产品特性

- 高性能、低功耗的 8 位 AVR® 微处理器
- 先进的 RISC 结构
  - 131 条指令 – 大多数指令执行时间为单个时钟周期
  - 32 个 8 位通用工作寄存器
  - 全静态工作
  - 工作于 16 MHz 时性能高达 16 MIPS
  - 只需两个时钟周期的硬件乘法器
- 非易失性程序和数据存储器
  - 32K 字节的系统内可编程 Flash
    擦写寿命 : 10,000 次
  - 具有独立锁定位的可选 Boot 代码区
    通过片上 Boot 程序实现系统内编程
    真正的同时读写操作
  - 1024 字节的 EEPROM
    擦写寿命 : 100,000 次
  - 2K 字节片内 SRAM
  - 可以对锁定位进行编程以实现用户程序的加密
- JTAG 接口 (与 IEEE 1149.1 标准兼容)
  - 符合 JTAG 标准的边界扫描功能
  - 支持扩展的片内调试功能
  - 通过 JTAG 接口实现对 Flash、 EEPROM、熔丝位和锁定位的编程
- 外设特点
  - 两个具有独立预分频器和比较器功能的 8 位定时器 / 计数器
  - 一个具有预分频器、比较功能和捕捉功能的 16 位定时器 / 计数器
  - 具有独立振荡器的实时计数器 RTC
  - 四通道 PWM
  - 8 路 10 位 ADC
    8 个单端通道
    TQFP 封装的 7 个差分通道
    2 个具有可编程增益 （1x, 10x, 或 200x）的差分通道
  - 面向字节的两线接口
  - 可编程的串行 USART
  - 可工作于主机 / 从机模式的 SPI 串行接口
  - 具有独立片内振荡器的可编程看门狗定时器
  - 片内模拟比较器
- 特殊的处理器特点
  - 上电复位以及可编程的掉电检测
  - 片内经过标定的 RC 振荡器
  - 片内 / 片外中断源
  - 6 种睡眠模式 : 空闲模式、ADC 噪声抑制模式、省电模式、掉电模式、Standby 模式以及扩展的 Standby 模式
- I/O 和封装
  - 32 个可编程的 I/O 口
  - 40 引脚 PDIP 封装 , 44 引脚 TQFP 封装 , 与 44 引脚 MLF 封装
- 工作电压
  - ATmega32L：2.7 - 5.5V
  - ATmega32：4.5 - 5.5V
- 速度等级
  - ATmega32L：0 - 8 MHz
  - ATmega32：0 - 16 MHz
- ATmega32L 在 1 MHz, 3V, 25°C 时的功耗
  - 正常模式 : 1.1 mA
  - 空闲模式 : 0.35 mA
  - 掉电模式 : < 1 µA

具有 32KB 系统内可编程 Flash 的 8 位 AVR® 微控制器

ATmega32
ATmega32L

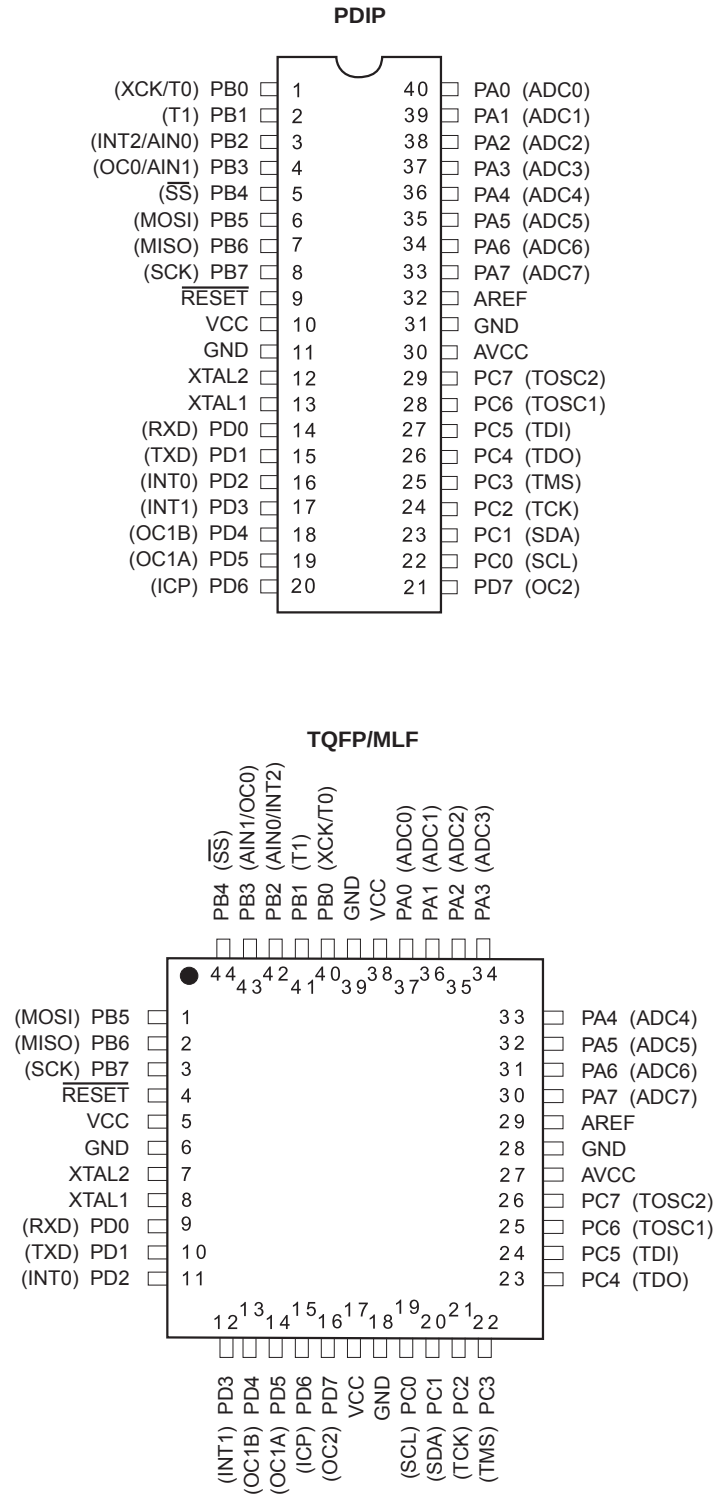
初稿

本文是英文数据手册的中文翻译，其目的是方便中国用户的阅读。它无法自动跟随原稿的更新，同时也可能存在翻译上的错误。读者应该以英文原稿为参考以获得更准确的信息。

2503F–AVR–12/03

引脚配置

Figure 1. ATmega32 的引脚



声明

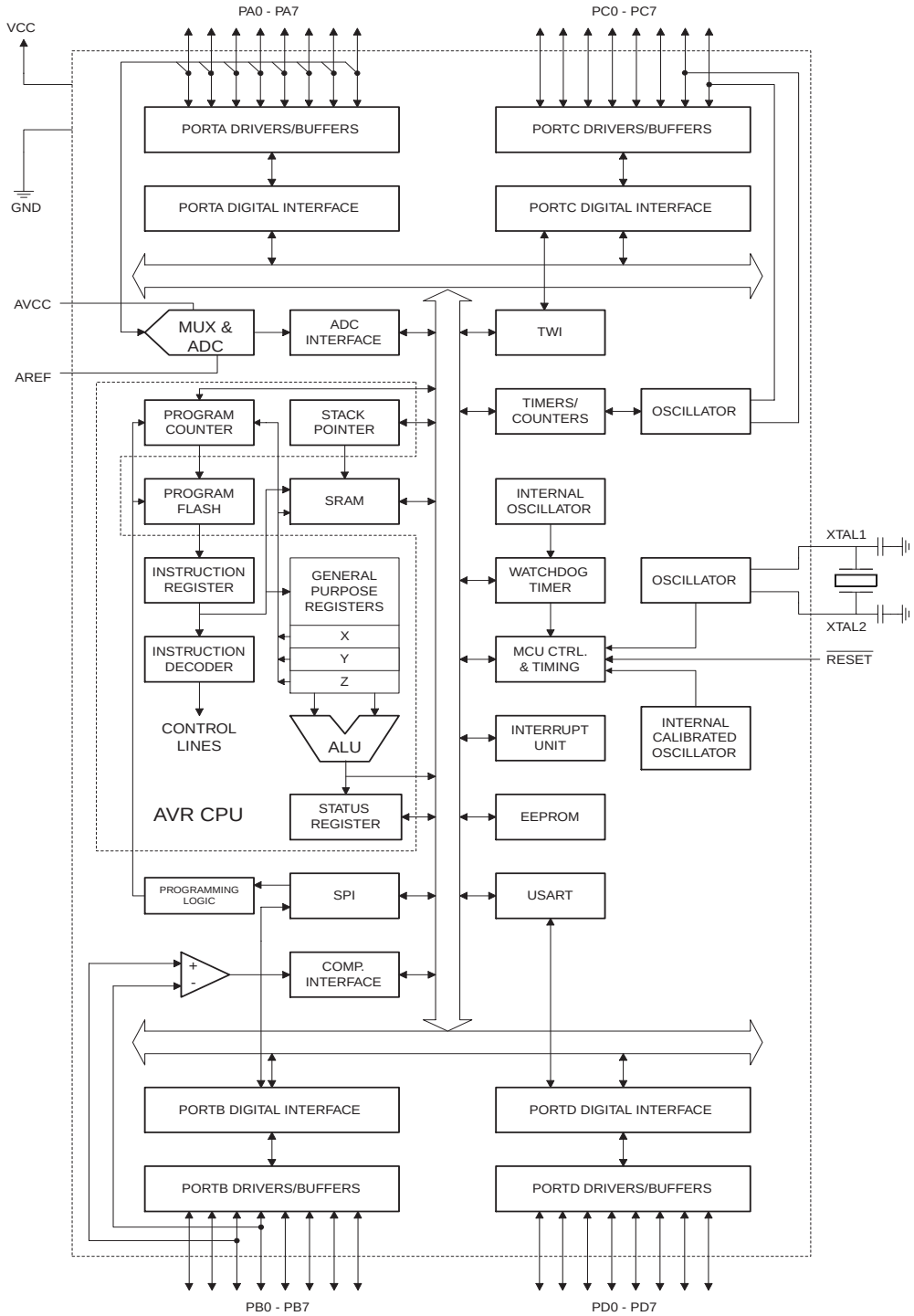
本数据手册的典型值来源于对器件的仿真，以及其他基于相同产生工艺的 AVR 微控制器的标定特性。本器件经过特性化之后将给出实际的最大值和最小值。

综述

ATmega32 是基于增强的 AVR RISC 结构的低功耗 8 位 CMOS 微控制器。由于其先进的指令集以及单时钟周期指令执行时间，ATmega32 的数据吞吐率高达 1 MIPS/MHz，从而可以缓减系统在功耗和处理速度之间的矛盾。

方框图

Figure 2. 结构框图



AVR 内核具有丰富的指令集和 32 个通用工作寄存器。所有的寄存器都直接与算逻单元 (ALU) 相连接，使得一条指令可以在一个时钟周期内同时访问两个独立的寄存器。这种结构大大提高了代码效率，并且具有比普通的 CISC 微控制器最高至 10 倍的数据吞吐率。

ATmega32有如下特点:32K 字节的系统内可编程Flash(具有同时读写的能力 ,即RWW)，1024 字节 EEPROM，2K 字节 SRAM ,32 个通用 I/O 口线 ,32 个通用工作寄存器 ,用于边界扫描的 JTAG 接口，支持片内调试与编程，三个具有比较模式的灵活的定时器 / 计数器 (T/C), 片内 / 外中断，可编程串行 USART，面向字节的两线串行接口，8 路 10 位具有可选差分输入级可编程增益 (TQFP 封装) 的 ADC ,具有片内振荡器的可编程看门狗定时器，一个 SPI 串行端口 ,以及六个可以通过软件进行选择的省电模式。工作于空闲模式时 CPU 停止工作，而 USART、两线接口、A/D 转换器、SRAM、T/C、SPI 端口以及中断系统继续工作；掉电模式时晶体振荡器停止振荡，所有功能除了中断和硬件复位之外都停止工作；在省电模式下，异步定时器继续运行，允许用户保持一个时间基准，而其余功能模块处于休眠状态； ADC 噪声抑制模式时终止 CPU 和除了异步定时器与 ADC 以外所有 I/O 模块的工作，以降低 ADC 转换时的开关噪声； Standby 模式下只有晶体或谐振振荡器运行，其余功能模块处于休眠状态，使得器件只消耗极少的电流，同时具有快速启动能力；扩展 Standby 模式下则允许振荡器和异步定时器继续工作。

本芯片是以 Atmel 高密度非易失性存储器技术生产的。片内 ISP Flash 允许程序存储器通过 ISP 串行接口，或者通用编程器进行编程，也可以通过运行于 AVR 内核之中的引导程序进行编程。引导程序可以使用任意接口将应用程序下载到应用Flash存储区(Application Flash Memory)。在更新应用Flash存储区时引导Flash区(Boot Flash Memory)的程序继续运行，实现了 RWW 操作。通过将 8 位 RISC CPU 与系统内可编程的 Flash 集成在一个芯片内， ATmega32 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。

ATmega32 具有一整套的编程与系统开发工具，包括 : C 语言编译器、宏汇编、程序调试器 / 软件仿真器、仿真器及评估板。

引脚说明

VCC 数字电路的电源

GND 地

端口 A(PA7..PA0) 端口 A 做为 A/D 转换器的模拟输入端。

端口 A 为 8 位双向 I/O 口 ,具有可编程的内部上拉电阻。其输出缓冲器具有对称的驱动特性，可以输出和吸收大电流。作为输入使用时，若内部上拉电阻使能，端口被外部电路拉低时将输出电流。在复位过程中，即使系统时钟还未起振，端口 A 处于高阻状态。

端口 B(PB7..PB0) 端口 B 为 8 位双向 I/O 口 ,具有可编程的内部上拉电阻。其输出缓冲器具有对称的驱动特性，可以输出和吸收大电流。作为输入使用时，若内部上拉电阻使能，端口被外部电路拉低时将输出电流。在复位过程中，即使系统时钟还未起振，端口 B 处于高阻状态。

端口 B 也可以用做其他不同的特殊功能，请参见 P55。

端口 C(PC7..PC0) 端口 C 为 8 位双向 I/O 口 ,具有可编程的内部上拉电阻。其输出缓冲器具有对称的驱动特性，可以输出和吸收大电流。作为输入使用时，若内部上拉电阻使能，端口被外部电路拉低时将输出电流。在复位过程中，即使系统时钟还未起振，端口 C 处于高阻状态。如果 JTAG接口使能 ,即使复位出现引脚 PC5(TDI)、PC3(TMS)与 PC2(TCK)的上拉电阻被激活。

除去移出数据的 TAP 态外， TD0 引脚为高阻态。

端口 C 也可以用做其他不同的特殊功能，请参见 P58。

端口 D(PD7..PD0)

端口 D 为 8 位双向 I/O 口，具有可编程的内部上拉电阻。其输出缓冲器具有对称的驱动特性，可以输出和吸收大电流。作为输入使用时，若内部上拉电阻使能，则端口被外部电路拉低时将输出电流。在复位过程中，即使系统时钟还未起振，端口 D 处于高阻状态。

端口 D 也可以用做其他不同的特殊功能，请参见 P60。

RESET

复位输入引脚。持续时间超过最小门限时间的低电平将引起系统复位。门限时间见 P35 Table 15 。持续时间小于门限间的脉冲不能保证可靠复位。

XTAL1

反向振荡放大器与片内时钟操作电路的输入端。

XTAL2

反向振荡放大器的输出端。

AVCC

AVCC是端口A与A/D转换器的电源。不使用ADC时，该引脚应直接与V_{CC}连接。使用ADC时应通过一个低通滤波器与 V_{CC} 连接。

AREF

A/D 的模拟基准输入引脚。

代码例子

本数据手册包含了一些简单的代码例子以说明如何使用芯片各个不同的功能模块。这些例子都假定在编译之前已经包含了正确的头文件。有些 C 编译器在头文件里并没有包含位定义，而且各个 C 编译器对中断处理有自己不同的处理方式。请注意查阅相关文档以获取具体的信息。

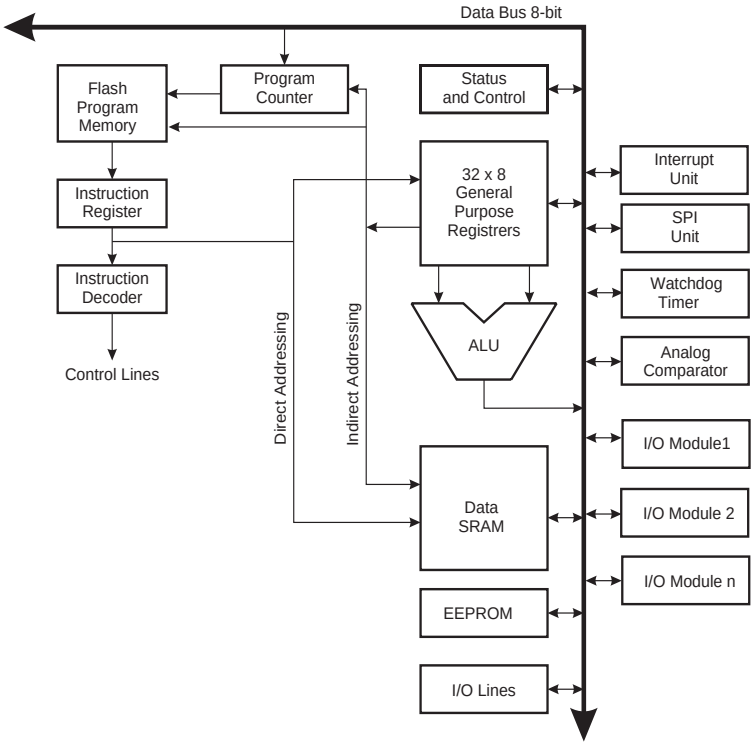
AVR CPU 内核

介绍

本节从总体上讨论 AVR 内核的结构。CPU 的主要任务是保证程序的正确执行。因此它必须能够访问存储器、执行运算、控制外设以及处理中断。

结构综述

Figure 3. AVR 结构的方框图



为了获得最高的性能以及并行性， AVR 采用了 Harvard 结构，具有独立的数据和程序总线。程序存储器里的指令通过一级流水线运行。 CPU 在执行一条指令的同时读取下一条指令（在本文称为预取）。这个概念实现了指令的单时钟周期运行。程序存储器是可以在线编程的 Flash 。

快速访问寄存器文件包括 32 个 8 位通用工作寄存器，访问时间为一个时钟周期。从而实现了单时钟周期的 ALU 操作。在典型的 ALU 操作中，两个位于寄存器文件中的操作数同时被访问，然后执行运算，结果再被送回到寄存器文件。整个过程仅需一个时钟周期。

寄存器文件里有 6 个寄存器可以用作 3 个 16 位的间接寻址寄存器指针以寻址数据空间，实现高效的地址运算。其中一个指针还可以作为程序存储器查询表的地址指针。这些附加的功能寄存器即为 16 位的 X、 Y、 Z 寄存器。

ALU支持寄存器之间以及寄存器和常数之间的算术和逻辑运算。ALU也可以执行单寄存器操作。运算完成之后状态寄存器的内容得到更新以反映操作结果。

程序流程通过有 / 无条件的跳转指令和调用指令来控制，从而直接寻址整个地址空间。大多数指令长度为 16 位，亦即每个程序存储器地址都包含一条 16 位或 32 位的指令。

程序存储器空间分为两个区：引导程序区 (Boot 区) 和应用程序区。这两个区都有专门的锁定位以实现读和读 / 写保护。用于写应用程序区的 SPM 指令必须位于引导程序区。

在中断和调用子程序时返回地址的程序计数器 (PC) 保存于堆栈之中。堆栈位于通用数据 SRAM，因此其深度仅受限于 SRAM 的大小。在复位例程里用户首先要初始化堆栈指针

SP。这个指针位于 I/O 空间，可以进行读写访问。数据 SRAM 可以通过 5 种不同的寻址模式进行访问。

AVR 存储器空间为线性的平面结构。

AVR有一个灵活的中断模块。控制寄存器位于I/O空间。状态寄存器里有全局中断使能位。每个中断在中断向量表里都有独立的中断向量。各个中断的优先级与其在中断向量表的位置有关，中断向量地址越低，优先级越高。

I/O 存储器空间包含 64 个可以直接寻址的地址，作为 CPU 外设的控制寄存器、SPI，以及其他 I/O 功能。映射到数据空间即为寄存器文件之后的地址 0x20 - 0x5F。

ALU - 算术逻辑单元

AVR ALU 与 32 个通用工作寄存器直接相连。寄存器与寄存器之间、寄存器与立即数之间的 ALU 运算只需要一个时钟周期。ALU 操作分为 3 类：算术、逻辑和位操作。此外还提供了支持无 / 有符号数和分数乘法的乘法器。具体请参见指令集。

状态寄存器

状态寄存器包含了最近执行的算术指令的结果信息。这些信息可以用来改变程序流程以实现条件操作。如指令集所述，所有 ALU 运算都将影响状态寄存器的内容。这样，在许多情况下就不需要专门的比较指令了，从而使系统运行更快速，代码效率更高。

在进入中断服务程序时状态寄存器不会自动保存，中断返回时也不会自动恢复。这些工作需要软件来处理。

AVR 中断寄存器 SREG 定义如下：

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – I: 全局中断使能**

I 置位时使能全局中断。单独的中断使能由其他独立的控制寄存器控制。如果 I 清零，则不论单独中断标志置位与否，都不会产生中断。任意一个中断发生后 I 清零，而执行 RETI 指令后 I 恢复置位以使能中断。 I 也可以通过 SEI 和 CLI 指令来置位和清零。

- **Bit 6 – T: 位拷贝存储**

位拷贝指令 BLD 和 BST 利用 T 作为目的或源地址。BST 把寄存器的某一位拷贝到 T，而 BLD 把 T 拷贝到寄存器的某一位。

- **Bit 5 – H: 半进位标志**

半进位标志 H 表示算术操作发生了半进位。此标志对于 BCD 运算非常有用。详见指令集的说明。

- **Bit 4 – S: 符号位 , S = N ⊕ V**

S 为负数标志 N 与 2 的补码溢出标志 V 的异或。详见指令集的说明。

- **Bit 3 – V: 2 的补码溢出标志**

支持 2 的补码运算。详见指令集的说明。

- **Bit 2 – N: 负数标志**

表明算术或逻辑操作结果为负。详见指令集的说明。

- **Bit 1 – Z: 零标志**

表明算术或逻辑操作结果为零。详见指令集的说明。

- **Bit 0 – C: 进位标志**

表明算术或逻辑操作发生了进位。详见指令集的说明。

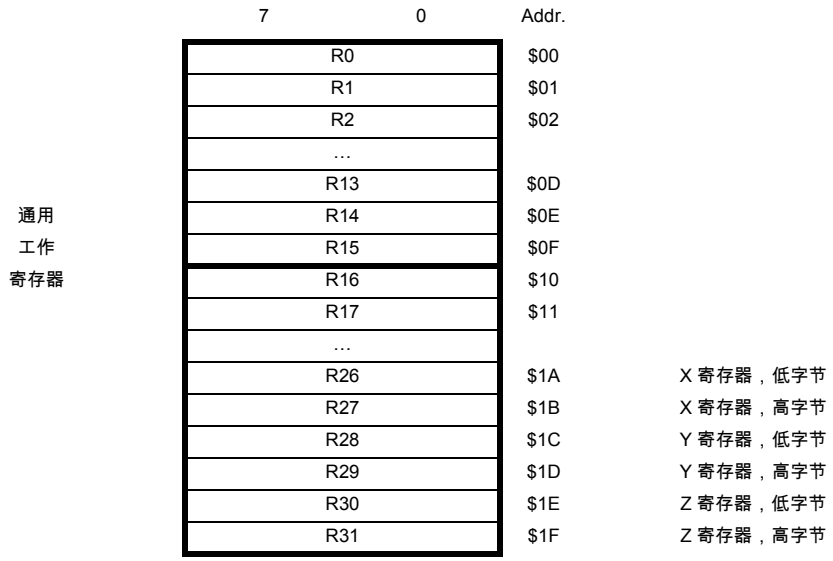
通用寄存器文件

寄存器文件针对 AVR 增强型 RISC 指令集做了优化。为了获得需要的性能和灵活性，寄存器文件支持以下的输入 / 输出方案：

- 输出一个 8 位操作数，输入一个 8 位结果
- 输出两个 8 位位操作数，输入一个 8 位结果
- 输出两个 8 位位操作数，输入一个 16 位结果
- 输出一个 16 位位操作数，输入一个 16 位结果

Figure 4 为 CPU 32 个通用工作寄存器的结构。

Figure 4. AVR CPU 通用工作寄存器



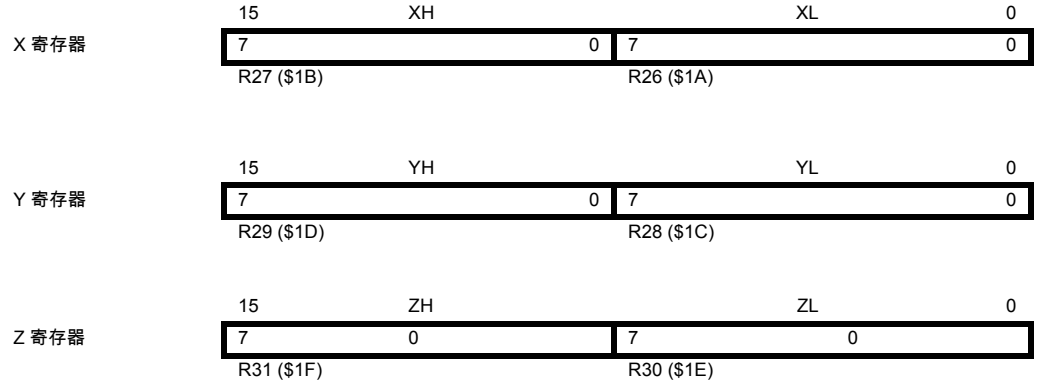
大多数操作寄存器文件的指令都可以直接访问所有的寄存器，而且多数这样的指令的执行时间为单个时钟周期。

如 Figure 4 所示，每个寄存器都有一个数据内存地址，将他们直接映射到用户数据空间的头 32 个地址。虽然寄存器文件的物理实现不是 SRAM，这种内存组织方式在访问寄存器方面具有极大的灵活性，因为 X、 Y、 Z 寄存器可以设置为指向任意寄存器的指针。

X、Y、Z 寄存器

寄存器 R26..R31 除了用作通用寄存器外，还可以作为数据间接寻址用的地址指针。这三个间接寻址寄存器示于 Figure 5。

Figure 5. X、 Y、 Z 寄存器



在不同的寻址模式中，这些地址寄存器可以实现固定偏移量，自动加一和自动减一功能。具体细节请参见指令集。

堆栈指针

堆栈指针主要用来保存临时数据、局部变量和中断 / 子程序的返回地址。堆栈指针总是指向堆栈的顶部。要注意 AVR 的堆栈是向下生长的，即新数据推入堆栈时，堆栈指针的数值将减小。

堆栈指针指向数据 SRAM 堆栈区。在此聚集了子程序堆栈和中断堆栈。调用子程序和使能中断之前必须定义堆栈空间，且堆栈指针必须指向高于 0x60 的地址空间。使用 PUSH 指令将数据推入堆栈时指针减一；而子程序或中断返回地址推入堆栈时指针将减二。使用 POP 指令将数据弹出堆栈时，堆栈指针加一；而用 RET 或 RETI 指令从子程序或中断返回时堆栈指针加二。

AVR的堆栈指针由I/O空间中的两个8位寄存器实现。实际使用的位数与具体器件有关。请注意某些 AVR 器件的数据区太小，用 SPL 就足够了。此时将不给出 SPH 寄存器。

Bit	15	14	13	12	11	10	9	8	
	SP15	**SP14**	**SP13**	**SP12**	**SP11**	**SP10**	**SP9**	**SP8**	**SPH**
	SP7	**SP6**	**SP5**	**SP4**	**SP3**	**SP2**	**SP1**	**SP0**	**SPL**
	7	6	5	4	3	2	1	0	
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

指令执行时序

这一节介绍指令执行过程中的访问时序。AVR CPU 由系统时钟 clk_{CPU} 驱动。此时钟直接来自选定的时钟源。芯片内部不对此时钟进行分频。

Figure 6 说明了由 Harvard 结构决定的并行取指和指令执行，以及可以进行快速访问的寄存器文件的概念。这是一个基本的流水线概念，性能高达 1 MIPS/MHz，具有优良的性价比、功能 / 时钟比、功能 / 功耗比。

Figure 6. 并行取指和指令执行

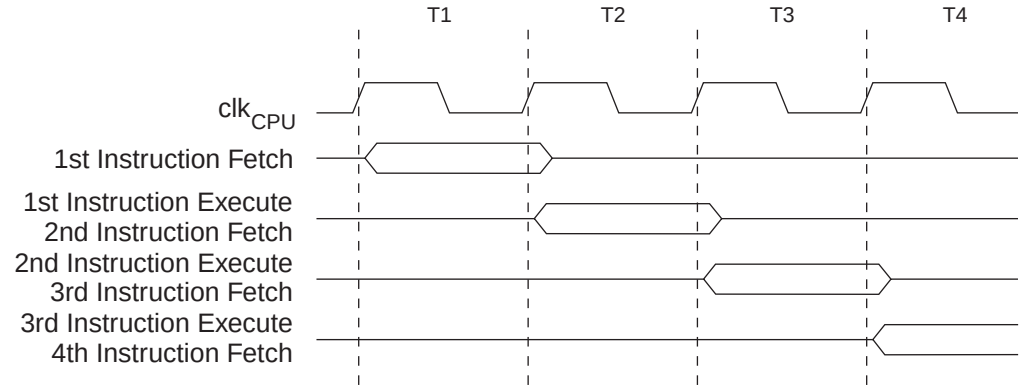
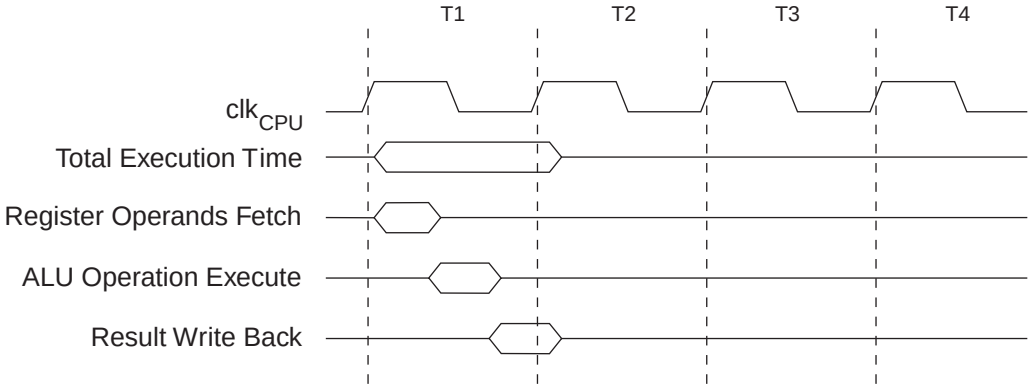


Figure 7 演示的是寄存器文件内部访问时序。在一个时钟周期里，ALU 可以同时对两个寄存器操作数进行操作，同时将结果保存到目的寄存器中去。

Figure 7. 单时钟周期 ALU 操作



复位与中断处理

AVR 有不同的中断源。每个中断和复位在程序空间都有独立的中断向量。所有的中断事件都有自己的使能位。当使能位置位，且状态寄存器的全局中断使能位 I 也置位时，中断可以发生。根据程序计数器 PC 的不同，在引导锁定位 BLB02 或 BLB12 被编程的情况下，中断可能被自动禁止。这个特性提高了软件的安全性。详见 P240 “ 存储器编程 ” 的描述。

程序存储区的最低地址缺省为复位向量和中断向量。完整的向量列表请参见P42 “中断”。列表也决定了不同中断的优先级。向量所在的地址越低，优先级越高。 RESET 具有最高的优先级，第二个为 INT0 – 外部中断请求 0。通过置位通用中断控制寄存器 (GICR) 的 IVSEL，中断向量可以移至引导Flash的起始处，参见P42 “中断” 。编程熔丝位BOOTRST 也可以将复位向量移至引导 Flash 的起始处。具体参见 P228 “ 支持引导装入程序 – 在写的同时可以读 (RWW, Read-While-Write) 的自我编程能力 ”。

任一中断发生时全局中断使能位 I 被清零，从而禁止了所有其他的中断。用户软件可以在中断程序里置位 I 来实现中断嵌套。此时所有的中断都可以中断当前的中断服务程序。执行 RETI 指令后 I 自动置位。

从根本上说有两种类型的中断。第一种由事件触发并置位中断标志。对于这些中断，程序计数器跳转到实际的中断向量以执行中断处理程序，同时硬件将清除相应的中断标志。中断标志也可以通过对其写 ”1” 的方式来清除。当中断发生后，如果相应的中断使能位为 "0"，则中断标志位置位，并一直保持到中断执行，或者被软件清除。类似的，如果全局中断标志被清零，则所有已发生的中断都不会被执行，直到 I 置位。然后挂起的各个中断按中断优先级依次执行。

第二种类型的中断则是只要中断条件满足，就会一直触发。这些中断不需要中断标志。若中断条件在中断使能之前就消失了，中断不会被触发。

AVR 退出中断后总是回到主程序并至少执行一条指令才可以去执行其他被挂起的中断。

要注意的是，进入中断服务程序时状态寄存器不会自动保存，中断返回时也不会自动恢复。这些工作必须由用户通过软件来完成。

使用 CLI 指令来禁止中断时，中断禁止立即生效。没有中断可以在执行 CLI 指令后发生，即使它是在执行 CLI 指令的同时发生的。下面的例子说明了如何在写 EEPROM 时使用这个指令来防止中断发生以避免对 EEPROM 内容的破坏。

汇编代码例程

```
in  r16, SREG      ; 保存SREG
cli     ; 禁止中断
sbi EECR, EEMWE    ; 启动 EEPROM 写操作
sbi EECR, EEWE
out SREG, r16      ; 恢复SREG (I 位)
```

C 代码例程

```
char cSREG;
cSREG = SREG;  /* 保存SREG */
/* 禁止中断 */
_CLI();
EECR |= (1<<EEMWE); /* 启动 EEPROM 写操作 */
EECR |= (1<<EEWE);
SREG = cSREG; /* 恢复SREG (I 位) */
```

使用 SEI 指令使能中断时，紧跟其后的第一条指令在执行任何中断之前一定会首先得到执行。

汇编代码例程

```
sei   ; 置位全局中断使能标志
sleep ; 进入休眠模式，等待中断发生
; 注意： 在执行任何被挂起的中断之前 MCU 将首先进入休眠模式
```

C 代码例程

```
_SEI(); /* 置位全局中断使能标志 */
_SLEEP(); /* 进入休眠模式，等待中断发生 */
/* 注意： 在执行任何被挂起的中断之前 MCU 将首先进入休眠模式 */
```

中断响应时间

AVR 中断响应时间最少为 4 个时钟周期。4 个时钟周期后，程序跳转到实际的中断处理例程。在这 4 个时钟期期间 PC 自动入栈。在通常情况下，中断向量为一个跳转指令，此跳转需要 3 个时钟周期。如果中断在一个多时钟周期指令执行期间发生，则在此多周期指令执行完毕后 MCU 才会执行中断程序。若中断发生时 MCU 处于休眠模式，中断响应时间还需增加 4 个时钟周期。此外还要考虑到不同的休眠模式所需要的启动时间。

中断返回需要 4 个时钟。在此期间 PC(两个字节) 将被弹出栈，堆栈指针加二，状态寄存器 SREG 的 I 置位。

AVR ATmega32 存储器

本节讲述 ATmega32 的存储器。AVR 结构具有两个主要的存储器空间：数据存储器空间和程序存储器空间。此外，ATmega32 还有 EEPROM 存储器以保存数据。这三个存储器空间都为线性的平面结构。

系统内可编程的 Flash 程序存储器

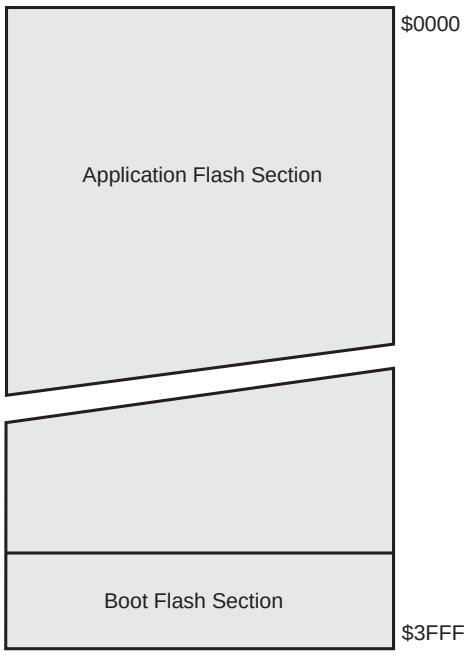
ATmega32 具有 32K 字节的在线编程 Flash，用于存放程序指令代码。因为所有的 AVR 指令为 16 位或 32 位，故而 Flash 组织成 16K x 16 位的形式。用户程序的安全性要根据 Flash 程序存储器的两个区：引导 (Boot) 程序区和应用程序区，分开来考虑。

Flash存储器至少可以擦写10,000次。ATmega32的程序计数器(PC)为14 位，因此可以寻址 16K 字的程序存储器空间。引导程序区以及相关的软件安全锁定位请参见 P228 “ 支持引导装入程序 – 在写的同时可以读(RWW, Read-While-Write)的自我编程能力”，而P240 “ 存储器编程 ” 详述了用 SPI 或 JTAG 接口实现对 Flash 的串行下载。

常数可以保存于整个程序存储器地址空间 (参考 LPM 加载程序存储器指令的说明)。

取指与执行时序图请参见 P11 “ 指令执行时序 ”。

Figure 8. 程序存储器映像



SRAM 数据存储器

Figure 9 给出了 ATmega32 SRAM 空间的组织结构。

前 2144 个数据存储器包括了寄存器文件、 I/O 存储器及内部数据 SRAM。起始的 96 个地址为寄存器文件与 I/O 存储器，接着是 2048 字节的内部数据 SRAM。

数据存储器的寻址方式分为 5 种：直接寻址、带偏移量的间接寻址、间接寻址、带预减量的间接寻址和带后增量的间接寻址。寄存器文件中的寄存器 R26 到 R31 为间接寻址的指针寄存器。

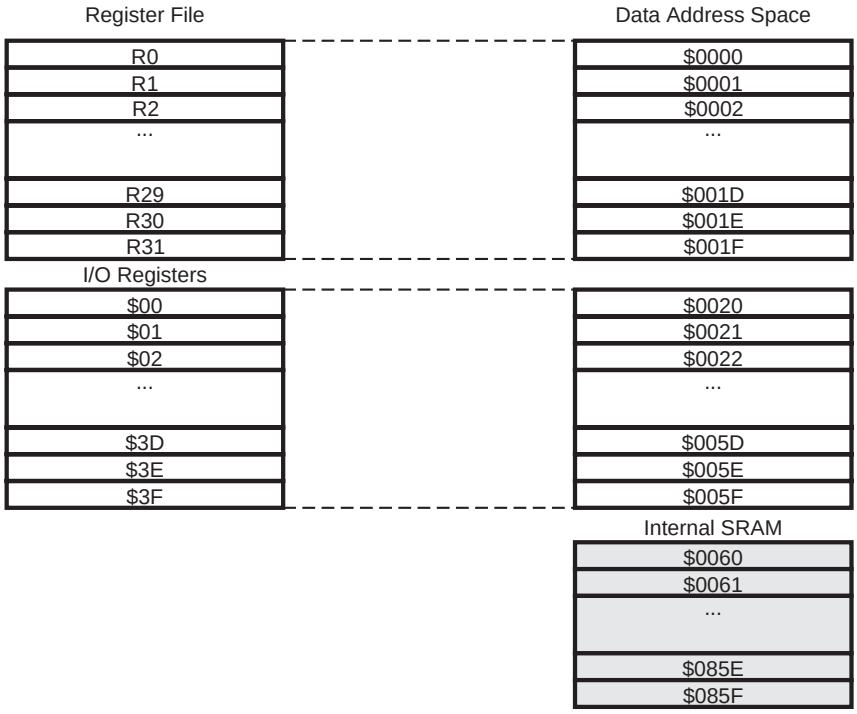
直接寻址范围可达整个数据区。

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。

在自动预减和后加的间接寻址模式中，寄存器 X、 Y 和 Z 自动增加或减少。

ATmega32的全部32个通用寄存器、64个I/O寄存器及 2048个字节的内部数据SRAM可以通过所有上述的寻址模式进行访问。寄存器文件的描述见 P8 “ 通用寄存器文件 ” 。

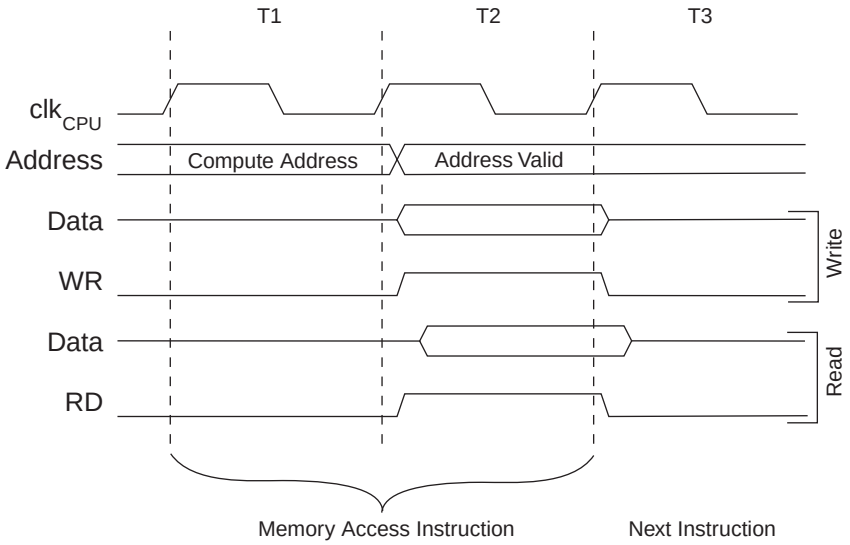
Figure 9. 数据存储器映像



数据存储器访问时间

本节说明访问内部存储器的时序。如 Figure 10 所示，内部数据 SRAM 访问时间为两个 clk_{CPU} 时钟。

Figure 10. 片上 SRAM 存取周期



EEPROM 数据存储器

ATmega32包含1024 字节的EEPROM数据存储器。它是作为一个独立的数据空间而存在的，可以按字节读写。 EEPROM 的寿命至少为 100,000 次擦除周期。 EEPROM 的访问由地址寄存器、数据寄存器和控制寄存器决定。

P240 “ 存储器编程 ” 包含使用 SPI、 JTAG 或并行编程模式对 EEPROM 编程。

EEPROM 读 / 写访问

EEPROM 的访问寄存器位于 I/O 空间。

EEPROM的写访问时间由Table 1给出。自定时功能可以让用户软件监测何时可以开始写下一字节。用户操作 EEPROM 需要注意如下问题：在电源滤波时间常数比较大的电路中，上电 / 下电时 V_{CC} 上升 / 下降速度会比较慢。此时 CPU 可能工作于低于晶振所要求的电源电压。请参见 P20 “ 防止 EEPROM 数据丢失 ” 以避免出现 EEPROM 数据丢失的问题。

为了防止无意识的 EEPROM 写操作，需要执行一个特定的写时序。具体参看 EEPROM 控制寄存器的内容。

执行 EEPROM 读操作时， CPU 会停止工作 4 个周期，然后再执行后续指令；执行 EEPROM 写操作时， CPU 会停止工作 2 个周期，然后再执行后续指令。

EEPROM 地址寄存器 - EEARH 和 EEARL

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	EEAR9	EEAR8	EEARH
	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
	7	6	5	4	3	2	1	0	
读 / 写	R	R	R	R	R	R	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	X	
	X	X	X	X	X	X	X	X	

