ATmega8
原理及应用手册

马　潮　詹卫前　耿德根　编著

.1

清华大学出版社

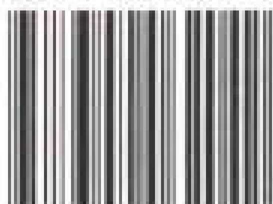
策划编辑：曾　刚
责任编辑：肖　丽
封面设计：秦　铭

ATmega8
原理及应用手册

ATmega8属于ATmega系列单片机（ATmega16/32/64/128）的一个子集，指令系统完全兼容。本书深入而细致地介绍了ATmega8单片机的硬件结构以及一些特殊功能的应用和设计，对掌握和使用其他ATmega系列的单片机具有极高的参考价值。

书中的程序均在广州市天河双龙电子有限公司的SL-MEGA8开发实验器上验证通过。源程序清单及硬件接线图、系统工作软件可通过HTTP://www.sl.com.cn下载获得。本书有较强的系统性和实用性，可作为高等院校自动化、计算机、仪器仪表、电子等专业的教学参考书以及有关工程技术人员和硬件工程师的应用手册。

ISBN 7-302-06204-8



定价：29.00元

738

ATmega8原理及应用手册

马 潮 詹卫前 耿德根 编著

A1017159

清华大学出版社
北 京

内容简介

ATmega8属于ATmega系列单片机（ATmega16/32/64/128）的一个子集，指令系统完全兼容。本书深入而细致地介绍了ATmega8单片机的硬件结构以及一些特殊功能的应用和设计，对掌握和使用其他ATmega系列的单片机具有极高的参考价值。

书中的程序均在广州市天河双龙电子有限公司的SL-MEGA8开发实验器上验证通过。源程序清单及硬件接线图、系统工作软件可通过HTTP://www.sl.com.cn下载获得。本书有较强的系统性和实用性，可作为高等院校自动化、计算机、仪器仪表、电子等专业的教学参考书以及有关工程技术人员和硬件工程师的应用手册。

版权所有，翻印必究。

本书封面贴有清华大学出版社激光防伪标签，无标签者不得销售。

图书在版编目（CIP）数据

ATmega8原理及应用手册/马潮，詹卫前，耿德根编著．—北京：清华大学出版社，2002

ISBN 7-302-06204-8

Ⅰ.A… Ⅱ.①马…②詹…③耿… Ⅲ.单片微型计算机，ATmega8-技术手册 Ⅳ.TP368.1-62

中国版本图书馆CIP数据核字（2002）第105905号

出 版 者： 清华大学出版社（北京清华大学学研大厦，邮编 100084）
http://www.tup.tsinghua.edu.cn

策划编辑： 曾 刚

责任编辑： 肖 丽

印 刷 者： 北京市密云胶印厂

发 行 者： 新华书店总店北京发行所

开　　本： 787×1092 1/16　**印张：** 19.5　**字数：** 446千字

版　　次： 2003年3月第1版　2003年3月第1次印刷

书　　号： ISBN 7-302-06204-8/TP・3710

印　　数： 0001～4000

定　　价： 29.00元

前　言

ATmega8 是 ATMEL 公司在 2002 年第一季度推出的一款新型 AVR 高档单片机。在 AVR 家族中，ATmega8 是一种非常特殊的单片机，它的芯片内部集成了较大容量的存储器和丰富强大的硬件接口电路，具备 AVR 高档单片机 MEGE 系列的全部性能和特点。但由于采用了小引脚封装（为 DIP 28 和 TQFP/MLF32），所以其价格仅与低档单片机相当，再加上 AVR 单片机的系统内可编程特性，使得无需购买昂贵的仿真器和编程器也可进行单片机嵌入式系统的设计和开发，同时也为单片机的初学者提供了非常方便和简捷的学习开发环境。

ATmega8 的这些特点，使其成为一款具有极高性能价格比的单片机，深受广大单片机用户的喜爱，在产品应用市场上极具竞争力，被很多家用电器厂商和仪器仪表行业看中，从而使 ATmega8 迅速进入大批量的应用领域。

ATmega 系列单片机属于 AVR 中的高档产品，它承袭了 AT90 所具有的特点，并在 AT90（如 AT90S8515、AT90S8535）的基础上，增加了更多的接口功能，而且在省电性能、稳定性、抗干扰性以及灵活性方面考虑得更加周全和完善。

目前在国内尚无对 ATmega 系列单片机的结构和使用进行详细介绍的资料和书籍，为此，本书以 ATmega8 为主线，详细介绍 ATmega 单片机的硬件结构、工作原理、指令系统和应用设计，并给出一些 ATmega 单片机所具有的特殊专用的硬件接口应用实例。ATmega8 属于 ATmega 系列单片机（ATmega16/32/64/128）的一个子集，指令系统完全兼容，所以学会 ATmega8 的应用，掌握其他 ATmega 系列的单片机就会驾轻就熟。

ATmega8 是一款采用低功耗 CMOS 工艺生产的基于 AVR RISC 结构的 8 位单片机。AVR 单片机的核心是将 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（算术逻辑单元）直接相连，实现了在一个时钟周期内执行的一条指令同时访问（读写）两个独立寄存器的操作。这种结构提高了代码效率，使得大部分指令的执行时间仅为一个时钟周期。因此，ATmega8 可以达到接近 1MIPS/MHz 的性能，运行速度比普通 CISC 单片机高出 10 倍。

ATmega8 的主要性能如下：

- 高性能、低功耗的 8 位 AVR 微控制器，先进的 RISC 精简指令集结构
  - ◇ 130 条功能强大的指令，大多数为单时钟周期指令
  - ◇ 32 个 8 位通用工作寄存器
  - ◇ 工作在 16MHz 时，具有 16MIPS 的性能
  - ◇ 片内集成硬件乘法器（执行速度为 2 个时钟周期）
- 片内集成了较大容量的非易失性程序和数据存储器以及工作存储器

◇ 8K 字节的 Flash 程序存储器，擦写次数：>10000 次
◇ 支持可在线编程（ISP）、可在应用自编程（IAP）
◇ 带有独立加密位的可选 BOOT 区，可通过 BOOT 区内的引导程序区（用户自己写入）来实现 IAP 编程。
◇ 512 个字节的 E^2PROM，擦写次数：100000 次
◇ 1K 字节内部 SRAM
◇ 可编程的程序加密位

- 丰富强大的外部接口（Peripheral）性能
  ◇ 2 个具有比较模式的带预分频器（Separate Prescale）的 8 位定时/计数器
  ◇ 1 个带预分频器（Separate Prescale），具有比较和捕获模式的 16 位定时/计数器
  ◇ 1 个具有独立振荡器的异步实时时钟（RTC）
  ◇ 3 个 PWM 通道，可实现任意<16 位、相位和频率可调的 PWM 脉宽调制输出
  ◇ 8 通道 A/D 转换（TQFP、MLF 封装），6 路 10 位 A/D + 2 路 8 位 A/D
  ◇ 6 通道 A/D 转换（PDIP 封装），4 路 10 位 A/D + 2 路 8 位 A/D
  ◇ 1 个 I^2C 的串行接口，支持主/从、收/发四种工作方式，支持自动总线仲裁
  ◇ 1 个可编程的串行 USART 接口，支持同步、异步以及多机通信自动地址识别
  ◇ 1 个支持主/从（Master/Slave）、收/发的 SPI 同步串行接口
  ◇ 带片内 RC 振荡器的可编程看门狗定时器
  ◇ 片内模拟比较器
- 特殊的微控制器性能
  ◇ 可控制的上电复位延时电路和可编程的欠电压检测电路
  ◇ 内部集成了可选择频率（1/2/4/8MHz）、可校准的 RC 振荡器（25℃、5V、1MHZ 时，精度为±1%）
  ◇ 外部和内部的中断源 18 个
  ◇ 五种睡眠模式：空闲模式（Idle）、ADC 噪声抑制模式（ADC Noise Reduction）、省电模式（Power-save）、掉电模式（Power-down）、待命模式（Standby）
- I/O 口和封装
  ◇ 最多 23 个可编程 I/O 口，可任意定义 I/O 的输入/输出方向；输出时为推挽输出，驱动能力强，可直接驱动 LED 等大电流负载；输入口可定义为三态输入，可以设定带内部上拉电阻，省去外接上拉电阻
  ◇ 28 脚 PDIP 封装，32 脚 TQFP 封装和 32 脚 MLF 封装
- 宽工作电压
  ◇ 2.7V~5.5V（ATmega8L）
  ◇ 4.5V~5.5V（ATmega8）
- 高运行速度
  ◇ 0~8MHz（ATmega8L）

◇ 0~16MHz（ATmega8）

● 低功耗（4MHz，3V，25℃）

◇ 正常模式（Active）：3.6mA

◇ 空闲模式（Idle Mode）：1.0mA

◇ 掉电模式（Power-down Mode）：0.5μA

为了使国内用户深入了解和掌握ATmega8的开发与应用，广州天河双龙电子有限公司已经研制开发出SL-MEGA8开发实验器（评估系统）。该开发实验器充分考虑到ATmega8的性能特点，在硬件上设计了与其配套的电路接口，同时也为用户提供相应的软件模块，使用户能快速上手，了解和熟悉ATmega8，设计出适合自己的产品。

ATmega8具有一整套的编程和系统开发工具，它包括宏汇编编译器、C语言编译器、BSCOM-BASIC语言编译器以及在线调试/仿真器和评估板。

本书共分5章，第1章ATmega8单片机简介；第2章ATmega8硬件结构；第3章ATmega8指令系统；第4章ATmega8开发工具；第5章ATmega8应用设计。书中所举硬件设计实例和软件均经实验通过。

本书由马潮（ma-chao@online.sh.cn）主编。第1、2、3章由马潮执笔，第4、5章由詹卫前、耿德根（sllg@sl.com.cn）执笔。书中插图由李青翻译制作，吴敏琪同学参与部分翻译工作。书中程序均在广州天河双龙电子有限公司的SL-MEGA8开发实验器上验证通过。广州市天河双龙电子有限公司（http://www.sl.com.cn）还提供AVR多媒体讲座及开发平台软件光盘，可作为本书的补充。

由于时间仓促，书中若有错误和疏漏之处，敬请读者批评指正。

编者

2002年9月于上海华东师范大学

为了方便广大读者学习、实践，本编辑部提供广州天河双龙电子有限公司的SL-MEGA8开发实验器以及本书的配套光盘。

地　址：清华大学校内金地公司　100084

电　话：010-62770384

联系人：韩丽娟

目　录

第 1 章　ATmega8 单片机简介 1
1.1　AVR 单片机 1
1.1.1　AVR 单片机简介 1
1.1.2　AVR 单片机的主要特点 2
1.1.3　AVR 单片机系列产品 4
1.2　ATmega8 单片机 7
1.2.1　ATmega8 单片机简介 7
1.2.2　ATmega8 单片机的结构与主要性能 8
1.2.3　ATmega8 单片机封装与引脚 11

第 2 章　ATmega8 硬件结构 14
2.1　ATmega8 MCU 内核 14
2.1.1　结构概述 14
2.1.2　微控制器（MCU） 16
2.1.3　MCU 工作时序 19
2.1.4　复位和中断处理 20
2.2　ATmega8 单片机存储器组织 21
2.2.1　支持可在线编程和可在应用自编程的 Flash 程序存储器 21
2.2.2　数据存储器（SRAM） 22
2.2.3　E^2PROM 数据存储器 23
2.2.4　I/O 寄存器 24
2.3　系统时钟和时钟选择 26
2.3.1　时钟系统和时钟分配 26
2.3.2　时钟源 27
2.3.3　外部晶振 28
2.3.4　外部低频率晶振 29
2.3.5　外部 RC 振荡器 30
2.3.6　可校准的内部 RC 振荡器 31
2.3.7　外部时钟源 32
2.3.8　定时器/计数器振荡器 33

2.4 电源管理和休眠模式......33
2.4.1 MCU 控制寄存器 MCUCR......34
2.4.2 空闲模式（IDLE MODE）......35
2.4.3 ADC 降噪模式（ADC Noise Reduction）......35
2.4.4 掉电模式（Power-down）......35
2.4.5 省电模式（Power-save）......36
2.4.6 等待模式（STANDBY MODE）......36
2.4.7 如何将功耗降到最低......37
2.5 系统复位......38
2.5.1 复位源......40
2.5.2 MCU 控制和状态寄存器 MCUCSR......42
2.5.3 内部参考电压源......43
2.5.4 看门狗定时器......43
2.6 中断向量......46
2.6.1 复位和中断向量表的移动......47
2.6.2 中断控制寄存器 GICR......49
2.7 I/O 端口......51
2.7.1 通用数字 I/O 接口......52
2.7.2 数字输入使能和休眠模式......56
2.7.3 端口的第二功能......57
2.8 外部中断......62
2.9 ATmega8 的定时器/计数器......65
2.9.1 定时器/计数器预定比例分频器......65
2.9.2 8 位定时器/计数器 0——T/C0......66
2.9.3 16 位定时器/计数器 1——T/C1......69
2.9.4 8 位定时器/计数器 2——T/C2......87
2.10 同步串行接口 SPI......101
2.10.1 SPI 接口控制与数据传输过程......101
2.10.2 与 SPI 接口相关的寄存器......107
2.11 通用同/异步串行接口 USART......109
2.11.1 概述......109
2.11.2 串行时钟发生......111
2.11.3 数据帧格式......113
2.11.4 USART 的初始化......114
2.11.5 数据发送......115
2.11.6 数据接收......118
2.11.7 异步串行数据的硬件扫描检测和接收时序......121

2.11.8 多机通信模式……122
2.11.9 USART 寄存器……123
2.11.10 串行通信波特率的设置与偏差……129
2.12 两线串行 TWI（I^2C）总线接口……131
2.12.1 两线串行总线接口定义……132
2.12.2 TWI 模块的概述……132
2.12.3 TWI 寄存器……134
2.12.4 使用 TWI 总线……137
2.12.5 多主机系统和仲裁……146
2.13 模拟比较器……147
2.14 模数转换功能 ADC……150
2.14.1 特点……150
2.14.2 启动 ADC 转换……152
2.14.3 预分频与转换定时……152
2.14.4 ADC 输入通道和参考电源的选择……154
2.14.5 ADC 噪声抑制器（NOISE CANCELER）……156
2.14.6 有关的 I/O 寄存器……158
2.15 引导加载支持的自编程功能……161
2.15.1 引导加载技术的实现……162
2.15.2 相关 I/O 寄存器……166
2.15.3 程序存储器 Flash 的自编程……169
2.15.4 一个简单的引导加载汇编程序……172
2.16 ATmega8 存储器编程……174
2.16.1 ATmega8 的锁定位、熔丝位、标识位和校正位……174
2.16.2 并行编程模式……177
2.16.3 串行编程模式……184
2.17 E^2PROM 数据存储器读/写访问……187
2.17.1 E^2PROM 读/写访问……187
2.17.2 寄存器描述……187
2.17.3 简单的读写 E^2PROM 例程……189

第 3 章 ATmega8 指令系统……192
3.1 ATmega8 指令总述……192
3.1.1 ATmega8 指令表……192
3.1.2 指令系统中使用的符号……200
3.1.3 寻址方式和寻址空间……201
3.2 算术和逻辑指令……204
3.2.1 加法指令……204

3.2.2 减法指令……205
3.2.3 取反码指令……206
3.2.4 取补码指令……207
3.2.5 比较指令……207
3.2.6 逻辑与指令……207
3.2.7 逻辑或指令……208
3.2.8 逻辑异或指令……209
3.2.9 乘法指令……209
3.3 转移指令……211
3.3.1 无条件转移指令……211
3.3.2 条件转移指令……212
3.3.3 子程序调用和返回指令……217
3.4 数据传送指令……219
3.4.1 直接寻址数据传送指令……219
3.4.2 间接寻址数据传送指令……220
3.4.3 从程序存储器中取数装入寄存器指令……221
3.4.4 写程序存储器指令……222
3.4.5 I/O口数据传送……222
3.4.6 堆栈操作指令……223
3.5 位操作和位测试指令……223
3.5.1 带进位逻辑操作指令……223
3.5.2 位变量传送指令……224
3.5.3 位变量修改指令……225
3.6 MCU控制指令……227
3.7 AVR汇编语言系统……228
3.7.1 汇编语言语句格式……228
3.7.2 汇编器伪指令……229
3.7.3 表达式……233
3.7.4 文件“m8def.inc”……235

第4章 ATmega8开发工具……237
4.1 AVR STUDIO（AVR集成开发环境）……237
4.1.1 汇编程序汇编器（AVR Assembler）……237
4.1.2 仿真调试……240
4.2 AVR单片机C编译器——ICCAVR的使用……241
4.2.1 ICCAVR编译器的安装……241
4.2.2 ICCAVR介绍……242
4.2.3 ICCAVR导游……244

4.2.4 ICCAVR 的 IDE 环境 246
4.2.5 C 库函数与启动文件 248
4.2.6 访问 AVR 硬件的编程 257
4.2.7 应用简单举例 268
4.3 SL-MEGA8 开发实验器 270
4.3.1 SL- MEGA8 开发实验器硬件结构 271
4.4 ATmega8 的编程操作 273

第 5 章 ATmega8 应用设计 279
5.1 硬件 I^2C 的应用 279
5.2 A/D 转换器的应用 284
5.3 USART 接口的应用 287
5.4 ATmega8 实时时钟的应用 290
5.5 BOOT 引导区的应用 293

第 1 章 ATmega8 单片机简介

1.1 AVR 单片机

1.1.1 AVR 单片机简介

ATMEL 公司是世界上有名的生产高性能、低功耗、非易失性存储器和各种数字模拟IC 芯片的半导体制造公司。在单片机微控制器方面，ATMEL 公司有 AT89、AT90 和 ARM 三个系列单片机的产品。ATMEL 公司在其单片机产品中，融入了先进的 E^2PROM 电可擦除和 Flash ROM 闪速存储器技术，使得该公司的单片机具备了优秀的品质，在结构、性能和功能等方面都有明显的优势。

自 1983 年 INTEL 公司推出 8051 单片机系列至今已有 20 年，ATMEL 公司把 8051 内核与其擅长的 Flash 制造技术相结合，推出了片内集成可重复擦写 1000 次以上 Flash 程序存储器、低功耗、8051 内核的 AT89 系列单片机。该系列的典型产品有 AT89C51、AT89C52、AT89C1051、AT89C2051，在我国的单片机市场上占有相当大的份额，得到了广泛的使用。

由于 8051 本身结构的先天性不足和近年来各种采用新型结构和新技术的单片机的不断涌现，现在的单片机市场是百花齐放。ATMEL 在这种强大市场压力下，发挥 Flash 存储器的技术特长，于 1997 年研发并推出了全新配置的、采用精简指令集 RISC（Reduced Instruction Set CPU）结构的新型单片机，简称 AVR 单片机。

精简指令集 RISC 结构是 20 世纪 90 年代开发出来的，综合了半导体集成技术和软件性能的新结构。AVR 单片机采用 RISC 结构，具有 1MIPS / MHz 的高速运行处理能力。

为了缩短产品进入市场的时间，简化系统的维护和支持，对于由单片机组成的嵌入式系统来说，用高级语言编程已成为一种标准编程方法。AVR 结构单片机的开发目的就在于能够更好地采用高级语言（例如 C 语言、BASIC 语言）来编写嵌入式系统的系统程序，从而能高效地开发出目标代码。为了对目标代码大小、性能及功耗进行优化，AVR 单片机的结构中采用了大型快速存取寄存器组和快速的单周期指令系统。

传统的基于累加器的结构单片机，如 8051，需要大量的程序代码，以实现在累加器和存储器之间的数据传送。而在 AVR 单片机中，采用 32 个通用工作寄存器组成快速存取寄存器组，用 32 个通用工作寄存器代替了累加器，从而避免了在传统结构中累加器和存储器之间数据传送造成的瓶颈现象。

AVR 单片机运用 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

一个周期执行指令。在其他的 CISC 以及类似的 RISC 结构的单片机中，外部振荡器的时钟被分频降低到传统的内部指令执行周期，这种分频最大达 12 倍（8051）。AVR 单片机是用一个时钟周期执行一条指令的，它是在 8 位单片机中第一个真正的 RISC 结构的单片机。

由于 AVR 单片机采用了 Harvard 结构，所以它的程序存储器和数据存储器是分开组织和寻址的。寻址空间分别为可直接访问 8M 字节的程序存储器和 8M 字节的数据存储器。同时，由 32 个通用工作寄存器所构成的寄存器组被双向映射，因此，可以采用读写寄存器和读写片内快速 SRAM 存储器两种方式来访问 32 个通用工作寄存器。

AVR 单片机采用低功率、非挥发的 CMOS 工艺制造，内部分别集成 Flash、E^2PROM 和 SRAM 三种不同性能和用途的存储器。除了可以通过 SPI 口和一般的编程器对 AVR 单片机的 Flash 程序存储器和 E^2PROM 数据存储器进行编程外，绝大多数的 AVR 单片机还具有在线编程（ISP）的特点，这给学习和使用 AVR 单片机带来了极大的方便。

1.1.2 AVR 单片机的主要特点

AVR 单片机吸取了 PIC 及 8051 单片机的优点，同时还做了一些重大改进，其主要的优点如下：

- 片内集成可擦写 10000 次以上的 Flash 程序存储器。由于 AVR 采用 16 位的指令，所以一个程序存储器的存储单元为 16 位，即 XXXX*16（也可理解为 8 位，即 2*XXXX*8）。AVR 的数据存储器还是以 8 个 Bit（位）为一个单元，因此 AVR 还是属于 8 位单片机。
- 采用 CMOS 工艺技术，高速度（50ns）、低功耗（μA）、具有 SLEEP（休眠）功能。AVR 的指令执行速度可达 50ns（20MHz），而耗电则在 1μA～2.5mA 之间（典型功耗，WDT 关闭时为 100nA）。AVR 运用 Harvard 结构概念，具有预取指令的特性，即对程序存储和数据存取使用不同的存储器和总线。当执行某一指令时，下一指令被预先从程序存储器中取出，这使得指令可以在每一个时钟周期内执行。
- 高度保密（LOCK）。可多次擦写的 FLASH 具有多重密码保护锁死（LOCK）功能，因此可低成本高速度地完成产品商品化，并且可多次更改程序（产品升级）而不必浪费 IC 或电路板，大大提高了产品的质量及竞争力。
- 工业级（WDT）产品。具有大电流（灌电流）10mA～20mA 或 40mA（单一输出）的特点，可直接驱动 SSR 或继电器。有看门狗定时器（WDT）安全保护，可防止程序走飞，提高产品的抗干扰能力。
- 超功能精简指令。具有 32 个通用工作寄存器（相当于 8051 中的 32 个累加器），克服了单一累加器数据处理造成的瓶颈现象，128～4K 字节 SRAM 可灵活使用指令运算，并可用功能很强的 C 语言编程，易学、易写、易移植。
- 程序写入器件可以并行写入（用编程器写入），也可使用串行在线编程（ISP）方法下载写入，也就是说不必将单片机芯片从系统上拆下，拿到万用编程器上烧写，

而可直接在电路板上进行程序的修改、烧写等操作，方便产品升级，尤其是采用SMD封装，更利于产品微型化。

- 除了并行I/O口输入/输出特性与PIC的HI/LOW输出及三态高阻抗HI-Z输入相同外，还设定与8051系列内部有上拉电阻的输入端功能相似的功能，以便适应各种实际应用特性所需（多功能I/O口）。只有AVR才是真正的I/O口，能正确反映I/O口的输入/输出的真实情况。
- 单片机内集成了模拟比较器，I/O口可作A/D转换用，组成廉价的A/D转换器。
- 像8051一样，AVR单片机有多个固定中断向量入口地址，可快速响应中断，而不会像PIC那样，所有中断都在同一向量地址发生，需要由程序判别后才可响应，从而失去了控制的最佳机会。
- 同PIC一样，AVR单片机可重新设置启动复位。AVR也有内部电源上电启动计数器，可将低电平复位（RESET）直接接到V_{CC}端。当系统上电时，利用内部的RC看门狗定时器可延迟MCU的启动，执行系统程序。这种延时可使I/O口稳定后再执行程序，提高了单片机工作的可靠性，同时也省略了外加的复位延时电路。
- 具有休眠省电功能（POWER DOWN）及闲置（IDLE）低功耗功能的工作方式。
- AT90S1200等部分AVR器件具有内部的RC振荡器，提供1MHz～8MHz的工作时钟，使该类单片机无需外加晶振等时钟电路元器件即可工作，简单方便。
- 有8位和16位的计数器/定时器（C/T），可作比较器、计数器、外部中断和PWM（也可作D/A）用于控制输出。
- 有串行异步通信UART硬件接口电路，采用单独的波特率发生器，并不占用定时器。有SPI传输功能。因其高速，故可以在一般标准整数频率下工作，而波特率可达576Kbps。
- AT90S4414及AT90S8515具有可扩展外部存储器达64KB的能力，它们的引脚排列及功能与8051相似，即可替代8051系列单片机（8751/8752）的应用系统。当然，在硬件、软件上也带来了很多优点（WDT看门狗，模拟比较器作A/D，PWM作D/A等）。
- 工作电压范围为2.7V～6.0V，电源抗干扰性能强。
- 多通道的10位A/D及实时时钟(RTC)。具有8路10位A/D器件的有AT90S4434、AT90S8535，具有6路10位A/D器件的有AT90S2333、AT90S4433。
- 高档AVR单片机MEGA系列的性能更加强大。如ATmega128有更大容量的存储器（Flash 128KB、E^2PROM 4KB、RAM 4KB），I/O端口53个、中断源34个、外部中断8个、SPI接口1个、SUART接口1个、I^2C接口1个、8位定时器2个、16位定时器2个、PWM接口8个，有看门狗定时器，有实时时钟RTC，模拟比较器，8路10位A/D，可在线编程（ISP）和在应用自编程（IAP），片内有RC振荡器、上电复位延时电路和可编程的欠电压检测电路，工作电压为2.7V～5.5V。
- AVR单片机还在片内集成了可擦写100000次的E^2PROM数据存储器，等于又增加了一个芯片，可用于保存系统的设定参数、固定表格和掉电后的数据保存，既方便

了使用，减小了系统的空间，又大大提高了系统的保密性。

1.1.3　AVR 单片机系列产品

AVR 单片机已形成一个系列。为满足不同的需求和应用，ATMEL 公司对 AVR 单片机的内部资源进行了相应的扩展和删减，推出了 ATtiny、AT90 和 ATmega 分别对应低、中和高三个不同档次的 50 多种型号的产品。它们的引脚数从 8 到 64，价格从几元到上百元，用户可以根据需要选择。

尽管 AVR 单片机产品的功能和内部配置不同，但其基本结构是一样的，指令系统是兼容的。表 1-1~1-3 为部分 AVR 系列单片机的选型表。

表 1-1　部分 AVR 系列单片机选型表①（录自 2002 年 ATMEL 网站）

内部资源	ATtiny11L	ATtiny11	ATtiny12V	ATtiny12L	ATtiny12	ATtiny15L	ATtiny26L	ATtiny26	ATtiny28V	ATtiny28L
Flash（KB）	1	1	1	1	1	1	2	2	2	2
E^{2}PROM（B）	-	-	64	64	64	64	128	128	-	-
RAM（B）	0	0	0	0	0	0	128	128	0	0
快速寄存器	32	32	32	32	32	32	32	32	32	32
指令条数	90	90	90	90	90	90	118	118	90	90
I/O Pins	6	6	6	6	6	6	16	16	20	20
中断数	4	4	5	5	5	8	11	11	5	5
外部中断数[1]	1(+5)	1(+5)	1(+5)	1(+5)	1(+5)	1(+5)	1(+8)	1(+8)	1(+8)	1(+8)
SPI	-	-	-	-	-	-	1[6]	1[6]	-	-
UART	-	-	-	-	-	-	1[5]	1[5]	-	-
TWI[4]	-	-	-	-	-	-	1[4]	1[4]	-	-
硬件乘法器	-	-	-	-	-	-	-	-	-	-
8 位定时器	1	1	1	1	1	2	2	2	1	1
16 位定时器	-	-	-	-	-	-	-	-	-	-
PWM	-	-	-	-	-	1	4	4	-	-
看门狗定时器	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
实时时钟	-	-	-	-	-	-	-	-	-	-
模拟比较器	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
10 位 A/D 通道	-	-	-	-	-	4	11	11	-	-
片内振荡器	Y	Y	Y[2]	Y[2]	Y[2]	Y[2]	Y[2]	Y[2]	Y[2]	Y[2]
BOD	-	-	Y	Y	Y	Y	Y	Y	-	-
在线编程（ISP）	Y[3]	Y[3]	Y	Y	Y	Y	Y	Y	Y[3]	Y[3]
自编程（SPM）	-	-	-	-	-	-	-	-	-	-
Vcc（V）（最低）	2.7	4.0	1.8	2.7	4.0	2.7	2.7	4.5	1.8	2.7
（最高）	5.5	5.5	5.5	5.5	5.5	5.5	5.5	5.5	5.5	5.5

续表

内部资源	ATtiny11L	ATtiny11	ATtiny12V	ATtiny12L	ATtiny12	ATtiny15L	ATtiny26L	ATtiny26	ATtiny28V	ATtiny28L
系统时钟(MHz)	0-2	0-6	0-1	0-4	0-8	1-6	0-8	0-16	0-1	0-4
封装形式	8-Pin DIP 8-Pin SOIC	8-Pin DIP 8-Pin SOIC	8-Pin DIP 8-Pin SOIC	8-Pin DIP 8-Pin SOIC	8-Pin DIP 8-Pin SOIC	8-Pin DIP 8-Pin SOIC	20-Pin DIP SOIC 32-Pin MLF	20-Pin DIP SOIC 32-Pin MLF	28-Pin DIP 32-Pin TQFP MLF	28-Pin DIP 32-Pin TQFP MLF

表1-2　部分AVR系列单片机选型表②（录自2002年ATMEL网站）

内部资源	AT90LS1200	AT90S1200	AT90LS2313	AT90S2313	AT90LS2323	AT90S2323	AT90LS2343	AT90S2343	AT90LS4433	AT90S4433
Flash（KB）	1	1	2	2	2	2	2	2	4	4
E^2PROM（B）	64	64	128	128	128	128	128	128	256	256
RAM（B）	0	0	128	128	128	128	128	128	128	128
快速寄存器	32	32	32	32	32	32	32	32	32	32
指令条数	89	89	120	120	120	120	120	120	120	120
I/O Pins	15	15	15	15	3	3	5	5	20	20
中断数	3	3	10	10	2	2	2	2	14	14
外部中断数[1]	1	1	2	2	1	1	1	1	2	2
SPI	-	-	-	-	-	-	-	-	1	1
UART	-	-	1	1	-	-	-	-	1	1
TWI[4]	-	-	-	-	-	-	-	-	-	-
硬件乘法器	-	-	-	-	-	-	-	-	-	-
8位定时器	1	1	1	1	1	1	1	1	1	1
16位定时器	-	-	1	1	-	-	-	-	1	1
PWM	-	-	1	1	-	-	-	-	1	1
看门狗定时器	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
实时时钟	-	-	-	-	-	-	-	-	-	-
模拟比较器	Y	Y	Y	Y	-	-	-	-	Y	Y
10位A/D通道	-	-	-	-	-	-	-	-	6	6
片内振荡器	Y	Y	-	-	-	-	Y	Y	-	-
BOD	-	-	-	-	-	-	-	-	Y	Y
在线编程（ISP）	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
自编程（SPM）	-	-	-	-	-	-	-	-	-	-
Vcc（V）（最低）	2.7	4.0	2.7	4.0	2.7	4.0	2.7	4.0	2.7	4.0
（最高）	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
系统时钟(MHz)	0-4	0-12	0-4	0-10	0-4	0-10	0-4	0-10	0-4	0-8

续表

内部资源	AT 90LS 1200	AT 90S 1200	AT 90LS 2313	AT 90S 2313	AT 90LS 2323	AT 90S 2323	AT 90LS 2343	AT 90S 2343	AT 90LS 4433	AT 90S 4433
封装形式	20-Pin DIP SOIC SSOP	20-Pin DIP SOIC SSOP	20-Pin DIP SOIC	20-Pin DIP SOIC	8-Pin DIP SOIC	8-Pin DIP SOIC	8-Pin DIP SOIC	8-Pin DIP SOIC	28-Pin DIP 32-Pin TQFP	28-Pin DIP 32-Pin TQFP

表 1-3 部分 AVR 系列单片机选型表③（录自 2002 年 ATMEL 网站）

内部资源	AT 90LS 8515	AT 90S 8515	AT 90LS 8535	AT 90S 8538	AT mega 8L	AT mega 8	AT mega 16	AT mega 32	AT mega 64	AT mega 128
Flash（KB）	8	8	8	8	8	8	16	32	64	128
E^2PROM（B）	512	512	512	512	512	512	512	1K	2K	4K
RAM（B）	512	512	512	512	1K	1K	1K	2K	4K	4K
快速寄存器	32	32	32	32	32	32	32	32	32	32
指令条数	120	120	120	120	130	130	130	130	133	133
I/O Pins	32	32	32	32	23	23	32	32	53	53
中断数	12	12	16	16	18	18	20	20	34	34
外部中断数[1]	2	2	2	2	2	2	3	3	8	8
SPI	1	1	1	1	1	1	1	1	1	1
UART	1	1	1	1	1[5]	1[5]	1[5]	1[5]	2[5]	2[5]
TWI[4]	-	-	-	-	1	1	1	1	1	1
硬件乘法器	-	-	-	-	Y	Y	Y	Y	Y	Y
8 位定时器	1	1	2	2	2	2	2	2	2	2
16 位定时器	1	1	1	1	1	1	1	1	2	2
PWM	2	2	3	3	3	3	4	4	6+2	6+2
看门狗定时器	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
实时时钟	-	-	Y	Y	Y	Y	Y	Y	Y	Y
模拟比较器	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
10 位 A/D 通道	-	-	8	8	6/8	6/8	8	8	8	8
片内振荡器	-	-	-	-	Y[2]	Y[2]	Y[2]	Y[2]	Y[2]	Y[2]
BOD	-	-	-	-	Y	Y	Y	Y	Y	Y
在线编程（ISP）	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
自编程（SPM）	-	-	-	-	Y	Y	Y	Y	Y	Y
Vcc（V）（最低）	2.7	4.0	2.7	4.0	2.7	4.5	4.5	4.5	4.5	4.0
（最高）	6.0	6.0	6.0	6.0	5.5	5.5	5.5	5.5	5.5	5.5
系统时钟(MHz)	0-4	0-8	0-4	0-8	0-8	1-16	0-16	0-16	0-16	0-16

续表

内部资源	AT 90LS 8515	AT 90S 8515	AT 90LS 8535	AT 90S 8538	AT mega 8L	AT mega 8	AT mega 16	AT mega 32	AT mega 64	AT mega 128
封装形式	40-Pin DIP 44-Pin PLCC TQFP	40-Pin DIP 44-Pin PLCC TQFP	40-Pin DIP 44-Pin PLCC TQFP	40-Pin DIP 44-Pin PLCC TQFP	28-Pin DIP 32-Pin MLF TQFP	28-Pin DIP 32-Pin MLF TQFP	40-Pin DIP 44-Pin MLF TQFP	40-Pin DIP 44-Pin MLF TQFP	64-Pin TQFP MLF	64-Pin TQFP MLF

说明：该表为ATMEL网站2002年7月刊登。

1. 外部中断和I/O引脚变化的唤醒。
2. 高精度（5%）可编程的内部RC振荡器
3. 编程时需在RESET脚加12V电源。
4. 兼容I^2C。
5. 可编程串行USART
6. 通过USI（Universal Serial Interface）实现。

1.2 ATmega8单片机

1.2.1 ATmega8单片机简介

在市场上，其他系列单片机的一般规律是：低档型单片机的引脚少，同时在内部集成的存储器容量也较小，功能也较简单；而中高档型的单片机则在内部集成的存储器容量较大，功能也多，但引脚数一般为（或大于）40个。对于一些不需要单片机有较多的外围引脚，同时系统需要大量的运算，程序代码较长，功能较复杂的应用，往往没有合适的芯片。

为了解决这一矛盾，ATMEL公司在2002年第一季度推出了一款新型AVR高档单片机（见表1-1）。在AVR家族中，ATmega8是一个非常特殊的单片机，它的芯片内部集成了较大容量的存储器和丰富强大的硬件接口电路，具备AVR高档单片机MEGE系列的全部性能和特点，但由于采用了小引脚封装（为DIP 28和TQFP/MLF32），所以其价格仅与低档单片机相当，成为具有极高性价比、深受广大用户喜爱的单片机。再加上AVR单片机的ISP性能，用户往往不需要购买昂贵的仿真器和编程器也可进行单片机嵌入式系统的开发应用，同时也为单片机的初学者提供了非常方便和简捷的学习开发环境。ATmega8的高性能低价格，在产品应用市场上具有强大的竞争力，被很多家用电器厂商和仪器仪表行业看中，从而使ATmega8 进入大批量的应用领域

为了使国内用户深入了解牢固掌握ATmega8的开发与应用，广州双龙电子有限公司迅速开发出SL-MEGA8开发实验器（评估系统）。该开发实验器充分考虑到ATmega8的性能

特点，在硬件上设计了与其配套电路接口，同时也为用户提供相应的软件模块，使用户能快速上手，了解和熟悉ATmega8，设计出适合自己的产品。ATmega8属于ATmega系列单片机（ATmega16/32/64/128）的一个子集，指令系统完全兼容，所以学会ATmega8的应用，对掌握其他ATmega系列的单片机非常有益。本书以ATmega8单片机为主，介绍AVR单片机的结构、指令系统，并给出ATmega8一些特殊专用硬件接口的应用实例。

1.2.2　ATmega8单片机的结构与主要性能

ATmega8是一款基于AVR RISC、低功耗CMOS的8位单片机，由于在一个时钟周期内执行一条指令，ATmega8可以达到接近1MIPS/MHz的性能。图1.1为ATmega8的结构图。

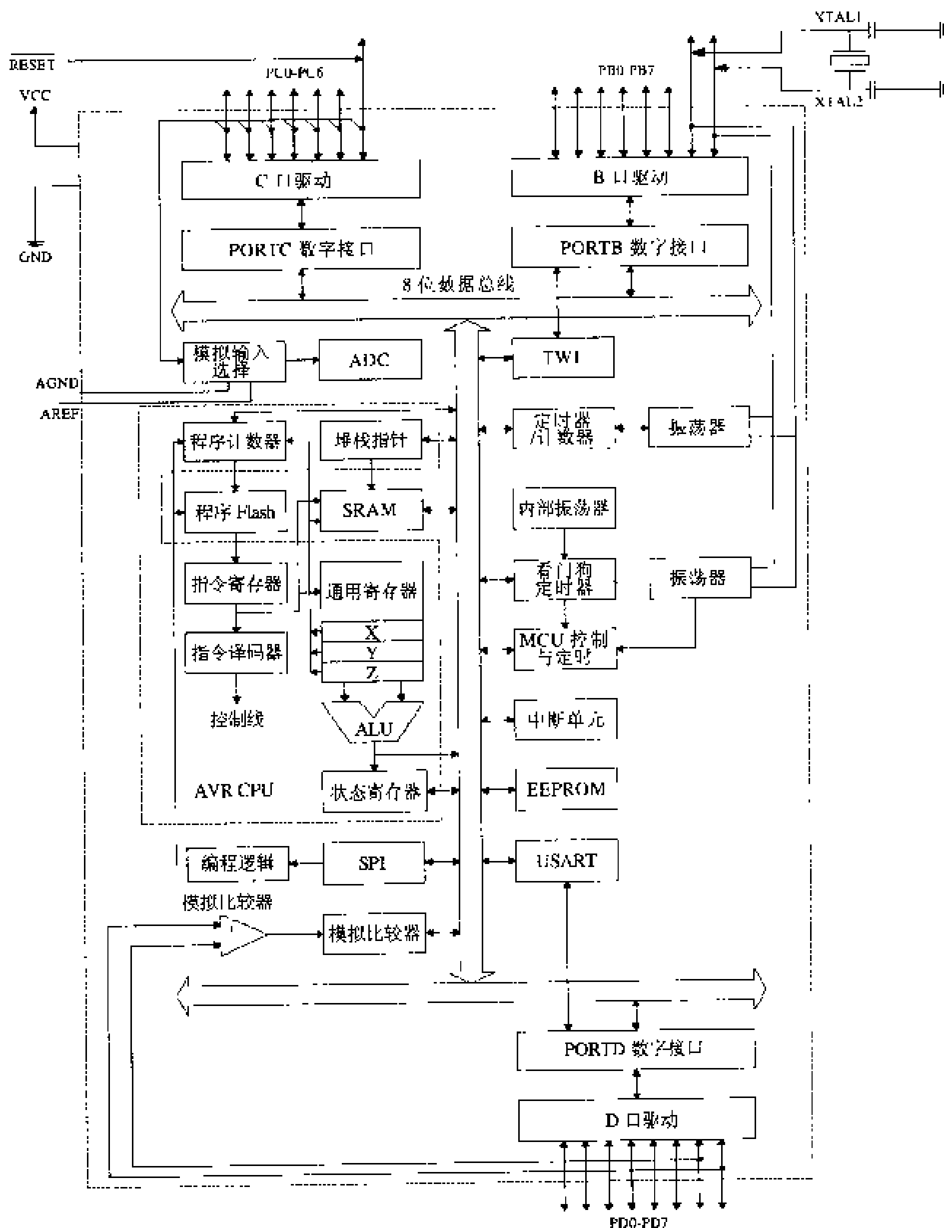


图1.1　ATmega8结构框图

AVR单片机的核心是将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（算术逻辑单元）直接相连，实现了在一个时钟周期内执行的一条指令可以同时访问两个独立的寄存器，这种结构提高了代码效率，使AVR的运行速度比普通CISC单片机高出10倍。

ATmega8具有以下特点：8K字节的在线编程/应用编程（ISP/IAP）Flash程序存储器，512字节E^2PROM，1K字节SRAM，32个通用工作寄存器，23个通用I/O口，3个带有比较模式灵活的定时器/计数器，18+2个内外中断源，1个可编程的SUART接口，1个8位I^2C总线接口，4（6）通道的10位ADC，2通道8位ADC，可编程的看门狗定时器，1个SPI接口和5种可通过软件选择的节电模式。

当单片机处于空闲模式时，CPU将停止运行，而SRAM、定时器/计数器、SPI口和中断系统则继续工作；处于掉电模式时，振荡器停止工作，所有其他功能都被禁止，但寄存器内容得到保留，只有在外部中断或硬件复位时才退出此状态；处于省电模式时，芯片的所有功能被禁止（处于休眠），只有异步时钟正常工作，以维持时间基准。当单片机处于ADC噪声抑制模式时，CPU和其他的I/O模块都停止运行，只有ADC和异步时钟正常工作，以减少ADC转换过程中的开关噪声；在待命模式时，CPU和其他的I/O模块都停止运行，但系统振荡器仍在运行，使得系统在低功耗时可以很快地启动。

ATmega8单片机采用了ATMEL的高密度非易失性内存技术，片内Flash可以通过SPI接口、通用编程器及自引导BOOT程序进行编程和自编程。利用自引导BOOT程序，可以使用任一硬件接口下载应用程序，并写入到Flash的应用程序区中。在更新Flash的应用程序区数据时，处在Flash的BOOT区中的自引导程序将继续执行，实现了同时读写（Read-While-Write）的功能（芯片自编程功能）。由于将增强RISC 8位CPU与在系统编程和在应用编程的Flash存储器集成在一个芯片内，ATmega8成为一个功能强大的单片机，为许多嵌入式控制应用提供了灵活而低成本的解决方案。

ATmega8具有一整套的编程和系统开发工具，它包括宏汇编编译器、C语言编译器、在线调试/仿真器和评估板。

ATmega8是AVR高档单片机中内部接口丰富、功能齐全、性能价格比最好的品种。它的主要性能如下：

- 高性能，低功耗的8位AVR微控制器
- 先进的RISC精简指令集结构
  - ◇ 130条功能强大的指令，大多数为单时钟周期指令
  - ◇ 32个8位通用工作寄存器
  - ◇ 工作在16MHz时具有16MIPS的性能
  - ◇ 片内集成硬件乘法器（执行速度为2个时钟周期）
- 片内集成了较大容量的非易失性程序和数据存储器以及工作存储器
  - ◇ 8K字节的在Flash程序存储器，擦写次数：>10000次
  - ◇ 支持在线编程（ISP）、在应用自编程（IAP）
  - ◇ 带有独立加密位的可选BOOT区，可通过BOOT区内的引导程序区（用户自己

写入）来实现 IAP 编程

◇ 512 个字节的 E^2PROM，擦写次数：100000 次

◇ 1K 字节内部 SRAM

◇ 可编程的程序加密位

● 外部（Peripheral）性能

◇ 2 个具有比较模式的带预分频器（Separate Prescale）的 8 位定时/计数器

◇ 1 个带预分频器（Separate Prescale），具有比较和捕获模式的 16 位定时/计数器

◇ 1 个具有独立振荡器的异步实时时钟（RTC）

◇ 3 个 PWM 通道，可实现任意<16 位、相位和频率可调的 PWM 脉宽调制输出

◇ 8 通道 A/D 转换（TQFP、MLF 封装），6 路 10 位 A/D + 2 路 8 位 A/D

◇ 6 通道 A/D 转换（PDIP 封装），4 路 10 位 A/D + 2 路 8 位 A/D

◇ 1 个 I^2C 的串行接口，支持主/从、收/发四种工作方式，支持自动总线仲裁

◇ 1 个可编程的串行 USART 接口，支持同步、异步以及多机通信自动地址识别

◇ 1 个主/从（Master/Slave）、收/发的 SPI 同步串行接口

◇ 带片内 RC 振荡器的可编程看门狗定时器

◇ 片内模拟比较器

● 特殊的微控制器性能

◇ 上电复位和可编程的欠电压检测电路

◇ 内部集成了可选择频率（1/2/4/8MHz）、可校准的 RC 振荡器（25℃、5V、1MHz 时精度为±1%）

◇ 外部和内部的中断源 18 个

◇ 五种休眠模式：空闲模式（Idle）、ADC 噪声抑制模式（ADC Noise Reduction）、省电模式（Power-save）、掉电模式（Power-down）、待命模式（Standby）

● I/O 口和封装

◇ 最多 23 个可编程 I/O 口，可任意定义 I/O 的输入/输出方向；输出时为推挽输出，驱动能力强，可直接驱动 LED 等大电流负载；输入口可定义为三态输入，可以设定带内部上拉电阻，省去外接上拉电阻

◇ 28 脚 PDIP 封装，32 脚 TQFP 封装和 32 脚 MLF 封装

● 工作电压

◇ 2.7V~5.5V（ATmega8L）

◇ 4.5V~5.5V（ATmega8）

● 运行速度

◇ 0~8MHz（ATmega8L）

◇ 0~16MHz（ATmega8）

● 功耗（4MHz，3V，25℃）

◇ 正常模式（Active）：3.6mA

◇ 空闲模式（Idle Mode）：1.0mA

◇ 掉电模式（Power-down Mode）：0.5μA

1.2.3　ATmega8 单片机封装与引脚

ATmega8 有三种不同形式的封装：PDIP、TQFP 和 MLF。图 1.2 为三种不同封装的式样图。

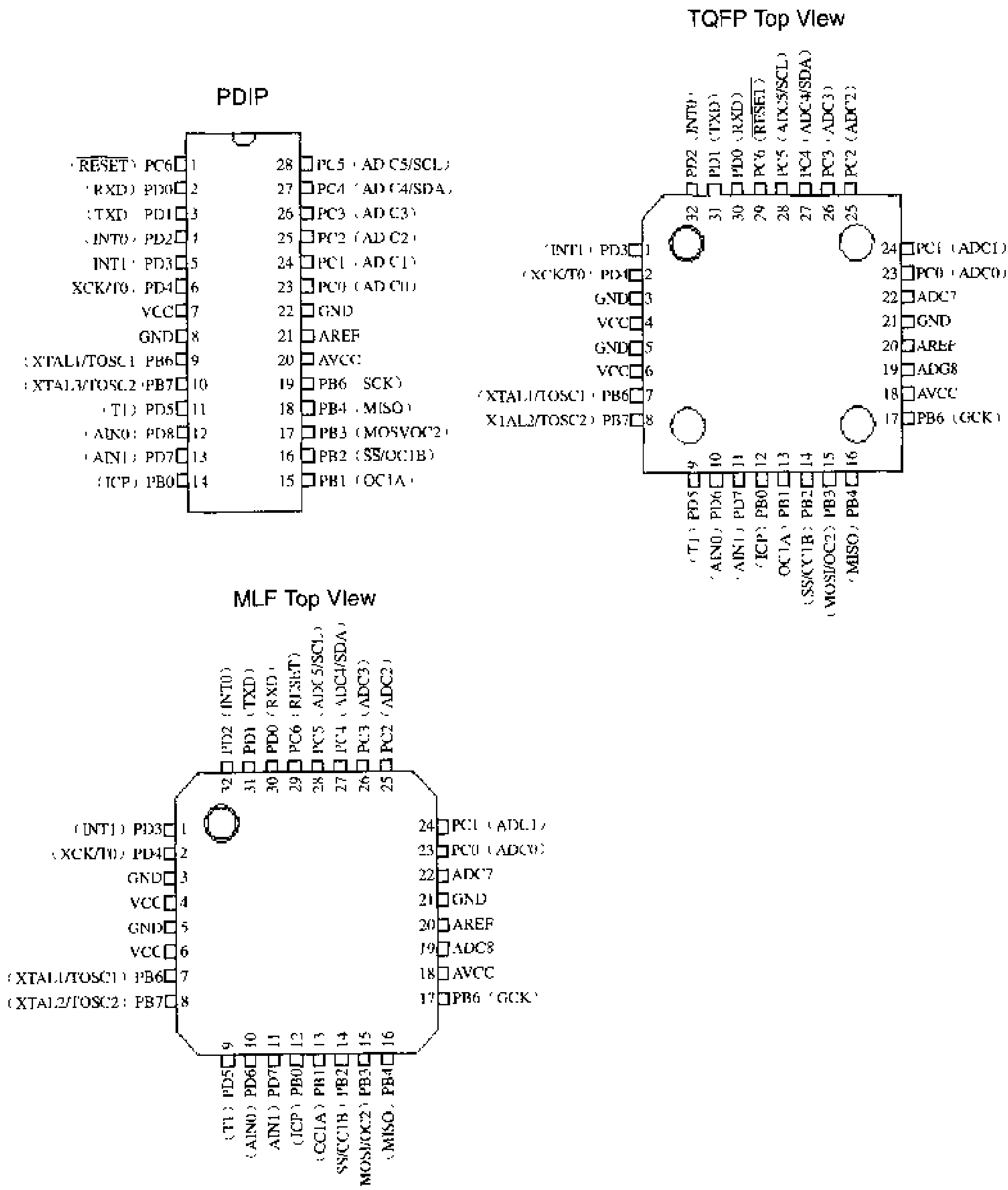


图 1.2　ATmega8 芯片引脚图

ATmega8 的外部引脚定义如下：

VCC 电源正（数字）

GND 电源地

Port B（PB7..PB0）/XTAL1/XTAL2/TOSC1/TOSC2

Port B 口是一个 8 位双向 I/O 口，每一个管脚都带有独立可控制的内部上拉电阻。B 口的输出缓冲器具有双向（输出和吸收）大电流的驱动能力。当 B 口为输入方式，且内部上拉电阻有效时，如果外部引脚被拉低，B 口将输出电流。在复位过程中，即使是在系统时钟还未起振的情况下，B 口仍呈现为三态。

通过对系统时钟选择位的设定，PB6 和 PB7 还可以作为振荡放大器的输入/输出口（使用外部晶体）和外部时钟脉冲信号输入口（PB6）。

当系统使用内部的 RC 振荡器时，通过设置 ASSR 寄存器的 AS2 位，可以将 PB6、PB7 作为异步实时时钟/计数器 2（Asynchronous Timer/Counter2）的输入口 TOSC1、TOSC2 使用（连接 32.768Hz 的时钟晶体）。

Port C（PC5..PC0）

Port C 口是一个 7 位双向 I/O 口，每一个管脚都带有独立可控制的内部上拉电阻。C 口的输出缓冲器具有双向（输出和吸收）大电流的驱动能力。当 C 口为输入方式，且内部上拉电阻有效时，如果外部引脚被拉低，C 口将输出电流。在复位过程中，即使是在系统时钟还未起振的情况下，C 口仍呈现为三态。

PC6/RESET

当 RSTDISBL 位被编程时，可将 PC6 作为一个 I/O 口使用，因此，PC6 引脚与 Port C 口其他引脚的电特性是有区别的。

当 RSTDISBL 位未被编程时，PC6 将作为复位输入引脚 RESET。在该引脚上，一个超过两个时钟周期的低电平将产生复位信号，使系统复位。

Port D（PD7..PD0）

Port D 口是一个 8 位双向 I/O 口，每一个管脚都带有独立可控制的内部上拉电阻。D 口的输出缓冲器具有双向（输出和吸收）大电流的驱动能力。当 D 口为输入方式，且内部上拉电阻有效时，如果外部引脚被拉低，D 口将输出电流。在复位过程中，即使是在系统时钟还未起振的情况下，D 口仍呈现为三态。

Port D 口是一个复用端口，还提供 ATmega8 单片机的许多特殊接口功能。

RESET 复位输入引脚。在该引脚上的一个超过两个时钟周期的低电平将产生复位信号，使系统进入复位过程。

XTAL1 内部振荡放大器的输入端。

XTAL2 内部振荡放大器的输出端。

AVCC

A/D 转换器的电源。当引脚 Port C（0..3）和 Port C（6..7）用于 ADC 时，AVCC 应通

过一个低通滤波器与 VCC 连接。在不使用 ADC 时，该引脚应直接与 VCC 连接。而 Port C（4..5）的电源则是由 VCC 提供的。

AREF A/D 转换器的参考电源输入端。

ADC7..6（TQFP 和 MLF 封装）

采用 TQFP 和 MLF 封装芯片的 ADC7 和 ADC6 引脚为两个 10 位 A/D 转换器的输入口，它们的电压由 AVCC 提供。

有关各个引脚以及特殊功能口的使用方法参考相关章节的内容。

第 2 章　ATmega8 硬件结构

ATMEL 公司推出的 AVR 单片机是一款基于增强 RISC 结构、低功耗、采用 CMOS 技术的 8 位微控制器（Enhanced RISC Microcontroller）。AVR 是一个系列产品，有 ATtiny、AT90 和 ATmega 三种类型，分别对应低、中和高三个档次共 50 多种型号，以满足不同的需求和应用。尽管这些产品的功能和内部配置不同，但基本结构是一样的，指令系统也是兼容的，区别仅在于对 AVR 单片机的内部资源进行了相应的扩展和删减。

在 AVR 家族中，ATmega8 是一种非常特殊的单片机，它的芯片内部集成了较大容量的存储器和丰富强大的硬件接口电路，具备 AVR 高档单片机 MEGE 系列的全部性能和特点。但由于采用了小引脚封装（DIP 28 和 TQFP/MLF32），所以其价格仅与低档单片机相当，使其成为一款具有极高性能价格比的单片机。

ATmega 系列单片机属于 AVR 中的高档产品，它承袭了 AT90 所具有的特点，并在 AT90（如 AT90S8515、AT90S8535）的基础上增加了更多的、功能更加完善的接口功能，而且在省电、稳定性、抗干扰性以及灵活性方面也考虑得更加周全和完善。因此，ATmega 系列单片机的硬件结构比通常的单片机要复杂得多。

目前在国内尚无对 ATmega 系列单片机的结构和使用进行详细介绍的资料和书籍。为了使用户能够全面地了解和更好地使用 ATmega8，本章将对其硬件结构做比较详细的介绍和说明。ATmega8 属于 ATmega 系列单片机（ATmega16/32/64/128）的一个子集，指令系统完全兼容，如果用户能够全面掌握 ATmega8 的硬件结构、特性以及使用，那么同时也就了解和掌握了其他型号 AVR 单片机，特别是 ATmega 系列单片机的硬件结构。

2.1　ATmega8 MCU 内核

2.1.1　结构概述

为了提高 MCU 并行处理的运行效率，AVR 单片机采用了程序存储器和数据存储器使用不同的存储空间和存取总线的 Harvard 结构。算术逻辑单元（ALU）使用单级流水线操作方式对程序存储器进行访问，在执行当前一条指令的同时，也完成了从程序存储器中取出下一条将要执行指令的操作，因此执行一条指令仅需要一个时钟周期。图 2.1 为 AVR 单片机的系统结构图。

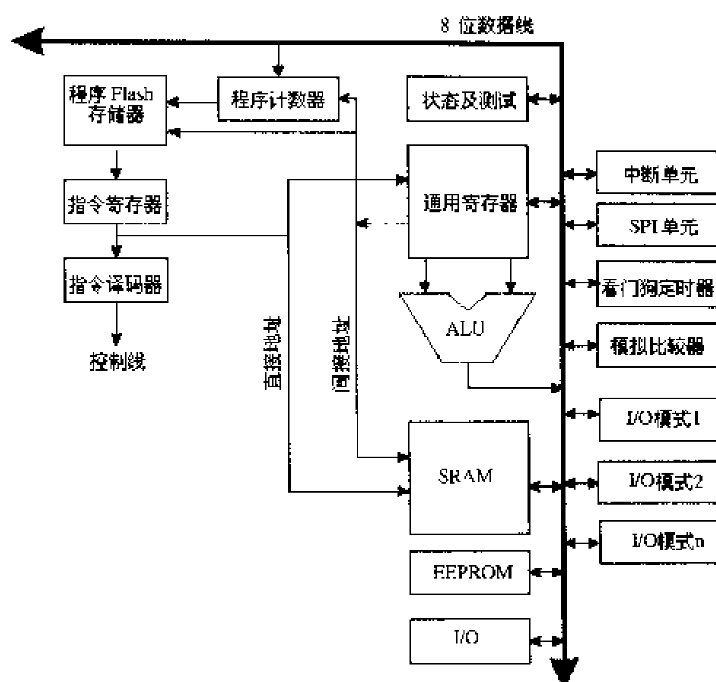


图 2.1　AVR 单片机的系统结构图

在 AVR 硬件内核中，有一个由 32 个访问操作时间只需要一个时钟周期的 8 位通用工作寄存器所组成的“快速访问寄存器组”。“快速访问”意味着在一个时钟周期内执行一个完整的 ALU 操作。在一个标准的 ALU 操作中包含了三个过程：即从寄存器组中取出两个操作数，操作数被执行，将执行结果写入目的寄存器中。这三个过程是在一个时钟周期内完成的，构成一个完整的 ALU 操作。

在 32 个通用工作寄存器中，有 6 个寄存器可以合并成为 3 个 16 位的，用于对数据存储器空间进行间接寻址的间接地址寄存器（存放地址指针），以实现高效的地址计算。这 3 个 16 位的间接地址寄存器称为：X 寄存器，Y 寄存器和 Z 寄存器。其中 Z 寄存器还能作为间接寻址程序存储器空间的地址寄存器，用于在 Flash 程序存储器空间进行查表操作。

算术逻辑单元（ALU）支持寄存器之间，立即数与寄存器之间的算术与逻辑运算功能，ALU 也执行单一的寄存器操作。每一次运算操作的结果将影响和改变状态寄存器的值。

使用条件转移、无条件转移和调用指令，可以直接访问全部 Flash 程序存储器空间。大部分 AVR 指令为单一的 16 位格式，只有少数指令为 32 位格式。因此，AVR 的程序存储器单元为 16 位，即每个地址单元存放一个单一的 16 位指令字，而一条 32 位的指令字则要占据 2 个程序存储器单元。

ATmega8 单片机的 Flash 程序存储器空间分成两段：引导程序段（Boot program section）和应用程序段（Application program section）。两个段的读写保护可以通过设置对应的锁定位（Lock bits）来实现。在引导程序段驻留的引导程序中可以使用 SPM 指令，用以实现对应用程序段的更新写操作（实现可在应用自编程，使更新系统持续）。

在进入中断和子程序调用过程时，程序计数器 PC 的返回地址将被存储于堆栈之中。堆栈空间将占用数据存储器（SRAM）中一段连续的地址，因此，堆栈空间的大小仅受到

系统总的数据存储器（SRAM）的大小以及系统程序对SRAM的使用量的限制。用户程序应在系统上电复位后，对一个16位的堆栈指针寄存器SP进行初始化设置（或在子程序和中断程序被执行之前）。可以使用对I/O寄存器空间进行读写访问的指令对堆栈指针寄存器SP进行操作。

在AVR中，所有的存储器空间都是线性的，数据存储器（SRAM）可以通过5种不同的寻址方式进行访问。

AVR的中断控制由I/O寄存器空间的中断控制寄存器和状态寄存器中的全局中断使能位组成。每个中断都分别对应一个中断向量（中断入口地址）。所有的中断向量构成了中断向量表，该中断向量表位于Flash程序存储器空间的最前面。中断的中断向量地址越小，其中断的优先级越高。

I/O空间为连续的64个I/O寄存器空间，它们分别对应MCU各个外围功能的控制和数据寄存器地址，如控制寄存器、定时器/计数器、A/D转换器及其他的I/O功能等。I/O寄存器空间可使用I/O寄存器访问指令直接访问，也可将其映射为通用工作寄存器组后的数据存储器空间，使用数据存储器访问指令进行访问。I/O寄存器空间在数据存储器空间的映射地址为$0020～$005F。

2.1.2 微控制器（MCU）

1. 算术逻辑单元（ALU）

AVR的算术逻辑单元（ALU）与32个通用工作寄存器直接相连。在一个系统时钟周期内，ALU可以完成一个寄存器与寄存器之间或寄存器与立即数之间的操作。这些ALU的操作分为3类：算术、逻辑以及位操作。

2. 状态寄存器（SREG）

状态寄存器（SREG）的作用很大，每一位都是一个标志位，代表不同的含义。许多指令的运行将对状态寄存器中的一些标志位置位或清零，它反映了每个ALU操作后结果的各种状态。这些标志位将作为程序跳转的判断条件。由于每个操作的结果都会影响和改变状态寄存器标志位的内容，因此在许多情况下，可以省掉使用专用的比较指令，从而使生成的执行代码更加简短、高效和快速。

在进入中断处理过程和中断返回时，状态寄存器的内容不会自动保护和恢复，因此需要在中断处理程序中使用相关的指令对状态寄存器内容进行现场的保护与恢复。

AVR的状态寄存器（SREG）在I/O空间的地址为$3F（数据空间地址为$005F），每一位的定义如下：

- 位7——I：全局中断允许

位	7	6	5	4	3	2	1	0	
\$3F（\$005F）	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	
复位值	0	0	0	0	0	0	0	0	

当全局中断允许 I 位为“1”时，全局中断使能允许。而单独的中断使能允许则由各自的中断控制寄存器控制。如果全局中断允许位清 0，则不论单独中断允许位是否置“1”，所有中断都被禁止，系统不响应任何中断。一旦系统响应一个中断，I 位将由硬件自动清“0”；而当执行 RETI（中断返回）指令时，I 位由硬件自动置位“1”，从而允许系统再次响应下一个中断的请求。用户也可以在程序中使用 SEI 和 CLI 指令对全局中断允许标志位 I 进行置位或清除（如要求系统实现中断嵌套响应）。

- 位 6——T：位复制存储

位复制指令 BLD（SREG 中 T 标志复制到寄存器某位）和 BST（寄存器某位复制到 SREG 中 T 标志）把位复制存储 T 位作为源操作位或目标操作位。32 个工作寄存器组中的任何一个寄存器的其中一位，可以通过 BST 指令被复制到标志 T；而使用 BLD 指令则可将 T 的值复制到 32 个工作寄存器组中指定寄存器的指定位。

- 位 5——H：半进位标志位

半进位标志位 H 指示了在一些 ALU 运算操作过程中有无半进位（低 4 位向高 4 位进位）的出现，它对于 BCD 码的运算是非常有用的。请参考相关的指令说明。

- 位 4——S：符号标志位，$S=N \oplus V$

符号标志 S 位是负数标志位 N 和 2 的补码溢出标志位 V 两者的异或值，请参考相关指令说明。

- 位 3——V：2 的补码溢出标志位

2 的补码溢出标志位 V 用于支持 2 的补码运算，请参考相关指令说明。

- 位 2——N：负数标志位

负数标志位 N 表示在一个算术或逻辑操作之后的结果是否为负数，请参考指令说明。

- 位 1——Z：零值标志位

零值标志位 Z 表示在一个算术或逻辑操作之后的结果是否为零，请参考指令说明。

- 位 0——C：进位标志位

进位标志位 C 表示在一个算术或逻辑操作之后有无产生进位，请参考指令说明。

3．通用工作寄存器组

在 AVR 中，由命名为 R0～R31 的 32 个 8 位通用工作寄存器构成一个“通用工作寄存器组”。图 2.2 为通用工作寄存器组的结构图。采用通用工作寄存器组的结构，使 AVR 单片机的增强 RISC 指令系统得到优化。为了使 ALU 能够高效和灵活地对寄存器组进行访问操作，寄存器组支持 4 种数据输入/输出的方式：

- 提供一个 8 位源操作数并保存一个 8 位结果
- 提供两个 8 位源操作数并保存一个 8 位结果

- 提供两个 8 位源操作数并保存一个 16 位结果
- 提供一个 16 位源操作数并保存一个 16 位结果

大多数的工作寄存器组操作指令都能够在单个时钟周期内直接访问所有的工作寄存器。如图 2.2 所示，每个通用寄存器都占据一个数据存储器 SRAM 空间的地址，它们被直接映射到用户数据空间的前 32 个地址。虽然寄存器组的物理结构与 SRAM 不同，但是这种内存空间的组织方式为访问工作寄存器提供了极大的灵活性，如可以利用地址指针寄存器 X、Y 或 Z 实现对通用寄存器组的间接寻址操作。

位 7......0	地址	
R0	$0000	
R1	$0001	
R2	$0002	
⋮		
R14	$000E	
R15	$000F	
R16	$0010	
⋮		
R26	$001A	X 寄存器低字节
R27	$001B	X 寄存器高字节
R28	$001C	Y 寄存器低字节
R29	$001D	Y 寄存器高字节
R30	$001E	Z 寄存器低字节
R31	$001F	Z 寄存器高字节

图 2.2 通用工作寄存器组

4．X、Y、Z 地址指针寄存器

寄存器 R26~R31 除了可用作通用寄存器外，还可两两合并，组成三个 16 位寄存器 X、Y、Z，作为间接寻址操作中的地址指针寄存器使用，这样就可以在整个数据空间实现间接寻址的操作。在不同的间接寻址方式中，地址指针寄存器中的内容或保持不变，或自动增量加 1，或自动减 1。X、Y、Z 寄存器的结构如图 2.3 所示。

5．堆栈指针寄存器（SP）

堆栈主要用于保存临时数据、局部变量、中断或子程序的返回地址。16 位的堆栈指针寄存器（SP）指示了堆栈顶部地址。需要注意的是，尽管堆栈区可在整个的数据存储器（SRAM）空间建立，但在 AVR 中，堆栈区在数据存储器空间内是由高端向低端发展的。这意味着执行一个进栈 PUSH 操作时，堆栈指针将自动减量（8051 的堆栈区在数据存储器空间内是由低端向高端发展的）。

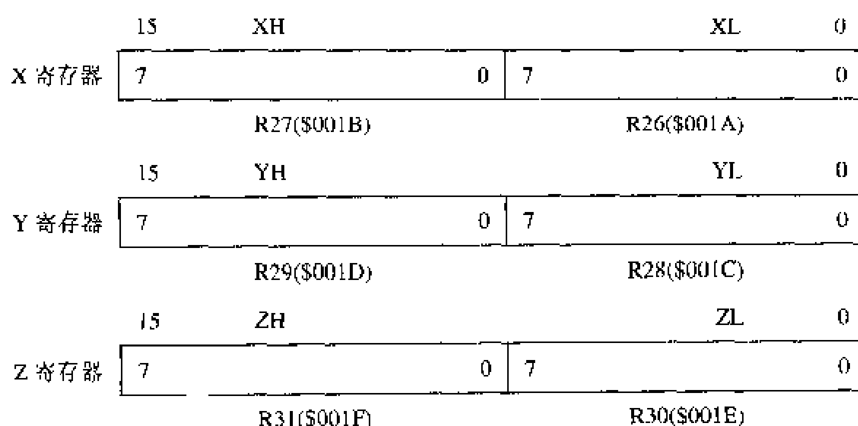


图 2.3　X、Y、Z 寄存器的结构

位	15	14	13	12	11	10	9	8	
$3E（$005E）	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
$3D（$005D）	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
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	
复位值	0	0	0	0	0	0	0	0	
复位值	0	0	0	0	0	0	0	0	

在 I/O 空间，地址为$3E（$005E）和$3D（$005D）的两个 8 位寄存器构成了一个 16 位宽的堆栈指针寄存器 SP。由于 AVR 单片机上电复位后，堆栈寄存器（SP）的初始值为 SPH=$00、SPL=$00，而且堆栈采用减 1 或减 2 的进栈操作，所以系统程序一开始必须对堆栈指针寄存器（SP）进行初始化（或在调用子程序及开放中断以前）。一般应将 SP 的值设在数据存储器（SRAM）空间的最高处。ATmega8 的内部 SRAM 为 1KB，又不能扩充外部的 SRAM，因此，堆栈初始值应设置为 SRAM 空间的$045F 处。

SP 堆栈指针寄存器指示了在 SRAM 空间堆栈区域的头部地址，子程序返回地址和中断返回地址将被放置在堆栈区域中。在数据存储器（SRAM）中，该堆栈空间的头部地址必须在系统程序初始化时由初始化程序定义和设置。当执行 PUSH 指令，数据被压入堆栈时，堆栈指针（SP 中的数据）自动减 1；当执行子程序调用指令 CALL 和中断响应时，指令将自动把返回地址（16 位数据）压入堆栈中，堆栈指针将自动减 2。反之，当执行 POP 指令，将数据从堆栈中弹出时，堆栈指针将自动加 1；当执行从子程序 RET 返回或从中断 RETI 返回指令时，返回地址将从堆栈中弹出，堆栈指针将自动加 2（注意：同 8051 结构的区别）。

2.1.3　MCU 工作时序

ATmega8 由 MCU 时钟信号 clk_{CPU} 驱动，该 clk_{CPU} 时钟信号由选定的系统时钟振荡源直接产生，在内部没有使用时钟分频。

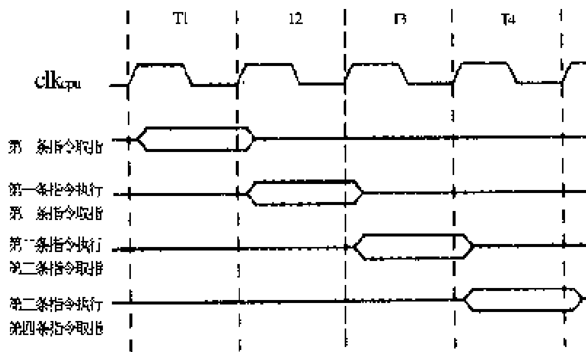


图 2.4 并行指令存取和指令执行

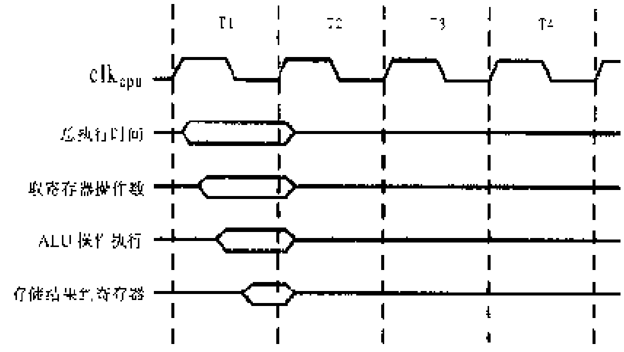


图 2.5 单时钟周期 ALU 操作时序

图 2.4 所示为 Harvard 结构和快速访问寄存器组的并行指令存取和指令执行时序。采用这种流水线结构的目的，是为了获得工作在 1MHz 下，高达 1MIPS / MHz 的效率。MCU 在第一个时钟周期 T1 取出第一条指令，在 T2 周期执行取出的指令，同时又取出第二条指令，依次进行，使 AVR 达到非常高的执行速度。

图 2.5 所示为单时钟周期 ALU 与工作寄存器组操作指令的执行时序。在一个时钟周期内，ALU 从 2 个寄存器中取出 2 个操作数进行相应的运算，并将结果写入目的寄存器。

2.1.4 复位和中断处理

ATmega8 有 18 个不同的中断源，每个中断源和系统复位在程序存储器空间都有一个独立的中断向量，每个中断事件都有各自独立的中断允许控制位，当某个中断源的中断允许位置“1”，且状态寄存器（SREG）中的全局中断允许位 I 也为“1”时，MCU 才能响应该中断。通过对启动锁定位（Boot Lock bits）BLB02 和 BLB12 编程，也能禁止 MCU 响应中断。利用这一特性，可以提高系统的安全性。

通常，Flash 程序存储器空间的最低位置定义为系统复位和中断向量。完整的中断向量见表 2-18。在中断向量表中，处于低地址的中断具有高的优先级，所以，系统复位 RESET 具有最高的优先级。

可以通过对 BOOTRST 熔丝位编程和 GICR 寄存器的 IVSEL 的设置，将系统复位向量和中断向量表置于 Flash 程序存储器的应用程序区的头部，或者转移到引导程序载入区的头部，或分开置于不同的两个区各自的头部。详见相关章节的介绍。

当 MCU 响应一个中断请求后，会自动将全局中断允许位 I 自动清零，此时后续中断的响应被屏蔽。当 MCU 执行中断返回指令 RETI 时，会将全局中断允许位 I 自动置“1”，以允许响应下一个中断。用户可在中断处理程序中将 I 位置“1”，打开中断响应，这样 MCU 就可以再次响应中断，实现了中断嵌套处理（注意与 8051 的不同）。

中断源基本上分为两种类型的中断：一类是事件触发型中断，另一类是条件中断。对于事件触发类中断（如时钟、计数、比较等），一旦事件产生后会将相应的中断标志位置位，申请中断处理。当 MCU 响应中断，跳转到实际中断向量处开始相应的中断处理程序时，

硬件自动清除对应的中断标志。这些中断标志位也可通过软件写“1”来清除。当一个符合条件的中断触发置位了中断标志位，但相应的中断允许位为“0”，此时，这个中断标志将挂起为“1”，一直保持到该中断被响应或中断标志被软件清为“0”。同样的道理，当一个或多个符合条件的中断触发置位了中断标志位，但全局中断允许位为“0”，此时，这个（些）中断标志将挂起保持为“1”，直到全局中断允许位置“1”，MCU根据中断的优先级先后响应中断，或者中断标志被软件清为“0”。

对于一些条件中断（如外部电平中断）来讲，它们没有中断标志位，在中断条件成立时，将一直不断地向MCU申请中断。如果在中断允许响应前，中断条件由成立状态变成不成立状态，该中断即宣告终止了。

AVR的中断响应时间最少为4个时钟周期。硬件系统需要4个时钟周期的时间自动将程序计数器PC的值压进堆栈，同时将堆栈指针（SP）减2，然后跳转到中断向量处，执行在中断向量处的指令。通常情况下，在中断向量处是一条跳转到中断处理程序的Jump指令，执行这条指令需要3个时钟周期。如果中断是在一条多周期指令执行期间发生的，则必须等待当前这条多周期指令执行完后，MCU才能响应中断。如果中断是在MCU处于休眠状态时发生的，此时的中断响应时间为4个时钟周期再加上系统唤醒延时的时间。系统唤醒延时时间的多少取决于MCU的休眠方式和相应设定。

中断返回也需要4个时钟周期。在此期间，硬件系统自动将PC的值从堆栈中弹出，将堆栈指针SP加2，同时将状态寄存器（SREG）的I位置“1”。

MCU由中断返回后，总是回到被中断打断的主程序，并至少执行一条主程序中的指令后才能响应下一个中断。

特别要注意的是，AVR在响应中断及从中断返回时，并不会对状态寄存器SREG自动进行保存和恢复操作，因此，状态寄存器（SREG）的中断保护与恢复必须由用户软件完成。

一旦执行CLI指令将全局中断允许位清零，则中断响应将被立即禁止，即使是与该指令执行同时所产生的中断也被禁止。当执行SEI指令允许全局中断响应后，MCU要再执行一条紧跟在SEI后的指令，才能开始响应中断。

2.2 ATmega8单片机存储器组织

2.2.1 支持可在线编程和可在应用自编程的Flash程序存储器

ATmega8单片机片内集成了8K字节的、支持可在线编程(ISP)和可在应用自编程(IAP)的Flash存储器，用于存放程序指令代码。图2.6为Flash程序存储器的结构图。由于大部分的AVR指令为16位宽(只有少数指令为32位宽)，因此程序存储器Flash的结构为4K×16位。为了程序的安全性，Flash存储器空间被分为两部分：引导程序段(Boot Program Section)

1017159

和应用程序段（Application Program Section）。可以通过对相应熔丝位的编程设定，选择是否需要使用引导程序段以及该段空间的大小。

Flash 存储器的使用寿命最少为 1000 次的擦写循环。ATmega8 的程序计数器（PC）的字长为 12 位宽，可以寻址整个的 4K 程序存储器空间。引导程序的应用以及用于程序保护的相关锁定位，将在引导程序载入支持的章节中详细描述。

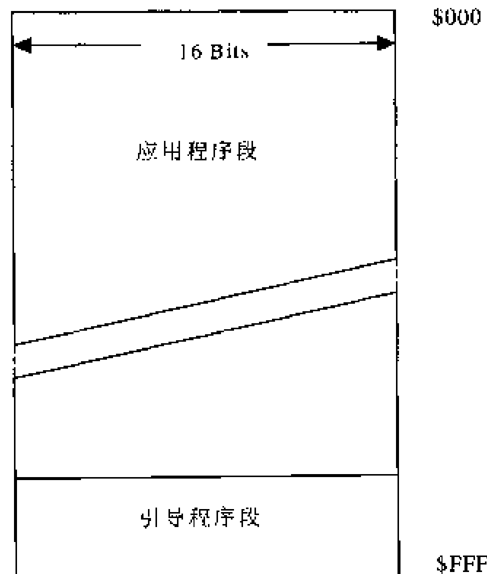


图 2.6 Flash 程序存储器结构

常量表也可以放置在程序存储器空间，请参考 LPM 装载程序存储器指令说明。

2.2.2 数据存储器（SRAM）

图 2.7 给出了 ATmega8 数据存储器（SRAM）空间的组织结构。由低端开始的 1120 个数据存储器空间依次分配给：32 个通用工作寄存器、64 个 I/O 寄存器和 1K 字节的内部 SRAM，即前 96 个地址分配给通用工作寄存器组空间和 I/O 寄存器空间（映射）使用，接下来的 1024 个地址用于内部 SRAM。

数据存储器的寻址模式分为 5 种：直接、带偏移量的间接、间接、带预减量的间接和带后增量的间接寻址。在寄存器组中，寄存器 R26~R31 构成间接寻址的指针寄存器。

数据存储器的直接寻址范围为整个数据存储器空间的 1120 个地址。带偏移量的间接寻址模式能够寻址到由寄存器 Y 和 Z 给定的基址附近的 63 个地址。在自动预减量和后增量的间接寻址模式中，寄存器 X、Y 和 Z 被用为地址指针寄存器，其内容将自动增加或减小。

全部 32 个通用工作寄存器、64 个 I/O 寄存器以及 1024 个字节的数据存储器（SRAM），可以通过上述的寻址模式进行读写访问。

32 个通用工作寄存器和 64 个 I/O 寄存器全部在 SRAM 空间有映射地址，因此可以采用访问 SRAM 的指令对这些寄存器进行操作，此时指令中使用该寄存器在 SRAM 空间的

映射地址。但最好还是使用专用的寄存器访问指令对这些寄存器进行操作，因为这类寄存器专用指令不仅操作功能强大，而且指令执行周期也短。使用专用的寄存器指令访问寄存器时，用寄存器在实际寄存器空间的地址，如访问工作寄存器，地址为 R0～R1；访问 I/O 寄存器，地址空间为$00~$3F。

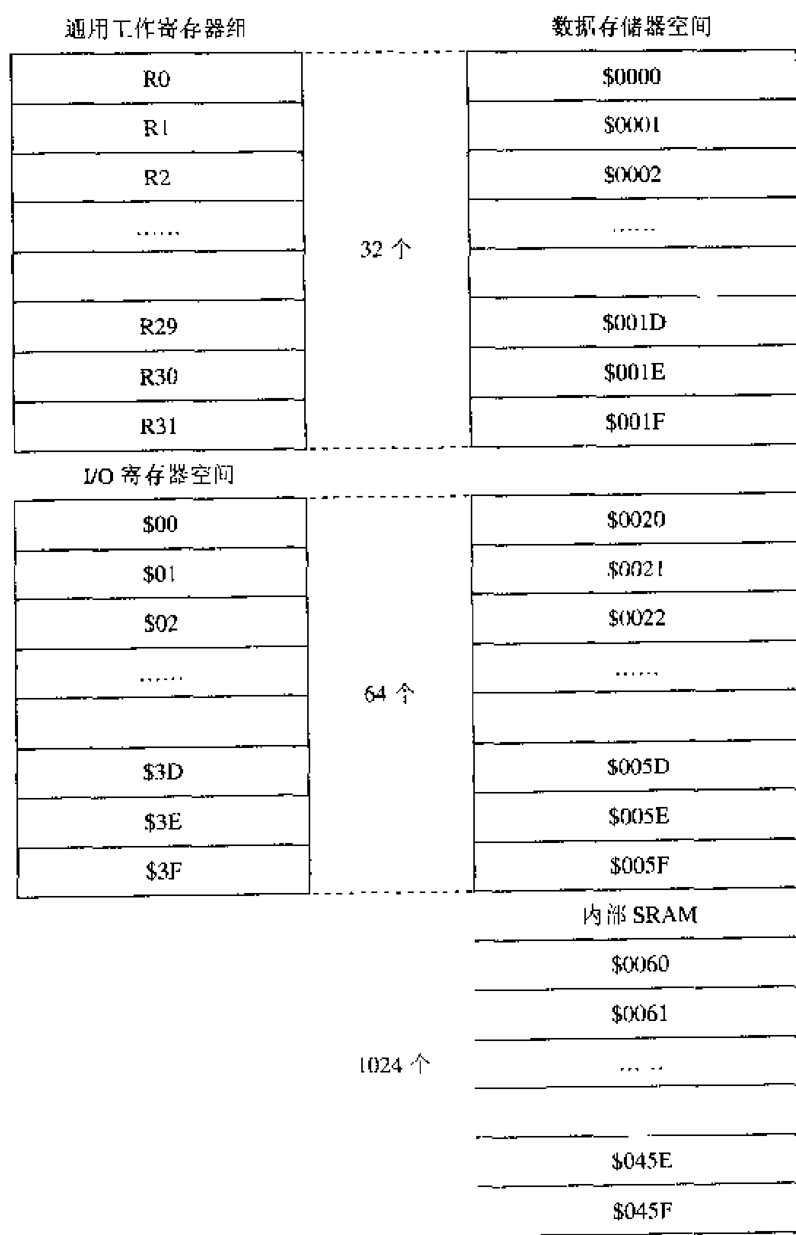


图 2.7　数据存储器空间组织

2.2.3　E²PROM 数据存储器

ATmega8 单片机内部有一个 512 字节的 E²PROM 数据存储器。它们组成一个单独的数据存储器空间，与 Flash 程序存储器空间和 SRAM 数据存储器空间相互独立。E²PROM 数据存储器空间的读写是以单字节为单位的。E²PROM 的使用寿命至少为 100000 次的擦写

循环。对 E²PROM 的访问应用，详见后面对 E²PROM 应用的说明。

2.2.4 I/O 寄存器

表 2-1 列出了 ATmega8 单片机的 I/O 寄存器的地址空间分配、名称和功能。表中不仅给出它们在 I/O 空间的地址，同时在括号中给出它们在 SRAM 空间的映射地址（+$0020）。

表 2-1 ATmega8 I/O 地址空间分配表

十六进制地址	名 称	功 能
$00 （$0020）	TWBR	I²C 波特率寄存器
$01 （$0021）	TWSR	I²C 状态寄存器
$02 （$0022）	TWAR	I²C 从机地址寄存器
$03 （$0023）	TWDR	I²C 数据寄存器
$04 （$0024）	ADCL	ADC 数据寄存器低字节
$05 （$0025）	ADCH	ADC 数据寄存器高字节
$06 （$0026）	ADCSR	ADC 控制和状态寄存器
$07 （$0027）	ADMUX	ADC 多路选择器
$08 （$0028）	ACSR	模拟比较控制和状态寄存器
$09 （$0029）	UBRRL	USART 波特率寄存器低 8 位
$0A （$002A）	UCSRB	USART 控制状态寄存器 B
$0B （$002B）	UCSRA	USART 控制状态寄存器 A
$0C （$002C）	UDR	USART I/O 数据寄存器
$0D （$002D）	SPCR	SPI 控制寄存器
$0E （$002E）	SPSR	SPI 状态寄存器
$0F （$002F）	SPDR	SPI I/O 数据寄存器
$10 （$0030）	PIND	D 口输入脚
$11 （$0031）	DDRD	D 口数据方向寄存器
$12 （$0032）	PORTD	D 口数据寄存器
$13 （$0033）	PINC	C 口输入脚
$14 （$0034）	DDRC	C 口数据方向寄存器
$15 （$0035）	PORTC	C 口数据寄存器
$16 （$0036）	PINB	B 口输入脚
$17 （$0037）	DDRB	B 口数据方向寄存器
$18 （$0038）	PORTB	B 口数据寄存器
$19 （$0039）	（Reserved）	保留
$1A （$003A）	（Reserved）	保留
$1B （$003B）	（Reserved）	保留
$1C （$003C）	EECR	E²PROM 控制寄存器
$1D （$003D）	EEDR	E²PROM 数据寄存器
$1E （$003E）	EEARL	E²PROM 地址寄存器低 8 位

续表

十六进制地址	名　称	功　能
$1F　($003F)	EEARH	E^2PROM 地址寄存器高 8 位
$20　($0040)	UCSRC	USART 控制状态寄存器 C
	UBRRH	USART 波特率寄存器高 4 位
$21　($0041)	WDTCR	看门狗定时器控制寄存器
$22　($0042)	ASSR	异步模式状态寄存器
$23　($0043)	OCR2	T/C2 输出比较寄存器
$24　($0044)	TCNT2	T/C2 计数器（8 位）
$25　($0045)	TCCR2	T/C2 控制寄存器
$26　($0046)	ICR1L	T/C1 输入捕获寄存器低 8 位
$27　($0047)	ICR1H	T/C1 输入捕获寄存器高 8 位
$28　($0048)	OCR1BL	T/C1 输出比较寄存器 B 低 8 位
$29　($0049)	OCR1BH	T/C1 输出比较寄存器 B 高 8 位
$2A　($004A)	OCR1AL	T/C1 输出比较寄存器 A 低 8 位
$2B　($004B)	OCR1AH	T/C1 输出比较寄存器 A 高 8 位
$2C　($004C)	TCNT1L	T/C1 计数器低 8 位
$2D　($004D)	TCNT1H	T/C1 计数器高 8 位
$2E　($004E)	TCCR1B	T/C1 控制寄存器 B
$2F　($004F)	TCCR1A	T/C1 控制寄存器 A
$30　($0050)	SFIOR	特殊功能 I/O 寄存器
$31　($0051)	OSCCAL	RC 振荡器校准值寄存器
$32　($0052)	TCNT0	T/C0 计数器（8 位）
$33　($0053)	TCCR0	T/C0 控制寄存器
$34　($0054)	MCUCSR	MCU 控制和状态寄存器
$35　($0055)	MCUCR	MCU 控制寄存器
$36　($0056)	TWCR	I^2C 总线控制寄存器
$37　($0057)	SPMCR	写程序存储器控制寄存器
$38　($0058)	TIFR	T/C 中断标志寄存器
$39　($0059)	TIMSK	T/C 中断屏蔽寄存器
$3A　($005A)	GIFR	通用中断标志寄存器
$3B　($005B)	GICR	通用中断控制寄存器
$3C　($005C)	(Reserved)	保留
$3D　($005D)	SPL	堆栈指针寄存器低 8 位
$3E　($005E)	SPH	堆栈指针寄存器高 8 位
$3F　($005F)	SREG	状态寄存器

ATmega8 所有的 I/O 及外围设备的控制寄存器和数据寄存器都被放置在 I/O 寄存器空间中。通过 IN 和 OUT 指令可以直接对整个 I/O 寄存器空间的寄存器进行访问，这些指令用于 32 个通用寄存器与 I/O 空间的寄存器之间的数据交换。地址范围在$00~$1F 之间的 I/O

寄存器可通过 SBI（置位 I/O 寄存器的指定位）和 CBI（清零 I/O 寄存器的指定位）指令实现位操作访问，而指令 SBIS 和 SBIC 则用于对这些寄存器的某位进行测试，判别该位是否为“1”或“0”。

使用 I/O 寄存器访问指令 IN（从 I/O 口输入）、OUT（输出到 I/O 口）时，要使用寄存器在 I/O 空间的地址$00~$3F。而使用 LD 和 ST 指令，把 I/O 空间的寄存器作为 SRAM 寻址时，I/O 寄存器的地址要加上$20，即使用 I/O 寄存器在 SRAM 空间的映射地址$0020~$005F。

为了与将来的器件兼容，I/O 寄存器中的保留位应置零，I/O 寄存器空间的保留地址应避免写入操作。

需要说明的是，I/O 寄存器中一些状态标志位的清除是通过写入逻辑“1”实现的，因此对这些特殊的标志位进行操作时千万不要混淆。如用 CBI 和 SBI 指令读取这些特殊的已置位的标志位时，指令会自动回写“1”，因此将会把这些标志位清零。

