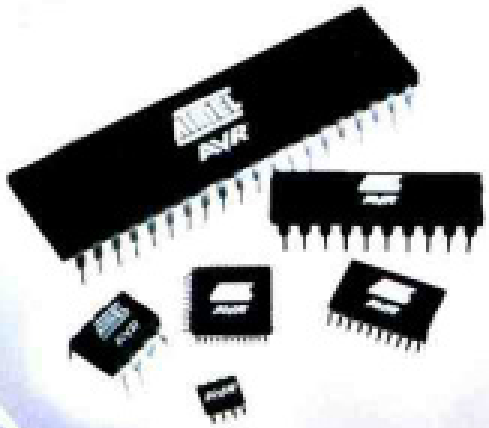
MCU

AVR 单片机应用设计



丁化成 耿德根 李君凯 编著

北京航空航天大学出版社
http://www.buaapress.com.cn

ATMEL 公司单片机系列（AVR，AT89）

- AVR 高速嵌入式单片机原理与应用　耿德根 等编　定价：40.00 元
- AVR 单片机应用技术　李 勋 等编著　定价：23.00 元
- AVR 单片机应用设计　丁化成 等编著　定价：22.00 元
- ATMEL89 系列单片机应用技术　余永权 编著　定价：32.00 元

ISBN 7-81077-179-5

9 787810 771795 >

ISBN 7-81077-179-5/TP · 100

定价：22.00 元

AVR 单片机应用设计

丁化成　耿德根　李君凯　编著

北京航空航天大学出版社

http://www.buaapress.com.cn

内 容 简 介

AVR 单片机是美国 ATMEL 公司 1997 年推出的单片机系列。本书以其代表型号 AT90S8535 为主线，讲述该系列单片机的内部结构、开发工具、指令系统、各种接口及其应用程序举例和设计方法。学习了这种功能较全的单片机，对于 AVR 系列其他型号单片机的应用就可以举一反三。

AVR 单片机具有高速度、高保密性、低功耗的特点。AT90S8535 内含可反复编程的 Flash 程序存储器、SRAM 和 EEPROM 两种数据存储器、定时器/计数器、方向可定义的 I/O 口、同步串行口、异步串行口、A/D 转换器及 PWM 等丰富的内部资源。一般的应用系统只需此一块芯片即可实现智能化。

本书可作为大专院校的单片机教材和科技人员的单片机应用参考书。

图书在版编目(CIP)数据

AVR 单片机应用设计/丁化成等编著. —北京：北京航空航天大学出版社，2002.5

ISBN 7-81077-179-5

Ⅰ. A… Ⅱ. 丁… Ⅲ. 单片微型计算机，AVR 程序设计 Ⅳ. TP368.1

中国版本图书馆 CIP 数据核字(2002)第 022097 号

AVR 单片机应用设计

丁化成 耿德根 李君凯 编著

责任编辑 王 瑛

责任校对 戚 爽

*

北京航空航天大学出版社出版发行

北京市海淀区学院路 37 号(100083) 发行部电话(010)82317024 传真(010)82328026

http://www.buaapress.com.cn

E-mail:pressell@publca.bj.cninfo.net

河北省涿州市新华印刷厂印制 各地书店经销

*

开本:787×1092 1/16 印张:14.5 字数:371 千字

2002 年 5 月第 1 版 2003 年 3 月第 2 次印刷 印数:5 001～10 000 册

ISBN 7-81077-179-5/TP·100 定价:22.00 元

前　言

AVR单片机是ATMEL公司1997年推出的全新配置精简指令集(RISC)单片机系列。片内程序存储器采用Flash存储器,可反复编程修改上千次,便于新产品开发;程序高度保密,避免非法窃取;速度快,大多数指令仅用1个晶振周期,而MCS-51单片机单周期指令也需12个晶振周期;能采用C语言编程,从而能高效快速地开发出目标产品;CMOS工艺生产,功耗低;有主电源3 V以下的品种,进一步降低功耗,一般只需几mA;还有多种低功耗方式,在掉电方式下,工作电流小于1 μA。

AVR单片机已形成系列产品,其中ATtiny,AT90及ATmega分别对应低、中、高档产品。根据用户的不同需要,现已推出了30多种型号,引脚为8～64脚,价格从几元到上百元人民币,内部配置也大不相同,但其基本结构和编程方法是一样的。

本书以AT90S8535单片机为主线讲述AVR单片机。AT90S8535单片机是AVR单片机中内部接口丰富、功能比较全、性能价格比高的品种,特点如下。

- AT90S8535片内有4 K字(8 KB)的Flash程序存储器,可擦写1 000次不损坏,且程序高度保密,避免非法窃取;
- 有512 B的SRAM;
- 有512 B的EEPROM(电擦写存储器),掉电不丢失信息,可在线擦写100 000次不损坏;
- 有32个I/O口,输入/输出的方向是可以定义的,输出口的驱动能力强,灌电流可达40 mA,能直接驱动LED、继电器等器件,省去驱动电路,输入口可以三态输入,也可带内部上拉电阻,省去外接上拉电阻;
- 有2个8位和1个16位的定时器/计数器,除定时计数功能外,有些还具有比较匹配输出和输入捕获功能;
- 有看门狗定时器,便于程序抗干扰,程序飞走进入死循环后,能自动复位重新启动;
- 有模拟比较器,便于发现输入模拟电压的变化;
- 有8路10位ADC,可直接输入模拟电压信号;
- 有2路10位和1路8位的PWM脉宽调制输出,经滤波输出模拟电压信号,可作为D/A转换器,这种模拟量输出很容易与主机隔离;
- 有UART异步串行接口,便于实现RS232-C和RS485通信接口;
- 有SPI同步串行接口;
- 有独立振荡器的实时时钟,在省电模式的低功耗方式下,时钟正常工作;
- 有16种中断源,每种中断源在程序空间都有一个独立的中断向量作相应的中断入口地址;
- 工作电压范围宽(2.7～6.0 V),抗电源波动能力强;
- 有商用级产品(工作温度0～70 ℃)和工业级产品(工作温度－40～85 ℃)供用户选用。

AT90S8535是AVR单片机中性能最强的品种之一。它与AT90S8515相比,增加了8路10位ADC;增加了一个可用异步时钟源的8位定时器/计数器,该定时器能用作实时时钟;增

加了一种省电低功耗方式，在此省电方式下，实时时钟照常运行；中断源由 12 个增加到 16 个。

学习了 AT90S8535 单片机，在今后的设计中，若其中某些功能用不到，可选用 ATtinyXX 或 AT90XXXXX 少引脚、低价格的品种，指令系统基本相同；有些型号少几条指令，可用别的指令代替，程序略做修改即可。

而若 AT90S8535 满足不了系统的要求，须用容量更大的 Flash 程序存储器、或容量更大的 SRAM、或容量更大的 EEPROM、或更多的 I/O 口，或者须用乘法指令进行快速计算等等，可改用 ATmegaXXX 单片机；但这些单片机只是内部资源有量的变化，质的变化不大，I/O 寄存器的访问和系统编程方法是一样的。有些品种多了几条指令，熟悉了 AT90S8535 单片机，使用这些单片机也是很容易的。

AT90S8535 功能比 AT90S8515 强，价格与 AT90S8515 相近，却比 ATmegaXXX 低得多；而且 AT90S8535 具有 SDIP 封装，便于学生做实验。所以我们以 AT90S8535 单片机为主线讲述 AVR 单片机。其他型号的单片机，查一下资料就可以举一反三了。

本书共分 16 章。第 1 章，单片机概述；第 2 章，8535 单片机系统结构概况；第 3 章，AVR 单片机指令系统；第 4 章，定点数运算程序设计及数制转换；第 5 章，浮点数运算程序设计；第 6 章，8535 单片机 EEPROM 读/写访问；第 7 章，8535 单片机 I/O 端口及其应用；第 8 章，中断系统及应用；第 9 章，8535 单片机定时器/计数器及其应用；第 10 章，8535 单片机模拟量输入接口；第 11 章，AVR 单片机串行接口及应用；第 12 章，AVR 单片机存储器编程；第 13 章，AVR 的 C 语言 IccAVR 及应用；第 14 章，AVR 单片机开发工具及应用；第 15 章，AVR 单片机的最新发展；第 16 章，整机设计中几个问题的处理方法。书中所举硬件设计、软件设计实例均经实验通过。

采用上述顺序，是因为考虑到学生的学习规律，使初学者一步步掌握这种新型单片机。本书可作为测控技术、自动化、计算机应用、通信工程等专业本科或研究生单片机课程教材。若学时紧张，第 5 章、第 13 章可不讲。本书也可作为相关技术人员的参考书。

本书由丁化成主编（E-mail：ding_new@sohu.com）。第 1，2，3，4，5，6，7，8，9，10，11，16 章由丁化成执笔；第 12，14，15 章由耿德根执笔；第 13 章由李君凯执笔。AVR 单片机开发实验设备由广州天河双龙电子有限公司提供。李青参加了第 4，5，6，13 章的编程实验工作，李立军参加了第 7，9，10，11 章的编程实验工作，以上程序均在双龙的 SL-AVRAD 开发实验器上验证通过。广州天河双龙电子有限公司提供 AVR 多媒体讲座及工作软件光盘，可作为本书的补充。

由于作者水平有限，加上时间仓促，书中错误疏漏之处，敬请读者批评指正。

编　者

2002 年 2 月 2 日于安徽工业大学

本书配套光盘的邮购方法

邮购地址：（邮编 510630）广州天河龙口西路龙苑大厦 A3 座新赛格电子城 331 室

邮购费：10 元（平寄）

联系人：耿德根（电话：020-85510191　E-mail：SLLG@SL.COM.CN）

目　录

第 12 章　AVR 单片机存储器编程

第 13 章　AVR 的 C 语言 ICCAVR 及应用

第 14 章　AVR 单片机开发工具及应用

第 1 章　单片机概述

1.1　单片机及其发展

所谓单片机，是指由一个芯片组成的微机系统。片内包括了 CPU、程序存储器、数据存储器、定时器/计数器及各种 I/O 口。

单片机又称微控制器(microcontroller)，主要用于现代智能化产品的设计。由 20 世纪 70 年代初至今，单片机已有 30 多年的发展历史了。随着产品智能化的需要和微电子技术的不断进步，目前已有上百家厂商生产几千种型号的单片机。单片机正向着高速度、低功耗、低成本、多档次、使用方便及外围接口丰富等方向发展。

1.2　单片机的应用领域

单片机具有极为广阔的应用前景，其主要应用领域可概括为以下几个方面。

1. 自动控制

单片机已在工业过程控制、机床控制、机器人控制、汽车控制以及飞行器制导系统等方面得到广泛的应用。由于单片机提供了串行口，很容易建立双机或多机之间的通信联系，从而为建立分布式控制系统创造了十分有利的条件。目前已有多种分布式控制系统用于工业控制之中。

2. 智能仪器仪表

由于单片机具有超微型化的特点，并且有无可比拟的高性能价格比，从而为仪器仪表的智能化提供了可能。

大量的智能仪器仪表应运而生，如智能化示波器、智能化数字仪表等。

3. 数据采集系统

由于单片机可提供多路 A/D 输入通道，因此很适用于模拟量(温度、压力及流量等)输入采样系统。

4. 计算机外设控制器

计算机的外部设备五花八门，随着单片机的发展，很多外部设备都使用单片机作为控制器，使这些外部设备智能化。

智能化键盘、智能化显示器、智能化打印机、智能化软盘和硬盘驱动器、智能化磁带驱动器及智能化绘图仪等，均可用单片机作为控制器。

5. 家用电器

家用电器更是单片机芯片生产厂家竞争非常激烈的应用领域。这个领域的应用特点是，量大、面广，并且要求价格低廉。

电饭锅、电子游戏机、电视机、录音机、组合音响、录像机、洗衣机、电冰箱以及电子玩具等，都使用单片机进行控制。

总之，单片机由于体积小、价格低、性能优越、可靠性高，已广泛地渗透到社会、生产、服务等领域，其应用前景是无限光明的；所以，工程技术人员有必要很好地掌握单片机原理及系统设计技术。

1.3 AVR 系列单片机简介

AVR 单片机是 ATMEL 公司 1997 年推出的全新配置精简指令集(RISC)单片机。

精简指令集 RISC 结构是 20 世纪 90 年代开发出来的，是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 AVR 单片机具有接近 1 MIPS/MHz 的高速处理能力。

为了加快进入市场和简化维护，对于单片机来说，用高级语言编程成为一种标准编程方法。

AVR 单片机的开发目的在于，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、性能及功耗进行优化，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，以实现累加器和存储器之间的数据传输；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输造成的瓶颈现象。

在 AVR 单片机中，运用 Harvard 结构，在前一条指令执行的时候，就取出现行的指令，然后以 1 个周期执行指令。

在其他的 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 1 个时钟周期执行 1 条指令的，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，因此程序存储器和数据存储器是分开的。可直接访问全部程序存储器和数据存储器，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，内载 Flash，EEPROM 及 SRAM 等不同用处的存储器。通过 SPI 口和一般的编程器，可对 AVR 单片机的 Flash 存储器进行编程。

AVR 单片机已形成系列产品，其中 ATtiny，AT90 及 ATmega 分别对应低、中、高档产品。根据用户的不同需要，现已推出了 30 多种型号，引脚为 8～64 脚，价格从几元到上百元人民币，内部配置也大不相同；但其基本结构和编程方法是一样的。表 1.1 为 AVR 系列单片机选型表。

表 1.1　AVR 单片机选型表(摘自 2002 年 ATMEL 网站)

	Flash /KB	EEPROM /B	RAM /B	Instructions	I/O Pins	Interrupts	Ext. Interrupts	SPI	UART	TWI[5]	Hardware Multiplier	8-bit Timer	16-bit Timer	PWM	Watchdog Timer	RTC Timer	Analog Comp	10-bit A/D Channels	On Chip Oscillator	Brown Out Detector	In System Programming	Self Program Memory	V_{cc} /V	Clock speed /MHz	Packages	Available
ATtiny11L	1	-	-	90	6	4	1[1]	-	-	-	-	1	-	-	Y	-	Y	-	Y	-	Y[3]	-	2.7～5.5	0～2	8-Pin DIP 8-Pin SOIC	now
ATtiny11	1	-	-	90	6	4	1[1]	-	-	-	-	1	-	-	Y	-	Y	-	Y	-	Y[3]	-	4.0～5.5	0～6	8-Pin DIP 8-Pin SOIC	now
ATtiny12V	1	64	-	90	6	5	1[1]	-	-	-	-	1	-	-	Y	-	Y	-	Y[2]	Y	Y	-	1.8～5.5	0～1	8-Pin DIP 8-Pin SOIC	now
ATtiny12L	1	64	-	90	6	5	1[1]	-	-	-	-	1	-	-	Y	-	Y	-	Y[2]	Y	Y	-	2.7～5.5	0～4	8-Pin DIP 8-Pin SOIC	now
ATtiny12	1	64	-	90	6	5	1[1]	-	-	-	-	1	-	-	Y	-	Y	-	Y[2]	Y	Y	-	4.0～5.5	0～8	8-Pin DIP 8-Pin SOIC	now
ATtiny15L	1	64	-	90	6	8	1[1]	-	-	-	-	2	-	1	Y[4]	-	Y	4	Y[2]	Y	Y	-	2.7～5.5	1～6	8-Pin DIP 8-Pin SOIC	now
ATtiny28V	2	-	-	90	20	5	2[1]	-	-	-	-	1	-	-	Y	-	Y	-	Y[2]	-	Y[3]	-	1.8～5.5	0～1	28-Pin DIP 32-Pin MLF 32-Pin TQFP	now
ATtiny28L	2	-	-	90	20	5	2[1]	-	-	-	-	1	-	-	Y	-	Y	-	Y[2]	-	Y[3]	-	2.7～5.5	0～4	28-Pin DIP 32-Pin MLF 32-Pin TQFP	now
AT90S1200	1	64	-	89	15	3	1	-	-	-	-	1	-	-	Y	-	Y	-	Y	-	Y	-	2.7～6.0	0～12	20-Pin DIP 20-Pin SOIC 20-Pin SSOP	now
AT90S2313	2	128	128	120	15	10	2	-	1	-	-	1	1	1	Y	-	Y	-	-	-	Y	-	2.7～6.0	0～10	20-Pin DIP 20-Pin SOIC	now
AT90LS2323	2	128	128	120	3	2	1	-	-	-	-	1	-	-	Y	-	-	-	-	-	Y	-	2.7～6.0	0～4	8-Pin DIP 8-Pin SOIC	now
AT90S2323	2	128	128	120	3	2	1	-	-	-	-	1	-	-	Y	-	-	-	-	-	Y	-	4.0～6.0	0～10	8-Pin DIP 8-Pin SOIC	now
AT90LS2343	2	128	128	120	4	2	1	-	-	-	-	1	-	-	Y	-	-	-	Y	-	Y	-	2.7～6.0	0～1	8-Pin DIP 8-Pin SOIC	now
AT90LS2343	2	128	128	120	4	2	1	-	-	-	-	1	-	-	Y	-	-	-	Y	-	Y	-	2.7～6.0	0～4	8-Pin DIP 8-Pin SOIC	now
AT90S2343	2	128	128	120	4	2	1	-	-	-	-	1	-	-	Y	-	-	-	Y	-	Y	-	4.0～6.0	0～10	8-Pin DIP 8-Pin SOIC	now
AT90LS4433	4	256	128	120	20	14	2	1	1	-	-	1	1	1	Y	-	Y	6	-	Y	Y	-	2.7～6.0	0～4	28-Pin DIP 32-Pin TQFP	now
AT90S4433	4	256	128	120	20	14	2	1	1	-	-	1	1	1	Y	-	Y	6	-	Y	Y	-	4.0～6.0	0～8	28-Pin DIP 32-Pin TQFP	now
AT90LS8515	8	512	512	120	32	11	2	1	1	-	-	1	1	2	Y	-	Y	-	-	-	Y	-	2.7～6.0	0～4	40-Pin DIP 44-Pin PLCC 44-Pin TQFP	now
AT90S8515	8	512	512	120	32	11	2	1	1	-	-	1	1	2	Y	-	Y	-	-	-	Y	-	4.0～6.0	0～8	40-Pin DIP 44-Pin PLCC 44-Pin TQFP	now

续表 1.1

	Flash /KB	EEPROM /B	RAM /B	Instructions	I/O Pins	Interrupts	Ext. Interrupts	SPI	UART	TWI[4]	Hardware Multiplier	8-bit Timer	16-bit Timer	PWM	Watchdog Timer	RTC Timer	Analog Comp.	10-bit A/D Channels	On Chip Oscillator	Brown Out Detector	In System Programming	Self Program Memory	V_{CC} /V	Clock speed /MHz	Packages	Available
AT90LS8535	8	512	512	120	32	15	2	1	1	-	-	2	1	3	Y	Y	Y	8	-	-	Y	-	2.7～6.0	0～4	40-Pin DIP 44-Pin PLCC 44-Pin TQFP	now
AT90S8535	8	512	512	120	32	15	2	1	1	-	-	2	1	3	Y	Y	Y	8	-	-	Y	-	4.0～6.0	0～8	40-Pin DIP 44-Pin PLCC 44-Pin TQFP	now
ATmega8L	8	512	1K	130	23	16	2	1	1[5]	1	Y	2	1	3	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7～5.5	0～8	28-Pin DIP 32-Pin MLF 32-Pin TQFP	QI02
ATmega8	8	512	1K	130	23	16	2	1	1[5]	1	Y	2	1	3	Y	Y	Y	8	Y[2]	Y	Y	Y	4.0～5.5	0～16	28-Pin DIP 32-Pin MLF 32-Pin TQFP	QI02
ATmega161L	16	512	1K	130	35	20	3	1	2	-	Y	2	1	4	Y	Y	Y	-	-	Y	Y	Y	2.7～5.5	0～4	40-Pin DIP 44-Pin TQFP	now
ATmega161	16	512	1K	130	35	20	3	1	2	-	Y	2	1	4	Y	Y	Y	-	-	Y	Y	Y	4.0～5.5	0～8	40-Pin DIP 44-Pin TQFP	now
ATmega163L	16	512	1K	130	32	17	2	1	1	1	Y	2	1	3	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7～5.5	0～4	40-Pin DIP 44-Pin TQFP	now
ATmega163	16	512	1K	130	32	17	2	1	1	1	Y	2	1	3	Y	Y	Y	8	Y[2]	Y	Y	Y	4.0～5.5	0～8	40-Pin DIP 44-Pin TQFP	now
ATmega16L	16	512	1K	130	32	17	3	1	1[5]	1	Y	2	1	3	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7～5.5	0～8	40-Pin DIP 44-Pin TQFP	QI02
ATmega16	16	512	1K	130	32	17	3	1	1[5]	1	Y	2	1	3	Y	Y	Y	8	Y[2]	Y	Y	Y	4.0～5.5	0～16	40-Pin DIP 44-Pin TQFP	QI02
ATmega323L	32	1K	2K	130	32	19	3	1	1	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7～5.5	0～4	40-Pin DIP 44-Pin TQFP	now
ATmega323	32	1K	2K	130	32	19	3	1	1	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	4.0～5.5	0～8	40-Pin DIP 44-Pin TQFP	now
ATmega103L	128	4K	4K	121	48	16	8	1	1	-	-	2	1	4	Y	Y	Y	8	-	-	Y	-	2.7～3.6	0～4	64-Pin TQFP	now
ATmega103	128	4K	4K	121	48	16	8	1	1	-	-	2	1	4	Y	Y	Y	8	-	-	Y	-	4.0～5.5	0～6	64-Pin TQFP	now
ATmega128L	128	4K	4K	133	48	27	8	1	2[5]	1	Y	2	2	6+2	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7～5.5	0～8	64-Pin TQFP	Eng. Samples Qt
ATmega128	128	4K	4K	133	48	27	8	1	2[5]	1	Y	2	2	6+2	Y	Y	Y	8	Y[2]	Y	Y	Y	4.0～5.5	0～16	64-Pin TQFP	Eng. Samples Qt

1 1个外部中断和唤醒引脚变化(所有 I/O 引脚)。

2 高精度(5%)可编程内部 RC 振荡器。

3 编程时需在 RESET 脚提供 12 V 信号。

4 兼容 I²C。

5 可编程串行 USART。

6 8个引脚可低电平中断。

1.4　AT90S8535 单片机的特点

AT90S8535 单片机是 AVR 单片机中内部接口丰富、功能比较全、性能价格比高的品种，特点如下。

(1) AT90S8535 片内有 4 K 字(8 KB)的 Flash 程序存储器。

程序存储器一次读取一个字(16 位)，速度加快了；可反复擦写、修改程序 1 000 次以上不损坏，便于新产品开发。

(2) 高速度。每个时钟周期执行一条指令，当主频 8 MHz 时，大多数指令仅需 125 ns。AVR 运用了 Harvard 结构概念，对程序和数据存储使用不同的存储器和总线，具有预取指令功能。当执行某一指令时，下一条指令被预先从程序存储器中取出，这样可以在每一个时钟周期内都执行指令。

(3) 高度保密性。可多次烧写的 Flash 具有多重密码保护、锁死(lock)功能。保密位在芯片底部，无法用电子显微镜看到。程序高度保密，避免非法窃取。

(4) 超功能精简指令。具有 32 个通用工作寄存器(均可作累加器，克服了单一累加器造成的瓶颈现象)及 512 字节的 SRAM，可灵活使用指令寻址运算。

(5) 低功耗。在主频 4 MHz，3 V 供电条件下，AT90LS8535 工作模式只需 6.4 mA 的供电电流，具有空闲、省电、掉电 3 种低功耗方式。掉电模式下工作电流小于 1 μA。

(6) 工作电压范围宽(2.7～6.0 V)，抗电源波动能力强。

(7) 有 512 B 的 EEPROM(电擦写存储器)，掉电不丢失信息，可在线改写。

(8) 有 32 个 I/O 口，输入/输出的方向是可以定义的。输出口的驱动能力强，灌电流可达 40 mA，能直接驱动 LED、继电器等器件，省去驱动电路；输入口可以三态输入，也可带内部上拉电阻，省去外接上拉电阻。

(9) 有 2 个 8 位和 1 个 16 位的定时器/计数器，除定时、计数功能外，有些还具有比较匹配输出和输入捕获功能。

(10) 有看门狗定时器，便于程序抗干扰。程序飞走进入死循环后，能自动复位，重新启动。

(11) 有模拟比较器，便于发现输入模拟电压的变化。

(12) 有 8 路 10 位 ADC，可直接输入模拟电压信号。

(13) 有 2 路 10 位和 1 路 8 位的 PWM 脉宽调制输出，经滤波输出模拟电压信号，可作为 D/A 转换器。这种模拟量输出很容易与主机隔离。

(14) 有 UART 异步串行接口，便于实现 RS232－C 和 RS485 通信接口。

(15) 有 SPI 同步串行接口。

(16) 有独立振荡器的实时时钟。在省电模式的低功耗方式下，时钟正常工作。

(17) 有 16 种中断源。每种中断源在程序空间都有一个独立的中断向量作相应的中断入口地址。

(18) 除用汇编语言外，还可使用 C 语言编程，易学、易写、易移植。

(19) 有商用级产品(工作温度 0～70 ℃)和工业级产品(工作温度 －40～85 ℃)供用户选用。

(20) 有 PDIP 40 脚、PLCC 44 脚及 TQFP 44 脚封装供用户选择。

1.5 以AT90S8535为主线讲述AVR单片机

AT90S8535是AVR单片机中档产品中性能最强的品种。它与AT90S8515相比，增加了8路10位ADC；增加了一个可用异步时钟源的8位定时器/计数器，该定时器能用作实时时钟；增加了一种省电低功耗方式，在此省电方式下，实时时钟照常运行；中断源由12个增加到16个。

学习了AT90S8535单片机，在今后的设计中，若其中某些功能用不到，可选用ATtinyXX或AT90XXXXX少引脚、低价格的品种，指令系统基本相同；有些型号少几条指令，可用别的指令代替，程序略做修改即可。

而若AT90S8535满足不了系统的要求，需要用容量更大的Flash程序存储器、或容量更大的SRAM、或容量更大的EEPROM、或更多的I/O口、或用乘法指令进行快速计算等等，可改用ATmegaXXX单片机；但这些单片机只是内部资源有量的变化，质的变化不大，I/O寄存器的访问和系统编程方法是一样的。有些品种多了几条指令，熟悉了AT90S8535单片机，使用这些单片机也是很容易的。

AT90S8535功能比AT90S8515强，价格与AT90S8515相近、比ATmegaXXX低得多；而且AT90S8535具有SDIP封装，便于学生做实验。

所以我们以AT90S8535单片机为主线讲述AVR单片机。其他型号的单片机，查一下资料就可以举一反三了。

第 2 章　8535 单片机系统结构概况

ATMEL 公司的 90 系列单片机是一种基于 AVR 增强性能、RISC 结构、低功耗、CMOS 技术、8 位微控制器(enhanced RISC microcontrollers)的单片机，通常简称为 AVR 单片机。目前有 AT90S1200，AT90S2313，AT90S2323，AT90S4433，AT90S8515，AT90S8535，ATmega8，ATmega16，ATmega32，ATmega128，ATmega103，ATtiny11，ATtiny12，ATtiny15，ATtiny28 等多种型号，它们的基本结构都比较相近。本章以 AT90S8535 单片机的内部结构为主，叙述 AVR 单片机的系统结构。

2.1　AVR 单片机 AT90S8535 的总体结构

2.1.1　AT90S8535 特点

(1) AVR RISC 结构。

(2) AVR 高性能低功耗 RISC 结构：

——118 条指令，大多数为单指令周期；

——32 个 8 位通用工作寄存器；

——工作在 8 MHz 时，具有 8 MIPS 的性能，即 125 ns 执行一条指令。

(3) 数据和非易失性程序内存：

——8 KB 的在线可编程 Flash(擦写次数 1 000 次)；

——512 B SRAM；

——512 B 在线可编程 EEPROM(擦写寿命 100 000 次)；

——程序加密位。

(4) 外围(peripheral)特点：

——2 个可预分频(prescale)的 8 位定时器/计数器，其中一个具有比较模式；

——1 个可预分频，具有比较、捕获功能和 2 个 8/9/10 位 PWM 功能的 16 位定时器/计数器；

——片内模拟比较器；

——可编程的看门狗定时器(由片内另一单独振荡器生成)；

——8 通道 10 位 ADC；

——全双工 UART。

(5) 特别的 MCU 特点：

——上电复位电路；

——具有计时功能，有独立振荡器的实时时钟(RTC)；

——低功耗空闲、省电及掉电模式；

内外部中断源。

(6) AT90LS8535 在 4 MHz,3 V,20 ℃条件下的功耗:

——工作模式,6.4 mA;

——空闲模式,1.9 mA;

——掉电模式,<1 μA。

(7) I/O 和封装:

32 个可编程的 I/O 脚;

——40 脚 PDIP、44 脚 PLCC 及 44 脚 TQFP 封装。

(8) 工作电压:

——2.7～6.0 V(AT90LS8535);

——4.0～6.0 V(AT90S8535)。

(9) 速　度:

——0～4 MHz(AT90LS8535);

——0～8 MHz(AT90S8535)。

2.1.2 描　述

AT90S8535 是一款基于 AVR RISC 的低功耗 CMOS 的 8 位单片机。

通过在 1 个时钟周期内执行 1 条指令,AT90S8535 可以取得接近 1 MIPS/MHz 的性能,从而使得设计人员可以在功耗和执行速度之间取得平衡。

图 2.1 为 AT90S8535 结构方框图。

AVR 核将 32 个工作寄存器和丰富的指令集联结在一起。

所有的工作寄存器都与 ALU(算逻单元)直接相连,允许在 1 个时钟周期内执行的单条指令同时访问 2 个独立的寄存器。这种结构提高了代码效率,使 AVR 得到了比普通 CISC 单片机高将近 10 倍的性能。

AT90S8535 具有以下特点:8 KB 的 Flash,512 B 的 EEPROM,512 B 的 SRAM,32 个通用 I/O 口,32 个通用工作寄存器,具有比较模式的灵活的定时器/计数器,内外中断源,可编程的 UART,可编程的看门狗定时器,SPI 口以及 3 种可通过软件选择的节电模式。

工作于空闲模式时,CPU 将停止运行,而寄存器、定时器/计数器、看门狗及中断系统继续工作;掉电模式时,振荡器停止工作,所有功能都被禁止,而寄存器内容得到保留,只有外部中断或硬件复位,才可以退出此状态;省电模式与掉电模式只有一点差别:省电模式下 T/C2 继续工作,以维持时间基准。

器件是以 ATMEL 的高密度非易失性内存技术生产的。

片内 Flash 可以通过 SPI 接口或通用编程器多次编程。

通过将增强的 RISC 8 位 CPU 与 Flash 集成在一个芯片内,AT90S8535 为许多嵌入式控制应用提供了灵活而低成本的方案。

AT90S8535 具有一整套的编程和系统开发工具:Studio 集成调试工具、宏汇编、软件模拟仿真器、在线实时仿真器、C 高级语言及 SL－AVRAD 开发实验器。

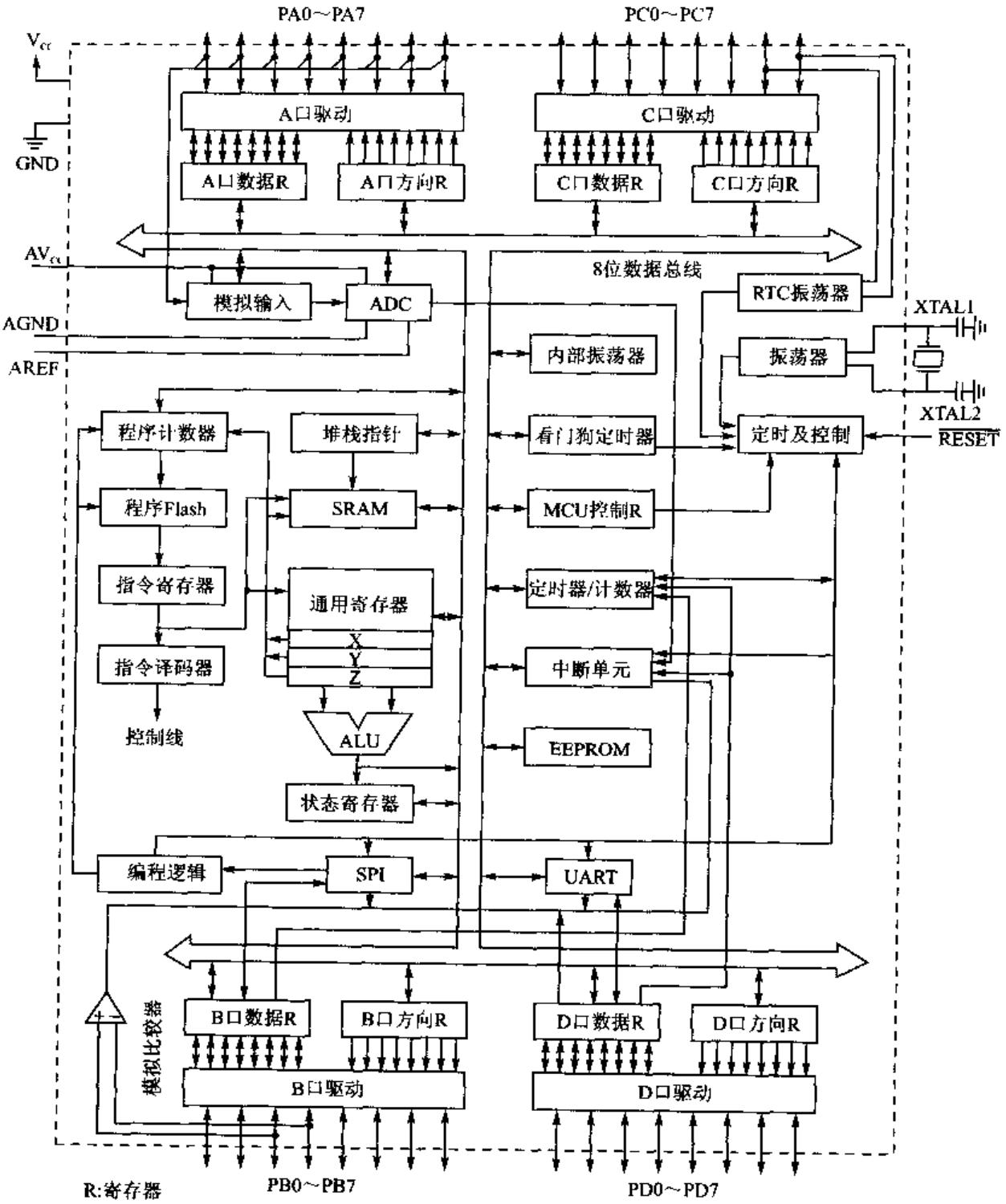


图 2.1　AT90S8535 结构方框图

2.1.3 引脚配置

图 2.2 为 AT90S8535 引脚配置图。

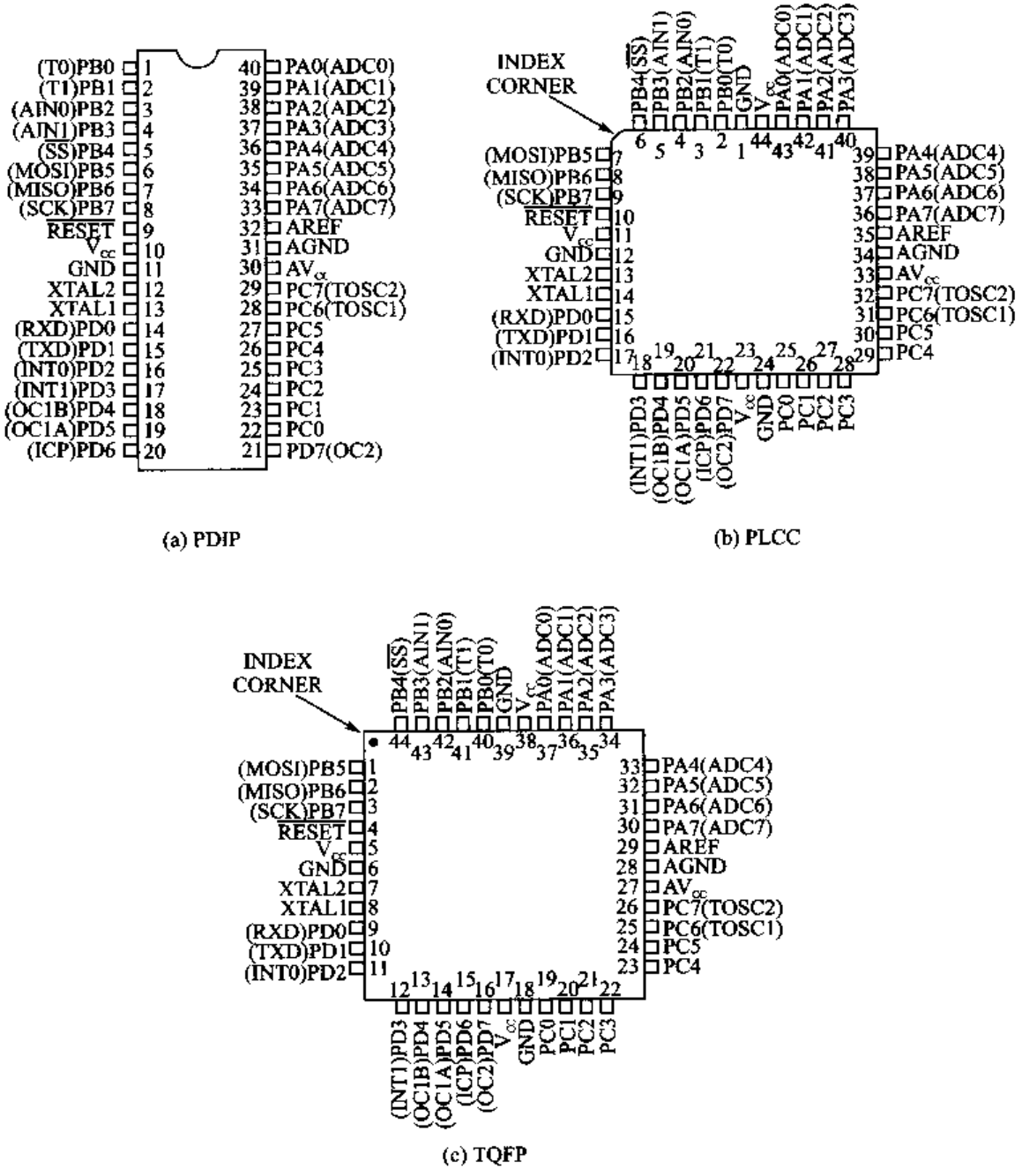


图 2.2 AT90S8535 引脚配置

2.1.4 引脚定义

(1) V_{CC}，GND：电源。

(2) A 口(PA7～PA0)：

A 口是一个 8 位双向 I/O 口，每一个引脚都有内部上拉电阻。A 口的输出缓冲器能够吸收 20 mA 的电流，可直接驱动 LED。当作为输入时，如果外部被拉低，由于上拉电阻的存在，

引脚将输出电流。在复位过程中，A 口为三态，即使此时时钟还未起振。

A 口还可以用作 ADC 的模拟输入口。

(3) B 口(PB7～PB0)：

B 口是一个 8 位双向 I/O 口，每一个引脚都有内部上拉电阻。B 口的输出缓冲器能够吸收 20 mA 的电流，可直接驱动 LED。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。在复位过程中，B 口为三态，即使此时时钟还未起振。

B 口作为特殊功能口的使用方法见以后章节。

(4) C 口(PC7～PC0)：

C 口是一个 8 位双向 I/O 口，每一个引脚都有内部上拉电阻。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。C 口的 2 个引脚还可以用作 T/C2 的振荡器。在复位过程中，C 口为三态，即使此时时钟还未起振。

(5) D 口(PD7～PD0)：

D 口是一个带内部上拉电阻的 8 位双向 I/O 口。输出缓冲器能够吸收 20 mA 的电流。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。在复位过程中，D 口为三态，即使此时时钟还未起振。

D 口作为特殊功能口的使用方法见以后章节。

(6) $\overline{\text{RESET}}$：复位输入。超过 50 ns 的低电平将引起系统复位；低于 50 ns 的脉冲不能保证可靠复位。

(7) XTAL1：振荡器放大器的输入端。

(8) XTAL2：振荡器放大器的输出端。

(9) AV_{CC}：A/D 转换器的电源，应通过一个低通滤波器与 V_{CC} 连接。

(10) AREF：A/D 转换器的参考电源，介于 AGND 与 AV_{CC} 之间。

(11) AGND：模拟地。

(12) 晶体振荡器：

XTAL1 和 XTAL2 分别是片内振荡器的输入、输出端，可使用晶体振荡器或陶瓷振荡器，如图 2.3 所示。

当使用外部时钟时，XTAL2 应悬空，如图 2.4 所示。

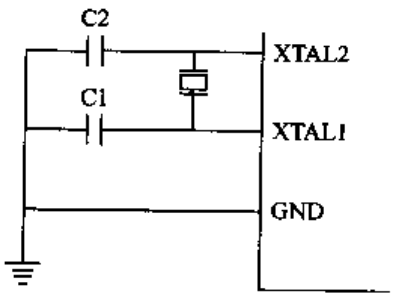


图 2.3　晶振连接

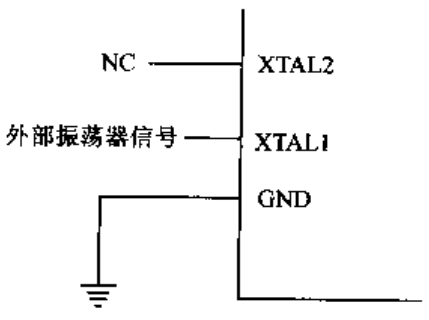


图 2.4　外部时钟驱动设置

(13) 实时时钟振荡器：

晶振可以直接连接到振荡器的引脚 PC6(TOSC1)和 PC7(TOSC2)而无需外部电容。

振荡器已经对 32 768 Hz 的晶振做了优化，对外加信号的带宽为 256 kHz。

2.2 AT90S8535 单片机的中央处理器 CPU

2.2.1 结构概述

快速访问寄存器文件包含 32 个 8 位可单周期访问的通用寄存器。这意味着在一个时钟周期内，ALU 可以完成一次如下操作：读取寄存器文件中的 2 个操作数，执行操作，将结果存回到寄存器文件。

寄存器文件中有 6 个可以组成 3 个 16 位用于数据寻址的间接寻址寄存器指针，以提高地址运算能力。其中 Z 指针还具有查表功能。

ALU 支持 2 个寄存器之间、寄存器和常数之间的算术和逻辑操作，以及单寄存器的操作。图 2.5 所示为 AT90S8535 单片机 AVR RISC 结构。

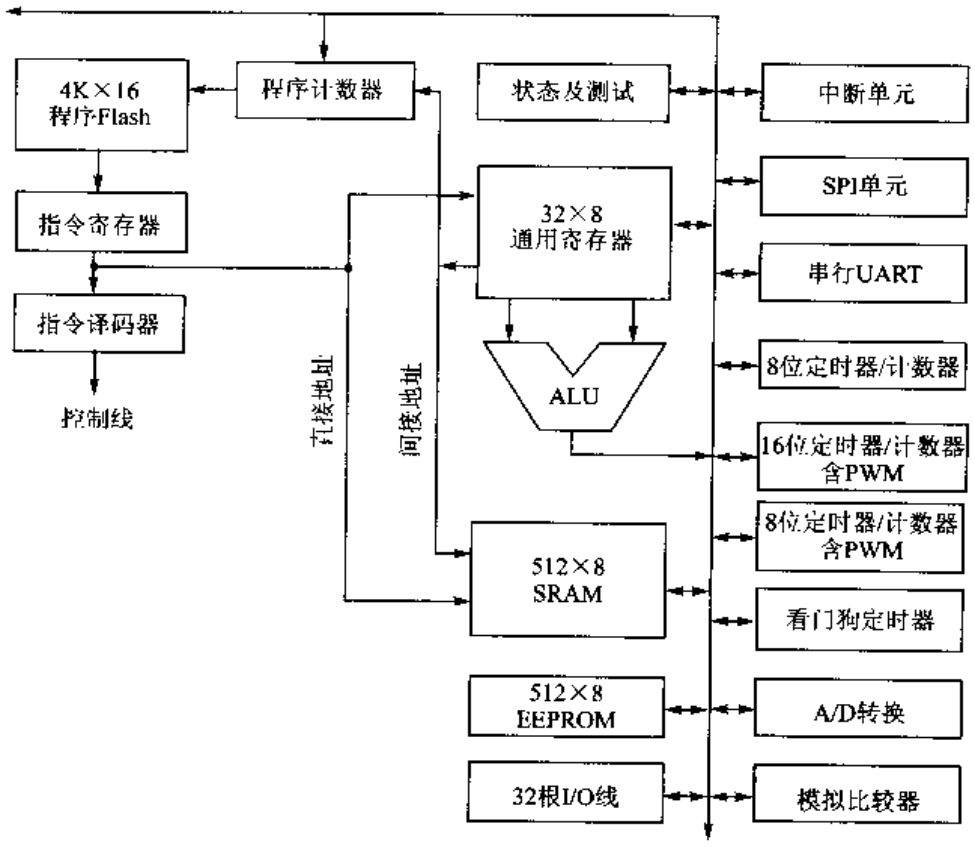


图 2.5 AT90S8535 AVR RISC 结构

除了寄存器操作模式，通常的内存访问模式也适用于寄存器文件。这是因为 AT90S8535 为寄存器文件分配了 32 个最低的数据空间地址（$00～$1F），允许其像普通内存地址一样访问。

I/O 内存空间包括 64 个地址，作为 CPU 外设的控制寄存器、T/C、A/D 转换器以及其他 I/O。I/O 内存可以直接访问，也可以作为数据地址（$20～$5F）访问。

AVR 采用了 Harvard 结构：程序和数据总线分离。程序内存通过 2 段式的管道（pipe-

line)进行访问:CPU 在执行一条指令的同时去取下一条指令。这种预取指的概念使得指令可以在一个时钟完成。

相对跳转和相对调用指令可以直接访问 4 K 字(8 KB)地址空间。多数 AVR 指令都为 16 位长。每个程序内存地址都包含一条 16 位或 32 位的指令。

当执行中断和子程序调用时,返回地址存储于堆栈中。堆栈分布于通用数据 SRAM 之中,堆栈大小只受 SRAM 数量的限制。堆栈放在 SRAM 最高处,进栈减 1(与 51 单片机进栈相反)。应在复位例程里就初始化 SP,SP 为可读/写的 16 位堆栈指针。

512 个 SRAM 可以通过 5 种不同的寻址方式很容易地进行访问。

AVR 结构的内存空间是线性的。

中断模块由 I/O 空间中控制寄存器和状态寄存器的全局中断使能位组成。每个中断都具有一个中断向量,由中断向量组成的中断向量表位于程序存储区的最前面。中断向量地址低的中断具有高的优先级。

2.2.2　通用工作寄存器文件

所有的寄存器操作指令都可以单指令的形式直接访问所有的寄存器。例外情况为 5 条涉及常数操作的指令:SBCI,SUBI,CPI,ANDI 及 ORI。这些指令只能访问通用寄存器文件的后半部分:R16～R31。

如图 2.6 所示,每个寄存器都有一个数据内存地址,它们被直接映射到用户数据空间的前 32 个地址。虽然寄存器文件的实现与 SRAM 不同,但是这种内存组织方式在访问寄存器方面具有极大的灵活性。图 2.7 所示为通用工作寄存器。

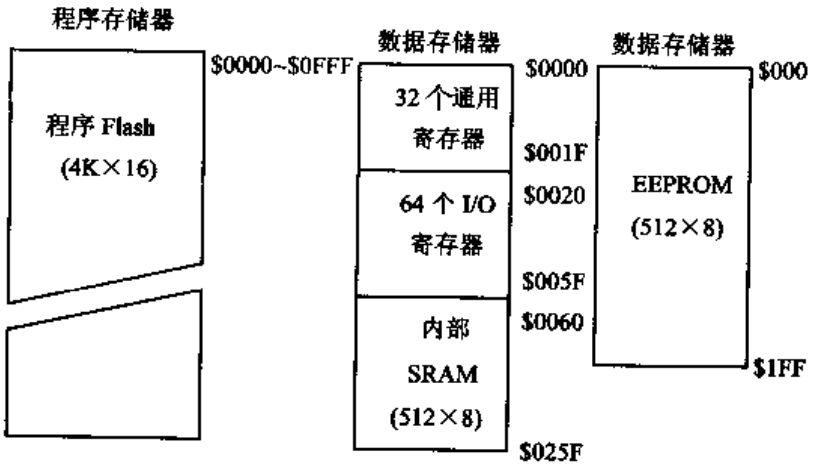


图 2.6　内存映像

图 2.7　通用工作寄存器

2.2.3　X,Y,Z 寄存器

寄存器 R26～R31 除了用作通用寄存器外,还可以作为数据间接寻址的地址指针,如图 2.8 所示。

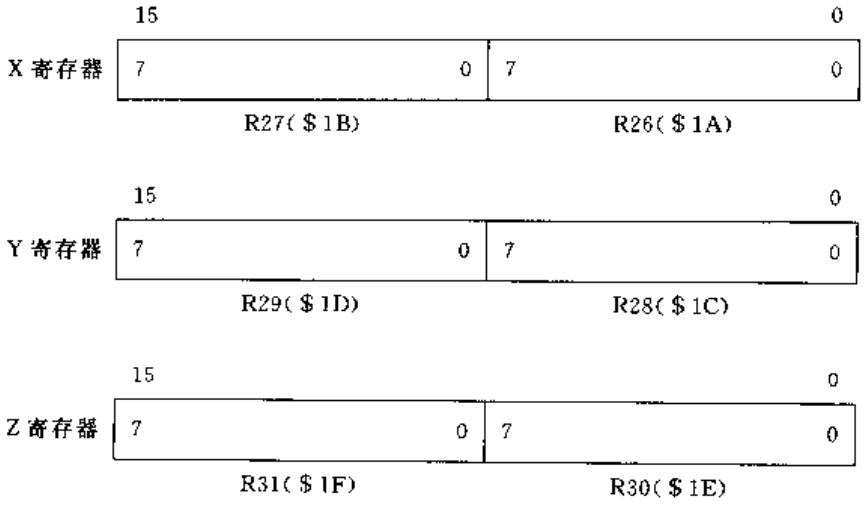


图 2.8 X,Y,Z 寄存器

2.2.4 ALU 运算逻辑单元

AVR ALU 与 32 个通用工作寄存器直接相连。ALU 操作分为 3 类:算术、逻辑及位操作。

2.3 AT90S8535 单片机存储器组织

2.3.1 在线可编程 Flash

AT90S8535 具有 8 KB 的 Flash。

因为所有的指令为 16 位宽,因此 Flash 结构为 4K×16。Flash 的擦写次数至少为 1 000 次。

AT90S8535 的程序计数器 PC 为 12 位宽,可以寻址到 4 096 个字的 Flash 程序区。

2.3.2 内部 SRAM 数据存储器

图 2.9 表明了 AT90S8535 的数据组织方式。

608 个数据地址用于寻址寄存器文件、I/O 寄存器及 SRAM。起始的 96 个地址为寄存器文件和 I/O 寄存器,其后的 512 个地址用于寻址 SRAM。

数据寻址模式分为 5 种:直接、带偏移量的间接、间接、预减的间接、后加的间接。寄存器 R26～R31 为间接寻址的指针寄存器。

直接寻址范围可达整个数据空间;带偏移量的间接寻址模式寻址到 Y,Z 指针给定地址附近的 63 个地址。

带预减和后加的间接寻址模式要用到 X,Y,Z 指针。

32 个通用寄存器、64 个 I/O 寄存器、512 B 的 SRAM 及最大可达 64 KB 的外部 SRAM 可以被所有的寻址模式访问。

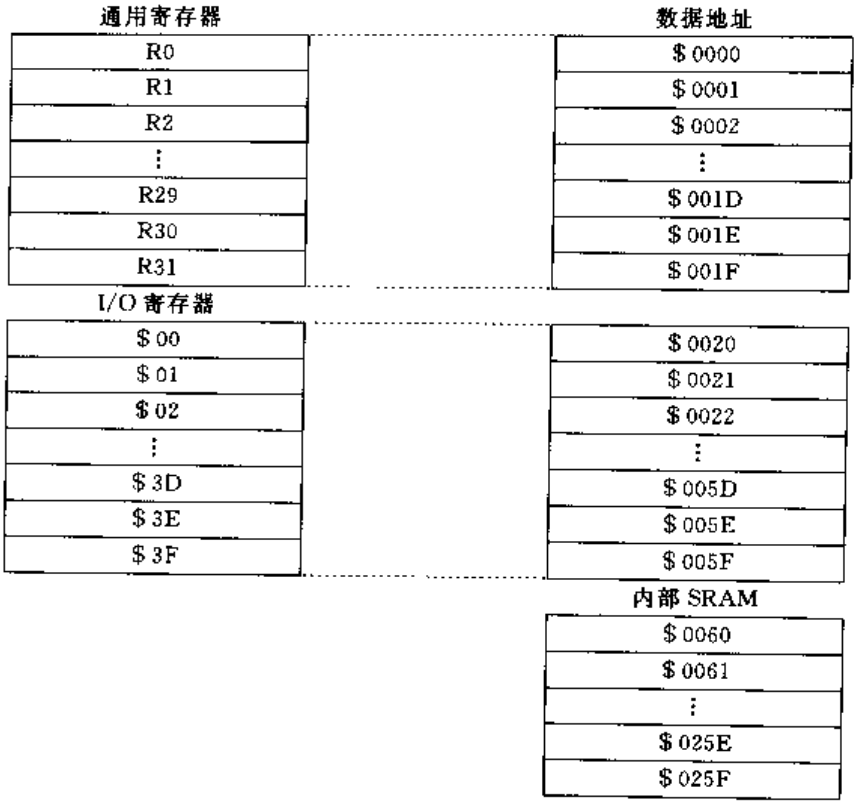


图 2.9　SRAM 分布

2.3.3　EEPROM 数据存储器

AT90S8535 包含 512 B 的 EEPROM。它是作为一个独立的数据空间而存在的，可以按字节读/写。EEPROM 的寿命至少 100 000 次(擦写)。EEPROM 的访问由地址寄存器、数据寄存器及控制寄存器决定。

2.3.4　I/O 寄存器

表 2.1 为 AT90S8535 的 I/O 空间。

表 2.1　AT90S8535 的 I/O 空间

I/O(数据)地址	名　称	功　能
$3F($5F)	SREG	状态寄存器
$3E($5E)	SPH	堆栈指针高字节
$3D($5D)	SPL	堆栈指针低字节
$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	T/C 中断屏蔽寄存器
$38($58)	TIFR	T/C 中断标志寄存器
$35($55)	MCUCR	MCU 控制寄存器
$34($54)	MCUSR	MCU 状态寄存器
$33($53)	TCCR0	T/C0 控制寄存器
$32($52)	TCNT0	T/C0 8 位计数器

续表 2.1

I/O(数据)地址	名　称	功　能
$2F($4F)	TCCR1A	T/C1 控制寄存器 A
$2E($4E)	TCCR1B	T/C1 控制寄存器 B
$2D($4D)	TCNT1H	T/C1 高字节
$2C($4C)	TCNT1L	T/C1 低字节
$2B($4B)	OCR1AH	T/C1 输出比较寄存器 A 高字节
$2A($4A)	OCR1AL	T/C1 输出比较寄存器 A 低字节
$29($49)	OCR1BH	T/C1 输出比较寄存器 B 高字节
$28($48)	OCR1BL	T/C1 输出比较寄存器 B 低字节
$27($47)	ICR1H	T/C1 输入捕获寄存器高字节
$26($46)	ICR1L	T/C1 输入捕获寄存器低字节
$25($45)	TCCR2	T/C2 控制寄存器
$24($44)	TCNT2	T/C2 8 位计数器
$23($43)	OCR2	T/C2 输出比较寄存器
$22($42)	ASSR	异步模式状态寄存器
$21($41)	WDTCR	看门狗控制寄存器
$1F($3F)	EEARH	EEPROM 高地址寄存器
$1E($3E)	EEARL	EEPROM 低地址寄存器
$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 口输入引脚
$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 口输入引脚
$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 口输入引脚
$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 口输入引脚
$0F($2F)	SPDR	SPI 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 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	模拟比较器控制和状态寄存器
$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 数据寄存器高字节
$04($24)	ADCL	ADC 数据寄存器低字节

AT90S8535 所有的外围 I/O 都被放置在 I/O 空间。IN 和 OUT 指令用来访问不同的 I/O 地址，以及在 32 个通用寄存器之间传输数据。地址为 $00～$1F 的 I/O 寄存器还可用 SBI 和 CBI 指令进行位寻址；而 SBIC 和 SBIS 则用来检查单个位置位与否。当使用 IN 和 OUT 指令时，地址必须在 $00～$3F 之间。如果要像 SRAM 一样访问 I/O 寄存器，则相应地址要加上 $20。在本文档里，所有 I/O 寄存器的 SRAM 地址写在括号中。

为了与后续产品兼容，保留未用的位应写“0”；而保留的 I/O 寄存器则不应写。

一些状态标志位的清除是通过写“1”实现的。CBI 和 SBI 指令读取已置位的标志位时，会回写“1”；因此会清除这些标志位。CBI 和 SBI 指令只对 $00～$1F 有效。

I/O 寄存器和外围 I/O 接口控制在后续章节介绍，这里只介绍以下 3 个 I/O 寄存器。

1. 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：$00

状态寄存器 SREG 的作用很大，各位代表不同含义。对各位的操作，有置 1 或清 0；反映运算、操作结果状态，有置 1、清 0、为 1 转移、为 0 转移等。状态寄存器 SREG 有 36 条指令供使用。

● 位 7——I：全局中断使能

置位时使全局中断使能。单独中断使能由不同的 I/O 寄存器（如 GIMSK，TIMSK 等）控制。如果全局中断位 I 被清 0，则不论单独中断标志置位与否，都不会产生中断。I 在复位时清 0，进入中断后 I 位也清 0，RETI（中断返回）指令执行后被置位。

● 位 6——T：位复制存储

位复制指令 BLD（SREG 中 T 标志到寄存器某位）和 BST（寄存器某位到 SREG 中 T 标志）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（低 4 位向高 4 位进位），请参考指令集说明。

● 位 4——S：标志位

S 位 N 和 V 的异或，请参考指令集说明。

● 位 3——V：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：0 值标志位

0 值标志位 Z 指示在不同的运算和逻辑操作之后的 0 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进位位,请参考指令集说明。

2. 堆栈指针——SP

在 I/O 地址 $3E($5E)和 $3D($5D)的 2 个 8 位寄存器构成了 90 系列单片机的 16 位堆栈指针。由于 AT90S8535 单片机 SRAM 的地址最高位为 $025F,又不能扩展外部 RAM,所以只用了 10 位。

	位 15	14	13	12	11	10	9	8	
$3E($5E)	—	—	—	—	—		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	
读/写:	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值:$00 00									

因为复位后堆栈为 SPH = $00,SPL = $00,AVR 单片机堆栈是减 1 或减 2 进栈,所以主程序一开始,堆栈指针必须设在 SRAM 最高处,如 AT90S8535 的堆栈设在 SRAM 的 $025F 处。堆栈有自动进栈(调用指令、中断指令)和人工进栈(压栈(PUSH)及出栈(POP))指令。

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触发之前被程序定义。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、将数据压入堆栈时,堆栈指针减少 2;当执行 POP 指令、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 RETI 返回、数据被从堆栈弹出时,堆钱指针增大 2。

2.4 AVR 单片机系统复位

复位,就是回到初始状态。复位后即 I/O 寄存器初始化、送规定的初始值;PC 指向 $000,程序从头开始。

2.4.1 复位源

90 系列单片机有 3 个复位源。

● 上电复位 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,且看门狗为触发时,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 地址中放置的指令需为某一 RJMP——相对转移指令,即到达复位处理路径的指令。若程序没有对中断源触发,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 2.10 的电路图说明了复位逻辑。表 2.2 定义了复位电路的时序和电参数。

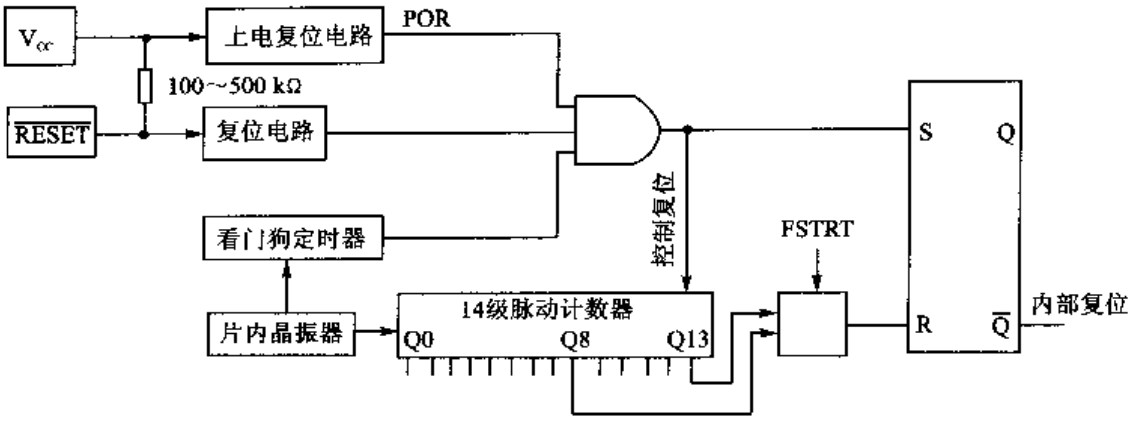


图 2.10　复位逻辑

表 2.2　复位特性(V_{CC}=5.0 V)

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

2.4.2　上电复位

上电复位(POR)保证器件在上电时正确复位。如图 2.11 所示，看门狗定时器驱动一个内部定时器，此定时器保证 MCU 只有在 V_{CC} 达到门限电压——V_{POT} 一定时间之后才启动。位于 Flash 内的 FSTRT 熔丝位编程后，可以使 MCU 以较短的时间启动(如图 2.12 所示)。如果使用了陶瓷谐振器或其他快速启动的振荡器，或内置于片内的启动时间足够的话，$\overline{RESET}$ 可以与 V_{CC} 直接相连，或是外接上拉电阻；如果在加上 V_{CC} 的同时保持 $\overline{RESET}$ 为低，则可以延长复位周期。请参考图 2.12 的时序例子。

