目　录

第一章　ATMEL 单片机简介

1.1　ATMEL 公司的产品特点

ATMEL 公司是世界上著名的高性能、低功耗、非易失性存储器和数字集成电路的一流半导体制造公司。ATMEL 公司最令人注目的是它的 EEPROM 电可擦除技术，闪速存储器技术和高质量、高可靠性的生产技术。在 CMOS 器件生产领域中，ATMEL 的先进设计水平、优秀的生产工艺及封装技术，一直处于世界的领先地位。这些技术用于单片机生产，使单片机也具有优秀的品质，在结构、性能和功能等方面都有着明显的优势。ATMEL 公司的单片机是目前世界上一种独具特色而性能卓越的单片机，它在计算机外部设备、通讯设备、自动化工业控制、宇航设备、仪器仪表和各种消费类产品中都有着广泛的应用前景。ATMEL 公司产品的主要特点表现为如下几点：

一、以 EEPROM 电可擦除及 Flash 技术为主导

ATMEL 公司把其 EEPROM 及 Flash 技术巧妙地用于形成特殊的集成电路，从而使一些芯片的应用领域扩大，其中闪速可编程逻辑器件 Flash PLD、Flash 存储器、AT89 系列 Flash 单片机、AVR 增强型单片机、智能 IC 卡等是最典型的产品。这些产品内部含有 Flash 存储器，从而使它们在无交流电环境下的便携类的产品中大有用场。含有 EEPROM 及 Flash 存储器是 ATMEL 公司有关产品的明显特色之一。

在 ATMEL 公司的 Flash 产品中，一共有商业 C 档、工业 I 档、汽车 A 档和军用 M 档四种档次的产品。C 档产品使用的温度为 0～＋70 ℃；I 档产品使用的温度为－40～＋85 ℃；A 档产品使用的温度为－40～＋125 ℃；M 档产品使用的温度为－55～＋150 ℃。

二、有多种封装形式和高的质量

ATMEL 公司有多种封装形式的集成电路，这些封装形式有：DIP、PGA、PQFP、TQFP、SOIC、CBGA、μBGA 和客户专门定制等多种。

ATMEL 公司的封装是按军工标准进行的，有优秀的商品质量。军工产品封装和测试按军用标准 MIL－STD－883 进行。所有的军工产品在制造和开发过程中均以 MIL－M－38510 标准说明书为依据，并且追踪这个标准的最新版本。同时，ATMEL 公司将统计过程控制 SPC 用于军用 IC 的装配和测试中，从而优化质量和产品的稳定性。

三、高标准的质量检测

ATMEL 公司具有高质量、高水准的检测能力，可以对数字集成电路和模拟集成电路进行质量检测，对军用集成电路进行质量检测。对 ATMEL 公司的军用集成电路产品而言，其工作性能是完全符合军用标准的，在－55～＋125 ℃范围内工作十分正常，甚至在高达＋150 ℃的条件下集成电路仍然能实现正常的输出功能。ATMEL 公司的 Audo、Sentry 和 Teraclyne 测试器符合统计过程控制 SPC 标准，并且依照美国国家标准局的测试标准执行。

由于 ATMEL 公司的产品有优秀的品质，故在航空航天仪器、雷达系统、导弹、智能自适应

仪器、机器人、各种武器电子系统、抗恶劣环境电子系统等都能广泛加以应用。

1.2 AT 89 系列单片机简介

89 系列单片机是 ATMEL 公司的 8 位 Flash 单片机系列。这个系列单片机的最大特点是在片内含有 Flash 存储器。因此，在应用中有着十分广泛的前景和用途，特别是在便携式、省电及特殊信息保存的仪器和系统中显得更为有用。

1.2.1 89 系列单片机特点

AT89 系列单片机是以 8031 核构成的，所以，它和 8051 系列单片机是兼容的系列。这个系列对于以 8051 为基础的系统来说，是十分容易进行取代和构造的。故而对于熟悉 8051 的用户来说，用 ATMEL 公司的 89 系列单片机进行取代 8051 的系统设计是轻而易举的事。

一、89 系列单片机的优点

(1) 内部含 Flash 存储器　在系统的开发过程中可以十分容易进行程序的修改，这就大大缩短了系统的开发周期。同时，在系统工作过程中能有效地保存一些数据信息，即使外界电源损坏也不会影响到信息的保存。

(2) 和 80C51 插座兼容　89 系列单片机的引脚是和 80C51 的引脚一样的，所以，当用 89 系列单片机取代 80C51 时，可以直接进行代换。这时，不管采用 40 引脚或是 44 引脚的产品，只要用相同引脚的 89 系列单片机取代 80C51 的单片机即可。

(3) 静态时钟方式　89 系列单片机采用静态时钟方式，所以可以节省电能，这对于降低便携式产品的功耗十分有用。

(4) 错误编程亦无废品产生　一般的 OTP 产品，一旦错误编程就成了废品。而 89 系列单片机内部采用了 Flash 存储器，所以，错误编程之后仍可以重新编程，直到正确为止，故不存在废品。

(5) 可进行反复系统试验　用 89 系列单片机设计的系统，可以反复进行系统试验；每次试验可以编入不同的程序，这样可以保证用户的系统设计达到最优。而且，随用户的需要和发展，还可以进行修改，使系统不断能追随用户的最新要求。

二、89 系列单片机的内部结构

89 系列单片机的内部结构和 80C51 相近，主要含有如下一些部件：

(1) 8031CPU
(2) 振荡电路
(3) 总线控制部件
(4) 中断控制部件
(5) 片内 Flash 存储器
(6)片内 RAM
(7)并行 I/O 接口
(8) 定时器
(9)串行 I/O 接口
(10)片内 EEPROM

在 89 系列单片机中，AT89C1051 的 Flash 存储器容量最小，只有 1K；而 AT89S55 的 Flash 存储器容量最大，有 20K。

在这个系列中，结构最简单的是 AT89C1051，它内部不含串行接口；最复杂的是 AT89S8252，它内部不但含标准的串行接口，还含有一个串行外围接口 SPI、Watchdog 定时器、双数据指针、EEPROM、电源下降的中断恢复等功能和部件。

89 系列单片机目前有多种型号,分别为 AT89C1051、AT89C2051、AT89C4051、AT89C51、AT89LV51、AT89C52、AT89LV52、AT89S8252、AT89LS8252、AT89C55、AT89LV55、AT89S53、AT89LS53、AT89S4D12。其中,AT89LV51、AT89LV52 和 AT89LV55 分别是 AT89C51、AT89C52 和 AT89C55 的低电压产品,最低电压可以低至 2.7 V;而 AT89C1051 和 AT89C2051 则是低档型低电压产品,它们仅有 20 个引脚,最低电压仅为 2.7 V。

三、89 系列单片机的型号编码

89 系列单片机的型号编码由三个部分组成,它们是前缀、型号和后缀。格式如下:

AT89C XXXXXXXX 其中,AT 是前缀 ,89CXXXX 是型号,XXXX 是后缀。

下面分别对这三个部分进行说明,并且对其中有关参数的表示和意义作出相应的解释。

(1) 前缀 由字母"AT"组成,表示该器件是 ATMEL 公司的产品。

(2) 型号 由"89CXXXX"或"89LVXXXX"或"89SXXXX"等表示。

"89CXXXX"中,9 是表示内部含 Flash 存储器,C 表示为 CMOS 产品。

"89LVXXXX"中,LV 表示低压产品。

"89SXXXX"中,S 表示含有串行下载 Flash 存储器。

在这个部分的"XXXX"表示器件型号数,如 51、1051、8252 等。

(3) 后缀 由"XXXX"四个参数组成,每个参数的表示和意义不同。在型号与后缀部分有"-"号隔开。

后缀中的第一个参数 X 用于表示速度,它的意义如下:

X=12,表示速度为 12 MHz。 X=20,表示速度为 20 MHz。

X=16,表示速度为 16 MHz。 X=24,表示速度为 24 MHz。

后缀中的第二个参数 X 用于表示封装,它的意义如下:

X=D,表示陶瓷封装。 X=Q,表示 PQFP 封装。

X=J,表示 PLCC 封装。 X=A,表示 TQFP 封装。

X=P,表示塑料双列直插 DIP 封装。 X=W,表示裸芯片。

X=S,表示 SOIC 封装。

后缀中第三个参数 X 用于表示温度范围,它的意义如下:

X=C,表示商业用产品,温度范围为 0～+70 ℃。

X=I,表示工业用产品,温度范围为 -40～+85 ℃。

X=A,表示汽车用产品,温度范围为 -40～+125 ℃。

X=M,表示军用产品,温度范围为 -55～+150 ℃。

后缀中第四个参数 X 用于说明产品的处理情况,它的意义如下:

X 为空,表示处理工艺是标准工艺。

X=/883,表示处理工艺采用 MIL-STD-883 标准。

例如:有一个单片机型号为"AT89C51-12PI",则表示意义为该单片机是 ATMEL 公司的 Flash 单片机,内部是 CMOS 结构,速度为 12 MHz,封装为塑封 DIP,是工业用产品,按标准处理工艺生产。

1.2.2 89 系列单片机分类

AT89 系列单片机可分为标准型号、低档型号和高档型号三类。

标准型有 AT89C51 等六种型号,它们的基本结构和 89C51 是类似的,是 80C51 的兼容产品。低档型有 AT89C1051 等两种型号,它们的 CPU 核和 89C51 是相同的,但并行 I/O 口较少。高档型有 AT89S8252 等型号,是一种可串行下载的 Flash 单片机,可以用在线方式对单片机进行程序下载

一、标准型单片机

标准型单片机有 89C51、89LV51、89C52、89LV52、89C55、89LV55 六种型号。

标准型 89 系列单片机是和 MCS－51 系列单片机兼容的。在内部含有 4K、8K 或 20K 可重复编程的 Flash 存储器,可进行 1 000 次擦写操作。全静态工作为 0～33 MHz,有三级程序存储器加密锁定,有内部含 128～256 字节的 RAM,有 32 条可编程的 I/O 端口,有 2～3 个 16 位定时器/计数器,有 6～8 级中断,有通用串行接口,有低电压空闲及电源下降方式。

在这六种型号中,AT89C51 是一种基本型号。AT89LV51 是一种能在低电压范围工作的改进型,可在 2.7～6 V 电压范围工作,其它功能和 89C51 相同。AT89C52 是在 AT89C51 的基础上,在存储器容量、定时器和中断能力上得到改进的型号。89C52 的 Flash 存储器容量为 8K,16 位定时器/计数器有 3 个,中断有 8 级。而 89C51 的 Flash 存储器容量为 4K,16 位定时器/计数器有 2 个,中断只有 6 级。AT89LV52 是 89C52 的低电压型号,可在 2.7～6 V 电压范围内工作。89C55 的 Flash 存储器容量为 20K,16 位定时/计数器有 3 个,中断有 8 级。AT89LV55 是 89C55 的低电压型号,可在 2.7～6 V 电压范围内工作。

二、低档型单片机

低档型的单片机有 AT89C1051 和 AT89C2051 两种型号。除并行 I/O 端口数较少之外,其它部件结构基本和 AT89C51 差不多。之所以被称为低档型,主要是因为它的引脚只有 20 条,比标准型的 40 引脚少得多。

AT89C1051 的 Flash 存储器只有 1K,RAM 只有 64 个字节,内部不含串行接口,内部的中断响应只有 3 种,保密锁定位只有 2 位。这些也是和标准型的 AT89C51 有区别的地方。AT89C2051 的 Flash 存储器只有 2K,RAM 只有 128 个字节,保密锁定位有 2 位。

也由于在上述有关部件上 AT89C1051、AT89C2051 的功能比标准型 AT89C51 要弱,所以它们就处于低档位置。

三、高档型单片机

高档型有 AT89S53、AT89S8252、AT89S4D12 等型号,是在标准型的基础上增加了一些功能形成的。增加的功能主要有如下几点:

(1) AT89S4D12 有 4K 可下载 Flash 存储器,AT89S8252 有 8K 可下载 Flash 存储器,AT89S53 有 12K 可下载 Flash 存储器。下载功能是由 IBM 微机通过 89 系列单片机的串行外围接口 SPI 执行的。

(2) 除 8K Flash 存储器外,AT89S8252 还含有一个 2K 的 EEPROM,从而提高了存储容量。

(3) 含有 9 个中断响应的能力。

(4) 含标准型和低档型所不具有的 SPI 接口。

(5) 含有 Watchdog 定时器。

(6) 含有双数据指针。

(7) 含有从电源下降的中断恢复。

(8) AT89S4D12 除了 4K 可下载 Flash 存储器之外,还一个 128K 片内 Flash 数据存储器,12 MHz 内部振荡器,5 个可编程 I/O 线。

1.3 AT90 系列单片机简介

ATMEL 公司的 90 系列单片机是增强 RISC 内载 Flash 的单片机,通常简称为 AVR 单片机。

90 系列单片机是基于新的精简指令 RISC 结构的。这种结构是在 90 年代开发出来的,综合了半导体集成技术和软件性能的新结构。这种结构使得在 8 位微处理器市场上,AVR 单片机具有最高 MIPS/mW 能力。

为了加快进入市场的时间和简化维护的支持,对于单片机来说,用高级语言编程成了一种标准编程方法。AVR 结构单片机的开发目的就是在于能采用 C 语言编程,从而能高效地开发出目标产品。为了对目标代码大小、性能及功耗的优化,AVR 单片机采用了大型快速存取寄存器文件和快速单周期指令。

快速存取 RISC 寄存器文件由 32 个通用工作寄存器组成。传统的基于累加器的结构需要大量的程序代码,以实现在累加器和存储器之间的数据传送。在 AVR 单片机中,用 32 个通用工作寄存器代替累加器,从而可以避免传统的累加器和存储器之间的数据传送。

在 AVR 单片机中,在前一条指令执行的时候,就取出现行的指令,然后以一个周期执行指令。在其它的 CISC 以及类似的 RISC 结构中,外部振荡器的时钟被分频降低到传统的内部执行周期,这种分频最大达 12 倍。AVR 单片机是用一个时钟周期执行一条指令的,它是在 8 位单片机中第一种真正的 RISC 单片机。

AVR 单片机有良好的性能价格比,这个系列有引脚少的器件,也有含较大容量存储器引脚较多的器件。由于 AVR 单片机是采用 Harvard 结构的,故它们的程序存储器和数据存储器是分开的。可直接访问 8M 字节程序存储器和 8M 字节数据存储器,寄存器文件被双向映射,并能被访问如同片内允许快速上下转换的那部分 SRAM 存储器。

AVR 单片机采用低功率、非挥发的 CMOS 工艺制造,通过 SPI 口和一般的编程器,可以对 AVR 单片机的 Flash 存储器进行编程。

AT90 系列单片机目前有 AT90S1200、AT90S2313、AT90S4414、AT90S8515、AT90S2323、AT90S2343、AT90SMEG603、AT90SMEG103、AT90S4434、AT90S8535 等多种型号,它们在功能和存储器容量等方面有一定的区别。但是,它们都是比 89 系列要强的单片机。表 1.1 为 ATMEL AVR－RISC 单片机 AT90 系列选型表。

表 1.1 AT90 系列选型表

AT90S 系列	1200	2313	4414	8515	2323	2343
Flash 编程	1K 字节 (512 * 16)	2K 字节 (1 024 * 16)	4K 字节 (2 048 * 16)	8K 字节 (4 096 * 16)	2K 字节 (1K * 16)	2K 字节 (1K * 16)
SRAM 数据	0	128 字节	256/64K 片外	512/64K 片外	128 字节	128 字节
EEPROM 工作寄存器	64 字节	128 字节	256 字节	512 字节	128 字节	128 字节
工作寄存器	32 字节	32 字节	32 字节	32 字节	32 字节	32 字节

表 1.1 续

AT90S 系列	1200	2313	4414	8515	2323	2343
I/O	15 20 mA	15 20 mA	32 20 mA	32 20 mA	5 20 mA	5 20 mA
A/D	No	No	No	No	No	No
PWM D/A	No	1～10 位	2～10 位	2～10 位	No	No
频率/MHz	4/8/12/16 M, WDT 片内晶振	4/8/12 M, WDT	4/8M, WDT	4/8M, WDT	8M, WDT	8M, WDT 片内晶振
定时器/ 计数器	1～8 位	1～8 位 1～16 位	1～8 位 1～16 位	1～8 位 1～16 位	1～8 位	1～8 位
ISP 编程	Yes	Yes	Yes	Yes	Yes	Yes
UART	No	1～8 位, 9 位	1～8 位, 9 位	1～8 位, 9 位	No	No
SPI	No	No	1－主,从	1－主,从	No	No
V_{CC}/V I_{CC}/ mA	2.7～6 2	2.7～6 2.5	2.7～6 3.5	2.7～6 3.5	2.7～6 1.1	2.7～6 1.1
电源存储	闲置(μA)掉电	闲置(μA)掉电	闲置(μA)掉电	闲置(μA)掉电	闲置(μA)掉电	闲置(μA)掉电
比较器	1	1, B.R. 直接	1	1	No	No
包装	dip20, soic20	dip20, soic20	dip40, plcc 44, tqfp44	dip40, plcc 44, tqfp44	dip8/soic8	dip8/soic8
AT90S 系列	MEG603	MEG103	4434	8535	TINY10	TINY24
Flash 编程	64K 字节 (32K * 16)	128K 字节 (64K * 16)	4K 字节 (2K * 16)	8K 字节 (4K * 16)	1K 字节 (512 * 16)	2K 字节 (1K * 16)
SRAM 数据	4K/64K 片外	4K/64K 片外	256/64K 片外	512/64K 片外	No	No
EEPROM 工作寄存器	2K 字节	4K 字节	256 字节	512 字节	No	128 字节
工作寄存器	32 字节	32 字节	32 字节	32 字节	32 字节	32 字节
I/O	32	32	32	32	5	17
A/D	8～10 位	8～10 位	8～10 位	8～10 位	No	No
PWM D/A	2～8 位 2～10 位	2～8 位 2～10 位	3～10 位	3～10 位	No	No
频率/ MHz	4/6M, WDT, 32 768RTC	4/6M, WDT, 32 768RTC	8M, WDT RTC	8M, WDT RTC	8/4/2M; WDT 片内晶振	1/4M, WDT
定时器/ 计数器	2～8 位 1～16 位	1～8 位 1～16 位	1～8 位 1～16 位	1～8 位 1～16 位	1～8 位	1～8 位
ISP 编程	Yes	Yes	Yes	Yes	No	Yes
UART	1～8 位, 9 位	1～8 位, 9 位	1～8 位, 9 位	1～8 位, 9 位	No	No
SPI	1－主,从	1－主,从	1－主,从	1－主,从	No	No
V_{CC}/V I_{CC}/mA	2.7～6 3.5	2.7～6 9.5	2.7～6 3.5	2.7～6 3.5	1.8～5.5 1.1	1.8～5.5 1.1
电源存储	闲置(μA)掉电	闲置(μA)掉电	闲置(μA)掉电	闲置(μA)掉电	闲置(μA)掉电	闲置(μA)掉电
比较器	1	1	1, B.R. 直接	1, B.R. 直接	1	1
包装	tqfp64	tqfp64	dip40, plcc 44, tqfp44	dip40, plcc 44, tqfp44	dip8/soic8	dip24 /soic24

1.4 AT91M 系列单片机

AT91M 是基于 ARM7TDMI 嵌入式处理器的 ATMEL 16/32 微处理器系列中的一个新成员。该处理器用高密度的 16 位指令集实现了高效的 32 位 RISC 结构，且功耗很低。此外，内部的工作寄存器很多，使该器件非常适用于实时控制的应用。该器件使用 ATMEL 公司的高密度 CMOS 技术，通过在一个单片上集成了 ARM7TDMI 和大量的 ROM 程序区，以及片内 RAM 和广泛的外设功能，使得 ATMEL 的 AT91M 成为一个强有力的微控制器，为许多需要加强运算的嵌入式控制器提供了高度的灵活性、高性能价格比的解决方案。

AT91M 使用了基于先进微控制器总线结构(AMBA)的模块化设计方法，具有综合、快速、高性能价格比的特点。

AT91M 系列单片机目前有 AT91M4020X、AT91M4120X、AT91M00100 等产品。

表 1.2 为 AT91M 系列部分产品的 ROM 大小表。

表 1.2 AT91M 系列产品的 ROM 大小表

产 品	ROM 大小	产 品	ROM 大小
AT91M40200	无 ROM	AT91M41200	无 ROM
AT91M40203	32K 字节	AT91M41203	32K 字节
AT91M40205	64K 字节	AT91M41205	64K 字节
AT91M40207	128K 字节	AT91M41207	128K 字节

第二章　AVR 单片机系统结构

ATMEL 公司的 90 系列单片机是一种基于 AVR 增强性能、RISC 结构的、低功耗、CMOS 技术、八位微控制器(Enhanced RISC Microcontrollers),通常简称为 AVR 单片机。目前有 AT90S1200、AT90S2313、AT90S4414、AT90S8515、AT90S2323、AT90S4434、AT90SMEG101、AT90SMEG103、AT90STINY8、AT90STINY24 等多种型号。它们的基本结构都比较相近,这一章以 AT90S8515 单片机的内部结构为主来叙述 AVR 单片机系统结构。

2.1　AVR 单片机总体结构

90 系列单片机通过在单一时钟周期内执行功能强大的指令,每 MHz 可实现 1MIPS 的处理能力,这使得设计者可以优化功耗与速度之间的矛盾。

AVR 核为 32 个通用寄存器与丰富指令集的组合。32 个寄存器全部直接地与运算逻辑单元(ALU)连接,这使得可以通过在一个时钟周期内执行一条指令来访问到两个独立的寄存器。这种组合机构具备的代码效率比完成同样处理能力的常规 CISC 微控制器要快十倍。

90 系列单片机具有以下特征:1 K～128 K 字节可下载的 Flash 存储器,64～4 K 字节 EEPROM,128～4 K 字节 RAM,5～32 条通用的 I/O 线,32 个通用寄存器,带比较模式的定时器/计数器,可编程的串行 UART,内部及外部中断,带内部晶振的可编程看门狗定时器,一个为下载程序而设计的 SPI 串行口,10 位 A/D 转换器,以及 2 个可通过软件选择的省电模式。闲置模式停止 CPU 的工作,而 SRAM、定时器/计数器、SPI 口及中断系统继续工作。掉电模式保留寄存器的内容,但冻结晶振,终止芯片的其它功能,直至下一次外部中断或硬件复位。

该器件的制造运用了 ATMEL 公司的高密度非易失存储器技术。芯片内可下载的 Flash 存储器可以通过 SPI 串行接口或通过通用的非易失存储器编程器对程序存储器进行系统内的重新编程。通过在单一芯片内将一个增强性能的 RISC 8 位 CPU 与可下载的 Flash 结合,使得 ATMEL 的 90 系列单片机成为一种适合于许多要求、具有高度灵活性和低成本的嵌入式控制应用的高效微控制器。

90 系列单片机 AVR 的全套编程和系统开发工具包括:C 编译器,宏汇编器,程序调试器/程序仿真器,系统在线仿真器和评估板。

图 2.1 为 AT90S8515 单片机方框图,说明了 90 系列单片机的内部结构。

一、引脚说明

(1) V_{CC}　V_{CC}为供电引脚,连接到正电源。

(2) GND　GND 为接地引脚,连接到电源地。

(3) A 口(PA7～PA0)　A 口为一个 8 位双向 I/O 口,每一引脚内部都有上拉电阻。A 口的输出缓冲器可以吸收 20 mA 的电流,因而能直接驱动 LED 显示器。当 A 口被用于输入且内部上拉被激活时,如果外部被拉低,则会输出电流。当使用外部 SRAM 时,A 口作为复用的地址/数据和输入/输出口。

(4) B 口(PB7～PB0)　B 口为一个 8 位双向 I/O 口,每一引脚内部都有上拉电阻。B 口的输出缓冲器可以吸收 20 mA 的电流,当 B 口被用于输入且内部上拉被激活时,如果外部被拉低,则会输出电流。B 口也提供后面列出的 90 系列单片机许多特殊功能。

(5) C 口(PC7～PC0)　C 口为一个 8 位双向 I/O 口,每一引脚内部都有上拉电阻。C 口

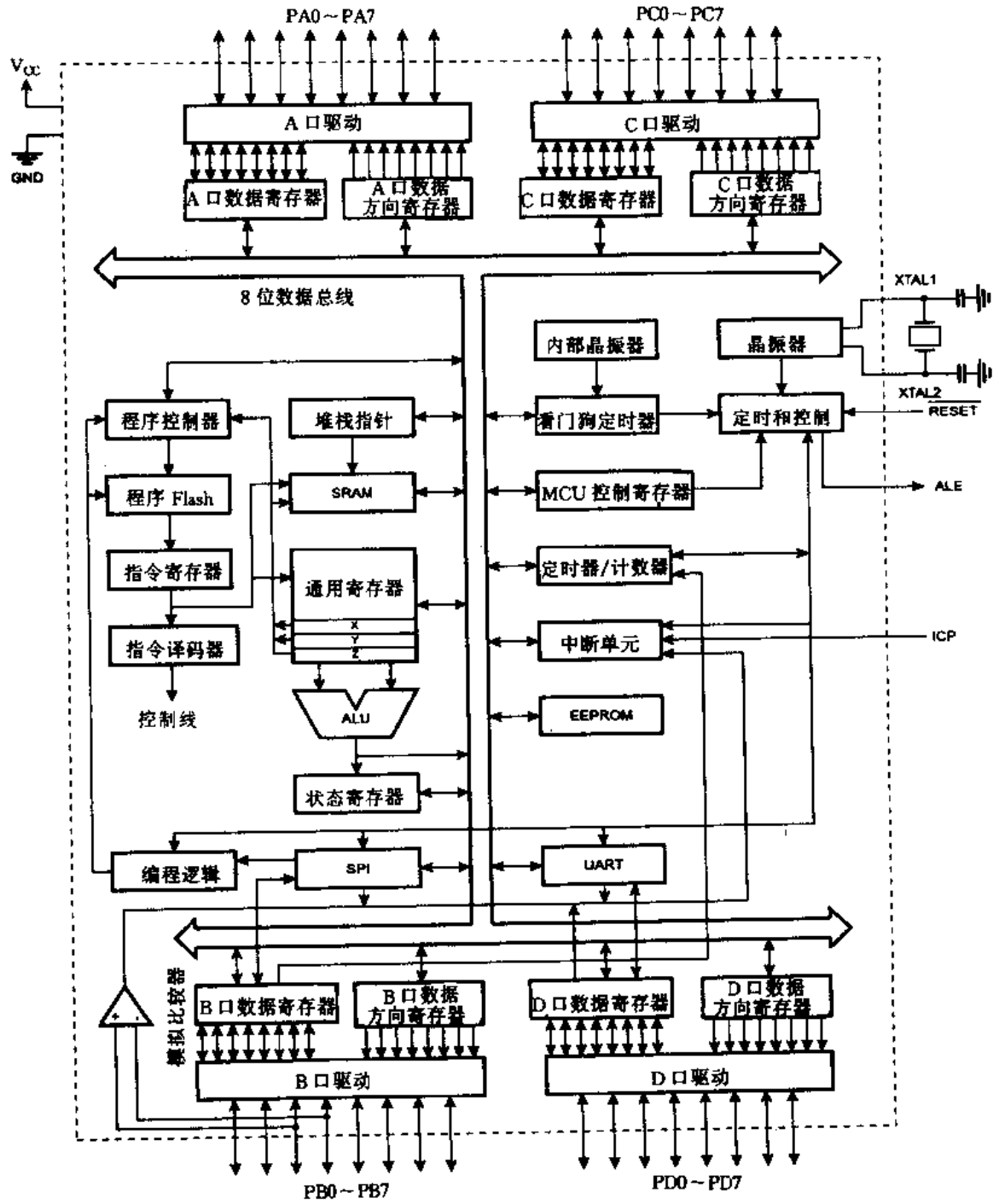


图 2.1 AT90S8515 单片机方框图

的输出缓冲器可以吸收 20 mA 的电流，当 C 口被用于输入且内部上拉被激活时，如果外部被拉低，则会输出电流。当使用外部 SRAM 时，C 口作为地址输出。

(6) D 口(PD7～PD0) D 口为带有内部拉高的 8 位双向 I/O 口。D 口的输出缓存器可以吸收 20 mA 的电流。当 D 口被用于输入且内部上拉被激活时，如果外部被拉低，则会输出

电流。D 口也提供后面列出的 90 系列单片机许多特殊功能。

(7) RESET　RESET 为复位输入。当晶振运行时,引脚上一个两周期的低电平可对器件进行复位。

(8) XTAL1　XTAL1 为晶振反相放大器的输入端和内部时钟操作电路的输入端。

(9) XTAL2　XTAL2 为晶振反相放大器的输出端。

(10) ICP　ICP 是定时器/计数器 1 的输入捕获功能的输入引脚。

(11) OC1B　OC1B 是定时器/计数器 1 的输出比较功能 B 的输出引脚。

(12) ALE　ALE 是使用外部存储器时的地址锁存使能端。ALE 选通门被用于在第一个访问周期中将低位地址锁存到地址锁存器中, 而 PD0～PD7 在第二个访问周期中被用作数据。

二、晶振器

XTAL1 和 XTAL2 单独地作为反相放大器的输入和输出,该放大器如图 2.2 所示,可被设置为片内的晶振器。可使用石英晶振或陶瓷谐振器。为了由外部源驱动器件,当 XTAL1 被驱动时,XTAL2 不能连接,如图 2.3 所示。

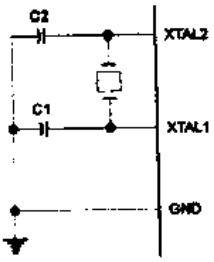


图 2.2　晶振连接

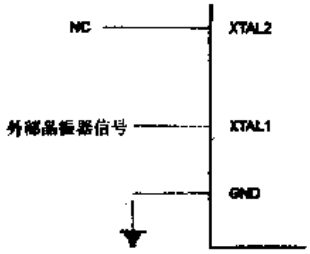


图 2.3　外部时钟驱动设置

2.2　AVR 单片机中央处理器 CPU

90 系列单片机 AVR RISC 微控制器向上与 AVR 增强 RISC 结构兼容。为 90 系列单片机 MCU 而编写的程序与 AVR 8 位 MCU(AT90SXXXX)全部系列产品的源代码和运行的时钟周期相兼容。

2.2.1　结构概述

快速访问寄存器文件概念包括 32 个带有单一时钟周期访问时间的 8 位通用寄存器。这意味着在一个单一时钟周期内,执行一个 ALU 操作(运算逻辑单元)。从寄存器文件中输出两个操作数,且操作数被执行, 运行结果被存储回寄存器文件——这些操作在一个时钟周期内完成。

32 个寄存器中的 6 个寄存器作为 3 个 16 位间接地址寄存器指针,被用于数据空间寻址,从而可以进行高效的地址计算。3 个寄存器中的一个还被用作地址指示器,完成常量表的查询功能。这些新加的功能寄存器为 16 位 X－寄存器,16 位 Y－寄存器,16 位 Z－寄存器。

ALU 支持寄存器之间的运算和逻辑功能,以及常数和寄存器之间的运算与逻辑功能。ALU 也执行单一的寄存器操作, 图 2.4 所示为 AT90S8515 单片机 AVR 的结构。

除了寄存器操作功能之外,常规存储器寻址模式还可用在寄存器文件上。寄存器文件被分配在最低的 32 个数据空间地址($00～ $1F),当使能时,即使它们为一般的存储地址,也能访问到。

I/O 存储器空间包含 64 个作为 CPU 外围功能的地址,为控制寄存器、定时器/计数器、A/D 转换器及其它 I/O 功能。I/O 存储器可被直接访问,或作为寄存器文件之后的数据空间,地

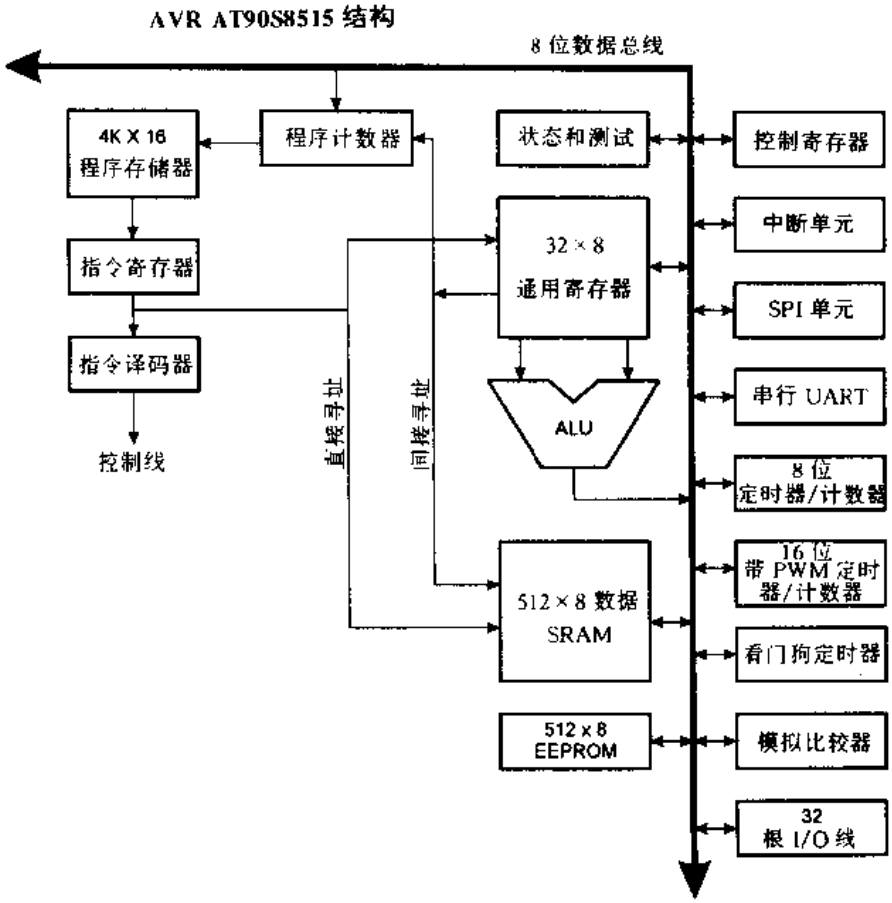


图 2.4 AT90S8515 单片机 AVR 增强性 RISC 结构

址为 $20～$5F。

AVR 运用 Harvard 结构概念,即对程序存储和数据带有不同的存储器和总线。通过单一级的流水线可对程序存储器进行访问。当执行某一指令时,下一指令被预先从程序存储器中取回。这使得指令可以在每一个时钟周期内被执行。芯片内的程序存储器为系统内可下载的 Flash 存储器。

通过相关的转移和调用指令,全部的 4K 地址空间可以被直接访问到。所有 AVR 指令均有一个单一的 16 位字格式,这意味着每个程序存储器地址包含了一个单一的 16 位或 32 位指令。

在中断和子程序调用过程中,返回地址程序计数器(PC)被存储于堆栈之中。该堆栈被高效率地放置在通用 SRAM 中,作为结果,堆栈的大小只被 SRAM 的的大小及对 SRAM 的使用而限制。所有的用户程序必须在复位状态初始化 SP(在子程序和中断程序被执行之前)。16 位的堆栈指针 SP 可在 I/O 空间之中被进行读/写访问。

128～4K 字节的数据 SRAM 可以通过 AVR 结构中的不同种寻址模式很容易地被访问。

AVR 结构中的存储器空间均为线性和有规律的存储器映射。图 2.5 为存储器映射图。

一个灵活的中断模块,在 I/O 空间中有它自己的控制寄存器,并且在状态寄存器中带有一个附加的全局中断使能位。所有不同的中断,在程序存储器开始位置的中断向量表中,带有一个独立分开的中断向量。不同的中断优先级与中断向量位置相一致。中断地址向量的位置越低,优先级越高。

2.2.2 通用寄存器文件

图 2.6 为 AVR 单片机 CPU 中 32 个通用寄存器的结构图。

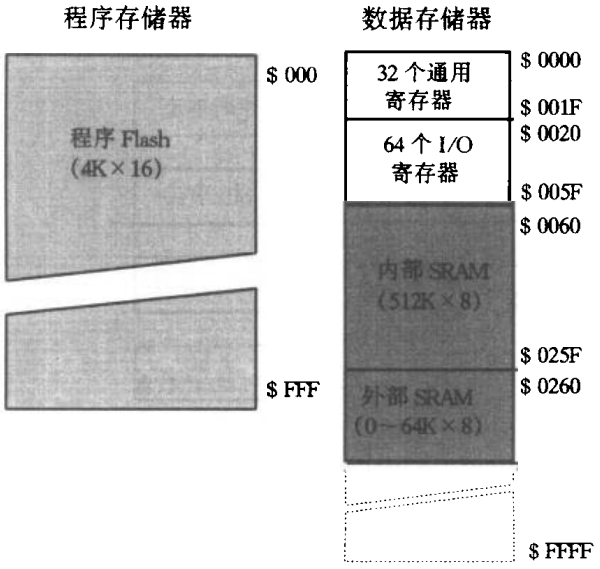


图 2.5 存储器映射图

图 2.6 AVR 单片机 CPU 通用寄存器

指令集中所有的寄存器操作指令均带有方向，并能在单一周期中访问所有的寄存器。唯一的例外是，存在于常量和寄存器之间的 5 个常量运算和逻辑指令 SBCI、SUBI、CPI、ANDI、ORI 以及直接装入常数的 LDI 指令。这些指令适用于寄存器文件(R16～R31)之中后半部分的寄存器。通用的 SBC、SUB、CP、AND、OR 指令，以及两个寄存器之间的操作指令或单一寄存器操作指令，在整个寄存器文件中都是适用的。

如图 2.6 所示，每个寄存器被分配给一个数据存储器地址，将其直接映射到用户数据空间的前 32 个地址。虽然寄存器文件并未像 SRAM 定位那样提供物理地址，但寄存器 X、Y、Z 可被设置用来对文件中的寄存器做索引，这种存储器的组成结构在访问寄存器时提供了极大的灵活性。

2.2.3 X、Y、Z 寄存器

为了实现通用目的，寄存器 R26～R31 具有一些新加的功能。这些寄存器作为对数据空间间接寻址的地址指针。这三个间接寄存器 X、Y、Z 由图 2.7 定义。在不同的寻址模式下，这些地址寄存器的功能为固定位移、自动增量和减量(请参考不同的指令)。

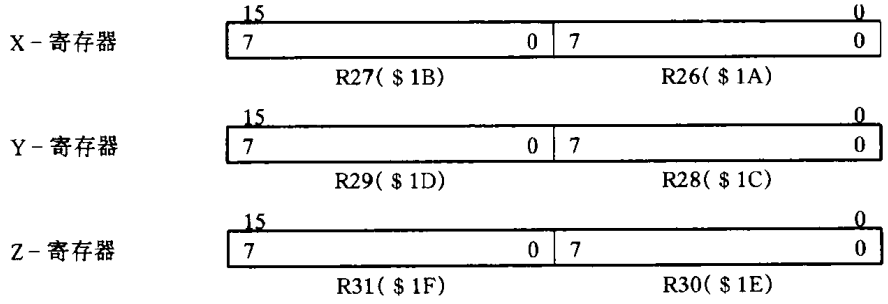


图 2.7 X、Y、Z 寄存器

2.2.4 ALU 运算逻辑单元

高性能的 AVR 运算逻辑单元的操作与所有的 32 个通用寄存器保持直接的连接。在单一时钟周期内,ALU 是在寄存器文件中的寄存器之间执行操作。ALU 操作被分为 3 个主要的目的:算法、逻辑和位功能。AVR 产品家族中一些微控制器的 ALU 运算部件中有硬件乘法器。

2.3 AVR 单片机存储器组织

2.3.1 可下载的 Flash 程序存储器

90 系列单片机包括 1 K～128 K 字节的片内可下载 Flash 存储器。由于所有指令为 16 位字或 32 位字,用于存储程序的 Flash 的结构为(512～64K)×16。Flash 存储器的使用寿命最少为 1 000 次写/擦循环。AT90S8515 单片机程序计数器宽为 12 位,以此来对 4K 个程序存储器地址寻址。

常量表必须被设定在 0～4 K 的地址之间(请参考 LPM——装载程序存储器指令说明)。

2.3.2 内部和外部的 SRAM 数据存储器

图 2.8 说明 AT90S8515 单片机的数据存储器组成。

低端 608 个数据存储器地址编址为寄存器文件、I/O 存储器和内部数据 SRAM。前 96 个地址编址为寄存器文件 + I/O 存储器,接下来的 512 个地址是内部数据 SRAM。可选的外部数据 SRAM 可以放置在同一个 SRAM 空间中,该 SRAM 将占据内部 SRAM 之后的地址直到 64K - 1,这由 SRAM 的大小来定。

当访问超出内部数据 SRAM 地址的数据存储器空间时就访问外部数据 SRAM,使用与访问内部数据 SRAM 同样的指令。当访问内部的 SRAM 时,读写控制信号($\overline{RD}$和$\overline{WR}$)在整个访问周期中是非激活的,外部的 SRAM 的操作通过设置 MCUCR 寄存器的 SRE 位来使能,详见后面 MCU 控制寄存器的说明。

访问外部 SRAM 存储器比内部 SRAM 多用一个时钟周期。使用命令 LD、ST、LDS、STS、PUSH、POP 能访问外部 SRAM 存储器。如果堆栈被放置在外部 SRAM 中,则中断程序调用和返回指令将多用 2 个时钟周期。当使用带有等待状态的外部 SRAM 时,外部的 SRAM 将额外花费 4 个时钟周期。

对数据存储器的 5 个不同寻址模式

图 2.8 SRAM 组织

为：直接、带位移的间接、间接、带预减量的间接和带后增量的间接寻址。在寄存器文件中，寄存器 R26～R31 具有间接寻址指针寄存器的特性。间接寻址到达数据地址空间的尽头。

带位移的间接寻址模式的特性为，它可到达由寄存器 Y 和 Z 给出的基本地址的 63 个地址位。当使用自动预减量和后增量的间接寻址模式时，地址寄存器 X、Y 和 Z 被使用，或被增大、减小。

32 个通用寄存器、64 个 I/O 寄存器，以及 AT90S8515 单片机中的 512 字节数据 SRAM 可通过所有这些地址模式被直接访问到。

2.3.3 EEPROM 数据存储器

90 系列单片机包括 64～4K 字节的 EEPROM 存储器。它被组织为一个分开的数据空间，这个数据空间用单字节可被读写。EEPROM 的使用寿命至少为 100 000 次写/擦循环。在 EEPROM 和 CPU 之间的访问详见后面对 EEPROM 的地址寄存器、数据寄存器和控制寄存器的说明。

2.3.4 存储器访问和指令执行时序

本节说明了 90 系列单片机指令执行和内部存储器访问的时序。

AVR CPU 由系统时钟 Φ 驱动，直接由芯片的外部时钟晶振激活，没有使用内部时钟分频。

图 2.9 所示为 Harvard 结构和快速访问寄存器文件概念使能的并行指令存取和指令执行

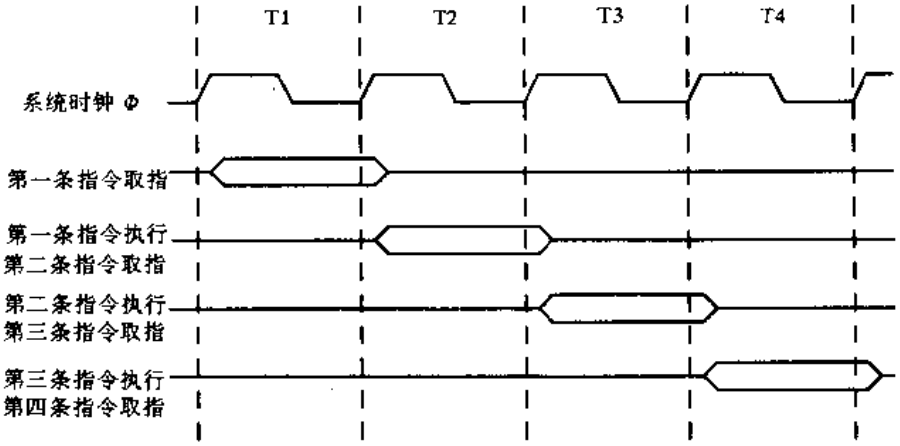


图 2.9 并行指令存取和指令执行

时序。这种基本的流水线概念目的是为了获得高达每 1 MIPS/MHz 的效率。

图 2.10 所示为寄存器文件的内部定时概念。在单一时钟周期内，使用 2 个寄存器操作数的一个 ALU 操作被执行，而其结果被存储回目的寄存器。

图 2.11 所示为在 2 个系统时钟周期内，完成内部数据 SRAM 的访问。

图 2.12 所示为在 2 个系统时钟周期内，完成外部数据 SRAM 的访问。

图 2.13 所示为含有等待状态的(等待状态激活)外部数据 SRAM 的访问时序。

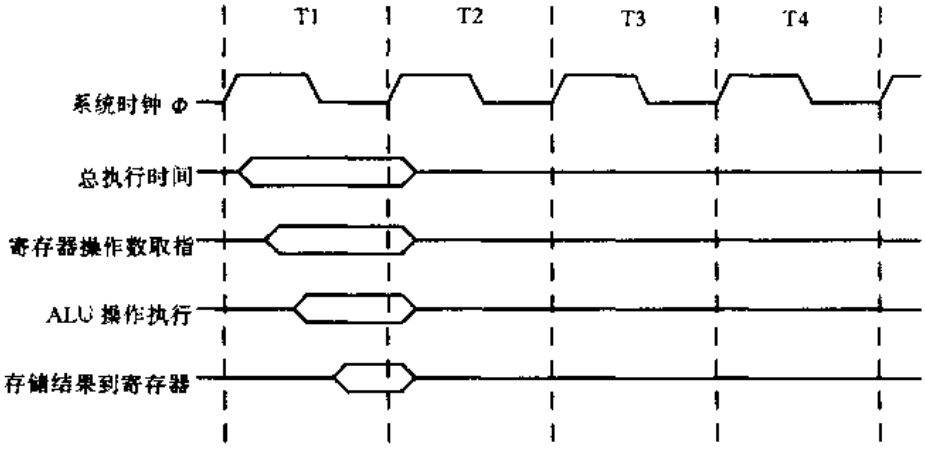


图 2.10 单周期 ALU 操作

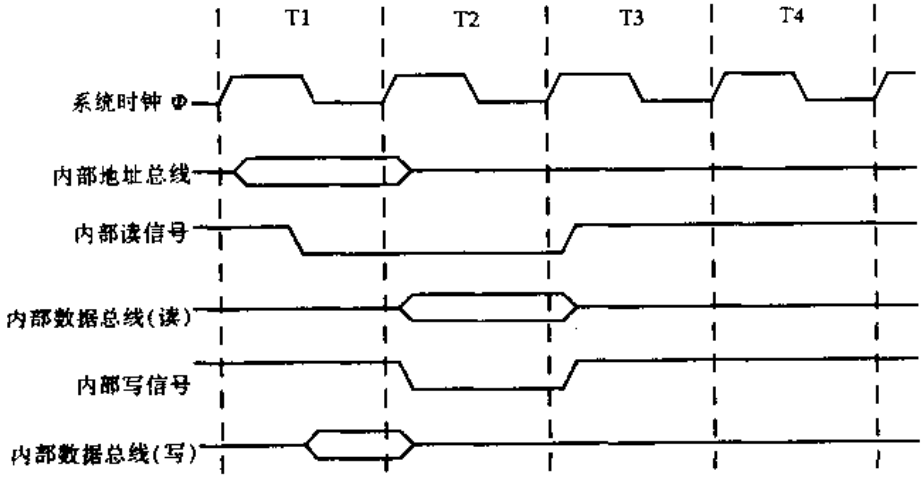


图 2.11 片内数据 SRAM 访问周期

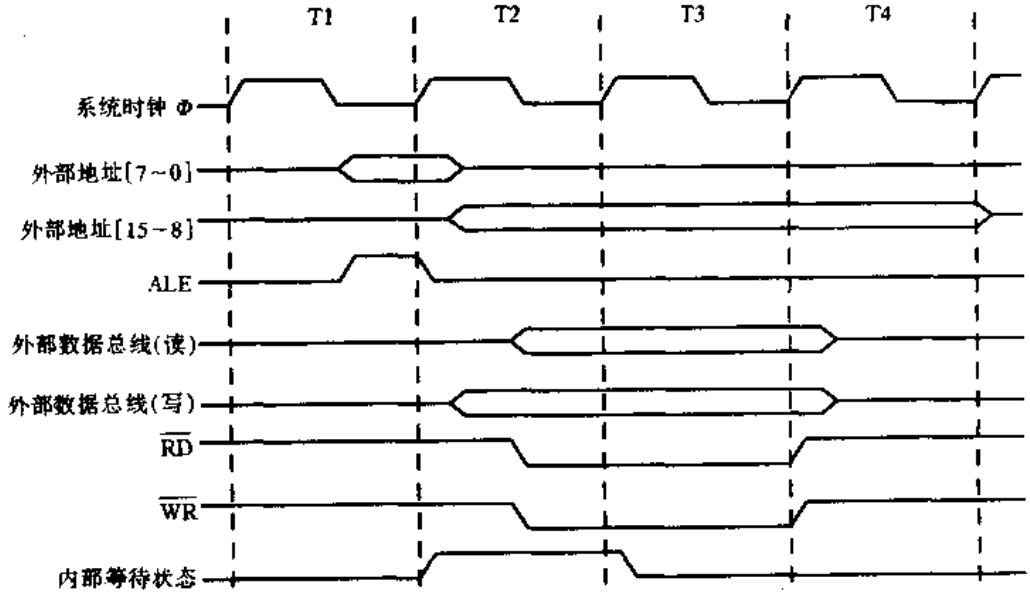


图 2.12 无等待状态的外部数据 SRAM 访问周期

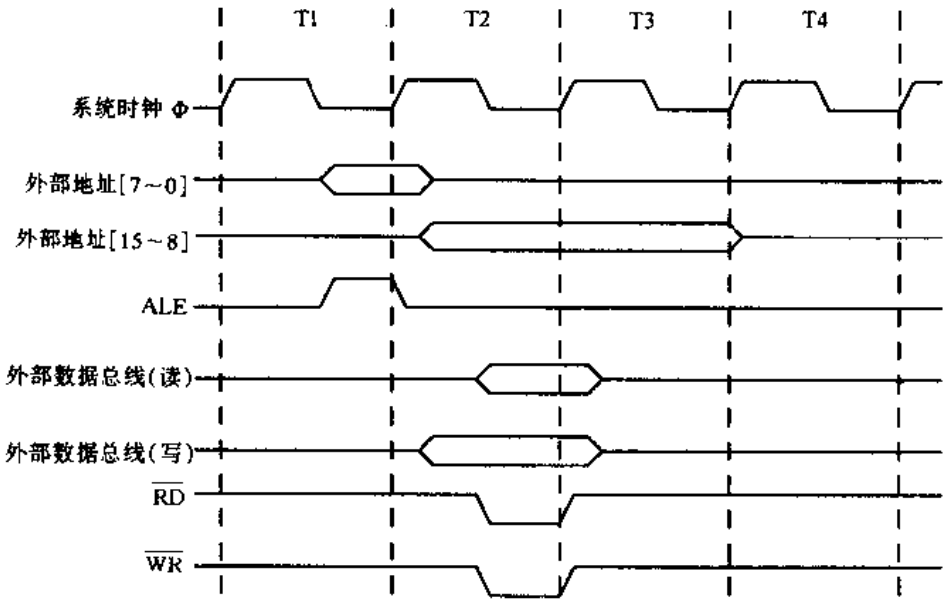


图 2.13 含有等待状态的外部数据 SRAM 访问周期

2.3.5 I/O 存储器

表 2.1 所示为 AT90S8515 单片机的 I/O 空间定义。

表 2.1 AT90S8515 I/O 空间

十六进制地址	名 称	功 能
$3F($5F)	SREG	状态寄存器
$3E($5E)	SPH	堆栈指针高
$3D($5D)	SPL	堆栈指针低
$3B($5B)	GIMSK	通用中断屏蔽寄存器
$3A($5A)	GIFR	通用中断标志寄存器
$39($59)	TIMSK	定时器/计数器中断屏蔽寄存器
$38($58)	TIFR	定时器/计数器中断标志寄存器
$35($55)	MCUCR	MCU 通用控制寄存器
$33($53)	TCCR0	定时器/计数器 0 控制寄存器
$32($52)	TCNT0	定时器/计数器 0(8 位)
$2F($4F)	TCCR1A	定时器/计数器 1 控制寄存器 A
$2E($4E)	TCNT1B	定时器/计数器 1 控制寄存器 B
$2D($4D)	TCNT1H	定时器/计数器 1 高字节
$2C($4C)	TCNT1L	定时器/计数器 1 低字节
$2B($4B)	OCR1AH	定时器/计数器 1 输出比较寄存器 A 高字节
$2A($4A)	OCR1AL	定时器/计数器 1 输出比较寄存器 A 低字节
$29($49)	OCR1BH	定时器/计数器 1 输出比较寄存器 B 高字节
$28($48)	OCR1BL	定时器/计数器 1 输出比较寄存器 B 低字节
$25($45)	ICR1H	T/C1 输入捕获寄存器高字节

表2.1续

十六进制地址	名　称	功　能
$24($44)	ICR1L	T/C1输入捕获寄存器低字节
$21($41)	WDTCR	看门狗定时控制寄存器
$1F($3F)	EEARH	EEPROM地址寄存器高字节
$1E($3E)	EEARL	EEPROM地址寄存器低字节
$1D($3D)	EEDR	EEPROM数据寄存器
$1C($3C)	EECR	EEPROM控制寄存器
$1B($3B)	PORTA	A口数据寄存器
$1A($3A)	DDRA	A口数据方向寄存器
$19($39)	PINA	A口输入脚
$18($38)	PORTB	B口数据寄存器
$17($37)	DDRB	B口数据方向寄存器
$16($36)	PINB	B口输入脚
$15($35)	PORTC	C口数据寄存器
$14($34)	DDRC	C口数据方向寄存器
$13 ($33)	PINC	C口输入脚
$12 ($32)	PORTD	D口数据寄存器
$11($31)	DDRD	D口数据方向寄存器
$10($30)	PIND	D口输入脚
$0F($2F)	SPDR	SPI I/O数据寄存器
$0E($2E)	SPSR	SPI状态寄存器
$0D($2D)	SPCR	SPI控制寄存器
$0C($2C)	UDR	UART I/O数据寄存器
$0B($2B)	USR	UART状态寄存器
$0A($2A)	UCR	UART控制寄存器
$09($29)	UBRR	UART波特率寄存器
$08($28)	ACSR	模拟比较控制和状态寄存器

注意:保留的和未用到的地址未列。

90系列单片机所有不同的I/O口和外围设备均在I/O空间中已设置好。不同的I/O地址可以通过IN和OUT指令访问,这些指令在32个通用寄存器与I/O空间之间传输数据。地址范围$00～$1F之间的寄存器可通过SBI和CBI指令直接一位一位的访问。使用SBIS和SBIC指令对这些寄存器中的每一位值进行检验。请参考指令系统一章。

当使用I/O特定的命令时,必须使用IN、OUT、SBIS和I/O地址的$00～$3F。当寻址的I/O寄存器为SRAM时,必须向该地址加入$20。本资料中的全部I/O寄存器地址均带有列在圆括号中的SRAM地址。以下部分说明了不同I/O和外围设备的控制寄存器。

一、状态寄存器——SREG

AVR的状态寄存器SREG在I/O空间的地址为$3F($5F),定义如下:

位	7	6	5	4	3	2	1	0	
$3F($5F)	I	T	H	S	V	N	Z	C	SREG
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

位 7—I：全局中断使能

全局中断使能位必须被设置(1)，以便允许中断，在这之后，单独的中断使能控制在中断屏蔽寄存器 GIMSK/TIMSK 中完成。如果全局中断使能寄存器被清空(0)，所有中断被禁止，GIMSK/TIMSK 值独立存在。在中断发生后，I 位由硬件清除，并由 RETI 指令设置，从而允许子序列的中断。

位 6—T：位复制存储

位复制指令 BLD 和 BST 使用 T 位作为源操作位和目标操作位。寄存器文件中某一寄存器的某一位可以通过 BST 指令被复制到 T，用 BLD 指令则可将 T 中的位值复制到寄存器文件中的某一寄存器的某一位。

位 5—H：半进位标志位

半进位标志位 H 指示了在一些运算操作过程中的半进位，请参考指令集说明。

位 4—S：标志位，S = N + V

S 位是负数标志位 N 和 2 的补码溢出标志位 V 两者异或值，请参考指令集说明。

位 3—V：2 补码溢出标志位

2 的补码溢出标志位 V 支持 2 的补码运算，请参考指令集说明。

位 2—N：负数标志位

负数标志位 N 指示在不同的运算和逻辑操作之后的负数结果，请参考指令集说明。

位 1—Z：零值标志位

零值标志位 Z 指示在不同的运算和逻辑操作之后的零值结果，请参考指令集说明。

位 0—C：进位标志位

进位标志位 C 指示在某一运算和逻辑操作中的某一进位，请参考指令集说明。

二、堆栈指针——SP

在 I/O 地址 $3E($5E)和 $3D($5D)的两个 8 位寄存器构成了 90 系列单片机的 16 位堆栈指针。由于 AT90S8515 单片机支持 64K 字节的 SRAM，所以所有 16 位都被使用。

位	15	14	13	12	11	10	9	8	
$3E($5E)	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
$3D($5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

堆栈指针指示了数据 SRAM 堆栈区域，子程序和中断堆栈被放置在该区域中。在数据 SRAM 中的该堆栈空间必须在执行任何子程序调用或中断使能之前被程序定义。当执行 PUSH 指令，数据被压入堆栈时，堆栈指针减少 1。当执行子程序 CALL 和中断而将数据压入堆栈时，堆栈指针减少 2。当执行 POP 指令而数据从堆栈弹出时，堆栈指针增大 1。当从子程序 RET 返回或从中断 IRET 返回数据被从堆栈弹出时，堆栈指针增大 2。

2.4 AVR 单片机系统复位

90 系列单片机提供 12 种不同的中断源。这些中断和与之分开的复位向量均在程序存储器空间中各自带有分开的程序向量。所有中断均分配给单独的使能位，这些使能位须与状态

寄存器中的I位一起被设为1,以便能执行中断。

程序存储器空间中最低的地址被自动地定义为复位和中断向量。如表2.2所列,表中还列出了不同中断的优先级。地址越低,优先级越高。$\overline{\text{RESET}}$有最高的优先级,以下依次为INT0…,即外部中断请求0等。

表2.2 复位和中断向量

向量号	程序地址	源	中断定义
1	$000	$\overline{\text{RESET}}$	硬件脚和看门狗复位
2	$001	INT0	外部中断请求0
3	$002	INT1	外部中断请求1
4	$003	TIMER1 CAPT	定时器/计数器1捕获事件
5	$004	TIMER1 COMPA	定时器/计数器1比较匹配A
6	$005	TIMER1 COMPB	定时器/计数器1比较匹配B
7	$006	TIMER1 OVF	定时器/计数器1溢出
8	$007	TIMER0 OVF	定时器/计数器0溢出
9	$008	SPI, STC	串行传送完成
10	$009	UART, RX	UART, RX完成
11	$00A	UART, UDRE	UART数据寄存器空
12	$00B	UART, TX	UART, TX完成
13	$00C	ANA_COMP	模拟比较器

以下为复位和中断向量地址的典型和通用的程序设置:

```
地 址      标 号        代 码              注 释
$000                rjmp    RESET       ;复位处理
$001                rjmp    EXT_INT0    ; IRQ0处理
$002                rjmp    EXT_INT1    ; IRQ1处理
$003                rjmp    TIM1_CAPT   ;定时器1捕获处理
$004                rjmp    TIM1_COMPA  ;定时器1比较A处理
$005                rjmp    TIM1_COMPB  ;定时器1比较B处理
$006                rjmp    TIM1_OVF    ;定时器1溢出处理
$007                rjmp    TIM0_OVF    ;定时器0溢出处理
$008                rjmp    SPI_HANDLE  ; SPI TX处理
$009                rjmp    UART_RXC    ;UART, RX完成处理
$00a                rjmp    UART_DRE    ;UDR空处理
$00b                rjmp    UART_TXC    ;UART, TX完成处理
$00c                rjmp    ANA_COMP    ;模拟比较器处理
;
$00d        MAIN:   <instr>  ×××        ;主程序开始
…                   …       …
```

2.4.1 复位源

90系列单片机有3个复位源:

- 加电复位。当供电电平加至V_{CC}和GND引脚时,MCU进行复位。
- 外部复位。当一个低电平加到$\overline{\text{RESET}}$引脚多于2个XTAL周期时,MCU进行复位。

· 看门狗复位。当看门狗定时器超时，且看门狗为使能时，MCU 进行复位。

在复位过程中，所有的 I/O 寄存器被设为初始值，程序从地址 $000 开始执行。$000 地址中放置的指令须为某一 RJMP——相关转移，即到达复位处理路径的指令。若程序从没有对中断源使能，则中断向量无法使用，正常的程序代码可以放置在这些地址中。图 2.14 的电路图说明了复位逻辑。表 2.3 定义了复位电路的时序和电参数。

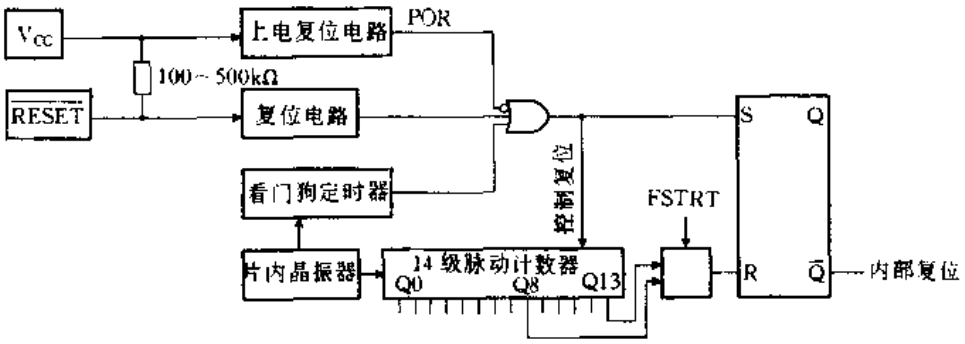


图 2.14　复位逻辑

表 2.3　复位特性(V_{CC} = 5.0 V)

符　号	参　　数	最小值	典型值	最大值	单　位
V_{POT}	上电复位门限电压	1.8	2	2.2	V
V_{RST}	复位脚门限电压		$V_{CC}/2$		V
t_{POR}	上电复位周期	2	3	4	ms
t_{TOUT}	复位延时定时输出 周期 FSTRT 不编程	11	16	21	ms
t_{TOUT}	复位延时定时输出 周期 FSTRT 编程	1.0	1.1	1.2	ms

2.4.2　加电复位

加电复位(POR)电路确保了只有当 V_{CC}达到一个安全电平时，器件才开始工作。如图 2.15 所示，当看门狗定时器晶振对内部定时器定时时，不启动 MCU，直到 V_{CC}到达 Power _ on 门槛电压——V_{POT}一定时间之后才启动 MCU(如图 2.16 和 2.17 所示)。全部的复位时间为 Power _ on 复位时间——t_{POR}加上延时时间 t_{TOUT}。

如果使用了陶瓷谐振器或其它快速启动的晶振来为 CPU 定时，Flash 中的 FSTRT 熔丝位可被编程来给出一个更短的启动时间。由于片内电阻将$\overline{\text{RESET}}$拉高，在无需外部复位时，$\overline{\text{RESET}}$引脚不连接。将$\overline{\text{RESET}}$与 V_{CC}连接，则产生同样效果。在对 V_{CC}加电一段时间后，通过将$\overline{\text{RESET}}$引脚拉低，加电复位时间可以被延长。请参考图 2.17 的时序例子。

2.4.3　外部复位

$\overline{\text{RESET}}$复位引脚上的低电压引发外部复位。该引脚必须被拉低至少 2 个晶振时钟周期，

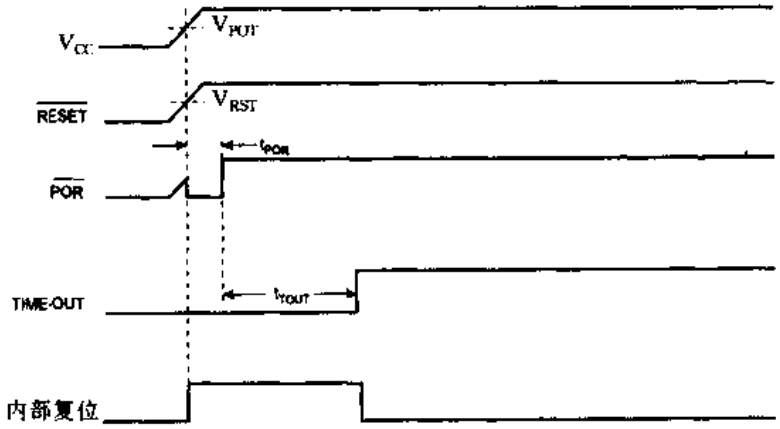


图 2.15 MCU 启动,RESET连或不连到 V_{CC},V_{CC}快速上升

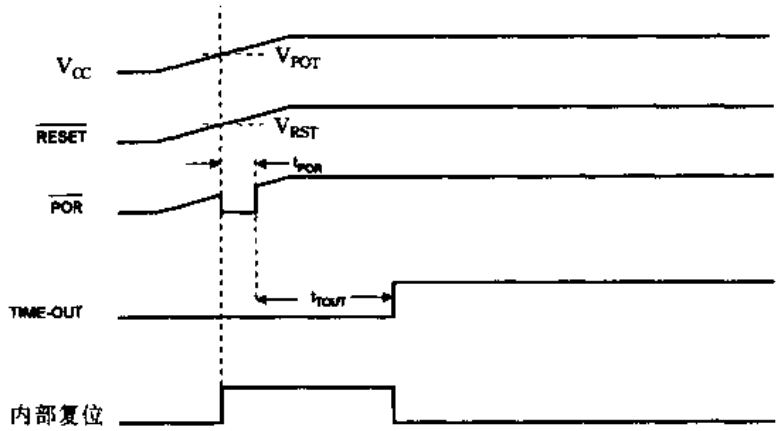


图 2.16 MCU 启动,RESET连或不连到 V_{CC},V_{CC}慢速上升

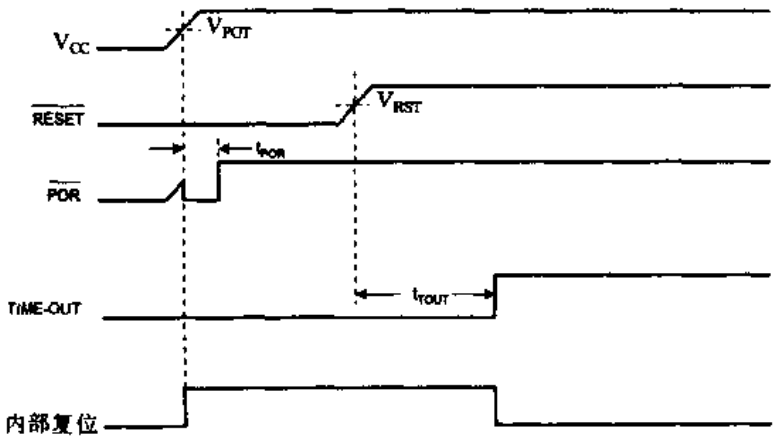


图 2.17 MCU 启动,RESET由外部控制

在 t_{TOUT}周期结束后,当在其正边缘达到复位门槛电压时,延迟定时器启动 MCU。图 2.18 为

在操作期间由外部复位的复位脉冲。

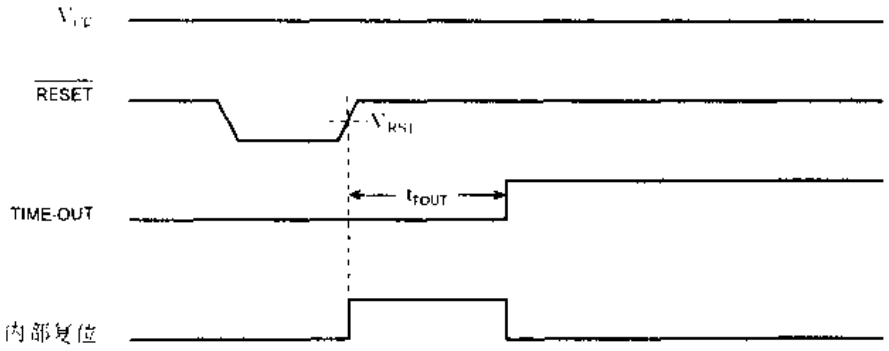


图 2.18 在操作期间由外部复位

2.4.4 看门狗复位

看门狗定时输出时，它将产生一个 XTAL 的短暂复位脉冲，在此脉冲的下降沿，延迟定时器开始对超时时间 t_{TOUT}计数。图 2.19 为在操作期间由看门狗复位的复位脉冲。

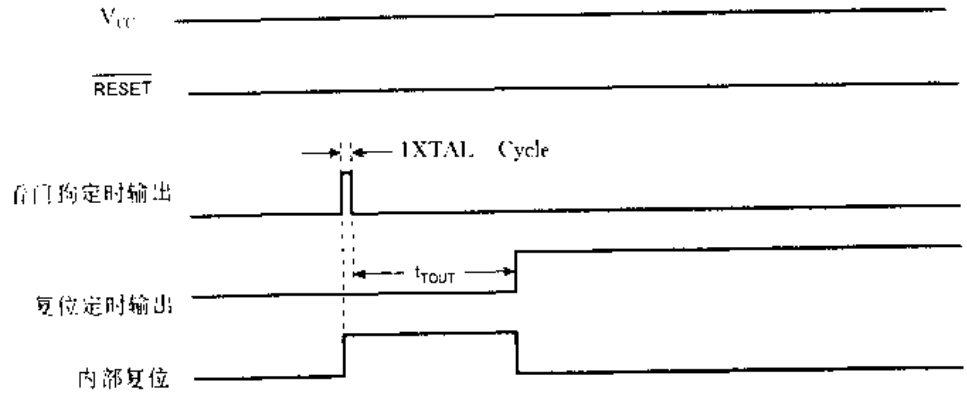


图 2.19 在操作期间由看门狗复位

2.5 AVR 单片机中断系统

2.5.1 中断处理

90 系列单片机有 2 个 8 位中断屏蔽控制寄存器，即 GIMSK——通用中断屏蔽寄存器和 TIMSK——定时/计数中断屏蔽寄存器。

当某一中断发生时，全局中断使能位 I 位被清空(0)，则全部中断被禁止。用户软件可以将 I 位改为 1 来对中断使能。当执行 RETI，从中断指令返回时，I 位被设为 1。

由于维持静态的事件引发中断(即输出比较寄存器 1 与定时器/计数器 1 的值相匹配)，当事件发生时，中断标志被设置。若中断位被清空，且中断条件维持原状，标志直到下次事件发生时才被设置。

当程序计数器指向现行中断时，执行中断处理程序，硬件将相应的产生中断的标志位

清空。

一、通用中断屏蔽寄存器——GIMSK

位	7	6	5	4	3	2	1	0	
$3B($5B)	INT1	INT0	-	-	-	-	-	-	GIMSK
读/写	R/W	R/W	R	R	R	R	R	R	
初始化值	0	0	0	0	0	0	0	0	

位 7—INT1：外部中断请求 1 使能

当 INT1 被置为 1，且状态寄存器(SREG)中 I 位被置为 1 时，外部引脚中断被激活。MCU 通用控制寄存器(MCUCR)中的中断检测控制 1 位 1/0(ISC11 和 ISC10)定义的外部中断，在 INT1 引脚上是在电平还是上升或下降边沿被激活。即使 INT1 被定义为输出，引脚上的事件也将引发一个中断请求。外部中断请求 1 的中断从程序存储器地址 $002 中被执行。

位 6—INT0：外部中断请求 0 使能

当 INT0 被设为 1，且状态寄存器(SREG)中 I 位被设为 1 时，外部引脚中断被激活。MCU 通用控制寄存器(MCUCR)中的中断检测控制 0 位 1/0(ISC01 和 ISC00)定义的外部中断，在 INT0 引脚上是在电平还是上升或下降边沿被激活。即使 INT0 被定义为输出，引脚上的事件也将引发一个中断请求。外部中断请求 0 的中断从程序存储器地址 $001 中被执行。

位 5～0—Res：保留位

90 系列单片机的该位为保留位，总读 0。

二、通用中断标志寄存器——GIFR

位	7	6	5	4	3	2	1	0	
$3A($5A)	INTF1	INTF0	-	-	-	-	-	-	GIFR
读/写	R/W	R/W	R	R	R	R	R	R	
初始化值	0	0	0	0	0	0	0	0	

位 7—INTF1：外部中断标志 1

当 INT1 引脚上的某一事件触发了某一中断请求，INTF1 置为设 1。若 SREG 中的 I 位以及 GIMSK 中的 INT1 位被置 1，MCU 将在地址 $002 向中断向量转移。当中断程序执行时，标志被清空。标志位还可以通过向其写入逻辑 1 来清空。

位 6—INTF0：外部中断标志 0

当 INT0 引脚上的某一事件触发了某一中断请求，INTF0 变为置 1。若 SREG 中的 I 以及 GIMSK 中的 INT0 位被设为了 1，MCU 将在地址 $002 向中断向量转移。当中断程序执行时，标志被清空。标志位还可以通过向其写入逻辑 1 来清空。

位 5～0—Res：保留位

90 系列单片机的该位为保留位，总读 0。

三、定时器/计数器中断屏蔽寄存器——TIMSK

位	7	6	5	4	3	2	1	0	
$39($59)	TOIE1	OCIE1A	OCIE1B	-	TICIE1	-	TOIE0	-	TIMSK
读/写	R/W	R/W	R/W	R	R/W	R	R/W	R	
初始化值	0	0	0	0	0	0	0	0	

位 7—TOIE1：定时器/计数器 1 溢出中断使能

当 TOIE1 被设为 1，且状态寄存器中 I 位被设为 1 时，定时器/计数器 1 溢出中断使能。如定时器/计数器 1 产生溢出，相应的中断(在向量 $006)被执行。在定时器/计数器中断标志寄存器——TIFR 中的溢出标志(定时器 1)被设置为 1。当定时器/计数器 1 处于 PWM 模式下，计数器在 $0000 变化计数方向时，定时器溢出标志被设置。

位 6—OCIE1A：定时器/计数器 1 输出比较匹配 A 中断使能

当 OCIE1A 被设为 1，且状态寄存器中的 I 位被设为 1 时，定时器/计数器的比较匹配 A 中断使能。若发生了定时器/计数器 1 中的比较匹配 A，则中断(在向量 $004)被执行。在定时器/计数器中断标志寄存器——TIFR 中的定时器/计数器 1 中的比较 A 标志被设为 1。

位 5—OCIE1B：定时器/计数器 1 输出比较匹配 B 中断使能

当 OCIE1B 被设为 1，且状态寄存器中的 I 位被设为 1 时，定时器/计数器 1 的比较匹配 B 中断使能。若发生了定时器/计数器 1 中的比较匹配 B，则中断(在向量 $005)被执行。在定时器/计数器中断标志寄存器——TIFR 中的定时器/计数器 1 中的比较 B 标志被设为 1。

位 4，2，0—Res：保留位

90 系列单片机的该位为保留位，总读 0。

位 3—TICIE1：定时器/计数器 1 输入捕获中断使能

当 TICIE1 被设为 1，且状态寄存器中的 I 位被设为 1 时，定时器/计数器 1 输入捕获事件中断使能。若在引脚 31，即 PD6(ICP)上发生捕获触发事件，则中断(在向量 $003)被执行。在定时器/计数器中断标志寄存器——TIFR 中的定时器/计数器 1 输入捕获标志被设置为 1。

位 1—TOIE0：定时器/计数器 0 溢出中断使能

当 TOIE0 被设为 0，且状态寄存器中的 I 位被设为 1 时，定时器/计数器 0 溢出中断使能。若在定时器/计数器 0 上发生溢出，则中断(在向量 $008)被执行。在定时器/计数器的中断标志寄存器——TIFR 中的溢出标志(定时器 0)被设置为 1。

四、定时器/计数器中断标志寄存器——TIFR

位	7	6	5	4	3	2	1	0	
$38($58)	TOV1	OCF1A	OCIFB	-	ICF1	-	TOV0	-	TIFR
读/写	R/W	R/W	R/W	R	R/W	R	R/W	R	
初始化值	0	0	0	0	0	0	0	0	

位 7—TOV1：定时器/计数器 1 溢出标志

当定时器/计数器 1 中产生溢出时，TOV1 位被设为 1。当执行相应中断处理向量时，TOV1 被硬件复位。TOV1 可由写入一个逻辑 1 到标志位而被清除。当 SREG(状态寄存器)的 I 位和 TOIE1(定时器/计数器 1 溢出中断使能)，以及 TOV1 被设为 1 时，定时器/计数器 1 的溢出中断被执行。在 PWM 模式下，当定时器/计数器 1 在 $0000 变化计数方向时，该位被设置。

位 6—OCF1A：输出比较标志 1A

当 OCR1A，即输出比较寄存器 1A 中的数据与定时器/计数器 1 之间发生比较匹配时，OCF1A 被设为 1。当执行相应中断处理向量时，OCF1A 由硬件清除。OCF1A 亦可通过向标志写逻辑 1 来清除。当 SREG 中的 I 位，OCIE1A(即定时器/计数器 1 比较匹配 A 中断使能)，以及 OCF1A 被设为 1 时，定时器/计数器 1 比较匹配中断使能被执行。

位 5—OCF1B：输出比较标志 1B

当 OCR1B，即输出比较寄存器 1B 中的数据与定时器/计数器 1 之间发生比较匹配时，OCF1B 被设为 1。当执行相应中断处理向量时，OCF1B 由硬件清除。OCF1B 亦可通过向标

志写逻辑 1 来清除。当 SREG 中的 I 位，OCIE1B(即定时器/计数器 1 比较匹配 B 中断使能)，以及 OCF1B 被设为 1 时，定时器/计数器 1 比较匹配中断使能被执行。

位 4,2,0—Res：保留位

90 系列单片机的该位为保留位，总读 0。

位 3—ICF1：中断捕获标志 1

当一个输入捕获事件发生时，ICF1 位标志被设为 1。表明定时器/计数器 1 的值已被输送到输入捕获寄存器——ICR1。当执行相应中断处理向量时，ICF1 被硬件清除。ICF1 亦可通过向标志写逻辑 1 来清除。

位 1—TOV0：定时器/计数器 0 溢出标志位

当定时器/计数器 0 中产生溢出时，TOV0 位被设为 1，当执行相应中断处理向量时，TOV0 被硬件清除。TOV0 可由写入一个逻辑 1 到标志位而被清除。当 SREG 的 I 位和 TOIE0(定时器/计数器 0 溢出中断使能)，以及 TOV0 被设为 1 时，定时器/计数器 0 的溢出中断被执行。

2.5.2 外部中断

外部中断由引脚 INT1 和 INT0 触发。请注意，在使能时，即使 INT0 和 INT1 引脚被设置为输出，亦会触发中断。这一特征可用来进行软件中断。外部中断可通过上升或下降边沿及低电平来触发。这在 MCU 控制寄存器——MCUCR 中已说明。当外部中断使能，且被设置为电平触发时，只要引脚保持低电平，中断将被触发。

外部中断的设置已在 MCU 控制寄存器——MCUCR 的规格中被说明。

2.5.3 中断应答时间

AVR 所有允许的中断执行应答最少为 4 个时钟周期。在 4 个时钟周期之后，用于现行中断控制程序的程序向量寻址被执行。在 4 个时钟周期中，程序计数器(2 个字节)被压入堆栈，且堆栈指针被减量 2。该向量为一个向中断程序的相关转移指令，需 2 个时钟周期。若在一个多周期指令的执行中发生中断，该指令在中断执行前结束。

从中断处理程序(像调用子程序)的返回需要 4 个时钟周期。在这 4 个时钟周期之中，程序计数器(2 个字节)从堆栈中被弹出，且堆栈指针被增量 2。当 AVR 从某一中断中出来时，它总是返回到主程序，并在任何挂起的中断执行前执行多于一条的指令。

注意：状态寄存器——SREG 不由 AVR 硬件处理，也不为中断或子程序工作。由于中断处理程序需要一个 SREG 的存储。这一切均由用户软件完成。

由事件发生的中断触发能保留状态，即当事件发生时，中断标志被设置。若中断标志已被清除，但中断条件持续，则在下次事件发生时，中断标志才被设置。

2.5.4 MCU 控制寄存器 MCUCR

MCU 控制寄存器包含通用 MCU 功能的控制位。

位	7	6	5	4	3	2	1	0	
$35($55)	SRE	SRW	SE	SM	ISC11	ISC10	ISC01	ISC00	MCUCR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

位 7—SRE：外部 SRAM 使能

当 SRE 被设置为 1 时，外部的 SRAM 被使能，且引脚 AD0～AD7(A 口)，A8～A15(C 口)，$\overline{RD}$和$\overline{WR}$(D 口)作为第二功能被激活。SRE 位覆盖任何的数据方向寄存器的方向设置，详见“2.3.2 节内部和外部的 SRAM 数据存储器”，该部分描述了外部 SRAM 的引脚功能。当 SRE 位被清除时，外部数据 SRAM 被禁止，作为通常的引脚且用于数据方向的设置。

位 6—SRW：外部 SRAM 等待状态

当 SRW 设置为 1 时，在外部的数据 SRAM 访问周期中插入一个周期的等待状态。当 SRW 被清 0 时，外部数据 SRAM 的访问使用正常的 2 周期方案。详见图 2.12 无等待状态的外部数据 SRAM 访问周期和图 2.13 含有等待状态的外部数据 SRAM 访问周期。

位 5—SE：休眠使能

SE 位须设为 1，以便当 SLEEP 指令执行时，使 MCU 进入休眠模式。除非出于编程的目的，为防止 MCU 进入休眠模式，建议用户只在执行 SLEEP 指令之前才设置休眠使能 SE 位。

位 4—SM：休眠模式

这一位可选择两种休眠模式。当 SM 被清为 0 时，闲置模式被选为休眠模式。当 SE 设为 1 时，掉电模式被选为休眠模式。请参考“2.6.1 节休眠状态”。

位 3,2—ISC11，ISC10：中断检测控制 1 位 1 和位 0

如果 SREG 的 I 标志位和 GIMSK 寄存器中的相应中断屏蔽已被设置，则外部中断 1 被外部引脚 INT1 激活。由表 2.4 定义激活中断的外部 INT1 引脚上的电平和边沿。

位 1,0—ISC01，ISC00：中断检测控制 0 位 1 和位 0

如果 SREG 的 I 标志位和 GIMSK 寄存器中的相应中断屏蔽已被设置，则外部中断 0 被外部引脚 INT0 激活。由表 2.5 定义激活中断的外部 INT0 引脚上的电平和边沿。

表 2.4 中断 1 检测控制

ISC11	ISC10	说　　明
0	0	INT1 的低电平产生一个中断请求
0	1	INT1 的高电平产生一个中断请求
1	0	INT1 的下降沿产生一个中断请求
1	1	INT1 的上升沿产生一个中断请求

表 2.5 中断 0 检测控制

ISC01	ISC00	说　　明
0	0	INT0 的低电平产生一个中断请求
0	1	INT0 的高电平产生一个中断请求
1	0	INT0 的下降沿产生一个中断请求
1	1	INT0 的上升沿产生一个中断请求

注意：当改变 ISC10/ISC00 位时，INT0 须通过清除其在 GIMSK 寄存器中的中断使能位来加以禁止，否则当该位改变时，一个中断就会产生。

2.6 AVR 单片机的节电方式

2.6.1 休眠状态

为了进入休眠状态，MCUCR 中的 SE 位被设为 1，且须执行一条 SLEEP 指令。当 MCU 在休眠模式下发生了一个允许的中断，MCU 唤醒，执行中断程序，并恢复 SLEEP 后面指令的执行。寄存器文件的内容和 I/O 存储器没有改变。若在休眠模式下发生复位，MCU 被唤醒，并从复位向量执行。

注意：若为了从掉电状态下唤醒 MCU 而使用了某一电平触发的中断，必须保持 16 ms 的低电平，即长于晶振启动的时间；否则，在 MCU 开始执行以前，中断标志位有可能返回零值。

2.6.2 闲置模式

当 SM 位被清为 0，SLEEP 指令使 MCU 进入闲置状态，这时 CPU 停止工作，而定时器/计数器、看门狗以及中断系统继续工作。这使得 CPU 可以由外部触发的中断来唤醒，亦可由内部诸如定时器溢出中断和看门狗复位来叫醒，不需要通过模拟比较器中断来唤醒 CPU，这样可通过设置模拟比较器的控制状态寄存器——ACSR 中的 ACD 位来对模拟比较器进行掉电。因此在闲置状态下减少了功耗。

2.6.3 掉电模式

当 SM 位被设为 1 时，SLEEP 指令使 MCU 进入掉电状态。在此模式下，外部晶振停止工作。用户可在掉电模式下选择看门狗是否使能。若看门狗为使能，当看门狗超过超时规定的时间时，它会唤醒 MCU。若看门狗为非使能，只有外部的复位或外部电平触发中断可唤醒 MCU。

2.7 AVR 单片机定时器/计数器

AT90S8515 单片机有 2 种通用定时器/计数器，即一个 8 位的定时器/计数器、一个 16 位的定时器/计数器。定时器/计数器从 10 位的预定比例定时器获取独立的预定比例区域。2 个定时器/计数器可被用作带内部时钟的时基定时器，或被用作带可触发计数的外部引脚连接的计数器。

2.7.1 定时器/计数器预定比例器

图 2.20 所示为通用定时器/计数器的预定比例器。

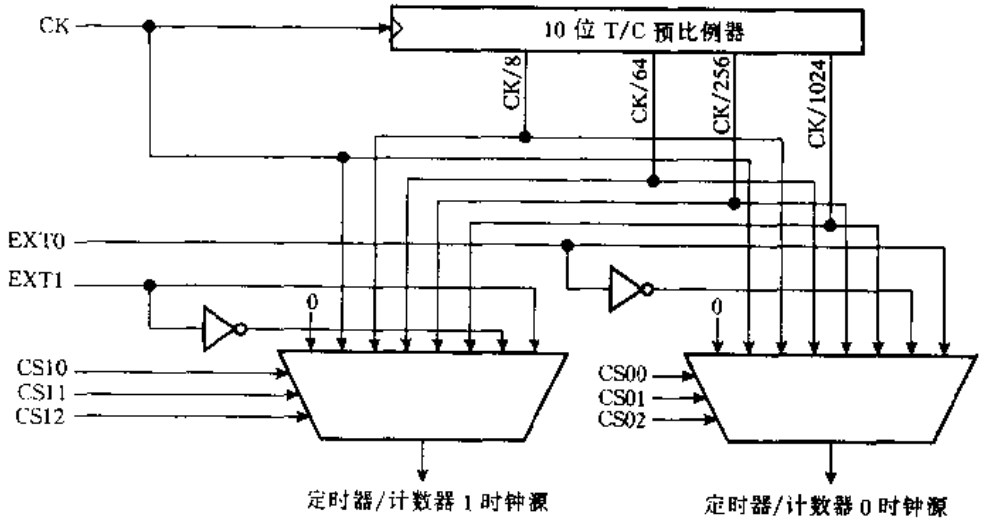


图 2.20 定时器/计数器预定比例器

四个不同的预定比例部分为：CK/8、CK/64、CK/256 和 CK/1 024，图中 CK 为晶振时钟。对于两个定时器/计数器，新加的部分为 CK。外部源和停止均可选为时钟源。

2.7.2　8 位定时器/计数器 0

图 2.21 所示为定时器/计数器 0 的方框图。

8 位的定时器/计数器 0 可从 CK、预定比例器时钟源，或外部引脚来选择时钟源。另外，它还可以像定时器/计数器 0 的控制寄存器——TCCR0 格式中所描述的那样而停止。

溢出状态比例标志位在定时器/计数器中断比例标志位寄存器——TIFR 中。定时器/计数器 0 的中断使能/禁止设置在定时器/计数器中断的控制屏蔽寄存器——TIMSK 中。

当定时器/计数器 0 被外部定时时，外部信号与 CPU 的振荡频率同步。为了确保外部时钟获取正确的采样，在两个外部时钟转换之间的最少时间必须维持一个内部 CPU 的时钟周期。外部时钟信号在内部 CPU 时钟的上升边沿被采样。

8 位的定时器/计数器 0 有机会用于更低的预定比例时，则具有高分辨率和高准确性的特性。类似地，高预定比例特性使得定时器/计数器 0 对低速功能，或不经常操作的精确定时功能很有用。

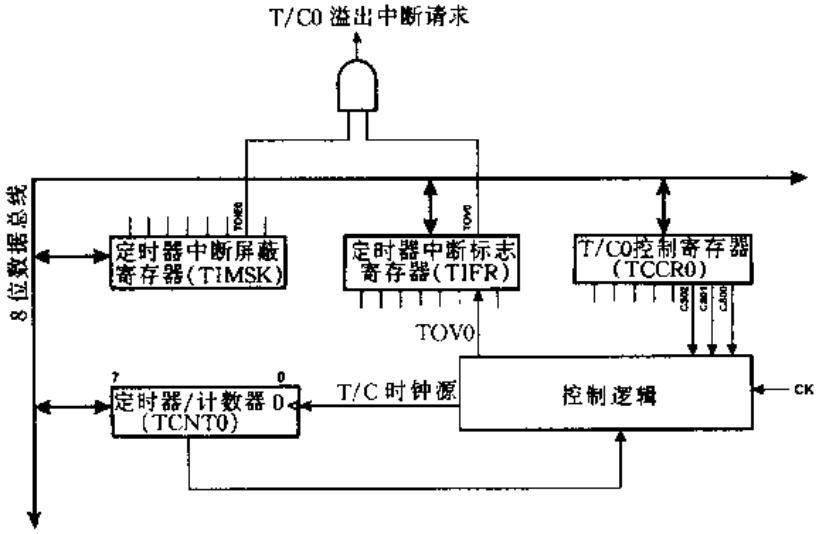


图 2.21　定时器/计数器 0 方框图

一、定时器/计数器 0 的控制寄存器——TCCR0

位	7	6	5	4	3	2	1	0	
$33($53)	–	–	–	–	–	CS02	CS01	CS00	TCCR0
读/写	R	R	R	R	R	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

位 7～3—Res：保留位

90 系列单片机的这些位为保留位，总读 0。

位 2,1,0—CS02，CS01，CS00：时钟选择 0 的位 2、1 和 0

时钟选择 0 的 2、1 和 0 位定义定时器 0 的预定比例源，见表 2.6。

停止条件提供定时器使能/禁止功能。时钟分频的模式从晶振时钟直接换算。若使用外部引脚模式，相应设置必须在实际数据方向控制寄存器中完成。

表 2.6 时钟 0 预定选择

CS02	CS01	CS00	说明	CS02	CS01	CS00	说明
0	0	0	停止,定时器/计数器 0 被停止	1	0	0	CK/256
0	0	1	CK	1	0	1	CK/1 024
0	1	0	CK/8	1	1	0	外部 T0 脚,下降沿
0	1	1	CK/64	1	1	1	外部 T0 脚,上升沿

二、定时器计数器 0——TCNT0

位	7	6	5	4	3	2	1	0	
$32($52)	MSB							LSB	TCNT0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

定时器/计数器 0 是带读写访问的向上计数器。若定时器/计数器 0 被写入,同时时钟源正被执行,定时器/计数器 0 在写入操作之后继续计数。

2.7.3 16 位定时器/计数器 1

图 2.22 所示为定时器/计数器 1 的方框图。

16 位的定时器/计数器 1 可以从 CK、预定比例器时钟源,或外部引脚中选择时钟源。另外,它还可以像在定时器/计数器 1 控制寄存器——TCCR1A、TCCR1B 中描述的那样被停止。在定时器/计数器控制寄存器——TCCR1A、TCCR1B 中可以找到不同的状态标志(溢出、比较匹配以及捕获事件)和控制信号。对定时器/计数器 1 的中断使能/禁止设置可以在定时器/计数器中断屏蔽寄存器——TIMSK 中找到。

当定时器/计数器 1 被外部定时时,外部信号与 CPU 振荡频率同步。为确保从外部时钟获取正确的采样,在 2 个外部时钟转换之间的最小时间至少为一个内部 CPU 时钟周期。外部时钟信号在内部 CPU 时钟的上升边沿被采样。

16 位的定时器/计数器 1 有机会用于更低的预定比例时,则具有高分辨率和高准确性的特性。类似地,高预定比例特性使得定时器/计数器 1 对低速功能,或不经常操作的精确定时功能很有用。

定时器/计数器 1 提供输出比较寄存器 1A 和 1B——OCR1A 和 OCR1B 的两个输出比较功能作为与定时器/计数器 1 内容进行比较的数据源。输出比较功能包括比较 A 匹配的计数器可选择的清除,以及在比较匹配输出比较引脚上的操作。

定时器/计数器可被用作 8 位、9 位或 10 位脉冲调制器。在此模式下,定时器和 OCR1A/OCR1B 寄存器具有集中脉冲双抗误操作独立的 PWM 能力。

定时器/计数器的输入捕获功能提供了对定时器/计数器 1 向输入捕获寄存器——ICR1 内容的捕获,该捕获操作由在输入捕获引脚——ICP 上的外部事件激活。实际的捕获事件设置由定时器/计数器 1 的控制寄存器——TCCR1B 来定义。另外,模拟比较器可触发该输入捕获。请参照"模拟比较器"部分。ICP 的引脚逻辑如图 2.23 所示。定时/计数器 1 的输入捕获噪声消除器示意图见图 2.24。

如果噪音清除器为使能,则捕获事件的实际触发条件在捕获被激活前受到多于 4 个采样的监测。输入引脚信号以 XTAL 的时钟频率被采样。

一、定时器/计数器 1 控制寄存器 A——TCCR1A

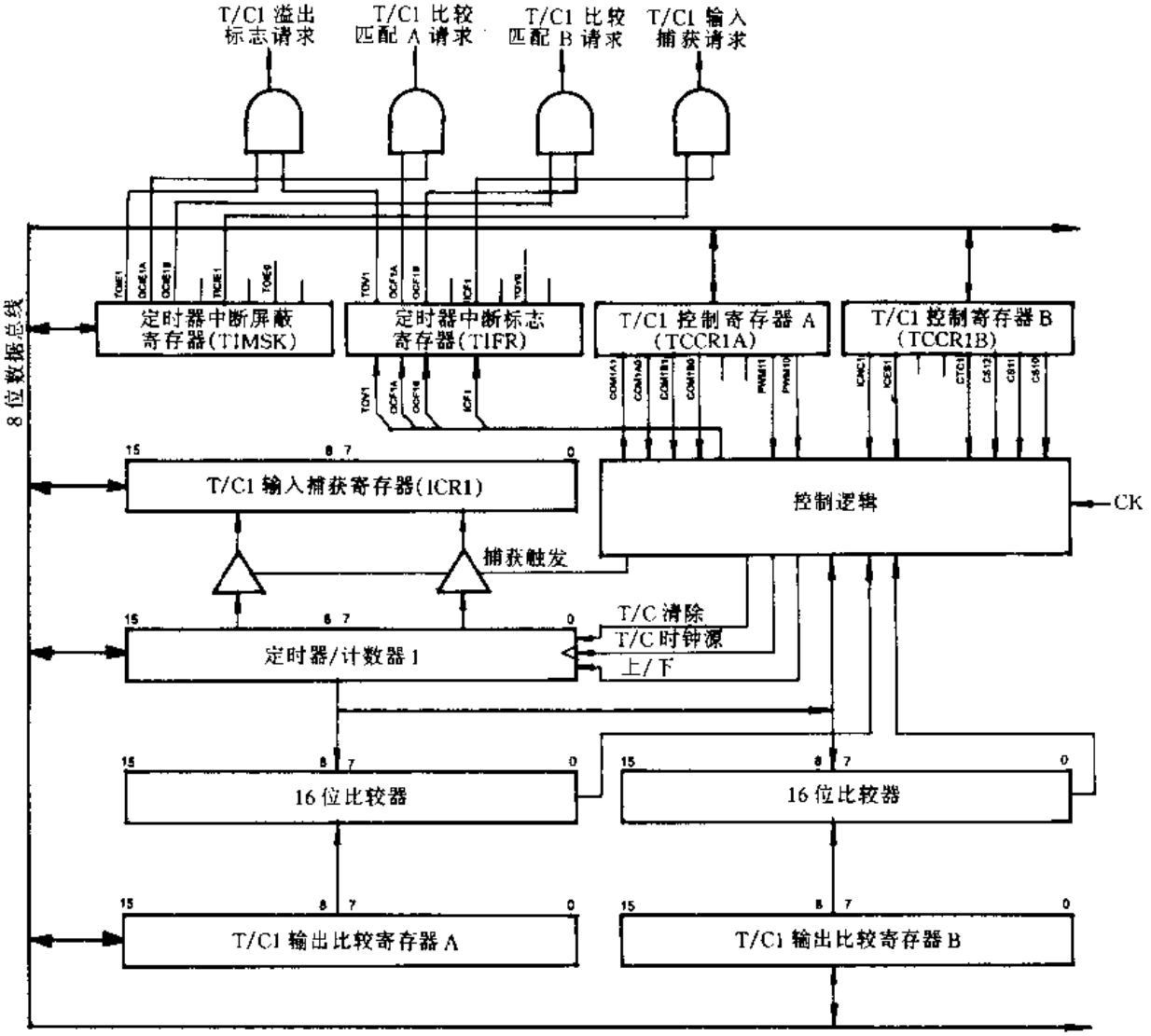


图 2.22 定时器/计数器 1 方框图

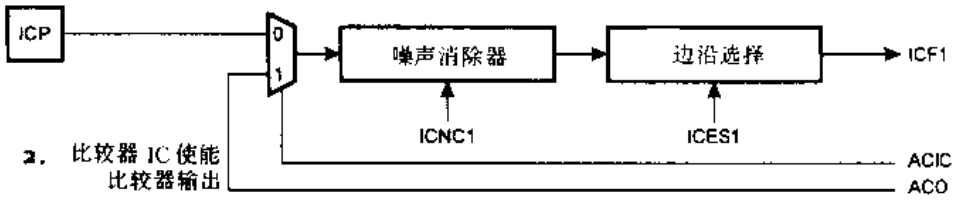


图 2.23 ICP 的引脚原理图

位	7	6	5	4	3	2	1	0	
$2F($4F)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	PWM11	PWM10	TCCR1A
读/写	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

位 7,6—COMM1A1,COM1A0:比较输出模式 1,位 1 和 0

COM1A1 和 COM1A0 控制位决定了在定时器/计数器 1 中比较匹配之后的输出引脚事件。输出引脚事件影响 OC1A,即输出比较 A 引脚 1。由于这是对 I/O 口的可替换功能,相应的方向控制位必须设为 1,以便对输出引脚进行控制。控制设置如表 2.7 所示。

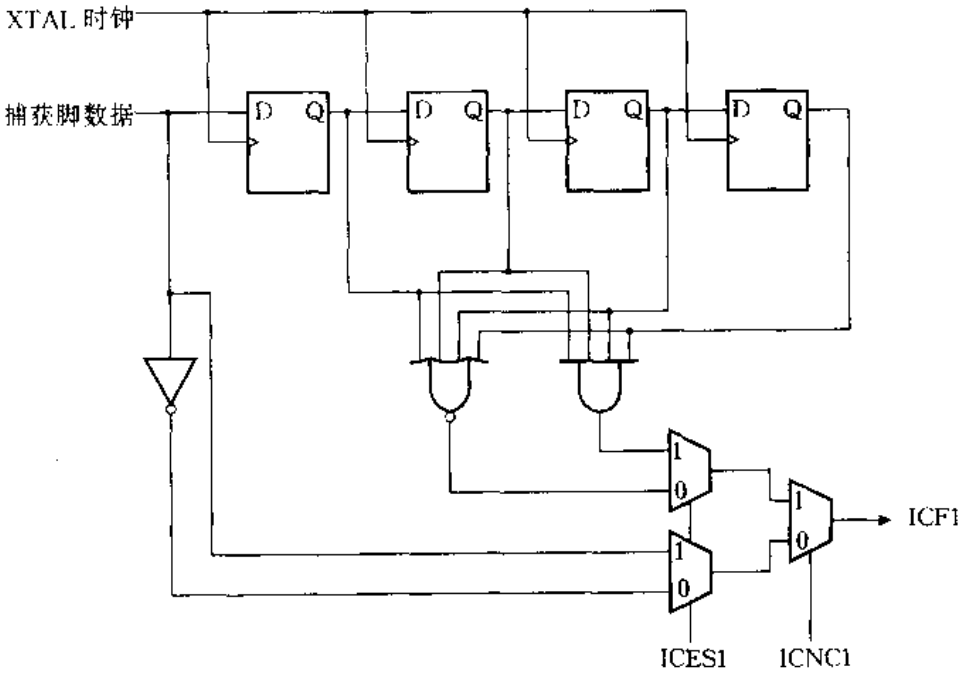


图 2.24 输入捕获噪声消除器示意图

位 5,4—COM1B1, COM1B0：比较输出模式 1,位 1 和 0

COM1B1 和 COM1B0 控制位决定了在定时器/计数器 1 中比较匹配之后的输出引脚事件。输出引脚事件影响 OC1B, 即输出比较 B 引脚 1。由于这是对 I/O 口的可替换功能,相应的方向控制位必须设为 1,以便对输出引脚进行控制。控制设置如表 2.7 所示。

在 PWM 模式下,这些位有不同的功能,请参考表 2.11 的具体说明。

当变换 COM1X1 和 COM1X0 位时,输出比较器中断 1 必须通过清除 TIMSK 寄存器中的中断使能位来禁止;否则在位变换时,会发生中断。

位 3,2—Res：保留位

90 系列单片机的该位为保留位,总读 0。

位 1, 0—PWM11, PWM10：脉冲宽度调制器选择位

这些位如表 2.8 中指明的一样选择定时器/计数器 1 的 PWM 操作。

表 2.7 比较 1 模式选择

COM1X1	COM1X0	说 明
0	0	定时器/计数器 1 与输出脚 OC1X 不连接
0	1	触发 OC1X 输出线
1	0	清除 OC1X 输出线(为 0)
1	1	设置 OC1X 输出线(为 1)

X=A 或 B

表 2.8 PWM 模式选择

PWM11	PWM10	说 明
0	0	禁止定时器/计数器 1 的 PWM 操作
0	1	定时器/计数器 1 为 8 位 PWM
1	0	定时器/计数器 1 为 9 位 PWM
1	1	定时器/计数器 1 为 10 位 PWM

二、定时器/计数器 1 控制寄存器 B——TCCR1B

位	7	6	5	4	3	2	1	0	
$2E($4E)	ICNC1	ICES1	–	–	CTC1	CS12	CS11	CS10	TCCR1B
读/写	R/W	R/W	R	R	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

位 7—ICNC1:输入捕获噪音清除器(4CKs)

当 ICNC1 位被清为 0 时,输入捕获触发噪音清除器功能被禁止。输入捕获在指定的 ICP,即输入捕获引脚上被采样的第一个上升/下降沿处被激活。当 ICNC1 被设为 1,4 个连续的采样成为 ICP,即输入捕获引脚上的测量值,所有的采样须为高/低,取决于 ICES1 位的输入捕获触发特性。实际的采样频率为 XTAL 时钟频率。

位 6—ICES1:输入捕获 1 边沿选择

当 ICES1 位被清为 0,定时器/计数器 1 的内容被传输到输入捕获寄存器——ICR1,即在输入捕获引脚 ICP 的下降边沿。当 ICES1 位被设为 1,定时器/计数器 1 的内容被传输到输入捕获寄存器——ICR1,即在输入捕获引脚 ICP 的上升边沿。

位 5,4 — Res:保留位

90 系列单片机的该位为保留位,总读 0。

位 3— CTC1:在比较匹配上清除定时器/计数器 0

当 CTC1 控制位被设为 1 时,在比较匹配之后,定时器/计数器 1 被复位到时钟周期中的 $0000。若 CTC1 控制位被清除时,定时器/计数器 1 继续计数,直到它被停止、清除、溢出,或被改变方向。在 PWM 模式下,该位无效。

位 2, 1, 0— CS12, CS11, CS10:时钟选择 1 的位 2、1 和 0

时钟选择 1 的位 2、1 和 0 定义了定时器/计数器 1 的预定比例源,见表 2.9

表 2.9 时钟预定比例选择

CS02	CS01	CS00	说 明	CS12	CS11	CS10	说 明
0	0	0	停止,定时器/计数器 1 被停止	1	0	0	CK/256
0	0	1	CK	1	0	1	CK/1 024
0	1	0	CK/8	1	1	0	外部 T1 脚,下降沿
0	1	1	CK/64	1	1	1	外部 T1 脚,上升沿

停止条件提供了定时器使能/禁止功能。时钟分频模式从晶振时钟直接换算。若使用了外部引脚模式,相应设置必须在实际方向控制寄存器中完成(被清除的 0 给出某一输入引脚)。

三、定时器/计数器 1——TCNT1H 和 TCNT1L

位	15	14	13	12	11	10	9	8	
$2D($4D)	MSB								TCNT1H
$2C($4C)								LSB	TCNT1L
	7	6	5	4	3	2	1	0	
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

这个 16 位的寄存器包括 16 位定时器/计数器 1 的预定比例值。为确保当 CPU 访问这些寄存器时,高低字节被同时读写,使用一个 8 位的暂存寄存器(TEMP)来完成访问。

1. TCNT1 定时器/计数器 1 写入

当 CPU 向高位字节 TCNT1H 写入时,写入的数据被放入 TEMP 寄存器中。然后,当 CPU 向低位字节 TCNT1L 写入时,数据的字节被与 TEMP 寄存器中的字节数据组合,且全部的 16 位被同步地向 TCNT1 定时器/计数器 1 寄存器写入。作为结果,高字节的 TCNT1H 必须被先访问,以便完成全 16 位寄存器的写入操作。