2.3 系统时钟和时钟选择

2.3.1 时钟系统和时钟分配

图 2.8 为 AVR 的主时钟系统和时钟分配图。并不是所有的时钟信号每时每刻都处于活动状态，为了节约功耗，某些不工作模块的时钟可以通过使用不同的休眠模式来暂停，请参考电源管理和休眠模式的相关章节。

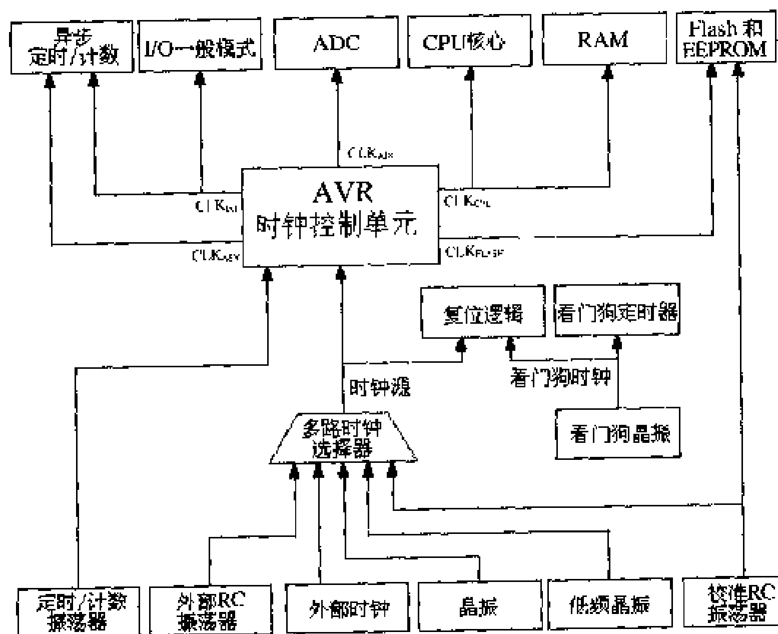


图 2.8 时钟系统和时钟分配

AVR 的主时钟系统将产生以下几种用于驱动芯片各个不同模块的时钟信号，它们是：

- CPU 时钟信号——clk_{CPU}

CPU 时钟信号连接到与 AVR 核心硬件系统操作有关的部分，如通用寄存器组、状态寄存器和数据存储器。暂停 CPU 时钟信号将停止核心系统的通用操作和运算。

- I/O 时钟信号——$clk_{I/O}$

I/O 时钟信号主要用于输入输出模块，如定时器/计数器、SPI 和 USART。I/O 时钟信号还用于外部中断模块，但某些外部中断信号是通过异步逻辑来检测，这样即使在 I/O 时钟信号暂停的情况下，也可以检测到这些中断信号。此外，I^2C 模块的地址识别也是异步进行的，因此，在 I/O 时钟信号暂停，系统处于休眠模式时，I^2C 模块的地址识别也不会停止。

- Flash 时钟信号——clk_{FLASH}

Flash 时钟信号控制着 Flash 接口的操作。当 CPU 时钟信号处于有效状态时，Flash 时钟信号同时也处于有效状态。

- 异步定时器时钟信号——clk_{ASY}

异步定时器时钟信号 clk_{ASY} 用于驱动异步定时器/计数器。即使系统处在休眠模式中，在 clk_{ASY} 的驱动下，异步定时器/计数器仍可作为实时时钟处于工作状态。clk_{ASY} 的产生有两种方式：第一种是外部连接一个 32kHz 晶振（Timer/Counter Oscillator），而由此产生一个外部异步定时器时钟信号；另一种是从片内振荡器引入 32kHz 时钟源，但此时芯片必须是使用片内 RC 振荡器作为系统的时钟源。

- ADC 时钟信号——clk_{ADC}

ADC 转换使用一个专用的时钟信号 clk_{ADC}。这样，就允许在 ADC 转换时暂停 CPU 和 I/O 的时钟，从而可以降低由于数字电路引起的噪声，使 ADC 转换的结果更加精确。

2.3.2　时钟源

通过对 ATmega8 的 Flash 熔丝位 CKSEL 编程设置，器件可选择如表 2-2 所示的 5 种类型的系统时钟源。选定时钟源的脉冲输入到 AVR 内部的时钟发生器，再分配到相应的模块。

表 2-2　时钟源选择

可选系统时钟源	熔丝位 CKSEL3..0[1]
外部晶振	1111-1010
外部低频晶振	1001
外部 RC 振荡	1000-0101
内部 RC 振荡	0100-0001
外部时钟	0000

注：“1”表示熔丝位未编程

“0”表示熔丝位被编程

当 CPU 从掉电（POWER-DOWN）或节电（POWER-SAVE）模式下被唤醒时，系统对选定的时钟源脉冲进行延时计数，经过若干个时钟脉冲后（Start-up Time，可设置选定），再正式启动 CPU 进入工作，这样保证了在 CPU 正式开始执行指令前，振荡器已达到稳定工作状态。

当 CPU 从上电复位启动后到 CPU 开始正常操作指令前，也有额外的延时，以保证系统电源达到稳定的电平。看门狗振荡器（Watchdog Oscillator）被用作该启动延时的定时器。这个 WDT 振荡器启动延时的时间周期见表 2-3。看门狗振荡器的频率由系统电源的电压决定。芯片出厂时，熔丝位的设置为：CKSEL= “0001”，SUT= “01”（使用 1MHz 内部 RC 振荡器，慢速率上升电源）。

表 2-3　WDT 典型延时启动时间和脉冲数

典型延时时间		延时脉冲个数
Vcc=5.0V	Vcc=3.0V	
4.1ms	4.3ms	4K（4096）
65ms	69ms	64K（65536）

2.3.3　外部晶振

ATmega8 的 XTAL1 和 XTAL2 引脚分别是片内振荡器的反相放大器输入、输出端，可在外部连接一个石英晶体或陶瓷振荡器，组成如图 2.9 所示的系统时钟源。熔丝位 CKOPT 用于选择两种不同振荡器的工作方式。当 CKOPT 被编程时，振荡器的输出为一个满幅（rail-to-rail）的振荡信号。对于系统能够适合在高噪声环境工作，或需要把 XTAL2 的时钟信号作为时钟输出驱动时可以采用此种方式。该方式有较宽的工作频率范围。当熔丝位 CKOPT 未被编程时，振荡器输出一个较小摆幅的振荡信号，此时相应地减少了功率消耗。但此方式的工作频率范围受限，振荡器的输出不能作为外部时钟驱动使用。

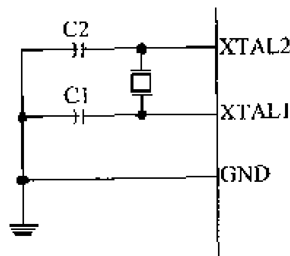


图 2.9　使用外部晶振

外接的陶瓷振荡器，在 CKOPT 未被编程时，最大工作频率为 8MHz，在 CKOPT 被编程时，最大工作频率为 16MHz。无论外接使用的是石英晶体还是陶瓷振荡器，电容 C1 和 C2 的值总是相等的。具体电容值的选择，取决于使用的石英晶体或陶瓷振荡器，以及总的引线电容和环境的电磁噪声等。表 2-4 给出了采用石英晶体时的电容选择参考值。使用陶瓷振荡器时，电容值应采用陶瓷振荡器的生产厂给出的值。

振荡器能够工作在 3 种不同的模式下，它们对特定的工作频率范围进行了优化。工作模式可通过熔丝位 CKSEL3..1 选择，具体情况参见表 2-4。

表 2-4　振荡器的不同工作模式

熔丝位		工作频率范围（MHz）	C1、C2 范围（使用石英晶体）
CKOPT	CKSEL3..1		
1	101	0.4~0.9	仅适合陶瓷振荡器
1	110	0.9~3.0	12~22
1	111	3.0~8.0	12~22
0	101,110,111	≤1.0	12~22

此外，通过对 CKSEL0 熔丝位和 SUT1..0 熔丝位的组合设置，可以选择系统唤醒的延时计数脉冲数和系统复位的延时时间，具体情况见表 2-5。

表 2-5　使用外部晶振时的唤醒脉冲和延时时间的选择设定

熔丝位		掉电和省电模式唤醒	复位延时启动时间（Vcc=5.0V）	适合应用条件
CKSEL 0	SUT 1..0			
0	00	258 CK	4.1ms	陶瓷振荡器快速上升电源
0	01	258 CK	65ms	陶瓷振荡器慢速上升电源
0	10	1K CK	-	陶瓷振荡器 BOD 方式
0	11	1K CK	4.1ms	陶瓷振荡器快速上升电源
1	00	1K CK	65ms	陶瓷振荡器慢速上升电源
1	01	16K CK	-	石英振荡器 BOD 方式
1	10	16K CK	4.1ms	石英振荡器快速上升电源
1	11	16K CK	65ms	石英振荡器慢速上升电源

2.3.4　外部低频率晶振

可以使用外接 32.768kHz 手表用振荡器作为器件的时钟源，此时通过设置熔丝位 CKSEL 为“1001”，来选择使用低频晶体振荡器的工作方式。外部低频晶体振荡器的连接也同图 2.9 所示。通过编程 CKOPT 熔丝位，可以选择使用与 XTAL1 和 XTAL2 连接的芯片内部的电容，此时就没有必要使用外接的电容 C1 和 C2 了。芯片内部的电容值为 36pF。

使用外部低频振荡器时，系统唤醒的延时计数脉冲数和系统复位的延时时间由熔丝位 SUT1..0 的组合确定，具体情况见表 2-6。

表 2-6　使用外部低频晶振时的唤醒脉冲和延时时间的选择设定

熔丝位		掉电和省电模式唤醒	复位延时启动时间（Vcc=5.0V）	适合应用条件
CKSEL 1..0	SUT 1..0			
1001	00	1K CK	4.1ms	快速上升电源或 BOD 方式
1001	01	1K CK	65ms	慢速上升电源
1001	10	32K CK	65ms	唤醒时频率已经稳定
1001	11	保留		

2.3.5　外部 RC 振荡器

对于定时要求不高的应用，可以在外部使用 RC 振荡回路，如图 2.10 所示。其工作频率可以用 f=1/（3RC）公式进行粗略估算。电容 C 至少为 22pF。通过对熔丝位 CKOPT 的编程，可以使用 XTAL1 与地之间的芯片内部 36pF 电容，此时可以省掉外部电容 C。

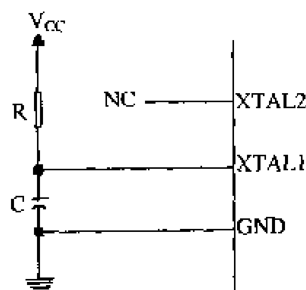


图 2.10　使用外部 RC 振荡器

采用外部 RC 振荡器时也有四种不同的模式，每种方式对特定的频率范围进行了优化。通过对熔丝位 CKSEL3..0 的编程，可以选择使用不同的工作模式，如表 2-7 所示。

表 2-7　使用外部 RC 振荡器的不同工作模式

熔丝位（CKSEL3..0）	工作频率范围（MHz）
0101	≤0.9
0110	0.9~3.0
0111	3.0~8.0
1000	8.0~12.0

使用外部 RC 振荡器时，系统唤醒的延时计数脉冲数和系统复位的延时时间由熔丝位 SUT1..0 的组合确定，具体见表 2-8。

表 2-8　使用外部 RC 振荡器时的唤醒脉冲和延时时间的选择设定

熔　丝　位（SUT 1..0）	掉电和省电模式唤醒	复位延时启动时间（Vcc=5.0V）	适合应用条件
00	18 CK	-	BOD 方式
01	18 CK	4.1ms	快速上升电源
10	18 CK	65ms	慢速上升电源
11	6 CK	4.1ms	快速上升电源或 BOD 方式（工作频率>8MHz 时不建议使用）

2.3.6　可校准的内部 RC 振荡器

在 ATmega8 芯片中集成了可校准的内部 RC 振荡器，它可以提供固定的 1.0、2.0、4.0 或 8.0MHz 时钟信号作为系统时钟源。上述的时钟工作频率是在 5V、25℃条件时的典型值。可以通过对 CKSEL 熔丝位编程而选用内部 RC 振荡器作为系统时钟源，见表 2-9。此时，无需使用外部引脚 XTAL1 和 XTAL2 连接任何的外部元件构成系统时钟源，并且 CKOPT 熔丝位应处在未被编程状态。

表 2-9　使用内部 RC 振荡器的不同工作模式

熔　丝　位（CKSEL3..1）	工作频率（MHz）
0001[1]	1.0
0010	2.0
0011	4.0
0100	8.0

注 1：芯片出厂设置值

系统复位时，硬件将自动把校准字装入 OSCCAL 寄存器，对内部的 RC 振荡器频率进行校准。在 5V，25℃和选择内部 RC 振荡器振荡频率为 1.0MHz 时，通过校准，能使振荡频率达到±1%的精度。当使用内部 RC 振荡器作为芯片的时钟源时，看门狗的振荡器将仍然用作看门狗定时器和复位延时的时钟源使用。

使用内部 RC 振荡器时，系统唤醒的延时计数脉冲数和系统复位的延时时间由熔丝位 SUT1..0 的组合确定，具体见表 2-10。

表 2-10　使用内部 RC 振荡器时的唤醒脉冲和延时时间的选择设定

熔　丝　位（SUT 1..0）	掉电和省电模式唤醒	复位延时启动时间（Vcc=5.0V）	适合应用条件
00	6 CK	-	BOD 方式
01	6 CK	4.1ms	快速上升电源

续表

熔丝位（SUT 1..0）	掉电和省电模式唤醒	复位延时启动时间（Vcc=5.0V）	适合应用条件
10[1]	6 CK	65ms	慢速上升电源
11	保留		

注 1：芯片出厂设置值

振荡器校准寄存器 OSCCAL 的定义如下：

寄存器位 7..0 定义名称为 CAL7..0，用于存放内部 RC 振荡器的校准字。

写入到寄存器 OSCCAL 中的数值，将作为频率校准字用于对内部 RC 振荡器的振荡频率进行调整。系统复位时，位于频率校准标签列的最高字节（0x00）处 1MHz 的校准值将自动由硬件读出并写入到 OSCCAL 寄存器。如果内部 RC 振荡器使用其他的频率，则与频率对应的校准字需要手动装载。可以先使用编程器读取频率校准标签列中与频率对应的相应校准字，再将其写到 Flash 或 E^2PROM 中，然后由系统程序读取这个值，写入到 OSCCAL 寄存器。

当OSCCAL寄存器为零时，选择获得最低的内部RC振荡频率。写非零值到该寄存器将提高内部RC振荡器的振荡频率。写0xFF到该寄存器，则得到最高振荡频率。校准后的内部RC振荡器，也用于对E^2PROM和Flash的访问定时，因此，如果程序要对E^2PROM或Flash写操作，则不要对标称的内部RC振荡频率的校准调整超过10%，否则，对E^2PROM或Flash的写入将会失败。注意，校准只是针对标称振荡频率1.0、2.0、4.0或8.0MHz进行调整，表2-11给出了校准频率的范围。

位	7	6	5	4	3	2	1	0	
$31（$0051）	CAL7	CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0	OSCCAL
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	器件设定值								

表 2-11 内部 RC 振荡器频率范围

OSCCAL 校准字	最低频率（与标称频率的百分比）	最高频率（与标称频率的百分比）
0x00	50%	100%
0x7F	75%	150%
0xFF	100%	200%

2.3.7 外部时钟源

可以使用外部时钟源作为系统时钟，如图 2.11 所示，外部时钟信号应从 XTAL1 输入。将 CKSEL 熔丝位编程为“0000”时，即选定系统使用外部时钟源。通过对熔丝位 CKOPT

的编程，可以使芯片内部 XTAL1 与地之间的 36pF 电容有效。

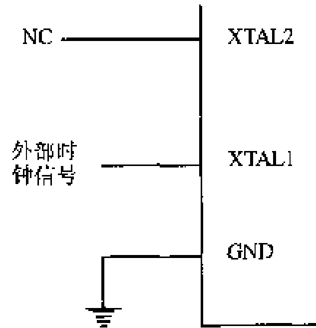


图 2.11　外部时钟源接法

当使用外部时钟源时，系统唤醒的延时计数脉冲数和系统复位的延时时间由熔丝位 SUT1..0 的组合确定，具体见表 2-12。

表 2-12　使用外部时钟源时的唤醒脉冲和延时时间的选择设定

熔丝位（SUT 1..0）	掉电和省电模式唤醒	复位延时启动时间（Vcc=5.0V）	适合应用条件
00	6 CK		BOD 方式
01	6 CK	4.1ms	快速上升电源
10	6 CK	65ms	慢速上升电源
11	保留		

为了保证 MCU 能够稳定工作，不能突然改变外部时钟源的振荡频率，工作频率突然变化超过 2%时，将会产生异常现象。最好是在 MCU 保持复位状态时改变外部时钟的振荡频率。

2.3.8　定时器/计数器振荡器

对于有定时器/计数器振荡器引脚（TOSC1 和 TOSC2）的 AVR 控制器，只需将晶振直接连到这两个引脚上，不需要外部电容。振荡器已对 32.768kHz 手表用的晶振进行了优化。最好不要直接从 TOSC1 引脚输入时钟脉冲信号。

2.4　电源管理和休眠模式

休眠模式意味着在系统程序中关掉 CPU 中未使用的模块，以此来达到省电的目的。AVR 提供了不同的休眠模式，允许用户根据实际应用的需求选定合适的省电休眠模式。

当 MCUCR 寄存器中的 SE 位为逻辑“1”，执行 SLEEP 指令后，MCU 便进入五种休眠模式中的一种。MCUCR 寄存器中的 SM2、SM1 和 SM0 的设置，决定了执行 SLEEP 指令后将进入哪一种休眠模式。这些休眠模式为：空闲（IDLE）、ADC 降噪（ADC Noise Reduction）、掉电（Power-down）、省电（Power-save）和等待（Standby）五种，参见表 2-13。

表 2-13　休眠模式设定

SM2	SM1	SM0	休眠模式
0	0	0	空闲（IDLE）
0	0	1	ADC 降噪（ADC Noise Reduction）
0	1	0	掉电（Power-down）
0	1	1	省电（Power-save）
1	0	0	保留（Reserved）
1	0	1	保留（Reserved）
1	1	0	等待（Standby）[1]

注 1：等待模式仅在使用外部晶体或外部振荡器时有效

单片机处于休眠模式时，一个允许中断信号的产生可把 CPU 从休眠模式中唤醒，这时，在一设定的时钟周期内（长度为：4 个时钟脉冲+唤醒脉冲值 Start-up Time）CPU 将处于暂停状态，以等待时钟振荡进入稳定，然后再转入执行中断处理程序，最后从 SLEEP 指令后的一条指令处重新开始执行程序。单片机从休眠模式唤醒时，寄存器和 SRAM 中的值不会改变。如果单片机处于休眠模式时发生了复位信号，单片机将被唤醒并从复位向量处开始执行程序。

需要注意的是，其他 AVR 系列单片机中的外部等待模式（Extended Standby），在 ATmega8 中由于 TOSC 和 XTAL 输入引脚合并为同一个物理引脚而被取消了。

2.4.1 MCU 控制寄存器 MCUCR

MCU 控制寄存器 MCUCR 包含了电源管理的控制位。

位	7	6	5	4	3	2	1	0	
$35（$0055）	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

- 位 7——SE：休眠使能

当需要执行 SLEEP 指令使 MCU 进入休眠模式时，SE 位必须写入逻辑“1”。为了避免 MCU 在意外情况下进入休眠模式，建议在执行 SLEEP 指令前将休眠使能位 SE 置位。

- 位 6..4——SM2..0：休眠模式选择位 2、1、0

这些位的设置如表 2-13 所示，有五种有效的休眠模式。

2.4.2 空闲模式（IDLE MODE）

当SM2..0被设置为000时，执行SLEEP指令可以使MCU进入空闲模式。此时，MCU停止工作，但SPI、USART、模拟比较器、两线串行接口（即I^2C接口）、定时器/计数器、看门狗和中断系统还是处于工作状态。这种休眠模式主要是暂停了MCU和Flash的时钟信号（即clk_{CPU}和clk_{FLASH}），而其他时钟信号保持运行。

空闲模式可以由外部触发的中断信号或内部中断信号唤醒，如定时器溢出中断、USART传送完成中断等。如果不需要由模拟比较器中断唤醒，可以通过对模拟比较器状态控制寄存器ACSR中的ACD位的设置将模拟比较器关闭，这可以减少空闲模式的功耗。如果模数转换ADC被使能，当进入空闲模式时，将自动启动一次AD转换。

2.4.3 ADC 降噪模式（ADC Noise Reduction）

当SM2..0位被设置为001时，执行SLEEP指令将使MCU进入ADC降噪模式。此时，MCU停止工作，而被使能的ADC外部中断，两线串行接口（即I^2C接口）、定时器/计数器2和看门狗将继续运行。这种休眠模式主要是暂停了MCU、I/O和Flash的时钟信号（即$clk_{I/O}$、clk_{CPU}和clk_{FLASH}）。

ADC降噪模式提高了ADC在噪声环境中的工作能力，使之能应用在高性能的测试中。当模数转换ADC被使能，进入ADC降噪模式时，将自动启动一次AD转换。除了ADC转换完成中断外，只有外部复位、看门狗复位、BROWN-OUT复位、两线接口（I^2C接口）的地址匹配中断、定时器/计数器2中断、SPM/E^2PROM准备好中断和外部电平中断（INT0或INT1）才能将MCU从ADC降噪模式中唤醒。

2.4.4 掉电模式（Power-down）

当SM2..0位被设置为010时，执行SLEEP指令将使MCU进入掉电模式。在这种模式下，外部晶振将停止工作，但外部中断、两线接口（即I^2C接口）地址匹配和看门狗继续工作。只有外部复位、看门狗复位、BROWN-OUT复位、两线接口（即I^2C接口）地址匹配中断和外部的电平中断（INT0和INT1）才能将MCU唤醒。这种休眠模式暂停了所有时钟信号，只有异步模块处于运行中。

如果使用一个外部电平中断触发将MCU从掉电模式唤醒，这个触发电平必须保持一定的时间，才能保证将MCU唤醒。

由掉电模式唤醒时，从唤醒条件成立到真正唤醒之间有一个延时，其目的是让已经停止的时钟重新开始工作，并达到稳定态。该唤醒延时时间是由CKSEL熔丝位定义的唤醒延时计数脉冲数（Start-up Time）来确定。

2.4.5 省电模式（Power-save）

当SM2..0位被设置为011时，执行SLEEP指令将使MCU进入省电模式。该模式等同于掉电模式，其区别仅在于Timer/Counter2工作在异步时钟驱动方式。省电模式暂停了所有时钟信号，只有异步模块处于运行中，而且Timer/Counter2工作在异步时钟驱动方式。

当ASSR寄存器中的AS2位置为“1”时，Timer/Counter2将工作在异步时钟驱动方式。在省电模式下，工作在异步时钟驱动方式的Timer/Counter2将继续运行。如果TIMSK寄存器中的定时器/计数器2中断使能位被置“1”，同时SREG寄存器中的全局中断使能位也被置“1”，那么Timer/Counter2的计数溢出中断或输出比较中断可以把MCU从休眠状态唤醒。

当ASSR寄存器中的AS2位为“0”时，Timer/Counter2不工作在异步时钟驱动方式，那么建议使用掉电模式代替省电模式，因为从省电模式唤醒后，Timer/Counter2寄存器中的值是不确定的。

2.4.6 等待模式（STANDBY MODE）

当SM2..0位被设置为110，并且选择外部石英晶体或陶瓷振荡器作为时钟源时，执行SLEEP指令将使MCU进入等待模式。在该模式下，除了振荡器仍然工作，其他等同于掉电模式。从等待模式唤醒的延时时间为6个时钟周期，如表2-14所示。

表 2-14 不同休眠模式下处在运行状态的时钟和唤醒源

	保持运行的时钟					振荡器		唤醒源					
休眠模式	clk_{CPU}	clk_{FLASH}	$clk_{I/O}$	clk_{ADC}	clk_{ASY}	主振荡器运行	实时时钟振荡运行	INT0 INT1	I^2C 地址匹配	时钟2	SPM/ E^2PROM READY	ADC	其他I/O中断
Idle			√	√	√	√	$√^2$	√	√	√	√	√	√
ADC Noise Redu-ction				√	√	√	$√^2$	$√^3$	√	√	√	√	

续表

休眠模式	保持运行的时钟					振荡器		唤醒源					
	clk_{CPU}	clk_{FLASH}	$clk_{I/O}$	clk_{ADC}	clk_{ASY}	主振荡器运行	实时时钟振荡运行	INT0 INT1	I^2C 地址匹配	时钟 2	SPM/ E^2PROM READY	ADC	其他 I/O 中断
Power Down								√[3]	√				
Power Save					√[2]		√[2]	√[3]	√	√[2]			
Stand-by[1]						√		√[3]	√				

注 1：选择外部石英晶体或陶瓷振荡器作为时钟源时

2：当寄存器 ASSR 的 AS2 位设置为“1”时

3：INT0 和 INT1 的电平中断

2.4.7　如何将功耗降到最低

当试图将 AVR 系统的功耗降到最低时，有几个问题必须考虑。通常来说，应尽可能地使用休眠模式，并且合理地选择休眠模式以使 AVR 中运行的功能模块尽可能的少，因此，所有不使用的功能模块应该禁止其处于运行状态。要试图达到最小的系统功耗，以下几个功能模块需要仔细考虑。

1．ADC 转换

如果 ADC 被使能，那么在所有的休眠模式下它都处在工作状态。为了降低功耗，应该在进入休眠前禁止 ADC。关断 ADC 再重新打开时，第一个 AD 转换是一个预启动转换，转换结果应舍弃。详细的细节请参见 ADC 的相关部分。

2．模拟比较器

进入空闲方式前，如果不使用模拟比较器，应将其关闭。当进入 ADC 降噪模式时，模拟比较器也应该被关闭。在其他休眠模式下，模拟比较器被自动关闭。但是如果模拟比较器被设置成使用内部参考电源，那么在所有的休眠模式中，都应关闭模拟比较器。否则，无论在何种休眠模式下，内部参考电源都将处在工作状态。

3．电压检测（BROWN-OUT）电路

如果在应用中不需要使用 BROWN-OUT 检测电路，则应关闭该检测模块。如果编程 BODEN 熔丝位允许使用 BROWN-OUT 检测电路，那么在所有的休眠模式中该模块都处于

运行状态，也就意味着该模块一直在消耗电能。在深度休眠模式下，它将成为主要的电流消耗源。详细情况请参考 BROWN-OUT 检测电路的相关部分。

4. 内部电压参考源

BROWN-OUT 检测电路、模拟比较器和 ADC 工作时，将使用内部电压参考源。如果这些功能模块被关闭了，内部电压参考源也应关闭，这样可以节省电源消耗。当上述功能模块再次被打开使用时，系统程序应先启动内部电压参考源。如果内部参考电压源在休眠模式时一直处于运行工作状态，其输出可以立即使用。细节请参考内部电压参考源的相关部分。

5. 看门狗定时器

如在应用中不需要使用看门狗，应关闭该模块。如果看门狗被使能，它在所有的休眠模式中都处在运行工作状态，因而一直会消耗电能。在深度的休眠模式下，它将成为主要的电流消耗源。详细情况请参考看门狗定时器的相关部分。

6. 端口引脚

当进入休眠模式时，所有端口的引脚应该设置为最小功耗方式。最主要的是不要使用输出引脚去驱动电阻性的负载。在休眠模式下，由于 I/O 时钟（$clk_{I/O}$）和 ADC 时钟（clk_{ADC}）都被停止了，因此输入缓冲部分也停止了工作，它减少了输入逻辑电路不必要的功率消耗。

在某些情况下，要使用输入引脚来检测唤醒逻辑，那么该输入逻辑电路在休眠模式下应处于工作状态。此时，输入信号的浮动或有一个接近 Vcc/2 的模拟信号，将会使输入缓冲有一个额外的功耗。

2.5 系统复位

在系统上电复位过程中，所有的 I/O 寄存器都被置为它们的初始值，程序从复位中断向量处开始执行。如果程序不使用任何中断源，那么就不需要使用中断向量表，因此可在这些中断向量表的地方可以放置正常的程序代码。图 2.12 的电路框图说明了复位逻辑关系，表 2-15 给出复位电特性参考值。

当任何一个复位信号产生时，AVR 的所有 I/O 端口都会立即复位成它们的初始值，而并不需要时钟源处于运行状态。在复位信号撤消后，硬件系统将调用一个计数延时过程，经过一定的延时后，才进行系统内部真正的复位启动。采用这种形式的复位启动过程保证了电源达到稳定后才使单片机进入正常的操作。复位启动的延时时间可以由用户通过 CSKEL 熔丝位的编程来定义。不同复位启动延时时间的选择参见 2.3 节内容。

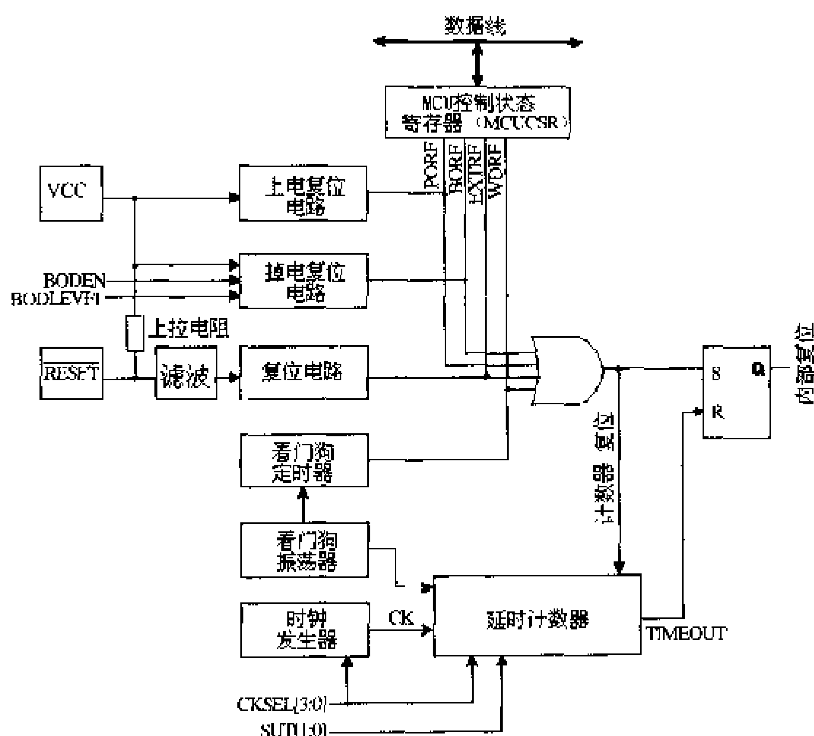


图 2.12　上电复位逻辑结构

表 2-15　系统复位电参数

符　号	参　数	条　件	最 小 值	典 型 值	最 大 值	单　位
VPOT	上电复位门限电压（上升沿）			1.4	2.3	V
	上电复位门限电压（下降沿）			1.3	2.3	V
VRST	复位引脚门限电压		0.1		0.9	Vcc
tRST	复位引脚的最小复位脉冲宽度			50		ns
VBOT	BOD 复位门限电压	BODLEVEL=1	2.5	2.7	3.2	V
		BODLEVEL=0	3.7	4.0	4.5	
tBOD	BOD 检测的低电压最小宽度	BODLEVEL=1		2		μs
		BODLEVEL=0		2		μs
VHYST	BOD 检测迟滞电压			130		mV

2.5.1 复位源

ATmega8 单片机有 4 个复位源

- 上电复位。当系统电源的电平低于上电复位门限电压 V_{POT} 时，MCU 产生的复位。
- 外部复位。当一个低电平加到 RESET 引脚超过 t_{RST} 时，MCU 产生的复位。
- 看门狗复位。当看门狗复位允许且看门狗定时器超时时，MCU 产生的复位。
- 电源电压检测 BOD 复位。当 BROWN-OUT 检测功能允许，且电源电压 V_{CC} 低于 BROWN-OUT 复位门限电压 V_{BOT} 时，MCU 产生的复位。

1．上电复位

上电复位（POR）脉冲由芯片内部的电源检测电路产生，检测电平门限见表 2-14。当 V_{CC} 低于上电复位 V_{BOT} 时，MCU 复位。上电复位电路既用于上电触发复位启动（Internal-Reset），也用于掉电复位启动。当系统上电时，上电复位 POR 电路使 MCU 处于复位状态，复位状态的保持时间为：V_{CC} 上升到 V_{POT} 的时间+启动延时时间。当系统电源电压 V_{CC} 下跌，低于 V_{POT} 时，无需经过任何延时，MCU 立即进入复位状态。图 2.13 和图 2.14 给出了两种不同情况的系统复位-启动的时序。

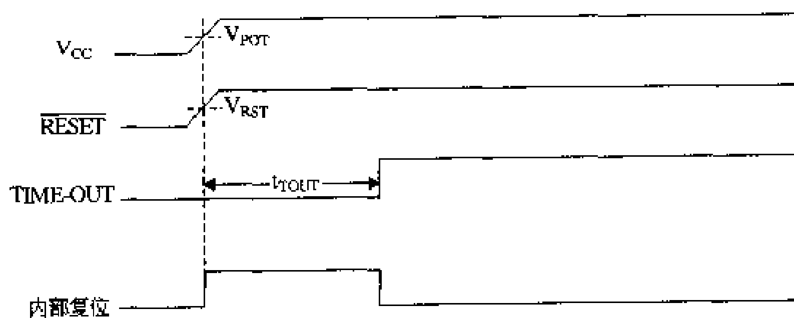


图 2.13　MCU 上电复位（RESET 引脚连接 V_{CC}）

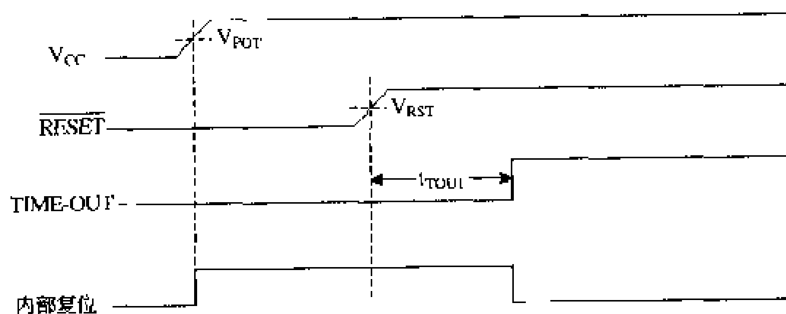


图 2.14　MCU 上电复位（RESET 引脚由外部控制）

2．外部复位

在 RESET 复位引脚上的一个低电平脉冲将产生外部复位。该低电平的宽度至少为 t_{RST}，见表 2-14。当 RESET 引脚上的电平由低变高，达到 V_{RST} 时，再经过设定的启动延时时间，

MCU 启动运行，见图 2.15。

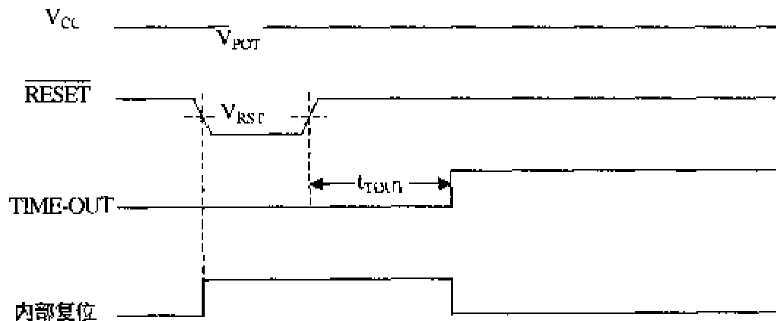


图 2.15　MCU 运行中 RESET 引脚由外部控制复位

3．看门狗复位

当看门狗定时器溢出时，它将触发产生一个时钟周期宽度的复位脉冲，从脉冲的下降开始，经过设定的启动延时时间，MCU 启动运行，见图 2.16。延时时间见表 2-16。

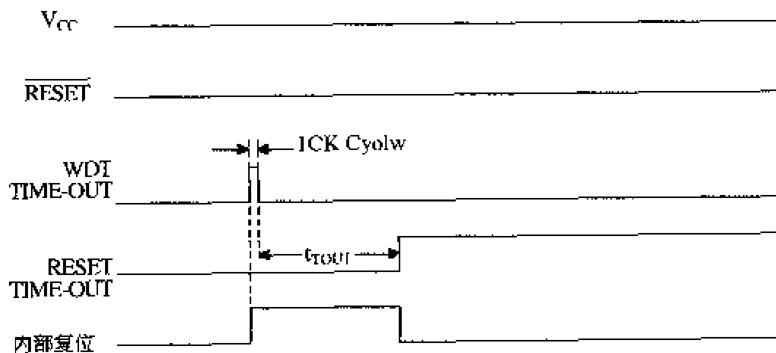


图 2.16　看门狗复位

4．电源电压检测 BOD 复位

ATmega8 有一个片内的 BROWN-OUT 检测电路，用于在系统运行时对系统电压 V_{CC} 的检测，并同一个固定的阈值电压相比较。BOD 检测阈值电压可以通过 BODLEVEL 熔丝位设定为 2.7V 或 4.0V。BOD 检测阈值电压有迟滞效应，以避免系统电源的毛刺误触发 BROWN-OUT 检测器。阈值电平的迟滞效应可以理解为：上阈值电压 $V_{BOT+}=V_{BOT}+V_{HYST}/2$，下阈值电压 $V_{BOT-}=V_{BOT}-V_{HYST}/2$。

BOD 检测电路可以通过编程 BODEN 熔丝位来置成有效或者无效。当 BOD 被置成有效并且 V_{CC} 电压跌到下阈值电压（V_{BOT-}，如图 2.17 所示）以下时，BROWN-OUT 复位立即有效，MCU 进入复位状态。当 V_{CC} 回升，而且超过上阈值电压（V_{BOT+}，如图 2.17 所示）后，再经过设定的启动延时时间，MCU 启动运行。只有当 V_{CC} 电压低于阈值电压并且持续 t_{BOD} 后，BOD 电路才起作用。

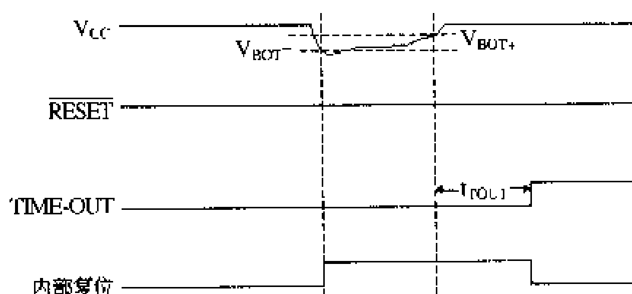


图 2.17 电源电压检测 BOD 复位

2.5.2 MCU 控制和状态寄存器 MCUCSR

MCU 控制和状态寄存器 MCUCSR 提供了系统是由哪一个复位源触发的复位信息。MCU 控制和状态寄存器 MCUCSR 在 I/O 空间的地址为$34（数据空间地址为$0054），每一位的定义如下：

- 位 7..4——Res：保留位

保留位，只读为“0”。

- 位 3——WDRF：看门狗复位标志位

当看门狗复位产生时，该位置“1”。上电复位或使用指令写入“0”时，该位清除。

- 位 2——BORF：Brown-out 复位标志位

当看 BOD 复位产生时，该位置“1”。上电复位或使用指令写入“0”时，该位清除。

- 位 1——EXTRF：外部复位标志位

当外部复位产生时，该位置“1”。上电复位或使用指令写入“0”时，该位清除。

- 位 0——PORF：上电复位标志位

位	7	6	5	4	3	2	1	0	
$34($0054)	-	-	-	-	WDRF	BORF	EXTRF	PORF	MCUCSR
读/写	R	R	R	R	R/W	R/W	R/W	R/W	
复位值	0	0	0	0		见说明			

0

0

0

当上电复位产生时，该位置“1”。使用指令写入“0”时，该位清除。

MCU 复位启动后，用户可以通过程序指令来测试这些复位标志位，以了解系统是由于哪种情况产生的复位。同时，当系统启动后，应及时将复位标志位清除。

2.5.3　内部参考电压源

ATmega8 内部有一个集成的参考电压源。该内部参考电压用于 BROWN-OUT 检测，也可作为模拟比较器的一个输入参考电压。内部参考电压源还产生一个 2.56V 的参考电压用于 ADC 转换。

出于省电的考虑，内部参考电压源一般处于关闭状态。在下面三种情况下，内部参考电源才开启工作：

1. 允许 BOD（熔丝位 BODEN 被编程）。
2. 连接到模拟比较器，作为一个输入参考电压（置寄存器 ACSR 的 ACBG 位为“1”）。
3. ADC 转换允许。

内部参考电压源开启后，需要一定的延时才能达到稳定。当不使用 BOD（禁止 BOD）时，在置位 ACBG 或允许 ADC 后，要等待一段延时时间 t_{BG} 后才能获得有效的模拟比较结果以及 ADC 的转换结果。表 2-16 为内部参考电压源的特性。

表 2-16　内部参考电压源特性

符　号	参　数	最 小 值	典 型 值	最 大 值	单　位
V_{BG}	参考电压	1.15	1.23	1.40	V
t_{BG}	启动稳定时间	1	40	70	μs
I_{BG}	电流消耗		10		μA

2.5.4　看门狗定时器

看门狗定时器由片内一个独立的振荡器驱动。在 Vcc=5V 时，典型的振荡频率为 1MHz。通过设置看门狗定时器的预分频器，可以改变看门狗的复位间隔。WDR 是看门狗计数器清零指令，其作用是对看门狗定时器进行清零。当看门狗被禁止或 MCU 复位时，看门狗也被清零。若看门狗定时器定时时间到，而没再执行看门狗定时器清零指令 WDR，单片机将进入复位状态。复位启动后，从系统复位向量开始执行程序。

用户程序必须按照一个特定的关断顺序来禁止看门狗定时器，以防止意外地关闭看门狗定时器。图 2.18 为看门狗定时器框图。

1. 看门狗定时器控制寄存器-WDTCR

看门狗定时器控制寄存器 WDTCR 用于对看门狗定时器进行控制和设定，每一位的定义如下：

位	7	6	5	4	3	2	1	0	
$21（$0041）	-	-	-	WDCE	WDE	WDP2	WDP1	WDP0	WDTCR
读/写	R	R	R	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

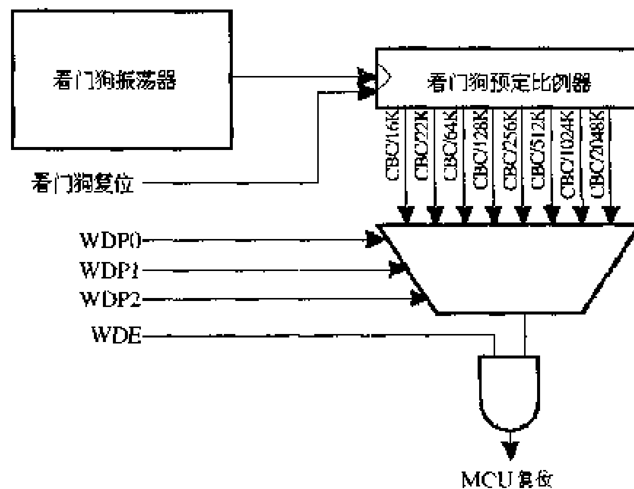


图 2.18　看门狗定时器

- 位 7..5——Res：保留位

保留位，只读为“0”。

- 位 4——WDCE：看门狗定时器改变允许标志位

当要禁止看门狗定时器时，该位必须置“1”，否则，看门狗将不会被禁止。一旦 WDCE 置“1”后，硬件在 4 个时钟周期后自动将该位清零。当需要重新设定看门狗定时器的预置分频器参数时，WDCE 也必须先置“1”。

- 位 3——WDE：看门狗允许标志位

当 WDE 位置“1”，则使能看门狗定时器。WDE 为“0”时，看门狗定时器功能被禁止。清零 WDE 的操作，必须在 WDCE 置“1”后的 4 个时钟周期内完成，因此，如要禁止看门狗，必须按照以下特定的关断操作顺序，以防止意外地关闭看门狗定时器：

①在同一个操作中，把 WDCE 和 WDE 置“1”，即使 WDE 原先已经为“1”，也必须对 WDE 写“1”。

②在随后的 4 个时钟周期内，清零 WDE，禁止看门狗定时器。

- 位 2..0——WDP2、WDP1、WDP0：看门狗定时器预分频器设置位

WDP2、WDP1、WDP0 为看门狗定时器预分频器设置位，用于设定看门狗的复位时间间隔，见表 2-17。

表 2-17　看门狗定时器预分频选择

WDP2	WDP1	WDP0	WDT 脉冲数	典型溢出时间 Vcc=3.0V	典型溢出时间 Vcc=5.0V
0	0	0	16K（16,384）	17.1ms	16.3ms
0	0	1	32K（32,768）	34.3ms	32.5ms
0	1	0	64K（65,536）	68.5ms	65ms
0	1	1	128K（131,072）	0.14s	0.13s
1	0	0	256K（262,144）	0.27s	0.26s
1	0	1	512K（524,288）	0.55s	0.52s

续表

WDP2	WDP1	WDP0	WDT 脉冲数	典型溢出时间 Vcc=3.0V	典型溢出时间 Vcc=5.0V
1	1	0	1,024K（1,048,576）	1.1s	1.0s
1	1	1	2,048K（2,097,152）	2.2s	2.1s

下面两段汇编和 C 语言的子程序代码给出了禁止看门狗定时器的特定关断操作的顺序。这里假定全局中断响应已经关闭。

汇编代码

```
WDT_off:
; Write logical one to WDCE and WDE
ldi r16, (1<<WDCE)|(1<<WDE)
out WDTCR, r16
; Turn off WDT
ldi r16, (0<<WDE)
out WDTCR, r16
ret
```

C 语言代码

```
void WDT_off(void)
{
/* Write logical one to WDCE and WDE */
WDTCR = (1<<WDCE)|(1<<WDE);
/* Turn off WDT */
WDTCR = (0<<WDE);
}
```

2．看门狗定时器安全级别

用户可通过对 ATmega8 熔丝位 WDTON 的编程选择 MCU 运行的安全级别：WDTON 未编程，为安全级别 1（Safety Level 1）；编程 WDTON，设置为安全级别 2（Safety Level 2）。

在安全级别 1 的模式下，MCU 复位启动初始化总是禁止看门狗定时器的。系统程序可以在任何时候将“1”写入 WDT 位，使能看门狗定时器。如要再次禁止看门狗或改变看门狗定时器的复位间隔时间，必须按照以下的操作顺序：

（1）在同一个操作中，把 WDCE 和 WDE 置“1”。即使 WDE 原先已经为“1”，也必须对 WDE 写“1”。

（2）在随后的 4 个时钟周期内，使用一个指令，对 WDE 位和各个 WDP 位以及 WDCE 位同时执行写入操作，即只能使用寄存器写入指令，对 WDTCR 寄存器进行写操作。在写

入数据中，WDE 和 WDP 的值为用户需要的设定值，但 WDCE 的值必须为“0”。

在安全级别 2 的模式下，WDT 位恒为“1”，看门狗定时器总是处在使能态。如要改变看门狗定时器的复位间隔时间，必须按照以下的操作顺序进行：

（1）在同一个操作中，把 WDCE 和 WDE 置“1”。即使 WDE 位恒为“1”，也必须对 WDE 写“1”。

（2）在随后的 4 个时钟周期内，使用一个指令，对 WDE 位和各个 WDP 位以及 WDCE 位同时执行写入操作，即只能使用寄存器写入指令，对 WDTCR 寄存器进行写操作。在写入数据中，WDP 的值为用户需要的设定值，但 WDCE 的值必须为“0”，而 WDE 可以是任意的值。

2.6 中断向量

ATmega8 有 18 个中断源。Flash 程序存储器空间的最低位置（0x000－0x012）定义为复位和中断向量空间。完整的中断向量见表 2-18。在中断向量表中，处于低地址的中断向量所对应的中断拥有高优先级，所以，系统复位 RESET 拥有最高优先级。

表 2-18 复位和中断向量表

中断向量号	向量地址	中断源	中断定义
1	0x000	RESET	上电、外部、BOD、看门狗复位
2	0x001	INT0	外部中断请求 0
3	0x002	INT1	外部中断请求 1
4	0x003	TIMER2 COMP	时钟/计数器 2 比较匹配
5	0x004	TIMER2 OVF	时钟/计数器 2 溢出
6	0x005	TIMER1 CAPT	时钟/计数器 2 捕获事件
7	0x006	TIMER1 COMPA	时钟/计数器 1 比较匹配 A
8	0x007	TIMER1 COMPB	时钟/计数器 1 比较匹配 B
9	0x008	TIMER1 OVF	时钟/计数器 1 溢出
10	0x009	TIMER0 OVF	时钟/计数器 0 溢出
11	0x00A	SPI，STC	SPI 串行传输完成
12	0x00B	USART，RCX	USART，Rx 完成
13	0x00C	USART，UDRE	USART，寄存器空
14	0x00D	USART，TXC	USART，Tx 完成
15	0x00E	ADC	ADC 转换完成
16	0x00F	EE_RDY	E^2PROM 准备好

续表

中断向量号	向 量 地 址	中 断 源	中 断 定 义
17	0x010	ANA_COMP	模拟比较
18	0x011	TWI	两线串行接口（I²C）
19	0x012	SPM_RDY	写程序存储器准备好

注 1：当 BOOTRST 熔丝位被编程时，器件在复位后将跳到引导载入区的起始地址处开始执行

2：当 GICR 寄存器的 IVSEL 位置"1"时，中断向量表将移到引导载入区的头部

每个中断向量的地址是该向量在向量表中的地址加上引导载入区的起始地址值

2.6.1 复位和中断向量表的移动

对于 ATmega8，可以通过对 BOOTRST 熔丝位编程和 GICR 寄存器的 IVSEL 标志位的设置将系统复位与中断向量置于 Flash 程序存储器的应用程序区（Application Section）的头部，或者引导程序载入区（Application Section）的头部，或分开置于不同的两个区各自的头部。表 2-19 给出复位地址和中断向量表在 BOOTRST 熔丝位和 IVSEL 位的不同组合设置下的不同位置。如果程序不使用任何的中断，也就是说中断向量表没有使用，那么正常的程序也能被放在这个地址单元内。

表 2-19　复位和中断向量表的位置

BOOTRST	IVSEL	RESET 复位起始地址	中断向量表起始地址
1（未编程）	0	0x000	0x001
1（未编程）	1	0x000	引导载入区起始地址+0x001
0（编程）	0	引导载入区起始地址	0x001
0（编程）	1	引导载入区起始地址	引导载入区起始地址+0x001

在通常情况下使用 ATmega8 时（BOOTRST=1，IVSEL=0），设置中断向量地址最典型的方法如下面例程所示。

```
地 址   标 号   代 码                解 释
$000            rjmp RESET           ;复位处理
$001            rjmp EXT_INT0        ;IRQ0 Handler
$002            rjmp EXT_INT1        ;IRQ1 Handler
$003            rjmp TIM2_COMP       ;Timer2 Compare Handler
$004            rjmp TIM2_OVF        ;Timer2 Overflow Handler
$005            rjmp TIM1_CAPT       ;Timer1 Capture Handler
$006            rjmp TIM1_COMPA      ;Timer1 CompareA Handler
$007            rjmp TIM1_COMPB      ;Timer1 CompareB Handler
$008            rjmp TIM1_OVF        ;Timer1 Overflow Handler
$009            rjmp TIM0_OVF        ;Timer0 Overflow Handler
```

```
$00a            rjmp SPI_STC             ;SPI Transfer Complete Handler
$00b            rjmp USART_RXC           ;USART RX Complete Handler
$00c            rjmp USART_UDRE          ;UDR Empty Handler
$00d            rjmp USART_TXC           ;USART TX Complete Handler
$00e            rjmp ADC                 ;ADC Conversion Complete Handler
$00f            rjmp EE_RDY              ;E²PROM Ready Handler
$010            rjmp ANA_COMP            ;Analog Comparator Handler
$011            rjmp TWSI                ;Two-wire Serial Interface Handler
$012            rjmp SPM_RDY             ;Store Program Memory Ready Handler
;
$013   RESET:   ldi  r16,high(RAMEND)    ;主程序开始
$014            out  SPH,r16             ;设置堆栈指针于 SRAM 高端(高 8 位)
$015            ldi  r16,low(RAMEND)
$016            out  SPL,r16             ;设置堆栈指针于 SRAM 高端(低 8 位)
$017            sei                      ;开全局中断
$018            <指令> XXX
...             ...
```

当 BOOTRST 熔丝位未被编程且程序存储器空间高端 2K 字节定义为引导程序载入区时，在所有的中断被使能前把 GICR 寄存器中的 IVSEL 位置为“1”后（BOOTRST=1，IVSEL=1），复位和中断向量将分开置于不同的两个区各自的头部：

```
地 址   标 号    代 码                    解 释
$000            rjmp RESET               ;复位处理
;
$001   RESET:   ldi  r16,high(RAMEND)    ;主程序开始
$002            out  SPH,r16             ;设置堆栈指针于 SRAM 高端(高 8 位)
$003            ldi  r16,low(RAMEND)
$004            out  SPL,r16             ;设置堆栈指针于 SRAM 高端(低 8 位)
$005            sei                      ;开全局中断
$006            <指令> XXX
;...            ...
.ORG $C01
$C01            rjmp EXT_INT0            ;IRQ0 Handler
$C02            rjmp EXT_INT1            ;IRQ1 Handler
...             ...
$C12            rjmp SPM_RDY             ;Store Program Memory Ready Handler
```

当 BOOTRST 熔丝位被编程且程序存储器空间高端 2K 字节定义为引导程序载入区时（BOOTRST=0，IVSEL=0），复位和中断向量的位置为：

```
地 址      标 号      代 码                        解 释
.ORG $001
$001                  rjmp EXT_INT0                ;IRQ0 Handler
$002                  rjmp EXT_INT1                ;IRQ1 Handler
...                   ...
$012                  rjmp SPM_RDY                 ;Store Program Memory Ready Handler
;
.ORG $C00
$C00                  rjmp RESET
;
$C01      RESET:      ldi r16,high(RAMEND)         ;主程序开始
$C02                  out SPH,r16                  ;设置堆栈指针于 SRAM 高端(高 8 位)
$C03                  ldi r16,low(RAMEND)
$C04                  out SPL,r16                  ;设置堆栈指针于 SRAM 高端(低 8 位)
$C05                  sei                          ;开全局中断
$C06                  <指令> XXX
```

当 BOOTRST 熔丝位被编程且程序存储器空间高端 2K 字节定义为引导程序载入区时，在所有的中断被使能前把 GICR 寄存器中的 IVSEL 位置为“1”后(BOOTRST=0，IVSEL=1)，复位和中断向量的位置为：

```
地 址      标 号      代 码                        解 释
.ORG $C00
$C00                  rjmp RESET                   ;复位处理
$C01                  rjmp EXT_INT0                ;IRQ0 Handler
$C02                  rjmp EXT_INT1                ;IRQ1 Handler
...                   ...
$C12                  rjmp SPM_RDY                 ;Store Program Memory Ready Handler
$C13      RESET:      ldi r16,high(RAMEND)         ;主程序开始
$C14                  out SPH,r16                  ;设置堆栈指针于 SRAM 高端(高 8 位)
$C15                  ldi r16,low(RAMEND)
$C16                  out SPL,r16                  ;设置堆栈指针于 SRAM 高端(低 8 位)
$C17                  sei                          ;开全局中断
$C18                  <指令> XXX
;...                  ...
```

2.6.2 中断控制寄存器 GICR

在通用中断寄存器 GICR 中，IVSEL 位和 IVCE 位用于控制中断向量位置的移动，通用中断寄存器每一位定义如下。

位	7	6	5	4	3	2	1	0	
$3B（$005B）	INT1	INT0	-	-	-	-	IVSEL	IVCE	GICR
读/写	R/W	R/W	R	R	R	R	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

● 位 1——IVSEL：中断向量表选择

当 IVSEL 位被清为零时，中断向量的位置定义在 Flash 存储器的起始处。当该位被置为“1”时，中断向量的位置定义在引导程序载入区的起始处。引导程序载入区在 Flash 空间的位置和大小由 BOOTSZ 熔丝位决定。请参考引导程序载入应用的相关部分。

为防止移动中断向量表位置的误操作，必须遵守一个特殊的写入规程来改变 IVSEL 位的值。

首先，置中断向量表移位使能位 IVCE 为“1”。

然后，必须在 4 个时钟周期内将需要的值写入 IVSEL 位。在写入 IVSEL 位的同时 IVCE 位将由硬件自动清零。

在这个执行过程中，中断将自动被屏蔽。中断在 IVCE 位被置“1”的指令周期中将被屏蔽，并保持屏蔽状态，直到写 IVSEL 位的指令被执行。如果 IVSEL 位没有被写入，中断屏蔽将被保持 4 个时钟周期。状态寄存器中的 I 位不受这种自动屏蔽的影响。

需要注意的是，如果中断向量表被放置在引导载入程序区中，且引导锁定熔丝位 BLB02 被编程，那么，当执行应用程序区中的程序时，中断将被屏蔽。如中断向量表被放置在应用程序区，且引导锁定熔丝位 BLB12 被编程，那么，在执行引导载入程序区的程序时，中断也被屏蔽。请参考“引导载入支持”相应部分，以了解引导锁定位的具体介绍。

● 位 0——IVCE：中断向量表移位允许位

IVCE 位必须被写入“1”才能允许对 IVSEL 位的更改。在 4 个时钟周期后，或 IVSEL 位写入后，IVCE 位由硬件自动清零。置位 IVCEL，将屏蔽中断，参见对 IVSEL 位的描述。下面是移动中断向量表的汇编和 C 语言的程序示例：

汇编代码

```
Move_interrupts:
; Enable change of interrupt vectors
ldi r16, (1<<IVCE)
out GICR, r16
; Move interrupts to boot Flash section
ldi r16, (1<<IVSEL)
out GICR, r16
ret
```

C 语言代码

```
void Move_interrupts(void)
{
/* Enable change of interrupt vectors */
GICR = (1<<IVCE);
/* Move interrupts to boot Flash section */
GICR = (1<<IVSEL);
}
```

2.7 I/O端口

AVR的I/O端口作为通用数字输入/输出口使用时，都具备真正的读-修改-写（Read-Modify-Write）特性。这意味着用SBI或CBI指令可以单独改变某个I/O引脚的输入/输出方向，或改变引脚的输出值，或在禁止/允许引脚的内部上拉电阻功能时不影响和改变其他引脚。每个I/O引脚采用推挽式驱动，不仅能提供大电流的输出驱动，而且也可以吸收20mA的电流，因此能直接驱动LED显示器。AVR采用3个8位寄存器来控制I/O端口，它们分别是方向寄存器DDRx，数据寄存器PORTx和输入引脚寄存器PINx，其中DDRx和PORTx是可读写寄存器，而PINx为只读寄存器。每个I/O引脚内部都有独立的上拉电阻电路，可通过程序设置内部上拉电阻的有效与否。此外，如置"1"，SFIOR寄存器中的上拉屏蔽位PUD为"1"，则会屏蔽掉所有端口引脚中的内部上拉电阻。每个I/O引脚在芯片内部都有对电源Vcc和对地GND的二极管钳位保护电路，如图2.19所示。

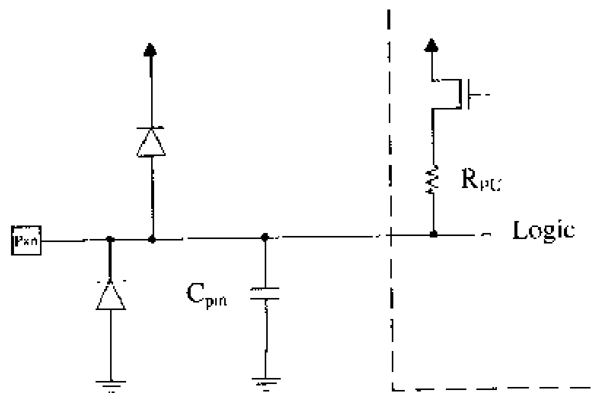


图2.19 I/O引脚等效电路

ATmega8多数的I/O口为复用口，除了作为通用数字I/O使用外，其第二功能则分别作为芯片内部其他外围电路的接口。

2.7.1 通用数字 I/O 接口

ATmega8 有 23 个 I/O 引脚，分成 3 个 8 位的端口 B、C 和 D，其中 C 口只有 7 位。所有的 I/O 端口都是双向口，每一个端口内部分别带有可选的上拉驱动电路。图 2.20 给出了一个数字 I/O 口引脚的逻辑结构。

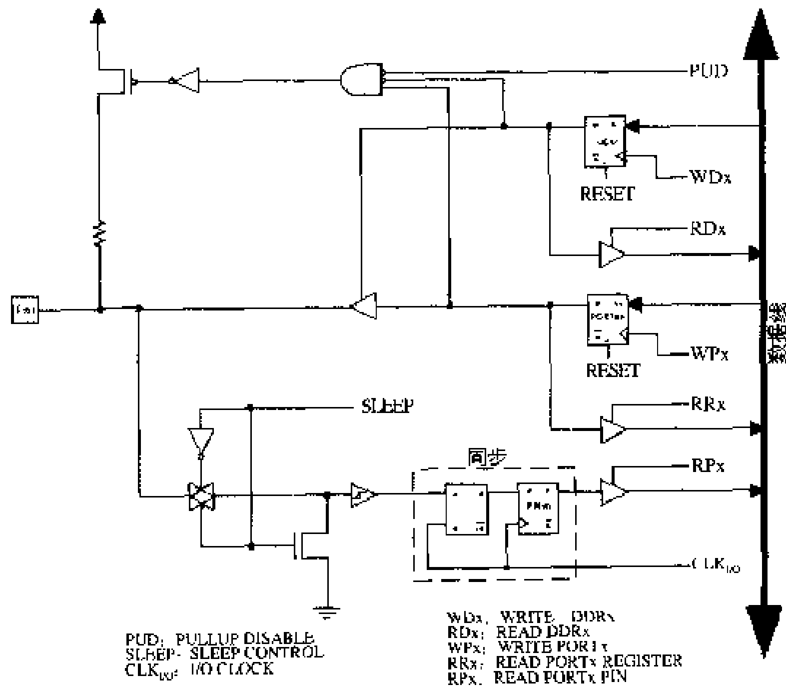


图 2.20 通用数字 I/O 口内部逻辑电路

每个 8 位的端口都有对应的 3 个 I/O 端口寄存器，它们分别是数据寄存器 PORTx、方向寄存器 DDRx 和输入引脚寄存器 PINx（x 为 B 或 C 或 D，分别代表 B 口、C 口或 D 口）。

- B 口数据寄存器——PORTB

位	7	6	5	4	3	2	1	0	
$18（$0038）	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	
复位值	0	0	0	0	0	0	0	0	

- B 口方向寄存器——DDRB

位	7	6	5	4	3	2	1	0	
$17（$0037）	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	
复位值	0	0	0	0	0	0	0	0	

- B 口输入引脚寄存器——PINB

位	7	6	5	4	3	2	1	0	
$16($0036)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
读/写	R	R	R	R	R	R	R	R	
复位值	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

● C 口数据寄存器——PORTC

位	7	6	5	4	3	2	1	0	
$15($0035)	-	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
读/写	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

● C 口方向寄存器——DDRC

位	7	6	5	4	3	2	1	0	
$14($0034)	-	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
读/写	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

● C 口输入引脚寄存器——PINC

位	7	6	5	4	3	2	1	0	
$13($0033)	-	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
读/写	R	R	R	R	R	R	R	R	
复位值	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

● D 口数据寄存器——PORTD

位	7	6	5	4	3	2	1	0	
$12($0032)	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	
复位值	0	0	0	0	0	0	0	0	

● D 口方向寄存器——DDRD

位	7	6	5	4	3	2	1	0	
$11($0031)	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	
复位值	0	0	0	0	0	0	0	0	

● D 口输入引脚寄存器——PIND

位	7	6	5	4	3	2	1	0	
$10($0030)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
读/写	R	R	R	R	R	R	R	R	
复位值	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

PORTxn、DDxn、PINxn 分别表示这三个 I/O 寄存器中相应的各个位，其中 n 为 0~7，

代表寄存器中的位值。

位于方向寄存器 DDRx 中的每个位 DDxn 用于控制一个 I/O 引脚的输入/输出方向。当 DDxn 为"1"时，对应的 Pxn 配置为输出引脚；而当 DDxn 写入"0"时，对应的 Pxn 配置为输入引脚。当 Pxn 定义为输出引脚（DDxn=1），PORTxn 中的数据为外部引脚的输出电平。即置 PORTxn 为"1"，端口引脚被强制驱动为高，输出高电平（输出电流）；清零 PORTxn，端口引脚被强制拉低，输出低电平（吸入电流）。

当 Pxn 定义为输入（DDxn=0），置 PORTxn 为"1"时，则配置该引脚的内部上拉电阻有效。要屏蔽掉内部上拉电阻，应将 PORTxn 清零，或将该引脚配置为输出。此外，通过对 I/O 特殊功能寄存器 SFIOR 中 PUD 位的设置，可以使所有引脚的上拉电阻处于无效状态。当芯片复位后，即使没有时钟脉冲，所有端口的引脚被置为高阻态。表 2-20 给出了 I/O 口的各种配置与性能。

表 2-20　I/O 口设置（n=7，6，…，1，0）

DDxn	PORTxn	PDU	I/O	上　拉	说　明
0	0	X	输入	无效	三态（高阻）
0	1	0	输入	有效	外部引脚拉低时输出电流
0	1	1	输入	无效	三态（高阻）
1	0	X	输出	无效	低电平推挽输出，吸收电流
1	1	X	输出	无效	高电平推挽输出，输出电流

I/O 特殊功能寄存器 SFIOR 的定义如下：

位	7	6	5	4	3	2	1	0	
$30（$0050）	–	–	–	ADHSM	ACME	PUD	PSR2	PSR10	SFIOR
读/写	R	R	R	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

- 位 2——PUD：上拉禁止位

当 PUD 位被置"1"后，所有 I/O 引脚的上拉电阻都无效。即使在 DDxn=0、PORTxn=1 的情况下，只要 PUD=1，则上拉电阻仍旧无效。

在将一个引脚从输入高阻态（DDxn=0，PORTxn=0）转换为高电平输出状态（DDxn=1，PORTxn=1）的过程中，会暂时出现上拉有效输入（DDxn=0，PORTxn=1）或低电平输出（DDxn=1，PORTxn=0）的中间过程。通常情况下，应先转换到上拉有效输入状态（DDxn=0，PORTxn=1），再转换为高电平输出状态（DDxn=1，PORTxn=1）。更严格的转换是先将 PUD 位置"1"，再进行上述的转换。

同样，在将一个引脚从上拉有效输入（DDxn=0，PORTxn=1）转换为低电平输出状态（DDxn=1，PORTxn=0）过程中也会产生类似的问题。要根据实际情况选择高阻态输入（DDxn=0，PORTxn=0）或高电平输出（DDxn=1，PORTxn=1）作为中间转换过程，再转换为低电平输出状态（DDxn=1，PORTxn=0）。

不管方向寄存器 DDnx 为“0”或“1”，总是可以通过读 PINxn 来获得外部引脚当前的实际电平。如图 2.20 所示，PINxn 和两个 D 触发器组成的同步锁存电路。采用这种结构的优点是，可以避免当外部引脚电平的改变出现在系统时钟边缘处时而产生一个不确定的值，但同时也形成引脚电平的变化到 PINxn 寄存器位变化之间的一个锁存延时。图 2.21 给出了读取引脚实际电平时的同步锁存时序，$t_{pd, max}$ 和 $t_{pd, min}$ 表示延时的最大值和最小值。

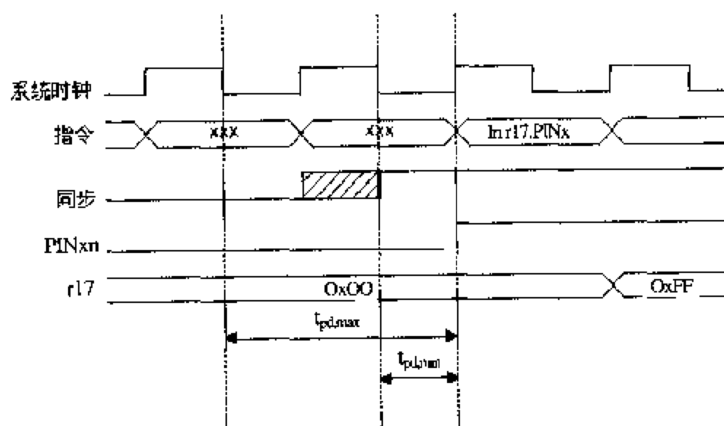


图 2.21　读取引脚实际电平时的同步锁存时序

同步锁存从第一个系统时钟的下降沿处开始。锁存器在时钟的低电平时为锁定状态，在时钟高电平时为导通状态，即为图 2.21 中“同步锁存”信号的阴影所示部分。当系统时钟变低时，外部引脚的值被锁入锁存器，在之后的时钟的上升沿处又被移入 PINxn 寄存器。如图中 tpd，max 和 tpd，min 所示，一个引脚上电平的变化到最后锁存在 PINxn 中，有 1/2 或 3/2 个系统时钟周期的延时。

如在应用中需要立即回读刚刚由程序输出到引脚的设定值，那么应在输出和输入指令之间插入一条 NOP 指令，如图 2.22 所示。此时，输出指令在系统时钟的上升沿处将同步锁存信号置“1”，将外部引脚实际电平锁存，在随后一个系统时钟的上升沿处再锁存到寄存器 PINxn 中，因此，此时的锁存延时时间 tpd 为一个系统时钟周期。

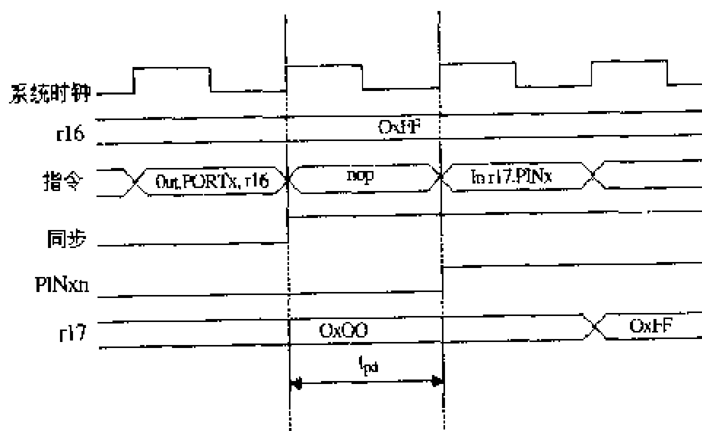


图 2.22　读取由程序设置的引脚实际电平时的同步锁存时序

下面的例程给出了如何将 PORTB 口的 0~3 号引脚配置为输出引脚，且 0、1 号引脚置

“1”，2、3 号引脚置“0”；并且配置 4~7 号引脚为输入引脚，同时 6、7 引脚内部予以上拉，然后回读设置的引脚电平值。如前所述，程序中插入了一个 NOP 指令，使得能够正确读回那些刚被设定的引脚值。

汇编的例子：

```
...
; Define pull-ups and set outputs high
; Define directions for port pins
ldi r16,(1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0)
ldi r17,(1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0)
out PORTB,r16
out DDRB,r17
; Insert nop for synchronization
nop
; Read port pins
in r16,PINB
...
```

C 语言的例子

```
unsigned char i;
...
/* Define pull-ups and set outputs high */
/* Define directions for port pins */
PORTB = (1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0);
DDRB = (1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0);
/* Insert nop for synchronization*/
_NOP();
/* Read port pins */
i = PINB;
...
```

2.7.2 数字输入使能和休眠模式

如图 2.20 所示，当引脚作为数字 I/O 口时，在施密特触发器的前端，输入信号与地之间有一个钳位电路。当 MCU 处于掉电模式（POWER-DOWN MODE）、省电模式（POWER-SAVE MODE）、等待模式（STANDBY MODE）和外部等待模式（EXTENDED STANDBY MODE）时，图中标示为 SLEEP 的信号将被置为“1”，将施密特触发器前端的输入信号钳位到地，以避免由于输入信号的浮动或有一个接近 Vcc/2 的模拟信号而引起较

高的功率消耗。

SLEEP 信号会跳过那些已经使能的外部中断源的引脚。如果外部中断源未被使能，则 SLEEP 信号仍将对这些引脚有效。SLEEP 信号也会跳过那些配置为第二功能的引脚。

如果逻辑高电平出现在一个被设置为“上升沿中断、下降沿中断或任何逻辑电平变化都引起中断”的外部异步中断引脚上，即使是该外部中断未被使能，但在 MCU 从休眠模式唤醒时，相应的外部中断标志位仍会被置“1”。这是由于 SLEEP 钳位引起的逻辑变化。

2.7.3 端口的第二功能

大多数的端口引脚除了可作为一般数字 I/O 口外，都有第二功能。图 2.23 给出了各种端口控制信号，这些控制信号可以将如图 2.20 所示引脚的通用数字 I/O 功能屏蔽掉，而将其转换用于第二功能。并不是所有的端口引脚都有这种控制屏蔽信号，但图 2.23 可以适用于 AVR 单片机系列所有端口引脚的一般描述。

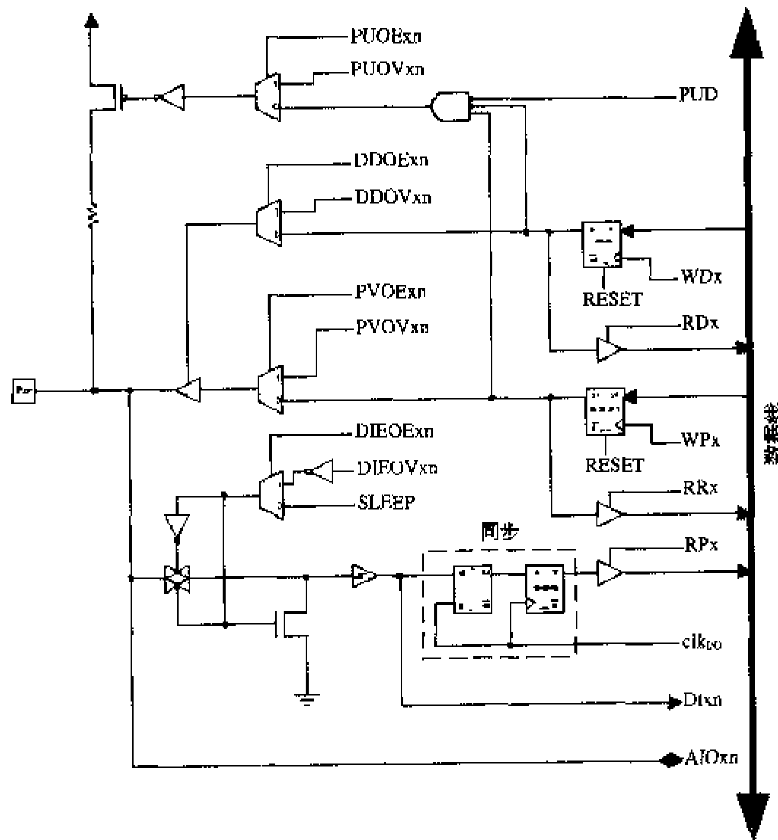


图 2.23　I/O 口第二功能控制逻辑电路

表 2-21 汇总了控制屏蔽信号的功能，这些控制屏蔽信号是由第二功能模块内部产生的。

表 2-21　第二功能模块控制屏蔽信号简述

信　号	全　名	描　述
PUOE	Pull-up Override Enable	此位被设置，上拉使能由 PUOV 控制；此位清零时，当{DDxn，PORTxn，PUD}=0b010 上拉使能
PUOV	Pull-up Override Value	如果 PUOE 被设置，上拉使能/禁止由 PUOE 的设置/清零控制，不管 DDxn，PORTxn，PUD 的状态
DDOE	Data Direction Override Enable	此位被设置，输出驱动使能由 DDOV 控制；此位清零时，由 DDxn 控制输出驱动使能
DDOV	Data Direction Override Value	如果 DDOE 被设置，输出驱动使能/禁止由 DDOV 的设置/清零控制，不管 DDxn 的状态
PVOE	Port Value Override Enable	如此位设置并且输出驱动使能，端口值由 PVOV 控制，如此位清零并且输出驱动使能，端口值由 PORTxn 控制
PVOV	Port Value Override Value	如 PVOE 设置，端口值由 PVOV 控制，不管 PORTxn 的状态
DIEOE	Digital Input Enable Override Enable	如此位设置，输入使能由 DIEOV 控制，如此位清零，由 MCU 的状态（正常模式，SLEEP 模式）决定输入使能
DIEOV	Digital Input Enable Override Value	如 DIEOE 被设置，输入使能/禁止由 DIEOV 的设置/清零控制，不管 MCU 的状态
DI	Digital Input	数字输入功能，连接同步施密特触发器的输出，当数字输入作为时钟源时，使用自己的同步信号
AIO	Analog Input/output	模拟输入/输出功能，直接连接到引脚，可双向使用

1．端口 B 的第二功能

表 2-22 给出了端口 B 各引脚的第二功能。

表 2-22　B 口引脚第二功能

引　脚	第 二 功 能
PB7	XTAL2（系统时钟晶振引脚 2） TOSC2（实时时钟晶振引脚 2）
PB6	XTAL1（系统时钟晶振引脚 1 或外部系统时钟输入口） TOSC1（实时时钟晶振引脚 1）
PB5	SCK（SPI 总线时钟）
PB4	MISO（SPI 总线主输入口/从输出口）
PB3	MOSI（SPI 总线主输出口/从输入口） OC2（T/C2 输出比较匹配输出口）
PB2	SS（SPI 总线主从选择） OC1B（T/C1 输出比较 B 匹配输出口）

续表

引　脚	第 二 功 能
PB1	OC1A（T/C1 输出比较 A 匹配输出口）
PB0	ICP（T/C1 输入捕获输入口）

● 端口 B，位 7——XTAL2/TOSC2

XTAL2：系统时钟晶振引脚 2。芯片使用外部晶振时，该引脚连接晶振的一个脚，此时，该引脚不能作为 I/O 引脚使用。当系统使用内部可校准的 RC 振荡器和外部时钟源时，PB7 可以作为一般 I/O 引脚使用。

TOSC2：实时时钟晶振引脚 2。只有当选择内部可校准的 RC 振荡器作为系统时钟源时，而且设置寄存器 ASSR 中的 AS2 位，允许使用异步时钟定时器时，PB7 才可用作 TOSC2。当 ASSR 寄存器的 AS2 位= “1”，使能定时器/计数器 2 的异步时钟功能时，PB7 与端口引脚脱离，作为振荡放大器的反向输出端。在该模式下，时钟晶体连接到该引脚，且不能作为 I/O 引脚。

如果 PB7 被用作晶振引脚，寄存器 DDB7、PORTB7 和 PINB7 读出都为 “0”。

● 端口 B，位 6——XTAL1/TOSC1

XTAL1：系统时钟晶振引脚 1。芯片使用外部晶振时，该引脚连接晶振的另一个脚，此时，该引脚不能作为 I/O 引脚使用。当系统使用内部可校准的 RC 振荡器时，PB6 可以作为一般 I/O 引脚使用。

TOSC1：实时时钟晶振引脚 1。只有当选择内部可校准的 RC 振荡器作为系统时钟源，而且设置寄存器 ASSR 中的 AS2 位允许使用异步时钟定时器时，PB6 才可用作 TOSC1。当 ASSR 寄存器的 AS2 位为 “1”，使能定时器/计数器 2 的异步时钟功能时，PB6 与端口引脚脱离，作为振荡放大器的反向输入端。在该模式下，时钟晶体连接到该引脚，且不能作为 I/O 引脚。

如果 PB6 被用作晶振引脚，寄存器 DDB6、PORTB6 和 PINB6 读出都为 “0”。

● 端口 B，位 5——SCK

SCK：用于使用 SPI 串行总线接口。当芯片作为主机时，SCK 为 SPI 总线的时钟输出端；芯片为从机时，SCK 为 SPI 总线的时钟输入端。当使能 SPI 且为从机时，无论 DDB5 为何设置，该引脚被强置为输入。尽管 SCK 引脚被 SPI 强制为输入，但内部上拉电阻仍然由 PORTB5 位控制。当使能 SPI 且为主机时，该引脚的数据方向由 DDB5 来控制。

● 端口 B，位 4——MISO

MISO：SPI 总线接口的主机数据输入/从机数据输出端。在使能 SPI 的情况下，为 SPI 主机模式时，无论 DDB4 为何值，PB4 被设置为输入；为 SPI 从机模式时，该引脚的数据方向由 DDB4 控制。当该引脚被 SPI 强制为输入时，内部上拉电阻仍然由 PORTB4 位控制。

● 端口 B，位 3——MOSI/OC2

MOSI：SPI 总线接口的主机数据输出/从机数据输入端。在使能 SPI 的情况下，为 SPI 从机模式时，无论 DDB3 为何值，PB3 被设置为输入；为 SPI 主机模式时，该引脚的数据

方向由 DDB3 控制。当该引脚被 SPI 强制为输入时，内部上拉电阻仍然由 PORTB3 位控制。

OC2：T/C2 比较匹配输出。PB3 引脚还可作为定时/计数器 2 比较匹配的外部输出口，此时，PB3 引脚必须设置为输出（DDB3=1）。在 PWM 应用中，OC2 引脚还作为 PWM 定时器模块的输出引脚。

● 端口 B，位 2——SS/OC1B

SS：SPI 总线从机选择输入。在使能 SPI，且为从机模式时，无论 DDB2 为何值，PB2 脚被设置为输入。MCU 作为 SPI 总线的从机，当 PB2 被外部拉低时，则 SPI 功能被激活。PB2 被 SPI 强制为输入时，上拉电阻仍然由 PORTB2 位控制。当使能 SPI，且为主机模式时，该引脚的数据方向由 DDB2 控制。

OC1B：T/C1 比较匹配 B 输出。PB2 引脚还可作为定时器/计数器 1 比较匹配 B 的外部输出口，此时，PB2 引脚必须设置为输出（DDB2=1）。在 PWM 应用中，OC1B 引脚还作为 PWM 定时器模块的输出引脚。

● 端口 B，位 1——OC1A

OC1A：T/C1 比较匹配 A 输出。PB1 引脚能作为定时器/计数器 1 比较匹配 A 的外部输出口，此时，PB1 引脚必须设置为输出（DDB1= “1”）。在 PWM 应用中，OC1A 引脚还作为 PWM 定时器模块的输出引脚。

● 端口 B，位 0——ICP

ICP：输入捕获的输入引脚。PB0 引脚能作为定时器/计数器 1 输入捕获功能的输入引脚。

2. 端口 C 的第二功能

表 2-23 给出了端口 C 各引脚的第二功能。

表 2-23 C 口引脚第二功能

引 脚	第 二 功 能
PC6	RESET（系统复位引脚）
PC5	ADC5（ADC 输入通道 5） SCL（2 线串行总线接口时钟线）
PC4	ADC4（ADC 输入通道 4） SDA（2 线串行总线接口数据输入/输出线）
PC3	ADC3（ADC 输入通道 3）
PC2	ADC2（ADC 输入通道 2）
PC1	ADC1（ADC 输入通道 1）
PC0	ADC0（ADC 输入通道 0）

● 端口 C，位 6——RESET

RESET：系统复位引脚。当 RSTDISBL 熔丝位被置位时，PC0 作为普通 I/O 引脚应用，此时，芯片内部的上电复位（POWER-UP）和 BROWN-OUT 复位电路将作为系统的复位源。当 RSTDISBL 熔丝位被清零时，内部复位电路将连接到该引脚，此时，引脚不作为 I/O 使用，当被外部拉成低电平时，产生系统复位。

如果 PC6 作为 RESET 复位引脚时，寄存器 DDC6、PORTC6 和 PINC6 读出为“0”。

● 端口 C，位 5——SCL/ADC5

SCL：两线串行总线的时钟线。当 TWCR 寄存器中的 TWEN 位被设置为 1，使能 TWI 接口时，PC5 引脚将与 I/O 端口脱离，成为 TWI 总线接口的串行时钟线。PC5 工作在 TWI 模式下时，有一个尖峰滤波器连接到该引脚，能够抑制输入信号中小于 50ns 的毛刺，同时引脚将由具有缓冲率限制（slew-rate limitation）的开漏驱动器驱动。

ADC5：PC5 也能作为 ADC 输入的通道 5。注意，ADC 输入通道 5 由数字电源端 Vcc 供电。

● 端口 C，位 4——SDA/ADC4

SDA：两线串行总线的数据线。当 TWCR 寄存器中的 TWEN 位被设置为 1，使能 TWI 接口时，PC4 引脚将与 I/O 端口脱离，成为 TWI 总线接口的串行数据线。PC4 工作在 TWI 模式下时，有一个尖峰滤波器连接到该引脚，能够抑制输入信号中小于 50ns 的毛刺，同时引脚将由具有缓冲率限制（slew-rate limitation）的开漏驱动器驱动。

ADC4：PC4 也能作为 ADC 输入的通道 4。注意，ADC 输入通道 4 由数字电源端 Vcc 供电。

● 端口 C，位 3——ADC3

ADC3：PC3 也作为 ADC 输入通道 3。ADC 输入通道 3 使用模拟电源端 AVcc 供电。

● 端口 C，位 2——ADC2

ADC2：PC2 也作为 ADC 输入通道 2。ADC 输入通道 2 使用模拟电源端 AVcc 供电。

● 端口 C，位 1——ADC1

ADC1：PC1 也作为 ADC 输入通道 1。ADC 输入通道 1 使用模拟电源端 AVcc 供电。

● 端口 C，位 0——ADC0

ADC0：PC0 也作为 ADC 输入通道 0。ADC 输入通道 0 使用模拟电源端 AVcc 供电。

3. 端口 D 的第二功能

表 2-24 给出了端口 D 各引脚的第二功能。

表 2-24　D 口引脚第二功能

引　脚	第 二 功 能
PD7	AIN1（模拟比较器负输入）
PD6	AIN0（模拟比较器正输入）
PD5	T1（T/C1 外部计数脉冲输入口）
PD4	XCK（USART 外部时钟输入/输出口） T0（T/C0 外部计数脉冲输入口）
PD3	INT1（外部中断 1 输入）
PD2	INT0（外部中断 0 输入）
PD1	TXD（USART 输出口）
PD0	RXD（USART 输入口）

- 端口 D，位 7——AIN1

AIN1：模拟比较器的反相输入。在使用模拟比较器功能时，应将 PD7 设置为输入，且关断内部上拉电阻，以避免数字口功能影响模拟比较器的性能。

- 端口 D，位 6——AIN0

AIN0：模拟比较器的正相输入。在使用模拟比较器功能时，应将 PD6 设置为输入，且关断内部上拉电阻，以避免数字口功能影响模拟比较器的性能。

- 端口 D，位 5——T1

T1：定时器/计数器 1 的外部计数脉冲输入口。

- 端口 D，位 4——XCK/T0

XCK：USART 串行总线外部时钟口。

T0：定时器/计数器 0 的外部计数脉冲输入口。

- 端口 D，位 3——INT1

INT1：外部中断源 1。PD3 引脚可作为一个外部中断源的输入口。

- 端口 D，位 2——INT0

INT0：外部中断源 0。PD2 引脚可作为一个外部中断源的输入口。

- 端口 D，位 1——TXD

TXD：USART 总线的数据输出口。当使用 USART 的传送输出功能时，不管 DDD1 为何设置，PD1 被配置为输出口。

- 端口 D，位 0——RXD

RXD：USART 总线的数据输入口。当使用 USART 的数据接收功能时，不管 DDD0 为何设置，PD0 被配置为输出口，此时引脚的内部上拉功能仍然由 PORTD0 位控制。

2.8 外部中断

外部中断是由 INT0 和 INT1 引脚触发的。需要注意的是，如果设置允许外部中断产生，即使是 INT0 和 INT1 引脚设置为输出方式，外部中断也会触发，这一特性提供了使用软件产生中断的途径。外部中断可选择采用上升沿触发、下降沿触发以及低电平触发。具体方式是由 MCU 控制寄存器 MCUCR 以及 MCU 控制和状态寄存器 MCUCSR 决定。当允许外部中断，且设置为电平触发方式时，只要中断输入引脚保持低电平，那么将一直触发产生中断。而对于识别在 INT0 和 INT1 引脚上的上升沿或下降沿的中断触发，则需要 I/O 时钟信号的存在。由于低电平触发中断的检测是采用异步方式检测的，不需要时钟信号，因此电平中断可以作为外部唤醒源，将处在各种休眠模式的 MCU 唤醒。这是由于只有在空闲模式中 I/O 时钟信号还保持继续工作，而在其他各种休眠模式下，I/O 时钟信号是不工作的。

如果用低电平触发中断作为唤醒源，将 MCU 从掉电模式中唤醒时，电平改变后仍需

要维持一段时间才能将 MCU 唤醒，这提高了 MCU 的抗噪性能。改变的触发电平将由看门狗的时钟信号采样两次。在通常的 5V 电源和 25℃时，看门狗的时钟周期为 1μs。如果输入电平符合中断触发电平的条件，且能保持 2 次采样周期的时间，或者一直保持到 MCU 启动延时（start-up time）过程之后，MCU 将被唤醒。如果该电平的保持时间能够满足看门狗的两次采样，但是在启动延时（start-up time）过程完成之前就消失了，那么 MCU 仍将被唤醒，但不会触发中断进入中断服务程序。所以，为了保证既能将 MCU 唤醒，又能触发中断，中断触发电平必须维持足够长的时间。

