W7100A中文数据手册

Ver.1.1.3

北京博控自动化技术有限公司

www.bocon.com.cn

1 概述

1.1 介绍

iMCU W7100A是一款功能强大的网络微处理器，它包含一个8051兼容微处理器内核，64K的SRAM，高性能的硬件的TCP/IP协议栈。

W7100A的TCP/IP内核是一个全硬件的、经过多年市场验证的TCP/IP协议栈，并集成了以太网的MAC和PHY。硬件的TCP/IP协议栈支持TCP、UDP、IPv4、ICMP、ARP、IGMP和PPPoE，这些硬件的协议核已经得到多年的应用并证明是成熟的。

1.2 W7100A特征

- 与工业标准的8051在软件上全兼容
- 管道流水线RISC结构，使得指令的运行速度比标准8051快4~5倍
- 内置10BaseT/100BaseTX以太网物理层
- 低功耗模式支持节能
- 硬件的TCP/IP协议，支持TCP、UDP、ICMP、IPv4、ARP、IGMP、PPPoE和以太网
- 支持自动握手协议（全双工/半双工），支持自动极性变换（MDI/MDIX）
- 支持ADSL连接（支持带PAP/CHAP认证模式的PPPoE协议）
- 支持8个独立的SOCKET同时工作
- 32K字节存储器用于网络数据通信
- 多功能LED信号输出（TX、RX、全双工/半双工，IP冲突、连接及速度等指示）
- 不支持IP分片功能
- 双数据指针（DPTR），可用于快速存储器块访问
  - 先进的INC和DEC模式
  - 自动切换当前DPTR
- 64K数据存储器（RAM）
- 256字节数据Flash存储器，用于存储需要掉电保护的数据
- 64K字节程序存储器器，2K字节启动代码存储器；
- 高达16M片外存储器的访问
- 中断控制器
  - 2级中断优先
  - 4个外部中断源
  - 1个看门狗中断
- 4个8位的I/O接口
- 3个16位定时器/计数器
- 全双工串行异步通信收发器（UART）接口
- 可编程看门狗定时器
- DoCD™调试单元

1.3 W7100A功能模块框图及特性

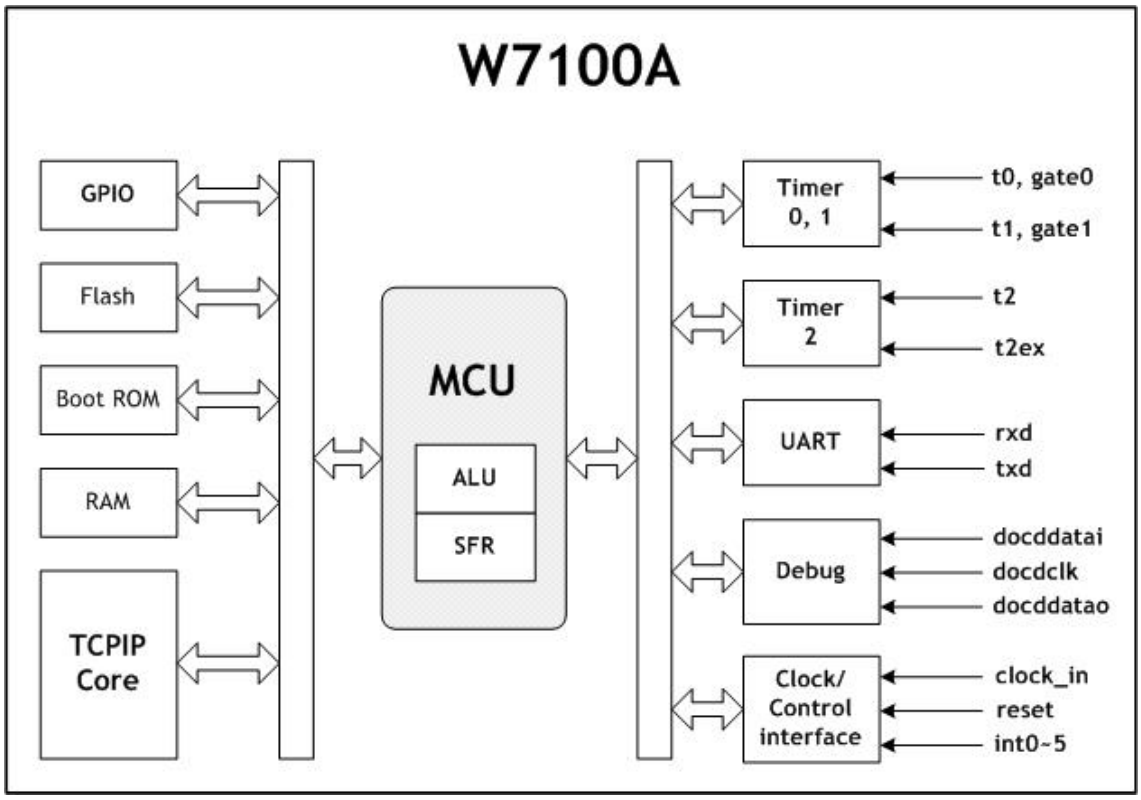


图1.1 W7100A功能框图

W7100A内核各功能模块框图如图1-1所示。每一部分的详细描述如下：

ALU-在指令运行期间执行算数和逻辑运算。它包括累加器（ACC），程序状态字（PSW），B寄存器和相关的逻辑器，如算数运算单元、逻辑运算单元、乘法器和除法器。

SFR - 对特殊寄存器访问的控制。它包括标准的和用户定义的寄存器，以及相关的逻辑器。使用直接寻址指令，可以快速访问用户定义的外设（读、写或修改）。

1.3.1 ALU (算数逻辑单元)

W7100A 与标准的 8051 单片机完全兼容，也保留了 8051 所有的指令助记符和二进制的兼容性。但是 W7100A 增强了很多功能，使得 W7100A 的 MCU 指令运行时速度更快、性能更高。

W7100A 的 ALU（算数逻辑单元）进行数据运算处理，它包括一个 8 位的算术逻辑单元（ALU）、累加器 ACC（0xE0）、B 寄存器（0xF0）和程序状态寄存器 PSW（0xD0）。

ACC (0xE0)

7	6	5	4	3	2	1	0	Reset
ACC.7	ACC.6	ACC.5	ACC.4	ACC.3	ACC.2	ACC.1	ACC.0	0x00

B寄存器在乘法和除法运算过程中使用。在其它情况下可以作为一个通用的特殊功能寄存 器（SFR）使用。

B (0xF0)

7	6	5	4	3	2	1	0	Reset
B.7	B.6	B.5	B.4	B.3	B.2	B.1	B.0	0x00

图1.3 B寄存器

ALU主要进行算数运算，诸如加法、减法、乘法和除法运算，还进行其它的一些操作，如增量/减量运算，BCD与十进制的加调整、比较等运算。在逻辑单元的操作是“与”、“或”、“异或”、“补码”及“ 反 转”等 运 算 。布尔处理单元主要进行“位”操作，如“置1”、“清0” 、“补码”、“如果设置即跳转”、“如果设置即跳转并清除”以及“移动/传送”等操作。

PSW (0xD0)

7	6	5	4	3	2	1	0	Reset
CY	AC	F0	RS1	RS0	OV	F1	P	0x00

图1.4 程序状态字寄存器PSW

CY	进位标志	
AC	辅助进位标志	
F0	通用标志 0	
RS[1:0]	寄存器区选择位	
	RS[1:0]	功能描述
	00	-Bank 0, 数据区地址 0x00 - 0x07
	01	-Bank 1, 数据区地址 0x08 - 0x0F
	10	-Bank 2, 数据区地址 0x10 - 0x17
	11	-Bank 3, 数据区地址 0x18 - 0x1F
OV	溢出标志	
F1	通用标志 1	
P	奇偶标志	

图1.5 PSW寄存器

PSW寄存器包含的位可以反映出MCU的当前运行状态。

1.3.2 TCPIP内核

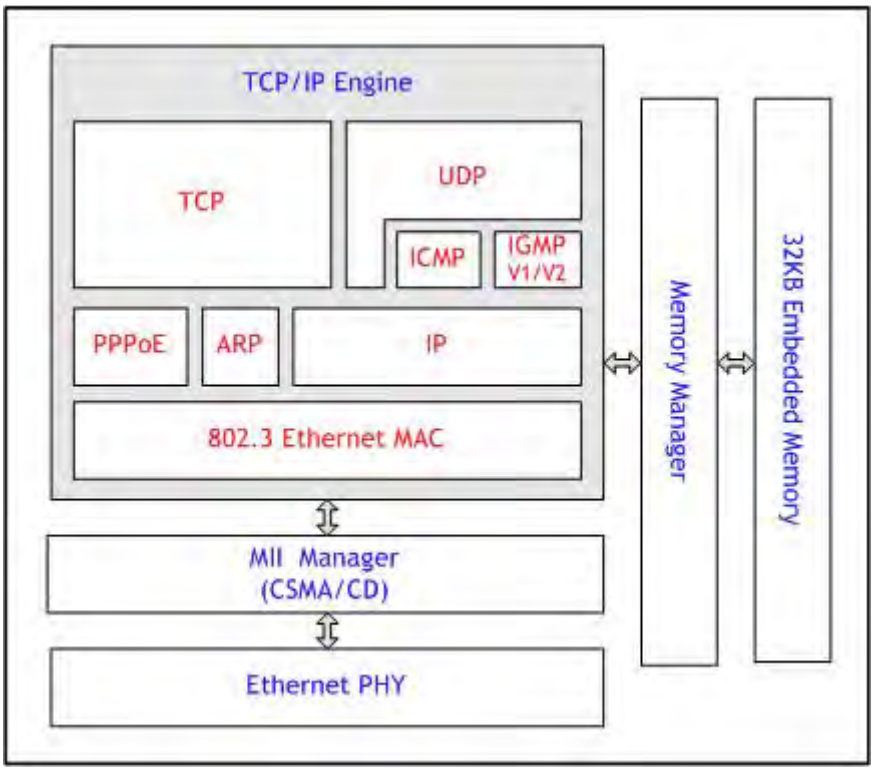


图1.6 TCPIP内核功能框图

以太网物理层（PHY）

W7100A 包含一个 10BaseT/100BaseTX 的以太网物理层。内部物理层支持半双工/全双工，自动握手和自动 MDI/MDIX（自动极性调整）。它还支持 6 个网络状态指示，如连接，TX/RX、IP 冲突、网络速度和全双工/半双工。

TCPIP引擎

TCPIP引擎是基于WIZnet技术的硬件逻辑网络协议。

- **802.3 Ethernet MAC（介质访问控制）**

它控制以太网的CSMA/CD（带冲突检测的载波侦听多路访问）访问。它是基于48位源/目的MAC地址的协议技术。

- **PPPoE（基于以太网的点对点的协议）**

这是在以太网上实现点对点（PPP）服务的协议技术。它将有效载荷（PPP数据帧）封装在以太网的数据帧中并发送。接收时，它解PPP数据帧。PPPoE支持与PPPoE服务器的PPP 通信，并进行PAP/CHAP认证。

- **ARP（地址解析协议）**

ARP是通过IP地址解析MAC地址的协议。它与对端交换ARP请求和ARP应答，以确定相互的MAC地址。

- **IP（网络协议）**

网络协议应用于IP层，提供IP层的数据通信。不支持IP分片，也不能接收分片的数据包。除了TCP或UDP，支持所有协议号。在TCP或UDP时，使用硬件TCPIP协议栈。

- **ICMP（网络控制信息协议）**

ICMP提供网络控制信息，如无法到达的目的地址等。当收到Ping请求的ICMP数据包时，它会响应Ping应答的ICMP数据包。

- **IGMPv1/v2（网络分组管理协议，Ver. 1/2）**

它处理IGMP信息，如加入/离开分组。IGMP只在UDP多播模式下有效。只支持IGMP的版本1和版本2。当使用更高版本的IGMP时，需要在IP层手动实现。

- **UDP（用户报文协议）**

该协议支持UDP层的数据通信。他支持单播、多播和广播等数据报文。

- **TCP（传输控制协议）**

该协议支持TCP层的数据通信。它支持TCP服务器和TCP客户端通信。

1.4 引脚描述

1.4.1 引脚排列

封装形式: LQFP 100

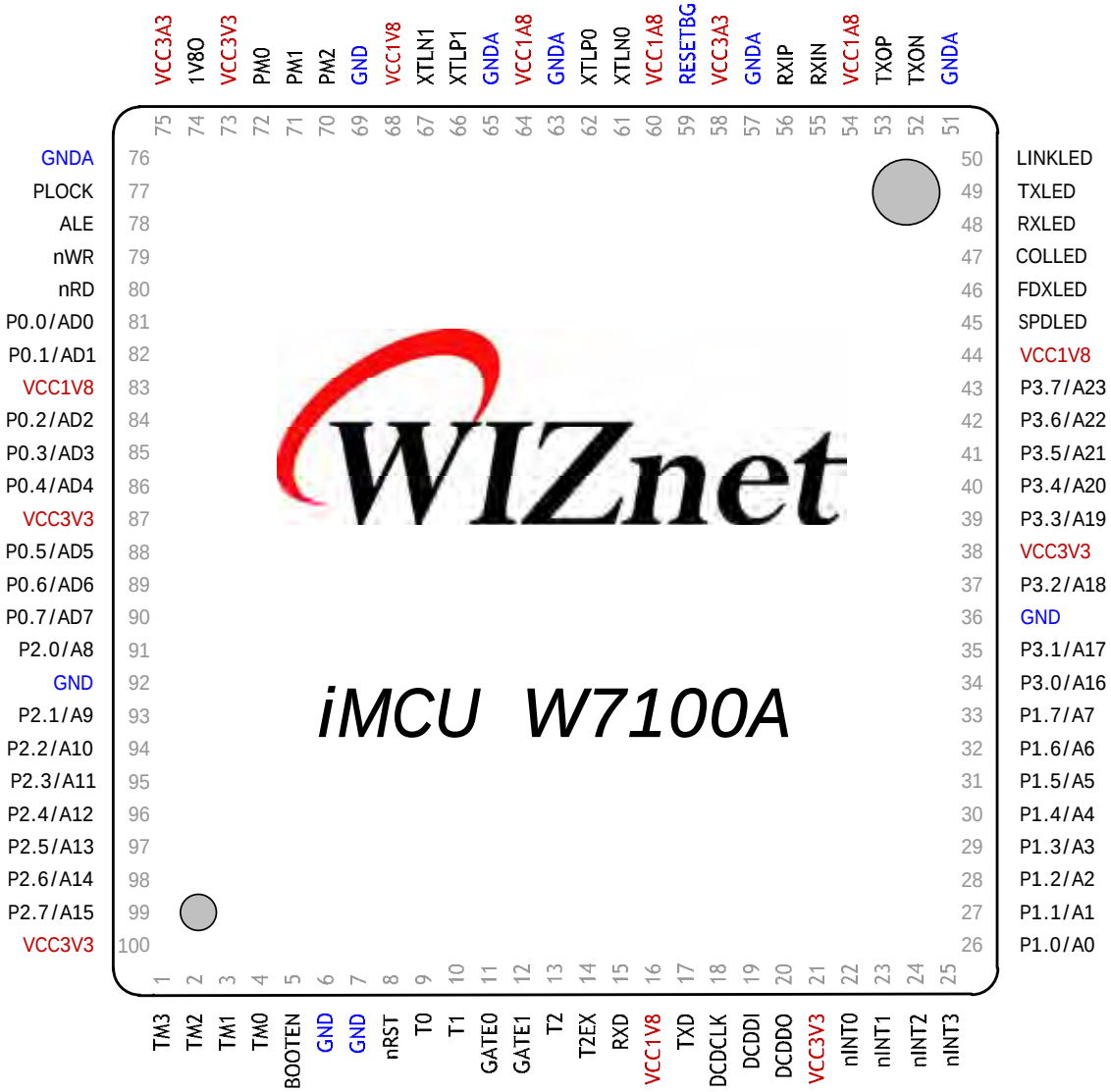


图1.7 W7100A-LQFP100引脚排列

封装形式: QFN 64

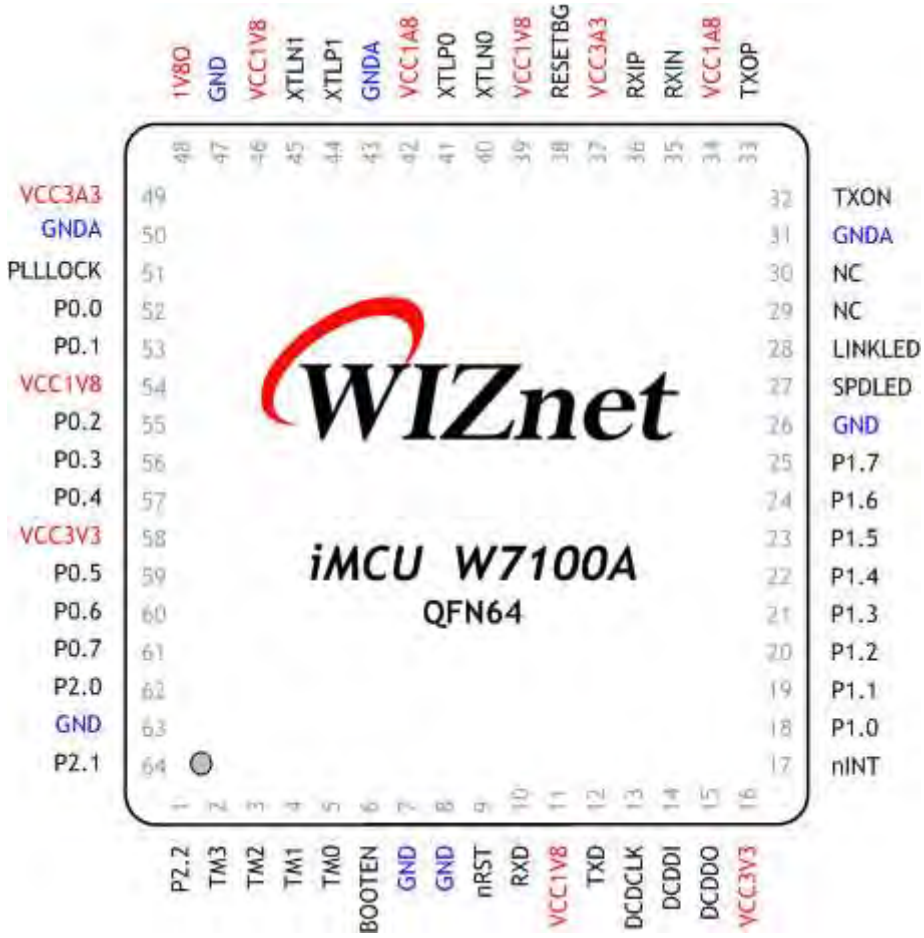


图1.8 W7100A-QFN64引脚排列

1.4.2 引脚描述

管脚的功能描述见下表。所有引脚都是单向的，没有三态输出引脚和内部信号

类型	描述
I	输入
O	8mA电流驱动输出
IO	输入/输出 (双向)
Pu	内部带4.7K Ω电阻上拉
Pd	内部带85KΩ电阻下拉

1.4.2.1配置引脚

<table>
<tr><th rowspan="2">引脚名称</th><th colspan="2">引脚号</th><th rowspan="2">I/O</th><th rowspan="2">Pu/Pd</th><th rowspan="2">Description</th></tr>
<tr><th>100pin</th><th>64pin</th></tr>
<tr><td>nRST</td><td>8</td><td>9</td><td>I</td><td>-</td><td>芯片异步复位，低电平有效</td></tr>
<tr><td>TM3-0</td><td>1,2,
3,4</td><td>2,3
4,5</td><td>I</td><td>Pd</td><td>必须接地，所有的值都为‘0000’</td></tr>
<tr><td>PM2 - 0</td><td>70,
71,
72</td><td>-</td><td>I</td><td>Pd</td><td>PHY模式
<table>
<tr><th colspan="3">PM</th><th rowspan="2">描述</th></tr>
<tr><th>2</th><th>1</th><th>0</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>正常运行模式
尽最大能力的自动握手</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>以100BASE-TX FDX/HDX自动握手</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>以10BASE-TX FDX/HDX自动握手</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>保留</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>手动选择100BASE-TX FDX</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>手动选择100BASE-TX HDX</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>手动选择10BASE-TX FDX</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>手动选择10BASE-TX HDX</td></tr>
</table>
FDX： 全双工, HDX： 半双工</td></tr>
<tr><td>BOOTEN</td><td>5</td><td>6</td><td>I</td><td>Pd</td><td>启动代码运行控制Boot Enable/Disable
0 – 启动用户应用程序模式
跳转到FLASH代码存储区0x0000,运行用户应用程序代码
1- 启动Boot运行模式
在Boot ROM中运行Boot程序</td></tr>
<tr><td>PLOCK</td><td>77</td><td>-</td><td>O</td><td>-</td><td>锁相环锁定指示，当内部PLL锁定后输出指示</td></tr>
</table>

1.4.2.2定时器

引脚名称	引脚		I/O	Pu/Pd	描述
	100pin	64pin			
Timer0, 1接口					
T0	9	-	I	-	Timer0外部时钟输入
T1	10	-	I	-	Timer1外部时钟输入
GATE0	11	-	I	-	Timer0时钟运行控制
GATE1	12	-	I	-	Timer1时钟运行控制
Timer2接口					
T2	13	-	I	-	Timer2外部时钟输入
T2EX	14	-	I	-	Timer2捕获/重装载触发器

1.4.2.3UART

引脚名称	引脚号		I/O	Pu/Pd	描述
	100pin	64pin			
RXD	15	10	I	-	串行接收输入端
TXD	17	12	O	-	串行发送输出端

1.4.2.4DoCD™ 兼容的调试器

引脚名称	引脚号		I/O	Pu/Pd	描述
	100pin	64pin			
DCDCLK	18	13	O	-	DoCD时钟信号
DCDDI	19	14	I	-	DoCD数据输入
DCDDO	20	15	O		DoCD数据输出

1.4.2.5中断/时钟

引脚名称	引脚号		I/O	Pu/Pd	描述
	100pin	64pin			
nINT0	22	17	I	-	外部中断0输入
nINT1	23	-	I	-	外部中断1输入
nINT2	24	-	I	-	外部中断2输入
nINT3	25	-	I	-	外部中断3输入
XTLN0	61	40	O	-	TCPIP内核时钟信号输出端。使用25MHz平行共鸣晶体连接到这个端口。如果使用外部振荡信号，该引脚悬空

XTLP0	62	41	I	-	TCPIP内核时钟信号输入端。使用25MHz平行共鸣晶体连接到这个端口。如果使用外部振荡信号，该引脚接振幅1.8v的信号
XTLN1	67	45	O	-	MCU内核晶体输出端。使用11.0592MHz平行共鸣晶体连接到这两个端口。如果使用外部振荡信号，该引脚悬空
XTLP1	66	44	I	-	MCU内核晶体输入端。使用11.0592MHz平行共鸣晶体连接到这两个端口。如果使用外部振荡信号，该引脚振幅1.8v的信号

1.4.2.6GPIO

引脚名称	引脚号		I/O	Pu/Pd	描述
	100pin	64pin			
P0.0	81	52	IO	-	端口0 输入/输出，外部存储器数据位0，地址位0
P0.1	82	53	IO	-	端口0 输入/输出，外部存储器数据位1，地址位1
P0.2	84	55	IO	-	端口0 输入/输出，外部存储器数据位2，地址位2
P0.3	85	56	IO	-	端口0 输入/输出，外部存储器数据位3，地址位3
P0.4	86	57	IO	-	端口0 输入/输出，外部存储器数据位4，地址位4
P0.5	87	59	IO	-	端口0 输入/输出，外部存储器数据位5，地址位5
P0.6	89	60	IO	-	端口0 输入/输出，外部存储器数据位6，地址位6
P0.7	90	61	IO	-	端口0 输入/输出，外部存储器数据位7，地址位7
P1.0	26	18	IO	-	端口0 输入/输出，外部存储器数据位0，地址位0
P1.1	27	19	IO	-	端口1输入/输出，外部存储器地址位1
P1.2	28	20	IO	-	端口1输入/输出，外部存储器地址位2
P1.3	29	21	IO	-	端口1输入/输出，外部存储器地址位3
P1.4	30	22	IO	-	端口1输入/输出，外部存储器地址位4
P1.5	31	23	IO	-	端口1输入/输出，外部存储器地址位5
P1.6	32	24	IO	-	端口1输入/输出，外部存储器地址位6
P1.7	33	25	IO	-	端口1输入/输出，外部存储器地址位7
P2.0	91	62	IO	-	端口2输入/输出，外部存储器地址位8
P2.1	93	64	IO	-	端口2输入/输出，外部存储器地址位9
P2.2	94	1	IO	-	端口2输入/输出，外部存储器地址位10
P2.3	95	-	IO	-	端口2输入/输出，外部存储器地址位11
P2.4	96	-	IO	-	端口2输入/输出，外部存储器地址位12
P2.5	97	-	IO	-	端口2输入/输出，外部存储器地址位13
P2.6	98	-	IO	-	端口2输入/输出，外部存储器地址位14
P2.7	99	-	IO	-	端口2输入/输出，外部存储器地址位15

P3.0	34	-	IO	-	端口3输入/输出，外部存储器地址位16
P3.1	35	-	IO	-	端口3输入/输出，外部存储器地址位17
P3.2	37	-	IO	-	端口3输入/输出，外部存储器地址位18
P3.3	39	-	IO	-	端口3输入/输出，外部存储器地址位19
P3.4	40	-	IO	-	端口3输入/输出，外部存储器地址位20
P3.5	41	-	IO	-	端口3输入/输出，外部存储器地址位21
P3.6	42	-	IO	-	端口3输入/输出，外部存储器地址位22
P3.7	43	-	IO	-	端口3输入/输出，外部存储器地址位23

说明:

1．用户可以通过PxPU/PxPD特殊功能寄存器控制GPIO的驱动电压

2．在那种情况下，GPIO0~3作为外部存储器的地址和数据总线。请参考“2.3 外部数据存储器的访问”

1.4.2.7外部存储器接口（只有LQFP100有此功能）

引脚名称	引脚号	I/O	描述
ALE	78	O	外部数据存储器地址总线[7:0]锁存
nWR	79	OL	外部数据存储器写使能
nRD	80	OL	外部数据存储器读使能

说明: 当用户按照标准的8051接口使用外部数据存储器时，P0[7:0]可传输数据位[7:0]，或通过ALE控制输出地址位[7:0]。

1.4.2.8介质接口信号

引脚名称	引脚号		I/O	Pu/Pd	描述
	100pin	64pin			
TXON	52	32	O	-	TXON/TXOP 信号对。数据的差分信号从 TXON/TXOP 信号端输出到介质
TXOP	53	33	O	-	
RXIN	55	35	I	-	RXIN/RXIP 信号对。外部介质的差分数据信号由 RXIN /RXIP 信号端输入
RXIP	56	36	I	-	
RESETBG	59	38	I	-	PHY 的片外电阻。连接一个 12.3 kΩ±1%的电阻到地。请参阅相关的设计手册

为了提高系统的性能，请注意以下几点：

1. 尽可能使RXIP/RXIN信号线的长度相等
2. 尽可能使TXOP/TXON信号线的长度相等
3. 尽可能使RXIP/RXIN信号线靠近
4. 尽可能使TXOP/TXON信号线靠近
5. 尽可能使RX和TX信号线远离噪声源，如偏置电阻、晶体振荡器等
6. 使RX和TX尽量成对走线

更详细的信息参考“W5100硬件电路设计”。

1.4.2.9网络指示LED

引脚名称	引脚号		I/O	Pu/Pd	描述
	100pin	64pin			
SPDLED	45	27	O	-	连接的速度LED指示输出 低电平：100Mbps 高电平：10Mbps
FDXLED	46	-	O	-	全双工LED指示输出 低电平：全双工 高电平：半双工
COLLED	47	-	O	-	IP地址冲突LED指示输出 低电平：地址冲突检测(只有在半双工时有效)
RXLED	48	-	O	-	接收数据LED指示输出 低电平：在RXIP/RXIN检测到数据
TXLED	49	-	O	-	发送数据LED指示输出 低电平：通过TXOP/TXON端发送数据
LINKLED	50	28	O	-	连接LED指示输出 低电平：检测到10/100M以太网连接

1.4.2.10 电源

引脚名称	引脚号		I/O	Pu/Pd	描述
	100pin	64pin			
VCC3A3	58, 75	37, 49	电源	-	模拟3.3V电源 在VCC3A3与GNDA之间连接一个10uF的钽电容以补偿电源的损耗
VCC3V3	21, 38, 73, 87, 100	16, 58	电源	-	数字3.3V电源 在每一个VCC3V3电源引脚与GND之间加一个0.1uF的去耦电容。使用1uH的铁电体电感将VCC3V3和VCC3A3分开
VCC1A8	54, 60, 64	34, 42	电源	-	模拟1.8V电源 在VCC1A8与GNDA之间连接一个10uF的钽电容和一个0.1uF的去耦电容，以降低内核电源噪声
VCC1V8	16, 44, 68, 83	39, 46, 54, 11	电源	-	数字1.8V电源 在每个VCC1V8引脚与GND之间加一个0.1uF的去耦电容
GNDA	51, 57, 63, 65, 76	31, 43, 50	电源	-	模拟地 在设计PCB时，将模拟电源地尽可能地设计的宽一些

GND	6, 7, 36, 69, 92	7, 8, 26, 47, 63	电源	-	数字电源地 在设计PCB时，将数字电源地尽可能地设计的宽一些
1V8O	74	48	电源	-	1.8V电压调整输出 通过内部电源调整器输出1.8V/150mA给W7100A的内核运行提供电源（VCC1A8，VCC1V8） 在1V8O与GND之间连接一个3.3uF的钽电容以进行输出频率补偿，再连接一个0.1uF的去耦电容去出电源噪声。 1V8O与VCC1V8直接连接，再通过一个1uH的铁电体的电感与VCC1A8连接 注意：1V8O只给W7100A内核提供电源，不能用于其它设备供电

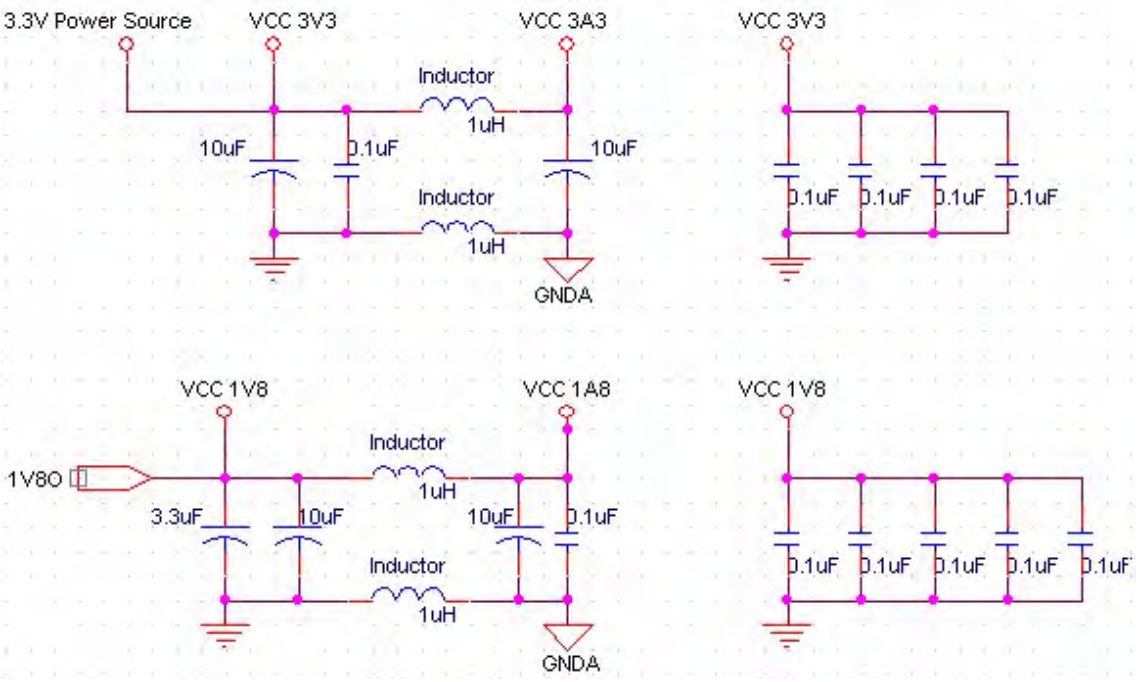


图1.9 电源设计

1.5 QFN64封装描述

1.5.1 LQFP100和QFN64封装的差异

差异	QFN64封装	LQFP100封装
删除的引脚	T0, T1, GATE0, GATE1, T2, T2EX, nINT1, nINT2, nINT3, FDXLED, COLLED, RXLED, TXLED, PM2, PM1, PM0, EXTALE, EXTDATAWR, EXTDATARD, GPIO3[0:7], GPIO2[3:7]	-
外部存储器接口	X	O
PHY模式设置	只能使用SFR	可以使用SFR或PM引脚
GPIO	最多19个引脚	最多可达32个引脚

说明：在QFN64封装中，PHY的工作模式是有由特殊功能寄存器PHYCONF确定的，用户必须设置MODE_EN位，才能运行配置MODE2~0位。设置好MODE2~0位后，通过PHY_RSTn位使PHY复位。复位后，芯片即可以成功地进行初始化并正常运行。用户使用QFN64封装的W7100A，在初始化过程中，必须运行以下的代码。

关于特殊功能寄存器 PHYCONF 更多的信息，请参考 2.5.10 节的“新增加的核扩展的SFR”。

```
PHYCONF |= 0x08;          // 设置MODE_EN位
PHYCONF &= 0xF8;          // 置MODE2 ~ 0位为0 (正常运行模式); 自动配置模式
PHYCONF |= 0x20;          // 设置PHY_RSTn位 (复位PHY)
Delay();                  // 延时以复位PHY (参考第10节 “复位时序”)
PHYCONF &= ~(0x20);       // 清除PHY_RSTn位
```

2 存储器

W7100A的存储器分为两种类型：程序存储器和数据存储器。每一种存储器都可以使用锁定功能。如果存储器被锁定，从外部访问内部存储器将会被拒绝，也不能使用W7100A调试器。关于存储器锁定的详细信息请参考“WizISP编程指南”。W7100A的存储器结构框图如2.1所示。

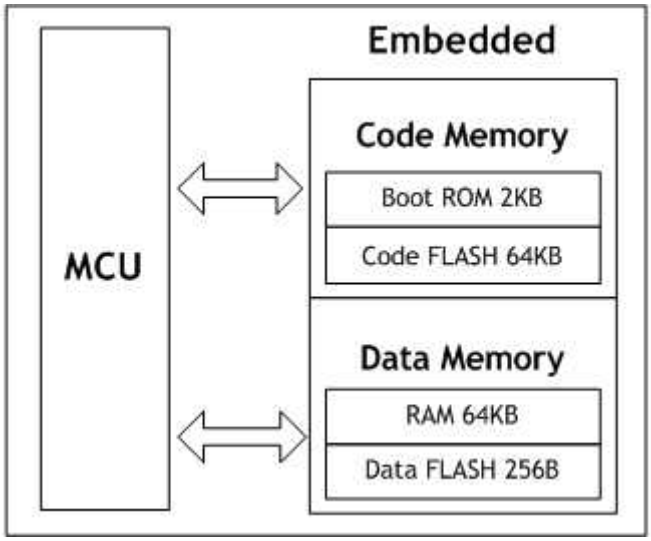


图2.1 程序存储器和数据存储器的结构图

2.1 程序存储器

程序存储器包含启动存储器（BOOT ROM，地址是从0x0000到0x07FFF）和用户代码存储器（地址是从0x0000到0xFFFF）。系统复位后，W7100A总是从程序存储器区运行启动代码。由BOOTEN引脚电平决定运行不同的启动代码。图2.2表示启动代码运行流程。W7100A启动运行后，系统究竟是进入ISP处理还是进入用户应用程序的入口是由BOOTEN引脚电平决定的。如果选择ISP处理（BOOTEN=1），将从BOOT ROM中载入ISP程序。如果选择用户应用程序入口（BOOTEN=0），系统将跳到用户应用程序入口，而不用装载BOOT ROM中的ISP代码。

ISP程序代码应用于WizISP程序装载用户程序代码到W7100A的FLASH存储器。应用程序入口（APP ENTRY）用于运行用户代码。应用程序入口包含“存储器映射开关代码”和跳转代码，使程序切换到FLASH存储器的用户应用程序的起始地址（0x0000）。存储器映射切换如下：

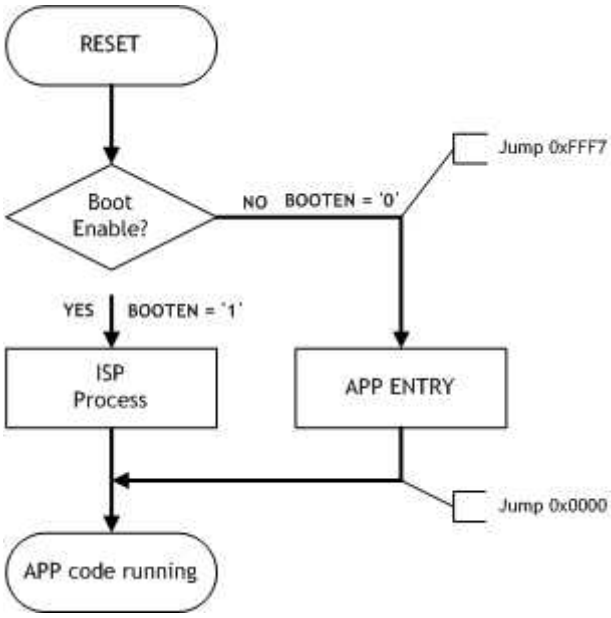


图2.2. 启动顺序流程图

W7100A有两种初始状态 - “BOOT ROM/APP入口”和“FLASH存储器”，如图2.3所示。但因为“BOOT ROM/APP入口”和“FLASH存储器”的地址是重叠的，它们都使用相同的地址值（0x0000 ~ 0x07FF/0xFFF7~0xFFFF）。因此，W7100A将“BOOT ROM/APP入口”和“FLASH（64K）” 分别映射为程序代码存储器和数据存储器。

