W5100 数据手册

Version 1.1.8

北京博控自动化技术有限公司

www.bocon.com.cn

简介

W5100 是一款多功能的单片网络接口芯片，内部集成有 10/100 以太网控制器，主要应用于高集成、高稳定、高性能和低成本的嵌入式系统中。使用 W5100 可以实现没有操作系统的 Internet 连接。W5100 与 IEEE802.3 10BASE-T 和 802.3u 100BASE-TX 兼容。

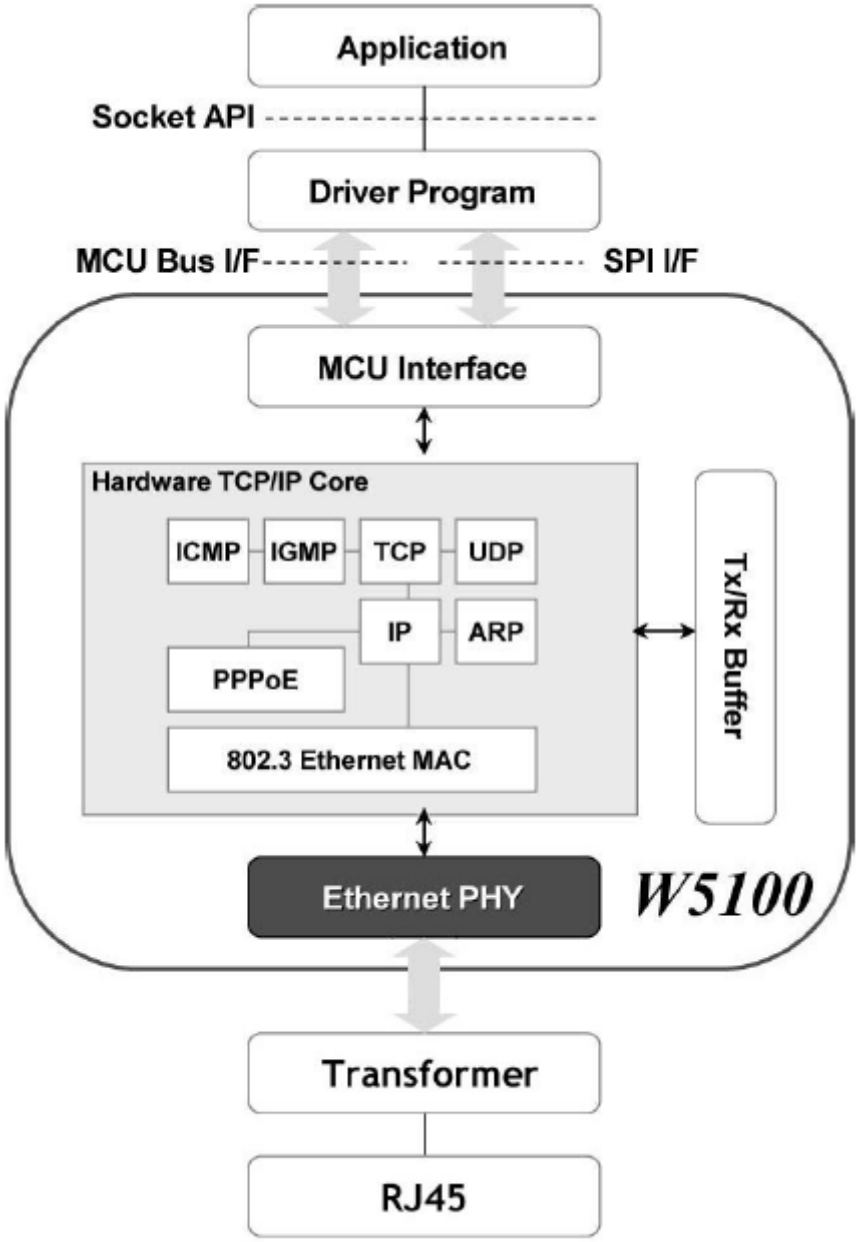
W5100 内部集成了全硬件的、且经过多年市场验证的 TCP/IP 协议栈、以太网介质传输层（MAC）和物理层（PHY）。硬件 TCP/IP 协议栈支持 TCP，UDP，IPv4，ICMP，ARP，IGMP 和 PPPoE，这些协议已经在很多领域经过了多年的验证。W5100 内部还集成有 16KB 存储器用于数据传输。使用 W5100 不需要考虑以太网的控制，只需要进行简单的端口（Socket）编程。

W5100 提供 3 种接口：直接并行总线、间接并行总线和 SPI 总线。W5100 与 MCU 接口非常简单，就像访问外部存储器一样。

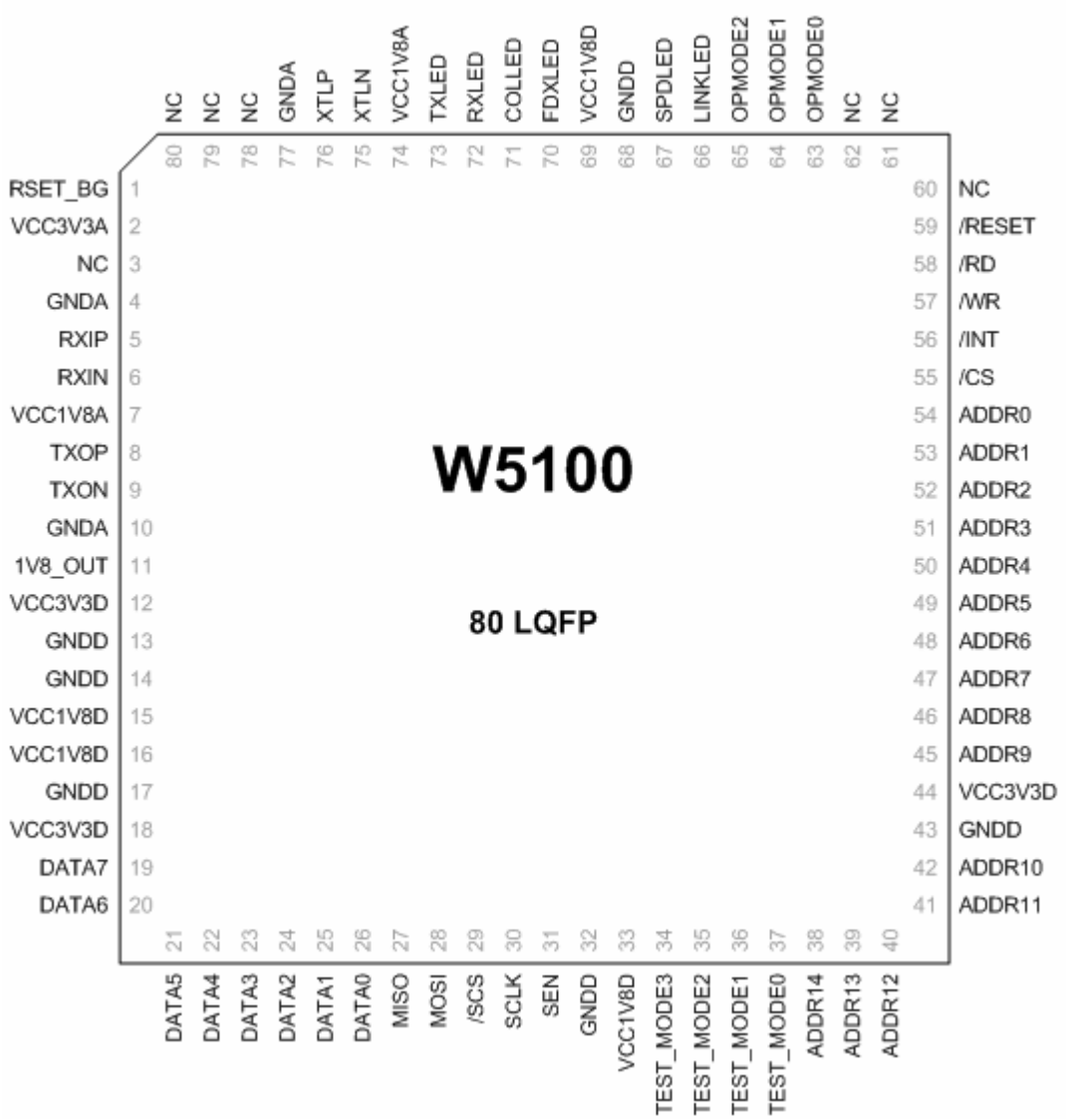
特点：

- 支持硬件化 TCP/IP 协议：TCP，UDP，ICMP，IPv4 ARP，IGMP，PPPoE，以太网
- 内嵌 10BaseT/100BaseTX 以太网物理层
- 支持自动通信握手（全双工和半双工）
- 支持自动 MDI/MDIX，自动校正信号极性
- 支持 ADSL 连接（支持 PPPoE 协议中的 PAP/CHAP 认证模式）
- 支持 4 个独立端口同时运行
- 不支持 IP 的分片处理
- 内部 16KB 存储器用于数据发送/接收缓存
- 0.18μm CMOS 工艺
- 3.3V 工作电压，I/O 口可承受 5V 电压
- 80 脚 LQFP 小型封装
- 环保无铅封装
- 支持 SPI 接口（SPI 模式 0）
- 多功能 LED 信号输出（TX、RX、全双工/半双工、地址冲突、连接、速度等）

结构图


Application
Socket API
Driver Program
MCU Bus I/F
SPI I/F
MCU Interface
Hardware TCP/IP Core
ICMP
IGMP
TCP
UDP
IP
ARP
PPPoE
802.3 Ethernet MAC
Tx/Rx Buffer
Ethernet PHY
W5100
Transformer
RJ45

1. 管脚定义



1.1 MCU 接口信号

符号	管脚	I/O	说明
/RESET	59	I	复置输入，低电平有效 低电平初始化或重新初始化 W5100 低电平持续时间不小于 2μs，所有内部寄存器均置为默认状态
ADDR[14～0]	38，39，40，41，42，45，46，47，48，49，50，51，52，53，54	I	地址总线 这些引脚用来选择寄存器或存储器，地址总线内部下拉为低电平
DATA[7-0]	19，20，21，22，23，24，25，26	I/O	数据总线 这些引脚用来读/写 W5100 内部寄存或存储器

/CS	55	I	片选，低电平有效 片选是用于 MCU 访问 W5100 内部寄存器或存储器，/WR 和/RD 选择数据传输方向
/INT	56	O	中断输出，低电平有效 当 W5100 在端口(Socket)产生连接、断开、接收数据、数据发送完成以及通信超时等条件下，该引脚输出信号以指示 MCU。中断将在写入中断寄存器 IR(中断寄存器)或 Sn_IR(端口 n 的中断寄存器)时自动解除。所有中断都可以屏蔽
/WR	57	I	写使能，低电平有效 MCU 发出信号写 W5100 内部寄存器或存储器，访问地址由 A[14～0]选择。数据在该信号的上升沿锁存到 W5100
/RD	58	I	读使能，低电平有效 MCU 发出信号读 W5100 内部寄存器或存储器，访问地址由 A[14～0]选择
SEN	31	I	SPI 接口使能 该引脚选择允许/禁止 SPI 模式 低电平=禁止 SPI 模式 高电平=允许 SPI 模式 如果不使用 SPI 模式，将该引脚接地
SCLK	30	I	SPI 时钟 该引脚用于 SPI 时钟输入
/SCS	29	I	SPI 从模式选择 该引脚用于 SPI 从模式选择输入，低电平有效
MOSI	28	I	SPI 主输出/从输入 该引脚用于 SPI 的 MOSI 信号
MISO	27	O	SPI 主输入/从输出 该引脚用于 SPI 的 MISO 信号

1.2 以太网物理层信号

符号	管脚	I/O	说明
RXIP	5	I	RXIP/RXIN 信号组 在 RXIP/RXIN 信号组接收到从介质传输来的差分数据信号
RXIN	6	I	
TXOP	8	O	TXOP/TXON 信号组 通过 TXOP/TXON 信号组向介质传输差分数据信号
TXON	9	O	
RSET_BG	1	O	物理层片外电阻 连接一个 12.3k±1%的电阻到地
OPMODE2-0	65，64，63	I	运行控制模式 [2:0] 描述 000 自动握手 001 100 BASE-TX FDX/HDX 自动握手 010 10 BASE-T FDX/HDX 自动握手 011 保留 100 手动选择 100 BASE-TX FDX 101 手动选择 100 BASE-TX HDX 110 手动选择 10 BASE-TX FDX 111 手动选择 10 BASE-TX HDX

1.3 其他

符号	引脚	I/O	说明
TEST_MODE 3-0	34，35，36，37	I	W5100 模式选择 通用模式：0000。其它模式作内部测试用
NC	3 ， 60 ， 61 ，62，78，79，80	I/O	NC 厂家测试用，用户不使用

1.4 电源

符号	引脚	I/O	说明
VCC3V3A	2	P	3.3V 模拟系统电源
VCC3V3D	12，18，44	P	3.3V 数字系统电源
VCC1V8A	7，74	P	1.8V 模拟系统电源
VCC1V8D	15，16，33，69	P	1.8V 数字系统电源
GNDA	4，10，77	P	模拟电源地
GNDD	13，14，17，32 43，68	P	数字电源地
V18	11	O	1.8v 电压输出

1.5 时钟信号

符号	引脚	I/O	说明
XTLP	76	I	25MHz 晶体输入/输出 外接 25MHz 晶体以稳定内部振荡电路 如果使用外部振荡信号，信号连接到 XTLN，而 XTLP 断开，振荡信号的幅度为 1.8V
XTLN	75	I	

1.6 LED 信号

符号	引脚	I/O	说明
LINKLED	66	O	连接 LED 指示 低电平表示 10/100M 连接状态正常 当连接正常时输出低电平，而在 TX/RX 状态时将闪烁
SPDLED	67	O	连接速度 LED 指示 低电平表示连接速度为 100Mbps
FDXLED	70	O	全双工 LED 指示 低电平表示全双工模式
COLLED	71	O	IP 地址冲突 LED 指示 低电平表示网络 IP 地址冲突
RXLED	72	O	接收状态 LED 指示 低电平表示当前接收数据
TXLED	73	O	发送状态 LED 指示 低电平表示当前发送数据

2. 存储器映像

W5100 内含公共寄存器，端口寄存器，发送存储器以及接收存储器，如下图所示。

3. W5100 寄存器

3.1 公共寄存器

地址	寄存器
0x0000	模式（MR）
0x0001 0x0002 0x0003 0x0004	网关地址 (GAR0) (GAR1) (GAR2) (GAR3)
0x0005 0x0006 0x0007 0x0008	子网掩码地址 (SUBR0) (SUBR1) (SUBR2) (SUBR3)
0x0009 0x000A 0x000B 0x000C 0x000D 0x000E	本机硬件地址 (SHAR0) (SHAR1) (SHAR2) (SHAR3) (SHAR4) (SHAR5)
0x000F 0x0010 0x0011 0x0012	本机 IP 地址 (SIPR0) (SIPR1) (SIPR2) (SIPR3)
0x0013 0x0014	保留地址
0x0015	中断(IR)
0x0016	中断屏蔽(IMR)
0x0017 0x0018	重发时间 (RTR0) (RTR1)
0x0019	重发计数(RCR)

地址	寄存器
0x001A	(RMSR)接收存储器大小
0x001B	(TMSR)发送存储器大小
0x001C 0x001D	PPPoE 模式下的认证类型 (PATR0) (PATR1)
0x001E ~ 0x0027	保留
0x0028	(PTIMER)PPP LCP 请求定时器
0x0029	(PMAGIC)PPP LCP 魔数值
0x002A 0x002B 0x002C 0x002D	不能达到 IP 地址 (UIPR0) (UIPR1) (UIPR2) (UIPR3)
0x002E 0x002F	不能达到的端口地址 (UPORT0) (UPORT1)
0x0030 ~ 0x03FF	保留

3.2 端口寄存器

端口 0：

地址	寄存器
0x0400	端口 0 模式(S0_MR)
0x0401	端口 0 命令(S0_CR)
0x0402	端口 0 中断(S0_IR)
0x0403	端口 0 状态(S0_SSR)
0x0404 0x0405	端口 0 的端口号 (S0_PORT0) (S0_PORT1)
0x0406 0x0407 0x0408 0x0409 0x040A 0x040B	端口 0 的目的物理地址 (S0_DHAR0) (S0_DHAR1) (S0_DHAR2) (S0_DHAR3) (S0_DHAR4) (S0_DHAR5)
0x040C 0x040D 0x040E 0x040F	端口 0 的目的 IP 地址 (S0_DIPR0) (S0_DIPR1) (S0_DIPR2) (S0_DIPR3)
0x0410 0x0411	端口 0 的目的端口号 (S0_DPORT0) (S0_DPORT1)
0x0412 0x0413	端口 0 的最大分片字节数 (S0_MSSR0) (S0_MSSR1)
0x0414	IP RAW 模式下端口 0 的协议 (S0_PROTO)