2. TCNT1 定时器/计数器 1 读取

当 CPU 读低位字节 TCNT1L 时，TCNT1L 低字节的数据被送到 CPU，且高字节TCNT1H的数据被放置于 TEMP 寄存器中。当 CPU 读高位字节 TCNT1L 时，CPU 接收 TEMP 寄存器中的数据。作为结果，低字节的 TCNT1L 必须先被访问，以便完成全 16 位寄存器的读取操作。定时器 /计数器 1 随着读和写访问，实行向上计数或向上/向下计数(在 PWM 方式)。如果定时器 /计数器已被写入且时钟源已被选择，则定时器 计数器 1 在被设置后的定时时钟周期内连续计数。

四、定时器/计数器 1 输出比较寄存器——OCR1AH 和 OCR1AL

位	15	14	13	12	11	10	9	8	
$2B($4B)	MSB								OCR1AH
$2A($4A)								LSB	OCR1AL
	7	6	5	4	3	2	1	0	
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

五、定时器/计数器 1 输出比较寄存器——OCR1BH 和 OCR1BL

位	15	14	13	12	11	10	9	8	
$29($49)	MSB								OCR1BH
$28($48)								LSB	OCR1BL
	7	6	5	4	3	2	1	0	
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

输出比较器为一个 16 位的读/写寄存器。定时器/计数器 1 输出比较寄存器包括了将要连续地与定时器/计数器 1 相比较的数据。比较匹配的操作在定时器 /计数器 1 的控制和状态寄存器中被区分。

由于输出比较寄存器——OCR1A 和 OCR1B 为一个 16 位的寄存器，当 OCR1A/B 被写入时，须使用临时寄存器 TEMP 来确保全部的字节被同时写入。当 CPU 写高位字节时，OCR1AH 或 OCR1BH 数据被临时地存储在寄存器 TEMP 中。当 CPU 写低位字节时，OCR1AL 或 OCR1BL、TEMP 寄存器被同步地向 OCR1AH 或 OCR1BH 写入。作为结果，高位的 OCR1AH 或 OCR1BH 必须被先写入，以便完成全部的 16 位寄存器写入操作。

六、定时器/计数器 1 输入捕获寄存器——ICR1H 和 ICR1L

位	15	14	13	12	11	10	9	8	
$25($45)	MSB								ICR1H
$24($44)								LSB	ICR1L
	7	6	5	4	3	2	1	0	
读/写	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
初始化值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

输入捕获寄存器为一个 16 位的只读寄存器。当在输入捕获引脚 ICP 上信号的上升或下降边沿(根据输入捕获边沿设置——ICES1)被检测到时,定时器/计数器 1 的当前值被传输到输入捕获寄存器——ICR1。同时,输入捕获标志——ICF1 被设为 1。

由于输入捕获寄存器——ICR1 为一个 16 位的寄存器,当 ICR1 被读出时,使用了一个临时寄存器 TEMP,以便确保全部的字节被同时读出。当 CPU 读取低位字节 ICR1L 时,数据被送入 CPU,且高位字节 ICR1H 的数据被放置在 TEMP 寄存器中。当 CPU 读取高位字节 ICR1H 中的数据时,CPU 接收 TMEP 寄存器中的数据。作为结果,低位字节 1CR1L 必须先被访问到,以便完成一个全 16 位寄存器的读取操作。

七、PWM 模式下的定时器/计数器 1

当选择 PWM 模式时,定时器/计数器 1 以及输出比较寄存器 OCR1A 和输出比较寄存器 OCR1B 形成一个双 8 位、9 位或 10 位的自运行,抗误操作,且在引脚 PD5(OC1A)和 OC1B 引脚上带有输出的节拍修正 PWM。定时器/计数器 1 作为向上/向下的计数器,从 $0000 向上计数到 TOP(参考表 2.10),在重复循环之前,它反转,并向下再次计数到 0。

当计数器值与 OCR1A、OCR1B 的最后 10 位相配时,PD5(OC1A)/OC1B 引脚根据在定时器/计数器 1 控制寄存器 TCCR1A 中 COM1A1/COM1A0 或 COM1B1/COM1B0 位的设置而被设置或被清除,请参考表 2.11。

表 2.10　定时器 TOP 值和 PWM 频率

PWM 分辨率	定时器 TOP 值	频　率
8 位	$00FF(255)	$f_{TCI}/510$
9 位	$01FF(511)	$f_{TCI}/1022$
10 位	$03FF(1 023)	$f_{TCI}/2046$

表 2.11　在 PWM 方式时比较 1 方式选择

COM1X1	COM1X0	在 OCX1 上的作用
0	0	不连接
0	1	不连接
1	0	清比较匹配,向上计数,置比较匹配,向下计数(PWM 不翻转)
1	1	清比较匹配,向下计数,置比较匹配,向上计数(PWM 翻转)

X = A or B

注意:在 PWM 模式下,当后 10 位 OCR1A/OCR1B 位被写入时,它们被送入临时地址。当定时器/计数器 1 到达 TOP 时,它们被锁存。这就防止了在非同步 OCR1A/OCR1B 写入事件中发生奇数长的 PWM 脉冲(误操作)。见图 2.25 的例子。

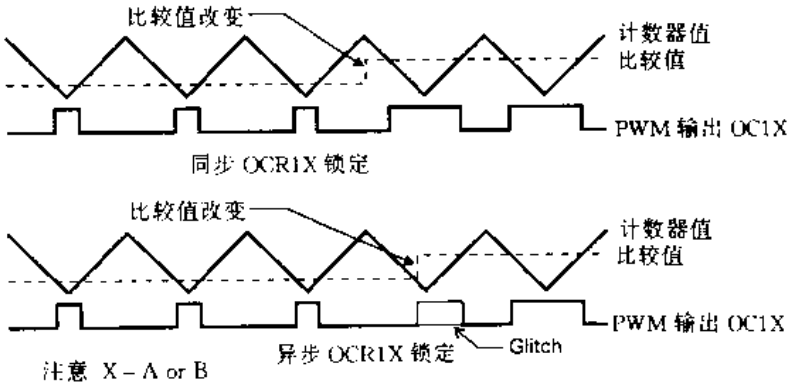


图 2.25　有效的非同步 OCR1 锁存

当 OCR1A 包含 $0000 或 TOP,输出 OC1A/OC1B,根据 COM1A1 和 COM1A0 或 COM1B1/COM1B0 的设置保持低或高,如表 2.12 所示。

在 PWM 模式下，定时器溢出标志 1、TOV1，当计数器在方向 $0000 时被设置。定时器溢出中断 1 以正常的定时器/计数器模式工作，比如，当 TOV1 被设置，从而提供了定时器溢出中断 1 和全局中断为使能时，它被执行。这也同样用于定时器输出比较 1 的标志和中断。

表 2.12 PWM 输出 OCR1X 等于 $0000 或 TOP

COM1X1	COM1X0	OCR1X	OC1X 输出
1	0	$0000	L
1	0	TOP	H
1	1	$0000	H
1	1	TOP	L

2.7.4 看门狗定时器

一、看门狗定时器

看门狗定时器由片内一个独立的 1 MHz 分开的晶振定时。通过控制看门狗定时器的预定比例器，看门狗的复位间歇从 16 周期到 2 048 周期之间调整。这些值适应 $V_{CC}=5$ V。请参考其它 V_{CC}电压下的 RC 晶振频率的特性化数据。WDR，即看门狗复位指令对看门狗定时器进行复位。有 8 种不同的时钟周期可供选择，来决定在 2 个 WDR 指令之间的最大时间，这样做是为了避免看门狗定时器对 MCU 进行复位。若复位周期结束而无其它的 WDR 指令，90 系列单片机进行复位，并从复位向量执行。参见图 2.26。

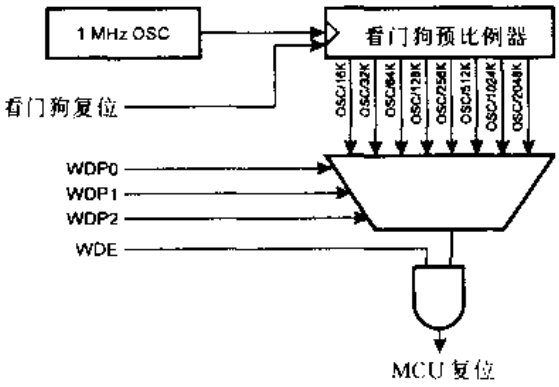


图 2.26 看门狗定时器

为了防止意外禁止看门狗定时器，当看门狗被禁止时必须服从一个特定的关断顺序，请详见看门狗的控制寄存器。

二、看门狗定时器控制寄存器——WDTCR

位	7	6	5	4	3	2	1	0	
$21($41)	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	WDTCR
读/写	R	R	R	R/W	R/W	R/W	R/W	R/W	R
初始化值	0	0	0	0	0	0	0	0	

位 7～5—Res：保留位

90 系列单片机的这些位为保留位，总读 0。

位 4—WDTOE：看门狗关断使能

当 WDE 被清除时该位必须被置 1，否则看门狗将不会被禁止，一旦置位后，硬件将在 4 个时钟周期后清除该位。请参照看门狗禁止过程的 WDE 位的描述。

位 3—WDE：看门狗使能

当WDE被设为1时，看门狗定时器使能。若WDE被清为0，看门狗定时器功能被禁止。WDE 仅在 WDTOE 位设置时被清除，为了禁止被使能的看门狗定时器，必须遵守以下过程：

(1) 在同一个操作中，把 WDTOE 和 WDE 写成 1，即使在禁止操作开始前 WDE 为 1，也必须把 1 写入 WDE。

(2) 在随后 4 个机器周期中，把 WDE 写为 0，这会禁止看门狗。

位 2～0—WDP2～0：看门狗定时器预定比例器 2、1 和 0

WDP2、WDP1、WDP 0 决定了当看门狗定时器使能时，看门狗定时器的预定比例。不同

的预定比例值以及它们相应的超时时间，如表 2.13 所示。

表 2.13 看门狗定时器预定比例选择(典型值 V_{CC} = 5V)

WDP2	WDP1	WDP0	定时器输出周期/ms	WDP2	WDP1	WDP0	定时器输出周期/ms
0	0	0	16	1	0	0	256
0	0	1	32	1	0	1	512
0	1	0	64	1	1	0	1 024
0	1	1	128	1	1	1	2 048

2.8 AVR 单片机 EEPROM 读/写访问

I/O 空间中可以访问 EEPROM 寄存器。

写入访问时间在 2.5～4 ms 之间。取决于 V_{CC}电压。自定时功能使得用户软件检测下一个字节何时被写入。若 V_{CC}低于一定的电平，EEPROM 降低电压被检测，防止了对 EEPROM 的写入。当 EEPROM 被读取或写入时，在下一个指令执行前，CPU 被中止达两个时钟周期。

一、EEPROM 地址寄存器——EEAR

位	15	14	13	12	11	10	9	8	
$1F($3F)	-	-	-	-	-	-	-	EEAR9	EEARH
$1E($3E)	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
	7	6	5	4	3	2	1	0	
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	8	0	0	0	0	0	0	0	
	8	0	0	0	0	0	0	0	

EEPROM 地址寄存器——EEARH 和 EEARL，指定了在 512 字节的 EEPROM 空间中的 EEPROM 地址。EEPROM 数据字节被在 0～511 之间线性地编址。

二、EEPROM 数据寄存器——EEDR

位	7	6	5	4	3	2	1	0	
$1D($3D)	MSB							LSB	EEDR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

位 7～0—EEDR7～0：EEPROM 数据

对于 EEPROM 写入操作，EEDR 寄存器包含了写入 EEPROM 的数据，由 EEAR 寄存器给出其地址。对于 EEPROM 的读取操作，EEDR 包含由 EEAR 给出的 EEPROM 地址，数据将从这一地址中读出。

三、EEPROM控制寄存器——EECR

位	7	6	5	4	3	2	1	0	
$1C($3C)	-	-	-	-	-	EEMWE	EEWE	EERE	EECR
读/写	R	R	R	R	R	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

位7~3—Res：保留位

90系列单片机的这些位为保留位，总读0。

位2—EEMWE：EEPROM的主写使能

EEMWE位决定了设置EEWE是否导致EERPOM被写入。当EEMWE被设为1时，设置EEWE将把数据写入EEPROM所选择的地址中。如果EEMWE为0，则设置EEWE无效。当EEMWE被软件设置后，4个周期后被硬件清除。详见EEPROM写过程中的EEWE位的描述。

位1—EEWE：EEPROM写入使能

EEPROM写入使能信号EEWE是对EEPROM的写入选通。若地址和数据被正确设置，EEWE位必须被设置从而写EEPROM。当EEWE被置1时，EEMWE必须被置1，否则不会发生EEPROM的写操作。当写入EERPOM时应服从以下的过程(第(2)步和第(3)步不是必须的)：

(1) 等待EEWE位变为0；

(2) 把新的EEPROM地址写入EEAR(可选)；

(3) 把新的EEPROM数据写入EEDR(可选)；

(4) 在EECR中的EEMWE位写逻辑1；

(5) 在设置EEMWE后的4个时钟周期内，在EEWE中写入逻辑1。

在写入访问时间(一般为5 V时2.5 ms，2.7 V时4 ms)过后，EEWE由硬件清零。用户软件在写入下一个字节之前，查询这一位，并等待零值。当EEWE被设过后，CPU在执行下一个指令前中止2个周期。

位0—EERE：EEPROM读取使能

EEPROM读取使能信号EERE是对EEPROM的读取选通。当EEAR寄存器中的地址被正确设置时，EERE必须被设置。当EEAR被硬件清空时，在EEDR寄存器中可找到所需数据。EEPROM读取访问占用1个指令，无需查询EERE位。一旦EERE被设过后，CPU在执行下一个指令前中止2个周期。在开始读操作之前，用户可以查询EEWE位。如果当新的数据或地址被写到EEPROM I/O寄存器时，一个写入操作在进行，则写入操作将被中断，并且结果是定义的。

2.9 AVR单片机串行接口

2.9.1 同步串行接口SPI

同步串行接口(SPI)允许在90系列单片机和外设或几个90系列单片机之间高速同步数据传送，见图2.27。90系列单片机SPI的特征如下：

(1) 全双工，3线同步数据传送。

(2) 主从操作。

(3) 5Mb/s的位传送频率(最大值)。

(4) LSB或MSB在先。

(5) 四种可编程的位速率。

(6) 传送停止的中断标志。

(7) 写冲突标志保护。

(8) 从闲置模式下唤醒(仅从模式)。

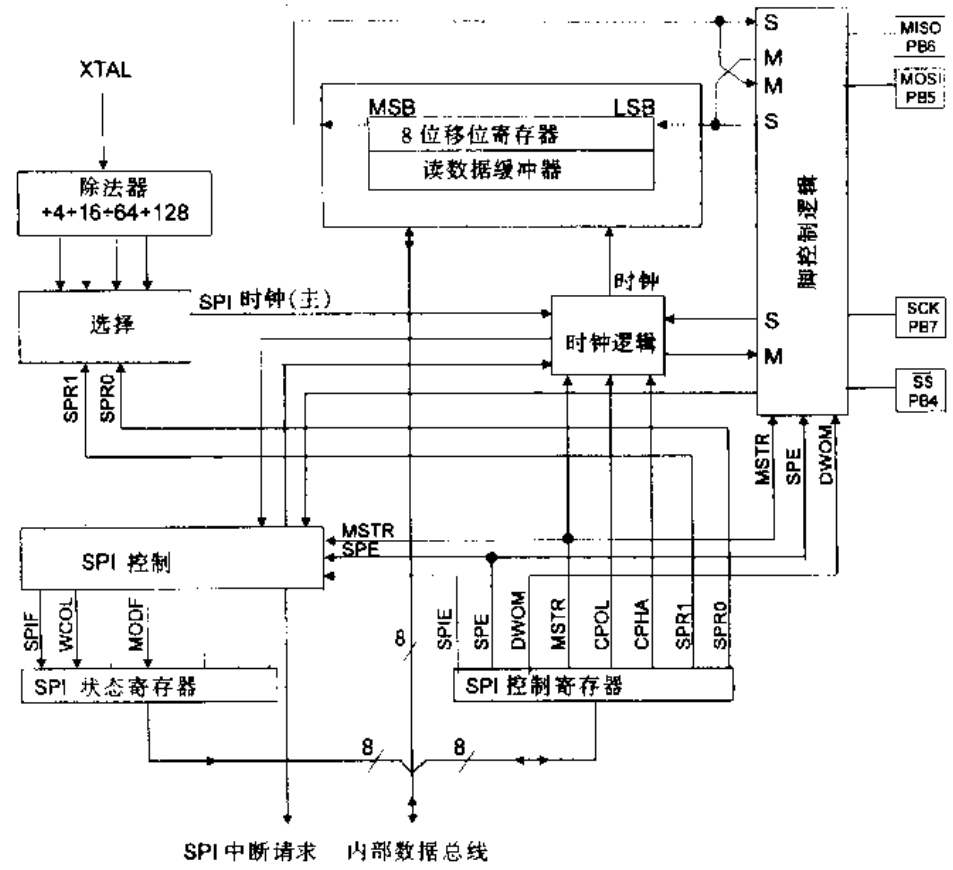


图 2.27 SPI 方框图

主从 CPU 之间的 SPI 连接如图 2.28 所示,PB7(SCK)引脚是主机模式的时钟输出和从机模式的时钟输入,把数据写入主 CPU 的 SPI 数据寄存器会启动 SPI 的时钟发生器,而数据从 PB5(MOSI)引脚移出和移入。在一个字节移出后,SPI 时钟发成器停止,设置传送停止标志位(SPIF)。如果 SPCR 寄存器中的中断使能位(SPIE)被设置,则生成一个中断请求。从机

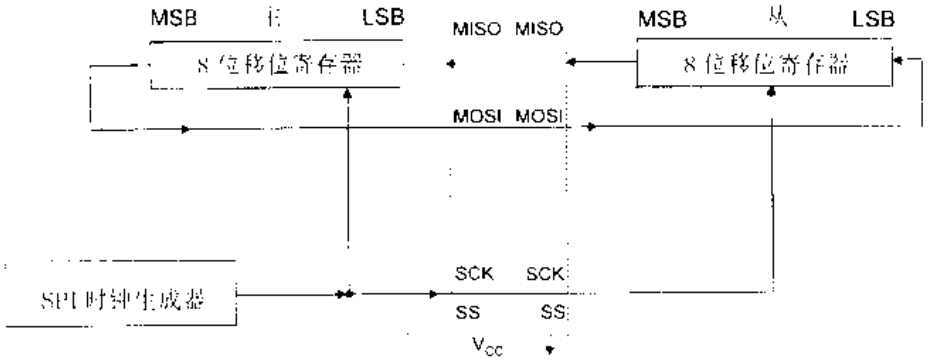


图 2.28 SPI 主从 CPU 内部连接

选择输入 PB4($\overline{SS}$),被设置为低来选择单独的 SPI 器件作为从机,主机和从机的两个移位寄存器可以被认为是一个分开的 16 位环形移位寄存器,如图 2.28 所示。当数据从主机移向从机,同时数据也移向相反的方向。这意味着在一个移位周期内,主机和从机的数据交换。

这个系统在发送方向上有一级缓冲而在接收方向有两级缓冲。这意味在全部的移位周期完成之前要被传送的字符不能被写入 SPI 数据寄存器。在接收数据时,在下一个字符被完全移入之前已经收到的数据必须从 SPI 数据寄存器中读走,否则,这个字符会丢失。

当 SPI 被使能时,MOSI、MISO、SCK 和$\overline{SS}$引脚的数据方向,按照表 2.14 中来配置。

表 2.14 SPI 脚配置

引脚	方向,主 SPI	方向,从 SPI	引脚	方向,主 SPI	方向,从 SPI
MOSI	用户定义	输入	SCIK	用户定义	输入
MISO	输入	用户定义	$\overline{SS}$	用户定义	输入

一、$\overline{SS}$引脚的功能

当 SPI 被配置为主机时(SPCR 的 MSTR 置 1),用户可以决定$\overline{SS}$引脚的方向。如果$\overline{SS}$引脚被设为输出,该引脚作为通用输出不影响 SPI 系统;如果$\overline{SS}$被设为输入,则必须保持为高来保证主机 SPI 的操作。如果,在主机模式下,$\overline{SS}$引脚为输入,而且被外设电路置低,则该系统认为另外的主机选择该 SPI 为它的从机并开始对它传递数据。为了防止总线相连,SPI 系统将采取以下动作:

(1) SPCR 的 MSTR 位被清除而且 SPI 系统变成从机,结果是 MOSI 和 SCK 引脚变成输入。

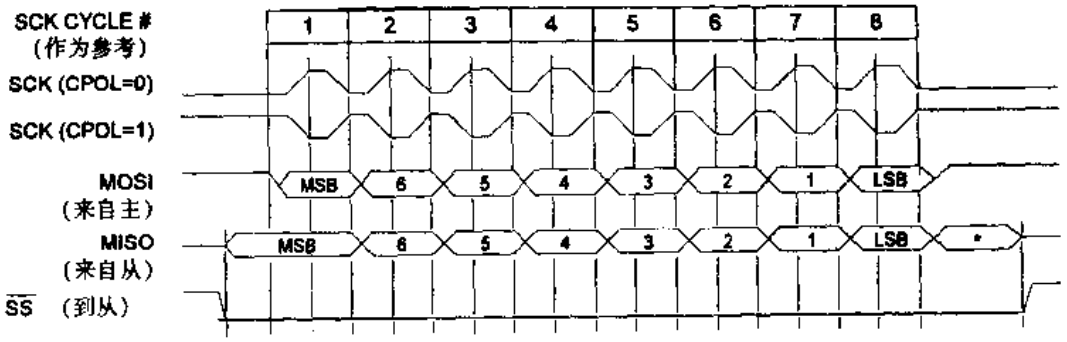
(2) SPSR 中的 SPIF 位被设置,SPI 中断被使能,中断程序被执行。

因此在主机模式下使用中断驱动的 SPI 发送时,存在$\overline{SS}$被置低的可能,中断应检查 MSTR 位是否被设置,一旦发现 MSTR 位被从机清 0,则必须被用户再设置。

当 SPI 被配置为从机时,$\overline{SS}$引脚应为输入。当$\overline{SS}$被置低时,SPI 被激活且 MISO 变为输出(如果被用户配置为输出的话)。在$\overline{SS}$被置低后,其余引脚都为输入。所有引脚都为输入,而且 SPI 是被动的,这意味着它不会接收输入的数据。

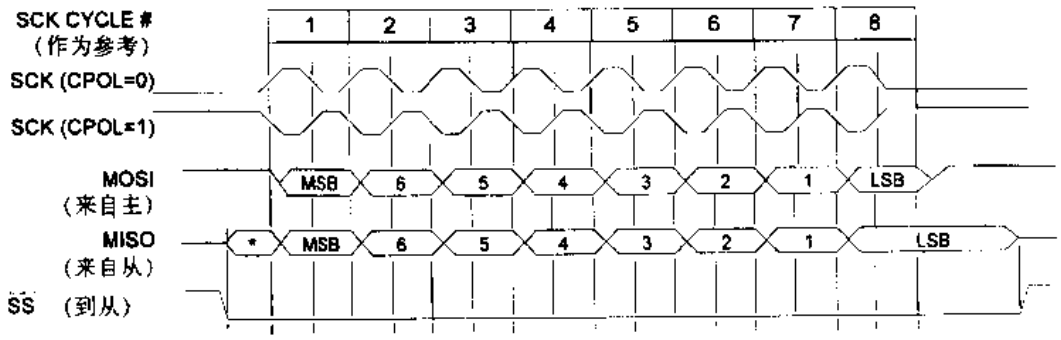
二、数据模式

相应于串行数据,SCK 相和极性有 4 种组合,由控制位 CPHA 和 CPOL 来决定。SPI 的传送格式如图 2.29 和 2.30 所示。



* 不定义,但在接收时字符的最高有效位正常。

图 2.29 当 CPHA=0 时,SPI 传送格式



* 不定义，但在发送时，先前的最低有效位正常。

图 2.30 当 CPHA = 1 时，SPI 传送格式

三、SPI 控制寄存器——SPCR

位	7	6	5	4	3	2	1	0	
$0D($2D)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	1	0	0	

位 7—SPIE：SPI 中断使能

如果全局中断使能，该位导致设置 SPSR 寄存器的 SPIF 位来执行 SPI 中断。

位 6—SPE：SPI 使能

当该位设置时，SPI 使能，要使能 SPI 任何操作必须设置该位。

位 5—DORD：数据的顺序

当 DORD 位被置 1 时，数据的 LSB(低位)被首先传送。

当 DORD 位被置 0 时，数据的 MSB(高位)被首先传送。

位 4—MSTR：主机/从机选择

当设置为 1 时选择主机 SPI 模式，当设置为 0 时选择从机 SPI 模式。如果$\overline{SS}$被设置为输入且在 MSTR 被设置时被置低，则 MSTR 将被清除。而当 SPSR 中的 SPIF 位被设置，用户应该设置 MSTR，再使能 SPI 主机模式。

位 3—CPOL：时钟极性

当该位被置 1 时，SCK 在闲置时是高电平；当 CPOL 被清 0 时，SCK 在闲置时是低电平，详见图 2.29 和图 2.30。

位 2—CPHA：时钟相位

该位的功能详见图 2.29 和图 2.30。

位 1，0—SPR1，SPR0：SPI 时钟速率选择 1 和 0

这两位控制主机模式下器件 SCK 的速率，SPR1 和 SPR2 对于从机无影响，SCK 和振荡器频率 f_{CL}之间的关系如表 2.15 所示。

表 2.15 SCK 和振荡器频率之间关系

SPRI	SPRO	SCK 频率	SPRI	SPRO	SCK 频率
0	0	$f_{CL}/4$	1	0	$f_{CL}/64$
0	1	$f_{CL}16$	1	1	$f_{CL}/128$

四、SPI的状态寄存器——SPSR

位	7	6	5	4	3	2	1	0	
$0E($2E)	SPIF	WCOL	–	–	–	–	–	–	SPSR
读/写	R	R	R	R	R	R	R	R	
初始化值	0	0	0	0	0	0	0	0	

位7—SPIF:SPI中断标志位

当串行传送完成时,SPIF位被设置1,且若SPCR中的SPIE被设置1和全局中断使能,则生成中断。如果$\overline{SS}$被设置为输入且在SPI是主机模式时被置低,这将设置SPIF标志。SPIF位在执行相应中断向量时被硬件清除。可选的,在首次读SPI状态寄存器时,如果SPIF为1,则先清除它再访问SPI数据寄存器(SPDR)。

位6—WCOL:写冲突位

如果在数据传送中SPI数据寄存器(SPDR)被写入。则设置WCOL位。在数据传送中,SPDR寄存器读出的结果是不正确的,写入也没有反应。在首次读SPI状态寄存器时,如果WCOL为1,则先清除它再访问SPI数据寄存器。

位5~0—保留位

在90S8515中这几位保留,读出为0。

90系列单片机的SPI接口也被用于程序存储器和数据EEPROM的编程下载和上载,详见串行编程和校验部分。

五、SPI数据寄存器——SPDR

位	7	6	5	4	3	2	1	0	
$0F($2F)	MSB							LSB	SPDR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

SPI数据寄存器可以读/写,用于在寄存器文件和SPI移位寄存器之间传递数据。写入该寄存器时初始化数据传送,读该寄存器时读到的是移位寄存器接收缓冲区的值。

2.9.2 通用串行接口UART

90系列单片机带有一个全双工的通用串行异步收发器(UART),主要特征如下:(1)波特率发生器可以生成任何波特率。

(2)在XTAL低频率下有高的波特率。

(3)8位和9位数据。

(4)噪声滤波。

(5)超越误差的探测。

(6)帧错误探测。

(7)错误起始位的探测。

(8)三个独立的中断,TX完成,TX数据寄存器为空,RX完成。

一、数据传送

UART传送器的方框示意图见图2.31。数据传送通过把被传送的数据写入UART I/O

寄存器 UDR 来初始化，在以下情况数据从 UDR 传送到移位寄存器中。

(1)当前一个字符的停止位被移出后新的字符被写入 UDR 寄存器，移位寄存器立即再被装入；

(2) 当前一个字符的停止位被移出前新的字符被写入 UDR 寄存器，移位寄存器在当前字符的停止位移出后被装入。

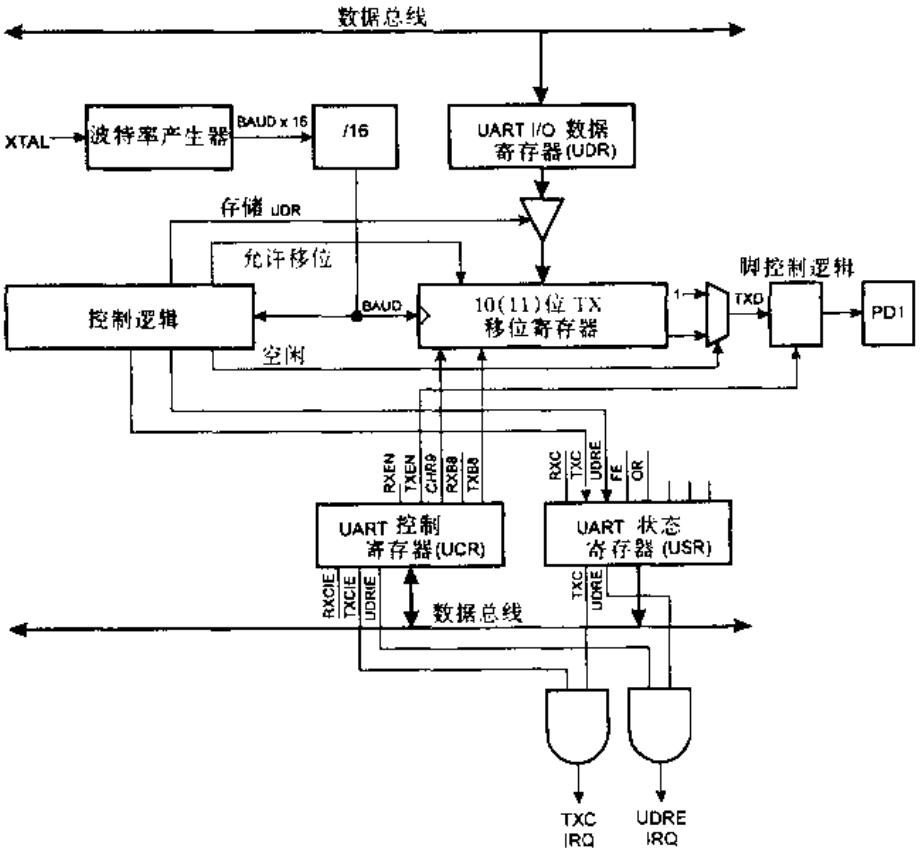


图 2.31 UART 传送器方框示意图

如果 10(11)位传送移位寄存器是空的，或当数据从 UDR 中传送到移位寄存器时，UART 状态寄存器，USR 的 UDRE 位(UART 状态寄存器空)被设置。当这位被设置为 1 时，UART 准备接收下一个字符；当数据从 UDR 传送到 10 位(11 位)的移位寄存器中时 移位寄存器的第 0 位(起始位)清 0 而第 9 或第 10 位置 1(停止位)。如果选择 9 位数据(UART 控制寄存器——UCR 的 CHR9 位置位)，UCR 的 TXB8 位被传送到移位寄存器的第 9 位。

在波特率时钟加载到移位寄存器的传送操作时，起始位从 TXD 引脚移出，然后是数据，最低位在先。当停止位被移出时，如果在传送中有新数据被写入 UDR 中，则被装入移位寄存器中。在装入过程中，UDRE 被设置。如果在停止位被移出时 UDR 寄存器中没有新的数据，UDRE 标志位将保持为 1 直到 UDR 被重写。当没有新的数据被写入时，而且停止位在 TXD 上保持了一位的长度，USR 的 TX 完成的标志位——TXC 被置位。

当 UCR 中的 TXEN 位被设置为 1 时允许 UART 传送，通过清除该位，PD1 引脚可以被用于通用的 I/O。当 TXEN 被设置时，UART 将被连到 PD1 引脚，而不管 DDRB 中的 DDD1 位的是否设置。

二、数据接收

图 2.32 为 UART 的接收器的示意图。

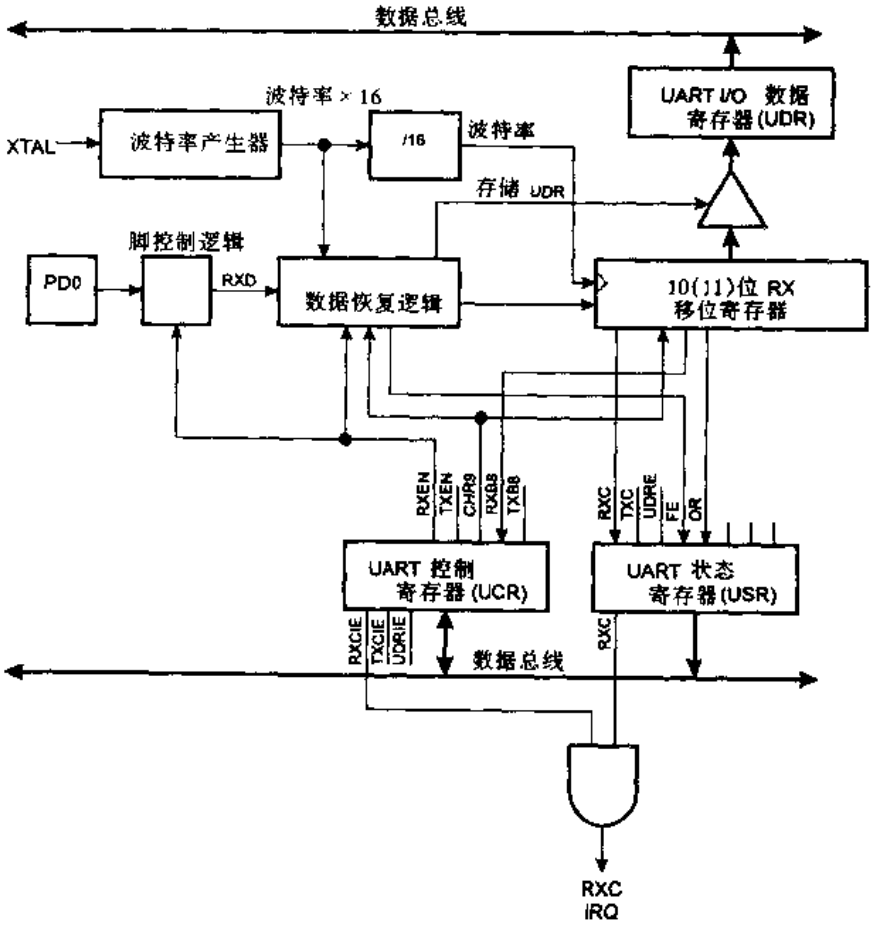


图 2.32 UART 的接收器

接收器前端的逻辑以 16 倍波特率采样 RXD 引脚的信号。当线路闲置时，一个逻辑 0 的采样将被转换为起始位的下降沿，并且起始位的探测序列被初始化。让采样 1 指示出第一个 0 采样，跟随 1 到 0 的转换之后，接收器在第 8、9 和 10 个采样点采样 RXD 引脚。如果三个采样中两个或两个以上是逻辑 1，则起始位是噪声尖峰而被拒绝，接收器继续探测下一个 1 到 0 的转换。

如果一个有效的起始位被发现，就开始起始位之后的数据位的采样，这些位也在第 8、9 和 10 处采样，3 取 2 作为该位的逻辑值，在采样的同时这些位被移入传送移位寄存器。采样输入的字符如图 2.33 所示。

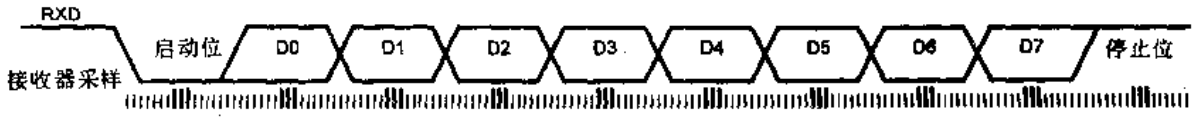


图 2.33 接收器数据采样

当停止位到来时，三个采样中的大数应为 1 才能接受该停止位。如果两个或更多为逻辑 0，UART 状态寄存器(USR)的帧错误(FE)标志被设置 1，在读 UDR 寄存器之前，用户应检查 FE 帧错误标志。一旦或无效的停止位在字符接收周期的结束时被收到，数据被传送到 UDR

寄存器而 USR 的 RXC 标志位被设置。UDR 实际上是两个物理分离的寄存器，一个发送数据，一个接收数据。当读 UDR 时，接收数据寄存器被访问；当写 UDR 寄存器时，传送数据寄存器被访问。如果选择了 9 位数据(UART 控制寄存器 UCR 中的 CHR9 位被设置)，在数据被传送到 UDR 时，传送移位寄存器的第 9 位被装入到 UCR 的 RXB8 位。

如果在收到一个字符的最后一个接收后 UDR 寄存器还没有被读走，UCR 中的超越误差标志(OR)被设置，这意味着移入移位寄存器的最后的数据字节不能被送到 UDR 中而丢失。OR 位被缓冲，当 UDR 中的有效的数据字节被读出时该位被更新，因此，用户在读 UDR 后应检查 OR 位来探测任何的超越错误。

通过清除 UCR 寄存器中的 RXEN 位使接收器被禁止，这意味着 PD0 可以被用做通用的 I/O 引脚，当 RXEN 被设置，USRT 接收器被连到 PD0 引脚而不管 DDRB 中的 DDD0 位是否设置。

三、UART 控制

1. UART I/O 数据寄存器——UDR

位	7	6	5	4	3	2	1	0	
$0C($2C)	MSB							LSB	UDR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

UDR 寄存器是两个物理分离的寄存器分享相同的 I/O 地址。当写入寄存器时，UART 的发送数据寄存器被写入；当读 UDR 时，读的是 UART 接收寄存器。

2. UART 状态寄存器——USR

位	7	6	5	4	3	2	1	0	
$0B($2B)	RXC	TXC	UDR	FE	OR	-	-	-	USR
读/写	R	R	R	R	R	R	R	R	
初始化值	0	0	0	0	0	0	0	0	

USR 寄存器是一个只读的寄存器，提供 UART 的状态信息。

位 7—RXC：UART 接收完成

当收到的字符从接收移位寄存器传到 UDR 中时该位被设置。不论探测到任何的帧错误，该位都被设置。当 UCR 中的 RXCIE 位被设置后，UART 接收完成中断将被执行(当 RXC 被设置)，RXC 在读 UDR 时被清除。当使用中断数据接收时，接受完成中断子程序必须读 UDR 而清除 RXC，否则在子程序完成时会引起新的中断。

位 6—TXC：UART 接收完成

当发送移位寄存器的全部数据被移出后且没有新的数据被写入 UDR 时该位被设置。这个标志位在半双工的通讯接口中很有用。当完成发送后立即释放通讯总线，并必须进入接收模式。当 UCR 中的 TXCIE 位被设置后，设置 TXC 将导致 UART 发送完成中断被执行，TXC 在执行相应的中断向量时被硬件清除。可选择的，TXC 位也可以通过在该位写一个逻辑 1 而被清除。

位 5—UDRE：UART 数据寄存器空

当写入 UDR 的字符被传送到发送移位寄存器中时该位被设置，设置该位指示出发送器

准备新的数据发送。当 UCR 中的 UDRIE 位被设置时，UART 发送完成中断将被执行。只要 UDRE 被设置，UDRE 可以通过写 UDR 而清除。当使用中断驱动的数据发送时，UART 数据寄存器空的中断服务程序应该写 UDR 来清除 UDRE，否则在中断子程序完成时将发生新的中断。在复位时，UDRE 被设置为 1 指示出准备传送。

位 4—FE：帧出错

如果在帧出错条件被检测到时该位被设置(如当收到数据的停止位为 0 时)。FE 在收到数据的停止位为 1 时被清除。

位 3—OR：超越出错

如果超越出错条件被检测到，该位被设置(如当 UDR 寄存器中的数据没有在新的数据被移入接收移位寄存器之前被读走)。OR 位被缓冲，这意味着一旦 UDRE 中有效的数据被读走后它将被设置。OR 位在收到的数据被传送到 UDR 中时被清除。

位 2～0—Res：保留位

在 90S8515 中这些位被保留，读出为 0。

3. UART 控制寄存器——UCR

位	7	6	5	4	3	2	1	0	
$0A($2A)	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8	UCR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R	W	
初始化值	0	0	0	0	0	0	0	0	

位 7—RXCIE：RX 完成中断使能

当该位被置 1 时，如果全局中断被使能，在 USR 中设置 RXC 位将导致接收完成中断被执行。

位 6—TXCIE：TX 完成中断使能

当该位被置 1 时，如果全局中断被使能，在 USR 中设置 TXC 位将导致发送完成中断被执行。

位 5—UDRIE：UART 数据寄存器空中断使能

当该位被置 1 时，如果全局中断被使能，在 USR 中设置 UDRE 位将导致 UART 数据寄存器空中断被执行。

位 4—RXEN：接收使能

当该位被设置时允许 UART 接收。当接收器被禁止时，TXC、OR 和 FE 位状态标志位不能设置。如果这些位被设置，在把 RXEN 关闭时不能清除它们。

位 3—TXEN：发送使能

当该位被设置为 1 时允许 UART 发送。如在发送数据时禁止发送器，则在移位寄存器的数据和后续 UDR 中的数据被全部发送完成之前发送器不会被禁止。

位 2—CHR9：9 位字符

当设置该位时，发送和接收的数据是 9 位加上起始和停止位。第 9 位通过 UCR 中的 RXB8 和 TXB8 位来分别读和写。第 9 位可以作为额外的停止位和奇偶位。

位 1—RXB8：收到的数据第 8 位

当 CHR9 被设置时，RXB8 是收到数据的第 9 数据位。

位 0—TXB8：发送的数据第 8 位

当 CHR9 被设置时，TXB8 是发送数据的第 9 数据位。

4. 波特率发生器

波特率发生器是依据以下等式的分频器来产生波特率：

$$BAUD = FCK/[16(UBRR+1)]$$

其中 BAUD 表示波特率；FCK 表示晶振频率；UBRR 表示 UART 波特率寄存器的值，UBRR(0～255)。

对于标准的晶振频率，可以通过表 2.16 来设置 UBRR 而产生常用的波特率，生成与实际波特率相差小于 2% 的 UBRR 的值。

表 2.16 不同晶振频率的 UBRR 设置

波特率	1 MHz	误差%	1.843 MHz	误差%	2 MHz	误差%	2.457 6 MHz	误差%
2400	UBRR= 25	0.2	UBRR= 47	0.0	UBRR= 51	0.2	UBRR= 63	0.0
4 800	UBRR= 12	0.2	UBRR= 23	0.0	UBRR= 25	0.2	UBRR= 31	0.0
9 600	UBRR= 65	7.5	UBRR= 11	0.0	UBRR= 12	0.2	UBRR= 15	0.0
14 400	UBRR= 3	7.8	UBRR= 7	0.0	UBRR= 8	3.7	UBRR= 10	3.1
19 200	UBRR= 2	7.8	UBRR= 5	0.0	UBRR= 6	7.5	UBRR= 7	0.0
28 800	UBRR= 1	7.8	UBRR= 3	0.0	UBRR= 3	7.8	UBRR= 4	6.3
57 600	UBRR= 0	7.8	UBRR= 1	0.0	UBRR= 1	7.8	UBRR= 2	12.5
115 200	UBRR= 0	84.3	UBRR= 0	0.0	UBRR= 0	7.8	UBRR= 0	25.0
波特率	3.276 8 MHz	误差%	3.686 4 MHz	误差%	4 MHz	误差%	4.608 MHz	误差%
2 400	UBRR= 84	0.4	UBRR= 95	0.0	UBRR= 103	0.2	UBRR= 119	0.0
4 800	UBRR= 42	0.8	UBRR= 47	0.0	UBRR= 51	0.2	UBRR= 59	0.0
9 600	UBRR= 20	1.6	UBRR= 23	0.0	UBRR= 25	0.2	UBRR= 29	0.0
14 400	UBRR= 13	1.6	UBRR= 15	0.0	UBRR= 16	2.1	UBRR= 19	0.0
19 200	UBRR= 10	3.1	UBRR= 11	0.0	UBRR= 12	0.2	UBRR= 14	0.0
28 800	UBRR= 6	1.6	UBRR= 7	0.0	UBRR= 8	3.7	UBRR= 9	0.0
57 600	UBRR= 3	12.5	UBRR= 3	0.0	UBRR= 3	7.8	UBRR= 4	0.0
115 200	UBRR= 1	12.5	UBRR= 1	0.0	UBRR= 1	7.8	UBRR= 2	20.0
波特率	7.372 8 MHz	误差%	8 MHz	误差%	9.216 MHz	误差%	11.059 MHz	误差%
2 400	UBRR= 191	0.0	UBRR= 207	0.0	UBRR= 239	0.0	UBRR= 287	0.0
4 800	UBRR= 95	0.0	UBRR= 103	0.0	UBRR= 119	0.0	UBRR= 143	0.0
9 600	UBRR= 47	0.0	UBRR= 51	0.0	UBRR= 59	0.0	UBRR= 71	0.0
14 400	UBRR= 31	0.0	UBRR= 34	0.0	UBRR= 29	0.0	UBRR= 47	0.0
19 200	UBRR= 23	0.0	UBRR= 25	0.0	UBRR= 19	0.0	UBRR= 35	0.0
28 800	UBRR= 15	0.0	UBRR= 16	0.0	UBRR= 14	0.0	UBRR= 23	0.0
57 600	UBRR= 7	0.0	UBRR= 8	0.0	UBRR= 9	0.0	UBRR= 11	0.0
115 200	UBRR= 3	0.0	UBRR= 3	0.0	UBRR= 4	0.0	UBRR= 5	0.0
波特率	14.746 MHz	误差%	16 MHz	误差%	18.432 MHz	误差%	20 MHz	误差%
2 400	UBRR= 383	-	UBRR= 416	-	UBRR= 479	-	UBRR= 520	-
4 800	UBRR= 191	0.0	UBRR= 207	0.2	UBRR= 239	0.0	UBRR= 257	-
9 600	UBRR= 95	0.0	UBRR= 103	0.2	UBRR= 119	0.0	UBRR= 129	0.2
14 400	UBRR= 63	0.0	UBRR= 68	0.6	UBRR= 79	0.0	UBRR= 86	0.2
19 200	UBRR= 47	0.0	UBRR= 51	0.2	UBRR= 59	0.0	UBRR= 64	0.2
28 800	UBRR= 31	0.0	UBRR= 34	0.8	UBRR= 39	0.0	UBRR= 42	0.9
57 600	UBRR= 15	0.0	UBRR= 16	2.1	UBRR= 19	0.0	UBRR= 21	1.4

5. 波特率寄存器——UBRR

位	7	6	5	4	3	2	1	0	
$09($29)	MSB							LSB	UBRR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

UBRR 是 8 位可以读/写的寄存器,用来确定波特率。

2.10 AVR 单片机模拟比较器

2.10.1 模拟比较器

模拟比较器对正极 PB2 引脚(AIN0)和负极 PB3 引脚(AIN1)之上的输入值进行比较。当 PB2 上的电压高于 PB3 的电压时,模拟比较器输出 ACO 被设为 1。比较器的输出可以被设置为触发定时/计数器 1 的输入捕获功能,此外,比较器的输出可以被设置为触发一个独立于模拟比较器的中断。用户可以选择比较器输出上升、下降,或触发的中断触发。比较器的方框图和周围电路如图 2.34 所示。

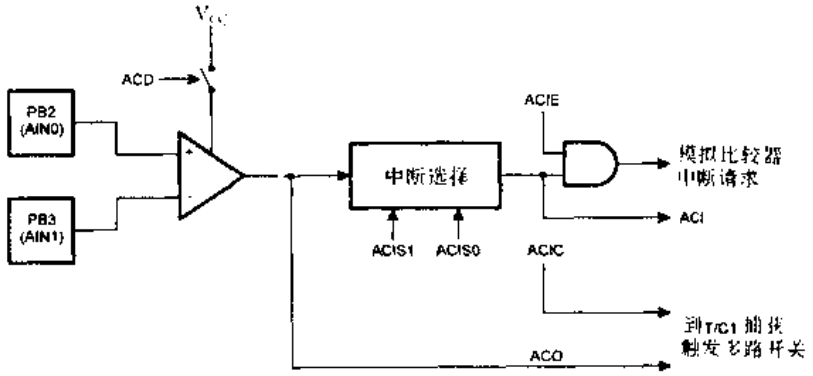


图 2.34 模拟比较器方框图

2.10.2 模拟比较器控制和状态寄存器 ACSR

位	7	6	5	4	3	2	1	0	
$08($28)	ACD	-	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
读/写	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

位 7—ACD: 模拟比较器禁止

当该位设为 1 时,模拟比较器的电源关闭。该位可以在任何时候被设置,以便关闭模拟比较器。在闲置模式的电源消耗为临界时,它最为常用,且无需从模拟比较器唤醒。改变 ACD 位时,模拟比较器中断必须通过清空 ACSR 中的 ACIE 位来禁止;否则,在该位改变时,会产生中断。

位 6—Res：保留位

90 系列单片机的该位为保留位，总读 0。

位 5—ACO：模拟比较器输出

ACO 直接与模拟比较器的输出相连。

位 4—ACI：模拟比较器中断标志位

当比较器输出事件触发由 ACI1 和 ACI0 定义的中断模式时，这一位被设为 1。若 ACIE 位被设为 1，且 SREG 中的 I 位被设为 1 时，执行模拟比较器的中断程序。当执行相应的中断处理向量时，ACI 被硬件清空。作为替换，ACI 通过对标志位写入逻辑 1 来清空。

位 3—ACIE：模拟比较器中断使能

当 ACIE 位设为 1，且状态寄存器中的 I 位被设为 1 时，模拟比较器中断被激活。当被清为 0 时，中断被禁止。

位 2—ACIC：模拟比较器输入捕获使能

当设置为 1 时，该位使能定时计数器 1 的输入捕获功能，由模拟比较器来触发。在这种情况下，模拟比较器的输出直接连到输入捕获前端逻辑，使比较器能利用定时器/计数器 1 输入捕获中断的噪声消除和边缘选择的特性。当该位被清除时，模拟比较器和输入捕获功能之间没有联系。为了使比较器触发定时器/计数器 1 的输入捕获中断，定时器中断屏蔽寄存器(TIMSK)的 TICIE1 位必须被设置。

位 1，0—ACIS1，ACIS0：模拟比较器中断模式选择

这些位决定了哪一比较器事件触发模拟比较器中断。表 2.17 所示为不同的设置。

表 2.17 ACIS1/ACIS0 设置

ACIS1	ACIS0	中断模式	ACIS1	ACIS0	中断模式
0	0	输出触发，比较器中断	1	0	下降输出沿，比较器中断
0	1	与上相反	1	1	上升输出沿，比较器中断

注意：当改变 ACIS1/ACIS0 位时，必须通过清除 ACSR 寄存器中的中断使能位来禁止模拟比较器中断。否则，当这些位改变时会发生中断。

2.11 AVR 单片机 I/O 端口

2.11.1 端口 A

一、A 口特性

A 口为一个 8 位的双向 I/O 口。

A 口分配有 3 个数据存储地址，分别为数据寄存器 PORTA $1B($3B)、数据方向寄存器 DDRA $1A($3A)和 A 口的输出引脚 PINA $19($39)。A 口的输入引脚地址为只读，而数据寄存器和数据方向寄存器为可读写。

所有的 A 口引脚都有独立可选的上拉，A 口输出缓冲器可以吸收 20 mA 的电流以直接驱动 LED 显示。当 PA0 到 PA7 引脚被用作输入且被外部拉低时，若内部拉高被激活，这些引脚将成为电流源(I_{IL})。

A 口引脚具有与可选的外部数据 SRAM 有关的第二功能，A 口在访问外部数据存储器时可以配置为复用的低位地址/数据线，在该模式下，A 口有内部的上拉。

当通过 MCU 控制寄存器 MCUCR 的 SRE，外部 SRAM 使能位把 A 口设置为第二功能，更改的设置会覆盖数据方向存储器。

1. A 口数据寄存器——PORTA

位	7	6	5	4	3	2	1	0	
$1B($3B)	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

2. A 口数据方向寄存器——DDRA

位	7	6	5	4	3	2	1	0	
$1A($3A)	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

3. A 口输入脚地址——PINA

位	7	6	5	4	3	2	1	0	
$19($39)	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
读/写	R	R	R	R	R	R	R	R	
初始化值	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	

A 口的输入引脚地址 PINA 不是一个寄存器，该地址允许对 A 口的每一个引脚的物理值进行访问。当读 PORTA 时，读到的是 PORTA 的数据锁存器；当读 PINA 时，引脚上的逻辑值被读取。

二、A 口作为通用的数字 I/O

当作为数字 I/O 口时 A 口所有的 8 位都等效。

PAn 为通用 I/O 引脚：DDRA 寄存器的 DDAn 位选择引脚的方向。如果 DDAn 设为 1，PAn 被配置为输出引脚；如果 DDAn 设为 0，PAn 被配置为输入引脚；如果 PORTAn 被设置为 1，DDAn 被配置为输入引脚，则 MOS 上拉电阻被激活。为了关断上拉电阻，PORTAn 位必须被清除或者引脚被配置为输出引脚。A 口引脚 DDAn 的作用见表 2.18。

表 2.18 A 口引脚的 DDAn 的作用

DDAn	PORTAn	I/O	上拉	注 释
0	0	输入	否	三态(高阻)
0	1	输入	是	上拉低 PAn 脚输出电流
1	0	输出	否	推挽 0 输出
1	1	输出	否	推挽 1 输出

n:7,6,5,…,0 为引脚数。

三、A 口原理图

A 口原理如图 2.35 所示(注意：所有引脚是同步的，同步锁存器在图中并未列出)。

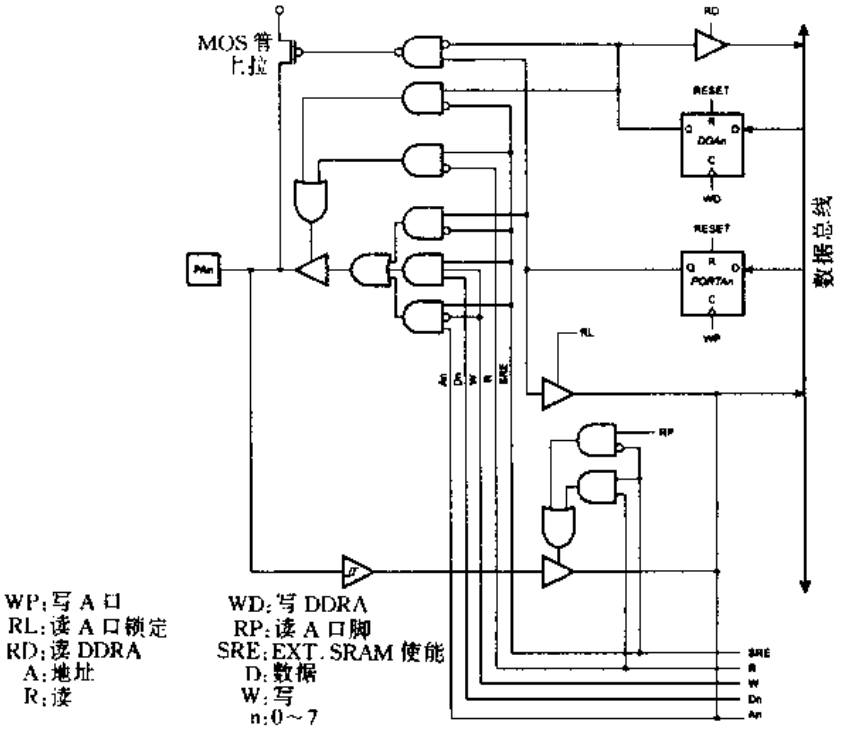


图 2.35 A 口原理图(脚 PA0～PA7)

2.11.2 端口 B

一、B 口 特性

B 口为一个 8 位的双向 I/O 口。

B 口分配有 3 个数据存储地址,分别为数据寄存器 PORTB $18($38)、数据方向寄存器 DDRB $17($37)和 B 口的输出引脚 PINB $16($36)。B 口的输入引脚地址为只读,而数据寄存器和数据方向寄存器为可读写。

所有的 B 口引脚均有单独的可选择拉高。B 口输出缓冲器可以吸收 20 mA 的电流以直接驱动 LED 显示。当 PB0 到 PB7 引脚被用作输入。且被外部拉低时,若内部拉高被激活,这些引脚将成为电流源(I_{IL})。

具有第二功能的 B 口引脚如表 2.19 所示。当引脚被用作第二功能时,DDRB 和 PORTB 寄存器必须根据第二功能说明来设置。

表 2.19 B 口引脚第二功能

口引脚	第 二 功 能	口引脚	第 二 功 能
PB0	T0(定时器/计数器 0 外部计数器输入)	PB4	$\overline{SS}$(SPI 从选择输入)
PB1	T1(定时器/计数器 1 外部计数器输入)	PB5	MOSI(SPI 总线主输出/从输入)
PB2	AIN0(模拟比较器正输入)	PB6	MOSO(SPI 总线主输出/从输入)
PB3	AIN1(模拟比较器负输入)	PB7	SCK(SPI 总线串行时钟)

1. B 口数据寄存器——PORTB

位	7	6	5	4	3	2	1	0	
$18($38)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

2. B 口数据方向寄存器——DDRB

位	7	6	5	4	3	2	1	0	
$17($37)	DDB7	DDB6	DDBB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

3. B 口输入引脚地址——PINB

位	7	6	5	4	3	2	1	0	
$16($36)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
读/写	R	R	R	R	R	R	R	R	
初始化值	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	

B 口输入引脚地址 PINB 并不是一个寄存器，这一地址允许对 B 口每个引脚的物理值进行访问。当读取 PORTB 时，PORTB 数据锁存器被读取，当读取 PINB 时，引脚上的当前逻辑值被读取。

二、B 口作为通用数字 I/O

当 B 口上的所有 8 个位用作数字 I/O 引脚时功能一样。

PBn 为通用 I/O 引脚：DDRB 寄存器中的 DDBn 位选择该引脚的方向。若 DDBn 被设为 1 时，PBn 设为输出引脚。若 DDBn 被清为 0 时，PBn 设为输入引脚。若 PORTBn 被设为 1，引脚被设为输入时，MOS 拉高电阻被激活。为关闭拉高电阻，PORTBn 必须被清为 0，或者该引脚必须被设置为输出引脚。B 口引脚的 DDBn 作用见表 2.20。

表 2.20 B 口引脚的 DDBn 作用

DDBn	PORTBn	I/O	上拉	注释
0	0	输入	否	三态(高阻)
0	1	输入	是	上拉低 PBn 脚输出电流
1	0	输出	否	推挽 0 输出
1	1	输出	否	推挽 1 输出

n：7，6，…，0 为引脚数。

三、B 口的可选择性功能：

SCK— B 口，位 7

SCK、SPI 的主时钟输出、从时钟输入。当 SPI 被使能为从机时，该引脚被作为输入而不管 DDB7 的设置；当 SPI 被使能为主机时，该引脚的数据方向由 DDB7 来控制；当该引脚被强制作为输入时，内部的上拉仍可以被 PORTB7 位来控制，详见 SPI 口的描述。

MISO— B 口，位 6

MISO、SPI 的主数据输入、从时的数据输出。当 SPI 被使能为主机时，该引脚被作为输入而不管 DDB6 的设置；当 SPI 被使能为从机时，该引脚的数据方向由 DDB6 来控；当该引脚被强制作为输入时，内部的上拉仍可以被 PORTB6 位来控制，详见 SPI 口的描述。

MOSI— B 口，位 5

MOSI、SPI 的主数据输出、从时的数据输入。当 SPI 被使能为从机时，该引脚被作为输入而不管 DDB5 的设置；当 SPI 被使能为主机时，该引脚的数据方向由 DDB5 来控制；当该引脚被强制作为输入时，内部的上拉仍可以被 PORTB5 位来控制，详见 SPI 口的描述。

$\overline{SS}$—B 口，位 4：

$\overline{SS}$为从机端口选择信号输入端。当 SPI 被使能为从机时，该引脚被作为输入而不管 DDB5 的设置。作为从机时，当该引脚被置低时 SPI 被激活。当 SPI 被使能为主机时，该引脚的数据方向由 DDB5 来控制。当该引脚被强制作为输入时，内部的上拉仍可以被 PORTB4 位来控制，详见 SPI 口的描述。

AIN1—B 口，位 3

AIN1 为模拟比较器负极输入。当被设置为输入(DDB3 被清为 0)，且内部 MOS 拉高电阻关闭(PB3 被清为 0)时，该引脚用作片内模拟比较器的负极输入。

AIN0— B 口，位 2

AIN0 为模拟比较器正极输入。当被设置为输入(DDB2 被清为 0)，且内部 MOS 拉高电阻关闭(PB2 被清为 0)时，该引脚用作片内模拟比较器的正极输入。

T1—B 口，位 1

T1 为定时器/计数器 1 的计数输入源，详见定时器部分。

T0—B 口，位 0

T0 为定时器/计数器 0 的计数输入源，详见定时器部分。

四、B 口原理图

B 口原理图如图 2.26～图 2.41 所示，注意：所有引脚为同步的。同步锁存器图中并未列出。

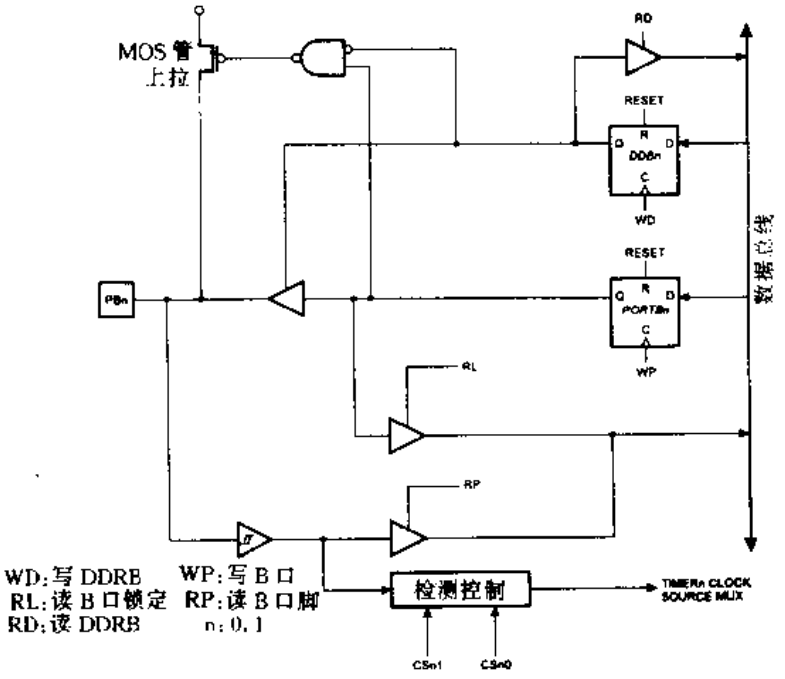


图 2.36 B 口原理图(PB0 和 PB1 脚)

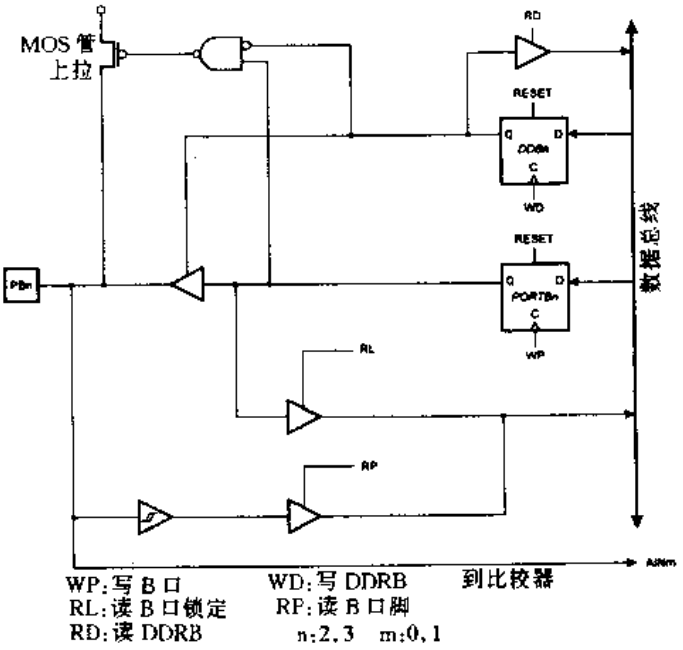


图 2.37 B 口原理图(PB2 和 PB3 脚)

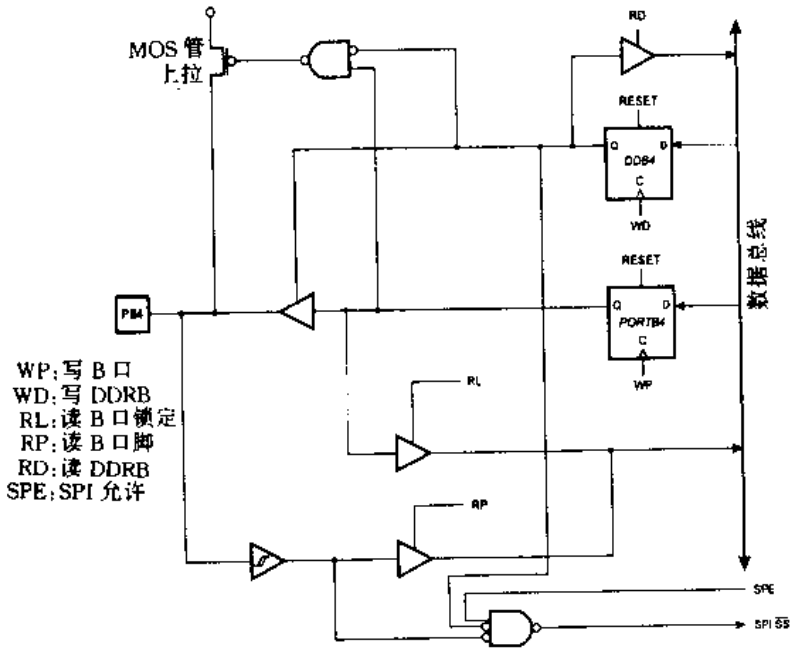


图 2.38 B 口原理图(PB4 脚)

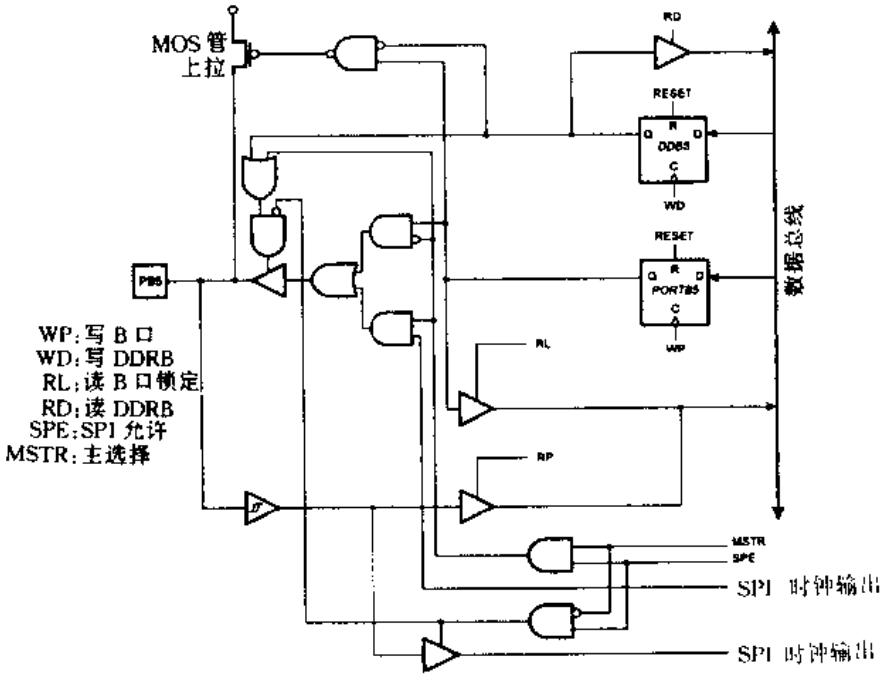


图 2.39 B 口原理图(PB5 脚)

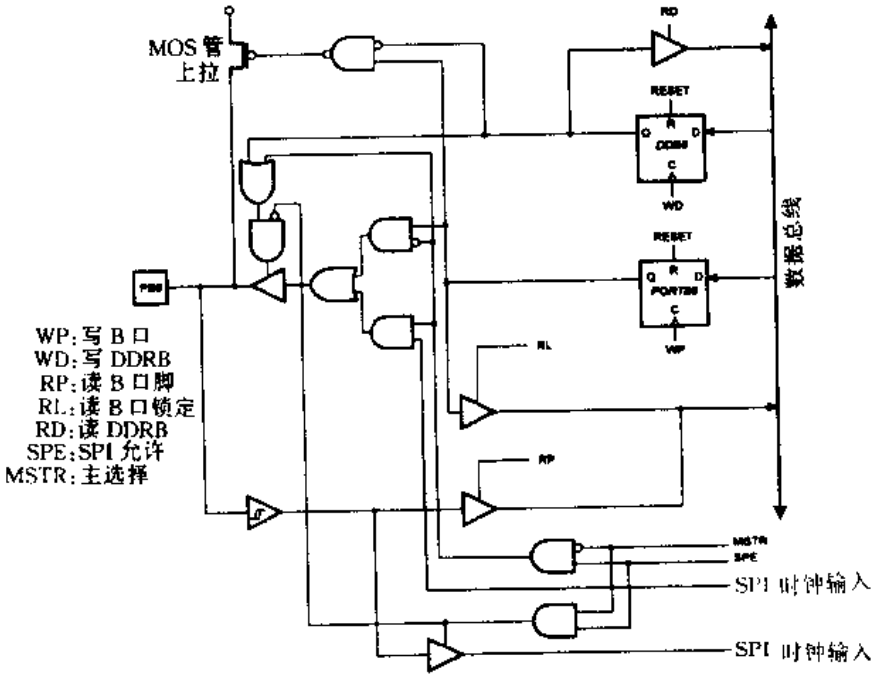


图 2.40 B 口原理图(PB6 脚)

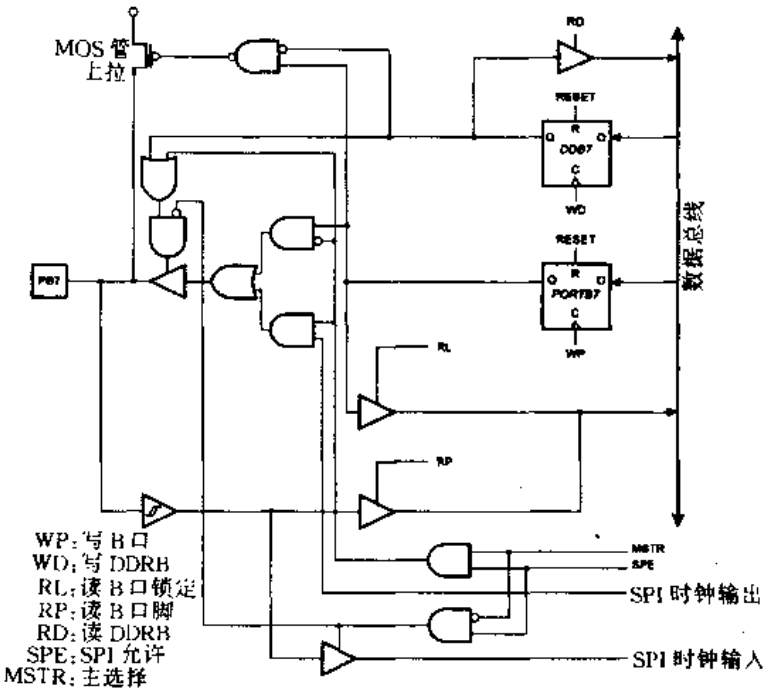


图 2.41 B 口原理图(PB7 脚)

2.11.3 端口 C

一、C 口

C 口为一个 8 位的双向 I/O 口。

C 口分配有 3 个数据存储地址,分别为数据寄存器 PORTC $ 15($ 35)、数据方向寄存器 DDRC $ 14($ 34)和 C 口的输出引脚 PINC $ 13($ 33)。C 口的输入引脚地址为只读,而数据寄存器和数据方向寄存器为可读写。

所有的 C 口引脚均有单独的可选择拉高。C 口输出缓冲器可以吸收 20 mA 的电流以直接驱动 LED 显示。当 PC0 到 PC7 引脚被用作输入且被外部拉低时,若内部拉高被激活,这些引脚将成为电流源(I_{IL})。

C 口引脚具有与可选的外部数据 SRAM 有关的第二功能,C 口在访问外部数据存储器时可以配置为复用的高位地址线,在该模式下,C 口输出 1 时使用内部的上拉。

当通过 MCU 控制寄存器 MCUCR 的 SRE,外部 SRAM 使能位把 C 口设置为第二功能,更改的设置会覆盖数据方向存储器。

1. C 口数据寄存器——PORTC

位	7	6	5	4	3	2	1	0	
$ 15($ 35)	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

2. C 口数据方向寄存器——DDRC

位	7	6	5	4	3	2	1	0	
$14($34)	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

3. C 口输入引脚地址——PINC

位	7	6	5	4	3	2	1	0	
$13($33)	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
读/写	R	R	R	R	R	R	R	R	
初始化值	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	

C 口的输入引脚地址 PINC 不是一个寄存器,该地址允许对端口 C 的每一个引脚进行存取。当读 PORTC 时,读到的是 PORTC 的数据锁存器;当读 PINC 时,引脚上的逻辑值被读取。

二、C 口作为通用的数字 I/O

当作为数字 I/O 口时 C 口所有的 8 位都等效。

PCn 为通用 I/O 引脚: DDRC 寄存器的 DDCn 位选择引脚的方向。如果 DDCn 设为 1, PCn 被配置为输出引脚;如果 DDCn 设为 0, PCn 被配置为输入引脚;如果 PORTCn 被设置为 1, DDCn 被配置为输入引脚,则 MOS 上拉电阻被激活,为了关断上拉电阻, PORTCn 位必须被清除或者引脚被配置为输出引脚。C 口引脚 DDCn 的作用见表 2.21。

表 2.21 C 口引脚的 DDCn 作用

DDCn	PORTCn	I/O	上 拉	注 释
0	0	输入	否	三态(高阻)
0	1	输入	是	上拉低 PCn 脚输出电流
1	0	输出	否	推挽 0 输出
1	1	输出	否	推挽 1 输出

n:7,…,0 为引脚数。

三、C 口原理图

C 口原理图如图 2.42 所示。注意:所有的引脚是同步的,同步锁存器图中并未列出。

2.11.4 端口 D

一、D 口特性

D 口是一个带内部上拉的 8 位双向 I/O 口。

D 口占了 3 个数据存储器的地址,一个是数据寄存器 PORTD $12($32)、数据方向寄存器 DDRD $11($31)和端口 D 输入引脚 PIND $10($30)。D 口的引脚地址是只读的,而数据寄存器和数据方向寄存器可以读写。

D 口的输出缓冲器可以吸收 20 mA 的电流。D 口的引脚在激活内部上拉时,如果外部被拉低就会成为电流源(I_{IL})。

某些 D 口的引脚是有第二功能如表 2.22 所示。

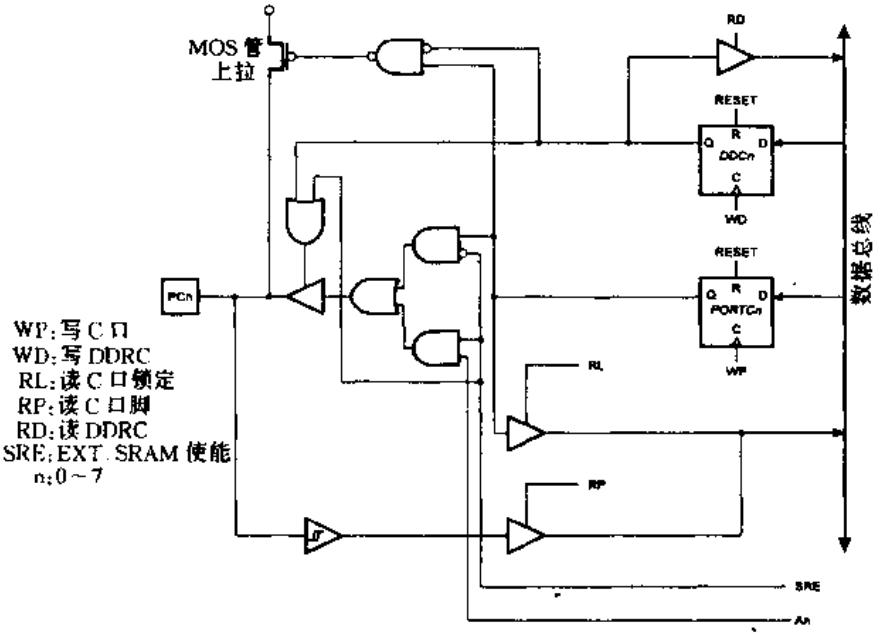


图 2.42 C 口原理图(PC0～PC7)

表 2.22 D 口引脚第二功能

口引脚	第 二 功 能	口引脚	第 二 功 能
PD0	RDX(UART 输入线)	PD5	OC1A(T/C1 输出比较 A 匹配输出)
PD1	TDX(UART 输出线)	PD6	$\overline{WR}$(写选通)
PD2	INT0(外部中断 0 输入)	PD7	$\overline{RD}$(读选通)
PD3	INT1(外部中断 1 输入		

当 D 口的引脚被用于第二功能时,DDRD 和 PORTD 寄存器必须按照第二功能来设置。

1. D 口数据寄存器——PORTD

位	7	6	5	4	3	2	1	0	
$12($32)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

2. D 口数据方向寄存器——DDRD

位	7	6	5	4	3	2	1	0	
$11($31)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

3. D 口输入引脚地址——PIND

位	7	6	5	4	3	2	1	0	
$10($30)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
读/写	R	R	R	R	R	R	R	R	
初始化值	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	

D 口的输入引脚地址 PIND 不是一个寄存器,该地址允许对端口 D 的每一个引脚进行存取。当读 PORTD 时,读到的是 PORTD 的数据锁存器;当读 PIND 时,引脚上的逻辑值被读取。

二、D 口作为通用的数字 I/O

PDn,通用 I/O 引脚: DDRD 寄存器的 DDDn 位选择引脚的方向。如果 DDDn 设为 1,PDn 被配置为输出引脚,如果 DDDn 设为 0,PDn 被配置为输入引脚;如果 PORTDn 被设置为 1,DDDn 被配置为输入引脚,则 MOS 上拉电阻被激活。为了关断上拉电阻,PORTDn 位必须被清除或者引脚被配置为输出引脚。D 口引脚 DDDn 的作用见表 2.23。

表 2.23 D 口引脚的 DDDn 作用

DDDn	PORTDn	I/O	上 拉	注 释
0	0	输入	否	三态(高阻)
0	1	输入	是	上拉低 PDn 脚输出电流
1	0	输出	否	推挽 0 输出
1	1	输出	否	推挽 1 输出

n:7,6,…,0 为引脚数。

三、D 口的第二功能

RD—PORTD, 位 7

$\overline{RD}$是外部数据存储器读选通。

WR—PORTD, 位 6

$\overline{WR}$是外部数据存储器写选通。

OC1—PORTD, 位 5

OC1 表示比较匹配的输出。PD5 可以作为定时器/计数器 1 的比较匹配的外部输出。为了实现该功能, PD5 应被配置为输出(DDD5 设置为 1)。详细说明和怎样使能该输出请见定时器/计数器 1 的描述。OC1 引脚还可以作为 PWM 模式下定时功能的输出。

INT1—PORTD, 位 3

INT1 为外部中断源 1。PD3 引脚可作为 MCU 的外部中断源,详见中断部分。

INT0—PORTD, 位 2

INT0 为外部中断 0。PD2 引脚可作为 MCU 的外部中断源,详见中断部分。

TXD—PORTD, 位 1

发送数据(UART 的数据输出引脚),当 UART 数据输出允许时,该引脚被作为输出而不管 DDRD1 的值。

RXD—PORTD, 位 0

接受数据(UART 的数据输入引脚),当 UART 数据输入允许时,该引脚被作为输出而不管 DDRD0 的值。当 UART 强制该引脚为输入时, PORTD0 的一个逻辑 1 将开启内部的上拉。

四、D 口 原理图

D 口原理图如图 2.43～2.49 所示,注意所有的端口引脚都是同步的,图中未画出同步锁存器。

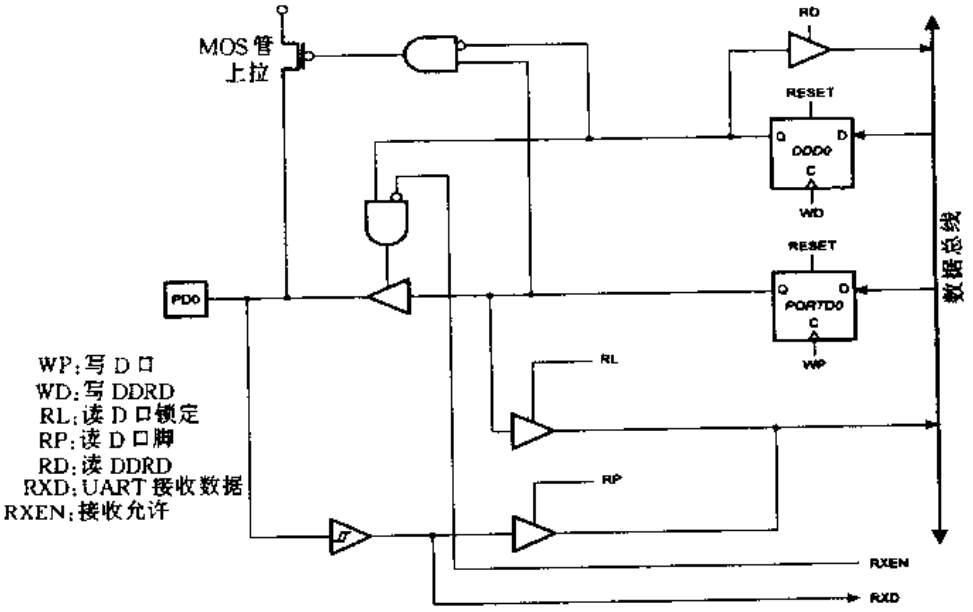


图 2.43 D 口原理图(PD0 脚)

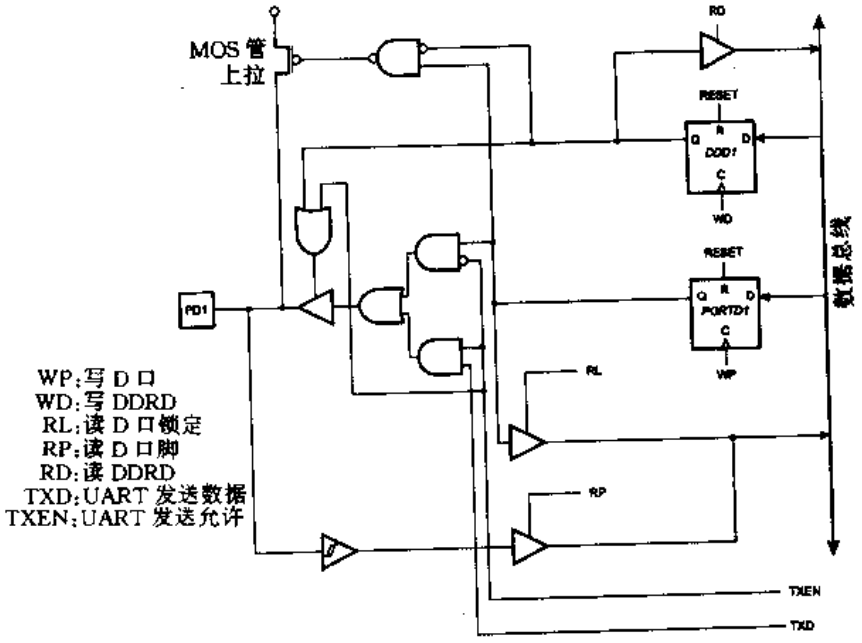


图 2.44 D 口原理图(PD1 脚)

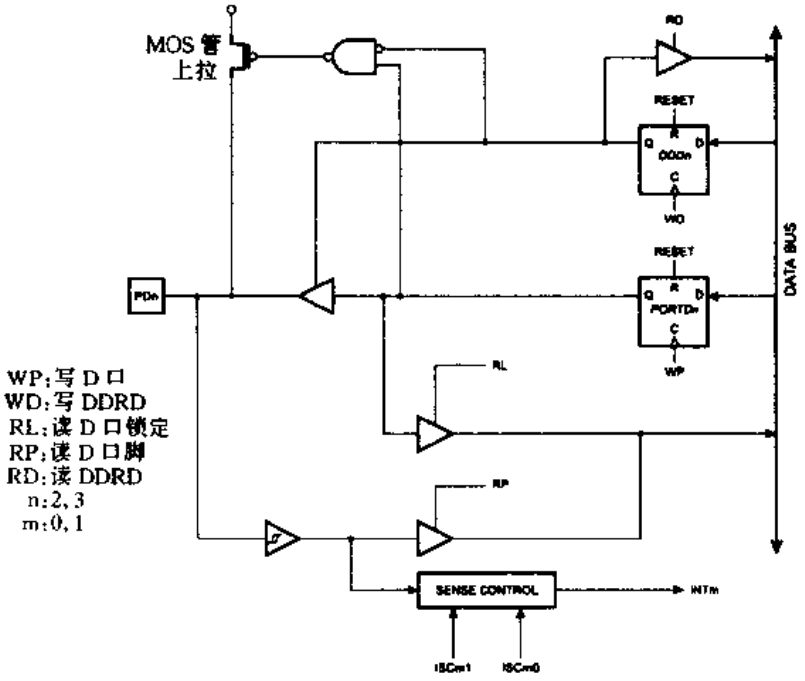


图 2.45 D 口原理图(PD2 和 PD3 脚)

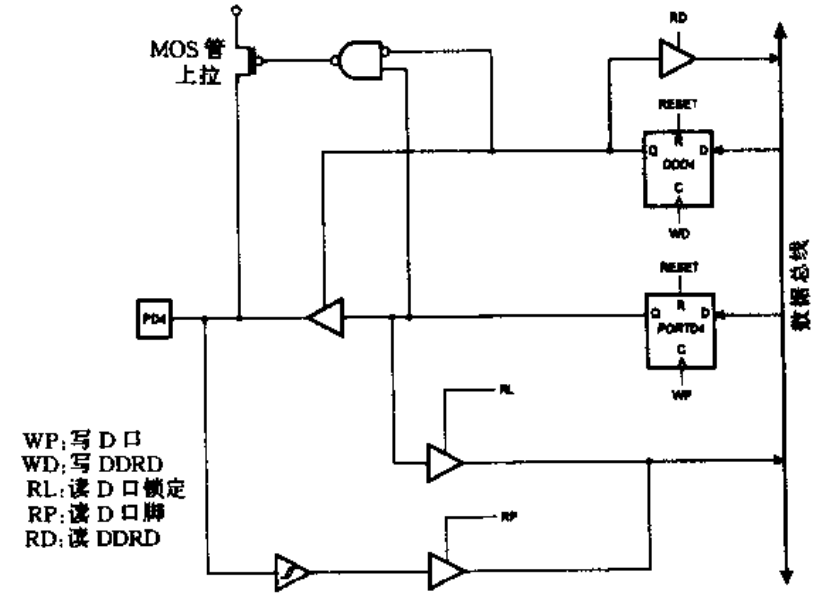


图 2.46 D 口原理图(PD4 脚)

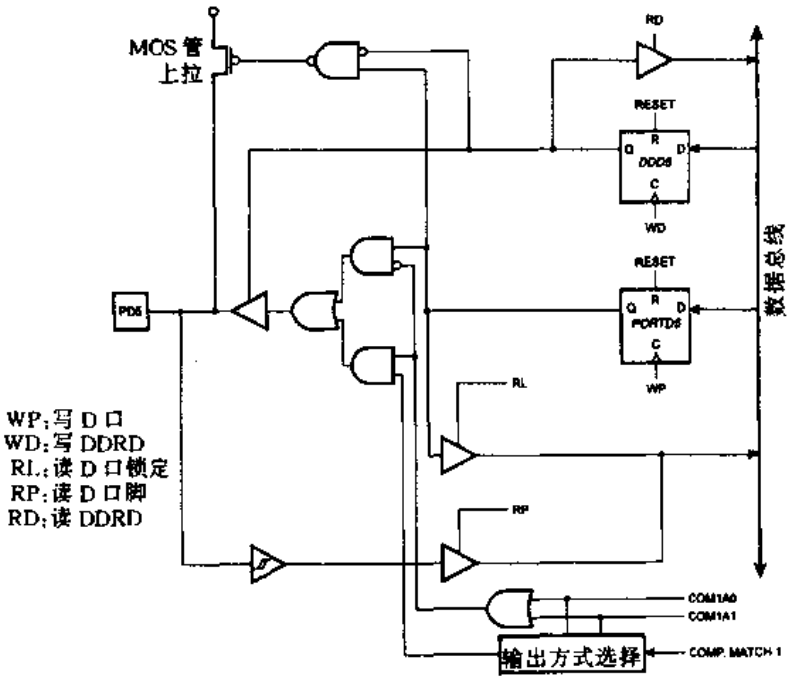


图 2.47 D 口原理图(PD5 脚)

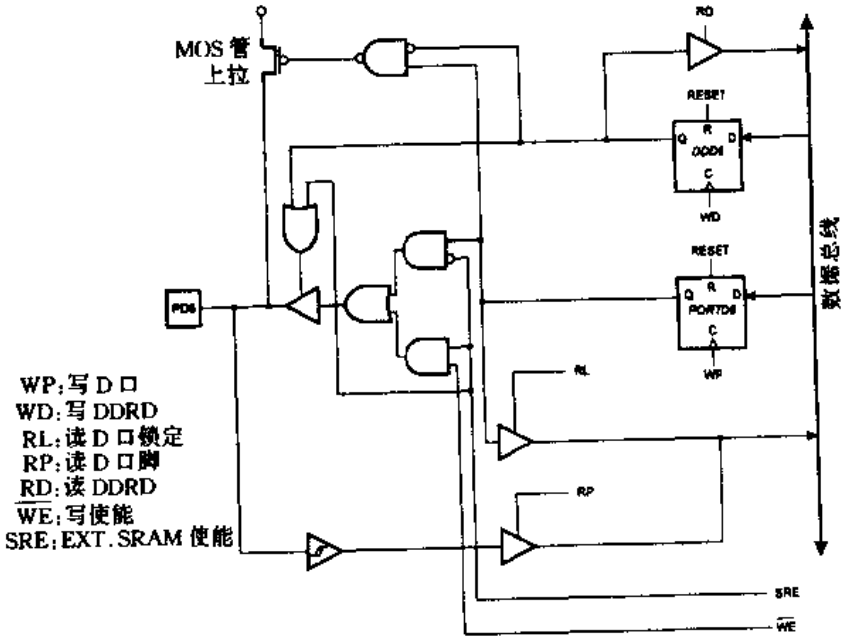


图 2.48 D 口原理图(PD6 脚)

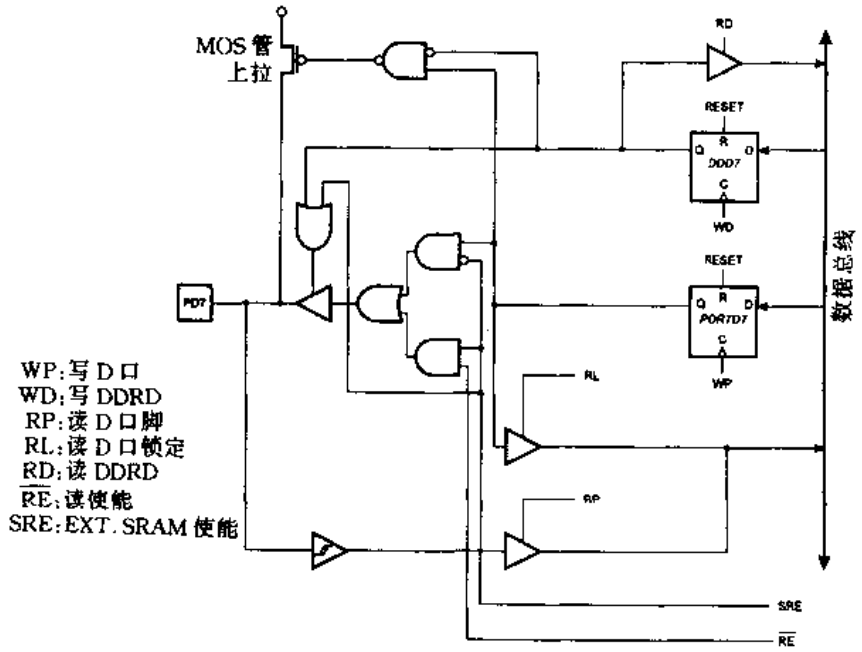


图 2.49 D口原理图(PD7 脚)

2.12 AVR 单片机存储器编程

2.12.1 编程存储器锁定位

90 系列单片机 MCU 提供 2 个加密锁定位,可以不编程(1)或编程为(0),而获得表 2.24 的额外特性。

表 2.24 锁定位保护方式

编程锁定位			保护类型
模式	LB1	LB2	
1	1	1	无编程锁定特征
2	0	1	Flash 的再编程被禁止
3	0	0	同 2 模式,但校验被禁止

注意:锁定位只能整片擦除时才能擦除。

2.12.2 熔断位

90 系列单片机有两个熔断位,SPIEN 和 FSTRT。

当 SPIEN 被编程为 0 时,串行编程下载被允许,缺省值是被编程的(0)。

当 FSTRT 被编程为 0 时,选择短的启动时间,,缺省值为擦除(1),定货时可以要求该位被编程。

这些位不能被串行编程模式访问,也不能被全片擦除所改变。

2.12.3 芯片代码

所有的ATMEL微控制器都有3字节的电子标签指定该器件的型号,该标签可以被串行模式和并行模式读出,这三个字节属于不同的地址空间,对于90系列单片机它们是:

(1) $000:$1E (指出厂商是ATMEL)。

(2) $001:$93 (指出4K字节的Flash存储器)。

(3) $002:$01 (当$001是$93时指出是AT90S8515器件)。

2.12.4 编程Flash和EEPROM

90系列单片机提供了8K字节的系统在线可编程的Flash程序存储器和512字节的EEPROM数据存储器。

90系列单片机通常被售出时,片内的Flash程序存储器和EEPROM数据存储器阵列是被擦除的状态(即内容=$FF),而可以被编程。该器件支持高压的(12 V)并行编程模式和低压的串行编程模式,+12 V电源仅用于编程使能,并不从该引脚获取意义上的电流,串行模式提供了对90系列单片机方便的在线程序和数据下载方式。

在两种模式下90系列单片机的程序和数据存储器阵列是按字节编程的。对于EEPROM,在串行编程模式中提供了一个自动擦除周期。

2.12.5 并行编程

该节描述了怎样并行编程和验证90系列单片机的Flash程序存储器、EEPROM数据存储器及程序存储器的锁定位和熔断位。AT908515的并行编程如图2.50所示。

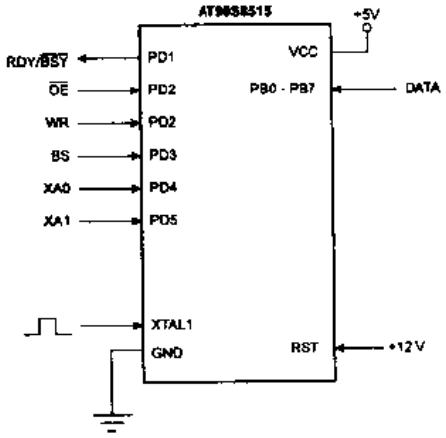


图2.50 并行编程

一、信号名称

本节中90系列单片机的一些引脚被定义为描述并行编程的信号名称,表2.25中没有描述的引脚仍用引脚名来表示。

XA1/XA0位决定了当XTAL1引脚给出一个正脉冲时的动作,这两位的设置见表2.26。

当脉冲是$\overline{WR}$或$\overline{OE}$时,装入的命令决定了输入或输出的行为,该命令是一个字节,每一位的功能见表2.27。

表 2.25 引脚名映射

编程方式的信号名	引脚名	I/O	功 能
RDY/$\overline{BSY}$	PD1	O	0:器件正在编程,1:器件就绪等待新的命令
$\overline{OE}$	PD2	I	输出允许(低激活)
$\overline{WR}$	PD3	I	写脉冲(低激活)
BS	PD4	I	字节选择
XA0	PD5	I	XTAL 用于位 0
XA1	PD6	I	XTAL 用于位 1

表 2.26 XA1 和 XA0 编码

XA1	XA0	当 FLASH 被 PULSED 时操作
0	0	装入 Flash 或 EEPROM 地址(由 BS 确定的 Fsash 高或低地址字节)
0	1	装入数据(由 BS 线确定的 Fsash 高或低数据字节)
1	0	装入命令
1	1	闲置

表 2.27 命令字节位编码

位号	当 设 置 时 的 定 义
7	芯片擦除
6	写熔断位,在数据字节中定位下列位位置:D5:SPIEN 熔断丝,D0:FSTRT 熔断丝(注意:写 "0"为编程,写 "1"为擦除)
5	写锁定位,在数据字节中定位下列位位置:D7:LB1,D6:LB2,D5:SPIEN 熔断,D0:FSTRT 熔断(注意: "0"为编程)
4	写 Flash 或 EEPROM 中读(由位 0 测定)
3	读标记行
2	读锁定位和熔断位,在数据字节中定位下列位位置:D1:LB1,D0:LB2(注意:写"0"为编程)
1	从 Flash 或 EEPROM 中读(由位 0 测定)
0	0:Flash 访问,1: EEPROM 访问

二、进入编程模式

下列算法使器件进入并行编程模式:

(1) 在 V_{CC}和 GND 之间加上 4.5~5.5 V 电压。

(2) 把$\overline{RESET}$和 BS 设置为 0,并等待至少 100 ns。

(3) 把$\overline{RESET}$加到 12 V 且在改变 BS 之前至少等待 100 ns。

三、全片擦除

全片的擦除功能将擦除 Flash 和 EEPROM 存储器和锁定位。锁定位在程序存储器被完全擦除之前不会被清除,熔断位并不改变。在编程之前必须执行一个全片擦除。

装载"全片擦除"命令:

(1) 把 XA1 和 XA0 设置为(10),使能命令装入。

(2) 把 BS 设置为 0。

(3) 设置 PB(7~0)为"1000 0000",这是全片擦除命令。

(4) 给 XTAL1 一个正脉冲,这将装入命令并且开始擦除 Flash 和 EEPROM 阵列。在 XTAL1 脉冲之后,给$\overline{WR}$一个负脉冲使得锁定位在擦除周期结束时被擦除。然后等待 10 ms 使擦除完成。全片擦除不生成 RDY/$\overline{BSY}$信号。

四、编程 Flash

装载"编程 Flash"命令:

(1) 设置 XA1、XA0 为"10",使能命令装入。

(2) 把 BS 设为 0。

(3) 把 PB(7~0)设为"0001 0000",这是 Flash 编程命令。

(4) 给 XTAL1 一个正脉冲,这将装入该命令。

装入地址低字节

(1) 设置 XA1、XA0 为"00",使能地址装入。

(2) 设置 BS 为 0,选择低位地址。

(3) 设置 PB(7~0)=低位地址字节($00~ $FF)。

(4) 给 XTAL1 一个正脉冲,这将装入地址低字节。

装入地址高字节

(1) 设置 XA1、XA0 为"00",使能地址装入。

(2) 设置 BS 为 1,选择高位地址。

(3) 设置 PB(7~0)=高位地址字节($00~ $0F)。

(4) 给 XTAL1 一个正脉冲,这将装入地址高字节。

装入数据字节

(1)设置 XA1、XA0 为"01",使能数据装入。

(2) 设置 PB(7~0)=数据低字节($00~ $FF)。

(3) 给 XTAL1 一个正脉冲,这将装入数据低字节。

写入数据低字节

(1) 设置 BS=0,选择数据低字节。

(2) 给$\overline{WR}$一个负脉冲,开始编程数据低字节,RDY/$\overline{BSY}$为低电平。

(3) 等待直到 RDY/$\overline{BSY}$变高时再编程另一个字节。

装入数据字节

(1) 设置 XA1、XA0 为"01",使能数据装入。

(2)设置 PB(7~0)=数据高字节($00~ $FF)。

(3) 给 XTAL1 一个正脉冲,这将装入数据高字节。

写入数据高字节

(1) 设置 BS=1,选择数据高字节。

(2) 给$\overline{WR}$一个负脉冲,开始编程数据低字节,RDY/$\overline{BSY}$为低电平。

(3) 等待直到 RDY/$\overline{BSY}$变高时再编程另一个字节。

装入的命令和地址在编程过程中保持不变,为了简化编程,请考虑如下:

· 编程 Flash 存储器的命令仅在编程第一个字节时需要被装入。

· 地址高字节仅在编程 Flash 中一个新的 256 字节页之前需要装入。

图 2.51 和图 2.52 为可编程 Flash 低字节和可编程 Fsash 高字节。

五、编程 EEPROM

编程 EEPROM 的算法如下(参考 Flash 编程部分的命令、地址和数据的装载):

(1) 装入命令"0001 0001"。

(2) 装入低位 EEPROM 地址($00~ $FF)。

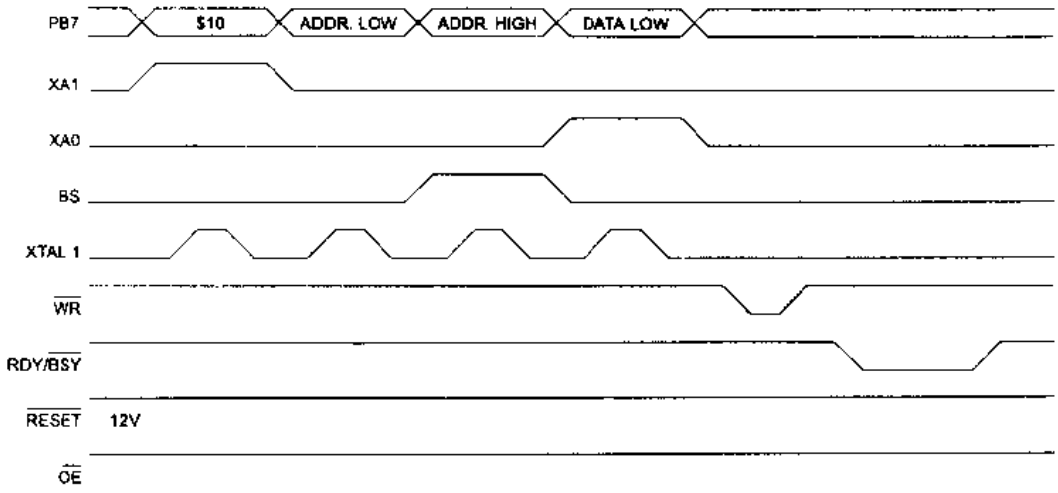


图 2.51 可编程 Flash 低字节

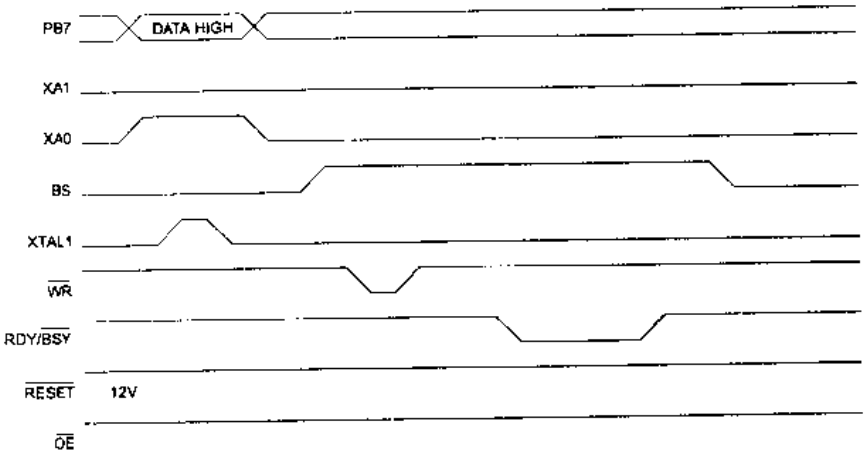


图 2.52 可编程 Flash 高字节

(3) 装入高位 EEPROM 地址($00～$01)。

(4) 装入低位 EEPROM 数据($00～$FF)。

(5) 给$\overline{WR}$一个负脉冲并等待 RDY/$\overline{BSY}$变为高电平。

仅在编程第一个字节之前需要装入命令。

六、读 Flash

读 Flash 的算法如下(参考 Flash 编程部分的命令、地址和数据的装载):

(1) 装入命令"0000 0010"。

(2) 装入低地址($00～$FF)。

(3) 装入高地址($00～$0F)。

(4) 设置$\overline{OE}$为 0,BS 为 0,这时数据低字节可从 PB(7～0)读出。

(5) 设置 BS 为 1,这时数据高字节可从 PB(7～0)读出。

(6) 设置$\overline{OE}$为 1。

仅在读第一个字节之前需要装入命令。

七、读 EEPROM

读 EEPROM 的算法如下(参考 Flash 编程部分的命令、地址和数据的装载):

(1) 装入命令"0000 0011"。

(2) 装入低地址($00~ $FF)。
(3) 装入高地址($00~ $01)。
(4) 设置$\overline{OE}$为0,BS为0,这时数据低字节可从PB(7~0)读出。
(5) 设置$\overline{OE}$为1。
仅在读第一个字节之前需要装入命令。

八、编程熔断位

编程熔断位的算法如下(参考Flash编程部分的命令、地址和数据的装载):
(1) 装入命令"0100 0000"。
(2) 装入数据:
位5=0,编程SPIEN熔断位;位5=1,擦除SPIEN熔断位。
位0=0,编程FSTRT熔断位;位5=1,擦除FSTRT熔断位。
(3) 给$\overline{WR}$一个负脉冲,然后等待RDY/$\overline{BSY}$变高。
注意:$\overline{WR}$必须保持低至少一个毫秒的电平。

