W3150A+ 数据手册

版本 2.0.4

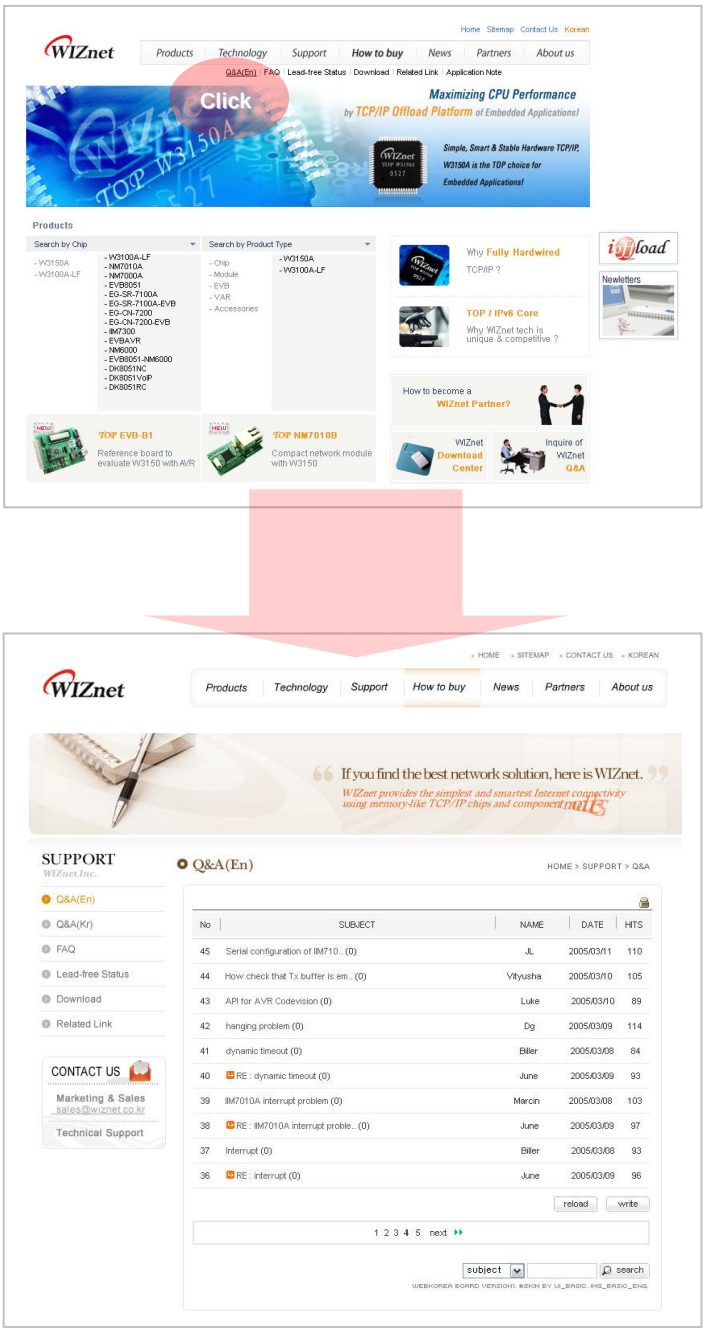
想了解更多，请登录我们的网站http://www.iwiznet.cn/

文档历史信息

版本	日期	说明
Ver. 1.0.0	2005-10-27	官方发布
Ver. 1.0.1	2005-11-21	替换，*1.8V*工作电压→ **3.3V**工作电压 (p.3) 改变框图 (p.4) 改变图 (p.32) 替换，*g_Sn_TX_BASE* → **g_Sn_RX_BASE** (p.33) 替换，memcpy(, ,*left_size*) → in memcpy(, ,**upper_size**) (p.40, p.41, p.47, p.48, p.49) 替换，get_offset = *Sn_TX_RR* & → get_offset = **Sn_TX_WR** & (p.41, p.49) 替换，*SOCK_UDP* → **SOCK_IPRAW** (p.51)
Ver. 1.0.2	2005-12-28	增加7.3 功耗 (p.56)
Ver. 2.0.0	2006-08-15	新版本发布(W3150A -> W3150A+) 增加SPI信息 在socket模式寄存器中增加ND选项 删除Memory测试模式 增加MACRAW模式
Ver. 2.0.1	2007-01-08	没有使用模式寄存器的LB位 W3150A+只用域大端排序
Ver. 2.0.2	2007-04-05	改变操作温度值(p.57)
Ver. 2.0.3	2007-05-02	修改Sn_IR寄存器中RECV_INT的解释(P. 27) 替换Sn_DHAR寄存器的复位值(0x00 to 0xFF, P. 30) 修改Sn_DIPR，Sn_DPORT寄存器的解释(P. 30) 替换Sn_MSS寄存器的复位值(0xFFFF to 0x0000, P. 31) 修改W3150A+ AC特性图(P. 58,59,60,62,63)
Ver. 2.0.4	2007-08-05	修改了W3150A+ AC 特征图 （增加了第7条关于SCLK高电平转/SS高电平的阐述，P.61）

WIZnet在线技术支持

如果您有任何关于我们产品的问题，请登录我们的网站http://www.iwiznet.cn/，在Q&A（在线问答）栏目提交您的问题，我们会尽快给您回复。



W3150A+ 数据手册

说明

W3150A+是一个嵌入了硬件协议栈的大规模集成电路, 通过使用简单的硬件实现TCP/IP协议栈，为数字设备的高速网络连接提供了简单且低成本的解决方案。

W3150A+提供了快速、简单的方法来为任何产品添加以太网网络功能。在一个系统里使用这种大规模集成电路能够完全提供网络连接和处理标准协议，减少了软件开发成本和开发时间，这对于缩短产品上市时间来说尤为重要。

W3150A+包括TCP/IP协议栈，如TCP，UDP，ICMP，IPv4，ARP和PPPoE协议，还有以太网协议如MAC协议。总内存大小是16K字节，用于数据传输和接收的缓冲区。

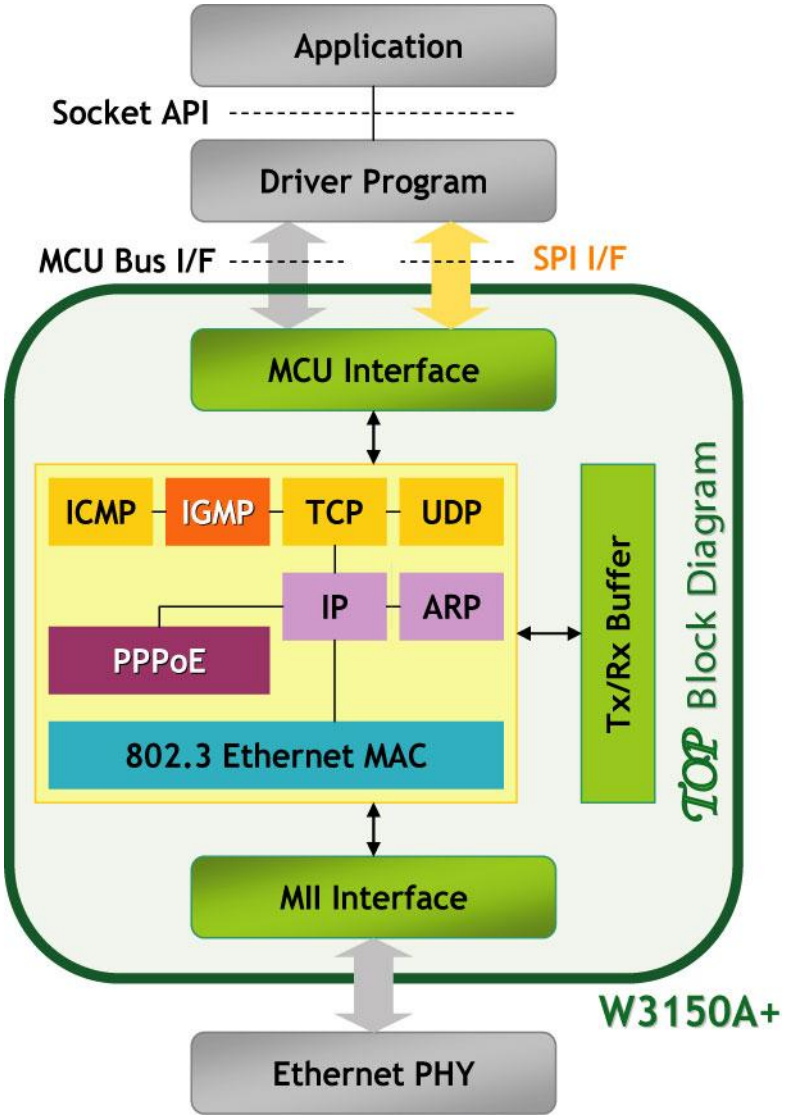
W3150A+提供三种不同的接口，像直接、间接的总线和SPI(Serial Peripheral Interface串行外设接口)用来连接MCU和由半字节数据总线组成的标准MII(Media Independent Interface)连接到以太网PHY设备。

W3150A+是最适合的嵌入式应用设备，包括IP-机顶盒，网络硬盘录像机（Internet-DVR），网络电话，VoIP SOC芯片，网络MP3播放器，手持医疗设备，各种用于监视和测量的工业系统，和一些其他不可移植的电子设备如大型消费电子产品。

特点

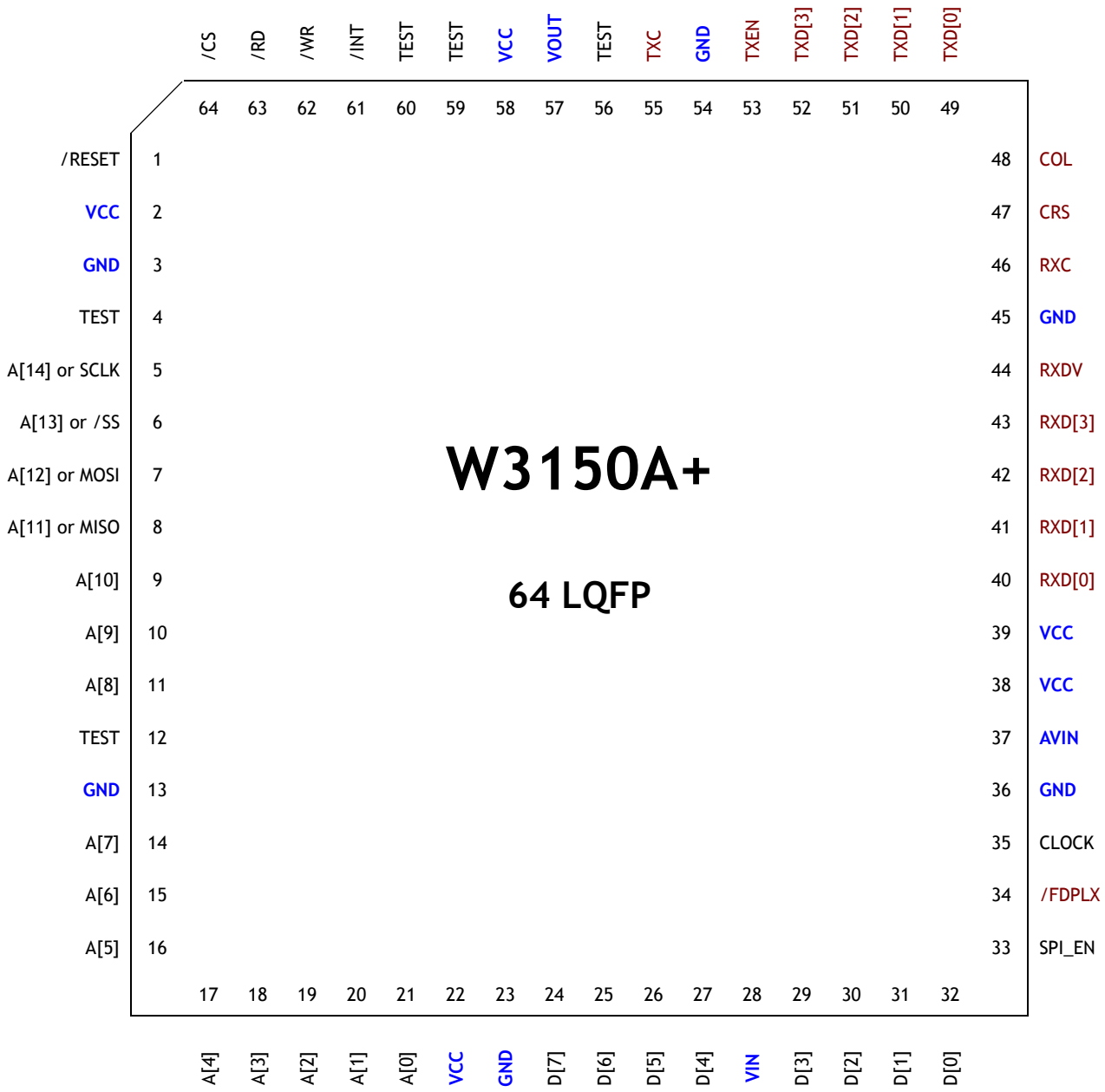
- 支持硬件TCP/IP协议：TCP, UDP, ICMP, IGMP, IPv4, ARP, PPPoE, Ethernet
- 支持ADSL连接(支持PPPoE协议，支持PAP/CHAP认证模式)
- 同时支持4个独立的socket
- 不支持IP分片
- Ethernet-PHY芯片的标准MII接口
- 支持10BaseT/100BaseTX
- 支持全双工模式
- 内置16KB存储器用于Tx/Rx Buffers
- 0.18 μm CMOS技术
- 3.3V工作电压，5V I/O信号公差
- 小的64引脚LQFP封装
- 无铅（Lead-Free）封装
- 支持SPI接口(SPI MODE 0, 3)

框图



目录

1. 引脚图



1.1. MII 信号说明

引脚号	信号	I/O	说明
55	TXC	I	**发送时钟** 该输入引脚需要一个持续时钟作为TXD[3:0]和TXEN的时序参考。 TXC由PHY提供，TXC在10 BASE-T半字节模式是2.5 MHz，在100 BASE-T半字节模式是25MHz。
53	TXEN	O	**发送使能** 该输出信号表明TXD[3:0]存在有效的半字节数据。当数据包的第一个半字节在TXD[3:0]上是有效的，该信号变为有效；当数据包的最后半字节离开TXD[3:0]时，该信号失效。该信号直接连到PHY设备。该信号高电平有效。
52 51 50 49	TXD[3] TXD[2] TXD[1] TXD[0]	O	**发送数据** 当TXEN有效时，这些引脚通过TXC同步的发送Nibble NRZ数据到PHY。
46	RXC	I	**接收时钟** 该输入引脚需要一个持续时钟作为RXD[3:0]和RXDV信号的时序参考。RXC 由PHY提供，RXC在10 BASE-T半字节模式是2.5 MHz，在100 BASE-T半字节模式是25MHz。
48	COL	I	**冲突检测** 该高电平有效信号表明在半双工模式检测到冲突。该信号是异步的，在全双工模式下被忽略。
47	CRS	I	**载波监听** 该高电平有效信号检测载波是否存在。
44	RXDV	I	**接收数据有效** 如果在该引脚上检测到高电平，RXD[3:0]存在有效数据。如果在有效数据包结尾检测到低电平，该信号在RXC的上升沿有效。
43 42 41 40	RXD[3] RXD[2] RXD[1] RXD[0]	I	**接收数据** 当RXDV有效，这些引脚通过RXC同步的发送Nibble NRZ数据到PHY。