用户应用程序代码可以写到FLASH中（映射为数据存储器）。但在这种状态下，FLASH不能作为程序存储器，因为这种状态是应用于写入应用程序代码。要将FLASH作程序存储器使用，需要进行存储器映射的切换。为了达到这个要求，用户需要设置BOOTEN为‘0’，以选择用户应用程序模式，启动代码立即跳到应用程序入口，然后应用程序入口将禁止BOOT ROM，且将FLASH存储器映射为程序代码存储器，如图2.3所示。改变程序代码存储器映射后，应用程序入口跳转到程序代码的起始地址（0x0000），工作过程如下：

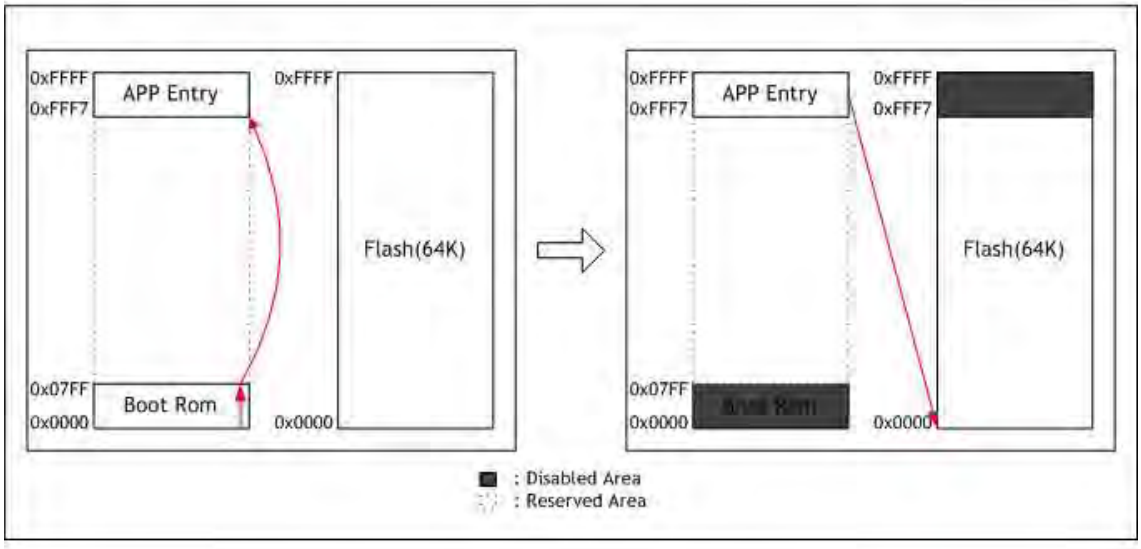


图2.3 应用程序入口过程处理

如果选择APP模式，整个64K的FLASH都将作为程序存储器。但FLASH和应用程序入口的地址都是重叠的。因此，为了完整使用64K FLASH，应用程序的入口地址是不允许从程序存储器映射的。为了不映射应用程序入口，用户需要将特殊功能寄存器WCONF（0xFF）的RB位需要清0。应用程序入口将不被映射，如图2.4所示。

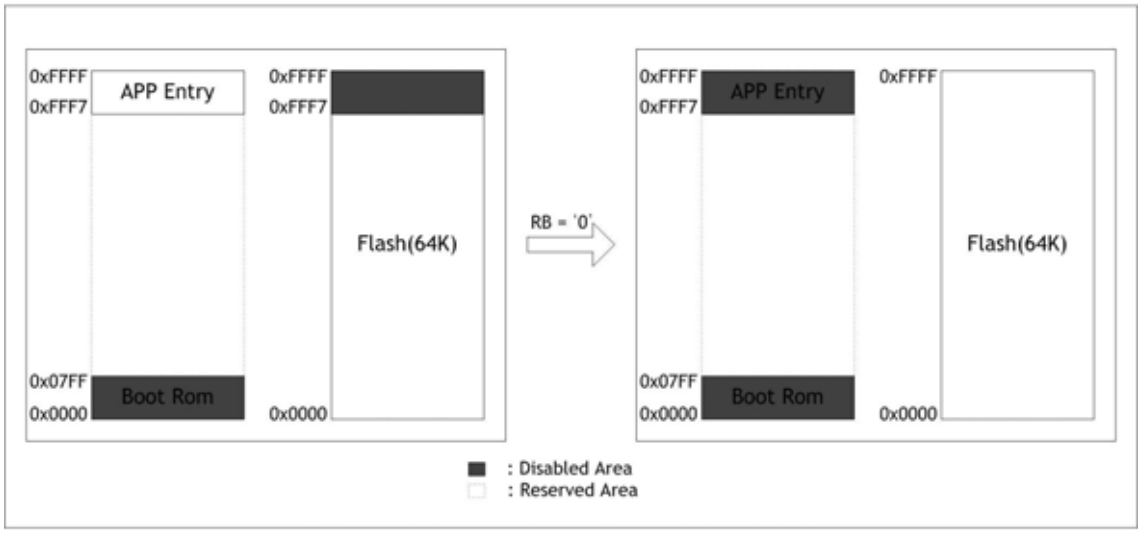


图2.4 RB=0时改变程序存储器状态

WCONF (0xFF)

7	6	5	4	3	2	1	0	Reset
RB	ISPEN	EM2	EM1	EM0	Reserved	FB	BE	0x00

当程序FLASH的地址超过0xFFF7时，下面的代码必须加入到启动代码中。如果使用这种方法，W7100A在系统复位后将立即禁止应用程序入口地址。

```
ANL     0FFH,     #07FH     ; Clear Reboot flag
```

设置引脚BOOTEN为‘0’，且在启动代码中清WCONF寄存器的RB位，那么W7100A内部的64K的FLASH将全部作为程序存储器使用。

2.1.1 程序存储器的等待状态

等待状态是由内部WTST（0x92）进行管理的，等待状态的周期是由WTST寄存器确定的。详细内容请参看2.5.10节“新的和扩展的特殊功能寄存器”。

2.2 数据存储器

W7100A内部有64K字节的RAM、64K字节TCP/IP内核存储器以及256字节的数据FLASH。数据FLASH用于存储一些用户信息，如IP地址，MAC地址，子网掩码，端口号等等。

W7100A-LQFP100还可以寻址外部16M字节的外部数据存储器。图2.5所示为数据存储器的映射。这些存储器只能通过MOVX指令访问。外部存储器可以由用户进行扩展。

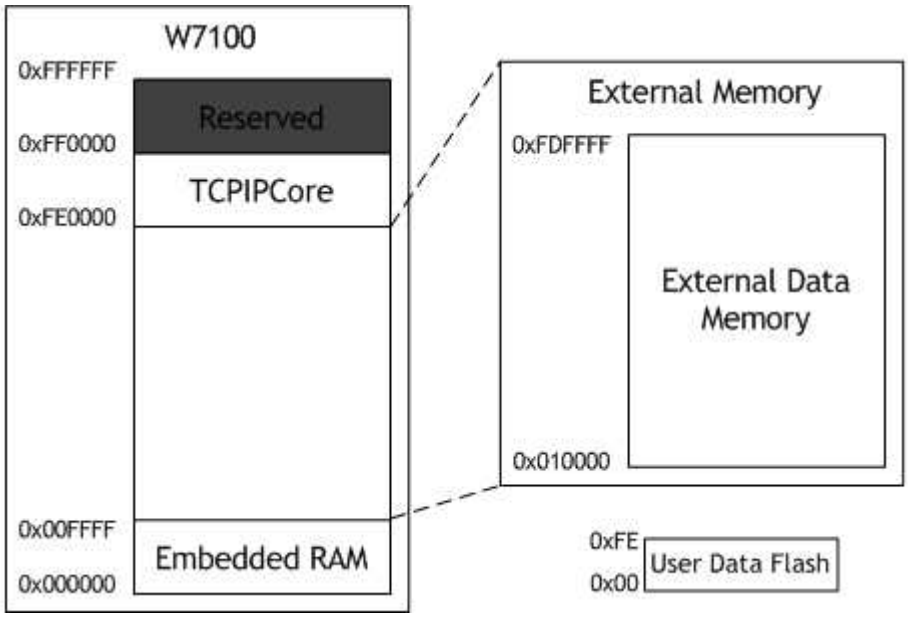


图2.5 数据存储器的映射

2.2.1 数据存储器的等待状态

数据存储器的等待状态是由CKCON（0x8E）进行管理的。详细内容请参看2.5.10节“新的和扩展的特殊功能寄存器”。

2.3 访问外部数据存储器

外部地址线和数据线有两种模式。第一种模式与标准8051相同，采用地址线锁存。第二种模式是地址线直接输出连接。用户也可以将地址线作为通用的IO口使用。请参考第10节“电气特性”，以了解外部访问的速度特性。

表2.1 外部存储器的访问模式

模式	EM[2:0]	P0	P1	P2	P3
标准1	001	Addr[7:0]/Data[7:0]	GPIO	Addr[15:8]	GPIO
标准2	011	Addr[7:0]/Data[7:0]	GPIO	Addr[15:8]	Addr[23:16]
直接1	101	Data[7:0]	Addr[7:0]	Addr[15:8]	GPIO
直接2	111	Data[7:0]	Addr[7:0]	Addr[15:8]	Addr[23:16]

2.3.1 标准的8051接口

该模式与标准的8051单片机访问外部存储器的方法相同。但寻址的空间有变化，参考特殊寄存器WCONG(0xFF)中的EM[2:0](外部存储器的访问模式)的设置。当用户设置EM[2:1]为

“001”时，PORT0用于数据/地址端口，而PORT2用于地址高位[A8~A15]输出。PORT1和PORT3作为通用IO口使用。如下图所示。

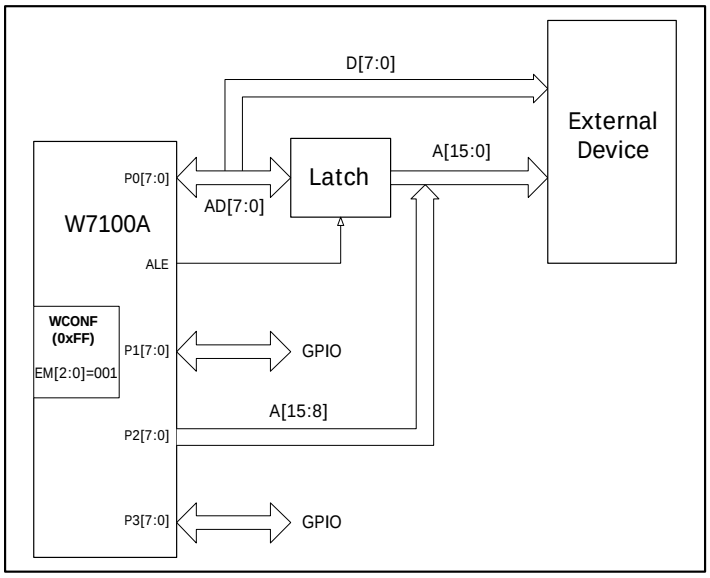


图2.6 标准8051寻址外部存储器 (EM[2:0] = “001”)

当EM[2:0]设置为“011”时，与前面情况一样，PORT0输出地址/数据，PORT2输出地址高8位A[15:8]。但PORT3用于输出最高位的地址A[23:16]，扩展了寻址范围。PORT1作为通用IO口使用。如下图所示。

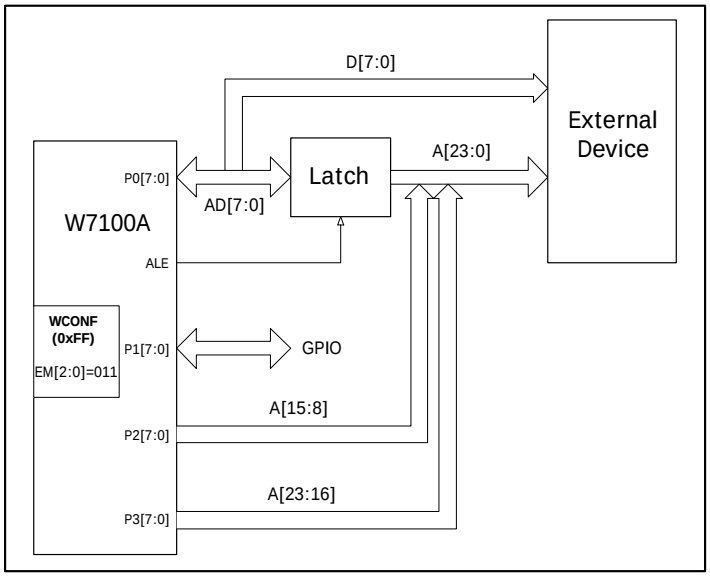


图2.7 标准8051寻址外部存储器(EM[2:0] = “011”)

在标准的8051访问外部存储器模式下，MCU控制ALE（地址锁存）信号以区分地址信号和数据信号。用户可以通过特殊功能寄存器ALECON(0x9F)配置ALE的持续时间。关于ALECON的请参考2.5.10节的“新的核扩展的特殊功能寄存器”。

2.3.2 直接接口

这种方法直接将地址线和数据线与外部存储器连接。当EM[2:0]设置为“101”时，PORT0用于数据线D[7:0]，PORT1用于低地址A[7:0]输出，PORT2用于高地址A[15:8]输出。PORT3作为通用的IO口。通过这种方法，用户可以将地址线直接连接而不需要外部锁存。如下图2.8所示。

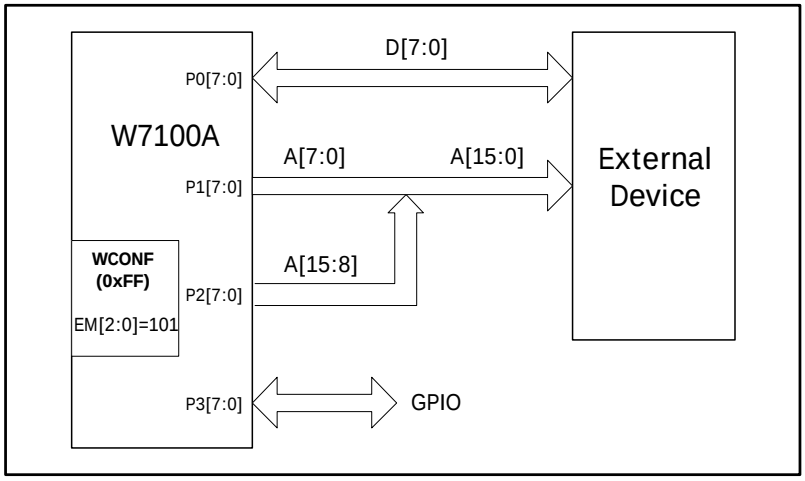


图2.8 直接8051模式访问外部存储器（EM[2:0] = “101”）

当EM[2:0]设置为“111”时，PORT0，PORT1和PORT2的用法语前面一种用法相同，但PORT3用于最高地址A[23:16]，使用这种方法将没有IO口可以使用。如下图所示。

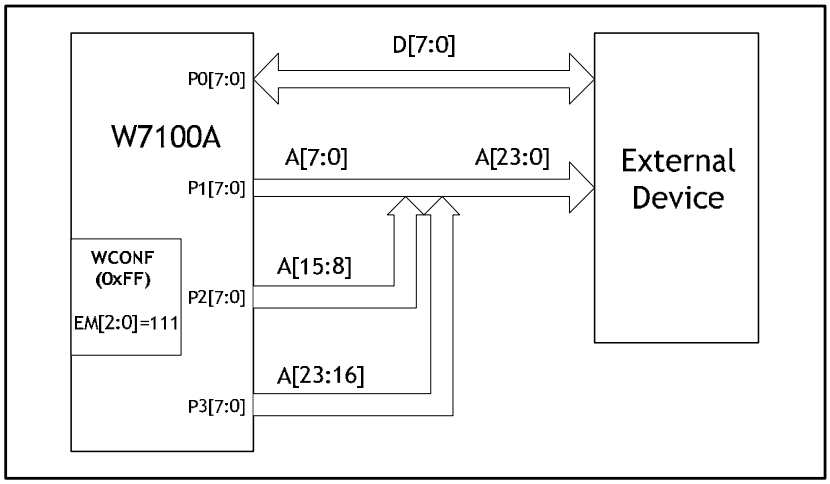


图2.9 直接8051模式访问外部存储器（EM[2:0] = “111”）

2.4 内部数据存储器和特殊功能寄存器(SFR)

下图所示为内部存储器和特殊功能寄存器(SFR)的映射。

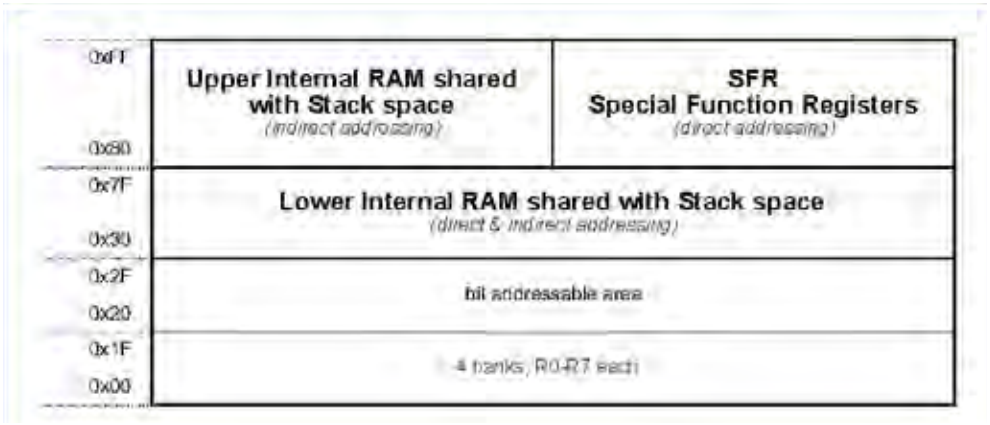


图2.10 内部存储器映射

内部RAM的最低地址有4组寄存器区，每个寄存器区包含8个寄存器；还有一个可以位寻址的16个字节的字节段，起始地址0x20，共128位。一个208字节的缓存区（0x30~0xFF）。0x30~0x7F可以采用直接寻址 ， 也可以间接寻址 ， 而0x80~0xFF则必须采用间接寻址。0x80~0xFF直接寻址区是作为特殊功能寄存器（SFR）。

0xF8	EIP	DPSBK					PHYCONF	WCONF	0xFF
0xF0	B	ISPID	ISPADDR	ISPADDR	ISPDATA	CKCBK	DPX0BK	DPX1BK	0xF7
0xE8	EIE		MXAX	P0_PU	P1_PU	P2_PU	P3_PU	PHY_IND	0xEF
0xE0	ACC			P0_PD	P1_PD	P2_PD	P3_PD		0xE7
0xD8	WDCON				CLK_CNT0	CLK_CNT1	CLK_CNT2	CLK_CNT3	0xDF
0xD0	PSW								0xD7
0xC8	T2CON		RLDL	RLDH	TL2	TH2			0xCF
0xC0						Reserved		TA	0xC7
0xB8	IP								0xBF
0xB0	P3								0xB7
0xA8	IE								0xAF
0xA0	P2								0xA7
0x98	SCON0	SBUF			INTWTST	EXTWTST		ALECON	0x9F
0x90	P1	EIF	WTST	DPX0		DPX1			0x97
0x88	TCON	TMOD	TL0	TL1	TH0	TH1	CKCON		0x8F
0x80	P0	SP	DPL0	DPH0	DPL1	DPH1	DPS	PCON	0x87

图2.11 特殊功能寄存器映射

新的SFR – 新增加的SFR，将在本节详细描述

扩展的SFR – 由标准8051扩展的SFR，也将在本节进行详细描述

标准的SFR – 标准的8051兼容的SFR，本节也有描述

在SFR映射图的左边的寄存器（地址末尾值是0或8）都是可以位寻址的。

2.5 特殊功能寄存器（SFR）的定义

下面章节将描述W7100A及其功能。关于外围设备的特殊功能寄存器，请参考2.5.11节“外设特殊功能寄存器”。

2.5.1 程序存储器“写”允许位

在 PCON 寄存器内有一个程序存储器器“写”允许/禁止位（PWE），它禁止或允许使用 MOVX 指令对程序存储器进行写操作。

当 PWE 设置‘1’时，“MOVX @DPTR, A”指令将数据从累加器 A 写到程序存储器中，地址由 DPTR 寄存器（DPH:DPL）确定。

“MOVX @Rx， A”指令将数据从累加器A写入到程序存储器中，存储器的地址由P2寄存器（位15:8）和Rx寄存器决定（位 7:0）。

PCON (0x87)

7	6	5	4	3	2	1	0	Reset
SMOD0	-	-	PWE	-	0	0	0	0x00

图2.13 PCON寄存器的PWE位

说明：PCON.2 ~ PCON.0是保留位，必须设置为‘0’。

2.5.2 程序存储器的等待状态寄存器

等待状态寄存器提供程序存储器访问时间信息。

WTST (0x92)

7	6	5	4	3	2	1	0	Reset
-	-	-	-	-	WTST.2	WTST.1	WTST.0	0x07

图2.14 程序存储器等待状态寄存器

说明：

1.这些位只有在MCU取指令和MOVC指令时有效。因为程序存储器写操作是MOVX指令，所以CKCON寄存器调节“代码写”的脉冲宽度。

2. 读’操作时间最小\4个时钟周期，最大8个时钟周期。

表2.2 WTST寄存器的值

WTST[2:0]	访问时间[clock]
7	8
6	7
5	6
4	5
3	4
2	未使用
1	未使用
0	未使用

取指令时，程序存储器只能由MOVC指令访问。读程序存储器的操作时间最小3个时钟等待周期，时序图如下。

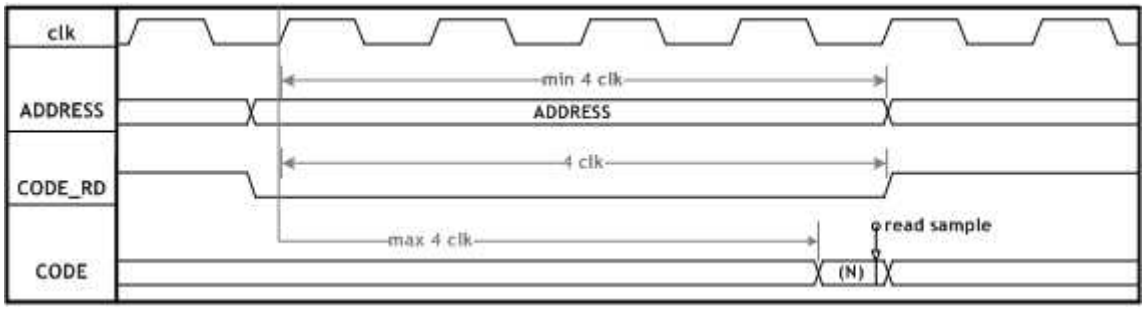


图2.12 最少等待状态（WTST=3）时，对程序存储器同步读的波形图

说明：

1. clk – 系统时钟频率（88.4736 MHz）
2. ADDRESS – 修改程序字节的实际地址
3. CODE_RD – 程序存储器的读信号
4. CODE – 写入到程序存储器的数据

程序存储器可以用MOVX指令以最小3个时钟周期写入。它可以满足W7100A内核对高速和低速存储器进行操作。时序图如下：

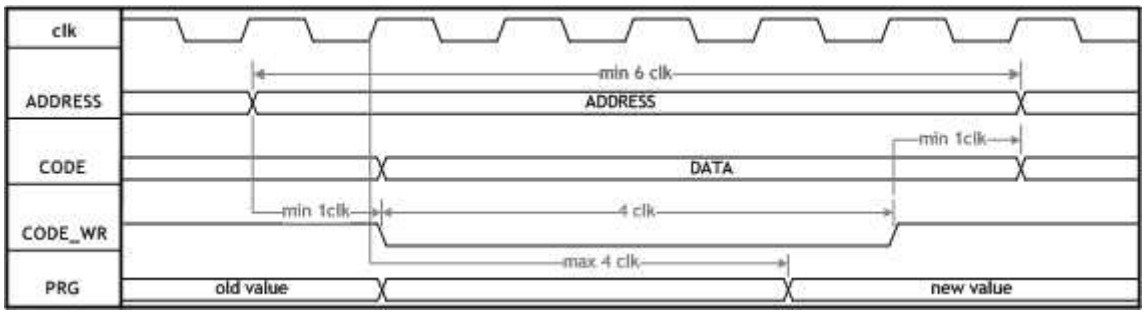


图2.13 最小等待状态（WTST = 3）程序存储器同步写时序图

说明：

1. clk –系统时钟频率（88.4736 MHz）
2. ADDRESS – 修改程序字节的实际地址
3. CODE – 修改程序字节的实际代码
4. CODE_WR – 程序存储器的写信号
5. PRG – 程序存储器

2.5.3 数据指针扩展的寄存器

当访问的地址空间超过64K时，扩展的数据指针寄存器、DPX0、DPX1和MXAX具有重要的意义。复位后，DPX0，DPX1和MXAX恢复为默认的0x00。

DPX0 (0x93)

7	6	5	4	3	2	1	0	Reset
DPXP.7	DPX.6	DPX.5	DPX.4	DPX.3	DPX.2	DPX.1	DPX.0	0x00

图2.17 数据指针扩展寄存器

DPX1 (0x95)

7	6	5	4	3	2	1	0	Reset
DPX1.7	DPX1.6	DPX1.5	DPX1.4	DPX1.3	DPX1.2	DPX1.1	DPX1.0	0x00

图2.18 数据指针扩展寄存器

MXAX (0xEA)

7	6	5	4	3	2	1	0	Reset
MXAM.7	MXAX.6	MXAX.5	MXAX.4	MXAX.3	MXAX.2	MXAX.1	MXAX.0	0x00

图2.19 MOVX @RI扩展寄存器

当MOVX指令使用DPTR0/DPTR1时，最高位地址A[23:16]总是等于DPX0（0x93）/DPX1（0x95）。当MOVX指令使用R0或R1时，最高位地址A[23:16]总是等于MXAX（0xEA）的值，A[15:8]的值等于P2（0xA0）。

2.5.4 数据指针寄存器

双数据指针寄存器应用于数据块的高速拷贝。DPTR0和DPTR1定位于4个特殊功能寄存器地址。当前使用的DPTR由SEL位（0x86.0）选择。如果SEL位为'0'，选择DPTR0（0x83:0x82），否则选择DPTR1（0x85:0x84）。

DPTR0(0x83:0x82)

DPH0(0x83)								DPL0(0x82)								Reset
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	0x0000

图2.20 数据指针寄存器DPTR0

DPTR1(0x85:0x84)

DPH1(0x85)								DPL1(0x84)								Reset
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	0x0000

图2.21 数据指针寄存器DPTR1

DPS (0x86)

7	6	5	4	3	2	1	0	Reset
ID1	ID0	TSL	-	-	-	-	SEL	0x00

图2.22 数据指针选择寄存器

Note: TSL - 交替联动选择允许位。当TSL置位，运行下面的指令时将联动SEL位。

INC DPTR

MOV DPTR, #data16

MOVC A, @A + DPTR

MOVX @DPTR, A

MOVX A, @DPTR

当TSL=0时，DPTR相关的指令将不影响SEL位。

没有应有的位读出时为‘0’或‘1’。

表2.3 DPTR0, DPTR1操作

ID1	ID0	SEL = 0	SEL = 1
0	0	INC DPTR	INC DPTR1
0	1	DEC DPTR	INC DPTR1
1	0	INC DPTR	DEC DPTR1
1	1	DEC DPTR	DEC DPTR1

在下面的指令中将使用选择的数据指针：

MOVX @DPTR, A

MOVX A, @DPTR

MOVC A, @A + DPTR

JMP @A + DPTR

INC DPTR

MOV DPTR, #data16

2.5.5 时钟控制寄存器

时钟控制寄存器CKCON(0x8E)包含MD[2:0]位，它所提供的信息用于控制数据存储器读/写信号的脉冲宽度。

CKCON (0x8E)

7	6	5	4	3	2	1	0	Reset
WD1	WD0	-	-	-	MD2	MD1	MD0	0x07

图2.23 时钟控制寄存器– 延展位

数据存储器的读/写信号在执行MOVX指令时激活。MD[2:0]的作用是调节与IO器件的通信速率，如低速的RAM，LCD显示器等等。复位后，MD[2:0]恢复默认值0x07，低速器件可以在这种状态下正常工作。用户可以通过调节MD[2:0]的值来增加或降低软件的运行速度。MD[2:0]的值可以在程序运行过程中的任何时候进行改变（比如，在两个速度不同的器件之间访问时，可以在相应的MOVX之间改变MD[2:0]的值）

表2.4 MD[2:0]位值

MD[2:0]	脉冲宽度 [clock]
7	8
…	…
2	3
1	未使用
0	未使用

读/写脉冲宽度最小3个时钟周期，而最大8个时钟周期。

2.5.6 内部存储器的等待状态

内部存储器等待状态寄存器INTWTST(0x9C)用于设置内部64K RAM存储器，TCPIP内核和256字节的FLASH的存取访问的时间。

INTWTST (0x9C)

7	6	5	4	3	2	1	0	Reset
RAM WTST		TCPIP内核WTST			Flash WTST			0xFF

图2.24 内部存储器等待状态寄存器

说明：

RAM WTST：设置64K RAM的存取访问时间，有2位，取值范围0~3。

TCPIP内核WTST：设置TCPIP内核的存取访问时间，有3位，取值范围0~7。

Flash WTST：设置内部FLASH的存取访问时间，有3位，取值范围0~7。

内部RAM的WTST值所代表的存取访问时间如表2.5所示。

表2.5 RAM WTST位值定义

WTST	脉冲宽度[clock]
3	5
2	4
1	3
0	2

TCPIP内核和内部FLASH的WTST值所代表的存取访问时间如表2.6所示。

表2.6 TCPIP内核/Flash WTST位值定义

WTST	脉冲宽度[clock]
7	10
6	9
5	8
4	7
3	6
2	5
1	4
0	3

2.5.7 地址锁存允许寄存器

特殊功能寄存器ALECON用于标准的8051外部访问模式。ALE（地址锁存允许）的持续时间可以由ALECON控制。

如果设置ALECON为‘1’，ALE引脚将在1个时钟后跳变为低电平。如果设置ALECON为‘n’，ALE引脚信号将保持“1+n”时钟周期，然后改变为低电平。

ALE的保持时间=“ALECON值 + 1”个时钟周期

ALECON的初始值为0xFF，根据外部器件的速度，用户可以改变这个值。

ALECON (0x9F)

7	6	5	4	3	2	1	0	Reset
AC.7	AC.6	AC.5	AC.4	AC.3	AC.2	AC.1	AC.0	0xFF

图2.25 内部存储器等待状态寄存器

2.5.8 外部存储器等待状态寄存器

特殊功能寄存器EXTWTST用于配置外部存储器的访问时间。EXTWTST寄存器为16位，用户可以设置的范围在0~65535之间。

EXTWTST0 (0x9D)

7	6	5	4	3	2	1	0	Reset
EW.7	EW.6	EW.5	EW.4	EW.3	EW.2	EW.1	EW.0	0xFF

图2.26 外部存储器等待状态的第一个字节

EXTWTST1 (0x9E)

7	6	5	4	3	2	1	0	Reset
EW.15	EW.14	EW.13	EW.12	EW.11	EW.10	EW.9	EW.8	0xFF

图2.27 外部存储器等待状态的第二个字节

2.5.9 堆栈指针

W7100A有一个8位的堆栈指针SP(0x81)，定位在内部RAM空间。

SP (0x81)

7	6	5	4	3	2	1	0	Reset
SP.7	SP.6	SP.5	SP.4	SP.3	SP.2	SP.1	SP.0	0x07

图2.28 堆栈指针寄存器

当数据以PUSH的方式压入堆栈、或执行CALL调用，堆栈指针值将增加。而当数据以POP方式从堆栈弹出、或执行RET、RETI指令时，堆栈指针值将减少。换句话说，堆栈指针永远指向堆栈中的最后一个有效的字节。

2.5.10 新增的和扩展的特殊功能寄存器

PHY_IND(0xEF)：PHY状态指示寄存器，指示W7100A内部物理层(PHY)当前的状态。

PHY_IND (0xEF)

7	6	5	4	3	2	1	0	Reset
					FDX	SPD	LINK	0x00

图2.29 PHY状态寄存器

说明：

FDX：0 – 全双工/ 1 – 半双工

SPD：0 – 100Mbps / 1 – 10Mbps

LINK：0 – 连接失败/1 – 连接成功

ISPID(0xF1)：ISP的ID寄存器

ISPADDR16(0xF2) ：ISP的16位地址寄存器

ISPDATA(0xF4)：ISP的数据寄存器

CKCBK(0xF5)：CKCON备份寄存器

DPX0BK(0xF6)：DPX0备份寄存器

DPX1BK(0xF7)：DPX1备份寄存器

DPSBK(0xF9)：DPX备份寄存器

RAMBA16(0xFA)：RAM基地址寄存器

RAMEA16(0xFC)：RAM终止地址寄存器

PHYCONF (0xFE：W7100A的PHY运行模式、复位、掉电设置寄存器。

PHYCONF (0xFE)

7	6	5	4	3	2	1	0	Reset
-	-	PHY_RSTn	PHY_PWDN	MODE_EN	MODE2	MODE1	MODE0	0x00

图2.30 内部PHY配置寄存器

说明：

PHY_RSTn：对W7100A内部PHY复位。如果用户想使用该位对PHY复位，只需要设置该位为‘1’，复位后然后再清‘0’。关于复位的时间，请参看第10节“电气特性参数”

PHY_PWDN：

1 - 低功耗模式，关闭内部嵌入的以太网物理层以降低电源消耗

0 – 正常运行模式

MODE_EN：

1 – 使用MODE2~0位配置W7100A的运行模式

0 – 不使用MODE2~0

对于QFN封装的W7100A，必须使用MODE2~0来配置W7100A。

MODE2 ~ 0：参考1.4.2节的引脚定义。MODE2~0位的功能与PM2~0完全相同。

保留，必须设置为‘0’。

例：使用MODE2~0选择运行模式

```
PHYCONF |= 0x08;       // MODE_EN位允许
PHYCONF &= 0xF8;       // MODE2 ~ 0为0 (普通运行模式)，自动配置模式
PHYCONF |= 0x20;       // 设置PHY_RSTn位，对W7100A的PHY复位
Delay();               // 延时以保证复位
PHYCONF &= ~(0x20); // 清除PHY_RSTn位，恢复W7100A的PHY工作
```

WCONF(0xFF)：W7100A配置寄存器

WCONF (0xFF)

7	6	5	4	3	2	1	0	Reset
RB	ISPEN	EM2	EM1	EM0	Reserved	FB	BE	0x00

图2.31 W7100A配置寄存器

说明：

RB：0 – ISP结束后不重新启动；1 – ISP结束后重新启动（APP入口（0xFFF7~0xFFFF）‘读/写’允许）

ISPEN：0 - 允许W7100A内置的ISP启动；1 – 禁止ISP启动

EM[2:0]：外部存储器模式。请参看2.3节“外部数据存储器的访问”

FB：ISP状态时Flash忙指示

BE：BOOT启动允许（1 – BOOT启动运行/ 0 – 应用程序运行），只读

CLKCNT0(0xDC)：W7100A内核时钟计数寄存器位0~7。

CLK_CNT0 (0xDC)

7	6	5	4	3	2	1	0	Reset
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	0x00

图2.32 内核时钟计数寄存器

说明：

CLK_CNT是一个32位的特殊功能寄存器。复位后为0。内核的每一个时钟周期其值都加1。该寄存器用于内核时钟的计数或定时。

例〉1秒大约等于88000000时钟计数（如果内核时钟是88MHz）

CLKCNT1(0xDD)：W7100A内核时钟计数寄存器位8~15。

CLK_CNT1 (0xDD)

7	6	5	4	3	2	1	0	Reset
Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8	0x00

图2.33 内核时钟计数寄存器

CLKCNT2(0xDE)：W7100A内核时钟计数寄存器位16~23。

CLK_CNT2 (0xDE)

7	6	5	4	3	2	1	0	Reset
Bit23	Bit22	Bit21	Bit20	Bit19	Bit18	Bit17	Bit16	0x00

图2.34 内核时钟计数寄存器

CLKCNT3(0xDF)：W7100A内核时钟计数寄存器位24~31。

CLK_CNT3 (0xDF)

7	6	5	4	3	2	1	0	Reset
Bit31	Bit30	Bit29	Bit28	Bit27	Bit26	Bit25	Bit24	0x00

图2.35 内核时钟计数寄存器

2.5.11 外设寄存器

P0, P1, P2, P3：IO端口寄存器，详细内容请参看第4节，以了解I/O端口功能。

TCON(0x88)： Timer0/1配置寄存器。请参考5.1节详细了解Timer 0和Time 1的功能。

TMOD(0x89)：Timer0/1控制寄存器。请参考5.1节详细了解Timer 0和Timer1的功能。

TH0(0x8C), TL0(0x8A)：Timer0计数寄存器。请参考5.1节详细了解Timer0的功能。

TH1(0x8D), TL1(0x8B)：Timer1计数寄存器。请参考5.1节详细了解Timer1的功能。

SCON(0x98)：UART配置寄存器。详细信息请参考第6节，了解UART功能。

SBUF(0x99)：UART数据缓冲寄存器。详细信息请参考第6节，了解UART功能。

IE(0xA8)：中断允许中的UART位。详细信息请参考第6节，了解UART功能。

IP(0xB8)：中断优先权寄存器中的UART位。详细信息请参考第6节，了解UART功能。

TA(0xC7)：定时访问寄存器。请参考第7节详细了解看门狗定时器的定时访问功能。

T2CON(0xC8)：Timer2配置寄存器。详细信息请参考5.2节，了解Timer2的功能。

RLDH(0xCB), RLDL(0xCA)：Timer2捕获寄存器。详细信息请参考5.2节了解Timer2的功能。

TH2(0xCD), TL2(0xCC)：Timer2计数寄存器。详细信息请参考5.2节了解Timer2的功能。

PSW(0xD0)：程序状态寄存器。详细信息请参考1.3.1节“ ALU ”。

WDCON(0xD8)：看门狗控制寄存器。详细信息请参考第7节。

3 中断

中断引脚的功能在下表中介绍。所有的引脚都是单向的。没有三态输出和内部信号。

表3.1 外部中断引脚描述

引脚	激活状态	类型	Pu/Pd	描述
nINT0/FA6	Low/Falling	I	-	外部中断0
nINT1/FA7	Low/Falling	I	-	外部中断1
nINT2/FA8	Falling	I	-	外部中断2
nINT3/FA9	Falling	I	-	外部中断3
nINT4			-	保留
TCPIPCore (nINT5)	Falling	I	-	TCPIP内核中断