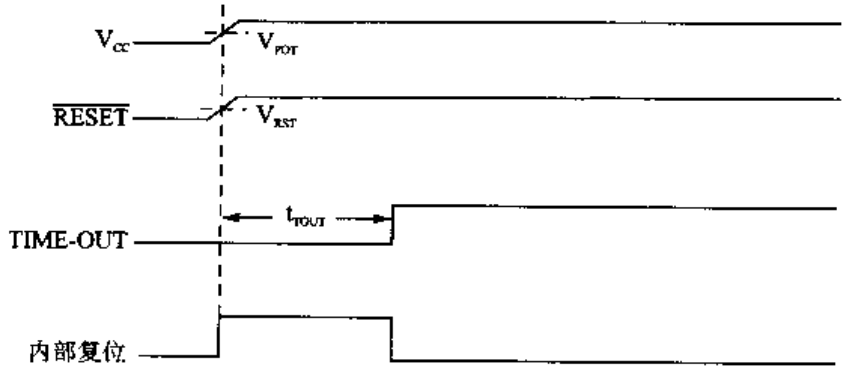


图 2.11　MCU 启动，$\overline{RESET}$ 连或不连到 V_{CC}，V_{CC} 快速上升

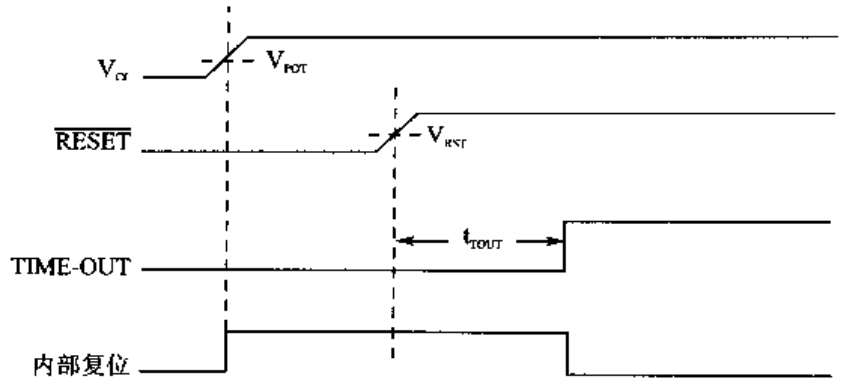


图 2.12 MCU 启动，RESET由外部控制

2.4.3 外部复位

RESET复位引脚上的低电压引发外部复位。大于 50 ns 的复位脉冲将造成芯片复位。施加短脉冲不能保证可靠复位。当外加信号达到复位门限电压 V_{RST}（上升沿）时，t_{TOUT} 延时周期开始，然后 MCU 启动，如图 2.13 所示。

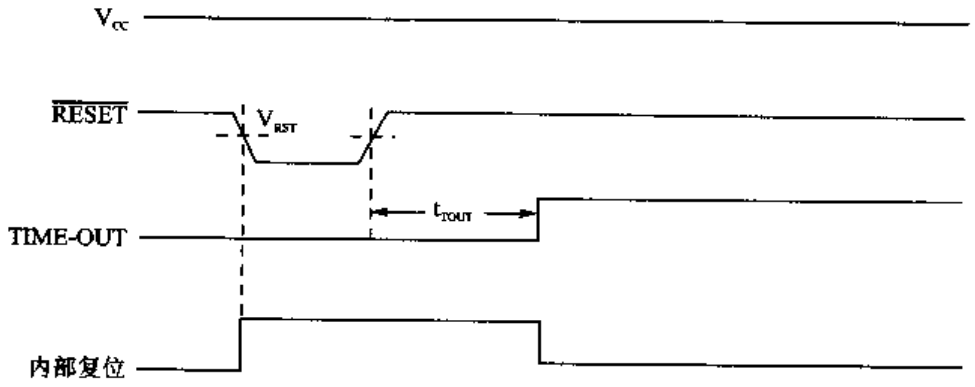


图 2.13 在工作期间由外部复位

2.4.4 看门狗复位

当看门狗定时器溢出时，将产生一个 XTAL 的短暂复位脉冲，在此脉冲的下降沿，延时定时器开始对超时时间 t_{TOUT} 计数。图 2.14 为在操作期间由看门狗复位的复位脉冲。

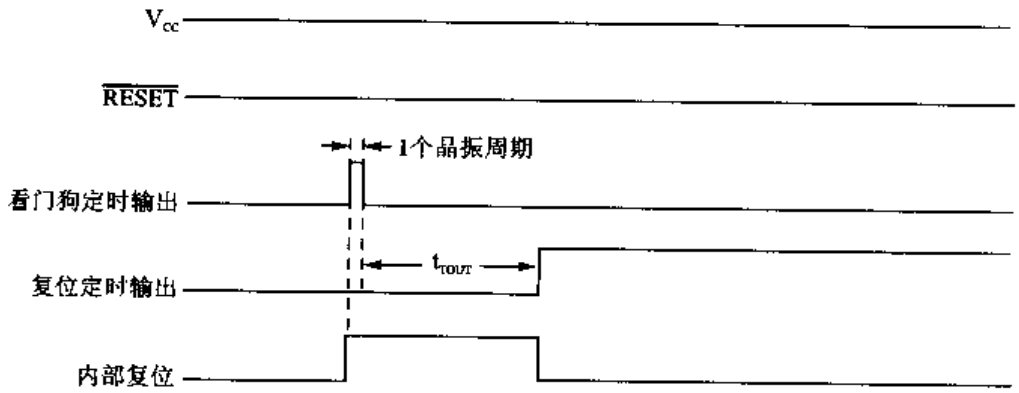
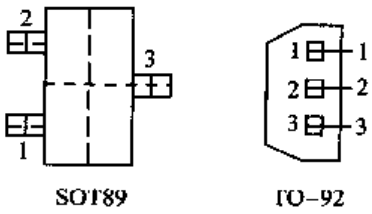


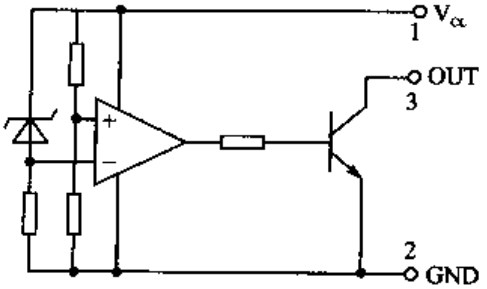
图 2.14 在工作期间由看门狗复位

使用 AVR 的 EEPROM 注意 3 点：① 避开 0 地址；② 程序在做初始化时，不要写 EEPROM，在进入读/写之前确保有数 ms 的延时，即等机器完全稳定以后，才能操作 EEPROM；③ 最安全的方法是加复位电压检测器，如图 2.15 所示。



1 V_{cc}；2 GND；3 OUT
MAX或IMP的809(高电平复位)
复位门限电平4.65 V

(a) 引脚图



器件名称	最小	典型	最大	单位
KIA7042AP/AF	4.05	4.2	4.35	V
KIA7045AP/AF	4.35	4.5	4.65	V

(b) 等价电路

图 2.15　复位检测器件引脚接线图

2.4.5　MCU 状态寄存器

MCU 状态寄存器定义如下：

	位 7	6	5	4	3	2	1	0	
$34($54)	—	—	—	—	—	—	EXTRF	PORF	MCUSR
读/写：	R	R	R	R	R	R	R/W	R/W	
初始化值：	0	0	0	0	0	0			

- 位 7～2——Res：保留位
- 位 1——EXTRF：外部复位标志

上电复位时没有定义(X)；外部复位时置位；看门狗复位对其没有影响。

- 位 0——PORF：上电复位标志

由上电复位置位。看门狗复位或外部复位对其没有影响。

上电复位、看门狗复位及外部复位对 PORF 和 EXTRF 的影响如表 2.3 所列。

表 2.3　复位后的 PORF 和 EXTRF

复位源	PORF	EXTRF
上电复位	1	没有定义
外部复位	不变化	1
看门狗复位	不变化	不变化

如果要利用 PORF 和 EXTRF 识别复位条件，软件要尽早对其清 0。检查 PORF 和 EXTRF 的语句在对其清 0 之前执行。如果某一位在外部复位或看门狗复位之前清 0，则复位可以通过表 2.4 找出来。

表 2.4 复位源鉴别

PORF	EXTRF	复位源
0	0	看门狗复位
0	1	外部复位
1	0	上电复位
1	1	上电复位

第 3 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、记忆及使用，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。

AVR 单片机目前用得比较多的器件有 118 条指令，这些器件是 AT90S2313，AT90S2323，AT90S2343，AT90S2333，AT90S4414，AT90S4433，AT90S4434，AT90S8515，AT90S8534，AT90S8535。AVR 大多数指令执行时间为单个时钟周期，本章主要分析 AVR 单片机指令系统的功能和使用方法。表 3.1 为常用 AVR 器件(118 条指令)指令表。

表 3.1　AT90S8535 指令速查表(118 条)

算术和逻辑运算指令(25 条)

指令	操作数	说　明	操　作	操作数范围	标　志	周 期
ADD	Rd,Rr	加法	Rd←Rd+ Rr	0≤d≤31	Z C N V H	1
ADC	Rd,Rr	带进位加	Rd←Rd+Rr+C	0≤d≤31	Z C N V H	1
ADIW	Rdl,K	加立即数	Rdl+1:Rdl←Rdl+1:Rdl+K	dl 取 24 26 28 30;0≤K≤63	Z C N V	2
INC	Rd	增 1	Rd←Rd+1	0≤d≤31	Z N V	1
SUB	Rd,Rr	减法	Rd←Rd−Rr	0≤d≤31	Z C N V H	1
SBC	Rd,Rr	带进位减	Rd←Rd−Rr−C	0≤d≤31	Z C N V H	1
SUBI	Rd,K	减立即数	Rd←Rd−K	16≤d≤31; 0≤K≤255	Z C N V H	1
SBCI	Rd,K	带进位减立即数	Rd←Rd−K−C	16≤d≤31; 0≤K≤255	Z C N V H	1
SBIW	Rdl,K	字减立即数	Rdl+1:Rdl←Rdl+1:Rdl−K	dl 取 24 26 28 30;0≤K≤63	Z C N V	2
DEC	Rd	减 1	Rd←Rd −1	0≤d≤31	Z N V	1
COM	Rd	取反	Rd← $FF−Rd	0≤d≤31	Z C_1 N V_0	1
NEG	Rd	取补	Rd← $00−Rd	0≤d≤31	Z C N V H	1
CP	Rd,Rr	比较	Rd−Rr	0≤d≤31	Z C N V H	1
CPC	Rd,Rr	带进位比较	Rd−Rr−C	0≤d≤31	Z C N V H	1
CPI	Rd,K	立即数比较	Rd−K	16≤d≤31; 0≤K≤255	Z C N V H	1
AND	Rd,Rr	逻辑与	Rd←Rd · Rr	0≤d≤31	Z N V_0	1
ANDI	Rd,K	与立即数	Rd←Rd · K	16≤d≤31; 0≤K≤255	Z N V_0	1
CBR	Rd,K	寄存器位清 0	Rd←Rd · ($FF−K)	16≤d≤31; 0≤K≤255	Z N V_0	1
TST	Rd	寄存器测试	Rd←Rd · Rd	0≤d≤31	Z N V_0	1
OR	Rd,Rr	逻辑或	Rd←Rd∨Rr	0≤d≤31	Z N V_0	1
ORI	Rd,K	或立即数	Rd←Rd∨K	16≤d≤31; 0≤K≤255	Z N V_0	1
SBR	Rd,K	寄存器位置位	Rd←Rd∨K	16≤d≤31; 0≤K≤255	Z N V_0	1

续表 3.1

算术和逻辑运算指令(25条)

指令 操作数	说 明	操 作	操作数范围	标 志	周 期
SER Rd	置位寄存器所有位	Rd←$FF	16≤d≤31		1
EOR Rd,Rr	异或	Rd←Rd⊕Rr	0≤d≤31	Z N V_0	1
CLR Rd	寄存器清0	Rd←Rd⊕Rd	0≤d≤31	Z_1 N_0 V_0	1

转移指令(31条)

指令 操作数	说 明	操 作	操作数范围	标 志	周 期
RJMP k	相对跳转	PC←PC+k+1	−2048≤k≤2047		2
IJMP	间接跳转	PC←(Z)	−2048≤k≤2047		2
BRBS s,k	SREG(s)位置1转移	若SREG(s)=1,PC←PC+k+1	0≤s≤7, −64≤k≤63		1或2
BRBC s,k	SREG(s)位清0转移	若SREG(s)=0,PC←PC+k+1	0≤s≤7, −64≤k≤63		1或2
BREQ k	相等转移	若Z=1, PC←PC+k+1	−64≤k≤63		1或2
BRNE k	不等转移	若Z=0, PC←PC+k+1	−64≤k≤63		1或2
BRCS k	C=1转移	若C=1, PC←PC+k+1	−64≤k≤63		1或2
BRCC k	C=0转移	若C=0, PC←PC+k+1	−64≤k≤63		1或2
BRSH k	大于或等于转移	若C=0, PC←PC+k+1	−64≤k≤63		1或2
BRLO k	小于转移	若C=1, PC←PC+k+1	−64≤k≤63		1或2
BRMI k	为负转移	若N=1, PC←PC+k+1	−64≤k≤63		1或2
BRPL k	为正转移	若N=0, PC←PC+k+1	−64≤k≤63		1或2
BRGE k	≥转移	若(N⊕V)=0,PC←PC+k+1	−64≤k≤63		1或2
BRLT k	<转移	若(N⊕V)=1,PC←PC+k+1	−64≤k≤63		1或2
BRHS k	H=1转移	若H=1, PC←PC+k+1	−64≤k≤63		1或2
BRHC k	H=0转移	若H=0, PC←PC+k+1	−64≤k≤63		1或2
BRTS k	T=1转移	若T=1, PC←PC+k+1	−64≤k≤63		1或2
BRTC k	T=0转移	若T=0, PC←PC+k+1	−64≤k≤63		1或2
BRVS k	V=1转移	若V=1, PC←PC+k+1	−64≤k≤63		1或2
BRVC k	V=0转移	若V=0, PC←PC+k+1	−64≤k≤63		1或2
BRIE k	I=1转移	若I=1, PC←PC+k+1	−64≤k≤63		1或2
BRID k	I=0转移	若I=0, PC←PC+k+1	−64≤k≤63		1或2
CPSE Rd,Rr	比较相等跳行	若Rd=Rr, PC←PC+2(或3)	0≤d≤31		1或2
SBRC Rr,b	寄存器位清0跳行	若Rd(b)=0, PC←PC+2(或3)	0≤b≤7		1或2
SBRS Rr,b	寄存器位置1跳行	若Rd(b)=1, PC←PC+2(或3)	0≤b≤7		1或2
SBIC P,b	I/O寄存器位清0跳行	若P(b)=0, PC←PC+2(或3)	0≤P≤31,0≤b≤7		2或3
SBIS P,b	I/O寄存器位置1跳行	若P(b)=1, PC←PC+2(或3)	0≤P≤31,0≤b≤7		2或3
RCALL k	相对调用	STACK←PC+1,PC←PC+k+1	−2048≤k≤2047		3
ICALL	间接调用	PC←(Z)			3
RET	从子程返回	PC←STACK			4
RETI	从中断返回	PC←STACK			4

续表 3.1

数据传送指令(31 条)

指令	操作数	说　明	操　作	操作数范围	标　志	周 期
MOV	Rd, Rr	寄存器传送	Rd←Rr	0≤d≤31		1
LDS	Rd, k	从 SRAM 中装入	Rd←(k)	0≤d≤31, 0≤k≤64 K		1
STS	k, Rr	数据送 SRAM	(k)←Rr	0≤k≤64 K		3
LDI	Rd, K	装入立即数	Rd←K	16≤d≤31, 0≤K≤255		2
LD	Rd, X	X 间址取数	Rd←(X)	0≤d≤31		2
LD	Rd, X+	X 间址取数后增 1	Rd←(X), X←X+1	0≤d≤31		2
LD	Rd, - X	X 减 1 后间址取数	X←X−1, Rd←(X)	0≤d≤31		2
ST	X,Rr	X 间址存数	(X)←Rr			2
ST	X+,Rr	X 间址存数后增 1	(X)←Rr, X←X+1			2
ST	−X, Rr	X 减 1 后间址存数	X←X−1, (X)←Rr			2
LD	Rd, Y	Y 间址取数	Rd←(Y)	0≤d≤31		2
LD	Rd, Y+	Y 间址取数后增 1	Rd←(Y), Y← Y+1	0≤d≤31		2
LD	Rd,−Y	Y 减 1 后间址取数	Y←Y−1, Rd←(Y)	0≤d≤31		2
LDD	Rd, Y+q	Y+q 变址取数	Rd←(Y+q)	0≤d≤31, 0≤q≤63		2
ST	Y,Rr	Y 间址存数	(Y)←Rr			2
ST	Y+, Rr	Y 间址存数后增 1	(Y)←Rr, Y←Y+1			2
ST	−Y, Rr	Y 减 1 后间址存数	Y←Y−1, (Y)←Rr			2
STD	Y+q, Rr	Y+q 变址存数	(Y+q)←Rr	0≤q≤63		2
LD	Rd, Z	Z 间址取数	Rd←(Z)	0≤d≤31		2
LD	Rd, Z+	Z 间址取数后增 1	Rd←(Z), Z←Z+1	0≤d≤31		2
LD	Rd,−Z	Z 减 1 后间址取数	Z←Z−1, Rd←(Z)	0≤d≤31		2
LDD	Rd, Z+q	Z+q 变址取数	Rd←(Z+q)	0≤d≤31, 0≤q≤63		2
ST	Z,Rr	Z 间址存数	(Z)←Rr			2
ST	Z+, Rr	Z 间址存数后增 1	(Z)←Rr, Z←Z+1			2
ST	−Z, Rr	Z 减 1 后间址存数	Z←Z−1, (Z)←Rr			2
STD	Z+q, Rr	Z+q 变址存数	(Z+q)←Rr	0≤q≤63		2
LPM		从程序区取数	R0←(Z)			3
IN	Rd, P	从 I/O 口取数	Rd←P	0≤d≤31,0≤P≤63		1
OUT	P,Rr	送 I/O 口	P←Rr	0≤P≤63		1
PUSH	Rr	压栈	STACK←Rr			2
POP	Rd	出栈	Rd←STACK	0≤d≤31		2

位指令和位测试指令(31 条)

指令	操作数	说　明	操　作	操作数范围	标　志	周 期
LSL	Rd	左移	C←b_7←b_6…b_1←b_0←0	0≤d≤31	Z C N V H	1
LSR	Rd	右移	0→b_7→b_6…b_1→b_0→C	0≤d≤31	Z C N V	1

续表 3.1

位指令和位测试指令(31 条)

指令	操作数	说 明	操 作	操作数范围	标 志	周 期
ROL	Rd	带 C 循环左移	C←b_7←b_6…b_1←b_0←C	0≤d≤31	Z C N V H	1
ROR	Rd	带 C 循环右移	C →b_7 →b_6…b_1→b_0→C	0≤d≤31	Z C N V	1
ASR	Rd	算术右移	b_7→b_7 →b_6…b_1→b_0→C	0≤d≤31	Z C N V	1
SWAP	Rd	半字节交换	b_7 b_6 b_5 b_4 ↔ b_3 b_2 b_1 b_0	0≤d≤31		1
BST	Rd, b	寄存器指定位送 T	T←Rd(b)	0≤d≤31, 0≤b≤7		1
BLD	Rd,b	T 送寄存器指定位	Rd(b)← T	0≤d≤31, 0≤b≤7		1
BSET	s	置 SREG 位	SREG(s)←1	0≤s≤7	相应位置 1	1
BCLR	s	清 SREG 位	SREG(s)←0	0≤s≤7	相应位清 0	1
SBI	P,b	置 I/O 位	I/O(P,b)←1	0≤P≤31, 0≤b≤7		2
CBI	P,b	清 I/O 位	I/O(P,b)← 0	0≤P≤31, 0≤b≤7		2
SEC		置 C	C←1		C_1	1
CLC		清 C	C←0		C_0	1
SEN		置 N	N←1		N_1	1
CLN		清 N	N←0		N_0	1
SEZ		置 Z	Z←1		Z_1	1
CLZ		清 Z	Z←0		Z_0	1
SEI		置 I	I←1		I_1	1
CLI		清 I	I←0		I_0	1
SES		置 S	S← 1		S_1	1
CLS		清 S	S←0		S_0	1
SEV		置 V	V←1		V_1	1
CLV		清 V	V←0		V_0	1
SET		置 T	T←1			1
CLT		清 T	T← 0			1
SEH		置 H	H←1		H_1	1
CLH		清 H	H←0		H_0	1
NOP		空操作				1
SLEEP		休眠				1
WDR		看门狗复位				1

注：① 源寄存器 Rn 均可为 R0～R31，表中操作数范围省略了 0≤n≤31。

② 表中标志是指对 Z,C,N,V,C 5 标志的影响，指令对其有影响的在表中列出来。
下标为 0 者执行指令后该标志清 0；下标为 1 者执行指令后该标志置 1；无下标者执行指令后该标志可能会变化，方向不定。

③ 影响 I,T 2 标志的指令很少，且明显，而 S=N ⊕ V,N 或 V 有一个变化 S 就变化；故表中未列对 I,T,S 3 标志的影响。

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。

1. 汇编语言源文件

汇编语言源文件包括指令助记符、标号及伪指令。

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、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 4 种形式：

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。

注释有下列形式：【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是被汇编器忽略的。

标号、指令及伪指令在后面有详细说明。

例：

```
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。

2. 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。

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)，也可能是常数表达式。

S：常数(0～7)，也可能是常数表达式。

p：常数(0～31/63)，也可能是常数表达式。

K：常数(0～255)，也可能是常数表达式。

k：常数值，范围取决于指令，也可能是常数表达式。

q：常数(0～63)，也可能是常数表达式。

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。全部伪指令如下：

(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 及 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有一个字定位计数器。ORG 伪指令用于放置代码段和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，每个表达式必须是一个 128～255 之间的有效值。如果表达式有效值是负数，则用 8 位二进制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 2 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，符号能在后面程序中再定义。

语法：.DEF 符号＝寄存器

(5) DEVICE——定义被汇编的器件

DEVICE 伪指令允许告知汇编器被执行的代码使用哪种器件。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|AT90S8535

(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。为了提供在 DW 伪指令前被保存的位置，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 位二进制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、地址及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、地址及操作代码文件列表。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 0；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、运算符及函数组成。表达式内部都为 32 位。

1. 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 0x0a，$0a，0xff，$0ff；

二进制数 0b00001010，0b11111111。

(5) PC：程序存储器定位计数器的当前值。

2. 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 2 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 3 个字节。

(5) BYTE4(表达式):返回一个表达式的第 4 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 的(表达式)次幂。

(10) LOG2(表达式):返回 LOG2(表达式)的整数部分。

3. 运算符

汇编器提供的部分运算符见表 3.2。优先级数越大的运算符,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 3.2　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;而表达式是 1 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,2 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,2 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,2 个表达式之间逐位与
17	逐位异或	^	7	二进制运算符,2 个表达式之间逐位异或
18	逐位或	|	6	二进制运算符,2 个表达式之间逐位或
19	逻辑与	&&	5	二进制运算符,2 个表达式之间逻辑与,非 0 则为 1;否则为 0
20	逻辑或	||	4	二进制运算符,2 个表达式之间逻辑或,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几种：单寄存器直接寻址、双寄存器直接寻址、I/O 寄存器直接寻址、数据存储器直接寻址、数据存储器间接寻址、带后增量的数据存储器间接寻址、带预减量的数据存储器间接寻址、带位移的数据存储器间接寻址、从程序存储器取常数寻址、程序间接寻址及程序相对寻址。

1. 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，机器码中给出该寄存器的直接地址。这种寻址方式称为单寄存器直接寻址。

例：INC Rd； 操作：Rn←Rn+1； 机器码：1001 010d dddd 0011。

可对 R0～R31 中任意寄存器增 1。不同寄存器只需改变机器码中 5 位地址码，如 INC R5，则机器码中的 5 个 d 取 00101，机器码即为 $9453。

2. 双寄存器直接寻址

由指令指出 2 个寄存器的内容作为操作数，机器码中给出该 2 个寄存器的直接地址。这种寻址方式称为双寄存器直接寻址。

例：ADD Rd，Rn； 操作：Rd←Rd+Rn； 机器码：0000 11rd dddd rrrr。

可对 R0～R31 中任意 2 个寄存器相加，这 2 个寄存器的各 5 位地址包含在机器码中。

3. I/O 寄存器直接寻址

指令可以直接对 I/O 寄存器进行操作，I/O 寄存器的直接地址包含在机器码中。

例：IN Rd，P； 操作：Rd←P； 机器码：1011 0ppd dddd pppp。

可把 64 个 I/O 寄存器中任意一个的内容输入到任一寄存器中，I/O 寄存器的 6 位地址和寄存器的 5 位地址都包含在机器码中。如 IN R5，$3E；(把 $3E 输入到 R5 中)机器码为 $B65E。

4. 数据存储器直接寻址

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，指令中高位字放 SRAM 存储器的 16 位地址。

例：LDS Rd，K； 操作：Rd←(K)； 机器码：1001 000d dddd 0000 KKKK KKKK KKKK KKK。

如 LDS R18，$100；(把 SRAM 地址为 $100 的内容传送到 R18 中)机器码为 $9120 0100

5. 数据存储器间接寻址

由指令指出某一个 16 位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，称为数据存储器间接寻址。操作数的地址在变址寄存器 X，Y 或 Z 中。

例：LD Rd，Y； 操作：Rd←(Y)，把以 Y 为指针的 SRAM 的内容送 Rd； 机器码：1000 000d dddd 1000。

如 LD R16,Y;(设 Y= $0567,即把 SRAM 地址为 $567 的内容传送到 R16 中)机器码为 $8108。

6. 带后增量的数据存储器间接寻址

类似数据存储器间接寻址方式;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。

例:LD Rd,Y+;　操作:Rd←(Y),Y=Y+1,先把以 Y 为指针的 SRAM 的内容送 Rd,再把 Y 增 1;　机器码:1001 000d dddd 1001。

如 LD R16,Y+;(设 Y= $0567,即把 SRAM 地址为 $567 的内容传送到 R16 中,且 Y= $0568)机器码为 $9109。

7. 带预减量的数据存储器间接寻址

类似数据存储器间接寻址;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。

例:LD Rd,−Y;　操作:Y=Y−1,Rd←(Y),先把 Y 减 1,再把以 Y 为指针的 SRAM 的内容送 Rd;　机器码:1001 000d dddd 1010

如 LD R16,−Y;(设 Y= $0567,即先把 Y 减 1,Y= $0566,再把 SRAM 地址为 $0566 的内容传送到 R16 中)机器码为 $910A。

8. 带位移的数据存储器间接寻址

带位移的数据存储器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 q 共同决定被存取 SRAM 存储器的地址。操作数的地址由 Y 或 Z 寄存器的内容加上机器码中给出的位移量 q。

例:LDD Rd,Y+q;　操作:Rd←(Y+q),其中 0≤q≤63,即把以 Y+q 为指针的 SRAM 的内容送 Rd,而 Y 寄存器的内容不变;　机器码:10q0 qq0d dddd 1qqq。

9. 从程序存储器取常数寻址

只有一条指令 LPM,也称查表指令。程序存储器中放常数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;而最低位 D0 用于存放字地址的高低字节。若最低位为 0,则选择低字节;若最低位为 1,则选择高字节。

例:LPM;　操作:R0←(Z),即把以 Z 为指针的程序存储器的内容送 R0。若 Z= $0100,即把地址为 $0080 的程序存储器的低字节内容送 R0;若 Z= $0101,即把地址为 $0080 的程序存储器的高字节内容送 R0。

10. 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,程序转到 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。

例:IJMP;　操作:PC←(Z),即把以 Z 为指针的 SRAM 的内容送程序计数器。若 Z= $0100,即把 $0100 送程序计数器,程序从 $0100 开始执行。

11. 程序相对寻址

程序相对寻址方式在机器码中包含了相对地址 k。

例:RJMP;　操作:PC←PC+1+k,−2048≤k≤2047,执行程序时,PC 首先自动加 1,再加 k。即执行该指令时,程序计数器自动加 1 后,再跳 k 步。

3.3 数据操作和指令类型

3.3.1 数据操作

AVR单片机是增强型RISC微控制器，具有高性能的数据处理能力，能对位、半字节、字节及双字节数据进行各种操作。它们包括算术和逻辑运算、数据传输、布尔处理及控制转移等操作。

3.3.2 指令类型

不同型号的AVR单片机指令数不同，有89～133条指令不等，AT90S2313/2323/2333/2343/4414/4433/4434/8515/8535等主要型号有118条指令。按功能分类则有25条算术和逻辑指令、31条转移指令、31条数据传输指令、31条位指令和位测试指令。先讲这118条指令。

3.3.3 指令集名词

1. 状态寄存器

SREG：状态寄存器。　　S：N⊕V符号测试位。

C：进位标志位。　　H：半进位标志位。

Z：0标志位。　　T：用于BLD和BST指令传送位。

N：负数标志位。　　I：全局中断触发禁止标志位。

V：2的补码溢出指示位。

2. 寄存器和操作码

Rd：寄存器区中的目的(或源)寄存器。

Rr：寄存器区中的源寄存器。

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：在寄存器区中或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. 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。

4. 标　志

⇔：由指令引起标志位双向变化时，写出该标志符号。

0：由指令引起标志位清0时，写出该标志符号，并带下标0。

1：由指令引起标志位置位时，写出该标志符号，并带下标1。

—：指令不引起某标志位变化时，不写出该标志符号。

3.4　算术和逻辑指令

AVR 的算术运算指令有加、减、取反、取补、比较、增量及减量指令。逻辑运算指令有与、或及异或指令等。

3.4.1　加法指令

1. 不带进位加法

ADD Rd,Rr　　0≤d≤31,0≤r≤31

说明:2 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　　机器码:0000 11rd dddd rrrr

对标志位的影响:H S V N Z C

例:add r16,r20;即 r16←r16+r20,机器码:$0f04

2. 带进位加法

ADC Rd,Rr　　0≤d≤31,0≤r≤31

说明:2 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　　机器码:0001 11rd dddd rrrr

对标志位的影响:H S V N Z C

例:求 r17:r16 与 r21:r20 两个双字节数之和。

```
add r16,r20
adc r17,r21
```

3. 加立即数

ADIW Rdl,K　　dl 为:24,26,28,30, 0≤K≤63

说明:寄存器对同立即数(0～63)相加,结果放到寄存器对。

操作:Rdh:Rdl←Rdh:Rdl+K　　机器码:1001 0110 KKdd KKKK

对标志位的影响:S V N Z C

例:adiw r24,$3F ;即 r25:r24←r25:r24+$3F。

注意:dl 只能取 24,26,28,30;K 为 6 位二进制无符号数(0～63)。

4. 增 1 指令

INC Rd　　0≤d≤31

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,所以允许 INC 指令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(相等转移)和 BRNE(不为 0 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　　机器码:1001 010d dddd 0011

对标志位的影响: S V N Z

例:inc r0;即将 r0 的值增 1。

3.4.2 减法指令

1. 不带进位减法

SUB Rd,Rr　　0≤d≤31,0≤r≤31

说明:2 个寄存器相减,结果送目的寄存器 Rd 中。

操作:Rd←Rd－Rr　　机器码:0001 10rd dddd rrrr

对标志位的影响:H S V N Z C

2. 立即数减法(字节)

SUBI Rd,K　　16≤d≤31,0≤K≤255

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～R31 之间,非常适合 X,Y 及 Z 指针的操作。

操作:Rd←Rd－K　　机器码:0101 KKKK dddd KKKK

对标志位的影响:H S V N Z C

3. 带进位减法

SBC Rd,Rr　　0≤d≤31,0≤r≤31

说明:2 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　　机器码:0000 10rd dddd rrrr

对标志位的影响:H S V N Z C

4. 带进位立即数减法

SBCI Rd,K　　16≤d≤31,0≤K≤255

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　　机器码:0100 KKKK dddd KKKK

对标志位的影响:H S V N Z C

5. 立即数减法(字)

SBIW Rdl,K　　dl 为 24,26,28,30, 0≤K≤63

说明:双寄存器与立即数 0～63 相减,结果送双寄存器。该指令仅用于最后 4 个寄存器对。

操作:Rdh:Rdl←Rdh:Rdl－K　　机器码:1001 0111 KKdd KKKK

对标志位的影响:S V N Z C

6. 减 1 指令

DEC Rd　　0≤d≤31

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,所以允许 DEC 指令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(不相等转移)和 BRNE(不为 0 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　　机器码:1001 010d dddd 1010

对标志位的影响:S V N Z

3.4.3　取反码指令

COM Rd　　0≤d≤31

说明：该指令完成寄存器 Rd 的二进制反码操作。

操作：Rd←$FF－Rd　　机器码：1001 010d dddd 0000

对标志位的影响：S V_0 N Z C_1

3.4.4　取补指令

NEG Rd　　0≤d≤31

说明：寄存器 Rd 的内容转换成二进制补码值。

操作：Rd←$00－Rd　　机器码：1001 010d dddd 0001

对标志位的影响：H S V N Z C

3.4.5　比较指令

1. 寄存器比较

CP Rd,Rr　　0≤d≤31,0≤r≤31

说明：该指令完成 2 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　　机器码：0001 01rd dddd rrrr

对标志位的影响：H S V N Z C

2. 带进位比较

CPC Rd,Rr　　0≤d≤31,0≤r≤31

说明：该指令完成寄存器 Rd 的值和寄存器 Rr 加 C 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　　机器码：0000 01rd dddd rrrr

对标志位的影响：H S V N Z C

3. 立即数比较

CPI Rd,K　　16≤d≤31,0≤K≤255

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　　机器码：0011 KKKK dddd KKKK

对标志位的影响：H S V N Z C

3.4.6　逻辑与指令

1. 寄存器逻辑与

AND Rd,Rr　　0≤d≤31,0≤r≤31

说明：寄存器 Rd 和寄存器 Rr 的内容逻辑与，结果送目的寄存器 Rd。

应用：清 0，使某位为 0，用 0 去与；保留，用 1 去逻辑与；代替硬件与门。

操作：Rd←Rd · Rr　　机器码：0010 00rd dddd rrrr

对标志位的影响：S V, N Z

2. 带立即数与

ANDI Rd,K　　16≤d≤31,0≤K≤255

说明：寄存器 Rd 的内容与常数逻辑与，结果送目的寄存器 Rd。

应用：清 0，使某位为 0，用 0 去与；保留，用 1 去逻辑与；代替硬件与门。

操作：Rd←Rd · K　　机器码：0111 KKKK dddd KKKK

对标志位的影响：S V, N Z

3. 清除寄存器位

CBR Rd,K　　16≤d≤31,0≤K≤255

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，其结果放在寄存器 Rd 中。

操作：Rd←Rd · ($FF−K)　　机器码：0111 KKKK dddd KKKK

对标志位的影响：S V, N Z

4. 测试 0 或负

TST Rd　　0≤d≤31

说明：测试寄存器是 0 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　　机器码：0010 00dd dddd dddd

对标志位的影响：S V, N Z

3.4.7 逻辑或指令

1. 寄存器逻辑或

OR Rd,Rr　　0≤d≤31,0≤r≤31

应用：置数，使某位为 1，用 1 去或；保留，用 0 去逻辑或；代替硬件或门。

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　　机器码：0010 10rd dddd rrrr

对标志位的影响：S V, N Z

2. 带立即数或

ORI Rd,K　　16≤d≤31,0≤K≤255

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　　机器码：0110 KKKK dddd KKKK

对标志位的影响：S V, N Z

3. 置寄存器位

SBR Rd,K　　16≤d≤31,0≤K≤255

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　　机器码：0110 KKKK dddd KKKK

对标志位的影响：S V, N Z

4. 置寄存器为 $FF

SER Rd　　16≤d≤31

说明：直接装入 $FF 到寄存器 Rd。

操作：Rd←$FF　　机器码：1110 1111 dddd 1111

对标志位的影响：无

3.4.8　逻辑异或指令

1. 寄存器异或

EOR Rd,Rr　　0≤d≤31,0≤r≤31

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　　机器码：0010 01rd dddd rrrr

对标志位的影响：S V_0 N Z

2. 清除寄存器

CLR Rd　　0≤d≤31

说明：寄存器清 0。该指令采用寄存器 Rd 与自己的内容相异或，实现寄存器的所有位都被清 0。

操作：Rd←Rd ⊕ Rd　　机器码：0010 01dd dddd dddd

对标志位的影响：S_0 V_0 N_0 Z_1

3.5　转移指令

3.5.1　无条件转移指令

1. 相对跳转

RJMP k　　−2K≤k≤2K

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，用目的地址的标号替代相对跳转字数 k。

操作：PC←(PC+1)+k　　机器码：1100 kkkk kkkk kkkk

对标志位的影响：无。

汇编语言中，只要欲转向的标号即可。

例：RJMP ABC

　　:

　　:

ABC:

2. 间接跳转

IJMP

说明：间接跳转到 Z 指针寄存器指向的 16 位地址。Z 指针寄存器是 16 位宽，允许在当前程序存储器空间 64 K 字(128 KB)内跳转。

IJMP 间接跳转的优点：转移范围大。缺点：作为子程序模块，移植时须修改转移地址。希望在子程序中不要使用，以免给自己带来麻烦！

操作：PC←Z(15～0)　　机器码：1001 0100 0000 1001

对标志位的影响：无。

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。

1. 测试条件符合转移指令

(1) 状态寄存器中位置位转移

BRBS s,k　　0≤s≤7,−64≤k≤63

说明：执行该指令时，PC 先加 1，再测试 SREG 的某一位。如果该位被置位，则跳转，跳转 k 个字，k 为 7 位带符号数。最多可向前跳 63 个字，向后跳 64 个字；否则顺序执行。在汇编程序中，用目的地址的标号替代相对跳转字数 k。

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

机器码：1111 00kk kkkk ksss。

对标志位的影响：无 。

(2) 状态寄存器中位清 0 转移

BRBC s,k　　0≤s≤7 ,−64≤k≤63

说明：执行该指令时，PC 先加 1，再测试 SREG 的某一位。如果该位被清 0，则跳转，跳转 k 个字，k 为 7 位带符号数。最多可向前跳 63 个字，向后跳 64 个字；否则顺序执行。在汇编程序中，用目的地址的标号替代相对跳转字数 k。

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

机器码：1111 01kk kkkk ksss。

对标志位的影响：无。

(3) 相等转移

BREQ k　　−64≤k≤63

说明：条件相对转移测试 0 标志 Z。如果 Z 位被置位，则相对 PC 值转移 k 个字。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 中数与寄存器 Rr 中数相等时，转移将发生。这条指令相当于指令 BRBS 1,k。

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

机器码：1111 00kk kkkk k001。

对标志位的影响：无。

(4) 不相等转移

BRNE k　　−64≤k≤63

说明：条件相对转移测试 0 标志 Z。如果 Z 位被清 0，则相对 PC 值转移 k 个字。这条指令相当于指令 BRBC 1,k。

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

机器码：1111 01kk kkkk k001。

对标志位的影响：无。

(5) C 标志位置位转移

BRCS k　　−64≤k≤63

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 k 个字。这条指令相当于指令 BRBS 0，k。

操作：If C=1，then PC←(PC+1)+k，else PC←PC+1。

机器码：1111 00kk kkkk k000。

对标志位的影响：无。

(6) C 标志位清除转移

BRCC k　　−64≤k≤63

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 k 个字。这条指令相当于指令 BRBC 0，k。

操作：If C=0，then PC←(PC+1)+k，else PC←PC+1。

机器码：1111 01kk kkkk k000。

对标志位的影响：无。

(7) 大于或等于转移(无符号数)

BRSH k　　−64≤k≤63

说明：条件相对转移测试进位标志 C。如果 C 位被清 0，则相对 PC 值转移 k 个字。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 中无符号二进制数大于等于寄存器 Rr 中无符号二进制数时，转移将发生。该指令相当于指令 BRBS 0，k。

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

机器码：1111 01kk kkkk k000。

对标志位的影响：无。

(8) 小于转移(无符号数)

BRLO k　　−64≤k≤63

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 k 个字。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 中无符号二进制数小于在寄存器 Rr 中无符号二进制数时，转移将发生。该指令相当于指令 BRBC 0，k。

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

机器码：1111 00kk kkkk k000。

对标志位的影响：无。

(9) 负数转移

BRMI k　　−64≤k≤63

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 k 个字。该指令相当于指令 BRBS 2，k。

操作：If N=1，then PC←(PC+1)+k，else PC←PC+1。

机器码：1111 00kk kkkk k010。

对标志位的影响：无。

(10) 正数转移

BRPL k　　−64≤k≤63

说明：条件相对转移测试负号标志 N。如果 N 被清 0，则相对 PC 值转移 k 个字。该指令相当于指令 BRBC 2，k。

操作：If N=0，then PC←(PC+1)+k，else PC←PC+1。

机器码：1111 01kk kkkk k010。

对标志位的影响：无。

(11) 大于或等于转移(带符号数)

BRGE k　　−64≤k≤63

说明：条件相对转移测试符号标志 S。如果 S 位被清 0，则相对 PC 值转移 k 个字。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。该指令相当于指令 BRBC 4，k。

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

机器码：1111 01kk kkkk k100。

对标志位的影响：无。

(12) 小于转移(带符号数)

BRLT k　　−64≤k≤63

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 k 个字。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。该指令相当于指令 BRBS 4，k。

操作：If Rd<Rf (N ⊕ V=1)，then PC←(PC+1)+k，else PC←PC+1。

机器码：1111 00kk kkkk k100。

对标志位的影响：无。

(13) 半进位标志置位转移

BRHS k　　−64≤k≤63

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 k 个字。该指令相当于指令 BRBS 5，k。

操作：If H=1，then PC←(PC+1)+k，else PC←PC+1。

机器码：1111 00kk kkkk k101。

对标志位的影响：无。

(14) 半进位标志清 0 转移

BRHC k　　−64≤k≤63

说明：条件相对转移测试半进位标志 H。如果 H 位被清 0，则相对 PC 值转移 k 个字。该指令相当于指令 BRBC 5，k。

操作：If H=0，then PC←(PC+1)+k，else PC←PC+1。

机器码：1111 01kk kkkk k101。

对标志位的影响：无。

(15) T 标志置位转移

BRTS k　　−64≤k≤63

说明：条件相对转移测试 T。如果标志 T 被置位，则相对 PC 值转移 k 个字。该指令相当于指令 BRBS 6，k。

操作：If T=1，then PC←(PC+1)+k，else PC←PC+1。

机器码：1111 00kk kkkk k110。

对标志位的影响：无。

(16) T 标志清 0 转移

BRTC k　　−64≤k≤63

说明：条件相对转移测试 T。如果标志 T 被清 0，则相对 PC 值转移 k 个字。该指令相当于指令 BRBC 6，k。

操作：If T=0，then PC←(PC+1)+k，else PC←PC+1。

机器码：1111 01kk kkkk k110。

对标志位的影响：无。

(17) 溢出标志置位转移

BRVS k　　−64≤k≤63

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 k 个字。该指令相当于指令 BRBS 3，k。

操作：If V=1，then PC←(PC+1)+k，else PC←PC+1。

机器码：1111 00kk kkkk k011。

对标志位的影响：无。

(18) 溢出标志清 0 转移

BRVC k　　−64≤k≤63

说明：条件相对转移测试溢出标志 V。如果 V 被清 0，则相对 PC 值转移 k 个字。该指令相当于指令 BRBC 3，k。

操作：If V=0，then PC←(PC+1)+k，else PC←PC+1。

机器码：1111 01kk kkkk k011。

对标志位的影响：无。

(19) 中断标志触发转移

BRIE k　　−64≤k≤63

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 k 个字。该指令相当于指令 BRBS 7，k。

操作：If I=1，then PC←(PC+1)+k，else PC←PC+1。

机器码：1111 00kk kkkk k111。

对标志位的影响：无。

(20) 中断标志禁止转移

BRID k　　−64≤k≤63

说明：条件相对转移测试全局中断标志 I。如果 I 被清 0，则相对 PC 值转移 k 个字。该指令相当于指令 BRBC 7，k。

操作：If I=0，then PC←(PC+1)+k，else PC←PC+1。

机器码：1111 01kk kkkk k111。

对标志位的影响：无。

2. 测试条件符合跳行转移指令

(21) 相等跳行

CPSE Rd，Rr　　0≤d≤31，0≤r≤31

说明：该指令完成 2 个寄存器 Rd 和 Rr 的比较。若 Rd=Rr，则跳一行执行指令。

操作：If Rd=Rr，then PC←PC+2(or3)，else PC←PC+1。

机器码：0001 00rd dddd rrrr。

对标志位的影响：无。

(22) 寄存器位清 0 跳行

SBRC Rr，b　　0≤r≤31，0≤b≤7

说明：该指令测试寄存器某位。如果该位被清 0，则跳一行执行指令。

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

机器码：1111 110r rrrr 0bbb。

对标志位的影响：无。

(23) 寄存器位置位跳行

SBRS Rr，b　　0≤r≤31，0≤b≤7

说明：该指令测试寄存器某位。如果该位被置位，则跳一行执行指令。

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

机器码：1111 111r rrrr 0bbb。

对标志位的影响：无。

(24) I/O 寄存器位清 0 跳行

SBIC P，b　　0≤P≤31，0≤b≤7

说明：该指令测试 I/O 寄存器某位。如果该位被清 0，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)=0，then PC←PC+2(or 3)，else PC←PC+1。

机器码：1001 1001 PPPP Pbbb。

对标志位的影响：无。

(25) I/O 寄存器位置位跳行

SBIS P，b　　0≤P≤31，0≤b≤7

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(or3)，else PC←PC+1。

机器码：1001 1011 PPPP Pbbb。

对标志位的影响：无。

3. 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子程序指令。调用子程序指令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；而转移指令既不保护断点，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，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保存和恢复 PC 内容的功能，即自动进栈，自动出栈。

(26) 相对调用

RCALL k　　$-2K \leqslant k \leqslant 2K$

说明：在 PC+ 1 后，前后 2K 字(4 KB)范围内调用子程序，返回地址(RCALL 后的指令地址)存储到堆栈。

操作：STACK←PC+1，PC←(PC+1)+k。机器码：1101 kkkk kkkk kkkk。

对标志位的影响：无。

(27) 间接调用

ICALL

说明：间接调用由寄存器区中的 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(128 KB)内的子程序。

操作：PC(15～0)←Z(15～0)。机器码：1001 0101 0000 1001。

对标志位的影响：无。

(28) 从子程序返回

RET

说明：从子程序返回，返回地址从堆栈中弹出。

操作：PC(15～0)←STACK。机器码：1001 0101 0000 1000。

对标志位的影响：无。

(29) 从中断程序返回

RETI

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。

注意：① 主程序应跳过中断区，防止修改补充中断程序带来麻烦；

② 不用的中断入口地址写上 RETI——中断返回，有抗干扰作用。

操作：PC(15～0)←STACK。机器码：1001 0101 0001 1000。

对标志位的影响：无。

3.6　数据传输指令

数据传输指令是在编程时使用最频繁的一类指令。数据传输指令是否灵活快速，对程序的执行速度产生很大影响。数据传输指令执行的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 端口之间的数据传输；另外还有从程序存储器直接取数指令 LPM、PUSH 压栈及 POP 出栈的堆栈指令。

3.6.1　直接寻址数据传输指令

(1) 寄存器间传输数据

MOV Rd，Rr　　$0 \leqslant d \leqslant 31, 0 \leqslant r \leqslant 31$

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。机器码：0010 11rd dddd rrrr。

所有的传输指令均对标志无影响。

(2) SRAM 数据直接送寄存器

LDS Rd,k　　0≤d≤31,0≤k≤65 535

说明：把 SRAM 中 1 个字节装入到寄存器，其中 k 为 SRAM 的 16 位地址。

操作：Rd←(k)。机器码：1001 000d dddd 0000 kkkk kkkk kkkk kkkk。

(3) 寄存器数据直接送 SRAM

STS k,Rr　　0≤r≤31,0≤k≤65 535

说明：将寄存器的内容直接存储到 SRAM，其中 k 为 SRAM 的 16 位地址。

操作：k←Rr。机器码：1001 001d dddd 0000 kkkk kkkk kkkk kkkk。

(4) 立即数送寄存器

LDI Rd,K　　16≤d≤31,0≤K≤255

说明：装入一个 8 位立即数到寄存器 R16～R31 中。

操作：Rd←K。机器码：1110 KKKK dddd KKKK。

3.6.2 间接寻址数据传输指令

1. 使用 X 指针寄存器间接寻址传输数据

(1) 使用 X 指针寄存器间接寻址将 SRAM 中内容装入到指定寄存器

① LD Rd,X　　0≤d≤31；将指针为 X 的 SRAM 中数送寄存器，指针不变。

操作：Rd←(X)。机器码：1001 000d dddd 1100。

② LD Rd,X+　　0≤d≤31；将指针为 X 的 SRAM 中数送寄存器，X 指针加 1。

操作：Rd←(X)，X←X+1。机器码：1001 000d dddd 1101。

③ LD Rd,－X　　0≤d≤31；X 指针减 1，将指针为 X 的 SRAM 中数送寄存器。

操作：X←X－1，Rd←(X)。机器码：1001 000d dddd 1110。

(2) 使用 X 指针寄存器间接寻址将寄存器内容存储到 SRAM

① ST X,Rr　　0≤r≤31；将寄存器内容送 X 为指针的 SRAM 中，X 指针不改变。

操作：(X)←Rr。机器码：1001 001r rrrr 1100。

② ST X+,Rr　　0≤r≤31；先将寄存器内容送 X 为指针的 SRAM 中，后 X 指针加 1。

操作：X←Rr，X←X+1。机器码：1001 001r rrrr 1101。

③ ST－X,Rr　　0≤r≤31；先 X 指针减 1，后将寄存器内容送 X 为指针的 SRAM 中。

操作：X←X－1，(X)←Rr。机器码：1001 001r rrrr 1110。

2. 使用 Y 指针寄存器间接寻址传输数据

(3) 使用 Y 指针寄存器间接寻址将 SRAM 中的内容装入寄存器

① LD Rd,Y　　0≤d≤31；将指针为 Y 的 SRAM 中数送寄存器，Y 指针不变。

操作：Rd←(Y)。机器码：1000 000d dddd 1000。

② LD Rd,Y+　　0≤d≤31；先将指针为 Y 的 SRAM 中数送寄存器，后 Y 指针加 1。

操作：Rd←(Y)，Y←Y+1。机器码：1001 000d dddd 1001。

③ LD Rd,－Y　　0≤d≤31；先 Y 指针减 1，后将指针为 Y 的 SRAM 中数送寄存器。

操作：Y←Y－1，Rd←(Y)。机器码：1001 000d dddd 1010。

④ LDD Rd Y+q　　0≤d≤31,0≤q≤63;将指针为 Y+q 的 SRAM 中数送寄存器,而 Y 指针不改变。

操作:Rd←(Y+q)。机器码:10q0 qq0d dddd 1qqq。

(4) 使用 Y 指针寄存器间接寻址将寄存器内容存储到 SRAM

① ST Y,Rr　　0≤r≤31;将寄存器内容送 Y 为指针的 SRAM 中,Y 指针不改变。

操作:Y←Rr。机器码:1000 001r rrrr 1000。

② ST Y+,Rr　　0≤r≤31;先将寄存器内容送 Y 为指针的 SRAM 中,后 Y 指针加 1。

操作:Y←Rr,Y←Y+1。机器码:1001 001r rrrr 1001。

③ ST－Y,Rr　　0≤r≤31;先 Y 指针减 1,后将寄存器内容送 Y 为指针的 SRAM 中。

操作:Y←Y－1,Y←Rr。机器码:1001 001r rrrr 1010。

④ STD Y+q,Rr　　0≤r≤31,0≤q≤63;将寄存器内容送 Y+q 为指针的 SRAM 中。

操作:(Y+q)←Rr。机器码:10q0 qq1r rrrr 1qqq。

3. 使用 Z 指针寄存器间接寻址传输数据

(5) 使用 Z 指针寄存器间接寻址将 SRAM 中内容装入到指定寄存器

① LD Rd,Z　　0≤d≤31 ;将指针为 Z 的 SRAM 中数送寄存器,Z 指针不变 。

操作:Rd←(Z)。机器码:1000 000d dddd 0000。

② LD Rd,Z+　　0≤d≤31;先将指针为 Z 的 SRAM 中数送寄存器,后 Z 指针加 1。

操作:Rd←(Z),Z←Z+1。机器码:1001 000d dddd 0001。

③ LD Rd,－Z　　0≤d≤31;先 Z 指针减 1,后将指针为 Z 的 SRAM 中数送寄存器。

操作:Z←Z－1,Rd←(Z)。机器码:1001 000d dddd 0010。

④ LDD Rd,Z+q　　0≤d≤31;将指针为 Z+q 的 SRAM 中数送寄存器,而 Z 指针不改变。

操作:Rd←(Z+q)。机器码:10q0 qq0d dddd 0qqq。

(6) 使用 Z 指针寄存器间接寻址将寄存器内容存储到 SRAM

① ST Z,Rr　　0≤r≤31;将寄存器内容送 Z 为指针的 SRAM 中,Z 指针不改变。

操作:Z←Rr。机器码:1000 001r rrrr 0000。

② ST Z+,Rr　　0≤r≤31;先将寄存器内容送 Z 为指针的 SRAM 中,后 Z 指针加 1。

操作:Z←Rr,Z←Z+1。机器码:1001 001r rrrr 0001。

③ ST－Z,Rr　　0≤r≤31;先 Z 指针减 1,后将寄存器内容送 Z 为指针的 SRAM 中。

操作:Z←Z－1,Z←Rr。机器码:1001 001r rrrr 0010。

④ STD Z+q,Rr　　0≤r≤31,0≤q≤63;将寄存器内容送 Z+q 为指针的 SRAM 中。

操作:(Z+q)←Rr。机器码:10q0 qq1r rrrr 0qqq。

操作以上 22 条指令后,X,Y,Z 指针寄存器要么不改变,要么加 1 或减 1。X,Y,Z 指针寄存器的这些特性特别适合用作访问矩阵表和堆栈指针等。

3.6.3　从程序存储器中取数装入寄存器指令

LPM

说明:将 Z 指向的一个字节装入 R0。由于程序存储器的地址是按字(双字节)排序的,Z(16 位)指针的高 15 位为程序存储器的地址。最低位 LSB 选择为 0 时,是指低字节;选择为 1

时，是指高字节。

该指令能寻址程序存储器为第1个64 KB(32 K字)。

操作：R0←(Z)。机器码：1001 0101 1100 1000。

例：取程序存储器地址为$100的低字节数和高字节数装入R16，R17。

```
        ldi r31,$02
        ldi r30,$00
        lpm
        mov r16,r0
        inc r30
        lpm
        mov r17,r0
h:      rjmp h
```

3.6.4 I/O口数据传输

(1) I/O口数据装入寄存器

IN Rd,P　　0≤d≤31,0≤P≤63

说明：将I/O空间(口、定时器、配置寄存器等)的数据传输到寄存器区中的寄存器Rd中。

操作：Rd←P。机器码：1011 0PPd dddd PPPP。

(2) 寄存器数据送I/O口

OUT P,Rr　　0≤r≤31,0≤P≤63

说明：将寄存器区中Rr的数据传输到I/O空间(口、定时器、配置寄存器等)。

操作：P←Rr。机器码：1011 1PPr rrrr PPPP。

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2条用于数据传输的栈操作指令。

(1) 进栈指令

PUSH Rr　　0≤r≤31,SP←SP+1

说明：该指令存储寄存器Rr的内容到堆栈。

操作：STACK←Rr。机器码：1001 001d dddd 1111。

(2) 出栈指令

POP Rd　　0≤d≤31,PC←PC+1

说明：该指令将堆栈中的字节装入到寄存器Rd中。

操作：Rd←STACK。机器码：1001 000d dddd 1111。

3.7 位指令和位测试指令

AVR单片机指令系统中有1/4的指令为位和位测试指令。这些指令的灵活应用，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) 寄存器逻辑左移

LSL Rd　　0≤d≤31

说明：寄存器 Rd 中所有位左移 1 位，第 0 位被清 0，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←b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0←0。机器码：0000 11dd dddd dddd。

对标志位的影响：H S V N Z C。

(2) 寄存器逻辑右移

LSR Rd　　0≤d≤31

说明：寄存器 Rd 中所有位右移 1 位，第 7 位被清 0，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：0→b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0→C。机器码：1001 010d dddd 0110。

对标志位的影响：S V N Z C。

(3) 寄存器通过进位左循环

ROL Rd　　0≤d≤31

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：C←b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0←C。机器码：0001 11dd dddd dddd。

对标志位的影响：H S V N Z C。

(4) 寄存器通过进位右循环

ROR Rd　　0≤d≤31

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→b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0→C。机器码：1001 010d dddd 0111。

对标志位的影响：S V N Z C。

(5) 寄存器算术右移

ASR Rd　　0≤d≤31

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：b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0→C。机器码：1001 010d dddd 0101。

对标志位的影响：S V N Z C。

(6) 寄存器半字节交换

SWAP Rd　　0≤d≤31

说明：寄存器中的高半字节和低半字节交换。

操作：b_7 b_6 b_5 b_4 ←→ b_3 b_2 b_1 b_0。机器码：1001 010d dddd 0010。

对标志位的影响：无。

3.7.2　位变量传输指令

(1) 寄存器中的位存储到 SREG 中的 T 标志。

BST Rd,b　　0≤d≤31，0≤b≤7

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)。机器码：1111 101d dddd 0bbb。

对标志位的影响：T。

(2) SREG 中的 T 标志装入寄存器中的某一位。

BLD Rd,b　　0≤d≤31，0≤b≤7

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。机器码：1111 100d dddd 0bbb。

对标志位的影响：无。

3.7.3 位变量修改指令

(1) 置状态寄存器的位

BSET s　　0≤s≤7

说明：置状态寄存器 SREG 的某一标志。

操作：SREG(s)←1。机器码：1001 0100 0sss 1000。

对标志位的影响：I T H S V N Z C。

(2) 清状态寄存器的位

BCLR s　　0≤s≤7

说明：清 0 SREG 状态寄存器中的一个标志位。

操作：SREG(s)←0。机器码：1001 0100 1sss 1000

对标志位的影响：I T H S V N Z C。

(3) 置 I/O 寄存器的位。

SBI P,b　　0≤P≤31,0≤b≤7

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。机器码：1001 1010 PPPP Pbbb。

对标志位的影响：无。

(4) 清 I/O 寄存器的位

CBI P,b　　0≤P≤31，0≤b≤7

说明：清 0 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。机器码：1001 1000 PPPP Pbbb。

对标志位的影响：无。

(5) 置进位位

SEC　　置位 SREG 状态寄存器中的进位标志 C。

操作：C←1。机器码：1001 0100 0000 1000。

对标志位的影响：C←1。

(6) 清进位位

CLC　　清 0 SREG 状态寄存器中的进位标志 C。

操作：C←0。机器码：1001 0100 1000 1000。

对标志位的影响：C←0。

(7) 置位负标志位

SEN　　置位 SREG 状态寄存器中的负数标志 N。

操作：N←1。机器码：1001 0100 0010 1000。

对标志位的影响：N←1。

(8) 清负标志位

CLN None PC←PC+1

说明：清 0 SREG 状态寄存器中的负数标志 N。

操作：N←0。机器码：1001 0100 1010 1000。

对标志位的影响：N←0。

(9) 置 0 标志位

SEZ　　置位 SREG 状态寄存器中的 0 标志 Z。

操作：Z←1。机器码：1001 0100 0001 1000。

对标志位的影响：Z←1。

(10) 清 0 标志位

CLZ　　清 0 SREG 状态寄存器中的 0 标志 Z。

操作：Z←0。机器码：1001 0100 1001 1000。

对标志位的影响：Z←0。

(11) 触发全局中断位

SEI　　置位 SREG 状态寄存器中的全局中断标志 I。

操作：I←1。机器码：1001 0100 0111 1000。

对标志位的影响：I←1。

(12) 禁止全局中断位

CLI　　清除 SREG 状态寄存器中的全局中断标志 I。

操作：I←0。机器码：1001 0100 1111 1000

对标志位的影响：I←0。

(13) 置 S 标志位

SES　　置位 SREG 状态寄存器中的符号标志 S。

操作：S←1。机器码：1001 0100 0100 1000。

对标志位的影响：S←1。

(14) 清 S 标志位

CLS　　清 0 SREG 状态寄存器中的符号标志 S。

操作：S←0。机器码：1001 0100 1100 1000。

对标志位的影响：S←0。

(15) 置溢出标志位

SEV　　置位 SREG 状态寄存器中的溢出标志 V。

操作：V←1。机器码：1001 0100 0011 1000。

对标志位的影响：V←1。

(16) 清溢出标志位

CLV　　清 0 SREG 状态寄存器中的溢出标志 V。

操作：V←0。机器码：1001 0100 1011 1000。

对标志位的影响：V←0。

(17) 置 T 标志位

SET　　置位 SREG 状态寄存器中的 T 标志。

操作：T←1。机器码：1001 0100 0110 1000。

对标志位的影响：T←1。

(18) 清 T 标志位

CLT　　清 0 SREG 状态寄存器中的 T 标志。

操作：T←0。机器码：1001 0100 1110 1000。

对标志位的影响：T←0。

(19) 置半进位标志

SEH　　置位 SREG 状态寄存器中的半进位标志 H。

操作：H←1。机器码：1001 0100 0101 1000。

对标志位的影响：H←1。

(20) 清半进位标志

CLH　　清 0 SREG 状态寄存器中的半进位标志 H。

操作：H←0。机器码：1001 0100 1101 1000。

对标志位的影响：H←0。

3.7.4　其他指令

(1) 空指令

NOP

说明：完成一个单周期空操作。

应用：延时等待；产生方波；抗干扰。在无程序单元写上空操作，空操作指令最后转到 $000H。

操作：无。

机器码：0000 0000 0000 0000。

对标志位的影响：无。

(2) 休眠指令

SLEEP

说明：设置电路休眠模式，由 MCU 控制寄存器定义。当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。

应用：省电，尤其对便携式仪器特别有用。

机器码：1001 0101 1000 1000。

对标志位的影响：无。

(3) 看门狗复位

WDR

说明：复位看门狗定时器。在 WD 预定比例器给出限定时间内，必须执行该指令。参见看门狗定时器硬件部分。

应用：抗干扰；延时。

操作：看门狗复位。

机器码：1001 0101 1010 1000。

对标志位的影响：无。

第 4 章　定点数运算程序设计及数制转换

定点数是小数点固定的数，分为整数、小数、混合小数等。小数或混合小数可以简化为整数乘以 10^{-n} 或 2^{-n} 来表示。例：十进制数 $12.56=1256*10^{-2}$，十六进制数 $1A.F8=1AF8*2^{-8}$。这样，可以将定点数先按整数运算，最后再考虑小数点的位置；所以下面所讲的程序主要是整数运算程序。另外，定点数又可分为无符号数和有符号数。无符号数是明确为正数的数，其符号省略了；有符号数可能是正数，也可能是负数。一般负数以补码表示，最高位为符号位。

4.1　加减运算程序

AT90S8535 单片机有加法和减法指令，可以直接调用相关指令来达到目的。这里列出了 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 位带立即数比较及 16 位取补程序。

```
add16:  add     r16,r18          ;r17:r16 + r19:r18→r17:r16
        adc     r17,r19
;*****************************************************************
addi16: subi    r16,low( -addi2)  ; r17:r16+addi2→r17:r16
        sbci    r17,high(-addi2)  ;addi2 为 16 位立即数
;*****************************************************************
sub16:  sub     r16,r18           ; r17:r16-r19:r18→r17:r16
        sbc     r17,r19
;*****************************************************************
subi16: subi    r16,low(subi2)    ; r17:r16-subi2→r17:r16
        sbci    r17,high(subi2)   ;subi2 为 16 位立即数
;*****************************************************************
cp16:   cp      r16,r18           ;r17:r16 与 r19:r18 相比较
        cpc     r17,r19
;*****************************************************************
cpi16:  cpi     r16,low(cp2)      ; r17:r16 与 16 位立即数 cp2 相比较
        ldi     r18, high(cp2)
        cpc     r17,r18
;*****************************************************************
beg16:  com     r16               ;r17:r16 取补并回存
        com     r17
        subi    r16,low(-1)
        sbci    r17,high(-1)
;*****************************************************************
```

32 位运算与 16 位运算相似。例:32 位加法程序和 32 位减法程序如下。