1.2. MCU 接口信号说明

引脚号	信号	I/O	说明
1	/RESET	I	**复位** 该引脚是低电平输入信号，用于初始化或重新初始化W3150A+。 将该引脚拉低最少2us将强制执行复位程序，所有内部寄存器将重新初始化到默认状态。
35	CLOCK	I	**时钟** 该引脚是W3150A+操作的主时钟。25MHz。通常，为了节省成本，可以共享PHY的驱动时钟。 注意) 共享晶振有源时钟给多个设备提供时钟可能会引起一些问题。在我们的参考设计中，我们使用一个晶振同时用于PHY和W3150A+，该设计经过了校验。
5	A[14]/ SCLK	I	**ADDRESS引脚或SCLK (Serial Clock)** * 该引脚用来选择一个寄存器或内存。 当SPI_EN引脚为高电平时，该引脚被用做SPI时钟信号引脚。
6	A[13]/ /SS	I	**ADDRESS引脚或/SS (Slave Select)** * 该引脚用来选择一个寄存器或内存。 当SPI_EN引脚为高电平，该引脚被用做SPI从选择信号引脚。只有在SPI模式，该引脚低电平有效。
7	A[12]/ MOSI	I	**ADDRESS引脚或MOSI (Master Out Slave In)** * 该引脚用来选择一个寄存器或内存。 当SPI_EN引脚为高电平，该引脚被用作SPI MOSI信号引脚。
8	A[11]/ MISO	I/O	**ADDRESS引脚或MISO (Master In Slave Out)** * 该引脚用来选择一个寄存器或内存。 当发现SPI_EN引脚为高电平，该引脚被用作SPI MISO信号引脚。
9:11 14:21	A[10:8] A[7:0]	I	**ADDRESS引脚** 该引脚用来选择一个寄存器或内存。
24:27, 29:32	D[7:4] D[3:0]	I/O	**DATA引脚** 该引脚用来读和写一个寄存器或内存数据。

* 和 W3150A不同

61	/INT	O	**中断** 该引脚表明在socket连接、断开、接收数据和超时后W3150A+需要MCU口令。通过写IR(中断寄存器)或Sn_IR(Socket nth中断寄存器)清除该中断。 所有中断都是可屏蔽的。该信号是低电平有效。
64	/CS	I	**片选** 片选用于MCU访问内部寄存器/内存。/WR和/RD选择数据的传输方向。该信号是低电平有效。
62	/WR	I	**写使能** 通过A[14:0]选择从MCU选通脉冲写一个内部寄存器/内存。在该信号的上升沿数据被锁定到W3150A+。该信号是低电平有效。
63	/RD	I	**读使能** 通过A[14:0]选择从MCU选通脉冲读一个内部寄存器/内存。该信号是低电平有效。

1.3. 其他信号说明

引脚号	信号	I/O	说明
34	/FDPLX	I	**全/半双工选择** 该引脚选择全/半双工操作模式。 为了配置W3150A+为全双工操作，该引脚必须被外部拉低(典型值 x kΩ)。 Low = 全双工 High = 半双工
33	SPI_EN	I	**SPI使能*** 该引脚选择SPI模式的启用/禁用。 以前的W3150A用户使用该引脚时，该引脚被内部下拉。即使没有信号连接到该引脚，它还是内部拉低的。因此，如果要改变到W3150A+，没有必要改变之前的板子设计。 Low = SPI Mode启用 High = SPI Mode禁止
4,12,56, 59,60	TEST	I	**出厂测试输入** 用来检查芯片的内部功能。该引脚在正常操作是应该被拉低。

* * 和W3150A不同

1.4. 电源信号说明

引脚号	信号	I/O	说明
2, 22, 38, 39, 58	VCC		**正极3.3V引脚**
28	VIN		**1.8V电源输入** 提供1.8V电源
37	AVIN		**1.8V模拟电源输入** 提供1.8V模拟电路电源；应该去耦。 参考图 1-1，电源输入参考原理图。
57	VOUT		**1.8V电源输出** 确定连接一个10uF钽电容和一个0.1uF电容以去噪。然后通过一个磁珠将该引脚连接到VIN和AVIN。
3, 13, 23, 36, 45, 54	GND		**负极(地)引脚**

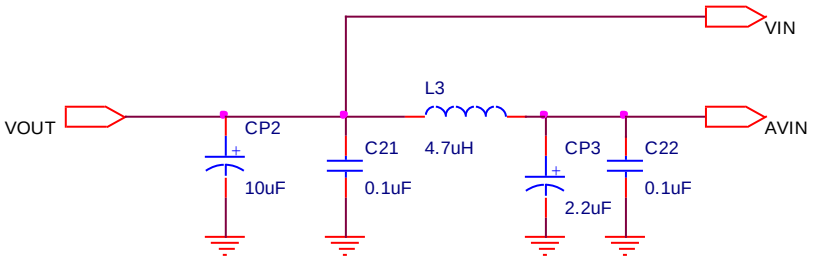
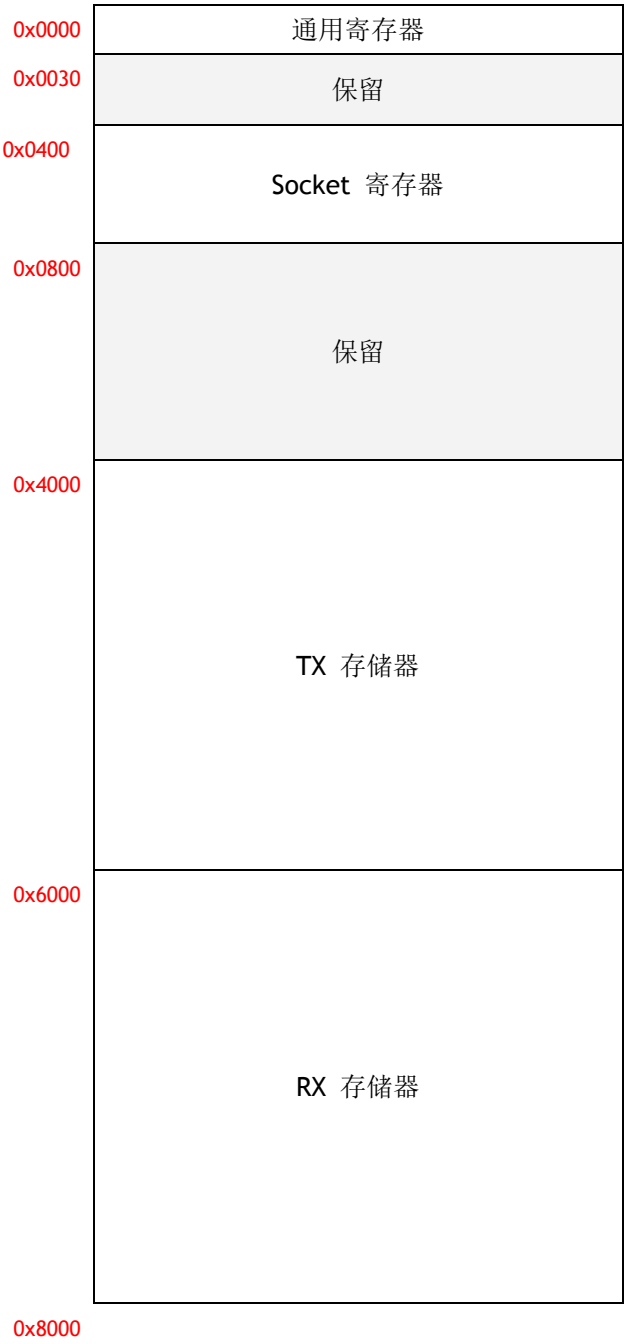


图 1-1. 电源输入参考原理图

2. 内存映射

W3150A+由通用寄存器，Socket寄存器，TX存储器和RX存储器组成。每个字段如下所示。



3. W3150A+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)
0x0004	(GAR3)
0x0005	子网掩码 (SUBR0)
0x0006	(SUBR1)
0x0007	(SUBR2)
0x0008	(SUBR3)
0x0009	源硬件地址 (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	中断屏蔽 (IMR)
0x0017	重试时间 (RTR0)
0x0018	(RTR1)
0x0019	重试次数(RCR)

地址	寄存器
0x001A	RX 存储大小 (RMSR)
0x001B	TX存储大小(TMSR)
0x001C	Authentication Type in PPPoE (PATR0)
0x001D	(PATR1)
0x001E ~ 0x□027	保留
0x0028	PPP LCP 请求定时器 (PTIMER)
0x0029	PPP LCP 魔术数r (PMAGIC)
0x002A	不可到达的IP 地址 (UIPR0)
0x002B	(UIPR1)
0x002C	(UIPR2)
0x002D	(UIPR3)
0x002E	不可到达端口 (UPORT0)
0x002F	(UPORT1)
0x0030 ~ 0x03FF	保留

3.2. Socket 寄存器

地址	寄存器
0x0400	Socket 0 模式 (S0_MR)
0x0▫01	Socket 0 命令 (S0_CR)
0x0402	Socket 0中断 (S0_IR)
0x0403	Socket 0 状态 (S0_SR)
0x0404 0x0405	Socket 0 源端口 (S0_PORT0) (S0_PORT1)
0x0406 0x0407 0x0408 0x0409 0x040A 0x040B	Socket 0 目的硬件地址 (S0_DHAR0) (S0_DHAR1) (S0_DHAR2) (S0_DHAR3) (S0_DHAR4) (S0_DHAR▫)
0x040C 0x040D 0x040E 0x040F	Socket 0 目的 IP 地址 (S0_DIPR0) (S0_DIPR1) (S0_DIPR2) ▫S0_DIPR3)
0x0410 0x0411	Socket 0 目的端口 (S0_DPORT0) (S0_DPORT1)
0x0412 0x0413	Socket 0 最大分段大小 (S0_MSSR0) (S0_MSSR1)
0x0414	IP Raw 模式时Socket 0 协议 (S0_PROTO)

地址	寄存器
0x0415	Socket 0 IP TOS (S0_TOS)
0x0416	Socket 0 IP TTL (S0_TTL)
0x0417 ~ 0x041F	Res▫rved
0x0420 0x0421	Socket 0 TX 剩余字节大小 (S0_TX_FSR0) (S0_TX_FSR1)
0x0422 0x0423	Socket 0 TX 读指针 (S0_TX_RD0) (S0_TX_RD1)
0x0424 0x0425	Socket 0 TX 写指针 (S0_TX_WR0) (S0_TX_▫R1)
0x0426 0x0427	Socket 0 RX 接收大小 (S0_RX_RSR0) (S0_RX_RSR1)
0x0428 0x0429	Socket 0 RX 读指针 (S0_RX_RD0) (S0_RX_RD1)
0x042A 0x042B	保留
0x042C ~ 0x04FF	保留

地址	寄存器
0x0500	Socket 1 模式 (S1_MR)
0x0501	Socket 1 命令 (S1_CR)
0x0502	Socket 1 中断 (S1_IR□
0x0503	Socket 1 状态(S1_SR)
0x0504 0x0505	Socket 1 源端口 (S1_PORT0) (S1_PORT1)
0x0506 0x0507 0x0508 0x0509 0x050A 0x050B	Socket 1 目的硬件地址 (S1_DHAR0) (S1_DHAR1) (S1_DHAR2) (S1_DHAR3) (S1_DHAR4) (S1_DHAR5)
0x050C 0x050D 0x050E 0x050F	Socket 1目的 IP 地址 (S1_DIPR0) (S1_DIPR1) (S1_DIPR2) (S1_DIPR3)
0x0510 0x0511	Socket 1目的端口 (S1_DPORT0) (S1_DPORT1)
0x0512 0x0513	Socket 1 最大分段大小 (S1_MSSR0) (S1_MSSR1)
0x0514	IP Raw 模式时的Socket 1 协议 (S1_PROTO)

地址	寄存器
0x0515	Socket 1 IP TOS (S1_TOS)
0x0516	Socket 1 IP TT□ (S1_TTL)
0x0517 ~ 0x051F	保留
0x0520 0x0521	Socket 1 TX 剩余字节大小 (S1_TX_FSR0) (S1□TX_FSR1)
0x0522 0x0523	Socket 1 TX读指针 (S1_TX_RD0) (S1_TX_RD1)
0x0524 0x0525	Socket 1 TX 写指针 (S1_TX_WR0) (S1_TX_WR1)
0x0526 0x0527	Socket 1 RX 接收大小 (S1_RX_RSR0) (S1_RX_RSR1)
0x0528 0x0529	Socket 1 RX 读指针 (S1_RX_RD0) (S1_RX_RD1)
0x052A 0x052B	保留
0x052C ~ 0x05FF	保留

地址	寄存器
0x0600	Socket 2 模式 (S2_MR)
0x0601	Socket 2 命令 (S2_CR)
0x0602	Socket 2 中断 (S2_IR)
0x0603	Socket 2 状态(S2_SR)
0x0604 0x0605	Socket 2 源端口 (S2_PORT0) (S2_PORT1)
0x0606 0x0607 0x0608 0x0609 0x060A 0x060B	Socket 2 目的硬件地址 (S2_DHAR0) (S2_DHAR1) (S2_DHAR2) (S2_DHAR3) (S2_DHAR4) (S2_DHAR5)
0x060C 0x060D 0x060E 0x060F	Socket 2 目的 IP 地址 (S2_DIPR0) (S2_DIPR1) (S2_DIPR2) (S2_DIPR3)
0x0610 0x0611	Socket 2 目的端口 (S2_DPORT0) (S2_DPORT1)
0x0612 0x0613	Socket 2 最大分段大小 (S2_MSSR0) (□2_MSSR1)
0x0614	IP Raw 模式时Socket 2 协议 (S2_PROTO)