九、编程加密位

编程加密位的算法如下(参考Flash编程部分的命令、地址和数据的装载):
(1) 装入命令"0010 0000"。
(2) 装入数据:
位2=0,编程加密位2;
位1=0,编程加密位1。
(3) 给$\overline{WR}$一个负脉冲,然后等待RDY/$\overline{BSY}$变高。
加密位仅在全片擦除的时候被清除。

十、读熔断和加密位

读熔断和加密位的算法如下(参考Flash编程部分的命令、地址和数据的装载):
(1) 装入命令"0000 0100"。
(2) 设置$\overline{OE}$为0,BS为1,这时熔断和加密位的状态可从PB(7~0)读出。
位7:加密位1(0表示已被编程)。
位6:加密位2(0表示已被编程)。
位5:SPIEN熔断位(0表示已被编程,1表示被擦除)。
位0:FSTRT熔断位(0表示已被编程,1表示被擦除)。
(3) 设置$\overline{OE}$为1。
特别注意BS需要设置为1。

十一、读电子标签字节

读电子标签字节的算法如下(参考Flash编程部分的命令、地址和数据的装载):
(1) 装入命令"0000 1000"。
(2) 装入低位地址($00~ $02)。
(3) 设置$\overline{OE}$为0,BS为0,从PB(7~0)可以读出选中的标签字节。
(4) 设置$\overline{OE}$为1。
仅在读第一个字节之前需要装入命令。

2.12.6 串行下载

一、串行下载

在$\overline{RESET}$接地时,所有的程序和数据存储器阵列都可以由串行SPI总线来编程。该串行接口包括引脚SCK、MOSI(输入)、MISO(输出)。当$\overline{RESET}$设为低电平后,应先执行编程允许

指令，再执行编程/擦除操作。

当编程 EEPROM 时，内部定时编程操作中包含了自动擦除周期(仅仅在串行编程模式下)而无须先执行全片擦除指令。全片擦除指令把程序和数据存储器阵列的每一地址都变成 $FF。

程序和 EEPROM 存储器阵列的地址空间是分开的，程序存储器为 $0000～$0FFF，而 EEPROM 存储器为 $0000～$01FF。

可以用通过 XTAL1 提供的外部时钟，也可以在 XTAL1 和 XTAL2 之间加上一个晶振，串行时钟的(SCK)低电平和高电平的最小时间定义如下：

Low：大于 1 个 XTAL1 时钟周期。

High：大于 4 个 XTAL1 时钟周期。

二、串行编程算法

以串行的方式编程和校验 90 系列单片机，可用以下的算法(见表 2.28 的 4 字节指令格式)：

(1) 上电过程：在 V_{CC}和 GND 之间上电，同时$\overline{\text{RESET}}$和 SCK 设置为 0。(如果编程器不能保证 SCK 在上电期间为低电平，则在 SCK 为低电平时$\overline{\text{RESET}}$必须给出一个正脉冲)。如果晶振没有被连到 XTAL1 和 XTAL2 上，则在 XTAL1 上加上 0 到 20 MHz 的时钟。

(2) 等待至少 20 ms，向 MOSI/PB5 送串行编程使能指令来使能串行编程，请参照上一节的串行时钟输入 SCK 的低电平和高电平的最小时间。

(3) 如果执行了全片擦除(在擦除 Flash 时必须执行)，等待 10 ms，给$\overline{\text{RESET}}$一个正脉冲然后再从第(2)步开始。

(4) 通过在相应的写指令中一起提供地址和数据，可以一次把一个字节写入 Flash 和 EEPROM 阵列中，EEPROM 存储器的每一地址在新数据写入之前被自动擦除。下一个字节 4 ms 后再写入。

(5) 任何存储器地址都可以通过读指令来校验，该读指令从串行输出 MISO/PB6 返回选中地址的内容。

(6) 在编程结束时，$\overline{\text{RESET}}$可以设置为高电平来开始正常操作。

(7) 下电过程(如果需要的话)：设置 XTAL1 为 0(如果未用晶振的话)；设置$\overline{\text{RESET}}$为 1；把 V_{CC}断开。

表 2.28 串行可编程指令设置

指令	指令格式				操作
	Byte1	Byte2	Byte3	Byte4	
编程允许	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	在复位变低后，允许串行编程
芯片擦除	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	芯片擦除存储器阵列为 8K 或 512K 字节
读程序存储器	0010 h000	bbbb bbbb	bbbb bbbb	oooo oooo	从字地址 a:b 处的程序存储器读 h(高或低)数据 o
写程序存储器	0100 h000	xxxx aaaa	bbbb bbbb	iiii iiii	写 h(高或低)数据 i 到从字地址 a:b 处的程序存储器中
读 EEPROM 存储器	1010 0000	xxxx aaaa	bbbb bbbb	oooo oooo	从地址 a:b 处的 EEPROM 中读数据 o
写 EEPEOM 存储器	1100 0000	xxxx xxx0	bbbb bbbb	iiii iiii	写数据 i 到地址 a:b 处的程序存储器中
写锁定位	1010 1100	111x x21x	xxxx xxxx	xxxx xxxx	写锁定位，置位 1、2 = '0'到程序锁定位
读器件代码	0011 0000	xxxx xxxx	xxxx xxxb	oooo oooo	从地址 b 处读器件代码 o

注意：a = 高位地址；b = 低位地址；h = 0 为低字节，h1 为高字节；o = 数据输出；i = 数据输入；x = 任意；1 = 加密位 1；2 = 加密位 2。

2.12.7 可编程特性

当把串行数据写入90系列单片机时,数据在CLK的上升沿被输入。

当从90系列单片机中读数据时,数据在CLK的下降沿输出,见图2.53的解释。图2.54是串行可编程和校验示意图。

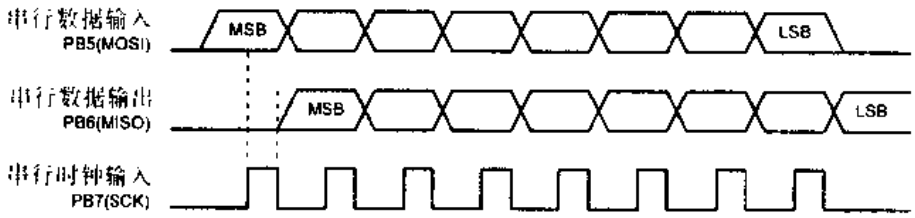


图2.53 串行下载波形图

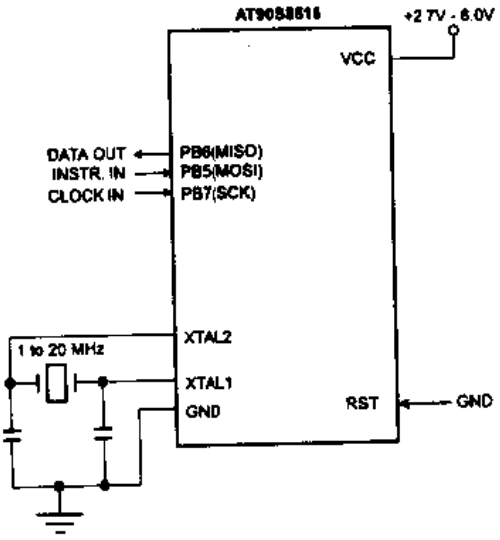


图2.54 串行可编程和校验

第三章　AVR单片机指令系统

计算机的指令系统是一套控制计算机操作的代码，称之为机器语言。计算机只能识别和执行机器语言的指令。为了便于人们理解、记忆和使用，通常用汇编语言指令来描述计算机的指令系统。汇编语言指令可通过汇编器翻译成计算机能识别的机器语言。

AVR单片机指令系统是RISC结构的精简指令集，是一种简明易掌握、效率高的指令系统。120条功能强大的指令，大多数执行时间为单个时钟周期。这一章主要分析AVR单片机指令系统的功能和使用方法。

3.1　指令格式

3.1.1　汇编指令

汇编语言源文件是由汇编语言代码和汇编程序指令所组成的ASCII字符文件。

一、汇编语言源文件

汇编语言源文件包括指令助记符、标号和伪指令。指令助记符和伪指令常带操作数。

每条程序输入行首先是标号，标号为字母数字串，并带一个冒号。使用标号的目的是为了跳转和转移指令及在程序存储器和SRAM中定义变量名。

程序输入行有下列四种型式：

(1)〖标号:〗伪指令〖操作数〗〖注释〗

(2)〖标号:〗指令〖操作数〗〖注释〗

(3)注释

(4)空行

注释有下列型式：〖文字〗

括号内的项是任选的。用于注释的分号(;)及到行结尾的文字，汇编器是忽略的。标号、指令和伪指令在后面有详细说明。

例子：

```
Label:  .EQU Var1 = 100        ;置 Var1 等于 100(伪指令)
        .EQU Var2 = 200        ;置 Var2 等于 200
test:   rjmp  test             ;无限循环(指令)
                               ;纯注释行
                               ;另一个注释行
```

注意：不限制有关标号、伪指令、注释或指令的列位置。

二、指令助记符

汇编器认可指令集中的指令助记符。指令集中综合了助记符并给出了参数。表3.1为指令集表。

表 3.1 指令集

伪指令	操作码	说　　明	操　　作	标　志	时钟数
算术和逻辑指令					
ADD	Rd, Rr	Add without Carry	Rd←Rd + Rr	Z, C, N, V, H	1
ADC	Rd, Rr	Add with Carry	Rd←Rd + Rr + C	Z, C, N, V, H	1
ADIW	Rd, K	Add Immediate to Word	Rd + 1:Rd←Rd + 1:Rd + K	Z, C, N, V	2
SUB	Rd, Rr	Subtret without Carry	Rd←Rd − Rr	Z, C, N, V, H	1
SUBI	Rd, K	Subtract Immediate	Rd←Rd − K	Z, C, N, V, H	1
SBC	Rd, Rr	Subtract with Carry	Rd←Rd − Rr − C	Z, C, N, V, H	1
SBCI	Rd, K	Subtract Immediate wit Carry	Rd←Rd − K · C	Z, C, N, V, H	1
SBIW	Rd, K	Subtract Immediate from Word	Rd + 1:Rd←Rd + 1:Rd − K	Z, C, N, V	2
AND	Rd, Rr	Logical AND	Rd←Rd·Rr	Z, N, V	1
ANDI	Rd, K	Logical AND with Immediate	Rd←Rd·K	Z, N, V	1
OR	Rd, Rr	Logical OR	Rd←Rd ∨ Rr	Z, N, V	1
ORI	Rd, K	Logical OR with Immediate	Rd←Rd ∨ K	Z, N, V	1
EOR	Rd, Rr	Exclusive OR	Rd←Rd ⊕ Rr	Z, N, V	1
COM	Rd	One's Complement	Rd← $FF − Rd	Z, C, N, V	1
NEG	Rd	Two's Complement	Rd← $00 − Rd	Z, C, N, V, H	1
SBR	Rd, K	Set Bit(s) in Register	Rd← $00 ∨ K	Z, N, V	1
CBR	Rd, K	Clear Bit(s) in Register	Rd←Rd·($FF − K)	Z, N, V	1
INC	Rd	Increment	Rd←Rd + 1	Z, N, V	1
DEC	Rd	Decrement	Rd←Rd − 1	Z, N, V	1
TST	Rd	Test for Zero or Minus	Rd←Rd·Rd	Z, N, V	1
CLR	Rd	Clear Register	Rd←Rd ⊕ Rd	Z, N, V	1
SER	Rd	Set Register	Rd← $FF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1, R0←Rd × Rr	C	2
条件转移指令					
RJMP	k	Relative Jump	PC←PC + k + 1	None	2
IJMP		Indirect Jump to(Z)	PC←Z	None	2
JMP	k	Jump	PC←k	None	3
RCALL	k	Relative Call Subroutine	PC←PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC←Z	None	3
CALL	k	Call Subroutine	PC←k	None	4
RET		Subroutine Return	PC←STACK	None	4
RETI		Interupt Return	PC←STACK	1	4
CPSE	Rd, Rr	Compare, Skip if Equal	if(Rd = Rr)PC←PC + 2 or 3	None	1/2

表3.1续

伪指令	操作码	说 明	操 作	标 志	时钟数
CP	Rd, Rr	Compare	Rd－Rr	Z, C, N, V, H,	1
CPC	Rd, Rr	Compare with Carry	Rd－Rr－C	Z, C, N, V, H,	1
CPI	Rd, K	Compare with Immediate	Rd－K	Z, C, N, V, H,	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if(Rr(b)＝0)PC←PC＋2 or 3	None	1/2
SBRS	Rr, b	Skip if Bit in Register Set	if(Rr(b)＝1)PC←PC＋2 or 3	None	1/2
SBIC	P, b	Skip if Bit in I/O Register Cleared	if(I/O(P, b)＝0)PC←PC＋2 or 3	None	2/3
SBIS	P, b	Skip if Bit in I/O Register Set	if(I/O(P, b)＝1)PC←PC＋2 or 3	None	2/3
BRBS	s, k	Branch if Status Flag Set	if(SREG(s)＝1)then PC←PC＋k＋1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if(SREG(s)＝0)then PC←PC＋k＋1	None	1/2
BREQ	k	Branch if Equal	if(Z＝1)then PC←PC＋k＋1	None	1/2
BRNE	k	Branch if Not Equal	if(Z＝0)then PC←PC＋k＋1	None	1/2
BRCS	k	Brench if Carry Set	if(C＝1)then PC←PC＋k＋1	None	1/2
BRCC	k	Brench if Carry Cleared	if(C＝0)then PC←PC＋k＋1	None	1/2
BRSH	k	Brench if Same or Higher	if(C＝0)then PC←PC＋k＋1	None	1/2
BRLO	k	Branch if Lower	if(C＝1)then PC←PC＋k＋1	None	1/2
BRMI	k	Branch if Minus	if(N＝1)then PC←PC＋k＋1	None	1/2
BRPL	k	Branch if Plus	if(N＝0)then PC←PC＋k＋1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if(N ⊕ V＝0)then PC←PC＋k＋1	None	1/2
BRLT	k	Branch if Less Than, Signed	if(N ⊕ V＝1)then PC←PC＋k＋1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if(H＝1)then PC←PC＋k－1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if(H＝0)then PC←PC＋k＋1	None	1/2
BRTS	k	Branch if T Flag Set	if(T＝1)then PC←PC＋k＋1	None	1/2
BRTC	k	Branch if T Flag Cleared	if(T＝0)then PC←PC＋k－1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if(V＝1)then PC←PC＋k＋1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if(V＝0)then PC←PC＋k＋1	None	1/2
BRIE	k	Branch if Interrupt Enabled	if(I＝1)then PC←PC＋k＋1	None	1/2
BRID	k	Branch if Interrupt Disabled	if(I＝0)then PC←PC＋k＋1	None	1/2
数据传送指令					
MOV	Rd, Rr	Copy Register	Rd←Rr	None	1
LDI	Rd, K	Load Immediate	Rd←K	None	1
LDS	Rd, k	Load Direct from SRAM	Rd←(k)	None	3
LD	Rd, X	Load Indirect	Rd←(X)	None	2
LD	Rd, X＋	Load Indirect and Post-Increment	Rd←(X), X←X＋1	None	2
LD	Rd, －X	Load Indirect and Pre-Decrement	X←X－1, Rd←(X)	None	2
LD	Rd, Y	Load Indirect	Rd←(Y)	None	2

表 3.1 续

伪指令	操作码	说　明	操　作	标　志	时钟数
LD	Rd, Y+	Load Indirect and Post-Increment	Rd←(Y), Y←Y+1	None	2
LD	Rd, −Y	Load Indirect and Pre-Decrement	Y←Y−1, Rd←(Y)	None	2
LDD	Rd, Y+q	Load Indirect with Displacement	Rd←(Y+q)	None	2
LD	Rd, Z	Load Indirect	Rd←(Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Increment	Rd←(Z), Z←Z+1	None	2
LD	Rd, −Z	Load Indirect and Pre-Decrement	Z←Z−1, Rd←(Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd←(Z+q)	None	2
STS	k, Rr	Store Direct to SRAM	(k)←Rr	None	3
ST	X, Rr	Store Indirect	(X)←Rr	None	2
ST	X+, Rr	Store Indirect and Post-Increment	(X)←Rr, X←X+1	None	2
ST	−X, Rr	Store Indirect and Pre-Decrement	X←X−1, (X)←Rr	None	2
ST	Y, Rr	Store Indirect	(Y)←Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Increment	(Y)←Rr, Y←Y+1	None	2
ST	−Y, Rr	Store Indirect and Pre-Decrement	Y←Y−1, (Y)←Rr	None	2
STD	Y+q, Rr	Store Indirect with Displacement	(Y+q)←Rr	None	2
ST	Z, Rr	Store Indirect	(Z)←Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Increment	(Z)←Rr, Z←Z+1	None	2
ST	−Z, Rr	Store Indirect and Pre-Decrement	Z←Z−1, (Z)←Rr	None	2
STD	Z+q, Rr	Store Indirect with Displacement	(Z+q)←Rr	None	2
LPM		Load Program Memory	R0←(Z)	None	3
IN	Rd, P	In Port	Rd←P	None	1
OUT	P, Rr	Out Port	P←Rr	None	1
PUSH	Rr	Push Register on Stack	STACK←Rr	None	2
POP	Rd	Pop Register from Stack	Rd←STACK	None	2
位指令和位测试指令					
LSL	Rd	Logical Shift Left	Rd(n+1)←Rd(n), Rd(0)←0, C←Rd(7)	Z, C, N, V, H	1
LSR	Rd	Logical Shift Right	Rd(n)←Rd(n+1), Rd(7)←0, C←Rd(0)	Z, C, N, V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C, Rd(n+1)←Rd(n), C←Rd(7)	Z, C, N, V, H	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C, Rd(n)←Rd(n+1), C←Rd(0)	Z, C, N, V	1
ASR	Rd	Arithmetic Shift Right	Rd(n)←Rd(n+1), n=0～6)	Z, C, N, V	1
SWAP	Rd	Swap Nibbles	Rd(3～0)↔Rd(7～4)	None	1
BSET	s	Flag Set	SREG(s)←1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s)←0	SREG(s)	1
SBI	P, b	Set Bit in I/O Register	I/O(P, b)←1	None	2
CBI	P, b	Clear Bit in I/O Register	I/O(P, b)←0	None	2

表3.1续

伪指令	操作码	说 明	操 作	标 志	时钟数
BST	Rr, b	Bit Store from Register to T	T←Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b)←T	None	1
SEC		Set Carry	C←1	C	1
CLC		Clear Carry	C←0	C	1
SEN		Set Negative Flag	N←1	N	1
CLN		Clear Negative Flag	N←0	N	1
SEZ		Set Zeto Flag	Z←1	Z	1
CLZ		Clear Zero Flag	Z←0	Z	1
SEI		Global Interrupt Enable	I←1	I	1
CLI		Global Interrupt Disable	I←0	I	1
SES		Set Signed Test Flag	S←1	S	1
CLS		Clear Signed Test Flag	S←0	S	1
SEV		Set Two's Complement Overflow	V←1	V	1
CLV		Clear Two's Complement Overflow	V←0	V	1
SET		Set T in SREG	T←1	T	1
CLT		Clear T in SREG	T←1	T	1
SEH		Set Half Carry Flag in SREG	H←1	H	1
CLH		Clear Half Carry Flag in SREG	H←0	H	1
NOP		No Operation		None	1
SLEEP		Sleep		None	1
WDR		Watchdog Reset		None	1

操作数有下列形式：

Rd ：R0～R31 或 R16～R31(取决于指令)。

Rr ：R0～R31。

b ：常数(0～7)，可能是常数表达式。

s ：常数(0～7)，可能是常数表达式 。

p ：常数(0～31/63)可能是常数表达式。

K ：常数(0～255)可能是常数表达式。

k ：常数，值范围取决于指令，可能是常数表达式。

q ：常数(0～63)，可能是常数表达式。

3.1.2 汇编器伪指令

汇编器提供一些伪指令，伪指令并不直接转换成操作数，而是用于调整存储器中程序的位置、定义宏、初始化存储器等。全部伪指令在表3.2中给出。

表 3.2 伪指令表

序号	指 令	说 明	序号	指 令	说 明
1	BYTE	Reserve byte to a variable	10	ESEG	EEPROM Segment
2	CSEG	Code Segment	11	EXIT	Exit from file
3	DB	Define constant byte(s)	12	INCLUDE	Read source from another file
4	DEF	Define a symbolic name on a register	13	LIST	Turn listfile generation on
5	DEVICE	Define which device to assemble for	14	LISTMAC	Turn macro expansion on
6	DSEG	Data Segment	15	MACRO	Begin macro
7	DW	Define constant word(s)	16	NOLIST	Turn listfile generation off
8	ENDMACRO	End macro	17	ORG	Set program origin
9	EQU	Set a symbol equal to an expression	18	SET	Set a symbol to an expression

1. BYTE——保存字节到变量

BYTE 伪指令保存存储的内容到 SRAM 中。为了能提供所要保存的位置，BYTE 伪指令前应有标号。该伪指令带一个表征被保存字节数的参数。该伪指令仅用在数据段内(见伪指令 CSEG, DSEG 和 ESEG)。注意：必须带一个参数，字节数的位置不需要初始化。

语法： LABEL: .BYTE 表达式

2. CSEG——代码段

CSEG 伪指令定义代码段的开始位置。一个汇编文件包含几个代码段，这些代码段在汇编时被连接成一个代码段。在代码段中不能使用 BYTE 伪指令，典型的缺省段为代码段。代码段有一个字定位计数器。ORG 伪指令用于放置代码段和放置程序存储器指定位置的常数。CSEG 伪指令不带参数。

语法： .CSEG

3. DB——在程序存储器或 EEPROM 存储器中定义字节常数

DB 伪指令保存数据到程序存储器或 EEPROM 存储器中。为了提供被保存的位置，在 DB 伪指令前必须有标号。DB 伪指令可带一个表达式表，至少有一个表达式。DB 伪指令必须放在代码段或 EEPROM 段。表达式表是一系列表达式，用逗号分隔。每个表达式必须是 -128～255 之间的有效值。如果表达式有效值是负数，则用 8 位 2 的补码放在程序存储器或 EEPROM 存储器中。如果 DB 伪指令用在代码段，并且表达式表多于一个表达式，则以两个字节组合成一个字放在程序存储器中。如果表达式表是奇数，那么最后一个表达式将独自以字格式放在程序存储器中，而不管下一行汇编代码是否是单个 DB 伪指令。

语法： LABEL: .DB 表达式

4. DEF——设置寄存器的符号名

DEF 伪指令允许寄存器用符号代替。一个定义的符号用在程序中，并指定一个寄存器，一个寄存器可以赋几个符号。符号在后面程序中能再定义。

语法： .DEF 符号 = 寄存器

5. DEVICE——定义被汇编的器件

DEVICE 伪指令允许用户告知汇编器被执行的代码使用那种器件。如果使用该伪指令，

若在代码中有指定的器件不提供的指令,则提示一个警告。如果代码段或EEPROM段的尺寸大于被指定器件的尺寸,也提示警告。如果不使用DEVICE伪指令,则假定器件提供所有的指令,也不限制存储器尺寸。

语法: .DEVICE AT90S1200 | AT90S2313 | AT90S4414 | AT90S8515

6. DSEG——数据段

DSEG伪指令定义数据段的开始。一个汇编文件能包含几个数据段,这些数据段在汇编时被连接成一个数据段。一个数据段正常仅由BYTE伪指令(和标号)组成。数据段有自己的定位字节计数器。ORG伪指令被用于在SRAM指定位置放置变量。DSEG伪指令不带参数。

语法: .DSEG

7. DW——在程序存储器和EEPROM存储器中定义字常数

DW伪指令保存代码到程序存储器或EEPROM存储器,为了提供被保存的位置,在DW伪指令前必须有标号。DW伪指令可带一个表达式表,至少有一个表达式。DW伪指令必须放在代码段或EEPROM段。表达式表是一系列表达式,用逗号分隔。每个表达式必须是—32 768～65 535之间的有效值。如果表达式有效值是负数,则用16位2的补码放在程序存储器中。

语法: LABEL: .DW 表达式表

8. ENDMACRO——宏结束

ENDMACRO伪指令定义宏定义的结束。该伪指令并不带参数,参见MACRO宏定义伪指令。

语法: .ENDMACRO

9. EQU——设置一个符号相等于一个表达式

EQU伪指令赋一个值到标号,该标号用于后面的表达式,用EQU伪指令赋值的标号是一个常数,不能改变或重定义。

语法: .EQU 标号 = 表达式

10. ESEG——EEPROM段

ESEG伪指令定义EEPROM段的开始位置。一个汇编文件包含几个EEPROM段,这些EEPROM段在汇编时被连接成一个EEPROM段。在EEPROM段中不能使用BYTE伪指令。EEPROM段有一个字节定位计数器。ORG伪指令用于放置EEPROM存储器指定位置的常数。ESEG伪指令不带参数。

语法: .ESEG

11. EXIT——退出文件

EXIT伪指令告诉汇编器停止汇编该文件。正常情况下,汇编器汇编到文件的结束。如果EXIT出现在包括文件中,则汇编器从文件中INCLUDE伪指令行继续汇编。

语法: .EXIT

12. INCLUDE——包括另外的文件

INCLUDE伪指令告诉汇编器从指定的文件开始读。然后汇编器汇编指定的文件,直到文件结束或遇到EXIT伪指令。一个包括文件也可能自己用INCLUDE伪指令来表示。

语法: .INCLUDE "文件名"

13. LIST——打开列表文件生成器

LIST伪指令告诉汇编器打开列表文件生成器。汇编器生成一个汇编源代码、地址和操作

代码的文件列表。列表文件生成器缺省值是打开。该伪指令总是与 NOLIST 伪指令一起出现,用于生成列表或汇编源文件有选择的列表。

语法： .LIST

14. LISTMAC——打开宏表达式

LISTMAC 伪指令告诉汇编器,当调用宏时,用列表生成器在列表文件中显示宏表达式。缺省值仅是在列表文件中显示宏调用参数。

语法： .LISTMAC

15. MACRO——宏开始

MACRO 伪指令告诉汇编器这是宏开始。MACRO 伪指令带宏名和参数。当后面的程序中写了宏名,被表达的宏程序在指定位置被调用。一个宏可带 10 个参数。这些参数在宏定义中用@0～@9 代表。当调用一个宏时,参数用逗号分隔。宏定义用 ENDMACRO 伪指令结束。缺省值为汇编器的列表生成器,仅列表宏调用。为了在列表文件中包括宏表达式,必须使用 LISTMAC 伪指令。在列表文件的操作代码域内宏用 a+作记号。

语法： .MACRO 宏名

16. NOLIST——关闭列表文件生成器

NOLIST 伪指令告诉汇编器关闭列表文件生成器。正常情况下,汇编器生成一个汇编源代码、地址和操作代码文件列表。缺省时为打开列表文件,但是可用该伪指令禁止列表。为了使被选择的汇编源文件部分产生列表文件,该伪指令可以与 LIST 伪指令一起使用。

语法： .NOLIST

17. ORG——设置程序起始位置

ORG 伪指令设置定位计数器一个绝对值。设置的值为一个参数。如果 ORG 伪指令放在数据段,则设置 SRAM 定位计数器;如果该伪指令放在代码段,则设置程序存储器计数器;如果该伪指令放在 EEPROM 段,则设置 EEPROM 定位计数器。如果该伪指令前带标号(在相同的源代码行),则标号由参数值给出。代码和 EEPROM 定位计数器的缺省值是零;而当汇编启动时,SRAM 定位计数器的缺省值是 32(因为寄存器占有地址为 0～31)。注意,EEPROM 和 SRAM 定位计数器按字节计数,而程序存储器定位计数器按字计数。

语法： .ORG 表达式

18. SET——设置一个与表达式相等的符号

SET 伪指令赋值给一个标号。这个标号能用在后面的表达式中。用 SET 伪指令赋值的标号在后面的程序中能改变。

语法： .SET 标号 = 表达式

3.1.3 表达式

汇编器包括一些表达式,表达式由操作数、运算符和函数组成。所有的表达式内部为 32 位。

一、操作数

(1) 用户定义的标号,该标号给出了放置标号位置的定位计数器的值。

(2) 用户用 SET 伪指令定义的变量。

(3) 用户用 EQU 伪指令定义的常数。

(4) 整数常数,包括下列几种形式:

· 十进制(缺省值):10,255

· 十六进制数(二进制表示法):oxoa, $ oa, oxff, $ ff

· 二进制数:ob00001010,ob11111111

(5) PC:程序存储器定位计数器的当前值。

二、函 数

(1) LOW(表达式) 返回一个表达式的低字节。

(2) HIGH(表达式) 返回一个表达式的第二个字节。

(3) BYTE2(表达式) 与HIGH函数相同。

(4) BYTE3(表达式) 返回一个表达式的第三个字节。

(5) BYTE4(表达式) 返回一个表达式的第四个字节。

(6) LWRD(表达式) 返回一个表达式的0~15位。

(7) HWRD(表达式) 返回一个表达式的16~31位。

(8) PAGE(表达式) 返回一个表达式的16~21位。

(9) EXP2(表达式) 返回2∧表达式。

(10) LOG2(表达式) 返回LOG2(表达式)的整数部分。

三、运算符

汇编器提供的部分运算符见表3.3。越高的运算符,优先级越高。表达式可以用括号括起来,并且与括号外任意表达式所组合的表达式总是有效的。

表3.3 部分运算符表

序号	名称	符号	优先级	说明
1	逻辑非	!	14	一元运算符,表达式是0返回1,而表达式为非0则返回0
2	逐位非	~	14	一元运算符,输入表达式的所有位倒置
3	负号	-	14	一元运算符,使表达式为算术负
4	乘法	*	13	二进制运算符,两个表达式相乘
5	除法	/	13	二进制运算符,左边表达式除以右边表达式,得整数的商值
6	加法	+	12	二进制运算符,两个表达式相加
7	减法	-	12	二进制运算符,左边表达式减去右边表达式
8	左移	<<	11	二进制运算符,左边表达式左移右边表达式给出的次数
9	右移	>>	11	二进制运算符,左边表达式右移右边表达式给出的次数
10	小于	<	10	二进制运算符,左边带符号表达式小于右边带符号表达式,则为1,否则为0
11	小于等于	<=	10	二进制运算符,左边带符号表达式小于或等于右边带符号表达式,则为1,否则为0
12	大于	>	10	二进制运算符,左边带符号表达式大于右边带符号表达式,则为1,否则为0
13	大于等于	>=	10	二进制运算符,左边带符号表达式大于或等于右边带符号表达式,则为1,否则为0
14	等于	==	9	二进制运算符,左边带符号表达式等于右边带符号表达式,则为1,否则为0
15	不等于	!=	9	二进制运算符,左边带符号表达式不等于右边带符号表达式,则为1,否则为0
16	逐位与	&	8	二进制运算符,两个表达式之间逐位与
17	逐位异或	∧	7	二进制运算符,两个表达式之间逐位异或
18	逐位或	|	6	二进制运算符,两个表达式之间逐位或
19	逻辑与	&&	5	二进制运算符,两个表达式逻辑与,非0则为1,否则为0
20	逻辑或	||	4	二进制运算符,两个表达式逻辑或,非0则为1,否则为0

3.2 寻址方式

指令的一个重要组成部分是操作数，指令给出参与运算的数据的方式称为寻址方式。AVR 单片机指令操作数的寻址方式有以下多种：单寄存器直接寻址、双寄存器直接寻址、I/O 直接寻址、数据直接寻址、带位移的数据间接寻址、数据间接寻址、带预减量的数据间接寻址、带后增量的数据间接寻址、常量寻址、程序直接寻址、程序间接寻址、程序相关寻址。

一、单寄存器直接寻址

由指令指出一个寄存器的内容作为操作数，这种寻址方式称为单寄存器直接寻址。寄存器寻址所选的工作寄存器为寄存器文件中的 031 区域。指令字的低 5 位(D0～D4 位)指出所用的寄存器 Rd。图 3.1 为单寄存器直接寻址示意图。

二、双寄存器直接寻址

双寄存器直接寻址方式同单寄存器直接寻址方式，指令字中指出两个寄存器 Rd 和 Rr。指令字的 D0～D4 位指出 Rd 寄存器，D5～D9 位指出 Rr 寄存器。结果存在 Rd 寄存器中。图 3.2 为双寄存器直接寻址示意图。

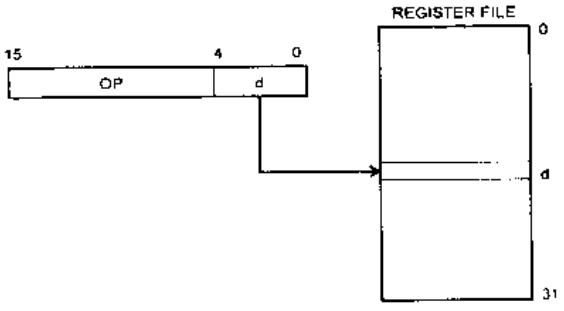


图 3.1 单寄存器直接寻址示意图

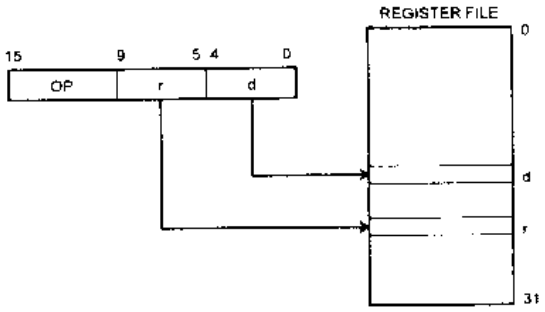


图 3.2 双寄存器直接寻址示意图

三、I/O 直接寻址

在 AVR 单片机的寄存器文件中映射有 I/O 寄存器。指令可以直接对 I/O 空间进行操作。操作数包含在指令字中的 D0～D5 位中，同时指令字中还包含了目的或源寄存器地址 n。图 3.3 为 I/O 直接寻址示意图。

四、数据直接寻址

数据直接寻址方式便于直接从 SRAM 存储器中存取数据。数据直接寻址为双字指令，一个 16 位的数据地址放在低字指令中，高字指令中的 Rd/Rr 指定了目的寄存器或源寄存器。存储器存取的范围限制在 SRAM 当前 64 字节页。图 3.4 为数据直接寻址示意图。

五、带位移的数据间接寻址

带位移的数据间接寻址方式是利用变址寄存器(Y 或 Z)及指令字中的位移量共同决定被存取 SRAM 存储器的地址。操作数的地址由 Y 或 Z 寄存器的内容加上指令字 D0～D5 位给出的位移量 a 给出。指令字中的 n 为目的或源寄存器地址。图 3.5 为带位移的数据间接寻址示意图。

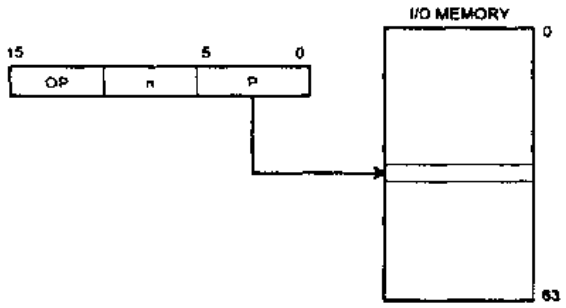


图 3.3 I/O直接寻址示意图

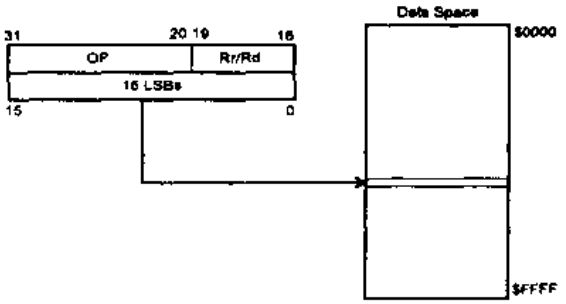


图 3.4 数据直接寻址示意图

六、数据间接寻址

由指令指出某一个寄存器的内容作为操作数的地址,该寻址方式称为寄存器间接寻址。AVR单片机中用变址寄存器X、Y或Z作为规定的寄存器,并对SRAM存储器存取操作,称为数据间接寻址。操作数的地址在变址寄存器(X、Y或Z)中。图3.6为数据间接寻址示意图。

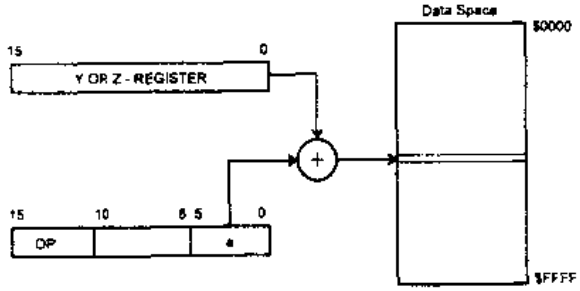


图 3.5 带位移的数据间接寻址示意图

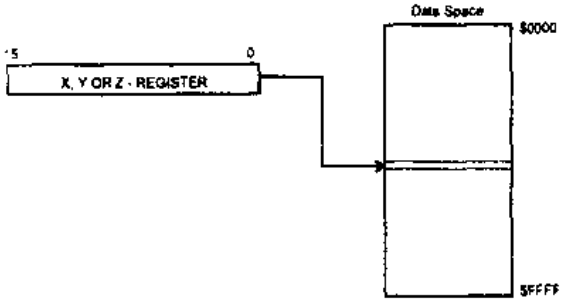


图 3.6 数据间接寻址示意图

七、带预减量的数据间接寻址

同数据间接寻址,但寄存器X、Y或Z的内容在操作之前先被减1,相减后的内容为操作数的地址。这种寻址方式特别适用于访问矩阵、查表等应用。图3.7为带预减量的数据间接寻址示意图。

八、带后增量的数据间接寻址

同数据间接寻址方式,但寄存器X、Y或Z的内容在操作后被加1,而操作数地址的内容为寄存器增量之前的内容。这种寻址方式特别适用于访问矩阵、查表等应用。图3.8为带后增量的数据间接寻址示意图。

九、常量寻址

常量寻址主要从程序存储器取常量。程序存储器中放常量字节的地址由寄存器Z的内容确定。Z寄存器的高15位用于选择字地址(0~4K),而Z寄存器的最低位(D0)用于写字地址的高低字节。若最低位被清除(LSB=0),则选择低字节;若最低位被置位(LSB=1),则选择高字节,例如LPM指令。图3.9为常量寻址示意图。

十、程序直接寻址

程序直接寻址方式中操作数包含在指令字中,即操作数以指令字的形式存放于程序存储

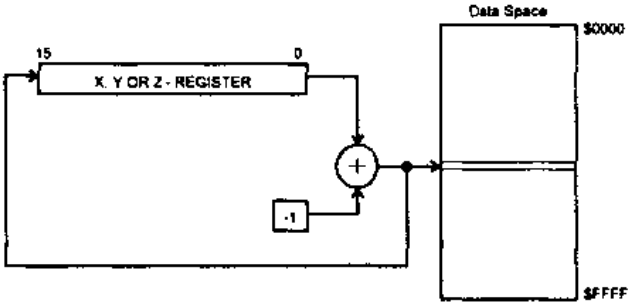


图 3.7 带预减量的数据间接寻址示意图

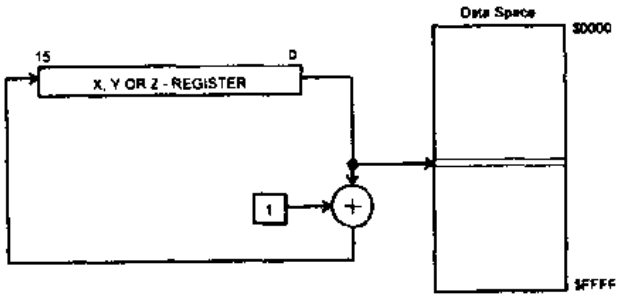


图 3.8 带后增量的数据间接寻址示意图

器中。程序在双指令字中操作数指定的立即地址处执行，如 JMP、CALL 指令。图 3.10 为程序直接寻址示意图。

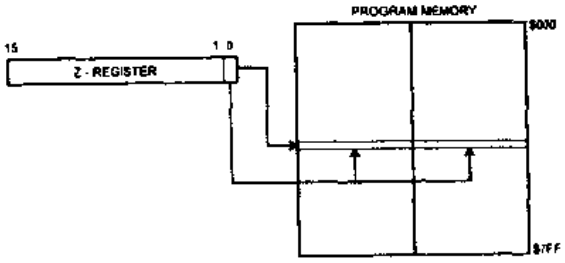


图 3.9 常量寻址示意图

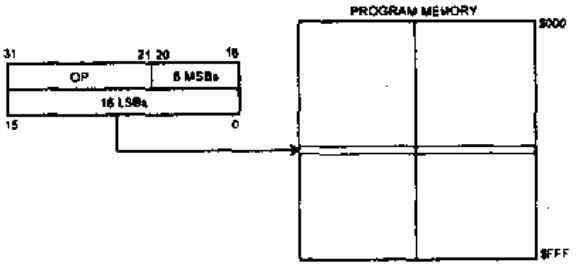


图 3.10 程序直接寻址示意图

十一、程序间接寻址

程序间接寻址方式中，使用 Z 寄存器存放要执行程序的地址。程序在 Z 寄存器的内容指定的地址处继续执行，即用寄存器 Z 的内容代替 PC 的值，如 IJMP、ICALL 指令。图 3.11 为程序间接寻址示意图。

十二、程序相关寻址

程序间接寻址方式中，在指定字中包含了相关地址 K。执行程序时，首先将 PC 值与相关地址 K 相加，得出程序需要继续执行的下一条指令的地址。然后程序在地址 PC + K 处继续执行。其范围从 - 2K 到(2K - 1)之中，如 RJMP、RCALL 指令。图 3.12 为程序相关寻址示意图。

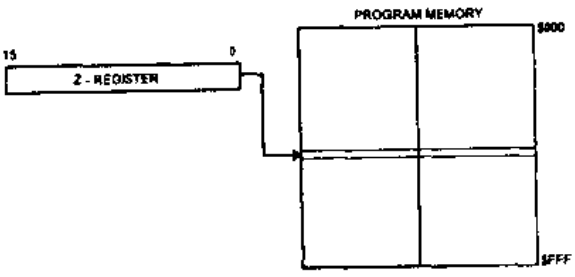


图 3.11 程序间接寻址示意图

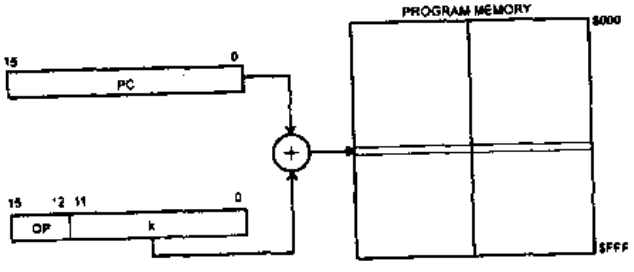


图 3.12 程序相关寻址示意图

3.3 数据操作和指令类型

3.3.1 数据操作

AVR单片机是一个增强型RISC微控制器，具有高性能的数据处理能力，能对位、半字节、字节和双字节数据进行各种操作。它们包括算术和逻辑运算、数据传送、布尔处理和控制转移等操作。

3.3.2 指令类型

AVR单片机共有120条指令。如果按指令字分类，则有116条单字指令、4条双字指令。若按指令执行时间分类，则有56条单周期指令、30条双周期指令、6条3周期指令、3条4周期指令、20条指令可为单周期或双周期、5条指令可为单周期或双周期或三周期指令。可见指令系统主要为单周期，具有存储效率高、执行速度快的特点。

按指令功能分类，则有23条算术和逻辑指令、36条条件转移指令、31条数据传送指令、31条位指令和位测试指令。

3.3.3 指令集名词

1. 状态寄存器

SREG：状态寄存器。

C：进行标志位。

Z：零标志位。

N：负数标志位。

V：2的补码溢出指示位。

S：N⊕V，符号测试位。

H：半进位标志位。

T：用于BLD和BST指令传送位。

I：全局中断使能/禁止标志位。

2. 寄存器和操作码

Rd：寄存器文件中的目的(或源)寄存器。

Rr：寄存器文件中的源寄存器。

R：指令执行后的结果。

K：常数项或字节数据(8位)。

k：程序计数器的常量地址数据。

b：在寄存器文件中或I/O寄存器(3位)中的位。

s：在状态寄存器(3位)中的位。

X，Y，Z：间接地址寄存器(X=R27:R26；Y=R29:R28；Z=R31:R30)。

P：I/O口地址。

q：直接寻址的偏移(6位)。

3. I/O寄存器

RAMPX，RAMPY，RAMPZ：X、Y、Z寄存器的级联寄存器，允许MCU在相连多于64K字节的SRAM整个范围内间接寻址。

4. 堆 栈

STACK:作为返回地址和压栈寄存器的堆栈。

SP:STACK 的堆栈指针。

5.标 志

⇔:由指令引起的有效标志。

0 :由指令清除的标志。

1 :由指令置位的标志。

– :由指令引起的无效标志。

3.4 算术和逻辑指令

AVR 的算术运算指令有加、减、乘、取反、取补、比较指令、增量和减量指令。逻辑运算指令有与、或、异或指令等。

3.4.1 加法指令

1. 不带进位加法

ADD——不带进位加

说明: 两个寄存器不带进位 C 标志加,结果送目的寄存器 Rd。

操作: Rd←Rd + Rr

语法:	操作码:	程序计数器:
ADD Rd, Rr	0≤d≤31, 0≤r≤31	PC←PC + 1

16 位操作码:

0000	11rd	dddd	rrrr

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: $Rd3\cdot Rr3 + Rr3 + \overline{R3} + \overline{R3}\cdot Rd3$

S: N⊕V,

V: $Rd7\cdot Rr7\cdot\overline{R7} + \overline{Rd7}\cdot\overline{Rr7}\cdot R7$

N: R7

Z: $\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

C: $Rd7\cdot Rr7 + Rr7\cdot\overline{R7} + \overline{R7}\cdot Rd7$

例子:
```
add   r1,  r2
add   r28, r28
```

Words: 1(2 bytes)

Cycles: 1

2. 带进位加法

ADC——带进位加

说明: 两个寄存器和 C 标志的内容相加,结果送目的寄存器 Rd。

操作: Rd←Rd + Rr + C

语法:	操作码:	程序计数器:
ADC Rd, Rr	0≤d≤31, 0 ≤ r ≤31	PC←PC + 1

16 位操作码：

0001	11rd	dddd	rrrr

状态寄存器(SREG)布尔格式：

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H：$Rd3 \cdot Rr3 + Rr3 + \overline{R3} + \overline{R3} \cdot Rd3$　　N：R7

S：N⊕V，　　Z：$\overline{Rd7} \cdot \overline{Rr7} \cdot \overline{Rr7} \cdot \overline{R7} \cdot \overline{R7} \cdot \overline{Rd7}$

V：$Rd7 \cdot Rr7 \cdot \overline{R7} \cdot \overline{Rd7} \cdot \overline{Rr7} \cdot R7$　　C：$Rd7 \cdot Rr7 + Rr7 \cdot \overline{R7} + \overline{R7} \cdot Rd7$

例子：
```
add   r2,   r0
adc   r3,   r1
```

Words：1(2 bytes)

Cycles：1

3. 直接数据加法(字)

ADIW——直接数加法

说明：寄存器对于直接数值(0～63)相加，结果放到寄存器对。

操作：Rdh:Rdl ←Rdh:Rdl + K

语法：	操作码：	程序计数器：
ADIW Rdl, K	dl ∈{24, 26, 28, 30}, 0≤K≤63	PC←PC + 1

16 位操作码：

1001	0110	KKdd	KKKK

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S：N⊕V

V：Rdh7R15

N：R15

Z：$\overline{R15} \cdot \overline{R14} \cdot \overline{R13} \cdot \overline{R12} \cdot \overline{R11} \cdot \overline{R10} \cdot \overline{R9} \cdot \overline{R8} \cdot \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

C：$\overline{R15} \cdot Rdh7$

例子：
```
adiw   r24,   1
adiw   r30,   63
```

Words：1(2 bytes)

Cycles：2

4. 增量指令

INC——加 1

说明：寄存器 Rd 的内容加 1，结果送目的寄存器 Rd 中。该操作不改变 SREG 中的 C 标志，所以 INC 指令允许在多倍字长计算中用作循环计数。当对无符号数操作时，仅有 BREQ 和 BRNE 转移指令有效。当对二进制补码值操作时，所有的带符号转移指令都有效。

操作：Rd←Rd + 1

语法： 操作码： 程序计数器：

INC Rd 0≤d ≤ 31 PC←PC + 1

16 位操作码：

1001	010d	dddd	0011

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	–

S：$N \oplus V$ N：R7

V：$R7 \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$ Z：$\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

例子：
```
clr   r22
loop: inc   r22
...
cpi   r22, $4F
brne  loop
nop
```

Words： 1(2 bytes)

Cycles： 1

3.4.2 减法指令

1. 不带进位减法

SUB——不带进位减

说明： 两个寄存器相减，结果送目的寄存器 Rd 中。

操作： Rd←Rd − Rr

语法： 操作码： 程序计数器：

SUB Rd, Rr 0≤d ≤ 31, 0 ≤ r ≤31 PC←PC + 1

16 位操作码：

0001	10rd	dddd	rrrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H：$\overline{Rd3} \cdot Rr3 + Rr3 \cdot R3 + R3 \cdot \overline{Rd3}$ N：R7

S：$N \oplus V$ Z：$\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{Rr3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

V：$Rd7 \cdot \overline{Rr7} \cdot \overline{R7} + \overline{Rd7} \cdot Rr7 \cdot R7$ C：$\overline{Rd7} \cdot Rr7 + Rr7 \cdot R7 + R7 \cdot \overline{Rd7}$

例子：
```
sub  r13, r12        ...
brne noteq           noteq: nop
```

Words： 1(2 bytes)

Cycles： 1

2. 直接数减法(字节)

SUBI——立即数减

说明： 一个寄存器和常数相减，结果送目的寄存器 Rd。该指令工作于寄存器 R16 到 R31 之间，非常适合 X、Y 和 Z 指针的操作。

操作：　Rd←Rd－K

语法：	操作码：	程序计数器：
SUBI Rd, K	16≤ d ≤ 31, 0 ≤ k ≤255	PC← PC + 1

16 位操作码：

0101	kkkk	dddd	kkkk

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
		↔	↔	↔	↔	↔	⇔

H：$\overline{Rd3}\cdot K3 + K3\cdot R3 + R3\cdot\overline{Rd3}$　　N：R7

S：N⊖V　　Z：$R7\cdot R6\cdot R5\cdot R4\cdot Rr3\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

V：$Rd7\cdot K7\cdot\overline{R7} + Rd7\cdot K7\cdot R7$　　C：$\overline{Rd7}\cdot Rr7 + Rr7\cdot R7 + R7\cdot\overline{Rd7}$

例子：
```
subi  r22, $11
brne noteq
...
noteq: nop
```

Words：　1(2 bytes)

Cycles：　1

3. 带进位减法

SBC——带进位减

说明：　两个寄存器随着 C 标志相减，结果放到目的寄存器 Rd 中。

操作：　Rd← Rd－Rr　C

语法：	操作码：	程序计数器：
SBC Rd, Rr	0≤ d ≤ 31, 0 ≤ r ≤31	PC←PC + 1

16 位操作码

0000	10rd	dddd	rrrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
		⇔	↔	⇔	↔	⇔	⇔

H：$\overline{Rd3}\cdot Rr3 + Rr3\cdot R3 + R3\cdot\overline{Rd3}$　　N：R7

S：N⊖V　　Z：$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot R3\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}\cdot Z$

V：$Rd7\cdot\overline{Rr7}\cdot\overline{R7} + \overline{Rd7}\cdot Rr7\cdot R7$　　C：$\overline{Rd7}\cdot Rr7 + Rr7\cdot R7 + R7\cdot\overline{Rd7}$

例子：
```
sub  r2, r0
sbc  r3, r1
```

Words：　1(2 bytes)

Cycles：　1

4. 带进位直接数减

SBCI——带进位立即数减

说明：　寄存器和立即数随着 C 标志相减，结果放到目的寄存器 Rd 中。

操作： Rd←Rd－K－C

语法：	操作码：	程序计数器：
SBCI Rd, K	16≤ d ≤31, 0 ≤ K ≤255	PC←PC+1

16 位操作码：

0100	KKKK	dddd	KKKK

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3}\cdot K3 + K3\cdot R3 + R3\cdot\overline{Rd3}$　　N: R7

S: N⊕V　　Z: $\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}\cdot Z$

V: $Rd7\cdot\overline{K7}\cdot\overline{R7} + \overline{Rd7}\cdot K7\cdot R7$　　C: $\overline{Rd7}\cdot K7 + K7\cdot R7 + R7\cdot\overline{Rd7}$

例子：
```
subi  r16, $r23
sbci  r17, $4F
```

Words： 1(2 bytes)

Cycles： 1

5. 直接数减法(字)

SBIW——立即数减法

说明： 双寄存器与立即数(0～63)减,结果送双寄存器。该指令操作于四个以上的寄存器对,比较适合对指针寄存器操作。

操作： Rdh:Rdl←Rdh:Rdl－K

语法：	操作码：	程序计数器：
SBIW Rdl, K	dl ∈{24, 26, 28, 30}, 0≤K≤63	PC←PC+1

16 位操作码：

1001	0111	KKdd	KKKK

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S: N⊕V

V: $Rdh7\cdot\overline{R15}$

N: R15

Z: $\overline{R15}\cdot\overline{R14}\cdot\overline{R13}\cdot\overline{R12}\cdot\overline{R11}\cdot\overline{R10}\cdot\overline{R9}\cdot\overline{R8}\cdot\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}\cdot Z$

C: $R15\cdot\overline{Rdh7}$

例子：
```
sbiw  r24,  1
sbiw  r28,  63
```

Words： 1(2 bytes)

Cycles： 1

6. 减量指令

DEC——减 1

说明： 寄存器 Rd 的内容减 1，结果送目的寄存器 Rd 中。该操作不改变 SREG 中的 C 标志，所以 DEC 指令允许在多倍字长计算中用作循环计数。当对无符号值操作时，仅有 BREQ 和 BRNE 转移指令有效。当对二进制补码值操作时，所有的带符号转移指令都有效。

操作： Rd←Rd－1

语法：	操作码：	程序计数器：
DEC Rd	0≤ d ≤31	PC←PC＋1

16 位操作码：

1001	010d	dddd	1010

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	⇔	⇔	⇔	⇔	-

S：N⊕V　　N：R7

V：$\overline{R7}\cdot R6\cdot R5\cdot R4\cdot R3\cdot R2\cdot R1\cdot R0$　　Z：$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

例子：
```
        ldi   r17, $10
loop:   add   r1, r2      brne  loop
        dec   r17         nop
```

Words： 1(2 bytes)

Cycles： 1

3.4.3 乘法指令

MUL——乘法

说明： 该指令完成 8 位×8 位→16 位的无符号数乘法操作。被乘数 Rr 和乘数 Rd 是两个寄存器。16 位结果放在 R1(高字节)和 R0(低字节)中。注意，如果被乘数和乘数选择了 R0 或 R1，则当进行乘法后，结果将溢出。

操作： R1，R0←Rr×Rd

语法：	操作码：	程序计数器：
MUL Rd，Rr	0≤ d ≤31，0 ≤ r ≤31	PC←PC＋1

16 位操作码：

0001	11rd	dddd	rrrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	⇔

C： R15

例子：
```
mul   r6,   r5
mov   r6,   r1
mov   r5,   r0
```

Words： 1(2 bytes)

Cycles： 2

3.4.4 取反码指令

COM——取二进制反码

说明： 该指令完成寄存器 Rd 的二进制反码操作。

操作： Rd ← $FF − Rd

语法：	操作码：	程序计数器：
COM Rd	0≤ d ≤ 31	PC←PC + 1

16 位操作码：

1001	010d	dddd	0000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	1

S：N⊕V　　Z：$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

V：0　　C：1

N：R7

例子：
```
com   r4
breq  zero
...
zero: nop
```

Words： 1(2 bytes)

Cycles： 1

3.4.5 取补指令

NEG——二进制补码

说明： 寄存器 Rd 的内容转换成二进制补码，值 $80 是不改变的。

操作： Rd← $00 − Rd

语法：	操作码：	程序计数器：
NEG Rd	0≤ d ≤ 31	PC←PC + 1

16 位操作码：

1001	010d	dddd	0001

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H：$R3\cdot\overline{Rd3}$　　N：R7

S：N⊕V　　Z：$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

V：$R7\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$　　C：R7 + R6 + R5 + R4 + R3 + R2 + R1 + R0

例子：
```
sub   r11,  r0
brpl  positive
```

```
        neg   r11
positive:  nop
```

Words: 1(2 bytes)

Cycles: 1

3.4.6 比较指令

1.寄存器比较

CP——比较

说明: 该指令完成两个寄存器 Rd 和 Rr 相比较操作,而寄存器的内容不改变。该指令后能使用所有条件转移指令。

操作: Rd－Rr

语法:	操作码:	程序计数器:
CP Rd, Rr	0≤ d ≤ 31, 0 ≤ r ≤31	PC←PC + 1

16 位操作码:

1001	01rd	dddd	rrrr

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3}\cdot Rr3 + Rr3\cdot R3 + R3\cdot\overline{Rd3}$　　N: R7

S: N⊕V　　Z: $Rd7\cdot Rr7\cdot\overline{Rr7}\cdot\overline{R7} + \overline{R7}\cdot Rd7$

V: $Rd7\cdot\overline{Rd7}\cdot\overline{R7} + \overline{Rd7}\cdot Rr7\cdot R7$　　C: $\overline{Rd7}\cdot Rr7 + Rr7\cdot R7 + R7\cdot\overline{Rd7}$

例子:

```
        cp    r14, r19
        brne  noteq
        ...
noteq:  nop
```

Words: 1(2 bytes)

Cycles: 2

2. 带进位比较

CPC——带进位比较

说明: 该指令完成寄存器 Rd 的值和寄存器 Rr 加前位进位的值相比较操作。而寄存器的内容不改变。该指令后能使用所有条件转移指令。

操作: Rd－Rr－C

语法:	操作码:	程序计数器:
CPC Rd, Rr	0≤ d ≤ 31, 0 ≤ r ≤31	PC←PC + 1

16 位操作码:

0000	01rd	dddd	rrrr

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3}\cdot Rr3 + Rr3\cdot R3 + R3\cdot\overline{Rd3}$　　N: R7

S: N⊕V　　Z: $\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}\cdot Z$

V: $Rd7\cdot\overline{Rr7}\cdot\overline{R7} + \overline{Rd7}\cdot Rr7\cdot R7$　　C: $\overline{Rd7}\cdot Rr7 + Rr7\cdot R7 + R7\cdot\overline{Rd7}$

例子：

```
cp    r2, r0
cpc   r3, r1
brne  noteq
...
noteq: nop
```

Words: 1(2 bytes)

Cycles: 1

3. 直接数比较

CPI——带直接数比较

说明：该指令完成寄存器 Rd 和常数的比较操作。寄存器的内容不改变。该指令后能使用所有条件转移指令。

操作：Rd－K

语法：	操作码：	程序计数器：
CPI Rd, K	16≤ d ≤31, 0 ≤ K ≤255	PC←PC+1

16 位操作码：

0011	KKKK	dddd	KKKK

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3}\cdot K3 + K3\cdot R3 + R3\cdot\overline{Rd3}$　　N: R7

S: N⊕V　　Z: $\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

V: $Rd7\cdot\overline{K7}\cdot\overline{R7} + \overline{Rd7}\cdot K7\cdot R7$　　C: $\overline{Rd7}\cdot K7 + K7\cdot R7 + R7\cdot\overline{Rd7}$

例子：

```
cpi   r19, 3
brne  error
...
error: nop
```

Words: 1(2 bytes)

Cycles: 1

3.4.7 逻辑与指令

1. 寄存器逻辑与

AND——逻辑与

说明：寄存器 Rd 和寄存器 Rr 的内容为逻辑与，结果送目的寄存器 Rd。

操作：Rd←Rd · Rr

语法： 操作码： 程序计数器：

AND Rd, Rr $0 \leqslant d \leqslant 31, 0 \leqslant r \leqslant 31$ PC←PC + 1

16 位操作码：

0010	00rd	dddd	rrrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S：N⊕V N：R7

V：0 Z：$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

例子：
```
add   r2, r3
ldi   r16, 1
and   r2, r16
```

Words： 1(2 bytes)

Cycles： 1

2. 带直接数与

ANDI——直接数逻辑与

说明： 寄存器 Rd 的内容与常数逻辑与，结果送目的寄存器 Rd。

操作： Rd←Rd · K

语法： 操作码： 程序计数器：

ANDI Rd, K $16 \leqslant d \leqslant 31, 0 \leqslant K \leqslant 255$ PC←PC + 1

16 位操作码：

0111	KKKK	dddd	KKKK

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S：N⊕V N：R7

V：0 Z：$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

例子：
```
andi   r17, $0F
andi   r18, $10
andi   r19, $AA
```

Words： 1(2 bytes)

Cycles： 1

3. 清除寄存器位

CBR——清除进位标志

说明： 清除寄存器 Rd 中的指定位。利用寄存器 Rd 的内容与常数表征码 K 的补码相与完成的，其结果放在寄存器 Rd 中。

操作： Rd←Rd·($FF − K)

语法： 操作码： 程序计数器：

CBR Rd, K 16≤ d ≤ 31, 0 ≤ K ≤255 PC←PC + 1

16 位操作码：

0111	KKKK	dddd	KKKK

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S：N⊕V N：R7

V：0 Z：R7·R6·R5·R4·R3·R2·R1·R0

例子： cbr r16, $F0

cbr r18, 1

Words： 1(2 bytes)

Cycles： 1

4. 测试零或负

TST——测试零或负

说明：测试寄存器是否是零或是负。完成同一寄存器之间的逻辑与操作，而寄存器内容不改变。

操作： Rd←Rd·Rd

语法： 操作码： 程序计数器：

TST Rd 0≤ d ≤ 31 PC←PC + 1

16 位操作码：

0011	00dd	dddd	dddd

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S：N⊕V N：R7

V：0 Z：$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

例子：

```
        tst   r0
        breq  zero
        ...
zero:   nop
```

Words： 1(2 bytes)

Cycles： 1

3.4.8 逻辑或指令

1. 寄存器逻辑或

OR——逻辑或

说明： 完成寄存器 Rd 与寄存器 Rr 的内容逻辑或操作，结果送目的寄存器 Rd 中。

操作： Rd←Rd ∨ Rr

语法：　　操作码：　　程序计数器：

OR Rd, Rr　　0≤ d ≤ 31, 0 ≤ r ≤31　　PC←PC + 1

16 位操作码：

0010	10rd	dddd	rrrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S：N⊕V　　N：R7

V：0　　Z：$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

例子：
```
or   r19, r16
bst  r15, 6
brts ok
...
ok:  nop
```

Words：　1(2 bytes)

Cycles：　1

2. 带直接数或

ORI——直接数逻辑或

说明：　完成寄存器 Rd 的内容与常量逻辑或操作，结果送目的寄存器 Rd 中。

操作：　Rd←Rd ∨ K

语法：　　操作码：　　程序计数器：

ORI Rd, K　　16≤ d ≤ 31, 0 ≤ K ≤255　　PC←PC + 1

16 位操作码：

0110	KKKK	dddd	KKKK

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S：N⊕V　　N：R7

V：0　　Z：$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

例子：
```
ori  r19, $F0
ori  r17, 1
```

Words：　1(2 bytes)

Cycles：　1

3. 置寄存器位

SBR——寄存器位置位

说明：　对寄存器 Rd 中指定位置位。完成寄存器 Rd 和常数表征码 K 之间的逻辑直接数或(ORI)，结果送目的寄存器 Rd。

操作：　Rd←Rd ∨ K

语法： 操作码： 程序计数器：

SBR Rd, K 16≤ d ≤ 31, 0 ≤ K ≤255 PC←PC + 1

16 位操作码：

0110	KKKK	dddd	KKKK

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S：N⊕V N：R7

V：0 Z：$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

例子：
```
sbr   r16, 3
sbr   r17, $17
```

Words： 1(2 bytes)

Cycles： 1

4. 置寄存器

SER——置位寄存器的所有位

说明： 直接装入 $FF 到寄存器 Rd。

操作： Rd← $FF

语法： 操作码： 程序计数器：

SER Rd 16≤ d ≤ 31 PC←PC + 1

16 位操作码：

1110	1111	dddd	1111

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

例子：
```
clr   r16          nop
ser   r17          out   $18, r17
out   $18, r16
```

Words： 1(2 bytes)

Cycles： 1

3.4.9 逻辑异或指令

1. 寄存器异或

EOR——异或

说明： 完成寄存器 Rd 和寄存器 Rr 的内容相逻辑异或操作，结果送目的寄存器 Rd。

操作： Rd←Rd ⊕ Rr

语法： 操作码： 程序计数器：

EOR Rd, Rr 0≤ d ≤ 31, 0 ≤ r ≤31 PC←PC + 1

16 位操作码：

0010	01rd	dddd	rrrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S：N⊕V　　N：R7

V：0　　Z：$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

例子：
```
eor   r4, r4
eor   r0, r22
```

Words：1(2 bytes)

Cycles：1

2. 清除寄存器

CLR——清除寄存器

说明：清除一个寄存器。该指令采用寄存器 Rd 与自己的内容相异或实现的。寄存器的所有位都被清除。

操作：Rd←Rd ⊕ Rd

语法：	操作码：	程序计数器：
CLR Rd	0≤ d ≤31	PC←PC + 1

16 位操作码：

0010	01rd	dddd	dddd

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	0	0	0	1	-

S：0　　N：0

V：0　　Z：1

例子：
```
      clr   r18      cpi   r18, $50
loop: inc   r18      brne  loop
      ...
```

Words：1(2 bytes)

Cycles：1

3.5 转移指令

3.5.1 无条件转移指令

1. 相对跳转

RJMP——相对跳转

说明：相对跳转到 PC－2K 和 PC＋2K(字)范围内的地址。在汇编程序中，标号用于替代相对操作。AVR 微控制器的程序存储器空间不超过 4K 字(8K 字节)，该指令能寻址整个

存储器空间的每个地址位置。

操作： PC←PC + k + 1

语法：	操作码：	程序计数器：
RJMP k	$-2K \leqslant k \leqslant 2K$	PC←PC + k + 1

16 位操作码：

1100	kkkk	kkkk	kkkk

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

例子：

```
cpi   r16, $42      erroer: add   r16, r17
brne  error                 inc   r16
rjmp                ok:     nop
```

Words： 1(2 bytes)

Cycles： 2

2. 间接跳转

IJMP——间接跳转

说明： 间接跳转到由寄存器文件中的 Z(16 位)指针寄存器指向的地址。Z 指针寄存器是 16 位宽，允许在当前程序存储器空间 64K 字(128K 字节)内跳转。

操作： PC←Z(15－0)

PC(15－0)←Z(15－0)

语法：	操作码：	程序计数器：
IJMP	None	See Operation

16 位操作码：

1001	0100	XXXX	1001

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

例子：

```
mov   r30, r0
ijmp
```

Words： 1(2 bytes)

Cycles： 1

3. 长跳转

JMP——跳转

说明： 在整个程序存储空间 4M(字)内跳转，见 RJMP。

操作： PC←k

语法：	操作码：	程序计数器：
JMP k	$0 \leqslant k \leqslant 4M$	PC← k

32 位操作码：

1001	010k	kkkk	110k
kkkk	kkkk	kkkk	kkkk

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：
```
mov  r0, r1
jump  farplc
...
farplc:  nop
```

Words: 2(4 bytes)

Cycles: 3

3.5.2 条件转移指令

条件转移指令是依某种特定的条件转移的指令。条件满足则转移，条件不满足时则顺序执行下面的指令。

一、测试条件符合转移指令

1. 状态寄存器中位置位转移

BRBS——SREG 中的位被置位转移

说明： 条件相对转移，测试 SREG 的某一位，如果该位被置位，则相对 PC 值转移。这条指令相对 PC 转移的方向为：PC－64≤目的寄存器≤PC＋63。参数 K 为 PC 的偏移，用 2 的补码表示。

操作： If SREG(s) = 1 then PC←PC + k + 1, eles PC←PC + 1

语法：	操作码：	程序计数器：
BRBS s, k	0≤ s ≤ 7, －64≤ k ≤ ＋63	PC←PC + k + 1
		PC←PC + 1

16 位操作码：

1111	00kk	kkkk	ksss

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：
```
bst  r0, 3
brbs  6, bitest
...
bitset:  nop
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

2. 状态寄存器中位清除转移

BRBC——SREG 中的位被清除转移

说明： 条件相对转移，测试 SREG 的某一位，如果该位被清除，则相对 PC 值转移。这条指令相对 PC 转移的方向为：PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移，用 2 的补码表示。

操作： If SREG(s)＝0 then PC←PC＋k＋1，eles PC←PC＋1

语法：	操作码：	程序计数器：
BRBC s，k	0≤ s ≤ 7，－64≤ k ≤ ＋63	PC←PC＋k＋1
		PC←PC＋1

16 位操作码：

1111	01kk	kkkk	ksss

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
－	－	－	－	－	－	－	－

例子：

```
cpi    r20，5
brbc   1，noteq
...
noteq: nop
```

Words： 1 (2 bytes)

Cycles： 1 if condition is false

2 if condition is true

3. 相等转移

BREQ——相等转移

说明： 条件相对转移，测试零标志(Z)，如果 Z 位被置位，则相对 PC 值转移。如果在执行 CP、CPI、SUB 或 SUBI 指令后立即执行该指令，且当寄存器 Rd 中无符号或有符号二进制数与寄存器 Rr 中无符号或有符号二进制数相等时，转移将发生。该指令相对 PC 转移的方向为：PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBS 1，K)。

操作： If Rd＝Rr(Z＝1) then PC←PC＋k＋1，eles PC←PC＋1

语法：	操作码：	程序计数器：
BREQ k	－64≤ k ≤ ＋63	PC←PC＋k＋1
		PC←PC＋1

16 位操作码：

1111	00kk	kkkk	k001

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
－	－	－	－	－	－	－	－

例子：

```
cpr1，r2
breqequal
```

```
        ...
equal:  nop
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false

2 if condition is true

4. 不相等转移

BRNE——不相等转移

说明:条件相对转移,测试零标志(Z),如果Z位被清除,则相对PC值转移。如果在执行CP、CPI、SUB或SUBI指令后立即执行该指令,且当在寄存器Rd中的无符号或带符号二进制数不等于寄存器Rr中的无符号或带符号二进制数时,转移将发生。该指令相对PC转移的方向为:PC-64≤目的≤PC+63。参数K为PC的偏移,用2的补码表示(相当于指令BRBC 1,K)。

操作: If Rd≠Rr(Z=0) then PC←PC+k+1, eles PC←PC+1

语法:	操作码:	程序计数器:
BRNE k	-64≤ k ≤ +63	PC←PC+k+1
		PC←PC+1

16位操作码:

1111	01kk	kkkk	k001

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子:

```
        eor     r27, r27
loop:   inc     r27
        ...
        cpi     r27, 5
        brne    loop
        nop
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false

2 if condition is true

5. C标志位置位转移

BRCS——进位位置位转移

说明: 条件相对转移,测试进位标志(C),如果C位被置位,则相对PC值转移。这条指令相对PC转移的方向为:PC-64≤目的≤PC+63。参数K为PC的偏移,用2的补码表示(相当于指令BRBS 0,K)。

操作: If C= 1 then PC←PC+k+1, eles PC←PC+1

语法:	操作码:	程序计数器:
BRCS k	-64≤ k ≤ +63	PC←PC+k+1
		PC←PC+1

16位操作码:

1111	00kk	kkkk	k000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：
```
cpi    r26, $56
brcs   carry
...
carry: nop
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

6. C 标志位清除转移

BRCC——进位位清除转移

说明： 条件相对转移，测试进位标志(C)，如果 C 位被清除，则相对 PC 值转移。这条指令相对 PC 转移的方向为：PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBC 0, K)。

操作： If C= 0 then PC←PC＋k＋1, eles PC←PC＋1

语法：	操作码：	程序计数器：
BRCC k	－64≤ k ≤ ＋63	PC←PC＋k＋1 PC←PC＋1

16 位操作码：

1111	01kk	kkkk	k000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：
```
addr22, r23
brcsnocarry
...
nocarry: nop
```

Words: 1 (2 btes)

Cycles: 1 if condition is false
2 if condition is true

7. 高于或等于转移

BRSH——高于等于转移(无符号)

说明： 条件相对转移，测试进位标志(C)，如果 C 被清零，则相对 PC 值转移。如果在执行 CP、CPI、SUB 或 SUBI 指令后立即执行该指令，且当在寄存器 Rd 中的无符号二进制数高于等于寄存器 Rr 中的无符号二进制数时，转移将发生。该指令相对 PC 转移的方向为：PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBC 0, K)。

操作： If Rd ≥ Rr(C=0)then PC←PC+k+1, eles PC←PC+1

语法：	操作码：	程序计数器：
BRSH k	−64≤ k ≤ +63	PC←PC+k+1
		PC←PC+1

16 位操作码：

1111	01kk	kkkk	k000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

例子：
```
        subi   r19, 4
        brsh   highsm
        ...
highsm: nop
```

Words： 1 (2 bytes)

Cycles： 1 if condition is false

2 if condition is true

8. 低于转移

BRLO——低于转移(无符号)

说明： 条件相对转移，测试进位标志(C)，如果 C 位被置位，则相对 PC 值转移。如果在执行 CP、CPI、SUB 或 SUBI 指令后立即执行该指令，且当在寄存器 Rd 中无符号二进制数低于在寄存器 Rr 中无符号二进制数时，转移将发生。该指令相对 PC 转移的方向为：PC−64≤目的≤PC+63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBS 0, K)。

操作： If Rd < Rr(C=1)then PC←PC+k+1, eles PC←PC+1

语法：	操作码：	程序计数器：
BRLO k	−64≤ k ≤ +63	PC←PC+k+1
		PC←PC+1

16 位操作码：

1111	00kk	kkkk	k000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

例子：
```
        eor   r19, 19      cpi   r19, $10
Loop:   inc   r19          brlo  loop
        ...                nop
```

Words： 1 (2 bytes)

Cycles： 1 if condition is false

2 if condition is true

9. 负数转移

BRMI——负数转移

说明： 条件相对转移，测试负号标志(N)，如果 N 被置位，则相对 PC 值转移。该指令相对 PC 转移的方向为：PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBS 2，K)。

操作： If N＝ 1 then PC←PC＋k＋1，eles PC←PC＋1

语法：	操作码：	程序计数器：
BRMI k	－64≤ k ≤ ＋63	PC←PC＋k＋1
		PC←PC＋1

16 位操作码：

1111	00kk	kkkk	k010

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
－	－	－	－	－	－	－	－

例子：

```
        subi    r18, 4
        brmi    negiative
        ...
negiative:  nop
```

Words： 1 (2 bytes)

Cycles： 1 if condition is false

2 if condition is true

10. 正数转移

BRPL——正数转移

说明： 条件相对转移，测试负号标志(N)，如果 N 被清零，则相对 PC 值转移。该指令相对 PC 转移的方向为：PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBC 2，K)。

操作： If N＝ 0 then PC←PC＋k＋1，eles PC←PC＋1

语法：	操作码：	程序计数器：
BRPL k	－64≤ k ≤ ＋63	PC←PC＋k＋1
		PC←PC＋1

16 位操作码：

1111	01kk	kkkk	k010

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
－	－	－	－	－	－	－	－

例子：

```
        subi    r26, $50
        brmi    positive
        ...
positive:  nop
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false

2 if condition is true

11. 大于或等于转移

BRGE——大于或等于转移(带符号)

说明: 条件相对转移,测试符号标志(S),如果 S 位被清零,则相对 PC 值转移。如果在执行 CP、CPI、SUB 或 SUBI 指令后立即执行该指令,且当在寄存器 Rd 中带符号二进制数大于或等于寄存器 Rr 中带符号二进制数时,转移将发生。该指令相对 PC 转移的方向为:PC - 64≤目的≤PC + 63。参数 K 为 PC 的偏移,用 2 的补码表示(相当于指令 BRBC 4, K)。

操作: If Rd ≥ Rr(N⊕V = 0)then PC←PC + k + 1, eles PC←PC + 1

语法:	操作码:	程序计数器:
BRGE k	- 64≤ k ≤ + 63	PC←PC + k + 1
		PC←PC + 1

16 位操作码:

1111	01kk	kkkk	k100

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子:

```
cprl1, r12
brgegreateq
...
greateq: nop
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false

2 if condition is true

12. 小于转移

BRLT——小于转移(有符号)

说明: 条件相对转移,测试符号标志(S),如果 S 位被置位,则相对 PC 值转移。如果在执行 CP、CPI、SUB 或 SUBI 指令后立即执行该指令,且当在寄存器 Rd 中带符号二进制数小于在寄存器 Rr 中带符号二进制数时,转移将发生。该指令相对 PC 转移的方向为:PC - 64≤目的≤PC + 63。参数 K 为 PC 的偏移,用 2 的补码表示(相当于指令 BRBS 4, K)。

操作: If Rd < Rr(N⊕V = 1)then PC←PC + k + 1, eles PC←PC + 1

语法:	操作码:	程序计数器:
BRLT k	- 64≤ k ≤ + 63	PC←PC + k + 1
		PC←PC + 1

16 位操作码:

1111	00kk	kkkk	k100

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

例子：
```
cp    r16, r1
brlt  less
...
less: nop
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

13. 半进位标志置位转移

BRHS——半进位标志置位转移

说明： 条件相对转移，测试半进位标志(H)，如果 H 被置位，则相对 PC 值转移。该指令相对 PC 转移的方向为：PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBS 5, K)。

操作： If H＝ 1 then PC←PC＋k＋1, eles PC←PC＋1

语法：	操作码：	程序计数器：
BRHS k	－64≤ k ≤ ＋63	PC←PC＋k＋1
		PC←PC＋1

16 位操作码：

1111	00kk	kkkk	k101

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

例子：
```
brshset
...
hset: nop
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

14. 半进位标志清除转移

BRHC——半进位标志被清除转移

说明： 条件相对转移，测试半进位标志(H)，如果 H 位被清零，则相对 PC 值转移。该指令相对 PC 转移的方向为：PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBC 5, K)。

操作： If H＝ 0 then PC←PC＋k＋1, eles PC←PC＋1

语法：	操作码：	程序计数器：
BRHC k	－64≤ k ≤ ＋63	PC←PC＋k＋1
		PC←PC＋1

16 位操作码：

1111	01kk	kkkk	k101

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：
```
brhc hcclear
...
hclear: nop
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

15. T标志置位转移

BRTS——T标志被置位转移

说明： 条件相对转移，测试T标志，如果T被置位，则相对PC值转移。该指令相对PC转移的方向为：PC－64≤目的≤PC＋63。参数K为PC的偏移，用2的补码表示(相当于指令BRBS6, K)。

操作： If T= 1 then PC←PC＋k＋1, eles PC←PC＋1

语法：	操作码：	程序计数器：
BRTS k	－64≤ k ≤ ＋63	PC←PC＋k＋1
		PC←PC＋1

16 位操作码：

1111	00kk	kkkk	k110

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：
```
bst r3, 5
brts tset
...
tset: nop
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

16. T标志清除转移

BRTC——T标志被清除转移

说明： 条件相对转移，测试T标志，如果T被清零，则相对PC值转移。该指令相对PC转移的方向为：PC－64≤目的≤PC＋63。参数K为PC的偏移，用2的补码表示(相当于指令BRBC 6, K)。

操作： If T= 0 then PC←PC＋k＋1, eles PC←PC＋1

语法：	操作码：	程序计数器：
BRTC k	$-64 \leqslant k \leqslant +63$	PC←PC + k + 1
		PC←PC + 1

16 位操作码：

1111	01kk	kkkk	k110

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：

```
bst   r3, 5
brtc  tclear
...
tclear:  nop
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false

2 if condition is true

17. 溢出标志置位转移

BRVS——溢出标志被置位转移

说明： 条件相对转移，测试溢出标志(V)，如果 V 被置位，则相对 PC 值转移。该对 PC 转移的方向为：PC - 64≤目的≤PC + 63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBS3, K)。

操作： If V = 1 then PC←PC + k + 1, eles PC←PC + 1

语法：	操作码：	程序计数器：
BRVS k	$-64 \leqslant k \leqslant +63$	PC←PC + k + 1
		PC←PC + 1

16 位操作码：

1111	00kk	kkkk	k011

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：

```
add   r3, r4
brvs  overfl
...
overfl:  nop
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false

2 if condition is true

18. 溢出标志清除转移

BRVC——溢出标志被转移清除

说明： 条件相对转移，测试溢出标志(V)，如果 V 被清除，则相对 PC 值转移。该对 PC

转移的方向为:PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移,用 2 的补码表示(相当于指令 BRBC3,K)。

操作：　If V＝ 0 then PC←PC＋k＋1, eles PC←PC＋1

语法：	操作码：	程序计数器：
BRVC k	－64≤ k ≤ ＋63	PC←PC＋k＋1
		PC←PC＋1

16 位操作码：

1111	01kk	kkkk	k011

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
－	－	－	－	－	－	－	－

例子：

```
add   r3, r4
brvc  noover
...
noover:  nop
```

Words:　1 (2 bytes)

Cycles:　1 if condition is false

　　2 if condition is true

19. 中断标志使能转移

BRIE——全局中断被使能转移

说明：　条件相对转移,测试全局中断标志(I),如果 I 被置位,则相对 PC 值转移。该指令相对 PC 转移的方向为:PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移,用 2 的补码表示(相当于指令 BRBS 7,K)。

操作：　If I＝ 1 then PC←PC＋k＋1, eles PC←PC＋1

语法：	操作码：	程序计数器：
BRIE k	－64≤ k ≤ ＋63	PC←PC＋k＋1
		PC←PC＋1

16 位操作码：

1111	00kk	kkkk	k111

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
－	－	－	－	－	－	－	－

例子：

```
brieinten
...
inten:  nop
```

Words:　1 (2 bytes)

Cycles:　1 if condition is false

　　2 if condition is true

20. 中断标志禁止转移

BRID——全局中断被禁止转移

说明： 条件相对转移，测试全局中断标志(I)，如果 I 被清零，则相对 PC 值转移。该指令相对 PC 转移的方向为：PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBC 7，K)。

操作： If I＝ 0 then PC←PC＋k＋1，eles PC←PC＋1

语法：	操作码：	程序计数器：
BRID k	－64≤ k ≤ ＋63	PC←PC＋k＋1
		PC←PC＋1

16 位操作码：

1111	01kk	kkkk	k111

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
－	－	－	－	－	－	－	－

例子：
```
brid   intdis
...
intdis:   nop
```

Words： 1 (2 bytes)

Cycles： 1 if condition is false

2 if condition is true

二、测试条件符合跳行转移指令

1. 相等跳行

CPSE——比较相等跳行

说明： 该指令完成两个寄存器 Rd 和 Rr 的比较，若 Rd＝Rr，则跳行执行指令。

操作： If Rd＝ Rr then PC←PC＋2(or 3)eles PC←PC＋1

语法：	操作码：	程序计数器：
CPSE Rd，Rr	0≤ d ≤ 31，0 ≤ r ≤31	PC←PC＋1
		PC←PC＋2
		PC←PC＋3

16 位操作码：

0001	00rd	dddd	rrrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
－	－	－	－	－	－	－	－

例子：
```
inc   r4          neg   r4
cpse  r4，r0      nop
```

Words： 1 (2 bytes)

Cycles： 1

2. 寄存器位清除跳行

SBRC——寄存器位被清零跳行

说明： 该指令测试寄存器某位，如果该位被清零，则跳下一行执行指令。

操作： If Rd(b) = 0 then PC←PC + 2(or 3)eles PC←PC + 1

语法：	操作码：	程序计数器：
SBRC Rr, b	0≤ r ≤31, 0 ≤ b ≤7	PC←PC + 1
		PC←PC + 2
		PC←PC + 3

16 位操作码：

1111	110r	rrrr	Xbbb

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子： sub r0, r1 sub r0, r1

sbrc r0, 7 nop

Words： 1 (2 bytes)

Cycles： 1 if condition is false(no skip)

2 if condition is true(skip is exectued)

3. 寄存器未置位跳行

SBRS——寄存器位置位跳行

说明： 该指令测试寄存器某位，如果该位被置位，则跳下一行执行指令。

操作： If Rr(b) = 1 then PC←PC + 2(or 3)eles PC←PC + 1

语法：	操作码：	程序计数器：
SBRS Rr, b	0≤ r ≤31, 0 ≤ b ≤7	PC←PC + 1
		PC←PC + 2
		PC←PC + 3

16 位操作码：

1111	111r	rrrr	Xbbb

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子： sub r0, r1 neg r0

sbrs r0, 7 nop

Words： 1 (2 bytes)

Cycles： 1 if condition is false(no skip)

2 if condition is true(skip is exectued)

4. I/O 寄存器位清除跳行

SBIC——I/O 寄存器的位清零跳行

说明： 该指令测试 I/O 寄存器某位，如果该位被清零，则跳一行执行指令。该指令在低 32 个 I/O 寄存器内操作，地址为 0～31。

操作： If I/OP, b= 0 then PC←PC + 2(or 3)eles PC←C + 1

语法：	操作码：	程序计数器：
SBIC P, b	0≤ P ≤ 31, 0 ≤ b ≤ 7	PC←PC + 1
		PC←PC + 2
		PC←PC + 3

16 位操作码：

1001	1001	PPPP	Pbbb

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

例子：
```
e2wait: sbic   $1c, 1
        rjmp   e2wait
        nop
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false(no skip)

2 if condition is true(skip is exectued)

5. I/O 寄存器位置位跳行

SBIS——I/O 寄存器的位置位跳行

说明： 该指令测试 I/O 寄存器某位，如果该位被置位，则跳一行执行指令。该指令在低 32 个 I/O 寄存器内操作，地址为 0～31。

操作： If I/O(P, b) = 1 then PC←PC + 2(or 3)eles PC←PC + 1

语法：	操作码：	程序计数器：
SBIS P, b	0≤ P ≤ 31, 0 ≤ b ≤ 7	PC←PC + 1
		PC←PC + 2
		PC←PC + 3

16 位操作码：

1001	1011	PPPP	Pbbb

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

例子：
```
waitset: sbis   $10, 0
         rjmp   waitset
         nop
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false(no skip)

2 if condition is true(skip is exectued)

三、调用和返回指令

在程序设计中通常把具有一定功能模块的公用程序段定义为子程序。为了实现调用子程序的功能,指令系统中都有调用指令。也称转移子程序指令,它与转移指令的区别如下。执行调用子程序时,把下一条指令地址(PC值)保留到堆栈中,即断点保护,然后把子程序的起始地址置入PC。子程序执行完毕,再从断点返回,从断点处继续执行原程序。而转移指令即不保护断点,也不返回原程序。在每个子程序中都必须有返回指令,返回指令的功能就是把调用前压入堆栈的断点弹出置入PC,恢复调用前的原程序。

在一个程序中,子程序中还会调用别的子程序,这称为子程序嵌套。每次调用子程序时必须将下条指令地址保存起来,返回时按后进先出原则依次取出旧PC值。堆栈就是按后进先出规律存取数据的,调用指令和返回指令具有自动的保存和恢复PC内容的功能。

1. 相对调用

RCALL——相对调用子程序

说明: 在(2K字(4K字节)范围内调用子程序。返回地址(RCALL后的指令地址)存储到堆栈(见CALL)。

操作: PC←PC+k+1

PC←PC+k+1

语法:	操作码:	程序计数器:
RCALL k	−2K≤ k ≤ 2K	PC≤ PC+k+1
RCALL k	−2K≤ k ≤ 2K	PC≤ PC+k+1

16位操作码:

1101	kkkk	kkkk	kkkk

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
–	–		–	–	–	–	–

```
例子: rcall routine          ...
      ...                    pop  r14
      routine: push  r14     ret
```

Words: 1(2bytes)

Cycles: 3

2. 间接调用

ICALL——间接调用子程序

说明: 间接调用由寄存器文件中的Z(16位)指针寄存器指向的子程序。Z指针寄存器是16位宽,允许调用当前程序存储空间64K字(128K字节)内的子程序。

操作: PC(15−0)←Z(15−0)

PC(15−0)←Z(15−0)

语法:	操作码:	程序计数器:
ICALL	None	See Operation

16位操作码:

1001	0101	XXXX	1001

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

例子： move r30, r0
icall

Words： 1(2bytes)

Cycles： 3

3. 长调用

CALL——子程序长调用

说明： 在整个程序存储器区内调用子程序。返回地址(调用后返回的指令地址)将存储在堆栈(见 RCALL 指令)中。

操作： PC←k
PC←k

语法：	操作码：	程序计数器：
CALL k	0≤ k ≤64K	PC←kSTACK←PC+2 SP←SP−2
CALL k	0← k ← 4M	PC←kSTACK←PC+2 SP←SP−3

32 位操作码：

1001	010k	kkkk	111k
kkkk	kkkk	kkkk	kkkk

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

```
例子：  mov   r16, r0                check:  cpi   r16, $ 42
        call  check                          breq  error
        nop                                  ret
        ...                                  ...
                                     error:  rjmp  error
```

Words： 2(4 bytes)

Cycles： 4

4. 从子程序返回

RET——子程序返回

说明： 从子程序返回。返回地址从堆栈中弹出。

操作： PC(15−0)←STACK
PC(21−0)←STACK

语法：	操作码：	程序计数器：	堆栈：
RET	None	See Operation	SP←SP+2
RET	None	See Operation	SP←SP+3

16 位操作码：

1001	0101	0XX0	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：
```
call routine        ...
...                 pop r14
routine: push r14   ret
```

Words： 1 (2bytes)

Cycles： 4

5. 从中断程序返回

RETI——中断返回

说明： 从中断程序中返回。返回地址从堆栈中弹出，且全局中断标志被置位。

操作： PC(15-0)←STACK

PC(21-0)←STACK

语法：	操作码：	程序计数器：	堆栈：
RETI	None	See Operation	SP←SP+2
RETI	None	See Operation	SP←SP+3

16 位操作码：

1001	0101	0XX0	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

I：1

例子：
```
...
extint: push r0     pop r0
...                 reti
```

Words： 1 (2bytes)

Cycles： 4

3.6 数据传送指令

数据传送指令是在编程时使用最频繁的一类指令。数据传送指令是否灵活快速对程序的执行速度产生很大影响。数据传送指令执行操作是寄存器与寄存器，寄存器与数据存储器(SRAM)，寄存器与I/O端口之间的数据传送。另外还有从程序存储器直接取数指令LPM以

及 PUSH(压栈)和 POP(出栈)的堆栈指令。

3.6.1 直接数据传送指令

1. 寄存器拷贝数据

MOV——寄存器拷贝

说明： 该指令将一个寄存器拷贝到另一个寄存器。源寄存器 Rr 的内容不改变,而目的寄存器 Rd 拷贝了 Rr 的内容。

操作： Rd←Rr

语法：	操作码：	程序计数器：
MOV Rd, Rr	0≤ d ≤ 31, 0 ≤ r ≤31	PC←PC + 1

16 位操作码：

0010	11rd	dddd	rrrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

例子：
```
mov   r16, r0
call check
...
check: cpi   r16, $11
...
ret
```

Words： 1 (2 bytes)

Cycles： 1

2. SRAM 数据直接送寄存器

LDS--直接从 SRAM 装入

说明： 把 SRAM 中 1 个字节装入到寄存器。必须提供一个 16 位地址。存储器访问被限制在当前 64K 字节的 SRAM 页。超过 64K 字节,LDS 指令使用 RAMPZ 寄存器访问。

操作： Rd←(k)

语法：	操作码：	程序计数器：
LDS Rd, k	0≤ d ≤ 31, 0 ≤ k ≤65535	PC←PC + 2

32 位操作码：

1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

例子：
```
lds   r2, $FF00
add   r2, r1
sts   $FF00, r2
```

Words： 2 (4 bytes)

Cycles： 3

3. 寄存器数据直接送 SRAM

STS——寄存器数据直接送 SRAM

说明： 将寄存器的内容直接存储到 SRAM。必须提供一个 16 位的地址。存储器访问被限制在当前 64K 字节的 SRAM 页。STS 指令使用 RAMPZ 寄存器访问存储器可超过 64K 字节。

操作： (k)←Rr

语法：	操作码：	程序计数器：
STS k, Rr	0≤ r ≤ 31, 0 ≤ k ≤65535	PC←PC + 2

32 位操作码：

1001	001d	dddd	0000
kkkk	kkkk	kkkk	kkkk

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：
```
Lds   r2, $FF00
add   t2, r1
sts   $FF00, r2
```

Words： 2 (4 bytes)

Cycles： 3

4. 立即数送寄存器

LDI——装入立即数

说明： 装入一个 8 位常数直接到寄存器文件中 16～31 的寄存器。

操作： Rd ← K

语法：	操作码：	程序计数器：
LDI Rd, K	16≤ d ≤ 31, 0 ≤ K ≤ 255	PC←PC + 2

16 位操作码：

1110	KKKK	dddd	KKKK

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：
```
clr   r31
ldi   r30, $F0
lpm
```

Words： 1 (2 bytes)

Cycles： 1

3.6.2 间接数据传送指令

一、使用 X 寄存器间接传送数据

1. 使用变址 X 间接将 SRAM 中内容装入到寄存器

LD——使用变址 X 间接将 SRAM 中内容装入到寄存器

说明： 从 SRAM 中间接装入一个字节到寄存器，SRAM 中的位置由寄存器文件中的 X(16 位)指针寄存器指出。存储器访问被限制在当前 64K 字节的 SRAM 页中。为访问另外 SRAM 页，则 I/O 范围内的寄存器 RAMPX 需改变。在指令执行后，X 指针寄存器值要么不改变，要么就加 1 或减 1 操作。使用 X 指针寄存器的这些特性，特别适合于访问矩阵、表和堆栈指针等。

操作： Rd ←(X)

Rd ←(X)　　X←X + 1

X ← X − 1　　Rd←(X)

语法：	操作码：	程序计数器：
LD Rd, X	0 ≤ d ≤31	PC←PC + 1
LD Rd, X +	0 ≤ d ≤31	PC←PC + 1
LD Rd, − X	0 ≤ d ≤31	PC←PC + 1

16 位操作码：

1.	1001	000d	dddd	1100
2.	1001	000d	dddd	1101
3.	1001	000d	dddd	1110

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
−	−	−	−	−	−	−	−

例子：
```
clr   r31                ldi   r30, $23
ldi   r30, $20           ld    r2, x
ld    r0, x+             ld    r3, -x
ld    r1, x              ld    r4, x+2
```

Words： 1(2 bytes)

Cycles： 2

其中包括 X 寄存器指定地址的内容送目的寄存器 Rd，带预减的 X 寄存器指定地址的内容送寄存器 Rd 和带后增量的 X 寄存器指定地址的内容送寄存器 Rd。

2. 使用变址 X 间接将寄存器内容存储到 SRAM

ST——使用变址 X 间接将寄存器内容存储到 SRAM

说明：间接将寄存器的一个字节存储到 SRAM。SRAM 的位置由寄存器文件中的 X(16 位)指针寄存器指出。存储器访问被限制在当前 64K 字节的 SRAM 页中。为访问另外 SRAM 页，则 I/O 范围的寄存器 RAMPX 将被修改。在操作之后，X 指针寄存器要 么不改变，要么是加 1 或减 1。使用 X 指针寄存器的这些特性，特别适合用作堆栈指针。

操作： (X)← Rr

(X)← Rr　　X←X + 1

X ← X − 1　　(X)←Rr

语法：	操作码：	程序计数器：
ST X, Rr	0 ≤ r ≤31	PC←PC + 1
ST X + , Rr	0 ≤ r ≤31	PC←PC + 1
ST X - , Rr	0 ≤ r ≤31	PC←PC + 1

16 位操作码：

1.	1001	001r	rrrr	1100
2.	1001	001r	rrrr	1101
3.	1001	001r	rrrr	1110

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

```
例子：clr   r27                    ldi   r26, $23
      ldi   r26, $24               st    r2 , x
      st    x+ , r0                st    r3, -x
      st    x , r1
```

Words:　1 (2 bytes)

Cycles:　2

其中包括源寄存器 Rr 数据送 X 寄存器指定的地址，源寄存器 Rr 数据送带预减的 X 寄存器指定的地址和源寄存器 Rr 数据送带后增量的 X 寄存器指定的地址 。

二、使用 Y 寄存器间接传送数据

1.使用变址 Y 间接将 SRAM 中的内容装入到寄存器

LD(LDD)——使用变址 Y 间接将 SRAM 中的内容装入到寄存器

说明： 带或不带偏移地间接从 SRAM 中装入一个字节到寄存器，SRAM 中的位置由寄存器文件中的 Y(16 位)指针寄存器指出。存储器访问被限制在当前 64K 字节的 SRAM 页中。为访问另外 SRAM 页，则 I/O 范围内的寄存器 RAMPY 需改变。在指令执行后，Y 指针寄存器值要么不改变，要么就加 1 或减 1 操作。使用 Y 指针寄存器的这些特性，特别适合于访问矩阵、表和堆栈指针等。

操作：　Rd ←(Y)

　　Rd ←(Y)

　　Y←Y - 1　　Y←Y + 1

　　Rd ←(Y + q)　　Rd←(Y)

语法：	操作码：	程序计数器：
LD Rd, Y	0 ≤ d ≤31	PC←PC + 1
LD Rd, Y +	0 ≤ d ≤31	PC←PC + 1
LD Rd, - Y	0 ≤ d ≤31	PC←PC + 1
LDD Rd, Y + q	0 ≤ d ≤31, 0 ≤ q ≤63	PC←PC + 1

16 位操作码：

1.	1000	000d	dddd	1000
2.	1001	000d	dddd	1001
3.	1001	000d	dddd	1010
4.	10q0	qq0d	dddd	1qqq

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：
```
clr   r29
ldi   r28, $20
ld    r0 , y+
ld    r1 , y
ldi   r28, $23
ld    r2 , y
ld    r3, -y
ldd   r4, y+2
```

Words： 1 (2 bytes)

Cycles： 2

其中包括 Y 寄存器指定地址的内容送目的寄存器 Rd，带预减的 Y 寄存器指定地址的内容送目的寄存器 Rd，带后增量的 Y 寄存器指定地址的内容送目的寄存器 Rd，带位移间接数据送目的寄存器 Rd。

2. 使用变址 Y 间接将寄存器内容存储到 SRAM

ST(STD)——使用变址 Y 间接将寄存器内容存储到 SRAM

说明： 间接将带或不带偏移的寄存器的一个字节存储到 SRAM。SRAM 的位置由寄存器文件中的 Y(16 位)指针寄存器指出。存储器访问被限制在当前 64K 字节的 SRAM 页。为访问另外 SRAM 页，则 I/O 范围的寄存器 RAMPY 将被修改。在操作之后，Y 指针寄存器要么不改变，要么是加 1 或减 1。使用 Y 指针寄存器的这些特性，特别适合用作堆栈指针。

操作： (Y)← Rr

(Y)← Rr　　Y←Y+1

Y ← Y-1　　(Y)←Rr

(Y+q)← Rr

语法：	操作码：	程序计数器：
ST Y, Rr	0 ≤ r ≤31	PC←PC+1
ST Y+, Rr	0 ≤ r ≤31	PC←PC+1
ST -Y, Rr	0 ≤ r ≤31	PC←PC+1
STD Y+q, Rr	0 ≤ r ≤31, 0 ≤ q ≤63	PC←PC+1

16 位操作码：

1.	1000	001r	rrrr	1000
2.	1001	001r	rrrr	1001
3.	1001	001r	rrrr	1010
4.	10q0	qq1r	rrrr	1qqq

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：
```
clr   r29
ldi   r28, $20
st    y+ , r0
st    y , r1
ldi   r28, $23
st    y , r2
st    -y, r3
std   y+2, r4
```

Words： 1 (2 bytes)

Cycles： 2

其中包括源寄存器 Rr 数据送 X 寄存器指定的地址，Rr 送带预减的 Y 寄存器指定的地址，Rr 送带后增量的 Y 寄存器指定的地址，源寄存器 Rd 数据送带位移间接的地址。

三、使用 Z 寄存器间接传送数据

1. 使用变址 Z 间接将 SRAM 中的内容装入到寄存器

LD(LDD)——使用变址 Z 间接将 SRAM 中的内容装入到寄存器

说明：带或不带偏移地间接从 SRAM 中装入一个字节到寄存器，SRAM 中的位置由寄存器文件中的 Z(16 位)指针寄存器指出。存储器访问被限制在当前 64K 字节的 SRAM 页中。为访问另外 SRAM 页，则 I/O 范围内的寄存器 RAMPZ 需改变。在指令执行后，Z 指针寄存器值要么不改变，要么就加 1 或减 1 操作。使用 Z 指针寄存器的这些特性，特别适合于堆栈指针，因为 Z 指针寄存器能用于直接子程序调用，直接跳转和查表。Z 指针寄存器用作为专用堆栈指针要比 X、Y 指针方便。

用 Z 指针在程序存储器中查表，可参见 LPM 指令。

操作：
```
Rd ←(Z)
Rd ←(Z)
Z ← Z - 1
Rd ←(Z + q)
```

语法：	操作码：	程序计数器：
LD Rd, Z	0 ≤ d ≤31	PC←PC + 1
LD Rd, Z+	0 ≤ d ≤31	PC←PC + 1
LD Rd, - Z	0 ≤ d ≤31	PC←PC + 1
LDD Rd, Z + q	0 ≤ d ≤31, 0 ≤ q ≤63	PC←PC + 1

16 位操作码：

1.	1000	000d	dddd	0000
2.	1001	000d	dddd	0001
3.	1001	000d	dddd	0010
4.	10q0	qq0d	dddd	0qqq

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：
```
clr   r31
ldi   r30, $20
ld    r0 , z+
ld    r1 , z
ldi   r30, $23
ld    r2 , z
ld    r3, -z
ldd   r4, z+2
```

Words: 1 (2 bytes)

Cycles: 2

其中包括 Z 寄存器指定地址的内容送目的寄存器 Rd,带预减的 Z 寄存器指定地址的内容送目的寄存器 Rd,带后增量的 Z 寄存器指定地址的内容送目的寄存器 Rd,带位移间接数据送目的寄存器 Rd。

2. 使用变址 Z 间接将寄存器内容存储到 SRAM

ST(STD)——使用变址 Z 间接将寄存器内容存储到 SRAM

说明： 间接将带或不带偏移的寄存器的一个字节存储到 SRAM。SRAM 的位置由寄存器文件中的 Z(16 位)指针寄存器指出。存储器访问被限制在当前 64K 字节的 SRAM 页。为访问另外 SRAM 页,则 I/O 范围的寄存器 RAMPZ 将被修改。在操作之后,Z 指针寄存器要么不改变,要么是加 1 或减 1。使用 Z 指针寄存器的这些特性,特别适合用作堆栈指针。因为 Z 指针寄存器能适用于间接子程序调用,间接跳转和查表,所以 Z 指针寄存器像一个专用堆栈指针,用起来比 X 和 Y 指针更方便。

操作： (Z)← Rr

(Z)← Rr　　Z←Z+1

Z← Z-1　　(Z)←Rr

(Z+q)← Rr

语法：	操作码：	程序计数器：
ST Z,Rr	0 ≤ r ≤31	PC←PC+1
ST Z+,Rr	0 ≤ r ≤31	PC←PC+1
ST -Z,Rr	0 ≤ r ≤31	PC←PC+1
STD Z+q,Rr	0 ≤ r ≤31, 0 ≤ q ≤63	PC←PC+1

16 位操作码：

1.	1000	001r	rrrr	0000
2.	1001	001r	rrrr	0001
3.	1001	001r	rrrr	0010
4.	10q0	qq1r	rrrr	0qqq

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：
```
clr   r31
ldi   r30, $20
st    z+ , r0
st    z , r1
ldi   r30, $23
st    z , r2
st    -z, r3
std   z+2, r4
```

Words: 1 (2 bytes)

Cycles: 2

其中包括源寄存器 Rr 数据送 Z 寄存器指定的地址,源寄存器 Rr 数据送带预减的 Z 寄存器指定的地址,源寄存器 Rr 数据送带后增量的 Z 寄存器指定的地址,源寄存器数据送带位移间接的地址。

3.6.3 从程序存储器直接取数据指令

LPM——装入程序存储器

说明: 将 Z 寄存器指向的一个字节装入到寄存器 0(R0)。该指令使 100% 空间有效常量初始化或常数取数特别有用。程序存储器被编为 16 位字,Z(16 位)指针的最低位(LSB)选择为 0 是低字节,选择为 1 是高字节。该指令能寻址程序存储器第一个 64K 字节(32 字)。

操作: R0 ←(Z)

语法:	操作码:	程序计数器:
LPM	None	PC←PC + 1

16 位操作码:

1001	0101	110X	1000

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子:
```
clr   r31
ldi   r30, $F0
lmp
```

Words: 1 (2 bytes)

Cycles: 3

3.6.4 I/O 口数据传送

1. I/O 口数据装入到寄存器

IN——I/O 口数据装入到寄存器

说明: 将 I/O 空间(口,定时器,配置寄存器等)的数据装入到寄存器文件中的寄存器 Rd 中。

操作: Rd ← P

语法:	操作码:	程序计数器:
IN Rd, P	0 ≤ d ≤31, 0 ≤ P ≤63	PC←PC + 1

16 位操作码:

1011	0PPd	dddd	PPPP

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：
```
        in   r25, $16
        cpi  r25, 4
        ...
exit:   nop
```
Words: 1 (2 bytes)

Cycles: 1

2. 寄存器数据送 I/O 口

OUT——寄存器数据送 I/O 口

说明：将寄存器文件中寄存器 Rr 的数据存储到 I/O 空间(口、定时器、配置寄存器等)。

操作：P ← Rr

语法：	操作码：	程序计数器：
OUT P, Rr	0 ≤ r ≤31, 0 ≤ P ≤63	PC←PC + 1

16 位操作码：

1011	1PPr	rrrr	PPPP

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：
```
clr   r16
ser   r17
out   $18, r16      nop
                    out   $18, r17
```
Words: 1 (2 bytes)

Cycles: 1

3.6.5 堆栈操作指令

AVR 单片机的特殊功能寄存器中有一个堆栈指针 SP。它指出栈顶的位置，在指令系统中有两条用于数据传送的栈操作指令。

1. 进栈指令

PUSH——压寄存器到堆栈

说明：该指令存储寄存器 Rr 的内容到堆栈。

操作：STACK ← Rr

语法：	操作码：	程序计数器：
PUSH Rr	0 ≤ d ≤31	PC←PC + 1SP←SP − 1

16 位操作码：

1001	001d	dddd	1111

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：

```
        call    routine          …
        …                        pop   r13
routine: push   r14              pop   r14
        push    r13              ret
```

Words： 1 (2 bytes)

Cycles： 2

2. 出栈指令

POP——堆栈弹出到寄存器

说明： 该指令将堆栈中的字节装入到寄存器 Rd 中。

操作： Rd ← STACK

语法：	操作码：	程序计数器：
POP Rd	0 ≤ d ≤31	PC←PC + 1SP←SP + 1

16 位操作码：

1001	000d	dddd	1111

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

例子：

```
call    routine          …
…                        pop   r13
routine: push   r14      pop   r14
push    r13              ret
```

Words： 1 (2 bytes)

Cycles： 2

3.7 位指令和位测试指令

AVR 单片机指令系统中有四分之一的指令为位和位测试指令。位指令的灵活应用，极大地提高了系统的逻辑控制和处理能力。

3.7.1 带进位逻辑操作指令

1. 逻辑左移

LSL——逻辑左移

说明： 寄存器 Rd 中所有位左移 1 位。第 0 位被清除，第 7 位移到 SREG 中的 C 标志。该指令完成一个无符号数乘 2 的操作。

操作： C ← b7……b0 ← 0

语法：	操作码：	程序计数器：
LSL Rd	0 ≤ d ≤31	PC←PC + 1

16 位操作码：

0000	11dd	dddd	dddd

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H：Rd3　　N：R7

S：N⊕V　　Z：$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

V：N⊕C　　C：Rd7

例子：
```
add   r0, r4
lsl   r0
```

Words：1(2 bytes)

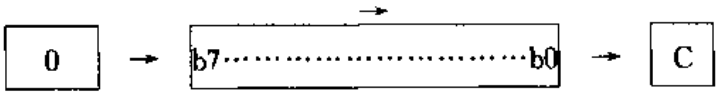
Cycles：1

2. 逻辑右移

LSR——逻辑右移

说明：寄存器 Rd 中所有位右移 1 位。第 7 位被清除，第 0 位移到 SREG 中的 C 标志。该指令完成一个无符号数除 2 的操作。C 标志被用于结果舍入。

操作：



语法：　　操作码：　　程序计数器：

LSR Rd　　0 ≤ d ≤31　　PC←PC + 1

16 位操作码：

1001	010d	dddd	0110

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	0	⇔	⇔

S：N⊕V　　Z：$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

V：N⊕C　　C：Rd0

N：0

例子：
```
add   r0, r4
lsr   r0
```

Words：1(2 bytes)

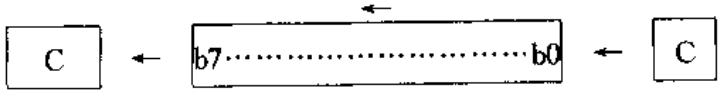
Cycles：1

3. 通过进位左循环

ROL——通过进位左循环

说明：寄存器 Rd 的所有位左移 1 位，C 标志被移到 Rd 的第 0 位，Rd 的第 7 位移到 C 标志。

操作：



语法： 操作码： 程序计数器：

ROL Rd 0 ≤ d ≤31 PC←PC+1

16 位操作码：

0001	11dd	dddd	dddd

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H：Rd3 N：R7

S：N(⊕V Z：$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

V：N⊕C C：Rd7

例子：
```
rolr15
brcsoneenc
...
oneenc:  nop
```

Words： 1(2 bytes)

Cycles： 1

4. 通过进位右循环

ROR——通过进位右循环

说明： 寄存器 Rd 的所有位右移 1 位，C 标志被移到 Rd 的第 7 位，Rd 的第 0 位 移到 C 标志。

操作： C → b7……b0 → C

语法： 操作码： 程序计数器：

ROR Rd 0 ≤ d ≤31 PC←PC+1

16 位操作码：

1001	010d	dddd	0111

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	⇔	⇔	⇔	⇔	⇔

S：N⊕V Z：$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

V：N⊕C C：Rd0

N：R7

例子：
```
rollr15
brcsoneenc
...
oneenc:  nop
```

Words： 1(2 bytes)

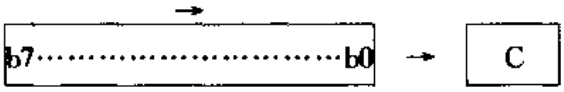
Cycles： 1

5. 算术右移

ASR——算术右移

说明： 寄存器 Rd 中的所有位右移 1 位，而位 7 保持常量，位 0 被装入 SREG 的 C 标 志位。这个操作实现 2 的补码值除 2，而不改变符号，进位标志用于结果的舍入。

操作：



语法：	操作码：	程序计数器：
ASR Rd	0 ≤ d ≤31	PC←PC + 1

16 位操作码：

1001	010d	dddd	0101

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S：N⊕V　　Z：R7·R6·R5·R4·R3·R2·R1·R0

V：N⊕C　　C：Rd0

N：R7

例子：
```
ldi   r16, $10
asr   r16
ldi   r17, $FC
asr   r17
```

Words： 1(2 bytes)

Cycles： 1

6. 半字节交换

SWAP——半字节交换

说明： 寄存器中的高半字节和低半字节交换。

操作： R(7～4)←Rd(3～0)，R(3～0)←Rd(7～4)

语法：	操作码：	程序计数器：
SWAP Rd	0 ≤ d ≤31	PC←PC + 1

16 位操作码：

1001	010d	dddd	0010

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

例子：
```
inc   r1          inc   r1
swap  r1          swap  r1
```

Words： 1(2 bytes)

Cycles： 1

3.7.2 位变量传送指令

1. 寄存器中的位存储到 SREG 中的 T 标志

BST——寄存器中的位存储到 SREG 中的 T 标志

说明： 把寄存器中的位 b 存储到 SREG(状态寄存器)中的 T 标志。

操作： T ←Rd(b)

语法：	操作码：	程序计数器：
BST Rd, b	0 ≤ d ≤31, 0 ≤ b ≤7	PC←PC + 1

16 位操作码：

1111	101d	dddd	Xbbb

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	⇔	-	-	-	-	-	-

T： 0 if bit b in Rd is cleared . Set to 1 otherwise.