1. MCU 控制寄存器 MCUCR

MCU 控制寄存器中包含中断方式控制位和一般的 MCU 功能控制位，每一位定义如下。

位	7	6	5	4	3	2	1	0	
$35($0055)	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

- 位 3、2——ISC11、ISC10：外部中断 1 的中断方式控制位 1 和位 0

如果 SREG 寄存器中的 I 位和 GICR 寄存器中相应的中断屏蔽位被置为“1”，外部中断 1 将会由外部引脚 INT1 上的电平变化而触发。INT1 的中断触发方式在表 2-25 中定义。MCU 对 INT1 引脚上电平值的采样在边沿检测前。如果选择脉冲边沿触发中断的方式，那么脉宽大于一个时钟周期的脉冲变化将触发中断，过短的脉冲则不能保证触发中断。如果选择低电平触发中断，那么低电平必须保持到当前指令执行完成才触发中断。

表 2-25 INT1 中断方式

ISC11	ISC10	中断方式
0	0	INT1 的低电平产生一个中断请求
0	1	INT1 的下降沿和上升沿都产生一个中断请求
1	0	INT1 的下降沿产生一个中断请求
1	1	INT1 的上升沿产生一个中断请求

- 位 1、0——ISC01、ISC00：外部中断 0 的中断方式控制位 1 和位 0

如果 SREG 寄存器中的 I 位和 GICR 寄存器中相应的中断屏蔽位被置为“1”，外部中断 0 将会由外部引脚 INT0 上的电平变化而触发。INT0 的中断触发方式在表 2-26 中定义。MCU 对 INT0 引脚上电平值的采样在边沿检测前。如果选择脉冲边沿触发中断的方式，那么脉宽大于一个时钟周期的脉冲变化将触发中断，过短的脉冲则不能保证触发中断。如果选择低电平触发中断，那么低电平必须保持到当前指令执行完成才触发中断。

表 2-26 INT0 中断方式

ISC01	ISC00	中断方式
0	0	INT0 的低电平产生一个中断请求
0	1	INT0 的下降沿和上升沿都产生一个中断请求

续表

ISC01	ISC00	中 断 方 式
1	0	INT0 的下降沿产生一个中断请求
1	1	INT0 的上升沿产生一个中断请求

2. 通用中断控制寄存器 GICR

位	7	6	5	4	3	2	1	0	
$3B($005B)	INT1	INT0					IVSEL	IVCE	GICR
读/写	R/W	R/W	R	R	R	R	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

● 位 7——INT1：外部中断请求 1 使能

当 INT1 位被置“1”，同时状态寄存器 SREG 的 I 位被置为“1”时，外部引脚中断 1 被使能。MCU 通用控制寄存器 MCUCR 中的中断 1 方式控制位 ISC11 和 ISC10 决定了外部中断 1 是由引脚上的低电平触发，还是由上升沿或者下降沿触发。即使 INT1 引脚被配置为输出方式，其上的有效电平信号的变化仍会触发中断。

● 位 6——INT0：外部中断请求 0 使能

当 INT0 位被置“1”，同时状态寄存器 SREG 的 I 位被置为“1”时，外部引脚中断 0 被使能。MCU 通用控制寄存器 MCUCR 中的中断 0 方式控制位 ISC01 和 ISC00 决定了外部中断 0 是由引脚上的低电平触发，还是由上升沿或者下降沿触发。即使 INT0 引脚被配置为输出方式，其上的有效电平信号的变化仍会触发中断。

3. 通用中断标志寄存器 GIFR

位	7	6	5	4	3	2	1	0	
$3A($005A)	INTF1	INTF0							GIFR
读/写	R/W	R/W	R	R	R	R	R	R	
复位值	0	0	0	0	0	0	0	0	

● 位 7——INTF1：外部中断标志位 1

当 INT1 引脚上的有效事件触发一个中断请求后，INTF1 位会变成“1”。如果 SREG 寄存器中 I 位和 GICR 寄存器中的 INT1 位被置为“1”，MCU 将跳至相应的中断向量处开始执行中断服务程序，同时硬件自动将 INTF1 标志位清零。如果写逻辑“1”到标志位 INTF1，该位将被清零。当 INT1 设置为低电平触发方式时，标志 INTF1 位始终为“0”。

● 位 6——INTF0：外部中断标志位 0

当 INT0 引脚上的有效事件触发一个中断请求后，INTF0 位会变成“1”。如果 SREG 寄存器中 I 位和 GICR 寄存器中的 INT0 位被置为“1”，MCU 将跳至相应的中断向量处开始执行中断服务程序，同时硬件自动将 INTF0 标志位清零。如果写逻辑“1”到标志位 INTF0，该位将被清零。当 INT0 设置为低电平触发方式时，标志 INTF0 位始终为“0”。

2.9 ATmega8 的定时器/计数器

ATmega8 单片机有 3 个定时/计数器：8-bit Timer/Counter0、16-bit Timer/Counter1 和 8-bit Timer/Counter2（以下表示为 T/C0、T/C1、T/C2）。这些定时/计数器除了能够实现通常的定时和计数功能外，还具有捕捉、比较、脉宽调制（PWM）输出、实时时钟计数等更为强大的功能。

2.9.1 定时器/计数器预定比例分频器

ATmega8 的 T/C0 和 T/C1 由一个 10 位预定比例分频器（Prescaler）提供时钟源。该预定比例分频器将系统时钟按设定的比例进行分频，以产生不同周期的时钟 clk_{T0}、clk_{T1}，分别作为时钟源提供给 T/C0 和 T/C1 使用，这使得 AVR 的定时/计数器的使用更加灵活和方便。图 2.24 所示为预定比例分频器结构图。

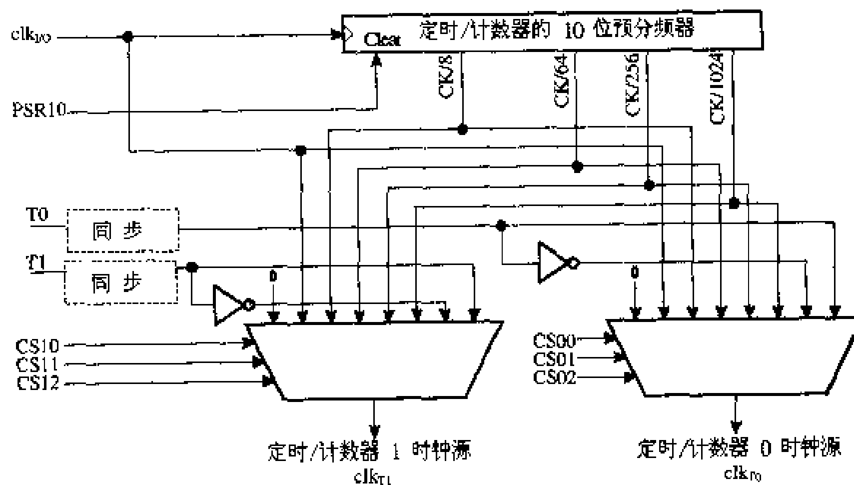


图 2.24 T/C0 和 T/C1 的预定比例分频器结构

1. T/C0 和 T/C1 的时钟源

T/C0 和 T/C1 的时钟源可来自芯片内部，也可来自外部引脚 T0 和 T1。

- 当 CSn[2:0]=1 时，系统内部时钟直接作为定时/计数器的时钟源，这也是最高频率的时钟源（$clk_{I/O}$）。
- 由预定比例分频器提供的时钟源。预定比例分频器对系统时钟 $clk_{I/O}$ 按四个不同的分频比例分频，输出 4 个不同周期的时钟信号 $clk_{I/O}/8$、$clk_{I/O}/64$、$clk_{I/O}/256$、$clk_{I/O}/1024$。两个定时/计数器可以分别选定其中一个作为时钟源。
- T/C0（T/C1）可以使用来自外部引脚 T0（T1）的时钟信号作为外部时钟源

clk_{T0}（clk_{T1}）。

2．外部时钟信号的检测

施加在外部引脚 T0/T1 的时钟信号，由引脚同步检测电路采样，每个系统时钟周期同步采样一次，采样输出信号进入边沿检测器（见图 2.25）。同步检测电路在系统时钟 $clk_{I/O}$ 的上升沿将外部信号电平打入寄存器，下降沿时打入的电平由寄存器输出。当系统时钟频率大大高于外部输入时钟频率时，同步检测寄存器电路可以看作是透明的。边沿检测器对同步检测的输出信号进行边沿检测，检测到信号一个正跳变产生一个 clk_{Tn} 脉冲（CSn[2:0]=7），或者一个负跳变产生一个 clk_{Tn} 脉冲（CSn[2:0]=6）。

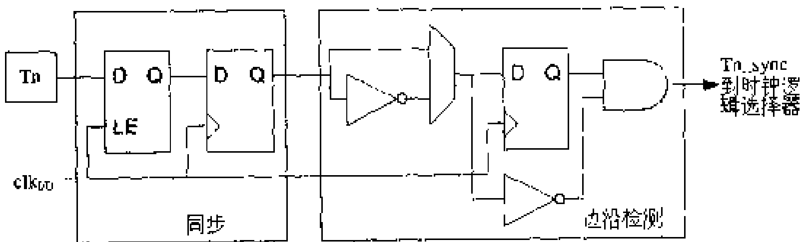


图 2.25 T/C0 和 T/C1 的外部时钟信号检测电路

由于引脚上同步检测电路和边沿检测电路的存在，在引脚 T0/T1 上的电平变化，需要延时 2.5~3.5 个系统时钟周期才能在边沿检测的输出反映出来。要使外部时钟能被引脚的检测电路采样，外部时钟的最高频率不能大于 $F_{CLK_I/O}/2.5$，脉冲宽度要大于一个系统时钟 $clk_{I/O}$ 周期。此外，外部时钟源是不进入预定比例分频器再次分频的。

3．特殊功能 I/O 寄存器—— SFIOR

位	7	6	5	4	3	2	1	0	
$30（$0050）	–	–	–	ADHSM	ACME	PUD	PSR2	PSR10	SFIOR
读/写	R	R	R	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

● 位 0——PSR10：预定比例分频器复位

当写入“1”到该位，将复位预定比例分频器，一旦预定比例分频器被复位，硬件自动清零该标志位。而写“0”到该位，则不会产生任何操作。由于预定比例分频器的输出为 T/C0 和 T/C1 共享，因此复位预定比例分频器会同时影响到两个定时/计数器。读取 PSR10 位的值时，总是为“0”。

2.9.2 8 位定时器/计数器 0——T/C0

ATmega8 的 T/C0 是一个通用 8 位定时/计数器，其主要特点是：

- 单通道计数器
- 频率发生器
- 外部事件计数

- 带 10 位预定比例分频器

1．8 位 T/C0 的寄存器

（1）T/C0 控制寄存器——TCCR0

位	7	6	5	4	3	2	1	0	
$33（$0053）	-	-	-	-	-	CS02	CS01	CS00	TCCR0
读/写	R	R	R	R	R	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

- 位[2:0]——CS0[2:0]：时钟源选择

这 3 个标志位用于选择 T/C0 的时钟源，见表 2-27。

表 2-27　T/C0 的时钟源选择

CS02	CS01	CS00	说　明
0	0	0	无时钟源（停止 T/C0）
0	0	1	$clk_{I/O}$（系统时钟）
0	1	0	$clk_{I/O}$/8（来自预分频器）
0	1	1	$clk_{I/O}$/64（来自预分频器）
1	0	0	$clk_{I/O}$/256（来自预分频器）
1	0	1	$clk_{I/O}$/1024（来自预分频器）
1	1	0	外部 T0 脚，下降沿驱动
1	1	1	外部 T0 脚，上升沿驱动

当选择使用外部时钟源时，无论 T0 引脚是否定义为输出功能，在 T0 引脚上的逻辑信号电平的变化都会驱动 C/T0 计数，这个特性允许用户通过软件来控制计数。

（2）T/C0 计数寄存器——TCNT0

位	7	6	5	4	3	2	1	0	
$32（$0052）	TCNT0[7..0]								TCNT0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

TCNT0 是 T/C0 的计数值寄存器，该寄存器可以直接被读写访问。写 TCNT0 寄存器将在下一个定时器时钟周期中阻塞比较匹配。

（3）定时/计数器中断屏蔽寄存器——TIMSK

位	7	6	5	4	3	2	1	0	
$39（$0059）	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	-	TOIE0	TIMSK
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

- 位 0——TOIE0：T/C0 溢出中断允许标志位

当 TOIE0 被设为“1”，且状态寄存器中的 I 位被设为“1”时，将使能 T/C0 溢出中断。

若在 T/C0 上发生溢出时（TOV0=1），则执行 T/C0 溢出中断服务程序。

（4）定时/计数器中断标志寄存器——TIFR

位	7	6	5	4	3	2	1	0	
$38（$0058）	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	–	TOV0	TIFR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

● 位 0——TOV0：T/C0 溢出中断标志位

当 T/C0 产生溢出时，TOV0 位被设置为“1”。当 MCU 转入 T/C0 溢出中断向量执行中断处理程序时，TOV0 由硬件自动清零。写入一个逻辑“1”到 TOV0 标志位将清除该标志位。当寄存器 SREG 中的 I 位、TOIE0 以及 TOV0 均为“1”时，T/C0 的溢出中断被响应。

2. 8 位定时/计数器 0

8 位 T/C0 单元是一个可编程的计数器，图 2.26 为它的逻辑功能图。计数器为单向加一计数器，对每一个时钟 clk_{T0} 加 1 计数。clk_{T0} 的来源由标志位 CS0[2:0]设定，当 CS0[2:0]=0 时，计数器停止计数（无计数时钟源）。

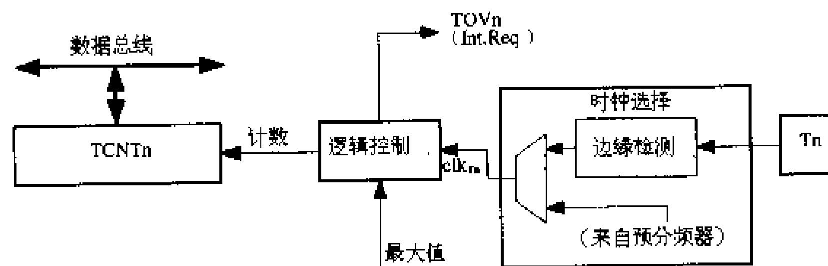


图 2.26 T/C0 的逻辑功能图

计数值保存在寄存器 TCNT0 中，MCU 可以在任何时间访问读/写 TCNT0。MCU 写入 TCNT0 的值将立即覆盖其中原有的内容，并会影响计数器的运行。

计数器单一向上计数，一旦寄存器 TCNT0 的值到达 0xFF，下一个计数脉冲到来时便恢复为 0x00，并继续向上开始计数。在 TCNT0 为 0x00 的同时，置溢出标志位 TOV0 为“1”。标志位 TOV0 可以用于申请中断，也可以作为计数器的第 9 位使用（使 T/C0 变成 9 位计数器）。用户可以通过写入 TCNT0 寄存器初值来调整计数器溢出的时间间隔。图 2.27、图 2.28 所示为 T/C0 的计数时序，图中 MAX=0xFF，BOTTOM=0x00。

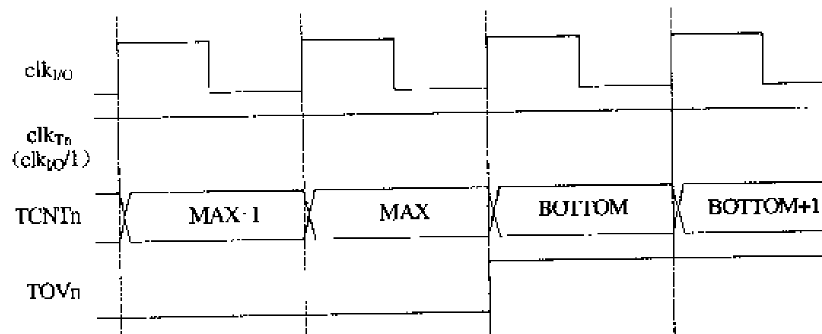


图 2.27 T/C0 计数时序（无预分频）

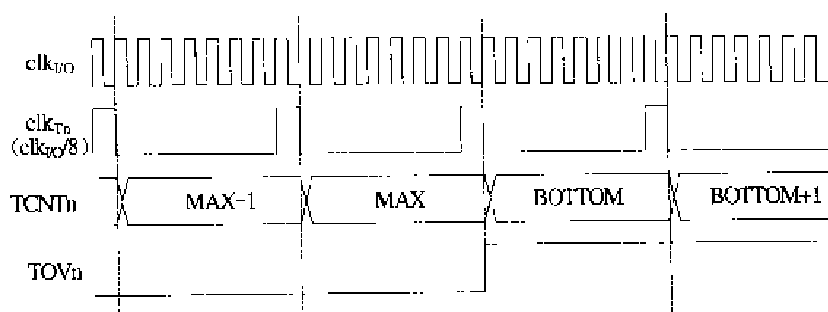


图 2.28　T/C0 计数时序（预分频为 fclk_I/O/8）

2.9.3　16 位定时器/计数器 1——T/C1

ATmega8 的 T/C1 是一个 16 位的多功能定时/计数器，其主要特点是：

- 真正的 16 位设计（如 16 位 PWM）；
- 两个独立的输出比较单元；
- 双缓冲输出比较寄存器；
- 一个输入捕获单元；
- 输入捕获噪声抑制；
- 比较匹配时清零计数器（自动重装）；
- 无奇边非对称，相位可调的脉宽调制输出（PWM）；
- 周期可调的 PWM；
- 频率发生器；
- 外部事件计数；
- 4 个独立的中断源（TOV1、OCF1A、OCF1B 和 ICF1）。

同早期 AVR 单片机的 16 位定时/计数器相比（如 AT90S8515、AT90S8535 等），ATmega8 的 16 位定时/计数器不仅兼容原有的功能，而且还增加了许多新的功能和特点。一些控制寄存器以及标志位的地址和名称也有所变化，同时还新增加了相关的控制寄存器和控制位标志。因此，原来熟悉和使用过早期 AVR 产品的用户在选用 MEGA 系列单片机时，应注意 T/C1 的变化和不同。

1．T/C1 的组成结构

图 2.29 为 16 位 T/C1 的结构框图。在图中给出了 MCU 可以操作的寄存器以及相关的标志位，其中，定时/计数器寄存器 TCNT1、输出比较寄存器 OCR1A/OCR1B 以及输入捕获寄存器 ICR1 都是 16 位的寄存器。对这些 16 位寄存器的读写操作应遵循特定的步骤，请参考以下详细的说明。T/C1 的中断请求信号可以在定时器中断标志寄存器 TIFR 中找到。在定时器中断屏蔽寄存器 TIMSK 中，可以找到各自独立的中断屏蔽位。

（1）T/C1 的时钟源

T/C1 时钟源可来自芯片内部，也可来自外部引脚 T1。T/C1 与 T/C0 共享一个预定比例分频器（见 2.9.1 小节内容）。时钟源的选择由寄存器 TCCR1B 中的标志位 CS1[2:0]确定。

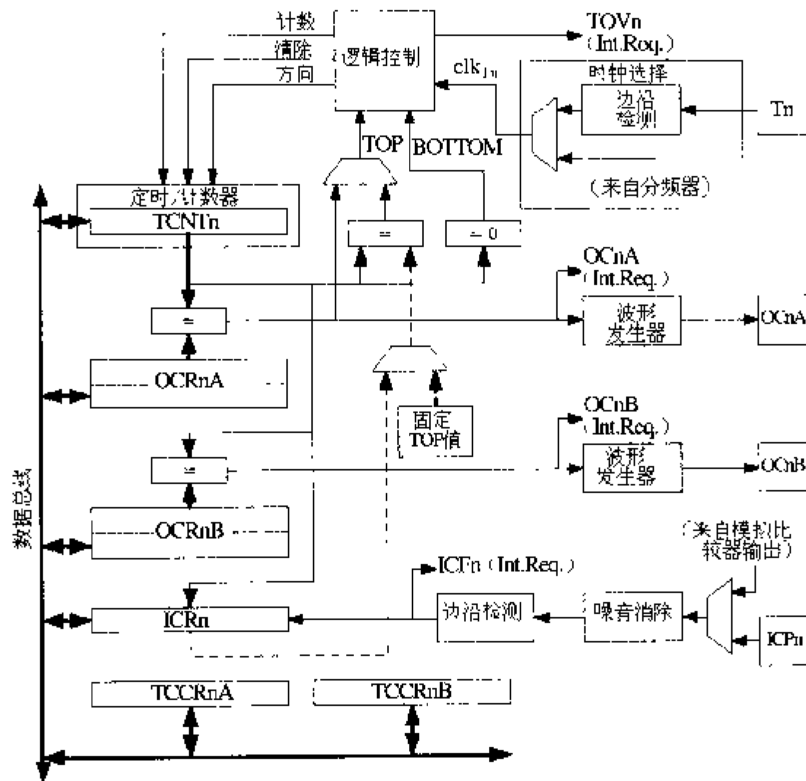


图 2.29　T/C1 的结构框图

（2）16 位 T/C1 的计数单元

T/C1 的计数单元是一个可编程的 16 位双向计数器，图 2.30 为它的逻辑功能图。根据计数器的工作模式，在每一个 clk_{T1} 时钟到来时，计数器进行加 1、减 1 或清零操作。clk_{T1} 的来源由标志位 CS1[2:0]设定，当 CS1[2:0]=0 时，计数器停止计数（无计数时钟源）。

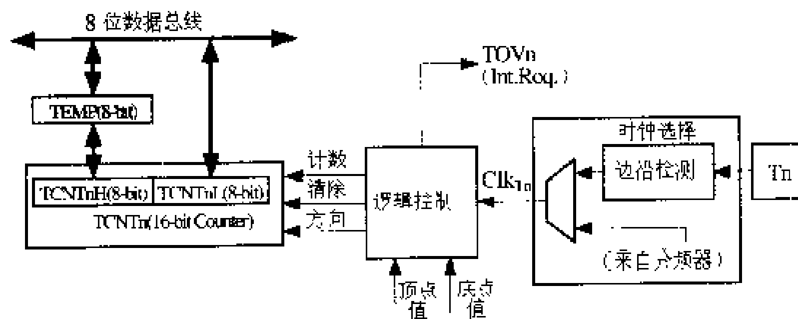


图 2.30　T/C1 计数器的逻辑功能图

图 2.30 中的符号所代表的意义如下：

- 计数（count）　　TCNT1 加 1 或减 1。
- 方向（direction）　　加或减的选择。
- 清除（clear）　　清零 TCNT1。
- clk_{T1}　　计数器时钟信号。

- 顶点值（TOP）　　　表示TCNT1计数值到达最大值。
- 底点值（BOTTOM）　　表示TCNT1计数值到达最小值（零）。

计数值保存在16位寄存器TCNT1中。TCNT1由两个8位寄存器：TCNT1H和TCNT1L组成。MCU对TCNT1的访问操作应遵循特定的步骤。

计数器的计数序列取决于寄存器TCCR1A和TCCR1B中的标志位WGM1[3:0]的设置。WGM1[3:0]的设置直接影响到计数器的运行方式、OC1A和OC1B的输出形式，同时也影响和涉及T/C1的溢出标志位TOV1的置位。标志位TOV1可以用于产生中断申请。

（3）输入捕获单元

T/C1内部的输入捕获单元（如图2.31所示）可应用于精确捕获一个外部事件的发生，以及事件发生的时间印记（Time-stamp）。外部事件发生的触发信号由引脚ICP1输入。此外，模拟比较器的ACO单元的输出信号也可作为外部事件捕获的触发信号。

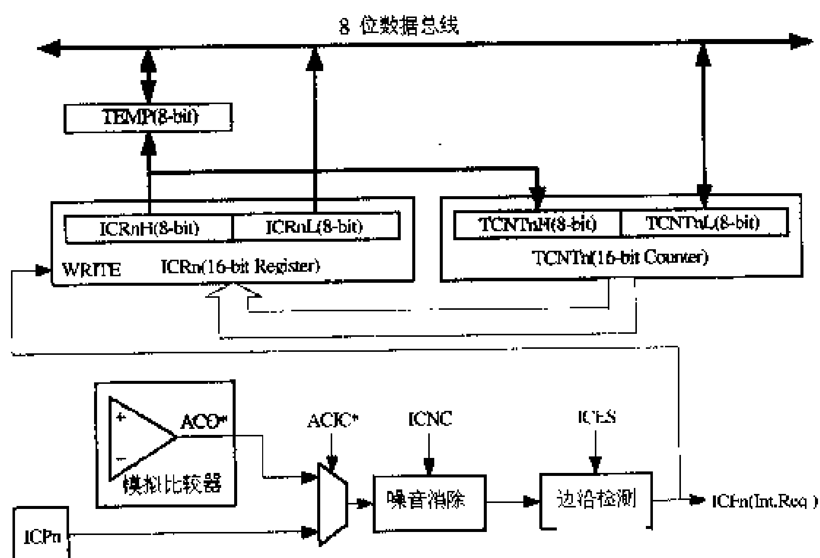


图2.31　T/C1的外部事件输入捕获单元

一个输入捕获发生在外部引脚ICP1上的逻辑电平变化，或者模拟比较器输出电平变化（事件发生），此时T/C1计数器TCNT1中的计数值被写入输入捕获寄存器ICR1中，并置位输入捕获标志位ICF1。输入捕获功能可用于频率和周期的精确测量。

当T/C1运行在PWM方式下时，允许对寄存器ICR1进行写操作。应在设置T/C1的运行方式为PWM后（写WGM1[2:0]），再设置ICR1，此时写入ICR1的值将作为计数器计数序列的上限值（TOP）。

置位标志位ICNC1将使能对输入捕获触发信号的噪声抑制功能。噪声抑制电路是一个数字滤波器，它对输入触发信号进行4次采样，当4次采样值相等才确认此触发信号。因此使能输入捕获触发信号的噪声抑制功能对输入触发信号的噪声抑制，但确认的触发信号比真实的触发信号延时了4个系统时钟周期。

（4）输出比较单元

图2.32为T/C1的输出比较单元逻辑功能图。在T/C1运行期间，输出比较单元一直将

寄存器 TCNT1 的计数值同寄存器 OCR1A 和 OCR1B 的内容进行比较，一旦发现与其中之一相等，比较器即会产生一个比较匹配信号，在下一个计数时钟脉冲到达时置位 OCF1A 或 OCF1B 中断标志位。根据 WGM1[2:0]和 COM1A[1:0]、COM1B[1:0]的不同设置，比较相等匹配的信号还用于控制和产生各种类型的脉冲（PWM）波形。

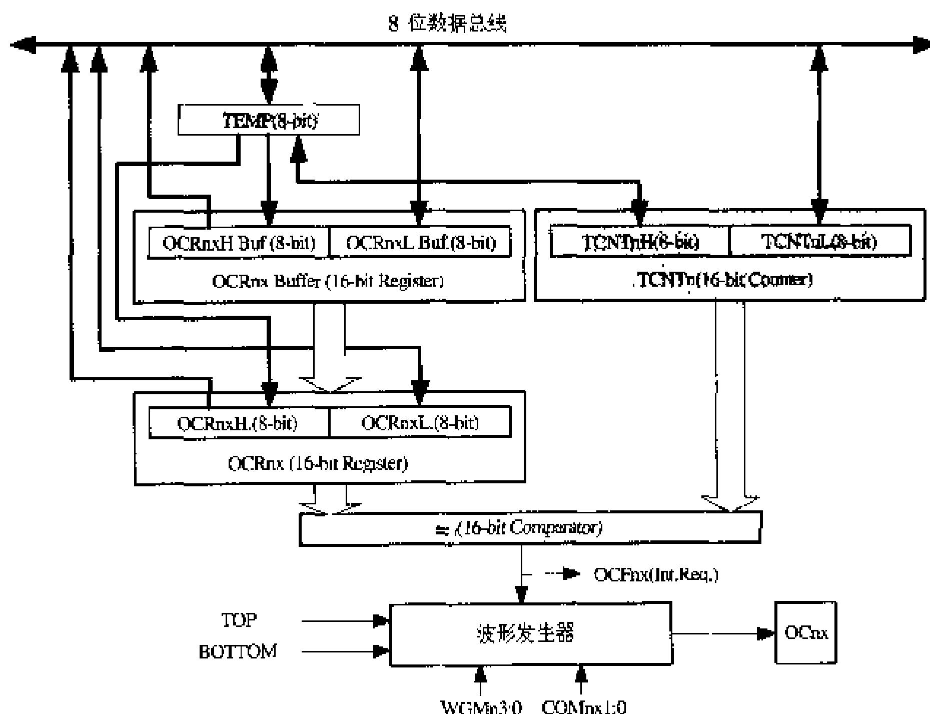


图 2.32 T/C1 输出比较单元逻辑功能图

寄存器 OCR1A 和 OCR1B 各自配置有一个辅助缓存器。当 T/C1 工作在非 PWM 模式时，该辅助缓存器被禁止。MCU 直接访问和操作寄存器 OCR1A 或 OCR1B 本身。

当 T/C1 工作在 12 种 PWM 模式中的任何一种时，OCR1A 和 OCR1B 的辅助缓存器投入使用。这时，MCU 对 OCR1A 或 OCR1B 的访问操作实际上是在对它们相对应的辅助缓存器操作。当计数器的计数值达到设定的最大值（TOP）或最小值（BOTTOM）时，辅助缓存器的内容将同步更新比较寄存器 OCR1A 或 OCR1B 的值。这有效地防止产生奇边非对称的 PWM 脉冲信号，使输出的 PWM 波中没有杂散脉冲。

（5）比较匹配输出单元

标志位 COM1A[1:0]（COM1B[1:0]）有两个作用：定义 OC1A（OC1B）的输出方式，以及控制外部引脚 OC1A（OC1B）是否输出 OC1A（OC1B）寄存器的值。图 2.33 为比较匹配输出单元的逻辑图。

当标志位 COM1A[1:0]（COM1B[1:0]）中任何一位为“1”时，波形发生器的输出 OC1A（OC1B）取代引脚原来的 I/O 功能，但引脚的方向寄存器 DDR 仍然控制 OC1A（OC1B）引脚的输入/输出方向。如果要在外部引脚输出 OC1A（OC1B）的逻辑电平，应设定 DDR 定义该引脚为输出脚。采用这种结构，用户可以先初始化 OC1A（OC1B）的状态，然后再允许其由引脚输出。

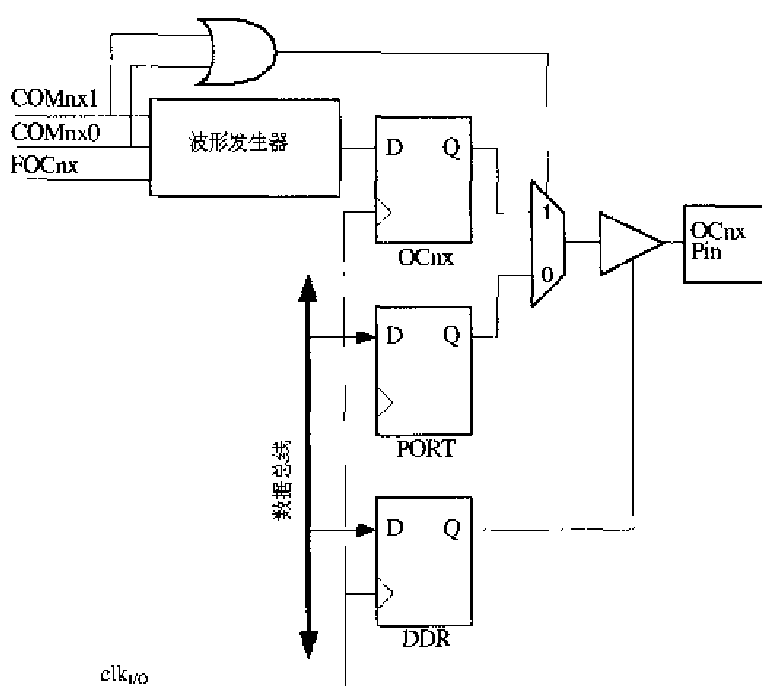


图 2.33 T/C1 的比较匹配输出单元逻辑图

COM1A[1:0]（COM1B[1:0]）的设置对输入捕获没有影响。

（6）比较输出模式和波形发生器

对于 T/C1 的各种工作模式，COM1A[1:0]（COM1B[1:0]）的不同设置都会影响到波形发生器产生的脉冲波形方式。但只要 COM1A[1:0]=0（COM1B[1:0]=0），波形发生器对 OC1A（OC1B）寄存器就没有任何作用。

2．16 位 T/C1 的寄存器

（1）T/C1 计数寄存器——TCNT1H 和 TCNT1L

位	15	14	13	12	11	10	9	8	
$2D($004D)	TCNT1[15:8]								TCNT1H
$2C($004C)	TCNT1[7:0]								TCNT1L
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	
复位值	0	0	0	0	0	0	0	0	
复位值	0	0	0	0	0	0	0	0	

TCNT1H 和 TCNT1L 组成 T/C1 的计数值寄存器 TCNT1，该寄存器可以直接被 CPU 读写访问，但应遵循特定的步骤。写 TCNT1 寄存器将在下一个定时器时钟周期中阻塞比较匹配。因此，在计数器运行期间修改 TCNT1 的内容，有可能丢失一次 TCNT1 与 OCR1A（OCR1B）的匹配比较操作。

（2）输出比较寄存器——OCR1AH 和 OCR1AL，OCR1BH 和 OCR1BL

位	15	14	13	12	11	10	9	8	
$2B($004B)	OCR1A[15:8]								OCR1AH
$2A($004A)	OCR1A[7:0]								OCR1AL
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	
复位值	0	0	0	0	0	0	0	0	
复位值	0	0	0	0	0	0	0	0	

位	15	14	13	12	11	10	9	8	
$29 ($0049)	OCR1B[15:8]								OCR1BH
$28 ($0048)	OCR1B[7:0]								OCR1BL
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	
复位值	0	0	0	0	0	0	0	0	
复位值	0	0	0	0	0	0	0	0	

OCR1AH 和 OCR1AL（OCR1BH 和 OCR1BL）组成 16 位输出比较寄存器 OCR1A（OCR1B）。该寄存器中的 16 位数据用于同 TCNT1 寄存器中的计数值进行连续的匹配比较。一旦 TCNT1 的计数值与 OCR1A（OCR1B）的数据匹配相等，将产生一个输出比较匹配相等的中断申请，或改变 OC1A（OC1B）的输出逻辑电平。

（3）输入捕获寄存器——ICR1H 和 ICR1L

位	15	14	13	12	11	10	9	8	
$27 ($0047)	ICR1[15:8]								ICR1H
$26 ($0046)	ICR1[7:0]								ICR1L
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	
复位值	0	0	0	0	0	0	0	0	
复位值	0	0	0	0	0	0	0	0	

ICR1H 和 ICR1L 组成 16 位输入捕获寄存器 ICR1。当外部引脚 ICP1 或模拟比较器有输入捕获触发信号产生时，计数器 TCNT1 中的计数值写入寄存器 ICR1 中。

在 PWM 方式下，ICR1 的设定值将作为计数器计数上限（TOP）值。

（4）定时/计数器中断屏蔽寄存器——TIMSK

位	7	6	5	4	3	2	1	0	
$39（$0059）	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	-	TOIE0	TIMSK
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

- 位 5——TICIE1：T/C1 输入捕获中断允许标志位

当 TICIE1 被设为“1”，且状态寄存器中的 I 位被设为“1”时，将使能 T/C1 的输入捕获中断。若在 T/C1 上发生输入捕获（ICF1=1），则执行 T/C1 输入捕获中断服务程序。

- 位 4——OCIE1A：T/C1 输出比较 A 匹配中断允许标志位

当 OCIE1A 被设为“1”，且状态寄存器中的 I 位被设为“1”时，将使能 T/C1 的输出比较 A 匹配中断。若在 T/C1 上发生输出比较 A 匹配（OCF1A=1），则执行 T/C1 输出比较 A 匹配中断服务程序。

- 位 3——OCIE1B：T/C1 输出比较 B 匹配中断允许标志位

当 OCIE1B 被设为“1”，且状态寄存器中的 I 位被设为“1”时，将使能 T/C1 的输出比较 B 匹配中断。若在 T/C1 上发生输出比较 B 匹配（OCF1B=1），则执行 T/C1 输出比较 B 匹配中断服务程序。

- 位 2——TOIE1：T/C1 溢出中断允许标志位

当 TOIE1 被设为“1”，且状态寄存器中的 I 位被设为“1”时，将使能 T/C1 溢出中断。若在 T/C1 上发生溢出时（TOV1=1），则执行 T/C1 溢出中断服务程序。

（5）定时/计数器中断标志寄存器——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/W	R/W	
复位值	0	0	0	0	0	0	0	0	

- 位 5——ICF1：T/C1 输入捕获中断标志位

当 T/C1 由外部引脚 ICP1 触发输入捕获时，ICF1 位被设为“1”。在 T/C1 运行方式为 PWM，寄存器 ICR1 内容作为计数器计数上限值时，一旦计数器 TCNT1 计数值与 ICR1 相等，也将置位 ICF1。当转入 T/C1 输入捕获中断向量执行中断处理程序时，ICF1 由硬件自动清零。写入一个逻辑“1”到 ICF1 标志位将清除该标志位。

- 位 4——OCF1A：T/C1 输出比较 A 匹配中断标志位

当 T/C1 输出比较 A 匹配成功（TCNT1=OCR1A）时，OCF1A 位被设为“1”。当转入 T/C1 输出比较 A 匹配中断向量执行中断处理程序时，OCF1A 由硬件自动清零。写入一个逻辑“1”到 OCF1A 标志位将清除该标志位。

设置强制输出比较 A 匹配（FOC1A=1）时，不会置位 OCF1A 标志位。

- 位 3——OCF1B：T/C1 输出比较 B 匹配中断标志位

当 T/C1 输出比较 B 匹配成功（TCNT1=OCR1B）时，OCF1B 位被设为“1”。当转入 T/C1 输出比较 B 匹配中断向量执行中断处理程序时，OCF1B 由硬件自动清零。写入一个逻辑“1”到 OCF1B 标志位将清除该标志位。

设置强制输出比较 B 匹配（FOC1B=1）时，不会置位 OCF1B 标志位。

- 位 2——TOV1：T/C1 溢出中断标志位

当 T/C1 产生溢出时，TOV1 位被设为“1”。当转入 T/C1 溢出中断向量执行中断处理程序时，TOV1 由硬件自动清零。写入一个逻辑“1”到 TOV1 标志位将清除该标志位。

TOV1 标志位置位的条件与 T/C1 的工作方式有关，详细见本节中有关 T/C1 各种运行方式的使用说明。

（6）T/C1 控制寄存器 A——TCCR1A

位	7	6	5	4	3	2	1	0	
$2F（$004F）	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
读/写	R/E	R/W	R/W	R/W	W	W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

- 位 7:6——COM1A[1:0]：比较 A 输出模式
- 位 5:4——COM1B[1:0]：比较 B 输出模式

这些位控制了比较输出引脚 OC1A（OC1B）的输出行为。如果 COM1A[1:0]中的任何一个位或两位被置“1”，OC1A（OC1B）的输出将覆盖 PB1（PB2）引脚的一般 I/O 端口功能，但是，OC1A（OC1B）输出的引脚的数据方向寄存器 DDR 位必须置为输出方式。当引脚 PB1（PB2）作为 OC1A（OC1B）输出引脚时，其输出方式取决于 COM1A[1:0]以及 WGM1[3:0]的设定。

表 2-28 给出了在 WGM1[3:0]的设置为一般模式和 CTC 模式（非 PWM）时，COM1A[1:0]（COM1B[1:0]）位的功能定义。表 2-29 给出了在 WGM1[3:0]的设置为快速 PWM 模式时，COM1A[1:0]（COM1B[1:0]）的功能定义。表 2-30 给出了在 WGM1[3:0]设置为相位可调以及相位频率可调的 PWM 模式时，COM1A[1:0]（COM1B[1:0]）位功能定义。

表 2-28 比较输出模式，非 PWM 模式

COM1A1/ COM1B1	COM1A0/ COM1B0	说 明
0	0	OC1A（OC1B）不占用引脚 PB1（PB2）
0	1	比较匹配时触发 OC1A/OC1B（OC1A/1B 为原 OC1A/1B 取反）
1	0	比较匹配时清零 OC1A/OC1B
1	1	比较匹配时置位 OC1A/0C1B

表 2-29 比较输出模式，快速 PWM 模式

COM1A1/ COM1B1	COM1A0/ COM1B0	说 明
0	0	OC1A/OC1B 不占用引脚 PB1/PB2
0	1	WGM13=0：OC1A（OC1B）不占用引脚 PB1（PB2） WGM13=1：比较匹配时触发 OC1A，OC1B 保留
1	0	比较匹配时置位 OC1A/OC1B，计数值为 TOP 时清零 OC1A/0C1B
1	1	比较匹配时清零 OC1A/OC1B，计数值为 TOP 时置位 OC1A/0C1B

表 2-30　比较输出模式，相位可调及相位频率可调 PWM 模式

COM1A1/COM1B1	COM1A0/COM1B0	说　明
0	0	OC1A/OC1B 不占用引脚 PB1/PB2
0	1	WGM13=0：OC1A/OC1B 不占用引脚 PB1/PB2 WGM13=1：比较匹配时触发 OC1A，OC1B 保留
1	0	向上计数过程中比较匹配时清零 OC1A/OC1B， 向下计数过程中比较匹配时置位 OC1A/OC1B
1	1	向上计数过程中比较匹配时置位 OC1A/OC1B， 向下计数过程中比较匹配时清零 OC1A/OC1B

- 位 3——FOC1A：强制输出比较 A
- 位 2——FOC1B：强制输出比较 B

FOC1A/FOC1B 位只在 WGM1[3:0]位被设置为非 PWM 模式下才有效，但为了保证同以后的器件兼容，在 PWM 模式下写 TCCR1A 寄存器时，该位必须被写“0”。当写一个逻辑“1”到 FOC1A/FOC1B 位时，将强加在波形发生器上一个比较匹配成功信号，使波形发生器依据 COM1A[1:0]/COM1B[1:0]位的设置而改变 OC1A/OC1B 输出状态。注意：FOC1A/FOC1B 的作用仅如同一个选通脉冲，而 OC1A/OC1B 的输出还是取决于 COM1A[1:0]/COM1B[1:0]位的设置。

一个 FOC1A/FOC1B 选通脉冲不会产生任何的中断申请，也不影响计数器 TCNT1 和寄存器 OCR1A 的值。读 FOC1A/FOC1B 总为零。

- 位 1:0——WGM1[1:0]：波形发生模式

这两个标志位与 WGM1[3:2]（位于寄存器 TCCR1B）相组合，用于控制 T/C1 的计数和工作方式——计数器计数的上限值和确定波形发生器的工作模式（见表 2-31）。T/C1 支持的工作模式有：一般模式，比较匹配时清零定时器(CTC)模式，以及两种脉宽调制(PWM)模式等 15 种。

（7）T/C1 控制寄存器 B——TCCR1B

位	7	6	5	4	3	2	1	0	
$2E ($004E)	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
读/写	R/E	R/W	R	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

- 位 7——ICNC1：输入捕获噪声抑制允许

设置 ICNC1 为“1”时，使能输入捕获噪声抑制功能。此时，外部引脚 ICP1 的输入捕获触发信号将通过噪声滤波抑制单元。使能该功能输入捕获触发信号比 ICP1 引脚上实际的触发信号（或由内部模拟比较器产生的触发信号）延时 4 个机器时钟周期。

- 位 6——ICES1：输入捕获触发方式选择

ICES1=0 时，外部引脚 ICP1 上逻辑电平变化的下降沿触发一次输入捕获；ICES1=1 时，

外部引脚 ICP1 上逻辑电平变化的上升沿触发一次输入捕获。

- 位 4:3——WGM1[3:2]：波形发生模式

这两个标志位与 WGM1[1:0]（位于寄存器 TCCR1A）相组合，用于控制 T/C1 的计数和工作方式。见寄存器 TCCR1A 功能描述。

表 2-31　波形产生模式

模　式	WGM1[3:0]	T/C1 工作模式	计数上限值（TOP）	OCR1A OCR1B 更新	TOV1 置位
0	0000	一般模式	0xFFFF	立即	0xFFFF
1	0001	8 位 PWM，相位可调	0x00FF	TOP	0x0000
2	0010	9 位 PWM，相位可调	0x01FF	TOP	0x0000
3	0011	10 位 PWM，相位可调	0x03FF	TOP	0x0000
4	0100	CTC	OCR1A	立即	0xFFFF
5	0101	8 位 PWM，快速	0x00FF	TOP	TOP
6	0110	9 位 PWM，快速	0x01FF	TOP	TOP
7	0111	10 位 PWM，快速	0x03FF	TOP	TOP
8	1000	PWM，相位和频率可调	ICR1	0x0000	0x0000
9	1001	PWM，相位和频率可调	OCR1A	0x0000	0x0000
10	1010	PWM，相位可调	ICR1	TOP	0x0000
11	1011	PWM，相位可调	OCR1A	TOP	0x0000
12	1100	CTC	ICR1	立即	0xFFFF
13	1101	保留	--	--	--
14	1110	PWM，快速	ICR1	TOP	TOP
15	1111	PWM，快速	OCR1A	TOP	TOP

- 位[2:0]——CS1[2:0]：T/C1 时钟源选择

这 3 个标志位用于选择 T/C1 的时钟源，见表 2-32。

表 2-32　T/C1 的时钟源选择

CS12	CS11	CS10	说　明
0	0	0	无时钟源（停止 T/C1）
0	0	1	$clk_{I/O}$（系统时钟）
0	1	0	$clk_{I/O}$/8（来自预分频器）
0	1	1	$clk_{I/O}$/64（来自预分频器）
1	0	0	$clk_{I/O}$/256（来自预分频器）
1	0	1	$clk_{I/O}$/1024（来自预分频器）
1	1	0	外部 T1 脚，下降沿驱动
1	1	1	外部 T1 脚，上升沿驱动

当选择使用外部时钟源时，无论 T1 引脚是否定义为输出功能，在 T1 脚上的逻辑信号

电平的变化都会驱动 C/T1 计数，这个特性允许用户通过软件来控制计数器。

3．16 位寄存器的读写操作步骤

T/C1 有 4 个 16 位的寄存器：TCNT1、OCR1A、OCR1B、ICR1。由于 AVR 的内部数据总线为 8 位，因此读写 16 位的寄存器需要分两次操作。为了能够同步读写 16 位寄存器，每一个 16 位寄存器分别配有一个 8 位的临时辅助寄存器（Temporary register），用于保存 16 位寄存器的高 8 位数据。要同步读写这些 16 位的寄存器，读写操作应遵循以下特定的步骤：

● 16 位寄存器的读操作

当 MCU 读取 16 位寄存器的低字节（低 8 位）时，16 位寄存器低字节内容被送到 MCU，而高字节（高 8 位）内容在读低字节操作的同时被放置于临时辅助（TEMP）寄存器中；当 MCU 读取高字节时，读到的是 TEMP 寄存器中的内容。因此，要同步读取 16 位寄存器中的数据，应先读取该寄存器的低位字节，再立即读取其高位字节。

● 16 位寄存器的写入操作

当 MCU 写入数据到 16 位寄存器的高位字节时，数据是写入到 TEMP 寄存器中；当 MCU 写入数据到 16 位寄存器的低位字节时，写入的 8 位数据与 TEMP 寄存器中的 8 位数据组合成一个 16 位数据，同步写入到 16 位寄存器中。因此，要同步写 16 位寄存器时，应先写入该寄存器的高位字节，再立即写入它的低位字节。

用户编写汇编程序时，如要对 16 位寄存器进行读写操作，应遵循以上特定的步骤。采用 C 等高级语言编写程序则可以直接对 16 位的寄存器进行操作，因为这些高级语言的编译系统会根据 16 位寄存器的操作步骤生成正确的执行代码。此外，在对 16 位寄存器操作时，最好将中断响应屏蔽，防止在主程序读写 16 位寄存器的两条指令之间插入一个含有对该寄存器操作的中断服务。如果这种情况发生，那么中断返回后，寄存器中的内容已经改变，会造成主程序中对 16 位寄存器的读写失误。

下面是一段读写 16 位寄存器的程序示例。

汇编代码：

```
TIME16_Read_WriteTCNT1:
;Save global interrupt flag
in r18,SREG
;Disable interrupts
cli
;Read TCNT1 into r17:r16
in r16,TCNT1L
in r17,TCNT1H
;Set TCNT1 to 0x01FF
ldi r17,0x01
ldi r16,0xFF
out TCNT1H,r17
```

```
out TCNT1l,r16
;Restore global interrupt flag
out SREG,r18
ret
```

C 程序代码：

```
unsigned int TIME16_Read_WriteTCNT1( void )
{
unsigned char sreg;
unsigned int i;
/* Save global interrupt flag */
sreg = SREG;
/* Disable interrupts */
_CLI();
/* Read TCNT1 into i */
i = TCNT1;
/* Set TCNT1 to 0x01FF */
TCNT1 = 0x01FF;
/* Restore global interrupt flag */
SREG = sreg;
return i;
}
```

4．T/C1 的应用

标志位 WGM1[3:0]和 COM1A[1:0]/COM1B[1:0]的组合构成 T/C1 的 16 种工作方式以及 OC1A/OC1B 不同模式的输出。

（1）一般模式（WGM1[3:0]=0）

T/C1 最简单的工作为一般模式，此时计数器为单向加 1 计数器，一旦寄存器 TCNT1 的计数值达到 0xFFFF（MAX），下一个计数脉冲到来时便恢复为 0x0000，并继续向上开始计数。在 TCNT1 为 0x0000 的同时，置溢出标志位 TOV1 为“1”。标志位 TOV1 可以用于申请中断，也可以作为计数器的第 17 位使用（使 T/C1 变成 17 位计数器）。用户可以在任何时候通过写入 TCNT1 寄存器初值来调整计数器溢出的时间间隔。

在一般模式中，可以方便地应用输入捕获功能，并正确地选择计数时钟源，使得两次输入捕获触发信号的时间间隔小于计数器完成一次单程计数的时间间隔。

在一般模式中，可以使用输出比较单元产生定时中断，但最好不要在一般模式下使用输出比较单元产生脉冲调制波形输出（PWM），因为这将占用过多的 MCU 的时间。

（2）比较匹配清零计数器 CTC 模式（WGM1[3:0]=4 或 WGM1[3: 0]=12）

T/C1 工作在 CTC 时，计数器为单向加 1 计数器，一旦寄存器 TCNT1 的值与 OCR1A

（WGM1[3:0]=4）或与 ICR1（WGM1[3:0]=12）的设定值相等，就将计数器 TCNT1 清零，然后继续向上加 1 计数。通过设置 OCR1A/ICR1 的值，可以方便地控制比较匹配输出的频率，也方便了外部事件计数的应用。图 2.34 为 CTC 模式的计数时序图。

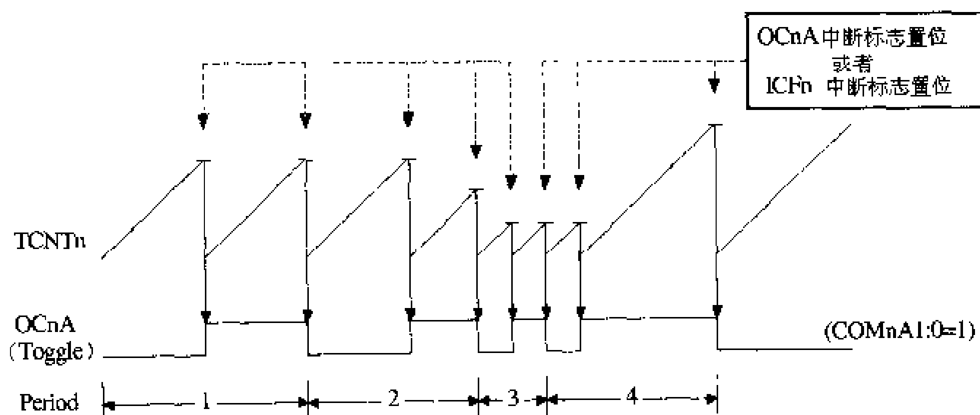


图 2.34 T/C1 的 CTC 模式计数时序

在 TCNT1 与 OCR1A/ICR1 匹配的同时，置比较匹配标志位 OCF1A/ICF1 为“1”。标志位 OCF1A/ICF1 可以用于申请中断。如果 MCU 响应该比较匹配中断，用户可以在中断服务程序中修改 OCR1A/ICR1 寄存器的值。

在 T/C1 采用的计数时钟频率较高，且写入 OCR1A/ICR1 的值与 0x0000 接近时，可能会丢失一次比较匹配成立条件。例如，当 TCNT1 的值与 OCR1A 相等，TCNT1 便被硬件清零并申请中断；在中断服务中重新改变设置 OCR1A 为 0x0005；但中断返回后 TCNT1 的计数值已经为 0x0010 了。此时便丢失了一次比较匹配成立条件，计数器将继续加一计数到 0xFFFF，然后返回 0x0000，当再次计数到 0x0005 时，才能产生比较匹配成功。

在 CTC 模式下产生波形输出时，应设置 OC1A 的输出方式为触发方式（COM1A[1:0]=1）。OC1A 输出波形的最高频率为 $F_{OC1A}=f_{clk_I/O}/2$（OCR1A=0x0000）。其他的输出频率由下式确定，式中 N 的取值为 1、8、64、256 或 1024。

$$f_{OC1A}=\frac{f_{clk_I/O}}{2N(1+OCR1A)}$$

除此之外，与一般模式相同，当计数器 TCNT1 的计数值由 0xFFFF 转到 0x0000 时，标志位 TOV1 置位。

（3）快速 PWM 模式（WGM1[3:0]=5，6，7，17 或 15）

T/C1 工作在快速 PWM 模式可以产生高速的 PWM 波形。当 T/C1 工作在此模式下时，计数器为单程向上的加 1 计数器，从 0x0000 一直加到 TOP，在下一计数脉冲到来时清零，然后再从 0x0000 开始加 1 计数。在设置正向比较匹配输出（COM1A[1:0]=2/COM1B[1:0]=2）模式中，当 TCNT1 的计数值与 OCR1A/OCR1B 的值相同匹配时置位 OC1A/OC1B，当计数器的值由 TOP 返回 0x0000 时清零 OC1A/OC1B。而在设置反向比较匹配输出（COM1A[1:0]=3/COM1B[1:0]=3）模式中，当 TCNT1 的计数值与 OCR1A/OCR1B 的值相同匹配时清零 OC1A/OC1B，当计数器的值由 TOP 返回 0x0000

时置位 OC1A/OC1B。图 2.35 为快速 PWM 工作时序图。

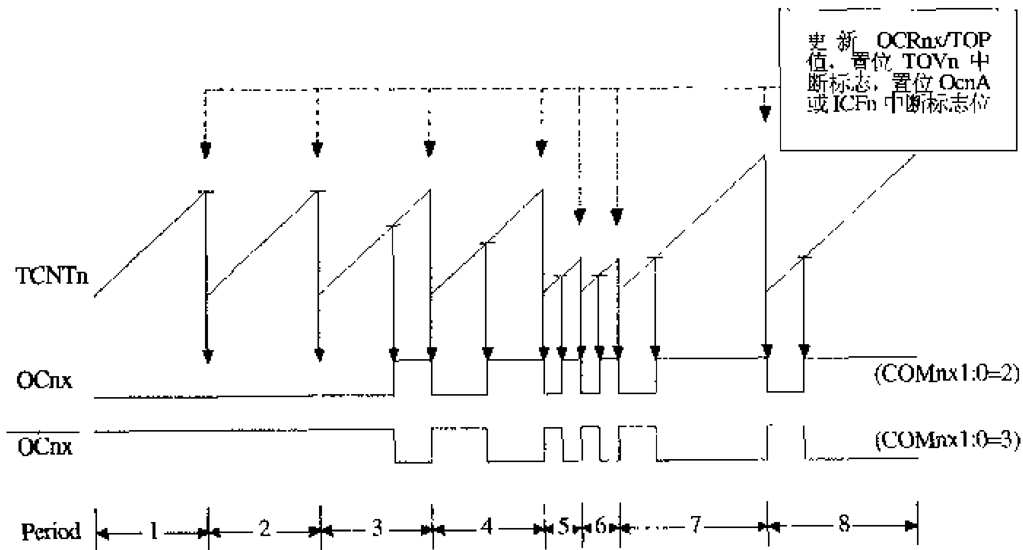


图 2.35 T/C1 快速 PWM 工作时序图

由于快速 PWM 模式采用单程计数方式，所以其产生 PWM 波的频率比另外两种 PWM 模式（相位可调以及相位频率可调）高一倍。因此快速 PWM 模式适用于电源调整、DAC 等应用。

快速 PWM 的精度（即 TOP 值）可以为固定的 8、9、10 位（0x00FF、0x01FF、0x03FF），或由寄存器 OCR1A、ICR1 设置值定义：最小精度为 2 位（OCR1A=0x0003 或 ICR1=0x0003），最大精度为 16 位（OCR1A=0xFFFF 或 ICR1=0xFFFF）。其他由 OCR1A 或 ICR1 设定值所定义的精度（单位 bits）可以由下式计算，式中 TOP 为寄存器 OCR1A/ICR1 的设定值。

$$R_{FPWM}=\log_2(TOP+1)$$

在 TCNT1 的计数值到达 TOP 时，置溢出标志位 TOV1 为“1”。此外，在使用寄存器 OCR1A/ICR1 的设定值作为计数器计数上限 TOP 值时，OC1A/ICF1 标志位也会与 TOV1 标志位一起置位。这些标志位都可以用于申请中断．如果响应中断，用户可以在中断服务程序中修改寄存器 OCR1A/ICR1 的值（即 TOP 值）。

当改变计数器的计数上限 TOP 值时，新的 TOP 值必须大于或等于比较寄存器 OCR1A/OCR1B 的设定比较值，否则比较匹配输出将不会发生。

使用寄存器 OCR1A/ICR1 的设定值作为计数器计数上限 TOP 值时，更新 ICR1 和 OCR1A 的过程是不同的。寄存器 ICR1 没有辅助缓冲器，因此当写入 ICR1 的设定 TOP 值小于当前计数器 TCNT1 的计数值时，将会丢失一次 TCNT1 与 TOP 相等匹配的产生，计数器要一直计数到 0xFFFF，再返回 0x0000 后，才能开始并产生与新的 TOP 值的比较匹配。寄存器 OCR1A 带有辅助缓冲器，当更新 OCR1A 时，数据只是写入到辅助缓冲器中，而 OCR1A 仍保持原 TOP 值。等到 TCNT1 与原 TOP 值相等匹配时，在 TCNT1 清零、TOV1 置位的同时，辅助缓冲器中数据才进入 OCR1A，使 OCR1A 真正得到更新。

因此，如不需要经常改变 TOP 值时，建议使用寄存器 ICR1 来设定计数器计数的上限

值，或采用固定的 8、9、10 位 TOP 值。这时，除了 OCR1B 外，寄存器 OCR1A 也可用于产生 PWM 脉冲（相当于有两个 PWM 输出）。如果在应用中需要经常改变计数器计数的上限 TOP 值，那么使用寄存器 OCR1A 作为 TOP 值的寄存器是最好的选择，但此时只能有一个 PWM 输出（OCR1B 控制的 OC1B 输出）。

在快速 PWM 模式下。OC1A/OC1B 输出 PWM 波形的频率输出由下式确定，式中 N 的取值为 1、8、64、256 或 1024。

$$f_{OC1APWM}=\frac{f_{clk_I/O}}{N(1+TOP)}$$

通过设置比较寄存器 OCR1A/OCR1B 的值，可以获得不同占空比的 PWM 脉冲波形。OCR1A/OCR1B 的一些特殊值会产生极端的 PWM 波形。当 OCR1A/OCR1B 的设置值与 0x0000 相近时，会产生窄脉冲序列。而设置 OCR1A/OCR1B 的值等于 TOP，OC1A/OC1B 的输出为恒定的高（低）电平。当设置 OCR1A 的值为 0x0000，且 OC1A 的输出方式为触发模式（COM1A[1:0]=1），OC1A 产生占空比为 50% 的最高频率 PWM 波形（$f_{OC1A}=f_{clk_I/O}/2$）。

（4）相位可调 PWM 模式（WGM1[3:0]=1，2，3，10 或 11）

相位可调 PWM 模式可以产生高精度相位可调的 PWM 波形。当 T/C1 工作在此模式下时，计数器为双程计数器：从 0x0000 一直加到 TOP，在下一个计数脉冲到达时，改变计数方向，从 TOP 开始减 1 计数到 0x0000。在设置正向比较匹配输出（COM1A[1:0]=2/COM1B[1:0]=2）模式下：正向加 1 过程中，TCNT1 的计数值与 OCR1A/OCR1B 的值相同匹配时清零 OC1A/OC1B；反向减 1 过程中，当计数器 TCNT1 的值与 OCR1A/OCR1B 相同时置位 OC1A/OC1B。设置反向比较匹配输出（COM1A[1:0]=3/COM1B[1:0]=3）模式下：正向加 1 过程中，TCNT1 的计数值与 OCR1A/OCR1B 的值相同匹配时置位 OC1A/OC1B；反向减 1 过程中，当计数器 TCNT1 的值与 OCR1A/OCR1B 相同时清零 OC1A/OC1B。图 2.36 为相位可调 PWM 工作时序图。

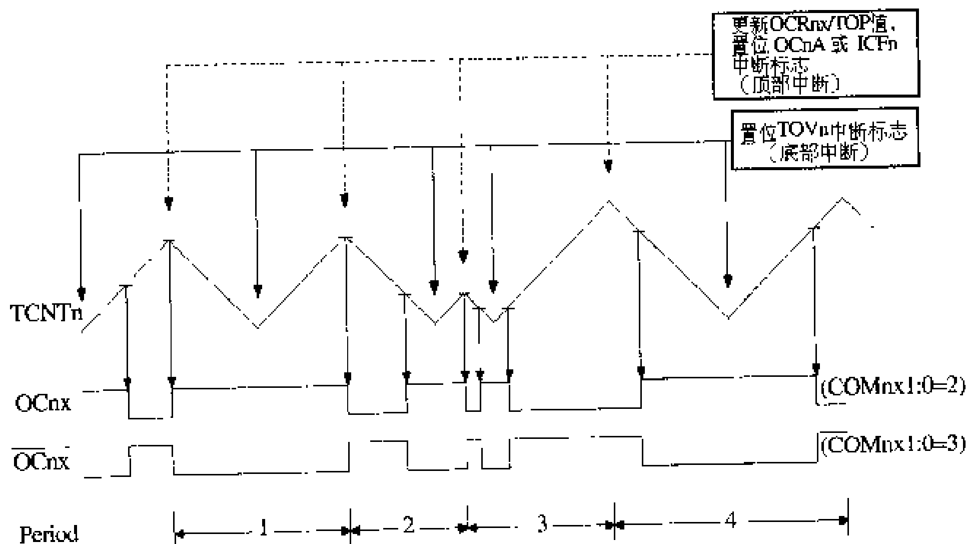


图 2.36　T/C1 相位可调 PWM 工作时序图

由于该 PWM 模式采用双程计数方式，所以它产生的 PWM 波的频率比快速 PWM 低。其相位可调的特性适用于马达控制一类的应用。

计数器计数上限 TOP 值的大/小决定了 PWM 输出频率的低/高，而比较寄存器的数值则决定了输出脉冲的起始相位和脉宽。

相位可调 PWM 的精度（即 TOP 值）可以为固定的 8、9、10 位（0x00FF、0x01FF、0x03FF），或由寄存器 OCR1A、ICR1 设置值定义。最小精度为 2 位（OCR1A=0x0003 或 ICR1=0x0003），最大精度为 16 位（OCR1A=0xFFFF 或 ICR1=0xFFFF）。其他由 OCR1A 或 ICR1 设定值所定义的精度（单位 bits）可以由下式计算，式中 TOP 为寄存器 OCR1A/ICR1 的设定值.

$$R_{PCPWM} = \log_2(TOP + 1)$$

在 TCNT1 的计数值到达 0x0000 时，置溢出标志位 TOV1 为“1”。如使用寄存器 OCR1A/ICR1 的设定值作为计数器计数上限 TOP 值，当在 TCNT1 计数到达 TOP 时，OC1A/ICF1 标志位置位，同时 OCR1A/OCR1B 自动更新（数据来源于各自的辅助缓冲器）。这些中断标志位都可以用于申请中断。

当改变计数器的计数上限 TOP 值时，新的 TOP 值必须大于或等于比较寄存器 OCR1A/OCR1B 的设定比较值，否则比较匹配输出将不会发生。

由于在相位可调 PWM 模式中，OCR1A/OCR1B 的更新发生在 TCNT1=TOP（即一个 PWM 的周期起始点在 TOP 处），因此，如果在应用中需要经常改变计数器计数的上限 TOP 值，那么建议使用频率周期可调 PWM 模式。

相位可调 PWM 模式中，OC1A/OC1B 输出的 PWM 波形的频率输出由下式确定，式中 N 的取值为 1、8、64、256 或 1024。

$$f_{OC1APCPWM} = \frac{f_{clk_I/O}}{2NTOP}$$

通过设置比较寄存器 OCR1A/OCR1B 的值，可以获得不同占空比的脉冲波型。OCR1A/OCR1B 的一些特殊值，会产生极端的 PWM 波形。当 OCR1A/OCR1B 的设置值与 0x0000 相近时，会产生窄脉冲序列。当 COM1A[1:0]=2/ COM1B[1:0]=2，且 OCR1A/OCR1B 的值为 TOP 时，OC1A/OC1B 的输出为恒定的高电平；而 OCR1A/OCR1B 的值为 0x0000 时，OC1A/OC1B 的输出为恒定的低电平。

（5）相位频率可调 PWM 模式（WGM1[3:0]=8 或 9）

相位频率可调 PWM 模式可以产生高精度相位和频率可调的 PWM 波形。当 T/C1 工作在此模式下时，计数器为双程计数器：从 0x0000 一直加到 TOP，在下一个计数脉冲到达时，改变计数方向，从 TOP 开始减 1 计数到 0x0000。在设置正向比较匹配输出（COM1A[1:0]=2/COM1B[1:0]=2）模式下：在正向加 1 过程中，TCNT1 的计数值与 OCR1A/OCR1B 的值相同匹配时清零 OC1A/OC1B；在反向减 1 过程中，当计数器 TCNT1 的值与 OCR1A/OCR1B 相同时置位 OC1A/OC1B。在设置反向比较匹配输出（COM1A[1:0]=3/COM1B[1:0]=3）模式下：在正向加 1 过程中，TCNT1 的计数值与 OCR1A/OCR1B 的值相同匹配时置位 OC1A/OC1B；在反向减 1 过程中，当计数器 TCNT1

的值与 OCR1A/OCR1B 相同时清零 OC1A/OC1B。图 2.37 为相位频率可调 PWM 工作时序图。

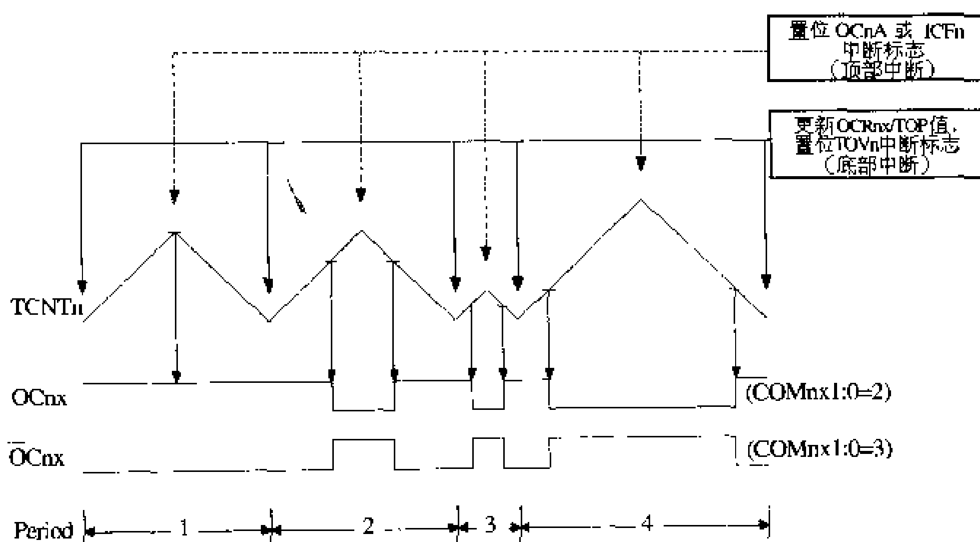


图 2.37　T/C1 相位频率可调 PWM 工作时序

由于该 PWM 模式采用双程计数方式，所以它产生的 PWM 波的频率比快速 PWM 低。其相位频率可调的特性适用于马达控制一类的应用。

计数器计数上限 TOP 值的大/小决定了 PWM 输出频率的低/高，而比较寄存器的数值则决定了输出脉冲的起始相位和脉宽。

相位频率可调 PWM 的精度（即 TOP 值）由寄存器 OCR1A、ICR1 设置值定义。最小精度为 2 位（OCR1A=0x0003 或 ICR1=0x0003），最大精度为 16 位（OCR1A=0xFFFF 或 ICR1=0xFFFF）。其他由 OCR1A 或 ICR1 设定值所定义的精度（单位 bits）可以由下式计算，式中 TOP 为寄存器 OCR1A/ICR1 的设定值。

$$R_{PFCPWM} = \log_2(TOP+1)$$

在 TCNT1 的计数值到达 0x0000 时，置溢出标志位 TOV1 为“1”，同时 OCR1A/OCR1B 自动更新，其更新数据来源于各自的辅助缓冲器。（注意：这是相位频率可调 PWM 模式与相位可调 PWM 模式的惟一区别之处，在相位可调 PWM 模式下，OCR1A/OCR1B 自动更新发生在 TCNT1 计数到达 TOP 时）。如使用寄存器 OCR1A/ICR1 的设定值作为计数器计数上限 TOP 值，当在 TCNT1 计数到达 TOP 时，OC1A/ICF1 标志位置位。这些中断标志位都可用于申请中断。

当改变计数器的计数上限 TOP 值时，新的 TOP 值必须大于或等于比较寄存器 OCR1A/OCR1B 的设定比较值，否则比较匹配输出将不会发生。

由于在相位频率可调 PWM 模式中，OCR1A/OCR1B 的更新发生在 TCNT1=0x0000（即一个 PWM 的周期起始点在 0x0000 处），因此，不管如何改变计数器计数的上限 TOP 值，都能产生对称的 PWM 输出波形，同时调整了频率。

使用寄存器 ICR1 来设定计数器计数的上限值，一般应用于固定频率 PWM 输出，这时，除了 OCR1B 外，寄存器 OCR1A 也可用于产生 PWM 脉冲（相当于有两个 PWM 输出）。

如果在应用中需要经常改变计数器计数的上限 TOP 值，那么使用寄存器 OCR1A 设定 TOP 值是最好的选择，但此时只能有一个 PWM 输出（OCR1B 控制的 OC1B 输出）。

相位频率可调 PWM 模式中，OC1A/OC1B 输出的 PWM 波形的频率输出由下式确定，式中 N 的取值为 1、8、64、256 或 1024。

$$f_{OC1APFCPWM} = \frac{f_{clk_I/O}}{2NTOP}$$

通过设置比较寄存器 OCR1A/OCR1B 的值，可以获得不同占空比的脉冲波形。OCR1A/OCR1B 的一些特殊值，会产生极端的 PWM 波形。当 OCR1A/OCR1B 的设置值与 0x0000 相近时，会产生窄脉冲序列。当 COM1A[1:0]=2/ COM1B[1:0]=2，且 OCR1A/OCR1B 的值为 TOP 时，OC1A/OC1B 的输出为恒定的高电平；而当 OCR1A/OCR1B 的值为 0x0000 时，OC1A/OC1B 的输出为恒定的低电平。

5．T/C1 计数器的计数时序

图 2.38、图 2.39、图 2.40 和图 2.41 给出了 T/C1 在同步工作情况下的各种计数时序，同时给出标志位 TOV1 和 OCF1A/OCF1B 的置位条件以及寄存器 OCR1A/OCR1B 的更新位置。

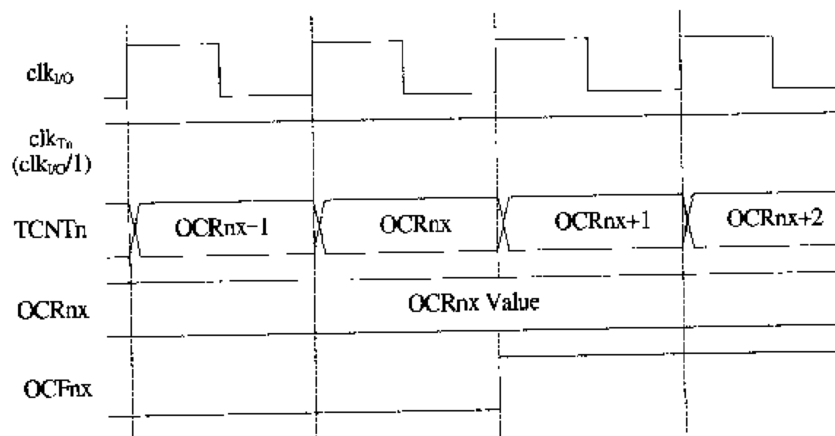


图 2.38　T/C1 计数时序，OCFnx 置位，无预分频

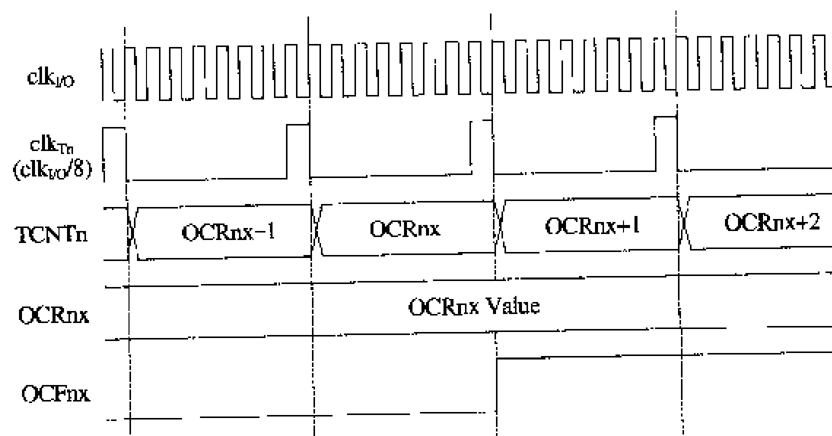


图 2.39　T/C1 计数时序，OCFnx 置位，带 1/8 预分频

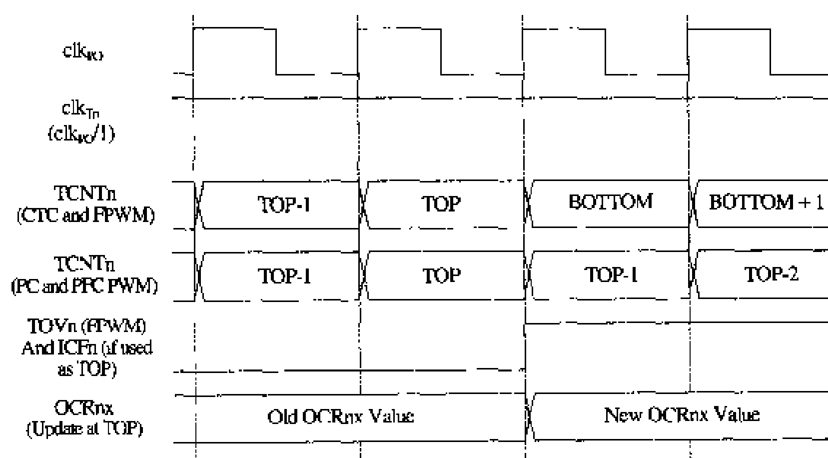


图 2.40 T/C1 计数时序，TOVn（ICFn）置位，OCRnx 更新，无预分频

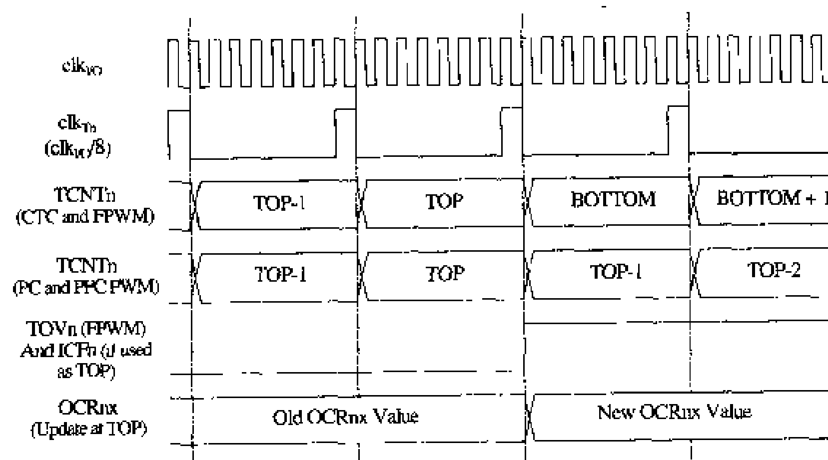


图 2.41 T/C1 计数时序，TOVn（ICFn）置位，OCRnx 更新，带 1/8 预分频

在图 2.40 和图 2.41 中，给出了在相位可调 PWM 和快速 PWM 两种模式下，当计数器 TCNTn 的计数值接近 TOP 时的计数时序。而在频率相位可调 PWM 模式下，OCRnx 的更新是在计数器 TCNTn 的计数值到达 BOTTOM 时，其计数时序与图 2.40 和图 2.41 类似，区别仅在于把 TOP 替换为 BOTTOM，TOP−1 替换为 BOTTOM+1，…，同时 TOVn 在 BOTTOM 处置位。

2.9.4 8 位定时器/计数器 2——T/C2

ATmega8 的 T/C2 是一个 8 位的多功能定时/计数器，其主要特点是：

- 单通道计数器；
- 比较匹配时清零定时器（自动重装特性，Auto Reload）；
- 无奇边非对称，相位可调的脉宽调制（PWM）；
- 频率发生器；
- 10 位的时钟预分频器；

- 溢出和比较匹配中断源；
- 允许使用从外部引脚的 32kHz 手表晶振作为独立的计数时钟源（实时时钟源）。

1. T/C2 的组成结构

图 2.42 为 8 位 T/C2 的结构框图。在图中给出了 MCU 可以操作的寄存器以及相关的标志位。其中，定时/计数器寄存器 TCNT2 和输出比较寄存器 OCR2 都是 8 位的寄存器；中断请求信号可以在定时器中断标志寄存器 TIFR 中找到。在定时器中断屏蔽寄存器 TIMSK 中，可以找到各自独立的中断屏蔽位。

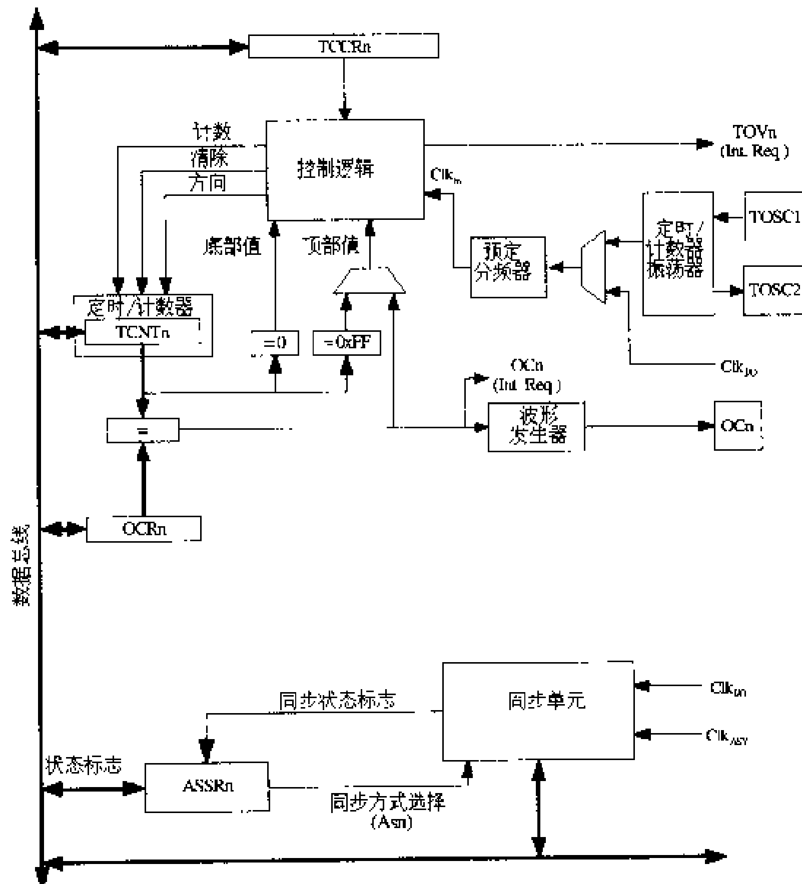


图 2.42 8 位 T/C2 的结构框图

（1）8 位 T/C2 的计数单元

T/C2 的计数单元是一个可编程的 8 位双向计数器，图 2.43 为它的逻辑功能图。根据计数器的工作模式，在每一个 clk_{T2} 时钟到来时，计数器进行加 1、减 1 或清零操作。clk_{T2} 的来源由标志位 CS2[2:0]设定。当 CS2[2:0]=0 时，计数器停止计数（无计数时钟源）。

图中的符号所代表的意义如下：

- 计数（count） TCNT2 加 1 或减 1。
- 方向（direction） 加或减的选择。
- 清除（clear） 清零 TCNT2。
- 顶部值（TOP） 表示 TCNT2 计数值到达最大值。
- 底部值（BOTTOM） 表示 TCNT2 计数值到达最小值（零）。

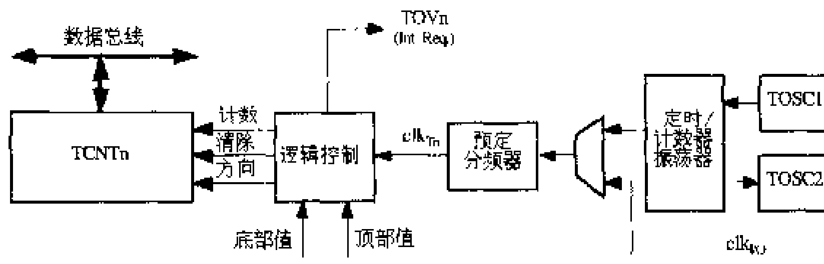


图 2.43 T/C2 逻辑功能图

计数值保存在寄存器 TCNT2 中，MCU 可以在任何时间访问读/写 TCNT2。MCU 写入 TCNT2 的值将立即覆盖其中原有的内容，并会影响计数器的运行。

计数器的计数序列取决于寄存器 TCCR2 中标志位 WGM2[2:0]的设置。WGM2[2:0]的设置直接影响到计数器的计数方式和 OC2 的输出，同时也影响和涉及 T/C2 的溢出标志位 TOV2 的置位。标志位 TOV2 可以用于产生中断申请。

（2）输出比较单元

图 2.44 为 T/C2 的输出比较单元逻辑功能图。在 T/C2 运行期间，输出比较单元一直将寄存器 TCNT2 的计数值同寄存器 OCR2 的内容进行比较。一旦两者相等，在下一个计数时钟脉冲到达时置位 OCF2 标志位。根据 WGM2[1:0]和 COM2[1:0]的不同设置，比较相等匹配的输出还控制产生各种类型的脉冲波形。

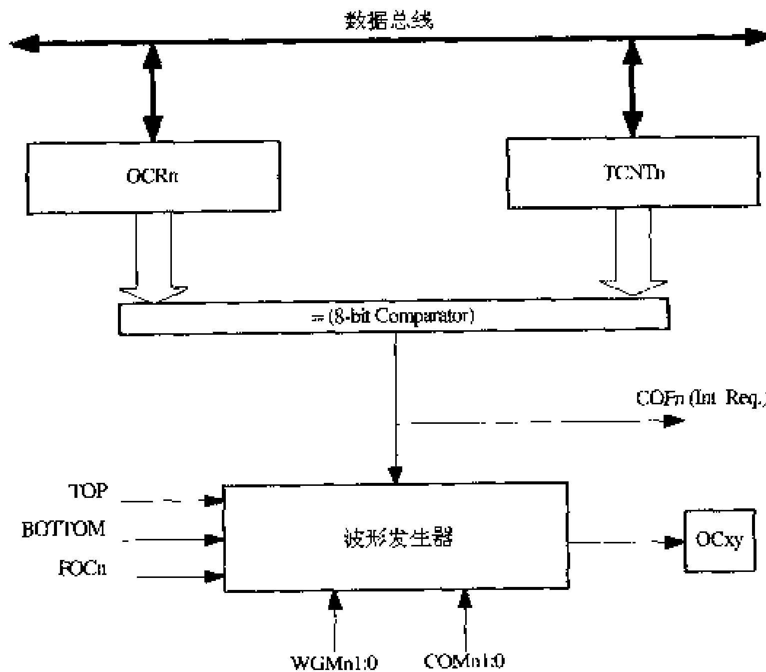


图 2.44 T/C2 输出比较单元逻辑功能图

寄存器 OCR2 配置了一个辅助缓存器。当 T/C2 工作在非 PWM 模式下时，该辅助缓存器被禁止，MCU 直接访问操作寄存器 OCR2。当 T/C2 工作在 PWM 模式时，该辅助缓存器投入使用，这时 MCU 对 OCR2 的访问操作，实际上是在对 OCR2 的辅助缓存器操作。当计数器的计数值达到设定的最大值（TOP）或最小值（BOTTOM）时，辅助缓存器中的

内容将同步更新比较寄存器 OCR2 的值。这有效地防止了产生奇边非对称的 PWM 脉冲信号，使输出的 PWM 波中没有杂散脉冲。

（3）比较匹配输出单元

标志位 COM2[1:0]有两个作用：定义 OC2 的输出状态，以及控制外部引脚 OC2 是否输出 OC2 寄存器的值。图 2.45 为比较匹配输出单元的逻辑图。

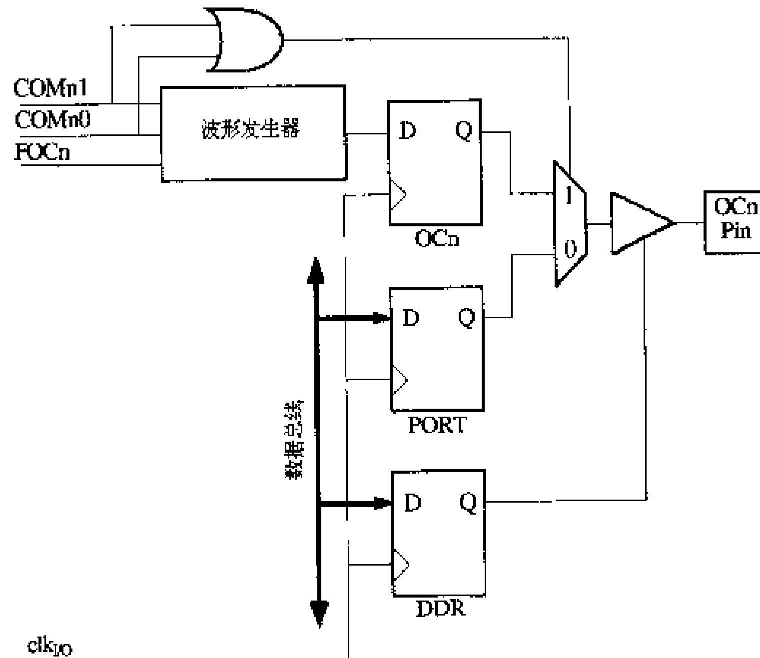


图 2.45 T/C2 比较匹配输出单元逻辑图

当标志位 COM2[1:0]中任何一位为“1”时，波形发生器的输出 OC2 取代引脚原来的 I/O 功能，但引脚的方向寄存器 DDR 仍然控制 OC2 引脚的输入/输出方向。如果要在外部引脚输出 OC2 的逻辑电平，应设定 DDR 定义该引脚为输出脚。采用这种结构，用户可以先初始化 OC2 的状态，然后再允许其由引脚输出。

（4）比较输出模式和波形发生器

在四种工作模式下，根据 COM2[1:0]的不同设定，波形发生器将产生各种不同的脉冲波形。但只要 COM2[1:0]=0，波形发生器对 OC2 寄存器没有任何作用。

2．T/C2 的时钟源与预定比例分频器

T/C2 的时钟源可来自芯片内部时钟源，也可来自外部引脚 TOSC1 和 TOSC2 的异步时钟源（通过寄存器 ASSR 的 AS2 设定）。图 2.46 所示为 T/C2 的时钟源和预定比例分频器。

C/T2 的时钟源称为 clk_{T2S}。clk_{T2S} 默认连接到系统 I/O 时钟 $clk_{I/O}$。通过设置 ASSR 寄存器中的 AS2 位，C/T2 可以使用 TOSC1 引脚上的时钟作为时钟源。这允许 C/T2 作为实时时钟计数器（RTC）使用。当 AS2 被置位时，TOSC1 和 TOSC2 引脚将脱离 PORTB，一个晶振可以连接在 TOSC1 和 TOSC2 引脚之间，作为 T/C2 的独立时钟源。片内的振荡器电路对 32.768kHz 的表用晶振进行了优化，因此最好不要在引脚 TOSC1 上单独施加一个外部时钟源。clk_{T2S} 经过 C/T2 的 10 位预定比例分频器产生 $clk_{T2S}/8$、$clk_{T2S}/32$、$clk_{T2S}/64$、