地址	寄存器
0x0415	端口 0 的 IP TOS (S0_TOS)
0x0416	端口 0 的数据包生存期 (S0_TTL)
0x0417 ～0x041F	保留
0x0420 0x0421	端口 0 的发送存储器剩余空间 (S0_TX_FSR0) (S0_TX_FSR1)
0x0422 0x0423	端口 0 的发送存储器读指针 (S0_TX_RD0) (S0_TX_RD1)
0x0424 0x0425	端口 0 的发送存储器写指针 S0_TX_WR0) S0_TX_WR1)
0x0426 0x0427	端口 0 的接收数据大小 (S0_RX_RSR0) (S0_RX_RSR1)
0x0428 0x0429	端口 0 的接收存储器读指针 (S0_RX_RD0) (S0_RX_RD1)
0x042A 0x042B	保留
0x042C～ 0x04FF	保留

端口 1：

地址	寄存器
0x0500	端口 1 模式(S1_MR)
0x0501	端口 1 命令(S1_CR)
0x0502	端口 1 中断(S1_IR)
0x0503	端口 1 状态(S1_SSR)
0x0504 0x0505	端口 1 的端口号 (S1_PORT0) (S1_PORT1)
0x0506 0x0507 0x0508 0x0509 0x050A 0x050B	端口 1 的目的物理地址 (S1_DHAR0) (S1_DHAR1) (S1_DHAR2) (S1_DHAR3) (S1_DHAR4) (S1_DHAR5)
0x050C 0x050D 0x050E 0x050F	端口 1 的目的 IP 地址 (S1_DIPR0) (S1_DIPR1) (S1_DIPR2) (S1_DIPR3)
0x0510 0x0511	端口 1 的目的端口号 (S1_DPORT0) (S1_DPORT1)
0x0512 0x0513	端口 1 的最大分片字节数 (S1_MSSR0) (S1_MSSR1)
0x0514	IP RAW 模式下端口 1 的协议 (S1_PROTO)

地址	寄存器
0x0515	端口 1 的 IP TOS (S1_TOS)
0x0516	端口 1 的数据包生存期 (S1_TTL)
0x0517 ~ 0x051F	保留
0x0520 0x0521	端口 1 的发送存储器剩余空间 (S1_TX_FSR0) (S1_TX_FSR1)
0x0522 0x0523	端口 1 的发送存储器读指针 (S1_TX_RD0) (S1_TX_RD1)
0x0524 0x0525	端口 1 的发送存储器写指针 (S1_TX_WR0) (S1_TX_WR1)
0x0526 0x0527	端口 1 的接收数据大小 (S1_RX_RSR0) (S1_RX_RSR1)
0x0528 0x0529	端口 1 的接收存储读指针 (S1_RX_RD0) (S1_RX_RD1)
0x052A 0x052B	保留
0x052C ~ 0x052FF	保留

端口 2：

地址	寄存器说明
0x0600	端口 2 模式(S2_MR)
0x0601	端口 2 指令(S2_CR)
0x0602	端口 2 中断(S2_IR)
0x0603	端口 2 状态(S2_SSR)
0x0604 0x0605	端口 2 的端口号 (S2_PORT0) (S2_PORT1)
0x0606 0x0607 0x0608 0x0609 0x060A 0x060B	端口 2 的目的物理地址 (S2_DHAR0) (S2_DHAR1) (S2_DHAR2) (S2_DHAR3) (S2_DHAR4) (S2_DHAR5)
0x060C 0x060D 0x060E 0x060F	端口 2 的目的 IP 地址 (S2_DIPR0) (S2_DIPR1) (S2_DIPR2) (S2_DIPR3)
0x0610 0x0611	端口 2 的目的端口号 (S2_DPORT0) (S2_DPORT1)
0x0612 0x0613	端口 2 的最大分片字节数 (S2_MSSR0) (S2_MSSR1)
0x0514	IP RAW 模式下端口 2 的协议 (S2_PROTO)

地址	寄存器说明
0x0615	端口 2 的 IP TOS (S2_TOS)
0x0616	端口 2 数据包生存期 (S2_TTL)
0x0617 ~ 0x061F	保留
0x0620 0x0621	端口 2 的发送存储器剩余空间 (S2_TX_FSR0) (S2_TX_FSR1)
0x0622 0x0623	端口 2 的发送存储器读指针 (S2_TX_RD0) (S2_TX_RD1)
0x0624 0x0625	端口 2 的发送存储器写指针 (S2_TX_WR0) (S2_TX_WR1)
0x0626 0x0627	端口 2 的接收数据大小 (S2_RX_RSR0) (S2_RX_RSR1)
0x0628 0x0629	端口 2 的接收存储读指针 (S2_RX_RD0) (S2_RX_RD1)
0x062A 0x062B	保留
0x062C ~ 0x062FF	保留

端口 3：

地址	寄存器
0x0700	端口 3 模式(S3_MR)
0x0701	端口 3 指令(S3_CR)
0x0702	端口 3 中断(S3_IR)
0x0703	端口 3 状态(S3_SSR)
0x0704 0x0705	端口 3 的端口号 (S3_PORT0) (S3_PORT1)
0x0706 0x0707 0x0708 0x0709 0x070A 0x070B	端口 3 的目的物理地址 (S3_DHAR0) (S3_DHAR1) (S3_DHAR2) (S3_DHAR3) (S3_DHAR4) (S3_DHAR5)
0x070C 0x070D 0x070E 0x070F	端口 3 的目的 IP 地址 (S3_DIPR0) (S3_DIPR1) (S3_DIPR2) (S3_DIPR3)
0x0710 0x0711	端口 3 的目的端口号 (S3_DPORT0) (S3_DPORT1)
0x0712 0x0713	端口 3 的最大分片字节数 (S3_MSSR0) (S3_MSSR1)
0x0714	IP RAW 模式下端口 3 的协议 (S3_PROTO)

地址	寄存器
0x0715	端口 3 的 IP TOS (S3_TOS)
0x0716	端口 3 数据包生存期 (S3_TTL)
0x0717 ~ 0x071F	保留
0x0720 0x0721	端口 3 的发送存储器剩余空间 (S3_TX_FSR0) (S3_TX_FSR1)
0x0722 0x0723	端口 3 的发送存储器读指针 (S3_TX_RD0) (S3_TX_RD1)
0x0724 0x0725	端口 3 发送存储器写指针 (S3_TX_WR0) (S3_TX_WR1)
0x0726 0x0727	端口 3 接收数据大小 (S3_RX_RSR0) (S3_RX_RSR1)
0x0728 0x0729	端口 3 的接收存储器读指针 (S3_RX_RD0) (S3_RX_RD1)
0x072A 0x072B	保留
0x072C ~ 0x072FF	保留

4. 寄存器功能描述

4.1 通用寄存器

MR（模式寄存器）[R/W] [0x0000] [0x00]

该寄存器用于软件复位、Ping 关闭模式、PPPoE 模式以及间接总线接口。

7	6	5	4	3	2	1	0
RST	-	-	PB	PPPoE	-	A1	IND

位	符号	说明
7	RST	软件复位 如被设置为“1”，芯片内部寄存器将被初始化。复位后自动清 0
6	Reserved	保留
5	Reserved	保留
4	PB	Ping 阻止模式 0：关闭 Ping 阻止 1：启动 Ping 阻止 如果该位为“1”， 将不响应 Ping 的请求
3	PPPoE	PPPoE 模式 0：关闭 PPPoE 模式 1：打开 PPPoE 模式 如果不经过路由器直接连接到 ADSL，该位必须置“1”以便与 ADSL 服务器连接。欲知详情，请参照“How to connect ADSL”应用笔记
2	NU	没使用
1	AI	间接总线接口模式下地址自动增加 0：禁止地址自动增加 1：开启地址自动增加 在间接总线接口模式下，当该位置“1”时，每次读/写操作后地址自动增加 1。详情可参考”6.2 间接总线接口模式”
0	IND	间接总线接口模式 0：禁止间接总线接口模式 1：启动间接总线接口模式 当该位置“1”时，采用间接总线接口模式。详情可参照”6. 应用信息” ,”6.2 间接总线接口模式”

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地址。

例：如网关地址为“192.168.0.1”，则

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(Subnet Mask)值

例：子网掩码为“255.255.255.0”，则

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]

该寄存器用来设置本机物理地址。

例：本机物理地址为“00.08.DC.01.02.03”，则

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]

该寄存器用来设置本机 IP 地址。

例：本机 IP 地址为“192.168.0.3”，则

0x000F	0x0010	0x0011	0x0012
192（0xC0）	168（0xA8）	0（0x00）	3（0x03）

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）中的位屏蔽。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。

7	6	5	4	3	2	1	0
CONFLICT	UNREACH	PPPoE	Reserved	S3_INT	S2_INT	S1_INT	S0_INT

位	符号	说明
7	CONFLICT	IP 地址冲突 当一个与本机 IP 地址相同 IP 地址作 ARP 请求时，该位被置“1”。对该位写”1”可清 0
6	UNREACH	无法到达地址 在 UDP 数据包发送过程中，如果目的 IP 地址不存在时，W5100 将会收到一 ICMP(目的无法到达)数据包。（参照 5.2.2.UDP）。在该状态下，IP 地址及端口号将被存到 UIPR 和 UPORT 寄存器，同时该位置“1”。对该位写”1”可清 0
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	S3_INT	端口 3 中断 端口 3 中断产生时，该位被置“1”。（详情请参照端口中断寄存器 S3_IR），当 S3_IR 清 0，该位自动清 0
2	S2_INT	端口 2 中断 端口 2 中断产生时，此位被置“1”。（详情请参照端口中断寄存器 S2_IR），当 S2_IR 清 0，该位自动清 0
1	S1_INT	端口 1 中断 端口 1 中断产生时，此位被置“1”。（详情请参照端口中断寄存器 S1_IR），当 S1_IR 清 0，该自动清 0
0	S0_INT	端口 0 中断 端口 0 中断产生时，此位被置“1”。（详情请参照端口中断寄存器 S0_IR），当 S0_IR 清 0，该自动清 0

IMR（中断屏蔽寄存器） [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，任何时候只要 IR 对应的位也置”1”，中断将会产生。而当 IMR 中屏蔽位被清“0”，即使对应的 IR 中断位被置“1”，中断也不会产生。

7	6	5	4	3	2	1	0
IM_IR7	IM_IR6	IM_IR5	保留	IM_IR3	IM_IR2	IM_IR1	IM_IR 0

位	标志	说明
7	IM_IR7	允许 IP 冲突产生中断
6	IM_IR6	允许地址无法到达产生中断
5	IM_IR5	允许 PPPoE 关闭产生中断
4	Reserved	保留。此位应被置为“0”
3	IM_IR3	允许端口 3 产生中断
2	IM_IR2	允许端口 2 产生中断
1	IM_IR1	允许端口 1 产生中断
0	IM_IR0	允许端口 0 产生中断

RTR（重发时间寄存器）[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），等于 200 毫秒。

例：设定 400 毫秒，其值为 4000（0x0FA0），则

0x0017	0x0018
0x0F	0xA0

当发出 CONNECT、DISCON、CLOSE、SEND、SEND_MAC 及 SEND_KEEP 等命令而没有收到远程对端的响应、或响应延时，都会导致重发过程。

RCR（重发计数寄存器）[R/W] [0x0019] [0x08]

该寄存器内的数值设定可重发的次数。如果重发的次数超过设定值，则产生超时中断（相关的端口中断寄存器中的 Sn_IR 超时位（TIMEOUT）置“1”）。

RMSR（接收存储器配置寄存器）[R/W] [0x001A] [0x55]

此寄存器配置全部 8K 的 RX 存储空间到各指定端口。

7	6	5	4	3	2	1	0
端口 3		端口 2		端口 1		端口 0	
S1	S0	S1	S0	S1	S0	S1	S0

存储器大小由 S0 与 S1 设定。

S1	S0	存储空间
0	0	1KB
0	1	2KB
1	0	4KB
1	1	8KB

在 8K 范围内，从端口 0 开始，由 S0 和 S1 确定 4 个端口的接收缓冲区的大小。如果有端口分配不到存储空间，该端口就不能使用。初始化值为 0x55，4 个端口分别分配 2K 的存储空间。

例： 如果 RMSR 设置为 0xAA，每个端口可分配 4KB 存储空间。但是总存储空间只有 8KB，因此只有端口 0 和端口 1 可以分配 4KB 存储空间，而端口 2 和端口 3 则没有。因此端口 2 和端口 3 绝对不能使用。

端口 3	端口 2	端口 1	端口 0
0KB	0KB	4KB	4KB

TMSR（发送存储配置寄存器）[R/W] [0x001B] [0x55]

该寄存器用来将 8K 的发送存储区分配给每个端口。发送存储区的设置方法与接收存储区的设置完全一样。初始化值为 0x55，即每端口分别分配 2KB 的空间。

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

在与 PPPoE 服务器连接时，该寄存器指示已经被通过的安全认证方法。W5100 只支持两种安全认证类型：PAP 及 CHAP。

数值	认证类型
0xC023	PAP
0xC223	CHAP

PTIMER（PPP LCP 请求计时寄存器）[R/W] [0x0028] [0x28]

该寄存器表示发出 LCP Echo(响应请求)所需要的时间间隔。每 1 单位大约 25 毫秒。

例：当 PTIMER = 200，200 x 25 毫秒 = 5000 毫秒 = 5 秒钟

PMAGIC（PPP LCP 魔数寄存器）[R/W] [0x0029] [0x00]

该寄存器用于 LCP 握手时采用的魔数选项。参照“How to connect ADSL”应用笔记。

UIPR（无法到达的 IP 地址寄存器）[R] [0x002A～0x002D] [0x00]

在 UDP 数据传输时（参照 5.2.2. UDP），如果目的 IP 地址不存在，将会收到一个 ICMP（地址无法到达）数据包。在这种情况下，无法到达的 IP 地址及端口号将分别存储到 UIPR 和 UPORT 中。

例：无法到达的 IP 地址为“192.168.0.11”，则

0x002A	0x002B	0x002C	0x002D
198(0xC0)	168(0xA8)	0(0x00)	11(0x0B)

UPORT（无法到达的端口号寄存器）[R] [0x002E～0x002F] [0x000C]

详情请参照 UIPR 所阐述的内容。

例：无法到达端口号为 5000（0x138），则

0x002E	0x002F
0x13	0x88

4.2 端口寄存器

Sn_MR（端口 n 模式寄存器）[R/W] [0x0400，0x0500，0x0600，0x0700] [0x00]

该寄存器设置相应端口的选项或协议类型。

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

<table>
<tr><th>位</th><th>标志</th><th>说明</th></tr>
<tr><td>7</td><td>MULTI</td><td>广播功能
0：关闭广播功能
1：启动多广播功能
广播功能只能在 UDP 模式下使用。使用广播时，必需在 OPEN 命令前，将广播组地址和端口号写入到端口 n 的目的 IP 地址寄存器和目的端口号寄存器（n 为 0~3）</td></tr>
<tr><td>6</td><td>Reserved</td><td>保留</td></tr>
<tr><td>5</td><td>ND/MC</td><td>使用无延时响应
0：禁止无延时响应选项
1：允许无延时响应选项
该功能只有在 TCP 模式下有效，为”1”时，无论什么时候从对端收到数据，都会发送一个 ACK 响应，为”0”时，将根据内部延时机制发送 ACK 响应
广播
0：使用 IGMPv2
1：使用 IGMPv1
该功能只有 MULTI 位为“1”时有效</td></tr>
<tr><td>4</td><td>Reserved</td><td>保留</td></tr>
<tr><td>3
2
1
0</td><td>P3
P2
P1
P0</td><td>协议类型。设定端口采用的协议类型，如 TCP，UDP，IP RAW 型
<table>
<tr><th>P3</th><th>P2</th><th>P1</th><th>P0</th><th>说明</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>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>IP RAW</td></tr>
</table>
端口 0 的协议类型增加了 MAC RAW 和 PPPoE 型
<table>
<tr><th>P3</th><th>P2</th><th>P1</th><th>P0</th><th>说明</th></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>MAC RAW</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>PPPoE</td></tr>
</table></td></tr>
</table>