W7100A内核有两极中断优先权控制。通过设置或清除对应的IP(0xB8)和EIP(0xF8)寄存器的中断优先级位，每一个外部中断可以设置为高优先级或低优先级。外部中断可以是下降沿信号触发或低电平触发。中断请求在系统时钟的上升沿采样。

表3.2 W7100A中断汇总

中断标志	功能	激活状态	标志复位	中断向量	中断号	优先级别
IE0	INT0引脚	Low/Falling	硬件	0x03	0	1
TF0	内部Timer0	-	硬件	0x0B	1	2
IE1	INT1引脚	Low/Falling	硬件	0x13	2	3
TF1	内部Timer1	-	硬件	0x1B	3	4
TI & RI	内部UART	-	软件	0x23	4	5
TF2	内部Timer2	-	软件	0x2B	5	6
INT2F	INT2引脚	Falling	软件	0x43	8	7
INT3F	INT3引脚	Falling	软件	0x4B	9	8
INT4F	保留					
INT5F TCPIPCore	TCPIP内核	Falling	软件	0x5B	11	10
WDIF	内部WATCHDOG	-	软件	0x63	12	11

通过改变IE（0xA8）和EIE（0xE8）对应位的值，每个中断向量都可以独立打开或关闭。IE寄存器包括一个中断总开关EA位，可以禁止（设置为0）和开启中断（设置为1）所有中断。

IE (0xA8)

7	6	5	4	3	2	1	0	Reset
EA	-	ET2	ES	ET1	EX1	ET0	EX0	0x00

图3.1 中断允许寄存器

说明：

EA - 中断总开关

EX0 - INT0中断允许

ET0 - Timer0中断允许

EX1 - INT1中断允许

ET1 - Timer1中断允许

ES – UART中断允许

ET2 - Timer2中断允许

所有这些可以产生中断的位可以通过软件设置或清除，其结果与硬件产生的中断相同。因此中断可以由软件产生或清除。唯一例外的是IE0和IE1的中断请求标志位。如果外部中断0和中断1编程设置为电平触发，IE0和IE1分别由外部引脚nINT0和nINT1控制。

IP (0xB8)

7	6	5	4	3	2	1	0	Reset
-	-	PT2	PS	PT1	PX1	PT0	PX0	0x00

图3.2 中断优先级寄存器

说明：

PX0 - INT0中断优先级控制（1是最高级）

PT0 - Timer0中断优先级控制（1是最高级）

PX1 - INT1中断优先级控制（1是最高级）

PT1 - Timer1中断优先级控制（1是最高级）

PS - UART中断优先级控制（1是最高级）

PT2 – Timer2中断优先级控制（1是最高级）

没有定义的位读出的值为‘0’或‘1’

TCON (0x88)

7	6	5	4	3	2	1	0	Reset
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0x00

图3.3 Timer0/1配置寄存器

说明：

IT0 - INT0电平触发控制，‘0’为电平触发，‘1’为边沿触发。

IT1 - INT1电平触发控制，‘0’为电平触发，‘1’为边沿触发。

IE0 - 当处理器跳转到中断服务程序时，INT0中断标志由硬件清除。

IE1 – 当处理器跳转到中断服务程序时，INT1中断标志由硬件清除。

TF0 - Timer0中断溢出标志。当处理器跳转到中断服务程序时由硬件清除。

TF1 - Timer1中断溢出标志。当处理器跳转到中断服务程序时由硬件清除。

SCON (0x98)

7	6	5	4	3	2	1	0	Reset
SM0	SM1	SM2	REN	TB8	RB8	TI	RI	0x00

图3.4 UART置寄存器

说明：

RI - UART接收中断标志。

TI - UART发送中断标志。

EIE (0xE8)

7	6	5	4	3	2	1	0	Reset
-	-	-	EWDI	EINT5	EINT4	EINT3	EINT2	0x00

图3.5 扩展的中断允许寄存器

说明：

EINT2 - INT2中断允许位。

EINT3 - INT3中断允许位。

EINT4 - 如果使用EIE，该位为‘0’。

EINT5 - TCP/IP内核中断允许位。

EWDI - 看门狗定时器中断允许位。

EIP (0xF8)

7	6	5	4	3	2	1	0	Reset
-	-	-	PWDI	PINT5	PINT4	PINT3	PINT2	0x00

图3.6 扩展的中断优先级寄存器

说明：

PINT2 - INT2中断优先级控制（1是最高级）。

PINT3 - INT3中断优先级控制（1是最高级）。

PINT4 – 如果使用EIP，该位为‘0’。

PINT5 - TCP/IP内核中断优先级控制（1是最高级）。

PWDI - 看门狗定时器中断优先级控制（1是最高级）。

EIF (0x91)

7	6	5	4	3	2	1	0	Reset
-	-	-	-	INT5F	INT4F	INT3F	INT2F	0x00

图3.7 扩展的中断标志寄存器

说明：

INT2F - INT2中断标志，必须由软件清除。

INT3F – INT3 INT3中断标志，必须由软件清除。

INT4F – 如果使用EIF，该位为‘0’。

INT5F - TCPIP内核中断标志，必须由软件清除。

WDCON (0xD8)

7	6	5	4	3	2	1	0	Reset
-	-	-	-	WDIF	WTRF	EWT	RWT	0x00

图3.8 看门狗控制寄存器

说明：

WDIF - 看门狗中断标志。WDIF与看门狗中断允许位（EIE.4）关联，而EWT提供看门狗定时器事件已经产生、或应该采取什么行动等信息。在退出中断服务程序前应该由软件清除这个标志位，否则将产生另外一次中断。通过软件设置WDIF，将产生看门狗中断。因此使能WDIF可以产生看门狗中断，定时访问寄存器可以修改该位。

4 I/O端口

IO口引脚功能描述如下表所示。

表4.1 I/O端口引脚描述

引脚	激活	类型	上拉/下拉	描述
P0[7:0]	-	IO	-	Port0输入/输出
P1[7:0]	-	IO	-	Port1输入/输出
P2[7:0]	-	IO	-	Port2输入/输出
P3[7:0]	-	IO	-	Port3输入/输出

P0 (0x80)

7	6	5	4	3	2	1	0	Reset
P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	0xFF

图4.1 Port0寄存器

P1 (0x90)

7	6	5	4	3	2	1	0	Reset
P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	0xFF

图4.2 Port1寄存器

P2 (0xA0)

7	6	5	4	3	2	1	0	Reset
P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	0xFF

图4.3 Port2寄存器

P3 (0xB0)

7	6	5	4	3	2	1	0	Reset
P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0	0xFF

图4.4 Port3寄存器

对I/O端口的读写操作都是通过对应的特殊功能寄存器（特殊功能寄存器的P0（0x80），P1（0x90），P2（0xA0），P3（0xB0））来实现的。有些读端口指令从寄存器读取数据，而有些从端口引脚读取数据。读-修改-写指令是直接操作寄存器，如下表所示。

表4.2 读-修改-写指令

指令	功能描述
ANL	Logic AND
ORL	Logic OR
XRL	Logic exclusive OR
JBC	Jump if bit is set and cleared
CPL	Complement bit
INC, DEC	Increment, decrement byte
DJNZ	Decrement and jump if not zero
MOV Px.y, C	Move carry bit to bit y of port x
CLR Px.y	Clear bit y of port x
SETB Px.y	Set bit y of port x

所有其它的读IO端口的指令都是读取端口的引脚。所有的端口引脚都可以作为GPIO使用。W7100A的GPIO输出可以根据特殊寄存器Px_PD或Px_PU的值下拉到0V或上拉到3.3V。

P0_PD(0xE3)：GPIO0下拉寄存器，设为‘1’使对应位的引脚下拉

P0_PD (0xE3)

7	6	5	4	3	2	1	0	Reset
Port0[7]	Port0[6]	Port0[5]	Port0[4]	Port0[3]	Port0[2]	Port0[1]	Port0[0]	0x00

图4.5 Port0下拉寄存器

P1_PD(0xE4)：GPIO1下拉寄存器，设为‘1’使对应位的引脚下拉

P1_PD (0xE4)

7	6	5	4	3	2	1	0	Reset
Port1[7]	Port1[6]	Port1[5]	Port1[4]	Port1[3]	Port1[2]	Port1[1]	Port1[0]	0x00

图4.6 Port1下拉寄存器

P2_PD(0xE5)：GPIO2下拉寄存器，设为‘1’使对应位的引脚下拉

P2_PD (0xE5)

7	6	5	4	3	2	1	0	Reset
Port2[7]	Port2[6]	Port2[5]	Port2[4]	Port2[3]	Port2[2]	Port2[1]	Port2[0]	0x00

图4.7 Port2下拉寄存器

P3_PD(0xE6)：GPIO3下拉寄存器，设为‘1’使对应位的引脚下拉

P3_PD (0xE6)

7	6	5	4	3	2	1	0	Reset
Port3[7]	Port3[6]	Port3[5]	Port3[4]	Port3[3]	Port3[2]	Port3[1]	Port3[0]	0x00

图4.8 Port3下拉寄存器

P0_PU(0xEB)：GPIO0上拉寄存器，设为‘1’使对应位的引脚下拉

P0_PU (0xEB)

7	6	5	4	3	2	1	0	Reset
Port0[7]	Port0[6]	Port0[5]	Port0[4]	Port0[3]	Port0[2]	Port0[1]	Port0[0]	0x00

图4.9 Port0上拉寄存器

P1_PU(0xEC)：GPIO1上拉寄存器，设为‘1’使对应位的引脚下拉

P1_PU (0xEC)

7	6	5	4	3	2	1	0	Reset
Port1[7]	Port1[6]	Port1[5]	Port1[4]	Port1[3]	Port1[2]	Port1[1]	Port1[0]	0x00

图4.10 Port1上拉寄存器

P2_PU(0xED)：GPIO2上拉寄存器，设为‘1’使对应位的引脚下拉

P2_PU (0xED)

7	6	5	4	3	2	1	0	Reset
Port2[7]	Port2[6]	Port2[5]	Port2[4]	Port2[3]	Port2[2]	Port2[1]	Port2[0]	0x00

图4.11 Port2上拉寄存器

P3_PU(0xEE)：GPIO3上拉寄存器，设为‘1’使对应位的引脚下拉

P3_PU (0xEE)

7	6	5	4	3	2	1	0	Reset
Port3[7]	Port3[6]	Port3[5]	Port3[4]	Port3[3]	Port3[2]	Port3[1]	Port3[0]	0x00

图4.12 Port3上拉寄存器

5 定时器

W7100A有两个16位的定时器/计数器，Timer 0和Timer 1。在定时器模式，定时器寄存器以1/12CLK的周期递增。在计数器模式，定时器寄存器在相对应引脚（T0或T1）的下降沿递增。输入引脚在每一个CLK周期都将采样。

5.1 Timers 0/1

5.1.1 综述

引脚的功能描述如下表所示。所有的引脚都是单向的。这些引脚没有三态输出和内部信号。

表5.1 Timers 0/1引脚描述

引脚	激活状态	类型	Pu/Pd	描述
T0/FCS	Falling	I	-	Timer0时钟
GATE0/FOE	High	I	-	Timer0时钟输入控制
T1/FAE	Falling	I	-	Timer1时钟
GATE1/FA0	High	I	-	Timer1时钟输入控制

Timer 0和Timer 1与标准的8051的定时器完全兼容。每一个定时器包括2个8位的寄存器，TH0（0x8C）和TL0（0x8A），TH1（0x8D）和TL1（0x8B）。定时器有四种工作模式，描述如下。

表5.2 Timers 0/1模式

M1	M0	模式	功能描述
0	0	0	THx作为8位的定时器/计数器，TLx的低5位作为预分频器（1/32时钟）
0	1	1	THx和TLx合成使用，作为16位的定时器/计数器
1	0	2	TLx作为8位定时器/计数器，自动装载THx
1	1	3	TL0作为一个8位定时器/计数器，由标准的Timer0位控制。TH0作为8位定时器，由Timer1的位控制。Timer1保持计数

TMOD (0x89)

Timer1				Timer0				
7	6	5	4	3	2	1	0	Reset
GATE	CT	M1	M0	GATE	CT	M1	M0	0x00

图5.1 Timer0/1控制模式寄存器

说明：

GATE - 选通控制

1：当GATEx引脚为高电平且TRx控制位为‘1’时，Timer x允许工作。

0：当TRx置‘1’时，Timer x允许工作。

CT - 定时器/计数器选择位

1：计数器模式，Timer x的时钟源来自Tx引脚。

0：定时器模式，使用内部时钟。

M1, M0 – 模式选择

TCON (0x88)

7	6	5	4	3	2	1	0	Reset
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0x00

图5.2 Timer0/1配置寄存器

说明：

TR0 - Timer0运行控制

1：运行

0：禁止

TR1 - Timer1运行控制

1：运行

0：禁止

外部输入引脚GATE0和GATE1可通过编程设置，实现脉冲宽度测量的功能。

5.1.2 中断

与Timer0/1中断相关的位如下。通过IE寄存器可以产生一个中断，中断优先级可以通过IP寄存器进行配置。

IE (0xA8)

7	6	5	4	3	2	1	0	Reset
EA	-	ET2	ES	ET1	EX1	ET0	EX0	0x00

图5.3 中断允许寄存器

说明：

EA - 中断总开关。

ET0 - Timer0中断允许

ET1 - Timer1中断允许

IP (0xB8)

7	6	5	4	3	2	1	0	Reset
-	-	PT2	PS	PT1	PX1	PT0	PX0	0x00

图5.4 中断优先级寄存器

说明：

PT0 - 允许Timer0中断优先。

PT1 - 允许Timer1中断优先。

未使用的位读出为‘0’或‘1’。

TCON (0x88)

7	6	5	4	3	2	1	0	Reset
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0x00

图5.5 Timer0/1配置寄存器

说明：

TF0 - Timer0中断（溢出）标志。当处理器进入中断处理程序时，由硬件清除。

TF1 - Timer1中断（溢出）标志。当处理器进入中断处理程序时，由硬件清除。

所有产生中断的位都可以由软件设置或清除，与硬件的作用效果相同。也即是说，中断可以由软件产生和清除。

表5.3 Timer0/1中断

中断标志	功能	电平	复位标志	向量	自然级别
TF0	内部Timer0	-	硬件	0x0B	2
TF1	内部Timer1	-	硬件	0x1B	4

5.1.3 Timer0 – 模式0

Timer0寄存器为13位的寄存器（8位定时器，5位定标器）。当计数器的值（所有有效位）从1翻转为0时，Timer0的中断标志位TF0置‘1’。当TCON.4=1且TMOD.3=0或GATE0=1时，定时器开始计数。通过设置TMOD.3=1，外部计数输入GATE0可以控制定时器0，进行脉冲宽度的计量。13位寄存器由8位TH0和5位TL0组成。TL0的高3位忽略。如下图所示。

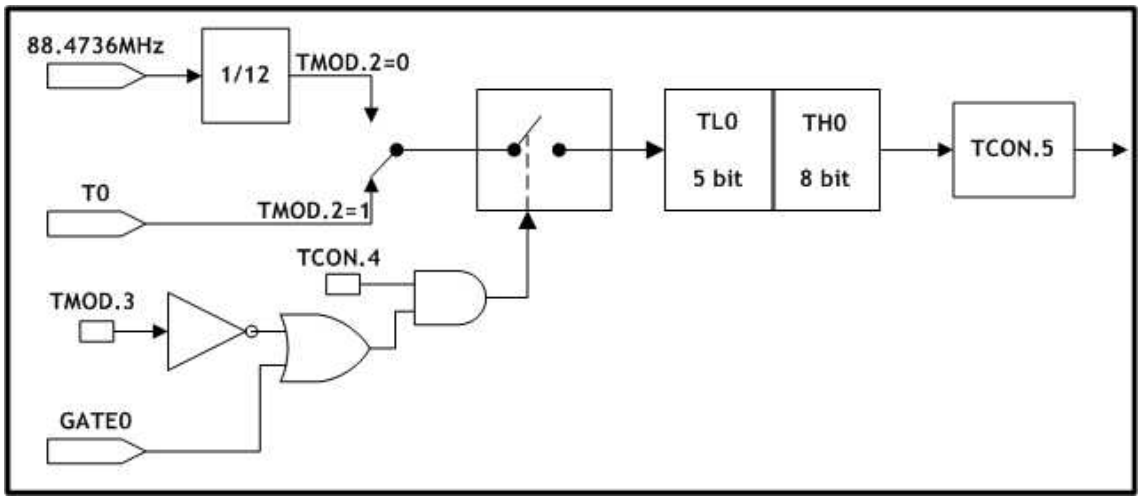


图5.6 定时器/计数器0模式0，13位定时器/计数器

5.1.4 Timer0 – 模式1

模式1与模式0相同。只是定时器的寄存器是16位。模式1如下图所示。

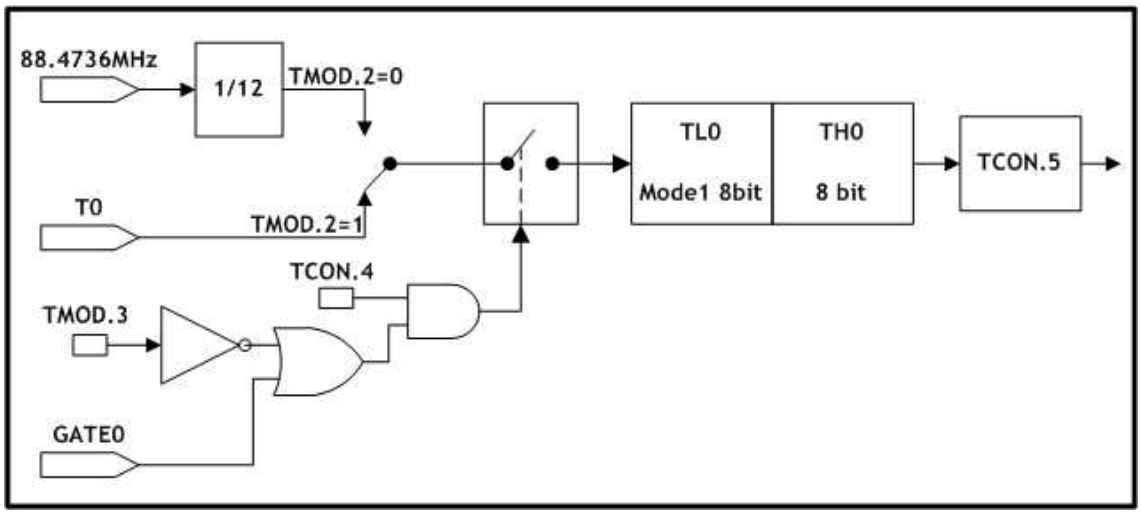


图5.7 定时器/计数器0模式1，16位定时器/计数器

5.1.5 Timer0 – 模式2

模式2的定时寄存器是8位的TL0，带自动装载功能，如下图所示。当TL0溢出时，它将TF0置‘1’，并将TH0的值装载到TL0。TH0在装载完成后保持不变。

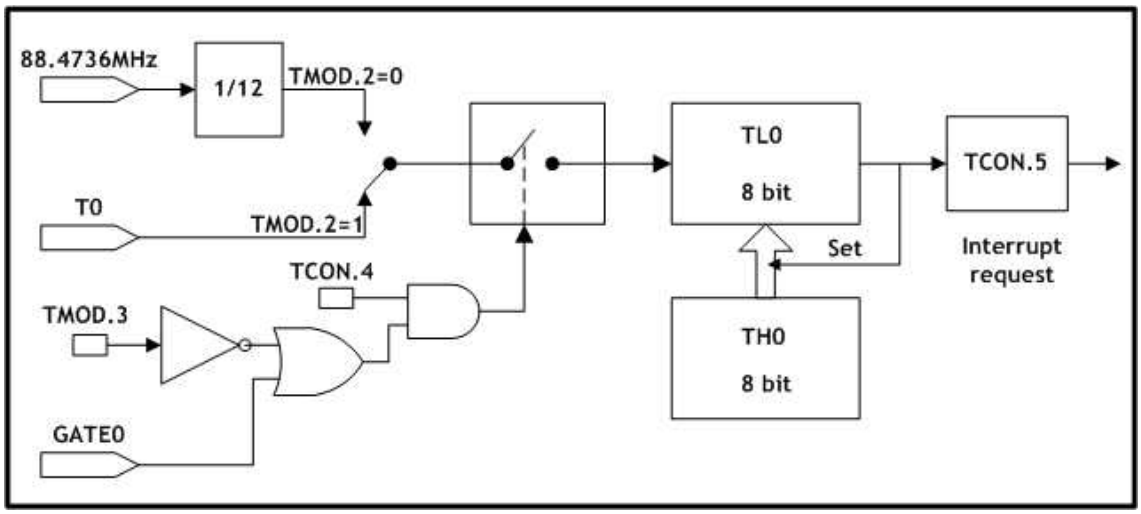


图5.8 定时器/计数器0模式2，8位定时器/计数器，带自装载功能

5.1.6 Timer0 – 模式3

在这种模式下，TL0和TH0被分为两个独立的计数器。Timer0运行在模式3下的逻辑关系如下图所示。TL0使用定时器0的控制位：C/T、GATE、TR0、GATE0和TF0。TH0锁定为定时器功能，使用定时器1的TR1和TF1，并控制Timer1的中断。模式3用于需要特别的8位定时

器/计数器的应用中。如果Timer0工作在模式3，通过切换为模式3，Timer1可开启或关闭，或作为串口通道的波特率发生器，或在某些应用中，不需要Timer1的中断。

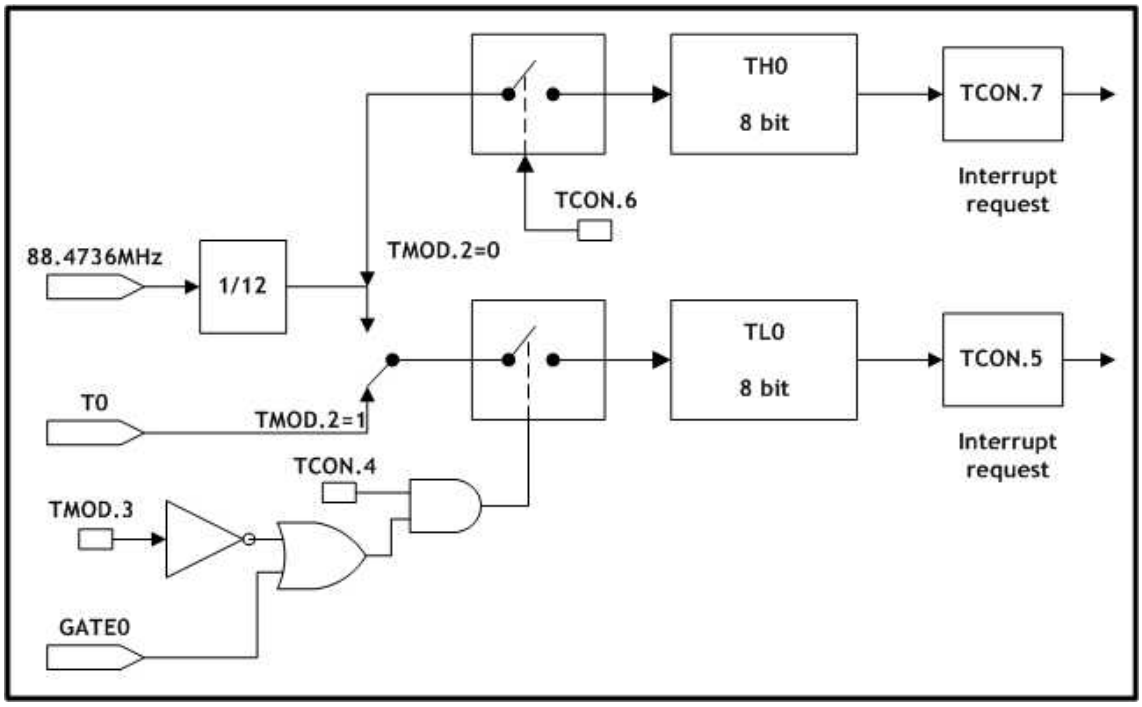


图5.9 定时器/计数器0模式3，2个8位定时器/计数器

5.1.7 Timer1 – 模式0

在这种模式下，Timer1寄存器为13位寄存器。当所有的有效位从‘1’翻转为‘0’时，Timer1中断标志TF1置‘1’。当TCON.6=1 且TMOD.6=0 或GATE1=1（设置TMOD.7=1，允许Timer1受外部输入信号GATE1的控制，该功能可用于脉冲宽度测量）时，开始对外部输入计数。13位寄存器由8位TH1和TL1的低5位组成。TL1的高3位未定义， 可以忽略。详细过程如下图所示。

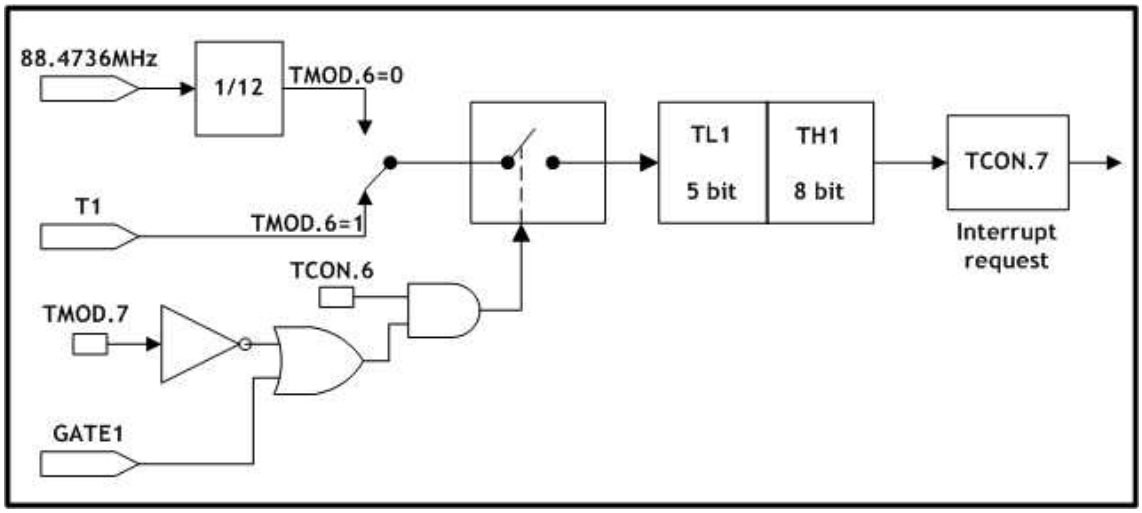


图5.10 定时器/计数器1模式0，13位定时器/计数器

5.1.8 Timer1 – 模式1

模式1与模式0相同，只是定时器是全部16位。模式1如下图所示。

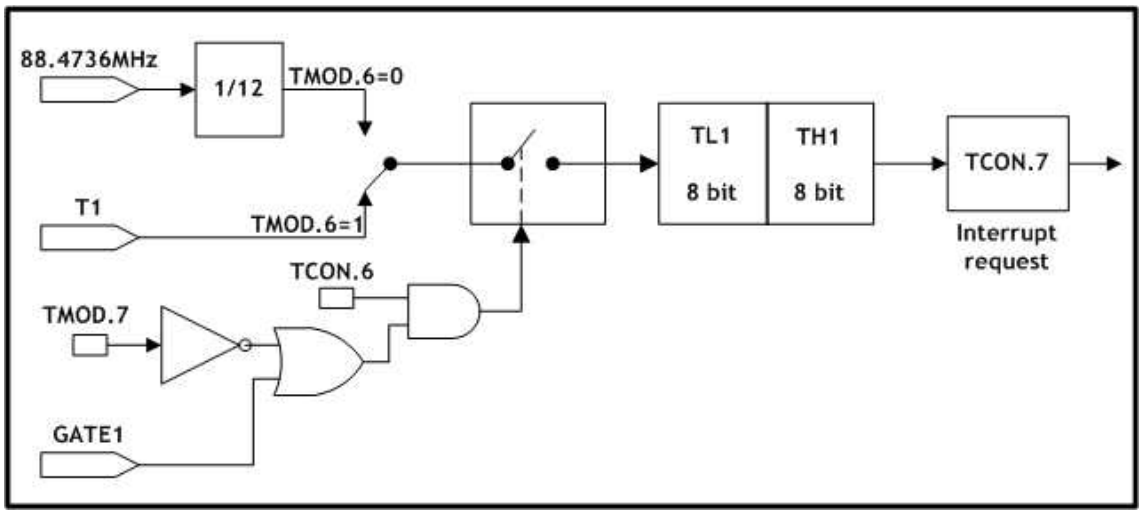


图5.11 定时器/计数器1模式1：16位定时器/计数器

5.1.9 Timer1 – 模式2

模式2配置为8位TL1定时器，带自动装载功能，如下图所示。只有TL1的溢出将置TF1标志位，但TH1的值将自动装载到TL1，装载后TH1的值保持不变。

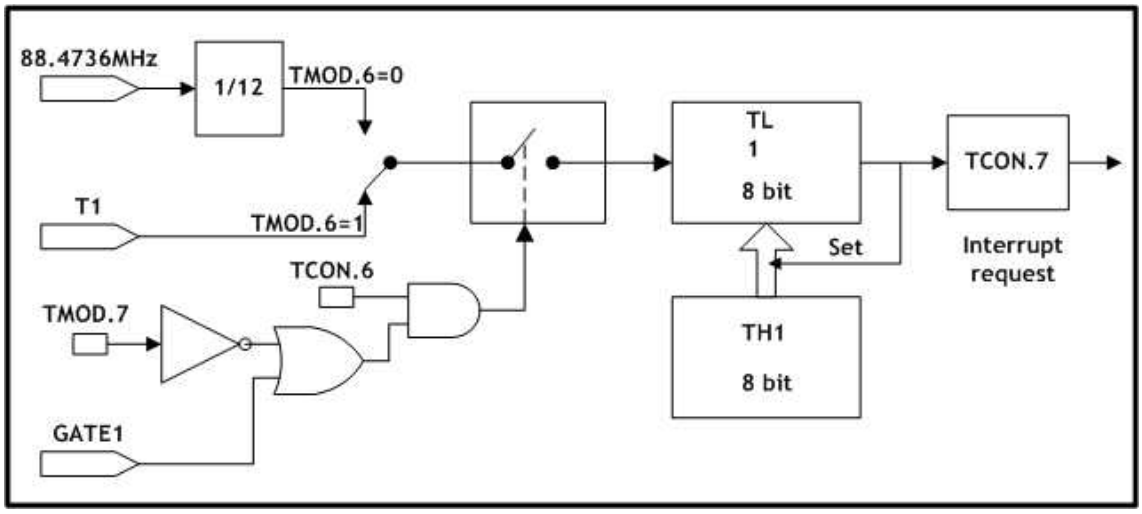


图5.12 定时器/计数器1模式2，8位定时器/计数器，带自装载功能

5.1.10 Timer1 – 模式3

Timer1的模式3保持计数功能。其影响将与TR1=0的设置相同，因为它用于Timer0的模式3。详细内容请参考“5.1.6 Timer0模式3”。

5.2 Timer2

5.2.1 综述

引脚的功能描述如下表所示。所有引脚都是单向的，这些引脚没有三态输出和内部信号。

表5.4 Timer2引脚描述

引脚	激活状态	类型	Pu/Pd	描述
T2/FA1	下降沿	I	-	Timer2 外部时钟输入
T2EX/FA2	下降沿	I	-	Timer2 俘获/重载触发

W7100A的Timer2与标准8051的Timer2完全兼容。Timer2的运行控制有5个特殊功能寄存器，TH2/TL2 (0xCD/0xCC) 计数寄存器，RLDH/RLDL (0xCB/0xCA) 捕获寄存器和T2CON (0xC8) 控制寄存器。Timer2有3种工作模式，由T2CON的位进行选择，如下表所示。

表5.5 Timer2模式

RCLK,TCLK	CPRL2	TR2	功能描述
0	0	1	16位自动装载模式。当Timer2溢出时TF2位置'1'。TH2和TL2寄存器自动从RLDH和RLDL装载16的值
0	1	1	16位捕获模式。当Timer2溢出时TF2位置'1'。当EXEN2=1且在T2EX引脚信号为下降沿时，TH2和TL2的值保存到RLDH和RLDL中
1	X	1	作为UART接口的波特率发生器
X	X	0	Timer2关闭

T2CON (0xC8)

7	6	5	4	3	2	1	0	Reset
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	CT2	CPRL2	0x00

图5.13 Timer2配置寄存器

说明：

EXF2 – 当EXEN2=1时，表示T2EX引脚有下降沿信号输入，必须由软件清零。

RCLK - UART接收器时钟允许

0：Timer1的溢出脉冲作为UART的接收器时钟。

1：Timer2的溢出脉冲作为UART的接收器时钟。

TCLK - UART发送器时钟允许

0：Timer1的溢出脉冲作为UART的发送器时钟。

1： Timer2的溢出脉冲作为UART的发送器时钟。

EXEN2 - T2EX引脚功能允许位。

0：忽略T2EX事件

1：T2EX引脚的下降沿允许捕获或重装载。

TR2 - 启动/停止Timer2

0：停止

1：启动

CT2 - 定时器/计数器选择

0：内部时钟定时器。

1：外部事件计数器，时钟源由T2引脚输入。

CPRL2 - 捕获/重装载选择

0：当Timer2溢出，或在EXEX2=1且T2EX引脚出现下降沿输入时，产生自动重装载。当RCLK或TCLK置位，忽略该位，并在Timer2溢出时自动重装载。

1：当EXEN2=1时，T2EX引脚出现下降沿输入时激活捕获功能。

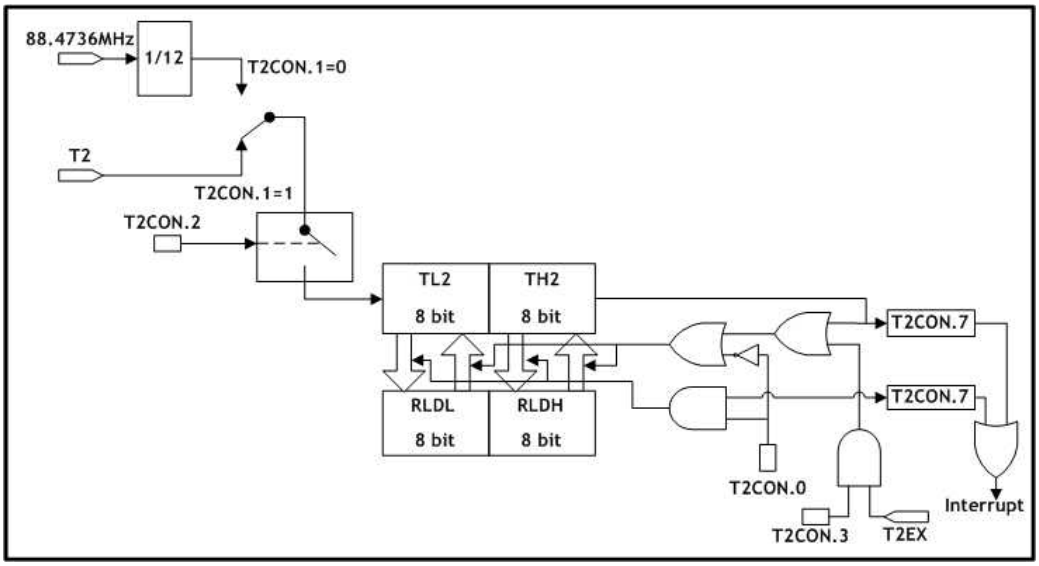


图5.14 Timer2带自动装载的16位定时器/计数器

5.2.2 中断

Timer2的中断如下。中断可以由IE寄存器的对应位触发，由IP寄存器的对应位配置其优先级。

IE (0xA8)

7	6	5	4	3	2	1	0	Reset
EA	-	ET2	ES	ET1	EX1	ET0	EX0	0x00

图5.15 Timer2中断允许寄存器

说明：

EA - 中断允许总开关

ET2 - Timer2中断允许。

IP (0xB8)

7	6	5	4	3	2	1	0	Reset
-	-	PT2	PS	PT1	PX1	PT0	PX0	0x00

图5.16 Timer2中断优先级寄存器

说明：

PT2 - Timer2优先级控制（1为高优先级）

未使用的位读出为‘0’或‘1’

T2CON (0xC8)

7	6	5	4	3	2	1	0	Reset
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	CT2	CPRL2	0x00

