power	11, 15, 23	3.3V 模拟部分的电源
VCC3V3	Power	27, 47	3.3V 数字部分的电源
VCC1V8	Power	8, 25	1.8V 数字部分的电源
GNDA	Ground	13, 19, 22, 24	模拟接地
GND	Ground	9, 10, 26, 28, 48	数字接地
1V8O	O	14	1.8V的稳压输出电压 内部稳压电源创建的1.8V/200mA电源是用于内核运行功率(VCC1V8)。 一定要确定在1V8O和GND之间连接钽电容器作为输

			出频率补偿，并选择性的连接0.1μF电容为高频噪音去耦 ***注意：1V8O是W5200内核运作的电源。它不应该被连接到其它设备的电源。**
XTALVDD	I	16	 **图 2 XTAL_VDD参考电路图** 连接一个10.1uF的电容且接地。 ※请参考'W5200E01-M3 的参考电路图'

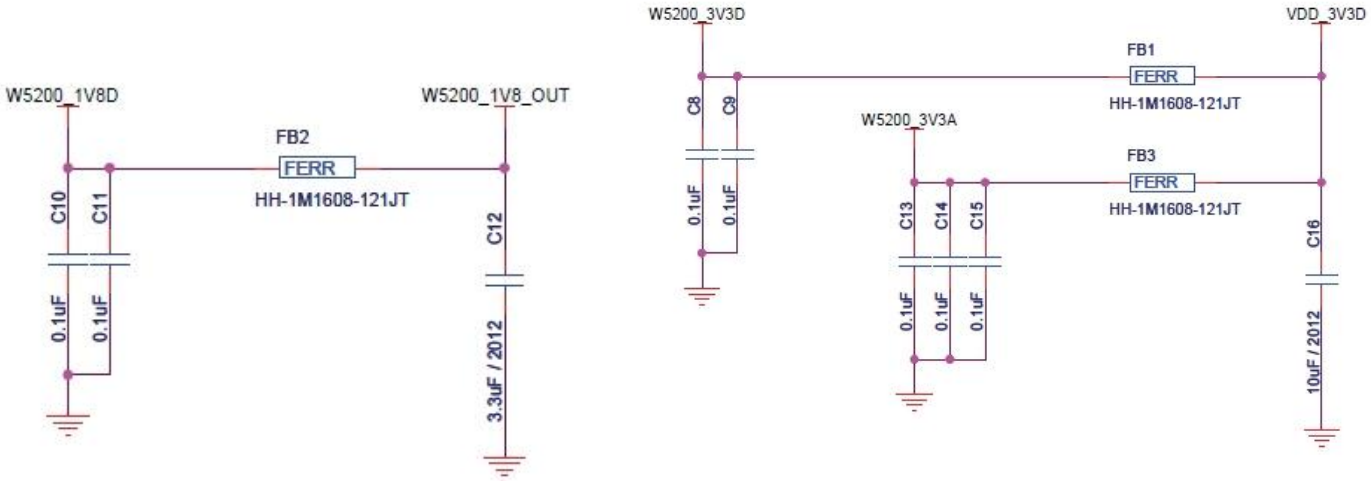


图 3 电源设计

推荐的电源设计:

1. **去耦电容尽可能靠近W5200。**
2. **尽可能保证地线层足够宽。**
3. **如果地线层宽度足够，具有独立的模拟地线层和数字地线层是很好的做法。**
4. **如果地线层不够宽，那么只能将模拟和数字地线层设计为一个单一的地线层，而不是将它们分开。**

1.5 时钟信号

代号 (Symbol)	类型 (Type)	引脚号码	说明
XI	I	1	25MHz的晶振输入/输出。利用25MHz晶振和振荡器来连接这些引脚。
XO	O	2	

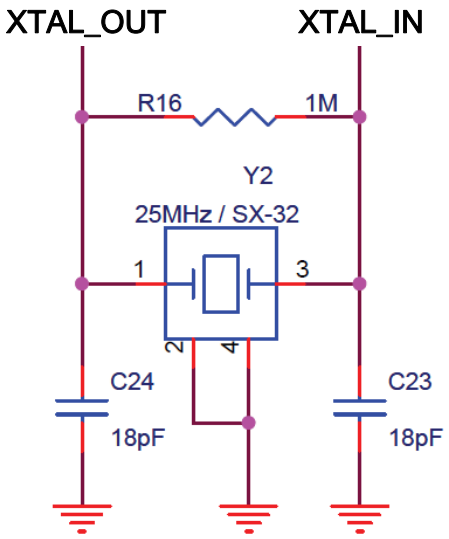


图 4 晶振体参考原理图

1.6 LED 信号

代号 (Symbol)	类型 (Type)	引脚号码	说明
nFDXLED/M2	O	3	全双工/冲突**LED** 低: 全双工 高: 半双工.
nSPDLED/M1	O	4	**LED**链接速度 低: 100Mbps 高: 10Mbps
nLINKLED/M0	O	5	**LED**链接 低:物理链接状态良好(10/100M) 高: 物理连接失败 闪烁: 有数据被发送获接收时

2 内存图(Memory Map)

W5200是由通用寄存器、Socket寄存器、TX的内存和RX的内存组成。如下图所示:

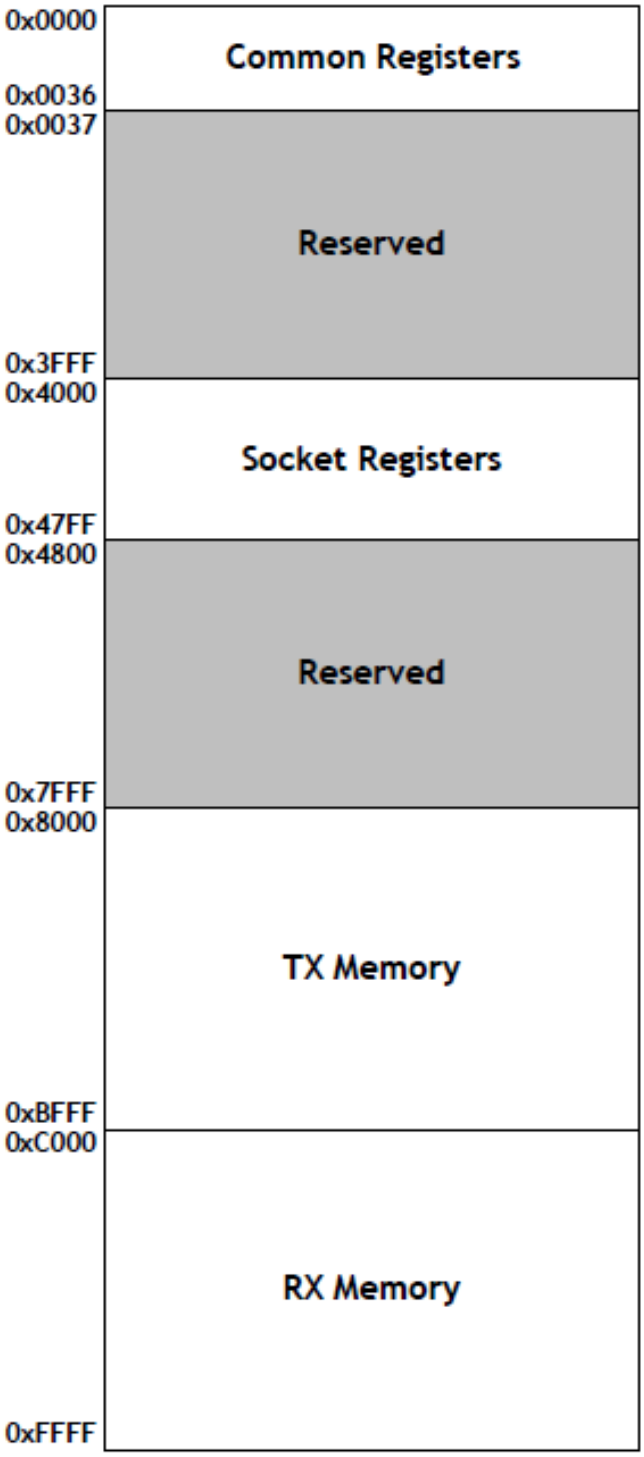


图 5 W5200内存图

3 W5200 寄存器

3.1 通用寄存器

地址	寄存器
0x0000	模式 (MR)
0x0001	网关地址 (GAR0)
0x0002	(GAR1)
0x0003	(GAR2)
0□0004	(GAR3)
0x0005	子网掩码地址 (SUBR0)
0x0006	(SUBR1)
0x0007	(SUBR2)
0x0008	(SUBR3)
0x0009	源MAC地址 (SHAR0)
0x000A	(SHAR1)
0x000B	(SHAR2)
0x000C	(SHAR3)
0x000D	(SHAR4)
0x000E	(SHAR5)
0x000F	源IP地址 (SIPR0)
0x0010	(SIPR1)
0x0011	(SIPR2)
0x0012	(SIPR3)
0x0013	保留
0x0014	
0x0015	中断 (IR)
0x0016	中断掩码 (IMR)
0x0017	重试时间 (RTR0)
0x0018	(RTR1)
0x0019	重试计数(RCR)
0x001A	保留
0x001B	

地址	寄存器
0x001C	PPPoE认证类型 (PATR0)
0x001D	(PATR1)
0x001E	PPPoE认证算法 (PPPALGO)
0x001F	芯片版本(VERSIONR)
0x0020 ~ 0x0027	保留
0x0028	PPP LCP的请求定时器(PTIMER)
0x0029	PPP LCP Magic number (PMAGIC)
0x002A ~ 0x002F	保留
0x0030	中断低电平计时器 (INTLEVEL0)
0x0031	(INTLEVEL1)
0x0032 ~ 0x0033	保留
0x0034	Socket 中断(IR2)
0x0035	PHY 状态(PSTATUS)
0x0036	Socket 中断掩码 (IMR2)

3.2 Socket寄存器

注意 ：n 是socket的数目 (0, 1, 2, 3, 4, 5, 6, 7)

地址	寄存器
0x4n00	Socket n 的模式 (Sn_MR)
0x4n01	Socket n 的命令(Sn_CR)
0x4n02	Socket n 的中断(Sn_IR)
0x4n03	Socket n 的状态(Sn_SR)
0x4n04	Socket n 的源端口 (SN_PORT0)
0x4n05	(SN_PORT1)
0x4n06	Socket n 的目的地MAC地址 (Sn_DHAR0)
0x4n07	(Sn_DHAR1)
0x4n08	(Sn_DHAR2)
0x4n09	(Sn_DHAR3)
0x4n0A	(Sn_DHAR4)
0x4n0B	(Sn_DHAR5)
0x4n0C	Socket 0 的目的地IP地址 (Sn_DIPR0)
0x4n0D	(Sn_DIPR1)
0x4n0E	(Sn_DIPR2)
0x4n0F	(Sn_DIPR3)
0x4n10	Socket 0 的目的地端口 (Sn_DPORT0)
0x4n11	(Sn_DPORT1)
0x4n12	Socket 0 的最大段字节大小 (Sn_MSSR0)
0x4n13	(Sn_MSSR1)
0x4n14	Socket 0 在IP Raw模式的协议 (Sn_PROTO)
0x4n15	Socket n IP TOS (Sn_TOS)
0x4n16	Socket n IP TTL (Sn_TTL)
0x4n17 ~ 0x4n1D	保留

地址	寄存器
0x4n1E	接收内存大小 (Sn_RXMEM_SIZE)
0x4n1F	传送内存大小 (Sn_TXMEM_SIZE)
0x4n20	Socket 0 TX 自由大小 (Sn_TX_FSR0)
0x4n21	(Sn_TX□FSR1)
0x4n22	Socket 0 TX 读指针 (Sn_TX_RD0)
0x4n23	(Sn_TX_RD1)
0x4n24	Socket 0 TX 写指针 (Sn_TX_WR0)
0x4n25	(Sn_TX_WR1)
0x4n26	Socket 0 RX 接收大小 (Sn_RX_RSR0)
0x4n27	(Sn_RX_R□R1)
0x4n28	Socket 0 RX 读指针 (Sn_RX_RD0)
0x4n29	(Sn_RX_RD1)
0x4n2A	Socket 0 RX 写指针 (Sn_RX_WR0)
0x4n2B	(Sn_RX_WR1)
0x4n2C	Socket 的中断掩码 (Sn_IMR)
0x4n2D	IP头地址的分段偏移 (Sn_FRAG0)
0x4n2E	(Sn_FRAG1)
0x4n30 ~ 0x4nFF	保留

4 寄存器说明

4.1 通用寄存器

MR (模式寄存器) [R/W] [0x0000] [0x00]

该寄存器用于 S/W 复位', ping block模式和PPPoE模式。.

7	6	5	4	3	2	1	0
RST			PB	PPPoE			

位(Bit)	代号 (Symbol)	说明
7	RST	**S/W 复位** 如果该位(bit)为'1', 内部寄存器将被初始化。它会在复位后自动清零。
6	Reserved	保留
5	Reserved	保留
4	PB	**Ping Block 模式** 0: 关闭Ping block 1: 启用Ping block 如果该位(bit)设置为'1', ping请求时就没有响应。
3	PPPoE	**PPPoE 模式** 0: 关闭PPPoE 模式 1: 启用PPPoE 模式 当在没有路由器或具有路由功能的设备情况下, 使用ADSL时,你应该设置该位(bit)为'1'来连接到ADSL服务器。如需详细资料, 请参考应用笔记, "如何连接ADSL"。
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]

该寄存器用来设置默认网关地址。

例) "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]

该寄存器用来设置子网掩码地址。

例) “255.255.255.0”

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

SHAR (源MAC地址寄存器) [R/W] [0x0009 – 0x000E] [0x00]

该寄存器用来设置源MAC地址。

Ex) 例如: “00.08.DC.01.02.03”

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

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

该寄存器用来设置源IP地址。

例) “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]

CPU通过访问该寄存器获得产生中断的来源。任何中断源都可以被中断屏蔽寄存器（IMR）进行位屏蔽。当任何一个未屏蔽的中断位为“1”,/INT的信号将保持低电平。只有当所有未屏蔽的中断位为0，/INT才恢复高电平。（也就是说，/INT为低电平意味着W5200有中断产生，高电平意味着无中断产生或者中断被屏蔽了）

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

位 (Bit)	代号 (Symbol)	说明
7	CONFLICT	**IP冲突** 当对一个与本机IP 地址相同的IP 地址作ARP 请求时，该位被置“1”。对该位写”1”可清0
6	Reserved	保留
5	PPPoE	**PPPoE 连接关闭** 在PPPoE 模式，如果PPPoE 连接被关闭，该位置“1”。对该位写”1”可清0
4	Reserved	保留
3	Reserved	保留
2	Reserved	保留

1	Reserved	保留
0	Reserved	保留

IMR (中断掩码寄存器) [R/W] [0x0016] [0x00]

中断屏蔽寄存器（IMR）用来屏蔽中断源。每个中断屏蔽位对应中断寄存器（IR）中的一个位。如果中断屏蔽位被置“1”时，无论何时IR对应的位也置”1”，中断即会产生。而当IMR中屏蔽位被清“0”，即使对应的IR中断位置“1”，也不会产生中断。（简单来讲，IMR中“1”就是允许中断的意思，“0”就是屏蔽中断的意思。）

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

位 (Bit)	代号 (Symbol)	说明
7	IM_IR7	启用IP冲突
6	Reserved	保留
5	IM_IR5	启用PPPoE关闭
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]

该寄存器用来设置溢出的时间值。每一单位数值为100 微秒。初始化时值设为2000（0x07D0），即相当于200 毫秒

例) 当超时周期被设置为400ms时，RTR = 400ms /100us = 4000(0x0FA0)

0x0017	0x0018
0x0F	0xA0

如果没有来自远程对等连接点的CONNECT、DISCON、CLOSE、SEND、SEND_MAC和SEND_KEEP命令回应，或回应的命令被延迟时，重新传送就会发生。

RCR(重试计数寄存器) (Retry Count Register) [R/W] [0xFE0019] [0x08]

该寄存器是设置重新传送的次数。当重新传送发生超过‘RCR+1’次，超时中断就会置‘1’。（中断寄存器（Sn_IR）的 ‘中断’位(‘TIMEOUT’ bit)设置为'1'）。

假如在TCP通讯中，Sn_SR(Socket n-th的状态寄存器)的数值会变为'SOCK_CLOSED'与此同时Sn_IR(Socket n-th的中断寄存器)（TIMEOUT）会置'1'。如果不是TCP通讯，就只有Sn_IR（TIMEOUT）会置 ‘1’。

例) RCR = 0x0007

0x0019
0x07

W5200的超时可以用RTR和RCR来配置。W5200的超时包括地址解析协议(ARP)和TCP重新传送超时。

在ARP的重新传送超时（请参阅 RFC 826 http://www.ietf.org/rfc.html），W5200会自动发送ARP请求去对方(peer)的IP地址，从而获取MAC地址信息（IP、UDP或TCP用于通信）。至于等待对方(peer)的ARP 响应方面，如在RTR中设置了重新传送时间时，对方(peer)的ARP没有响应，超时发生和ARP将会请求重新传送。一直重复此步骤达'RCR+1'次。

如果在ARP重复请求重新传送次数达到'RCR+1'次时，仍然没有得到ARP响应，就会触发最终超时中断，Sn_IR(TIMEOUT)会变为'1'。

ARP请求的最终超时值(ARP_{TO})如下:

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

在RTR和RCR设置了的时间期间，发生TCP数据包重新传送超时，W5200就会发送 TCP 数据包(SYN、 FIN、RST、数据包）和等待确认（ACK）。如果没有对方(peer)的ACK响应，就会触发超时且TCP数据包（较早前传送的）会重新传送。直到重新传送'RCR+1'次。即使TCP数据包重新传送'RCR+1'次，如果对方(peer)仍然没有的ACK回应，就会触发最终超时，Sn_SR会变为 'SOCK_CLOSED，且同一时间Sn_IR(TIMEOUT)='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: 重试计数，0 <= N <= M

M: 最少数值当 $RTR \times 2^{(M+1)} > 65535$ 和 0 <= M <= RCR

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 + 0xFA0 + 0xFA00) X 0.1ms

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

= 318000 X 0.1ms = 31.8s

PART(PPPoE模式的身份验证模式) [R] [0x001C-0x001D] [0x0000]

该寄存器显示已经被PPPoE 服务器识别的身份认证类型。W5200支持PAP和CHAP两种类型的身份验证方法。

数值	身份验证类型
0xC023	PAP
0xC223	CHAP

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

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

VERSIONR (W5200芯片版本寄存器）[R][0x001F][0x03]

该寄存器是W5200芯片版本寄存器。

PTIMER（PPP 连接控制协议请求定时寄存器）[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) 如果 PTIMER是 200，

200 * 25(ms) = 5000(ms) = 5 seconds

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

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

INTLEVEL（低电平中断定时器寄存器） [R/W][0x0030 - 0x0031][0x0000]