Sn_CR（端口 n 命令寄存器）[R/W] [0x0401，0x0501，0x0601，0x0701] [0x00]

该寄存器用来设置端口的初始化、关闭、建立连接、断开连接、数据传输以及命令接收等。命令执行后，寄存器的值自动清 0x00。

数值	类型	说明
0x01	OPEN	用于初始化端口。根据端口 n 模式寄存器(Sn_MR)的设置，端口 n 的状态寄存器(Sn_SSR)的值将改变为 SOCK_INIT、SOCK_UDP、SOCK_IPRAW 或 SOCK_MACRAW。详情参照”5. 功能描述”
0x02	LISTEN	只有 TCP 模式有效 它将改变端口 n 的状态寄存器(Sn_SSR)为 SOCK_LISTEN，以便等待远程对端发送的连接请求。详情参照”5.2.1.1 服务器模式”
0x04	CONNECT	只有 TCP 模式有效 它发送一个连接请求到远程对端服务器。如连接失败，将产生超时中断。详情参照”5.2.1.2 客户端模式”
0x08	DISCON	只有 TCP 模式时有效 它发送终止连接请求。如连接终止失败，将产生时间溢时中断。详情参照”5.2.1.1 服务器模式” 如果用 CLOSE 命令代替 DISCON 命令，只有端口 n 的状态寄存器(Sn_SSR)改变成 SOCK_CLOSED，而不进行终止连接过程
0x10	CLOSE	该命令用于关闭端口。它改变端口 n 的状态寄存器(Sn_SSR)为 SOCK_CLOSED
0x20	SEND	按照端口 n 的 TX 写指针增加的大小发送数据。详情参考端口 n 发送剩余空间寄存器(Sn_TX_FSR)、端口 n 发送写指针寄存器(Sn_TX_WR)、端口 n 发送读指针寄存器(Sn_TX_RR)，或”5.2.1.1 服务器模式”
0x21	SEND_MAC	它在 UDP 模式有效 基本功能与 SEND 相同。通常 SEND 操作过程需要用到由 ARP 解析的目标硬件地址。而 SEND_MAC 操作时使用用户设置的目的物理地址(Sn_DHAR)，而没有 ARP 过程
0x22	SEND_KEEP	只有 TCP 模式有效 它通过发送 1 个字节的数据检查端口的连接状态。如果连接已经终止或对端没有响应，将产生超时中断
0x40	RECV	按照端口 n 的读指针寄存器(Sn_RX_RR)中的数据接收处理完成。 详情请参照端口 n 接收数据大小寄存器(Sn_RX_RSR)、端口 n 接收写指针寄存器(Sn_RX_WR)、端口 n 接收读指针寄存器(Sn_RX_RR)

Sn_IR（端口 n 中断寄存器）[R] [0x0402，0x0502，0x0602，0x0702] [0x00]

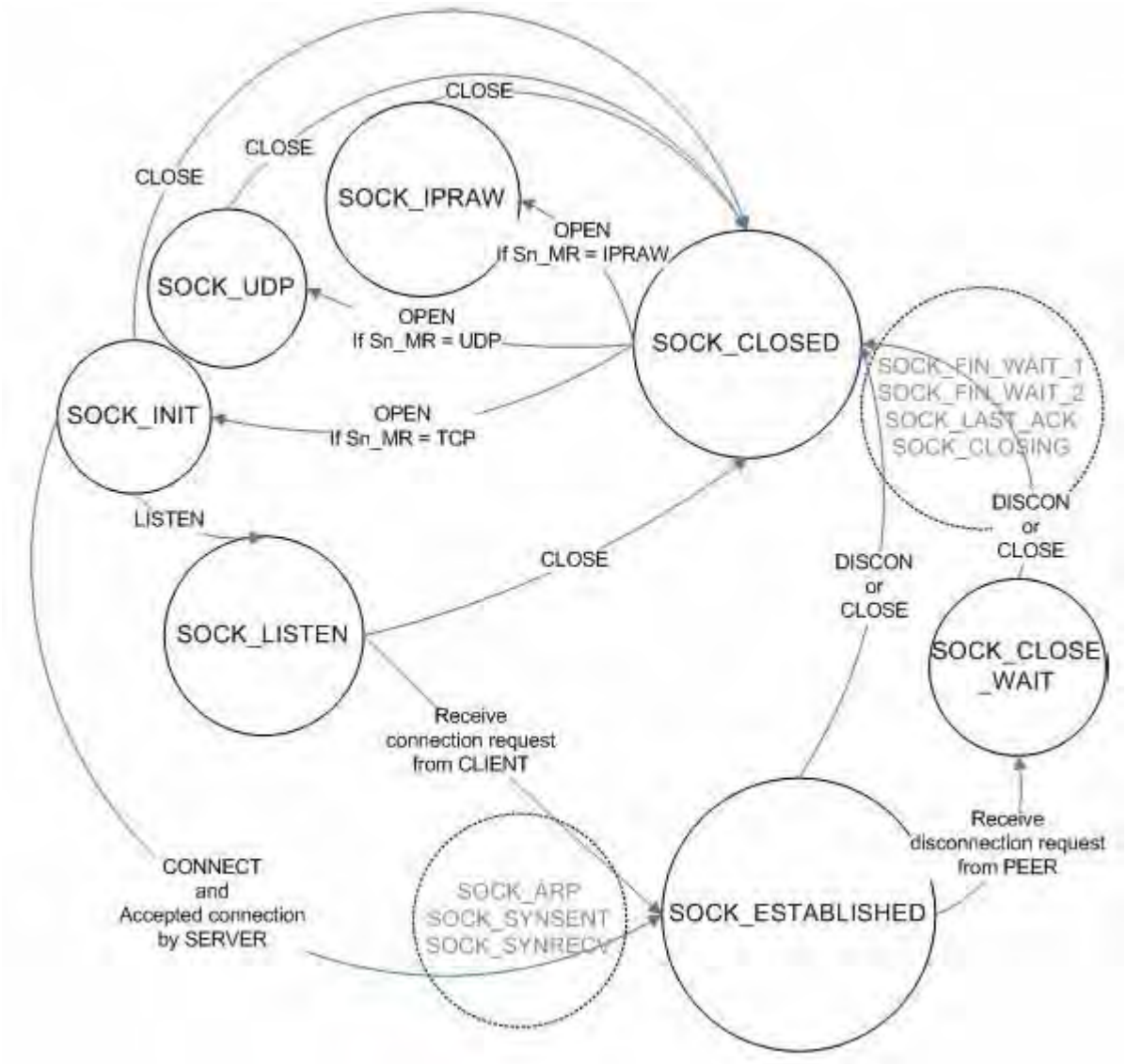
该寄存器指示建立和终止连接、接收数据、发送完成以及时间溢出等信息。寄存器中相应的位被置”1”后必须写入”1”才清 0。

7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	SEND_OK	TIMEOUT	RECV	DISCON	CON

位	类型	说明
7	Reserved	保留
6	Reserved	保留
5	Reserved	保留
4	SEND_OK	数据发送操作完成该位被置”1”
3	TIMEOUT	在连接或终止、数据发送等过程中超时，该位被置“1”
2	RECV	当端口接收到数据时置“1”，当执行 CMD_RECV 命令后数据仍然保留，该位也为”1”
1	DISCON	当收到终止连接请求或终止连接过程已结束时，该位被置“1”
0	CON	当连接成功时，该位被置“1”

Sn_SR（端口 n 状态寄存器）[R] [0x0403，0x0503，0x0603，0x0703] [0x00]

该寄存器指示端口 n 的状态数值。端口 n 的主要状态如下图：



数值	符号	说明
0x00	SOCK_CLOSED	当向 Sn_CR 写入 CLOSE 命令产生超时中断或连接被终止，将出现 SOCK_CLOSED 状态。在 SOCK_CLOSED 状态不产生任何操作，释放所有连接资源
0x13	SOCK_INIT	当 Sn_MR 设为 TCP 模式时，向 Sn_CR 中写入 OPEN 命令时将出现 SOCK_INIT 状态。这是端口建立 TCP 连接的开始。在 SOCK_INIT 状态，Sn_CR 命令的类型决定了操作类型—TCP 服务器模式或 TCP 客户端模式
0x14	SOCK_LISTEN	当端口处于 SOCK_INIT 状态时，向 Sn_CR 写入 LISTEN 命令时将产生 SOCK_LISTEN 状态。相应的端口设置为 TCP 服务器模式，如果收到连接请求将改变为 ESTABLISHED 状态
0x17	SOCK_ESTABLISHED	当端口建立连接时将产生 SOCK_ESTABLISHED 状态，在这种状态下可以发送和接收 TCP 数据
0x1C	SOCK_CLOSE_WAIT	当收到来自对端的终止连接请求时，将产生 SOCK_CLOSE_WAIT 状态。在这种状态，从对端收到响应信息，但没有断开。当收到 DICON 或 CLOSE 命令时，连接断开

0x22	SOCK_UDP	当 Sn_MR 设为 UDP 模式，向 Sn_CR 写入 OPEN 命令时将产生 SOCK_UDP 状态，在这种状态下通信不需要与对端建立连接，数据可以直接发送和接收
0x32	SOCK_IPRAW	当 Sn_MR 设为 IPRAW 模式，向 Sn_CR 写入 OPEN 命令时将产生 SOCK_IPRAW 状态，在 IP RAW 状态， IP 层以上的协议将不处理。详情请参考"IPRAW"
0x42	SOCK_MACRAW	当 Sn_MR 设为 MACRAW 模式，向 S0_CR 写入 OPEN 命令时将产生 SOCK_MACRAW 状态。在 MAC RAW 状态，所有协议数据包都不处理，详情请参考"MAC RAW"
0x5F	SOCK_PPPoE	当 Sn_MR 设为 PPPoE 模式，向 S0_CR 写入 OPEN 命令时将产生 SOCK_PPPOE 状态

以下是端口状态改变时产生的状态。

数值	符号	说明
0x15	SOCK_SYNSENT	端口 n 在 SOCK_INIT 状态，向 Sn_CR 写入 CONNECT 命令时将出现 SOCK_SYNSENT 状态。如果连接成功，将自动改变为 SOCK_ESTABLISH
0x16	SOCK_SYNRECV	当接收到远程对端(客户端)发送的连接请求时，将出现 SOCK_SYNRECV 状态。响应请求后，状态改变为 SOCK_ESTABLISH
0x18	SOCK_FIN_WAIT	在连接终止过程中将出现这些状态。如果连接终止已完成，或产生了时间溢出中断，状态将自动被改变为 SOCK_CLOSED
0x1A	SOCK_CLOSING	
0x1B	SOCK_TIME_WAIT	
0x1D	SOCK_LAST_ACK	
0x11 0x21 0x31	SOCK_ARP	在 TCP 模式下发出连接请求或在 UDP 模式下发送数据时，当发出 ARP 请求以获得远程对端的物理地址时，将出现 SOCK_ARP 状态。如果收到 ARP 响应，将产生 SOCK_SYNSENT、SOCK_UDP 或 SOCK_ICMP 状态，以便下面的操作

Sn_PORT（端口 n 的端口号寄存器）[R/W] [0x0404～0x0405，0x0504～0x0505，0x0604～0x0605，0x0704～0x0705] [0x00]

该寄存器在 TCP 或 UDP 模式下设定对应端口的端口号。这些端口号必须在进行 OPEN 指令之前完成。

例：如端口 0 的端口号为 5000（0x1388），则

0x0404	0x0405
0x13	0x88

Sn_DHAR（端口 n 的目的物理地址寄存器）[RXW] [0x0406～0x040B，0x0506～0x050B，0x0606～0x060B，0x0706～0x070B] [0x00]

该寄存器设置每个端口的目的物理地址。

例：如端口 0 的目的物理地址为 08.DC.00.01.02.10，则

0x0406	0x0407	0x0408	0x0409	0x040A	0x040B
0x08	0xDC	0x00	0X01	0x02	0x0A

Sn_DIPR（端口 n 的目的 IP 地址寄存器）[R/W] [0x040C～0x040F，0x050C～0x050F，0x060C～0x060F，0x070C～0x070F] [0x00]

在 TCP 模式，该寄存器设置端口的目的 IP 地址。在主动模式（客户端模式），目的 IP 地址必需设置后才能执行连接(CONNECT)命令。在被动模式时（服务器模式），W5100 建立连接后，内部自动刷新目的 IP 地址。

在 UDP 模式，收到对端的 ARP 响应后，该寄存器才确定为用户写入的值。没有收到对端的 ARP 响应之前，该寄存器复位。

例：如端口 0 的目的 IP 地址为 192.168.0.11，则

0x040C	0x040D	0x040E	0x040F
192（0xC0）	168(0xA8)	0(0x00)	11(0x0B)

Sn_DPORT（端口 n 的目的端口号寄存器）[R/W] [0x0410～0x0411，0x0510～0x0511，0x0610～0x0611，0x0710～0x0711] [0x00]

在 TCP 模式，该寄存器设置端口的目的端口号。在主动模式下（客户端模式），目的端口号必需设置后再执行连接(CONNECT)命令。在被动模式下（服务器模式），W5100 建立连接后，内部自动刷新目的端口号。

在 UDP 模式，收到对端的 ARP 响应后，该寄存器才确定为用户写入的值。在没有收到对端的 ARP 响应之前，该寄存器复位。

例：如端口 0 的目的端口号为 5000（0x1388），则

0x0410	0x0411
0x13	0x88

Sn_MSS（端口 n 最大分片长度寄存器）[R/W] [0x0412～0x0413，0x0512～0x0513，0x0612～0x0613，0x0712～0x0713] [0x00]

在 TCP 模式，该寄存器设置端口的最大分片长度，如果 TCP 在被动模式(Passive Mode)下，该寄存器的值由其它部分设置。

例：设置端口 0 的 MSS 为 1460（0x05B4），则

0x0412	0x0413
0x05	0xB4

Sn_PROTO（端口 n 的 IP 协议寄存器）[R/W] [0x0414，0x0514，0x0614，0x0714] [0x00]

在 IP RAW 模式下，IP 协议寄存器用来设置 IP 数据包头的协议字段（Protocol Field）的值。IANA 预先注册了一些协议号可以使用。IANA 网站可查出全部的 IP 层协议代码。参考 IANA 的在线文档（http://www.iana.org/assignments/protocol-numbers）

例：ICMP 的协议号为 0x01，IGMP 的协议号为 0x02。

Sn_TOS（端口 n 的 IP 服务类型寄存器）[R/W] [0x0415，0x0515，0x0615，0x0715] [0x00]

该寄存器用来设置 IP 数据包头中的服务类型(TOS)字段的值。

Sn_TTL（端口 n 的 IP 数据包生存期寄存器）[R/W] [0x0416，0x0516，0x0616，0x0716] [0x80]

该寄存器用来设置 IP 数据包头中的生存期(TTL)字段的值。

Sn_TX_FSR（端口 n 发送存储器剩余空间寄存器）[R] [0x0420～0x0421，0x0520～0x0521，0x0620～0x0621，0x0720～0x0721] [0x0800]

该寄存器指示用户可使用的发送数据空间的大小。在发送数据时，用户必需先检查剩余空间的大小，然后控制发送数据的字节数。检查该寄存器时，必需先读高字节（0x0420，0x0520，0x0620，0x0720），然后再读低字节（0x0421，0x0521，0x0621，0x0721）。

例：端口 0 的剩余空间为 2048 字节（0x0800），则

0x0420	0x0421
0x08	0x00

端口发送总空间的大小由发送存储器空间寄存器(TMSR)确定。在数据发送处理过程中，剩余空间的大小将因写入数据而减少，发送完成后自动增加。

Sn_TX_RR（端口 n 发送存储器读指针寄存器）[R] [0x0422～0x0423，0x0522～0x0523，0x0622～0x0623，0x0722～0x0723] [0x0000]

该寄存器指示端口在发送过程完成后发送存储器的当前位置。当端口 n 的命令寄存器收到 SEND 命令，从当前 Sn_TX_RR 到 Sn_TX_WR 的数据将发送出去，发送完成后，Sn_TX_RR 的值自动改变。因此发送完成后，Sn_TX_RR 的值与 Sn_TX_WR 的值相等。用户读取该寄存器时，必须先读高字节(0x0422，0x0522，0x0622，0x0722)，然后再读低字节(0x0423，0x0523，0x0623，0x0723)。