图5.17 Timer2配置寄存器

说明：

TF2 – Timer2中断（溢出）标志。必须由软件清除。而RCLK或TCLK置‘1’不影响该标志。

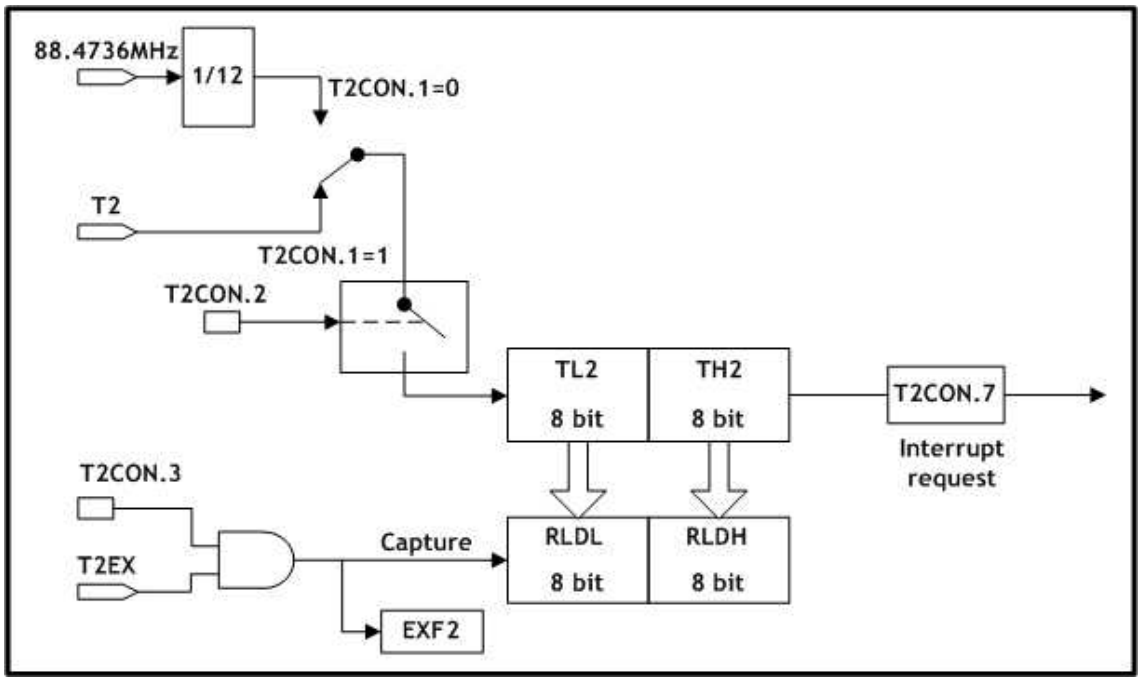


图5.18 Timer/Counter2 - 16位带捕获功能定时器/计数器

所有产生中断的位可以由软件设置或清除。与硬件产生的结果相同。因此中断可以由软件产生或清除。

表5.6 Timer2中断

Interrupt Flag	Function	Active Level/Edge	Flag Resets	Vector	Natural Priority
TF2	Internal, Timer2	-	Software	0x2B	6

当EXEN2位置，‘ 1 ’时，中断由T2EX引脚的下降沿触发产生。使用0x2B中断矢量，EXF2由这个中断设置，但TF2标志保持不变。

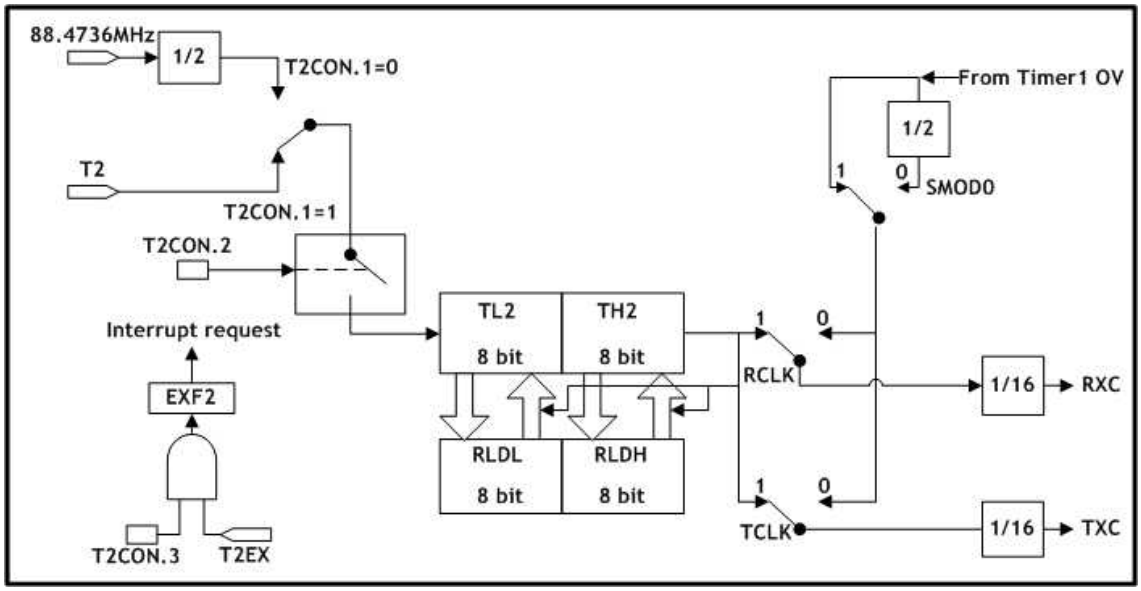


图5.19 Timer2作波特率发生器模式

6 UART

W7100A的UART工作在全双工状态，允许同时进行接收和发送操作。因为W7100A是双数据缓冲区，接收器在第1个字节没有被读取时，仍然可以接收数据。在读操作时，从接收缓冲区SBUF读取数据。另一方面，在发送过程中，SBUF将数据装载到发送缓冲区。UART有4种不同的工作模式，一种是同步模式，3种异步模式。模式2和模式3的一些特殊的特性主要用于多机通信。通过设置SCON寄存器的SM2位可以使用该特性。主处理器首先发送地址字节以确定目标从机。地址字节与数据字节在第9位进行区分，第9位为‘1’则表示是地址字节，为‘0’是数据字节。当SM2=1，数据字节不会使从机产生中断，而地址字节则将使所有从机产生中断。选中的从机将SM2清零，准备接收即将到来的数据。而没有被选中的从机SM2仍然为‘1’，忽略所有接收的数据。

UART的引脚功能描述如下：

表6.1 UART引脚描述

Pin	Active	Type	Pu/Pd	Description
RXD	-	I	Pu	串行接收器的输入/输出
TXD	-	O	-	串行输出

W7100A与标准的8051的UART是全兼容的。与UART有关的寄存器是：SBUF（0x99）、SCON（0x98）、PCON（0x87）、IE（0xA8）和IP（0xB8）。UART数据缓冲器（SBUF）由发送寄存器和接收寄存器组成。当数据写入到SBUF发送寄存器时，启动发送过程。同样，在接收过程中，数据从SBUF接收寄存器读出。

SBUF (0x99)

7	6	5	4	3	2	1	0	Reset
SB7	SB6	SB5	SB4	SB3	SB2	SB1	SB0	0x00

图6.1 UART数据缓冲寄存器

SCON (0x98)

7	6	5	4	3	2	1	0	Reset
SM0	SM1	SM2	REN	TB8	RB8	TI	RI	0x00

图6.2 UART配置寄存器

说明：

SM2 - 允许多处理器通信。

SM1 - 设置波特率。

SM0 - 设置波特率。

REN – 1：允许串行接收；0：禁止串行接收

TB8 - 在模式2和模式3，发送数据的第9位。根据MCU的运行情况启用这一位（奇偶校验，多机通信等）。

RB8 - 在模式2和模式3， 接收数据的第9位。在模式1，如果SM2=0，RB8是停止位，在模式0，这位没有使用。

UART模式如下表所示：

表6.2 UART模式

SM0	SM1	模式	描述	波特率
0	0	0	Shift register	f_{osc}/12
0	1	1	8-bit UART	Variable
1	0	2	9-bit UART	f_{osc}/32 or /64
1	1	3	9-bit UART	Variable

波特率如下表所示：

表6.3 UART波特率

模式	波特率
Mode0	f_{osc}/12
Mode1,3	Time1 overflow rate or Timer2 overflow rate
Mode2	SMOD0 = 0 f_{osc}/64
	SMOD0 = 1 f_{osc}/32

SMOD0位于PCON寄存器

PCON (0x87)

7	6	5	4	3	2	1	0	Reset
SMOD0	SMOD1	-	PWE	-	0	0	0	0x00

图6.3在电源配置寄存器中的UART相关位

说明：

SMOD0 - 用于UART波特率的位

未使用的位读出为‘0’或‘1’，位2~0必须写入0

6.1 中断

与UART中断有关的位如下。中断可以由IE寄存器触发，中断优先级可以由IP寄存器配置。

IE (0xA8)

7	6	5	4	3	2	1	0	Reset
EA	-	ET2	ES	ET1	EX1	ET0	EX0	0x00

图6.4 UART中断允许位

说明：ES - RI和TI中断允许标志。

IP (0xB8)

7	6	5	4	3	2	1	0	Reset
-	-	PT2	PS	PT1	PX1	PT0	PX0	0x00

图6.5 UART中断优先级

说明：

PS- UART设置为高优先级（PS=1时）

未使用的位读出为‘0’或‘1’

SCON (0x98)

7	6	5	4	3	2	1	0	Reset
SM0	SM1	SM2	REN	TB08	RB08	TI	RI	0x00

图6.6 UART配置寄存器

说明：

TI – 发送中断标志。完成一次发送后由硬件置‘1’，必须由软件清除。

RI – 接收中断标志，接收一个串行数据后由硬件置‘1’，必须由软件清除。

所有可以产生中断的位都可以通过软件设置或清除，与硬件产生的效果相同。也即是说，中断可以由软件产生或清除

表6.4 UART中断

中断标志	功能	激活电平	标志复位	向量	中断级别
TI & RI	内部UART	-	软件	0x23	5

6.2 模式0，同步通信模式

同步通信时，TXD输出同步时钟，波特率固定为1/12时钟频率（CLK）。传输的8位数据低位在前。通过设置RI0=0和REN0=1，即可启动接收。

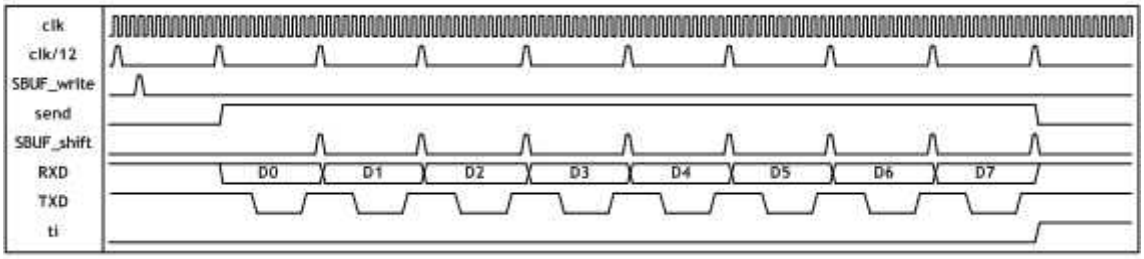


图6.7 模式0同步传输时序图(clk = 88.4736 MHz)

6.3 模式1，8位UART模式，波特率可改变，Timer1或Timer2作为时钟源

RXD引脚作为串行通信的输入端，TXD引脚作为串行通信的输出端。10位传输的数据顺序是：1个起始位（0）、8位数据位（低位在前）、一个停止位（1）。在接收过程中，通过起始位同步传输过程，下一步即可以通过SBUF读取到8位数据，停止位用于触发特殊功能寄存器SCON（0x98）的RB08位，根据Timer1或Timer2的模式可以改变通信波特率。要使用Timer2

作为时钟源，需要设置T2CON（0xC8）寄存器的TCLK和RCLK位。

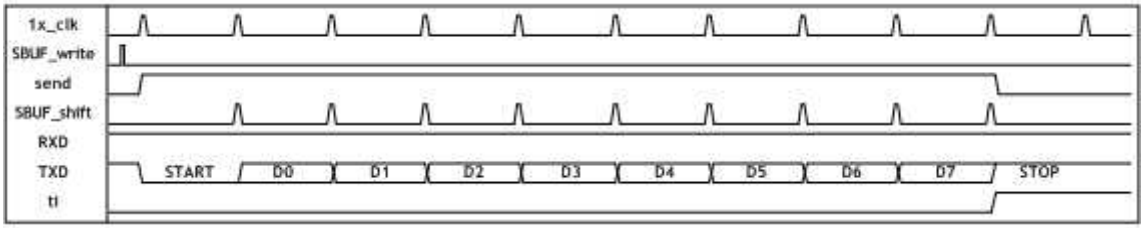


图6.8 UART模式1的传输时序图

6.4 模式2，9位UART，波特率固定

模式2与模式1相同。但波特率固定为1/32或1/64时钟频率，传输的数据为11位：一个起始位（0），8位数据位（低位在前），一个可编程的第9位和一个停止位（1）。第9位可用于UART数据的奇偶校验。在发送过程中，SCON寄存器的TB08位是作为数据的第9位输出的。在接收过程中，接收的第9位将保存在SCON的RB08位。

图6.9 UART模式2的传输时序图

6.5 模式3，9位UART，波特率可改变，Timer1或Timer2作时钟源

模式3的波特率是可变的，这是与模式2的唯一区别。当REN0=1时，允许数据接收。波特率根据Timer1或Timer2是可变化的。为了使用Timer2时钟，需要设置T2CON（0xC8）寄存器的TCLK和RCLK位为‘1’。

图6.10 UART模式3的传输时序图

6.6 波特率设置

表6.5 波特率设置示例

Baud Rate(bps)	Timer 1 / Mode2		Timer 2
	TH1(0x8D)		RLDH(0xCB), RLDL(0xCA)
	SMOD = '0'	SMOD = '1'	
2400	160(0xA0)	64(0x40)	64384(0XFB80)
4800	208(0xD0)	160(0xA0)	64960(0xFDC0)
9600	232(0xE8)	208(0xD0)	65248(0xFEE0)
14400	240(0xF0)	224(0xE0)	65344(0XFF40)
19200	244(0xF4)	232(0xE8)	65392(0XFF70)
28800	248(0xF8)	240(0xF0)	65440(0xFFA0)
38400	250(0xFA)	244(0xF4)	65464(0XFFB8)
57600	252(0xFC)	248(0xF8)	65488(0xFFD0)
115200	254(0xFE)	252(0xFC)	65512(0xFFE8)
230400	255(0xFF)	254(0xFE)	65524(0xFFF4)

说明：

波特率计算公式：

使用Timer1：波特率 = (2^{SMOD} / 32) * (Clock Frequency / 12(256 – TH1))

使用Timer2：波特率 = Clock Frequency / (32 * (65536 – (RLDH, RLDL)))

7 看门狗定时器

7.1 综述

看门狗定时器由系统时钟经过一系列的分频器提供时钟信号，如下图所示。分频器输出可通过选择它决定定时器超时的间隔。当看门狗定时器产生超时的时候，相应的中断标志将置位（如果允许的话），然后对系统复位。当中断允许控制位和中断总开关都开启时，中断标志将触发中断。复位和中断是完全不同的两个功能，根据应用系统的要求，可分别得到响应、或单独得到响应、或忽略之。

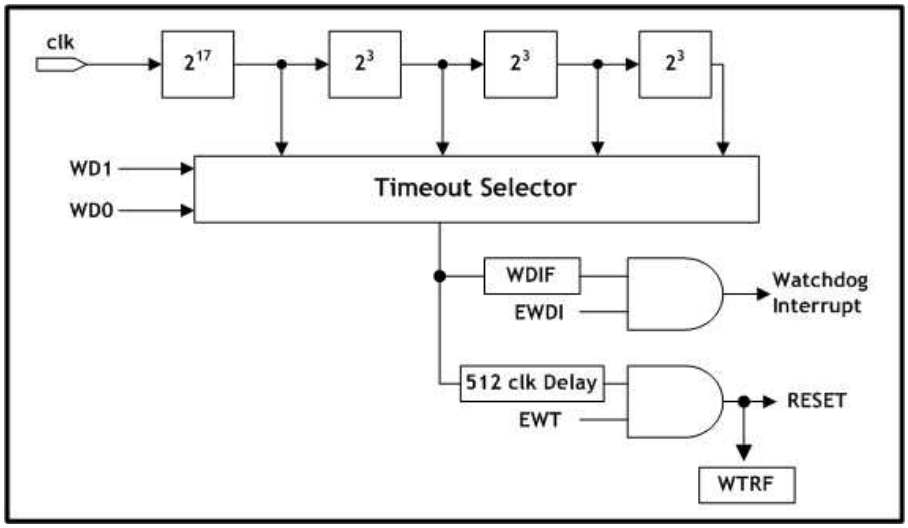


图7.1 看门狗定时器结构

7.2 中断

与看门狗中断有关的'位'如下图所示。中断可以由IE（0xA8）和EIE（0xE8）寄存器打开或关闭。由EIP寄存器设置其优先级。IE中的中断总开关EA可打开或关闭所有中断。

IE (0xA8)

7	6	5	4	3	2	1	0	Reset
EA	-	ET2	ES	ET1	EX1	ET0	EX0	0x00

图7.2 中断允许寄存器

EIE (0xE8)

7	6	5	4	3	2	1	0	Reset
-	-	-	EWDI	EINT5	EINT4	EINT3	EINT2	0x00

图7.3 扩展的中断允许寄存器

说明：

EA - 中断总开关

EWDI - 允许看门狗定时器中断

EIP (0xF8)

7	6	5	4	3	2	1	0	Reset
-	-	-	PWDI	PINT5	PINT4	PINT3	PINT2	0x00

图7.4 扩展的中断优先寄存器

说明：

PWDI - 看门狗中断优先级控制（1为高优先级）。

未使用的位读出为‘0’或‘1’

WDCON (0xD8)

7	6	5	4	3	2	1	0	Reset
-	-	-	-	WDIF	WTRF	EWT	RWT	0x00

图7.5 看门狗定时器控制寄存器

说明：

WDIF - 看门狗中断标志。WDIF与看门狗中断允许位（EIE.4）和EWT位有关，它指示看门狗定时器事件是否发生，是否产生动作等。在退出中断处理程序前必须由软件清除WDIF，否则将产生另外一次中断。如果允许，设置WDIF位将产生看门狗中断。对该位的修改必须遵循**顺序访问寄存器**规则。请参阅7.8节“顺序访问”处理。

所有产生中断的位都可以通过软件置位或清零，与硬件产生中断的效果相同。也即是说，中断可以由软件产生和清除

表7.1 看门狗中断

中断标志	功能	电平/边沿激活	标志复位	向量	中断优先级
WDIF	内部看门狗	-	软件	0x63	11

7.3 看门狗定时器复位

看门狗定时器复位的操作如下：一旦设定溢出的时间间隔，系统首先通过RWT重新启动看门狗。然后，通过启动看门狗定时器复位（WDCON.1）位，启动复位模式。在定时器到达用户设定的定时值之前，软件可以置复位看门狗定时器（WDCON.0）位为‘1’。如果在定时器时间溢出发生之前设置RWT，定时器将重新启动；如果在RWT置位之前发生了定时器溢出，看门狗将对MCU复位。软件对RWT设置以后，硬件将自动清除RWT。当产生一次复位。看门狗定时器的复位标志（WDCON.2）将自动设置，以指示当前复位的类型。必须由软件清除该位。

7.4 简单定时器

看门狗定时器是一个独立运行的定时器。在定时器模式下启动复位（EWT=0）和禁止中断（EWDI=0），定时器开始向WD[1:0]预先设置的时间计数，并将允许看门狗中断标志。对RWT位置‘1’，定时器将工作在时间溢出检测模式。WDIF位可以由软件清‘0’或置‘1’。看门狗中断可以用于需要长时间定时的应用系统中，中断由看门狗中断允许使能位（EIE.4）开启。那么当产生时间溢出，看门狗定时器将设置WDIF位（WDCON.3）为‘1’，如果中断总开关开启，那么将产生中断。注意，在一个可能的看门狗复位之前，WDIF将在512个时钟之后置位。看门狗中断标志指示中断的来源，必须由软件清除。如果看门狗中断应用得当，看门狗复位将使得中断程序可以监控任何系统错误。

7.5 系统监控

当使用看门狗定时器作为系统监控时，要使用看门狗复位功能。如果使用中断功能，就没有达到看门狗的目的。例如，系统在看门狗中断之前正在运行错误代码，中断会使系统暂时处理中断服务程序。重启看门狗并由RETI或RET退出后，将返回到中断前的位置继续运行。如果使用看门狗复位功能，处理器将整个程序从新启动，使系统回到正常的状态。

7.6 与看门狗相关的寄存器

看门狗定时器在运行期间与多个特殊功能寄存器的位相关。这些位可用于复位源、中断源、软件检测定时器，也可以是这三者的组合。复位和中断都有状态标志。看门狗还有一个位是重启定时器。下表是对这些位的详细介绍。

表7.2 与看门狗定时器相关的寄存器位的汇总

位名	寄存器	位所在位置	描述
EWDI	EIE	EIE.4	允许看门狗定时器中断
PWDI	EIP	EIP.4	看门狗定时器中断的级别
WD[1:0]	CKCON	CKCON.7-6	看门狗时间间隔
RWT	WDCON	WDCON.0	复位看门狗定时器
EWT		WDCON.1	允许看门狗定时器复位
WTRF		WDCON.2	看门狗定时器复位标志
WDIF		WDCON.3	看门狗中断标志

在看门狗时间溢出复位期间不能禁止看门狗定时器，但它重新启动定时器。通常来讲，软件可以设置看门狗为期望的状态。控制看门狗‘位’的操作在下面章节叙述。

7.7 看门狗控制

下面将介绍看门狗的控制位。请注意，对该寄存器的访问必须按照定**顺访问寄存器**的时序进行。

WDCON (0xD8)

7	6	5	4	3	2	1	0	Reset
-	-	-	-	WDIF	WTRF	EWT	RWT	0x00

图7.6 看门狗定时器控制寄存器

说明：

WTRF - 看门狗定时器复位标志。当该标志由硬件置位时，标志着已经产生了看门狗定时器复位。然而如果由软件对该标志置位，不会触发看门狗定时器复位。在复位期间，该标志会被清除，否则需要软件清除。如果EWT被清除，看门狗定时器对该位不产生影响。

EWT – 允许看门狗定时器复位。该位控制看门狗定时器对微处理器的复位，而对看门狗中断没有影响。必须使用顺序访问对该位进行修改。

0：看门狗定时器溢出不对微处理器复位。

1：看门狗定时器溢出对微处理器复位。

RWT – 对看门狗定时器复位。对RWT置'1'将对看门狗定时器复位。在看门狗定时器溢出之前，必须按照顺序访问的过程对该位进行置位操作，否则将产生复位或中断

未使用的位读出为1或0。

表7.3总结了控制看门狗的'位'及其功能。

表7.3 看门狗寄存器的位及其作用

EWT	EWDI	WDIF	结果
X	X	0	没有看门狗定时器事件产生。
0	0	1	看门狗定时器超时已经溢出，没有中断产生。
0	1	1	看门狗定时器中断已经产生。
1	0	1	看门狗定时器已经溢出，还没有中断产生。如果RWT没有触发，看门狗定时器将在512个时钟周期（CLK引脚）后对系统复位。.
1	1	1	看门狗定时器中断已经产生。如果RWT没有使用顺序访问进行设置，看门狗定时器将在512个时钟周期（CLK引脚）后对系统复位。

7.7.1 时钟控制

如下表所示，看门狗定时器溢出的时间选择使用‘位’WD[1:0]。

CKCON (0x8E)

7	6	5	4	3	2	1	0	Reset
WD1	WD0	-	-	-	MD2	MD1	MD0	0x03

图7.7 时钟控制 - 看门狗位

时钟控制寄存器CKCON（0x8E）包含WD[1:0]位，用于选择看门狗定时器溢出时间。看门狗的时钟直接来源于CLK引脚输入。看门狗有四种时间溢出选择（基于输入的CLK时钟），如表7.4所示。这是一个预选的时钟数。因此实际的时钟溢出时间与CLK频率是相关的。

*W7100A时钟频率 = 88.4736MHz

表7.4 看门狗时间间隔

WD[1:0]	Watchdog Interval	Number of Clocks
00	2^{17}	131072
01	2^{20}	1048576
10	2^{23}	8388608
11	2^{26}	67108864

上表所示的时间间隔是产生中断事件。如果允许产生复位，那么将在512个时钟之后产生复位，而不管中断的出现。因此，实际看门狗溢出时间就是选择的看门狗定时时钟周期再加上512个时钟周期（CLK引脚信号）

7.8 顺序访问寄存器

顺序访问寄存器内部建立了一套机制可以防止意外写入。TA位于特殊功能寄存器的0xC7地址。为了正确地写该寄存器，必须按照下面的顺序操作。

```
MOV TA, #0xAA

MOV TA, #0x55

;任何直接寻址指令对顺序访问寄存器操作
```

用户设置WDCON时必须遵循访问顺序。

表7.5 顺序访问寄存器

寄存器名称	描述
WDCON(0xD8)	看门狗配置

8 TCPIP内核

8.1 存储器映射

TCP/IP内核由通用寄存器、SOCKET寄存器、TX存储器和RX存储器组成，如下图所示。

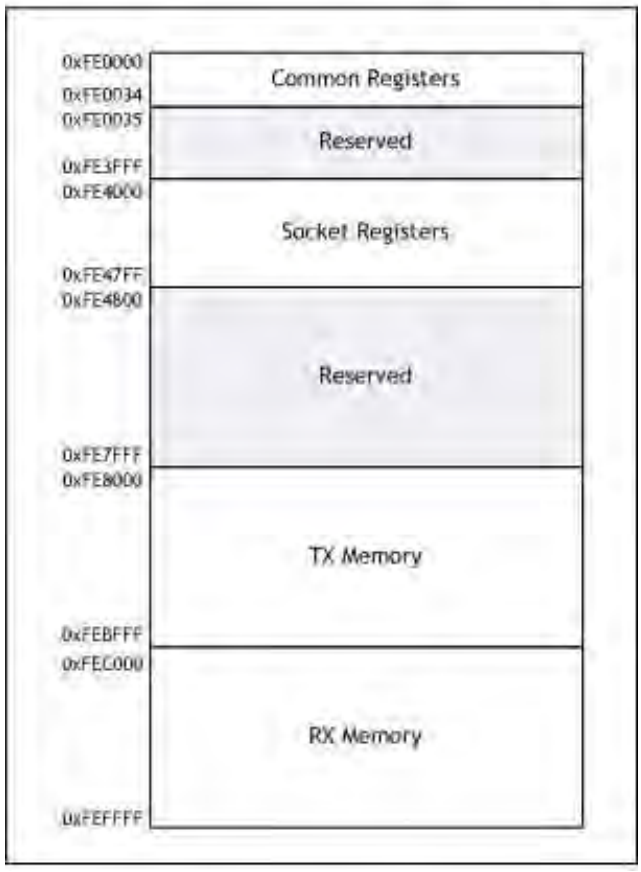


图8.1 TCPIP内核存储器映射

8.2 寄存器列表

8.2.1 通用寄存器

地址偏移量	符号	描述
0xFE0000	MR	模式寄存器
0xFE0001	GAR0	网关IP地址寄存器GAR
0xFE0002	GAR1	
0xFE0003	GAR2	
0xFE0004	GAR3	

0xFE0005	SUBR0	子网掩码寄存器SUBR
0xFE0006	SUBR1	
0xFE0007	SUBR2	
0xFE0008	SUBR3	
0xFE0009	SHAR0	本机物理地址寄存器SHAR
0xFE000A	SHAR1	
0xFE000B	SHAR2	
0xFE000C	SHAR3	
0xFE000D	SHAR4	
0xFE000E	SHAR5	
0xFE000F	SIPR0	本机IP地址寄存器SIPR
0xFE0010	SIPR1	
0xFE0011	SIPR2	
0xFE0012	SIPR3	
0xFE0013		保留
0xFE0014		
0xFE0015	IR	中断寄存器
0xFE0016	IMR	中断屏蔽寄存器
0xFE0017	RTR0	重发超时时间寄存器RTR
0xFE0018	RTR1	
0xFE0019	RCR	重发次数计数寄存器RCR
0xFE001A		保留
0xFE001B		
0xFE001C	PATR0	认证方法寄存器PART
0xFE001D	PATR1	
0xFE001E	PPPALGO	PPPoE认证算法寄存器
0xFE001F	VERSIONR	W7100A版本号寄存器
0xFE0020 ~ 0xFE0027		保留
0xFE0028	PTIMER	PPP LCP请求定时寄存器
0xFE0029	PMAGIC	PPP LCP魔术寄存器
0xFE002A ~ 0xFE002F		保留

0xFE0030	INTLEVEL0	中断输出的低电平定时寄存器INTLEVEL
0xFE0031	INTLEVEL1	
0xFE0032		Reserved
0xFE0033		
0xFE0034	IR2	SOCKET中断寄存器

8.2.2 SOCKET（端口）寄存器

地址偏移	符号	描述
0xFE4000	S0_MR	SOCKET 0模式寄存器
0xFE4001	S0_CR	SOCKET 0命令寄存器
0xFE4002	S0_IR	SOCKET 0中断寄存器
0xFE4003	S0_SR	SOCKET 0端口状态寄存器
0xFE4004	S0_PORT0	SOCKET0端口号寄存器
0xFE4005	S0_PORT1	
0xFE4006	S0_DHAR0	SOCKET 0目的硬件物理地址寄存器
0xFE4007	S0_DHAR1	
0xFE4008	S0_DHAR2	
0xFE4009	S0_DHAR3	
0xFE400A	S0_DHAR4	
0xFE400B	S0_DHAR5	
0xFE400C	S0_DIPR0	SOCKET 0目的IP地址寄存器
0xFE400D	S0_DIPR1	
0xFE400E	S0_DIPR2	
0xFE400F	S0_DIPR3	
0xFE4010	S0_DPORT0	SOCKET 0目的端口号寄存器
0xFE4011	S0_DPORT1	
0xFE4012	S0_MSSR0	SOCKET 0最大分片长度寄存器
0xFE4013	S0_MSSR1	
0xFE4014	S0_PROTO	在IPRAW模式，SOCKET 0 IP头字段的协议寄存器
0xFE4015	S0_TOS	SOCKET 0的IP服务类型（TOS）寄存器
0xFE4016	S0_TTL	SOCKET 0的IP生存时间（TTL）寄存器

0xFE4017 ~ 0xFE401D		保留
0xFE401E	S0_RXMEM_SIZE	SOCKET 0接收数据存储器大小寄存器
0xFE401F	S0_TXMEM_SIZE	SOCKET 0发送数据存储器大小寄存器
0xFE4020	S0_TX_FSR0	S0_TX_FSR (SOCKET 0发送数据存储器剩余的空间大小寄存器)
0xFE4021	S0_TX_FSR1	
0xFE4022	S0_TX_RD0	S0_TX_RD0 (SOCKET 0发送数据存储器读指针寄存器)
0xFE4023	S0_TX_RD1	
0xFE4024	S0_TX_WR0	S0_TX_WR (SOCKET 0发送数据存储器写指针寄存器)
0xFE4025	S0_TX_WR1	
0xFE4026	S0_RX_RSR0	S0_RX_RSR (SOCKET 0接收数据字节长度寄存器)
0xFE4027	S0_RX_RSR1	
0xFE4028	S0_RX_RD	S0_RX_RD (SOCKET 0接收数据存储器读指针寄存器)
0xFE4029	S0_RX_RD1	
0xFE402A	S0_RX_WR	S0_RX_WR (SOCKET 0接收数据存储器写指针寄存器)
0xFE402B	S0_RX_WR1	
0xFE402C	S0_IMR	SOCKET 0中断屏蔽寄存器
0xFE402D	S0_FRAG0	S0_FRAG (在SOCKET1的IP头中的分片字段值寄存器)
0xFE402E	S0_FRAG1	
0xFE402F ~ 0xFE40FF		保留
0xFE4100	S1_MR	SOCKET 1模式寄存器
0xFE4101	S1_CR	SOCKET 1命令寄存器
0xFE4102	S1_IR	SOCKET 1中断寄存器
0xFE4103	S1_SR	SOCKET 1端口状态寄存器
0xFE4104	S1_PORT0	SOCKET1端口号寄存器
0xFE4105	S1_PORT1	
0xFE4106	S1_DHAR0	SOCKET 1目的硬件物理地址寄存器
0xFE4107	S1_DHAR1	
0xFE4108	S1_DHAR2	
0xFE4109	S1_DHAR3	
0xFE410A	S1_DHAR4	
0xFE410B	S1_DHAR5	

0xFE410C	S1_DIPR0	SOCKET 1目的IP地址寄存器
0xFE410D	S1_DIPR1	
0xFE410E	S1_DIPR2	
0xFE410F	S1_DIPR3	
0xFE4110	S1_DPORT0	SOCKET 1目的端口号寄存器
0xFE4111	S1_DPORT1	
0xFE4112	S1_MSSR0	SOCKET1最大分片长度寄存器
0xFE4113	S1_MSSR1	
0xFE4114	S1_PROTO	在IPRAW模式，SOCKET 1 IP头字段的协议寄存器
0xFE4115	S1_TOS	SOCKET 1的IP服务类型（TOS）寄存器
0xFE4116	S1_TTL	SOCKET 1的IP生存时间（TTL）寄存器
0xFE4117 ~ 0xFE411D		保留
0xFE411E	S1_RXMEM_SIZE	SOCKET 1接收数据存储器大小寄存器
0xFE411F	S1_TXMEM_SIZE	SOCKET 1发送数据存储器大小寄存器
0xFE4120	S1_TX_FSR0	S1_TX_FSR (SOCKET 1发送数据存储器剩余的空间大小寄存器)
0xFE4121	S1_TX_FSR1	
0xFE4122	S1_TX_RD0	S1_TX_RD (SOCKET 1发送数据存储器读指针寄存器)
0xFE4123	S1_TX_RD1	
0xFE4124	S1_TX_WR0	S1_TX_WR (SOCKET 1发送数据存储器写指针寄存器)
0xFE4125	S1_TX_WR1	
0xFE4126	S1_RX_RSR0	S1_RX_RSR (SOCKET 1接收数据字节长度寄存器)
0xFE4127	S1_RX_RSR1	
0xFE4128	S1_RX_RD0	S1_RX_RD (SOCKET 1接收数据存储器读指针寄存器)
0xFE4129	S1_RX_RD1	
0xFE412A	S1_RX_WR0	S1_RX_WR (SOCKET 1接收数据存储器写指针寄存器)
0xFE412B	S1_RX_WR1	
0xFE412C	S1_IMR	SOCKET 1中断屏蔽寄存器
0xFE412D	S1_FRAG0	S1_FRAG (在SOCKET1的IP头中的分片字段值寄存器)
0xFE412E	S1_FRAG1	
0xFE412F ~ 0xFE41FF		保留
0xFE4200	S2_MR	SOCKET 2模式寄存器
0xFE4201	S2_CR	SOCKET 2命令寄存器

0xFE4202	S2_IR	SOCKET 2中断寄存器
0xFE4203	S2_SR	SOCKET 2端口状态寄存器
0xFE4204	S2_PORT0	SOCKET2端口号寄存器
0xFE4205	S2_PORT1	
0xFE4206	S2_DHAR0	SOCKET 2目的硬件物理地址寄存器
0xFE4207	S2_DHAR1	
0xFE4208	S2_DHAR2	
0xFE4209	S2_DHAR3	
0xFE420A	S2_DHAR4	
0xFE420B	S2_DHAR5	
0xFE420C	S2_DIPR0	SOCKET 2目的IP地址寄存器
0xFE420D	S2_DIPR1	
0xFE420E	S2_DIPR2	
0xFE420F	S2_DIPR3	
0xFE4210	S2_DPORT0	SOCKET 2目的端口号寄存器
0xFE4211	S2_DPORT1	
0xFE4212	S2_MSSR0	SOCKET 2最大分片长度寄存器
0xFE4213	S2_MSSR1	
0xFE4214	S2_PROTO	在IPRAW模式，SOCKET 2 IP头字段协的议寄存器
0xFE4215	S2_TOS	SOCKET 2的IP服务类型（TOS）寄存器
0xFE4216	S2_TTL	SOCKET 2的IP生存时间（TTL）寄存器
0xFE4217 ~ 0xFE421D		保留
0xFE421E	S2_RXMEM_SIZE	SOCKET 2接收数据存储器大小寄存器
0xFE421F	S2_TXMEM_SIZE	SOCKET 2发送数据存储器大小寄存器
0xFE4220	S2_TX_FSR0	S2_TX_FSR (SOCKET 2发送数据存储器剩余的空间大小寄存器)
0xFE4221	S2_TX_FSR1	
0xFE4222	S2_TX_RD0	S2_TX_RD (SOCKET 2发送数据存储器读指针寄存器)
0xFE4223	S2_TX_RD1	
0xFE4224	S2_TX_WR0	S2_TX_WR (SOCKET 2发送数据存储器写指针寄存器)
0xFE4225	S2_TX_WR1	
0xFE4226	S2_RX_RSR0	S2_RX_RSR (SOCKET 2接收数据字节长度寄存器)
0xFE4227	S2_RX_RSR1	

0xFE4228	S2_RX_RD0	S2_RX_RD (SOCKET 2接收数据存储器读指针寄存器)
0xFE4229	S2_RX_RD1	
0xFE422A	S2_RX_WR0	S2_RX_WR (SOCKET 2接收数据存储器写指针寄存器)
0xFE422B	S2_RX_WR1	
0xFE422C	S2_IMR	SOCKET 2中断屏蔽寄存器
0xFE422D	S2_FRAG0	S2_FRAG (在SOCKET2的IP头中的分片字段值寄存器)
0xFE422E	S2_FRAG1	
0xFE422F ~ 0xFE42FF		保留
0xFE4300	S3_MR	SOCKET 3模式寄存器
0xFE4301	S3_CR	SOCKET 3命令寄存器
0xFE4302	S3_IR	SOCKET 3中断寄存器
0xFE4303	S3_SR	SOCKET 3端口状态寄存器
0xFE4304	S3_PORT0	SOCKET3端口号寄存器
0xFE4305	S3_PORT1	
0xFE4306	S3_DHAR0	SOCKET 3目的硬件物理地址寄存器
0xFE4307	S3_DHAR1	
0xFE4308	S3_DHAR2	
0xFE4309	S3_DHAR3	
0xFE430A	S3_DHAR4	
0xFE430B	S3_DHAR5	
0xFE430C	S3_DIPR0	SOCKET 3目的IP地址寄存器
0xFE430D	S3_DIPR1	
0xFE430E	S3_DIPR2	
0xFE430F	S3_DIPR3	
0xFE4310	S3_DPORT0	SOCKET 3目的端口号寄存器
0xFE4311	S3_DPORT1	
0xFE4312	S3_MSSR0	SOCKET 3最大分片长度寄存器
0xFE4313	S3_MSSR1	
0xFE4314	S3_PROTO	在IPRAW模式，SOCKET 3 IP头字段协的议寄存器
0xFE4315	S3_TOS	SOCKET 3的IP服务类型（TOS）寄存器
0xFE4316	S3_TTL	SOCKET 3的IP生存时间（TTL）寄存器