- **Bits 15..10 – Res: 保留**

保留位，读操作返回值为零。

- **Bits 9..0 – EEAR9..0: EEPROM 地址**

EEPROM地址寄存器– EEARH和EEARL指定了1024字节的EEPROM空间。EEPROM地址是线性的，从 0 到 1023。 EEAR 的初始值没有定义。在访问 EEPROM 之前必须为其赋予正确的数据。

EEPROM 数据寄存器 - EEDR

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	EEDR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bits 7..0 – EEDR7.0: EEPROM 数据**

对于 EEPROM 写操作， EEDR 是需要写到 EEAR 单元的数据；对于读操作， EEDR 是从地址 EEAR 读取的数据。

EEPROM 控制寄存器 - EECR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	EERIE	EEMWE	EEWE	EERE	EECR
读 / 写	R	R	R	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	X	0	

- **Bits 7..4 – Res: 保留**

保留位，读操作返回值为零。

- **Bit 3 – EERIE: EEPROM 准备好中断使能**

若 SREG 的 I 为 "1"，则置位 EERIE 将使能 EEPROM 准备好中断。清零 EERIE 则禁止此中断。当 EEWE 清零时 EEPROM 准备好中断即可发生。

- **Bit 2 – EEMWE: EEPROM 主机写使能**

EEMWE决定了EEWE置位是否可以启动EEPROM写操作。当EEMWE为"1"时，在4个时钟周期内置位 EEWE 将把数据写入 EEPROM 的指定地址；若 EEMWE 为 "0“，则操作 EEWE 不起作用。 EEMWE 置位后 4 个周期，硬件对其清零。见 EEPROM 写过程中对 EEWE 位的描述。

- **Bit 1 – EEWE: EEPROM 写使能**

EEWE 为 EEPROM 写操作的使能信号。当 EEPROM 数据和地址设置好之后，需置位 EEWE 以便将数据写入 EEPROM。此时 EEMWE 必须置位，否则 EEPROM 写操作将不会发生。写时序如下 (第 3 步和第 4 步的次序并不重要) :

1. 等待 EEWE 位变为零
2. 等待 SPMCSR 中的 SPMEN 位变为零
3. 将新的 EEPROM 地址写入 EEAR(可选)
4. 将新的 EEPROM 数据写入 EEDR(可选)
5. 对 EECR 寄存器的 EEMWE 写 "1"，同时清零 EEWE
6. 在置位 EEMWE 的 4 个周期内，置位 EEWE

在 CPU 写 Flash 存储器的时候不能对 EEPROM 进行编程。在启动 EEPROM 写操作之前软件必须检查 Flash 写操作是否已经完成。步骤 (2) 仅在软件包含引导程序并允许 CPU 对 Flash 进行编程时才有用。如果 CPU 永远都不会写 Flash，步骤 (2) 可省略。请参见 P228 “支持引导装入程序 – 在写的同时可以读(RWW, Read-While-Write)的自我编程能力”。

注意：如果在步骤 5 和 6 之间发生了中断，写操作将失败。因为此时 EEPROM 写使能操作将超时。如果一个操作 EEPROM 的中断打断了另一个 EEPROM 操作，EEAR 或 EEDR 寄存器可能被修改，引起 EEPROM 操作失败。建议此时关闭全局中断标志 I。

经过写访问时间之后，EEWE 硬件清零。用户可以凭借这一位判断写时序是否已经完成。EEWE 置位后， CPU 要停止两个时钟周期才会运行下一条指令。

- **Bit 0 – EERE: EEPROM 读使能**

EERE为EEPROM读操作的使能信号。当EEPROM地址设置好之后，需置位EERE以便将数据读入 EEAR。 EEPROM 数据的读取只需要一条指令，且无需等待。读取 EEPROM 后 CPU 要停止 4 个时钟周期才可以执行下一条指令。

用户在读取 EEPROM 时应该检测 EEWE。如果一个写操作正在进行，就无法读取 EEPROM，也无法改变寄存器 EEAR。

经过校准的片内振荡器用于 EEPROM 定时。Table 1 为 CPU 访问 EEPROM 的典型时间。

Table 1. EEPROM 编程时间

符号	校准的 RC 振荡器周期数 (1)	典型的编程时间
EEPROM 写操作 (CPU)	8448	8.5 ms

Note: 1. 使用时钟频率为 1 MHz，不倚赖 CKSEL 熔丝位的设置。

下面的代码分别用汇编和 C 函数说明如何实现 EEPROM 的写操作。在此假设中断不会在执行这些函数的过程当中发生。同时还假设软件没有 Boot Loader。若 Boot Loader 存在，则 EEPROM 写函数还需要等待正在运行的 SPM 命令的结束。

汇编代码例程

```
EEPROM_write:
  ; 等待上一次写操作结束
  sbic EECR,EEWE
  rjmp EEPROM_write
  ; 设置地址寄存器 (r18:r17)
  out  EEARH, r18
  out  EEARL, r17
  ; 将数据写入数据寄存器(r16)
  out  EEDR,r16
  ; 置位 EEMWE
  sbi  EECR,EEMWE
  ; 置位 EEWE 以启动写操作
  sbi  EECR,EEWE
  ret
```

C 代码例程

```
void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{
  /* 等待上一次写操作结束 */
  while(EECR & (1<<EEWE))
    ;
  /* 设置地址和数据寄存器*/
  EEAR = uiAddress;
  EEDR = ucData;
  /* 置位 EEMWE */
  EECR |= (1<<EEMWE);
  /* 置位 EEWE 以启动写操作 */
  EECR |= (1<<EEWE);
}
```

下面的例子说明如何用汇编和 C 函数来读取 EEPROM，在此假设中断不会在执行这些函数的过程当中发生。

汇编代码例程

```
EEPROM_read:
  ; 等待上一次写操作结束
  sbic EECR,EEWE
  rjmp EEPROM_read
  ; 设置地址寄存器 (r18:r17)
  out  EEARH, r18
  out  EEARL, r17
  ; 设置EERE 以启动读操作
  sbi  EECR,EERE
  ; 自数据寄存器读取数据
  in   r16,EEDR
  ret
```

C 代码例程

```
unsigned char EEPROM_read(unsigned int uiAddress)
{
  /* 等待上一次写操作结束 */
  while(EECR & (1<<EEWE))
    ;
  /* 设置地址寄存器*/
  EEAR = uiAddress;
  /* 设置EERE 以启动读操作 */
  EECR |= (1<<EERE);
  /* 自数据寄存器返回数据 */
  return EEDR;
}
```

在掉电休眠模式下的 EEPROM 写操作

若程序执行掉电指令时 EEPROM 的写操作正在进行，EEPROM 的写操作将继续，并在指定的写访问时间之前完成。但写操作结束后，振荡器还将继续运行，芯片并非处于完全的掉电模式。因此在执行掉电指令之前应结束 EEPROM 的写操作。

防止 EEPROM 数据丢失

若电源电压过低，CPU 和 EEPROM 有可能工作不正常，造成 EEPROM 数据的毁坏 (丢失)。这种情况在使用独立的 EEPROM 器件时也会遇到。因而需要使用相同的保护方案。

由于电压过低造成 EEPROM 数据损坏有两种可能：一是电压低于 EEPROM 写操作所需要的最低电压；二是 CPU 本身已经无法正常工作。

EEPROM 数据损坏的问题可以通过以下方法解决：

当电压过低时保持AVR RESET信号为低。这可以通过使能芯片的掉电检测电路BOD来实现。如果 BOD 电平无法满足要求则可以使用外部复位电路。若写操作过程当中发生了复位，只要电压足够高，写操作仍将正常结束。

I/O 存储器

ATmega32 的 I/O 空间定义见 P283 “ 寄存器概述 ”。

ATmega32所有的I/O及外设都被放置于I/O空间。所有的I/O位置都可以通过IN 与OUT指令来访问，在 32 个通用工作寄存器和 I/O 之间传输数据。地址为 0x00 - 0x1F 的 I/O 寄存器还可用 SBI 和 CBI 指令直接进行位寻址，而 SBIS 和 SBIC 则用来检查某一位的值。更多内容请参见指令集。使用 IN 和 OUT 指令时地址必须在 0x00 - 0x3F 之间。如果要象 SRAM 一样通过 LD 和 ST 指令访问 I/O 寄存器，相应的地址要加上 0x20。

为了与后续产品兼容，保留未用的未应写 "0"，而保留的 I/O 寄存器则不应进行写操作。

一些状态标志位的清除是通过写 "1" 来实现的。要注意的是，与其他大多数 AVR 不同，CBI 和 SBI 指令只能对某些特定的位进行操作，因而可以用于包含这些状态标志的寄存器。 CBI 与 SBI 指令只对 0x00 到 0x1F 的寄存器有效。

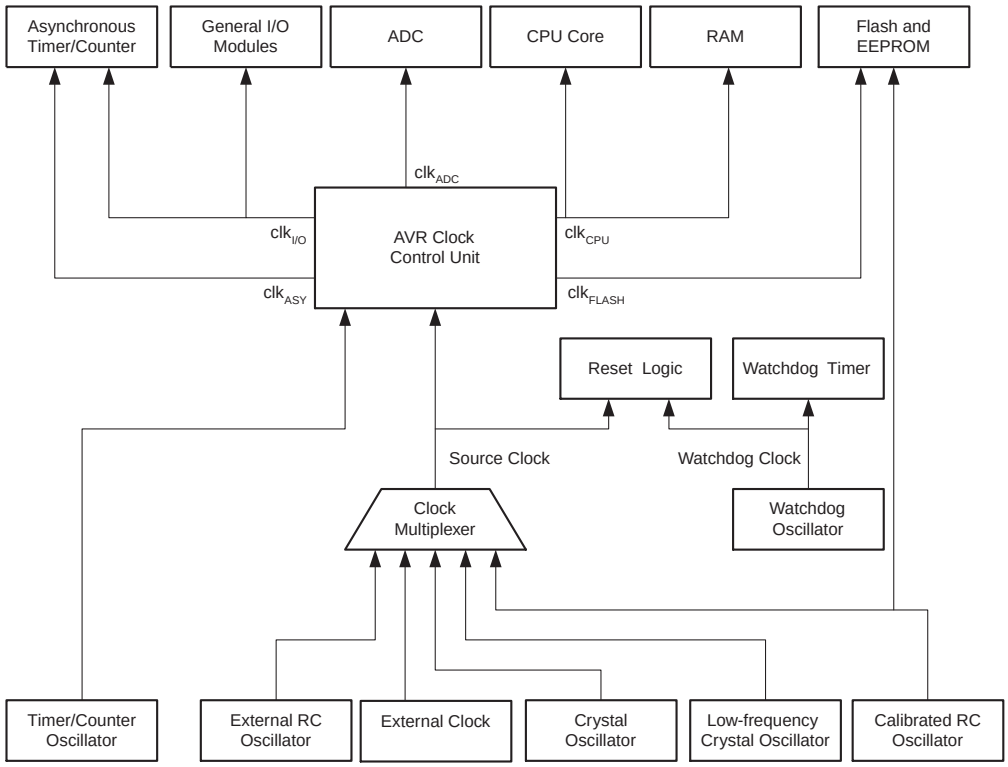
I/O 和外设控制寄存器在后续其他章节进行介绍。

系统时钟及时钟选项

时钟系统及其分布

Figure 11为AVR的主要时钟系统及其分布。这些时钟并不需要同时工作。为了降低功耗，可以通过使用不同的睡眠模式来禁止无需工作的模块的时钟，详见 P30 “ 电源管理及睡眠模式 ”。时钟系统详见 Figure 11。

Figure 11. 时钟分布



CPU 时钟 - clk_{CPU}

CPU时钟与操作AVR内核的子系统相连，如通用寄存器文件、状态寄存器及保存堆栈指针的数据存储器。终止 CPU 时钟将使内核停止工作和计算。

I/O 时钟 - $clk_{I/O}$

I/O 时钟用于主要的 I/O 模块，如定时器 / 计数器、SPI 和 USART。I/O 时钟还用于外部中断模块。要注意的是有些外部中断由异步逻辑检测，因此即使 I/O 时钟停止了这些中断仍然可以得到监控。此外， USI 模块的起始条件检测在没有 $clk_{I/O}$ 的情况下也是异步实现的，使得这个功能在任何睡眠模式下都可以正常工作。

Flash 时钟 - clk_{FLASH}

Flash 时钟控制 Flash 接口的操作。此时钟通常与 CPU 时钟同时挂起或激活。

异步定时器时钟 - clk_{ASY}

异步定时器时钟允许异步定时器 / 计数器直接由外部 32 kHz 时钟晶体驱动。使得此定时器 / 计数器即使在睡眠模式下仍然可以为系统提供一个实时时钟。

ADC 时钟 - clk_{ADC}

ADC具有专门的时钟。这样可以在ADC工作的时候停止CPU和I/O时钟以降低数字电路产生的噪声，从而提高 ADC 转换精度。

时钟源

ATmega32芯片有如下几种通过Flash熔丝位进行选择的时钟源。时钟输入到AVR时钟发生器，再分配到相应的模块。

Table 2. 时钟源选择 (1)

芯片时钟选项	CKSEL3..0
外部晶体 / 陶瓷振荡器	1111 - 1010
外部低频晶振	1001
外部 RC 振荡器	1000 - 0101
标定的内部 RC 振荡器	0100 - 0001
外部时钟	0000

Note: 1. 对于所有的熔丝位，“1” 表示未编程，“0” 代表已编程。

不同的时钟选项将在后续部分进行介绍。当 CPU 自掉电模式或省电模式唤醒之后，被选择的时钟源用来为启动过程定时，保证振荡器在开始执行指令之前进入稳定状态。当CPU从复位开始工作时，还有额外的延迟时间以保证在 MCU 开始正常工作之前电源达到稳定电平。这个启动时间的定时由看门狗振荡器完成。看门狗溢出时间所对应的 WDT 振荡器周期数列于 Table 3。看门狗振荡器的频率由工作电压决定，详见 P283 “ 寄存器概述 ” 。

Table 3. 看门狗振荡器周期数

典型的溢出时间 (V_{CC} = 5.0V)	典型的溢出时间 (V_{CC} = 3.0V)	时钟周期数
4.1 ms	4.3 ms	4K (4,096)
65 ms	69 ms	64K (65,536)

缺省时钟源

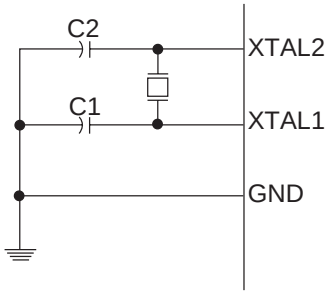
芯片出厂时 CKSEL = “0001”， SUT = “10”。这个缺省设置的时钟源是内部 RC 振荡器，启动时间为最长。这种设置保证用户可以通过 ISP 或并行编程器得到所需的时钟源。

晶体振荡器

XTAL1 与 XTAL2 分别为用作片内振荡器的反向放大器的输入和输出，如 Figure 12 所示，这个振荡器可以使用石英晶体，也可以使用陶瓷谐振器。熔丝位 CKOPT 用来选择这两种放大器模式的其中之一。当 CKOPT 被编程时振荡器在输出引脚产生满幅度的振荡。这种模式适合于噪声环境，以及需要通过 XTAL2 驱动第二个时钟缓冲器的情况。而且这种模式的频率范围比较宽。当保持 CKOPT 为未编程状态时，振荡器的输出信号幅度比较小。其优点是大大降低了功耗，但是频率范围比较窄，而且不能驱动其他时钟缓冲器。

对于谐振器， CKOPT 未编程时的最大频率为 8 MHz， CKOPT 编程时为 16 MHz。 C1 和 C2 的数值要一样，不管使用的是晶体还是谐振器。最佳的数值与使用的晶体或谐振器有关，还与杂散电容和环境的电磁噪声有关。 Table 4 给出了针对晶体选择电容的一些指南。对于陶瓷谐振器，应该使用厂商提供的数值。若想得到更多的有关如何选择电容以及振荡器如何工作的信息，请参考多用途振荡器应用手册。

Figure 12. 晶体振荡器连接图



振荡器可以工作于三种不同的模式，每一种都有一个优化的频率范围。工作模式通过熔丝位 CKSEL3..1 来选择，如 Table 4 所示。

Table 4. 晶体振荡器工作模式

CKOPT	CKSEL3..1	频率范围 (MHz)	使用晶体时电容 C1 和 C2 的推荐范围 (pF)
1	101(1)	0.4 - 0.9	–
1	110	0.9 - 3.0	12 - 22
1	111	3.0 - 8.0	12 - 22
0	101, 110, 111	1.0 ≤	12 - 22

Note: 1. 此选项不适用于晶体，只能用于陶瓷谐振器。

如 Table 5 所示，熔丝位 CKSEL0 以及 SUT1..0 用于选择启动时间。

Table 5. 晶体振荡器时钟选项对应的启动时间

CKSEL0	SUT1..0	掉电与节电模式下的启动时间	复位时额外的延迟时间 (V_{CC} = 5.0V)	推荐用法
0	00	258 CK(1)	4.1 ms	陶瓷谐振器，电源快速上升
0	01	258 CK(1)	65 ms	陶瓷谐振器，电源缓慢上升
0	10	1K CK(2)	–	陶瓷谐振器， BOD 使能
0	11	1K CK(2)	4.1 ms	陶瓷谐振器，电源快速上升
1	00	1K CK(2)	65 ms	陶瓷谐振器，电源缓慢上升
1	01	16K CK	–	石英振荡器，BOD 使能
1	10	16K CK	4.1 ms	石英振荡器，电源快速上升
1	11	16K CK	65 ms	石英振荡器，电源慢速上升

Notes: 1. 这些选项只能用于工作频率不太接近于最大频率，而且启动时的频率稳定性对于应用而言不重要的情况。不适用于晶体。
2. 这些选项是为陶瓷谐振器设计的，可以保证启动时频率足够稳定。若工作频率不太接近于最大频率，而且启动时的频率稳定性对于应用而言不重要时也适用于晶体。

低频晶体振荡器

为了使用 32.768 kHz 钟表晶体作为器件的时钟源，必须将熔丝位 CKSEL 设置为 “1001” 以选择低频晶体振荡器。晶体的连接方式如 Figure 12 所示。通过对熔丝位 CKOPT 的编程，用户可以使能 XTAL1 和 XTAL2 的内部电容，从而去除外部电容。内部电容的标称数值为 36 pF。

选择了这个振荡器之后，启动时间由熔丝位 SUT 确定，如 Table 6 所示。

Table 6. 低频晶体振荡器的启动时间

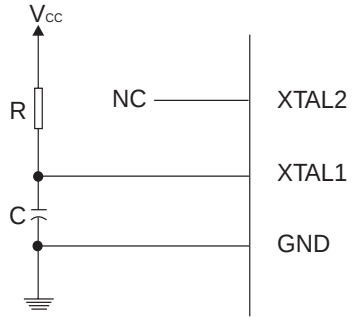
SUT1..0	掉电模式和省电模式的启动时间	复位时的额外延迟时间 (V_{CC} = 5.0V)	推荐用法
00	1K CK(1)	4.1 ms	电源快速上升，或是 BOD 使能
01	1K CK(1)	65 ms	电源缓慢上升
10	32K CK	65 ms	启动时频率已经稳定
11	保留		

Note: 1. 这些选项只能用于启动时的频率稳定性对应用而言不重要的情况。

外部 RC 振荡器

对于时间不敏感的应用可以使用 Figure 13 的外部 RC 振荡器。频率可以通过方程 f = 1/(3RC) 进行粗略地鼓估计。电容 C 至少要 22 pF。通过编程熔丝位 CKOPT，用户可以使能 XTAL1 和 GND 之间的片内 36 pF 电容，从而无需外部点燃。若想获取有关振荡器如何工作以及如何选择 R 和 C 的具体信息，请参考外部 RC 振荡器应用手册。

Figure 13. 外部 RC 配置



振荡器可以工作于四个不同的模式，每个模式有自己的优化频率范围。工作模式通过熔丝位 CKSEL3..0 选取，如 Table 7 所示。

Table 7. 外部 RC 振荡器工作模式

CKSEL3..0	频率范围 (MHz)
0101	≤ 0.9
0110	0.9 - 3.0
0111	3.0 - 8.0
1000	8.0 - 12.0

选择了这个振荡器之后，启动时间由熔丝位 SUT 确定，如 Table 8 所示。

Table 8. 外部 RC 振荡器的启动时间

SUT1..0	掉电模式和省电模式的启动时间	复位时的额外延迟时间 (V_{CC} = 5.0V)	推荐用法
00	18 CK	–	BOD 使能
01	18 CK	4.1 ms	电源快速上升
10	18 CK	65 ms	电源缓慢上升
11	6 CK(1)	4.1 ms	电源快速上升，或是 BOD 使能

Note: 1. 这些选项只能用于工作频率不太接近于最大频率时的情况。

标定的片内 RC 振荡器

标定的片内 RC 振荡器提供了固定的 1.0、 2.0、 4.0 或 8.0 MHz 的时钟。这些频率都是 5V、 25°C 下的标称数值。这个时钟也可以作为系统时钟，只要按照 Table 9 对熔丝位 CKSEL进行编程即可。选择这个时钟(此时不能对CKOPT进行编程)之后就无需外部器件了。复位时硬件将标定字节加载到 OSCCAL 寄存器，自动完成对 RC 振荡器的标定。在 5V， 25°C 和频率为 1.0 MHz 时，这种标定可以提供标称频率 ± 3% 的精度；使用 www.atmel.com/avr 中所给出的方法，可在任何电压、任何温度下，使精度达到 ± 1% 。当使用这个振荡器作为系统时钟时，看门狗仍然使用自己的看门狗定时器作为溢出复位的依据。更多的有关标定数据的信息请参见 P242 “ 标定字节 ” 。

Table 9. 片内标定的 RC 振荡器工作模式

CKSEL3..0	标称频率 (MHz)
0001(1)	1.0
0010	2.0
0011	4.0
0100	8.0

Note: 1. 出厂时的设置。

选择了这个振荡器之后，启动时间由熔丝位 SUT 确定，如 Table 10 所示。 XTAL1 和 XTAL2 要保持为空 (NC)。

Table 10. 内部标定 RC 振荡器的启动时间

SUT1..0	掉电模式与省电模式的启动时间	复位时的额外延迟时间 (V_{CC} = 5.0V)	推荐用法
00	6 CK	–	BOD 使能
01	6 CK	4.1 ms	电源快速上升
10(1)	6 CK	65 ms	电源缓慢上升
11	保留		

Note: 1. 出厂时的设置。

振荡器标定寄存器 - OSCCAL

Bit	7	6	5	4	3	2	1	0	
	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	
初始值	标定数据								

- **Bits 7..0 – CAL7..0: 振荡器标定数据**

将标定数据写入这个地址可以对内部振荡器进行调节以消除由于生产工艺所带来的振荡器频率偏差。复位时 1 MHz 的标定数据 (标识数据的高字节，地址为 0x00) 自动加载到 OSCCAL 寄存器。如果需要内部 RC 振荡器工作于其他频率，标定数据必须人工加载：首先通过编程器读取标识数据，然后将标定数据保存到 Flash 或 EEPROM 之中。这些数据可以通过软件读取，然后加载到 OSCCAL 寄存器。当 OSCCAL 为零时振荡器以最低频率工作。当对其写如不为零的数据时内部振荡器的频率将增长。写入 0xFF 即得到最高频率。标定的振荡器用来为访问 EEPROM 和 Flash 定时。有写 EEPROM 和 Flash 的操作时不要将频率标定到超过标称频率的 10%，否则写操作有可能失败。要注意振荡器只对 1.0、2.0、4.0和8.0 MHz这四种频率进行了标定，其他频率则无法保证，如Table 11所示。

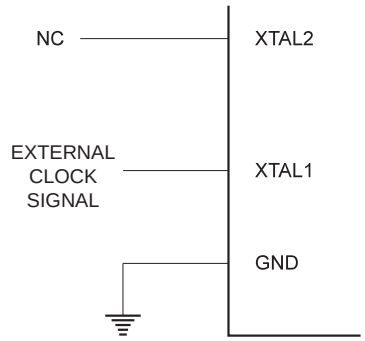
Table 11. 内部 RC 振荡器频率范围 .

OSCCAL 数值	最小频率，标称频率的百分比 (%)	最大频率，标称频率的百分比 (%)
$00	50	100
$7F	75	150
$FF	100	200

外部时钟

为了从外部时钟源驱动芯片， XTAL1 必须如 Figure 14 所示的进行连接。同时，熔丝位 CKSEL必须编程为“0000”。若熔丝位CKOPT也被编程，用户就可以使用内部的XTAL1和 GND 之间的 36 pF 电容。

Figure 14. 外部时钟驱动配置图



选择了这个振荡器之后，启动时间由熔丝位 SUT 确定，如 Table 12 所示。

Table 12. 外部时钟的启动时间

SUT1..0	掉电模式与省电模式的启动时间	复位时的额外延迟时间 (V_{CC} = 5.0V)	推荐用法
00	6 CK	–	BOD 使能
01	6 CK	4.1 ms	电源快速上升
10	6 CK	65 ms	电源缓慢上升
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。工作频率突变超过 2% 将会产生异常现象。应该在 MCU 保持复位状态时改变外部时钟的振荡频率。

定时器 / 计时器振荡器

对于拥有定时器 / 振荡器引脚 (TOSC1 和 TOSC2) 的 AVR 微处理器，晶体可以直接与这两个引脚连接，无需外部电容。此振荡器针对 32.768 kHz 的钟表晶体作了优化。不建议在 TOSC1 引脚输入振荡信号。

电源管理及睡眠模式

睡眠模式可以使应用程序关闭 MCU 中没有使用的模块，从而降低功耗。 AVR 具有不同的睡眠模式，允许用户根据自己的应用要求实施剪裁。

进入睡眠模式的条件是置位寄存器 MCUCR 的 SE，然后执行 SLEEP 指令。具体哪一种模式 (空闲模式、ADC 噪声抑制模式、掉电模式、省电模式、Standby 模式和扩展 Standby 模式) 由 MCUCR 的 SM2、 SM1 和 SM0 决定，如 Table 13 所示。使能的中断可以将进入睡眠模式的 MCU 唤醒。经过启动时间，外加 4 个时钟周期后， MCU 就可以运行中断例程了。然后返回到 SLEEP 的下一条指令。唤醒时不会改变寄存器文件和 SRAM 的内容。如果在睡眠过程中发生了复位，则 MCU 唤醒后从中断向量开始执行。

P22 Figure 11 介绍了 ATmega32 不同的时钟系统及其分布。此图在选择合适的睡眠模式时非常有用。

MCU 控制寄存器 - MCUCR

控制寄存器包含了电源管理的控制位。

Bit	7	6	5	4	3	2	1	0	
	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	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – SE: 休眠使能**

为了使 MCU 在执行 SLEEP 指令后进入休眠模式， SE 必须置位。为了确保进入休眠模式是程序员的有意行为，建议仅在 SLEEP 指令的前一条指令置位 SE。MCU 一旦唤醒立即清除 SE。

- **Bits 6...4 – SM2..0: 休眠模式选择位 2、 1 和 0**

如 Table 13 所示，这些位用于选择具体的休眠模式。

Table 13. 休眠模式选择

SM2	SM1	SM0	休眠模式
0	0	0	空闲模式
0	0	1	ADC 噪声抑制模式
0	1	0	掉电模式
0	1	1	省电模式
1	0	0	保留
1	0	1	保留
1	1	0	Standby(1) 模式
1	1	1	扩展 Standby(1) 模式

Note: 1. 仅在使用外部晶体或谐振器时 Standby 模式与扩展 Standby 模式才可用。

空闲模式

当 SM2..0 为 000 时，SLEEP 指令将使 MCU 进入空闲模式。在此模式下，CPU 停止运行，而 SPI、 USART、模拟比较器、 ADC、两线串行接口、定时器 / 计数器、看门狗和中断系统继续工作。这个睡眠模式只停止了 clk_{CPU} 和 clk_{FLASH}，其他时钟则继续工作。

象定时器溢出与 USART 传输完成等内外部中断都可以唤醒 MCU。如果不需要从模拟比较器中断唤醒 MCU，为了减少功耗，可以切断比较器的电源。方法是置位模拟比较器控制和状态寄存器 ACSR 的 ACD。如果 ADC 使能，进入此模式后将自动启动一次转换。

ADC 噪声抑制模式

当 SM2..0 为 001 时，SLEEP 指令将使 MCU 进入噪声抑制模式。在此模式下，CPU 停止运行，而 ADC、外部中断、两线接口地址配置、定时器 / 计数器 2 和看门狗继续工作。这个睡眠模式只停止了 $clk_{I/O}$、 clk_{CPU} 和 clk_{FLASH}，其他时钟则继续工作。

此模式提高了 ADC 的噪声环境，使得转换精度更高。 ADC 使能的时候，进入此模式将自动启动一次 AD 转换。ADC 转换结束中断、外部复位、看门狗复位、 BOD 复位、两线接口地址匹配中断、定时器 / 计数器 2 中断、SPM/EEPROM 准备好中断、外部电平中断 INT0 或 INT1，或外部中断 INT2 可以将 MCU 从 ADC 噪声抑制模式唤醒。

掉电模式

当 SM2..0 为 010 时， SLEEP 指令将使 MCU 进入掉电模式。在此模式下，外部晶体停振，而外部中断、两线接口地址匹配及看门狗 （如果使能的话）继续工作。只有外部复位、看门狗复位、 BOD 复位、两线接口地址匹配中断、外部电平中断 INT0 或 INT1，或外部中断 INT2 可以使 MCU 脱离掉电模式。这个睡眠模式停止了所有的时钟，只有异步模块可以继续工作。

当使用外部电平中断方式将 MCU 从掉电模式唤醒时，必须保持外部电平一定的时间。具体请参见 P64 “ 外部中断 ”。

从施加掉电唤醒条件到真正唤醒有一个延迟时间，此时间用于时钟重新启动并稳定下来。唤醒周期与由熔丝位 CKSEL 定义的复位周期是一样的，如 P23 “ 时钟源 ”。

省电模式

当 SM2..0 为 011 时， SLEEP 指令将使 MCU 进入省电模式。这一模式与掉电模式只有一点不同：

如果定时器 / 计数器 2 为异步驱动，即寄存器 ASSR 的 AS2 置位，则定时器 / 计数器 2 在睡眠时继续运行。除了掉电模式的唤醒方式，定时器 / 计数器 2 的溢出中断和比较匹配中断也可以将 MCU 从休眠方式唤醒，只要 TIMSK 使能了这些中断，而且 SREG 的全局中断使能位 I 置位。

如果异步定时器不是异步驱动的，建议使用掉电模式，而不是省电模式。因为在省电模式下，若 AS2 为 0，则 MCU 唤醒后异步定时器的寄存器数值是没有定义的。

这个睡眠模式停止了除 clk_{ASY} 以外所有的时钟，只有异步模块可以继续工作。

Standby 模式

当 SM2..0 为 110 时， SLEEP 指令将使 MCU 进入 Standby 模式。这一模式与掉电模式唯一的不同之处在于振荡器继续工作。其唤醒时间只需要 6 个时钟周期。

扩展 Standby 模式

当 SM2..0 为 111 时， SLEEP 指令将使 MCU 进入扩展的 Standby 模式。这一模式与省掉电模式唯一的不同之处在于振荡器继续工作。其唤醒时间只需要 6 个时钟周期。

Table 14. 在不同睡眠模式下活动的时钟以及唤醒源

	工作的时钟					振荡器		唤醒源					
睡眠模式	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	使能的主时钟	使能的定时器时钟	**INT2 INT1 INT0**	**TWI 地址匹配**	**定时器 2**	**SPM/ EEP ROM 准备好**	**A D C**	**其他 I/O**
空闲模式			X	X	X	X	X(2)	X	X	X	X	X	X
ADC 噪声抑制模式				X	X	X	X(2)	X(3)	X	X	X	X	
掉电模式								X(3)	X				
省电模式					X(2)		X(2)	X(3)	X	X(2)			
Standby 模式 (1)						X		X(3)	X				
扩展的 Standby 模式 (1)					X(2)	X	X(2)	X(3)	X	X(2)			

Notes: 1. 时钟源为外部晶体或谐振器
2. ASSR 的 AS2 置位
3. INT2 或电平中断 INT1 与 INT0

最小化功耗

试图降低 AVR 控制系统的功耗时需要考虑几个问题。一般来说，要尽可能利用睡眠模式，并且使尽可能少的模块继续工作。不需要的功能必须禁止。下面的模块需要特殊考虑以达到尽可能低的功耗。

模数转换器

使能时， ADC 在睡眠模式下继续工作。为了降低功耗，在进入睡眠模式之前需要禁止 ADC。重新启动后的第一次转换为扩展的转换。具体请参照 P187 “ 模数转换器 ” 。

模拟比较器

在空闲模式时，如果没有使用模拟比较器，可以将其关闭。在 ADC 噪声抑制模式下也是如此。在其他睡眠模式模拟比较器是自动关闭的。如果模拟比较器使用了内部电压基准源，则不论在什么睡眠模式下都需要关闭它。否则内部电压基准源将一直使能。请参见 P184 “ 模拟比较器 ” 以了解如何配置模拟比较器。

掉电检测 BOD

如果系统没有使用掉电检测器 BOD ,这个模块也可以关闭。如果熔丝位 BODEN 被编程，从而使能了 BOD 功能，它将在各种休眠模式下继续工作。在深层次的休眠模式下，这个电流将占总电流的很大比重。请参看 P37 “ 掉电检测 ” 以了解如何配置 BOD。

片内基准电压

使用 BOD、模拟比较器和 ADC 时可能需要内部电压基准源。若这些模块都禁止了，则基准源也可以禁止。重新使能后用户必须等待基准源稳定之后才可以使用它。如果基准源在休眠过程中是使能的，其输出立即可以使用。请参见 P39 “ 片内基准电压 ” 以了解基准源启动时间的细节。

看门狗定时器

如果系统无需使用看门狗，这个模块也可以关闭。若使能，则在任何休眠模式下都持续工作，从而消耗电流。在深层次的睡眠模式下，这个电流将占总电流的很大比重。请参看 P39 “ 看门狗定时器 ” 以了解如何配置看门狗定时器。

端口引脚

进入休眠模式时，所有的端口引脚都应该配置为只消耗最小的功耗。最重要的是避免驱动电阻性负载。在休眠模式下 I/O 时钟 $clk_{I/O}$ 和 ADC 时钟 clk_{ADC} 都被停止了，输入缓冲器也禁止了，从而保证输入电路不会消耗电流。在某些情况下输入逻辑是使能的，用来检测唤醒条件。用于此功能的具体引脚请参见 P51 “ 数字输入使能和睡眠模式 ”。如果输入缓冲器是使能的，此时输入不能悬空，信号电平也不应该接近 $V_{CC}/2$，否则输入缓冲器会消耗额外的电流。

JTAG 接口与片上调试系统

如果通过熔丝位 OCDEN 使能了片上调试系统，当芯片进入掉电或省电模式时主时钟保持运行。在休眠模式中这个电流占总电流的很大比重。下面有三种替代方法：

- 不编程 OCDEN
- 不编程 JTAGEN
- 置位 MCUCSR 的 JTD

当 JTAG 接口使能而 JTAG TAP 控制器没有进行数据交换时，引脚 TDO 将悬空。如果与 TDO 引脚连接的硬件电路没有上拉电阻，功耗将增加。器件的引脚 TDI 包含一个上拉电阻，因此在扫描链中无需为下一个芯片的 TDO 引脚设置上拉电阻。通过置位 MCUCSR 寄存器的 JTD 或不对 JTAG 熔丝位编程可以禁止 JTAG 接口。

系统控制与复位

复位 AVR

复位时所有的 I/O 寄存器都被设置为初始值，程序从复位向量处开始执行。复位向量处的指令必须是绝对跳转 JMP 指令，以使程序跳转到复位处理例程。如果程序永远不利用中断功能，中断向量可以由一般的程序代码所覆盖。这个处理方法同样适用于当复位向量位于应用程序区，中断向量位于 Boot 区 — 或者反过来 — 的时候。 Figure 15 为复位逻辑的电路图。 Table 15 则定义了复位电路的电气参数。

复位源有效时 I/O 端口立即复位为初始值。此时不要求任何时钟处于正常运行状态。

所有的复位信号消失之后，芯片内部的一个延迟计数器被激活，将内部复位的时间延长。这种处理方式使得在 MCU 正常工作之前有一定的时间让电源达到稳定的电平。延迟计数器的溢出时间通过熔丝位 SUT 与 CKSEL 设定。延迟时间的选择请参见 P23 “ 时钟源 ” 。

复位源

ATmega32 有 5 个复位源：

- 上电复位。电源电压低于上电复位门限 V_{POT} 时， MCU 复位。
- 外部复位。引脚 $\overline{\text{RESET}}$ 上的低电平持续时间大于最小脉冲宽度时 MCU 复位。
- 看门狗复位。看门狗使能并且看门狗定时器溢出时复位发生。
- 掉电检测复位。掉电检测复位功能使能，且电源电压低于掉电检测复位门限 V_{BOT} 时 MCU 即复位。
- JTAG AVR 复位。复位寄存器为 1 时 MCU 复位。详见 P210 “IEEE 1149.1 (JTAG) 边界扫描 ” 。

Figure 15. 复位逻辑

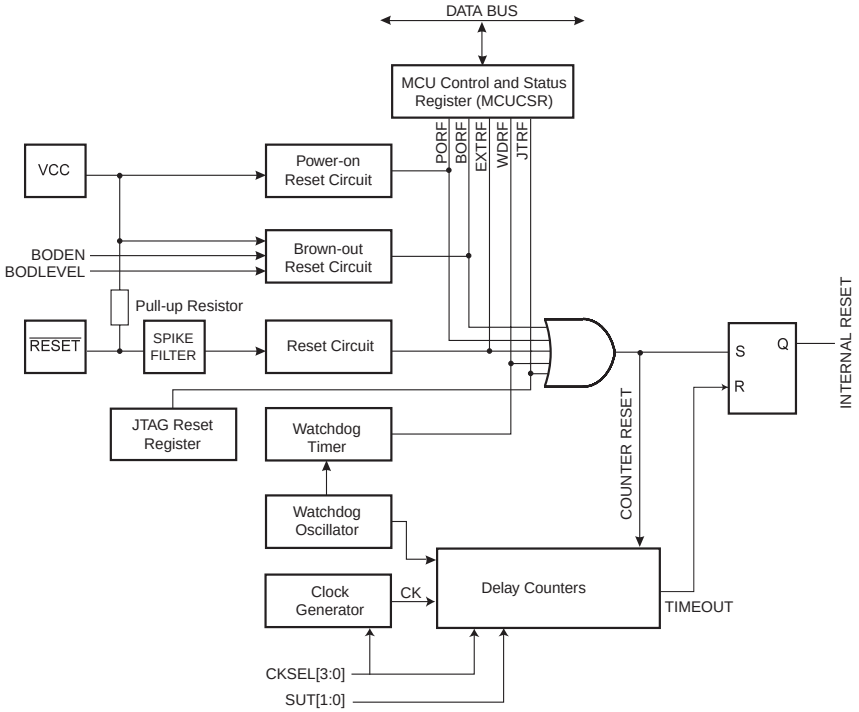


Table 15. 复位特性

符号	参数	条件	最小值	典型值	最大值	单位
V_{POT}	上电复位门限电压 (电压由低到高上升)			1.4	2.3	V
	上电复位门限电压 (电压由高到低跌落)(1)			1.3	2.3	V
V_{RST}	$\overline{\text{RESET}}$ 门限电压		0.1 V_{CC}		0.9V_{CC}	V
t_{RST}	$\overline{\text{RESET}}$ 最小脉冲宽度				1.5	µs
V_{BOT}	掉电检测复位门限电压 (2)	BODLEVEL = 1	2.5	2.7	3.2	V
		BODLEVEL = 0	3.7	4.0	4.2	
t_{BOD}	触发掉电检测复位的低电平的最小持续时间	BODLEVEL = 1		2		µs
		BODLEVEL = 0		2		µs
V_{HYST}	掉电检测器的容限			50		mV

Notes: 1. 电压下降时，只有电压低于 V_{POT} 时复位才会发生。
2. 一些器件的 V_{BOT} 可能比标称的最小工作电压还要低。这些器件在生产测试过程中进行了 $V_{CC} = V_{BOT}$ 的测试，保证在 V_{CC} 下降到处理器无法正常工作之前产生掉电检测复位。ATmega32L 的测试条件为 BODLEVEL=1，ATmega32 的测试条件为 BODLEVEL=0。BODLEVEL=1 不适用于 ATmega32。

上电复位

上电复位 (POR) 脉冲由片内检测电路产生。检测电平请参见 Table 15。无论何时 V_{CC} 低于检测电平 POR 即发生。 POR 电路可以用来触发启动复位，或者用来检测电源故障。

POR 电路保证器件在上电时复位。V_{CC} 达到上电门限电压后触发延迟计数器。在计数器溢出之前器件一直保持为复位状态。当 V_{CC} 下降时，只要低于检测门限，RESET 信号立即生效。

Figure 16. MCU 启动过程，$\overline{\text{RESET}}$ 连接到 V_{CC}.