```
ADD32:                  ;32 位加法程序
      ADD R16,R20
      ADC R17,R21
      ADC R18,R22
      ADC R19,R23
;*****************************************************************
SUB32:                  ;32 位减法程序
      SUB R16,R20
      SBC R17,R21
      SBC R18,R22
      SBC R19,R23
```

4.2　乘除运算子程序

乘除运算中列出了 8 位、16 位、32 位有或无符号数的运算子程。由于 AT90S8535 单片机没有乘法和除法指令,其无符号二进制数乘法采用移位加法实现,除法采用移位减法实现;而对有符号数则对符号位单独考虑。

对子无符号数定点乘、除法,整数和小数没有区别;对于有符号数,还应对结果进行进一步的处理。

4.2.1　乘法运算子程序

列出了“mpy8u”——8 位 * 8 位无符号乘法,“mpy8s”——8 位 * 8 位带符号乘法,“mpy16u”——16 位 * 16 位无符号乘法,“mpy16s”——16 位 * 16 位带符号乘法,“mpy32u”——32 位 * 32 位无符号乘法,“mpy32s”——32 位 * 32 位带符号乘法子程序。

1. “mpy8u”——8 位 * 8 位无符号乘法

(1) 程序功能:r16 (被乘数) * r17(乘数)→r18:r17(结果)。

该子程序是将 2 个寄存器 r16 和 r17 相乘,结果送至寄存器 r18 和 r17。

(2) 程序清单:

```
mpy8u:                  ;8 位 * 8 位无符号乘法
    clr r18             ;清结果高字节
    ldi r19,8           ;初始化循环计数器
    lsr r17             ;乘数循环
m8u_1:
    brcc    m8u_2       ;进位置位
    add r18,r16         ;加被乘数到结果高字节
m8u_2:
    ror r18             ;结果高字节右循环
    ror r17             ;结果低字节和乘数右循环
    dec r19             ;循环计数器减 1
```

```
    brne    m8u_1       ;如没完成,再循环
    ret
```

2. "mpy8s"——8 位 * 8 位带符号乘法

(1) 程序功能：r16（被乘数）* r17(乘数)→r18:r17(结果)。

该子程序是将 2 个寄存器 r16 和 r17 相乘，结果送至寄存器 r18 和 r17。r18 最高位为符号位：为 0，代表正；为 1，代表负。结果为 16 位。该程序采用 Booth 算法。

(2) 程序清单：

```
mpy8s:                  ;8 位 * 8 位带符号乘法
    sub r18,r18         ;清除结果高字节和进位位
    ldi r19,8           ;初始化循环计数器
m8s_1:
    brcc    m8s_2       ;如果进位位置位
    add r18,r16         ;加被乘数到结果高字节
m8s_2:
    sbrc r17,0          ;如果不够减
    sub r18,r16         ;再从结果高字节减去被乘数
    asr r18             ;算术右移结果高字节
    ror r17             ;右移结果低字节和乘数
    dec r19             ;循环计数器减 1
    brne    m8s_1       ;如没完成,再循环
    ret
```

3. "mpy16u"——16 * 16 位无符号乘法

(1) 程序功能：(r17:r16) * (r19:r18)→r21:r20:r19:r18。

该子程序是将寄存器 r17 和 r16 的值与寄存器 r19 和 r18 的值相乘，结果送至寄存器 r21,r20,r19 及 r18。结果为 32 位。

(2) 程序清单：

```
mpy16u:                 ;16 * 16 位无符号乘法
    clr r21             ;清结果高 2 字节
    clr r20
    ldi r22,16          ;初始化循环计数器
    lsr r19             ;右移乘数
    ror r18
m16u_1:
    brcc    noad8       ;进位位为 0,跳至 noad8
    add r20,r16         ;进位位为 1
    adc r21,r17         ;结果加被乘数
noad8:
    ror r21             ;结果和乘数右移
    ror r20
    ror r19
```

```
    ror r18
    dec r22                    ;减循环计数器
    brne        m16u_1         ;如果没完成,再循环
    ret
```

4. "mpy16s"——16 * 16 位带符号乘法

(1) 程序功能：(r17:r16) * (r19:r18)→r21:r20:r19:r18。

该子程序是将寄存器 r17 和 r16 的值与寄存器 r19 和 r18 的值相乘，结果送至寄存器 r21,r20,r19 及 r18。寄存器 r21 最高位为 0 代表正数，为 1 代表负数。结果为 32 位。该程序采用 Booth 算法。

(2) 程序清单：

```
mpy16s:                        ;16 位 * 16 位带符号乘法
    clr r21                    ;清结果和进位位
    sub r20,r20
    ldi r22,16                 ;初始化循环计数器
m16s_1:
    brcc        m16s_2         ;进位位为 0,跳至 m16s-2
    add r20,r16                ;进位位为 1
    adc r21,r17                ;结果 3 字节、2 字节加被乘数
m16s_2:
    sbrc r18,0                 ;如果当前位置位
    sub r20,r16                ;从结果字节 2 中减被乘数低字节
    sbrc r18,0                 ;如果当前位置位
    sbc r21,r17                ;从结果字节 2 中减被乘数低字节
    asr r21                    ;算术右移结果和乘数
    ror r20
    ror r19
    ror r18
    dec r22                    ;循环计数器减 1
    brne        m16s_1         ;如没完成,再循环
    ret
```

5. "mpy32u"——32 位 * 32 位无符号乘法

(1) 程序功能：(r19:r18:r17:r16) * (r23:r22:r21:r20)，结果为 64 位，存放于 r27:r26:r25:r24:r23:r22:r21:r20。

(2) 程序清单：

```
mpy32u:                        ;32 位 * 32 位无符号乘法
    ldi r28,32                 ;初始化循环计数器
    clr r27                    ;清结果高字节
    clr r26
    clr r25
    clr r24
```

```
    lsr r23                    ;右移乘数
    ror r22
    ror r21
    ror r20
m32u1:
    brcc m32u2                 ;进位位为0,跳至m32u2
    add r24,r16                ;进位位为1
    adc r25,r17                ;结果高字节·被乘数
    adc r26,r18
    adc r27,r19
m32u2:
    ror r27                    ;右移结果
    ror r26
    ror r25
    ror r24
    ror r23
    ror r22
    ror r21
    ror r20
    dec r28                    ;计数器减1
    brne      m32u1            ;不为0,再循环
    ret                        ;为0,返回
```

6. "mpy32s"——32位*32位带符号乘法

(1) 程序功能：(r19:r18:r17:r16)*(r23:r22:r21:r20)，结果为64位，存放于r27:r26:r25:r24:r23:r22:r21:r20。寄存器r27最高位为0，代表正数；为1则代表负数。

(2)程序清单：

```
mpy32s:                        ;32位*32位带符号乘法
    clr r27                    ;清结果高32位和进位位
    sub r26,r26
    sub r25,r25
    sub r24,r24
    ldi r28,32                 ;初始化循环计数器
m32s_1:
    brcc      m32s_2           ;C=0,跳转至m32s_2
    add r24,r16                ;C=1,被乘数加到结果高32位
    adc r25,r17
    adc r26,r18
    adc r27,r19
m32s_2:
    sbrc r20,0                 ;判当前位是否置位
    rjmp      m32s1            ;为1,置位,跳转至m32s1
    rjmp      m32s2            ;为0,跳转至m32s2
```

```
m32s1:
    sub r24,r16              ;结果高 32 位中减去被乘数
    sbc r25,r17
    sbc r26,r18
    sbc r27,r19
m32s2:
    asr r27                  ;算术右移 r27
    ror r26                  ;循环右移结果
    ror r25
    ror r24
    ror r23
    ror r22
    ror r21
    ror r20
    dec r28                  ;计数器减 1
    brne      m32s_1         ;不为 0,继续循环
    ret                      ;返回
```

4.2.2　除法运算子程序

列出了“div8u”——8 位/8 位无符号除法，“div8s”——8 位/8 位带符号除法，“div16u”——16 位/16 位无符号除法，“div16s”——16 位/16 位带符号除法，“d16v8u”——16 位/8 位无符号除法，“div32u”——32 位/32 位无符号除法，“div32s”——32 位/32 位带符号除法，“d32v16u”——32 位/16 位无符号除法运算子程序 。

1. “div8u”——8 位/8 位无符号除法

(1) 程序功能：r16 (被除数)/r17(除数)→r16(结果)……r15(余数)。

该子程序是将 2 个寄存器 r16 和 r17 相除，结果送至寄存器 r16，余数送 r15。

(2) 程序清单：

```
div8u:                       ;8 位/8 位无符号除法
    sub r15,r15              ;清余数和进位
    ldi r18,9                ;初始化循环计数器
d8u_1:
    rol r16                  ;左移结果(被除数)
    dec r18                  ;减计数器
    brne      d8u_2          ;不为 0,跳至 d8u_2
    ret
d8u_2:
    rol r15                  ;左移余数(被除数移到除数)
    sub r15,r17              ;余数－除数
    brcc      d8u_3          ;够减,跳至 d8u_3
    add r15,r17              ;不够减,再加除数
    clc                      ;清进位位
```

```
        rjmp        d8u_1
d8u_3:
        sec                         ;置进位位
        rjmp        d8u_1
```

2. "div8s"——8 位/ 8 位带符号除法

(1) 程序功能：r16 (被除数)/r17(除数)→r16(结果)……r15(余数)。

该子程序是将 2 个寄存器 r16 和 r17 相除，结果送至寄存器 r16，余数送 r15。r16 最高位为 0，代表结果为正数；为 1，则为负数。

(2) 程序清单：

```
div8s:                          ;8 位/8 位带符号除法
        mov r14,r16             ;确定结果符号
        eor r14,r17
        sbrc r17,7              ;判除数的正负
        neg r17                 ;为负，除数取补
        sbrc r16,7              ;判被除数的正负
        neg r16                 ;为负，被除数取补
        sub r15,r15             ;清余数和进位
        ldi r18,9               ;初始化循环计数器
d8s_1:
    rol r16                     ;左移结果(被除数)
        dec r18                 ;减计数器
        brne        d8s_2       ;不为 0，跳至 d8s_2
        sbrc r14,7              ;判结果的正负
        neg r16                 ;为负，结果取补
        ret
d8s_2:
        rol r15                 ;左移余数(被除数移到除数)
        sub r15,r17             ;余数－除数
        brcc        d8s_3       ;够减，跳至 d8s_3
        add r15,r17             ;不够减，再加除数
        clc                     ;清进位位
        rjmp        d8s_1
d8s_3:
        sec                     ;置进位位
        rjmp        d8s_1
```

3. "d16v8u"——16 位/ 8 位无符号除法

(1) 程序功能：(r15:r16)/ r17→r16……r15(余数)。

该子程序是将寄存器对 r15:r16 的值与寄存器 r17 的值相除。若 T＝0，商为 8 位送至寄存器 r16，余数送 r15；若 T＝1，商超过 8 位，溢出无运算结果。

(2) 程序清单：

```
d16v8u:                         ;16 位/8 位无符号除法
```

```
        cp r15,r17              ;被除数高字节>除数
        brcc        ddd         ;结果溢出
        ldi r18,8               ;初始化循环计数器
        rol r16                 ;左移余数(被除数低字节)
    aaa:
        bst r15,7               ;被除数高位存到 T 位
        rol r15                 ;左移被除数高字节
        sub r15,r17             ;r15 减 r17
        brts        bbb
        brcc        bbb
        add r15,r17             ;结果为负,则 r15 加 r17
        clc                     ;清进位位
        rjmp        ccc
    bbb:
        sec                     ;够减,置进位位
    ccc:
        rol r16                 ;左移余数
        dec r18                 ;减计数器
        brne        aaa         ;没完成再循环
        clt
        ret
    ddd:
        set
        ret
```

4. "div16u"——16 位/ 16 位无符号除法

(1) 程序功能：(r17:r16)/(r19:r18)→(r17:r16)……(r15:r14)(余数)。

该子程序是将寄存器 r17 和 r16 的值与寄存器 r19 和 r18 的值相除，结果送至寄存器 r17 和 r16，余数送 r15 和 r14。结果为 16 位。

(2) 程序清单：

```
div16u:                         ;16 位/16 位无符号除法
    clr     r14                 ;清除余数和进位
    sub     r15,r15
    ldi     r20,17              ;初始化循环计数器
d16u_1:
    rol     r16                 ;左移被除数
    rol     r17
    dec     r20                 ;计数器减 1
    brne        d16u_2          ;不为 0,跳至 d16u_2
    ret                         ;为 0,返回
d16u_2:
    rol     r14                 ;左移余数(被除数移到除数)
```

```
        rol     r15
        sub     r14,r18             ;余数－除数
        sbc     r15,r19
        brcc        d16u_3          ;够减,跳至 d16u_3
        add     r14,r18             ;不够减,再加除数
        adc     r15,r19
        clc                         ;清进位位
        rjmp        d16u_1
    d16u_3:
        sec                         ;置进位位
        rjmp        d16u_1
```

5. "div16s"——16 位/16 位带符号除法

(1) 程序功能：(r17:r16)/(r19:r18)→(r17:r16)……(r15:r14)(余数)。

该子程序是将寄存器 r17 和 r16 的值与寄存器 r19 和 r18 的值相除,结果送至寄存器 r17 和 r16,余数送 r15 和 r14。寄存器 r17 最高位为 0,代表正数;为 1,代表负数。结果为 16 位。

(2) 程序清单：

```
    div16s:                         ; 16 位/16 位带符号除法
        mov r13,r17                 ;确定结果符号
        eor r13,r19
        sbrs r17,7                  ;判被除数的正负
        rjmp        d16s1
        com r17                     ;为负,被除数取补
        com r16
        subi r16,low(-1)
        sbci r17,high(-1)
    d16s1:
        sbrs r19,7                  ;判除数的正负
        rjmp        d16s2
        com r19                     ;为负,除数取补
        com r18
        subi r18,low(-1)
        sbci r19,high(-1)
    d16s2:
        clr r14                     ;清除余数和进位
        sub r15,r15
        ldi r20,17                  ;初始化循环计数器
    d16s3:
        rol r16                     ;左移被除数
        rol r17
        dec r20                     ;计数器减 1
        brne        d16s5
```

```
        sbrs r13,7                  ;判结果的正负
        rjmp        d16s4           ;不为 0,跳至 d16s4
        com r17                     ;为负,结果取补
        com r16
        subi r16,low(-1)
        sbci r17,high(-1)
    d16s4:
        ret
    d16s5:
        rol r14                     ;左移余数(被除数移到除数)
        rol r15
        sub r14,r18                 ;余数-除数
        sbc r15,r19
        brcc        d16s6           ;够减,跳至 d16s6
        add r14,r18
        adc r15,r19
        clc                         ;清进位位
        rjmp        d16s3
    d16s6:
        sec                         ;置进位位
        rjmp        d16s3
```

6. "d32v16u"——32 位/16 位无符号除法

(1) 程序功能：(r19:r18:r17:r16)/(r21:r20)。商为 16 位，存放于 r17:r16，余数存于 r19:r18。

(2) 程序清单：

```
    d32v16u:                        ; 32 位/16 位无符号除法
        cp r18,r20                  ;被除数高 16 位>除数
        cpc r19,r21                 ;结果溢出
        brcc        cc
        ldi r22,$10                 ;初始化循环计数器
        rol r16                     ;左移被除数
        rol r17
    aa:
        rol r18                     ;左移余数(被除数移到除数)
        bst r19,7
        rol r19
        sub r18,r20                 ;余数-除数
        sbc r19,r21
        brts        loop            ;够减,跳至 loop
        brcc        loop
        add r18,r20                 ;不够减,再加除数
        adc r19,r21
```

```
        clc                         ;清进位位
        rjmp    loop1
    loop:
        sec                         ;置进位位
    loop1:
        rol r16                     ;左移结果
        rol r17
        dec r22                     ;计数器减 1
        brne    aa                  ;不为 0,再循环
        clt                         ;清 T 标志
        ret
    cc:
        set                         ;置 T 标志
        ret
```

7. "div32u"——32 位/32 位无符号除法

(1) 程序功能：(r19:r18:r17:r16)/(r23:r22:r21:r20)。商为 32 位，存放于 r19:r18:r17:r16，余数送 r27:r26:r25:r24。

(2) 程序清单：

```
    div32u:                         ; 32 位/32 位无符号除法
        ldi r28,33                  ;初始化循环计数器
        clr r27                     ;清除余数
        clr r26
        clr r25
        clr r24
    d32u_1:
        cp r24,r20                  ;余数与除数相比较
        cpc r25,r21
        cpc r26,r22
        cpc r27,r23
        brcs    d32u_2              ;进位位为 1,即后者大,跳至 d32u_2
        sub r24,r20                 ;余数－除数
        sbc r25,r21
        sbc r26,r22
        sbc r27,r23
        sec                         ;置进位位
        rjmp    d32u_3
    d32u_2:
        clc                         ;清进位位
    d32u_3:
        rol r16                     ;左移结果(被除数)
        rol r17
        rol r18
```

```
    rol r19
    rol r24                    ;左移余数(被除数移到除数)
    rol r25
    rol r26
    rol r27
    dec r28                    ;计数器减 1
    brne      d32u_1           ;不为 0,跳至 d32u_1
    ror r27                    ;右移余数
    ror r26
    ror r25
    ror r24
    ret
```

8. "div32s"——32 位/32 位带符号除法

(1) 程序功能：(r19:r18:r17:r16)/(r23:r22:r21:r20)。商为 32 位，存放于 r19:r18:r17:r16，余数送 r27:r26:r25:r24。寄存器 r19 最高位为 0，代表正数；为 1，则代表负数。

(2) 程序清单：

```
div32s:                        ; 32 位/32 位带符号除法
    clr r27                    ;清除余数
    clr r26
    clr r25
    clr r24
    clr r30                    ;清符号位
    cpi r19,$7f                ;判被除数的正负
    brcc      qubus1           ;为负,被除数取补
d32s_1:
    cpi r23,$7f                ;判除数的正负
    brcc      qubus2           ;为负,除数取补
d32s_2:
    ldi r28,33                 ;初始化循环计数器
d32s_3:
    cp r24,r20                 ;余数与除数相比较
    cpc r25,r21
    cpc r26,r22
    cpc r27,r23
    brcs      d32s_4           ;进位位为 1,即后者大,跳至 d32s_4
    sub r24,r20                ;余数-除数
    sbc r25,r21
    sbc r26,r22
    sbc r27,r23
    sec                        ;置进位位
    rjmp      d32s_5
```

```
d32s_4:
    clc                         ;清进位位
d32s_5:
    rol r16                     ;左移结果(被除数)
    rol r17
    rol r18
    rol r19
    rol r24                     ;左移余数(被除数移到除数)
    rol r25
    rol r26
    rol r27
    dec r28                     ; 计数器减 1
    brne        d32s_3          ;不为 0 ,跳至 d32s_3
    ror r27                     ;右移余数
    ror r26
    ror r25
    ror r24
    cpi r30, $ 01               ;判符号位
    breq     qubus3             ;为 1,代表负,结果取补
d32s_6:
    ret                         ;返回
qubus1:
    inc r30                     ;符号位加 1
    com r16                     ;取反加 1(取补)
    com r17
    com r18
    com r19
    subi r16, $ ff
    sbci r17, $ ff
    sbci r18, $ ff
    sbci r19, $ ff
    rjmp        d32s_1
qubus2:
    inc r30                     ;符号位加 1
    com r20                     ;取反加 1(取补)
    com r21
    com r22
    com r23
    subi r20, $ ff
    sbci r21, $ ff
    sbci r22, $ ff
    sbci r23, $ ff
    rjmp        d32s_2
```

```
qubus3:
    com r16                    ;取反加 1(取补)
    com r17
    com r18
    com r19
    subi r16, $ ff
    sbci r17, $ ff
    sbci r18, $ ff
    sbci r19, $ ff
    rjmp        d32s_6
```

4.3　数制转换子程序

本节给出了“b16td5”16 位二进制到 5 位 BCD 码的转换、“d5tb16”5 位 BCD 码到 16 位二进制的转换、“yd5tb16”5 位压缩 BCD 码到 16 位二进制的转换子程序。

4.3.1　“b16td5”——16 位二进制数转换成 BCD 码

(1) 程序功能：将(r17:r16)中 16 位二进制转换成 BCD 码，个、十、百、千及万位分别存放子 r16,r17,r18,r19 及 r20 中。

程序思想：(r17:r16)－10 000 够减 X 次，则万位为 X；差值－1 000 够减 Y 次，则千位为 Y；差值－100 够减 Z 次，则百位为 Z；差值－10 够减 U 次，则十位为 U；差值为个位。

(2) 程序清单：

```
b16td5:
    ser r20                    ;r20 先送－1
b16td5_1:
    inc r20                    ;r20 增 1
    subi r16,low(10000)        ;(r17:r16)－10 000
    sbci r17,high(10000)
    brcc     b16td5_1          ;够减则返回 b16td5_1
    subi r16,low(－10000)      ;不够减＋10 000,恢复余数
    sbci r17,high(－10000)
    ser r19                    ;r19 先送－1
b16td5_2:
    inc r19                    ;r19 增 1
    subi r16,low(1000)         ;(r17:r16)－1 000
    sbci r17,high(1000)
    brcc     b16td5_2          ;够减则返回 b16td5_2
    subi r16,low(－1000)       ;不够减＋1 000,恢复余数
    sbci r17,high(－1000)
    ser r18                    ;r18 先送－1
b16td5_3:
```

```
    inc r18                    ;r18 增 1
    subi r16,low(100)          ;(r17:r16)－100
    sbci r17,high(100)
    brcc    b16td5_3           ;够减则返回 b16td5_3
    subi r16,low(－100)        ;不够减＋100,恢复余数
    sbci r17,high(－100)
    ser r17                    ;r17 先送－1
b16td5_4:
    inc r17                    ;r17 增 1
    subi r16,10                ;(r17:r16)－10
    brcc    b16td5_4           ;够减则返回 b16td5_4
    subi r16,－10              ;不够减＋10,恢复余数
    ret
```

4.3.2 “d5tb16”——5 位 BCD 码转换成 16 位二进制数

(1) 程序功能：将个、十、百、千及万位分别放在 r16,r17,r18,r19 及 r20 中,5 位 BCD 码转换为 16 位二进制数,存放于 r17:r16 中。

转换公式为:r16＋10＊r17＋100＊r18＋1 000＊r19＋10 000＊r20＝r17:r16

(2) 程序清单：

```
d5tb16:
        tst r17                ;测试 r17
        rjmp        d5tb16_2
d5tb16_1:
        subi r16,－10          ;r16 加 10
        dec    r17             ;r17 减 1
d5tb16_2:
        brne        d5tb16_1   ;非 0 转 d5tb16_1
        tst r18                ;测试 r18
        rjmp        d5tb16_4
d5tb16_3:
        subi r16,low(－100)    ;(r17:r16)加 100
        sbci r17,high(－100)
        dec r18                ;r18 减 1
d5tb16_4:
        brne d5tb16_3          ;非 0 转 d5tb16_3
        tst r19                ;测试 r19
        rjmp        d5tb16_6
d5tb16_5:
        subi r16,low(－1000)   ;(r17:r16)加 1 000
        sbci r17,high(－1000)
        dec r19                ;r19 减 1
```

```
d5tb16_6:
        brne        d5tb16_5        ;非 0 转 d5tb16_5
        tst r20                     ;测试 r20
        rjmp        d5tb16_8
d5tb16_7:
        subi r16,low(-10000)        ;(r17:r16)加 10 000
        sbci r17,high(-10000)
        dec r20                     ;r20 减 1
d5tb16_8:
        brne        d5tb16_7        ;非 0 转 d5tb16_7
        ret
```

4.3.3 “yd5tb16”——5 位压缩 BCD 码转换成 16 位二进制数

(1) 程序功能：将 5 位压缩 BCD 码转换为 16 位二进制数。5 位压缩 BCD 码存于 r18:r17:r16(r18 高 4 位为 0)，二进制存放于 r17:r16。

(2) 程序思想：先将压缩 BCD 码转换为 BCD 码，放到 r16(个)、r17(十)、r18(百)、r19(千)、r20(万)5 个字节中，再调用 d5tb16 子程即可。

(3) 程序清单：

```
yd5tb16:
    mov r20,r18                 ;取出万位
    mov r19,r17                 ;令 r19=r17
    mov r18,r17                 ;令 r18=r17
    mov r17,r16                 ;令 r17=r16
    andi r16,$0f                ;取出个位
    andi r17,$f0
    swap r17                    ;取出十位
    andi r18,$0f                ;取出百位
    andi r19,$f0
    swap r19                    ;取出千位
    rcall  d5tb16               ;调用(d5tob16)5 位 BCD 到 16 位二进制的转换
    ret
```

4.4 开方运算程序

在流量仪表中常用到开方运算。

下面给出了 16 位和 32 位开方运算程序。

4.4.1 “kf16”——16 位开方运算

(1) 程序功能：

实现 16 位定点数的开方运算。

Sqrt(r19:r18)=r25

使用的寄存器：r18,r19,r25,r26。

(2) 程序思想：

根据等差级数的求和公式，有

$$1+3+5+\cdots+(2n-1)=(1/2)*(1+2n-1)n=n^2$$

对于任一正整数 N，总可以找到 n，使式：$N=n^2+\epsilon,0\leqslant\epsilon<2n+1$ 成立

这里□为误差，n 为 N 的平方根，计算时，按下式进行：

$$N=1+3+5+\cdots+(2n-1)+\epsilon=\sum(2i-1)+\epsilon$$

因此，只要从 N 中依次减去 2i−1(i=1,2,…,n)，到不够减时为止。不够减时的减数右移一位，即为平方根的整数部分。

(3) 程序清单：

```
kf16:
        ldi r25,$ff                ;(r26:r25)送-1
        ldi r26,$ff
tf:
        subi r25,$fe               ;(r26:r25)加 2
        sbci r26,$ff
        sub r18,r25                ;(r19:r18)减(r26:r25)
        sbc r19,r26
        brcc        tf             ;够减则返回 tf
        ror r26                    ;不够减则将(r26:r25)除以 2,即为结果
        ror r25
        ret
```

4.4.2 “kf32”——32 位开方运算

(1) 程序功能：

实现 32 位定点数的开方运算。

Sqrt(r19:r18:r17:r16)=(r25:r24)

使用的寄存器：r0,r16,r17,r18,r19,r24,r25,r26。

(2) 程序思想：

若采用 4.4.1 所述方法，32 位无符号数开方要做 64K 32 位减法，速度极慢。采用先对高 16 位开方，求出结果的高 8 位；再在此基础上做 24 位减法，被减数减去(2n−1)，直到不够减为止。这样处理可大大减少减法次数，加快运算速度。

(3) 程序清单

```
kf32:
        rcall       kf16a
        clr r0
        ldi r24,$ff                ;r24 送 $ff
```

```
kf32_1:
        adiw r24,2              ;(r26:r25:r24)+2
        adc r26,r0
        sub r16,r24             ;被开方数-(r26:r25:r24)
        sbc r17,r25
        sbc r18,r26
        sbc r19,r0
        brcc    kf32_1          ;够减,循环
        ror r26                 ;不够减,(r26:r25:r24)右移 1 位
        ror r25
        ror r24
        ret
kf16a:
        ldi r25,$ff             ;(r26:r25)送-1
        ldi r26,$ff
tf:
        subi r25,$fe            ;(r26:r25)+2
        sbci r26,$ff
        sub r18,r25             ;(r19:r18)-(r26:r25)
        sbc r19,r26
        brcc    tf              ;够减循环
        add r18,r25             ;不够减恢复原被减数
        adc r19,r26
        subi r25,2              ;恢复原减数
        sbci r26,0
        ret
```

第 5 章　浮点数运算程序设计

5.1　4 字节浮点格式

进行数值运算时，采用定点数运算往往不能满足要求。例如短整数(8 位带符号数)表示数的范围为－128～＋127；整数(16 位带符号数)表示数的范围为－32 768～＋32 767；长整数(32 位带符号数)表示数的范围为-2^{31}～$(2^{31}-1)$。每增加或减少 1 个数字单位恒为 1，这样在绝对值很小的时候，表示的数的精度就很低。为了满足数的范围和精度的要求，常采用浮点数的格式。

二进制浮点数一般采用$\pm M\times 2^{E}$的形式表示。其中 M 为尾数，是定点数(无符号数)；±为数符；E 为指数。

若用1位二进制数表示符号位，0为正，1为负；24位二进制数表示尾数$0.1\underbrace{00\cdots0}_{23位}$～$0.1\underbrace{11\cdots1}_{23位}$；8 位二进制数表示指数，取－125～＋128；则表示数的范围就可以扩大为$\pm(0.5\times 2^{-125}\sim 1.0\times 2^{128})$，即$\pm(1.2\times 10^{-38}\sim 3.4\times 10^{38})$；精度为$2^{-24}$，即$5.9\times 10^{-8}$。

IEEE 的 4 字节浮点数标准，恰好能很好地表示这个范围的浮点数，其格式为：

数符	阶码		尾数	
31 位	30	23	22	0

最高位(第 31 位)为数符位，0 为正，1 为负；低 23 位(0～22 位)为尾数，实际尾数为$0.1\underbrace{00\cdots0}_{23位}$～$0.1\underbrace{11\cdots1}_{23位}$(小数点后第 1 位恒为 1)。

第 23～30 位这 8 位为阶码。由于指数取值范围为－125～＋128，可正可负，为简单起见，阶码采用指数的移码，阶码＝指数＋＄7E。这样阶码恒为正。

阶码为 0，即浮点数为 0(小于1.2×10^{-38})；

阶码为＄78，指数为＄78－＄7E＝－6；

阶码为＄88，指数为＄88－＄7E＝10；

阶码为＄FF，指数为＄FF－＄7E＝＄81，超出 4 字节浮点数的表示范围，表示溢出(即绝对值大于 3.4 * 10^{38})。

为了便于理解该符点数的格式，下面举几个实际例子。

例：十进制数 1＝0.5 * 2^{1}，数符为 0；阶码为 1＋＄7E＝＄7F；

尾数 0.5＝0.100…0b，去掉最高位 1，为 00…0(23 位)；

4 字节浮点数为 0011 1111 10…0b，即为＄3F 80 00 00。

例：0.5＝0.5＊2^0，数符为 0；阶码为 $7E；尾数去掉最高位 1，为 0000…0b(23 位)；

4 字节浮点数为 0011 1111 0000…0b，即为 $3F 00 00 00。

例：－5＝－101＝－0.101＊2^3

数符为 1；阶码为 3＋ $7E＝ $81；尾数去掉最高位 1，为 0100…0b(23 位)；

4 字节浮点数为 1100 0000 1010 0000…0b，即为 $C0 A0 00 00。

5.2　4 字节浮点运算子程序库——AVR32FP.INC

该浮点运算子程序主要有最基本的数值计算(加、减、乘、除运算)和整数到浮点、浮点到整数的转换。该浮点运算子程序库占 353 个字，共使用寄存器 16 个，并使用了 SRAM 5 个字节。**注意**，调用加、减、乘、除子程序前，必须将 Y 寄存器置初值，使用的 SRAM 5 个字节的地址为 Y－5，Y－4，Y－3，Y－2，Y－1，且应使这 5 个字节与堆栈区和 SRAM 其他工作区不重复。

浮点运算子程序库中包含以下 8 个子程序：

INT2FP ——16 位整数转换成 4 字节浮点数运算子程；

LONG2FP—— -32 位长整数转换成 4 字节浮点数运算子程；

FP2INT——4 字节浮点数转换成 16 位整数运算子程；

FP2LONG——4 字节浮点数转换成 32 位长整数运算子程；

ADD32F——4 字节浮点加法运算子程；

SUB32F——-4 字节浮点减法运算子程；

DIV32F——-4 字节浮点除法运算子程；

MPY32F——4 字节浮点乘法运算子程。

由于这 8 个浮点子程序都是常用的，所以将这些子程序编写在一起，各有自己的入口，又有一些共同的程序段。这样编写可以节省程序存储单元。浮点程序库总共只占 353 个字，还不到 AT90S8535 内部 Flash 的 1/10，一般不影响存放用户程序。

以下简单介绍各浮点子程序的程序思想。

1. 定点长整数(或整数)转换成浮点数

若为整数，先将其转换成长整数(正数高 16 位添 0，负数高 16 位添 1)；取出符号位及绝对值(正数绝对值不变，负数绝对值取补)；将 31 位绝对值左移至最高位为 1，其后 23 位数即为浮点数尾数(不足 23 位用 0 补足)；浮点数的阶码为 $9d 减去左移次数。

2. 浮点数取长整(或取整)

先判断阶码是否小于 $7F，若小于则结果为 0；再看阶码是否大于 $9D(或 $8D)，若大于则溢出。正数取 $7FFFFFFF(或 $7FFF)，负数取 $80000000(或 $8000)；否则，尾数最高位补 1，右移 24 位尾数($96——阶码)次，即可求得尾数绝对值。取补即得到长整数(或整数)。**注意**，($96——阶码)为负时，为低位补 0 次数。

3. 浮点加减法运算

减法可对减数改变符号做加法运算。如一个加数为 0 或两数的阶码值相差大于尾数长度(24 位)，则可忽略较小的加数，之和取较大的加数。做加法运算前先对阶，小阶对大阶，较小的数尾数右移阶码之差次，尾数做加法运算；结果再浮点规格化，转化成 24 位尾数，最高位刚

好为1，化成规定浮点格式。

4. 浮点乘法运算

不需要对准小数点，只要把阶码相加，再减去$7E；尾数相乘，符号相异或，然后对结果进行必要的规格化。

5. 浮点数除法运算

在执行浮点数除法时，应先调整被除数的阶码，使被除数的尾数小于除数的尾数（使商不大于1）；然后把阶码相减，再加上$7E，尾数相除，符号相异或。

一般浮点库使用者不必详细了解浮点于程的编写原理，只要求正确使用这些浮点子程。注意入口、出口、使用的寄存器（必要时这些寄存器应加以保护）和使用的内部SRAM地址（应使这些SRAM不作其他长期保留数据的空间和远离堆栈区）。

6. 浮点子程序库（AVR32FP.INC）清单

INT2FP—— 16位整数转换成4字节浮点数运算子程。

入口：r17：r16；出口：r19：r18：r17：r16。

使用的寄存器：r16，r17，r18，r19，r20，r26。

LONG2FP——32位长整数转换成4字节浮点数运算子程。

入口：r19：r18：r17：r16；出口：r19：r18：r17：r16。

使用的寄存器：r16，r17，r18，r19，r20，r26。

FP2INT—— 4字节浮点数转换成16位整数运算子程。

入口：r19：r18：r17：r16；出口：r19：r18。

使用的寄存器：r16，r17，r18，r19，r20，r26。

FP2LONG——4字节浮点数转换成32位长整数运算子程。

入口：r19：r18：r17：r16；出口：r19：r18：r17：r16。

使用的寄存器：r16，r17，r18，r19，r20，r26。

ADD32F——4字节浮点加法运算子程。

入口：r19：r18：r17：r16 + r24：r23：r22：r21；出口：r19：r18：r17：r16。

使用的寄存器：r16，r17，r18，r19，r20，r21，r22，r23，r24，r25，r26，r28，r29。

使用的SRAM：Y－5，Y－4，Y－3，Y－2，Y－1。

SUB32F——4字节浮点减法运算子程。

入口：r19：r18：r17：r16 － r24：r23：r22：r21；出口：r19：r18：r17：r16。

使用的寄存器：r16，r17，r18，r19，r20，r21，r22，r23，r24，r25，r26，r28，r29。

使用的SRAM：Y－5，Y－4，Y－3，Y－2，Y－1。

DIV32F——4字节浮点除法运算子程。

入口：r19：r18：r17：r16 / r24：r23：r22：r21；出口：r19：r18：r17：r16。

使用的寄存器：r13，r14，r15，r16，r17，r18，r19，r20，r21，r22，r23，r24，r25，r26，r28，r29。

使用的SRAM：Y－1。

MPY32F——4字节浮点乘法运算子程。

入口：r19：r18：r17：r16 * r24：r23：r22：r21；出口：r19：r18：r17：r16。

使用的寄存器：r13，r14，r15，r16，r17，r18，r19，r20，r21，r22，r23，r24，r25，r26，r28，r29。

使用的SRAM：Y－1。

下面给出 4 B 浮点子程序库 AVR32FP.inc，程序清单如下：

```
INT2FP:                          ;16 位整数转换成 4 字节浮点数运算子程
        CLR     R18
        SBRC    R17,7            ;R17:R16 为待转换的整数
        COM     R18
        CLR     R19              ;16 位整数按数符位扩展成 32 位长整数
        SBRC    R18,7            ;为正,R19,R18 清 0
        COM     R19              ;为负,R19,R18 置为 $FFFF
LONG2FP:                         ;32 位长整数转换成 4 字节浮点数运算子程
        CLR     R20              ;清符号位 R20
        AND     R19,R19          ;先判+/-
        BRPL    LONG2FP_1        ;为正,跳至 LONG2FP_1
        RCALL   QUBU             ;为负,调 QUBU 子程    -取补
        COM     R20              ;符号位 R20 取反
LONG2FP_1:
        MOV     R26,R16          ;再判 R19:R18:R17:R16 是否全 0
        OR      R26,R17
        OR      R26,R18
        OR      R26,R19
        BRNE    LONG2FP_2        ;不为 0 跳至 LONG2FP_2
        RJMP    JGW0             ;为 0 跳至 JGW0- -结果为 0
LONG2FP_2:
        LDI     R26,$16          ;令 R26 = 22
        RJMP    LONG2FP_4
LONG2FP_3:
        INC     R26
        LSR     R19
        ROR     R18
        ROR     R17
        ROR     R16
LONG2FP_4:
        AND     R19,R19          ;判高位是否为 0
        BRNE    LONG2FP_3        ;不为 0,右移 1 位,阶码加 1
LONG2FP_5:
        AND     R18,R18          ;判次高位是否为 0
        BRNE    LONG2FP_7
        SUBI    R26,$08          ;为 0,阶码减 8
        MOV     R18,R17          ;数左移 8 位
        MOV     R17,R16
        LDI     R16,$00          ;低位以 $00 填充
        RJMP    LONG2FP_5
LONG2FP_6:
```

```
        DEC     R26
        ADD     R16,R16
        ADC     R17,R17
        ADC     R18,R18
LONG2FP_7:
        BRPL    LONG2FP_6           ;R18 第 7 位为 0,再左移 1 位
        MOV     R19,R26             ;阶码减 1,直到 R18 第 7 位为 1
        RJMP    GGH
FP2INT:                             ;4 字节浮点数转换成 16 位整数运算子程
        LDI     R26,$0E             ;令 R26 = 14
        RJMP    FP2LONG_1           ;跳至 FP2LONG_1
FP2LONG:                            ;4 字节浮点数转换成 32 位长整数运算子程
        LDI     R26,$1E             ;令 R26 = 30
FP2LONG_1:
        RCALL   FP2LONG_11          ;调 FP2LONG_11 子程
        BREQ    FP2LONG_4           ;相等,即阶码为 0,按 0 处理
        SUB     R26,R19             ;R26(14)-阶码差值
        BREQ    FP2LONG_2           ;为 0,则跳至 FP2LONG_2
        BRPL    FP2LONG_5           ;为正,则跳至 FP2LONG_5——无溢出
FP2LONG_2:                          ;有溢出
        AND     R20,R20             ;符号为正,
        BRMI    FP2LONG_3           ;符号为负,则跳至 FP2LONG_3
        LDI     R16,$FF
        LDI     R17,$FF
        LDI     R18,$FF
        LDI     R19,$7F
        RET                         ;正向溢出,结果置为 $7F FF FF FF
FP2LONG_3:
        LDI     R16,$00
        LDI     R17,$00
        LDI     R18,$00
        LDI     R19,$80
        RET                         ;负向溢出,结果置为 $80 00 00 00
FP2LONG_4:
        LDI     R16,$00
        LDI     R17,$00
        LDI     R18,$00
        LDI     R19,$00
        RET                         ;结果为 0,置为 $00 00 00 00
FP2LONG_5:                          ;无溢出
        INC     R19                 ;阶码差值+1
        BRMI    FP2LONG_4           ;为负,即(阶码<$7E)按 0 处理
        LDI     R19,$00             ;清 R19
```

```
        SUBI    R26,$08             ;R26(14-阶码差值) -8
        BRPL    FP2LONG_7           ;为正转 FP2LONG_7
        SUBI    R26,$F8             ;不够减则加 8
        MOV     R19,R18             ;左移 8 位
        MOV     R18,R17
        MOV     R17,R16
        LDI     R16,$7F             ;低位以 $7F 填充
        RJMP    FP2LONG_8           ;跳至 FP2LONG_8
FP2LONG_6:
        MOV     R16,R17             ;右移 8 位
        MOV     R17,R18
        LDI     R18,$00             ;高位以 $00 填充
        SUBI    R26,$08             ;R26-8
FP2LONG_7:
        CPI     R26,$08             ;R26 值与 8 相比较
        BRCC    FP2LONG_6           ;有借位,跳至 FP2LONG_6
FP2LONG_8:                          ;无借位
        AND     R26,R26
        BREQ    FP2LONG_10
FP2LONG_9:
        LSR     R19                 ;右移 1 位
        ROR     R18
        ROR     R17
        ROR     R16
        DEC     R26                 ;R26-1,不为 0 跳至 FP2LONG_9
        BRNE    FP2LONG_9           ;直到 R26 为 0 为止
FP2LONG_10:                         ;考虑符号位
        SBRC    R20,7
        RJMP    QUBU                ;为负,则跳至 QUBU ---取补返回
        RET                         ;为正,返回
FP2LONG_11:
        MOV     R20,R19             ;取浮点数数符存于 R20 最高位
        ANDI    R20,$80
        ADD     R18,R18             ;将阶码移至 R19
        ADC     R19,R19
        SUBI    R19,$80             ;阶码减 $80 存于 R19
        SEC
        ROR     R18                 ;恢复尾数最高位 1
        CPI     R19,$80             ;阶码差值与 $80 相比较
        RET
ADD32F_1:                           ;存储结果
        MOV     R20,R25
        MOV     R19,R24
```

```
        MOV     R18,R23
        MOV     R17,R22
        MOV     R16,R21
ADD32F_2:
        RJMP    GGH             ;跳至 GGH——处理结果
SUB32F:                         ;4 字节浮点减法运算子程
        SUBI    R24,$80         ;减数取反,视为浮点加法运算
ADD32F:                         ;4 字节浮点加法运算子程
        RCALL   YCL             ;调 YCL 子程
        CPI     R24,$80         ;先判加数是否为 0
        BREQ    ADD32F_2        ;为 0,则和为被加数,跳至 ADD32F_2
        CPI     R19,$80         ;再判被加数是否为 0
        BREQ    ADD32F_1        ;为 0,则和为加数,跳至 ADD32F_1
ADD32F_3:
        MOV     R26,R19         ;转存被加数阶码
        SUB     R26,R24         ;R26=被加数阶码-加数阶码
        BRVS    ADD32F_2        ;溢出,跳至 ADD32F_2,即加数可忽略
        BRMI    ADD32F_4        ;为负,跳至 ADD32F_4
        BRNE    ADD32F_5        ;不等,跳至 ADD32F_5
        CP      R16,R21
        CPC     R17,R22
        CPC     R18,R23
        BRCC    ADD32F_5
ADD32F_4:
        RCALL   ADD32F_16       ;调 ADD32F_16,被加数和加数相交换
        RJMP    ADD32F_3        ;跳至 ADD32F_3
ADD32F_5:
        CPI     R26,$18         ;阶码差值与 24 相比较
        BRCS    ADD32F_6        ;C=1,跳至 ADD32F_6,即<24
        CLR     R21             ;>24 加数清 0
        CLR     R22
        CLR     R23
ADD32F_6:
        CPI     R26,$08         ;阶码差值与 8 相比较
        BRCS    ADD32F_7        ;C=1,跳至 ADD32F_7(直至<8)
        MOV     R21,R22         ;加数尾数右移 8 位
        MOV     R22,R23         ;高 8 位清 0
        CLR     R23
        SUBI    R26,$08         ;阶码差值再减 8
        RJMP    ADD32F_6        ;跳至 ADD32F_6
ADD32F_7:
        AND     R26,R26
        BREQ    ADD32F_9        ;直至 R26=0,跳至 ADD32F_9
```

```
ADD32F_8:
        LSR     R23                     ;加数尾数右移 1 位
        ROR     R22
        ROR     R21
        DEC     R26                     ;阶码差值减 1
        BRNE    ADD32F_8                ;不为 0,跳至 ADD32F_8
ADD32F_9:
        MOV     R26,R20                 ;被加数和加数是否同号
        EOR     R26,R25
        BRMI    ADD32F_10               ;异号,跳至 ADD32F_10
        RCALL   ADD32F_13               ;同号,调 ADD32F_13 子程——尾数相加
        BRCC    ADD32F_2                ;C=0,无溢出,跳至 ADD32F_2
        ROR     R18                     ;C=1,溢出,尾数右移 1 位
        ROR     R17
        ROR     R16
        SUBI    R19,$FF                 ;阶码加 1
        BRVC    ADD32F_2                ;无溢出,跳至 ADD32F_2,处理结果
        RJMP    JGZD                    ;溢出,跳至 JGZD,出结果
ADD32F_10:
        RCALL   ADD32F_14               ;调 ADD32F_14 子程——尾数相减
        BRNE    ADD32F_11               ;不等,跳至 ADD32F_11
        RJMP    JGW0                    ;跳至 JGW0,出结果
ADD32F_11:
        BRCC    ADD32F_12               ;C=0,够减,跳至 ADD32F_12
        RCALL   ADD32F_15               ;C=1,不够减,调 ADD32F_15 子程——取补
ADD32F_12:
        AND     R18,R18                 ;判 R18 最高位是否为 1
        BRMI    ADD32F_2                ;为 1,跳至 ADD32F_2,处理结果
        ADD     R16,R16                 ;为 0,尾数左移 1 位
        ADC     R17,R17
        ADC     R18,R18
        SUBI    R19,$01                 ;阶码减 1
        BRVC    ADD32F_12               ;无溢出,跳至 ADD32F_12
        RJMP    JGZD                    ;溢出,跳至 JGZD,出结果
ADD32F_13:                              ;尾数相加
        ADD     R16,R21
        ADC     R17,R22
        ADC     R18,R23
        RET
ADD32F_14:                              ;尾数相减
        SUB     R16,R21
        SBC     R17,R22
        SBC     R18,R23
```

```
        RET

ADD32F_15:                          ;结果取补
        COM     R17
        COM     R18
        NEG     R16
        SBCI    R17,$FF
        SBCI    R18,$FF
        RET
ADD32F_16:                          ;2 个数相交换
        ST      -Y,R21
        ST      -Y,R22
        ST      -Y,R23
        ST      -Y,R24
        ST      -Y,R25
        MOV     R24,R19
        MOV     R21,R16
        MOV     R22,R17
        MOV     R23,R18
        MOV     R25,R20
        LD      R20,Y+
        LD      R19,Y+
        LD      R18,Y+
        LD      R17,Y+
        LD      R16,Y+
        RET
YCL:
        MOV     R20,R19
        LDI     R26,$80
        ADD     R18,R18
        ADC     R19,R19             ;将阶码移至 R19
        EOR     R19,R26             ;阶码减 $80 存于 R19
        ADD     R26,R26
        ROR     R18                 ;恢复尾数最高位 1
        ANDI    R20,$80             ;取浮点数数符存于 R20 最高位
        MOV     R25,R24
        LDI     R26,$80
        ADD     R23,R23             ;将阶码移至 R24
        ADC     R24,R24
        EOR     R24,R26             ;阶码减 $80 存于 R24
        ADD     R26,R26
        ROR     R23                 ;恢复尾数最高位 1
        ANDI    R25,$80             ;取浮点数数符存于 R25 最高位
```

```
        CPI     R19,$80
        RET
GGH:                                ;规格化
        ADD     R18,R18             ;隐含尾数最高位为 1
        LDI     R26,$80             ;考虑符号位
        EOR     R26,R19
        ADD     R20,R20
        ROR     R26                 ;右移 R26,R18
        ROR     R18
        MOV     R19,R26             ;R26 移至 R19
        RET
DIV32F_1:
        ST      -Y,R26              ;转存 R26
        CLR     R13                 ;清 R15:R14:R13
        CLR     R14
        CLR     R15
        LDI     R26,$18             ;令 R26=$18(24)
DIV32F_2:
        CP      R16,R21             ;被除数(余数)与除数两尾数相比
        CPC     R17,R22
        CPC     R18,R23
        BRCS    DIV32F_3            ;被除数(余数)<除数
        SUB     R16,R21             ;余数=被除数-除数
        SBC     R17,R22
        SBC     R18,R23
        SEC
        RJMP    DIV32F_4
DIV32F_3:
        CLC                         ;清除进位位
DIV32F_4:
        ADC     R13,R13             ;商左移 1 位,并加上进位位
        ADC     R14,R14
        ADC     R15,R15
        ADD     R16,R16             ;余数左移 1 位
        ADC     R17,R17
        ADC     R18,R18
        DEC     R26                 ; R26-1
        BRNE    DIV32F_2            ;循环 24 次
        MOV     R16,R13             ;取出商
        MOV     R17,R14
        MOV     R18,R15
        LD      R26,Y+              ;恢复 R26
        RET
```

```
DIV32F:                          ;4 字节浮点除法运算子程
        AND     R24,R24
        BREQ    DIV32F_7         ;跳至 DIV32F_7,出结果
        AND     R19,R19          ;判被除数是否为 0
        BREQ    JGW0             ;为 0,则结果为 0
        RCALL   YCL              ;调 YCL 子程
        BREQ    JGW0             ;跳至 JGW0,结果为 0
        EOR     R20,R25          ;取商的符号位存于 R20
        SEC
        SBC     R19,R24          ;取出商的阶码
        BRVS    JGZD             ;溢出,跳至 JGZD
        LSR     R23              ;无溢出
        ROR     R22              ;将被除数与除数的尾数
        ROR     R21              ;右移 1 位,最高位置 0
        LSR     R18
        ROR     R17
        ROR     R16
        RCALL   DIV32F_1         ;调 DIV32F_1 子程,进行运算
        AND     R18,R18          ;判 R18 最高位是否为 0
        BRMI    DIV32F_5         ;为 1
        ADD     R16,R16          ;为 0,左移被除数尾数
        ADC     R17,R17
        ADC     R18,R18
        SUBI    R19,$01          ;阶码减 1
        BRVS    JGZD             ;溢出,跳至 JGZD
DIV32F_5:                        ;无溢出
        MOV     R26,R16
        LSR     R26
        BRCS    DIV32F_6         ;进位位为 1,跳至 DIV32F_6
        AND     R16,R16
        BRPL    DIV32F_6         ;为正,跳至 DIV32F_6
        AND     R17,R17
        BRPL    DIV32F_6         ;为正,跳至 DIV32F_6
        LDI     R26,$01          ;尾数加 1
        ADD     R16,R26
        CLR     R26
        ADC     R17,R26
        ADC     R18,R26
DIV32F_6:
        RJMP    GGH
DIV32F_7:
        RJMP    JGW0
JGZD:                            ;结果置为 $7FFFFFFF
```

```
        LDI     R26,$7F
        MOV     R19,R26
        OR      R18,R26
        LDI     R26,$FF
        MOV     R16,R26
        MOV     R17,R26
        RET
JGW0:                           ;结果置为 $00000000
        CLR     R16
        CLR     R17
        CLR     R18
        CLR     R19
        CLR     R20
        RET
MPY32F:                         ;4 字节浮点乘法运算子程
        RCALL   YCL             ;调 YCL 子程,并判乘数是否为 0
        BREQ    JGW0            ;被乘数为 0,跳至 JGW0 ……结果为 0
        CPI     R24,$80         ;判乘数是否为 0
        BREQ    JGW0            ;乘数为 0,跳至 JGW0——结果为 0
        EOR     R20,R25         ;符号位相异或
        SEC
        ADC     R19,R24         ;恢复阶码,存于 R19
        BRVS    JGZD            ;溢出,跳至 JGZD
        RCALL   MPY32F_2        ;无溢出,调 MPY32F_2 子程
        AND     R18,R18         ;判 R18 最高位是否为 1
        BRMI    MPY32F_1        ;为负,即 R18 最高位为 1,跳至 MPY32F_1
        DEC     R19             ;为正,即 R18 最高位为 0,阶码减 1
        ADD     R15,R15         ;尾数左移 1 位
        ADC     R16,R16
        ADC     R17,R17
        ADC     R18,R18         ;直至 R18 最高位为 1 止
MPY32F_1:
        SUBI    R19,$FF         ;阶码加 1
        BRVS    JGZD            ;溢出,跳至 JGZD
        RJMP    GGH             ;跳至 GGH——出结果
MPY32F_2:
        ST      -Y,R24          ;转存 R24
        CLR     R13             ;清 R26,R15,R14,R13
        CLR     R14
        CLR     R15
        CLR     R26
        LDI     R24,$18         ;令 R24=$18(24)
MPY32F_3:
```

```
        ADD     R13,R13             ;积的尾数在 R18:R17:R16:R15:R14:R13
        ADC     R14,R14
        ADC     R15,R15
        ADC     R16,R16             ;尾数左移
        ADC     R17,R17
        ADC     R18,R18
        BRCC    MPY32F_4            ;无进位,R24 减 1,不加乘数
        ADD     R13,R21             ;有进位,R24 减 1,加乘数到尾数低位
        ADC     R14,R22
        ADC     R15,R23
        ADC     R16,R26
        ADC     R17,R26
        ADC     R18,R26
MPY32F_4:
        DEC     R24                 ;循环 24 次
        BRNE    MPY32F_3
        LD      R24,Y+              ;恢复 R24
        RET                         ;取高 24 位
QUBU:                               ;取补运算
        COM     R16
        COM     R17
        COM     R18
        COM     R19
        SUBI    R16,$FF
        SBCI    R17,$FF
        SBCI    R18,$FF
        SBCI    R19,$FF
        RET
```

5.3 应用举例

例 1:用浮点子程序库计算 X=(A+B)×(C−D)/E

设 A=1,B=2,C=3,D=−4,E=−5。计算时把整数转换成浮点数进行四则运算,结果再转换成整数。

程序如下:

```
.include "8535def.inc"
ldi r16,$5f
out spl,r16
ldi r16,$02
out sph,r16
ldi YL,$50
```

```
ldi YH, $01
ldi r16,low(-5)
ldi r17,high(-5)
rcall       INT2FP
push r16
push r17
push r18
push r19
ldi r16, $02
ldi r17, $00
rcall       INT2FP
mov r21,r16
mov r22,r17
mov r23,r18
mov r24,r19
ldi r16, $01
ldi r17, $00
rcall       INT2FP
rcall       ADD32F
push r16
push r17
push r18
push r19
ldi r16,low(-4)
ldi r17,high(-4)
rcall       INT2FP
mov r21,r16
mov r22,r17
mov r23,r18
mov r24,r19
ldi r16, $03
ldi r17, $00
rcall       INT2FP
rcall       SUB32F
pop r24
pop r23
pop r22
pop r21
rcall       MPY32F
pop r24
pop r23
pop r22
pop r21
```

```
    rcall       DIV32F
    rcall       FP2INT
here:           rjmp here
    .include "AVR32FP.inc"
```

执行该程序后，r17:r16 应为 $FFFB(−5)，−4.2 取整为−5；若改为 E=5，则结果为 $0004，+4.2 取整为 4。

例 2：编程求一常数−12.345 的 4 字节浮点数。

程序如下：

```
    .include "8535def.inc"
    ldi r16,$5f
    out spl,r16
    ldi r16,$02
    out sph,r16
    ldi YL,$50
    ldi YH,$01
    ldi r16,low(1000)
    ldi r17,high(1000)
    rcall       INT2FP
    mov r21,r16
    mov r22,r17
    mov r23,r18
    mov r24,r19
    ldi r16,low(−12345)
    ldi r17,high(−12345)
    rcall       INT2FP
    rcall       DIV32F
here:           rjmp here
    .include "AVR32FP.inc"
```

执行该程序后，r19:r18:r17:r16 中即为此浮点数($C145851E)。任一常数的浮点数都可这样求得，在编程时可直接引用这 4 字节浮点立即数。

第 6 章　8535 单片机 EEPROM 读/写访问

6.1　8535 单片机 EEPROM 读/写

6.1.1　概　述

写入 EEPROM 的时间约在 2.5～4 ms 之间，取决于 V_{cc} 电压。自定时功能可使用户软件检测何时写入下一个字节。

在电源滤波时间常数比较大的电路中，上电/下电时，V_{cc} 上升/下降会比较慢，此时 MCU 将工作于低于晶振所要求的电源电压。在这种情况下，程序指针有可能跑飞，并执行 EEP 误写操作。为了保证 EEP 数据的正确性，建议使用电压复位电路。

为了防止无意识的 EEPROM 写入，需要执行一个特定的写程序。具体参看后面的内容。

当执行 EEPROM 读或写时，CPU 会停止工作 2 个时钟周期后，再执行下一条指令。

6.1.2　有关的 I/O 寄存器

用于访问 EEPROM 的寄存器位于 I/O 空间。有以下 4 个 I/O 寄存器。

1. EEPROM 地址寄存器——EEAR

	位 15	14	13	12	11	10	9	8	
$1F($3F)	—	—	—	—	—	—	—	EEAR8	EEARH
$1E($3E)	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
	位 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	
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

初始化值：0b0000 000X XXXX XXXX

EEPROM 地址寄存器——EEARH 和 EEARL，指定了在 512 B 的 EEPROM 空间中的 EEPROM 地址。EEPROM 数据在 0～511 B 之间被线性编址。

2. 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：$00

位 7～0——EEDR7～EEDR0：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。

3. EEPROM 控制寄存器——EECR

	位 7	6	5	4	3	2	1	0	
$1C($3C)	—	—	—	-	EERIE	EEMWE	EEWE	EERE	EECR
读/写：	R	R	R	R	R/W	R/W	R/W	R/W	

初始化值：$00

位 7～4——Res：保留位。

90 系列单片机的这些位为保留位，总读 0。

位 3——EERIE：EEPROM 准备好中断使能。

当 I 和 EERIE 置位时，EEPROM 准备好中断使能。EEWE 为“0”时，EEPROM 准备好中断，将产生中断(constant)。

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 EEPROM 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 0。用户软件在写入下一个字节之前，查询这一位，并等待 0 值。当 EEWE 被设置后，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，无须查询 EERE 位。一旦 EERE 被设置后，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。

6.2　片内 EEPROM 读/写举例

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”及一个测试程序。

程序清单如下：

```
.include "8535def.inc"
rjmp    RESET                ;处理复位
EEWrite:                     ;写 EEPROM 子程,r18:r17 放写入地址,r16 放要写入的数据
    sbic  EECR,EEWE          ;如果 EEWE 不清除
    rjmp  EEWrite            ;等待
    out   EEARH,r18          ;输出地址
    out   EEARL,r17
    out   EEDR,r16           ;输出数据
    sbi   EECR,EEMWE         ;设置 EEPROM 写选通
    sbi   EECR,EEWE          ;该指令需 4 个时钟周期,由于它暂停 CPU 2 个时钟周期
    ret
EERead:                      ;读 EEPROM 子程,r18:r17 放读出地址,r16 放读到的数据
    sbic  EECR,EEWE          ;如果 EEWE 不清除
    rjmp  EERead             ;等待
    out   EEARH,r18          ;输出地址
    out   EEARL,r17
    sbi   EECR,EERE          ;设置 EEPROM 读选通
    in    r16,EEDR           ;读入数据
    ret
EEWrite_seq:                 ;连续写 EEPROM 子程,写入地址(r25:r24)+1,写入的数据放 r16
    sbic  EECR,EEWE          ;如果 EEWE 不清除
    rjmp  EEWrite_seq        ;等待
    in    r24,EEARL          ;得到地址
    in    r25,EEARH
    adiw  r24,0x01           ;地址加 1
    out   EEARL,r24          ;输出地址
    out   EEARH,r25
    out   EEDR,r16           ;输出数据
    sbi   EECR,EEMWE         ;设置 EEPROM 写选通
    sbi   EECR,EEWE
    ret
EERead_seq:                  ;连续读 EEPROM 子程,读出地址(r25:r24)+1,读到的数据放 r16
    sbic  EECR,EEWE          ;如果 EEWE 不清除
    rjmp  EERead_seq         ;等待
```

```
    in    r24,EEARL          ;得到地址
    in    r25,EEARH
    adiw  r24,0x01           ;地址加 1
    out   EEARL,r24          ;输出地址
    out   EEARH,r25
    sbi   EECR,EERE          ;设置 EEPROM 读选通
    in    r16,EEDR           ;读入数据
    ret
;测试程序
RESET:
    ldi   r16,$02            ;栈指针置初值
    out   SPH,r16
    ldi   r16,$5F
    out   SPL,r16
;*****将 aa 写入 EEPROM 的 $40 地址,再读出来送 B 口输出
    ldi   r16,$ff            ;定义 B 口为输出
    out   DDRB,r16
    ldi   r16,$aa
    ldi   r18,$00
    ldi   r17,$40
    rcall EEWrite            ;存储 $AA 到 EEPROM 的 $40 地址
    ldi   r18,$00
    ldi   r17,$40
    rcall EERead             ;读 $10 地址
    out   PORTB,r16          ;输出到 PORT B 口
;*****以 $55,$aa,$55,$aa,…模式填充 EEPROM
    ldi   r19,16             ;初始化循环计数器
    ser   r20                ;r20←$FF
    out   EEARH,r20          ;EEARH←$FF
    ser   r20
    out   EEARL,r20          ;EEAR←$FF (start address-1)
loop1:  ldi   r16,$55
    rcall EEWrite_seq        ;写 $55 到 EEPROM
    ldi   r16,$aa
    rcall EEWrite_seq        ;写 $AA 到 EEPROM
    dec   r19                ;计数器减 1
    brne  loop1              ;未完成循环
;*****拷贝 EEPROM 最前 10 个字节到 r2-r11
    ser   r20
    out   EEARH,r20          ;EEARH←$FF
    ser   r20
    out   EEARL,r20          ;EEAR←$FF (start address-1)
    clr   ZH
```