例子：
```
bst   r1, 2
bld   r0, 4
```

Words： 1(2 bytes)

Cycles： 1

2. SREG 中的 T 标志装入寄存器中的某一位

BLD——位装入，将 SREG 中的 T 标志装入到寄存器中的某一位。

说明： 拷贝 SREG(状态寄存器)的 T 标志到寄存器 Rd 中的位 b。

操作： Rd(b)←T

语法：	操作码：	程序计数器：
BLD Rd, b	0 ≤ d ≤31, 0 ≤ b ≤7	PC←PC + 1

16 位操作码：

1111	100d	dddd	0bbb

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：
```
bst   r1, 2
bld   r0, 4
```

Words： 1(2 bytes)

Cycles： 1

3.7.3 位变量修改指令

1. 置状态寄存器的位

BSET——置状态寄存器的位

说明： 置状态寄存器(SREG)的某一标志或某一位。

操作： SREG(s)← 1

语法：	操作码：	程序计数器：
BSET s	0 ≤ s ≤7	PC←PC + 1

16 位操作码：

1001	0100	0sss	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
⇔	⇔	⇔	⇔	⇔	⇔	⇔	⇔

I： 1 if s = 7　　V： 1 if s = 3

T： 1 if s = 6　　N： 1 if s = 2

H： 1 if s = 5　　Z： 1 if s = 1

S： 1 if s = 4　　C： 1 if s = 0

例子： bset 6

bset 7

Words： 1(2 bytes)

Cycles： 1

2. 清状态寄存器的位

BCLR——SREG 中的位清除

说明： 清除 SREG 状态寄存器中的一个标志位。

操作： SREG(s)← 0

语法：	操作码：	程序计数器：
BCLR s	0 ≤ s ≤7	PC←PC + 1

16 位操作码：

1001	0100	1sss	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
⇔	⇔	⇔	⇔	⇔	⇔	⇔	⇔

I： 0 if s = 7　　V： 0 if s = 3

T： 0 if s = 6　　N： 0 if s = 2

H： 0 if s = 5　　Z： 0 if s = 1

S： 0 if s = 4　　C： 0 if s = 0

例子： bclr 0

bclr 7

Words： 1(2 bytes)

Cycles： 1

3. 置 I/O 寄存器的位

SBI——置 I/O 寄存器的位

说明： 对 I/O 寄存器指定的位置位，该指令在低 32 个 I/O 寄存器内操作，I/O 寄存器地

址为0~31。

操作： I/O(P,b)← 1

语法：	操作码：	程序计数器：
SBI P,b	0 ≤ P ≤31,0 ≤ b ≤7	PC←PC+1

16 位操作码：

1001	1010	PPPP	Pbbb

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

例子：
```
out   $1E, r0
sbi   $1C, 0
in    r1, $1D
```

Words： 1(2 bytes)

Cycles： 2

4. 清 I/O 寄存器的位

CBI——清 I/O 寄存器的位

说明： 清除 I/O 寄存器中的指定位,该指令用在寄存器最低的 32 个 I/O 寄存器上,I/O 寄存器地址为 0~31。

操作： I/O(P,b)← 0

语法：	操作码：	程序计数器：
CBI P,b	0 ≤ P ≤31,0 ≤ b ≤7	PC←PC+1

16 位操作码：

1001	1000	PPPP	Pbbb

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

C：1

例子：
```
cbi   $12, 7
```

Words： 1(2 bytes)

Cycles： 2

5. 置进位位

SEC——置位进位标志

说明： 置位 SREG(状态寄存器)中的进位标志(C)。

操作： C ← 1

语法：	操作码：	程序计数器：
SEC	None	PC←PC+1

16 位操作码：

1001	0100	0000	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	1

C：1

例子： sec
adc r0, r1

Words： 1(2 bytes)

Cycles： 1

6. 清进位位

CLC——清除进位标志

说明： 清除 SREG(状态寄存器)中的进位标志(C)。

操作： C ← 0

语法：	操作码：	程序计数器：
CLC	None	PC←PC + 1

16 位操作码：

1001	0100	1000	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	0

C：0

例子： add r0, r0
clc

Words： 1(2 bytes)

Cycles： 1

7. 置负标志位

SEN——置位负数标志

说明： 置位 SREG(状态寄存器)中的负数标志(N)。

操作： N ←1

语法	操作码：	程序计数器：
SEN	None	PC←PC + 1

16 位操作码：

1001	0100	0010	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	1	–	–

N：1

例子： add r2, r19
sen

Words： 1(2 bytes)

Cycles： 1

8. 清负标志位

CLN—清除负数标志

说明:清除 SREG(状态寄存器)中的负数标志(N)。

操作： N ← 0

语法：	操作码：	程序计数器：
CLN	None	PC←PC + 1

16 位操作码：

1001	0100	1010	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	0	-	-

N: 0

例子： add r2, r3
cln

Words： 1(2 bytes)

Cycles： 1

9. 置零标志位

SEZ——置位零标志

说明： 置位 SREG(状态寄存器)中的零标志(Z)。

操作： Z ← 1

语法：	操作码：	程序计数器：
SEZ	None	PC←PC + 1

16 位操作码：

1001	0100	0001	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	1	-

Z: 1

例子： add r2, r19
sez

Words： 1← 2 bytes)

Cycles： 1

10. 清零标志位

CLZ——清除零标志

说明： 清除 SREG(状态寄存器)中的零标志(Z)。

操作： Z ← 0

语法：	操作码：	程序计数器：
CLZ	None	PC←PC + 1

16 位操作码：

1001	0100	1001	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	0	–

Z：0

例子：
```
add   r2, r3
clz
```

Words： 1(2 bytes)

Cycles： 1

11. 使能全局中断位

SEI——置位全局中断标志

说明： 置位 SREG(状态寄存器)中的全局中断标志(I)。

操作： I ← 1

语法：	操作码：	程序计数器：
SEI	None	PC←PC + 1

16 位操作码：

1001	0100	0111	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
1	–	–	–	–	–	–	–

I：1

例子：
```
cli
in    r13, $16
sei
```

Words： 1(2 bytes)

Cycles： 1

12. 禁止全局中断位

CLI——清除全局中断标志

说明： 清除 SREG(状态寄存器)中的全局中断标志(I)。

操作： I ← 0

语法：	操作码：	程序计数器：
CLI	None	PC←PC + 1

16 位操作码：

1001	0100	1111	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
0	–	–	–	–	–	–	–

I：0

例子： cli
in r11，$16
sei

Words： 1(2 bytes)

Cycles： 1

13. 置S标志位

SES——置位符号标志

说明： 置位SREG(状态寄存器)中的符号标志(S)。

操作： S ← 1

语法：	操作码：	程序计数器：
SES	None	PC←PC + 1

16位操作码：

1001	0100	0100	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	1	–	–	–	–

S：1

例子： add r2，r19
ses

Words： 1(2 bytes)

Cycles： 1

14. 清S标志位

CLS——清除符号标志

说明： 清除SREG(状态寄存器)中的符号标志(S)。

操作： S ← 0

语法：	操作码：	程序计数器：
CLS	None	PC←PC + 1

16位操作码：

1001	0100	1100	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	0	–	–	–	–

S：0

例子： add r2, r3
cls
Words： 1(2 bytes)
Cycles： 1

15. 置溢出标志位

SEV——置位溢出标志

说明： 置位 SREG(状态寄存器)中的溢出标志(V)。

操作： V ← 1

语法：	操作码：	程序计数器：
SEV	None	PC←PC + 1

16 位操作码：

1001	0100	0011	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	1	–	–	–

V：1

例子： add r2, r19
sev
Words： 1(2 bytes)
Cycles： 1

16. 清溢出标志位

CLV——清除溢出标志

说明： 清除 SREG(状态寄存器)中的溢出标志(V)。

操作： V ← 0

语法：	操作码：	程序计数器：
CLV	None	PC←PC + 1

16 位操作码：

1001	0100	1011	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	0	–	–	–

V：0

例子： add r2, r3
clv
Words： 1(2 bytes)
Cycles： 1

17. 置 T 标志位

SET——置位 T 标志

说明:置位 SREG(状态寄存器)中的 T 标志。

操作： T ← 1

语法：	操作码：	程序计数器：
SET	None	PC←PC + 1

16 位操作码：

1001	0100	0110	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
	1	–	–	–	–	–	–

T：1

例子： set

Words： 1(2 bytes)

Cycles： 1

18. 清 T 标志位

CLT——清除 T 标志

说明： 清除 SREG(状态寄存器)中的 T 标志。

操作： T ← 0

语法：	操作码：	程序计数器：
CLT	None	PC←PC + 1

16 位操作码：

1001	0100	1110	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	0	–	–	–	–	–	–

T：0

例子： clt

Words： 1(2 bytes)

Cycles： 1

19. 置半进位标志

SEH——置位半进位标志

说明： 置位 SREG(状态寄存器)中的半进位标志(H)。

操作： H ← 1

语法：	操作码：	程序计数器：
SEH	None	PC←PC + 1

16 位操作码：

1001	0100	0101	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	1	–	–	–	–	–

H: 1

例子： seh

Words： 1(2 bytes)

Cycles： 1

20. 清半进位标志

CLH——清除半进位标志

说明： 清除 SREG(状态寄存器)中的半进位标志(H)。

操作： H ← 0

语法：	操作码：	程序计数器：
CLH	None	PC←PC + 1

16 位操作码：

1001	0100	1101	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	0	–	–	–	–	–

H: 0

例子： clh

Words： 1(2 bytes)

Cycles： 1

3.7.4 其它指令

1. 空指令

NOP——空操作

说明： 该指令完成一个单周期空操作。

操作： No

语法：	操作码：	程序计数器：
NOP	None	PC←PC + 1

16 位操作码：

0000	0000	0000	0000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

例子：
```
clr   r16
ser   r17
out   $18, r16
nop
```

```
out   $18, r17
```

Words: 1(2 bytes)

Cycles: 1

2. 休眠指令

SLEEP——休眠

说明: 该指令设置电路休眠模式,由 MCU 控制寄存器定义。当在休眠状态由一个中断唤醒时,在中断程序执行后,紧跟在休眠指令后的指令被执行。

操作:

语法:	操作码:	程序计数器:
SLEEP	None	PC←PC+1

16 位操作码:

1001	0101	100x	1000

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子:
```
mov   r0, r11
sleep
```

Words: 1(2 bytes)

Cycles: 1

3. 看门狗复位

WDR——看门狗复位

说明: 该指令复位看门狗定时器,在 WD 预定比例器给出限定时间内必须执行。参见看门狗定时器硬件部分。

操作: WD timer restart.

语法:	操作码:	程序计数器:
WDR	None	PC←PC+1

16 位操作码:

1001	0101	101x	1000

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子:
```
wdr
```

Words: 1(2 bytes)

Cycles: 1

第四章　AVR 单片机 AT90 系列介绍

4.1　AT90S1200 单片机

AT90S1200 是 AVR 增强性能 RISC 结构的低功耗 CMOS 技术八位微控制器，图 4.1 为其结构图。

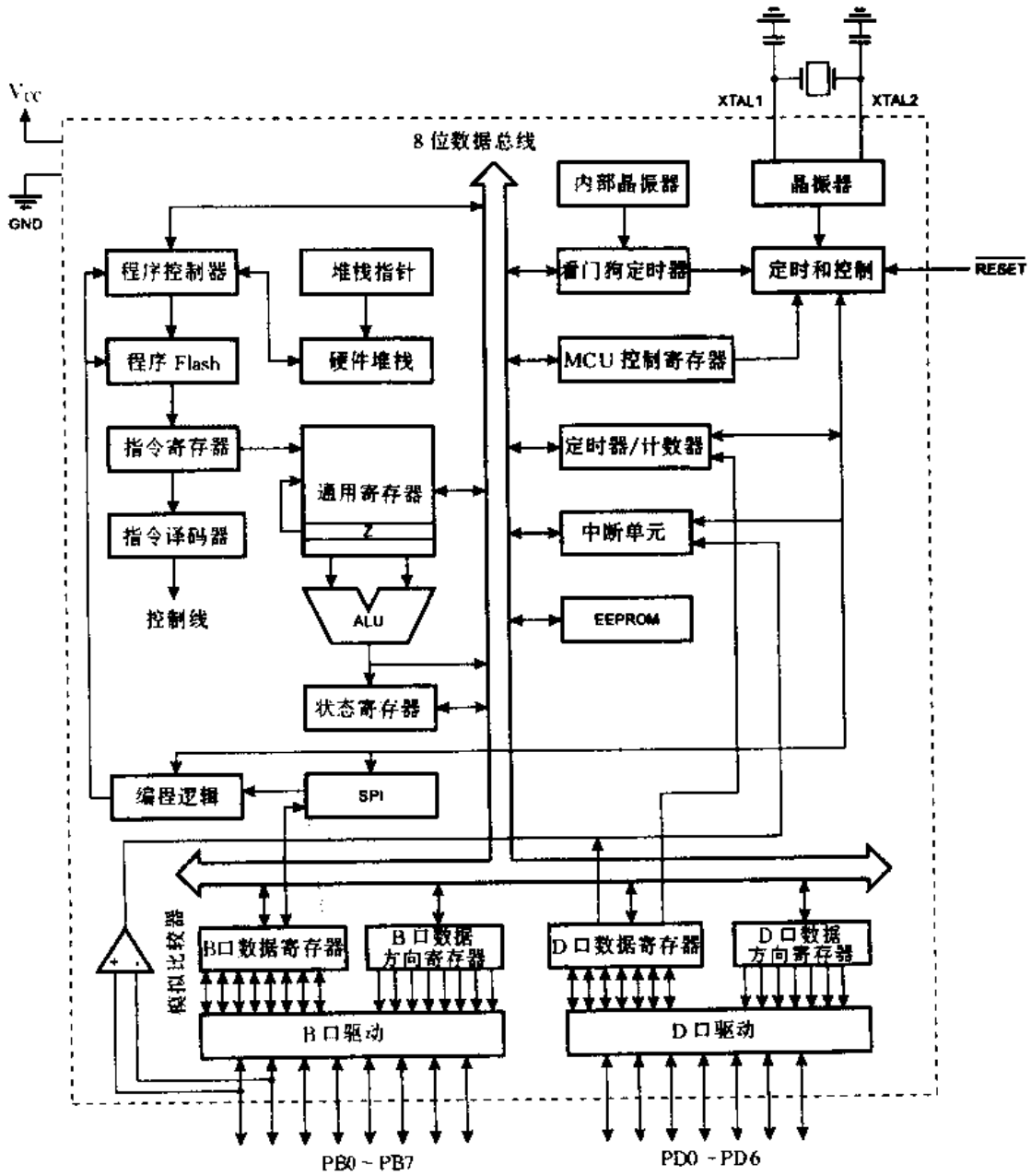


图 4.1　AT90S1200 结构图

该结构有效地支持高级语言,以及密集度极大的汇编器代码程序。该单片机具有以下特征:供电电压在 V_{CC}为 2.7 ~ 6 V 内可以全静态工作范围为 0~16 MHz;89 条功能强大的指令,大多数执行时间为单个时钟周期,指令周期时间为 62.5 ns @ 16 MHz,1K 字节可下载的 Flash 存储器(程序下载采用 SPI 串行接口,使用寿命为 1 000 次),64 字节 EEPROM(使用寿命为 10 万次);15 条可编程 I/O 线,32 个 8 位通用寄存器;内部及外部中断,片内模拟比较器,带有一个 8 位可预分频的定时器/计数器,带内部晶振的可编程看门狗定时器;可选的片内 RC 振荡器,无需外部器件;一个为下载程序而设计的串行口,以及 2 个可通过软件选择的电源保留模式。闲置模式停止 CPU 的工作,而寄存器、定时器/计数器、看门狗及中断系统继续工作。掉电模式保留寄存器的内容,但冻结晶振、终止芯片的其它功能,直至下一次外部中断或硬件复位。

4.1.1 引脚说明

AT90S1200 为 20 引脚 PDIP 和 SOIC 封装的单片机,图 4.2 为其引脚图。

一、引脚定义

V_{CC}(20 脚):供电引脚。

GND (10 脚):接地引脚。

B 口(PB7~PB0):B 口为一个 8 位双向 I/O 口,引脚可提供内部拉高(供每一位选用)。PB0 和 PB1 还单独作为片内模拟比较器的正极输入(AIN0)和负极输入(AIN1)。

$\overline{\text{RESET}}$(1 脚):复位输入。当晶振运行时,引脚上一个两周期的低电流可对器件进行复位。

XTAL1(5 脚):向晶振放大器的输入和向内部时钟操作电路的输入。

XTAL2(4 脚):从反转晶振放大器的输出。

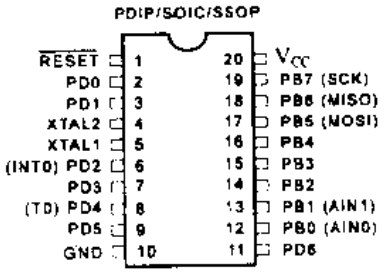


图 4.2 AT90S1200 的引脚图

二、晶振器

XTAL1 和 XTAL2 单独地作为反转放大器的输入和输出,该放大器可被设置为片内的晶振器。为了由外部源驱动器件,当 XTAL1 被驱动时,XTAL2 不能连接。详情可参考第二章。

4.1.2 片内 RC 晶振器

AT90S1200 单片机片内有一个 RC 晶振器,它以固定的 1 MHz 频率运行,该频率可被选为 MCU 的时钟源。当该振荡器激活时,AT90S1200 即可运行而无需外部部件。被编程后(0),Flash 存储器中的控制位 RCEN 选择片内的 RC 晶振作为时钟源。AT90S1200 发货时,该位未被编程(1)。这一位已被编程的器件名称为 AT90S1200A。RCEN 位只能通过并行编程来改变。当使用片内 RC 振荡器进行串行程序下载时,RCEN 位必须先由并行编程模式编程。

4.1.3 AVR RISC 微控制器 CPU

AT90S1200 AVR RISC 微控制器向上与 AVR 增强 RISC 结构兼容。为 AT90S1200 MCU 而编写的程序,与 AVR 8 位 MCU(AT90SXXXX)系列产品的源代码和运行的时钟周期相兼容。

一、AVR 的结构

图 4.3 所示为 AT90S1200 AVR 的结构。

通过相关的转移和调用指令，全部的 512 个地址空间可以被直接访问到。所有 AVR 指令均有一个单一的 16 位字格式，这意味着每个程序存储器地址包含了一个单一的 16 位指令。

I/O 存储器空间包含 64 个作为 CPU 外围功能的地址。为控制寄存器、定时器/计数器及其它 I/O 功能，AVR 结构中的存储器空间全部为线性的和规律的存储器映射。

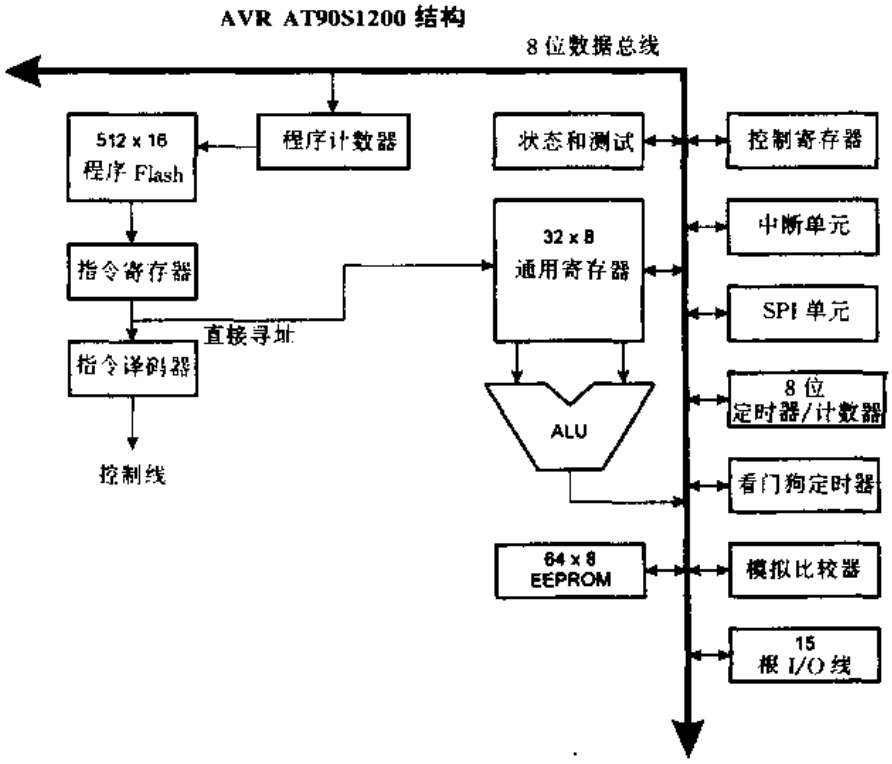


图 4.3 AT90S1200 AVR 的结构

二、通用寄存器文件

图 4.4 所示为在 CPU 中，32 个通用寄存器文件的结构图。寄存器 30 还作为寄存器文件间接地址的 8 位指针。

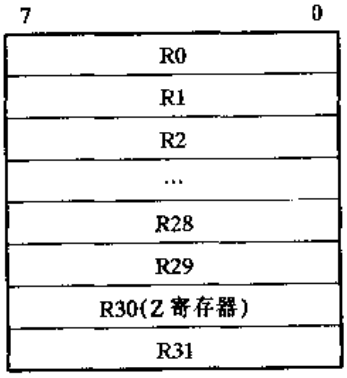


图 4.4 32 个通用寄存器文件的结构图

三、ALU 运算逻辑单元

高性能的 AVR 运算逻辑单元的操作与所有的 32 个通用寄存器保持直接的连接。在单一时钟周期内，ALU 是在寄存器文件中的寄存器之间执行操作。ALU 操作被分为 3 个主要的目的：算法、逻辑和位功能。AVR 产品家族中一些微控制器的 ALU 运算部件中有硬件乘法器。

四、可下载的 Flash 程序存储器

AT90S1200 包括 1K 字节的片内可下载 Flash 存储器。由于所有指令均为单一 16 位字，用于存储程序的 Flash 的结构为 512 字。Flash 存储器的使用寿命最少为 1 000 次写/擦循环。AT90S1200 程序设计器宽为 9 位，以此来对 512 字的 Flash 程序存储器寻址。

五、编程和数据寻址模式

AT90S1200 单片机的寻址模式为寄存器直接寻址(单一寄存器 Rd)、寄存器间接、寄存器直接(两个寄存器 Rd 和 Rr)、I/O 直接和相关程序寻址 5 种。有关寻址模式请 参考第三章。

六、子程序和中断硬件堆栈

AT90S1200 运用一个 3 级深的硬件堆栈来为子程序和中断服务。硬件堆栈为 9 位宽,且当子程序和中断被执行时,存储程序计数器 PC 返回地址。

七、EEPROM 数据存储器

AT90S1200 包括 64 字节的 EEPROM 存储器。它被组织为一个分开的数据空间,在其间单一的字节可被读写。EEPROM 的使用寿命为至少 100 000 次写/擦循环。

八、存储器访问指令执行时序

CPU 由系统时钟 Φ 驱动,直接由芯片的外部时钟晶振激活,没有使用内部时钟部分。AVR 单片机的 Harvard 结构和快速访问寄存器文件概念,能使得并行指令存取和指令执行。这种基本的流水线概念目的是为了获得高达每 MHz 1MIPS 的效率。有关指令执行和内部存储器访问的通用访问时序请参考第二章。

九、I/O 存储器

表 4.1 所示为 AT90S1200 的 I/O 空间定义。

表 4.1 AT90S1200 的 I/O 空间

十六进制地址	名 称	功 能
$3F($5F)	SREG	状态寄存器
$3B($5B)	GIMSK	通用中断屏蔽寄存器
$39($59)	TIMSK	定时器/计数器中断屏蔽寄存器
$38($58)	TIFR	定时器/计数器中断标志寄存器
$35($55)	MCUCR	MCU 通用控制寄存器
$33($53)	TCCR0	定时器/计数器 0 控制寄存器
$32($52)	TCNT0	定时器/计数器 0(8 位)
$21($41)	WDTCR	看门狗定时控制寄存器
$1E($3E)	EEAR	EEPROM 地址寄存器
$1D($3D)	EEDR	EEPROM 数据寄存器
$1C($3C)	EECR	EEPROM 控制寄存器
$18($38)	PORTB	B 口数据寄存器
$17($37)	DDRB	B 口数据方向寄存器
$16($36)	PINB	B 口输入脚
$12 ($32)	PORTD	D 口数据寄存器
$11($31)	DDRD	D 口数据方向寄存器
$10($30)	PIND	D 口输入脚
$08($28)	ACSR	模拟比较控制和状态寄存器

注意:保留的和未用到的地址未列。

AVR 的状态寄存器 SREG 在 I/O 空间的地址为 $3F,SREG 的位定义请参考第二章。

十、复位和中断处理

1. 复位源

AT90S1200 提供 3 种不同的中断源。这些中断和与之分开的复位向量均在程序存储器空间中各自带有分开的程序向量。所有中断均分配给单独的使能位，这些使能位须与状态寄存器中的 I 位一起被设为 1，以便能执行中断。程序存储器空间中最下面的地址被自动地定义为复位和中断向量，如表 4.2 所列。表中还列出了不同中断的优先级，地址越低，优先级越高。RESET 有最高的优先级，以下依次为 INT0……。

表 4.2 复位和中断向量

向量号	程序地址	源	中断定义
1	$000	RESET	硬件脚和看门狗复位
2	$001	INT0	外部中断请求 0
3	$002	INT1	外部中断请求 1
4	$003	TIMER0, OVF0	定时器/计数器 0 溢出
5	$004	ANA-COMP	模拟比较器

AT90S1200 有 3 个复位源：

- 加电复位。当供电电流加至 V_{CC}和 GND 引脚时，MCU 进行复位。
- 外部复位。当一个低电平加到$\overline{\text{RESET}}$引脚多于 2 个 XTAL 周期时，MCU 进行复位。
- 看门狗复位。当看门狗定时器超时，且看门狗为使能时，MCU 进行复位。

在复位过程中，I/O 寄存器被设为初始值，程序从地址 $000 开始执行。$000 地址中放置的指令须为某一 RJMP——相关转移，即到达复位处理路径的指令。若程序从没有对中断源使能，则中断向量无法使用，正常的程序代码可以放置在这些地址中。图 4.5 的电路图说明

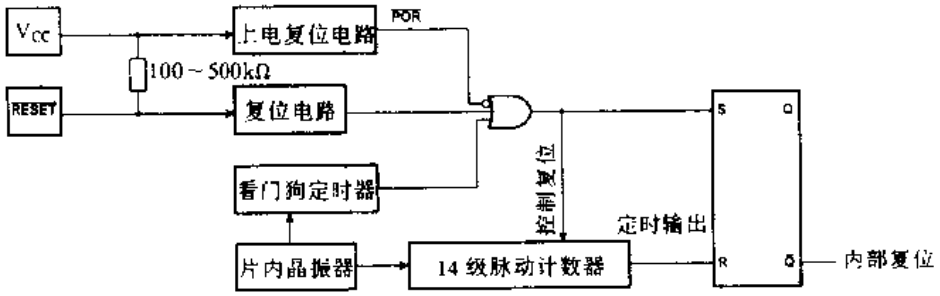


图 4.5 复位电路图

了复位逻辑。表 4.3 定义了复位电路的时序和电参数。注意，加电复位由内部 PC 晶振定时，请参考其它电压下 RC 晶振的特性化数据。有关复位电路分析参考第二章。

表 4.3 复位特性(V_{CC} = 5.0 V)

符 号	参 数	最小值	典型值	最大值	单位值
V_{POT}	上电复位门限电压	1.8	2	2.2	V
V_{RST}	复位脚门限电压		$V_{CC}/2$		V
t_{POR}	上电复位周期	2	3	4	ms
t_{TOUT}	复位延时定时输出。周期 FSTRT 编程	11	16	21	ms

2. 中断处理

AT90S1200 有 2 个中断屏蔽控制寄存器。它们是位于 I/O 空间 $3B 位的通用中断屏蔽寄存器 GIMSK 和位于 I/O 空间 $39 位的定时器/计数器中断屏蔽寄存器。

十一、休眠模式

为了进入休眠状态,MCUCR 中的 SE 位被设为 1,且须执行一条 SLEEP 指令。当 MCU 在休眠模式下发生了一个允许的中断,MCU 唤醒,执行中断例行程序,并恢复 SLEEP 指令的执行。寄存器文件的内容和 I/O 存储器不可改变。若在休眠模式下发生复位,MCU 唤醒,并由复位向量执行。

注意:若为了从掉电状态下唤醒 MCU 而使用了某一电平触发的中断,必须保持 16 ms 的低电平,即长于晶振启动的时间。否则,在 MCU 开始执行以前,中断标志位有可能返回零值。

1. 闲置模式

当 SM 位被清为 0,SLEEP 指令使 MCU 进入闲置状态,这时 CPU 停止工作,而定时器/计时器、看门狗以及中断系统继续工作。这使得 CPU 可以由外部触发的中断来唤醒,亦可由内部诸如定时器溢出中断和看门狗复位来唤醒。若通过模拟比较器唤醒,需要一个中断,可通过设置模拟比较器和状态寄存器 ACSR 中的 ACD 位来对模拟比较器进行掉电。这就在闲置状态下减少了电的功耗。

2. 掉电模式

当 SM 位被设为 1 时,SLEEP 指令使 MCU 进入掉电状态。在此模式下,外部晶振停止工作。用户可选择在掉电模式下看门狗是否使能。若看门狗为使能,当看门狗超过超时规定的时间时,它会唤醒 MCU。若看门狗为非使能,只有外部的复位或外部电平触发中断可唤醒 MCU。

4.1.4 定时器/计数器

AT90S1200 提供一种通用目的的 8 位定时器/计数器。定时器/计数器从 10 位的预定标定时器获取预定标时钟。定时器/计数器可被用作带内部时基的定时器,或被用作带外部引脚连接的计数器,其引脚连接触发计数。

一、定时器/计数器预定标

通用定时器/计数器的预定标器请参考第二章。

二、8 位定时器/计数器 0

8 位的定时器/计数器 0 可从 CK、预定标 CK,或外部引脚来选择时钟源。另外,它还可以像定时器/计数器 0 的控制寄存器 TCCR0 格式中所描述的那样而停止。定时器/计数器 0 的 TCCR0 和 TCNT0 请参考第二章。

4.1.5 看门狗定时器

看门狗定时器由片内一个 1 MHz 分开的晶振定时。通过控制看门狗定时器的预定标器,看门狗的复位间歇从 16 周期到 2 048 周期之间调整。这些值适应 V_{CC} = 5 V。请参考其它 V_{CC}电压下的 RC 晶振频率的特性化数据。WDR,即看门狗复位指令对看门狗定时器进行复位。有 8 种不同的时钟周期可供选择,来决定在 2 个 WDR 指令之间的最大时间,这样作是为了避免看门狗定时器对 MCU 进行复位。若超过了复位时间而无其它的 WDR 指令,AT90S1200 进行复位,并靠复位向量执行。

为了防止意外禁止看门狗定时器，当看门狗被禁止时必须服从一个特定的关断顺序。

4.1.6 EEPROM 读/写访问

I/O 空间中可以访问 EEPROM 访问寄存器。AT90S1200 的 EEPROM 控制寄存器与其它型号不同，控制寄存器中不包含 EEMWE 位，也不必设置 EEMWE 位。

写入访问时间在 2.5～4 ms 之间，取决于 V_{CC}电压。自定时功能使得用户软件检测下一个字节何时被写入。若 V_{CC}低于一定的电平，EEPROM 降低电压被检测，防止了对 EEPROM 的写入。当 EEPROM 被读取或写入时，在下一个指令执行前，CPU 被中止达两个时钟周期。详情请参考第二章。

4.1.7 模拟比较器

模拟比较器对正极 PB0 引脚(AIN0)和负极 PB1 引脚(AIN1)之上的输入值进行比较。当 PB0 的电压高于 PB1 的电压时，模拟比较器输出 ACO 被设为 1。比较器的输出可以被设置，来触发模拟比较器的中断。用户可以选择比较器输出上升、下降，或触发器的中断触发。详情请参考第二章。

4.1.8 I/O 口

一、B 口特性

B 口为一个 8 位的双向 I/O 口。B 口分配有 3 个数据存储地址，分别为数据寄存器 PORTB($18)、数据方向寄存器 DDRB($17)和 B 口的输出引脚($16)。B 口的输入引脚地址为只读，而数据寄存器和数据方向寄存器为可读写。

所有的 B 口引脚均有单独的可选择拉高。B 口输出缓存可以吸收 20 mA 的电流以直接驱动 LED 显示。当 PB0～PB1 引脚被用作输入且被外部拉低时，若内部拉高被激活，这些引脚将成为电流源(I_{IL})。具有可选择功能的 B 口引脚如表 4.4 所示。

表 4.4 B 口引脚可选择功能

口引脚	第 二 功 能
PB0	AIN0 (模拟比较器正输入)
PB1	AIN1 (模拟比较器负输入)
PB5	MOSI (储存器下载数据输入线)
PB6	MOSO (储存器下载数据输出线)
PB7	SCK (串行时钟输入)

当引脚被用作可选择的功能时，DDR 和 B 口数据寄存器必须根据可供选择的功能说明来设置。

二、D 口特性

D 口占了 3 个数据存储器的地址，一个是数据寄存器 PORTD($12)、数据方向寄存器 DDRD ($11)和 D 口输入引脚 PIND ($10)。D 口的引脚地址为只读，而数据寄存器和数据方向寄存器可以读/写。

D 口有 7 个带内部上拉的双向 I/O 口，PD6～PD0，D 口的输出缓冲器可以吸入 20 mA，作为输入时，如果上拉被激活，则在外部被拉低时将输出电流。

D 口有第二功能的引脚如表 4.5 所示。

表 4.5 D 口引脚管可选择功能

口引脚	第 二 功 能
PD2	INT0 (外中断 0)
PD4	T0 (定时/计数器 0 的外输入)

有关存储器编程的内容请参考第二章。

4.2 AT90S2313 单片机

AT90S2313 是 AVR 增强性能 RISC 结构的低功耗 CMOS 技术八位微控制器，图 4.6 为其结构图。

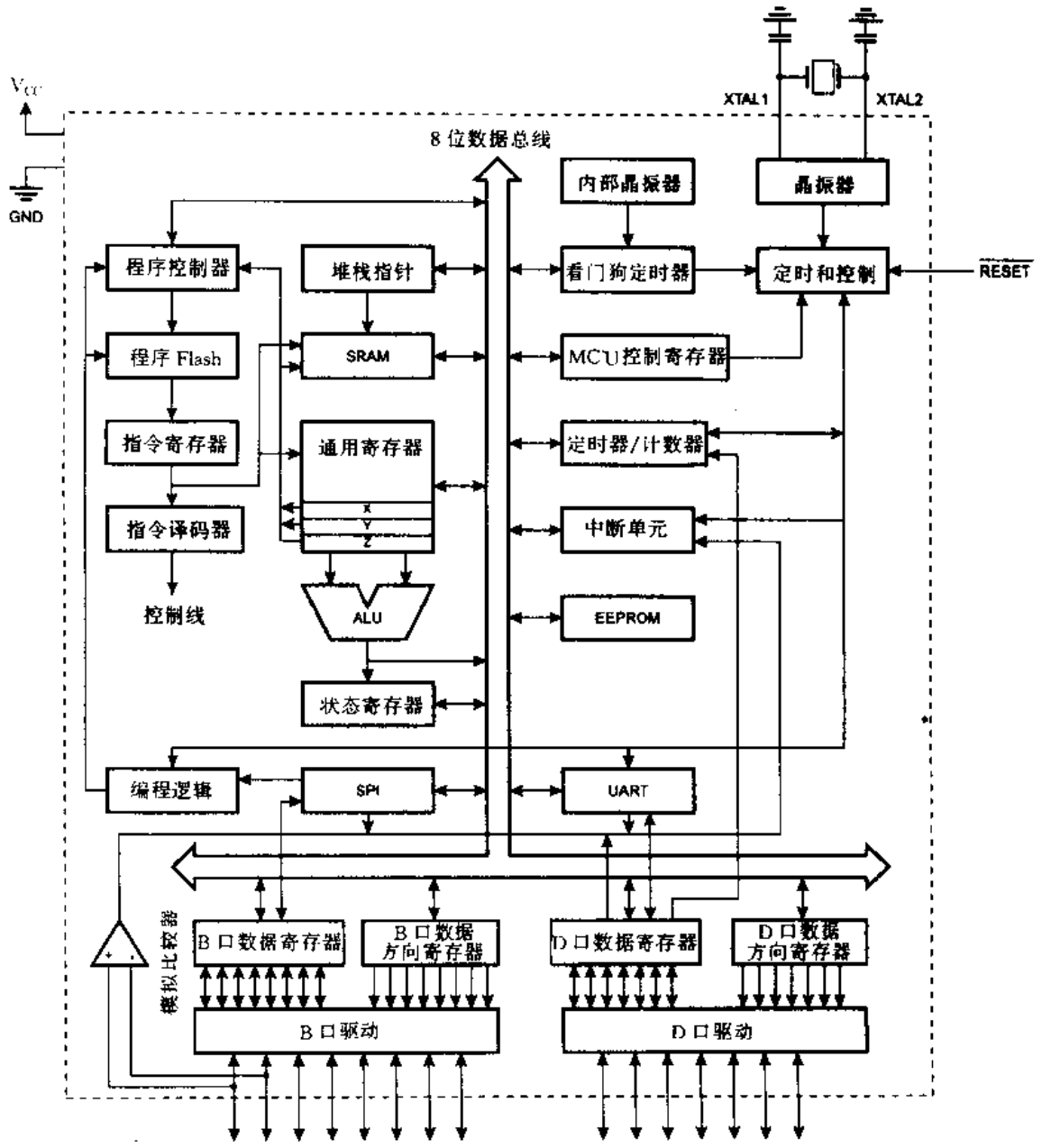


图 4.6 AT90S2313 结构图

该结构有效地支持高级语言,以及密集度极大的汇编器代码程序。该单片机具有以下特征:供电电压在 V_{CC}为 2.7 ~ 6 V 内可以全静态工作范围为 0~20 MHz;120 条功能强大的指令,大多数执行时间为单个时钟周期,指令周期时间为 50 ns @ 20 MHz,2K 字节可下载的 Flash 存储器(程序下载采用 SPI 串行接口,使用寿命为 1 000 次),128 字节 EEPROM(使用寿命为 10 万次);128 字节内部 RAM,15 条可编程 I/O 线,32 个 8 位通用寄存器;内部及外部中断,片内模拟比较器,带有一个 8 位可预分频的定时器/计数器,带有比较和捕获模式的 16 位的预分频的定时器/计数器,全双工 UART,可选择的 8 位、9 位或 10 位 PWM,外部和内部中断源,可编程的看门狗和片内晶振器;一个为下载程序而设计的串行口,以及 2 个可通过软件选择的电源保留模式。闲置模式停止 CPU 的工作,而寄存器、定时器/计数器、看门狗及中断系统继续工作。掉电模式保留寄存器的内容,但冻结晶振、终止芯片的其它功能,直至下一次外部中断或硬件复位。

4.2.1 引脚说明

AT90S2313 为 20 引脚 PDIP 和 SOIC 封装的单片机,图 4.7 为其引脚图。

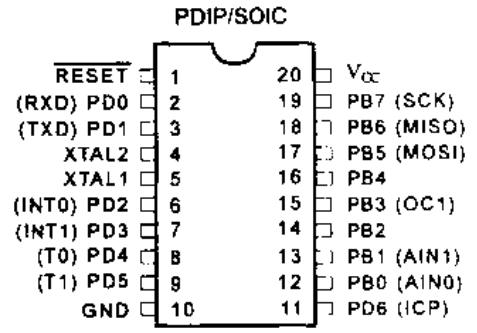


图 4.7 AT90S2313 的引脚图

一、引脚定义

V_{CC}(20 脚):供电引脚。

GND(10 脚):接地引脚。

B 口(PB7~PB0):B 口为一个 8 位双向 I/O 口,引脚可提供内部拉高(供每一位选用)。PB0 和 PB1 还单独作为片内模拟比较器的正极输入(AIN0)和负极输入(AIN1)。

D 口(PD6~PD0):D 口为带有 7 个有内部拉高 PD6~PD0 的双向 I/O 引脚。D 口输出缓存器吸收 20 mA 的电流。作为输入,若拉高被激活,被外部拉低的 D 口引脚吸收电流(I_{IL})。

$\overline{\text{RESET}}$(1 脚):复位输入。当晶振运行时,引脚上一个两周期的低电流可对器件进行复位。

XTAL1(5 脚):向晶振放大器的输入和向内部时钟操作电路的输入。

XTAL2(4 脚):从反转晶振放大器的输出。

二、晶振器

XTAL1 和 XTAL2 单独地作为反转放大器的输入和输出,该放大器可被设置为片内的晶振器。为了由外部源驱动器件,当 XTAL1 被驱动时,XTAL2 不能连接。详情可参考第二章。

4.2.2 AVR RISC 微控制器 CPU

AT90S2313 AVR RISC 微控制器向上与 AVR 增强 RISC 结构兼容。为 AT90S2313 MCU 而编写的程序,与 AVR 8 位 MCU(AT90SXXXX)全部系列产品的源代码和运行的时钟周期相兼容。

一、AVR 的结构

快速访问寄存器文件概念包括 32 个带有单一时钟周期访问时间的 8 位通用寄存器。

32 个寄存器中的 6 个为进行数据空间寻址可被用作 3 个 16 位间接地址寄存器指示器,

从而可以进行高效的地址计算。3 个寄存器中的一个还被用作地址指示器，完成常量表的查询功能。这些新加的功能寄存器为 16 位 X－寄存器、Y－寄存器、Z－寄存器。

ALU 支持寄存器之间，以及常数和寄存器之间的运算与逻辑功能。图 4.8 所示为 AT90S2313 AVR 的 RISC 结构。

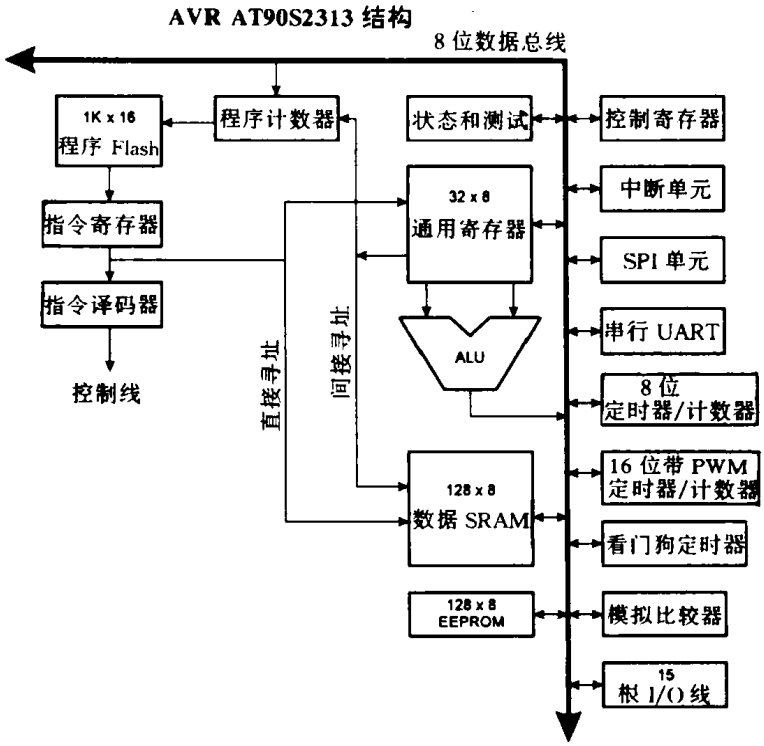


图 4.8　AT90S2313 AVR 的 RISC 结构

AVR 结构中的存储器空间均为线性和规律的存储器映射。图 4.9 为存储器映射图。

二、通用寄存器文件

图 4.10 所示为在 CPU 中，32 个通用寄存器文件的结构图。每个寄存器被分配给一个数据存储器地址，将其直接映射，如用户数据空间的前 32 个地址。

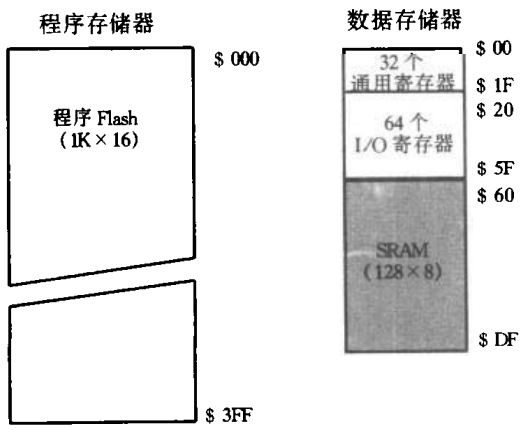


图 4.9　存储器映射

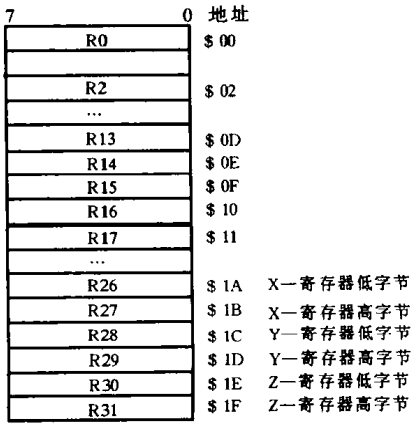


图 4.10　32 个通用寄器文件图

三、ALU 运算逻辑单元

高性能的 AVR 运算逻辑单元的操作与所有的 32 个通用寄存器保持直接的连接。在单一时钟周期内,ALU 是在寄存器文件中的寄存器之间执行操作。ALU 操作被分为 3 个主要的目的:算法、逻辑和位功能。

四、可下载的 Flash 程序存储器

AT90S2313 包括 2K 字节的片内可下载 Flash 存储器。由于所有指令为 16 位字或 32 位字,用于存储程序的 Flash 的结构为 1K×16。Flash 存储器的使用寿命最少为 1 000 次写/擦循环。AT90S2313 程序计数器宽为 10 位,以此来对 1 024 个程序存储器地址寻址。常量表必须被设定在 0~2K 的地址之间(请参考 LPM——装入程序存储器指令说明)。

五、EEPROM 数据存储器

AT90S2313 包括 128 字节的 EEPROM 存储器。它被组织为一个分开的数据空间,在其间单一的字节可被读写。EEPROM 的使用寿命为至少 100 000 次写/擦循环。

六、SRAM 数据存储器

图 4.11 说明 AT90S2313 的 SRAM 数据存储器组成。

224 个数据存储器地址对寄存器文件、I/O 存储器和数据 SRAM 寻址。前 96 个地址对寄存器文件 + I/O 存储器寻址,接下来的 128 个地址对数据 SRAM 寻址。

32 个通用工作寄存器、64 个 I/O 寄存器,以及 AT90S2313 中的 128 字节数据 SRAM 可通过所有这些地址模式被直接访问到。

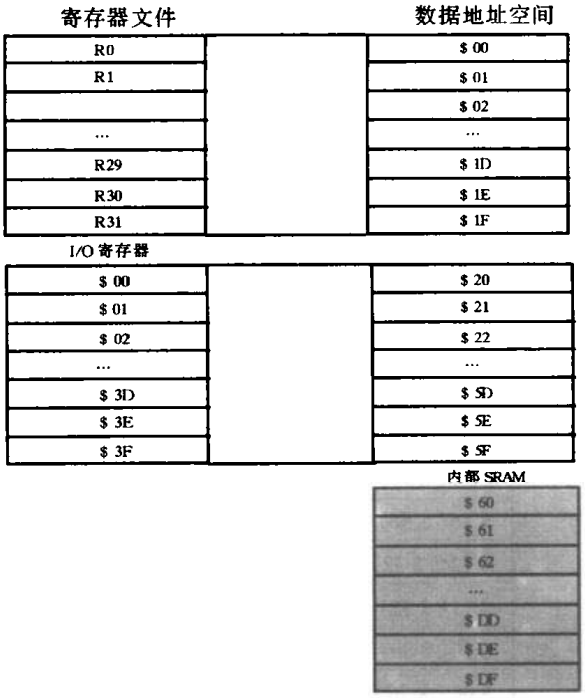


图 4.11 AT90S2313 的数据存储器

七、编程和数据寻址模式

AT90S2313 AVR 增强 RISC 微控制器支持对程序存储器(Flash)和数据存储器访问的功能强大且效率高的寻址模式,AT90S2313 结构所支持的寻址模式有单一寄存器直接(Rd)、两个寄存器直接(Rd 和 Rr)、I/O 直接、数据直接、带位移的数据间接、数据间接、带预减量的数据间接、带后增量的数据间接、使用 LPM 指令的常量地址、间接程序寻址、相关程序寻址等寻址模式。

八、存储器访问和指令执行时序

CPU 由系统时钟 Φ 驱动,直接由芯片的外部时钟晶振激活,没有使用内部时钟部分。AVR 单片机的 Harvard 结构和快速访问寄存器文件概念,能使得并行指令存取和指令执行。这种基本的流水线概念目的是为了获得高达每 1MIPS/MHz 的功效。有关指令执行和内部存储器访问的通用访问时序请参考第二章。

九、I/O 存储器

表 4.6 所示为 AT90S2313 的 I/O 空间定义。

表4.6 AT90S2313的I/O空间

十六进制地址	名 称	功 能
$3F($5F)	SREG	状态寄存器
$3D($5D)	SPL	堆栈指针低
$3B($5B)	GIMSK	通用中断屏蔽寄存器
$3A($5A)	GIFR	通用中断标志寄存器
$39($59)	TIMSK	定时器/计数器中断屏蔽寄存器
$38($58)	TIFR	定时器/计数器中断标志寄存器
$35($55)	MCUCR	MCU通用控制寄存器
$33($53)	TCCR0	定时器/计数器0控制寄存器
$32($52)	TCNT0	定时器/计数器0(8位)
$2F($4F)	TCCR1A	定时器/计数器1控制寄存器A
$2E($4E)	TCNT1B	定时器/计数器1控制寄存器B
$2D($4D)	TCNT1H	定时器/计数器1高字节
$2C($4C)	TCNT1L	定时器/计数器1低字节
$2B($4B)	OCR1H	定时器/计数器1输出比较寄存器高字节
$2A($4A)	OCR1L	定时器/计数器1输出比较寄存器低字节
$25($45)	ICR1H	T/C1输入捕获寄存器高字节
$24($44)	ICR1L	T/C1输入捕获寄存器低字节
$21($41)	WDTCR	看门狗定时控制寄存器
$1E($3E)	EEAR	EEPROM地址寄存器
$1D($3D)	EEDR	EEPROM数据寄存器
$1C($3C)	EECR	EEPROM控制寄存器
$18($38)	PORTB	B口数据寄存器
$17($37)	DDRB	B口数据方向寄存器
$16($36)	PINB	B口输入脚
$12($32)	PORTD	D口数据寄存器
$11($31)	DDRD	D口数据方向寄存器
$10($30)	PIND	D口输入脚
$0C($2C)	UDR	UART I/O数据寄存器
$0B($2B)	USR	UART状态寄存器
$0A($2A)	UCR	UART控制寄存器
$09($29)	UBRR	UART波特率寄存器
$08($28)	ACSR	模拟比较控制和状态寄存器

注意:保留的和未用到的地址未列。

为使用I/O特定的命令时,必须使用IN、OUT、SBIS和I/O地址的$00～$3F。当写址的I/O寄存器为SRAM时,必须向该地址加入$20。

十、复位和中断处理

1.复位源

AT90S2313提供10种不同的中断源。这些中断和与之分开的复位向址均在程序存储器空间中各自带有分开的程序向量。所有中断均分配给单独的使能位,这些使能位须与状态寄存器中的I位一起被设为1,以便能执行中断。程序存储器空间中最下面的地址被自动地定

义为复位和中断向量，如表 4.7 所列，表中还列出了不同中断的优先级，地址越低，优先级越高。RESET 有最高的优先级，以下依次为 INT0……。

表 4.7 复位和中断向量

向量号	程序地址	源	中断定义
1	$000	RESET	硬件脚和看门狗复位
2	$001	INT0	外部中断请求 0
3	$002	INT1	外部中断请求 1
4	$003	TIMER1 CAPT1	定时器/计数器 1 捕获事件
5	$004	TIMER1 COMP1	定时器/计数器 1 比较匹配
6	$005	TIMER1 OVF1	定时器/计数器 1 溢出
7	$006	TIMER0 OVF0	定时器/计数器 0 溢出
8	$007	UART, RX	UART, RX 完成
9	$008	UART, UDRE	UART 数据寄存器空
10	$009	UART, TX	UART, TX 完成
11	$00A	ANA_COMP	模拟比较器

AT90S2313 有与 AT90S1200 相同的 3 个复位源，且复位过程也与之相同，见 4.1.3 节。

2. 中断处理

AT90S2313 有 2 个 8 位中断屏蔽控制寄存器，即 GIMSK 通用中断屏蔽寄存器和 TIMSK 定时器/计数器中断寄存器。

十一、休眠模式

休眠模式 AT90S2313 同于 AT90S1200，见 4.1.3 节。

4.2.3 定时器/计数器

AT90S2313 有 2 种通用定时器/计数器，即一个 8 位的 T/C、一个 16 位 T/C。定时器/计数器从 10 位的预定比例定时器获取独立的预定比例区域。2 个定时器/计数器可被用作带内部时基的定时器，或被用作带可触发计数的外部引脚连接的计数器。

一、定时器/计数器预定标

通用定时器/计数器的预定标器请参考第二章。

二、8 位定时器/计数器 0

8 位的定时器/计数器 0 可从 CK、预定比例 CK，或外部引脚来选择时钟源。另外，它还可以像定时器/计数器 0 的控制寄存器 TCCR0 规定中所描述的那样而停止。

溢出状态比例标志位可在定时器/计数器中断比例标志位寄存器 TIFR 中。定时器/计数器 0 的中断使能/禁止设置在定时器/计数器中断的控制屏蔽寄存器 TIMSK 中。

三、16 位定时器/计数器 1

16 位的定时器/计数器 1 可以从 CK、预定比例 CK，或外部引脚中选择时钟源。另外，它还可以像在定时器/计数器 1 控制寄存器 TCCR1A 中描述的那样被停止。在定时器/计数器中断标志寄存器 TIFR 中可以找到不同的标志(溢出、比较匹配以及捕获事件)和控制信号。对定时器/计数器 1 的使能/禁止设置可以在定时器/计数器中断屏蔽寄存器 TIMSK 中找到。

定时器/计数器可被用作带调制器的 8 位、9 位或 10 位脉冲。在此模式下，定时器和 OCR1 寄存器具有集中脉冲双抗误操作独立的 PWM 能力。

定时器/计数器的输入捕获功能提供了对定时器/计数器 1 向输入捕获寄存器 IOR1 内容

的捕获。该捕获操作由在输入捕获引脚 ICP 上的外部事件激活。实际的捕获事件设置由定时器/计数器 1 的控制寄存器 TCCR1 来定义。

4.2.4 看门狗定时器

看门狗定时器性能 AT90S2313 同于 AT90S1200, 见 4.1.5 节。

4.2.5 EEPROM 读/写访问

I/O 空间中可以访问 EEPROM 访问寄存器。写入访问时间在 2.5～4 ms 之间。取决于 V_{CC}电压。自定时功能使得用户软件检测下一个字节何时被写入。若 V_{CC}低于一定的电平, EEPROM 降低电压被检测, 防止了对 EEPROM 的写入。当 EEPROM 被读取或写入时, 在下一个指令执行前, CPU 被中止达两个时钟周期。

4.2.6 通用串行接口 UART

AT90S2313 带有一个全双工的通用串行异步收发器(UART), 主要特征如下:

- 波特率发生器可以生成任何波特率。
- 在 XTAL 低频率下有高的波特率。
- 8 位和 9 位数据。
- 噪声滤波。
- 超越误差的检测。
- 帧错误检测。
- 错误启始位的检测。
- 三个独立的中断, TX 完成、TX 数据寄存器为空、RX 完成。

一、数据传送

数据传送通过把被传送的数据写入 UART I/O 寄存器 UDR 来初始化。当 UCR 中的 TXEN 位被设置为 1 时使能 UART 的传送。

二、数据接收

接收器前端的逻辑以 16 倍波特率采样 RXD 引脚的信号, 当线路闲置时, 一个逻辑 0 的采样将被转换为启始位的下降沿, 并且启动位的探测序列被初始化。让采样 1 指示出第一个 0 采样, 跟随 1 到 0 的转换之后, 接收器在第 8、9 和 10 个采样点采样 RXD 引脚。如果三个采样中两个或两个以上是逻辑 1, 则起始位是噪声尖峰而被拒绝, 接收器继续探测下一个 1 到 0 的转换。

如果一个有效的起始位被发现, 就开始起始位之后的数据位的采样。这些位也在第 8、9 和 10 处采样, 3 取 2 作为该位的逻辑值, 在采样的同时这些位被移入传送移位寄存器。

三、UART 控制

(1) UART I/O 数据寄存器——UDR: UDR 寄存器是两个物理分离的寄存器分享相同的 I/O 地址。当写入寄存器时, UART 的发送数据寄存器被写入; 当读 UDR 时, 读的是 UDR 接收寄存器。

(2) UART 状态寄存器——USR: USR 寄存器是一个只读的寄存器, 提供 UART 的状态信息。

(3) UART 控制寄存器——UCR: UCR 寄存器是一个可读写的控制 UART 的寄存器。

(4) 波特率发生器: 波特率发生器是依据以下等式的分频器来产生波特率:

$$BAUD = FCK/[16(UBRR + 1)]$$

其中:BAUD 为 Band－Rate 波特率; FCK 为晶振频率; UBRR 为 UART 波特率寄存器的值, UBRR 为 0～255。

(5) 波特率寄存器——UBRR:UBRR 是 8 位可以读/写的寄存器,用来确定波特率。

4.2.7 模拟比较器

模拟比较器的性能 AT90S2313 同于 AT90S1200,见 4.1.7 节。

4.2.8 I/O 口

一、B 口特性

B 口为一个 8 位的双向 I/O 口。B 口分配有 3 个数据存储地址,分别为数据寄存器 PORTB $18($38)、数据方向寄存器 DDRB $17($38)和 B 口的输出引脚 $16($36)。B 口的输入引脚地址为只读,而数据寄存器和数据方向寄存器为可读写。

所有的 B 口引脚均有单独的可选择拉高。B 口输出缓存可以吸收 20 mA 的电流以直接驱动 LED 显示。当 PB0～PB7 引脚被用作输入且被外部拉低时,若内部拉高被激活,这些引脚将成为电流源(I_{IL})。具有可选择功能的 B 口引脚如表 4.8 所示。

表 4.8 B 口引脚可选择功能

口引脚	第 二 功 能
PB0	AIN0(模拟比较器正输入)
PB1	AIN1(模拟比较器负输入)
PB3	OC1(定时器/计数器 1 输出比较匹配输出)
PB5	MOSI(储存器下载数据输入线)
PB6	MOSO(储存器下载数据输出线)
PB7	SCK(串行时钟输入)

当引脚被用作可选择的功能时,DDRB 和 PORTB 寄存器必须根据可供选择的功能说明来设置。

二、D 口特性

D 口占了 3 个数据存储器的地址,一个是数据寄存器 PORTD $12($32)、数据方向寄存器 DDRD $11($31)和 D 口输入引脚 PIND $10($30)。D 口的引脚地址为只读,而数据寄存器和数据方向寄存器可以读/写。

D 口有 7 个带内部上拉的双向 I/O 口。PD6～PD0。D 口的输出缓冲器可以吸入 20 mA。作为输入时,如果上拉被激活,则在外部被拉低时将输出电流。同时某些端口 D 的引脚有第二功能如表 4.9 所示。

表 4.9 D 口引脚可选择功能

口引脚	第 二 功 能	口引脚	第 二 功 能
PD0	RXD(UART 输入线)	PD4	T0(定时器/计数器 0 外部输入)
PD1	TXD(UART 输出线)	PD5	T1(定时器/计数器 1 外部输入)
PD2	INT0(外部中断 0 输入)	PD6	ICP(定时器/计数器 1 输入捕获脚)
PD3	INT1(外部中断 1 输入)		

当 D 口的引脚被用于第二功能时,DDRD 和 PORTD 寄存器必须按照第二功能来设置。

D 口的输入引脚地址 PIND 不是一个寄存器,该地址允许对 D 口的每一个引脚进行存取。当读 PORTD 时,读到的是 PORTD 的数据锁存器;当读 PIND 时,引脚上的逻辑值被读入。

有关存储器编程的内容请参考第二章。

4.3 AT90S4414 单片机

AT90S4414 是 AVR 增强性能 RISC 结构的低功耗 CMOS 技术八位微控制器，图4.12 为其结构图。

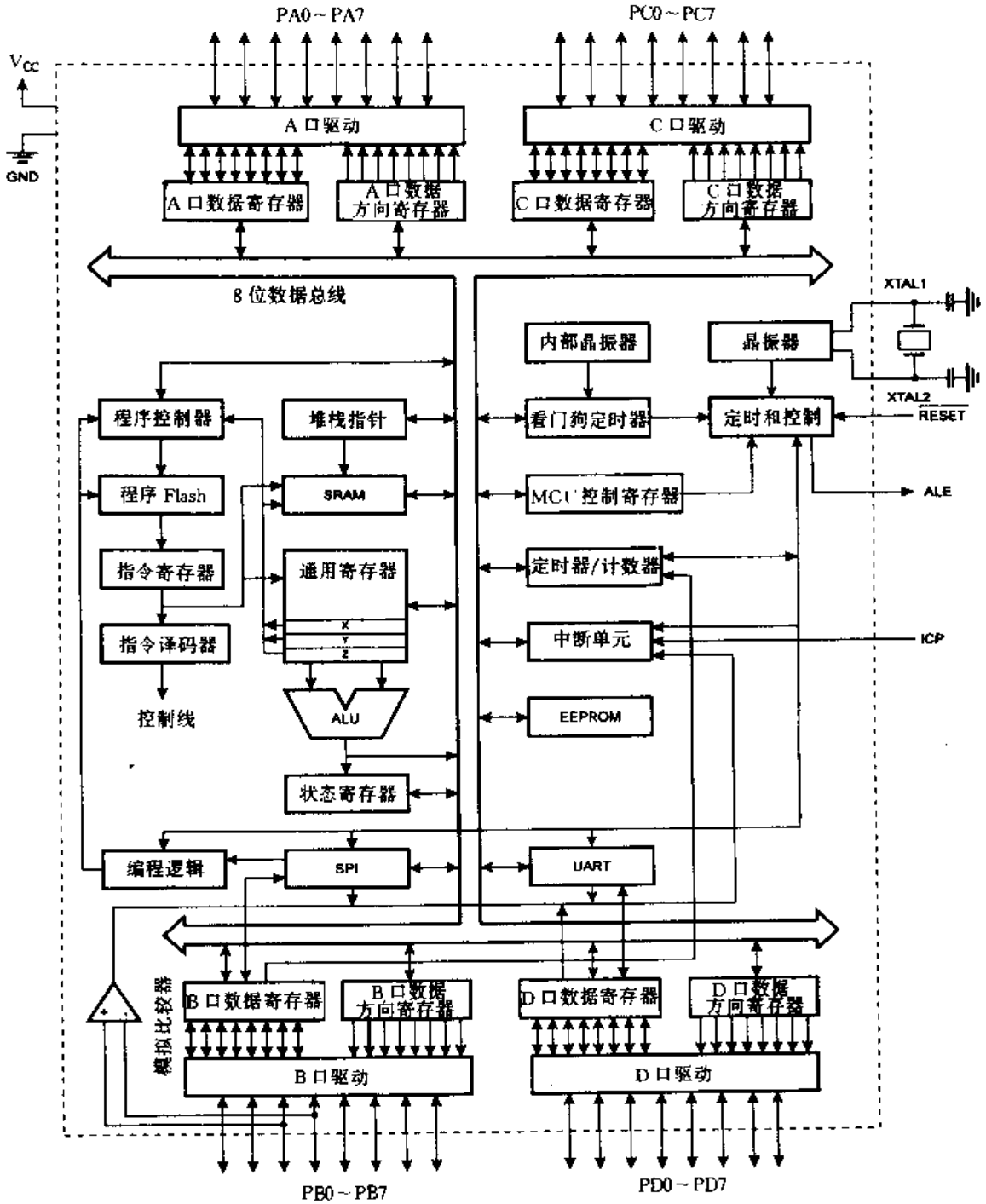


图 4.12 AT90S4414 结构图

该结构有效地支持高级语言,以及密集度极大的汇编器代码程序。该单片机具有以下特征:供电电压在 V_{CC}为 2.7 ~6 V 内可以全静态工作范围为 0~20 MHz;120 条功能强大的指令,大多数执行时间为单个时钟周期,指令周期时间为 50 ns @ 20 MHz,4K 字节可下载的 Flash 存储器(程序下载采用 SPI 串行接口,使用寿命为 1 000 次),256 字节 EEPROM(使用寿命为 10 万次);256 字节内部 RAM ,32 条可编程 I/O 线,32 个 8 位通用寄存器;带有一个 8 位可预分频的定时器/计数器,带有比较和捕获模式的 16 位的预分频的定时器/计数器,可编程的串行口 UART、双口 PWM、外部和内部中断源;可编程的看门狗和片内晶振器,片内模拟比较器;一个为下载程序而设计的 SPI 串行口,以及 2 个可通过软件选择的电源保留模式。闲置模式停止 CPU 的工作,而寄存器、定时器/计数器、看门狗及中断系统继续工作。掉电模式保留寄存器的内容,但冻结晶振、终止芯片的其它功能,直至下一次外部中断或硬件复位。

4.3.1 引脚说明

AT90S4414 为 40 引脚 PDIP 和 PLCC 封装的单片机,图 4.13 为其引脚图。

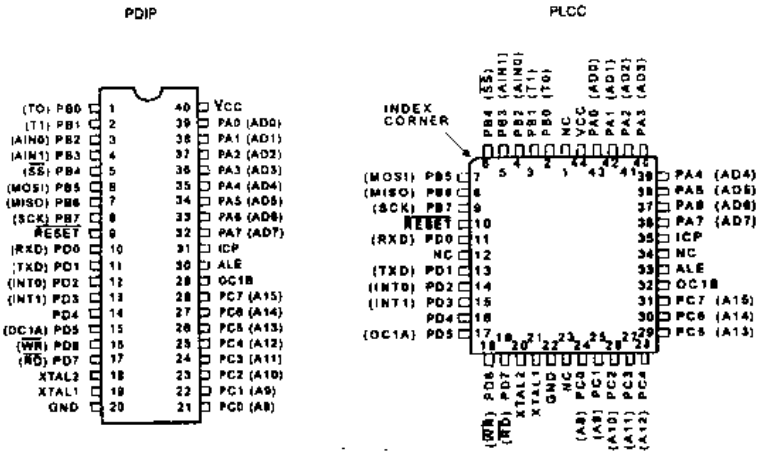


图 4.13 AT90S4414 的引脚图

一、引脚定义

V_{CC}(40 脚):供电引脚。

GND(20 脚):接地引脚。

A 口(PA7~PA0):A 口为一个 8 位双向 I/O 口,引脚可提供内部拉高(供每一位选用)。PORTA 的输出缓冲器可以吸收 20 mA 的电流, 因而能直接驱动 LED 显示器。当 PA0~PA7 被用于输入且内部上拉被激活时,如果外部被拉低,则会输出电流。当使用外部 SRAM 时 PORTA 作为复用的地址/数据和输入/输出口。

B 口(PB7~PB0):B 口为一个 8 位双向 I/O 口,引脚可提供内部拉高。当 B 口被用于输入且内部上拉被激活时,如果外部被拉低,则会输出电流。B 口还提供一些特殊功能。

C 口(PC7~PC0):C 口为一个 8 位双向 I/O 口,引脚可提供内部拉高。C 口的输出缓冲器可以吸收 20 mA 的电流。当 PC0~PC6 被用于输入且内部上拉被激活时,如果外部被拉低,则会输出电流。当使用外部 SRAM 时 C 口作为地址输出。

D 口 (PD7～PD0):D 口为带有 7 个有内部拉高的双向 I/O 引脚。D 口输出缓存器吸收 20mA 的电流。作为输入,若拉高被激活,被外部拉低的 D 口引脚输出电流(I_{IL})。D 口还有一些特殊功能。

$\overline{\text{RESET}}$(9 脚):复位输入。当晶振运行时,引脚上一个两周期的低电流可对器件进行复位。

XTAL1(19 脚):向晶振放大器的输入和向内部时钟操作电路的输入。

XTAL2(18 脚):从反转晶振放大器的输出。

ICP:ICP 是定时/计数器 1 的输入捕获功能的输入引脚。

OC1B:OC1B 是定时/计数器 1 的输出比较功能 B 的输出引脚。

ALE:ALE 是使用外部存储器时的地址锁存使能。ALE 控制门被用于在第一个访问周期中将低位地址锁存到地址锁存器中,而 PA0～PA7 在第二个访问周期中被用作数据。

二、晶振器

XTAL1 和 XTAL2 单独地作为反转放大器的输入和输出,该放大器可被设置为片内的晶振器。为了由外部源驱动器件,当 XTAL1 被驱动时,XTAL2 不能连接。详情可参考第二章。

4.3.2 AVR RISC 微控制器 CPU

AT90S4414 AVR RISC 微控制器向上与 AVR 增强 RISC 结构兼容。为 AT90S4414 MCU 而编写的程序,与 AVR 8 位 MCU(AT90SXXXX)全部系列产品的源代码和运行的时钟周期相兼容。

一、AVR 的结构

快速访问寄存器文件概念包括 32 个带有单一时钟周期访问时间的 8 位通用寄存器。这意味着在一个单一时钟周期内,执行一个 ALU(运算逻辑单元)。两个操作数为寄存器文件的输出,操作数被执行,运行结果被存储回寄存器文件。这些均在一个时钟周期内完成。

32 个寄存器中的 6 个为进行数据空间写址可被用作 3 个 16 位间接地址寄存器指针,从而可以进行高效的地址计算。3 个寄存器中的一个还被用作地址指示器,完成常量表的查询功能。这些新加的功能寄存器为 16 位 X－寄存器、Y－寄存器、Z－寄存器。

ALU 支持寄存器之间,以及常数和寄存器之间的运算与逻辑功能。图 4.14 所示为 AT90S4414 AVR 的 RISC 结构。

除了寄存器操作功能之外,常规存储器写址模式还可用在寄存器文件上。寄存器文件由最低的 32 个数据空间地址($00～ $1F)分派而使能,从而使得即使它们为一般的存储地址,也可被访问到。I/O 存储器空间包含 64 个作为 CPU 外围功能的地址,为控制寄存器、定时器/计数器,A/D 转换器,及其它 I/O 功能。I/O 存储器可被直接访问,或作为寄存器文件之后的数据空间地址 $20～ $5F。

256 字节的数据 SRAM＋寄存器文件和 I/O 寄存器,可以通过 AVR 结构中的 5 种不同写址模式被很容易地访问。

AVR 结构中的存储器空间均为线性和 规律的存储器映射。图 4.15 为存储器映射图。

二、通用寄存器文件

在 CPU 中,AT90S4414 的 32 个通用寄存器文件的结构图与图 4.10 所示的相同。每个寄存器被分配给一个数据存储器地址,将其直接映射,如用户数据空间的前 32 个地址。虽然寄存器文件并未被物理地作为 SRAM 地址而提供出来,但由于寄存器 X、Y 和 Z 可被设置,来

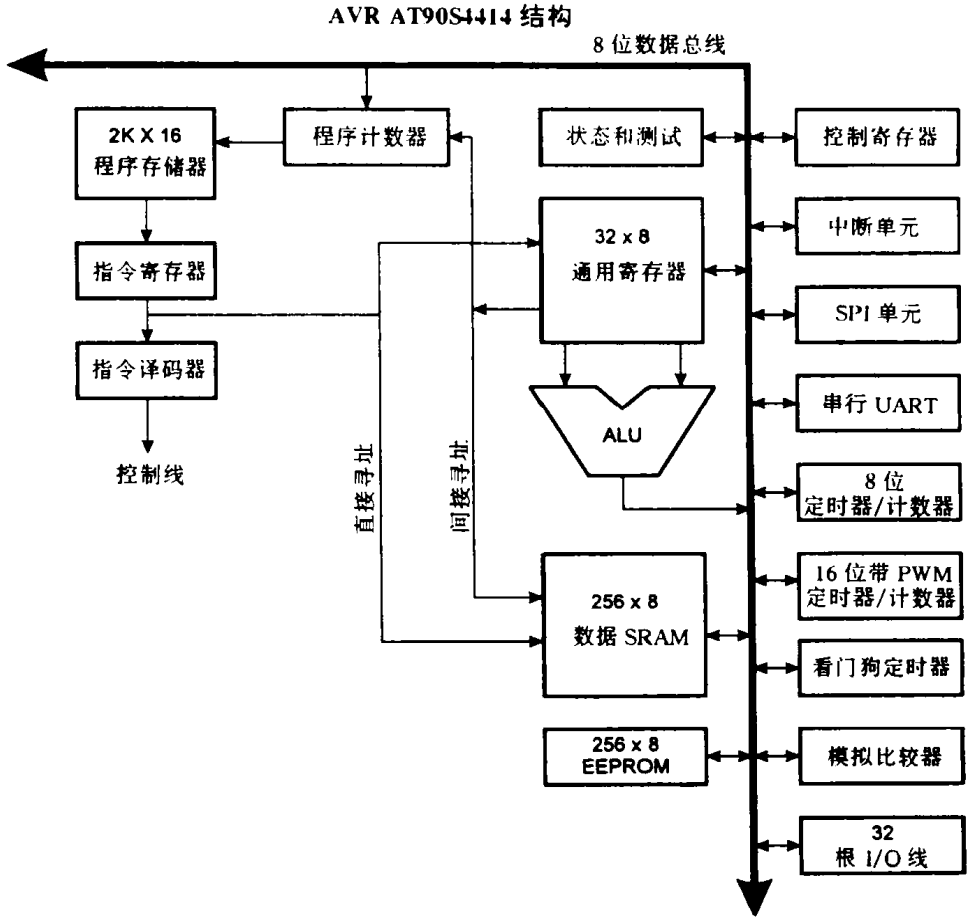


图 4.14 AT90S4414 AVR 的 RISC 结构

对文件中的寄存器做索引。这种存储器的组成结构在访问寄存器时提供了极大的灵活性。

三、ALU 运算逻辑单元

高性能的 AVR 运算逻辑单元的操作与所有的 32 个通用寄存器保持直接的连接。在单一时钟周期内,ALU 是在寄存器文件中的寄存器之间执行操作。ALU 操作被分为 3 个主要的目的:算法、逻辑和位功能。AVR 产品家族中一些微控制器的 ALU 运算部件中有硬件乘法器。

四、可下载的 Flash 程序存储器

AT90S4414 包括 4K 字节的片内可下载 Flash 存储器。由于所有指令为 16 位字或 32 位字,用于存储程序的 Flash 的结构为 2K×16。Flash 存储器的使用寿命最少为 1 000 次写/擦循环。AT90S4414 程序设计器宽为 11 位,以此来对 2 048 个程序存储器地址写址。

常量表必须被设定在 0~2K 的地址之间(请参考 LPM——装入程序存储器指令说明)。

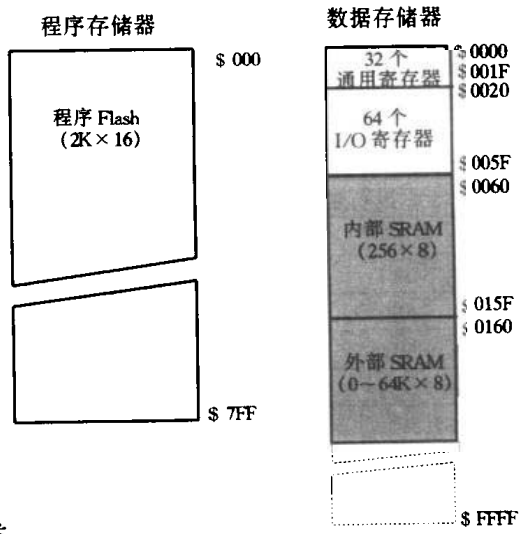


图 4.15 存储器映射

五、EEPROM 数据存储器

AT90S4414 包括 256 字节的 EEPROM 存储器。它被组织为一个分开的数据空间，在其间单一的字节可被读写。EEPROM 的使用寿命为至少 100 000 次写/擦循环。

六、SRAM 数据存储器

图 4.16 说明 AT90S4414 的 SRAM 数据存储器组成。

低端 352 个数据存储器地址编址为寄存器文件、I/O 存储器、和数据 SRAM。前 96 个地址编址为寄存器文件 + I/O 存储器，接下来的 256 个地址对数据 SRAM 寻址。可选的外部数据 SRAM 可以放置在同一个 SRAM 空间中，该 SRAM 将占据内部 SRAM 之后的地址直到 64K - 1，这由 SRAM 的大小来定。

数据存储器的 5 个不同写址模式包括：直接、带位移的间接、间接、带预减量的间接和带后增量的间接。在寄存器文件中，寄存器 R26 到 R31 具有间接写址指示器寄存器的特性。

32 个通用工作寄存器、64 个 I/O 寄存器，以及 AT90S4414 中的 256 字节数据 SRAM，可通过所有这些地址模式被直接访问到。

图 4.16 AT90S4414 的数据存储器

七、编程和数据寻址模式

编程和数据寻址模式，AT90S4414 同于 AT90S2313，见 4.2.2 节。

八、存储器访问和指令执行时序

存储器访问和指令执行时序，AT90S4414 同于 AT90S2313，见 4.2.2 节。

九、I/O 存储器

表 4.10 所示为 AT90S4414 的 I/O 空间定义。

表 4.10 AT90S4414 的 I/O 空间

十六进制地址	名 称	功 能
$ 3F($ 5F)	SREG	状态寄存器
$ 3E($ 5E)	SPH	堆栈指针高
$ 3D($ 5D)	SPL	堆栈指针低
$ 3B($ 5B)	GIMSK	通用中断屏蔽寄存器
$ 3A($ 5A)	GIFR	通用中断标志寄存器
$ 39($ 59)	TIMSK	定时器/计数器中断屏蔽寄存器
$ 38($ 58)	TIFR	定时器/计数器中断标志寄存器
$ 35($ 55)	MCUCR	MCU 通用控制寄存器
$ 33($ 53)	TCCR0	定时器/计数器 0 控制寄存器
$ 32($ 52)	TCNT0	定时器/计数器 0(8 位)

表 4.10 续

十六进制地址	名　称	功　能
$2F($4F)	TCCR1A	定时器/计数器 1 控制寄存器 A
$2E($4E)	TCNT1B	定时器/计数器 1 控制寄存器 B
$2D($4D)	TCNT1H	定时器/计数器 1 高字节
$2C($4C)	TCNT1L	定时器/计数器 1 低字节
$2B($4B)	OCR1AH	定时器/计数器 1 输出比较寄存器 A 高字节
$2A($4A)	OCR1AL	定时器/计数器 1 输出比较寄存器 A 低字节
$29($49)	OCR1BH	定时器/计数器 1 输出比较寄存器 B 高字节
$28($48)	OCR1BL	定时器/计数器 1 输出比较寄存器 B 低字节
$25($45)	ICR1H	T/C1 输入捕获寄存器高字节
$24($44)	ICR1L	T/C1 输入捕获寄存器低字节
$21($41)	WDTCR	看门狗定时控制寄存器
$1E($3E)	EEAR	EEPROM 地址寄存器
$1D($3D)	EEDR	EEPROM 数据寄存器
$1C($3C)	EECR	EEPROM 控制寄存器
$1B($3B)	PORTA	A 口数据寄存器
$1A($3A)	DDRA	A 口数据方向寄存器
$19($39)	PINA	A 口输入脚
$18($38)	PORTB	B 口数据寄存器
$17($37)	DDRB	B 口数据方向寄存器
$16($36)	PINB	B 口输入脚
$15($35)	PORTC	C 口数据寄存器
$14($34)	DDRC	C 口数据方向寄存器
$13 ($33)	PINC	C 口输入脚
$12 ($32)	PORTD	D 口数据寄存器
$11($31)	DDRD	D 口数据方向寄存器
$10($30)	PIND	D 口输入脚
$0F($2F)	SPDR	SPI I/O 数据寄存器
$0E($2E)	SPSR	SPI 状态寄存器
$0D($2D)	SPCR	SPI 控制寄存器
$0C($2C)	UDR	UART I/O 数据寄存器
$0B($2B)	USR	UART 状态寄存器
$0A($2A)	UCR	UART 控制寄存器
$09($29)	UBRR	UART 波特率寄存器
$08($28)	ACSR	模拟比较控制和状态寄存器

注意:保留的和未用到的地址未列。

十、复位和中断处理

1.复位源

AT90S4414 提供 12 种不同的中断源。这些中断和与之分开的复位向量均在程序存储器空间中各自带有分开的程序向量。所有中断均分配给单独的使能位,这些使能位须与状态寄存器中的 I 位一起被设为 1,以便能执行中断。

程序存储器空间中最下面的地址被自动地定义为复位和中断向量,如表 4.11 所列。表中还列出了不同中断的优先级,地址越低,优先级越高。RESET 有最高的优先级,以下依次为 INT0……。

表 4.11 复位和中断向量

向量号	程序地址	源	中断定义
1	$000	RESET	硬件脚和看门狗复位
2	$001	INT0	外部中断请求 0
3	$002	INT1	外部中断请求 1
4	$003	TIMER1 CAPT	定时器/计数器 1 捕获事件
5	$004	TIMER1 COMPA	定时器/计数器 1 比较匹配 A
6	$005	TIMER1 COMPB	定时器/计数器 1 比较匹配 B
7	$006	TIMER1 OVF	定时器/计数器 1 溢出
8	$007	TIMER0 OVF	定时器/计数器 0 溢出
9	$008	SPI, STC	串行传送完成
10	$009	UART, RX	UART, RX 完成
11	$00A	UART, UDRE	UART 数据寄存器空
12	$00B	UART, TX	UART, TX 完成
13	$00C	ANA_COMP	模拟比较器

AT90S4414 有与 AT90S1200 相同的 3 个复位源,且复位过程也与之相同,见 4.1.3 节。

2. 中断处理

AT90S4414 有 2 个 8 位中断屏蔽控制寄存器,即 GIMSK 通用中断屏蔽寄存器和 TIMSK 定时器/计数器中断寄存器。

十一、休眠模式

休眠模式 AT90S4414 同于 AT90S1200,见 4.1.3 节。

4.3.3 定时器/计数器

AT90S4414 有 2 种通用定时器/计数器,即一个 8 位的 T/C 和一个 16 位 T/C。定时器/计数器从 10 位的预定比例定时器获取独立的预定比例区域。2 个定时器/计数器可被用作带内部时基的定时器,或被用作带可触发计数的外部引脚连接的计数器。

一、定时器/计数器预定比例器

通用定时器/计数器的预定标器请参考第二章。

二、8 位定时器/计数器 0

8 位的定时器/计数器 0 可从 CK、预定比例 CK,或外部引脚来选择时钟源。另外,它还可以像定时器/计数器 0 的控制寄存器 TCCR0 规格中所描述的那样而停止。

溢出状态比例标志位可在定时器/计数器中断比例标志位寄存器 TCCR0 中。定时器/计数器 0 的中断使能/禁止设置在定时器/计数器中断的控制屏蔽寄存器 TIMSK 中。

三、16 位定时器/计数器 1

16 位的定时器/计数器 1 可以从 CK、预定比例 CK,或外部引脚中选择时钟源。另外,它还可以像在定时器/计数器 1 控制寄存器 TCCR1A、TCCR1B 中描述的那样被停止。在定时器/计数器控制寄存器 TCCR1A、TCCR1B 中可以找到不同的标志(溢出、比较匹配以及捕获事件)和控制信号。对定时器/计数器 1 的使能/禁止设置可以在定时器/计数器中断屏蔽寄存器

TIMSK 中找到。

4.3.4 看门狗定时器

看门狗定时器性能 AT90S4414 同于 AT90S1200，见 4.1.5 节。

4.3.5 EEPROM 读/写访问

EEPROM 读/写访问 AT90S4414 同于 AT90S2313，见 4.2.5 节。

4.3.6 串行外设接口 SPI

串行外设接口(SPI)允许在 AT90S4414 和外设或几个 AT90S1200 之间高速同步数据传送，AT90S4414 SPI 的特征如下：

- 全双工、3 线同步数据传送。
- 主从操作。
- 5 Mb/s 的位传送频率。
- LSB 或 MSB 在先数据传输。
- 四种可编程的位速率。
- 传送停止的中断标志。
- 写冲突标志保护。
- 从闲置模式下唤醒(仅在从模式下)。

4.3.7 通用串行接口 UART

通用串行接口 UART，AT90S4414 同于 AT90S2313，见 4.2.6 节。

4.3.8 模拟比较器

模拟比较器的性能 AT90S4414 同于 AT90S1200，见 4.1.7 节。

4.3.9 I/O 口

一、A 口特性

A 口为一个 8 位的双向 I/O 口。A 口分配有 3 个数据存储地址，分别为数据寄存器 PORTA $1B($3B)、数据方向寄存器 DDRA $1A($3A)和 B 口的输出引脚 $19($39)。A 口的输入引脚地址为只读，而数据寄存器和数据方向寄存器为可读写。

A 口引脚具有与可选的外部数据 SRAM 有关的第二功能，A 口在访问外部数据存储器时可以配置为复用的低位地址/数据线，在该模式下，A 口有内部的上拉。

当通过 MCU 控制寄存器——MCUCR 的 SRE——外部 SRAM 使能位把 A 口设置为第二功能，更改的设置会覆盖数据方向存储器。

A 口的输入引脚地址 PINA 不是一个寄存器，该地址允许对 A 口的每一个引脚进行存取。当读 PORTA 时，读到的是 A 口的数据锁存器；当读 PINA 时，引脚上的逻辑值被读入。

二、B 口特性

B 口为一个 8 位的双向 I/O 口。B 口分配有 3 个数据存储地址，分别为数据寄存器 PORTB $18($38)、数据方向寄存器 DDRB $17($37)和 B 口的输出引脚 $16($36)。B 口的输入引脚地址为只读，而数据寄存器和数据方向寄存器为可读写。具有可选择功能的 B 口引脚如表 4.12 所示。

表 4.12 B 口引脚选择功能

口引脚	第二功能
PB0	T0 (定时器/计数器 0 外部计数器输入)
PB1	T1 (定时器/计数器 1 外部计数器输入)
PB2	AIN0 (模拟比较器正输入)
PB3	AIN1 (模拟比较器负输入)
PB4	$\overline{SS}$ (SPI 从选择输入)
PB5	MOSI (SPI 总线主输出/从输入)
PB6	MOSO (SPI 总线主输出/从输入)
PB7	SCK (SPI 总线串行时钟)

当引脚被用作可选择的功能时,DDRB 和 PORTB 寄存器必须根据可供选择的功能说明来设置。

三、C 口特性

C 口为一个 8 位的双向 I/O 口。C 口分配有 3 个数据存储地址,分别为数据寄存器 PORTC $15($35)、数据方向寄存器 DDRC $14($34)和 C 口的输出引脚 $13($33)。C 口的输入引脚地址为只读,而数据寄存器和数据方向寄存器为可读写。

C 口引脚具有与可选的外部数据 SRAM 有关的第二功能。C 口在访问外部数据存储器时可以配置为复用的高位地址线,在该模式下,C 口输出 1 时使用内部的上拉。

当通过 MCU 控制寄存器——MCUCR 的 SRE——外部 SRAM 使能位把 C 口设置为第二功能,更改的设置会覆盖数据方向存储器。

C 口的输入引脚地址 PINC 不是一个寄存器,该地址允许对端口 C 的每一个引脚进行存取。当读 C 口时,读到的是 C 口的数据锁存器;当读 PINC 时,引脚上的逻辑值被读入。

四、D 口特性

D 口是一个带内部上拉的 8 位双向 I/O 口。D 口占了 3 个数据存储器的地址,一个是数据寄存器 PORTD $12($32)、数据方向寄存器 DDRD $11($31)和端口 D 输入引脚 PIND $10($30)。D 口的引脚地址是只读的,而数据寄存器和数据方向寄存器可以读/写。某些 D 口的引脚有第二功能如表 4.13 所示。

表 4.13 D 口引脚选择功能

口引脚	第二功能	口引脚	第二功能
PD0	RDX(UART 输入线)	PD5	OC1A(T/C1 输出比较 A 匹配输出)
PD1	TDX(UART 输出线)	PD6	$\overline{WR}$(写选通)
PD2	INT0(外部中断 0 输入)	PD7	$\overline{RD}$(读选通)
PD3	INT1(外部中断 1 输入		

当 D 口的引脚被用于第二功能时,DDRD 和 PORTD 寄存器必须按照第二功能来设置。

有关存储器编程的内容请参考第二章。

4.4 AT90S2323 单片机

AT90S2323 是 AVR 增强性能 RISC 结构的低功耗 CMOS 技术八位微控制器，图 4.17 为其结构图。

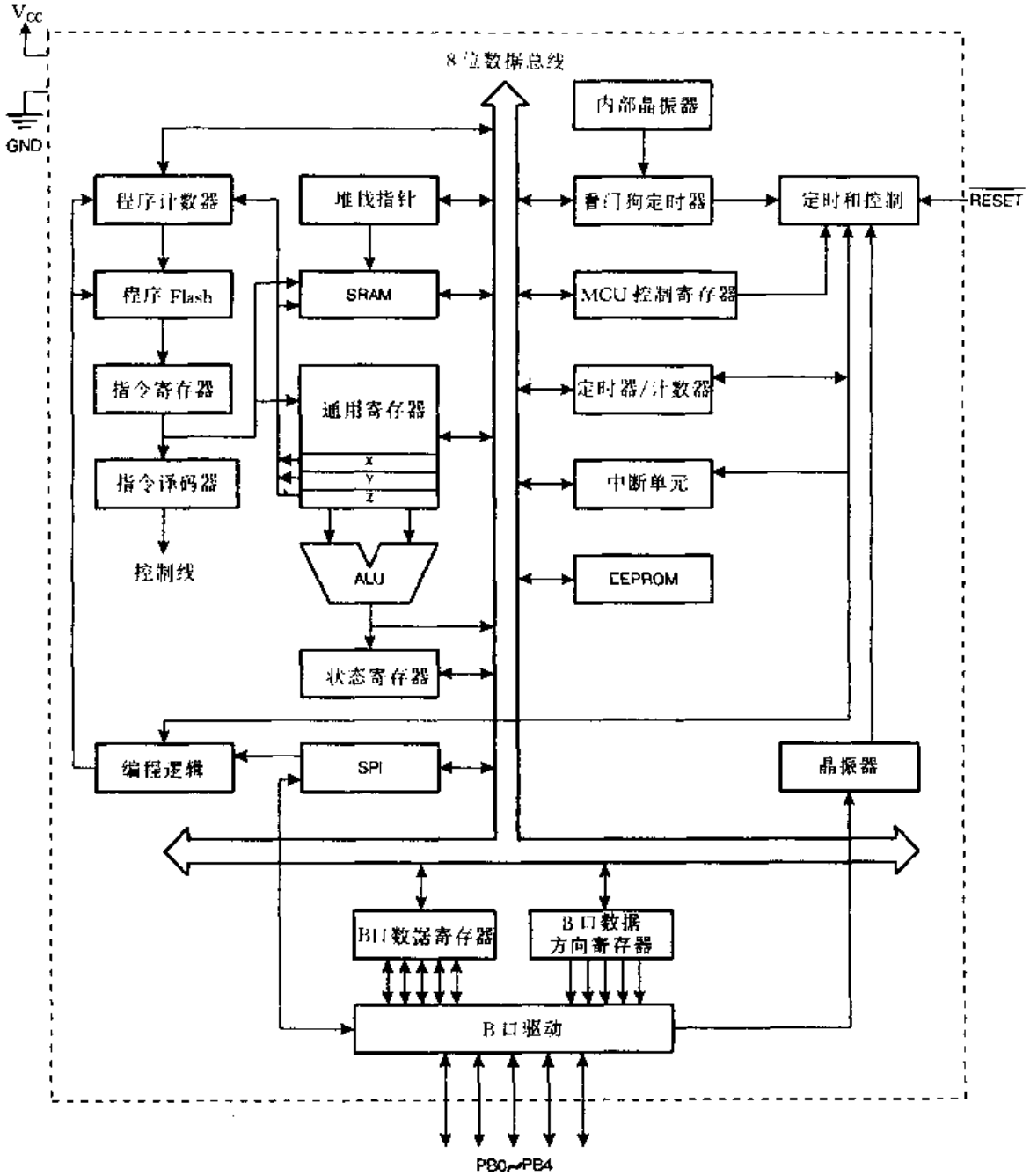


图 4.17 AT90S2323 结构图

该结构有效地支持高级语言，以及密集度极大的汇编器代码程序。该单片机具有以下特征：供电电压在 V_{CC}为 4 ～ 6 V 内可以全静态工作范围为 0～8 MHz；120 条功能强大的指令，大多数执行时间为单个时钟周期，指令周期时间为 125 ns @ 8 MHz，2K 字节可下载的 Flash

存储器(程序下载采用 SPI 串行接口,使用寿命为 1 000 次),128 字节 EEPROM(使用寿命为 10 万次);128 字节内部 RAM ,5 条可编程 I/O 线,32 个 8 位通用寄存器;内部及外部中断,带有一个 8 位可预分频的定时器/计数器,可编程的看门狗和片内晶振器,片内模拟比较器;一个为下载程序而设计的串行口,以及 2 个可通过软件选择的电源保留模式。闲置模式停止 CPU 的工作,而寄存器、定时器/计数器、看门狗及中断系统继续工作。掉电模式保留寄存器的内容,但冻结晶振、终止芯片的其它功能,直至下一次外部中断或硬件复位。

4.4.1　引脚说明

AT90S2323 为 8 引脚 PDIP 和 SOIC 封装单片机的,图 4.18 为其引脚图。

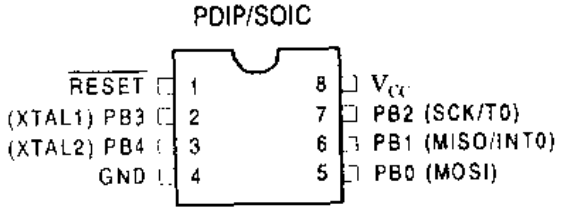


图 4.18　AT90S2323 的引脚图

一、引脚定义

V_{CC}(8 脚):供电引脚。

GND(4 脚):接地引脚。

B 口(PB4 – PB0):B 口为一个 5 位双向 I/O 口,引脚可提供内部拉高。当 B 口使用片内的晶振器时,B 口为一个 3 位双向 I/O 口。

$\overline{\text{RESET}}$(1 脚):复位输入。当晶振运行时,引脚上一个两周期的低电流可对器件进行复位。

XTAL1(2 脚):向晶振放大器的输入和向内部时钟操作电路的输入。

XTAL2(3 脚):从反转晶振放大器的输出。

二、晶振器

XTAL1 和 XTAL2 单独地作为反转放大器的输入和输出,该放大器可被设置为片内的晶振器。为了由外部源驱动器件,当 XTAL1 被驱动时,XTAL2 不能连接。详情可参考第二章。

4.4.2　AVR RISC 微控制器 CPU

AT90S2323 AVR RISC 微控制器向上与 AVR 增强 RISC 结构兼容。为 AT90S2323 MCU 而编写的程序,与 AVR 8 位 MCU(AT90SXXXX)全部系列产品的源代码和运行的时钟周期相兼容。

一、AVR 的结构

快速访问寄存器文件概念包括 32 个带有单一时钟周期访问时间的 8 位通用寄存器。这意味着在一个单一时钟周期内,执行一个 ALU(运算逻辑单元)。两个操作数为寄存器文件的输出,操作数被执行,运行结果被存储回寄存器文件。

32 个寄存器中的 6 个为进行数据空间寻址可被用作 3 个 16 位间接地址寄存器指针,从而可以进行高效的地址计算。3 个寄存器中的一个还被用作地址指示器,完成常量表的查询功能。这些新加的功能寄存器为 16 位 X – 寄存器、Y – 寄存器、Z – 寄存器。ALU 支持寄存器之间的运算与逻辑功能,以及常数和寄存器之间的运算与逻辑功能,图 4.19　所示为 AT90S2323 AVR 的 RISC 结构。

除了寄存器操作功能之外,常规存储器寻址模式还可用在寄存器文件上。寄存器文件由最低的 32 个数据空间地址($00 ~ $1F)分派而使能,从而使得即使它们为一般的存储地址,也可被访问到。I/O 存储器空间包含 64 个作为 CPU 外围功能的地址,为控制寄存器、定时器

/计数器、A/D 转换器，及其它 I/O 功能。I/O 存储器可被直接访问，或作为寄存器文件之后的数据空间地址 $20～$5F。128 字节的数据 SRAM + 寄存器文件和 I/O 寄存器，可以通过 AVR 结构中的 5 种不同写址模式被很容易地访问。

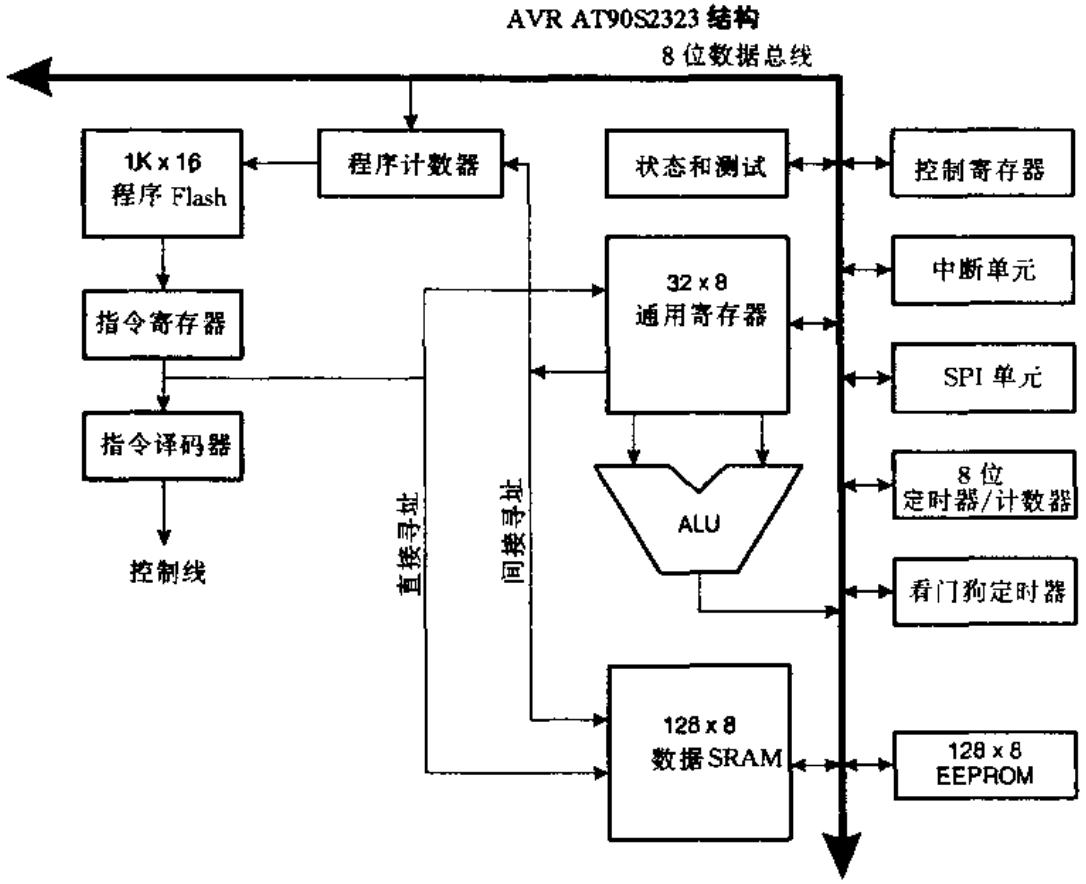


图 4.19 AT90S2323 AVR 的 RISC 结构

AVR 结构中的存储器空间均为线性和规律的存储器映射，存储器映射图同图 4.9 所示。

二、通用寄存器文件

在 CPU 中，AT90S2313 的 32 个通用寄存器文件的结构图与图 4.10 所示的相同。每个寄存器被分配给一个数据存储器地址，将其直接映射，如用户数据空间的前 32 个地址。虽然寄存器文件并未被物理地作为 SRAM 地址而提供出来，但由于寄存器 X、Y 和 Z 可被设置，来对文件中的寄存器做索引。这种存储器的组成结构在访问寄存器时提供了极大的灵活性。