Sn_TX_WR（端口 n 传输写指针寄存器）[R/W] [0x0424～0x0425，0x0524～0x0525，0x0624～0x0625，0x0724～0x0725] [0x0000]

该寄存器指示在向 TX 存储器写入数据时的地址。用户读取该寄存器时，必须先读高字节(0x0424，0x0524，0x0624，0x0724)，然后再读低字节(0x0425，0x0525，0x0625，0x0725)。

例：如 0 端口的 S0_TX_WR 的值为 2048（0x0800），则

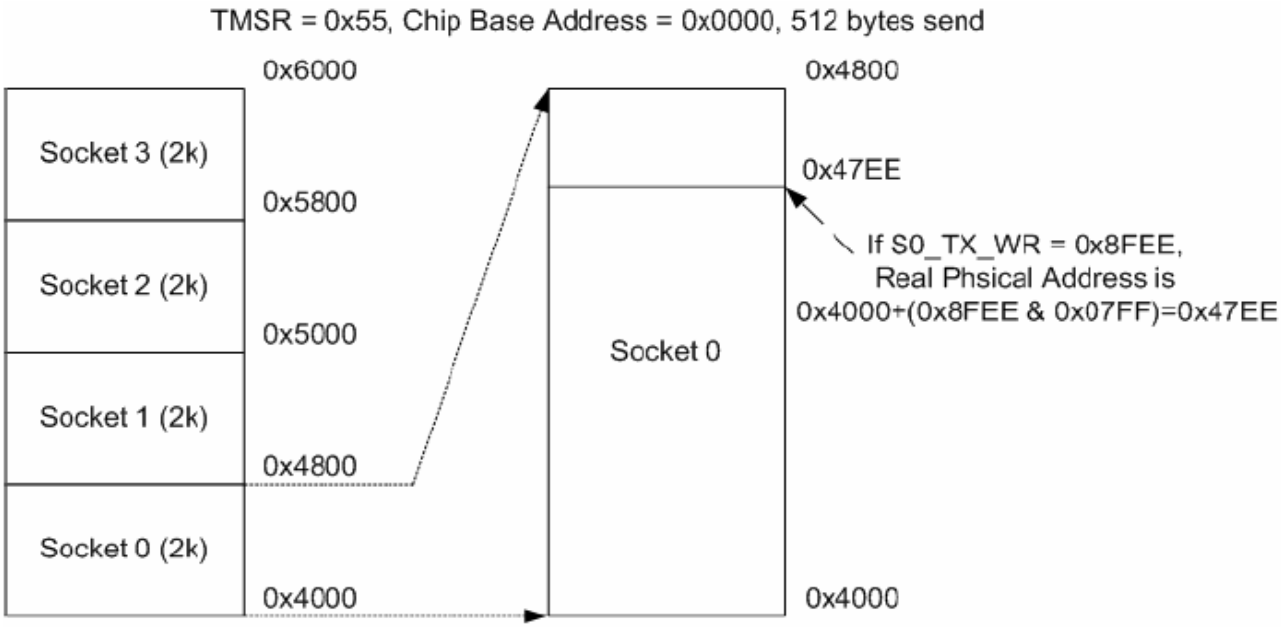
0x0424	0x0425
0x08	0x00

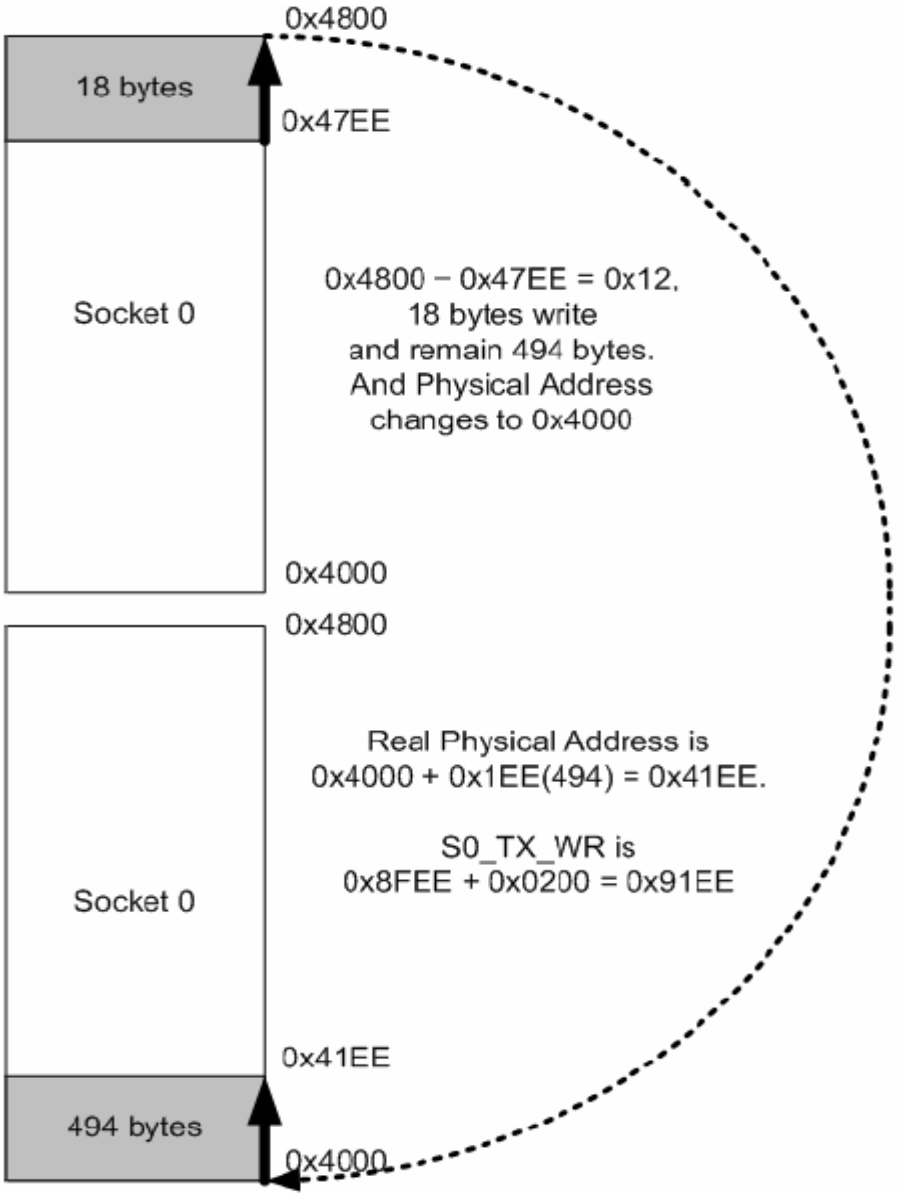
从该寄存器读出的地址值不是实际访问的物理地址，实际物理地址需要按照以下方法计算：

1. 端口 n 的发送存储器的基地址（以下称 gSn_TX_BASE）及端口 n 的发送屏蔽地址（以下称 gSn_TX_MASK）由 TMSR 的值确定。
2. 将 Sn_TX_WR 与 gSn_TX_MASK 的值作"位与"运算，得到该端口在存储区间范围内的地址偏移量（以下称 get_offset）
3. 将偏移量 get_offset 与基地址 gSn_TX_BASE 相加，得到实际访问的物理地址(以下称 get_start_address)。

从地址 get_start_address 写入要发送的数据。（在写入过程中，如果物理地址到达端口设定的高限地址，此时，先 将数据写入高限地址，然后将当前的物理地址改成基地址（gSn_TX_BASE），并从基地址继续写入剩余的数据）。

写完数据后，将 Sn_TX_WR 中的值加上写入的数据长度，然后再写入到 Sn_TX_WR，以指示发送数据的长度，最后才在 Sn_CR（端口 n 指令寄存器）中写入 SEND 指令。（详情参照"5.2.1.1TCP 服务器"中的发送篇）。





Sn_RX_RSR（接收数据字节数寄存器）[R] [0x0426～0x0427，0x0526～0x0527，0x0626～0x0627，0x0726～0x0727] [0x0000]

该寄存器指示端口接收数据缓冲区中接收数据的字节数。这个值是由 Sn_RX_RR 与 Sn_RX_WR 的值计算得出的，向端口 n 命令寄存器（Sn_CR）写入 RECV 命令后寄存器的值自动改变，并可以接收远程对端的数据。读取寄存器时，必须先读高字节（0x0426，0x0526，0x0626，0x0726），然后再读低字节（0x0427，0x0527，0x0627，0x0727）。

例：如端口 0 的 S0_RX_RSR 值为 2048（0x0800），则

0x0426	0x0427
0x08	0x00

该寄存器的最大值由 RMSR 确定。

Sn_RX_RD（端口 n 接收缓冲区读指针寄存器）[R/W] [0x0428～0x0429，0x0528～0x0529，0x0628～0x0629，0x0728～0x0729] [0x0000]

该寄存器指示端口接收过程完成后的读地址信息。读取该寄存器时，必须先读高字节(0x0428，0x0528，0x0628，0x0728)，然后再读低字节(0x0429，0x0529，0x0629，0x0729)。

例：如端口 0 的 S0_RX_RR 值为 2048（0x0800），则

0x0428	0x0429
0x08	0x00

从该寄存器读出的地址值不是实际访问的物理地址。实际物理地址按以下方法计算：

1. 端口 n 接收缓冲区 RX 的基地址（以下称 gSn_RX_BASE）及端口 n 的 RX 屏蔽地址（以下称 gSn_RX_MASK）由 RMSR 的值确定。（详情可参照 5.1 初始化设置部分）
2. 将 Sn_RX_WR 与 gSn_RX_MASK 的值作"位与"运算，得到该端口接收存储区内的偏移量（以下称 get_offset）
3. 将偏移量 get_offset 与基地址 gSn_RX_BASE 相加，得到实际访问的物理地址（以下称 get_start_address）

现在可以根据物理地址(get_start_address)读取数据。（在读数据过程中，如果物理地址到达该端口设定的高限地址，此时，先读取高限地址的数据，然后将物理地址改为基地址 gSn_RX_BASE，并从基地址继续读取剩余的数据）。

读完所有的数据后，将 Sn_RX_RR 的值加上读取的数据长度，然后写入到 Sn_RX_RR，最后向端口 n 的指令寄存器（Sn_CR）写入 RECV 命令。（详细内容请参照 TCP 服务器模式中的接收部分）。

5. 功能描述

通过设置寄存器和存储器，W5100 就可以进行 Internet 连接。这一章叙述操作过程。

5.1 初始化

基本设置

W5100 的操作需要设置以下寄存器的参数：

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. 重发计数寄存器（RCR）

以上寄存器的详细资料请参考“寄存器描述”一节。

设置网络信息

下面的寄存器是关于网络的基本配置，需要根据网络环境来进行设置。

1. 网关地址寄存器（GAR）
2. 本机物理地址寄存器（SHAR）
3. 子网掩码寄存器（SUBR）
4. 本机 IP 地址寄存器（SIPR）

本机物理地址寄存器（SHAR）的地址是 MAC 层的硬件地址，这是生产商指定使用的地址。MAC 地址可以由 IEEE 指定。（详情请参照 IEEE 网站）

设置端口存储器信息

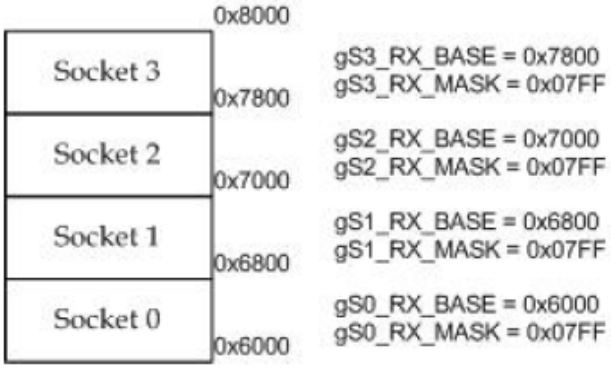
这一步设置端口 Tx/Rx 存储器信息，每个端口的基地址和屏蔽地址在这里确定并保存。

```
In case of, assign 2K rx memory per socket.
{
        RMSR = 0x55;                        // assign 2K rx memory per socket.
        gS0_RX_BASE = chip_base_address + RX_memory_base_address(0x6000);
        gS0_RX_MASK = 2K – 1 ;              // 0x07FF, for getting offset address within assigned socket 0 RX memory.
        gS1_RX_BASE = gS0_BASE + (gS0_MASK + 1);
        gS1_RX_MASK = 2K – 1 ;
        gS2_RX_BASE = gS1_BASE + (gS1_MASK + 1);
        gS2_RX_MASK = 2K – 1 ;
        gS3_RX_BASE = gS2_BASE + (gS2_MASK + 1);
        gS3_RX_MASK = 2K – 1 ;
```

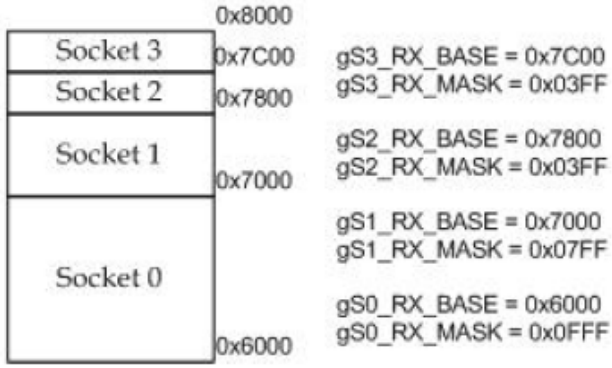
```
        TMSR = 0x55;                    // assign 2K tx memory per socket.
        Same method, set gS0_TX_BASE, gS0_TX_MASK, gS1_TX_BASE, gS1_TX_MASK,
        gS2_TX_BASE, gS2_TX_MASK, gS3_TX_BASE and gS3_TX_MASK.
}

In case of, assign 4K,2K,1K,1K.
{
        RMSR = 0x06;                    // assign 4K,2K,1K,1K rx memory per socket.
        gS0_RX_BASE = chip_base_address + RX_memory_base_address(0x6000);
        gS0_RX_MASK = 4K – 1 ; // 0x0FFF, for getting offset address within assigned socket 0 RX memory.
        gS1_RX_BASE = gS0_BASE + (gS0_MASK + 1);
        gS1_RX_MASK = 2K – 1 ; // 0x07FF
        gS2_RX_BASE = gS1_BASE + (gS1_MASK + 1);
        gS2_RX_MASK = 1K – 1 ; // 0x03FF
        gS3_RX_BASE = gS2_BASE + (gS2_MASK + 1);
        gS3_RX_MASK = 1K – 1 ; // 0x03FF
        TMSR = 0x06;                    // assign 4K,2K,1K,1K rx memory per socket.
        Same method, set gS0_TX_BASE, gS0_TX_MASK, gS1_TX_BASE, gS1_TX_MASK,
        gS2_TX_BASE, gS2_TX_MASK, gS3_TX_BASE and gS3_TX_MASK.
}
```