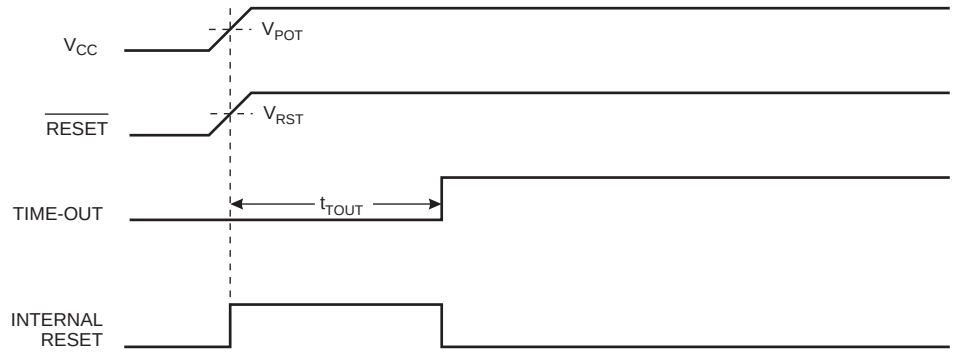
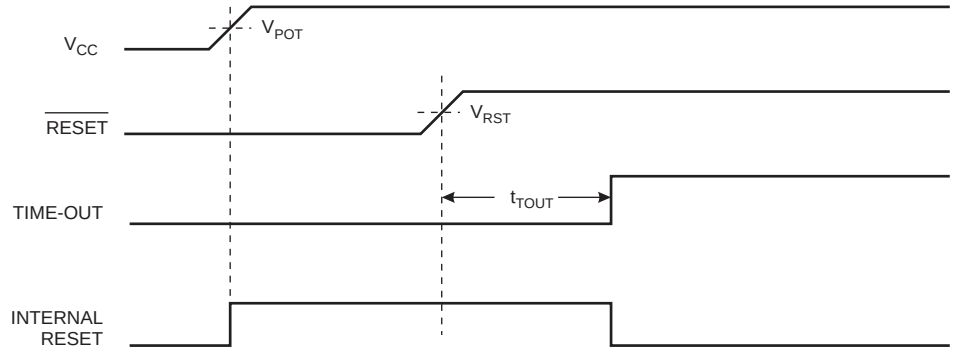


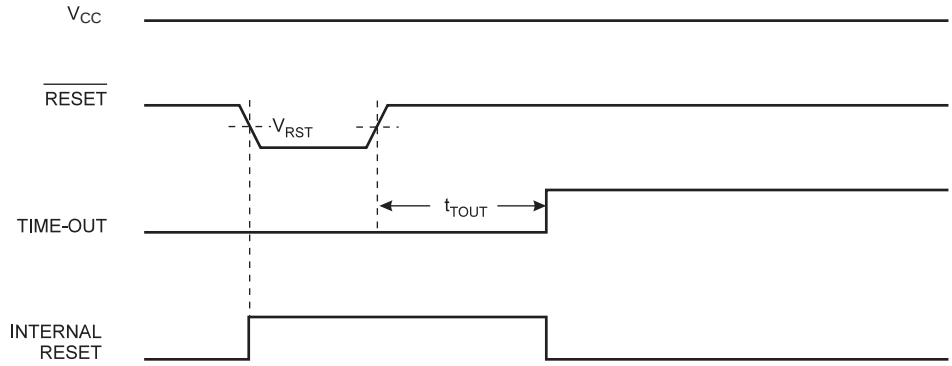
Figure 17. MCU 启动过程，$\overline{\text{RESET}}$ 由外电路控制



外部复位

外部复位由外加于 $\overline{\text{RESET}}$ 引脚的低电平产生。当复位低电平持续时间大于最小脉冲宽度时 (参见 Table 15) 即触发复位过程，即使此时并没有时钟信号在运行。当外加信号达到复位门限电压 V_{RST}(上升沿) 时，t_{TOUT} 延时周期开始。延时结束后 MCU 即启动。

Figure 18. 工作过程中发生外部复位



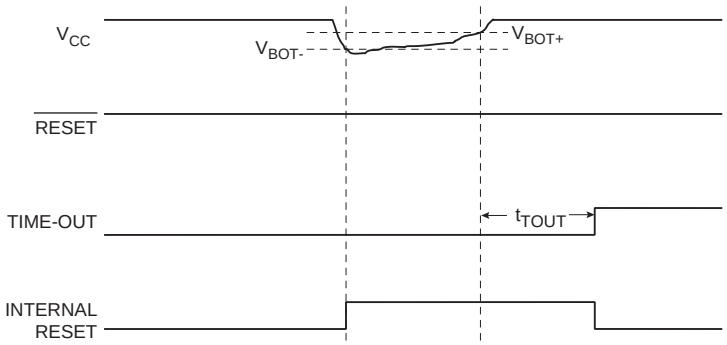
掉电检测

ATmega32 具有片内 BOD(Brown-out Detection) 电路，通过与固定的触发电平的对比来检测工作过程中 V_{CC} 的变化。此触发电平通过熔丝位 BODLEVEL 来设定，2.7V (BODLEVEL 未编程)，4.0V (BODLEVEL 已编程)。BOD 的触发电平具有迟滞功能以消除电源尖峰的影响。这个迟滞功能可以解释为 $V_{BOT+} = V_{BOT} + V_{HYST}/2$ 以及 $V_{BOT-} = V_{BOT} - V_{HYST}/2$。

BOD 电路的开关由熔丝位BODEN控制。当BOD使能后(BODEN被编程)，一旦V_{CC}下降到触发电平以下 (V_{BOT-}， Figure 19)， BOD 复位立即被激发。当 V_{CC} 上升到触发电平以上时(V_{BOT+} ,Figure 19) ,延时计数器开始计数，一旦超过溢出时间t_{TOUT} ,MCU即恢复工作。

如果 V_{CC} 一直低于触发电平并保持如 Table 15 所示的时间 t_{BOD}， BOD 电路将只检测电压跌落。

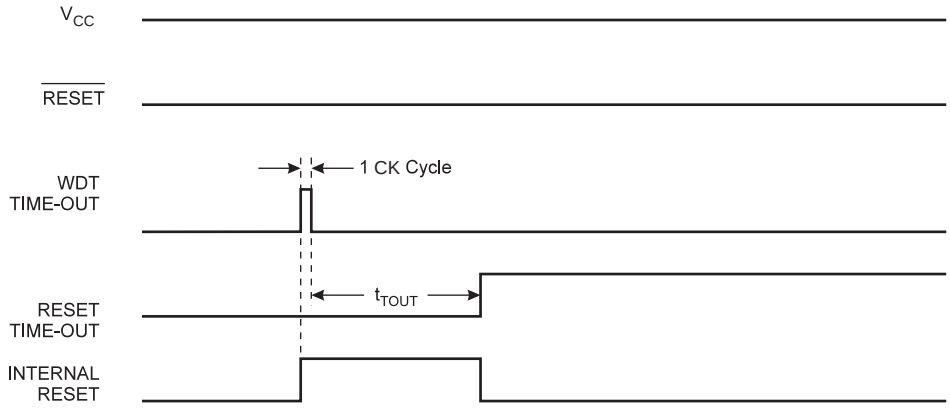
Figure 19. 工作过程中发生掉电检测复位



看门狗复位

看门狗定时器溢出时将产生持续时间为 1 个 CK 周期的复位脉冲。在脉冲的下降沿，延时定时器开始对 t_{TOUT} 记数。请参见 P39 以了解看门狗定时器的具体操作过程。

Figure 20. 工作过程中发生看门狗复位



MCU 控制和状态寄存器 - MCUCSR

MCU 控制和状态寄存器提供了有关引起 MCU 复位的复位源的信息。

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
读 / 写	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	见位说明					

- **Bit 4 – JTRF: JTAG 复位标志**

通过 JTAG 指令 AVR_RESET 可以使 JTAG 复位寄存器置位，并引发 MCU 复位，并使 JTRF 置位。上电复位将使其清零，也可以通过写 ”0” 来清除。

- **Bit 3 – WDRF: 看门狗复位标志**

看门狗复位发生时置位。上电复位将使其清零，也可以通过写 ”0” 来清除。

- **Bit 2 – BORF: 掉电检测复位标志**

掉电检测复位发生时置位。上电复位将使其清零，也可以通过写 ”0” 来清除。

- **Bit 1 – EXTRF: 外部复位标志**

外部复位发生时置位。上电复位将使其清零，也可以通过写 ”0” 来清除。

- **Bit 0 – PORF: 上电复位标志**

上电复位发生时置位。只能通过写 ”0” 来清除。

为了使用这些复位标志来识别复位条件，用户应该尽早读取此寄存器的数据，然后将其复位。如果在其他复位发生之前将此寄存器复位，则后续复位源可以通过检查复位标志来了解。

片内基准电压

ATmega32 具有片内能隙基准源，用于掉电检测，或者是作为模拟比较器或 ADC 的输入。ADC 的 2.56V 基准电压由此片内能隙基准源产生。

基准电压使能信号和启动时间

电压基准的启动时间可能影响其工作方式。启动时间列于 Table 16。为了降低功耗，可以控制基准源仅在如下情况打开：

1. BOD 使能 (熔丝位 BODEN 被编程)
2. 能隙基准源连接到模拟比较器 (ACSR 寄存器的 ACBG 置位)
3. ADC 使能

因此，当 BOD 被禁止时，置位 ACBG 或使能 ADC 后要启动基准源。为了降低掉电模式的功耗，用户可以禁止上述三种条件，并在进入掉电模式之前关闭基准源。

Table 16. 内部电压基准源的特性

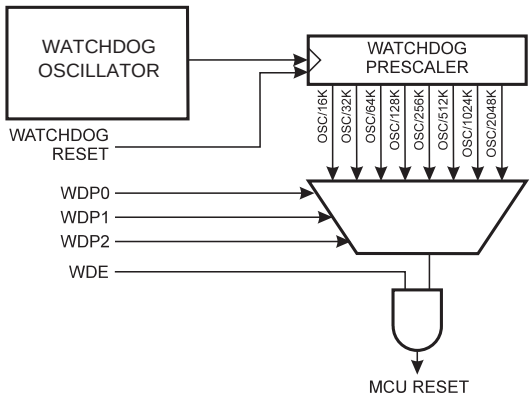
符号	参数	最小值	典型值	最大值	单位
V_{BG}	能隙基准源电压	1.15	1.23	1.35	V
t_{BG}	能隙基准源启动时间		40	70	µs
I_{BG}	能隙基准源功耗		10		µA

看门狗定时器

看门狗定时器由独立的 1 Mhz 片内振荡器驱动。这是 V_{CC} = 5V 时的典型值。请参见特性数据以了解其他 V_{CC} 电平下的典型值。通过设置看门狗定时器的预分频器可以调节看门狗复位的时间间隔，如 P40 Table 17 所示。看门狗复位指令 WDR 用来复位看门狗定时器。此外，禁止看门狗定时器或发生复位时定时器也被复位。复位时间有 8 个选项。如果没有及时复位定时器，一旦时间超过复位周期， ATmega32 就复位，并执行复位向量指向的程序。具体的看门狗复位时序在 P38 有说明。

为了防止无意之间禁止看门狗定时器，在看门狗禁用后必须跟一个特定的修改序列。详见看门狗定时器控制寄存器。

Figure 21. 看门狗定时器



看门狗定时器控制寄存器 - WDTCR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	WDTCR
读 / 写	R	R	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bits 7..5 – Res: 保留**

保留位，读操作返回值为零。

- **Bit 4 – WDTOE: 看门狗关使能**

清零 WDE 时必须置位 WDTOE，否则不能禁止看门狗。一旦置位，硬件将在紧接的 4 个时钟周期之后将其清零。请参考有关 WDE 的说明来禁止看门狗。

- **Bit 3 – WDE: 看门狗使能**

WDE 为 "1“ 时，看门狗使能，否则看门狗将被禁止。只有在 WDTOE 为 "1“ 时 WDE 才能清零。以下为关闭看门狗的步骤：

1. 在同一个指令内对 WDTOE 和 WDE 写 "1“，即使 WDE 已经为 "1“
2. 在紧接的 4 个时钟周期之内对 WDE 写 "0”

- **Bits 2..0 – WDP2, WDP1, WDP0: 看门狗定时器预分频器 2, 1 和 0**

WDP2、WDP1 和 WDP0 决定看门狗定时器的预分频器，其预分频值及相应的溢出周期如 Table 18 所示。

Table 17. 看门狗定时器预分频器选项

WDP2	WDP1	WDP0	WDT 振荡器周期	V_{CC} = 3.0V 时典型的溢出周期	V_{CC} = 5.0V 时典型的溢出周期
0	0	0	16K (16,384)	17.1 ms	16.3 ms
0	0	1	32K (32,768)	34.3 ms	32.5 ms
0	1	0	64K (65,536)	68.5 ms	65 ms
0	1	1	128K (131,072)	0.14 s	0.13 s
1	0	0	256K (262,144)	0.27 s	0.26 s
1	0	1	512K (524,288)	0.55 s	0.52 s
1	1	0	1,024K (1,048,576)	1.1 s	1.0 s
1	1	1	2,048K (2,097,152)	2.2 s	2.1 s

下面的例子分别用汇编和 C 实现了关闭 WDT 的操作。在此假定中断处于用户控制之下（比如禁止全局中断），因而在执行下面程序时中断不会发生。

汇编代码例程

```
WDT_off:
  ; 置位 WDTOE 和 WDE
  ldi  r16, (1<<WDTOE)|(1<<WDE)
  out  WDTCR, r16
  ; 关闭 WDT
  ldi  r16, (0<<WDE)
  out  WDTCR, r16
  ret
```

C 代码例程

```
void WDT_off(void)
{
  /* 置位 WDTOE 和 WDE */
  WDTCR = (1<<WDTOE) | (1<<WDE);
  /* 关闭 WDT */
  WDTCR = 0x00;
}
```

中断

本节说明ATmega32的中断处理。更一般的AVR中断处理请参见 P11 “复位与中断处理” 。

ATmega32 的中断向量

Table 18. 复位和中断向量

向量号	程序地址 (2)	中断源	中断定义
1	$000(1)	RESET	外部引脚电平引发的复位，上电复位，掉电检测复位，看门狗复位，以及 JTAG AVR 复位
2	$002	INT0	外部中断请求 0
3	$004	INT1	外部中断请求 1
4	$006	INT2	外部中断请求 2
5	$008	TIMER2 COMP	定时器 / 计数器 2 比较匹配
6	$00A	TIMER2 OVF	定时器 / 计数器 2 溢出
7	$00C	TIMER1 CAPT	定时器 / 计数器 1 事件捕捉
8	$00E	TIMER1 COMPA	定时器 / 计数器 1 比较匹配 A
9	$010	TIMER1 COMPB	定时器 / 计数器 1 比较匹配 B
10	$012	TIMER1 OVF	定时器 / 计数器 1 溢出
11	$014	TIMER0 COMP	定时器 / 计数器 0 比较匹配
12	$016	TIMER0 OVF	定时器 / 计数器 0 溢出
13	$018	SPI, STC	SPI 串行传输结束
14	$01A	USART, RXC	USART， Rx 结束
15	$01C	USART, UDRE	USART 数据寄存器空
16	$01E	USART, TXC	USART， Tx 结束
17	$020	ADC	ADC 转换结束
18	$022	EE_RDY	EEPROM 就绪
19	$024	ANA_COMP	模拟比较器
20	$026	TWI	两线串行接口
21	$028	SPM_RDY	保存程序存储器内容就绪

Notes: 1. 熔丝位 BOOTRST 被编程时 ,MCU 复位后程序跳转到 Boot Loader。请参见 P228 “ 支持引导装入程序 – 在写的同时可以读 (RWW, Read-While-Write) 的自我编程能力 ” 。
2. 当寄存器GICR的IVSEL置位时 ,中断向量转移到Boot区的起始地址。此时各个中断向量的实际地址为表中地址与 Boot 区起始地址之和。

Table 19给出了不同的BOOTRST/IVSEL设置下的复位和中断向量的位置。如果程序永远不使能中断，中断向量就没有意义。用户可以在此直接写程序。同样，如果复位向量位于应用区，而其他中断向量位于 Boot 区，则复位向量之后可以直接写程序。反过来亦是如此。

Table 19. 复位和中断向量位置的确定 (1)

BOOTRST	IVSEL	复位地址	中断向量起始地址
1	0	$0000	$0002
1	1	$0000	Boot 区复位地址 + $0002
0	0	Boot 区复位地址	$0002
0	1	Boot 区复位地址	Boot 区复位地址 + $0002

Note: 1. Boot 区复位地址列于 P237 Table 100 。对于熔丝位 BOOTRST ,“1” 表示未编程，“0” 表示已编程。

ATmega32 典型的复位和中断设置如下：

```
地址              符号    代码                说明
$000              jmp   RESET             ; 复位中断向量
$002              jmp   EXT_INT0          ; IRQ0 中断向量
$004              jmp   EXT_INT1          ; IRQ1 中断向量
$006              jmp   EXT_INT2          ; IRQ2 中断向量
$008              jmp   TIM2_COMP         ; Timer2 比较中断向量
$00A              jmp   TIM2_OVF          ; Timer2 溢出中断向量
$00C              jmp   TIM1_CAPT         ; Timer1 捕捉中断向量
$00E              jmp   TIM1_COMPA        ; Timer1 比较 A 中断向量
$010              jmp   TIM1_COMPB        ; Timer1 比较 B 中断向量
$012              jmp   TIM1_OVF          ; Timer1 溢出中断向量
$014              jmp   TIM0_COMP         ; Timer0 比较中断向量
$016              jmp   TIM0_OVF          ; Timer0 溢出中断向量
$018              jmp   SPI_STC           ; SPI 传输结束中断向量
$01A              jmp   USART_RXC         ; USART RX 结束中断向量
$01C              jmp   USART_UDRE        ; UDR 空中断向量
$01E              jmp   USART_TXC         ; USART TX 结束中断向量
$020              jmp   ADC               ; ADC 转换结束中断向量
$022              jmp   EE_RDY            ; EEPROM 就绪中断向量
$024              jmp   ANA_COMP          ; 模拟比较器中断向量
$026              jmp   TWI               ; 两线串行接口中断向量
$028              jmp   SPM_RDY           ; SPM 就绪中断向量
;
$02A     RESET:   ldi   r16,high(RAMEND) ; 主程序
$02B              out   SPH,r16           ; 设置堆栈指针为 RAM 的顶部
$02C              ldi   r16,low(RAMEND)
$02D              out   SPL,r16
$02E              sei                     ; 使能中断
$02F              <instr>  xxx
...      ...      ...
```

当熔丝位 BOOTRST 未编程，Boot 区为 4K 字节，且寄存器 GICR 的 IVSEL 置位时，典型的复位和中断设置如下：

```
地址              符号     代码                说明
$000      RESET:    ldi   r16,high(RAMEND) ; 主程序
$001                out   SPH,r16          ; 设置堆栈指针为 RAM 的顶部
$002              ldi   r16,low(RAMEND)
$003                out   SPL,r16
$004                sei                    ; 使能中断
$005                <instr>  xxx
;
.org $3802
$3802               jmp   EXT_INT0         ; IRQ0 中断向量
$3804               jmp   EXT_INT1         ; IRQ1 中断向量
...       ....      ..                     ;
$3828               jmp   SPM_RDY          ; SPM 就绪中断向量
```

当熔丝位 BOOTRST 已编程，且 Boot 区为 4K 字节时，典型的复位和中断设置如下：

```
地址              符号     代码                说明
.org $002
$002                jmp   EXT_INT0         ; IRQ0 中断向量
$004                jmp   EXT_INT1         ; IRQ1 中断向量
...       ....      ..                     ;
$028                jmp   SPM_RDY          ; SPM 就绪中断向量
;
.org $3800
$3800     RESET:    ldi   r16,high(RAMEND) ; 主程序
$3801               out   SPH,r16          ; 设置堆栈指针为 RAM 的顶部
$3802               ldi   r16,low(RAMEND)
$3803               out   SPL,r16
$3804               sei                    ; 使能中断
$3805               <instr>  xxx
```

当熔丝位 BOOTRST 已编程，Boot 区为 4K 字节，且寄存器 GICR 的 IVSEL 置位时，典型的复位和中断设置如下：

```
地址              符号     代码                说明
.org $3800
$3800               jmp   RESET        ; Reset 中断向量
$3802               jmp   EXT_INT0     ; IRQ0 中断向量
$3804               jmp   EXT_INT1     ; IRQ1 中断向量
...       ....      ..                 ;
$3828               jmp   SPM_RDY      ; SPM 就绪中断向量
;
$382A     RESET:    ldi   r16,high(RAMEND) ; 主程序
$382B               out   SPH,r16          ; 设置堆栈指针为 RAM 的顶部
$382C               ldi   r16,low(RAMEND)
$382D               out   SPL,r16
$382E               sei                    ; 使能中断
$382F               <instr>  xxx
```

在应用区和 Boot 区之间移动中断

通用中断控制寄存器决定中断向量表的放置地址。

通用中断控制寄存器 - GICR

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
读 / 写	R/W	R/W	R/W	R	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 1 – IVSEL: 中断向量选择**

当 IVSEL 为 "0“ 时，中断向量位于 Flash 存储器的起始地址；当 IVSEL 为 "1“ 时，中断向量转移到 Boot 区的起始地址。实际的 Boot 区起始地址由熔丝位 BOOTSZ 确定。具体请参考 P228 “ 支持引导装入程序 – 在写的同时可以读 (RWW, Read-While-Write) 的自我编程能力 ” 。为了防止无意识地改变中断向量表，修改 IVSEL 时需要遵照如下过程：

1. 置位中断向量修改使能位 IVCE
2. 在紧接的 4 个时钟周期里将需要的数据写入 IVSEL，同时对 IVCE 写 ”0”

执行上述序列时中断自动被禁止。其实，在置位 IVCE 时中断就被禁止了，并一直保持到写 IVSEL 操作之后的下一条语句。如果没有 IVSEL 写操作，则中断在置位 IVCE 之后的 4 个时钟周期保持禁止。需要注意的是，虽然中断被自动禁止，但状态寄存器的位 I 的值并不受此操作的影响。

Note: 若中断向量位于Boot区，且Boot锁定位BLB02被编程，则执行应用区的程序时中断被禁止；若中断向量位于应用区，且 Boot 锁定位 BLB12 被编程，则执行 Boot 区的程序时中断被禁止。有关 Boot 锁定位的细节请参见 P228 “ 支持引导装入程序 – 在写的同时可以读 (RWW, Read-While-Write) 的自我编程能力 ” 。

- **Bit 0 – IVCE: 中断向量修改使能**

改变 IVSEL 时 IVCE 必须置位。在 IVCE 或 IVSEL 写操作之后 4 个时钟周期， IVCE 被硬件清零。如前面所述，置位 IVCE 将禁止中断。代码如下：

汇编代码例程：

```
Move_interrupts:
  ; 使能中断向量的修改
  ldi  r16, (1<<IVCE)
  out  GICR, r16
  ; 将中断向量转移到 boot Flash 区
  ldi  r16, (1<<IVSEL)
  out  GICR, r16
  ret
```

C 代码例程

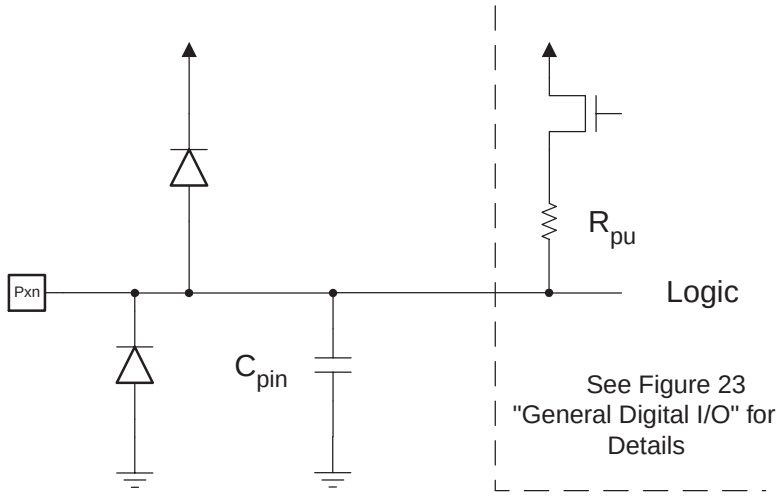
```
void Move_interrupts(void)
{
  /* 使能中断向量的修改 */
  GICR = (1<<IVCE);
  /* 将中断向量转移到 boot Flash 区 */
  GICR = (1<<IVSEL);
}
```

I/O 端口

介绍

作为通用数字 I/O 使用时，所有 AVR I/O 端口都具有真正的读 - 修改 - 写功能。这意味着用 SBI 或 CBI 指令改变某些管脚的方向 (或者是端口电平、禁止 / 使能上拉电阻) 时不会无意地改变其他管脚的方向 (或者是端口电平、禁止 / 使能上拉电阻)。输出缓冲器具有对称的驱动能力，可以输出或吸收大电流，直接驱动 LED。所有的端口引脚都具有与电压无关的上拉电阻。并有保护二极管与 V_{CC} 和地相连，如 Figure 22 所示。请参见 P269 “ 电气特性 ” 以了解完整的参数列表。

Figure 22. I/O 引脚等效原理图



本节所有的寄存器和位以通用格式表示：小写的 “x” 表示端口的序号，而小写的 “n” 代表位的序号。但是在程序里要写完整。例如， PORTB3 表示端口 B 的第 3 位，而本节的通用格式为 PORTxn。物理 I/O 寄存器和位定义列于 P61 “I/O 端口寄存器的说明 ” 。

每个端口都有三个 I/O 存储器地址：数据寄存器 – PORTx、数据方向寄存器 – DDRx 和端口输入引脚 – PINx。数据寄存器和数据方向寄存器为读 / 写寄存器，而端口输入引脚为只读寄存器。但是需要特别注意的是，对 PINx 寄存器某一位写入逻辑 "1“ 将造成数据寄存器相应位的数据发生 "0“ 与 “1“ 的交替变化。当寄存器 SFIOR 的上拉禁止位 PUD 置位时所有端口引脚的上拉电阻都被禁止。

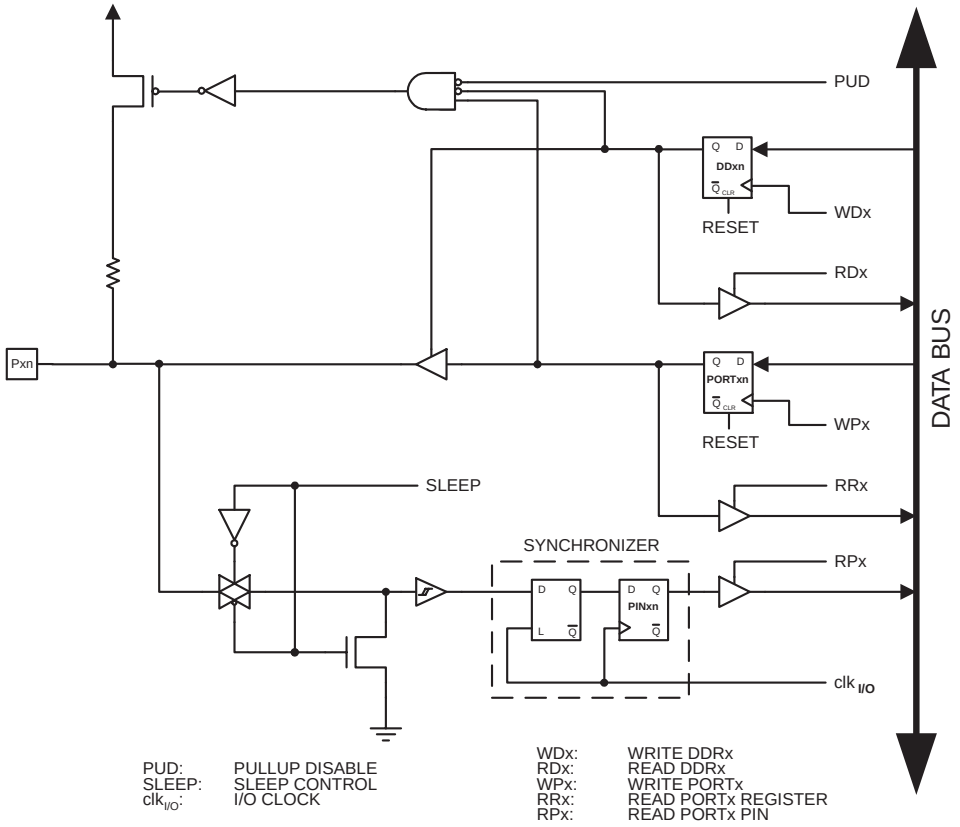
作为通用数字 I/O 时的端口请参见 P47 “ 作为通用数字 I/O 的端口 ” 。多数端口引脚是与第二功能复用的，如 P52 “ 端口的第二功能 ” 所示。请参见各个模块的具体说明以了解引脚的第二功能。

使能某些引脚的第二功能不会影响其他属于同一端口的引脚用于通用数字 I/O 目的。

作为通用数字 I/O 的端口

端口为具有可选上拉电阻的双向 I/O 端口。Figure 23 为一个 I/O 端口引脚的说明。

Figure 23. 通用数字 I/O(1)



Note: 1. WPx, WDx, RRx, RPx 和 RDx 对于同一端口的所有引脚都是一样的。$clk_{I/O}$, SLEEP 和 PUD 则对所有的端口都是一样的。

配置引脚

每个端口引脚都具有三个寄存器位 : DDxn、PORTxn 和 PINxn，如 P61 “I/O 端口寄存器的说明 ” 所示。 DDxn 位于 DDRx 寄存器， PORTxn 位于 PORTx 寄存器， PINxn 位于 PINx 寄存器。

DDxn 用来选择引脚的方向。 DDxn 为 "1“ 时， Pxn 配置为输出，否则配置为输入。

引脚配置为输入时，若 PORTxn 为 "1“，上拉电阻将使能。如果需要关闭这个上拉电阻，可以将 PORTxn 清零，或者将这个引脚配置为输出。复位时各引脚为高阻态，即使此时并没有时钟在运行。

当引脚配置为输出时，若 PORTxn 为 "1“，引脚输出高电平 ("1“)，否则输出低电平 (“0“)。

在 (高阻态) 三态 ({DDxn, PORTxn} = 0b00) 输出高电平 ({DDxn, PORTxn} = 0b11) 两种状态之间进行切换时，上拉电阻使能 ({DDxn, PORTxn} = 0b01) 或输出低电平 ({DDxn, PORTxn} = 0b10) 这两种模式必然会有一个发生。通常，上拉电阻使能是完全可以接受的，因为高阻环境不在意是强高电平输出还是上拉输出。如果使用情况不是这样子，可以通过置位 SFIOR 寄存器的 PUD 来禁止所有端口的上拉电阻。

在上拉输入和输出低电平之间切换也有同样的问题。用户必须选择高阻态 ({DDxn, PORTxn} = 0b00) 或输出高电平 ({DDxn, PORTxn} = 0b11) 作为中间步骤。

Table 20 总结了引脚的控制信号。

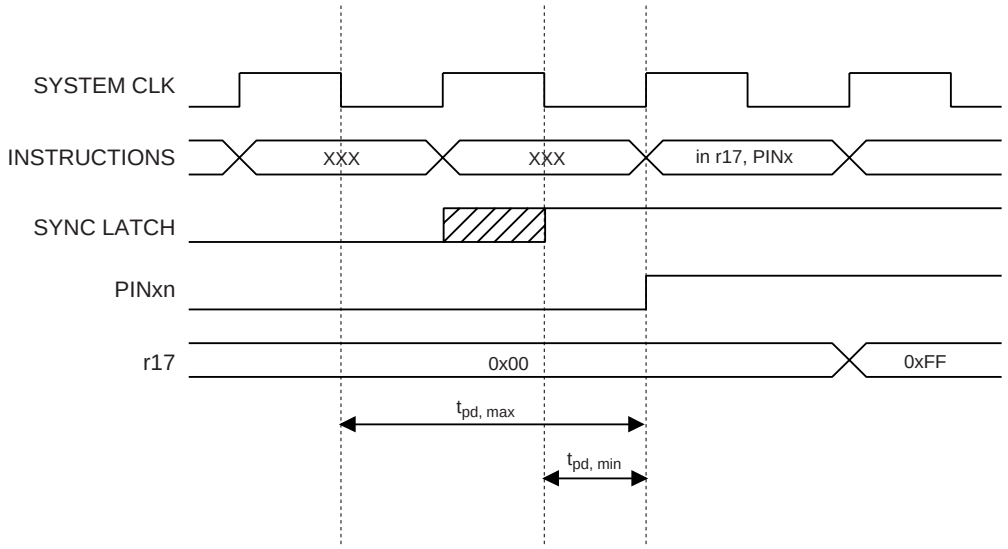
Table 20. 端口引脚配置

DDxn	PORTxn	PUD (in SFIOR)	I/O	上拉电阻	说明
0	0	X	输入	No	高阻态 (Hi-Z)
0	1	0	输入	Yes	被外部电路拉低时将输出电流
0	1	1	输入	No	高阻态 (Hi-Z)
1	0	X	输出	No	输出低电平 (漏电流)
1	1	X	输出	No	输出高电平 (源电流)

读取引脚上的数据

不论如何配置 DDxn , 都可以通过读取 PINxn 寄存器来获得引脚电平。如 Figure 23 所示，PINxn 寄存器的各个位与其前面的锁存器组成了一个同步器。这样就可以避免在内部时钟状态发生改变的短时间范围内由于引脚电平变化而造成的信号不稳定。其缺点是引入了延迟。 Figure 24 为读取引脚电平时同步器的时序图。最大和最小传输延迟分别为 $t_{pd,max}$ 和 $t_{pd,min}$。

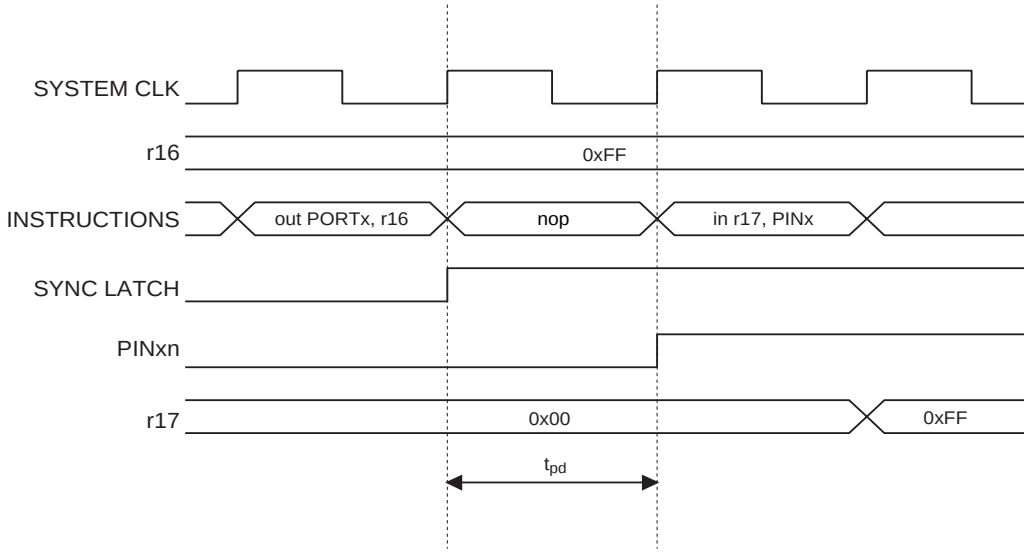
Figure 24. 读取引脚数据时的同步



下面考虑第一个系统时钟下降沿之后起始的时钟周期。当时钟信号为低时锁存器是关闭的；而时钟信号为高时信号可以自由通过，如图中 SYNC LATCH 信号的阴影区所示。时钟为低时信号即被锁存 , 然后在紧接着的系统时钟上升沿锁存到 PINxn 寄存器。如 $t_{pd,max}$ 和 $t_{pd,min}$ 所示，引脚上的信号转换延迟界于 ½ ~ 1½ 个系统时钟。

如 Figure 25 所示 , 读取软件赋予的引脚电平时需要在赋值指令 *out* 和读取指令 *in* 之间有一个时钟周期的间隔，如 *nop* 指令。*out* 指令在时钟的上升沿置位 SYNC LATCH 信号。此时同步器的延迟时间 t_{pd} 为一个系统时钟。

Figure 25. 读取软件赋予的引脚电平的同步



下面的例子演示了如何置位端口 B 的引脚 0 和 1，清零引脚 2 和 3，以及将引脚 4 到 7 设置为输入，并且为引脚 6 和 7 设置上拉电阻。然后将各个引脚的数据读回来。如前面讨论的那样，我们在输出和输入语句之间插入了一个 *nop* 指令。

汇编代码例程 (1)

```
...
;定义上拉电阻和设置高电平输出
; 为端口引脚定义方向
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
; 为了同步插入 nop 指令
nop
; 读取端口引脚
in    r16,PINB
...
```

C 代码例程 (1)

```
unsigned char i;
...
/* 定义上拉电阻和设置高电平输出 */
/* 为端口引脚定义方向 */
PORTB = (1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0);
DDRB = (1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0);
/* 为了同步插入 nop 指令 */
_NOP();
/* 读取端口引脚 */
i = PINB;
...
```

Note: 1. 在汇编程序里使用了两个暂存器。其目的是为了使整个操作过程的时间最短。通过拉高引脚 0、 1、 6 与 7，直到方向位设置正确，定义位 2、 3 为低，且重新定义为 0 与 1 为强驱动。

数字输入使能和睡眠模式

如 Figure 23 所示，数字输入信号 (施密特触发器的输入) 可以钳位到地。图中的 SLEEP 信号由 MCU 休眠控制器在各种掉电模式、省电模式以及 Standby 模式下设置，以防止在输入悬空或模拟输入电平接近 $V_{CC}/2$ 时消耗太多的电流。

引脚作为外部中断输入时 SLEEP 信号无效。但若外部中断没有使能， SLEEP 信号仍然有效。引脚的第二功能使能时 SLEEP 也让位于第二功能，如 P52 “ 端口的第二功能 ” 里描述的那样。

如果逻辑高电平 (“1”) 出现在一个被设置为 " 上升沿、下降沿或任何逻辑电平变化都引起中断 " 的外部异步中断引脚上，即使该外部中断未被使能，但从上述休眠模式唤醒时，相应的外部中断标志位仍会被置 "1”。这是因为引脚电平在休眠模式下被钳位到 "0” 电平。唤醒过程造成了引脚电平从 "0” 到 ”1” 的变化。

未连接引脚的处理

如果有引脚未被使用，建议给这些引脚赋予一个确定电平。虽然如上文所述，在深层休眠模式下大多数数字输入被禁用，但还是需要避免因引脚没有确定的电平而造成悬空引脚在其它数字输入使能模式 (复位、工作模式、空闲模式) 消耗电流。

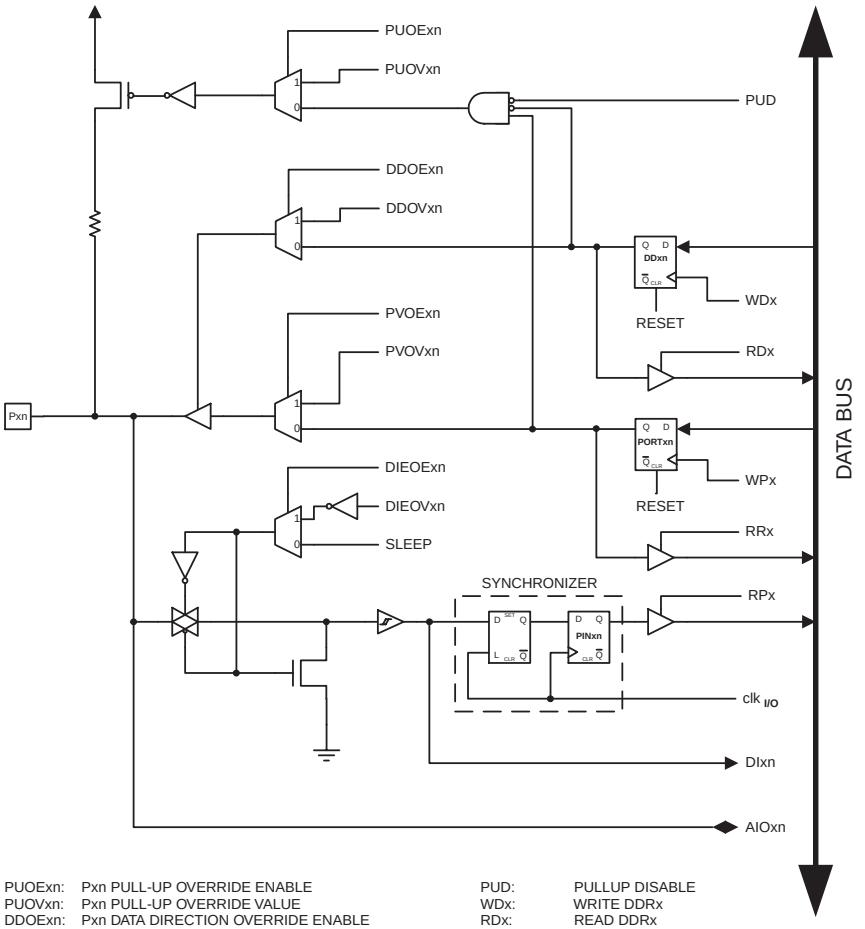
最简单的保证未用引脚具有确定电平的方法是使能内部上拉电阻。但要注意的是复位时上拉电阻将被禁用。如果复位时的功耗也有严格要求则建议使用外部上拉或下拉电阻。不

推荐直接将未用引脚与 V_{CC} 或 GND 连接 , 因为这样可能会在引脚偶然作为输出时出现冲击电流。

端口的第二功能

除了通用数字 I/O 功能之外 , 大多数端口引脚都具有第二功能。Figure 26 说明了由 Figure 23 简化得出的端口引脚控制信号是如何被第二功能取代的。这些被重载的信号不会出现在所有的端口引脚 , 但本图可以看作是适合于 AVR 系列处理器所有端口引脚的一般说明。

Figure 26. 端口的第二功能 (1)



PUOExn:	Pxn PULL-UP OVERRIDE ENABLE	PUD:	PULLUP DISABLE
PUOVxn:	Pxn PULL-UP OVERRIDE VALUE	WDx:	WRITE DDRx
DDOExn:	Pxn DATA DIRECTION OVERRIDE ENABLE	RDx:	READ DDRx
DDOVxn:	Pxn DATA DIRECTION OVERRIDE VALUE	RRx:	READ PORTx REGISTER
PVOExn:	Pxn PORT VALUE OVERRIDE ENABLE	WPx:	WRITE PORTx
PVOVxn:	Pxn PORT VALUE OVERRIDE VALUE	RPx:	READ PORTx PIN
DIEOExn:	Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE	$clk_{I/O}$:	I/O CLOCK
DIEOVxn:	Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE	DIxn:	DIGITAL INPUT PIN n ON PORTx
SLEEP:	SLEEP CONTROL	AIOxn:	ANALOG INPUT/OUTPUT PIN n ON PORTx

Note: 1. WPx, WDx, RRx, RPx和RDx对于同一个端口的所有引脚都是一样的。$clk_{I/O}$, SLEEP和 PUD 则对所有的端口都是一样的。其他信号只对某一个引脚有效。

Table 21 为重载信号的简介。表中没有给出 Figure 26 的引脚和端口索引。这些重载信号是由第二功能模块产生的。

Table 21. 第二功能重载信号的一般说明

信号名称	全 称	说 明
PUOE	上拉电阻 重载使能	若此信号置位，上拉电阻使能将受控于 PUOV ；若此信号清零，则 {DDxn, PORTxn, PUD} = 0b010 时上拉电阻使能。
PUOV	上拉电阻 重载值	若 PUOE 置位，则不论 DDxn、PORTxn 和 PUD 寄存器各个位如何配置， PUOV 置位 / 清零时上拉电阻使能 / 禁止
DDOE	数据方向 重载使能	如果此信号置位，则输出驱动使能由 DDOV 控制；若此信号清零，输出驱动使能由 DDxn 寄存器控制。
DDOV	数据方向 重载值	若 DDOE 置位，则 DDOV 置位 / 清零时输出驱动使能 / 禁止，而不管 DDxn 寄存器的设置如何。
PVOE	端口数据 重载使能	如果这个信号置位，且输出驱动使能，端口数据由 PVOV 控制；若 PVOE 清零，且输出驱动使能，端口数据由寄存器 PORTxn 控制。
PVOV	端口数据 重载值	若 PVOE 置位，端口值设置为 PVOV，而不管寄存器 PORTxn 如何设置。
DIEOE	数字输入使能 覆盖使能	如果这个信号置位，数字输入使能由 DIEOV 控制；若 DIEOE 清零，数字输入使能由 MCU 的状态确定 (正常模式，睡眠模式)。
DIEOV	数字输入使能 覆盖值	若 DIEOE 置位， DIEOV 置位 / 清零时数字输入使能 / 禁止，而不管 MCU 的状态如何 (正常模式，睡眠模式)。
DI	数字输入	此信号为第二功能的数字输入。在图中，这个信号与施密特触发相连，并且在同步器之前。除非数字输入用作时钟源，否则第二功能模块将使用自己的同步器。
AIO	模拟信号 输入 / 输出	模拟输入 / 输出。信号直接与引脚接点相连，而且可以用作双向端口。

下面的几小节将简单地说明每个端口的第二功能以及相关的信号。具体请参考有关第二功能的说明。

特殊功能 I/O 寄存器 - SFIOR

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
读 / 写	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 2 – PUD: 禁用上拉电阻**

置位时，即使将寄存器 DDxn 和 PORTxn 配置为使能上拉电阻 ({DDxn, PORTxn} = 0b01)，I/O 端口的上拉电阻也被禁止。请参见 P48 “ 配置引脚 ” 。

端口 A 的第二功能

端口 A 作为 ADC 模拟输入的第二功能示于 Table 22。如果端口 A 的部分引脚置为输出，当转换时不能切换，否则会影响转换结果。

Table 22. 端口 A 的第二功能

端口引脚	第二功能
PA7	ADC7 (ADC 输入通道 7)
PA6	ADC6 (ADC 输入通道 6)
PA5	ADC5 (ADC 输入通道 5)
PA4	ADC4 (ADC 输入通道 4)
PA3	ADC3 (ADC 输入通道 3)
PA2	ADC2 (ADC 输入通道 2)
PA1	ADC1 (ADC 输入通道 1)
PA0	ADC0 (ADC 输入通道 0)

Table 23 和 Table 24 给出了端口 A 第二功能与 P52 Figure 26 重载信号的对应关系。

Table 23. PA7..PA4 的第二功能重载信号

信号名称	PA7/ADC7	PA6/ADC6	PA5/ADC5	PA4/ADC4
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	ADC7 输入	ADC6 输入	ADC5 输入	ADC4 输入

Table 24. PA3..PA0 的第二功能重载信号

信号名称	PA3/ADC3	PA2/ADC2	PA1/ADC1	PA0/ADC0
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	ADC3 输入	ADC2 输入	ADC1 输入	ADC0 输入

端口 B 的第二功能

端口 B 的第二功能列于 Table 25。

Table 25. 端口 B 的第二功能

端口引脚	第二功能
PB7	SCK (SPI 总线的串行时钟)
PB6	MISO (SPI 总线的主机输入 / 从机输出信号)
PB5	MOSI (SPI 总线的主机输出 / 从机输入信号)
PB4	$\overline{SS}$ (SPI 从机选择引脚)
PB3	AIN1 (模拟比较负输入) OC0 (T/C0 输出比较匹配输出)
PB2	AIN0 (模拟比较正输入) INT2 (外部中断 2 输入)
PB1	T1 (T/C1 外部计数器输入)
PB0	T0 (T/C0 外部计数器输入) XCK (USART 外部时钟输入 / 输出)