0xFE4317 ~ 0xFE431D		保留
0xFE431E	S3_RXMEM_SIZE	SOCKET 3接收数据存储器大小寄存器
0xFE431F	S3_TXMEM_SIZE	SOCKET 3发送数据存储器大小寄存器
0xFE4320	S3_TX_FSR0	S3_TX_FSR (SOCKET 3发送数据存储器剩余的空间大小寄存器)
0xFE4321	S3_TX_FSR1	
0xFE4322	S3_TX_RD0	S3_TX_RD (SOCKET 3发送数据存储器读指针寄存器)
0xFE4323	S3_TX_RD1	
0xFE4324	S3_TX_WR0	S3_TX_WR (SOCKET 3发送数据存储器写指针寄存器)
0xFE4325	S3_TX_WR1	
0xFE4326	S3_RX_RSR0	S3_RX_RSR (SOCKET 3接收数据字节长度寄存器)
0xFE4327	S3_RX_RSR1	
0xFE4328	S3_RX_RD0	S3_RX_RD (SOCKET 3接收数据存储器读指针寄存器)
0xFE4329	S3_RX_RD1	
0xFE432A	S3_RX_WR0	S3_RX_WR (SOCKET 3接收数据存储器写指针寄存器)
0xFE432B	S3_RX_WR1	
0xFE432C	S3_IMR	SOCKET 3中断屏蔽寄存器
0xFE432D	S3_FRAG0	S3_FRAG (在SOCKET3的IP头中的分片字段值寄存器)
0xFE432E	S3_FRAG1	
0xFE432F ~ 0xFE43FF		保留
0xFE4400	S4_MR	SOCKET 4模式寄存器
0xFE4401	S4_CR	SOCKET 4命令寄存器
0xFE4402	S4_IR	SOCKET 4中断寄存器
0xFE4403	S4_SR	SOCKET 4端口状态寄存器
0xFE4404	S4_PORT0	SOCKET4端口号寄存器
0xFE4405	S4_PORT1	
0xFE4406	S4_DHAR0	SOCKET 4目的硬件物理地址寄存器
0xFE4407	S4_DHAR1	
0xFE4408	S4_DHAR2	
0xFE4409	S4_DHAR3	
0xFE440A	S4_DHAR4	
0xFE440B	S4_DHAR5	

0xFE440C	S4_DIPR0	SOCKET 4目的IP地址寄存器
0xFE440D	S4_DIPR1	
0xFE440E	S4_DIPR2	
0xFE440F	S4_DIPR3	
0xFE4410	S4_DPORT0	SOCKET 4目的端口号寄存器
0xFE4411	S4_DPORT1	
0xFE4412	S4_MSSR0	SOCKET4最大分片长度寄存器
0xFE4413	S4_MSSR1	
0xFE4414	S4_PROTO	在IPRAW模式，SOCKET4 IP头字段协的议寄存器
0xFE4415	S4_TOS	SOCKET 4的IP服务类型（TOS）寄存器
0xFE4416	S4_TTL	SOCKET 4的IP生存时间（TTL）寄存器
0xFE4417 ~ 0xFE441D		保留
0xFE441E	S4_RXMEM_SIZE	SOCKET 4接收数据存储器大小寄存器
0xFE441F	S4_TXMEM_SIZE	SOCKET 4发送数据存储器大小寄存器
0xFE4420	S4_TX_FSR0	S4_TX_FSR (SOCKET 4发送数据存储器剩余的空间大小寄存器)
0xFE4421	S4_TX_FSR1	
0xFE4422	S4_TX_RD0	S4_TX_RD (SOCKET 4发送数据存储器读指针寄存器)
0xFE4423	S4_TX_RD1	
0xFE4424	S4_TX_WR0	S4_TX_WR (SOCKET 4发送数据存储器写指针寄存器)
0xFE4425	S4_TX_WR1	
0xFE4426	S4_RX_RSR0	S4_RX_RSR (SOCKET 4接收数据字节长度寄存器)
0xFE4427	S4_RX_RSR1	
0xFE4428	S4_RX_RD0	S4_RX_RD (SOCKET 4接收数据存储器读指针寄存器)
0xFE4429	S4_RX_RD1	
0xFE442A	S4_RX_WR0	S4_RX_WR (SOCKET 4接收数据存储器写指针寄存器)
0xFE442B	S4_RX_WR1	
0xFE442C	S4_IMR	SOCKET 4中断屏蔽寄存器
0xFE442D	S4_FRAG0	S4_FRAG (在SOCKET4的IP头中的分片字段值寄存器)
0xFE442E	S4_FRAG1	
0xFE442F ~ 0xFE44FF		保留
0xFE4500	S5_MR	SOCKET 5模式寄存器
0xFE4501	S5_CR	SOCKET 5命令寄存器

0xFE4502	S5_IR	SOCKET 5中断寄存器
0xFE4503	S5_SR	SOCKET 5端口状态寄存器
0xFE4504	S5_PORT0	SOCKET5端口号寄存器
0xFE4505	S5_PORT1	
0xFE4506	S5_DHAR0	SOCKET 5目的硬件物理地址寄存器
0xFE4507	S5_DHAR1	
0xFE4508	S5_DHAR2	
0xFE4509	S5_DHAR3	
0xFE450A	S5_DHAR4	
0xFE450B	S5_DHAR5	
0xFE450C	S5_DIPR0	SOCKET 5目的IP地址寄存器
0xFE450D	S5_DIPR1	
0xFE450E	S5_DIPR2	
0xFE450F	S5_DIPR3	
0xFE4510	S5_DPORT0	SOCKET 5目的端口号寄存器
0xFE4511	S5_DPORT1	
0xFE4512	S5_MSSR0	SOCKET 5最大分片长度寄存器
0xFE4513	S5_MSSR1	
0xFE4514	S5_PROTO	在IPRAW模式，SOCKET 5 IP头字段协的议寄存器
0xFE4515	S5_TOS	SOCKET 5的IP服务类型（TOS）寄存器
0xFE4516	S5_TTL	SOCKET 5的IP生存时间（TTL）寄存器
0xFE4517 ~ 0xFE451D		保留
0xFE451E	S5_RXMEM_SIZE	SOCKET 5接收数据存储器大小寄存器
0xFE451F	S5_TXMEM_SIZE	SOCKET 5发送数据存储器大小寄存器
0xFE4520	S5_TX_FSR0	S5_TX_FSR (SOCKET 5发送数据存储器剩余的空间大小寄存器)
0xFE4521	S5_TX_FSR1	
0xFE4522	S5_TX_RD0	S5_TX_RD (SOCKET 5发送数据存储器读指针寄存器)
0xFE4523	S5_TX_RD1	
0xFE4524	S5_TX_WR0	S5_TX_WR (SOCKET 5发送数据存储器写指针寄存器)
0xFE4525	S5_TX_WR1	
0xFE4526	S5_RX_RSR0	S5_RX_RSR (SOCKET 5接收数据字节长度寄存器)
0xFE4527	S5_RX_RSR1	

0xFE4528	S5_RX_RD0	S5_RX_RD (SOCKET 5接收数据存储器读指针寄存器)
0xFE4529	S5_RX_RD1	
0xFE452A	S5_RX_WR0	S5_RX_WR (SOCKET 5接收数据存储器写指针寄存器)
0xFE452B	S5_RX_WR1	
0xFE452C	S5_IMR	SOCKET 5中断屏蔽寄存器
0xFE452D	S5_FRAG0	S5_FRAG (在SOCKET5的IP头中的分片字段值寄存器)
0xFE452E	S5_FRAG1	
0xFE452F ~ 0xFE45FF		保留
0xFE4600	S6_MR	SOCKET 6模式寄存器
0xFE4601	S6_CR	SOCKET 6命令寄存器
0xFE4602	S6_IR	SOCKET 6中断寄存器
0xFE4603	S6_SR	SOCKET 6端口状态寄存器
0xFE4604	S6_PORT0	SOCKET6端口号寄存器
0xFE4605	S6_PORT1	
0xFE4606	S6_DHAR0	SOCKET6目的硬件物理地址寄存器
0xFE4607	S6_DHAR1	
0xFE4608	S6_DHAR2	
0xFE4609	S6_DHAR3	
0xFE460A	S6_DHAR4	
0xFE460B	S6_DHAR5	
0xFE460C	S6_DIPR0	SOCKET 6目的IP地址寄存器
0xFE460D	S6_DIPR1	
0xFE460E	S6_DIPR2	
0xFE460F	S6_DIPR3	
0xFE4610	S6_DPORT0	SOCKET 6目的端口号寄存器
0xFE4611	S6_DPORT1	
0xFE4612	S6_MSSR0	SOCKET 6最大分片长度寄存器
0xFE4613	S6_MSSR1	
0xFE4614	S6_PROTO	在IPRAW模式，SOCKET 6 IP头字段协的议寄存器
0xFE4615	S6_TOS	SOCKET 6的IP服务类型（TOS）寄存器
0xFE4616	S6_TTL	SOCKET 6的IP生存时间（TTL）寄存器

0xFE4617 ~ 0xFE461D		保留
0xFE461E	S6_RXMEM_SIZE	SOCKET 6接收数据存储器大小寄存器
0xFE461F	S6_TXMEM_SIZE	SOCKET 6发送数据存储器大小寄存器
0xFE4620	S6_TX_FSR0	S6_TX_FSR (SOCKET 6发送数据存储器剩余的空间大小寄存器)
0xFE4621	S6_TX_FSR1	
0xFE4622	S6_TX_RD0	S6_TX_RD (SOCKET 6发送数据存储器读指针寄存器)
0xFE4623	S6_TX_RD1	
0xFE4624	S6_TX_WR0	S6_TX_WR (SOCKET 6发送数据存储器写指针寄存器)
0xFE4625	S6_TX_WR1	
0xFE4626	S6_RX_RSR0	S6_RX_RSR (SOCKET 6接收数据字节长度寄存器)
0xFE4627	S6_RX_RSR1	
0xFE4628	S6_RX_RD0	S6_RX_RD (SOCKET 6接收数据存储器读指针寄存器)
0xFE4629	S6_RX_RD1	
0xFE462A	S6_RX_WR0	S6_RX_WR (SOCKET 6接收数据存储器写指针寄存器)
0xFE462B	S6_RX_WR1	
0xFE462C	S6_IMR	SOCKET 6中断屏蔽寄存器
0xFE462D	S6_FRAG0	S6_FRAG (在SOCKET6的IP头中的分片字段值寄存器)
0xFE462E	S6_FRAG1	
0xFE462F ~ 0xFE46FF		保留
0xFE4700	S7_MR	SOCKET 7模式寄存器
0xFE4701	S7_CR	SOCKET 7命令寄存器
0xFE4702	S7_IR	SOCKET 7中断寄存器
0xFE4703	S7_SR	SOCKET 7端口状态寄存器
0xFE4704	S7_PORT0	SOCKET7端口号寄存器
0xFE4705	S7_PORT1	
0xFE4706	S7_DHAR0	SOCKET 7目的硬件物理地址寄存器
0xFE4707	S7_DHAR1	
0xFE4708	S7_DHAR2	
0xFE4709	S7_DHAR3	
0xFE470A	S7_DHAR4	
0xFE470B	S7_DHAR5	

0xFE470C	S7_DIPR0	SOCKET 7目的IP地址寄存器
0xFE470D	S7_DIPR1	
0xFE470E	S7_DIPR2	
0xFE470F	S7_DIPR3	
0xFE4710	S7_DPORT0	SOCKET 7目的端口号寄存器
0xFE4711	S7_DPORT1	
0xFE4712	S7_MSSR0	SOCKET 7最大分片长度寄存器
0xFE4713	S7_MSSR1	
0xFE4714	S7_PROTO	在IPRAW模式，SOCKET 7 IP头字段协的议寄存器
0xFE4715	S7_TOS	SOCKET 7的IP服务类型（TOS）寄存器
0xFE4716	S7_TTL	SOCKET 7的IP生存时间（TTL）寄存器
0xFE4717 ~ 0xFE471D		保留
0xFE471E	S7_RXMEM_SIZE	SOCKET 7接收数据存储器大小寄存器
0xFE471F	S7_TXMEM_SIZE	SOCKET 7发送数据存储器大小寄存器
0xFE4720	S7_TX_FSR0	S7_TX_FSR (SOCKET 7发送数据存储器剩余的空间大小寄存器)
0xFE4721	S7_TX_FSR1	
0xFE4722	S7_TX_RD0	S7_TX_RD (SOCKET 7发送数据存储器读指针寄存器)
0xFE4723	S7_TX_RD1	
0xFE4724	S7_TX_WR0	S7_TX_WR (SOCKET 7发送数据存储器写指针寄存器)
0xFE4725	S7_TX_WR1	
0xFE4726	S7_RX_RSR0	S7_RX_RSR (SOCKET 7接收数据字节长度寄存器)
0xFE4727	S7_RX_RSR1	
0xFE4728	S7_RX_RD0	S7_RX_RD (SOCKET 7接收数据存储器读指针寄存器)
0xFE4729	S7_RX_RD1	
0xFE472A	S7_RX_WR0	S7_RX_WR (SOCKET 7接收数据存储器写指针寄存器)
0xFE472B	S7_RX_WR1	
0xFE472C	S7_IMR	SOCKET 7中断屏蔽寄存器
0xFE472D	S7_FRAG0	S7_FRAG (在SOCKET7的IP头中的分片字段值寄存器)
0xFE472E	S7_FRAG1	
0xFE472F ~ 0xFE47FF		保留

8.3 Register Description寄存器描述

8.3.1 模式寄存器

MR (模式寄存器) [R/W] [0xFE0000] [0x00]

该寄存器用于软件复位、Ping指令阻止和PPPoE模式。

7	6	5	4	3	2	1	0
RST			PB	PPPoE			

位	符号	描述
7	RST	**软件复位，**对这一位置'1'使内部寄存器初始化。复位完成后自动清0
6	Reserved	保留
5	Reserved	保留
4	PB	**Ping阻止模式** 0：允许Ping 1：禁止Ping 如果PB=1，将不会响应Ping请求
3	PPPoE	**PPPoE模式** 0：禁止PPPoE模式 1：允许PPPoE模式 如果用户没有经过路由器而直接使用ADSL，该位需要置'1'与ADSL服务器连接。详细过程参考应用笔记"怎样与ADSL连接？"
2	Reserved	保留
1	Reserved	保留
0	Reserved	保留

GAR (网关IP地址寄存器) [R/W] [0xFE0001 – 0xFE0004] [0x00]

该寄存器设置默认的网关地址。

例) 网关地址为"192.168.0.1"

0xFE0001	0xFE0002	0xFE0003	0xFE0004
192 (0xC0)	168 (0xA8)	0 (0x00)	1 (0x01)

SUBR (子网掩码寄存器) [R/W] [0xFE0005 – 0xFE0008] [0x00]

该寄存器设置子网掩码值。

例：设置子网掩码为"255.255.255.0"

0xFE0005	0xFE0006	0xFE0007	0xFE0008
255 (0xFF)	255 (0xFF)	255 (0xFF)	0 (0x00)

SHAR (本机物理地址寄存器) [R/W] [0xFE0009 – 0xFE000E] [0x00]

该寄存器设置本机物理地址。

例：设置物理地址为“00.08.DC.01.02.03”。

0xFE0009	0xFE000A	0xFE000B	0xFE000C	0xFE000D	0xFE000E
0x00	0x08	0xDC	0x01	0x02	0x03

SIPR (本机IP地址) [R/W] [0xFE000F – 0xFE0012] [0x00]

该寄存器设置本机IP地址。

例：设置IP地址为“192.168.0.2”。

0xFE000F	0xFE0010	0xFE0011	0xFE0012
192 (0xC0)	168 (0xA8)	0 (0x00)	2 (0x02)

IR (中断寄存器) [R] [0xFE0015] [0x00]

W7100A的MCU读取该寄存器，以确定产生中断的来源。IR的任意位置位后，INT5（TCP/IP内核中断）都将产生低电平信号。它将一直保持低电平直到中断寄存器中的所有位都清除。

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

位	符号	描述
7	CONFLICT	**IP冲突** 如果在ARP请求时获得了一个与本机相同的IP地址，这一位将置‘1’。对该位写‘1’将清除
6	UNREACH	**目的地址无法到达** 在UDP传输时，如果目的地址不存在，W7100A将收到ICMP（地址无法到达）的数据包信息。在这种情况下IP地址和端口号将保存到无法到达的IP地址寄存器（UIPR）和无法到达的端口号寄存器（UPORT），同时将UNREACH置‘1’。对该位写‘1’将被清除
5	PPPoE	**PPPoE连接关闭** 在PPPoE模式，如果关闭PPPoE连接，该位置‘1’。对该位写‘1’将被清除
4	Reserved	保留
3	Reserved	保留
2	Reserved	保留
1	Reserved	保留
0	Reserved	保留

IMR (中断屏蔽寄存器) [R/W] [0xFE0016] [0x00]

中断屏蔽寄存器用于屏蔽中断。每个中断屏蔽位对应中断寄存器（IR2）中的一个位。如果中断屏蔽位被置‘1’时 ， 在 任何时候只要IR2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IR2中断位被置‘1’也不会产生中断。

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

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

RTR (重发时间周期寄存器) [R/W] [0xFE0017 – 0xFE0018] [0x07D0]

该寄存器用来设置时间溢出的值。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

0xFE0017	0xFE0018
0x0F	0xA0

如果对端没有响应、或响应延迟都将产生重发。

RCR (重发计数寄存器) [R/W] [0xFE0019] [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RCR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’）

在TCP通信模式，Sn_IR的TIMEOUT=1时，Sn_SR的状态改变为“SOCK_CLOSED”状态。在其它通信模式，只是Sn_IR的TIMEOUT=1.

W7100A的超时可以通过RTR和RCR设置。W7100A的超时有ARP超时和TCP重发超时。

在ARP模式（参考RFC 826， http://www.ietf.org/rfc.html）下重发超时，W7100A自动发送ARP请求到对端IP地址，以获得对端的MAC地址信息（在IP，UDP或TCP模式中都使用）。由于等待对端的ARP响应，如果在RTR时间内没有响应，将产生超时并再次发送ARP请求。这个过程一直要重复“RCR+1”次。

即使重复“RCR+1”次的ARP请求都没有得到ARP响应，超时将会产生，Sn_IR的TIMEOUT=1。

ARP请求的超时值计算如下：

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

在TCP数据包重发超时过程中，W7100A发送TCP数据包（SYN，FIN，RST，DATA数据包）并等待对端的响应（ACK），等待时间由RTR确定，重发次数由RCR确定。如果对端没有响应（ACK），将产生超时，先前的TCP数据包将重发。重发次数为“RCR+1”。即使这个TCP数据包重发了“RCR+1”次都没有得到对端的响应（ACK），最终将产生超时，Sn_SSR的状态将改变为“SOCK_CLOSED”，且Sn_IR的TIMEOUT=1。

TCP重复发送的最终超时计算如下：

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

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

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

ARP超时= 2000 X 0.1ms X 9 = 1800ms = 1.8s

TCP超时= (0x07D0 + 0x0FA0 + 0x1F40 + 0x3E80 + 0x7D00 + 0xFA00 + 0xFA00 + 0xFA00 + 0xFA00) X 0.1ms

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

= 318000 X 0.1ms = 31.8s

PATR (PPPoE模式下认证类型) [R] [0xFE001C-0xFE001D] [0x0000]

在与PPPoE服务器连接时，该寄存器指示安全认证的方法。W5100只支持两种安全认证类型：PAP及CHAP。

值	认证类型
0xC023	PAP
0xC223	CHAP

PPPALGO (PPPoE模式下认证算法)[R][0xFE001E][0x00]

该寄存器用于指示PPPoE连接时的认证算法。详细信息请参考PPPoE应用笔记。

PTIMER (PPP连接控制协议（LPC）请求定时器寄存器) [R/W] [0xFE0028] [0x28]

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

例：设置PTIMER = 200，

200 * 25(ms) = 5000(ms) = 5 秒

PMAGIC (PPP连接控制协议（LPC）魔数寄存器) [R/W] [0xFE0029][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。

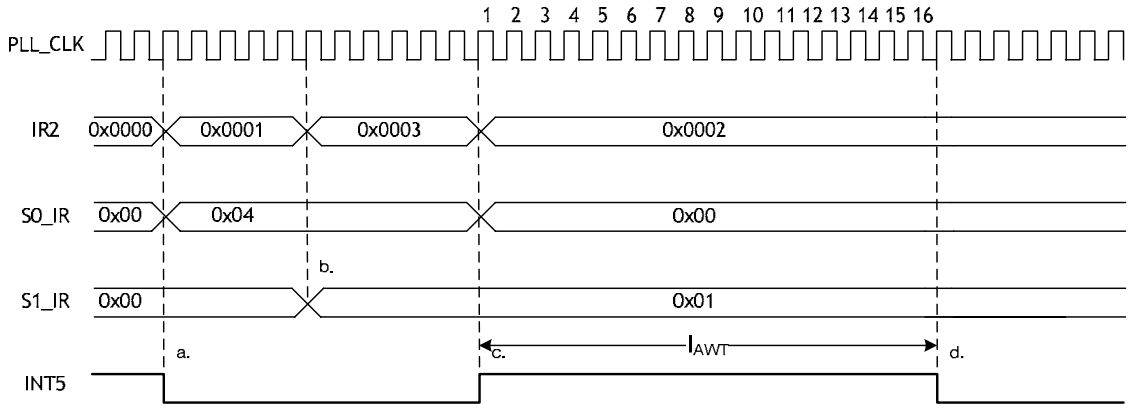
VERSIONR (W7100A芯片版本寄存器)[R][0xFE001F][0x02]

该寄存器存储W7100A的版本信息。

INTLEVEL (中断信号低电平时间寄存器)[R/W][0xFE0030 – 0xFE0031][0x0000]

INTLEVEL寄存器设置中断信号低电平等待时间(I_{AWT})，它配置内部INT5信号低电平持续的时间，直到下一次中断的产生。如果用户想使用TCP/IP内核中断，INTLEVEL的值必须大于0x2B00，否则TCP/IP内核中断可能被忽略。

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



a. 假设SOCKET0产生中断（S0_IR(3)=1），且对应的IR2 位被设置（IR2（S0_IR）=1），那么内部INT5信号将改变为低电平。

b. 假设SOCKET1继续产生中断（S1_IR(0)=1），且对应的IR2位被置位（IR2（S1_IR）=1）。

c. 当主机将S0_IR清零时，对应的IR2的位也将清零（IR2(S0_IR)=0）。INT5将由低电平变成高电平。

d. 当S1_IR清除后，因为SOCKET1中断，对应的IR2并不为0x00， 内部INT5信号应该为低电平。

然而，由于INTLEVEL值为0x000F，内部INT5信号还要持续I_{AWT}(16 PLL_CLK)时间。

IR2 (W7100A端口中断寄存器)[R/W][0xFE0034][0x00]

IR2寄存器用于通知主机W7100A产生端口中断。当中断产生后，在IR2的相关位置‘1’。此时，INT5（TCPIP内核中断）引脚输出低电平信号，直到IR2的所有的位都为‘0’。一旦通过Sn_IR将IR2的所有位都清零，INT5就变成高电平。

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

位	符号	描述
7	S7_INT	**SOCKET7产生中断** 当SOCKET7产生中断时，这一位置‘1’，这是S7_IR中断信息。当主机将S7_IR清0后，这一位自动清除
6	S6_INT	**SOCKET6产生中断** 当SOCKET6产生中断时，这一位置‘1’，这是S6_IR中断信息。当主机将S6_IR清0后，这一位自动清除
5	S5_INT	**SOCKET5产生中断** 当SOCKET5产生中断时，这一位置‘1’，这是S5_IR中断信息。当主机将S5_IR清0后，这一位自动清除
4	S4_INT	**SOCKET4产生中断** 当SOCKET4产生中断时，这一位置‘1’，这是S4_IR中断信息。当主机将S4_IR清0后，这一位自动清除
3	S3_INT	**SOCKET3产生中断** 当SOCKET3产生中断时，这一位置‘1’，这是S3_IR中断信息。当主机将S3_IR清0后，这一位自动清除
2	S2_INT	**SOCKET2产生中断** 当SOCKET2产生中断时，这一位置‘1’，这是S2_IR中断信息。当主机将S2_IR清0后，这一位自动清除
1	S1_INT	**SOCKET1产生中断** 当SOCKET1产生中断时，这一位置‘1’，这是S1_IR中断信息。当主机将S1_IR清0后，这一位自动清除
0	S0_INT	**SOCKET0产生中断** 当SOCKET0产生中断时，这一位置‘1’，这是S0_IR中断信息。当主机将S0_IR清0后，这一位自动清除

8.3.2 端口寄存器

Sn_MR (SOCKET *n* 模式寄存器)[R/W][0xFE4000 + 0x100n][0x0000]

该寄存器配置SOCKET n的协议或选项。

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

位	符号	描述
7	MULTI	**多播，**只有在UDP模式下有效(P3-P0：0010) 0：禁止多播；1：允许多播 为了使用多播，在打开SOCKET之前，将多播分组的IP地址写到SOCKET n的目的IP寄存器，将多播分组的端口号写到SOCKET n的目的端口号寄存器。
6	MF	MAC地址过滤 0：禁止MAC地址过滤；1：允许MAC地址过滤 除了本机MAC地址和广播MAC地址以外，过滤其它MAC地址
5	ND/MC	**使用无延迟响应（ACK），**只在TCP（P3-P0：0001）模式下有效。 0：禁止无延迟ACK；1：启动无延迟ACK 如果这一位设置为'1', 在收到对端的数据包后马上响应一个ACK数据包。如果这一位清'0', ACK将根据内部时间溢出机制进行响应 **多播，**只有在UDP模式下（P3-P0：0010），且MULTI位有效的情况下才有效 0：使用IGMP版本2；1：使用IGMP版本1 另外多播也可以在诸如在线/离开/报告给多播分组等的IGMP信息中发送版本号
4	Reserved	保留
3 2 1 0	P3 P2 P1 P0	**协议** 设置相应的SOCKET为TCP、UDP、或IP RAW等模式

Symbol	P3	P2	P1	P0	Meaning
Sn_MR_CLOSE	0	0	0	0	Closed
Sn_MR_TCP	0	0	0	1	TCP
Sn_MR_UDP	0	0	1	0	UDP
Sn_MR_IPRAW	0	0	1	1	IPRAW
S0_MR_MACRAW	0	1	0	0	MAC RAW
S0_MR_PPPoE	0	1	0	1	PPPoE

S0_MR_MACRAW 和S0_MR_PPPoE只有SOCKET0有效

S0_MR_PPPoE是临时用于与PPPoE服务器的连接和中断，连接建立以后，该端口可以用于其它协议

Sn_CR (SOCKET *n*命令寄存器)[R/W][0xFE4001 + 0x100n][0x00]

该寄存器用于设置SOCKET n的命令，诸如：打开、关闭、连接、侦听、发送、接收等动作。W7100A确认该命令以后，Sn_CR寄存器自动清零(0x00)。即使Sn_CR清零，指令仍然在处理中。为了验证命令是否执行完成，可以检查Sn_IR寄存器或Sn_SR寄存器。

<table>
<tr><th>值</th><th>符号</th><th>描述</th></tr>
<tr><td>0x01</td><td>OPEN</td><td>根据Sn_MR(P3:P0)所选择的协议初始化并打开SOCKET n。下表显示对应Sn_MR的Sn_SR的值。
<table>
<tr><td>Sn_MR(P3:P0)</td><td>Sn_SR</td></tr>
<tr><td>Sn_MR_CLOSE(0x00)</td><td>-</td></tr>
<tr><td>Sn_MR_TCP(0x01)</td><td>SOCK_INIT(0x13)</td></tr>
<tr><td>Sn_MR_UDP(0x02)</td><td>SOCK_UDP(0x22)</td></tr>
<tr><td>Sn_MR_IPRAW(0x03)</td><td>SOCK_IPRAW(0x32)</td></tr>
<tr><td>S0_MR_MACRAW(0x04)</td><td>SOCK_MACRAW(0x42)</td></tr>
<tr><td>S0_MR_PPPoE(0x05)</td><td>SOCK_PPPoE(0x5F)</td></tr>
</table></td></tr>
<tr><td>0x02</td><td>LISTEN</td><td>该命令只有在TCP模式下有效 (Sn_MR(P3:P0) = Sn_MR_TCP)。
在这种模式下， SOCKET n配置为TCP服务器， 等待其它TCP客户端的连接请求（SYN数据包）。Sn_SR寄存器从SOCK_INIT状态改变为SOCK_LISTEN状态。
当一个客户端的连接请求成功建立，Sn_SR寄存器的状态将从SOCK_LISTEN改变为SOCK_ESTABLISHED，且SN_SR(0)置‘1’。相反，如果连接失败(SYN/ACK数据包发送失败)，Sn_SR(3)置‘1’且Sn_SR变为SOCK_CLOSED。
例如，如果在连接请求过程中TCP客户端的目标端口号不存在，W7100A将发送一个RST数据包，且Sn_SR的状态不变。</td></tr>
<tr><td>0x04</td><td>CONNECT</td><td>该操作只有在TCP模式下有效，且SOCKET n设置为TCP客户端。
向TCP服务器发出连接请求（SYN数据包）。TCP服务器的IP地址和端口号由目的IP地址寄存器(Sn_DIPR0)和目的端口号寄存器(Sn_DPORT0)确定。
当客户端的连接请求成功建立，Sn_SR寄存器将改变为SOCK_ESTABLISHED，且Sn_IR(0)改变为‘1’。在下面几种情况下，连接请求失败：
当一个ARP超时异常发生(Sn_IR(s)=1)， 表明没有能够通过ARP过程获得目的硬件地址。
如果没有接收到SYN/ACK数据包，且产生TCP超时异常(Sn_IR(3)=1)。
如果接收到RST数据包而不是SYN/ACK数据包。
在这些情况下，Sn_SR状态改变为SOCK_CLOSED。</td></tr>
<tr><td>0x08</td><td>DISCON</td><td>只有在TCP模式下有效。
不管是“TCP服务器”还是“TCP客户端”，断开连接处理：
主动关闭：它将发送断开连接请求（FIN数据包）到连接的对端。
被动关闭：当接收到对端的FIN数据包，响应一个FIN数据包到对端。
当收到FIN/ACK数据包时，Sn_SR的状态改变为SOCK_CLOSED。</td></tr>
</table>

		如果没有收到断开连接的请求，将产生TCP超时（Sn_IR(3)=1），且Sn_SR的状态改变为SOCK_CLOSED。 例如：如果用CLOSE代替DISCON，Sn_SR的状态将变成CLOSED，但不会有断开连接的过程 （ 不产生断开连接的请求 ） 。如果在通信过程中接收到RST数据包，Sn_SR无条件地改变为SOCK_CLOSED。
0x10	CLOSE	关闭SOCKET n。 Sn_SR的状态为SOCK_CLOSED。
0x20	SEND	SEND命令将TX数据存储器中的所有数据都发送出去。详细资料请查看SOCKET n的TX剩余空间大小寄存器（Sn_TX_FSR0），SOCKET n的TX写指针寄存器（Sn_TX_RD0）。
0x21	SEND_MAC	只有在UDP模式有效。 其基本操作与SEND相同。但SEND操作需要通过ARP（地址解析协议）过程获得目的硬件物理地址。SEND_MAC不需要通过ARP获得目的硬件物理地址，由用户设置SOCKET n的目的硬件物理地址。
0x22	SEND_KEEP	只有在TCP模式下有效。 通过发送一个字节的数据检查连接的状态。如果没有对端的响应或中断，将产生超时中断。
0x40	RECV	RECV命令通过RX读指针寄存器(Sn_RX_RD0)处理数据的接收。 详细的信息请参考9.2.1.1服务器模式接收处理，SOCKET n的RX接收数据长度寄存器(Sn_RX_RSR0)，SOCKET n的RX写指针寄存器(Sn_RX_WR0)，SOCKET n的RX读指针寄存器(Sn_RX_RD0)。

下面的命令只用于SOCKET0和S0_MR(P3:P0)=S0_MR_PPPoE。

详细信息请参考"How to use ADSL"

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

Sn_IR (SOCKET *n* 中断寄存器)[R/W][0xFE4002 + 0x100n][0x00]

Sn_IR寄存器提供SOCKET n的中断类型信息（比如建立连接，中断连接、收到数据、超时等）。当产生一个中断、且中断屏蔽寄存器Sn_IMR的对应位为'1'，那么Sn_IR的中断位将置'1'。

为了清除Sn_IR位，主机需要向该位写入'1'。当Sn_IR所有的位都清除，IR(n)将自动清除。

它将影响INT5（TCPIP内核中断）的信号。

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

位	符号	描述
7	PRECV	PPP接收中断，收到不支持的选项。
6	PFAIL	PPP错误中断，PAP认证失败。
5	PNEXT	进入PPP下一阶段中断，在ADSL连接过程中， 阶段发生改变。
4	SENDOK	SEND OK中断，SEND命令执行结束。
3	TIMEOUT	TIMEOUT中断，产生ARP或TCP超时。
2	RECV	接收中断，收到对端的数据包
1	DISCON	断开连接中断，接收到对端来的FIN/ACK的FIN数据包。
0	CON	连接中断，端口与对端建立连接，端口状态改变为SOCKET_ESTABLISHED

Sn_IMR (SOCKET *n* 中断屏蔽寄存器)[R/W][0xFE402C + 0x100n][0xFF]

它配置SOCKET n的中断类型并通知主机。Sn_IMR的中断屏蔽位对应Sn_IR的中断位。如果端口产生中断，那么对应的Sn_IR的位置‘1’，如果对应的Sn_IMR和Sn_IR位都为‘1’，那么IR(n)将变为‘1’。这时如果IMR(n)也为‘1’，那么将产生中断（INT5输出低电平）。

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

位	符合	描述
7	PRECV	Sn_IR(PRECV)中断屏蔽 只有在‘SOCKET = 0’ 且 ‘S0_MR(P3:P0) = S0_MR_PPPoE’时有效
6	PFAIL	Sn_IR(PFAIL)中断屏蔽 只有在 ‘SOCKET = 0’ 且 ‘S0_MR(P3:P0) = S0_MR_PPPoE’时有效
5	PNEXT	Sn_IR(PNEXT)中断屏蔽 只有在 ‘SOCKET = 0’ 且 ‘S0_MR(P3:P0) = S0_MR_PPPoE’时有效
4	SENDOK	Sn_IR(SENDOK)中断屏蔽
3	TIMEOUT	Sn_IR(TIMEOUT)中断屏蔽
2	RECV	Sn_IR(RECV)断屏蔽
1	DISCON	Sn_IR(DISCON)中断屏蔽
0	CON	Sn_IR(CON) 中断屏蔽

Sn_SR (SOCKET *n* 状态寄存器)[R][0xFE4003 + 0x100n][0x00]

该寄存器提供SOCKET n的状态。在对Sn_CR寄存器进行操作或数据包的收发过程中，SOCKET n的状态将发生改变。

SOCKET n的不同状态将在下表进行描述。

值	符号	描述
0x00	SOCK_CLOSED	当执行DISCON或CLOSE命令，或发生ARP超时、TCP超时，不管以前是什么状态，其当前的状态都将改变为SOCK_CLOSED。
0x13	SOCK_INIT	在这种状态下，端口以TCP方式打开，并初始化为TCP连接前的状态。此时用户可以使用LISTEN和CONNECT命令。 当Sn_MR(P3:P0)设置为Sn_MR_TCP并执行OPEN命令，此时的状态将为SOCK_INIT。
0x14	SOCK_LISTEN	端口工作在TCP服务器模式，等待TCP客户端的连接请求（SYN数据包）。 当执行LISTEN命令后进入到SOCK_LISTEN状态 。 一旦建立连接， 端 口的状态将由SOCK_LISTEN改变为SOCK_ESTABLISHED；如果连接失败，或产生TCP超时(Sn_IR(TIME_OUT)=1)，状态变为SOCK_CLOSED。
0x17	SOCK_ ESTABLISHED	一 旦 收到TCP客户端 的 SYN数据包 ， 端口状态将从SOCK_LISTEN或CONNETCT改变到SOCK_ESTABLISHED。这时可以使用SEND或RECV命令进行数据交换。
0x1C	SOCK_ CLOSE_WAIT	在这种状态下，收到对端的断开连接的请求（FIN数据包）。虽然TCP连接半关断，但数据包仍然可以交换。为了完全断开TCP连接，需要使用DISCON命令。如果端口不通过断开连接的过程而直接关闭，可以使用CLOSE命令。
0x22	SOCK_UDP	端口以UDP模式打开。当Sn_MR(P3:P0)=Sn_MR_UDP且执行OPEN命令后，端口的状态改变为SOCK_UDP。与TCP模式不同，端口在UDP模式下不需要建立连接就可以进行数据通信。
0x32	SOCK_IPRAW	端口以IPRAW模式打开。当Sn_MR(P3:P0)=Sn_MR_IPRAW且执行OPEN命令后，端口的状态将改变为SOCK_IPRAW。与UDP相同，不需要建立连接就可以进行IP层的数据通信。
0x42	SOCK_ MACRAW	端口0以MACRAW模式打开。当S0_MR(P3:P0)=S0_MR_MACRAW且执行OPEN命令后，端口的状态将改变为SOCK_MACRAW。与UDP相同，不需要建立连接就可以进行MAC层的数据通信（以太网数据包）。
0x5F	SOCK_PPPOE	端口0以PPPoE模式打开。当S0_MR(P3:P0)=S0_MR_PPPOE且执行OPEN命令后，端口0的状态将改变为SOCK_PPPOE。