三、ALU 运算逻辑单元

高性能的 AVR 运算逻辑单元的操作与所有的 32 个通用寄存器保持直接的连接。在单一时钟周期内，ALU 是在寄存器文件中的寄存器之间执行操作。ALU 操作被分为 3 个主要的目的：算法、逻辑和位功能。AVR 产品家族中一些微控制器的 ALU 运算部件中有硬件乘法器。

四、可下载的 Flash 程序存储器

AT90S2323 包括 2K 字节的片内可下载 Flash 存储器。由于所有指令为 16 位字或 32 位字，用于存储程序的 Flash 的结构为 1K × 16。Flash 存储器的使用寿命最少为 1 000 次写/擦循环。AT90S2323 程序设计器宽为 10 位，以此来对 1 024 个程序存储器地址写址。

常量表必须被设定在 0～2K 的地址之间(请参考 LPM——装入程序存储器指令说明)。

五、EEPROM 数据存储器

AT90S2323 包括 128 字节的 EEPROM 存储器。它被组织为一个分开的数据空间，在其间单一的字节可被读写。EEPROM 的使用寿命为至少 100 000 次写/擦循环。

六、SRAM 数据存储器

AT90S2323 的 SRAM 数据存储器组成与图 4.11 AT90S2313 的数据存储组成相同。

低端 224 个数据存储器地址编址为寄存器文件、I/O 存储器和数据 SRAM。前 96 个地址编址为寄存器文件 + I/O 存储器，接下来的 128 个地址是对数据 SRAM。

七、编程和数据寻址模式

编程和数据寻址模式，AT90S2323 同于 AT90S2313，见 4.2.2 节。

八、存储器访问和指令执行时序

存储器访问和指令执行时序，AT90S2323 同于 AT90S2313，见 4.2.2 节。

九、I/O 存储器

表 4.14 所示为 AT90S2323 的 I/O 空间定义。

表 4.14 AT90S2323 的 I/O 空间

十六进制地址	名 称	功 能
$3F($5F)	SREG	状态寄存器
$3D($5D)	SPL	堆栈指针低
$3B($5B)	GIMSK	通用中断屏蔽寄存器
$3A($5A)	GIFR	通用中断标志寄存器
$39($59)	TIMSK	定时器/计数器中断屏蔽寄存器
$38($58)	TIFR	定时器/计数器中断标志寄存器
$35($55)	MCUCR	MCU 通用控制寄存器
$33($53)	TCCR0	定时器/计数器 0 控制寄存器
$32($52)	TCNT0	定时器/计数器 0(8 位)
$21($41)	WDTCR	看门狗定时控制寄存器
$1E($3E)	EEAR	EEPROM 地址寄存器
$1D($3D)	EEDR	EEPROM 数据寄存器
$1C($3C)	EECR	EEPROM 控制寄存器
$18($38)	PORTB	B 口数据寄存器
$17($37)	DDRB	B 口数据方向寄存器
$16($36)	PINB	B 口输入脚

注意：保留的和未用到的地址未列。

十、复位和中断处理

1. 复位源

AT90S2323 提供 2 种中断源。这些中断和与之分开的复位向量均在程序存储器空间中各自带有分开的程序向量。所有中断均分配给单独的使能位，这些使能位须与状态寄存器中的 I 位一起被设为 1，以便能执行中断。

程序存储器空间中最下面的地址被自动地定义为复位和中断向量，如表 4.15 所列。表中还列出了不同中断的优先级，地址越低，优先级越高。RESET 有最高的优先级，以下依次为 INT0……。

表 4.15 复位和中断向量

向量号	程序地址	源	中断定义
1	$000	RESET	硬件脚和看门狗复位
2	$001	INT0	外部中断请求 0
3	$002	TIMER0 OVF0	定时器/计数器 0 溢出

AT90S2323 有与 AT90S1200 相同的 3 个复位源，且复位过程也与之相同，见 4.1.3 节。

2. 中断处理

AT90S2323 有 2 个 8 位中断屏蔽控制寄存器，即 GIMSK 通用中断屏蔽寄存器和 TIMSK 定时器/计数器中断寄存器。

十一、休眠模式

休眠模式 AT90S2323 同于 AT90S1200，见 4.1.3 节。

4.4.3 定时器/计数器

AT90S2323 有一个 8 位通用定时器/计数器。定时器/计数器从 10 位的预定比例定时器获取独立的预定比例区域。定时器/计数器可被用作带内部时基的定时器，或被用作带可触发计数的外部引脚连接的计数器。

一、定时器/计数器预定比例器

通用定时器/计数器的预定标器请参考第二章。

二、8 位定时器/计数器 0

8 位的定时器/计数器 0 可从 CK、预定比例 CK，或外部引脚来选择时钟源。另外，它还可以像定时器/计数器 0 的控制寄存器 TCCR0 规格中所描述的那样而停止。

溢出状态比例标志位可在定时器/计数器中断比例标志位寄存器 TCCR0 中。定时器/计数器 0 的中断使能/禁止设置在定时器/计数器中断的控制屏蔽寄存器 TIMSK 中。

4.4.4 看门狗定时器

看门狗定时器 AT90S2323 同于 AT90S1200，见 4.1.5 节。

4.4.5 EEPROM 读/写访问

EEPROM 读/写访问，AT90S2323 同于 AT90S2313，见 4.2.5 节。

4.4.6 I/O 口

B 口为一个 5 位的双向 I/O 口。B 口分配有 3 个数据存储地址，分别为数据寄存器 PORTB $18($38)、数据方向寄存器 DDRB $17($37)和 B 口的输出引脚 $16($36)。B 口的输入引脚地址为只读，而数据寄存器和数据方向寄存器为可读写。具有可选择功能的 B 口引脚如表 4.16 所示。

表 4.16 B 口引脚选择功能

口 引 脚	第 二 功 能
PB0	MOSI （储存器下载数据输入线）
PB1	MISO （储存器下载数据输入线） INT0 （外部中断 0 输入）
PB2	SCK （串行时钟输入） T0 （定时器/计数器 0 计数时钟输入）
PB3	XTAL1（晶振输入）
PB4	XTAL2（晶振输出）

当引脚被用作可选择的功能时，DDRB 和 PORTB 寄存器必须据可供选择的功能来设置。

有关存储器编程的内容请参考第二章。

4.5 AT90S8515 单片机

4.5.1 概 述

AT90S8515 是 AVR® 增强性能 RISC 结构的低功耗 CMOS 技术八位微控制器，图 4.21 为其结构图。

AT90S8515 系列单片机具有以下特征：8K 字节可下载的 Flash 存储器，512 字节 EEPROM，512 字节 RAM，32 条通用的 I/O 线，32 个通用寄存器，带比较模式的灵活性定时器/计时器，可编程的串行 UART，内部及外部中断，带内部晶振的可编程看门狗定时器；一个为下载程序而设计的 SPI 串行口，以及 2 个可通过软件选择的省电模式。闲置模式停止 CPU 的工作，而 SRAM、定时器/计数器、SPI 口，及中断系统继续工作。掉电模式保留寄存器的内容，但冻结晶振、终止芯片的其它功能，直至下一次外部中断或硬件复位。

4.5.2 引脚说明

AT90S8515 为 40 引脚 PDIP 和 SOIC 封装的单片机。图 4.20 为其引脚图。具体内容请参考第二章。

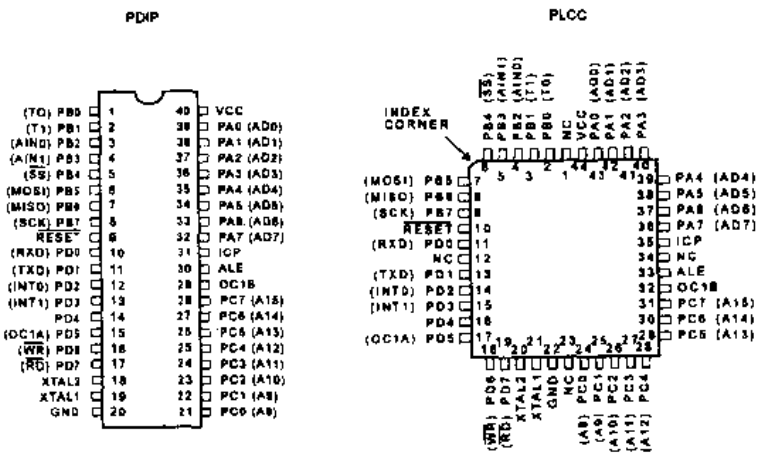


图 4.20 AT90S8515 的引脚图

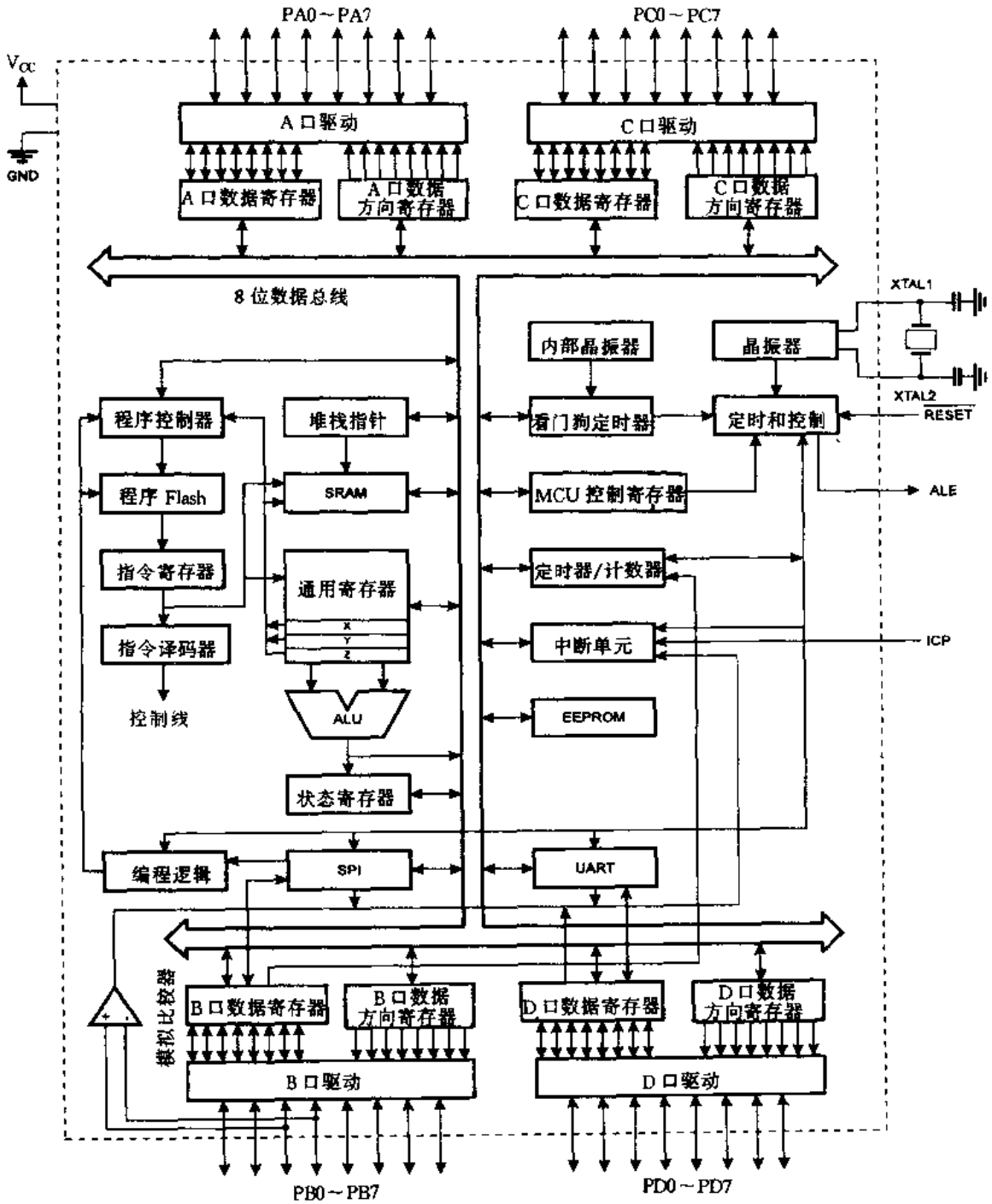


图 4.21 AT90S8515 结构图

4.6 AT90SMEG103 单片机

AT90SMEG103 是 AVR 增强性能 RISC 结构的低功耗 CMOS 技术八位微控制器。图 4.22 为其结构图。

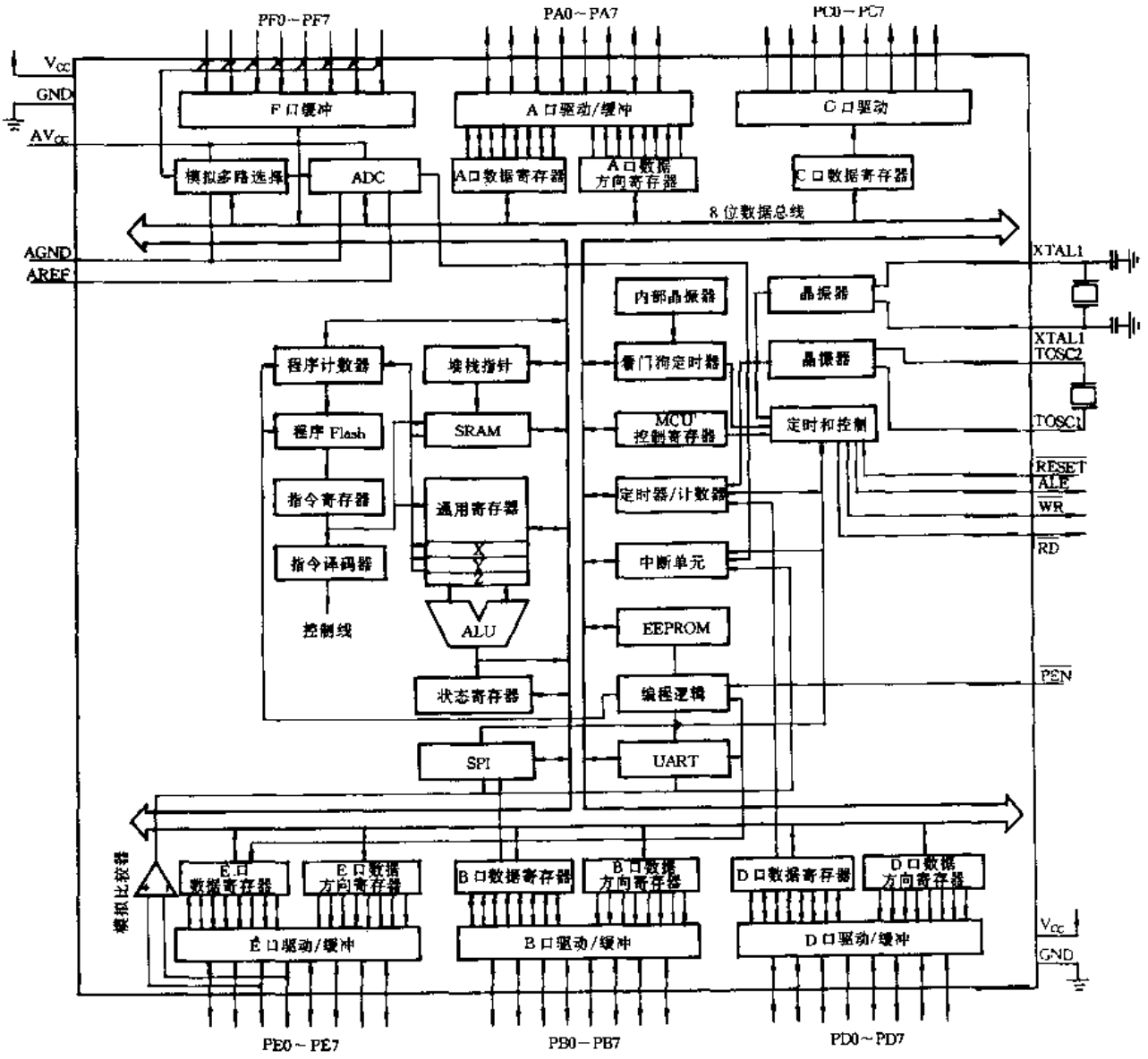


图 4.22 AT90SMEG103 结构图

该结构有效地支持高级语言,以及密集度极大的汇编器代码程序。该单片机具有以下特征:供电电压在 V_{CC}为 2.7 ～ 6 V 内可以全静态工作范围为 0～12 MHz;120 条功能强大的指令,大多数执行时间为单个时钟周期,指令周期时间为 83.8 ns @ 12 MHz,4K 字节可下载的 Flash 存储器(程序下载采用 SPI 串行接口,使用寿命为 1 000 次),128 字节 EEPROM(使用寿命为 10 万次);4K 字节内部 RAM ,32 条可编程 I/O 线,8 条输入线、8 条输出线,32 个 8 位通用寄存器,内部及外部中断,2 个 8 位带预分频器及 PWM 的定时器 / 计数器,1 个 16 位带单独预分频器、比较模拟及双工 8 位、9 位或 10 位 PWM 的定时器 / 计数器,可编程串行 UART + SPI 串行接口,外部和内部中断源,带片内晶振器的可编程看门狗定时器,带单独晶振的 RTC,8 通道、10 位 ADC,片内模拟比较器,以及 2 个可通过软件选择的电源保留模式。闲置

模式停止 CPU 的工作，而寄存器、定时器/计数器、看门狗及中断系统继续工作。掉电模式保留寄存器的内容，但冻结晶振、终止芯片的其它功能，直至下一次外部中断或硬件复位。

4.6.1 引脚说明

AT90SMEG103 为 40 引脚 PDIP 和 SOIC 封装的单片机，图 4.23 为其引脚图。

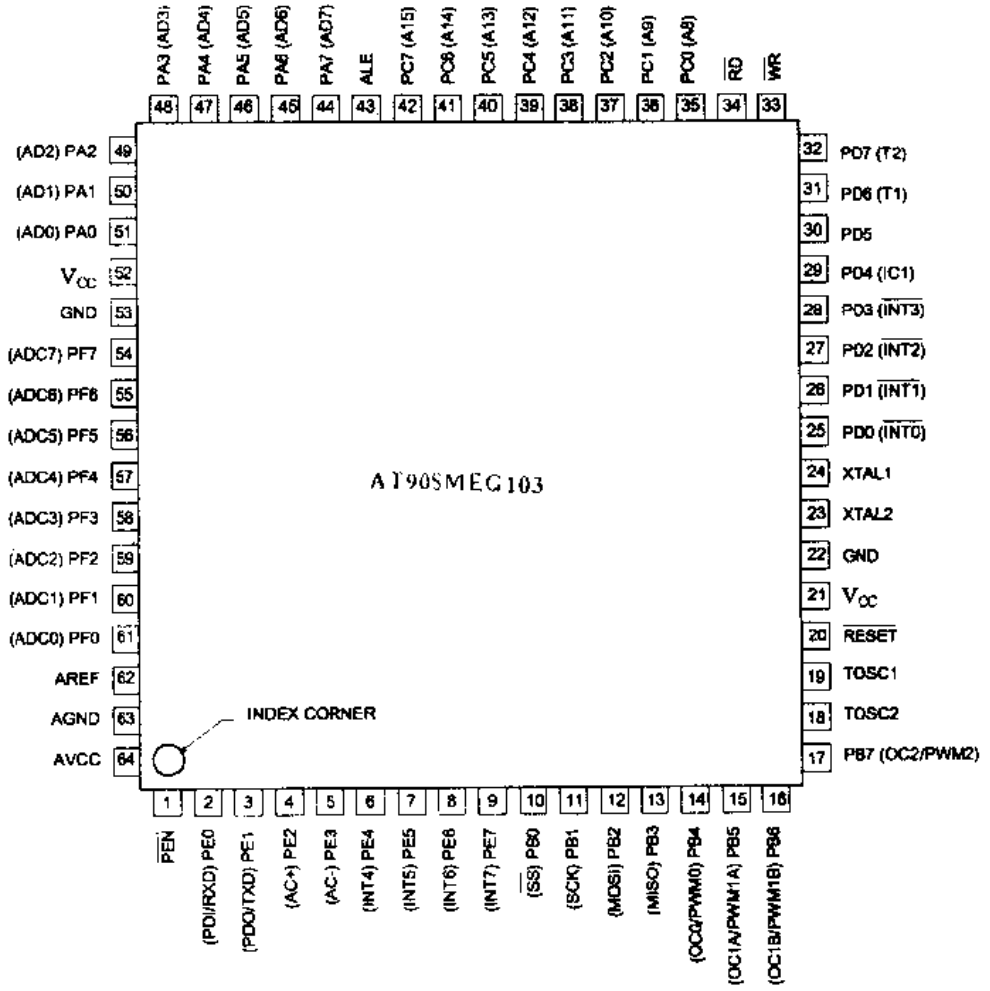


图 4.23 AT90SMEG103 的引脚图

一、引脚定义

V_{CC}：供电引脚。

GND：接地引脚。

A 口(PA7～PA0)：A 口为一个 8 位双向 I/O 口，口引脚可提供内部拉高(为每一位而选)。A 口输出缓冲器可吸引 40 mA 的电流，且直接驱动 LED 显示。当引脚 PA0～PA7 用作输入且被外部拉低时，若内部拉高被激活，它们将提供源电流(I_{IL})。当使用外围 SRAM 时，A 口作为多路传输地址/数据总线。

B 口(PB7～PB0)：B 口为带内部拉高的 8 位双向 I/O 口。B 口输出缓冲器可吸引 40 mA 的电流。作为输入，被外部拉低的 B 口引脚，若拉高被激活，则提供电流(I_{IL})。

C 口(PC7～PC0)：C 口为 8 位输出口。C 口输出缓冲器可吸引 40mA 的电流。当使用外

部SRAM时,C口也作为地址输出。

D口(PD7～PD0):D口为带内部拉高的8位双向I/O口。D口输出缓冲器可吸引40 mA的电流。作为输入,若拉高被激活,D口被外部拉低的引脚将提供源电流(I_{IL})。

E口(PE7～PE0):E口为带内部拉高的8位双向I/O口。E口输出缓冲器可吸引40 mA的电流。作为输入,若拉高被激活,E口被外部拉低的引脚将提供源电流(I_{IL})。

F口(PF7～PF0):F口为带内部拉高的8位双向I/O口。F口也为ADC的模拟输入。

$\overline{\text{RESET}}$:输入。当晶振器运行时,该引脚上的2个机械周期对器件进行复位。

XTAL1:向反转晶振放大器的输入和向内部时钟操作电路的输入。

XTAL2:从反转晶振放大器发出的输入。

TOSC1:向反转定时器/计数器晶振放大器的输入。

TOSC2:从反转定时器/计数器晶振放大器的输出。

WR:外部SRAM写入选通。

RD:外部SRAM读取选通。

ALE:当外部存储器使能时,ALE为用到的地址锁使能。在第一次访问周期中,ALE用来将低序地址(8位)锁入一个地址锁;在第二个访问周期中,AD0～AD7引脚用作数据。

AV_{CC}:向A/D转换器的支持电压,须通过低通过滤器在外部与V_{CC}相连。见ADC操作。

AREF:ADC转换器的模拟参考输入。对ADC操作,须对该引脚加2.7 V到AV_{CC}的电压。

AGND:若电路板有一个单独的模拟地电平,该引脚须与地平电连接;否则,GND连接。

PEN:低电压串行编程模式下的编程使能引脚。通过在加电复位过程中将该引脚拉低,器件进入串行编程模式。

二、晶振器

XTAL1和XTAL2单独地作为反转放大器的输入和输出,该放大器可被设置为片内的晶振器。为了由外部源驱动器件,当XTAL1被驱动时,XTAL2不能连接。详情可参考第二章。

4.6.2 AVR RISC微控制器CPU

AT90SMEG103 AVR RISC微控制器向上与AVR增强RISC结构兼容。为AT90SMEG103MCU而编写的程序,与AVR 8位MCU(AT90SXXXX)全部系列产品的源代码和运行的时钟周期相兼容。

一、AVR的结构

快速访问寄存器文件概念包括32个带有单一时钟周期访问时间的8位通用工作寄存器。这意味着在一个单一时钟周期内,执行一个ALU(运算器)操作,两个操作数从寄存器文件输出,操作被执行,运行结果被存储回寄存器文件。

ALU支持寄存器之间,以及常数和寄存器之间的运算与逻辑功能。图4.24所示为AT90SMEG103 AVR的结构。

AVR结构中的存储器空间全部为线性有规律的存储器映射。图4.25为存储器映射图。

二、通用寄存器文件

在CPU中,AT90SMEG103的32个通用寄存器文件的结构图与图4.10所示的相同。每个寄存器被分配给一个数据存储器地址,将其直接映射如用户数据空间的前32个地址。虽然

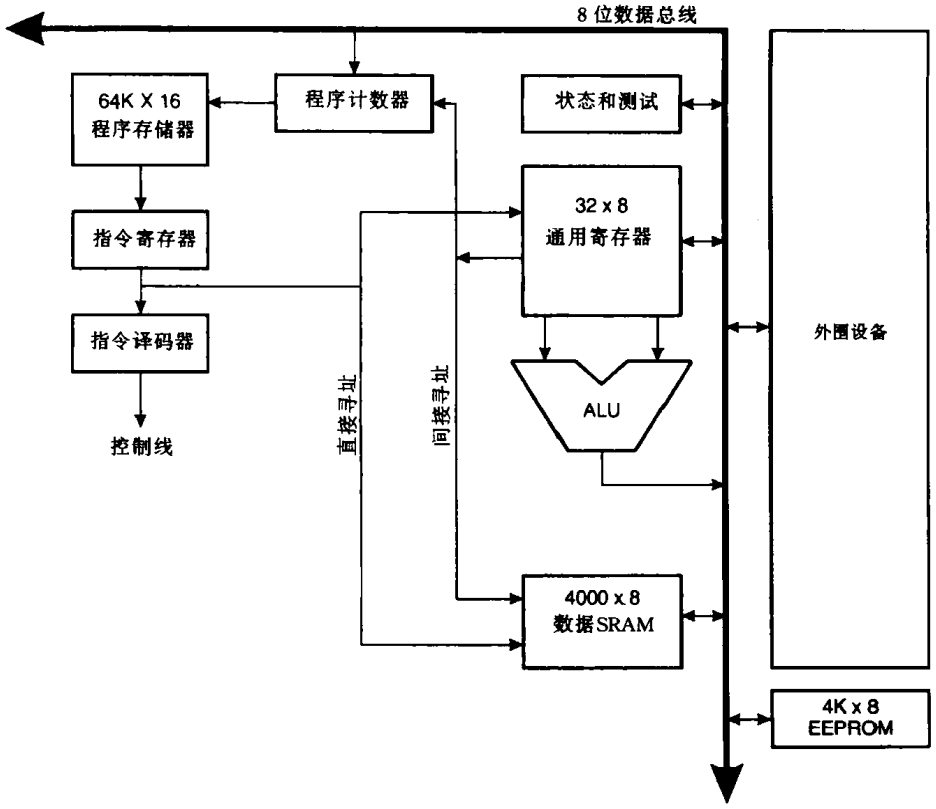


图 4.24 AT90SMEG103 AVR 结构

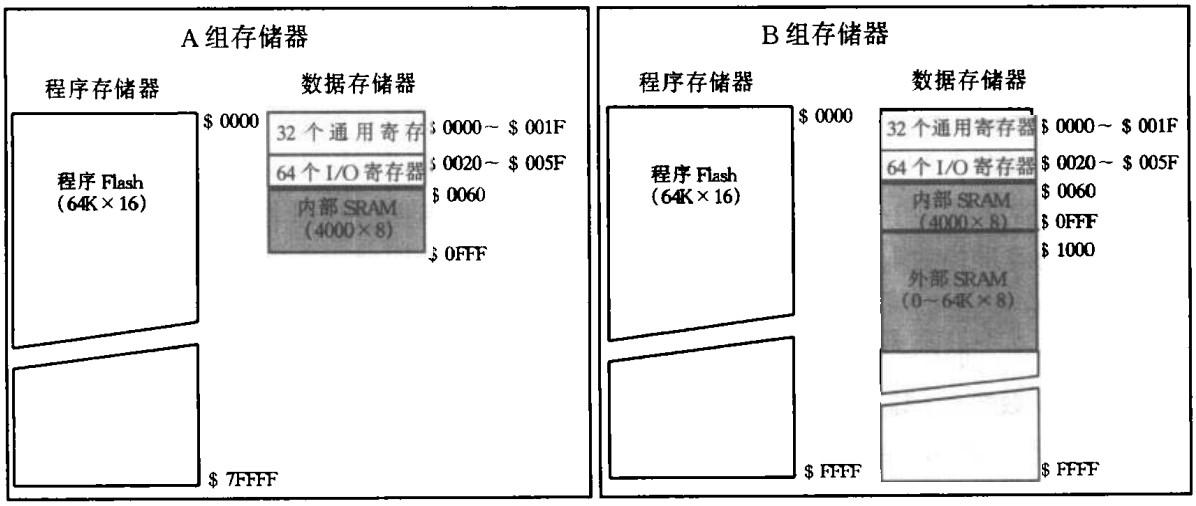


图 4.25 存储器映射

寄存器文件并未被物理地作为 SRAM 地址而提供出来，但由于寄存器 X、Y 和 Z 可被设置，来对文件中的寄存器做索引，这种存储器的组成结构在访问寄存器时却提供了极大的灵活性。

SRAM 的 4K 字节为通用数据而用，其地址从 $ 0060～ $ 0FF。

三、ALU 运算逻辑单元

高性能的 AVR 运算逻辑单元的操作与所有的 32 个通用寄存器保持直接的连接。在单一时钟周期内，ALU 是在寄存器文件中的寄存器之间执行操作。ALU 操作被分为 3 个主要的目的：

算法、逻辑和位功能。AVR 产品家族中一些微控制器的 ALU 运算部件中有硬件乘法器。

四、可下载的 Flash 程序存储器

AT90SMEG103 包括 128K 字节的片内可下载 Flash 存储器。由于所有指令为 16 位字或 32 位字,用于存储程序的 Flash 的结构为 64K×16。Flash 存储器的使用寿命最少为 1 000 次写/擦循环。

五、存储器配置

AT90SMEG103 支持表 4.17 所示的 2 种存储器配置。

表 4.17 2 种存储器配器

配 置	内部 SRAM 数据储存器	外部 SRAM 数据储存器
A	4 000	无
B	4 000	到 64K

开始的 4 096 个数据存储器地址是寄存器文件、I/O 存储器,以及内部数据 SRAM 地址。开始的 96 个地址是寄存器文件、I/O 存储器,后面的 4 000 个地址是内部数据 SRAM。

一个可选的外部数据 SRAM 可被 AT90SMEG103 使用。该 SRAM 占有 64K 地址空间中剩下的区域。该区域在 Data Flash 后面的地址处开始。若该 Flash 被用作程序存储器,它在内部 SRAM 后面的地址处开始。若使用 64K 的外部 SRAM,外部存储器中的 4K 丢失,作为被内部存储器所占用的地址。

当访问 SRAM 存储器空间的地址超过内部数据存储器的空间,外部数据 SRAM 通过使用相同的指令来访问,作为对内部数据存储器的访问。当内部数据存储器被访问到时,读写选通引脚($\overline{RD}$及$\overline{WR}$)在整个访问周期中不激活。外部 SRAM 操作通过设置 MCUCR 寄存器中的 SRE 位来使能。

六、编程和数据寻址模式

AT90SMEG103 AVR 增强 RISC 微控制器支持对程序存储器(Flash)和数据存储器(SRAM、寄存器文件和 I/O 存储器)访问的功能强大且效率高的寻址模式。AT90SMEG103 结构所支持的寻址模式有单一寄存器直接(Rd)、两个寄存器直接(Rd 和 Rr)、I/O 直接、数据直接、带位移的数据间接、数据间接、带预减量的数据间接、带后增量的数据间接、使用 LPM 指令的常量地址、间接程序寻址、相关程序寻址等寻址模式。

七、EEPROM 数据存储器

AT90SMEG103 带有 4K 字节的数据 EEPROM 存储器。其构成为一个单独的数据空间,其中的单一字节可以被读取和写入。EEPROM 的使用寿命至少为 10 万次写/擦。后面对 EEPROM 与 CPU 之间的访问有描述,确定了 EEPROM 地址寄存器、数据寄存器及控制寄存器。

八、存储器访问及指令执行时序

CPU 由系统时钟 Φ 驱动,直接由芯片的外部时钟晶振激活,没有使用内部时钟部分。AVR 单片机的 Harvard 结构和快速访问寄存器文件概念,能使得并行指令存取和指令执行。这种基本的流水线概念目的是为了获得高达每 1MIPS/MHz 的效率。有关指令执行和内部存储器访问的通用访问时序请 参考第二章。

九、I/O 存储器

表 4.18 所示为 AT90SMEG103 单片机的 I/O 空间定义。

表 4.18 AT90SMEG103 I/O 空间

十六进制地址	名 称	功 能
$3F($5F)	SREG	状态寄存器
$3E($5E)	SPH	堆栈指针高
$3D($5D)	SPL	堆栈指针低
$3C($5C)	XDIV	XTAL 分频控制器
$3B($5B)	RAMPZ	RAM 页 Z 选择寄存器
$3A($5A)	EICR	外部中断控制志存器
$39($59)	EIMSK	外部中断屏蔽寄存器
$38($58)	EIFR	外部中断标志寄存器
$37($57)	TIMSK	定时器/计数器中断屏蔽寄存器
$36($56)	TIFR	定时器/计数器中断标志寄存器
$35($55)	MCUCR	MCU 通用控制寄存器
$34($54)	MCUSR	MCU 通用控制寄存器
$33($53)	TCCR0	定时器/计数器 0 控制寄存器
$32($52)	TCNT0	定时器/计数器 0(8 位)
$31($51)	OCR0	定时器/计数器 0 输出比较寄存器
$30($50)	ASSR	异步方式状态寄存器
$2F($4F)	TCCR1A	定时器/计数器 1 控制寄存器 A
$2E($4E)	TCNT1B	定时器/计数器 1 控制寄存器 B
$2D($4D)	TCNT1H	定时器/计数器 1 高字节
$2C($4C)	TCNT1L	定时器/计数器 1 低字节
$2B($4B)	OCR1AH	定时器/计数器 1 输出比较寄存器 A 高字节
$2A($4A)	OCR1AL	定时器/计数器 1 输出比较寄存器 A 低字节
$29($49)	OCR1BH	定时器/计数器 1 输出比较寄存器 B 高字节
$28($48)	OCR1BL	定时器/计数器 1 输出比较寄存器 B 低字节
$27($47)	ICR1H	定时器/计数器 1 输入捕获寄存器高字节
$26($46)	ICR1L	定时器/计数器 1 输入捕获寄存器低字节
$25($45)	TCCR2	定时器/计数器 2 控制寄存器
$24($44)	TCNT2	定时器/计数器 2(8 位)
$23($43)	OCR2	定时器/计数器 2 输出比较寄存器
$21($41)	WDTCR	看门狗定时控制寄存器
$1F($3F)	EEARH	EEPROM 地址寄存器高字节
$1E($3E)	EEARL	EEPROM 地址寄存器低字节
$1D($3D)	EEDR	EEPROM 数据寄存器
$1C($3C)	EECR	EEPROM 控制寄存器
$1B($3B)	PORTA	A 口数据寄存器
$1A($3A)	DDRA	A 口数据方向寄存器
$19($39)	PINA	A 口输入脚
$18($38)	PORTB	B 口数据寄存器
$17($37)	DDRB	B 口数据方向寄存器
$16($36)	PINB	B 口输入脚
$15($35)	PORTC	C 口数据寄存器
$12 ($32)	PORTD	D 口数据寄存器

表 4.18 续

十六进制地址	名 称	功 能
$11($31)	DDRD	D 口数据方向寄存器
$10($30)	PIND	D 口输入脚
$0F($2F)	SPDR	SPI I/O 数据寄存器
$0E($2E)	SPSR	SPI 状态寄存器
$0D($2D)	SPCR	SPI 控制寄存器
$0C($2C)	UDR	UART I/O 数据寄存器
$0B($2B)	USR	UART 状态寄存器
$0A($2A)	UCR	UART 控制寄存器
$09($29)	UBRR	UART 波特率寄存器
$08($28)	ACSR	模拟比较控制和状态寄存器
$07($27)	ADMUX	ADC 多路选择寄存器
$06 ($26)	ADCSR	ADC 控制和状态寄存器
$05 ($25)	ADCH	ADC 数据寄存器高
$04 ($24)	ADCL	ADC 数据寄存器低
$03 ($23)	PORTE	E 口数据寄存器
$02($22)	DDRE	E 口数据方向寄存器
$01($21)	PINE	E 口输入脚
$00($20)	PINF	F 口输入脚

AT90SMEG103 所有的 I/O 口和外围设备均在 I/O 空间中置好。不同的 I/O 地址可以通过 IN 和 OUT 指令访问。当使用 IN 和 OUT 时，使用 I/O 寄存器地址的 $00～$3F。由于 I/O 寄存器也在 SRAM 地址空间中被表示，它们亦可写址为地址空间 $20～$5F 间的二进制 SRAM 地址。通过加 $20 到直接 I/O 地址而获得 SRAM 地址。本文中 SRAM 地址在 I/O 直接地址之后的括号中给出。在地址范围 $00($20)～$1F($3F)中的 I/O 寄存器，使用 SBI 和 CBI 指令可直接访问到位。详情请 参考第二章。

1. RAM 页 Z 选寄存器——RAMPZ

位	7	6	5	4	3	2	1	0	
$3B ($5B)	–	–	–	–	–	–	–	RAMPZ0	RAMPZ
读/写	R	R	R	R	R	R	R	R	
初始化值	0	0	0	0	0	0	0	0	

RAMPZ 寄存器通常用来选择 64K 的 RAM 页被指针 Z 访问。由于 AT90SMEG103 不支持多于 64K 的 SRAM 存储器，该寄存器只用来选择当使用 ELPM 指令时，程序存储器中的哪一页可以访问。RAMPZ0 位的不同设置有不同的作用：

RAMPZ0 = 0：程序存储器地址 $0000～$7FFF(低位 64K 字节)由 ELPM 访问。

RAMPZ0 = 1：程序存储器地址 $8000～$FFFF(高位 64K 字节)由 ELPM 访问。

2. MCU 控制寄存器——MCUCR

位	7	6	5	4	3	2	1	0	
$35($55)	SRE	SRW	SE	SM1	SM0	-	-	-	MCUCR
读/写	R/W	R/W	R/W	R/W	R/W	R	R	R	
初始化值	0	0	0	0	0	0	0	0	

MCU 控制寄存器包含通用 MCU 功能的控制位。

位 7—SRE:外部 SRAM 使能

当 SRE 位被设为 1 时,外部数据 SRAM 使能,且引脚功能 AD0～AD7(A 口)、A8～15(C 口)和(D 口)作为可选的引脚功能被激活。之后,SRE 位越过独立的数据方向寄存器中任何引脚方向的设置。当 SRE 位被清 0,外部数据 SRAM 禁止,且正常的引脚和数据方向设置可用。

位 6—SRW:外部 SRAM 等待状态

当 SRW 位被设为 1 时,一个周期的等待状态被插入外部数据 SRAM 访问周期。当 SRW 位被清为 0 时,外部数据 SRAM 访问被一个三周期的方案执行。

位 5—SE:休眠使能

SE 位必须被设为 1,以便使 CPU 在执行 SLEEP 指令时进入休眠状态,除非是出于编程器的目的。为防止 CPU 进入休眠状态,推荐只在执行 SLEEP 指令之前设置休眠使能位为使能。

位 4,3—SM1,SM0:休眠模式选择位 1 和 0

该位在表 4.19 所提供的 3 种休眠模式之中选择其一。

表 4.19 3 种休眠模式

SM1	SM0	休眠模式	SM1	SM0	休眠模式
0	0	闲置模式	1	0	掉电模式
0	1	备用模式	1	1	省电模式

位 2～0—Res:保留位

AT90SMEG103 中的该位为保留位,总被读 0。

3. XTAL 除控制寄存器

位	7	6	5	4	3	2	1	0	
$3C($5C)	XDIVEN	XDIV6	XDIV5	XDIV4	XDIV3	XDIV2	XDIV1	XDIV0	XDIV
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

XTAL 除控制寄存器通过 0～129 的数字来除 CPU 的时钟频率。这一特性可以在需要处理低电压的情况下用来减少功耗。

位 7—XDIVEN:XTAL 除使能

当 XDIVEN 位被设为 1 时,CPU 的时钟频率及所有外围设备被由从 XDIV6～XDIV0 定义的因子除。该位根据应用可以设置及清除运行时间到不同的时钟频率。

位 6～0—XDIV6～XDIV0:XTAL 除选择位 6～0

这些位定义除因子,该因子当 XDIVEN 位设为 1 时工作。若这些位的值为 d,下列公式定义 CPU 的时钟频率 f_{CLK}:

$$f_{CLK} = XTAL/(129 - d)$$

只有当 XDIVEN 为 0 时，这些位的值才可被更改。当 XDIVEN 被设为 1 时，被同步写入 XDIV6～XDIV0 的值被提取作为划分因子。当 XDIVEN 被清为 0 时，被同步写入 XDIV6～XDIV0 的值被拒绝。由于划分器将上时钟输出划分进入 MCU，所有外围设备在使用划分因子时速度降低。

十、复位和中断处理

1. 复位源

AT90SMEG103 提供 23 种不同的中断源。这些中断和与之分开的复位向量均在程序存储器空间中各自带有分开的程序向量。所有中断均分配给单独的使能位，这些使能位须与状态寄存器中的 I 位一起被设为 1，以便能执行中断。

程序存储器空间中最下面的地址被自动地定义为复位和中断向量，如表 4.20 所列。表中还列出了不同中断的优先级，地址越低，优先级越高。RESET 有最高的优先级，以下依次为 INT0……。

表 4.20 复位和中断向量

向量号	程序地址	源	中断定义
1	$0000	RESET	硬件脚和看门狗复位
2	$0002	INT0	外部中断请求 0
3	$0004	INT1	外部中断请求 1
4	$0006	INT2	外部中断请求 2
5	$0008	INT3	外部中断请求 3
6	$000A	INT4	外部中断请求 4
7	$000C	INT5	外部中断请求 5
8	$000E	INT6	外部中断请求 6
9	$0010	INT7	外部中断请求 7
10	$0012	TIMER2 COMP	定时/计数器 2 比较匹配
11	$0014	TIMER2 OVF	定时/计数器 2 溢出
12	$0016	TIMER1 CAPT	定时/计数器 1 捕获事件
13	$0018	TIMER1 COMPA	定时/计数器 1 比较匹配 A
14	$001A	TIMER1 COMPB	定时/计数器 1 比较匹配 B
15	$001C	TIMER1 OVF	定时/计数器 1 溢出
16	$001E	TIMER0 COMP	定时/计数器 0 比较匹配
17	$0020	TIMER0 OVF	定时/计数器 0 溢出
18	$0022	SPI, STC	串行传送完成
19	$0024	UART, RX	UART, RX 完成
20	$0026	UART, UDRE	UART 数据寄存器空
21	$0028	UART, TX	UART, TX 完成

表 4.20 续

向量号	程序地址	源	中断定义
22	$002A	ADC	ADC 转换器完成
23	$002C	EE READY	EEPROM 准备
24	$002E	ANA-COMP	模拟比较器

AT90SMEG103 有 3 个复位源：

· 加电复位。当供电电平加至 V_{CC}和 GND 引脚时，MCU 进行复位。

· 外部复位。当一个低平电平加到引脚上多于 2 个 XTAL 周期时，MCU 进行复位。

· 看门狗复位。当看门狗定时器超时，且看门狗为使能时，MCU 进行复位。

在复位过程中，除了 MCU 状态寄存器之外，I/O 寄存器被设为初始值，程序从地址 $000 开始执行。$000 地址中放置的指令须为某一 JMP——相关转移，即，到达复位处理路径的指令。若程序从没有对中断源使能，则中断向量无法使用，正常的程序代码可以放置在这些地址中。

2. 中断处理

AT90SMEG103 有 2 个 8 位中断屏蔽控制寄存器，即 EIMSK 外部中断屏蔽寄存器和 TIMSK 定时器/计数器中断屏蔽寄存器。

当某一中断发生时，总中断使能位 I 位被清空(0)，则全部中断被禁止。用户软件必须将该位改为 1 来使中断使能。当执行 RETI，从中断指令返回时，I 位被设为 1。

当程序计数器指向现行中断时，执行中断处理程序，硬件将相应的产生中断的标志位清空。

· 外部中断屏蔽寄存器——EIMSK

位	7	6	5	4	3	2	1	0	
$39($59)	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	EIMSK
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

位 7～4～INT7～INT4：外部中断请求 7～4 使能

当 INT7～INT4 的某一位被设置，且状态寄存器(SREG)中 I 位被设 1 时，通讯外部引脚中断使能。外部中断控制寄存器 EICR 中的内部传感控制位定义外部中断是否在上升沿。下降沿或检测过的级上被触发。这些引脚上的任何事件将触发中断请求，即使引脚被使能为输入。这提供了一种产生软件中断的方法。

位 3～0—INT3～INT2：外部中断请求 3～0 使能

当 INT7～INT4 的某一位被设置，且状态寄存器(SREG)中 I 位被设 1 时，通讯外部引脚中断使能。外部中断永远为低电平触发的中断. 这些引脚上的任何事件将触发中断请求，即使引脚被使能为输入。这提供了一种产生软件中断的方法。当为使能时，只要该引脚被拉低，一个电平触发的中断将产生一个中断请求。

· 外部中断标志寄存器——EIFR

位	7	6	5	4	3	2	1	0	
$38($58)	INTF7	INTF6	INTF5	INTF4	-	-	-	-	EIFR
读/写	R/W	R/W	R/W	R/W	R	R	R	R	
初始化值	0	0	0	0	0	0	0	0	

位7～4—INTF7～INTF4:外部中断7～4标志

当INT7～INT4引脚上的某一事件触发中断请求时,通讯中断标志INTF7～INTF4设置1。若SREG中I位及EIMSK中的通讯中断使能位INT7～INT4均设为1,MCU将跳到中断向量。当中断执行常规时,标志被清空。标志也可通过向这位写逻辑1来清除。

位3～0—Res:保留位

AT90SMEG103的该位为保留位,总读0。

·外部中断控制寄存器——EICR

位	7	6	5	4	3	2	1	0	
$3A($5A)	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

位7～0—ISCX1～ISCX0:外部中断7～4传感控制位

若SREG中的I位和EIMSK中的中断屏蔽被置位,则ISCX1～ISCX0决定外部中断的触发方式。

·定时器/计数器中断屏蔽寄存器——TIMSK

位	7	6	5	4	3	2	1	0	
$37($57)	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

位7—OCIE2:定时器/计数器2输出比较中断使能

当OCIE2被设为1,且状态寄存器中的I位被设为1时,定时器/计数器2比较A匹配中断使能。若发生了定时器/计数器2中的比较匹配,通讯中断(在向量$0012)被执行。定时器/计数器2中的比较标志在定时器/计数器中断标志寄存器TIFR中被设为1。

位6—TOIE2:定时器/计数器2溢出中断使能

当TOIE2被设为1,且状态寄存器中的I位被设为1时,定时器/计数器2溢出中断使能。若在定时器/计数器2上发生溢出,则通讯中断(在向量$0014)被执行。定时器/计数器2溢出标志在定时器/计数器的中断标志定时器TIFR中被设置为1。

位5—TICIE1:定时器/计数器1输入捕获中断使能

当TICIE1被设为1,且状态寄存器中的I位被设为1时,定时器/计数器1输入捕获事件中断使能。若在引脚29,即PD4(IC1)上发生捕获触发事件,则通讯中断(在向量$0016)被执行。定时器/计数器1中的输入捕获标志被在定时器/计数器中断标志寄存器TIFR中设置为1。

位4—OCIE1A:定时器/计数器1输出比较A匹配中断使能

当OCIE1A被设为1,且状态寄存器中的I位被设为1时,定时器/计数器比较A匹配中

断使能。若发生了定时器/计数器 1 中的比较 A 匹配,中断(在向量 $0018)被执行。定时器/计数器 1 中的比较标志在定时器/计数器中断标志寄存器 TIFR 中被设为 1。

位 3—OCIE1B:定时器/计数器 1 输出比较 B 匹配中断使能

当 OCIE1B 被设为 1,且状态寄存器中的 I 位被设为 1 时,定时器/计数器比较 B 匹配中断使能。若发生了定时器/计数器 1 中的比较 B 匹配,中断(在向量 $001A)被执行。定时器/计数器 1 中的比较标志在定时器/计数器中断标志寄存器 TIFR 中被设为 1。

位 2—TOIE1:定时器/计数器 1 溢出中断使能

当 TOIE1 被设为 1,且状态寄存器中 I 位被设为 1 时,定时器/计数器 1 溢出中断为使能。如定时器/计数器 1 产生溢出,相应的中断(在向量 $001C)被执行。定时器/计数器 1 溢出标志位在定时器/计数器中断标志位寄存器 TIFR 中被设为 1。当定时器/计数器 1 处于 PWM 模式下,计数器在 $0000 变化计数方向时,定时器溢出标志被设置。

位 1—OCIE0:定时器/计数器 1 输出比较 A 匹配中断使能

当 OCIE0 被设为 1,且状态寄存器中的 I 位被设为 1 时,定时器/计数器 0 比较匹配中断使能。若发生了定时器/计数器 0 中的比较匹配,中断(在向量 $001E)被执行。定时器/计数器 2 中的比较标志在定时器/计数器中断标志寄存器 TIFR 中被设为 1。

位 0—TOIE0:定时器/计数器 0 溢出中断使能

当 TOIE1 被设为 1,且状态寄存器中的 I 位被设为 1 时,定时器/计数器 0 溢出中断使能。若在定时器/计数器 0 上发生溢出,则通讯中断(在向量 $0020)被执行。定时器/计数器 0 溢出标志在定时器/计数器的中断标志定时器 TIFR 中被设置为 1。

· 定时器/计数器中断标志位寄存器——TIFR

位	7	6	5	4	3	2	1	0	
$36($56)	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

位 7—OCF2:输出比较标志 2

当定时器/计数器 2 和 OCR2 中的数据,即输出比较寄存器 2 之间的比较匹配发生时,OCF2 被设为 1。当执行通讯中断处理向量时,OCF2 由硬件清除。OCF2 亦可通过向标志写逻辑 1 来清除。当 SREG 中的 I 位,OCIE2(即定时器/计数器比较中断使能),以及 OCF2 被设为 1 时,执行定时器/计数器 2 输入比较中断。

位 6—TOV2:定时器/计数器 2 溢出标志

当定时器/计数器 2 中产生溢出时,TOV2 位被设为 1。当执行相应中断处理向量时,TOV2 被硬件复位。TOV2 可由写入一个逻辑 1 到标志位清除。当 SREG 的 I 位和 TOIE2(定时器/计数器 1 溢出中断使能),以及 TOV2 被设为 1 时,定时器/计数器 2 的溢出中断被执行。在 PWM 模式下,当定时器/计数器 1 在 $00 变化计数方向时,该位被设置。

位 5—ICF1:中断捕获标志 1

ICF1 位被设为 1,来标志一个输入捕获事件。表明定时器/计数器 1 的值以被输送到输入捕获寄存器 ICR1。当执行通讯中断处理向量时,ICF1 被由硬件清除。ICF1 亦可通过向标志写逻辑 1 来清除。

位4—OCF1A：输出比较标志1A

当定时器/计数器1与OCF1A中的数据，即输出比较寄存器1A之间发生比较匹配时，OCF1A被设为1。当执行通讯中断处理向量时，OCF1A由硬件清除。OCF1A亦可通过向标志写逻辑1来清除。当SREG中的I位，OCIE1A(即定时器/计数器1比较匹配中断使能)，以及OCF1A被设为1时，定时器/计数器1比较匹配中断使能被执行。

位3—OCF1B：输出比较标志1B

当定时器/计数器1与OCF1B中的数据，即输出比较寄存器1B之间发生比较匹配时，OCF1B被设为1。当执行通讯中断处理向量时，OCF1B由硬件清除。OCF1B亦可通过向标志写逻辑1来清除。当SREG中的I位，OCIE1B(即定时器/计数器1比较匹配中断使能)，以及OCF1B被设为1时，定时器/计数器1比较匹配中断使能被执行。

位2—TOV1：定时器/计数器1溢出标志位

当定时器/计数器1中产生溢出时，TOV1位被设为1。当执行相应中断处理向量时，TOV1被硬件复位。TOV1可由写入一个逻辑1到标志位清除。当SREG的I位和TOIE1(定时器/计数器1溢出中断使能)，以及TOV1被设为1时，定时器/计数器1溢出中断被执行。在PWM模式下，当定时器/计数器1在$0000改变计数方向时，该位被设置。

位1—OCF0：输出比较标志0

当定时器/计数器0和OCR0中的数据，即输出比较寄存器0之间的比较匹配发生时，OCF0被设为1。当执行通讯中断处理向量时，OCF0由硬件清除。OCF0亦可通过向标志写逻辑1来清除。当SREG中的I位，OCIE0(即定时器/计数器2比较中断使能)，以及OCF0被设为1时，执行定时器/计数器0输入比较中断。

位0—TOV0：定时器/计数器0溢出标志

当定时器/计数器0中产生溢出时，TOV0位被设为1。当执行相应中断处理向量时，TOV0被硬件复位。TOV0可由写入一个逻辑1到标志位清除。当SREG的I位和TOIE0(定时器/计数器0溢出中断使能)，以及TOV0被设为1时，定时器/计数器0的溢出中断被执行。在PWM模式下，当定时器/计数器1在$00变化计数方向时，该位被设置。

·中断应答时间

AVR所有允许的中断执行应答最少为4个时钟周期。在4个时钟周期之后，用于现行中断控制程序的程序向量寻址被执行。在4个时钟周期中，程序计数器(2个字节)被推到堆栈上，且堆栈指示器被减量2。该向量为一个向中断程序的相关转移，需3个时钟周期。若在一个多周期指令的执行中发生中断，该指令在中断执行前结束。

从中断处理程序的返回需要4个时钟周期。在这4个时钟周期之中，程序计数器(2个字节)被从堆栈中弹出，且堆栈指示器被增量2。当AVR从某一中断中出来时，它总是返回到主程序，并在任何挂起的中断执行前多执行一条指令。

注意：状态寄存器SREG不由AVR硬件处理，也不为中断或子程序工作。由于程序需要一个SREG的存储。这一切均由用户软件完成。

十一、休眠模式

为了进入休眠状态，MCUCR中的SE位被设为1，且须执行一条SLEEP指令。MCUCR寄存器中的SM1和SM0位选择休眠模式、闲置模式、掉电模式或省电模式，且由SLEEP指令触发。

当MCU在休眠模式下发生了一个允许的中断，MCU唤醒，执行中断程序，并恢复SLEEP后面指令的执行。寄存器文件的内容和I/O存储器不可改变。若在休眠模式下发生复位，MCU唤醒，并从复位向量执行。

注意：若为了从掉电状态下唤醒MCU而使用了某一电平触发的中断，必须保持长于晶振启动的时间。否则，在MCU开始执行以前，中断标志位有可能返回零值。

1. 闲置模式

当SM1/SM0位被清为00，SLEEP指令使MCU进入闲置状态，这时CPU停止工作，而定时器/计时器、看门狗以及中断系统继续工作。这使得CPU可以由外部触发的中断来唤醒，亦可由外部触发的中断，内部定时器溢出，及UART接收完成中断唤醒。若通过模拟比较器唤醒，需要一个中断，可通过设置模拟比较器和状态寄存器ACSR中的ACD位来对模拟比较器进行掉电。这就在闲置状态下减少了电的功耗。当MCU从闲置模式唤醒时，CPU立即开始执行程序。

2. 掉电模式

当SM1/SM0位为10，SLEEP指令强迫MCU进入掉电模式。在此模式下，内部晶振停止。用户可选择在掉电模式过程中看门狗是否被使能。若看门狗使能，当看门狗超过超时范围时，它唤醒MCU。若看门狗被禁止，只有外部复位或外部电平触发的中断可唤醒MCU。当向XTAL1加外部时钟源时，可完成从掉电模式的苏醒，这无须通常用来稳定XTAL晶振的延时。这种模式的使能是通过对FLASH中的SUT0/SUT1熔丝编程来完成的。

3. 省电模式

当SM1/SM0位为11，SLEEP指令使MCU进入省电模式。该模式同掉电模式，有一例外：若定时器/计数器0被同步锁存，如A/SSR中AS0位被设置，定时器/计数器0将在休眠状态中运行。器件可从定时器溢出唤醒，或从定时器/计数器0发出的输出比较中断唤醒。

4.6.3 定时器/计数器

AT90SMEG103有三个全程目的定时器/计数器，即2个8位T/C和一个16位T/C。定时器/计数器0作为选择，可与外部晶振异步。该晶振最好用32.768 kHz晶振，使得定时器/计数器0作为实时时钟(RTC)。定时器/计数器0带有自己的预分频器。定时器/计数器1和2带有从相同10位预分频定时器发生的单独的预分频选择。定时器/计数器可用作带内部时基或用作与外部引脚连接，可触发计数的计数器。

一、定时器/计数器预定比例器

对于定时器/计数器1和2，四个不同的预定比例部分为：CK/8、CK/64、CK/256和CK/1 024。CK为晶振时钟。对于定时器/计数器1和2，新加的部分为CK。外部源和停止均可作为CK的时钟源。

定时器/计数器0的时钟源为TCK0。TCK0通过系统设置与主系统时钟CK连接。通过设置TCCR0中A/SSR位，定时器/计数器0被从TOSC1引脚同步定时。这可使定时器/计数器0作为实时时钟(RTC)。一个晶振可以在TOSC1和TOSC2之间连接，作为定时器/计数器0的独立时钟源。这个晶振最好为32.768 kHz。

二、8位定时器/计数器T/C0和T/C2

8位定时器/计数器0可以选择从TCK0，或预分频的TCK0发出的时钟源。

8位定时器/计数器2可以选择从CK、预分频的CK,或外部引脚发出的时钟源。

两个定时器/计数器可按定时器/计数器控制寄存器TCCRO及TCCR2的说明停止。

不同的状态标志(溢出、比较匹配及捕获事件)在定时器/计数器中断标志寄存器TIFR中被找到。控制信号可在定时器/计数器控制寄存器TCCRO和TCCR2中找到。中断使能/禁止设置在定时器/计数器中断屏蔽寄存器TIMSK中找到。

当定时器/计数器2被外部定时时,外部信号以与CPU晶振相同的频率同步发生。为了能从外部时钟进行正确的取样,在两个外部时钟转换之间的最少时间必须维持一个内部CPU的时钟周期。外部时钟信号在内部CPU时钟的上升边沿被取样。

定时器/计数器使用输出比较寄存器COR0和COR2,作为将要与定时器/计数器内容相比较的数据源,支持2种输出比较功能。输出比较功能包括可选的在比较匹配时对计数器的清除,及输出比较引脚PB4(CO0/PWM0)和PB7(CO2/PWM2)在比较匹配时动作的清除。

定时器/计数器0和2也可用作为8位脉冲宽度调制器。在此模式下,定时器/计数器及输出比较寄存器作为无误操作、带中央脉冲的独特PWM。

1. 定时器/计数器控制寄存器——TCCR0

位	7	6	5	4	3	2	1	0	
$33($53)	-	PWM0	COM01	COM00	CTC0	CS02	CS01	CS00	TCCR0
读/写	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

2. 定时器/计数器控制寄存器——TCCR2

位	7	6	5	4	3	2	1	0	
$25($45)	-	PWM2	COM21	COM20	CTC2	CS22	CS21	CS20	TCCR2
读/写	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

位7—Res:保留位

AT90SMEG103的该位为保留位,总读0。

位6—PWM0/PWM2:脉冲宽度调制器使能

当设置为1时,这一位对定时器/计数器0或定时器/计数器2 PWM使能。

位5,4—COM01,COM00/COM21,COM20:比较输出模式,位1和位0

COMN1和COMN0控制位决定定时器/计数器中某一比较匹配之后的输引脚活动。输出引脚活动影响引脚PB4(OC0/PWM0)或PB7(OC2/PWM2)。由于这对I/O口来说是可选功能,相应的方向控制位必须设为1来控制输出引脚。

位3—CTC0/CTC2:清除比较匹配上的定时器/计数器

当CTC0和CTC2控制位设为1时,定时器/计数器在某一比较匹配之后,在CPU时钟周期中复位到$00。若控制位被清除,定时器/计数器继续计数,直到它换行(溢出)或改变方向。在PWM模式下,该位无效。

位2,1,0—CS02,CS01,CS00/CS22,CS21,CS20:时钟选择位2,1和0

时钟选择位2,位1和位0来决定定时器/计数器的预分频源。

3. 定时器/计数器0——TCNT0

位	7	6	5	4	3	2	1	0	
$22($42)	MSB							LSB	TCNT0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

4. 定时器/计数器 2——TCNT2

位	7	6	5	4	3	2	1	0	
$24($44)	MSB							LSB	TCNT2
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

这些 8 位寄存器包含定时器/计数器的值。

定时器/计数器是带读取访问的向上/向下大计数器(PWM 模式下)。若定时器/计数器被写入,同时时钟源正被执行,它在带写入值的预复位之后继续在定时器时钟周期中计数。

5. 定时器/计数器 0 输出比较寄存器——OCR0

位	7	6	5	4	3	2	1	0	
$31($51)	MSB							LSB	OCR0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

6. 定时器/计数器 2 输出比较寄存器——OCR2

位	7	6	5	4	3	2	1	0	
$23($43)	MSB							LSB	OCR2
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

输出比较器为一个 8 位的读/写寄存器。

定时器/计数器输出比较寄存器包括了将要连续地与定时器/计数器相比较的数据。比较匹配上的活动被在 TCCR0 和 TCCR2 中确定。对定时器/计数器或比较寄存器的软件写入使内容相等,将引起比较匹配。

当定时器/计数器 0 在异步模式下操作时,须注意,如 TCCR0 中的 AC0 位为 1 时,当写入 OCR0 时,值在写入操作之后,向 TCK0 时钟上的寄存器传递。

7. PWM 模式下的定时器 0/计数器 2

当选择 PWM 模式时,定时器/计数器以及输出比较寄存器 OCR0 和 OCR0 形成一个 8 位的自己运行、抗误操作,且在引脚 PD4(OC0/PWM0)和 PB7(OC2/PWM2)引脚上带有输出的节拍修正 PWM。定时器/计数器作为向上/向下的计数器,从 $00 向上计数到 $FF,在重复循环之前,它反转,并向下再次计数到 0。当计数器值与输出比较寄存器的内容相配时,PB4(OC0/PWM0)或 PB7(OC2/PWM2)引脚根据 COM01/COM00 的设置,或定时器/计数器控制寄存器 TCCR0 和 TCCR2 中 COM21/COM20 位的设置而被设置或被清除。

注意:在 PWM 模式下,当写入时,输出比较寄存器被送入临时地址。当定时器/计数器到达 $FF 时,它们被锁住。这就防止了在非同步 OCR0/OCR2 写入事件中发生奇数长的 PWM

脉冲(误操作)。

当 OCR2 包含 $00 或 $FF，输出 PB7(OC2/PWM2)根据 COM21 和 COM20 的设置被拉低或拉高。

在 PWM 模式下，定时器溢出标志 1、TOV0 和 TOV2 当计数器在方向 $00 时被设置。定时器溢出中断 0 和 2 以正常的定时器/计数器模式工作。比如，当 TOV0 和 TOV2 被设置，从而提供了定时器溢出中断 1 和通用中断为使能时，它被执行。这也同样使用于定时器输出比较 1 的标志和中断。PWM 的频率被定时器时钟频率除以 510。

8. 定时器/计数器 0 的异步操作

当定时器/计数器 0 同步操作时，所有操作及时序与定时器/计数器 2 的相同。在异步操作过程中，须注意以下事项：

(1) 在定时器/计数器 0 时序的同步与异步之间转换时，定时器值可能被破坏。

(2) 在定时器/计数器 0 时序的时钟信号频率必须比主时钟频率低 1/4。注意，若 XTAL 分频使能，CPU 时钟频率可能低于 XTAL 频率。

(3) 当向寄存器 TCNT0、OCR0 或 TCCR0 写入时，值被传入一个临时寄存器，且在 TOSC1 上 2 个正极沿之后被锁定。用户在临时寄存器中的内容被传输到目的地之前不可写入一个新值。以上提到的 3 个寄存器带有单独的临时寄存器，即向 TCNT0 的写入不干扰 OCR0 的写入进程。对目的地寄存器传输的检测需要空间，异步状态寄存器 ASSR 已提供。

(4) 在向 TCNT0、OCR0 或 TCCR0 写入之后，在进入休眠状态时，若使用定时器/计数器 0 来唤醒器件，用户须等到已写入的寄存器被升级之时。否则，在改变生效之前，MCU 进入休眠状态。这在使用输出比较 0 中断来唤醒器件时极为重要。在向 OCR0 或 TCNT0 写入过程中，输出比较被禁止。若(在 OCR0UB 位返回到 0 之前，用户进入休眠状态)写入循环未结束，器件将永远得不到一个比较匹配值，且 MCU 不醒。

(5) 若定时器/计数器 0 用来从省电模式唤醒器件，若用户想重新进入省电模式，须注意：中断逻辑需要一个 TOSC1 周期来复位。若在唤醒和重新进入省电模式的周期短于 TOSC1 周期，不发生中断，且器件将不唤醒。若用户担心在重新进入省电模式的时间是否充足，可使用下列算法来保证 TOSC1 周期已完成：

- 向 TCCR0、TCNT0 和 OCR0 写一个值。
- 在 ASSR 中通讯升级忙标志返回零之前等待。
- 进入省电模式。

除非在省电模式下之外，定时器/计数器 0 的 32 kHz 晶振总是运行。在加电复位或掉电唤醒之后，用户注意晶振可能需要 1 秒钟来稳定。建议用户在加电或从掉电状态下唤醒之后，至少等 1 秒钟再使用定时器/计数器 0。

9. 异步状态寄存器——ASSR

位	7	6	5	4	3	2	1	0	
$3C($5C)	-	-	-	-	AS0	TCN0UB	OCR0UB	TCR0UB	ASSR
读/写	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

位 7～4—Res：保留位

AT90SMEG103 的该位为保留位，总读 0。

位 3—AS0：异步定时器/计数器 0

当设为 1 时，定时器/计数器 0 从 TOSC1 引脚上定时。当清为零时，定时器/计数器 0 从内部系统时钟，CK 上定时。当这一位的内容改变时，TCNT0 的内容可能有误。

位 2—TCN0UB：定时器/计数器 0 修改忙

当定时器/计数器 0 异步操作，且 TCNT0 被写入，该位为 1。当向 TCNT0 写入的值从临时存储寄存器修改时，这位由硬件清空为 0。该位中的逻辑 0 表示 TCNT0 已准备好被新值修改。

位 1—OCR0UB：输出比较寄存器 0 修改忙

当定时器/计数器 0 异步操作，且 OCR0 被写入，该位为 1。当向 OCR0 写入的值从临时存储寄存器修改时，这位由硬件清空为 0。该位中逻辑 0 表示 OCR0 已准备好被新值修改。

位 0—TCR0UB：定时器/计数器控制寄存器 0 修改忙

当定时器/计数器 0 异步操作，且 TCCR0 被写入，该位为 1。当向 TCCR0 写入的值从临时存储寄存器修改时，这位由硬件清空为 0。该位中的逻辑 0 表示 TCCR0 已准备好被新值修改。

若在修改忙标志设为 1 的同时，在三个定时器/计数器 0 寄存器中的任何一个写入时，修改的值可能有误，且引发意外的中断。

当读取 TCNT0、OCR0 和 TCCR0 时，结果不同。当读 TCNT0 时，实际定时器值被读取。当读 OCR0 或 TCCR0 时，临时存储寄存器中的值被读取。

三、16 位定时器/计数器 1

16 位的定时器/计数器 1 可以从 CK、预定比例 CK，或外部引脚中选择时钟源。另外，它还可以像在定时器/计数器 1 控制寄存器 TCCR1B 中描述的那样被停止。在定时器/计数器控制寄存器 TCCR1A、TCCR1B 中可以找到不同的标志(溢出、比较匹配及捕获事件)和控制信号。对定时器/计数器 1 的使能/禁止设置，可以在定时器/计数器中断屏蔽寄存器 TIMSK 中找到。

当定时器/计数器被外部定时时，外部信号以 CPU 频率同步发生。为确保从外部时钟获取正确的取样，在 2 个外部时钟转换之间的最小时间至少为一个内部 CPU 时钟周期。外部时钟信号在内部 CPU 时钟的上升边沿被取样。

16 位的定时器/计数器 0 同时具有高分辨能力，并有以更低的预定比例进行高准确性运用的特性。类似地，高预定比例使得定时器/计数器 0 对低速功能，或带不连续操作的精确定时功能很有用。

计数器 1 支持运用了输入比较寄存器 1A 和 1B——OCR1A 和 OCR1B 的输出比较功能来作为将与定时器/计数器 1 内容进行比较的数据源。输出比较功能包括比较匹配的计数器可选清除，以及在比较匹配输出比较引脚上的动作。

定时器/计数器可被用作带调制器的 8 位、9 位或 10 位脉冲。在此模式下，定时器和 OCR1A/OCR1B 寄存器发挥带集中脉冲抗误操作独立 PWM 的功能。

定时器/计数器的输入捕获功能提供了对定时器/计数器 1 向输入捕获寄存器 ICR1 内容的捕获，该捕获操作由在输入捕获引脚 PD4/(IC1)上的外部事件激活。实际的捕获事件设置由定时器/计数器 1 的控制寄存器 TCCR1B 来定义。另外，模拟比较器可触发该输入捕获。请参照“模拟比较器”部分。

如果噪音清除器为使能，则捕获事件的实际触发条件在捕获被激活前受到多于 4 个取样

的监测。输入引脚信号以 XTAL 的时钟频率被取样。请参考第二章。

4.6.4 看门狗定时器

看门狗定时器由片内一个独立的 1 MHz 分开的晶振定时。通过控制看门狗定时器的预定比例器,看门狗的复位间歇从 16～2 048 ms 之间调整。WDR,即看门狗复位指令对看门狗定时器进行复位,如果超时且没有其他的看门狗复位,则 AT90SMEG103 被复位且从复位指针执行。当使能看门狗时,看门狗定时器的状态是不定的,建议用户在使能看门狗之前执行 WDR 指令,否则在使能之后的第一个 WDR 执行时器件可能复位。

为防止意外禁止看门狗,当看门狗被禁止时必须使用特定的关断过程。请参照看门狗定时控制寄存器。

4.6.5 EEPROM 读/写访问

可以访问 I/O 空间中 EEPROM 访问寄存器。写入访问时间在 2.5～4 ms 之间,取决于 V_{CC}电压。自定时功能使得用户软件检测下一个字节何时被写入。当 EEPROM 可以接受新的数据时,EEPROM 写完成中断可以通过设置来触发。

若 V_{CC}低于一定的电平,EEPROM 降低电压被检测,防止了对 EEPROM 的写入。为了防止对 EEPROM 的意外写入,必须服从一个特定的写入过程。当 EEPROM 被读取或写入时,在下一个指令执行前,CPU 被中止达两个时钟周期。

EEPROM 地址寄存器 EEARH 和 EEARL,指定了在 4K 字节的 EEPROM 空间中的 EEPROM 地址。EEPROM 数据字节被在 0～4 095 之间线性地编址。

一、EEPROM 地址寄存器——EEARH, EEARL

位	15	14	13	12	11	10	9	8	
$1F($3F)	-	-	-	-	EEAR11	EEAR10	EEAR9	EEAR8	EEARL
	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARH
	7	6	5	4	3	2	1	0	
读/写	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

二、EEPROM 数据寄存器——EEDR

位	7	6	5	4	3	2	1	0	
$1D($3D)	MSB							LSB	EEDR
读/写	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

位 7～0—EEDR7～0: EEPROM 数据

对于 EEPROM 写入操作,EEDR 寄存器包含了写入 EEPROM 的数据,由 EEAR 寄存器给出其地址。对于 EEPROM 的读取操作,EEDR 包含由 EEAR 给出的 EEPROM 地址,数据将从这一地址中读出。

三、EEPROM 控制寄存器——EECR

位	7	6	5	4	3	2	1	0	
$3C($5C)	–	–	–	–	EERIE	EEMWE	EEWE	EERE	EECR
读/写	R	R	R	R	R	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

位 7～4—Res：保留位

AT90SMEG103 的该位为保留位，总读 0。

位 3—EERIE：EEPROM 准备中断允许

当 SREG 和 EERIE 寄存器中的 1 位被设置，EEPROM 准备中断被使能。当被清除时，该中断被禁止。EEPROM 准备中断常在 EEWE 被清除时生成中断请求。

位 2—EEMWE：EEPROM 的主写使能

EEMWE 位决定了设置 EEWE 是否导致 EERPOM 被写入。当 EEMWE 被设为 1 时，设置 EEWE 将把数据写入 EEPROM 所选择的地址中。如果 EEMWE 为 0，则设置 EEWE 没有反应。当 EEMWE 被软件设置后，4 个周期后被硬件清除。详见 EEPROM 的写过程中 EEWE 位的描述。

位 1—EEWE：EEPROM 写入使能

EEPROM 写入使能信号 EEWE 是对 EEPROM 的写入选通。若地址和数据被正确设置，EEWE 位必须被设置从而写 EEPROM。当 EEWE 被置 1 时，EEMWE 必须被置 1，否则不会发生 EEPROM 的写操作。当写入 EERPOM 时应服从以下的过程(第(2)步和第(3)步不是必须的)：

(1) 等待 EEWE 位变为 0；

(2) 把新的 EEPROM 地址写入 EEAR(可选)；

(3) 把新的 EEPROM 数据写入 EEDR(可选)；

(4) 在 EECR 中的 EEMWE 位写逻辑 1；

(5) 在设置 EEMWE 后的 4 个时钟周期内，在 EEWE 中写入逻辑 1。

在写入访问时间(一般为 5 V 时 2.5 ms，2.7 V 时 4 ms)过后，EEWE 由硬件清空。用户软件在写入下一个字节之前，查询这一位，并等待零值。当 EEWE 被设过后，CPU 在执行下一个指令前中止 2 个周期。

位 0—EERE：EEPROM 读取使能

EEPROM 读取使能信号 EERE 是对 EEPROM 的读取选通。当 EEAR 寄存器中的地址被正确设置时，EERE 必须被设好。当 EEAR 被硬件清空时，在 EEDR 寄存器中可找到所需数据。EEPROM 读取访问占用 1 个指令，无需查询 EERE 位。一旦 EERE 被设过后，CPU 在执行下一个指令前中止 2 个周期。

4.6.6 串行外设接口 SPI

串行外设接口(SPI)允许在 AT90SMEG103 和外设，或几个 AT90SMEG103 之间高速同步数据传送。AT90SMEG103 SPI 的特征同于 AT90S4414 SPI，见 4.3.6 节。

4.6.7 通用串行接口 UART

AT90SMEG103 特性为一个全双工通用异步接收器和传输器(UART)。主要特性为：

- 产生任何波特率的波特率发生器。
- 低 XTAL 频率的高波特率。
- 8 位或 9 位数据。
- 噪音过滤。
- 误差检测。
- 帧错误检测。
- 错误起始位检测。
- TX 完成、TX 数据寄存器为空及 RX 完成时，有三种单独的中断。

一、数据传送

数据传输通过将要向 UART I/O 数据寄存器 UDR 传输的数进行写操作而触发。当下列情况发生时，数据从 UDR 向传输移位寄存器传输：

· 在停止位从前一个字符被移出之后，一个新字符向 UDR 写入。移位寄存器立即被装入。

· 在停止位从前一个字符被移出之前，一个新字符向 UDR 写入。当正被传输的字符的停止位被移出时，移位寄存器被装入。

若 10(11)位传输器寄存器为空，数据从 UDR 向移位寄存器传输，此时在 UART 中的 UDRE位 USR 被设置。当该位设置为 1 时，UART 准备接收下一个字符。向 UDR 的写入操作清除 UDRE。同时，由于数据从 UDR 向 10(11)位移位寄存器传输，移位寄存器的 0 位被清为 0，且位 9 或位 10 被设置(停止位)。若选择 9 位数据字(UART 控制寄存器中的 CHR9 位，UCR 被设置)，UCR 中的 TXB，位 8 被向传输移位寄存器中的位 9 传输。

在向移位寄存器传输操作之后的波特率时钟上，开始位从 TXD 引脚上移出。LSB 之后是数据。当停止位被移出，在传输过程中，若有任何新数据被向 UDR 写入，移位寄存器被装入。装入过程中，UDRE 被设置。若当停止位被移出时，UDR 寄存器中没有新的数据发送，UDRE 标志将保留设置。在此条件下，在停止位被提供在 TXD 上一位长之后，TX 完成标志和 TXC 位在 USR 中设置。

在 CUR 中的 TXEN 位设 1 时，UART 传输使能。通过清除该位(0)，PE1 引脚可用作通用 I/O。当 TXEN 被设置，UART 传输器将与 PE1 引脚连接，不管 DDER 中 DDE1 位的设置。

二、数据接收

接收器前端逻辑以波特率 16 倍的频率向 RXD 引脚上的信号取样。当线闲置时，逻辑 0 的一个信号取样，将被解释为开始位的下降沿，且起始位检测序列被激活。让取样 1 给第一个 0 取样。1 到 0 转换之后，接收器以取样 8、9 和 10 对 RXD 引脚取样。若 2 个或多于此 3 个的取样为逻辑 1，启动位发出噪音以示拒绝，接收器开始搜寻下一个 1 到 0 的转换。

然而，若某一有效的开始位被检测到，起始位之后的数据位的取样完成。这些位同样为取样 8、9 和 10。在 3 个取样中至少 2 个的逻辑值作为位值被取样。由于是取样，所有的位移位进转换移位寄存器。

当停止位进入接收器，停止位中 3 个取样中多数必为 1。若 2 个或更多的取样为逻辑 0，UART 状态寄存器(USR)中的帧错误(FE)标志，在收到被转换到 UDR 的字节时被设置。在读取 UDR 寄存器之前，用户需先检测 FE 位，来检测帧错误。当 UDR 被设置时，FE 被清除。

无论字符接收循环结束时的某一有效停止位是否被检测到，数据被传输到 UDR，且 USR 中的标志被设置。UDR 实际为 2 个物理上分开的寄存器，一个用来传输数据，另一个用来接

收数据。当 UDR 被读取时,接收数据寄存器被访问到;当 UDR 被写入时,传输数据寄存器被访问。若选择 9 位数据字(UART 控制寄存器中的 CHR9,UCR 已设置),当数据传输到 UDR 时,UCR 中的 RXB 位 8 被传输移位寄存器中的位 9 装入。

若在收到字符之后,UDR 寄存器从上次接收后没有被访问到,超越误差(OR)标志被设置。这意味着将要传输到移位寄存器的数据不能被传输到 UDR,丢失了。OR 位被缓存,当 UDR 中的有效数据被读出时 OR 位可用。用户总需要在从 UDR 寄存器读取后检测 OR,以便检测任何超越误差。

通过清除 UCR 寄存器中的 RXEN 位,接收器被禁止。

三、UART 控制

(1) UART I/O 数据寄存器——UDR:UDR 寄存器实际上为物理上分开的两个寄存器,共有相同的 I/O 地址。当使寄存器写入时,UART 传输数据寄存器被写入。当从 CPU 读取时,UART 接收数据寄存器被读取。

(2) UART 状态寄存器——USR:USR 寄存器为一个提供 UART 状态的只读寄存器。

(3) UART 控制寄存器——UCR:UCR 寄存器是一个可读写的控制 UART 的寄存器。

(4) 波特率发生器:波特率发生器是依据以下等式的分频器来产生波特率:

$$BAUD = FCK/[16(UBRR + 1)]$$

其中:BAUD 为波特率;FCK 为晶振时钟频率;UBRR 为 UAUD 波特率发生器的内容,UBRR 为 0~255。

(5) UART 波特率寄存器——UBRR:UBRR 寄存器为一个 8 位读取/写入寄存器。

4.6.8 模拟比较器

模拟比较器对正极 PB2 引脚(AC+)和负极 PB3 引脚(AC-)之上的输入值进行比较。当 PB2(AC+)的电压高于 PB3 的电压时,模拟比较器输出 ACO 被设为 1。比较器的输出可以置位定时器/计数器 1 输入捕获功能的触发器,此外,比较器的输出可以被设置为触发一个独立于模拟比较器的中断。用户可以选择比较器输出的上升、下降沿触发中断或触发器。

一、模拟比较器控制和状态寄存器——ACSR

有关内容参考 2.10.2 节内容。

二、模拟到数字的转换器

特性表:

- 10 位运算。
- ±1 LSB 精确。
- 50 ~200 μs 转换时间。
- 8 个多功能输入通道。
- 自由运行或单一转换模式。
- ADC 完成的中断。
- 休眠模式噪音清除器。

AT90SMEG103 为一个 10 位连续近似值 ADC。ADC 与一个 8 通道的模拟多路转换器相连,该器件 F 口的每一引脚作为 ADC 的输入。ADC 包含一个取样和保留放大器,它们使得到 ADC 的输入电压在转换过程中保持一个连续的电平。ADC 的方框图如图 4.26 所示。

ADC有两个单独分开的模拟供电引脚，AV_{CC}和 AGND。AGND 必须与 GND 相连，且 AV_{CC}的电压需与 V_{CC}的电压保持 +0.4 V 的不同。

一个外部的参考电压须加到 AREF 引脚上。该电压范围为 AGND ～ AV_{CC}。

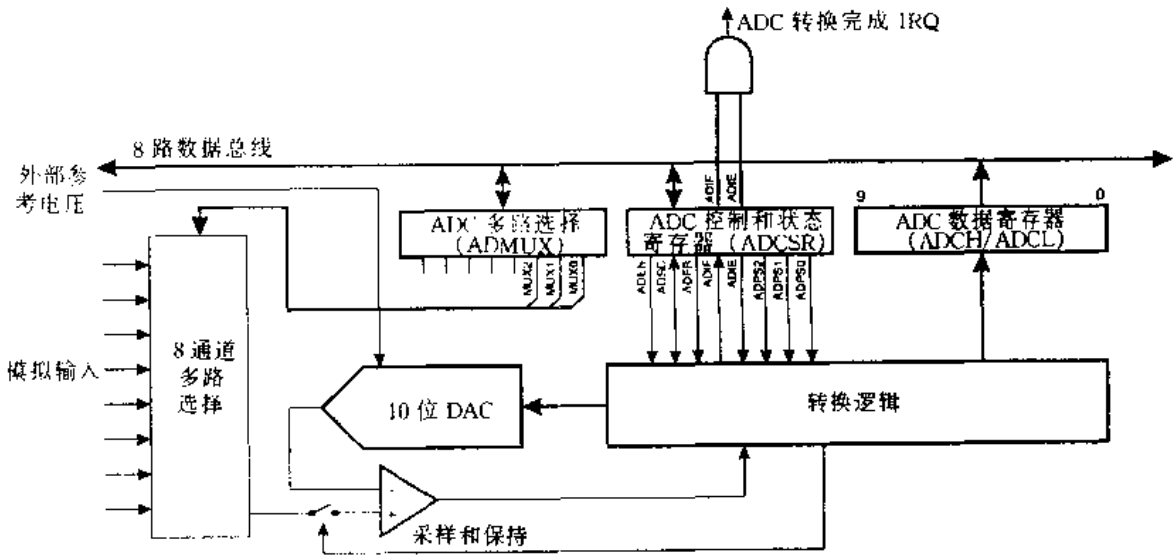


图 4.26　ADC 的方框图

三、操　作

ADC 以两种模式操作，单一转换和自由运行模式。在单一转换模式下，每一转换须由用户触发。在自由运行模式下，ADC 连续取样，且对 ADC 数据寄存器修改。ADCSR 中的ADFR 位选择这两种模式。

通过向 ADC 使能位写入逻辑 1，ADC 被允许，ADEN 在 ADCSR 中。

转换的开始是通过向 ADC 开始转换位 ADSC 写入逻辑 1。在转换进行中，该位保持高电平，且当转换结束时通过硬件设成 0。

若在转换过程中，选择不同的数据通道，在完成通道变化之前 ADC 将停止当前的转换。

由于 ADC 产生一个 10 位的结果，数据寄存器，ADCH 和 ADCL 须被读取，以便在转换完成时获取数据。特殊数据保护逻辑用来当被读取时，数据寄存器的内容拥有同样的结果。其机械工作如下：读数据时，须先读 ADCH；一旦 ADCH 被读取，ADC 对数据寄存器的访问被锁住。即如果 ADCH 被读，且在 ADCL 被读之前转换已结束，寄存器不修改，且转换结果丢失。当 ADCL 被读时，ADC 对 ADCH 和 ADCL 寄存器的访问被重新允许。

ADC 有自己的中断，它在转换完成时被激活。ADC 在当 ADCH 和 ADCL 读取之间，对数据寄存器的访问被禁止时，即使结果丢失，中断将触发。

四、预分频

ADC 接收在 50～200 kHz 之间的时钟频率。ADC 需 10 个时钟脉冲来完成转换，即需要 50～200 μs。若输入时钟超出范围，ADC 的输入不保证正确。ADC 预分频器 ADCPS 保证 XTAL 频率超过 50 kHz 时，ADC 时钟输入频率正确。

五、ADC 噪音清除功能

ADC 有一个噪音清除器,它使得在闲置模式转换过程中减少包括从 CPU 核出来的噪音。使用下列步骤来使用这一特性:

(1) 请确保 ADC 使能,且转换不忙。须选单一转换模式,且 ADC 转换完成中断使能。

ADEN = 1 ; ADSC = 0 ; ADFR = 0 ; ADIE = 1。

(2) 进入休闲模式。一旦 CPU 被挂起,ADC 将开始转换。

(3) 若在 ADC 转换完成之前发生另一中断,ADC 中断将唤醒 CPU,并执行 ADC 转换完成中断工作。

六、ADC 多路转换器选择寄存器——ADMUX

位	7	6	5	4	3	2	1	0	
$07($27)	-	-	-	-	-	MUX2	MUX1	MUX0	ADMUX
读/写	R	R	R	R	R	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

位 7~3—Res:保留位

AT90SMEG103 的该位为保留位,总读 0。

位 2~0—MUX2~MUX0:模拟通道选择位 2~0

这三种的值选择 7~0 的哪一个与 ADC 相连。

七、ADC 控制和状态寄存器——ADCSR

位	7	6	5	4	3	2	1	0	
$06($26)	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

位 7—ADEN:ADC 使能

对这一位写 1,对 ADC 使能。通过向该位写 0,关闭 ADC。

位 6—ADSC:ADC 开始转换

在单一转换模式下,为开始每一转换,须向该位写 1。转换过程中,该位被读为 1。转换结束时,返回 0。在自由运行模式下,该位保持 1。在过程中写入逻辑 0 将停止自由运行操作。在单一转换模式下,对该位写 0 无意义。

位 5—ADFR:ADC 自由运行选择

在自由运行模式下,当该位设 1 时,ADC 工作。在此模式下,ADC 取样,且将数据寄存器连续更新。

位 4—ADIF:ADC 中断标志

当某一 ADC 转换完成,且数据寄存器被修改时,该位设为 1。ADC 转换完成中断在 ADCIE 位和 SREG 中 I 位被设为 1 时执行。在执行通讯中断处理向量时,ADIF 由硬件清除。ADIF 也可通过向这一标志写入逻辑 1 来清除。注意,若在 ADCSR 上执行读—写—读的操作,一个暂挂中断被禁止。这对于使用 SBI 和 CBI 指令时也适用。

位 3—ADIE:ADC 中断使能

当这一位设为 1,且 SREG 中的 I 位设为 1 时,ADC 转换完成中断被激活。

位 2～0—ADPS2～ADPS0：ADC 预分频器选择位

这一位决定在 XTAL 频率和对 ADC 的输入时钟之间的因子分裂。

表 4.21 为 ADC 比例因子选择表。

表 4.21　ADC 比例因子选择

ADPS2	ADPS1	ADPS0	部分系数	ADPS2	ADPS1	ADPS0	部分系数
0	0	0	非法	1	0	0	16
0	0	1	2	1	0	1	32
0	1	0	4	1	1	0	64
0	1	1	8	1	1	1	128

八、ADC 数据寄存器——ADCL 和 ADCH

位	15	14	13	12	11	10	9	8	
$2D($4D)	-	-	-	-	-	-	ADC9	ADC8	ADCH
$2D($4D)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
读/写	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
初始化值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

当 ADC 转换完成时，结果存于这两个寄存器中。在自由运行模式下，需要 2 个寄存器都被读取，且 ADCH 在 ADCL 之前。

九、扫描多通道

由于模拟通道的变换总是延迟，直到转换完成，自由运行模式可用来扫描多通道，而不干扰转换器。ADC 转换完成中断通常用来完成通道的移位。用户需注意：一旦结果将要读出，中断发生。在自由运行模式下，在中断发生之后一个 ADC 时钟周期后，下一转换开始。若在中断发生之后一个 ADC 时钟周期内 ADMUX 内容改变，ADMUX 设置将适用于将开始的转换。若 ADMUX 的改变在中断发生的一个时钟周期之后，则下一转换已开始且使用旧的设置。

十、ADC 噪音消除技术

AT90SMEG103 内部和外部的数字电路产生 EMI，EMI 可能影响模拟测量的精确度。若转换精确值临界，通过使用下列技术可以减少噪音级：

(1) AT90SMEG103 的模拟部分和所有应用中的模拟部件在 PCB 上带有单独的模拟地平。该地平通过 PCB 上的一个单点与数字地平连接。

(2) 模拟信号尽可能短。注意模拟轨道在模拟地平上运行，且与高速开关数字轨道分开。

(3) AT90SMEG103 上的 AV_{CC}引脚须通过 RC 网络与数字 V_{CC}供电电压相连。

(4) 用 ADC 噪音消除功能减少包括 CPU 的噪音。

(5) 若一些 PORTF 用作数字输入，在转换过程中很重要的一点是这些不转换。

4.6.9　I/O 口

一、A 口特性

A 口为一个 8 位的双向 I/O 口。A 口分配有 3 个数据存储地址，分别为数据寄存器 POR-

TA $1B($3B)、数据方向寄存器 DDRA $1A($3A)、和 B 口的输入引脚 PINA $19($39)。A 口的输入引脚地址为只读,而数据寄存器和数据方向寄存器为可读写。

所有的 A 口引脚都有独立可选的上拉,A 口输出缓存可以吸收 40 mA 的电流以直接驱动 LED 显示。当 PA0~PA7 引脚被用作输入,且被外部拉低时,若内部拉高被激活,这些引脚将成为电流源(I_{IL})。

A 口引脚具有与可选的外部数据 SRAM 有关的第二功能,PORTA 在访问外部数据存储器时可以配置为复用的低位地址/数据线。

当通过 MCU 控制寄存器——MCUCR 的 SRE——外部 SRAM 使能位把 PORTA 设置为第二功能,更改的设置会覆盖数据方向存储器。

A 口的输入引脚地址 PINA 不是一个寄存器,该地址允许对 A 口的每一个引脚进行存取。当读 A 口时,读到的是 A 口的数据锁存器;当读 PINA 时,引脚上的逻辑值被读入。

二、B 口特性

B 口为一个带内部拉高的 8 位双向 I/O 口。B 口分配有 3 个数据存储地址,分别为数据寄存器 PORTB $18($38)、数据方向寄存器 DDRB $17($37)和 B 口的输出引脚 $16($36)。B 口的输入引脚地址为只读,而数据寄存器和数据方向寄存器为可读写。

所有的口线引脚均有单独的可选择拉高。B 口输出缓存可以吸收 40 mA 的电流以直接驱动 LED 显示。当 PB0 到 PB7 引脚被用作输入,且被外部拉低时,若内部拉高被激活,这些引脚将成为电流源(I_{IL})。具有可选择功能的 B 口引脚如表 4.22 所示。

表 4.22 B 口引脚选择功能

口引脚	第二功能
PB0	$\overline{SS}$(SPI 从选择输入)
PB1	SCK (SPI 总线串行时钟)
PB2	MOSI (SPI 总线主输出/从输入)
PB3	MOSO (SPI 总线主输出/从输入)
PB4	OC0/PWM0 (输出比较和 T/C0 的 PWM 输出)
PB5	OC1A/PWM1A (输出比较和 T/C1 的 PWM 输出 A)
PB6	OC1B/PWM1B (输出比较和 T/C1 的 PWM 输出 B)
PB7	OC2/PWM2 (输出比较和 T/C2 的 PWM 输出)

当引脚被用作可选择的功能时,DDRB 和 PORTB 寄存器必须根据可供选择的功能说明来设置。

B 口的可选择性功能如下:

OC2/PWM2,位 7:

OC2/PWM2,定时器/计数器 2 的输出比较输出,或在 PWM 模式下定时器/计数器 2 的 PWM 输出。此功能下该引脚必须被配置为输出。

OC1B/PWM1B,位 6:

OC1B/PWM1B,为定时器/计数器 1 的输出比较输出 B,或在 PWM 模式下时的定时器/计数器 1PWM 输出 B。此功能下,该引脚必须配置为输出。

OC1A/PWM1A,位 5:

OC1A/PWM1A,定时器/计数器 1 的输出比较输出 A,或在 PWM 模式下的 PWM 输出 A。此功能下,该引脚必须配置为输出。

OC0/PWM0,位 4:

OC0/PWM0,定时器/计数器 0 的输出比较输出,或在 PWM 模式下的定时器/计数器 0PWM 输出。

MISO—口 B,位 3:

MISO 为主机数据输入,SPI 通道的附属数据输出引脚。

MISI—口 B,位 2:

MOSI 为 SPI 主数据输出,SPI 通道的附属数据输入。

SCK—口 B,位 1:

SCK 为主时钟输出,SPI 通道的附属时钟输入。

$\overline{SS}$—口 B,位 0:

附属口选择输入。

三、C 口特性

C 口为一个 8 位的输出口。

PORTC 数据寄存器——PORTC:PORTC 引脚具有与可选的外部数据 SRAM 有关的第二功能,。当与外部 SRAM 一起使用时,在访问外部存储器过程中 C 口输出高序地址字节。

四、D 口特性

D 口是一个带内部上拉的 8 位双向 I/O 口。D 口占了 3 个数据存储器的地址,一个是数据寄存器 PORTD $12($32)、数据方向寄存器 DDRD $11($31)和端口 D 输入引脚 PIND $10($30)。D 口的引脚地址是只读的,而数据寄存器和数据方向寄存器可以读/写。

D 口的输出缓冲器可以吸入 40 mA,D 口的引脚在激活内部上拉时如果外部被拉低就会输出电流(I_{IL})。某些 D 口的引脚有第二功能,如表 4.23 所示。

表 4.23 D 口引脚选择功能

口引脚	第二功能	口引脚	第二功能
PD0	INT0(外部中断 0 输入)	PD5	IC1(T/C1 输入捕获触发)
PD1	INT1(外部中断 1 输入)	PD6	T1(T/C1 时钟输入)
PD2	INT2(外部中断 2 输入)	PD7	T2(T/C2 时钟输入)
PD3	INT3(外部中断 3 输入)		

当 D 口的引脚被用于第二功能时,DDRD 和 PORTD 寄存器必须按照第二功能来设置。D 口的输入引脚地址 PIND 不是一个寄存器,该地址允许对端口 D 的每一个引脚进行存取。当读 PORTD 时,读到的是 PORTD 的数据锁存器,当读 PIND 时,引脚上的逻辑值被读入。

五、E 口特性

E 口为一带内部拉高的 8 位双向 I/O 口。E 口占了 3 个数据存储器的地址,一个是数据寄存器 PORTE $03($23)、数据方向寄存器 DDRE $02($22)和 E 口输入引脚 PINE $01($21)。E 口的引脚地址是只读的,而数据寄存器和数据方向寄存器可以读/写。

E 口的输出缓冲器可以吸入 40 mA,E 口的引脚在激活内部上拉时如果外部被拉低就会输出电流(I_{IL})。某些 E 口的引脚有第二功能,如表 4.24 所示。

表 4.24 E 口引脚选择功能

口引脚	第二功能
PE0	PDI/RXD (可编程数据输入或 UART 接收脚)
PE1	PDO/TXD (可编程数据输出或 UART 发送脚)
PE2	AC+ (模拟比较器正输入)
PE3	AC- (模拟比较器负输入)
PE4	INT4 (外部中断 4 输入)
PE5	INT5 (外部中断 5 输入)
PE6	INT6 (外部中断 6 输入)
PE7	INT7 (外部中断 7 输入)

当 E 口的引脚被用于第二功能时,DDRE 和 PORTE 寄存器必须按照第二功能来设置。E 口的输入引脚地址 PINE 不是一个寄存器,该地址允许对 E 口的每一个引脚进行存取。当读 PORTE 时,读到的是 PORTE 的数据锁存器;当读 PINE 时,引脚上的逻辑值被读入。

六、F 口特性

F 口为一个 8 位输入口。F 口的一个 I/O 存储器地址被分配,口输入引脚 PINE $00 ($20)。

F 口所有与 A/D 转换器相连的模拟多功能器连接。F 口的数字输入功能可被用作 A/D 转换功器,使得用户可以同时使用 F 口的一些引脚、数字输入,以及模拟输入。

有关存储器编程的内容请参考第二章。

第五章　实用程序设计

5.1　程序设计方法

程序设计就是用计算机所能接受的语言把解决问题的步骤描述出来，也就是编制计算机的程序。AVR 单片机程序设计语言有：C 编译高级语言和宏汇编汇编语言。

在设计应用系统时，软件的编制是重要环节。软件的质量直接影响整个系统功能的实现。本章从应用的角度出发，介绍一些实用子程序，读者既可按需要改编调用，也可以吸收其设计方法，以便更好地设计出适合于自己系统的实用软件。

5.1.1 程序设计步骤

应用程序的设计因系统而异，因人而异。尽管如此，程序设计还是有共同特点及其规律的。在编写程序时，设计人员可以采取如下几个步骤：

(1) 分析问题，明确所要解决问题的要求。将软件分成若干个相对独立的部分。根据功能关系和时序关系，设计出合理的软件总体结构。

(2) 建立正确的数学模型。即根据功能要求，描述出各个输入和输出变量之间的数学关系，并确定采用的计算公式和计算方法。

(3) 制定程序框图。根据所选择的计算方法，制定出运算的步骤和顺序，并画出程序框图。这不仅是程序设计的一个重要组成部分，而且是决定成败的关键部分。

(4) 合理分配系统资源，包括程序 Flash、EEPROM、SRAM、定时器/计数器、中源、堆栈等。确定数据格式，分配好工作单元，进一步将程序框图画成详细的操作流程。

(5) 根据程序的流程图和指令系统，编写出程序。注意在程序的有关位置处写上功能注释，提高程序的可读性。

(6) 程序调试。通过编辑软件编辑出的源程序，必须用编译程序汇编后生成目标代码。

如果源程序有语法错误，需修改源文件后继续编译，直到无语法错误为止。这之后利用目标码通过仿真器进行程序调试，排除设计和编程中的错误，直到成功。

(7) 程序优化。使各功能程序实行模块化、子程序化，缩短程序的长度，加快运算速度和节省数据存储空间，减少程序执行的时间。

5.1.2　程序设计技术

1. 模块化程序设计

模块化程序设计是单片机应用中常用的一种程序设计技术。它是把有关功能完整的、较长的程序，分解为若干个功能相对独立的、较小的程序模块，各个程序模块分别进行设计、编程和调试，最后把各功能模块集成为所需的程序。

模块化程序设计的优点是，单个功能明确的程序模块的设计和调试比较方便、容易完成。一个模块可以为多个程序所共享，也可利用现成的程序模块。

2. 自上而下的程序设计

自上而下的程序设计时，先从主程序开始设计，从属的程序和子程序用符号来代替。主程

序编好后，再编制各个从属程序和子程序，最后完成整个系统软件的设计。调试也按这个次序进行。

自上而下程序设计的优点是，比较习惯人们的日常思维，设计、调试和连接同时按一个线索进行，程序错误可以较早发现。缺点是修改比较麻烦。

3. 软件抗干扰设计

用于生产现场的单片机应用系统，易受各种干扰侵袭，直接影响到系统的可靠性。因此，应用系统的抗干扰设计是非常重要的。

在实际情况中，针对不同的干扰后果，采用不同的软件对策。在实时数据采集系统中，为了消除传感器通道中的干扰信号，可采用软件数据滤波，如算术平均法、比较舍取法、中值法、一阶递推数字滤波法等；在开关量控制系统中，为防止干扰进入系统，造成各种控制条件超差、输出失控，可采取软件冗余、程序自检等措施；为防止程序计数器失控，造成程序盲目运行或"死机"，可设置软件"看门狗"来监视程序运行状态，也可在非程序区设置软件陷井，强行使程序拉回复位状态，重新启动。

5.2 应用程序举例

应用程序中包括一些算术运算、代码转换、数据传送、滤波算法、串行通信、A/D 转换等子程序。根据需要，可以应用其中的一些子程序。

5.2.1 内部寄存器和位定义文件

AVR 单片机内部寄存器和位定义文件，用于定义器件内部的寄存器名和寄存器的位名。在汇编程序文件中如包括了定义文件，则所有数据块中 I/O 寄存器名和 I/O 寄存器位名都能使用。寄存器名用 16 进制地址表示，寄存器位名用 0～7 位表示。

另外，被配置命名的 XL～ZH 6 个寄存器形成 3 个数据指针 X、Y 和 Z，作为内部 SRAM 高端 RAM 地址同样被定义。

注意，在指令中使用的位名是意义不同的。如"sbr"/"cbr"指令表示，若位被置位/清除则跳行执行。

一、AT90S1200 单片机内部寄存器和位定义文件

```
;***** I/O寄存器定义
    .equ SREG       = $3f
    .equ GIMSK      = $3b
    .equ TIMSK      = $39
    .equ TIFR       = $38
    .equ MCUCR      = $35
    .equ TCCR0      = $33
    .equ TCNT0      = $32
    .equ WDTCR      = $21
    .equ EEAR       = $1e
    .equ EEDR       = $1d
    .equ EECR       = $1c
    .equ PORTB      = $18
    .equ DDRB       = $17
    .equ PINB       = $16
    .equ PORTD      = $12
    .equ DDRD       = $11
    .equ PIND       = $10
    .equ ACSR       = $08
;***** 位定义
    .equ INT0       =6
    .equ TOIE0      =1
    .equ TOV0       =1
    .equ SE         =5
    .equ SM         =4
    .equ ISC01      =1
    .equ ISC00      =0
    .equ CS02       =2
    .equ CS01       =1
    .equ CS00       =0
    .equ WDE        =3
    .equ WDP2       =2
    .equ WDP1       =1
    .equ WDP0       =0
```

```
.equ EEWE       =1
.equ EERE       =0
.equ PB7        =7
.equ PB6        =6
.equ PB5        =5
.equ PB4        =4
.equ PB3        =3
.equ PB2        =2
.equ PB1        =1
.equ PB0        =0
.equ DDB7       =7
.equ DDB6       =6
.equ DDB5       =5
.equ DDB4       =4
.equ DDB3       =3
.equ DDB2       =2
.equ DDB1       =1
.equ DDB0       =0
.equ PINB7      =7
.equ PINB6      =6
.equ PINB5      =5
.equ PINB4      =4
.equ PINB3      =3
.equ PINB2      =2
.equ PINB1      =1
.equ PINB0      =0
.equ PD6        =6
.equ PD5        =5
.equ PD4        =4
.equ PD3        =3
.equ PD2        =2
.equ PD1        =1
.equ PD0        =0
.equ DDD6       =6
.equ DDD5       =5
.equ DDD4       =4
.equ DDD3       =3
.equ DDD2       =2
.equ DDD1       =1
.equ DDD0       =0
.equ PIND6      =6
.equ PIND5      =5
.equ PIND4      =4
.equ PIND3      =3
.equ PIND2      =2
.equ PIND1      =1
.equ PIND0      =0
.equ ACD        =7
.equ ACO        =5
.equ ACI        =4
.equ ACIE       =3
.equ ACIS1      =1
.equ ACIS0      =0
.equ INT0addr= $001 ;外部中断0向量地址
.equ OVF0addr= $002 ;溢出0中断向量地址
.equ ACIaddr= $003  ;模拟比较中断向量地址
.def ZL         =r30
```