引脚配置如下：

- **SCK – 端口 B, Bit 7**

SCK : SPI 通道的主机时钟输出 , 从机时钟输入端口。工作于从机模式时 , 不论 DDB7 设置如何 , 这个引脚都将设置为输入。工作于主机模式时 , 这个引脚的数据方向由 DDB7 控制。设置为输入后，上拉电阻由 PORTB7 控制。

- **MISO – 端口 B, Bit 6**

MISO : SPI 通道的主机数据输入 , 从机数据输出端口。工作于主机模式时 , 不论 DDB6 设置如何 , 这个引脚都将设置为输入。工作于从机模式时 , 这个引脚的数据方向由 DDB6 控制。设置为输入后，上拉电阻由 PORTB6 控制。

- **MOSI – 端口 B, Bit 5**

MOSI: SPI 通道的主机数据输出,从机数据输入端口。工作于从机模式时,不论 DDB5 设置如何,这个引脚都将设置为输入。当工作于主机模式时,这个引脚的数据方向由 DDB5 控制。设置为输入后,上拉电阻由 PORTB5 控制。

- **$\overline{SS}$ – 端口 B, Bit 4**

$\overline{SS}$:从机选择输入。工作于从机模式时,不论 DDB4 设置如何,这个引脚都将设置为输入。当此引脚为低时 SPI 被激活。工作于主机模式时,这个引脚的数据方向由 DDB4 控制。设置为输入后,上拉电阻由 PORTB4 控制。

- **AIN1/OC0 – 端口 B, Bit 3**

AIN1,模拟比较负输入。配置该引脚为输入时,切断内部上拉电阻,防止数字端口功能与模拟比较器功能相冲突。

OC0,输出比较匹配输出:PB3 引脚可作为 T/C0 比较匹配的外部输出。实现该功能时,PB3 引脚必须配置为输出 (设 DDB3 为 1) 。在 PWM 模式的定时功能中,OC0 引脚作为输出。

- **AIN0/INT2 – 端口 B, Bit 2**

AIN0,模拟比较正输入。配置该引脚为输入时,切断内部上拉电阻,防止数字端口功能与模拟比较器功能相冲突。

INT2,外部中断源 2:PB2 引脚作为 MCU 的外部中断源。

- **T1 – 端口 B, Bit 1**

T1,T/C1 计数器源。

- **T0/XCK – 端口 B, Bit 0**

T0, T/C0 计数器源。

XCK ,USART 外部时钟。数据方向寄存器 (DDB0) 控制时钟为输出 (DDB0 置位) 还是输入 (DDB0 清零)。只有当 USART 工作在同步模式时, XCK 引脚激活。

Table 26 与 Table 27 给出了端口 B 第二功能与 P52 Figure 26 重载信号的对应关系。SPI MSTR INPUT 和SPI SLAVE OUTPUT 构成了MISO信号,而MOSI可以分解为SPI MSTR OUTPUT 和 SPI SLAVE INPUT。

Table 26. PB7..PB4 的第二功能重载信号

信号名称	PB7/SCK	PB6/MISO	PB5/MOSI	PB4/$\overline{SS}$
PUOE	SPE • $\overline{MSTR}$	SPE • MSTR	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$
PUOV	PORTB7 • $\overline{PUD}$	PORTB6 • $\overline{PUD}$	PORTB5 • $\overline{PUD}$	PORTB4 • $\overline{PUD}$
DDOE	SPE • $\overline{MSTR}$	SPE • MSTR	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$
DDOV	0	0	0	0
PVOE	SPE • MSTR	SPE • $\overline{MSTR}$	SPE • MSTR	0
PVOV	SCK 输出	SPI 从机输出	SPI 主机输出	0
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	SCK 输入	SPI 主机输入	SPI 从机输入	SPI $\overline{SS}$
AIO	–	–	–	–

Table 27. PB3..PB0 的第二功能重载信号

信号名称	PB3/OC0/AIN1	PB2/INT2/AIN0	PB1/T1	PB0/T0/XCK
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	OC0 使能	0	0	UMSEL
PVOV	OC0	0	0	XCK 输出
DIEOE	0	INT2 使能	0	0
DIEOV	0	1	0	0
DI	–	INT2 输入	T1 输入	XCK 输入 /T0 输入
AIO	AIN1 输入	AIN0 输入	–	–

端口 C 的第二功能

端口 C 的第二功能示于 Table 28。若 JTAG 接口使能，即使出现复位，引脚 PC5(TDI)、PC3(TMS) 与 PC2(TCK) 的上拉电阻将被激活。

Table 28. 端口 C 的第二功能

端口引脚	第二功能
PC7	TOSC2 (定时振荡器引脚 2)
PC6	TOSC1 (定时振荡器引脚 1)
PC5	TDI (JTAG 测试数据输入)
PC4	TDO (JTAG 测试数据输出)
PC3	TMS (JTAG 测试模式选择)
PC2	TCK (JTAG 测试时钟)
PC1	SDA (两线串行总线数据输入 / 输出线)
PC0	SCL (两线串行总线时钟线)

第二功能配置如下：

- **TOSC2 – 端口 C, Bit 7**

TOSC2 , 定时振荡器引脚 2: 当寄存器 ASSR 的 AS2 位置 1 , 使能 T/C2 的异步时钟 , 引脚 PC7 与端口断开，成为振荡器放大器的反向输出。在这种模式下，晶体振荡器与该引脚相联，该引脚不能作为 I/O 引脚。

- **TOSC1 – 端口 C, Bit 6**

TOSC1 , 定时振荡器引脚 1: 当寄存器 ASSR 的 AS2 位置 1 , 使能 T/C2 的异步时钟 , 引脚 PC6 与端口断开，成为振荡器放大器的反向输出。在这种模式下，晶体振荡器与该引脚相联，该引脚不能作为 I/O 引脚。

- **TDI – 端口 C, Bit 5**

TDI ,JTAG 测试数据输入 : 串行输入数据移入指令寄存器或数据寄存器 (扫描链)。当 JTAG 接口使能，该引脚不能作为 I/O 引脚。

- **TDO – 端口 C, Bit 4**

TDO， JTAG 测试数据输入：串行输入数据移入指令寄存器或数据寄存器 (扫描链)。当 JTAG 接口使能，该引脚不能作为 I/O 引脚。

TD0 引脚在除 TAP 状态情况外为三态，进入移出数据状态。

- **TMS – 端口 C, Bit 3**

TMS，JTAG 测试模式选择：该引脚作为 TAP 控制器状态工具的定位。当 JTAG 接口使能，该引脚不能作为 I/O 引脚。

- **TCK – 端口 C, Bit 2**

TCK， JTAG 测试时钟：JTAG 工作在同步模式下。当 JTAG 接口使能，该引脚不能作为 I/O 引脚。

- **SDA – 端口 C, Bit 1**

SDA , 两线串行接口数据 : 当寄存器 TWCR 的 TWEN 位置 1 使能两线串行接口 , 引脚 PC1 不与端口相联，且成为两线串行接口的串行数据 I/O 引脚。在该模式下，在引脚处使用窄

带滤波器抑制低于 50 ns 的输入信号，且该引脚由斜率限制的开漏驱动器驱动。当该引脚使用两线串行接口，仍可由 PORTC1 位控制上拉。

- **SCL – 端口 C, Bit 0**

SCL，两线串行接口时钟：当 TWCR 寄存器的 TWEN 位置 1 使能两线串行接口，引脚 PC0 未与端口连接，成为两线串行接口的串行时钟 I/O 引脚。在该模式下，在引脚处使用窄带滤波器抑制低于 50 ns 的输入信号，且该引脚由斜率限制的开漏驱动器驱动。当该引脚使用两线串行接口，仍可由 PORTC0 位控制上拉。

Table 29 和 Table 30 给出了端口 C 第二功能与 P52 Figure 26 重载信号的对应关系。

Table 29. PC7..PC4 的第二功能重载信号

信号名称	PC7/TOSC2	PC6/TOSC1	PC5/TDI	PC4/TDO
PUOE	AS2	AS2	JTAGEN	JTAGEN
PUOV	0	0	1	0
DDOE	AS2	AS2	JTAGEN	JTAGEN
DDOV	0	0	0	SHIFT_IR + SHIFT_DR
PVOE	0	0	0	JTAGEN
PVOV	0	0	0	TDO
DIEOE	AS2	AS2	JTAGEN	JTAGEN
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	T/C2 OSC 输出	T/C2 OSC 输入	TDI	–

Table 30. PC3..PC0 的第二功能重载信号 (1)

信号名称	PC3/TMS	PC2/TCK	PC1/SDA	PC0/SCL
PUOE	JTAGEN	JTAGEN	TWEN	TWEN
PUOV	1	1	PORTC1 • $\overline{PUD}$	PORTC0 • $\overline{PUD}$
DDOE	JTAGEN	JTAGEN	TWEN	TWEN
DDOV	0	0	SDA_OUT	SCL_OUT
PVOE	0	0	TWEN	TWEN
PVOV	0	0	0	0
DIEOE	JTAGEN	JTAGEN	0	0
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	TMS	TCK	SDA 输入	SCL 输入

Note: 1. 使能后，两线串行接口使能输出引脚 PC0 与 PC1 的斜率控制。这在图中并未示出。另外，窄带滤波器连在图中给出的 AIO 输出端口与 TWI 的数字逻辑模块之间。

端口 D 的第二功能

端口 D 的第二功能列于 Table 31。

Table 31. 端口 D 的第二功能

端口引脚	第二功能
PD7	OC2 (T/C2 输出比较匹配输出)
PD6	ICP1 (T/C1 输入捕捉引脚)
PD5	OC1A (T/C1 输出比较 A 匹配输出)
PD4	OC1B (T/C1 输出比较 B 匹配输出)
PD3	INT1 (外部中断 1 的输入)
PD2	INT0 (外部中断 0 的输入)
PD1	TXD (USART 输出引脚)
PD0	RXD (USART 输入引脚)

第二功能配置如下：

- **OC2 – 端口 D, Bit 7**

OC2，T/C2 输出比较匹配输出：PD7 引脚作为 T/C2 输出比较外部输入。在该功能下引脚作为输出 (DDD7 置 1) 。在 PWM 模式的定时器功能中， OC2 引脚作为输出。

- **ICP – 端口 D, Bit 6**

ICP1 – 输入捕捉引脚：PD6 作为 T/C1 的输入捕捉引脚。

- **OC1A – 端口 D, Bit 5**

OC1A，输出比较匹配 A 输出：PD5 引脚作为 T/C1 输出比较 A 外部输入。在该功能下引脚作为输出 (DDD5 置 1) 。在 PWM 模式的定时器功能中， OC1A 引脚作为输出。

- **OC1B – 端口 D, Bit 4**

OC1A，输出比较匹配 A 输出：PD5 引脚作为 T/C1 输出比较 A 外部输入。在该功能下引脚作为输出 (DDD5 置 1) 。在 PWM 模式的定时器功能中， OC1A 引脚作为输出。

- **INT1 – 端口 D, Bit 3**

INT1，外部中断 1。 PD3 引脚作为 MCU 的外部中断源。

- **INT0 – 端口 D, Bit 2**

INT0，外部中断 0。 PD2 引脚作为 MCU 的外部中断源。

- **TXD – 端口 D, Bit 1**

TXD 是 USART 的数据发送引脚。当使能了 USART 的发送器后，这个引脚被强制设置为输出，此时 DDD1 不起作用。

- **RXD – 端口 D, Bit 0**

RXD 是 USART 的数据接收引脚。当使能了 USART 的接收器后，这个引脚被强制设置为输出，此时 DDD0 不起作用。但是 PORTD0 仍然控制上拉电阻。

Table 32 与 Table 33 将端口 D 的第二功能与 P52 Figure 26 的重载信号关联在了一起。

Table 32. PD7..PD4 的第二功能

信号名称	PD7/OC2	PD6/ICP	PD5/OC1A	PD4/OC1B
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	OC2 使能	0	OC1A 使能	OC1B 使能
PVOV	OC2	0	OC1A	OC1B
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	–	ICP 输入	–	–
AIO	–	–	–	–

Table 33. PD3..PD0 的第二功能

信号名称	PD3/INT1	PD2/INT0	PD1/TXD	PD0/RXD
PUOE	0	0	TXEN	RXEN
PUOV	0	0	0	PORTD0 • $\overline{PUD}$
DDOE	0	0	TXEN	RXEN
DDOV	0	0	1	0
PVOE	0	0	TXEN	0
PVOV	0	0	TXD	0
DIEOE	INT1 使能	INT0 使能	0	0
DIEOV	1	1	0	0
DI	INT1 输入	INT0 输入	–	RXD
AIO	–	–	–	–

I/O 端口寄存器的说明

端口 A 数据寄存器 - PORTA

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 A 数据方向寄存器 - DDRA

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 A 输入引脚地址 - PINA

Bit	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
读 / 写	R	R	R	R	R	R	R	R	
初始值	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

端口 B 数据寄存器 - PORTB

Bit	7	6	5	4	3	2	1	0	
	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 B 数据方向寄存器 - DDRB

Bit	7	6	5	4	3	2	1	0	
	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 B 输入引脚地址 - PINB

Bit	7	6	5	4	3	2	1	0	
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
读 / 写	R	R	R	R	R	R	R	R	
初始值	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

端口 C 数据寄存器 - PORTC

Bit	7	6	5	4	3	2	1	0	
	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 C 数据方向寄存器 - DDRC

Bit	7	6	5	4	3	2	1	0	
	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 C 输入引脚地址 - PINC

Bit	7	6	5	4	3	2	1	0	
	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
读 / 写	R	R	R	R	R	R	R	R	
初始值	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

端口 D 数据寄存器 - PORTD

Bit	7	6	5	4	3	2	1	0	
	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 D 数据方向寄存器 - DDRD

Bit	7	6	5	4	3	2	1	0	
	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD

读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

端口 D 输入引脚地址 - PIND

Bit	7	6	5	4	3	2	1	0	
	PIND7	**PIND6**	**PIND5**	**PIND4**	**PIND3**	**PIND2**	**PIND1**	**PIND0**	**PIND**
读 / 写	R	R	R	R	R	R	R	R	
初始值	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

外部中断

外部中断通过引脚 INT0、 INT1 与 INT2 触发。只要使能了中断，即使引脚 INT0..2 配置为输出，只要电平发生了合适的变化，中断也会触发。这个特点可以用来产生软件中断。通过设置 MCU 控制寄存器 MCUCR 与 MCU 控制与状态寄存器 MCUCSR，中断可以由下降沿、上升沿，或者是低电平触发 (INT2 为边沿触发中断)。当外部中断使能并且配置为电平触发 (INT0/INT1)，只要引脚电平为低，中断就会产生。若要求 INT0 与 INT1 在信号下降沿或上升沿触发， I/O 时钟必须工作，如 P22 “ 时钟系统及其分布 ” 说明的那样。INT0/INT1 的中断条件检测 INT2 则是异步的。也就是说，这些中断可以用来将器件从睡眠模式唤醒。在睡眠过程 (除了空闲模式) 中 I/O 时钟是停止的。

通过电平方式触发中断，从而将 MCU 从掉电模式唤醒时，要保证电平保持一定的时间，以降低 MCU 对噪声的敏感程度。电平以看门狗的频率检测两次。在 5.0V、 25°C 的条件下，看门狗的标称时钟周期为 1 µs。看门狗时钟受电压的影响，具体请参考 P269 “ 电气特性 ” 。只要在采样过程中出现了合适的电平，或是信号持续到启动过程的末尾， MCU 就会唤醒。启动过程由熔丝位 SUT 决定，如 P22 “ 系统时钟及时钟选项 ” 所示。若信号出现于两次采样过程，但在启动过程结束之前就消失了， MCU 仍将唤醒，但不再会引发中断了。要求的电平必须保持足够长的时间以使MCU结束唤醒过程，然后触发电平中断。

MCU 控制寄存器 - MCUCR

MCU 控制寄存器包含中断触发控制位与通用 MCU 功能

Bit	7	6	5	4	3	2	1	0	
	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	
初始值	0	0	0	0	0	0	0	0	

- **Bit 3, 2 – ISC11, ISC10: 中断触发方式控制 1 Bit1 与 Bit 0**

如果 SREG 寄存器的 I 标志位和相应的中断屏蔽位置位的话，外部中断 1 由引脚 INT1 激发。触发方式如 Table 34 所示。在检测边沿前 MCU 首先采样 INT1 引脚上的电平。如果选择了边沿触发方式或电平变化触发方式，那么持续时间大于一个时钟周期的脉冲将触发中断，过短的脉冲则不能保证触发中断。如果选择低电平触发方式，那么低电平必须保持到当前指令执行完成。

Table 34. 中断 1 触发方式控制

ISC11	ISC10	说明
0	0	INT1 为低电平时产生中断请求
0	1	INT1 引脚上任意的逻辑电平变化都将引发中断
1	0	INT1 的下降沿产生异步中断请求
1	1	INT1 的上升沿产生异步中断请求

- **Bit 1, 0 – ISC01, ISC00: 中断 0 触发方式控制 Bit 1 与 Bit 0**

如果 SREG 寄存器的 I 标志位和相应的中断屏蔽位置位的话。触发方式如 Table 35 所示，外部中断 0 由引脚 INT0 激发。在检测边沿前 MCU 首先采样 INT0 引脚上的电平。如果选择了边沿触发方式或电平变化触发方式，那么持续时间大于一个时钟周期的脉冲将触发中断，过短的脉冲则不能保证触发中断。如果选择低电平触发方式，那么低电平必须保持到当前指令执行完成。

Table 35. 中断 0 触发方式控制

ISC01	ISC00	说明
0	0	INT0 为低电平时产生中断请求
0	1	INT0 引脚上任意的逻辑电平变化都将引发中断
1	0	INT0 的下降沿产生异步中断请求
1	1	INT0 的上升沿产生异步中断请求

MCU 控制与状态寄存器 - MCUCSR

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
读 / 写	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	见位说明					

- **Bit 6 – ISC2: 中断 2 触发方式控制**

如果 SREG 寄存器的 I 标志和 GICR 寄存器相应的中断屏蔽位置位的话，异步外中断 2 由外部引脚 INT2 激活。若 ISC2 写 0，INT2 的下降沿激活中断。若 ISC2 写 1，INT2 的上升沿激活中断。INT2 的边沿触发方式是异步的。只要 INT2 引脚上产生宽度大于 Table 36 所示数据的脉冲就会引发中断。若选择了低电平中断，低电平必须保持到当前指令完成，然后才会产生中断。而且只要将引脚拉低，就会引发中断请求。改变 ISC2 时有可能发生中断。因此建议首先在寄存器 GICR 里清除相应的中断使能位 INT2，然后再改变 ISC2。最后，不要忘记在重新使能中断之前通过对 GIFR 寄存器的相应中断标志位 INTF2 写 '1' 使其清零。

Table 36. 异步 (外部) 中断特性

符号	参数	条件	最小值	典型值	最大值	单位
t_{INT}	异步 (外部) 中断的最小脉冲宽度			50		ns

通用中断控制寄存器 - GICR

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
读 / 写	R/W	R/W	R/W	R	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – INT1: 外部中断请求 1 使能**

当 INT1 为 '1'，而且状态寄存器 SREG 的 I 标志置位，相应的外部引脚中断就使能了。MCU通用控制状态寄存器– MCUCSR的中断敏感电平控制1位 1/0 (ISC11与ISC10)决定中断是由上升沿、下降沿，还是 INT1 电平触发的。只要使能，即使 INT1 引脚被配置为输出，只要引脚电平发生了相应的变化，中断将产生。

- **Bit 6 – INT0: 外部中断请求 0 使能**

当 INT0 为 '1'，而且状态寄存器 SREG 的 I 标志置位，相应的外部引脚中断就使能了。MCU通用控制状态寄存器– MCUCSR的中断敏感电平控制0位 1/0 (ISC01与ISC00)决定中断是由上升沿、下降沿，还是 INT0 电平触发的。只要使能，即使 INT0 引脚被配置为输出，只要引脚电平发生了相应的变化，中断将产生。

- **Bit 5 – INT2: 外部中断请求 2 使能**

当 INT2 为 '1'，而且状态寄存器 SREG 的 I 标志置位，相应的外部引脚中断就使能了。MCU通用控制状态寄存器– MCUCSR的中断敏感电平控制2位 1/0 (ISC2与ISC2)决定中断是由上升沿、下降沿，还是 INT2 电平触发的。只要使能，即使 INT2 引脚被配置为输出，只要引脚电平发生了相应的变化，中断将产生。

通用中断标志寄存器 - GIFR

Bit	7	6	5	4	3	2	1	0	
	INTF1	INTF0	INTF2	–	–	–	–	–	GIFR
读 / 写	R/W	R/W	R/W	R	R	R	R	R	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – INTF1: 外部中断标志 1**

INT1 引脚电平发生跳变时触发中断请求，并置位相应的中断标志 INTF1。如果 SREG 的位 I 以及 GICR 寄存器相应的中断使能位 INT1 为 "1"，MCU 即跳转到相应的中断向量。进入中断服务程序之后该标志自动清零。此外，标志位也可以通过写入 "1" 来清零。

- **Bit 6 – INTF0: 外部中断标志 0**

INT0 引脚电平发生跳变时触发中断请求，并置位相应的中断标志 INTF0。如果 SREG 的位 I 以及 GICR 寄存器相应的中断使能位 INT0 为 "1"，MCU 即跳转到相应的中断向量。进入中断服务程序之后该标志自动清零。此外，标志位也可以通过写入 "1" 来清零。当 INT0 配置为电平中断时，该标志会被清零。

- **Bit 5 – INTF2: 外部中断标志 2**

INT2 引脚电平发生跳变时触发中断请求，并置位相应的中断标志 INTF2。如果 SREG 的位 I 以及 GICR 寄存器相应的中断使能位 INT2 为 "1"，MCU 即跳转到相应的中断向量。进入中断服务程序之后该标志自动清零。此外，标志位也可以通过写入 "1" 来清零。注意，当 INT2 中断禁用进入某些休眠模式时，该引脚的输入缓冲将禁用。这会导致 INTF2 标志设置信号的逻辑变化，详见 P51 " 数字输入使能和睡眠模式 " 。

具有 PWM 功能的 8 位定时器 / 计时器 0

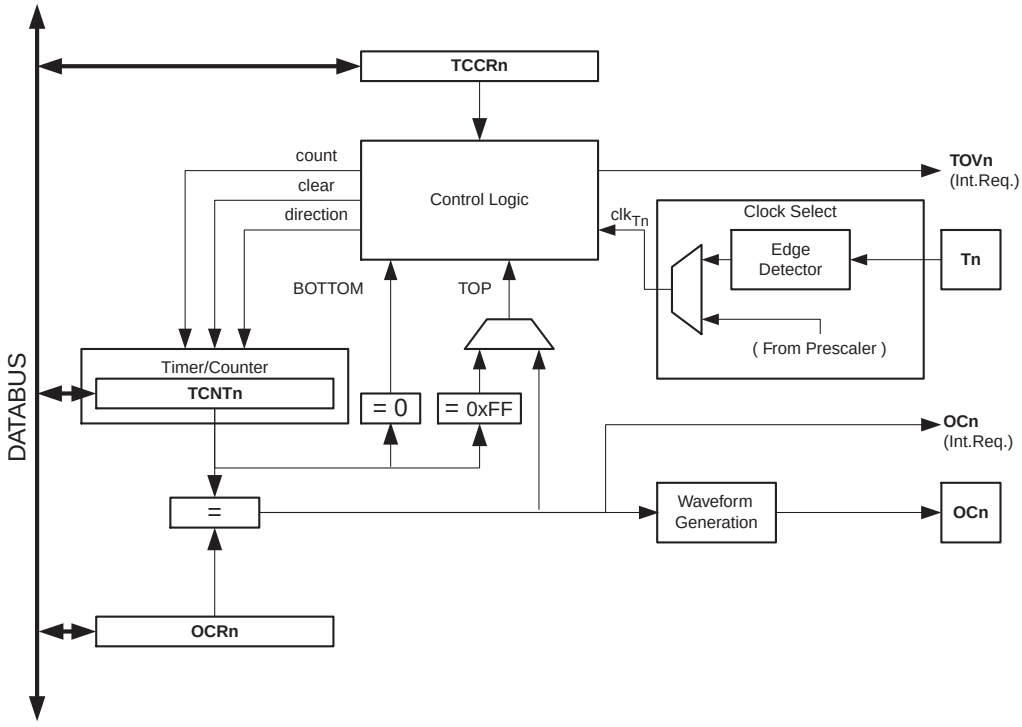
T/C0 是一个通用的单通道 8 位定时器 / 计数器模块。其主要特点如下：

- **单通道计数器**
- **比较匹配发生时清除定时器 (自动加载)**
- **无干扰脉冲，相位正确的 PWM**
- **频率发生器**
- **外部事件计数器**
- **10 位的时钟预分频器**
- **溢出和比较匹配中断源 (TOV0 和 OCF0)**

综述

Figure 27 为 8 位定时器 / 计数器的简化框图。实际引脚排列请参考 P2 “ATmega32 的引脚 ”。CPU 可以访问的 I/O 寄存器，包括位和引脚，以粗体显示。I/O 寄存器和位的位置列于 P76 “8 位定时器 / 计数器寄存器的说明 ”。

Figure 27. 8 位 T/C 方框图



寄存器

T/C(TCNT0) 和输出比较寄存器 (OCR0) 为 8 位寄存器。中断请求 (图中简写为 Int.Req.) 信号在定时器中断标志寄存器 TIFR 都有反映。所有中断都可以通过定时器中断屏蔽寄存器 TIMSK 单独进行屏蔽。由于 TIFR 和 TIMSK 寄存器是与其他定时器单元共享，因此图中没有给出。

T/C 可以通过预分频器由内部时钟源驱动，或者是通过 T0 引脚的外部时钟源来驱动。时钟选择逻辑模块控制使用哪一个时钟源与什么边沿来增加 (或降低)T/C 的数值。如果没有选择时钟源 T/C 就不工作。时钟选择模块的输出定义为定时器时钟 clk_{T0}。

双缓冲的输出比较寄存器 OCR0 一直与 T/C 的数值进行比较。比较的结果可用来产生 PWM 波，或在输出比较引脚 OC0 上产生变化频率的输出，如 P68 “输出比较单元 ” 说明的那样。比较匹配事件还将置位比较标志 OCF0。此标志可以用来产生输出比较中断请求。

定义

本文的许多寄存器及其各个位以通用的格式表示。小写的 “n” 取代了 T/C 的序号，在此即为 0。但是在写程序时要使用精确的格式，例如使用 TCNT0 来访问 T/C0 计数器值，等等。

Table 37 的定义适用于全文。

Table 37. 定义

BOTTOM	计数器计到 0x00 时即达到 BOTTOM。
MAX	计数器计到 0xFF (十进制的 255) 时即达到 MAX。
TOP	计数器计到计数序列的最大值时即达到 TOP。TOP 值可以为固定值 0xFF (MAX)，或是存储于寄存器 OCR0 里的数值，具体由工作模式确定

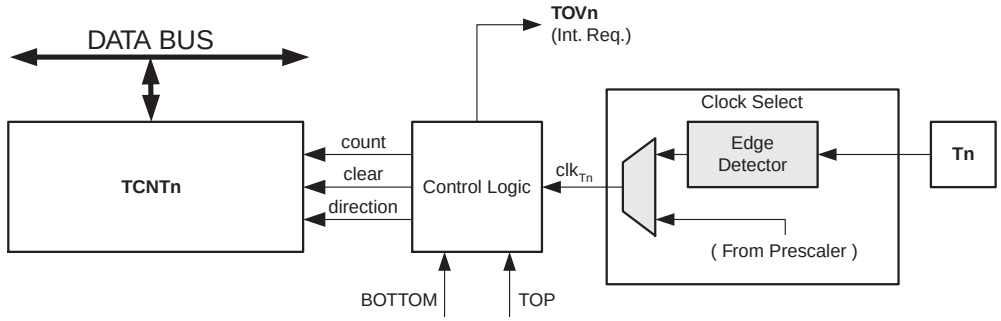
T/C 的时钟源

T/C 可以由内部同步时钟或外部异步时钟驱动。时钟源是由时钟选择逻辑决定的，而时钟选择逻辑是由位于 T/C 控制寄存器 TCCR0 的时钟选择位 CS02:0 控制的。P79 “T/C0 与 T/C1 的预分频器 ” 对时钟源与预分频有详尽的描述。

计数器单元

8位T/C的主要部分为可编程的双向计数单元。Figure 28 即为计数器和周边电路的框图。

Figure 28. 计数器单元方框图



信号说明 (内部信号) :

count 使 TCNT0 加 1 或减 1。

direction 选择加操作或减操作。

clear 清除 TCNT0 (将所有的位清零)。

clk_{Tn} T/C 的时钟，clk_{T0} 。

top 表示 TCNT0 已经达到了最大值。

bottom 表示 TCNT0 已经达到了最小值 (0)。

根据不同的工作模式，计数器针对每一个 clk_{T0} 实现清零、加一或减一操作。clk_{T0} 可以由内部时钟源或外部时钟源产生，具体由时钟选择位 CS02:0 确定。没有选择时钟源时 (CS02:0 = 0) 定时器即停止。但是不管有没有 clk_{T0} ,CPU 都可以访问 TCNT0。CPU 写操作比计数器其他操作 (如清零、加减操作) 的优先级高。

计数序列由 T/C 控制寄存器 (TCCR0) 的 WGM01 和 WGM00 决定。计数器计数行为与输出比较 OC0 的波形有紧密的关系。有关计数序列和波形产生的详细信息请参考 P70 “ 工作模式 ” 。

T/C溢出中断标志TOV0根据WGM01:0 设定的工作模式来设置。TOV0可以用于产生CPU中断。

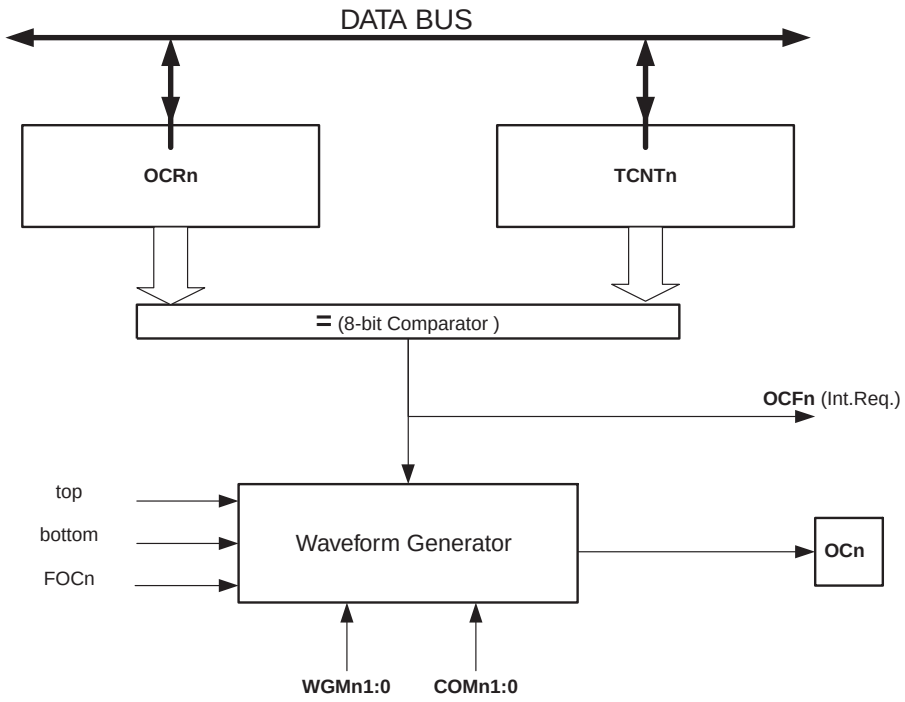
输出比较单元

8位比较器持续对TCNT0和输出比较寄存器OCR0进行比较。一旦TCNT0等于OCR0 ,比较器就给出匹配信号。在匹配发生的下一个定时器时钟周期输出比较标志 OCF0 置位。若此时 OCIE0 = 1 且 SREG 的全局中断标志 I 置位，CPU 将产生输出比较中断。执行中断服务程序时 OCF0 自动清零，或者通过软件写 ”1” 的方式来清零。根据由 WGM01:0 和

COM01:0 设定的不同的工作模式，波形发生器利用匹配信号产生不同的波形。同时，波形发生器还利用 max 和 bottom 信号来处理极值条件下的特殊情况 (P70 “ 工作模式 ”)。

Figure 29 为输出比较单元的方框图。

Figure 29. 输出比较单元方框图



使用 PWM 模式时 OCR0 寄存器为双缓冲寄存器；而在正常工作模式和匹配时清零模式双缓冲功能是禁止的。双缓冲可以将更新 OCR0 寄存器与 top 或 bottom 时刻同步起来，从而防止产生不对称的 PWM 脉冲，消除了干扰脉冲。

访问 OCR0 寄存器看起来很复杂，其实不然。使能双缓冲功能时，CPU 访问的是 OCR0 缓冲寄存器；禁止双缓冲功能时 CPU 访问的则是 OCR0 本身。

强制输出比较

工作于非 PWM 模式时，可以通过对强制输出比较位 FOC0 写 ”1” 的方式来产生比较匹配。强制比较匹配不会置位 OCF0 标志，也不会重载 / 清零定时器，但是 OC0 引脚将被更新，好象真的发生了比较匹配一样 (COM01:0 决定 OC0A 是置位、清零，还是 "0“-”1” 交替变化)。

写 TCNT0 操作将阻止比较匹配

CPU 对 TCNT0 寄存器的写操作会在下一个定时器时钟周期阻止比较匹配的发生 ,即使此时定时器已经停止了。这个特性可以用来将 OCR0 初始化为与 TCNT0 相同的数值而不触发中断。

使用输出比较单元

由于在任意模式下写 TCNT0 都将在下一个定时器时钟周期里阻止比较匹配，在使用输出比较时改变 TCNT0 就会有风险，不论 T/C 此时是否在运行与否。如果写入的 TCNT0 的数值等于 OCR0，比较匹配就被丢失了，造成不正确的波形发生结果。类似地，在计数器进行降序计数时不要对 TCNT0 写入等于 BOTTOM 的数据。

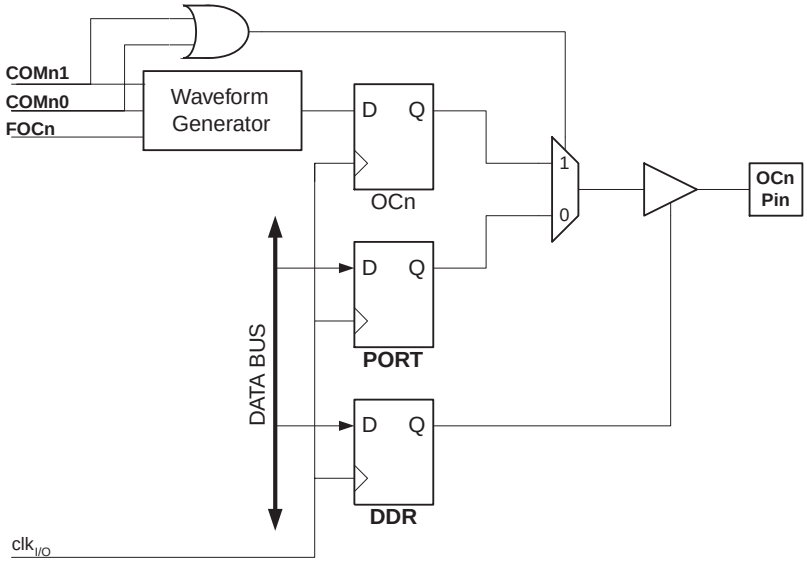
OC0 的设置应该在设置数据方向寄存器之前完成。最简单的设置 OC0 的方法是在普通模式下利用强制输出比较 FOC0。即使在改变波形发生模式时 OC0 寄存器也会一直保持它的数值。

注意 COM01:0 和比较数据都不是双缓冲的。 COM01:0 的改变将立即生效。

比较匹配输出单元

比较匹配模式控制位 COM01:0 具有双重功能。波形发生器利用 COM01:0 来确定下一次比较匹配发生时的输出比较状态 (OC0) ； COM01:0 还控制 OC0 引脚输出信号的来源。Figure 30为受COM01:0设置影响的简化逻辑框图。I/O寄存器、I/O位和I/O引脚以粗体表示。图中只给出了受 COM01:0 影响的通用 I/O 端口控制寄存器 (DDR 和 PORT)。谈及 OC0 状态时指的是内部 OC0 寄存器，而不是 OC0 引脚。系统复位时 OC0 寄存器清零。

Figure 30. 比较匹配输出单元原理图



如果 COM01:0 不全为零，通用 I/O 口功能将被波形发生器的输出比较功能取代。但 OC0 引脚为输入还是输出仍然由数据方向寄存器 DDR 控制。在使用 OC0 功能之前首先要通过数据方向寄存器的 DDR_OC0 位将此引脚设置为输出。端口功能与波形发生器的工作模式无关。

输出比较逻辑的设计允许 OC0 状态在输出之前首先进行初始化。要注意某些 COM01:0 设置保留给了其他操作类型，详见 P76 “8 位定时器 / 计数器寄存器的说明 ” 。

比较输出模式和波形产生

波形发生器利用 COM01:0 的方法在普通模式、 CTC 模式和 PWM 模式下有所区别。对于所有的模式，设置 COM01:0 = 0 表明比较匹配发生时波形发生器不会操作 OC0 寄存器。非 PWM 模式的比较输出请参见 P76 Table 39 ；快速 PWM 的比较输出示于 P77 Table 40 ；相位修正 PWM 的比较输出在 P77 Table 41 有描述。

改变 COM01:0 将影响写入数据后的第一次比较匹配。对于非 PWM 模式，可以通过使用 FOC0 来立即产生效果。

工作模式

工作模式 - T/C 和输出比较引脚的行为 - 由波形发生模式 (WGM01:0) 及比较输出模式 (COM01:0) 的控制位决定。比较输出模式对计数序列没有影响，而波形产生模式对计数序列则有影响。 COM01:0 控制 PWM 输出是否为反极性。非 PWM 模式时 COM01:0 控制输出是否应该在比较匹配发生时置位、清零，或是电平取反 (P70 “ 比较匹配输出单元 ”)。

具体的时序信息请参考 P74 “T/C 时序图 ” 之 Figure 34、Figure 35、Figure 36 与 Figure 37。

普通模式

普通模式 (WGM01:0 = 0) 为最简单的工作模式。在此模式下计数器不停地累加。计到 8 比特的最大值后 (TOP = 0xFF)，由于数值溢出计数器简单地返回到最小值 0x00 重新开始。在 TCNT0 为零的同一个定时器时钟里 T/C 溢出标志 TOV0 置位。此时 TOV0 有点象第 9 位，只是只能置位，不会清零。但由于定时器中断服务程序能够自动清零 TOV0，因

此可以通过软件提高定时器的分辨率。在普通模式下没有什么需要特殊考虑的，用户可以随时写入新的计数器数值。

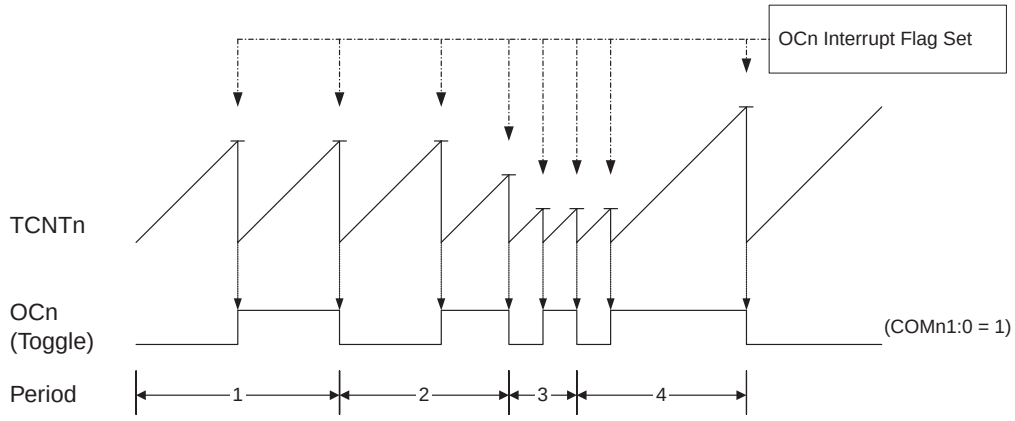
输出比较单元可以用来产生中断。但是不推荐在普通模式下利用输出比较来产生波形，因为这会占用太多的 CPU 时间。

CTC(比较匹配时清零定时器) 模式

在 CTC 模式 (WGM01:0 = 2) 下 OCR0 寄存器用于调节计数器的分辨率。当计数器的数值 TCNT0 等于 OCR0 时计数器清零。OCR0 定义了计数器的 TOP 值，亦即计数器的分辨率。这个模式使得用户可以很容易地控制比较匹配输出的频率，也简化了外部事件计数的操作。

CTC模式的时序图为 Figure 31。计数器数值TCNT0一直累加到TCNT0与OCR0匹配，然后 TCNT0 清零。

Figure 31. CTC 模式的时序图



利用 OCF0 标志可以在计数器数值达到 TOP 时产生中断。在中断服务程序里可以更新 TOP的数值。由于CTC模式没有双缓冲功能，在计数器以无预分频器或很低的预分频器工作的时候将 TOP 更改为接近 BOTTOM 的数值时要小心。如果写入的 OCR0 数值小于当前 TCNT0 的数值，计数器将丢失一次比较匹配。在下一次比较匹配发生之前，计数器不得不先计数到最大值 0xFF，然后再从 0x00 开始计数到 OCF0。

为了在 CTC 模式下得到波形输出，可以设置 OC0 在每次比较匹配发生时改变逻辑电平。这可以通过设置 COM01:0 = 1 来完成。在期望获得 OC0 输出之前，首先要将其端口设置为输出。波形发生器能够产生的最大频率为 $f_{OC0} = f_{clk\_I/O}/2$ (OCR0 = 0x00)。频率由如下公式确定：

$$f_{OCn} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot (1 + OCRn)}$$

变量 N 代表预分频因子 (1、 8、 64、 256 或 1024)。

在普通模式下， TOV0 标志的置位发生在计数器从 MAX 变为 0x00 的定时器时钟周期。

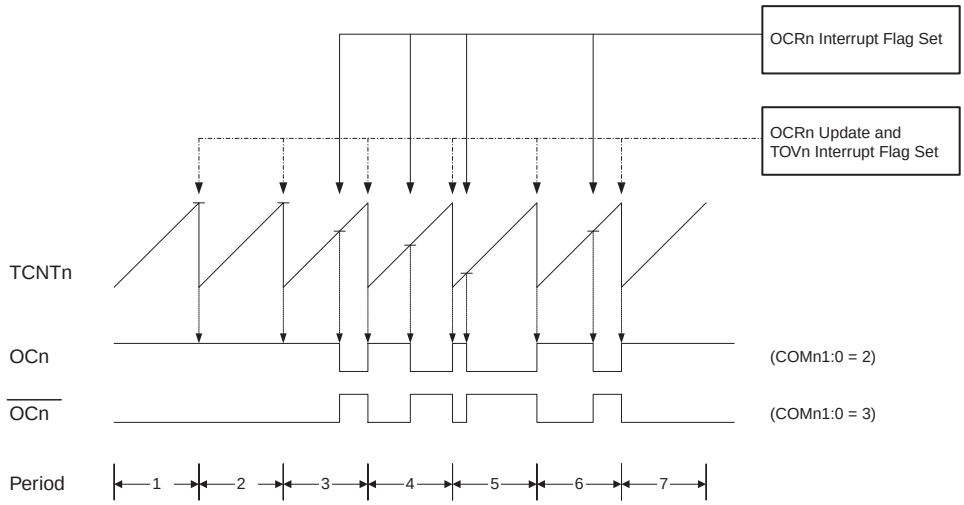
快速 PWM 模式

快速 PWM 模式 (WGM01:0 = 3) 可用来产生高频的 PWM 波形。快速 PWM 模式与其他 PWM模式的不同之处是其单斜坡工作方式。计数器从BOTTOM计到MAX，然后立即回到 BOTTOM 重新开始。对于普通的比较输出模式，输出比较引脚OC0 在TCNT0 与OCR0 匹配时清零，在 BOTTOM 时置位；对于反向比较输出模式， OC0 的动作正好相反。由于使用了单斜坡模式，快速 PWM 模式的工作频率比使用双斜坡的相位修正 PWM 模式高一倍。此高频操作特性使得快速 PWM 模式十分适合于功率调节，整流和 DAC 应用。高频可以减小外部元器件 (电感，电容) 的物理尺寸，从而降低系统成本。