地址	寄存器
0x0615	Socket 2 IP TOS (S2_TOS)
0x0616	Socket 2 IP TTL (S2_TTL)
0x0617 ~ 0x061F	保留
0x0620 0x0621	Socket 2 TX 剩余字节大小 (S2_TX_FSR0) (S2_TX_FSR1)
0x0622 0x0623	Socket 2 TX 读指针 (S2_TX_RD0) (S2_TX_RD1)
0x0624 0x0625	Socket 2 TX 写指针 (S2_TX_WR0) (S2_TX_WR1)
0x0626 0x0627	Socket 2 RX 接收大小 (S2_RX_RSR0) (S2_RX_RSR1)
0x0628 0x0629	Socket 2 RX 读指针 (S2_RX_RD0) (S2_RX_RD1)
0x062A 0x062B	保留
0x062C ~ 0x06FF	保留

地址	寄存器
0x0700	Socket 3 模式 (S3_MR¤
0x0701	Socket 3 命令(S3_CR)
0x0702	Socket 3 中断 (S3_IR)
0x0703	Socket 3 状态(S3_SR)
0x0704 0x0705	Socket 3 源端口 (S3_PORT0) (S3_PORT1)
0x0706 0x0707 0x0708 0x0709 0x070A ¤x070B	Socket 3 目的硬件地址 (S3_DHAR0) (S3_DHAR1) (S3_DHAR2) (S3_DHAR3) (S3_DHAR4) (S3_¤HAR5)
0x070C 0x070D 0x070E 0x070F	Socket 3目的IP 地址 (S3_DIPR0) (S3_DIPR1) (S3_DIPR2) (S3_DIPR3)
0x0710 0x0711	Socket 3 目的端口 (S3_DPORT0) (S3_DPORT1)
0x0712 0x0713	Socket 3 最大分段寄存器 (S3_MSSR0) (S3_MSSR1)
0x0714	IP Raw 模式时Socket 3 协议 (S3_PROTO)

地址	寄存器
0x0715	Socket 3 IP TOS (S3_TOS)
0x0716	Socket 3 IP TTL (S3_TTL)
0x0717 ~ 0x071F	保留
0x0720 0x0721	Socket 3 TX 剩余字节大小 (S3_TX_FSR0) (S3_TX_FSR1)
0x0722 0x0723	Socket 3 TX 读指针 (S3_TX_RD0) (S3_TX_RD1)
0x0724 0x0725	Socket 3 TX 写指针 (S3_TX_WR0) (S3_TX_W¤1)
0x0726 0x0727	Socket 3 RX 接收大小 (S3_RX_RSR0) (S3_RX_RSR1)
0x0728 0x0729	Socket 3 RX 读指针 (S3_RX_RD0) (S3_RX_RD1)
0x072A 0x072B	保留
0x072C ~ 0x07FF	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][1]

该寄存器用来S/W复位、内存检测模式、ping block模式、PPPoE模式和间接总线I/F。

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

位	标志	说明
7	RST	**S/W重置** 如果该位是‘1’，内部寄存器将被初始化。在复位后该位会自动清除。
6	保留	保留
5	保留	保留
4	PB	**Ping Block模式** 0：禁止Ping block 1：启用Ping block 如果该位是‘1’，ping结果没有响应。
3	PPPoE	**PPPoE模式** 0：禁止PPPoE模式 1：启用PPPoE模式 如果你使用ADSL，而且没有使用路由或其他网络设备，你应该将这位设置为‘1’，连接到ADSL服务器。获得更多信息，请参考应用笔记“如何连接ADSL”。
2	不用	没有使用
1	AI	**在间接总线I/F模式下地址自动增长** 0：禁止自动增长 1：启用自动增长 在间接总线I/F模式，如果这位设置为‘1’，无论读还是写操作地址都会自动的增长。获得更多信息，请参考6.1.2 间接总线 IF 模式。
0	IND	**直接总线I/F模式** 0：禁止间接总线I/F模式 1：启用间接总线I/F模式 如果这位设置为‘1’，设置为间接总线I/F模式。获得更多信息，请参考6.1.2 间接总线IF 模式。

[1] [读/写] [地址] [复位值]

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

该寄存器设置默认网关地址。

例) 如果是“192.168.0.1”

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

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

该寄存器设置子网掩码地址。

Ex) 如果是“255.255.255.0”

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

SHAR (源硬件地址寄存器) [R/W] [0x0009 – 0x000E] [0x00]

该寄存器设置源硬件地址。

Ex) 如果是“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]

该寄存器设置源IP地址。

Ex) 如果是“192.168.0.3”

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

IR (中断寄存器) [R] [0x0015] [0x00]

该寄存器通过主处理器访问以知道有中断发生。

任何中断都可以通过设置中断屏蔽寄存器(IMR)被屏蔽。只要任何屏蔽信号被设置，/INT信号保持低电平，直到该寄存器的所有屏蔽位被清除。

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

位	标志	说明
7	CONFLICT	**IP冲突** 当同样的源IP地址发出ARP请求，该位被设置为‘1’。往该位写‘1’可将该位清零。
6	UNREACH	**目的地址不可到达** 在UDP数据传输阶段，如果不是现有的IP地址在传输，W3150A+将接收ICMP(目的地址不可达)数据包(参考5.2.2. UDP)。如果是这种情况，IP地址和端口号将被保存在不可达IP地址寄存器(UIPR)和不可达端口寄存器(UPORT)，而且该位将被设置为‘1’。往该位写‘1’可将该位清零。
5	PPPoE	**PPPoE关闭** 在PPPoE模式，如果PPPoE连接被关闭，该位被设置为‘1’。往该位写‘1’可将该位清零。
4	Reserved	保留
3	S3_INT	**Socket3发生Socket中断** 该位将被置位如果socket3发生socket中断。获得更多socket中断的信息，请参考“Socket3中断寄存器(S3_IR)”。当S3_IR被清除为0x00，该位将自动清除。
2	S2_INT	**Socket2发生Socket中断** 该位将被置位如果socket2发生socket中断。获得更多socket中断的信息，请参考“Socket2中断寄存器(S2_IR)”。当S2_IR被清除为0x00，该位将自动清除。
1	S1_INT	**Socket1发生Socket中断** 该位将被置位如果socket1发生socket中断。获得更多socket中断的信息，请参考“Socket1中断寄存器(S1_IR)”。当S1_IR被清除为0x00，该位将自动清除。
0	S0_INT	**Socket0发生Socket中断** 该位将被置位如果socket0发生socket中断。获得更多socket中断的信息，请参考“Socket0中断寄存器(S0_IR)”。当S1_IR被清除为0x00，该位将自动清除。

IMR (中断屏蔽寄存器) [R/W] [0x0016] [0x00]

该中断屏蔽寄存器是用来屏蔽中断。每个中断屏蔽位对应中断寄存器(IR)中的一位。如果一个中断屏蔽位被置位，不管中断寄存器（IR）中的相应位什么时候置位，将执行中断。如果中断屏蔽寄存器中的任何位被设置为‘0’，中断将不会发生，即使IR中的相应位被设置。I

7	6	5	4	3	2	1	0
IM_IR7	IM_IR6	IM_IR5	保留	IM_IR3	IM_IR2	IM_IR1	IM_IR0

位	标志	说明
7	IM_IR7	**IP冲突使能**
6	IM_IR6	**目的地址不可用使能**
5	IM_IR5	**PPPoE关闭使能**
4	保留	**应被设置为‘0’**
3	IM_IR3	**Socket3发生Socket中断使能**
2	IM_IR2	**Socket2发生Socket中断使能**
1	IM_IR1	**Socket1发生Socket中断使能**
0	IM_IR0	**Socket0发生Socket中断使能**

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

该寄存器设置超时时间。1代表100us。初始化值是2000(0x07D0)，那意味着200ms。

例)设置为400ms，设为4000(0x0FA0)

0x0017	0x0018
0x0F	0xA0

如果来自远端的CONNECT，DISCON，CLOSE，SEND，SEND_MAC和SEND_KEEP命令没有响应或其他响应延时将发生重传。

RCR (重试计数寄存器) [R/W] [0x0019] [0x08]

该计数器设置重传的次数。如果重传发生的次数大于RCR设置的值，将发生超时中断（Socket n中断寄存器(Sn_IR)的超时位将被设置为‘1’）。

RMSR(RX内存大小寄存器) [R/W] [0x001A] [0x55]

该寄存器总共分配8K的RX内存给每个socket。

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

S1，S2的配置决定内存大小如下。

S1	S0	Memory size
0	0	1KB
0	1	2KB
1	0	4KB
1	1	8KB

按照S1和S0的值，内存分配给socket一共8KB的范围。初始值是0x55，给每个socket分别分配2KB的内存。

例) 当设置为0xAA，每个socket应该分配4K的内存。

但是，总共内存才8KB。内存就常规的分配给socket 0和1，不分配给socket 2和3。因此，socket 2和3没有被完全的使用。

Socket 3	Socket 2	Socket 1	Socket 0
0KB	0KB	4KB	4KB

TMSR(TX内存大小寄存器) [R/W] [0x001B] [0x55]

该寄存器总共分配8K的TX内存给每个socket。通过设置RX内存大小寄存器(RMSR)也可以完成相同的配置。初始值是0x55，给每个socket分别分配2K的内存。

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

该寄存器通知已经同意连接到PPPoE服务器的认证方法。W3150A+支持两种认证类型-PAP和CHAP。

Value	Authentication Type
0xC023	PAP
0xC223	CHAP

PTIMER (PPP连接控制协议请求时间寄存器) [R/W] [0x0028] [0x28]

该寄存器表明持续发送LCP Echo请求。1表示25ms.

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]

该寄存器被用来在LCP协商中的魔术数选项。参考应用笔记"如何连接ADSL"。

UIPR (不可达IP地址寄存器) [R] [0x002A - 0x002D] [0x00]

如果是用UDP数据传输(参考5.2.2. UDP)，如果传输到一个不存在的IP地址，ICMP(目的地址不可达)数据包将被接收。如果是这种情况，那个IP地址和端口号将被分别存放在不可达IP地址寄存器(UIPR)和不可达端口寄存器(UPORT)。

例)如果是 "192.168.0.11",

0x002A	0x002B	0x002C	0x002D
192 (0xC0)	168 (0xA8)	0 (0x00)	11 (0x0B)

UPORT (不可达端口寄存器) [R] [0x002E - 0x002F] [0x0000]

参考不可达IP地址寄存器(UIPR)

例)如果是5000(0x1388),

0x002E	0x002F
0x13	0x88

4.2. Socket 寄存器

Sn[1]_MR (Socket n模式寄存器) [R/W] [0x0400, 0x0500, 0x0600, 0x0700] [0x00][2]

该寄存器设置socket选项或每个socket的协议类型。

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

位	标志	说明
7	MULTI	**多播** 0 : 禁止多播 1 : 启用多播 只在UDP的情况下使用。 为了使用多播，在OPEN命令之前写多播组地址到Socket *n* 目的IP和多播组端口到Socket *n*目的端口寄存器。
6	Reserved	保留
5	ND/MC	**使用No Delayed ACK** 0 : 禁止No Delayed ACK选项 1 : 启用No Delayed ACK选项 只在TCP的情况下使用。如果这位被设置为‘1’，当从一端接收到数据包，ACK包被发送。如果这位被设置为‘0’，ACK包被发送按照内部超时机制。 **多播** 0 : 使用IGMP版本2 1 : 使用IGMP版本1 只在MULTI位为‘1’的情况。
4	Reserved	保留

[1] *n* 是socket号 (0, 1, 2, 3).
[2] [读/写] [socket 0地址, socket 1地址, socket 2地址, socket 3地址] [复位值]

3	P3	**协议** 设置相应socket 为TCP, UDP或 IP RAW 模式。
2	P2	
1	P1	
0	P0	*** socket 0时，存在MACRAW和PPPoE 模式。**

P3	P2	P1	P0	意思
0	0	0	0	关闭
0	0	0	1	TCP
0	0	1	0	UDP
0	0	1	1	IPRAW

P3	P2	P1	P0	意思
0	1	0	0	MACRAW
0	1	0	1	PPPoE

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

该寄存器用于socket *n*的初始化、关闭、建立连接、终止、数据传输和接收命令。执行命令后，该寄存器的值将被自动清除至0x00。

值	标志	说明
0x01	OPEN	用来初始化socket。按照Socket *n*模式寄存器(Sn_MR)的值，Socket *n*状态寄存器(Sn_SR)值被改变为SOCK_INIT，SOCK_UDP，SOCK_IPRAW或SOCK_MACRAW。 获得很多信息，请参考5. 功能说明。
0x02	LISTEN	只在TCP模式下使用。 改变Socket *n*状态寄存器(Sn_SR)的值到SOCK_LISTEN，为了等待一个来自远端(TCP客户端)的连接请求。 获得更多信息，请参考5.2.1.1. 服务器。
0x04	CONNECT	只在TCP模式下使用。 发送一个连接请求到远端(TCP服务器)。如果连接失败了，将发生超时中断。 获得更多信息，请参考5.2.1.2. 客户端。
0x08	DISCON	只在TCP模式下使用。 发送一个连接终止请求。如果连接终止失败了，将发生超时中断。 获得更多信息，请参考5.2.1.1. 服务器。 ** 如果使用CLOSE命令代替DISCON，只有在Socket n状态寄存器(Sn_SR)的值被改变为SOCK_CLOSED并且没有连接终止进程。*
0x10	CLOSE	用来关闭socket。改变Socket n状态寄存器(Sn_SR)的值为SOCK_CLOSED。
0x20	SEND	按Socket *n* TX写指针增长大小发送数据。获得更多信息，参考Socket *n* TX空闲大小寄存器 (Sn_TX_FSR)， Socket *n* TX写指针寄存器(Sn_TX_WR)，和Socket *n* TX读指针寄存器(Sn_TX_RR)或5.2.1.1. 服务器。

0x21	SEND_MAC	只适用于UDP模式。 基本操作和SEND一样。一般的SEND操作需要能够被ARP(地址解析协议)进程接收目的硬件地址。SEND_MAC使用Socket *n*目的硬件地址(S*n*_DHAR)，该地址有用户写入并且不使用ARP进程。
0x22	SEND_KEEP	啊值适用于TCP模式。 通过发送1byte数据检查连接状态。如果连接已经终止或一端没有响应，将发生超时中断。
0x40	RECV	接收是与Socket *n* RX读指针寄存器(S*n*_RX_RD)处理。 获得更多的信息，请参考5.2.1.1. 服务器接收进程与Socket *n* RX接收大小寄存器(S*n*_RX_RSR)，Socket *n* RX写指针寄存器(S*n*_RX_WR)，和Socket *n* RX读指针寄存器(S*n*_RX_RD)。

S*n*_IR (Socket *n*中断寄存器) [R] [0x0402, 0x0502, 0x0602, 0x0702] [0x00]