下面是Sn_SR在改变过程中的临时状态。

值	符号	描述
0x15	SOCK_ SYNSENT	该状态表示发送了一个连接请求（SYN数据包）给TCP服务器。 SYNSENT是在SOCK_INIT和SOCK_ESTABLISHED之间的一个中间状态，如果收到TCP服务器的接受连接的数据包（SYN/ACK），端口的状态自动改变为SOCK_ESTABLISHED。但是，如果产生TCP超时之前（Sn_IR(TIMEOUT)=1）没有收到SYN/ACK数据包，状态将变为SOCK_CLOSED。
0x16	SOCK_ SYNRECV	该状态表示端口接收到TCP客户端的连接请求（SYN数据包）。 当端口成功地给TCP客户端发出接受连接（SYN/ACK）的数据包后，端口状态改变为SOCK_ESTABLISHED。如果发送不成功，或产生TCP超时(Sn_IR(TIMEOUT)=1)，端口将改变为SOCK_CLOSED。
0x18	SOCK_FIN_WAIT	这些都是终止连接过程的状态。如果终止连接成功，或产生超时中断，端口状态都改变为SOCK_CLOSED。
0x1A	SOCK_CLOSING	
0X1B	SOCK_TIME_WAIT	
0X1D	SOCK_LAST_ACK	
0x01	SOCK_ARP	该状态指示端口正在发送ARP请求到对端，以获得对端的物理地址。 在UDP、IPRAW和TCP模式使用SEND命令时，端口都将变为SOCK_ARP 如果W7100A获得了目的端的物理地址（接收到ARP响应），端口状态将改变为SOCK_UDP、SOCK_IPRAW 或SOCK_SYNSENT。 如果 W 7 1 0 0 A 没有获得对端的物理地址 ， 而 且产生了ARP超时(Sn_IR(TIMEOUT)=1) ， 这时端口将返回到UDP或IPRAW先前的状态。在TCP模式，端口进入SOCK_CLOSED状态。 例：在UDP和IPRAW模式，Sn_DIPR寄存器将把当前的值和先前的值进行比较，如果不同则进行ARP。

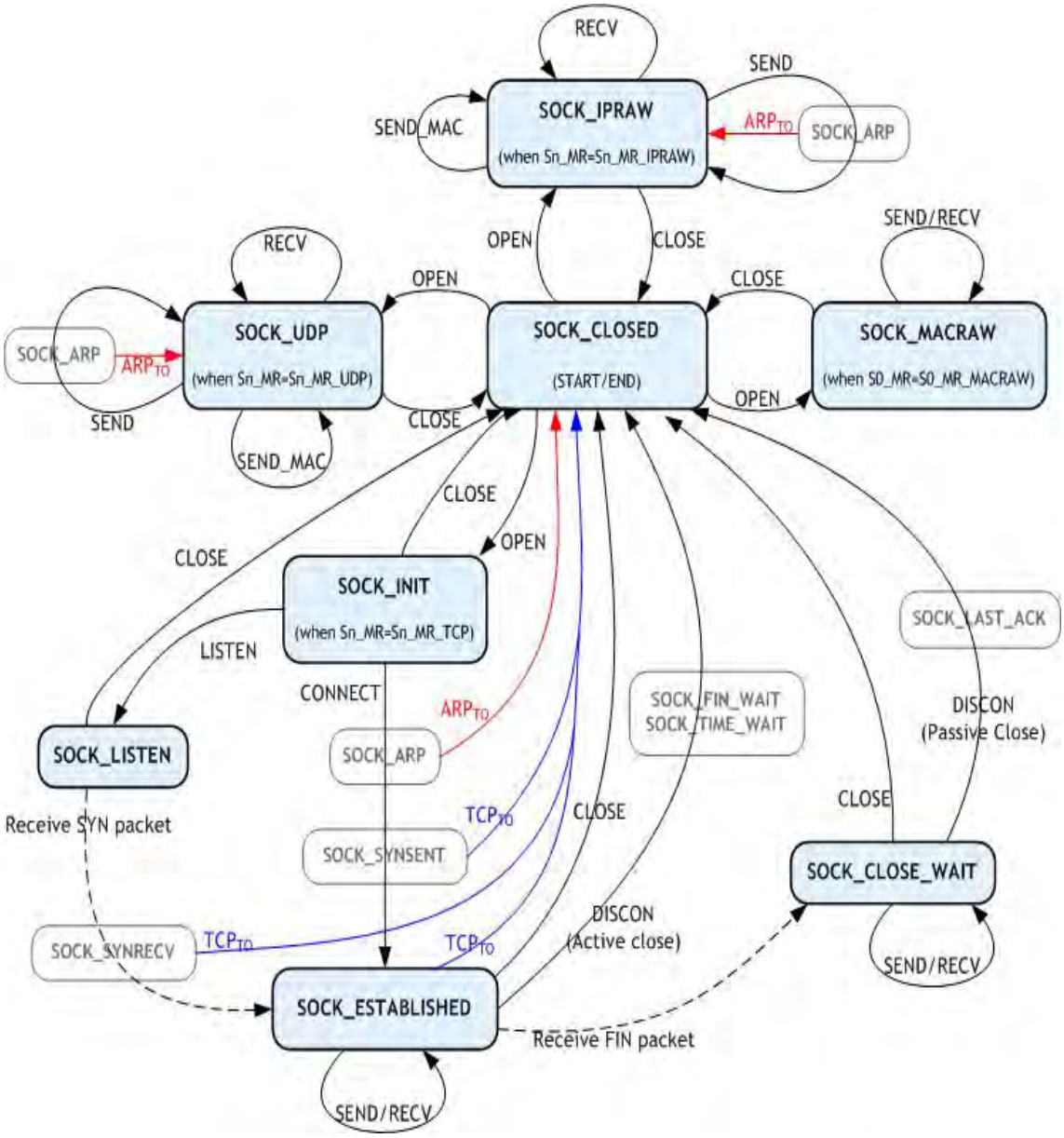


图8.2 SOCKET *n*状态变化

Sn_PORT (SOCKET *n* 端口号寄存器)[R/W][(0xFE4004 + 0x100n) – (0xFE4005 + 0x100n)][0x0000]

设置本机的源端口号。

端口号只有SOCKET工作在TCP、UDP模式下有效，在其它模式下无效。

端口号必须在OPEN命令之前设置。

例：设置SOCKET 0的端口号为5000(0x1388)，配置如下：

0xFE4004	0xFE4005
0x13	0x88

Sn_DHAR (SOCKET *n*目的物理地址寄存器)[R/W][(0xFE4006 + 0x100n) – (0xFE400B + 0x100n)][FF.FF.FF.FF.FF.FF]

它设置、或被设置为SOCKET n的目的端物理地址。如果SOCKET 0用于PPPoE模式，S0_DHAR则设置为PPPoE服务器的物理地址，这个物理地址是已经知道的。

在UDP或IPRAW模式使用SEND_MAC命令时，需要将它设置为SOCKET n的目的端的物理地址。在TCP、UDP、IPRAW模式，使用CONNECT命令或SEND命令时，Sn_DHAR是通过ARP过程获得目的端的物理地址。在成功运行CONNECT或SEND命令后 ， 主机可以通过Sn_DHAR获得目的端的物理地址。

当使用W7100A的PPPoE时，不需要设置PPPoE服务器的物理地址。

然而，即使不用W7100A的PPPoE处理而是使用MACRAW模式，为了传输和接收PPPoE数据包，PPPoE服务器的物理地址（通过PPPoE过程获得）、PPPoE服务器的IP地址、PPP会话的ID值都需要设置，MR（PPPoE）也需要设置为‘1’。

S0_DHAR在OPEN命令之前就设置为PPPoE服务器的物理地址。由S0_DAHAR设置的PPPoE服务器的物理地址在OPEN命令之后应用于PDHAR。

PPPoE的配置信息属于内部信息，即使在CLOSE命令之后仍然有效。

例：SOCKET 0的目的端的物理地址为：00.08.DC.01.02.10，配置如下：

0xFE4006	0xFE4007	0xFE4008	0xFE4009	0xFE400A	0xFE400B
0x00	0x08	0xDC	0x01	0x02	0x10

Sn_DIPR (SOCKET *n*目的IP地址寄存器)[R/W][(0xFE400C + 0x100n) – (0xFE400F + 0x100n)][00.00.00.00]

它设置、或被设置为SOCKET n的目的端IP地址。如果SOCKET 0用于PPPoE模式，S0_DIPR0将设置为已知的PPPoE服务器的IP地址。

只有在TCP、UDP、IPRAW或PPPoE模式下有效，在MACRAW模式下无效。

当SOCKET工作在TCP客户端时，在运行CONNECT命令之前，它必须设置为TCP服务器的IP地址。而当工作在TCP服务器模式时， 当成功建立连接以后， 它内部自动设置为TCP客户端的IP地址。

在UDP或IPRAW模式，为了传输UDP或IPRAW数据包，在使用SEND或SEND_MAC命令之前，必须将目的端的IP地址设置到Sn_DIPR中。

在PPPoE模式，S0_DIPR设置为已知的PPPoE服务器的IP地址。

例：SOCKET 0的目的端的IP地址为：192.168.0.11，设置如下：

0xFE400C	0xFE400D	0xFE400E	0xFE400F
192 (0xC0)	168 (0xA8)	0 (0x00)	11 (0x0B)

Sn_DPORT (SOCKET *n*目的端口号寄存器)[R/W][(0xFE4010 + 0x100n) – (0xFE4011 + 0x100n)][0x0000]

SOCKET n的目的端口号由Sn_DPORT设置。如果SOCKET 0工作在PPPoE模式，

Sn_DPORT0设置为已知的会话ID。

只有在TCP、UDP和PPPoE模式下有效，其它模式下无效。

在TCP客户端模式时，在运行CONNECT命令之前，必须先将Sn_DPORT设置为TCP服务器的侦听端口。

在UDP模式下，在SEND命令或SEND_MAC命令之前，需要先设置好UDP数据包的目的端口号Sn_DPORT。

在PPPoE模式，S0_DPORT设置为已知的PPP会话ID。PPP会话ID在OPEN命令之后应用于PSIDR。

例：设置SOCKET 0的目的端口号为5000(0x1388)，配置如下：

0xFE4010	0xFE4011
0x13	0x88

Sn_MSSR (SOCKET *n*最大分片长度寄存器)[R/W][(0xFE4012 + 0x100n) – (0xFE4013 + 0x100n)][0x0000]

它设置SOCKET n的最大传输单元（MTU），或指示MTU已经设置。

如果主机不设置Sn_MSSR，那么将使用默认的设置。

它支持TCP、或UDP模式。当使用PPPoE（MR(PPPoE)=1），TCP或UDP的最大传输单元（MTU）是由PPPoE的MTU范围确定的。

在IPRAW和MACRAW模式，MTU不是由内部处理的，但是使用默认的MTU。因此当传输的数据比默认的MTU大，主机需要手动进行分片使其在默认的MTU范围内。

在TCP和UDP模式下，如果传输的数据字节数比MTU大，W7100A会自动将数据分片在MTU范围内。

在TCP模式，MTU就是我们所知道的MSS。通过选择主机写入的值和对端的MSS，在TCP连接过程中MSS自动选择较小的值。

在UDP模式下没有TCP的连接过程，只使用主机写入的值。当与不同MTU的对端通信时，W7100A可以接收到ICMP（分片的MTU）数据包。当IR(FMTU)=1时，无法实现与对端的UDP通信。因此用户必须关闭SOCKET，将Sn_MSSR设置为FMTU，然后再试着用OPEN命令打开端口进行通信。

模式	**正常运行**(MR(PPPoE)='0')		**PPPoE** (MR(PPPoE)='1')	
	默认MTU	取值范围	默认MTU	取值范围
TCP	1460	1 ~ 1460	1452	1 ~ 1452
UDP	1472	1 ~ 1472	1464	1 ~ 1464
IPRAW	1480		1472	
MACRAW	1514			

例：SOCKET 0的MSS为1460（0x05B4），配置如下：

0xFE4012	0xFE4013
0x05	0xB4

Sn_PROTO (SOCKET *n*协议号寄存器)[R/W][0xFE4014 + 0x100n][0x00]

这是一个1字节的寄存器，用于设置IP层数据包中IP头的协议号字段。

只有在IPRAW模式下有效，而在其它模式下都无效。Sn_PROTO必须在OPEN命令之前设置。当SOCKET n以IPRAW模式打开时，它发送和接收的是由Sn_PROTO设置的协议号的数据。Sn_PROTO的赋值范围在0x00~0xFF之间。但W7100 A 不支持TCP（0x06）和UDP(0x11)的协议号。

协议号由IANA（Internet assigned numbers authority）定义。详细内容请参考在线信息：http://www.iana.org/assignments/protocol-numbers

例：网络控制信息协议（ICMP）的协议号为0x01，网络分组管理协议的协议号为0x02。

Sn_TOS (SOCKET *n*服务类型寄存器)[R/W][0xFE4015 + 0x100n][0x00]

它设置服务类型（TOS），这是IP层数据包中IP头中的一个字段。必须在OPEN命令之前进行设置。请参考：http://www.iana.org/assignments/ip-parameters.

Sn_TTL (SOCKET *n*生存时间寄存器)[R/W][0xFE4016 + 0x100n][0x80]

它设置IP层数据包中IP头中的生存时间（TTL）字段。必须在OPEN命令之前进行设置。请参考：http://www.iana.org/assignments/ip-parameters

Sn_RXMEM_SIZE (SOCKET *n*接收存储器大小寄存器)[R/W][0xFE401E + 0x100n][0x02]

它用于配置每一个SOCKET的RX存储器的大小。

每个SOCKET的RX存储器大小可配置为1、2、4、8K字节。复位后默认为2K字节。8个SOCKET的Sn_RXMEM_SIZE的总和为16K字节。

例：SOCKET 0：8KB, SOCKET 1：2KB

例1：SOCKET 0 : 8KB, SOCKET 1 : 2KB

0xFE401E	0xFE411E
0x08	0x02

例2：SOCKET 2 : 1KB, SOCKET 3 : 1KB

0xFE421E	0xFE431E
0x01	0x01

例3：SOCKET 4 : 1KB, SOCKET 5 : 1KB

0xFE441E	0xFE451E
0x01	0x01

例4：SOCKET 6 : 1KB, SOCKET 7 : 1KB

0xFE461E	0xFE471E
0x01	0x01

如上例1 ~ 例4所示，SOCKET接收存储器的大小总和（$Sn\_RXMEM\_SIZE_{SUM}$）为16K字节。

Sn_TXMEM_SIZE (SOCKET *n*发送存储器大小寄存器)[R/W][0xFE401F + 0x100n][0x02]

它配置SOCKET的内部TX存储器。每个SOCKET的TX存储器可配置的大小为1、2、4、8K字节。复位后默认为2K字节。8个SOCKET的TX存储器之和$Sn_TXMEM_SIZE_{SUM}$为16K字节。

例5：SOCKET 0 : 4KB, SOCKET 1 : 1KB

0xFE401F	0xFE411F
0x04	0x01

例6：SOCKET 2 : 2KB, SOCKET 3 : 1KB

0xFE421F	0xFE431F
0x02	0x01

例7：SOCKET 4 : 2KB, SOCKET 5 : 2KB

0xFE441F	0xFE451F
0x02	0x02

例8：SOCKET 6 : 2KB, SOCKET 7 : 2KB

0xFE461F	0xFE471F
0x02	0x02

如上例5 ~ 例8所示，SOCKET发送存储器的大小总和 ($Sn\_TXMEM\_SIZE_{SUM}$) 为16K字节。

Sn_TX_FSR (SOCKET n发送存储器剩余空间大小寄存器)[R][(0xFE4020 + 0x100n) – (0xFE4021 + 100n)][0x0000]

它指示SOCKET n的内部TX存储器可使用的空间大小（可写入的传输数据的字节数）。主机不能写入比Sn_TX_FSR更多的数据到TX存储器。因此， 在向TX存储器写入发送数据前检查Sn_TX_FSR，如果要写入的数据字节数小于或等于Sn_TX_FSR，才可以写入数据到TX存储器然后用SEND或SEND_MAC命令发送。

在TCP模式，如果对端接收到所发送的数据包（如果收到从对端来的DATA/ACK数据包），Sn_TX_FSR将自动增加，增加量就是已发送的字节数。在其它模式，只要Sn_IR(SENDOK)=1，Sn_TX_FSR就自动增加，增加量就是传输的数据长度。

例：S0_TX_FSR0的值为2048，

0xFE4020	0xFE4021
0x08	0x00

Sn_TX_RD (SOCKET *n*发送存储器读指针寄存器)[R][(0xFE4022 + 0x100n) – (0xFE4023 + 0x100n)][0x0000]

该寄存器显示TX存储器最后结束传输的地址值 。 对SOCKET n的命令控制寄存器写入SEND命令，它将从当前的Sn_TX_RD地址开始传输数据，直到Sn_WR_WR地址为止。在传输结束后该寄存器的值自动改变。因此在传输结束后，Sn_TX_RD和Sn_TX_WR的值是相等

的。读该寄存器时，先读取它的高字节（0xFE4022, 0xFE4122, 0xFE4222, 0xFE4322, 0xFE4422, 0xFE4522, 0xFE4622, 0xFE4722)，然后再读它的低字节(0xFE4023, 0xFE4123, 0xFE4223, 0xFE4323, 0xFE4423, 0xFE4523, 0xFE4623, 0xFE4723)，这样读取的数据才正确。

Sn_TX_WR (SOCKET *n*发送存储器写指针寄存器)[R/W][(0xFE4024 + 0x100n) – (0xFE4025 + 0x100n)][0x0000]

该寄存器提供定位信息，指示数据应该写入到什么位置。读取该寄存器时，先读取高字节（0xFE4024, 0xFE4124, 0xFE4224, 0xFE4324, 0xFE4424, 0xFE4524, 0xFE4624, 0xFE4724)，然后再读取低字节（0xFE4025, 0xFE4125, 0xFE4225, 0xFE4325, 0xFE4425, 0xFE4525, 0xFE4625,0xFE4725)，这样读取的数据才正确。

例：S0_TX_WR的值为2048(0x0800)。

0xFE4024	0xFE4025
0x08	0x00

但这个值本身不是可以直接访问的物理地址。实际访问的物理地址计算如下：

1. 从Sn_TXMEM_SIZE(n)计算出端口n TX存储器的基地址（SBIUFBASEADDRES(n)）和掩码地址（SMASK(n)），详细内容参看所提供的源代码。

2. 将Sn_TX_WR0和SMASK(n)进行‘位与’运算，其结果就是在端口n的TX存储器范围内的偏移地址（dst_mask）。

3. 将dst_mask和SUBFBASEADDRESS(n)相加得到实际访问的物理地址(dst_ptr)。

现在可以将需要传输的数据写到dst_ptr。（有一种情况需要注意，写入数据时可能会超过端口n的TX存储器的上界。这时将数据写入上边界地址后，再从SBUFBASEADDRESS(n)开始写入剩余的数据，如此循环写入操作。）

操作完成后，Sn_TX_WR的值必须加上当前写入数据的字节数。最后向Sn_CR（端口n的命令寄存器）发出SEND命令。详细信息参考TCP服务器模式下发送数据的源代码。

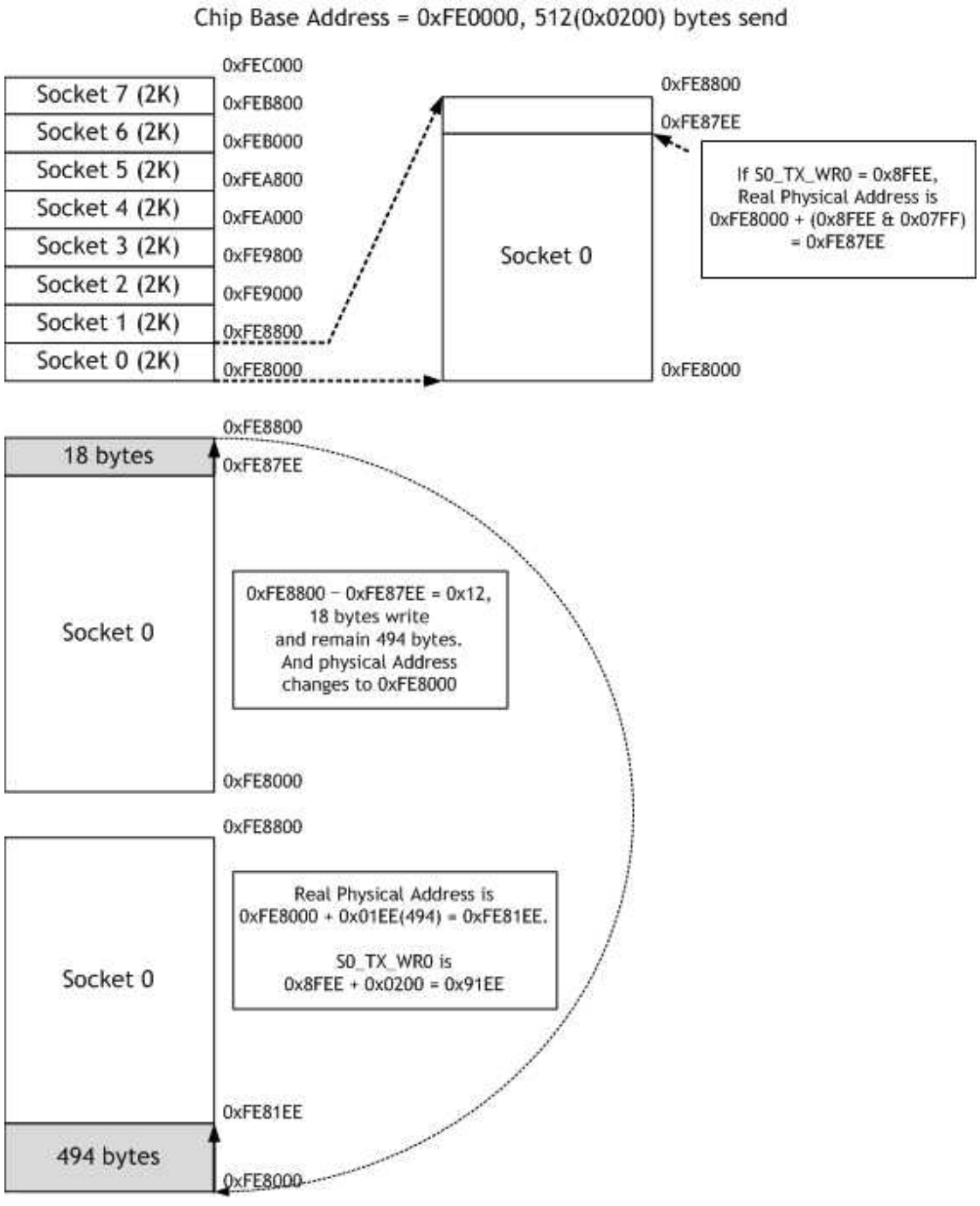


图8.3 计算物理地址

Sn_RX_RSR (SOCKET *n*接收数据长度寄存器)[R][(0xFE4026 + 0x100n) – (0xFE4027 + 0x100n)][0x0000]

它指示端口n内部RX存储器中接收数据的字节数。由于该值是由Sn_Rx_RD和Sn_Rx_WR的在内部计算得出的，对SOCKET n的命令寄存器（Sn_CR）写入RECV命令且接收到远程的数据时，它将自动改变。当读取该寄存器时，用户应该首先读取高字节（0xFE4026, 0xFE4126, 0xFE4226，0xFE4326，0xFE4426，0xFE4526，0xFE4626，0xFE4726），然后再读低字节（ 0xFE4027， 0xFE4127， 0xFE4227， 0xFE4327， 0xFE4427， 0xFE4527， 0xFE4627, 0xFE4727），这样才能够得到正确的值。

例：S0_RX_RSR0的值为2048(0x0800)

0xFE4026	0xFE4027
0x08	0x00

这个值的总长度是由RX存储器大小寄存器决定的。

Sn_RX_RD (SOCKET *n*接收存储器读指针寄存器)[R/W][(0xFE4028 + 0x100n) – (0xFE4029 + 0x100n)][0x0000]

该寄存器确定接收数据的读取地址信息。当读取该寄存器时先读高字节 （ 0xFE4028, 0xFE4128, 0xFE4228, 0xFE4328, 0xFE4428, 0xFE4528, 0xFE4628, 0xFE4728），然后再读低字节（0xFE4029, 0xFE4129, 0xFE4229, 0xFE4329, 0xFE4429, 0xFE4529, 0xFE4629, 0xFE4729），这样读取的信息才正确。

例：S0_RX_RD02048的值为(0x0800)

0x0428	0x0429
0x08	0x00

但这个值不是实际要读取的物理地址。实际的物理地址需要由下面的关系计算获得：

1. 由Sn_RXMEM_SIZE(n)获得端口n的RX存储器的基地址（RBUFBASEADDRESS(n)）和SOCKET n的RX掩码地址（RMASK(n)）。

2. 将Sn_RX_RD0和RMASK(n)进行‘位与’运算，得到端口的RX存储器地址范围内的偏移地址(src_mask)。

3. 将src_mask和RBUFBASEADDRESS(n)相加得到实际要访问的物理地址。

现在可以从src_ptr地址读取接收的数据（有一种特殊情况要注意，读取的地址超过了端口的RX存储器的上边界，这时读完上边界地址的数据后，返回到RBUFBASEADDRESS(n)地址读取剩余的数据，如此循环访问）。

完成操作后，Sn_RX_RD的值必须加上当前读取的字节数（一定不能超过你读取的字节数）。最后对Sn_CR命令寄存器写入RECV命令，即完成操作。

详细信息参考TCP服务器模式下接收数据的源代码。

Sn_RX_WR (SOCKET *n*接收存储器写指针寄存器)[R/W][(0xFE402A + 0x100n) – (0xFE402B + 0x100n)][0x0000]

该寄存器定位接收数据的写入地址信息。当读取该寄存器时先读高地址字节(0xFE402A, 0xFE412A, 0xFE422A, 0xFE432A, 0xFE442A, 0xFE452A, 0xFE462A, 0xFE472A)，然后再读低地址字节 （0xFE402B, 0xFE412B, 0xFE422B, 0xFE432B, 0xFE442B, 0xFE452B, 0xFE462B, 0xFE472B)，这样读取的值才正确。

例：S0_RX_WR0的值为2048(0x0800)。

0xFE402A	0xFE402B
0x08	0x00

Sn_FRAG (SOCKET *n* 分片寄存器)[R/W][(0xFE402D + 0x100n) – (0xFE402E + 0x100n)][0x4000]

它设置IP层的IP头中的分片字段。W7100A不支持IP层的分片。即使配置了Sn_FRAG，IP数据包也不能够分片。它需要在OPEN命令之前设置该寄存器。

例：Sn_FRAG0 = 0x4000（不分片）。

0xFE402D	0xFE402E
0x40	0x00

9 功能描述

因为W7100A内部嵌入一个8051兼容的MCU内核和硬件的TCP/IP内核，它可以不需要其它器件而独立实现以太网的应用。在这一节，将通过一些软件源代码，讲解W7100A的初始化和每一种协议（TCP、UDP、IPRAW、MACRAW）的通信方法。

9.1 初始化

W7100A的初始化分三个步骤：设置8051 MCU，网络信息和内部TX/RX存储器。

- **步骤1：初始化MCU**

1. 中断设置

允许或禁止8051的中断。详细信息参考第3节“中断”。

2. 存储器访问时间设置

通过CKCON (0x8E) 和WTST (0x92) 寄存器来设置存储器的访问时间。CKCON (0x8E) 控制数据存储器的访问时间，而WTSR （0x92） 控制程序存储器的访问时间，设置值为0~7之间。但W7100A的CKCON可以设置的值为0~7，而WTST（0x92）的值只能是4~7，其它值都无效。如果用户设置的值是一个无效值，W7100A将不能够正常工作。详细信息请参考2.4节“SFR的定义”。

例：禁止中断，访问数据存储器2个时钟周期，访问程序存储器7个时钟周期，设置如下：

```
EA = 0;                 // Disable all interrupts
CKCON = 0x01;           // Set data memory access time
WTST = 0x06;            // Set code memory access time
```

3. 串口波特率、寄存器、和中断设置

1) 对串口通信来说，需要设置W7100A的相关的寄存器。

与W7100A的串口通信相关的寄存器有TMOD，PCON和SCON，如下所示。

① TMOD(89H)：决定串口通信的定时器/计数器的工作模式。

GATE	C/T	M_1	M_0	GATE	C/T	M_1	M_0

表9.1 Timer / Counter模式

M_1	M_0	Mode
0	0	0
0	1	1
1	0	2
1	1	3

② PCON(87H):决定串口通信速率的SMOD位。

SMOD	-	-	-	-	-	-	-

表9.2 波特率

Mode	SMOD = '0'	SMOD = '1'
1, 3	Timer/Counter 1时间溢出的一半	Timer/Counter 1时间溢出
2	XTAL的1/4	XTAL的1/2

③ SCON(98H)：控制UART

SM_0	SM_1	SM_2	REN	TB_8	RB_8	TI	RI

表9.3 UART的模式

SM_0	SM_1	Mode
0	0	0
0	1	1
1	0	2
1	1	3

SM_2：在模式2和模式3时使用。假设这位为'1'，如果接收到的第9位为'1'，则接收该数据，如果第9位为'0'，则忽略该数据。

REN：接收允许（'1'允许接收）。

TB_8：在模式2和模式3，发送的第8位数据位。

RB_8：在模式2和模式3，接收的第8位数据位。

TI：发送完成中断标志

RI：接收中断标志

2) 初始化串口通信时必须设置中断状态

因为串口通信使用中断，因此在初始化串口通信时用户必须禁止其它相关的中断。

3) 设置用户使用的波特率。请参考6.6节了解W7100A用于产生波特率的定时器。定时器的波特率计算如下：

用于产生波特率的定时器计算如下：

① 使用timer1的计算公式：

TH1 = 256 – ((K * 88.4736MHz) / (384 * baud rate))

K = '1' at SMOD = '0', K = '2' at SMOD = '1'

② 使用timer2的计算公式：

(RCAP2H, RCAP2L) = 65536 – (88.4736MHz / (32 * baud rate))

例：使用Timer1的模式2，SMOD = 1，时钟频率 = 88.4736MHz，波特率 = 115200

```
ET1= 0;              // Timer1 INT disable
TMOD = 0x20;         // TIMER MODE2
PCON |= 0x80;        // SMOD = 1
TH1 = 0xFC;          // x2 115200(SMOD = 1) at 88.4736MHz
TR1 = 1;             // Start the TIMER1
SCON = 0x50;         // Serial MODE1, REN = 1, TI = 0, RI = 0
ES = 0;              // Serial interrupt disable
RI = 0;              // Receive interrupt disable
TI = 0;              // Transmit interrupt disable
```

4) 如果使用TCPIP内核中断，由于TCPIP内核中断的处理机制，必须设置INTLEVEL寄存器的值大于0x2B00。

例：设置INTLEVEL寄存器的值为0x2B00：

```
IINCHIP_WRITE (INTLEVEL0,  0x2B);  //write high byte of INTLEVEL TCPIPCore register
IINCHIP_WRITE (INTLEVEL0 + 1,  0x00);//write low byte of INTLEVEL TCPIPCore register
```

- **步骤2：设置网络信息**

1. 网络通信的基本信息

必须设置的网络基本信息有：

（1）SHAR（本机物理地址寄存器）

本机物理地址由SHAR设置，必须说明的是在以太网MAC层的物理地址（MAC地址）一定是唯一的。IEEE管理MAC地址的分配。网络设备的制造商给产品分配MAC地址。

物理地址分配到详细信息请参考下面网址：

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

（2）GAR（网关地址寄存器）

（3）SUBR（子网掩码寄存器）

（4）SIPR（本机IP地址寄存器）

2. 设置重发时间和重发次数（数据包发送失败时）

为了设置重发时间，寄存器需要设置如下：

（1）RTR（重发时间寄存器），RTR的‘1’代表‘100us’。

（2）RCR（重发次数寄存器）

- **步骤3：分配SOCKET n内部TX/RX存储器大小**

每一个可配置的TX/RX存储器的最大长度为16K字节，在16K字节的范围内，用户可以将存储器给8个SOCKET任意分配为1K、2K、4K和8K字节。但TX和RX存储器的总合不能超过16K字节（TX_{max} = 16KB, RX_{max} = 16KB）。

In case of, assign 2KB rx, tx memory per SOCKET

{

gS0_RX_BASE = 0xFE0000(Chip base address) + 0xFEC000(Internal RX buffer address); // Set base address of RX memory for SOCKET 0

Sn_RXMEM_SIZE(ch) = (uint8 *) 2; // Assign 2K rx memory per SOCKET

gS0_RX_MASK = 2K – 1; // 0x07FF, for getting offset address within assigned SOCKET 0 RX memory

gS1_RX_BASE = gS0_RX_BASE + (gS0_RX_MASK + 1);

gS1_RX_MASK = 2K – 1;

gS2_RX_BASE = gS1_RX_BASE + (gS1_RX_MASK + 1);

gS2_RX_MASK = 2K – 1;

gS3_RX_BASE = gS2_RX_BASE + (gS2_RX_MASK + 1);

gS3_RX_MASK = 2K – 1;

gS4_RX_BASE = gS3_RX_BASE + (gS3_RX_MASK + 1);

gS4_RX_MASK = 2K – 1;

gS5_RX_BASE = gS4_RX_BASE + (gS4_RX_MASK + 1);

gS5_RX_MASK = 2K – 1;

gS6_RX_BASE = gS5_RX_BASE + (gS5_RX_MASK + 1);

gS6_RX_MASK = 2K – 1;

gS7_RX_BASE = gS6_RX_BASE + (gS6_RX_MASK + 1);

gS7_RX_MASK = 2K – 1;

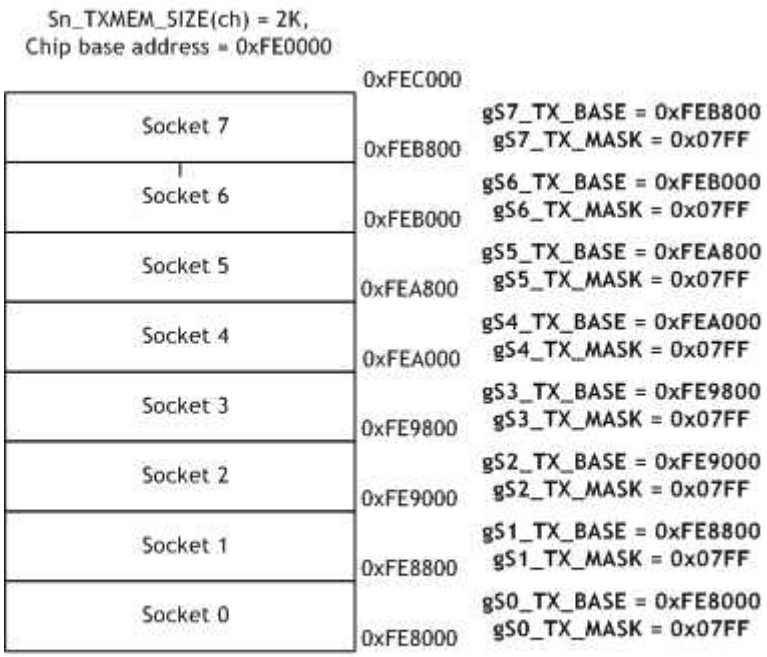
gS0_TX_BASE = 0xFE0000(Chip base address) + 0xFE8000(Internal TX buffer address); // Set base address of TX memory for SOCKET 0

Sn_TXMEM_SIZE(ch) = (uint8 *) 2; // Assign 2K rx memory per SOCKET

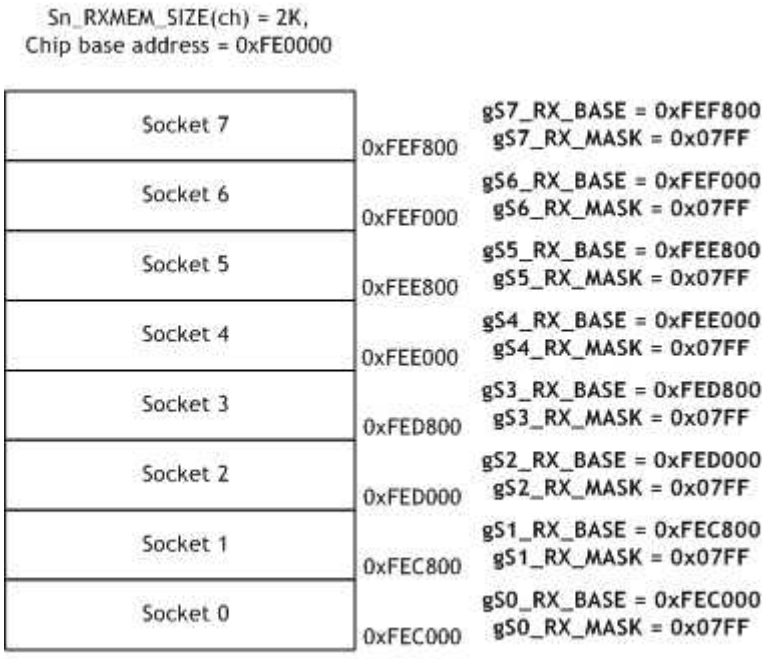
gS0_TX_MASK = 2K – 1;

Same method, set gS1_TX_BASE, gS1_TX_MASK, gS2_TX_BASE, gS2_TX_MASK, gS3_TX_BASE, gS3_TX_MASK, gS4_TX_BASE, gS4_TX_MASK, gS5_TX_BASE, gS5_TX_MASK, gS6_TX_BASE, gS6_tx_MASK, gS7_TX_BASE, gS7_TX_MASK.