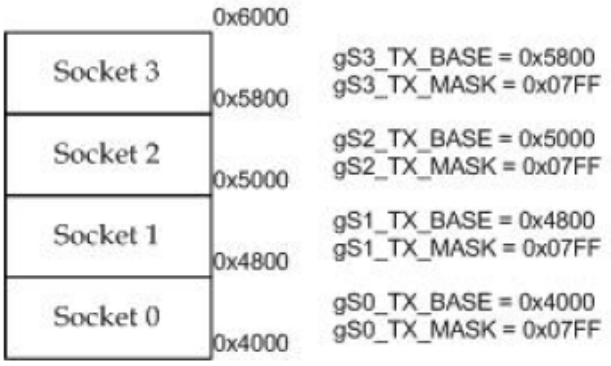
RMSR = 0x55, Chip Base Address = 0x0000



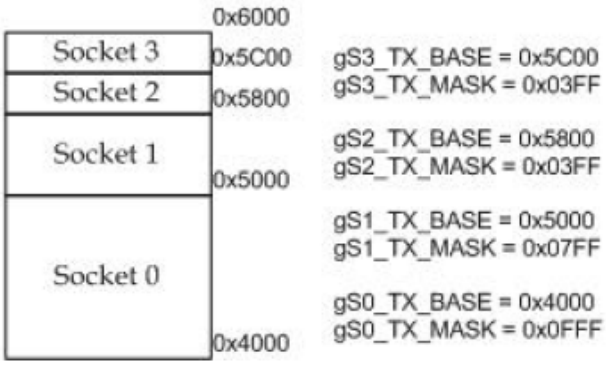
RMSR = 0x06



TMSR = 0x55, Chip Base Address = 0x0000



TMSR = 0x06

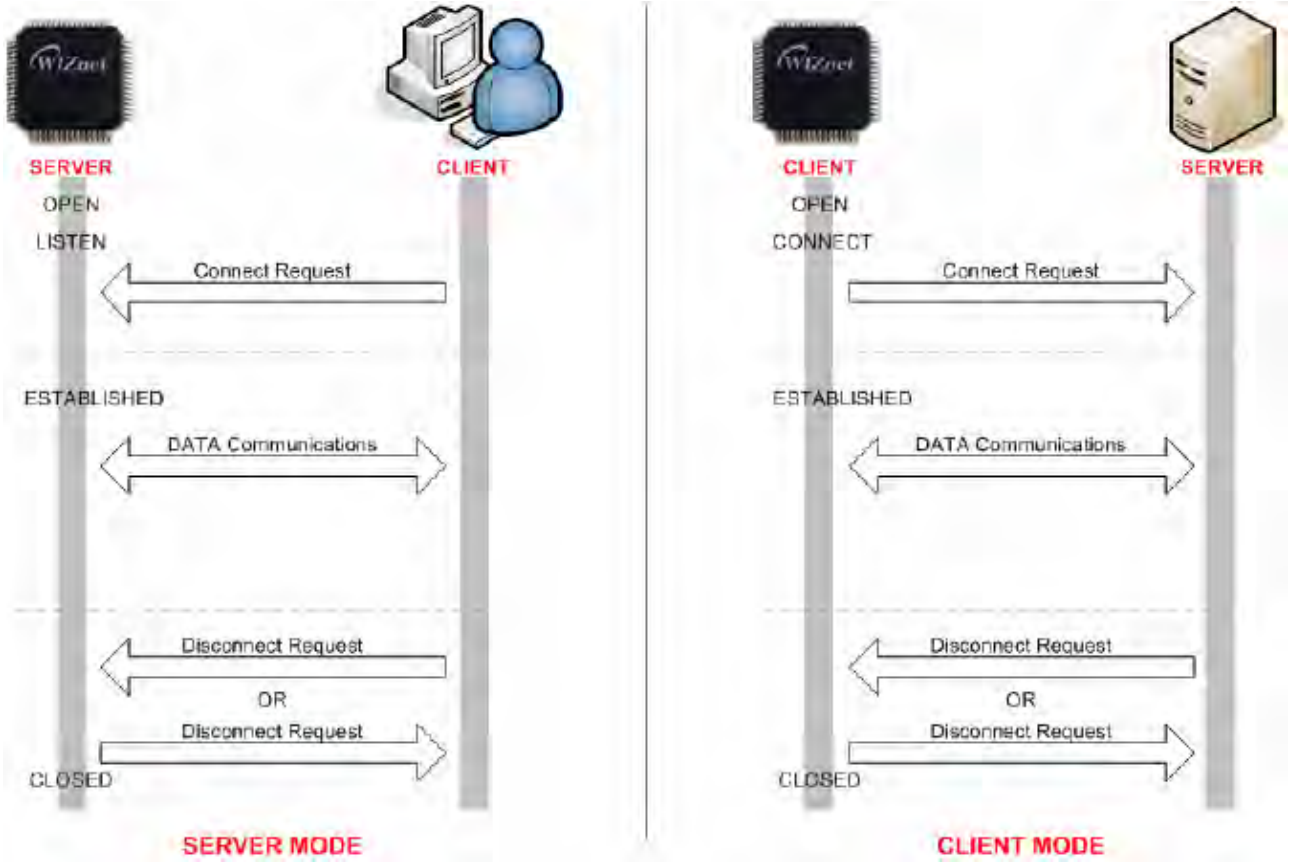


5.2 数据通信

通过 TCP、UDP、IP_RAW 和 MAC_RAW 模式进行数据通信。在端口 n 的模式寄存器（Sn_MR）的协议类型选择通信模式（W5100 总共支持 4 个端口）。

5.2.1 TCP

TCP 是以连接为基础的通信方式，它必须首先建立连接，然后利用连接的 IP 地址和端口号进行数据传输。TCP 有两种连接方式：一种是服务器模式（被动开启），即等待接收连接请求以建立连接；另一种是客户端模式（主动开启），即发送连接请求到服务器。



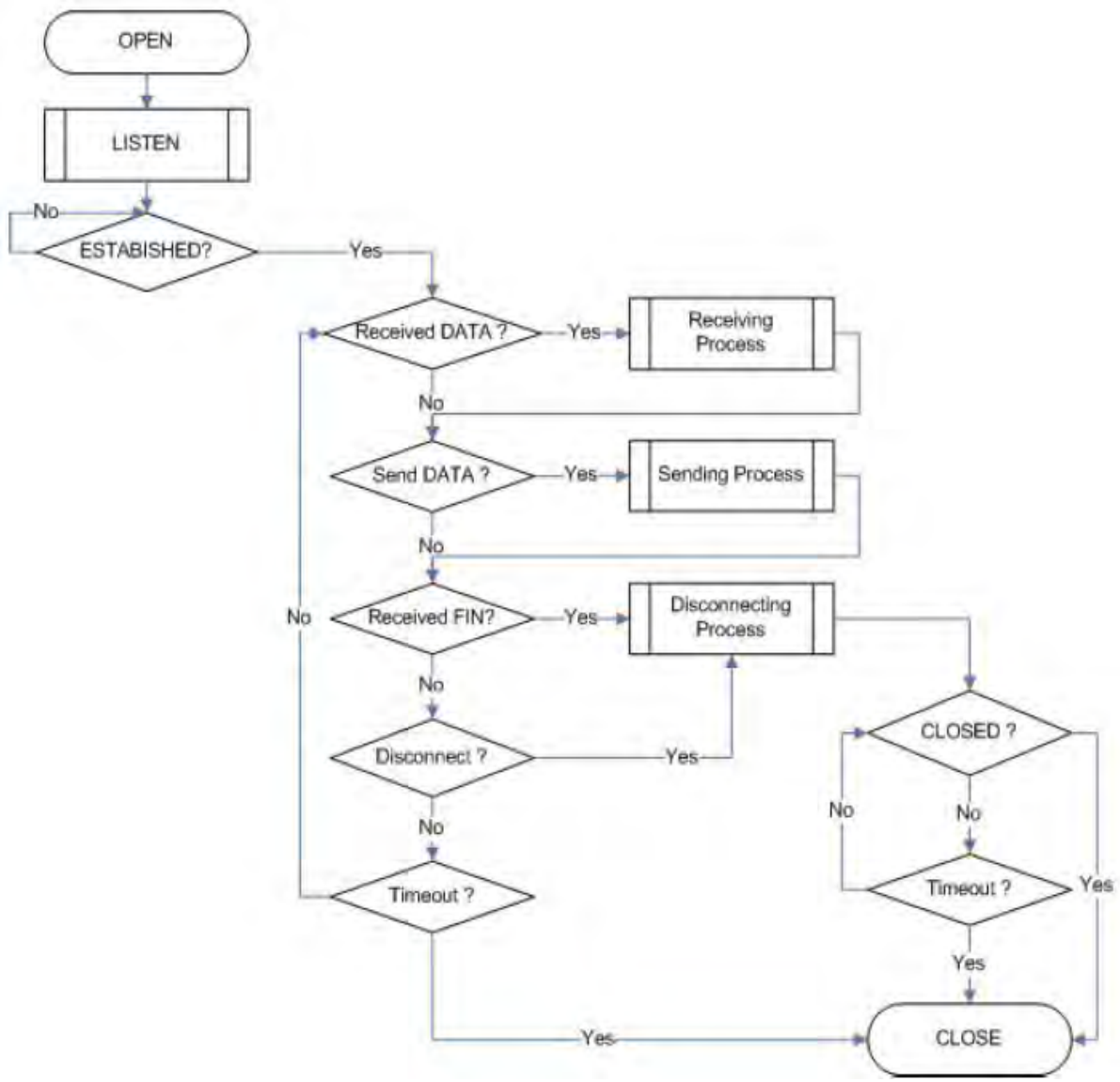
服务器模式　　客户端模式

5.2.1.1 服务器模式

端口初始化

初始化一个端口需要设置运行模式和端口号，并在端口命令寄存器打开（OPEN）端口。端口初始化涉及到以下寄存器：

- 端口 n 模式寄存器（Sn_MR）
- 本机端口 n 的端口号（Sn_PORT）
- 端口 n 命令寄存器（Sn_CR）



将端口 n 初始化为 TCP 模式

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

侦听

在命令寄存器设置 LISTEN 命令，涉及到寄存器是。

- 端口 n 命令寄存器（Sn_CR）

```
{
    Sn_CR = LISTEN;                    /* listen socket */
    if (Sn_SR != SOCK_LISTEN)
    {
        Sn_CR = CLOSE;
        goto START;                    // check socket status
    }
}
```

连接成功？

当接收到远程对端发来的连接请求（SOCK_SYNRECV 状态），W5100 将回复 ACK 数据包，并将状态改变成 SOCK_ESTABLISHED。该状态可以用以下方法检测到。

第一种方法：

```
{
    If (Sn_IR(CON bit) == '1')
        goto ESTABLISHED stage;
    /* In this case, if the interrupt of Socket n is activated, interrupt occurs. Refer to Interrupt Register(IR),
    Interrupt Mask Register (IMR) and Socket n Interrupt Register (Sn_IR). */
}
```

第二种方法：

```
{
    If (Sn_SR == SOCK_ESTABLISHED)
        goto ESTABLISHED stage;
}
```

一旦建立连接，即可以发送和接收数据。

连接成功：收到数据？

通过以下方法可检测到是否收到从远程对端发来的数据。

第一种方法：

```
{
    If (Sn_IR(RECV bit) == '1')
        goto Receiving Process stage;
    /* In this case, if the interrupt of Socket n is activated, interrupt occurs. Refer to Interrupt
    Register(IR), Interrupt Mask Register (IMR) and Socket n Interrupt Register (Sn_IR). */
}
```

第二种方法：

```
{
    if (Sn_RX_RSR != 0x0000)
        goto Receiving Process stage;
}
```

连接成功：读取数据过程

数据读取过程如下：

```
{
    /* first, get the received size */
    get_size = Sn_RX_RSR;
    /* calculate offset address */
    get_offset = Sn_RX_RD & gSn_RX_MASK;
    /* calculate start address(physical address) */
    get_start_address = gSn_RX_BASE + get_offset;
    /* if overflow socket RX memory */
    if ( (get_offset + get_size) > (gSn_RX_MASK + 1) )
    {
        /* copy upper_size bytes of get_start_address to destination_addr */
        upper_size = (gSn_RX_MASK + 1) – get_offset;
        memcpy(get_start_address, destination_addr, upper_size);
        /* update destination_addr*/
        destination_addr += upper_size;
        /* copy left_size bytes of gSn_RX_BASE to destination_addr */
        left_size = get_size – upper_size;
        memcpy(gSn_RX_BASE, destination_addr, left_size);
    }
    else
    {
        /* copy get_size bytes of get_start_address to destination_addr */
        memcpy(get_start_address, destination_addr, get_size);
```

```
        }
        /* increase Sn_RX_RD as length of get_size */
        Sn_RX_RD += get_size;
        /* set RECV command */
        Sn_CR = RECV;
}
```

连接成功：发送数据? / 发送过程

数据发送过程如下：

```
{
        /* first, get the free TX memory size */
FREESIZE:
        get_free_size = Sn_TX_FSR;
        if (get_free_size < send_size)
                goto FREESIZE;
        /* calculate offset address */
        get_offset = Sn_TX_WR & gSn_TX_MASK;
        /* calculate start address(physical address) */
        get_start_address = gSn_TX_BASE + get_offset;
        /* if overflow socket TX memory */
        if ( (get_offset + send_size) > (gSn_TX_MASK + 1) )
        {
                /* copy upper_size bytes of source_addr to get_start_address */
                upper_size = (gSn_TX_MASK + 1) – get_offset;
                memcpy(source_addr, get_start_address, upper_size);
                /* update source_addr*/
                source_addr += upper_size;
                /* copy left_size bytes of source_addr to gSn_TX_BASE */
                left_size = send_size – upper_size;
                memcpy(source_addr, gSn_TX_BASE, left_size);
        }
        else
        {
                /* copy send_size bytes of source_addr to get_start_address */
                memcpy(source_addr, get_start_address, send_size);
        }
        /* increase Sn_TX_WR as length of send_size */
        Sn_TX_WR += send_size;
        /* set SEND command */
        Sn_CR = SEND;
}
```

连接成功：接收完成?

等待接收远程对端发来的终止连接请求。可以通过以下方法检测远程对端发送的终止连接请求。

第一种方法：

```
{
    If (Sn_IR(DISCON bit) == '1')
        goto CLOSED stage;
    /* In this case, if the interrupt of Socket n is activated, interrupt occurs. Refer to Interrupt
    Register(IR), Interrupt Mask Register (IMR) and Socket n Interrupt Register (Sn_IR). */
}
```

第二种方法：

```
{
    If (Sn_SR == SOCK_CLOSE_WAIT)
        goto CLOSED stage;
}
```

连接成功：断开连接？/ 断开过程

检查是否有终止连接的请求。终止连接的处理过程如下：

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

连接成功：关闭端口？

没有连接状态。可按检查如下：

第一种方法：

```
{
    If (Sn_IR(DISCON bit) == '1')
        goto CLOSED stage;
    /* In this case, if the interrupt of Socket n is activated, interrupt occurs. Refer to Interrupt
    Register(IR), Interrupt Mask Register (IMR) and Socket n Interrupt Register (Sn_IR). */
}
```

第二种方法：

```
{
    If (Sn_SR == SOCK_CLOSED)
        goto CLOSED stage;
}
```

连接成功：超时

在数据接收或连接终止过程中，因远程对端的故障而终止连接，数据传输不能正常进行。这种情况下，等待一段时间后，将产生超时错误。

第一种方法：

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

第二种方法：

```
{
    If (Sn_SR == SOCK_CLOSED)
        goto CLOSED stage;
}
```

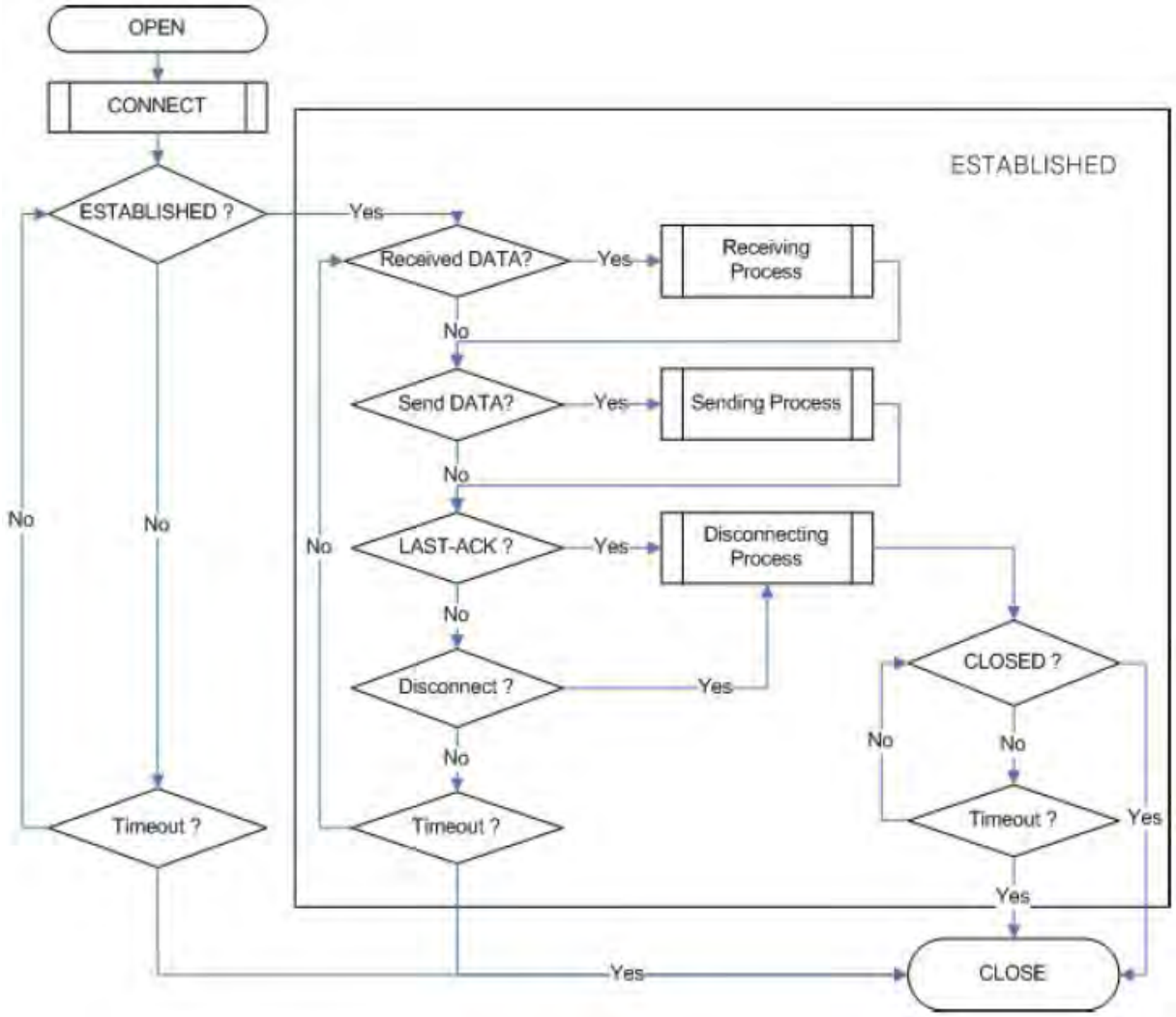
端口关闭

数据交换完成后，端口需要终止连接；或产生超时错误时，端口必须关闭；或因非正常操作而使端口强迫终止连接。在这几种情况下都必须执行端口关闭，操作如下：

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

5.2.1.2 客户端模式

处理过程如下：



端口初始化

请参照 5.2.1.1 服务器模式（与服务器程序同）

连结

发送连接请求到远端主机服务器，程序如下：

```
{
    /* Write the value of server_ip, server_port to the Socket n Destination IP Address
    Register(Sn_DIPR), Socket n Destination Port Register(Sn_DPORT). */
    Sn_DIPR = server_ip;
    Sn_DPORT = server_port;
    /* set CONNECT command */
    Sn_CR = CONNECT;
}
```

建立连接?