该寄存器用来通知连接建立和终止，接收数据和超时。Socket *n* 中断寄存器通过写‘1’清空。

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

位	标志	说明
7	Reserved	保留
6	Reserved	保留
5	Reserved	保留
4	SEND_OK	如果发送操作完成，这位将被设为‘1’。
3	TIMEOUT	如果在连接建立或终止和数据发送期间发生超时，该位被设为‘1’。
2	RECV	每当w3150a+接收数据时，该位被设为‘1’。 如果在执行CMD_RECV后接收数据保留，该位也被设为‘1’。
1	DISCON	如果连接终止被请求或终止，该位被设为‘1’。
0	CON	如果连接被建立，该位被设为‘1’。

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

该寄存器保存socket *n*的状态值。主要的状态如下图所示。

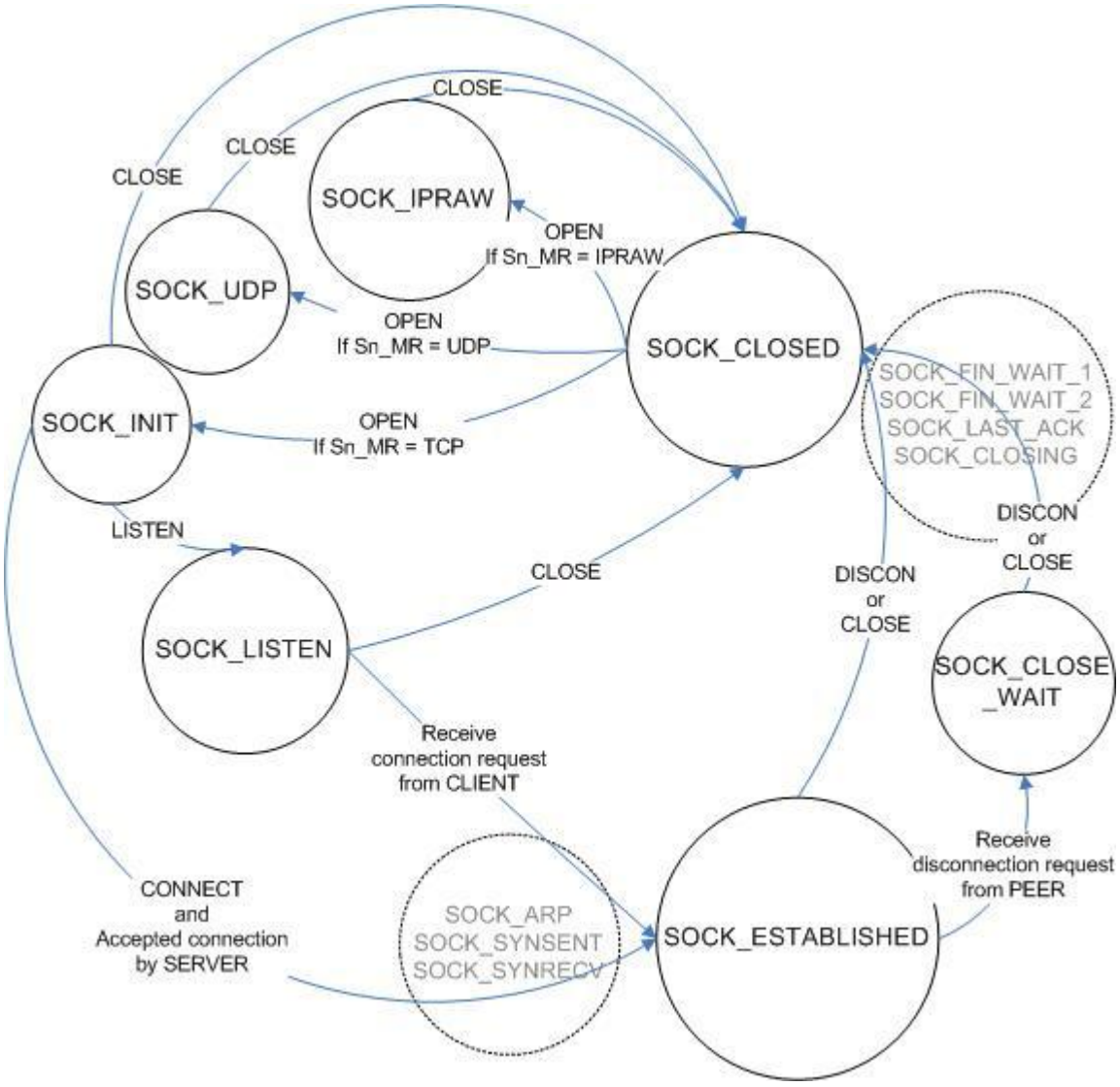


图 4-1. 状态图

值	标志	说明
0x00	SOCK_CLOSED	用于以下情况：CLOSE命令发送到S*n*_CR，并且超时中断发生或连接终止。
0x13	SOCK_INIT	用于以下情况：S*n*_MR被设置为TCP，并且OPEN命令发送给S*n*_CR。
0x14	SOCK_LISTEN	用于以下情况：在SOCK_INIT状态时LISTEN命令发送给S*n*_CR
0x17	SOCK_ESTABLISHED	用于以下情况：连接建立
0x1C	SOCK_CLOSE_WAIT	用于以下情况：来自主机端的连接终止请求被接收到
0x22	SOCK_UDP	用于以下情况：当S*n*_MR被设置为UDP时OPEN命令被发送给S*n*_CR。
0x32	SOCK_IPRAW	用于以下情况：S*n*_MR被设置为IPRAW时OPEN命令被发送给S*n*_CR。
0x42	SOCK_MACRAW	用于以下情况：S0_MR被设置为MACRAW时OPEN命令被发送给S0_CR。
0x5F	SOCK_PPPOE	用于以下情况：S0_MR被设置为PPPoE时OPEN命令被发送给S0_CR。

改变状态期间如下所示。

值	符号	描述
0x15	SOCK_SYNSENT	表明在SOCK_INIT状态如果CONNECT命令被发送给Socket *n*命令寄存器(S*n*_CR)。当连接建立，它将自动的改变到SOCK_ESTABLISH状态。
0x16	SOCK_SYNRECV	表明如果来自远端(客户端)连接请求被接收。它通常会响应该请求并改变状态为SOCK_ESTABLISH。
0x18	SOCK_FIN_WAIT	表明连接终止的过程。如果连接终止被正常的处理或超时中断发生，它将自动的改变状态为SOCK_CLOSED。
0x1A	SOCK_CLOSING	
0X1B	SOCK_TIME_WAIT	
0X1D	SOCK_LAST_ACK	
0x11 0x21 0x31	SOCK_ARP	表明当TCP模式发送连接请求或UDP模式发送数据，为了获取远端的硬件地址ARP请求被发送。如果ARP回答被收到，将改变状态为SOCK_SYNSENT，SOCK_UDP或SOCK_ICMP，为了下次操作。

Sn_PORT (Socket n源端口寄存器) [R/W] [0x0404-0x0405, 0x0504-0x0505, 0x0604-0x0605, 0x0704-0x0705] [0x00]

当使用TCP或UDP模式时，该寄存器设置每个Socket的源端口号，且在执行OPEN命令之前该寄存器进行一些必要的设置。

例)如果Socket 0 Port = 5000(0x1388)，如下配置，

0x0404	0x0405
0x13	0x88

Sn_DHAR (Socket n目的硬件地址寄存器) [R/W] [0x0406-0x040B, 0x0506-0x050B, 0x0606-0x060B, 0x0706-0x070B] [0xFF]

该寄存器设置每个Socket目的硬件地址。

例)如果Socket 0的目的硬件地址 = 08.DC.00.01.02.10FF0C，如下配置，

0x0406	0x0407	0x0408	0x0409	0x040A	0x040B
0x08	0xDC	0x00	0x01	0x02	0x0A

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

在设置TCP连接的时候，该寄存器设置每个Socket目的IP地址。在主动模式下，IP地址需要在执行Connect命令之前设置。在静态模式下， W3150A+建立连接然后被对等点IP内部更新。

在UDP模式，在接收对等点的ARP响应之后。该寄存器的值决定用户写入的值。在接收对等点的ARP响应之前，该寄存器需要重置该值。

例)如果Socket 0目的IP地址 = 192.168.0.11，如下配置。

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

Sn_DPORT (Socket n 目的端口寄存器) [R/W] [0x0410-0x0411, 0x0510-0x0511, 0x0610-0x0611, 0x0710-0x0711] [0x00]

该寄存器设置每个设置TCP连接时用到的socket目的端口号。在主动模式，端口号需要在执行Connect命令之前设置。在静态模式下， W3150A+建立连接然后被对等点端口号内部更新。

在UDP模式，在接收对等点的ARP响应之后。该寄存器的值决定用户写入的值。在接收对等点的ARP响应之前，该寄存器需要重置该值。

例)如果Socket 0目的端口 = 5000(0x1388)，配置如下所示，

0x0410	0x0411
0x13	0x88

Sn_MSS (Socket n最大报文长度寄存器) [R/W] [0x0412-0x0413, 0x0512-0x0513, 0x0612-0x0613, 0x0712-0x0713] [0x0000]

该寄存器用来设置一个包的最大报文长度。

按照通信模式的不同，该寄存器会有不同的值。

例) 1. 普通TCP模式，MSS = 1460(0x05B4)

2. PPPoE-TCP模式，MSS = 1452(0x05AC)

3. 普通UDP模式，MSS = 1472(0x05C0)

4. PPPoE-UDP模式，MSS = 1464(0x05B8)

普通的TCP模式如下设置，

0x0412	0x0413
0x05	0xB4

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

该IP协议寄存器是用来设置IP层原始模式的IP头协议字段。通过注册IANA可以提前定义一些协议码号。对于整个IP使用的上层协议识别码，参考IANA在线文档(http://www.iana.org/assignments/protocol-numbers)。

例) 网络控制消息协议(ICMP) = 0x01，网络组管理协议 = 0x02。

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

该寄存器设置IP头的TOS字段。

Sn_TTL (Socket *n* IP生存时间寄存器) [R/W] [0x0416,0x0516,0x0616,0x0716] [0x80]

该寄存器设置IP头的TTL字段。

Sn_TX_FSR (Socket *n* TX空闲大小寄存器)[R] [0x0420-0x0421, 0x0520-0x0521, 0x0620-0x0621, 0x0720-0x0721] [0x0800]

该寄存器通知用户可以传输的数据大小信息。为了数据的传输，用户应该首先检查该值然后控制传输的数据大小。当检查该寄存器时，用户应该首先读取上层的字节*(0x0420,0x0520,0x0620,0x0720)*然后读取下层的字节*(0x0421,0x0521,0x0621,0x0721)*来获得正确的值。

例)如果S0_TX_FSR中是2048(0x0800)，

0x0420	0x0421
0x08	0x00

整个大小可以根据TX内存空间寄存器决定。在传输的过程中，它会随着数据的传输而减少，而且在数据传输结束后会自动的增加。

Sn_TX_RR (Socket *n* TX读指针寄存器) [R] [0x0422-0x0423, 0x0522-0x0523, 0x0622-0x0623, 0x0722-0x0723] [0x0000]

该指针表明了TX内存中传输结束的地址。随着Socket *n*命令寄存器的SEND命令，它传送数据从现在的Sn_TX_RR到Sn_TX_WR，并且在传送结束后自动的改变。因此，在传送结束后，Sn_TX_RR和Sn_TX_WR将有同样的值。当读该寄存器时，用户应该首先读取上层的字节(0x0422，0x0522，0x0622，0x0722)然后读取下层的

字节(0x0423, 0x0523, 0x0623, 0x0723)来获得正确的值。

Sn_TX_WR (Socket *n* TX写指针寄存器) [R/W] [0x0424-0x0425, 0x0524-0x0525, 0x0624-0x0625, 0x0724-0x0725] [0x0000]

该寄存器提供些传输数据的位置信息。当读该寄存器时，用户应该首先读取上层的字节(0x0424，0x0524，0x0624，0x0724)然后读取下层的字节(0x0425，0x0525，0x0625，0x0725)来获得正确的值。

例)如果S0_TX_WR是2048(0x0800)，

0x0424	0x0425
0x08	0x00

但该值本身并不是要写的物理地址。所以，物理地址应该按下面计算。

1. Socket *n* TX基地址(在这以后称*gSn_TX_BASE)*和Socket *n* TX掩码地址(在这以后称 *gSn_TX_MASK)* 的计算基于TMSR值。*获取更多细节，请参考5.1 初始化 的伪代码。*
2. 这两个值的按位与操作，在socket的TX内存范围中Sn_TX_WR和*gSn_TX_MASK*给出了偏移地址(在这以后称*get_offset*)的结果。
3. *get_offset*和*gSn_TX_BASE*这两个值加到一起可以得出物理地址(在这以后称*get_start_address*)。

现在，可以往*get_start_address*写你想传送的数据大小。(* 这里有一种情况当写入的大小超过socket的TX内存上限。如果是这种情况，写该传送的数据到上限，然后改变物理地址到*gSn_TX_BASE*。接着，写剩下的要发送的数据。)