二、AT90S2313单片机内部寄存器和位定义文件

```
;***** I/O寄存器定义
.equ SREG       = $3f
.equ SPL        = $3d
.equ GIMSK      = $3b
.equ GIFR       = $3a
.equ TIMSK      = $39
.equ TIFR       = $38
.equ MCUCR      = $35
.equ MCUSR      = $34
.equ TCCR0      = $33
.equ TCNT0      = $32
.equ TCCR1A     = $2f
.equ TCCR1B     = $2e
.equ TCNT1H     = $2d
.equ TCNT1L     = $2c
.equ OCR1AH     = $2b
.equ OCR1AL     = $2a
.equ ICR1H      = $25
.equ ICR1L      = $24
.equ WDTCR      = $21
.equ EEAR       = $1e
.equ EEARL      = $1e
.equ EEDR       = $1d
.equ EECR       = $1c
.equ PORTB      = $18
.equ DDRB       = $17
.equ PINB       = $16
.equ PORTD      = $12
.equ DDRD       = $11
.equ PIND       = $10
.equ UDR        = $0c
.equ USR        = $0b
.equ UCR        = $0a
.equ UBRR       = $09
.equ ACSR       = $08
;***** 位定义
.equ SP7        =7
.equ SP6        =6
```

```
.equ SP5        =5
.equ SP4        =4
.equ SP3        =3
.equ SP2        =2
.equ SP1        =1
.equ SP0        =0
.equ INT1       =7
.equ INT0       =6
.equ INTF1      =7
.equ INTF0      =6
.equ TOIE1      =7
.equ OCIE1A     =6
.equ TICIE      =3
.equ TOIE0      =1
.equ TOV1       =7
.equ OCF1A      =6
.equ ICF1       =3
.equ TOV0       =1
.equ SE         =5
.equ SM         =4
.equ ISC11      =3
.equ ISC10      =2
.equ ISC01      =1
.equ ISC00      =0
.equ PORF       =1
.equ EXTRF      =0
.equ CS02       =2
.equ CS01       =1
.equ CS00       =0
.equ COM1A1     =7
.equ COM1A0     =6
.equ PWM11      =1
.equ PWM10      =0
.equ ICNC1      =7
.equ ICES1      =6
.equ CTC1       =3
.equ CS12       =2
.equ CS11       =1
.equ CS10       =0
.equ WDTOE      =4
.equ WDE        =3
.equ WDP2       =2
.equ WDP1       =1
.equ WDP0       =0
.equ EEMWE      =2
.equ EEWE       =1
.equ EERE       =0
.equ PB7        =7
.equ PB6        =6
.equ PB5        =5
.equ PB4        =4
.equ PB3        =3
.equ PB2        =2
.equ PB1        =1
.equ PB0        =0
.equ DDB7       =7
.equ DDB6       =6
.equ DDB5       =5
.equ DDB4       =4
.equ DDB3       =3
.equ DDB2       =2
.equ DDB1       =1
.equ DDB0       =0
.equ PINB7      =7
.equ PINB6      =6
.equ PINB5      =5
.equ PINB4      =4
.equ PINB3      =3
.equ PINB2      =2
.equ PINB1      =1
.equ PINB0      =0
.equ PD6        =6
.equ PD5        =5
.equ PD4        =4
.equ PD3        =3
.equ PD2        =2
.equ PD1        =1
.equ PD0        =0
.equ DDD6       =6
.equ DDD5       =5
.equ DDD4       =4
.equ DDD3       =3
.equ DDD2       =2
.equ DDD1       =1
.equ DDD0       =0
.equ PIND6      =6
.equ PIND5      =5
.equ PIND4      =4
.equ PIND3      =3
.equ PIND2      =2
.equ PIND1      =1
.equ PIND0      =0
.equ RXC        =7
.equ TXC        =6
.equ UDRE       =5
.equ FE         =4
.equ OR         =3
.equ RXCIE      =7
```

```
.equ TXCIE      = 6     .def YL         = r28
.equ UDRIE      = 5     .def YH         = r29
.equ RXEN       = 4     .def ZL         = r30
.equ TXEN       = 3     .def ZH         = r31
.equ CHR9       = 2     .equ RAMEND     = $ df
.equ RXB8       = 1     .equ INT0addr   = $ 001   ;外部中断 0 向量地址
.equ TXB8       = 0     .equ INT1addr   = $ 002   ;外部中断 1 向量地址
.equ ACD        = 7     .equ ICP1addr   = $ 003   ;输入捕获 1 向量地址
.equ ACO        = 5     .equ OC1addr    = $ 004   ;输出比较 1 向量地址
.equ ACI        = 4     .equ OVF1addr   = $ 005   ;溢出 1 中断向量地址
.equ ACIE       = 3     .equ OVF0addr   = $ 006   ;溢出 0 中断向量地址
.equ ACIC       = 2     .equ URXCaddr   = $ 007   ;UART 接收完成中断向量地址
.equ ACIS1      = 1     .equ UDREaddr   = $ 008   ;UART 数据寄存器空中断向量地址
.equ ACIS0      = 0     .equ UTXCaddr   = $ 009   ;UART 发送完成中断向量地址
.def XL         = r26   .equ ACIaddr    = $ 00a   ;模拟比较中断向量地址
.def XH         = r27
```

三、AT90S4414 单片机内部寄存器和位定义文件

```
;* * * * * I/O 寄存器定义
.equ SREG       = $ 3f
.equ SPH        = $ 3e
.equ SPL        = $ 3d
.equ GIMSK      = $ 3b
.equ GIFR       = $ 3a
.equ TIMSK      = $ 39
.equ TIFR       = $ 38
.equ MCUCR      = $ 35
.equ TCCR0      = $ 33
.equ TCNT0      = $ 32
.equ TCCR1A     = $ 2f
.equ TCCR1B     = $ 2e
.equ TCNT1H     = $ 2d
.equ TCNT1L     = $ 2c
.equ OCR1AH     = $ 2b
.equ OCR1AL     = $ 2a
.equ OCR1BH     = $ 29
.equ OCR1BL     = $ 28
.equ ICR1H      = $ 25
.equ ICR1L      = $ 24
.equ WDTCR      = $ 21
.equ EEAR       = $ 1e
.equ EEARL      = $ 1e
.equ EEDR       = $ 1d
.equ EECR       = $ 1c
.equ PORTA      = $ 1b
.equ DDRA       = $ 1a
.equ PINA       = $ 19
.equ PORTB      = $ 18
.equ DDRB       = $ 17
.equ PINB       = $ 16
.equ PORTC      = $ 15
.equ DDRC       = $ 14
.equ PINC       = $ 13
.equ PORTD      = $ 12
.equ DDRD       = $ 11
.equ PIND       = $ 10
.equ SPDR       = $ 0f
.equ SPSR       = $ 0e
.equ SPCR       = $ 0d
.equ UDR        = $ 0c
.equ USR        = $ 0b
.equ UCR        = $ 0a
.equ UBRR       = $ 09
.equ ACSR       = $ 08
;* * * * * 位定义
.equ INT1       = 7
.equ INT0       = 6
.equ INTF1      = 7
.equ INTF0      = 6
.equ TOIE1      = 7
.equ OCIE1A     = 6
.equ OCIE1B     = 5
.equ TICIE1     = 3
.equ TOIE0      = 1
.equ TOV1       = 7
.equ OCF1A      = 6
.equ OCF1B      = 5
.equ ICF1       = 3
.equ TOV0       = 1
.equ SRE        = 7
.equ SRW        = 6
.equ SE         = 5
```

```
.equ SM      =4
.equ ISC11   =3
.equ ISC10   =2
.equ ISC01   =1
.equ ISC00   =0
.equ CS02    =2
.equ CS01    =1
.equ CS00    =0
.equ COM1A1  =7
.equ COM1A0  =6
.equ COM1B1  =5
.equ COM1B0  =4
.equ PWM11   =1
.equ PWM10   =0
.equ ICNC1   =7
.equ ICES1   =6
.equ CTC1    =3
.equ CS12    =2
.equ CS11    =1
.equ CS10    =0
.equ WDTOE   =4
.equ WDE     =3
.equ WDP2    =2
.equ WDP1    =1
.equ WDP0    =0
.equ EEMWE   =2
.equ EEWE    =1
.equ EERE    =0
.equ PB7     =7
.equ PB6     =6
.equ PB5     =5
.equ PB4     =4
.equ PB3     =3
.equ PB2     =2
.equ PB1     =1
.equ PB0     =0
.equ DDB7    =7
.equ DDB6    =6
.equ DDB5    =5
.equ DDB4    =4
.equ DDB3    =3
.equ DDB2    =2
.equ DDB1    =1
.equ DDB0    =0
.equ PINB7   =7
.equ PINB6   =6
.equ PINB5   =5
.equ PINB4   =4
.equ PINB3   =3
.equ PINB2   =2
.equ PINB1   =1
.equ PINB0   =0
.equ PD6     =6
.equ PD5     =5
.equ PD4     =4
.equ PD3     =3
.equ PD2     =2
.equ PD1     =1
.equ PD0     =0
.equ DDD6    =6
.equ DDD5    =5
.equ DDD4    =4
.equ DDD3    =3
.equ DDD2    =2
.equ DDD1    =1
.equ DDD0    =0
.equ PIND6   =6
.equ PIND5   =5
.equ PIND4   =4
.equ PIND3   =3
.equ PIND2   =2
.equ PIND1   =1
.equ PIND0   =0
.equ RXC     =7
.equ TXC     =6
.equ UDRE    =5
.equ FE      =4
.equ OR      =3
.equ SPIE    =7
.equ SPE     =6
.equ DORD    =5
.equ MSTR    =4
.equ CPOL    =3
.equ CPHA    =2
.equ SPR1    =1
.equ SPR0    =0
.equ SPIF    =7
.equ WCOL    =6
.equ RXCIE   =7
.equ TXCIE   =6
.equ UDRIE   =5
.equ RXEN    =4
.equ TXEN    =3
.equ CHR9    =2
.equ RXB8    =1
.equ TXB8    =0
.equ ACD     =7
.equ ACO     =5
```

```
.equ ACI        =4      .equ INT0addr   = $001    ;外部中断 0 向量地址
.equ ACIE       =3      .equ INT1addr   = $002    ;外部中断 1 向量地址
.equ ACIC       =2      .equ ICP1addr   = $003    ;输入捕获 1 向量地址
.equ ACIS1      =1      .equ OC1Aaddr   = $004    ;输出比较 1A 向量地址
.equ ACIS0      =0      .equ OC1Baddr   = $005    ;输出比较 1B 向量地址
.def XL         =r26    .equ OVF1addr   = $006    ;溢出 1 中断向量地址
.def XH         =r27    .equ OVF0addr   = $007    ;溢出 0 中断向量地址
.def YL         =r28    .equ SPIaddr    = $008    ;SPI 中断向量地址
.def YH         =r29    .equ URXCaddr   = $009    ;UART 接收完成中断向量地址
.def ZL         =r30    .equ UDREaddr   = $00a    ;UART 数据寄存器空中断向量地址
.def ZH         =r31    .equ UTXCaddr   = $00b    ;UART 发送完成中断向量地址
.equ RAMEND     = $15F  .equ ACIaddr    = $00c    ;模拟比较中断向量地址
```

四、AT90S8515 单片机内部寄存器和位定义文件

```
;***** I/O 寄存器定义
.equ SREG       = $3f
.equ SPH        = $3e
.equ SPL        = $3d
.equ GIMSK      = $3b
.equ GIFR       = $3a
.equ TIMSK      = $39
.equ TIFR       = $38
.equ MCUCR      = $35
.equ TCCR0      = $33
.equ TCNT0      = $32
.equ TCCR1A     = $2f
.equ TCCR1B     = $2e
.equ TCNT1H     = $2d
.equ TCNT1L     = $2c
.equ OCR1AH     = $2b
.equ OCR1AL     = $2a
.equ OCR1BH     = $29
.equ OCR1BL     = $28
.equ ICR1H      = $25
.equ ICR1L      = $24
.equ WDTCR      = $21
.equ EEARH      = $1f
.equ EEARL      = $1e
.equ EEDR       = $1d
.equ EECR       = $1c
.equ PORTA      = $1b
.equ DDRA       = $1a
.equ PINA       = $19
.equ PORTB      = $18
.equ DDRB       = $17
.equ PINB       = $16
.equ PORTC      = $15
.equ DDRC       = $14
.equ PINC       = $13
.equ PORTD      = $12
.equ DDRD       = $11
.equ PIND       = $10
.equ SPDR       = $0f
.equ SPSR       = $0e
.equ SPCR       = $0d
.equ UDR        = $0c
.equ USR        = $0b
.equ UCR        = $0a
.equ UBRR       = $09
.equ ACSR       = $08
;***** 位定义
.equ INT1       =7
.equ INT0       =6
.equ INTF1      =7
.equ INTF0      =6
.equ TOIE1      =7
.equ OCIE1A     =6
.equ OCIE1B     =5
.equ TICIE1     =3
.equ TOIE0      =1
.equ TOV1       =7
.equ OCF1A      =6
.equ OCF1B      =5
.equ ICF1       =3
.equ TOV0       =1
.equ SRE        =7
.equ SRW        =6
.equ SE         =5
.equ SM         =4
.equ ISC11      =3
.equ ISC10      =2
.equ ISC01      =1
.equ ISC00      =0
.equ CS02       =2
.equ CS01       =1
.equ CS00       =0
```

```
.equ COM1A1      =7
.equ COM1A0      =6
.equ COM1B1      =5
.equ COM1B0      =4
.equ PWM11       =1
.equ PWM10       =0
.equ ICNC1       =7
.equ ICES1       =6
.equ CTC1        =3
.equ CS12        =2
.equ CS11        =1
.equ CS10        =0
.equ WDTOE       =4
.equ WDE         =3
.equ WDP2        =2
.equ WDP1        =1
.equ WDP0        =0
.equ EEMWE       =2
.equ EEWE        =1
.equ EERE        =0
.equ PB7         =7
.equ PB6         =6
.equ PB5         =5
.equ PB4         =4
.equ PB3         =3
.equ PB2         =2
.equ PB1         =1
.equ PB0         =0
.equ DDB7        =7
.equ DDB6        =6
.equ DDB5        =5
.equ DDB4        =4
.equ DDB3        =3
.equ DDB2        =2
.equ DDB1        =1
.equ DDB0        =0
.equ PINB7       =7
.equ PINB6       =6
.equ PINB5       =5
.equ PINB4       =4
.equ PINB3       =3
.equ PINB2       =2
.equ PINB1       =1
.equ PINB0       =0
.equ PD6         =6
.equ PD5         =5
.equ PD4         =4
.equ PD3         =3
.equ PD2         =2
.equ PD1         =1
.equ PD0         =0
.equ DDD6        =6
.equ DDD5        =5
.equ DDD4        =4
.equ DDD3        =3
.equ DDD2        =2
.equ DDD1        =1
.equ DDD0        =0
.equ PIND6       =6
.equ PIND5       =5
.equ PIND4       =4
.equ PIND3       =3
.equ PIND2       =2
.equ PIND1       =1
.equ PIND0       =0
.equ RXC         =7
.equ TXC         =6
.equ UDRE        =5
.equ FE          =4
.equ OR          =3
.equ SPIE        =7
.equ SPE         =6
.equ DORD        =5
.equ MSTR        =4
.equ CPOL        =3
.equ CPHA        =2
.equ SPR1        =1
.equ SPR0        =0
.equ SPIF        =7
.equ WCOL        =6
.equ RXCIE       =7
.equ TXCIE       =6
.equ UDRIE       =5
.equ RXEN        =4
.equ TXEN        =3
.equ CHR9        =2
.equ RXB8        =1
.equ TXB8        =0
.equ ACD         =7
.equ ACO         =5
.equ ACI         =4
.equ ACIE        =3
.equ ACIC        =2
.equ ACIS1       =1
.equ ACIS0       =0
.def XL          =r26
.def XH          =r27
.def YL          =r28
```

```
.def YH                 = r29
.def ZL                 = r30
.def ZH                 = r31
.equ RAMEND             = $25F
.equ INT0addr           = $001          ;外部中断 0 向量地址
.equ INT1addr           = $002          ;外部中断 1 向量地址
.equ ICP1addr           = $003          ;输入捕获 1 向量地址
.equ OC1Aaddr           = $004          ;输出比较 1A 向量地址
.equ OC1Baddr           = $005          ;输出比较 1B 向量地址
.equ OVF1addr           = $006          ;溢出 1 中断向量地址
.equ OVF0addr           = $007          ;溢出 0 中断向量地址
.equ SPIaddr            = $008          ;SPI 中断向量地址
.equ URXCaddr           = $009          ;UART 接收完成中断向量地址
.equ UDREaddr           = $00a          ;UART 数据寄存器空中断向量地址
.equ UTXCaddr           = $00b          ;UART 发送完成中断向量地址
.equ ACIaddr            = $00c          ;模拟比较中断向量地址
```

5.2.2 访问内部 EEPROM

该应用程序说明如何读 EEPROM 数据和写数据到 EEPROM。其中包括：写 EEPROM 子程序“EEWrite”，读 EEPROM 子程序“EERead”，按序写 EEPROM 子程序“EEWrite _ seq”、按序读 EEPROM 子程序“EERead _ seq”和一个测试程序。

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* Title:                  存取 AT90S1200 EEPROM
;* Target:                 AT90S1200
;* 这个例子说明如何读 EEPROM 数据和写数据到 EEPROM。随机存取和按序存取的例行程序列表如下。
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
.include "1200def.inc"
    rjmp RESET          ;复位处理
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* EEWrite
;* 该子程序等待 EEPROM 就绪后编程，使用寄存器变量“EEdwr”和地址变量“EEawr”对 EEPROM 编程。
;* Number of words        :5 + return
;* Number of cycles       :10 + return ( 如果 EEPROM 已就绪)
;* Low Registers used     :None
;* High Registers used:   :2 (EEdwr, EEawr)
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* * * * * 子程序寄存器变量
.def EEdwr = r16            ;用于把数据字节写到 EEPROM
.def EEawr = r17            ;用于写地址
;* * * * * 代码
EEWrite:
    sbic    EECR, EEWE      ;如果 EEWE 不清除
    rjmp    EEWrite         ;等待
    out     EEAR, EEawr     ;输出地址
    out     EEDR, EEdwr     ;输出数据
    sbi     EECR, EEWE      ;设置 EEPROM 写选通
                            ;该指令需 4 个时钟周期，由于它暂停 CPU 2 个时钟周期
```

```
    ret
;*********************************************************************
;* EERead
;* 该子程序等待 EEPROM 就绪后才编程,然后从地址变量"EEard"中读寄存器变量"EEdrd"中的数据。
;* Number of words:6 + return
;* Number of cycles          :12 + return (如果 EEPROM 已就绪)
;* Low Registers used        :1 (EEdrd)
;* High Registers used:      :1 (EEard)
;*********************************************************************
;***** 子程序寄存器变量
.def EEdrd=r0               ;结果数据字节
.def EEard=r16              ;读数地址
;***** 代码
EERead:
    sbic    EECR,EEWE       ;如果 EEWE 不清除
    rjmp    EERead          ;等待
    out     EEDR,EEard      ;输出地址
    sbi     EECR,EERE       ;设置 EEPROM 读选通
                            ;该指令需 4 个时钟周期,由于它暂停 CPU 2 个时钟周期
    sbi     EECR,EERE       ;第 2 次设置 EEPROM 读选通
                            ;该指令需 4 个时钟周期,由于它暂停 CPU 2 个时钟周期
    in      EEdrd,EEDR      ;获得数据
    ret
;*********************************************************************
;* EEWrite_seq
;* 该子程序使 EEPROM 地址加 1,并等待 EEPROM 就绪后编程,然后随着寄存器变
;* 量 "EEdwr_s"对 EEPROM 编程。
;* Number of words:7 + return
;* Number of cycles          :10 + return (如果 EEPROM 已就绪)
;* Low Registers used        :1 (EEwtmp)
;* High Registers used:      :1 (EEdwr_s)
;*********************************************************************
;***** 子程序寄存器变量
.def EEwtmp    =r0          ;地址暂存
.def EEdwr_s   =r16         ;写数据
;***** 代码
EEWrite_seq:
    sbic    EECR,EEWE       ;如果 EEWE 不清除
    rjmp    EEWrite_seq     ;等待
    in      EEwtmp,EEAR     ;获得地址
    inc     EEwtmp          ;地址加 1
    out     EEAR,EEwtmp     ;输出地址
    out     EEDR,EEdwr_s    ;输出数据
    sbi     EECR,EEWE       ;设置 EEPROM 写选通
                            ; 该指令需 4 个时钟周期,由于它暂停 CPU 2 个时钟周期
    ret
;*********************************************************************
;* EERead_seq
;* 该子程序使 EEAR 存储地址加 1,并读 EEPROM 内容到寄存器变量"EEdrd_s"。
```

```
;* Number of words :6 + return
;* Number of cycles              :12 + return (如果 EEPROM 已就绪)
;* Low Registers used            :2 (EErtmp, EEdrd_s)
;* High Registers used:          :无
;*********************************************************************
;***** 子程序寄存器变量
.def        EErtmp = r0          ;地址暂存
.def        EEdrd_s = r1         ;数据字节结果
;***** 代码
EERead_seq:
    in      EErtmp, EEAR         ;获得地址
    inc     EErtmp               ;地址加 1
    out     EEAR, EErtmp         ;输出地址
    sbi     EECR, EERE           ;设置 EEPROM 读选通
                                 ;该指令需 4 个时钟周期,由于它暂停 CPU 2 个时钟周期
    sbi     EECR, EERE           ;第 2 次设置 EEPROM 读选通
                                 ;该指令需 4 个时钟周期,由于它暂停 CPU 2 个时钟周期
    in      EEdrd_s, EEDR        ;获得数据
    ret
;*********************************************************************
;* 测试/例子程序
;*********************************************************************
;***** 主程序寄存器变量
.def        counter              = r18
.def        temp                 = r19
;***** 代码
RESET:
;***** 随机定位程序
    ldi     EEdwr, $aa
    ldi     EEawr, $10
    rcall   EEWrite              ;存储 $aa 到 EEPROM 的 $10 位置
;***** 从随机定位地址读
    ldi     EEard, $10
    rcall   EERead               ;读 $10 地址
    out     PORTB, EEdrd         ;输出值到 B 口
;***** 以 $55, $aa, $55, $aa,…模式填充 EEPROM
    ldi     counter, 64          ;初始化循环计数器
    ser     temp
    out     EEAR, temp           ;EEAR <- $ff (start address - 1)
loop1:  ldi  EEdwr_s, $55
    rcall   EEWrite_seq          ;写 $55 到 EEPROM
    ldi     EEdwr_s, $aa
    rcall   EEWrite_seq          ;写 $aa 到 EEPROM
    dec     counter              ;计数器减 1
    brne    loop1                ;未完成循环
;***** 拷贝 EEPROM 最前 10 个字节到 r2--r11
    ldi     temp, $ff
    out     EEAR, temp           ;EEAR <- $ff (start address - 1)
    ldi     ZL, 2                ;Z 指针指向 r2
```

```
loop2:      rcall EERead_seq        ;得到 EEPROM 数据
    st      Z,EEdrd_s               ;存储到 SRAM
    inc     ZL
    cpi     ZL,12                   ;到结尾?
    brne    loop2                   ;未完成循环
forever:    rjmp forever            ;无限循环
```

5.2.3 数据块传送

该应用程序说明，如何传送程序存储器的一个数据块到 SRAM 和把 SRAM 中的一个数据块传送到另一个 SRAM。其中包括，一个程序存储器块传送子程序"flash 2ram"、SRAM 块传送子程序"ram 2ram"和一个测试程序。

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* Title:                     块拷贝例行程序
;* Target:                    AT90SXX1X (Devices with SRAM)
;* 该例说明怎样拷贝程序存储器的一个数据块到 SRAM，或从一个 SRAM 到另一个 SRAM 的数据拷贝。
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
.include "8515def.inc"
 rjmp RESET ;复位处理
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* "flash2ram"
;* 该子程序拷贝程序存储器 (Flash) 的一个数据块到内部 SRAM。下列参数在调用子程序前必须设置：
;*     Z-pointer: Flash 块开始地址(2 (代码段用字);
;*     Y-pointer: RAM 块开始地址;
;*     romsize: 块尺寸。
;* Number of words :5 + return
;* Number of cycles            :10 × block size + return
;* Low Registers used          :1 (r0)
;* High Registers used         :1 (flash 尺寸)
;* Pointers used               :Y, Z
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* * * * * 子程序寄存器变量
.def flashsize=r16             ;拷贝的块尺寸
;* * * * * 代码
flash2ram:
    lpm                        ;得到常数
    st Y+,r0                   ;存储到 SRAM，并使 Y 指针加 1
    adiw ZL,1                  ; Z 指针加 1
    dec flashsize
    brne flash2ram             ;如不是表尾，循环
    ret
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* "ram2ram"
;* 该子程序拷贝 SRAM 的一个数据块到另一个 SRAM。下列参数在调用子程序前必须设置。
;*     Z-pointer: 需拷贝 RAM 的开始范围;
;*     Y-pointer: 拷贝到 RAM 的开始范围 ;
;*     ramsize: 拷贝的块尺寸。
;* Number of words             :4 + return
;* Number of cycles            :6 × block size + return
```

```
; * Low Registers used          ;1 (ramtemp)
; * High Registers used         ;1 (ramsize)
; * Pointers used               ;Y, Z
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * 子程序寄存器变量
.def ramtemp = r1               ;暂存寄存器
.def ramsize = r16              ;拷贝的块尺寸
; * * * * * 代码
ram2ram:
    ld ramtemp, Z+              ;从 BLOCK1 获得数据
    st Y+, ramtemp              ;存储数据到 BLOCK2
    dec ramsize
    brne ram2ram                ;如果未结束,循环
    ret
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * 测试程序
; * 该程序拷贝程序存储器的 20 个字节数据到由 BLOCK1 定位的 SRAM 范围,然后再
; * 把 SRAM 范围内数据拷贝到由 BLOCK2 定位的 SRAM 内。
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
.equ BLOCK1 = $60               ;SRAM 范围开始地址 #1
.equ BLOCK2 = $80               ;SRAM 范围开始地址 #2
; * * * * * 主程序寄存器变量
.def temp = r16                 ;暂存变量
; * * * * * 代码
RESET:
    ldi temp, low(RAMEND)
    out SPL, temp               ;初始化堆栈指针
    ldi temp, high(RAMEND)
    out SPH, temp
; * * * * * 拷贝 ROM 中 20 字节到 RAM
    ldi ZH, high(F_TABLE * 2)
    ldi ZL, low(F_TABLE * 2)    ;初始化 Z 指针
    ldi YH, high(BLOCK1)
    ldi YL, low(BLOCK1)         ;初始化 Y 指针
    ldi flashsize, 20
rcall flash2ram                 ;拷贝 20 字节
    ; * * * * * 拷贝 RAM 中 20 字节到 RAM
    ldi ZH, high(BLOCK1)
    ldi ZL, low(BLOCK1)         ;初始化 Z 指针
    ldi YH, high(BLOCK2)
    ldi YL, low(BLOCK2)         ;初始化 Y 指针
    ldi ramsize, 20
    rcall ram2ram               ;拷贝 20 字节
forever: rjmp forever           ;无限循环
F_TABLE:                        ;表开始 (20 字节)
    .db 0, 1                                    .db 8, 9
    .db 2, 3                                    .db 10, 11
    .db 4, 5                                    .db 12, 13
    .db 6, 7                                    .db 14, 15
```

```
    .db 16,17                                   .db 18,19
```

5.2.4 乘法和除法运算应用一

该应用程序列出了 8×8 位无符号乘法子程序，16×16 位无符号乘法子程序，8/8 位无符号除法子程序，16/16 位无符号除法子程序和一个测试程序。

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* Title:                                    乘法和除法例行程序
;* 该程序列出下列乘法/除法应用子程序:8×8 位无符号;8×8 位带符号;16×16 位无符号; 16×16 位带符号;8/8 位无符号;8/8 位带符号;16/16 位无符号;16/16 位带符号。
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
.include "1200def.inc"
    rjmp RESET                              ;复位处理
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* "mpy8u" 8×8 位无符号乘法
;* 该子程序是两个寄存器变量 mp8u 和 mc8u 相乘,结果送至寄存器 m8uH 和 m8uL。
;* Number of words :9 + return
;* Number of cycles                        :58 + return
;* Low registers used                      :None
;* High registers used                     :4 (mp8u, mc8u/m8uL, m8uH, mcnt8u)
;*注意:结果的低字节和乘数用相同的寄存器,这是因为乘数被结果覆盖了。
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* * * * * 子程序寄存器变量
.def    mc8     =r16            ;被乘数
.def    mp8u    =r17            ;乘数
.def    m8uL    =r17            ;结果低字节
.def    m8uH    =r18            ;结果高字节
.def    mcnt8u  =r19            ;循环计数器
;* * * * * 代码
mpy8u:  clr     m8uH            ;清结果高字节
        ldi     mcnt8u,8        ;初始化循环计数器
        lsr     mp8u            ;乘数循环
m8u_1:  brcc    m8u_2           ;进位置位
        add     m8uH,mc8u       ;加被乘数到结果高字节
m8u_2:  ror     m8uH            ;结果高字节右循环
        ror     m8uL            ;结果低字节和乘数右循环
        dec     mcnt8u          ;循环计数器减 1
        brne    m8u_1           ;如没完成,再循环
        ret
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* "mpy8s" 8×8 位带符号乘法
;* 该子程序是两个寄存器变量 mp8s 和 mc8s 带符号相乘,结果送寄存器 m8sH 和 m8sL。
;* 该程序是 Booth 算法的执行程序。如果结果需要所有 16 位,应该避免调用带有 -128
;* ($80) 值的被乘数。
;* Number of word :10 + return
;* Number of cycles                        :73 + return
;* Low registers used                      :None
;* High registers used                     :4 (mc8s, mp8s/m8sL, m8sH, mcnt8s)
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
;*****  子程序寄存器变量
.def    mc8s     =r16            ;被乘数.def mp8s =r17 ;乘数
.def    m8sL     =r17            ;结果低字节
.def    m8sH     =r18            ;结果高字节
.def    mcnt8s   =r19            ;循环计数器
;*****  代码
mpy8s:  sub      m8sH,m8sH       ;清除结果高字节和进位位
        ldi      mcnt8s,8        ;初始化循环计数器
m8s_1:  brcc     m8s_2           ;如果进位位置位
        add      m8sH,mc8s       ;加被乘数到结果高字节
m8s_2:  sbrc     mp8s,0          ;如果当前位置位
        sub      m8sH,mc8s       ;从结果高字节减被乘数
        asr      m8sH            ;右移结果高字节
        ror      m8sL            ;右移结果低字节和乘数
        dec      mcnt8s          ;减循环计数器
        brne     m8s_1           ;如果没完成,再循环
        ret
;*************************************************************************
;*"mpy16u"16×16位无符号乘法
;* 这个子程序是两个16位寄存器变量(mp16uH:mp16uL和mc16uH:mc16uL)相乘。
;* 结果送至寄存器m16u3:m16u2:m16u1:m16u0。
;* Number of words :14 + return
;* Number of cycles           :153 + return
;* Low registers used         :None
;* High registers used        :7 (mp16uL,mp16uH,mc16uL/m16u0,mc16uH/m16u1,m16u2,
                                  m16u3,mcnt16u)
;*************************************************************************
;*****  子程序寄存器变量
.def    mc16uL   =r16            ;被乘数低字节
.def    mc16uH   =r17            ;被乘数高字节
.def    mp16uL   =r18            ;乘数低字节
.def    mp16uH   =r19            ;乘数高字节
.def    m16u0    =r18            ;结果位0 (LSB)
.def    m16u1    =r19            ;结果位1
.def    m16u2    =r20            ;结果位2
.def    m16u3    =r21            ;结果位3 (MSB)
.def    mcnt16u  =r22            ;循环计数器
;*****  代码
mpy16u: clr      m16u3           ;清结果最高2字节
        clr      m16u2
        ldi      mcnt16u,16      ;初始化循环计数器
        lsr      mp16uH
        ror      mp16uL
m16u_1: brcc     noad8           ;如果乘数位0置位
        add      m16u2,mc16uL    ;加被乘数低位到结果的2字节
        adc      m16u3,mc16uH    ;加被乘数高位到结果的3字节
noad8:  ror      m16u3           ;右移结果3字节
        ror      m16u2           ;右循环结果2字节
        ror      m16u1           ;循环结果1字节和乘数高
```

```
        ror     m16u0           ;循环结果 0 字节和乘数低
        dec     mcnt16u         ;减循环计数器
        brne    m16u_1          ;如果没完成,再循环
        ret
;*****************************************************
;* "mpy16s"16×16 位带符号乘法
;* 这个子程序是两个 16 位寄存器变量 (mp16sH:mp16sL 和 mc16sH:mc16sL)相乘。
;* 结果送至寄存器 m16s3:m16s2:m16s1:m16s0。
;* 该程序是 Booth 算法的执行程序。如果结果需要所有 32 位,应该避免调用带有 -32 768
;* ( $8000) 值的被乘数。
;* Number of words :16 + return
;* Number of cycles                :210/226 (Min/Max) + return
;* Low registers used              :None
;* High registers used             :7 (mp16sL, mp16sH, mc16sL/m16s0, mc16sH/m16s1, m16s2,
;*                                     m16s3, mcnt16s)
;*****************************************************
;***** 子程序寄存器变量
.def    mc16sL  = r16           ;被乘数低字节
.def    mc16sH  = r17           ;被乘数高字节
.def    mp16sL  = r18           ;乘数低字节
.def    mp16sH  = r19           ;乘数高字节
.def    m16s0   = r18           ;结果位 0 (LSB)
.def    m16s1   = r19           ;结果位 1
.def    m16s2   = r20           ;结果位 2
.def    m16s3   = r21           ;结果位 3 (MSB)
.def    mcnt16s = r22           ;循环计数器
;***** 代码
mpy16s: clr     m16s3           ;清除结果字节 3
        sub     m16s2, m16s2    ;清除结果字节 2 和进位
        ldi     mcnt16s, 16     ;初始化循环计数器
m16s_1: brcc    m16s_2          ;如果进位位置位
        add     m16s2, mc16sL   ;加被乘数低字节到结果字节 2
        adc     m16s3, mc16sH   ;加被乘数高字节到结果字节 3
m16s_2: sbrc    mp16sL, 0       ;如果当前位置位
        sub     m16s2, mc16sL   ;从结果字节 2 中减被乘数低字节
        sbrc    mp16sL, 0       ;如果当前位置位
        sbc     m16s3, mc16sH   ;从结果字节 3 中减被乘数高字节
        asr     m16s3           ;右移结果和乘数
        ror     m16s2
        ror     m16s1
        ror     m16s0
        dec     mcnt16s         ;减计数器
        brne    m16s_1          ;如果没完成,再循环
        ret
;*****************************************************
;* "div8u" 8/8 位无符号除法
;* 该子程序是两个寄存器变量 dd8u(被除数)和 dv8u(除数)相除,结果送至寄存器 dres8u,
;* 余数送至寄存器 drem8u。
;* Number of words :14
```

```
;*  Number of cycles                    :97
;*  Low registers used                  :1 (drem8u)
;*  High registers used                 :3 (dres8u/dd8u, dv8u, dcnt8u)
;***************************************************************
;*****子程序寄存器变量
.def    drem8u   =r15               ;余数
.def    dres8u   =r16               ;结果
.def    dd8u     =r16               ;被除数
.def    dv8u     =r17               ;除数
.def    dcnt8u   =r18               ;循环计数器
;***** 代码
div8u:  sub      drem8u, drem8u     ;清除余数和进位
        ldi      dcnt8u, 9          ;初始化循环计数器
d8u_1:  rol      dd8u               ;左移被除数
        dec      dcnt8u             ;减计数器
        brne     d8u_2              ;如完成
        ret                         ;返回
d8u_2:  rol      drem8u             ;被除数被移到余数
        sub      drem8u, dv8u       ;余数=余数-除数
        brcc     d8u_3              ;如结果为负
        add      drem8u, dv8u       ;存储余数
        clc                         ;清进位并移到结果
        rjmp     d8u_1              ;否则
d8u_3:  sec                         ;置进位并移到结果
        rjmp     d8u_1
;***************************************************************
;*"div8s" 8/8 位带符号除法
;*  该子程序是两个寄存器变量 dd8s (被除数)和 dv8s (除数)相除,结果送至寄存器 dres8s,
;*  余数送至寄存器 drem8s。
;*  Number of words                     :22
;*  Number of cycles                    :103
;*  Low registers used                  :2 (d8s, drem8s)
;*  High registers used                 :3 (dres8s/dd8s, dv8s, dcnt8s)
;***************************************************************
;*****子程序寄存器变量
.def    d8s      =r14               ;符号寄存器
.def    drem8s   =r15               ;余数
.def    dres8s   =r16               ;结果
.def    dd8s     =r16               ;被除数
.def    dv8s     =r17               ;除数
.def    dcnt8s   =r18               ;循环计数器
;***** 代码
div8s:  mov      d8s, dd8s          ;移被除数到符号寄存器
        eor      d8s, dv8s          ;符号寄存器与除数相异或
        sbrc     dv8s, 7            ;如除数最高位(MSB)置位
        neg      dv8s               ;改变除数的符号
        sbrc     dd8s, 7            ;如被除数最高位(MSB)置位
        neg      dd8s               ;改变除数的符号
        sub      drem8s, drem8s     ;清除余数和进位
```

```
        ldi     dcnt8s, 9           ;初始化循环计数器
d8s_1:  rol     dd8s                ;左移被除数
        dec     dcnt8s              ;减计数器
        brne    d8s_2               ;如果完成
        sbrc    d8s, 7              ;如果符号寄存器的最高位(MSB)置位
        neg     dres8s              ;改变结果的符号
        ret                         ;返回
d8s_2:  rol     drem8s              ;移被除数到余数
        sub     drem8u, dv8s        ;余数 = 余数 - 除数
        brcc    d8s_3               ;如果结果为负
        add     drem8u, dv8s        ;存储余数
        clc                         ;清进位并移到结果
        rjmp    d8s_1               ;否则
d8s_3:  sec                         ;置进位并移到结果
        rjmp    d8s_1
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*"div16u" 16/16 位无符号除法
;* 该子程序是两个 16 位数"dd16uH:dd16uL"(被除数)和"dv16uH:dv16uL"(除数)相除。
;* 结果送至"dres16uH:dres16uL",余数送至"drem16uH:drem16uL"。
;* Number of words            :19
;* Number of cycles           :235/251 (Min/Max)
;* Low registers used         :2 (drem16uL, drem16uH)
;* High registers used        :5 (dres16uL/dd16uL, dres16uH/dd16uH, dv16uL, dv16uH, dcnt16u)
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* * * * * 子程序寄存器变量
.def    drem16uL                    = r14
.def    drem16uH                    = r15
.def    dres16uL                    = r16
.def    dres16uH                    = r17
.def    dd16uL                      = r16
.def    dd16uH                      = r17
.def    dv16uL                      = r18
.def    dv16uH                      = r19
.def    dcnt16u                     = r20
;* * * * * 代码
div16u: clr     drem16uL            ;清除余数低字节
        sub     drem16uH,drem16uH   ;清除余数高字节和进位
        ldi     dcnt16u, 17         ;初始化循环计数器
d16u_1: rol     dd16uL              ;左移被除数
        rol     dd16uH
        dec     dcnt16u             ;减计数器
        brne    d16u_2              ;如完成
        ret                         ;返回
d16u_2: rol     drem16uL            ;移被除数到余数
        rol     drem16uH
        sub     drem16uL, dv16uL    ;余数 = 余数 - 除数
        sbc     drem16uH, dv16uH
        brcc    d16u_3              ;如结果为负
        add     drem16uL, dv16uL    ;存储余数
```

```
        adc     drem16uH, dv16uH
        clc                         ;清进位并移到结果
        rjmp    d16u_1              ;否则
d16u_3: sec                         ;置进位并移到结果
        rjmp    d16u_1
;*****************************************************
;* "div16s" 16/16 位带符号除法
;* 该子程序是两个 16 位数"dd16sH:dd16sL"(被除数)和"dv16sH:dv16sL"(除数)带符号相除。
;* 结果送至"dres16sH:dres16sL", 余数送至"drem16sH:drem16sL"。
;* Number of words :39
;* Number of cycles                  :247/263 (Min/Max)
;* Low registers used                :3 (d16s, drem16sL, drem16sH)
;* High registers used               :7 (dres16sL/dd16sL, dres16sH/dd16sH, dv16sL, dv16sH, dcnt16sH)
;*****************************************************
;***** 子程序寄存器变量
.def    d16s    = r13               ;符号寄存器
.def    drem16sL = r14              ;余数低字节
.def    drem16sH = r15              ;余数高字节
.def    dres16sL = r16              ;结果低字节
.def    dres16sH = r17              ;结果高字节
.def    dd16sL  = r16               ;被除数低字节
.def    dd16sH  = r17               ;被除数高字节
.def    dv16sL  = r18               ;除数低字节
.def    dv16sH  = r19               ;除数高字节
.def    dcnt16s = r20               ;循环计数器
;***** 代码
div16s: mov     d16s, dd16sH        ;移被除数到符号寄存器
        eor     d16s, dv16sH        ;除数高字节与符号寄存器相异或
        sbrs    dd16sH, 7           ;如被除数最高位(MSB)置位
        rjmp    d16s_1
        com     dd16sH              ;改变除数的符号
        com     dd16sL
        subi    dd16sL, low(-1)
        sbci    dd16sL, high(-1)
d16s_1: sbrs    dv16sH, 7           ;如除数最高位(MSB)置位
        rjmp    d16s_2
        com     dv16sH              ;改变除数的符号
        com     dv16sL
        subi    dv16sL, low(-1)
        sbci    dv16sL, high(-1)
d16s_2: clr     drem16sL            ;清除余数低字节
        sub     drem16sH, drem16sH  ;清除余数高字节和进位
        ldi     dcnt16s, 17         ;初始化循环计数器
d16s_3: rol     dd16sL              ;左移被除数
        rol     dd16sH
        dec     dcnt16s             ;减计数器
        brne    d16s_5              ;如完成
        sbrs    d16s, 7             ;如是符号寄存器的最高位(MSB)置位
        rjmp    d16s_4
```

```
        com     dres16sH         ;改变结果的符号
        com     dres16sL
        subi    dres16sL, low(-1)
        sbci    dres16sH, high(-1)
d16s_4: ret                      ;返回
d16s_5: rol     drem16sL         ;移被除数到余数
        rol     drem16sH
        sub     drem16sL, dv16sL ;余数=余数-除数
        sbc     drem16sH, dv16sH
        brcc    d16s_6           ;如结果为负
        add     drem16sL, dv16sL ;存储余数
        adc     drem16sH, dv16sH
        clc                      ;清进位并移到结果
        rjmp    d16s_3           ;否则
d16s_6: sec                      ;置进位并移到结果
        rjmp    d16s_3
;********************************************
;* 测试程序
;********************************************
;***** 主程序寄存器变量
.def    temp    =r16             ;暂存变量
;***** 代码
RESET:
;-------------------------------------------
;如用 SRAM 除法应包括下列程序行:
    ldi   temp, low(RAMEND)
    out   SPL, temp
    ldi   temp, high(RAMEND)
    out   SPH, temp ;初始化堆栈
;-------------------------------------------
;***** 两个无符号 8 位数乘 (250 * 4)
    ldi  mc8u, 250
    ldi  mp8u, 4
    rcall mpyu                  ;结果: m8uH:m8uL= $03e8 (1 000)
;***** 两个带符号 8 位数乘 (-99 * 88)
    ldi  mc8s, -99
    ldi  mp8s, 88
    rcall mpy8s                 ;结果: m8sH:m8sL= $ddf8 (-8 712)
;***** 两个无符号 16 位数乘 (5050 * 10000)
    ldi  mc16uL, low(5050)
    ldi  mc16uH, high(5050)
    ldi  mp16uL, low(10000)
    ldi  mp16uH, high(10000)
    rcall mpy16u                ;结果: m16u3:m16u2:m16u1:m16u0
                                ;=030291a0 (50 500 000)
;***** 两个带符号 16 位数乘 (-12 345*(-4 321))
    ldi  mc16sL, low(-12345)
    ldi  mc16sH, high(-12345)
    ldi  mp16sL, low(-4321)
```

```
    ldi  mp16sH, high(-4321)
    rcall mpy16s                 ;结果: m16s3:m16s2:m16s1:m16s0
                                 ;= $032df219 (53342745)
;* * * * * 两个无符号 8 位数除 (100/3)
    ldi  dd8u, 100
    ldi  dv8u, 3
    rcall div8u                  ;结果: $21 (33)
                                 ;余数: $01 (1)
;* * * * * 两个带符号 8 位数除 (-110/-11)
    ldi  dd8s, -110
    ldi  dv8s, -11
    rcall div8s                  ;结果: $0a (10)
                                 ;余数: $00 (0)
;* * * * * 两个无符号 16 位数除 (50000/60000)
    ldi  dd16uL, low(50000)
    ldi  dd16uH, high(50000)
    ldi  dv16uL, low(60000)
    ldi  dv16uH, high(60000)
    rcall  div16u                ;结果: $0000 (0)
                                 ;余数: $c350 (50 000)
;* * * * * 两个带符号 16 位数除 (-22222/10)
    ldi  dd16sL, low(-22222)
    ldi  dd16sH, high(-22222)
    ldi  dv16sL, low(10)
    ldi  dv16sH, high(10)
    rcall div16s                 ;结果: $f752 (-2 222)
                                 ;余数: $0002 (2)
forever: rjmp   forever
```

5.2.5 乘法和除法运算应用二

该应用程序列出了 8×8 位无符号乘、8×8 位带符号乘，16×16 位无符号乘、16×16 位带符号乘，8/8 位无符号除、8/8 位带符号除，16/16 位无符号除、16/16 位带符号除子程序和一个测试程序。

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*        乘法和除法例行程序
;*  该程序列出下列乘法/除法应用子程序。例行程序已实现优化。
;*      8 × 8 = 16 位无符号数
;*      16 × 16 = 32 位无符号数
;*      8 / 8 = 8 + 8 位无符号数
;*      16 / 16 = 16 + 16 位无符号数
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
.include "1200def.inc"
    rjmp RESET                    ;复位处理
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* "mpy8u"8×8 位无符号乘法
;*  该子程序是两个寄存器变量 mp8u 和 mc8u 相乘。结果送至寄存器 m8uH 和 m8uL。
;*  Number of words        :34 + return
;*  Number of cycles       :34 + return
;*  Low registers used     :None;
;*  High registers used    :3 (mc8u, mp8u/m8uL, m8uH);
```

注意:结果的低字节和乘数用相同的寄存器,这是因为乘数被结果覆盖了。

```
;*********************************************************
;***** 子程序寄存器变量
.def    mc8u    = r16           ;被乘数
.def    mp8u    = r17           ;乘数
.def    m8uL    = r17           ;结果低字节
.def    m8uH    = r18           ;结果高字节
;***** 代码
mpy8u:  clr     m8uH            ;清除结果高字节
        lsr     mp8u            ;移位乘数
        brcc    noad80          ;如进位置位
        add     m8uH, mc8u      ;加被乘数到结果高字节
noad80: ror     m8uH            ;右移结果高字节
        ror     m8uL            ;结果低字节和乘数循环右移
        brcc    noad81          ;如进位置位
        add     m8uH, mc8u      ;加被乘数到结果高字节
noad81: ror     m8uH            ;右移结果高字节
        ror     m8uL            ;结果低字节和乘数循环右移
        brcc    noad82          ;如进位置位
        add     m8uH, mc8u      ;加被乘数到结果高字节
noad82: ror     m8uH            ;右移结果高字节
        ror     m8uL            ;结果低字节和乘数循环右移
        brcc    noad83          ;如进位置位
        add     m8uH, mc8u      ;加被乘数到结果高字节
noad83: ror     m8uH            ;右移结果高字节
        ror     m8uL            ;结果低字节和乘数循环右移
        brcc    noad84          ;如进位置位
        add     m8uH, mc8u      ;加被乘数到结果高字节
noad84: ror     m8uH            ;右移结果高字节
        ror     m8uL            ;结果低字节和乘数循环右移
        brcc    noad85          ;如进位置位
        add     m8uH, mc8u      ;加被乘数到结果高字节
noad85: ror     m8uH            ;右移结果高字节
        ror     m8uL            ;结果低字节和乘数循环右移
        brcc    noad86          ;如进位置位
        add     m8uH, mc8u      ;加被乘数到结果高字节
noad86: ror     m8uH            ;右移结果高字节
        ror     m8uL            ;结果低字节和乘数循环右移
        brcc    noad87          ;如进位置位
        add     m8uH, mc8u      ;加被乘数到结果高字节
noad87: ror     m8uH            ;右移结果高字节
        ror     m8uL            ;结果低字节和乘数循环右移
        ret
;*********************************************************
;* "mpy16u"16×16 位无符号乘法;
;* 这个子程序是两个 16 位寄存器变量 (mp16uH:mp16uL 和 mc16uH:mc16uL)相乘 。
;* 结果送至寄存器 m16u3 :m16u2 :m16u1 :m16u0。
;* Number of words       :105 + return
;* Number of cycles      :105 + return
;* Low registers used    :None
;* High registers used   :6 (mp16uL, mp16uH, mc16uL/m16u0, mc16uH/m16u1, m16u2, m16u3)
```

```
;*********************************************************
;***** 子程序寄存器变量
.def    mc16uL    =r16          ;被乘数低字节
.def    mc16uH    =r17          ;被乘数高字节
.def    mp16uL    =r18          ;乘数低字节
.def    mp16uH    =r19          ;乘数高字节
.def    m16u0     =r18          ;结果位 0 (LSB)
.def    m16u1     =r19          ;结果位 1
.def    m16u2     =r20          ;结果位 2
.def    m16u3     =r21          ;结果位 3 (MSB)
;***** 代码
mpy16u: clr       m16u3         ;清结果最高 2 字节
        clr       m16u2
        lsr       mp16uH        ;乘数低位循环
        ror       mp16uL        ;乘数高位循环
        brcc      noadd0        ;如进位置位
        add       m16u2, mc16uL ;加被乘数低位和结果的 2 字节
        adc       m16u3, mc16uH ;加被乘数高位和结果的 3 字节
noadd0: ror       m16u3         ;结果 3 字节右移
        ror       m16u2         ;结果 2 字节循环右移
        ror       m16u1         ;结果 1 字节和乘数高位循环
        ror       m16u0         ;结果 0 字节和乘数低位循环
        brcc      noadd1        ;如进位置位
        add       m16u2, mc16uL ;加被乘数低位和结果的 2 字节
        adc       m16u3, mc16uH ;加被乘数高位和结果的 3 字节
noadd1: ror       m16u3         ;结果 3 字节右移
        ror       m16u2         ;结果 2 字节循环右移
        ror       m16u1         ;结果 1 字节和乘数高位循环
        ror       m16u0         ;结果 0 字节和乘数低位循环
        brcc      noadd2        ;如进位置位
        add       m16u2, mc16uL ;加被乘数低位和结果的 2 字节
        adc       m16u3, mc16uH ;加被乘数高位和结果的 3 字节
noadd2: ror       m16u3         ;结果 3 字节右移
        ror       m16u2         ;结果 2 字节循环右移
        ror       m16u1         ;结果 1 字节和乘数高位循环
        ror       m16u0         ;结果 0 字节和乘数低位循环
        brcc      noadd3        ;如进位置位
        add       m16u2, mc16uL ;加被乘数低位和结果的 2 字节
        adc       m16u3, mc16uH ;加被乘数高位和结果的 3 字节
noadd3: ror       m16u3         ;结果 3 字节右移
        ror       m16u2         ;结果 2 字节循环右移
        ror       m16u1         ;结果 1 字节和乘数高位循环
        ror       m16u0         ;结果 0 字节和乘数低位循环
        brcc      noadd4        ;如进位置位
        add       m16u2, mc16uL ;加被乘数低位和结果的 2 字节
        adc       m16u3, mc16uH ;加被乘数高位和结果的 3 字节
noadd4: ror       m16u3         ;结果 3 字节右移
        ror       m16u2         ;结果 2 字节循环右移
        ror       m16u1         ;结果 1 字节和乘数高位循环
```

```
        ror     m16u0           ;结果 0 字节和乘数低位循环
        brcc    noadd5          ;如进位置位
        add     m16u2,mc16uL    ;加被乘数低位和结果的 2 字节
        adc     m16u3,mc16uH    ;加被乘数高位和结果的 3 字节
noadd5: ror     m16u3           ;结果 3 字节右移
        ror     m16u2           ;结果 2 字节循环右移
        ror     m16u1           ;结果 1 字节和乘数高位循环
        ror     m16u0           ;结果 0 字节和乘数低位循环
        brcc    noadd6          ;如进位置位
        add     m16u2,mc16uL    ;加被乘数低位和结果的 2 字节
        adc     m16u3,mc16uH    ;加被乘数高位和结果的 3 字节
noadd6: ror     m16u3           ;结果 3 字节右移
        ror     m16u2           ;结果 2 字节循环右移
        ror     m16u1           ;结果 1 字节和乘数高位循环
        ror     m16u0           ;结果 0 字节和乘数低位循环
        brcc    noadd7          ;如进位置位
        add     m16u2,mc16uL    ;加被乘数低位和结果的 2 字节
        adc     m16u3,mc16uH    ;加被乘数高位和结果的 3 字节
noadd7: ror     m16u3           ;结果 3 字节右移
        ror     m16u2           ;结果 2 字节循环右移
        ror     m16u1           ;结果 1 字节和乘数高位循环
        ror     m16u0           ;结果 0 字节和乘数低位循环
        brcc    noadd8          ;如进位置位
        add     m16u2,mc16uL    ;加被乘数低位和结果的 2 字节
        adc     m16u3,mc16uH    ;加被乘数高位和结果的 3 字节
noadd8: ror     m16u3           ;结果 3 字节右移
        ror     m16u2           ;结果 2 字节循环右移
        ror     m16u1           ;结果 1 字节和乘数高位循环
        ror     m16u0           ;结果 0 字节和乘数低位循环
        brcc    noadd9          ;如进位置位
        add     m16u2,mc16uL    ;加被乘数低位和结果的 2 字节
        adc     m16u3,mc16uH    ;加被乘数高位和结果的 3 字节
noadd9: ror     m16u3           ;结果 3 字节右移
        ror     m16u2           ;结果 2 字节循环右移
        ror     m16u1           ;结果 1 字节和乘数高位循环
        ror     m16u0           ;结果 0 字节和乘数低位循环
        brcc    noad10          ;如进位置位
        add     m16u2,mc16uL    ;加被乘数低位和结果的 2 字节
        adc     m16u3,mc16uH    ;加被乘数高位和结果的 3 字节
noad10: ror     m16u3           ;结果 3 字节右移
        ror     m16u2           ;结果 2 字节循环右移
        ror     m16u1           ;结果 1 字节和乘数高位循环
        ror     m16u0           ;结果 0 字节和乘数低位循环
        brcc    noad11          ;如进位置位
        add     m16u2,mc16uL    ;加被乘数低位和结果的 2 字节
        adc     m16u3,mc16uH    ;加被乘数高位和结果的 3 字节
noad11: ror     m16u3           ;结果 3 字节右移
        ror     m16u2           ;结果 2 字节循环右移
        ror     m16u1           ;结果 1 字节和乘数高位循环
```

```
        ror     m16u0              ;结果 0 字节和乘数低位循环
        brcc    noad12             ;如进位置位
        add     m16u2, mc16uL      ;加被乘数低位和结果的 2 字节
        adc     m16u3, mc16uH      ;加被乘数高位和结果的 3 字节
noad12: ror     m16u3              ;结果 3 字节右移
        ror     m16u2              ;结果 2 字节循环右移
        ror     m16u1              ;结果 1 字节和乘数高位循环
        ror     m16u0              ;结果 0 字节和乘数低位循环
        brcc    noad13             ;如进位置位
        add     m16u2, mc16uL      ;加被乘数低位和结果的 2 字节
        adc     m16u3, mc16uH      ;加被乘数高位和结果的 3 字节
noad13: ror     m16u3              ;结果 3 字节右移
        ror     m16u2              ;结果 2 字节循环右移
        ror     m16u1              ;结果 1 字节和乘数高位循环
        ror     m16u0              ;结果 0 字节和乘数低位循环
        brcc    noad14             ;如进位置位
        add     m16u2, mc16uL      ;加被乘数低位和结果的 2 字节
        adc     m16u3, mc16uH      ;加被乘数高位和结果的 3 字节
noad14: ror     m16u3              ;结果 3 字节右移
        ror     m16u2              ;结果 2 字节循环右移
        ror     m16u1              ;结果 1 字节和乘数高位循环
        ror     m16u0              ;结果 0 字节和乘数低位循环
        brcc    noad15             ;如进位置位
        add     m16u2, mc16uL      ;加被乘数低位和结果的 2 字节
        adc     m16u3, mc16uH      ;加被乘数高位和结果的 3 字节
noad15: ror     m16u3              ;结果 3 字节右移
        ror     m16u2              ;结果 2 字节循环右移
        ror     m16u1              ;结果 1 字节和乘数高位循环
        ror     m16u0              ;结果 0 字节和乘数低位循环
        ret
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* "div8u"8/8 位无符号除法
;*  该子程序是两个寄存器变量 dd8u (被除数)和 dv8u (除数)相除。
;*  结果送至寄存器 dres8u, 余数送至寄存器 drem8u。
;*  Number of words :66 + return
;*  Number of cycles      :50/58/66 (Min/Avg/Max) + return
;*  Low registers used    :1 (drem8u)
;*  High registers used   :2 (dres8u/dd8u, dv8u)
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* * * * * 子程序寄存器变量
.def    drem8u    = r15            ;余数
.def    dres8u    = r16            ;结果
.def    dd8u      = r16            ;被除数
.def    dv8u      = r17            ;除数
;* * * * * 代码
div8u:  sub     drem8u, drem8u     ;清除余数和进位
        rol     dd8u               ;除数左移
        rol     drem8u             ;除数移到余数寄存器
        sub     drem8u, dv8u       ;余数 = 余数 - 除数
```

```
        brcc    d8u_1           ;如结果为负
        add     drem8u, dv8u    ;存储余数
        clc                     ;清进位并移到结果
        rjmp    d8u_2           ;否则
d8u_1:  sec                     ;置进位并移到结果
d8u_2:  rol     dd8u            ;除数左移
        rol     drem8u          ;除数移到余数寄存器
        sub     drem8u, dv8u    ;余数=余数-除数
        brcc    d8u_3           ;如结果为负
        add     drem8u, dv8u    ;存储余数
        clc                     ;清进位并移到结果
        rjmp    d8u_4           ;否则
d8u_3:  sec                     ;置进位并移到结果
d8u_4:  rol     dd8u            ;除数左移
        rol     drem8u          ;除数移到余数寄存器
        sub     drem8u, dv8u    ;余数=余数-除数
        brcc    d8u_5           ;如结果为负
        add     drem8u, dv8u    ;存储余数
        clc                     ;清进位并移到结果
        rjmp    d8u_6           ;否则
d8u_5:  sec                     ;置进位并移到结果
d8u_6:  rol     dd8u            ;除数左移
        rol     drem8u          ;除数移到余数寄存器
        sub     drem8u, dv8u    ;余数=余数-除数
        brcc    d8u_7           ;如结果为负
        add     drem8u, dv8u    ;存储余数
        clc                     ;清进位并移到结果
        rjmp    d8u_8           ;否则
d8u_7:  sec                     ;置进位并移到结果
d8u_8:  rol     dd8u            ;除数左移
        rol     drem8u          ;除数移到余数寄存器
        sub     drem8u, dv8u    ;余数=余数-除数
        brcc    d8u_9           ;如结果为负
        add     drem8u, dv8u    ;存储余数
        clc                     ;清进位并移到结果
        rjmp    d8u_10          ;否则
d8u_9:  sec                     ;置进位并移到结果
d8u_10: rol     dd8u            ;除数左移
        rol     drem8u          ;除数移到余数寄存器
        sub     drem8u, dv8u    ;余数=余数-除数
        brcc    d8u_11          ;如结果为负
        add     drem8u, dv8u    ;存储余数
        clc                     ;清进位并移到结果
        rjmp    d8u_12          ;否则
d8u_11: sec                     ;置进位并移到结果
d8u_12: rol     dd8u            ;除数左移
        rol     drem8u          ;除数移到余数寄存器
        sub     drem8u, dv8u    ;余数=余数-除数
        brcc    d8u_13          ;如结果为负
```

```
        add     drem8u, dv8u    ;存储余数
        clc                     ;清进位并移到结果
        rjmp    d8u_14          ;否则
d8u_13: sec                     ;置进位并移到结果
d8u_14: rol     dd8u            ;除数左移
        rol     drem8u          ;除数移到余数寄存器
        sub     drem8u, dv8u    ;余数=余数-除数
        brcc    d8u_15          ;如结果为负
        add     drem8u, dv8u    ;存储余数
        clc                     ;清进位并移到结果
        rjmp    d8u_16          ;否则
d8u_15: sec                     ;置进位并移到结果
d8u_16: rol     dd8u            ;除数左移
        ret
;*********************************************************************
;*"div16u"16/16 位无符号除法
;* 该子程序是两个 16 位数"dd16uH:dd16uL"(被除数)和"dv16uH:dv16uL"(除数)相除。
;* 结果送至"dres16uH:dres16uL",余数送至"drem16uH:drem16uL"。
;*Number of words :196 + return
;*Number of cycles                :148/173/196 (Min/Avg/Max)
;*Low registers used              :2 (drem16uL, drem16uH)
;*High registers used             :4 (dres16uL/dd16uL, dres16uH/dd16uH, dv16uL, dv16uH)
;*********************************************************************
;***** 子程序寄存器变量
.def    drem16uL    =r14            .def    dd16uL      =r16
.def    drem16uH    =r15            .def    dd16uH      =r17
.def    dres16uL    =r16            .def    dv16uL      =r18
.def    dres16uH    =r17            .def    dv16uH      =r19
;***** 代码
div16u:     clr     drem16uL        ;清除余数低字节
            sub     rem16uH, drem16uH   ;清除余数高字节和进位
            rol     d16uL           ;除数左移
            rol     d16uH
            rol     rem16uL         ;除数移到余数寄存器
            rol     rem16uH
            sub     rem16uL, dv16uL ;余数=余数-除数
            sbc     rem16uH, dv16uH
            brcc    16u_1           ;如结果为负
            add     rem16uL, dv16uL ;存储余数
            adc     rem16uH, dv16uH
            clc                     ;清进位并移到结果
            rjmp    16u_2           ;否则
d16u_1:     sec                     ;置进位并移到结果
d16u_2:     rol     d16uL           ;除数左移
            rol     d16uH
            rol     rem16uL         ;除数移到余数寄存器
            rol     rem16uH
            sub     rem16uL, dv16uL ;余数=余数-除数
            sbc     rem16uH, dv16uH
```

```
            brcc    d16u_3              ;如结果为负
            add     drem16uL, dv16uL    ;存储余数
            adc     drem16uH, dv16uH
            clc                         ;清进位并移到结果
            rjmp    d16u_4              ;否则
d16u_3:     sec                         ;置进位并移到结果
d16u_4:     rol     dd16uL              ;除数左移
            rol     dd16uH
            rol     drem16uL            ;除数移到余数寄存器
            rol     drem16uH
            sub     drem16uL, dv16uL    ;余数=余数-除数
            sbc     drem16uH, dv16uH
            brcc    d16u_5              ;如结果为负
            add     drem16uL, dv16uL    ;存储余数
            adc     drem16uH, dv16uH
            clc                         ;清进位并移到结果
            rjmp    d16u_6              ;否则
d16u_5:     sec                         ;置进位并移到结果
d16u_6:     rol     dd16uL              ;除数左移
            rol     dd16uH
            rol     drem16uL            ;除数移到余数寄存器
            rol     drem16uH
            sub     drem16uL, dv16uL    ;余数=余数-除数
            sbc     drem16uH, dv16uH
            brcc    d16u_7              ;如结果为负
            add     drem16uL, dv16uL    ;存储余数
            adc     drem16uH, dv16uH
            clc                         ;清进位并移到结果
            rjmp    d16u_8              ;否则
d16u_7:     sec                         ;置进位并移到结果
d16u_8:     rol     dd16uL              ;除数左移
            rol     dd16uH
            rol     drem16uL            ;除数移到余数寄存器
            rol     drem16uH
            sub     drem16uL, dv16uL    ;余数=余数-除数
            sbc     drem16uH, dv16uH
            brcc    d16u_9              ;如结果为负
            add     drem16uL, dv16uL    ;存储余数
            adc     drem16uH, dv16uH
            clc                         ;清进位并移到结果
            rjmp    d16u_10             ;否则
d16u_9:     sec                         ;置进位并移到结果
d16u_10:    rol     dd16uL              ;除数左移
            rol     dd16uH
            rol     drem16uL            ;除数移到余数寄存器
            rol     drem16uH
            sub     drem16uL, dv16uL    ;余数=余数-除数
            sbc     drem16uH, dv16uH
            brcc    d16u_11             ;如结果为负
```

```
                add     drem16uL, dv16uL    ;存储余数
                adc     drem16uH, dv16uH
                clc                         ;清进位并移到结果
                rjmp    d16u_12             ;否则
d16u_11:        sec                         ;置进位并移到结果
d16u_12:        rol     dd16uL              ;除数左移
                rol     dd16uH
                rol     drem16uL            ;除数移到余数寄存器
                rol     drem16uH
                sub     drem16uL, dv16uL    ;余数=余数-除数
                sbc     drem16uH, dv16uH
                brcc    d16u_13             ;如结果为负
                add     drem16uL, dv16uL    ;存储余数
                adc     drem16uH, dv16uH
                clc                         ;清进位并移到结果
                rjmp    d16u_14             ;否则
d16u_13:        sec                         ;置进位并移到结果
d16u_14:        rol     dd16uL              ;除数左移
                rol     dd16uH
                rol     drem16uL            ;除数移到余数寄存器
                rol     drem16uH
                sub     drem16uL, dv16uL    ;余数=余数-除数
                sbc     drem16uH, dv16uH
                brcc    d16u_15             ;如结果为负
                add     drem16uL, dv16uL    ;存储余数
                adc     drem16uH, dv16uH
                clc                         ;清进位并移到结果
                rjmp    d16u_16             ;否则
d16u_15:        sec                         ;置进位并移到结果
d16u_16:        rol     dd16uL              ;除数左移
                rol     dd16uH
                rol     drem16uL            ;除数移到余数寄存器
                rol     drem16uH
                sub     drem16uL, dv16uL    ;余数=余数-除数
                sbc     drem16uH, dv16uH
                brcc    d16u_17             ;如结果为负
                add     drem16uL, dv16uL    ;存储余数
                adc     drem16uH, dv16uH
                clc                         ;清进位并移到结果
                rjmp    d16u_18             ;否则
d16u_17:        sec                         ;置进位并移到结果
d16u_18:        rol     dd16uL              ;除数左移
                rol     dd16uH
                rol     drem16uL            ;除数移到余数寄存器
                rol     drem16uH
                sub     drem16uL, dv16uL    ;余数=余数-除数
                sbc     drem16uH, dv16uH
                brcc    d16u_19             ;如结果为负
                add     drem16uL, dv16uL    ;存储余数
```

```
            adc     drem16uH, dv16uH
            clc                         ;清进位并移到结果
            rjmp    d16u_20             ;否则
d16u_19:    sec                         ;置进位并移到结果
d16u_20:    rol     dd16uL              ;除数左移
            rol     dd16uH
            rol     drem16uL            ;除数移到余数寄存器
            rol     drem16uH
            sub     drem16uL, dv16uL    ;余数=余数-除数
            sbc     drem16uH, dv16uH
            brcc    d16u_21             ;如结果为负
            add     drem16uL, dv16uL    ;存储余数
            adc     drem16uH, dv16uH
            clc                         ;清进位并移到结果
            rjmp    d16u_22             ;否则
d16u_21:    sec                         ;置进位并移到结果
d16u_22:    rol     dd16uL              ;除数左移
            rol     dd16uH
            rol     drem16uL            ;除数移到余数寄存器
            rol     drem16uH
            sub     drem16uL, dv16uL    ;余数=余数-除数
            sbc     drem16uH, dv16uH
            brcc    d16u_23             ;如结果为负
            add     drem16uL, dv16uL    ;存储余数
            adc     drem16uH, dv16uH
            clc                         ;清进位并移到结果
            rjmp    d16u_24             ;否则
d16u_23:    sec                         ;置进位并移到结果
d16u_24:    rol     dd16uL              ;除数左移
            rol     dd16uH
            rol     drem16uL            ;除数移到余数寄存器
            rol     drem16uH
            sub     drem16uL, dv16uL    ;余数=余数-除数
            sbc     drem16uH, dv16uH
            brcc    d16u_25             ;如结果为负
            add     drem16uL, dv16uL    ;存储余数
            adc     drem16uH, dv16uH
            clc                         ;清进位并移到结果
            rjmp    d16u_26             ;否则
d16u_25:    sec                         ;置进位并移到结果
d16u_26:    rol     dd16uL              ;除数左移
            rol     dd16uH
            rol     drem16uL            ;除数移到余数寄存器
            rol     drem16uH
            sub     drem16uL, dv16uL    ;余数=余数-除数
            sbc     drem16uH, dv16uH
            brcc    d16u_27             ;如结果为负
            add     drem16uL, dv16uL    ;存储余数
            adc     drem16uH, dv16uH
```

```
            clc                         ;清进位并移到结果
            rjmp    d16u_28             ;否则
d16u_27:    sec                         ;置进位并移到结果
d16u_28:    rol     dd16uL              ;除数左移
            rol     dd16uH
            rol     drem16uL            ;除数移到余数寄存器
            rol     drem16uH
            sub     drem16uL, dv16uL    ;余数 = 余数 - 除数
            sbc     drem16uH, dv16uH
            brcc    d16u_29             ;如结果为负
            add     drem16uL, dv16uL    ;存储余数
            adc     drem16uH, dv16uH
            clc                         ;清进位并移到结果
            rjmp    d16u_30             ;否则
d16u_29:    sec                         ;置进位并移到结果
d16u_30:    rol     dd16uL              ;除数左移
            rol     dd16uH
            rol     drem16uL            ;除数移到余数寄存器
            rol     drem16uH
            sub     drem16uL, dv16uL    ;余数 = 余数 - 除数
            sbc     drem16uH, dv16uH
            brcc    d16u_31             ;如结果为负
            add     drem16uL, dv16uL    ;存储余数
            adc     drem16uH, dv16uH
            clc                         ;清进位并移到结果
            rjmp    d16u_32             ;否则
d16u_31:    sec                         ;置进位并移到结果
d16u_32:    rol     dd16uL              ;除数左移
            rol     dd16uH
            ret
;*********************************************************
;* 测试程序
;*********************************************************
;***** 主程序寄存器变量
.def        temp    =r16                ;暂存变量
;***** 代码
RESET:
;-----------------------------------------------------------
;如用 SRAM 除法应包括下列程序行:
;   ldi     temp, low(RAMEND)
;   out     SPL, temp
;   ldi     temp, high(RAMEND)
;   out     SPH, temp                   ;初始化堆栈
;-----------------------------------------------------------
;***** 两个无符号 8 位数乘 (250 * 4)
;   ldi     mc8u, 250
;   ldi     mp8u, 4
;   rcall   mpy8u                       ;结果: m8uH:m8uL = $03e8 (1 000)
;***** 两个无符号 16 位数乘 (5 050 * 10 000)
```

```
;       ldi     mc16uL, low(5050)
;       ldi     mc16uH, high(5050)
;       ldi     mp16uL, low(10000)
;       ldi     mp16uH, high(10000)
;       rcall   mpy16u                  ;结果: m16u3:m16u2:m16u1:m16u0 = $030291a0 (50500000)
;*****两个无符号 8 位数除 (100/3)
;       ldi     dd8u, 100
;       ldi     dv8u, 3
;       rcall   div8u                   ;结果: $21 (33)
                                        ;余数: $01 (1)
;*****两个无符号 16 位数除 (50 000/24 995)
;       ldi     dd16uL, low(50000)
;       ldi     dd16uH, high(50000)
;       ldi     dv16uL, low(24995)
;       ldi     dv16uH, high(24995)
;       rcall   div16u                  ;结果: $0002 (2)
                                        ;余数: $000a (10)
;       forever: rjmp forever
```

5.2.6 16 位运算

该应用程序列出了 16 位加法、16 位带立即数加法、16 位减法、16 位带立即数减法、16 位比较、16 位带立即数比较、16 位取反子程序和一个测试程序。

```
;**********************************************************
;* Title:        16 位运算
;* Target:       AT90SXXXX (All AVR Devices)
;* 该项应用列出以下加/减/比较运算:
;* "add16"       ADD 16 + 16
;* "adddi16"     ADD 16 + Immediate(16)
;* "sub16"       SUB 16 - 16
;* "subi16"      SUB 16 - Immediate(16)
;* "cp16"        COMPARE 16/16
;* "cpi16" COMPARE 16/Immediate
;* "neg16"       NEGATION 16
;**********************************************************
.cseg
        ldi   r16, 0x12                 ;设置一些寄存器用于下面子程序
        ldi   r17, 0x34
        ldi   r18, 0x56                 ;所有期望结果由注释呈现
        ldi   r19, 0x78
;**********************************************************
;* "add16" 16 位寄存器加法
;* 该例是两个寄存器变量(add1l, add1h)和(add2l, add2h)相加, 结果送至(add1l, add1h)。
;* Number of words :2
;* Number of cycles                :2
;* Low registers used              :None
;* High registers used             :4
;* 注意:和与被加数用相同的寄存器,因为被加数被和覆盖了。
;**********************************************************
```

```
;* * * * * 寄存器变量
.def    add1l    = r16
.def    add1h    = r17
.def    add2l    = r18
.def    add2h    = r19
;* * * * * * 代码
add16:  add      add1l, add2l          ;加低字节
        adc      add1h, add2h          ;加带进位高字节
                                       ;期望值为 0xAC68
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* "addi16" 16 位带立即数加法
;* 该例是一个寄存器变量(addi1l,addi1h)和由等式定义的 16 位立即数相加,结果送至寄存器(addi1l, addi1h)。
;* Number of words :2
;* Number of cycles                    :2
;* Low registers used                  :None
;* High registers used                 :2
;* 注意:和与被加数用相同的寄存器,因为被加数被和覆盖了。
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* * * * * 寄存器变量
.def    addi1l   = r16
.def    addi1h   = r17
;* * * * * 16 位立即数
.equ    addi2    = 0x1234
;* * * * * 代码
addi16: subi     add1l, low( - addi2)  ;加低字节 ( x -( - y)) = x + y
        sbci     add1h, high( - addi2) ;加带进位高字节
                                       ;期望值为 0xBE9C
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* "sub16" 16 位寄存器减法
;* 该例是两个双寄存器变量(sub1l,sub1h)和(sub2l, sub2h)相减,结果送至寄存器(sub1l, sub1h)。
;* Number of words :2
;* Number of cycles                    :2
;* Lowregisters used                   :None
;* High registers used                 :4
;* 注意:结果与"sub1"用相同寄存器,因为"sub1"被结果覆盖了。
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* * * * * 寄存器变量
.def    sub1l    = r16
.def    sub1h    = r17
.def    sub2l    = r18
.def    sub2h    = r19
;* * * * * 代码
sub16:  sub      sub1l, sub2l          ;减低字节
        sbc      sub1h, sub2h          ;加带进位高字节
                                       ;期望值为 0x4646
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* "subi16" 16 位寄存器与 16 位立即数减法
;* 该例是从 16 位寄存器 (subi1l, subi1h) 中减去 16 位立即数 (subi2),结果送至寄存器; (subi1l, subi1h)。
;* Number of words :2
```

```
; * Number of cycles                    :2
; * Low registers used                  :None
; * High registers used                 :2
; * 注意:结果与"subi1"用相同寄存器,因为"subi1"被结果覆盖了。
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * 寄存器变量
.def        subi1l      = r16
.def        subi1h      = r17
; * * * * * 16 位立即数
.equ        subi2       = 0x1234
; * * * * * 代码
subi16:     subi        subi1l, low(subi2)      ;减低字节
            sbci        subi1h, high(subi2)     ;减带进位高字节
                                                ;期望值为 0x3412
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * "cp16" 16 位数比较
; * 该例是对寄存器对(cp1l, cp1h)和(cp2l, cp2h)进行比较,如相等零标志置 1,否则清 0
; * Number of words:2
; * Number of cycles                    :2
; * Low registers used                  :None
; * High registers used                 :4
; * 注意:"cp1"的内容将被覆盖。
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * 寄存器变量
.def        cp1l        = r16
.def        cp1h        = r17
.def        cp2l        = r18
.def        cp2h        = r19
; * * * * * 代码
cp16:       cp          cp1l, cp2l              ;比较低字节
            cpc         cp1h, cp2h              ;比较先前操作中带进位高字节
ncp16:
                                                ;期望值为 Z=0
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * " cpi16" 16 位寄存器与 16 位立即数比较
; * 该例是对寄存器对(cpi1l, cpi1h)和 cpi2 的值进行比较,如相等零标志置 1,否则清 0。这是由 AVR 的零
; * 函数使能的。如结果是负,则进位被置 1,这意味着在比较之后,所有的条件转移指令都有用。
; * Number of words :3
; * Number of cycles                    :3
; * Low registers used                  :None
; * High registers used                 :3
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * 寄存器变量
.def        cp1l        = r16
.def        cp1h        = r17
.def        c_tmp       = r18
.equ        cp2         = 0x3412                ;与立即数比较
; * * * * * 代码
cpi16:      cpi         cp1l, low(cp2)          ;比较低字节
```

```
        ldi     c_tmp,high(cp2)
        cpc     cp1h,c_tmp              ;比较高字节
                                        ;期望值为 Z=1
;*****************************************************************
;* "neg16"16 位寄存器求反
;* 该例是对寄存器对(ng1l,ng1h)求反,其结果将覆盖该寄存器对。
;* Number of words :4
;* Number of cycles                    :4
;* Low registers used                  :None
;* High registers used                 :2
;*****************************************************************
;***** 寄存器变量
.def    ng1l    = r16
.def    ng1h    = r17
;***** 代码
ng16:   com     ng1l                    ;低字节取反
        com     ng1h                    ;高字节取反
        subi    ng1l,low(-1)            ;加低字节
        sbci    ng1h,high(-1)           ;加高字节
                                        ;期望值为 0xCBEE
forever:rjmp forever
```

5.2.7 BCD 运算

该应用程序列出了 16 位二进制转换成 BCD、8 位二进制转换成 BCD、BCD 转换成 16 位二进制、BCD 转换成 8 位二进制、二位数压缩 BCD 加法、二位数压缩 BCD 减法和一个测试程序。

```
;*****************************************************************
;* Title:          BCD 算法
;* Target:         AT90SXXXX (All AVR Devices)
;* 该项应用列出二进制编码的十进制算法应用:
;* Binary 16 to BCD Conversion (special considerations for AT90SXX0X)
;* Binary 8 to BCD Conversion
;* BCD to Binary 16 Conversion
;* BCD to Binary 8 Conversion
;* 2-Digit BCD Addition
;* 2-Digit BCD Subtraction
;*****************************************************************
.include "8515def.inc"
    rjmp    RESET                       ;复位处理
;*****************************************************************
;* "bin2BCD16" 16 位二进制到 BCD 转换
;* 该子程序是 16 位数(fbinH:fbinL)转换为 5 位压缩 BCD 数,用 3 个字节(tBCD2:tBCD1: tBCD0)表示,5
;* 位 BCD 最高有效位放在 tBCD2 的最低半字节中。
;* Number of words :25
;* Number of cycles                    :751/768 (Min/Max)
;* Low registers used                  :3 (tBCD0,tBCD1,tBCD2)
;* High registers used                 :4(fbinL,fbinH,cnt16a,tmp16a)
;* Pointers used                       :Z
;*****************************************************************
```

```
;* * * * * 子程序寄存器变量
.equ    AtBCD0  = 13            ;tBCD0 的地址
.equ    AtBCD2  = 15            ;tBCD1 的地址
.def    tBCD0   = r13           ;BCD 值数字 1 和 0
.def    tBCD1   = r14           ;BCD 值数字 3 和 2
.def    tBCD2   = r15           ;BCD 值数字 4
.def    fbinL   = r16           ;二进制值低字节
.def    fbinH   = r17           ;二进制值高字节
.def    cnt16a  = r18           ;循环计数器
.def    tmp16a  = r19           ;暂存值
;* * * * * 代码
bin2BCD16:
        ldi     cnt16a,16       ;初始化循环计数器
        clr     tBCD2           ;清结果(3 字节)
        clr     tBCD1
        clr     tBCD0
        clr     ZH              ;清 ZH (AT90SXX0X 不需要)
bBCDx_1: lsl    fbinL           ;移位输入值
        rol     fbinH           ;所有位循环
        rol     tBCD0
        rol     tBCD1
        rol     tBCD2
        dec     cnt16a          ;减循环计数器
        brne    bBCDx_2         ;如计数器不为 0
        ret                     ;返回
bBCDx_2: ldi    r30,AtBCD2+1    ;Z 指针指向结果 MSB + 1
bBCDx_3: ld     tmp16a,-Z       ;获得预减的 Z
;-----------------------------------------------------------
; 对于 AT90SXX0X,替换下列行:
;       dec     ZL
;       ld      tmp16a,Z
;-----------------------------------------------------------
        subi    tmp16a,$03      ;加 0x03
        sbrc    tmp16a,3        ;如位 3 不清除
        st      Z,tmp16a        ;存储返回
        ld      tmp16a,Z        ;获得 Z
        subi    tmp16a,-$30     ;加 0x30
        sbrc    tmp16a,7        ;如位 3 不清除
        st      Z,tmp16a        ;存储返回
        cpi     ZL,AtBCD0       ;所有完成否?
        brne    bBCDx_3         ;如无再循环
        rjmp    bBCDx_1
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* "bin2BCD8" 8 位二进制到 BCD 转换
;* 该子程序是 8 位数(fbin)转换成 2 位 BCD 数(tBCDH:tBCDL)。
;* Number of words :6 + return
;* Number of cycle                :5/50 (Min/Max) + return
;* Low registers used             :None
;* High registers used            :2 (fbin/tBCDL,tBCDH)
```

```
;* 在代码行中加代替压缩 BCD 输出的程序行。
;****************************************************************
;***** 子程序寄存器变量
.def    fbin     =r16              ;8 位二进制值
.def    tBCDL    =r16              ;BCD 结果最高有效值
.def    tBCDH    =r17              ;BCD 结果最低有效值
;***** 代码
bin2BCD8:
        clr      tBCDH             ;清结果最高有效位
bBCD8_1:subi     fbin,10           ;输出=输出-10
        brcs     bBCD8_2           ;如进位置位放弃
        inc      tBCDH             ;加最高有效位
;----------------------------------------------------------------
;                                  ;这行程序代替上面行作为压缩 BCD 输出
;       subi tBCDH, -$10           ;tBCDH= tBCDH + 10
;----------------------------------------------------------------
        rjmp     bBCD8_1           ;继续循环
bBCD8_2:subi     fbin,-10          ;补偿额外部分
;----------------------------------------------------------------
;       add      fbin,tBCDH        ;加这行程序作为 BCD 输出
;----------------------------------------------------------------
        ret
;****************************************************************
;* "BCD2bin16" BCD 到 16 位二进制转换
;* 该子程序是用 3 字节 (fBCD2:fBCD1:fBCD0) 表示 5 位压缩 BCD 码转换成 16 位二进制数 (tbinH:
;* tbinL)。5 位数的最高有效位必须放在 fBCD2 的最低半字节。
;* "abcde"代表 5 位数。转换公式为:10(10(10(10a+b)+c)+d)+e。
;* 该子程序在运算期间用 4 次乘和加运算来代替。
;* Number of words :30
;* Number of cycles                :108
;* Low registers used              :4 (copyL,copyH,mp10L/tbinL,mp10H/tbinH)
;* High registers used             :4 (fBCD0,fBCD1,fBCD2,adder)
;****************************************************************
;***** "mul10a"/"mul10b"子程序寄存器变量
.def    copyL    =r12              ;暂存寄存器
.def    copyH    =r13              ;暂存寄存器
.def    mp10L    =r14              ;乘数的低字节乘 10
.def    mp10H    =r15              ;乘数的高字节乘 10
.def    adder    =r19              ;相乘后的数值相加
;***** 代码
mul10a: ***** 带 10 的乘法并加到加法器高半字节
mul10b: ***** 带 10 的乘法并加到加法器低半字节
        mov      copyL,mp10L       ;拷贝
        mov      copyH,mp10H
        lsl      mp10L             ;原始值乘 2
        rol      mp10H
        lsl      copyL             ;拷贝值乘 2
        rol      copyH
        lsl      copyL             ;拷贝值再乘 2
```

```
        rol     copyH
        lsl     copyL               ;拷贝值再乘 2
        rol     copyH
        add     mp10L, copyL        ;拷贝值和原始值相加
        adc     mp10H, copyH
        andi    adder, 0x0f         ;屏蔽加法器的上半字节
        add     mp10L, adder        ;加加法器的下半字节
        brcc    m10_1               ;如进位没被清
        inc     mp10H               ;高字节加 1
m10_1:  ret
;***** 主程序寄存器变量
.def    tbinL   =r14                ;二进制结果的低字节(同 mp10L)
.def    tbinH   =r15                ;二进制结果的高字节(同 mp10H)
.def    fBCD0   =r16                ;BCD 值数字 1 和 0
.def    fBCD1   =r17                ;BCD 值数字 3 和 2
.def    fBCD2   =r18                ;BCD 值数字 5
;***** 代码
BCD2bin16:
        andi    fBCD2, 0x0f         ;屏蔽 fBCD2 的上半字节
        clr     mp10H
        mov     mp10L, fBCD2        ;mp10H:mp10L= a
        mov     adder, fBCD1
        rcall   mul10a              ;mp10H:mp10L= 10a+b
        mov     adder, fBCD1
        rcall   mul10b              ;mp10H:mp10L= 10(10a+b)+c
        mov     adder, fBCD0
        rcall   mul10a              ;mp10H:mp10L= 10(10(10a+b)+c)+d
        mov     adder, fBCD0
        rcall   mul10b              ;mp10H:mp10L= 10(10(10(10a+b)+c)+d)+e
        ret
;*************************************************************
;* "BCD2bin8" BCD 到 8 位二进制转换
;* 该子程序是将 2 位 BCD 码(fBCDH:fBCDL)转换为 8 位二进制数(tbin)。
;* Number of words :4 + return
;* Number of cycles                  :3/48 (Min/Max) + return
;* Low registers used                :None
;* High registers used               :2 (tbin/fBCDL, fBCDH)
;* 在代码注释中包括了使用压缩 BCD 码的转换程序。如果使用,则 fBCDH 将装载转换前的 BCD 数,再
;* 调用子程序。
;*************************************************************
;***** 子程序寄存器变量
.def    tbi     =r16                ;二进制结果
.def    fBCDL   =r16                ;BCD 输入的低数字
.def    fBCDH   =r17                ;BCD 输入的高数字
;***** 代码
BCD2bin8:
;*************************************************************
;*      mov     tbin, fBCDH         ;作为压缩 BCD 输入,加这些行拷贝输入到结果
;*      andi    tbin, $0f           ;清除结果的最高半字节
```

```
;*****************************************************************
BCDb8_0:subi    fBCDH,1             ;fBCDH= fBCDH - 1
        brcs    BCDb8_1             ;如进位不置位
;*****************************************************************
;*                                  ;作为压缩 BCD 输入,用这些行代替上两行程序
;*      subi    fBCDH, $10          ;MSD= MSD - 1
;*      brmi    BCDb8_1             ;如果 0 标志不置位
;*****************************************************************
        subi    tbin, -10           ;结果=结果+10
        rjmp    BCDb8_0             ;再循环
BCDb8_1:ret                         ;否则返回
;*****************************************************************
;* "BCDadd" 2 位压缩 BCD 数加法
;* 该子程序是两个无符号 2 位 BCD 数(BCD1 和 BCD2)相加,结果置入"BCD1",进位溢出置入"BCD2"。
;* Number of words :19
;* Number of cycles                 :17/20 (Min/Max)
;* Low registers used               :None
;* High registers used              :3 (BCD1, BCD2, tmpadd)
;*****************************************************************
;***** 子程序寄存器变量
.def    BCD1    =r16                ;BCD 输入值#1
.def    BCD2    =r17                ;BCD 输入值#2
.def    tmpadd  =r18                ;暂存寄存器
;***** 代码
BCDadd:
        ldi     tmpadd,6            ;相加后的值
        add     BCD1,BCD2           ;二进制数相加
        clr     BCD2                ;清 BCD 进位位
        brcc    add_0               ;如进位不清除
        ldi     BCD2,1              ;置 BCD 进位
add_0:  brhs    add_1               ;如半进位不置位
        add     BCD1,tmpadd         ;最低有效位加 6
        brhs    add_2               ;如半进位不置位 (LSD <= 9)
        subi    BCD1,6              ;存储值
        rjmp    add_2               ;否则
add_1:  add     BCD1,tmpadd         ;最低有效位加 6
add_2:  swap    tmpadd
        add     BCD1,tmpadd         ;最高有效位加 6
        brcs    add_4               ;如进位不置位(MSD <= 9)
        sbrs    BCD2,0              ;如前进位不置位
        subi    BCD1,$60            ;存储值
add_3:  ret                         ;否则
add_4:  ldi     BCD2,1              ;置位 BCD 进位
        ret
;*****************************************************************
;* "BCDsub" 2 位压缩 BCD 数减法
;* 该子程序是两个无符号 BCD 数(BCDa 和 BCDb)相减,结果置入"BCDa",进位下溢出置入"BCDb"。
;* Number of words :13
;* Number of cycles                 :12/17 (Min/Max)
```

```
;* Low registers used                    :None
;* High registers used                   :2 (BCDa, BCDb)
;*********************************************************************
;***** 子程序寄存器变量

.def    BCDa    =r16                     ;BCD 输入值#1
.def    BCDb    =r17                     ;BCD 输入值#2
;***** 代码
BCDsub:
        sub     BCDa, BCDb               ;二进制数相减
        clr     BCDb
        brcc    sub_0                    ;如进位不清除
        ldi     BCDb, 1                  ;存储进位于 BCDb
1sub_0: brhc    sub_1                    ;如半进位不清除
        subi    BCDa, $06                ;LSD= LSD - 6
sub_1:  sbrs    BCDb, 0                  ;如前进位不置位
        ret                              ;返回
        subi    BCDa, $60                ;最高有效位减 6
        ldi     BCDb, 1                  ;置下溢出进位
        brcc    sub_2                    ;如进位不清除
        ldi     BCDb, 1                  ;清下溢出进位
sub_2:  ret
;*********************************************************************
;* 测试程序:该程序所调用的所有子程序都经过正确校验。
;*********************************************************************
;***** 主程序寄存器变量
.def    temp    =r16                     ;暂存存储变量
;***** 代码
复位:
        ldi     temp, low(RAMEND)
        out     SPL, temp
        ldi     temp, high(RAMEND)
        out     SPH, temp                ;初始化堆栈指针 (remove for AT90SXX0X)
;***** 转换 54 321 到 2.5 字节的压缩 BCD 格式
        ldi     fbinL, low(54321)
        ldi     fbinH, high(54321)
        rcall   bin2BCD16                ;结果: tBCD2:tBCD1:tBCD0= $054321
;***** 转换 55 到 2 字节 BCD
        ldi     fbin, 55
        rcall   bin2BCD8                 ;结果: tBCDH:tBCDL= 0505
;***** 转换 $065535 到 16 位二进制数
        ldi     fBCD2, $06
        ldi     fBCD1, $55
        ldi     fBCD0, $35
        rcall   BCD2bin16                ;结果: tbinH:tbinL= $ffff (65 535)
;***** 转换 $0403 (43)到 6 位二进制数
        ldi     fBCDL, 3
        ldi     fBCDH, 4
        rcall   BCD2bin8                 ;结果: tbin= $2b (43)
```

```
; * * * * * BCD数51与79相加
          ldi       BCD1, $51
          ldi       BCD2, $79
          rcall     BCDadd                 ;结果：BCD2:BCD1 = $0130
; * * * * * BCD数72减28
          ldi       BCDa, $72
          ldi       BCDb, $28
          rcall     BCDsub                 ;结果：BCDb = $00 (正结果), BCDa=44
; * * * * * BCD数0减90
          ldi       BCDa, $00
          ldi       BCDb, $90
          rcall     BCDsub                 ;结果：BCDb = $01 (负结果), BCDa=10
forever:rjmp forever
```

5.2.8 冒泡分类算法

该应用程序列出了如何用代码有效冒泡分类算法分类SRAM的数据块子程序和一个测试程序。

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * Title:          冒泡分类算法
; * Target:         AT90SXX1X (Devices with SRAM)
; * 该程序列出如何用代码有效冒泡分类算法分类SRAM的块,在程序中包括了一个测试程序将程序存储器的60字节数据块拷贝到SRAM中进行数据分类。
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
.include "8515def.inc"
.equ      SIZE      =60                    ;数据块尺寸
.equ      TABLE_L= $60                     ;第一个数据元素的低SRAM地址
.equ      TABLE_H= $00                     ;第一个数据元素的高SRAM地址
          rjmp   RESET                     ;复位处理
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * 冒泡法
; * 该子程序磁泡分类在"cnt1" + 1中能找到SRAM定位 "last"的最后元素的字节数。这个程序在SRAM最低地
; * 址处完成对最高元素的分类;分类指令能用改变"brlo";语句到"brsh" 来倒置,符号分类能用"brlt"或"brge"来获得。
; * Number of words :13 + return
; * Number of cycles                   ;6 * (SIZE - 1) + 10 * (SIZE(SIZE - 1)) + return (Min)
; *                                     6 * (SIZE - 1) + 13 * (SIZE(SIZE - 1)) + return (Max)
; * Low registers used                 ;3 (A, B, cnt2)
; * High registers used                ;3 (cnt1, endL, endH)
; * Pointers used                      ;Z
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * 子程序寄存器变量
.def      A         = r13              ;第一个被比较的值
.def      B         = r14              ;第二个被比较的值
.def      cnt2      = r15              ;内循环计数器
.def      cnt1      = r16              ;外循环计数器
.def      endL      = r17              ;数据进位低地址的结尾
.def      endH      = r18              ;数据进位高地址的结尾
; * * * * * 代码
bubble:
```

```
        mov     ZL, endL
        mov     ZH, endH            ;初始化 Z 指针
        mov     cnt2, cnt1          ;counter2 <- counter1
i_loop: ld      A, Z                ;获得第一个字节, A (n)
        ld      B, -Z               ;Z 减 1 并获得第二字节, B (n-1)
        cp      A, B                ;A 与 B 比较
        brlo    L1                  ;若 A 不低于
        st      Z, A                ;交换存储
        std     Z+1, B
L1:     dec     cnt2
        brne    i_loop              ;内部循环结束
        dec     cnt1
        brne    bubble              ;外部循环结束
        ret
;*********************************************************************
;*测试程序:该程序从程序存储器拷贝 60 字节数到 SRAM,然后调用"冒泡"程序对数据分类。
;*********************************************************************
RESET:
;*****主程序寄存器变量
.def    temp    =r16
;***** 代码
        ldi     temp, low(RAMEND)
        out     SPL, temp
        ldi     temp, high(RAMEND)
        out     SPH, temp           ;初始化堆栈指针
;**** 存储器填充
        clr     ZH
        ldi     ZL, tableend*2+1    ;Z-pointer <- ROM table end + 1
        ldi     YL, low(256*TABLE_H+TABLE_L+SIZE)
        ldi     YH, high(256*TABLE_H+TABLE_L+SIZE)
                                    ;Y pointer <- SRAM table end + 1
loop:   lpm                         ;获得 ROM 常数
        st      -Y, r0              ;存储到 SRAM 并且 Y 指针减 1
        sbiw    ZL, 1               ;Z 指针减 1
        cpi     YL, TABLE_L         ;若未完成
        brne    loop                ;再循环
        cpi     YH, TABLE_H
        brne    loop
;***** 分类数据
sort:   ldi     endL, low(TABLE_H*256+TABLE_L+SIZE-1)
        ldi     endH, high(TABLE_H*256+TABLE_L+SIZE-1)
                                    ;Z <- end of array address
        ldi     cnt1, SIZE-1        ;cnt1 <- size of array - 1
        rcall   bubble
forever:rjmp forever
;***** 60 个 ROM 常数
table:
.db     120, 196                    .db     78, 216
```

```
.db     78,100                          .db     252,102
.db     43,39                           .db     198,0
.db     241,43                          .db     171,239
.db     62,172                          .db     107,114
.db     109,69                          .db     172,170
.db     48,184                          .db     17,45
.db     215,231                         .db     42,55
.db     63,133                          .db     34,174
.db     216,8                           .db     229,250
.db     121,126                         .db     12,179
.db     188,98                          .db     187,243
.db     168,205                         .db     44,231
.db     157,172                         tableend:
.db     108,233                         .db     76,48
.db     80,255
```

5.2.9 设置和使用模拟比较器

该应用程序列出设置和使用模拟比较器的子程序。

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* Title: 设置和使用模拟比较器
;* Target: AT90SXXXX (Devices with Analog Comparator)
;* 这个例子说明怎样使能和使用 AVR 在片精密模拟比较器的一些特性。
;* 该例子说明完成下列任务所需的一个程序例子:等待一个由比较器输出转态的正输出边沿;等待一个由中断标
;* 志转态的正输出边沿;使能比较器输出触发中断。中断子程序每次执行时,加一个 16 位寄存器计数器。
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
.include "1 200def.inc"
;* * * * * 全局寄存器变量
.def    temp            = r16           ;暂存寄存器
.def    cntL            = r17           ;寄存器计数器低字节
.def    cntH            = r18           ;寄存器计数器高字节
;* * * * * 中断向量
        rjmp            RESET           ;复位处理
.org    ACIaddr
        rjmp            ANA_COMP        ;模拟比较器处理
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* "ANA_COMP"
;* 该中断子程序用于每次设置 ACSR 寄存器的 ACI,提供模拟比较器中断使能(ACIE 被设置),每次运行子
;* 程序加一个 16 位计数器。
;* Number of words :5
;* Number of cycles                        :8
;* Low registers used                      :1 (ac_tmp)
;* High registers used                     :2 (cntL, cntH)
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* * * * * 中断子程序寄存器变量
.def    ac_tmp          = r0            ;作为 SREG 的暂存寄存器
;* * * * * 代码
ANA_COMP:
        in              ac_tmp, SREG    ;暂存状态寄存器
```

```
        subi    cntL, low(-1)
        sbci    cntH, high(-1)      ;counter = counter + 1
        out     SREG, ac_tmp        ;存储状态寄存器
        reti
;*********************************************************
;* 程序开始执行
;*********************************************************
RESET:
;***** 如果使用的器件无 RAM 应包括下列程序行:
;       ldi     temp, low(RAMEND)
;       out     SPL, temp
;       ldi     temp, high(RAMEND)
;       out     SPH, temp
;*********************************************************
;* "wait_edge1"
;* 该程序段一直等待到比较器的输出为高,这样做不需要设置。但极短的脉冲可能错过,因为程序运行时在每
;* 次比较器被检测需三个时钟周期。另一个是程序需等待输出的第一个负脉冲到来,而在轮询时,输出是正的。
;* Number of words :4
;* Number of cycles                    :4
;* per loop. Response time             :3 - 5 clock cycles
;* Low registers used                  :None
;* High registers used                 :None
;*********************************************************
;***** 代码
wait_edge1:
        sbic    ACSR, ACO           ;如果输出是高
        rjmp    wait_edge1          ;等待
wel_1:  sbis    ACSR, ACO           ;如果输出是低
        rjmp    wel_1               ;等待
;*********************************************************
;* "wait_edge2"
;* 该程序段一直等到比较器的输出为高。由于中断标志被轮询,这是比较可靠的方法。这允许用户在等待循
;* 环期间插入指令,因为硬件能记忆此轮询区间较短宽度的脉冲。另外正的特性是不需要等待负边沿处理的。
;* Number of words :5
;* Number of cycles                    :Inital setup :2
;* Flag clearing                       :1
;* Loop                                :4
;* Response time                       :3 - 5
;* Low registers used                  :None
;* High registers used                 :None
*********************************************************
;***** 代码
wait_edge2:
;***** 初始化硬件配置(从复位后假定 ACIE = 0)
        sbi     ACSR, ACIS0
        sbi     ACSR, ACIS1         ;使能中断输出上升沿
;***** 等待
        sbi     ACSR, ACI           ;写"1"到 ACI 标志并清除
        we2_1:                      ;用户代码
```

```
        sbis        ACSR,ACI        ;如果 ACI 是低
        rjmp        we2_1           ;等待
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* "ana_init"
;* 这段程序使能模拟比较器输出触发中断,然后程序进入无穷循环。
;* 16 位计数器先前使能中断被清除。特征图仅用于中断初始化。
;* Number of words :4
;* Number of cycles                    :5
;* Low registers used                  :None
;* High registers used                 :1 (temp)
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* * * * * 寄存器变量
.def        temp            =r16        ;暂存寄存器
;* * * * * 代码
ana_init:
;* * * * * 清除 16 位计数器
        clr         cntL
        clr         cntH
;* * * * * 使能中断 (从复位后假定 ACIE= 0)
        ldi         temp,(ACI<<1)       ;清除中断标志和 ACIS1/ACIS0…
        out         ACSR,temp           ;到选择触发中断
        sei                             ;使能全局中断
        sbi         ACSR,ACIE           ;使能模拟比较器中断
forever:rjmp forever
```

5.2.10 8 点平均滤波

该应用程序为实现 8 点平均滤波的子程序。

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* Title:          8 点移动平均滤波器
;* Target:         AT90SXX1X (Devices with SRAM)
;* 该程序列出完成 8 点移动平均滤波器操作。该程序滤波一个包括开始元素在 T_START 的,范围为 SI
;* ZE 的时数据阵列。程序中包括一个测试程序,从程序存储器拷贝 60 字节块数据 SRAM 并滤波这数据。
;* 注意,最尾的一个数据阵列不参加滤波过程。
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* .include "8515def.inc"
.equ        SIZE            =60         ;数据阵列尺寸
.equ        TABLE_L         =$60        ;第一个数据元素的低 SRAM 地址
.equ        TABLE_H         =$00        ;第一个数据元素的高 SRAM 地址
        rjmp RESET                      ;复位处理
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* "mav8"
;* 该子程序滤波在 SRAM 中定位在 T_START 处开始的共 SIZE 字节的数据阵列,滤波器使用工作寄存
;* 器 R0~R7 的 8 字节环型缓冲器,在滤波过程中,该环型缓冲器总是包含取平均的 8 字节窗口。
;* Number of words :31 + return
;* Number of cycles                    :59 + (SIZE-7)*75 + return
;* Low registers used                  :11 (r0~r7,mav_tmp,AL,AH)
;* High registers used                 :1 (t_size)
;* Pointers used                       :X,Y,Z
```

```
;*********************************************************************
;***** 子程序寄存器变量
.def    mav_tmp      =r8            ;暂存寄存器
.def    AL           =r9            ; 环型缓冲器和低字节
.def    AH           =r10           ; 环型缓冲器和高字节
.def    t_size       =r16           ; 表的尺寸
;***** 代码
mav8:
        clr          YH
        clr          YL             ; 初始化指针
        clr          XH
        clr          XL
;***** 填充环型缓冲器的第一个8字节数据段
mav8_1: ld           mav_tmp,Z+     ; 获得SRAM数据
        st           Y+,mav_tmp     ; 存储在寄存器
        cpi          YL,8           ; 所有都获得?
        brne         mav8_1         ; 如没有,再循环
        sbiw         ZL,5           ; Z指针指向第一个滤波的值
mav8_2:
;***** 寻找平均值
        clr          AH             ; 清除平均值高字节
        clr          AL             ; 清除平均值低字节
        clr          YL             ; 初始化Y指针
mav8_3: ld           mav_tmp,Y+
        add          AL,mav_tmp     ; 加值到AL
        adc          AH,XH          ; 加进位到AH
        cpi          YL,8           ; 全加完?
        brne         mav8_3         ; 如没有,再循环
        lsr          AH             ; 除8
        ror          AL
        lsr          AH
        ror          AL
        lsr          AH
        ror          AL
        ldd          mav_tmp,Z+5    ; 得下一个缓冲器的值
        st           X+,mav_tmp     ; 存储缓冲器
        andi         XL,$07         ; 屏蔽指针的高位
        st           Z+,AL          ; 存储平均值
        dec          t_size
        cpi          t_size,4       ; 进位结束?
        brne         mav8_2         ; 如没有,再循环
        ret
;*********************************************************************
;* 测试程序:该程序从程序存储器拷贝60字节数据到SRAM,并调用"mav8"得到数据滤波。
;*********************************************************************
RESET:
;***** 主程序寄存器变量
.def    temp         =r16
;***** 代码
```

```
        ldi     temp, low(RAMEND)
        out     SPL, temp
        ldi     temp, high(RAMEND)
        out     SPH, temp               ;初始化堆栈指针
;***** 填充存储器
        clr     ZH
        ldi     ZL, tableend * 2 + 1    ;Z - pointer <- ROM table end + 1
        ldi     YL, low(256 * TABLE_H + TABLE_L + SIZE)
        ldi     YH, high(256 * TABLE_H + TABLE_L + SIZE)
                                        ;Y pointer <- SRAM table end + 1
loop:   lpm                             ;获得 ROM 常数
        st      -Y, r0                  ;储存到 SRAM, 并且 Y 指针减 1
        sbiw    ZL, 1                   ;Z 指针减 1
        cpi     YL, TABLE_L             ;如果未完成
        brne    loop                    ;继续循环
        cpi     YH, TABLE_H
        brne    loop
;***** 滤波数据
        ldi     ZL, TABLE_L
        ldi     ZH, TABLE_H
        ldi     T_size, SIZE
        rcall   mav8
forever: rjmp forever
;***** 60 个 ROM 内容
table:
.db     120, 196
.db     78, 216
.db     78, 100
.db     43, 39
.db     241, 43
.db     62, 172
.db     109, 69
.db     48, 184
.db     215, 231
.db     63, 133
.db     216, 8
.db     121, 126
.db     188, 98
.db     168, 205
.db     157, 172
.db     108, 233
.db     80, 255
.db     252, 102
.db     198, 0
.db     171, 239
.db     107, 114
.db     172, 170
.db     17, 45
.db     42, 55
.db     34, 174
.db     229, 250
.db     12, 179
.db     187, 243
.db     44, 231
tableend:
.db     76, 48
```

5.2.11 半双工中断方式 UART 应用一

该应用程序包含了一个非常有效的 UART 软件，并给出一个接收字符，然后返回一个应答的子程序。

```
;*********************************************************
;* Title:          UART 半双工中断软件
;* Target:         AT90SXXXX (All AVR Devices)
```