该寄存器用于设置中断生效等待的时间(I_{AWT})。它配置nINT低等待时间直到下一个中断生效。

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

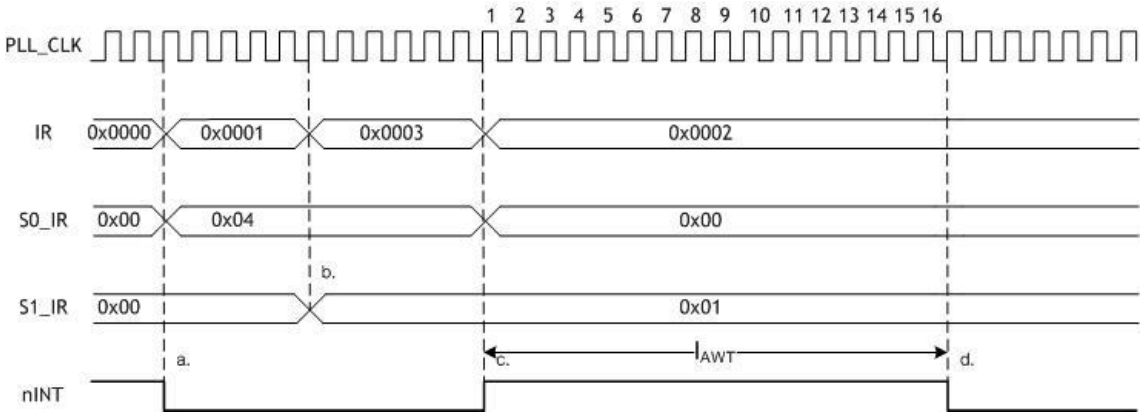


图 6 INTLEVEL的时序

A. 对Socket0而言，当触发接收中断（即：S0_IR(3) = ‘1’），那么相应的IR2位就会被置1（即：IR(S0_IR) = ‘1’）.一旦IR2的某位被置1了，nINT就将变为低电平。B，C同理。

B. 在Socket 1中，触发连接中断（S1_IR（0）='1'）和相应的位(bit)设置为'1' （IR（S1_IR）='1'）。

C.主机清零S0_IR（S0_IR=0x00）则相应的IR位(bit)也会自动被清零(IR(S0_IR) = ‘0’)。nINT信号变为高电平。

D.虽然S0_IR被清零了，但是由于socket1的中断还存在，所以IR2的值不为0.既然IR2的值不为0，那么nINT引脚就应该变为低电平。那么nINT从高电平变为低电平需要多长时间呢？这就要看INTLEVEL register的设置情况了。如果INTLEVEL register的值是0x000F的话，nINT将在 IAWT(16 PLL_CLK)后变为低电平。

IR2(W5200 SOCKET中断寄存器)[R/W][0x0034][0x00]

IR2这个寄存器用来通知主机W5200中socket发生中断。如果任何中断发生时，IR2相关的位(bit)会被设置为‘1’。当相关的掩码位(bit)被设置为‘1’时，nINT信号为低电平。nINT会保持低电平直到所有Sn_IR的位(bit) 被设置为‘0’时,它才会变为高电平。

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 (Bit)	代号 (Symbol)	说明
7	S7_INT	当SOCKET 7有一个中断发生，它(S7_INT)就变成'1'。与该中断的信息相对应的为S7_IR。当S7_IR被主机清零为0x00时,该位(bit)会被自动清零。
6	S6_INT	当SOCKET 6有一个中断发生，它(S6_INT)就变成'1'。与该中断的信息相对应的为S6_IR。当S6_IR被主机清零为0x00时,该位(bit)会被自动清零。
5	S5_INT	当SOCKET 5有一个中断发生，它(S5_INT)就变成'1'。与该中断的信息相对应的为S5_IR。当S5_IR被主机清零为0x00时,该位(bit)会被自动清零。
4	S4_INT	当SOCKET 4有一个中断发生，它(S4_INT)就变成'1'。与该中断的信息相对应的为S4_IR。当S4_IR被主机清零为0x00时,该位(bit)会被自动清零。
3	S3_INT	当SOCKET 3有一个中断发生，它(S3_INT)就变成'1'。与该中断的信息相对应的为S3_IR。当S3_IR被主机清零为0x00时,该位(bit)会被自动清零。
2	S2_INT	当SOCKET 2有一个中断发生，它(S2_INT)就变成'1'。与该中断的信息相对应的为S2_IR。当S2_IR被主机清零为0x00时,该位(bit)会被自动清零。

1	S1_INT	当SOCKET 1有一个中断发生，它(S1_INT)就变成'1'。与该中断的信息相对应的为S1_IR。当S1_IR被主机清零为0x00时,该位(bit)会被自动清零。
0	S0_INT	当SOCKET 0有一个中断发生，它(S0_INT)就变成'1'。与该中断的信息相对应的为S0_IR。当S0_IR被主机清零为0x00时,该位(bit)会被自动清零。

PHYSTATUS(W5200 物理层状态寄存器)[R/W][0x17]

PHYSTATUS是用来指示W5200物理层状态的寄存器。

位 (Bit)	代号 (Symbol)	说明
7	Reserved	保留
6	Reserved	保留
5	LINK	**连接状态寄存器[只读]** 该寄存器用来指示连接状态。 0:无效连接状态 1:有效连接状态
4	Reserved	保留
3	POWERDOWN	**物理层的掉电模式[只读]** 该寄存器用来指示电源掉电模式的状态 0: 关闭电源掉电模式（运作正常模式） 1: 启用电源掉电模式
2	Reserved	保留
1	Reserved	保留
0	Reserved	保留

IMR2(中断掩码寄存器2)[R/W][0x0036][0x00]

该中断掩码寄存器是用来掩码中断。每个中断掩码位(bit)对应于中断寄存器2（IR2）。当中断掩码位被设置为“1”且中断寄存器2的相关位也置1时，中断发生。如果任何位(bit)在中断掩码寄存器(IMR)设置为’0'时，就不会发生中断。

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 (Bit)	代号 (Symbol)	说明
7	S7_INT	IR(S7_INT)中断掩码
6	S6_INT	IR(S6_INT) 中断掩码
5	S5_INT	IR(S5_INT) 中断掩码

4	S4_INT	IR(S4_INT) 中断掩码
3	S3_INT	IR(S3_INT) 中断掩码
2	S2_INT	IR(S2_INT) 中断掩码
1	S1_INT	IR(S1_INT) 中断掩码
0	S0_INT	IR(S0_INT) 中断掩码

4.2 Socket 寄存器

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

该寄存器用于配置所有SOCKET的选项或协议类型

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

位(Bit)	代号 (Symbol)	说明
7	MULTI	**广播(Multicasting)** 0: 关闭广播 1: 启用广播 这个仅适用于UDP 当使用广播时，用户使用打开(OPEN)命令前，要将广播组地址和广播组端口分别写入SOCKET n-th的目的地IP和目的地端口寄存器。
6	MF	**MAC过滤器** 0: 关闭 MAC 过滤 1: 启用 MAC 过滤 这个仅适用于 MACRAW (P3~P0: “0100”). 当该位(bit)被设置为'1'时，W5200可以接收属于自己或广播的数据包。当该位(bit)被设置为'0'时，W5200可以接收所有以太网数据包。当使用混合的TCP/ IP协议栈时，则建议设置为'1’以减少主机的接收开销。
5	ND/MC	**使用无延迟的ACK** 0: 关闭无延迟的ACK选项 1: 启用无延迟的ACK选项， 这个仅适用于 TCP (P3-P0: “0001”) 如果此位(bit)被设置为 '1'时，当从一个对方(peer)接收到数据包后，ACK数据包就会立即传送。如果该位(bit)被清零时，那么只会根据内部超时机制来决定是否发送ACK数据包。

[1] *n*是 Socket n-th 的数目 (0, 1, 2, 3, 4, 5, 6, 7).

[2] [读/写] [socket 0地址, socket 1地址, socket 2地址, socket 3地址, socket 4地址, socket 5地址,socket 6地址, socket 7地址] [复位数值]

<table>
<tr><td></td><td></td><td>广播(Multicast)
0: 使用IGMP版本2
1: 使用IGMP版本1
当启用MULTI位(bit)和UDP模式使用中(P3-P0: “0010”)时，此位(bit)才有效。此外，广播可以用来在IGMP消息发送版本号码(version
number)。例如: 加入/离开 /向广播组报告</td></tr>
<tr><td>4</td><td>Reserved</td><td>保留</td></tr>
<tr><td>3</td><td>P3</td><td rowspan="4">协议栈
设置相应的SOCKET为TCP、UDP或IP RAW模式<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>Closed</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>* S0_MR_MACRAW和S0_MR_PPPoE仅适用于SOCKET 0。<table><tr><th>P3</th><th>P2</th><th>P1</th><th>P0</th><th>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仅适用于SOCKET 0。
S0_MR_PPPoE暂时用于PPPoE服务器连接/终止。建立连接后，它可以被用作其他协议。</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>

Sn_CR (Socket n-th 命令寄存器) [R/W] [0x4001+0x0n00] [0x00]

这是用来设置Socket n-th的命令如OPEN、CLOSE、CONNECT、LISTEN、END和RECEIVE。

经W5200识别这一命令后，Sn_CR寄存器会自动清零为 0x00。尽管Sn_CR被清零为 0x00，但命令仍在处理中。为了验证该命令是否完成，请检查Sn_IR或Sn_SR寄存器。

数值 (Value)	代号 (Symbol)	说明
0x01	OPEN	按照Sn_MR(P3:P0)的协议选择来初始化和打开(open) Socket n-th。下表显示了Sn_SR和Sn_MR的对应值 Sn_MR(P3:P0) / Sn_SR Sn_MR_CLOSE (0x00) / - Sn_MR_TCP (0x01) / SOCK_INIT (0x13) Sn_MR_UDP (0x02) / SOCK_UDP (0x22) Sn_MR_IPRAW (0x03) / SOCK_IPRAW (0x32) S0_MR_MACRAW (0x04) / SOCK_MACRAW (0x42) S0_MR_PPPoE (0x05) / SOCK_PPPoE (0x5F)
0x02	LISTEN	这是只适用于TCP模式(Sn_MR(P3:P0) = Sn_MR_TCP)。在这种模式下，Socket n-th被配置为一个TCP服务器,它是等待“TCP客户端”的连接请求（SYN数据包）。该Sn_SR寄存器由SOCK_INIT改变为SOCK_LISTEN。 当一个客户端的连接请求成功后，该Sn_SR寄存器由SOCK_LISTEN改变为SOCK_ESTABLIESHE，与此同时Sn_IR(0)会变为'1'。另一方面，当连接失败时，Sn_IR(3)被设置为'1'，Sn_SR改变为SOCK_CLOSED（SYN/ACK数据包无法转移）。 当一个连接请求时，如果目的地的TCP客户端端口不存在的话，W5200将发送一个RST数据包并且Sn_SR保持不变。
0x04	CONNECT	此模式只适用于TCP模式和运作Socket n-th作为TCP客户端。通过与存储在目的地址寄存器和端口号寄存器中的IP地址和端口号进行连接，一个连接请求被发送到TCP服务器。当一个客户端的连接请求成功后，Sn_SR寄存器改为 SOCK_ESTABLIESHED，Sn_IR(0)会变为'1'。 以下三种情况意味着连接请求失败： 1. ARPTO发生(Sn_IR(s)=‘1’)。因为目的地的MAC地址不能通过ARP过程中获取。 2. 当没有收到SYN/ACK数据包，而引起TCPTO(Sn_IR(3))被设置为'1'时。 3. 当RST数据包而不是SYN/ACK数据包被接收时。 以上三种情况下，Sn_SR会改为SOCK_CLOSED。
0x08	DISCON	只适用于TCP模式 不论“TCP服务器”或“TCP客户端”，都要断开。

		主动关闭：它传输断开请求（FIN数据包）到所连接的对方(peer)。 被动关闭：当从对方(peer)收到FIN数据包时，回复一个FIN数据包到 对方(peer)。 当FIN/ACK数据包被接收时，Sn_SR改为SOCK_CLOSED。 当断开请求没有收到ACK时，TCPTO就会发生(Sn_IR(3)='1')，Sn_SR改为 SOCK_CLOSED。 如果用CLOSE命令来代替disconnect命令的话，只有Sn_SR变为SOCK_CLOSED. FIN/ACK这种断开机制不被执行。如果当沟通期间从一个对方(peer)接收到一个RST数据包，Sn_SR是无条件地更改为SOCK_CLOSED。
0x10	CLOSE	关闭 Socket n-th. Sn_SR改为SOCK_CLOSED。
0x20	SEND	发送(SEND)传送所有发送(TX)内存的缓冲数据。欲想了解更多详情，请参阅Socket n-th 发送(TX)自由尺寸寄存器(Sn_TX_FSR0)，Socketn-th 发送(TX)写指针寄存器(Sn_TX_WR0)和Socket n-th发送(TX)读指针寄存器(Sn_TX_RD0)。
0x21	SEND_MAC	只适用于UDP模式 基本操作是与发送(SEND)相同的。发送(SEND)操作通常需要通过ARP（地址解析协议）过程中得到目的地MAC地址。而SEND_MAC使用的Socket n-th目的地MAC地址(Sn_DHAR0)是由用户选择而不是通过ARP过程得到的。
0x22	SEND_KEEP	只适用于TCP模式 (我们常说的keep alive机制。注意：要想让keep alive机制正常工作，W5200必须先向对方（peer）发送过数据。这是keep alive协议规定的。) 它检查发送1字节数据时的连接状态。如果连接没有对方(peer)回应或终止，就会发生超时中断。
0x40	RECV	接收(RECV)的过程是用接收(RX)读指针寄存器(Sn_RX_RD)来接收数据的。 欲想了解更多详情，请参阅5.2.1.1服务器模式接收过程与Socket n-th接收(RX)的接收尺寸寄存器(Sn_RX_RSR0)，Socket n-th接收(RX)写指针寄存器(Sn_RX_WR)和Socket n-th接收(RX)读指针寄存器(Sn_RX_RD)。

以下命令只适用于SOCKET 0 和 S0_MR(P3:P0) = S0_MR_PPPoE。更多细节请参考“如何使用ADSL”。

数值 (Value)	代号 (Symbol)	说明
0x23	PCON	通过发送PPPoE发现数据包来开始PPPoE连接
0x24	PDISCON	关闭PPPoE连接
0x25	PCR	在每个阶段(phase)，它传输REQ信息
0x26	PCN	在每个阶段(phase),它传输NAK信息
0x27	PCJ	在每个阶段(phase),它传输拒绝(REJECT)信息

Sn_IR (Socket n-th 中断寄存器) [R] [0x4002+0x0n00] [0x00]

Sn_IR寄存器用于提供给Socket n-th中断类型信息，如建立(Establishment)、终止(Termination)、接收数据(Receiving data)和超时(Timeout)。当触发一个中断即Sn_IMR的掩码位是'1'的时候，Sn_IR的中断位将会变成‘1’。

如果想把Sn_IR位清零的话，主机应该将该位置‘1’。以当所有Sn_IR的位被清零后(‘0’)，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	代号	说明
7	PRECV	**Sn_IR(PRECV) 中断掩码** 只会在‘SOCKET=0’和‘S0_MR(P3:P0) = S0_MR_PPPoE’时才会有效 当选项不支持接收的时候，为‘PPP RECEVICE中断’
6	PFAIL	**Sn_IR(PFAIL) 中断掩码** 只会在‘SOCKET=0’和‘S0_MR(P3:P0) = S0_MR_PPPoE’时方会有效 当 PAP验证失败的时候，为‘PPP FAIL中断’
5	PNEXT	**Sn_IR(PNEXT) 中断掩码** 只会在‘SOCKET=0’和‘S0_MR(P3:P0) = S0_MR_PPPoE’时方会有效 当位相在ADSL连接过程中改变，为‘PPP NEXT PHASE中断’
4	SEND_OK	**Sn_IR(SENDOK)中断掩码** 当SEND命令完成时，为‘SEND OK中断’
3	TIMEOUT	**Sn_IR(TIMEOUT)中断掩码** 当ARP $_{TO}$或TCP$_{TO}$发生时，为‘TIMEOUT中断’
2	RECV	**Sn_IR(RECV)中断掩码** 每当数据包从一个节点接收时，为‘RECEIVE中断’
1	DISCON	**Sn_IR(DISCON) 中断掩码** 当FIN/ACK数据包从一个节点接收时，为‘DISCONNECT中断’
0	CON	**Sn_IR(CON) 中断掩码** 当与一个节点成功连接时，为‘CONNECT中断’

Sn_SR (Socket n-th状态寄存器) [R] [0x4003+0x0n00] [0x00]

该寄存器提供 Socket n-th 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-th 的状态。

数值	代号	说明
0x00	SOCK_CLOSED	这是SOCKETn的资源被释放的状态。不管以前的数值，当执行DISCON或CLOSE的命令，或ARP_{TO}或TCP_{TO}发生时，它的数值将会更改为 SOCK_CLOSED。
0x13	SOCK_INIT	如果Sn_MR设置为TCP和OPEN命令是给Sn_CR的话,它将会显示。 而当Sn_MR(P3:P0)是Sn_MR_TCP和执行OPEN的命令时,它将被更改为SOCK_INIT。这是建立TCP连接的第一步。 它可以在“TCP SERVER“模式下进行LISTEN的命令和“TCP客户端” 下进行CONNECT的命令。 这是 SOCKETn 以“TCP SERVER”进行运作下，等待从“TCP CLIENT”的连接请求（SYN packet）的状态。
0x14	SOCK_LISTEN	SOCKET n-th 以“TCP SERVER”进行运作下，等待从“TCP CLIENT”的连接请求（SYN packet）。 当使用“LISTEN”命令时，将转换成SOCK_LISTEN状态。 当成功建立连接后，SOCKET的状态会从 SOCK_LISTEN转到SOCK_ESTABLISHED。但是，如果连接失败，将出现TCP_{TO}(Sn_IR(TIME_OUT) = '1') 和状态将转换为SOCK_CLOSED。
0x17	SOCK_ESTABLISHED	它会在成功建立连接后显示。当“TCP CLIENT”里的SYN数据包成功在SOCK_LISTEN中处理，或者是成功执行CONNECTS 命令的时候，它的状态将会转成SOCK_ESTABLISHED。在此状态，数据包是可供传送的，也就是说SEND或是RECV的命令是可以使用。
0x1C	SOCK_CLOSE_WAIT	这是当收到对方请求断开连接的一个状态。即使TCP连接是半关闭的，它亦可以传送数据包。所以，如想完成整个TCP的断线程序，必须执行DISCON命令。 对于没有经过中断程序而把SOCKETn关闭，应当执行CLOSE命令。
0x22	SOCK_UDP	这是把 SOCKETn 开启作为 UDP 模式的一个状态。当 Sn_MR（P3:P0）是 Sn_MR_UDP 和 OPEN 命令被执行时，它将被更改为 SOCK_UDP。不同于 TCP 模式，在这个模式下，数据包是可以在无连接的情况下继续传送。
0x32	SOCK_IPRAW	这个SOCKET会在IPRAW模式下打开。当 Sn_MR（P3:P0）是Sn_MR_IPRAW和OPEN命令被执行时，它的状态将被更