在这之后，确定增长的Sn_TX_WR值和数据大小一样多，这表明写数据的大小。最后，发送SEND命令到Sn_CR (Socket *n*命令寄存器)。

获取恒多细节，参考5.2.1.1. TCP 服务器模式 传输部分的伪代码。

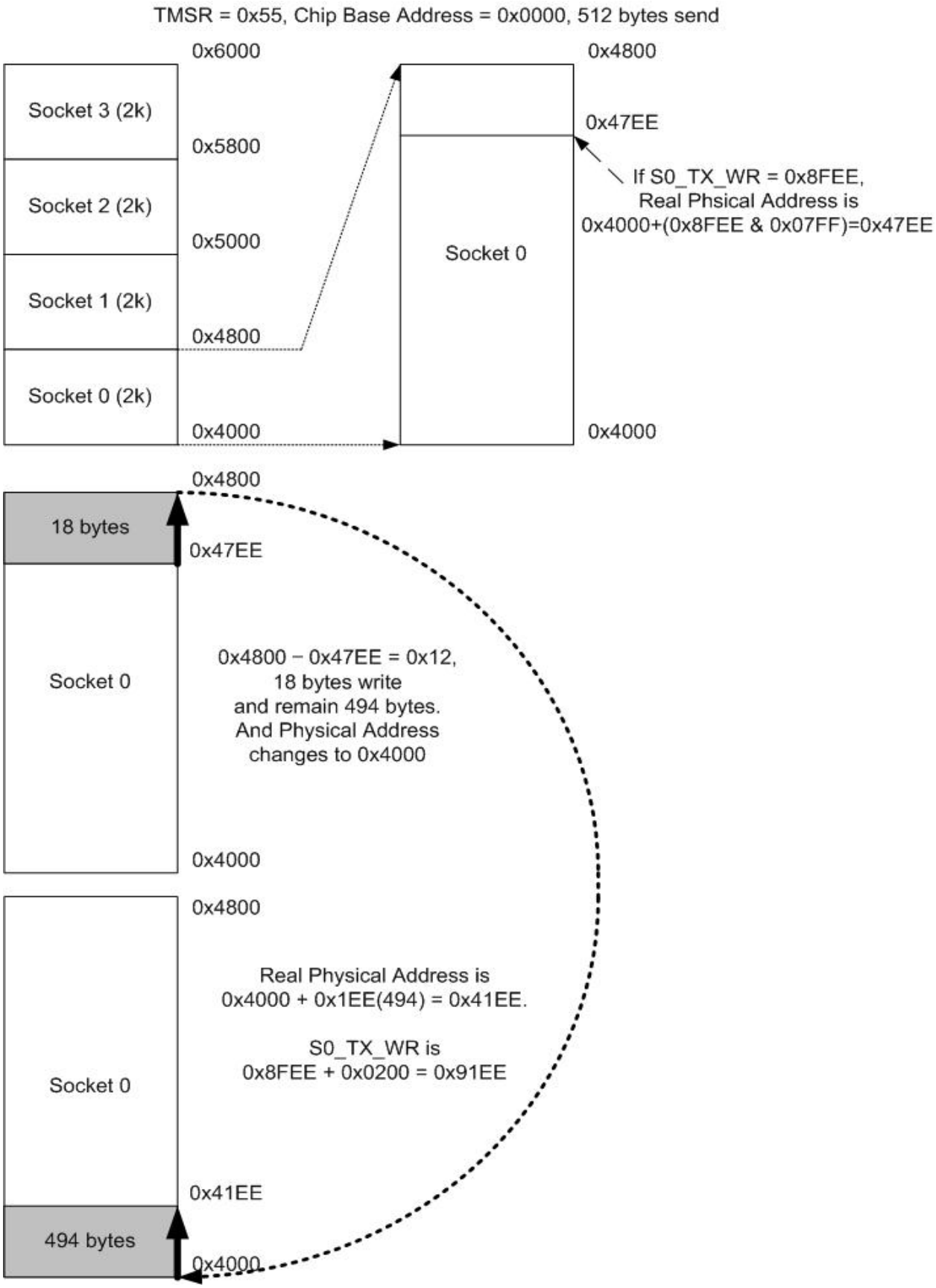


图 4-2. 计算物理地址

Sn_RX_RSR (RX接收大小寄存器) [R] [0x0426-0x0427, 0x0526-0x0527, 0x0626-0x0627, 0x0726-0x0727] [0x0000]

该寄存器通知RX内存接收到的数据大小。该值是通过Sn_RX_RD和n_RX_WR的值内部计算出来的，它会通过Socket n的RECV命令自动的改变。

命令寄存器(Sn_CR)和为远端接收出具。当读该寄存器时，当读该寄存器时，用户应该首先读取上层的字节*(0x0426,0x0526,0x0626,0x0726)*然后读取下层的字节*(0x0427,0x0527,0x0627,0x0727)*来获得正确的值。

例) 如果S0_RX_RSR中是2048(0x0800)，

0x0426	0x0427
0x08	0x00

整个大小可以根据RX内存大小寄存器决定。

Sn_RX_RD (Socket n RX读指针寄存器) [R/W] [0x0428-0x0429, 0x0528-0x0529, 0x0628-0x0629, 0x0728-0x0729] [0x0000]

该寄存器读取接收数据的位置信息。当读该寄存器时，用户应该首先读取上层的字节(0x0428, 0x0528, 0x0628, 0x0728)然后读取下层的字节(0x0429, 0x0529, 0x0629, 0x0729)来获得正确的值。

例) 如果S0_RX_RD中是2048(0x0800)，

0x0428	0x0429
0x08	0x00

但该值本身并不是要读的物理地址。所以，物理地址应该按下面计算。

1. Socket *n* RX基地址(在这以后称*gSn_RX_BASE)*和Socket *n* RX掩码地址(在这以后称 *gSn_RX_MASK)* 的计算基于RMSR值。*获取更多细节，请参考5.1 初始化 的伪代码。*
2. 这两个值的按位与操作，在socket的RX内存范围中Sn_RX_WR和*gSn_RX_MASK*给出了偏移地址(在这以后称*get_offset*)的结果。
3. *get_offset*和*gSn_RX_BASE*这两个值加到一起可以得出物理地址(在这以后称*get_start_address*)。

现在，可以从*get_start_address*读取接收的数据大小。(* 这里有一种情况当读取的大小超过socket的RX内存上限。如果是这种情况，读取接收数据到上限，然后改变物理地址到*gSn_RX_BASE*。接着，读剩下的接收的数据。)

在这之后，确定增长的Sn_RX_RD值和数据大小一样多，这表明读取数据的大小。(* 必须保证增长的值不能超过接收数据的大小。因此接收阶段之前必须检查Sn_RX_RSR。)最后，发送RECV命令到Sn_CR(Socket *n* 命令寄存器)。

获取恒多细节，参考5.2.1.1. TCP 服务器模式 传输部分的伪代码。

5. 功能描述

通过设置一些寄存器和内存，W3150A+将具有联网的功能。本章描述如何操作。

5.1. 初始化

- 设置网络信息

下面这些寄存器用来根据网络环境来设置基本的网络配置信息。

1. 网关地址寄存器 (GAR)
2. 源硬件地址寄存器 (SHAR)
3. 子网掩码寄存器 (SUBR)
4. 源IP地址寄存器 (SIPR)

源硬件地址寄存器(SHAR)是MAC层用的硬件地址，这是制造商已经分配好的地址。MAC地址是由IEEE分配。获得更多信息，请参考IEEE主页。

- 设置socket内存信息

该一步设置socket tx/rx内存信息。该阶段，每个socket的基地址和掩码地址被固定和保存下来。

```
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.
}
```

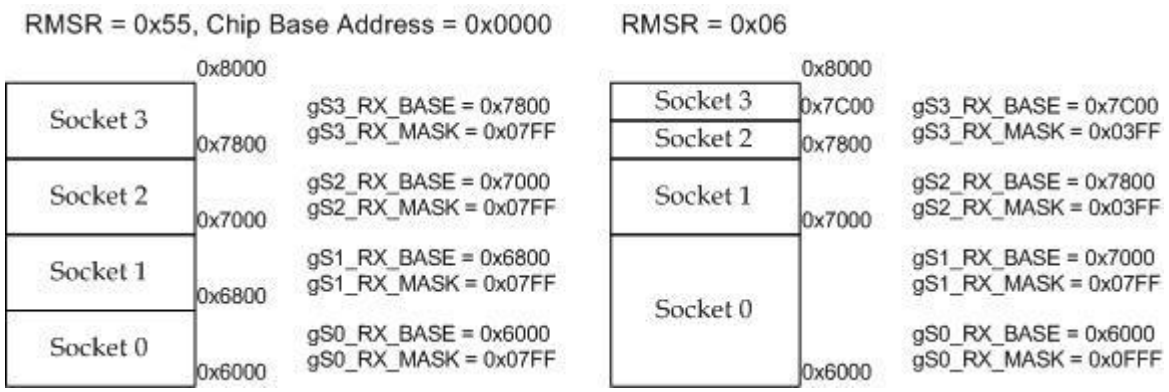


图 5-1. 如果RMSR = 0x55

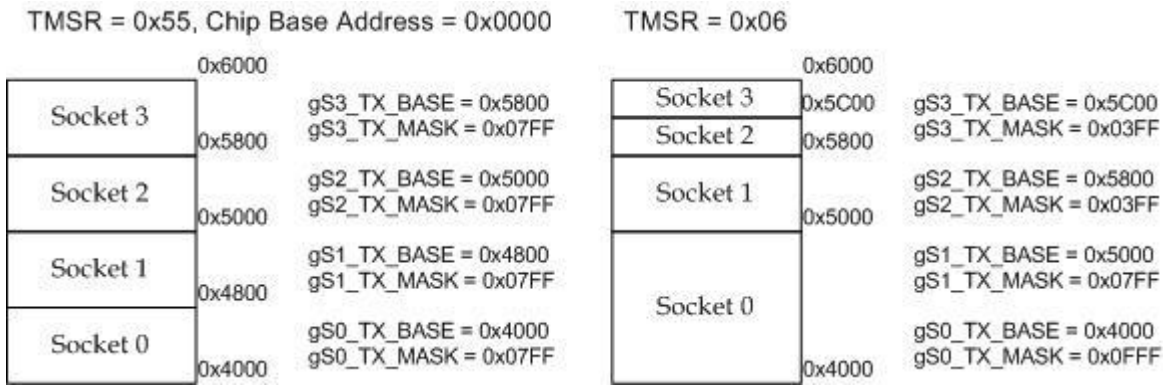


图 5-2. 如果TMSR = 0x55

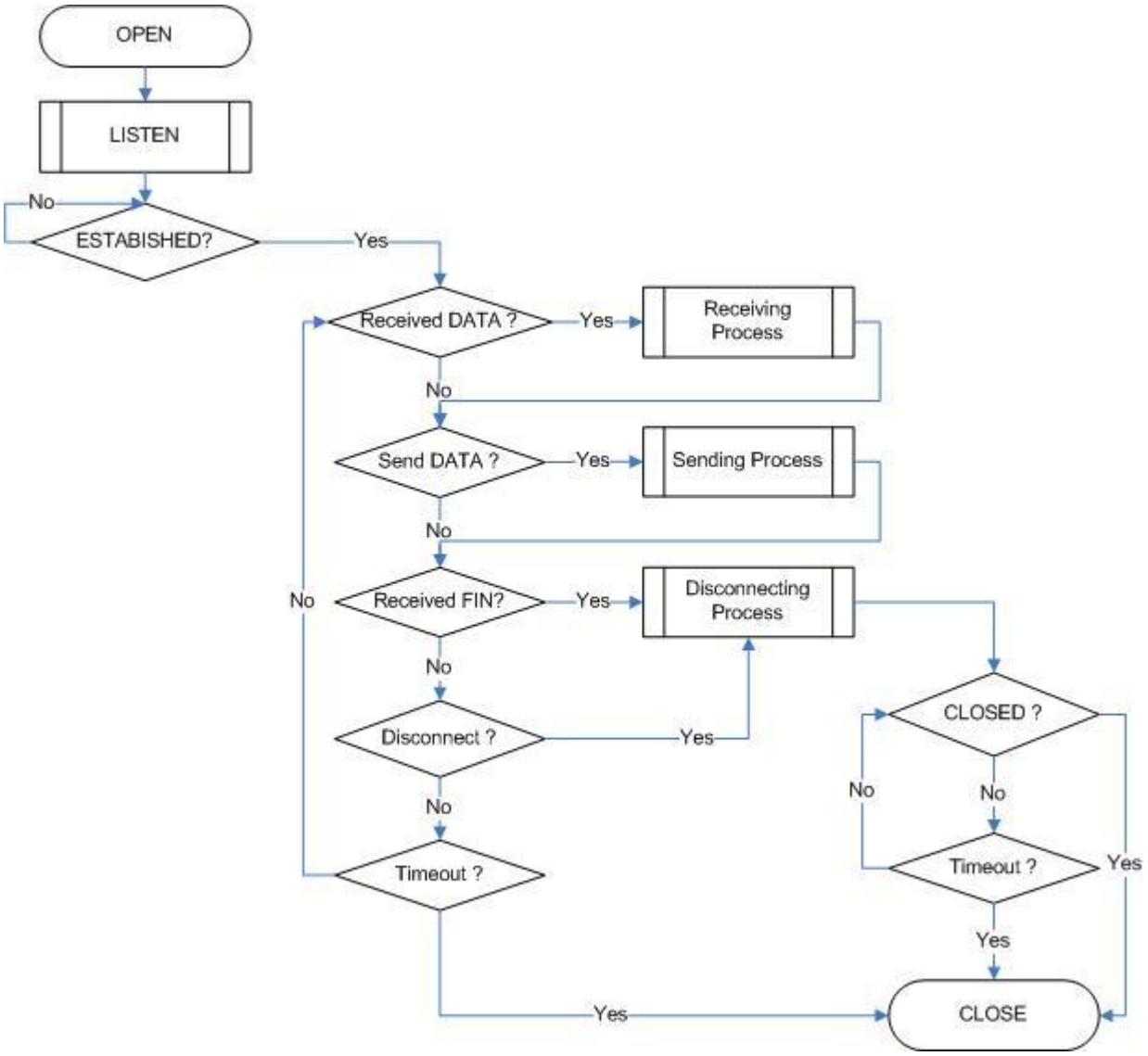
5.2. 数据通信

数据通信可通过TCP、UDP、IP-Raw和MAC-Raw实现。为了选择通信方式，通信的socket(W3150A+支持4个sockets)可以配置Socket *n*模式寄存器(S*n*_MR)的协议字段。

5.2.1. TCP

TCP 是面向连接的通信方法，需要提前建立连接，通过系统 IP 地址和端口号的连接传递数据。有两种方法建立此连接。一种方法服务器模式(被动打开)，等待连接请求。另一种方法是客户端模式(主动打开)，发送连接请求到服务器。

5.2.1.1. 服务器模式



■ Socket初始化

初始化socket *n*为TCP方式，

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

■ 监听

用于等待来自远端主机的连接请求。

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

■ 建立？

如果接收到来自远端主机的连接请求(SOCK_SYNRECV的状态)，W3150A+发送ACK包并且改变为SOCK_ESTABLISHED状态。该状态能被下面方法检查到。

```
First method :
{
    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). */
}
```

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

连接已经建立，就可以进行数据传送和接收。

■ 建立：接收数据？

按下面方法检查来自远端的数据是否被接收到。

```
First method :
{
    if (Sn_RX_RSR != 0x0000) goto Receving Process stage;
}
```

```
Second Method :
{
    If (Sn_IR(RECV bit) == '1') goto Receving 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). */
}
```

■ 建立：接收处理

接收的数据按下面方式被处理

```
{
    /* 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;
}
```

■ 建立：发送数据?/发送处理

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;
}
```

■ 建立：接收FIN?

等待来自远端的连接终止连接请求。

是否接收到远端终止连接请求按下面方法检查。

```
First method :
{
    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). */
}
```

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

■ 建立：断开连接？/断开连接处理

检查用户是否请求断开该连接。

为了断开连接，如下处理，

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

■ 建立：关闭?

没有连接状态。按下面方法检查，

```
First method :
{
    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). */
}
```

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

■ 建立：超时

在数据接收或者连接关闭阶段如果连接关闭是因为远端错误，数据传输将不能被正常处理。这种情况下一段时间之后将发生超时。