$clk_{T2S}/128$、$clk_{T2S}/256$ 和 $clk_{T2S}/1024$ 的时钟信号，以上时钟信号均可作为 T/C2 的时钟源。此外，还可以直接选择 clk_{T2S} 作为时钟源，或者无时钟源（C/T2 停止）。通过置 SFIOR 寄存器中的 PSR2 位为“1”可以复位预分频器，这允许用户对预分频器进行可预置的操作。

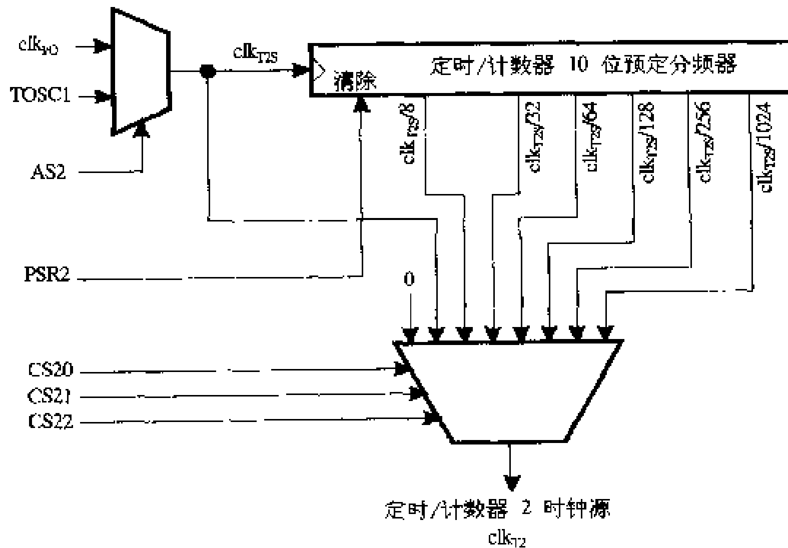


图 2.46 T/C2 的时钟源和预定比例分频器

3．8 位 T/C2 的寄存器

（1）T/C2 计数寄存器——TCNT2

位	7	6	5	4	3	2	1	0	
\$24（\$0044）	TCNT2[7..0]								TCNT2
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

TCNT2 是 T/C2 的计数值寄存器，该寄存器可以直接被读写访问。写 TCNT2 寄存器将在下一个定时器时钟周期中阻塞比较匹配。因此，在计数器运行期间修改 TCNT2 的内容，有可能将丢失一次 TCNT2 与 OCR2 的匹配比较操作。

（2）输出比较寄存器——OCR2

位	7	6	5	4	3	2	1	0	
\$23（\$0043）	OCR2[7..0]								OCR2
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

该寄存器中的 8 位数据用于同 TCNT2 寄存器中的计数值进行连续的匹配比较。一旦 TCNT2 的计数值与 OCR2 的数据匹配相等，将产生一个输出比较匹配相等的中断申请，或改变 OC2 的输出逻辑电平。

（3）定时/计数器中断屏蔽寄存器——TIMSK

位	7	6	5	4	3	2	1	0	
$39（$0059）	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	-	TOIE0	TIMSK
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

● 位 7——OCIE2：T/C2 输出比较匹配中断允许标志位

当 OCIE2 被设为“1”，且状态寄存器中的 I 位被设为“1”时，将使能 T/C2 的输出比较匹配中断。若在 T/C2 上发生输出比较匹配时（OCF2=1），则执行 T/C2 输出比较匹配中断服务程序。

● 位 6——TOIE2：T/C2 溢出中断允许标志位

当 TOIE2 被设为“1”，且状态寄存器中的 I 位被设为“1”时，将使能 T/C2 溢出中断。若在 T/C2 上发生溢出时（TOV2=1），则执行 T/C2 溢出中断服务程序。

（4）定时/计数器中断标志寄存器——TIFR

位	7	6	5	4	3	2	1	0	
$38（$0058）	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	-	TOV0	TIFR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

● 位 7——OCF2：T/C2 输出比较匹配中断标志位

当 T/C2 输出比较匹配成功（TCNT2=OCR2）时，OCF2 位被设为“1”。当转入 T/C2 输出比较匹配中断向量执行中断处理程序时，OCF2 由硬件自动清零。写入一个逻辑“1”到 OCF2 标志位将清除该标志位。当寄存器 SREG 中的 I 位、OCIE2 以及 OCF2 均为“1”时，T/C2 的输出比较匹配中断被执行。

● 位 6——TOV2：T/C2 溢出中断标志位

当 T/C2 产生溢出时，TOV2 位被设为“1”。当转入 T/C2 溢出中断向量执行中断处理程序时，TOV2 由硬件自动清零。写入一个逻辑“1”到 TOV2 标志位将清除该标志位。当寄存器 SREG 中的 I 位、TOIE2 以及 TOV2 均为“1”时，T/C2 的溢出中断被执行。在 PWM 模式中，当 T/C2 计数器的值为 0x00 并改变计数方向时，TOV2 被置为“1”。

（5）特殊功能 I/O 寄存器——SFIOR

位	7	6	5	4	3	2	1	0	
$30（$0050）	-	-	-	ADHSM	ACME	PUD	PSR2	PSR10	SFIOR
读/写	R	R	R	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

● 位 1——PSR2：T/C2 预定比例分频器复位

当写入“1”到该位，将复位 T/C2 预定比例分频器，一旦预定比例分频器复位，硬件自动清零该标志位。而写“0”到该位，则不会产生任何操作。当 T/C2 使用内部时钟源时，读取 PSR2 位的值，总是为“0”。当 T/C2 工作在异步方式下，写“1”到该位后，PSR2 将一直保持为“1”，直到预定比例分频器复位。

（6）T/C2 控制寄存器——TCCR2

位	7	6	5	4	3	2	1	0	
$25（$0045）	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	TCCR2
读/写	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

● 位 7——FOC2：强制输出比较

FOC2 位只在 WGM 位被设置为非 PWM 模式下才有效，但为了保证同以后的器件兼容，在 PWM 模式下写 TCCR2 寄存器时，该位必须被写零。当写一个逻辑“1”到 FOC2 位时，将强加在波形发生器上一个比较匹配成功信号，使波形发生器依据 COM2[1:0]位的设置而改变 OC2 输出状态。注意：FOC2 的作用仅如同一个选通脉冲，而 OC2 的输出还是取决于 COM2[1:0]位的设置。

一个 FOC2 选通脉冲不会产生任何的中断申请，也不影响计数器 TCNT2 和寄存器 OCR2 的值。一旦一个真正的比较匹配发生，OC2 的输出将根据 COM2[1:0]位的设置而更新。

● 位 3，6——WGM2[1:0]：波形发生模式

这两个标志位控制 T/C2 的计数和工作方式：计数器计数的上限值和确定波形发生器的工作模式（见表 2-33）。T/C2 支持的工作模式有：一般模式，比较匹配时定时器清零（CTC）模式，以及两种脉宽调制（PWM）模式。

表 2-33 波形产生模式

模 式	WGM21	WGM20	T/C2 工作模式	计数上限值	OCR2 更新	TOV2 置位
0	0	0	一般模式	0xFF	立即	0xFF
1	0	1	PWM，相位可调	0xFF	0xFF	0x00
2	1	0	CTC 模式	OCR2	立即	0xFF
3	1	1	快速 PWM	0xFF	0xFF	0xFF

● 位 5:4——COM2[1:0]：比较匹配输出模式

这些位控制比较输出引脚 OC2 的输出行为。如果 COM2[1:0]中的任何一个位或两位被置“1”，OC2 的输出将覆盖 PB3 引脚的一般 I/O 端口功能，但是，OC2 输出的引脚的数据方向寄存器 DDR 位必须置为输出方式。当引脚 PB3 作为 OC2 输出引脚时，其输出方式取决于 COM2[1:0]和 WGM2[1:0]的设定。

表 2-34 给出了在 WGM2[1:0]的设置为一般模式和 CTC 模式（非 PWM）时，COM2[1:0]位的功能定义。表 2-35 给出了在 WGM2[1:0]的设置为快速 PWM 模式时，COM2[1:0]位的功能定义。表 2-36 给出了在 WGM2[1:0]设置为相位可调的 PWM 模式时，COM2[1:0]位功能定义。

表 2-34 比较输出模式，非 PWM 模式

COM21	COM20	说 明
0	0	OC2 不占用引脚 PB3
0	1	比较匹配时触发 OC2（OC2 为原 OC2 取反）

续表

COM21	COM20	说　明
1	0	比较匹配时清零 OC2
1	1	比较匹配时置位 OC2

表 2-35　比较输出模式，快速 PWM 模式

COM21	COM20	说　明
0	0	OC2 不占用引脚 PB3
0	1	保留
1	0	比较匹配时清零 OC2，计数值为 0xFF 时置位 OC2
1	1	比较匹配时置位 OC2，计数值为 0xFF 时清零 OC2

表 2-36　比较输出模式，相位可调 PWM 模式

COM21	COM20	说　明
0	0	OC2 不占用引脚 PB3
0	1	比较匹配时触发 OC2（OC2 为原 OC2 取反）
1	0	向上计数过程中比较匹配时清零 OC2 向下计数过程中比较匹配时置位 OC2
1	1	向上计数过程中比较匹配时置位 OC2 向下计数过程中比较匹配时清零 OC2

● 位[2:0]——CS2[2:0]：T/C2 时钟源选择

这 3 个标志位用于选择 T/C2 的时钟源，见表 2-37。

表 2-37　T/C2 的时钟源选择

CS22	CS21	CS20	说　明
0	0	0	无时钟源（停止 T/C2）
0	0	1	clk_{T2S}（不经过分频器）
0	1	0	$clk_{T2S}/8$（来自预分频器）
0	1	1	$clk_{T2S}/32$（来自预分频器）
1	0	0	$clk_{T2S}/64$（来自预分频器）
1	0	1	$clk_{T2S}/128$（来自预分频器）
1	1	0	$clk_{T2S}/256$（来自预分频器）
1	1	1	$clk_{T2S}/1024$（来自预分频器）

4．T/C2 的应用

标志位 WGM2[1:0]和 COM2[1:0]的组合构成 T/C2 的四种工作方式以及 OC2 不同模式的输出。

（1）一般模式（WGM2[1:0]=0）

T/C2 工作在一般模式下时，计数器为单向加 1 计数器，一旦寄存器 TCNT2 的值到达 0xFF，下一个计数脉冲到来时便恢复为 0x00，并继续向上开始计数。在 TCNT2 为 0x00 的同时，置溢出标志位 TOV2 为“1”。标志位 TOV2 可以用于申请中断，也可以作为计数器的第 9 位使用（使 T/C2 变成 9 位计数器）。用户可以在任何时候通过写入 TCNT2 寄存器初值来调整计数器溢出的时间间隔。

在一般模式中，可以使用输出比较单元产生定时中断信号。但最好不要在一般模式下使用输出比较单元来产生 PWM 波形输出，因为这将占用过多的 MCU 的时间。

（2）比较匹配清零计数器 CTC 模式（WGM2[1:0]=2）

T/C2 工作在 CTC 模式下时，计数器为单向加 1 计数器，一旦寄存器 TCNT2 的值与 OCR2 的设定值相等，就将计数器 TCNT2 清零，然后继续向上加 1 计数。通过设置 OCR2 的值，可以方便地控制比较匹配输出的频率，也方便了外部事件计数的应用。图 2.47 为 CTC 模式的计数时序图。

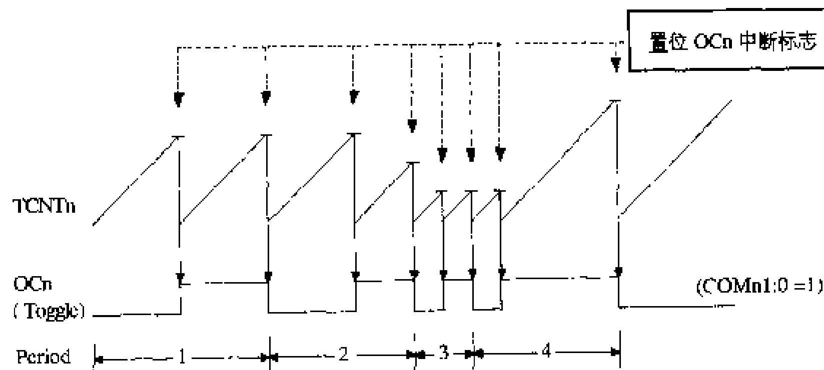


图 2.47　T/C2 的 CTC 模式计数时序

在 TCNT2 与 OCR2 匹配的同时，置比较匹配标志位 OCF2 为“1”。标志位 OCF2 可以用于申请中断。如果响应比较匹配中断，用户可以在中断服务程序中修改 OCR2 的值。

在 T/C2 采用的计数时钟频率比较高时，当写入 OCR2 的值与 0x00 接近，可能会丢失一次比较匹配成立条件。例如：当 TCNT2 的值与 OCR2 相等，TCNT2 被硬件清零并申请中断；在中断服务中重新改变设置 OCR2 为 0x05；但中断返回后 TCNT2 的计数值已经为 0x10 了。此时丢失了一次比较匹配成立条件，计数器将继续加 1 计数到 0xFF，然后返回 0x00，当再次计数到 0x05 时，才能产生比较匹配成功。

在 CTC 模式下产生波形输出时，应设置 OC2 的输出方式为触发方式（COM2[1:0]=1）。OC2 输出波形的最高频率为 $f_{OC2}=f_{clk_I/O}/2$（OCR2=0x00）。其他的频率输出由下式确定，式中 N 的取值为 1、8、32、64、128、256 或 1024。

$$f_{OC2} = \frac{f_{clk_I/O}}{2N(1 + OCR2)}$$

除此之外，与一般模式相同，当计数器的计数值由 0xFF 转到 0x00 时，标志位 TOV2 置位。

（3）快速 PWM 模式（WGM2[1:0]=3）

T/C2 工作在快速 PWM 模式可以产生较高频率的 PWM 波形。当 T/C2 工作在此模式下时，计数器为单程向上的加 1 计数器，从 0x00 一直加到 0xFF，然后再从 0x00 开始加 1 计数。在设置正向比较匹配输出（COM2[1:0]=2）模式中，当 TCNT2 的计数值与 OCR2 的值相同匹配时清零 OC2，当计数器的值由 0xFF 返回 0x00 时置位 OC2。而在设置反向比较匹配输出（COM2[1:0]=3）模式中，当 TCNT2 的计数值与 OCR2 的值相同匹配时置位 OC2，当计数器的值由 0xFF 返回 0x00 时清零 OC2。图 2.48 为快速 PWM 工作时序图。

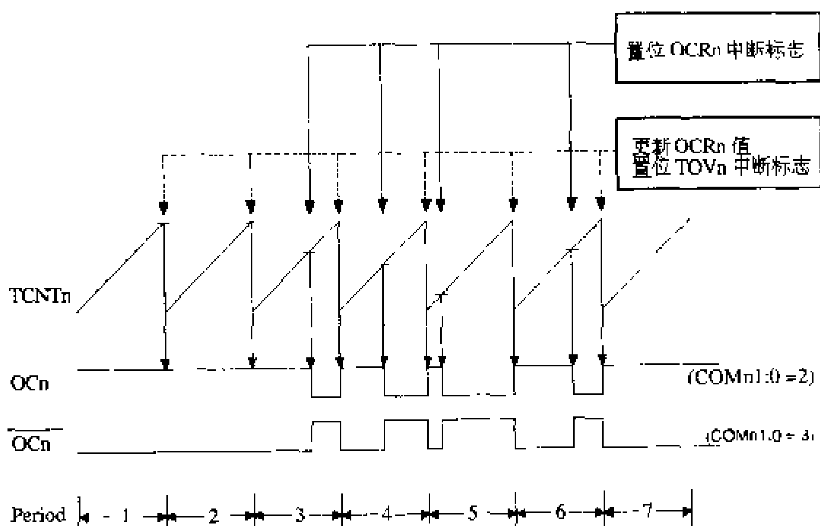


图 2.48 T/C2 快速 PWM 工作时序

由于快速 PWM 模式采用单程计数方式，所以它可以产生比相位可调 PWM 模式高一倍频率的 PWM 波。因此快速 PWM 模式适用于电源调整、DAC 等应用。

在 TCNT2 的计数值到达 0xFF 时，置溢出标志位 TOV2 为“1”。标志位 TOV2 可以用于申请中断。如果响应溢出中断，用户可以在中断服务程序中修改 OCR2 的值。

OC2 输出的 PWM 波形的频率输出由下式确定，式中 N 的取值为 1、8、32、64、128、256 或 1024。

$$f_{OC2PWM} = \frac{f_{clk_I/O}}{256N}$$

通过设置寄存器 OCR2 的值，可以获得不同占空比的脉冲波形。OCR2 的一些特殊值，会产生极端的 PWM 波形。当 OCR2 的设置值与 0x00 或 0xFF 相近时，会产生窄脉冲序列。而设置 OCR2 的值为 0xFF，OC2 的输出为恒定的高（低）电平。当设置 OCR2 的值为 0x00，且 OC2 的输出方式为触发模式（COM2[1:0]=1），T/C2 产生占空比为 50%的最高频率 PWM 波形 $f_{OC2}=f_{clk_I/O}/2$。

（4）相位可调 PWM 模式（WGM2[1:0]=1）

相位可调 PWM 模式可以产生高精度相位可调的 PWM 波形。当 T/C2 工作在此模式下时，计数器为双程计数器：从 0x00 一直加到 0xFF，在下一个计数脉冲到达时，改变计数方向，从 0xFF 开始减 1 计数到 0x00。设置正向比较匹配输出（COM2[1:0]=2）模式：在

正向加 1 过程中，TCNT2 的计数值与 OCR2 的值相同匹配时清零 OC2；在反向减 1 过程中，当计数器 TCNT2 的值与 OCR2 相同时置位 OC2。设置反向比较匹配输出（COM2[1:0]=3）模式：在正向加 1 过程中，TCNT2 的计数值与 OCR2 的值相同匹配时置位 OC2；在反向减 1 过程中，当计数器 TCNT2 的值与 OCR2 相同时清零 OC2。图 2.49 为相位可调 PWM 工作时序图。

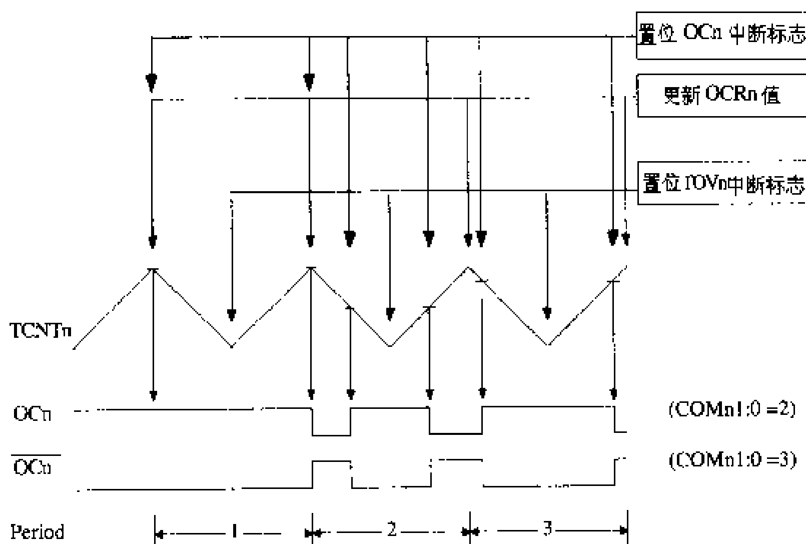


图 2.49　T/C2 相位可调 PWM 工作时序

由于该 PWM 模式采用双程计数方式，所以它产生的 PWM 波的频率比快速 PWM 低。其频率可调的特性（即 OC2 逻辑电平的改变不是固定在 TCNT2=0x00 处），适用于马达控制一类的应用。

在 TCNT2 的计数值到达 0x00 时，置溢出标志位 TOV2 为“1”。标志位 TOV2 可以用于申请溢出中断。

OC2 输出的 PWM 波形的频率输出由下式确定，式中 N 的取值为 1、8、32、64、128、256 或 1024。

$$f_{OC2PCPWM} = \frac{f_{clk\ I/O}}{512N}$$

通过设置寄存器 OCR2 的值，可以获得不同占空比的脉冲波形。OCR2 的一些特殊值，会产生极端的 PWM 波形。当 OCR2 的设置值与 0x00 或 0xFF 相近时，会产生窄脉冲序列。当 COM2[1:0]=2 且 OCR2 的值为 0xFF 时，OC2 的输出为恒定的高电平；而 OCR2 的值为 0x00 时，OC2 的输出为恒定的低电平。

5．T/C2 计数器的计数时序

图 2.50、图 2.51、图 2.52 和图 2.53 给出了 T/C2 在同步工作情况下的各种计数时序，同时给出标志位 TOV2 和 OCF2 的置位条件。

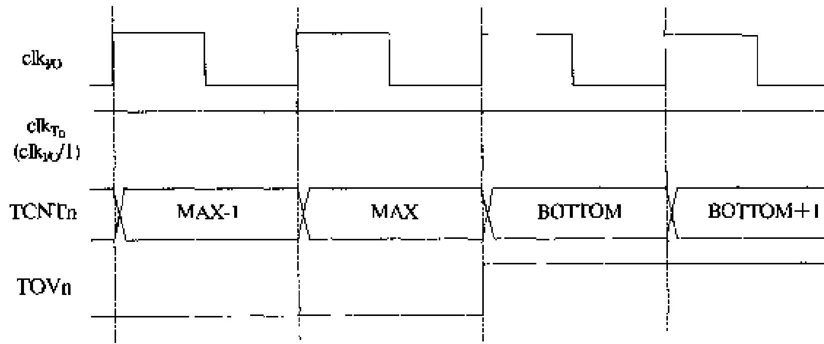


图 2.50 T/C2 计数时序，TOVn 置位，无预分频

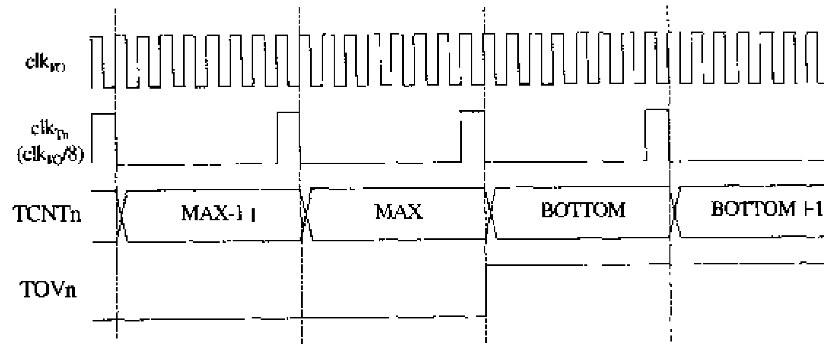


图 2.51 T/C2 计数时序，TOVn 置位，带 1/8 预分频

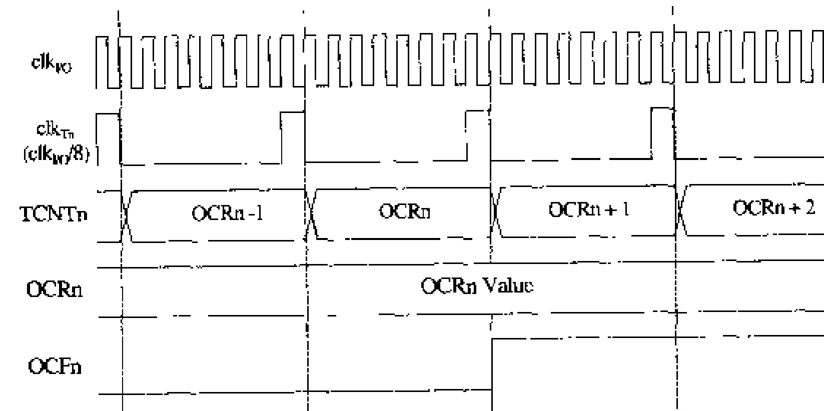


图 2.52 T/C2 计数时序，OCFn 置位，带 1/8 预分频（CTC 模式除外）

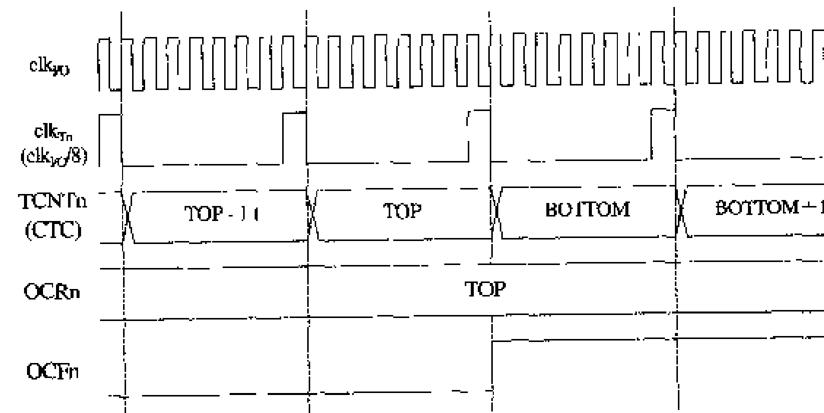


图 2.53 T/C2 计数时序，OCFn 置位，带 1/8 预分频（CTC 模式）

6. T/C2 的异步操作方式

（1）异步状态寄存器——ASSR

位	7	6	5	4	3	2	1	0	
$22（$0042）					AS2	TCN2UB	OCR2UB	TCR2UB	ASSR
读/写	R	R	R	R	R/W	R	R	R	
复位值	0	0	0	0	0	0	0	0	

- 位 3——AS2：异步 T/C2 设定位

当 AS2 写为“0”时，T/C2 使用系统 I/O 时钟——$clk_{I/O}$ 作为时钟源（同步方式）。当 AS2 写为“1”时，T/C2 使用连接在 TOSC1 引脚上的晶振作为时钟源（异步方式）。当 AS2 的值被改变时，寄存器 TCNT2、OCR2 和 TCCR2 的内容会被改变。

- 位 2——TCN2UB：TCNT2 更新忙

当 T/C2 采用异步工作方式，同时写入 TCNT2 时，该位被置“1”。当 TCNT2 已经由临时缓存寄存器更新完成，该位由硬件自动清零。该位为“0”表示 TCNT2 可以被更新。

- 位 1——OCR2UB：OCR2 更新忙

当 T/C2 采用异步工作方式，同时写入 OCR2 时，该位置“1”。当 OCR2 已经由临时缓存寄存器更新时，该位将被硬件清零。该位为“0”时表示 OCR2 可以被更新。

- 位 0——TCR2UB：TCCR2 更新忙

当 T/C2 采用异步工作方式，同时写入 TCCR2 时，该位置位。当 TCCR2 已经由临时缓存寄存器更新时，该位将由硬件自动清零。该位为“0”时表示 TCCR2 可以被更新。

如果上述这些更新忙标志位为“1”时，对 T/C2 的 3 个寄存器进行写操作，则写入的值可能会失败，并引起意外的中断发生。

读取 TCNT2、OCR2 和 TCCR2 寄存器的机制是不同的。当读 TCNT2 寄存器时，读取的是定时器的实际值。当读取 OCR2 和 TCCR2 寄存器时，读取的是临时寄存器中的值。

（2）T/C2 的异步操作

当 T/C2 采用异步工作方式时（AS2=1），计数时钟直接来自外部引脚 TOSC1，因此计数时钟与系统时钟是不同步的，所以在使用异步方式时必须注意以下几个方面：

① 当在同步和异步方式之间切换时，寄存器 TCNT2、OCR2 和 TCCR2 的内容会受到破坏。安全改变时钟源（同步异步转换）的顺序如下：

- 通过清零 OCIE2 和 TOIE2 标志位，屏蔽 T/C2 的中断；
- 选择时钟源 AS2，设置为相应的值；
- 写新的值到 TCNT2、OCR2 和 TCCR2 寄存器；
- 同步转异步时，须等待 TCN2UB、OCR2UB 和 TCR2UB 为“0”；
- 清 T/C2 的中断标志位；
- 如果需要时，使能 T/C2 的中断。

② 芯片已经对 32.768kHz 的手表用晶振进行了优化。加一个外部时钟到 TOSC1 引脚可能会导致 T/C2 不正常的工作。CPU 的主时钟频率必须大于外部时钟频率的 4 倍。

③ 当写入 TCNT2、OCR2 或 TCCR2 寄存器中的一个时，其值先写入到临时缓冲寄存器中，在 TOSC1 上的时钟两个上升沿后，临时缓冲寄存器中的值被锁存到寄存器中。在临时缓冲寄存器中的值未锁存到其目标寄存器前，不能写一个新值到临时缓冲寄存器。以上三个寄存器的每一个都有独立的临时缓冲寄存器，这意味着写 TCNT2 寄存器不会干扰一个正在进行的写 OCR2 寄存器过程。可以通过检测异步状态寄存器（ASSR），判别对指定寄存器的写入是否已经完成。

④ 在设置 TCNT2、OCR2 或 TCCR2 寄存器后，如要进入省电模式（POWER-SAVE）或等待模式（STANDBY）时，同时要将 T/C2 用作唤醒源的话，用户必须要等到被写入的寄存器完成更新后才能进入休眠状态。否则，这些寄存器的改变还没有生效，MCU 就进入休眠了。特别是使用输出比较匹配 OCF2 中断作为唤醒器件的唤醒源时，这一点尤其重要。因为在写 OCR2 或 TCNT2 期间，输出比较功能单元将被屏蔽。如果写入的内容没有生效（如 OCR2UB=1），MCU 就进入了休眠模式，器件永远都不会接收到比较匹配中断，MCU 将不会被唤醒。

⑤ 如果 T/C2 被用作将器件从省电（POWER-SAVE）或等待模式（STANDBY）唤醒，用户必须注意，当要再次进入这些休眠模式时，中断逻辑需要一个 TOSC1 周期来复位。如果器件唤醒和再次进入休眠模式之间的时间少于一个 TOSC1 周期，中断将不会发生，器件将不能被唤醒。如果不能确定再次进入省电或等待模式前的时间是否足够，可使用以下算法来保证有一个 TOSC1 的时钟周期：

- 写一个值到 TCCR2、TCNT2 或 OCR2 寄存器；
- 等待 ASSR 寄存器中相应的更新忙标志位返回“0”；
- 再次进入省电或等待模式。

⑥ 当选择了异步方式，T/C2 的 32.768kHz 晶振始终在运行状态，除非进入掉电模式（POWER-DOWN）或等待模式。在上电复位（POWER UP）或从掉电（POWER-DOWN）模式或等待模式下被唤醒时，晶振需要大约 1 秒钟的时间来稳定。建议用户在上电复位（POWER UP）或从掉电（POWER-DOWN）模式或等待模式下被唤醒后，等待 1 秒钟后再使用 T/C2。

无论 T/C2 使用内部计数时钟还是外部 TOSC1 引脚上的计数时钟，当器件从掉电或等待模式状态下被唤醒后，所有 T/C2 寄存器中的内容均被认为已经丢失（由于不稳定的时钟信号造成）。

⑦ 当使用异步方式时，从省电或等待模式下唤醒的过程为：当符合中断条件产生后，唤醒过程将在下一个计数器的时钟周期开始。器件被唤醒后，MCU 暂停 4 个时钟周期，然后执行中断处理程序，最后从 SLEEP 指令后的指令重新开始执行。

⑧ 在器件被唤醒后，立即读取 TCNT2 寄存器将返回不正确的值。由于 TCNT2 是由 TOSC1 异步时钟驱动的，而读取 TCNT2 必须通过一个寄存器，并且应同内部 I/O 时钟同步。这个读取同步发生在每个 TOSC1 的上升沿。当从省电模式唤醒，I/O 时钟（$clk_{I/O}$）再次有效，立即读取 TCNT2 则读出的为进入休眠前的值，因为直到 TOSC1 的下一个上升沿，寄存器中的值才会更新。由于在从省电模式下唤醒后，TOSC1 时钟的相位是不可预测的，

建议按照以下的顺序读取 TCNT2 寄存器：

- 写任何值到 OCR2 或 TCCR2 寄存器；
- 等待 ASSR 寄存器中相应的更新忙标志位返回为"0"，
- 读取 TCNT2 寄存器。

⑨ 在异步操作中，对于异步定时器的中断标志位的同步过程需要 3 个处理器周期加 1 个定时器周期。

2.10 同步串行接口 SPI

同步串行接口 SPI 允许在 ATmega8 和外设之间，或几个 AVR 单片机之间，以与标准 SPI 接口协议兼容的方式进行高速的同步数据传输。ATmega8 单片机的 SPI 接口的主要特征如下：

- 全双工、3 线同步数据传输；
- 可选择的主/从操作模式；
- 数据传送时，可选择 LSB 方式或 MSB 方式；
- 七种可编程的位传送速率；
- 数据传送结束的中断标志；
- 写冲突标志保护；
- 从闲置模式下被唤醒（从机模式下）；
- 倍速（CK/2）SPI 传送（主机模式下）。

图 2.54 为 ATmega8 的 SPI 接口电路方框图，而图 2.55 给出了采用 SPI 方式进行数据通信时，主-从机的连接与数据传送方式。

2.10.1 SPI 接口控制与数据传输过程

1．控制与传输过程

如图 2.55 所示，SPI 数据传输系统是由主机和从机两部分构成，主要由主、从机双方的两个移位寄存器和主机 SPI 时钟发生器组成，主机为 SPI 数据传输的控制方。由 SPI 的主机将 SS 输出线的电平拉低，作为同步数据传输的初始化信号，通知从机进入传输状态。然后主机启动时钟发生器，产生同步时钟信号 SCK；预先将在两个移位寄存器中的数据在 SCK 的驱动下进行循环移位操作，实现了主-从之间的数据交换。主机的数据由 MOSI（主机输出-从机输入）进入从机，而同时从机的数据由 MISO（主机输入-从机输出）进入主机。数据传送完成，主机将 SS 线拉高，表示传输结束。

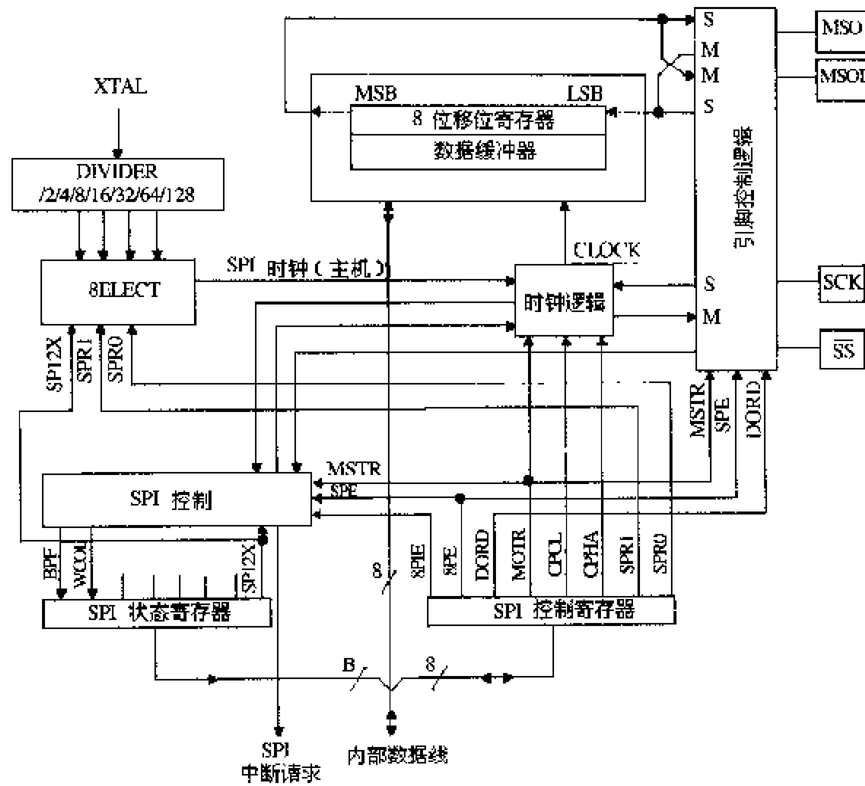


图 2.54　ATmega8 的 SPI 接口电路方框图

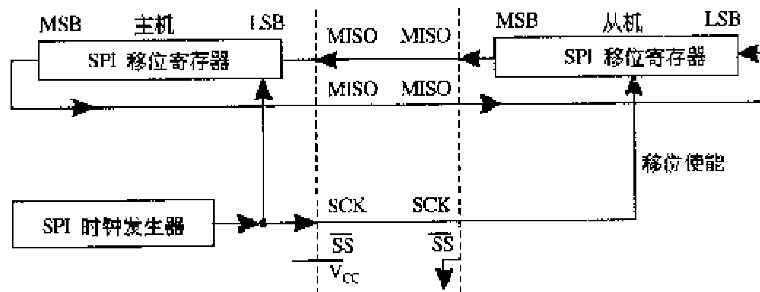


图 2.55　SPI 数据通信时的主-从机连接与数据传送方式

当 SPI 接口设置为主机方式时，其硬件接口电路是不会自动控制 SS 引脚的。因此在 SPI 通信前，应先由主机方的软件程序控制 SS，将其拉为低电平（SS 输出“0”）。此后，当把数据写入主机的 SPI 数据寄存器 SPDR 后，主机 SPI 接口将自动启动时钟发生器，在硬件电路的控制下，移位传送 8 次，通过 MOSI 将数据移出，由 MISO 移入数据。在一个字节移出后，SPI 时钟发生器停止，并置位 SPI 传送停止标志位 SPIF，如果 SPCR 寄存器中的 SPI 中断允许位 SPIE 为“1”时，则产生一个中断请求。当一个字节传送结束后，主机程序可以再次将数据写入主机的 SPI 数据寄存器，继续 SPI 的数据传输；或将 SS 信号置“1”，表示整个数据包传送完成。最后移入主机 SPDR 寄存器中的数据将被保留。

当 SPI 接口设置为从机方式时，从机的 SS 口由外部引脚驱动。当 SS 被外部拉高时，MISO 为高阻三态，SPI 接口处于休眠方式。此时，从机软件可以更新 SPI 数据寄存器 SPDR 中的数据，但不会在 SCK 的作用下做移出操作。只有在 SS 被外部拉低的条件下，SPDR

中的数据才能在 SCK 的作用下移出。在一个字节移出后，置位 SPI 传送停止标志位 SPIF；如果 SPCR 寄存器中的 SPI 中断允许位 SPIE 为“1”时，则产生一个中断请求。当一个字节传送结束后，从机程序可在读取移入的数据前，将需要继续移出的数据写入 SPI 数据寄存器。最后移入从机 SPDR 寄存器中的数据将被保留。

ATmega8 所组成的 SPI 系统，在发送方向上仅有一级缓冲，而在接收方向有两级缓冲。这意味着，在移出数据时，在前一个字节没有全部移出前，新的字符不能写入 SPI 数据寄存器中；而在接收数据时，在下一个字符被完全移入之前，前面已经收到的数据必须从 SPI 数据寄存器中读出，否则，前一个接收到的字符就会丢失。

作为从机的 SPI 接口，其内部的控制逻辑电路完成对外部 SCK 引脚信号的扫描检测。为了保证能正确地对外部 SCK 引脚信号的扫描检测，SPI 的时钟信号不能超过 fosc/4。

当 SPI 接口被使能时，MOSI、MISO、SCK 和 SS 引脚的控制与数据方向如表 2-38 所示。

表 2-38　SPI 引脚配置

引　脚	方向（主 SPI）	方向（从 SPI）
MOSI	用户定义	输入
MISO	输入	用户定义
SCK	用户定义	输入
SS	用户定义	输入

2．SPI 初始化和数据传送程序示例

下面给出两个简单的 SPI 初始化设置和数据传送的例程。在实际应用时，例程中的 DDR_SPI 等采用器件的实际引脚定义。例如：DD_MOSI 为 ATmega8 的 PB3，DDR_SPI 为 DDRB。

- 设置 SPI 为主机方式

汇编程序：

```
SPI_MasterInit:
        ;Set MOSI and SCK output, all others input
        ldi r17,(1<<DD_MOSI)|(1<<DD_SCK)
        out DDR_SPI,r17
        ;Enable SPI, Master, Set clock rate fck/16
        ldi r17, (1<<SPE)|(1<<MSTR)| (1<<SPR0)
        out SPCR,r17
        ret

SPI_MasterTransmit:
        ;Start transmission of data (r16)
        out SPDR,r16
Wait_Transmit:
```

```
        ;Wait for transmission complete
        sbis SPSR,SPIF
        rjmp Wait_Transmit
        ret
```

C 语言程序:

```
void SPI_MasterInit(void)
{
  /* Set MOSI and SCK output, all others input */
  DDR_SPI = (1<<DD_MOSI)|(1<<DD_SCK);
  /* Enable SPI, Master, Set clock rate fck/16 */
  SPCR = (1<<SPE)|(1<<MSTR)| (1<<SPR0);
}

void SPI_MasterTransmit(char cData)
{
  /* Start transmission */
  SPDR = cData;
  /* Wait for transmission complete */
  while(!(SPSR & (1<<SPIF)))
{};
}
```

● 设置 SPI 为从机方式

汇编程序:

```
SPI_SlaveInit:
        ;Set MISO output, all others input
        ldi r17,(1<<DD_MISO)
        out DDR_SPI,r17
        ;Enable SPI
        ldi r17, (1<<SPE)
        out SPCR,r17
        ret

SPI_SlaveReceive:
        ;Wait for reception complete
        sbis SPDR,SPIF
        rjmp SPI_SlaveReceive
        ;Read received data and return
```

```
        in r16,SPDR
        ret
```

C 语言程序：

```
void SPI_SlaveInit(void)
{
  /* Set MISO output, all others input */
  DDR_SPI = (1<<DD_Miso);
  /* Enable SPI */
  SPCR = (1<<SPE);
}

char SPI_SlaveReceive(void)
{
  /* Wait for reception complete */
  while(!(SPSR & (1<<SPIF)))
  {};
  /* Return data register */
  return SPDR;
}
```

3. SS 引脚的功能

● 主机方式

当 SPI 被配置为主机时（寄存器 SPCR 的 MSTR 位置“1”），用户可以决定 SS 引脚的方向。如果 SS 引脚被设为输出，该引脚将作为通用输出口，不影响 SPI 系统，通常用于驱动从机的 SS 引脚。如果 SS 被设为输入口，则必须保持高电平，用以保证主机 SPI 的操作。如果在主机模式下，SS 引脚设置为输入，而且被外部电路置低，则系统认为另外的主机选择自己为它的从机，并开始传递数据。为了防止数据总线的冲突，主机方式的 SPI 系统将采取以下动作：

①寄存器 SPCR 的 MSTR 位被清除，SPI 系统由主机变成从机，同时将 MOSI 和 SCK 引脚变成输入。

②寄存器 SPSR 中的 SPIF 位被设置，如在 SPI 中断允许的条件下，将触发引起中断。

因此在主机模式下使用中断方式来驱动 SPI 发送时，存在 SS 被置低的可能性。因此在中断程序中，应随时检查 MSTR 位是否为“1”，一旦发现 MSTR 位被清 0，则必须在程序中再次将其置位，以恢复 SPI 的主机方式。

● 从机方式

当 SPI 被配置为从机时，SS 总是作为输入口。当 SS 被外部置低时，SPI 被触发，且 MISO 变为输出口（如果被用户配置为输出口时）。在 SS 被置“1”后，所有引脚都为输入

口，同时 SPI 处于禁止状态，这意味着它不会接收输入的数据。

对于一个其至多个字节的数据包的传送，引脚 SS 的作用是非常重要的，它保证了主、从机之间与主机时钟保持同步的数据传送。一旦从机的 SS 引脚被拉高，从机的数据传送和数据接收逻辑电路马上被复位，在移位寄存器中所收到的部分不完整数据将丢失。

4．数据传送模式

在 SPI 串行同步数据时，传送的数据与不同的 SCK 相位和极性相组合构成四种 SPI 数据传送模式。图 2.56 和图 2.57 所示为 SPI 四种数据传输模式的格式。SPI 控制寄存器中的 CPHA 和 CPOL 位决定了采用哪一种数据传送模式，见表 2-39。

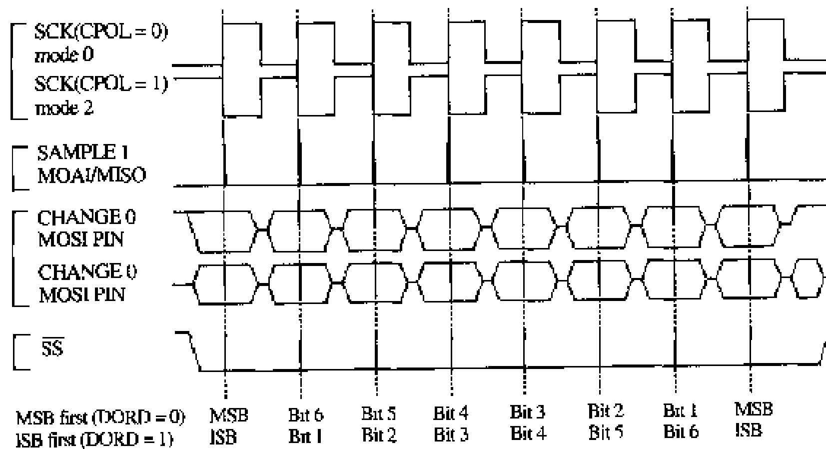


图 2.56 SPI 传送模式（CPHA=0）

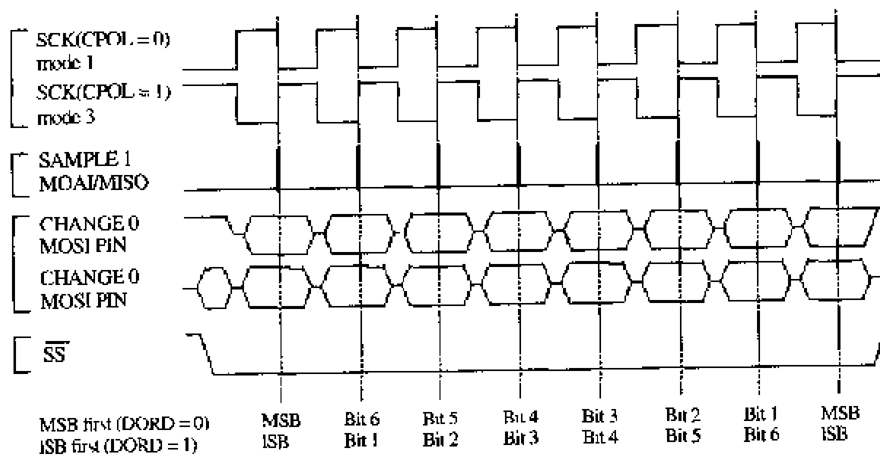


图 2.57 SPI 传送模式（CPHA=1）

表 2-39 SPI 数据传送模式

SPI 数据传送模式	CPOL	CPHA	前 沿	后 沿
0	0	0	上升沿数据锁存	下降沿数据建立
1	0	1	上升沿数据建立	下降沿数据锁存
2	1	0	下降沿数据锁存	上升沿数据建立
3	1	1	下降沿数据建立	上升沿数据锁存

2.10.2　与 SPI 接口相关的寄存器

1．SPI 控制寄存器——SPCR

位	7	6	5	4	3	2	1	0	
$0D($002D)	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	
复位值	0	0	0	0	0	0	0	0	

- 位 7——SPIE：SPI 中断允许

如果全局中断触发允许标志位 I 为"1"，且 SPIE 为"1"时，当 SPSR 寄存器的中断标志 SPIF 位为"1"，则系统响应 SPI 中断。

- 位 6——SPE：SPI 允许

当该位写入"1"时，允许 SPI 接口。在进行 SPI 的任何操作时，必须将该位置位。

- 位 5——DORD：数据移出顺序

当 DORD = 1 时，数据传送顺序为 LSB，即低位在先。

当 DORD = 0 时，数据传送顺序为 MSB，即高位在先。

- 位 4——MSTR：主/从机选择

当该位设置为"1"时，选择主机 SPI 模式；设置为"0"时，选择从机 SPI 模式。如果 SS 端口被设置为输入，且在 MSTR 为"1"时被外部拉低，则 MSTR 将被清除，SPSR 中的 SPIF 位被置"1"。此后，用户需要重新置位 MSTR，再次将 SPI 设置为主机模式。

- 位 3——CPOL：SCK 时钟极性选择

当该位被设置为"1"时，SCK 在闲置时是高电平；当 CPOL 为"0"时，SCK 在闲置时是低电平。参见表 2-38、图 2.56、图 2.57。

- 位 2——CPHA：SCK 时钟相位选择

CPHA 位的设置决定了串行数据的锁存采样是在 SCK 时钟的前沿还是 SCK 时钟的后沿。参见表 2-38、图 2.56、图 2.57。

- 位 1，0——SPR1，SPR0：SPI 时钟速率选择

这两个标志位用于控制主机模式下产生的串行时钟 SCK 的速率，SPR1 和 SPR0 对于从机模式无影响，SCK 和振荡器频率 f_{OSC} 之间的关系如表 2-40 所示。

表 2-40　SPI 时钟 SCK 选择

SPI2X（SPSR.0）	SPR1	SPR0	SCK 频率（MHz）
0	0	0	$f_{OSC}/4$
0	0	1	$f_{OSC}/16$
0	1	0	$f_{OSC}/64$
0	1	1	$f_{OSC}/128$
1	0	0	$f_{OSC}/2$

续表

SPI2X（SPSR.0）	SPR1	SPR0	SCK 频率（MHz）
1	0	1	$f_{OSC}/8$
1	1	0	$f_{OSC}/32$
1	1	1	$f_{OSC}/64$

2．SPI 的状态寄存器——SPSR

位	7	6	5	4	3	2	1	0	
$0E（$002E）	SPIF	WCOL						SPI2X	SPSR
读/写	R	R	R	R	R	R	R	R/W	
复位值	0	0	0	0	0	0	0	0	

● 位 7——SPIF：SPI 中断标志

当串行传送完成时，SPIF 位被置“1”，若 SPCR 中的 SPIE 为“1”和全局中断允许位 I 为“1”，则产生中断。如果 SS 被设置为输入，且在 SPI 为主机模式时被外部拉低，也会置位 SPIF 标志。SPIF 标志位的属性为只读，清零 SPIF 有两种方式：

（1）硬件方式。MCU 响应 SPI 中断，转入 SPI 中断向量的同时，SPIF 位由硬件自动清除。

（2）软件方式。先读取 SPI 状态寄存器 SPSR（读 SPSR 的操作将会自动清除 SPIF 位），然后再进行对 SPI 数据寄存器 SPDR 的操作处理。

● 位 6——WCOL：写冲突标志

如果在 SPI 接口的数据传送过程中向 SPI 的数据寄存器 SPDR 写入数据，则会置位 WCOL。清零 WCOL 标志只能通过以下软件方式：先读取 SPI 状态寄存器 SPSR（读 SPSR 的操作将会自动清除 SPIF 位和 WCOL 位），然后再进行对 SPI 数据寄存器 SPDR 的操作处理。

● 位 5..1：保留位

在 ATmega8 中，这几位保留，读出为“0”。

● 位 0——SPI2X：倍速 SPI

在主机 SPI 模式下，当该位被写为逻辑“1”时，SPI 的速度（SCK 的频率）将加倍（参见表 2-39），这意味着产生最小的 SCK 周期是 MCU 时钟周期的两倍。当 SPI 被设置为从机模式时，SCK 必须低于 1/4 的系统时钟频率，才能确保有效的数据传送。

ATmega8 单片机的 SPI 接口也被用于程序存储器和数据 E^2PROM 的编程下载和上传，详见程序存储器的串行编程和校验部分。

3．SPI 数据寄存器——SPDR

位	7	6	5	4	3	2	1	0	
$0F（$002F）	MSB							LSB	SPDR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	x	x	x	x	x	x	x	x	

SPI 数据寄存器为可读/写的寄存器，用于在寄存器组和 SPI 移位寄存器之间传送数据。写数据到该寄存器时，将传送初始化数据；读该寄存器时，读到的是移位寄存器接收缓冲区中的值。

2.11　通用同/异步串行接口 USART

ATmega8 单片机带有一个全双工通用同步/异步串行收发模块 USART，该接口是一个高度灵活的串行通信设备。其主要特点如下：

- 全双工操作（相互独立的接收数据寄存器和发送数据寄存器）；
- 支持同步或异步操作；
- 同步操作时，可主机时钟同步，也可从机时钟同步；
- 独立的高精度波特率发生器，不占用定时/计数器；
- 支持 5、6、7、8 和 9 位数据位，1 位或 2 位停止位的串行数据帧结构；
- 由硬件支持的奇偶校验位发生和校验；
- 数据溢出检测；
- 帧错误检测；
- 包括错误起始位的检测的噪声滤波器和数字低通滤波器；
- 三个完全独立的中断，TX 发送完成、TX 发送数据寄存器空、RX 接收完成；
- 支持多机通信模式；
- 支持倍速异步通信模式。

2.11.1　概述

图 2.58 为 ATmega8 的全双工通用同步/异步串行收发模块 USART 收发器的接口硬件结构方框图。

图中的虚线框将 USART 收发模块分为三大部分：时钟发生器、数据发送器和接收器。控制寄存器为所有的模块共享。时钟发生器由同步逻辑电路（在同步从模式下由外部时钟输入驱动）和波特率发生器组成。发送时钟引脚 XCK 仅用于同步发送模式下，发送器部分由一个单独的写入缓冲器（发送 UDR）、一个串行移位寄存器、校验位发生器和用于处理不同帧结构的控制逻辑电路构成。使用写入缓冲器，实现了连续发送多帧数据无延时的通信。接收器是 USART 模块中最复杂的部分，最主要的是时钟和数据接收单元。数据接收单元用作异步数据的接收。除了接收单元，接收器还包括校验位校验器、控制逻辑、移位寄存器和两级接收缓冲器（接收 UDR）。接收器支持与发送器相同的帧结构，同时支持帧错误、数据溢出和校验错误的检测。

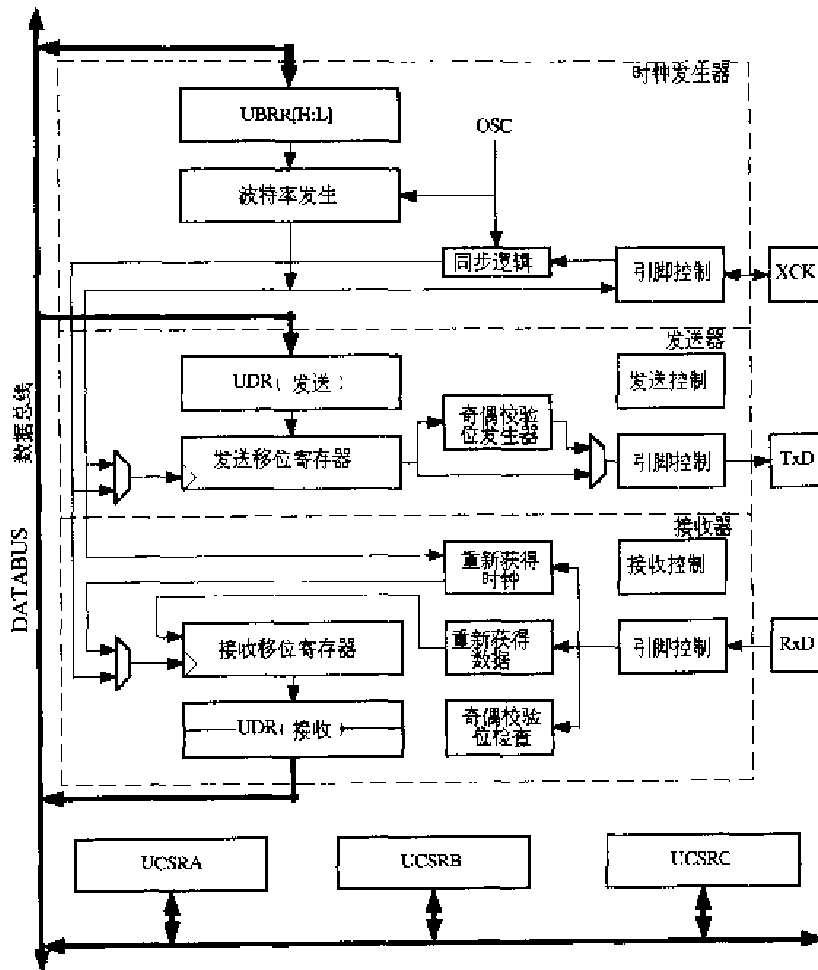


图 2.58 ATmega8 全双工通用同步/异步串行收发模块 USART 收发器接口硬件结构图

1. AVR 的 USART 接口与 UART 接口之间的兼容性

AVR 系列的单片机在片内集成的通用串行通信接口有两种形式，一种为普通的支持异步通信的 UART 接口，另一种为加强型的支持同步/异步通信的 USART 接口。ATmega8 配备的是加强型的支持同步/异步通信的 USART 接口。

AVR 单片机的 USART 在以下几个方面与 UART 完全兼容：

- USART 的寄存器内部包含 UART 的控制位，位置相同；
- 波特率发生；
- 发送器操作；
- 发送缓冲器功能；
- 接收器操作。

但是，由于 USART 接收缓冲器的两个改进，在某些特殊的情况下会影响兼容性：

- USART 增加了一个接收缓冲寄存器，两个缓冲寄存器如同一个循环的 FIFO 缓冲器一样工作。因此对于每一个接收的数据，只能读一次 UDR 寄存器。更重要的是，错误标志位（FE 和 DOR）以及第 9 位数据（RXB8）也保留在接收缓冲器中，因此必须要先读取这些标志位，然后再读取 UDR 寄存器，否则，错误标志会随着缓

冲区的读出而丢失。

- USART 的接收移位寄存器可看作为第三个接收缓冲器。如果接收缓冲寄存器已满，新接收的数据可以保留在串行移位寄存器中，一直到下一个新的开始位被检测到。这样，更加提高了防止数据接收溢出的性能。

以下 URAT 的 2 个控制位在 USART 中改变了名字，但是它们的功能和在寄存器中的位置是相同的：

- CHR9 位变为 UCSZ2 位。
- OR 位变为 DOR 位。

2.11.2　串行时钟发生

时钟发生逻辑为发送器和接收器提供基本的时钟。USART 支持 4 种时钟工作模式：普通异步模式、两倍速异步模式、主机同步模式和从机同步模式。USART 控制和状态寄存器 C（UCSRC）中的 UMSEL 位用于选择同步或异步模式。双倍速模式（只有异步模式有效）由 UCSRA 寄存器中的 U2X 位控制。当使用同步模式时，XCK 引脚的数据方向寄存器（DDR_xck）控制了时钟源是来自内部的（主机模式）还是由外部驱动（从机模式）。XCK 引脚只在使用同步模式下有效。图 2.59 为时钟发生逻辑的方框图。

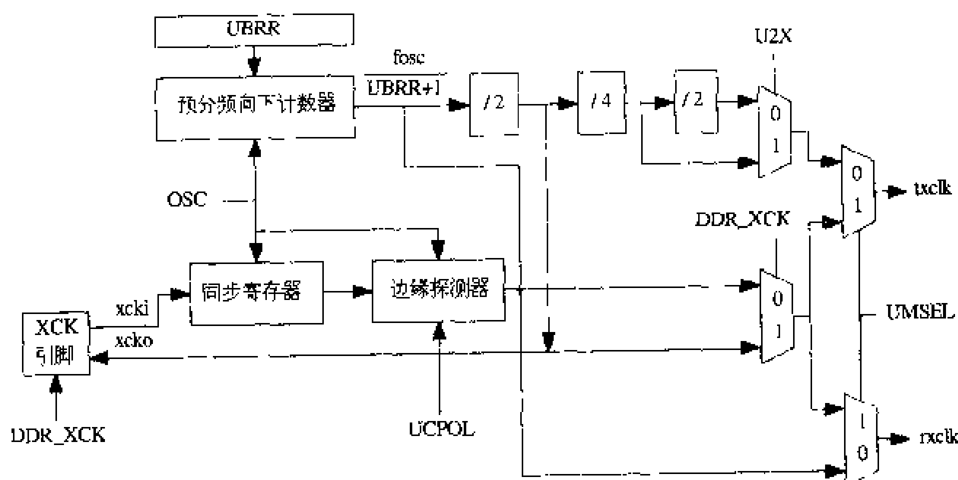


图 2.59　USART 时钟发生逻辑方框图

图中信号的意义为：

txclk：发送器时钟（内部信号）；

rxclk：接收器基本时钟（内部信号）；

xcki：从 XCK 引脚输入的时钟（内部信号，用于同步通信从机模式）。

xcko：输出到 XCK 引脚的时钟（内部信号，用于同步通信主机模式）。

fosc：XTAL 引脚的时钟频率（系统时钟）。

1．内部时钟发生——波特率发生器

内部时钟发生被用于异步模式和同步主机模式，参见图 2.59。USART 的波特率寄存器 UBRR 和预分频向下计数器（DOWN-COUNTER）相连接，一起构成可编程的预分频器或波特率发生器。向下计数器对系统时钟计数，当其计数到零或 UBRRL 寄存器被写时，会自动装入 UBRR 寄存器的数值。当计数到零时产生一个时钟，该时钟作为波特率发生器的输出时钟，输出时钟的频率为 fosc/（UBRR+1）。发送器对波特率发生器的输出时钟进行 2、8 或 16 的分频，具体情况取决于工作模式。波特率发生器的输出被直接用作接收器和数据接收单元的时钟。然而，接收单元使用了一个有 2、8 或 16 个状态的状态机，具体状态数由 UMSEL、U2X 和 DDR_xck 位设定的工作模式决定。表 2-41 给出了计算波特率和计算每一种使用内部时钟源工作的模式的 UBRR 值的公式。

表 2-41　波特率计算公式

使用模式	波特率的计算	UBRR 值的计算
异步正常模式 U2X=0	$BAUD=\frac{fosc}{16(UBRR+1)}$	$BARR=\frac{fosc}{16\times BAUD}-1$
异步倍速模式 U2X=1	$BAUD=\frac{fosc}{8(UBRR+1)}$	$BARR=\frac{fosc}{8\times BAUD}-1$
同步主机模式	$BAUD=\frac{fosc}{2(UBRR+1)}$	$BARR=\frac{fosc}{2\times BAUD}-1$

2．双倍速工作模式（U2X）

通过设定 UCSRA 寄存器中的 U2X 位可以使传输速率加倍。设置该位只对异步工作模式有效。当工作于同步模式时，设置该位为“0”。

设置该位把波特率分频器的分频值从 16 降到 8，使异步通信的传输速率加倍。注意在这种情况下，接收器只使用一半的采样数对数据进行采样和时钟校正，因此在该种模式下更精确的系统时钟和更精确的波特率设置是必需的。

3．外部时钟

在同步从机模式操作中是由外部时钟驱动的，如图 2.59 所示。输入到 XCK 引脚的外部时钟由同步寄存器进行采样，用以减小亚稳定（META-STABILITY）的可能性。同步寄存器的输出通过一个边沿检测器，然后应用于传送器和接收器。这一过程引入了两个 CPU 时钟周期的延时，因此外部 XCK 的最大时钟频率由以下公式限制：

$$fxck<\frac{fosc}{4}$$

fosc 由系统时钟的稳定性决定，为了防止因频率的漂移而丢失数据，建议留有一定余地。

4．同步时钟操作

当使用同步模式时（UMSEL=1），XCK 引脚被用作时钟的输出（主机模式）或时钟的输入（从模式）。时钟的边沿、数据的采样和数据的变化之间的关系的基本规律是：在改变

数据输出端 TxD 的 XCK 时钟的相反边沿对数据输入端 RxD 进行采样。例如，在 XCK 的下降沿时改变 TxD 的数据，上升沿对 RxD 数据采样（UCPOL=1）。

UCRSC 寄存器中的 UCPOL 位选择使用 XCK 时钟的哪个边沿对数据采样和改变输出数据。如图 2.60 所示，当 UCPOL=0 时，在 XCK 的上升沿改变输出数据，在 XCK 的下降沿进行数据采样。当 UCPOL=1 时，在 XCK 的下降沿改变输出数据，在 XCK 的上升沿进行数据采样。

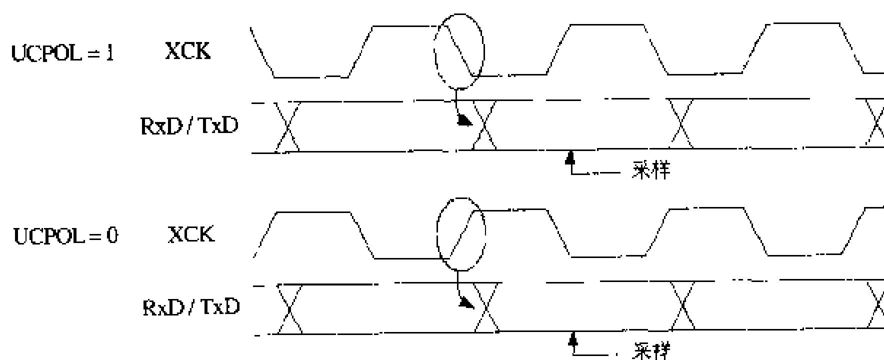


图 2.60 同步模式时的 XCK 时序

2.11.3 数据帧格式

一个串行数据帧是由一个数据位字加上同步位（开始和结束位）以及作为检错的检验位三部分构成。ATmega8 的 USART 可以使用以下几种有效组合的数据帧格式：

- 1 个起始位（start bit）；
- 5、6、7、8 或 9 位数据位（data bits）；
- 一个无、奇校验或偶校验位（no、even or add parity bit）；
- 1 或 2 个停止位（stop bits）。

一个数据帧是以起始位开始；紧接着是数据字的最低位，数据字最多可以有 9 个数据位，数据位以数据的最高位结束。如果使能了校验位，校验位将接着数据位，最后是结束位。当一个完整的数据帧传输后，可以直接跟着传送下一个新的数据帧，或者使通信线路处于空闲状态。图 2.61 所示是可能的数据帧结构组合。

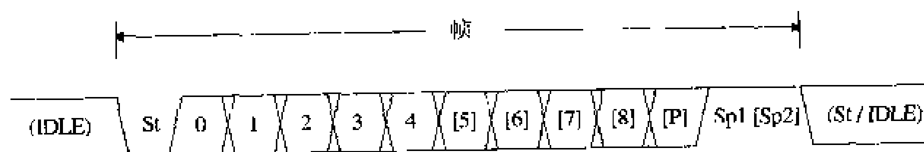


图 2.61 USART 的数据帧结构组合

图中符号的意义为：

St: 启始位，低电平

n: 数据位（0~8），低位在前

P: 校验位

Sp: 停止位，高电平

IDLE: 线路空闲，线路空闲时线路保持高电平

数据帧的结构由 UCSRB 和 UCSRC 寄存器中的 UCSZ2:0、UPM1:0 和 USBS 位设置和定义，接收和发送使用同样的定义设置。注意：任何这些设置的改变，都会打断正在进行的数据传送和接收通信。

USART 帧的字长位（UCSZ2..0）确定了数据帧的数据位数。USART 的校验模式位（UPM1..0）用于使能和决定校验的类型。选择一位或两位结束位由 USART 的 USBS 位设置。但接收器是忽略第二个停止位的，因此帧错误（FE）只会在第一个结束位为“0”时被检测到。

校验位的计算是对数据位的各个位进行异或运算，其结果再同“0”或“1”进行异或运算：

$$P_{even} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$

$$P_{odd} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

式中： Peven 偶校验位值

Podd 奇校验位值

dn 数据的第 n 位

如在数据传输格式中定义使用校验位，则校验位值将出现在最后一个数据位和第一个停止位之间。

2.11.4 USART 的初始化

USART 接口在通信前，必须首先对其进行初始化。初始化过程通常包括波特率的设定，帧结构的设定和根据需要的接收器或发送器的使能。对于中断驱动的 USART 操作，在初始化时，全局中断标志位应该先被清零（全局中断被屏蔽），然后再进行 USART 的初始化（如改变波特率或帧结构）。重新改变 USART 的设置应该在没有数据传输的情况下进行。TXC 标志位可以用来检验一个数据帧的发送是否已经完成，RXC 标志位可以用来检验是否在接收缓冲中还有数据未读出。在每次发送前（在写发送数据寄存器 UDR 前），TXC 标志位必须被清零。

以下是 USART 初始化程序示例。

● 汇编程序：

```
USART_Init:
; Set baud rate
out UBRRH, r17
out UBRRL, r16
; Enable receiver and transmitter
```

```
ldi r16, (1<<RXEN)|(1<<TXEN)
out UCSRB,r16
; Set frame format: 8data, 2stop bit
ldi r16, (1<<URSEL)|(1<<USBS)|(3<<UCSZ0)
out UCSRC,r16
ret
```

● C 程序代码：

```
void USART_Init( unsigned int baud )
{
/* Set baud rate */
UBRRH = (unsigned char)(baud>>8);
UBRRL = (unsigned char)baud;
/* Enable receiver and transmitter */
UCSRB = (1<<RXEN)|(1<<TXEN);
/* Set frame format: 8data, 2stop bit */
UCSRC = (1<<URSEL)|(1<<USBS)|(3<<UCSZ0);
}
```

2.11.5　数据发送

USART 数据发送是由 UCSRB 寄存器中的发送允许位 TXEN 设置。当被 TXEN 使能时，TxD 引脚的通用 I/O 性能将被 USART 代替，作为发送器的串行输出引脚。传送的波特率、工作模式和帧结构必须先于发送设置完成。如果使用同步发送模式，内部产生的发送时钟信号施加在 XCK 引脚上，作为串行数据发送的时钟。

1．发送 5 到 8 位数据位的帧

数据传送是通过把将要传送的数据放到发送缓冲中来初始化的。CPU 通过写入到 UDR 发送数据寄存器来加载发送缓冲器。当移位寄存器为发送下一帧准备就绪时，缓冲的数据将被移到移位寄存器中。如果移位寄存器处于空闲状态或刚结束前一帧的最后一个停止位的传送，它将装载新的数据。一旦移位寄存器中装载了新的数据，就会按照设定的数据帧模式和速率完成一帧数据的发送。

以下程序段给出一个采用轮询（polling）方式发送数据的例子。寄存器 R16 中为要发送的数据，程序循环检测数据寄存器空标志位 UDRE，一旦该标志位置位，将数据写入发送数据寄存器 UDR 后由硬件自动将其发送。如果发送的数据少于 8 位，则高位的数据将不会被移出发送而放弃。

● 汇编程序代码：

```
USART_Transmit:
; Wait for empty transmit buffer
sbis UCSRA,UDRE
rjmp USART_Transmit
; Put data (r16) into buffer, sends the data
out UDR,r16
ret
```

- C 程序代码：

```
void USART_Transmit( unsigned char data )
{
/* Wait for empty transmit buffer */
while ( !( UCSRA & (1<<UDRE)) )
;
/* Put data into buffer, sends the data */
UDR = data;
}
```

2．发送 9 位数据位的帧

如果设置为发送9位数据的数据帧(UCSZ=7)，应先将数据的第9位写入寄存器UCSRB的 TXB8 标志位中，然后再将低 8 位数据写入发送数据寄存器 UDR 中。第 9 位数据在多机通信中用于表示地址（1）或数据（0）帧，或在同步通信中作为握手协议使用。

以下程序段给出一个采用轮询(polling)方式发送 9 位数据的数据帧例子。寄存器 R17：R16 中为要发送的数据（R17 的第 0 位为发送数据的第 9 位），程序循环检测数据寄存器空标志位 UDRE，一旦该标志位置位，将数据写入数据寄存器 UDR 后由硬件自动将其发送。

- 汇编程序代码：

```
USART_Transmit:
; Wait for empty transmit buffer
sbis UCSRA,UDRE
rjmp USART_Transmit
; Copy 9th bit from r17 to TXB8
cbi UCSRB,TXB8
sbrc r17,0
sbi UCSRB,TXB8
; Put LSB data (r16) into buffer, sends the data
out UDR,r16
ret
```

● C 程序代码：

```
void USART_Transmit( unsigned int data )
{
/* Wait for empty transmit buffer */
while ( !( UCSRA & (1<<UDRE))) )
;
/* Copy 9th bit to TXB8 */
UCSRB &= ~(1<<TXB8);
if ( data & 0x0100 )
UCSRB |= (1<<TXB8);
/* Put data into buffer, sends the data */
UDR = data;
}
```

3．传送标志位和中断

USART 的发送器有两个标志位：USART 数据寄存器空 UDRE 标志和传送完成 TXC 标志，两个标志位都能发生中断。

数据寄存器空 UDRE 标志位表示发送缓冲器是否就绪，可以接受一个新的数据。该位在发送缓冲器空时被置“1”；当发送缓冲区内含有正在发送的数据时，该位为“0”。为了和其他的器件兼容，建议在写 UCSRA 寄存器时，该位写为“0”。

当 UCSRB 寄存器中的数据寄存器空中断允许位 UDRIE 为“1”时，只要 UDRE 被置位，就将产生 USART 数据寄存器空中断申请。UDRE 位在发送寄存器 UDR 的写入后被自动清零。当采用中断方式的数据传送时，在数据寄存器空中断服务程序中必须写一个新的数据到 UDR 中，以清零 UDRE；或者屏蔽掉数据寄存器空中断标志。否则，一旦该中断程序结束后，一个新的中断将再次产生。

在整个数据帧移出发送移位寄存器，同时发送缓冲器中又没有新的数据时，发送完成标志位 TXC 将置位。TXC 标志位对于采用如 RS485 标准的半双工通信接口十分有用。在这种情况下，传送完毕后，应用程序必须释放通信总线，进入接受状态。

当发送完成中断允许位 TXCIE 和全局中断允许位均被置为“1”时，随着 TXC 标志位的置位，USART 发送完成中断将被执行。一旦进入执行发送完成中断服务程序时，TXC 标志位即会被硬件自动清零，因此在中断处理程序中不必清零 TXC 标志位。向 TXC 标志位写入一个“1”，也能清零该标志位。

4．校验位

在数据发送中，校验位发生电路会根据发送的数据和设定的校验方式自动计算和产生相应的校验位。当需要发送校验位时（UPM1=1），发送逻辑控制电路会在发送数据的最后一位和第一个停止位之间插入校验位。

5．禁止发送

设置标志位 TXEN 为零将禁止数据发送。将 TXEN 置为零后，要等正在进行的发送完成后设置才生效。当发送被禁止后，USART 发送器将不再占用 TxD 引脚。

2.11.6 数据接收

USART 数据接收允许是由 UCSRB 寄存器中的接收允许位 RXEN 设置。当 RXEN 使能时，RxD 引脚的通用 I/O 性能被 USART 代替，作为接收器的串行输入引脚。传送的波特率、工作模式和帧结构必须先于允许接收的设置完成。如果使用同步接收模式，外部施加在 XCK 引脚上的时钟作为串行数据接收时钟。

1．接收 5 到 8 位数据的帧

当接收硬件单元电路检测到有效的数据帧的起始位时，它将开始接收数据。在起始位后的每一位都以波特率或 XCK 的时钟进行采样，并移入接收移位寄存器中，直到第一个停止位。第二个停止位将被接收电路忽略。当第一个停止位被接收时，移位寄存器中的内容将被移到接收缓冲器中，接收缓冲器能通过接收寄存器 UDR 进行读取。

以下程序段给出一个采用轮询（polling）方式接收数据的例子。程序循环检测接收完成标志位 RXC，一旦该标志位置位，即从数据寄存器 UDR 中读出接收的数据。如果接收数据帧的格式少于 8 位，则 UDR 相应的高位为“0”。

● 汇编程序代码：

```
USART_Receive:
; Wait for data to be received
sbis UCSRA, RXC
rjmp USART_Receive
; Get and return received data from buffer
in r16, UDR
ret
```

● C 程序代码：

```
unsigned char USART_Receive( void )
{
/* Wait for data to be received */
while ( !(UCSRA & (1<<RXC)) )
;
/* Get and return received data from buffer */
return UDR;
}
```

2．接收 9 个数据位的帧

如果接收的是 9 位数据的数据帧（UCSZ=7），那么必须先从寄存器 UCSRB 的 RXB8 位中读取第 9 位数据，然后从 UDR 中读取数据的低 8 位。这一规则同样适用于读取 FE、DOR 和 PE 等状态标志位寄存器。也就是说，应先读取状态寄存器 UCSRA，再读取 UDR。因为，读取 UDR 寄存器会改变状态寄存器中各个标志位的值。这样最大程度地优化了接收缓冲器的性能，一旦 UDR 中的数据被读出，缓冲器将自动开始下一个数据的接收。

以下程序段给出一个采用轮询（polling）方式接收 9 位数据帧的例子。程序循环检测接收完成标志位 RXC，一旦该标志位置位，先读出所有的状态标志位，最后将数据从数据寄存器 UDR 中读出。

- 汇编程序代码：

```
USART_Receive:
; Wait for data to be received
sbis UCSRA, RXC
rjmp USART_Receive
; Get status and 9th bit, then data from buffer
in r18, UCSRA
in r17, UCSRB
in r16, UDR
; If error, return -1
andi r18,(1<<FE)|(1<<DOR)|(1<<PE)
breq USART_ReceiveNoError
ldi r17, HIGH(-1)
ldi r16, LOW(-1)
USART_ReceiveNoError:
; Filter the 9th bit, then return
lsr r17
andi r17, 0x01
ret
```

- C 程序代码：

```
unsigned int USART_Receive( void )
{
unsigned char status, resh, resl;
/* Wait for data to be received */
while ( !(UCSRA & (1<<RXC)) )
;
/* Get status and 9th bit, then data from buffer */
status = UCSRA;
resh = UCSRB;
```

```
resl = UDR;
/* If error, return -1 */
if ( status & (1<<FE)|(1<<DOR)|(1<<PE) )
return -1;
/* Filter the 9th bit, then return */
resh = (resh >> 1) & 0x01;
return ((resh << 8) | resl);
}
```

3. 接收完成标志和中断

USART 的接收器有一个状态标志位 RXC。接收器接收一个完整的数据帧后，接收到的数据驻留在接收缓冲器中，此时 RCX 标志位会置“1”，表示接收器收到一个数据在接收缓冲器中，未被读走。RCX 为零时，表示数据接收器为空。当设置接收器禁止接收时（RXEN=0），接收缓冲器中的数据将被清除，RCX 标志位自动清零。

当 UCSRB 寄存器中的接收完成中断允许位 RXCIE 为“1”时，只要 RXC 被置位，将产生数据接收完成中断申请。RXC 位在寄存器 UDR 数据被读出后被自动清零。当使用中断方式数据接收时，在接收完成中断服务程序中必须读取数据寄存器 UDR，以清零 RXC；或者屏蔽掉数据接收完成中断标志。否则，一旦该中断程序结束后，一个新的中断将再次产生。

4. 接收错误标志

USART 的接收器有 3 个状态标志位：接收帧错 FE、接收数据溢出 DOR 和校验错 PE。它们指出了当前接收数据的错误状态，但不会产生中断申请。通过读取 UCSRA 寄存器可以获得这些标志位的内容，但这些错误标志位是不能采用软件设置来改变它们的内容的。由于读取 UDR 寄存器会改变这些标志位的值，所以应在读取 UDR 之前读取寄存器 UCSRA 获取错误标志。在重新改变 USART 的设置时，这些标志位应写入“0”。

标志位 FE 表示刚接收到的数据帧中第一停止位是否正确。FE=0 表示正确收到该数据帧的停止位（停止位为“1”），FE=1 表示收到的停止位有误（停止位为“0”）。该标志可用于检测数据与时钟是否同步，数据传送是否被打断以及握手协议等。无论数据帧采用一位还是 2 位停止位，FE 标志仅对第一停止位进行检测，因此不受 USBS 位设置的影响。

标志位 DOR 表示接收到的数据是否产生溢出丢失的情况。当接收缓冲器满（即有两个接收到的字符，前一个字符在 UDR 中，移位寄存器中还有一个新接收到的字符在等待），同时接收器又检测到一个新的起始位时，接收数据溢出出错发生，该位置“1”。DOR=1 表示在最后一次读取 UDR 中接收的数据后，发生了一个或多个接收数据的溢出丢失。一旦接收缓冲器 UDR 被读取，DOR 被自动清零。重写寄存器 UCSRA 的操作总是设置 DOR 标志位为“0”。

标志位 PE 表示刚接收到数据的检验是否正确。当设置为无校验时（UPM1..2=00），PE 总是为“0”。重写寄存器 UCSRA 的操作总是设置 DOR 标志位为“0”。

5. 校验

标志位 UPM1=1 表示允许校验，标志位 UPM2 确定为偶校验还是奇校验。当使能校验功能，硬件在接收一帧数据的同时，计算其校验位的值，并同接收到的数据帧中的校验位进行比较。如果不相同，将置位 PE，表示接收的数据发生校验错误。用户程序可以读取 PE 标志位，判别接收数据是否有校验错误。

6. 禁止接收功能

与禁止发送功能不同，一旦设置禁止接收（RXEN=0），接收器将立即停止接收数据，因此正在接收过程中的数据将会丢失。接收功能禁止后，接收器将不再占用 RxD 引脚，接收缓冲器将随着接收功能的禁止而被清空，其中的数据将会丢失。一般情况下，应先检测标志 RXC，待 RXC 置位后，将 UDR 中最后的数据读出，然后禁止接收功能。

2.11.7 异步串行数据的硬件扫描检测和接收时序

USART 接收器的硬件电路包括时钟定位复原和数据定位复原单元，用于处理异步数据的扫描检测和接收。时钟定位复原用于调整和定位内部产生的波特率时钟，使其同 RxD 引脚的输入数据同步。数据定位复原用于扫描检测输入数据的逻辑值。

图 2.62 所示为检测输入数据帧的起始位的时钟定时和同步过程。在正常情况下，接收器的时钟定位复原电路以 16 倍波特率的采样频率扫描采样 RxD 引脚信号（倍速模式以 8 倍波特率的采样频率扫描采样 RxD 引脚信号），图中每个垂直箭头线表示一次采样。当 RxD 处于闲置状态时（无数据传送），采样点均定义为 0（见图 2.62）。

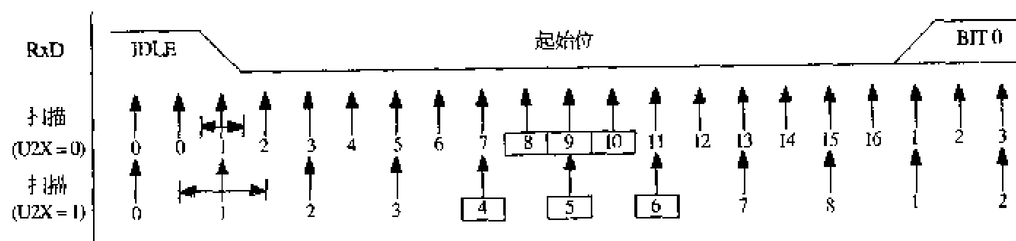


图 2.62 起始位同步检测

当时钟定位复原电路在 RxD 上检测到一个由高到低的下降沿时，便开始一个数据帧起始位的探测序列，并初始化（同步）第一个检测到低电平的采样点为序列 1。在连续的 16 个对起始位的采样点中（倍速方式为 8 个），取第 8、9、10（倍速方式为 4、5、6）三个点的采样值作为起始位的判定。如果 3 个采样值有 2 个或 3 个为高电平，则认为检测到的是一个尖峰噪声信号，而不是起始信号，时钟定位复原电路将继续探测下一个 1 到 0 的电平转换。如果在 3 个采样值中有 2 个或 3 个为低电平，则判定检测到一个起始位，时钟的波特率与数据实现同步，进入扫描检测数据位。

在一个有效的起始位被检测到后，数据定位复原电路开始对数据位进行扫描采样。图 2.63 为扫描检测输入数据位的探测序列图。

数据位的扫描检测序列也是 16 倍（8 倍）波特率，对应一个数据位有 16（8）个采样

点。同样，在连续 16 个采样点中（倍速方式为 8 个），取第 8、9、10（倍速方式为 4、5、6）三个点的采样值作为数据位的判定。如果 3 个采样值有 2 个或 3 个为高电平，则认为检测到的数据位为“1”；而如果 3 个采样值有 2 个或 3 个为低电平，则认为检测到的数据位为“0”。数据位的逻辑值判定以后被送入移位寄存器中。这个数据位的扫描检测过程一直重复到所有数据位检测完成，并延续到第一个停止位的扫描检测。

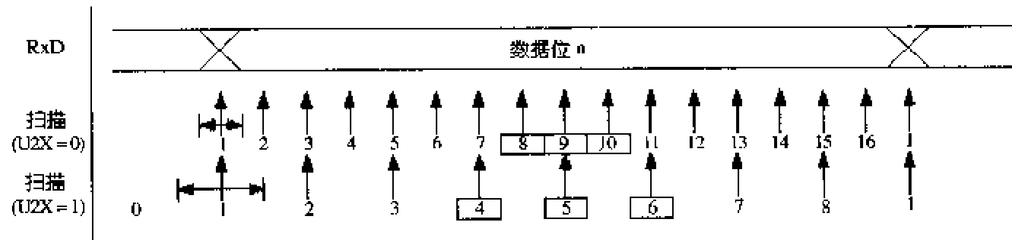


图 2.63　数据位同步检测

对于停止位的扫描检测的探测序列同数据位相同，但如果检测判定停止位为“0”，则帧错误标志位 FE 将置位，表示接收帧出错。与数据位扫描检测不同的是，在检测停止位的第 11 个（7 个）采样点后，便进入了下一个起始位（1 到 0 的转换）的检测，而不是在 16 个（8 个）采样点后再进入起始位的检测（见图 2.64）。采用这种停止位的探测方式，可以在一定范围内调整接收数据的波特率与内部产生的波特率的匹配误差。图中的 A 点（倍速方式为 B 点）表示最早的下一个起始位允许到来的时刻。对于采用正常方式传送数据，如果数据帧的格式为一个起始位，8 个数据位、无校验位和一个停止位，则传送数据波特率与接收器本身内部产生的波特率的最大误差调整率为+4.58%/−4.54%（快/慢），典型误差调整率为±2.0%。也就是说，数据与接收器内部的波特率快慢相差为 4%时，接收器还是可以正确地检测和接收数据的。

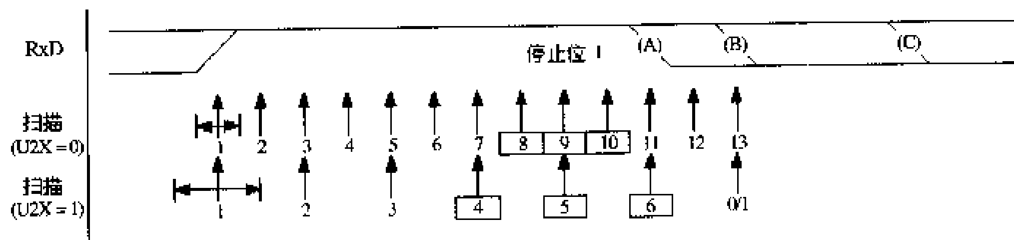


图 2.64　停止位同步检测和波特率调整

2.11.8　多机通信模式

设置 UCSRA 寄存器中的标志位 MPCM，可以使能 USART 接收器对接收的数据帧进行过滤的功能。如果使能了过滤功能，接收器对接收到的那些不是地址信息帧的数据帧将进行过滤，不将其放入接收缓冲器中，这在多机通信中有效地减少了 MCU 处理数据帧的数目。发送器不受 MPCM 位设置的影响，但如果工作在多机通信方式下时，其工作处理方式应与一般的异步通信方式有所不同。

在多机通信中，如果接收器被设置为接收 5~8 个数据位的数据帧，第一个停止位则表征该帧包含的是数据信息还是地址信息。如果接收器被设置为接收 9 个数据位的帧结构，则第 9 位（RXB8）被用作表征该帧是地址帧还是数据帧。当帧类型表征位为“1”时，表示该帧数据为一个地址帧；当帧类型表征位为“0”时，表示这个帧为一个数据帧。

多机通信模式允许多个从机并在通信线路上接收一个主机发出的数据。通过对接收到的地址帧中的地址进行解码，确定哪个从机被主机寻址。如果某个从机被主机寻址，它将接收接下来主机发出的数据帧，而其他的从机将忽略数据帧，直到再次接收到一个地址帧。

对于在多机通信系统中的主机 MCU，可以设置使用 9 位数据帧结构（UCSZ=7）。当发送地址帧时，置第 9 位为“1”；发送数据帧时，置第 9 位为“0”。在这种情况下，从机也必须设置成接收 9 位数据帧结构。

多机通信方式的数据交换过程如下：

1．设置所有从机工作在多机通信模式（MPCM=1）。

2．主机发送一个地址帧，所有从机接收和读取该地址帧。所有从机的 MCU 中，RXC 标志位被置位，表示接收到地址帧。

3．每一个从机 MCU 读取 UDR 寄存器，并判断自己是否被主机寻址。如果被寻址，清 UCSAR 寄存器中的 MPCM 位，等待接收数据；否则保持 MPCM 为“1”，等待下一个地址帧的接收（该步应由用户软件处理实现）。

4．被寻址的从机 MCU 将接收之后所有主机下发的数据帧，直到下一个地址帧为止。其他从机 MCU 忽略这些数据帧。

5．当被寻址的从机 MCU 接收完最后一个数据帧后将 MPCM 位置位，等待下一个地址帧的出现。然后从步骤 2 开始重复。

虽然在多机通信中可以使用 5 或 8 位数据位的帧结构，但由于接收方必须在 N 和 N+1 的帧结构之间来回切换，所以不实用，同时也使得实现双工通信的操作变得更加复杂化。如必须使用 5~8 位的数据帧结构，则必须采用两个停止位，因为第一个停止位被用于判别帧的类型。

2.11.9 USART 寄存器

1．USART 数据寄存器——UDR

位	7	6	5	4	3	2	1	0	
$0C($002C)	MSB	RXB[7..0]						LSB	UDR（Read）
$0C($002C)	MSB	TXB[7..0]						LSB	UDR（Write）
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