工作于快速 PWM 模式时，计数器的数值一直增加到 MAX，然后在后面的一个时钟周期清零。具体的时序图为 Figure 32。图中柱状的 TCNT0 表示这是单边斜坡操作。方框图同

时包含了普通的 PWM 输出以及反向 PWM 输出。 TCNT0 斜坡上的短水平线表示 OCR0 和 TCNT0 的比较匹配。

Figure 32. 快速 PWM 模式时序图



计时器数值达到 MAX 时 T/C 溢出标志 TOV0 置位。如果中断使能，在中断服务程序可以更新比较值。

工作于快速 PWM 模式时，比较单元可以在 OC0 引脚上输出 PWM 波形。设置 COM01:0 为 2 可以产生普通的 PWM 信号；为 3 则可以产生反向 PWM 波形 (参见 P77 Table 40)。要想在引脚上得到输出信号还必须将OC0的数据方向设置为输出。产生PWM波形的机理是 OC0 寄存器在 OCR0 与 TCNT0 匹配时置位 (或清零) ，以及在计数器清零 (从 MAX 变为 BOTTOM) 的那一个定时器时钟周期清零 (或置位)。

输出的 PWM 频率可以通过如下公式计算得到：

$$f_{OCnPWM} = \frac{f_{clk\_I/O}}{N \cdot 256}$$

变量 N 代表分频因子 (1、8、64、256 或 1024)。

OCR0 寄存器为极限值时表示快速PWM模式的一些特殊情况。若 OCR0 等于 BOTTOM，输出为出现在第 MAX+1 个定时器时钟周期的窄脉冲； OCR0 为 MAX 时，根据 COM01:0 的设定，输出恒为高电平或低电平。

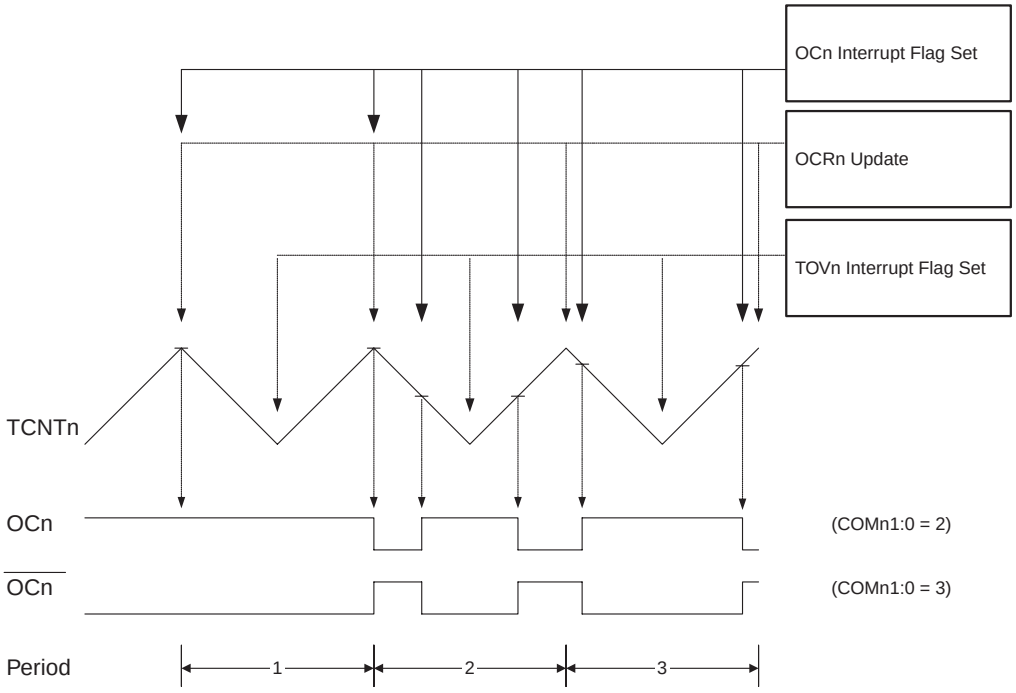
通过设定 OC0 在比较匹配时进行逻辑电平取反 (COM01:0 = 1)，可以得到占空比为 50% 的周期信号。 OCR0 为 0 时信号有最高频率 $f_{OC0} = f_{clk\_I/O}/2$。这个特性类似于 CTC 模式下的 OC0 取反操作，不同之处在于快速 PWM 模式具有双缓冲。

相位修正 PWM 模式

相位修正 PWM 模式 (WGM01:0 = 1) 为用户提供了一个获得高精度相位修正 PWM 波形的方法。此模式基于双斜坡操作。计时器重复地从 BOTTOM 计到 MAX，然后又从 MAX 倒退回到BOTTOM。在一般的比较输出模式下，当计时器往MAX计数时若发生了TCNT0与OCR0的匹配，OC0将清零为低电平；而在计时器往BOTTOM计数时若发生了TCNT0与 OCR0 的匹配， OC0 将置位为高电平。工作于反向输出比较时则正好相反。与单斜坡操作相比，双斜坡操作可获得的最大频率要小。但由于其对称的特性，十分适合于电机控制。

相位修正 PWM 模式的 PWM 精度固定为 8 比特。计时器不断地累加直到 MAX，然后开始减计数。在一个定时器时钟周期里 TCNT0 的值等于 MAX。时序图可参见 Figure 33。图中 TCNT0 的数值用柱状图表示，以说明双斜坡操作。本图同时说明了普通 PWM 的输出和反向 PWM 的输出。 TCNT0 斜坡上的小横条表示 OCR0 与 TCNT0 的比较匹配。

Figure 33. 相位修正 PWM 模式的时序图



当计时器达到 BOTTOM 时 T/C 溢出标志位 TOV0 置位。此标志位可用来产生中断。

工作于相位修正 PWM 模式时，比较单元可以在 OC0 引脚产生 PWM 波形：将 COM01:0 设置为 2 产生普通相位的 PWM，设置 COM01:0 为 3 产生反向 PWM 信号（参见 P77 Table 41）。要想在引脚上得到输出信号还必须将 OC0 的数据方向设置为输出。OCR0 和 TCNT0 比较匹配发生时 OC0 寄存器将产生相应的清零或置位操作，从而产生 PWM 波形。工作于相位修正模式时 PWM 频率可由下式公式获得：

$$f_{OCnPCPWM} = \frac{f_{clk\_I/O}}{N \cdot 510}$$

变量 N 表示预分频因子 (1、8、64、 256 或 1024)。

OCR0 寄存器处于极值代表了相位修正 PWM 模式的一些特殊情况。在普通 PWM 模式下，若 OCR0 等于 BOTTOM，输出一直保持为低电平；若 OCR0 等于 MAX，则输出保持为高电平。反向 PWM 模式则正好相反。

在 Figure 33 的第 2 个周期，虽然没有发生比较匹配， OCn 也出现了一个从高到低的跳变。其目的是保证波形在 BOTTOM 两侧的对称。没有比较匹配时有两种情况会出现跳变：

- 如 Figure 33 所示，OCR0A 的值从 MAX 改变为其他数据。当 OCR0A 值为 MAX 时，引脚 OCn 的输出应该与前面降序记数比较匹配的结果相同。为保证波形在 BOTTOM 两侧的对称，当 T/C 的数值为 MAX 时，引脚 OCn 的输出又必须符合后面升序记数比较匹配的结果。
- 定时器从一个比 OCR0A 高的值开始记数，并因而丢失了一次比较匹配。系统因此引入发生 OCn 却仍然有跳变的现象。

T/C 时序图

T/C 是同步电路，因此其时钟 clk_{T0} 可以表示为时钟使能信号，如下图所示。图中还说明了中断标志设置的时间。 Figure 34 给出了基本的 T/C 工作时序，以及除了相位修正 PWM 模式之外其他模式接近 MAX 时的记数序列。

Figure 34. T/C 时序图，无预分频器

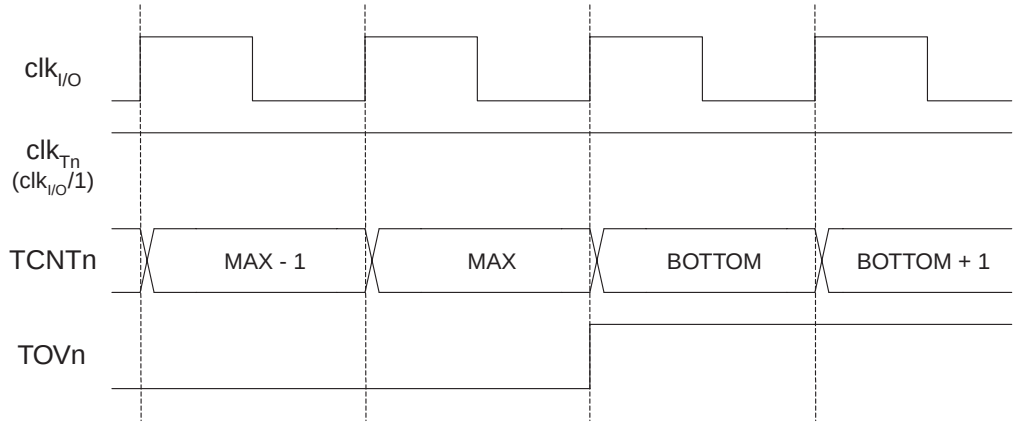


Figure 35 所示为相同的工作时序，但有预分频。

Figure 35. T/C 时序图，预分频器为 $f_{clk\_I/O}/8$

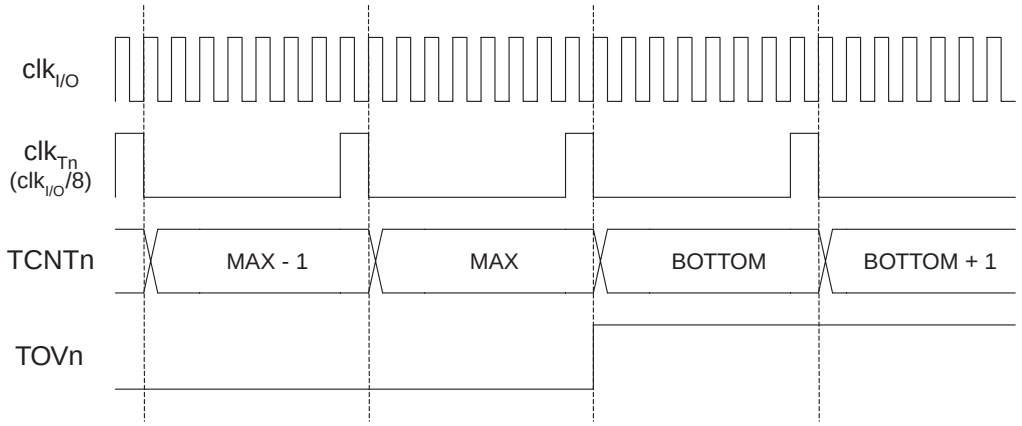


Figure 36 给出了各种模式下 (除了 CTC 模式)OCF0 的置位情况。

Figure 36. T/C 时序图， OCF0 置位，预分频器为 $f_{clk\_I/O}/8$

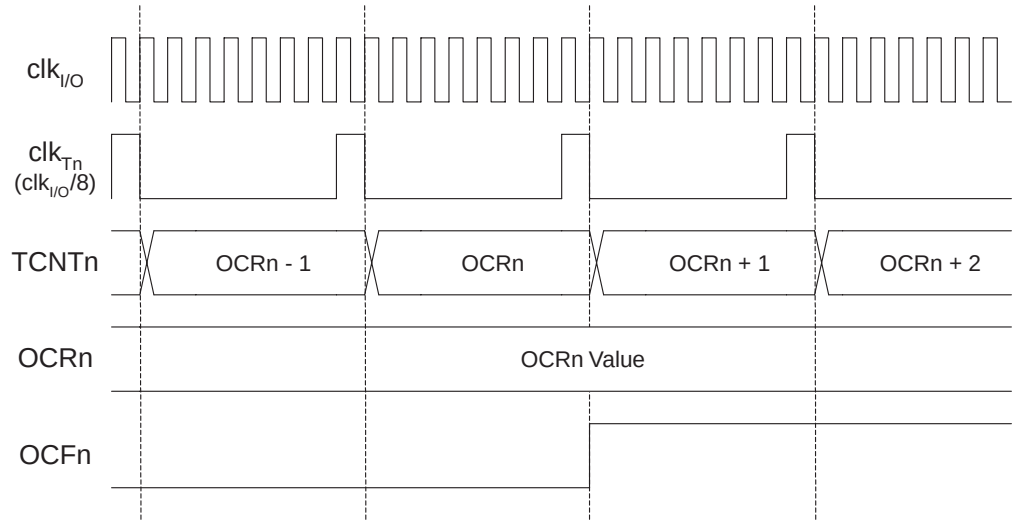
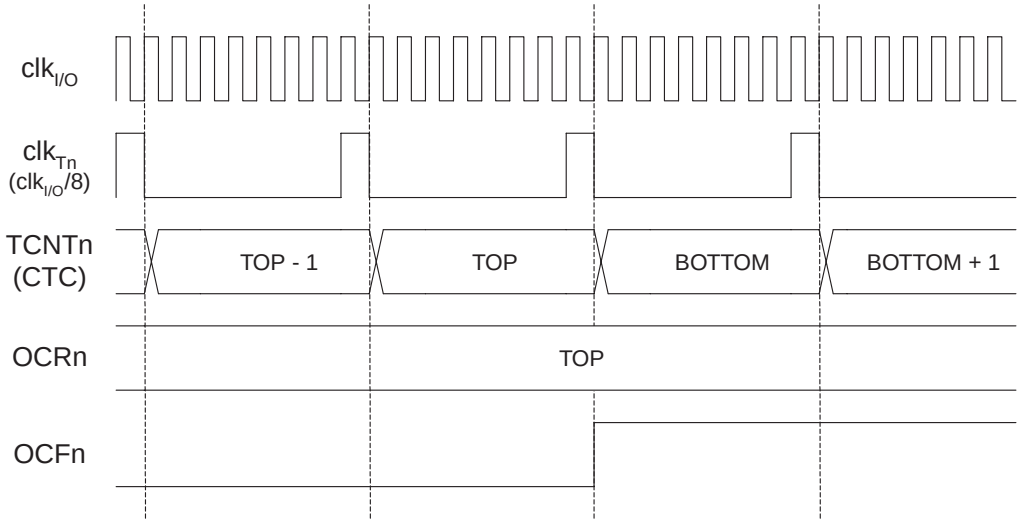


Figure 37 给出了 CTC 模式下 OCF0 置位和 TCNT0 清除的情况。

Figure 37. T/C 时序图， CTC 模式，预分频器为 $f_{clk\_I/O}/8$



8 位定时器 / 计数器寄存器的说明

T/C 控制寄存器 - TCCR0

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
读 / 写	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC0: 强制输出比较**

FOC0 仅在 WGM00 指明非 PWM 模式时才有效。但是，为了保证与未来器件的兼容性，在使用 PWM 时，写 TCCR0 要对其清零。对其写 1 后，波形发生器将立即进行比较操作。比较匹配输出引脚 OC0 将按照 COM01:0 的设置输出相应的电平。要注意 FOC0 类似一个锁存信号，真正对强制输出比较起作用的是 COM01:0 的设置。

FOC0 不会引发任何中断，也不会在利用 OCR0 作为 TOP 的 CTC 模式下对定时器进行清零的操作。

读 FOC0 的返回值永远为 0。

- **Bit 6, 3 – WGM01:0: 波形产生模式**

这几位控制计数器的计数序列，计数器的最大值 TOP，以及产生何种波形。 T/C 支持的模式有：普通模式，比较匹配发生时清除计数器模式 (CTC)，以及两种 PWM 模式，详见 Table 38 与 P70 “ 工作模式 ” 。

Table 38. 波形产生模式的位定义 (1)

模式	WGM01 (CTC0)	WGM00 (PWM0)	T/C 的工作模式	TOP	OCR0 的更新时间	TOV0 的置位时刻
0	0	0	普通	0xFF	立即更新	MAX
1	0	1	PWM，相位修正	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	立即更新	MAX
3	1	1	快速 PWM	0xFF	TOP	MAX

Note: 1. 位定义 CTC0 和 PWM0 已经不再使用了，要使用 WGM01:0。但是功能和位置与以前版本兼容。

- **Bit 5:4 – COM01:0: 比较匹配输出模式**

这两位决定了比较匹配发生时输出引脚 OC0 的电平。如果 COM01:0 中的一位或全部都置位， OC0 以比较匹配输出的方式进行工作。同时其方向控制寄存器位要设置为 1 以使能输出驱动器。

当 OC0 连接到物理引脚上时， COM01:0 的功能依赖于 WGM01:0 的设置。 Table 39 给出了当 WGM01:0 设置为普通模式或 CTC 模式时 COM01:0 的功能。 .

Table 39. 比较输出模式，非 PWM 模式

COM01	COM00	说明
0	0	正常的端口操作，不与 OC0 相连接
0	1	比较匹配发生时 OC0 取反
1	0	比较匹配发生时 OC0 清零
1	1	比较匹配发生时 OC0 置位

Table 40 给出了当 WGM01:0 设置为快速 PWM 模式时 COM01:0 的功能。

Table 40. 比较输出模式，快速 PWM 模式 (1)

COM01	COM00	说明
0	0	正常的端口操作，不与 OC0 相连接
0	1	保留
1	0	比较匹配发生时 OC0A 清零，计数到 TOP 时 OC0 置位
1	1	比较匹配发生时 OC0A 置位，计数到 TOP 时 OC0 清零

Note: 1. 一个特殊情况是 OCR0 等于 TOP，且 COM01 置位。此时比较匹配将被忽略，而计数到 TOP 时 OC0 的动作继续有效。详细信息请参见 P71 “ 快速 PWM 模式 ” 。

Table 41 给出了当 WGM01:0 设置为相位修正 PWM 模式时 COM01:0 的功能。

Table 41. 比较输出模式，相位修正 PWM 模式 (1)

COM01	COM00	说明
0	0	正常的端口操作，不与 OC0 相连接
0	1	保留
1	0	在升序计数时发生比较匹配将清零 OC0 ；降序计数时发生比较匹配将置位 OC0
1	1	在升序计数时发生比较匹配将置位 OC0 ；降序计数时发生比较匹配将清零 OC0

Note: 1. 一个特殊情况是 OCR0 等于 TOP，且 COM01 置位。此时比较匹配将被忽略，而计数到 TOP 时 OC0 的动作继续有效。详细信息请参见 P72 “ 相位修正 PWM 模式 ” 。

- **Bit 2:0 – CS02:0: 时钟选择**

用于选择 T/C 的时钟源。

Table 42. 时钟选择位说明

CS02	CS01	CS00	说明
0	0	0	无时钟， T/C 不工作
0	0	1	$clk_{I/O}$/1 (没有预分频)
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	时钟由 T0 引脚输入，下降沿触发
1	1	1	时钟由 T0 引脚输入，上升沿触发

如果 T/C0 使用外部时钟，即使 T0 被配置为输出，其上的电平变化仍然会驱动记数器。利用这一特性可通过软件控制记数。

T/C 寄存器 - TCNT0

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

通过 T/C 寄存器可以直接对计数器的 8 位数据进行读写访问。对 TCNT0 寄存器的写访问将在下一个时钟阻止比较匹配。在计数器运行的过程中修改 TCNT0 的数值有可能丢失一次 TCNT0 和 OCR0 的比较匹配。

输出比较寄存器 - OCR0

Bit	7	6	5	4	3	2	1	0	
	OCR0[7:0]								OCR0
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

输出比较寄存器包含一个 8 位的数据，不间断地与计数器数值 TCNT0 进行比较。匹配事件可以用来产生输出比较中断，或者用来在 OC0 引脚上产生波形。

T/C 中断屏蔽寄存器 - TIMSK

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 1 – OCIE0: T/C0 输出比较匹配中断使能**

当 OCIE0 和状态寄存器的全局中断使能位 I 都为 ”1” 时 ,T/C0 的输出比较匹配中断使能。当 T/C0 的比较匹配发生，即 TIFR 中的 OCF0 置位时，中断服务程序得以执行。

- **Bit 0 – TOIE0: T/C0 溢出中断使能**

当 TOIE0 和状态寄存器的全局中断使能位 I 都为 ”1” 时，T/C0 的溢出中断使能。当 T/C0 发生溢出，即 TIFR 中的 TOV0 位置位时，中断服务程序得以执行。

T/C 中断标志寄存器 - TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 1 – OCF0: 输出比较标志 0**

当 T/C0 与 OCR0(输出比较寄存器 0) 的值匹配时，OCF0 置位。此位在中断服务程序里硬件清零，也可以对其写 1 来清零。当 SREG 中的位 I、 OCIE0(T/C0 比较匹配中断使能) 和 OCF0 都置位时，中断服务程序得到执行。

- **Bit 0 – TOV0: T/C0 溢出标志**

当 T/C0 溢出时， TOV0 置位。执行相应的中断服务程序时此位硬件清零。此外， TOV0 也可以通过写 1 来清零。当 SREG 中的位 I、 TOIE0(T/C0 溢出中断使能) 和 TOV0 都置位时，中断服务程序得到执行。在相位修正 PWM 模式中，当 T/C0 在 0x00 改变记数方向时， TOV0 置位。

T/C0 与 T/C1 的预分频器

T/C1 与 T/C0 共用一个预分频模块，但它们可以有不同的分频设置。下述内容适用于 T/C1 与 T/C0。

内部时钟源

当 CSn2:0 = 1 时，系统内部时钟直接作为 T/C 的时钟源，这也是 T/C 最高频率的时钟源 $f_{CLK\_I/O}$，与系统时钟频率相同。预分频器可以输出 4 个不同的时钟信号 $f_{CLK\_I/O}/8$、$f_{CLK\_I/O}/64$、$f_{CLK\_I/O}/256$ 或 $f_{CLK\_I/O}/1024$。

分频器复位

预分频器是独立运行的。也就是说，其操作独立于 T/C 的时钟选择逻辑，且它由 T/C1 与 T/C0 共享。由于预分频器不受 T/C 时钟选择的影响，预分频器的状态需要包含预分频时钟被用到何处这样的信息。一个典型的例子发生在定时器使能并由预分频器驱动 (6 > CSn2:0 > 1)的时候：从计时器使能到第一次开始计数可能花费 1 到N+1个系统时钟周期，其中 N 等于预分频因子 (8、 64、 256 或 1024)。

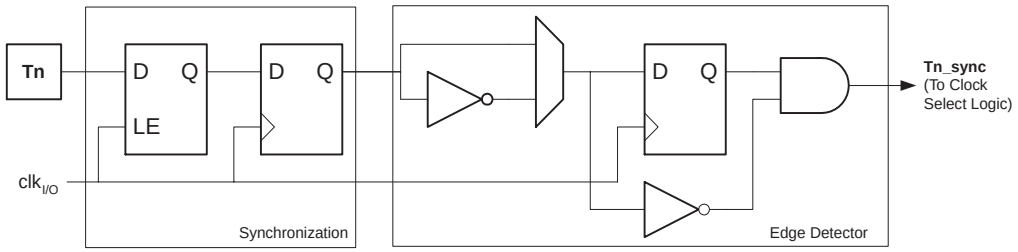
通过复位预分频器来同步 T/C 与程序运行是可能的。但是必须注意另一个 T/C 是否也在使用这一预分频器，因为预分频器复位将会影响所有与其连接的 T/C。

外部时钟源

由 T1/T0 引脚提供的外部时钟源可以用作 T/C 时钟 clk_{T1}/clk_{T0}。引脚同步逻辑在每个系统时钟周期对引脚 T1/T0 进行采样。然后将同步（采样）信号送到边沿检测器。 Figure 38 给出了 T1/T0 同步采样与边沿检测逻辑的功能等效方框图。寄存器由内部系统时钟 $clk_{I/O}$ 的上跳沿驱动。当内部时钟为高时，锁存器可以看作时透明的。

CSn2:0 = 7 时边沿检测器检测到一个正跳变产生一个 clk_{T1} 脉冲； CSn2:0 = 6 时一个负跳变就产生一个 clk_{T0} 脉冲。

Figure 38. T1/T0 引脚采样



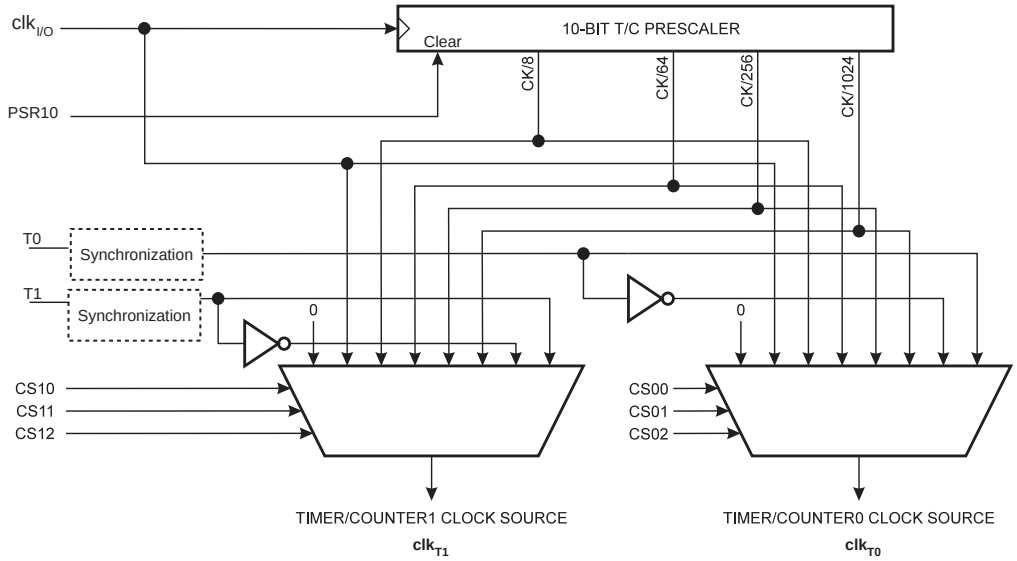
由于引脚上同步与边沿监测电路的存在，引脚 T1/T0 上的电平变化需要延时 2.5 到 3.5 个系统时钟周期才能使计数器进行更新。

禁止或使能时钟输入必须在 T1/T0 保持稳定至少一个系统时钟周期后才能进行，否则有产生错误 T/C 时钟脉冲的危险。

为保证正确的采样，外部时钟脉冲宽度必须大于一个系统时钟周期。在占空比为 50% 时外部时钟频率必须小于系统时钟频率的一半 ($f_{ExtClk} < f_{clk\_I/O}/2$)。由于边沿检测器使用的是采样这一方法，它能检测到的外部时钟最多是其采样频率的一半 (Nyquist 采样定理)。然而，由于振荡器（晶体、谐振器与电容）本身误差带来的系统时钟频率及占空比的差异，建议外部时钟的最高频率不要大于 $f_{clk\_I/O}/2.5$。

外部时钟源不送入预分频器。

Figure 39. T/C0 与 T/C1 预分频器 (1)



Note: 1. 输入引脚 (T1/T0) 的同步逻辑见 Figure 38。

特殊功能 IO 寄存器 - SFIOR

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
读 / 写	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 0 – PSR10: T/C1 与 T/C0 预分频器复位**

置位时 T/C1 与 T/C0 的预分频器复位。操作完成后这一位由硬件自动清零。写入零时不会引发任何动作。 T/C1 与 T/C0 共用同一预分频器，且预分频器复位对两个定时器均有影响。该位总是读为 0。

16 位定时器 / 计数器 1

16 位的 T/C 可以实现精确的程序定时 (事件管理)、波形产生和信号测量。其主要特点如下

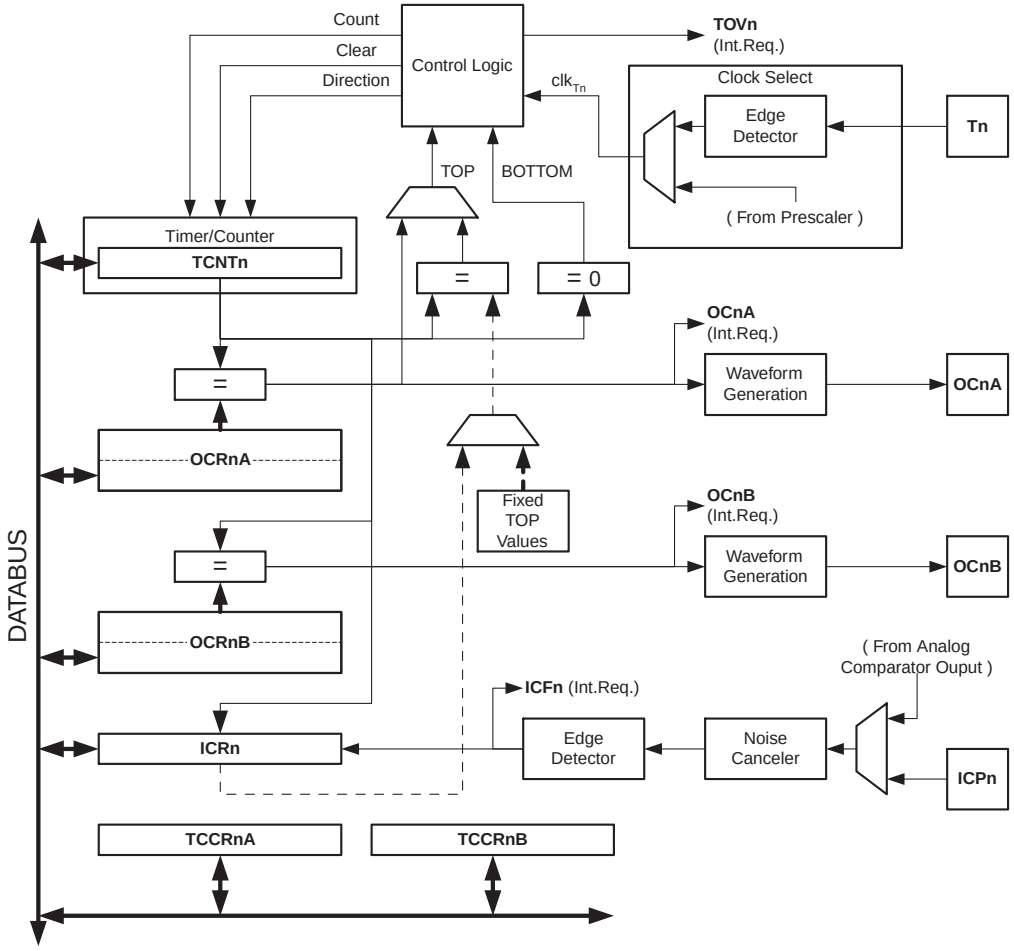
- **真正的 16 位设计 (即允许 16 位的 PWM)**
- **2 个独立的输出比较单元**
- **双缓冲的输出比较寄存器**
- **一个输入捕捉单元**
- **输入捕捉噪声抑制器**
- **比较匹配发生时清除寄存器 (自动重载)**
- **无干扰脉冲，相位正确的 PWM**
- **可变的 PWM 周期**
- **频率发生器**
- **外部事件计数器**
- **4 个独立的中断源 (TOV1、OCF1A、 OCF1B 与 ICF1)**

综述

本节大多数的寄存器和位定义以通用的方式表示。小写 “n” 表示 T/C 序号，小写 “x” 表示输出比较通道号。但是在写程序时要用完整的、精确的名称。如用 TCNT1 表示访问 T/C1 计数器值等。

16 位 T/C 的简化框图示于 Figure 40。I/O 引脚的实际位置请参见 P2 Figure 1 。CPU 可访问的 I/O 寄存器 , 包括 I/O 位和 I/O 引脚以粗体表示。器件具体 I/O 寄存器与位定位见 P100 “16 位定时器 / 计数器寄存器的说明 ” 。

Figure 40. 16 位 T/C 框图 (1)



Note: 1. 请参考 P2 Figure 1 ， P55 Table 25 和 P60 Table 31 以获得 T/C1 的引脚定义。

寄存器

定时器 / 计数器 TCNT1、输出比较寄存器 OCR1A/B 与输入捕捉寄存器 ICR1 均为 16 位寄存器。访问 16 位寄存器必须通过特殊的步骤，详见 P84 “ 访问 16 位寄存器 ” 。T/C 控制寄存器 TCCR1A/B 为 8 位寄存器，没有 CPU 访问的限制。中断请求 (图中简写为 Int.Req.) 信号在中断标志寄存器 TIFR1 都有反映。所有中断都可以由中断屏蔽寄存器 TIMSK1 单独控制。图中未给出 TIFR1 与 TIMSK1。

T/C可由内部时钟通过预分频器或通过由T1引脚输入的外部时钟驱动。引发T/C数值增加(或减少) 的时钟源及其有效沿由时钟选择逻辑模块控制。没有选择时钟源时 T/C 处于停止状态。时钟选择逻辑模块的输出称为 clk_{T1}。

双缓冲输出比较寄存器 OCR1A/B 一直与 T/C 的值做比较。波形发生器用比较结果产生PWM或在输出比较引脚OC1A/B输出可变频率的信号。参见 P89 “输出比较单元” 。比较匹配结果还可置位比较匹配标志 OCF1A/B，用来产生输出比较中断请求。

当输入捕捉引脚 ICP1 或模拟比较器输入引脚 (见 P184 “ 模拟比较器 ”) 有输入捕捉事件产生 (边沿触发) 时，当时的 T/C 值被传输到输入捕捉寄存器保存起来。输入捕捉单元包括一个数字滤波单元 (噪声消除器) 以降低噪声干扰。

在某些操作模式下， TOP 值或 T/C 的最大值可由 OCR1A 寄存器、 ICR1 寄存器，或一些固定数据来定义。在 PWM 模式下用 OCR1A 作为 TOP 值时， OCR1A 寄存器不能用作 PWM 输出。但此时 OCR1A 是双向缓冲的， TOP 值可在运行过程中得到改变。当需要一个固定的 TOP 值时可以使用 ICR1 寄存器 ,从而释放 OCR1A 来用作 PWM 的输出。

定义

以下定义适用于本节：

Table 43. 定义

BOTTOM	计数器计到 0x0000 时即达到 BOTTOM
MAX	计数器计到 0xFFFF (十进制的 65535) 时即达到 MAX
TOP	计数器计到计数序列的最大值时即达到 TOP。 TOP 值可以为固定值 0x00FF、0x01FF 或 0x03FF ,或是存储于寄存器 OCR1A 或ICR1 里的数值 ,具体有赖于工作模式

兼容性

16位T/C是从以前版本的16位AVRT/C改进和升级得来的。它在如下方面与以前的版本完全兼容：

- 包括定时器中断寄存器在内的所有 16 位 T/C 相关的 I/O 寄存器的地址
- 包括定时器中断寄存器在内的所有 16 位 T/C 相关的寄存器位定位
- 中断向量

下列控制位名称已改，但具有相同的功能与寄存器单元：

- PWM10 改为 WGM10
- PWM11 改为 WGM11
- CTC1 改为 WGM12

16 位 T/C 控制寄存器中添加了下列位：

- TCCR1A 中加入 FOC1A 与 FOC1B
- TCCR1B 中加入 WGM13

16 位 T/C 的一些改进在某些特殊情况下将影响兼容性。

访问 16 位寄存器

TCNT1、OCR1A/B与ICR1是AVR CPU 通过8位数据总线可以访问的16位寄存器。读写16位寄存器需要两次操作。每个16位计时器都有一个8位临时寄存器用来存放其高8位数据。每个 16 位定时器所属的 16 位寄存器共用相同的临时寄存器。访问低字节会触发 16 位读或写操作。当 CPU 写入数据到 16 位寄存器的低字节时，写入的 8 位数据与存放在临时寄存器中的高 8 位数据组成一个 16 位数据，同步写入到 16 位寄存器中。当 CPU 读取 16 位寄存器的低字节时，高字节内容在读低字节操作的同时被放置于临时辅助寄存器中。

并非所有的 16 位访问都涉及临时寄存器。对 OCR1A/B 寄存器的读操作就不涉及临时寄存器。

写 16 位寄存器时，应先写入该寄存器的高位字节。而读 16 位寄存器时应先读取该寄存器的低位字节。

下面的例程说明了如何访问 16 位定时器寄存器。前提是假设不会发生更新临时寄存器内容的中断。同样的原则也适用于对 OCR1A/B 与 ICR1 寄存器的访问。使用 “C” 语言时，编译器会自动处理 16 位操作。

汇编代码例程 (1)

```
...
; 设置 TCNT1 为 0x01FF
ldi r17,0x01
ldi r16,0xFF
out TCNT1H,r17
out TCNT1L,r16
; 将 TCNT1 读入 r17:r16
in  r16,TCNT1L
in  r17,TCNT1H
...
```

C 代码例程 (1)

```
unsigned int i;
...
/* 设置 TCNT1 为 0x01FF */
TCNT1 = 0x1FF;
/* 将 TCNT1 读入 i */
i = TCNT1;
...
```

Note: 1. 本代码假定已经包含了合适的头文件。

汇编代码例程中 TCNT1 的返回值在 r17:r16 寄存器对中。

注意到 16 位寄存器的访问是一个基本操作是非常重要的。在对 16 位寄存器操作时，最好首先屏蔽中断响应，防止在主程序读写 16 位寄存器的两条指令之间发生这样的中断：它也访问同样的寄存器或其他的 16 位寄存器，从而更改了临时寄存器。如果这种情况发生，那么中断返回后临时寄存器中的内容已经改变，造成主程序对 16 位寄存器的读写错误。

下面的例程给出了读取 TCNT1 寄存器内容的基本操作。对 OCR1A/B 或 ICR1 的读操作可以使用相同的方法。

汇编代码例程 (1)

```
TIM16_ReadTCNT1:
  ; 保存全局中断标志
  in  r18,SREG
  ; 禁用中断
  cli
  ; 将 TCNT1 读入 r17:r16
  in  r16,TCNT1L
  in  r17,TCNT1H
  ; 恢复全局中断标志
  out SREG,r18
  ret
```

C 代码例程 (1)

```
unsigned int TIM16_ReadTCNT1( void )
{
  unsigned char sreg;
  unsigned int i;
  /* 保存全局中断标志 */
  sreg = SREG;
  /* 禁用中断 */
  _CLI();
  /* 将 TCNT1 读入 i */
  i = TCNT1;
  /* 恢复全局中断标志 */
  SREG = sreg;
  return i;
}
```

Note: 1. 本代码假定已经包含了合适的头文件。

汇编代码例程中 TCNT1 的返回值在 r17:r16 寄存器对中。

下面的例程给出了写 TCNT1 寄存器的基本操作。对 OCR1A/B 或 ICR1 的写操作可以使用相同的方法。

汇编代码例程 (1)

```
TIM16_WriteTCNT1:
  ; 保存全局中断标志
  in  r18,SREG
  ; 禁用中断
  cli
  ; 设置 TCNT1 到 r17:r16
  out TCNT1H,r17
  out TCNT1L,r16
  ; 恢复全局中断标志
  out SREG,r18
  ret
```

C 代码例程 (1)

```
void TIM16_WriteTCNT1 ( unsigned int i )
{
  unsigned char sreg;
  unsigned int i;
  /* 保存全局中断标志 */
  sreg = SREG;
  /* 禁用中断 */
  _CLI();
  /* 设置 TCNT1 到 i */
  TCNT1 = i;
  /* 恢复全局中断标志*/
  SREG = sreg;
}
```

Note: 1. 本代码假定已经包含了合适的头文件。

汇编代码例程中 r17:r16 寄存器对保存的是 TCNT1 的写入数据。

临时寄存器的重用

如果对不只一个 16 位寄存器写入数据而且所有的寄存器高字节相同，则只需写一次高字节。前面讲到的基本操作在这种情况下同样适用。

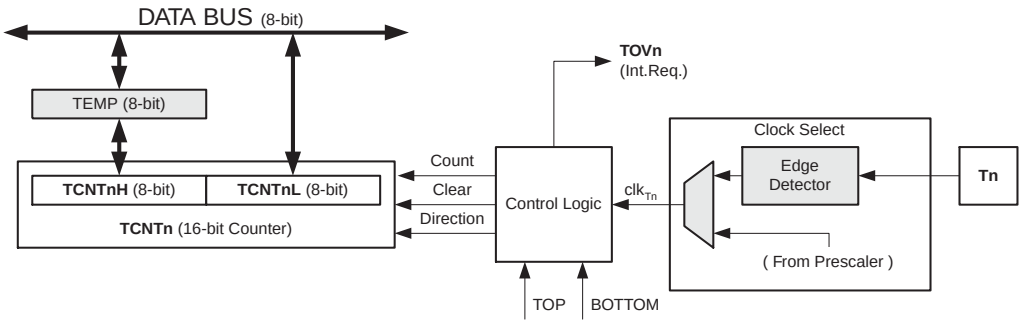
T/C 时钟源

T/C 时钟源可以来自内部，也可来自外部，由位于 T/C 控制寄存器 B(TCCR1B) 的时钟选择位 (CS12:0) 决定。时钟源与预分频器的描述见 P79 “T/C0 与 T/C1 的预分频器 ” 。

计数器单元

16 位 T/C 的主要部分是可编程的 16 位双向计数器单元。Figure 41 给出了计数器与其外围电路方框图。

Figure 41. 计数器单元方框图



信号描述 (内部信号) :

Count TCNT1 加 1 或减 1

Direction 确定是加操作还是减操作

Clear TCNT1 清零

clk_{T1} 定时器 / 计数器时钟信号

TOP 表示 TCNT1 计数器到达最大值

BOTTOM 表示 TCNT1 计数器到达最小值 (0)

16 位计数器映射到两个 8 位 I/O 存储器位置 : TCNT1H 为高 8 位 , TCNT1L 为低 8 位。CPU 只能间接访问 TCNT1H 寄存器。 CPU 访问 TCNT1H 时，实际访问的是临时寄存器 (TEMP)。读取 TCNT1L 时 , 临时寄存器的内容更新为 TCNT1H 的数值；而对 TCNT1L 执行写操作时，TCNT1H 被临时寄存器的内容所更新。这就使 CPU 可以在一个时钟周期里通过 8 位数据总线完成对 16 位计数器的读、写操作。此外还需要注意计数器在运行时的一些特殊情况。在这些特殊情况下对 TCNT1 写入数据会带来未知的结果。在合适的章节会对这些特殊情况进行具体描述。

根据工作模式的不同，在每一个 clk_{T1} 时钟到来时，计数器进行清零、加 1 或减 1 操作。clk_{T1} 由时钟选择位 CS12:0 设定。当 CS12:0= 0 时 , 计数器停止计数。不过 CPU 对 TCNT1 的读取与 clk_{T1} 是否存在无关。 CPU 写操作比计数器清零和其他操作的优先级都高。

计数器的计数序列取决于寄存器 TCCR1A 和 TCCR1B 中标志位 WGM13:0 的设置。计数器的运行 (计数) 方式与通过 OC1x 输出的波形发生方式有很紧密的关系。计数序列与波形产生的详细描述请参见 P91 “ 工作模式 ” 。