```
        ldi    ZL,1                ;Z 指针指向 r1
    loop2:
        rcall  EERead_seq          ;得到 EEPROM 数据
        st Z, r16                  ;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,原地踏步
```

第 7 章　8535 单片机 I/O 端口及其应用

7.1　8535 的 I/O 口

AT90S8535 有 4 个 8 位的 I/O 口，分别是端口 A、端口 B、端口 C、端口 D。这 32 个引脚均可以由程序定义为输入口或者输出口。本章只讲基本的输入、输出功能。这 32 个引脚还有第 2 功能，在以后几章再讲述。

7.1.1.　有关 I/O 口的寄存器

(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：$00									

(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：$00									

(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	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。

(4) 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：$00

(5) B 口数据方向寄存器——DDRB

	位 7	6	5	4	3	2	1	0	
$17($37)	DDB7	DDB6	DDB5	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：$00

(6) 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。

(7) 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：$00

(8) 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：$00

(9) 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。

(10) 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：$00

(11) 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：$00

(12) 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。

7.1.2 I/O 口内部电路及工作原理

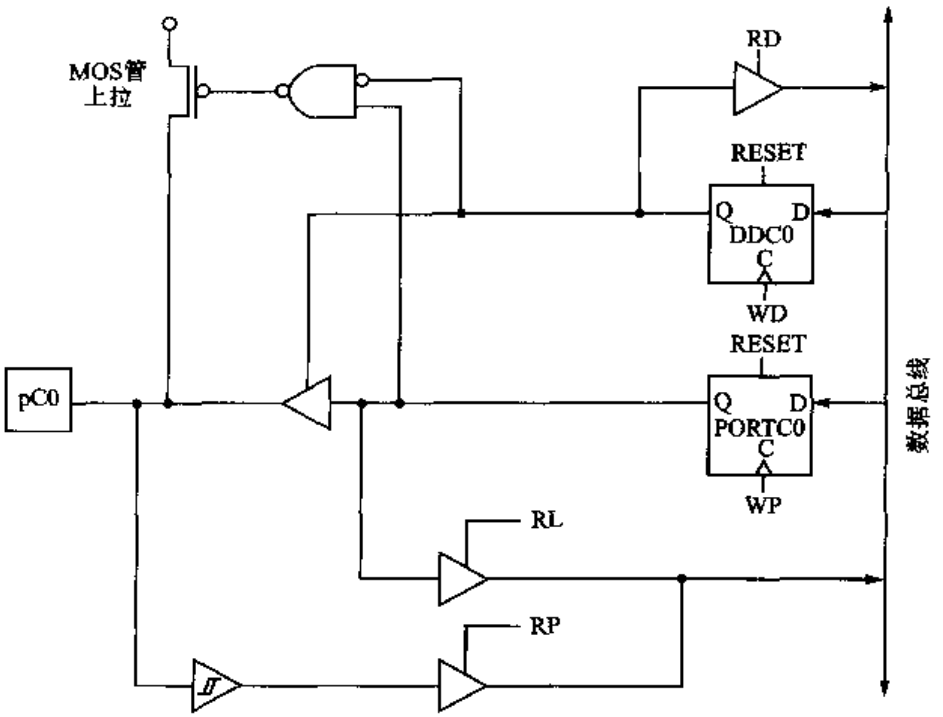
以 PC0 为例说明不考虑端口的第 2 功能时，I/O 口内部电路及工作原理，如图 7.1 所示。

① 当 C 口方向寄存器 DDRC 的第 0 位置位(DDC0＝1)时，PC0 口定义输出。由 PC0 口的内部工作原理图可见，DDC0 的 Q 端为 1，MOS 管上拉截止；同时 PORTC0 的三态门打开。PC0 引脚的输出电平取决于 C 口数据寄存器 PORTC 的第 0 位(PORTC0)的输出 Q 端值，即：DDC0＝1(PC0 定义为输出)，PORTC0＝1 时，PC0 输出高电平；PORTC0＝0 时，PC0 输出低电平。

② 当 DDC0＝0 时，PC0 口定义输入。PORTC0＝0 时(上拉 MOS 截止)，PC0 口作三态输入；PORTC0＝1 时(上拉 MOS 激活)，PC0 口作带上拉电阻的输入。后一种输入方式可省去键盘、开关、继电器等接口电路的上拉电阻。

DDC0 是可读的，用指令 in r16，DDRC，再看 r16 的第 0 位；PORTC0 的输出 Q 也是可读的，用指令 in r16，PORTC ，再看 r16 的第 0 位，读的是锁存器(PORTC)；PINC0 是可读的，用指令 in r16，DDRC，再看 r16 的第 0 位，读的是引脚电平；DDC0，PORTC0 都是可写的(可定义输入或输出和确定输出电平的高低等)；但 PINC 是不可写的(引脚电平的高低是外部客观存在)。

PORTC 的另外 7 个端子不考虑端口的第 2 功能时，I/O 口基本输入、输出功能是一样的。



WP：写C口；WD：写DDRC；RL：读C口锁存器；RP：读C口脚；RD：读DDRC

图 7.1　PC0 口的内部工作原理图

表 7.1 列出了 C 口引脚的 DDCn 的作用。

表 7.1　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	外部拉低时 PCn 引脚会输出电流
1	0	输　出	否	推挽 0 输出
1	1	输　出	否	推挽 1 输出

n：7，6，…，0 为引脚号

③ C 口特性

C 口是一个带内部上拉的 8 位双向 I/O 口。

C 口占了 3 个 I/O 寄存器地址，分别是 C 口数据寄存器 PORTC $15($35)、C 口数据方向寄存器 DDRC $14($34)及 C 口输入引脚 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 口引脚都有独立可选的上拉，C 口的输出缓冲器可以吸收 20 mA 的电流，以直接驱动 LED 显示。当 PC0～PC7 引脚被用作输入且被外部拉低时，若内部拉高被触发，这些引脚将成为电流源(I_{IL})。

某些 C 口的引脚有第 2 功能，这点后面有关章节再讲。

另外 3 个 A 口、B 口、D 口，在不考虑端口的第 2 功能时，I/O 口基本输入、输出功能也是一样的，不再重复。

7.1.3 I/O 口的特点

(1) 作输入或输出可定义。

(2) 输出时,低电平灌电流大于 20 mA。若允许输出口电平升至 1 V 以上,灌电流可达 40 mA。

(3) 输入时,可三态输入(不带上拉,大于 2.2 V 为 1);也可带上拉,可省去外电路的上拉电阻。

7.2 I/O 口的应用

7.2.1 I/O 口使用注意事项

(1) 先定义 I/O 口方向,对方向寄存器的某位置 1 为输出,清 0 为输入。

(2) 作输入口时,若需上拉电阻,可对口数据寄存器相应位置 1,这样省去外部电路的上拉电阻。可作为拨动开关、继电器接点、键盘及数字拨码盘的输入口。

(3) 作输出口时,上拉电阻已断开,对口数据寄存器相应位置 1,推挽输出高电平;对口数据寄存器相应位清 0,推挽输出低电平。

从驱动能力看,低电平时的灌电流更强些。若不介意电压升到 1 V,灌电流可达 40 mA,可直接驱动继电器和 LED 数码管。

7.2.2 I/O 口应用举例

1. 数字拨码盘输入 2 位 BCD 码,数码管静态显示出来

如图 7.2 所示,B 口定义为带内部上拉的输入口,外接 2 个 BCD 码数字拨盘,每个 BCD 码拨盘后面有 5 位引出线,其中 1 位为输入控制线(编号为 A),另外 4 位是数据线(编号为 8,4,2,1)。拨盘被拨到某一个位置时,输入控制线(A)分别与 4 位数据线中的某几位接通,A 端接地。例如把拨盘拨为 3,则数据线 2,1 与 A 相通,读入低电平;8,4 与 A 不通,读入高电平,即读入为 $0C,取反即为 3。C 口、D 口定义为输出,经 300 Ω 电阻接 2 共阴数码管各段,数码

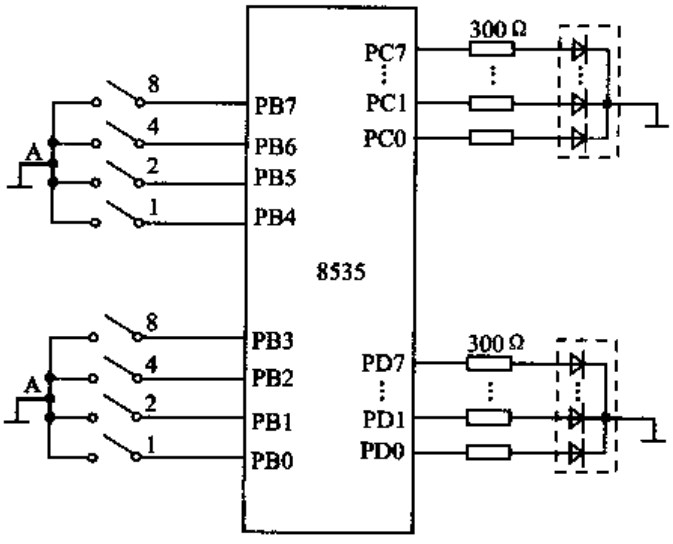


图 7.2 拨码盘数码管静态显示

管公共端接地。程序如下：

```
.include "8535def.inc"
    rjmp reset
tab:.db $3f,$06,$5b,$4f,$66,$6d,$7d,$07,$7f,$6f    ;7 段码表
reset:ldi r16,$02                                  ;栈指针置初值
    out sph,r16
    ldi r16,$5f
    out spl,r16
    ldi r16,0                                      ;定义 B 口输入带上拉
    out ddrb,r16
    ldi r16,$ff
    out portb,r16
    out ddrc,r16                                   ;定义 C 口、D 口为输出
    out ddrd,r16
    in r16,pinb                                    ;读 B 口引脚
    com r16                                        ;取反
    mov r17,r16
    andi r16,$0f                                   ;取个位 BCD 码
    swap r17                                       ;半字节交换
    andi r17,$0f                                   ;取 10 位 BCD 码
    ldi ZH,high(tab*2)                             ;查个位 7 段码
    ldi ZL,low(tab*2)
    add ZL,r16
    lpm
    out portc,r0                                   ;送 C 口静态输出
    ldi ZH,high(tab*2)                             ;查 10 位 7 段码
    ldi ZL,low(tab*2)
    add ZL,r17
    lpm
    out portd,r0                                   ;送 D 口静态输出
h:  rjmp  h
```

2. 动态扫描 5 位数码管显示

由上例，静态显示 1 位数码需占 8 位口(包括小数点)，要显示 5 位数码需 40 位输出口，这显然是不实用的。为了节省输出口，可采用动态扫描方式，显示 5 位数码只需 13 位输出口。电路如图 7.3 所示。

C 口作字线、D 口低 5 位作位线，动态扫描显示 5 位数码，数码管是共阴的。待显示的 16 位二进制数在 r17:r16 中，将其二转十，5 位 BCD 码的个、十、百、千、万分别存于 r18，r19，r20，r21 及 r22 中。每隔 1 ms 将 1 位 BCD 码查出 7 段码送 B 口输出，送相应的位线由 D 口输出，延时 1 ms，再显示下一位。这样，每隔 1 ms 显示 1 位，5 ms 显示 1 遍，反复调用扫描显示子程，每秒可显示 200 遍。实验证明，每秒显示 30 遍以上，人眼就可看见稳定的 5 位数字。编程如下：

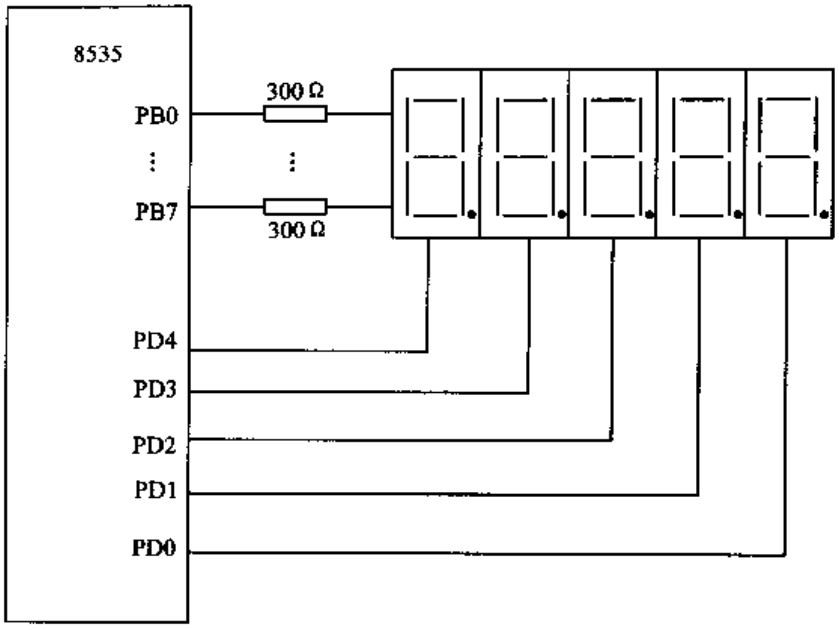


图 7.3 动态扫描显示

```
.include "8535def.inc"
.org $0000
rjmp reset
tab:.db $3f,$06,$5b,$4f,$66,$6d,$7d,$07,$7f,$6f
reset: ldi r16,low(ramend)          ;栈指针置初值
       out spl,r16
       ldi r16,high(ramend)
       out sph,r16
       ldi r16,$ff                  ;定义 PB,PD 为输出口
       out ddrb,r16
       out ddrd,r16
       ldi r16,$ff                  ;设待显示数为 $FFFF
       ldi r17,$ff
       rcall b16td5                 ;调二转十子程见 4.3.1
       mov r22,r20                  ;将 BCD 码送 r18～r22
       mov r21,r19
       mov r20,r18
       mov r19,r17
       mov r18,r16
aa:    rcall smiao                  ;调动态扫描子程
       rjmp aa

smiao: ldi r16,$fe                  ;送个位位线
       out portd, r16
       mov r23,r18                  ;将个位的 BCD 码送 r23
       rcall cqb                    ;查 7 段码,送 B 口输出
```

```
        rcall t1ms              ;延时 1 ms
        ldi r16, $ fd           ;送十位位线
        out portd, r16
        mov r23,r19             ;将十位的 BCD 码送 r23
        rcall cqb               ;查 7 段码,送 B 口输出
        rcall t1ms              ;延时 1 ms
        ldi r16, $ fb           ;送百位位线
        out portd,r16
        mov r23,r20             ;将百位的 BCD 码送 r23
        rcall cqb               ;查 7 段码,送 B 口输出
        rcall t1ms              ;延时 1 ms
        ldi r16, $ f7           ;送千位位线
        out portd,r16
        mov r23,r21             ;将千位的 BCD 码送 r23
        rcall cqb               ;查 7 段码,送 B 口输出
        rcall t1ms              ;延时 1 ms
        ldi r16, $ ef           ;送万位位线
        out portd,r16
        mov r23,r22             ;将万位的 BCD 码送 r23
        rcall cqb               ;查 7 段码,送 B 口输出
        rcall t1ms              ;延时 1 ms
        ret

cqb:    ldi zh,high(tab * 2)    ;7 段码的首地址给 Z
        ldi zl,low(tab * 2)
        add zl,r23              ;首地址+偏移量
        lpm                     ;查表送 B 口输出
        out portb,r0
        ret

t1ms:   ldi r24,101             ;延时 1 ms 子程
        push r24
del2:   push r24
del3:   dec r24
        brne del3
        pop r24
        dec r24
        brne del2
        pop r24
        ret
```

第 8 章　中断系统及应用

8.1　中断源

AT90S8535 有 16 个中断源。每个中断源在程序空间都有一个独立的中断向量。所有的中断事件都有自己的使能位。在使能位置位，且 I 也置位的情况下，中断可以发生。

程序空间的最低位置定义为复位及中断向量。完整的中断表见表 8.1。在中断向量表中，处于低地址的中断具有高的优先级；所以，RESET 具有最高的优先级。

表 8.1　复位与中断向量

向量号	程序地址	来　源	定　义
1	$000	RESET	硬件引脚上电复位和看门狗复位
2	$001	INT0	外部中断 0
3	$002	INT1	外部中断 1
4	$003	TIMER2 COMP	T/C2 比较匹配
5	$004	TIMER2 OVF	T/C2 溢出
6	$005	TIMER1 CAPT	T/C1 捕捉事件
7	$006	TIMER1 COMPA	T/C1 比较匹配 A
8	$007	TIMER1 COMPB	T/C1 比较匹配 B
9	$008	TIMER1 OVF	T/C1 溢出
10	$009	TIMER0 OVF	T/C0 溢出
11	$00A	SPI,STC	串行传输结束
12	$00B	UART,RX	UART 接收结束
13	$00C	UART,UDRE	UART 数据寄存器空
14	$00D	UART,TX	UART 发送结束
15	$00E	ADC	ADC 转换结束
16	$00F	EE_RDY	EEPROM 准备好
17	$010	ANA_COMP	模拟比较器

设置中断向量地址最典型的方法如下：

```
地　址　　标　号　　代　码　　注　释
$000　　RJMP RESET　　;复位
$001　　RJMP EXT_INT0　　;IRQ0
$002　　RJMP EXT_INT1　　;IRQ1
$003　　RJMP TIM2_COMP　　;T2 比较匹配
$004　　RJMP TIM2_OVF　　;T2 溢出
```

```
$005          RJMP TIM1_CAPT          ;T1 捕捉
$006          RJMP TIM1_COMPA         ;T1 比较 A 匹配
$007          RJMP TIM1_COMPB         ;T1 比较 B 匹配
$008          RJMP TIM1_OVF           ;T1 溢出
$009          RJMP TIM0_OVF           ;T0 溢出
$00a          RJMP SPI_STC            ;SPI 传输结束
$00b          RJMP UART_RXC           ;UART 接收结束
$00c          RJMP UART_DRE           ;UART 数据空
$00d          RJMP UART_TXC           ;UART 发送结束
$00e          RJMP ADC                ;A/D 转换结束
$00f          RJMP EE_RDY             ;EEP 准备好
$010          RJMP ANA_COMP           ;模拟比较器
$011    MAIN: LDI  R16,HIGH(REMEND)   ;主程序开始
$012          OUT SPH, R16
$013          LDI  R16,LOW(REMEND)
$014          OUT SPL, R16
$015          <指令> XXX
```

8.2　中断处理

AT90S8535 有 2 个中断屏蔽控制寄存器，GIMSK——通用中断屏蔽寄存器和 TIMSK——T/C 中断屏蔽寄存器。

一个中断产生后，全局中断使能位 I 将被清 0，后续中断被屏蔽。用户可以在中断例程里对 I 置位，从而开放中断。执行 RETI 后，I 重新置位。

当程序计数器指向实际中断向量开始执行相应的中断例程时，硬件清除对应的中断标志。一些中断标志位也可以通过软件写“1”清除。

当一个符合条件的中断发生后，如果相应的中断使能位为“0”，则中断标志位挂起，并一直保持到中断执行，或者被软件清除。

如果全局中断标志被清 0，则所有的中断都不会被执行，直到 I 置位；然后被挂起的各个中断按中断优先级依次中断。

注意：外部电平中断没有中断标志位；因此当电平变为非中断电平后，中断条件即终止。

8.3　有关的 I/O 寄存器

1. 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：$00

位 7　　INT1：外部中断 1 请求使能。

当 INT1 和 I 都为“1”时，外部引脚中断使能。MCU 通用控制寄存器(MCUCR)中的中断检测控制位 I/O(ISC11 和 ISC10)定义中断 1 是上升沿中断还是下降沿中断，或者是低电平中断。即使引脚被定义为输出，中断仍可产生。

位 6——INT0：外部中断 0 请求使能。

当 INT0 和 I 都为“1”时，外部引脚中断使能。MCU 通用控制寄存器(MCUCR)中的中断检测控制位 I/O(ISC01 和 ISC00)定义中断 0 是上升沿中断还是下降沿中断，或者是低电平中断。即使引脚被定义为输出，中断仍可产生。

位 5～0——Res：保留。

2. 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：$00

位 7——INTF1：外部中断标志 1。

当 INTF1 引脚有事件触发中断请求时，INTF1 置位(“1”)。如果 SREG 中的 I 及 GIMSK 中的 INT1 都为“1”，则 MCU 将跳转到中断地址 $002。中断例程执行后，此标志被清除。另外，标志也可以通过对其写“1”来清除。

位 6——INTF0：外部中断标志 0。

当 INTF0 引脚有事件触发中断请求时，INTF0 置位(“1”)。如果 SREG 中的 I 及 GIMSK 中的 INT0 都为“1”，则 MCU 将跳转到中断地址 $001。中断例程执行后，此标志被清除。另外，标志也可以通过对其写“1”来清除。

位 5～0——Res：保留位。

3. T/C 中断屏蔽寄存器——TIMSK

	位 7	6	5	4	3	2	1	0	
$39($59)	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	—	TOIE0	TIMSK
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	

初始化值：$00

位 7——OCIE2：T/C2 输出比较匹配中断使能。

当 TOIE2 和 I 都为“1”时，输出比较匹配中断使能。当 T/C2 的比较匹配发生，或 TIFR 中的 OCF2 置位，中断例程($003)将执行。

位 6——TOIE2：T/C2 溢出中断使能。

当 TOIE2 和 I 都为“1”时，T/C2 溢出中断使能。当 T/C2 溢出，或 TIFR 中的 TOV2 位置位时，中断例程($004)得到执行。

位 5——TICIE1：T/C1 输入捕捉中断使能。

当 TICIE1 和 I 都为“1”时，输入捕捉中断使能。当 T/C1 的输入捕捉事件发生(ICP)，或

TIFR 中的 ICF1 置位，中断例程（$005）将执行。

位 4——OCIE1A：T/C1 输出比较 A 匹配中断使能。

当 TOIE1A 和 I 都为“1”时，输出比较 A 匹配中断使能。当 T/C1 的比较 A 匹配发生，或 TIFR 中的 OCF1A 置位，中断例程（$006）将执行。

位 3——OCIE1B：T/C1 输出比较 B 匹配中断使能。

当 TOIE1B 和 I 都为“1”时，输出比较 B 匹配中断使能。当 T/C1 的比较 B 匹配发生，或 TIFR 中的 OCF1B 置位，中断例程（$007）将执行。

位 2——TOIE1：T/C1 溢出中断使能。

当 TOIE1 和 I 都为“1”时，T/C1 溢出中断使能。当 T/C1 溢出，或 TIFR 中的 TOV1 位置位时，中断例程（$008）得到执行。

位 1——Res：保留位。

位 0——TOIE0：T/C0 溢出中断使能。

当 TOIE0 和 I 都为“1”时，T/C0 溢出中断使能。当 T/C0 溢出，或 TIFR 中的 TOV0 位置位时，中断例程（$009）得到执行。

4. T/C 中断标志寄存器——TIFR

	位 7	6	5	4	3	2	1	0	
$38($58)	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	—	TOV0	TIFR
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	

初始化值：$00

位 7——OCF2：T/C2 输出比较标志。

当 T/C2 与 OCR2 的值匹配时，OCF2 置位。此位在中断例程里硬件清 0，或者通过对其写“1”来清 0。当 SREG 中的位 I，OCIE2 及 OCF2 一同置位时，中断例程得到执行。

位 6——TOV2：T/C2 溢出中断标志位。

当 T/C2 溢出时，TOV2 置位。执行相应的中断例程后，此位硬件清 0；此外，TOV2 也可以通过写“1”来清 0。当 SREG 中的位 I，TOIE2 及 TOV2 一同置位时，中断例程得到执行。在 PWM 模式中，当 T/C2 在 $00 改变计数方向时，TOV2 置位。

位 5——ICF1：输入捕获标志位。

当输入捕获事件发生时，ICF1 置位，表明 T/C1 的值已经送到输入捕获寄存器 ICR1。此位在中断例程里硬件清 0，或者通过对其写“1”来清 0。当 SREG 中的位 I，TICIE1A 及 ICF1 一同置位时，中断例程得到执行。

位 4——OCF1A：输出比较标志 1A。

当 T/C1 与 OCR1A 的值匹配时，OCF1A 置位。此位在中断例程里硬件清 0，或者通过对其写“1”来清 0。当 SREG 中的位 I，OCIE1A 及 OCF1A 一同置位时，中断例程得到执行。

位 3——OCF1B：输出比较标志 1B。

当 T/C1 与 OCR1B 的值匹配时，OCF1B 置位。此位在中断例程里硬件清 0，或者通过对其写“1”来清 0。当 SREG 中的位 I，OCIE1B 及 OCF1B 一同置位时，中断例程得到执行。

位 2——TOV1：T/C1 溢出中断标志位。

当 T/C1 溢出时，TOV1 置位。执行相应的中断例程后，此位硬件清 0。此外，TOV1 也可

以通过写"1"来清0。当SREG中的位I,TOIE1及TOV1一同置位时,中断例程得到执行。在PWM模式中,当T/C1在$0000改变计数方向时,TOV1置位。

位1—— Res:保留位。

位0——TOV0:T/C0溢出中断标志位。

当T/C0溢出时,TOV0置位。执行相应的中断例程后,此位硬件清0。此外,TOV0也可以通过写"1"来清0。当SREG中的位I,TOIE0及TOV0一同置位时,中断例程得到执行。

8.4 外部中断

外部中断由INT0和INT1引脚触发。应当注意,如果中断使能,则即使INT0/INT1配置为输出,中断照样会被触发。此特点提供了一个产生软件中断的方法。触发方式可以为上升沿、下降沿或低电平。这些设置由MCU控制寄存器MCUCR决定。当设置为低电平触发时,只要电平为低,中断就一直触发。

8.5 中断响应时间

AVR中断响应时间最少为4个时钟周期。在这4个时钟期间,PC(2个字节)自动入栈,而SP减2。在通常情况下,中断向量处为一个相对跳转指令,此跳转要花2个时钟周期。如果中断在一个多周期指令执行期间发生,则在此一个多周期指令执行完后,MCU才会执行中断程序。

中断返回亦需4个时钟。在此期间,PC将被弹出栈,SREG的位I被置位。如果在中断期间发生了其他中断,则AVR在退出中断程序后,要执行一条主程序指令之后,才能再响应被挂起的中断。

要注意AVR硬件在中断或子程序中并不操作状态寄存器——SREG。SREG的存储由用户软件完成。对于由可以保持为静态的事件(如输出比较寄存器1与T/C1值相匹配)驱动的中断,事件发生后中断标志将置位。如果中断标志被清除而中断条件仍然存在,则标志只有在新事件发生后才会置位。外部电平中断会一直保持到中断条件结束。

8.6 MCU控制寄存器——MCUCR

	位 7	6	5	4	3	2	1	0	
$35($55)	—	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
读/写:	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

初始化值:$00

位7——Res:保留位。

位6——SE:休眠使能。

执行SLEEP指令时,SE必须置位,才能使MCU进入休眠模式。为了防止无意间使MCU进入休眠,建议与SLEEP指令相连使用。

位 5,4——SM1,SM0:休眠模式。

这 2 位用于选择休眠模式,如表 8.2 所列。

表 8.2　休眠模式选择

SM1	SM0	休眠模式
0	0	空　闲
0	1	保　留
1	0	掉　电
1	1	省　电

位 3,2——ISC11,ISC10:中断 1 检测控制,位 1 和位 0。

选择 INT1 中断的边沿或电平,如表 8.3 所列。

表 8.3　中断 1 检测控制

ISC11	ISC10	描　述
0	0	低电平中断
0	1	保　留
1	0	下降沿中断
1	1	上升沿中断

注意:改变 ISC11/ISC10 时,首先要禁止 INT1(清除 GIMSK 的 INT1 位);否则可能引发不必要的中断。

位 1,0——ISC01,ISC00:中断 0 检测控制,位 1 和位 0。如表 8.4 所列。

表 8.4　中断 0 检测控制

ISC01	ISC00	描　述
0	0	低电平中断
0	1	保　留
1	0	下降沿中断
1	1	上升沿中断

注意:改变 ISC01/ISC00 时,首先要禁止 INT0(清除 GIMSK 的 INT0 位);否则可能引发不必要的中断。

INTn 引脚的电平在检测边沿之前采样。如果边沿中断使能,则大于 1 个 MCU 时钟的脉冲将触发中断。如果选择了低电平触发,则此电平必须保持到当前执行的指令结束。

8.7　中断应用举例——打印机接口设计

打印机是计算机的主要外围设备之一,用来把测量、运算结果或程序清单打印出来,有些打印机还可以打印表格和图形。打印机的种类很多,有字符式、针式、激光、笔描及热灼式等,工作原理也各不相同,价格由 100 元到几万元不等。其内部是由一些单片机、集成电路、机械

机构及微电机等部分组成的机电一体化系统；但其与计算机接口方法基本上是相近的，目前打印机与单片机的接口大多采用标准的 Centronic 打印机接口。Centronic 接口的打印机一般采用 8 位数据线和 3 根基本的应答控制线$\overline{\text{STB}}$，BUSY，$\overline{\text{ACK}}$。$\overline{\text{STB}}$为选通信号，由单片机发出，可把数据线上的打印机数据存入打印机的缓冲区中，送满一行后启动打印机打印一行字符。BUSY 表示打印机是否处于忙的状态，如它等于 1(处于忙状态)，则不能接收新的数据；如等于 0(处于空闲状态)，则可以接收新的打印数据。$\overline{\text{ACK}}$是打印机完成一次打印后的应答信号。

以 PP40 彩色绘图打印机为例，接口时序如图 8.1 所示。

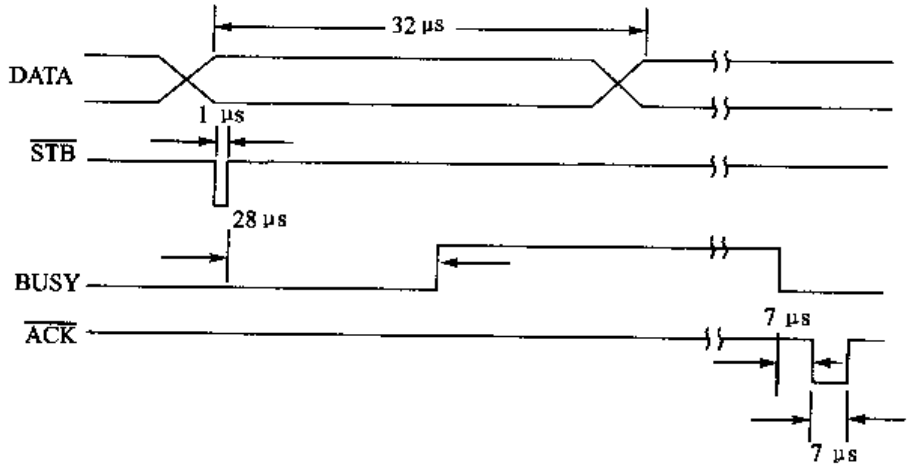


图 8.1 PP40 与主机的通信时序

8535 与 PP40 的接口电路如图 8.2 所示。

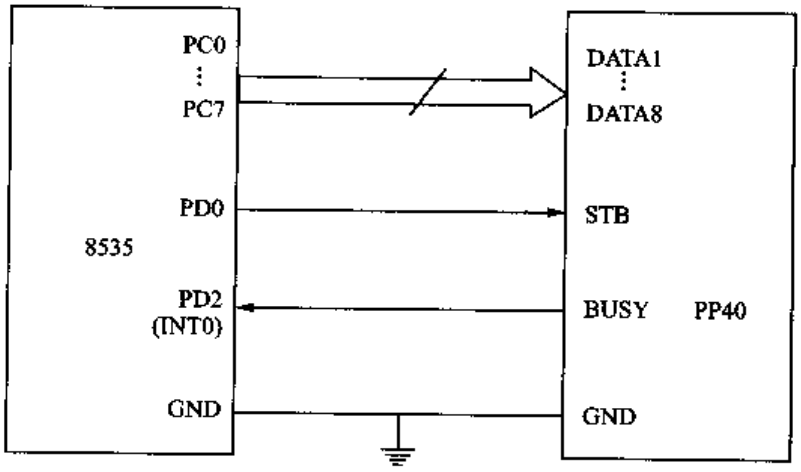


图 8.2 8535 与 PP40 接口电路

若打印如下 2 行字符：

t：32 ℃

P：102 kPa

则要把以下 ASCⅡ码送给打印机：$74(t)，$3A(:)，$20(空格)，$33(3)，$32(2)，$6F(°)，$43(C)，$0A(换行)，$50(P)，$3A(:)，$20(空格)，$31(1)，$30(0)，$32(2)，$6B(k)，$50(P)，$61(a)，$0A(换行)。事先已将这些 ASCⅡ码放在 SRAM 中 $100 开始的单元中。

给打印机送数据可以采用查询的方法。单片机每送一个数据，发选通脉冲后，打印机忙线变高，同时接收处理该数据；完成后忙线变低，单片机查到忙线变低后，再送下一个数据。采用查询的方法，程序如下：

```
.include "8535def.inc"
RESET:  ldi r16,low(ramend)        ;栈指针置初值
        out spl,r16
        ldi r16,high(ramend)
        out sph,r16
        ldi r16,$ff                ;定义 C 口为输出
        out ddrc,r16
        ldi r16,$01                ;定义 PD0 为输出,PD2 为输入
        out ddrd,r16
        sbi portd,0                ;先使 PD0 输出为高
        ldi xh,$01                 ;X 指向打印缓冲区首址
        ldi xl,$00
        ldi r25,18                 ;要打印的字符数
loop:   ld r24,x+                  ;向打印机数据口送 1 个字符
        out portc,r24
        cbi portd,0                ;发选通脉冲(STB)
        rcall t1us
        sbi portd,0
        rcall t1us                 ;延时 3 μs
        rcall t1us
        rcall t1us
loop1:  sbic pind,2                ;等待忙线变低
        rjmp loop1
        dec r25                    ;字节数是否发完
        brne loop                  ;没发完再发下一个
here:   rjmp here
```

打印机中的微电机和机械的动作是一个慢过程，需要几十毫秒才能传输一个字节数据，用查询的方法送一组数据给打印机有时需要数秒。这样单片机在这段时间内就不能干别的事情，有些情况是不允许的。可采用中断的方法，其工作过程如下。

主程序送第一个数据给打印机数据口，接着发选通脉冲；打印机接收处理该数据时，忙线变高，直到处理好此数据后，忙线变低；BUSY 的下降沿产生一个 INT0 中断，在中断服务子程序中再送下一个数据，发送选通脉冲后，立即返回主程序。这样，送字符和发选通信号是在外中 0 服务子程序中完成，单片机主程序照常执行。每送一个字符，只打断几微秒。采用中断的方法程序如下：

```
.include "8535def.inc"
.org $000
rjmp RESET
```

```
    rjmp EXT_INT0
RESET:  ldi r16,low(ramend)        ;栈指针置初值
        out spl,r16
        ldi r16,high(ramend)
        out sph,r16
        ldi r16,$ff                ;定义 C 口为输出
        out ddrc,r16
        ldi r16,$01                ;定义 PD0 为输出,PD2 为输入
        out ddrd,r16
        sbi portd,0                ;先使 PD0 输出为高
        ldi r16,$02                ;定义 INT0 下降沿申请中断
        out mcucr,r16
        clr r16                    ;清中断标志寄存器
        out gifr,r16
        ldi xh,$01                 ;X 指向打印缓冲区首址
        ldi xl,$00
        ldi r25,17                 ;要打印的字符数
        ld r24,x+                  ;向打印机数据口送第一个字符
        out portc,r24
        cbi portd,0                ;发选通脉冲(STB)
        rcall t1us
        sbi portd,0
        ldi r24,$40                ;使能 INT0 中断
        out gimsk,r24
        sei                        ;开中断
here:   rjmp here

EXT_INT0: in r1,sreg               ;保护标志寄存器
        ld r24,x+                  ;向打印机数据口送字符
        out portc,r24
        cbi portd,0                ;发选通脉冲(STB)
        rcall t1us
        sbi portd,0
        dec r25                    ;是否发完
        brne ext_int01
        ldi r24,$00                ;发完则关 INT0 中断
        out gimsk,r24
ext_int01:
        out sreg,r1                ;恢复标志寄存器
        reti
```

第 9 章　8535 单片机定时器/计数器及其应用

AT90S8535 单片机有 3 个通用定时器/计数器，即 2 个 8 位的定时器/计数器(T/C0 和 T/C2)、1 个 16 位的定时器/计数器(T/C1)。定时器/计数器 0(T/C0)和定时器/计数器 1(T/C1)从同一个 10 位的预分频定时器取得预分频时钟；定时器/计数器 2(T/C2)有自己独立的预分频器，可以选择异步外部时钟。定时器/计数器常用作带内部时钟的时基定时器或用作外部引脚上的脉冲计数器，其中有些还具有输入捕获、比较匹配、PWM 脉宽调制输出等功能。

AT90S8535 单片机还有一个看门狗定时器(WDT)，用于程序的抗干扰。

9.1　定时器/计数器 0 和定时器/计数器 1 的预定比例器

图 9.1 所示为通用定时器/计数器的预定比例器。

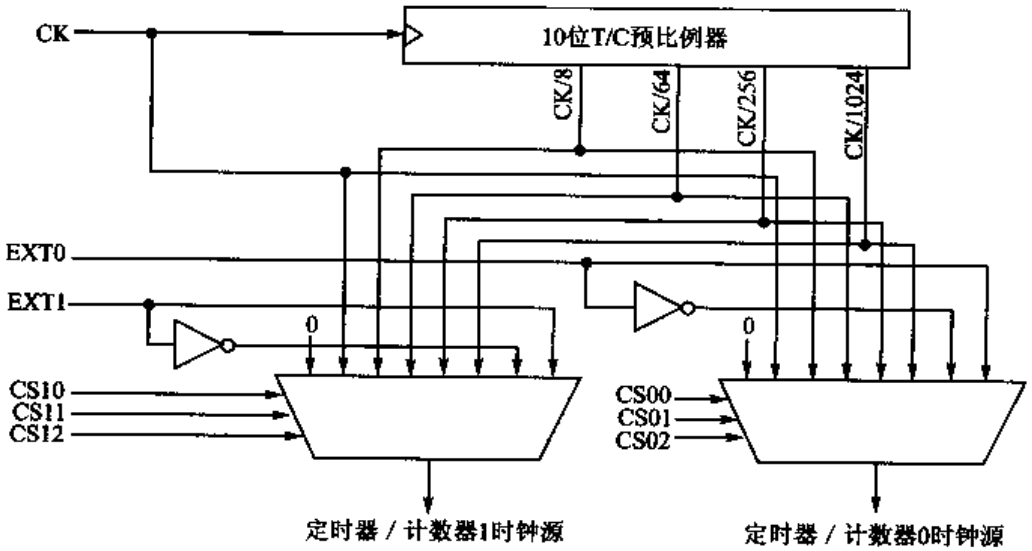


图 9.1　通用定时器/计数器的预定比例器

2 个定时器/计数器的时钟可选 CK 或 4 种不同的预定比例 CK/8，CK/64，CK/256 及 CK/1024；还可选外部时钟和定时器/计数器停止不用。

9.2　定时器/计数器 0

9.2.1　定时器/计数器 0 的结构特点和作用

图 9.2 所示为定时器/计数器 0 的方框图。它为 8 位加 1 计数器，由 $00 开始计数，计到 $FF 后再来一时钟则溢出，计数器清 0。可用作定时和计数。用作定时，时钟来自晶振时钟

CK或其4种分频，由于时钟频率准确，溢出的时间间隔是准确的；用作外计数时，外部引脚T0输入信号，可选上升沿或下降沿计数；另外，定时器/计数器0还可以停止不用。

定时器/计数器0的控制寄存器——TCCR0控制定时器/计数器0的工作方式。溢出状态标志位在定时器/计数器中断标志寄存器——TIFR中。定时器/计数器0的中断使能/禁止位设置在定时器/计数器中断的控制屏蔽寄存器——TIMSK中。当定时器/计数器0用T0引脚外计数时，为了确保CPU对外部信号获取正确的采样，外部信号两种电平转换之间的最少时间必须维持1个内部CPU的时钟周期。外部时钟信号是在内部CPU时钟的上升边沿被采样的；所以外部信号高、低电平时间均应大于1个CPU时钟周期。若外部信号是对称方波，其信号最高频率也应低于时钟频率的1/2。

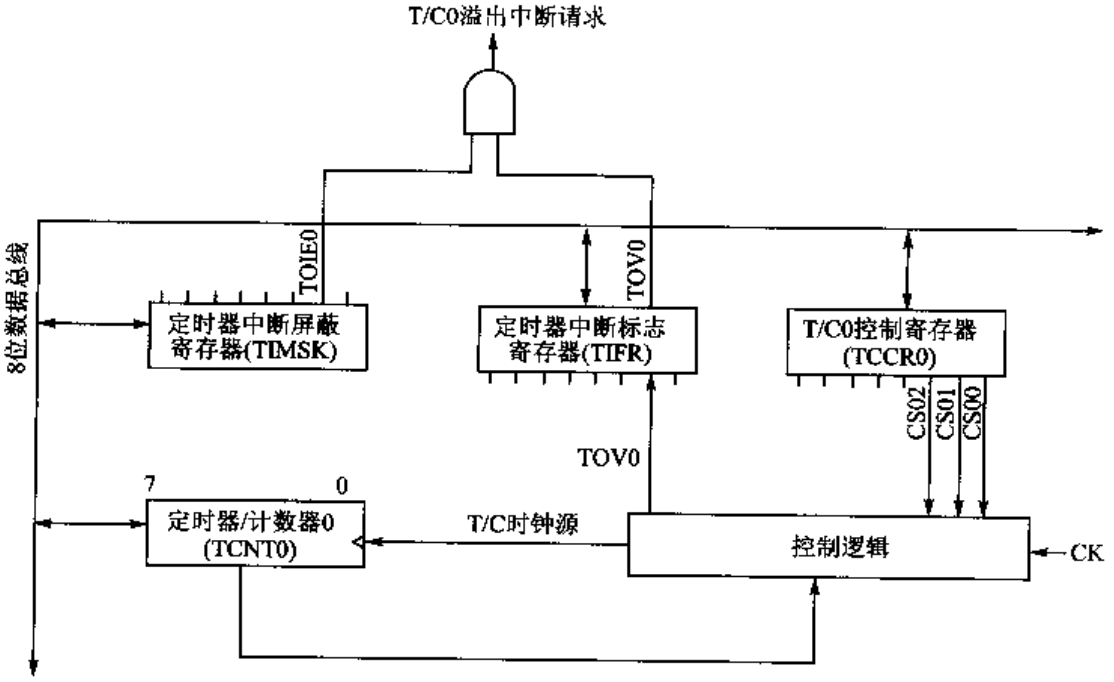


图9.2 定时器/计数器0方框图

9.2.2 定时器/计数器0有关的I/O寄存器

1. 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：$00

位7～3——Res：保留位。

这些位为保留位，总读0。

位2，1，0——CS02，CS01，CS00：T/C0时钟选择位2，1及0。

T/C0时钟选择的位2，1及0定义定时器/计数器0的预定比例源，见表9.1。

表 9.1　T/C0 时钟预定选择

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/1024
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

2. 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：$00

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。它是 8 位计数器,由 $00 开始计数,计到 $FF 后再来一时钟则溢出,计数器清 0。定时器/计数器 0 溢出后,定时器/计数器中断标志寄存器(TIFR)中的 TOV0 位置 1。若定时器/计数器中断屏蔽寄存器(TIMSK)中的定时器/计数器 0 溢出中断使能位(TOIE0)为 1,且 SREG 中的 I 位为 1,则可产生定时器/计数器 0 溢出中断。

9.3　定时器/计数器 0 应用举例

1. T/C0 作计数器

脉冲信号从 PB0(T0)引脚输入,T/C0 作计数器,计数结果由 PC 口以二进制从发光二极管显示输出。电路如图 9.3 所示。

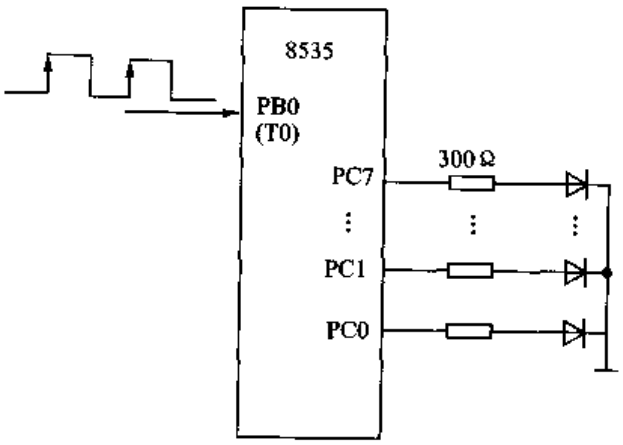


图 9.3　T/C0 外计数发光二极管显示电路图

程序如下。

```
.include "8535def.inc"
.org $000
 rjmp main
main:   ldi r16,low(ramend)        ;栈指针置初值
        out spl,r16
        ldi r16,high(ramend)
        out spH,r16
        ldi r16,$07                ;上升沿计数
        out tccr0,r16
        ldi r16,0                  ;T/C0 置初值 0
        out tcnt0,r16
        ldi r16,$ff                ;PC 口作输出
        out ddrc,r16
loop:   in r16,tcnt0
        out portc,r16
        rjmp loop
```

2. T/C0 作定时器

T/C0 作定时器,8 MHz 晶振,1 024 分频,128 μs 计 1 个数。T/C0 初值为 131,每计 125 个数(16 ms),T/C0 溢出 1 次。其中断服务子程序使 PC0 改变方向,产生 32 ms 的对称方波。程序如下。

```
.include "8535def.inc"
.org $000
rjmp main
.org $009
rjmp tim0_ovf
main:   ldi r16,low(ramend)        ;栈指针置初值
        out spl,r16
        ldi r16,high(ramend)
        out spH,r16
        ldi r16,$01                ;允许 T0 溢出中断
        out timsk,r16
        ldi r16,$05                ;1 024 分频
        out tccr0,r16
        ldi r17,131                ;T/C0 置初值 131
        out tcnt0,r17
        out tifr,r16
        ldi r16,$ff                ;PC 口作输出
        out ddrc,r16
        sei
herc:   rjmp here
tim0_ovf: in r1,sreg               ;保存 sreg
```

```
        ldi r17,131
        out tccr0,r17
        in r18,portc            ;读 C 口数据寄存器
        com r18                 ;取反
        out portc,r18           ;送 C 口数据寄存器
        out sreg,r1             ;恢复 sreg
        reti
```

改变分频系数和中断服务子程中给 T/C0 的初值，均可改变中断时间间隔。

3. T/C0 溢出中断动态扫描 5 位数码管显示

7.2.2 节所述的数码管动态扫描程序是在主程序中个、十、百、千及万位采用延时的方法，每位分时输出字线和位线，占用了程序 90%以上的时间，在快速系统中这是不允许的。下面介绍一种采用 T/C0 溢出中断的方法，每 2 ms 溢出中断一次，在中断服务程序中轮流改变字线和位线。显示某位，随即返回主程序，可以不占用主程序大量时间来管理动态扫描程序。主程序只需把要显示的各位 7 段码放到相应的 SRAM 中即可。电路同图 7.3 所示。主程序把要显示的 16 位二进制数放在 r17:r16 中，经二转十、查 7 段码放在 $100～$104 中，位线码送初值 $FE，并对 T/C0 溢出中断初始化。在中断服务程序中判位线码，该显示哪位就送相应的字线和位线，再修改位线码，为下次中断做好准备，然后立刻返回主程序。2 ms 以后，T/C0 又溢出中断，送字线和位线，开始显示下一位，并修改位线码，中断返回。这样个、十、百、千及万位循环反复显示，每位显示 2 ms，10 ms 显示 1 遍(5 位)，每秒显示 100 遍。这样可以看到 5 位稳定的数码显示。T/C0 溢出中断占用的时间只是几微秒，绝大多数时间都可以给主程序使用。

```
.include "8535def.inc"
.org $0000
rjmp reset
.org $009
rjmp tim0_ovf
tab:.db $3f,$06,$5b,$4f,$66,$6d,$7d,$07,$7f,$6f
reset:  ldi r16,low(ramend)     ;栈指针置初值
        out spl,r16
        ldi r16,high(ramend)
        out sph,r16
        ldi r16,$ff             ;定义 PB,PD 为输出口
        out ddrb,r16
        out ddrd,r16
        ldi r17,$ff             ;设初值在 r17:r16
        ldi r16,$ff
        rcall b16td             ;调用二转十子程
        mov r23,r16             ;查 7 段码,送给 $100～$104
        rcall cqm1
        sts $100,r0
        mov r23,r17
```

```
        rcall cqm1
        sts $101,r0
        mov r23,r18
        rcall cqm1
        sts $102,r0
        mov r23,r19
        rcall cqm1
        sts $103,r0
        mov r23,r20
        rcall cqm1
        sts $104,r0
        ldi r16,$01          ;允许 T/C0 溢出中断
        out timsk,r16
        ldi r16,$03          ;64 分频,2 ms 1 位
        out tccr0,r16
        ldi r16,$00          ;T/C0 置初值 0
        out tcnt0,r16
        out tifr,r16
        ldi r21,$fe          ;位线置初值
        sei
here:   rjmp here
tim0_ovf:
        in r1,sreg           ;保存 SREG
        cpi r21,$fe          ;该显示个位?
        brne t21             ;否则转 t21
        lds r20,$100         ;送个位 7 段码给字线
        out portb,r20
        out portd,r21        ;送个位位线
        ldi r21,$fd          ;修改位线(下次显示十位)
        rjmp t25
t21:    cpi r21,$fd          ;该显示十位?
        brne t22             ;否则转 t22
        lds r20,$101         ;送十位 7 段码给字线
        out portb,r20
        out portd,r21        ;送十位位线
        ldi r21,$fb          ;修改位线(下次显示百位)
        rjmp t25
t22:    cpi r21,$fb          ;该显示百位?
        brne t23             ;否则转 t23
        lds r20,$102         ;送百位 7 段码给字线
        out portb,r20
        out portd,r21        ;送百位位线
        ldi r21,$f7          ;修改位线(下次显示千位)
```

```
        rjmp t25
t23:    cpi r21,$f7          ;该显示千位?
        brne t24             ;否则转 t24
        lds r20,$103         ;送千位 7 段码给字线
        out portb,r20
        out portd,r21        ;送千位位线
        ldi r21,$ef          ;修改位线(下次显示万位)
        rjmp t25
t24:    lds r20,$104         ;送万位 7 段码给字线
        out portb,r20
        out portd,r21        ;送万位位线
        ldi r21,$fe          ;修改位线(下次显示个位)
t25:    out sreg,r1          ;恢复 sreg
        reti
```

9.4　定时器/计数器 1

9.4.1　定时器/计数器 1 的结构、特点及作用

图 9.4 所示为定时器/计数器 1 的方框图。

定时器/计数器 1 是 16 位加 1 计数器。它可以用于定时(时钟源选择 CK 或其分频),也可用于对外部引脚 T1 脉冲信号计数,还可以停止不用。这些与定时器/计数器 0 功能一样,只不过定时器/计数器 1 是 16 位计数器,计数值为 $0000～$FFFF,再加 1 则溢出,且计数器清 0。由定时器/计数器 1 控制寄存器——TCCR1B 中的低 3 位确定是外计数,还是定时,及其对主频的分频系数。溢出状态标志位在定时器/计数器中断标志寄存器——TIFR 中。定时器/计数器 1 的中断使能/禁止位设置在定时器/计数器中断的控制屏蔽寄存器—— TIMSK 中。当定时器/计数器 1 外计数时,为了确保 CPU 对外部信号获取正确的采样,外部信号 2 种电平转换之间的最少时间必须维持 1 个内部 CPU 的时钟周期。由于外部时钟信号是在内部 CPU 时钟的上升边沿被采样的,所以外部信号高、低电平时间均应大于 1 个 CPU 时钟周期。若外部信号是对称方波,其信号频率也应低于时钟频率的 1/2。

此外,定时器/计数器 1 还具有比较匹配输出功能。定时器/计数器 1 内有 2 个输出比较寄存器——OCR1A 和 OCR1B。T/C1 的值与其中一个相等时,相应比较输出引脚自动产生跳变,还可清除 T/C1。这种比较匹配输出功能可实现使引脚在事先预定的时刻发生跳变或中断,而不用 CPU 随时关照,以节省 CPU 的时间。

定时器/计数器 1 可用作 8 位、9 位或 10 位 PWM 脉冲调制器。在此模式下,定时器和 OCR1A/OCR1B 寄存器用于 2 个无尖峰干扰的中心对称的 PWM。PWM 脉冲经滤波,可得到模拟电压信号。改变 PWM 脉冲的占空比,即可改变模拟电压的大小。这种 PWM 脉冲还便于隔离抗干扰,只用 1 个光耦隔离即可。

定时器/计数器 1 还具有输入捕获功能。它内有输入捕获寄存器——ICR1,当输入引脚发生规定的跳变时,T/C1 的值被传送输入捕获寄存器——ICR1 中。这样,不用 CPU 随时关

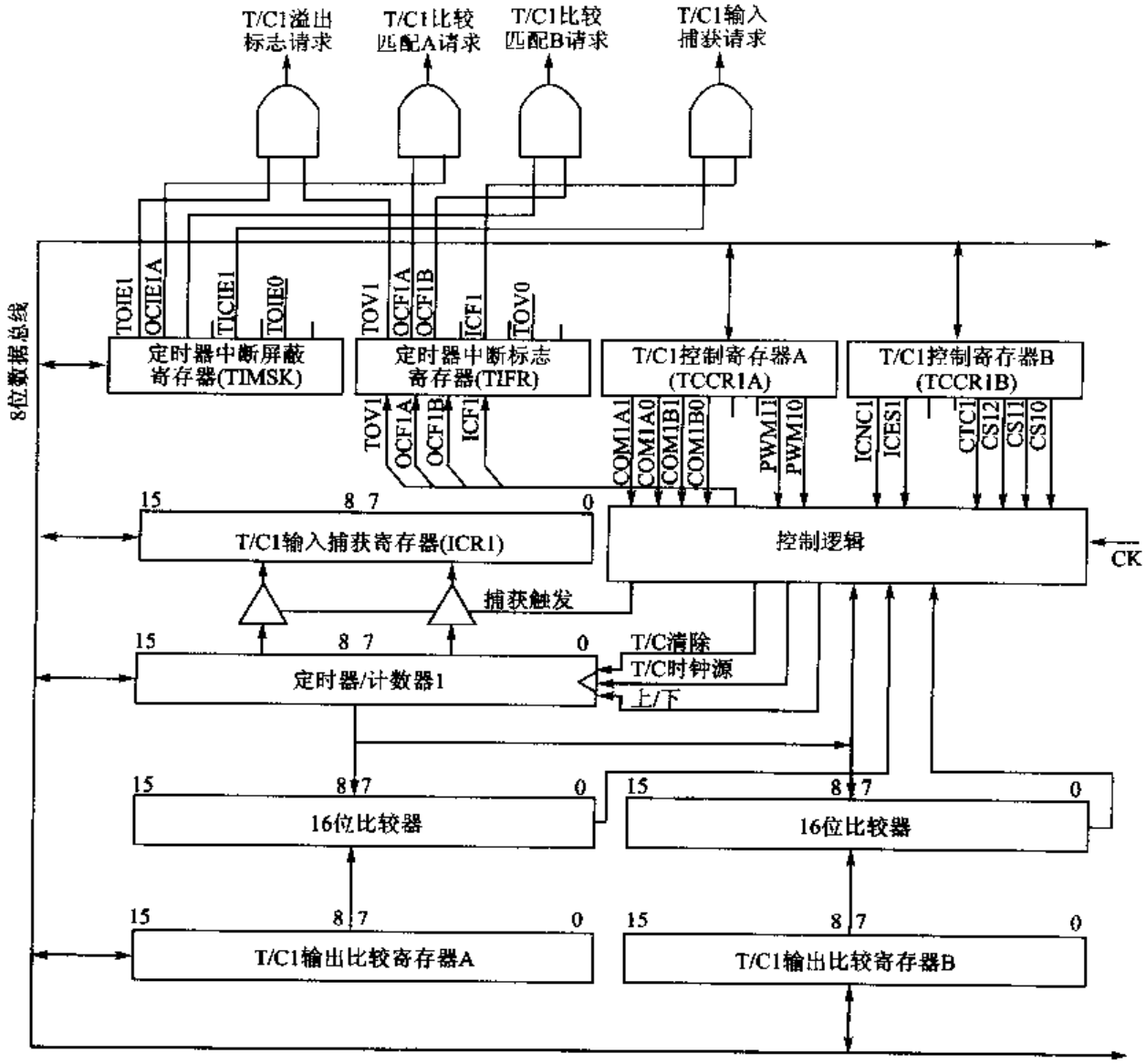


图 9.4 定时器/计数器 1 方框图

照，可自动记录事件发生时间，以节省 CPU 的时间。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，模拟比较器也可触发该输入捕获。请参照“模拟比较器”部分。ICP 的引脚逻辑如图 9.5 所示。

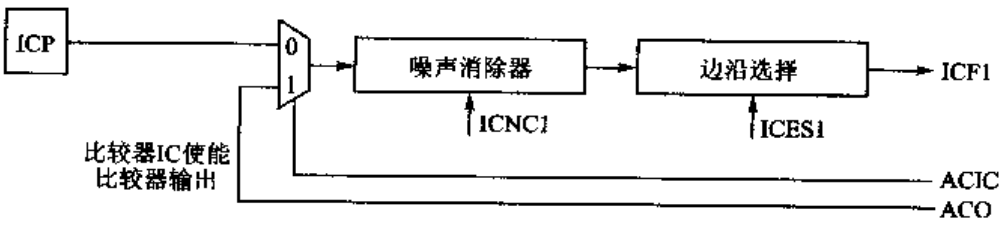


图 9.5 ICP 的引脚原理图

如果噪音清除器使能，则触发信号要进行 4 次采样。只有当 4 个采样值都相等时，才会触发捕获标志。输入引脚信号以 XTAL 的时钟频率被采样。

9.4.2　定时器/计数器 1 有关的 I/O 寄存器

1. 定时器/计数器 1 控制寄存器 A——TCCR1A

	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：$00

位 7,6——COM1A1,COM1A0：比较输出模式 1A，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 9.2 所列。

位 5,4——COM1B1,COM1B0：比较输出模式 1B，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 9.2 所列。

在 PWM 模式下，这些位有不同的功能，请参考表 9.6 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 9.3 中指明的、选择定时器/计数器 1 的 PWM 操作。

表 9.2　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 9.3　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

2. 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：$00

位7——ICNC1：输入捕获噪音清除器(4CKs)。

当ICNC1位被清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，即在输入捕获引脚上被采样的第1个上升/下降沿处被激活。当ICNC1被设为1时，4个延续的采样成为ICP，即输入捕获引脚上的测量值，所有的采样需为高/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0时，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时，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，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及0。

时钟选择1的位2,1及0定义了定时器/计数器1的预定比例源，见表9.4

表9.4 时钟预定比例选择

CS12	CS11	CS10	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/1024
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

3. 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：$00 00

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 TCNT1H 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 1 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。

4. 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：$00 00

5. 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：$00 00

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，以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，OCR1BL 或 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。

6. 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	
$27($47)	MSB			—					ICR1H
$26($46)								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：$00 00

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 TEMP 寄存器中的数据。作为结果，低位字节 ICR1L 必须先被访问到，以便完成一个全 16 位寄存器的读取操作。

7. 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 位无尖峰、自运行的 PWM。定时器/计数器 1 作为向上/向下的计数器，从 $000 计到顶，然后反向减到 $000，不断循环重复。当计数器中的值和 OCR1A/OCR1B 的值(低 8,9,10 位)相匹配时，OC1A/OC1B 引脚按照定时器/计数器 1 控制寄存器 TCCR1A 中 COM1A1/COM1A0 或 COM1B1/COM1B0 位的设置而动作(被设置或被清除)。

表 9.5 为定时器 TOP 值和 PWM 频率，表 9.6 为在 PWM 方式时比较 1 方式选择。

表 9.5 定时器 TOP 值和 PWM 频率

PWM 分辨率	定时器 TOP 值	频 率
8 位	$00FF(255)	$f_{TC1}/510$
9 位	$01FF(511)	$f_{TC1}/1022$
10 位	$03FF(1023)	$f_{TC1}/2046$

表 9.6 在 PWM 方式时比较 1 方式选择

COM1X1	COM1X0	在 OCX1 上的作用
0	0	不用作 PWM
0	1	不用作 PWM
1	0	向上计数时匹配清除 OC1；向下计数时匹配置位 OC1(正向 PWM)
1	1	向下计数时匹配清除 OC1；向上计数时匹配置位 OC1(反向 PWM)

X=A 或 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 9.6 的例子。

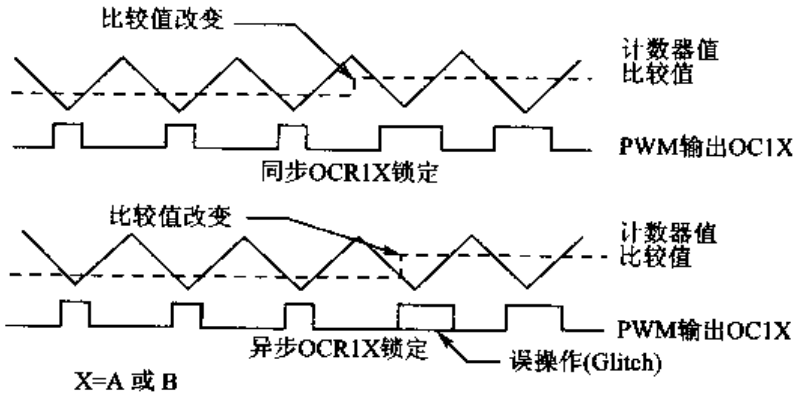


图 9.6　有效的非同步 OCR1 锁存

当 OCR1 包含 $0000 或 TOP 时，输出 OC1A/OC1B 根据 COM1A1 和 COM1A0 或 COM1B1/COM1B0 的设置保持低或高，如表 9.7 所列。

表 9.7　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

在 PWM 模式下，当计数器在方向 $0000 时，定时器溢出标志 1、TOV1 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。

9.5　定时器/计数器 1 应用举例

1. 测量脉冲频率

电路如图 9.7 所示。脉冲加到 PB1(T1)引脚，5 位数码管动态扫描显示脉冲频率。频率即单位时间的脉冲数。T/C1 用于外计数方式，每上升沿计数 1 次；T/C0 为定时方式，8 MHz 的晶振频率，256 分频，每 32 μs 计 1 个数。若 T/C0 每次置初值 6，即每计 250 次溢出 1 次，则溢出时间的间隔为 8 ms。这样每溢出 125 次即达 1 s。每隔 1 s，求出定时器/计数器 1 的增加量，即为脉冲频率。

可以用 T/C2 产生的 15 686 Hz 的 PWM 方波作为信号源来验证，程序如下。

```
.include "8535def.inc"
.org $000
rjmp main
```

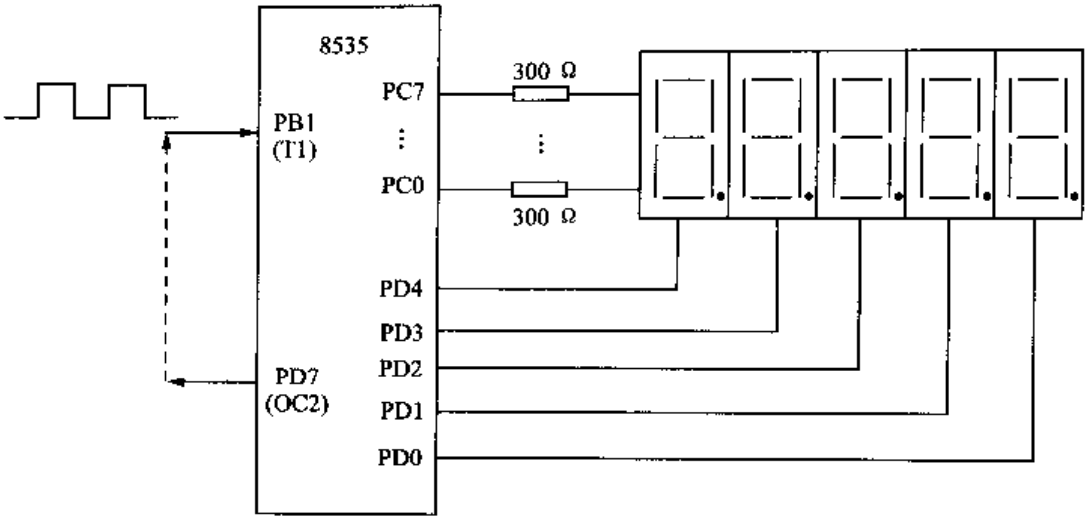


图 9.7 频率测量及显示电路图

```
.org $009
rjmp tim0_ovf
tab:.db $3f,$06,$5b,$4f,$66,$6d,$7d,$07,$7f,$6f
main:   ldi r16,low(ramend)          ;栈指针置初值
        out spl,r16
        ldi r16,high(ramend)
        out sph,r16
        ldi r16,$ff                  ;定义C口、D口为输出
        out ddrd,r16
        out ddrc,r16
        ldi r16,$01                  ;允许T/C0中断
        out timsk,r16
        ldi r16,$04                  ;定时器8分频
        out tccr0,r16
        ldi r16,6                    ;定时器0置初值0
        out tcnt0,r16
        cbi ddrb,1                   ;PB1定义为输入口
        ldi r24,$06                  ;T1(PB1)引脚每一次上升沿计数1次
        out tccr1b,r24
        ldi r27,125
        rcall t2pwm1                 ;PD7产生频率为15 686 Hz的PWM方波
        sei
loop:   mov r16,r10                  ;将TCNT1的增量值送r17:r16
        mov r17,r11
        rcall b16td                  ;二转十子程见4.3.1
        mov r22,r20                  ;送r18~r22
        mov r21,r19
        mov r20,r18
        mov r19,r17
```