		改为SOCK_IPRAW。而IP数据包是可以在无连接的情况下继续传送(类似UDP模式)。
0x42	SOCK_MACRAW	如果S0_CR= OPEN 和 S0_MR (P3:P0) = S0_MR_MACRAW 的时候，它将被更改为 SOCK_MACRAW。 MACRAW数据包(以太网帧)是可以 像UDP模式般传送。
0x5F	SOCK_PPPOE	这是把 SOCKET0 开启作为 PPPoE 模式的一个状态。如果 S0_CR= OPEN 和 S0_MR (P3:P0) = S0_MR_PPPoE 的时候，它将被更改为 SOCK_PPPoE。这是暂时用于 PPPoE 连接。

以下是转换状态的过程。

数值	代号	说明
0x15	SOCK_SYNSENT	此状态表示一个连接请求(SYN数据包)已发送到“TCP SERVER”。此状态显示利用由SOCK_INIT到SOCK_ESTABLISHED的CONNECT命令的一个转换过程。在此状态下，如果收到由“TCP SERVER”的连接授权(SYN/ACK数据包)，它会自动更改为 SOCK_ ESTBLISHED。如果在TCPTO(Sn_IR(超时)='1') 出现之前，还没有收到“TCP SERVER” 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 CLIENT”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 CLIENT”，它就会自动更改为SOCK_ESTABLISHED。如果失败就会触发TCPTO(Sn_IR(超时)='1')，并更改状态为 SOCK_CLOSED。
0x18	SOCK_FIN_WAIT	这些状况表示SOCKET n-th已关闭。这是对断线程序中的“主动关闭”或“被动关闭”时的一个观察。当断线程序成功完成或TCPTO(Sn_IR(超时)='1')发生时，它便会更改为SOCK_CLOSED。
0x1A	SOCK_CLOSING	
0X1B	SOCK_TIME_WAIT	
0X1D	SOCK_LAST_ACK	
0x01	SOCK_ARP	此状态表示已发送ARP-request以获取目标硬件的地址。这是对当执行SEND命令中的SOCK_UDP或SOCK_IPRAW时，或执行CONNECT命令中的SOCK_INIT时的一个监视 。 如果成功从目的地获取目标硬件的地址(当收到ARP-response)，它便会更改为SOCK_UDP、SOCK_IPRAW或是SOCK_SYNSENT。 如果失败，ARPTO(Sn_IR(超时)='1')便会发生。如发生在UDP或IPRAW模式，它便会回到以前的状态(SOCK_UDP或SOCK_IPRAW) 。如发生在TCP模式，它便会转到 SOCK_CLOSED。 对比 - 当之前和目前的Sn_DIPR值是不同的时候，ARP程序

		便会在SOCK_UDP或SOCK_IPRAW工作。如果之前和目前的Sn_DIPR值是相同的时候，ARP程序便不会工作，因为目标硬件的地址已经获得了。

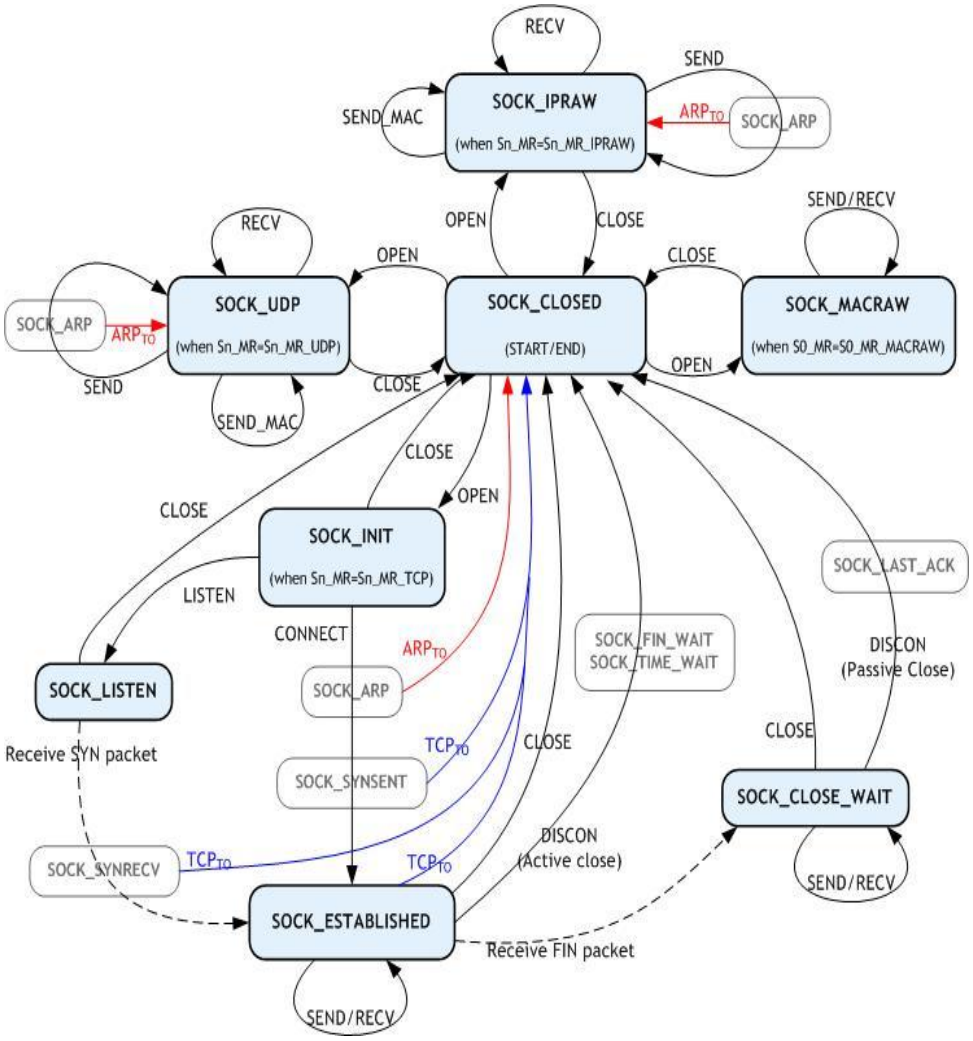


图 7 Socket的状态转换图

Sn_PORT (Socket n-th源端口寄存器) [R/W] [0x4004+0x0n00-0x4005+0x0n00] [0x0000]

当使用TCP或UDP模式时，该寄存器会为每个SOCKET设置一个源端口号，而这个设置需要在执行OPEN命令之前完成。

例) 如SOCKET 0 的端口=5000(0x1388)，配置应如下，

0x4004	0x4005
0x13	0x88

Sn_DHAR (Socket n-th目标MAC地址寄存器) [R/W] [0x4006+0x0n00-0x400B+0x0n00] [0xFFFFFFFFFFFF]

该寄存器会为Socket n-th 设定目标MAC地址。此外，如果Socket0已用于PPPoE模式，那S0_DHAR会为 PPPoE服务器硬件设置一个已知的地址。

当在UDP或IPRAW模式下使用SEND_MAC的命令时，它会为Socket n-th 设定目标MAC地址。而在TCP、UDP和IPRAW模式时，Sn_DHAR会根据定CONNECT或SEND命令中的ARP程序取得的目标MAC地址作出设定。当成功执行CONNECT或SEND命令后，主机可以通过 Sn_DHAR取得目标MAC地址。

当使用W5200中的PPPoE程序，PPPoE服务器MAC地址并不需要进行设置。然而，即使不使用W5200中的PPPoE程序，仍须自行实施MACRAW模式。如想要发送或接收PPPoE的数据包的话，那PPPoE服务器MAC地址(从PPPoE程序中取得)、PPPoE服务器的IP地址、 和PPP session ID就应须被设置。此外，MR(PPPoE) 也应设置为'1'。

S0_DHAR会在OPEN命令之前设置 PPPoE服务器MAC地址。 PPPoE服务器MAC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)如Socket 0的目标MAC地址 = 08.DC.00.01.02.10，配置应如下，

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

Sn_DIPR (Socket n-th目标IP地址寄存器)[R/W][0x400C+0x0n00 0x400F+0x0n00] [0x00000000]

该寄存器会为Socket n-th 设定目标IP地址。如果Socket0已用于PPPoE模式，那S0_DIPR会为 PPPoE服务器IP设置一个已知的地址。它只有在TCP、UDP、IPRAW或PPPoE模式下才有效，但是会在MACRAW模式下被忽略。

在TCP模式下，当作为“TCP客户端”的时候，它会在执行CONNECT命令之前，为“TCP服务器“设置IP地址。而当作为“TCP服务器”的时候，它会在成功建立连接之后，在内部为“TCP客户端”设置 IP地址。

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

例) 如Socket 0的目标IP地址= 192.168.0.11，配置应如下，

0x400C	0x400D	0x400E	0x0400F
192 (0xC0)	168 (0xA8)	0 (0x00)	11 (0x0B)

Sn_DPORT (Socket n-th目标端口寄存器)[R/W][0x4010+0x0n00-0x4011+0x0n00] [0x00]

目标端口号是在Socket n-th 的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”的时候，它会在执行CONNECT命令之前，监听“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模式下，已知的PPP session ID是在S0_DPORT中设置。 在执行OPEN命令之前，PPP session ID(由S0_DPORT0设置)会被用于PSIDR。

例)如Socket 0的目标端口号= 5000(0x1388)，配置应如下，

0x4010	0x4011
0x13	0x88

Sn_MSS (Socket n-th最大分段寄存器)[R/W][0x4012+0x0n00-0x4013+0x0n00] [0x0000]

该寄存器用于TCP的MSS - Maximum Segment Size(最大报文长度) 。当TCP是在被动模式下被启动，该寄存器会显示MSS的设置。而它只是支持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	正常(MR(PPPoE)=‘0’)		PPPoE (MR(PPPoE)=‘1’)	
	预设 MTU	范围	预设 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(最大传输单元-maximum transmission unit)并不是在内部处理的，但会使用预设的MTU。因此，当传输的数据大于默认的MTU时，主机应该以手动方式将数据划分成默认MTU的单元大小。

在TCP或UDP模式下，如果传输的数据大于MTU时，W5200会自动将数据划分成MTU的单元大小。在TCP模式下，MTU会被称为MSS。MSS是从主机的最大读写长度和对方的最大报文长度中取较小者作为TCP通信过程中的最大报文长度。

例)如Socket 0的MSS = 1460(0x05B4)，配置应如下，

0x4012	0x4013
0x05	0xB4

Sn_PROTO (Socket n-th IP协议寄存器) [R/W] [0x4014+0x0n00] [0x00]

这是一个1字节的寄存器。它用于设置在IP层里IP报头（IP header）的协议号码字段。它只会在 IPRAW模式下才有效，而在其他模式下会被忽略。 Sn_PROTO需在执行OPEN命令之前设置。当Socket n-th在IPRAW模式下开启，它会发送和接收在Sn_PROTO中设置的协议号码数据。Sn_PROTO是可以被分配在由0x00～0xFF的范围，但W5200不支持TCP(0x06) 和UDP(0x11)的协议号码。

协议号码已在IANA(互联网地址编码分配机构)里定义。如想阅读细节，请参考在线文檔 Uhttp://www.iana.org/assignments/protocol-numbersU.

例) Internet Control Message Protocol (ICMP - 互联网控制信息协议) = 0x01, Internet Group Management Protocol (互联网组群管理协议) = 0x02

Sn_TOS (Socket n-th IP服务类型寄存器) [R/W] [0x4015+0x0n00] [0x00]

该寄存器设置在IP层里IP header的TOS(Type of Service - 服务类型) 字段。它应在执行OPEN命令之前设置。请参考Uhttp://www.iana.org/assignments/ip-parametersU.

Sn_TTL (Socket n-th IP生存时间寄存器) [R/W] [0x4016+0x0n00] [0x80]

该寄存器设置在IP层里IP header的TTL(Time-To-Live - 生存时间) 字段。它应在执行OPEN命令之前设置。请参考Uhttp://www.iana.org/assignments/ip-parametersU.

Sn_RXMEM_SIZE (Socket n-th 接收内存大小寄存器) [R/W] [0x401E+0x0n00] [0x02]

该寄存器为每个SOCKET配置接收内存的大小。每个SOCKET的接收内存大小是可以配置为 1、2、4、8和16K字节。 当每次系统复位(Reset)时，它将被分配为2K字节。 而每个SOCKET的Sn_RXMEM_SIZE的总和(Sn_RXMEM_SIZESUM) 应是16KB。

数值	0x00	0x01	0x02	0x04	0x08	0x10
内存大小	0KB	1KB	2KB	4KB	8KB	16KB

例1) SOCKET 0 : 8KB, SOCKET 1 : 2KB

0x401E	0x411E
0x08	0x02

例2) SOCKET 2 : 1KB, SOCKET 3 : 1KB

0x421E	0x431E
0x01	0x01

例3) SOCKET 4 : 1KB, SOCKET 5 : 1KB

0x441E	0x451E
0x01	0x01

例4) SOCKET 6 : 1KB, SOCKET 7 : 1KB

0x461E	0x471E
0x01	0x01

Sn_TXMEM_SIZE (Socket n-th 传输内存大小寄存器) [R/W] [0x401E+0x0n00] [0x02]

该寄存器为每个SOCKET配置传输内存的大小。每个SOCKET的传输内存大小是可以配置为 1、2、4、8和16K字节。 当每次系统复位(Reset)时，它将被分配为2K字节。 而每个SOCKET的Sn_TXMEM_SIZE的总和(Sn_TXMEM_SIZESUM) 应是16KB。

例1) SOCKET 0 : 4KB, SOCKET 1 : 1KB

0x401F	0x411F
0x04	0x01

例2) SOCKET 2 : 2KB, SOCKET 3 : 1KB

0x421F	0x431F
0x02	0x01

例3) SOCKET 4 : 2KB, SOCKET 5 : 2KB

0x441F	0x451F
0x02	0x02

例4) SOCKET 6 : 2KB, SOCKET 7 : 2KB

0x461F	0x471F
0x02	0x02

Sn_TX_FSR (Socket n-th 传输空间大小寄存器) [R] [0x4020+0x0n00-0x4021+0x0n00] [0x0800]

该寄存器提供Socket n-th传输内存的可用大小空间(可供传输数据的字节大小)。请留意，主机不能编写大于Sn_TX_FSR的数据。因此，在发送数据之前，切记检查 Sn_TX_FSR。如果你的数据大小小于或等于 Sn_TX_FSR时，在复制数据后，可执行SEND或SEND_MAC命令来传输数据。

在TCP模式下，当W5200通过DATA/ACK机制确认已成功发送了某SIZE的data，那么FSR寄存器将自动增加SIZE大小的空间。

在其他模式下，当 Sn_IR(SENDOK)是'1' 时，Sn_TX_FSR会自动增加已传输的数据大小。

当检查该寄存器时，用户应先阅读高字节*(0x4020，0x4120，0x4220，0x4320，0x4420，0x4520，0x4620，0x4720)*，然后阅读低字节*(0x4021，0x4121，0x4221，0x4321，0x4421，0x4521，0x4621，0x4721)* 才会得到正确的数值。

例) 如2048(0x0800) 在S0_TX_FSR时，

0x4020	0x4021
0x08	0x00

Sn_TX_RD (Socket n-th 传输读指针寄存器) [R] [0x4022+0x0n00-0x4023+0x0n00] [0x0000]

该寄存器显示传输内存里最后一个传输的地址。它可使用Socket n-th的命令寄存器中的SEND命令，把数据从目前 的Sn_TX_RD传输到Sn_TX_WR，并在传送完成后自动更新。因此，在传输完成后，Sn_TX_RD和Sn_TX_WR将具有相同的数值。

当检查该寄存器时，用户应先阅读高字节*(0x4022, 0x4122, 0x4222, 0x4322, 0x4422, 0x4522,0x4622, 0x4722)*，然后阅读低字节 *(0x4023, 0x4123, 0x4223, 0x4323, 0x4423, 0x4523, 0x4623, 0x4723)*才会得到正确的数值。

Sn_TX_WR (Socket n-th 传输写指针寄存器) [R/W] [0x4024+0x0n00-0x4025+0x0n00] [0x0000]

该寄存器提供位置的信息用作编写传输数据。当检查该寄存器时，用户应先阅读高字节*(0x4024,0x4124, 0x4224, 0x4324, 0x4424, 0x4524, 0x4624, 0x4724)*，然后阅读低字节*(0x4025, 0x4125, 0x4225, 0x4325, 0x4425, 0x4525, 0x4625, 0x4725)* 才会得到正确的数值。