可按照下面方法检测端口是否连接成功。

第一种方法：

```
{
    If (Sn_IR(CON bit) == '1')
        goto ESTABLISHED stage;
    /* In this case, if the interrupt of Socket n is activated, interrupt occurs. Refer to Interrupt
    Register(IR), Interrupt Mask Register (IMR) and Socket n Interrupt Register (Sn_IR). */
}
```

第二种方法：

```
{
    If (Sn_SR == SOCK_ESTABLISHED)
        goto ESTABLISHED stage;
}
```

超时

当远程对端没有响应而产生超时并使端口关闭，检测如下。

第一种方法：

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

第二种方法：

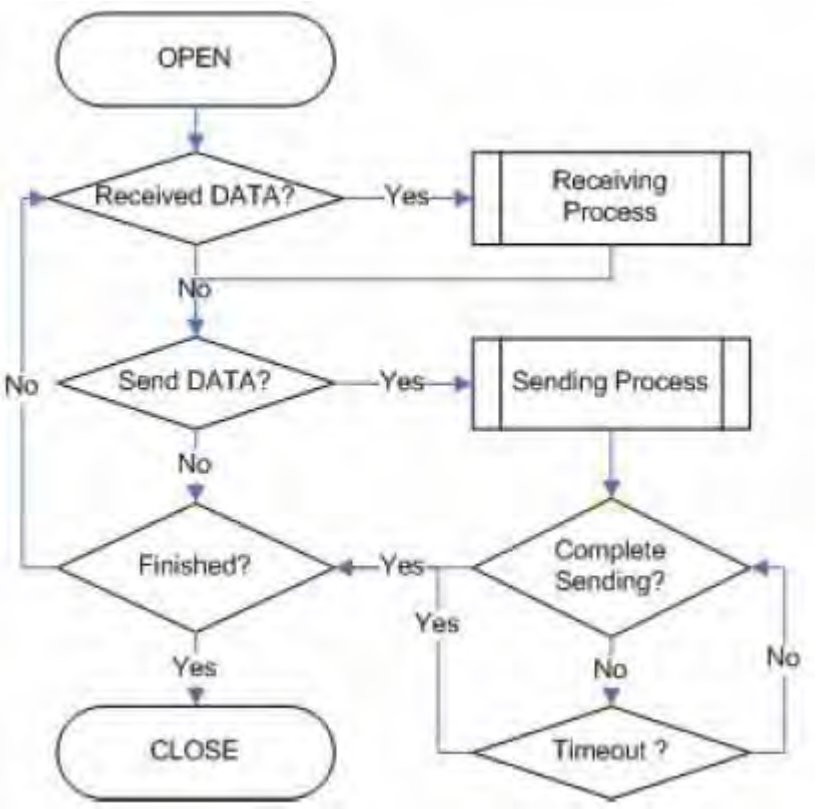
```
{
    If (Sn_SR == SOCK_CLOSED)
        goto CLOSED stage;
}
```

建立连接

请参照 5.2.1.1 服务器一节（与服务器处理过程相同）

5.2.2 UDP

UDP 是一种不可靠的、无连接的数据传输方式。它不需要建立连接就可以进行数据传输，因此 UDP 的信息可能丢失、覆盖或反转。由于数据包传输的速度可能比较快，接收方可能无法及时处理数据包，因此，用户在应用层必须确保数据传输的可靠性。UDP 的传输过程如下：



端口初始化

UDP 的端口初始化过程如下：

```
{
START:
        /* sets UDP mode */
        Sn_MR = 0x02;
        /* sets source port number */
        /* The value of Source Port can be appropriately delivered when remote HOST knows it. */
        Sn_PORT = source_port;
        /* sets OPEN command */
        Sn_CR = OPEN;
        /* Check if the value of Socket n Status Register(Sn_SR) is SOCK_UDP. */
        if (Sn_SR != SOCK_UDP)
        {
                Sn_CR = CLOSE;
                goto START;
        }
}
```

接收到数据?

可通过下面方法检测是否接收到远程数据。

第一种方法：

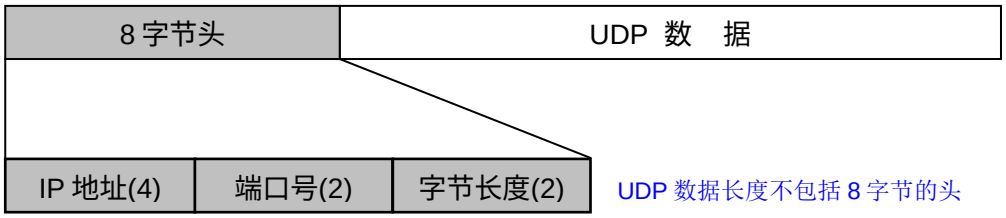
```
{
    if (Sn_RX_RSR != 0x0000)
        goto Receiving Process stage;
}
```

第二种方法：

```
{
    If (Sn_IR(RECV bit) == '1')
        goto Receiving Process stage;
    /* In this case, if the interrupt of Socket n is activated, interrupt occurs. Refer to Interrupt
    Register(IR), Interrupt Mask Register (IMR) and Socket n Interrupt Register (Sn_IR). */
}
```

接收处理

接收的数据可以按照下面的方法处理。在 UDP 传输时，在接收的数据前面有一个 8 字节的头。其结构如下：




```
{
    /* first, get the received size */
    get_size = Sn_RX_RSR;
    /* calculate offset address */
    get_offset = Sn_RX_RD & gSn_RX_MASK;
    /* calculate start address(physical address) */
    get_start_address = gSn_RX_BASE + get_offset;
    /* read head information (8 bytes) */
    header_size = 8;
    /* if overflow socket RX memory */
    if ( (get_offset + header_size) > (gSn_RX_MASK + 1) )
    {
```

```
        /* copy upper_size bytes of get_start_address to header_addr */
        upper_size = (gSn_RX_MASK + 1) – get_offset;
        memcpy(get_start_address, header_addr, upper_size);
        /* update header_addr*/
        header_addr += upper_size;
        /* copy left_size bytes of gSn_RX_BASE to header_addr */
        left_size = header_size – upper_size;
        memcpy(gSn_RX_BASE, header_addr, left_size);
        /* update get_offset */
        get_offset = left_size;
}
else
{
        /* copy header_size bytes of get_start_address to header_addr */
        memcpy(get_start_address, header_addr, header_size);
        /* update get_offset */
        get_offset += header_size;
}
/* update get_start_address */
get_start_address = gSn_RX_BASE + get_offset;
/* 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 ( (get_offset + get_size) > (gSn_RX_MASK + 1) )
{
        /* copy upper_size bytes of get_start_address to destination_addr */
        upper_size = (gSn_RX_MASK + 1) – get_offset;
        memcpy(get_start_address, destination_addr, upper_size);
        /* update destination_addr*/
        destination_addr += upper_size;
        /* copy left_size bytes of gSn_RX_BASE to destination_addr */
        left_size = get_size – upper_size;
        memcpy(gSn_RX_BASE, destination_addr, left_size);
}
else
{
        /* copy get_size bytes of get_start_address to destination_addr */
        memcpy(get_start_address, destination_addr, get_size);
}
/* increase Sn_RX_RD as length of get_size+header_size */
```

```
        Sn_RX_RD = Sn_RX_RD + get_size + header_size;
        /* set RECV command */
        Sn_CR = RECV;
}
```

发送数据？/发送处理

数据发送过程如下：

```
{
        /* first, get the free TX memory size */
FREESIZE:
        get_free_size = Sn_TX_FSR;
        if (get_free_size < send_size)
                goto FREESIZE;
        /* Write the value of remote_ip, remote_port to the Socket n Destination IP Address
        Register(Sn_DIPR), Socket n Destination Port Register(Sn_DPORT). */
        Sn_DIPR = remote_ip;
        Sn_DPORT = remote_port;
        /* calculate offset address */
        get_offset = Sn_TX_WR & gSn_TX_MASK;
        /* calculate start address(physical address) */
        get_start_address = gSn_TX_BASE + get_offset;
        /* if overflow socket TX memory */
        if ( (get_offset + send_size) > (gSn_TX_MASK + 1) )
        {
                /* copy upper_size bytes of source_addr to get_start_address */
                upper_size = (gSn_TX_MASK + 1) – get_offset;
                memcpy(source_addr, get_start_address, upper_size);
                /* update source_addr*/
                source_addr += upper_size;
                /* copy left_size bytes of source_addr to gSn_TX_BASE */
                left_size = send_size – upper_size;
                memcpy(source_addr, gSn_TX_BASE, left_size);
        }
        else
        {
                /* copy send_size bytes of source_addr to get_start_address */
                memcpy(source_addr, get_start_address, send_size);
        }
        /* increase Sn_TX_WR as length of send_size */
        Sn_TX_WR += send_size;
```

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

发送完成?

在发送(SEND)命令后，可以通过下面的方法检测数据是否全部发送完成。

第一种方法：

```
{
    If (Sn_CR == 0x00)
        transmission is completed;
}
```

第二种方法：

```
{
    If (Sn_IR(SENDOK bit) == '1')
        transmission is completed;
    /* In this case, if the interrupt of Socket n is activated, interrupt occurs. Refer to Interrupt
    Register(IR), Interrupt Mask Register (IMR) and Socket n Interrupt Register (Sn_IR). */
}
```

超时

当远程对端不存在、或数据传输不正常时将产生超时错误。可以通过下面方法检测：

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

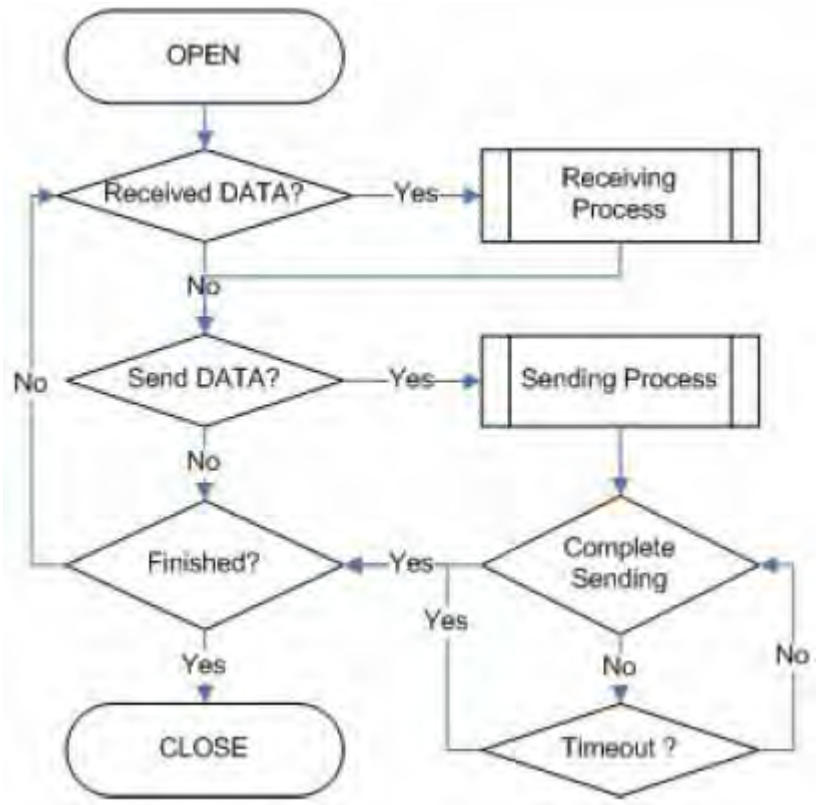
完成?/ 端口关闭

当发送操作全部完成后，关闭端口。

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

5.2.3 IP RAW

W5100 不能支持传输层的某些协议，如 ICMP 或 IGMP，可使用 IP RAW 模式来实现。处理过程如下：



端口初始化

IP RAW 端口的初始化过程如下：

```
{
START:
        /* sets IP raw mode */
        Sn_MR = 0x03;
        /* sets Protocol value */
        /* The value of Protocol is used in Protocol Field of IP Header.
        For the list of protocol identification number of upper classification, refer to on line
        documents of IANA (http://www.iana.org/assignments/protocol-numbers). */
        Sn_PROTO = protocol_value;
        /* sets OPEN command */
        Sn_CR = OPEN;
        /* Check if the value of Socket n Status Register(Sn_SR) is SOCK_IPRAW. */
        if (Sn_SR != SOCK_IPRAW)
        {
                Sn_CR = CLOSE;
                goto START;
        }
}
```

IP RAW 收到数据？

与 UDP 相同，请参照 5.2.2 UDP

接收数据处理

除了数据包头信息和大小与 UDP 不同，其它与 UDP 相同，参照 5.2.2 UDP。IP RAW 数据前面有 6 字节的数据，其结构如下：

IP RAW 发送数据？/发送处理

除了不需要远程端口号以外，其它与 UDP 相同。参照 5.2.2 UDP。

发送完成

超时

完成?/端口关闭

处理过程与 UDP 相同。参照 5.2.2 UDP。

5.2.4 MAC raw

只有端口 0 支持 MAC Raw 功能。

端口初始化

将端口 0 初始化为 MAC Raw 的过程如下：

```
{
START:
        /* sets MAC raw mode */
        Sn_MR = 0x04;
        /* sets OPEN command */
        Sn_CR = OPEN;
        /* Check if the value of Socket n Status Register(Sn_SR) is SOCK_MACRAW. */
        if (Sn_SR != SOCK_MACRAW)
        {
                Sn_CR = CLOSE;
                goto START;
        }
}
```