```
            mov r18,r16
            rcall smiao                ;动态扫描显示子程见 7.2.2
            rjmp loop
t1ms:       ldi r24,71                 ;延时 1 ms 子程
            push r24
del2:       push r24
del3:       dec r24
            brne del3
            pop r24
            dec r24
            brne del2
            pop r24
            ret
t2pwm1:
            ldi r16,$71                ;PWM2 使能,向上计数置引脚
            out tccr2,r16              ;向下计数清引脚,时钟 1 分频
            ldi r16,$80
            out ocr2,r16
            ret
tim0_ovf:   in r1,sreg                 ;保护现场
            ldi r24,6                  ;T/C0 送初值
            out TCNT0,r24
            subi r27,1                 ;中断计数减 1
            brne qq                    ;不为 0,则返回
            in r10,tcnt1l              ;读 TCNT1 计数值到 r11:r10
            in r11,tcnt1h
            push r10                   ;入栈保存
            push r11
            sub r10,r12                ;求 2 次 T/C1 计数差值
            sbc r11,r13
            pop r13                    ;将本次 T/C1 计数值放入 r13:r12 中
            pop r12                    ;为下次计算计数器差值做准备
            ldi r27,125
qq:         out sreg,r1                ;恢复现场
            reti
```

2. T/C1 比较匹配中断

每隔 1 s 使 PC0 取反 1 次,采用 T/C1 比较匹配中断,时钟为 8 MHz,256 分频,每 32 μs 计数 1 次,1 s 需计数 31 250 次,T/C1 比较匹配值取 $7A12(即 31 250)。程序如下。

```
.include "8535def.inc"
.org $0000
rjmp main
.org $0006
```

```
    rjmp tim1_compa
main:   ldi r16,low(ramend)          ;栈指针置初值
        out spl,r16
        ldi r16,high(ramend)
        out sph,r16
        ldi r16,$01                  ;PC0 口定义为输出口
        out ddrc,r16
        ldi r16,$10                  ;允许 T1 比较匹配 A 中断
        out timsk,r16
        clr r16                      ;置 TCNT1 初值为 0
        out tcnt1l,r16
        out tcnt1h,r16
        ldi r16,$7a                  ;ocr1a 置 $7A12,即 1 s 中断 1 次
        out ocr1ah,r16
        ldi r16,$12
        out ocr1al,r16
        ldi r16,$0c                  ;T/C1 对主频 256 分频定时
        out tccr1b,r16
        sei
here:   rjmp here
tim1_compa:
        in r1,sreg                   ;保护标志
        in r2,portc                  ;PC 口取反
        com r2
        out portc,r2
        out sreg,r1                  ;恢复标志
        reti
```

采用 T/C1 溢出中断送初值 $85EE,也可以实现上述功能;但每次中断服务子程序中都要送初值。如果没有及时进入该中断服务子程,则定时的间隔时间就会变长,程序也不如比较匹配中断简短;所以一般 T/C1 和 T/C2 能用比较匹配中断就不用定时器溢出中断。

3. T/C1 比较匹配中断产生连续方波

用 T/C1 比较匹配中断产生连续方波。程序达到比较匹配值时,使 OC1A 发生跳变,并产生比较匹配中断,在中断服务子程序中,给出下次跳变的方向和时间,这样就可以产生连续的方波。用这种方法产生的方波,其高电平时间和低电平时间可以做得十分精确。产生高电平时间 4 992 μs,低电平时间隔 9 984 μs 的连续方波程序如下:

```
.include "8535def.inc"
.org $000
rjmp main
.org $006
rjmp tim1_cmp
main:   ldi r16,low(ramend)          ;初始化堆栈指针
```

```
        out spl,r16
        ldi r16,high(ramend)
        out sph,r16
        ldi r16,$c0                ;达比较匹配值时,OC1A 变高
        out tccr1a,r16
        ldi r16,$0C                ;256 分频,CTC=1,匹配时清定时器 1
        out tccr1b,r16
        ldi r16,$20                ;PD5 作输出
        out ddrd,r16
        clr r16                    ;使 TCNT1 初值为 0
        out tcnt1h,r16
        out tcnt1l,r16
        ldi r18,$01                ;送比较匹配值 312
        out ocr1ah,r18
        ldi r18,$38
        out ocr1al,r18
        ldi r16,$10                ;允许定时器 1 比较匹配中断
        out timsk,r16
        clr r16                    ;清中断标志
        out tifr,r16
        sei
here:   rjmp here

tim1_cmp:
        in r1,sreg                 ;保护标志
        in r18,tccr1a              ;读 TCCR1A
        sbrs r18,6                 ;判 COM1A0 是否为 1
        rjmp aa
        ldi r18,$00                ;若为 1,送下次比较匹配值 156
        out ocr1ah,r18
        ldi r18,156
        out ocr1al,r18
        ldi r18,$80                ;下次达到比较匹配值 156 时,OC1A 引脚变低
        out tccr1a,r18
bc:     out sreg,r1                ;恢复标志
        reti
aa:     ldi r18,$c0                ;若 COM1A0 为 0,下次达到比较匹配值时,OC1A 引脚变高
        out tccr1a,r18
        ldi r18,$01                ;送下次比较匹配值 312
        out ocr1ah,r18
        ldi r18,$38
        out ocr1al,r18
        rjmp bc
```

4. 输入捕获中断应用——测方波周期

电路如图 9.8 所示，将方波信号加到 PD6(ICP)引脚，5 位数码管动态扫描显示脉冲周期。

程序采用 T/C1 捕获中断，捕获每次上跳时 T/C1 的计数值。T/C1 为定时方式，对时钟 8 分频，每计 1 个数需 1 μs，则每 2 次捕获值之差即为方波的周期。

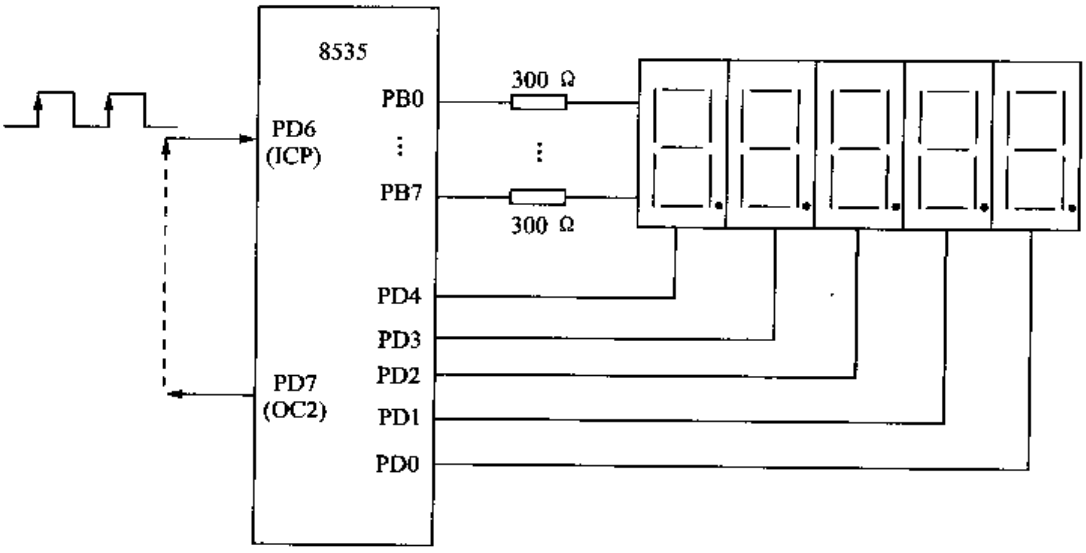


图 9.8 周期测量及显示电路图

可用 PD7(OC2)引脚产生周期为 8 160 μs 的方波作信号源，加到 ICP 引脚得以验证。程序如下。

```
.include "8535def.inc"
.org $000
 rjmp main
.org $005
rjmp tim1_capt
tab:.db $3f,$06,$5b,$4f,$66,$6d,$7d,$07,$7f,$6f   ;7 段码表
main:  ldi r16,low(ramend)                        ;栈指针置初值
       out spl,r16
       ldi r16,high(ramend)
       out sph,r16
       ldi r16,$ff                                ;定义 B 口为输出口
       out ddrb,r16
       ldi r16,0b10111111                         ;定义 PD6 为输入,其余引脚为输出
       out ddrd,r16
       rcall t2pwm2                               ;调产生周期为 8 160 μs 方波子程
       clr r12                                    ;清上次捕获值
       clr r13
       ldi r16,$20                                ;允许 TIM1_CAPT
       out timsk,r16
       ldi r16,$c2                                ;上升沿捕获,8 分频
       out tccr1b,r16
```

```
                clr r16                 ;清中断标志寄存器
                out tifr,r16
                sei                     ;CPU 开中断
aa:             mov r17,r11             ;捕获偏差值送 r17:r16
                mov r16,r10
                rcall b16td5            ;调二转十子程见 4.3.1
                mov r22,r20             ;送显示缓冲区
                mov r21,r19
                mov r20,r18
                mov r19,r17
                mov r18,r16
                rcall smiao             ;动态扫描子程见 7.2.2
                rjmp aa
tim1_capt:      in r1,sreg              ;标志入栈
                in r10,icr1l            ;读输入捕获寄存器且将其入栈保护
                in r11,icr1h
                push r10
                push r11
                sub r10,r12             ;减其上次捕获时值
                sbc r11,r13
                pop r13                 ;将本次捕获值差送入 r11:r10 中
                pop r12
                out sreg,r1
                reti

t2pwm2:
                ldi r16,$75             ;定义 PWM2 使能,向上计数置引脚
                out tccr2,r16           ;向下计数清引脚,时钟 128 分频
                ldi r16,$80
                out ocr2,r16
                ret
```

5. PWM 输出作 D/A 转换器

PD4(OC1B),PD5(OC1A)可以产生 PWM 脉宽调制输出,PWM 的精度、周期、相位及占空比都是可以改变的。下面以 OC1B 为例产生 10 位的 PWM 方波,经滤波输出模拟电压。电路如图 9.9 所示。

正向 PWM 方波的占空比为(r19:r18)/$3FF,经 2 级 CMOS 反向器 4049 限幅,其 0 电平为 0 V,1 电平为 V_{CC},经滤波可产生模拟电压输出。为减少输出阻抗,可加电压跟随器。改变占空比的大小,就成比例地改变了电压的输出。

$$V_{OUT} = V_{CC} \times (r19:r18)/1\ 023$$

若使输出与主机隔离,可采用图 9.10 所示的电路。用一个光电耦合器,即可实现与主机隔离;但光耦三极管后面的电路必须另用一组隔离的电源供电。若输出 4～20 mA 的电流,则

把跟随器改为 V/I 转换电路即可。

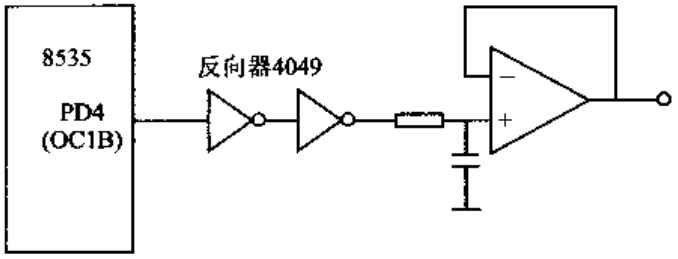


图 9.9 DAC 模拟电压输出电路图

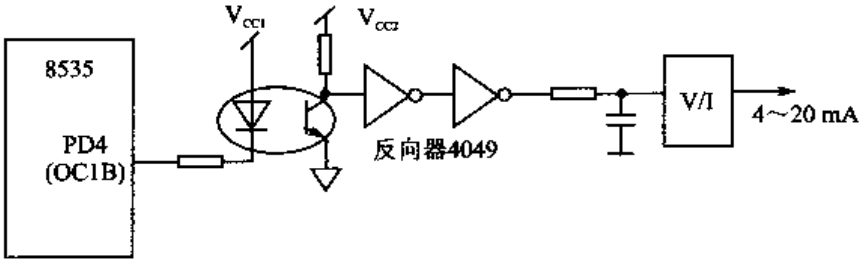


图 9.10 DAC 4～20 mA 电流输出简化电路

程序如下：

```
.include "8535def.inc"
        ldi r16,$01             ;T1 不分频
        out TCCR1B,r16
        ldi r16,$23             ;OC1B 口 10 位正向 PWM 输出
        out TCCR1A,r16
        sbi DDRD,4              ;PD4(OC1B)引脚定义为输出
        ldi r18,0               ;设占空比为 $200/$3FF
        ldi r19,2
        out OCR1BH,r19
        out OCR1BL,r18
here:   rjmp here
```

9.6 定时器/计数器 2

9.6.1 定时器/计数器 2 的预分频器

图 9.11 所示为定时器/计数器 2 的预定比例器。

T/C2 的时钟源称为 PCK2。若清 0 ASSR 的 AS2 位，PCK2 与系统主时钟连接；若置位 ASSR 的 AS2 位，T/C2 将由 PC6(TOSC1)异步驱动，使得 T/C2 可以作为一个实时时钟。如果 AS2 置位，则 PC6(TOSC1)和 PC7(TOSC2)从 C 口脱离。引脚上既可外接一个时钟晶振，也可以在 PC6(TOSC1)上直接施加时钟信号。此时钟频率必须低于 MCU 主时钟的 1/4，并且不能高于 256 kHz。

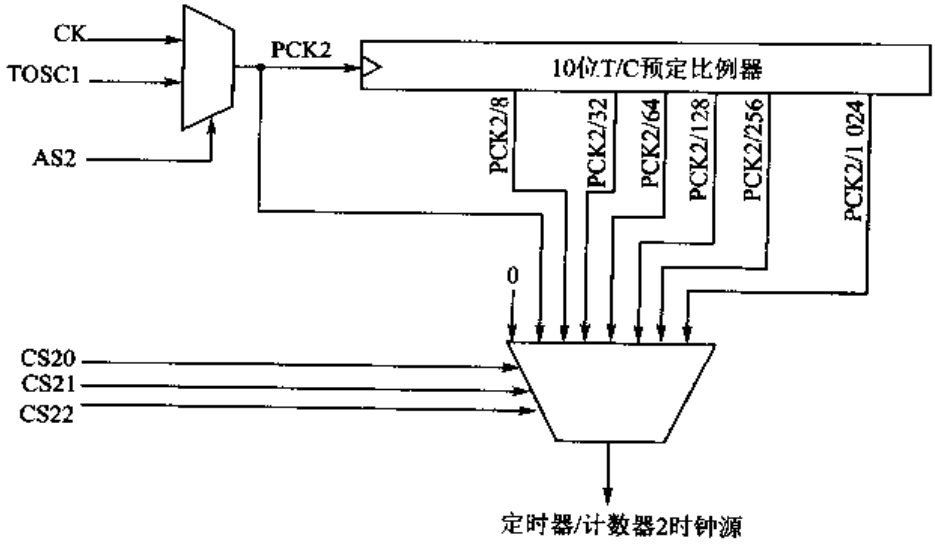


图 9.11　T/C2 的预分频器

9.6.2　定时器/计数器 2 的结构、特点及作用

图 9.12 所示为 T/C2 的框图。

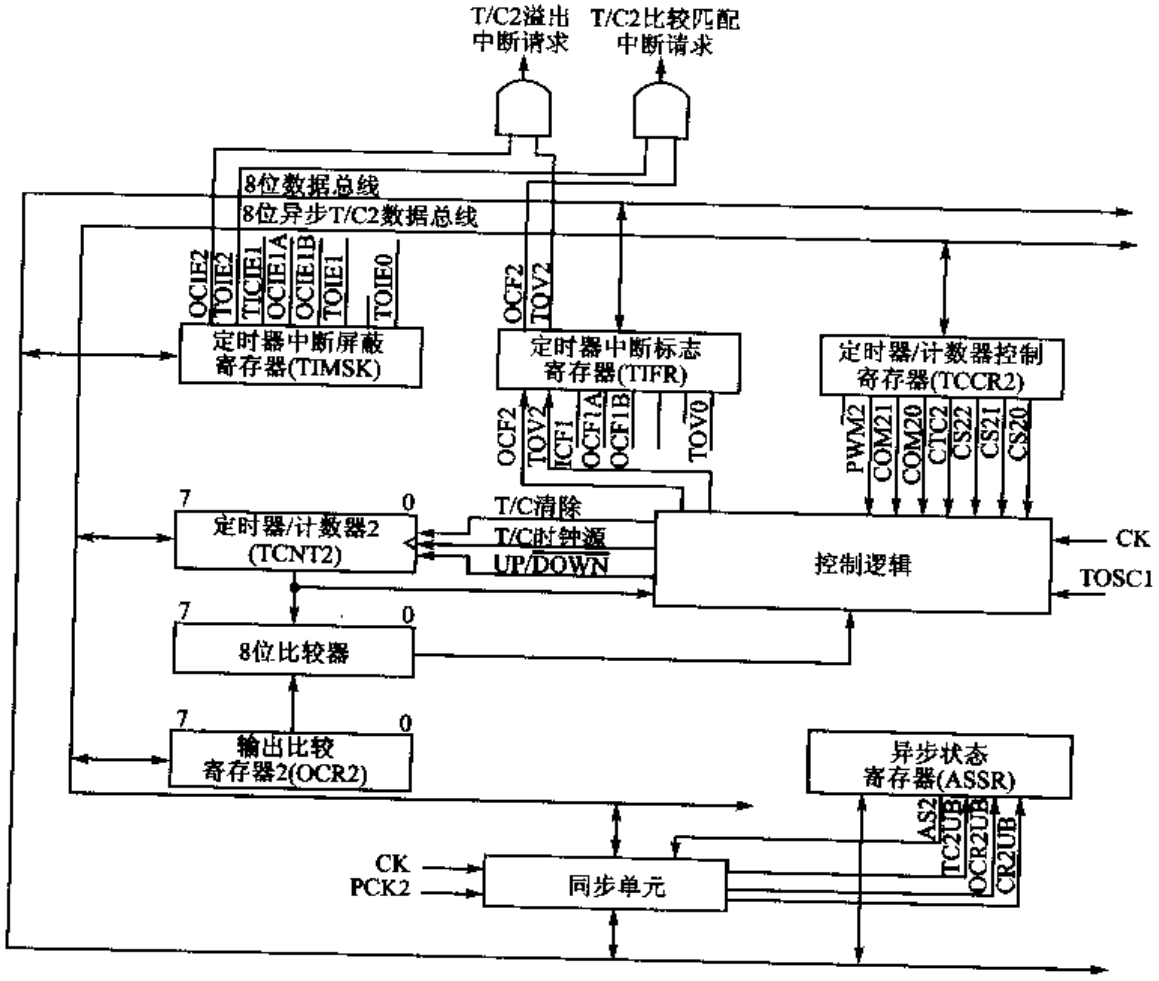


图 9.12　T/C2 工作框图

T/C2 的时钟可以选择 PCK2 或 6 种预分频的 PCK2，另外还可以停止不用，这由 T/C2 控制寄存器 TCCR2 来控制。T/C2 可以用来定时、外计数及停止。定时可用主时钟或外接实时时钟，将 32 768 Hz 的晶振接在 PC6(TOSC1)和 PC7(TOSC2)引脚上；外计数时上述引脚不接晶振，输入脉冲由 PC6(TOSC1)引脚输入。TIFR 为状态标志寄存器，TCCR2 为控制寄存器，而 TIMSK 控制 T/C2 的中断屏蔽。

T/C2 还可以实现比较匹配的输出功能。当 T/C2 的值增加到与比较寄存器 OCR2 相等时，清除 T/C2 和比较输出引脚 PD7(OC2)的跳变。

T/C2 还可以用作 8 位 PWM 调制器。在此模式下，计数器和 OCR2 寄存器用于无尖峰干扰的、中心对称的 PWM。

9.6.3 定时器/计数器 2 有关的 I/O 寄存器

1. T/C2 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/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

初始化值：$00

位 7——Res：保留位。

位 6——PWM2：PWM 使能。

位 5,4 · COM21/20：比较匹配模式位 1/0。

表 9.8 所列为比较匹配模式选择。

表 9.8 比较匹配模式选择

COM21	COM20	OC2
0	0	不用作 PWM 功能
0	1	输出变换
1	0	清 0
1	1	置位

位 3——CTC2：比较匹配时清除 T/C2。

CTC2 为“1”时，比较匹配事件发生后，TCNT2 将复位为 0。若 CTC2 为“0”，则 T/C2 将连续计数而不受比较匹配的影响。由于比较匹配事件的检测发生在匹配发生之后的一个 CPU 时钟，故而定时器的预分频比率的不同，将引起此功能有不同的表现。当预分频为 1、比较匹配寄存器的值设置为 C 时，定时器的计数方式为：

…| C−2 | C−1 | C | 0 | 1 |…

而当预分频为 8 时，定时器的计数方式则为：

…| C−2, C−2, C−2, C−2, C−2, C−2, C−2, C−2 | C−1, C−1, C−1, C−1, C−1, C−1, C−1, C−1 | C, 0, 0, 0, 0, 0, 0, 0 |…

在 PWM 模式下，这一位没有作用。

位 2,1,0——CS22,CS21,CS20：时钟选择。

表 9.9 为 T/C2 预分频选择。

表 9.9　T/C2 预分频选择

CS22	CS21	CS20	描　述
0	0	0	停止
0	0	1	PCK2
0	1	0	PCK2/8
0	1	1	PCK2/32
1	0	0	PCK2/64
1	0	1	PCK2/128
1	1	0	PCK2/256
1	1	1	PCK2/1 024

停止条件提供了一个定时器使能/禁止的功能。预分频的 PCK2 直接由时钟振荡器分频而来。

2. T/C2 计数器——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：$00

T/C2 是可以进行读/写访问的 8 位向上计数器。只要有时钟输入，T/C2 就会在写入值的基础上向上计数。

3. T/C2 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：$00

T/C2 输出比较寄存器包含与 T/C2 值连续比较的数据。如果 T/C2 的值与 OCR2 相等，则比较匹配发生，结果由 TCCR2 决定。用软件写操作，将 TCNT2 和 OCR2 设置为相等，不会引发比较匹配。匹配发生后，T/C 中断标志寄存器 TIFR 中的匹配中断标志 OCF2 置位。

9.6.4　PWM 模式下的 T/C2

选择 PWM 模式后，T/C2 和输出比较寄存器 OCR2 共同组成一个 8 位的、无尖峰的、自由运行的 PWM。T/C2 作为上/下计数器，从 0 计数到 TOP，然后反向计数回到 0。当计数器中的数值和 OCR2 的数值一致时，OCR2 引脚按照 COM21/COM20 的设置动作。表 9.10 为 PWM 模式下的比较模式选择。

表 9.10 PWM 模式下的比较模式选择

COM21	COM20	OC2
0	0	不用作 PWM 功能
0	1	不用作 PWM 功能
1	0	向上计数时的匹配清除 OC2；而向下计数时的匹配置位 OC2(正向 PWM)
1	1	向下计数时的匹配清除 OC2；而向上计数时的匹配置位 OC2(反向 PWM)

注意：在 PWM 模式下，OCR2 首先存储在一个临时的位置，等到 T/C2 达到 TOP 时，才真正存入 OCR2。这样可以防止在写 OCR2 时，由于失步而出现奇数长度的 PWM 脉冲。图 9.13 为失步的 OCR2 锁存。

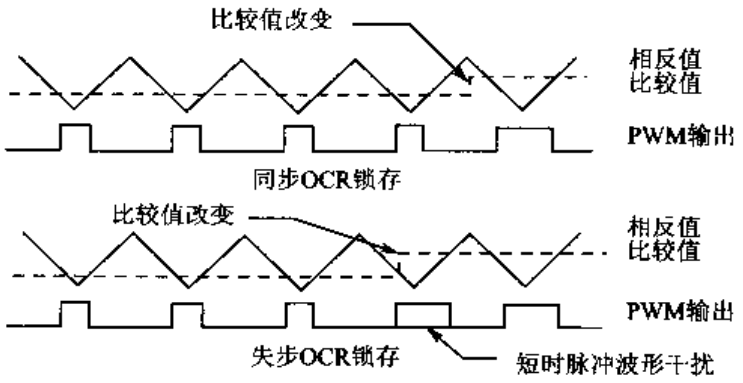


图 9.13 失步的 OCR2 锁存

如果在执行写和锁存操作时读取 OCR2，读到的是临时位置的数据。OCR2 的值为 $0000 或 TOP 时，PWM 的输出见表 9.11。

表 9.11 OCR2 = $0000 或 TOP 时的 PWM 输出

COM21	COM20	OCR2	输 出
1	0	$0000	L
1	0	TOP	H
1	1	$0000	H
1	1	TOP	L

在 PWM 模式下，当计数器达到 $00 时，将置位 TOV2。此时发生的中断与正常情况下的中断是完全一样的。

9.6.5 异步时钟信号的驱动

1. 异步状态寄存器——ASSR

位	7	6	5	4	3	2	1	0	
$22($42)	—	—	—	—	AS2	TCN2UB	OCR2UB	TCR2UB	ASSR
读/写：	R	R	R	R	R/W	R/W	R/W	R/W	

初始化值：$00

周期，中断将不再发生，器件再也无法唤醒。如果用户怀疑自己程序是否满足这一条件，则可以采取如下方法：

① 对 TCNT2，OCR2 及 TCCR2 写入一个合适的值。

② 等待更新忙标志变低。

③ 进入省电模式。

● 若选择了异步工作模式，T/C2 的振荡器将一直工作，除非进入掉电模式。应该注意，此振荡器的稳定时间可能长达 1 s；因此，建议在器件从掉电模式唤醒或上电时至少等待 1 s 后，再使用 T/C2。

● 省电模式唤醒过程：中断条件满足后，在下一个定时器时钟里唤醒过程启动。在 MCU 时钟启动后的 3 个周期，中断标志置位。在此期间，MCU 执行其他指令；但中断条件还不可读，中断例程也不会执行。

● 在异步模式下，中断标志的同步需要 3 个处理器周期加 1 个定时器周期。输出比较引脚的变化与定时器时钟同步，而不与处理器时钟同步。

9.7 定时器/计数器 2 应用举例

1. T/C2 用作实时时钟

T/C2 的时钟源来自 PC6(TOSC1)，PC7(TOSC2)的 32 768 Hz 的晶振。对其 128 分频，计满 256 溢出 1 次，刚好为 1 s。允许 T/C2 溢出中断，在中断服务子程序中，每秒加 1，满 60 s，分加 1，秒清 0；满 60 min，时加 1，分清 0；满 24 h，时清 0。将时、分、秒寄存器中的数转换成十进制数，在 6 位数码管中显示出来，电路如图 9.14 所示，程序如下：

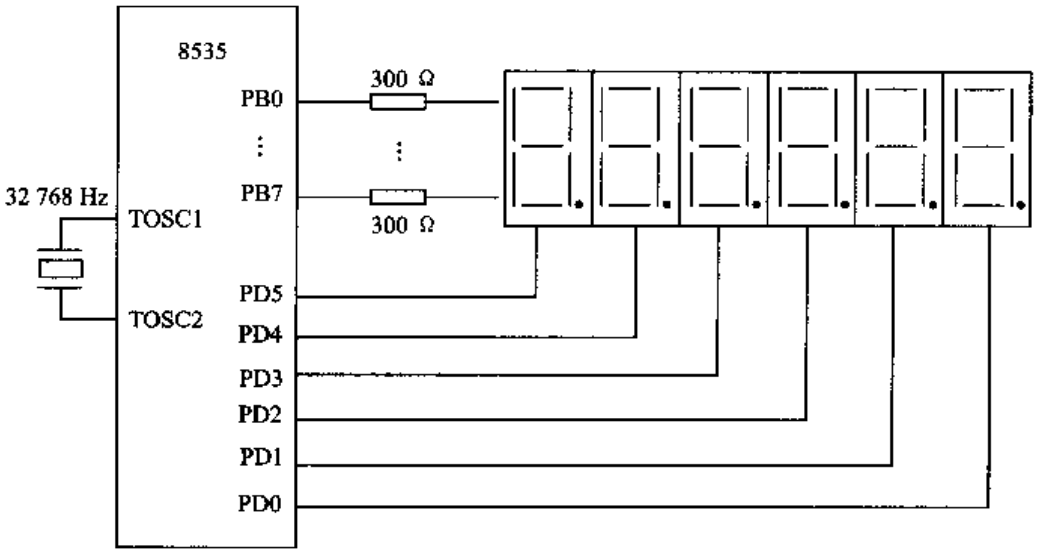


图 9.14 实时时钟及显示电路图

```
.include "8535def.inc"
.org $0000
rjmp reset
.org $004
```

```
    rjmp TIM2_OVF
tab:.db $3f,$06,$5b,$4f,$66,$6d,$7d,$07,$7f,$6f
reset:  ldi r16,low(ramend)          ;栈指针置初值
        out spl,r16
        ldi r16,high(ramend)
        out sph,r16
        ldi r16,$ff                  ;定义 PB,PD 为输出口
        out ddrb,r16
        out ddrd,r16
        ldi r26,0                    ;设时、分、秒初值为 00:00:00
        ldi r27,0
        ldi r28,0
        ldi r16,$08                  ;使用异步时钟
        out assr,r16
        ldi r16,$40                  ;允许 T2 溢出中断
        out timsk,r16
        ldi r16,$05                  ;128 分频,1 s 中断 1 次
        out tccr2,r16
        ldi r16,$00                  ;T/C2 置初值 0
        out tifr,r16
        sei
aa:     mov r16,r26                  ;秒寄存器中数二转十,送 r19,r18
        rcall b8td
        mov r19,r17
        mov r18,r16
        mov r16,r27                  ;分寄存器中数二转十,送 r21,r20
        rcall b8td
        mov r21,r17
        mov r20,r16
        mov r16,r28                  ;时寄存器中数二转十,送 r25,r22
        rcall b8td
        mov r25,r17
        mov r22,r16
        rcall smiao                  ;调动态扫描子程见 7.2.2
        rjmp aa

TIM2_OVF:
        in r1,sreg                   ;保护标志
        inc r26                      ;秒增 1
        cpi r26,60                   ;到 60 s?
        brne tt
        clr r26                      ;到了,则秒清 0
        inc r27                      ;分增 1
```

```
        cpi r27,60              ;到 60 min?
        brne tt
        clr r27                 ;到了,则分清 0
        inc r28                 ;时增 1
        cpi r28,24              ;到 24 h?
        brne tt
        clr r28                 ;到了,则时清 0
tt:     out sreg,r1
        reti

b8td:   clr r17                 ;将 r16 中的二进制数转换为十进制数,十位送 r17,个位送 r16
b8td1:  subi r16,10
        brcs b8td2
        inc r17
        rjmp b8td1
b8td2:  subi r16,(-10)
        ret
```

2. T/C2 的 OC2 引脚产生的 PWM 脉宽调制输出

与 T/C1 的 OC1A 和 OC1B 类似,T/C2 的 OC2 引脚可产生 8 位 PWM 信号,经滤波,既可作模拟电压信号,也可直接产生方波作为脉冲信号。9.5 节中测频率和周期的 2 个例子中的脉冲信号源都是用这种方法产生的。

```
t2pwm1:                         ;产生 15 686 Hz 的方波
        sbi ddrd,7
        ldi r16,$71             ;PWM2 使能,向上计数置引脚
        out tccr2,r16           ;向下计数清引脚,时钟 1 分频
        ldi r16,$80
        out ocr2,r16
        ret

t2pwm2:                         ;产生周期为 8 160 μs 的方波
        sbi ddrd,7
        ldi r16,$75             ;PWM2 使能,向上计数置引脚
        out tccr2,r16           ;向下计数清引脚,时钟 128 分频
        ldi r16,$80
        out ocr2,r16
        ret
```

9.8 看门狗定时器

9.8.1 看门狗定时器的结构、特点及作用

看门狗定时器由片内一个独立的振荡器驱动。在 $V_{cc}=5$ V 的条件下,典型振荡频率为

1 MHz。通过调整看门狗定时器的预定比例器，可以改变看门狗的复位间隔。WDR 是看门狗复位指令，其作用是对看门狗定时器进行清 0。若定时时间到，而没再执行其他的 WDR 指令，单片机将复位，并从复位向量执行程序。参见图 9.15。

看门狗定时器的这个特性，可用于程序的抗干扰。在程序运行通道上，每隔一段加上 WDR 指令，启动看门狗定时器，并经常复位看门狗定时器。程序正常运行时不受影响；但由于干扰，程序飞离正常路线，可能进入死循环，不再执行 WDR 指令，而看门狗定时器的复位，就使程序从头执行了。避免了程序进入死循环后，深陷其中不能自拔。使用看门狗定时器时，要注意选择看门狗定时器复位间隔时间小于各分段程序执行时间的 1/2，以免正常运行时看门狗定时器复位。

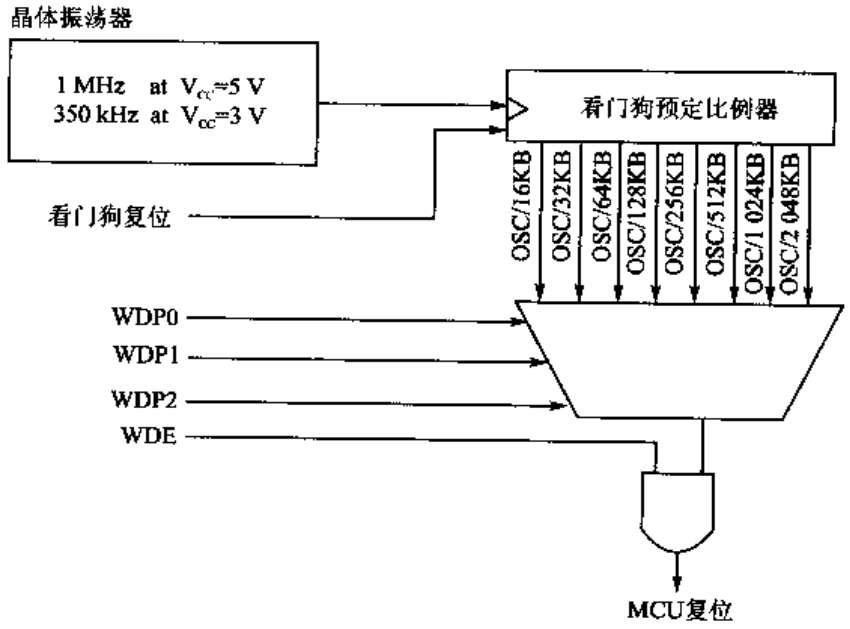


图 9.15　看门狗定时器框图

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。

9.8.2　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	

初始化值：$00

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 WDTOE 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:看门狗触发。

当 WDE 改设为 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 2,1,0——WDP2,WDP1,WDP0:看门狗定时器预定比例器 2,1,0

WDP2,WDP1,WDP0 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 9.12 所列。

表 9.12 看门狗定时器预分频选择

WDP2	WDP1	WDP0	分频系数/K	典型溢出时间/s(V_{CC}=3 V)	典型溢出时间/s(V_{CC}=5 V)
0	0	0	16	0.047	0.015
0	0	1	32	0.094	0.030
0	1	0	64	0.19	0.060
0	1	1	128	0.38	0.12
1	0	0	256	0.75	0.24
1	0	1	512	1.5	0.49
1	1	0	1 024	3.0	0.97
1	1	1	2 048	6.0	1.9

注意:看门狗的振荡频率与电压有关。

WDR 应该在看门狗使能之前执行一次。如果看门狗在复位之前使能,则看门狗定时器有可能不是从 0 开始计数。

9.8.3 看门狗定时器应用编程

看门狗定时器应用编程格式如下:

```
        :                   ;初始化
        :
        wdr                 ;启动看门狗定时器,看门狗定时器溢出间隔 1.9 s
        ldi r16, $0f
        out wdtcr, r16
        :
aaa:    :
        :
        wdr
        :
        :
        wdr
        :
```

```
:
wdr
rjmp aaa
```

调试程序时，先不启动看门狗定时器；程序调试好以后，为了抗干扰，再加上看门狗定时器。

看门狗定时器关断步骤如下：

```
ldi r16, $1f
out wdtcr, r16
ldi r16, $17
out wdtcr,r16
```

第 10 章　8535 单片机模拟量输入接口

10.1　模/数转换器

1. 特　点

- 10 位精度。
- ±2LSB 精确度。
- 0.5LSB 集成非线性度。
- 65～260 μs 转换时间。
- 8 通道。
- 自由运行模式和单次转换模式。
- ADC 转换结束中断。
- 休眠模式噪声消除。

AT90S8535 具有 10 位精度的逐次逼近型 A/D 转换器。ADC 与一个 8 通道的模拟多路转换器相连，这样就允许 A 口作为 ADC 的输入引脚。ADC 包含一个采样保持器。ADC 框图见图 10.1。

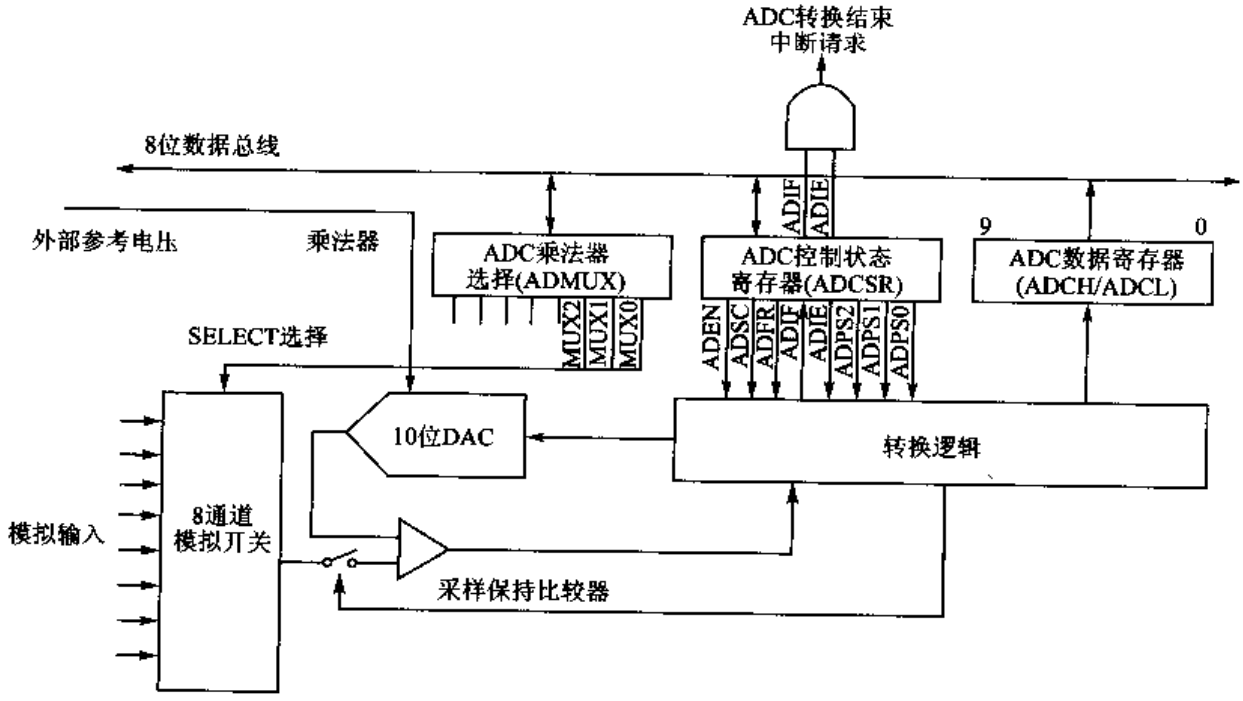


图 10.1　ADC 框图

ADC 具有 2 个模拟供电引脚 AV_{cc}和 AGND。AGND 必须与 GND 相连，且 AV_{cc}与 V_{cc}引脚电压的差别不能大于±0.3 V 。

AREF 为外部参考电压输入端。此电压介于 AGND 与 AV_{CC}之间。

2. 操　作

ADC 可以工作于两种模式，单次转换及自由运行模式。在单次转换模式下，必须启动每一次转换；在自由运行模式下，ADC 会连续采样并更新 ADC 数据寄存器。ADCSR 的 ADFR 位用于选择模式。

ADC 由 ADCSR 的 ADEN 位控制使能。使能 ADC 后，第一次转换将引发一次哑转换过程，以初始化 ADC，然后才真正进行 A/D 转换。对用户而言，此次转换过程比其他转换过程要多 12 个 ADC 时钟周期。

ADSC 置位将启动 A/D 转换。在转换过程中，ADSC 一直保持为高；转换结束后，ADSC 硬件清 0。如果在转换过程中通道改变了，ADC 首先要完成当前的转换，然后通道才会改变。

ADC 产生 10 位的结果，ADCH 和 ADCL。为了保证正确读取数据，系统采用了如下保护逻辑：读数据时，首先要读 ADCL。一旦开始读 ADCL，ADC 对数据寄存器的访问就被禁止了。也就是说，如果读取了 ADCL，那么即使在读 ADCH 之前，另一次 ADC 结束了，2 个寄存器的值也不会被新的 ADC 结果更新，此次转换的数据将丢失。当读完 ADCH 之后，ADC 才能继续对 ADCH 和 ADCL 进行访问。

ADC 结束后会置位 ADIF。即使发生如上所述的，由于 ADCH 未被读取而丢失转换数据的情况，ADC 结束中断仍将触发。

3. 预分频器

ADC 预分频器如图 10.2 所示。

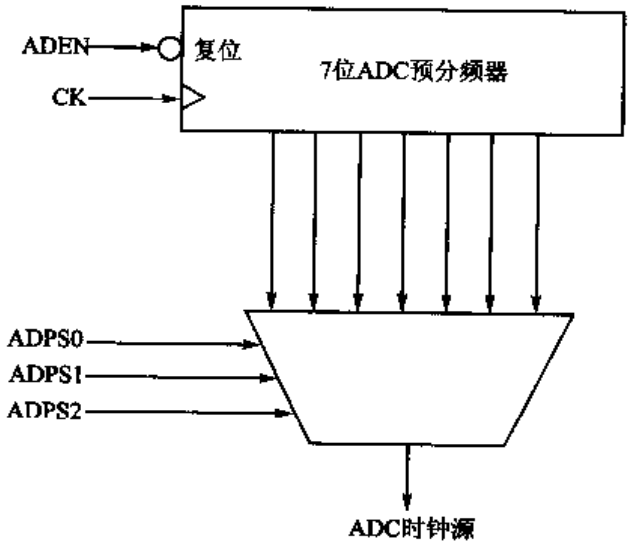


图 10.2　ADC 预分频器

ADC 有一个预分频器，可以将系统时钟调整到可接受的 ADC 时钟(50～200 kHz)。过高的频率将导致采样精度降低。

ADCSR 的 ADPS0 ～ADPS2 用于产生合适的 ADC 时钟。一旦 ADCSR 的 ADEN 置位，预分频器就开始连续不断地计数，直到 ADEN 清 0。ADSC 的作用是对 ADC 进行初始化。A/D 转换在 ADC 时钟的上升沿启动。采样/保持要花费 1.5 倍 ADC 时钟。在第 13 个时钟 ADC 转换结束，数据进入 ADC 数据寄存器。对于单次转换模式，在进行下一次转换时，需要

一个额外的 ADC 时钟(如图 10.3 所示),如果此时 ADSC 为"1",转换可继续进行;在自由运行模式下,ADC 结果写入寄存器后,立即进行下一次转换。工作于 200 kHz 的自由运行模式具有最快的转换速度:65 μs,亦即 15.4 ksps(samplings per second)。转换时序见表 10.1。

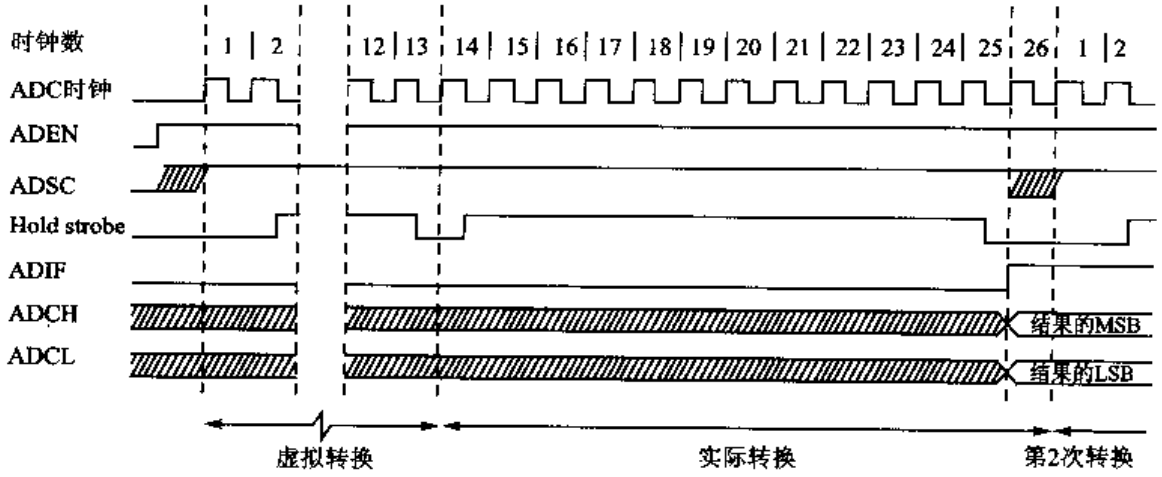


图 10.3 首次转换的时序(单次转换模式)

表 10.1 ADC 时序

条 件	采样周期	得到结果的周期	总的转换周期数	总的转换时间/μs
第 1 次转换,自由运行	14	25	25	125～500
第 1 次转换,单次转换	14	25	26	130～520
自由运行模式	2	13	13	65～260
单次转换模式	2	13	14	70～280

单次转换的时序和自由运行的时序见图 10.4 和图 10.5。

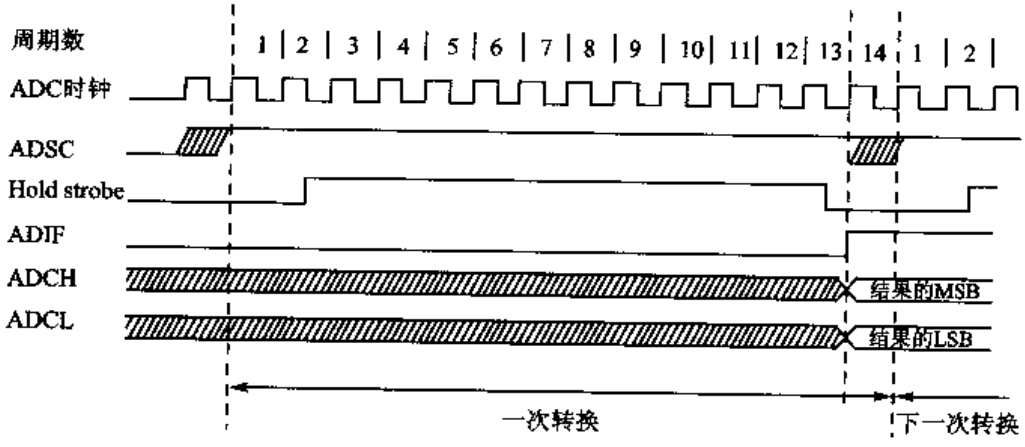


图 10.4 单次转换的时序

4. ADC 噪声抑制功能

ADC 具有消除由 CPU 核引入的噪声的功能。实现过程如下:

① 使能 ADC,选择单次转换模式,并使能转换结束中断。

ADEN=1;ADSC=0;ADFR=0;ADIE=1。

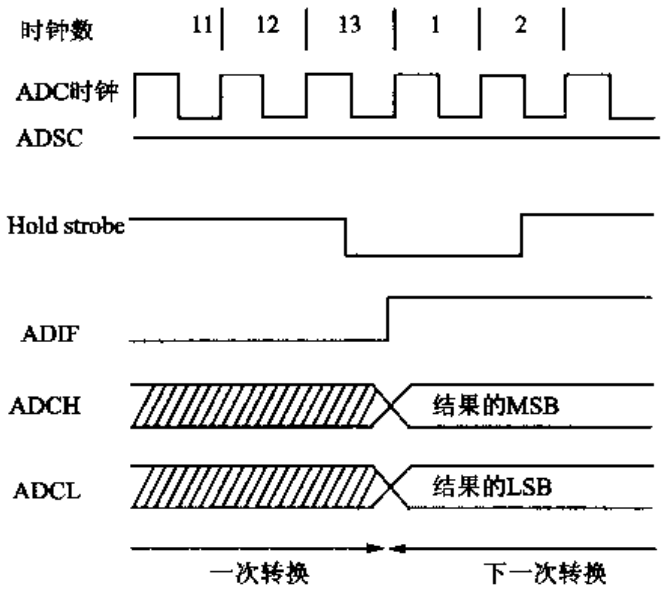


图 10.5　自由运行的时序

② 进入空闲状态。一旦 CPU 停止，ADC 将开始转换。

③ 如果在 ADC 转换结束中断之前没有发生其他中断，则 ADC 转换结束中断将唤醒 MCU，并执行中断例程。

5. 有关的 I/O 寄存器

(1) ADC 多路选择寄存器——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：$00

位 7～3——保留位。

位 2,1,0——MUX2～MUX0：模拟通道选择位。

用于选择 ADC 的模拟输入通道 0～7。

(2) 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：$00

位 7——ADEN：ADC 使能。

位 6——ADSC：ADC 开始转换。

当 ADC 工作于单次转换模式时，这一位必须设置为“1”，以启动每一次转换；而对于自由运行模式，则只需在第一次转换时设置一次。不论设置动作是在 ADEN 置位之后，还是同时进行，ADC 都将进行一次哑转换，以初始化 ADC。

转换过程中 ADSC 保持为高。实际转换过程结束后，但在转换结果进入 ADC 数据寄存器之前(差 1 个 ADC 时钟)，ADSC 变为低。这样就允许在当前转换完成之前(ADSC 变低之时)，对下一次转换进行初始化。一旦当前转换彻底完成，立即就可以进行新的一次转换过程。在哑转换过程当中，ADSC 保持为高。对 ADSC 写 0 没有意义。

位 5-- ADFR：ADC 自由运行模式选择。

这一位置位后，ADC 工作于自由运行模式。ADC 将连续不断地进行采样和数据更新。

位 4——ADIF：ADC 中断标志。

ADC 完成及数据更新完成后，ADIF 置位。如果 I 和 ADIE 置位，则 ADC 结束中断发生。在中断例程里，ADIF 硬件清 0。写"1"也可以对其清 0；因此要注意对 ADCSR 执行读—修改--写操作时，会使即将到来的中断无效。

位 3——ADIE：ADC 中断使能。

位 2，1，0——ADPS2～ADPS0：ADC 预分频器选择，详见表 10.2。

表 10.2 ADC 预分频器选择

ADPS2	ADPS1	ADPS0	分频因子
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

(3) ADC 数据寄存器——ADCL 和 ADCH

	位 7	6	5	4	3	2	1	0	
$05($25)	—	—	—	—	—	—	ADC9	ADC8	ADCH
$04($24)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
读/写：	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	

初始化值：$00

$00

6. 扫描多个通道

由于模拟通道的转换总是要延迟到转换结束；因此，自由运行模式可以用来扫描多个通道而不中断转换器。一般情况下，ADC 转换结束中断用于修改通道，但需注意，中断在转换结果可读时触发。在自由运行模式下，下一次转换在中断触发的同时启动。ADC 中断触发，即新一次转换开始后，改变 ADMUX 将不起作用。

7. ADC 噪声消除技术

AT90S8535 的内外部数字电路会产生 EMI 电磁干扰，从而影响模拟测量精度。如果转换精度要求很高，则需要应用如下技术，以减少噪声。

① AT90S8535 的模拟部分及其他模拟器件在 PCB 上要有独立的地线层。模拟地线与数字地线单点相连。

② 使模拟信号通路尽量短。要使模拟走线在模拟地上通过，并尽量远离高速数字通路。

③ AV_{CC} 要通过一个 RC 网络连接到 V_{CC}。

④ 利用 ADC 的噪声消除技术减少 CPU 引入的噪声。

⑤ 如果 A 口的一些引脚用作数字输出口，则在 ADC 转换过程中不要改变其状态。

注意，由于 AV_{CC} 同时也为 A 口输出驱动提供电源；因此，如果 A 口有输出引脚，则 RC 网络不要使用。

8. ADC 特性

ADC 的特性如表 10.3 所列。

表 10.3　ADC 特性

符　号	参　数	条　件	Min	典型值	Max
	精度/bit			10	
	绝对精度/LSB	V_{REF}=4 V，ADC 时钟=200 kHz		1	2
	绝对精度/LSB	V_{REF}=4 V，ADC 时钟=1 MHz		4	
	绝对精度/LSB	V_{REF}=4 V，ADC 时钟=2 MHz		16	
	整体非线性/LSB	V_{REF}>2 V		0.5	
	差分非线性/LSB	V_{REF}>2 V		0.5	
	零误差偏移/LSB			1	
	转换时间/μs		65		260
	时钟频率/kHz		50		200
AV_{CC}/V	模拟电源		V_{CC}−0.3①		V_{CC}+0.3②
V_{REF}/V	参考电源		AGND		AV_{CC}
R_{REF}/kΩ	参考输入电阻		6	10	13
R_{AIN}/MΩ	模拟输入电阻			100	

注：t_{amb}=−40～85 ℃

① AV_{CC} 的最小值为 2.7 V；

② AV_{CC} 的最大值为 6.0 V；

由表 10.3 可见，ADC 时钟频率对其精度影响很大。

10.2　模/数转换应用举例

测量 8535 的 ACH6 和 ACH7 2 路模拟电压信号的电路如图 10.6 所示。

2 路输入信号经 RC 滤波去除交流分量。6.2 V 稳压管起保护作用，高于 6.2 V 的输入信号被限幅在 6.2 V 之内。对负的输入信号，稳压管反向导通，限幅在−0.7 V 之内，以避免损

坏输入引脚。

基准电压的稳定、准确事关 ADC 转换结果的精确性。这里采用 LM336(5.0 V)的 3 端精密并联式二极管。通过调节精密多圈电位器,可调节 V_{REF} 的电压值。AVcc 的供电经 100 Ω 电阻和 0.1 μF 电容滤波,以减少交流分量对 ADC 的影响。PB 口 8 位线作动态扫描数码管字线,PD 口低 5 位作动态扫描数码管位线,数码管用共阴极。5 位数码管最左边显示测量的路号,右边 4 位显示 A/D 转换的数字量。每隔 1 s 轮换显示 1 次。当 ADC 采用自由模式时,每秒换路 1 次(读完 ADC 结果改变多路开关),ADC 不断进行,读的总是最新结果。

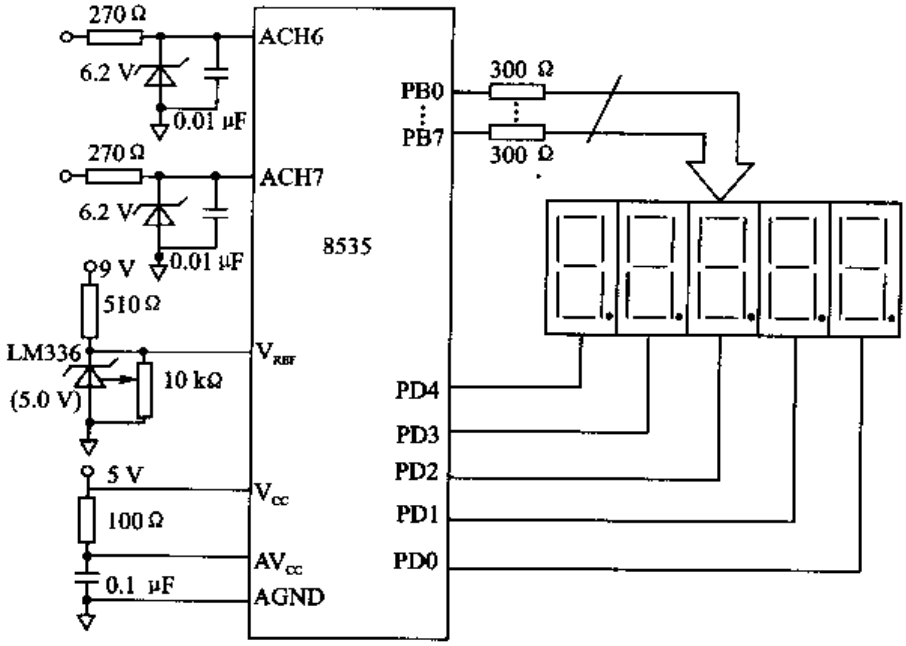


图 10.6 测量并显示 2 路模拟电压信号电路图

采用自由模式程序如下:

```
.include"8535def.inc"
.org $0000
rjmp reset
tab:.db $3f,$06,$5b,$4f,$66,$6d,$7d,$07,$7f,$6f   ;7 段码表
reset:  ldi r16,low(ramend)                ;栈指针置初值
        out spl,r16
        ldi r16,high(ramend)
        out sph,r16
        ldi r16,$ff                        ;定义 PB,PD 为输出口
        out ddrb,r16
        out ddrd,r16
        ldi r16,$00                        ;定义 PA 口为输入口,不带内部上拉电阻
        out ddra,r16
        ldi r16,$00
        out porta,r16
        ldi r16,$07                        ;先第 7 路 ADC
```

```
        out admux,r16
        ldi r18,$e6                ;允许 ADC,启动 ADC,自由模式
        out adcsr,r18              ;64 分频作 A/D 时钟
        rcall t1ms                 ;延时 1 ms
aa:     in r16,adcl                ;读 A/D 结果放入 r17:r16 中
        in r17,adch
        ldi r18,$06                ;改变 ADMUX 为第 6 路
        out admux,r18
        rcall b16td5               ;调用二转十子程见 4.3.1
        ldi r22,7                  ;万位显示路号 7
        mov r21,r19                ;4 位 ADC 结果送显示缓冲区
        mov r20,r18
        mov r19,r17
        mov r18,r16
        ldi r17,200                ;每一路 A/D 扫描 200 次,恰好 1 s
bb:     rcall smiao                ;调动态扫描子程序见 7.2.2
        dec r17
        brne bb
        in r16,adcl                ;读 A/D 结果放入 r17:r16 中
        in r17,adch
        ldi r18,$07                ;改变 ADMUX 为第 7 路
        out admux,r18
        rcall b16td5               ;调用二转十子程
        ldi r22,6                  ;万位显示路号 6
        mov r21,r19                ;4 位 ADC 结果送显示缓冲区
        mov r20,r18
        mov r19,r17
        mov r18,r16
        ldi r17,200                ;每一路 A/D 扫描 200 次,恰好 1 s
cc:     rcall smiao                ;调动态扫描子程序见 7.2.2
        dec r17
        brne cc
        rjmp aa
```

10.3　模拟比较器

10.3.1　模拟比较器概述

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 被置位。比较器的输出可用来触发模拟比较器中断(上升沿、下降沿或电平变换),也可触发定时器/计数器 1 的输入捕获功能。其方框图和周围电路如图 10.7 所示。

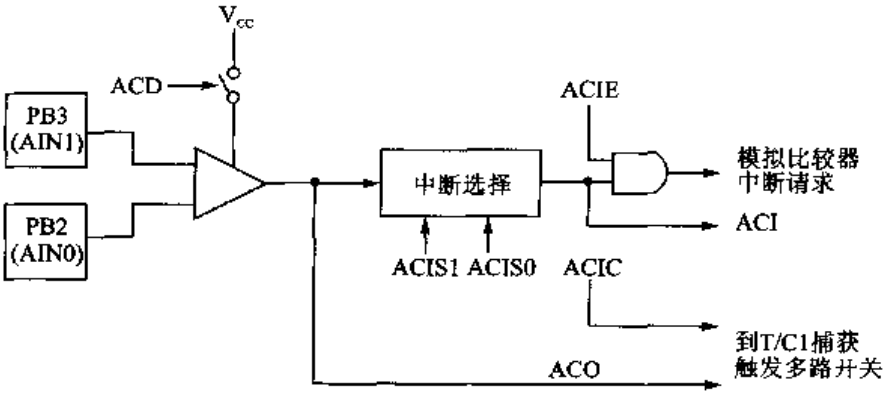


图 10.7 模拟比较器方框图

10.3.2 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：$00

位 7——ACD：模拟比较器禁止位。

当该位设为 1 时，模拟比较器的电源关闭。可以在任何时候对其置位，以便关闭模拟比较器，这样可以减少器件功耗。常用于休闲模式下又不需从模拟比较器中断唤醒的情况。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。

当比较器输出触发中断时，ACI 将置位。中断方式由 ACIS1 和 ACIS0 决定。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 0。另外，ACI 也可以通过对此位写入逻辑 1 来清 0。**注意**，如果 ACSR 的另一些位被 SBI 或 CBI 指令修改时，ACI 亦被清 0。

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，模拟比较器中断被触发。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，该位触发定时器/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 2 位决定了哪一比较器事件触发模拟比较器中断。表 10.4 所列为不同的设置。

表 10.4　ACIS1/ACIS0 设置

ACIS1	ACIS0	中断模式	ACIS1	ACIS0	中断模式
0	0	电平变换引发中断	1	0	ACO 下降沿中断
0	1	保　留	1	1	ACO 上升沿中断

注意:改变 ACIS1/ACIS0 时，要禁止模拟比较器的中断，否则有可能引发不必要的中断。

10.4　模拟比较器应用举例

电路如图 10.8 所示，AIN1 引脚加 0.5V_{CC} 的固定电压，AIN0 引脚输入变化电压如图 10.9所示。

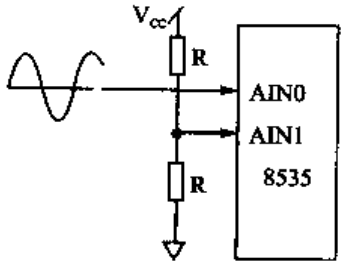


图 10.8　模拟比较器应用电路图

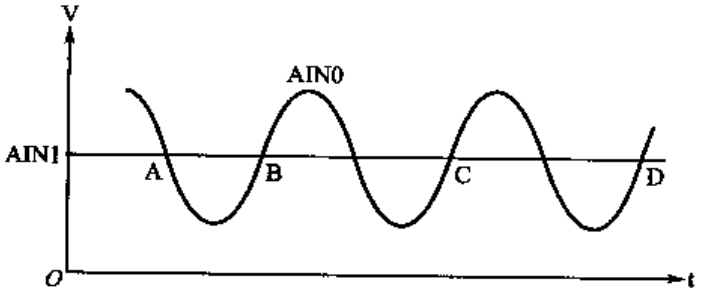


图 10.9　AIN0 电压随时间变化曲线图

当 AIN0 的输入电压向下穿越 AIN1 上的电压(到达 A 点)时，I/O 寄存器的 ACSR 中的 ACO 位由 1 变 0。程序可检测到 ACO 的变化。

当 AIN0 的输入电压向上穿越 AIN1 上的电压(到达 B 点)时，I/O 寄存器的 ACSR 中的 ACO 位由 0 变 1。程序检测到 ACO 变 1 后，使能模拟比较器上升沿中断。

当 AIN0 的输入电压再次向上穿越 AIN1 上的电压(到达 C 点)时，将产生新的模拟比较器中断标志 ACI。程序清 ACI 标志，开模拟比较器中断和全局中断。

此后从 D 点开始，AIN0 每次向上穿越 AIN1 时，均产生模拟比较器中断，并使 16 位计数寄存器 r18:r17 增 1。

编程如下。

```
.include"8535def.inc"
.org $000
rjmp RESET
.org $010
rjmp ANA_COMP
RESET:   ldi r16,low(ramend)          ;栈指针置初值
```