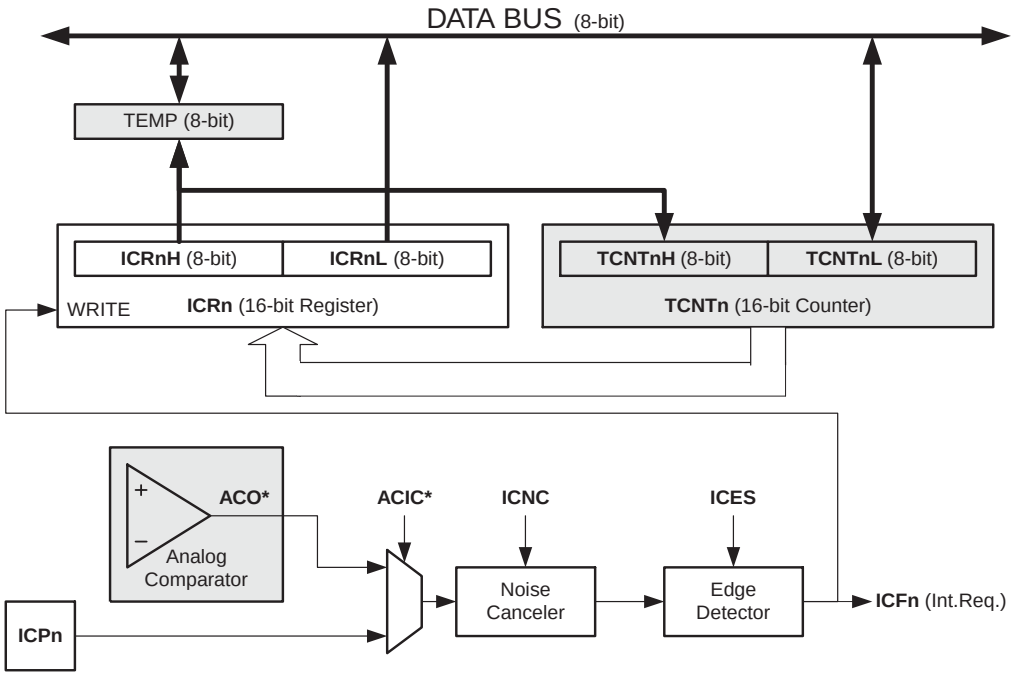
通过 WGM13:0 确定了计数器的工作模式之后，TOV1 的置位方式也就确定了。TOV1 可以用来产生 CPU 中断。

输入捕捉单元

T/C 的输入捕捉单元可用来捕获外部事件，并为其赋予时间标记以说明此时间的发生时刻。外部事件发生的触发信号由引脚 ICP1 输入，也可通过模拟比较器单元来实现。时间标记可用来计算频率、占空比及信号的其它特征，以及为事件创建日志。

输入捕捉单元方框图见 Figure 42。图中不直接属于输入捕捉单元的部分用阴影表示。寄存器与位中的小写 “n” 表示定时器 / 计数器编号。

Figure 42. 输入捕捉单元方框图



当引脚 ICP1 上的逻辑电平（事件）发生了变化，或模拟比较器输出 ACO 电平发生了变化，并且这个电平变化为边沿检测器所证实，输入捕捉即被激发 : 16 位的 TCNT1 数据被拷贝到输入捕捉寄存器 ICR1，同时输入捕捉标志位 ICF1 置位。如果此时 ICIE1 = 1，输入捕捉标志将产生输入捕捉中断。中断执行时 ICF1 自动清零，或者也可通过软件在其对应的 I/O 位置写入逻辑 "1” 清零。

读取 ICR1 时要先读低字节 ICR1L，然后再读高字节 ICR1H。读低字节时，高字节被复制到高字节临时寄存器 TEMP。 CPU 读取 ICR1H 时将访问 TEMP 寄存器。

对 ICR1 寄存器的写访问只存在于波形产生模式。此时 ICR1 被用作计数器的 TOP 值。写 ICR1 之前首先要设置 WGM13:0 以允许这个操作。对 ICR1 寄存器进行写操作时必须先将高字节写入 ICR1H I/O 位置，然后再将低字节写入 ICR1L。

请参见 P84 “ 访问 16 位寄存器 ” 以了解更多的关于如何访问 16 位寄存器的信息。

输入捕捉触发源

输入捕捉单元的主要触发源是 ICP1。T/C1 还可用模拟比较输出作为输入捕捉单元的触发源。用户必须通过设置模拟比较控制与状态寄存器 ACSR 的模拟比较输入捕捉位 ACIC 来做到这一点。要注意的是，改变触发源有可能造成一次输入捕捉。因此在改变触发源后必须对输入捕捉标志执行一次清零操作以避免出现错误的结果。

ICP1 与 ACO 的采样方式与 T1 引脚是相同的 (P79 Figure 38), 使用的边沿检测器也一样。但是使能噪声抑制器后，在边沿检测器前会加入额外的逻辑电路并引入 4 个系统时钟周期的延迟。要注意的是，除去使用 ICR1 定义 TOP 的波形产生模式外， T/C 中的噪声抑制器与边沿检测器总是使能的。

输入捕捉也可以通过软件控制引脚 ICP1 的方式来触发。

噪声抑制器

噪声抑制器通过一个简单的数字滤波方案提高系统抗噪性。它对输入触发信号进行 4 次采样。只有当 4 次采样值相等时其输出才会送入边沿检测器。

置位 TCCR1B 的 ICNC1 将使能噪声抑制器。使能噪声抑制器后，在输入发生变化到 ICR1 得到更新之间将会有额外的 4 个系统时钟周期的延时。噪声抑制器使用的是系统时钟，因而不受预分频器的影响。

输入捕捉单元的使用

使用输入捕捉单元的最大问题就是分配足够的处理器资源来处理输入事件。事件的时间间隔是关键。如果处理器在下一次事件出现之前没有读取 ICR1 的数据， ICR1 就会被新值覆盖，从而无法得到正确的捕捉结果。

使用输入捕捉中断时，中断程序应尽可能早的读取 ICR1 寄存器。尽管输入捕捉中断优先级相对较高，但最大中断响应时间与其它正在运行的中断程序所需的时间相关。

在任何输入捕捉工作模式下都不推荐在操作过程中改变 TOP 值。

测量外部信号的占空比时要求每次捕捉后都要改变触发沿。因此读取 ICR1 后必须尽快改变敏感的信号边沿。改变边沿后，ICF1 必须由软件清零 (在对应的 I/O 位置写 "1")。若仅需测量频率，且使用了中断发生，则不需对 ICF1 进行软件清零。

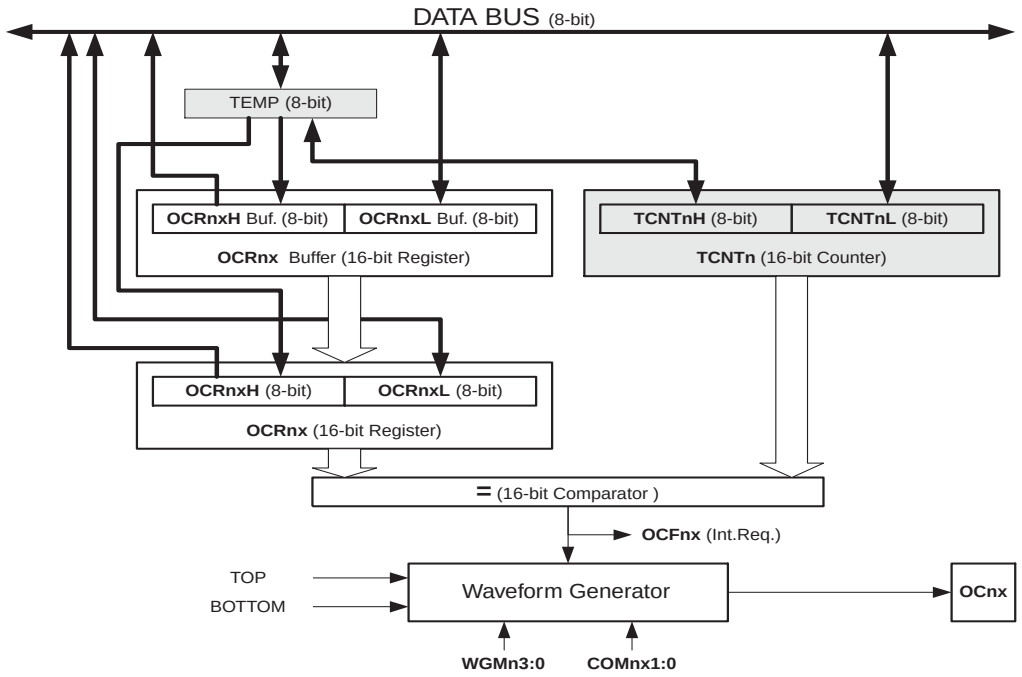
输出比较单元

16 位比较器持续比较 TCNT1 与 OCR1x 的内容，一旦发现它们相等，比较器立即产生一个匹配信号。然后 OCF1x 在下一个定时器时钟置位。如果此时 OCIE1x = 1， OCF1x 置位将引发输出比较中断。中断执行时 OCF1x 标志自动清零，或者通过软件在其相应的 I/O 位置写入逻辑 "1" 也可以清零。根据 WGM13:0 与 COM1x1:0 的不同设置，波形发生器用匹配信号生成不同的波形。波形发生器利用 TOP 和 BOTTOM 信号处理在某些模式下对极值的操作 (P91 “ 工作模式 ”)。

输出比较单元 A 的一个特质是定义 T/C 的 TOP 值 (即计数器的分辨率)。此外， TOP 值还用来定义通过波形发生器产生的波形的周期。

Figure 43 给出输出比较单元的方框图。寄存器与位上的小写 “n” 表示器件编号 (n = 1 表示 T/C1)， “x” 表示输出比较单元 (A/B)。框图中非输出比较单元部分用阴影表示。

Figure 43. 输出比较单元方框图



当 T/C 工作在 12 种 PWM 模式种的任意一种时， OCR1x 寄存器为双缓冲寄存器；而在正常工作模式和匹配时清零模式 (CTC) 双缓冲功能是禁止的。双缓冲可以实现 OCR1x 寄存器对 TOP 或 BOTTOM 的同步更新，防止产生不对称的 PWM 波形，消除毛刺。

访问 OCR1x 寄存器看起来很复杂，其实不然。使能双缓冲功能时，CPU 访问的是 OCR1x 缓冲寄存器；禁止双缓冲功能时 CPU 访问的则是 OCR1x 本身。 OCR1x(缓冲或比较) 寄存器的内容只有写操作才能将其改变 (T/C 不会自动将此寄存器更新为 TCNT1 或 ICR1 的内容)，所以 OCR1x 不用通过 TEMP 读取。但是象其他 16 位寄存器一样首先读取低字节是一个好习惯。由于比较是连续进行的，因此在写 OCR1x 时必须通过 TEMP 寄存器来实现。首先需要写入的是高字节 OCR1xH。当 CPU 将数据写入高字节的 I/O 地址时，TEMP 寄存器的内容即得到更新。接下来写低字节 OCR1xL。在此同时，位于 TEMP 寄存器的高字节数据被拷贝到 OCR1x 缓冲器，或是 OCR1x 比较寄存器。

请参见 P84 “ 访问 16 位寄存器 ” 以了解更多的关于如何访问 16 位寄存器的信息。

强制输出比较

工作于非 PWM 模式时，可以通过对强制输出比较位 FOC1x 写 ”1” 的方式来产生比较匹配。强制比较匹配不会置位 OCF1x 标志，也不会重载 / 清零定时器，但是 OC1x 引脚将被更新，好象真的发生了比较匹配一样 (COMx1:0 决定 OC1x 是置位、清零，还是交替变化)。

写 TCNT1 操作阻止比较匹配

CPU对TCNT1寄存器的写操作会阻止比较匹配的发生。这个特性可以用来将OCR1x初始化为与 TCNT1 相同的数值而不触发中断。

使用输出比较单元

由于在任意模式下写 TCNT1 都将在下一个定时器时钟周期里阻止比较匹配，在使用输出比较时改变TCNT1就会有风险，不管T/C是否在运行。若写入TCNT1的数值等于OCR1x，比较匹配就被忽略了，造成不正确的波形发生结果。在 PWM 模式下，当 TOP 为可变数值时，不要赋予 TCNT1 和 TOP 相等的数值。否则会丢失一次比较匹配，计数器也将计到0xFFFF。类似地，在计数器进行降序计数时不要对TCNT1写入等于BOTTOM的数据。

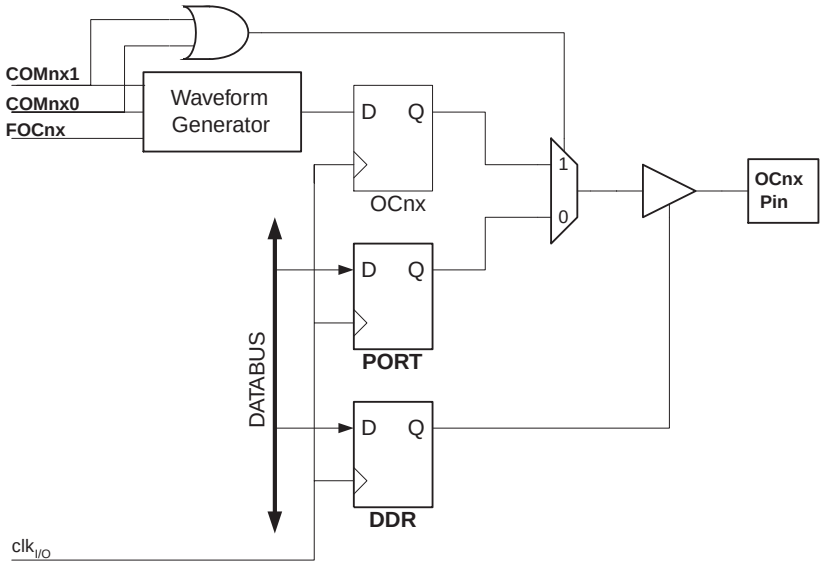
OC1x 的设置应该在设置数据方向寄存器之前完成。最简单的设置 OC1x 的方法是在普通模式下利用强制输出比较 FOC1x。即使在改变波形发生模式时 OC1x 寄存器也会一直保持它的数值。

COM1x1:0 和比较数据都不是双缓冲的。 COM1x1:0 的改变将立即生效。

比较匹配输出单元

比较匹配模式控制位 COM1x1:0 具有双重功能。波形发生器利用 COM1x1:0 来确定下一次比较匹配发生时的输出比较 OC1x 状态； COM1x1:0 还控制 OC1x 引脚输出的来源。Figure 44 为受 COM1x1:0 设置影响的逻辑的简化原理图。I/O 寄存器、I/O 位和 I/O 引脚以粗体表示。图中只给出了受 COM1x1:0 影响的通用 I/O 端口控制寄存器 (DDR 和 PORT)。谈及 OC1x 状态时指的是内部 OC1x 寄存器，而不是 OC1x 引脚的状态。系统复位时 COM1x 寄存器复位为 "0”。

Figure 44. 比较匹配输出单元原理图



只要 COM1x1:0 不全为零，波形发生器的输出比较功能就会重载 OC1x 的通用 I/O 口功能。但是 OC1x 引脚的方向仍旧受控于数据方向寄存器 (DDR)。从 OC1x 引脚输出有效信号之前必须通过数据方向寄存器的 DDR_OC1x 将此引脚设置为输出。一般情况下功能重载与波形发生器的工作模式无关，但也由一些例外，详见 Table 44、Table 45 与 Table 46 。

输出比较逻辑的设计允许 OC1x 在输出之前首先进行初始化。要注意某些 COM1x1:0 设置在某些特定的工作模式下是保留的，如 P100 “16 位定时器 / 计数器寄存器的说明 ”。

COM1x1:0 不影响输入捕捉单元。

比较输出模式和波形产生

波形发生器利用 COM1x1:0 的方法在普通模式、 CTC 模式和 PWM 模式下有所区别。对于所有的模式，设置 COM1x1:0 = 0 表明比较匹配发生时波形发生器不会操作 OC1x 寄存器。非PWM模式的比较输出请参见 P99 Table 44 ；快速PWM的比较输出于 P100 Table 45 ；相位修正 PWM 的比较输出于 P100 Table 46 。

改变 COM1x1:0 将影响写入数据后的第一次比较匹配。对于非 PWM 模式，可以通过使用 FOC1x 来立即产生效果。

工作模式

工作模式 - T/C 和输出比较引脚的行为 - 由波形发生模式 (WGM13:0) 及比较输出模式 (COM1x1:0) 的控制位决定。比较输出模式对计数序列没有影响，而波形产生模式对计数序列则有影响。 COM1x1:0 控制 PWM 输出是否为反极性。非 PWM 模式时 COM1x1:0 控制输出是否应该在比较匹配发生时置位、清零，或是电平取反 (P91 “ 比较匹配输出单元 ”)。

具体的时序信息请参考 P97 “ 定时器 / 计数器时序图 ”。

普通模式

普通模式 (WGM13:0 = 0) 为最简单的工作模式。在此模式下计数器不停地累加。计到最大值后 (TOP = 0xFFFF) 由于数值溢出计数器简单地返回到最小值 0x0000 重新开始。在TCNT1为零的同一个定时器时钟里T/C溢出标志TOV1置位。此时TOV1有点象第17位，只是只能置位，不会清零。但由于定时器中断服务程序能够自动清零 TOV1，因此可以通过软件提高定时器的分辨率。在普通模式下没有什么需要特殊考虑的，用户可以随时写入新的计数器数值。

在普通模式下输入捕捉单元很容易使用。要注意的是外部事件的最大时间间隔不能超过计数器的分辨率。如果事件间隔太长，必须使用定时器溢出中断或预分频器来扩展输入捕捉单元的分辨率。

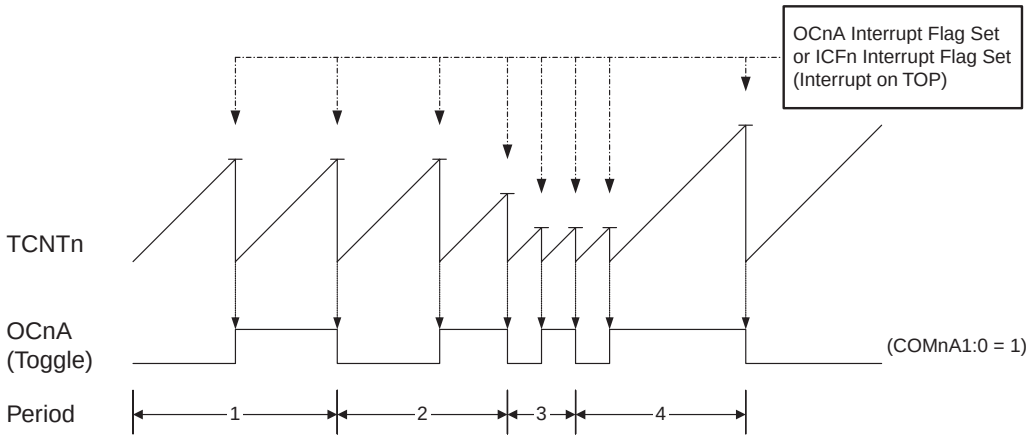
输出比较单元可以用来产生中断。但是不推荐在普通模式下利用输出比较来产生波形，因为会占用太多的 CPU 时间。

CTC(比较匹配时清零定时器) 模式

在 CTC 模式 (WGM13:0 = 4 或 12) 里 OCR1A 或 ICR1 寄存器用于调节计数器的分辨率。当计数器的数值 TCNT1 等于 OCR1A(WGM13:0 = 4) 或等于 ICR1 (WGM13:0 = 12) 时计数器清零。 OCR1A 或 ICR1 定义了计数器的 TOP 值，亦即计数器的分辨率。这个模式使得用户可以很容易地控制比较匹配输出的频率，也简化了外部事件计数的操作。

CTC模式的时序图为Figure 45。计数器数值TCNT1一直累加到TCNT1与OCR1A 或ICR1匹配，然后 TCNT1 清零。

Figure 45. CTC 模式的时序图



利用 OCF1A 或 ICF1 标志可以在计数器数值达到 TOP 时产生中断。在中断服务程序里可以更新 TOP 的数值。由于 CTC 模式没有双缓冲功能，在计数器以无预分频器或很低的预分频器工作的时候将 TOP 更改为接近 BOTTOM 的数值时要小心。如果写入的 OCR1A 或 ICR1 的数值小于当前 TCNT1 的数值，计数器将丢失一次比较匹配。在下一次比较匹配发生之前，计数器不得不先计数到最大值 0xFFFF，然后再从 0x0000 开始计数到 OCR1A 或 ICR1。在许多情况下，这一特性并非我们所希望的。替代的方法是使用快速 PWM 模式，该模式使用 OCR1A 定义 TOP 值 (WGM13:0 = 15)，因为此时 OCR1A 为双缓冲。

为了在CTC模式下得到波形输出，可以设置OC1A在每次比较匹配发生时改变逻辑电平。这可以通过设置 COM1A1:0 = 1 来完成。在期望获得 OC1A 输出之前，首先要将其端口设置为输出 (DDR_OC1A = 1)。波形发生器能够产生的最大频率为 $f_{OC1A} = f_{clk\_I/O}/2$ (OCR1A = 0x0000)。频率由如下公式确定：

$$f_{OCnA} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot (1 + OCRnA)}$$

变量 N 代表预分频因子 (1、 8、 64、 256 或 1024)。

在普通模式下，TOV1 标志的置位发生在计数器从 MAX 变为 0x0000 的定时器时钟周期。

快速 PWM 模式

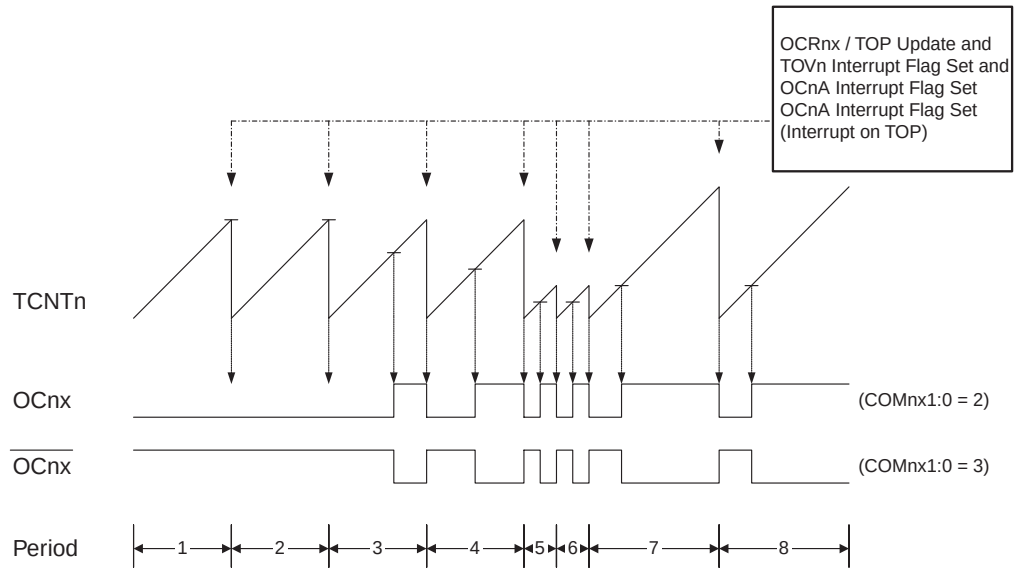
快速 PWM 模式 (WGM13:0 = 5、 6、7、 14 或 15) 可用来产生高频的 PWM 波形。快速 PWM 模式与其他 PWM 模式的不同之处是其单边斜坡工作方式。计数器从 BOTTOM 计到 TOP ,然后立即回到 BOTTOM 重新开始。对于普通的比较输出模式 ,输出比较引脚 OC1x 在 TCNT1 与 OCR1x 匹配时置位，在 TOP 时清零；对于反向比较输出模式， OCR1x 的动作正好相反。由于使用了单边斜坡模式 ,快速 PWM 模式的工作频率比使用双斜坡的相位修正 PWM 模式高一倍。此高频操作特性使得快速 PWM 模式十分适合于功率调节 ,整流和 DAC 应用。高频可以减小外部元器件 (电感 , 电容) 的物理尺寸 ,从而降低系统成本。

工作于快速 PWM 模式时 ,PWM 分辨率可固定为 8、9 或 10 位 ,也可由 ICR1 或 OCR1A 定义。最小分辨率为 2 比特 (ICR1 或 OCR1A 设为 0x0003)，最大分辨率为 16 位 (ICR1 或 OCR1A 设为 MAX)。 PWM 分辨率位数可用下式计算：

$$R_{FPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

工作于快速 PWM 模式时 ,计数器的数值一直累加到固定数值 0x00FF、0x01FF、0x03FF (WGM13:0 = 5、6 或 7)、ICR1 (WGM13:0 = 14) 或 OCR1A (WGM13:0 = 15) ,然后在后面的一个时钟周期清零。具体的时序图为 Figure 46。图中给出了当使用 OCR1A 或 ICR1 来定义 TOP 值时的快速 PWM 模式。图中柱状的 TCNT1 表示这是单边斜坡操作。方框图同时包含了普通的 PWM 输出以及反向 PWM 输出。 TCNT1 斜坡上的短水平线表示 OCR1x 和 TCNT1 的匹配比较。比较匹配后 OC1x 中断标志置位。

Figure 46. 快速 PWM 模式时序图



计时器数值达到 TOP 时 T/C 溢出标志 TOV1 置位。另外若 TOP 值是由 OCR1A 或 ICR1 定义的 ,则 OC1A 或 ICF1 标志将与 TOV1 在同一个时钟周期置位。如果中断使能，可以在中断服务程序里来更新 TOP 以及比较数据。

改变TOP值时必须保证新的TOP值不小于所有比较寄存器的数值。否则TCNT1与OCR1x不会出现比较匹配。使用固定的 TOP 值时，向任意 OCR1x 寄存器写入数据时未使用的位将屏蔽为 "0”。

定义 TOP 值时更新 ICR1 与 OCR1A 的步骤时不同的。ICR1 寄存器不是双缓冲寄存器。这意味着当计数器以无预分频器或很低的预分频工作的时候，给 ICR1 赋予一个小的数值时存在着新写入的 ICR1 数值比 TCNT1 当前值小的危险。结果是计数器将丢失一次比较匹配。在下一次比较匹配发生之前，计数器不得不先计数到最大值 0xFFFF，然后再从 0x0000 开始计数 ,直到比较匹配出现。而 OCR1A 寄存器则是双缓冲寄存器。这一特性决定 OCR1A 可以随时写入。写入的数据被放入 OCR1A 缓冲寄存器。在 TCNT1 与 TOP 匹

配后的下一个时钟周期，OCR1A 比较寄存器的内容被缓冲寄存器的数据所更新。在同一个时钟周期 TCNT1 被清零，而 TOV1 标志被设置。

使用固定 TOP 值时最好使用 ICR1 寄存器定义 TOP。这样 OCR1A 就可以用于在 OC1A 输出 PWM 波。但是，如果 PWM 基频不断变化 (通过改变 TOP 值)，OCR1A 的双缓冲特性使其更适合于这个应用。

工作于快速 PWM 模式时，比较单元可以在 OC1x 引脚上输出 PWM 波形。设置 COM1x1:0 为 2 可以产生普通的 PWM 信号；为 3 则可以产生反向 PWM 波形 (参见 P99 Table 44)。此外，要真正从物理引脚上输出信号还必须将 OC1x 的数据方向 DDR_OC1x 设置为输出。产生 PWM 波形的机理是 OC1x 寄存器在 OCR1x 与 TCNT1 匹配时置位 (或清零)，以及在计数器清零 (从 TOP 变为 BOTTOM) 的那一个定时器时钟周期清零 (或置位)。

输出的 PWM 频率可以通过如下公式计算得到：

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot (1 + TOP)}$$

变量 N 代表分频因子 (1、8、64、 256 或 1024)。

OCR1x 寄存器为极限值时说明了快速 PWM 模式的一些特殊情况。若 OCR1x 等于 BOTTOM(0x0000)，输出为出现在第 TOP+1 个定时器时钟周期的窄脉冲；OCR1x 为 TOP 时，根据 COM1x1:0 的设定，输出恒为高电平或低电平。

通过设定 OC1A 在比较匹配时进行逻辑电平取反 (COM1A1:0 = 1)，可以得到占空比为 50% 的周期信号。这只适用于 OCR1A 用来定义 TOP 值的情况 (WGM13:0 = 15)。OCR1A 为 0(0x0000) 时信号有最高频率 $f_{OC1A} = f_{clk\_I/O}/2$。这个特性类似于 CTC 模式下的 OC1A 取反操作，不同之处在于快速 PWM 模式具有双缓冲。

相位修正 PWM 模式

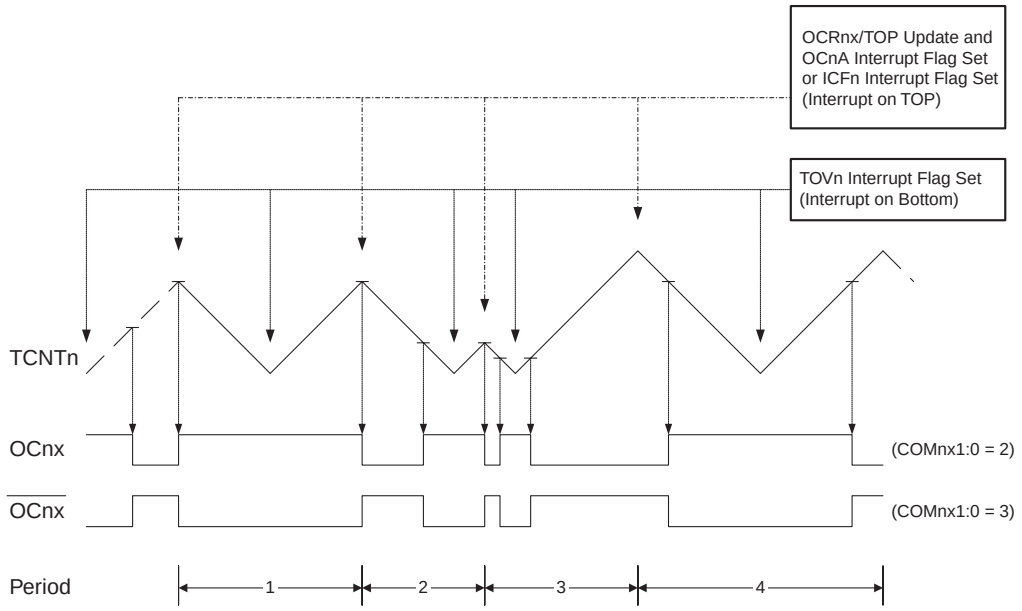
相位修正 PWM 模式 (WGM13:0 = 1、2、3、10 或 11) 为用户提供了一个获得高精度的、相位准确的 PWM 波形的方法。与相位和频率修正模式类似，此模式基于双斜坡操作。计时器重复地从 BOTTOM 计到 TOP，然后又从 TOP 倒退回到 BOTTOM。在一般的比较输出模式下，当计时器往 TOP 计数时若 TCNT1 与 OCR1x 匹配，OC1x 将清零为低电平；而在计时器往 BOTTOM 计数时若 TCNT1 与 OCR1x 匹配，OC1x 将置位为高电平。工作于反向比较输出时则正好相反。与单斜坡操作相比，双斜坡操作可获得的最大频率要小。但其对称特性十分适合于电机控制。

相位修正 PWM 模式的 PWM 分辨率固定为 8、 9 或 10 位，或由 ICR1 或 OCR1A 定义。最小分辨率为 2 比特 (ICR1 或 OCR1A 设为 0x0003)，最大分辨率为 16 位 (ICR1 或 OCR1A 设为 MAX)。 PWM 分辨率位数可用下式计算：

$$R_{PCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

工作于相位修正 PWM 模式时，计数器的数值一直累加到固定值 0x00FF、 0x01FF、0x03FF (WGM13:0 = 1、2 或 3)、ICR1 (WGM13:0 = 10) 或 OCR1A (WGM13:0 = 11)，然后改变计数方向。在一个定时器时钟里 TCNT1 值等于 TOP 值。具体的时序图为 Figure 47。图中给出了当使用 OCR1A 或 ICR1 来定义 TOP 值时的相位修正 PWM 模式。图中柱状的 TCNT1 表示这是双边斜坡操作。方框图同时包含了普通的 PWM 输出以及反向 PWM 输出。TCNT1 斜坡上的短水平线表示 OCR1x 和 TCNT1 的匹配比较。比较匹配后 OC1x 中断标志置位。

Figure 47. 相位修正 PWM 模式的时序图



计时器数值达到 BOTTOM 时 T/C 溢出标志 TOV1 置位。若 TOP 由 OCR1A 或 ICR1 定义，在 OCR1x 寄存器通过双缓冲方式得到更新的同一个时钟周期里 OC1A 或 ICF1 标志置位。标志置位后即可产生中断。

改变TOP值时必须保证新的TOP值不小于所有比较寄存器的数值。否则TCNT1与OCR1x不会出现比较匹配。使用固定的 TOP 值时，向任意 OCR1x 寄存器写入数据时未使用的位将屏蔽为 "0"。在 Figure 47 给出的第三个周期中，在 T/C 运行于相位修正模式时改变TOP值导致了不对称输出。其原因在于OCR1x寄存器的更新时间。由于OCR1x的更新时刻为定时器 / 计数器达到 TOP 之时，因此 PWM 的循环周期起始于此，也终止于此。就是说，下降斜坡的长度取决于上一个 TOP 值，而上升斜坡的长度取决于新的 TOP 值。若这两个值不同，一个周期内两个斜坡长度不同，输出也就不对称了。

若要在 T/C 运行时改变 TOP 值，最好用相位与频率修正模式代替相位修正模式。若 TOP 保持不变，那么这两种工作模式实际没有区别。

工作于相位修正 PWM 模式时，比较单元可以在 OC1x 引脚输出 PWM 波形。设置COM1x1:0为2可以产生普通的PWM，设置COM1x1:0为3可以产生反向PWM (参见P99 Table 44)。要真正从物理引脚上输出信号还必须将OC1x的数据方向DDR_OC1x设置为输出。 OCR1x 和 TCNT1 比较匹配发生时 OC1x 寄存器将产生相应的清零或置位操作，从而产生 PWM 波形。工作于相位修正模式时 PWM 频率可由如下公式获得：

$$f_{OCnxPCPWM} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot TOP}$$

变量 N 表示预分频因子 (1、8、64、256 或 1024)。

OCR1x寄存器处于极值时表明了相位修正PWM模式的一些特殊情况。在普通PWM模式下，若 OCR1x 等于 BOTTOM，输出一直保持为低电平；若 OCR1x 等于 TOP，输出则保持为高电平。反向 PWM 模式正好相反。如果 OCR1A 用来定义 TOP 值 (WGM13:0 = 11) 且 COM1A1:0 = 1， OC1A 输出占空比为 50% 的周期信号。

相位与频率修正 PWM 模式

相位与频率修正 PWM 模式 (WGM13:0 = 8 或 9) - 以下简称相频修正 PWM 模式 - 可以产生高精度的、相位与频率都准确的 PWM 波形。与相位修正模式类似，相频修正 PWM 模式基于双斜坡操作。计时器重复地从 BOTTOM 计到 TOP，然后又从 TOP 倒退回到BOTTOM。在一般的比较输出模式下，当计时器往TOP计数时若TCNT1与OCR1x匹配，

OC1x将清零为低电平；而在计时器往BOTTOM计数时TCNT1与OCR1x匹配，OC1x将置位为高电平。工作于反向输出比较时则正好相反。与单斜坡操作相比，双斜坡操作可获得的最大频率要小。但其对称特性十分适合于电机控制。

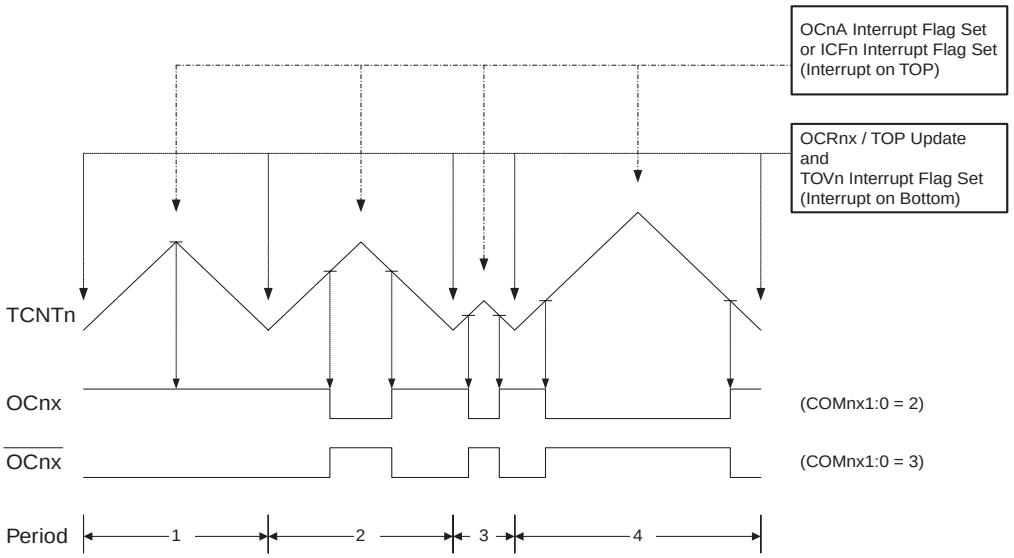
相频修正修正 PWM 模式与相位修正 PWM 模式的主要区别在于 OCR1x 寄存器的更新时间，详见 Figure 47 与 Figure 48。

相频修正修正 PWM 模式的 PWM 分辨率可由 ICR1 或 OCR1A 定义。最小分辨率为 2 比特 (ICR1 或 OCR1A 设为 0x0003)，最大分辨率为 16 位 (ICR1 或 OCR1A 设为 MAX)。PWM 分辨率位数可用下式计算：

$$R_{PFCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

工作于相频修正PWM模式时，计数器的数值一直累加到ICR1 (WGM13:0 = 8)或OCR1A (WGM13:0 = 9)，然后改变计数方向。在一个定时器时钟里TCNT1值等于TOP值。具体的时序图为 Figure 48。图中给出了当使用 OCR1A 或 ICR1 来定义 TOP 值时的相频修正PWM模式。图中柱状的TCNT1表示这是双边斜坡操作。方框图同时包含了普通的PWM输出以及反向 PWM 输出。TCNT1 斜坡上的短水平线表示 OCR1x 和 TCNT1 的匹配比较。比较匹配发生时， OC1x 中断标志将被置位。

Figure 48. 相位与频率修正 PWM 模式的时序图



在 OCR1x 寄存器通过双缓冲方式得到更新的同一个时钟周期里 T/C 溢出标志 TOV1 置位。若 TOP 由 OCR1A 或 ICR1 定义，则当 TCNT1 达到 TOP 值时 OC1A 或 CF1 置位。这些中断标志位可用来在每次计数器达到 TOP 或 BOTTOM 时产生中断。

改变TOP值时必须保证新的TOP值不小于所有比较寄存器的数值。否则TCNT1与OCR1x不会产生比较匹配。

如 Figure 48 所示，与相位修正模式形成对照的是，相频修正 PWM 模式生成的输出在所有的周期中均为对称信号。这是由于 OCR1x 在 BOTTOM 得到更新，上升与下降斜坡长度始终相等。因此输出脉冲为对称的，确保了频率是正确的。

使用固定 TOP 值时最好使用 ICR1 寄存器定义 TOP。这样 OCR1A 就可以用于在 OC1A 输出 PWM 波。但是，如果 PWM 基频不断变化 (通过改变 TOP 值)， OCR1A 的双缓冲特性使其更适合于这个应用。

工作于相频修正 PWM 模式时，比较单元可以在 OC1x 引脚上输出 PWM 波形。设置 COM1x1:0 为 2 可以产生普通的 PWM 信号；为 3 则可以产生反向 PWM 波形。(参见 P100 Table)。要想真正输出信号还必须将 OC1x 的数据方向设置为输出。产生 PWM 波形的机理是 OC1x 寄存器在 OCR1x 与升序记数的 TCNT1 匹配时置位 (或清零) ,与降序记数的 TCNT1 匹配时清零 (或置位)。输出的 PWM 频率可以通过如下公式计算得到：

$$f_{OCnxPFCPWM} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot TOP}$$

变量 N 代表分频因子 (1、8、64、 256 或 1024)。

OCR1x 寄存器处于极值时说明了相频修正 PWM 模式的一些特殊情况。在普通 PWM 模式下，若 OCR1x 等于 BOTTOM，输出一直保持为低电平；若 OCR1x 等于 TOP，则输出保持为高电平。反向 PWM 模式则正好相反。如果 OCR1A 用来定义 TOP 值 (WGM13:0 = 9) 且 COM1A1:0 = 1， OC1A 输出占空比为 50% 的周期信号。

定时器 / 计数器时序图

定时器 / 计数器为同步电路，因而时钟 clk_{T1} 表示为时钟使能信号。图中说明了何时设置中断标志及何时使用 OCR1x 缓冲器中的数据更新 OCR1x 寄存器 (工作于双缓冲器模式时)。 Figure 49 给出了置位 OCF1x 的时序图。

Figure 49. T/C 时序图 ， OCF1x 置位，无预分频器

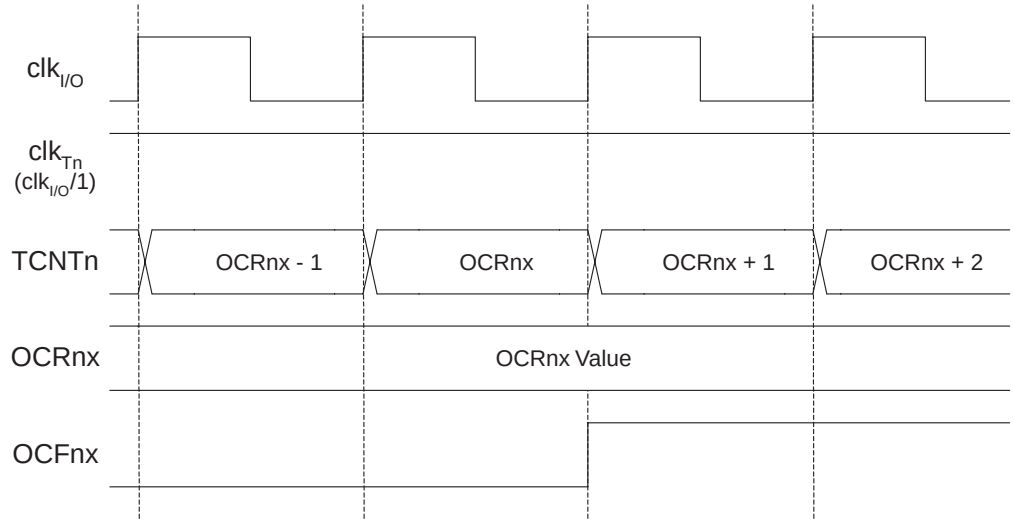


Figure 50 给出相同的时钟数据，但预分频使能。

Figure 50. T/C 时序图，置位 OCF1x，预分频器为 $f_{clk\_I/O}/8$

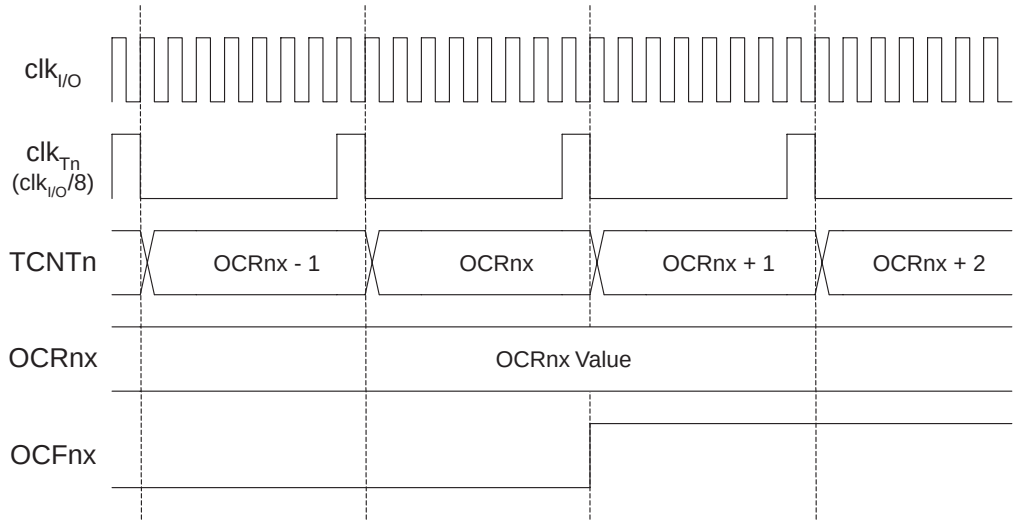


Figure 51 给出工作在不同模式下接近 TOP 值时的计数序列。工作于相频修正 PWM 模式时， OCR1x 寄存器在 BOTTOM 被更新。时序图相同，但 TOP 需要用 BOTTOM 代替，BOTTOM+1 代替 TOP-1 ,等等。同样的命名规则也适用于那些在 BOTTOM 置位 TOV1 标志的工作模式。