UDR 寄存器是两个物理上分离的寄存器，它们分享相同的 I/O 地址。当写入寄存器时，USART 的发送数据寄存器 TXB 被写入；当读 UDR 时，读取的是 USART 的接收寄存器 RXB。对于 5、6 或 7 位的数字帧，高位未用到的位在发送时被忽略，在接收时由硬件自动置为“0”。

只有在 USCRA 寄存器中的 UDRE 位被置为“1”时，UDR 才能被写入，否则写入的数据将被 USART 忽略。在发送使能情况下，写入 UDR 的数据将进入发送器的移位寄存器，由引脚 TxD 串行移出。

接收数据寄存器是一个两级的 FIFO 结构，只要该接收缓冲 RXB 被访问，就会改变 FIFO 的状态。由于这种特性，访问接收缓冲 RXB 不要使用 SBI 和 CBI 之类的读-改-写指令，并且应小心使用 SBIC 和 SBIS 之类的位检测指令，因为有可能改变 FIFO 的状态。

2. USART 控制和状态寄存器 A——UCSRA

位	7	6	5	4	3	2	1	0	
$0B($002B)	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
读/写	R	R/W	R	R	R	R	R/W	R/W	
复位值	0	0	1	0	0	0	0	0	

● 位 7——RXC：USART 接收完成

当收到的字符从接收移位寄存器传到 UDR 中并未被读取时，该位被设置。不论是否探测到接收错误，该位都被设置。当设置禁止接收时，UCR 中的数据就会刷新，同时清零 RXC 标志。RXC 的置位产生中断请求。RXC 在读 UDR 时被自动清零。

● 位 6——TXC：USART 发送完成

当发送移位寄存器的全部数据被移出后，且在数据寄存器 UDR 中没有待发送的数据时该位置位。TXC 的置位会产生发送完成中断请求，在转入执行发送完成中断时，TXC 由硬件自动清零。TXC 位也可以通过向该位写一个逻辑“1”而被清零。

● 位 5——UDRE：USART 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。该标志位设置表示 UDR 可以写入新的发送数据。UDRE 的置位会产生发送数据寄存器空的中断请求。系统复位时，UDRE 被设置为“1”，表示数据发送已准备好。

● 位 4——FE：接收帧出错

如果在接收缓冲器中刚收到的数据被检测到帧出错时，该位被设置（例如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，读取 UDR 的操作也会将 FE 标志位清除。此外，重写寄存器 UCSRA 的操作总是设置 FE 标志位为“0”。

● 位 3——DOR：接收数据溢出出错

如果接收数据溢出条件被检测到，该位被设置。当接收缓冲器满（即有两个接收到的字符，前一个字符在 UDR 中，移位寄存器中还有一个新接收到的字符在等待），同时接收器又检测到一个新的起始位时，接收数据溢出出错发生。该位将一直保持为“1”，直到接收缓冲（UDR）被读取。重写寄存器 UCSRA 的操作总是设置 DOR 标志位为“0”。

● 位 2——PE：校验错误

在接收允许和校验位比较允许都使能时，接收器检测到刚接收的数据检验出错，那么该位将被置“1”。该位将一直保持为“1”，直到接收缓冲（UDR）被读取。重写寄存器 UCSRA 的操作总是设置 PE 标志位为“0”。

- 位 1——U2X：USART 传输速率倍速

该位只有在异步模式下有效，当使用同步模式时，应设置该位为“0”。设置该位为“1”，将使波特率分频器的分频比由 16 降为 8，其效果是使异步通信的传输速率加倍。

- 位 0——MPCM：多机通信模式允许

该位使能多机通信模式。当 MPCM 位被写为“1”时，所有接收到的数据帧，如果不包括地址信息的话，将被 USART 接收器忽略。USART 的发送模块不受 MPCM 设置的影响。具体应用见“多机通信模式”。

3、USART 控制和状态寄存器 B——UCSRB

位	7	6	5	4	3	2	1	0	
$0A($002A)	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
复位值	0	0	0	0	0	0	0	0	

- 位 7——RXCIE：RX 接收完成中断允许

当该位被置“1”时，表示允许响应接收完成中断请求。如果全局中断标志 I 为“1”，且 RXCIE 为“1”时，当标志位 RXC 置“1”，一个接收完成中断服务被执行。

- 位 6——TXCIE：TX 发送完成中断允许

当该位被置“1”时，表示允许响应发送完成中断请求。如果全局中断标志 I 为“1”，且 TXCIE 为“1”时，当标志位 TXC 置“1”，一个发送完成中断服务被执行。

- 位 5——UDRIE：USART 数据寄存器空中断允许

当该位被置“1”时，表示允许响应发送数据寄存器 UDR 空中断请求。如果全局中断标志 I 为“1”，且 UDRIE 为“1”时，当标志位 UDRE 置“1”，一个发送数据寄存器空中断服务被执行。

- 位 4——RXEN：数据接收允许

当该位被置“1”时，允许 USART 接收数据。当接收器被使能时，引脚 PD0 的特性由通用 I/O 口转变为 RxD。禁止数据接收，将清除接收缓冲器中的数据，并使 FE、DOR、PE 标志位无效。当接收数据被禁止后，USART 发送器将不再占用 RxD 引脚。

- 位 3——TXEN：发送数据允许

当该位被置为“1”时，允许 USART 发送数据。当发送器被使能时，引脚 PD1 的特性由通用 I/O 口转变为 TxD。如在发送数据时禁止发送器（写 TXEN=0），则在移位寄存器中的数据和后续 UDR 中的数据被全部发送完成之后，发送器才会被禁止。当发送器被禁止后，USART 发送器将不再占用 TxD 引脚。

- 位 2——UCSZ2：数据位数大小

该位同 UCSRC 寄存器中的 UCSZ1:0 位一起使用，它们设置接收和发送数据帧中数据

位的个数（5、6、7、8、9 位）。

- 位 1——RXB8：接收数据的第 8 位

当采用接收的数据格式为 9 位数据帧时，RXB8 中是接收到数据的第 9 数据位。RXB8 标志位必须在 URD 读之前读取。

- 位 0——TXB8：发送数据的第 8 位

当采用发送数据格式为 9 位数据帧时，TXB8 中是发送数据的第 9 数据位。TXB8 标志位必须在 URD 写入前写入。

4. USART 控制和状态寄存器 C——UCSRC

UCSRC 寄存器与 UBRRH 寄存器共享一个 I/O 空间的地址，如何读写这两个共享同一地址的寄存器见下面的说明。

位	7	6	5	4	3	2	1	0	
$20（$0040）	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	1	0	0	0	0	1	1	0	

- 位 7——URSEL：寄存器选择

该位用于选择要操作的 UCSRC 或 UBRRH 寄存器。当读取 UCSRC 寄存器时，该位读出为“1”。当写 UCSRC 寄存器时，该位必须写入“1”。

- 位 6——UMSEL：USART 工作模式选择

该位用于选择 USART 为同步或异步工作模式，如表 2-42 所示。

表 2-42 USART 工作模式

UMSEL	USART 工作模式
0	异步模式
1	同步模式

- 位 5..4——UPM1..0：校验方式

这些位用于允许和选择产生或验证校验位的类型。如果使能校验模式，发送器将根据发送的数据，自动产生符合要求的校验位，并附加在每一个数据帧后发送。接收器将对接收的数据帧进行校验，产生校验位，并同 UPM0 的设置进行比较。如果不匹配，UCSRA 寄存器中的 PE 标志位将被置“1”，如表 2-43 所示。

表 2-43 校验方式

UPM1	UPM2	校 验 方 式
0	0	无校验
0	1	保留
1	0	使能偶校验
1	1	使能奇校验

● 位 3——USBS：停止位选择，如表 2-44 所示。

表 2-44 停止位个数

USBS	停止位个数
0	1 位停止位
1	2 位停止位

该位用于选择插入到发送帧中的停止位的个数。接收器不受对该位设置的影响。

● 位 2..1——UCSZ1..0：传送或接收字符长度

该两位同 UCSRB 寄存器中的 UCSZ2 位一起使用，定义和设置接收和发送数据帧中的数据位数。

● 位 0——UCPOL：时钟极性，如表 2-45 所示。

该位只在同步模式下使用。在异步模式下，应将该位写为“0”。UCPOL 位设定了串行输出数据变化和数据输入采样与同步时钟 XCK 之间的关系。

表 2-45 时钟极性

UCPOL	串行输出数据的变化（TxD 的输出）	串行输入数据的采样（RxD 的输入）
0	XCK 上升沿	XCK 下降沿
1	XCK 下降沿	XCK 上升沿

5．波特率寄存器——UBRRL 和 UBRRH

UBRRH 寄存器与 UCSRC 寄存器共享一个 I/O 空间的地址，如何读写这两个共享同一地址的寄存器见下面的说明。

位	15	14	13	12	11	10	9	8	
$20（$0040）	URSEL	-	-	-	UBRR[11..8]				UBRRH
$09（$0029）	UBRR[7..0]								UBRRL
	7	6	5	4	3	2	1	0	
读/写	R/W	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

● 位 15——URSEL：寄存器选择

该位用于选择确定 UCSRC 寄存器和 UBRRH 寄存器的操作。如果读 UBRRH 寄存器，该位为“0”。当写 UBRRH 寄存器时，该位必须写入“0”。

● 位 14..12——保留位

为将来的使用和兼容起见，写入时置这些位为 0。

● 位 11..0——UBRR11..0：USART 波特率设置寄存器

由寄存器UBRRH低4位和寄存器UBRRL的8位构成一个12位的寄存器，用于USART传送或接收波特率的设置。如果波特率设置被改变，正在进行的接收和发送将被打断。写UBRRL将立即更新对波特率预分频的设置。

6. UBRRH/UCSRC 寄存器的读写操作

UBRRH 寄存器与 UCSRC 寄存器在 I/O 空间共享同一个地址（$20），因此对这两个寄存器的读写操作要按照一些特殊的规范进行。

（1）写操作

在对 I/O 空间地址$20 写入数据时，写入数据的最高位（即对应两个寄存器的 URSEL 位）的值用于选定写入的寄存器。最高位为“1”时，表示写入数据到寄存器 UCSRC；最高位为“0”，表示写入数据到寄存器 UBRRH。下面是一个汇编代码例子：

```
...
; Set UBRRH to 2
ldi r16,0x02
out UBRRH,r16
...
; Set the USBS and the UCSZ1 bit to one, and
; the remaining bits to zero.
ldi r16,(1<<URSEL)|(1<<USBS)|(1<<UCSZ1)
out UCSRC,r16
...
```

（2）读操作

读取 I/O 空间地址$20 处（UBRRH 或 UCSRC）的数据比写入操作复杂一些，读取寄存器的选择是通过读取时间的先后序列确定的。读一次 I/O 空间地址$20，返回的是寄存器 UBRRH 的内容；在接下来的一个时钟周期内再次读地址$20，则返回寄存器 UCSRC 的内容。也就是说在两个连续的时钟周期中，连续执行两次读 I/O 空间地址$20 的操作，那么第二次读取的是 UCSRC 寄存器。因此，读地址$20，一般读到的是寄存器 UBRRH。如要读取寄存器 UCSRC，必须连续读两次，中间不能插入任何其他操作（如不能有中断发生）。下面是一个汇编代码例子：

```
USART_ReadUCSRC:
; Read UCSRC
in r16,UBRRH
in r16,UCSRC
ret
```

2.11.10 串行通信波特率的设置与偏差

对于一些常用标准频率的晶体，大多数异步操作的波特率设置可以从表 2-46、表 2-47、表 2-48、表 2-49 中获得。使用表中的设置值产生的时钟波特率与实际的波特率的偏差小于 0.5%。虽然更高的波特率偏差也可以使用，但会降低接收器的抗干扰性。误差可以通过以下公式计算：

$$Error[\%]=(\frac{BaudRate_{ClosestMatch}}{BaudRate}-1)\times 100\%$$

表 2-46 不同晶振频率的 UBRR 设置（1）

Baud Rate (bps)	f_{osc}=1.0000MHz				f_{osc}=1.8432MHz				f_{osc}=2.0000MHz			
	U2X=0		U2X=1		U2X=0		U2X=1		U2X=0		U2X=1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	-	-	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	-	-	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	-	-	-	-	-	-	0	0.0%	-	-	-	-
250k	-		-	-	-	-	-	-	-	-	0	0.0%
Max[1]	62.5kbps		125kbps		115.2kbps		230.4Mbps		125kbps		250kbps	

注：1. UBRR=0；Error=0.0%

表 2-47 不同晶振频率的 UBRR 设置（2）

Baud Rate (bps)	f_{osc}=3.6864MHz				f_{osc}=4.0000MHz				f_{osc}=7.3728MHz			
	U2X=0		U2X=1		U2X=0		U2X=1		U2X=0		U2X=1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	95	0.0%	191	0.0%	103	0.2%	207	0.2%	191	0.0%	383	0.0%
4800	47	0.0%	95	0.0%	51	0.2%	103	0.2%	95	0.0%	191	0.0%
9600	23	0.0%	47	0.0%	25	0.2%	51	0.2%	47	0.0%	95	0.0%
14.4k	15	0.0%	31	0.0%	16	2.1%	34	-0.8%	31	0.0%	63	0.0%
19.2k	11	0.0%	23	0.0%	12	0.2%	25	0.2%	23	0.0%	47	0.0%

续表

Baud Rate (bps)	f_{osc}=3.6864MHz				f_{osc}=4.0000MHz				f_{osc}=7.3728MHz			
	U2X=0		U2X=1		U2X=0		U2X=1		U2X=0		U2X=1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
28.8k	7	0.0%	15	0.0%	8	-3.5%	16	2.1%	15	0.0%	31	0.0%
38.4k	5	0.0%	11	0.0%	6	-7.0%	12	0.2%	11	0.0%	23	0.0%
57.6k	3	0.0%	7	0.0%	3	8.5%	8	-3.5%	7	0.0%	15	0.0%
76.8k	2	0.0%	5	0.0%	2	8.5%	6	-7.0%	5	0.0%	11	0.0%
115.2k	1	0.0%	3	0.0%	1	8.5%	3	8.5%	3	0.0%	7	0.0%
230.4k	0	0.0%	1	0.0%	0	8.5%	1	8.5%	1	0.0%	3	0.0%
250k	0	-7.8%	1	-7.8%	0	0.0%	1	0.0%	1	-7.8%	3	-7.8%
0.5M	-	-	0	-7.8%	-	-	0	0.0%	0	-7.8%	1	-7.8%
1M	-	-	-	-	-	-	-	-	-	-	0	-7.8%
Max[1]	230.4kbps		460.8kbps		250kbps		0.5Mbps		460.8kbps		921.6kbps	

注：1．UBRR=0；Error=0.0%

表 2-48　不同晶振频率的 UBRR 设置（3）

Baud Rate (bps)	f_{osc}=8.0000MHz				f_{osc}=11.0592MHz				f_{osc}=14.7456MHz			
	U2X=0		U2X=1		U2X=0		U2X=1		U2X=0		U2X=1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	383	0.0%	767	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	191	0.0%	383	0.0%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	95	0.0%	191	0.0%
14.4k	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	63	0.0%	127	0.0%
19.2k	25	0.2%	51	0.2%	35	0.0%	71	0.0%	47	0.0%	95	0.0%
28.8k	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	31	0.0%	63	0.0%
38.4k	12	0.2%	25	0.2%	17	0.0%	35	0.0%	23	0.0%	47	0.0%
57.6k	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	15	0.0%	31	0.0%
76.8k	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	11	0.0%	23	0.0%
115.2k	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	7	0.0%	15	0.0%
230.4k	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	0.0%	7	0.0%
250k	1	0.0%	3	0.0%	2	-7.8%	5	-7.8%	3	-7.8%	6	5.3%
0.5M	0	0.0%	1	0.0%	-	-	2	-7.8%	1	-7.8%	3	-7.8%
1M	-	-	0	0.0%	-	-	-	-	0	-7.8%	1	-7.8%
Max[1]	0.5Mbps		1Mbps		691.2kbps		1.3824Mbps		921.6kbps		1.8432Mbps	

注：1．UBRR=0；Error=0.0%

表 2-49 不同晶振频率的 UBRR 设置（4）

Baud Rate (bps)	f_{osc}=16.0000MHz				f_{osc}=18.4320MHz				f_{osc}=20.0000MHz			
	U2X=0		U2X=1		U2X=0		U2X=1		U2X=0		U2X=1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4k	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2k	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8k	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4k	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6k	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8k	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%
230.4k	3	8.5%	8	-3.5%	4	0.0%	9	0.0%	4	8.5%	10	-1.4%
250k	3	0.0%	7	0.0%	4	-7.8%	8	2.4%	4	0.0%	9	0.0%
0.5M	1	0.0%	3	0.0%	-	-	4	-7.8%	-	-	4	0.0%
1M	0	0.0%	1	0.0%	-	-	-	-	-	-	-	-
Max[1]	1Mbps		2Mbps		1.152Mbps		2.304Mbps		1.25Mbps		2.5Mbps	

注：1．UBRR=0；Error=0.0%

2.12 两线串行 TWI（I²C）总线接口

ATmega8 单片机提供了实现标准两线串行总线通信的硬件接口 TWI（即 I²C 总线）。其主要的性能和特点有：

- 只需两根线的强大而灵活的串行通信接口；
- 支持主控器/被控器操作模式；
- 器件可作为发送器或接受器；
- 7 位的地址空间，支持最大为 128 个从机地址；
- 支持多主机模式；
- 高达 400kHz 的数据传输率；
- 斜率受限的输出驱动器；
- 噪声监控电路防止总线上的毛刺；
- 全可编程的从机地址；
- 地址监听中断使 AVR 从休眠模式唤醒。

2.12.1 两线串行总线接口定义

两线串行接口（TWI）是单片机应用的理想接口。采用 TWI 协议，系统设计者通过两根双向的总线（一根称为时钟线 SCL，另一根称为数据线 SDA）连接 128 个从设备。实现这种总线连接时，惟一需要增加的外部器件是每个总线上的上拉电阻（见图 2.65）。所有与总线相连的设备都有各自的设备地址。

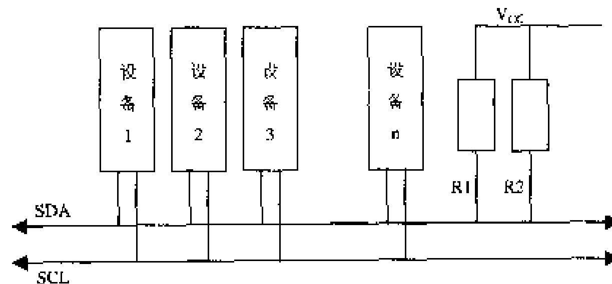


图 2.65 TWI 总线连接

TWI 和 I^2C 是兼容的，关于 TWI 的协议可参见有关 I^2C 总线的定义和说明。

2.12.2 TWI 模块的概述

ATmega8 的 TWI 模块由几个子模块构成，如图 2.66 所示。所有寄存器可通过 CPU 数据总线进行读写。

1. SCL 和 SDA 引脚

SCL 和 SDA 为 MCU 的 TWI 接口的引脚。引脚的输出驱动器包含一个斜率限制器以服从 TWI 规范。引脚输入部分包括毛刺抑制单元以去除小于 50ns 的毛刺。注意，当对应的端口设置为 SCL 和 SDA 引脚时，可以设定相应 I/O 口内部的上拉电阻有效，这样能省掉外部的拉高电阻。

2. 波特率发生器

TWI 工作在主控器模式下时，由该单元控制产生时钟信号并驱动时钟线 SCL。时钟 SCL 的周期由 TWI 状态寄存器 TWSR 中的预分频位和 TWI 波特率寄存器 TWBR 设定。当 TWI 工作在被控器模式下时，不需要对波特率或预分频进行设定，但作为被控器，其 CPU 的时钟频率必须大于 TWI 时钟线 SCL 频率的 16 倍。SCL 的频率依据以下的等式产生：

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \times 4^{\text{TWPS}}}$$

其中，TWBR 为 TWI 波特率寄存器的值；TWPS 为 TWI 状态寄存器预分频位的值。在主机模式下，TWBR 的值应大于 10，否则可能会产生不正确的输出。

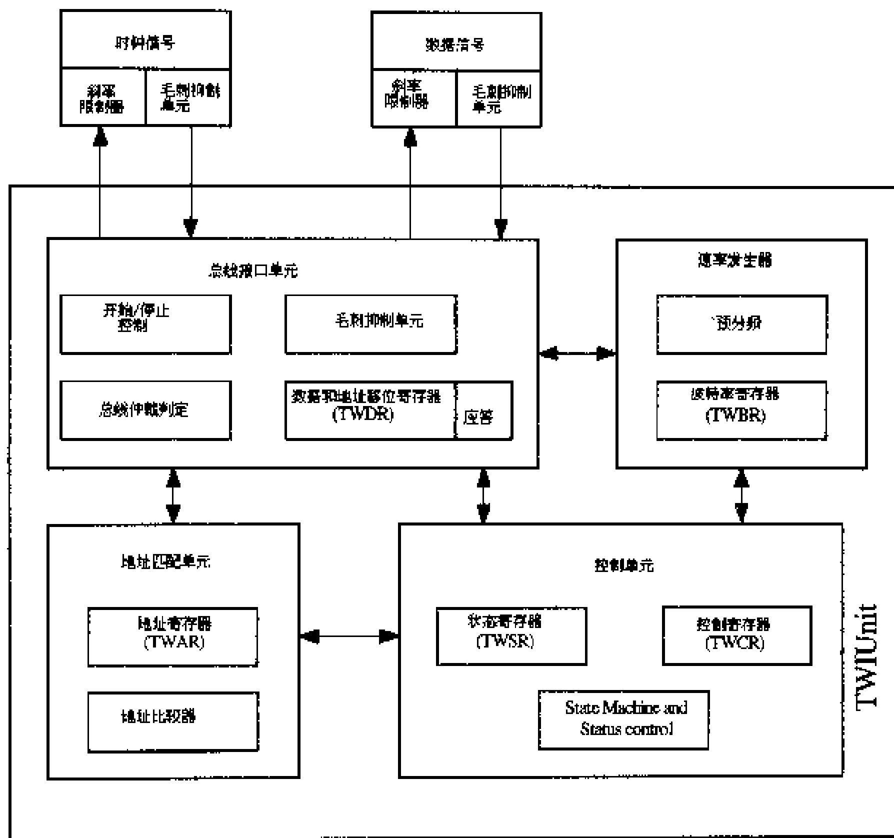


图 2.66　ATmega8 的 TWI 模块结构图

3．总线接口单元

这个单元包括：数据和地址移位寄存器 TWDR，起始/终止信号（START/STOP）控制和总线仲裁判定的硬件电路。TWDR 寄存器用于存放传送或接收的数据和地址。除了 8 位的 TWDR，总线接口单元还有一个寄存器，含有用于传送或接收应答（ACK）位——（N）ACK。这个（N）ACK 寄存器不能由程序直接读写。当接收数据时，它可以通过 TWI 控制寄存器 TWCR 来置“1”或清“0”。在发送数据时，（N）ACK 值由 TWSR 的设置决定。起始/终止信号（START/STOP）控制电路负责 TWI 总线上的 START、REPEATED START 和 STOP 逻辑时序的发生和检测。当 MCU 处于休眠状态时，START/STOP 控制器能够检测 TWI 总线上的 START/STOP 条件，当检测到被 TWI 总线上主控器寻址访问时，将 MCU 从休眠状态唤醒。

如果设置 TWI 接口作为主控器，在发送数据前，总线仲裁判定硬件电路会持续监控总线，以确定是否可以通过仲裁获得总线控制权。如果总线仲裁单元检测到自己在总线仲裁中丢失总线控制权，则通知 TWI 控制单元进行正确的总线行为的转换。

4．地址匹配单元

地址匹配单元将检测从总线上接收到的地址是否与 TWAR 寄存器中的 7 位地址相匹配。如果 TWAR 寄存器中的 TWI 广播应答位 TWGCE 被写为“1”，所有从总线上接收到的地址也会和广播地址进行比较。一旦地址匹配成功，将通知控制单元转入适当的操作状态。TWI 可以响应或不响应主控器对其的寻址访问，这取决于 TWCR 寄存器中的设置。当

MCU 处于休眠状态中，地址匹配单元仍可继续工作，在使能被主控器寻址唤醒，且地址匹配单元检验到接收的地址与自己地址匹配时，将 MCU 从休眠状态唤醒。在 TWI 由于地址匹配将 MCU 从掉电状态唤醒期间，如有其他中断发生，TWI 将放弃操作，返回其空闲状态。如果这会引起其他的问题，请在进入掉电休眠时，保证只有 TWI 地址匹配中断被使能。

5．控制单元

控制单元监视 TWI 总线，并根据 TWI 控制寄存器 TWCR 的设置做出相应的响应。当在 TWI 总线上产生需要应用程序干预处理的事件时，先对 TWI 的中断标志位 TWINT 进行相应设置，在下一个时钟周期时，将表示这个事件的状态字写入 TWI 状态寄存器 TWSR 中。在其他情况下，TWSR 中的内容为一个表示无事件发生的状态字。一旦 TWINT 标志位置“1”，就会将时钟线 SCL 拉低，暂停 TWI 总线上的传送，让用户程序处理事件。

在下列状态（事件）出现时，TWINT 标志位设置为“1”：

- 在 TWI 传送完一个起始或再次起始（START/REPEATED START）信号后；
- 在 TWI 传送完一个主控器寻址读写（SLA+R/W）数据后；
- 在 TWI 传送完一个地址字节后；
- 在 TWI 丢失总线控制权后；
- 在 TWI 被主控器寻址（地址匹配成功）后；
- 在 TWI 接收到一个数据字节后；
- 在作为被控器时，TWI 接收到终止或再次起始信号（STOP/REPEATED START）后；
- 由于非法的起始或终止信号造成总线上冲突出错时。

2.12.3 TWI 寄存器

1．TWI 波特率寄存器——TWBR

位	7	6	5	4	3	2	1	0	
\$00（\$0020）	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

- 位 7..0——TWBRn：TWI 波特率寄存器

TWBR 用于设置波特率发生器的分频因子。波特率发生器是一个频率分频器，当工作在主控器模式（MASTER MODE）下，它产生和提供 SCL 引脚上的时钟信号。计算公式见波特率发生器的介绍。

2．TWI 控制寄存器——TWCR

TWCR 寄存器用于 TWI 接口模块的操作控制。如使能 TWI 接口；在总线上加起始信号（START）来初使化一次主控器的寻址访问；产生 ACK 应答；产生终止信号（STOP）；在写入数据到 TWDR 寄存器时控制总线的暂停（HALTING）等。在禁止访问 TWDR 期间，如试图将数据写入到 TWDR 时要给出写入冲突标志。

位	7	6	5	4	3	2	1	0	
$36（$0056）	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
读/写	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
复位值	0	0	0	0	0	0	0	0	

● 位 7——TWINT：TWI 中断标志位

当 TWI 接口完成当前工作，期待应用程序响应时，该位被置位。如果 SREG 寄存器中的 I 位和 TWCR 寄存器中的 TWIE 位为“1”时，MCU 将跳到 TWI 中断向量。一旦 TWINT 标志位被置位，时钟线 SCL 将被拉为低。在执行中断服务程序时，TWINT 标志位不会由硬件自动清零，必须通过由软件写入逻辑“1”来清零。清零 TWINT 标志位将开始 TWI 接口的操作，因此对 TWI 地址寄存器 TWAR、TWI 状态寄存器 TWSR 和 TWI 数据寄存器 TWDR 的访问，必须在清零 TWINT 标志位前完成。

● 位 6——TWEA：TWI 应答（ACK）允许

TWEA 位控制应答 ACK 信号的发生。如果 TWEA 位被置“1”时，在以下情况下 ACK 脉冲将在 TWI 总线上发生：

（1）器件作为被控器时，接收到呼叫自己的地址；

（2）当 TWAR 寄存器中的 TWGCE 位被置位时，接收到一个通用呼叫地址；

（3）器件作为主控器接收器或被控器接收器时，接收到一个数据字节。

如果清零 TWEA 位，将使器件暂时虚拟地脱离 TWI 总线。地址识别匹配功能需通过置 TWEA 位为“1”来重新开始。

● 位 5——TWSTA：TWI 起始（START）信号状态位

当要将器件设置为串行总线上的主控器时，则需设置 TWSTA 位为“1”。TWI 接口硬件将检查总线是否空闲。如果总线空闲，将在总线上发出一个起始（START）信号。但是如果总线并不空闲，TWI 将等到总线上一个终止（STOP）信号被检测到后，再发出一个新的起始（START）信号，以获得总线的控制权而成为主控器。当起始（START）信号发出后，TWSTA 位将由硬件清零。

● 位 4——TWSTO：TWI 终止（STOP）信号状态位

当芯片在主控器模式时，设置 TWSTO 位为“1”，将在总线上发出一个终止（STOP）信号。当终止（STOP）信号发出后，TWSTO 位将被自动清零。当芯片在被控器模式时，置位 TWSTO 位用于从错误状态恢复。此时，TWI 接口并不发出终止（STOP）信号，但硬件接口模块返回正常的初始未被寻址的被控器模式，并释放 SCL 和 SDA 线为高阻状态。

● 位 3——TWWC：TWI 写冲突标志位

在 TWINT 位为“0”时，试图向 TWI 数据寄存器 TWDR 写数据，TWWC 位将被置位。在 TWINT 位为“1”时，写 TWDR 寄存器将自动清零 TWWC 标志位。

● 位 2——TWEN：TWI 允许位

TWEN 位用于使能 TWI 接口操作和激活 TWI 接口。当 TWEN 位被写为“1”时，TWI 接口模块将 I/O 引脚 PC5 和 PC4 转换成 SCL 和 SDA 引脚，使能斜率限制器和毛刺滤波器。

如果该位被清零，TWI 接口模块将被关闭，所有 TWI 传输将被终止。

- 位 1——保留

该位被保留，读出总为“0”。

- 位 0——TWIE：TWI 中断使能

当该位被写为“1”，同时 SREG 寄存器中的 I 位被置位时，只要 TWINT 标志位为“1”时，TWI 中断请求即被使能。

3. TWI 状态寄存器——TWSR

位	7	6	5	4	3	2	1	0	
$01（$0021）	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
读/写	R	R	R	R	R	R	R	R/W	
复位值	1	1	1	1	1	1	0	0	

- 位 7..3——TWS：TWI 状态

这 5 位反映了 TWI 逻辑状态和 TWI 总线的状态。不同状态码将在下一节描述。注意，从 TWSR 寄存器中读取的值包括了 5 位状态值和 2 位预分频值。因此，当检查状态位时，应该将预分频器位屏蔽，使状态检验与预分频器无关。

- 位 2——保留

该位被保留，读出始终为“0”。

- 位 1..0——TWPS：TWI 预分频器位

这些位能被读或写，用于设置波特率的预分频率（见表 2-50）。详见比特率发生单元部分来了解如何计算比特率。

表 2-50 TWI 波特率预分频率设置

TWPS1	TWPS0	预 分 频 值
0	0	1
0	1	4
1	0	16
1	1	64

4. TWI 数据寄存器——TWDR

在发送模式下，TWDR 寄存器的内容为下一个要传送的字节。在接收模式下，TWDR 寄存器中的内容为最后接收的字节。当 TWI 不处在字节移位操作过程时，该寄存器可以被写，即当 TWI 中断标志位（TWINT）由硬件置位时，其可以被写。注意：在第一次 TWI 中断发生前，数据寄存器不能由用户初使化。当 TWINT 位被置位时，TWDR 中的数据保持稳定。当数据被移出时，总线上的数据同时也被移入，因此，TWDR 的内容总是总线上出现的最后字节，除非当 MCU 从休眠模式中由 TWI 中断而唤醒。当 MCU 由 TWI 中断唤醒时，TWDR 中的内容不是确定的。在丢失总线的控制权，器件由主控器转变为被控器的过程中，数据不会丢失。TWI 硬件逻辑电路自动控制 ACK 的处理，CPU 不能直接访问 ACK 位。

位	7	6	5	4	3	2	1	0	
\$03（\$0023）	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	1	1	1	1	1	1	1	1	

- 位 7..0——TWD：TWI 数据寄存器

这 8 位包括将要传送的下一个数据字节，或 TWI 总线上的最后一个接收到的数据字节。

5．TWI（被控器）地址寄存器——TWAR

TWAR 寄存器的高 7 位的内容为被控器的 7 位地址字。当 TWI 被设置为被控接收器或被控发送器时，在 TWAR 中应设置被控器寻址地址。而在主控器模式下，不需要设置 TWAR。在多主竞争的总线系统中，如果器件的角色既可为主控器又可为被控器时，必须设置 TWAR 寄存器。TWAR 寄存器的最低位用作通用地址（或广播地址 0x00）的识别允许位。相应的地址比较单元将会在接收的地址中寻找从机地址或通用呼叫地址（或广播地址）。如果发现总线下发的地址与 TWAR 指定地址匹配，将产生 TWI 中断请求。

位	7	6	5	4	3	2	1	0	
\$02（\$0022）	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	1	1	1	1	1	1	1	0	

- 位 7..1——TWA：TWI 被控器地址寄存器

该 7 位用作存放 TWI 单元的被控器地址。

- 位 0——TWGCE：TWI 通用呼叫（或广播呼叫）识别允许位

如果该位被置位，将使能对 TWI 总线上的通用地址的呼叫（或广播呼叫）和识别。

2.12.4 使用 TWI 总线

AVR 的 TWI 接口是面向字节和基于中断的。在所有总线事件后，如接收到一个字节或发送了一个起始（START）信号等，都将产生一个 TWI 中断。由于 TWI 接口是基于中断的，因此 TWI 接口在字节传送和接收过程中，不需要应用程序的干预。TTWCR 寄存器中的 TWI 中断允许 TWIE 位和 SREG 寄存器中的全局中断允许 I 位一起决定了应用程序是否响应 TWINT 标志位的有效而发生中断请求。如果 TWIE 被清零，应用程序只能采用轮循 TWINT 标志位的方法来检测 TWI 总线的状态。

当 TWINT 标志位置“1”时，表示 TWI 接口完成了当前的操作，等待应用程序的响应。在这种情况下，TWI 状态寄存器 TWSR 含有表明当前 TWI 总线状态的值。应用程序可以读取 TWSR 的状态码，判别此时的状态是否正确，并通过设置 TWCR 和 TWDR 寄存器，决定在下一个 TWI 总线周期中，TWI 接口应该如何工作。

连接在 TWI 串行总线上的单片机或集成电路芯片，通过一条数据线（SDA）和一条时钟线（SCL），按照 TWI 通信协议（与 I^2C 兼容）进行寻址和信息传输。TWI 总线上的器

件，根据它的不同工作状态，可分为主控发送器（MT）、主控接收器（MR）、被控发送器（ST）、被控接收器（SR）。在实际应用中，器件的工作状态可以根据需要进行转换。

关于 TWI 的协议和应用，可参考有关 I^2C 总线的定义和说明。下面给出 ATmega8 的 TWI 接口在 4 种工作状态时的相应的状态字，以及进一步的操作（如表 2-51~2-55 所示）。

表 2-51 TWI 主控发送器模式状态字

状态字（TWSR）低 3 位屏蔽 0	TWI 接口总线状态	应用程序响应操作					TWI 接口下一步的动作
		读/写 TWDR	写 TWCR				
			STA	STO	TWINT	TWEA	
0x08	START 信号已发出	写 SLA+W	X	0	1	X	发送 SLA+W 接收 ACK/NACK 信号
0x10	REPEATED START 信号已发出	写 SLA+W	X	0	1	X	发送 SLA+W， 接收 ACK/NACK 信号
		写 SLA+R	X	0	1	X	发送 SLA+R， 接收 ACK/NACK 信号
0x18	SLA+W 已发出并收到 ACK	写DATA字节	0	0	1	X	发送 DATA， 接收 ACK/NACK 信号
		无操作	1	0	1	X	发送 REPEATED START
		无操作	0	1	1	X	发送 STOP 信号， 清零 TWSTO
		无操作	1	1	1	X	发送 START、STOP 信号，清零 TWSTO
0x20	SLA+W 已发出并收到 NACK	写DATA字节	0	0	1	X	发送 DATA， 接收 ACK/NACK 信号
		无操作	1	0	1	X	发送 REPEATED START
		无操作	0	1	1	X	发送 STOP 信号， 清零 TWSTO
		无操作	1	1	1	X	发送 START、STOP 信号，清零 TWSTO
0x28	DATA 已发出并收到 ACK	写DATA字节	0	0	1	X	发送 DATA， 接收 ACK/NACK 信号
		无操作	1	0	1	X	发送 REPEATED START
		无操作	0	1	1	X	发送 STOP 信号， 清零 TWSTO
		无操作	1	1	1	X	发送 START、STOP 信号，清零 TWSTO

续表

状态字（TWSR）低 3 位屏蔽 0	TWI 接口总线状态	应用程序响应操作					TWI 接口下一步的动作
		读/写 TWDR	写 TWCR				
			STA	STO	TWINT	TWEA	
0x30	DATA 已发出并收到 NACK	写DATA字节	0	0	1	X	发送 DATA，接收 ACK/NACK 信号
		无操作	1	0	1	X	发送 REPEATED START
		无操作	0	1	1	X	发送 STOP 信号，清零 TWSTO
		无操作	1	1	1	X	发送 START、STOP 信号，清零 TWSTO
0x38	丢失总线控制权	无操作	0	0	1	X	释放总线，转入被控器初始状态
		无操作	1	0	1	X	如果总线空闲，发送 START 信号

表 2-52　TWI 主控接收器模式状态字

状态字（TWSR）低 3 位屏蔽 0	TWI 接口总线状态	应用程序响应操作					TWI接口下一步的动作
		读/写 TWDR	写 TWCR				
			STA	STO	TWINT	TWEA	
0x08	START 信号已发出	写 SLA+R	X	0	1	X	发送 SLA+R 接收 ACK/NACK 信号
0x10	REPEATED START 信号已发出	写 SLA+R	X	0	1	X	发送 SLA+R，接收 ACK/NACK 信号
		写 SLA+W	X	0	1	X	发送 SLA+W，接收 ACK/NACK 信号
0x38	丢失总线控制权未收到应答信号	无操作	0	0	1	X	释放总线,转入被控器初始状态
		无操作	1	0	1	X	如果总线空闲，发送 START 信号
0x40	SLA+R 已发出并收到 ACK	无操作	0	0	1	0	接收 DATA，发送 ACK 信号
		无操作	0	0	1	1	接收 DATA，发送 NACK 信号

续表

状态字（TWSR）低 3 位屏蔽 0	TWI 接口总线状态	应用程序响应操作					TWI 接口下一步的动作
		读/写 TWDR	写 TWCR				
			STA	STO	TWINT	TWEA	
0x48	SLA+R 已发出并收到 NACK	无操作	1	0	1	X	发送 REPEATED START
		无操作	0	1	1	X	发送 STOP 信号，清零 TWSTO
		无操作	1	1	1	X	发送 START、STOP 信号，清零 TWSTO
0x50	DATA 已收到 ACK 已发出	读 DATA 数据	0	0	1	0	接收 DATA，发送 ACK 信号
		读 DATA 数据	0	0	1	1	接收 DATA，发送 NACK 信号
0x58	DATA 已收到 NACK 已发出	读 DATA 数据	1	0	1	X	发送 REPEATED START
		读 DATA 数据	0	1	1	X	发送 STOP 信号，清零 TWSTO
		读 DATA 数据	1	1	1	X	发送 START、STOP 信号，清零 TWSTO

表 2-53 TWI 被控接收器模式状态字

状态字（TWSR）低 3 位屏蔽 0	TWI 接口总线状态	应用程序响应操作					TWI 接口下一步的动作
		读/写 TWDR	写 TWCR				
			STA	STO	TWINT	TWEA	
0x60	收到本机 SLA+W ACK 已发出	无操作	X	0	1	0	接收 DATA，发送 NACK 信号
		无操作	X	0	1	1	接收 DATA，发送 ACK 信号
0x68	主控器发出 SAL+R/W 后丢失总线控制权 收到本机 SLA+W ACK 已发出	无操作	X	0	1	0	接收 DATA，发送 NACK 信号
		无操作	X	0	1	1	接收 DATA，发送 ACK 信号
0x70	收到广播呼叫 ACK 已发出	无操作	X	0	1	0	接收 DATA，发送 NACK 信号
		无操作	X	0	1	1	接收 DATA，发送 ACK 信号

续表

状态字（TWSR）低 3 位屏蔽 0	TWI 接口总线状态	应用程序响应操作					TWI 接口下一步的动作
		读/写 TWDR	写 TWCR				
			STA	STO	TWINT	TWEA	
0x78	主控器发出 SAL+R/W 丢失总线控制权 收到广播呼叫 ACK 已发出	无操作	X	0	1	0	接收 DATA，发送 NACK 信号
		无操作	X		1	1	接收 DATA，发送 ACK 信号
0x80	已被 SLA+W 寻址 DATA 已收到 ACK 已发出	读 DATA 数据	X	0	1	0	接收 DATA，发送 NACK 信号
		读 DATA 数据	X	0	1	1	接收 DATA，发送 ACK 信号
0x88	已被 SLA+W 寻址 DATA 已收到 NACK 已发出	读 DATA 数据	0	0	1	0	转入被控器初始状态 不进行本机 SLA 和广播呼叫匹配
		读 DATA 数据	0	0	1	1	转入被控器初始状态 进行本机 SLA 匹配 如 TWGCE=1，进行广播呼叫匹配
		读 DATA 数据	1	0	1	0	转入被控器初始状态 不进行本机 SLA 和广播呼叫匹配 如果总线空闲，发送 START 信号
		读 DATA 数据	1	0	1	1	转入被控器初始状态 进行本机 SLA 匹配 如 TWGCE=1，进行广播呼叫匹配 如果总线空闲，发送 START 信号
0x90	已被广播呼叫寻址 DATA 已收到 ACK 已发出	读 DATA 数据	X	0	1	0	接收 DATA，发送 NACK 信号
		读 DATA 数据	X	0	1	1	接收 DATA，发送 ACK 信号

续表

状态字（TWSR）低3位屏蔽0	TWI接口总线状态	应用程序响应操作					TWI接口下一步的动作
		读/写TWDR	写TWCR				
			STA	STO	TWINT	TWEA	
0x98	已被广播呼叫寻址 DATA已收到 NACK已发出	读DATA数据	0	0	1	0	转入被控器初始状态 不进行本机SLA和广播呼叫匹配
		读DATA数据	0	0	1	1	转入被控器初始状态 进行本机SLA匹配 如TWGCE=1，进行广播呼叫匹配
		读DATA数据	1	0	1	0	转入被控器初始状态 不进行本机SLA和广播呼叫匹配 如果总线空闲，发送START信号
		读DATA数据	1	0	1	1	转入被控器初始状态 进行本机SLA匹配 如TWGCE=1，进行广播呼叫匹配 如果总线空闲，发送START信号
0xA0	仍处在被寻址的被控器状态 STOP或REPEATED START已收到	读DATA数据	0	0	1	0	转入被控器初始状态 不进行本机SLA和广播呼叫匹配
		读DATA数据	0	0	1	1	转入被控器初始状态 进行本机SLA匹配 如TWGCE=1，进行广播呼叫匹配
		读DATA数据	1	0	1	0	转入被控器初始状态 不进行本机SLA和广播呼叫匹配 如果总线空闲，发送START信号
		读DATA数据	1	0	1	1	转入被控器初始状态 进行本机SLA匹配 如TWGCE=1，进行广播呼叫匹配 如果总线空闲，发送START信号

表 2-54　TWI 被控发送器模式状态字

状态字（TWSR）低 3 位屏蔽 0	TWI 接口总线状态	应用程序响应操作					TWI 接口下一步的动作
		读/写 TWDR	写 TWCR				
			STA	STO	TWINT	TWEA	
0xA8	收到本机 SLA+R ACK 已发出	写DATA字节	X	0	1	0	发送最后一个 DATA，接收 NACK 信号
		写DATA字节	X	0	1	1	发送 DATA，接收 ACK 信号
0xB0	主控器发出 SAL+R/W 后丢失总线控制权 收到本机 SLA+R ACK 已发出	写DATA字节	X	0	1	0	发送最后一个 DATA，接收 NACK 信号
		写DATA字节	X	0	1	1	发送 DATA，接收 ACK 信号
0xB8	DATA 已收发出 收到 ACK 信号	写DATA字节	X	0	1	0	发送最后一个 DATA，接收 NACK 信号
		写DATA字节	X	0	1	1	发送 DATA，接收 ACK 信号
0xC0	DATA 已收发出 收到 NACK 信号	无操作	0	0	1	0	转入被控器初始状态 不进行本机 SLA 和广播呼叫匹配
		无操作	0	0	1	1	转入被控器初始状态 进行本机 SLA 匹配 如 TWGCE=1，进行广播呼叫匹配
		无操作	1	0	1	0	转入被控器初始状态 不进行本机 SLA 和广播呼叫匹配 如果总线空闲，发送 START 信号
		无操作	1	0	1	1	转入被控器初始状态 进行本机 SLA 匹配 如 TWGCE=1，进行广播呼叫匹配 如果总线空闲，发送 START 信号

续表

状态字（TWSR）低3位屏蔽0	TWI 接口总线状态	应用程序响应操作					TWI接口下一步的动作
		读/写 TWDR	写 TWCR				
			STA	STO	TWINT	TWEA	
0xC8	最后一个DATA 已发出（TWEA=0）收到 ACK 信号	无操作	0	0	1	0	转入被控器初始状态不进行本机 SLA 和广播呼叫匹配
		无操作	0	0	1	1	转入被控器初始状态进行本机 SLA 匹配如 TWGCE=1，进行广播呼叫匹配
		无操作	1	0	1	0	转入被控器初始状态不进行本机 SLA 和广播呼叫匹配如果总线空闲，发送 START 信号
		无操作	1	0	1	1	转入被控器初始状态进行本机 SLA 匹配如 TWGCE=1，进行广播呼叫匹配如果总线空闲，发送 START 信号

表 2-55 TWI 其他状态字

状态字（TWSR）低3位屏蔽0	TWI 接口总线状态	应用程序响应操作					TWI接口下一步的动作
		读/写 TWDR	写 TWCR				
			STA	STO	TWINT	TWEA	
0xF8	无相应有效状态 TWINT = 0	无操作	无操作				等待或继续当前传送
0x00	由于非法的 START 和 STOP 信号引起总线错误	无操作	0	1	1	X	仅本机硬件 STOP，并不发送到总线，释放总线，清零 TWSTO

图 2.67 是一个应用程序如何应用 TWI 硬件接口的简单视图和例程。在该例中主控器采用轮循方式向被控器发送一个单字节的数据汇编程序代码，如表 2-56 所示。

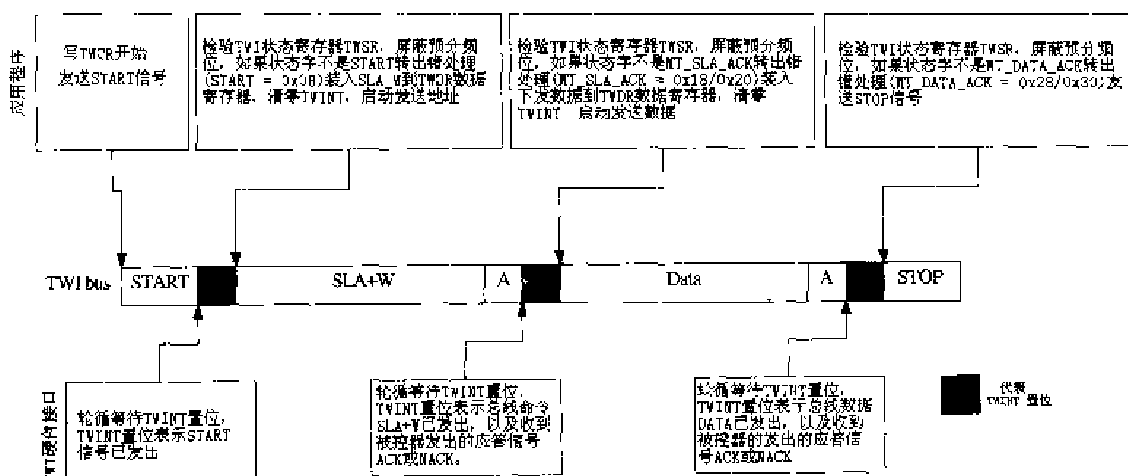


图 2.67　TWI 数据发送处理过程

表 2-56　汇编程序代码

	汇编程序代码	说　明
1	ldi r16,（1<<TWINT）|（1<<TWSTA）|（1<<TWEN） out TWCR, r16	发送 START 信号
2	wait1: in r16,TWCR sbrs r16,TWINT rjmp wait1	轮循等待 TWINT 置位， TWINT 置位表示 START 信号已发出
3	in r16,TWSR andi r16, 0xF8 cpi r16, START brne ERROR	检验 TWI 状态寄存器 TWSR， 屏蔽预分频位， 如果状态字不是 START 转出错处理（START = 0x08）

	ldi r16, SLA_W
	out TWDR, r16
	ldi r16,（1<<TWINT）|（1<<TWEN）
	out TWCR, r16
		装入 SLA_W 到 TWDR 数据寄存器，
		清零 TWINT，启动发送地址

4	wait2: in r16,TWCR sbrs r16,TWINT rjmp wait2	轮循等待 TWINT 置位， TWINT 置位表示总线命令 SLA+W 已发出，以及收到被控器发出的应答信号 ACK 或 NACK
5	in r16,TWSR andi r16, 0xF8 cpi r16, MT_SLA_ACK brne ERROR	检验 TWI 状态寄存器 TWSR， 屏蔽预分频位， 如果状态字不是 MT_SLA_ACK 转出错处理（MT_SLA_ACK = 0x18/0x20）

	ldi r16, DATA
	out TWDR, r16
	ldi r16,（1<<TWINT）|（1<<TWEN）
	out TWCR, r16
		装入下发数据到 TWDR 数据寄存器，
		清零 TWINT，启动发送数据

续表

	汇编程序代码	说　明
6	wait3: in r16,TWCR sbrs r16,TWINT rjmp wait3	轮循等待 TWINT 置位， TWINT 置位表示总线数据 DATA 已发出，以及收到被控器发出的应答信号 ACK 或 NACK
7	In r16,TWSR andi r16, 0xF8 cpi r16, MT_DATA_ACK brne ERROR	检验 TWI 状态寄存器 TWSR， 屏蔽预分频位，如果状态字不是 MT_DATA_ACK 转出错处理（MT_DATA_ACK = 0x28/0x30）
	ldi r16,（1<<TWINT）|（1<<TWEN）|（1<<TWSTO） out TWCR, r16	发送 STOP 信号

2.12.5　多主机系统和仲裁

如果有多个主控器单元连接在同一总线上，它们中的一个或多个也许会同时开始一个数据传送。TWI 协议确保在这种情况下，通过一个仲裁过程，允许其中的一个主控器进行传送而不会丢失数据。图 2.68 所示为多主控器的 TWI 系统，一个总线仲裁的例子如下所述，该例子中有两个主机试图向从接收器发送数据。

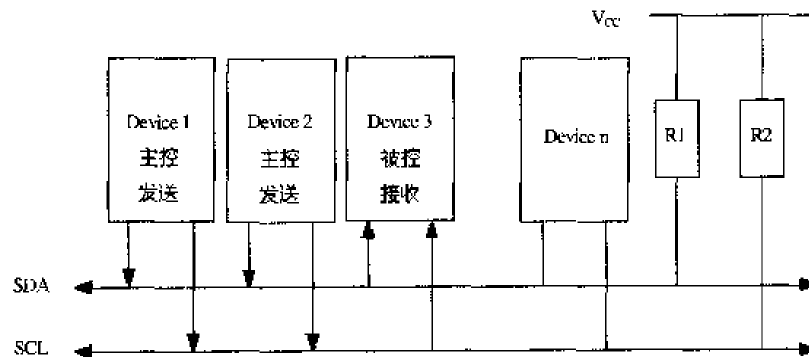


图 2.68　多主控器的 TWI 系统

有几种不同的情况会产生总线仲裁过程：

- 两个或更多的主控器同时与同一个被控器进行通信。在这种情况下，无论主控器或被控器都不知道有总线的竞争。
- 两个或更多的主控器同时对同一个被控器进行不同数据或方向的访问。在这种情况下，会在读/写或数据之间发生总线仲裁。主控器试图在 SDA 线上输出一个高电平时，如有其他的主控器已经输出了一个零，该主控器则在总线仲裁中失败。失败的主控器将转换成未被寻址的被控器模式，或等待总线空闲后发送一个新的开始条件（取决于应用程序）。
- 两个或更多的主控器访问不同的被控器。在这种情况下，总线仲裁在 SLA（被控器

地址）发生。主控器试图在 SDA 线上输出一个高电平时，如有其他的主控器已经输出了一个零，则该主控器将在总线仲裁中失败。在 SLA 总线仲裁中失败的主控器将切换到被控器模式，并检查自己是否被获得总线控制权的主控器寻址。如果被寻址，它将进入 SR（被控接收）或 ST（被控发送）模式，这取决于 SLA 的读/写位的值。如果它未被寻址，将转换到未被寻址的被控器模式或等待总线空闲，发送一个新的开始条件（这取决于应用程序）。

图 2.69 描述了总线仲裁的过程，图中的数字为 TWI 的状态值。

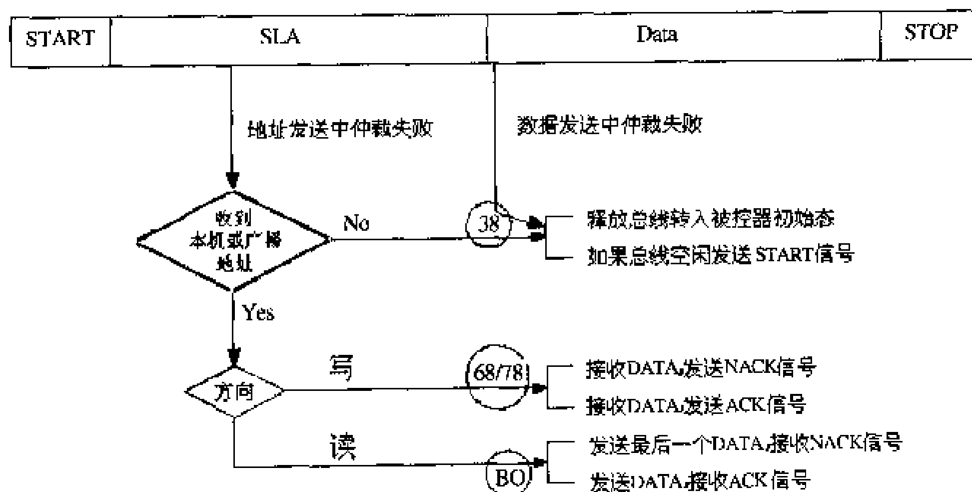


图 2.69　总线仲裁的过程

2.13　模拟比较器

模拟比较器对两个模拟输入端（正极 AIN0、负极 AIN1）的输入电压进行比较。当 AIN0 上的电压高于 AIN1 的电压时，模拟比较器输出 ACO 被设为“1”。比较器的输出可以被设置成定时/计数器 1 输入捕获功能的触发信号。此外，比较器的输出可以触发一个独立的模拟比较器中断。用户可以选择使用比较器输出的上升沿、下降沿或事件触发作为模拟比较器中断的触发信号。比较器的方框图和周围电路如图 2.70 所示。

1．特殊功能 IO 寄存器——SFIOR

位	7	6	5	4	3	2	1	0	
$30（$0050）	–	–	–	ADHSM	ACME	PUD	PSR2	PSR10	SFIOR
读/写	R	R	R	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

● 位 3——ACME：模拟比较器多路使能

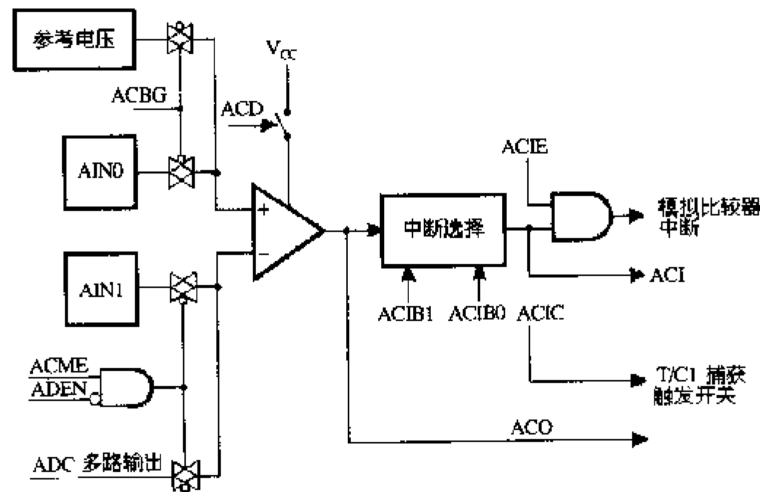


图 2.70 模拟比较器的方框图

当该位为逻辑“1”，同时模数转换（ADC）功能被关闭（ADCSRA 寄存器中的 ADEN 使能位为“0”）时，可使用 ADC 多路复用器选择 ADC 的端口作为模拟比较器反向端的信号源。当该位为零时，AIN1 引脚的信号将加到模拟比较器反向端。

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($0028)	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
读/写	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	N/A	0	0	0	0	0	

- 位 7——ACD：模拟比较器禁止

当该位设为“1”时，提供给模拟比较器的电源关闭。该位可以在任何时候被置位，从而关闭模拟比较器。在 MCU 闲置模式，且无需将模拟比较器作为唤醒源的情况下，关闭模拟比较器可以减少电源的消耗。要改变 ACD 位的设置时，应该先通过清空寄存器 ACSR 中的 ACIE 位，把模拟比较器中断禁止掉。否则，在 ADC 位改变设置时会产生一个中断。

- 位 6——ACBG：模拟比较器的间隙参考源选择

当该位为“1”时，芯片内部一个固定的能隙（Bandgap）参考电源将代替 AIN0 的输入，作为模拟比较器的正极输入端。当该位被清零时，AIN0 的输入仍然作为模拟比较器的正极输入端。

- 位 5——ACO：模拟比较器输出

模拟比较器的输出信号经过同步处理后直接与 ACO 相连。由于同步处理，ACO 与模拟比较器的输出之间，会有 1~2 个时钟的延时。

- 位 4——ACI：模拟比较器中断标志

当模拟比较器的输出事件符合中断触发条件时（中断触发条件由 ACIS1 和 ACIS0 定义），ACI 由硬件置“1”。若 ACIE 位置“1”，且 SREG 中的 I 位为“1”时，MCU 响应模拟比较器中断。当转入中断处理向量时，ACI 被硬件自动清空。此外，也可使用软件方式

清零 ACI：对 ACI 标志位写入逻辑“1”来清空该位。

- 位 3——ACIE：模拟比较器中断允许

当 ACIE 位设为“1”，且状态寄存器中的 I 位被设为“1”时，允许模拟比较器中断触发。当 ACIE 被清“0”时，模拟比较器中断被禁止。

- 位 2——ACIC：模拟比较器输入捕获允许

当设置为“1”时，定时/计数器 1 的输入捕获功能将允许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电路，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。要使能比较器触发定时器/计数器 1 的输入捕获中断，定时器中断屏蔽寄存器 TIMSK 的 TICIE1 位必须被设置。

- 位 1、0——ACIS1、ACIS0：模拟比较器中断模式选择

这些位决定哪种模拟比较器的输出事件可以触发模拟比较器的中断。不同的设置请参见表 2-57。

表 2-57　模拟比较器中断模式选择

ACIS1	ACIS0	中断模式
0	0	比较器输出的上升沿和下降沿都触发中断
0	1	保留
1	0	比较器输出的下降沿触发中断
1	1	比较器输出的上升沿触发中断

注意：当要改变 ACIS1、ACIS0 时，必须先清除 ACSR 寄存器中的中断允许位来禁止模拟比较器中断；否则，当这些位被改变时，会发生中断。

3．模拟比较器的多路输入

可以选择 ADC7..0 引脚中的任一路的模拟信号代替 AIN1 引脚，作为模拟比较器的反向输入端。模数转换的 ADC 多路复用器提供这种选择的能力，但此时，必须关闭芯片的 ADC 功能。当模拟比较器的多路选择使能位（SFIOR 中的 ACME 位）置“1”，同时 ADC 被关闭（ADCSRA 中的 ADEN 位为“0”）时，ADMUX 中的 MUX2..0 位的选择将代替 AIN1 输入，作为模拟比较器中的反向输入端的输入引脚，如表 2-58 所示。如果 ACME 被清零，或 ADEN 被置 1，则 AIN1 仍将为模拟比较器的反向输入端。

表 2-58　模拟比较器多路输入选择

ACME	ADEN	MUX2..0	模拟比较器反向输入端
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0
1	0	001	ADC1
1	0	010	ADC2

续表

ACME	ADEN	MUX2..0	模拟比较器反向输入端
1	0	011	ADC3
1	0	100	ADC4
1	0	101	ADC5
1	0	110	ADC6（TQFP 和 MLF 有效）
1	0	111	ADC7（TQFP 和 MLF 有效）

2.14 模数转换功能 ADC

2.14.1 特点

模数转换功能（ADC）有下列特点：

- 10 位精度（ADC4 和 ADC5 两个通道为 8 位精度）；
- 0.5LSB 的非线性度；
- ±2LSB 的绝对精度；
- 65μs~260μs 的转换时间；
- 每秒最大为 15kSPS 的采样速率；
- 6 路输入复用可选的单端输入通道；
- 附加 2 路输入的双路复用单端输入通道（仅 TQFP 和 MLF 封装形式有效）；
- ADC 的电压输入范围 0~Vcc；
- 可选的 2.56V 的 ADC 参考电压源；
- 连续转换模式和单次转换模式；
- ADC 转换完成触发中断；
- 休眠模式下的噪声抑制器（NOISE CANCELER）。

ATmega8 有一个 10 位的逐次比较（successive approximation）的 ADC。ADC 与一个 8 通道的模拟多路复用器连接，能够对以 PORTC 口作为 ADC 输入引脚的 8 路单端电压输入进行采样。单端电压输入以 0V（GND）为参考。注意，ADC4 和 ADC5 两个通道只提供 8 位的转换精度，其他通道提供 10 位转换精度。

ADC 包括采样保持电路，以确保输入电压在 ADC 转换过程中保持恒定。ADC 的方框图如图 2.71 所示。

ADC 功能单元由独立的专用模拟电源引脚 AVcc 供电。AVcc 和 Vcc 的电压差别不能大于±0.3V。关于如何使用该引脚，请参见“ADC 噪声消除器”部分。

ADC 转换的参考电源可采用芯片内部的 2.56V 参考电源，或采用 AVcc，也可采用外部的参考电源。使用外部参考电源时，外部参考电源由引脚 ARFE 接入。使用内部电压参

考源时，可以通过在 AREF 引脚外部并接一个电容来提高 ADC 的抗噪性能。

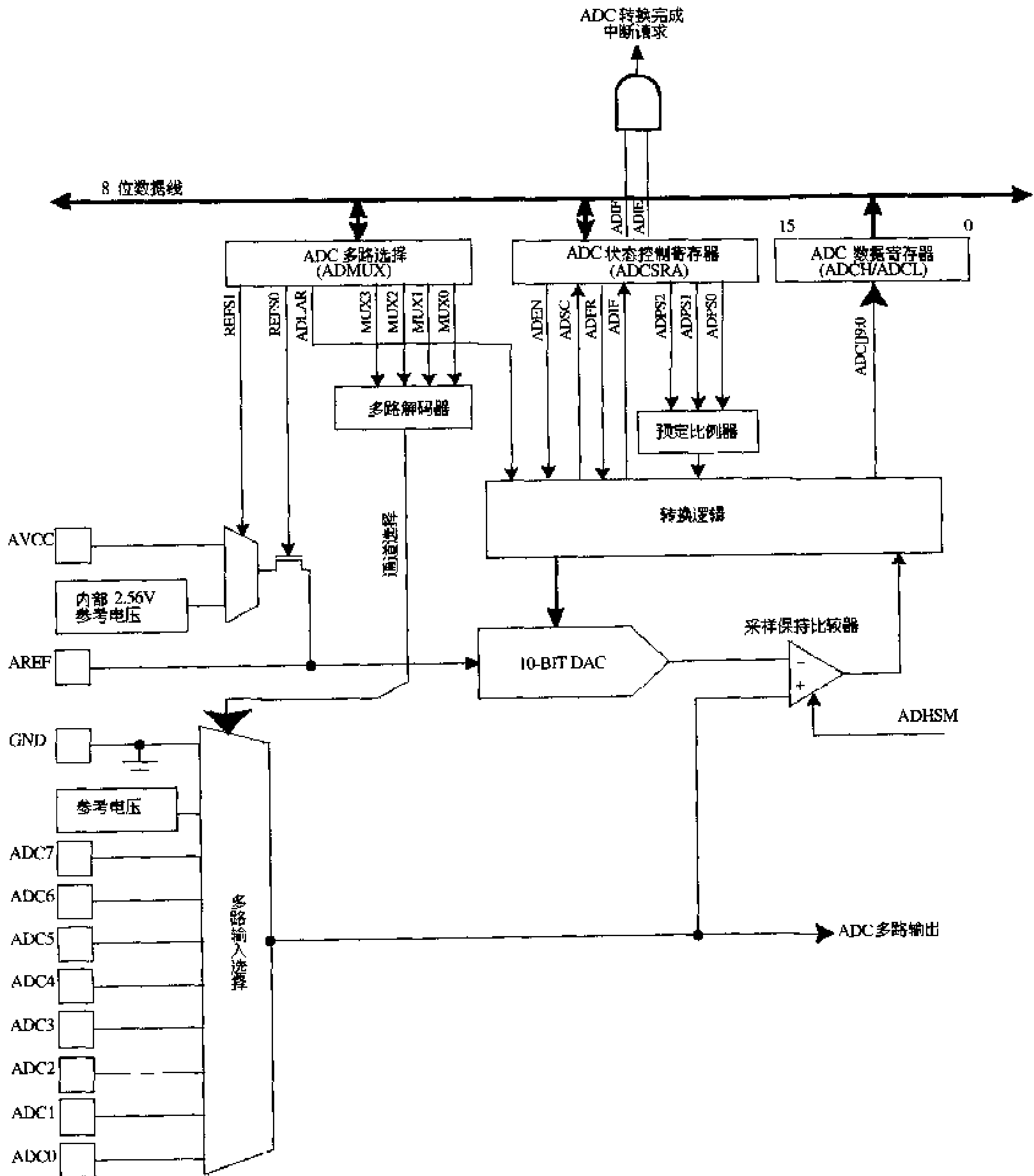


图 2.71　ADC 的方框图

ADC 通过逐次比较（successive approximation）方式，将输入端的模拟电压转换成 10 位的数字量。最小值代表地，最大值为 AREF 引脚上的电压值减 1 个 LSB。可以通过 ADMUX 寄存器中 REFSn 位的设置，选择将芯片内部参考电源（2.56V）或 AVcc 连接到 AREF，作为 A/D 转换的参考源。这时，内部电压参考源可以通过外接于 AREF 引脚的电容来稳定，以改进抗噪特性。

模拟输入通道的选择是通过 ADMUX 寄存器中的 MUX 位设定的。任何一个 ADC 的输入引脚，包括地（GND）以及内部的恒定间隙（fixed banddap）电压参考源，都可以被选择用来作为 ADC 的单端输入信号。通过设置 ADCSRA 寄存器中的 ADC 使能位 ADEN 来使能 ADC 功能 当 ADEN 位被清“0”后，ADC 不消耗能量，因此建议在进入节电休眠模式前将 ADC 关掉。

ADC 将 10 位的转换结果放在 ADC 数据寄存器中（ADCH 和 ADCL）。默认情况下，转换结果为右端对齐(RIGHT ADJUSTED)的。但可以通过设置 ADMUX 寄存器中 ADLAR 位，调整为左端对齐（LEFT ADJUSTED）。

如果转换结果是左端对齐，并且只需要 8 位的精度，那么只需读取 ADCH 寄存器的数据作为转换结果就达到要求了。否则，必须先读取 ADCL 寄存器，然后再读取 ADCH 寄存器，以保证数据寄存器中的内容是同一次转换的结果。因为一旦 ADCL 寄存器被读取，就中断了 ADC 对 ADC 数据寄存器的操作。这就意味着，一旦指令读取了 ADCL，那么必须紧接着读取一次 ADCH；如果在读取 ADCL 和读取 ADCH 的过程中正好有一次 ADC 转换完成，ADC 的 2 个数据寄存器的内容是不会被更新的，该次转换的结果将丢失。只有当 ADCH 寄存器被读取后，ADC 才可以继续对 ADCL 和 ADCH 寄存器操作更新。

ADC 有自己的中断，当转换完成时中断将被触发。尽管在顺序读取 ADCL 和 ADCH 寄存器过程中，ADC 对 ADC 数据寄存器的更新被禁止，转换的结果丢失，但仍会触发 ADC 中断。

2.14.2 启动 ADC 转换

将逻辑“1”写入 ADSC 位（ADC 转换开始）将启动一次 A/D 转换。在转换过程中，该位保持为“1”，直到 A/D 转换结束后由硬件自动清零。如果在 A/D 转换过程中改变 ADC 输入通道的选择，ADC 将在完成本次转换后再进行通道的转换。

通过置位 ADCSRA 寄存器的 ADFR 位，ADC 能被设置为连续转换模式。在连续转换模式下，ADC 将连续对输入进行采样和更新 ADC 数据寄存器。在连续转换模式下，也必须通过写入逻辑“1”到 ADCSRA 寄存器中的 ADSC 位来启动第一次的 A/D 转换；然后，ADC 将一直连续地进行逐次比较转换，无论 ADC 中断标志位 ADIF 是否清零与还是置位。

2.14.3 预分频与转换定时

图 2.72 所示为 ADC 的预分频电路。

在默认情况下，ADC 的逐次比较（SUCCESSIVE APPROXIMATION）转换电路需要一个 50kHz~200kHz 之间采样时钟。在要求转换精度低于 10 位的情况下，ADC 的采样时钟可以高于 200kHz，以获得更高的采样率。另外，可以设置 SFIOR 寄存器中的 ADHSM

位，来提高ADC的时钟频率，但这需要较高的功耗。

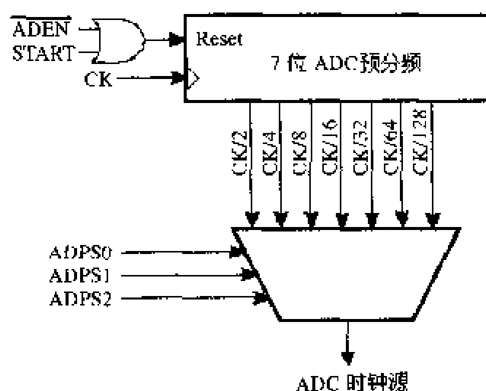


图2.72　ADC的预分频电路

ADC模块中包含一个预分频器，它对输入的系统时钟CK（>100KHz）进行分频，以获得合适的ADC时钟。预分频率是由ADCSRA寄存器中的ADPS位设置的。一旦寄存器ADCSRA中的ADEN位置"1"，预分频器就启动开始计数。ADEN位为"1"时，预分频器将一直工作；ADEN位为"0"时，预分频器一直处在复位状态。

当ADCSRA寄存器中的ADSC位置位，启动一次单次转换时，ADC转换将在随后的ADC时钟的上升沿开始。一次常规的A/D转换需要13个ADC时钟周期。而通过置位ADCSRA寄存器的ADEN位，使ADC启动进行的第一次A/D转换，因为要初始化模拟电路，所以需要25个ADC采样时钟周期。

在一次常规的A/D转换结束后，需要1.5个ADC时钟周期的采样保持时间。而对于ADC由禁止状态启动后的首次A/D转换，则需要13.5个ADC时钟周期的采样保持时间。当一次A/D转换完成后，转换结果写入ADC数据寄存器，ADIF（ADC中断标志位）将被置位。在单次转换模式下，ADSC也同时被清零。用户程序可以再次置位ADSC位，新的一次转换将在下一个ADC时钟的上升沿开始。

在连续转换模式下，一次转换完毕后马上开始一次新的转换，此时，ADSC位一直保持为"1"。表2-59所示为ADC的转换和采样保持时间。

表2-59　ADC转换和采样保持时间

转换形式	采样保持时间	转换时间
启动ADC，第一次转换	13.5个ADC时钟	25个ADC时钟
常规单次转换	1.5个ADC时钟	13个ADC时钟

图2.73、图2.74、图2.75所示分别为单次ADC转换方式（首次启动ADC、常规ADC）和连续ADC转换的时序。

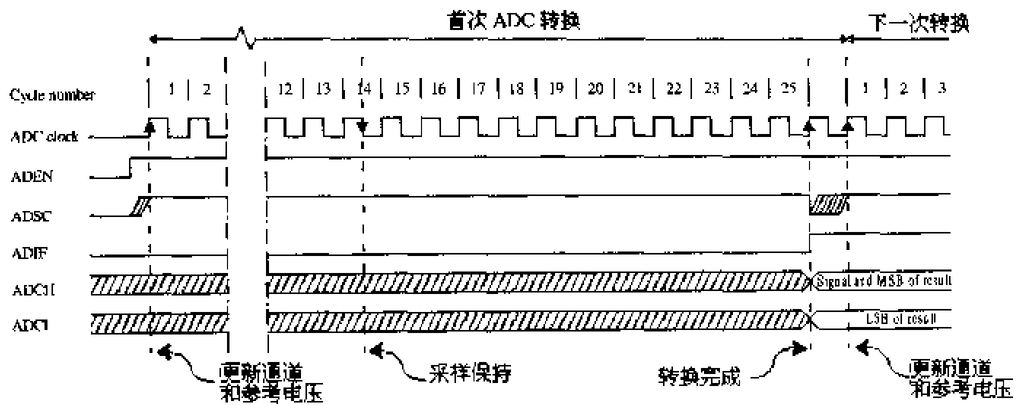


图 2.73　单次 ADC 转换时序（首次启动 ADC）

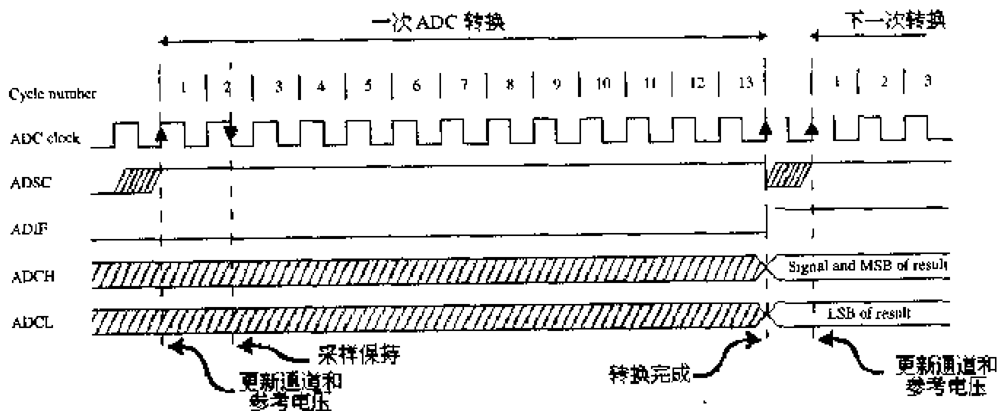


图 2.74　单次 ADC 转换时序（常规 ADC）

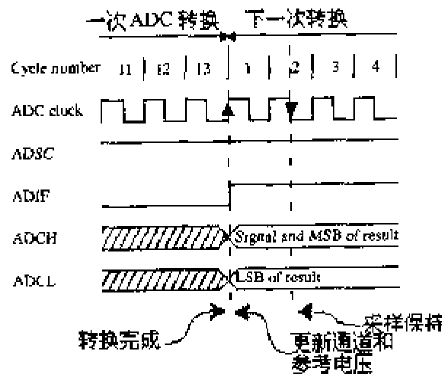


图 2.75　连续 ADC 转换时序

2.14.4　ADC 输入通道和参考电源的选择

寄存器 ADMUX 中的 MUXn 和 REF1、REF0 位是与一个 MCU 可以随机读取的临时寄存器相连通的缓冲器，这种结构保证了 ADC 输入通道和参考电源只能在 ADC 转换过程中的安全点被改变。在转换开始前，通道和参考电源不断被更新。一旦转换开始，通道和参考电源将被锁定，并保持足够时间，以确保 ADC 转换的正常进行。在转换完成前的最后一

个 ADC 时钟周期，通道和参考电源又开始重新更新。注意，由于 A/D 转换开始于置位 ADSC 后的第一个 ADC 时钟的上升沿，因此，在置位 ADSC 后的一个 ADC 时钟周期内不要将一个新的通道或参考电源写入到 ADMUX 寄存器中。

如果 ADFR 和 ADEN 位同时为“1”时，随时将会产生一个 ADC 中断。此时如果改变 ADMUX 寄存器，则无法知道下一次转换将基于新的或旧的设置。可靠更新 ADMUX 寄存器中的设置应在以下几种情况：

- 当 ADFR 或 ADEN 位为零时。
- 如在转换期间改变，至少要在开始转换后的一个 ADC 时钟周期以后。
- 在转换完成后，在被用作中断触发的标志位被清零前。

在以上的一种情况下更新 ADMUX 寄存器的设置后，新的设置将在下一次 A/D 转换时生效。

1．ADC 输入通道

当要改变 ADC 输入通道时，应该遵守以下方式，以保证能够选择到正确的通道：

在单次转换模式下，总是在开始转换前改变通道设置，在 ADSC 位被写入“1”后的 1 个 ADC 时钟周期内，输入通道改变为所设置的通道。然而，最简单的方法是等到转换完成后，再改变通道选择。

在连续转换模式下，总是在启动 ADC 开始第一次转换前改变通道设置。在 ADSC 位被写入“1”后的 1 个 ADC 时钟周期内，输入通道改变为所设置的通道。然而，最简单的方法是等到第一次转换完成后再改变通道的设置。此时，由于新一次的转换已经自动开始，所以，当前这次的转换结果仍为以前通道的反映，而下一次的转换结果将为新设置通道的反映。

2．ADC 电压参考源

ADC 的参考电压（V_{REF}）决定了 A/D 转换的范围。如果单端通道的输入电压超过 V_{REF}，将导致转换结果接近于 0x3FF。ADC 的参考电压 V_{REF} 可以选择为 AV_{CC} 或芯片内部的 2.56V 参考源，或者为外接在 AREF 引脚上的参考电压源。

AV_{CC} 通过一个无源开关连接到 ADC。内部 2.56V 参考源是由内部间隙参考源（V_{BC}）通过内部的放大器产生的。无论选用什么参考源，外部 AREF 引脚都是直接与 ADC 相连的，因此，可以通过外部在 AREF 引脚和地之间并接一个电容，使各种参考电源更加稳定和抗噪。可以通过用高阻电压表测量 AREF 引脚，来获得参考电源 V_{REF} 的电压值。由于 V_{REF} 是一个高阻源，因此，只有容性负载可以连接到该引脚。

如果将一个固定的电压源连接到 AREF 引脚，那么，就不能使用任何的内部参考电源，否则就会使外部电压源短路。如果没有外部电压源施加到 AREF 引脚，则可以选择使用 AV_{CC} 或内部的 2.56V 参考源作为 ADC 的参考源。参考源改变后的第一次 ADC 转换结果可能不太准确，建议抛弃该次转换结果。

2.14.5 ADC 噪声抑制器（NOISE CANCELER）

ADC 有一个噪声抑制器。在休眠模式下进行 A/D 转换时，应用该特性可以降低由 MCU 内核和 I/O 外围设备引入的噪声。噪声抑制器能够在 ADC 降噪（ADC Noise Reduction）和空闲（idle）模式下使用。实现过程如下：

（1）保证 ADC 处于使能，且系统不需要进行繁忙的 A/D 转换。此时，必须选择 ADC 为单次转换模式，且必须允许响应 ADC 转换完成中断（ADEN=1、ADSC=0、ADFR=0、ADIE=1、I=1）。

（2）进入 ADC 降噪模式（或空闲模式），一旦 MCU 运行挂起暂停，ADC 就开始转换。

（3）如果没有其他的中断在 ADC 转换结束前发生，ADC 转换完成后的中断将唤醒 MCU，并执行 ADC 中断服务程序。如果其他中断在 ADC 转换完成前把 MCU 唤醒，该中断服务程序将被执行；同时，ADC 的转换将继续进行，直到 ADC 转换完成后产生 ADC 中断请求。

（4）一旦 MCU 被唤醒后，将保持活动状态，直到下一条休眠指令被执行。

需要注意的是，MCU 进入其他休眠模式后，ADC 也不会自动关闭。因此，建议在进入这些休眠模式前，将 ADEN 位清零，以避免更多的功率消耗。

1．模拟输入电路

单端 ADC 的模拟输入电路如图 2.76 所示。无论 ADCn 引脚是否被选择为 ADC 的输入，加至该引脚的模拟信号源都应该能克服由于引脚的电容和漏电流所造成的影响。当通道被选择为 ADC 的输入时，模拟信号源必须能够通过引脚内部的串联电阻驱动 S/H 电容（采样保持电容）。

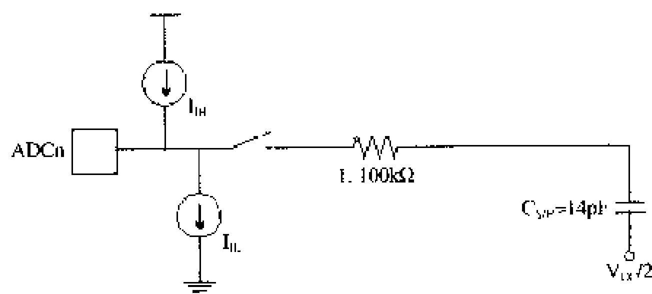


图 2.76 单端 ADC 的模拟输入等效电路

对于输出阻抗为 10kΩ 或更小的信号源的模拟输入信号，ADC 在内部进行了优化。如果采用这样的模拟信号源，可以忽略采样的时间。如果使用高阻抗的模拟信号源，则采样时间取决于信号源对采样保持电容的充电时间，其值会在很大的范围内变化。建议采用低阻抗和缓变型的模拟信号源，因为此时的 S/H 电容（采样保持电容）充电时间很小。

如果信号中有高于奈奎斯特采样频率（f_{ADC}）的成分时，为避免不可预测的信号混叠，建议使用低通滤波器，将高频成分在加到 ADCn 引脚前滤除。

2．模拟噪声的抑制

器件外部和内部的数字电路会产生电磁干扰，并会影响模拟测量的精度。如果 A/D 转换精度要求很高，可以采用以下的技术来降低噪声的影响：

（1）使模拟信号的通路尽可能的短。模拟信号连线应从模拟地的布线盘上通过，并使它们尽可能远离高速开关数字信号线。

（2）器件上的 AVcc 引脚应该通过 LC 网络（如图 2.77 所示）与数字端电源 Vcc 相连。

（3）采用 ADC 噪声抑制器功能来降低来自 MCU 内部的噪声。

（4）如果 ADC[3..0]作为通用数字输出口使用，那么在 ADC 转换过程中，千万不要改变这些引脚的状态。然而，在使用 TWI 总线接口时（ADC4 和 ADC5），只会影响 ADC4 和 ADC5 的转换，而不会影响 ADC 的其他通道。

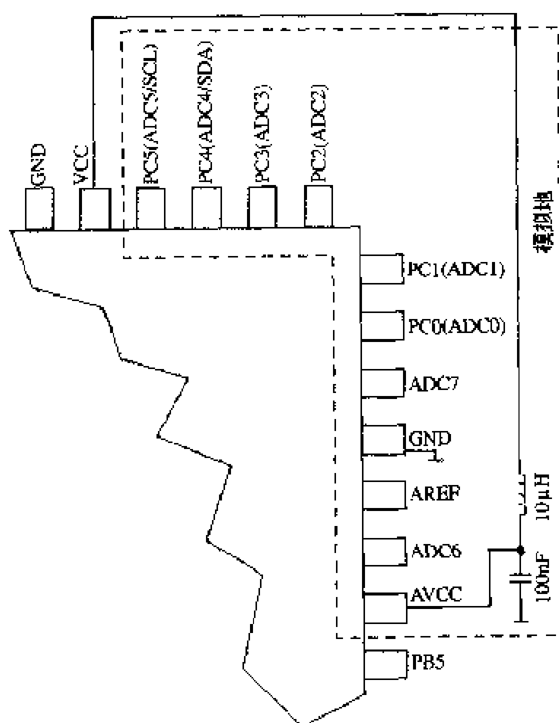


图 2.77　AVcc 引脚的连接

3．ADC 转换结果

A/D 转换结束后（ADIF = 1），在 ADC 数据寄存器（ADCL 和 ADCH）中可以取得转换的结果。对于单端输入的 A/D 转换，其转换结果为：

$$ADC = (V_{IN} \times 1024) / V_{REF}$$

其中 V_{IN} 表示选定的输入引脚上的电压，V_{REF} 表示选定的参考电源的电压。0x000 表示输入引脚的电压为模拟地，0x3FF 表示输入引脚的电压为参考电压值减去一个 LSB。

2.14.6 有关的 I/O 寄存器

1．ADC 多路复用器选择寄存器——ADMUX

位	7	6	5	4	3	2	1	0	
$07（$0027）	REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0	ADMUX
读/写	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

● 位 7，6——REFS1，REFS0：ADC 参考电源选择

这些位用于选择 ADC 的参考电压源，见表 2-60。如果这些位在 ADC 转换过程中被改变，新的选择将在该次 ADC 转换完成后（ADCSRA 中的 ADIF 被置位）才生效。一旦选择内部参考源（AVcc、2.56V）为 ADC 的参考电压后，AREF 引脚上不得施加外部的参考电源，只能与 GND 之间并接抗干扰电容。

表 2-60 ADC 参考电源选择

REFS1	REFS0	ADC 参考电源
0	0	外部引脚 AREF，断开内部参考源连接
0	1	AVcc，AREF 外部并接电容
1	0	保留
1	1	内部 2.56V，AREF 外部并接电容

● 位 5——ADLAR：ADC 结果左对齐选择

ADLAR 位决定转换结果在 ADC 数据寄存器中的存放形式。写“1”到 ADLAR 位，将使转换结果左对齐（LEFT ADJUST）；否则，转换结果为右对齐（RIGHT ADJUST）。无论 ADC 是否正在进行转换，改变 ADLAR 位都将会立即影响 ADC 数据寄存器。

● 位 3..0——MUX3:0：模拟通道选择

这些位用于对连接到 ADC 的输入通道进行选择设置。详见表 2-61。

表 2-61 ADC 参考电源选择

MUX3..0	单端输入口
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7

续表

MUX3..0	单端输入口
1000	
1001	
1010	
1011	
1100	
1101	
1110	1.23V（VBG）
1111	0V（GND）

2．ADC 控制和状态寄存器 A——ADCSRA

位	7	6	5	4	3	2	1	0	
$06（$0026）	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

- 位 7——ADEN：ADC 使能

该位写入“1”使能 ADC，写入“0”关闭 ADC。如在 ADC 转换过程中将 ADC 关闭，该次转换随即停止。

- 位 6——ADSC：ADC 开始转换

在单次转换模式下，置该位为“1”，将启动一次转换。在连续转换模式下，该位写入“1”将启动第一次转换。先置位 ADEN 位使能 ADC，再置位 ADSC；或置位 ADSC 的同时使能 ADC，即 ADC 使能后的第一次转换将需要 25 个 ADC 时钟周期，而不是常规转换 13 个 ADC 时钟周期，这是因为需要完成对 ADC 的初始化。

在转换进行过程中，ADSC 将始终读出为“1” 当转换完成时，它将转变为“0”。强制写入“0”是无效的。

- 位 5——ADFR：ADC 连续转换模式选择

当该位被置为“1”时，ADC 工作在连续转换模式下。在该模式下，ADC 不断地采样和更新 ADC 数据寄存器。清零该位将中止连续转换模式。

- 位 4——ADIF：ADC 中断标志位

当 ADC 转换完成并且 ADC 数据寄存器被更新后该位被置位。如果 ADIE 位（ADC 转换结束中断允许位）和 SREG 寄存器中的 I 位被置“1”，ADC 中断服务程序将被执行。ADIF 在执行相应的中断处理向量时被硬件自动清零。此外，ADIF 位可以通过写入逻辑“1”来清零。

注意，如果对 ADCSRA 寄存器执行读—改写操作时，中断请求可以被屏蔽。使用 SBI 和 CBI 指令也同样有效。

- 位 3——ADIE：ADC 中断允许

当该位和 SREG 寄存器中的 I 位同时被置位时，允许 ADC 完成转换中断。

● 位 2..0——ADPS2:0：ADC 预分频选择

这些位决定了 XTAL 时钟与输入到 ADC 的 ADC 时钟之间分频数，见表 2-62。

表 2-62 ADC 时钟分频

ADPS2..0	分 频 率
000	2
001	2
010	4
011	8
100	16
101	32
110	64
111	128

3．ADC 数据寄存器——ADCL 和 ADCH

位	15	14	13	12	11	10	9	8	
$05（$0025）	-	-	-	-	-	-	ADC9	ADC8	ADCH
$04（$0024）	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
位	7	6	5	4	3	2	1	0	
读/写	R	R	R	R	R	R	R	R	
读/写	R	R	R	R	R	R	R	R	
复位值	0	0	0	0	0	0	0	0	
复位值	0	0	0	0	0	0	0	0	