```
        out spl,r16
        ldi r16,high(ramend)
        out sph,r16
wait_edge1:
        sbic acsr,aco                    ;等待输入信号(AIN0)<(AIN1)
        rjmp wait_edge1
wel_1:  sbis acsr,aco                    ;等待输入信号(AIN0)>(AIN1)
        rjmp wel_1
;说明:AIN0 穿越 AIN1 时,模拟比较器输出(ACO)就改变一次,程序可检测此标志
wait_edge2:
        sbi acsr,acis0                   ;选择模拟比较器中断模式为上升沿
        sbi acsr,acis1
        sbi ACSR,ACI                     ;清模拟比较器中断标志 ACI
we2_1:  sbis ACSR,ACI                    ;等待模拟比较器中断标志变 1
        rjmp we2_1
;说明:模拟比较器中断标志(ACI)按 ACIS1 和 ACIS0 所规定的模式置位,程序可检测此标志
ana_init:  clr r17                       ;清计数寄存器的高、低字节
        clr r18
        ldi r16,$13                      ;清模拟比较器中断标志 ACI,选上升沿触发
        out ACSR,r16
        sei                              ;开中断
        sbi ACSR,ACIE                    ;使能模拟比较器中断
here:   rjmp here

ANA_COMP:in r1,sreg                      ;保护标志寄存器
        subi r17,low(-1)                 ;计数器加 1
        sbci r18,high(-1)
        out sreg,r1                      ;恢复标志寄存器
        reti
;说明:在模拟比较器中断使能位(ACIE)置 1 和全局中断使能后,就进入模拟比较器中断
```

第 11 章　AVR 单片机串行接口及应用

11.1　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多种波特率;

② 在 XTAL 低频率下,仍可产生较高的波特率;

③ 8 位和 9 位数据;

④ 噪声滤波;

⑤ 过速的检测;

⑥ 帧错误检测;

⑦ 错误起始位的检测;

⑧ 3 个独立的中断,即发送(TX)完成,发送数据寄存器空,接收(RX)完成。

11.1.1　数据传送

数据传送通过把被传送的数据写入 UART 的 I/O 寄存器 UDR 初始化。

在以下情况下,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。

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(11)位移位寄存器中时,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。如果在 UDR 里有新数据,则 UART 会在停止位发送完毕后,自动加载数据。在加载数据的同时,UDRE 被置位,并一直保持到有新数据写入 UDR。当没有新的数据被写入,而且停止位在 TXD 上保持了一位的长度时,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,使能 UART 发送器;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 输出将被连到 PD1 引脚作输出,而不管方向寄存器的设置。

UART 传送器的方框示意图见图 11.1。

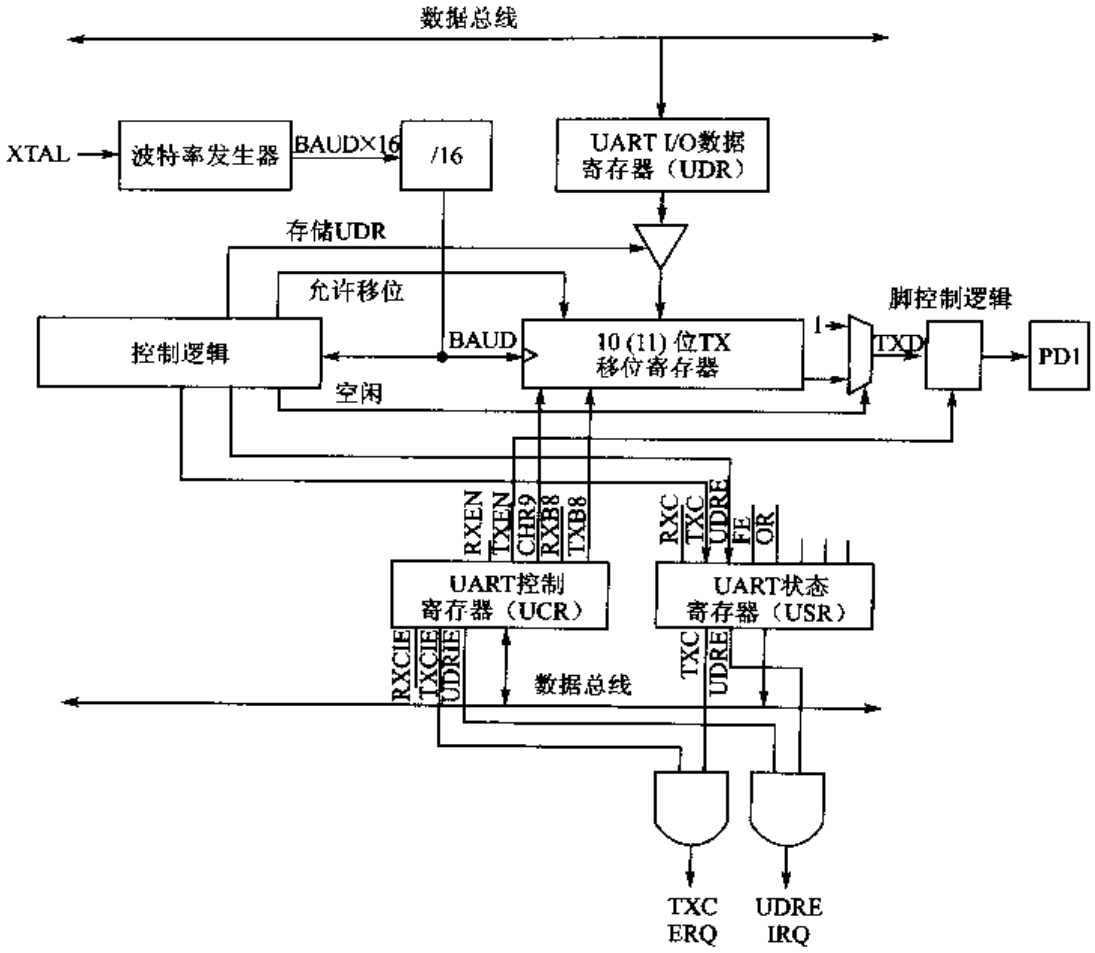


图 11.1 UART 传送器方框示意图

11.1.2 数据接收

图 11.2 为 UART 接收器的示意图。

接收器前端的逻辑以 16 倍波特率对 RXD 引脚采样。当线路闲置时，一个逻辑 0 的采样将被认为是起始位的下降沿，并且起始位的探测序列开始。设采样 1 为第 1 个 0 采样，接收器在第 8,9 及 10 个采样点采样 RXD 引脚。如果 3 个采样中 2 个或 2 个以上是逻辑 1，则认为该“起始位”是噪声尖峰引起，是假的，而被丢弃，接收器继续检测下一个 1～0 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 及 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 11.3 所示。

当停止位到来时，3 个采样中的大数应为 1，才能接收该停止位。如果 2 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，应检查 FE 帧错误标志。一旦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 2 个物理上分离的寄存器，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，发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)，当数据被传送到 UDR 时，传送移位寄存器的第 9 位被装入到 UCR 的 RXB8 位。

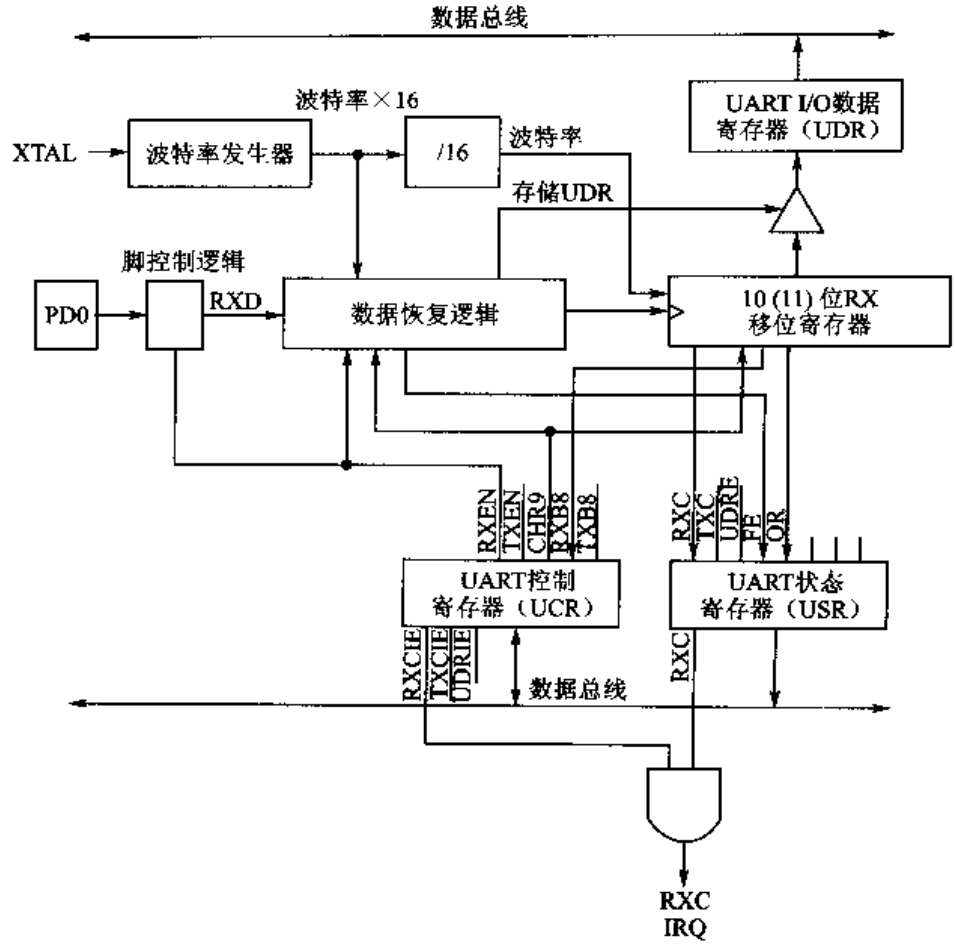


图 11.2　UART 的接收器示意图

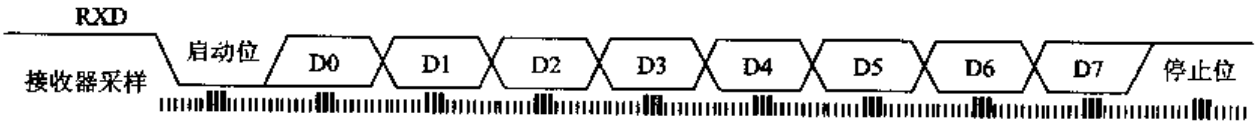


图 11.3　接收器数据采样

如果在读取 UDR 寄存器之前，UART 又收到一个字符，则 UCR 中的过速标志(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 位一直保留到 UDR 被读取；因此，在读 UDR 后，应检查 OR 位。

通过清除 UCR 寄存器中的 RXEN 位，使接收器被禁止。这意味着 PD0 可以被用作通用的 I/O 引脚。当 RXEN 被设置时，UART 接收器被连到 PD0 引脚而不管方向寄存器的设置。

11.1.3　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：$00									

UDR 寄存器是 2 个物理上分离的寄存器，分享同一个 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	UDRE	FE	OR	—	—	—	USR
读/写：	R	R	R	R	R	R	R	R	
初始化值：	0	0	1	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，接收完成中断子程序必须读 UDR，而清除 RXC；否则，在中断完成后，会引起新的中断。

位 6——TXC：UART 发送完成。

当发送移位寄存器的全部数据被移出，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信接口中很有用。

当完成发送后，立即释放通信总线，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；或者 TXC 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 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。

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：过速(超越出错)。

如果 UDR 寄存器中旧的数据还没被读走，新的数据又进入接收移位寄存器，则 OR 位被置 1。

位 2～0—— Res：保留位。

在 AT90S8535 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	R/W	

初始化值：$00

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 及 FE 无法置位。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

波特率发生器依据以下等式的分频器产生波特率：

$$BAUD=f_{CK}/[16(UBRR+1)]$$

其中 BAUD 表示波特率；f_{CK} 表示晶振频率；UBRR 表示 UART 波特率寄存器的值，为 0～255。

对于标准的晶振频率，可以通过表 11.1 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 11.1 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/%	2.4576 MHz	误差/%
2400	UBRR=25	0.2	UBRR=17	0.0	UBRR=51	0.2	UBRR=63	0.0
4800	UBRR=12	0.2	UBRR=23	0.0	UBRR=25	0.2	UBRR=31	0.0
9600	UBRR=6	7.5	UBRR=11	0.0	UBRR=12	0.2	UBRR=15	0.0
14400	UBRR=3	7.8	UBRR=7	0.0	UBRR=8	3.7	UBRR=10	3.1
19200	UBRR=2	7.8	UBRR=5	0.0	UBRR=6	7.5	UBRR=7	0.0
28800	UBRR=1	7.8	UBRR=3	0.0	UBRR=3	7.8	UBRR=4	6.3
38400	UBRR=1	22.9	UBRR=2	0.0	UBRR=2	7.8	UBRR=3	0.0
57600	UBRR=0	7.8	UBRR=1	0.0	UBRR=1	7.8	UBRR=2	12.5
76800	UBRR=0	22.9	UBRR=1	33.3	UBRR=1	22.9	UBRR=1	0.0
115200	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/%
2400	UBRR=84	0.1	UBRR=95	0.0	UBRR=103	0.2	UBRR=119	0.0
4800	UBRR=42	0.8	UBRR=47	0.0	UBRR=51	0.2	UBRR=59	0.0
9600	UBRR=20	1.6	UBRR=23	0.0	UBRR=25	0.2	UBRR=29	0.0
14400	UBRR=13	1.6	UBRR=15	0.0	UBRR=16	2.1	UBRR=19	0.0
19200	UBRR=10	3.1	UBRR=11	0.0	UBRR=12	0.2	UBRR=14	0.0
28800	UBRR=6	1.6	UBRR=7	0.0	UBRR=8	3.7	UBRR=9	0.0
38400	UBRR=4	6.3	UBRR=5	0.0	UBRR=6	7.5	UBRR=7	6.7
57600	UBRR=3	12.5	UBRR=3	0.0	UBRR=3	7.8	UBRR=4	0.0
76800	UBRR=2	12.5	UBRR=2	0.0	UBRR=2	7.8	UBRR=3	6.7
115200	UBRR=1	12.5	UBRR=1	0.0	UBRR=1	7.8	UBRR=2	20.0
波特率	7.327 8 MHz	误差/%	8 MHz	误差/%	9.216 MHz	误差/%	11.059 MHz	误差/%
2400	UBRR=191	0.0	UBRR=207	0.2	UBRR=239	0.0	UBRR=287	—
4800	UBRR=95	0.0	UBRR=103	0.2	UBRR=119	0.0	UBRR=143	0.0
9600	UBRR=47	0.0	UBRR=51	0.2	UBRR=59	0.0	UBRR=71	0.0
14400	UBRR=31	0.0	UBRR=34	0.8	UBRR=39	0.0	UBRR=47	0.0
19200	UBRR=23	0.0	UBRR=25	0.2	UBRR=29	0.0	UBRR=35	0.0
28800	UBRR=15	0.0	UBRR=16	2.1	UBRR=19	0.0	UBRR=23	0.0
38400	UBRR=11	0.0	UBRR=12	0.2	UBRR=14	0.0	UBRR=17	0.0
57600	UBRR=7	0.0	UBRR=8	3.7	UBRR=9	0.0	UBRR=11	0.0
76800	UBRR=5	0.0	UBRR=6	7.5	UBRR=7	6.7	UBRR=8	0.0
115200	UBRR=3	0.0	UBRR=3	7.8	UBRR=4	0.0	UBRR=5	0.0

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：$00

UBRR 是 8 位可以读/写的寄存器，用来确定波特率。

11.2　异步串行接口 UART 应用举例

11.2.1　异步串行口应用

UART 异步串行口可直接用于机内传送数据。在一个智能仪器和控制设备内采用串行接口传送数据，可用 TTL 电平直接传送。在室内 2 个智能仪表或设备之间传送数据时，可以采用 RS－232C 标准。将单片机的串行口经 MAX232 芯片转换成 RS－232C 标准电平，传送距离可达 15 m。例如 PC 机与 8535 单片机间传送数据。

RS－232C 电气特性如下：

逻辑 0——＋5 V，＋15 V；

逻辑 1——－5 V，－15 V；

波特率——20 000 之内；

传送距离——15 m 之内。

MAX232 芯片可完成 TTL 电平与 RS－232C 电平间的转换。其供电为单－5 V 电源，片内有升压电路。这样不必增加电源品种，简单且可靠。通过 RS－232C 使用 MODEM 还可通过电话线远传。

工厂内的智能仪表或控制设备与上位机的通信，常采用 RS485 或 RS422 标准。其中 RS422 是全双工 4 线传输，RS485 是半双工 2 线传输。其逻辑电平均为：逻辑 0，－0.2～－6 V；逻辑 1，＋0.2～＋6 V。波特率可达 10 Mbps，传送距离 1.2 km，降低波特率还可增加传送距离。

单片机的 UART 接口经 MAX488，MAX490 等芯片驱动，可实现 RS422 接口；经 MAX481，MAX483，MAX485 及 MAX487 等芯片驱动，可实现 RS485 接口。串行接口的很多标准都可由单片机的 UART 接口经驱动器实现。由于篇幅所限，这里就不再赘述了。

11.2.2　串行口编程注意的问题

① 定义 TXD，RXD 引脚(对 UCR 中的 RXEN 和 TXEN 位置 1)。

② 定义波特率，给 UBRR 送一个适当值。注意应使互相通信的设备和仪表的波特率一致。

③ 启动串行发送是用指令 OUT UDR，Rn 实现的。

④ 由于串行发送过程较慢，发一个字节需要较长时间，连续发送数据时，可采用查询(查 USR 的 TXC 位或 UDRE 位为 1)、延时(延时时间略大于发送总位数/波特率)的方法；也可采用中断(UART 数据寄存器空中断或 UART 发送结束中断)的方法，在中断服务子程序中再发送下一个字节。

⑤ 使用 UART 中断发送数据时，注意中断初始化。把 UCR 中的 TXC 或 UDR 位置 1 使能中断和用 SEI 指令使能全局中断。

⑥ 用指令 IN Rn，UDR 将接收到的串行数据读到寄存器中。

⑦ 何时读串行数据，可采用查询方法，查询 USR 中的 RX 位为 1；但用此法 MCU 就不能干别的事了，所以很少使用。中断法采用 UART 接收中断，在中断服务子程序中，读 UDR 中

串行接收数据。

⑧ UART 接收中断应注意中断初始化，即把 UCR 中的 RXCIE 位(UART 接收中断允许位)置 1 和用 SEI 指令使能全局中断。

11.2.3 UART 串行通信举例

2 个 8535 单片机通信电路如图 11.4 所示。

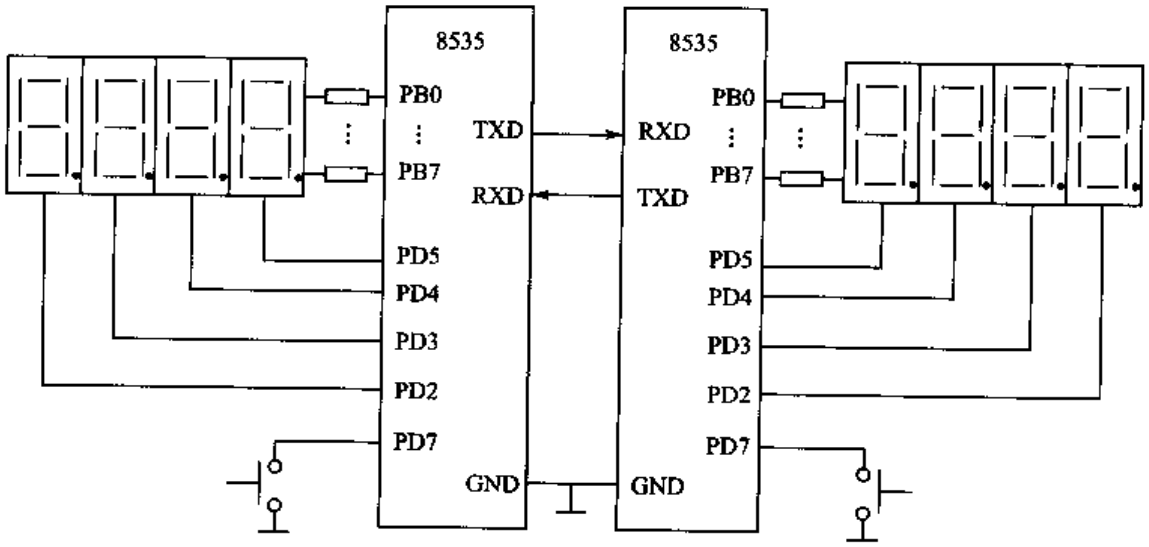


图 11.4 2 个单片机 UART 异步通信及显示电路图

2 个单片机各自进行动态扫描显示 0,1,2,3 四个数字。当甲机键按下后，甲机向乙机通过串行口发送 5,6,7,8 四个数字，乙机收到后，将收到的数字 5,6,7,8 显示出来。若乙机按一下键后，也同样向甲机发送 5,6,7,8 四个数字，甲机收到后，显示此 4 个数字。

1. 发送采用查询 USR 中的 UDRE 位，接收采用中断方式

程序如下：

```
.include "8535def.inc"
.org $0000
rjmp reset
.org $00b
rjmp UART_RXC
tab:.db $3f,$06,$5b,$4f,$66,$6d,$7d,$07,$7f,$6f    ;7 段码表
reset:  ldi r16,low(ramend)              ;栈指针置初值
        out spl,r16
        ldi r16,high(ramend)
        out sph,r16
        ldi r16,$ff                      ;定义 PB 口为输出口
        out ddrb,r16
        ldi r16,$7f                      ;定义 PD7 为输入,PD0～PD6 为输出
        out ddrd,r16
        sbi portd,7                      ;定义 PD7 带上拉输入
```

```
        ldi r16, $98                ;定义串收、串发且允许中断
        out ucr,r16
        sei
        ldi r16,51                  ;定义波特率为 9 600
        out ubrr,r16
        ldi r18,0                   ;显示缓冲区先送 0,1,2,3
        ldi r19,1
        ldi r20,2
        ldi r21,3
        ldi Xl, $10                 ;发送缓冲区指针 X 置初值
        ldi Xh, $01
        ldi Yl, $12                 ;接收缓冲区指针 Y 置初值
        ldi Yh, $00
        ldi r16,5                   ;SRAM $110～$113 送 5,6,7,8
        sts $110,r16
        ldi r16,6
        sts $111,r16
        ldi r16,7
        sts $112,r16
        ldi r16,8
        sts $113,r16
aa:     rcall smiao4                ;动态扫描
        in r16,pind                 ;读 PD7,有键按下就转异步发送
        sbrs r16,7
        rjmp bb
        rjmp aa                     ;否则继续动态扫描
bb:     ldi r17,4                   ;共发 4 个字节
cc:     ld r16,X+                   ;发 1 个字节
        out udr,r16
dd:     sbis usr,udre
        rjmp dd
        dec r17                     ;没发完 4 个,继续发
        brne cc
ee:     rcall smiao4                ;动态扫描
        rjmp ee
smiao4: ldi r16, $fb                ;选中 PD2,先显示个位
        out portd, r16
        mov r23,r18                 ;将待显示的数放在 r23 中
        rcall cqb                   ;查 7 段码送字线,见 7.2.2
        rcall t1ms                  ;延时 1 ms
        ldi r16, $f7                ;选中 PD3,先显示十位
        out portd, r16
        mov r23,r19                 ;将待显示的数放在 r23 中
```

```
        rcall cqb                    ;查7段码送字线
        rcall t1ms                   ;延时1 ms
        ldi r16,$ef                  ;选中PD4,先显示百位
        out portd,r16
        mov r23,r20                  ;将待显示的数放在r23中
        rcall cqb                    ;查7段码送字线
        rcall t1ms                   ;延时1 ms
        ldi r16,$df                  ;选中PD5,先显示千位
        out portd,r16
        mov r23,r21                  ;将待显示的数放在r23中
        rcall cqb                    ;查7段码送字线
        rcall t1ms                   ;延时1 ms
        ret
UART_RXC:in r1,sreg                  ;保护标志寄存器
        in r22,udr                   ;读UART数据寄存器
        st Y+,r22                    ;送r18~r21中1个寄存器
        out sreg,r1                  ;恢复标志寄存器
        reti
```

除接收采用中断方式以外,发送也采用中断方式。可采用UART数据寄存器空中断或UART发送完成中断两者之一。2种方案编程差不多,下面只给出了采用UART发送完成中断加接收完成中断的程序。

```
.include "8535def.inc"
.org $0000
rjmp reset
.org $00b
rjmp UART_RXC
.org $00d
rjmp UART_TXC
tab:.db $3f,$06,$5b,$4f,$66,$6d,$7d,$07,$7f,$6f    ;7段码表
reset: ldi r16,low(ramend)           ;栈指针置初值
       out spl,r16
       ldi r16,high(ramend)
       out sph,r16
       ldi r16,$ff                   ;定义PB口为输出口
       out ddrb,r16
       ldi r16,$7f                   ;定义PD7为带上拉的输入,PD0~PD6为输出
       out ddrd,r16
       sbi portd,7
       ldi r16,$d8                   ;允许串发、串收及相应中断
       out ucr,r16
       ldi r16,51                    ;波特率为9 600
       out ubrr,r16
```

```
        sei                         ;开中断
        ldi r18,0                   ;4 位显示送初值 BCD 码
        ldi r19,1
        ldi r20,2
        ldi r21,3
        ldi Xl,$10                  ;X 发送缓冲区指针置初值
        ldi Xh,$01
        ldi Yl,$12                  ;Y 接收缓冲区(显示)指针置初值
        ldi Yh,$00
        ldi r16,5                   ;发送缓冲区(4 个 BCD 码)送初值
        sts $110,r16
        ldi r16,6
        sts $111,r16
        ldi r16,7
        sts $112,r16
        ldi r16,8
        sts $113,r16
aa:     rcall smiao4
        in r16,pind                 ;读 PD7,有键按下就转异步发送
        sbrs r16,7
        rjmp bb
        rjmp aa                     ;否则继续动态扫描
bb:     ldi r17,$03                 ;要发送 4 个字节
        ld r16,X+                   ;主程序先发送第 1 个字节
        out udr,r16
cc:     rcall smiao4                ;动态扫描显示子程,见本节前一程序
        rjmp cc
UART_RXC:                           ;串收中断子程
        in r1,sreg                  ;保护标志寄存器
        in r22,udr                  ;读串收数据寄存器
        st Y+,r22                   ;送接收缓冲区
        out sreg,r1                 ;恢复标志寄存器
        reti
UART_TXC:                           ;串收中断子程
        in r1,sreg                  ;保护标志寄存器
        ld r24,X+                   ;串发 1 个字节
        out udr,r24
        dec r17
        brne ee                     ;没发完,中断返回,下次中断再发
        cbi ucr,txcie               ;发完规定的字节数,清发送中断使能位
ee:     out sreg,r1                 ;恢复标志寄存器
        reti
```

为了简化实验，也可以用一个单片机自发自收的方法，将 TXD，RXD 两引脚连起来即可。发送也可采用查询 TXC 位或延时 1 ms 的方法实现，程序改动不大，就不赘述了。

11.3 同步串行接口 SPI

同步串行接口(SPI)允许在 90 系列单片机与外设或几个 90 系列单片机之间高速同步数据传送，如图 11.5 所示。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 可选 LSB 在先或 MSB 在先。

④ 4 种可编程的位速率。

⑤ 传送结束中断标志。

⑥ 写冲突标志保护。

⑦ 可从闲置模式下唤醒(仅从模式)。

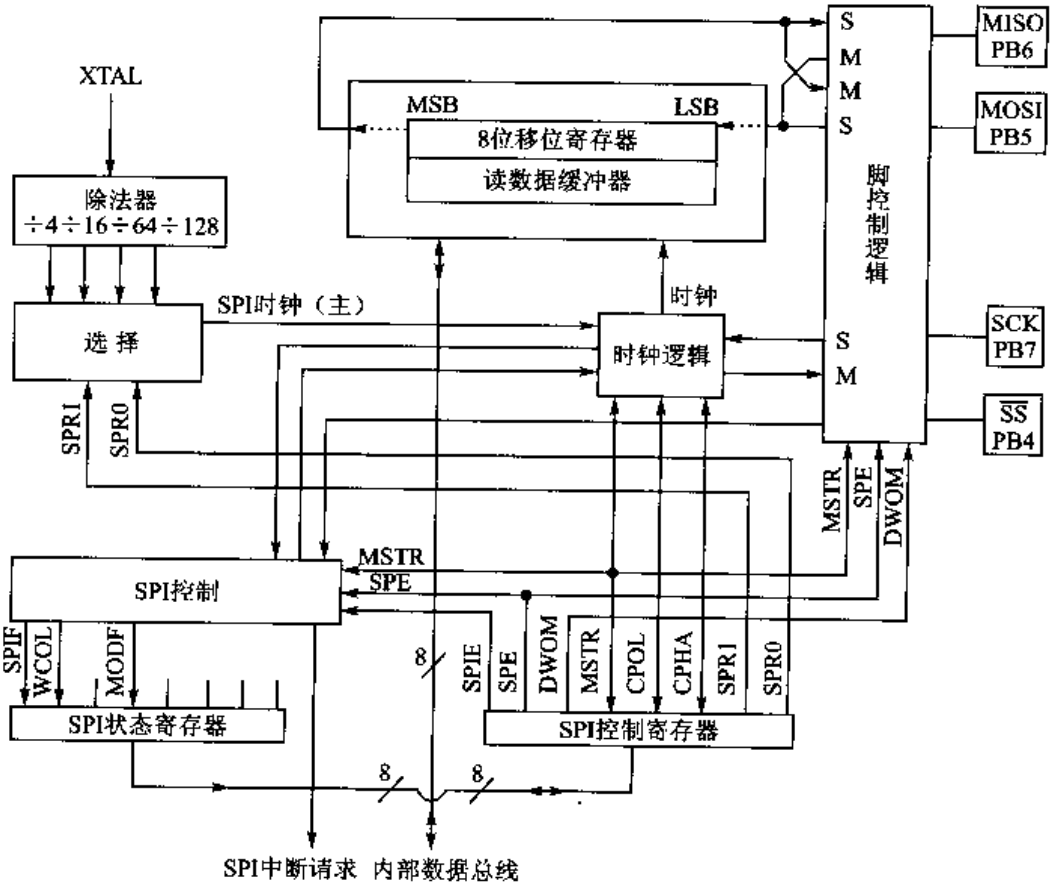


图 11.5 SPI 方框图

主从 CPU 之间的 SPI 连接如图 11.6 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 时钟发生器停止，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

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 2 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 11.6 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 1 个移位周期内，主机和从机的数据交换。

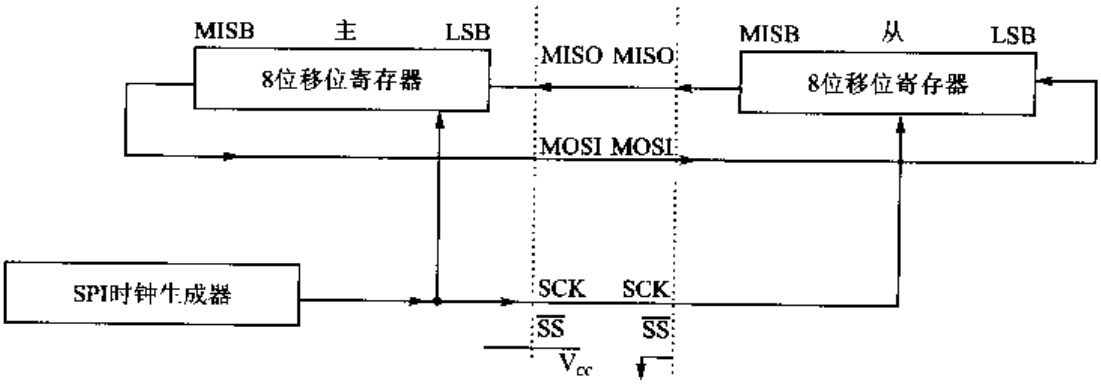


图 11.6　SPI 主从 CPU 内部连接

这个系统在发送方向上有 1 级缓冲，而在接收方向有 2 级缓冲。这意味着，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。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 及$\overline{SS}$引脚的数据方向，按照表 11.2 配置。

表 11.2　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	SCK	用户定义	输　入
MISO	输　入	用户定义	$\overline{SS}$	用户定义	输　入

1. $\overline{SS}$引脚的功能

当 SPI 被配置为主机时(SPCR 的 MSTR 置 1)，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；如果需被设为输入，则必须保持为高，以保证主机 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。为了防止总线冲突，SPI 系统将遵循以下规则：

① SPCR 的 MSTR 位被清除，则 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。

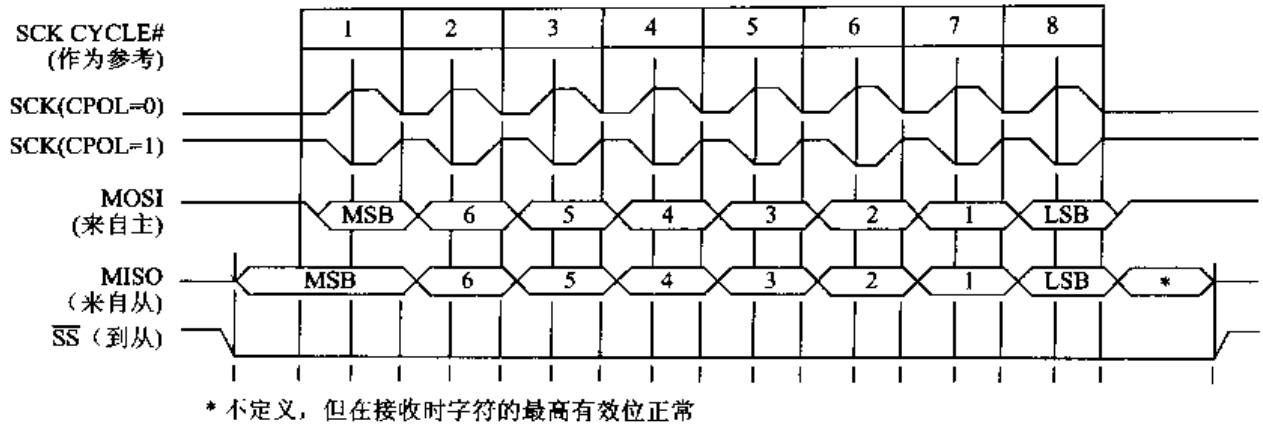
因此在主机模式下使用中断驱动的 SPI 发送时，存在$\overline{SS}$被拉低的可能。中断应检查 MSTR 位是否被设置，一旦发现 MSTR 位被清 0，则必须被用户再置位，以便进入主机模式。

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 功能激活，MISO 变为输出引脚，而其他引脚成为输入。如果$\overline{SS}$为高，则所有相关引脚都为输入，SPI 不接收任何数据。要注意的是，若$\overline{SS}$拉高，SPI 逻辑将复位。如果在发送过程中$\overline{SS}$被拉高，则数据传输马上中断，数据丢失。

2. 数据模式

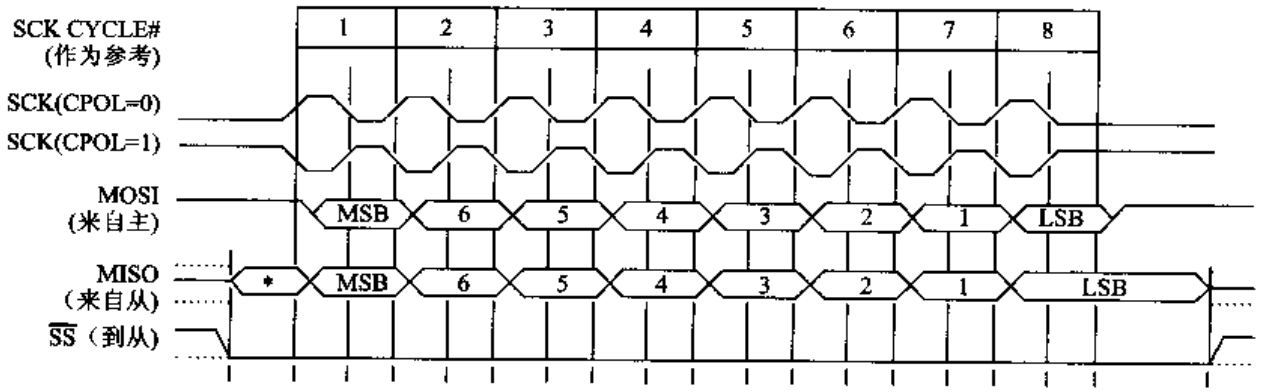
SCK 相位、极性与串行数据有 4 种组合，由控制位 CPHA 和 CPOL 来决定。SPI 的传输

格式如图 11.7 和 11.8 所示。



* 不定义，但在接收时字符的最高有效位正常

图 11.7 当 CPHA=0 时，SPI 传送格式



* 不定义，但在接收时字符的最高有效位正常

图 11.8 当 CPHA=1 时，SPI 传送格式

3. 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：$00

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，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；当该位被清 0 时，SCK 在闲置时是低电平。详见图 11.7 和图 11.8。

位 2——CPHA：时钟相位。

该位的功能详见图 11.7 和图 11.8。

位 1，0——SPR1，SPR0：SPI 时钟速率选择 1 和 0。

这 2 位控制主机模式下器件 SCK 的速率。SPR1 和 SPR0 对于从机无影响。SCK 和振荡器频率 f_{CL} 之间的关系如表 11.3 所列。

表 11.3　SCK 和振荡器频率之间的关系

SPR1	SPR0	SCK 速率	SPR1	SPR0	SCK 速率
0	0	$f_{CL}/4$	1	0	$f_{CL}/64$
0	1	$f_{CL}/16$	1	1	$f_{CL}/128$

4. SPI 状态寄存器——SPSR

	位 7	6	5	4	3	2	1	0	
$0E($2E)	SPIF	WCOL	—	—	—	—	—	—	SPSR
读/写：	R/W	R/W	R	R	R	R	R	R	

初始化值：$00

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；或者可以通过先读 SPI 状态寄存器(SPSR)，再读 SPI 数据寄存器(SPDR)，对 SPIF 清 0。

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 WCOL 被置位。可以通过先读 SPI 状态寄存器(SPSR)，再读 SPI 数据寄存器(SPDR)，对 WCOL 清 0。

位 5～0——保留位。

在 AT90S8535 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。

5. 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：$00

SPI 数据寄存器可以读/写，用于在寄存器文件和 SPI 移位寄存器之间传送数据。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。

11.4 同步串行接口 SPI 应用举例

2 个单片机采用 SPI 同步通信交换数据电路如图 11.9 所示。其功能是各把自己 SRAM $100，$101，$102，$103 中 4 个字节送给对方，而把从对方传来的 4 个字节数送 SRAM $110，$111，$112，$113 中。2 机均采用 SPI 中断方式。

甲机为主方式，主程序把 $100 中的数送给 SPDR，启动 SPI 发送。在 SPI 中断子程中，把从乙机传来的数送到接收缓冲区，再把下一个数从发送缓冲区取来送给 SPDR，启动下一次发送。直到第 4 次 SPI 中断，甲机只接收对方发来的数据，不再给 SPDR 送数，启动 SPI 同步串行通信。

乙机为从方式，主程序把 $100 中的数送给 SPDR，等待甲机启动 SPI。在 SPI 中断服务子程中，把从甲机传来的数送接收缓冲区，再把下一个数从发送缓冲区取来送 SPDR，等待甲机启动 SPI。直到第 4 次 SPI 中断，乙机只接收对方发来的数据，不再给 SPDR 送数。

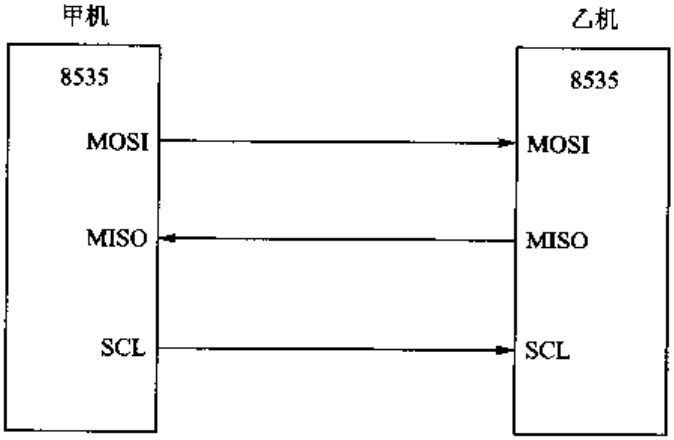


图 11.9 2 个单片机 SPI 同步数据传送电路图

甲机程序如下。乙机程序与甲机基本相同，只是把第 14 条指令改为 LDI R16，$E0，也就是改为从同步方式即可。注意 2 机运行时，乙机先开机工作等待甲机 SPI 发送。

```
.include "8535def.inc"
.org $0000
rjmp reset
.org $00a
rjmp SPI-STC
reset:  ldi r16,low(ramend)          ;栈指针置初值
        out spl,r16
        ldi r16,high(ramend)
        out sph,r16
        ldi X1, $00                  ;发送缓冲区指针 X 置初值
```

```
        ldi Xh, $01
        ldi Yl, $10                    ;接收缓冲区指针 Y 置初值
        ldi Yh, $01
        ldi r16, $f0                   ;甲机定义主同步方式,允许同步中断,对振荡器 4 分频
;***乙机定义从同步方式,允许同步中断,对振荡器 4 分频。将上条指令改为 ldi r16, $e0
        out spcr,r16
        ldi r16,0                      ;清中断标志
        out spsr,r16
        ldi r17, $03                   ;发 4 个字节
bb:     sei
        ld r16,X+                      ;读 $100
        out spdr,r16                   ;送 SPDR,主方式则发送
cc:     rjmp cc
  SPI-STC:
        in r1,sreg                     ;保护标志寄存器
        in r22,spdr                    ;读 SPI 数据寄存器
        st Y+,r22                      ;送 $110~$113 之一
        dec r17
        brne dd
        rjmp ee
dd:     ld r22,X+                      ;读 $101~$103 之一
        out spdr,r22                   ;送 SPI 数据寄存器
ee:     out sreg,r1                    ;恢复标志寄存器
        reti
```

第 12 章　AVR 单片机存储器编程

12.1　AVR 单片机编程

12.1.1　概　述

AVR 单片机编程有 4 种方式：利用计算机 RS232 串行口实行 ISP 串行下载编程；利用计算机并口(打印口)实行高速编程；利用 JTGA 接口实行编程，以上 3 种方式在 ATMEL AVR 集成软件 Studio3.52 以上版本均可支持；还有利用编程器实现并行编程。

12.1.2　ISP 串行下载编程接口

ISP 串行下载编程接口有 8 根信号线，如图 12.1 所示。

在$\overline{\text{RESET}}$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(时钟信号)、MOSI(输入)、MISO(输出)。当$\overline{\text{RESET}}$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。

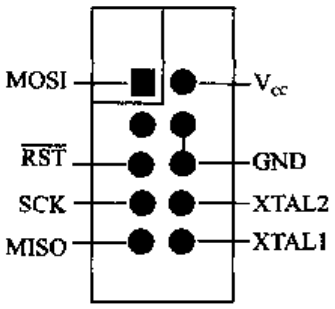


图 12.1　SL-AVRAD 开发实验器 ISP 信号座引脚功能

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)，而无需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。AT90S8535 程序存储器为 $0000～$0FFF；而 EEPROM 存储器为 $0000～$01FF。

可以用 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(SCK)的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。

12.1.3　ISP 串行下载编程操作

(1) 在光盘 *:\AVR\AVR\AVR PROG 下，双击图标 

进入串行下载窗口；也可把图标移到桌面成快捷菜单，双击图标进入串行下载操作窗口。如图 12.2 所示。

(2) 或安装双龙 AVR 下载软件，见图 12.3。使用 AVRPROG133 或 AVRPROG133 汉化版，均可出现图 12.2 所示 ISP 串行下载编程操作窗口。

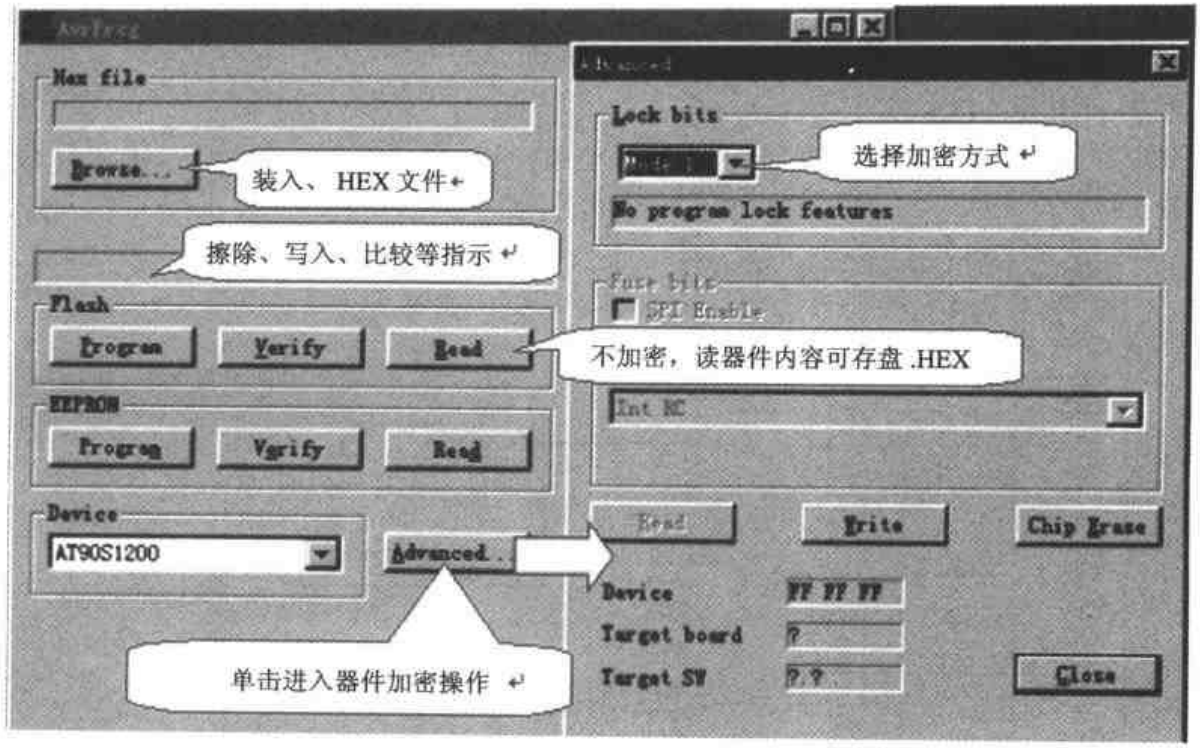


图 12.2　ISP 串行下载编程操作窗口

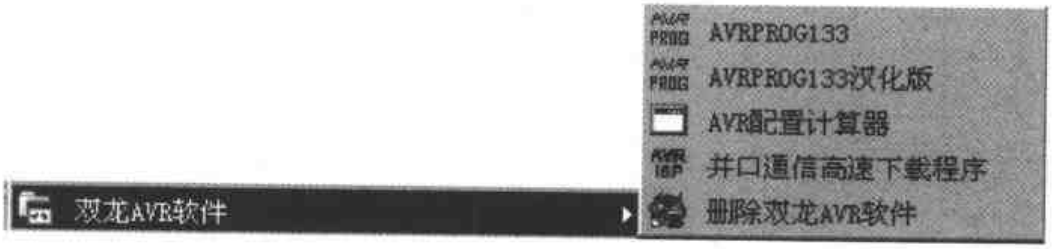


图 12.3　双龙 AVR 下载软件

(3) 启动 AVR Studio3.53 程序。如果对芯片编程，可以单击 TOOLS 菜单中的 STK500/AVR ISP/JTAG ICE 按钮。如图 12.4 所示。或送 AVR Prog 选项进入串行下载，或 Alt+9 快捷方式进入。

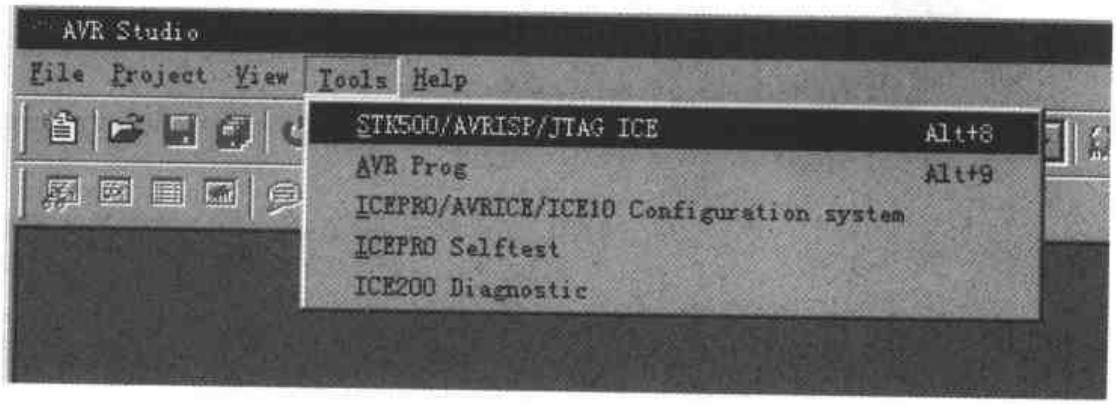


图 12.4　进入 AVR ISP

注意：必须在通电并连接 SL－AVRAD 开发实验器情况下，才会出现下载窗口。

12.1.4 并行下载编程接口电缆

1. 并行下载接口电缆

利用并行接口(打印口)可自制 AVR 下载电缆,如图 12.5 所示。如加上 74LS244 缓冲器则更好。

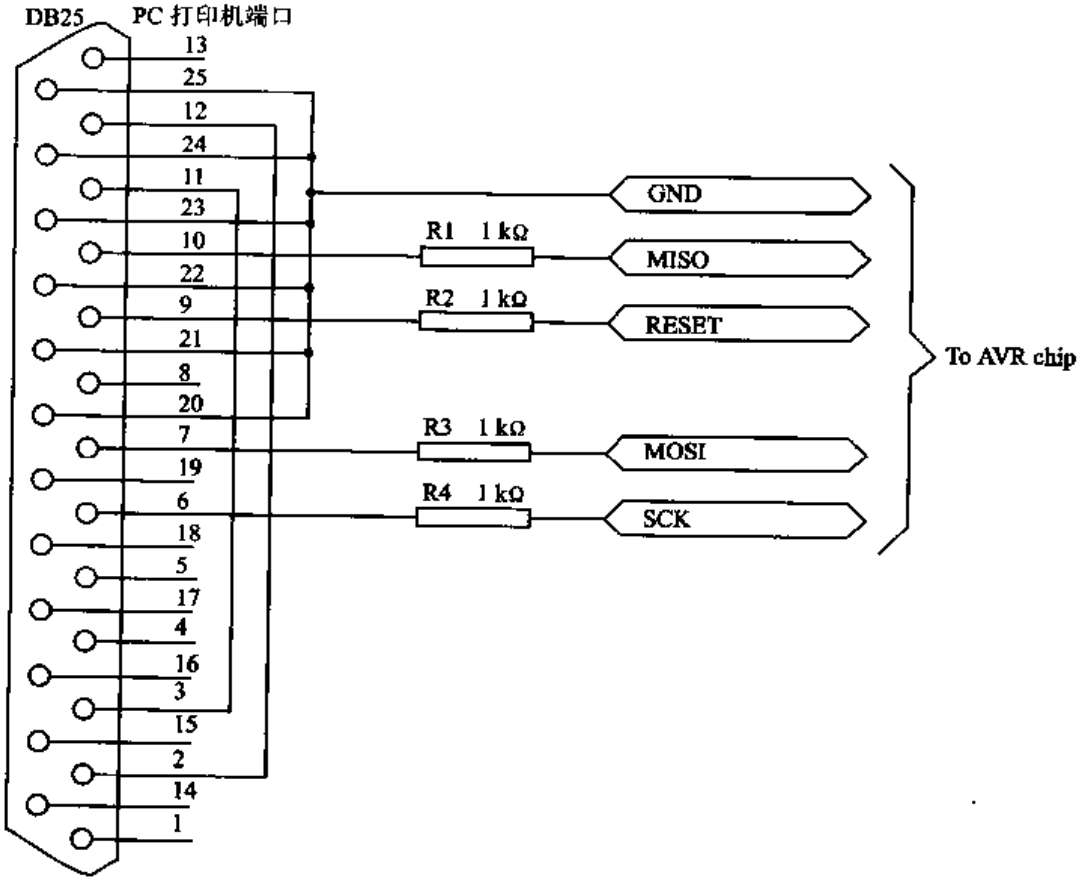


图 12.5 最简并口下载电缆线接线图

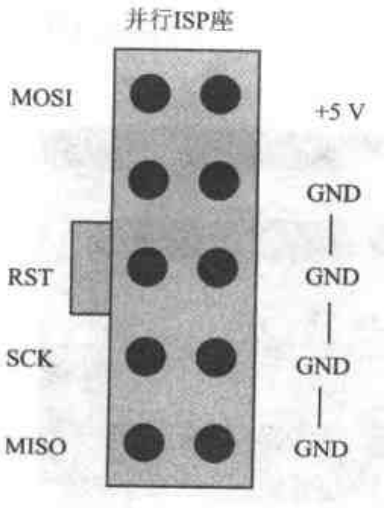


图 12.6 并行下载编程接口

2. 并行下载编程接口

SL－AVRISPL 开发实验器并行 ISP 接口插座接线如图 12.6 所示。

3. 并口下载编程操作

① 如图 12.3 所示,选【并口通信高速下载程序】选项进入。

② 如图 12.4 所示,选 AVR Prog 选项进入串行下载,或 Alt＋9 快捷方式进入。

4. 并口下载编程操作窗口

图 12.7 所示为并口下载编程操作窗口。

图 12.7　并口下载编程操作窗口

12.1.5　JTGA 下载编程操作

ATMEL AVR 高档 ATmega 系列单片机如 ATmega16/ATmega32/ATmega64/ATmega128 等器件，具有 JTGA ICE 仿真接口，可实现在线实时仿真及下载编程功能。

(1) JTGA ICE 接口插座及与 AVR 接线

JTGA ICE 接口插座引脚功能及与 AVR 单片机接线如图 12.8，图 12.9 所示。

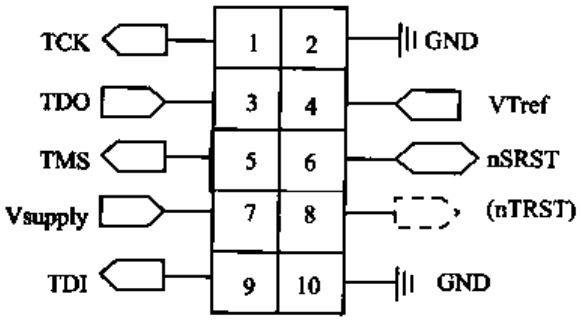


图 12.8　JTGA ICE 接口插座引脚功能

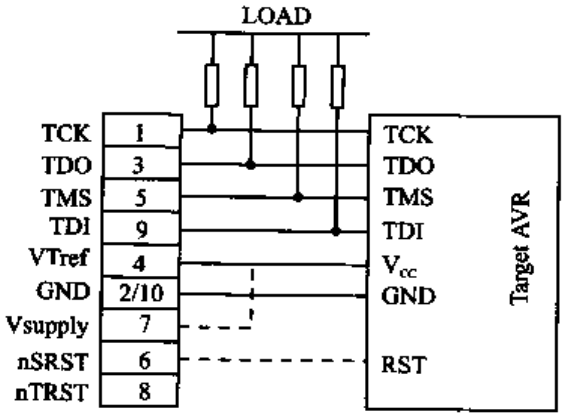


图 12.9　JTGA ICE 与 AVR 单片机接线图

(2) 操　作

如图 12.4 所示，选 TOOLS 菜单中的 STK500/AVR ISP/JTAG ICE 选项进入，或 Alt+8 快捷方式进入。

12.1.6　并行编程(万用编程器)

并行编程可参阅光盘内双龙电子书第 2 章 2.12.5 内容。

第 13 章　AVR 的 C 语言 ICCAVR 及应用

13.1　简　介

C 语言是一门具有结构化的语言，其特点是效率高和与系统十分接近。它的功能十分强。用 C，可以实现采用少量的结构解决复杂的问题；同时，C 表达式的经济性和它丰富的操作符集合也是它的一个特点。因而，采用 C 语言对单片机进行编程，在很多情况下，可以使程序简单，增加可读性。

13.1.1　C 程序的剖析

一个 C 程序必须定义一个 main 调用函数。编译器会将程序与启动代码和库函数链接成一个“可执行”文件，可以在目标系统中执行它。启动代码的用途在启动文件中被很详细地描述了。一个 C 程序需要设定目标环境，启动代码初始化这个目标，使其满足所有的要求。

通常，main 例程完成一些初始化后，无限循环地运行。下面例程的作用是点亮 LED。

```
#include<io8535.h>
void Delay()
{
unsigned char a,b;
for(a=1;a;a++)
    for(b=1;b;b++)
}

void LED_On(int i)
{
PORTB=~BIT(i);          /*低电平输出使 LED 点亮*/
Delay();
}

void main()
{ int i;
DDRB=0xFF;              /*定义 B 口输出*/
PORTB=0xFF;             /*B 口全部为高电平，对应 LED 熄灭*/
While (1)
{
/*LED 向前步进*/
for (i=0;i<8;i++)
```

```
        LED_On(i);
/*LED 向后步进*/
for(i=8;i>0;i--)
        LED_On(i);
/*LED 跳跃*/
     for (i=0;i<8;i+=2)
     LED_On(i);
     for (i=7;i>0;i-=2)
     LED_On(i);
     }
}
```

这个 main 例程很简单。在初始化一些 I/O 寄存器后，它运行在一个无限循环中，并且在这个循环中改变 LED 的步进图案。LED 是在 LED_On 例程中被改变的。在 LED_On 例程中直接写正确的数值到 I/O 端口。因为 CPU 运行很快，为能够看见图案变化，LED_On 例程调用了延时例程。

13.1.2　C 的运行结构

1. 数据类型

表 13.1 为数据类型表。

表 13.1　数据类型

类　型	长度/B	范　围
unsigned char	1	0～255
signed char	1	－128～127
char(*)	1	0～255
unsigned short	2	0～65 535
(signed)short	2	－32 768～32 767
unsigned int	2	0～65 535
(signed)int	2	－32 768～32 767
unsigned long	4	0～4 294 967 295
(signed)long	4	－2 147 483 648～2 147 483 647
Float	4	+/－1.175e－38～3.40e+38
double	4	+/－1.175e－38～3.40e+38

* 等同于“unsigned char”。

2. 程序与数据区的使用

(1) 程序存储器

程序存储器用于保存程序代码、常数表及确定数据的初始值如字符串、全局变量。编译器可以生成一个对应程序存储器映像的输出文件(HEX 文件)。这个文件可以被编程器用来对芯片编程。通常，编程器不能使用任意 64 KB 以上的程序存储器。

(2) 数据存储器(仅指内部 SRAM)

数据存储器用于保存变量、堆栈结构及动态内存分配的堆。通常它们不产生输出文件,但在程序运行时使用。程序使用数据内存如图 13.1 所示。内存图的底部是地址 $00,开始的 96 B($60)是 CPU 寄存器的 I/O 寄存器。编译器从 $60 往上放置全局变量和字符串,在变量区域的顶部可以分配动态内存。高端地址为硬件堆栈开始与 SRAM 的最后位置,在它的下面是向下生长的软件堆栈。程序设计人员要确保硬件堆栈不生长进软件堆栈,而软件堆栈不生长进硬件堆栈;否则,将导致意外的结果。

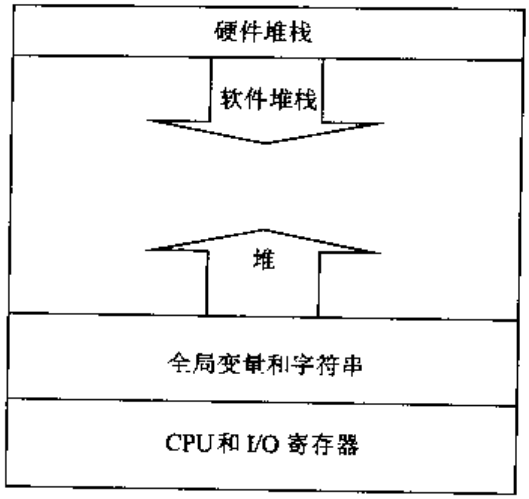


图 13.1 内存图

(3) 数据存储器(外部 SRAM)

如果选择带有 32 KB 或 64 KB 外部 SRAM 的目标装置,那么堆栈放置在内部 SRAM 的顶部,并且向下朝低端地址生长。数据内存开始于硬件堆栈的顶部,并且向上生长。这样分配的原因是,多数场合访问内部 SRAM 比访问外部 SRAM 的速度快,分配堆栈到较快的内存是有很多好处的。

3. 编程区域

编程器生成代码和数据到不同的区域(area)。区域按照内存地址增高的顺序被编译器使用。

(1) 只读存储器

interrupt vector——这个区域包括中断向量。

func-lit——函数表区。这个区域的每个字包括了函数入口的地址。

lit——这个区域包括了整型数和浮点数常量。

idata——全局变量和字符串的初始值保存在这个区域。

text——这个区域包括程序代码。

(2) 数据内存

data——这个区域包括全局变量、静态变量及字符串。全局变量和字符串初始值保存在"idata"区域,并且在启动时被拷贝进数据区。

bss——这个区域包括未初始化的全局变量。

(3) EEPROM 存储器

EEPROM——这个区域包括 EEPROM 数据。

13.2　AVR 硬件访问的编程

AVR 系列使用 C 语言编程时，允许对访问目标 MCU 的底层硬件进行访问。头文件 io*.h(如 io8515.h,io8535.h)定义了指定 AVR MCU 的 I/O 寄存器细节。

13.2.1　位操作

位操作的任务是编程微控制器 MCU 打开或关闭 I/O 寄存器的一些位(bit)。标准 C 有较好的和适用的位操作功能，而没有借助于汇编指令或其他非标准 C 结构。C 定义了一些按位进行的运算：

● a|b——按位或。这个表达式指示 a 被表达式中的 b 按位进行或运算。常用于打开某些位，尤其常用|=的形式。

如：PORTA|=0x80;//打开 a 口的位 7(最高位)

● a&b—— 按位与。此运算在检查某些位是否置 1 时很有用。

如：If((PORTA&0x81)=0)//检查位 7 和位 0

注意：圆括号应括在运算符的周围，因为它和“=”相比运算优先级较低。

● a^b—— 按位异或。此运算对一位取反有用。

如：PORTA^0x80;//翻转位 7

● ～a——按位取反。在表达式中这个运算执行一个取反。当用按位与运算关闭某些位时，与这个运算组合使用尤其有用。

如：PORTA&=～0x80;//关闭位 7

13.2.2　程序存储器和常量数据

AVR 是 Harvard 结构的 MCU。它的程序存储器和数据存储器是分开的。这样的结构具有一定的优越性。如分开的地址空间允许 AVR 装置比传统结构访问更多的存储器；但是，C 指针是任意一个数据指针或函数指针，C 规则已经指定，不可能假设数据和函数指针能被向前或向后修改。可 Harvard 结构的 AVR，要求数据指针能指向任一个数据内存和程序内存；因而 ImageCraft AVR 编译器使用“const”限定词，表示项目是在程序存储器中。

注意：这个 const 限定词可以应用于不同的场合，不管是限定指针变量自身，还是指向项目的指针。

如：const int table[]={1,2,3};