}



(a) TX memory allocation



(b) RX memory allocation

图9.1 SOCKET n内部TX/RX存储器的分配

完成W7100A的这三步初始化设置，W7100A就可以通过以太网进行数据传输。此刻，W7100A可以响应来自于网络的Ping请求。

9.2 数据通信

初始化完成以后，W7100A就可以以TCP、UDP、IPRAW或MACRAW的模式打开端口，并发送和接收数据。W7100A支持8个端口以不同的方式同时进行工作。在这一节将介绍每一种方式的通信方法。

9.2.1 TCP

TCP是一种连接通信的协议。TCP使用本机IP地址/端口号和目的IP地址/端口号产生连接。发送和接收数据都是通过这个连接的端口。

连接的方法有“TCP服务器”和“TCP客户端”。它们的区别是谁主动发出连接请求（SYN数据包）。

TCP服务器侦听来自TCP客户端的连接请求，接收发送的连接请求（被动打开），并产生连接。

TCP客户端发出连接请求到TCP服务器（主动打开），并产生连接。

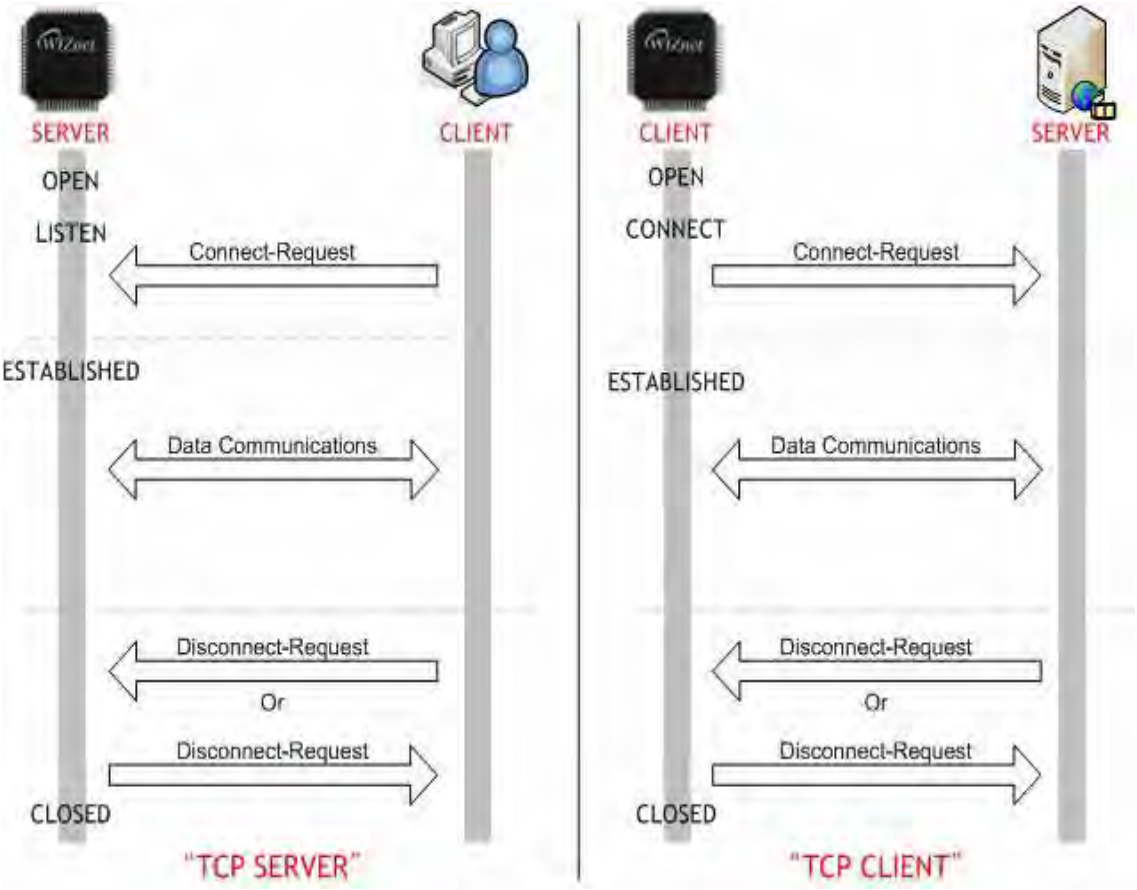


图9.2 TCP服务器和客户端

9.2.1.1 TCP服务器

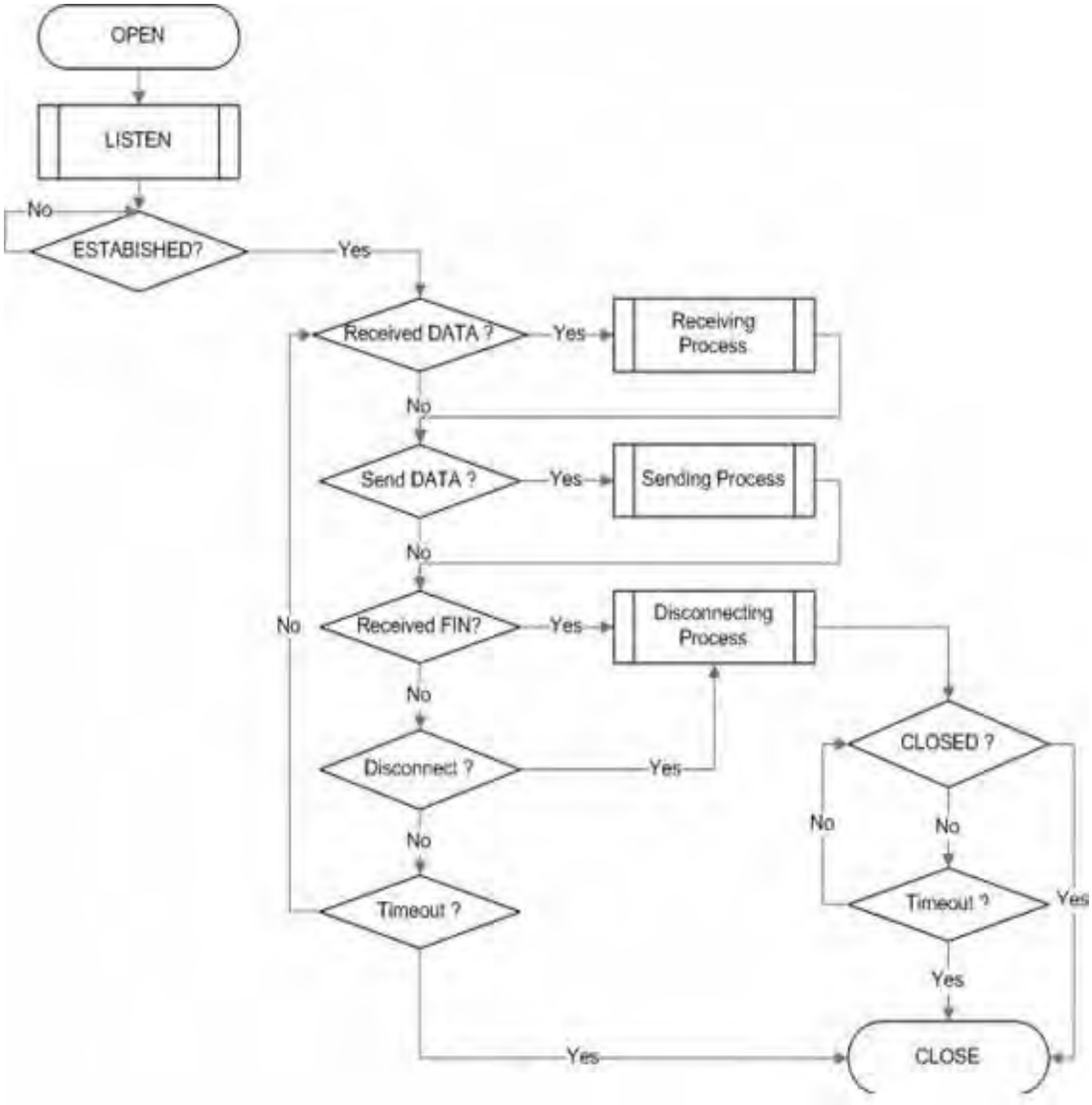


图9.3 TCP服务器操作流程

■ SOCKET初始化

TCP数据通信需要对端口进行初始化设置。初始化过程是首先选择W7100A的一个端口（W7100A有8个端口），设置协议模式（Sn_MR（P3:P0））和设置本机端口号Sn_PORT0（TCP服务器的侦听端口号）。然后运行OPEN指令。执行OPEN命令后如果端口的状态Sn_SR改变为SOCK_INIT，那么端口的初始化就完成了。

TCP服务器和TCP客户端的端口初始化是完全相同的。端口初始化为TCP模式的操作如下：

```
{
START:
  Sn_MR = 0x0001;                    // sets TCP mode
  Sn_PORT = source_port;             // sets source port number
```

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

■ 侦听

运行LISTEN命令设置端口为TCP服务器模式。

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

■ 建立连接

当端口的状态Sn_SR为SOCK_LISTEN时，如果它收到SYN数据包，Sn_SR的状态将改变为SOCK_SYNRECV，并发送一个SYN/ACK数据包，然后端口建立连接。端口建立连接以后才允许进行数据通信。有两种方法可以验证端口是否建立连接。

```
第一种方法：
{
   if (Sn_IR(CON) == '1')   Sn_IR(CON) = '1'; goto ESTABLISHED stage;
   /* In this case, if the interrupt of SOCKET n is activated, interrupt occurs. Refer to IR, IMR Sn_IMR and
      Sn_IR. */
}
```

```
第二种方法：
{
   if (Sn_SR == SOCK_ESTABLISHED) goto ESTABLISHED stage;
}
```

■ 建立连接：检查接收的数据

确定接收到TCP数据

```
第一种方法：
{
   if (Sn_IR(RECV) == '1') Sn_IR(RECV) = '1'; goto Receiving Process stage;
   /* In this case, if the interrupt of SOCKET n is activated, interrupt occurs. Refer to IR, IMR Sn_IMR and
      Sn_IR. */
}
```

```
第二种方法：
{
    if (Sn_RX_RSR != 0x00000000) goto Receiving Process stage;
}
```

第1种方法：当收到数据包时Sn_IR（RECV）=1，如果主机在接收下一个数据包之前没有置上一个Sn_IR（RECV）为‘1’，那么W7100A将不能够识别下一个数据包的Sn_IR（RECV），这是由于上一个Sn_IR（RECV）和后面的Sn_IR（RECV）重叠所致。因此，如果主机不能完全处理每一个Sn_IR（RECV）的数据包，则不推荐使用这种方法。

■ 建立连接：接收数据的过程

在这个过程中，它处理内部RX存储器接收的数据。在TCP模式，如果接收的数据的字节长度超过端口当前RX存储器的剩余空间 ， W7100A不能接收数据。如果发生这种情况 ，W7100A将保持连接（暂停），并等待RX的剩余空间大于需要接收的数据字节长度。

wizmemcpy函数，用于快速处理数据的接收和发送，在W7100A的wizmemcpy.c中定义。更多详细的W7100A的性能信息请参考“W7100A性能的提高”，使用方法请参考“W7100A的驱动指南”。如果用户不使用wizmemcpy函数，用户可以自己定义存储器拷贝函数。

因为W7100A内部有数据存储器和TCPIP内核存储器，用户需要通过地址来区分。因此用户需要把高地址加上‘0xFE’，作为TCPIP内核的存储器，或在将TCPIP内核的存储器数据拷贝到数据存储器中时，将DPX0设置为‘0xFE’。更多的wizmemcpy函数信息请参考“W7100A驱动指南”。

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

    /* if overflow SOCKET RX memory */
    If((src_mask + len) > (gSn_RX_MASK + 1))
    {
       /* copy upper_size bytes of get_start_address to destination_address */
       upper_size = (gSn_RX_MASK + 1) – src_mask;
       wizmemcpy((0xFE0000 + src_ptr), (0x000000 + destination_address), upper_size);
       /* update destination_address */
       destination_address += upper_size;
       /* copy left_size bytes of gSn_RX_BASE to destination_address */
```

```
        left_size = len – upper_size;
        wizmemcpy((0xFE0000 + src_ptr), (0x000000 + destination_address), left_size);
    }
    else
    {
        copy len bytes of src_ptr to destination_address */
        wizmemcpy((0xFE0000 + src_ptr), (0x000000 + destination_address), len);
    }
    /* increase Sn_RX_RD as length of len */
    Sn_RX_RD += len;
    /* set RECV command */
    Sn_CR = RECV;
}
```

注意：wizmemcpy函数在编译链接后必须定位在Flash存储器的0x0800之后，否则将引起运行错误。

■ 建立连接：检查发送的数据/启动发送过程

发送数据的字节长度不能超过端口TX存储器的大小。如果要传输的数据长度大于设置的MSS，它将按照MSS分片传输。

为了下一次数据的传输，用户必须检查上一次的SEND命令是否执行完成。如果上一次的SEND命令还没有执行完成又开始下一次的SEND命令，将会产生错误。数据越多，执行SEND命令所花费的时间越长。因此用户可以适当地将数据分片发送。

在发送过程中，用户必须将‘0xFE’加到地址的高字节，以指向TCPIP内核的存储器。

```
{
    /* first, get the free TX memory size */
FREESIZE:
    freesize = Sn_TX_FSR;
    if (freesize < len) goto FREESIZE;     // len is send size
    /* calculate offset address */
    dst_mask= Sn_TX_WR & gSn_TX_MASK;       // dst_mask is offset address
    /* calculate start address(physical address) */
    dst_ptr = gSn_TX_BASE + dst_mask;       // dst_ptr is physical start address
    /* if overflow SOCKET TX memory */
    if ( (dst_mask + len) > (gSn_TX_MASK + 1) )
    {
        /* copy upper_size bytes of source_addr to dst_ptr */
        upper_size = (gSn_TX_MASK + 1) – dst_mask;
```

```
        wizmemcpy((0x000000 + source_addr), (0xFE0000 + dst_ptr), upper_size);
        /* update source_addr*/
        source_addr += upper_size;
        /* copy left_size bytes of source_addr to gSn_TX_BASE */
        left_size = len – upper_size;
        wizmemcpy((0x000000 + source_addr), (0xFE0000 + gSn_TX_BASE), left_size);
    }
    else
    {/* copy len bytes of source_addr to dst_ptr */
        wizmemcpy((0x000000 + source_addr), (0xFE0000 + dst_ptr), len);
    }
    /* increase Sn_TX_WR as length of len */
    Sn_TX_WR += send_size;
    /* set SEND command */
    Sn_CR = SEND;
}
```

■ 建立连接：检查断开连接的请求（FIN数据包）

检查是否收到断开连接的请求（FIN数据包）。用户可以通过以下两种方法判断断开连接的请求。

```
第一种方法：
{
    if (Sn_IR(DISCON) == ‘1’) Sn_IR(DISCON)=‘1’; goto CLOSED stage;
    /* In this case, if the interrupt of SOCKET n is activated, interrupt occurs. Refer to IR, IMR Sn_IMR and
       Sn_IR. */
}
```

```
第二种方法：
{
    if (Sn_SR == SOCK_CLOSE_WAIT) goto CLOSED stage;
}
```

■ 建立连接：检查断开连接/断开连接的处理

如果用户不需要再进行数据通信，或收到FIN数据包，那么可以断开连接。

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

■ 建立连接：关闭端口

确认端口断开连接、或被DISCON关闭、或被关闭命令关闭。

```
第一种方法：
{
    if (Sn_IR(DISCON) == ‘1’) goto CLOSED stage;
    /* In this case, if the interrupt of SOCKET n is activated, interrupt occurs. Refer to IR, IMR Sn_IMR and
       Sn_IR. */
}
```

```
第二种方法：
{
    if (Sn_SR == SOCK_CLOSED) goto CLOSED stage;
}
```

■ 建立连接：超时

超时可能会发生在连接请求（SYN数据包）或其响应（SYN/ACK数据包）、数据通信或其响应（DATA/ACK数据包）、断开连接请求（FIN数据包）或其响应（FIN/ACK数据包），以及所有其它TCP数据包的传输。如果不能在RTR定义的时间和RCR定义的重发次数范围内完成上述数据包的传输，都将产生TCP的超时，Sn_SR的状态变为SOCK_CLOSED。确定TCP超时的方法如下：

```
第一种方法：
{
    if (Sn_IR(TIMEOUT bit) == ‘1’) Sn_IR(TIMEOUT)=‘1’; goto CLOSED stage;
    /* In this case, if the interrupt of SOCKET n is activated, interrupt occurs. Refer to IR, IMR Sn_IMR and
       Sn_IR. */
}
```

```
第二种方法：
{
    if (Sn_SR == SOCK_CLOSED) goto CLOSED stage;
}
```

■ 关闭端口

用于关闭已经断开连接的端口，或由于TCP超时引起的关闭，或没有断开的过程而是经过主机直接关闭。

```
{/* clear the remained interrupts of SOCKET n*/
    Sn_IR = 0x00FF;
    IR(n) = ‘1’;
    /* set CLOSE command */
    Sn_CR = CLOSE;
}
```

9.2.1.2 TCP客户端

除了‘CONNECT’状态以外，其它与TCP服务器相同，用户可以参考“9.2.1.1节的TCP服务器”。

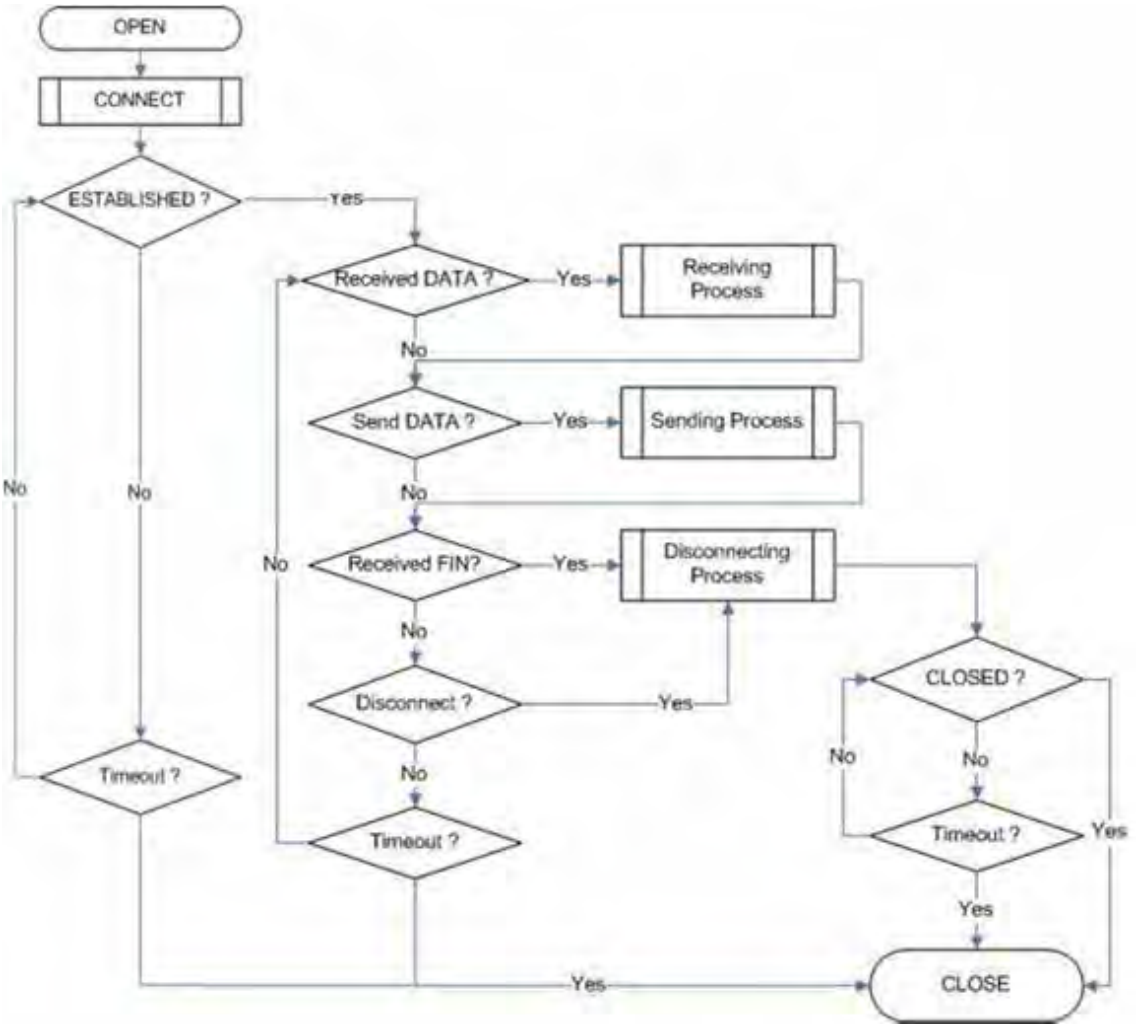


图9.4 TCP客户端操作流程

■ 连接

向TCP服务器发出连接请求（SYN数据包）。在与TCP服务器连接的过程中可能产生ARP超时或TCP超时。

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

9.2.2 UDP

UDP是无连接的通信，它的通信是不需要端口建立连接。TCP是一种连接的、可以保证可靠性的通信协议，但UDP采用数据报文的通信方式，数据传输的可靠性没有保障。但是，因为UDP不使用连接的端口，因此它可以与多个IP地址的多个端口进行数据交换，这是它的优势。使用一个端口与其它端口通信也会带来很多问题， 比如丢失数据、或接收到其它端口来的不需要的数据等。在UDP模式，为了避免出现这些问题，保证数据通信的可靠性，主机需要重发被损坏的数据或丢掉那些不需要的数据。UDP协议支持单播、广播和多播等通信方式。它遵循以下通信流程。

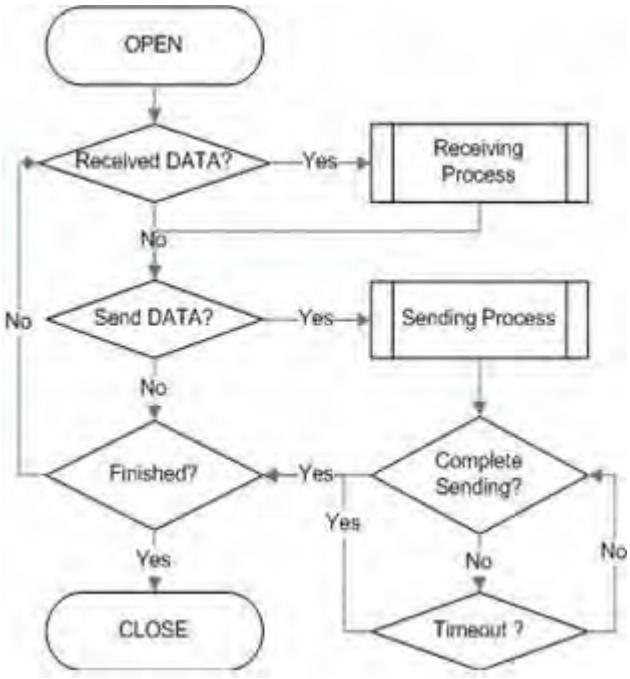


图9.5 UDP操作流程

9.2.2.1单播和广播

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

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

- 怎样建立广播IP？

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

描述	十进制	二进制
HOST IP	222.098.173.123	11011110.01100010.10101101.01111011
Bit Complement Subnet mask	000.000.000.255	00000000.00000000.00000000.11111111

Bitwise OR	-	-
Broadcast IP	222.098.173.255	11011110.01100010.10101101.11111111

■ 端口初始化

要实现UDP通信，端口必须进行初始化设置。打开端口的操作过程如下：首先在W7100A的8个端口中选择一个端口为UDP的工作，并设置为UDP模式（Sn_MR（P3:P0)），然后设置本机端口号（Sn_PORT0），最后运行OPEN命令。执行OPEN命令后，如果端口的Sn_SR状态改变为SOCK_UDP，则完成了端口的初始化设置。

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

■ 检查接收的数据

检查UDP接收数据的方法与TCP相同。参看“9.2.1.1 TCP服务器”中介绍的方法。

第一种方法：

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

第二种方法：

```
{
    if (Sn_RX_RSR != 0x00000000) goto Receiving Process stage;
}
```

■ 接收数据的处理

处理接收的UDP数据，在内部RX存储器中接收的数据包格式如下：

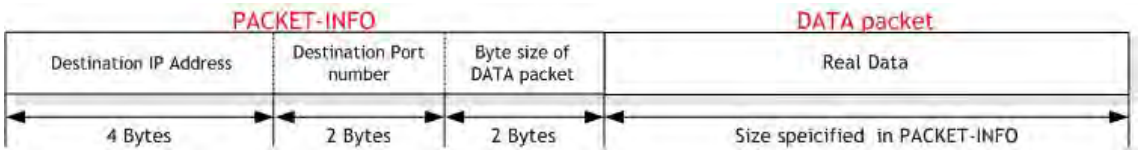


图9.6 接收的UDP数据的格式

接收的UDP数据包含8个字节的数据包信息和有效数据。数据包信息包括两个部分：发送

者的信息（IP地址和端口号）和数据包的长度。UDP可以接收其它的很多UDP数据，用户可以通过发送者的信息区分UDP数据来源。它也接收以“255.255.255.255”的广播地址发送的信息。因此主机可以通过分析发送者的信息，丢掉那些不需要的数据。

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

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

    /* read head information (8 bytes) */
    header_size = 8;
    /* if overflow SOCKET RX memory */
    if ( (src_mask + header_size) > (gSn_RX_MASK + 1) )
    {/* copy upper_size bytes of src_ptr to header_addr */
        upper_size = (gSn_RX_MASK + 1) – src_mask;
        wizmemcpy((0xFE0000 + src_ptr), (0x000000 + 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;
        wizmemcpy((0xFE0000 + gSn_RX_BASE), (0x000000 + header_addr), left_size);
        /* update src_mask */
        src_mask = left_size;
    }
    else
    {/* copy header_size bytes of get_start_address to header_addr */
        wizmemcpy((0xFE0000 + src_ptr), (0x000000 + header_addr), header_size);
        /* update src_mask */
        src_mask += header_size;
    }
    /* update src_ptr */
    src_ptr = gSn_RX_BASE + src_mask;
    /* save remote peer information & received data size */
```

```
    peer_ip = header[0 to 3];
    peer_port = header[4 to 5];
    get_size = header[6 to 7];
    /* if overflow SOCKET RX memory */
    if ( (src_mask + get_size) > (gSn_RX_MASK + 1) )
    {/* copy upper_size bytes of src_ptr to destination_addr */
        upper_size = (gSn_RX_MASK + 1) – src_mask;
        wizmemcpy((0xFE0000 + src_ptr), (0x000000 + 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;
        wizmemcpy((0xFE0000 + gSn_RX_BASE), (0x000000 + destination_addr), left_size);
    }
    else
    {/* copy len bytes of src_ptr to destination_addr */
        wizmemcpy((0xFE0000 + src_ptr), (0x000000 + destination_addr), get_size);
    }
    /* increase Sn_RX_RD as length of len + header_size */
    Sn_RX_RD = Sn_RX_RD + get_size + header_size;
    /* set RECV command */
    Sn_CR = RECV;
}
```

■ 检查发送的数据/发送过程

要发送的数据不能比端口内部TX存储器定义的空间大。但如果比MTU大，他将根据MTU自动进行分片然后再发送。

如果用户希望广播发送，则将Sn_DIPR0设置为“255.255.255.255”。

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

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

■ 检查发送完成/超时

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

```
第一种方法：
{/* check SEND command completion */
   while(Sn_IR(SENDOK)==‘0’)    /* wait interrupt of SEND completion */
   {/* check ARPTO */
      if (Sn_IR(TIMEOUT)==‘1’) Sn_IR(TIMEOUT)=‘1’; goto Next stage;
   }
   Sn_IR(SENDOK) = ‘1’; /* clear previous interrupt of SEND completion */
}
```

```
第一种方法：
{
```

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

}
```

■ 检查通信结束/关闭端口

如果不需要再进行数据通信，则将端口关闭

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

9.2.2.2 多播

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

分组硬件地址的选择范围在“01:00:5e:00:00:00”到“01:00:5e:7f:ff:ff”之间，而分组IP地址则使用D类地址， 范围从“ 224.0.0.0 ”到“ 239.255.255.255 ”。详细内容请参考官方网站 ：http://www.iana.org/assignments/multicast-addresses。在选择时， 6个字节的高23位硬件地址和4个字节的组IP地址必须相同。例如，如果用户选择的组IP地址为“244.1.1.11”，那么组硬件地址为“01:00:5e:01:01:0b”。详细信息请参考RFC1112：http://www.ietf.org/rfc.html。

在W7100A内部，IGMP处理多播注册是由内部自动完成的。当用户以多播的模式打开端口时，“JOIN”信息将在内部自动传送。如果用户关闭端口，“LEAVE”信息将在内部自动传送。端口打开以后，“REPORT”信息将在数据传输过程中每隔一定的时间传送。

W7100A支持IGMP v1和v2版本。如果用户想使用一个升级的版本，主机可以使用IPRAW模式直接处理IGMP。

■ 端口初始化

在W7100 A 的8个端口中选择一个 端 口 作为多播通信的工作端口。设置Sn_DHAR0为多播组硬件地址，设置Sn_DIPR0为多播组IP地址。然后设置Sn_PORT0和Sn_DPORT0为多播组端口号，将Sn_MR (P3:P0) 设置为UDP，且将Sn_MR (MULTI) 设置为‘1’。最后运行OPEN

命令。如果Sn_SR的状态在执行完OPEN命令后改变为SOCK_UDP，端口的初始化即完成。

```
{
START:
    /* set Multicast-Group information */
    Sn_DHAR0 = 0x01;          /* set Multicast-Group H/W address(01:00:5e:01:01:0b) */
    Sn_DHAR1 = 0x00;
    Sn_DHAR2 = 0x5E;
    Sn_DHAR3 = 0x01;
    Sn_DHAR4 = 0x01;
    Sn_DHAR5 = 0x0B;
    Sn_DIPR0 = 211;           /* set Multicast-Group IP address(211.1.1.11) */
    Sn_DIPR1 = 1;
    Sn_DIPR2 = 1;
    Sn_DIRP3 = 11;
    Sn_DPORT = 0x0BB8;        /* set Multicast-Group Port number(3000) */
    Sn_PORT = 0x0BB8;         /* set Source Port number(3000) */
    Sn_MR = 0x02 | 0x80; /* set UDP mode & Multicast on SOCKET n Mode Register */
    Sn_CR = OPEN;             /* set OPEN command */
    /* wait until Sn_SR is changed to SOCK_UDP */
    if (Sn_SR != SOCK_UDP) Sn_CR = CLOSE; goto START;
}
```

- 检查接收的数据

  参考“9.2.2.1 单播和广播”

- 接收数据

  参考“9.2.2.1 单播和广播”。

- 检查发送/发送过程

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

```
{/* first, get the free TX memory size */
FREESIZE:
    freesize = Sn_TX_FSR;
    if (freesize < len) goto FREESIZE;      // len is send size
    /* calculate offset address */
    dst_mask = Sn_TX_WR & gSn_TX_MASK;      // dst_mask is offset address
```

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

■ 检查发送完成/超时

因为主机管理所有的数据通讯协议的处理过程，因此不会出现超时的现象

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

■ 完成操作/关闭端口

参考“9.2.2.1 单播和广播”。

9.2.3 IPRAW

IPRAW属于IP层的数据通信，它是比TCP、UDP低一层协议。IPRAW支持IP层的协议，如ICMP（0x01）和IGMP（0x02），由协议号决定。ICMP的‘PING’功能和IGMP v1/v2已经在W7100A中由硬件实现。如果用户需要，主机可以将SOCKET n以IPRAW的模式打开，直接

处理IPRAW的数据。在使用IPRAW模式时，用户必须设置IP头中用户所使用的协议号。协议号由IANA定义，请参考官方网站：http://www.iana.org/assignments/protocol-numbers。在打开端口之前，必须由Sn_PROTO先定义协议号。W7100A在IPRAW模式下不支持TCP（0x06）和UDP（0x11）。IPRAW模式的端口通信只支持设定的协议号通信。ICMP端口不能接收非设定的协议数据，如IGMP。

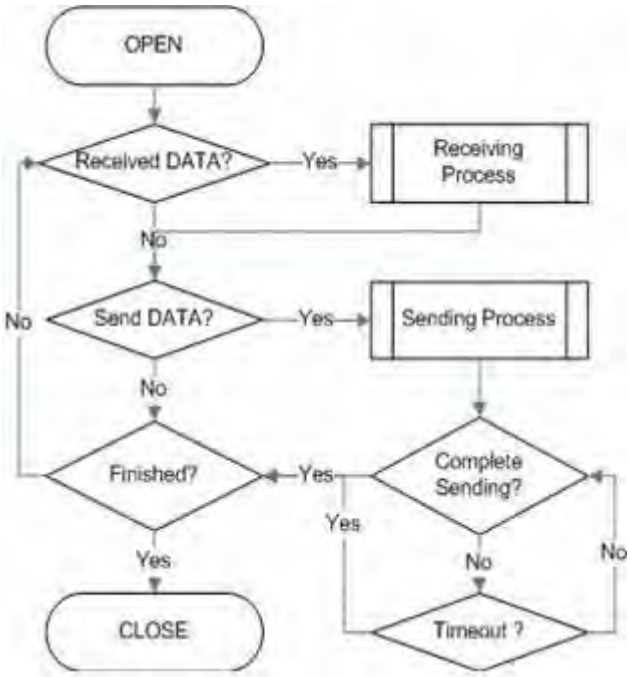


图9.7 IPRAW操作流程

■ 初始化端口

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

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

■ 检查接收的数据

参考“9.2.2.1 单播和广播”。

■ 接收数据

处理内部RX存储器的IPRAW数据。IPRAW的数据结构如下所示。

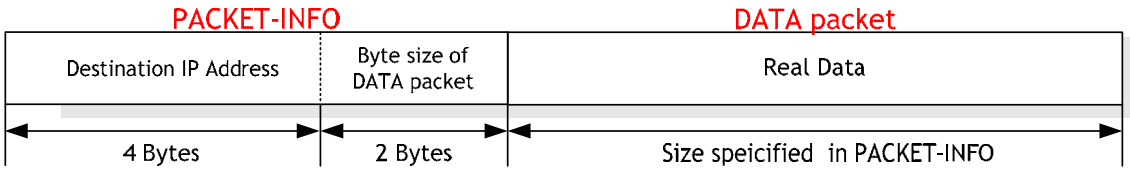


图9.8 IPRAW接收的数据结构

IPRAW的数据包括6个字节的数据包信息和数据。数据包信息包括：发送者信息（IP地址）和数据长度信息。IPRAW的数据包与UDP数据包结构相同，只是IPRAW数据包中没有端口号的信息。请参考“9.9.9.1 单播和广播”。

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

■ 检查发送的数据/发送处理

用户要发送的数据长度不能比内部TX存储器大，也不能超过默认的MTU。IPRAW数据传输与UDP的数据传输相同，只是没有目标端口号。参考“9.2.2.1 单播和广播”

■ 完成发送/超时

与UDP相同，参考“9.2.2 UDP”。

■ 操作完成/关闭端口

与UDP相同，参考“9.2.2 UDP”。

9.2.4 MACRAW

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

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

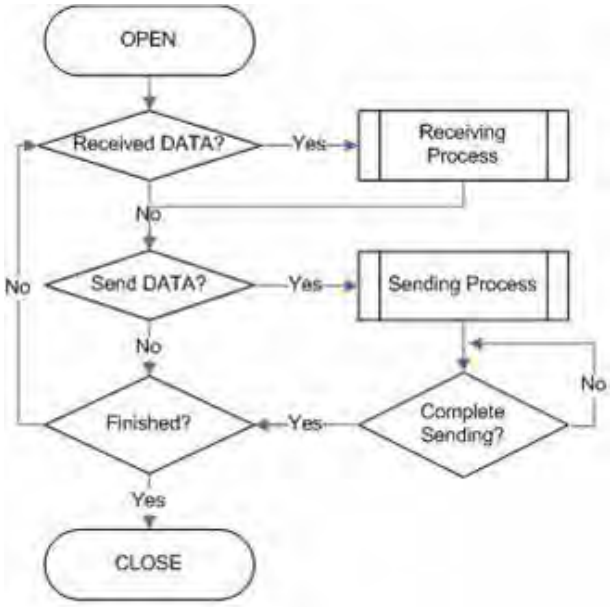


图9.9 MACRAW操作流程

- 端口初始化

选择一个端口，设置Sn_MR（P3:P0）为MACRAW模式，然后运行“OPEN”命令。执行“OPEN”命令后，Sn_SR的状态改变为“SOCK_MACRAW”，则完成端口初始化。因为所有的通信信息（本机物理地址、IP地址和端口号，目的物理地址、IP地址和端口号）都是“MACRAW”数据，因此不需要设置更多的寄存器。

```
{
START:
    /* sets MAC raw mode */
    S0_MR = 0x04;
    /* sets OPEN command */
    S0_CR = OPEN;
    /* wait until Sn_SR is changed to SOCK_MACRAW */
    if (S0_SR != SOCK_MACRAW) S0_CR = CLOSE; goto START;
}
```

- 检查接收的数据

参考“9.2.2.1 单播和广播”。

- 接收处理

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

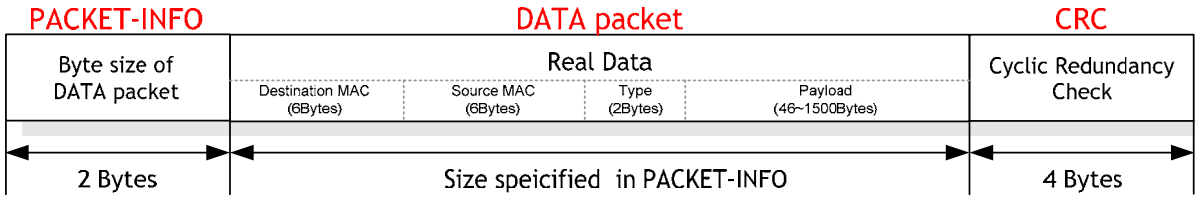


图9.10 MACRAW接收的数据结构

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

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

```
{    /* calculate offset address */
    src_mask = S0_RX_RD & gS0_RX_MASK;       // src_mask is offset address
    /* calculate start address(physical address) */
    src_ptr = gS0_RX_BASE + src_mask;       // src_ptr is physical start address
    /* get the size of packet */
    len = get_Byte_sizeof_DATA_packet();       // Read the 2bytes PACKET-INFO
    /* if overflow SOCKET RX memory */
    If((src_mask + len) > (gS0_RX_MASK + 1))
    {   /* copy upper_size bytes of get_start_address to destination_address */
        upper_size = (gS0_RX_MASK + 1) – src_mask;
        wizmemcpy((0xFE0000 + src_ptr), (0x000000 + destination_address), upper_size);
        /* update destination_address */
        destination_address += upper_size;
        /* copy left_size bytes of gSn_RX_BASE to destination_address */
        left_size = len – upper_size;
        wizmemcpy((0xFE0000 + src_ptr), (0x000000 + destination_address), left_size);
    }
    else
    {   /* copy len bytes of src_ptr to destination_address */
        wizmemcpy((0xFE0000 + src_ptr), (0x000000 + destination_address), len);
    }
    /*    increase Sn_RX_RD as length of len */
    S0_RX_RD += len;
    /* extract 4 bytes CRC from internal RX memory and then ignore it */
    wizmemcpy((0xFE0000 + src_ptr), (0x000000 + dummy), len);
    /* set RECV command */
```