注意：此寄存器的数值会在成功执行SEND命令到Sn_CR后被改变。

例) 如2048(0x0800) 在S0_TX_WR时，

0x4024	0x4025
0x08	0x00

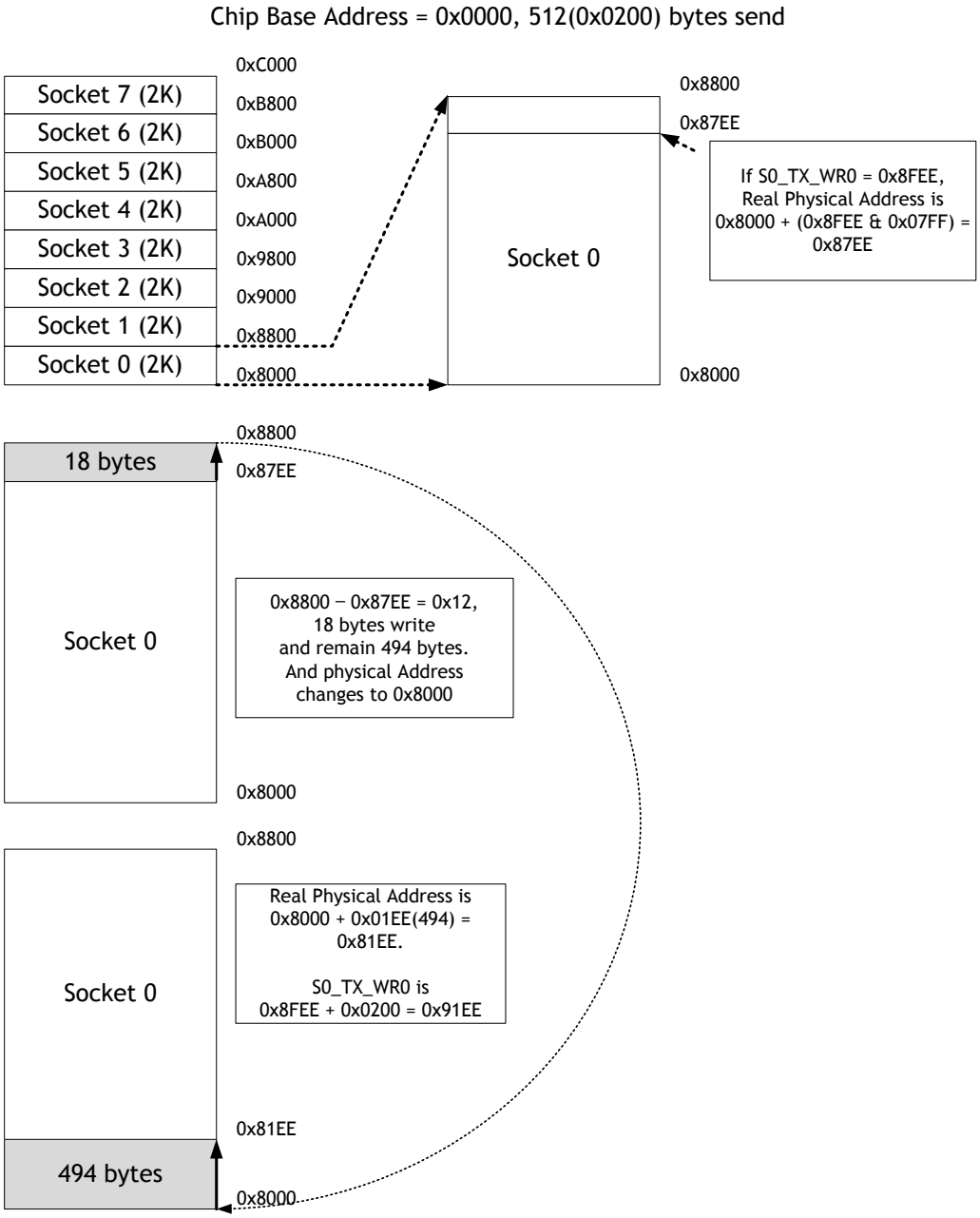


图 8 物理地址的计算

但这读取的数值本身并不是物理地址。因此，物理地址的计算方法如下。

1. Socket n-th传输基址地址(此后会称为gSn_TX_BASE) 和Socket n-th传输掩码地址(此后会称为gSn_TX_MASK)将会以TMSR数值计算。如有需要，请参考psedo的初始化代码。
2. 将Sn_TX_WR和gSn_TX_MASK进行位与运算(bitwise-AND operation)，将得到的结果赋予Socket的传输内存中的偏移地址(此后会称为 get_offset)。
3. 把get_offset和gSn_TX_BASE的数值相加起来，然后将结果赋予物理地址(此后会称为get_start_address)。

现在，编写一个任何大小的传输数据到get_start_address。 (*如在编写的时候超过了Socket传输内存的上界，在这种情况下，先把传输数据编写到上界和把物理地址转到gSn_TX_BASE，然后再编写其余部分的传输数据。) 之后，切记要把Sn_TX_WR的数值增加到跟编写数据的大小相同。最后，提供一个SEND命令给Sn_CR (Socket n-th 命令寄存器)。

如果有需要，请参考TCP服务器模式下传输部分的psedo代码 。

Sn_RX_RSR (已接收大小寄存器) [R] [0x4026+0x0n00-0x4027+0x0n00] [0x0000]

该寄存器向用户提供在Socket n-th接收内存里已接收数据的字节大小。由于这是以Sn_RX_RD和Sn_RX_WR的数值进行内部计算，它会因Sn_CR (Socket n-th 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。

当检查该寄存器时，用户应先阅读高字节*(0x4026, 0x4126, 0x4226, 0x4326, 0x4426, 0x4526, 0x4626, 0x4726)*，然后阅读低字节*(0x4027, 0x4127, 0x4227, 0x4327, 0x4427, 0x4527, 0x4627, 0x4727)*才会得到正确的数值。

例) 如2048(0x0800) 在S0_RX_RSR时，

0x4026	0x04027
0x08	0x00

这个数值的总大小可以根据接收内存大小寄存器的数值来决定。

Sn_RX_RD (Socket n-th 接收读指针寄存器) [R/W] [0x4028+0x0n00-0x4028+0x0n00] [0x0000]

该寄存器提供的位置信息读取接收数据。当检查该寄存器时，用户应先阅读高字节 *(0x4028, 0x4128, 0x4228, 0x4328, 0x4428, 0x4528, 0x4628, 0x4728)*，然后阅读低字节*(0x4029, 0x4129, 0x4229, 0x4329, 0x4429, 0x4529, 0x4629, 0x4729)* 才会得到正确的数值。

注意：此寄存器的数值会在成功执行SEND命令到Sn_CR后被改变。

例) 如2048(0x0800) 在S0_RX_RD时，

0x4028	0x4029
0x08	0x00

但阅读这数值本身并不是物理地址。因此，物理地址的计算方法如下。

1. Socket n-th接收基址地址(此后会称为gSn_RX_BASE) 和Socket n-th接收掩码地址(此后会称为gSn_RX_MASK)将会以RMSR数值计算。如有需要，请参考 psedo的5.1初始化代码。
2. 将Sn_RX_WR和gSn_RX_MASK进行位与运算(bitwise-AND operation)，将得到的结果赋予Socket的传输内存中的偏移地址(此后会称为 get_offset)。
3. 把get_offset和gSn_RX_BASE的数值相加起来，然后将结果赋予物理地址(此后会称为get_start_address)。

Sn_RX_WR (Socket n-th 接收写指针寄存器)[R/W][(0xFE402A + 0xn00) - (0xFE402B + 0xn00)][0x0000]

该寄存器提供位置信息以供编写接收数据。当检查该寄存器时，用户应先阅读高字节*(0x402A, 0x412A, 0x422A, 0x432A, 0x442A, 0x452A, 0x462A, 0x472A)*，然后阅读低字节*(0x402B, 0x412B,0x422B, 0x432B, 0x442B, 0x452B, 0x462B, 0x472B)*才会得到正确的数值。

例) 如2048(0x0800) 在S0_RX_WR时，

0x402A	0x402B
0x08	0x00

Sn_IMR (Socket n-th 中断掩码寄存器)[R/W][0x402C+0x0n00][0xFF]

该寄存器配置Socket n-th的中断，以及通知到主机。Sn_IMR的中断掩码位跟Sn_IR的中断位是对应的。如果中断发生在任何Socket时，该位会设置为‘1’，而与其对应的Sn_IR位也会设置为‘1’。当Sn_IMR和Sn_IR的位同时是‘1’时，IR(n)将成为‘1’。在这个时候，如果IMR(n) 是‘1’，中断将会被发送到主机。 (‘nINT’信号为低电平)

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

位	代号	说明
7	PRECV	Sn_IR(PRECV) 中断掩码 只会在‘SOCKET=0’和‘S0_MR(P3:P0) = S0_MR_PPPoE’时方会有效
6	PFAIL	Sn_IR(PFAIL) 中断掩码 只会在‘SOCKET=0’和‘S0_MR(P3:P0) = S0_MR_PPPoE’时方会有效
5	PNEXT	Sn_IR(PNEXT) 中断掩码 只会在‘SOCKET=0’和‘S0_MR(P3:P0) = S0_MR_PPPoE’时方会有效
4	SENDOK	Sn_IR(SENDOK)中断掩码
3	TIMEOUT	Sn_IR(TIMEOUT) 中断掩码
2	RECV	Sn_IR(RECV)中断掩码
1	DISCON	Sn_IR(DISCON)中断掩码
0	CON	Sn_IR(CON) 中断掩码

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

它设置了IP 层中IP 报头的分段字段。W5200并不支持在IP层的分段数据包。尽管Sn_FRAG已配置，但IP数据并没有分段，同时也不建议在此分段。它应该在执行OPEN命令之前被配置。

例) Sn_FRAG0 = 0x4000 (不要分段)

0x402D	0x402E
0x40	0x00

5 功能说明

通过设置一些寄存器和内存操作，W5200会提供互联网连接。本章描述了它如何可以操作。

5.1 初始化

基本设置

对于 W5200操作，选择和利用适当的寄存器(如下所示)。

1. 模式寄存器(MR)
2. 中断掩码寄存器(IMR)
3. 重试时间值寄存器(RTR)
4. 重试计算寄存器(RCR)

如欲了解更多以上寄存器的信息，请参阅“寄存器说明”。

网络设置信息

基本的网络信息设置以:

它必须设置基本的网络信息。

① SHAR(硬件源地址寄存器)

SHAR中的硬件源地址已经被规定，它使用的是以太网MAC层的唯一的MAC地址作为其硬件地址。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。

如欲了解更多MAC地址分配详情，请参阅以下网址:

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(IP源地址寄存器)

设置Socket 的内存信息

这一步骤设置Socket发送或接收的内存信息。每个Socket的基址地址和掩码地址在这个步骤中被确定并保存。

```
In case of, assign 2KB rx, tx memory per SOCKET
{
gS0_RX_BASE = 0x0000(Chip base address) + 0xC000(Internal RX buffer address); //替Socket 0设置接收内存中的基址地址
Sn_RXMEM_SIZE(ch) = (uint8 *) 2;   // 分配2K接收内存給每个SOCKET
gS0_RX_MASK = 2K - 1; // 0x07FF, 在指定的Socket 0接收内存取得偏移地址
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); // 替Socket 0设置发送内存中的基址地址
Sn_TXMEM_SIZE(ch) = (uint8 *) 2;   //分配2K发送内存给每个SOCKET
gS0_TX_MASK = 2K - 1;
/* 同样的方法，设置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 */
}
```

Sn_TXMEM_SIZE(ch) = 2K,
Chip base address = 0x0000

	0xC000	
Socket 7	0xB800	gS7_TX_BASE = 0xB800 gS7_TX_MASK = 0x07FF
Socket 6	0xB000	gS6_TX_BASE = 0xB000 gS6_TX_MASK = 0x07FF
Socket 5	0xA800	gS5_TX_BASE = 0xA800 gS5_TX_MASK = 0x07FF
Socket 4	0xA000	gS4_TX_BASE = 0xA000 gS4_TX_MASK = 0x07FF
Socket 3	0x9800	gS3_TX_BASE = 0x9800 gS3_TX_MASK = 0x07FF
Socket 2	0x9000	gS2_TX_BASE = 0x9000 gS2_TX_MASK = 0x07FF
Socket 1	0x8800	gS1_TX_BASE = 0x8800 gS1_TX_MASK = 0x07FF
Socket 0	0x8000	gS0_TX_BASE = 0x8000 gS0_TX_MASK = 0x07FF

(a) TX memory

Sn_RXMEM_SIZE(ch) = 2K,
Chip base address = 0x0000

Socket 7	0xF800	gS7_RX_BASE = 0xF800 gS7_RX_MASK = 0x07FF
Socket 6	0xF000	gS6_RX_BASE = 0xF000 gS6_RX_MASK = 0x07FF
Socket 5	0xE800	gS5_RX_BASE = 0xE800 gS5_RX_MASK = 0x07FF
Socket 4	0xE000	gS4_RX_BASE = 0xE000 gS4_RX_MASK = 0x07FF
Socket 3	0xD800	gS3_RX_BASE = 0xD800 gS3_RX_MASK = 0x07FF
Socket 2	0xD000	gS2_RX_BASE = 0xD000 gS2_RX_MASK = 0x07FF
Socket 1	0xC800	gS1_RX_BASE = 0xC800 gS1_RX_MASK = 0x07FF
Socket 0	0xC000	gS0_RX_BASE = 0xC000 gS0_RX_MASK = 0x07FF

(b) RX memory

图 9 Socket n-th内部发送或接收的内存配置数据通信

5.2 数据传输

完成初始化过程后，W5200可以‘开启’TCP、UDP、 IPRAW、MACRAW模式的Socket，对外传输和接收数据。 W5200可同时支持8个Sockets独立使用。在本节中，将会介绍每个模式的通讯方式。

5.2.1 TCP

TCP是一个面向连接(connection-oriented) 的协议。TCP使用其自己的IP地址、端口号、目标IP地址、目标端口号来建立连接Socket。然后通过此Socket发送和接收数据。

利用“TCP服务器”和“TCP客户端”两个方法来连接到Socket，然后通过传输连接请求(SYN数据包)来分开。

“TCP服务器”监听(Listen)由“TCP客户端”发出的连接请求，然后通过接受已发出的连接请求(被动打开) 建立连接Socket。

在“TCP客户端”先发送一个连接请求到“TCP服务器”来进行连接(主动打开)。

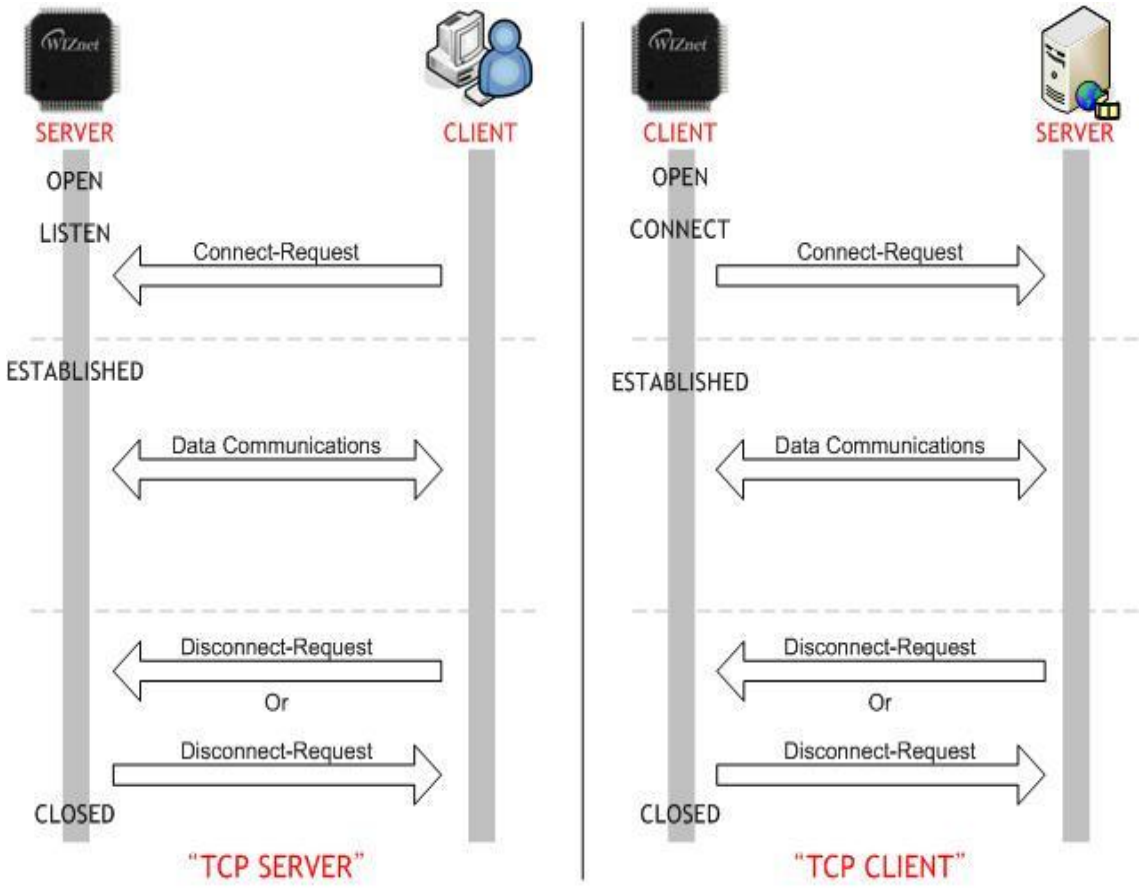


图 10 TCP服务器和TCP客户端

5.2.1.1 TCP 服务器

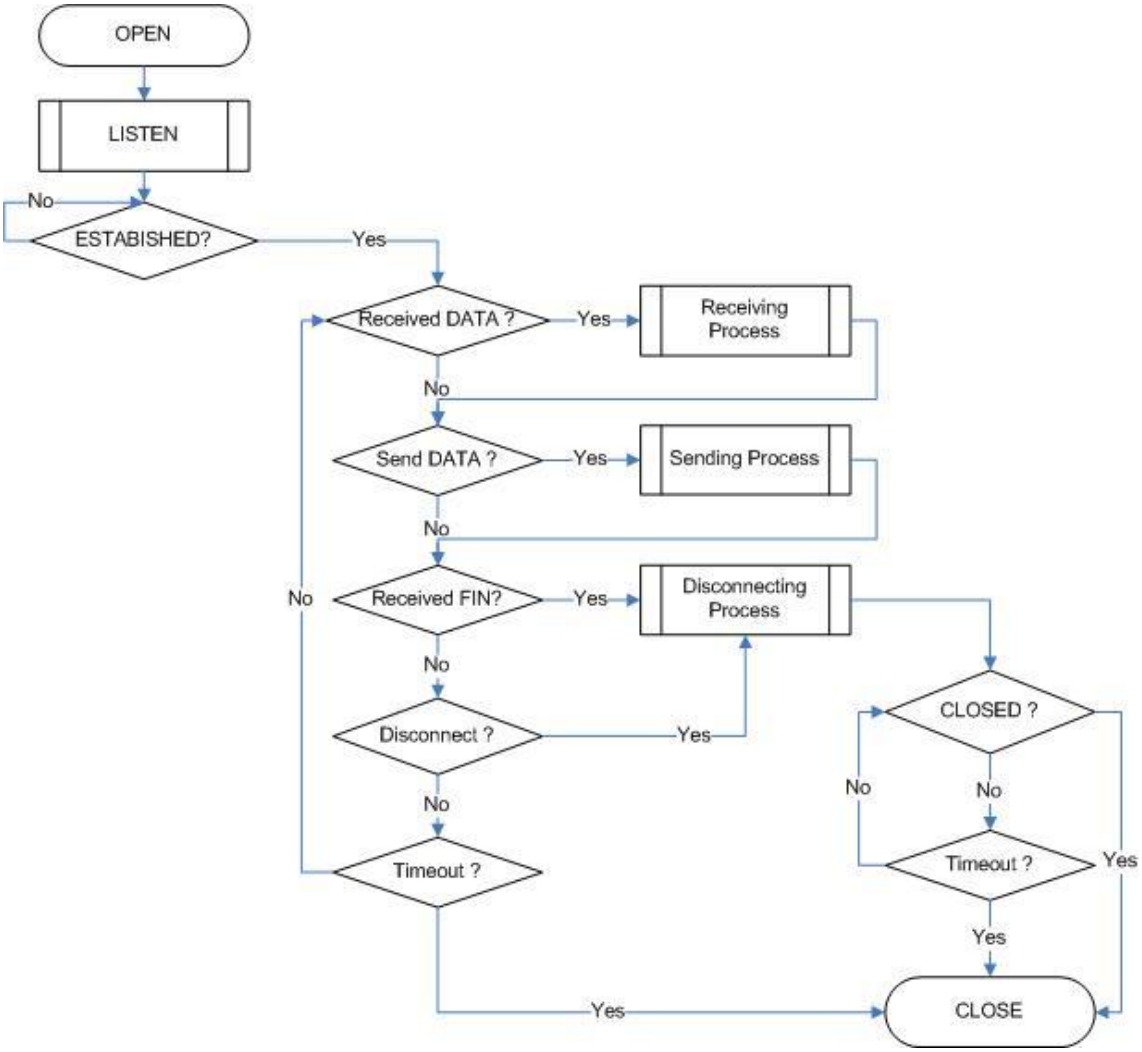


图 11 TCP服务器操作流程

套接字初始化

TCP数据通信必需要套接字初始化。初始化会开启套接字。套接字的开启过程是选择W5200其中一个套接字，并在该套接字设置协议模式(Sn_MR)和源端口号(Sn_PORT0 - 在“TCP服务器”中监听端口号)，然后执行OPEN命令。执行命令之后，如果Sn_SR的状态被更改为 SOCK_INIT，表示SOCKET初始化过程已完成。