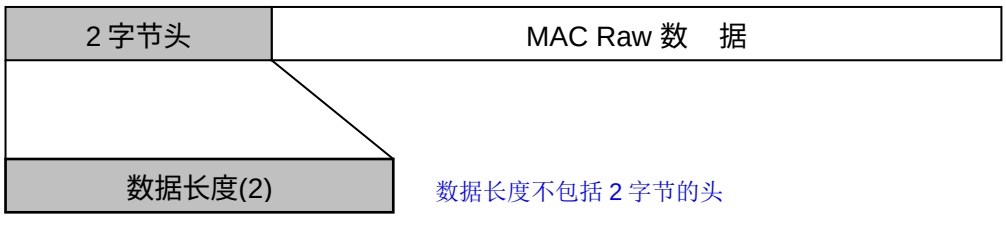
收到数据？

处理过程与 UDP 相同，参考 5.2.2 UDP

接收数据处理

MAC Raw 收到的是以太网的数据包，并带有数据包长度信息。

在 MAC Raw 数据包中，前面有两个字节的头，头的结构如下：



发送数据/发送处理

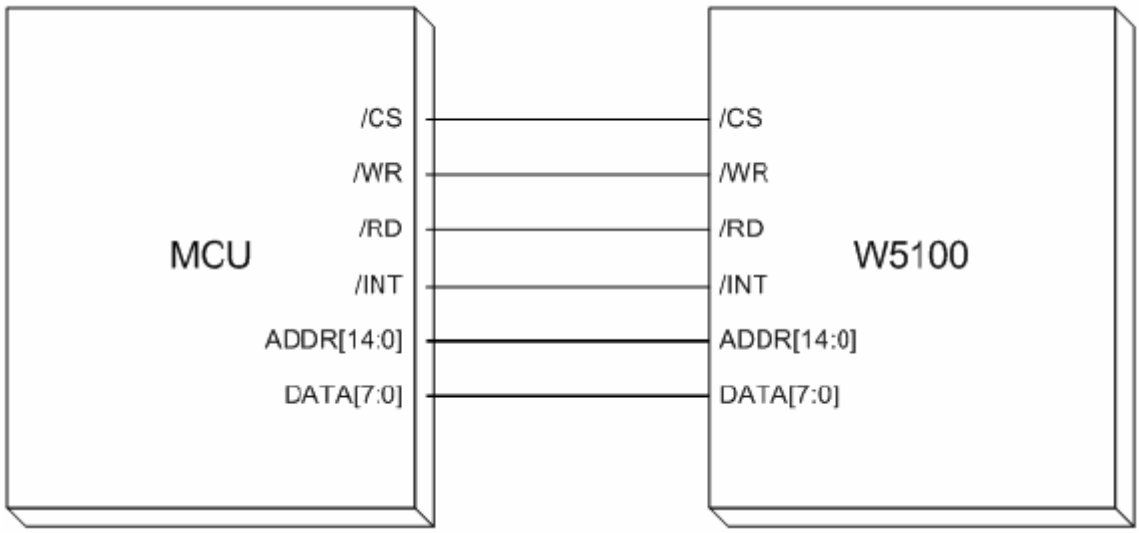
与 UDP 的处理相同，只是不需要远程端口信息。参考 5.2.2 UDP。

6. 应用资料

W5100 有三种方式与 MCU 接口：直接总线接口、间接总线接口和 SPI 总线接口。

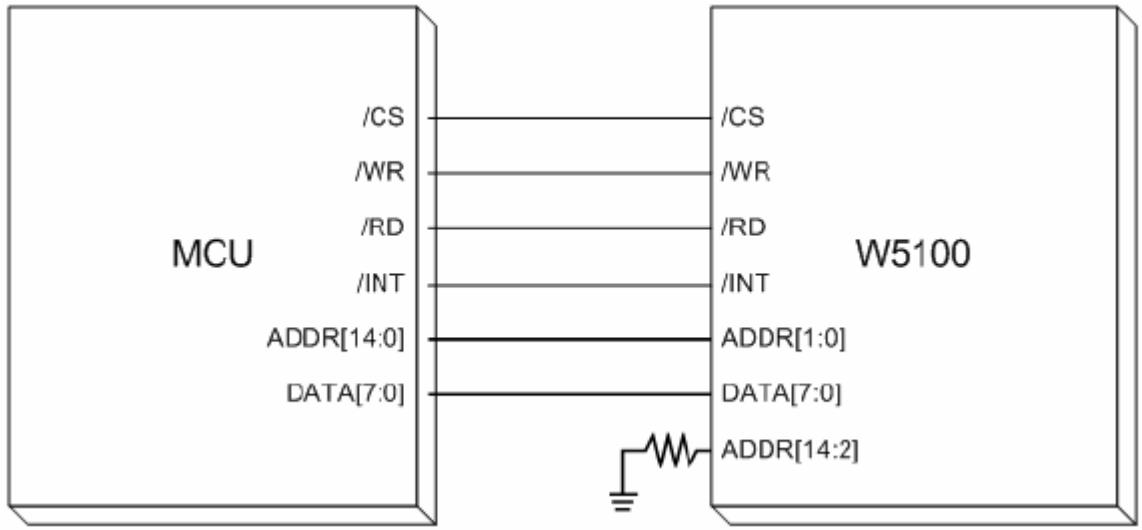
6.1 直接总线接口

直接总线接口采用 15 位地址线，8 位数据线，另加/CS，/RD，/WR 及/INT 等信号线。



6.2 间接总线接口

间接总线接口采用 2 位地址线，8 位数据线，另加/CS，/RD，/WR 及/INT 等信号线。[14:2]没用到的地址线经过电阻接地。



间接总线接口相关的寄存器说明如下：

<table>
<tr><th>数值</th><th>符号</th><th>说明</th></tr>
<tr><td>0x00</td><td>MR</td><td>它选择间接总线接口模式，以及地址自动增加。详情参照第 4 章寄存器说明。</td></tr>
<tr><td>0x01
0x02</td><td>IDM_AR0
IDM_AR1</td><td>间接总线模式下的地址寄存器。只在大端模式（Big-endian）下使用。
<table>
<tr><th>0x01</th><th>0x02</th></tr>
<tr><td>IDM_AR0：MSB</td><td>IDM_AR1：LSB</td></tr>
</table>
例：读取端口 0 的命令寄存器 S0_CR（0x0401），则
<table>
<tr><th>0x01（IDM_AR0）</th><th>0x02（IDM_AR1）</th></tr>
<tr><td>0x04</td><td>0x01</td></tr>
</table>
</td></tr>
<tr><td>0x03</td><td>IDM_DR</td><td>间接总线接口模式下的数据寄存器</td></tr>
</table>

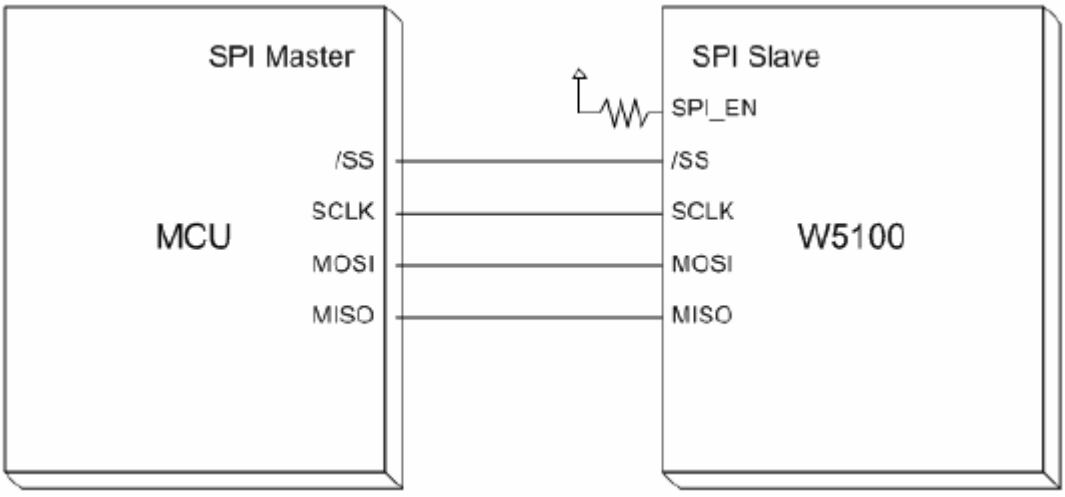
读/写内部寄存器或存储器的过程如下：

1. 先将要读写的地址写入到 IDM_AR0 和 IDM_AR1 寄存器
2. 再从 IDM_DR 寄存器读写数据

如果要对某个地址顺序地读写，则可以将模式寄存器 MR 的 AI 置”1”，然后执行一次上述第 1 项后，在读写 IDM_DR，IDM_AR 的值将自动加 1。这样，只需要连续对 IDM_DR 读写，数据就可以连续地读出或写入。

6.3 SPI 总线接口

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、/SS、MOSI、MISO。W5100 的 SPI_EN 引脚选择 SPI 操作。



6.3.1 设备操作

主控制器（SPI 的主设备）发出一系列指令控制 W5100 的运行。SPI 主设备通过四个信号线与 W5100 通信：从设备选择（/SS）、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 时钟极性和相位，它控制数据在 SPI 总线上传输。

W5100 工作在 SPI 从设备的模式 0，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。

6.3.2 命令

根据 SPI 协议，SPI 设备之间只有 2 条数据线。因此需要定义操作代码（OP-Code）。W5100 使用两种操作代码——读代码和写代码。除了这两种代码，其它的操作码都不响应。

在 SPI 模式，W5100 使用“完整 32 位数据流”。

完整的 32 位数据流包括一个字节的操作码、2 个字节的地址码和 1 个字节的数据。

操作码、地址和数据字节传输都是高位（MSB）在前低位（LSB）在后。换句话说，SPI 数据的第一位是操作码的高位（MSB），最后一位是数据的低位（LSB）。W5100 的 SPI 数据格式如下：

命令	操作码		地址	数据
写操作	0xF0	1111 0000	2 字节	1 字节
读操作	0x0F	0000 1111	2 字节	1 字节

6.3.3 SPI 主设备操作

1．配置 SPI 主设备输入/输出方向

- /SS（从设备选择）：输出
- SCLK（串行时钟）：输出
- MOSI（主输出从输入）：输出
- MISO（主输入从输出）：输入

2．将/SS 置高电平

3．设置 SPI 主设备的寄存器

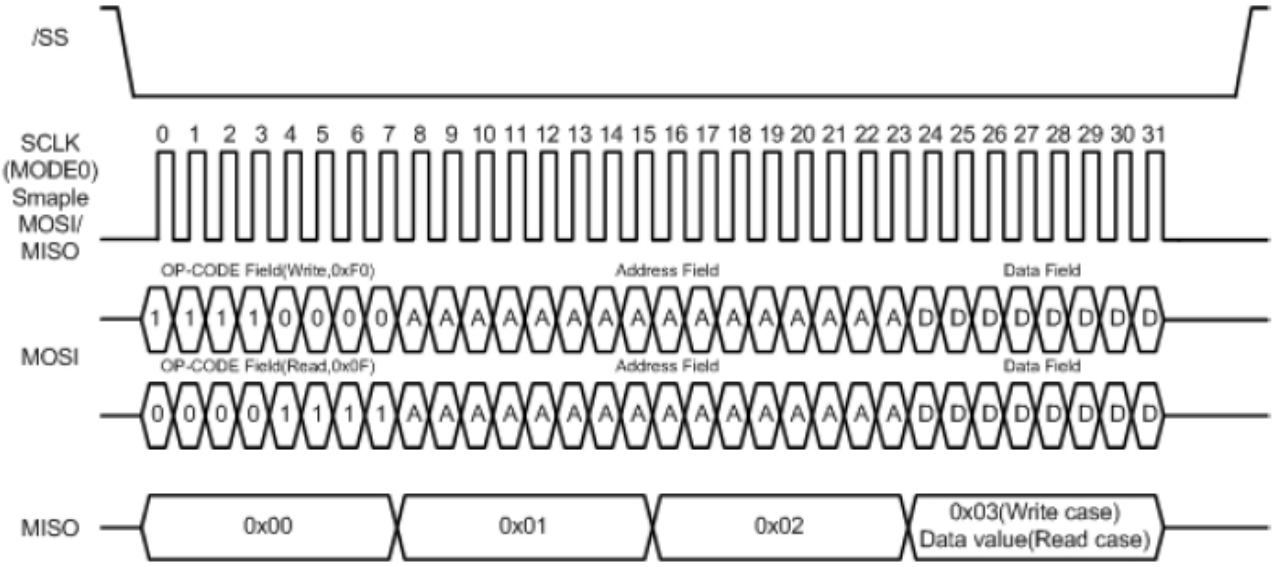
- SPI 允许位在 SPCR 寄存器
- 主/从设备选择位在 SPCR 寄存器
- SPI 模式选择在 SPCR 寄存器
- SPI 数据速率位在 SPCR 寄存器和 SPSR 寄存器（SPI 状态寄存器）

4．向 SPI 数据寄存器（SPDR）写入要传输的数据

5．将/SS 置低电平

6．等待接收完成

7．如果所有数据都传输完成，将/SS 置高电平



7. 电气规格

7.1 极限值

Symbol	Parameter	Rating	Unit
V_{DD}	DC Supply voltage	-0.5 to 3.6	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

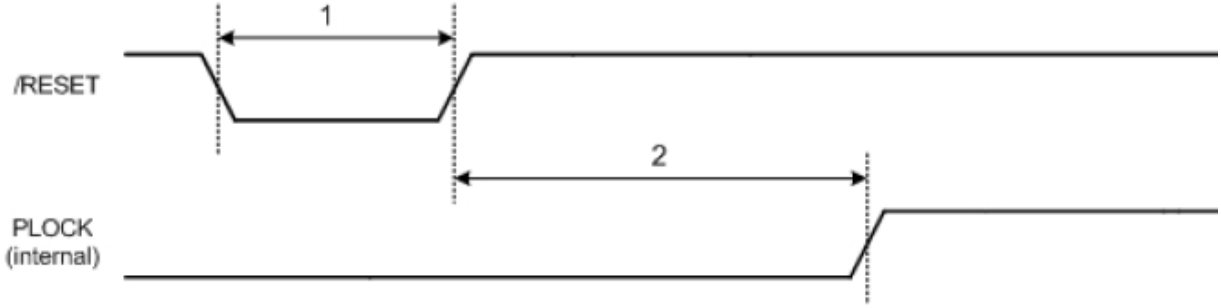
Symbol	Parameter	Test Condition	Min	Typ	Max	Unit
V_{DD}	DC Supply voltage	Junction temperature is from -55°C to 125°C	3.0		3.6	V
V_{IH}	High level input voltage		2.0		5.5	V
V_{IL}	Low level input voltage		- 0.5		0.8	V
V_{OH}	High level output voltage	I_{OH} = 2, 4, 8, 12, 16, 24 mA	2.0		3.6	V
V_{OL}	Low level output voltage	I_{OL} = -2, -4, -8, -12, -16, -24 mA	0.0		0.4	V
I_I	Input Current	$V_{IN} = V_{DD}$			±5	µA

7.3 功耗

Symbol	Parameter	Test Condition	Min	Typ	Max	Unit
P_{10Base}	Power consumption in 10BaseT	Vcc 3.3V Temperature 25°C	-	138	183	mA
$P_{100Base}$	Power consumption in 100BaseT	Vcc 3.3V Temperature 25°C	-	146	183	mA