```
    S0_CR = RECV;
}
```

注意：

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

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

在下面的端口初始化的例程中设置S0_MR的MF为“1”，

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

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

```
{/* check the free size of internal RX memory */
   if((S0_RXMEM_SIZE(0) * 1024) – S0_RX_RSR(0) < 1528)
   {
        received_size = S0_RX_RSR(0);  /* backup Sn_RX_RSR */
        S0_CR = CLOSE;                      /* SOCKET Closed */
        while(S0_SR != SOCK_CLOSED);              /* wait until SOCKET is closed */
        /* process all data remained in internal RX memory */
        while(recved_size > 0)
        {/* calculate offset address */
              src_mask = S0_RX_RD & gS0_RX_MASK;     // src_mask is offset address
              /* calculate start address(physical address) */
              src_ptr = gS0_RX_BASE + src_mask;    // src_ptr is physical start address
              /* if overflow SOCKET RX memory */
              If((src_mask + len) > (gS0_RX_MASK + 1))
```

```
            {/* copy upper_size bytes of get_start_address to destination_address */
               upper_size = (gS0_RX_MASK + 1) – src_mask;
            wizmemcpy((0xFE0000 + src_ptr), (0x000000 + destination_address), upper_size);
               /* update destination_address */
               destination_address += upper_size;
               /* copy left_size bytes of gSn_RX_BASE to destination_address */
               left_size = len – upper_size;
               wizmemcpy((0xFE0000 + src_ptr), (0x000000 + destination_address), left_size);
            }
            else
            {   /* copy len bytes of src_ptr to destination_address */
               wizmemcpy((0xFE0000 + src_ptr), (0x000000 + destination_address), len);
            }
            /* increase Sn_RX_RD as length of len */
            S0_RX_RD += len;
            /* extract 4 bytes CRC from internal RX memory and then ignore it */
            wizmemcpy((0xFE0000 + src_ptr), (0x000000 + dummy), len);
            /* calculate the size of remained data in internal RX memory*/
            recved_size = recved_size – 2 – len – 4;
         }
         /* Reopen the SOCKET */
         /* sets MAC raw mode with enabling MAC filter */
         S0_MR = 0x44; /* or S0_MR = 0x04 */
         /* sets OPEN command */
         S0_CR = OPEN;
         /* wait until Sn_SR is changed to SOCK_MACRAW */
         while (S0_SR != SOCK_MACRAW);
      }
      else /* process normally the DATA packet from internal RX memory */
      {/* This block is same as the code of “Receiving process” stage*/
      }
}
```

■ 检查发送数据/处理发送过程

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

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

    /* if overflow SOCKET TX memory */
    if ( (dst_mask + len) > (gSn_TX_MASK + 1) )
    {/* copy upper_size bytes of source_addr to dst_ptr */
        upper_size = (gSn_TX_MASK + 1) – dst_mask;
        wizmemcpy((0x000000 + source_addr), (0xFE0000 + dst_ptr), upper_size);
        /* update source_addr*/
        source_addr += upper_size;
        /* copy left_size bytes of source_addr to gSn_TX_BASE */
        left_size = len – upper_size;
        wizmemcpy((0x000000 + source_addr), (0xFE0000 + gS0_TX_BASE), left_size);
    }
    else
    {/* copy len bytes of source_addr to dst_ptr */
        wizmemcpy((0x000000 + source_addr), (0xFE0000 + dst_ptr), len);
    }
    /* increase Sn_TX_WR as length of len */
    S0_TX_WR += send_size;
    /* set SEND command */
    S0_CR = SEND;
}
```

■ 检查发送是否完成

因为主机管理所有协议处理进行数据通信，因此不会发生超时的现象

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

■ 检查操作是否结束/关闭端口

参考“9.2.2.1 单播和广播”。

10 电气特性

10.1 极限参数

符号	参数	取值范围	单位
V_{DD}	直流供电电压	-0.5 to 3.6	V
V_{IN}	直流输入电压	-0.5 to 5.5 (5V tolerant)	V
V_{OUT}	直流输出电压	0 to 3.3 (GPIO)	V
		-0.5 to 3.6 (Others)	
I_{IN}	直流输入电流	±5	mA
I_{OUT}	直流输出电流	2 to 8	mA
T_{OP}	工作温度	-40 to 80	°C
T_{STG}	储藏温度	-55 to 125	°C

说明：器件工作在极限条件下将会造成永久损坏

10.2 直流参数

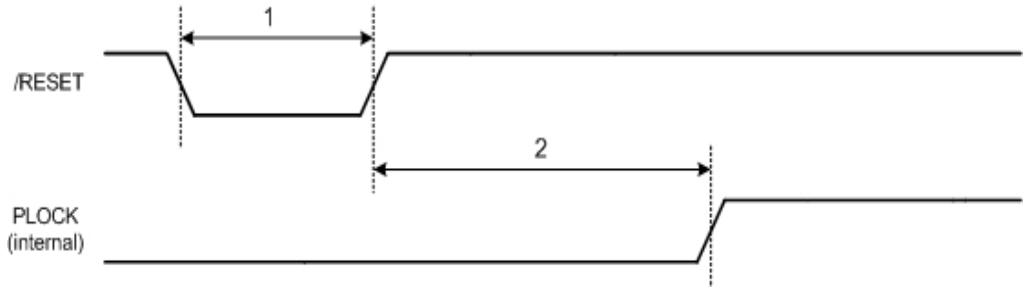
符号	参数	测试条件	最小	典型	最大	单位
V_{DD}	直流供电电压	节点温度从-55°C到 125°C	3.0	3.3	3.6	V
V_{IH}	高电平输入电压		2.0		5.5	V
V_{IL}	低电平输入电压		- 0.5		0.8	V
V_{OH}	高电平输出电压	I_{OH} = 2 ~ 16 mA	2.4			V
V_{OL}	低电平输出电压	I_{OL} = -2 ~ -12 mA			0.4	V
I_I	输入电流	$V_{IN} = V_{DD}$			±5	μA
I_O	输出电流	$V_{OUT} = V_{DD}$	2		8	mA

10.3 功耗(工作电压3.3V)

符号	参数	测试条件	最大	单位
I_{Boot}	电流消耗	运行内部Boot	250	mA
I_{Idle}	电流消耗	休眠状态	220	mA
I_{Active}	电流消耗	全部8个SOCKET工作	220	mA
$I_{Power-down}$	电流消耗	低功耗模式	108	mA

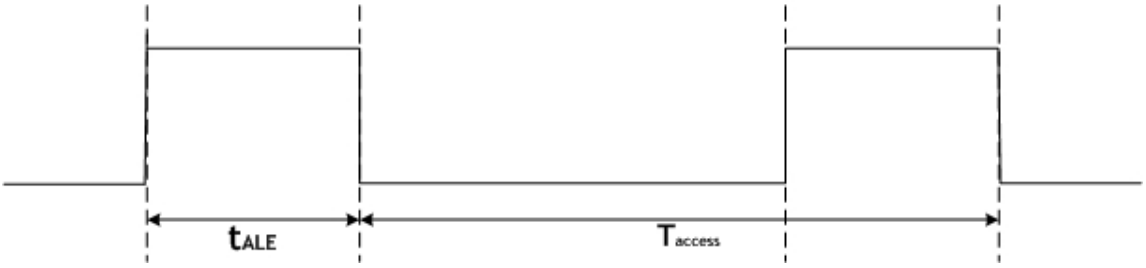
10.4 交流特性

复位时序



描述		Min	Max
1	Reset Cycle Time	2 us	-
2	PLL Lock-in Time	50 us	10 ms

外部存储器访问时序



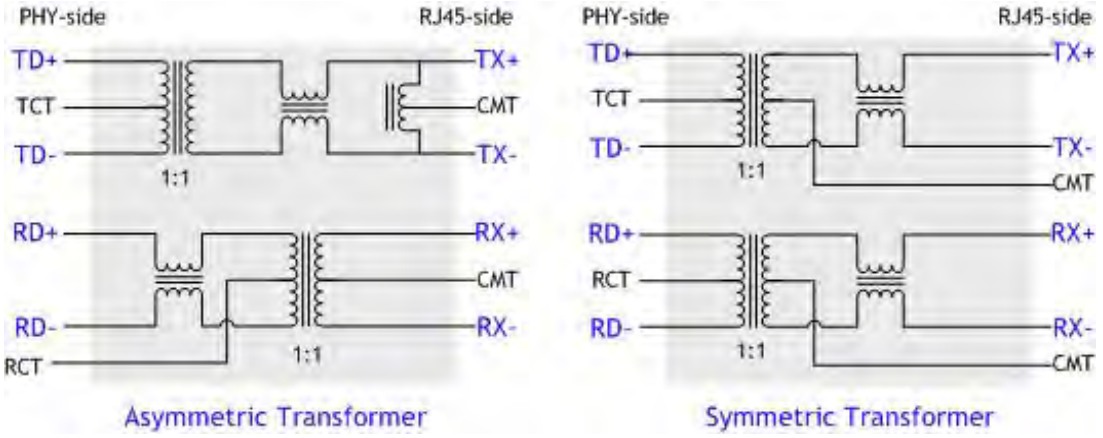
描述		Min	Max
t_{ALE}	ALE signal duration = ALECON register value + 1 clock (clock speed = 88.4736MHz)	1 clock	256 clock
T_{access}	External memory access period = 3us + EXTWTST register value	3 us	744us

10.5 晶体特性参数

参数	范围
频率	25 MHz
频率偏差 (25℃环境)	±30 ppm
电容偏移	7pF （最大）
驱动能力	1 ~ 500uW (典型值100uW)
负载电容	18pF
老化 (25℃环境)	±3ppm / year Max

10.6 变压器特性参数

参数	发送端	接收端
变比	1:1	1:1
电感	350 uH	350 uH



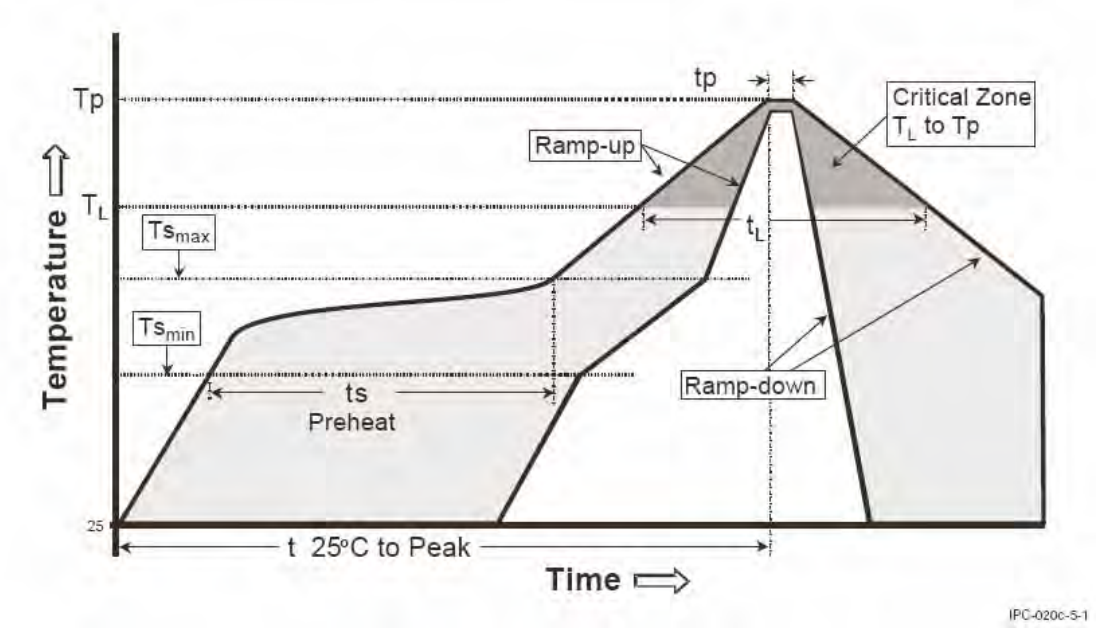
在使用内部PHY模式时，一定要使用对称的变压器，以便可以支持自动MDI/MDIX（交叉）。
使用外部PHY模式时，要根据外部PHY的特性选择变压器。

11 IR Reflow Temperature Profile (Lead-Free)

Moisture Sensitivity Level: 3

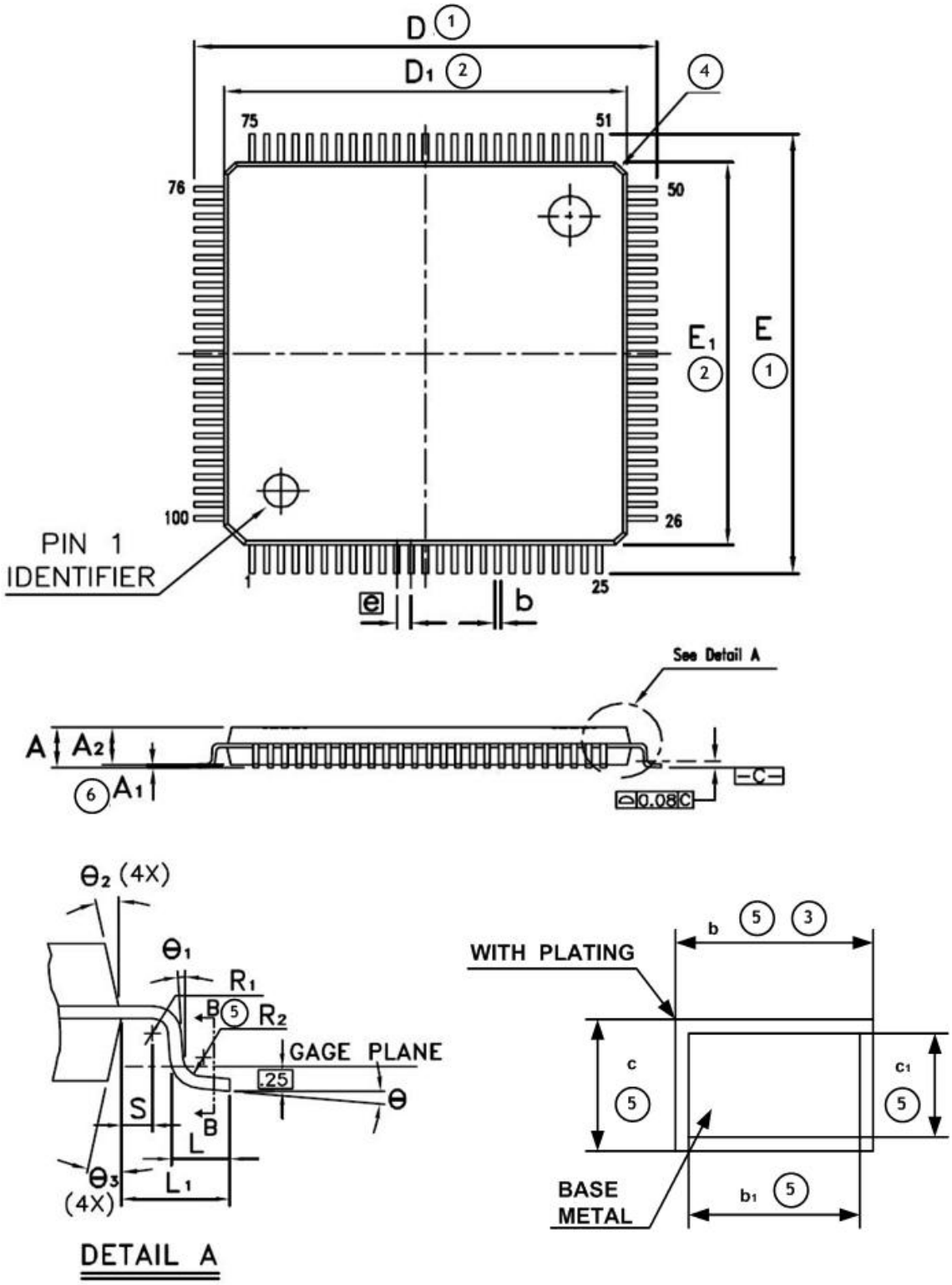
Dry Pack Required: Yes

Average RAMp-Up Rate (Ts_{max} to Tp)	3° C/second max.
Preheat	
– Temperature Min (Ts_{min})	150 °C
– Temperature Max (Ts_{max})	200 °C
– Time (ts_{min} to ts_{max})	60-180 seconds
Time maintained above:	
– Temperature (TL)	217 °C
– Time (tL)	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.



12 封装

12.1 LQFP 100封装尺寸



符号	毫米			英寸		
	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.
A	-	-	1.60	-	-	0.063
A_1	0.05	-	0.15	0.002	-	0.006
A_2	1.35	1.40	1.45	0.053	0.055	0.057
b	0.17	0.22	0.27	0.007	0.009	0.011
b_1	0.17	0.20	0.23	0.007	0.008	0.009
c	0.09	-	0.20	0.004	-	0.008
c_1	0.09	-	0.16	0.004	-	0.006
D	15.85	16.00	16.15	0.624	0.630	0.636
D_1	13.90	14.00	14.10	0.547	0.551	0.555
E	15.85	16.00	16.15	0.624	0.630	0.636
E_1	13.90	14.00	14.10	0.547	0.551	0.555
e		0.50 BSC			0.020 BSC	
L	0.45	0.60	0.75	0.018	0.024	0.030
L_1		1.00 REF			0.039 REF	
R_1	0.08	-	-	0.003	-	-
R_2	0.08	-	0.20	0.003	-	0.008
S	0.20	-	-	0.008	-	-
θ	0°	3.5°	7°	0°	3.5°	7°
θ_1	0°	-	-	0°	-	-
θ_2		12° TYP			12° TYP	
θ_3		12° TYP			12° TYP	

Note :

① To be determined at seating plane – C -.

② Dimensions 'D_1' and 'E_1' do not include mold protrusion. D_1' and 'E_1' are maximum plastic body size dimensions including mold mismatch.

③ Dimension 'b' does not include dambar protrusion. Dambar cannot be located on the lower radius or the foot.

④ Exact shape of each corner is optional

⑤ These Dimensions apply to the flat section of the lead between 0.10mm and 0.25mm from the lead tip.

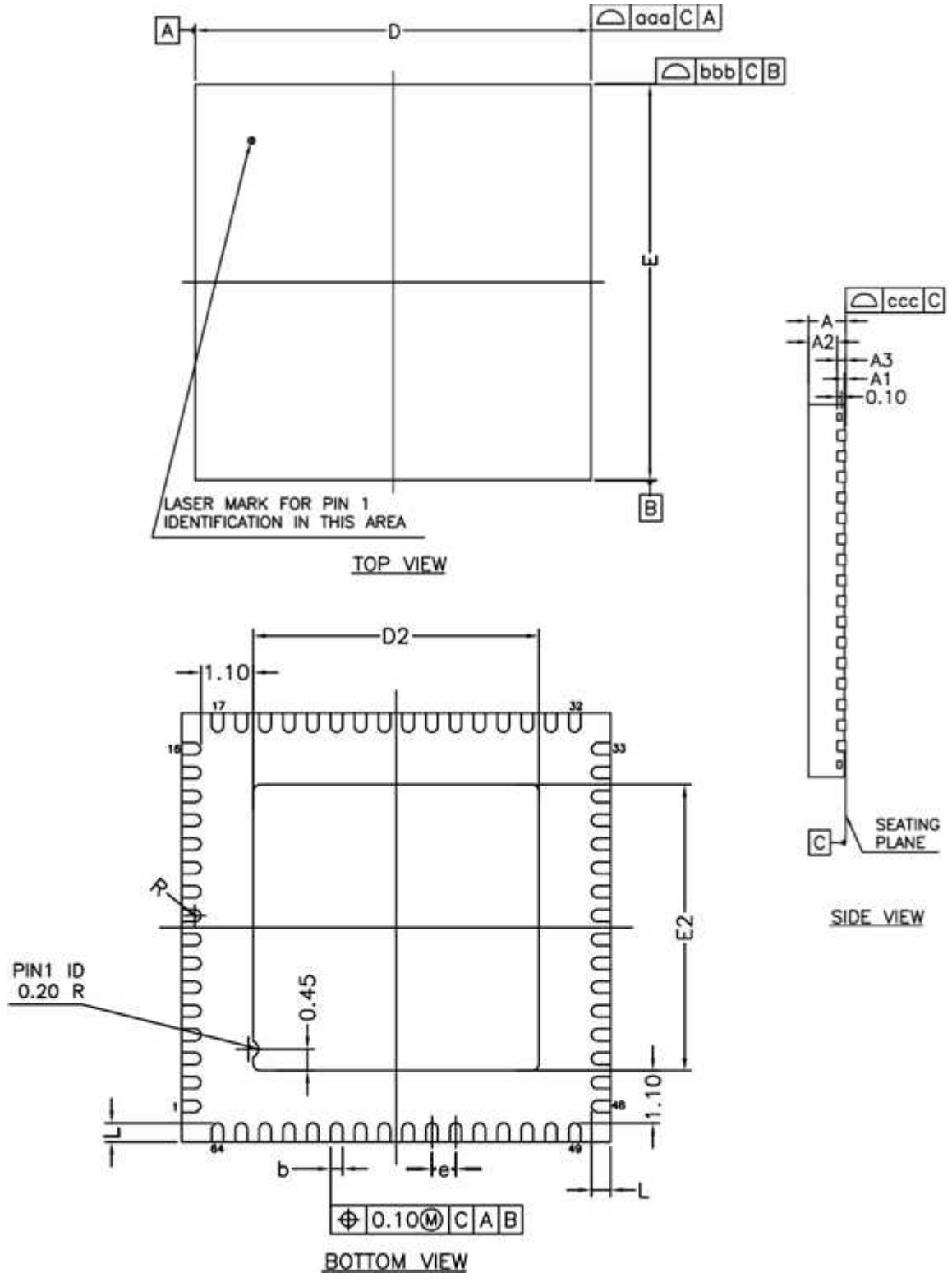
⑥ A_1 is defined as the distance from the seating plane to the lowest point of the package body.

⑦ Controlling dimension : Millimeter

⑧ Reference Document : JEDEC MS-026 , BED.

12.2 QFN 64封装尺寸

单位：mm



符号	毫米			英寸		
	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.
A	-	-	0.90	-	-	0.035
A1	-	-	0.05	-	-	0.002
A2	-	0.65	0.70	-	0.026	0.028
A3		0.200 REF.			0.008 REF.	
b	0.18	0.25	0.30	0.007	0.010	0.012
D	15.85	16.00	16.15	0.624	0.630	0.636
D		9.00 bsc			0.354 bsc	
D2	5.90	6.00	6.10	0.232	0.236	0.240
E		9.00 bsc			0.354 bsc	
E2	5.90	6.00	6.10	0.232	0.236	0.240
L	0.35	0.40	0.45	0.014	0.016	0.018
e		0.50 bsc			0.020 bsc	
R	0.09	-	-	0.004	-	-
TOLERANCES OF FORM AND POSITION						
aaa		0.10			0.004	
bbb		0.10			0.004	
ccc		0.05			0.002	

Note:

① All dimensions are in millimeters.

② Die thickness allowable is 0.305mm maximum (0.012 inches maximum).

③ Dimensioning & tolerances conform to ASME Y14.5M. -1994.

④ The pin #1 identifier must be placed on the top surface of the package by using indentation mark of other feature of package body.

⑤ Exact shape and size of this feature is optional.

⑥ Package WARPAGE max 0.08mm.

⑦ Applied for exposed pad and terminals, exclude embedding part of exposed pad from measuring.

⑧ Applied to terminals.

13 Appendix:Performance Improvement about W7100A

This section presents the benefits gained about calculation by using W7100A over standard 8051 family.

13.1 Summary

The 8-bit operation cycles of the 80C51 and W7100A with addition, subtraction, multiplication and division are as below. It is briefly shows its performance. The W7100A with 'wizmemcpy' (supported by WIZnet) function is almost 9 times faster than the 80C51.

	80C51 cycle	**W7100A** cycle / with user code	**W7100A** cycle / with wizmemcpy
ADD	36	12	**4**
SUB	36	12	**4**
MUL	96	12	**6**
DIV	96	20	**10**

In the succeeded section shows more detail performance.

13.2 8–Bit Arithmetic Functions

13.2.1 Addition

- Immediate data

The following code performs immediate data (constant) addition to an 8-bit register.

RX = RX + #*n*

Mnemonic		Opcode	Byte	80C51 Cycle	W7100A Cycle	
					ISP / wizmemcpy	FLASH / user code
MOV	A, Rx	E8h – EFh	1	12	1	4
ADD	A, #*n*	24h	2	12	2	4
MOV	Rx, A	F8h – FFh	1	12	1	4
Sum :				36	4	12

Note. 'wizmemcpy' function are built-in inside Boot ROM in W7100A. Refer to the 'Driver Guide.'

- Direct addressing

The following code performs direct addressing addition to an 8-bit register.

Rx = Rx + (dir)

Mnemonic		Opcode	Byte	80C51 Cycle	W7100A Cycle	
					ISP / wizmemcpy	FLASH / user code
MOV	A, Rx	E8h – EFh	1	12	1	4
ADD	A, dir	25h	2	12	2	4
MOV	Rx, A	F8h – FFh	1	12	1	4
		Sum :		36	4	12

■ Indirect addressing

The following code performs indirect addressing addition to an 8-bit register.

Rx = Rx + (@Rx)

Mnemonic		Opcode	Byte	80C51 Cycle	W7100A Cycle	
					ISP / wizmemcpy	FLASH / user code
MOV	A, Rx	E8h – EFh	1	12	1	4
ADD	A, @Rx	26h – 27h	1	12	2	4
MOV	Rx, A	F8h – FFh	1	12	1	4
		Sum :		36	4	12

■ Register addressing

The following code performs an 8-bit register to register addition.

Rx = Rx + Ry

Mnemonic		Opcode	Byte	80C51 Cycle	W7100A Cycle	
					ISP / wizmemcpy	FLASH / user code
MOV	A, Rx	E8h – EFh	1	12	1	4
ADD	A, Ry	28h – 2Fh	1	12	1	4
MOV	Rx, A	F8h – FFh	1	12	1	4
		Sum :		36	3	12

13.2.2 Subtraction

- Immediate data

The following code performs immediate data (constant) subtraction from an 8-bit register.

Rx = Rx - #*n*

Mnemonic		Opcode	Byte	80C51 Cycle	W7100A Cycle	
					ISP / wizmemcpy	FLASH / user code
MOV	A, Rx	E8h – EFh	1	12	1	4
SUBB	A, #*n*	24h	2	12	2	4
MOV	Rx, A	F8h – FFh	1	12	1	4
		Sum :		36	4	12

- Direct addressing

The following code performs direct addressing subtraction from an 8-bit register.

Rx = Rx – (dir)

Mnemonic		Opcode	Byte	80C51 Cycle	W7100A Cycle	
					ISP / wizmemcpy	FLASH / user code
MOV	A, Rx	E8h – EFh	1	12	1	4
SUBB	A, dir	25h	2	12	2	4
MOV	Rx, A	F8h – FFh	1	12	1	4
		Sum :		36	4	12

- Indirect addressing subtraction

The following code performs indirect addressing subtraction from an 8-bit register.

Rx = Rx – (@Ry)

Mnemonic		Opcode	Byte	80C51 Cycle	W7100A Cycle	
					ISP / wizmemcpy	FLASH / user code
MOV	A, Rx	E8h – EFh	1	12	1	4
SUBB	A, @Ry	26h – 27h	1	12	2	4
MOV	Rx, A	F8h – FFh	1	12	1	4
		Sum :		36	4	12

- Register addressing subtraction

The following code performs an 8-bit register from register subtraction.

Rx = Rx - Ry

Mnemonic		Opcode	Byte	80C51 Cycle	W7100A Cycle	
					ISP / wizmemcpy	FLASH / user code
MOV	A, Rx	E8h – EFh	1	12	1	4
SUBB	A, Ry	28h – 2Fh	1	12	1	4
MOV	Rx, A	F8h – FFh	1	12	1	4
		Sum :		36	3	12

13.2.3 Multiplication

The following code performs the 8-bit register multiplication.

Rx = Rx * Ry

Mnemonic		Opcode	Byte	80C51 Cycle	W7100A Cycle	
					ISP / wizmemcpy	FLASH / user code
MOV	A, Rx	E8h – EFh	1	12	1	4
MOV	B, Ry	88h – 8Fh	2	24	2	4
MUL	AB	A4h	1	48	2	4
MOV	Rx, A	F8h – FFh	1	12	1	4
		Sum :		96	6	12

13.2.4 Division

The following code performs the 8-bit register division.

Rx = Rx / Ry

Mnemonic		Opcode	Byte	80C51 Cycle	W7100A Cycle	
					ISP / wizmemcpy	FLASH / user code
MOV	A, Rx	E8h – EFh	1	12	1	4
MOV	B, Ry	88h – 8Fh	2	24	2	4
DIV	AB	84h	1	48	6	8
MOV	Rx, A	F8h – FFh	1	12	1	4
		Sum :		96	10	20

13.3 16-Bit Arithmetic Functions

13.3.1 Addition

The following code performs 16-bit addition. The first operand and result are located in

registers pair RaRb. The second operand is located in registers pair RxRy.

RaRb = RaRb + RxRy

Mnemonic		Opcode	Byte	80C51 Cycle	W7100A Cycle	
					ISP / wizmemcpy	FLASH / user code
MOV	A, Rb	E8h – EFh	1	12	1	4
ADD	A, Ry	28h – 2Fh	1	12	1	4
MOV	Rb, A	F8h – FFh	1	12	1	4
MOV	A, Ra	E8h – EFh	1	12	1	4
ADDC	A, Rx	38h – 3Fh	1	12	1	4
MOV	Ra, A	F8h – FFh	1	12	1	4
		Sum :		72	6	24

13.3.2 Subtraction

The following code performs 16-bit subtraction. The first operand and result are located in registers pair RaRb. The second operand is located in registers pair RxRy.

RaRb = RaRb – RxRy

Mnemonic		Opcode	Byte	80C51 Cycle	W7100A Cycle	
					ISP / wizmemcpy	FLASH / user code
CLR	C	C3h	1	12	1	4
MOV	A, Rb	E8h – EFh	1	12	1	4
SUBB	A, Ry	98h – 9Fh	1	12	1	4
MOV	Rb, A	F8h – FFh	1	12	1	4
MOV	A, Ra	E8h – EFh	1	12	1	4
SUBB	A, Rx	98h – 9Fh	1	12	1	4
MOV	Ra, A	F8h – FFh	1	12	1	4
		Sum :		84	7	28

13.3.3 Multiplication

The following code performs 16-bit multiplication. The first operand and result are located in registers pair RaRb. The second operand is located in registers pair RxRy.

RaRb = RaRb * RxRy

Mnemonic	Opcode	Byte	80C51 Cycle	W7100A Cycle	
				ISP / wizmemcpy	FLASH / user code

MOV	A, Rb	E8h – EFh	1	12	1	4
MOV	B, Ry	88h – 8Fh	2	24	2	4
MUL	AB	A4h	1	48	2	4
MOV	Rz, B	A8h – AFh	2	24	3	4
XCH	A, Rb	C8h – CFh	1	12	2	4
MOV	B, Rx	88h – 8Fh	2	24	2	4
MUL	AB	A4h	1	48	2	4
ADD	A, Rz	28h - 2Fh	1	12	1	4
XCH	A, Ra	C8h – CFh	1	12	2	4
MOV	B, Ry	88h – 8Fh	2	24	2	4
MUL	AB	A4h	1	48	2	4
ADD	A, Ra	28h – 2Fh	1	12	1	4
MOV	Ra, A	F8h – FFh	1	12	1	4
		Sum :		312	23	52

13.4 32-bit Arithmetic Functions

13.4.1 Addition

The following code performs 32-bit addition. The first operand and result are located in four registers RaRbRcRd. The second operand is located in four registers RvRxRyRz.

RaRbRcRd = RaRbRcRd + RvRxRyRz

Mnemonic		Opcode	Byte	80C51 Cycle	W7100A Cycle	
					ISP / wizmemcpy	FLASH / user code
MOV	A, Rd	E8h – EFh	1	12	1	4
ADD	A, Rz	28h – 2Fh	1	12	1	4
MOV	Rd, A	F8h – FFh	1	12	1	4
MOV	A, Rc	E8h – EFh	1	12	1	4
ADDC	A, Ry	38h – 3Fh	1	12	1	4
MOV	Rc, A	F8h – FFh	1	12	1	4
MOV	A, Rb	E8h – EFh	1	12	1	4
ADDC	A, Rx	38h – 3Fh	1	12	1	4
MOV	Rb, A	F8h – FFh	1	12	1	4
MOV	A, Ra	E8h – EFh	1	12	1	4
ADDC	A, Rv	38h – 3Fh	1	12	1	4
MOV	Ra, A	F8h – FFh	1	12	1	4
		Sum :		144	12	48

13.4.2 Subtraction

The following code performs 32-bit subtraction. The first operand and result are located in four registers RaRbRcRd. The second operand is located in four registers RvRxRyRz.

RaRbRcRd = RaRbRcRd – RvRxRyRz

Mnemonic		Opcode	Byte	80C51 Cycle	W7100A Cycle	
					ISP / wizmemcpy	FLASH / user code
CLR	C	C3h	1	12	1	4
MOV	A, Rd	E8h – EFh	1	12	1	4
SUBB	A, Rz	98h – 9Fh	1	12	1	4
MOV	Rd, A	F8h – FFh	1	12	2	4
MOV	A, Rc	E8h – EFh	1	12	1	4
SUBB	A, Ry	98h – 9Fh	1	12	2	4
MOV	Rc, A	F8h – FFh	1	12	2	4
MOV	A, Rb	E8h – EFh	1	12	1	4
SUBB	A, Rx	98h – 9Fh	1	12	1	4
MOV	Rb, A	F8h – FFh	1	12	1	4
MOV	A, Ra	E8h – EFh	1	12	1	4
SUBB	A, Rv	98h – 9Fh	1	12	1	4
MOV	Ra, A	F8h – FFh	1	12	1	4
Sum :				156	13	52

13.4.3 Multiplication

The following code performs 32-bit multiplication. The first operand and result are located in four registers RaRbRcRd. The second operand is located in four registers RvRxRyRz.

RaRbRcRd = RaRbRcRd * RvRxRyRz

Mnemonic		Opcode	Byte	80C51 Cycle	W7100A Cycle	
					ISP / wizmemcpy	FLASH / user code
MOV	A, R0	E8h – EFh	1	12	1	4
MOV	B, R7	88h – 8Fh	2	24	2	4
MUL	AB	A4h	1	48	2	4
XCH	A, R4	C8h – CFh	1	12	2	4
MOV	B, R3	88h – 8Fh	2	24	2	4
MUL	AB	A4h	1	48	2	4
ADD	A, R4	28h – 2Fh	1	12	1	4

MOV	R4, A	F8h – FFh	1	12	1	4
MOV	A, R1	E8h – EFh	1	12	1	4
MOV	B, R6	88h – 8Fh	2	24	2	4
MUL	AB	A4h	1	48	2	4
ADD	A, R4	28h – 2Fh	1	12	1	4
MOV	R4, A	F8h – FFh	1	12	1	4
MOV	B, R2	88h – 8Fh	2	24	2	4
MOV	A, R5	E8h – EFh	1	12	2	4
MUL	AB	A4h	1	48	2	4
ADD	A, R4	28h – 2Fh	1	12	1	4
MOV	R4, A	F8h – FFh	1	12	1	4
MOV	A, R2	E8h – EFh	1	12	1	4
MOV	B, R6	88h – 8Fh	2	24	2	4
MUL	AB	A4h	1	48	2	4
XCH	A, R5	C8h – CFh	1	12	2	4
MOV	R0, B	A8h – AFh	2	24	3	4
MOV	B, R3	88h – 8Fh	2	24	2	4
MUL	AB	A4h	1	48	2	4
ADD	A, R5	28h – 2Fh	1	12	1	4
XCH	A, R4	C8h – CFh	1	12	2	4
ADDC	A, R0	38h – 3Fh	1	12	1	4
ADD	A, B	25h	2	12	2	4
MOV	R5, A	F8h – FFh	1	12	1	4
MOV	A, R1	E8h – EFh	1	12	1	4
MOV	B, R7	88h – 8Fh	2	24	2	4
MUL	AB	A4h	1	48	2	4
ADD	A, R4	28h – 2Fh	1	12	1	4
XCH	A, R5	C8h – CFh	1	12	2	4
ADDC	A, B	35h	2	12	2	4
MOV	R4, A	F8h – FFh	1	12	1	4
MOV	A, R3	E8h – EFh	1	12	1	4
MOV	B, R6	88h – 8Fh	2	24	2	4
MUL	AB	A4h	1	48	2	4
MOV	R6, A	F8h – FFh	1	12	1	4
MOV	R1, B	A8h – AFh	2	24	3	4
MOV	A, R3	E8h – EFh	1	12	1	4

MOV	B, R7	88h – 8Fh	2	24	2	4
MUL	AB	A4h	1	48	2	4
XCH	A, R7	C8h – CFh	1	12	2	4
XCH	A, B	C5h	2	12	3	4
ADD	A, R6	28h – 2Fh	1	12	1	4
XCH	A, R5	C8h – CFh	1	12	2	4
ADDC	A, R1	38h – 3Fh	1	12	1	4
MOV	R6, A	F8h – FFh	1	12	1	4
CLR	A	E4h	1	12	1	4
ADDC	A, R4	38h – 3Fh	1	12	1	4
MOV	R4, A	F8h – FFH	1	12	1	4
MOV	A, R2	E8h – EFh	1	12	1	4
MUL	AB	A4h	1	48	2	4
ADD	A, R5	28h – 2Fh	1	12	1	4
XCH	A, R6	C8h – CFh	1	12	2	4
ADDC	A, B	38h – 3Fh	2	12	2	4
MOV	R5, A	F8h – FFh	1	12	1	4
CLR	A	E4h	1	12	1	4
ADDC	A, R4	38h – 3Fh	1	12	1	4
MOV	R4, A	F8h – FFh	1	12	1	4
		Sum :		1248	99	252