```
const char *ptr1;
char *const ptr2;
const char *const ptr3;
```

“table”是按表格式样分配进程序存储器的；“ptr1”是一个项目在数据存储器，而指向数据的指针在程序存储器；“ptr2”是一个项目在程序存储器，而指向数据的指针在数据存储器；“ptr3”是项目在程序存储器，而指向数据的指针也在程序存储器。在大多数的例子中，“table”和“ptr1”是很典型的。

13.2.3 堆 栈

生成代码使用2个堆栈:一个是用于子程序调用和中断操作的硬件堆栈;一个是用于以堆栈结构传递参数、临时变量及局部变量的软件堆栈。

硬件堆栈起初用于存储函数返回的地址,代表了许多小的软件堆栈。通常,如果程序没有子程序调用,也不调用像带有%f格式的printf()等库函数,那么默认的16 B在大多数情况下,应该能良好地工作。在绝大多数程序中,除了很繁琐的递归调用程序,最多40 B的硬件堆栈就足够了。硬件堆栈是从数据内存的顶部开始分配的;而软件堆栈是在它下面一定数量字节处分配。硬件堆栈和数据内存的大小是受在编译器选项中的目标装置项设定限制的。

1. 堆栈检查

任意一个程序失败的重要原因是,堆栈溢出到其他数据内存的范围。2个堆栈中的任意一个都可能溢出,并且当一个堆栈溢出时,会偶然产生坏的影响。可使用堆栈检查函数,检测溢出的情况。

2. 堆栈检查函数

有几个库函数是用于检查堆栈是否溢出的。见图13.2。

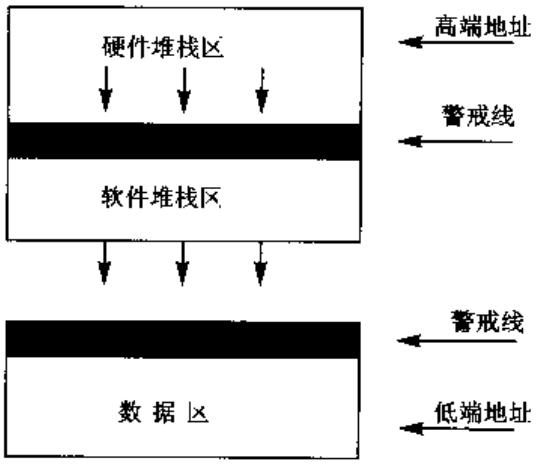


图13.2 内存图

如果硬件堆栈增长到软件堆栈中,那么软件堆栈的内容将会改变,即局部变量和别的堆栈项目被改变。硬件堆栈用作函数的返回地址,若函数的调用层次太深,偶然会出现这种情况。同样,若软件堆栈溢出进数据区,将会改变全局变量或其他静态分配的项目。这种情况在定义了太多的局部变量时,或一个局部集合变量太大时,也会偶然出现。

● 警戒线 启动代码写了一个正确的、关于数据区的地址字节和一个类似的、正确的、关于软件堆栈的地址字节作为警戒线。

● 堆栈检查 调用_StackCheck(void)函数检查堆栈溢出。如果警戒线字节仍然保持正确的值,那么函数检查通过;如果堆栈溢出,那么警戒线字节将可能被破坏。当程序堆栈溢出时,程序可能运行不正常或偶然崩溃。当_StackCheck检查错误条件时,调用了带一个参数的函数_StackOverflowed(char c)。如果参数是1,那么硬件堆栈有过溢出;如果参数是0,那么软件堆栈曾经溢出。

13.2.4　在线汇编

除了在汇编文件中写汇编函数外，在线汇编允许将汇编代码写进 C 文件中。在线汇编的语法是：asm("<string>");。

多个汇编声明可以被符号\n 分隔成新的一行。"string"可以被用来指定多个声明。为了在汇编声明中访问一个 C 的变量，可使用%〈变量名〉格式。如：

```
register unsigned char uc;
  asm("mov %uc,R0\n"
      "sleep\n");
```

任意一个 C 变量都可以被引用。如果在汇编指令中需使用一个 CPU 寄存器，必须使用寄存器存储类(register)，强制分配一个局部变量到 CPU 中。

通常使用在线汇编引用局部寄存器的能力是有限的。如果在函数中描述了太多的寄存器变量，就可能没有寄存器可用。在这种情况下，汇编程序时将出错。在线汇编可以用在 C 函数的内部或外部。编译器将在线汇编的每行都分解成可读的。

13.2.5　中断操作

C 可以使用中断操作。无论函数定义在文件的什么地方，必须用一个附注(pragma)在函数定义之前通知编译器，这个函数是一个中断操作：

```
#pragma interrupt_handler<name>:<vector number>
```

"vector number"是中断的向量号。向量号是从 1 开始的，那是复位向量。这个附注的作用是：对中断操作函数，编译器生成 RETI 指令，代替 RET 指令，而且保存和恢复在函数中用过的全部寄存器。

例如：

```
#pragma interrupt_handler timer_handler:4
...
void timer_handler()
{
  ...
}
```

编译器生成的指令：

```
rjmp_timer_handler;
```

上述指令定位在$06。使用 2 个字作为中断向量。

如果希望对多个中断入口使用同一个中断操作，可以在一个 interrupt_handler 附注中放置多个用空格分开的名称，分别带有多个不同的向量号。如：

```
#pragma interrupt_handler timer_ovf:7 timer_ovf:8
```

13.2.6　访问 UART

默认的库函数 getchar 和 putchar 使用查询模式从 UART 中进行读写。

13.2.7 访问 EEPROM

EEPROM 在运行时可以使用库函数访问，在调用这些函数之前加入#include<eeprom.h>。

1. 内部函数

如果需要下列函数可以直接使用：

unsigned char EEPROMread(int location)　　从 EEPROM 指定位置读取一个字节。

int EEPROMwrite(int location,unsigned char byte)　　写一个字节到 EEPROM 指定位置，如果成功，返回 0。

void EEPROMReadBytes(int location,void * ptr,int size)——从 EEPROM 指定位置处开始读取“size”个字节至由“ptr”指向的缓冲区。

void EEPROMWriteBytes(int location,void * ptr,int size)——从 EEPROM 指定位置处开始写“size”个字节，写的内容由“ptr”指向的缓冲区提供。

2. 宏

EEPROM_READ(int location,object)——调用了 EEPROMReadBytes 函数，从 EEPROM 指定位置读取数据送给数据对象。“object”可以是任意程序变量，包括结构和数据。如：

```
int i;
EEPROM_READ(0x1,i);//读 2 个字节给 i
```

EEPROM_WRITE(int location,object)——调用了 EEPROMWriteBytes 函数，将数据对象写入到 EEPROM 的指定位置。“object”可以是任意程序变量，包括结构和数组。如：

```
int i;
EEPROM_WRITE(0x1,i);//写 2 个字节至 0x1
```

3. 初始化 EEPROM

EEPROM 可以在程序源文件中初始化。在 C 源文件中，它作为一个全局变量被分配到特殊调用区域“eeprom.”中。这可通过附注实现，产生扩展名为.eep 的输出文件。如：

```
#pragma data:eeprom
  int foo=0x1234;
  char table[ ]={0,1,2,3,4,5};
#pragma data:data
...
int i;
EEPROM_READ((int)&foo,I); //I 等于 0x1234
```

上面的例子中，第 2 个附注是必须的。为返回默认的“data.”区域，需要重设数据区名称。

13.3 常用库函数

13.3.1 头文件

io*.h(io2313.h,io8515.h,io8535.h)——这些文件是与所选用的 ATMEL 芯片对应的。

macros. h——这个文件包含了许多有用的宏和定义。

下列标准的 C 头文件是被支持的。如果程序使用了头文件所列出的函数，那么应该包含该头文件。在使用浮点数和长整型数的程序中，必须使用 # include 预编译指令，包含这些包含了函数原形的头文件：

assert. h——assert()，声明宏。

ctype. h——字符类型函数。

float. h——浮点数原形。

limits. h——数据类型的大小和范围。

math. h——浮点运算函数。

stdrag. h——变量参数表。

stddef. h——标准定义。

stdio. h——标准输入输出(I/O)函数。

stdlib. h——包含内存分配函数的标准库。

string. h——字符串处理函数。

13.3.2　字符类型库

下列函数按照输入的 ASCII 字符集字符分类。使用这些函数之前，应当用 # include <ctype. h>包含。

int isalnum(int c)——如果 c 是字母或数字，返回非 0 数值；否则返回 0。

int iscntrl(int c)——如果 c 是控制字符(如 FF，BELL，LF…等)，返回非 0 值；否则返回 0。

int isalpha(int c)——如果 c 是字母，返回非 0 值；否则返回 0。

int isdigit(int c)——如果 c 是数字，返回非 0 值；否则返回 0。

int isgraph(int c)——如果 c 是一个可打印字符而非空格，返回非 0 值；否则返回 0。

int islower(int c)——如果 c 是小写字母，返回非 0 值；否则返回 0。

int isprint(int c)——如果 c 是一个可打印字符，返回非 0 值；否则返回 0。

int ispunct(int c)——如果 c 是一个可打印字符，而不是空格、数字或字母，返回非 0 值；否则返回 0。

int isspace(int c)——如果 c 是一个空格字符，返回非 0 值，包括空格 CR，FF，HT，NL 及 VT；否则返回 0。

int isupper(int c)——如果 c 是大写字母，返回非 0 值；否则返回 0。

int isxdigit(int c)——如果 c 是十六进制数字，返回非 0 值；否则返回 0。

int tolower(int c)——如果 c 是大写字母，返回 c 对应的小写字母；其他类型返回 c。

int toupper(int c)——如果 c 是小写字母，返回 c 对应的大写字母；其他类型返回 c。

13.3.3　浮点类型库

下列函数支持浮点数运算。使用这些函数之前，必须用“include <math. h>”包含。

float asin(float x)——以弧度形式返回 x 的反正弦值。

float acos(float x)——以弧度形式返回 x 的反余弦值。

float atan(float x)——以弧度形式返回 x 的反正切值。

float atan2(float x,float y)——返回 y/x 的反正切,其范围在$-\pi\sim+\pi$之间。

float ceil(float x)——返回对应 x 的一个整型数,小数部分四舍五入。

float cos(float x)——返回以弧度形式表示的 x 的余弦值。

float cosh(float x)——返回 x 的双曲余弦值。

float exp(float x)——返回以 e 为底的 x 的幂,即 e^x。

float exp10(float x)——返回以 10 为底的 x 的幂,即 10^x。

float fabs(float x)——返回 x 的绝对值。

float floor(float x)——返回不大于 x 的最大整数。

float fmod(float x,float y)——返回 x/y 的余数。

float frexp(float x,int * pexp)——把浮点数 x 分解成数字部分 y(尾数)和 2 的 n 次幂两个部分,即 $x=y*2^n$。y 的范围为 $0.5\leqslant y<1$,y 值被函数返回;而 n 值存放在 pexp 指向的变量中。

float fround(float x)——返回最接近 x 的整型数。

float ldexp(float x,int exp)——返回 $x*2^{exp}$。

float log(float x)——返回 x 的自然对数。

float log10(float x)——返回 x 的以 10 为底的对数。

float modf(float x,float * pint)——把浮点数分解成整数部分和小数部分。整数部分存放到 pint 指向的变量;小数部分应当大于或等于 0 而小于 1,并且作为函数返回值返回。

float pow(float x,float y)——返回 x^y 值。

float sqrt(float x)——返回 x 的平方根。

float sin(float x)——返回以弧度形式表示的 x 的正弦值。

float sinh(float x)——返回 x 的双曲正弦函数值。

float tan(float x)——返回以弧度形式表示的 x 的正切值。

float tanh(float x)——返回 x 的双曲正切函数值。

13.3.4 标准输入输出库

标准的文件输入输出是不能真正植入微控制器(MCU)的。标准 stdio.h 的许多内容不可以使用,但有一些 I/O 函数是被支持的。同样使用之前,应用"#include<stdio.h>"预处理,并且需要初始化输出端口。最低层的 I/O 程序是单字符的输入(getchar)和输出(putchar)程序。如果需要针对不同的装置使用高层的 I/O 函数,例如用 printf 输出 LCD,则需要全部重新定义最低层的函数。

为在 ATMEL 的 AVR Studio 模拟器(终端 I/O 窗口)使用标准 I/O 函数,应当在编译选项中选中相应的单选钮。

int getchar()——使用查寻方式从 UART 返回一个字符。

int printf(char * fmt,..)——按照格式说明符输出格式化文本 fmt 字符串。格式说明符是标准格式的一个子集。

%d——输出有符号十进制整数。

%o——输出无符号八进制整数。

%x——输出无符号十六进制整数。

%X——除大写字母使用 A～F 外，同%x。

%u　　输出无符号十进制整数。

%s——输出一个以 C 中空字符 NULL 结束的字符串。

%c——以 ASCII 字符形式输出，只输出一个字符。

%f——以小数形式输出浮点数。

%S ——输出在 Flash 存储器中的字符串变量。

printf 支持 3 个版本，取决于需要和代码的大小(越高的要求，代码越大)。

基本型：只有%c，%d，%x，%u 及%s 格式说明符是承认的。

长整型：针对长整型数，%ld，%lu，%lx 被支持，以适用于精度要求较高的领域。

浮点型：全部格式包括%f 被支持。

可使用编译选项对话框选择版本，代码大小的增加是值得关注的。

int putchar(int c)——输出单个字符。这个库程序使用了 UART，以查寻方式输出单个字符。注意输出"\n"字符至程序终端窗口。

int puts(char * s)——输出以 NL 结尾的字符串。

int sprintf(char * buf,char * fmt)——按照格式说明符输出格式化文本 fmt 字符串到一个缓冲区，格式说明符同 printf()。

"const char * "支持功能——cprintf 和 csprintf 将 Flash 中的格式字符串分别以 prinf 和 sprinf 形式输出。

13.3.5　标准库和内存分配函数

标准库头文件<stdlib.h>定义了宏 NULL，RAND_MAX 及新定义的类型 size_t，并且描述了下列函数。**注意**：在调用任意内存分配程序(如 calloc，malloc 及 realloc)之前，必须调用 New_Heap，初始化堆 heap。

int abs(int I)——返回 i 的绝对值。

int atoi(char * s)——转换字符串 s 为整型数并返回它。字符串 s 起始必须是整型数形式字符；否则返回 0。

double atof(const char * s)——转换字符串 s 为双精度浮点数并返回它。字符串 s 起始必须是浮点数形式字符串。

long atol(char * s)——转换字符串 s 为长整型数并返回它。字符串 s 起始必须是长整型数形式字符串；否则返回 0。

void * calloc(size_t nelem,size_t soze)——分配"nelem"个数据项的内存连续空间。每个数据项的大小为 size 字节，并且初始化为 0。如果成功，返回分配内存单元的首地址；否则返回 0。

void exit(status)——终止程序运行，典型的是无限循环。它担任拥护 main 函数的返回点。

void free(void * ptr)——释放 ptr 所指向的内存区。

void * malloc(size_t size)——分配 size 字节的存储区。如果分配成功，则返回内存区地址；如内存不够分配，则返回 0。

void NewHeap(void * start,void * end)——初始化内存分配程序的堆。一个典型的调用是将符号_bss_end＋1 的地址用作"start"值。符号_bss_end 定义为，编译器用来存放全局变量和字符串的数据内存的结束。加 1 的目的是，堆栈检查函数使用_bss_end 字节存储为标

志字节，这个结束值不能被放入堆栈中。

```
extern char_bss_end;
_NewHeap(&_bss_end-1,&_bss_end+201);//初始化200 B大小的堆
```

int rand(void)——返回一个在0和RAND_MAX之间的随机数。

void *realloc(void *ptr,size_t size)——重新分配ptr所指向的内存区大小为size字节。size可比原来大或小，返回指向给内存区的地址指针。

void srand(unsigned seed)——初始化随后调用的随机数发生器的种子数。

long strtol(char *s,char **endptr,int base)——按照"base."的格式转换"s"中起始字符为长整型数。如果"endptr"不为空，"*endptr"将设定"s"中转换结束的位置。

unsigned long strtoul(char *s,char **endptr,int base)——除返回类型为无符号长整型数外，其余同"strtol"。

13.3.6 字符串函数

用"#include <string.h>"预处理后，编译器支持下列函数。<string>定义了NULL、类型size_t及下列字符串和字符阵列函数。

void *memchr(void *s,int c,size_t n)——在字符串s中搜索n个字节长度，以寻找与c相同的字符。如果成功，返回匹配字符的地址指针；否则返回NULL。

int memcmp(void *s1,void *s2,size_t n)——对字符串s1和s2的前n个字符进行比较。如果相同，返回0；如果s1中字符大于s2中字符，则返回1；如果s1中字符小于s2中字符，则返回-1。

void *memmove(void *s1,void *s2,size_t n)——拷贝s2中n个字符至s1。其与memcpy基本相同；但拷贝区可以重叠。

void *memset(void *s,int c,xize_t n)——在s中填充n个字节的c，它返回s1。

char *strcat(char *s1,char *s2)——拷贝s2到s1的结尾，返回s1。

char *strchr(char *s1,int c)——在s1中搜索第1个出现的c，包括NULL结束字符。如果成功，返回指向匹配字符的指针；如果没有匹配字符找到，返回空指针。

int strcmp(char *s1,char *s2)　比较2个字符串。如果相同，返回0；如果s1>s2，则返回1；如果s1<s2，则返回-1。

char *strcpy(char *s1,char *s2)——拷贝字符串s2至字符串s1，返回s1。

size_t strcspn(char *s1,char *s2)——在字符串s1搜索与字符串s2匹配的第1个字符。包括NULL结束字符，其返回s1中找到的匹配字符的索引。

size_t strlen(char *s)——返回字符串s的长度，不包括NULL结束字符。

char *strncat(char *s1,char *s2,size_t n)——拷贝字符串s2(不包含NULL结束字符)中n个字符到s1。如果s2长度比n小，则只拷贝s2，返回s1。

int strncmp(char *s1,char *s2,size_t n)——基本和strcmp函数相同，但其只比较前n个字符。

char *strncpy(char *s1,char *s2,size_t n)——基本和strcpy函数相同，但其只拷贝前n个字符。

char * strpbrk(char * s1,char * s2)——基本和 strcspn 函数相同，但它返回的是在 s1 匹配字符的地址指针；否则返回 NULL 指针。

char * strrchr(char * s,int c)——在字符串 s 中搜索最后出现的 c，并返回它的指针；否则返回 NULL。

size_t strspn(char * s1,char * s2)——在字符串 s1 搜索与字符串 s2 不匹配的第 1 个字符，包括 NULL 结束字符。其返回 s1 中找到的第 1 个不匹配字符的索引。

char * strstr(char * s1,char * s2)—— 在字符串 s1 中找到与 s2 匹配的子字符串。如果成功，返回 s1 中匹配字符串的地址指针；否则返回 NULL。

13.3.7　变量参数函数

<stdarg.h>——提供再入式函数的变量参数处理。它定义了不确定的类型 va_list 和 3 个宏。

va_start(va_list foo,<last-arg>)—— 初始化变量 foo。

va_arg(va_list foo,<promoted type>)——访问下一个参数，分派指定的类型。注意类型必须是高级类型，如 int，long 或 double；低级的整型类型如"char"不能被支持。

va_end(va_list foo)——结束变量参数处理。

例如：printf()可以使用 vprintf()来实现。

```
#include<stdarg.h>
int printf(char * fmt,…)
{
va_list ap;
va_start(ap,fmt);
  vfprintf(fmt,ap);
  va_end(ap);
  }
```

13.4　ICCAVR 的 IDE 环境

1. 编译一个单独的文件

建立一个输出文件应首先建立一个工程文件，并且定义属于这个工程的所有文件；但是，有时需要将一个文件单独地编译为目标文件或最终的输出文件。可以这样操作：从 IDE 菜单 File 中选择 Compile File…命令，执行 to Object 和 to Output 中的任意一个。当调用这个命令时，文件应该是打开的，并且在编辑窗口中可以编辑。

编译一个文件为目标文件(to Object)，对检查语法错误和编译一个新的启动文件是很有用的；编译一个文件为输出文件(to Output)，对较小的并且是一个文件的程序较为有用。

2. 创建一个新的工程

为创建一个新的工程，从菜单 Project 中选择 New 命令。IDE 会弹出一个对话框，在对话框中可以指定工程的名称，这也是输出文件的名称。如果使用一些已经建立的源文件，则可在菜单 Project 中选择 AddFile(s)命令。

另外，还可以在菜单 File 中选择 New 命令，建立一个新的源文件，以输入用户代码。可在

菜单 File 中选择 Save 或 Save as 命令保存文件。然后可以像上面所述，调用 AddFile(s)命令，将文件加入到工程中；也可在当前编辑窗口中右击 Add to Project，将文件加入已打开的工程列表中。通常输出的源文件在工程的同一个目录中，但也可不这样要求。

工程的编译选项菜单使用菜单 Project 中的 Option 命令。

3. 工程管理

工程管理允许将多个文件组织进同一个工程，而且可定义它们的编译选项。这个特性允许将工程分解成许多小的模块。当处理工程构筑时，只有一个文件被修改和重新编译。如果一个头文件做了修改，当编译包含这个头文件的源文件时，IDE 会自动重新编译已经改变的头文件。

一个源文件可以写成 C 或汇编格式的任意一种。C 文件必须使用".c"扩展名，汇编文件必须使用".s"扩展名。可以将任意文件放在工程列表中，例如可以将一个工程文档文件放在工程管理窗口中。工程管理器在构筑工程时，对源文件以外的文件不予理睬。

对目标器件不同的工程，可以在编译选项中设置有关参数。当新建一个工程时，使用默认的编译选项。可以将现有编译选项设置成默认选项，也可将默认编译选项装入现有工程中。默认编译选项保存在 default.prj 文件中。

为避免工程目录混乱，可以指定输出文件和中间文件到一个指定的目录。通常这个目录是工程目录的一个子目录。

4. 编译窗口

编译窗口是用户与 IDE 交流信息的主要区域。在这个窗口中，可以修改相应的文件。当编译存在错误时，单击有关错误信息，编译器会自动将光标定位在错误行的位置。

注意：对 C 源文件中缺少分号";"的错误，编译器定位于其下面一行。

5. 应用构筑向导

应用构筑向导是用于创建外围设备初始化代码的一个图形界面。可以单击工具条中的 Wizard 按钮或菜单 Tools 中的 ApplicationBuilder 命令调用它。

应用构筑向导显示目标 MCU 的每一个外围设备子系统，它的使用是很显而易见的。在这里可以设置 MCU 所具有的中断、内存、定时器、I/O 端口、UART、SPI 及模拟量比较器等外围设备，并产生相应的代码。如果需要的话，还可产生 main()函数。

6. 状态窗口

状态窗口显示 IDE 的状态信息。

7. 终端仿真

IDE 有一个内置的终端仿真器。它不包含任意一个 ISP(在系统编程)功能；但它可以作为一个简单的终端，或许可以显示调试信息，也可下载一个 ASCII 码文件。

13.5 实 例

1. 访问外围

在 AVR 中，所有的 I/O 寄存器在头文件里都被定义为特殊功能寄存器，可以像普通变量一样访问。

```
#include <io8535.h>                    /* 定义所用芯片为 8535 */
void main(void)
```

```
{
DDRD=0xFF;                    /* Port D 输出 */
}
```

2. 读/写口

```
#include <io8535.h>           /* 定义所用芯片为 8535 */
void main(void)
{
char c;
DDRB=0xFF;                    /* PortB 输出 */
  for(;;)                     /* 死循环 */
{
  c=PIND;                     /* 读 Port D */
  PORTB=c;                    /* 回写到 Port B */
}
}
```

3. 延时函数

```
#include <io8535.h>
void delay()
{
unsigned int i,delayValue;
for(i=0;i<delayValue;i++)
{;}

}
```

4. 读/写 EEPROM

```
#include "io8535.h"
#include "eeprom.h"
void main()
{
 int i,k,p;
 i=2000;
 EEPROM_WRITE(0x10,i);        //向 EEPROM 置入参数
 EEPROM_READ(0x10,p);         //读取 EEPROM 参数
 i=0;
 EEPROM_WRITE(0x12,i);        //向 EEPROM 置入参数
 EEPROM_READ(0x12,p);         //读取 EEPROM 参数
}
```

5. 8 个数据排序

8 个数据存放在数组 a[8]中，现在要对它们进行从大到小排序。采用的方法是两两比较，

若顺序不对，就交换位置。其程序如下：

```
#include "io8535.h"
#include "math.h"
void main()
{
int a[9];
int i,j;
for(i=0;i<8;i++)
 {for(j=0;j<8-i;j++)
  {if(a[j]<a[j+1])
   {a[8]=a[j];
a[j]=a[j+1];
a[j+1]=a[8];}                    //以 a[8]作为中间变量进行数据交换
  }
 }
}
```

6. A/D 转换

```
#include "io8535.h"
void main()
{
 int reg,adc;
 SP=0x025F;
 DDRA=0x00;
 ADCSR=0xE5;                     //A/D 采用 32 分频
 ADMUX=6;                        //采集第 6 路的数据
for(reg=0;reg<3000;reg++)
  {;}                            //延时
 adc=ADC;                        //ADC 中存放 A/D 转换结果
}
```

7. 综合应用程序

量程为 0～4 MPa 的压力变送器输出 4～20 mA 信号，经 250 Ω 电阻转换成 1～5 V DC 信号。调节 V_{REF}，使模拟输入信号为 1 V 时，A/D 转换的数字量为 200；5 V 时为 1 000。该输入信号经 8 次 A/D 转换，转换结果按大小排序，取中间 4 个值的平均值，根据标度变换公式可求得：p=5x－1 000。计算结果二转十后，由 4 位数码管动态扫描显示。数码管共阴极，PB 作字线，PD 低 4 位作位线。采用定时器 2 的溢出中断，8 MHz 晶振，每 3.9 ms 溢出中断 1 次，扫描 1 位，每 15.6 ms 扫描 1 遍。

A/D 转换并定时中断动态扫描显示程序：

```
#include "io8535.h"
#include "math.h"
#pragma interrupt_handler timer:5
```

```
int word,data,data_temp,out_temp,adc[9],ip,jp,reg;
float temp;
static int s[16]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x6F};
void paixu()
{
 for(ip=0;ip<8;ip++)
 {
  for(jp=0;jp<8-ip;jp++)
   {
   if(adc[jp]<adc[jp+1])
    {
     adc[8]=adc[jp];
     adc[jp]=adc[jp+1];
     adc[jp+1]=adc[8];
    }
   }
 }
 adc[8]=adc[2]+adc[3]+adc[4]+adc[5];
}
void main()
{
 SP=0x025F;
 DDRA=0x00;
 DDRB=0xFF;
 DDRD=0xFF;
 PORTB=0x00;
 TIMSK=0x40;          //允许 T/C2 溢出中断
 TCNT2=0x00;          //T/C2 计数初值
 TCCR2=0x05;          //T/C2 采用 128 分频
 ADCSR=0xE5;          //ADC 采用 32 分频
 ADMUX=6;             //采集第 6 路的数据
 word=0xfe;           //置位线初值
 SREG=0x80;           //允许中断
 for(;;)
 {
  for(ip=0;ip<8;ip++)
  {
   for(reg=0;reg<3000;reg++)
   {;}
   adc[ip]=ADC;
   }
  paixu();             //调用排序函数,对数据进行处理
  temp=adc[8]/4;
```