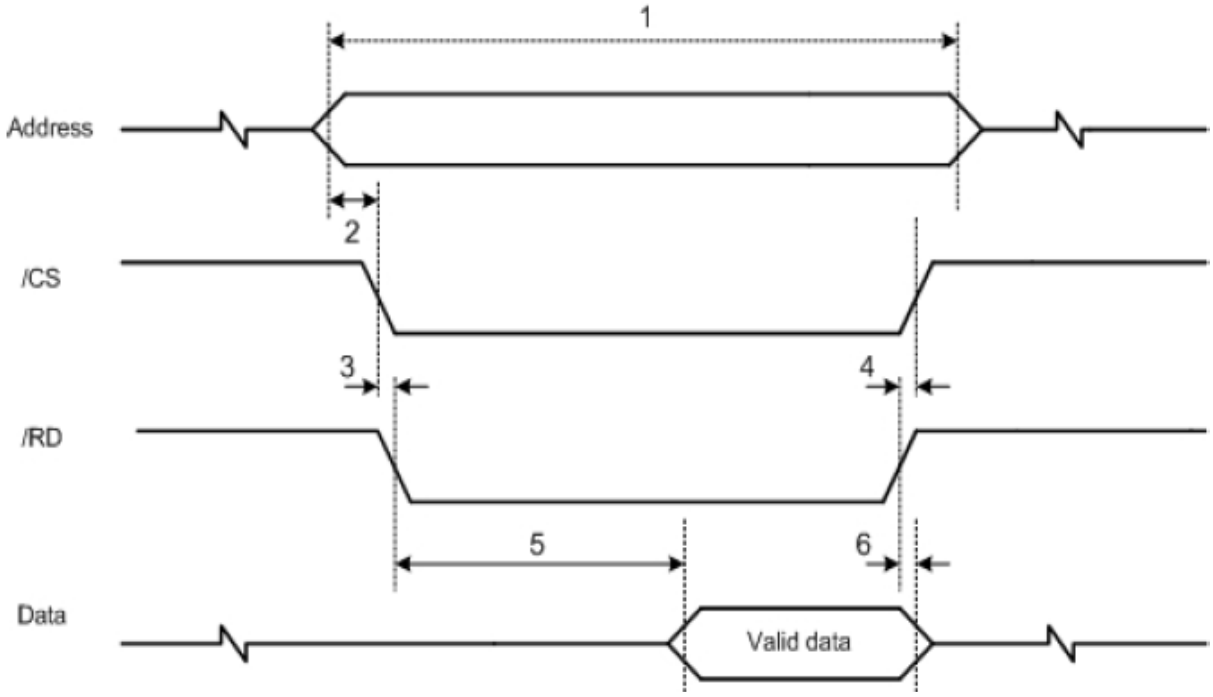
7.4 特征

7.4.1 复位时钟



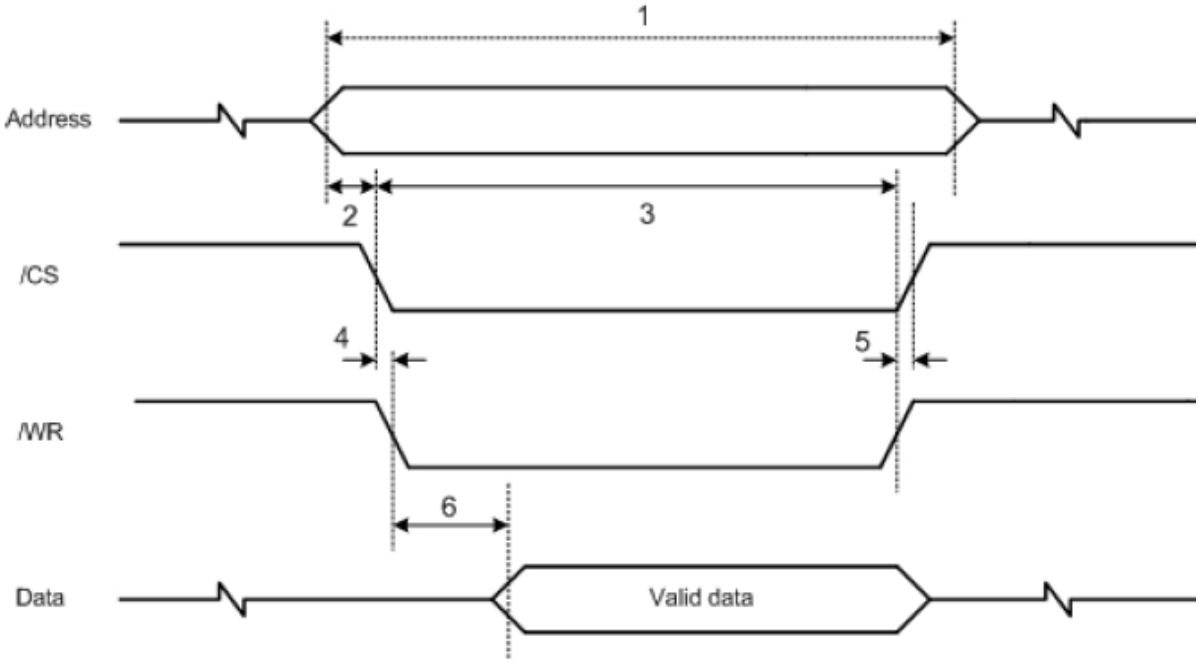
	Description	Min	Max
1	Reset Cycle Time	2 us	-
2	/RESET to internal PLOCK	-	10 ms

7.4.2 寄存器 / 存储器读出时钟图



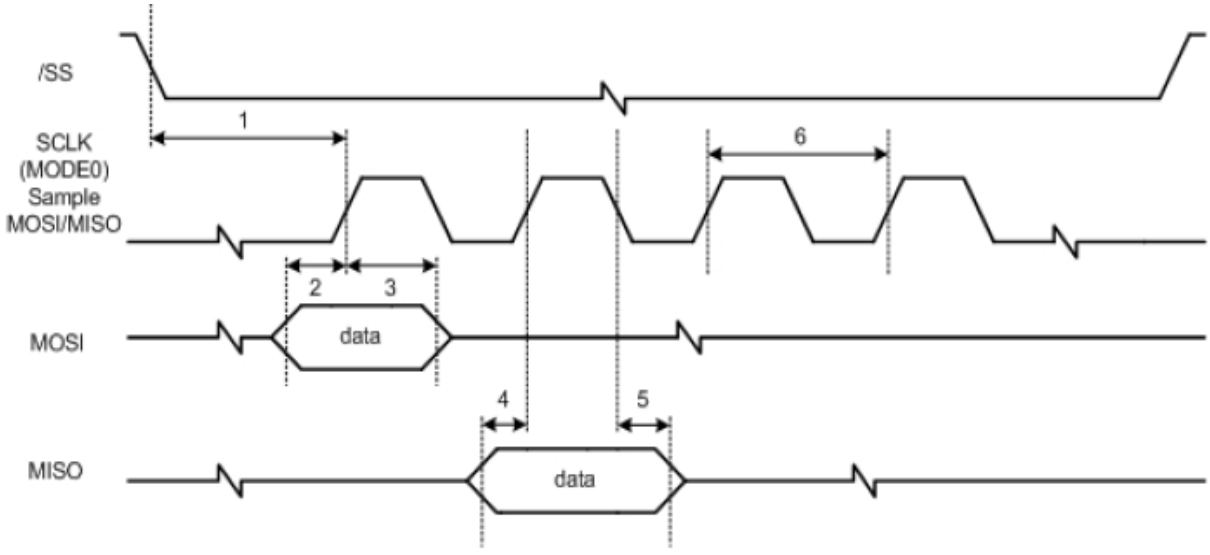
	Description	Min	Max
1	Read Cycle Time	80 ns	-
2	Valid Address to /CS low time	8 ns	-
3	/CS low to /RD low time	-	1 ns
4	/RD high to /CS high time	-	1 ns
5	/RD low to Valid Data Output time	-	80 ns
6	/RD high to Data High-Z Output time	-	1 ns

7.4.3 寄存器/存储写入时钟图



	Description	Min	Max
1	Write Cycle Time	70 ns	-
2	Valid Address to /CS low time	7 ns	-
3	/CS low to /WR high time	70 ns	-
4	/CS low to /WR low time	-	1 ns
5	/WR high to /CS high time	-	1 ns
6	/WR low to Valid Data time	-	14 ns

7.4.4 SPI 时钟图



	Description	Mode	Min	Max
1	/SS low to SCLK high	Slave	21 ns	-
2	Input setup time	Slave	7 ns	-
3	Input hold time	Slave	28 ns	-
4	Output setup time	Slave	7 ns	14 ns
5	Output hold time	Slave	21 ns	-
6	SLKC time	Slave	70 ns	
7	SCLK high to /SS high	Slave	21ns	

7.4.5 晶体特性

Parameter	Range
Frequency	25 MHz
Frequency Tolerance (at 25℃)	±30 ppm
Shunt Capacitance	7pF Max
Drive Level	100uW
Load Capacitance	27pF
Aging (at 25℃)	±3ppm / year Max

7.4.6 变压器特性

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

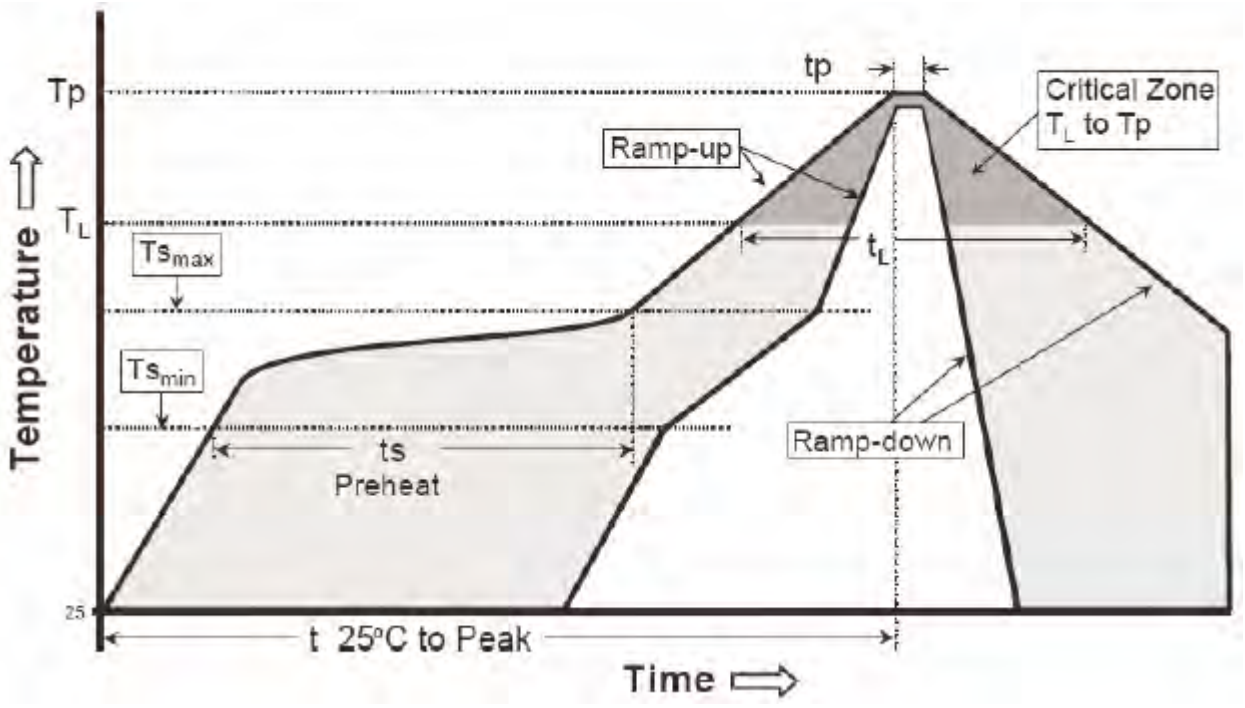
Symmetrical TX & RX channels for auto MDI/MDIX capability

8 回流焊温度表图(无铅封装)

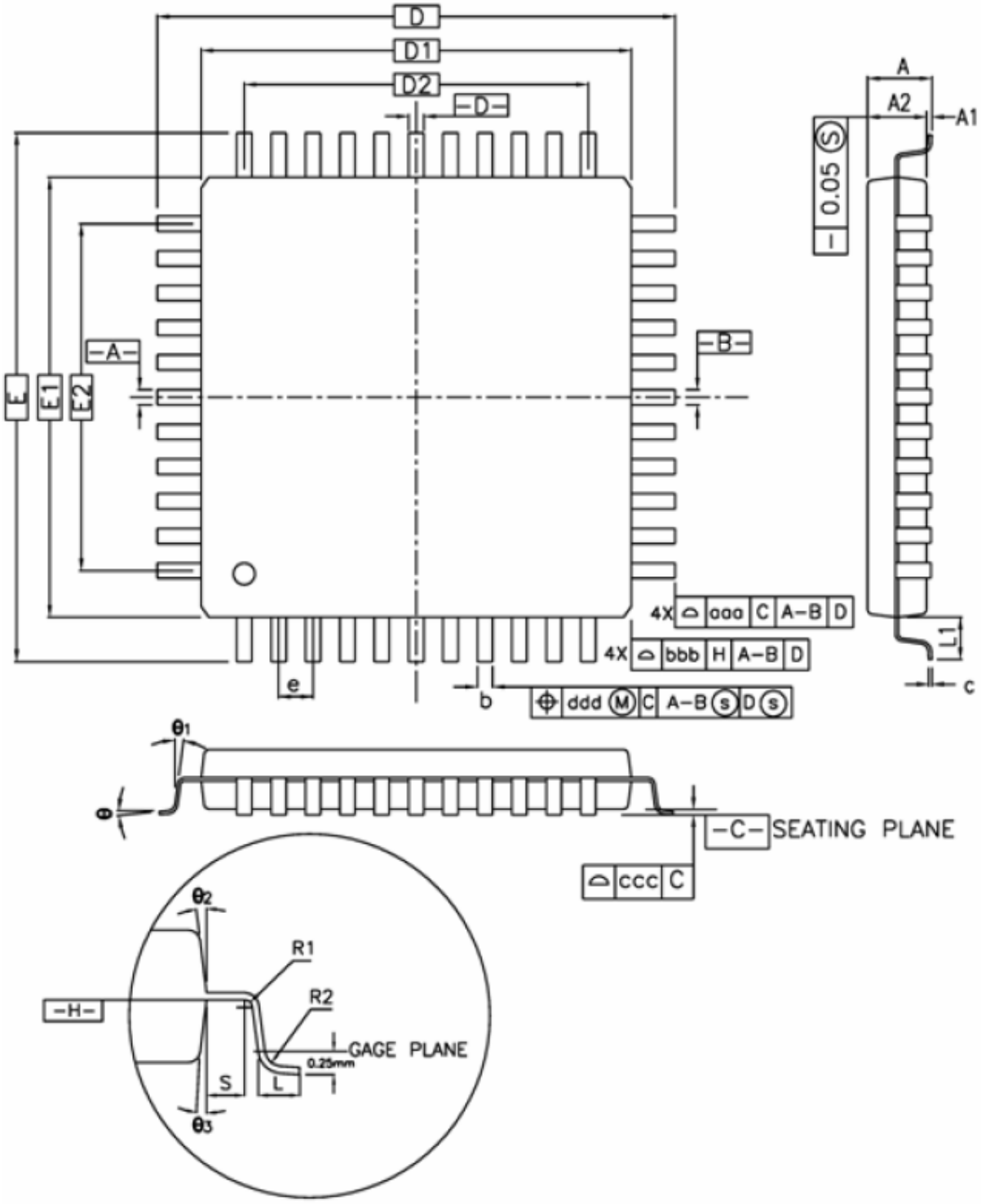
湿度感应度：3 级

需要干燥包装

Average Ramp-Up Rate (Ts_{max} to Tp)	3° C/second max.
Preheat – Temperature Min (Ts_{min}) – Temperature Max (Ts_{max}) – Time (ts_{min} to ts_{max})	 150 °C 200 °C 60-180 seconds
Time maintained above: – Temperature (TL) – Time (tL)	 217 °C 60-150 seconds
Peak/Classification Temperature (Tp)	260 + 0 °C
Time within 5 °C of actual Peak Temperature (tp)	20-40 seconds
Ramp-Down Rate	6 °C/second max.
Time 25 °C to Peak Temperature	8 minutes max.



9. 封装概述


D
D1
D2
-D-
A
A2
A1
0.05 S
-A-
-B-
E
E1
E2
4X aaa C A-B D
4X bbb H A-B D
L1
c
e
b
ddd M C A-B S D S
θ1
θ
-C- SEATING PLANE
ccc C
θ2
R1
R2
-H-
GAGE PLANE
0.25mm
S
L
θ3

SYMBOL	MILLIMETER			INCH		
	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.
A	-	-	1.60	-	-	0.063
A1	0.05	-	0.15	0.002	-	0.006
A2	1.35	1.40	1.45	0.053	0.055	0.057
D		12.00 BSC.			0.472 BSC.	
D1		10.00 BSC.			0.393 BSC.	
E		12.00 BSC.			0.472 BSC.	
E1		10.00 BSC.			0.393 BSC.	
R2	0.08	-	0.20	0.003	-	0.008
R1	0.08	-	-	0.003	-	-
θ	0°	3.5°	7°	0°	3.5°	7°
θ_1	0°	-	-	0°	-	-
θ_2	11°	12°	13°	11°	12°	13°
θ_3	11°	12°	13°	11°	12°	13°
c	0.09	-	0.20	0.004	-	0.008
L	0.45	0.60	0.75	0.018	0.024	0.030
L1		1.00 REF			0.039 REF	
S	0.20	-	-	0.008	-	-
b	0.13	0.16	0.23	0.005	0.006	0.009
e		0.40 BSC			0.016 BSC	
D2		7.60			0.299	
E2		7.60			0.299	
aaa		0.20			0.008	
bbb		0.20			0.008	
ccc		0.08			0.003	
ddd		0.07			0.003	