```
First method :
{
    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). */
}
```

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

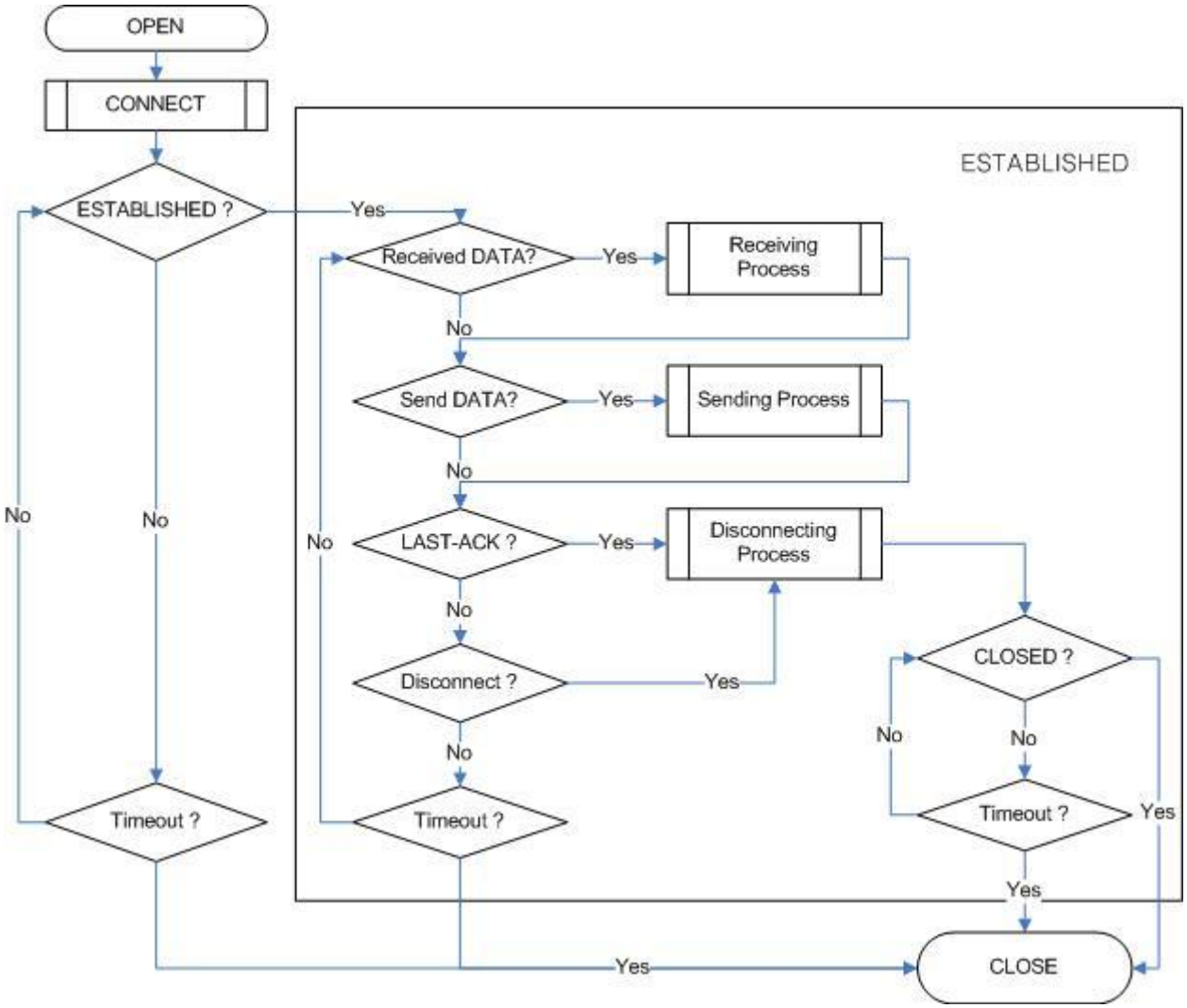
■ Socket关闭

该处理是在数据交换之后连接关闭的情况，超时发生时socke应该关闭，异常操作时有必要强制断开连接。

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

5.2.1.2. 客户端模式

整个过程如下所示。



■ Socket初始化

参考5.2.1.1 服务器(和服务器操作一样)。

■ 连接

发送连接请求到远端主机(服务器)如下所示。

```
{
    /* 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?

连接已经建立。能够按下面方法检查，

```
First method :
{
    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). */
}
```

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

■ 超时

远端没有响应发生超时，Socket被关闭。能够按下面方法检查，

```
First method :
{
    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). */
}
```

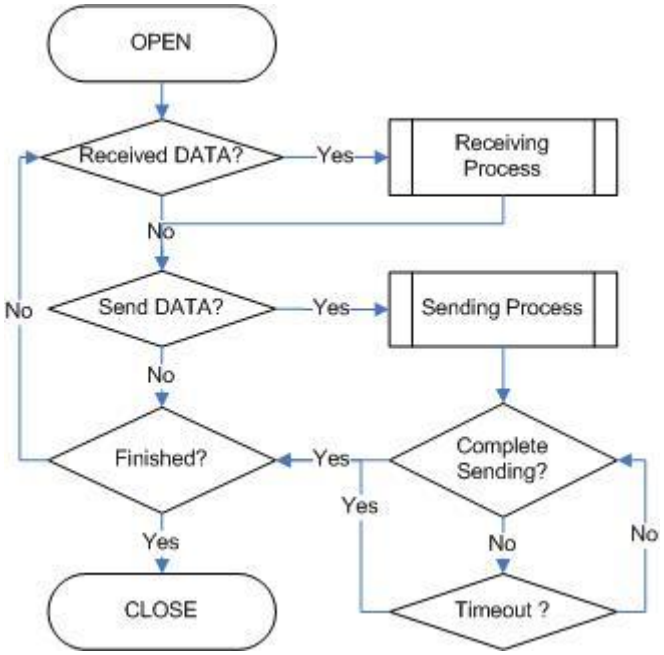
```
Second method :
{
    If (Sn_SR == SOCK_CLOSED) goto CLOSED stage;
}
```

■ 建立

参考5.2.1.1. 服务器(操作和服务器模式一样)。

5.2.2. UDP

UDP提供不可靠和非面向连接的数据报传输结构。UDP处理数据不需要连接建立。因此，UDP消息可能会丢失、重叠或颠倒顺序。如果数据包到达的过快，接收端不能处理所有的数据。如果是这种情况，用户应用需要保证数据传输的可靠性。UDP传输可按下面处理，



■ Socket初始化

初始化socket *n*为UDP

```
{
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;
}
```

■ 接收数据？

如果来自远端的数据被接收到，可按下面方法处理。

```
First method :
{
    if (Sn_RX_RSR != 0x0000) goto Receving Process stage;
}
```

```
Second Method :
{
    If (Sn_IR(RECV bit) == '1') goto Receving 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

接收数据的处理如下。如果是UDP方式，8字节的数据头被加到接收的数据上。数头的结果如下所示。

Destination IP Address (4)	Destination Port (2)	Data size (2) (*data size except for 8byte of header)

```
{
    /* 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？/发送过程

数据传输处理如下。

```
{
    /* 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;
}
```

■ 完成发送?

SEND命令后应该检查发送是否完成。

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

■ 超时

如果远端不存在或者数据传输没有被正常处理将发生超时。能够按下面方法检查。

```
{
    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). */
}
```

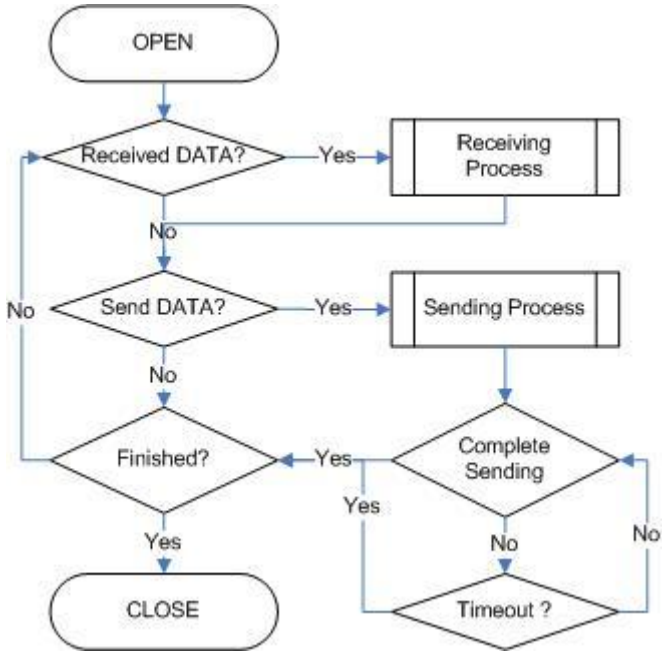
■ 完成? /Socket关闭

如果所有的动作都结束了，关闭socket。

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

5.2.3. IP raw

如果传输层的协议是W3150A+不支持的ICMP或IGMP，可以使用IP Raw模式，需要如下处理。



■ Socket初始化

初始化socket为IP raw。

```
{
START:
    /* sets IP raw mode */
    Sn_MR = 0x03;
    /* sets Protocol value */
    /* The value of Protocol is the value 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;
}
```

■ 接收数据

和UDP一样。参考5.2.2 UDP。

■ 接收处理

和UDP一样。参考5.2.2 UDP，除了数据头信息和数据头大小。

如果是IP raw，6byte数据头被加到接收到的数据上。数据头结构如下。

Destination IP Address (4)	Data Size (2) (*Data size except for 6 bytes of header)

■ 发送数据？/发送过程

和UDP一样。参考5.2.2 UDP，除了远端端口信息没有被使用。

■ 完成发送

■ 超时

■ 完成？/Socket关闭

下一步和UDP一样。参考5.2.2 UDP。

5.2.4. MAC raw

MAC Raw模式(只在socket 0中支持)能被使用。

■ Socket初始化

初始化socket为MAC raw。

```
{
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;
}
```

■ 接收数据？

和UDP一样。参考5.2.2 UDP。

■ 接收过程

MAC raw接收到的以太网数据包包含数据包大小信息。

在MAC raw情况下，2byte数据头被加到接收的数据中。数据头结构如下。

Data Size (2) (*Data size include 2 bytes of header)

■ 发送数据？/发送过程

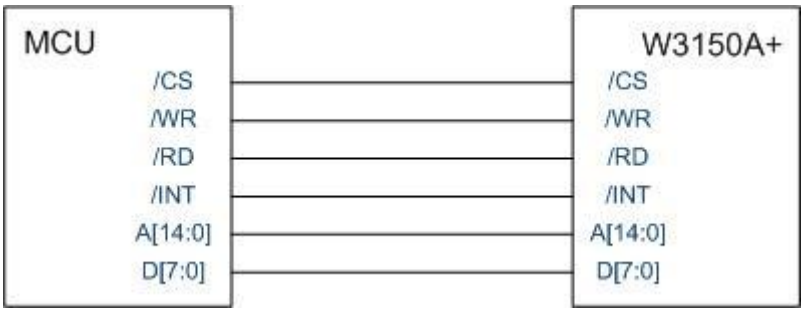
和UDP一样。参考5.2.2 UDP，除了远端端口信息是不需要的。

6. 应用信息

为了与MCU通信，W3150A+提供直接和间接的总线I/F，和SPI I/F模式。为了和以太网PHY通信，使用MII。

6.1. 直接总线 I/F 模式

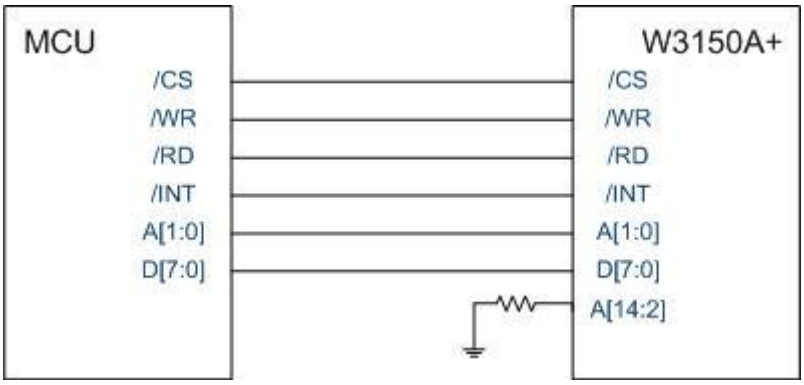
直接总线I/F模式使用15位地址线和8位地址线，/CS，/RD，/WR，/INT。



6.2. 间接总线 I/F 模式

间接总线I/F模式使用2为地址线和8位地址线，/CS，/RD，/WR，/INT。

[14:2]，其他地址线应该下拉。



间接总线I/F模式相关的寄存器如下所示。

值	标志	说明
0x00	MR	执行间接总线I/F模式，地址自动增长。获得更多信息，请参考4. 寄存器说明。
0x01 0x02	IDM_AR0 IDM_AR1	间接总线I/F模式地址寄存器 W3150A+只用大端模式 . 大端模式时 0x01 0x02 IDM_AR0 : MSB | IDM_AR1 : LSB 例)如果读S0_CR(0x0401), 0x01(IDM_AR0) 0x02(IDM_AR1) 0x04 | 0x01
0x03	IDM_DR	间接总线I/F模式数据寄存器

为了读或写内部寄存器或内部TX/RX存储器，

1. 需要写该地址来在IDM_AR0,1上进行读或写操作。
2. 读或写IDM_DR。

为了读或写数据到该有序地址，设置MR(模式寄存器)的AI位。当读或写IDM_DR，IDM_AR值为自动增长1。 该值会在有序地址上被处理，只需要继续读IDM_DR进行读或写。

6.3. 串行外围接口(SPI)模式

串行外围接口模式只使用4个引脚进行数据通信。

4个引脚分别是SCLK、/SS、MOSI和MISO。

W3150A+使用一个引脚来使能SPI操作。该引脚是SPI_EN。

通过把SPI_EN引脚置高，A[14~11]引脚是SCLK，/SS，MOS，MISO。

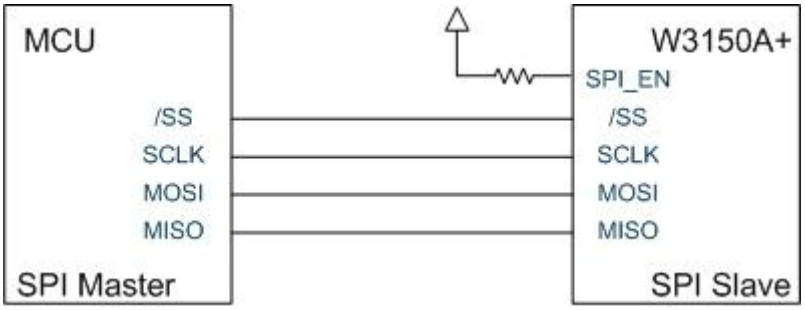


图 6-1.MCU和W3150A+之间的连接

▪ 6.3.1 设备操作

W3150A+由一组指令控制，这组指令通常由处于SPI主方式的主控制器发来。SPI主方式控制器与W3150A+通过SPI总线通信，SPI总线由4个信号组成：：从选择(/SS)，串行时钟(SCLK)，MOSI(主出从进)，MISO(主进从出)。SPI协议定义了4种操作模式(模式0，1，2，3)。每种模式根据SCLK的极性和相位的不同而不同-极性和相位如何控制SPI总线上数据流

W3150A+是SPI从设备，并且支持最常用的模式-SPI模式0和3.