套接字初始化过程在“TCP服务器”和“TCP客户端” 的应用是相同的。Socket n-th在TCP模式下的初始化过程如下所示。

```
{
START:
Sn_MR = 0x01;            //设置 TCP模式
Sn_PORT0 = source_port;          //设置源端口号
```

```
Sn_CR = OPEN;                    // 设置OPEN命令
/*等到Sn_SR改为 SOCK_INIT */
if (Sn_SR != SOCK_INIT) Sn_CR = CLOSE; goto START;
}
```

监听 (LISTEN)

用LISTEN命令以“TCP服务器”方式运行。

```
{
/* 监听SOCKET */
Sn_CR = LISTEN;
/* 等到Sn_SR改为 SOCK_LISTEN */
 if (Sn_SR != SOCK_LISTEN) Sn_CR = CLOSE; goto START;
}
```

建立 (ESTABLISHMENT)

当Sn_SR的状态是SOCK_LISTEN时，如收到一个SYN数据包，Sn_SR的状态将被改为SOCK_SYNRECV并会发送SYN/ACK数据包。之后，Socket n-th会建立一个连接。与Socket n-th连接后，开始数据通信。有两种方法来确认Socket n-th的连接。

```
方法一:
{
 if (Sn_IR(CON) == ‘1’)  Sn_IR(CON) = ‘1’; goto ESTABLISHED stage;
 /*在这种情况下，如果Socket n-th的中断被启动，中断将发生。参照IR、IMR
   Sn_IMR和Sn_IR */
}
```

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

建立(ESTABLISHMENT)：检查已接收的数据

确认TCP数据的接收。

```
方法一:
{
  if (Sn_IR(RECV) == '1') Sn_IR(RECV) = '1'; goto Receiving Process stage;
  /*在这种情况下，如果Socket n-th的中断被启动，中断将发生。参照IR、IMR
    Sn_IMR和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'。如主机接收到下一个数据包时没有把Sn_IR(RECV)设置为'1'，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跟接下来的Sn_IR(RECV)重迭。，如果主机不能完全处理每个Sn_IR(RECV)的数据包，这个方法便但是不建议。

建立(ESTABLISHMENT)：接收过程

在这个过程中，它处理已接收在内部接收内存的TCP数据。在TCP模式下，如已接收的数据的大小大于Socket n-th的接收内存空间大小，W5200便无法接收数据。如果提前将这种情况告知对方，W5200便会暂停连接，并等待，直到接收内存空间大小大于已接收的数据的大小。

```
{
/*先取得接收大小*/
 len = Sn_RX_RSR;      // len是已接收的大小
 /*计算偏移地址*/
 src_mask = Sn_RX_RD&gSn_RX_MASK;      // src_mask是偏移地址
 /*计算起始地址 (物理地址) */
 src_ptr = gSn_RX_BASE + src_mask;      // src_ptr是物理起始地址

 /* 如果SOCKET 接收内存溢出*/
 If((src_mask + len) > (gSn_RX_MASK + 1))
 {
   /*复制 source_ptr 的upper_size字节到destination_address */
   upper_size = (gSn_RX_MASK + 1) - src_mask;
   memcpy(src_ptr, dst_ptr, upper_size);
```

```
    /*更新destination_ptr */
    dst_address += upper_size;
    /*复制gSn_RX_BASE 的left_size字节到destination_address */
    left_size = len - upper_size;
    memcpy(gSn_RX_BASE, dst_address, left_size);
  }
  else
  {
    复制source_ptr的len字节到destination_address */
    memcpy(src_ptr, dst_ptr, len);
  }
  /*增加Sn_RX_RD的长度和 Len相同*/
  Sn_RX_RD += len;
  /* 设置RECV命令 */
  Sn_CR = RECV;
 }
```

建立(ESTABLISHMENT): 检查发送数据/发送过程

发送数据的大小不能大于Socket n-th已分配内部的发送内存。如果传输数据的大小大于已配置的MSS，它会被划分为MSS的大小后再传输。要发送下一个数据，用户必须确定前一个的SEND命令已完成。如果前一个的SEND命令未完成，继而执行令一个新的SEND命令时，可能会出现错误。而较大的数据将会需要更多的时间来完成SEND命令。因此，用户应正确地划分数据传输。

如想检查SEND命令是否完成，应检查发送数据的长度是否等于实际发送的数据长度。实际发送的数据长度的计算方法是由之前Sn_TX_RD的数值与执行SEND命令之后的数值的差别计算出来。如果实际发送的数据少于发送数据长度，SEND指令将会重试发送剩余的数据。因此，当实际发送数据的总与发送数据的长度是相等时，表示SEND过程已完成。一个发送过程的简单的例子如下:

例) Send Data Length Size = 10,

1) **连同发送的数据长度，执行SEND命令**
2) **计算的实际发送的数据长度**
   如果实际发送的数据长度为7 (=Sn_TX_RD_after_SEND-Sn_TX_RD_befor_SEND),
   剩余的数据长度= 3
3) **重试SEND命令直到实际发送的数据长度的总和与发送数据的长度是相同。**

注意：不要复制的数据直到实际发送的数据长度的总是与发送数据的长度。

```
{
  /*先取得发送内存的大小空间*/
FREESIZE:
  freesize = Sn_TX_FSR;
  if (freesize<len) goto FREESIZE;     // len是传送的大小

  /*计算偏移地址*/
  dst_mask= Sn_TX_WR0 &gSn_TX_MASK;       // dst_mask是偏移地址
  /*计算起始地址  (物理地址) */
  dst_ptr = gSn_TX_BASE + dst_mask;       // destination_address是物理起始地址
  /*如果SOCKET  发送内存溢出*/
  if ( (dst_mask + len) > (gSn_TX_MASK + 1) )
  {
      /*复制source_addr的upper_size字节到destination_address */
      upper_size = (gSn_TX_MASK + 1) - dst_mask;
      memcpy(src_addr, dst_ptr, upper_size);
      /*  更新source_addr*/
      source_addr += upper_size;
      /*  复制source_addr的left_size字节到gSn_TX_BASE */
      left_size = len - upper_size;
      mememcpy(source_addr, gSn_TX_BASE, left_size);
  }
  else
  {
      /*  复制source_addr的len字节到destination_address */
      memcpy(source_addr, dst_ptr, len);
  }
/*增加Sn_TX_WR0的长度和 Len相同*/
  Sn_TX_WR0 += send_size;
/*设置SEND命令*/
Sn_CR = SEND;
/*返回实际数据包大小*/
return ( read_ptr_after_send - read_ptr_befor_send )
/*如果返回数值不等于 len(len是传送的大小),
 重试发送剩余的数据  (不要复制数据)*/
}
```

建立(ESTABLISHMENT)：检查中断请求（FIN数据包）

检查是否收到中断请求(FIN包)。用户可以确认FIN数据包的接收如下。

```
方法一:
{
  if (Sn_IR(DISCON) == '1') Sn_IR(DISCON)='1'; goto CLOSED stage;
  /* 在这种情况下，如果Socket n-th的中断被启动，中断将发生。参照IR、IMR
     Sn_IMR和Sn_IR */
}
```

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

建立(ESTABLISHMENT)：检查中断或中断程序

当用户不再需要数据通讯，或当接收一个FIN数据包时，会中断Socket连接。

```
{
/* 设置DISCON命令*/
Sn_CR = DISCON;
}
```

建立(ESTABLISHMENT)：检查关闭

确认Socket n-th 是由DISCON或CLOSE命令而中断或关闭。

```
方法一:
{
  if (Sn_IR(DISCON) == '1') goto CLOSED stage;
  /* 在这种情况下，如果Socket n-th的中断被启动，中断将发生。参照IR、IMR
     Sn_IMR和Sn_IR. */
}
```

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

建立(ESTABLISHMENT)：超时

超时会由以下情况触发：连接请求(SYN数据包)或它响应(SYN/ACK数据包)、数据包或它响应(DATA/ACK数据包)、中断请求(FIN数据包)或它响应(FIN/ACK数据包)、传输所有的TCP数据包。如果它不能在RTR和RCR中配置的超时时间之内完全传输以上的数据包，那TCP会触发最终超时(TCPTO)和Sn_SR的状态会设置为 SOCK_CLOSED。TCPTO的确认方法如下：

```
方法一：
{
  if (Sn_IR(TIMEOUT bit) == ‘1’) Sn_IR(TIMEOUT)=‘1’; goto CLOSED stage;
  /*在这种情况下，如果Socket n-th的中断被启动，中断将发生。参照IR、IMR
     Sn_IMR和Sn_IR.*/
}
```

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

SOCKET关闭

它可以用来关闭 Socket n-th。其中可利用中断程序来中断、或利用TCPTO来关闭、或利用主机的需要来关闭(没有中断程序)。

```
{
  /* 清除Socket n-th剩余的中断*/
Sn_IR = 0xFF;
    IR(n) = ‘1’;
  /* 设置CLOSE命令*/
Sn_CR = CLOSE;
}
```

5.2.1.2 TCP 客户端

除了“CONNECT”状态外，与TCP服务器完全一样。用户可参考“5.2.1.1 TCP服务器”。

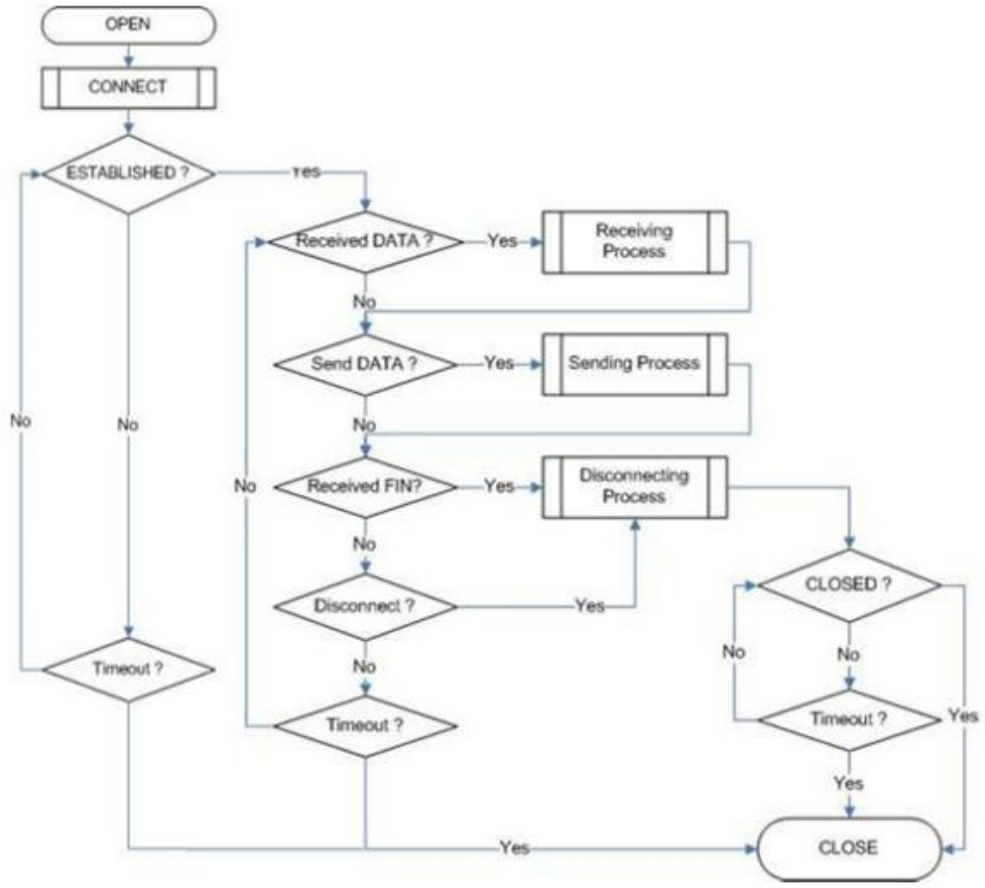


图 12 TCP客户端操作流程图

连接

发送连接请求（SYN包）到“TCP服务器”。当用套接字去连接服务器时，可能会触发像ARP_{TO}、TCP_{TO}这样的超时现象。

```
{
        Sn_DIPR0=server_ip; /*设置TCP服务器IP地址*/
        Sn_DPORT0=server_port;/*设置TCP服务器监听端口*/
        Sn_CR=CONNECT;  /*设置连接命令*/
}
```

5.2.2 UDP

UDP是一个非连接协议。它的通信不用“连接套接字”。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“连接套接字”，因此它可以通过已知的宿主IP地址和端口号与其他很多设备进行通信。只用一个Socket端口就能与其他设备进行通信，这是一个很大的优势；但也有很多缺点，比如说丢失所传数据和从其他设备收到非计划接收数据等问题。为了避免这些问题以及保证可靠性，宿主采取重新传输损坏的数据或忽略从其他设备发来的非计划接收数据。UDP协议支持单播、广播、多播的通信。它遵循下面的通信流程：

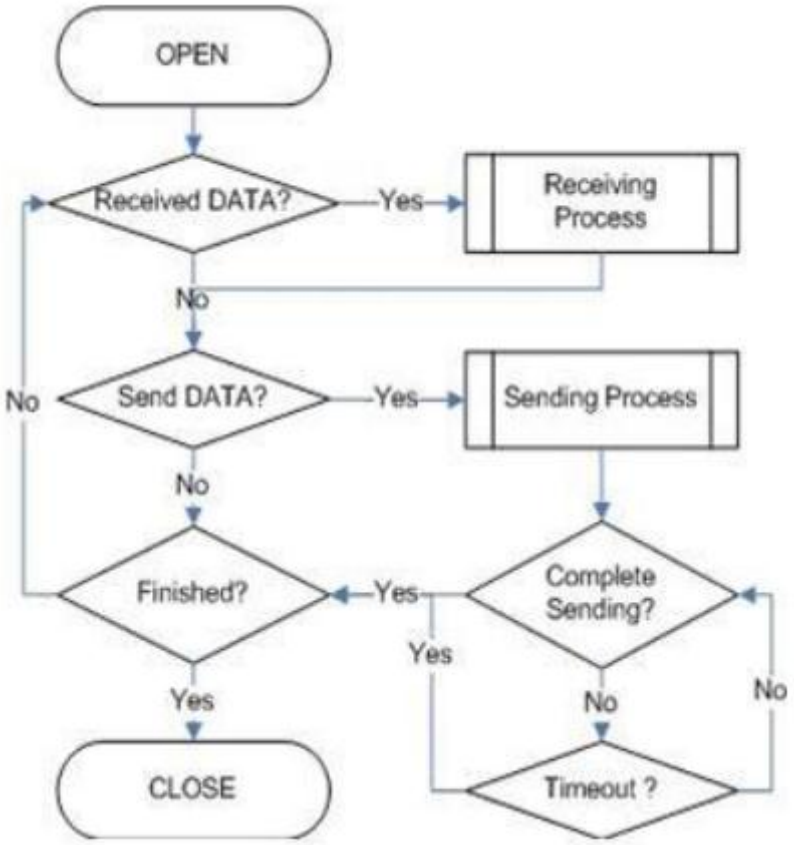


图 13 UDP操作流程

5.2.2.1 单播和广播方式

单播方式是UDP通信中的一种。它一次性将数据发到目的地。而广播式的通信则是用‘广播IP地址（255.255.255.255）’将数据发送到所有可以接收的目的终端。例如，假设用户把数据发向A、B、C三个终端：单播传输模式每一次只能将数据发向A、B和C中的一个目的终端，此时，如果用户有A、B和C目的终端的MAC地址则即使有ARP_{TO}也能实现通信。否则，用户不能将数据发送给有ARP_{TO}的终端。广播方式传输能用“255.255.255.255”或“本地地址|（子网地址）”一次性地同时将数据发到A、B和C这三个目的终端。这时候，没必要获取A、B和C的目的终端MAC地址，而且ARP_{TO}也不会被触发。

注意：广播IP

=>广播IP地址能通过子网掩码的位补码和宿主IP的‘位或’逻辑运算得到。

举例：如果IP是“222.98.173.123”，子网掩码是“255.255.255.0”，那广播IP应为“222.98.173.255”

描述	十进制	二进制
宿主IP	222.098.173.123	11011110.01100010.10101101.01111011
子网掩码的位补码	000.000.000.255	00000000.00000000.00000000.11111111
或逻辑	-	-
广播IP	222.098.173.255	11011110.01100010.10101101.11111111

套接字初始化

对于UDP数据传输来说，套接字的初始化是必须的，它打开了套接字。套接字的打开过程如下所示：首先，选择W5200的其中的一个套接字（共8个），然后设置已选套接字的协议类型（Sn_MR(P3:P0)），然后设置传输所用的源端口号Sn_PORT0。最后执行OPEN命令。Sn_SR的状态将会被改为SOCK_UDP.套接字初始化就完成了。

```
{
START:
Sn_MR = 0x02;                    /*设置UDP模式*/
Sn_PORT0 = source_port;          /*设置源端口号*/
Sn_CR = OPEN;                    /*设置打开命令*/
/*等待Sn_SR转变到SOCK_UDP*/
if (Sn_SR != SOCK_UDP) Sn_CR = CLOSE; goto START;
}
```

检查收到的数据

从目的终端检查收到的UDP数据。用户也可以检查通过TCP接收到的数据。但由于和TCP有相同的问题存在，强烈建议用户用第二种方法。请参考“5.2.1.1TCP服务器”

```
第一种方法：
{
  if (Sn_IR(RECV) == ‘1’) Sn_IR(RECV) = ‘1’; goto Receiving Process stage;
   /*在这种情况下，如果套接字的中断开启，中断就会产生，参考IR、IMR Sn_IMR 和
   Sn_IR*/
}
```