```
    temp=5 * temp;          //标度变换(y=5 * x-1000)
    temp=temp-1000;
    data_temp=temp;
    data=data_temp%10;data_temp=data_temp/10;data=s[data];
    * (volatile unsigned char * )0x100=data;
    data=data_temp%10;data_temp=data_temp/10;data=s[data];
    * (volatile unsigned char * )0x101=data;
    data=data_temp%10;data_temp=data_temp/10;data=s[data];
    * (volatile unsigned char * )0x102=data;
    data=data_temp%10;data=s[data];
    * (volatile unsigned char * )0x103=data;
  }//二转十,求得个、十、百、千位,并查出相应位的7段码,依次存于SRAM的$100~$103
}
void timer()
{ // T/C2中断执行程序,完成4路数码动态扫描显示
 int sreg_temp;
 sreg_temp=SREG;
 switch(word)
 {
  case 0xfe:
   PORTD=0xfe;
   out_temp= * (volatile unsigned char * )0x100;
   PORTB=out_temp;
   word=0xfd;break;
  case 0xfd:
   PORTD=0xfd;
   out_temp= * (volatile unsigned char * )0x101;
   PORTB=out_temp;
   word=0xfb;break;
  case 0xfb:
   PORTD=0xfb;
   out_temp= * (volatile unsigned char * )0x102;
   PORTB=out_temp;
   word=0xf7;break;
  case 0xf7:
   PORTD=0xf7;
   out_temp= * (volatile unsigned char * )0x103;
   PORTB=out_temp;
   word=0xfe;break;
 }
 SREG=sreg_temp;
}
```

第 14 章　AVR 单片机开发工具及应用

14.1　AVR 的开发工具

AVR 的开发工具可分为硬件仿真器、软件模拟仿真器(开发实验器)。它们均在 AVR 集成开发环境(IDE)下工作,并有多种 C 高级语言支持。

14.2　AVR 实时在线仿真器 ICE－200

广州市天河双龙电子有限公司引进美国原装 AVR 实时在线仿真器 ICE－200,如图 14.1 所示。它可在线仿真 AVR 的器件 ATtiny10/11/12(V/L),AT90S1200/2313,AT90(L)S2333/4433,AT90S4414/8515,AT90(L)S4434/8535。由于仿真器的电源不对外,所以 ICE－200 也支持低电压器件。

ICE－200 采用 AVR 专用仿真 CPU 与监控 CPU 独立设计的方案,充分提供各种调试手段,真实再现被仿 AVR 的各种特性。

ICE－200 仿真软件的最新版集成环境(IDE)为 Stuido3. X。除支持以上 11 种 AVR 外,还可模拟其他 AVR 器件的运行,支持汇编和 C 高级语言的调试、JTGA ICE 仿真及程序下载等操作。

其中汇编级编译器免费提供;C 编译器只提供 Image Craft Inc. 的 30 天免费试用版 ICCAVR demo,30 天后转为 2 KB 限制版。该软件及其升级版均可从互联网(www. image-craft. com)上免费获得。

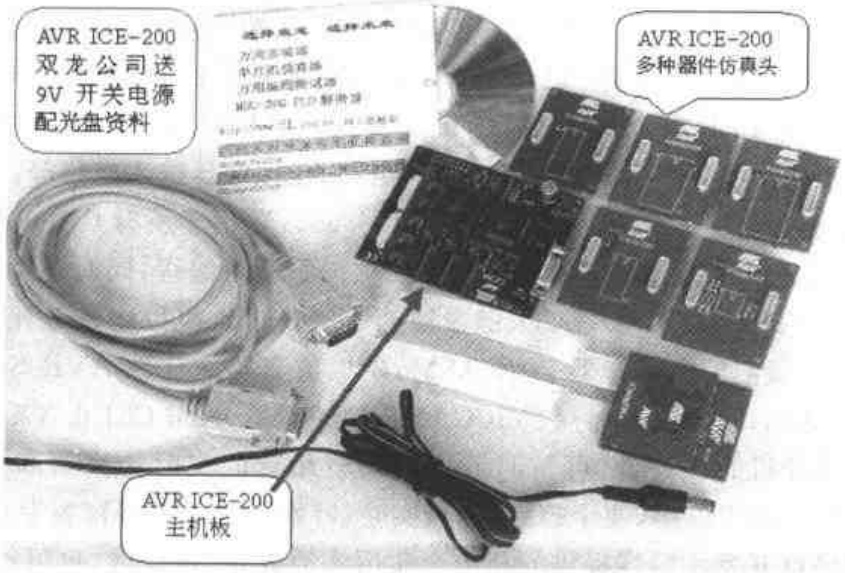


图 14.1　ICE－200

14.3 JTGA ICE 仿真器

AVR 高档单片机 ATmega 系列 ATmega16，ATmega32，ATmega323，ATmega64 及 ATmega128 等具有 JTGA 仿真接口，如图 14.2 所示，可以实现在线实时仿真。这在 8 位单片机中属于领先技术。JTGA 仿真，对不同封装器件可以先焊接，然后进行实时仿真、编程。与传统的芯片仿真头相比，可大大降低仿真器的价格。JTGA ICE 与 AVR 的连线如图 14.3 所示。

图 14.2 JTGA ICE

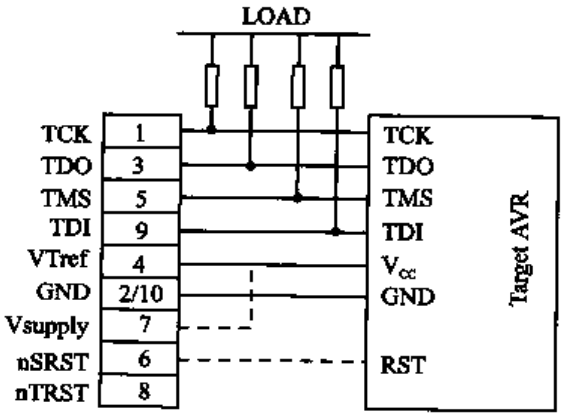


图 14.3 JTGA ICE 与 AVR 连线

14.4 开发下载实验器 SL－AVRAD

SL－AVRAD 为“四合一”增强型 AVR 嵌入式单片机开发下载实验器，是为配合本书的出版研制的新产品。“四合一”即 AVR 编程器＋模拟仿真器＋实验器＋科研样机，见图 14.4。SL－AVRAD 硬件采用模块化设计，便于灵活组成科研项目所需的硬件结构。硬件有 RS232 通信接口；串行下载监控；DIP8～DIP40 通用插座，DIP40 端口用短路块连接作输出，用 LED 发光二极管显示器件引脚高低电平，也可用短路块断开作输入或其他用途；有 6 位 LED 数码管作显示；2×16 点阵 LCD 液晶显示器；17 键的键盘；PC 机键盘接口插座；模拟比较输入电路；音响电路；单片机通用(AVR/MCS－51)复位电路；模拟电压输入电路等。随机附 120 mm×170 mm 万通实验板和一片带 AD 器件的 AT90S8535，适用于所有具有串行 ISP 下载编程功能的 AVR/AT89S 单片机。用户板上的器件无须拆下即可编程，同时还可做 AVR/AT89S 单片机的 I/O 口、A/D、D/A、LED、LCD、键盘输入、音频输出、模拟比较等开发实验。同时提供功能强大的 WIN 版汇编级编译器 AVRASM、模拟仿真调试软件 AVR Studio3. X 及串行下载软件 AVR PROG；也提供限时版的 ICCAVR C 编译器、Keil C51 6. XX 编译器。对初学 AVR/MCS－51 单片机的设计者，可暂时节省购买较昂贵的实时仿真器和万用编程器的费用。本机工作晶振可插拔更换，便于超频、降频实验，特别适合于仪器仪表专业、科技人员开发之用。SL－AVRAD 开发实验器提供的几十个实用实验程序，可修改，也可改变硬件接口，实现源程序的功能。这对大专院校学生发挥其创造性及动手能力的培养特别有用。本开发实验

器也可当科研样机使用。

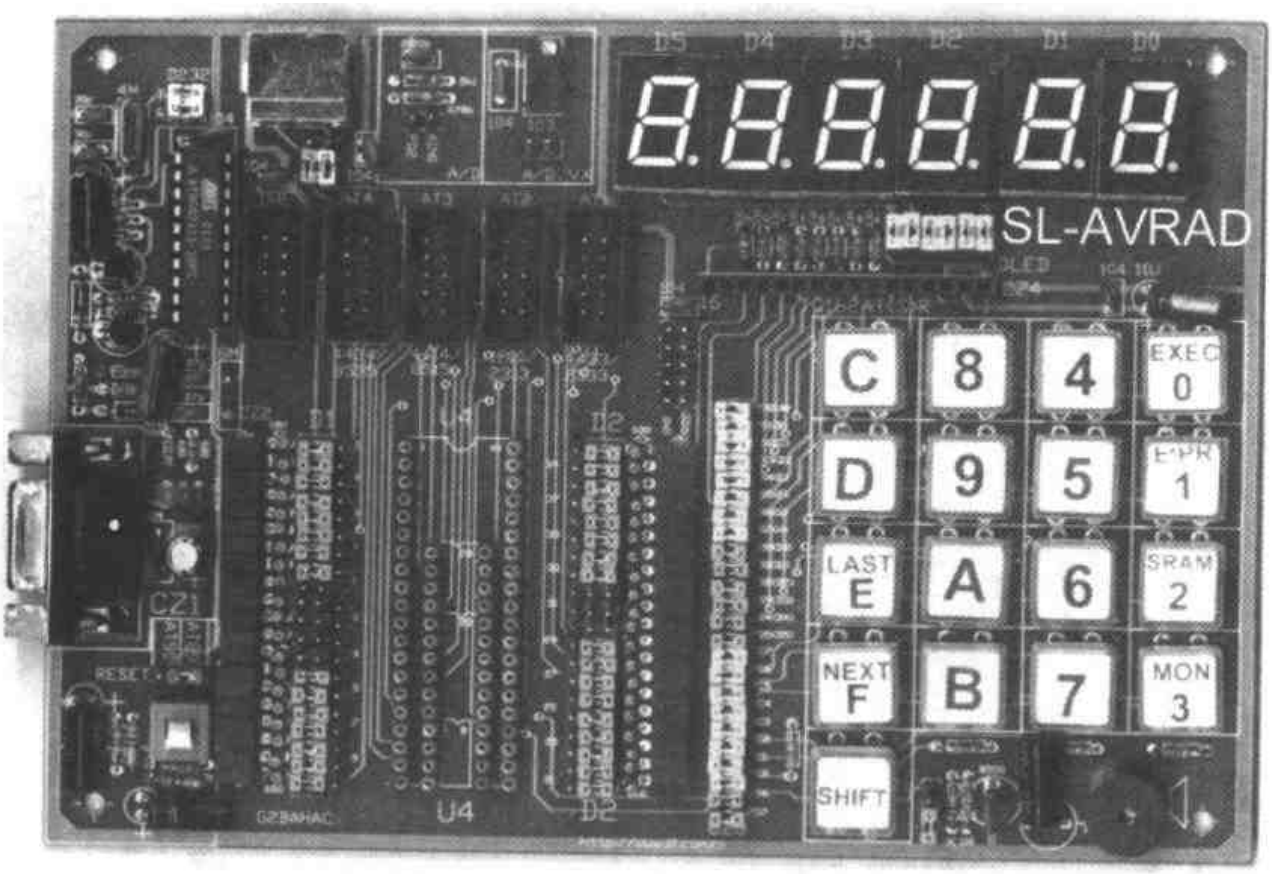


图 14.4　SL - AVRAD

SL - AVRAD 硬件接口电路见图 14.5，各器件说明如下。

① D232：D232 通信短路块。插上短路块，接到 AT90S2313；拔出短路块，可从 T，R 端用插针线接到单片机通信口。可作单片机异步通信 UART 或主从同步通信 SPI 或 ISP 下载通信，见图 14.6。

② AT90S2313：AVR 开发实验器监控芯片。AT90S2313 为增强型开发实验器通用单片机(AVR/AT89S)监控。

③ ISP：该列 DC10 的 (ISP)插座，即 AVR 单片机的下载信号插座，见图 14.7。

本开发实验器配一片 AT90S8535 器件，绝大多数实验使用该器件。硬件(用短路块)连接出厂时，也按该器件连接，其他器件作为选购件。

● ISP 下载插座：引脚功能分别为 V_{cc}，GND，XTAL2，XTAL1，MOSI，$\overline{\text{RESET}}$，SCK，MISO。随机附有一条 10 线信号线，由用户接插到对应 AVR 单片机(AT1～AT4)的信号脚上。ISP 也可接到用户板作 AVR 单片机(或用转接线)的串行下载编程用。如用户板有晶振，则 XTAL1/XTAL2 两信号线无须接出。

● AT1 插针座为 AT90S4433/AT90S2333 ISP 下载信号线座；AT2 插针座为 AT90S1200/AT90S2313 ISP 下载信号线座；AT3 插针座为 AT90S4414/AT90S8515 及 AT89S8252/AT89S53 ISP 下载信号线座；AT4 插针座为 AT90S4443/AT90S8535 ISP 下载信号线座。

如用于其他 AVR 单片机，用转接线接到其他 AVR 单片机下载信号引脚上。

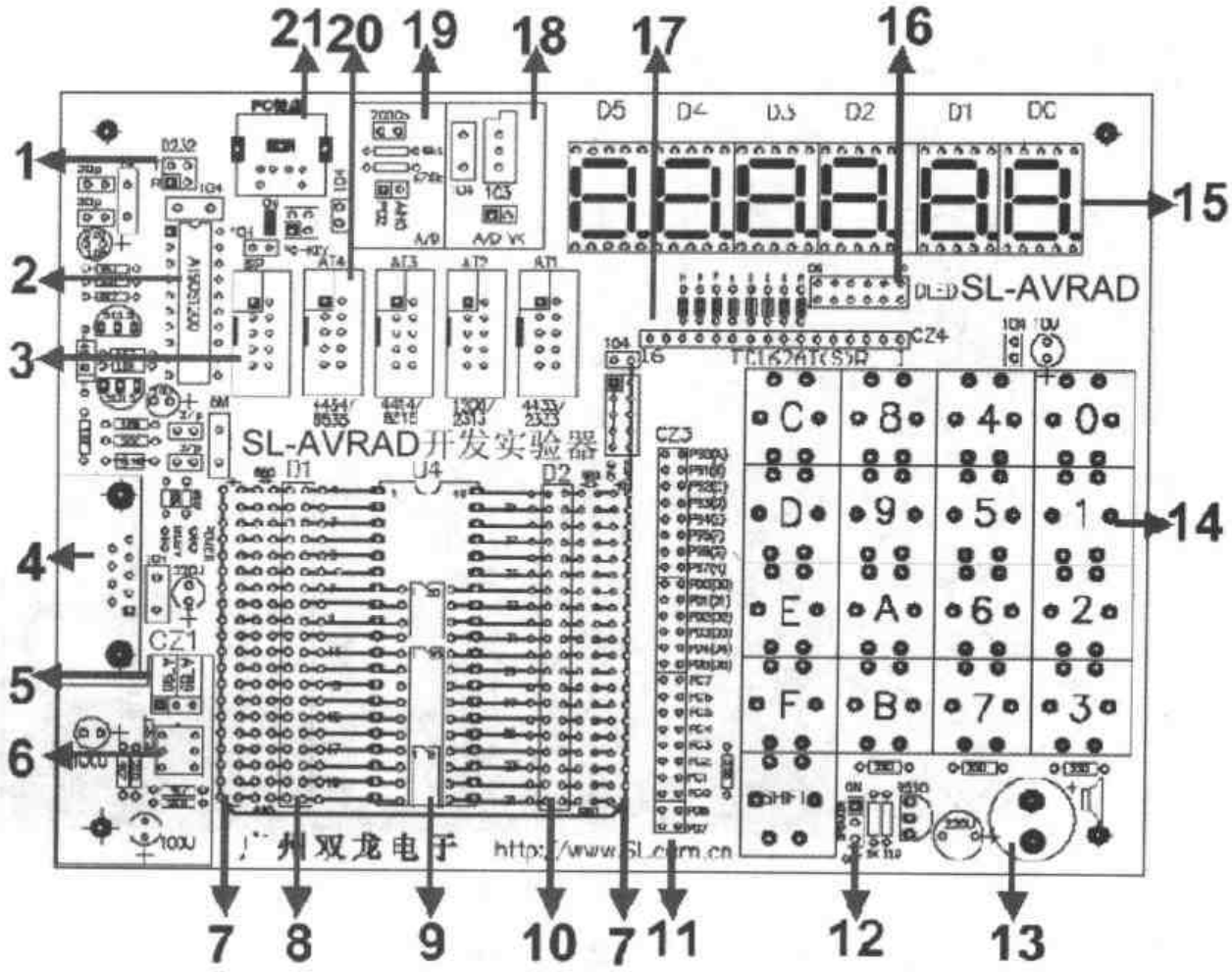


图 14.5 SL－AVRAD 硬件接口电路

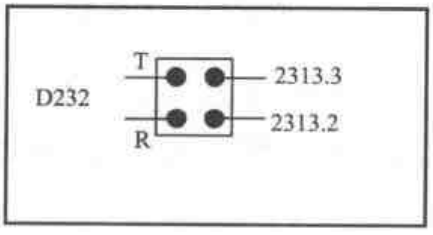


图 14.6 D232 通信短路块

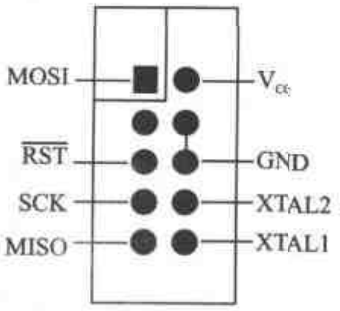


图 14.7 SL－AVRAD 开发实验器 ISP 信号座引脚功能

④ CZ1(双龙标准)：电源及通信下载插座。电源线为地和＋5 V，通信电缆一头接 CZ1，另一头接计算机 RS232 九针插座。

POWER：红色 LED，电源指示；BUSY：绿色 LED，下载通信工作忙指示。

⑤ 复位选择：用短路块选择 AVR 或 AT89S 单片机做开发下载实验。

⑥ 单片机 RST 复位按钮：作为程序下载后再启动复位用。一般程序下载结束或开机通

电时，程序自动执行。

⑦ LED 发光二极管：当 I/O 口输出指示时，低电平有效。

⑧ (8 与 10) D1，D2 短路块列：使单片机引脚作输出或输入用。插上短路块，引脚作输出，用 LED 显示高低电平，低电平 LED 灯亮；不插短路块，器件引脚可作输入或作其他用途，如电源引线等。

⑨ U4：作为 AVR 单片机 4 种 DIP(DIP8/20/28/40)封装器件下载插座。器件插放时缺口向上，向下插座底部对齐。如作生产用 ISP 编程，可用进口通用锁紧插座转接到 U4。

⑩ 与⑧同。

⑪ CZ3 接线排针：为 AT90S8535 对应引脚 PB0～PB7，PD0～PD5，PC0～PC7。插上短路块，接到键盘(17 键)和 LED/LCD 显示器；拔出短路块，用接插线可把键盘与显示改为其他 I/O 口，也可改用其他器件(如 AT90S2313/AT90S8515 等)接到器件相应引脚上(D1，D2 短路块处)。

本开发机提供的几十个实用实验程序，可改变硬件接口，也可修改程序，实现源程序的功能。

⑫ SPEAKER：音响输入端，接相应器件的右下脚(5/11/15/21)。插上短路块，接通 AT90S8535 的 PC0 的音响信号；也可用接插线接到单片机任何 I/O 脚上作音响输出。

⑬ 无源音响器：由单片机发出音响。

⑭ 17 键键盘模块：接 AT90S8535 的 PC 口。16 个数字键，1 个 SHIFT 换档键，接 AT90S8535 的 PD7。先按下 SHIFT 键，再按数字键，即实现数字键上的命令键功能。

当然根据程序的要求，这些数字的名称可以重新定义，不妨一试。

⑮ LED 模块：由 6 只数码管组成。D5～D2 作地址，D1～D0 作数据。数码管字位口对应接 AT90S8535 的 PD5～PD0 口，字形(abcdefgh)对应接 AT90S8535 的 PB7～PB0 口。

⑯ DLED 短路块：LED 位线，用 DLED 短路块连接。断开短路块，位线也可作其他用途。

⑰ CZ4 接线座：为 2 行×16 字 LCD 液晶显示模块插座，用 AT90S8535 的 PB，PD 口。

⑱ A/D VX：多圈电位器作为模拟信号输入。两边分别接 V_{cc}，GND，中间头(VX)用连接线接到单片机，作模拟信号输入脚(如 AIN1)。

⑲ A/D：片内模拟比较器作 A/D 转换外部元件电路(最好 R 精度为 1%，C 精度为 5%，所接阻值仅供参考)，AIN0，PD2 为接线端。

⑳ AT1～AT4：对应器件下载插座。

㉑ PC 机键盘插座：PD2 接小口键盘。

为便于用户能使用与自己工作样机相一致的晶振，提高工作可靠性，特别对晶振设置做了改进。在工作时，随机所附晶振接插小印板(附插 8 MHz 晶振)，应插在对应器件的晶振引脚上(对应器件旁插针)。晶振数值可根据实际工作改变(如超频、降频等)。

14.5　AVR 集成开发环境

ATMEL AVR Studio3.53 集成开发环境(IDE)，包括 AVR Assembler 编译器、AVR Studio 调试功能、AVR Prog 串行和 JTGA 下载功能、JTGA ICE 仿真功能等。AVR IDE 安装后，双击 AVR Studio 图标，则出现 AVR Studio3.53 集成开发环境窗口，如图 14.8 所示。

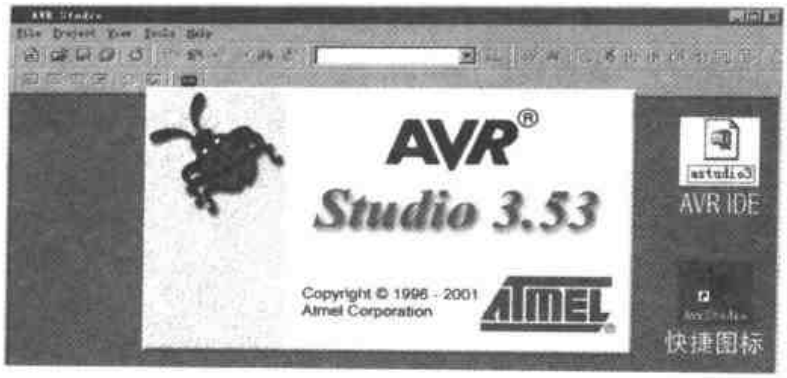


图 14.8 AVR 集成开发环境窗口

14.5.1 AVR Assembler 编译器

AVR Assembler 编译器可对有源文件进行编辑、汇编(生成. OBJ/. HEX/. LIS 文件)、搜寻、选项(生成汇编文件格式)、窗口、帮助等操作。汇编出错有错误定位、错误指示,便于源文件排错。

(1) 新建工程项目

① 选择 Project→New,出现 Select new project 窗口,新建工程项目;

② 必须输入工程项目名字和项目的类型,例:SL. APR 工程项目(. APR 也可缺省,则默认为. APR);

③ 选择存放工程项目路径;

④ 选中 AVR Assembler 汇编;

⑤ 单击 OK 按钮;

⑥ 完成自动新建工程项目,如图 14.9 所示。

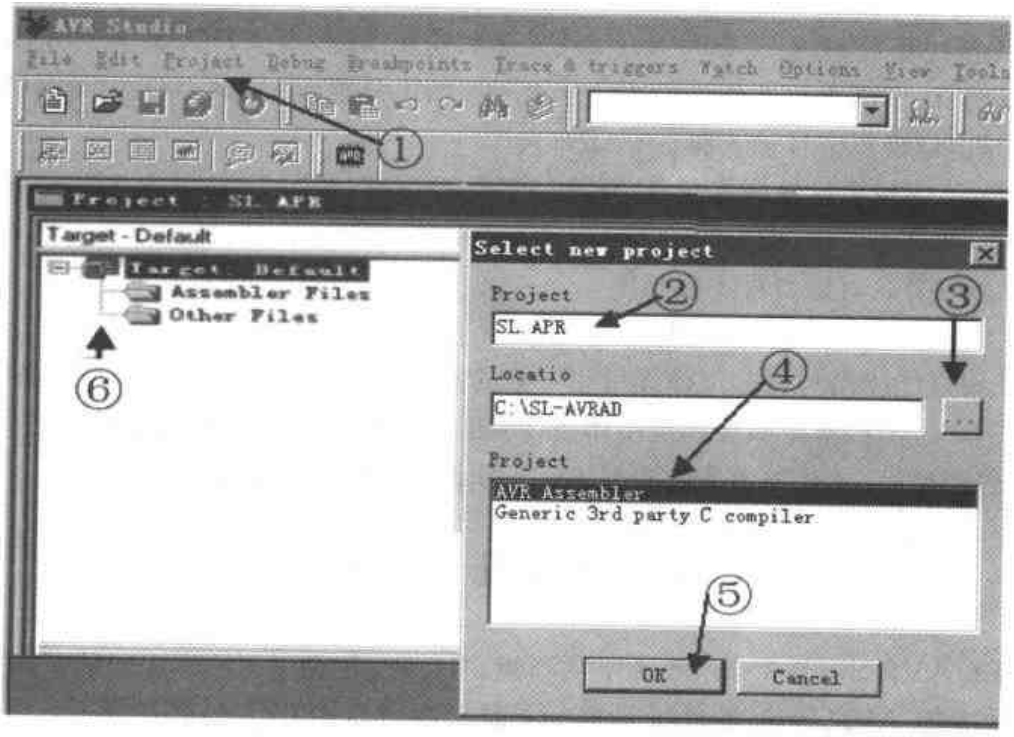


图 14.9 新建工程项目

(2) 打开已保存的工程项目

选择 Project→Open 路径，打开已存在的工程项目。

(3) 新建汇编文件名

① 选择 File→New text file，出现新建文件窗口 Create new file；

② 输入汇编文件名，例：SL. ASM；

③ 选择存放路径；

④ 单击 OK 按钮；

⑤ 源文件添加到工程项目中，并出现新的源文件编辑窗口，可编辑新的源程序，如图 14.10 所示。

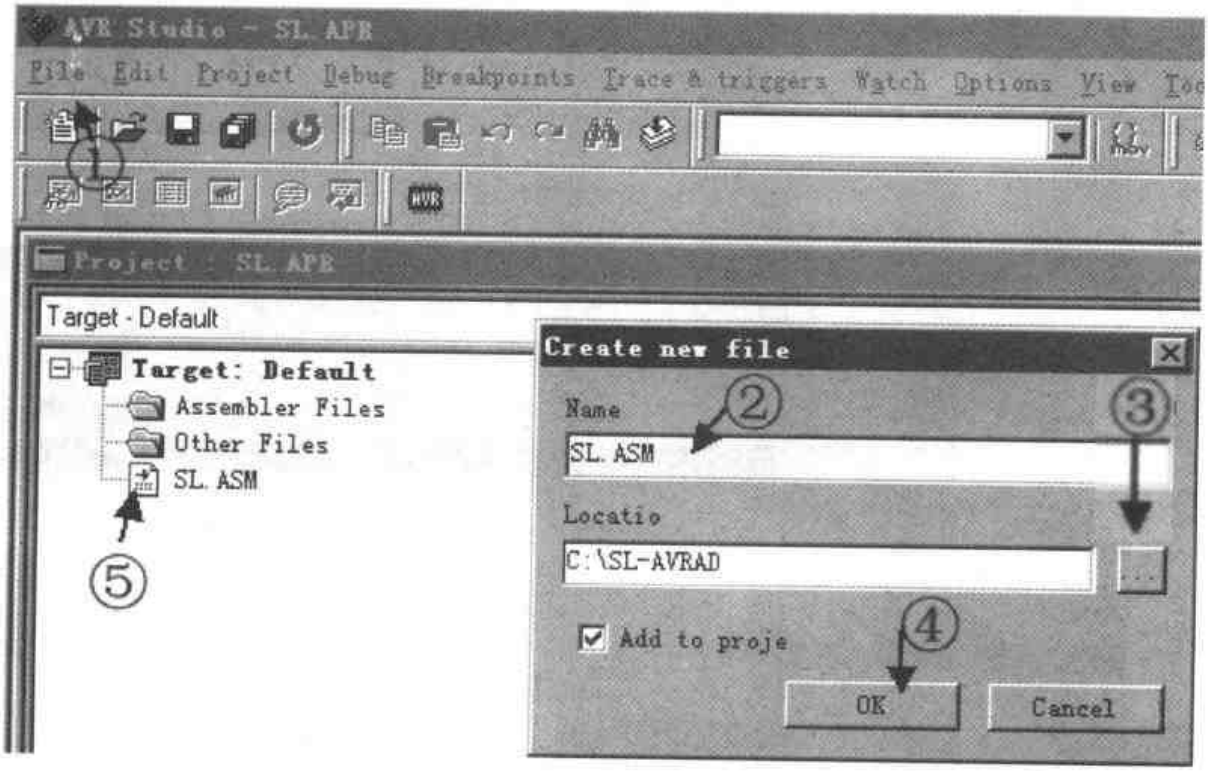


图 14.10　新建汇编文件名

(4) 打开已保存的汇编文件

选择 File→Open 路径，打开已存在的汇编文件。

(5) 源文件编译选项

① 选 Project 菜单；

② 选 Project Settings 编译项目设置，出现 AVR Assembler Options 选择窗口；

③ 选 Output file 中 Intel Intellec 8/MDS…，生成 Intel 格式 hex 文件；

④ 单击 OK 按钮，设置完成，如图 14.11 所示。

(6) 新文件的编辑与编译

① 窗口输入源文件 SL. asm；

② 选 Project 菜单；

③ 选 Project 下拉菜单的 Assemble 编译选项，进行编译；

④ 编译通过，显示编译提示，如图 14.12 所示。

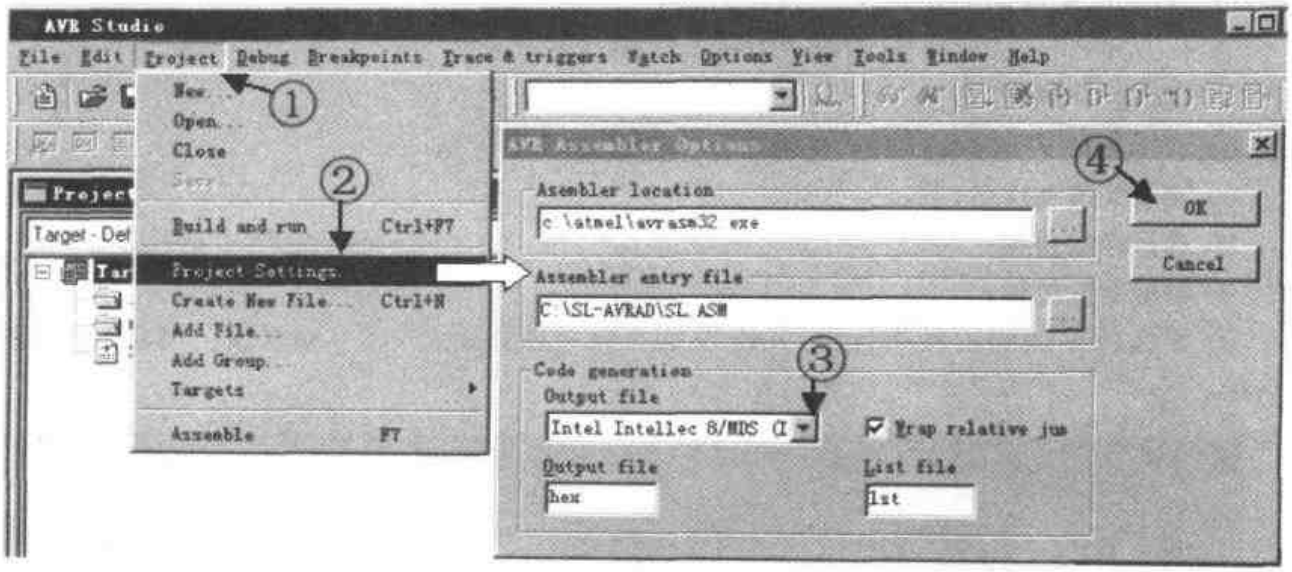


图 14.11 源文件编译选项

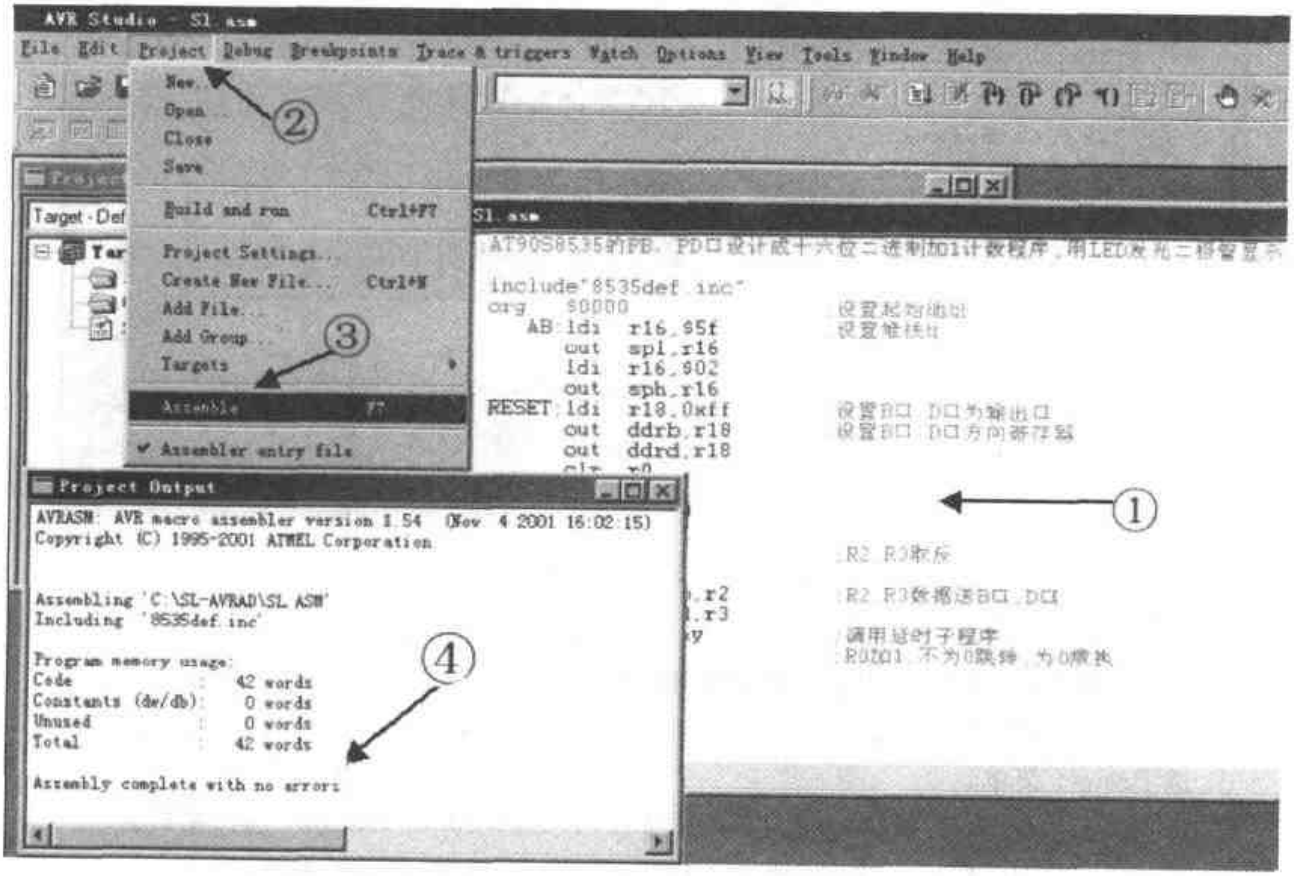


图 14.12 新文件的编辑与编译

14.5.2 AVR Studio

AVR Studio 可对源文件进行 DEBUG 调试(装入.OBJ 目标文件,以源文件格式显示调试;如装入.HEX 文件,以反汇编格式显示地址、机器码、指令等格式调试)排错、断点、单步、自

动单步、触发、注视、选项、查看、窗口、帮助等操作。调试中可打开多种窗口：I/O 窗口、源文件窗口、CPU 窗口、记录窗口、数据窗口等。

(1) 进入 Debug 调试选项窗口

选菜单 Debug→GO，进入 Debug 调试选项窗口

① 源文件调试窗口；

② 打开 Processor 观察窗口，可观察 CPU 的各种参数变化情况；

③ 打开 Standard 观察窗口，可观察 I/O 口电平变化情况；

④ Debug 调试快捷按钮；

⑤ 观察窗口快捷按钮。

根据源程序调试要求，还可打开更多的观察窗口，如图 14.13 所示。

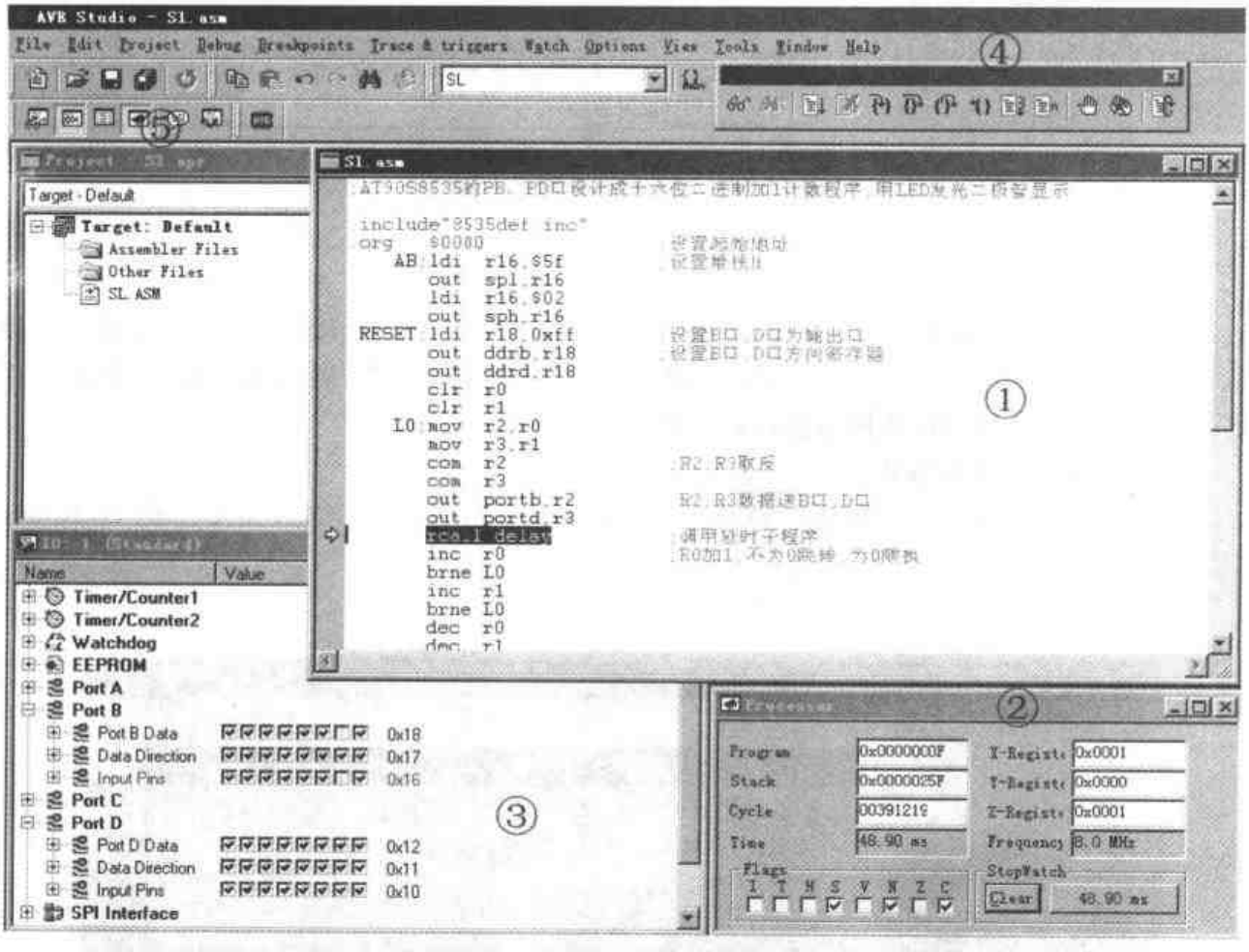


图 14.13　Debug 调试选项窗口

(2) 打开已有文件进入调试

① 选择菜单 File；

② 选择下拉菜单 Open；

③ 在一定路经下找到已有文件，文件类型可以是(*.obj；*.d90；*.cof；*.hex)，也可以是源文件、工程项目文件等，如图 14.14 所示。

④ 单击【打开】按钮。

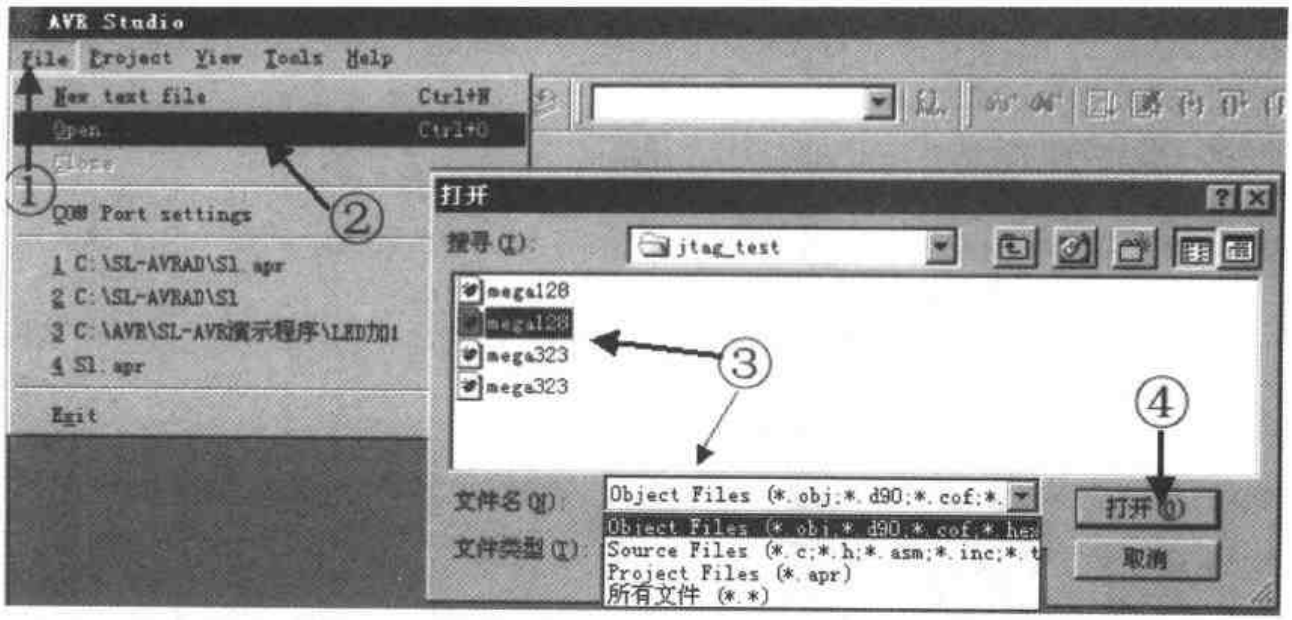


图 14.14 打开已有文件进入调试

14.5.3 AVR Prog

AVR Prog 为串行下载软件。当接通下载线时，一头接 PC 机 RS232 串行口，另一头接 SL-AVRAD 下载开发实验器，并接上 5 V 电源(红色线接+5 V，黑色线接地)。如连机正确，则出现下载提示窗口，按提示窗口要求操作。

(1) AVR ISP 串行编程

启动 AVR Studio3.53 程序。如果对芯片编程，可以启动 TOOLS 菜单中的 STK500/AVR ISP/JTAG ICE 下载编程，或单击 Alt+8 按钮。

如图 14.15 所示，根据窗口提示操作。

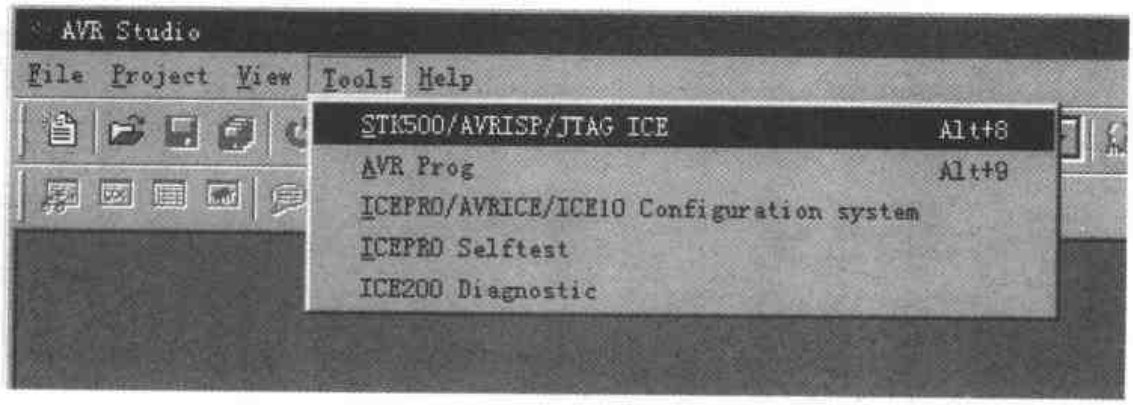


图 14.15 进入 AVR ISP

(2) AVR Prog 并行编程

见 12.1.4 小节“并行下载编程接口电缆”。

源文件说明

大量 AVR 应用实验源程序及编译、调试、下载工作软件可上网下载：www.sl.com.cn。

光盘中提供多种源文件：

① ATMEL 公司提供的见光盘文件 *：\AVR\AVR\asmpack\appnotes\AVR *. ASM（实验程序）；

② 双龙公司提供的见光盘文件 *：\AVR\AVR\SL-AVR\ *. ASM（实验应用程序，也适用 SL-AVR+）；

③ 双龙公司提供的见光盘文件 *：\AVR\AVR\指令 ASM\ *. ASM（指令实验调试程序）；

④ 双龙公司提供的见光盘文件 *：\AVR\AVR\SL-AVRAD\程汇\ *. ASM（实验程序）；

⑤ 双龙公司提供的见光盘文件 *：\AVR\AVR\SL AVRAD\ *. ASM（实验程序）；

⑥ 双龙公司提供的见光盘文件 *：\AVR\AVR\SL-AVR+\ *. ASM（实验程序）。

注意：光盘文件拷到计算机里是只读属性，只有去掉只读属性，才能进行修改、运行等操作。

第 15 章　AVR 单片机的最新发展

15.1　AVR 发展方向

ATMEL AVR 在制造工艺上有重大改进，以 ATMEL 0.35 的 4 层金属 CMOS 工艺制作，降低了器件成本。尤其高档 ATmega 系列器件，具有较高的性能价格比。

AVR 单片机与 FPGA，SRAM 结合，形成性能更强的高档产品 FPSLIC，即 AT94K 系列现场可编程系统标准集成电路，进入芯片级系统开发。

15.1.1　ATmega 系列特点

ATmega 系列按 Flash 容量(KB)分类，有 ATmega8，ATmega16，ATmega32，ATmega64，ATmega128，ATmega256(暂未推向市场)，以满足不同容量的应用，使单片机真正做到只用 1 片。

ATmega 系列增加了只需 2 个时钟周期的硬件乘法器；通过芯片上的 BOOT 区实现程序在线自编程，真正的读和写同时操作；具有上电复位和可编程的电源检测；具有 2 线 I^2C 接口；具有 8 路 10 位 A/D 转换；具有独立振荡器的实时时钟 RTC；绝大多数器件增加了 JTAG 仿真接口，支持片内在线调试，通过 JTAG 可编程 Flash，EEPROM，熔丝及加密位。

15.1.2　ATmega8/ATmega8L

特　点

(1) 高性能、低功耗 AVR 8 位单片机。

(2) 高性能 RISC 结构：

——130 条指令，大多数为单指令周期；

——32 个 8 位通用(工作)寄存器；

——完全的静态操作；

——最高工作在 16 MHz 时，具有 16 MIPS 的性能；

——只需 2 个时钟的硬件乘法器。

(3) 数据和非易失性程序内存：

——8 KB 的在线可编程 Flash(擦除次数：1 000 次)；

——具有独立加密位的 BOOT 代码区，通过芯片上的 BOOT 区程序实现在线自编程，真正的读和写同时操作；

——512 B 在线可编程 EEPROM(擦写寿命：100 000 次)；

——1 KB SRAM 存储器；

——为软件安全设计的加密位。

(4) 外围(peripheral)特点：

——2 个具有比较模式的可预分频(prescale)8 位定时器/计数器；

——1 个可预分频，具有比较、捕捉功能的 16 位定时器/计数器；

——具有独立振荡器的实时时钟 RTC；

——2 个 8 位 PWM 通道；

——1～16 位可编程 6 个 PWM 通道；

——8 路 10 位 A/D 转换，8 个单通道，7 个差分通道，2 个可编程 1x，10x 或 200x 差分通道；

——2 线 I^2C 接口；

——1 个可编程的 UART；

——主/从 SPI 接口；

——可编程的看门狗定时器(由片内振荡器生成)；

——片内模拟比较器。

(5) 特别的 MCU 特点：

——上电复位和可编程的电源检测；

——可校准的内部 RC 振荡器；

——内外部中断源；

——5 种休眠模式：空闲、ADC 噪声抑制、省电模式、掉电模式及后备模式；

——可通过软件选择时钟频率。

(6) I/O 和封装：

——23 个可编程的 I/O 口；

——28 脚 PDIP，32 脚 TQFP 及 32－pad MLF。

(7) 工作电压：

——2.7～5.5 V(ATmega8L)；

——4.5～5.5 V(ATmega8)。

(8) 速　度：

——0～8 MHz(ATmega8L)；

——0～16 MHz(ATmega8)。

(9) 上电复位检测：

——工作方式，TBD；

——空闲方式，TBD；

——掉电方式，TBD。

15.1.3　ATmega16/ATmega16L

特　点

(1) 高性能、低功耗 AVR 8 位单片机。

(2) 高性能 RISC 结构：

——130 条指令，大多数为单指令周期；

——32 个 8 位通用(工作)寄存器；

——完全的静态操作；

——最高工作在 16 MHz 时，具有 16 MIPS 的性能；

——只需 2 个时钟的硬件乘法器。

(3) 数据和非易失性程序内存：

——16 KB 的在线可编程 Flash(擦/写次数：1 000 次)；

——具有独立加密位的 BOOT 代码区，通过芯片上的 BOOT 区实现程序在线自编程，真正的读和写同时操作；

——512 B 在线可编程 EEPROM(擦/写寿命：100 000 次)；

——1 KB SRAM 存储器；

——为软件安全设计的加密位。

(4) JTAG (符合 IEEE std. 1149.1 标准)接口：

——边界扫描功能符合 JTAG 标准；

——支持片内在线调试；

——通过 JTAG 可编程 Flash，EEPROM，熔丝及加密位。

(5) 外围(peripheral)特点：

——2 个具有比较模式的可预分频(prescale)8 位定时器/计数器；

——1 个可预分频，具有比较、捕捉功能的 16 位定时器/计数器；

——具有独立振荡器的实时时钟 RTC；

——4 路 PWM；

——8 路 10 位 A/D 转换，8 个单通道，7 个差分通道，2 个可编程 1x，10x 或 200x 差分通道；

——2 线 I^2C 接口；

——1 个可编程的 UART；

——主/从 SPI 接口；

——可编程的看门狗定时器(由片内振荡器生成)；

——片内模拟比较器。

(6) 特别的 MCU 特点：

——上电复位和可编程的电源检测；

——可校准的内部 RC 振荡器；

——内外部中断源；

——6 种休眠模式：空闲、ADC 噪声抑制、省电模式、掉电模式、后备模式及扩展的后备模式。

(7) I/O 和封装：

——32 个可编程的 I/O 口；

——40 脚 PDIP 和 44 脚 TQFP。

(8) 工作电压：

——2.7～5.5 V(ATmega16L)；

——4.5～5.5 V(ATmega16)。

(9) 速　度：

——0～8 MHz(ATmega16L)；

——0～16 MHz(ATmega16)。

15.1.4　ATmega323/ATmega323L(兼容 ATmega32/L)

特　点

(1) 高性能、低功耗 AVR 8 位单片机。

(2) 高性能 RISC 结构：

——130 条指令，大多数为单指令周期；

——32 个 8 位通用(工作)寄存器；

——完全的静态操作；

——最高工作在 8 MHz 时，具有 8 MIPS 的性能；

——只需 2 个时钟的硬件乘法器。

(3) 数据和非易失性程序内存：

——32 KB 的在线可编程 Flash(擦/写次数：1 000 次)；

——具有独立加密位的 BOOT 代码区，通过芯片上的 BOOT 区实现程序在线自编程；

——1 KB 在线可编程 EEPROM(擦/写寿命：100 000 次)；

——2 KB SRAM 存储器；

——为软件安全设计的加密位。

(4) JTAG (符合 IEEE std. 1149.1 标准)接口：

——支持片内在线调试；

——通过 JTAG 可编程 Flash，EEPROM，熔丝及加密位；

——边界扫描功能符合 JTAG 标准。

(5) 外围(peripheral)特点：

——2 个具有比较模式的可预分频(prescale)8 位定时器/计数器；

——1 个可预分频，具有比较、捕捉功能的 16 位定时器/计数器；

——具有独立振荡器的实时时钟 RTC；

——4 路 PWM；

——8 路 10 位 A/D 转换；

——2 线 I^2C 接口；

——1 个可编程的 UART；

——主/从 SPI 接口；

——可编程的看门狗定时器(由片内振荡器生成)；

——片内模拟比较器。

(6) 特别的 MCU 特点：

——上电复位和可编程的电源检测；

——可校准的内部 RC 振荡器；

——内外部中断源；

——6 种休眠模式：空闲、ADC 噪声抑制、省电模式、掉电模式、后备模式和扩展的后备

模式。

(7) I/O 和封装：

——32 个可编程的 I/O 口；

——40 脚 PDIP 和 44 脚 TQFP。

(8) 工作电压：

——2.7～5.5 V(ATmega323L)；

——4.0～5.5 V(ATmega323)。

(9) 速　度：

——0～4 MHz(ATmega323L)；

——0～8 MHz(ATmega323)。

15.1.5　ATmega64/ATmega64L

特　点

(1) 高性能、低功耗 AVR 8 位单片机。

(2) 高性能 RISC 结构：

——130 条指令，大多数为单指令周期；

——32 个 8 位通用(工作)寄存器；

——完全的静态操作；

——最高工作在 16 MHz 时，具有 16 MIPS 的性能；

——只需 2 个时钟的硬件乘法器。

(3) 数据和非易失性程序内存：

——64 KB 的在线可编程 Flash(擦/写次数：1 000 次)；

——具有独立加密位的 BOOT 代码区，通过芯片上的 BOOT 区实现程序在线自编程，真正的读和写同时操作；

——2 KB 在线可编程 EEPROM(擦/写寿命：100 000 次)；

——4 KB SRAM 存储器；

——SPI 接口，同时可用作在线下载。

(4) JTAG (符合 IEEE std. 1149.1 标准)接口：

——边界扫描功能符合 JTAG 标准；

——支持片内在线调试；

——通过 JTAG 可编程 Flash，EEPROM，熔丝及加密位。

(5) 外围(peripheral)特点：

——2 个具有比较模式的可预分频(prescale)8 位定时器/计数器；

——2 个可预分频，具有比较、捕捉功能的 16 位定时器/计数器；

——具有独立振荡器的实时时钟 RTC；

——2 个 8 位 PWM 通道；

——1～16 位可编程 6 个 PWM 通道；

——8 路 10 位 A/D 转换，8 个单通道，7 个差分通道，2 个可编程 1x，10x 或 200x 差分通道；

——2 线 I^2C 接口；

2 个可编程的 UART；

——主/从 SPI 接口；

——可编程的看门狗定时器(由片内振荡器生成)；

——片内模拟比较器。

(6) 特别的 MCU 特点：

——上电复位和可编程的电源检测；

——可校准的内部 RC 振荡器；

——内外部中断源；

——6 种休眠模式：空闲、ADC 噪声抑制、省电模式、掉电模式、后备模式及扩展的后备模式；

——可通过软件选择时钟频率；

——兼容 ATmega103；

——全局上拉电阻断、通。

(7) I/O 和封装：

——53 个可编程的 I/O 口；

——64 脚 TQFP 封装。

(8) 工作电压：

——2.7～5.5 V(ATmega64L)；

——4.5～5.5 V(ATmega64)。

(9) 速　度：

——0～8 MHz(ATmega64L)；

——0～16 MHz(ATmega64)。

15.1.6　ATmega128/ATmega128L

特　点

(1) 高性能、低功耗 AVR 8 位单片机。

(2) 高性能 RISC 结构：

——133 条指令，大多数为单指令周期；

——32 个 8 位通用(工作)寄存器；

——完全的静态操作；

——最高工作在 16 MHz 时，具有 16 MIPS 的性能；

——只需 2 个时钟的硬件乘法器。

(3) 数据和非易失性程序内存：

——128 KB 的在线可编程 Flash(擦/写次数：1 000 次)；

——具有独立加密位的 BOOT 代码区，通过芯片上的 BOOT 区实现程序在线自编程，真正的读和写同时操作；

——4 KB 在线可编程 EEPROM(擦/写寿命：100 000 次)；

——4 KB SRAM 存储器；

——最大 64 KB 外部寻址空间；

——为软件安全设计的加密位；

——SPI 接口，同时可用作在线下载。

(4) JTAG (符合 IEEE std. 1149.1 标准)接口：

——边界扫描功能符合 JTAG 标准；

——支持片内在线调试；

——通过 JTAG 可编程 Flash，EEPROM，熔丝及加密位。

(5) 外围(peripheral)特点：

——2 个具有比较模式的可预分频(prescale)8 位定时器/计数器；

——2 个可预分频，具有比较、捕捉功能的 16 位定时器/计数器；

——具有独立振荡器的实时时钟 RTC；

——2 个 8 位 PWM 通道；

——1～16 位可编程 6 个 PWM 通道；

——8 路 10 位 ADC，8 个单通道，7 个差分通道，2 个可编程 1x，10x 或 200x 的差分通道；

——2 线 I^2C 接口；

——2 个可编程的 UART；

——主/从 SPI 接口；

——可编程的看门狗定时器(由片内振荡器生成)；

——片内模拟比较器。

(6) 特别的 MCU 特点：

——上电复位和可编程的电源检测；

——可校准的内部 RC 振荡器；

——内外部中断源；

——6 种休眠模式：空闲、ADC 噪声抑制、省电模式、掉电模式、后备模式及扩展的后备模式；

——可通过软件选择时钟频率；

——兼容 ATmega103；

——全局上拉电阻断、通。

(7) I/O 和封装：

——53 个可编程的 I/O 口；

——64 脚 TQFP 封装。

(8) 工作电压：

——2.7～5.5 V(ATmega128L)；

——4.5～5.5 V(ATmega128)。

(9) 速　度：

——0～8 MHz(ATmega128L)；

——0～16 MHz(ATmega128)。

15.2　AT94K 系列现场可编程系统标准集成电路

AT94K 系列(FPSLIC family) 整合了 ATMEL AT40K 系列 SRAM FPGA 和高性能的、带标准外设的 ATMEL AVR 8 位 RISC 微控制器。此器件中包含了扩展数据和指令 SRAM 及器件控制和管理逻辑，以 ATMEL 0.35 的 4 层金属 CMOS 工艺制作。10～40 K 门的 AT40K FPGA 带 8 位微控制器和 36 KB 的 SRAM。AT40K FPGA 核心是一个完全符合 3.3 V PCI 标准、带 10 ns 分布式同步/异步可编程的全双工口/单工口的 SRAM；是具有 8 个全局时钟、Cache Logic 性能(部分或全部可重新设置而不丢失数据) 及 10～40 K 的可用门数的、基于 SRAM 的 FPGA 。

AT94K 系列单片机的特点

AT94K 系列单片机是大规模现场可编程系统标准集成电路。

AT40K 基于 SRAM 的 FPGA，具有嵌入式高性能的 RISC AVR 核心及扩展的数据和指令的 SRAM。

特　点

(1) 10～40 K 门基于专利 SRAM 的 AT40K FPGA 带 FreeRAM：

—— 4.6～18.4 K 位的分布式单/双口 FPGA 的用户 SRAM；

—— 高性能 DSP 优化的 FPGA 核心单元；

——内置动态可重新编程；

——可存取设置 FPGA。

(2) AVR 微控制器核心片内支持 Cache Logic 设计：

——极低的静态和动态功耗；

——最适于轻便及手持式的应用。

(3) 专利 AVR 扩展 RISC 结构：

——120 条功能强大的指令；

——绝大多数执行周期为单时钟周期；

—— 基于 DSP 系统的高性能硬件乘法器；

——可用超过 30 MIPS Performance；

——带 32 个内部寄存器的“C” 代码优化结构；

——低电压休眠、省电及掉电模式。

(4) 32 KB 动态分配指令和数据 SRAM：

——最多 16 K×16 内部 15 ns 指令 SRAM；

——最多 14 K×8 内部 15 ns 数据 SRAM；

(5) AVR Fixed 外设：

——工业标准的 2 线 I^2C 接口；

——2 个可编程串行 UART；

——2 个带分立预定比例器和 PWM 的 8 位定时器/计数器，1 个带分立预定比例器、比较、捕获模式及 8 位、9 位或 10 位 PWM 的 16 位定时器/计数器 。

(6) 支持 FPGA 标准的外设：

- - -AVR 外设控制 ；

-----16 解码 AVR 地址线可直接存取 FPGA；

——标准外设的 FPGA 宏功能库。16FPGA 给 AVR 提供内部中断，最多给 AVR 4 个外部中断。

(7) 8 个全局 FPGA 时钟：

-— 2 个从 AVR 逻辑驱动的 FPGA 时钟；

-----可从 FPGA 核心存取 FPGA 全局时钟。

(8) 复合振荡器电路：

-----带片内振荡器的可编程看门狗定时器；

——AVR 内部时钟电路振荡器；

-- 可软件选择时钟频率；

——定时器/计数器实时时钟振荡器。

(9) V_{CC}：3.0～3.6 V。

(10) 3.3 V，33 MHz PCI 标准的 FPGA I/O：

-24 mA 灌电流，高性能 I/O 结构；

——所有 FPGA I/O 单独可编程，引脚与 ATMEL AT40K 系列 FPGA 兼容。

(11) 高性能，低电压 0.35 CMOS 4 层金属处理。

(12) State-of-the-art 基于 PC 的包含协检验的集成软件。

表 15.1 所列为 AT94K 系列集成电路。

表 15.1 AT94K 系列

器 件	AT94K10	AT94K20	AT94K40
FPGA 门数/K	10	20	40
FPGA 核心单元	576	1 024	2 304
FPGA SRAM 位数	4 096	8 192	18 432
FPGA 寄存器数(全部)	864	1 408	2 880
最多 FPGA 用户 I/O	144	192	288
可编程 SRAM/KB	20～32	20～32	20～32
数据 SRAM/KB	4～16	4～16	4～16
硬件类乘法器(8 位)	有	有	有
2 线串行接口	有	有	有
UART	2	2	2
看门狗定时器	有	有	有
定时器/计数器	3	3	3
实时时钟	有	有	有
典型的 AVR 吞吐量@40 MHz/MIPS	30	30	30
工作电压/V	3.0～3.6	3.0～3.6	3.0～3.6

图 15.1 所示为 AT94K 系列的结构。

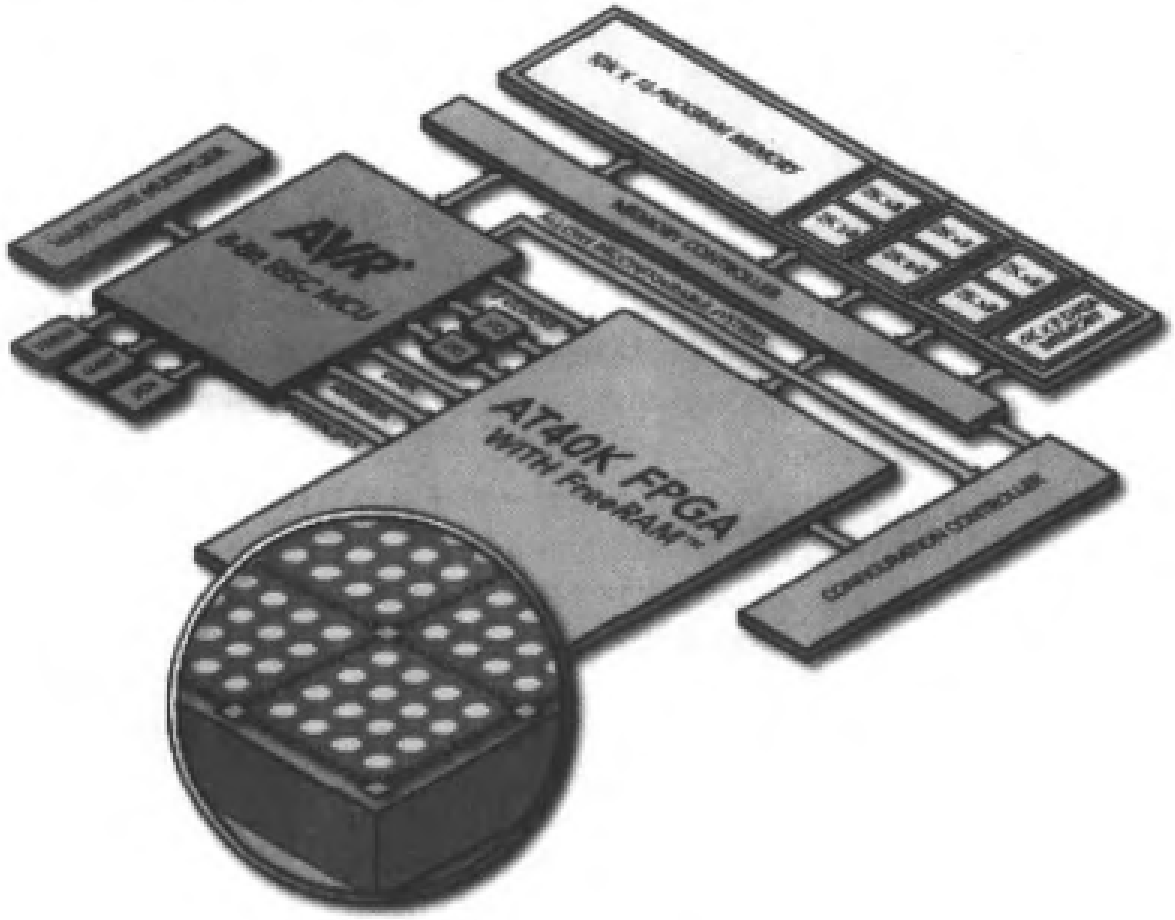


图 15.1　AT94K 系列的结构

第 16 章　整机设计中几个问题的处理方法

16.1　AVR 单片机的外围扩展

AVR 单片机有低、中、高档几十种型号，引脚数目 8～64 脚，价格从几元到近百元。在应用设计时，应选择能满足系统要求的、价格最低的型号，以降低产品成本。

若单个单片机满足不了系统的要求，可以进行外部扩展，有以下几种扩展方案。

1. 通过系统并行总线扩展

AVR 系列单片机中，AT90S8535 等多数单片机无并行总线接口，只有 AT90S4434/8515，ATmega603/103，Atmega161 等少数型号有并行总线接口。与 MCS－51 系列单片机相似，可以扩展外部数据存储器，如 RAM62256，EEPROM2864 等。

8515 扩展 RAM62256 的电路如图 16.1 所示。程序上把 MCUCR 中的 SRE 位置 1 后，则 8515 的 A 口变为 AD0～AD7(低 8 位地址数据总线)；C 口变为 A8～A15(高 8 位地址数据总线)；PD6，PD7 也变成$\overline{WR}$和$\overline{RD}$引脚。用 LDS Rd，k 指令、STS k，Rr 指令直接寻址，或者用 LD Rd，X 指令、ST X，Rr 等指令间接寻址，读写外部数据存储器。

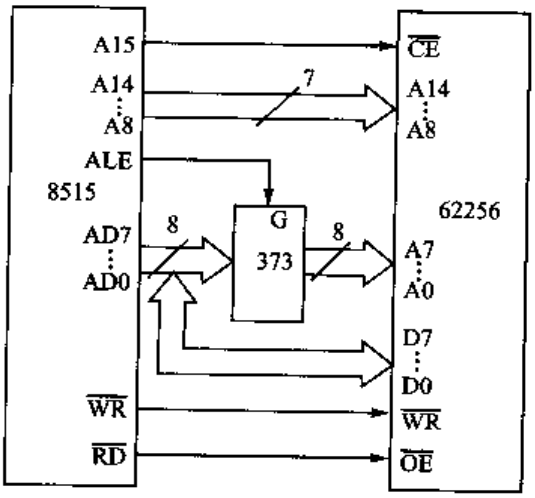


图 16.1　AT90S8515 外部扩展 62256 电路图

通过并行总线扩展外存，8515 的 AD0～AD7，A8～A15，$\overline{WR}$和$\overline{RD}$占用了 18 个口，不能再用作输入输出。另外，同样晶振频率的情况下，8515 要比 MCS－51 系列的 8031 读写外部数据存储器快 12 倍，一般外部数据存储器达不到这么高的速度；所以要想与外部数据存储器读写匹配，应大大降低单片机晶振频率。这样会降低 8515 的运行速度，有时也是不允许的。

外部程序存储器不能扩展，若程序存储器容量不够，只能选更高档次的单片机，例如 ATmega128 有 128 KB 的 Flash 存储器。

有些 A/D 转换器、D/A 转换器及 I/O 扩展器等可以接在数据总线上，按读写外部数据存

储器的方法访问。如 AD574 等带输出三态缓冲器的 ADC 芯片；DAC0832 带有数据寄存器的 DAC 芯片；带并行总线接口的 8255，8155 及 8279 等 I/O 接口芯片。

由于用并行总线扩展，占用了单片机的 AD0～AD7，A8～A15，$\overline{WR}$和$\overline{RD}$ 18 个 I/O 口，又要降低晶振频率，才能与扩展芯片匹配，建议不用此法扩展。

2. 通过 I/O 口扩展 A/D 转换器和 D/A 转换器

如 4 位半 BCD 码双积分 A/D 转换器 ICL7135 不带输出三态缓冲器，直接分时输出 5 位 BCD 码，单片机需 12 根输入线与其接口。

12 位二进制逐次逼近式 A/D 转换器 AD578 也不带输出三态缓冲器，单片机需 13 根输入线、1 根输出线（启动 ADC）与其接口。

12 位 D/A 转换芯片 AD7531，DAC1220 等片内无数据寄存器，有 12 个输入端，单片机只要 12 个输出口与其接口即可。

通过 I/O 口扩展 A/D 转换器和 D/A 转换器，接口电路简单，但占用 I/O 线较多。

3. 通过 I^2C 总线

I^2C 总线是飞利浦（PHILIPS）公司开发的一种简单、双向 2 线制同步串行总线，直译为集成电路内部总线（intd-integrated circuit bus），简称为 I^2C 总线，ATMEL 称 2 总线。

I^2C 总线的主要特性是：只需 2 根线，串行数据总线 SDI 和串行时钟总线 SCL；每个连到总线上的器件都可由软件以唯一的地址寻址；双向数据传送可达 100 kbps；很多集成电路可连接到同一总线上，集成电路数主要受 400 pF 的电容限制。典型的 I^2C 总线结构如图 16.2 所示。

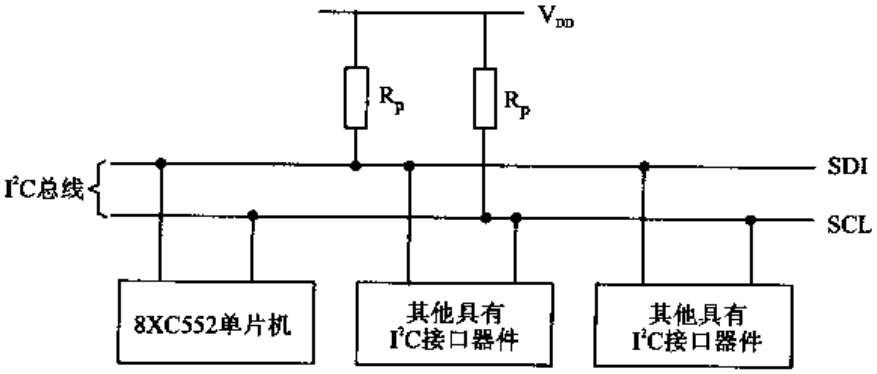


图 16.2　典型的 I^2C 总线结构

目前拥有 I^2C 总线的接口有 8XC550，8XC552，8XC652，8XC654，8XC751，8XC752，8XCL410 等单片微机，以及上百种接口芯片，包括 LED 驱动器，LCD 驱动器，A/D、D/A 转换器，RAM，EEPROM 等。其他集成电路厂家也有这种总线。

AVR 系列单片机中 ATmega8/32/128 等型号单片机有 I^2C 总线接口。其他无 I^2C 总线接口的单片机也可用 2 条 I/O 线通过软件编程实现 I^2C 总线接口。原来熟悉 I^2C 总线接口的设计者，可用此总线扩展外围接口或存储器。详细资料请参考飞利浦公司有关资料。

4. 通过 SPI 同步串行口外部扩展

若 AVR 单片机的 I/O 口或数据存储器不够用，可通过 SPI 同步串行口外部扩展成另一个 AVR 单片机。其中一个作为主机，另一个作从机，完成显示、键盘或打印等辅助工作。扩展方法详见 11.4 所述例子。

近些年新推出的 RAM，EEPROM，ADC，DAC 等芯片大多带有标准 SPI 接口。这些芯片

都直接通过2根线与AVR单片机SPI总线接口。

SPI同步接口硬件连接及编程简单，但一个SPI同步接口只能外扩展一个器件。如果AVR单片机总线接口不够用，也可用2条I/O线通过软件实现另一个SPI总线。

外围接口芯片还有3线制同步串行接口或某些特殊功能串行口。根据这些接口特性，都可以通过几条I/O线，用软件实现相应的串行接口。

SL279就是用SPI设计成的键盘显示接口芯片。见双龙工作光盘内资料。

16.2 低功耗设计

16.2.1 低功耗设计方法概述

单片机低功耗系统广泛用于军事、航天、气象、地质及手持式仪表等不方便用交流市电作电源，而用蓄电池供电的领域。单片机低功耗设计方案主要从以下几个方面考虑：

(1) 采用CMOS低功耗的单片机。像AVR单片机就属于这一类。

(2) 尽量降低单片机的供电电源。如AT90LS8535，供电电压在2.7 V就能正常工作；Attiny12，甚至供电电压低到1.8 V也能正常工作。随着供电电压降低，供电电流大幅减少。AT90LS8535主频4 MHz、供电电压5 V时，供电电流需11.5 mA；而供电电压3 V时，供电电流只需6.4 mA。

(3) 降低晶体振荡器的频率。一般来说主频下降一半，功耗也下降接近一半。AT90S8535在3 V供电的情况下，主频8 MHz需9 mA；主频4 MHz需5 mA；主频2 MHz只需3 mA。

(4) 采用低功耗的外围电路。如显示器LED改用液晶的，所需电流只有μA量级，可显著降低功耗。

(5) 单片机使用休眠模式。适当的休眠模式既能保证系统功能，又可大大降低功耗。

16.2.2 AT90S8535单片机的休眠状态

单片机进入休眠状态，停止正常程序执行，以减少功耗。休眠状态有以下3种模式。

1. 闲置模式

此模式下CPU停止运行；而SPI、UART、模拟比较器、ADC、定时器/计数器、看门狗及中断系统继续工作。内外部中断都可以唤醒MCU，如果不需要从模拟比较器中断唤醒MCU，为了减少功耗，可以切断比较器的电源。方法是置位I/O寄存器ACSR中的ACD位。如果MCU从闲置模式唤醒，CPU将立即执行指令。AT90LS8535单片机主频4 MHz、3 V供电时，闲置模式下只需1.9 mA电流。

2. 掉电模式

在此模式下外部晶振停振，而外部中断及看门狗在使能的前提下继续工作。只有外部复位、看门狗复位及外部电平中断INT0和INT1可以使MCU脱离掉电模式。可在掉电模式下选择看门狗定时器是否触发。若看门狗定时器为触发，当看门狗定时器溢出时，将唤醒MCU；若看门狗定时器为非触发，则只有外部复位或外部电平触发中断可唤醒MCU。用外部电平中断方式将MCU从掉电模式唤醒时，必须保持外部电平一定的时间，这样可以减少

MCU 对噪声的敏感。

从施加掉电唤醒条件到真正唤醒有一个延迟时间，此时间用于晶振重新启动并稳定下来。唤醒周期与复位周期是一样的。

唤醒条件必须保持到 MCU 真正醒过来；否则，MCU 又会回到掉电模式。

AT90LS8535 单片机主频 4 MHz、3 V 供电时，不启动看门狗定时器，掉电模式下所需电流不到 1 μA。

3. 省电模式

这一模式与掉电模式只有一点不同：如果 T/C2 异步驱动，即 I/O 寄存器 ASSR 中的 AS2 置位，则 T/C2 在休眠时继续运行。除了掉电模式的唤醒方式，T/C2 的溢出中断和比较匹配中断也可以将 MCU 从休眠状态唤醒。在省电模式下，如果由外部电平中断唤醒，则在中断标志更新之前，MCU 要执行 2 个周期；而若从 T/C2 唤醒，则需要 3 个周期。在此期间，处理器执行指令；但是中断条件不可读，中断例程也不执行。

即使全局中断是禁止的，异步方式的 T/C2 中断也将把 MCU 从省电模式唤醒。

进入休眠状态的条件是 MCUCR 中的 SE 位为 1，然后执行 SLEEP 指令。进入哪一种模式取决于 MCUCR 中 SM1 和 SM2 两位：00 为空闲模式；01 没定义；10 为掉电模式；11 为省电模式。

在休眠期间，寄存器文件及 I/O 寄存器的内容不会丢失。如果在休眠模式下复位，则 MCU 从 RESET 向量 $000 处开始运行。某些使能的中断将唤醒 MCU，完成中断例程后，MCU 执行 SLEEP 以后的指令。

AT90LS8535 单片机主频 4 MHz、3 V 供电时，不启动看门狗定时器，省电模式下所需电流不到 5 μA。

16.3　数字滤波

一般在单片机应用系统的模拟输入信号中，均含有种种噪音和干扰。它们来自被测信号源本身、传感器、外界干扰等。为了进行准确测量和控制，必须消除被测信号中的噪音和干扰。噪音有 2 大类：一类随机信号，另一类脉冲信号。随机信号不是周期信号。对于随机干扰，可以用数字滤波方法予以削弱或滤除。所谓数字滤波，就是通过一定的计算或判断程序减少干扰在有用信号中的比重；故实际上它是一种程序滤波。数字滤波克服了模拟滤波器的不足，与模拟滤波器相比，有以下几个优点：

(1) 数字滤波是用程序实现的，不需要增加硬设备；所以可靠性高，稳定性好。

(2) 数字滤波可以对频率很低(如 0.01Hz)的信号实现滤波，克服了模拟滤波器的缺陷。

(3) 数字滤波可以根据信号的不同，采用不同的滤波方法或滤波参数，具有灵活、方便、功能强的特点。

由于数字滤波具有以上优点，所以数字滤波在微机应用系统中得到了广泛的应用。

16.3.1　平滑滤波法

叠加在有用数据上的随机噪声在很多情况下可以近似地认为是白噪声。白噪声具有一个很重要的统计特性，即它的统计平均值为 0；因此，可以采用求平均值的方法消除随机误差。

这就是所谓平滑滤波。

1. 算术平均值法

算术平均值法用于对一般的、具有随机干扰的信号的滤波。它特别适于信号本身在某一数值范围附近上下波动的情况，如流量、液平面等信号的测量。

算术平均滤波法就是把 n 个采样值相加，然后取其算术平均值作为本次有效的采样值。

这种滤波的具体做法是，以第 1 次采样时刻为基准，向后递推 n 个周期；计算机存储 n 个周期的采样输入值，并累计采样次数；当采样数为 n 个，则进行累加运算，并求平均值，作为第 1 次的真实采样值送入计算机进行运算。

一般算术平均值滤波采样次数流量取 12 次，压力取 4 次，温度如无噪声可不平均。

2. 递推平均滤波法

算术平均滤波法每计算一次数据，需测量 n 次，对于测量速度较慢或要求计算速率较高的实时系统，则无法使用。如果在存储器中开辟一个区域作为暂存队列使用，队列的队长固定为 n，每进行一次新的测量，把测量结果放入队尾，而扔掉原来队首的数据，这样在队列中始终有 n 个“最新的”数据。这种方法就是递推平均滤波法。

递推平均项数的选取是比较重要的环节。n 选得过大，平均效果好，但对参数变化的反映不灵敏；n 选得过小，滤波效果不显著。关于 n 的选择，与算术平均滤波法一致。

3. 防脉冲干扰平均值

在工业控制等应用场合中，经常会遇到尖脉冲干扰的现象。

干扰通常只影响个别采样点的数据，此数据与其他采样点的数据相差比较大。如果采用一般的平均值法，则干扰将平均到计算结果中去。如果先对平均值法的 n 个数据进行比较，去掉其中最大值和最小值，然后计算余下的 n－2 个数据的算术平均值，这样既可滤去脉冲干扰，又可滤去小的随机干扰。

16.3.2 中位值滤波法

除了程序限幅滤波外，对某些变化速度不是太快的参数，为了去掉脉冲性干扰，也常常采用中位值法进行滤波。中位值滤波法的原理是：对被测参数连续采样 m 次（m≥3），并按大小顺序排列，以首尾部分各截去 1/3 个大小居中的数据进行平均，作为有效检测信号。中位值滤波法虽然对滤去脉冲性噪声比较有效，但对流量快速变化的过程不宜采用。

16.3.3 程序判断滤波法

当采样信号由于随机干扰和误检或者变送器不稳定而引起严重失真时，可采用程序判断滤波。程序判断的方法是根据经验，确定出 2 次采样输入信号可能出现的最大偏差 ΔY。若超过此偏差值，则表明该输入信号中串入干扰信号，应当舍弃；若小于此偏差值，可将信号作为本次采样值。

程序判断滤波方法有限幅滤波和限速滤波 2 种。所谓限速滤波，就是把 2 次相邻的采样值相减，求出其增量（以绝对值表示），然后与 2 次采样允许的最大差值（由被控对象的实际情况决定）ΔY 进行比较。如果小于或等于 ΔY，则取本次采样值；如果大于 ΔY，则仍取上次采样值作为本次采样值。即：

| Y(K)－Y(K－1) | ≤ΔY，则 Y(K)＝Y(K)取本次采样值；

$|Y(K)-Y(K-1)|>\Delta Y$，则 $Y(K)=Y(K-1)$取上次采样值。

式中：Y(K)—— 第 K 次采样值；

Y(K－1)——第 K－1 次采样值；

ΔY——2 次采样值所允许的最大偏差，其大小取决于采样周期 T 及 Y 值的变化动态响应。

这种程序滤波方法，主要用于变化比较缓慢的参数，如温度、物位等测量系统；也可在大电流、大电感负载切断时，即干扰的特点为时间短、但幅值却很大(有时可达 100～200 mV)的情况下使用。

这种方法对滤去脉冲性干扰十分有效。使用时 ΔY 为已知数，Y(K－1)为上次采样值(已予以保存)。本次采样后，计算机只要进行比较，就可以确定真正输入计算机的采样值。以后将第 K 次采样值存入第 K－1 次采样值所在的内存单元进行保存，以备下次使用。所以这种方法关键是选择 ΔY。太大，则易使干扰信号串入，使系统误差增大；太小，又可能会滤去有用信号，不能完全跟踪参数对象。因此 ΔY 必须选择恰当，一般可根据经验数据和由实验方法获得。

16.3.4　一阶滞后滤波法

在测量回路中，常常伴随有电源干扰及工业干扰。这些干扰的特点是频率很低(如频率为 0.01 Hz)。对这样低频的干扰信号，采用 RC 滤波显然是不适宜的，因为 C 太大很难做到；但在计算机控制系统中，用数字滤波的方法则容易解决。一阶递推滤波公式如下：

$$\overline{Y}(K)=(1-Q)\overline{Y}(K-1)+QX(K)$$

滤波系数 $Q=T/(T+\tau)$，其中 T 为采样周期，一般情况下是已知的；τ 为数字滤波器的时间常数，需通过实际运行求取最优值。方法是不断地改变 τ 值，当低频周期性噪声减至最弱、全部消除时，即为该系统的 τ 值。

从公式可见，这是一种迭代运算。Q 的选取决定了重视本次采样，还是重复前次输出值。一阶滞后滤波法缺点是，造成信号的相位滞后。滞后相位的大小与 Q 值有关，如果相位滞后太大，还必须采取其他补救措施。

16.4　标度变换

控制系统中检测到的各种物理量都通过传感器变为电信号，再经过一定的处理，送往 A/D 转换器，得到与被测物理量相对应的数字量。在检测系统中，同样的数字量表示不同物理量的不同值；所以必须通过一定的处理，将这些数字量转换成具有不同量纲的物理量，这就是“标度变换”技术。

标度变换的原理如下：

对于一般的线性控制系统来说，其标度变换公式为：

$$A_x=A_0+(A_m-A_0)(N_x-N_0)/(N_m-N_0)$$

式中：A_0——测量值的下限；

A_m——测量值的上限；

A_x——实际测量值(工程量)；

N_0——下限对应的数字量；

N_m——上限对应的数字量；

N_x——实际测量值对应的数字量。

该公式的实质是过测量值上限和下限 2 点的一条直线方程。其中 A_0，A_m，N_0，N_m 对应某一个固定的被测参数来说，是常数，可以事先存入存储器中。不同的参数可有不同的值；因此存储器中实际应存入许多组这样的数据，在进行标度变换时，可以根据需要调入不同的常数来计算。当然若把上式简化成最简形式，参数更少，编程也就更容易些。

例：某智能压力测控仪表输入信号来自压力变送器的 $V_{DC}=1\sim5$ V 信号，压力变送器的量程为 0～4 000 kPa，此 1～5 V 输入信号经某路 A/D 转换为 200～1 000 数字量（可以通过调节 $V_{REF}=5.120$ V 实现）。求测得压力 A_X 与数字量 N_X 的关系式。

解：根据标度变换公式，$A_0=0$ kPa，$A_m=4\ 000$ kPa，$N_0=200$（1 V 对应的数字量），$N_m=1\ 000$（5 V 对应的数字量）。

则
$$A_x=0+4\ 000(N_x-200)/(1\ 000-200)=5N_x-1\ 000$$

实现上述算式的编程是很容易的。

16.5 非线性关系的处理

16.5.1 查表法

在智能仪表设计过程中，经常遇到非线性关系，像热电偶在冷端温度固定的情况下，温度与其热电势的关系是非线性的。

例：一个测温仪表，测量元件采用 K 型热电偶，量程范围为0～1 200 ℃。当 1 200 ℃时，热电势为 48.83 mV，若放大器放大倍数为 100，经放大后为0～4.883 V。AT90S8535 单片机的 V_{REF} 取 5.120 V，由于 $X/1024=Et*100/V_{REF}$，所以 $X=100*Et*1024/5120=20Et$。

即 Et 每增加 0.05 mV，X 增 1，列表取热电势为 0 mV，0.05 mV，0.10 mV，0.15 mV，…，51.10 mV，51.15 mV 的值对应的温度值 0，1，3，4，…，1263，1265，则可根据 A/D 结果的数字量 X 查表求出对应的温度值。假设 A/D 结果在 r17：r16 中，如下程序可求出温度值在 r19：r18 中。

```
.include"8535def.inc"
        rjmp   RESET
tab:.dw 0,1,3,4,……,1263,1265        ;由数字量查温度的表，共 1 024 个字
RESET: add r16,r16                   ;A/D 结果 * 2
        adc r17,r17
        ldi Zh,high(tab * 2)         ;查温度值
        ldi Zl,low(tab * 2)
        add Zl,r16
        adc Zh,r17
        lpm
        mov r18,r0
        adiw Zl,1
        lpm
        mov r19,r0
here:   rjmp here
```

16.5.2　查表加线性插值法

对非线性关系可用查表法。热电偶的热电势与温度的关系，由于采用了放大器和 A/D 转换，转变成为 A/D 结果的数字量与温度的关系。A/D 结果可能为 0～1 023，对应 1 024 个温度值，每个温度值占 1 个字，整个表则占 2 KB。这种方法表格计算输入繁琐，且占用很大的程序存储器空间，一般不太常用。

查表加线性插值法是把整个区间等分成若干段，每段端点的值查表求出，段内按直线计算。为计算方便，分成 2^n 段，n 为自然数。常分成 8 段(9 个节点)、16 段(17 个节点)及 32 段(33 个节点)。一般来说，段数分的越多，查表误差就越小。当然，误差也与对象本身的特性有关，对象越接近线性，误差就越小。对于 16.5.1 中的例子也可采用这种查表加线性插值法，将区间分成 16 段(17 个节点)，见表 16.1。

表 16.1　数字量、热电势与温度对照表

数字量	Et/mv	t/℃	数字量	Et/mv	t/℃
0	0	0	576	28.8	692
64	3.2	78	640	32.0	769
128	6.4	157	704	35.2	847
192	9.6	236	768	38.4	927
256	12.8	314	832	41.6	1 009
320	16.0	391	896	44.8	1 092
384	19.2	466	960	48.0	1 177
448	22.4	541	1 024	51.2	1 266
512	25.6	616			

当数字量在节点上时，可以直接查表求出温度值；当数字量在 2 个节点之间时，过这 2 个节点做一条直线，就认为在这段内温度与数字量是线性的，计算方法如下。

例：已知 A/D 结果数字量 x＝900(0b1110000100)时，查表得出 x1＝896(0b1110000000)时的 y1＝1092，x2＝960(0b1111000000)时的 y2＝1177，则

$$y=y1+(y2-y1)(x-x1)/(x2-x1)=y1+(y2-y1)(900-x1)/64=1097$$

用这种思想编程如下。r17:r16 中为 A/D 转换数字量。

```
.include"8535def.inc"
      rjmp   RESET
tab:.dw 0,78,157,236,314,393,466,541,616 ,692,769,847,927,1009,1092,1177,1266 ;数字量温度对照表
RESET:mov   r20,r16            ;A/D 结果低 6 位送 r20
      andi r20,$3f
      rol   r16                ;取 A/D 结果高 4 位送 r17
      rol   r17
      rol   r16
      rol   r17
```

```
        add   r17,r17              ;r17 * 2
        ldi   Zh,high(tab * 2)     ;取 y1 送 r19:r18
        ldi   Zl,low(tab * 2)
        add   Zl,r17
        lpm
        mov   r18,r0
        adiw Zl,1
        lpm
        mov   r19,r0
        adiw Zl,1                  ;取 y2 送 r17:r16
        lpm
        mov   r16,r0
        adiw Zl,1
        lpm
        mov   r17,r0
        sub   r16,r18              ;求 y2-y1
        sbc   r17,r19
        mov   r23,r18              ;保存 y1 送 r24:r23
        mov   r24,r19
        mov   r18,r20              ;求(y2-y1) * (x-x1)/64
        clr   r19
        rcall mpy16s
        rol   r18
        rol   r19
        rol   r20
        rol   r18
        rol   r19
        rol   r20
        add   r23,r19              ;求 y
        adc   r24,r20
here: rjmp here

mpy16s:                            ;16 位 * 16 位带符号乘法
      clr     r21                  ;清结果和进位位
      sub     r20,r20
      ldi     r22,16               ;初始化循环计数器
m16s_1:
      brcc    m16s_2               ;进位位为 0,跳至 m16s-2
      add     r20,r16              ;进位位为 1
      adc     r21,r17              ;结果 3 字节,2 字节加被乘数
m16s_2:
      sbrc    r18,0                ;如果当前位置位
      sub     r20,r16              ;从结果字节 2 中减被乘数低字节
```

```
sbrc    r18,0          ;如果当前位置位
sbc     r21,r17        ;从结果字节 2 中减被乘数低字节
asr     r21            ;算术右移结果和乘数
ror     r20
ror     r19
ror     r18
dec     r22            ;循环计数器减 1
brne    m16s_1         ;如没完成,再循环
ret
```

16.5.3　用代数多项式近似非线性关系

在单片机应用系统中,经常会遇到需对传感器的输入进行非线性补偿或需进行复杂计算的情况。一般来说,不少传感器的传输特性是非线性的,并且有些没有明显的解析表达式,这时就可采用下面一个代数多项式来近似:

$$y=a_n x^n+a_{n-1}x^{n-1}+\cdots+a_1 x+a_0$$

这个代数多项式也叫插值多项式。有了这个多项式,很容易由 x 求出 y。如何确定这个多项式,关键是找到合适的 a_n,a_{n-1},…,a_1,a_0 及 n。下面以一种热敏电阻的补偿为例,说明插值代数多项式的求法。

热敏电阻具有灵敏度高、价格低廉等特点,在温度变化时,它的电阻值也发生较大的变化,因此对模拟电路包括放大器等要求较低。但是热敏电阻的温度特性一般是非线性的,需要用计算进行补偿。例如有一种热敏电阻,它的温度-电阻值特性 t=f(R)如表 16.2 所列。

表 16.2　一种热敏电阻的温度-电阻值特性

温度 t/℃	电阻值 R/kΩ	温度 t/℃	电阻值 R/kΩ	温度 t/℃	电阻值 R/kΩ
10	8.000	20	6.667	31	5.633 7
11	7.843 1	21	6.557 4	32	5.555 4
12	7.692 3	22	6.451 6	33	5.479 3
13	7.547 1	23	6.349 1	34	5.405 3
14	7.407 4	24	6.250	35	5.333 2
15	7.272 7	25	6.153 8	36	5.263
16	7.142 8	26	6.060 6	37	5.194 6
17	7.017 4	27	5.970 1	38	5.128 1
18	6.895 6	28	5.882 3	39	5.063 1
19	6.779 6	29	5.797	40	5.000
		30	5.714 2		

按照这个表,可使用计算机求出一个插值多项式,然后在程序中按这个多项式来进行补偿计算。计算时,首先必须根据所需要的逼近精度决定多项式的次数 n。具体次数与所要逼近的函数有关。例如函数关系接近线性的,可用一次多项式来逼近(n=1);接近抛物线的,可用二次多项式来逼近(n=2),…。同时,多项式次数还与自变量的范围有关。一般来说,自变量

的允许范围越大(即插值区间越大),达到同样精度时的多项式次数也较高。对于无法预先决定多项式次数的情况,可采用试探法。先选取一个较小的 n 值,看看逼近误差是否接近所要求的精度,如误差太大,则把 n 加 1,再试一次,直到误差接近精度要求为止。在能达到精度要求的情况下,n 不应取得太大,以免增加计算时间。

在决定多项式次数 n 后,应选择 n+1 个自变量 x 值和函数 y 值。由于一般表格中的函数关系对的数目均大于 n+1,故应选择适当的插值节点 x 和 y。插值节点的选择与插值多项式的误差大小有很大的关系。在同样的 n 值的条件下,选择合适的 x 和 y 值,可减小误差。在开始时,可先选择等分值的 x 和 y,以后再根据误差的分布情况,改变 x,y 的取值。现以表 16.2 为例,说明具体的计算方法。

假设要求在 10～40 ℃范围内使用该温度传感器,并且要求误差小于 0.05 ℃,可使用前面的插值多项式计算。这里 x 为电阻值 R(单位 kΩ),y 为温度值 t(单位为℃)。先试探 n=2,并选择插值节点为(8.000,10),(6.1538,25),(5.000,40)。求出一个插值多项式,用这个多项式计算出的 t 值与表 16.2 所列 t 值相比较。在校验中可发现,当 R=7.27 kΩ 左右时,计算出来的 t 值与表中的值相差 0.4 ℃左右,远大于要求值 0.05 ℃。这说明二次多项式不足以补偿该热敏电阻的温度特性;所以取 n=3,如按 t 等分值的原则可取插值节点为(8.000,10),(6.667,20),(5.714 2,30),(5.000,40),这时得到的最大误差约为 0.05 ℃。并且可以发现,最大误差在 15 ℃左右时发生,而在 25 ℃或 35 ℃左右时,误差小于 0.03 ℃,这说明插值节点取得不太合适,使误差分布不均匀,应在 10～20 ℃附近加密节点,以减小 15 ℃左右的误差。于是取插值节点为(8.000,10),(6.895 6,18),(5.882 3,28),(5.063 1,39)再计算一次,可看到最大误差约为 0.04 ℃,已小于允许值,满足设计要求。如果取插值节点为(8.000,10),(7.017 4,17),(5.970 1,27),(5.063 1,39),可得到最大误差约为 0.036 ℃。这时的多项式为:

$$t=f(R)=-0.234\ 698\ 9\ R^3+6.120\ 273R^2-59.280\ 43\ R+212.711\ 8$$

若 R 由 I mA 的恒流源供电,测得其两端压降的 A/D 转换数字量为 x,则电阻 R 可求得:

$$R=V_{REF}*x/(1024*I)$$

再将其转化为浮点数,温度 t 就可以计算出来了。

参 考 文 献

1 耿德根主编. AVR 高速嵌入式单片机原理与应用. 北京:北京航空航天大学出版社,2001

2 http://www.atmel.com

3 http://www.sl.com.cn

的允许范围越大(即插值区间越大),达到同样精度时的多项式次数也较高。对于无法预先决定多项式次数的情况,可采用试探法。先选取一个较小的 n 值,看看逼近误差是否接近所要求的精度,如误差太大,则把 n 加 1,再试一次,直到误差接近精度要求为止。在能达到精度要求的情况下,n 不应取得太大,以免增加计算时间。

在决定多项式次数 n 后,应选择 n+1 个自变量 x 值和函数 y 值。由于一般表格中的函数关系对的数目均大于 n+1,故应选择适当的插值节点 x 和 y。插值节点的选择与插值多项式的误差大小有很大的关系。在同样的 n 值的条件下,选择合适的 x 和 y 值,可减小误差。在开始时,可先选择等分值的 x 和 y,以后再根据误差的分布情况,改变 x,y 的取值。现以表 16.2 为例,说明具体的计算方法。

假设要求在 10～40 ℃范围内使用该温度传感器,并且要求误差小于 0.05 ℃,可使用前面的插值多项式计算。这里 x 为电阻值 R(单位 kΩ),y 为温度值 t(单位为℃)。先试探 n=2,并选择插值节点为(8.000,10),(6.1538,25),(5.000,40)。求出一个插值多项式,用这个多项式计算出的 t 值与表 16.2 所列 t 值相比较。在校验中可发现,当 R=7.27 kΩ 左右时,计算出来的 t 值与表中的值相差 0.4 ℃左右,远大于要求值 0.05 ℃。这说明二次多项式不足以补偿该热敏电阻的温度特性;所以取 n=3,如按 t 等分值的原则可取插值节点为(8.000,10),(6.667,20),(5.714 2,30),(5.000,40),这时得到的最大误差约为 0.05 ℃。并且可以发现,最大误差在 15 ℃左右时发生,而在 25 ℃或 35 ℃左右时,误差小于 0.03 ℃,这说明插值节点取得不太合适,使误差分布不均匀,应在 10～20 ℃附近加密节点,以减小 15 ℃左右的误差。于是取插值节点为(8.000,10),(6.895 6,18),(5.882 3,28),(5.063 1,39)再计算一次,可看到最大误差约为 0.04 ℃,已小于允许值,满足设计要求。如果取插值节点为(8.000,10),(7.017 4,17),(5.970 1,27),(5.063 1,39),可得到最大误差约为 0.036 ℃。这时的多项式为:

$$t=f(R)=-0.234\ 698\ 9\ R^3+6.120\ 273R^2-59.280\ 43\ R+212.711\ 8$$

若 R 由 I mA 的恒流源供电,测得其两端压降的 A/D 转换数字量为 x,则电阻 R 可求得:

$$R=V_{REF}*x/(1024*I)$$

再将其转化为浮点数,温度 t 就可以计算出来了。

参 考 文 献

1 耿德根主编. AVR 高速嵌入式单片机原理与应用. 北京:北京航空航天大学出版社,2001

2 http://www.atmel.com

3 http://www.sl.com.cn