● ADLAR = 0，右对齐

● ADLAR = 1，左对齐

当 ADC 转换完成时，可以在这两个寄存器读取转换的结果。当 ADCL 寄存器被读以后，ADC 数据寄存器就不能被 ADC 更新，直到 ADCH 寄存器被读取为止。因此，如果结果是左对齐（ADLAR=1），且不需要大于 8 位的精度的话，仅仅读取 ADCH 寄存器就足够了。否则，必须先读取 ADCL 寄存器，再读取 ADCH 寄存器。ADMUX 寄存器中的 ADLAR 位决定了从 ADC 数据寄存器中读取结果的格式。如果 ADLAR 位为“1”，结果将是左对齐；如果 ADLAR 位为“0”（默认情况），结果将是右对齐。

4．特殊功能 IO 寄存器——SFIOR

位	7	6	5	4	3	2	1	0	
$30（$0050）	-	-	-	ADHSM	ACME	PUD	PSR2	PSR10	SFIOR
读/写	R	R	R	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

位	15	14	13	12	11	10	9	8	
$05 ($0025)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
$04 ($0024)	ADC1	ADC0	-	-	-	-	-	-	ADCL
位	7	6	5	4	3	2	1	0	
读/写	R	R	R	R	R	R	R	R	
读/写	R	R	R	R	R	R	R	R	
复位值	0	0	0	0	0	0	0	0	
复位值	0	0	0	0	0	0	0	0	

● 位 4——ADHSM：ADC 高速模式

该位置“1”，将使 ADC 处在高速模式。该模式将以高功耗为代价，使 ADC 能够以更高的采样速率进行转 A/D 转换。

2.15　引导加载支持的自编程功能

ATmega8 具备引导加载支持的用户程序自编程功能（In-System Programming by On-chip Boot Program），它提供了一个真正的由 MCU 本身自动下载和更新（采用读/写同时“Read-While-Write”进行的方式）程序代码的系统程序自编程更新的机制。使用该功能时，MCU 可以灵活地运行一个常驻 Flash 的引导加载程序（Boot Loader Program），实现对用户应用程序的在线自编程更新。引导加载程序能使用任何可用的数据接口和相关的协议读取代码，或者从程序存储器中读取代码，然后将代码写入（编程）到 Flash 存储器中。引导加载程序有能力读写整个 Flash 存储器，包括引导加载程序所在的引导加载区本身。引导加载程序还可以对自身进行更新修改，甚至可以将自身删除，使系统的自编程能力消失。引导加载程序区的大小可以由芯片的熔丝位设置，该段程序区还提供两组锁定位，以便用户选择对该段程序区不同级别的保护。

引导加载支持的自编程主要特点为：

- 读/写同时进行的自编程；
- 可灵活设置引导加载程序区的大小；
- 高度安全性；
- 熔丝位设定选择复位向量；
- 优化的页大小；
- 高效的代码算法；
- 高效的读改写（Read-Modify-Write）支持。

2.15.1 引导加载技术的实现

1. 引导加载程序区和应用程序区

ATmega8 的 Flash 程序存储器空间分为两个部分：引导加载程序区和应用程序区（见图 2.78），两个区的大小由 BOOTSZ1 和 BOOTSZ2 熔丝位确定（见表 2-62）。两个区都由各自独立的锁定位控制，因此可以采用不同级别的保护。

表 2-62 Flash 程序存储区分配

BOOTSZ1	BOOTSZ0	应用程序区	加载程序区	加载区大小（字节）	占页数 32 字/页
1	1	0x000-0xF7F	0x800-0xFFF	128	4
1	0	0x000-0xEFF	0xF00-0xFFF	256	8
0	1	0x000-0xDFF	0xE00-0xFFF	512	16
0	0	0x000-0xBFF	0xC00-0xFFF	1024	32

- 应用程序区

在 Flash 程序存储器空间中，应用程序区是用来驻留应用程序代码的。应用程序区的保护级别由应用程序区锁定位（Boot Lock bits 0）设定（见表 2-58）。如果驻留在应用程序区中的应用程序内含有 SPM 指令，则该应用程序执行时 SMP 指令将被屏蔽，因此，引导加载程序不能放置在应用程序区中。

- 加载程序区

引导加载程序必须驻留在引导加载区中。这是因为，只有 MCU 执行在引导加载区中的引导加载程序时，其中的 SPM 指令才可以对 Flash 程序存储器进行初始化编程。此时，SPM 指令可以读写包括引导加载区在内的整个 Flash 程序存储器。引导加载区的保护级别由引导加载锁定位（Boot Lock bits 1）决定（见表 2-59）。

2. 可同时读写和非同时读写区

除了前面所述的 Flash 存储器可根据 BOOTSZ 熔丝位的设置分为应用程序区和引导加载区两部分外，Flash 存储器还被固定地分为两部分：可同时读写区 RWW（Read-While-Write）和非同时读写区 NRWW（No Read-While-Write），见图 2.79 所示。RWW 区和 NRWW 区主要区别有以下两点：

（1）当对一个位于 RWW 区的页进行擦除或写入操作时，可同时对 NRWW 进行读操作。

（2）当对一个位于 NRWW 区的页进行擦除或写入操作时，在操作的整个过程中 MCU 处于暂停挂起状态。

因此，MCU 对程序存储器进行操作时，由于区域的不同，决定了 MCU 是否支持同时读写方式。

特别需要指出的是，在进行自引导加载更新程序的过程中，程序永远不能读取位于 RWW 区中的代码。所谓可同时读写区（Read-While-Write section）的概念，是指正在被编

程（擦除和写入）的区域，而不是指自引导加载更新程序当前正在读取的代码所在区域。

- RWW——可同时读写区

在引导加载程序对 RWW 区中的一个页进行更新编程的过程中，MCU 只能读取驻留在 NRWW 区中的代码。在连续的编程过程中，必须保证程序不读取 RWW 区。如果程序试图读取位于 RWW 区中的代码（如使用 CALL、JMP、LPM 或中断），程序有可能终止于未知状态。为避免这种情况的发生，中断应被屏蔽，或将其（包括中断向量以及中断服务程序）移到引导加载区。引导加载区总是位于 NRWW 区中。当读取 RWW 区的操作被阻断时，程序存储器控制寄存器 SPMCR 中的标志位 RWWSB（RWW 区忙标志）将保持为“1”。编程结束后，在读取 RWW 中的代码前，必须在程序中先将标志位 RWWSB 清零。

- NRWW——非同时读写区

在引导加载程序对 RWW 区中的一个页进行更新编程的过程中，MCU 能读取驻留在 NRWW 区中的代码。而在引导加载程序对 NRWW 区中的一个页进行更新编程的过程中，MCU 将一直保持挂起暂停的状态。

表 2-64、表 2-65、图 2.78 和图 2.79 给出了 RWW 区和 NRWW 区的分布以及特性。

表 2-64　RWW 和 NRWW 空间

区　域	地 址 空 间	占 页 数
RWW	0x000-0xBFF	96
NRWW	0xC00-0xFFF	32

表 2-65　Read-While-Write 特性

在编程过程中 Z 寄存器指向的区域	在编程过程中 可读取的区域	MCU 是否 暂停	是否支持同时 读写方式
RWW	NRWW	No	Yes
NRWW	None	Yes	No

3. 引导加载锁定位

如果系统不需要自引导加载编程能力，整个 Flash 存储器都可用于放置应用程序代码。引导加载系统有两组可独立设置的引导锁定位（Boot Lock bits），提供了可灵活选择的不同级别的保护措施，能够：

- 锁定整个 Flash 存储器，使其免于被 MCU 软件自编程更新。
- 只锁定引导加载区，使其免于被 MCU 软件自编程更新。
- 只锁定应用程序区，使其免于被 MCU 软件自编程更新。
- 无锁定，允许整个 Flash 存储器可被 MCU 软件自编程更新。

参见表 2-66 和表 2-67 以了解详细情况。可以在运行程序中使用相应的指令置位引导锁定位，或通过串行或并行编程的方式引导锁定位置位。但引导锁定位只能通过芯片擦除命令将其清零。芯片的写保护（Lock Bit mode 2）并不能防止使用 SPM 指令对 Flash 存储器的编程。同样，芯片的读/写保护（Lock Bit mode 3）也不能防止使用 LPM 或 SPM 指令

对 Flash 存储器的读或写操作。

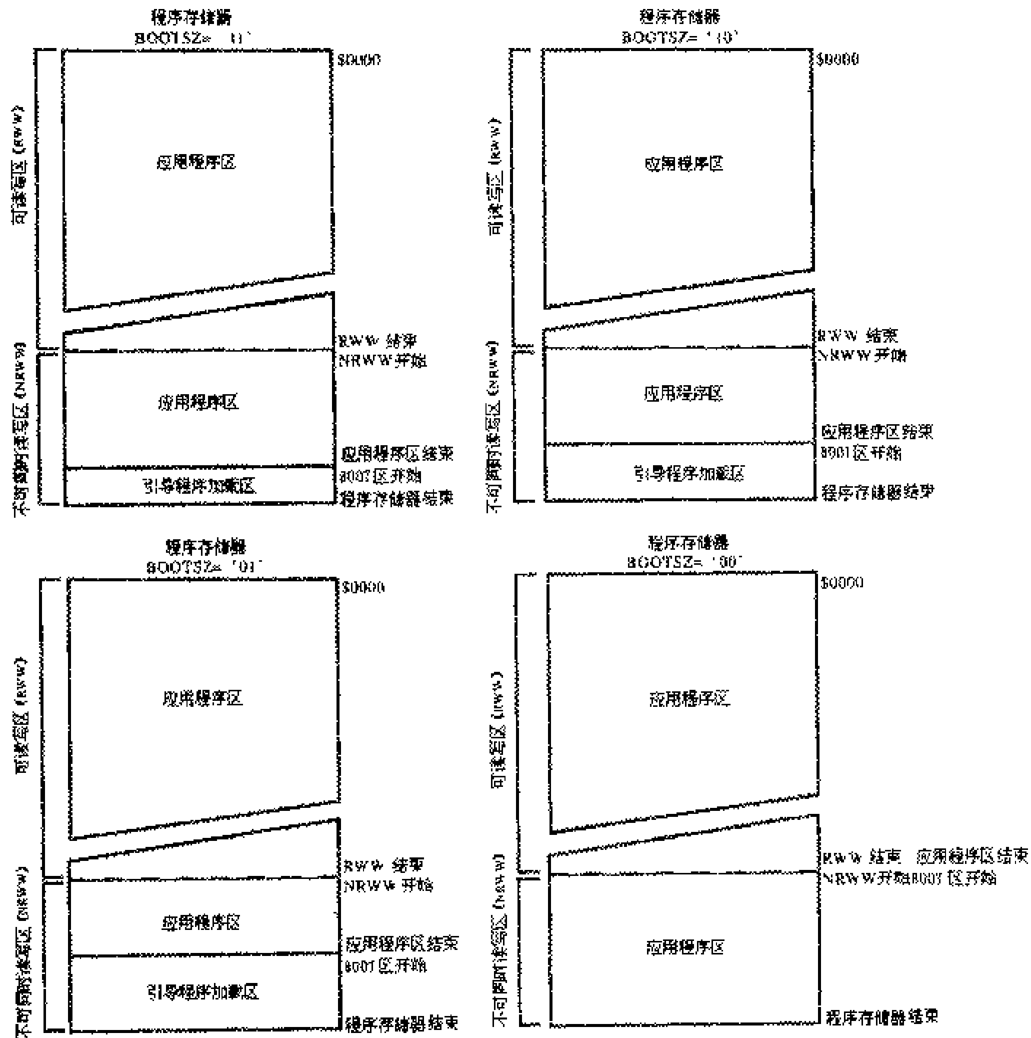


图 2.78 ATmega8 的程序存储器空间的 4 种分配图

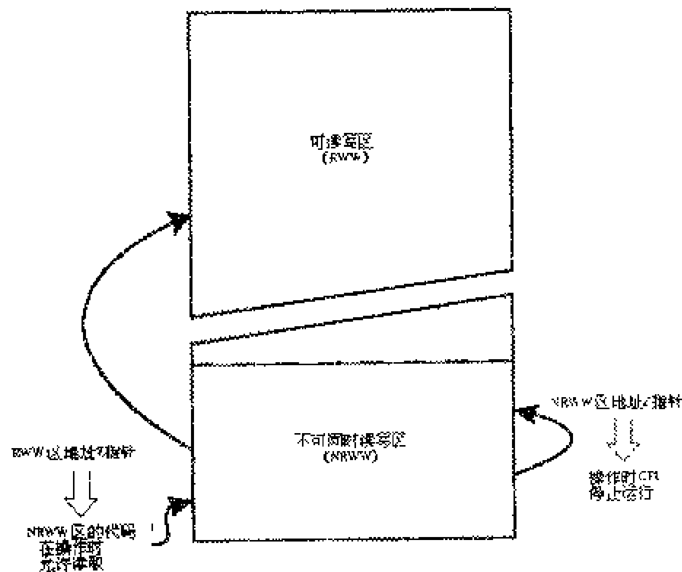


图 2.79 ATmega8 程序存储器的划分和性质

表 2-66　Boot Lock Bit0 的保护模式

BLB0 模式	BLB02	BLB01	对应用程序区的保护
Mode1	1	1	不限制 SPM 和 LPM 指令对应用程序区的操作
Mode2	1	0	禁止 SPM 指令对应用程序区的写操作
Mode3	0	0	禁止 SPM 指令对应用程序区的写操作 在执行驻留于引导加载区的引导加载程序过程中，禁止其中 LPM 指令对应用程序区的读操作 如果中断向量驻留在引导加载区，则在 MCU 执行驻留于应用程序区的程序过程中，禁止中断响应
Mode4	0	1	在执行驻留于引导加载区的引导加载程序过程中，禁止其中 LPM 指令对应用程序区的读操作 如果中断向量驻留在引导加载区，则在 MCU 执行驻留于应用程序区的程序过程中，禁止中断响应

注："1" 表示未编程，"0" 表示被编程

表 2-67　Boot Lock Bit1 的保护模式

BLB1 模式	BLB12	BLB11	对引导加载区的保护
Mode1	1	1	不限制 SPM 和 LPM 指令对引导加载区的操作
Mode2	1	0	禁止 SPM 指令对引导加载区的写操作
Mode3	0	0	禁止 SPM 指令对引导加载区的写操作 在执行驻留于应用程序区的应用程序过程中，禁止其中 LPM 指令对引导加载区的读操作 如果中断向量驻留在应用程序区，则在 MCU 执行驻留于引导加载区的加载程序过程中，禁止中断响应
Mode4	0	1	在执行驻留于应用程序区的应用程序过程中，禁止其中 LPM 指令对引导加载区的读操作 如果中断向量驻留在应用程序区，则在 MCU 执行驻留于引导加载区的加载程序过程中，禁止中断响应

注："1" 表示未编程，"0" 表示被编程

4．引导加载程序的执行

可以在应用程序中使用 JUMP 或 CALL 指令，跳转执行驻留于引导加载区中的引导加载程序。如通过 USART 或 SPI 接口，接收一个命令来触发跳转。另一种方式是编程 Boot Reset 熔丝位，使复位向量指向引导加载区的起始地址（见表 2-68）。在这种情况下，引导加载程序将在系统复位后开始执行，当加载程序把应用程序代码载入并写入应用程序区后，再转入执行应用程序代码。由于 Boot Reset 熔丝位不能被 MCU 自己改变，所以一旦 Boot Reset 熔丝位被编程，系统的复位向量将始终指向引导加载区的起始地址。熔丝位的改变只

有通过串行或并行编程接口才能实现。

表 2-68 复位向量的设定

BOOTRST	复位向量地址
1	应用程序区起始地址 0x000
0	引导加载区起始地址（见表 2-55）

注："1" 表示未编程，"0" 表示被编程

2.15.2 相关 I/O 寄存器

1. 程序存储器存储控制寄存器——SPMCR

程序存储器存储控制寄存器包括引导加载操作所需的控制位。

位	7	6	5	4	3	2	1	0	
$37($0057)	SPMIE	RWWSB	-	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	SPMCR
读/写	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

- 位 7——SPMIE：SPM 中断允许

当 SPMIE 位被置"1"，同时状态寄存器中的 I 位被置"1"时，SPM 完成中断将被允许。只要 SPMCR 寄存器中的 SPMEN 位被清"0"，MCU 就会响应和执行 SPM 中断服务。

- 位 6——RWWSB：RWW 区忙标志

当开始对 RWW 区进行自编程（页擦除或页写入）操作时，RWWSB 位将被硬件自动置"1"。当 RWWSB 位被置"1"后，对 RWW 区的读操作被阻断。在自编程操作完成后，向 RWWSRE 位写入"1"，会将 RWWSB 位清"0"。此外，如果开始一个页读取操作时，也会将 RWWSB 位自动清"0"。

- 位 4——RWWSRE：读 RWW 区允许

当开始对 RWW 区进行自编程（页擦除或页写入）操作时，RWWSB 位将被硬件自动置"1"，对 RWW 区的读操作被阻断。如要再次开放对 RWW 区的读操作，要在自编程操作完成后（SPMEN = 0），同时将 RWWSRE 位和 SPMEN 位置为"1"，在其后的 4 个时钟周期内的 SPM 指令将使 RWW 区开放。自编程过程中（SPMEN = 1），不能开放对 RWW 区的读操作。如果在加载 Flash 期间对 RWWSRE 位进行写操作，Flash 的加载操作将被放弃，加载的数据将丢失。

- 位 3——BLBSET：引导锁定位（Boot Lock Bit）设置

如果该位与 SPMEN 位被同时置"1"，在其后 4 个时钟周期内的一个 SPM 指令根据寄存器 R0 中数据对引导锁定位进行设置，而寄存器 R1 和 Z 寄存器中的地址则被忽略。在引导锁定位设置操作完成后，或在其后 4 个时钟周期内没有执行 SPM 指令的话，BLBSET 位将自动被清零。

在 BLBSET 位和 SPMEN 位被置"1"后的 3 个时钟周期内，执行 LPM 指令可将锁定

位（Lock-bits）或熔丝位（Fuse）（由 Z 寄存器的 Z0 位确定）内容读入目的寄存器中。详见本节中“用程序读取熔丝位和锁定位”。

● 位 2——PGWRT：页写入

如果该位与 SPMEN 位被同时置“1”，在其后 4 个时钟周期内的一个 SPM 指令将执行页写入操作，写入页的地址从 Z 指针中的高位部分取出，数据取自临时缓冲器，而寄存器 R1 和 R0 中数据则被忽略。在页写入操作完成后，或在其后 4 个时钟周期内没有执行 SPM 指令的话，PGWRT 位将自动被清零。如果被写入的页位于 NRWW 区，MCU 将在整个页擦除期间处于暂停状态。

● 位 1——PGERS：页擦除

如果该位与SPMEN位被同时置“1”，在其后4个时钟周期内的一个SPM指令将执行页擦除操作，擦除页的地址从Z指针中的高位部分取出。在页擦除操作完成后，或在其后4个时钟周期内没有执行SPM指令的话，PGERS位将自动被清零。如果被擦除的页位于NRWW区，MCU将在整个页擦除期间处于暂停状态。

● 位 0——SPMEN：程序存储器写允许

当 SPMEN 位和 RWWSRE、BLBSET、PGWRT、PGERS 四个标志位的其中之一被同时设置为“1”时，那么在其后的 4 个时钟周期内，将允许 SPM 指令对程序存储器进行写操作。如果仅仅只是 SPMEN 位设置为“1”，接下来的 SPM 指令只是把 R1 和 R0 中的值写入到由 Z 指针寻址的临时缓冲页中。此时，地址指针寄存器 Z 的最低位被忽略。在 SPM 指令完成后，或者置“1”SPMEN 后的 4 个时钟周期内没有执行 SPM 指令的话，SPMEN 位将被自动清零。在页擦除和页写入的操作过程中，SPMEN 位一直保持为“1”，直到操作完成。

SPMEN 标志位应和 RWWSRE、BLBSET、PGWRT、PGERS 四个标志位的其中之一同时置位，因此，SPMCR 寄存器的低 5 位的设置值只有“10001”、“01001”、“00101”、“00011”、“00001”五种组合，其他的组合是无效的。

2. 自编程时的 Flash 存储器寻址

位	15	14	13	12	11	10	9	8	
$31($001F)	Z15	Z14	Z13	Z12	Z11	Z10	Z9	Z8	ZH
$30($001E)	Z7	Z6	Z5	Z4	Z3	Z2	Z1	Z0	ZL
位	7	6	5	4	3	2	1	0	

地址指针寄存器 Z 被用作 SPM 指令的间接寻址寄存器。

AVR 的 Flash 程序存储器是按页构成的，因此程序计数器（IP）被分作两段。一段为 IP 的低位部分，用于对页内的字寻址；而 IP 的高位部分用于对页的寻址。如图 2.80 所示。由于页擦除和页写入操作是独立寻址的，因此，引导加载程序的页擦除和页写入操作只能在同一页进行（即按页寻址操作）。一旦自编程操作开始，Z 寄存器中的地址被锁存，此后 Z 寄存器可以用于其他操作。

惟一不用 Z 寄存器寻址的 SPM 操作是对引导加载锁定位的设置。此时，Z 寄存器的内

容被忽略，对操作无影响。LPM 指令也使用 Z 寄存器存放地址指针，由于该指令对 Flash 的寻址是以字节为单位的，所以 Z 寄存器的最低位（Z0）也要用于寻址。

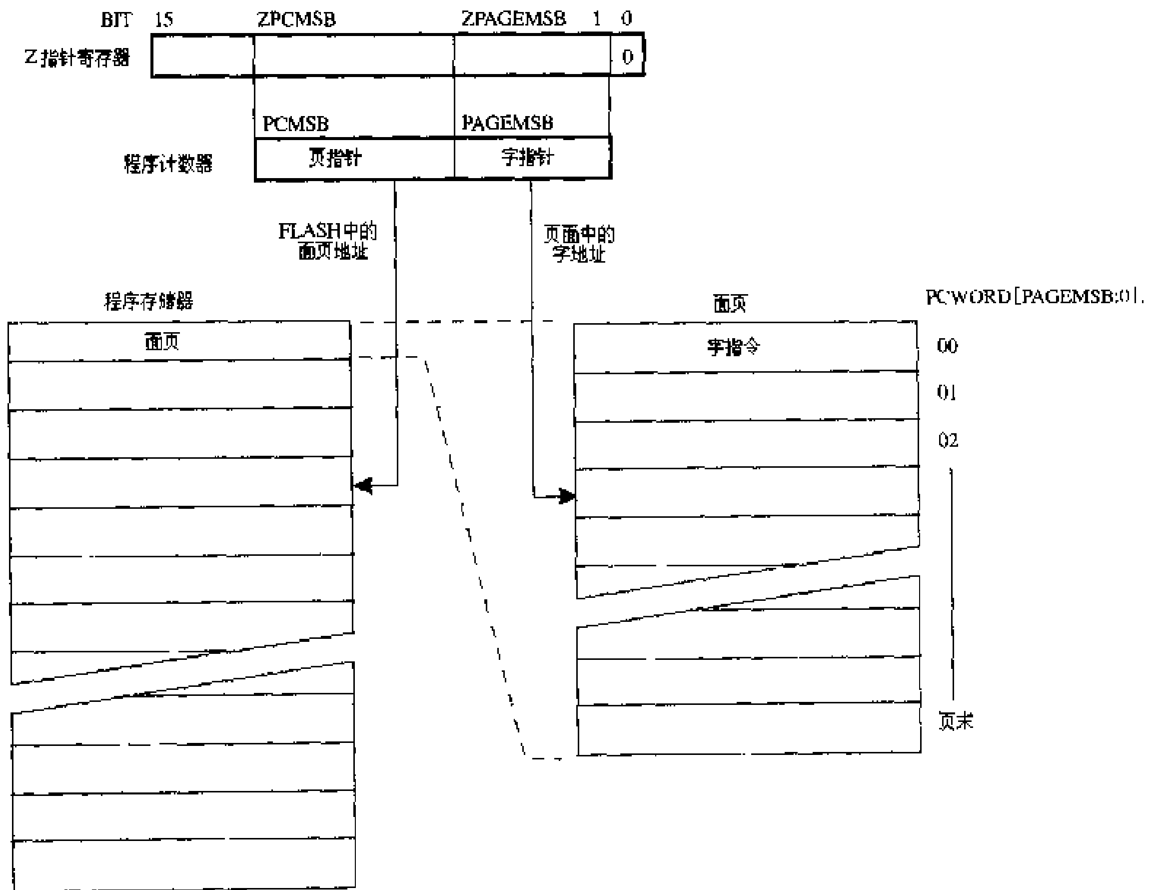


图 2.80 SPM 指令的 Flash 存储器寻址

图中符号代表的意义见表 2-69。

表 2-69 图 2.80 中各变量的含义和 Z 指针的结构定义

符 号	取 值	Z 寄存器的对应值	说 明
PCMSB	11		程序计数器最高位 （程序计数器为 12 位，PC[11:0]）
PAGEMSB	4		页内字寻址最高位 （每页 32 个字，需要 5 位，PC[4:0]）
ZPCMSB		Z12	在 Z 寄存器中对应 PCMSB 的位 （由于 Z0 不使用，ZPCMSB=PCMSB+1）
ZPAGEMSB		Z5	在 Z 寄存器中对应 PAGEMSB 的位 （由于 Z0 不使用，ZPAGEMSB=PAGEMSB+1）
PCPAGE	PC[11:5]	Z12:Z6	页寻址（用于页擦除和页写入）
PCWORD	PC[4:0]	Z5:Z1	字寻址（用于向临时缓冲页的写入） 页写入时，这 5 位必须为“0”

注：Z15:Z13 忽略；对所有的 SPM 指令 Z0 应为“0”；在 LPM 指令中 Z0 用于字节选择

2.15.3　程序存储器 Flash 的自编程

程序存储器的编程是按页操作更新的。在将临时缓冲页中的内容写入到某个页前，该页必须先被擦除。临时缓冲页每次只能用 SPM 指令填充一个字的内容。缓冲页的填充可在页的擦除前，或者在页擦除和页写入之间：

（1）在页擦除前填充缓冲页

- 填充临时缓冲页
- 完成页擦除
- 完成页写入

（2）在页擦除后填充缓冲页

- 完成页擦除
- 填充临时缓冲页
- 完成页写入

如果只有页内的一部分内容需要改变，那么页的其余部分必须在擦除前保存起来（如放在临时缓冲页中），然后再重新写入。当使用方法（1）时，引导加载可以使用有效的读-修改-写（Read-Modify-Write）的功能，即允许程序先读取页中原来的内容，然后做必要的修改，最后回写调整后的数据。如果使用方法（2），不可能在加载前读取原来页中的数据，因为页已经被擦除了。临时缓冲器页能够随机访问。特别重要的是，页擦除和页写入操作的寻址应在同一页。请参见“简单的引导加载汇编代码实例”部分。

1．通过 SPM 指令实施页擦除

要实现页擦除：应先在 Z 寄存器中设置要擦除页的地址（页地址必须写在 Z 寄存器中的 PCPAGE 段中）；然后将“x0000011”写入 SPMCR 寄存器；在设置 SPMCR 后的 4 个时钟周期内执行 SPM 指令。页擦除操作时，寄存器 R1 和 R0 中的数据被忽略，Z 寄存器的其他位对操作无影响。

（1）在对 RWW 区中的页擦除期间，可以读取 NRWW 区。

（2）在对 NRWW 区中的页擦除期间，MCU 挂起，处于暂停工作状态。

2．对临时缓冲页的填充

要向临时缓冲页写入一个指令字：应先将指令字放入寄存器 R1:R0 中；并在 Z 寄存器中设置该指令在缓冲页内的字地址（页内字寻址地址必须写在 Z 寄存器中的 PCWORD 段中）；然后将“00000001”写入 SPMCR 寄存器；在设置 SPMCR 后的 4 个时钟周期内执行 SPM 指令。在页写入操作后，或通过写 SPMCR 寄存器中的 RWWSRE 位以及系统复位后，临时缓冲页的内容都会被自动擦除。注意，临时缓冲页在未擦除前，不能对其中同一个地址填写两次内容。

3．实施页写入

要实现页写入：应先在 Z 寄存器中设置要写入页的地址（页地址必须写在 Z 寄存器中

的 PCPAGE 段中）；然后将"x0000101"写入 SPMCR 寄存器；在设置 SPMCR 后的 4 个时钟周期内执行 SPM 指令。页写入操作时，寄存器 R1 和 R0 中的数据被忽略，Z 寄存器的其他位对操作无影响。

（1）在对 RWW 区中的页写入期间，可以读取 NRWW 区。

（2）在对 NRWW 区中的页写入期间，MCU 挂起，处于暂停工作状态。

4．使用 SPM 中断

如果 SPM 中断响应被允许，一旦 SPMCR 寄存器中的 SPMEN 位被清零，就会产生 SPM 中断申请。这意味着使用 SPM 中断，可以代替软件中对 SPMCR 寄存器的轮循。在使用 SPM 中断时，中断向量表应该移到引导程序加载区中，以避免由于 RWW 区的读操作被阻断，使产生的中断无法进入 RWW 区。如何移动中断向量表，请参见中断处理部分。

5．对 BLS（引导程序加载区）的自编程

当引导锁定位 11 未被编程（BLB11 = 1）时，则允许对引导程序加载区进行自编程更新。此时要特别注意，一旦意外地对引导程序加载区的写入，会破坏整个引导程序加载区中的程序，造成以后的引导加载无法实现。如果引导加载程序自己更新自己不是必需的，建议对引导锁定位 11 进行编程（BLB11 = 0），以保护引导加载区内容不会被改变。

6．在自编程过程中防止对 RWW 区的读操作

在自编程（页擦除和页写入）期间，对 RWW 区的读操作被阻断。程序应避免在自编程操作过程中对 RWW 区寻址。当 RWW 区忙时（页擦除和页写入的期间），SPMCR 寄存器中的 RWWSB 位始终被置位。在自编程期间，中断向量表应该移到引导加载程序区或者将中断屏蔽掉。在自编程操作完成后，用户程序必须将 RWWSRE 寄存器中的 RWWSB 位清零，然后才能对 RWW 区再次寻址操作。

7．通过 SPM 指令对引导加载锁定位进行设置

要设置引导加载锁定位：应先把要设置的数据写入寄存器 R0；然后将"x0001001"写入 SPMCR 寄存器；在设置 SPMCR 后的 4 个时钟周期内执行 SPM 指令。引导锁定位是惟一由用户程序可设置的锁定位，用于锁定应用程序区和引导加载区，使其不能被程序更新（表 2-59、表 2-60）。

寄存器 R0 的 5-2 位为"0"时，在设置 SPMCR 寄存器（BLBSET = 1，SPMEN = 1）后的 4 个时钟周期内执行 SPM 指令，则相应的引导锁定位将被编程。在该操作中，Z 寄存器中的值无需考虑，但为了以后的兼容性，建议将其设置为&0001。同时，考虑到以后的兼容，寄存器 R0 的其他位应设置为"1"。在对引导加载锁定位的设置期间，可以读取整个 Flash 空间。

位	7	6	5	4	3	2	1	0
R0	1	1	BLB12	BLB11	BLB02	BLB01	1	1

8．E^2PROM 的写操作阻断自编程

一个 E^2PROM 的写操作，将阻断当前对 Flash 编程的处理，读取熔丝位和锁定位的操作也被阻断。建议在设置 SPMCR 寄存器前，先检查 EECR 寄存器中的 EEWE 位，确认其

已被清零。

9. 用程序读取熔丝位和锁定位

用户程序可以读取熔丝位和锁定位的设置。要读取熔丝位和锁定位：应先将Z寄存器设置为“0$0001”；然后将“x0001001”写入SPMCR寄存器；在设置SPMCR后的3个时钟周期内执行LMP指令，将熔丝位和锁定位的值读入目的寄存器中。在LPM指令执行后，或者在3个时钟周期内没有执行LPM指令，SPMEN位和BLBSET位将被自动清零。当BLBSET位和SPMEN位为“0”时，执行LPM指令的操作请查阅相关的指令部分。

使用上述相同的操作，可以读取不同的熔丝位和锁定位的设置值，仅需要改变Z寄存器中的地址。

位	7	6	5	4	3	2	1	0
Rd	FLB7	FLB6	FLB5	FLB4	FLB3	FLB2	FLB1	FLB0

- 读取系统熔丝位低位字节（Z = 0$0000）
- 读取系统锁定位（Z = 0$0001）
- 读取系统熔丝位高位字节（Z = 0$0003）

已编程的熔丝位和锁定位读出为“0”。未编程的读出为“1”。

位	7	6	5	4	3	2	1	0
Rd	-	-	BLB12	BLB11	BLB02	BLB01	LB2	LB1

位	7	6	5	4	3	2	1	0
Rd	-	-	BLB12	BLB11	BLB02	BLB01	LB2	LB1

10. 防止Flash被破坏

当系统电压Vcc过低时，会导致MCU和Flash存储器无法正常工作，造成程序存储器Flash中的内容被破坏。形成这种情况的原因有：系统电压低于常规的Flash写操作所需的最低电压；系统电压低于MCU执行指令所需的最低电压，引起MCU非正常执行指令。

通过以下这些措施可以避免和防止Flash被破坏（任意一条已经足够）：

（1）如果没有必要使用引导加载功能更新系统，将引导加载锁定位编程，以避免任何的自编程更新。

（2）当电源电压不足时，保持AVR的复位为有效（低电平）。如果工作电压与BROWN-OUT检测电压相匹配，使能芯片内部BROWN-OUT检测器；如果不匹配，建议使用外部低电压复位保护电路。如果在Flash写操作过程中，出现了复位信号，只要有足够的电压，MCU则在该次写操作完成后才进入复位状态。

（3）在Vcc过低时，保持AVR内核工作在掉电休眠模式。这可以避免MCU试图译码和执行指令，从而可以有效保护SPMCR和Flash被意外地写操作。

11. 使用SPM指令的编程时间

内部可校准的RC振荡器为访问Flash的自编程操作提供时钟。编程Flash的时间见

表 2-70。

表 2-70　SPM 编程时间

SPM 对 Flash 操作	最短时间	最长时间
页擦除、页写入、写锁定位	3.7ms	4.5ms

2.15.4　一个简单的引导加载汇编程序

```
;The routine writes one page of data from RAM to Flash
;The first data location in RAM is pointed to by the Y pointer
;The first data location in Flash is pointed to by the Z pointer
;Error handling is not included
;The routine must be placed inside the boot space
;(at least the Do_spm sub routine). Only code inside NRWW section can
;be read during self programming (page erase and page write).
;Registers used: r0, r1, temp1 (r16), temp2 (r17), looplo (r24),
;loophi (r25), spmcrval (r20)
;Storing and restoring of registers is not included in the routine
;Register usage can be optimized at the expense of code size
;It is assumed that either the interrupt table is moved to the Boot
;loader section or that the interrupts are disabled.

.equ PAGESIZEB = PAGESIZE*2
    ;PAGESIZEB is page size in BYTES, not words
.org SMALLBOOTSTART
Write_page:
    ;page erase
    ldi spmcrval, (1<<PGERS) | (1<<SPMEN)
    call Do_spm

    ;re-enable the RWW section
    ldi spmcrval, (1<<RWWSRE) | (1<<SPMEN)
    call Do_spm

    ;transfer data from RAM to Flash page buffer
    ldi looplo, low(PAGESIZEB)       ;init loop variable
    ldi loophi, high(PAGESIZEB) ;not required for PAGESIZEB<=256
Wrloop:
    ld r0, Y+
```

```
    ld r1, Y+
    ldi spmcrval, (1<<SPMEN)
    call Do_spm
    adiw ZH:ZL, 2
    sbiw loophi:looplo, 2          ;use subi for PAGESIZEB<=256
    brne Wrloop

    ;execute page write
    subi ZL, low(PAGESIZEB)      ;restore pointer
    sbci ZH, high(PAGESIZEB)         ;not required for PAGESIZEB<=256
    ldi spmcrval, (1<<PGWRT) | (1<<SPMEN)
    call Do_spm

    ;re-enable the RWW section
    ldi spmcrval, (1<<RWWSRE) | (1<<SPMEN)
    call Do_spm

    ;read back and check, optional
    ldi looplo, low(PAGESIZEB)       ;init loop variable
    ldi loophi, high(PAGESIZEB) ;not required for PAGESIZEB<=256
    subi YL, low(PAGESIZEB)      ;restore pointer
    sbci YH, high(PAGESIZEB)
Rdloop:
    lpm r0, Z+
    ld r1, Y+
    cpse r0, r1
    jmp Error
    sbiw loophi:looplo,1           ;use subi for PAGESIZEB<=256
    brne Rdloop

    ;return to RWW section
    ;verify that RWW section is safe to read
Return:
    in temp1, SPMCR
    sbrs temp1, RWWSB    ;If RWWSB is set, the RWW section is not ready yet
    ret
    ;re-enable the RWW section
    ldi spmcrval, (1<<RWWSRE) | (1<<SPMEN)
    call Do_spm
    rjmp Return
```

```
Do_spm:
    ;check for previous SPM complete
Wait_spm:
    in temp1, SPMCR
    sbrc temp1, SPMEN
    rjmp Wait_spm
    ;input: spmcrval determines SPM action
    ;disable interrupts if enabled, store status
    in temp2, SREG
    cli
    ;check that no EEPROM write access is present
Wait_ee:
    sbic EECR, EEWE
    rjmp Wait_ee
    ;SPM timed sequence
    out SPMCR, spmcrval
    spm
    ;restore SREG (to enable interrupts if originally enabled)
    out SREG, temp2
    ret
```

2.16 ATmega8 存储器编程

本节主要介绍 ATmega8 的 Flash 程序存储器和 E^2PROM 数据存储器，以及熔丝位、锁定位和校正位的功能。并详细说明如何在外部采用并行/串行方式读取、编程存储器，设置熔丝位、锁定位。

2.16.1 ATmega8 的锁定位、熔丝位、标识位和校正位

1．程序和数据存储器锁定位

ATmega8 提供一个字节宽度的锁定位，共包括 6 个锁定标志位（见表 2-71），它们的默认状态为“1”（未编程），对这些位编程后，其状态为“0”（被编程）。使用芯片擦除命令，可擦除所有的锁定位，使它们恢复为“1”（未编程）。表 2-72 给出了锁定位 LB2、LB1 的加密锁定模式。BLB1x 和 BLB0x 的锁定模式见 2.15 节的表 2-66 和表 2-67。

表 2-71　锁定字节

位名称	位	用途	默认值（出厂值）
	7		1（未编程）
	6		1（未编程）
BLB12	5	引导锁定位	1（未编程）
BLB11	4	引导锁定位	1（未编程）
BLB02	3	引导锁定位	1（未编程）
BLB01	2	引导锁定位	1（未编程）
LB2	1	加密锁定位	1（未编程）
LB1	0	加密锁定位	1（未编程）

表 2-72　编程锁定保护模式

加密锁定位			保护类型（用于芯片加密）
锁定方式	LB2	LB1	
1	1	1	无任何编程加密锁定保护
2	1	0	禁止串/并行方式对 Flash 和 E^2PROM 的再编程 禁止串/并行方式对熔丝位的编程
3	0	0	禁止串/并行方式对 Flash 和 E^2PROM 的再编程和校验 禁止串/并行方式对熔丝位的编程

2．熔丝位

ATmega8 有两个字节的熔丝位：熔丝位高字节（FHB）和熔丝位低字节（FLB），表 2-73、表 2-74 给出了它们的名称、作用和出厂设定值。熔丝位未编程的状态为“1”，被编程后的状态为“0”。

在表 2-73、表 2-74 中，熔丝位 SPINE 不能通过串行方式编程；熔丝位 CKOPT 的作用与 CKSEL 有关（详见“系统时钟源”）。熔丝 BOOTSZ1..0 的默认值定义的引导加载区为最大值 1024 字（见本章 2.15 节、表 2-63）。

表 2-73　熔丝高位字节

熔丝位名称	位	用途	默认值（出厂值）
RSEDISBL	7	设定引脚 PC6 为 I/O 口或 RESET	1（PC6 为 RESET 引脚）
WDTON	6	WDT 总为有效	1（由 WDTCR 位设定 WDT 是否有效）
SPINE	5	允许串行编程和数据下载	0（允许 SPI 编程）
CKOPT	4	晶振选项	1
EESAVE	3	芯片擦除时保护 E^2PROM	1（E^2PROM 无保护）
BOOTSZ1	2	设置引导加载区大小	0
BOOTSZ0	1	设置引导加载区大小	0
BOOTRST	0	设置复位向量	1

表 2-74 熔丝低位字节

熔丝位名称	位	用　途	默认值（出厂值）
BODLEVEL	7	BOD 触发电平	1
BODEN	6	BOD 允许	1（禁止 BOD）
SUT1	5	设置复位启动延时时间	1
SUT0	4	设置复位启动延时时间	0
CKSEL3	3	选择时钟源	0
CKSEL2	2	选择时钟源	0
CKSEL1	1	选择时钟源	0
CKSEL0	0	选择时钟源	1

熔丝 SUT1..0 的默认值定义了最大的复位启动延时时间；熔丝 CKSEL3..0 的默认值定义使用芯片内部的 RC 振荡源（1MHz）为系统时钟源。

芯片擦除的操作不能改变所有熔丝位的状态。当加密锁定熔丝位 LB1 被编程后，其他所有的熔丝位被锁定，因此，应在编程 LB1 前改变其他熔丝位的状态。

3．芯片标识位

所有 ATMEL 的微控制器都有 3 字节的芯片标识位，用以指定该器件的型号。无论芯片是否被加密锁定，芯片标识位总是可以由串行模式和并行模式读出的。在 ATmega8 中除了 Flash 程序存储器空间、E^2PROM 数据存储器空间之外，还有一个独立的只读存储器空间。该只读存储器空间的存储单元宽度为 16 位（字），芯片标识位的 3 个字节驻留在该只读存储器空间中，占据前 3 个存储单元的低位字节。ATmega8 单片机的芯片标识如表 2-75 所示。

表 2-75 ATmega8 单片机的芯片标识

地　址	高位字节内容	校 正 字 节	低位字节内容	标 识 意 义
0x000	0xnn	1M 校正参数	0x1E	厂商 ATMEL
0x001	0xnn	2M 校正参数	0x93	8KB 程序存储器 Flash
0x002	0xnn	4M 校正参数	0x07	器件型号 ATmega8
0x003	0xnn	8M 校正参数		

4．内部 RC 振荡器频率校正字节（Calibration Byte）

ATmega8 有 4 个用于对内部 RC 振荡器频率进行校正的校正字节，它们驻留在只读存储器空间中，占据前 4 个存储单元的高位字节。芯片复位启动时，由硬件自动将 1MHz 频率的校正参数装入寄存器 OSCCAL 中。如系统选用其他 3 个频率的内部 RC 振荡器作为系统时钟源，那么对应频率的校正参数需要采用人工方式装入。

2.16.2 并行编程模式

本节讲述怎样采用并行方式，编程和验证 ATmega8 单片机的 Flash 程序存储器、E^2PROM 数据存储器及程序存储器的锁定位和熔丝位。编程时钟宽度最小为 250ns。并行编程 ATmega8 的电路如图 2.81 所示。

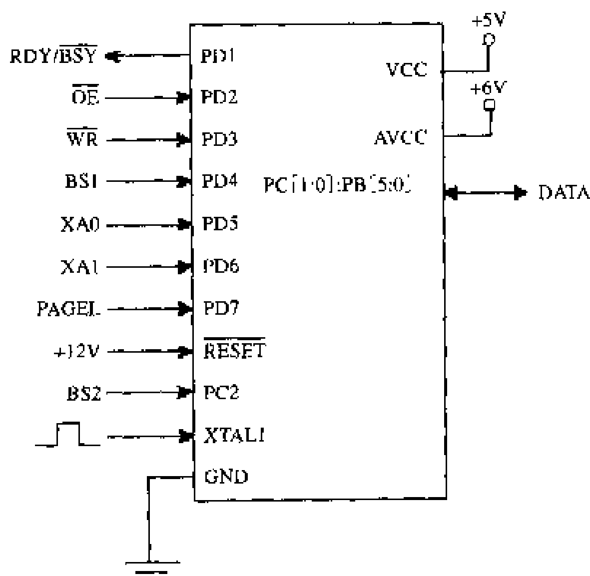


图 2.81 并行编程 ATmega8 的电路连接

1. 信号名称

在图 2.81 中，使用了一些描述并行编程的信号名称代替 ATmega8 的一些引脚名，在表 2-76 中没有说明的引脚仍用芯片原引脚名来表示。

表 2-76 并行编程方式的引脚对应表

编程信号	引脚名	I/O	功能说明
RDY/BSY	PD1	O	0：器件正在编程，1：器件就绪等待命令
OE	PD2	I	输出允许（低有效）
WR	PD3	I	写脉冲（低有效）
BS1	PD4	I	字节选择 1（0：选低字节，1：选高字节）
XA0	PD5	I	XTAL 编程动作位 0
XA1	PD6	I	XTAL 编程动作位 1
PAGEL	PD7	I	Flash 和 E^2PROM 页装入
BS2	PC2	I	字节选择 2（0：选低字节，1：选高字节）
DATA	{PC[1:0]:PC[5:0]}	I/O	双向数据口（OE = 0 时为输出）

信号 XA1 / XA0 位决定了当 XTAL1 引脚给出一个正脉冲时编程的动作，这两位的作

用见表 2-77。在芯片装入了编程命令后（命令字见表 2-78 所示），由输入的 WR 或 OE 脉冲启动命令的执行。

表 2-77 XA1 和 XA0 的动作

XA1	XA0	XTAL1 为时钟高电平时的动作
0	0	装入 Flash 或 E²PROM 地址，由 BS1 决定为地址的高/低字节
0	1	装入数据，由 BS1 决定为 Flash 地址的高/低字节
1	0	装入命令（写入/读出 Flash 或 E²PROM 操作开始）
1	1	无动作，闲置

表 2-78 命令字列表

命令字节	命令字功能定义
0x80	芯片擦除
0x40	写熔丝位
0x20	写锁定位
0x10	写 Flash
0x11	写 E²PROM
0x08	读芯片标识位字节和校正熔丝位字节
0x04	读熔丝和锁定位
0x02	读 Flash
0x03	读 E²PROM

表 2-79 给出了 ATmega8 的 Flash 存储器和 E²PROM 存储器的页分配参数。

表 2-79 ATmega8 的 Flash 和 E²PROM 的页分配

Flash Size	Page Size	PCWORD	No. of Pages	PCPAGE	PCMSB
4K 字（8K 字节）	32 字	PC[4..0]	128	PC[11..5]	11
E²PROM Size	Page Size	PCWORD	No. of Pages	PCPAGE	EEAMSB
512 字节	4 字节	EEA[1..0]	128	EEA[8..2]	8

2. 进入并行编程模式

按下面步骤可使芯片进入并行编程模式：

（1）在 V_{CC} 和 GND 之间加上 4.5V～5.5V 电压，并等待至少 100μs 时间。

（2）把 RESET 设置为“0”，并等待至少 6 个 XTAL1 时钟脉冲触发。

（3）设置 PAGEL、XA1、XA2、BS1 全部为“0”，并等待至少 100ns 时间。

（4）在 RESET 引脚加 11.5V～12.5V 电压，并等待至少 100ns 时间。在此期间，不得改变 PAGEL、XA1、XA2、BS1 信号，否则芯片将不能进入并行编程模式。

如果外部引脚 PC6 已由熔丝位 RSTDISBL 设定，不作为 RESET 使用，以及系统振荡源设定使用外部 RC 振荡器或外接晶振，XTAL1 引脚上不能提供合适编程时钟信号时，就

不能按上面的方式进入并行编程模式。在此情况下，先进行以下处理，再按上面方式进入并行编程模式。

①设置 PAGEL、XA1、XA2、BS1 全部为“0”；

②在 V_{CC} 和 GND 之间加上 4.5V～5.5V 电压的同时，在 RESET 引脚加 11.5V～12.5V 电压；

③等待 100ns；

④执行芯片擦除操作，擦除加密锁定位；

⑤重新编程相关的熔丝位，选定系统振荡源为外部时钟驱动（CKSEL3..0 = 0000），PC6 作为 RESET 输入引脚（RSTDISBL = 1）；

⑥撤消系统电源，或将 RESET 引脚拉为低电平，退出编程模式；

⑦按前面的方式进入并行编程模式。

由于装载的命令字和地址字在编程期间不会改变，因此以下几点可以提高编程的效率。

- 在同一存储器空间连续多次读取或写入时（寻址地址可以不同），命令字只需要装载一次。
- 芯片擦除后，如果写入 Flash 或 E^2PROM 的数据为 0xFF 时，可以跳过。
- 在连续读取或写入 Flash 时，只有当寻址为一个新的 256 个字节窗口时（E^2PROM 为新的 256 个字节窗口）才需要重新装载一次地址的高位字节。

3．芯片擦除

芯片擦除功能是擦除 Flash 和 E^2PROM 存储器的内容以及将芯片的加密锁定位 LB2 和 LB1 置“1”（去除对存储器的加密锁定保护）。擦除操作先将存储器内容全部擦除，然后再擦除（置位）加密锁定位。但芯片擦除功能并不对其他熔丝位进行任何操作，熔丝位的状态并不改变。如果熔丝位 EESAVE 为“0”时，芯片擦除操作不能将 E^2PROM 存储器的内容擦除掉。在编程之前必须执行一个芯片擦除操作。

①设置 XA1、XA0 为“10”，允许装入命令；

②设置 BS1 为“0”；

③设置 DATA =“0x80”，为芯片擦除命令字；

④XTAL1 输入的正脉冲将命令字装入芯片；

⑤WR 输入一个负脉冲，开始执行命令擦除芯片。此时 RDY/BSY 输出低电平；

⑥等待 RDY/BSY 变高，然后装入下一个命令。

4．编程 Flash

Flash 存储器是按页组织的，编程过程中，首先把写入 Flash 的数据锁存到临时缓冲页中，然后再将缓冲页中的数据整个写入 Flash。因此，可以仅编程 Flash 存储器的一页。

（1）A：装载“编程 Flash”命令

①设置 XA1、XA0 为“10”，允许装入命令；

②设置 BS1 为“0”；

③设置 DATA =“0x10”，为写 Flash 命令；

④XTAL1 输入的正脉冲将命令字装入芯片。

（2）B：装入地址低位字节

①设置 XA1、XA0 为“00”，允许装入地址；

②设置 BS1 为“0”，选择低位地址；

③设置 DATA = 地址低位字节（0x00～0xFF）；

④XTAL1 输入的正脉冲将地址低位字节装入芯片。

（3）C：装入数据低位字节

①设置 XA1、XA0 为“01”，允许装入数据；

②设置 DATA = 数据低位字节（0x00～0xFF）；

③XTAL1 输入的正脉冲将数据低位字节装入芯片。

（4）D：装入数据高位字节

①设置 BS1 为“1”，选择数据高位字节；

②设置 XA1、XA0 为“01”，允许装入数据；

③设置 DATA = 数据高位字节（0x00～0xFF）；

④XTAL1 输入的正脉冲将数据高位字节装入芯片。

（5）E：锁存数据

①设置 BS1 为“1”，选择数据高位字节；

②PAGEL 输入的正脉冲将装入的高位和低位数据锁存到临时缓冲页中。

（6）F：重复步骤（2）～（5），直到缓冲页中填满数据。在数据装入中，对于 ATmega8，低位地址字节中的第 7～5 位作为页寻址地址的低 3 位，而 4～0 位作为页中的字寻址地址（5 位 32 个地址）。

（7）G：装入地址高位字节

①设置 XA1、XA0 为“00”，允许装入地址；

②设置 BS1 为“1”，选择高位地址；

③设置 DATA = 地址高位字节（0x00～0xFF）；

④XTAL1 输入的正脉冲将地址高位字节装入芯片，地址高位字节同低位地址的高 3 位组成页地址。

（8）H：编程 Flash 页

①设置 BS1 为“0”；

②WR 输入一个负脉冲，开始执行页编程操作。此时 RDY/BSY 输出低电平；

③等待 RDY/BSY 变高，然后装入下一个命令。

（9）I：重复步骤（2）～（8），直到编程整个 Flash 空间，或仅仅编程几个页。

（10）J：结束页编程

①设置 XA1、XA0 为“10”，允许装入命令；

②设置 DATA =“0x00”，无操作命令；

③XTAL1 输入的正脉冲将命令装入芯片，内部写信号复位。

编程时，Flash 存储器的页面寻址以及一个页面内的字寻址方式参见图 2.80。图 2.82 为编程 Flash 程序存储器的时序图。

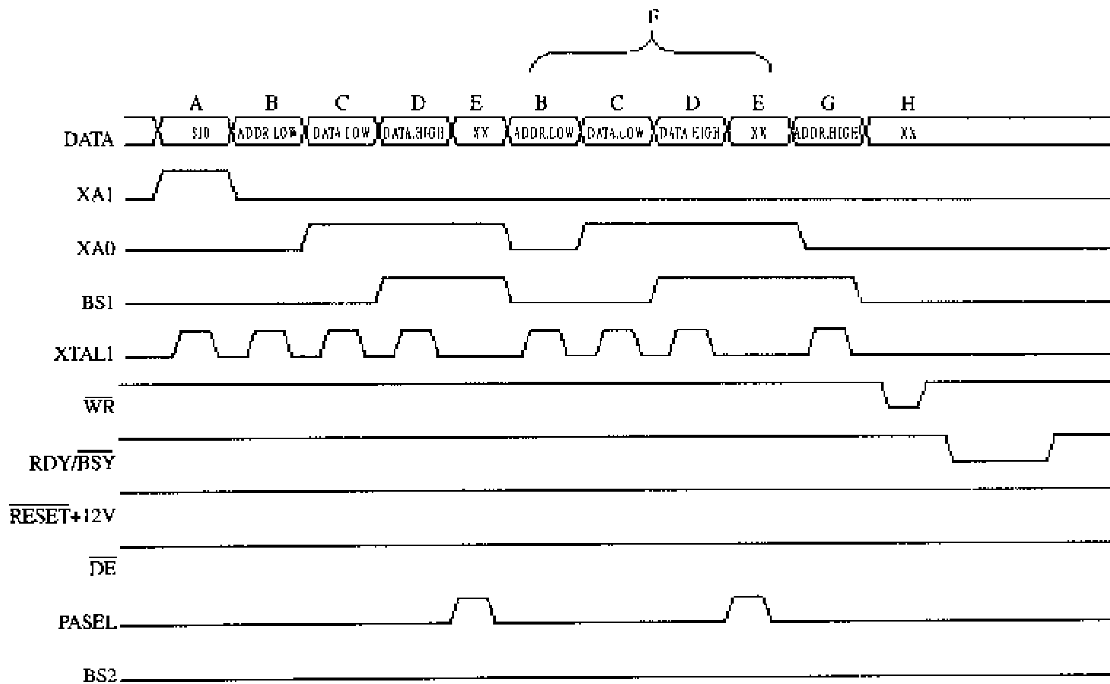


图 2.82 编程 Flash 存储器的时序

5. 编程 E^2PROM

E^2PROM 存储器也是按页组织的，编程过程中，首先把写入 E^2PROM 的数据锁存到临时缓冲页中，然后再将缓冲页中的数据整个写入 E^2PROM。因此，可以仅编程 E^2PROM 存储器中的一页。编程 E^2PROM 的步骤如下（参考 Flash 编程部分的命令、地址和数据的装载）：

（1）A：装入写 E^2PROM 命令“0x11”；

（2）G：装入 E^2PROM 地址高位（0x00～0xFF）；

（3）B：装入 E^2PROM 地址低位（0x00～0xFF）；

（4）C：装入数据（0x00～0xFF）；

（5）E：锁存数据到缓冲页（PAGEL 输入一个正脉冲）；

（6）K：重复步骤（3）～（5），直到缓冲页中填满数据（4 字节/页）；

（7）L：编程 E^2PROM 页。

- 设置 BS1 为“0”；
- WR 输入一个负脉冲，开始执行 E^2PROM 页编程操作。此时 RDY/BSY 输出低电平；
- 等待 RDY/BSY 变高，然后装入下一个命令。

图 2.83 为编程 E^2PROM 存储器时序图。

6. 读 Flash 存储器

读 Flash 存储器的过程如下（参考 Flash 编程部分的命令、地址和数据的装载）：

（1）A：装入读 Flash 命令“0x02”；

（2）G：装入 Flash 地址高位（0x00～0xFF）；

（3）B：装入 Flash 地址低位（0x00～0xFF）；

（4）设置 OE 和 BS1 为“00”，从 DATA 口读取 Flash 数据低字节；

（5）设置 BS1 为“1”，从 DATA 口读取 Flash 数据高字节；

（6）设置 OE 为“1”。

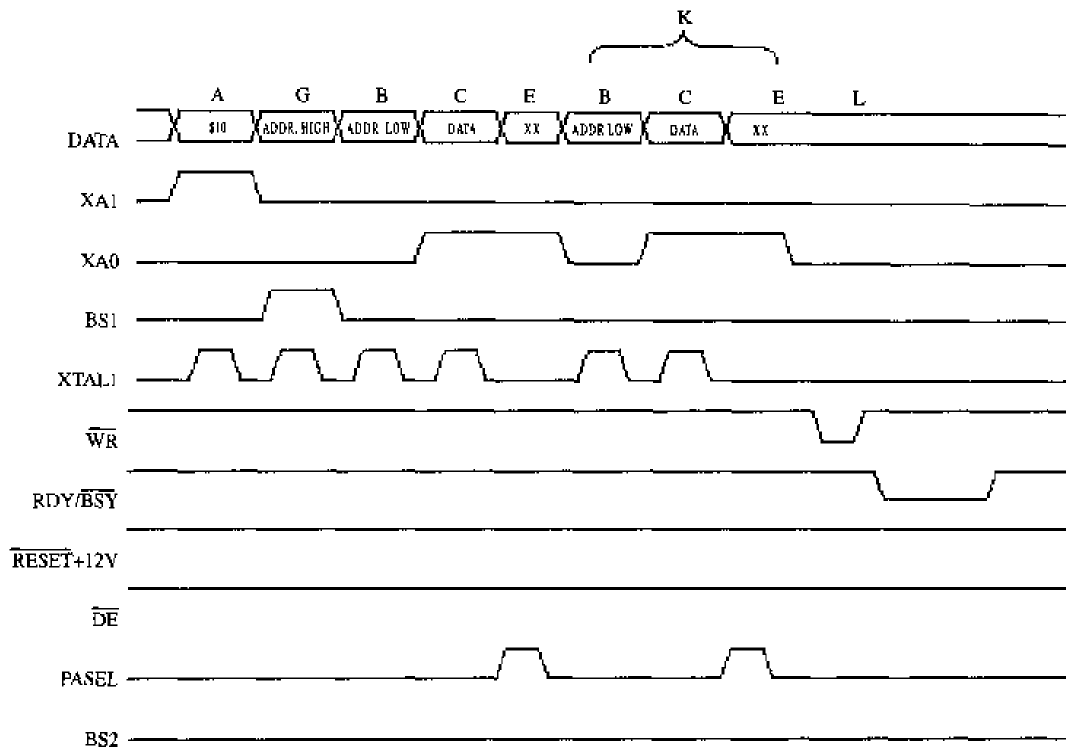


图 2.83 编程 E^2PROM 存储器时序图

7．读 E^2PROM 存储器

读 E^2PROM 存储器的过程如下（参考 Flash 编程部分的命令、地址和数据的装载）：

（1）A：装入读 E^2PROM 命令“0x03”；

（2）G：装入 E^2PROM 地址高位（0x00～0xFF）；

（3）B：装入 E^2PROM 地址低位（0x00～0xFF）；

（4）设置 OE 和 BS1 为“00”，从 DATA 口读取 E^2PROM 数据；

（5）设置 OE 为“1”。

8．编程熔丝位低字节

编程熔丝位低字节的过程如下（参考 Flash 编程部分的命令、地址和数据的装载）：

（1）A：装入写熔丝位命令“0x40”；

（2）C：装入一个字节数据（“1”表示擦除熔丝，“0”表示编程熔丝）；

（3）设置 BS1 和 BS2 为“00”；

（4）WR 输入一个负脉冲，开始执行写熔丝位操作。此时 RDY/BSY 输出低电平；

（5）等待 RDY/BSY 变高，然后装入下一个命令。

9．编程熔丝位高字节

编程熔丝位高字节的过程如下（参考 Flash 编程部分的命令、地址和数据的装载）：

（1）A：装入写熔丝位命令“0x40”；

（2）C：装入一个字节数据（“1”表示擦除熔丝，“0”表示编程熔丝）；

（3）设置 BS1 和 BS2 为“10”，表示选择数据为高位字节；

（4）WR 输入一个负脉冲，开始执行写熔丝位操作。此时 RDY/BSY 输出低电平；

（5）等待 RDY/BSY 变高；

（6）设置 BS1 为“0”，表示选择数据低位字节。

10．编程加密锁定位

编程加密锁定位的过程如下（参考 Flash 编程部分的命令、地址和数据的装载）：

（1）A：装入写锁定位命令“0x20”；

（2）C：装入一个字节数据（“0”表示编程加密锁定位）；

（3）WR 输入一个负脉冲，开始执行写加密锁定位操作。此时 RDY/BSY 输出低电平；

（4）等待 RDY/BSY 变高。

注意：加密锁定位的擦除，只能使用芯片擦除命令。

11．读熔丝位和加密锁定位

读熔丝位和加密锁定位过程如下（参考 Flash 编程部分的命令、地址和数据的装载）：

（1）A：装入读熔丝位和锁定位命令“0x04”；

（2）设置 OE、BS2、BS1 为“000”，从 DATA 口读取低字节熔丝位的状态（“0”表示已编程状态）；

（3）设置 OE、BS2、BS1 为“011”，从 DATA 口读取高字节熔丝位的状态（“0”表示已编程状态）；

（4）设置 OE、BS2、BS1 为“001”，从 DATA 口读取锁定位的状态（“0”表示已编程状态）；

（5）设置 OE 为“1”。

12．读芯片标识位

读芯片标识位过程如下（参考 Flash 编程部分的命令、地址和数据的装载）：

（1）A：装入读芯片标识位命令“0x08”；

（2）B：装入地址的低位字节（0x00～0x02）；

（3）设置 OE 和 BS1 为“00”，从 DATA 口读取指定的芯片标识位字节数据；

（4）设置 OE 为“1”。

13．读校正位

读校正位过程如下（参考 Flash 编程部分的命令、地址和数据的装载）：

（1）A：装入读校正位命令“0x08”；

（2）B：装入地址的低位字节（0x00～0x03）；

（3）设置 OE 和 BS1 为“01”，从 DATA 口读取校正位字节数据；

（4）设置 OE 为“1”。

2.16.3 串行编程模式

在引脚 RESET 接地时，Flash 程序存储器、E^2PROM 数据存储器、熔丝位和加密锁定位都可以通过串行 SPI 总线来编程。该串行接口包括引脚 SCK（串行时钟）、MOSI（输入）、MISO（输出）。当 RESET 引脚为低电平后，应先执行串行编程允许指令，再执行编程 / 擦除操作。图 2.84 为串行编程接线图。

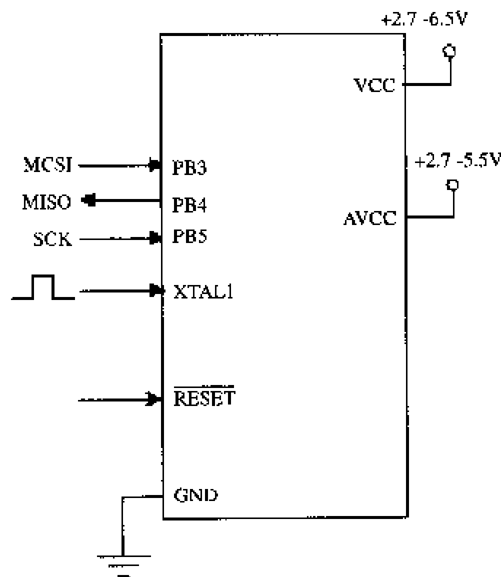


图 2.84 串行编程接线图

如果已经设置使用芯片内部 RC 振荡器，XTAL1 可以不用。

当仅需要编程 E^2PROM 时，由于内部定时的编程操作中已经包含了对 E^2PROM 的自动擦除周期（仅在串行编程模式下），因此无需先执行芯片擦除指令。芯片擦除指令把程序和数据存储器的每一单元都变成 0xFF。

根据系统时钟源的不同，串行编程时钟 SCK 必须同系统时钟相配合，SCK 的低电平和高电平的最小时间定义如下：

Low：大于 2 个 MCU 时钟周期($f_{CK}<12MHz$)；大于 3 个 MCU 时钟周期($f_{CK}\geq 12MHz$)。

High：大于 2 个 MCU 时钟周期($f_{CK}<12MHz$)；大于 3 个 MCU 时钟周期($f_{CK}\geq 12MHz$)。

1. 串行编程时序与命令

在串行编程中，输入的串行数据由 SCK 的上升沿输入，输出的串行数据由 SCK 的下降沿输出。串行方式编程和校验 ATmega8 单片机时序和命令见图 2.85 和表 2-80。

2. 串行编程的实现

（1）上电过程：在 V_{CC} 和 GND 之间上电，同时将 RESET 和 SCK 设置为“0”（如果不能保证 SCK 在上电期间一直保持为低电平，则在拉低 SCK 时，控制 RESET 引脚，输入一个至少为 2 个 MCU 时钟周期的正脉冲）。

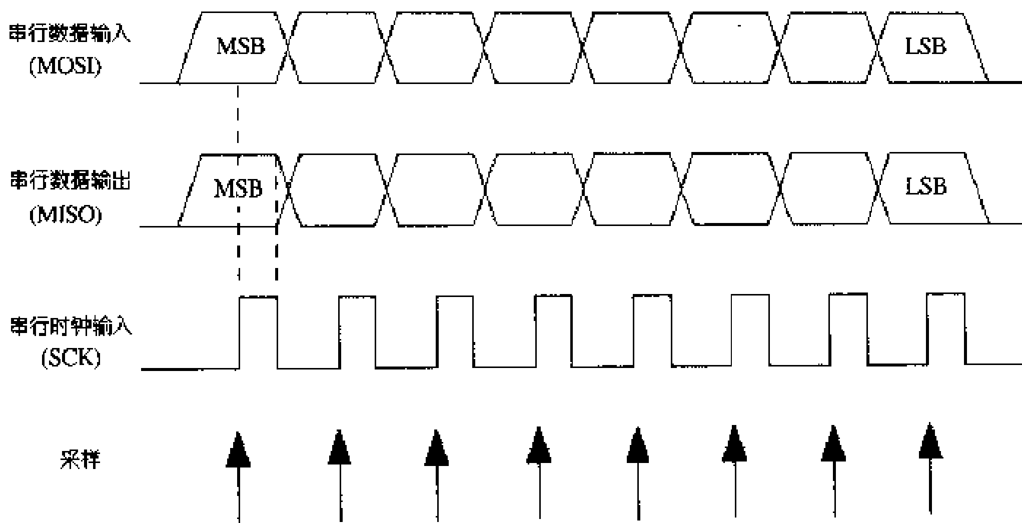


图 2.85　串行编程时序图

表 2-80　串行编程命令表

命　令	命令字格式				操　作
	Byte1	Byte2	Byte3	Byte4	
编程允许	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	在复位拉低后，允许串行编程
芯片擦除	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	芯片擦除
读 Flash	0010 H000	0000 aaaa	bbbb bbbb	oooo oooo	从 Flash 的字地址 a:b 处读 H（高/低）数据 oooo oooo
写缓冲页	0100 H000	0000 xxxx	xxxb bbbb	iiii iiii	写 H（高/低）数据 iiii iiii 到缓冲页字地址 b 处，在同一地址处先写低位字节再写高位字节
写 Flash 页	0100 1100	0000 aaaa	bbbx xxxx	xxxx xxxx	将缓冲页数据写入 Flash 的 a:b 页中
读 E^2PROM	1010 0000	00xx xxxa	bbbb bbbb	oooo oooo	从 E^2PROM 的字节地址 a:b 处读数据 oooo oooo
写 E^2PROM	1100 0000	00xx xxxa	bbbb bbbb	iiii iiii	写数据 iiii iiii 到 E^2PROM 字节地址 a:b 处
读锁定位	0101 1000	0000 0000	xxxx xxxx	xxoo oooo	读锁定位状态 xxoo oooo
写锁定位	1010 1100	111x xxxx	xxxx xxxx	11ii iiii	设置锁定位状态为 11ii iiii
读芯片标识	0011 0000	00xx xxxx	xxxx xxbb	oooo oooo	读地址为 bb 的芯片标识字 oooo oooo
写熔丝低字节	1010 1100	1010 0000	xxxx xxxx	iiii iiii	设置低字节熔丝位状态为 iiii iiii
写熔丝高字节	1010 1100	1010 1000	xxxx xxxx	iiii iiii	设置高字节熔丝位状态为 iiii iiii
读熔丝低字节	0101 0000	0000 0000	xxxx xxxx	oooo oooo	读低字节熔丝位状态 oooo oooo

续表

命令	命令字格式				操作
	Byte1	Byte2	Byte3	Byte4	
读熔丝高字节	0101 1000	0000 1000	xxxx xxxx	oooo oooo	读高字节熔丝位状态 oooo oooo
读校正字节	0011 1000	00xx xxxx	xxxx xxbb	oooo oooo	读地址为 bb 的校正字 oooo oooo

注：a=高位地址，b=低位地址，H=0（低字节）/L（高字节），o=数据输出，i=数据输入，x=任意

（2）等待至少 20ms，由 MOSI 引脚送入允许串行编程指令，使芯片进入串行编程状态。

（3）命令的输入必须与 SCK 时钟同步，否则命令是不能被执行的。如果时序正确，在输入允许串行编程的第 3 个字节时，芯片将回送出该命令的第 2 个字节（0x53）。当命令 4 个字节全部输入后，如果没有收到芯片回送的 0x53，控制 RESET 输入一个正脉冲，然后再次输入允许串行编程命令。

（4）需要时先输入芯片擦除命令。等待 9ms 再进行下一个编程的过程，确保芯片擦除操作的完成。

（5）Flash 是按页编程的，即一次写操作将对一个页（ATmega8 一页为 32 个字）编程，写入的数据应先装入临时缓冲页，写入地址为页内字寻址地址，宽度为 5 位（32）。对相同的一个字地址，应先装入字的低位字节，再装入字的高位字节。缓冲页填充完成后，使用写 Flash 命令将缓冲页内容写入程序存储器中，该命令中的 7 位地址为页寻址地址。如果不采用轮循方式写存储器，则要等待 4.5ms 再进行下一页的写操作过程，以保证当前页写操作正常进行。在当前页写操作期间，不能进行其他的写操作。

（6）E^2PROM 是按字节编程的，即一次写操作可以直接编程一个字节，因此命令中要同时给出寻址地址和数据。在写 E^2PROM 的操作中，先自动对指定字节进行擦除，再写入数据。如果不采用轮循方式写存储器，则要等待 9ms 再进行下一字节的写操作过程，以保证当前字节写操作正常进行。

（7）任何存储器空间的存储单元内容都可以通过读命令读出并校验。读命令读出的指定地址的数据从串行输出引脚 MISO 输出。

（8）在编程结束后，RESET 引脚可以设置为高电平，使芯片开始正常执行写入的程序。

（9）掉电过程。如果芯片编程完毕需要取消电源时，应设置 RESET 为“1”，然后断开 Vcc。

3．使用轮循方式写 Flash

在一个写 Flash 页期间，可以读取该页中任何一个地址的内容。如该地址还未被编程，则读出值为 0xFF；如已被编程，则读出的是写入数据。因此，可以使用查询的方法，确定是否当前页已经编程完毕，可以开始新的页的编程操作。注意，如果写入的值为 0xFF 时，该值是不能用于轮循判断的，此时必须等待 4.5ms，再开始写新的页。

4．使用轮循方式写 E^2PROM

在写一个字节到 E^2PROM 间，可以读取该字节的内容。如该地址还未被编程，则读出

值为 0xFF；如已被编程，则读出的是写入数据。因此，可以使用查询的方法，确定是否当前字节已经编程完毕，可以开始下一个字节的编程操作。注意，如果写入的值为 0xFF 时，该值是不能用于轮循判断的，此时必须等待 9ms，再开始写下一个字节。

2.17　E²PROM 数据存储器读/写访问

ATmega8 有 512 个字节的 E²PROM 数据存储器，其构成一个独立的存储器空间。每一个字节都能被读取或写入。E²PROM 至少有 10 万次写入/擦除周期的寿命。本节介绍如何在用户程序中访问 E²PROM。

2.17.1　E²PROM 读/写访问

在程序中访问 E²PROM 是通过 I/O 空间的寄存器实现的。

ATmega8 采用芯片内部可校准的 RC 振荡器的 1MHz（与 CKSEL 的状态无关）作为访问 E²PROM 的定时时钟。E²PROM 编程时间为 8448 个周期，典型值为 8.5ms。自定时功能的应用，可使用户程序检测什么时候能够写入下一个字节。如果用户程序包含了对 E²PROM 进行写入的指令，必须注意一些问题。在有大滤波电容的电源供电下，Vcc 在上电和掉电的过程中，上升或下降得很慢。这将导致器件在某一段时间内会工作在低于最小工作电压下，引起 E²PROM 的误写入。

为了防止无意的 E²PROM 写入，必须遵照一个规范的写入顺序。

当读取 E²PROM 时，CPU 将暂停 4 个时钟周期，然后再执行下一条指令。当写 E²PROM 时，CPU 暂停 2 个时钟周期，然后再执行下一条指令。

2.17.2　寄存器描述

1．E²PROM 地址寄存器——EEARH 和 EEARL

位	15	14	13	12	11	10	9	8	
$1F($003F)								EEAR8	EEARH
$1E($003E)	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
位	7	6	5	4	3	2	1	0	
读/写	R	R	R	R	R	R	R	R/W	
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	x	

● 位 15..9——保留位

读出始终为“0”。

● 位 8..0——EEAR8..0：E^2PROM 地址

E^2PROM 地址寄存器（EEARH 和 EEARL）指定了 512 个字节的 E^2PROM 空间的地址。E^2PROM 地址空间是线性排列的，从 0 到 511。系统初始时，EEAR 中的值是不确定的，因此在读取 E^2PROM 前必须写入一个正确的地址值。

2．E^2PROM 数据寄存器——EEDR

位	7	6	5	4	3	2	1	0	
$1D($003D)	MSB							LSB	EEDR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

● 位 7..0——EEDR7..0：E^2PROM 数据

对于写 E^2PROM 操作，EEDR 寄存器包含了将要写入 E^2PROM 中的数据，EEAR 寄存器给出其地址。对于读 E^2PROM 操作，EEAR 寄存器为指定的地址，读出的数据在 EEDR 寄存器中。

3．E^2PROM 控制寄存器——EECR

位	7	6	5	4	3	2	1	0	
$1C($003C)					EERIE	EEMWE	EEWE	EERE	EECR
读/写	R	R	R	R	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	X	0	

● 位 7..4——保留位

读出始终为“0”。

● 位 3——EERIE：E^2PROM 准备好中断允许

如果 SREG 寄存器中的 I 位为“1”，置位 EERIE 将使能 E^2PROM 准备好中断。清零 EERIE 则将屏蔽该中断。只要 EEWE 位被清零，E^2PROM 准备好中断申请就会产生。

● 位 2——EEMWE：E^2PROM 主机写入允许

EEMWE 位决定当设置 EEWE 位为“1”时，是否导致 E^2PROM 被写入。当 EEMWE 被置位时，在 EEWE 为“1”的 4 个时钟周期内，将写数据到指定的地址；如果 EEMWE 为零，设置 EEWE 为“1”不能触发写 E^2PROM 操作。当 EEMWE 被软件设置为“1”后，在 4 个时钟周期后，硬件自动清零该位。

● 位 1——EEWE：E^2PROM 写允许

EEWE 位作为 E^2PROM 的写触发。当地址和数据被正确设置后，EEWE 位必须被写入为“1”触发数据写入到 E^2PROM 的操作。在置 EEWE 为“1”前，EEMWE 位必须为“1”（使能主机写 E^2PROM），否则，不能触发写 E^2PROM 的操作。写数据到 E^2PROM 应该遵守以下顺序（其中第（3）步和第（4）步不是必需的）：

（1）等待 EEWE 位变为零；

（2）等待 SPMCR 寄存器中的 SPMEN 位变为零；

（3）写新的 E^2PROM 地址到寄存器 EEAR（可选）；

（4）写新的 E^2PROM 数据到寄存器 EEDR（可选）；

（5）写逻辑“1”到 EEMWE 位，并同时写“0”到 EEWE 位；

（6）在置“1”EEMWE 位后的 4 个时钟周期内，写逻辑“1”到 EEWE 位。

E^2PROM 的编程操作不能在 CPU 写 Flash 存储器过程中进行。在开始一个 E^2PROM 写入前，软件必须检验 Flash 编程是否已经完成。步骤（2）只适合当程序包含引导加载，允许 CPU 对 Flash 编程。如果 CPU 从不更新 Flash，步骤（2）可以省略。

注意：在步骤 5 和步骤 6 之间发生中断将使写入过程失败，这是由于 E^2PROM 主机写入允许（EEMWE）超时。如果一个中断程序访问 E^2PROM 打断另一个对 E^2PROM 的访问，EEAR 或 EEDA 寄存器的值将被改变，导致被中断的 E^2PROM 访问操作失败。建议在所有以上步骤中清零全局中断允许标志位。

当写 E^2PROM 操作所需的时间过后，EEWE 位将被硬件自动清零。用户程序可以轮循 EEWE 标志等待其变为零。当 EEWE 被置“1”后，CPU 暂停两个时钟周期，然后再执行下一条指令。

- 位 0——EERE：E^2PROM 读允许

E^2PROM 读允许标志作为读取 E^2PROM 操作的触发。当 EEAR 寄存器被设置了正确的地址后，向 EERE 位写入逻辑“1”，来触发 E^2PROM 的读取操作。E^2PROM 的读取访问需要一个指令，并立即可以获得访问地址的的数据。当读取 E^2PROM 时，CPU 将暂停 4 个时钟周期，然后再执行下一个指令。

在开始读取 E^2PROM 前，用户程序应该轮循 EEWE 标志位，如果一个写 E^2PROM 的操作正在进行，此时既不可以读 E^2PROM，也不可以改变 EEAR 寄存器内容。

4．防止 E^2PROM 的误写入

当系统电压 Vcc 过低时，会导致 CPU 和 E^2PROM 存储器无法正常工作，而造成 E^2PROM 中的内容被破坏。形成这种情况的原因有：系统电压低于常规的 E^2PROM 写操作所需的最低电压；系统电压低于 CPU 执行指令所需的最低电压，引起 CPU 非正常执行指令。

通过以下这些措施可以避免和防止 E^2PROM 被破坏：

当电源电压不足时，保持 AVR 的复位为有效（低电平）。如果工作电压与 BROWN-OUT 检测电压相匹配，使能芯片内部 BROWN-OUT 检测器；如果不匹配，使用外部低电压复位保护电路。如果在 E^2PROM 写操作过程中，出现了复位信号，在电源电压有效时，CPU 将在完成该次写操作后进入复位状态。

2.17.3　简单的读写 E^2PROM 例程

下面给出读写 E^2PROM 的例程。在例程中，假定中断已经屏蔽，在读写 E^2PROM 期间不响应任何的中断。同时无对 Flash 程序存储器写的操作。

1. 写 E²PROM

● 汇编代码

```
EEPROM_write:
; Wait for completion of previous write
sbic EECR,EEWE
rjmp EEPROM_write
; Set up address (r18:r17) in address register
out EEARH, r18
out EEARL, r17
; Write data (r16) to data register
out EEDR,r16
; Write logical one to EEMWE
sbi EECR,EEMWE
; Start eeprom write by setting EEWE
sbi EECR,EEWE
ret
```

● C 程序代码

```
void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{
/* Wait for completion of previous write */
while(EECR & (1<<EEWE));
/* Set up address and data registers */
EEAR = uiAddress;
EEDR = ucData;
/* Write logical one to EEMWE */
EECR |= (1<<EEMWE);
/* Start eeprom write by setting EEWE */
EECR |= (1<<EEWE);
}
```

2. 读 E²PROM

● 汇编代码

```
EEPROM_read:
; Wait for completion of previous write
sbic EECR,EEWE
rjmp EEPROM_read
; Set up address (r18:r17) in address register
```

```
out EEARH, r18
out EEARL, r17
; Start eeprom read by writing EERE
sbi EECR,EERE
; Read data from data register
in r16,EEDR
ret
```

- C 程序代码

```
unsigned char EEPROM_read(unsigned int uiAddress)
{
/* Wait for completion of previous write */
while(EECR & (1<<EEWE));
/* Set up address register */
EEAR = uiAddress;
/* Start eeprom read by writing EERE */
EECR |= (1<<EERE);
/* Return data from data register */
return EEDR;
}
```

第3章　ATmega8指令系统

单片机的指令系统是一套控制计算机操作的代码，每一条指令规定 MCU 完成某种操作。实际的指令代码是由一组二进制数“0”和“1”组成的，称为机器语言。计算机只能识别和执行机器语言的指令代码。为了便于人们理解、记忆和使用，通常用汇编语言的形式（用助记符和专门的语言规则表示指令的功能和特征）来描述单片机的指令系统。用汇编语言指令编写的程序，必须通过汇编语言编译器（汇编语言开发平台）把它翻译成计算机能识别的机器码。

3.1　ATmega8 指令总述

3.1.1　ATmega8 指令表

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。ATmega8 单片机完全兼容 AVR 的指令系统，具有高性能的数据处理能力，能对位、半字节、字节和双字节数据进行各种操作，包括算术和逻辑运算、数据传送、布尔处理、控制转移和硬件乘法等操作。

ATmega8 共有 130 条指令，按功能可分为五大类：

- 算术和逻辑运算指令（28 条）如表 3-1 所示；
- 比较和转移指令（36 条）如表 3-2 所示；
- 数据传送指令（35 条）如表 3-3 所示；
- 位操作和位测试指令（28 条）如表 3-4 所示；
- MCU 控制指令（3 条）如表 3-5 所示。

下面分别给出 ATmega8 的 5 类共 130 条指令的简表。

表 3-1　算术和逻辑指令

算术和逻辑运算指令（28 条）						
指令	操作数	说明	操作	操作数范围	影响标志	指令周期
ADD	Rd,Rr	加法	Rd←Rd+Rr	0≤d≤31;0≤r≤31	Z,C,N,V,H,S	1
ADC	Rd,Rr	带进位加	Rd←Rd+Rr+C	0≤d≤31;0≤r≤31	Z,C,N,V,H,S	1

续表

算术和逻辑运算指令（28 条）						
指令	操作数	说明	操作	操作数范围	影响标志	指令周期
ADIW	Rdl,K	字加立即数	Rdh:Rdl←Rdh+Rdl+K	dl=24/26/28/30;0 ≤ K ≤63	Z,C,N,V,S	2
SUB	Rd,Rr	减法	Rd←Rd-Rr	0≤d≤31;0≤r≤31	Z,C,N,V,H,S	1
SUBI	Rd,K	减立即数	Rd←Rd-K	16≤d≤31;0≤K ≤255	Z,C,N,V,H,S	1
SBC	Rd,Rr	带进位减	Rd←Rd-Rr-C	0≤d≤31;0≤r≤31	Z,C,N,V,H,S	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,S	1
SBIW	Rdl,K	字减立即数	Rdh:Rdl←Rdh+Rdl-K	dl=24/26/28/30;0≤K ≤63	Z,C,N,V,S	2
AND	Rd,Rr	逻辑与	Rd←Rd • Rr	0≤d≤31;0≤r≤31	Z,N,V,S	1
ANDI	Rd,K	与立即数	Rd←Rd • K	16≤d≤31;0≤K ≤255	Z,N,V,S	1
OR	Rd,Rr	逻辑或	Rd← Rd∨Rr	0≤d≤31;0≤r≤31	Z,N,V,S	1
ORI	Rd,K	或立即数	Rd←Rd∨K	16≤d≤31;0≤K≤255	Z,N,V,S	1
EOR	Rd,Rr	异或	Rd←Rd⊕Rr	0≤d≤31;0≤r≤31	Z,N,V,S	1
COM	Rd	取反	Rd←0xFF-Rd	0≤d≤31	Z,C,N,V,S	1
NEG	Rd	取补	Rd←0x00-Rd	0≤d≤31	Z,C,N,V,H,S	1
SBR	Rd,K	寄存器位置位	Rd←Rd∨K	16≤d≤31;0≤K≤255	Z,N,V,S	1
CBR	Rd,K	寄存器位清零	Rd←Rd •（0xFF-K）	16≤d≤31;0≤K≤255	Z,N,V,S	1
INC	Rd	加 1	Rd←Rd+1	0≤d≤31	Z,N,V,S	1
DEC	Rd	减 1	Rd←Rd-1	0≤d≤31	Z,N,V,S	1
TST	Rd	测寄存器为零或负	Rd←Rd • Rd	0≤d≤31	Z,N,V,S	1
CLR	Rd	寄存器清零	Rd←0x00r	0≤d≤31	Z,N,V,S	1
SER	Rd	寄存器置全 1	Rd←0xFF	16≤d≤31		1
MUL	Rd,Rr	无符号数相乘	R1:R0←Rd x Rr	0≤d≤31;0≤r≤31	Z,C	2
MULS	Rd,Rr	有符号数相乘	R1:R0←Rd x Rr	16≤d≤31;16≤d≤31	Z,C	2
MULSU	Rd,Rr	无与有符号数相乘	R1:R0←Rd x Rr	16≤d≤23;16≤d≤23	Z,C	2

续表

算术和逻辑运算指令（28 条）						
指令	操作数	说明	操作	操作数范围	影响标志	指令周期
FMUL	Rd,Rr	无符号小数相乘	R1:R0←Rd x Rr<<1	16≤d≤23;16≤r≤23	Z,C	2
FMULS	Rd,Rr	有符号小数相乘	R1:R0←Rd x Rr<<1	16≤d≤23;16≤d≤23	Z,C	2
FMULSU	Rd,Rr	无与有符号小数乘	R1:R0←Rd x Rr<<1	16≤d≤23;16≤d≤23	Z,C	2

表 3-2　比较和转移指令

比较和转移指令（36 条）						
指令	操作数	说明	操作	操作数范围	影响标志	指令周期
RJMP	k	相对转移	PC←PC+k+1	-2048≤k≤2047		2
IJMP		间接转移到（Z）	PC←Z			2
JMP	k	直接转移	PC←k	0≤k≤4194303		3
RCALL	k	相对子程序调用	STACK←PC+1, SP←SP-2, PC←PC+k+1	-2048≤k≤2047		3
ICALL		间接子程序调用到（Z）	STACK←PC+1, SP←SP-2,PC←Z			3
CALL	k	直接子程序调用	STACK←PC+2, SP←SP-2,PC←k	0≤k≤65535		4
RET		子程序返回	SP←SP+2, PC←STACK			4
RETI		中断返回	SP←SP+2, PC←STACK		I	4
CPSE	Rd,Rr	比较相等跳行	if（Rd=Rr）PC←PC+2（or 3）	0≤d≤31 0≤r≤31		1/2/3
CP	Rd,Rr	比较	Rd-Rr	0≤d≤31 0≤r≤31	Z,N,V,C,H,S	1
CPC	Rd,Rr	带进位比较	Rd-Rr-C	0≤d≤31 0≤r≤31	Z,N,V,C,H,S	1
CPI	Rd,K	与立即数比较	Rd-K	16≤d≤31 0≤K≤255	Z,N,V,C,H,S	1

续表

比较和转移指令（36 条）						
指令	操作数	说明	操作	操作数范围	影响标志	指令周期
SBRC	Rr,b	寄存器位为 0 跳行	if（Rr（b）=0）PC←PC+2（or 3）	0≤r≤31 0≤b≤7		1/2/3
SBRS	Rr,b	寄存器位为 1 跳行	if（Rr（b）=1）PC←PC+2（or 3）	0≤r≤31 0≤b≤7		1/2/3
SBIC	P,b	I/O 位为 0 跳行	if（P（b）=0）PC←PC+2（or 3）	0≤P≤31 0≤b≤7		1/2/3
SBIS	P,b	I/O 位为 1 跳行	if（P（b）=1）PC←PC+2（or 3）	0≤P≤31 0≤b≤7		1/2/3
BRBS	s,k	SREG（s）位为 1 转移	if（SREG（s））=1 PC←PC+k+1	0≤s≤7 -64≤k≤63		1/2
BRBC	s,k	SREG（s）位为 0 转移	if（SREG（s））=0 PC←PC+k+1	0≤s≤7 -64≤k≤63		1/2
BREQ	k	相等转移	if Z=1 PC←PC+k+1	-64≤k≤63		1/2
BRNE	k	不相等转移	if Z=1 PC←PC+k+1	-64≤k≤63		1/2
BRCS	k	C=1 转移	if C=1 PC←PC+k+1	-64≤k≤63		1/2
BRCC	k	C=0 转移	if C=0 PC←PC+k+1	-64≤k≤63		1/2
BRSH	k	大于等于转移	if C=0 PC←PC+k+1	-64≤k≤63		1/2
BRLO	k	小于转移	if C=1 PC←PC+k+1	-64≤k≤63		1/2
BRMI	k	为负转移	if N=1 PC←PC+k+1	-64≤k≤63		1/2
BRPL	k	为正转移	if N=0 PC←PC+k+1	-64≤k≤63		1/2
BRGE	k	大于等于转移（符号）	if（N⊕V）=0 PC←PC+k+1	-64≤k≤63		1/2