```
第二种方法 ：
{
    if (Sn_RX_RSR0 != 0x0000) goto Receiving Process stage;
}
```

接收过程

在内部Rx缓存中处理接收到的UDP数据

接收到的UDP的数据结构如下所示：

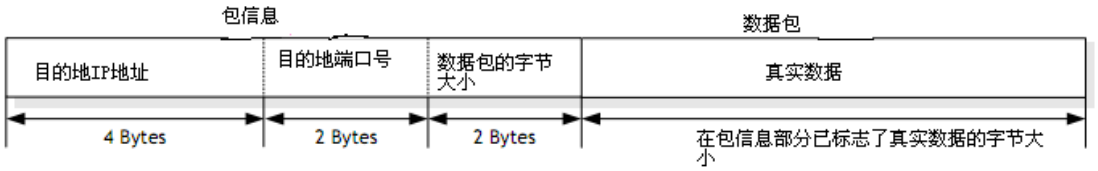


图 14 接收UDP数据的格式

所接收到的UDP数据包括8个字节的PACKET-INFO（包信息）案的DATA包（数据包）。其中，包信息包含发送者的信息（IP地址和端口号）和数据包的字节长度。UDP能从其他许多地方接收到UDP数据。用户能通过报信息中的发送者信息鉴别出是从哪儿接收到的数据。UDP也同样能通过“255.255.255.255”这个IP地址接收到广播套接字。应此主机应该通过发送者信息分析以忽略掉那些不希望接收的数据。

如果所用的Socket处理的数据字节大小超过了内部的Rx自由缓存空间的大小，用户不能接收数据也不能接收数据段。

```
{
  /*计算偏移地址*/
  src_mask = Sn_RX_RD &g Sn_RX_MASK;     //src_mask是偏移地址
  /*计算起始地址（物理地址）*/
  src_ptr = gSn_RX_BASE + src_mask;       //src_ptr是物理起始地址

   /*读头信息（8字节）*/
  header_size = 8;
   /*如果超出了套接字的接收缓存大小*/
  if ( (src_mask + header_size) > (gSn_RX_MASK + 1) )
  {
      /*将src_ptr的前面字节拷贝到头地址*/
      upper_size = (gSn_RX_MASK + 1) - src_mask;
      memcpy(src_ptr, header, upper_size);
      /*更新头地址*/
      header_addr += upper_size;
      /*将gSn_RX_BASE的剩下字节拷贝到头地址*/
      left_size = header_size - upper_size;
      memcpy(gSn_RX_BASE, header, left_size);
     /*更新src_mask*/
      src_mask = left_size;
```

```
}
else
{
    /*将get_start_address的头字节拷贝到头地址*/
    memcpy(src_ptr, header, header_size);
    /*更新src_mask*/
    src_mask += header_size;
}

/*更新src_ptr*/
src_ptr = gSn_RX_BASE + src_mask;

/*保存远程peer信息和接收到的数据字节*/
peer_ip = header[0 to 3];
peer_port = header[4 to 5];
get_size = header[6 to 7];

/*如果超出了套接字接收缓存*/
if ( (src_mask + get_size) > (gSn_RX_MASK + 1) )
{
    /*将src_ptr的前面字节拷贝到目的地址*/
    upper_size = (gSn_RX_MASK + 1) - src_mask;
    memcpy(src_ptr, destination_addr, upper_size);
    /*更新目的地址*/
    destination_addr += upper_size;
    /*将gSn_RX_BASE的剩下字节到目的地址*/
    left_size = get_size - upper_size;
    memcpy(gSn_RX_BASE, destination_addr, left_size);
}
else
{
      /*将src_ptr的字节长度拷贝到目的地址*/
    memcpy(src_ptr, destination_addr, get_size);
}
  /*将Sn_RX_RD增加到len与header_size之和*/
 Sn_RX_RD = Sn_RX_RD + header_size + get_size;
/*设置RECV命令*/
Sn_CR = RECV;
}
```

检查发送数据/发送过程

用户想发送的数据的大小不能超过内部TX缓冲器能容纳的范围。如果比MTU大的话，会自动以MTU为单位进行划分然后发送。当用户想用广播方式时，Sn_DIPR0应被设置成“255.255.255.255”

```
{
 /*第一步，获取TX自由空间大小*/
 FREESIZE:
 freesize = Sn_TX_FSR0;
 if (freesize<len) goto FREESIZE;   //len是发送数据大小

  /*把remote_ip、remote_port赋值给所用套接字目的寄存器（Sn_DIPR）和所用套接字
    目的端口（Sn_DPORT）*/
   Sn_DIPR0 = remote_ip;
   Sn_DPORT0 = remote_port;

  /*计算偏移地址*/
 dst_mask = Sn_TX_WR0 & gSn_TX_MASK;   //dst_mask是偏移地址
  /*计算起始地址（物理地址）*/
 dst_ptr = gSn_TX_BASE + dst_mask;     //dst_ptr是物理起始地址

  /*如果超出套接字发送缓冲*/
 if ( (dst_mask + len) > (gSn_TX_MASK + 1) )
 {
   /*将源地址的前面字节拷贝到dst_ptr*/
    upper_size = (gSn_TX_MASK + 1) - dst_mask;
    memcpy(src_ptr, destination_addr, upper_size);

    /*更新源地址*/
    source_address += upper_size;
    /* 将源地址的左面字节拷贝到gSn_TX_BASE */
    left_size = send_size - upper_size;
    memcpy(src_ptr, destination_addr, left_size);
 else
 {
   /*将源地址的len长度的字节拷贝到dst_ptr*/
    memcpy(src_ptr, destination_addr, len);
 }
```

```
/*将Sn_TX_WR0增加到len长度*/
 Sn_TX_WR0 += len;
/*设置发送命令*/
Sn_CR = SEND;
}
```

检查发送完毕/超时

在继续发送数据之前，用户必须检查先前的SEND命令是不是已经完成了。发送的数据越多，发送需要的时间就越长。因此用户必须合理地将其要发送的数据进行划分。当用户发送UDP数据时是可能触发ARP_{TO}的。如果ARP_{TO}被触发，则传输UDP数据失败。

```
第一种办法：
 {
 /*检查SEND命令是否结束*/
   while(Sn_IR(SENDOK)==‘0’)   /*等待SEND完成的中断*/
 {
 /*检查ARPTO*/
   if (Sn_IR(TIMEOUT)==‘1’) Sn_IR(TIMEOUT)=‘1’; goto Next stage;
 }
   Sn_IR(SENDOK) = ‘1’;/*清除先前SEND结束的中断*/
```

```
第二种方法：
 {
 If (Sn_CR == 0x00) transmission is completed.
 If (Sn_IR(TIMEOUT bit) == ‘1’) goto next stage;
 /*在这种情况下，如果所选择的套接字中断被激活，中断就会发生。参考Interrupt
 Register（IR）、中断掩码寄存器（IMR）和套接字中断寄存器（Sn_IR）*/
 }
```

检查完成/套接字关闭

如果用户不再需要通信，关闭所使用的Socket端口。

```
 {
   /*清除剩余中断位*/
 Sn_IR = 0x00FF;
    IR(n) = ‘1’;
   /*设置关闭命令*/
 Sn_CR = CLOSE;
 }
```

5.2.2.2 多播

广播通信可以跟许多其他设备（不用指明）进行通信。但是多播是针对多播组中的指定的注册目标进行通信的。假设A、B、C是注册在一个指定的多播组中的三个用户。如果用户发送数据到这个组中的A用户，B、C也能收到此数据。使用这种方式时，目的地址列表要使用IGMP协议在多播组中注册。多播组包含“组MAC地址”、“组IP地址”、“组端口号”。用户不能改变“组MAC地址”和“组IP地址”，然而“组端口号可以被改变”。

“组MAC地址”可以在指定的范围选择（从“01:00:5e:00:00:00”到“01:00:5e:7f:ff:ff”），”组IP地址”可以在D段
IP地址中选择（从“224.0.0.0”到“239.255.255.255”），请参考下面的网站：http://www.iana.org/assignments/multicast-addresses.

在选择时，必须保证“组MAC地址”（6个字节）的前23位与“组IP地址”（4字节）是一样的。举例来说，如果用户选择“组IP地址”为“244.1.1.11”，“组MAC地址”被选为“01:00:5e:01:01:0b”。请参考“RFC1112”（http://www.ietf.org/rfc.html）.

在W5200中，处理多播中进行组注册的IGMP是在内部（自动）经行的。当用户以多播方式打开他所用的Socket n-th时，会内部发送“连接（join）”信息，如果用户将其关掉，则会内部发送“离开（leave）”信息。当套接字打开后，当用户通信时，“Report（报告）”信息周期性地在内部被发送。

W5200只支持IGMP版本1和版本2。如果用户想用一个升级了的版本，主机会直接用套接字的IPRAW模式去处理IGMP。

套接字初始化

从W5200的8个套接字中选择一个以进行多播。将“多播组MAC地址”设为Sn_DHAR0,将“多播组IP地址”设为Sn_DIPR0。然后将“多播组端口号”设为Sn_PORT0和Sn_DPORT0。设置Sn_MR(P3:P0)成为UDP模式,将Sn_MR(MULT1)设置为1.最后执行OPEN（打开）命令。如果Sn_SR的状态在OPEN（打开）命令之后被改为SOCK_UDP，套接字初始化就完成了。

```
{
START:
  /*设置多播组信息*/
  Sn_DHAR0 = 0x01;      /*设置多播组H/W地址(01:00:5e:01:01:0b) */
  Sn_DHAR1 = 0x00;
  Sn_DHAR2 = 0x5E;
  Sn_DHAR3 = 0x01;
  Sn_DHAR4 = 0x01;
  Sn_DHAR5 = 0x0B;
  Sn_DIPR0 = 211;       /*设置多播组IP地址（211.1.1.11）*/
  Sn_DIPR1 = 1;
  Sn_DIPR2 = 1;
  Sn_DIRP3 = 11;
  Sn_DPORT0 = 0x0BB8; /*设置多播组端口号（3000）*/
```

```
  Sn_PORT0 = 0x0BB8;   /*设置源端口号*/
  Sn_MR = 0x02 | 0x80; /*在Socket n-th 模式寄存器设置UDP模式和多播 */

  Sn_CR = OPEN;          /*设置打开命令*/

  /*等待Sn_SR改到SOCK_UDP*/
  if (Sn_SR != SOCK_UDP) Sn_CR = CLOSE; goto START;
 }
```

检查接收到的数据

请参考“5.2.2.1单播方式和广播方式”

接收过程

请参考“5.2.2.1单播方式和广播方式”

检查发送数据/发送过程

因为用户在套接字初始化中设置了多播组的信息，因此用户不必再设目的设备的IP地址和端口号。然后，复制要传送的数据到内部TX缓冲区，执行SEND命令就可以了。

```
 {
 /*首先，获得自由的TX缓冲字节数*/
  FREESIZE:
  freesize = Sn_TX_FSR;
  if (freesize<len) goto FREESIZE;     //len是发送字节数

  /*计算偏移地址*/
  dst_mask = Sn_TX_WR0 &gSn_TX_MASK;     //dst_mask是偏移地址
  /*计算起始地址（物理地址）*/
  dst_ptr = gSn_TX_BASE + dst_mask;       //dst_ptr是物理起始地址
  /*如果超过了套接字发送缓冲大小*/
  if ( (dst_mask + len) > (gSn_TX_MASK + 1) )
  {
      /*复制源地址的前面字节到目的地址*/
      upper_size = (gSn_TX_MASK + 1) - dst_mask;
      wizmemcpy((0x000000 + source_addr), (0xFE0000 + dst_ptr), upper_size);
      /*更新源地址*/
      source_addr += upper_size;
      /*拷贝源地址剩下的字节到gSn_TX_BASE*/
      left_size = len - upper_size;
```

```
        wizmemcpy( source_addr, gSn_TX_BASE, left_size);
    }
    else
    {
        /*拷贝源地址的len字节到dst_ptr*/
        wizmemcpy( source_addr, dst_ptr, len);
    }
/*将Sn_TX_WR增加到len长度*/
    Sn_TX_WR0 += send_size;
/*设置SEND命令*/
    Sn_CR = SEND;
  }
```

检查发送完成/超时

因为主机负责所有关于数据传输的协议，因此超时是可能发生的。

```
  {
  /*检查SEND命令是否完成*/
 while(S0_IR(SENDOK)=='0'); /*  等待发送完成的中断*/
 S0_IR(SENDOK) = '1';           /*清除先前的发送完成中断位*/
  }
```

检查完成/套接字关闭

参考"5.2.2.1单播&广播"

5.2.3 IPRAW（以IP层为上限的处理模式）

IPRAW是一种用TCP、UDP和IP层进行数据通信的模式，这些网络层都是较低层的协议层，IPRAW支持IP层协议，比如根据协议号有ICMP（0x01）和IGMP（0x02）。ICMP的“Ping”和IGMP的v1/v2应经被通过硬件逻辑包含进了W5200。如果用户需要，主机能通过打开套接字到IPRAW模式而直接处理IPRAW。在这种情况下，用户必须设置他想要的IP报头的协议段号。协议号是被IANA定义的，请参考网站：http://www.iana.org/assignments/protocol-numbers协议号必须在套接字打开之前被设置进Sn_PROTO。在IPRAW模式下，W5200不支持TCP（0x06）或UDP（0x11）协议号。IPRAW的套接字通信只允许指定协议号的通信。比如IGMP。

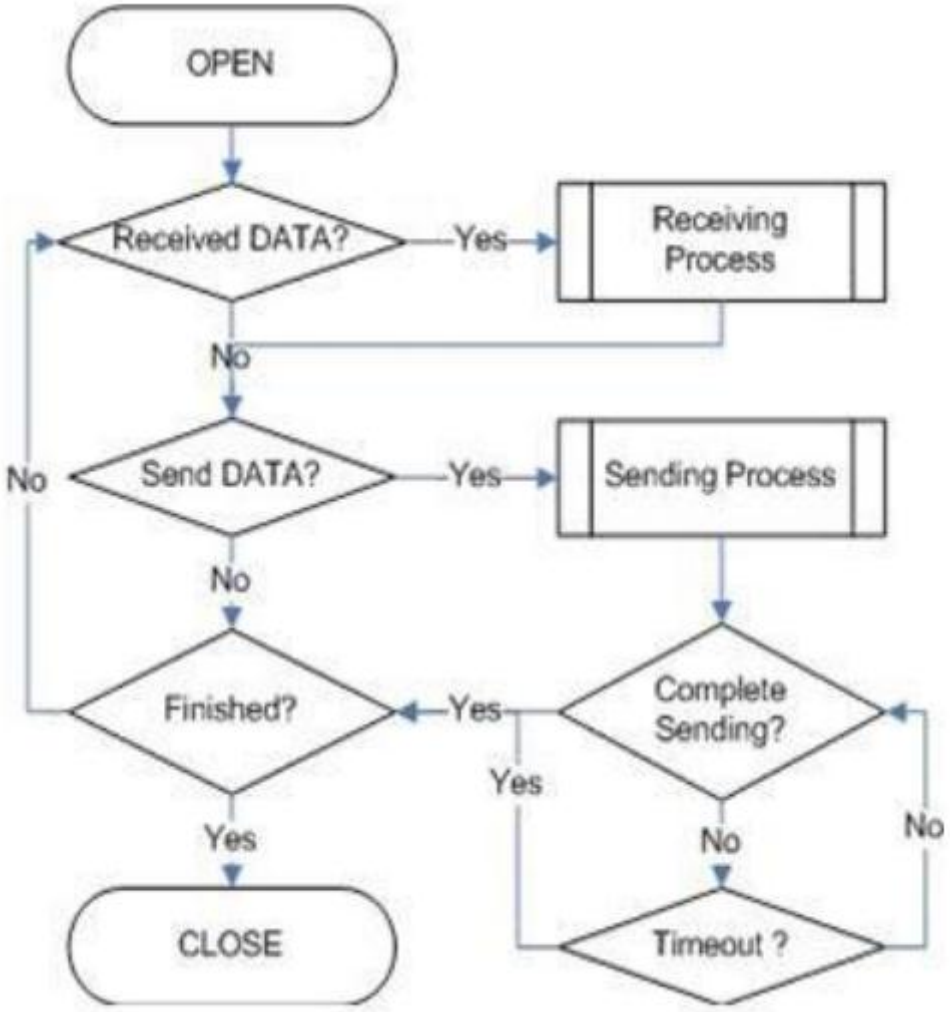


图 15 IPRAW操作流程图

套接字初始化

选择套接字，设置协议号，然后设置Sn_MR(P3:P0)到IPRAW模式，执行“OPEN”命令。如果Sn_SR在“OPEN”命令之后转化到了SOCK_IPRAW，套接字初始化完成了。