SPI模式0和3唯一的不同是在闲置状态他们的SCLK信号不同。在SPI模式0和3，数据在SCLK的上升沿被锁存，在SCLK时钟的下降沿输出。

▪ 6.3.2命令

按照SPI协议，在SPI设备之间只有两条数据线。所以，有必要定义操作码(OP-Code)。W3150A+使用两种操作码(OP-Code)，读操作码(OP-Code)和写操作码(OP-Code)。除了这两种操作码(OP-Code)，W3150A+忽略其他操作码，不会开始任何操作。

在SPI模式，W3150A+操作在“32位数据流单元”。

32位数据流单元由1字节的操作码(OP-Code)字段、2字节的地址字段和1字节的数据字段组成。

操作码(OP-Code)、地址和数据是先传输最高位有效(MSB)，最后传输最低位有效(LSB)。换句话说，SPI数据的第一位是操作码字段的最高有效位，最后一位是数据字段的最低位有效。W3150A+ SPI数据格式如下。

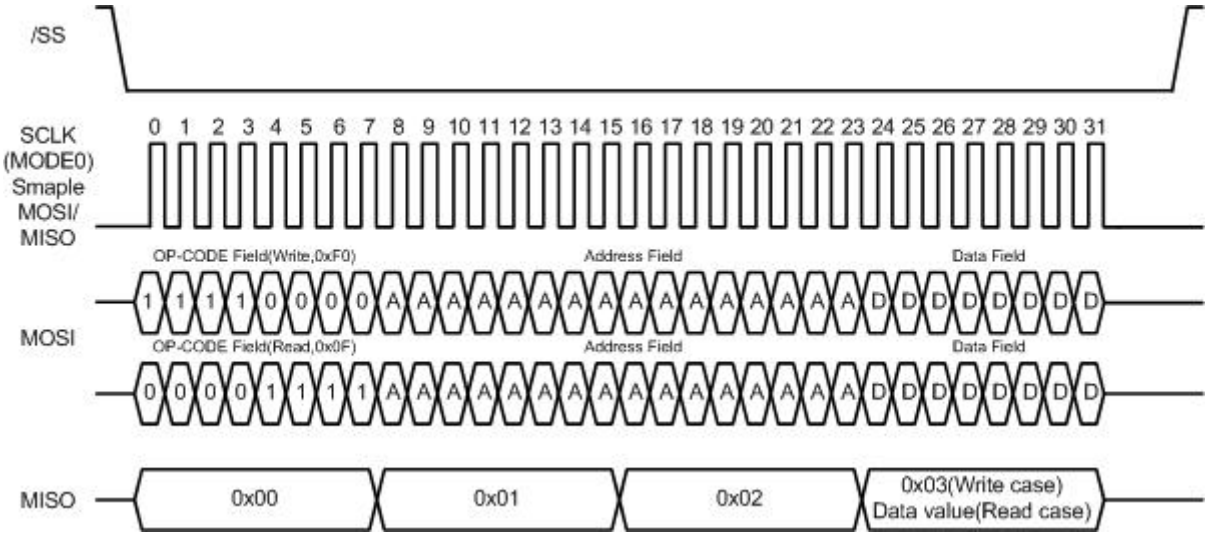
命令	OP-Code字段		地址字段	数据字段
Write operation	0xF0	1111 0000	2 字节	1 字节
Read operation	0x0F	0000 1111	2 字节	1字节

▪ 6.3.3 使用一般的SPI主设备的过程 (根据SPI协议)

- 1. 在SPI主设备引脚上配置输入/输出方向。
  - * /SS (从选择)：输出引脚
  - * SCLK (串行时钟)：输出引脚
  - * MOSI (主出从进)：输出引脚
  - * MISO (主进从出)：输入引脚
- 2. 配置/SS为‘高’
- 3. 配置SPI主设备的寄存器。
  - * SPCR寄存器的SPI使能位(SPI控制寄存器)
  - * SPCR寄存器的主/从选择位
  - * SPCR寄存器的SPI模式位
  - * SPCR寄存器和SPSR寄存器(SPI状态寄存器)的SPI数据速率位
- 4. 在SPDR寄存器(SPI数据寄存器)上写想要传输的值
- 5. 配置/SS为‘低’(数据传输开始)
- 6. 等待接收完成

▫ 7. 如果所有的数据都传输完毕，配置/SS为‘高’

SPI 写/读操作的时序如下。



6.4. MII (介质无关接口)

MII处理W3150A+和物理层设备的数据传输。

MII由用于发送数据的TX_CLK、TXE和TXD[0:3]信号和用于接收数据的RX_CLK、 CRS、RXDV、 RXD[0:3]和COL COL信号组成。

当从W3150A+发送数据，当来自物理层设备的TX_CLK输入变为下降沿时，TXE和TXD[0:3]同步输出数据，因为物理层设备通常识别TX_CLK的上升沿。

当接收数据时，一般物理层设在RX_CLK的下降沿备同步输出CRS、RXDV、RXD[0:3]和COL信号，所以W3150A+识别RX_CLK的上升沿信号。

7. 电气特性

7.1. 极限参数

标志	参数	额定值	单位
V_{DD}	直流电压输出	-0.5到3.6	V
V_{IN}	直流电压输入	-0.5到5.5 (5V允许)	V
I_{IN}	直流输入电流	±5	mA
T_{OP}	操作温度	-40到80 (*参考我们网站的专业报告)	°C
T_{STG}	存储温度	-55到125	°C

***批注：设备超过“最大额定功率”将引起永久损伤。**

7.2. 直流特性

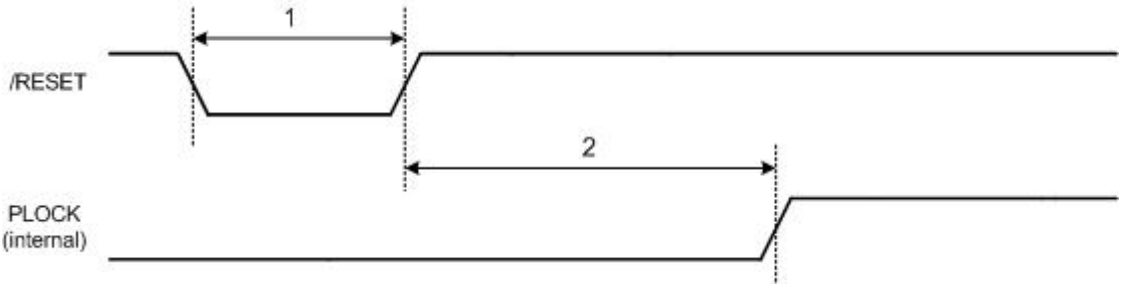
标志	参数	测试条件	最小	类型	最大	单位
V_{DD}	直流电压输出	节点温度从-55°C到125°C	3.0		3.6	V
V_{IH}	输入高电平电压		2.0		5.5	V
V_{IL}	输入低电平电压		- 0.5		0.8	V
V_{OH}	输出到电平电压	I_{OH} = 2, 4, 8, 12, 16, 24 mA	2.0		3.6	V
V_{OL}	输出低电平电压	I_{OL} = -2, -4, -8, -12, -16, -24 mA	0.0		0.4	V
I_I	输入电流	$V_{IN} = V_{DD}$			±5	µA

7.3. 功耗

标志	参数	测试条件	最小	类型	最大	单位
P_{10Base}	10BaseT功耗			16		mA
$P_{100Base}$	100BaseT功耗			24		mA

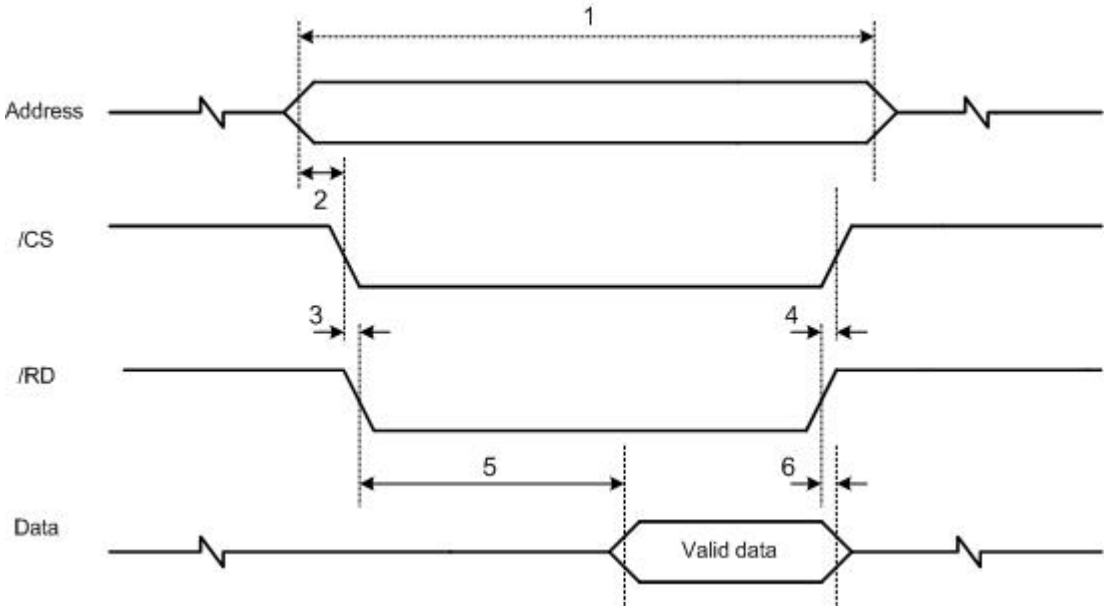
7.4. 交流特性

7.4.1. 复位时序



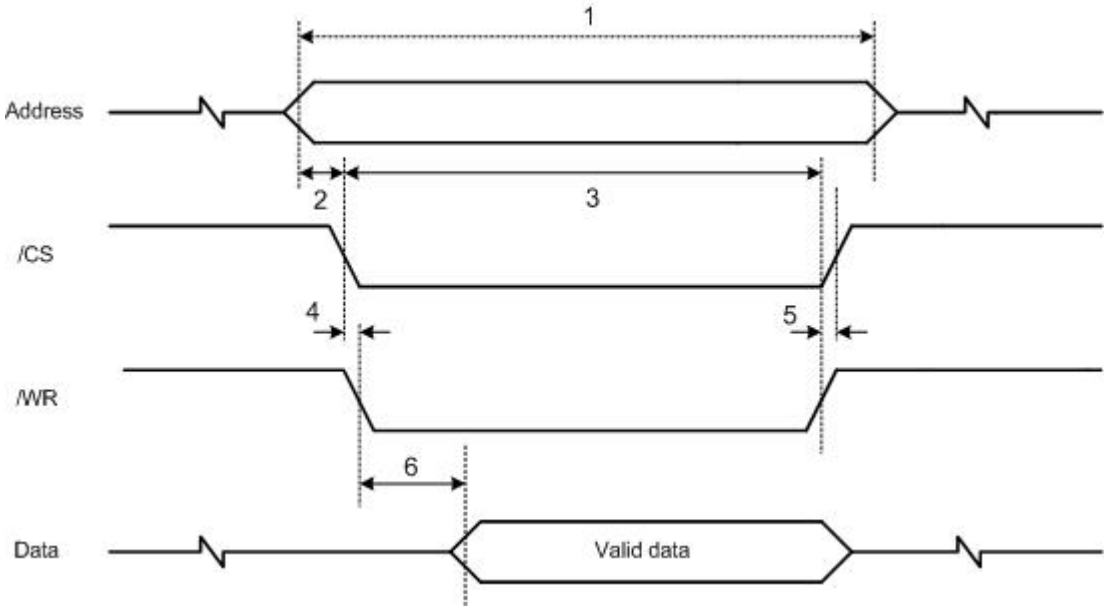
说明	最小	最大
1. 复位周期时间	2 us	-
2. /RESET到内部PLOCK时间	-	10 ms

7.4.2. 寄存器/内存读时序



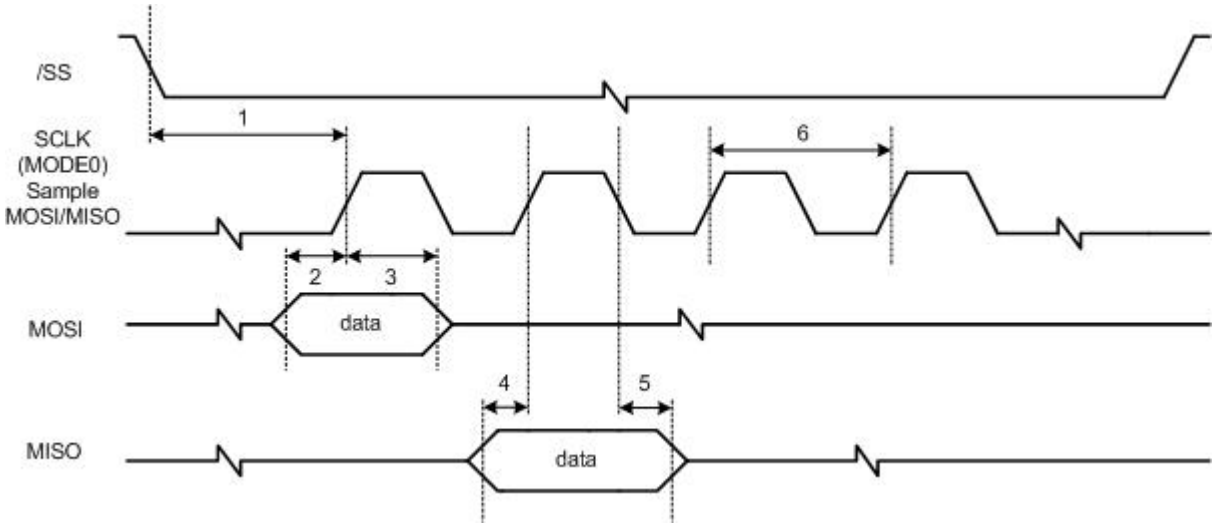
说明	最小	最大
1. 读周期时间	80 ns	-
2. 有效地址到/CS 低电平时间	8 ns	-
3. /CS 低电平到 /RD 低电平时间	-	1 ns
4. /RD高电平到 /CS 高电平时间	-	1 ns
5. /RD低电平到有效数据输出时间	-	80 ns
6. /RD高电平到数据High-Z 输出时间	-	1 ns