续表

比较和转移指令（36 条）						
指令	操作数	说明	操作	操作数范围	影响标志	指令周期
BRLT	k	小于转移（带符号）	if（N⊕V）=1 PC←PC+k+1	-64≤k≤63		1/2
BRHS	k	半进位标志 H=1 转移	if H=1 PC←PC+k+1	-64≤k≤63		1/2
BRHC	k	半进位标志 H=0 转移	if H=0 PC←PC+k+1	-64≤k≤63		1/2
BRTS	k	标志位 T=1 转移	if T=1 PC←PC+k+1	-64≤k≤63		1/2
BRTC	k	标志位 T=0 转移	if T=0 PC←PC+k+1	-64≤k≤63		1/2
BRVS	k	溢出标志 V=1 转移	if V=1 PC←PC+k+1	-64≤k≤63		1/2
BRVC	k	溢出标志 V=0 转移	if V=0 PC←PC+k+1	-64≤k≤63		1/2
BRIE	k	中断允许位 I=1 转移	if I=1 PC←PC+k+1	-64≤k≤63		1/2
BRID	k	中断允许位 I=0 转移	if I=0 PC←PC+k+1	-64≤k≤63		1/2

表 3-3 数据传送指令

数据传送指令（35 条）						
指令	操作数	说明	操作	操作数范围	影响标志	指令周期
MOV	Rd,Rr	寄存器传送	Rd←Rr	0≤d≤31;0≤r≤31		1
MOVW	Rd,Rr	寄存器字传送	Rd+1:Rd←Rr+1:Rr	d∈{0,2,…30} r∈{0,2,…30}		1
LDI	Rd,K	装入立即数	Rd←K	16≤d≤31;0≤K≤255		1
LD	Rd,X	X 间址取数	Rd←（X）	0≤d≤31		2
LD	Rd,X+	X 间址取数后加 1	Rd←（X），X←X+1	0≤d≤31		2

续表

数据传送指令（35 条）						
指令	操作数	说明	操作	操作数范围	影响标志	指令周期
LD	Rd,-X	X 减 1 后间址取数	X←X-1,Rd←（X）	0≤d≤31		2
LD	Rd,Y	Y 间址取数	Rd←（Y）	0≤d≤31		2
LD	Rd,Y+	Y 间址取数后加 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
LD	Rd,Z	Z 间址取数	Rd←（Z）	0≤d≤31		2
LD	Rd,Z+	Z 间址取数后加 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	Y+q 变址取数	Rd←（Z+q）	0≤d≤31;0≤q≤63		2
LDS	Rd,K	从 SRAM 中取数	Rd←（k）	0≤r≤31;0≤k≤65535		2
ST	X,Rr	X 间址存数	（X）←Rr	0≤r≤31		2
ST	X+,Rr	X 间址存数后加 1	（X）←Rr,X←X+1	0≤r≤31		2
ST	-X,Rr	X 减 1 后间址接存数	X←X-1,（X）←Rr	0≤r≤31		2
ST	Y,Rr	Y 间址存数	（Y）←Rr	0≤r≤31		2
ST	Y+,Rr	Y 间址存数后加 1	（Y）←Rr,Y←Y+1	0≤r≤31		2
ST	Y,Rr	Y 减 1 后间址接存数	Y←Y-1,（Y）←Rr	0≤r≤31		2
STD	Y+q,Rr	Y+q 变址存数	（Y+q）←Rr	0≤r≤31;0≤q≤63		2

续表

数据传送指令（35 条）						
指令	操作数	说明	操作	操作数范围	影响标志	指令周期
ST	Z,Rr	Z 间址存数	（Z）←Rr	0≤r≤31		2
ST	Z+,Rr	Z 间址存数后加 1	（Z）←Rr,Z←Z+1	0≤r≤31		2
ST	–Z,Rr	Z 减 1 后间址接存数	Z←Z-1,（Z）←Rr	0≤r≤31		2
STD	Z+q,Rr	Z+q 变址存数	（Z+q）←Rr	0≤r≤31;0≤q≤63		2
STS	k,Rr	数据送 SRAM	（k）←Rr	0≤r≤31;0≤k≤65535		2
LPM		从程序区取数	R0←（Z）			3
LPM	Rd,Z	从程序区取数	Rd←（Z）	0≤d≤31		3
LPM	Rd,Z+	从程序区取数后 Z 加 1	Rd←（Z），Z←Z+1	0≤d≤31		3
SPM		写数据到程序区	（Z）←R1:R0			-
IN	Rd,P	从 I/O 口读数	Rd←P	0≤d≤31;0≤P≤63		1
OUT	P,Rr	数据送 I/O 口	P←Rr	0≤r≤31;0≤P≤63		1
PUSH	Rr	压栈	STACK←Rr,SP←SP-1	0≤r≤31		2
POP	Rd	出栈	SP←SP+1,Rd←STACK	0≤d≤31		2

表 3-4 位操作和位测试指令

位操作和位测试指令（28 条）						
指令	操作数	说明	操作	操作数范围	影响标志	指令周期
SBI	P,b	置位 I/O 位	I/O（P,b）←1	0≤P≤31;0≤b≤7		2
CBI	P,b	清零 I/O 位	I/O（P,b）←0	0≤P≤31;0≤b≤7		2
LSL	Rd	左移	C←b7←b6…b1←b0←0	0≤d≤31	Z,C,N,V,H	1

续表

位操作和位测试指令（28 条）						
指令	操作数	说明	操作	操作数范围	影响标志	指令周期
LSR	Rd	右移	0→b7→b6…b1→b0→C	0≤d≤31	Z,C,N,V	1
ROL	Rd	带进位左循环	C←b7←b6…b1←b0←C	0≤d≤31	Z,C,N,V,H	1
ROR	Rd	带进位右循环	C→b7→b6…b1→b0→C	0≤d≤31	Z,C,N,V	1
ASR	Rd	算术右移	b7→b7→b6…b1→b0→C	0≤d≤31	Z,C,N,V	1
SWAP	Rd	半字节交换	b7b6b5b4←→b3b2b1b0	0≤d≤31		1
BSET	s	置位 SREG	SREG（s）=1	0≤s≤7	SREG（s）	1
BCLR	s	清零 SREG	SREG（s）=0	0≤s≤7	SREG（s）	1
BST	Rr,b	Rr 的 b 位送 T	T←Rr（b）	0≤r≤31; 0≤b≤7	T	1
BLD	Rd	T 送 Rd 的 b 位	Rd（b）←T	0≤d≤31; 0≤b≤7		1
SEC		置位 C	C←1		C	1
CLC		清零 C	C←0		C	1
SEN		置位 N	N←1		N	1
CLN		清零 N	N←0		N	1
SEZ		置位 Z	Z←1		Z	1
CLZ		清零 Z	Z←0		Z	1
SEI		置位 I	I←1		I	1
CLI		清零 I	I←0		I	1
SES		置位 S	S←1		S	1
CLS		清零 S	S←0		S	1
SEV		置位 V	V←1		V	1
CLV		清零 V	V←0		V	1
SET		置位 T	T←1		T	1
CLT		清零 T	T←0		T	1
SHE		置位 H	H←1		H	1
CLH		清零 H	H←0		H	1

表 3-5　MCU 控制指令

MCU 控制指令（3 条）						
指令	操作数	说明	操作	操作数范围	影响标志	指令周期
NOP		空操作				1
SLEEP		进入休眠	见相关说明			1
WDR		清零看门狗	见相关说明			1

3.1.2　指令系统中使用的符号

在上面的指令表中，汇总给出了 ATmega8 全部指令的汇编助记符、操作数及相应的操作。在指令的描述说明中采用了一些符号代码。下面对所使用符号的意义进行简单的说明。

1．状态寄存器与标志位

SREG：状态寄存器

C：进位标志位

Z：零标志位

N：负数标志位

V：2 的补码溢出标志位

S：N ⊕ V，用于符号测试标志位

H：半进位标志位

T：用于 BLD 指令和 BST 指令传送位

I：全局中断触发/禁止标志位

2．寄存器和操作码

Rd：目的寄存器，取值为 R0～R31 或 R16～R31（取决于指令）。

Rr：源寄存器，取值为 R0～R31。

P：I/O 寄存器，取值为 0～63 或 0～31（取决于指令）。

b：寄存器的指定位，常数（0～7）。

s：状态寄存器 SREG 的指定位，常数（0～7）。

K：立即数，常数（0～255）。

k：地址常数，值范围取决于指令。

q：地址偏移量常数（0～63）。

X、Y、Z：地址指针寄存器（X=R27:R26；Y=R29:R28；Z=R31:R30）。

3．堆栈

STACK：作为返回地址和压栈寄存器的堆栈

SP：堆栈 STACK 的指针

3.1.3 寻址方式和寻址空间

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、程序存储器取常量寻址、程序存储器空间直接寻址、程序存储器空间间接寻址、程序相对寻址等。

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，在指令中给出寄存器的直接地址，这种寻址方式称为单寄存器直接寻址。单寄存器寻址的地址范围限制为通用工作寄存器组中的32个寄存器R0~R31，或后16个寄存器R16~R31（取决于不同指令）。

例：INC Rd；操作：Rd←Rd+1。

INC R5；将寄存器R5内容加1回放。

2．双寄存器直接寻址

双寄存器直接寻址方式与单寄存器直接寻址方式相似，它是将指令指出的两个寄存器Rd和Rr的内容作为操作数，而结果存放在Rd寄存器中。指令中同时给出两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。双寄存器寻址的地址范围限制为通用工作寄存器组中的32个寄存器R0~R31，或后16个寄存器R16~R31，或后8个寄存器R16~R23（取决于不同指令）。

例：ADD Rd，Rr；操作：Rd←Rd+Rr。

ADD R0，R1；将R0和R1寄存器内容相加，结果回放R0。

3．I/O寄存器直接寻址

由指令指出一个I/O寄存器的内容作为操作数。在指令中直接给出I/O寄存器的地址，这种寻址方式称为I/O寄存器直接寻址。I/O寄存器直接寻址的地址使用I/O寄存器空间的地址$00~$3F，共64个，取值为0～63或0～31（取决于指令）。

例：IN Rd，P；操作：Rd←P。

IN R5，$3E；读I/O空间地址为$3E寄存器（SPH）的内容，放入寄存器R5。

4．数据存储器空间直接寻址

数据存储器空间直接寻址方式便于直接从SRAM存储器中存取数据。数据存储器空间直接寻址为双字节指令，在指令的低字节中指出一个16位的SRAM地址。

例：LDS Rd，K；操作：Rd←（K）。

LDS R18，$100；读地址为$100的SRAM中内容，传送到R18中。

指令中16位SRAM的地址字长度限定了SRAM的地址空间为64K字节，该地址空间实际包含了32个通用寄存器和64个I/O寄存器。因此，也可使用数据存储器空间直接寻址的方式读取通用寄存器或I/O寄存器中的内容（需使用寄存器在SRAM空间的映射地

址），但效率比使用寄存器直接寻址的方式要低。原因在于数据存储器空间直接寻址的指令为双字节指令，而且指令周期为 2 个系统时钟。

5．数据存储器空间的寄存器间接寻址

由指令指出某一个 16 位寄存器的内容作为操作数在 SRAM 中的地址，该寻址方式称为数据存储器空间的寄存器间接寻址。AVR 单片机中使用 16 位寄存器 X、Y 或 Z 作为规定的地址指针寄存器，因此操作数的 SRAM 地址在间址寄存器 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。

LD R16，Y；设 Y=$0567，即把 SRAM 地址为$0567 的内容传送到 R16 中。

6．带后增量的数据存储器空间的寄存器间接寻址

这种寻址方式类似于数据存储器空间的寄存器间接寻址方式，间址寄存器 X、Y、Z 中的内容仍为操作数在 SRAM 空间的地址，但指令在间接寻址操作后，再自动把间址寄存器中的内容加 1。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：Rd←（Y），Y=Y+1，先把以 Y 为指针的 SRAM 的内容送 Rd，再把 Y 增 1。

LD R16，Y+；设原 Y=$0567，指令把 SRAM 地址为$0567 的内容传送到 R16 中，再将 Y 的值加 1，操作完成后 Y=$0568。

7．带预减量的数据存储器空间寄存器间接寻址

这种寻址方式类似于数据存储器空间的寄存器间接寻址方式，间址寄存器 X、Y、Z 中的内容仍为操作数在 SRAM 空间的地址，但指令在间接寻址操作之前，先自动将间址寄存器中的内容减 1，然后把减 1 后的内容作为操作数在 SRAM 空间的地址。这种寻址方式也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。

LD R16，-Y；设原 Y=$0567，指令即先把 Y 减 1，Y=$0566，再把 SRAM 地址为$0566 的内容传送到 R16 中。

8．带位移的数据存储器空间寄存器间接寻址

带位移的数据存储器空间寄存器间接寻址方式是：由间址寄存器（Y 或 Z）及指令字中给出的地址偏移量共同决定操作数在 SRAM 空间的地址，偏移量的范围为 0～63。

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。

LDD R16，Y+31；设 Y=$0567，把 SRAM 地址为$0598 的内容传送到 R16 中，Y 寄存器的内容不变。

9．程序存储器空间取常量寻址

程序存储器空间取常量寻址主要从程序存储器 Flash 中取常量，此种寻址方式只用于指令 LPM。程序存储器中常量字节的地址由地址寄存器 Z 的内容确定。Z 寄存器的高 15 位用于选择字地址（程序存储器的存储单元为字），而 Z 寄存器的最低位 Z（d0）用于确定字地址的高/低字节。若 d0=0，则选择字的低字节；d0=1，则选择字的高字节。

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。

例：LPM R16，Z；操作：R16←（Z），即把以 Z 为指针的程序存储器的内容送 R16。

若 Z=$0100，即把地址为$0080 的程序存储器的低字节内容送 R16。

若 Z=$0101，即把地址为$0080 的程序存储器的高字节内容送 R16。

10．带后增量的程序存储器空间取常量寻址

带后增量的程序存储器空间取常量寻址主要从程序存储器 Flash 中取常量，此种寻址方式只用于指令 LPM Rd，Z+。程序存储器中常量字节的地址由地址寄存器 Z 的内容确定。Z 寄存器的高 15 位用于选择字地址（程序存储器的存储单元为字），而 Z 寄存器的最低位 Z（d0）用于确定字地址的高/低字节。若 d0=0，则选择字的低字节；d0=1，则选择字的高字节。寻址操作后，Z 寄存器的内容加 1。

例：LPM R16，Z+；操作：R16←（Z）；Z←Z+1，即把以 Z 为指针的程序存储器的内容送 R16，然后 Z 的内容加 1。

若 Z=$0100，即把地址为$0080 的程序存储器的低字节内容送 R16，完成后 Z=$0101。

若 Z=$0101，即把地址为$0080 的程序存储器的高字节内容送 R16，完成后 Z=$0102。

11．程序存储器空间写数据寻址

程序存储器空间写数据寻址主要用于可进行在系统自编程的 AVR 单片机，此种寻址方式只用于指令 SPM。该指令将寄存器 R1 和 R0 中的内容组成一个字 R1:R0，然后写入由 Z 寄存器的内容作为地址的程序存储器单元中。

例：SPM；操作：（Z）←R1:R0，把 R1:R0 内容写入以 Z 为指针的程序存储器单元。

12．程序存储器空间直接寻址

程序存储器空间直接寻址方式用于程序的无条件转移指令 JMP、CALL。指令中含有一个 16 位的操作数，指令将操作数存入程序计数器 PC 中，作为下一条要执行指令在程序存储器空间的地址。JMP 类指令和 CALL 类指令的寻址方式相同，但 CALL 类的指令还包括了返回地址的压进堆栈和堆栈指针寄存器 SP 内容减 2 的操作。

例：JMP $0100；操作：PC←$0100。程序计数器 PC 的值设置为$0100，接下来执行程序存储器$0100 单元的指令代码。

例：CALL $0100；操作：STACK←PC+2；SP←SP-2；PC←$0100。先将程序计数器 PC 的当前值加 2 后压进堆栈（CALL 指令为 2 个字长），堆栈指针计数器 SP 内容减 2，然后 PC 的值为$0100，接下来执行程序存储器$0100 单元的指令代码。

13．程序存储器空间 Z 寄存器间接寻址

程序存储器空间间接寻址方式是使用 Z 寄存器存放下一步要执行指令代码程序地址，程序转到 Z 寄存器内容所指定程序存储器的地址处继续执行，即用寄存器 Z 的内容代替 PC 的值。此寻址方式用于 IJMP、ICALL 指令。

例：IJMP；操作：PC←Z，即把 Z 的内容送程序计数器 PC。若 Z=$0100，即把$0100 送程序计数器 PC，接下来执行程序存储器$0100 单元的指令代码。

例：ICALL；操作：STACK←PC+1；SP←SP-2；PC←Z。若 Z=$0100，先将程序计数器 PC 的当前值加 1 后压进堆栈，堆栈指针计数器 SP 内容减 2，然后 PC 的值为$0100，接下来执行程序存储器$0100 单元的指令代码。

14．程序存储器空间相对寻址

在程序存储器空间相对寻址方式中，在指令中包含一个相对偏移量 k，指令执行时，首先将当前程序计数器 PC 值加 1 后再与偏移量 k 相加，作为程序下一条要执行指令的地址。此寻址方式用于 RJMP、RCALL 指令。

例：RJMP $0100；操作：PC←PC+1+$0100。若当前指令地址为$0200（PC=$0200），即把$0301 送程序计数器 PC，接下来执行程序存储器$0301 单元的指令代码。

例：RCALL $0100；操作：STACK←PC+1；SP←SP-2；PC←PC+1+$0100。若当前指令地址为$0200（PC=$0200），先将程序计数器 PC 的当前值加 1 后压进堆栈，堆栈指针计数器 SP 内容减 2，然后 PC 的值为$0301，接下来执行程序存储器$0301 单元的指令代码。

15．数据存储器空间堆栈寄存器 SP 间接寻址

数据存储器空间堆栈寄存器 SP 间接寻址是将 16 位的堆栈寄存器 SP 的内容作为操作数在 SRAM 空间的地址，此寻址方式用于 PUSH、POP 指令。

例：PUSH R0；操作：STACK←R0；SP←SP-1。若当前 SP=$045F，先把寄存器 R0 的内容送到 SRAM 的$045F 单元，再将 SP 内容减 1，即 SP=$045E。

例：POP R1；操作：SP←SP+1；R1←STACK。若当前 SP=$045E，先将 SP 内容加 1，再把 SRAM 的$045F 单元内容送到寄存器 R1，此时 SP=$045F。

此外，在 CALL 一类的子程序调用指令和 RET 一类的子程序返回指令中，都隐含着使用堆栈寄存器 SP 间接寻址的方式。

3.2 算术和逻辑指令

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

3.2.1 加法指令

1．不带进位位加法

ADD Rd，Rr　　0≤d≤31，0≤r≤31

说明：两个寄存器不带进位 C 标志相加，结果送目的寄存器 Rd。

操作：Rd←Rd+Rr　　PC←PC+1　　机器码：0000 11rd dddd rrrr

对标志位的影响：H S V N Z C

2．带进位位加法

ADC Rd，Rr　　0≤d≤31，0≤r≤31

说明：两个寄存器和C标志的内容相加，结果送目的寄存器Rd。

操作：Rd←Rd+Rr+C　PC←PC+1　　机器码：0001 11rd dddd rrrr

对标志位的影响：H S V N Z C

3．字加立即数

ADIW Rdl,K　　dl为：24、26、28、30，0≤K≤63

说明：寄存器对（字）同立即数（0~63）相加，结果放到寄存器对。

操作：Rdh:Rdl←Rdh:Rdl+K　　PC←PC+1　　机器码：1001 0110 KKdd KKKK

对标志位的影响：S V N Z C

注意：dl只能取24、26、28、30，即仅用于最后4个寄存器对。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INC指令允许在多倍字长计算中用作循环计数。当对无符号数操作时，仅有BREQ（相等转移）和BRNE（不为零转移）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　　PC←PC+1　　机器码：1001 010d dddd 0011

对标志位的影响：S V N Z

3.2.2　减法指令

1．不带进位位减法

SUB Rd，Rr　　0≤d≤31，0≤r≤31

说明：两个寄存器相减结果送目的寄存器Rd中。

操作：Rd←Rd-Rr　　PC←PC+1　　机器码：0001 10rd dddd rrrr

对标志位的影响：H S V N Z C

2．减立即数（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和Z指针的操作。

操作：Rd←Rd-K　　PC←PC+1　　机器码：0101 KKKK dddd KKKK

对标志位的影响：H S V N Z C

3．带进位位减法

SBC Rd，Rr　　0≤d≤31，0≤r≤31

说明：两个寄存器带着 C 标志相减，结果放到目的寄存器 Rd 中。

操作：Rd←Rd-Rr-C　PC←PC+1　　机器码：0000 10rd dddd rrrr

对标志位的影响：H S V N Z C

4．带进位位减立即数（字节）

SBCI Rd，K　　16≤d≤31，0≤K≤255

说明：寄存器和立即数带着 C 标志相减，结果放到目的寄存器 Rd 中。

操作：Rd←Rd-K-C　PC←PC+1　　机器码：0100 KKKK dddd KKKK

对标志位的影响：H S V N Z C

5．减立即数（字）

SBIW Rdl，K　　dl 为 24、26、28、30，0≤K≤63

说明：寄存器对（字）与立即数 0～63 相减，结果放入寄存器对。

操作：Rdh:Rdl←Rdh:Rdl-K PC←PC+1　机器码：1001 0111 KKdd KKKK

注意：dl 只能取 24、26、28、30，即仅用于最后 4 个寄存器对。K 为 6 位二进制无符号数（0～63）。

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 DEC 指令允许在多倍字长计算中用作循环计数。当对无符号值操作时，仅有 BREQ（不相等转移）和 BRNE（不为零转移）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　PC←PC+1　　机器码：1001 010d dddd 1010

对标志位的影响：S V N Z

3.2.3 取反码指令

COM Rd　　0≤d≤31

说明：该指令完成对寄存器 Rd 的二进制反码操作。

操作：Rd←$FF-Rd　PC←PC+1　　机器码：1001 010d dddd 0000

对标志位的影响：S N Z V（0）C（1）

3.2.4 取补码指令

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　　PC←PC+1　　机器码：1001 010d dddd 0001

对标志位的影响：H S V N Z C

3.2.5 比较指令

1．寄存器比较

CP Rd，Rr　　0≤d≤31，0≤r≤31

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

操作：Rd-Rr　　PC←PC+1　　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　　PC←PC+1　　机器码：0000 01rd dddd rrrr

对标志位的影响：H S V N Z C

3．与立即数比较

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　　PC←PC+1　　机器码：0011 KKKK dddd KKKK

对标志位的影响：H S V N Z C

3.2.6 逻辑与指令

1．寄存器逻辑与

AND Rd，Rr　　0≤d≤31，0≤d≤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　　PC←PC+1　　机器码：0010 00rd dddd rrrr

对标志位的影响：S V（0） N Z

2．与立即数

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　　PC←PC+1　　机器码：0111 KKKK dddd KKKK

对标志位的影响：S N Z V（0）

3．寄存器位清零

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，其结果放在寄存器 Rd 中。

操作：Rd←Rd •（$FF-K）　PC←PC+1　　机器码：0111 $\overline{KKKK}$ dddd $\overline{KKKK}$（$\overline{KKKK}$ 为 KKKK 的补码）

对标志位的影响：S N Z V（0）

4．测试寄存器为零或负

TST Rd　　0≤d≤31

说明：测试寄存器为零或负，实现寄存器内容自己同自己的逻辑与操作，而寄存器内容不改变。

操作：Rd←Rd • Rd　　PC←PC+1　　机器码：0010 00dd dddd dddd

对标志位的影响：S N Z V（0）

3.2.7 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　　PC←PC+1　　机器码：0010 10rd dddd rrrr

对标志位的影响：S N Z V（0）

2．或立即数

ORI Rd，K　　16≤d≤31，　0≤K≤255

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

操作：Rd←Rd∨K　　PC←PC+1　　机器码：0110 KKKK dddd KKKK

对标志位的影响：S N Z V（0）

3．置寄存器位

SBR Rd，K　　16≤d≤31，　0≤K≤255

说明：用于对寄存器 Rd 中指定位置位。完成寄存器 Rd 和常数表征码 K 之间的逻辑或操作，结果送目的寄存器 Rd。

操作：Rd←Rd∨K　　PC←PC+1　　机器码：0110 KKKK dddd KKKK

对标志位的影响：S N Z V（0）

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　　PC←PC+1　　机器码：1110 1111 dddd 1111

对标志位的影响：无

3.2.8　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 的内容逻辑异或操作，结果送目的寄存器 Rd。

操作：Rd←Rd⊕Rr　　PC←PC+1　　机器码：0010 01rd dddd rrrr

对标志位的影响：S N Z V（0）

2．寄存器清零

CLR Rd　　0≤d≤31

说明：寄存器清零。该指令采用寄存器 Rd 与自己的内容相异或实现的寄存器的所有位都被清零。

操作：Rd←Rd⊕Rd　　PC←PC+1　　机器码：0010 01dd dddd dddd

对标志位的影响：S（0）　V（0）　N（0）　Z（1）

3.2.9　乘法指令

1．无符号数乘法

MUL Rd，Rr　　0≤d≤31，　0≤r≤31

说明：该指令完成将寄存器 Rd 和寄存器 Rr 的内容作为两个无符号 8 位数的乘法操作，结果为 16 位的无符号数，保存在 R1:R0 中，R1 为高 8 位，R0 为低 8 位。如果操作数为寄存器 R1 或 R0，则结果会将原操作数覆盖。

操作：R1:R0=Rd×Rr　　PC←PC+1　　机器码：1001 11rd dddd rrrr

对标志位的影响：Z C

2．有符号数乘法

MULS Rd，Rr　　16≤d≤31，　16≤r≤31

说明：该指令完成将寄存器 Rd 和寄存器 Rr 的内容作为两个 8 位有符号数的乘法操作，

结果为 16 位的有符号数，保存在 R1:R0 中，R1 为高 8 位，R0 为低 8 位。源操作数为寄存器 R16~R31。

操作：R1:R0=Rd×Rr　　PC←PC+1　　机器码：0000 0010 dddd rrrr

对标志位的影响：Z C

3．有符号数与无符号数乘法

MULSU Rd，Rr　　16≤d≤23，16≤r≤23

说明：该指令完成将寄存器 Rd（8 位，有符号数）和寄存器 Rr（8 位，无符号数）的内容相乘操作，结果为 16 位的有符号数，保存在 R1:R0 中，R1 为高 8 位，R0 为低 8 位。源操作数为寄存器 R16~R23。

操作：R1:R0=Rd×Rr　　PC←PC+1　　机器码：0000 0011 0ddd 0rrr

对标志位的影响：Z C

4．无符号定点小数乘法

FMUL Rd，Rr　　16≤d≤23，16≤r≤23

说明：该指令完成将寄存器 Rd（8 位无符号数）和寄存器 Rr（8 位无符号数）的内容相乘操作，结果为 16 位的无符号数，并将结果左移一位后保存在 R1:R0 中，R1 为高 8 位，R0 为低 8 位。源操作数为寄存器 R16~R23。

操作：R1:R0=Rd×Rr　（unsigned（1.15）=unsigened（1.7）×unsigened（1.7））

PC←PC+1

机器码：0000 0011 0ddd 1rrr

对标志位的影响：Z C

注：（n.q）表示一个小数点左边有 n 个二进制数位、小数点右边有 q 个二进制数位的小数。以（n1.q1）和（n2.q2）为格式的两个小数相乘，产生格式为（(n1+n2)．(q1+q2)）的结果。对于要有效保留小数位的处理应用，输入的数据通常采用（1.7）的格式，产生的结果为（2.14）格式。因此将结果左移一位，以使高字节的格式与输入的相一致。FMUL 指令的执行周期与 MUL 指令相同，但比 MUL 指令增加了左移操作。

被乘数 Rd 和乘数 Rr 是两个包含无符号定点小数的寄存器，小数点固定在第 7 位和第 6 位之间。结果为 16 位无符号定点小数，其小数点固定在第 15 位和第 14 位之间。

5．有符号定点小数乘法

FMULS Rd，Rr　　16≤d≤23，16≤r≤23

说明：该指令完成寄存器 Rd（8 位带符号数）和寄存器 Rr（8 位带符号数）的内容相乘操作，结果为 16 位的带符号数，并将结果左移一位后保存在 R1:R0 中，R1 为高 8 位，R0 为低 8 位。源操作数为寄存器 R16~R23。

操作：R1:R0=Rd×Rr　（signed（1.15）=sigened（1.7）×sigened（1.7））　PC←PC+1

机器码：0000 0011 1ddd 0rrr

对标志位的影响：Z C

注：（n.q）表示一个小数点左边有 n 个二进制数位、小数点右边有 q 个二进制数位的小数。以（n1.q1）和（n2.q2）为格式的两个小数相乘，产生格式为（(n1+n2)．(q1+q2)）

的结果。对于要有效保留小数位的处理应用，输入的数据通常采用（1.7）的格式，产生的结果为（2.14）格式。因此将结果左移一位，以使高字节的格式与输入的相一致。FMULS 指令的执行周期与 MULS 指令相同，但比 MULS 指令增加了左移操作。

被乘数 Rd 和乘数 Rr 是两个包含带符号定点小数的寄存器，小数点固定在第 7 位和第 6 位之间。结果为 16 位带符号的定点小数，其小数点固定在第 15 位和第 14 位之间。

6．有符号定点小数和无符号定点小数乘法

FMULSU Rd，Rr　　16≤d≤23，　16≤r≤23

说明：该指令完成寄存器 Rd（8 位带符号数）和寄存器 Rr（8 位无符号数）的内容相乘操作，结果为 16 位的带符号数，并将结果左移一位后保存在 R1:R0 中，R1 为高 8 位，R0 为低 8 位。源操作数为寄存器 R16~R23。

操作：R1:R0=Rd×Rr　（signed（1.15）=sigened（1.7）×unsigened（1.7））

机器码：0000 0011 1ddd 1rrr

对标志位的影响：Z C

注：（n.q）表示一个小数点左边有 n 个二进制数位、小数点右边有 q 个二进制数位的小数。以（n1.q1）和（n2.q2）为格式的两个小数相乘，产生格式为（(n1+n2)．(q1+q2)）的结果。对于要有效保留小数位的处理应用，输入的数据通常采用（1.7）的格式，产生的结果为（2.14）格式。因此将结果左移一位，以使高字节的格式与输入的相一致。FMULSU 指令的执行周期与 MULSU 指令相同，但比 MULSU 指令增加了左移操作。

被乘数 Rd 为一个包含带符号定点小数的寄存器，乘数 Rr 是一个包含无符号定点小数的寄存器，小数点固定在第 7 位和第 6 位之间。结果为 16 位带符号的定点小数，其小数点固定在第 15 位和第 14 位之间。

3.3 转 移 指 令

3.3.1 无条件转移指令

1．相对跳转

RJMP k　　-2048≤k≤2047

说明：相对跳转到 PC-2048~PC+2047 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 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，只要使用欲转向的标号即可。

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64K字（128K字节）内跳转。

IJMP间接跳转优点：转移范围大；缺点：作为子程序模块，移植时需修改转移地址，所以一般在子程序中不要使用。

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

对标志位的影响：无

3．直接跳转

JMP k　　　0≤k≤4194303

说明：直接跳转到k地址处，在汇编程序中，用目的地址的标号替代跳转字k。

操作：PC←k　　　机器码：1001 010k kkkk 110k kkkk kkkk kkkk kkkk

对标志位的影响：无

汇编语言中，只要使用欲转向的标号即可。

例：　　JMP ABC

……

ABC：……

3.3.2 条件转移指令

条件转移指令是依照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）状态寄存器中位为“1”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s位，如果该位被置位，则跳转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）状态寄存器中位为“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 s 位，如果该位被清零，则跳转 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 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，测试零标志位 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，将发生转移。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，测试零标志位 Z，如果 Z 位被清零，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）进位标志位 C 为“1”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）进位标志位 C 为“0”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（对无符号数）

BRSH k　　　-64≤k≤63

说明：条件相对转移，测试进位标志 C，如果 C 位被清零，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或等于寄存器 Rr 中无符号二进制数时，将发生转移。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（对无符号数）

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，将发生转移。该指令相当于指令 BRBS 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）结果为负转移

BRMI k　　−64≤k≤63

说明：条件相对转移，测试负号标志 N，如果 N 位被置位，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）结果为正转移

BRPL k　　−64≤k≤63

说明：条件相对转移，测试负号标志 N，如果 N 位被清零，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 位被清零，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，将发生转移。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，将发生转移。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）半进位标志为“1”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 位被置位，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）半进位标志为“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 位被清零，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 标志为“1”转移

BRTS k -64≤k≤63

说明：条件相对转移，测试标志位 T，如果标志位 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 标志为“0”转移

BRTC k -64≤k≤63

说明：条件相对转移，测试 T 标志位，如果标志位 T 被清零，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）溢出标志为“1”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 位被置位，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=l then PC←（PC+l）+k，else PC←PC+l

机器码：1111 00kk kkkk k011

对标志位的影响：无

（18）溢出标志为“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 位被清零，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）中断标志为“1”转移

BRIE k　　-64≤k≤63

说明：条件相对转移，测试全局中断允许标志 I，如果 I 位被置位，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=l 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）中断标志为“0”转移

BRID k　　-64≤k≤63

说明：条件相对转移，测试全局中断允许标志 I，如果 I 位被清零，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

（1）相等跳行

CPSE Rd，Rr　　0≤d≤31，　0≤r≤31

说明：该指令完成两个寄存器 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（or 3）　else PC←PC+1

机器码：0001 00rd dddd rrrr

对标志位的影响：无

（2）寄存器位为“0”跳行

SBRC Rr，b　　0≤r≤31，　0≤b≤7

说明：该指令测试寄存器第 b 位，如果该位被清零，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

（3）寄存器位为“1”跳行

SBRS Rr，b　　0≤r≤31，　0≤b≤7

说明：该指令测试寄存器第 b 位，如果该位被置位，则跳下一行执行指令。

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

机器码：1111 111r rrrr 0bbb

对标志位的影响：无

（4）I/O 寄存器位为“0”跳行

SBIC P，b　　0≤P≤31，　0≤b≤7

说明：该指令测试 I/O 寄存器第 b 位，如果该位被清零，则跳一行执行指令。该指令只在低 32 个 I/O 寄存器内操作，地址为 I/O 寄存器空间的 0～31。

操作：If 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

（5）I/O 寄存器位为“1”跳行

SBIS P，b　　0≤P≤31，0≤b≤7

说明：该指令测试 I/O 寄存器第 b 位，如果该位被置位，则跳一行执行指令。该指令只在低 32 个 I/O 寄存器内操作，地址为 I/O 寄存器空间的 0～31。

操作：If 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

3.3.3 子程序调用和返回指令

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，子程序执行完毕返回时，将断点由堆栈中弹出到 PC，然后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，恢复执行调用子程序前的原程序。

在一个程序中，子程序中还会调用别的子程序，这称为子程序嵌套。每次调用子程序时必须将下条指令地址保存起来，返回时，按后进先出原则依次取出相应的 PC 值。堆栈

就是按后进先出规则存取数据的，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。

1．相对调用

RCALL k　　-2048≤k≤2047

说明：将 PC+1 后的值（RCALL 指令后的下一条指令地址）压入堆栈，然后调用在当前 PC 前或后 k+1 处地址的子程序。

操作：STACK←PC+1，SP←SP-2，PC←（PC+1）+k　机器码：1101 kkkk kkkk kkkk

对标志位的影响：无

2．间接调用

ICALL

说明：间接调用由 Z 寄存器中（16 位指针寄存器）指向的子程序。地址指针寄存器 Z 为 16 位，允许调用在当前程序存储空间 64K 字（128K 字节）内的子程序。

操作：STACK←PC+1，SP←SP-2，PC←Z　　机器码：1001 0101 0000 1001

对标志位的影响：无

3．直接调用

CALL k　　0≤k≤65535

说明：将 PC+2 后的值（CALL 指令后的下一条指令地址）压入堆栈，然后直接调用 k 处地址的子程序。

操作：STACK←PC+2，SP←SP-2，PC←k

机器码：1001 010k kkkk 111k kkkk kkkk kkkk kkkk

对标志位的影响：无

4．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：SP←SP+2，PC←STACK　　机器码：1001 0101 0000 1000

对标志位的影响：无

5．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。

操作：SP←SP+2，PC←STACK　　机器码：1001 0101 0001 1000

对标志位的影响：I（1）

注意：（1）主程序应跳过中断向量区，防止给修改补充中断程序带来麻烦。

（2）应在中断向量区中不用的中断入口地址写上 RETI——中断返回指令，有抗干扰作用。

3.4　数据传送指令

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

所有的传送指令的操作对标志位均无影响。

3.4.1　直接寻址数据传送指令

1．工作寄存器间传送数据

MOV Rd，Rr　　0≤d≤31，0≤r≤31

说明：该指令将一个寄存器内容传送到另一个寄存器中，源寄存器 Rr 的内容不改变，而目的寄存器 Rd 复制了 Rr 的内容。

操作：Rd←Rr　　PC←PC+1　　机器码：0010 11rd dddd rrrr

2．SRAM 数据直接送寄存器

LDS Rd，k　　0≤d≤31，　0≤k≤65535

说明：把 SRAM 中一个存储单元的内容（字节）装入到寄存器，其中 k 为该存储单元的 16 位地址。

操作：Rd←（k）　　PC←PC+2　　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≤65535

说明：将寄存器的内容直接存储到 SRAM 中，其中 k 为存储单元的 16 位地址。

操作：（k）←Rr　　PC←PC+1　　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　　PC←PC+1　　机器码：1110 KKKK dddd KKKK

3.4.2 间接寻址数据传送指令

1．使用 X 指针寄存器间接寻址传送数据

（1）使用地址指针寄存器 X 间接寻址将 SRAM 内容装入到指定寄存器

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

操作：Rd←（X）　　PC←PC+1　　机器码：1001 000d dddd 1100

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

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

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

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

（2）使用地址指针寄存器 X 间接寻址将寄存器内容存储到 SRAM

①ST X，Rr　　0≤d≤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　　PC←PC+1　　机器码：1001 001r rrrr 1100

②ST X，Rr　　0≤d≤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 PC←PC+1　　机器码：1001 001r rrrr 1101

③ST –X，Rr　　0≤d≤31；先将 X 指针减 1，然后将寄存器内容送 X 为指针的 SRAM 中。

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

2．使用 Y 指针寄存器间接寻址传送数据

（1）使用地址指针寄存器 Y 间接寻址将 SRAM 中的内容装入寄存器

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

操作：Rd←（Y）　　PC←PC+1　　机器码：1000 000d dddd 1000

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

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

③LD Rd，-Y　　0≤d≤31；先将 Y 指针减 1，将指针为 Y 的 SRAM 中的数送寄存器。

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

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

操作：Rd←（Y+q）　　PC←PC+1　　机器码：10q0 qq0d dddd 1qqq

（2）使用地址指针寄存器 Y 间接寻址将寄存器内容存储到 SRAM

①ST Y，Rr　　0≤d≤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　　PC←PC+1　　机器码：1000 001r rrrr 1000

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

操作：（Y）←Rr，Y←Y+1 PC←PC+1　　机器码：1001 001r rrrr 1001

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

操作：Y←Y-1，（Y）←Rr　PC←PC+1　　机器码：1001 001r rrrr 1010

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

操作：（Y+q）←Rr　　PC←PC+1　　机器码：10q0 qq1r rrrr 1qqq

3．使用Z指针寄存器间接寻址传送数据

（1）使用地址指针寄存器Z间接寻址将SRAM中的内容装入到指定寄存器

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

操作：Rd←（Z）　　PC←PC+1　　机器码：1000 000d dddd 0000

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

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

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

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

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

操作：Rd←（Z+q）　　PC←PC+1　　机器码：10q0 qq0d dddd 0qqq

（2）使用地址指针寄存器Z间接寻址将寄存器内容存储到SRAM

①ST Z，Rr　　0≤d≤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　　PC←PC+1　　机器码：1000 001r rrrr 0000

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

操作：（Z）←Rr，Z←Z+1　　PC←PC+1　机器码：1001 001r rrrr 0001

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

操作：Z←Z-1，（Z）←Rr　PC←PC+1　　机器码：1001 001r rrrr 0010

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

操作：（Z+q）←Rr　　PC←PC+1　　机器码：10q0 qq1r rrrr 0qqq

以上22条指令操作之后，X、Y、Z指针寄存器要么不改变，要么是加1或减1。使用X、Y、Z指针寄存器的这些特性特别适用于访问矩阵表和堆栈指针等。

3.4.3 从程序存储器中取数装入寄存器指令

1．从程序存储器中取数装入寄存器R0

LPM

说明：将Z指向的程序存储器空间的一个字节装入寄存器R0。

操作：R0←（Z）　　PC←PC+1　　机器码：1001 0101 1100 1000

注释：由于程序存储器的地址是以字（双字节）为单位的，因此，16 位地址指针寄存器 Z 的高 15 位为程序存储器的字地址，最低位 LSB 为“0”时，指字的低字节；为“1”时，指字的高字节。该指令能寻址程序存储器空间为一个 64K 字节的页（32k 字）。

2．从程序存储器中取数装入寄存器

LPM Rd，Z　0≤d≤31

说明：将 Z 指向的程序存储器空间的一个字节装入寄存器 Rd。

操作：Rd←（Z）　　PC←PC+1　　　　机器码：1001 000d dddd 0100

注释：由于程序存储器的地址是以字（双字节）为单位的，因此，16 位地址指针寄存器 Z 的高 15 位为程序存储器的字地址，最低位 LSB 为“0”时，指字的低字节；为“1”时，指字的高字节。该指令能寻址程序存储器空间为一个 64K 字节的页（32k 字）。

3．带后增量的从程序存储器中取数装入寄存器 Rd

LPM Rd，Z+

说明：将 Z 指向的程序存储器空间的一个字节装入 Rd，然后 Z 指针加 1。

操作：Rd←（Z），Z←Z+1 PC←PC+1　　　机器码：1001 000d dddd 0101

注释：由于程序存储器的地址是以字（双字节）为单位的，因此，16 位地址指针寄存器 Z 的高 15 位为程序存储器的字地址，最低位 LSB 为“0”时，指字的低字节；为“1”时，指字的高字节。该指令能寻址程序存储器空间为一个 64K 字节的页（32k 字）。

3.4.4 写程序存储器指令

SPM

说明：将寄存器对 R1:R0 的内容（16 位字）写入 Z 指向的程序存储器空间。

操作：（Z）←R1:R0　　　PC←PC+1　　　机器码：1001 0101 1110 1000

注释：该指令用于具有在应用自编程性能的 AVR 单片机，应用这一特性，单片机系统程序可以在运行中更改程序存储器中的程序，实现动态修改系统程序的功能。由于程序存储器的地址是以字（双字节）为单位的，因此，寄存器 R1:R0 的内容组成一个 16 位的字，其中 R1 为字的高位字节，R0 为字的低位字节。该指令能寻址程序存储器空间为一个 64K 字节的页（32k 字）。具体应用见相关章节的介绍。

3.4.5 I/O 口数据传送

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）的数据传送到寄存器区中的寄存器 Rd 中。

操作：Rd←P　　PC←PC+1　　　　　　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的数据传送到I/O空间（口、定时器、配置寄存器等）。

操作：P←Rr　　PC←PC+1　　机器码：1011 1PPr rrrr PPPP

3.4.6 堆栈操作指令

AVR单片机的特殊功能寄存器中有一个16位的堆栈指针寄存器SP，它指出栈顶的位置，在指令系统中有两条用于数据传送的栈操作指令。

1．进栈指令

PUSH Rr　　0≤d≤31

说明：该指令存储寄存器Rr的内容到堆栈。

操作 STACK←Rr，SP←SP-1　PC←PC+1　　机器码：1001 001d dddd 1111

2．出栈指令

POP Rd　　0≤d≤31

说明：该指令将堆栈中的字节装入到寄存器Rd中。

操作：SP←SP+1，Rd←STACK PC←PC+1　　机器码：1001 000d dddd 1111

3.5 位操作和位测试指令

AVR单片机指令系统中有四分之一的指令为位操作和位测试指令，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。

3.5.1 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位被清零，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：$C \leftarrow b_7b_6b_5b_4b_3b_2b_1b_0 \leftarrow 0$　PC←PC+1　　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位被清零，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：$0 \rightarrow b_7b_6b_5b_4b_3b_2b_1b_0 \rightarrow C$　PC←PC+1　　机器码：1001 010d dddd 0110

对标志位的影响：S V N（0） Z C

3．带进位位的寄存器逻辑循环左移

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：$C \leftarrow b_7b_6b_5b_4b_3b_2b_1b_0 \leftarrow C$　PC←PC+1　　机器码：0001 11dd dddd dddd（参见 ADC 指令）

对标志位的影响：H S V N Z C

4．带进位位的寄存器逻辑循环右移

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：$C \rightarrow b_7b_6b_5b_4b_3b_2b_1b_0 \rightarrow C$　PC←PC+1　　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，而第 7 位保持原逻辑值，第 0 位被装入 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：$b_7 \rightarrow b_7b_6b_5b_4b_3b_2b_1b_0 \rightarrow C$　PC←PC+1　　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：$b_7b_6b_5b_4 \leftrightarrow b_3b_2b_1b_0$　PC←PC+1　　机器码：1001 010d dddd 0010

对标志位的影响：无

3.5.2 位变量传送指令

1．寄存器中的位存储到 SREG 中的 T 标志

BST Rr，b　　0≤d≤31，0≤b≤7

说明：把寄存器 Rr 中的位 b 存储到 SREG 状态寄存器中的 T 标志位中。

操作：T←Rr（b）　　PC←PC+1　　机器码：1111 101d dddd 0bbb

对标志位的影响：T

2．SREG 中的 T 标志位值装入寄存器 Rd 中的某一位

BLD Rd，d　　0≤d≤31，0≤b≤7

说明：复制 SREG 状态寄存器的 T 标志到寄存器 Rd 中的位 b。

操作：Rd（b）←T　　PC←PC+1　　机器码：1111 100d dddd 0bbb

对标志位的影响：无

3.5.3 位变量修改指令

1．状态寄存器 SREG 的指定位置“1”

BSET s　　0≤s≤7

说明：置“1”状态寄存器 SREG 的某一标志位。

操作：SREG（s）←1　　PC←PC+1　　机器码：1001 0100 0sss 1000

对标志位的影响：I T H S V N Z C

2．状态寄存器 SREG 的指定位清“0”

BCLR s　　0≤s≤7

说明：清零 SREG 状态寄存器 SREG 中的一个标志位。

操作：SREG（s）←0　　PC←PC+1　　机器码：1001 0100 1sss 1000

对标志位的影响：I T H S V N Z C

3．I/O 寄存器的指定位置“1”

SBI P，b　　0≤P≤31，0≤b≤7

说明：对 P 指定的 I/O 寄存器的指定位置“1”。该指令只在低 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　　PC←PC+1　　机器码：1001 1010 PPPP Pbbb

对标志位的影响：无

4．I/O 寄存器的指定位置“0”

CBI P，b　　0≤P≤31，0≤b≤7

说明：清零指定 I/O 寄存器中的指定位。该指令只用在低 32 个 I/O 寄存器上操作，I/O 寄存器地址为 0~31。

操作：I/O（P，b）←0　　PC←PC+1　　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　　PC←PC+1　　机器码：1001 0100 0000 1000

对标志位的影响：C（1）

6．清进位位

CLC　　清零 SREG 状态寄存器中的进位标志 C

操作：C←0　　PC←PC+1　　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　　PC←PC+1　　机器码：1001 0100 0010 1000

对标志位的影响：N（1）

8．清负标志位

CLN　　清零 SREG 状态寄存器中的负数标志 N

操作：N←0　　PC←PC+1　　机器码：1001 0100 1010 1000

对标志位的影响：N（0）

9．置零标志位

SEZ　　置位 SREG 状态寄存器中的零标志 Z

操作：Z←1　　PC←PC+1　　机器码：1001 0100 0001 1000

对标志位的影响：Z（1）

10．清零标志位

CLZ　　清零 SREG 状态寄存器中的零标志 Z

操作：Z←0　　PC←PC+1　　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　　PC←PC+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　　PC←PC+1　　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　　PC←PC+1　　机器码：1001 0100 0100 1000

对标志位的影响：S（1）

14．清 S 标志位

CLS　　清零 SREG 状态寄存器中的符号标志 S

操作：S←0　　PC←PC+1　　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　　PC←PC+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　　清零 SREG 状态寄存器中的溢出标志 V

操作：V←0　　PC←PC+1　　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　　PC←PC+1　　机器码：1001 0100 0110 1000

对标志位的影响：T（1）

18. 清 T 标志位

CLT　　清零 SREG 状态寄存器中的 T 标志

操作：T←0　　PC←PC+1　　机器码：1001 0100 1110 1000

对标志位的影响：T（0）

19. 置半进位标志

SHE　　置位 SREG 状态寄存器中的半进位标志 H

操作：H←1　　PC←PC+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　　清零 SREG 状态寄存器中的半进位标志 H

操作：H←0　　PC←PC+1　　机器码：1001 0100 1101 1000

对标志位的影响：H（0）

3.6 MCU 控制指令

MCU 控制指令有 3 条，主要用于控制 MCU 的运行方式以及清零看门狗定时器。

1. 空操作指令

NOP

说明：该指令完成一个单周期空操作。

操作：无　　PC←PC+1　　机器码：0000 0000 0000 0000

对标志位的影响：无

应用：延时等待、产生方波。抗干扰处理：在空的程序存储器单元中写上空操作，空操作指令最后加一跳转指令，转到$000H。

2. 进入休眠方式指令

SLEEP

说明：该指令使 MCU 进入休眠方式运行。休眠模式由 MCU 控制寄存器定义。当 MCU 在休眠状态下由一个中断被唤醒时，在中断程序执行后，紧跟在休眠指令后的指令将被

执行。

操作：PC←PC+1　　MCU 进入由 MCU 控制寄存器定义的休眠方式运行

机器码：1001 0101 1000 1000

应用：省电，尤其对绿色、便携式仪器特别有用。

对标志位的影响：无

3．清零看门狗计数器

WDR

说明：该指令清零看门狗定时器。在允许使用看门狗定时器情况下，系统程序在正常运行中必须在 WD 预定比例器给出限定时间内执行一次该指令，以防止看门狗定时器溢出，造成系统复位。参见看门狗定时器硬件部分。

操作：清零看门狗定时器　　PC←PC+1　　机器码：1001 0101 1010 1000

对标志位的影响：无

应用：抗干扰、延时、提高系统的稳定性。

3.7 AVR 汇编语言系统

汇编语言是一种符号化语言，它使用助记符（特定的英文字符）来代替实际的二进制机器指令代码。例如用 ADD 表示“加”，用 MOV 表示传送等，本章就是以汇编形式描述 ATmega8 的指令系统。

用汇编语言编写的程序称为汇编语言程序，或称源程序。显然汇编语言源程序比二进制的机器语言更容易学习和掌握。但是，单片机不能直接识别和执行汇编语言程序，因此需要使用一个专用的软件系统，将汇编语言的源程序“翻译”成二进制的机器语言程序——目标程序（执行代码）。这个专用软件系统就是汇编语言编译软件。

ATMEL 公司提供免费的 AVR 开发平台——AVR Studio 集成开发环境（IDE），其中就包括 AVR Assembler 汇编编译器。

3.7.1 汇编语言语句格式

汇编语言源程序是由一系列汇编语句组成的。汇编语言语句的标准格式有以下 4 种：

（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

标号

标号是语句地址的标记符号，用于引导对该语句的访问和定位。使用标号的目的是为了跳转和转移指令及在程序存储器、数据存储器SRAM以及E²PROM中定义变量名。有关标号的一些规定如下：

（1）标号一般由ASCII字符组成，第一个字符为字母；

（2）同一标号在一个独立的程序中只能定义一次；

（3）不能使用汇编语言中已定义的符号（保留字），如指令字、寄存器名、伪指令字等。

伪指令

在汇编语言程序中可以使用一些伪指令。伪指令并不产生实际的目标机器操作代码，只是用于在汇编程序中对地址、寄存器、数据、常量等进行定义说明，以及对编译过程进行某种控制等。AVR指令系统不包括伪指令，伪指令通常由汇编编译系统给出。

指令

指令是汇编程序中主要的部分，汇编程序中使用指令集中给出的全部指令。

操作数

操作数是指令操作时所需要的数据或地址。汇编程序完全支持指令系统所定义的操作数格式。但指令系统采用的操作数格式通常为数字形式，在编写程序时使用起来不太方便，因此，在编译器的支持下，可以使用多种形式的操作数，如数字、标识符、表达式等。

注释

注释部分仅用于对程序和语句进行说明，帮助程序设计人员阅读、理解和修改程序。只要有“;”符号，后面即为注释内容，注释内容长度不限，注释内容换行时，开头部分还要使用符号“;”。编译系统对注释内容不予理会，不产生任何代码。

分隔符

汇编语句中，“:”用于标号之后；空格用于指令字和操作数的分隔；指令有两个操作数时用“,”分隔两个操作数；“;”用于注释之前；“[ ]”中的内容表示可选项。

3.7.2 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生成操作执行代码，而是用于调整存储器中程序的位置、定义宏、初始化存储器、对编译过程进行某种控制等。全部伪指令在表3-6中给出。

表3-6 伪指令表

序号	伪指令	说明	序号	伪指令	说明
1	BYTE	定义预留存储单元	10	ESEG	E²PROM段
2	CSEG	代码段	11	EXIT	退出文件
3	DB	定义字节常数	12	INCLUDE	包含指定的文件
4	DEF	定义寄存器符号名	13	LIST	列表文件生成允许器

续表

序　号	伪 指 令	说　明	序　号	伪 指 令	说　明
5	DEVICE	指定为何器件生成汇编代码	14	LISTMAC	列表宏表达式
6	DSEG	数据段	15	MACRO	宏定义开始
7	DW	定义字常数	16	NOLIST	关闭列表文件生成
8	ENDMACRO	宏结束	17	ORG	设置程序起始位置
9	EQU	定义标识符常量	18	SET	赋值给标识符

1．BYTE——定义预留存储单元

BYTE 伪指令是从指定的地址开始，在 SRAM 中保留若干字节的存储空间备用。备用存储空间以字节计算，个数由 BYTE 伪指令的参数，即表达式的值确定。BYTE 伪指令前应使用一个标号，以标记备用存储空间在 SRAM 中的起始位置。该伪指令有一个参数，表示保留存储空间的字节数。BYTE 伪指令仅能用在数据段内（见伪指令 CSEG，DSEG 和 ESEG）。注意：BYTE 伪指令必须带一个参数，字节数的位置不需要初始化。

语法：LABEL：.BYTE 表达式

示例：

```
.DSEG                        ;数据段(SRAM)
var1:   .BYTE 1              ;保留 1 个字节的存储单元，用 var1 标识
table:  .BYTE tab_size       ;保留 tab_size 个字节的存储空间

.CSEG                        ;代码段开始(Flash)
    ldi r30, low(var1)       ;将保留存储单元 var1 起始地址的低 8 位装入 Z
ldi r31, high(var1)          ;将保留存储单元 var1 起始地址的高 8 位装入 Z
ld r1, Z                     ;将保留存储单元的内容读到寄存器 R1
```

2．CSEG——代码段

CSEG 伪指令定义代码段的起始（在 Flash 中）。一个汇编程序可包含几个代码段，这些代码段在编译过程中被连接成一个代码段。在代码段中不能使用 BYTE 伪指令。程序中缺省的段为代码段。每个代码段内部都有自己的字定位计数器。可使用 ORG 伪指令定义该字定位计数器的初始值，作为代码段在程序存储器中的起始位置。CSEG 伪指令不带参数。

语法：.CSEG

3．DB——在程序存储器或 E^2PROM 存储器中定义字节常数

DB 伪指令是从程序存储器或 E^2PROM 存储器的某个地址单元开始，存入一组规定的 8 位二进制常数（字节常数）。DB 伪指令只能出现在代码段或 E^2PROM 段。DB 伪指令前应使用一个标号，以标记所定义的字节常数区域的起始位置。DB 伪指令为一个表达式列表，表达式列表由多个表达式组成，但至少要含有一个表达式，表达式之间用逗号分隔。每个

表达式值的范围必须在-128～255之间。如果表达式的值是负数，则用8位2的补码表示，存入程序存储器或E^2PROM存储器中。如果DB伪指令用在代码段，并且表达式表中多于一个表达式，则以两个字节组合成一个字放在程序存储器中。如果表达式的个数是奇数，不管下一行汇编代码是否仍是DB伪指令，最后一个表达式的值将单独以字的格式放在程序存储器中。

语法：LABEL：.DB 表达式表

4．DEF——定义寄存器符号名

DEF伪指令给寄存器定义一个替代的符号名。在后序程序中可以使用定义的符号名来表示被定义的寄存器。可以给一个寄存器定义多个符号名。符号名在后面程序中可以重新定义指定。

语法：.DEF 符号名 = 寄存器

5．DEVICE——指定为何器件生成汇编代码

DEVICE 伪指令允许用户告知汇编器为何器件编译产生执行代码。如果在程序中使用该伪指令指定了器件型号，那么在编译过程中，若存在指定器件所不支持的指令，编译器则给出一个警告。如果代码段或E^2PROM段所使用的存储器空间大于指定器件本身所能提供的存储器容量，编译器也会给出警告。如果不使用DEVICE伪指令，则假定器件支持所有的指令，也不限制存储器容量的大小。

语法：.DEVICE ATmega8

6．DSEG——数据段

DSEG 伪指令定义数据段的起始。一个汇编程序文件可以包含几个数据段，这些数据段在汇编过程中被连接成一个数据段。在数据段中，通常是仅由BYTE伪指令（和标号）组成。每个数据段内部都有自己的字节定位计数器。可使用ORG伪指令定义该字节定位计数器的初始值，作为数据段在SRAM中的起始位置。DSEG伪指令不带参数。

语法：.DSEG

7．DW——在程序存储器或E^2PROM存储器中定义字常数

DW伪指令是从程序存储器或E^2PROM存储器的某个地址单元开始，存入一组规定的16位二进制常数（字常数）。DW伪指令只能出现在代码段或E^2PROM段。DW伪指令前应使用一个标号，以标记所定义的字常数区域的起始位置。DW伪指令为一个表达式列表，表达式列表由多个表达式组成，但至少要含有一个表达式，表达式之间用逗号分隔。每个表达式值的范围必须在-32768～65535之间。如果表达式的值是负数，则用16位2的补码表示。

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——定义标识符常量

EQU 伪指令将表达式的值赋给一个标识符，该标识符为一个常量标识符，可以用于后面的表达式中，但该标识符的值不能改变或重新定义。

语法：.EQU 标号 = 表达式

10．ESEG——E^2PROM 段

ESEG 伪指令定义 E^2PROM 段的开始。一个汇编文件可以包含几个 E^2PROM 段，这些 E^2PROM 段在汇编编译过程中被连接成一个 E^2PROM 段。在 E^2PROM 段中不能使用 BYTE 伪指令。每个 E^2PROM 段内部都有自己的字节定位计数器。可使用 ORG 伪指令定义该字节定位计数器的初始值，作为数据段在 E^2PROM 中的起始位置。ESEG 伪指令不带参数。

语法：.ESEG

11．EXIT——退出文件

EXIT 伪指令告诉汇编编译器停止汇编该文件。在正常情况下，汇编编译器的编译过程一直到文件的结束。如果 EXIT 出现在包含文件中，则汇编编译器将结束对包含文件的编译，然后从本文件当前 INCLUDE 伪指令的下一行语句处开始继续编译。

语法：.EXIT

12．INCLUDE——包含指定的文件

INCLUDE 伪指令告诉汇编编译器开始从一个指定的文件中读入程序语句，并对读入的语句进行编译，直到该包含文件结束或遇到该文件中的 EXIT 伪指令，然后再从本文件当前 INCLUDE 伪指令的下一行语句处继续开始编译。在一个包含文件中，也可以使用 INCLUDE 伪指令来包含另外一个指定的文件。

语法：.INCLUDE “文件名”

13．LIST——打开列表文件生成器

LIST 伪指令告诉汇编编译器打开列表文件生成器。正常情况下，汇编编译器在编译过程中将生成一个由汇编源代码、地址和操作码组成的列表文件。默认时为允许生成列表文件。该伪指令可以与 NOLIST 伪指令配合使用，以选择某一部分的汇编源文件产生列表文件。

语法：.LIST

14．LISTMAC——列表宏表达式

LISTMAC 伪指令告诉汇编编译器，在生成的列表文件中，显示所调用宏的表达式。默认情况下，仅在列表文件中显示所调用的宏名和参数。

语法：.LISTMAC

15．MACRO——宏开始

MACRO 伪指令告诉汇编器一个宏程序的开始。MACRO 伪指令将宏程序名作为参数。当后面的程序中使用宏程序名，则表示在该处调用宏程序。一个宏程序中可带 10 个参数。这些参数在宏定义中用@0～@9 代表。当调用一个宏程序时，参数用逗号分隔。伪指令 ENDMACRO 定义宏程序的结束。

默认情况下，在汇编编译器生成的列表文件中仅给出宏的调用。如要在列表文件中给出宏的表达式，则必须使用 LISTMAC 伪指令。在列表文件的操作码域中，宏带有 a+ 的记号。

语法：.MACRO 宏名

示例：

```
.MACRO SUBI16                ;宏定义开始
    subi @1,low(@0)          ;减少字节
    sbci @2,high(@0)         ;减高字节
.ENDMACRO                    ;宏定义结束

.CSEG                        ;代码段开始
    SUBI16 0x1234,r16,r17    ;r17:r16 = r17:r16 - 0x1234
```

16. NOLIST——关闭列表文件生成器

NOLIST 伪指令告诉汇编编译器关闭列表文件生成器。正常情况下，汇编编译器在编译过程中将生成一个由汇编源代码、地址和操作码组成的列表文件，默认情况下为允许生成列表文件。可以使用该伪指令禁止文件列表的产生。该伪指令可以与LIST伪指令配合使用，以选择某一部分的汇编源文件产生列表文件。

语法：.NOLIST

17. ORG——定义代码起始位置

ORG 伪指令设置定位计数器为一个绝对数值，该数值为表达式的值，作为代码的起始位置。如果ORG伪指令出现数据段中，则设定SRAM定位计数器；如果该伪指令出现在代码段中，则设定程序存储器计数器；如果该伪指令出现在E^2PROM段中，则设定E^2PROM定位计数器。如果该伪指令前带标号（在相同的语句行），则标号的定位由ORG的参数值定义。代码段和E^2PROM段定位计数器的默认值是零；而当汇编器启动时，SRAM定位计数器的默认值是32（因为寄存器占有地址为0~31）。注意，E^2PROM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——设置标识符与一个表达式值相等

SET 伪指令将表达式的值赋值给一个标识符。这个标识符能用在后面的表达式中。用SET伪指令赋值的标识符能在后面的程序中重新设置改变。

语法：.SET 标号 = 表达式

3.7.3 表达式

在标准指令系统中，操作数通常只能使用纯数字格式，这给程序的编写带来了许多不便。但是在编译系统的支持下，在编写汇编程序时允许使用表达式，以方便程序的编写。AVR编译器支持的表达式是由操作数、函数和运算符组成。所有的表达式内部都是32位的。

操作数

操作数有以下几种形式：

（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：

- 十进制数（默认），如：10、255；
- 十六进制数，如：0x0a、$0a、0xff、$ff；
- 二进制数，如：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）　　返回一个表达式值的低字节。

（2）HIGH（表达式）　　返回一个表达式值的第二个字节。

（3）BYTE2（表达式）　　与 HIGH 函数相同。

（4）BYTE3（表达式）　　返回一个表达式值的第三个字节。

（5）BYTE4（表达式）　　返回一个表达式值的第四个字节。

（6）LWRD（表达式）　　返回一个表达式值的 0～15 位。

（7）HWRD（表达式）　　返回一个表达式值的 16～31 位。

（8）PAGE（表达式）　　返回一个表达式值的 16～21 位。

（9）EXP2（表达式）　　返回（表达式值）2 次幂的值。

（10）LOG2（表达式）　　返回 Log_2（表达式值）的整数部分。

3．运算符

汇编器提供的部分运算符见表 3-7。优先级数越高的运算符，其优先级也越高。表达式可以用小括号括起来，并且与括号外其他任意的表达式再组合成表达式。

表 3-7　部分运算符表

序　号	运　算　符	名　称	优　先　级	说　明
1	!	逻辑非	14	一元运算符，表达式是 0 返回 1，表达式是 1 返回 0
2	~	逐位非	14	一元运算符，将表达式的值按位取反
3	-	负号	14	一元运算符，使表达式为算术负
4	*	乘法	13	二进制运算符，两个表达式相乘
5	/	除法	13	二进制运算符，左边表达式除以右边表达式，得整数的商值
6	+	加法	12	二进制运算符，两个表达式相加
7	−	减法	12	二进制运算符，左边表达式减去右边表达式
8	<<	左移	11	二进制运算符，左边表达式值左移右边表达式给出的次数

续表

序　号	运 算 符	名　称	优 先 级	说　明
9	>>	右移	11	二进制运算符，左边表达式值右移右边表达式给出的次数
10	<	小于	10	二进制运算符，左边带符号表达式值小于右边带符号表达式值，则为 1，否则为 0
11	<=	小于等于	10	二进制运算符，左边带符号表达式值小于或等于右边带符号表达式值，则为 1，否则为 0
12	>	大于	10	二进制运算符，左边带符号表达式值大于右边带符号表达式值，则为 1，否则为 0
13	>=	大于等于	10	二进制运算符，左边带符号表达式值大于或等于右边带符号表达式值，则为 1，否则为 0
14	==	等于	9	二进制运算符，左边带符号表达式值等于右边带符号表达式值，则为 1，否则为 0
15	!=	不等于	9	二进制运算符，左边带符号表达式值不等于右边带符号表达式值，则为 1，否则为 0
16	&	逐位与	8	二进制运算符，两个表达式值之间逐位与
17	^	逐位异或	7	二进制运算符，两个表达式值之间逐位异或
18	|	逐位或	6	二进制运算符，两个表达式值之间逐位或
19	&&	逻辑与	5	二进制运算符，两个表达式值之间逻辑与，全非 0 则为 1，否则为 0
20	||	逻辑或	4	二进制运算符，两个表达式值之间逻辑或，非 0 则为 1，全 0 为 0

3.7.4　文件“m8def.inc”

在编写 AVR 汇编程序时，在程序的开始处应使用伪指令“.INCLUDE”引用编译系统中的器件标识定义文件“****def.inc”。该文件已将该器件所有的 I/O 寄存器、标志位等进行了标称化的符号定义，这些标称化的符号与硬件结构的命名是相同的。这样在程序中就可直接使用标称化的符号，而不必去记住它的实际地址。如使用 PORTB 来代替 B 口数据寄存器的地址$18。读者可具体查看相关器件的定义文件。下面是一个标准的汇编程序的开始部分：

```
.INCLUDE "M8def.inc"          ;引用器件 I/O 配置文件
.DEF TEMP1 = r20              ;定义标识符 TEMP1 代表工作寄存器 R20
......

.ORG  $0000                   ;代码段起始定位
rjmp RESET                    ;系统上电复位，跳转到主程序
```

```
.ORG  $0013                          ;代码段定位，跳过中断向量区
                                     ;初始化，设置 ATmega8 的堆栈指针为$045F
RESET:ldi r16,high(RAMEND)           ;RAMEND 在配置文件"M8def.inc"中已定义为$045F
        out SPH,r16                  ;将 RAMEND 的高位送堆栈寄存器 SP 高位字节中
    ldi r16,low(RAMEND)
     out SPL,r16                     ;将 RAMEND 的低位送堆栈寄存器 SP 低位字节中

        ser temp1                    ;将 temp1 即寄存器 R20 置为$FF
out DDRD,temp1                       ;R20 值送 DDRD，D 口方向寄存器为$FF,设定为输出
......
```

在上面的程序段中使用的 RAMEND、SPH、SPL、DDRD 均在文件“M8def.inc”中进行了定义，分别为：

```
.EQU SPH = $3E
.EQU SPL = $3D
......
......
.EQU DDRD = $11
......
......
.EQU RAMEND = $45F
......
......
```

第 4 章 ATmega8 开发工具

4.1 AVR STUDIO（AVR 集成开发环境）

ATMEL 开发的 AVR STUDIO3 软件是一个用于开发 AVR 系列单片机的集成工作环境。该软件有以下功能：汇编程序汇编器、模拟仿真功能、实时仿真功能（需仿真机配合）、AVR Prog 串行编程、STK500/AVRISP/JTAG ICE 编程等功能。AVR STUDIO 支持 ATMEL 全系列仿真器。AVR STUDIO 安装后，从菜单：开始→程序→ATMEL AVR TOOLS→AVR STUDIO3.55 打开 AVR STUDIO，可以看到如图 4.1 所示的程序界面。

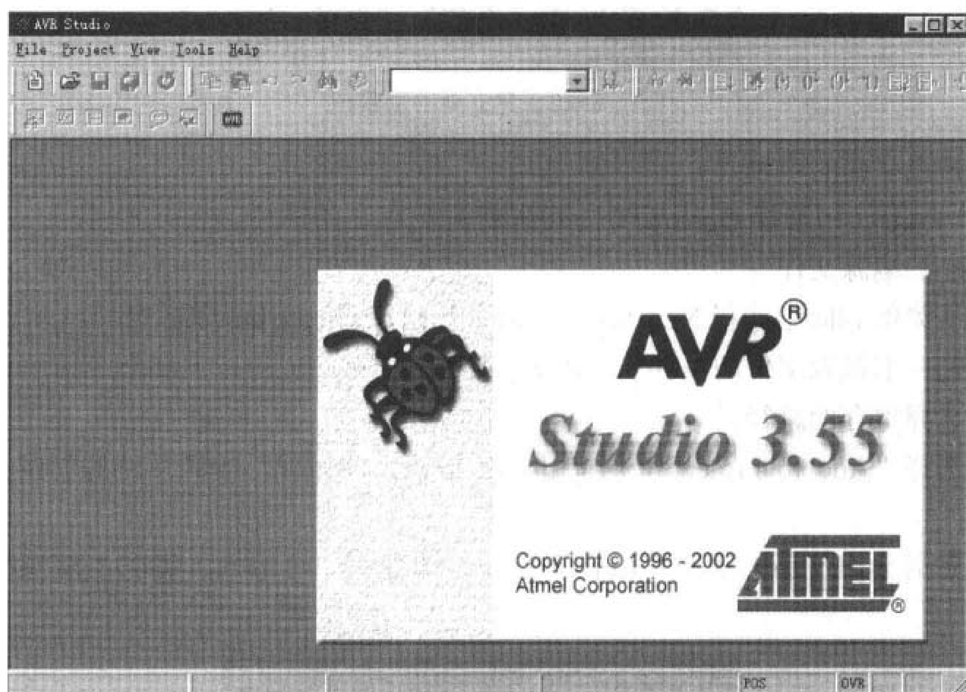


图 4.1 AVR STUDIO3 程序界面

4.1.1 汇编程序汇编器（AVR Assembler）

在 AVR STUDIO 中可以完成所有的操作，如建立项目、建立和编辑汇编源文件、汇编（可以生成 OBJ 和 HEX 文件）、调试、程序下载等，一气呵成，十分方便。

1．建立新的设计项目

（1）从菜单 Project 中选择 New 命令，出现 Select new project 窗口；

（2）输入项目名称；

（3）选择项目保存的路径；

（4）用鼠标选中 AVR Assembler；

（5）再单击 OK 按钮即完成项目的新建，如图 4.2 所示。

图 4.2 建立新的项目

2．打开已存在的项目

从菜单 Project 中选择 Open 命令，可以打开已存在的项目。

3．新建汇编源文件

（1）从菜单 File 中选择 New text file 命令，出现 Create new file 窗口；

（2）输入汇编源文件的文件名，如 SL.ASM；

（3）选择保存的路径；

（4）选择“Add to Project”，再单击 OK 按钮，即完成汇编源文件的创建并自动添加到项目中。

（5）然后会出现新的源文件编辑窗口，在该窗口中可输入、编辑汇编源程序，如图 4.3 所示。

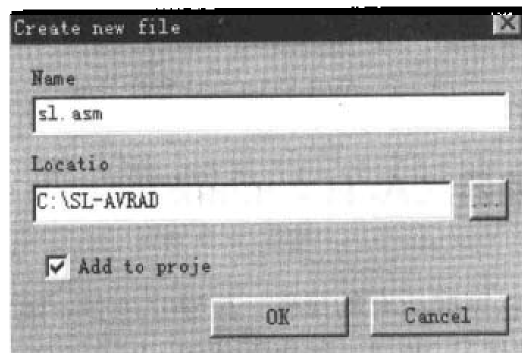


图 4.3 新建汇编源文件

4. 打开已存在的汇编源文件

从菜单 File 中选择 Open 命令，可以打开已存在的汇编文件。

5. 将已存在的汇编源文件加入到当前打开的项目中

从 Project 菜单中选择 Add File...命令，可以将已经存在的汇编源文件加入到一个项目中去。

6. 项目编译设定

（1）选择 Project 菜单；

（2）选择 Project Settings 命令，出现 AVR Assembler Options 窗口；

（3）选择 Output file 中的 Intel Intellec8/MDS...生成 Intel hex 格式文件；

（4）单击 OK 按钮完成项目设定，如图 4.4 所示。

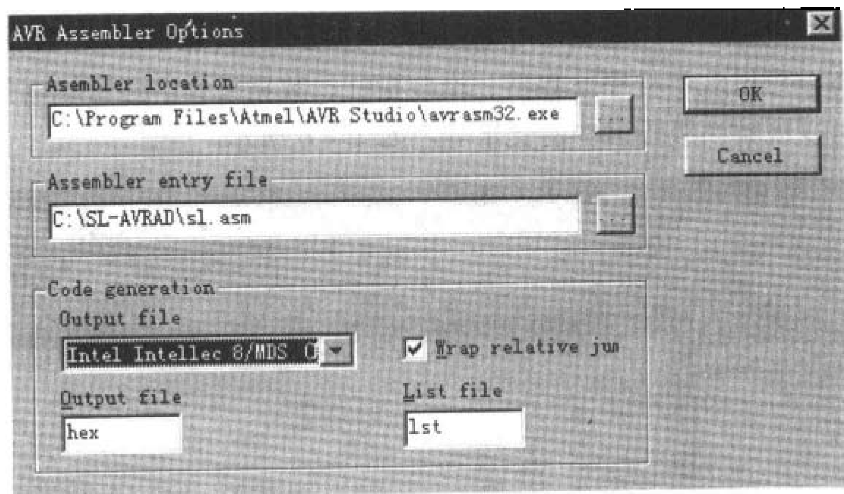


图 4.4　项目编译设定

注意：如果不做上述第 3 步操作的话，汇编器不生成烧写芯片用的 INTEL HEX 文件。

7. 汇编和运行

从菜单 Project 中选择 Assemble 命令只对项目进行汇编，但不调试运行；如果选择 Build and Run 命令，则在汇编完成后，如果没有错误则立刻进入调试运行状态。汇编的结果会在图 4.5 所示的窗口中显示。

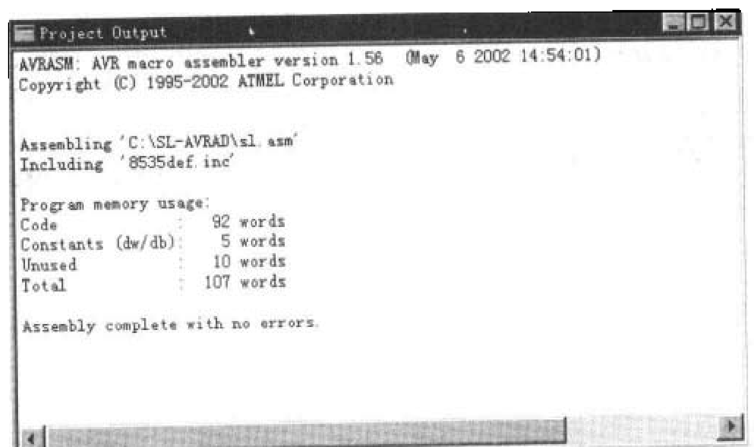


图 4.5　汇编结果显示

4.1.2 仿真调试

在 AVR STUDIO 中选择 Build and Run 命令，在汇编完成后如果没有错误，就自动进入调试运行状态。对在其他环境中生成的调试文件，只要使用 Open 命令打开相应的调试文件（如 COFF、D90 和 HEX 等文件）就可进入调试运行状态。

AVR STUDIO3 对仿真器的识别是自动的，如果仿真机连接正常，则进入实时仿真状态，否则进入模拟调试状态。如果 AVR STUDIO 打开的是 HEX 文件，则以反汇编的形式进行仿真调试。如果打开的是 COFF 等调试文件，则以源代码形式进行调试，AVR STUDIO 支持 C 语言程序源代码级的调试。

在调试时可以设置断点，观察通用寄存器、RAM 数据、E^2PROM 数据、IO 空间的寄存器和端口状态，可以观察程序计数器、堆栈指针、数据指针、运行机器周期数等重要处理器状态，也可以观察高级语言中的变量等，如图 4.6 所示。

图 4.6 仿真调试窗口

4.2 AVR 单片机 C 编译器——ICCAVR 的使用

C 语言目前已成为设计嵌入式系统的标准语言，它既有普通高级语言的结构化编程、可读性好、维护方便的特点，又具有汇编等低级语言对硬件访问的方便、代码效率高的特点，因此十分适用于嵌入式系统的程序设计。

ICCAVR 是美国 ImageCraft 公司推出的一种 C 编译器，主要用于开发 ATMEL 公司的 AVR 系列单片机。ICCAVR 是一种用 ANSI 标准 C 语言开发微控制器的一种程序设计语言，其集成开发环境（IDE）除了集成项目管理、程序编辑、程序编译、错误提示等常用的功能以外，还包含了函数浏览、应用程序向导、ISP 下载工具和 AVR 资源配置计算器等方便实用的工具。无论对初学者，还是对一个专业工作者来讲，ICCAVR 都是一款价廉物美的开发工具。注意，ICCAVR 只适合于开发带有 RAM 存储器的 AVR 单片机，对不带 RAM 存储器的 AVR 单片机应该使用 ImageCraft 公司推出的另一种 C 编译器——ICCTINY。

4.2.1 ICCAVR 编译器的安装

1．运行光盘上的 SETUP.EXE 程序进行安装。

方法一：（1）打开“我的电脑”；

（2）打开光盘驱动器所对应的盘符；

（3）双击光盘中文件“SETUP.EXE”的图标；

（4）按照屏幕提示，选定一个安装路径后进行安装。

方法二：（1）在“开始”菜单中选择运行项目；

（2）在“运行”对话框中填入“drive:\setup.exe”

（注意 drive 对应机器中的光盘驱动器盘符）；

（3）按“确定”键开始安装；

（4）其余同方法一。

2．对安装完成的软件进行注册

对首次安装并且使用期未超过 30 天的用户，可按下列步骤注册，如图 4.7 所示：

- 启动 ICC AVR 编译器的集成开发环境（IDE）。
- 将软件附带的一张名为“Unlock Disk”的软盘插入机器的软盘驱动器中。
- 在 IDE 的“Help”菜单中寻找标题为“Importing a License from a Floppy Disk”的选项，并且单击。
- 然后按照提示进行软件注册，当注册完成后会提示注册文件已从软盘移走。当确定并再次重新启动 ICCAVR 后，软件即完成注册。

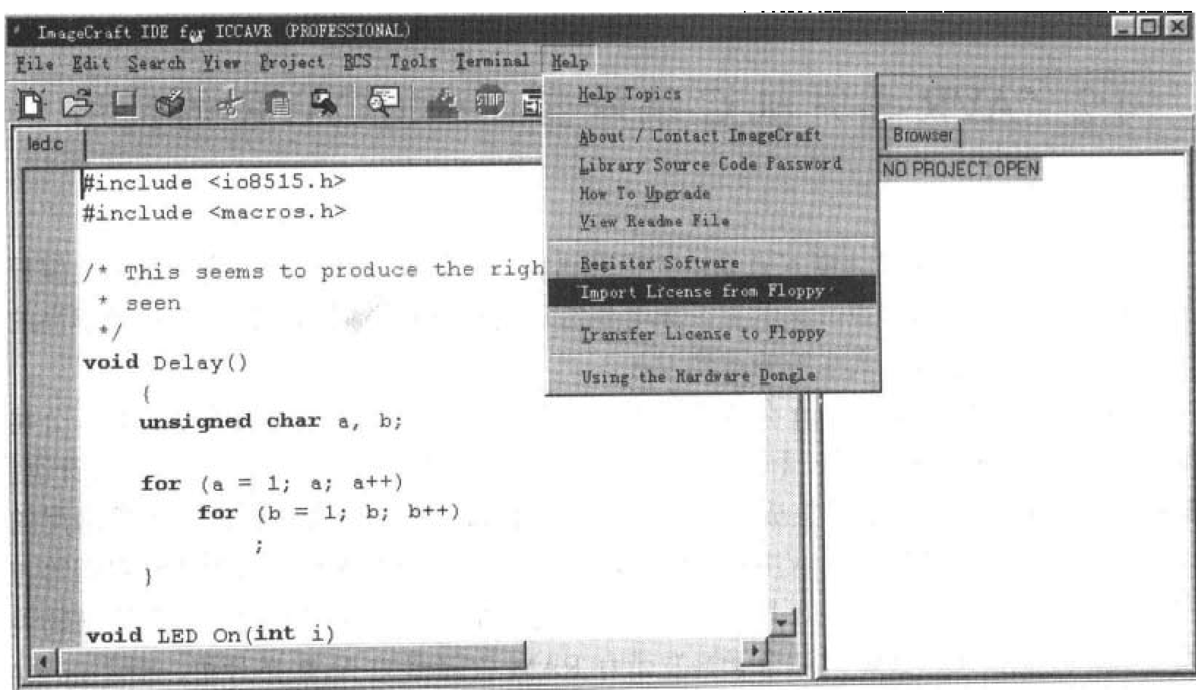


图 4.7 ICCAVR 工作窗口

对不是首次安装或使用时间已超过 30 天的用户，可按下列步骤注册：

- 对这类用户在程序启动时已不能进入 IDE 环境，而是出现一个提示注册的对话框，此时应该选择“YES”按钮。
- 出现一个注册对话框，该对话框上有一个标题为“Importing a License from a Floppy Disk”的按钮。
- 将正式版中附带的一张名为“Unlock Disk”的软盘插入软盘驱动器中，单击上一步中提到的按钮。
- ICCAVR 软件自动进行注册，当注册完成后会提示注册文件已从软盘移走。在确定并再次重新启动 ICCAVR 后，软件即已经完成注册。

注意：

- “Unlock Disk”软盘在注册时应打开写保护，否则无法完成注册。
- 完成注册后，“Unlock Disk”软盘成为一张空盘，不能在另一台机器上进行安装和注册。
- 当需要在不同的电脑中使用 ICCAVR 或在同一台电脑中将 ICCAVR 重新安装在与原来不同的路径时，应该首先在“Help”菜单中选择“Transferring Your License to a Floppy Disk”选项，将注册文件传送到一张软盘上，然后再按上述方法进行安装注册。

4.2.2 ICCAVR 介绍

1．ImageCraft 的 ICCAVR 介绍

ImageCraft 的 ICCAVR 是一种使用符合 ANSI 标准的 C 语言来开发微控制器（MCU）

程序的一个工具，它有以下几个主要特点：

ICCAVR 是一个综合了编辑器和项目管理器的集成开发环境（IDE），其可在 Windows9X/NT 下工作。

源文件全部被组织到项目之中，文件的编辑和项目的编译也在这个环境中完成。编译错误显示在状态窗口中，并且当用鼠标单击编译错误时，光标会自动跳转到编辑窗口中引起错误的那一行。这个项目管理器还能直接产生希望得到的、可以直接使用的 INTEL HEX 格式文件。INTEL HEX 格式文件可被大多数的编程器所支持，用于下载程序到芯片中去。

ICCAVR 是一个 32 位的程序，支持长文件名。

由于篇幅所限，本书并不介绍通用的 C 语言语法知识，仅介绍使用 ICC AVR 所必须具备的知识，因此要求读者在阅读本书内容之前，应对 C 语言有一定程度的了解。

2．ICCAVR 中的文件类型及其扩展名

文件类型是由它们的扩展名决定的，IDE 和编译器可以使用以下几种类型的文件。

输入文件：

.c　扩展名——表示是 C 语言源文件

.s　扩展名——表示是汇编语言源文件

.h　扩展名——表示是 C 语言的头文件

.prj　扩展名——表示是项目文件，这个文件保存由 IDE 所创建和修改的一个项目的有关信息。

.a　扩展名——库文件，它可以由几个库封装在一起。libcavr.a 是一个包含了标准 C 的库和 AVR 特殊程序调用的基本库。如果库被引用，链接器会将其链接到模块或文件中。用户也可以创建或修改一个符合需要的库。

输出文件：

.s　对应每个 C 语言源文件，由编译器在编译时产生的汇编输出文件。

.o　由汇编文件汇编产生的目标文件，多个目标文件可以链接成一个可执行文件。

.hex　INTEL HEX 格式文件，其中包含了程序的机器代码。

.eep　INTEL HEX 格式文件，其中包含了 E^2PROM 的初始化数据。

.cof　COFF 格式输出文件，用于在 ATMEL 的 Avr Studio 环境下进行程序调试。

.lst　列表文件，在这个文件中列出了目标代码对应的最终地址。

.mp　内存映像文件，它包含了程序中有关符号及其所占内存大小的信息。

.mak　项目编译时使用的 make 文件。

3．附注和扩充

#pragma（编译附注）

这个编译器接受以下附注：

（1）#pragma interrupt_handler <func1>:<vector number> <func2>:<vector> ...

这个附注必须在函数之前定义，它说明函数 func1、func2 是中断操作函数，所以编译器在中断操作函数中生成中断返回指令 reti 来代替普通返回指令 ret，并且保存和恢复函数所使用的全部寄存器；同样编译器根据中断向量号 vector number 生成中断向量地址。

（2）#pragma ctask <func1> <func2>...