```
{
```

```
  START:
/*设置协议号*/
 /*协议号在IP头的协议区*/
    Sn_PROTO = protocol_num;
    /*设置IP raw模式*/
    Sn_MR = 0x03;
    /*设置打开命令*/
    Sn_CR = OPEN;
    /*等待Sn_SR  转变到SOCK_IPRAW模式*/
    if (Sn_SR != SOCK_IPRAW) Sn_CR = CLOSE; goto START;
  }
```

检查接收数据

参考“5.2.2.1单播和广播模式”

接收过程

处理内部接收缓冲区的IPRAW数据。接收到的IPRAW数据的结构如下：

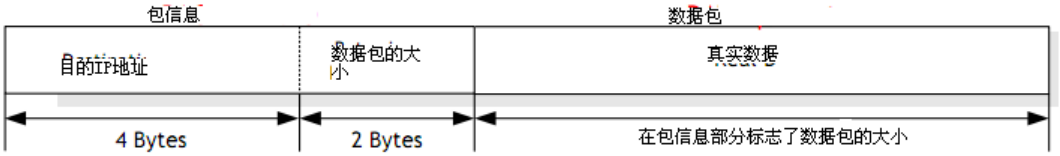


图15 接收IPRAW的数据格式

IPRAW数据包含了6个字节的包信息和数据包，包信息包含了发送者（IP地址）和数据包的长度。除了处理在UDP套接字信息中关于发送者的端口号外，IPRAW的数据接收与UDP数据接收基本上一样。参考“5.2.2.1单播和广播模式”。如果所传输的数据比选用套接字的接收缓冲自由字节数大的话，用户不能接收数据也不能接收数据段。

检查发送数据/发送过程

用户想发送的数据大小不能超过内部发送缓冲大小和缺省的MTU。IPRAW数据的传输同UDP数据传输相比，除了设置“目的端口号”外是一样的。，参考“5.2.2.1单播&广播模式”

完成发送/超时

同UDP一样，请参考“5.2.2UDP”

检查完成/套接字关闭

同UDP一样，请参考“5.2.2UDP”

5.2.4 MACRAW(以MAC层为上限的数据处理模式)

MACRAW通信是基于以太网MAC的，它可以使用户的上层协议灵活地适应主机的需要。MACRAW模式只能用于只有一个套接字工作的情况下，如果用户使套接字0工作于MACRAW，

套接字1-7不仅可以用于硬件连接的TCP/IP协议栈，而且它（套接字0）能被用于NIC（网络接口控制器）中。因此，套接字1-7中的任何一个都可以用于软件TCP/IP协议栈。因为W5200支持硬件TCP/IP协议栈和软件TCP/IP协议栈，它被称做混合TCP/IP协议栈，如果用户除了W5200支持的8个套接字之外，还需要更多的套接字，在这种情况下，用户需要把硬件TCP/IP协议栈用于高性能需求的应用，其他的应该用MACRAW模式的软件TCP/IP协议栈来实现。因此它克服了8个套接字的上限，套接字0的MACRAW能处理除了用套接字1-7实现之外的所有协议。因为MACRAW通信是纯的以太网包通信（没有其他的处理过程），MACRAW数据应该至少包括6字节的源MAC地址，6字节的目的MAC地址和2字节的以太网类型，这是因为它是基于以太网MAC的。

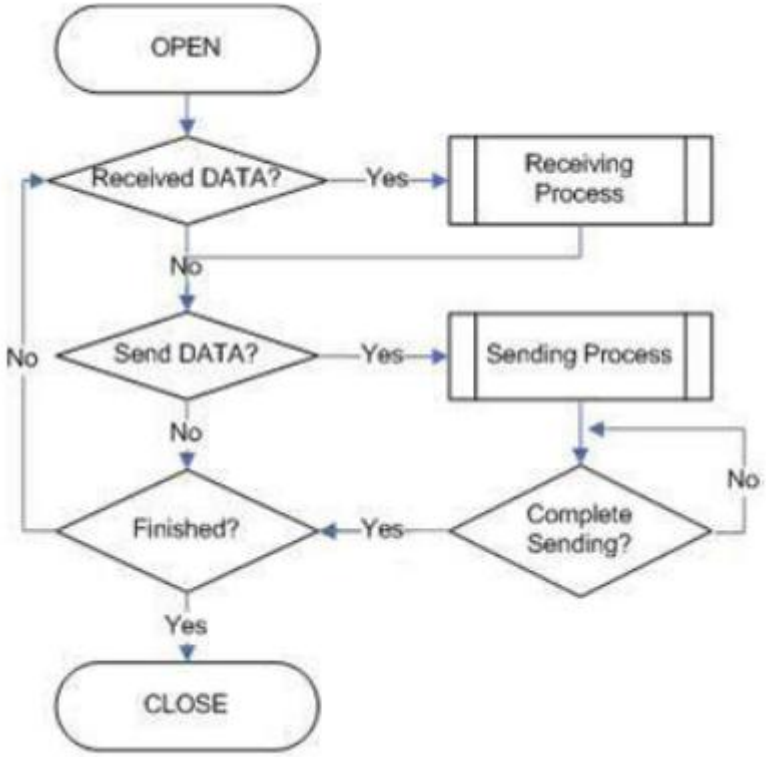


图 16 MACRAW操作流程

套接字初始化

选择套接字，设置SN_MR(P3:P0)到MACRAW模式。然后执行“OPEN”命令。“OPEN”命令过后，如果Sn_SR成功地转到“SOCK_MACRAW”,套接字初始化就完成了。因为关于通信的所有信息（源MAC地址，源IP地址，源端口号，目的MAC地址，目的IP地址，目的设备端口号，协议头等）都在“MACRAW数据”中，没有再多的寄存器设置了。

```
{
START:
  /*设置MACRAW模式*/
  S0_MR = 0x04;
  /*设置OPEN命令*/
  S0_CR = OPEN;
  /*在Sn_SR转到SOCK_MACRAW之前保持等待*/
  if (Sn_SR != SOCK_MACRAW) S0_CR = CLOSE; goto START;
}
```

检查接收数据

参考“5.2.2.1单播和广播模式”

接收过程

处理所用套接字内部接收缓存中MACRAW数据

MACRAW数据的结构如下所示：

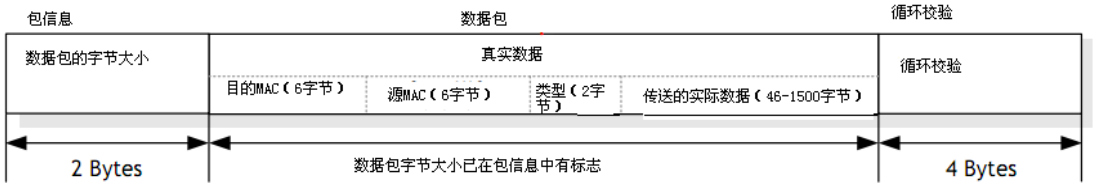


图 17 接收MACRAW数据格式

MACRAW数据包含“包信息”，“数据包”和4字节的循环校验码。“包信息”是数据包的长度，“数据包”包括6字节的“目的MAC地址”，6字节的“源MAC地址”和2字节的“类型”，46-1500字节的实际传输数据。“实际传输数据”包括网络协议，如根据“类型”而定的ARP，IP。关于详细的“类型”信息，清参考下面网站：

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

```
{
  /*计算偏移地址*/
  src_mask = Sn_RX_RD & gSn_RX_MASK;      //src_mask是偏移地址
  /*计算起始地址（物理地址）*/
```

```
    src_ptr = gSn_RX_BASE + src_mask;       //src_ptr是物理起始地址
    /*获取接收到的字节*/
    len = get_Byte_Size_Of_Data_packet //从包信息中获取数据包的字节大小
    /*如果超出了套接字接收缓存*/
    If((src_mask + len) > (gSn_RX_MASK + 1))
    {
    /*将起始地址的前面字节拷贝到目的地址*/
    upper_size = (gSn_RX_MASK + 1) - src_mask;
   memcpy(src_ptr, dst_addr, upper_size);
    /*更新目的地址*/
    dst_addr += upper_size;
/*将gSn_RX_BASE的剩余字节拷贝到目的地址*/
    left_size = len - upper_size;
      memcpy(src_ptr, dst_addr, left_size);
    }
    else
    {
      /*将src_ptr的len长度字节拷贝到目的地址*/
    memcpy(src_ptr, dst_addr, len);
    }
 /*将Sn_RX_RD增加到len长度*/
    Sn_RX_RD += len;
      /*从内部RX缓存中提取4字节的CRC并忽略它*/
    memcpy(src_ptr, dst_addr, len);
      /*设置RECV命令*/
    Sn_CR = RECV;
   }
```

<注意>

如果内部接收缓存的自由空间比MACRAW数据小的话，存于内部RX缓存中的一些包信息和数据包会偶然性地出现一些问题。因此出现包信息的分析错误，所以它不能正确地处理MACRAW数据。越接近内部接收缓存大小，发生错误的概率也大。如果用户可以允许MACRAW数据有部分丢失，这个问题就不是太敏感了。

解决办法如下：

- 尽快地处理内部接收缓存中的数据以避免它接近缓存的上限
- 通过例程里面套接字初始化中的MF位，设置只接受MACRAW数据，以减少接收数据的负担

```
  {
  START:
/*通过使能MAC过滤来设置MACraw 模式*/
  S0_MR = 0x44;
 /*设置打开命令*/
  S0_CR = OPEN;
 /*等待Sn_SR转换到SOCK_MACRAW*/
  if (Sn_SR != SOCK_MACRAW) S0_CR = CLOSE; goto START;
  }
```

•如果内部接收缓存的自由空间小于‘1528-缺省MTU（1514）+包信息（2）+数据包（8）+CRC（4）’，则关闭套接字，然后处理收到的所有数据。之后再重新打开套接字。关闭套接字之后，从关闭开始起接收到的MACRAW数据将会丢失。

```
  {
  /*检查内部接收缓存的自由空间字节数*/
  if((Sn_RXMEM_SIZE(0) * 1024) - Sn_RX_RSR0(0) < 1528)
    {
      recved_size = Sn_RX_RSR0(0);         /*保存Sn_RX_RSR*/
        Sn_CR0 = CLOSE;              /*关闭套接字*/
      while(Sn_SR != SOCK_CLOSED);        /*等待套接字关闭*/
      /*处理内部接收缓存中的任然有的所有数据*/
    while(recved_size> 0)
        {
 /*计算偏移地址*/
          src_mask = Sn_RX_RD&gSn_RX_MASK;//src_mask是偏移地址
          /*计算起始地址（物理地址）*/
          src_ptr = gSn_RX_BASE + src_mask;//src_ptr是物理起始地址
         /*如果超出了套接字的接收缓存*/
          If((src_mask + len) > (gSn_RX_MASK + 1))
          {
            /*将起始地址的前面字节拷贝到目的地址*/
            upper_size = (gSn_RX_MASK + 1) - src_mask;
            memcpy(src_ptr, dst_addr, upper_size);
            /*更新目的地址*/
            dst_address += upper_size;
 /*将gSn_RX_BASE的剩余字节拷贝到目的地址*/
            left_size = len - upper_size;
```

```
            memcpy(src_ptr, dst_addr, left_size);
          }
          else
         {
/*拷贝src_ptr的len个字节到目的地址*/
            memcpy(src_ptr, dst_addr, len);
          }
 /*将Sn_RX_RD增加到len长度*/
          Sn_RX_RD += len;
  /*从内部接收缓存中提取4字节CRC，然后忽略它*/
          memcpy(src_ptr, dst_addr, len);
 /*计算内部接收缓存中仍然有的字节*/
        recved_size = recved_size - 2 - len - 4;
        }
         /*重新打开套接字*/
         /*通过使能MAC filter来设置MAC raw模式*/
         S0_MR = 0x44; /*或者S0_MR=0x04*/
         /*设置打开命令*/
      S0_CR = OPEN;
         /*在Sn_SR转变到SOCK_MACRAW之前等待*/
      while (Sn_SR != SOCK_MACRAW);
    }
  else/*正常处理内部接收缓存里面的数据包*/
    {/*这一部分与“接收过程”那一步是一样的*/
    }
  }
```

检查发送数据/发送过程

用户想发送的数据的大小不能超过内部发送缓存的大小和缺省MTU。主机产生与“接收过程”的数据包一样格式的MACRAW数据，然后发送它。这个时候，如果产生的数据小于60字节，发送的以太网包在内部会以0填充的方式扩充到60字节，然后发送。

```
 {
   /*首先，获得发送缓存的自由空间大小*/
      FREESIZE:
       freesize=S0_TX_FSR;
       if(freesize<send_size) goto FREESIZE;
       /*计算偏移地址*/
       dst_mask=Sn_TX_WR0 & gSn_tX_MASK; //dst_mask是偏移地址
```

```
        /*计算起始地址（物理地址）*/
        dst_ptr=gSn_TX_BASE+dst_mask; //dst_ptr是物理起始地址
        /*如果超过了套接字发送缓存*/
        if((dst_mask+len)>(gSn_TX_MASK+1))
        {
                /*将源地址的前面字节拷贝到目的地址*/
                upper_size=(gSn_TX_MASK+1)-dst_mask;
                memcpy(src_ptr, dst_addr,upper_size);
                /*更新源地址*/
                source_addr+=upper_size;
                /*将源地址的剩余字节拷贝到gSn_TX_BASE*/
                 left_size=len-upper_size;
                memcpy(src_ptr,dst_addr,left_size);
    }
    else
    {
        /*将源地址的len字节拷贝到目的地址*/
        memcpy(src_ptr,dst_addr,len);
    }
    /*将Sn_TX_WR增加到len长度*/
    Sn_TX_WR+=send_size;
    /*设置发送命令*/
    S0_CR=SEND;
}
```

检查发送完成

因为主机管理着所有协议栈数据处理器的通信，超时是不可能发生的

```
{
        /*检查发送结束*/
        while(S0_IR(SENDOK)==’0’); /*等待发送结束的中断*/
        S0_IR(SENDOK)=’1’; /*清先前的发送接收中断位*/
}
```

检查结束/套接字关闭

参考“5.5.2.1单播和广播”

6 外部接口

W5200与MCU通信接口：快速SPI接口

W5200与以太网PHY通信接口：MII接口

6.1 SPI 接口

串行接口模式只需要4个引脚进行数据通信。这4 个引脚的定义分别为：SCLK、nSCS、MOSI、MISO。

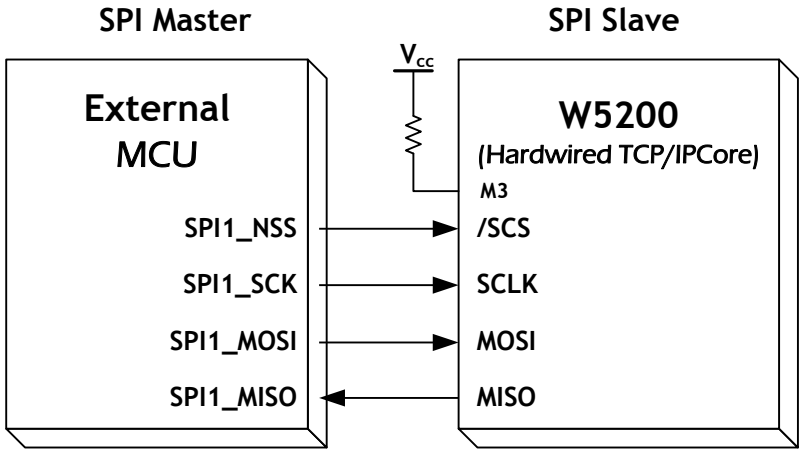


图 18 SPI接口

6.2 设备操作

主控制器（SPI 的主设备）发出一系列指令控制W5200 的运行。SPI 主设备通过四个信号线与W5200 通信：从设备选择（nSCS）、串行时钟（SCLK）、MOSI（主出从入）和MISO（主入从出）。SPI 协议定义了四种操作模式（模式0、1、2、3），每种模式的差异在于SCLK 时钟极性和阶段，它控制数据在SPI 总线上传输。W5200 工作在SPI 从设备的模式0和3，这是最通用的工作模式。

模式 0 和模式3 的唯一差别在于非工作状态时的时钟SCLK 的极性。在SPI 模式0 和模式3，数据在时钟SCLK 的上升沿锁定，在时钟SCLK 的下降沿输出。

6.3 SPI 主设备操作

1. 配置SPI主设备输入输出方向
2. 将nSCS置高电平（无效）
3. 向SPDR写入目标通信地址(SPI数据寄存器)
4. 向SPDR写入操作码和传输数据长度
5. 向SPDR写入要传输的数据
6. 将nSCS置为低电平 (启动数据传输)
7. 等待接收完成
8. 如果所有数据都传输完成，将nSCS置高电平置回高电平

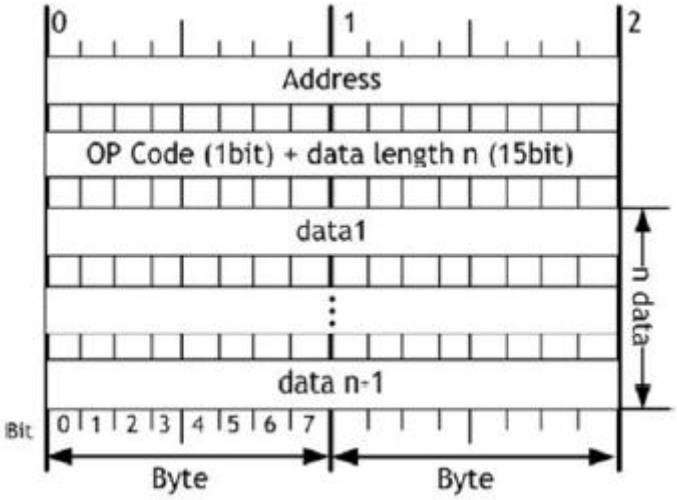


图 19 W5200 SPI帧格式

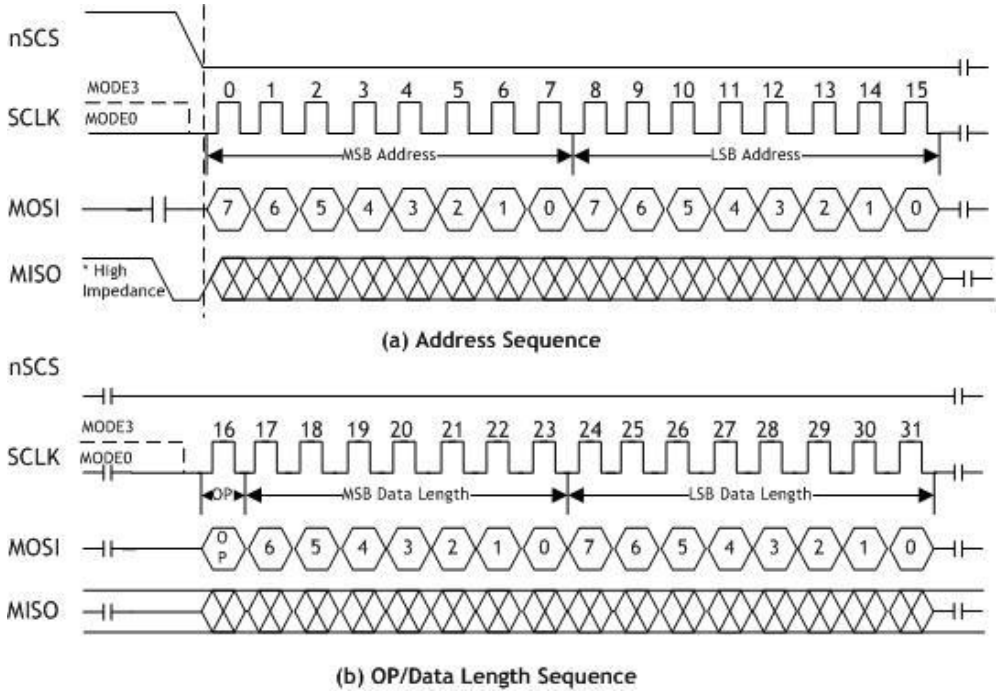


图 20 地址和OP/DATA长度时序图

读操作

图20所示的是读操作的时序图。通过将nSCS置为低电平启动读操作。接着通过MOSI传输地址、操作码、数据长度，最后传输数据。图19是地址和操作码/数据长度时序图。操作码定义读写操作类型：OP = 0，启动读操作．否则，当OP=1时,启动写操作。

在W5200 SPI模式下提供了字节读操作和多字节读操作。字节读取操作需要4个指令：16位的地址，1位的操作码（0x1），15位的数据长度和8位的数据。否则，只需要处理设置脉冲读取操作的脉冲读取数据指令。这里，我们用数据长度来区分字节读取和脉冲读取操作。如果数据长度为1，则进行字节读操作。否则，当数据长度大于2时，执行脉冲读操作。注意：只有在nSCS下降沿之后，通过将MISO电平拉低选中MISO引脚。

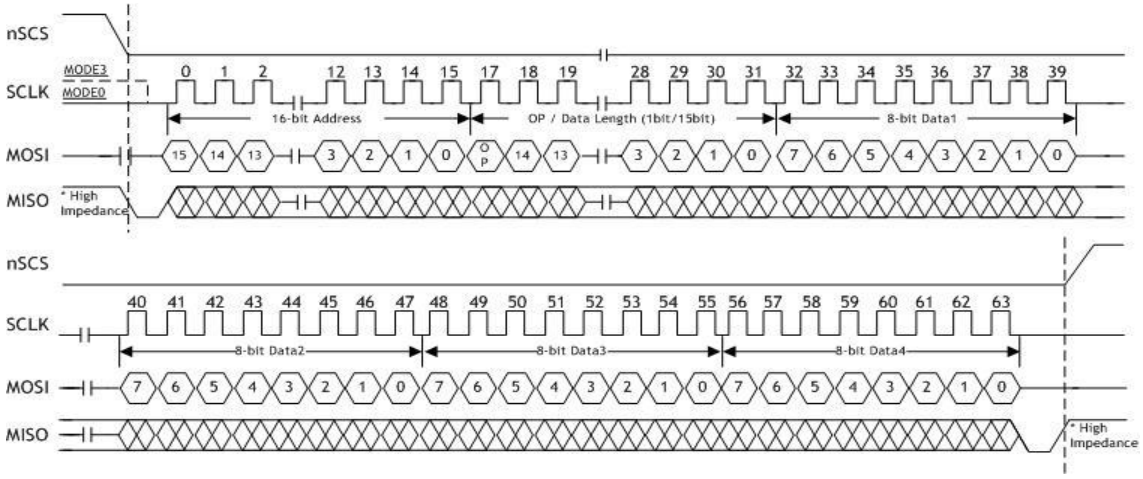


图 21 读时序