Figure 51. T/C 时序图，无预分频器

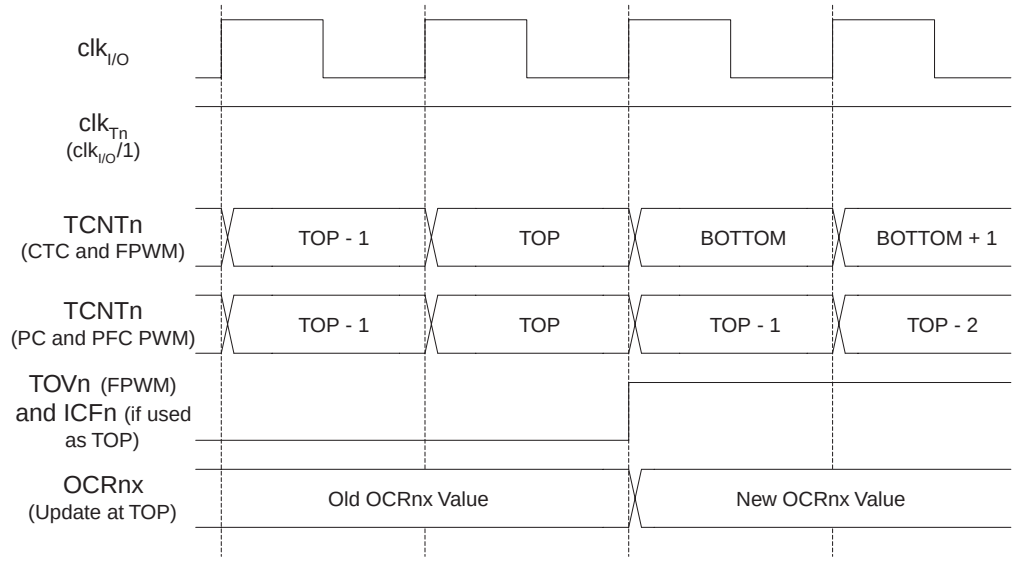
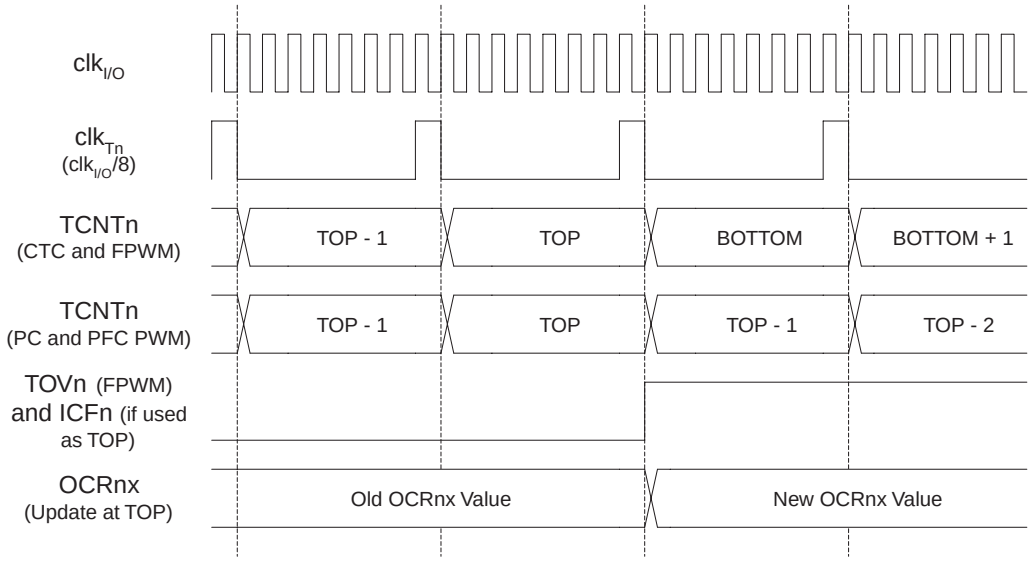


Figure 52 给出相同的时钟数据，但预分频使能。

Figure 52. T/C 时序图，预分频器为 $f_{clk\_I/O}/8$



16 位定时器 / 计数器寄存器的说明

T/C1 控制寄存器 A - TCCR1A

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
读 / 写	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7:6 – COM1A1:0: 通道 A 的比较输出模式**

- **Bit 5:4 – COM1B1:0: 通道 B 的比较输出模式**

COM1A1:0 与 COM1B1:0 分别控制 OC1A 与 OC1B 状态。如果 COM1A1:0(COM1B1:0) 的一位或两位被写入 "1”, OC1A(OC1B) 输出功能将取代 I/O 端口功能。此时 OC1A(OC1B) 相应的输出引脚数据方向控制必须置位以使能输出驱动器。

OC1A(OC1B) 与物理引脚相连时 ,COM1x1:0 的功能由 WGM13:0 的设置决定。Table 44 给出当 WGM13:0 设置为普通模式与 CTC 模式 (非 PWM) 时 COM1x1:0 的功能定义。

Table 44. 比较输出模式，非 PWM

COM1A1/COM1B1	COM1A0/COM1B0	说明
0	0	普通端口操作， OC1A/OC1B 未连接
0	1	比较匹配时 OC1A/OC1B 电平取反
1	0	比较匹配时清零 OC1A/OC1B(输出低电平)
1	1	比较匹配时置位 OC1A/OC1B (输出高电平)

Table 45 给出 WGM13:0 设置为快速 PWM 模式时 COM1x1:0 的功能定义。

Table 45. 比较输出模式，快速 PWM(1)

COM1A1/COM1B1	COM1A0/COM1B0	说明
0	0	普通端口操作， OC1A/OC1B 未连接
0	1	WGM13:0 = 15: 比较匹配时 OC1A 取反 ， OC1B 不占用物理引脚。 WGM13:0 为其它值时为普通端口操作， OC1A/OC1B 未连接
1	0	比较匹配时清零 OC1A/OC1B，OC1A/OC1B 在 TOP 时置位
1	1	比较匹配时置位 OC1A/OC1B， OC1A/OC1B 在 TOP 时清零

Note: 1. 当 OCR1A/OCR1B 等于 TOP 且 COM1A1/COM1B1 置位时，比较匹配被忽略，但 OC1A/OC1B 的置位 / 清零操作有效。详见 P93 “ 快速 PWM 模式 ” 。

Table 46给出当WGM13:0设置为相位修正PWM模式或相频修正PWM模式时COM1x1:0的功能定义。

Table 46. 比较输出模式，相位修正及相频修正 PWM 模式 (1)

COM1A1/COM1B1	COM1A0/COM1B0	说明
0	0	普通端口操作， OC1A/OC1B 未连接
0	1	WGM13:0 = 9 或 14: 比较匹配时 OC1A 取反， OC1B 不占用物理引脚。WGM13:0为其它值时为普通端口操作， OC1A/OC1B 未连接
1	0	升序记数时比较匹配将清零 OC1A/OC1B，降序记数时比较匹配将置位 OC1A/OC1B
1	1	升序记数时比较匹配将置位 OC1A/OC1B，降序记数时比较匹配将清零 OC1A/OC1B

Note: 1. OCR1A/OCR1B 等于 TOP 且 COM1A1/COM1B1 置位是一个特殊情况。详细信息请参见 P94 “ 相位修正 PWM 模式 ” 。

- **Bit 3 – FOC1A: 通道 A 强制输出比较**
- **Bit 2 – FOC1B: 通道 B 强制输出比较**

FOC1A/FOC1B只有当WGM13:0指定为非PWM模式时被激活。为与未来器件兼容，工作在 PWM 模式下对 TCCR1A 写入时，这两位必须清零。当 FOC1A/FOC1B 位置 1 ，立即强制波形产生单元进行比较匹配。 COM1x1:0 的设置改变 OC1A/OC1B 的输出。注意 FOC1A/FOC1B 位作为选通信号。 COM1x1:0 位的值决定强制比较的效果。

在 CTC 模式下使用 OCR1A 作为 TOP 值， FOC1A/FOC1B 选通即不会产生中断也不会清除定时器。

FOC1A/FOC1B 位总是读为 0。

- **Bit 1:0 – WGM11:0: 波形发生模式**

这两位与位于 TCCR1B 寄存器的 WGM13:2 相结合，用于控制计数器的计数序列——计数器计数的上限值和确定波形发生器的工作模式见 Table 47。T/C 支持的工作模式有 : 普

通模式 (计数器) , 比较匹配时清零定时器 (CTC) 模式 , 及三种脉宽调制 (PWM) 模式 , 见 P91 “ 工作模式 ” 。

Table 47. 波形产生模式的位描述 (1)

模式	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	定时器 / 计数器工作模式	TOP	OCR1x 更新时刻	TOV1 置位时刻
0	0	0	0	0	普通模式	0xFFFF	立即更新	MAX
1	0	0	0	1	8 位相位修正 PWM	0x00FF	TOP	BOTTOM
2	0	0	1	0	9 位相位修正 PWM	0x01FF	TOP	BOTTOM
3	0	0	1	1	10 位相位修正 PWM	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	立即更新	MAX
5	0	1	0	1	8 位快速 PWM	0x00FF	TOP	TOP
6	0	1	1	0	9 位快速 PWM	0x01FF	TOP	TOP
7	0	1	1	1	10 位快速 PWM	0x03FF	TOP	TOP
8	1	0	0	0	相位与频率修正 PWM	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	相位与频率修正 PWM	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	相位修正 PWM	ICR1	TOP	BOTTOM
11	1	0	1	1	相位修正 PWM	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	立即更新	MAX
13	1	1	0	1	保留	–	–	–
14	1	1	1	0	快速 PWM	ICR1	TOP	TOP
15	1	1	1	1	快速 PWM	OCR1A	TOP	TOP

Note: 1. CTC1 和 PWM11:0 的定义已经不再使用了，要使用 WGM12:0。但是两个版本的功能和位置是兼容的。

T/C1 控制寄存器 B - TCCR1B

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
读 / 写	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – ICNC1: 输入捕捉噪声抑制器**

置位 ICNC1 将使能输入捕捉噪声抑制功能。此时外部引脚 ICP1 的输入被滤波。其作用是从 ICP1 引脚连续进行 4 次采样。如果 4 个采样值都相等，那么信号送入边沿检测器。因此使能该功能使得输入捕捉被延迟了 4 个时钟周期。

- **Bit 6 – ICES1: 输入捕捉触发沿选择**

该位选择使用 ICP1 上的哪个边沿触发捕获事件。 ICES 为 "0" 选择的是下降沿触发输入捕捉； ICES1 为 "1" 选择的是逻辑电平的上升沿触发输入捕捉。

按照 ICES1 的设置捕获到一个事件后 ,计数器的数值被复制到 ICR1 寄存器。捕获事件还会置为 ICF1。如果此时中断使能，输入捕捉事件即被触发。

当 ICR1 用作 TOP 值 (见 TCCR1A 与 TCCR1B 寄存器中 WGM13:0 位的描述) 时 ,ICP1 与输入捕捉功能脱开，从而输入捕捉功能被禁用。

- **Bit 5 – 保留位**

该位保留。为保证与将来器件的兼容性，写 TCCR1B 时，该位必须写入 "0"。

- **Bit 4:3 – WGM13:2: 波形发生模式**

见 TCCR1A 寄存器中的描述。

- **Bit 2:0 – CS12:0: 时钟选择**

这 3 位用于选择 T/C 的时钟源，见 Figure 49 与 Figure 50。

Table 48. 时钟选择位描述

CS12	CS11	CS10	说明
0	0	0	无时钟源 (T/C 停止)
0	0	1	$clk_{I/O}$/1 (无预分频)
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 引脚，下降沿驱动
1	1	1	外部 T1 引脚，上升沿驱动

选择使用外部时钟源后，即使 T1 引脚被定义为输出，引脚上的逻辑信号电平变化仍然会驱动 T/C1 计数，这个特性允许用户通过软件来控制计数。

T/C1 - TCNT1H 与 TCNT1L

Bit	7	6	5	4	3	2	1	0	
	TCNT1[15:8]								TCNT1H
	TCNT1[7:0]								TCNT1L

读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

TCNT1H与TCNT1L组成了T/C1的数据寄存器TCNT1。通过它们可以直接对定时器/计数器单元的 16 位计数器进行读写访问。为保证 CPU 对高字节与低字节的同时读写，必须使用一个 8 位临时高字节寄存器 TEMP。TEMP 是所有的 16 位寄存器共用的，详见 P84 “ 访问 16 位寄存器 ”。

在计数器运行期间修改TCNT1的内容有可能丢失一次TCNT1与OCR1x的比较匹配操作。

写 TCNT1 寄存器将在下一个定时器周期阻塞比较匹配。

输出比较寄存器 1A - OCR1AH 与 OCR1AL

Bit	7	6	5	4	3	2	1	0	
	OCR1A[15:8]								OCR1AH
	OCR1A[7:0]								OCR1AL
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

输出比较寄存器 1B - OCR1BH 与 OCR1BL

Bit	7	6	5	4	3	2	1	0	
	OCR1B[15:8]								OCR1BH
	OCR1B[7:0]								OCR1BL
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

该寄存器中的 16 位数据与 TCNT1 寄存器中的计数值进行连续的比较，一旦数据匹配，将产生一个输出比较中断，或改变 OC1x 的输出逻辑电平。

输出比较寄存器长度为 16 位。为保证 CPU 对高字节与低字节的同时读写，必须使用一个 8 位临时高字节寄存器 TEMP。TEMP 是所有的 16 位寄存器共用的，详见 P84 “ 访问 16 位寄存器 ”。

输入捕捉寄存器 1 - ICR1H 与 ICR1L

Bit	7	6	5	4	3	2	1	0	
	ICR1[15:8]								ICR1H
	ICR1[7:0]								ICR1L
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

当外部引脚ICP1(或T/C1的模拟比较器)有输入捕捉触发信号产生时，计数器TCNT1中的值写入 ICR1 中。 ICR1 的设定值可作为计数器的 TOP 值。

输入捕捉寄存器长度为 16 位。为保证 CPU 对高字节与低字节的同时读，必须使用一个 8 位临时高字节寄存器 TEMP。 TEMP 是所有的 16 位寄存器共用的，详见 P84 “ 访问 16 位寄存器 ”。

T/C1 中断屏蔽寄存器 - TIMSK[1]

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

Note: 1. 该寄存器包含几个T/C的中断控制位，但本节中只对T1位进行说明，其余位将在各自的小节中加以说明。

- **Bit 5 – TICIE1: T/C1 输入捕捉中断使能**

当该位被设为 "1”，且状态寄存器中的 I 位被设为 "1” 时， T/C1 的输入捕捉中断使能。一旦 TIFR 的 ICF1 置位 ,CPU 即开始执行 T/C1 输入捕捉中断服务程序 (见 P42 “ 中断 ”)。

- **Bit 4 – OCIE1A: T/C1 输出比较 A 匹配中断使能**

当该位被设为 "1”，且状态寄存器中的 I 位被设为 "1” 时， T/C1 的输出比较 A 匹配中断使能。一旦 TIFR 上的 OCF1A 置位，CPU 即开始执行 T/C1 输出比较 A 匹配中断服务程序 (见 P42 “ 中断 ”) 。

- **Bit 3 – OCIE1B: T/C1 输出比较 B 匹配中断使能**

当该位被设为 "1”，且状态寄存器中的 I 位被设为 "1” 时，使能 T/C1 的输出比较 B 匹配中断使能。一旦 TIFR 上的 OCF1B 置位，CPU 即开始执行 T/C1 输出比较 B 匹配中断服务程序 (见 P42 “ 中断 ”) 。

- **Bit 2 – TOIE1: T/C1 溢出中断使能**

当该位被设为 "1”，且状态寄存器中的 I 位被设为 ”1” 时， T/C1 的溢出中断使能。一旦 TIFR 上的 TOV1 置位， CPU 即开始执行 T/C1 溢出中断服务程序 (见 P42 “ 中断 ”) 。

T/C 中断标志寄存器 - TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

Note: 该寄存器包含几个 T/C 的标志位 ,但本节中只对 T1 位进行说明 ,其余位将在各自的小节中加以说明。

- **Bit 5 – ICF1: T/C1 输入捕捉标志**

外部引脚 ICP1 出现捕捉事件时 ICF1 置位。此外，当 ICR1 作为计数器的 TOP 值时，一旦计数器值达到 TOP， ICF1 也置位。

执行输入捕捉中断服务程序时 ICF1 自动清零。也可以对其写入逻辑 "1” 来清除该标志位。

- **Bit 4 – OCF1A: T/C1 输出比较 A 匹配标志**

当 TCNT1 与 OCR1A 匹配成功时，该位被设为 "1”。

强制输出比较 (FOC1A) 不会置位 OCF1A。

执行强制输出比较匹配 A 中断服务程序时 OCF1A 自动清零。也可以对其写入逻辑 "1” 来清除该标志位。

- **Bit 3 – OCF1B: T/C1 输出比较 B 匹配标志**

当 TCNT1 与 OCR1B 匹配成功时，该位被设为 "1”。

强制输出比较 (FOC1B) 不会置位 OCF1B。

执行强制输出比较匹配 B 中断服务程序时 OCF1B 自动清零。也可以对其写入逻辑 "1” 来清除该标志位。

- **Bit 2 – TOV1: T/C1 溢出标志**

该位的设置与 T/C1 的工作方式有关。工作于普通模式和 CTC 模式时 ,T/C1 溢出时 TOV1 置位。对工作在其它模式下的 TOV1 标志位置位，见 P101 Table 47 。

执行溢出中断服务程序时 TOV1 自动清零。也可以对其写入逻辑 "1” 来清除该标志位。

8 位有 PWM 与异步操作的定时器 / 计数器 2

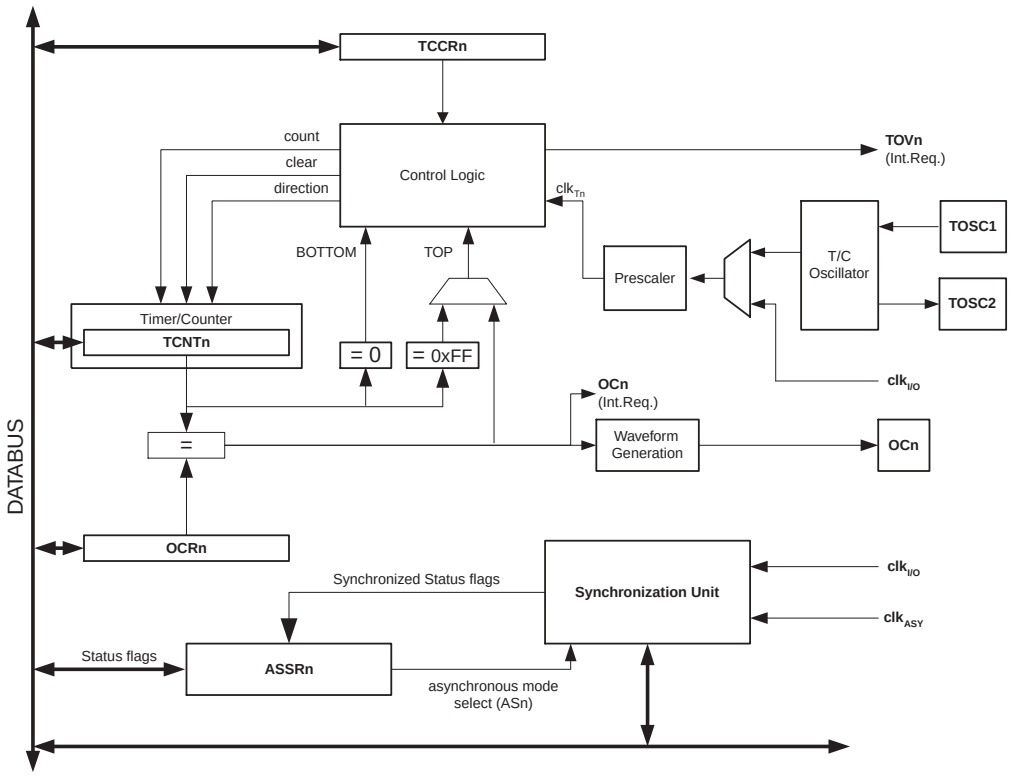
T/C2 是一个通用单通道 8 位定时 / 计数器，其主要特点如下：

- **单通道计数器**
- **比较匹配时清零定时器 (自动重载)**
- **无干扰脉冲 , 相位正确的脉宽调制器 (PWM)**
- **频率发生器**
- **10 位时钟预分频器**
- **溢出与比较匹配中断源 (TOV2 与 OCF2)**
- **允许使用外部的 32 kHz 手表晶振作为独立的 I/O 时钟源**

综述

Figure 53 为 8 位 T/C 的方框图。实际的引脚图请参见 P2 “ATmega32 的引脚 ” 。CPU 可访问的 I/O 寄存器 , 包括 I/O 位和 I/O 引脚以粗体表示。器件具体 I/O 寄存器与位定义见 P116 “8 位 T/C 寄存器说明 ” 。

Figure 53. 8 位 T/C 方框图



寄存器

定时器 / 计数器 TCNT2、输出比较寄存器 OCR2 为 8 位寄存器。中断请求 (图中简写为 Int.Req.)。信号在定时器中断标志寄存器 TIFR 都有反映。所有中断都可以通过定时器中断屏蔽寄存器 TIMSK 单独进行屏蔽。图中未给出 TIFR 与 TIMSK。

T/C 的时钟可以为通过预分频器的内部时钟或 通过由 TOSC1/2 引脚接入的异步时钟 , 详见本节后续部分。异步操作由异步状态寄存器 ASSR 控制。时钟选择逻辑模块控制引起 T/C计数值增加(或减少)的时钟源。没有选择时钟源时T/C处于停止状态。时钟选择逻辑模块的输出称为 clk_{T2}。

双缓冲的输出比较寄存器 OCR2 一直与 TCNT2 的数值进行比较。波形发生器利用比较结果产生 PWM 波形或在比较输出引脚 OC2 输出可变频率的信号。参见 P109 “ 输出比较单元 ” 。比较匹配结果还会置位比较匹配标志 OCF2，用来产生输出比较中断请求。

定义

本文的许多寄存器及其各个位以通用的格式表示。小写的 “n” 表示定时器 / 计数器的序号，在此即为 2。但是在写程序时要使用精确的格式 (例如使用 TCNT2 来访问 T/C2 计数器值)。 Table 49 中定义适用于本节。

Table 49. 说明

BOTTOM	计数器计到 0x00 时即达到 BOTTOM
MAX	计数器计到 0xFF (十进制的 255) 时即达到 MAX
TOP	计数器计到计数序列的最大值时即达到 TOP。TOP 值可以为固定值 0xFF (MAX)，或是存储于寄存器 OCR2 里的数值，具体由工作模式确定

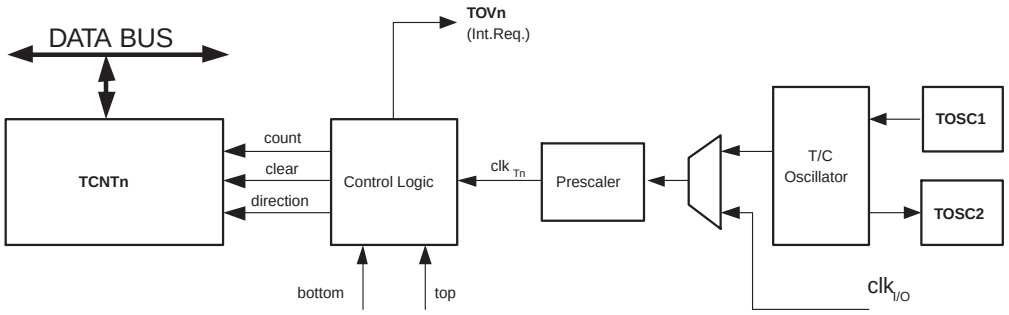
T/C 的时钟源

T/C 可以由内部同步时钟或外部异步时钟驱动。 clk_{T2} 的缺省设置为 MCU 时钟 $clk_{I/O}$。当 ASSR 寄存器的 AS2 置位时，时钟源来自于 TOSC1 和 TOSC2 连接的振荡器。详细的异步操作可以参考 P119 “ 异步状态寄存器 –ASSR” ；详细的时钟源与预分频器的内容请参考 P121 “ 定时器 / 计数器预分频器 ” 。

计数器单元

8位T/C的主要部分为可编程的双向计数单元。Figure 54 即为计数器和它周边电路的方框图。

Figure 54. 计数器单元方框图



信号说明 (内部信号) :

count 使 TCNT2 加 1 或减 1

direction 选择加操作或减操作

clear 清零 TCNT2 (将所有的位清零)

clk_{T2} T/C 的时钟

top 表示 TCNT2 已经达到了最大值

bottom 表示 TCNT2 已经达到了最小值 (0)

根据不同的工作模式，计数器针对每一个 clk_{T2} 实现清零、加一或减一操作。clk_{T2} 可以由内部时钟源或外部时钟源产生，具体由时钟选择位 CS22:0 确定。没有选择时钟源时 (CS22:0 = 0) 定时器停止。但是不管有没有 clk_{T2}，CPU 都可以访问 TCNT2。CPU 写操作比计数器其他操作 (清零、加减操作) 的优先级高。

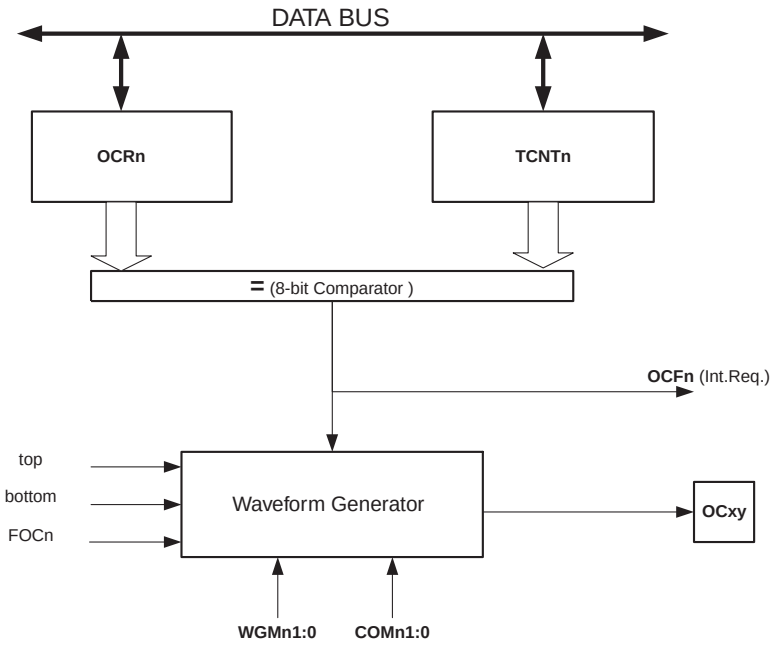
计数序列由 T/C 控制寄存器 (TCCR2) 的 WGM21 和 WGM20 决定。计数器计数行为与输出比较 OC2 的波形有紧密的关系。有关计数序列和波形产生的详细信息请参考 P110 “ 工作模式 ” 。

T/C溢出中断标志TOV2根据WGM21:0 设定的工作模式来设置。TOV2可以用于产生CPU中断。

输出比较单元

8 位比较器持续对 TCNT2 和输出比较匹配寄存器 OCR2 进行比较。一旦 TCNT2 等于 OCR2，比较器就给出匹配信号。在匹配发生的下一个定时器时钟周期里输出比较标志 OCF2 置位。若 OCIE2 = 1 还将引发输出比较中断。执行中断服务程序时 OCF2 将自动清零，也可以通过软件写 ”1” 的方式进行清零。根据 WGM21:0 和 COM21:0 设定的不同工作模式，波形发生器可以利用匹配信号产生不同的波形。同时，波形发生器还利用 max 和 bottom 信号来处理极值条件下的特殊情况 (见 P110 “ 工作模式 ”)。Figure 55 为输出比较单元的方框图。

Figure 55. 输出比较单元方框图



使用 PWM 模式时 OCR2 寄存器为双缓冲寄存器；而在正常工作模式和匹配时清零模式双缓冲功能是禁止的。双缓冲可以将更新 OCR2 寄存器与 top 或 bottom 时刻同步起来，从而防止产生不对称的 PWM 脉冲，消除毛刺。

访问 OCR2 寄存器看起来很复杂，其实不然。使能双缓冲功能时，CPU 访问的是 OCR2 缓冲寄存器；禁止双缓冲功能时 CPU 访问的则是 OCR2 本身。

强制输出比较

工作于非 PWM 模式时，可以对强制输出比较位 FOC2 写 ”1” 来产生比较匹配。强制比较匹配不会置位 OCF2 标志，也不会重载 / 清零定时器，但是 OC2 引脚将被更新，好象真的发生了比较匹配一样 (COM21:0 决定 OC2 是置位、清零，还是交替变化)。

写 TCNT2 操作阻止比较匹配

CPU 对 TCNT2 寄存器的写操作会在下一个定时器时钟周期阻止比较匹配的发生，即使此时定时器已经停止了。这个特性可以用来将 OCR2 初始化为与 TCNT2 相同的数值而不触发中断。

使用输出比较单元

由于在任意模式下写 TCNT2 都将在下一个定时器时钟周期里阻止比较匹配，在使用输出比较时改变 TCNT2 就会有风险，不管 T/C 是否在运行。如果写入的 TCNT2 的数值等于 OCR2，比较匹配就被忽略了，造成不正确的波形发生结果。类似地，在计数器进行降序计数时不要对 TCNT2 写入 BOTTOM。

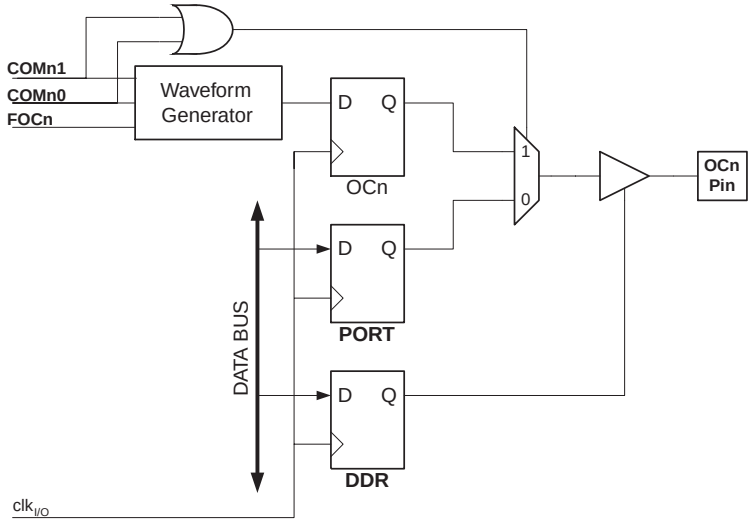
OC2 的设置应该在设置数据方向寄存器之前完成。最简单的设置 OC2 的方法是在普通模式下利用强制输出比较 FOC2。即使在改变波形发生模式时 OC2 寄存器也会一直保持它的数值。

注意 COM21:0 和比较数据都不是双缓冲的。 COM21:0 的改变将立即生效。

比较匹配输出单元

比较匹配模式控制位 COM21:0 具有双重功能。波形发生器利用 COM21:0 来确定下一次比较匹配发生时的输出比较状态 (OC2) ； COM21:0 还控制 OC2 引脚输出信号的来源。Figure 56为受COM21:0设置影响的逻辑的简化原理图。I/O寄存器、I/O位和I/O引脚以粗体表示。图中只给出了受 COM21:0 影响的通用 I/O 端口控制寄存器 (DDR 和 PORT)。谈及 OC2 状态时指的是内部 OC2 寄存器，而不是 OC2 引脚。

Figure 56. 比较匹配输出单元原理图



只要 COM21:0 的一个或两个置位，波形发生器的输出比较功能 OC2 就会取代通用 I/O 口功能。但是 OC2 引脚的方向仍然受控于数据方向寄存器 (DDR)。在使用 OC2 功能之前首先要通过数据方向寄存器的 DDR_OC2 位将此引脚设置为输出。端口功能与波形发生器的工作模式无关。

输出比较逻辑的设计允许 OC2 状态在输出之前首先进行初始化。要注意某些 COM21:0 设置保留给了其他操作类型，详见 P116 “8 位 T/C 寄存器说明 ” 。

比较输出模式和波形产生

波形发生器利用 COM21:0 的方式在普通、 CTC 和 PWM 三种模式下有所区别。对于所有的模式，COM21:0 = 0 表明比较匹配发生时波形发生器不会操作 OC2 寄存器。非 PWM 模式的比较输出请参见 P115 Table 51 ；快速 PWM 的比较输出于 P116 Table 52 ；相位修正 PWM 的比较输出于 P116 Table 53 。

改变 COM21:0 将影响写入数据后的第一次比较匹配。对于非 PWM 模式，可以通过使用 FOC2 来强制立即产生效果。

工作模式

工作模式 - T/C 和输出比较引脚的行为 - 由波形发生模式 (WGM21:0) 及比较输出模式 (COM21:0) 的控制位决定。比较输出模式对计数序列没有影响，而波形产生模式对计数序列则有影响。 COM21:0 控制 PWM 输出是否反极性。非 PWM 模式时 COM21:0 控制输出是否应该在比较匹配发生时置位、清零，或是电平取反 (P110 “ 比较匹配输出单元 ”)。

具体的时序信息请参考 P113 “T/C 时序图 ” 。

普通模式

普通模式 (WGM21:0 = 0) 为最简单的工作模式。在此模式下计数器不停地累加。计到 8 比特的最大值后 (TOP = 0xFF)，由于数值溢出计数器简单地返回到最小值 0x00 重新开始。在 TCNT0 为零的同一个定时器时钟里 T/C 溢出标志 TOV2 置位。此时 TOV2 有点象第 9 位，只是只能置位，不会清零。但由于定时器中断服务程序能够自动清零 TOV2，因此可以通过软件提高定时器的分辨率。在普通模式下没有什么需要特殊考虑的，用户可以随时写入新的计数器数值。

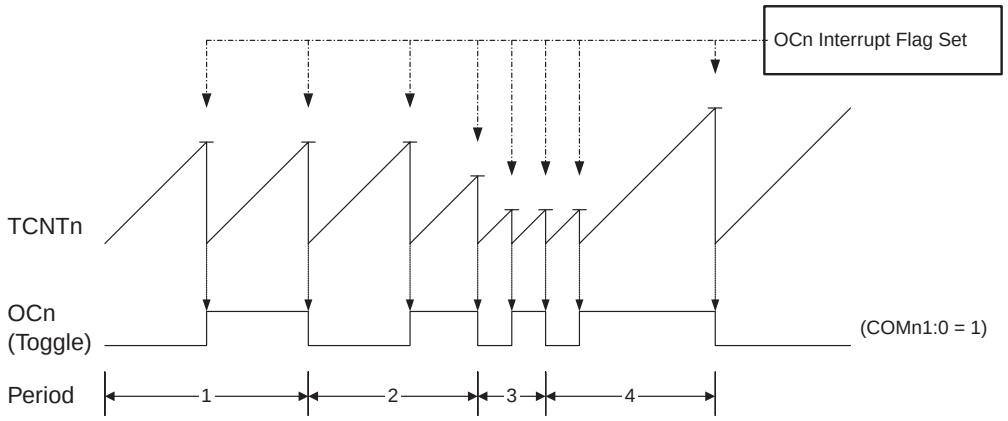
输出比较单元可以用来产生中断。但是不推荐在普通模式下利用输出比较产生波形，因为会占用太多的 CPU 时间。

CTC(比较匹配时清除定时器) 模式

在 CTC 模式 (WGM21:0 = 2) 里 OCR2 寄存器用于调节计数器的分辨率。当计数器的数值 TCNT2 等于 OCR2 时计数器清零。OCR2 定义了计数器的 TOP 值，亦即计数器的分辨率。这个模式使得用户可以很容易地控制比较匹配输出的频率，也简化了外部事件计数的操作。

CTC 模式的时序图为 Figure 57。计数器数值 TCNT2 一直累加到 TCNT2 与 OCR2 匹配，然后 TCNT2 清零。

Figure 57. CTC 模式的时序图



利用 OCF2 标志可以在计数器数值达到 TOP 即产生中断。在中断服务程序里可以更新 TOP 的数值。由于 CTC 模式没有双缓冲功能，在计数器以无预分频器或很低的预分频器工作的时候将 TOP 更改为接近 BOTTOM 的数值时要小心。如果写入 OCR2 的数值小于当前 TCNT2 的数值，计数器将丢失一次比较匹配。在下一次比较匹配发生之前，计数器不得不先计数到最大值 0xFF，然后再从 0x00 开始计数到 OCR2。

为了在 CTC 模式下得到波形输出，可以设置 OC2 在每次比较匹配发生时改变逻辑电平。这可以通过设置 COM21:0 = 1 来完成。在期望获得 OC2 输出之前，首先要将其端口设置为输出。波形发生器能够产生的最大频率为 $f_{OC2} = f_{clk\_I/O}/2$ (OCR2 = 0x00)。频率由如下公式确定：

$$f_{OCn} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot (1 + OCRn)}$$

变量 N 代表预分频因子 (1、8、 32、 64、 128、256 或 1024)。

在普通模式下， TOV2 标志的置位发生在计数器从 MAX 变为 0x00 的定时器时钟周期。

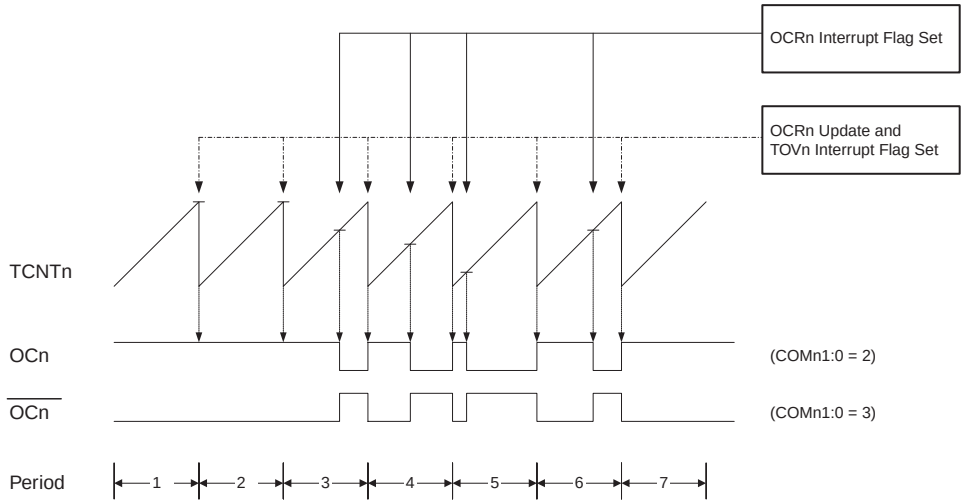
快速 PWM 模式

快速 PWM 模式 (WGM21:0 = 3) 可用来产生高频的 PWM 波形。快速 PWM 模式与其他 PWM 模式的不同之处是其单边斜坡工作方式。计数器从 BOTTOM 计到 MAX，然后立即回到 BOTTOM 重新开始。对于普通的比较输出模式，输出比较引脚 OC2 在 TCNT2 与 OCR2 匹配时清零，在 BOTTOM 时置位；对于反向比较输出模式， OC2 的动作正好相反。由于使用了单边斜坡模式，快速 PWM 模式的工作频率比使用双斜坡的相位修正 PWM 模式

高一倍。此高频操作特性使得快速 PWM 模式十分适合于功率调节，整流和 DAC 应用。高频可以减小外部元器件 (电感，电容) 的物理尺寸，从而降低系统成本。

工作于快速 PWM 模式时，计数器的数值一直增加到 MAX，然后在后面的一个时钟周期清零。具体的时序图为 Figure 58。图中柱状的 TCNT0 表示这是单边斜坡操作。方框图同时包含了普通的 PWM 输出以及方向 PWM 输出。 TCNT2 斜坡上的短水平线表示 OCR2 和 TCNT2 的比较匹配。

Figure 58. 快速 PWM 模式时序图



计时器数值达到 MAX 时 T/C 溢出标志 TOV2 置位。如果中断使能，在中断服务程序中断服务程序可以更新比较值。

工作于快速 PWM 模式时，比较单元可以在 OC2 引脚上输出 PWM 波形。设置 COM21:0 为 2 可以产生普通的 PWM 信号；为 3 则可以产生反向 PWM 波形 (参见 P116 Table 52)。要想在引脚上得到输出信号还必须将OC2的数据方向设置为输出。产生PWM波形的机理是 OC2 寄存器在 OCR2 与 TCNT2 匹配时置位 (或清零)，以及在计数器清零 (从 MAX 变为 BOTTOM) 的那一个定时器时钟周期清零 (或置位)。

输出的 PWM 频率可以通过如下公式计算得到：

$$f_{OCnPWM} = \frac{f_{clk\_I/O}}{N \cdot 256}$$

变量 N 代表分频因子 (1、8、 32、 64、 128、 256 或 1024)。

OCR2 寄存器为极限值时表示快速PWM模式的一些特殊情况。若 OCR2 等于 BOTTOM，输出为出现在第 MAX+1 个定时器时钟周期的窄脉冲； OCR2 为 MAX 时，根据 COM21:0 的设定，输出恒为高电平或低电平。

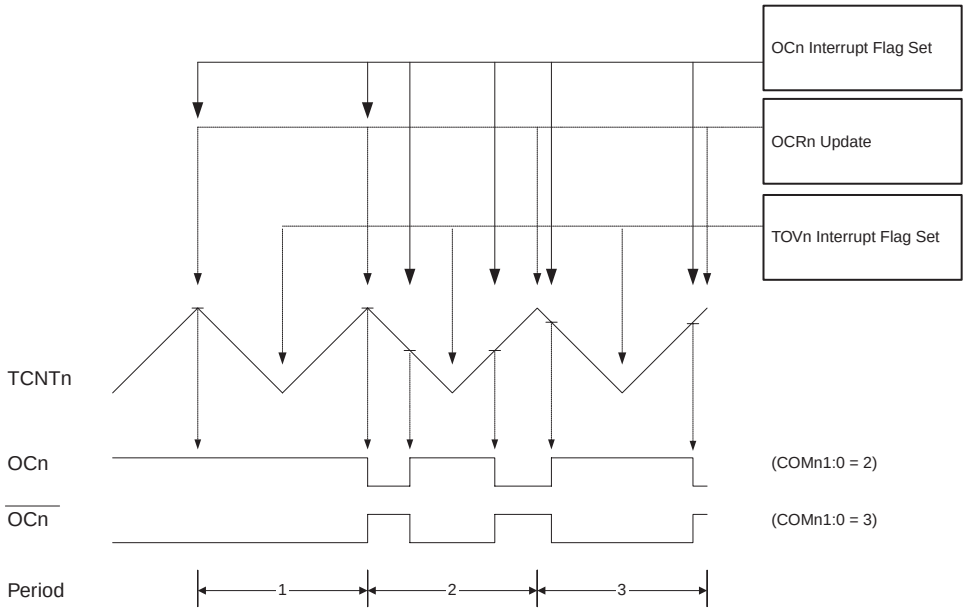
通过设定 OC2 在比较匹配时进行逻辑电平取反 (COM21:0 = 1)，可以得到占空比为 50% 的周期信号。OCR2 为 0 时信号有最高频率 $f_{oc2} = f_{clk\_I/O}/2$。这个特性类似于 CTC 模式下的 OC2 取反操作，不同之处在于快速 PWM 模式具有双缓冲。

相位修正 PWM 模式

相位修正 PWM 模式 (WGM21:0 = 1) 为用户提供了一个获得高精度相位修正 PWM 波形的方法。此模式基于双斜坡操作。计时器重复地从 BOTTOM 计到 MAX，然后又从 MAX 倒退回到BOTTOM。在一般的比较输出模式下，当计时器往MAX计数时若发生了TCNT2于OCR2的匹配，OC2将清零为低电平；而在计时器往BOTTOM计数时若发生了TCNT2于 OCR2 的匹配， OC2 将置位为高电平。工作于反向输出比较时则正好相反。与单斜坡操作相比，双斜坡操作可获得的最大频率要小。但由于其对称的特性，十分适合于电机控制。