这个附注指定了函数不生成挥发寄存器来保存和恢复代码，它的典型应用是在 RTOS 实时操作系统中让 RTOS 核直接管理寄存器。

（3）#pragma text:<name>

改变代码段名称，使其与命令行选项相适应。

（4）#pragma data:<data>

改变数据段名称，使其与命令行选项相适应。这个附注在分配全局变量至 E^2PROM 中时必须被使用，读者可参考访问 EEPROM 的例子。

（5）#pragma abs_address:<address>

函数与全局数据不使用浮动定位（重定位），而是从<address>开始分配绝对地址。这在访问中断向量和其他硬件项目时特别有用。

（6）#pragma end_abs_address

结束绝对定位，使目标程序使用正常浮动定位。

（7）C++ 注释

如果选择了编译扩充(Project->Options->Compiler)，可以在源代码中使用 C++ 的 // 类型的注释。

（8）二进制常数

如果选择了编译扩充（Project->Options->Compiler），可以使用 0b<1|0>* 来指定二进制常数，例如，0b10101 等于十进制数 21。

（9）在线汇编

可以使用 asm（"string"）函数来指定在线汇编代码，读者可参考在线汇编。

4.2.3 ICCAVR 导游

1．起步

自启动 IDE 后，首先从 Project 菜单中选择 Open 命令，进入\icc\examples.avr 目录并且选择打开“led”项目，项目管理器显示在这个项目中只有一个文件 led.c。然后从 Project 菜单中选择 Options 命令打开项目编译选项，在“Target”标号下选择目标处理器。然后从 Project 菜单中选择 Make Project 命令，IDE 将调用编译器编译这个项目文件，并且在状态窗口中显示所有的信息。

如果没有错误，在与源文件同一个目录（在这个例子中是\icc\examples.avr）中输出一个文件 led.hex。这个文件是 INTEL HEX 格式，大多数支持 AVR MCU 的编译器和模拟器都支持这种格式，并且能下载这个程序进入目标系统，这样就完成了一个程序的编译。

如果希望用支持 COFF（公用目标文件格式）调试信息的工具来测试程序，比如 AVR Studio，那么需要从 Project 菜单中选择 Options 命令，在编译选项卡下选择 COFF 输出文件格式。对于一些常用的功能，也可使用工具条或单击鼠标右键弹出菜单。例如，可以在项

目窗口单击鼠标右键选择编译选项。

在项目窗口中双击文件名，IDE 将使用编辑器打开这个文件。按这个方法打开 led.c，作为试验可设置一些错误，例如从一行中删除分号“;”。现在从 Project 菜单中选择 Make Project 命令，IDE 首先自动保存已经改变的文件，并且编译这个文件。这时在状态窗口中会显示错误信息，单击状态窗口中错误信息行，或单击其左边的错误符号，光标将移到编辑器中错误行的下面一行上（基本上所有 C 编译器都是这样）。

开始一个新的项目

从 Project 菜单中选择 New 命令，并且浏览至希望输出项目文件的目录，输出文件的名称取决于项目文件名称。例如，如果创建一个名为 foo.prj 的项目，那么输出文件名为 foo.hex 或 foo.cof 等。

自从创建自己的项目后，用户可以开始写源代码（C 或汇编格式），并且将这个文件加入到项目文件中。单击工具栏中“Build”图标，可以很容易地编译这个项目，IDE 输出与 ATMEL 的 AVR Studio 完全兼容的 COFF 格式文件，可以使用 ATMEL 的 AVR Studio 来调试代码。

为了更容易地使用这个开发工具，可以通过应用程序向导来生成一些使用有关硬件的初始化代码。

2. 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，因此也可以在目标系统中执行它。启动代码的用途在启动文件中很详细地描述了，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，是无限循环地运行。看看 \icc\examples 目录中的文件 led.c。

```
#include <io8515.h>
  /* 为了能够看清 LED 的变化图案，延时程序需要有足够的延时时间*/
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()4
    {
    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);
         }
      }
```

主 main 例程是很简单的，在初始化一些 IO 寄存器后之后，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 IO 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。因为延时的实际延时值不能确定，这一对嵌套循环只能给出延时的近似延时时间；如果这个实际定时时间非常重要，那么这个例程应该使用硬件定时器来完成延时。

其他的例子如 8515intr.c 程序很简单，但同样清楚地显示了如何用 C 语言写一个中断处理过程，这两个例子可以作为程序的起点。

4.2.4 ICCAVR 的 IDE 环境

1. 编译一个单独的文件

正常建立一个输出文件的次序是，首先建立一个项目文件并且定义属于这个项目的所有文件。然而，有时也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。这时可以这样操作：从 IDE 菜单“File”中选择“Compile File...”命令，来执行“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，并且是在编辑窗口中可

以编辑的。

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。注意：这里使用默认的编译选项。

2. 创建一个新的项目

为创建一个新的项目，从菜单“Project”中选择“New”命令，IDE 会弹出一个对话框，在该对话框中可以指定项目的名称，这也是输出文件的名称。如果使用一些已经建立的源文件，可在菜单“Project”中选择“AddFile（s）”命令。

另外，可以在菜单“File”中选择“New”命令来建立一个新的源文件来输入代码，可以在菜单“File”中选择“Save”或“Save As”命令来保存文件。然后可以像上面所述调用“AddFile(s)”命令将文件加入到项目中，也可在当前编辑窗口中单击鼠标右键选择“Add to Project”将文件加入已打开的项目列表中。通常输出源文件在项目的同一个目录中，但也可不作这样要求。

项目的编译选项使用菜单中“Project”中的“Options”命令。

3. 项目管理

项目管理允许将多个文件组织进同一个项目，而且定义它们的编译选项，这个特性允许将项目分解成许多小的模块。当编译整个项目时，如果只有一个文件被修改则只重新编译该文件；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项目列表中，例如可以将一个项目文档文件放在项目管理窗口中，项目管理器在编译项目时对源文件以外的文件不予理睬。

对目标器件不同的项目，可以在编译选项中设置有关参数。当新建一个项目时，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项目中。默认编译选项保存在 default.prj 文件中。

为避免项目目录混乱，可以指定输出文件和中间文件到一个指定的目录。通常这个目录是项目目录的一个子目录。

4. 编辑窗口

编辑窗口是用户与 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，用鼠标单击有关错误信息时，编辑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，编辑器定位于其下面一行。

5. 应用程序向导

应用程序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”命令来调用它。

应用程序向导使用编译选项中指定的目标 MCU 来产生相应的选项和代码。

应用程序向导显示目标 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、IO 端口、UART、SPI 和模拟量比较

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．超级终端

IDE 有一个内置的超级终端，注意它不包含任意一个 ISP（在线编程）功能，但它可以作为一个简单的调试终端，或许可以显示目标装置的调试信息，也可下载一个 ASCII 码文件。

8．ISP 编程，ICCAVR 支持多种 ISP 下载电缆和 STK500 起步工具。

4.2.5 C 库函数与启动文件

1．启动文件

这个链接器会自动将启动文件连接到用户的程序之前，并将标准库 libcavr.a 与程序相连接。启动文件，根据目标 MCU 的不同，在 crtavr.o 和 crtatmega.o 中间任意选择一个。启动文件定义了一个全局符号__start，它也是程序的起点。启动文件的功能有：

（1）初始化硬件和软件堆栈指针。

（2）从 idata 区复制初始化数据到直接寻址数据区 data 区。

（3）将 bss 区全部初始化为零。

（4）调用用户主例程 main 函数。

（5）定义一个退出点，如果主函数 main()一旦退出，它将进入这个退出点进行无限循环。

启动文件也定义了复位向量，用户不需要修改启动文件来使用别的中断，详情可参考中断操作部分。

为修改和使用新的启动文件：

cd \icc\libsrc.avr ；进入安装的编译器路径

<edit crtavr.s> ；编辑修改 crtavr.s 文件

<open crtavr.s using the IDE> ；用 IDE 打开 crtavr.s 文件

<Choose "Compile File To->Object"> ；选择编译到目标文件，创建一个新的 crtavr.o

copy crtavr.o ..\lib ；复制到库目录

如果使用的目标 MCU 是 ATMEGA 系列 MCU，应该用“crtatmega”代替“crtavr”，注意，ATMEGA 系列 MCU 的每个中断入口向量占用两个字（word），而非 Mega 芯片每一个中断入口向量占用一个字（word）。

也可以有多个启动文件，可以在项目选项对话框中很方便地直接指定一个启动文件加入项目中。注意，必须指定启动文件的绝对路径或启动文件必须位于项目选项中库文件所指向的路径中。

2．常用库函数

（1）库源代码

这个库源代码(默认路径为 c:\icc\libsrc.avr\libsrc.zip)是一个密码保护的 ZIP 压缩文件，可以从互联网上任意下载一个 UNZIP 程序进行解压缩。当本软件被开锁后，密码显示在“About”对话框中。例如：

unzip -s libsrc.zip

; unzip 提示输入密码

（2）AVR 专用函数——ICCAVR 有许多访问 UART、E^2PROM 和 SPI 的函数，堆栈检查函数对检测堆栈是否溢出很有用。另外我们的互联网上有一个网页专门存放用户写的源代码。

（3）io*.h （io8515.h,io8515v.h, iom8.h,iom128v.h，… 等）

这些文件是从 ATMEL 官方公开的定义 IO 寄存器的源文件经过修改而得到的，应该用这些文件来代替原有的 avr.h 文件。通过这些定义，C 语言可以访问一些硬件 IO 寄存器，如：

PORTB = 1;

uc = PORTA;

（4）macros.h

这个文件包含了许多有用的宏和定义。

（5）其他头文件

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，那么包含头文件是一个好习惯，在使用浮点数和长整型数的程序中必须用 #include 预编译指令包含这些函数原形的头文件。

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

stdarg.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

3. 字符类型库

下列函数按照输入的 ASC II 字符集字符分类，使用这些函数之前应当包含“ctype.h”头文件。

int isalnum（int c）

如果 c 是数字或字母返回非零数值，否则返回零。

int isalpha（int c）

如果 c 是字母返回非零数值，否则返回零。

int iscntrl（int c）

如果 c 是控制字符（如 FF、BELL、LF…等）返回非零数值，否则返回零。

int isdigit（int c）

如果 c 是数字返回非零数值，否则返回零。

int isgraph（int c）

如果 c 是一个可打印字符而非空格返回非零数值，否则返回零。

int islower（int c）

如果 c 是小写字母返回非零数值，否则返回零。

int isprint（int c）

如果 c 是一个可打印字符返回非零数值，否则返回零。

int ispunct（int c）

如果 c 是一个可打印字符而不是空格、数字或字母返回非零数值，否则返回零。

int isspace（int c）

如果 c 是一个空格字符返回非零数值，包括空格 CR、FF、HT、NL 和 VT，否则返回零。

int isupper（int c）

如果 c 是大写字母返回非零数值，否则返回零。

int isxdigit（int c）

如果 c 是十六进制数字返回非零数值，否则返回零。

int tolower（int c）

如果 c 是大写字母则返回 c 对应的小写字母，其他类型仍然返回 c。

int toupper（int c）

如果 c 是小写字母则返回 c 对应的大写字母，其他类型仍然返回 c。

4．浮点运算库

下列函数支持浮点数运算，使用这些函数之前必须包含“math.h”头文件。

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-π~+π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 的双曲余弦函数值。

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）和以2为底的指数n两个部分，即$x=y\times 2^n$，y的范围为$0.5\leqslant y<1$，y值被函数返回，而n值存放到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$x\times 2^{exp}$。

float log（float x）

返回x的自然对数。

float log10（float x）

返回以10为底的x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。

5. 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，不过有一些 IO 函数是受支持的，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。最底层的 IO 程序是单字符的输入（getchar）和输出（putchar）程序，如果针对不同的装置使用高层的 IO 函数，例如用 printf 输出 LCD，需要全部重新定义最底层的函数。

为在 ATMEL 的 AVR Studio 模拟器（模拟超级终端窗口）使用标准 IO 函数，应当在编译选项中选中“AVR Studio Simulator IO”检查框。

注意：默认状态下，单字符输出函数 putchar()是输出到 UART 设备的，如果想用模拟器的超级终端显示输出字符，则‘\n’ 字符必须被映射为成对的回车和换行（CR/LF）。

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 frm 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——除了大写字母使用'A'~'F'外，同 %x。

%u——输出无符号十进制整数。

%s——输出一个以 C 语言中的空字符 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 存储器中的字符串常量。

printf 支持三个版本，取决于用户的特殊需要和代码空间的大小（越高级的版本，代码需求越大）：

基本型：只有 %c、%d、%x、%u 和 %s 格式说明符是允许的。

长整型：针对长整形数的 %ld、%lu、%lx 格式符被支持，以适用于精度要求较高的领域。

浮点型：全部格式包括%f 格式被支持。

可以使用编译选项对话框来选择不同的 PRINTF 版本，但要注意程序代码的增加。

int putchar(int c)

输出单个字符，使用 UART 以查寻方式输出单个字符，注意输出至超级终端窗口必须有'\n'字符。

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 frm 字符串到一个缓冲区，格式说明符同 printf( ) 函数。

6. 标准库和内存分配函数

标准库头文件<stdlib.h>定义了宏 NULL 和 RAND_MAX 和新定义的类型 size_t，并且描述了下列函数。注意，在调用任意内存分配程序（比如.. calloc、malloc 和 realloc）之前，必须调用_NewHeap 来初始化堆 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，否则返回 0。

void *calloc（size_t nelem, size_t size）

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。如果分配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，它是担任用户 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 字节大小的堆

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 所指向内存区的大小为 size 字节，size 可比原来大或小，返回指向该内存区的地址指针。

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）

除了返回类型为无符号长整型数外，其余同“strtol”。

7．字符串函数

用“#include <string.h>”预处理后，编译器支持下列函数。<string.h>定义了 NULL、类型 size_t 和下列字符串及字符阵列函数。

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，如果相同则返回 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 *memcpy（void *s1, void *s2, size_t n）

复制 s2 中 n 个字符至 s1，但复制区不可以重叠。

void *memmove（void *s1, void *s2, size_t n）

复制 s2 中 n 个字符至 s1，返回 s1，其与 memcpy 基本相同，但复制区可以重叠。

void *memset（void *s, int c, size_t n）

在 s 中填充 n 个字节的 c，它返回 s。

char *strcat（char *s1, char *s2）

复制 s2 到 s1 的结尾，返回 s1。

char *strchr（char *s, int c）

在 s1 中搜索第一个出现的 c，包括结束 NULL 字符。如果成功返回指向匹配字符的指针；如果没有找到的匹配字符，返回空指针。

int strcmp（char *s1, char *s2）

比较两个字符串，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）

复制字符串 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 匹配的第一个字符，包括结束 NULL 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，不包括结束 NULL 字符。

char *strncat（char *s1, char *s2, size_t n）

复制字符串 s2（不含结束 NULL 字符）中 n 个字符到 s1，如果 s2 长度比 n 小，则只复制 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，但其只复制前 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 不匹配的第一个字符，包括结束 NULL 字符，其返回 s1 中找到的第一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，它返回 s1 中匹配子字符串的地址指针；否则返回 NULL。

8．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。注意那个类型必须是高级类型，如 int、 long 或 double，小的整型类型如“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 vfprintf()来实现

```
#include <stdarg.h>
int printf(char *fmt, ...)
    {
    va_list ap;
va_start(ap, fmt);
    vfprintf(fmt, ap);
    va_end(ap);
    }
```

9．堆栈检查函数

有几个库函数是用于检查堆栈是否溢出，内存图如图 4.8 所示。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被改变，也就是说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，如果函数调用层次太深，则可能会发生这种情况。

同样，软件堆栈溢出进数据区域将会改变全局变量或其他静态分配的项目（如果使用

动态分配内存，还会改变堆项目)。这种情况在定义了太多的局部变量或一个局部集合变量太大的情况下也会偶然发生。

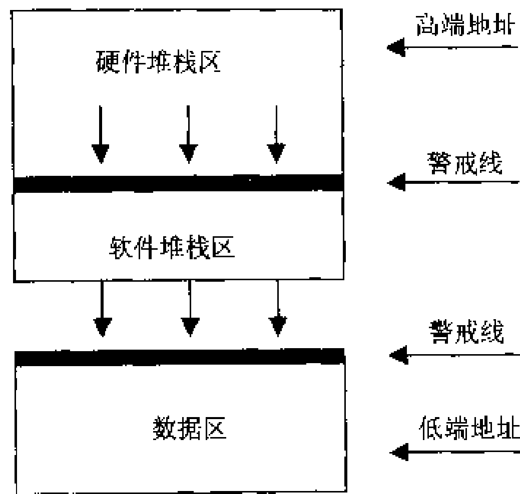


图 4.8 警戒线

标准的 C 启动代码会写一个标志字节到数据区和软件堆栈区的警戒线地址，作为堆栈溢出检测的依据。

注意：如果使用动态分配内存，必须跳过警戒线字节_bss_end 来分配堆，参考内存分配函数。

堆栈检查

可以调用_StackCheck(void)函数来检查堆栈溢出，如果警戒线标志字节仍然保持正确的值，那么意味着堆栈没有溢出；反之，如果堆栈溢出，那么警戒线标志字节将被破坏。

注意：当程序堆栈溢出的时候，程序可能会运行不正常或崩溃。当_StackCheck 检查错误条件时，它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。在 ICCAVR 的例子中安排了两个功能调用，它是两个堆栈都可能溢出的。无论如何，在_StackOverflowed 执行起作用时，第二个调用不可以出现，如果函数复位了 CPU，那么将不能返回_StackCheck 函数。

默认的 _StackOverflowed 函数

当它被调用时，库会用一个默认的_StackOverflowed 函数来跳转到 0 的位置，这样就复位了 CPU 和程序。用户也可能希望用一个函数来代替它，以指示更多的错误条件，因为它可能会切断所有的中断并且点亮 LED。注意，如果堆栈溢出指示程序出现故障，_StackOverflowed 函数可能不能执行太复杂的操作或实现程序的正常工作。

这两个函数的原型在头文件 macros.h 中。

4.2.6 访问 AVR 硬件的编程

1．访问 AVR 底层硬件

AVR 系列 MCU 使用高级语言编程时有很高的 C 语言密度，它允许对 MCU 的底层硬件进行访问。除非是为了最大程度地优化代码或对时序要求特别严格，否则没有必要使用汇编。ICCAVR 也提供了在线汇编和宏定义的来处理硬件寄存器的方法。

头文件 io*.h（如 io8515.h、iom603.h 等）定义了指定 AVR MCU 的 IO 寄存器细节。这些文件是从 ATMEL 官方发布的文件，经过修改，以匹配这个编译器的语法要求。文件 macros.h 定义了许多有用的宏，例如宏 UART_TRANSMIT_ON()能使 UART 开始工作。

此编译器的效率很高，当访问由 IO 寄存器映射的内存时能产生单周期指令，如 in、out、sbis、sbi 等。

注意：原来的头文件 avr.h 定义 IO 寄存器的位有一些模糊，虽然 io*.h 文件定义了各种 IO 寄存器的位，但很多时候仍然需要使用定义在 macros.h 文件中的 BIT()宏。例如：

```
avr.h:
#define SRE     0x80      // 外部 RAM 使能
... (C 程序)
MCUCR |= SRE;
io8515.h:
#define SRE     7
  ... (C 程序)
  #include <macros.h>
  MCUCR |= BIT(SRE);    // 外部 RAM 使能
```

2．位操作

嵌入式系统编程一个共同的任务是打开或关闭 IO 寄存器的一些位（位操作），幸运的是标准 ANSI C 提供了比较强大的位运算功能，而不需要借助于汇编指令或其他非标准 ANSI C 的语法，从而使代码的兼容性更好。

a | b——按位或运算，表示 a 和 b 按位进行逻辑或运算，这个逻辑运算常用于将某些位置 1，尤其经常用“|=”的形式，例如：

```
PORTA |= 0x80;//将最高位置 1
```

a & b——按位与运算，表示 a 和 b 按位进行逻辑与运算，这个逻辑运算在检查某些位是否为 1 或将某些置为 0 时很有用，例如：

```
If ((PORTA & 0x81) == 0);          // 检查位 7 和位 0
PORTB&=0x7f;                       // 将 PORTB 的最高位清零
```

注意：&运算符的优先级比== 运算符低，因此在某些表达式中要使用括号将&运算括起来，这是初学 C 语言程序设计中常犯的一个错误。

a ^ b——按位异或运算，表示 a 和 b 按位进行逻辑异或运算，这个逻辑运算通常用来对一个位取反。例如，在下面的例子中，位 7 是被翻转的：

```
PORTA ^= 0x80; // 翻转位 7
```

~a——按位取反运算，表示对 a 按位进行逻辑非运算，当用&运算将某些位置 0 时，与这个运算组合使用很方便，如：

```
PORTA &= ~0x80; // 将位 7 置 0
```

这个编译器对这些运算能产生最理想的机器指令，如 sbic 指令可以用在根据位的状态进行条件分支的&运算中。

3. 程序存储器和常量数据

AVR 是哈佛结构的 MCU，它的程序存储器和数据存储器是分开的。这样的设计是有一些优点的。例如，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，如 ATmega 系列允许有超过 64K 字（WORD）的程序存储器和 64K 字节的数据存储器。将来的 MCU 可能用到更多的程序存储器，而程序计数器仍保留在 16 位上。

但是，C 不是在这种结构的 CPU 上发明的。特别是 C 的指针是任意一个数据指针或函数指针，C 规则已经指定不可以假设数据和函数指针能被向前和向后修改。可是哈佛结构的 AVR，要求数据指针能指向任一个数据内存和程序内存。非标准 C 解决了这个问题，ImageCraft AVR 编译器使用“const”限定词表示数据是在程序存储器中，例如：

```
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数组的，“ptr1”表示数据在数据存储器而指向数据的指针在程序存储器，“ptr2”是一个数据在程序存储器而指向数据的指针在数据存储器，最后，“ptr3”是数据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，C 编译器生成 LPM 指令来访问程序存储器。

最后，注意只有常量以文件形式可以放入 FLASH 存储中，例如定义在函数体外的变量（全局变量）或有静态存储类型限制的变量的初始化值。如果使用有 const 限制的局部变量，将不会被放入 FLASH 存储器中，因为这样可能会导致不明确的结果。

4. 字符串

一种方法是将字符串同时分配到数据和程序存储器中，在程序启动时，字符串是由程序存储器复制到数据存储器中的。这种类型的字符串可以使用标准 C 的字符串处理函数。

如果希望节省空间，可使用常量字符型数组来将字符串只分配进程序存储器中。例如：

```
const char hello[] = "Hello World";
```

在这个例子中，hello 可以在上下文中作为字符串使用，但不能用作标准 C 库中字符串函数的参数。

Printf 已被扩展成带%S 格式字符来输出只存储于 FLASH 中字符串，另外，新的字符串函数已加入了对只存储于 FLASH 中字符串的支持。

在 ICCAVR 中，可以设置将字符串只分配到 FLASH 存储器中，当对应"Project→Options→Target→Strings In FLASH Only"检查框被选中时，可以设置编译器将字符串只放在 FLASH 中，这时必须很小心地调用库函数。如果这个选项选中，字符串类型"const char *"是有效的，并且必须保证函数获得了合适的参数类型。除了新的"const char *"与字符串有关系外，ICCAVR 还创建了 cprintf 和 csprintf 函数支持字符串的 const 类型。

注意：当只分配全部字符串到 FLASH 存储器中时，应使用 cprintf()，对 const char*及 const char ptr[ ]类型字符串操作，必须使用%S 参数。对第一种分配情况应当使用 printf()，对 const char*及 const char ptr[ ]类型字符串，也必须使用%S 参数。

5．堆栈

C 语言程序使用两个堆栈：一个是用于子程序调用和中断操作的硬件堆栈，一个是用以传递参数、临时变量和局部变量的软件堆栈。

硬件堆栈是用于存储函数返回的地址，如果程序中没有太多的子程序调用，也不调用像带有%f 格式的 printf()等库函数，那么默认的 16 字节就能够很好地工作。除了很频繁的递归调用程序（再入式函数），在绝大多数程序中，40 个字节的硬件堆栈就足够了。

硬件堆栈是从片内数据存储器的顶部开始分配的，而软件堆栈是在它下面一定数量字节处开始分配。硬件堆栈和数据存储器的大小是受在编译器选项中的设置限制的。数据区从 IO 空间后面 0x60 开始分配，允许数据区和软件堆栈彼此相向生长。

如果选择的目标 MCU 带有 32K 或 64K 的外部 SRAM，那么堆栈也是放在内部 SRAM 的顶部，而且向低内存地址方向生长，以提高程序运行的速度。

堆栈检查

程序不能正常运行的重要原因之一是堆栈溢出，两个堆栈中的任意一个都有可能溢出，当任意一个堆栈溢出时都会造成程序运行不正常，可以使用堆栈检查函数检测溢出情况。

6．在线汇编

除了在单独的汇编文件中写汇编函数外，在线汇编允许在 C 语言程序文件中嵌入汇编代码。当然，使用单独的汇编源文件作为项目的一部分是一个良好的习惯。

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，但不支持定义之外的 ASM 关键词。为了在汇编程序中访问一个 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）来强制分配一个变量到 CPU 寄存器中。

通常，使用在线汇编引用寄存器的能力是有限的。如果在函数中声明了太多的寄存器变量，就很可能没有寄存器可用。在这种情况，将从汇编程序得到一个错误，也不能控制寄存器变量的分配，所以在线汇编指令很可能就会失败。例如，使用 LDI 指令需要使用 R16~R31 中的一个寄存器，但这里没有请求使用在线汇编，同样也没有引用上半部分的整数寄存器。

在线汇编可以被用在 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。不像 AVR 汇编器，ImageCraft 汇编器允许标签放置在任意地方，所以可以在用户的在线汇编代码中创建标签。当汇编声明在函数外部时，用户可能会得到一个警告，当然此警告可以不予理睬。

7. IO 寄存器

对包括状态寄存器 SREG 在内的 IO 寄存器，可以以两种方法访问。一种方法是使用在 0x00 和 0x3f 之间的 IO 地址，此时可以使用 IN 和 OUT 指令读写 IO 寄存器；另一种方法是使用映射到 0x20 和 0x5F 之间的数据内存地址，此时可以使用普通数据访问指令和地址模式。两种方法在 C 中都可使用：

数据内存地址，一个直接地址可以通过加指针类型符号直接访问。例如，SREG 的数据内存地址是 0x5F：

```
unsigned char c = *(volatile unsigned char *) 0x5F; // 读 SREG
*(volatile unsigned char *) 0x5F |= 0x80;  // 打开全局断位
```

注意：数据内存地址 0~31 涉及到 CPU 寄存器 R0~R31，注意不要无意地改变了 CPU 寄存器的内容。

当访问在 IO 寄存器范围中的数据内存时，编译器自动生成 in、out、sbrs、sbrc 等一类的操作指令。

IO 地址，可以使用在线汇编和预处理宏来访问 IO 地址：

```
register unsigned char uc;
asm("in %uc,$3F"); //  读 SREG
asm("out $3F,%uc");     // 打开全局中断位
```

8. 绝对内存地址

用户的程序可能需要使用绝对内存地址，例如 LCD 界面和双口 SRAM 等一些外部 IO 设备通常被映射成特殊的内存，可以使用在线汇编或单独的汇编文件来描述那些定位在特殊内存地址的数据。

在下面例子中，假设有一个两字节的 LCD 控制寄存器定位在 0x1000 地址，一个两字

节的LCD数据寄存器定位在0x1002地址，并且有一个100字节的双口SRAM定位在0x2000地址。

使用汇编模式，在一个汇编文件中输入以下内容。

```
.area memory (abs)
.org 0x1000
_LCD_control_register:: .blkw 1
_LCD_data_register:: .blkw 1
org 0x2000
_dual_port_SRAM::   .blkb 100
```

在C文件中必须这样描述：

```
extern unsigned int LCD_control_register, LCD_data_register;
extern char dual_port_SRAM[100];
```

注意：规定在汇编文件中外部变量名称是带‘_’前缀的，并且使用两个冒号定义为全局变量。

使用在线汇编

在线汇编遵守同样的汇编语法规则，只是它被附加在一个 asm()函数之内。在 C 文件中，上面的汇编代码可以这样书写：

```
asm(".area memory(abs)"
    ".org 0x1000"
    "_LCD_control_register:: .blkw 1"
    "_LCD_data_register:: .blkw 1");
asm(".org 0x2000"
    "_dual_port_SRAM:: .blkb 100");
```

在 C 中仍然要使用“extern”描述变量，正像上面使用单独的汇编文件那样，否则 C 编译器不会真正知道在 asm()中的声明。

9. C 多任务操作

作为一种规则，编译器通常生成代码来保存和恢复一些需要保护的寄存器。在一些情况下，这种行为可能是不合适的。例如，如果使用 RTOS（实时操作系统），RTOS 管理着寄存器的保存和恢复并作为任务切换处理的一部分，编译器如果再插入这些代码就变得多余了。

为了禁止这种行为，可以使用“#pragma ctask”，例如：

```
#pragma ctask drive_motor emit_siren
```

```
....
void drive_motor() { ... }
void emit_siren() {...}
```

这个附注（pragma）必须被用在函数定义之前。默认的情况下，从不返回的程序“main”是有这个属性的，它也没有必要为返回保存和恢复任意一个寄存器。

10. 中断操作

在 C 程序中可以使用中断操作，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”中断的向量号，注意向量号是从复位向量的向量号 1 开始。这个附注有两个作用：

（1）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。

（2）编译器根据向量号和目标 MCU 生成相应的的中断向量入口。

例如：

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

编译器生成的指令为

```
rjmp _timer_handler;  对普通 AVR  MCU
```

或者

```
jmp _timer_handler  ;  对 Mega  MCU
```

上述指令定位在 0x06（字节地址，针对普通 AVR）和 0x0c（字节地址，针对 MegaAVR）。（Mega 使用 2 个字作为中断向量，非 Mega 使用 1 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。例如：

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

汇编中断操作

可以用汇编语言写中断操作。如果在汇编程序内调用 C 函数，无论如何要小心。汇编程序要保存和恢复一些挥发寄存器，C 函数不保护挥发寄存器。

如果使用汇编写中断操作，那么必须自己定义中断向量。使用“abs”属性描述绝对区域，用“.org”来声明rjmp或jmp指令的正确地址。注意这个“.org”声明使用的是字节地址。

对除ATmega以外的MCU，可以这样定义：

```
.area vectors(abs)      ; 中断向量
.org 0x6
rjmp _timer
```

对ATmega MCU，可以这样定义

```
.area vectors(abs)      ; 中断向量
.org 0xC
jmp _timer
```

11．访问UART

默认的库函数getchar()和putchar()使用查寻模式访问UART。在\icc\examples.avr目录中，有一个以中断方式工作的UART例子程序。

12．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>。

EEPROM_READ(int location, object)

这个宏调用了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，包括结构和数组。例如：

```
int i;
EEPROM_Read(0x1, i);    // 读2个字节给i
EEPROM_WRITE(int location, object)
```

这个宏调用了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。例如：

```
int i;
EEPROM_WRITE(0x1, i);   //写两个字节至0x1
```

这些宏和函数可以用于任意AVR芯片。对EEPROM单元少于256字节的MCU，这个宏不是最好的，因为此时不需要使用高位字节的地址。如果关系重大，可以为EEPROM较少的目标MCU重新编译库源代码。

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”中的，这是可以用附注实现的，结果是产生扩展名为.eep 的输出文件。例如：

```
#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
```

第二个附注是必需的，为返回默认的“data”区域需要重设数据区名称。

注意，因为AVR的硬件原因，不要使用EEPROM的零地址，在C程序中初始化数据时，会自动避开0地址。

注意，当使用外部描述（比如访问在另一个文件中的 foo），不需要加入这个附注，例如：

```
extern int foo;
int i;
EEPROM_READ((int)&foo, i);
```

如果需要，下列函数也可以直接用来访问EEPROM：

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”个字节到由“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。

13．访问SPI

ICCAVR提供了一个以查寻模式访问SPI的范例程序，更多的信息可以参考spi.h。

14．相对转移/调用的地址范围

一个带8K程序存储器的MCU，全部范围内的跳转可以使用相对转移和调用指令(rjmp和 rcall)。相对转移和调用的范围是以8K为分界的，例如一个较远的跳转，跳转到0x2100字节处（0x2000为8K），实际上会跳转到地址0x100处。

15．ICCAVR的数据类型

数据类型 长度（字节） 范围

unsigned char	1	0..256
signed char	1	-128..127
char	1	0..256
unsigned short	2	0..65535
(signed)short	2	-32768..32767
unsigned int	2	0..65535
(signed) int	2	-32768..32767
unsigned long	4	0..4294967295
(signed) long	4	-2147483648..2147483647
float	4	+/-1.175e-38..3.40e+38
double	4	+/-1.175e-38..3.40e+38

注意："char" 等同于 "unsigned char"。

f loats 和 doubles 是 IEEE 标准 32 位格式，7 位表示指数，23 位表示尾数，1 位表示符号。

位域类型必须被赋予 unsigned 或 signed 关键字，而且被包含在一个较小的空间中。如可定义成结构：

```
struct {
    unsigned a : 1, b : 1;
    };
```

这个结构体的长度只有 1 个字节。位域是从右往左放置的。

16. 汇编界面和调用规则

（1）名称

C 语言中的名称在汇编文件中是以下划线为前缀的，如函数 main()在汇编模块中是以 _main 引用的。名称的有效长度为 32 个字符，在名称后面加两个冒号(::)，可以定义成一个全局变量。例如：

```
_foo::
    .word 1
(在 C 文件中)
    extern int foo;
```

（2）传递参数和返回值所使用的寄存器

第一个参数若是整型则通过 R16/R17 传递，第二个参数则通过 R18/R19 传递。如果第一个参数是长整型或浮点数则通过 R16/R17/R18/R19 传递；其余参数通过软件堆栈传递。比整型参数小的（如 char）参数扩展成整型（int）长度传递，即使函数原型是 char 的声明。如果 R16/R17 已传递了第一个参数，而第二个参数是长整型或浮点数，则第二个参数的低

半部分通过 R18/R19 传递，而高半部分通过软件堆栈传递。

整型返回值是通过 R16/R17 返回，而长整型或浮点数返回则通过 R16/R17/R18/R19 返回。

（3）保护的寄存器

在汇编函数中必须保护和恢复下列寄存器：

R28/R29 或 Y，这是结构指针。

R10/R11/R12/R13/R14/R15/R20/R21/R22/R23，这些寄存器的内容在被汇编语言函数调用后必须保持不变。

（4）挥发寄存器

R0/R1/R2/R3/R4/R5/R6/R7/R8/R9/R24/R25/R26/R27/R30/R31

SREG

可以在汇编语言函数中使用这些寄存器，而不用保护和恢复，这些寄存器的内容在被函数调用后可以改变。

（5）中断处理

这不同于普通的函数调用，在中断操作中必须保护和恢复它所使用的全部寄存器。如果使用 C 函数来描述中断处理，那么编译器有能力自动完成。如果使用汇编写中断处理，而它又调用了普通的 C 函数，那么汇编操作必须保护和恢复挥发性寄存器，因为普通 C 函数调用不保护它们。中断处理操作同普通程序操作是异步的，中断处理或它的函数调用不能改变任意一个 MCU 寄存器。

17．函数返回非整型值

在声明函数时可能返回一个长整型、浮点数或结构。作为例子，在调用任意浮点函数之前，应当用#include 语句包含头文件<math.h>，否则，当从调用的浮点函数返回时，程序可能不正常。

返回长整型数或浮点数

长整型数或浮点数返回值是保存在寄存器 R16~R19 中。

传递结构值

如果传递结构值，结构只通过堆栈传递，而不是通过寄存器。传递结构指针（也就是传递结构的地址）和传递任意数据类型的地址是相同的，都是通过一个 2 字节的指针。

返回结构值

当一个返回结构的函数被调用时，这个调用函数分配一个临时数据结构，并且传递一个隐藏指针给调用函数。当这个函数返回时，它复制返回值到这个临时数据结构中。

18．程序和数据区的使用

程序存储器是被用于保存程序代码、常数表和确定数据的初始值，比如字符串、全局变量等。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，编译器不能使用任意 64K 字节以上的程序存储器。为了访问 64K 字节边界以上的存储器（如在 Mega103 装置中），需要在设定 RAMPZ 寄存器后直接调用 ELPM 指令。

数据存储器（仅指内部 SRAM）

这个数据存储器是用于保存变量、堆栈结构和动态内存分配的堆。通常，它们不产生输出文件，但在程序运行时使用。一个程序使用数据内存如图 4.9 所示。

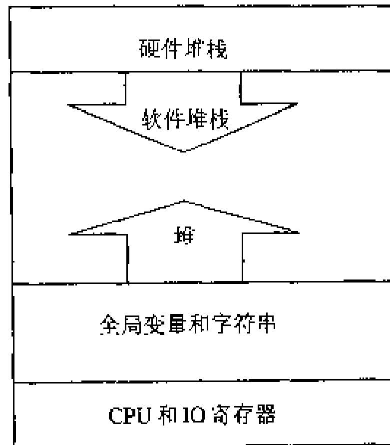


图 4.9　内存图

内存图的底部是地址 0，开始的 96（0x60）字节是 CPU 寄存器和 IO 寄存器。编译器从 0x60 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。在高端地址，硬件堆栈开始于 SRAM 的最后位置，在它的下面是向下生长的软件堆栈。它要求作为程序设计师，要确保硬件堆栈不生长进软件堆栈，而软件堆栈不生长进堆，否则将导致意外的结果。

数据存储器（外部 SRAM）

如果选择带有 32K 或 64K 外部 SRAM 的目标 MCU，那么堆栈还是放置在内部 SRAM 的顶部并且是朝低端内存地址方向向下生长，数据内存是开始于硬件堆栈的顶部并且向上生长。这样分配的原因是在多数场合下访问内部 SRAM 比访问外部 SRAM 的速度要快，分配堆栈到较快的内存有很多好处。

19. ICCAVR 中区域的划分

ICCAVR 编译器生成代码和数据到不同的区域 “areas.”，区域按照内存地址增高的顺序被编译器使用。

只读存储器中包含以下几个区域：

interrupt vectors——中断向量区，这个区域包括中断向量。

func_lit——函数表区。这个区的每个字包括了函数入口的地址。为了完全兼容代码压缩，所有间接的函数的调用必须通过这个区域。如果在 C 中使用函数指针调用函数，这是自动完成的。在汇编中，举例如下：

```
; 假设 _foo 是函数的名称
.area func_lit
PL_foo:: .word _foo ; 创建函数表入口
. area text
```

```
ldi R30,<PL_foo
ldi R31,>PL_foo
rcall xicall
```

可以间接地将函数表入口地址送入 R30/R31 寄存器对后，使用库函数 xicall 调用这个函数。

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。

数据存储器包含以下几个区域：

data——这个区域包括全局变量、静态变量和字符串，全局变量和字符串的初始值保存在"data"域，并且是在启动时被复制到数据区的。

bss——这个区域包括未初始化的全局变量，按 ANSI C 定义这些变量在启动时将被初始化为 0。

EEPROM 存储器只有一个区域：

eeprom——这个区域包括 EEPROM 数据，EEPROM 数据是写进扩展名为.eep 的输出文件，为 INTEL HEX 文件格式。

20. 调试

ICCAVR 可以输出 COFF 格式调试文件，使用户可在 ATMEL 的 AVRStudio 中进行源程序级的调试。如果用户想使用 AVRStudio 中的模拟 IO 及终端仿真器，那么在 ICCAVR 的编译选项中必须将"AVR Studio Simulator IO"一项打勾。注意：AVRSTUDIO 有 3.xx 和 4.0xx 两种版本，其支持的 COFF 文件格式是不一样的。如果使用 AVR STUDIO4.xx 调试程序，应当选中项目设置对话框中的"AVR STUDIO4.0 Compatilbe"检查框；如果使用 AVR STUDIO3.xx 调试程序，则不要选中这个检查框。

4.2.7 应用简单举例

1. 读/写 IO 端口

```
#include <io8515.h>
//PD 口的输入控制 PB 口的输出
void main(void)
{
 unsigned char achar;
 DDRB = 0xFF;           //PB 口作为输出
 DDRD = 0x00;           //PD 口作为输入，内部上拉
 PORTD= 0xFF;
 for(;;)
```

```
    {
     achar = PIND;          //PD 口的管脚状态
      PORTB = achar;        //PB 口输出所读到的 PD 口的管脚状态
    }
}
```

2. 延时函数

```
#include <io8515.h> /* 定义 8515 */
void delay(int delayValue)
{
 int i;                              //定义整型变量 i [-32768,32767]
 for(i=0;i<delayValue;i++) ;         //i<delayValue 循环
}
void main(void)
{
 unsigned char runner = 0x01;        //定义字符型变量 runner
 DDRB = 0xff; /* Port B 输出 */
 for (;;) /* 死循环 */
      {
       if (runner) runner <<= 1;     //runner!=0   右移一位
       else runner = 0x01;           // runner==0  则重置 runner 值
PORTB = runner;                      // PB 口输出 runner 值
       delay(32767);                 // 调用延时函数
       }
}
```

3. 读/写 E²PROM 存储器

```
//int EEPROMwrite( int location, unsigned char);
//unsigned char EEPROMread( int);
#include <io8515.h>
#include <eeprom.h>
void main(void)
{
  unsigned char temp =0xaa,i;
    EEPROMwrite(0x20,temp); /* 写 E²PROM 地址 0x20  */
    i=EEPROMread(0x20);      /* 读 E²PROM 地址 0x20 */
     i++;                    //读到的值加 1
     EEPROMwrite(0x30,I);    //写 E²PROM 地址 0x30
}
```

4. AVR 的 PB 口变速移位

```
#include <io8515.h>
#define BIT(x) (1<<(x))          //定义移位函数 BIT()
void delay(void)
{                                //延时函数
 unsigned char i,j;
    for (i=1;i;i++)
        for(j=1;j;j++);
}
void led_pb(void)
{                                //PB 口输出函数
 unsigned char i;
 DDRB=0xff;                      //定义 PB 口作为输出口
 for (i=0;i<8;i++)
   {
    PORTB=~BIT(i);               //取反后输出
    delay();                     //调用延时函数
   }
}
void main (void)
{
  while (1)                      //死循环
    led_pb();                    //调用 PB 口输出函数
}
```

4.3 SL-MEGA8 开发实验器

SL-MEGA8 是广州双龙电子有限公司开发的一种实验评估板，主要针对 ATMEL 新出品的 ATmega8 芯片设计。在 SL-MEGA8 实验评估板的设计中，充分考虑了 ATmega8 芯片的硬件特性，为其配套了相应的外围电路，从而方便用户在 SL-MEGA8 上做有关 ATmega8 的硬件实验。出于对客户的支持考虑，SL-MEGA8 配套了相应的 C 语言程序的源代码，这些代码既可以充分展示 ATmega8 芯片和 SL-MEGA8 实验评估板的特性，又可以帮助用户掌握 ATmega8 的硬件特性，从而加速用户程序的开发。

SL-MEGA8 实验评估板支持对 ATmega8 的 ISP 下载和 IAP 下载，并且提供相应的源代码。

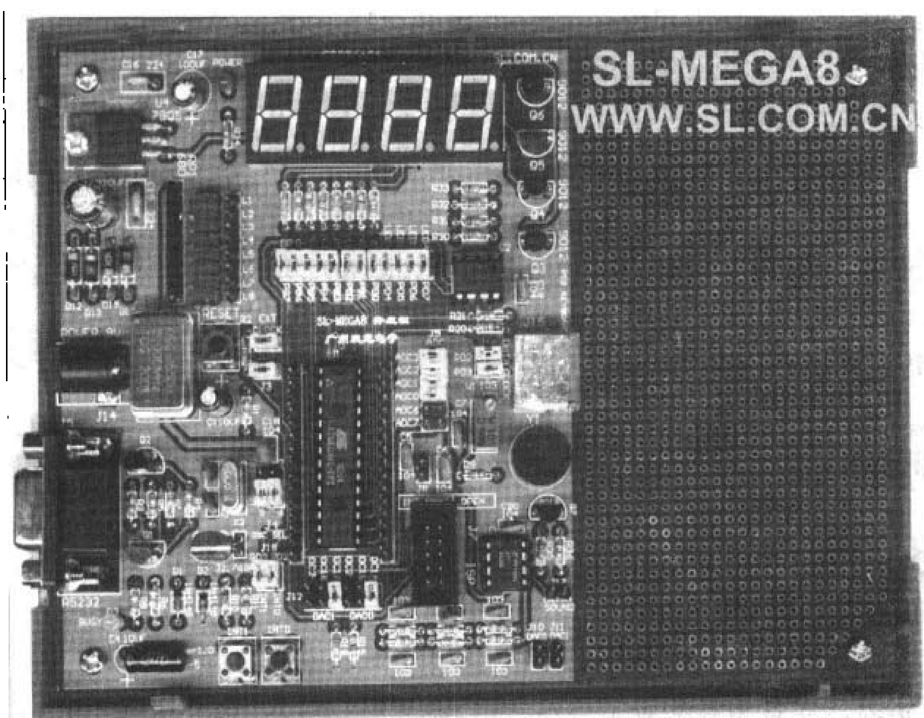


图 4.10　SL_MEGA8 实验图

4.3.1　SL- MEGA8 开发实验器硬件结构

为了充分应用 ATmega8 的 I/O 口的功能，SL-MEGA8 采用插拔短路块方式及用专用接插线连接来改变 ATmega8 的 I/O 口的用途。例：第 1 脚是 PC6，既可当 I/O 口用，又可当复位引脚用；第 9、10 脚有三种不同的用途。SL-MEGA8 开发实验器（评估系统）硬件电原理图见附件 PCB_SCH 文件夹（购机提供）。

SL- MEGA8 开发实验器主要器件如图 4.11 所示，各器件功能如下：

输入的 AC 9V 电源经整流稳压滤波后产生+5V，供整个电路用。

J1、J2 对应 ATmega8 芯片两旁的两排引脚输出插针，供用户实验用，也可作为 32 脚 TQFP 封装转接用（选购件）。

J3 为 PC6 选择跳线：如果用作外部复位，J3 必须短路，以连接复位按钮；如果 PC6 用作 I/O 口用，J3 开路。

J4 为晶振工作选择用：当 1、3 与 2、4 插上短路块时，是用外接无源晶振；当 3、5 与 4、6 插上短路块时，是用 32.768K 晶振，作实时时钟实验用；当 J4 不插短路块时，则用 ATmega8 内部 RC 振荡器。ATmega8 的时钟共有三种选择：外接 8MHz 无源晶振；8MHz 有源晶振（当用户配置错误，造成外接无源晶振不能起振时，用有源晶振就能帮助用户重新配置芯片）；内部 RC 振荡器，可做成无外接器件应用系统。插上 EXTCLOCK 短路块，则用有源晶振，该处平时不插短路块。

J5 是 RS232 接口，可用来做异步串行通信实验。双龙提供的 IAP 例程，就是使用这个

RS232 口和 PC 机通信，将用户程序（Flash/EEPOM）下载到 ATmega8 芯片中去；另外，利用这个 RS232 口，借助于超级终端软件，用户可以做一些通信的实验，也可以借助于超级终端来调试程序，从而方便了 ATmega8 程序的开发。SL-MEGA8 出厂时 ATmega8 的 BOOT 区已写入监控程序，只需用标准 RS232 通信电缆与 PC 机连接，接通电源就能做 IAP 下载。

J6 为 8 位 LED 发光二极管，用接插线供 ATmega8 的 I/O 口作电平指示实验用，低电平 LED 灯亮。

J7 为 ISP 编程接口，供 ATmega8 下载程序、配置熔丝用（随机配 SL-AVRISP 并口通信电缆）。

J8 为外部基准电压选择，用户可用来选择外部或内部电压基准电压作为 AD 转换的参考电压。

J9 为 ADC 信号通道输入接口，多圈电位器 W1（10K）为 ADC 提供可变的输入电压，在 J9 处用短路块可选择相应的 ADC 输入通道。

J10、J11 为 DAC0、DAC1 信号输出接口，在 SL-MEGA8 上使用 ATMEGA8 的二路 PWM 可以做 DAC 实验，并且为其配置了三级低通滤波器、电压缓冲运算放大器等电路。

J12 可以使用户在 ATmega8 的三个 PWM 和两路低通滤波器通道之间进行自由的选择。

J13 为 4 位 LED 数码管的段、位选短路块。

J14 为外接 AC 9V 电源插座，SL-MEGA8 有单独的电源系统，大大方便了用户对电源选择的自由。

J15 为标准的 PC 机 PS/2 键盘接口，J16 可以用来选择 PS/2 键盘和 ATmega8 的连接方式，如果插上短路块，则可能和 SL-MEGA8 的配套程序相适应，是默认的选择。

J17 为音响输出短路块。

J18 有两个按键 S2、S3 可以作外部中断 INT0、INT1 实验，另外使用这两个按钮也可编程键盘程序，J18 提供了用户接线的灵活性。

U1 为 ATmega8 DIP 插座，芯片缺口向上。

U5 为 4 位 LED 数码管作显示用。

Y1 为无源音响器，使实验有声有色。

D3 作 BUSY 通信忙指示。

POWER 为电源指示。

U2 为 AT24C02，用户可以做 ATmega8 的硬件 I^2C 实验。

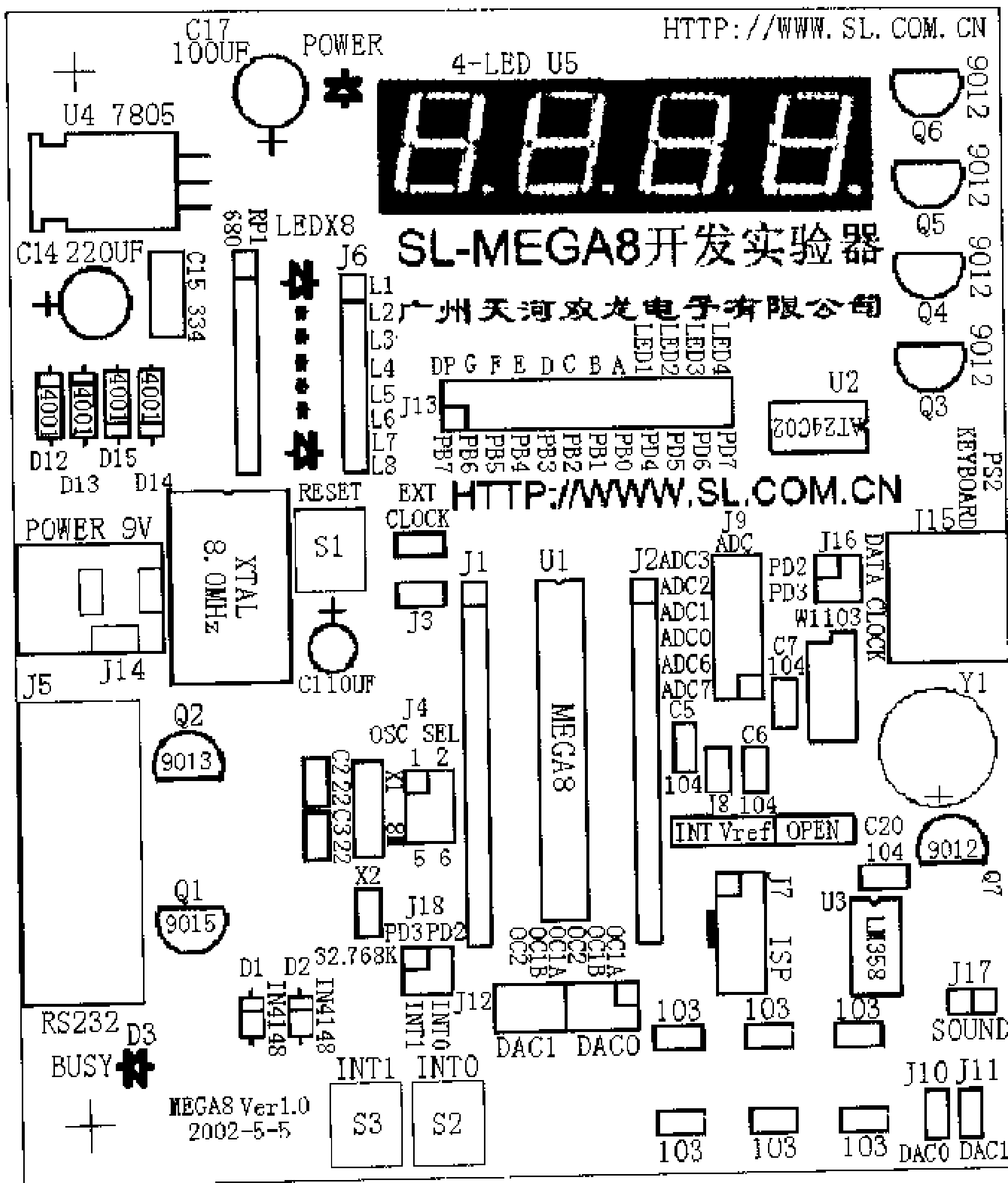


图4.11 SL_MEGA8开发实验器器件布置图

4.4 ATmega8的编程操作

ATmega8既可以用外部下载电缆对其做ISP编程，也可用其内部的监控程序做IAP编程。注意，只有双龙SL-MEGA8配套的ATmega8才有IAP的监控程序，例：双龙电子的SL-AVRISP串行下载电缆、SL-AVR/SL-AVR+/SL-AVRAD开发实验器等均可对ATmega8芯片编程。AVR Studio中内嵌的AVRPROG软件支持串行下载编程。双龙开发实验器下载电缆一头接PC机的RS232串行口，另一头接开发实验器，接上5V电源（红色线接+5V，黑色线接地），按图4.12操作，如果连机正确，则出现如图4.13所示的窗口，然后根据需

要进行相应的操作。

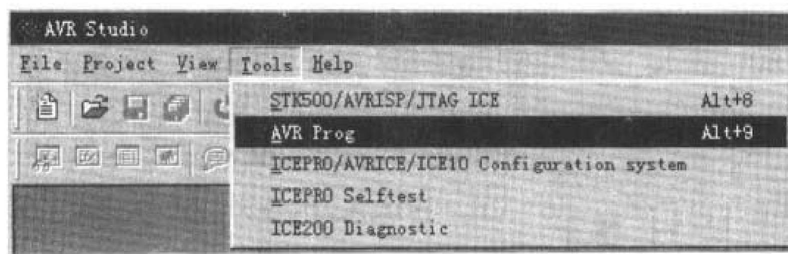


图 4.12 串行下载选项

1. 启动 AVR Prog：启动 AVR Stuio 程序，通过 TOOLS 菜单中的 AVR Prog 命令（或快捷键 Alt+9，见图 4.12），进入 AVR Prog 操作窗口，如图 4.13 所示，根据窗口提示操作。

2. 双龙电子提供 AVRProg 软件的汉化版，如图 4.13 所示。

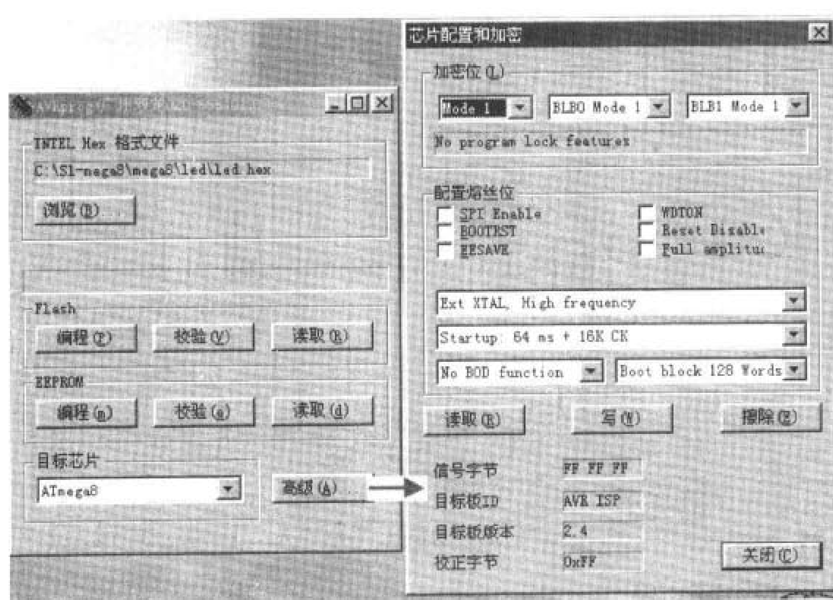


图 4.13 下载选项

3. 双龙并口通信下载电缆的使用

双龙并口通信下载电缆可以使用共享软件 PonyProg2000 进行操作，使用之前应进行相应的设置和校正。

（1）从双龙光盘上安装 PonyProg2000 软件，并拖放到桌面上；也可从互联网下载安装，网址为 http://www.LancOS.COM，也提供了相应的汉化版。

（2）双击 PonyProg 桌面快捷图标，出现 PonyProg2000 的版权提示窗口（如图 4.14 所示），确认后进入 ISP 下载操作窗口。

（3）PonyProg2000 软件操作窗口

中文操作窗口菜单功能从略，实际操作可用鼠标单击快捷按钮进行，如图 4.15 所示。

图 4.14　PonyProg2000 的版权提示窗口

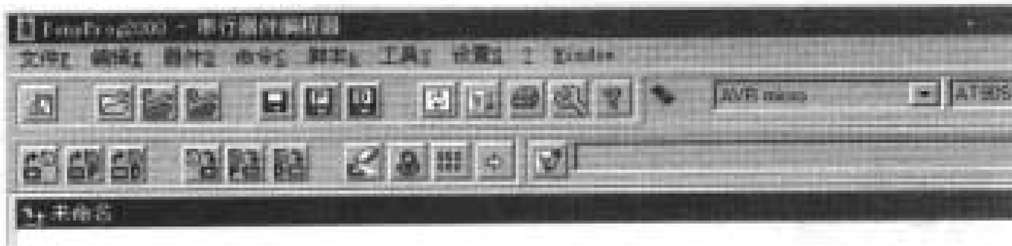


图 4.15　快捷按钮

（4）ISP 下载通信接口设置

打开设置下拉菜单，选接口设置，选择并行接口 LPT1，设置通信协议，如图 4.16 所示。

图 4.16　并口下载选项

（5）器件选择：先选 AVR micro，再选用 ATmega8，如图 4.17 所示。

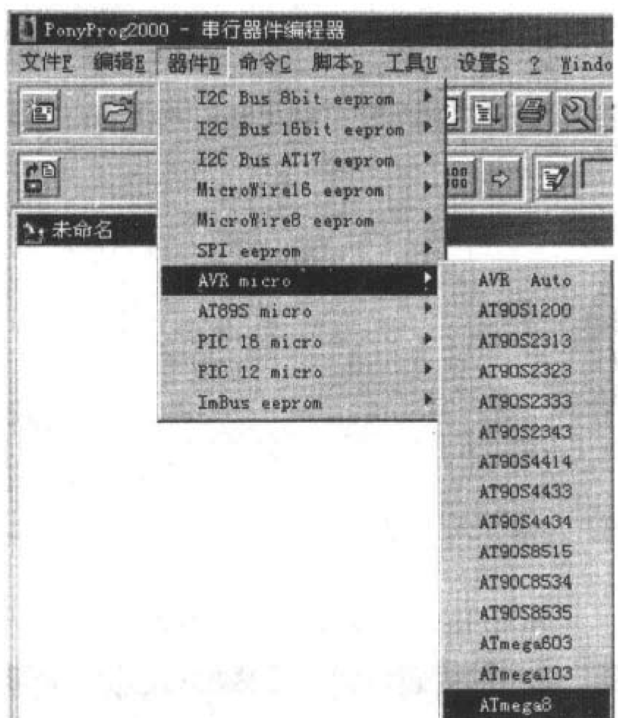


图 4.17 器件选择

（6）编程选项

①打开命令下拉菜单；

②选中编程选项；

③根据实际需要选择，打勾处为选中；

④单击确认，如图 4.18 所示。

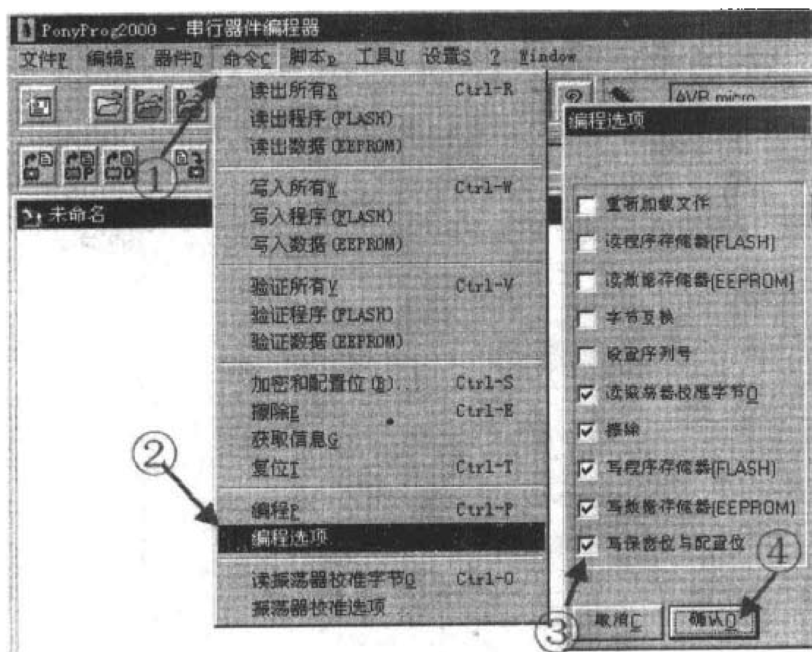


图 4.18 编程选项

（7）装入 INTEL HEX 格式文件

①打开器件内容文件；

②文件类型选 HEX；

③选文件路径；

④选择将要下载的文件名；

⑤按打开按钮，该 HEX 文件装入计算机缓冲区，并把机器码显示在下载窗口中，如图 4.19 所示。

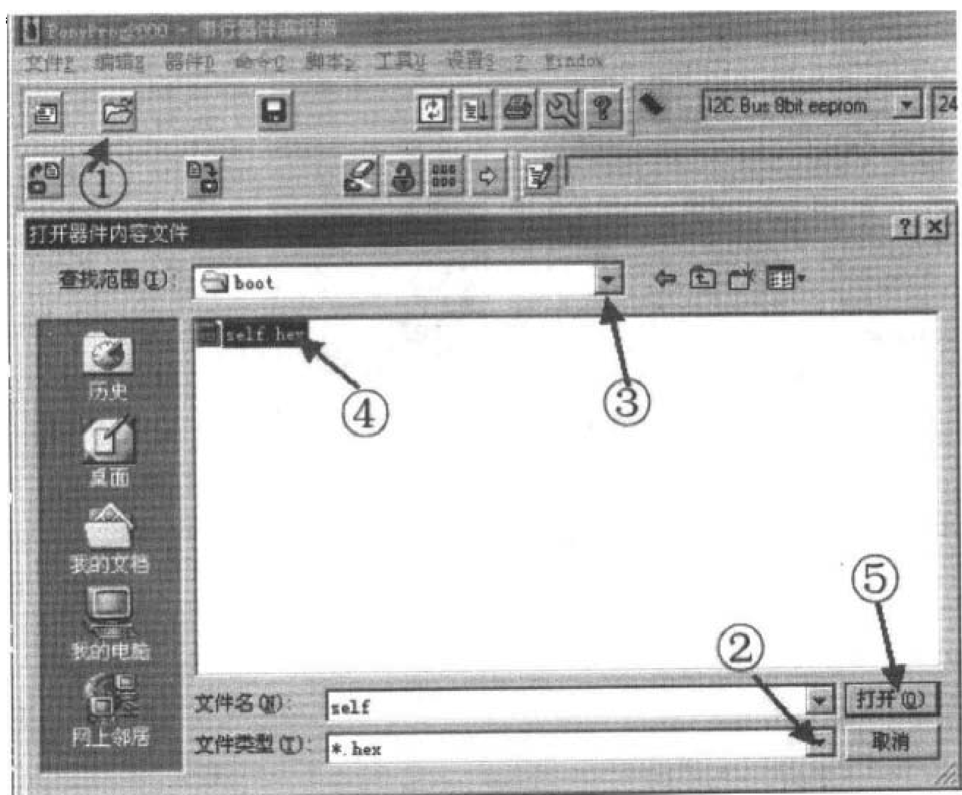


图 4.19　打开器件 HEX 文件

（8）程序下载操作

①鼠标单击器件下载快捷按钮；

②确认对器件操作正确，按 Yes 键进行下载操作；

③下载状态过程显示；

④下载成功后，下载程序在 SL-MEGA8 开发实验器上立即运行，如图 4.20 所示。

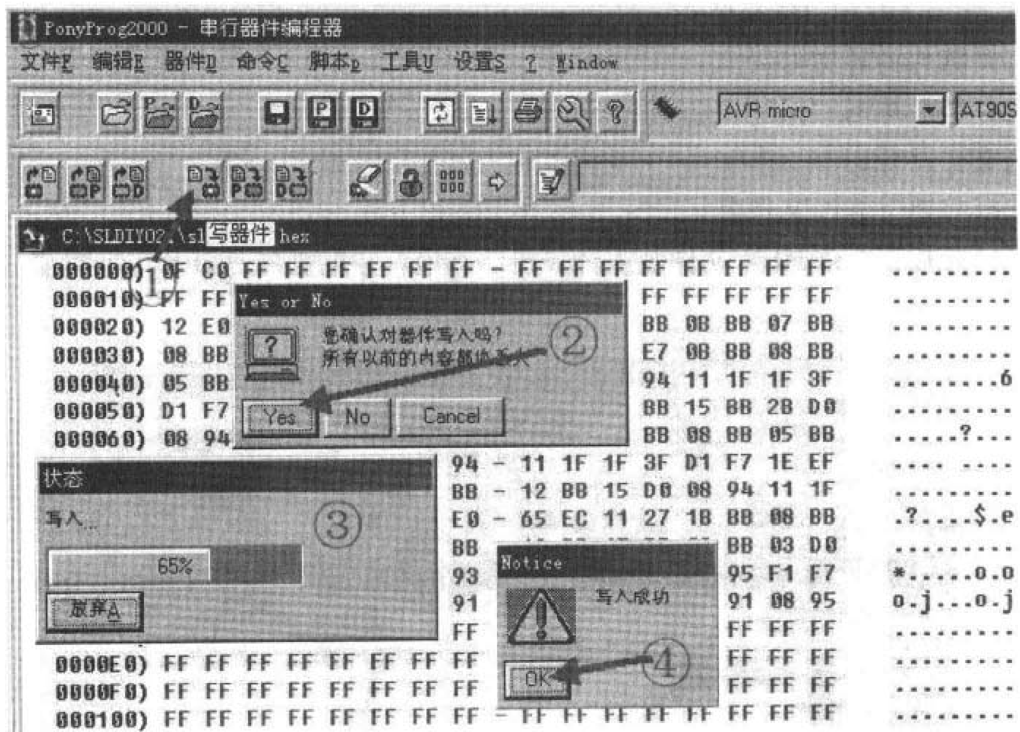


图 4.20 程序下载操作过程

第 5 章　ATmega8 应用设计

5.1　硬件 I²C 的应用

ATmega8 具有硬件的二线总线（TWI）接口，完全兼容于 Philips 的 I²C 总线。ATmega8 的二线总线（TWI）接口是以字节为单位进行处理的，支持中断或查询方式访问，比软件模拟 I²C 总线有更好的实时性和代码效率。另外，由于 ATmega8 是以字节为单位进行处理的，在编程时会比其他厂家的 MCU 更加方便。

以下是以查询方式编写的用来访问 AT24C02 程序的 I²C 源代码，用户移植后也可用于其他的 I²C 器件。

本范例程序的使用说明：

1．LED 数码管的右边两位（LED0，LED1）显示从 AT24C02 读出的数据，左边两位（LED2，LED3）显示 AT24C02 当前的地址

2．可以使用 INT0/INT1 键来修改地址或数据，单击 INT0 按键为增大，单击 INT1 按键为减小。

3．当 LED0 小数点点亮时为修改数据模式；当 LED2 小数点点亮时为修改地址模式。

4．如果在按住 INT0 或 INT1 键不放的同时，按了另外一个按键，则进行修改模式切换。如果是从修改数据模式切换到修改地址模式，则当前的数据被写入到 AT24C02 的当前地址。

5．本范例程序没有进行超时处理，用户可自行加入超时处理代码。

6．外部函数声明的函数定义在文件 numled.c 中。

```
/*********************************************************/
/*                 广州天河双龙电子公司                  */
/*                 http://www.sl.com.cn                  */
/*                 硬件 I2C 总线演示程序                 */
/*                 作者：ntzwq@wx88.net                  */
/*                   2002 年 5 月 12 日                  */
/*   目标 MCU：MEGA8     晶振：内部 RC(INT) 8MHz         */
/*********************************************************/

#include<iom8v.h>     //包含定义 ATmega8 寄存器的头文件
```

```
#include"TWI.H"      //包含了一些常用二线总线(TWI、I²C)状态和操作的定义
#define rd_device_add 0xa1                    //定义从器件读地址
#define wr_device_add 0xa0                    //定义从器件写地址
extern const unsigned char seg_table[16];     //定义 LED 数码管段表
extern unsigned char led_buff[4];             //定义 LED 显示缓冲区
extern void delay_ms(unsigned int time);      //ms 级延时函数
extern void display(void);                    //LED 动态扫描函数

/*******************************************
I²C 总线写一个字节
返回 0：写成功
返回非 0：写失败
Wdata：写入到 AT24C02 的数据
RomAddress：写入到 AT24C02 的地址
*******************************************/

unsigned char i2c_Write(unsigned char Wdata,unsigned char RomAddress)
{
      Start();//I²C 启动
      Wait();
      if(TestAck()!=START) return 1;//测试响应
      Write8Bit(wr_device_add);     //写 I²C 从器件地址并设置为写方式
      Wait();
      if(TestAck()!=MT_SLA_ACK) return 1;// 测试响应
      Write8Bit(RomAddress);//写 24C02 的 ROM 地址
      Wait();
      if(TestAck()!=MT_DATA_ACK) return 1;// 测试响应
      Write8Bit(Wdata);//写数据到 24C02 的 ROM
      Wait();
      if(TestAck()!=MT_DATA_ACK) return 1;// 测试响应
      Stop();//I²C 停止
      delay_ms(10);//延时等 EEPROM 写完
      return 0;
}

/*******************************************
I²C 总线读一个字节
如果读失败也返回 0
*******************************************/
```

```
unsigned char i2c_Read(unsigned char RomAddress)
    {
     unsigned char temp;
     Start();//I²C启动
     Wait();
     if (TestAck()!=START) return 0;//测试响应
     Write8Bit(wr_device_add);//写I²C从器件地址并且设置为写方式
     Wait();
     if (TestAck()!=MT_SLA_ACK) return 0;//测试响应
     Write8Bit(RomAddress);//写24C02的ROM地址
     Wait();
     if (TestAck()!=MT_DATA_ACK) return 0;//测试响应
     Start();//I²C重新启动
     Wait();
     if (TestAck()!=RE_START)  return 0;// 测试响应
     Write8Bit(rd_device add);//写I²C从器件地址并且设置为读方式
     Wait();
     if(TestAck()!=MR_SLA_ACK)  return 0;//测试响应
     Twi();//启动主I²C读方式
     Wait();
     if(TestAck()!=MR_DATA_NOACK) return 0;// 测试响应
     temp=TWDR;//读取I²C接收数据
     Stop();//I²C停止
     return temp;
    }
unsigned char i2c_add,i2c_dat;//定义变量存放AT24C02的ROM地址和数据
unsigned char mode;//修改模式标志
/*            刷新LED缓冲区函数            */
void fill_buff(void)
  {
   led_buff[1]=seg_table[i2c_dat/16];//刷新数据
   led_buff[0]=seg_table[i2c_dat%16];
   led_buff[3]=seg_table[i2c_add/16];//刷新地址
   led_buff[2]=seg_table[i2c_add%16];
   if (mode==0)                        //根据修改模式设置状态标志
       led_buff[0]&=0x7f;
   else
           led_buff[2]&=0x7f;
  }
/*                main函数                */
void main(void)
 {
```

```
unsigned char i;
unsigned char add_old,dat_old,mode_old;
TWBR=32;//对TWI总线进行设置，产生合适的SCK频率
TWSR=00;//清除TWI状态
mode=0xff;//mode=0修改数据，!=0修改地址
for(i=0;i<4;i++)//复位后4位LED数码管全部显示8
  {
   led_buff[i]=seg_table[8];
  }
for(i=0;i<200;i++)//显示8888保持一定的时间
    display();
i2c_add=0;//初始化AT24C02地址为0
i2c_dat=i2c_Read(i2c_add);//读AT24C02的数据
fill_buff();//刷新LED缓冲区
add_old=i2c_add;//保存当前状态
dat_old=i2c_dat;
mode_old=mode;
while(1)
  {
    if(add_old!=i2c_add)//如果地址有改变刷新数据
      {
       add_old=i2c_add;
       i2c_dat=i2c_Read(i2c_add);//读AT24C02
       dat_old=i2c_dat;
      fill_buff();//刷新LED缓冲区
      }
   display();
    i=PIND&0x0c;//判断按键
   if(i!=0x0c)
     {
      display();//按键消抖
      if(i==(PIND&0x0c))
        {
         if (i==0x08)//INT0键按下,递增
           {
            if (mode!=0)//修改地址
               {
                if(i2c_add<0xff)
                   i2c_add++;
                else
                   i2c_add=0;
               }
```

```
    else //修改数据
      {
       if(i2c_dat<0xff)
          i2c_dat++;
       else
          i2c_dat=0;
      }
   }
if (i==0x04)//INT1 键按下,递减
  {
   if (mode!=0)//修改地址
      {
       if(i2c_add>0)
          i2c_add--;
       else
          i2c_add=0xff;
      }
   else //修改数据
      {
       if(i2c_dat>0)
          i2c_dat--;
       else
          i2c_dat=0xff;
      }
   }
fill_buff(); //刷新缓冲区
 i=0;
while((PIND&0x0c)!=0x0c)//检查按键释放
    {
     display();
     while(((PIND&0x0c)==0)&(i==0))//判断两键是否全部按下
          {
           i=0xff;
           if (mode==0)//修改模式切换
             {
              mode=0xff;
              led_buff[2]&=0x7f;
              led_buff[0]|=0x80;
              if (dat_old!=i2c_dat) //如果数据有修改，写入到AT24C02中
                  {
                  dat_old=i2c_dat;
                  i2c_Write(i2c_dat,i2c_add);
```

```
                        }
                    }
                else
                    {
                      mode=0;
                      led_buff[0]&=0x7f;
                      led_buff[2]|=0x80;
                    }
                }
            }
        }
    }
}
```

numled.c 略

思考：

1. 用中断模式写一个 TWI 总线的程序，比较和查询方式有何区别？
2. 和 Philips 的 I²C 总线读写时序对比一下，看看 TWI 和 I²C 有何区别？

5.2 A/D 转换器的应用

ATmega8 有多路 10 位精度的逐次比较式 AD 转换器，在用户的测量控制系统中可以显著地降低成本。由于 ATmega8 的 AD 转换器转换速度比较高，用户可以采取一些数字滤波算法来得到较为精确和稳定的转换结果。通过控制 AD 转换的速度和节奏，在克服工频干扰方面完全可以达到双积分式 AD 的性能。

以下是 ADC 模数转换实验的源程序：

说明：1. 本范例程序使用内部 RC 振荡，频率为 8MHz。

2. XTAL1 和 XTAL2 作为 IO 口使用，分别控制 LED 数码管的 G、DP 段。
3. 使用 INT0 或 INT1 按钮切换 ADC 通道。
4. 所有外部函数的定义在文件 numled.c 中。

```
/*********************************************************/
/*                  广州天河双龙电子公司                  */
/*                  http://www.sl.com.cn                 */
/*                  多通道 AD 转换演示程序                */
/*                  作者：ntzwq@wx88.net                  */
```

```
/*                    2002 年 5 月 11 日                        */
/*   目标 MCU：MEGA8    晶振：内部 RC(INT) 8MHz      */
/*******************************************************/

//使用内部 RC 振荡,PB6-G,PB7-DP 短路块连接
#include <iom8v.h>
#include <macros.h>
#define osccal 0x7d //内部 RC 校正常数，对不同的芯片是不同的
#define Vref   500  //参考电压值
extern const unsigned char seg_table[16];
extern unsigned char led_buff[4];
extern void delay_ms(unsigned int time);
extern void display(void);
unsigned int adc_rel;//AD 转换结果
unsigned char adc_mux;//AD 通道
/*            ADC 初始化函数                */
void adc_init(void)
{
 DDRC=0x00;//C 口设置为高阻态输入
 PORTC=0x00;
 ADCSRA = 0x00;
 ADMUX =(1<<REFS0)|(adc_mux&0x0f);//选择内部 AVCC 为基准
 ACSR  =(1<<ACD);//关闭模拟比较器
 ADCSRA=(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1);//64 分频
}
/*            ADC 完成中断处理函数                */
#pragma interrupt_handler adc_isr:iv_ADC
void adc_isr(void)
{
 adc_rel=ADC&0x3ff;//读取 ADC 结果
 ADMUX=(1<<REFS0)|(adc_mux&0x0f);//选择内部 AVCC 为基准
 ADCSRA|=(1<<ADSC);//启动 AD 转换
}
/*            将 AD 转换的结果转换成电压值                */
void ADCtoBCD(unsigned int temp)
 {
  unsigned char i;
  temp=(unsigned int)(((unsigned long)((unsigned long)temp*Vref))/0x3ff);
  for(i=0;i<4;i++)//刷新 LED 缓冲区
  {
   led_buff[i]=seg_table[temp%10];
```

```
    temp=temp/10;
    }
  led_buff[2]&=0x7f; //设置小数点
  }
/*          MAIN 函数          */
void main(void)
{
 unsigned char i;
 unsigned int adc_old;
 DDRD=0xf0;
 PORTD=0xff;
 OSCCAL=osccal;//校正内部 RC 振荡的频率
 adc_mux=0;//设置为 0 通道
 adc_init();//ADC 初始化
 SEI();      //开放全局中断
 for(i=0;i<4;i++)//复位后显示一段时间 8888
  {
   led_buff[i]=seg_table[8];
  }
 for(i=0;i<200;i++)
   display();
 adc_old=0;
 adc_rel=0;
 while(1)
  {
   if(adc_old!=adc_rel)//如果 AD 转换的结果有改变，则刷新显示缓冲区
     {
      adc_old=adc_rel;
      ADCtoBCD(adc_old);
     }
   display(),
   i=PIND&0x0c;//读取按键状态
   if(i!=0x0c)//判断是否有按键按下
     {
      display();//按键消抖
      if(i==(PIND&0x0c))
        {
         CLI();//关闭全局中断
         adc_rel=0;
         adc_old=0;
         if (i==0x08)//INT0 键按下
            {
```

```
                if(adc_mux<3)
                   adc_mux++;//ADC 通道递增
               else
                   adc_mux=0;
               }
            if (i==0x04)//INT1 键按下
               {
                if(adc_mux>0)
                   adc_mux--;//ADC 通道递减
                else
                   adc_mux=3;
               }
            led_buff[0]=seg_table[adc_mux];//显示 ADC 通道号码
            led_buff[1]=0xbf;//-
            led_buff[2]=0b10001001;//h
            led_buff[3]=0b11000110;//C
            while((PIND&0x0c)!=0x0c)//检查按键释放
               display();
            SEI();         //开放全局中断
            }
        }
    }
}
```

numled.c 略

思考:

1. 尝试加入数字滤波，看看数字滤波对显示有何影响?
2. 思考一下，如何控制 AD 转换的节奏来最大程度地消除工频干扰?

5.3　USART 接口的应用

ATmega8 的 USART 是一个功能很强的通信接口，其可以做全双工的同步或异步通信。通信格式灵活多样，数据位长度可以是 5、6、7、8 或 9 位，停止位可以是 1 位或 2 位；有硬件奇偶校验、超越检测、帧错误检测；有多机通信模式；在异步通信时还有双倍速模式，可以在较低晶振频率下获得高波特率。

以下是 ATmega8 串行异步通信的源程序：

使用说明：

1. 使用 ICCAVR 的超级终端调试窗口（Terminal），进行通信调试。

2. 对 ICCAVR 的超级终端调试窗口进行设置，设置串口为 com1 或 com2，通信波特率为 19200（Tools->Environment Options...）。

3. 在打开通信口后，将 PC 屏幕光标定位于调试窗口中。

```
/*****************************************************/
/*               广州天河双龙电子公司                 */
/*               http://www.sl.com.cn                */
/*               RS232 通信演示程序                   */
/*               作者：ntzwq@wx88.net                 */
/*               2002 年 5 月 10 日                   */
/*  目标 MCU：MEGA8    晶振：外部(EXT) 8MHz           */
/*****************************************************/

#include <iom8v.h>
#define fosc 8000000 //晶振 8MHz
#define baud 19200   //通信波特率
/*       字符输出函数         */
void putchar(unsigned char c)
    {
    while (!(UCSRA&(1<<UDRE)));//判断上次发送有没有完成
    UDR=c;                          //发送数据
    }
/*       字符输入函数         */
unsigned char getchar(void)
    {
     while(!(UCSRA& (1<<RXC)));//判断有没有接收到数据
    return UDR;                //接收数据
    }
/*           带回车换行控制的字符串输出函数           */
int puts(char *s)
    {
    while (*s)
        {
        putchar(*s);
        s++;
        }
    putchar(0x0a);//回车换行
    putchar(0x0d);
    return 1;
    }
```

```
/*          不带回车换行的字符串输出函数        */
void putstr(char *s)
   {
   while (*s)
       {
       putchar(*s);
       s++;
       }
   }
/*            UART 初始化函数                  */
void uart_init(void)
{
 UCSRB=(1<<RXEN)|(1<<TXEN);//允许发送和接收
 UBRRL=(fosc/16/(baud+1))%256;//设置波特率寄存器低位字节
 UBRRH=(fosc/16/(baud+1))/256;//设置波特率寄存器高位字节
 UCSRC=(1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);//8 位数据+1 位 STOP 位
}
/*            RS232 通信 main 函数          */
void main(void)
{
 unsigned char i;
 uart_init();
 puts("广州天河双龙电子公司 RS232 通信演示程序");
 puts("HTTP://WWW.SL.COM.CN");
 puts("MAIL TO:sllg@sl.com.cn");
 while(1)
   {
    i=getchar(); //从串口接收数据
    if (i!=0)//按 PC 键盘开始测试
      {
       putstr("当前按键是: ");
       putchar(i);
       putchar(0x0a);
       putchar(0x0d);
       puts("广州天河双龙电子公司 RS232 通信演示程序");
       puts("HTTP://WWW.SL.COM.CN");
       puts("MAIL TO:sllg@sl.com.cn");
       putchar(0x0a);
       putchar(0x0d);
      }
    }
}
```

思考：

编写一个以中断方式处理的 UART 程序，比较和查询方式有什么不同？

5.4 ATmega8 实时时钟的应用

利用外接的 32.768kHz 的晶振，ATmega8 可以提供实时时钟（RTC）功能。但要注意的一点是：ATmega8 的 RTC 和一些专用的 RTC 芯片不同，即 ATmega8 没有单独的定时/计数电路，在做 RTC 时必须借助于定时器 T2 才能进行定时计数；另外在计数进位处理方面也不是由硬件完成进位，而必须借助于软件进行处理。

ATmega8 使用实时时钟的条件：1．系统时钟必须编程为使用内部 RC 振荡，这也带来了高可靠性的优点。2．T2 定时器配置为异步时钟，在 TOSC1 和 TOSC2 的位置接上 32.768kHz 的晶振，这样在使用内部 RC 振荡的条件下，也为系统提供了精确的定时基准。

以下是实时时钟实验的源程序，配合 SL-MEGA8 通信程序使用。

```
/****************************************************/
/*                 广州大河双龙电子公司               */
/*                 http://www.sl.com.cn              */
/*                 实时时钟演示程序                   */
/*                 作者：ntzwq@wx88.net              */
/*                 2002 年 5 月 12 日                */
/*   目标 MCU：MEGA8        晶振：内部 RC 8MHz        */
/****************************************************/

#include<iom8v.h>          //包含 ATmega8 的头文件
#include<macros.h>         //包含一些常用宏的定义
#include"uart.h"           //项目包含的另一个文件(uart.c)中的函数声明
unsigned char hour;        //定义小时变量
unsigned char minute;      //定义分钟变量
unsigned char second;      //定义秒变量
/*          定时进位处理函数              */
void time(void)
     {
      if (second<60)
          return;
      else                     //秒计数满 60，分钟计数进位
        {
         second-=60;
```

```
        if (minute<59)
           minute++;
        else             //分钟计数满 60，小时计数进位
          {
           minute=0;
            if (hour<23)//24 小时制小时处理
               hour++;
            else
               hour=0;
          }
        }
    }
/*          定时器 T2 初始化函数               */
void timer2_init(void)
{
 TCCR2 = 0x00;
 ASSR  = 1<<AS2;     //异步时钟
 TCNT2 = 0xE0;       //定时时间 1 秒
 TCCR2 = (1<<CS22)|(1<<CS21)|(1<<CS20); //预分频比 1024
}
/*          定时器 T2 溢出中断处理函数          */
#pragma interrupt_handler timer2_ovf_isr:iv_TIMER2_OVF
void timer2_ovf_isr(void)
{
 TCNT2 = 0xE0;  //重装定时常数
 second++;      //秒计数器加 1
}
/*          将时间转换成 ASCII 码后从串口输出                  */
void put_time(void)
 {
  putchar(hour/10+0x30);       //输出小时高位 ACSII 码
  putchar(hour%10+0x30);       //输出小时低位 ACSII 码
  putchar(':');                //输出时间分隔符
  putchar(minute/10+0x30);     //输出分钟高位 ACSII 码
  putchar(minute%10+0x30);     //输出分钟高位 ACSII 码
  putchar(':');                //输出时间分隔符
  putchar(second/10+0x30);     //输出秒高位 ACSII 码
  putchar(second%10+0x30);     //输出秒低位 ACSII 码
 }
/*          UART 接收中断处理函数，可以从 PC 机调整当前时间              */
#pragma interrupt_handler uart0_rx_isr:iv_USART_RXC
void uart0_rx_isr(void)
```

```
{
 unsigned char i;
 i=UDR;
 if (i=='t')//判断是否是调整时间，并且将 ASCII 码转换成数字
   {
    hour=(getchar()-0x30)*10;          //调整小时高位
     hour=hour+(getchar()-0x30);       //调整小时低位
     minute=(getchar()-0x30)*10;       //调整分钟高位
     minute=minute+(getchar()-0x30);//调整分钟低位
     second=(getchar()-0x30)*10;       //调整秒高位
     second=second+(getchar()-0x30);//调整秒低位
   }
}
/*            main 函数           */
void main(void)
{
 unsigned char second_old;
 uart_init();           //初始化 UART
 timer2_init();         //初始化定时器 T2
 TIMSK =1<<TOIE2;       //开放 T2 溢出中断
 SEI();                 //开放全局中断
 while(1)
   {
    if (second!=second_old)       //判断时间有没有更新
        {
          time();                 //处理时间进位
          second_old=second;      //保存当前时间
          put_time();             //从串口输出时间
        }
   }
}
```

UART.C 程序略。

思考：

1. 用结构的形式定义时间参数，将程序改写一下，比较有什么优缺点？
2. 本程序是 24 小时制定时，尝试改为 12 小时定时程序。

5.5 BOOT 引导区的应用

ATmega8 的程序区可以分成两段：应用程序区和 BOOT 引导区。利用 BOOT 引导区可以做一些特殊的应用，如在应用编程（IAP）、系统智能升级等。对 ATmega8 的 BOOT 引导区注意这样几点：

1. ATmega8 的 BOOT 引导区的大小是可以通过熔丝位（BOOTSZ0、BOOTSZ1）进行调整的。

2. 如果想从 BOOT 启动程序，应该对熔丝位（BOOTRST）编程。

3. 如果用户不想使用 BOOT 引导区，那么 BOOT 引导区也可以作为正常的程序区来使用。

4. 通过熔丝位（BLB01、02 和 BLB11、12），用户可以对应用程序区和 BOOT 引导区进行不同的安全保护。

以下是 ATmega8 的 IAP 实验的源程序：

```
/*****************************************************/
/*                  广州天河双龙电子公司                */
/*                  http://www.sl.com.cn               */
/*                 BOOT 引导 IAP 应用演示程序            */
/*                 作者：ntzwq@wx88.net                 */
/*                    2002 年 5 月 11 日                */
/*    目标 MCU：MEGA8      晶振：外部(EXT) 8MHz         */
/*****************************************************/

#include "self.h"         //定义了一些参数
#include "assembly.h"     //包含了项目中另一个汇编语言文件(assembly.s)的函数声明
#include<iom8v.h>
#define fosc 8000000      //晶振频率 8MHz
#define baud 19200        //通信波特率
/*        字符输出函数            */
void sendchar(unsigned char c)
    {
    while (!(UCSRA&(1<<UDRE)));
     UDR=c;
}
/*        字符输入函数            */
unsigned char recchar(void)
```

```
    {
     while(!(UCSRA& (1<<RXC)));
    return UDR;
    }
/*            UART 初始化函数                    */
void uart_init(void)
{
UCSRB=(1<<RXEN)|(1<<TXEN);        //允许发送和接收
  UBRRL=(fosc/16/(baud+1))%256; //设置波特率寄存器低位字节
  UBRRH=(fosc/16/(baud+1))/256;  //设置波特率寄存器高位字节
  UCSRC=(1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);//设置通信格式 8 位数据+1 位 STOP 位
}
/*                  IAP 通信及编程函数                    */
void iap(void)
{
    unsigned int intval,address,data;
    unsigned char val, ldata;
     for(;;)
      {
       val=recchar();
       if(val=='a')                                //判断是否支持地址自动增量
          {
           sendchar('Y');                          //确定支持地址自动增量
          }
       else if(val=='A')                           //设置地址
          {
           address=recchar();                      //读取字节地址高 8 位
           address=(address<<8)|recchar();         //读取字节地址低 8 位
           address=address<<1;                     //转换成字地址
           sendchar('\r');                         //发送应答
          }
        else if(val=='c')                          //写程序存储器低位字节
          {
           ldata=recchar();
           sendchar('\r');
          }
        else if(val== 'C')                         //写程序存储器高位字节
          {
           data=ldata|(recchar()<<8);
           fill_temp_buffer(data,(address)); //填充临时缓冲页
           address+=2;
           sendchar(''r');
```

```
}
else if(val=='e')                          //芯片擦除
{
 for(address=0;address < APP_END;address += PAGESIZE)//根据页面数循环
  {
   write_page(address,0x03);               //执行页擦除
}
 sendchar('\r');
}
else if(val=='l')                          //写加密位
{
 write_lock_bits(recchar());
 sendchar('\r');
}
else if(val== 'm')                         //写执行页
{
 write_page((address),0x05);
 sendchar('\r');
}
else if((val=='P')||(val=='L'))            //进入编程模式
{
 sendchar('\r');
}
else if (val=='p')                         //询问编程器类型
{
 sendchar('S');
}
else if(val=='R')                          //读程序存储器
{
  if (address>=APP_END)
   {
    sendchar(0xff);
    sendchar(0xff);
    address+=2;
   }
  else
   {
   intval=read_program_memory(address,0x00);
   sendchar((char)(intval>>8));            //发送高位字节
   sendchar((char)intval);                 //发送低位字节
   address+=2;
   }
```

```
}
else if (val == 'D')                                          //写 E²PROM 存储器
{
 EEARL = address;
 EEARH = (address >> 8);
 address++;
 EEDR = recchar();
 EECR |= (1<<EEMWE);
 EECR |= (1<<EEWE);
 while (EECR & (1<<EEWE))
    ;
 sendchar('\r');
 }
else if (val == 'd')                                          //读 E²PROM 存储器
 {
  EEARL = address;
  EEARH = (address >> 8);
  address++;
  EECR |= (1<<EERE);
  sendchar(EEDR);
 }
else if(val=='F')                                             //读熔丝位
 {
  sendchar(read_program_memory(0x0000,0x09));
 }
else if(val=='r')                                             //读加密位
 {
  sendchar(read_program_memory(0x0001,0x09));
 }
else if(val=='N')                                             //读熔丝位高位字节
 {
  sendchar(read_program_memory(0x0003,0x09));
  }
else if(val=='t')                                             //询问支持芯片型号
 {
  sendchar(device);
  sendchar(0);
  }
else if ((val=='x')||(val=='y')||(val=='T'))                  //亮灯等其他处理
  {
   recchar();
   sendchar('\r');
```

```
        }
      else if (val=='S')                    //询问软件标识
        {
         sendchar('A');
         sendchar('V');
         sendchar('R');
         sendchar('B');
         sendchar('O');
         sendchar('O');
         sendchar('T');
        }
      else if (val=='V')                    //询问软件版本号
        {
         sendchar('1');
         sendchar('0');
        }
      else if (val=='s')                    //询问芯片 ID 码
        {
         sendchar(sig_byte3);
         sendchar(sig_byte2);
         sendchar(sig_byte1);
        }
      else if(val!=0x1b)                    //检测通信启动标志
        {
         sendchar('?');
        }
     }
}
/*          main 函数          */
void main(void)
{
   void (*funcptr)( void ) = 0x0000;        //定义函数指针
   DDRD=0;                                  //初始化 D 口
   PORTD=0xff;
   uart_init();                             //初始化 UART
   if ((PIND&0x04)==0)  //上电或复位后，如果 PIND.4 是低电平，则进入 IAP 程序
     {
      DDRD=0;
      iap();
     }
   else                         //否则执行用户程序
     funcptr();
```

```
}
```

assembly.s 程序略

思考：

1．从这个范例程序中学习 C 和汇编混合编程的方法，自己尝试写一个混合编程的程序。

2．这个程序修改后可适用于 ATmega163/323/128 等支持 BOOT 功能的 I^2C，尝试自己写一个。