```
;*                    Timer/Counter0 overflow interrupt
;* 该应用程序描述怎样用 8 位定时器/计数器 0 和外部中断的 AVR 单片机实现半双工 UART 的软件。与
;* 大多数通信应用控制一样,仅在单次和单方向的半双工 UART,将限制了 MCU 资源的应用。
;* 常数 N 和 R 决定了数据率。R 为时钟频率选择,即在 AVR 数据手册中描述的 T/C 预比例器。如果
;* T/C 预比例系数表示为 C,那么波特率由下列等式表示:
;*          XTAL
;* BAUD   = ···              min. N * C = 17
;*          N * C            max. N = 170
;* N * C 的绝对最小值为 17 (原因是在中断完成前中断标志再次被置位的最小时间)。绝对最大值为 170
;* (原因是使接收到的位正确必须最长为 1.5 位)。
;* UART 使用 PD2 作为接收管脚,原因是它有利于外部中断。在这个例子中发送管脚是 PD4,该管脚也能
;* 作其它脚。由于 UART 是半双工,所以只能发送数据或接收数据,它不能同时进行。当闲置时,UART
;* 将自动接收数据。但是,如果正在发送数据时到来数据,将被放弃。同时,如果调用 u_transmit 程序,而
;* 没有等到"u_status"寄存器中的"READY"位被清除,它将放弃任何接收式发送操作。
;* 初始化:1. 调用 uart_init 子程序;2. 使能全局中断( 和"SEI" )
;* 接收:1. 等待"u_status"中的 RDR 变成置位;2. 读"u_buffer"。
;* 发送:(0. 执行 uart_init 初始化 UART 用 SEI 指令);1. 等待"u_status"中的 READY 变成零;2. 置"u_
;* buffer";3. 调用"u_transmit"。
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
.include "1200def.inc"
;* 波特率设置
                                        ; BAUDRATES @1MHz XTAL AND R = 1
.equN  = 104                            ; 9600
.equN  = 52                             ;19200
.equ   N                = 26            ;38400
.equ   C                = 1             ;Divisor
.equ   R                = 1             ;R = 1 when C = 1
;* UART 全局寄存器
.def   u_buffer         = r14           ; 串行缓冲器
.def   u_sr             = r15           ; 状态寄存器存储
.def   u_tmp            = r16           ; 暂存寄存器
.def   u_bit_cnt        = r17           ; 位计数器
.def   u_status         = r18           ; 状态缓冲器
.def   u_reload         = r19           ; 重复装载寄存器
.def   u_transmit       = r20           ; 发送的数据
;* 在状态寄存器中位位置
.equ   RDR              = 0             ; 接收数据就绪位
.equ   TD               = 6             ; 发送数据(内部只读)
.equ   BUSY             = 7             ; 忙标志(内部只读)
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* 程序开始执行
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      cseg
      org       $0000
      rjmp      start                   ; 复位处理
      org       INT0addr
      rjmp      ext_int0                ; 外部中断处理
      org       OVF0addr
      rjmp      tim0_ovf                ; 定时器 0 溢出处理
```

```
      org     ACIaddr
      reti                                   ; 模拟比较器处理
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* 外部中断程序 0
;* 该子程序当检测到外部串行信号负沿时执行,它禁止另外的外部中断并使能定时器中断,因为 UART 现
;* 在必须接收传来的数据。该子程序置位 GIMSK、TIFR 和 TIMSK 寄存器,在该程序中,当所有被置位时,
;* 覆盖了所有其它位。原因是在低时钟率和高波特率操作时,丢失有效周期是存在的。
;* Total number of words                     : 12
;* Total number of cycles                    : 15 (incl. reti)
;* Low register usage                        : 1 (u _ sr)
;* High register usage                       : 4 (u _ bit _ cnt, u _ tmp, u _ status, u _ reload)
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
ext _ int0:
      in      u _ sr, SREG                   ; 存储状态寄存器
      ldi     u _ status, 1<<BUSY            ; 置忙标志(清所有其它位)
      ldi     u _ tmp, (256-(N+N/2)+(29/C))  ; 置定时器重装载值(1.5 位长)
      out     TCNT0, u _ tmp                 ; 29=时间延时,已经在中断脉冲
                                             ; 被用。该时间用于定时器中断
                                             ; 请求和第一次数据位实际采样
      ldi     u _ tmp, 1<<TOIE0              ; 置 u _ tmp 中的位 1
      out     TIFR, u _ tmp                  ; 清除 T/C0 溢出标志
      out     TIMSK, u _ tmp                 ; 使能 T/C0 溢出中断
      clr     u _ bit _ cnt                  ; 清除位计数器
      out     GIMSK, u _ bit _ cnt           ; 禁止外部中断
      ldi     u _ reload, (256-N+(8/C))      ; 置重装载值(常数)
      out     SREG, u _ sr                   ; 存储 SREG
      reti
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* 定时器/计数器 0 溢出中断
;* 该程序协调位的发送和接收。当速率等于波特率时,该程序自动执行。在发送时,程序移该位并发送它。
;* 在接收时,该程序采样该位并移到缓冲器。
;* 串行程序使用一个状态寄存器(u _ statas):只读
;* BUSY                                       UART 忙时该位随时指示。
;* TD                                         发送数据,当 UART 正发送时置位。
;* RDR                                        接收数据就绪,当数据接收完成准备下次读时,置 1。
;* 当 RDR 标志被置位,新数据能从 u-buffer 读出。
;* 该程序总是置位 TIMSK 中的 TIMSK 位,见 ext _ into 描述。
;* Total number of words                      : 35
;* Total number of cycles                     : min. 18, max. 28 - depending on if it is
;* receiving or transmitting, what bit is
;* pending, etc.
;* Low register usage                         : 2 (u _ sr, u _ buffer)
;* High register usage                        : 4 (u _ bit _ cnt, u _ tmp, u _ status, u _ reload)
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
tim0 _ ovf:
    in u _ sr, SREG                           ; 存储状态寄存器
    out TCNT0, u _ reload                     ; 重装载定时器
    inc u _ bit _ cnt                         ; 位计数加 1
    sbrs u _ status, TD                       ; 如果发送位置 1
```

```
    rjmp tim0 _ receive                    ; 去接收
    sbrc u _ bit _ cnt, 3                  ; 如果 u _ bit _ cnt 位 3 置位
    rjmp tim0 _ stopb                      ; 跳到 stop - bit - part
    sbrc u _ buffer, 0                     ; 如果缓冲器的最低有效位是 1
    sbi PORTD, PD4                         ; 置发送为 1
    sbrs u _ buffer, 0                     ; 如果缓冲器的最低有效位是 0
    cbi PORTD, PD4                         ; 置发送为 0
    lsr u _ buffer                         ; 右移缓冲器
    out SREG, u _ sr                       ; 存储 SREG
    reti
tim0 _ stopb:
    sbi PORTD, PD4                         ; 产生停止位
    sbrs u _ bit _ cnt, 0                  ; 如果 u _ bit _ cnt = 8 (stop - bit)
    rjmp tim0 _ ret                        ; 跳出
tim0 _ complete:
    ldi u _ tmp, 1<<INT0                   ; (u _ bit _ cnt = = 9):
    out GIMSK, u _ tmp                     ; 使能外部中断
    clr u _ tmp
    out TIMSK, u _ tmp                     ; 禁止定时器中断
    cbr u _ status, (1<<BUSY)|(1<<TD)      ; 清忙标志和发送标志
tim0 _ ret:
    out SREG, u _ sr                       ; 存储状态寄存器
    reti
tim0 _ receive:
    sec                                    ; 置进位
    sbis PIND, 2                           ; 如果 PD2 = 低, 则 < = = 这个采样
    clc                                    ; 清进位
    ror u _ buffer                         ; 移进位到数据
    in u _ tmp, SREG                       ; 存储 SREG
    cpi u _ bit _ cnt, 9                   ; 如果 u _ bit _ cnt!  = 9 (必须采样停止位)
    brne tim0 _ ret                        ; 退出中断
    out SREG, u _ tmp                      ; 得到旧的 SREG
    rol u _ buffer                         ; 循环返回数据
    sbr u _ status, 1<<RDR                 ; 清忙标志
    rjmp tim0 _ complete
;*************************************************************************
;* UART 初始化子程序
;* 该子程序初始化 UART,置定时器并使能外部中断(接收)。要使能 UART,则全局中断标志必须被置位
;* (用 SEI 指令)。
;* Total number of words                   : 8
;* Total number of cycles                  : 11 (including the RET instructions)
;* Low register usage                      : None
;* High register usage                     : 2 (u _ tmp, u _ status)
;*************************************************************************
uart _ init:
    ldi u _ tmp, R
    out TCCR0, u _ tmp                     ; 启动定时器和置时钟源
    ldi u _ tmp, 1<<INT0
    out GIMSK, u _ tmp                     ; 使能外部中断 0
```

```
    in u_tmp,1<<ISC01
    out MCUCR,u_tmp                          ;在下降沿
    clr u_status                             ;清除状态位
    ret
;*********************************************************************
;* UART 发送子程序
;* 该子程序初始化 UART 并发送数据,数据必须装到 u_trasmit 中。
;* 警告:这个程序取消所有其它的发送和接收。要小心,如果一个字节已被接收,定时器中断正在运行,发
;* 送可能不可靠。为安全发送,应该校验 u_status 寄存器中的 READY 位和 TD 位。
;* 该程序总是置位 TIMSK 中的 TIMSK 位,见 ext_into 描述。
;* Total number of words                   : 13
;* Total number of cycles                  : 17 (including RET)
;* Low register usage                      : 1 (u_buffer)
;* High register usage                     : 4 (u_bit_cnt,u_tmp,u_transmit,u_reload)
;*********************************************************************
uart_transmit:
    ldi u_status,(1<<BUSY)|(1<<TD)           ;仅置忙和发送标志
    clr u_tmp
    out GIMSK,u_tmp                          ;禁止外部中断
    ser u_bit_cnt                            ;清除位计数器
    mov u_buffer,u_transmit                  ;拷贝发送数据到缓冲器
    ldi u_tmp,1<<TOIE0                       ;置 u_tmp 中的位 1
    out TIFR,u_tmp                           ;清除 T/C0 溢出标志
    out TIMSK,u_tmp                          ;并使能 T/C0 溢出中断
    ldi u_reload,(256-N+(8/C))               ;置重装载值
    ldi u_tmp,(256-N+(14/C))                 ;置定时器延时到第一位
    out TCNT0,u_tmp                          ;置定时器重装载位(1 位)
    cbi PORTD,PD4                            ;清除输出(开始位)
    ret
;*********************************************************************
;* 测试程序
;* 该例子程序作为 PC 与终端仿真器之 UART 的评价程序,如果在 PC 键盘上按一个键,见一个信息"You
;* typed <characher>"被发回,字符同时表示在 B 口上。
;*********************************************************************
.def   tmp      = r21                        ;暂存寄存器
.def   buffer   = r22                        ;接收到字节
.def   adr      = r23                        ;EEPROM 地址
start: ser      tmp                          ;初始化
       out      PORTD,tmp                    ;置 D 口为输入上拉
       sbi      DDRD,DDD4                    ;除 PD4 外
       out      DDRB,tmp                     ;置 B 口为输出 1
       out      PORTB,tmp
       rcall    uart_init                    ;初始化 UART
       sei                                   ;使能中断
idle:  sbrs     u_status,RDR                 ;等待字符
       rjmp     idle
       mov      buffer,u_buffer              ;得到接收的字符
       out      PORTB,u_buffer               ;输出字节到 B 口
       ldi      adr,example_data             ;设置/存储指针到 EEPROM 数据区
```

```
loop:  out     EEAR, adr                ; 设置 EEPROM 地址
       sbi     EECR, EERE               ; 发送读选通
       in      u_transmit, EEDR         ; 数据放到发送寄存器
       rcall   uart_transmit            ; 发送数据
wait:  sbrc    u_status, TD             ; 等待数据发送状态位
       rjmp    wait
       inc     adr                      ; 指针加 1
       cpi     adr, example_data + 12   ; 到达字节 12? (结束?)
       breq    idle                     ; 是,等待下一个字符。
       cpi     adr, example_data + 10   ; 到达字节 10?
       brne    loop                     ; 不,跳回
       mov     u_transmit, buffer       ; 数据放到发送寄存器
       rcall   uart_transmit            ; 发送数据
wait2: sbrc    u_status, TD             ; 等待发送状态位
       rjmp    wait2
       rjmp    loop                     ; 继续发数据
;**********************************************************
;* 测试程序数据:这是当一个字符被接收后,将被发回的数据。
;**********************************************************
       .eseg
example_data:
       .db     89                       ;'Y'
       .db     111                      ;'o'
       .db     117                      ;'u'
       .db     32                       ;' '
       .db     116                      ;'t'
       .db     121                      ;'y'
       .db     112                      ;'p'
       .db     101                      ;'e'
       .db     100                      ;'d'
       .db     32                       ;' '
       .db     13                       ;<CR>
       .db     10                       ;<LF>
```

5.2.12 半双工中断方式 UART 应用二

该应用程序给出一个怎样用 8 位定时器/计数器 0 和外部中断实现半双工 UART 的软件。

```
;**********************************************************
;* Title:          UART 半双工中断软件
;* Target:         AT90SXXXX (All AVR Device)
;* 该程序包含了非常有效的 UART 软件,是接收一个字符并返回一个回答信号。
;**********************************************************
.include "1200def.inc"
;*引脚定义
.equ    RxD      =0             ;接收引脚是 PD0
.equ    TxD      =1             ;发送引脚是 PD1
;*全局寄存器变量
.def    bitcnt   =R16           ;位计数器
.def    temp     =R17           ;暂存寄存器
```

```
.def     Txbyte    =R18            ;数据被发送
.def     Rxbyte    =R19            ;接收数据
.cseg
.org     0
;*****************************************************************
;* "putchar"
;*该子程序发送一个存储在"Txbyte"寄存器中的字节,用得停止位数以常数 sb 设置。
;* Number of words :14 including return
;* Number of cycles                 ;Depens on bit rate
;* Low registers used               ;None
;* High registers used              ;2 (bitcnt,Txbyte)
;* Pointers used                    ;None
;*****************************************************************
.equ      sb       =1              ;停止数 (1, 2, …)
putchar:  ldi      bitcnt,9+sb     ;1+8+sb (sb is # of stop bits)
          com      Txbyte          ;倒置
          sec                      ;启动位
putchar0: brcc     putchar1        ;如果进位置位
          cbi      PORTD,TxD       ;发一个"0"
          rjmp     putchar2        ;否则
putchar1: sbi      PORTD,TxD       ;发一个"1"
          nop
putchar2: rcall    UART_delay      ;1 位延时
          rcall    UART_delay
          lsr      Txbyte          ;得到下一位
          dec      bitcnt          ;如果所有位未发完
          brne     putchar0        ;发送下一个
                   ;否则
          ret                      ;返回
;*****************************************************************
;* "getchar"
;* 该子程序接收一个字节,并放到"Rxbyte"寄存器。
;* Number of words :14 including return
;* Number of cycles                 ;Depens on when data arrives
;* Low registers used               ;None
;* High registers used              ;2 (bitcnt,Rxbyte)
;* Pointers used                    ;None
;*****************************************************************
getchar:  ldi      bitcnt,9        ;8 位数据位+ 1 位停止位
getchar1: sbic     PIND,RxD        ;等待启动位
          rjmp     getchar1
          rcall    UART_delay      ;0.5 位延时
getchar2: rcall    UART_delay      ;1 位延时
          rcall    UART_delay
          clc                      ;清进位
          sbic     PIND,RxD        ;如果 Rx 引脚高
          sec
          dec      bitcnt          ;如果位是停止位
          breq     getchar3        ;返回
```

```
                                        ;否则
        ror         Rxbyte              ;移位到 Rxbyte
        rjmp        getchar2            ;得到下一位
getchar3:   ret
;*************************************************************
;* "UART _ delay"
;* 这个延时子程序产生一个在发送和接收字节的位之间所需延时,总的执行时间由常数"b"设置。
;* 3words + 7 cycles (including rcall and ret)
;* Number of words :4 including return
;* Low registers used                   :None
;* High registers used                  :1 (temp)
;* Pointers used                        :None
;*************************************************************
; Some b values :(See also table in Appnote documentation)
; 1 MHz crystal:
; 9 600         bps - b     =14
; 19 200        bps - b     =5
; 28 800        bps - b     =2
; 2 MHz crystal:
; 19 200        bps - b     =14
; 28 800        bps - b     =8
; 57 600        bps - b     =2
; 4 MHz crystal:
; 19 200        bps - b     =31
; 28 800        bps - b     =19
; 57 600        bps - b     =8
; 115 200       bps - b     =2
.equ            b           =31             ;19 200 bps @ 4 MHz crystal
UART _ delay:   ldi         temp, b
UART _ delay1:  dec         temp
                brne        UART _ delay1
                ret
;*************************************************************
;* 程序在这里开始执行启动
;* 测试程序
;*************************************************************
reset:          sbi         PORTD, TxD      ;初始化口引脚
                sbi         DDRD, TxD
                ldi         Txbyte, 12      ;清除终端
                rcall       putchar
forever:        rcall       getchar
                mov         Txbyte, Rxbyte
                rcall       putchar         ;回答接收字符
                rjmp        forever
```

5.2.13 8 位精度 A/D 转换器

该应用程序给出了用在片内模拟比较器和几个外部元件实现双斜率 A/D 转换器的软件。

```
;*************************************************************
```

```
;* Title:              8 位精度 A/D 转换器
;* Target:             AT90SXXXX (All AVR Devices with analog comparator)
;* 该程序显示如何使用单片机片内模拟比较器和几个外部元件实现双斜率 A/D 转换器,包括一个测试程
;* 序,完成外循环的转换并输出到 8 位 LED 显示。
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* * * * * mpy9u 乘法子程序使用的寄存器
.def            mc9u        = r0            ;乘数子程序使用的被乘数
.def            mp9u        = r1            ;乘数子程序使用的乘数
.def            m9uL        = r1            ;结果低字节
.def            m9uH        = r2            ;结果高字节
;* * * * * div17u 除法子程序使用的寄存器
.def            didL        = r1            ;被除数
.def            didH        = r2
.def            dresL       = r1            ;除法的保持结果
.def            dresH       = r2
.def            divL        = r3            ;除数
.def            divH        = r4
.def            remL        = r5            ;除法子程序使用的余数
.def            remH        = r6
.def            TinH        = r14           ;定时得到输入电压
.def            TinL        = r15
.def            Tref        = r16           ;定时得到参考电压
.def            TH          = r17           ;定时器变量
.def            Vref        = r18           ;被计算的 VREF
.def            temp        = r19
.def            temp2       = r20
;Port B pins
.equ            AIN0        = 0
.equ            AIN1        = 1
.equ            Ref1        = 2
.equ            Ref2        = 3
.equ            LED         = 4
.equ            T           = 7
.equ            PRESC       = 2             ;定时器 CK/8 时钟
.equ            VrefAddr    = 0             ;EEPROM 保持 VREF的地址
.include "1200def.inc"
.cseg
.org            0
                rjmp        reset           ;复位处理器
                reti
.org OVF0addr
;* * 定时器/计数器 0 溢出中断
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
T0_int:         inc         TH              ;定时器高字节加 1
                reti
;* * * 复位处理器 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
reset:          sbi DDRB, LED               ;PB4 和 D 口用于输出
                ser temp                    ;驱动 LED 显示器
                out DDRD, temp
```

```
                ldi temp, (1<<TOIE0)        ;使能定时器中断
                out TIMSK, temp
                sei                         ;使能全局中断
                sbi PORTB, T                ;打开 T 管脚的上拉
                rcall delay
                sbic PINB, T                ;如引脚没下拉低
                rjmp main                   ;跳到 main
                                            ;否则
calibrate:      sbis PINB, T                ;等待管脚释放
                rjmp calibrate
                cbi PORTB, T                ;关闭 T 引脚的上拉
                rcall delay                 ;让标准电压稳定
                rcall reference             ;测量 TREF
                rcall delay
                rcall input                 ;测量 TCAL
                clc
                mov mp9u, Tref              ;TREF→乘数
                ldi temp, 128               ;128→被乘数(=2.5 V)
                mov mc9u, temp
                rcall mpy9u                 ;TREF×128
                clr divH                    ;(TREF×128)
                mov divL, TinL              ;……
                rcall div17u                ;TCAL
                ldi temp, VrefAddr          ;储存 VREF
                out EEAR, temp
                out EEDR, dresL
                sbi EECR, EEWE
calibrate1:     sbic EECR, EEWE
                rjmp calibrate1
;主程序
main:           cbi PORTB, T                ;关闭 T 引脚的上拉
                ldi temp, VrefAddr          ;从 EEPROM 读 VREF
                out EEAR, temp
                sbi EECR, EERE
                in Vref, EEDR
loop:           rcall reference             ;测量 TREF
                rcall delay                 ;一个小的延时让
                                            ;电容器不改变值
                rcall input                 ;测量 TIN
                brts error                  ;如果 VIN>VCC
calc:           lsr TinH                    ;TINH→C(乘数)
                mov mp9u, TinL              ;TINL→乘数
                mov mc9u, Vref              ;TREF→被乘数
                rcall mpy9u                 ;TIN×TREF
                clr divH                    ;(TIN×VREF)
                mov divL, Tref              ;……
                rcall div17u                ;TREF
                tst dresH
                breq write
```

```
error:          ldi temp,255          ;VIN = 255
                mov dresL,temp
write:          com dresL             ;在 LED 上显示值
                rcall long_delay
                rcall long_delay
                rcall long_delay
                out PORTD,dresL
                rol dresL
                brcs wr1
                cbi PORTB,LED
                rjmp loop
wr1:            sbi PORTB,LED
                rjmp loop
;*** 延时子程序 ************************************
delay:          ldi temp,$FF
d1:             dec temp
                brne d1
                ret
long_delay:     ser temp2
ld1:            rcall delay
                dec temp2
                brne ld1
                ret
;*** 基准子程序 ************************************
;* 测量 TREF
reference:      sbi DDRB,AIN0         ;电容器放电
                sbi DDRB,Ref1         ;打开 VREF
                sbi PORTB,Ref1
                sbi DDRB,Ref2
                rcall delay           ;让电容器完全放电
                completely
                clr TH                ;复位定时器
                out TCNT0,TH
                cbi DDRB,AIN0         ;AIN0 脚作为输入
                ldi temp,PRESC
                out TCCR0,temp
                sbi DDRB,T            ;打开晶体管
ref_wait:       sbic ACSR,ACO         ;如电容器电压>基准电压
                rjmp ref_ok           ;转换完成
                cpi TH,1              ;如果定时器溢出则继续
                brlo ref_wait
ref_ok:         in Tref,TCNT0         ;存储 TREF
                clr temp              ;停止定时器
                out TCCR0,temp
                cbi DDRB,T            ;关闭晶体管
                sbi DDRB,AIN0         ;电容器放电
                cbi PORTB,Ref1        ;关闭 VREF
                cbi DDRB,Ref1
                cbi DDRB,Ref2
```

```
                ret
;***输入子程序*************************************
;* 测量 TIN
;* 如果定时器溢出,则带 T 标志返回 (i.e. VIN> VCC)
input:          cbi DDRB, AIN0              ;Tri-state AIN0
                clt                         ;清错误 flag
                clr TH                      ;清定时器
                out TCNT0, TH
                ldi temp, PRESC             ;启动定时器
                out TCCR0, temp
                sbi DDRB, T                 ;打开晶体管
input_wait:     sbic ACSR, ACO              ;如果电容器电压 > VIN
                rjmp input_ok               ;充电完成
                brlo input_wait
                set                         ;T=1 指示 VIN> VCC
input_ok:       in TinL, TCNT0              ;存储 TIN
                mov TinH, TH
input_exit:     clr temp                    ;停止定时器
                out TCCR0, temp
                cbi DDRB, T                 ;关闭晶体管
                sbi DDRB, AIN0              ;电容器放电
                ret
;**************************************************
;* 该子程序为 17 位数(进位:didH:didLL) 除以 16 位数(divH:divL)。结果放在 (dresH:dresL)。
;* 在子程序执行前被除数的第 17 位必须带进位位。
;* 子程序基于 div16u——16/16 位无符号数除法子程序。
;**************************************************
div17u:         clr remL                    ;清余数位低字节
clr remH                                    ;清余数位高字节
                ldi temp, 17                ;初始化循环计数器
d17u_1:         rol remL                    ;移被除数到余数
                rol remH
                sub remL, divL              ;余数 = 余数 - 除数
                sbc remH, divH
                brcc d17u_2                 ;如果结果为负
                add remL, divL              ;存储余数
                adc remH, divH
                clc                         ; 清被移到结果的进位位
                rjmp d17u_3                 ;否则
d17u_2:         sec
d17u_3:         rol didL                    ;左移被除数
                rol didH
                dec temp                    ;计数器减 1
                brne d17u_1                 ;如果是
                ret                         ;则带 didL 返回
;**************************************************
;* "mpy9u"###9×8 无符号乘法
;* 该子程序为寄存器变量 (进位:mp9u) 和 mc9u 相乘。结果放在 (进位:m8uH:m8uL)中。
;* Number of words :11 (return included)
```

```
;* Number of cycles                               :(return included)
;* Low registers used                             :3 (mp9u, mc9/m9uL, m9uH)
;* High registers used                            :1 (temp)
;* 注意：结果低字节和乘数共用同一个寄存器,这导致乘数被结果覆盖。
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
mpy9u:          clr m9uH                          ;清结果高字节
                ldi temp,9                        ;初始化循环计数器
                ror mp9u
m9u_1:          brcc m9u_2                        ;如果乘数位0置位
                add m9uH,mc9u                     ; 加被乘数到结果高字节
m9u_2:          dec temp                          ;循环计数器减1
                brne m9u_3                        ;如果不是,再循环
                ret
m9u_3:          ror m9uH                          ;右移结果高字节
                ror m9uL                          ;乘数和结果低字节循环右移
                rjmp m9u_1
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

第六章　AVR 单片机的应用

ATMEL 公司的 AVR 单片机，是增强型 RISC 内载 Flash 的单片机。芯片上的 Flash 存储器附在用户的产品中，可随时编程、再编程，使用户的产品设计容易，更新换代方便。AVR 单片机采用增强的 RISC 结构，使其具有高速处理能力，在一个时钟周期内可执行复杂的指令，每 MHz 可实现 1MIPS 的处理能力。AVR 单片机工作电压为 2.7～6.0 V，可以实现耗电最优化。AVR 的单片机广泛应用于计算机外部设备，工业实时控制，仪器仪表，通讯设备，家用电器，宇航设备等各个领域。

6.1　廉价的 A/D 转换器

AVR 单片机的 AT90 系列片内置有模拟比较器。这一节介绍用 AT90SXXXX 单片机实现的廉价 A/D 转换器。

一、硬件设计

使用 AVR 单片机及一个外部电阻和一个外部电容器设计成一个 A/D 转换器，并使用片内的定时器/计数器中断和模拟比较器中断。采用 RC 模拟转换的原理。这种转换方法在精确度和转换时间的花费上是极低的。

如图 6.1 所示，RC 模拟转换电路仅由 AVR 单片机外加一个电阻和一个电容器构成。电阻一端接 D 口管脚，另一端与电容器接成 RC 振荡器，电容器的正端接单片机的 AIN0（内部比较器正）端，单片机的 AIN1（内部比较器负）端接外部的未知输入电压。通过 D 口端的控制对电容器进行充电。

通过电容器的电压按指数曲线关系下降，当压缩转换的电压范围到 $2/5V_{DD}$ 时，指数曲线接近水平线。若通过电容器的电压相等于被转换的电压时，叫完成一次简单的测量。用模拟比较器来实现比较，当通过电容器的电压上升超过测量电压时，模拟比较器给出中断，完成一次测量。输出可分成 64 级。

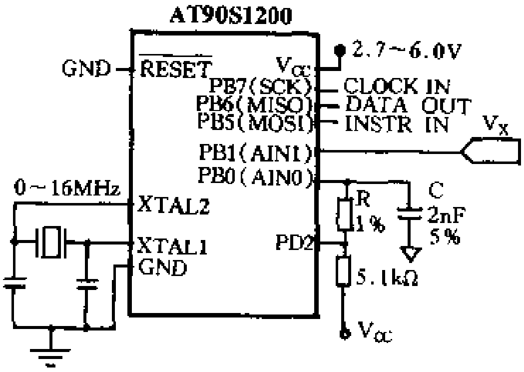


图 6.1　A/D 转的换器

为了确保定时时间，RC 网络的时间常数要满足：

$$512\times(1/f) = -R\times C\times \ln(1-2/5)$$

为了使 A/D 转换能正常工作，电容器必须在每次转换后完全放电，放电的时间允许最小为 200 μs。

二、软件编程

下面介绍的是实现上述廉价 A/D 转换器的程序。该程序包括：A/D 转换器初始化子程序，模拟比较器中断子程序，定时器/计数器中断子程序，A/D 转换子程序等。

```
;*****************************************************
;* Titlle:            低成本 A/D 转换
;* Target:            AT90SXXXX (All AVR Devices)
;* Interrupt usage:   定时器/计数器 0 溢出中断
;*                    模拟比较器中断
```

```
;* 初始化:1. 调转换初始化子程序。2. 使能全局中断。
;* A/D 转换:1. 调 A/D 转换子程序。2. 等待转换完成 (小于 521 周期)。3. 从结果读出数据。
;*********************************************************
.include "1200def.inc"
;*****常数
.equ                preset = 192                ;T/C0 比例常数 (256~64)
;***** A/D 转换全局寄存器
.def                result = r16                ;结果和中间数据
.def                temp = r17                  ;暂存寄存器
;*********************************************************
;*程序开始
;*********************************************************
.cseg
    .org ¥0000
    rjmp RESET                                  ;复位处理
    .org OVF0addr
    rjmp ANA_COMP                               ;T0 溢出处理
    .org ACIaddr
    rjmp ANA_COMP                               ;模拟比较器处理
;*********************************************************
;*ANA_COMP——模拟比较器中断子程序
;* 该子程序在两个事件中的一个发生时执行:1 定时器/计数器 0 溢出中断;2 模拟比较器中断。
;* 转换结束的两个事件之一:如果信号超出范围,则定时器溢出中断;如果模拟比较器超出范围,则发生中断。
;* 位置被确定,并且 T 标志被置位。
;* 由于中断处理需要一些时钟周期,必须比上述的时钟周期多减 1 个。
;* 字的总数          : 7       周期总数          : 10
;* 低寄存器使用      : 0       高寄存器使用      : 2 (result, temp.)
;* 状态标志          : 1 (t flag)
ANA_COMP:
    in              result, TCNT0               ;装入时间值
                    clr temp                    ;停止 T0
                    out TCCR0, temp
                    cbi PORTD, PD2              ;开始放电
                    set                         ;置转换完成标志
                    reti                        ;中断返回
;*********************************************************
;*A/D 转换的初始化子程序
;* 该子程序初始化 A/D 转换器。模拟比较器中断初始化用 ACO 的上升沿初始化。为了使能 A/D 转换器
;* 全局中断标志必须设置 SEI。
;* 转换完成标志(T)被清除
;* 字的总数          : 6       周期总数          : 10
;* 低寄存器使用      : 0       高寄存器使用      : 1 (result)
;* 状态标志          : 0
;*********************************************************
convert_init:
    ldi             result, ¥0B                 ;初始化比较器
    out             ACSR, result                ;使能比较器中断
    ldi             result, ¥02                 ;使能定时器中断
    out             TIMSK, result
    sbi             PORTD, PD2                  ;设置转换器充电/放电
```

```
                                                        ;作为输出
    ret                                                 ;返回子程序
;*****************************************************************************
;* A/D 转换器启动子程序
;* 该子程序启动转换器。将位移值装入 T0 并启动定时器。同时启动电容器充电。
;* 字的总数          : 7          周期总数          : 10
;* 低寄存器使用      : 0          高寄存器使用      : 1 (result)
;* 状态标志          : 1 (t flag)
;*****************************************************************************
AD_convert:
    ldi         result, preset                          ;清计数器
    out         TCNT0, result                           ;装入位移值
    clt                                                 ;清转换器完成标志(T)
    ldi         result, ¥02                             ;启动 f/8 比例尺的 T0
    out         TCCR0, result
    sbi         PORTB, PB2                              ;启动电容器充电
    ret                                                 ;返回子程序
;*****************************************************************************
;* 例子程序
;* 这个程序是一个怎样适当设置 A/D 转换器的例子。
;* 注意! 为确保合适的操作,必须使电容器的放电周期在每个转换前大于 200 μs。结果在 B 口输出。
;* 为确保正确的放电,应加一个延时循环,循环为 11 000 个周期。在 20 MHz 晶振时给出 550 μs 延时。
;*****************************************************************************
RESET:
                rcall convert_init                      ;初始化 A/D 转换器
                sei                                     ;使能全局中断
                ldi result, ¥ff                         ;置 B 口为输出
                out DDRB, result
Delay:          clr result                              ;清暂存计数器 1
                ldi temp, ¥f0                           ;复位暂存计数器 2
loop1:          inc result                              ;暂存计数器 1 加计数
                brne loop1                              ;检验是否内部循环完成
                inc temp                                ;暂存计数器 2 加计数
                brne loop1                              ;检验是否延时完成
                rcall AD_convert                        ;启动转换器
Wait:           brtc Wait                               ;等待直到转换完成
                out PORTB, result                       ;写结果到 B 口
                rjmp Delay                              ;重复转换
```

6.2 用 AVR 单片机控制 FPGA 配置

一、简 介

基于 SRAM 的 FPGA 如 ATMEL AT6000 有诸多优点,因此已越来越多地被使用。它们的重新配置功能允许用户将比 FPGA 实际所拥有的更多的逻辑门提供给应用,实行起来只不过是把所需的逻辑门进入 FPGA。这又叫超高速缓存逻辑(Cache Logic)。若要使超高速缓存逻辑能高效率地使用,FPGA 必须符合下列要求:部分重配置能力即一种快速的重配置过程,并且结构上是对称的。

FPGA 能够自己控制和改变它的配置,但也可以由单片机用一种很巧妙的方法来做到。在配置过程后或者在两个配置循环之间,它可用于其它目的,并且不会因此而丢失。本节列出

了可达到节省空间、保护设计或快速灵活的几种不同的重配置方法。

二、FPGA 和单片机之间的配置数据传输

构成 FPGA 的配置信息的信息量称为比特流。它作为一个文件存储在存储器中。图 6.2 为比特流的结构。

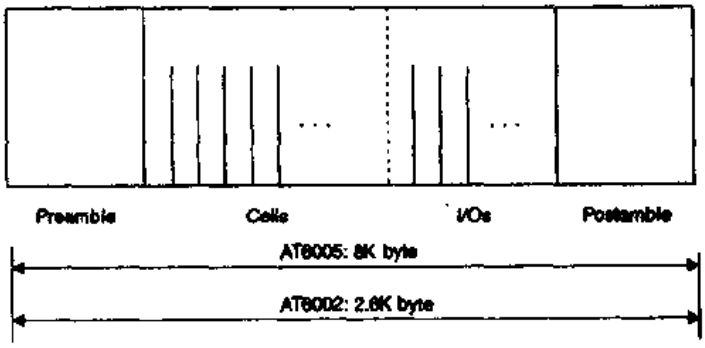


图 6.2 比特流结构

比特流以一个标志开头，指示首端的开始且包含了与整个配置周期相关的综合信息。后面的是核心的配置信息和 I/O 配置。尾端表示比特流的结尾。

这些数据就像一个文本文件发送到打印机的一样，一个接一个发送到 FPGA 中。这要求以串行或并行方式完成，并需要一个与打印机商品传输协议一样的传输协议。

FPGA 和单片机之间串行数据传输的连接如图 6.3 所示，只有 7 条线被用来控制配置循环。

单片机可以将状态控制线都配置为 0，以把整个 FPGA 完全重新设置。FPGA 所输入的状态也就是上电后要输入的状态。所有的 I/O 都处于三态。精确的定时如图 6.4 所示。

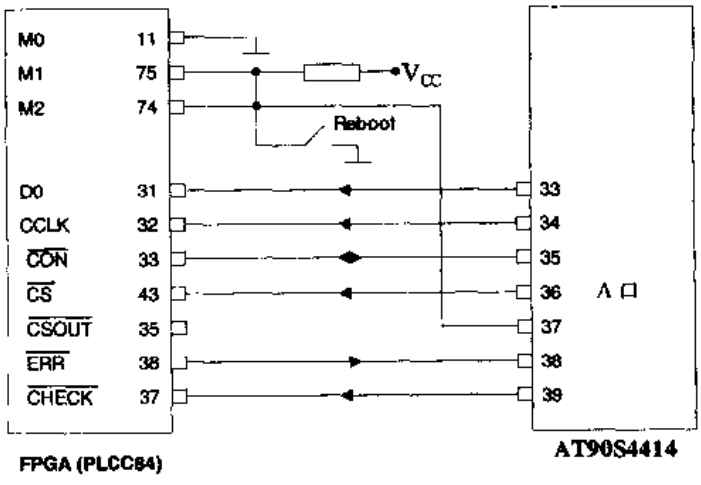


图 6.3 AT90SXXXX 和 FPGA 的连接

在触发重新引导循环后，FPGA 将

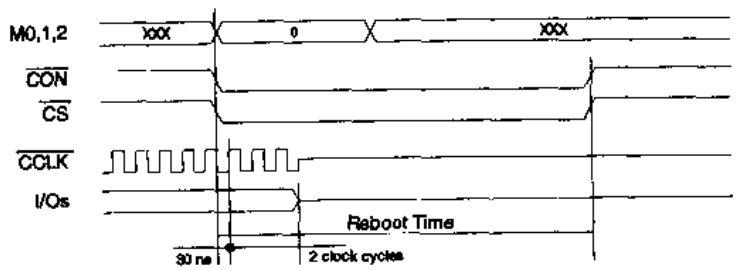


图 6.4 FPGA 的复位时序

$\overline{CON}$线降为低电平 。单片机可以检验这条线并检测循环的结束。

在重新引导循环后，$\overline{CS}$和$\overline{CON}$仍在两个时钟周期为高电平。单片机可以通过$\overline{CS}$线来选择一个 FPGA 做配置，并将$\overline{CON}$置为低来启动配置周期。CCLK 线上的每一次时钟脉冲，便会有一比特的比特流传送给 FPGA。首先传输的比特是首端的 LSB，最后传输的是尾端的 MSB。

为了保证 FPGA 内部状态机器的正确功能，为了在两个配置之间正确输入备用状态，24 个时钟脉冲被加在一个配置周期的前面和后面。这些时钟脉冲是在$\overline{CON}$置为低之前。$\overline{CON}$置为高之后。

当由低到高过渡，FPGA 指出配置周期已结束。精确定时如图 6.5 所示。

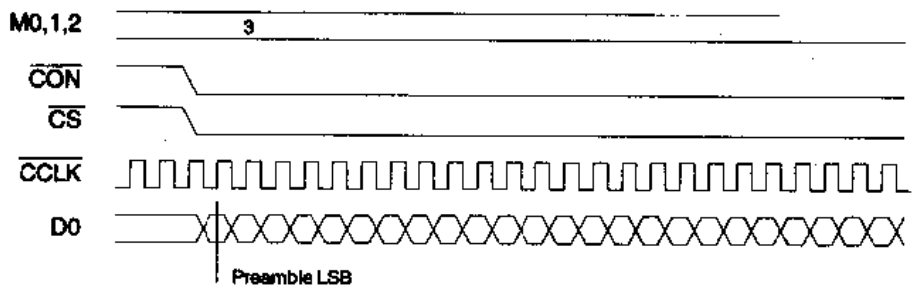


图 6.5 FPGA 的配置周期时序

配置状态 3 是用来配置几个串联的 FPGA。在这种情况下，除$\overline{CS}$线以外的所有线与所存的 FPGA 进行平行连接。这些与链中前面的 FPGA 的$\overline{CS}$ OUT 针相连。

只需一个包含几个配置的比特流就可以为所有 FPGA 服务。该链中的第一个 FPGA 遇到一个新的首端，而不是一个尾端时，它全激活$\overline{CS}$ OUT 针。链中的下一个 FPGA 准备接受新的配置信息，以此类推。

另外一个方法是将$\overline{CS}$线连到单片机上，这样它能通过选择下一个 FPGA 来配置。所有其它线都与所有的 FPGA 并行连接。这种情况下，一般的比特流已足够了，而不是那种像上述的明确生成并包含几种首端的特殊比特流。

如果配置期间出现错误，单片机可以通过$\overline{ERR}$线检测到。当一个错误从 FPGA 检测到，例如一个无效的首端，这条线就会被置为低来指示给单片机；当这个错误用几个 FPGA 时，错误线会被连起来，因为它可以被线或(WIRE－OR ed)。单片机用$\overline{CHECK}$线来决定 FPGA 是否已重新配置。它还将接收到的比特流与已经存储在 FPGA 内配置存储器里的信息做比较。如出现不同，$\overline{ERR}$线就被启用。

若 FPGA 的重新配置不要错误检测、线的数目可减少，而将$\overline{CS}$与 GND 相连，这样 FPGA 总被选中。单片机只需提供信号$\overline{CON}$、配置时钟 CCLK 和数据线。此方案使用可节省空间。

如图6.6所示的平行数据传输，8条数据被用来传输一个数据字节，而不是一个数据比特

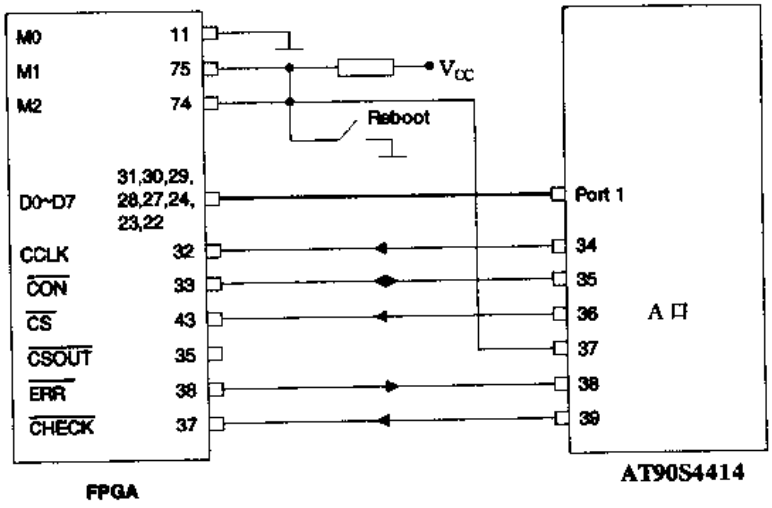


图 6.6 并行数据传送

每时钟周期。这样配置时间就短多了。为了做这些数据线,单片机的普通数据总线也被使用,而数据传输是由控制器的$\overline{WR}$信号来控制的。FPGA 的重新配置就像写入一个外部 RAM。

对于一个 FPGA 的重新配置构成固定系统部门功能的一个系统来说,这可能是较灵活的解决办法。在 FPGA 的附加参考数据中所阐述的状态 6 中,几个 FPGA 也可以串联。这种情况下,除了$\overline{CS}$以外所有线都要平行地连到所有 FPGA 上。$\overline{CS}$与链中前一个 FPGA 的 LCSOUT 相连,以一个比特流来配置几个 FPGA。在一个存储器映射中,单片机控制 LCS 针来选择要重新配置的 FPGA。精确定时如图 6.7 所示。

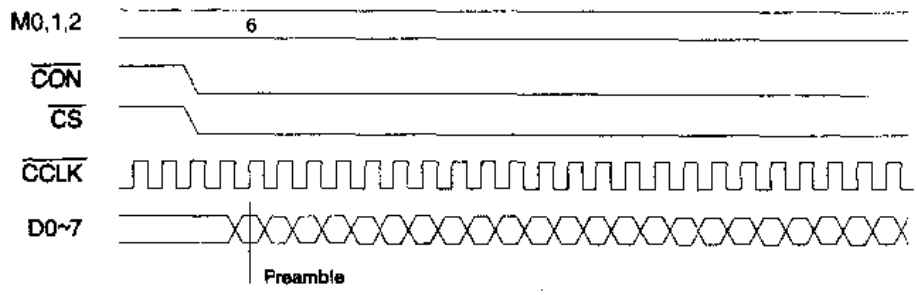


图 6.7 FPGA 选择时序

三、储存配置数据的选项

将配置数据储存在系统内方法有多种,每种方法都有优点和缺点。

第一种方法是使用控制器的内部存储器来储存配置信息。乍看上去,这可能是一种浪费,但是它提供了与众不同的优点,以使每种应用能够实现。单片机 AT90S4414 有 4K 的内部存储器,这样 AT6002 的满配置和附加的 1.4K 程序可被储存。这种情况下只有两个芯片便组成整个系统,并节省了空间。配置数据被控制器的锁定位保护,以防止被反向设计(Reverse-engineered)。

另一种方法是将数据储存在单片机片内的并行 Fsash 或 EPROM 存储器。如果一个 Fsash 存储器被使用,将新的配置下载到 Flash 存储器即可使用。配置通过单片机的单链而改变,FPGA 的配置数取决于存储器的大小。例如:对于一个 5 000 门 FPGA,AT6005 只需 8K 配置数据。AT90S4414 片外的 64K 数据存储器可达到 9 个满配置,或者更多的部分配置,并且其比特流更小了,这主要看一个周期内有多少改变了。

控制 FPGA 配置的软件驻在单片机的内部存储器中,并只需要知道比特流的起始地址。它可以用寻找尾端的方法自己检测到结尾。另一种方法是一个控制程序,上电后在这个数据存储器中寻找首端(比特流);然后,控制器收到命令将第二个配置通过串行口下载到第三个 FPGA,并将这个配置下载到 FPGA 中,并依次类推。要控制这个进程,仔细的系统计划是必要的,以避免毁掉一个工作功能。

当使用并行存储器,配置非常快。串行存储器有更大的空间效益,但很慢。在一些对空间很敏感的应用中,这可能是一种解决办法。有一些串行存储器可直接与一个 FPGA 连接,但只允许一个配置,并且很昂贵。当使用一个单片机时,一般都使用标准串行存储器,它更便宜,还能装不止一个配置。例如:如果使用一个有 I^2C 总线接口的 AT24C64 串行存储器,可为 AT6002 储存 3 个以上的全配置。

四、密码与安全

SRAM 的 FPGA 总是从外界接收它们的配置。因而可能会有些问题,如重新配置和测试性、基于安全和设计保护问题等。当配置是存在一个串行或并行存储器中,并由 FPGA 直接读取,那么这个存储器就可以被拷贝。这种情况下,没有有效的保护措施。

这个问题并不像它看起来那么难,因为单单有配置拷贝还不行。这只能用来拷贝系统,但

是 FPGA 逻辑功能是很难从比特流编译出来的。比特流中的某一个比特和它控制的功能是很难确定的。所以,FPGA 实现的那个环路很难 Reverse-engineered。

当使用一个单片机来配置 FPGA 时,要补充附加的保护装置。比特流可以在贮存到系统存储器之前编码,这样单片机可在发送给 FPGA 之前将比特流解码。密匙被藏在单片机中或用外部方法,如智能卡或身份码。

甚至只用非常基本的操作,就可达到一个很高水平的安全。例如:如果比特流的所有比特都倒转了,配置比特流就会对 FPGA 毫无用处。另一种办法是通过一个表将一些字节与另一些字节交换。这是一个很简单而且快速的操作,它可以将配置进程稍做慢了点,却达到一个高层次的保护。

用上述的方法(将配置信号存在单片机的内部存储器中)有其它优点。有了单片机的锁定位,存储器被禁进入,甚至单片机被放在编程器(开发器)上都不行。有了 ATMEL 单片机的片内擦新功能,当系统部分被进入,如:打开 case 或连续三次输入错误 id,整个存储器数组可在 10 ms 之内被清除。当配置并没存在单片机中,只有密匙存起时,这种方法也是可行的。

这个系统仍然还有弱点。这是由 FPGA 和单片机之间的数据和控制线造成的。配置信息可用逻辑分析仪取样并从时序图中推断出来,虽然困难,但并非不可能。只须知道那些在系统中使用的部件、正确的密匙,或 ID 和一个反编译的运行系统。只有这样,才知道某一给定时刻和配置,采取的方法可把设计分为 2 部分或 3 部分。主要部分没有编码,这样的系统是很难被拷贝或被 Reverse-engineered。另外的小技巧:将 FPGA(客户定制 ASIC),附加电源和接地脚来掩饰被使用的部分。提供了所有这些方法,拷贝设计的过程是复杂的,没有绝对的安全。

6.3 串行 EPROM 接口方法

一、AT90SXXXX 单片机与 AT24CXX 串行 EEPROM 的接口方法

与并行器件相比,串行存储器有差异于数据传输率低、芯片体积小,并且需要的引脚少。

1. 硬 件

图 6.8 给出了 AT90SXXXX 单片机和 AT24CXX 串行 EEPROM 的典型连接方法。如图

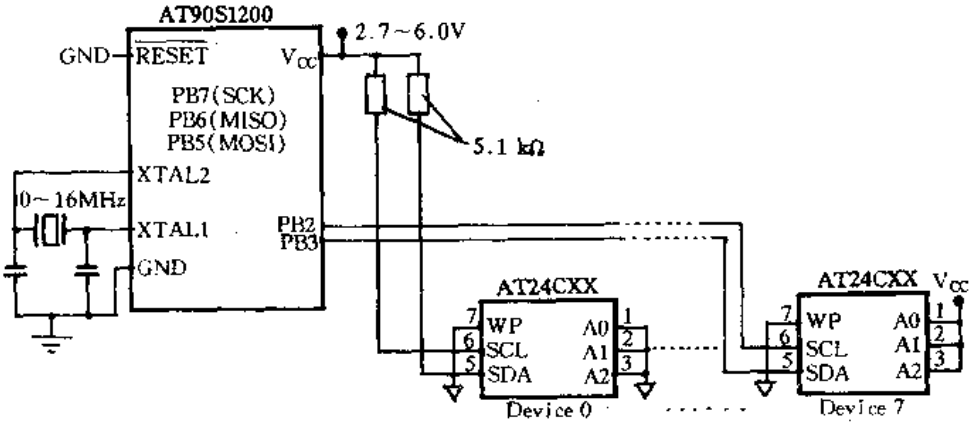


图 6.8 AT90SXXXX 和 AT93CXX 的二线接口

所示,八个 AT24CXX EEPROM 可以共用一条总线,共用单片机的 2 个 I/O 引脚。总线上每

一个存储器的地址输入(A0、A1 和 A2)的硬件连线是唯一的。在这个图中,第一个存储器的识别地址为 0(A0、A1 和 A2 接地),第八个存储器的识别地址为 7(A0、A1 和 A2 接高电平)。并非所有的 AT24CXX 的成员都识别所有的三个地址,此时总线连接该类存储数目必须小于 8 个。表 6.1 给出了公用总线存储的每种类型的精确数据。

表 6.1 公用总线存储的每种类型的数据

器 件	尺寸(字节)	页尺寸(字节)	最大总线	使用地址	器 件	尺寸(字节)	页尺寸(字节)	最大总线	使用地址
AT24C01	1K	8	1	无	AT24C16	16K	16	1	无
AT24C01A	1K	8	8	A0, A1, A2	AT24C164	16K	16	8	A0, A1, A2
AT24C02	2K	8	8	A0, A1, A2	AT24C32	32K	32	8	A0, A1, A2
AT24C04	4K	16	4	A1, A2	AT24C64	64K	32	8	A0, A1, A2
AT24C08	8K	16	2	A2					

2. 双向数据传输协议

双向数据传输协议是利用一个通用的双线总线。AT24CXX 系列允许一定数量的兼容器件共用这条双线总线。这种总线由一条串行时钟线(SCL)和一条串行数据线(SDA)组成。时钟是由总线主控器产生,且数据是在数据线上串行传送时最高有效位在前面(传送的高位在前),同步于时钟,这个通讯协议支持八位的双向数据传送。

在这个应用中,单片机作为一个总线主控器,启动所有传送数据和生成用于调节数据流的时钟。串行存储器是从属部件连在总线上,由主控器控制数据的接收和发送。

总线控制器通过在总线上产生启动信号来启动数据传送,随后则是送一个含有预定接收地址的字节。这个器件地址由 4 个固定部分和 3 个编程部分组成。3 个编程部分用作器件识别地址,而 4 个固定部分则作为存储器地址。当总线有多个类似器件供主控器选择时,固定部分和可编程部分必须配对使用。

AT24CXX 串行 EEPROM 响应这器件地址(固定部分为"1010"和可编程地址符合输入地址(A0、A1 和 A2))。

器件地址的第八位用于说明读或写的操作。在第八位发送后,主控器将释放这条数据线并生成第九个时钟,如果从属部件识别出来器件的地址,它将会在数据线上生成一个确认信息来响应第九个时钟。当从属器件忙时,将不对地址产生确认。比如用 AT24CXX 正在进行写操作时。收到从属器件的地址确认信号以后,主控器继续传送数据。若定义为写操作,则主控器会在收到从属器件确认收到一位的信号后发送剩余的数据。若定义为读操作,主控器将会在从属器件发送数据时放弃数据线和时钟。在收到每一个字节后,主控器将在总线上生成一个确认信号。在收到最后一位字节后确认信号将被省略。主控器通过在总线上产生一个停止信号来终止所有操作,也可以在任何时候通过这个停止信号来中断数据的传送。

AT24CXX 的操作和双向数据传送协议总线时序的细节,可查阅 AT24CXX 的数据资料。

二、AT90SXXXX 单片机与 AT93CXX 串行 EEPROM 的接口方法

连接 AT93CXX 和 AT90SXXXX 的方式可以是三线结构(如图 6.9 所示),或四线结构。在三线结构中 EEPROM 串行数据输入(DI)和串行数据输出(DO)引脚都连到同一条单片机的 I/O 引脚上,从而可以节省一个引脚。可以这样做是因为单片机的 I/O 引脚可以动态地设置为输入或输出。

注意,图 6.9 和图 6.10 中 AT93CXXORG31 脚的连接 ORG(内部组织)引脚接地时,为 8 位;ORG 引脚悬空或接 V_{CC},为 16 位。图 6.9 和图 6.10 的 ORG 仅用于说明连接方式,8 位操作或 16 位操作都可以选择三线结构或四线结构。

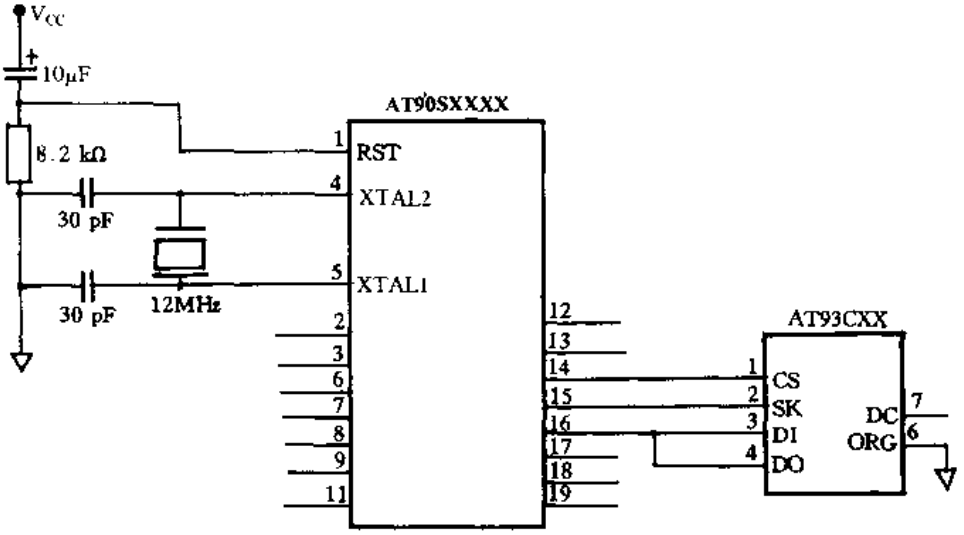


图 6.9 AT90SXXXX 和 AT93CXX 的三线接口

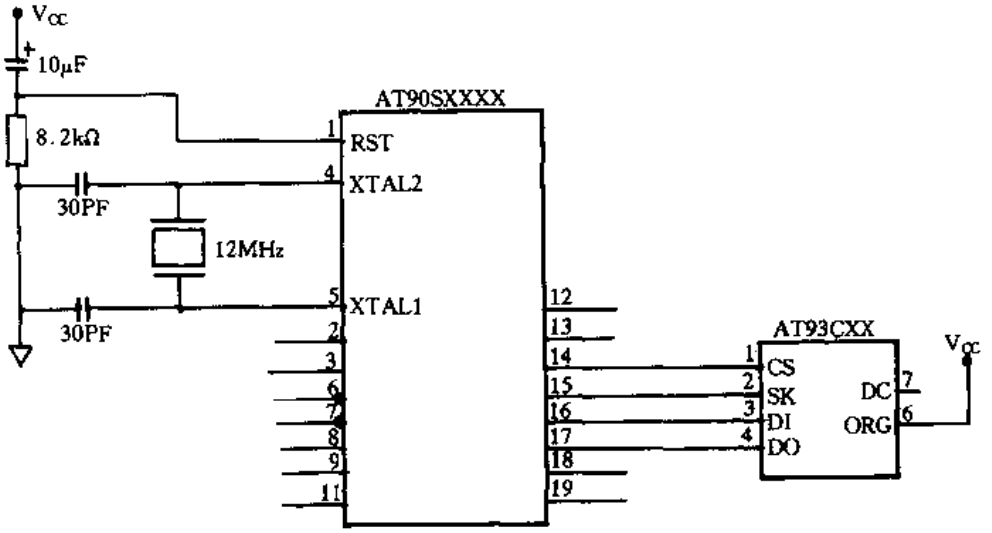


图 6.10 AT90SXXXX 和 AT93CXX 的四线接口

6.4 电冰箱控制器

采用 AT90S1200 单片机构成的电冰箱控制器，电路简单、外围器件少，两路温度测量使用脉冲计数的方法来实现温度的 A/D 转换，利用片内 EEPROM 可以设置参数，通过可编程 I/O 线可以测试欠压、过压状态及输出控制阀门和压缩机。

一、程序流程

电冰箱控制器的程序流程如图 6.11 所示。首先初始化单片机，设置 I/O 口状态，并从片内 EEPROM 中读出温度设置标志以确定温度设定值；启动 A/D 进行温度测量，并显示设定温度；判断是否欠压、过压，确定是否进行欠压、过压处理；判断箱内外的温度，确定是否启动压缩机；采样温度，进行控制；查询按键确定是否升或降温，并改变温度显示设定值；判断是否掉电，如掉电，将设定值写入片内 EEPROM，然后再进入测温、控制循环。

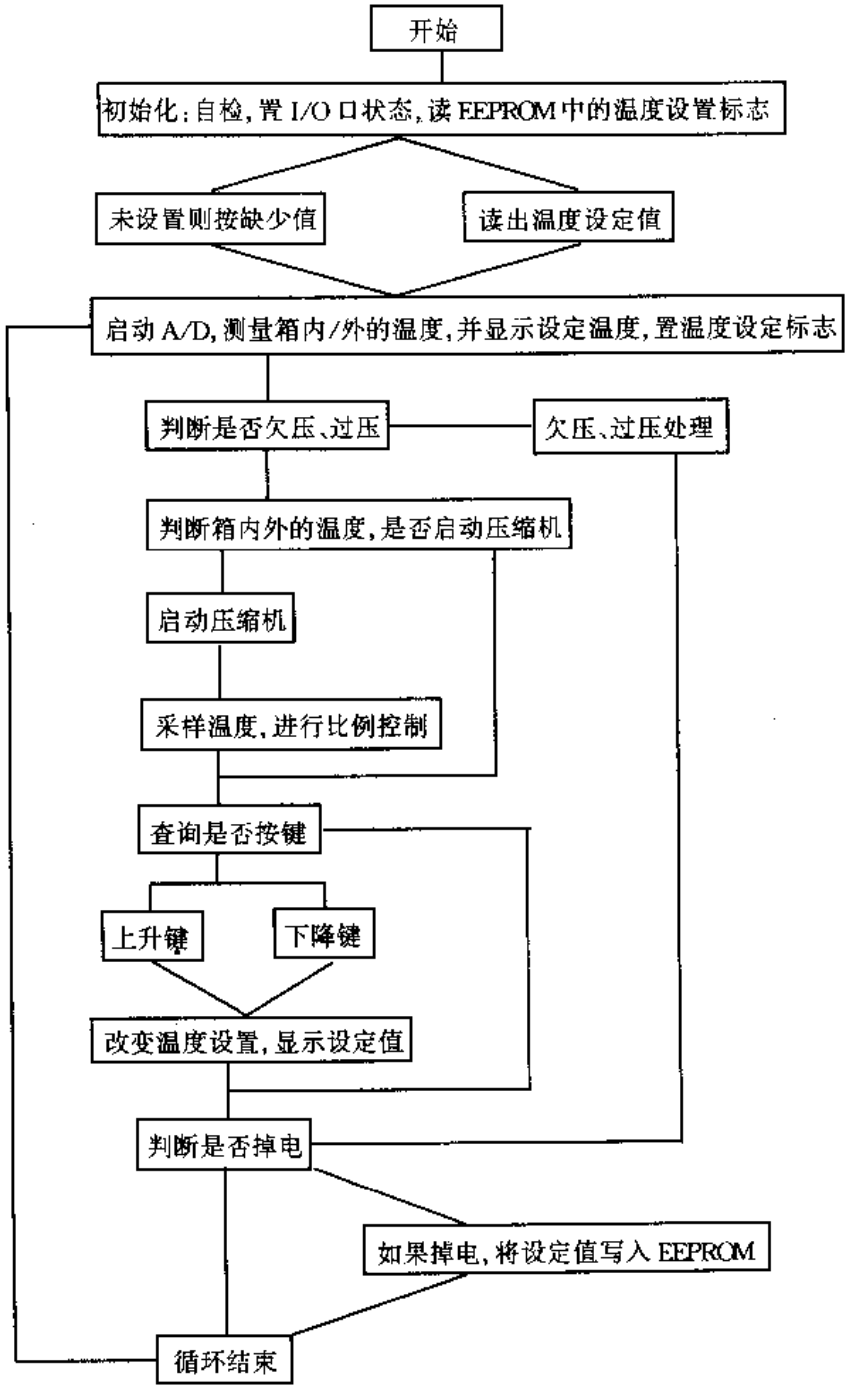


图 6.11 采用 AT90S1200 单片机的电冰箱控制流程图

二、硬 件

如图 6.12 所示为电冰箱控制器的电原理图。由 NE556 和温度传感器及电容组成脉冲振荡器产生计数脉冲，计数脉冲送 AT90S1200 单片机的定时器/计数器 T0 端。当启动 A/D 时，AT90S1200 单片机进行计数来测定电冰箱内/外的温度。PB0(PB3 作为键盘控制线，构成键盘用于温度参数的设定和功能控制；PB4、PB5 作为 NE556 的控制线，用于清除脉冲信号；PB6、PB7 作为显示输出控制线，输出串行信号控制 74LS164 串并转换驱动 LED 显示；PD1～PD6 输出线用于控制压缩机启动或停止。

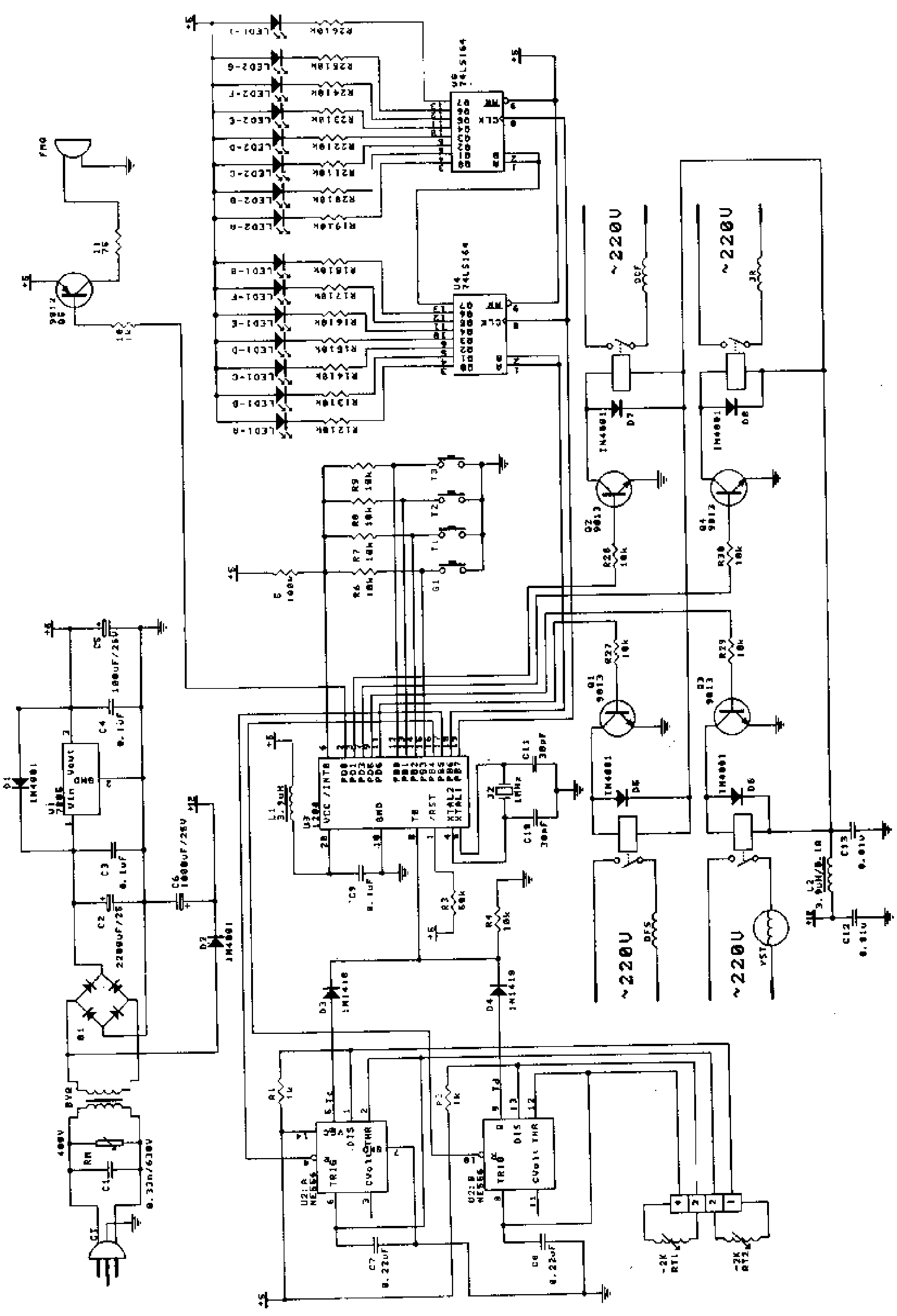


图 6.12 电冰箱控制器电原理

第七章　开发工具

7.1　AVR Studio 调试工具

AVR Studio是AVR微处理器的开发工具。AVR Studio允许用户在AVR在线仿真器或内建AVR指令集模拟器上(软件模拟)控制程序的运行。AVR Studio支持为AVR微控制器编写的汇编程序(用ACMEL公司的AVR汇编器编译)和C程序(用IAR系统的ICCA90 C编译器编译)的源代码层次的执行。

AVR Studio在Microsoft Windows 95和Microsoft Windows NT上运行。

7.1.1　AVR Studio 工具的安装

AVR Studio分装在两张盘上。注意,由于某些原因,第二张磁盘将不被安装程序使用,这是因为运行AVR Studio所需要的一些文件已经存于系统上。

在Windows95和Windows NT 4.0下安装AVR Studio:

(1) 把标有AVR Studio Diskete 1的磁盘插入A驱;

(2) 在任务条上按下开始按钮,选择Run;

(3) 在打开栏中键入"A:SETUP",然后按下OK按钮;

(4) 以后按程序的指示进行安装。

在Windows NT 3.51下安装AVR Studio:

(1) 把标有AVR Studio Diskete 1的磁盘插入A驱;

(2) 从File菜单中选择Run;

(3) 在命令行栏中键入"A:SETUP",然后按下OK按钮;

(4) 以后按程序的指示进行安装。

也可以将两张盘的内容一起拷贝到硬盘上,再安装。

一旦安装了AVR Studio,就能够通过双击AVR Studio的图标运行它。如果一个仿真器是期望的执行对象,记住在运行AVR Studio之前连接好AVR在线仿真器。

下面将对AVR Studio主要特性进行简要的描述。AVR Studio允许在AVR在线仿真器或内建AVR指令集模拟器上运行AVR程序。用AVR Studio运行程序,必须首先用IAR系统的C编译器或用ATEML的AVR汇编器生成一个能被AVR Studio识别的目标文件。

AVR Studio在执行一个程序时的状态如图7.1所示。另外相对于源程序窗口,AVR Studio定义了一些窗口用于观察微控制器的不同源程序。

在AVR Studio中,最关键的窗口是源程序窗口。当打开目标文件时,源程序窗口也就自动产生了。源程序窗口显示在执行对象(如仿真器或模拟器)中,当前正被执行的代码。文本指示总是放在下一条将被执行的语句上,状态条指出执行目标是AVR在线仿真器还是内建指令集模拟器。

缺省时,假设是在源代码层次上执行。所以,如果源程序存在,程序将在源代码层次模式启动。另外,相对于C和汇编程序的源代码层次执行,AVR Studio也能够在反汇编层次上执行程序。当一个执行的程序被停止后,用户可以在源模式和反汇编模式下互相转换。

在AVR Studio中所有必要的执行命令都是有效的,用户可以通过跟踪执行、单步执行功

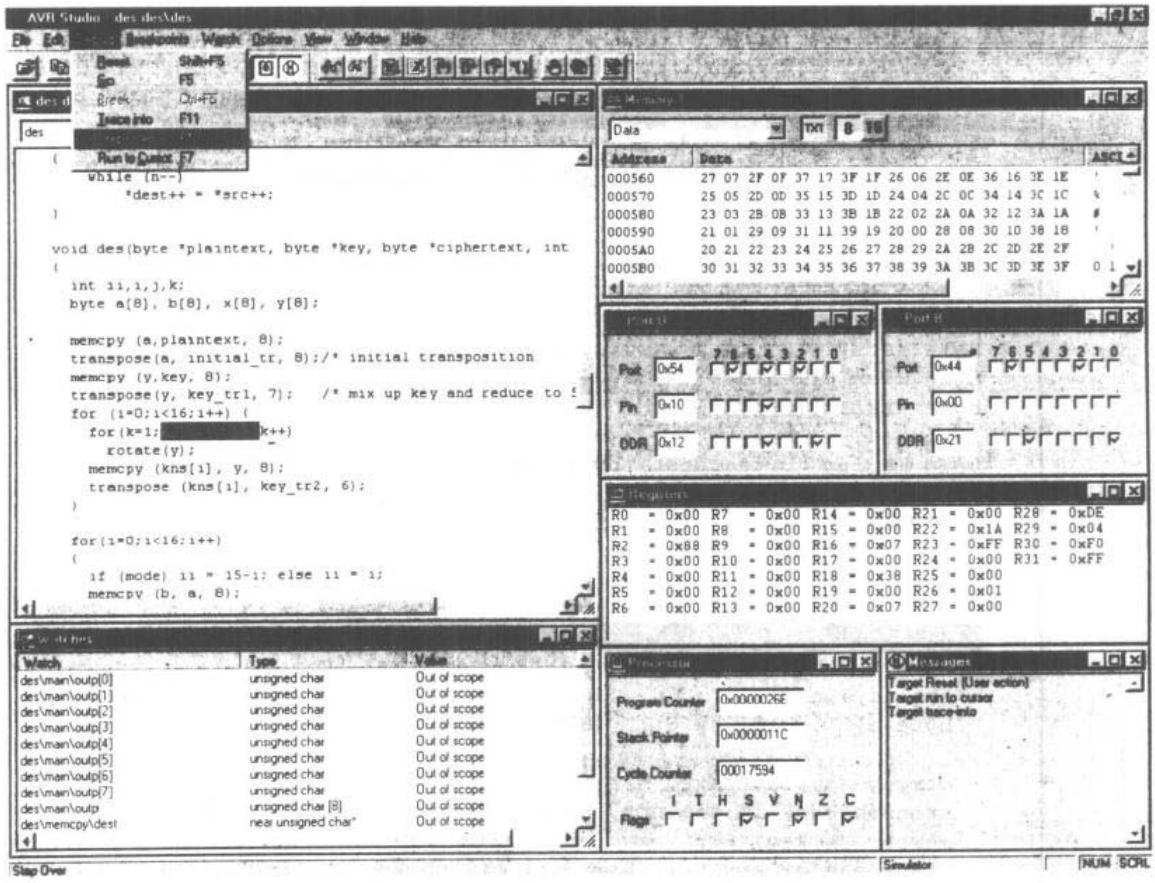


图 7.1

能,把光标放在一条语句上,直到执行到那条语句,停止执行来执行程序。另外用户可以有不限数量的代码断点,每个断点都能定义为可用或不可用。断点在对话中被装入。

源程序窗口给出关于程序控制结果的信息。另外,AVR Studio 提供一些其它的窗口,便于用户对执行目标的每个元素的状态都能完全控制。有如下可用的窗口。

(1) 监视窗口:显示定义符号的值。在监视窗口中,用户可以监视如 C 程序中变量的值。

(2) 寄存器窗口:显示寄存器文档的内容。当执行停止后,可用于修改寄存器的内容。

(3) 存储器窗口:显示程序存储器的内容。如数据存储器、I/O 存储器或 EEPROM 存储器,这些存储器可用十六进制值或 ASCII 码观察。当执行停止后,存储器的内容都能被修改。

(4) 外设窗口:显示与不同外设相联系的状态寄存器的内容,如 EEPROM 寄存器、I/O 端口、定时器等。

(5) 信息窗口:显示 AVR 给用户的信息。

(6) 处理器窗口:显示执行目标的重要信息。包括程序计数器、堆栈指针状态寄存器、时钟周期数。当执行停止后,这些单元可以被修改。

当首次打开一个目标文件,用户需要设置一些便于程序观察的窗口,由此把屏幕上的信息做成特殊的工程文件。当下次这个目标文件被调入时,设置就自动的恢复了。

7.1.2 AVR Studio 窗口

一、源程序窗口

在 AVR Studio 对话框中源程序窗口是一个重要的窗口。当打开一个目标文件时它就被

创建了,并一直贯穿于整个过程。如果关掉源程序窗口,对话也就被中断。

源程序窗口显示当前正被执行的代码。图 7.2 就是一个源窗口的例子。

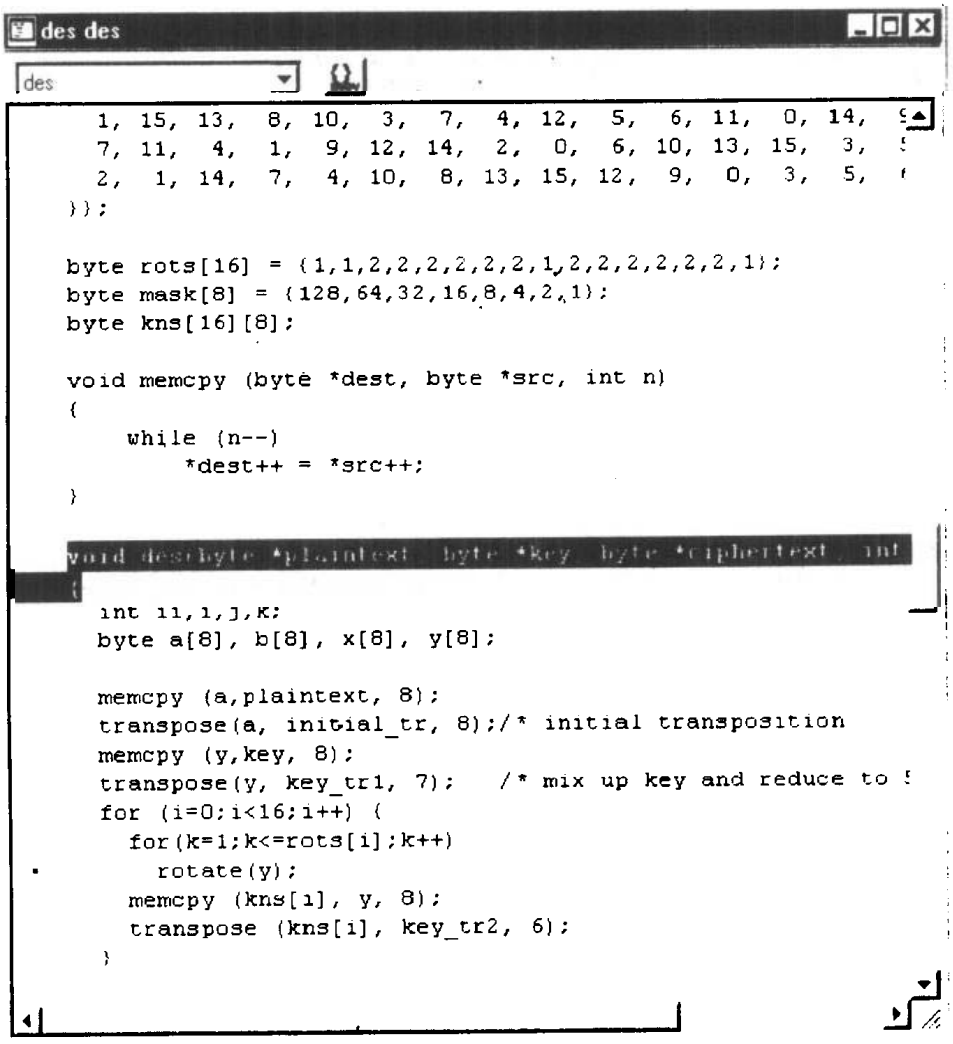


图 7.2

下一条将被执行的指令总是被 AVR Studio 标注着。如果标注被用户移动,在先前被标注的文本变成红色之后,这下一条语句仍被识别。

源程序窗口中,在语句的左边有断点的地方用一个圆点作为断点的标识。

如果按下模式选择框的右边按键,源窗口将在源代码层次和反汇编层次执行上互相转换。当 AVR Studio 在反汇编模式时,所有的操作,例如单步执行将在反汇编级别上执行。如在一些情况中,没有源代码层次的信息可用,(如,把一个 Intel 十六进制文件作为目标文件,没有源代码层次信息可用时),执行只能在反汇编层次上进行。

触发断点,运行到光标处和复制功能,也可以通过在源程序窗口中按下鼠标右键来实现。当鼠标右键被按下,一个菜单就出现在屏幕上,如图 7.3 所示。

如果光标放在一条指令上,一个运行到光标处的命令被给出,则程序将一直执行到光标放置的那条指令上。以同样的方式可以设置断点:把光标放在一条语句上,给出一个 Toggle breakpoint 的命令。如果断点已经设置在这条语句上,则断点将被解除。如果没有断点设置在这条语句上,则就产生一个断点。

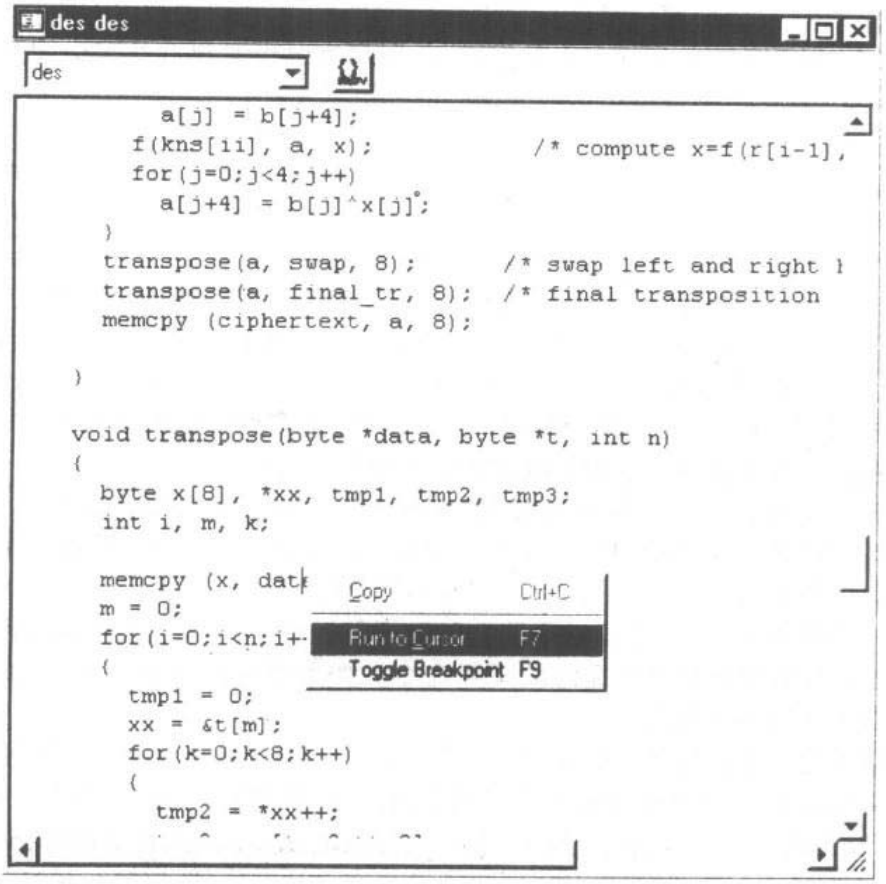


图 7.3

一个目标文件可以由几个模块构成，在同一时刻只能显示一个模块，但是用户可以通过选择在源程序窗口左上方的选择框转换到别的模块。这是一个有用的特性，可以在一个当前模块激活时，在其它模块观察和设置断点。

源程序窗口支持 Windows 剪贴板。用户可以选择源程序窗口中全部或一部分内容，然后通过从编辑菜单中选择复制的方法把它复制到剪贴板上。

二、监视窗口

监视窗口可以显示像 C 程序中变量一样符号的类型和值。因为 AVR 汇编器不会产生任何符号信息，这个窗口只能在执行 C 程序时有意义。图 7.4 给出一个监视窗口的例子。

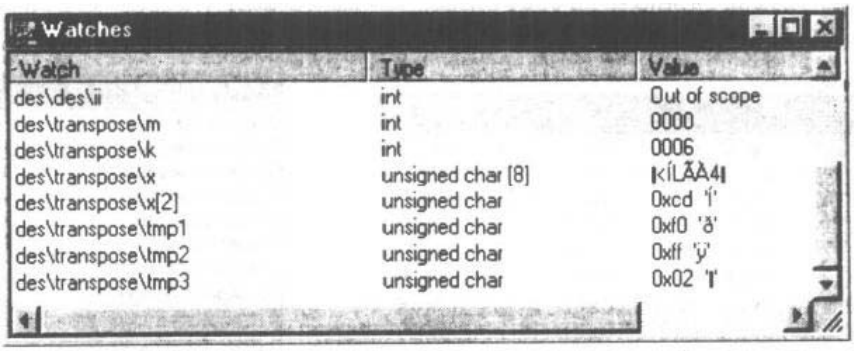


图 7.4

这个监视窗口有三部分,第一部分是被监视符号的名字,第二部分是符号的类型,第三部分是符号的值。监视窗口在缺省状态下是空的,即用户所有想监视的变量必须被加到监视窗口中。一旦一个符号加了进去,在下次程序执行时它就会重新出现。当监视窗口关闭时,这些加入的监视量也被保存。这里有增加监视量、删除监视量和删除所有监视量的命令。一个监视量的加入可以通过从调试条或监视菜单中给出一个增加监视量命令。如果监视窗口是活动窗口,也可以通过按下 Ins 键给出一个增加监视量命令。增加监视命令给出后,用户必须键入符号的名字。用户也可以输入一个带有或没有范围信息的符号名字。

AVR Studio 先以符号包含范围信息来搜寻符号。如果那样的符号没有找到,AVR Studio 把符号的名字放在当前的范围上,然后搜寻这个新符号。如果仍没有这样的符号被发现,这个符号就被解除了。在类型部分出现"???",值的部分一直空着。如果找到了这个符号名称,这个符号就被限制,带有范围信息的符号就显示在监视块,类型和值域也被填充。每当程序执行停止时,AVR Studio 都试着用当前的范围来赋值无限制的符号。

浮点符号是不可用的。一旦符号被限制,它将保持赋值。这些监视量在对话中被保存。不论符号是否被赋值都是这些信息的一部分。如果程序进入一个范围,那里一个被赋值的符号是无效的 ,值域将变成"out of scope"。

为了删除一个监视量,符号名称必须首先用鼠标左键点上。当用这种方法标注了一个符号,AVR Studio 接收监视菜单中的删除监视量命令。如果监视窗口是当前活动窗口,标注的符号也可以通过 Del 键进行删除。

监视窗口可以用于像监视单个变量一样监视 C 数组和结构体。语法同 C 语言(用"[]"定义数组,用"."定义结构体)。没有提及的指针不被支持,当监视数组时,变量可以用于动态地索引数组。例如,可以监视"my _ array[i]",如果 i 是一个与数组 my array 同样范围的整型变量。

同一时刻只能有一个激活的监视窗口,监视符号在对话中恢复,监视窗量也同样地被恢复和保存。

三、寄存器窗口

寄存器窗口显示 AVR 寄存器文件中的 32 个寄存器的内容。图 7.5 所示是一个寄存器窗口的例子。

当寄存器窗口大小变化时,里面的内容会为更好地适应窗口形状而重新组织。当执行停止后,寄存器窗口里的值可以被改变。为了改变寄存器的内容,首先确定执行是停止的,然后把光标放置在要改变的寄存器上,按鼠标左键两下(不是双击,要在两次击打之间有一个停顿),寄存器就可以被改变了,以十六进制的形式键入新的内容。最后,按下回车键确认或 Esc 键撤消改变。

同一时刻只能有一个寄存器窗口被激活。

四、信息窗口

信息窗口显示 AVR Studio 给用户的信息。当一个复位命令执行后,信息窗口的内容被消除。图 7.6 是一个信息窗口的例子。

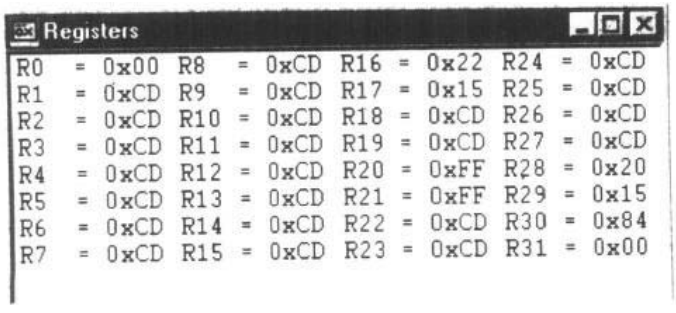


图 7.5

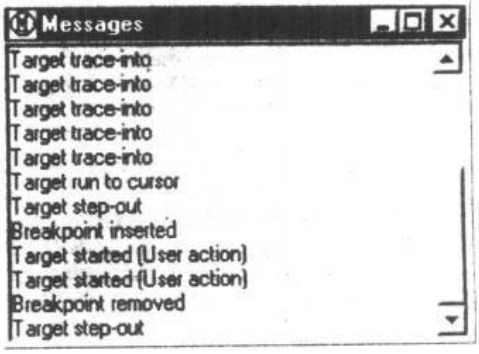


图 7.6

当信息窗口被关闭和再打开时，里面的内容也被恢复，在同一时刻也只能有一个被激活的信息窗口。

五、存储器窗口

存储器窗口允许用户观察和修改当前执行对象的不同存储器的内容。同样的窗口可以用来观察所有的存储器形式，存储器窗口可以用来观察数据存储器、程序存储器、I/O 存储器和 EEPROM 存储器。

用户可以有几个共存的存储器窗口。图 7.7 是一个存储器窗口的例子。

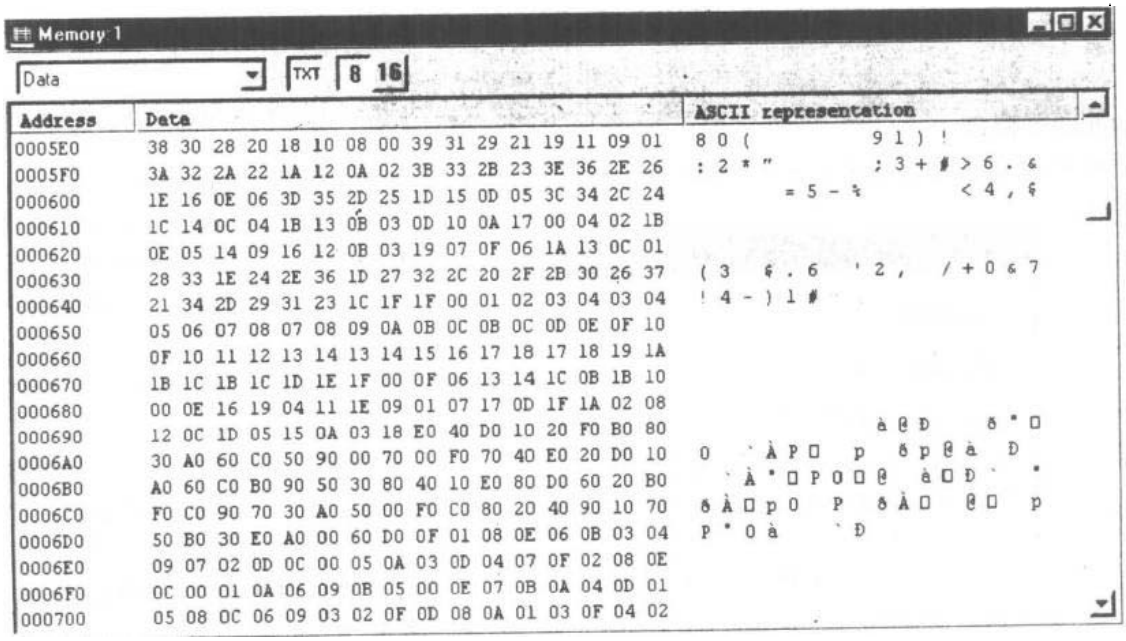


图 7.7

想看哪一种存储器形式，可在存储器窗口左上角的存储器选择框中进行选择。当打开一个新的存储器窗口，数据存储器是缺省的存储形式。

十六进制表示的存储器地址和内容总是显示着的。另外，用户还可以看到存储器内容的 ASCII 码表示。用户也可以选择把十六进制值组织成 十六位组或八位组。

当看程序存储器时，在地址列中显示的是字地址。在数据列中，MSB 列在 LSB 列前。

用户可以通过在包含被修改项的行上双击来修改存储器内容。当存储器上的一行被双击，一个窗口出现在屏幕上。如果存储器以八位组观察，修改也是以八位组操作；如果以十六位组观察，则修改也是以十六位组操作。

当以八位组操作，就会出现如图 7.8 所示的窗口。

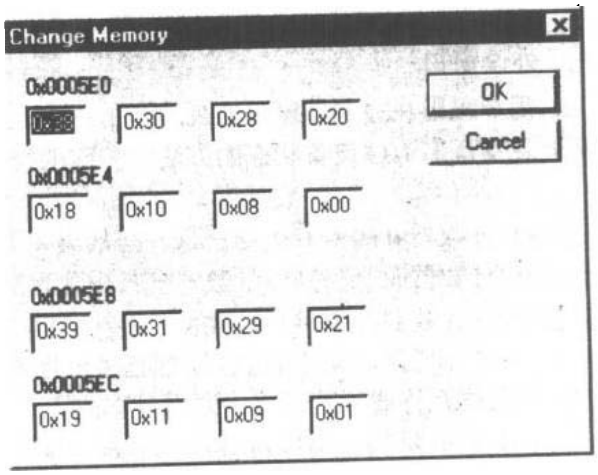


图 7.8

当以十六位组操作，就会出现如图 7.9 所示的窗口。

在两种情况下操作是相同的。如果按下 Cancel 键，不会有任何改变；如果按下 OK 键，存储器内容就会根据改变而更新。

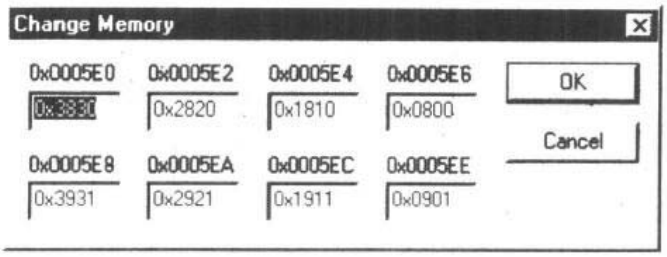


图 7.9

六、处理器窗口

处理器窗口包含执行对象的重要信息。图 7.10 是一个处理器窗口的例子。

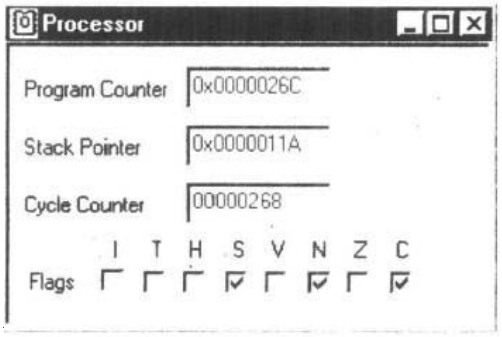


图 7.10

程序计数器指出下一条将被执行指令的地址，且以十六进制形式显示，当执行停止后也可以被改变。当前的指令在程序计数器改变时被取消了。程序计数器改变后，用户必须执行单步功能跳到新的地址上。

堆栈指针装着放在 I/O 区的堆栈指针的当前值。如果执行对象有一个硬堆栈代替一个基于 SRAM 的堆栈，就会在堆栈指针域中指出。当执行停止后，堆栈指针值可以被改变。

周期计数器给出自上次复位取消后时钟周期的数据，AVR Studio 在线仿真器运行时不支持周期计数器，因此以仿真器为执行对象时，周期计数器一直为 0，且以十进制显示，在执行停止后也能被改变。

标志位显示当前状态寄存器的值。当执行停止时，这些位可以通过在这些标志上单击来改变。复选的标志指出这个标志被置位(在状态寄存器中相应位是 1)。

在同一时刻只能有一个处理器窗口被激活。

七、外设窗口

用户可以在存储器窗口监视 I/O 的内容，但以连续存储结构的方式查看 I/O 区域，并不是方便地观察众多 I/O 设备状态的方法。特殊的外设窗口更具体，使观察 I/O 设备变得简单。

(1) 8 位定时器/计数器窗口(或 16 位定时器/计数器) 8 位定时器/计数器窗口显示 8 位定时器/计数器 0 的所有重要信息。当从菜单 View→Peripheral→8bit Timer/Counter 中选择 8 位定时器/计数器时，定时器/计数器窗口出现在屏幕上，如图 7.11 所示。定时器/计数器框给出 8 位定时器/计数器的值，初值框给入相应的初值，0 通常指出定时器/计数器是关闭的。初值必须在 0～7 之间选择。溢出标志控制框在溢出标志位为 1 时复选，在溢出标志位为 0 时为空。溢出中断框根据允许或屏蔽溢出中断而复选或空。当执行停止后，所有的值都可以被用户改变。

(2) 端口窗口 端口窗口显示通常与一个端口联系的三个不同 I/O 寄存器。当用户从菜单 View→Peripheral→port 中选择一个端口，相应的端口窗口就会出现如图 7.12 所示。端口窗口以十六进制值和一位值显示 I/O 区域的端口的设置，引脚和直接数据寄存器的值。当执行停止后，寄存器的值都可以被改变。

(3) EEPROM 寄存器 当从菜单 View→Peripheral 中选择 EEPROM 寄存器时，图 7.13

所示的窗口就会出现。AVR Studio 知道在执行对象上有多少 EEPROM 存储器可用，因此如果需要，地址框中包含地址的高位和低位。

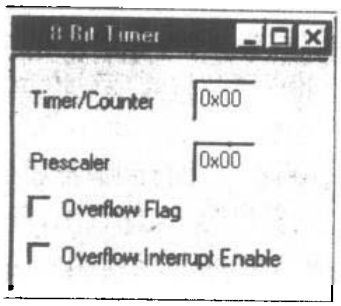


图 7.11

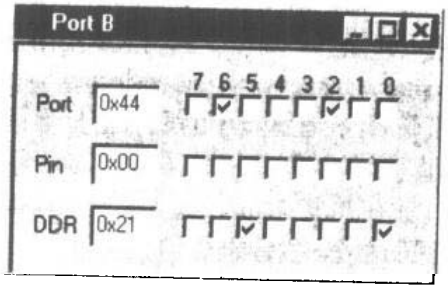


图 7.12

图 7.13

7.1.3 AVR Studio 命令

AVR Studio 使用了一定数量的命令。命令可用通过不同的方式给出：菜单选择、工具条按钮和热键。

一、管 理

(1) 打开文件 当从 File 菜单中选择了 Open，一个文件选择对话框就会出现在屏幕上(注意 AVR 设定文件扩展名为 OBJ，因此在缺省状态下，仅是有 OBJ 扩展名的文件被列出)。用户必须选择目标文件去执行，AVR Studio 支持下面的格式：

- IAR UBROF
- 通过 ATEML AVR 汇编器生成的 AVR 目标文件
- Intel－Hex

AVR Studio 自动识别目标文件格式。四个最近使用的文件可以在 File 菜单中直接被选择装入。

当打开文件时，AVR Studio 寻找具有相同名字但扩展名为 AVD 的文件。它包含关于工程的信息，包括窗口的位置。如果 AVD 工程文件没有找到，那就仅创建一个源程序窗口。

AVD 文件也包含关于断点的信息。在上次对话中定义的断点重新被装入。如果目标文件比工程文件新，断点就被忽略了。

如果源代码层次信息可用，程序将执行到第一条源语句。

(2) 关闭文件 当从 File 菜单中选择 Close，对话中所有的窗口都被关闭。AVR Studio 还在同目标文件一样的目录下写一个文件，包含项目信息。这个文件与目标文件的名称相同，但扩展名为 AVD。

(3) 复制文本 用户可以在源程序窗口中标记文本，并通过选择 Edit 菜单中的 Copy 命令把文本复制到 Windows 剪贴板中。

二、执行控制

执行控制用于控制程序运行，所有执行命令通过菜单、热键和调试工具条给出都有效。

(1) 全速执行 Debug 菜单中的 GO 命令开始(或确定)程序的执行。程序一直执行到被停止(用户操作)或断点出现，GO 命令仅当程序停止时有效。热键为 F5。

(2) 运行停止 Debug 菜单中的 Break 命令用于停止程序的执行。当运行停止后，窗口中所有的信息都被更新，Break 命令只在程序运行时有效。热键为 Ctrl + F5。

(3) 跟踪进入 Debug 菜单中的 Trace into 命令执行一条指令。当 AVR Studio 是在源模式，一条源代码层次的指令被执行。如果在反汇编层次，一条汇编指令被执行。当 Trace into

执行后,所有窗口中的信息都被更新。热键为 F11。

(4) 单步执行 Debug 菜单中的 Step over 命令执行一条指令。如果指令包含一个功能/子程序调用,功能/子程序调用也被执行。如果在 Step over 中有用户断点,执行就被挂起。当 Step over 执行后,所有的窗口中的信息都被更新。热键为 F10。

(5) 单步退出 Debug 菜单中的 Step out 命令执行直到当前功能完成。如果在 Step over 中有用户断点,执行就挂起。如果当程序在顶层时 Step out 命令就发出,程序将一直执行到一个断点或被用户停止。当 Step out 命令完成后,所有窗口中的所有信息都被更新。热键为 Shift + F11。

(6) 运行到光标处 Debug 菜单中的 Run to Cursor 命令执行,直到程序运行到源程序窗口中光标指出的那条指令。如果在执行 Run to Cursor 命令时有一个用户断点,执行不被挂起。如果光标指出的指令不能到达,程序只能被用户停止。当 Run to Cursor 命令完成后,所有窗口的信息都被更新。热键为 F7。

(7) 复位 Reset 命令完成执行对象的复位功能。如果程序正在执行,当这个命令给出时,执行将被停止。如果用户在源代码层次,程序在复位后,将一直执行到第一条源语句。当复位命令完成后,所有窗口中的信息都被更新。热键为 Shift + F5。

三、监视量

当在 C 语言源层次上执行,监视窗口可以用来监视符号。当执行由 ATEML AVR 汇编器生成的目标文件时,没有符号信息是可用的,因此监视窗口不能用来监视任何信息。

(1) 添加监视量 为了插入一个新的监视量,用户必须在监视窗口中选择 Add Watch 命令或者在 Debug 工具条中按下 Add Watch 按钮。如果当 Add Watch 命令给出时,监视窗口未被打开,将创建一个监视窗口,已经定义的监视量被插入。如果监视窗口是活动窗口,增加一个新的监视也可以通过按下 Ins 键。

(2) 删除监视量 用户可以删除一个监视量。首先在监视窗口选定要删除的符号(通过移动鼠标指针指向监视的名字,按下鼠标左键选定一个监视量),然后从 Watch 菜单或调试工具条中给出一个 Delete Watch 命令。如果监视窗口是活动窗口,一个选定的符号也可以通过按下 Del 键进行删除。

(3) 删除所有的监视量 Watch 菜单中的 Delete all watches 命令是有效的。当使用了这个命令,所有定义的监视量都将被从监视窗口中删除。

四、断 点

用户可以设置不限数量的代码断点。除非产生了一个新的目标文件,否则断点就在对话中恢复。如果目标文件比工程文件新,断点也会被忽略。

当在某处设置了一个断点,在指令左边将有一个圆点指出这是一个断点。

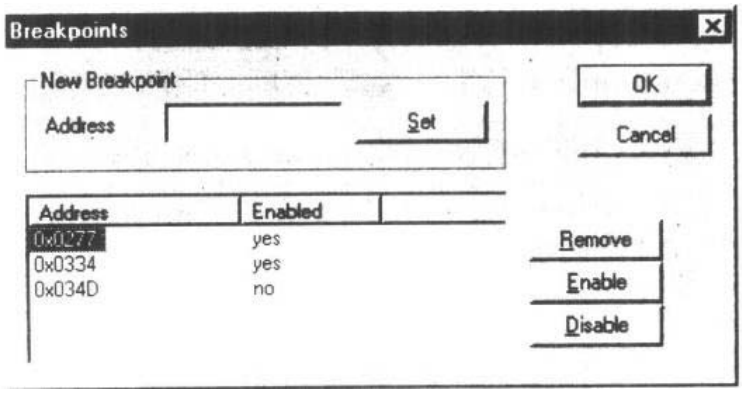


图 7.14

(1) 触发断点 Toggle break point 命令触发光标处指令的断点状态。注意,这个功能仅当源视图是活动视图时有效。

(2) 清除所有的断点 这个功能清除所有定义的断点,包括已经被忽略的断点。

(3) 查看列表 当 Show list 被选择,图 7.14 所示的对话框就会出现。

在断点对话框中,用户可

以观察已经存在的断点，增加一个新的断点，删除一个断点或允许/屏蔽断点。

五、工具条

AVR Studio 包含下面描述的三种不同的工具条。工具条可以被单独地插 入和移出。如果需要，也可通过 View→Toolbars 菜单控制或解控。

(1) 通用工具条 包括标准窗口命令的按钮。通用工具条有图 7.15 所示的按钮。

图 7.15

(2) 调试工具条 包含执行控制和监视窗口控制的按钮。调试工具条有图 7.16所示的按钮。

图 7.16

(3) 视图工具条 包含允许和屏蔽许多常用窗口和增加存储器窗口的按钮。视图工具条有图 7.17 所示按钮。

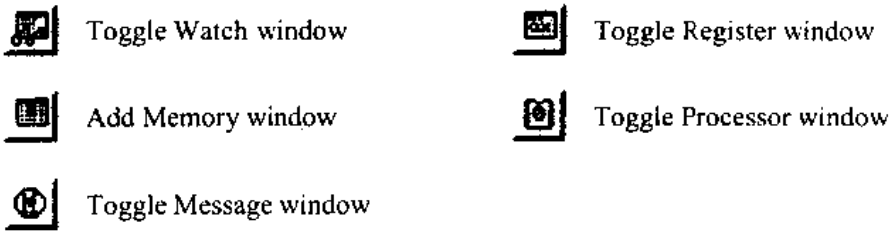


图 7.17

六、热键摘要

部分热键在 AVR Studio 中的定义见表 7.1。

表 7.1 部分热键定义

命 令	热 键	命 令	热 键	命 令	热 键
改变寄存器窗口	Alt + 0	复制到剪贴板	Ctrl + C	运行到光标处	F7
改变监视窗口	Alt + 1	打开文件	Ctrl + O	改变断点	F9
改变信息窗口	Alt + 2	帮助	F1	单步执行	F10

表 7.1 续

命 令	热 键	命 令	热 键	命 令	热 键
改变处理器窗口	Alt + 3	全速执行	F5	跟踪进入	F11
增加存储器窗口	Alt + 4	运行停止	Ctrl + F5	单步退出	Shift + Fll
显示断点列表	Ctrl + B	复位	Shift + F5		

7.1.4 执行对象

AVR Studio 可以面向一个 AVR 在线仿真器或内建 AVR 指令集模拟器。当用户打开一个文件，AVR Studio 自动地识别系统的一个串行端口是否有一个仿真器可用。如果发现了一个仿真器，就把它作为执行对象。如果没有仿真器，则执行被 AVR 内建指令集模拟器(软件模拟)代替。状态条指出执行目标是在线仿真器还是内建指令集模拟器。

一、AVR 在线仿真器

如果系统中 AVR 在线仿真器有效，它就会自动地被作为执行对象。仿真器必须连接在一个串行口上。如果仿真器已与系统相接，但是不能被识别，关掉文件，复位仿真器，再试一次。想了解关于 AVR 在线仿真器的信息，请看 AVR ICE 用户手册。

若用户想用模拟器，即使仿真器在系统上，也可用在打开文件前断开或关掉仿真器。

当执行停止后，仿真器的速度可以改变。为了改变速度，从 Option 菜单中选择 Emulator 选项。图 7.18 所示的窗口就会出现。

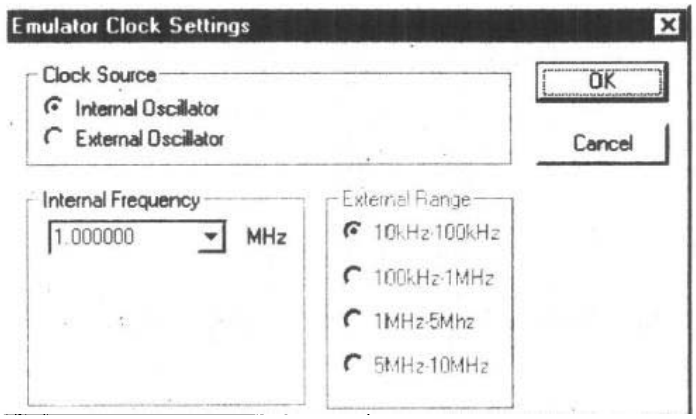


图 7.18

用户可以选择仿真器是用板上的可编程时钟，还是用外部时钟(请看在线仿真器手册以获取更多的信息)。如果内部晶振作为时钟来源，用户可以选择从 400～20MHz 的频率。用户可以选择列表中的典型频率，也可以在信息窗口自定义一个频率。在对话框中仿真器的速度被恢复。