7.4.3. 寄存器/内存写时序



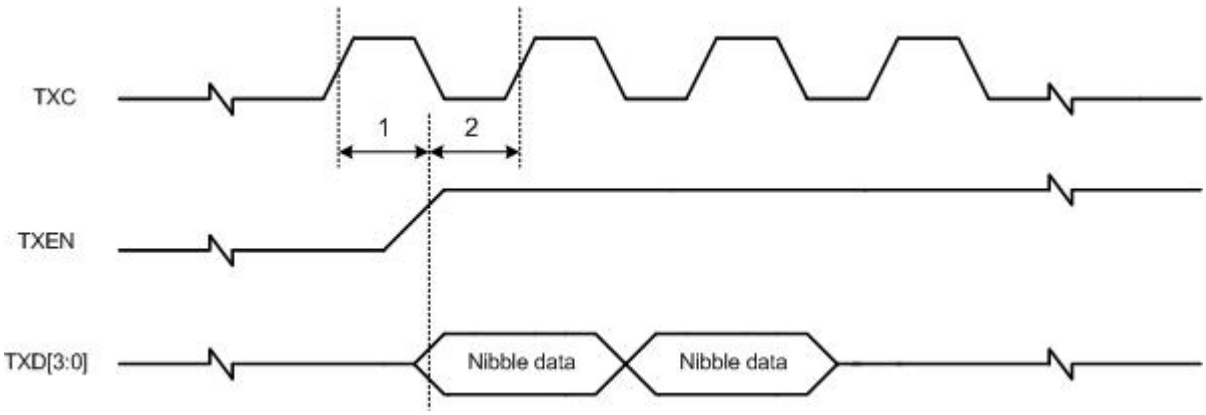
说明	最小	最大
1. 写周期时间	70 ns	-
2. 有效地址到 /CS低电平时间	7 ns	-
3. /CS 低电平到/WR 高电平时间	70 ns	
4. /CS低电平到/WR低电平时间	-	1 ns
5. /WR 高电平到 /CS 高电平时间	-	1 ns
6. /WR低电平到有效数据时间	-	14 ns

7.4.4. SPI 时序



	说明	模式	最小	最大
1	/SS低电平到SCLK	从	21 ns	-
2	输入设置时间	从	7 ns	-
3	输入保持时间	从	28 ns	-
4	输出设置时间	从	7 ns	14 ns
5	输出保持时间	从	21 ns	-
6	SCLK时间	从	70 ns	-
7	SCLK高电平到/SS高电平	从	21 ns	-

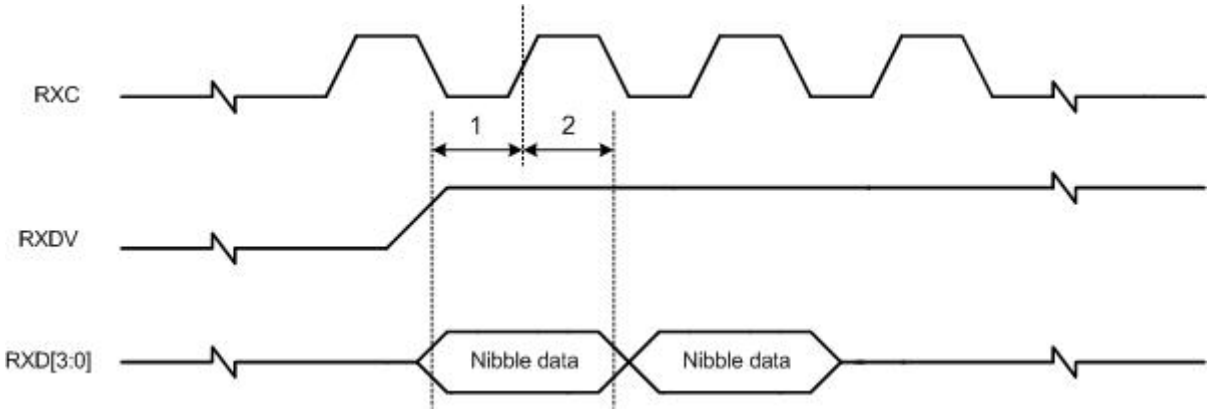
MII(介质无关接口) 时序



■ MII Tx时序

说明	备注	最小	类型	最大
1. TX_CLK 到 TXD, TX_EN	10 Mbps	202 ns	-	205 ns
2. TXD, TX_EN 设置时间到TX_CLK	10 Mbps	195 ns	-	198 ns
1. TX_CLK 到TXD, TX_EN	100 Mbps	22 ns	-	25 ns
2. TXD, TX_EN 设置时间到 TX_CLK	100 Mbps	15 ns	-	18 ns

■ MII Rx时序

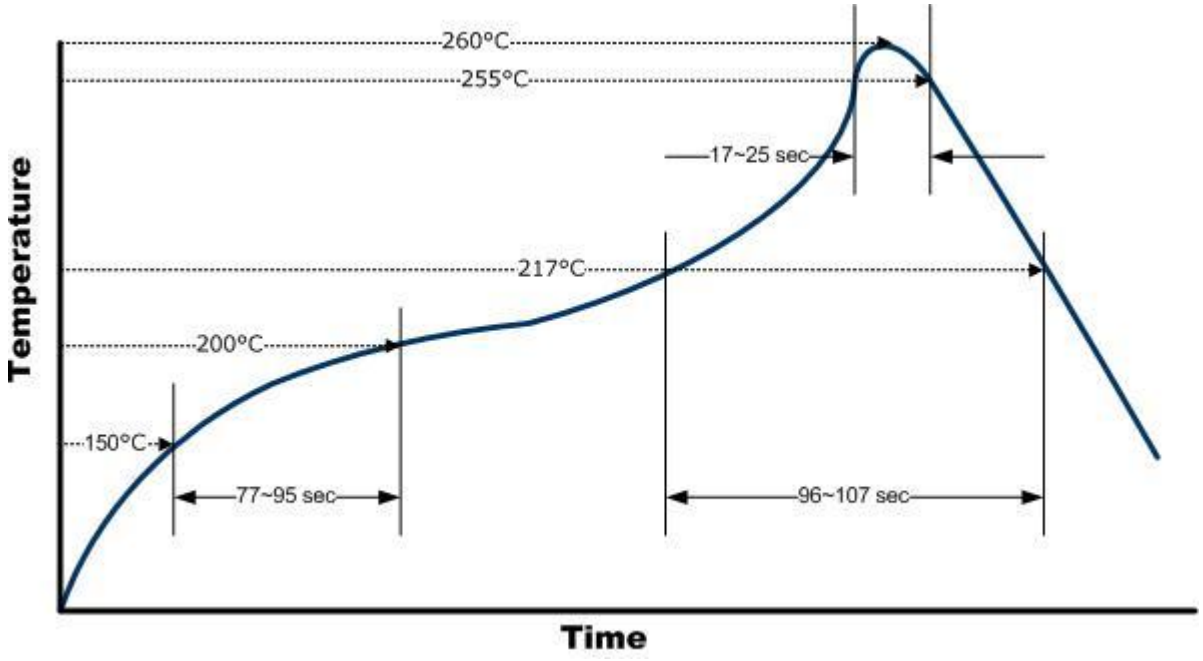


说明	备注	最小	类型	最大
1. 有效数据到RX_CLK 时间(设置时间)	10 Mbps	5 ns	-	-
2. RX_CLK到有效数据时间 (保持时间)	10 Mbps	5 ns	-	-
1. 有效数据到RX_CLK 时间(设置时间)	100 Mbps	5 ns	-	-
2. RX_CLK到有效数据时间 (保持时间)	100 Mbps	5 ns	-	-

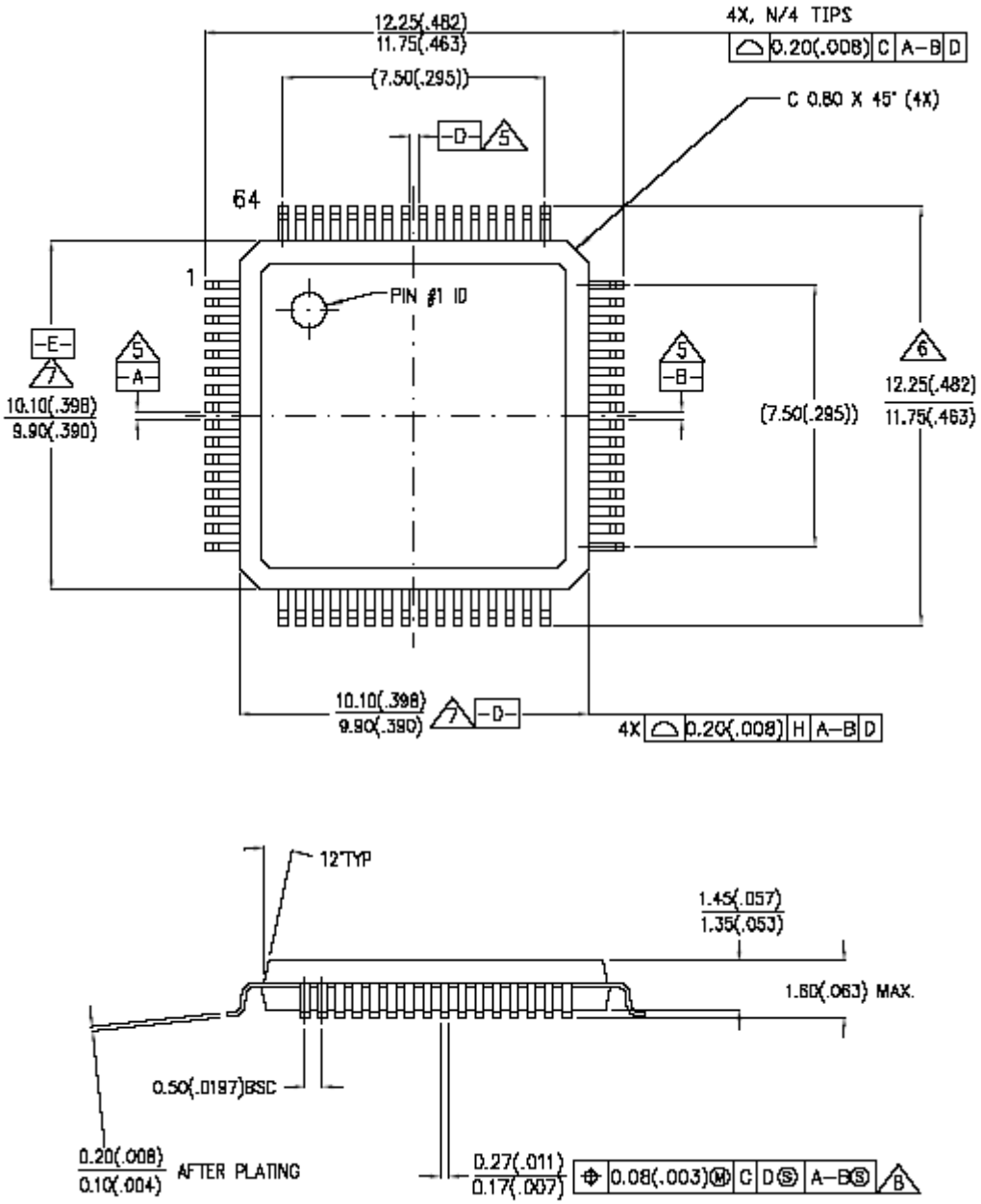
8. 红外回流焊温度曲线(无铅)

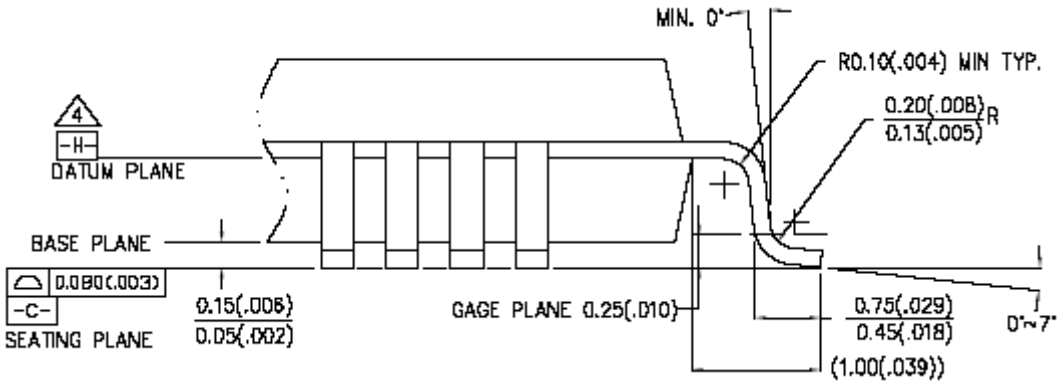
- Moisture Sensitivity Level at 260oC IR Condition: 2.
- Dry Bag Required: Yes
- 1 year out of bag time at max 30°C /60%RH.

Max. Temperature 260°C	
Ramp up rate	< 3°C /second
Pre-heat temperature at 175°C(±25°C)	77-95 seconds
Temperature above 217°C	96-107 seconds
Time within 5°C of actual peak temperature	17-25 seconds
Peak temperature range	258-260°C
Ramp-down rate	< 6°C /second



9. 封装说明





NOTES :

1. PACKAGE DIMENSIONS CONFORM TO JEDEC REGISTRATION MO – 136 – BCD.
2. CONTROLLING DIMENSIONS : MILLIMETERS. INCH ARE SHOWN IN PARENTHESES.
3. DIMENSIONS AND TOLERANCING PER ANSI Y 14.5–1982.
4. DATUM PLANE "H" IS LOCATED AT MOLD PARTING LINE AND IS COINCIDENT WITH THE LEAD EXITS THE PLASTIC BODY AT BOTTOM OF THE PARTING LINE.
5. DATUMS "A–B" AND "D" TO BE DETERMINED AT DATUM PLANE "H".
6. TO BE DETERMINED AT THE SEATING PLANE "C"
7. THESE DIMENSIONS TO BE DETERMINED AT DATUM PLANE "H". DIMENSIONS D AND E DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 0.25MM(.010") PER SIDE.
8. LEAD WIDTH DOES NOT INCLUDE DAMBAR PROTRUSION. ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.08 MM/0.003" TOTAL IN EXCESS OF THIS DIMENSIONS AT MAXIMUM MATERIAL CONDITION.