```
/*每8bit数据包读取数据的虚拟程序代码*/
#define data_read_command0x00
uint16 addr;          //地址: 16bits
int16 data_len;     //数据长度 :15bits
uint8 data_buf[];//  数据数组
SpiSendData();     //由MCU传送数据到 W5200
SpiRecvData();     //由W5200接收数据到 MCU

{
ISR_DISABLE();//禁止中断服务程序
CSoff();// CS=0, SPI 开始

//Spi传送数据
SpiSendData(((addr+idx) & 0xFF00) >> 8); //地址字节 1
SpiSendData((addr+idx) & 0x00FF);       //地址字节2
```

```
// 读数据命令 + 数据长度上限 7bits
SpiSendData((data_read_command| ((data_len& 0x7F00) >> 8)));
// 最后的数据长度8bits
SpiSendData((data_len& 0x00FF));

//读数据:在 data_len> 1, 突發读取處理模式(Burst Read Processing Mode)
for(intidx = 0; idx<data_len; idx++)
{
    SpiSendData(0);//虚拟数据(dummy data)
    data_buf[idx] = SpiRecvData(idx);//读数据
}
CSon();// CS=1, SPI 完结
ISR_ENABLE();// 禁止中断服务程序
}
```

写操作

图21所示的是写操作的时序图。通过将nSCS置为低电平启动写操作。接着通过MOSI传输地址、操作码、数据长度，最后传输数据。

在W5200 SPI模式下提供了字节写入操作和脉冲写入操作。字节写入操作需要4个指令：16位的地址，1位的操作码（0x1），15位的数据长度和8位的数据。否则，只需要处理设置脉冲写入操作的脉冲写入数据指令。这里，我们用数据长度来区分字节写入和脉冲写入操作。如果数据长度为1，则进行字节写操作。否则，当数据长度大于2时，执行脉冲写操作。在nSCS下降沿之后，通过将MOSI电平拉低选中MOSI引脚。

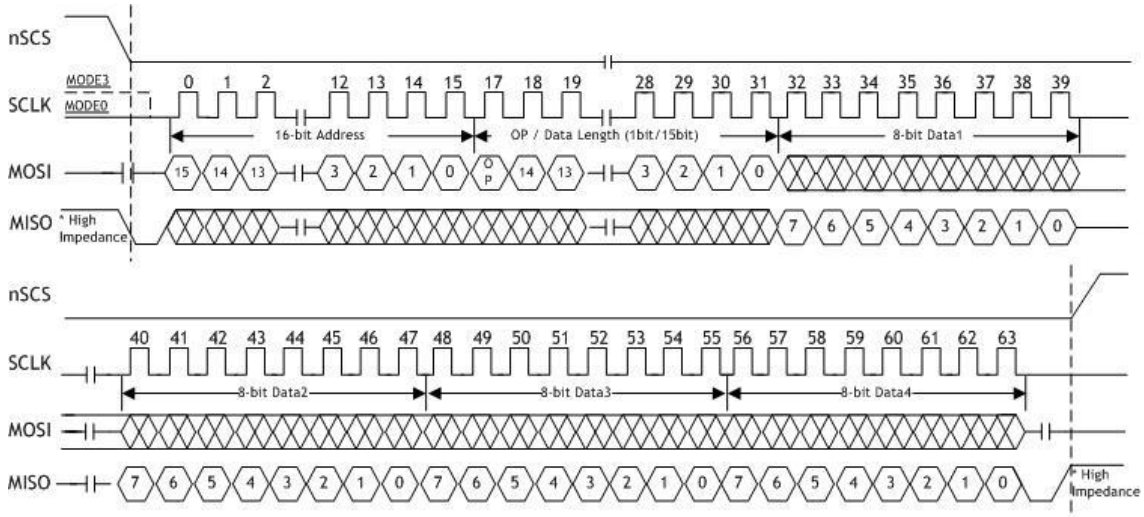


图 22 写时序

```
/* 每8bit数据包写数据的虚拟程序代码 */
#define data_write_command0x80
uint16 addr;          //地址: 16bits
int16 data_len;     //数据长度 :15bits
uint8 data_buf[];//  数据数组
SpiSendData();     //由MCU传送数据到 W5200

{

ISR_DISABLE();//  禁止中断服务程序
CSoff();// CS=0, SPI 开始

SpiSendData(((addr+idx) & 0xFF00) >> 8);//地址字節 1
SpiSendData((addr+idx) & 0x00FF);//地址字節 2
```

```
// 写数据命令 + 数据长度上限 7bits
SpiSendData((data_write_command | ((data_len& 0x7F00) >> 8)));
// 最后的数据长度8bits
SpiSendData((data_len& 0x00FF));

//写数据:在 data_len> 1, 突發读取處理模式(Burst Read Processing Mode)
for(intidx = 0; idx<data_len; idx++)
    SpiSendData(data_buf[idx]);

CSon();// CS=1, SPI 完结
ISR_ENABLE();// 禁止中断服务程序
}
```

7 电器规格

7.1 极限值

Symbol	Parameter	Rating	Unit
V_{DD}	DC Supply voltage	-0.5 to 3.63	V
V_{IN}	DC input voltage	-0.5 to 5.5 (5V tolerant)	V
I_{IN}	DC input current	±5	mA
T_{OP}	Operating temperature	-40 to 85	°C
T_{STG}	Storage temperature	-55 to 125	°C

***注意:**超过这些极限值可能会造成器件永久损坏。

7.2 直流特征

Symbol	Parameter	Test Condition	Min	Typ	Max	Unit
VDD	DC Supply voltage	Junction temperature is from -55°C to 125°C	2.97		3.63	V
VIH	High level input voltage		2.0		5.5	V
VIL	Low level input voltage		- 0.3		0.8	V
VOH	High level output voltage	IOH = 4 ~8 mA	2.4			V
VOL	Low level output voltage	IOL = 4 ~8mA			0.4	V
II	Input Current	VIN = VDD			±5	μA

7.3 功耗 (Vcc 3.3V 温度 25°C)

Condition	Min	Typ	Max	Unit
100M Link	-	160	175	mA
10M Link	-	110	125	mA
Loss Link	-	125	140	mA
100M Transmitting	-	160	175	mA
10M Transmitting	-	110	125	mA
Power Down mode	-	2	4	mA

7.4 特征

7.4.1 复位时钟

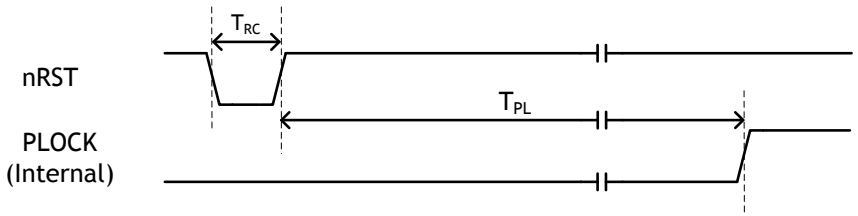


图 23 复位时钟

Symbol	Description	Min	Max
TRC	Reset Cycle Time	2 us	-
TPL	nRST internal PLOCK	-	150 ms

7.4.2 晶体特性

Parameter	Range
Frequency	25 MHz
Frequency Tolerance (at 25℃)	±30 ppm
Shunt Capacitance	7pF Max
Drive Level	59.12uW/MHz
Load Capacitance	27pF
Aging (at 25℃)	±3ppm / year Max

7.4.3 SPI时钟图

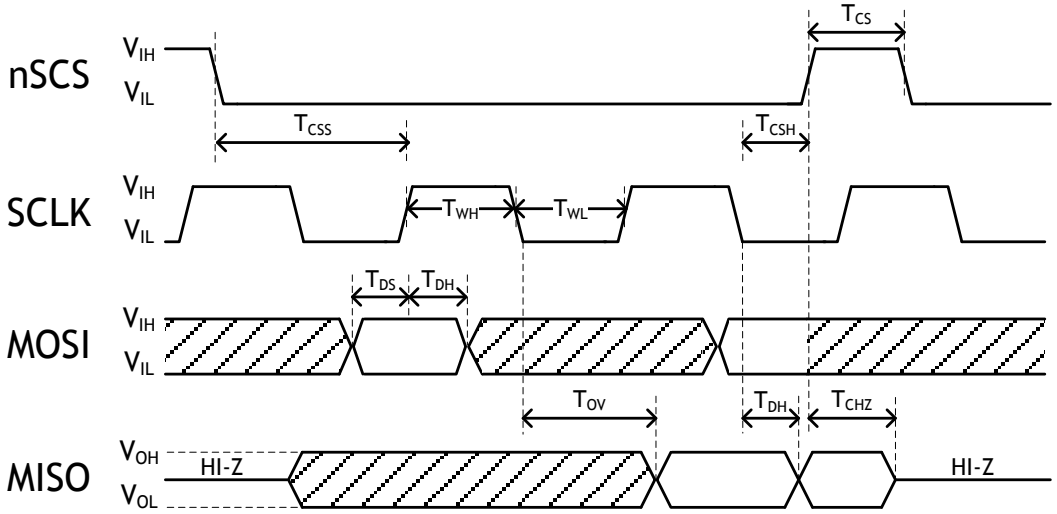


图 24 SPI时钟图

Symbol	Description	Min	Max	Units
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 变压器特性

Parameter	Transmit End	Receive End
Turn Ratio	1:1	1:1
Inductance	350 uH	350 uH

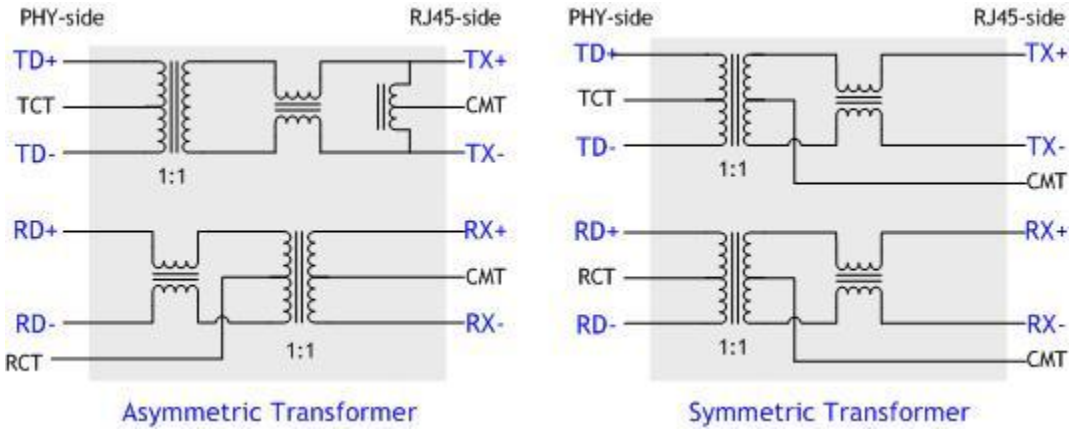


图 25 变压器特性

在使用内部的PHY模式时，一定要使用对称的变压器，以便可以支持自动MDI/MDIX(交叉)。

8 IR Reflow Temperature Profile (Lead-Free)

湿度感应度：3级

需要干燥包装

Average Ramp-Up Rate (Ts_{max} to Tp)	3° C/second max.
Preheat	
- Temperature Min (Ts_{min})	150 °C
- Temperature Max (Ts_{max})	200 °C
- Time (ts_{min} to ts_{max})	60-120 seconds
Time maintained above:	
- Temperature (TL)	217 °C
- Time (tL)	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.

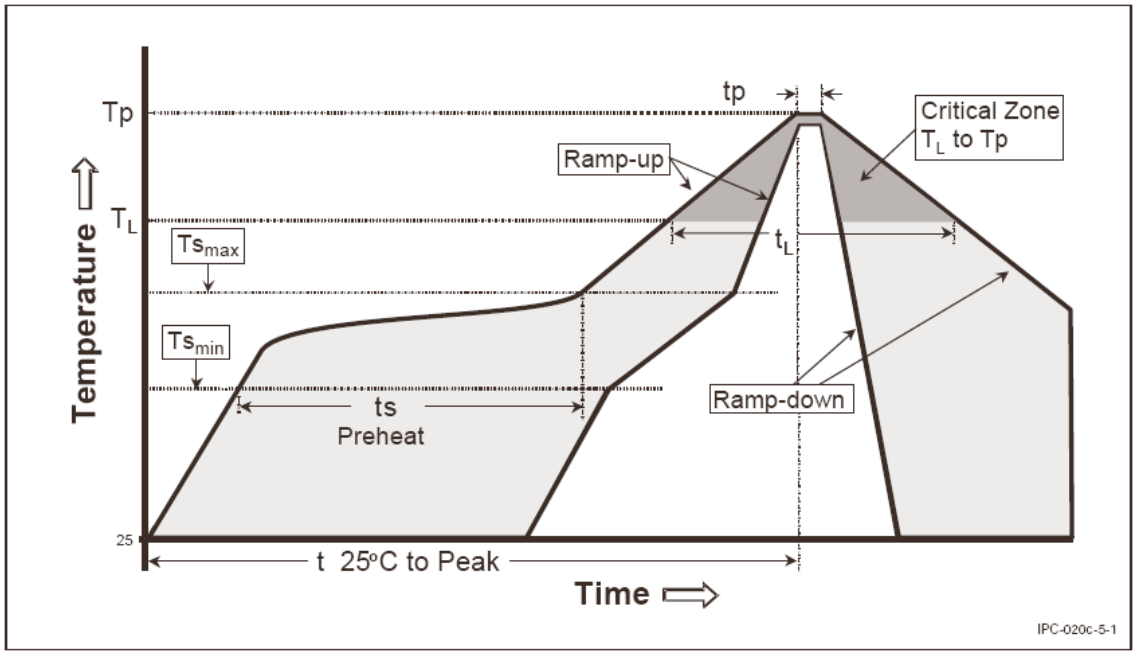
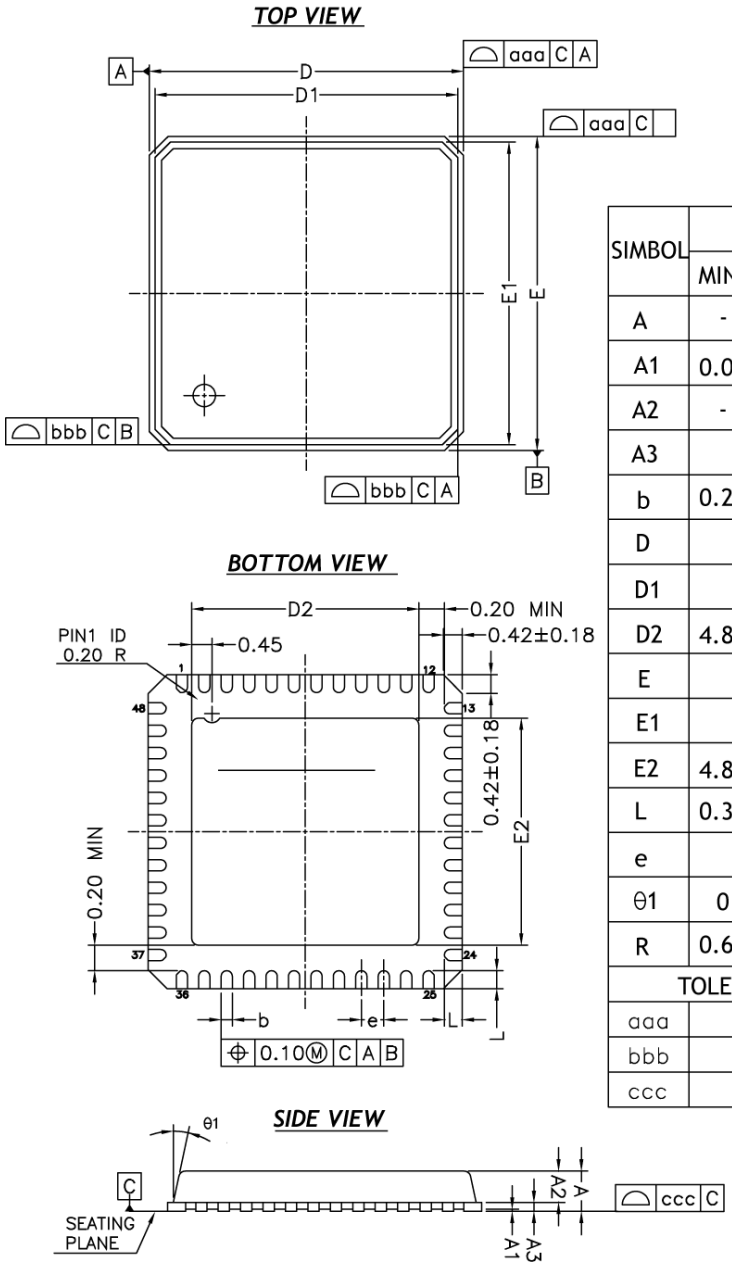


图 26 IR Reflow Temperature Profile

9 封装概述



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		

图 27 封装概述

注意:

1. 所有的尺寸都以毫米为单位。
2. 厚度最大为0.0304mm（0.012英寸）
3. 尺寸和公差符合Y14.5M. -1994.
4. 薄板的尺寸在0.20mm和0.25mm之间。
5. 引脚1的标识符使用凹点标记在封装的顶部以却别其他引脚。。
6. 精确的形状和大小这个功能是可选的。
7. 封装翘曲度最大 0.08 mm.
8. 表面涂敷焊盘和终端，不包括在测量需要的嵌入焊盘。

9. 只适用于终端。
10. 除非另有规定，否则封装角为R0.175+/- 0.025mm

10 文件历史信息

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.76) Added the values of maximum at 7.3 power dissipation (P.76)
Ver.1.2.1	2AUG2011	Fixed the description of READ processing at 6.3 Processing of using general SPI Master device (P.72)
Ver.1.2.2	25NOV2011	Fixed the Block Diagram (P.4)
Ver.1.2.3	3FEB2012	Added the Figure of XTAL_VDD at 1.4 Power Supply Signals (P.11)