二、AVR 指令集模拟器

如果 AVR Studio 不能成功地证明存在一个仿真器，它就用一个内建模拟器代替。模拟器也支持一些 I/O 设备，有不少不同的选项。如果从 Option 菜单中选择 Simulator Options，图 7.19所示的对话框就会出现在屏幕上。

在对话框中模拟器选项也被恢复。对话框中所有的数据都是十进制值，改变这些值的任何一个都会迫使 AVR Studio 执行一个复位命令。

(1) 设选择　在设备盒中，用户可以在 6 种不同的标准配置中选择。如果选择了一个标

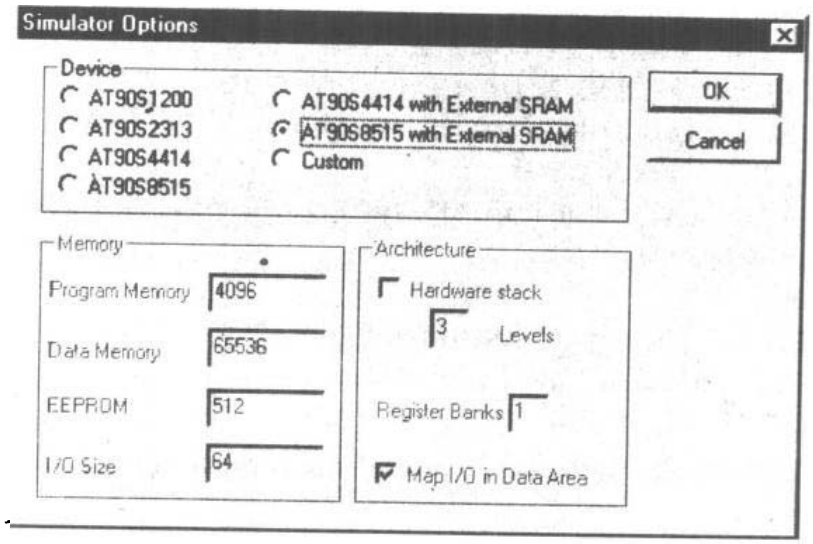


图 7.19

准配置,存储器配置和结构细节就相应地填写了,用户也可以定制配置。

(2) 定制选择 如果定制选择按钮被按下,则允许用户输入存储器和结构栏的值。

· 程序存储器大小以字计算。可以键入的最大值为 65 536。超出这个选择范围,操作是不确定的。

· 数据存储器以字节计算。可以键入的最大值为 65 536(64K 字节)。注意指针寄存器不是循环的。如果用户使用 SRAM 超过选择的范围,操作是不确定的。在数据存储器栏中的值是能用于寻址 SRAM 的最高地址。寄存器文档的大小和 I/O 区(如果映射在 SRAM 区)必须加到 SRAM 的值上,以得到这个栏的正确值。

· EEPROM 的大小以字节计算。可键入的最大值为 65 536(64K 字节)。EEPROM 地址寄存器仅仅包括能够寻址 EEPROM 的必要多的位。

· I/O 大小以字节计算。I/O 大小允许的值为 64,128,256。

· 用户可以选择模拟器是否使用硬堆栈。若选择了硬堆栈,用户可以设置硬堆栈的数目。

· 用户不能让模拟器管理几个寄存器文档。

· 用户可以选择 I/O 区域是否占用 SRAM 的地址空间。如果 I/O 可以在 SRAM 区寻址,它将在地址 0x20 以上使用。

(3) 记录端口 用户可以记录输出端口的活动。如果从选项菜单中选择了 Simulator Port Logging,图 7.20 所示的对话框就会出现在屏幕上。

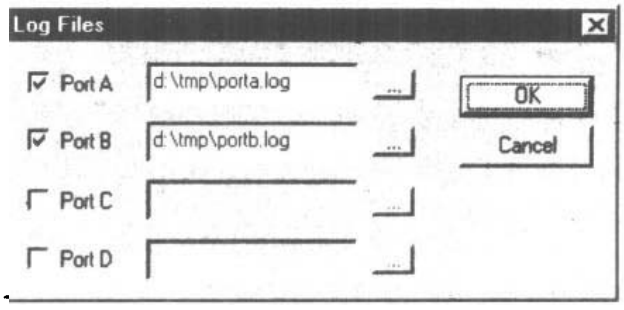


图 7.20

用户必须选择的那个端口将被记录。如果选择了一个端口,用户还必须选择一个文件用于放置记录数据。文件上的内容是端口寄存器的内容。记录文件的每行都有下面的格式,周期:数据。周期栏中以十进制格式,数据栏中以十六进制格式。如果在一个周期中端口寄存器的内容没有改变,就没有输出产生。记录文件在每次程序复位时被删除。在每次程序装入AVR Studio时,记录都必须被人工激活。

下面是一个AVR Studio产生的记录文件的例子。在这个例子中,FF是在周期43时写到端口的,而FE是在周期80时写到端口的。

000000000:00 000000090:AB
000000043:FF 000001021:FF
000000080:FE

三、外部激励

用户可以设置端口的值。如果从Options菜单中选择Simulator Port Stimuli,图7.21所示对话框将出现在屏幕上。

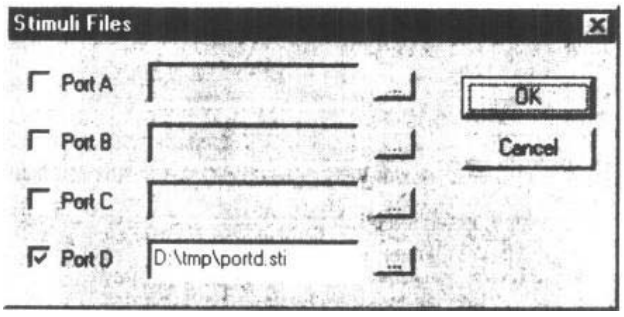


图 7.21

如果选择了一个端口,用户必须指出激励文件的位置。激励文件中的值将在指定的周期放在指定端口的PIN寄存器中。激励文件的格式与端口记录文件格式相同。注意,仅当引脚设为输入时有效。

四、定时器/计数器0

模拟器支持定时器/计数器0。如果选择了一个标准设备,定时器/计数器0的溢出中断向量和外围计数器引脚被相应地设置。如果选择了定制外围设备,定时器/计数器0的溢出中断向量和外围计数器引脚同AT90S1200设置。

五、外部中断

模拟器支持外部中断。如果选择了一个标准外围设备,外部中断就被相应地设置了。如果选择了一个定制设备,对于AT90S1200将有一个外部中断可用。

7.2 AVR汇编器

AVR汇编器覆盖了AT90S微控制器家族的全部范围。汇编器用于把汇编代码编译成目标代码,生成的目标代码可以用于模拟器的输入或ATMEL AVR在线仿真器的输入。汇编器还能产生能够直接写入程序存储器和EEPROM存储的PROM(可编程只读存储器)代码和一个任意EEPROM文件。

汇编器产生固定的代码分配,因此没有链接的必要。

汇编器可在Microsoft Windows 3.11、Microsoft Windows 95和Microsoft Windows NT下运行。另外,还有一个MS-DOS版本。Windows版本包括一个在线帮助功能,覆盖了所有这

些说明。

7.2.1 编译器快速启动家庭教师

家庭教师确信 AVR 编译器和所有配的程序文件都正确地安装在你的计算机上，请参考安装指令。

一、开 始

开始 AVR 汇编器。通过从菜单中选择“File→Open”或按下工具条中的图标，打开文件“tutorl.asm”。这样就把汇编文件装入了编辑窗口，读读程序头，看看程序，但是不要改动它。

二、编译第一个文件

一旦看完了这个程序，从菜单中选择编译，第二个窗口（信息窗口）就会出现，并包括一些错误信息。这个窗口将会覆盖编辑窗口，所以在屏幕上应先清除工作空间。选择包含程序代码的编辑窗口，并从菜单中选择“Window→Tile Horizontal”。让编辑窗口比信息窗口大一些是比较好的，所以让信息窗口向下移动一点，并让它挨着编辑窗口的底部，屏幕上将会出现如图 7.22 所示的内容。

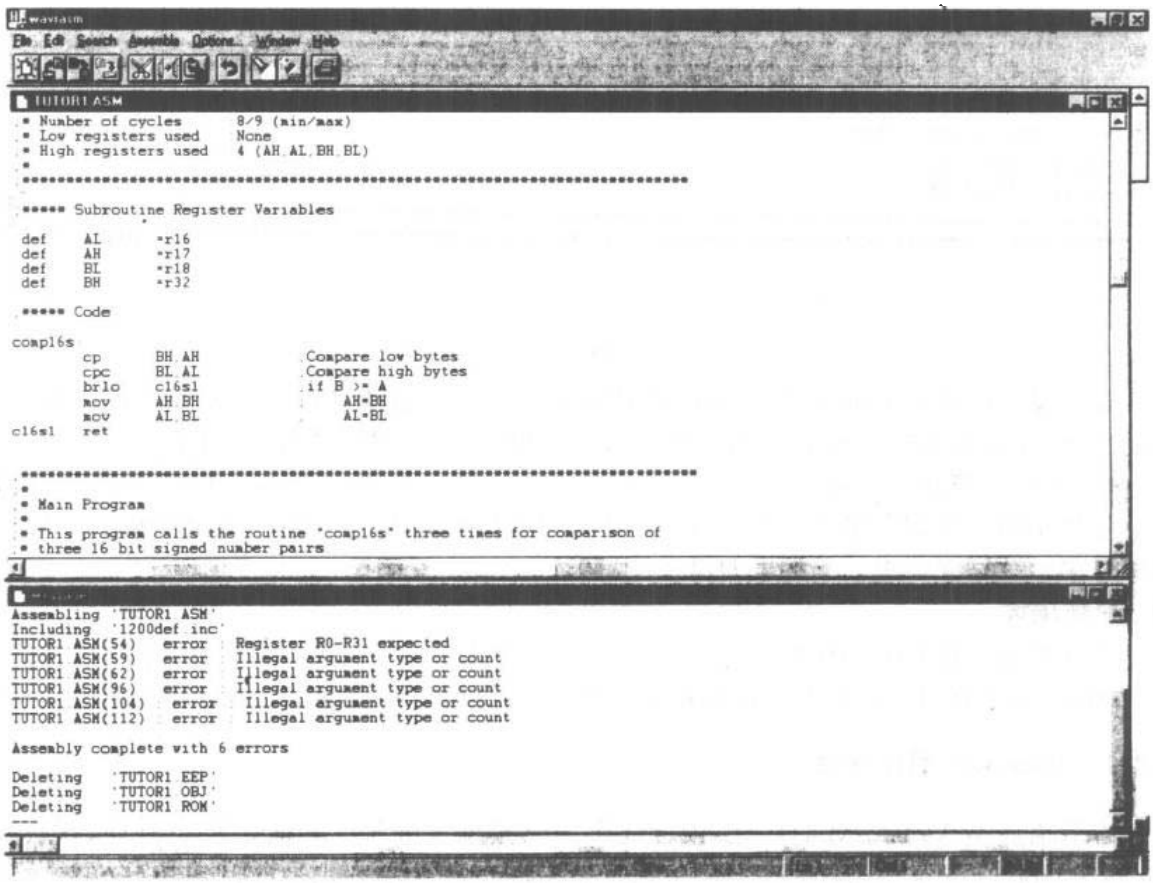


图 7.22

三、寻找和纠正错误

从信息窗口来看，好像正尽力编译一个有很多缺陷的程序，为了进行下一步，错误必须被发现和纠正。指向信息窗口中的第一个错误（在 54 行），按下鼠标左键。注意，在编辑窗口，一个红色的横条覆盖在 54 行上。错误信息指出仅有寄存器 R0～R31 能被设为变量名。AVR 有 32 个通用寄存器从 R0～R31，而“tutorl.asm”试图给寄存器 32 命名，看图 7.23。

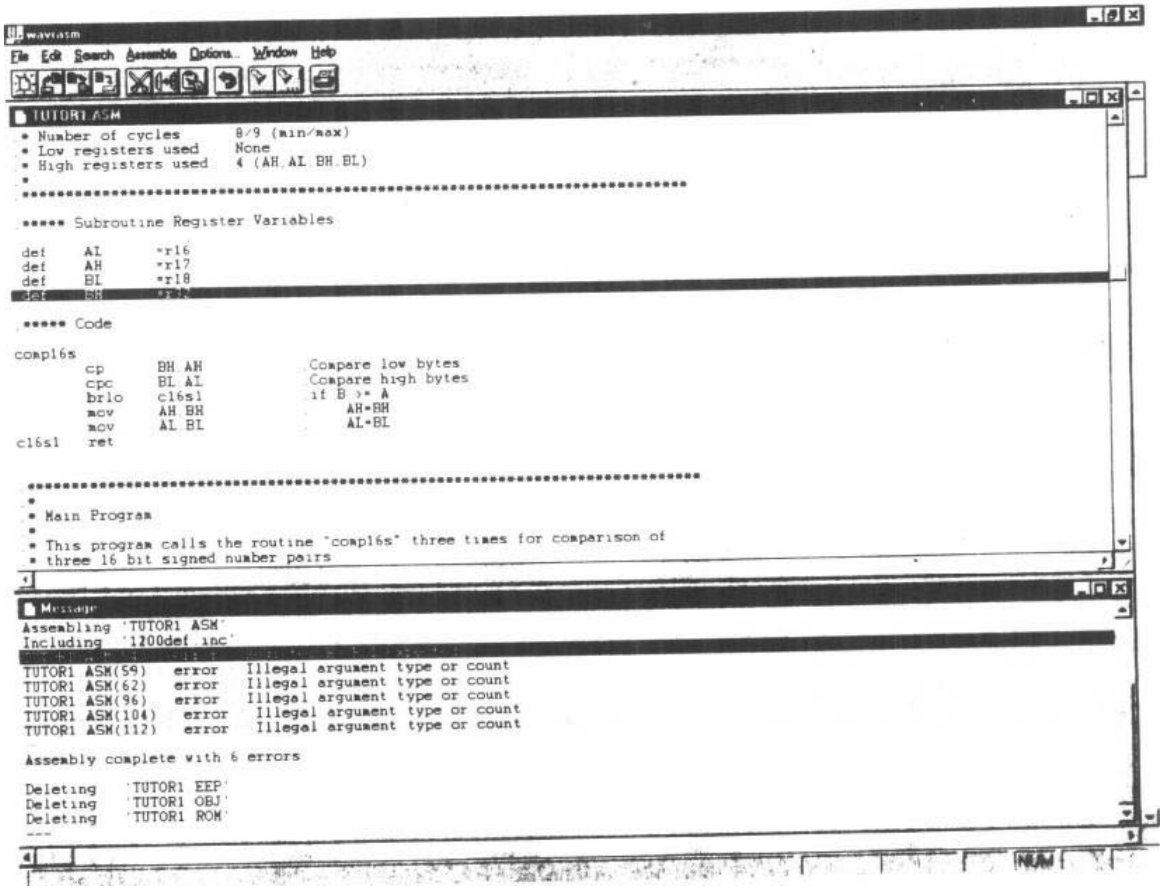


图 7.23

在信息窗口中双击错误信息,可以看到编辑窗口变为活动窗口。光标处在包含错误信息行的头上,在编辑窗口中把 R32 改为 R19 以改正错误,一个错误消失,还有五个。点在列表中下一个错误上,信息"Illegal argument type or count",说出比较(CP)指令中的参数有些错误。注意参数中的一个寄存器名字为"BH",这是我们刚改正过的一个变量,通过单击剩下的错误,表明是第一个错误产生了一连串的错误信息。

四、重新编译

为查明是否所有的错误都已改正,双击任意一个错误(为了激活编辑窗口)或在再次编译之前,单击编辑窗口。如果到现在你都这样做了,信息窗口会告诉你已经顺利完成了。

7.2.2 Microsoft 窗口特性

本节描述 WAVRASM 的特性,仅描述汇编器菜单项的特性。这是假设用户已经对 Serach 和 Windows 菜单项比较熟悉。一个汇编器编辑的典型例子如图 7.24 所示。

一、打开一个汇编文件

可以在 WAVRASM 中打开一个新的或已经存在的文件。理论上来说一次打开多少个文件是没有限制的,但是 MS - Windows 有一个限制,就是每个文件的尺寸必须限制在 28K 以下。编辑比这大的汇编文件也是可能的,但是它们不能在一个完整的编辑器中编辑。每打开一个汇编文件,都将产生一个新的编辑窗口。

单击工具条中的按钮或从菜单中选择"File→New"(Alt - F N)以创建一个新的汇编文件。单击工具条中的按钮或从菜单中选择"File→Open"(Alt - F O)以打开一个已经存在的

图 7.24

汇编文件。

二、完整的编辑器

当 WAVRASM 装入了一个文件,文本编辑器将被激活。一旦一个文件被装入汇编器的编辑窗口,插入点就出现在窗口的左上角。

三、输入和格式文本

当输入时,插入点向右移动。如果文本输入超过右边边界,文本将自动向左滚动以使插入点可见。

四、移动插入点

只要把鼠标光标移动到想放插入点的地方并按下左键,插入点就可以移动到任何地方。用键或键的组合移动插入点,见表 7.2。

表 7.2 键盘移动插入法

移动插入点	按 键	移动插入点	按 键
向文本的右边移动	右方向键	到一行文本的起点	Home 键
向文本的左边移动	左方向键	到一行文本的结尾	End 键
向文本的上边移动	上方向键	到文件的起点	Ctrl + Home 键
向文本的下边移动	下方向键	到文件的结尾	Ctrl + End 键

五、格式文本

表 7.3 中描述的键是为了得到一定文本形式的必要操作。

表 7.3 用键格式文本

操 作	按 键	操 作	按 键
插入一个空格	Spacebar	结束行	Enter
向右删除一个字符	Del	缩排一行	Tab
向左删除一个字符	Backspace	插入一个制表停止位	Tab

为了分开一行，可以把插入点移动到要断开的位置，然后按下 Enter 键。为了连接两行，可以把插入点移动到要移动行的开始位置，然后按下 Backspace 键，编辑器就会把这一行连接到前一行上。

六、滚 动

如果文本的一行要比上一次能够显示的长或者宽，这个文件可以通过滚动条来移动。

七、编辑文本

编辑菜单中包含一些功能，能够对编辑工作提供很多帮助。文本可以被删除、移动或复制到新的位置。Undo 命令可以用于取消上一次的编辑操作。文本与别的窗口或应用程序之间的文本传输可以通过剪贴板实现。当文本通 Cut 或 Copy 命令删除或复制，这些文本就放置在剪贴板中。粘贴命令把文本从剪贴板复制到编辑器中。

八、选择文本

从 Edit 菜单中选择一个命令去编辑文本之前，被操作的文本必须首先被选择。

用键盘来选择文本：

(1) 用方向键把插入点移动到要选择文本的起始部位。

(2) 按住 Shift 键，直到把插入点移动到要选择文本的结束部位，释放 Shift 键。取消这次选择，按任一个方向键。

用鼠标来选择文本：

(1) 把鼠标光标移动到要选择文本的起始部位；

(2) 按住鼠标左键，直到把插入点移动到要选择文本的结束部位，释放鼠标键。

(3) 取消这次选择，按下鼠标左键或任一个方向键。

九、替换文本

当文本被选择时，可以通过输入新的文本立即替换被选择的文本。当第一个新的字符被输入时，所选择的文本就被删除了。

要替换文本：(1) 选择要被替换的文本；(2) 输入新的文本。

要删除文本：(1) 选择要被删除的文本；(2) 按下 Del 键。

要恢复被删除的文本。在删除文本后，应立即单击工具条中的按钮或从菜单中选择“Edit→Undo”(Alt + Backspace)。

十、移动文本

要想在编辑器中移动文本，可以通过 Cut 命令把要移动的文本复制到剪贴板中，然后用 Paste 命令把它粘贴到新的位置 。

要移动文本：

(1) 选择要移动的文本；

(2) 在工具条中按下按钮或从菜单中选择“Edit→Cut”(Shift + Del)，文本就被放置在剪贴板中；

(3) 把插入点移动到新的位置；

(4) 在工具条中按下按钮或从菜单中选择“Edit→Paste”(Shift + Ins)复制文本。

如果一些文本要用到一次以上,不必每次都重新输入它。文本可以用 Copy 命令复制到剪贴板中,然后用 Paste 命令把它粘贴到其它地方。

要复制文本:

(1) 选择要复制的文本;

(2) 在工具条中按下 按钮或从菜单中选择"Edit→Copy"(Ctrl + Ins),文本就被放置在剪贴板中;

(3) 把插入点移动到要放置文本的位置;

(4) 在工具条中按下 按钮或从菜单中选择"Edit→Paste"(Shift + Ins)复制文本。

十一、取消一次编辑操作

Undo 命令可以用于取消上一次的编辑操作。例如,文本可能被意外地删除或复制到一个错误的位置。如果在错误发生后立即选择 Undo 命令,文本将被恢复到错误发生以前的状态。

为了取消上一次的编辑操作,在工具条中按下 按钮或从菜单中选择"Edit→Undo"(Alt + Backspace)。

十二、单击错误信息

汇编器有一个单击错误信息的功能。当编译完一个程序时,一个信息窗口将出现在屏幕上。如果有错误出现,这些错误将排列在信息窗口中。如果信息窗口中的一个错误信息被单击,相应的源代码行就变成了红色。如果错误信息出现在包含文件中,什么也不会发生。这个特性如图 7.25 所示。

图 7.25

如果信息窗口行被双击,包含错误信息文件的窗口将变成活动窗口,光标将放置在包含错误信息行的起始部位。如果包含错误信息的文件未被打开,这个文件将被自动打开。注意这个功能仅对编译过的文件有效。这就是说,如果源代码文件的行被增加或重新移动过,这个文件必须被重新编译,以得到正确的行数。

十三、设置程序选项

WAVRASM 的一些缺省值可以在选项菜单中被修改。如果在菜单中选择 Option,图 7.26 所示的对话框将被弹出。

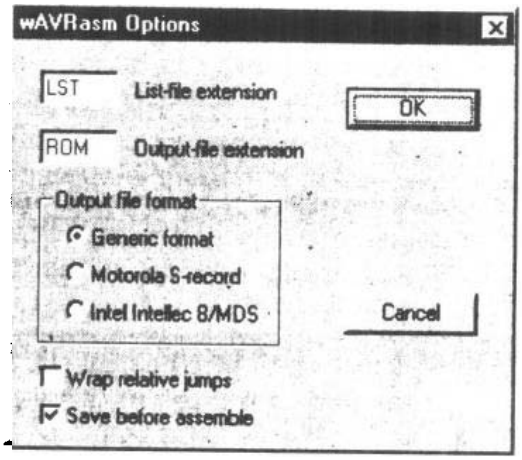


图 7.26

在标有"List - file extension"的框中,缺省的列表文件扩展名被填写。在标有"Output - file extension"的框中,缺省的输出文件扩展名"HEX"被填写。在标有"Output file format "的框中,输出文件的格式可以被选择。如果单击了 OK 按钮,这些值将在以后汇编器运行时出现。注意目标文件(用于模拟)将不被这些选项影响。目标文件的扩展名总是 OBJ,格式也是相同的。如果在源代码中定义了 EEPROM 段,汇编器将产生一个以 EEP 为扩展名的文件。这个文件用于初始化 EEPROM 存储器的值。EEPROM 初始化文件的格式与 被选择的输出文件的格式相同。

"Wrap relative jumps"选项告诉汇编器使用地址约束方式。这个特性仅用于有 4K 程序存储器的器件的编译。在那样的器件上用此选项,相应的跳转指令和调用指令将能到达所有的程序空间。

"Save before assemble"选项使汇编器在编译之前自动地保存编辑器中的内容。

十四、命令行方案

在 MS - DOS 命令行方案中,汇编器可以通过命令调用。

AVRASM [- m | - i | - g][- w] input.asm output.lst output.rom

AVRASM 将从 input.asm 中读入源代码,产生列表文件 output.lst、output.rom 和目标文件 input.obj。目标文件将被 MS - Windows 模拟器调用。

用户可以通过选项 - m(Motolora S - record)、- i(Intel Hex)、- g(Generic)中的一个选择所产生的输出文件的格式。缺省值是 Generic 文件格式。

- w 选项告诉汇编器使用地址约束方式。这个特性仅用于有 4K 程序存储器的器件的编译。在那些器件上用此选项,相应的跳转指令和调用指令将能到达所有的程序空间。

7.3 AVR 串行下载板

一、AVR 单片机串行下载板

如图 7.27 所示为 AVR 单片机串行下载板。

(1) 40 脚单片机插座可用于 AT90S4414、AT90S8515、AT89S8252,20 脚单片机插座可用于 AT90S1200、AT90S2313。

(2) 转换插针连通时用于对 AT89S8252 的操作,不连通时用于对 AVR 系列单片机的

操作。

二、系统连接

如图 7.28 所示为 AVR 单片机串行下载板的系统连接图。

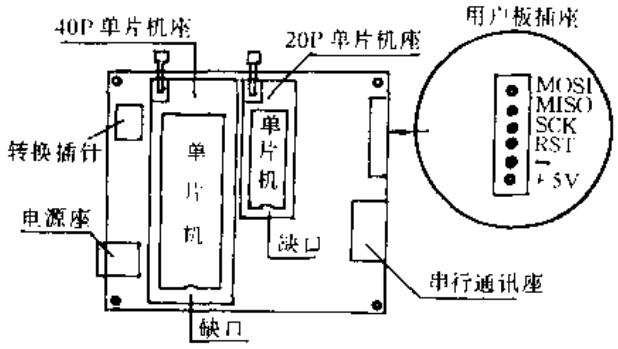


图 7.27 AVR 单片机串行下载板

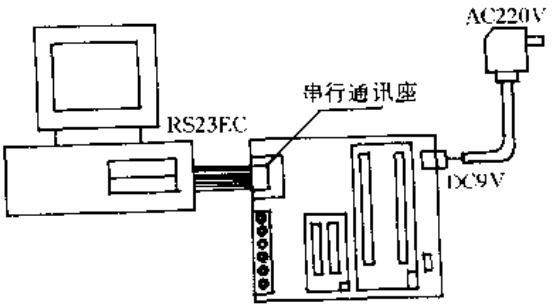


图 7.28 AVR 单片机串行下载板的系统连接

(1) 在计算机关闭电源情况下,用专用串行通讯电缆一端接到计算机 RS232C 九针插座上,另一端与串行下载板的"串行通讯插座"相接。

(2) 将 9 V 电源接到串行下载板的"电源插座"上,此时,串行下载板的红色信号灯亮。

(3) 将 AVR 单片机插入串行下载板的"20 脚单片机插座"或"40 脚单片机插座"上并锁紧。

(4) 接通微机电源开始计算机操作。

三、串行下载板与用户板的连接

如图 7.29 所示为 AVR 单片机串行下载板与用户板的连接图。

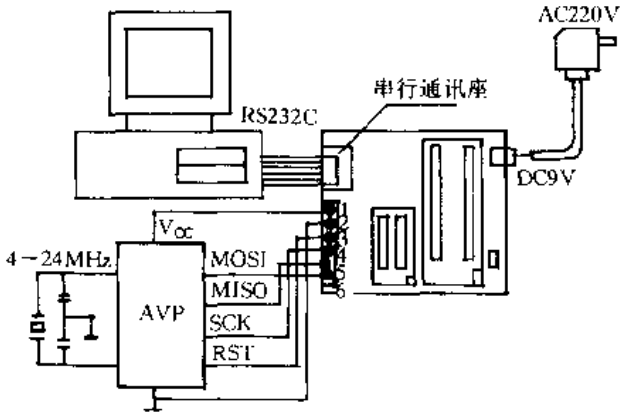


图 7.29 串行下载板与用户板的连接

(1) 按图 7.28 所示的接线顺序与用户板连接。

(2) 在计算机关闭电源情况下,用户专用串行通讯电缆一端接到计算机 RS232C 九针插座上,另一端与串行下载板的"串行通讯插座"相接。

(3) 将 9 V 电源接到串行下载板的"电源插座"上,此时,串行下载板的红色信号灯亮。

(4) 接通微机电源,开始计算机操作。

附录 A　指令集综合

AT90SXXXX Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	# Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd←Rd + Rr	Z, C, N, V, H	1
ADC	Rd, Rr	Add with Carry tow Registers	Rd←Rd + Rr + C	Z, C, N, V, H	1
ADIW	Rdl, K	Add Immediate to Word	Rdh:Rdl←Rdh:Rdl + K	Z, C, N, V, S	2
SUB	Rd, Rr	Subtract two Registers	Rd←Rd − Rr	Z, C, N, V, H	1
SUBI	Rd, K	Subtract Constant from Register	Rd←Rd − K	Z, C, N, V, H	1
SBIW	Rdl, K	Subtract Immediate from Word	Rdh:Rdl←Rdh:rdl − K	Z, C, N, V, S	2
SBC	Rd, Rr	Subtract with Carry two Registers	Rd←Rd − Rr − C	Z, C, N, V, H	1
SBCI	Rd, K	Subtract with Cary Constant from Reg.	Rd←Rd − K − C	Z, C, N, V, H	1
AND	Rd, Rr	Logical AND Registers	Rd←Rd•Rr	Z, N, V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd←Rd•K	Z, N, V	1
OR	Rd, Rr	Logical OR Registers	Rd←Rd V Rr	Z, N, V	1
ORI	Rd, K	Logical OR Register and Constant	Rd←Rd V K	Z, N, V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd←Rd⊕Rr	Z, N, V	1
COM	Rd	One's Complement	Rd←$FF − Rd	Z, C, N, V	1
NEG	Rd	Two's Complement	Rd←$00 − Rd	Z, C, N, V, H	1
SBR	Rd, K	Set Bit(s) in Register	Rd←Rd V K	Z, N, V	1
CBR	Rd, K	Clear Bit(s) in Register	Rd←Rd•($FF − K)	Z, N, V	1
INC	Rd	Increment	Rd←Rd + 1	Z, N, V	1
DEC	Rd	Decrement	Rd←Rd − 1	Z, N, V	1
TST	Rd	Test for Zero or Minus	Rd←Rd•Rd	Z, N, V	1
CLR	Rd	Clear Register	Rd←Rd⊕Rd	Z, N, V	1
SER	Rd	Set Register	Rd←$FF	None	1
BRANCH INTRUCTIONS					
RJMP	k	Relative Jump	PC←PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC←Z	None	2
RCALL	k	Relative Subroutine Call	PC←PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC←Z	None	3
RET		Subroutine Return	PC←STACK	None	4
RETI		Interrupt Return	PC←STACK	I	4
CPSE	Rd, Rr	Compare, Skip if Equal	if (Rd = Rr) PC←PC + 2 or 3	None	1/2
CP	Rd, Rr	Compare	Rd − Rr	Z, N, V, C, H	1
CPC	Rd, Rr	Compare with Carry	Rd − Rr − C	Z, N, V, C, H	1
CPI	Rd, K	Compare Register with Immediate	Rd − K	Z, N, V, C, H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if(Rr(b) = 0) PC←PC + 2 or 3	None	1/2
SBRS	Rr, b	Skip if Bit in Register is Set	if(Rr(b) = 1) PC←PC + 2 or 3	None	1/2
SBIC	P, b	Skip if Bit in I/O Register Cleared	if(P(b) = 0) PC←PC + 2 or 3	None	1/2

附录 A 续

Mnemonics	Operands	Description	Operation	Flags	# Clocks
SBIS	P, b	Skip if Bit in I/O Register is Set	if(R(b) = 1) PC←PC + 2 or 3	None	1/2
BRBS	s, k	Branch if Status Flag Set	if (SREG(s)) = 1 then PC←PC + k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s)) = 0 then PC←PC + k + 1	None	1/2
BREQ	k	Branch if Equal	if(Z = 1) then PC←PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if(Z = 0) then PC←PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if(C = 1) then PC←PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if(C = 0) then PC←PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if(C = 0) then PC←PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if(C = 1) then PC←PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if(N = 1) then PC←PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if(N = 0) then PC←PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal Signed	if(N + V = 0) then PC←PC + k + 1	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	if(N + V = 1) then PC←PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if(H = 1) then PC←PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if(H = 0) then PC←PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if(T = 1) then PC←PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if(T = 0) then PC←PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if(V = 1) then PC←PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if(V = 0) then PC←PC + k + 1	None	1/2
BRIE	k	Branch if Interrupt Enabled	if(I = 1) then PC←PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if(I = 0) then PC←PC + k + 1	None	1/2
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd←Rr	None	1
LDI	Rd, K	Load Immediate	Rd←K	None	1
LD	Rd, X	Load Indirect	Rd←(X)	None	2
LD	Rd, X +	Load Indirect and Pstt - Increment	Rd←(X)←X + 1	None	2
LD	Rd, - X	Load Indirect and Pre - Decrement	X←X - 1, Rd←(X)	None	2
LD	Rd, Y	Load Indirect	Rd←(Y)	None	2
LD	Rd, Y +	Load Indirect and Post - Inc	Rd←(Y), Y←Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre - Dec	Y←Y - 1, Rd←(Y)	None	2
LDD	Rd, Y + q	Load Indirect with Displacement	Rd←(Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd←(Z)	None	2
LD	Rd, Z +	Load Indirect and Post - Inc	Rd←(Z), Z←Z + 1	None	2
LD	Rd, - Z	Load Indirect and Pre - Dec	Z←Z - 1, Rd←(Z)	None	2
LDD	Rd, Z + q	Load Indirect with Displacement	Rd←(Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd←(k)	None	3
ST	X, Rd	Store Indirect	(X)←Rr	None	2
ST	X + , Rd	Store Indirect and Post - Inc	(X)←Rr, X←X + 1	None	2
ST	- X, Rd	Store Indirect and Pre - Dec	X←X - 1, (X)←Rr	None	2
ST	Y, Rd	Store Indirect	(Y)←Rr	None	2
ST	Y + , Rd	Store Indirect and Pos: - Inc	(Y)←Rr, Y←Y + 1	None	2
ST	- Y, Rd	Store Indirect and Pre - Dec	Y←Y - 1, (Y)←Rr	None	2
STD	Y + q, Rd	Store Indirect with Displacement	(Y + q)←Rr	None	2
ST	Z, Rr	Store Indirect	(Z)←Rr	None	2

附录 A 续

Mnemonics	Operands	Description	Operation	Flags	# Clocks
ST	Z+, Rr	Store Indirect and Post-Inc	(Z)← Rr, Z←Y+1	None	2
ST	Z, Rr	Store Indirect and Pre-Dec	Z←Z-1, (Z)←Rr	None	2
STD	Z+q, Rr	Store Indirect with Displacement	(Z+q)← Rr	None	2
STS	k, Rr	Store Direct SRAM	(k)←Rr	None	3
LPM		Load Program Memory	R0← (Z)	None	3
IN	Rr, p	In Port	Rd← P	None	1
OUT	p, Rd	Out Port	P←Rr	None	1
PUSH	Rr	Push Register on Stack	STACK←Rr	None	2
POP	Rd	Pop Registerfrom Stack	Rd← STACK	None	2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P, b	Set Bit in I/O Register	I/O(P, b)←1	None	2
CBI	P, b	Clear Bit in I/O Register	I/O(P, b)←0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1)←Rd(n) Rd(0)←0	Z, C, N, V	1
LSR	Rd	Logical Shift Right	Rd(n)←Rd(n+1), Rd(7)←0	Z, C, N, V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C, Rd(n+1)← Rd(n)←, C← Rd(7)	Z, C, N, V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C, Rd(n)←Rd(n+1), C←Rd (0)	Z, C, N, V	1
ASR	Rd	Arithmetic Shift Right	Rd(n)←Rd(n+1)n=0~6	Z, C, N, V	1
SWAP	Rd	Swap Nibbles	Rd(3~0)←Rd(7~4)←Rd(3~0)	None	1
BSET	s	Flag Set	SREG(s)←1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s)←0	SREG(s)	1
BST	Rr, b	Bit Store Register to T	T←Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b)← T	None	1
SEC		Set Carry	C← 1	C	1
CLC		Clear Carry	C←0	C	1
SEN		Set Negative Flag	N←0	N	1
CLN		Clear Negative Flag	N←0	N	1
SEZ		Set Zero Flag	Z← 1	Z	1
CLZ		Cleat Zero Flag	Z←0	Z	1
SEI		Global Interrupt Enable	I← 1	I	1
CLI		Global Interrupt Disable	I←0	I	1
SES		Set signed Test Flag	S←1	S	1
CLS		Clearigned Test Flag	S←0	S	1
SEV		Set Twos Compement Overflow	V←1	V	1
CLV		Clear Twos Compement Overllow	V← 0	V	1
SET		Set T in SREG	T← 1	T	1
CLT		Clear T in SREG	T← 0	T	1
SEH		Set Half Carry Flag in SREG	H←1	H	1
CLH		Clear Half Carry Flag in SREG	H← 0	H	1
NOP		No Operation		None	1
SLEEP		Sleep	(see speciffic descr. for Sleep function)	None	3
WDR		Watchdog Reset	(see speciffic descr. for WDR/timer)	None	1

附录 B　寄存器综合

AT90S1200 Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$3F	SREG	I	T	H	S	V	N	Z	C
$3E	Reserved								
$3D	Reserved								
$3C	Reserved								
$3B	GIMSK	-	INT0	-	-	-	-	-	-
$3A	Reserved								
$39	TIMSK	-	-	-	-	-	-	TOIE0	-
$38	TIFR	-	-	-	-	-	-	TOV0	-
$37	Reserved								
$36	Reserved								
$35	MCUCR	-	-	SE	SM	-	-	ISC01	ISC00
$34	Reserved								
$33	TCCR0	-	-	-	-	-	CS02	CS01	CS00
$32	TCNT0	Timer/Counter0 (8 Bit)							
$31	Reserved								
$30	Reserved								
$2F	Reserved								
$2E	Reserved								
$2D	Reserved								
$2C	Reserved								
$2B	Reserved								
$2A	Reserved								
$29	Reserved								
$28	Reserved								
$27	Reserved								
$26	Reserved								
$25	Reserved								
$24	Reserved								
$23	Reserved								
$22	Reserved								
$21	WDTCR	-	-	-	-	WDE	WDP2	WDP1	WDP0
$20	Reserved								
$1F	Reserved								
$1E	EEAR	-	EEPROM Address Register						
$1D	EEDR	EEPROM Data Register							
$1C	EECR	-	-	-	-	-	-	EEWE	EERE
$1B	Reserved								
$1A	Reserved								
$19	Reserved								
$18	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
$17	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
$16	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
$15	Reserved								
$14	Reserved								
$13	Reserved								
$12	PORTD	-	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
$11	DDRD	-	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
$10	PIND	-	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
$0F	Reserved								
$0E	Reserved								
$0D	Reserved								
$0C	Reserved								
$0B	Reserved								
$0A	Reserved								
$09	Reserved								
$08	ACSR	ACD	-	ACO	AC	ACIE	-	ACIS1	ACIS0
...	Reserved								
$00	Reserved								

AT90S2313 Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C
$3E ($5E)	Reserved								
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
$3C ($5C)	Reserved								
$3B ($5B)	GIMSK	INT1	INT0						
$3A ($5A)	GIFR	INTF1	INTF0						
$39 ($59)	TIMSK	TOIE1	OCIE1A			TICIE1		TOIE0	
$38 ($58)	TIFR	TOV1	OCF1A			ICF1		TOV0	
$37 ($57)	Reserved								
$36 ($56)	Reserved								
$35 ($55)	MCUCR			SE	SM	ISC11	ISC10	ISC01	ISC00
$34 ($54)	Reserved								
$33 ($53)	TCCR0						CS02	CS01	CS00
$32 ($52)	TCNT0	Timer/Counter0 (8 Bit)							
$31 ($51)	Reserved								
$30 ($50)	Reserved								
$2F ($4F)	TCCR1A	COM1A1	COM1A0					PWM11	PWM10
$2E ($4E)	TCCR1B	ICNC1	ICES1			CTC1	CS12	CS11	CS10
$2D ($4D)	TCNT1H	Timer/Counter1 - Counter Register High Byte							
$2C ($4C)	TCNT1L	Timer/Counter1 - Counter Register Low Byte							
$2B ($4B)	OCR1AH	Timer/Counter1 - Compare Register High Byte							
$2A ($4A)	OCR1AL	Timer/Counter1 - Compare Register Low Byte							
$29 ($49)	Reserved								
$28 ($48)	Reserved								
$27 ($47)	Reserved								
$26 ($46)	Reserved								
$25 ($45)	ICR1H	Timer/Counter1 - Input Capture Register High Byte							
$24 ($44)	ICR1L	Timer/Counter1 - Input Capture Register Low Byte							
$23 ($43)	Reserved								
$22 ($42)	Reserved								
$21 ($41)	WDTCR				WDTOE	WDE	WDP2	WDP1	WDP0
$20 ($40)	Reserved								
$1F ($3F)	Reserved								
$1E ($3E)	EEAR	EEPROM Address Register							
$1D ($3D)	EEDR	EEPROM Data register							
$1C ($3C)	EECR						EEMWE	EEWE	EERE
$1B ($3B)	Reserved								
$1A ($3A)	Reserved								
$19 ($39)	Reserved								
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
$15 ($35)	Reserved								
$14 ($34)	Reserved								
$13 ($33)	Reserved								
$12 ($32)	PORTD		PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
$11 ($31)	DDRD		DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
$10 ($30)	PIND		PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
$0F ($2F)	Reserved								
$0E ($2E)	Reserved								
$0D ($2D)	Reserved								
$0C ($2C)	UDR	UART I/O Data Register							
$0B ($2B)	USR	RXC	TXC	UDRE	FE	OR			
$0A ($2A)	UCR	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8
$09 ($29)	UBRR	UART Baud Rate Register							
$08 ($28)	ACSR	ACD		ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
...	Reserved								
$00 ($20)	Reserved								

AT90S4414 Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C
$3E ($5E)	SPH	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
$3C ($5C)	Reserved								
$3B ($5B)	GIMSK	INT1	INT0						
$3A ($5A)	GIFR	INTF1	INTF0						
$39 ($59)	TIMSK	TOIE1	OCIE1B	OCIE1A		TICIE1		TOIE0	
$38 ($58)	TIFR	TOV1	OCF1B	OCF1A		ICF1		TOV0	
$37 ($57)	Reserved								
$36 ($56)	Reserved								
$35 ($55)	MCUCR	SRE	SRW	SE	SM	ISC11	ISC10	ISC01	ISC00
$34 ($54)	Reserved								
$33 ($53)	TCCR0						CS02	CS01	CS00
$32 ($52)	TCNT0	Timer/Counter0 (8 Bit)							
$31 ($51)	Reserved								
$30 ($50)	Reserved								
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0			PWM11	PWM10
$2E ($4E)	TCCR1B	ICNC1	ICES1			CTC1	CS12	CS11	CS10
$2D ($4D)	TCNT1H	Timer/Counter1 - Counter Register High Byte							
$2C ($4C)	TCNT1L	Timer/Counter1 - Counter Register Low Byte							
$2B ($4B)	OCR1AH	Timer/Counter1 - Output Compare Register A High Byte							
$2A ($4A)	OCR1AL	Timer/Counter1 - Output Compare Register A Low Byte							
$29 ($49)	OCR1BH	Timer/Counter1 - Output Compare Register B High Byte							
$28 ($48)	OCR1BL	Timer/Counter1 - Output Compare Register B Low Byte							
$27 ($47)	Reserved								
$26 ($46)	Reserved								
$25 ($45)	ICR1H	Timer/Counter1 - Input Capture Register High Byte							
$24 ($44)	ICR1L	Timer/Counter1 - Input Capture Register Low Byte							
$23 ($43)	Reserved								
$22 ($42)	Reserved								
$21 ($41)	WDTCR				WDTCE	WDE	WDP2	WDP1	WDP0
$20 ($40)	Reserved								
$1F ($3F)	EEARH								EEAR9
$1E ($3E)	EEARL	EEPROM Address Register Low							
$1D ($3D)	EEDR	EEPROM Data Register							
$1C ($3C)	EECR						EEMWE	EEWE	EERE
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
$0F ($2F)	SPDR	SPI Data Register							
$0E ($2E)	SPSR	SPIF	WCOL						
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
$0C ($2C)	UDR	UART I/O Data Register							
$0B ($2B)	USR	RXC	TXC	UDRE	FE	OR			
$0A ($2A)	UCR	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8
$09 ($29)	UBRR	UART Baud Rate Register							
$08 ($28)	ACSR	ACD		ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
...	Reserved								
$00 ($20)	Reserved								

AT90S2323 Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C
$3E ($5E)	Reserved								
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
$3C ($5C)	Reserved								
$3B ($5B)	GIMSK		INT0						
$3A ($5A)	GIFR		INTF0						
$39 ($59)	TIMSK							TOIE0	
$38 ($58)	TIFR							TOV0	
$37 ($57)	Reserved								
$36 ($56)	Reserved								
$35 ($55)	MCUCR			SE	SM			ISC01	ISC00
$34 ($54)	MCUSR							EXTRF	PORF
$33 ($53)	TCCR0						CS02	CS01	CS00
$32 ($52)	TCNT0	Timer/Counter0 (8 Bit)							
$31 ($51)	Reserved								
$30 ($50)	Reserved								
$2F ($4F)	Reserved								
$2E ($4E)	Reserved								
$2D ($4D)	Reserved								
$2C ($4C)	Reserved								
$2B ($4B)	Reserved								
$2A ($4A)	Reserved								
$29 ($49)	Reserved								
$28 ($48)	Reserved								
$27 ($47)	Reserved								
$26 ($46)	Reserved								
$25 ($45)	Reserved								
$24 ($44)	Reserved								
$23 ($43)	Reserved								
$22 ($42)	Reserved								
$21 ($41)	WDTCR				WDTOE	WDE	WDP2	WDP1	WDP0
$20 ($40)	Reserved								
$1F ($3F)	Reserved								
$1E ($3E)	EEAR		EEPROM Address Register						
$1D ($3D)	EEDR	EEPROM Data register							
$1C ($3C)	EECR						EEMWE	EEWE	EERE
$1B ($3B)	Reserved								
$1A ($3A)	Reserved								
$19 ($39)	Reserved								
$18 ($38)	PORTB				PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
$17 ($37)	DDRB				DDB4	DDB3	DDB2	DDB1	DDB0
$16 ($36)	PINB				PINB4	PINB3	PINB2	PINB1	PINB0
$15 ($35)	Reserved								
$14 ($34)	Reserved								
$13 ($33)	Reserved								
$12 ($32)	Reserved								
$11 ($31)	Reserved								
$10 ($30)	Reserved								
$0F ($2F)	Reserved								
$0E ($2E)	Reserved								
$0D ($2D)	Reserved								
$0C ($2C)	Reserved								
$0B ($2B)	Reserved								
$0A ($2A)	Reserved								
$09 ($29)	Reserved								
$08 ($28)	Reserved								
...	Reserved								
$00 ($20)	Reserved								

AT90S8515 Register Summary

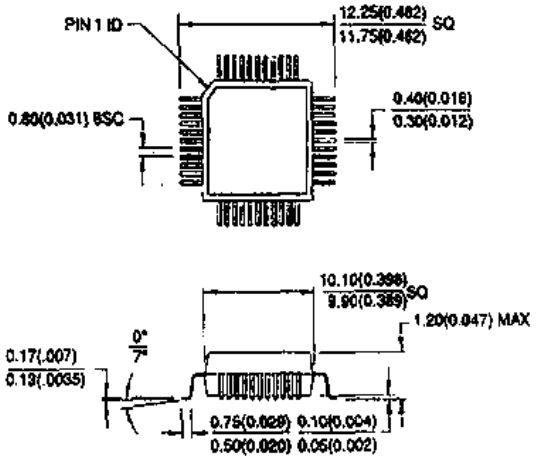
Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C
$3E ($5E)	SPH	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
$3C ($5C)	Reserved								
$3B ($5B)	GIMSK	INT1	INT0						
$3A ($5A)	GIFR	INTF1	INTF0						
$39 ($59)	TIMSK	TOIE1	OCIE1A	OCIE1B		TICIE1		TOIE0	
$38 ($58)	TIFR	TOV1	OCF1A	OCF1B		ICF1		TOV0	
$37 ($57)	Reserved								
$36 ($56)	Reserved								
$35 ($55)	MCUCR	SRE	SRW	SE	SM	ISC11	ISC10	ISC01	ISC00
$34 ($54)	Reserved								
$33 ($53)	TCCR0						CS02	CS01	CS00
$32 ($52)	TCNT0	Timer/Counter0 (8 Bit)							
$31 ($51)	Reserved								
$30 ($50)	Reserved								
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0			PWM11	PWM10
$2E ($4E)	TCCR1B	ICNC1	ICES1			CTC1	CS12	CS11	CS10
$2D ($4D)	TCNT1H	Timer/Counter1 - Counter Register High Byte							
$2C ($4C)	TCNT1L	Timer/Counter1 - Counter Register Low Byte							
$2B ($4B)	OCR1AH	Timer/Counter1 - Output Compare Register A High Byte							
$2A ($4A)	OCR1AL	Timer/Counter1 - Output Compare Register A Low Byte							
$29 ($49)	OCR1BH	Timer/Counter1 - Output Compare Register B High Byte							
$28 ($48)	OCR1BL	Timer/Counter1 - Output Compare Register B Low Byte							
$27 ($47)	Reserved								
$26 ($46)	Reserved								
$25 ($45)	ICR1H	Timer/Counter1 - Input Capture Register High Byte							
$24 ($44)	ICR1L	Timer/Counter1 - Input Capture Register Low Byte							
$23 ($43)	Reserved								
$22 ($42)	Reserved								
$21 ($41)	WDTCR				WDTOE	WDE	WDP2	WDP1	WDP0
$20 ($40)	Reserved								
$1F ($3F)	Reserved								
$1E ($3E)	EEARL	EEPROM Address Register							
$1D ($3D)	EEDR	EEPROM Data Register							
$1C ($3C)	EECR						EEMWE	EEWE	EERE
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
$0F ($2F)	SPDR	SPI Data Register							
$0E ($2E)	SPSR	SPIF	WCOL						
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
$0C ($2C)	UDR	UART I/O Data Register							
$0B ($2B)	USR	RXC	TXC	UDRE	FE	OR			
$0A ($2A)	UCR	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8
$09 ($29)	UBRR	UART Baud Rate Register							
$08 ($28)	ACSR	ACD		ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
...	Reserved								
$00 ($20)	Reserved								

ATmega103 Register Summary

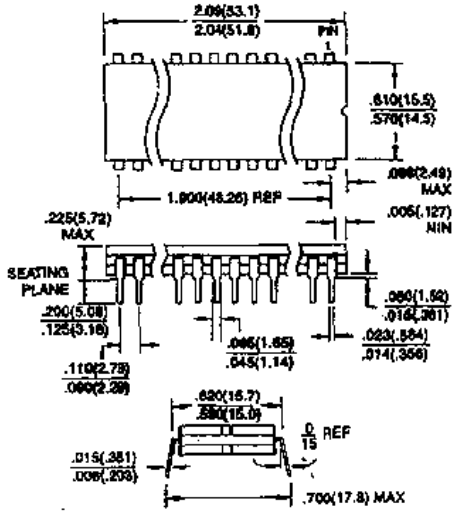
Address	Name	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C
$3E ($5E)	SPH	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
$3C ($5C)	XDIV	XDIVEN	XDIV6	XDIV5	XDIV4	XDIV3	XDIV2	XDIV1	XDIV0
$3B ($5B)	RAMPZ	-	-	-	-	-	-	-	RAMPZ0
$3A ($5A)	EICR	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40
$39 ($59)	EIMSK	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0
$38 ($58)	EIFR	INTF7	INTF6	INTF5	INTF4	-	-	-	-
$37 ($57)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
$36 ($56)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
$35 ($55)	MCUCR	SRE	SRW	SE	SM1	SM0	-	-	-
$34 ($54)	MCUSR	-	-	-	-	-	-	PORF	EXTRF
$33 ($53)	TCCR0	-	PWM0	COM01	COM00	CTC0	CS02	CS01	CS00
$32 ($52)	TCNT0	Timer/Counter0 (8 Bit)							
$31 ($51)	OCR0	Timer/Counter0 Output Compare Register							
$30 ($50)	ASSR	-	-	-	-	AS0	TCN0UB	OCR0UB	TCR0UB
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	-	-	PWM11	PWM10
$2E ($4E)	TCCR1B	ICNC1	ICES1	-	-	CTC1	CS12	CS11	CS10
$2D ($4D)	TCNT1H	Timer/Counter1 - Counter Register High Byte							
$2C ($4C)	TCNT1L	Timer/Counter1 - Counter Register Low Byte							
$2B ($4B)	OCR1AH	Timer/Counter1 - Output Compare Register A High Byte							
$2A ($4A)	OCR1AL	Timer/Counter1 - Output Compare Register A Low Byte							
$29 ($49)	OCR1BH	Timer/Counter1 - Output Compare Register B High Byte							
$28 ($48)	OCR1BL	Timer/Counter1 - Output Compare Register B Low Byte							
$27 ($47)	ICR1H	Timer/Counter1 - Input Capture Register High Byte							
$26 ($46)	ICR1L	Timer/Counter1 - Input Capture Register Low Byte							
$25 ($45)	TCCR2	-	PWM2	COM21	COM20	CTC2	CS22	CS21	CS20
$24 ($44)	TCNT2	Timer/Counter2 (8 Bit)							
$23 ($43)	OCR2	Timer/Counter2 Output Compare Register							
$21 ($42)	WDTCR	-	-	-	WDTOE	WDE	WDP2	WDP1	WDP0
$1F ($3F)	EEARH	-	-	-	-	EEAR11	EEAR10	EEAR9	EEAR8
$1E ($3E)	EEARL	EEPROM Address Register L							
$1D ($3D)	EEDR	EEPROM Data Register							
$1C ($3C)	EECR	-	-	-	-	EERIE	EEMWE	EEWE	EERE
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
$0F ($2F)	SPDR	SPI Data Register							
$0E ($2E)	SPSR	SPIF	WCOL	-	-	-	-	-	-
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
$0C ($2C)	UDR	UART I/O Data Register							
$0B ($2B)	USR	RXC	TXC	UDRE	FE	OR	-	-	-
$0A ($2A)	UCR	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8
$09 ($29)	UBRR	UART Baud Rate Register							
$08 ($28)	ACSR	ACD	-	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
$07 ($27)	ADMUX	-	-	-	-	-	MUX2	MUX1	MUX0
$06 ($26)	ADCSR	ADEN	ADSY	ADRF	ADIF	ADIE	ADPS2	ADPS1	ADPS0
$05 ($25)	ADCH	-	-	-	-	-	-	ADC9	ADC8
$04 ($24)	ADCL	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
$03 ($23)	PORTE	PORTE7	PORTE6	PORTE5	PORTE4	PORTE3	PORTE2	PORTE1	PORTE0
$02 ($22)	DDRE	DDE7	DDE6	DDE5	DDE4	DDE3	DDE2	DDE1	DDE0
$01 ($21)	PINE	PINE7	PINE6	PINE5	PINE4	PINE3	PINE2	PINE1	PINE0
$00 ($20)	PINF	PINF7	PINF6	PINF5	PINF4	PINF3	PINF2	PINF1	PINF0

附录C 包　　装

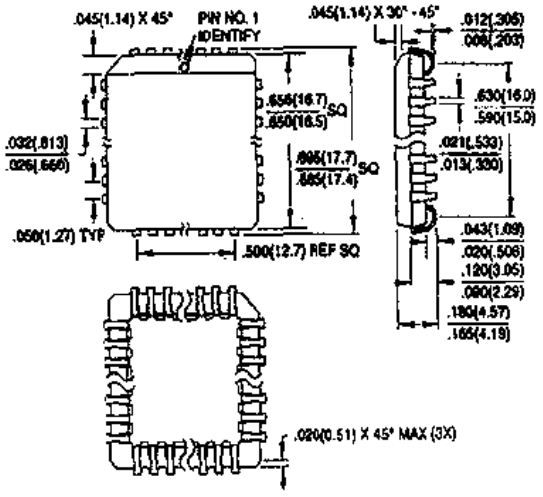
44A, 44 Lead, Thin (1.0 mm)
Plastic Gull Wing Quad Flat Package (TQFP)
Dimensions in Millimeters and (Inches)



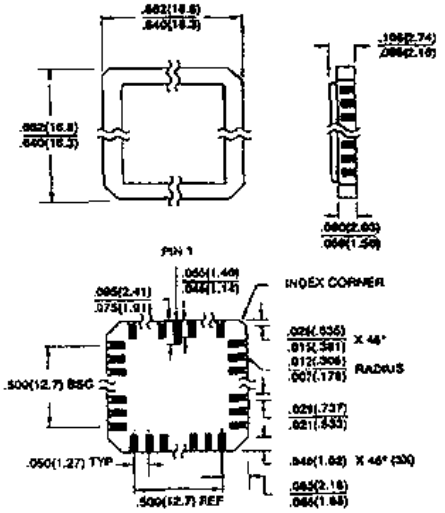
40D6, 40 Lead, 0.600" Wide, Non-Windowed,
Ceramic Dual Inline Package (Cerdip)
Dimensions in Inches and (Millimeters)
MIL-STD-1835 D-5 CONFIG A



44J, 44 Lead, Plastic J-Leaded Chip Carrier (PLCC)
Dimensions in Inches and (Millimeters)
JEDEC STANDARD MS-018 AC

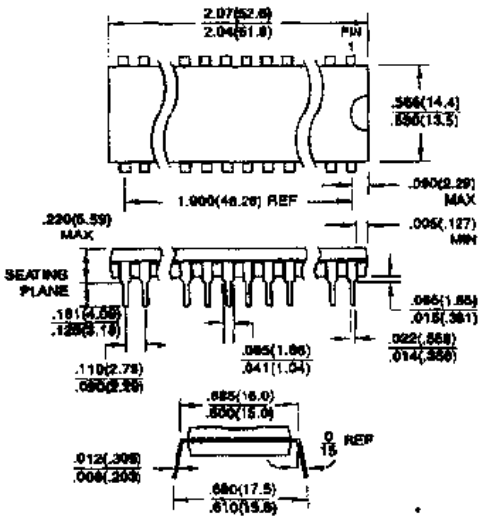


44L, 44 Lead, Non-Windowed,
Ceramic Leadless Chip Carrier (LCC)
Dimensions in Inches and (Millimeters)*
MIL-STD-1835 C-5

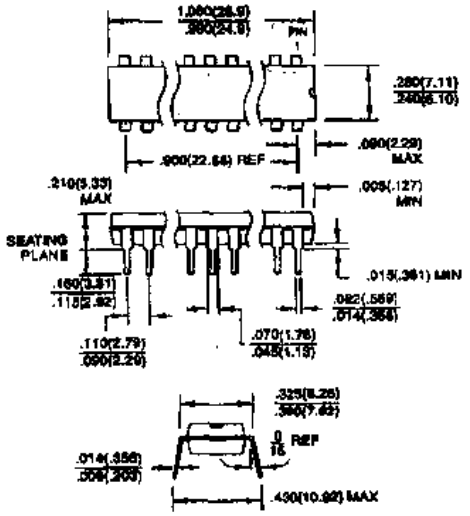


*Ceramic lid standard unless specified.

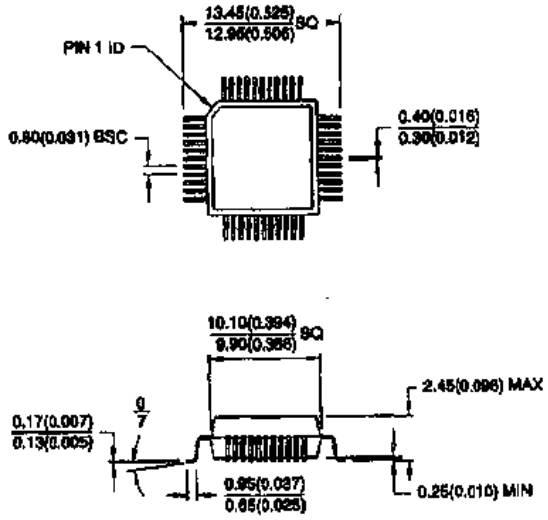
40P6, 40 Lead, 0.600" Wide,
Plastic Dual Inline Package (PDIP)
Dimensions in Inches and (Millimeters)
JEDEC STANDARD MS-011 AC



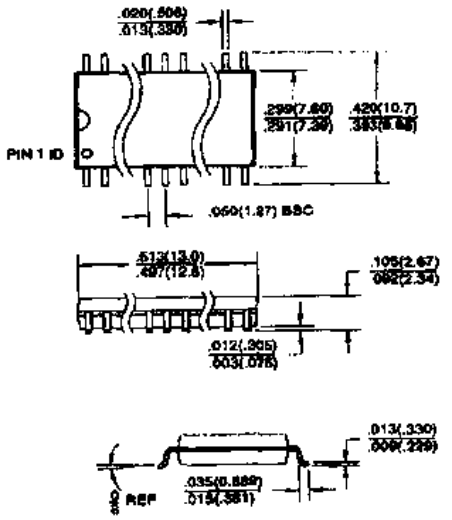
20P3, 20 Lead, 0.300" Wide,
Plastic Dual Inline Package (PDIP)
Dimensions in Inches and (Millimeters)
JEDEC STANDARD MS-011 AD



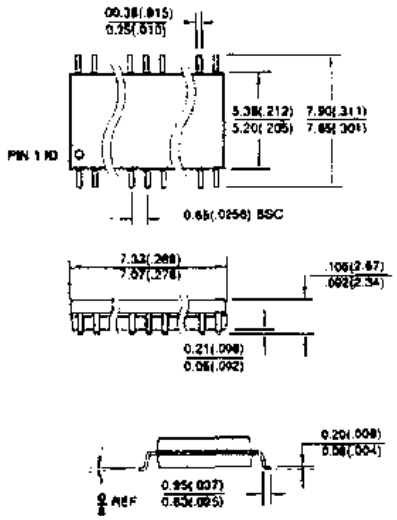
44Q, 44 Lead,
Plastic Gull Wing QuadFlat Package (PQFP)
Dimensions in Inches and (Millimeters)



20S, 20 Lead, 0.300" Wide,
Plastic Gull Wing Small Outline (SOIC)
Dimensions in Inches and (Millimeters)



20Y, 20 Lead, 5.3 mm Wide,
Plastic Shrink Small Outline (SSOP)
Dimensions in Millimeters and (Inches)



参 考 文 献

[1] AVR ENHANCED RISC MICROCONTROLLER DATA BOOK
[2] ATMEL DEVELOPMENT TOOL USER'S GUIDE
[3] 张友德等. 单片微型机原理、应用与实验. 上海：复旦大学出版社，1992

ATMEL 公司的产品目录

(1998 年 4 月)

表 1　一次性可编程存储器

产品型号	构　成	速　度	特　性	供　货
Battery - Voltage™(2.7~3.6V)　电池电压型				
AT27BV256	32K×8	70~150 ns	256Kb, 2.7~3.6V EPROM	Now
AT27BV512	64K×8	70~150 ns	512Kb, 2.7~3.6V EPROM	Now
AT27BV010	128K×8	90~150 ns	1Mb, 2.7~3.6V EPROM	Now
AT27BV1024	64K×16	90~150 ns	1Mb, 2.7~3.6V EPROM	Now
AT27BV020	256K×8	120 ns	2Mb, 2.7~3.6V EPROM	Now
AT27BV040	512K×8	120 ns	4Mb, 2.7~3.6V EPROM	Now
AT27BV4096	256K×16	120 ns	4Mb, 2.7~3.6V EPROM	Now
AT27BV400	512K×8/256K×16	150 ns	4Mb, 2.7~3 6V EPROM	May 98
AT27BV800	1024K×8/512K×16	150 ns	8Mb, 2.7~3.6V EPROM	May 98
Low-Voltage(3.0~3.6V)　低电压型				
AT27LV256A	32K×8	70~150 ns	256Kb 3V EPROM	Now
AT27LV512A	64K×8	70~150 ns	512Kb 3V EPROM	Now
AT27LV010A	128K×8	70~150 ns	1Mb 3V EPROM	Now
AT27LV020A	256K×8	90~150 ns	2Mb 3V EPROM	Now
AT27LV040A	512K×8	90~150 ns	4Mb 3V EPROM	Now
Standard Voltage (5.0V)　标准低电压型				
AT27C256R	32K×8	45~150 ns	256K, 5V EPROM	Now
AT27C512R	64K×8	45~150 ns	512K, 5V EPROM	Now
AT27C516	32K×16	45~100 ns	512K, 5V EPROM	Now
AT27C520	64K×8(ALE)	70, 90 ns	512K, Latched 5V EPROM	Now
AT27C010, L	128K×8	45~150 ns	1Mb 5V EPROM Standard & Low-Power	Now
AT27C1024	64K×16	45~150 ns	1Mb, 5V EPROM	Now
AT27C020	256K×8	55~150 ns	2Mb, 5V EPROM	Now
AT27C2048	128K×16	55~150 ns	2Mb, 5V EPROM	Now
AT27C040	512K×8	70~150 ns	4Mb, 5V EPROM	Now
AT27C4096	256K×16	55~150 ns	4Mb, 5V EPROM	Now
AT27C400	512K×8/256K×16	70~150 ns	4Mb, Byte - Selectable 5V EPROM	May 98
AT27C080	1024K×8	90~150 ns	8Mb, 5V EPROM	Now
AT27C800	1024K×8/512K×16	100~150 ns	8Mb, Byte-Selectable 5V EPROM	May 98

表 2 并行闪速擦除存储器

产品型号	构 成	速 度	特 性	供货
Lithium Battery - Voltage (2.5～3.0V) 锂电池电压型				
AT49LBV010	128K×8	120～200 ns	1M - bit, 2.5 - Volt Read and 2.5 - Volt Byte with Flash	Now
AT49LBV020	256K×8	120～200 ns	2M - bit, 2.5 - Volt Read and 2.5 - Volt Byte - with Flash	Now
AT49LBV040	512K×8	120 - 200 ns	4M - bit, 2.5 - Volt Read and 2.5 - Volt Byte with Flash	Now
AT49LBV080	1M×8	120～200 ns	8M - bit, 2.5 - Volt Read and 2.5 - Volt Byte with Flash	Now
AT49LBV080T	1M×8	120 - 200 ns	8M - bit, 2.5 - Volt Read and 2.5 - Volt Byte with Flash (Top Boot)	Now
AT49LBV008	1M×8	120 - 200 ns	8M - bit, 2.5 - Volt Read and 2.5 - Volt Byte with Flash	Now
AT49LBV008T	1M×8	120～200 ns	8M bit, 2.5 - Volt Read and 2.5 - Volt Byte - with Flash (Top Boot)	Now
Battery Voltage (2.7V～3.6V) 电池电压型				
AT29BV010A	128K×8	100 - 250 ns	1M - bit, 2.7 - Volt Read and 2.7 Volt Write, Sectored Flash	Now
AT29BV020	256K×8	250 ns	2M - bit, 2.7 - Volt Read and 2.7 - Volt Write, Sectored Flash	Now
AT49BV040A	512K×8	250 ns	4M - bit, 2.7 Volt Read and 2.7 - Volt Write, Sectored Flash	Now
AT49BV512	64K×8	120 - 150 ns	512K, 2.7 - Volt Read and 2.7 Volt Byte Write Flash	Now
AT49BV010	128K×8	120 - 150 ns	1M - bit, 2.7 - Volt Read and 2.7 Volt Byte - Write Flash	Now
AT49HBV010	128K×8	90 - 150 ns	1M bit, 2.7 - Volt Read and 2.7 Volt Byte - Write Flash (High - Speed)	Now
AT49BV001T	128K×8	70～120 ns	1M bit, 2.7 Volt Read and 2.7 - Volt Byte Write Sectored Flash (Top Boot)	Now
AT49BV020	256K×8	90 - 150 ns	2M bit, 2.7 - Volt Read and 2.7 Volt Byte - Write Flash	Now
AT49BV002T	256K×8	70 - 120 ns	2M - bit, 2.7 - Volt Read and 2.7 Volt Byte - Write Sectored Flash (Top Boot)	Now
AT49BV2048	128K×16	120～150 ns	2M - bit, 2.7 - Volt Read and 2.7 Volt Byte - Write Flash	Now
AT49BV040	512K×8	120～150 ns	4M bit, 2.7 Volt Read and 2.7 - Volt Byte - Write Flash	Now
AT49BV4096	256K×16	120 - 150 ns	4M - bit, 2.7 - Volt Read and 2.7 Volt Byte Write Flash	Now
AT49BV080	1024K×8	120 - 150 ns	8M bit, 2.7 Volt Read and 2.7 - Volt Byte - Write Flash	Now
AT29BV080T	1024K×8	120 - 150 ns	8M bit, 2.7 Volt Read and 2.7 - Volt Byte - Write Flash (Top Boot)	Now
AT49BV008	1024K×8	120～150 ns	8M - bit, 2.7 - Volt Read and 2.7 - Volt Byte Write Flash	Now
AT49BV8192	512K×16	120～150 ns	8M - bit, 2.7 - Volt Read and 2.7 - Volt Byte - Write Flash	Now
AT49BV8192T	512K×16	120 - 150 ns	8M - bit, 2.7 - Volt Read and 2.7 - Volt Byte Write Flash (Top Boot)	Now
AT49BV1604	1024K×16	120 - 150 ns	16M - bit, 2.7 - Volt Read and 2.7 Vot Byte - Write Sectored Flash	Now
Low - Voltage (3.0V～3.6V) 低电压型				
AT29LV256	32K×8	150～250 ns	256K, 3 Volt Read and 3 - Volt Write, Sectored Flash	Now
AT29LV512	64K×8	150 - 250 ns	512K, 3 Volt Read and 3 - Volt Write, Sectored Flash	Now
AT29LV010A	128K×8	150～250 ns	1M - bit, 3 - Volt Read and 3 Volt Write, Sectored Flash	Now
AT29LV1024	64K×8	150 - 250 ns	1M - bit, 3 Volt Read and 3 Volt Write, Sectored Flash	Now
AT29LV020	256K×8	200 - 250 ns	2M bit, 3 - Volt Read and 3 Volt Write, Sectored Flash	Now
AT29LV040A	512K×8	200 - 250 ns	4M bit, 3 - Volt Read and 3 - Volt Write, Sectored Flash	Now
AT49LV512	642K×8	90～120 ns	512K, 3 - Volt Read and 2.7 Volt Byte Write Flash	Now
AT49LV001T	128K×8	55 - 90 ns	1M bit, 3 Volt Read and 2.7 - Volt Byte Write Sectored Flash (Top Boot)	Now
AT49LV010	128K×8	120 - 150 ns	1M - bit, 3 - Volt Read and 3 Volt Byte - Write Flash	Now
AT49HLV010	128K×8	90 ns	1M - bit, 3 - Volt Read and 3 Volt Byte - Write Flash (High - Speed)	Now
AT49LV002T	256K×8	55 - 90 ns	2M bit, 3 Volt Read and 2.7 - Volt Byte - Write Flash (Top Boot)	Now
AT49LV020	256K×8	90 - 150 ns	2M bit, 3 Volt Read and 3 - Volt Byte Write Flash	Now

表 2 （续）

产品型号	构 成	速 度	特 性	供货
AT49LV2048	128K×16	120～150 ns	2M－bit, 3－Volt Read and 3－Volt Byte Write Flash	Now
AT49LV040	512K×8	90～150 ns	4M－bit, 3－Volt Read and 3－Volt Byte－Write Flash	Now
AT49LV4096	256K×16	120～150 ns	4M－bit, 3－Volt Read and 3－Volt Byte－Write Flash	Now
AT49LV080	1 024K×8	120～150 ns	8M－bit, 3－Volt Read and 3－Volt Byte－Write Flash	Now
AT49LV080T	1 024K×8	120～150 ns	8M－bit, 3－Volt Read and 3－Volt Byte－Write Flash (Top Boot)	Now
AT49LV008	1024K×8	120～150 ns	8M－bit, 3－Volt Read and 3－Volt Byte Write Flash	Now
AT49LV8192	512K×16	120～150 ns	8M－bit, 3－Volt Read and 3 Volt Byte－Write Flash	Now
AT49LV8192T	512K×16	120～150 ns	8M－bit, 3－Volt Read and 3－Volt Byte－Write Flash (Top Boot)	Now
Standard Voltage (5.0V) 标准电压型				
AT29C256	32K×8	70～150 ns	256K, 5－Volt Read and 5－Volt Write, Sectored Flash	Now
AT29C257	32K×8	70～150 ns	256K, 5－Volt Read and 5－Volt Write, Sectored Flash	Now
AT29C512	64K×8	70～150 ns	512K, 5－Volt Read and 5－Volt Write, Sectored Flash	Now
AT29C1024	64K×16	70～150 ns	1M－bit, 5－Volt Read and 5－Volt Write, Sectored Flash	Now
AT29C010A	128K×8	70～150 ns	1M－bit, 5－Volt Read and 5－Volt Write, Sectored Flash	Now
AT29C020	256K×8	90～150 ns	2M－bit, 5－Volt Read and 5 Volt Write, Sectored Flash	Now
AT29C040A	512K×8	100～200 ns	4M－bit, 5－Volt Read and 5－Volt Write, Sectored Flash	Now
AT49F512	64K×8	55～90 ns	512K, 5－Volt Read and 5－Volt Byte－Write, Sectored Flash	Now
AT49F010	128K×8	70～120 ns	1M－bit, 5－Volt Read and 5－Volt Write, Sectored Flash	Now
AT49F001T	128K×8	55～90 ns	1M－bit, 5－Volt Read and 5 Volt Byte－Write, Sectored Flash (Top Boot)	Now
AT49F010	128K×8	45～55 ns	1M－bit, 5 Volt Read and 5－Volt Byte－Write, Flash (High－Speed)	Now
AT49F1 024	64K×16	55～90 ns	1M－bit, 5－Volt Read and 5 Volt Byte－Write, Flash	Now
AT49F1 025	64K×16	55～90 ns	1M－bit, 5－Volt Read and 5－Volt Byte－Write, Flash	Now
AT49F020	256K×8	55～120 ns	2M－bit, 5－Volt Read and 5 Volt Byte－Write, Flash	Now
AT49F002T	256K×8	50～70 ns	2M－bit, 5－Volt Read and 5－Volt Byte－Write, Sectored Flash	Now
AT49F2048	128K×16	70～120 ns	2M－bit, 5 Volt Read and 5－Volt Byte－Write, Flash	Now
AT49F040	512K×8	70～150 ns	4M－bit, 5－Volt Read and 5 Volt Byte－Write, Flash	Now
AT49F4096	255K×16	90～120 ns	4M－bit, 5 Volt Read and 5 Volt Byte－Write, Flash	Now
AT49F080	1M×8	90～150 ns	8M－bit, 5－Volt Read and 5－Volt Byte－Write, Flash	Now
AT49F080T	1 024K×8	90～150 ns	8M－bit, 5－Volt Read and 5 Volt Byte－Write, Flash (Top Bool)	Now
AT49F008	1 024K×8	90～150 ns	8M－bit, 5 Volt Read and 5－Volt Byte－Write, Flash	Now
AT49F8192	512K×16	90～120 ns	8M－bit, 5 Volt Read and 5 Volt Byte－Write, Flash	Now
AT49F8192T	512K×16	90～120 ns	8M－bit, 5－Volt Read and 5－Volt Byte－Write, Flash (Top Bool)	Now
AT49F1064	1 024K×16	55～90 ns	16M－bit, 5－Volt Read and 5－Volt Byte－Write, Sectored Flash	Now

表3 串行海量电可擦写存储器

产品型号	速 度	容 量	特 性	供货
Battery Voltage (2.7~3.6V) 电池电压型				
AT45DB011	10 MHz	1Mb	2.7 Volt Only Serial-Interface Flash with Orte 264 - Byte SRAM Buffer	3Q98
AT45DB021	5 MHz	2Mb	2.7 Volt Only Serial-Interface Flash with Two 264 - Byte SRAM Buffers	Now
AT45DB041	5 MHz	4Mb	2.7 Volt Only Serial-Interface Flash with Two 264 - Byte SRAM Buffers	Now
AT45DB081	10 MHz	8Mb	2.7 - Volt Only Serial-Interface Flash with Two 264 - Byte SRAM Buffers	Now
AT45DB161	13 MHz	16Mb	2.7 - Volt Only Serial Interface Flash with Two 264 Byte SRAM Buffers	2Q98
Standard Voltage (5.0V) 标准电压型				
AT45DB011	10 MHz	8Mb	5.0 - Volt Only Serial Interface Flash with One 264 Byte SRAM Buffers	3Q98
AT45DB021	10 MHz	2Mb	5.0 Volt Only Serial-Interface Flash with Tow 264 - Byte SRAM Buffers	Now
AT45DB041	10 MHz	4Mb	5.0 - Volt Only Serial-Interface Flash with Tow 264 - Byte SRAM Buffers	Now
AT45DB081	10 MHz	8Mb	5.0 - Volt Only Serial Interface Flash with Tow 264 - Byte SRAM Buffers	Now
AT45DB161	13 MHz	16Mb	5.0 - Volt Only Serial Interface Flash with Tow 264 - Byte SRAM Buffers	2Q98

表4 门阵列/嵌入式阵列

产品型号	门 数	引脚数	特 性	供货
ATL35 Series	Up to 3.7M	Up to 976	0.35 Micron CMOS Gate Array/Embedded Array, 1.0 - Volt to 3.3 - Volt Operation, 23 Versions with Various Pin & Gate Count, Memory, Megacells	Now
ATL50 Series	Up to 590K	Up to 480	0.5 Micron CMOS Gate Array/Embedded Array, 2.0 - Volt to & 3.3 - Volt Mixed Voltage Operation, 16 Versions with Various Pin & Gate Counts, Memory, Megacells	Now
ATLS60 Series	Up to 88K	Up to 256	0.6 - Micron CMOS Gate Array/Embedded Array, 3.3 Volt & 5.0 Volt Operation, Staggered Row Bond Pads, 8 Versions with Various Pin & Gate Counts, Memory, Megacells	Now
ATLS60 Series	Up to 590K	Up to 480	0.6 Micron CMOS Gate Array/Embedded Array, 3.3 Volt & 5.0 - Volt Operation, 16 Versions with Various Pin & Gate Counts, Memory, Megacells	Now
Macrocells			ARM7TDMI™, AVR®(8 bit RISC), OakDSPCore™, LodeDSPCore™, Ethernet MAC, USB Cores, PCI Cores	Now
Memory			SRAM, ROM, Dual Port SRAM, FIFO and CAM	Now
I/O Interfaces			CMOS, TTL, LVDS, PCI, USB, SCSI, LVD SCSI, PLL	Now

表5 PCMCIA标准闪速储存卡

产品型号	构 成	Vcc	特 性	供 货
AT5FC001	1Mb	5V	PCMCIA Compatible Flash Memory Card	Now
AT5FC002	2Mb	5V	PCMCIA Compatible Flash Memory Card	Now
AT5FC004	4Mb	5V	PCMCIA Compatible Flash Memory Card	Now
AT5FC008	8Mb	5V	PCMCIA Compatible Flash Memory Card	Now

表 6 PLDS 可编程逻辑部件

产品型号	引 脚	速 度	特 性	供 货
5 – Volt Electrically Erasable 5V 电可擦除				
ATF16V8B	20 – Pin	7.5~25 ns	8 FFs, 8 I/O Pins, Standard – Power	Now
ATF16V8BQ/BQL	20 – Pin	10~25 ns	8 FFs, 8 I/O Pins, Quarter – Power, Low Power	Now
ATF16V8C	20 – Pin	5~7.5 ns	8 FFs, 8 I/O Pins, Standard – Power	Now
ATF16V8CZ	20 – Pin	12~15 ns	8 FFs, 8 I/O Pins, Zero – Power	Now
ATF20V8B	24.28 – Pin	7.5~25 ns	8 FFs, 8 I/O Pins, Standard – Power	Now
ATF20V8BQ/BQL	24, 28 – Pin	10~25 ns	8 FFs, 8 I/O Pins, Quarter – Power, Low – Power	Now
ATF20V8C	24.28 – Pin	5~7 ns	8 FFs, 8 I/O Pins, Standard – Power	2H98
ATF20V8CZ	24,28 – Pin	12~15 ns	8 FFs, 8 I/O Pins, Zero – Power	2H98
ATF22V10B	24.28 – Pin	7.5~25 ns	10 FFs, 10 I/O Pins, Standard – Power	Now
ATF22V10BQ/BQL	24, 28 – Pin	15~25 ns	10 FFs, 10 I/O Pins, Quarter – Power, Low – Power	Now
ATF22V10C	24.28 – Pin	5~10 ns	10 FFs, 10 I/O Pins, Standard – Power	Now
ATF22V10CZ	24,28 – Pin	12~15 ns	10 FFs, 10 I/O Pins, Zero – Power	Now
ATFV750C/CL	24, 28 – Pin	7.5~25 ns	20 FFs, 10 I/O Pins, Standard & Low – Power	2Q98
ATFV2500C/CL	40, 44 – Pin	20~25 ns	48 FFs, 24 I/O Pins, Standard & Low – Power	2Q98
ATFV250CQ/CQL	40, 44 – Pin	20~25 ns	48 FFs, 24 I/O Pins, Quarter – Power, Low – Power	2Q98
ATFV1500/L	44 – Pin	7.5~25 ns	32 Macrocell, Standard & Low – Power	Now
ATFV1500A/AL	44 – Pin	7.5~25 ns	32 Macrocell, Standard & Low – Power	Now
ATFV1502AS/L	44 – Pin	7.5~25 ns	32 Macrocell w/ISP, Standard & Low – Power	2Q98
ATFV1504AS/L	44, 68, 84, 100 – Pin	7.5~25 ns	64 Macrocell w/ISP, Standard & Low – Power	1Q98
ATFV1508AS/L	68, 84, 100, 160 – Pin	7.5~25 ns	128 Macrocell w/ISP, Standard & Low – Power	Now
ATV1516AS/L	160, 192, 208 – Pin	10~25 ns	256 Macrocell w/ISP, Standard & Low – Power	3Q98
Low – Voltage (3.0V) Electrically Erasable 低电压电可擦除				
ATF16V8C	20 – Pin	10~15 ns	8 FFs, 8 I/O Pins, Low – Voltage	Now
ATF16V8CZ	20 – Pin	15~25 ns	8 FFs, 8 I/O Pins, Low – Voltage, Zero – Power	Now
AT22LV10/L	24, 28 Pin	20~30 ns	10 FFs, 10 I/O Pins, Standard & Low – Power (EPROM – based)	Now
ATF1500ABV	44 – Pin	12~15 ns	32 FFs, 32 I/O Pins, Low – Voltage	Now
ATF1500ABVL	44 – Pin	25 ns	32 FFs, 32 I/O Pins, Low – Voltage & Low – Power	Now
ATF22LV10C	24, 28 Pin	10~15 ns	10 FFs, 10 I/O Pins, Low – Voltage	Now
ATF22LV10CZ	24, 28 – Pin	15~25 ns	10 FFs, 10 I/O Pins, Low – Voltage, Zero – Power	Now
5 – Volt EPROM – Based 基于 5V EPROM				
ATV750/L	24, 28 – Pin	20~25 ns	20FFs, 10 I/O Pins, Standard & Low-Power	Now
ATV50B/BL	24, 28 – Pin	7.5~25 ns	20FFs, 10 I/O Pins, Standard & Low – Power	Now
ATV2500H/L	40, 44 – Pin	25~35 ns	48FFs, 24 I/O Pins, Standard & Low – Power	Now
ATV2500B/BL	44 Pin	12~20 ns	48 FFs, 24 I/O Pins, Standard & Low – Power	Now
ATV2500BQ/BQL	40, 44 – Pin	20~25 ns	48 FFs, 24 I/O Pins, Quarter – Power, Low – Pwer	Now

表 7 单元库

产品型号	特 性	供货
ECPD07	0.8 - Micron CMOS Cell Based IC Family, 5.0 Volt Operation, Digital, Analog, Memory, Megacells	Now
AT55K	0.5 - Micron CMOS Cell - Based IC Family, 3.3 - Volt Operation, Digital, Analog, Memory, Megacells	Now
AT56K	0.35 - Micron CMOS Cell - Based IC Family, 3.3 Volt Operation, Digital, Analog, Memory, Megacells	Now
AT57K	0.25 - Micron CMOS Cell - Based IC Family, 3.3 Volt Operation, Digital, Analog, Memory, Megacells	4Q98
AT19.6K	0.8 - Micron E^2PROM with Logic IC Family, 5.0 Volt Opera ration	Now
AT19.9K	0.6 - Micron E^2PROM with Logic IC Family, 5.0 Volt Opera ration	Now
AT55.8K	0.5 - Micron CMOS with Embeeded E^2PROM with Logic IC Family, 3.3 - Volt Opera	Now
AT56.35K	0.4 - Micron E^2PROM with Logic IC Family, 3.3 Volt Opera ration	Now
Macrocells	ARM7TDMI, AVR, OaKDSP™, Ethernet MAC, AT8051, AT8237, AT8251, AT8254, AT8255, AT8259, AT82530, AT14818, AT16450, RAM, Dual - port RAM, ROM, Flash, E^2PROM, PCI, SPI, USB, Codec, A/D, D/A, OPAmp, Comp., Osc., etc.	4Q98 Now

表 8 可编程逻辑部件开发工具——软件和硬件

产品型号	特 性	供 货
ATDS1100PC	Atmel - Synario Entry (Includes ABEL, Schematic Entry, Simulation)	Now
ATDS1120PC	Atmel - Synario Verilog Simulation	Now
ATDS1130PC	Atmel - Synario VHDL Synthesis	Now
ATDS1150PC	Atmel ISP Kit	Now
ATDS1160PC	Atmel - ISP Board	Now
ATDS1161PC	Atmel - 44 - Pin PLCC Adaptor Board	Now
ATDS1162PC	Atmel - 44 Pin TQFP Adaptor Board	Now
ATDS1163PC	Atmel - 68 - Pin PLCC Adaptor Board	Now
ATDS1164PC	Atmel - 100 - Pin TQFP Adaptor Board	Now
ATDS1165PC	Atmel - 100 - Pin TQFP Adaptor Board	Now
ATDS1166PC	Atmel - 160 Pin PQFP Adaptor Board	Now
ATDS1170PC	Atmel - 1500AS Family Demo Board	Now

表 9 场可编程门阵列电可擦写结构存储器

产品型号	内 存	特 性	供 货
Standard Voltage (5.0V) 标准电压型			
AT17C65	65 536×1	65K FPGA Configuration E^2PROM	Now
AT17C128	131072×1	128K FPGA Configuration E^2PROM	Now
AT17C256	262 144×1	256K FPGA Configuration E^2PROM	Now
AT17C512	524 288×1	512K FPGA Configuration E^2PROM	Now
AT17C010	1 048, 576×1	1Meg FPGA Configuration E^2PROM	Now
Low - Voltage (3.3V) 低电压型			
AT17LV65	65 536×1	65K FPGA Configuration E^2PROM, 3.3 - Volt	Now
AT17LV128	131072×1	128K FPGA Configuration E^2PROM, 3.3 Volt	Now
AT17LV256	262 144×1	256K FPGA Configuration E^2PROM, 3.3 - Volt	Now
AT17LV512	524 288×1	512K FPGA Configuration E^2PROM, 3.3 - Volt	Now
AT17LV010	1 048 576×1	1Meg FPGA Configuration E^2PROM, 3.3 - Volt	Now

表 10 场可编程门阵列——AT6000

产品型号	寄存器	可用门数	频 率	特 性	供 货
Standard Voltage (5.0V) 标准电压型					
AT6002	1 024	6K	350 MHz	96 I/O Pins, 5 - Volt, Very Low Power	Now
AT6003	1 600	9K	350 MHz	120 I/O Pins, 5 - Volt, Very Low Power	Now
AT6005	3 136	15K	350 MHz	140 I/O Pins, 5 - Volt, Very Low Power	Now
AT6010	6 400	30K	350 MHz	204 I/O Pins, 5 - Volt, Very Low Power	Now
Low - Voltage (3.3 V) 低电压型					
AT6002LV	1 024	6K	250 MHz	96 I/O Pins, 3.3 - Volt, Very Low Power	Now
AT6003LV	1 600	9K	250 MHz	120 I/O Pins, 3.3 - Volt, Very Low Power	Now
AT6006LV	3 136	15K	250 MHz	140 I/O Pins, 3.3 - Volt, Very Low Power	Now
AT6010LV	6 400	30K	250 MHz	204 I/O Pins, 3.3 - Volt, Very Low Power	Now

表 11 场可编程门阵列——AT40 系列

产品型号	寄存器	可用门数	频率	RAM	特 性	供 货
AT40K05	256	5K~10K	250MHz	2,048	128 I/O Pins, 5 - Volt, Very Low Power	Now
AT40K10	576	10K~20K	250MHz	4,096	192 I/O Pins, 5 - Volt, Very Low Power	2Q98
AT40K20	1 024	20K~30K	250MHz	8,192	256 I/O Pins, 5 - Volt, Very Low Power	Now
AT40K30	1 600	30K~40K	250MHz	12,800	320 I/O Pins, 5 - Volt, Very Low Power	2Q98
At40K40	2 304	40K~50K	250MHz	18,432	384 I/O Pins, 5 - Volt, Very Low Power	2Q98
Low - Voltage (3.3V) 低电压型						
AT40K05LV	256	5K~10K	250MHz	2 048	128 I/O Pins, 3.3 - Volt, Very Low Power	2Q98
At40K10LV	576	10K~20K	250MHz	4 096	192 I/O Pins, 3.3 - Volt, Very Low Power	2Q98
AT40K20LV	1 024	20K~30K	250MHz	8 192	256 I/O Pins, 3.3 - Volt, Very Low Power	Now
AT40K30LV	1 600	30K~40K	250KHz	12 800	320 I/O Pins, 3.3 - Volt, Very Low Power	2Q98
AT40K40LV	2 304	40K~50K	250MHz	18 432	384 I/O Pins, 3.3 - Volt, Very Low Power	2Q98

FPGA Design Development Software

FPGA design tools are available across a broad range of CAE tool vendors and PC and worKstation platforms. Design methods supported include: schematic capture, logic synthesis (VHDL and Verilog), PLD entry, (ABEL and CUPL), and automatic component generation of hard macros for user - parametrized structured logic (arithmetic elements, counters, registers, encoders, decoders, and other common functions). Refer to current Configurable Logic Data BooK.

CAE Tool Support:

Cadence, Everest, Exemplar, Mentor, OrCAD, Synopsys, Synario, Veribest, Verilog, View-Logic.

Platform Support:

PC(Windows 3.1, 95, NT), SUN WorKstations, HP WorKstations.

表 12 AT91 系列 16 位/32 位精简指令集单片机

产品型号	处理器	特 性	供 货
AT91M40100	ARM7TDMI	General Purpose, 16/32 - bit RISC Microcontroller with 1K bytes RAM, 100 - lead TQFP pacKage	2Q98
AT91M40400	ARM7TDMI	General Purpose, 16/32 - bit RISC Microcontroller with 4K bytes RAM, 100 - lead TQFP pacKage	Now
AT91M4040x	ARM7TDMI	General Purpose, 16/32 - bit RISC Microcontroller with 4K bytes RAM and up to 2M bit ROM, 100 - lead TQFP pacKage	2Q98

表 13 AT91 系列单片机和开发工具

产品型号	特 性	供 货
AT91SDT	ARM Software Development ToolKit	Now
AT91DB01	AT91 Development Board V1	Now
AT91EICE	ARM Embedded ICE Box	Now
AT91EB01	AT91 Evaluation Board V1	2Q98

表 14 AT90 系列增强型精简指令集 8 位单片机

产品型号	处理器	特 性	供 货
AT90S1200	AVR	AVR RISC In System Programmable Microcontroller with 1K byte Flash and 64 bytes E^2PROM, 20 - Pin PDIP, SOIC, and SSOP PacKages	Now
AT90S2313	AVR	AVR RISC In - System Progarammable Microcontroller with 2K bytes Flash, 128 bytes E^2PROM and 128 bytes SRAM, 20 Pin PDIP and SOIC PacKages	Now
AT90S2323	AVR	AVR RISC In - System Programmable Microcontroller with 2K bytes Flash, 128 bytes E^2PROM and 128 bytes SRAM, 8 - Pin SOIC PacKage	Now
AT90AS2343	AVR	AVR RISC In - System Programmable Microcontroller with 2K bytes Flash, 128 bytes E^2PROM and 128 bytes SRAM, 8 - Pin SOIC PacKage	Now
AT90LS2323	AVR	Low - Voltage, AVR RISC In - System Programmable Microcontroller with 2K bytes Flash, 128 bytes E^2PROM and 128 bytes SRAM, 8 - Pin SOIC PacKage	Now
AT90LS2343	AVR	Low - Voltage, AVR RISC In - System Programmable Microcontrollet with 2K bytes Flash, 128 bytes E^2PROM and 128 bytes SRAM, 8 - Pin SOIC Package	Now
AT90S4414	AVR	AVR RISC In - System Programmable Microcontroller with 4K bytes Flash, 256 bytes E^2PROM and 256 bytes SRAM, 40 - Pin PDIP, 44 - Pin TQFP and PLCC PacKages	Now
AT90S8515	AVr	AVR RISC In - System Programmable Microcontroller with 8K bytes Flash, 512 bytes E^2PROM and 512 bytes SRAM, 40 - Pin PDIP, 44 - Pin TQFP and PLCC PacKages	Now
ATmega103	AVR	AVR RISC In System Programmable Microcontroller with 128K bytes Flash, 4K bytes E^2PROM and 4K bytes SRAM, 64 - Pin TQFP PacKage	Now
ATmega103L	AVR	Low Voltage, AVR RISC In - System Programmable Microcontroller with 128K bytes Flash, 4K bytes E^2PROM and 4K bytes SRAM, 64 - Pin TQFP PacKage	Now

表 15 AT90 系列单片机开发工具

开发工具	特 性	供 货
AT901CEPRO	Atmel In - Circuit Emulator	Now
MCU00100	AT89/90 Series Flash Microcontroller Starter Kit with Development Board	Now

表 16 AT89 系列 8 位单片机

产品型号	内 存	特 性	供 货
AT80F51	4K×8	80C31 Microcontroller with 4K ROM replacement	Now
AT80F52	8K×8	80C32 Microcontroller with 8K ROM replacement	Now
AT89C51	4K×8	80C31 Microcontroller with 4K bytes Flash	Now
AT89LV51	4K×8	2.7－Volt, 80C31 Microcontroller with 4K bytes Flash	Now
AT89C52	8K×8	80C32 Microcontroller with 8K bytes Flash	Now
AT89LV	8K×8	2.7－Volt, 80C32 Microcontroller with 8K bytes Flash	Now
AT89C1051	1K×8	80C31 Microcontroller with 1K bytes Flash, 20－Pin PacKage	Now
AT89C2051	2K×8	80C31 Microcontroller with 2K bytes Flash, 20－Pin PacKage	Now
AT89S8252	8K×8	In－System Programmable Microcontroller with 8K bytes Flash and 2K bytes E^2PROM	Now
AT89LS8252	8K×8	Low－Voltage, In System Programmable Microcontroller with 8K bytes Flash and 2K bytes E^2PROM	Now
AT89S53	12K×8	In－System Programmable Microcontroller with 12K bytes Flash	Now
AT89LS53	12K×8	Low－Voltage, In－System Programmable Microcontroller with 12K bytes Flash	Now
AT89C55	20K×8	80C32 Microcontroller with 20K bytes Flash	Now
AT89LV55	20K×8	2.7－Volt, 80C32 Microcontroller with 20K bytes Flash	Now

表 17 逻辑加密型 IC 卡

产品型号	内 存	特 性	供 货
AT88SC101	1 024×1	1K Serial E^2PROM with Security, 1 Memory Zone, 1 024 Bits	Now
AT88SC102	1 024×1	1K Serial E^2PROM with Security, 2 Memory Zones, 512 Bits Each	Now
AT88SC103	1 536×1	1.5K Serial E^2PROM with Security, 3 Memory Zones, 512 Bits Each	Now
AT88SC153	192×8	1.5K Serial E^2PROM with Security, and Authentication, 3 Memory Zones, 512 Bits Each	2Q98
AT88SC1601	15 872×1	16K Serial E^2PROM with Security, 1Memory Zone, 15 872 Bits	Now
AT88SC1604	15 968×1	16K Serial E^2PROM with Security, 3 Memory Zones, 4 096 Bits Each, and 1 Memory Zone, 3 680 Bits	Now
AT88SC1608	2 048×8	16K Serial E^2PROM with Security and Authentication, 8 Memory Zones, 2 048 Bits Each, and 1 Configuration Zone, 1 024 Bits	Now

表 18 超级 CPU IC 卡

产品型号	程序内存	应用内存 Flash/E^2PROM	RAM	电源电压	加密器	RF 射频接口	供 货
AT90SC1616C	16K Flash	16K Bytes	1K Bytes	2.7～5.5 V	Yes	No	4Q98
AT90SC3232CR	32K ROM	32K Bytes	1K Bytes	2.7～5.5 V	Yes	No	4Q98
AT90SC3232CRF	32K ROM	32K Bytes	1K Bytes	2.7～5.5 V	Yes	Yes	1Q99

表 19 非标触型 IC 卡

产品型号	E^2PROM 内存	特 征	供 货
AT88RF256	256×1	Read/Write RFID Transponder with Passwords and Data LocKing	3Q98
AT88RF8714	2K×8	Contactless Card IC with AVR Microprocessor, 8K Bytes ROM, 256 Bytes SRAM	4Q98

表 20 CPU 型 IC 卡

产品型号	Flash	E^2PROM	RAM	T=0 Hardware	电源电压	供 货
AT89SC168	16K Bytes	8K Bytes	256 Bytes	Yes	5.0V	Now
At89Sc168A	16K Bytes	8K Bytes	512 Bytes	No	2.7～5.5 V	1Q98
AT89SC1616A	16K Bytes	16K Bytes	512 Bytes	No	2.7～5.5 V	2Q98
AT89SC248A	24K Bytes	8K Bytes	512 Bytes	No	2.7～5.5 V	2Q98

表 21 并行电可擦存储器

产品型号	构 成	速 度	特 性	供 货
High-Speed 高速度型				
AT28HC64B	8K×8	55～120 ns	64K E^2PROM with 64-Byte Page & Software Data Protection	Now
AT28HC256	32K×8	70～120 ns	256K E^2PROM with 64-Byte Page & Software Data Protection	Now
AT28HC256E	32K×8	70～120 ns	256K E^2PROM with Extended Endurance	Now
AT28HC256F	32K×8	70～120 ns	256K E^2PROM with Fast Write	Now
Battery-Voltage (2.7 V to 3.6 V) 电池电压型				
AT28BV16	2K×8	200～250 ns	16K E^2PROM, 2.7-Volt	Now
AT28BV64	8K×8	200～250 ns	64K E^2PROM, 2.7-Volt	Now
Low-Voltage (3.0 V to 3.6 V) 低电压型				
AT28LV010	128K×8	200～250 ns	1M-bit E^2PRom with 128-Byte Page & Software Data Protection, 3.0-Volt	Now
AT28LV64B	8K×8	200～250 ns	64K E^2PROM with 64-Byte Page & Software Data Protection, 3.0-Volt	Now
AT28LV256	32K×8	200～250 ns	256K E^2PROM with 64-Byte Page & Software Data Protection, 3.0-Volt	Now
Standard Voltage (5.0 V) 标准电压型				
AT28C16	2K×8	150 ns	16K E^2PROM	Now
AT28C16E	2K×8	150 ns	16K E^2PROM with Extended Endurance & Fast Write	Now
AT28C17	2K×8	150 ns	16K E^2PROM with Ready/Busy	Now
AT28C17E	2K×8	150 ns	16K E^2PROM with Ready/Busy & Extended Endurance & Fast Write	Now
AT28C64	8K×8	120～250 ns	64K E^2PROM	
AT28C64E	8K×8	120～250 ns	64K E^2PROM with Extended Endurance & Fast Write	Now
AT28C64X	8K×8	120～250 ns	64K E^2PROM without Ready/Busy	Now
AT28C64B	8K×8	125～250 ns	64K E^2PROM with 64-Byte Page & Software Data Protection	Now
AT28C256	32K×8	150～250 ns	256K E^2PROM with 64-Byte Page & Software Data Protection	Now
AT28C256E	32K×8	150～250 ns	256K E^2PROM with Extended Endurance	Now
AT28C256F	32K×8	150～250 ns	256K E^2PROM with Fast Write	Now
AT28C010	128K×8	120～250 ns	1M-bit E^2PROM with 128-Byte Page & Software Data Protection	Now
AT28C010E	128K×8	120～250 ns	1M-bit E^2PROM with 128-Byte Page & Extended Endurance & Software Data Protection	Now
AT28C040	512K×8	200～250 ns	4M-bit E^2PROM with 256-Byte Page & Software Data Protection	Now

表 22 各种串行电可擦存储器

产品型号	构 成	V_{CC}	特 性	供 货
AT24C01	128×8	1.8, 2.5, 2.7, 50 V	1K, 2 - Wire Bus Serial E^2PROM, Non - Cascadable	Now
AT24C21	128×8	2.5 V	1K, 2 - Wire Bus Serial E^2PROM, Dual Mode, Plug & Play Operation	Now
AT24C01A	128×8	1.8, 2.5, 2.7, 5.0 V	1K, 2 - Wire Bus Serial E^2PROM with	Now
AT24C02	256×8	1.8, 2.5, 2.7, 5.0 V	2K, 2 - Wire Bus Serial E^2PROM with Full Hardware Write Protection	Now
AT24C02A	256×8	1.8, 2.5, 2.7, 5.0 V	2K, 2 - Wire Bus Serial E^2PROM with Ful Hardware Write Protection	Now
AT34C02	256×8	1.8, 2.7, 5.0 V	2K, 2 - Wire Serial E^2PROM with Software Write Protection	Now
AT24C04	512×8	1.8, 2.5, 2.7, 5.0 V	4K, 2 - Wire Bus Serial E^2PROM with Full Hardware Write Protection	Now
AT24C04A	512×8	1.8, 2.5, 2.7, 5.0 V	4K, 2 - Wire Bus Serial E^2PROM with Full Hardware Write Protection	Now
AT24C08	1 024×8	1.8, 2.5, 2.7, 5.0 V	8K, 2 - Wire Bus Serial E^2PROM	Now
AT24C08A	1 024×8	1.8, 2.5, 2.7, 5.0 V	8K, 2 - Wire Bus Serial E^2PROM with Full hardware Write Protection	Now
AT24C16	2 048×8	1.8, 2.5, 2.7, 5.0 V	16K, 2 - Wire Bus Serial E^2PROM with Half Hardware Write Protection	Now
AT24C164	2 048×8	1.8, 2.5, 2.7, 5.0 V	16K, 2 - Wire bus Serial E^2PROM with Cascadable Feature	Now
AT24C32	4 096×8	1.8, 2.5, 2.7, 5.0 V	32K, 2 - Wire Bus Serial E^2PROM with Cascadable Feature	Now
AT24C64	8 192×8	1.8, 2.5, 2.7, 5.0 V	64K, 2 - Wire Bus Serial E^2PROM with Casscadable Feature	Now
AT24C128	16 384×8	1.8, 2.7, 5.0 V	128K, 2 - Wire Bus Serial E^2PROM with Casscadable Feature	Now
AT24C256	32 768×8	1.8, 2.7, 5.0 V	256K, 2 - Wire Bus Serial E^2PROM with Cascadable Feature	Now
AT25010	128×8	1.8, 2.7, 5.0 V	1K, SPI Bus Serial E^2PROM, SPI Mode 0 and 3	Now
AT25020	256×8	1.8, 2.7, 5.0 V	2K, SPI Bus Serial E^PROM, SPI Mode 0 and 3	Now
AT25040	512×8	1.8, 2.7, 5.0 V	4K, SPI Bus Serial E^2PROM, SPI Mode 0 and 3	Now
AT25080	1 024×8	1.8, 2.7, 5.0 V	8K, SPI Bus Serial E^2PROM, SPI Mode 0 and 3	Now
AT25160	2 048×8	1.8, 2.7, 5.0 V	16K, SPI Bus Serial E^2PROM, SPI Mode 0 and 3	Now
AT25320	4 096×8	1.8, 2.7, 5.0 V	32K, SPI Bus Serial E^2PROM, SPI Mode 0 and 3	Now
AT25640	8192×8	1.8, 2.7, 5.0 V	64K, SPI Bus Serial E^2PROM, SPI Mode 0 and 3	Now
AT25128	16 384×8	1.8, 2.7, 5.0 V	128K, SPI Bus Serial E^2PROM, SPI Mode 0 and 3	Now
AT25256	32 768×8	1.8, 2.7, 5.0 V	256K, SPI Bus Serial E^2PROM, SPI MOde 0 and 3	Now
AT25P1024	131 072×8	1.8, 2.7, 5.0 V	1M, SPI Bus Serial E^2PROM, Page - Write Only, SPI Mode 0 and 3	Now
AT93C46	64×16/128×8	1.8, 2.5, 2.7, 5.0 V	1K, 3 - Wire Bus Serial E^2PROM	Now
AT93C46A	64×16	1.8, 2.5, 2.7, 5.0 V	1K, 3 - Wire Bus Serial E^2PROM	Now
AT93C56	128×16/256×8	2.5, 2.7, 5.0 V	2K, 3 - Wire Bus Serial E^2PROM	
AT93C57	128×16/256×8	2.5, 2.7, 5.0 V	2K, 3 - Wire Bus Serial E^2PROM with Special Address	Now
AT93C66	256×16/512×8	2.5, 2.7, 5.0 V	4K, 3 - Wire Bus Serial E^2PROM	
AT59C11	64×16/128×8	2.5, 2.7, 5.0 V	1K, 4 - Wire Bus Serial E^2PROM	Now
AT59C22	128×16/256×8	2.5, 2.7, 5.0 V	2K, 4 - Wire Bus Serial E^2PROM	Now
AT59C13	256×16/512×8	2.5, 2.7, 5.0 V	4K, 4 - Wire Bus Serial E^2PROM	Now