相位修正 PWM 模式的 PWM 精度固定为 8 比特。计时器不断地累加直到 MAX，然后开始减计数。在一个定时器时钟周期里 TCNT2 的值等于 MAX。时序图可参见 Figure 59。图中 TCNT2 的数值用柱状图表示，以说明双斜坡操作。本图同时说明了普通 PWM 的输出和反向 PWM 的输出。 TCNT2 斜坡上的小横条表示 OCR2 和 TCNT2 的比较匹配。

Figure 59. 相位修正 PWM 模式的时序图



当计时器达到 BOTTOM 时 T/C 溢出标志位 TOV2 置位。此标志位可用来产生中断。

工作于相位修正 PWM 模式时，比较单元可以在 OC2 引脚产生 PWM 波形：将 COM21:0 设置为 2 产生普通相位的 PWM，设置 COM21:0 为 3 产生反向 PWM 信号 (参见 P116 Table 53)。要想在引脚上得到输出信号还必须将 OC2 的数据方向设置为输出。OCR2 和 TCNT2 比较匹配发生时 OC2 寄存器将产生相应的清零或置位操作，从而产生 PWM 波形。工作于相位修正模式时 PWM 频率可由下式公式获得：

$$f_{OCnPCPWM} = \frac{f_{clk\_I/O}}{N \cdot 510}$$

变量 N 表示预分频因子 (1、8、32、64、 128、 256 或 1024)。

OCR2 寄存器处于极值代表了相位修正 PWM 模式的一些特殊情况。在普通 PWM 模式下，若 OCR2 等于 BOTTOM，输出一直保持为低电平；若 OCR2 等于 MAX，则输出保持为高电平。反向 PWM 模式则正好相反。

在 Figure 59 的第 2 个周期，虽然没有发生比较匹配， OCn 也出现了一个从高到低的跳变。其目的是保证波形在 BOTTOM 两侧的对称。没有比较匹配时有两种情况会出现跳变

- 如 Figure 59 所示，OCR2A 的值从 MAX 改变为其他数据。当 OCR2A 值为 MAX 时，引脚 OCn 的输出应该与前面降序记数比较匹配的结果相同。为保证波形在 BOTTOM 两侧的对称，当 T/C 的数值为 MAX 时，引脚 OCn 的输出又必须符合后面升序记数比较匹配的结果。此时就出现了虽然没有比较匹配发生 OCn 却仍然有跳变的现象。
- 定时器从一个比 OCR2A 大的值开始记数，并因而丢失了一次比较匹配。因此引入了没有比较匹配发生 OCn 却仍然有跳变的现象。

T/C 时序图

下面图中给出的 T/C 为同步电路，因此其时钟 clk_{T2} 可以表示为时钟使能信号。在异步模式下， $clk_{I/O}$ 由 T/C 振荡器时钟所取代。图中还说明了中断标志设置的时间。 Figure 60 包含了 T/C 的基本时序。图中给出了除相位修正 PWM 模式的接近 MAX 值的计数序列。

Figure 60. T/C 时序图，无预分频器

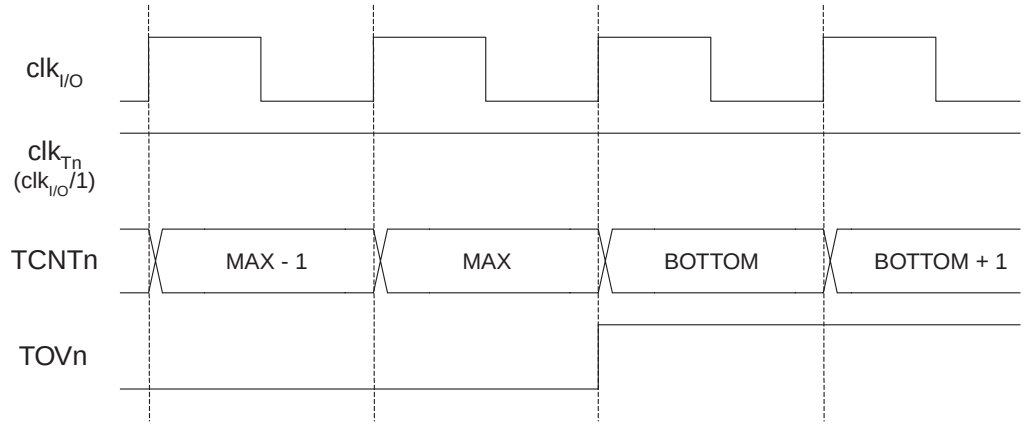


Figure 61 给出相同的定时器数据，但预分频器使能。

Figure 61. T/C 时序图，预分频器为 $f_{clk\_I/O}/8$

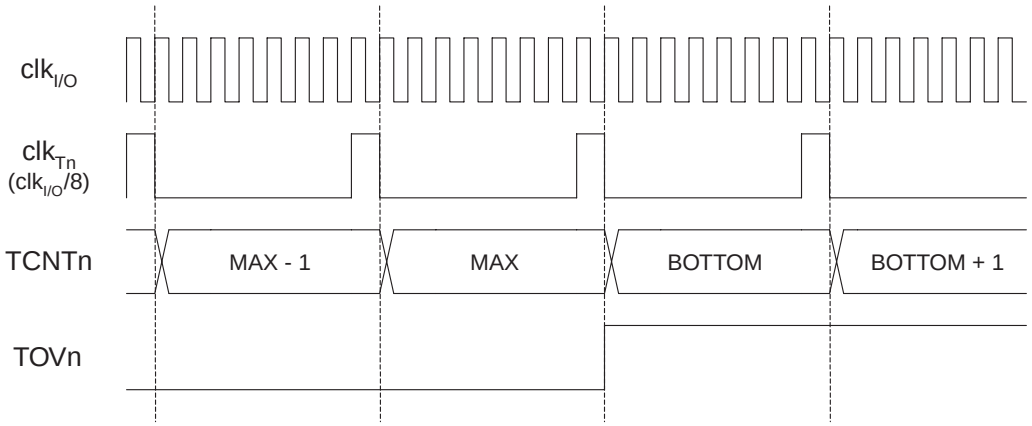


Figure 62 给出了各种模式下 (除了 CTC 模式)OCF2 的置位情况。

Figure 62. T/C 时序图， OCF2 置位，预分频器为 $f_{clk\_I/O}/8$

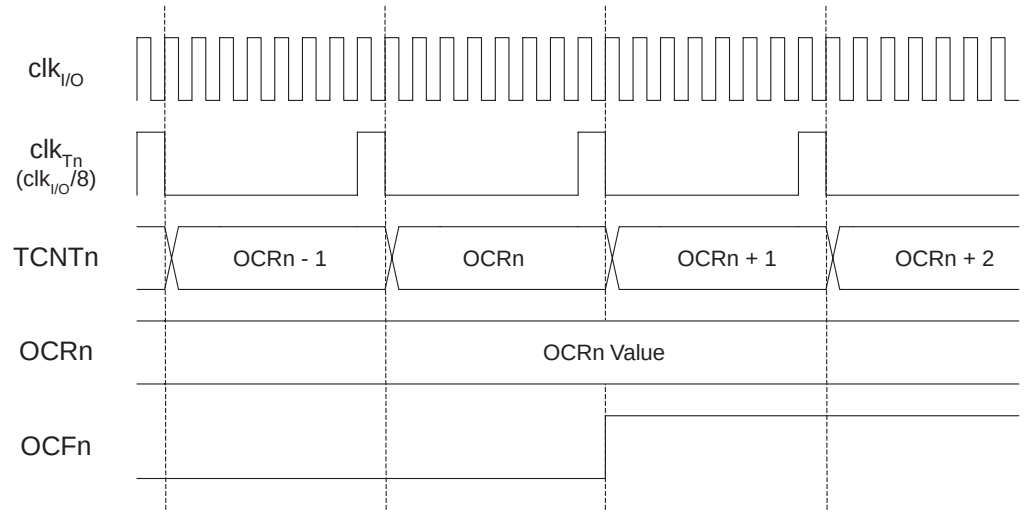
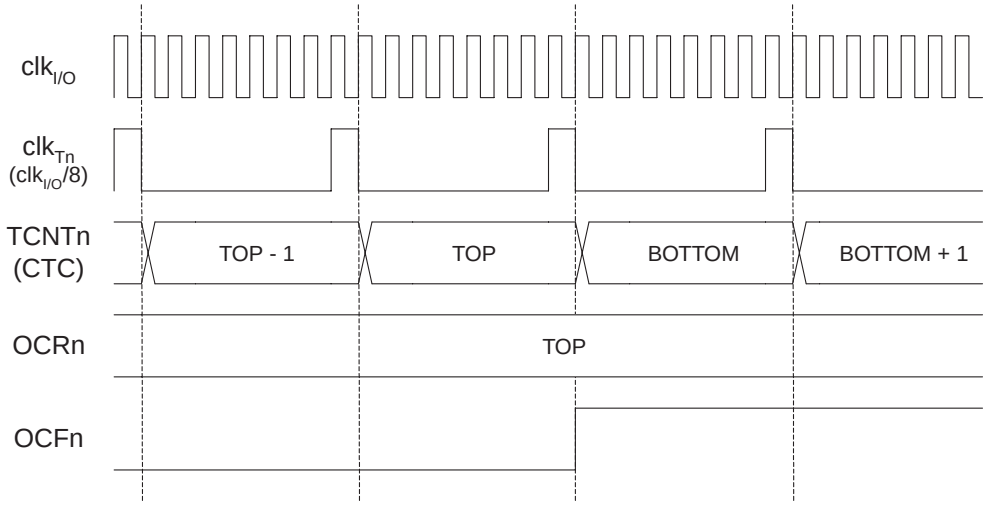


Figure 63 给出了 CTC 模式下 OCF2 置位和 TCNT2 清除的情况。

Figure 63. T/C 时序图， CTC 模式，预分频器为 $f_{clk\_I/O}/8$



8 位 T/C 寄存器说明

T/C 控制寄存器 - TCCR2

Bit	7	6	5	4	3	2	1	0	
	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	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC2: 强制输出比较**

FOC2 仅在 WGM 指明非 PWM 模式时才有效。但是，为了保证与未来器件的兼容性，使用 PWM 时，写 TCCR2 要对其清零。写 1 后，波形发生器将立即进行比较操作。比较匹配输出引脚 OC2 将按照 COM21:0 的设置输出相应的电平。要注意 FOC2 类似一个锁存信号，真正对强制输出比较起作用的是 COM21:0 的设置。

FOC2 不会引发任何中断，也不会在使用 OCR2 作为 TOP 的 CTC 模式下对定时器进行清零。

读 FOC2 的返回值永远为 0。

- **Bit 6, 3 – WGM21:0: 波形产生模式**

这几位控制计数器的计数序列，计数器最大值 TOP 的来源，以及产生何种波形。 T/C 支持的模式有：普通模式，比较匹配发生时清除计数器模式 (CTC)，以及两种 PWM 模式，详见 Table 50 与 P110 “ 工作模式 ” 。

Table 50. 波形产生模式的位定义 (1)

模式	WGM21 (CTC2)	WGM20 (PWM2)	T/C 的工作模式	TOP	OCR2 的更新时间	TOV2 的置位时刻
0	0	0	普通	0xFF	立即更新	MAX
1	0	1	相位修正 PWM	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR2	立即更新	MAX
3	1	1	快速 PWM	0xFF	TOP	MAX

Note: 1. 位定义 CTC2 和 PWM2 已经不再使用了，要使用 WGM21:0。但是功能和位置与以前版本兼容。

- **Bit 5:4 – COM21:0: 比较匹配输出模式**

这些位决定了比较匹配发生时输出引脚 OC0 的电平。如果 COM01:0 中的一位或全部都置位， OC0 以比较匹配输出的方式进行工作。同时其方向控制位要设置为 1 以使能输出驱动。

当 OC0 连接到物理引脚上时， COM01:0 的功能依赖于 WGM01:0 的设置。Table 51 给出了当 WGM01:0 设置为普通模式或 CTC 模式时 COM01:0 的功能。

Table 51. 比较输出模式，非 PWM 模式

COM21	COM20	说明
0	0	正常的端口操作， OC2 未连接
0	1	比较匹配发生时 OC2 取反
1	0	比较匹配发生时 OC2 清零
1	1	比较匹配发生时 OC2 置位

Table 52 给出了当 WGM21:0 设置为快速 PWM 模式时 COM21:0 的功能。

Table 52. 比较输出模式，快速 PWM 模式 (1)

COM21	COM20	说明
0	0	正常的端口操作， OC2 未连接
0	1	保留
1	0	比较匹配发生时 OC2 清零 ，计数到 TOP 时 OC0 置位
1	1	比较匹配发生时 OC2 置位 ，计数到 TOP 时 OC0 清零

Note: 1. 一个特殊情况是 OCR2 等于 TOP ,且 COM21 置位。此时比较匹配将被忽略 ,而计数到 TOP 时的动作继续有效。详细信息请参见 P111 “ 快速 PWM 模式 ” 。

Table 53 给出了当 WGM21:0 设置为相位修正 PWM 模式时 COM21:0 的功能。

Table 53. 比较输出模式，相位修正 PWM 模式 (1)

COM21	COM20	说明
0	0	正常的端口操作，不与 OC2 相连接
0	1	保留
1	0	在升序计数时发生比较匹配将清零 OC2 ；降序计数时发生比较匹配将置位 OC2
1	1	在升序计数时发生比较匹配将置位 OC2 ；降序计数时发生比较匹配将清零 OC2

Note: 1. 一个特殊情况是 OCR2 等于 TOP ,且 COM21 置位。此时比较匹配将被忽略 ,而计数到 TOP 时的动作继续有效。详细信息请参见 P112 “ 相位修正 PWM 模式 ” 。

- **Bit 2:0 – CS22:0: 时钟选择**

这三位时钟选择位用于选择 T/C 的时钟源，见 Table 54。

Table 54. 时钟选择位说明

CS22	CS21	CS20	说明
0	0	0	无时钟， T/C 不工作
0	0	1	clk_{T2S}/(没有预分频)
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)

定时器 / 计数器寄存器 - TCNT2

Bit	7	6	5	4	3	2	1	0	
	TCNT2[7:0]								TCNT2
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

通过 T/C 寄存器可以直接对计数器的 8 位数据进行读写访问。对 TCNT2 寄存器的写访问将在下一个时钟阻止比较匹配。在计数器运行的过程中修改 TCNT2 的数值有可能丢失一次 TCNT2 和 OCR2 的比较匹配。

输出比较寄存器 - OCR2

Bit	7	6	5	4	3	2	1	0	
	OCR2[7:0]								OCR2
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

输出比较寄存器包含一个 8 位的数据，不间断地与计数器数值 TCNT2 进行比较。匹配事件可以用来产生输出比较中断，或者用来在 OC2 引脚上产生波形。

定时器 / 计数器的异步操作

异步状态寄存器 - ASSR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB	ASSR
读 / 写	R	R	R	R	R/W	R	R	R	
初始值	0	0	0	0	0	0	0	0	

- **Bit 3 – AS2: 异步 T/C2**

AS2为"0”时T/C2由I/O时钟$clk_{I/O}$驱动；AS2为"1”时T/C2由连接到TOSC1引脚的晶体振荡器驱动。改变 AS2 有可能破坏 TCNT2、 OCR2 与 TCCR2 的内容。

- **Bit 2 – TCN2UB: T/C2 更新中**

T/C2工作于异步模式时，写TCNT2将引起TCN2UB置位。当TCNT2从暂存寄存器更新完毕后 TCN2UB 由硬件清零。 TCN2UB 为 0 表明 TCNT2 可以写入新值了。

- **Bit 1 – OCR2UB: 输出比较寄存器 2 更新中**

T/C2工作于异步模式时，写OCR2将引起OCR2UB置位。当OCR2从暂存寄存器更新完毕后 OCR2UB 由硬件清零。 OCR2UB 为 0 表明 OCR2 可以写入新值了。

- **Bit 0 – TCR2UB: T/C2 控制寄存器更新中**

T/C2工作于异步模式时，写TCCR2将引起TCR2UB置位。当TCCR2从暂存寄存器更新完毕后 TCR2UB 由硬件清零。 TCR2UB 为 0 表明 TCCR2 可以写入新值了。

如果在更新忙标志置位的时候写上述任何一个寄存器都将引起数据的破坏，并引发不必要的中断。

读取 TCNT2、 OCR2 和 TCCR2 的机制是不同的。读取 TCNT2 得到的是实际的值，而 OCR2 和 TCCR2 则是从暂存寄存器中读取的。

定时器 / 计数器 2 的异步操作

T/C2 工作于异步模式时要考虑如下几点：

- 警告 : 在同步和异步模式之间的转换有可能造成 TCNT2、OCR2 和 TCCR2 数据的损毁。安全的步骤应该是：
  1. 清零 OCIE2 和 TOIE2 以关闭 T/C2 的中断
  2. 设置 AS2 以选择合适的时钟源
  3. 对 TCNT2、 OCR2 和 TCCR2 写入新的数据
  4. 切换到异步模式：等待 TCN2UB、 OCR2UB 和 TCR2UB 清零
  5. 清除 T/C2 的中断标志
  6. 需要的话使能中断
- 振荡器最好使用 32.768 kHz 手表晶振。给 TOSC1 提供外部时钟，可能会造成 T/C2 工作错误。系统主时钟必须比晶振高 4 倍以上。
- 写TCNT2，OCR2和TCCR2时数据首先送入暂存器，两个TOSC1时钟正跳变后才锁存到对应到的寄存器。在数据从暂存器写入目的寄存器之前不能执行新的数据写入操作。 3 个寄存器具有各自独立的暂存器，因此写 TCNT2 并不会干扰 OCR2 的写操作。异步状态寄存器 ASSR 用来检查数据是否已经写入到目的寄存器。
- 如果要用 T/C2 作为 MCU 省电模式或扩展 Standby 模式的唤醒条件，则在 TCNT2，OCR2A和TCCR2A更新结束之前不能进入这些休眠模式，否则MCU可能会在T/C2设置生效之前进入休眠模式。这对于用 T/C2 的比较匹配中断唤醒 MCU 尤其重要，因为在更新 OCR2 或 TCNT2 时比较匹配是禁止的。如果在更新完成之前 (OCR2UB

为 0)MCU 就进入了休眠模式，那么比较匹配中断永远不会发生， MCU 也永远无法唤醒了。

- 如果要用T/C2作为省电模式或扩展Standby模式的唤醒条件，必须注意重新进入这些休眠模式的过程。中断逻辑需要一个 TOSC1 周期进行复位。如果从唤醒到重新进入休眠的时间小于一个 TOSC1 周期，中断将不再发生，器件也无法唤醒。如果用户怀疑自己程序是否满足这一条件，可以采取如下方法：
  1. 对 TCCR2、 TCNT2 或 OCR2 写入合适的数据
  2. 等待 ASSR 相应的更新忙标志清零
  3. 进入省电模式或扩展 Standby 模式
- 若选择了异步工作模式，T/C2 的 32.768 kHz 振荡器将一直工作，除非进入掉电模式或 Standby 模式。用户应该注意，此振荡器的稳定时间可能长达 1 秒钟。因此，建议用户在器件上电复位，或从掉电 /Standby 模式唤醒时至少等待 1 秒钟后再使用 T/C2。同时，由于启动过程时钟的不稳定性，唤醒时所有的 T/C2 寄存器的内容都可能不正确，不论使用的是晶体还是外部时钟信号。用户必须重新给这些寄存器赋值。
- 使用异步时钟时省电模式或扩展 Standby 模式的唤醒过程：中断条件满足后，在下一个定时器时钟唤醒过程启动。也就是说，在处理器可以读取计数器的数值之前计数器至少又累加了一个时钟。唤醒后 MCU 停止 4 个时钟，接着执行中断服务程序。中断服务程序结束之后开始执行 SLEEP 语句之后的程序。
- 从省电模式唤醒之后的短时间内读取 TCNT2 可能返回不正确的数据。因为 TCNT2 是由异步的 TOSC 时钟驱动的，而读取 TCNT2 必须通过一个与内部 I/O 时钟同步的寄存器来完成。同步发生于每个 TOSC1 的上升沿。从省电模式唤醒后 I/O 时钟重新激活，而读到的 TCNT2 数值为进入休眠模式前的值，直到下一个 TOSC1 上升沿的到来。从省电模式唤醒时 TOSC1 的相位是完全不可预测的，而且与唤醒时间有关。因此，读取 TCNT2 的推荐序列为：
  1. 写一个任意数值到 OCR2 或 TCCR2
  2. 等待相应的更新忙标志清零
  3. 读 TCNT2
- 在异步模式下，中断标志的同步需要 3 个处理器周期加一个定时器周期。在处理器可以读取引起中断标志置位的计数器数值之前计数器至少又累加了一个时钟。输出比较引脚的变化与定时器时钟同步，而不是处理器时钟。

定时器 / 计数器中断屏蔽寄存器 - TIMSK

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – OCIE2: T/C2 输出比较匹配中断使能**

当 OCIE2 和状态寄存器的全局中断使能位 I 都为 "1" 时，T/C2 的输出比较匹配 A 中断使能。当 T/C2 的比较匹配发生，即 TIFR 中的 OCF2 置位时，中断服务程序得以执行。

- **Bit 6 – TOIE2: T/C2 溢出中断使能**

当 TOIE2 和状态寄存器的全局中断使能位 I 都为 "1" 时，T/C2 的溢出中断使能。当 T/C2 发生溢出，即 TIFR 中的 TOV2 位置位时，中断服务程序得以执行。

定时器 / 计数器 中断标志寄存器 - TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

初始值	0	0	0	0	0	0	0	0

- **Bit 7 – OCF2: 输出比较标志 2**

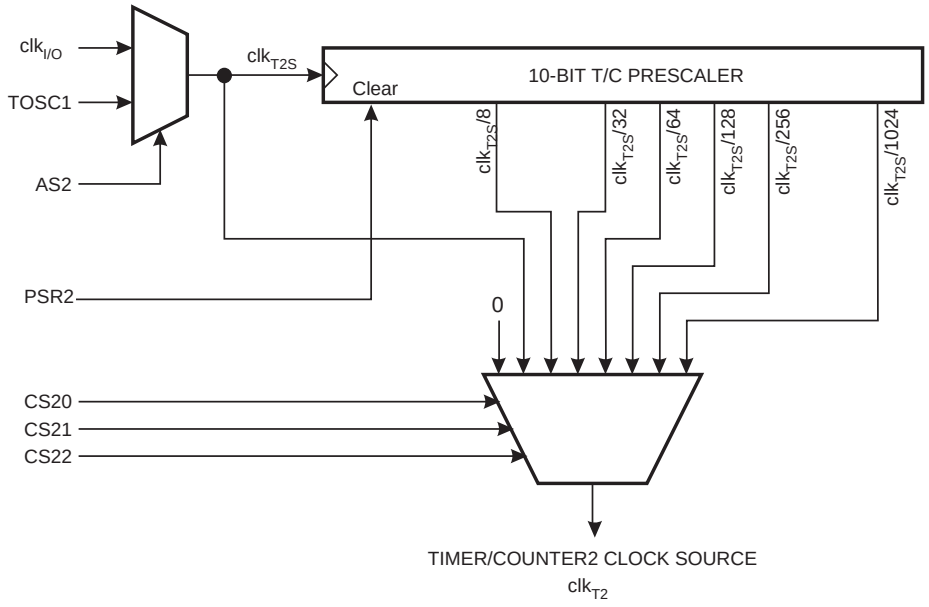
当 T/C2 与 OCR2(输出比较寄存器 2) 的值匹配时， OCF2 置位。此位在中断服务程序里硬件清零，也可以通过对其写 1 来清零。当 SREG 中的位 I、OCIE2 和 OCF2 都置位时，中断服务程序得到执行。

- **Bit 6 – TOV2: T/C2 溢出标志**

当 T/C2 溢出时， TOV2 置位。执行相应的中断服务程序时此位硬件清零。此外， TOV2 也可以通过写 1 来清零。当 SREG 中的位 I、 TOIE2 和 TOV2 都置位时，中断服务程序得到执行。在 PWM 模式中，当 T/C2 在 0x00 改变记数方向时， TOV2 置位。

定时器 / 计数器预分频器

Figure 64. T/C2 的预分频器



T/C2 预分频器的输入时钟称为 clk_{T2S}。缺省地， clk_{T2S} 与系统主时钟 $clk_{I/O}$ 连接。若置位 ASSR 的 AS2 , T/C2 将由引脚 TOSC1 异步驱动 , 使得 T/C2 可以作为一个实时时钟 RTC。此时 TOSC1 和 TOSC2 从端口 C 脱离，引脚上可外接一个时钟晶振 (内部振荡器针对 32.768 kHz 的钟表晶体进行了优化)。不推荐在 TOSC1 上直接施加外部时钟信号。

T/C2 的可能预分频选项有：clk_{T2S}/8、 clk_{T2S}/32、 clk_{T2S}/64、 clk_{T2S}/128、 clk_{T2S}/256 和 clk_{T2S}/1024。此外还可以选择 clk_{T2S} 和 0 (停止工作)。置位 SFIOR 寄存器的 PSR2 将复位预分频器，从而允许用户从可预测的预分频器开始工作。

特殊功能 IO 寄存器 - SFIOR

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
读 / 写	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 1 – PSR2: 预分频复位 T/C2**

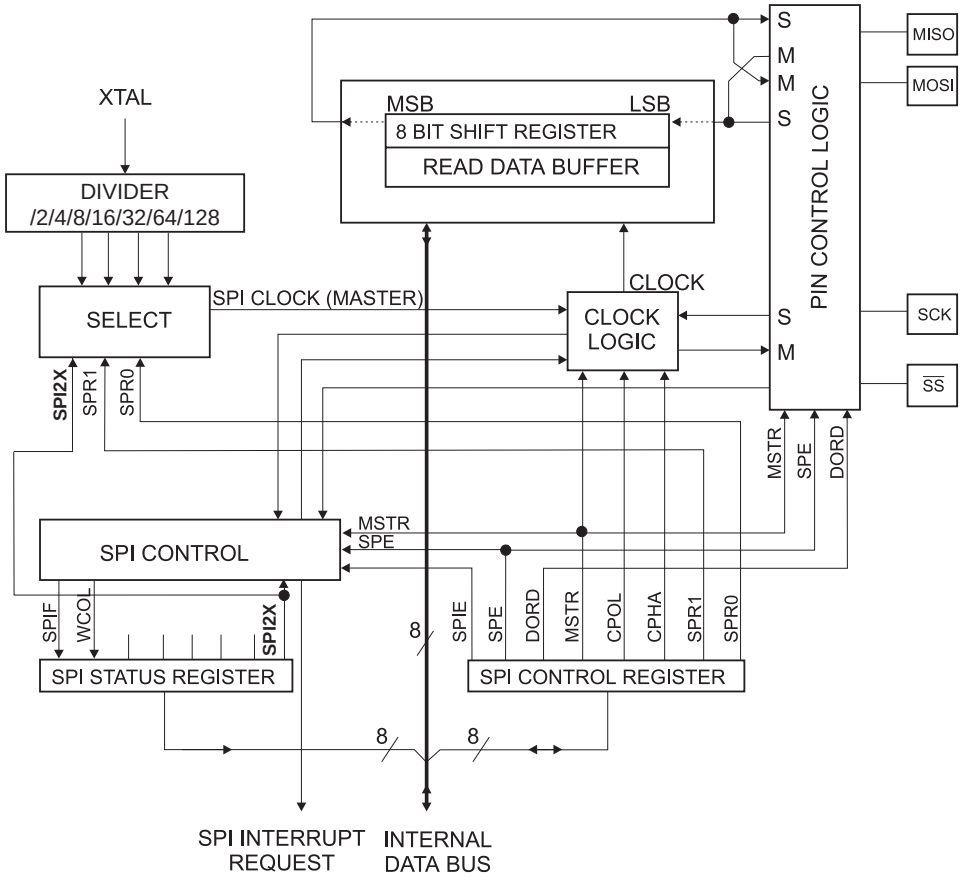
当该位置 1， T/C2 预分频器复位。操作完成后，该位被硬件清零。该位写 0 无效。若内部 CPU 时钟作为 T/C2 时钟，该位读为 0。当 T/C2 工作在异步模式时，直到预分频器复位该位保持为 1。

串行外设接口 - SPI

串行外设接口 SPI 允许 ATmega32 和外设或其他 AVR 器件进行高速的同步数据传输。ATmega32 SPI 的特点如下：

- **全双工，3 线同步数据传输**
- **主机或从机操作**
- **LSB 首先发送或 MSB 首先发送**
- **7 种可编程的比特率**
- **传输结束中断标志**
- **写碰撞标志检测**
- **可以从闲置模式唤醒**
- **作为主机时具有倍速模式 (CK/2)**

Figure 65. SPI 方框图 (1)



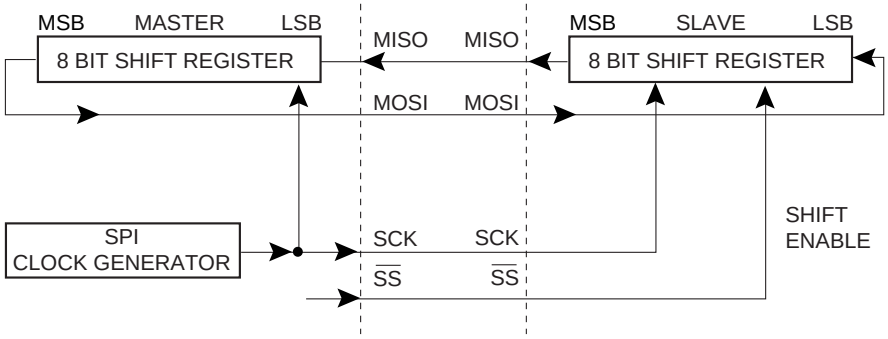
Note: 1. SPI 的引脚排列请参见 P2 Figure 1 与 P55 Table 25 。

主机和从机之间的 SPI 连接如 Figure 66 所示。系统包括两个移位寄存器和一个主机时钟发生器。通过将需要的从机的 $\overline{SS}$ 引脚拉低，主机启动一次通讯过程。主机和从机将需要发送的数据放入相应的移位寄存器。主机在 SCK 引脚上产生时钟脉冲以交换数据。主机的数据从主机的 MOSI 移出，从从机的 MOSI 移入；从机的数据由从机的 MISO 移出，从主机的 MISO 移入。主机通过将从机的 $\overline{SS}$ 拉高实现与从机的同步。

配置为 SPI 主机时，SPI 接口不自动控制 $\overline{SS}$ 引脚，必须由用户软件来处理。对 SPI 数据寄存器写入数据即启动 SPI 时钟，将 8 比特的数据移入从机。传输结束后 SPI 时钟停止，传输结束标志 SPIF 置位。如果此时 SPCR 寄存器的 SPI 中断使能位 SPIE 置位，中断就会发生。主机可以继续往 SPDR 写入数据以移位到从机中去，或者是将从机的 $\overline{SS}$ 拉高以说明数据包发送完成。最后进来的数据将一直保存于缓冲寄存器里。

配置为从机时，只要 $\overline{SS}$ 为高，SPI 接口将一直保持睡眠状态，并保持 MISO 为三态。在这个状态下软件可以更新 SPI 数据寄存器 SPDR 的内容。即使此时 SCK 引脚有输入时钟，SPDR 的数据也不会移出，直至 $\overline{SS}$ 被拉低。一个字节完全移出之后，传输结束标志 SPIF 置位。如果此时 SPCR 寄存器的 SPI 中断使能位 SPIE 置位，就会产生中断请求。在读取移入的数据之前从机可以继续往 SPDR 写入数据。最后进来的数据将一直保存于缓冲寄存器里。

Figure 66. SPI 主机 - 从机的互连



SPI 系统的发送方向只有一个缓冲器，而在接收方向有两个缓冲器。也就是说，在发送时一定要等到移位过程全部结束后才能对 SPI 数据寄存器执行写操作。而在接收数据时，需要在下一个字符移位过程结束之前通过访问 SPI 数据寄存器读取当前接收到的字符。否则第一个字节将丢失。

工作于 SPI 从机模式时，控制逻辑对 SCK 引脚的输入信号进行采样。为了保证对时钟信号的正确采样，SPI 时钟不能超过 $f_{osc}/4$

SPI 使能后，MOSI、MISO、SCK 和 $\overline{SS}$ 引脚的数据方向将按照 Table 55 所示自动进行配置。更多自动重载信息请参考 P52 “ 端口的第二功能 ” 。

Table 55. SPI 引脚重载

引脚	方向，SPI 主机	方向，SPI 从机
MOSI	用户定义	输入
MISO	输入	用户定义
SCK	用户定义	输入
$\overline{SS}$	用户定义	输入

Note: 请参考 P55 “ 端口 B 的第二功能 ” 以了解如何定义由用户定义的 SPI 引脚。

下面的例程说明如何将 SPI 初始化为主机，以及如何进行简单的数据发送。例子中 DDR_SPI 必须由实际的数据方向寄存器代替；DD_MOSI、DD_MISO 和 DD_SCK 必须由

实际的数据方向代替。比如说，MOSI 为 PB5 引脚，则 DD_MOSI 要用 DDB5 取代，DDR_SPI 则用 DDRB 取代。

汇编代码例程 (1)

```
SPI_MasterInit:
  ; 设置 MOSI 和 SCK 为输出，其他为输入
  ldi   r17,(1<<DD_MOSI)|(1<<DD_SCK)
  out   DDR_SPI,r17
  ; 使能 SPI 主机模式，设置时钟速率为 fck/16
  ldi   r17,(1<<SPE)|(1<<MSTR)|(1<<SPR0)
  out   SPCR,r17
  ret

SPI_MasterTransmit:
  ; 启动数据传输 (r16)
  out   SPDR,r16
Wait_Transmit:
  ; 等待传输结束
  sbis  SPSR,SPIF
  rjmp  Wait_Transmit
  ret
```

C 代码例程 (1)

```
void SPI_MasterInit(void)
{
  /* 设置 MOSI 和 SCK 为输出，其他为输入 */
  DDR_SPI = (1<<DD_MOSI)|(1<<DD_SCK);
  /* 使能 SPI 主机模式，设置时钟速率为 fck/16 */
  SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
}

void SPI_MasterTransmit(char cData)
{
  /* 启动数据传输 */
  SPDR = cData;
  /* 等待传输结束 */
  while(!(SPSR & (1<<SPIF)))
    ;
}
```

Note: 1. 程序假定已经包含了正确的头文件。

下面的例子说明如何将 SPI 初始化为从机，以及如何进行简单的数据接收。

汇编代码例程 (1)

```
SPI_SlaveInit:
  ; 设置 MISO 为输出，其他为输入
  ldi   r17,(1<<DD_MISO)
  out   DDR_SPI,r17
  ; 使能 SPI
  ldi   r17,(1<<SPE)
  out   SPCR,r17
  ret

SPI_SlaveReceive:
  ; 等待接收结束
  sbis  SPSR,SPIF
  rjmp  SPI_SlaveReceive
  ; 读取接收到的数据，然后返回
  in    r16,SPDR
  ret
```

C 代码例程 (1)

```
void SPI_SlaveInit(void)
{
  /* 设置 MISO 为输出，其他为输入 */
  DDR_SPI = (1<<DD_MISO);
  /* 使能 SPI */
  SPCR = (1<<SPE);
}

char SPI_SlaveReceive(void)
{
  /* 等待接收结束 */
  while(!(SPSR & (1<<SPIF)))
    ;
  /* 返回数据 */
  return SPDR;
}
```

Note: 1. 例程假定已经包含了正确的头文件。

$\overline{SS}$ 引脚的功能

从机模式

当 SPI 配置为从机时，从机选择引脚 $\overline{SS}$ 总是为输入。$\overline{SS}$ 为低将激活 SPI 接口，MISO 成为输出（用户必须进行相应的端口配置）引脚，其他引脚成为输入引脚。当 $\overline{SS}$ 为高时所有的引脚成为输入，SPI 逻辑复位，不再接收数据。

$\overline{SS}$ 引脚对于数据包/字节的同步非常有用，可以使从机的位计数器与主机的时钟发生器同步。当 $\overline{SS}$ 拉高时 SPI 从机立即复位接收和发送逻辑，并丢弃移位寄存器里不完整的数据。

主机模式

当 SPI 配置为主机时 (MSTR 的 SPCR 置位)，用户可以决定 $\overline{SS}$ 引脚的方向。

若 $\overline{SS}$ 配置为输出，则此引脚可以用作普通的 I/O 口而不影响 SPI 系统。典型应用是用来驱动从机的 $\overline{SS}$ 引脚。

如果 $\overline{SS}$ 配置为输入，必须保持为高以保证 SPI 的正常工作。若系统配置为主机，$\overline{SS}$ 为输入，但被外设拉低，则 SPI 系统会将此低电平解释为有一个外部主机将自己选择为从机。为了防止总线冲突，SPI 系统将实现如下动作：

1. 清零 SPCR 的 MSTR 位，使 SPI 成为从机，从而 MOSI 和 SCK 变为输入。
2. SPSR 的 SPIF 置位。若 SPI 中断和全局中断开放，则中断服务程序将得到执行。

因此，使用中断方式处理 SPI 主机的数据传输，并且存在 $\overline{SS}$ 被拉低的可能性时，中断服务程序应该检查 MSTR 是否为 "1”。若被清零，用户必须将其置位，以重新使能 SPI 主机模式。

SPI 控制寄存器 - SPCR

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – SPIE: SPI 中断使能**

置位后，只要 SPSR 寄存器的 SPIF 和 SREG 寄存器的全局中断使能位置位，就会引发 SPI 中断。

- **Bit 6 – SPE: SPI 使能**

SPE 置位将使能 SPI。进行任何 SPI 操作之前必须置位 SPE。

- **Bit 5 – DORD: 数据次序**

DORD 置位时数据的 LSB 首先发送；否则数据的 MSB 首先发送。

- **Bit 4 – MSTR: 主 / 从选择**

MSTR 置位时选择主机模式，否则为从机。如果 MSTR 为 "1”，$\overline{SS}$ 配置为输入，但被拉低，则 MSTR 被清零，寄存器 SPSR 的 SPIF 置位。用户必须重新设置 MSTR 进入主机模式。

- **Bit 3 – CPOL: 时钟极性**

CPOL 置位表示空闲时 SCK 为高电平；否则空闲时 SCK 为低电平。请参考 Figure 67 与 Figure 68。CPOL 功能总结如下：

Table 56. CPOL 功能

CPOL	起始沿	结束沿
0	上升沿	下降沿
1	下降沿	上升沿

- **Bit 2 – CPHA: 时钟相位**

CPHA 决定数据是在 SCK 的起始沿采样还是在 SCK 的结束沿采样。请参考 Figure 67 与 Figure 68 。

Table 57. CPHA 功能

CPHA	起始沿	结束沿
0	采样	设置
1	设置	采样

- **Bits 1, 0 – SPR1, SPR0: SPI 时钟速率选择 1 与 0**

确定主机的 SCK 速率。SPR1 和 SPR0 对从机没有影响。SCK 和振荡器的时钟频率 f_{osc} 关系如下表所示：

Table 58. SCK 和振荡器频率的关系

SPI2X	SPR1	SPR0	SCK 频率
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	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

SPI 状态寄存 - SPSR

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
读 / 写	R	R	R	R	R	R	R	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – SPIF: SPI 中断标志**

串行发送结束后，SPIF 置位。若此时寄存器 SPCR 的 SPIE 和全局中断使能位置位，SPI 中断即产生。如果 SPI 为主机， $\overline{SS}$ 配置为输入，且被拉低， SPIF 也将置位。进入中断服务程序后SPIF自动清零。或者可以通过先读SPSR，紧接着访问SPDR来对SPIF清零。

- **Bit 6 – WCOL: 写冲突标志**

在发送当中对 SPI 数据寄存器 SPDR 写数据将置位 WCOL。WCOL 可以通过先读 SPSR，紧接着访问 SPDR 来清零。

- **Bit 5..1 – Res: 保留**

保留位，读操作返回值为零。

- **Bit 0 – SPI2X: SPI 倍速**

置位后 SPI 的速度加倍。若为主机 (见 Table 58)，则 SCK 频率可达 CPU 频率的一半。若为从机，只能保证 $f_{osc}/4$。

ATmega32 的 SPI 接口同时还用来实现程序和 EEPROM 的下载和上载。请参见 P253 的 SPI 串行编程和校验。

SPI 数据寄存器 - SPDR

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	SPDR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	X	X	X	X	X	X	X	X	未定义

SPI 数据寄存器为读/写寄存器，用来在寄存器文件和SPI移位寄存器之间传输数据。写寄存器将启动数据传输，读寄存器将读取寄存器的接收缓冲器。

数据模式

相对于串行数据，SCK 的相位和极性有 4 种组合。CPHA 和 CPOL 控制组合的方式。SPI 数据传输格式见 Figure 67 与 Figure 68。每一位数据的移出和移入发生于 SCK 不同的信号跳变沿，以保证有足够的时间使数据稳定。这个过程在 Table 56 与 Table 57 有清楚的说明：

Table 59. CPOL 与 CPHA 功能

	起始沿	结束沿	SPI 模式
CPOL = 0, CPHA = 0	采样 (上升沿)	设置 (下降沿)	0
CPOL = 0, CPHA = 1	设置 (上升沿)	采样 (下降沿)	1
CPOL = 1, CPHA = 0	采样 (下降沿)	设置 (上升沿)	2
CPOL = 1, CPHA = 1	设置 (下降沿)	采样 (上升沿)	3

Figure 67. CPHA = 0 时 SPI 的传输格式

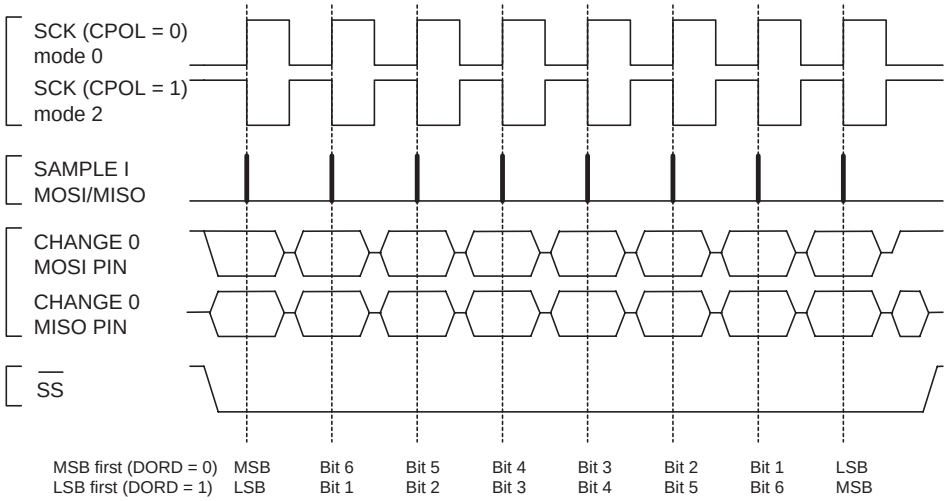
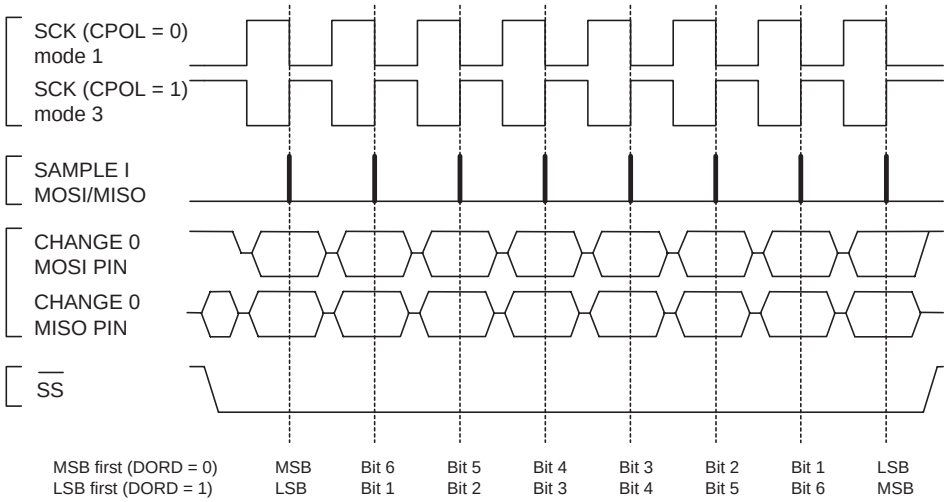


Figure 68. CPHA = 1 时 SPI 的传输格式



USART

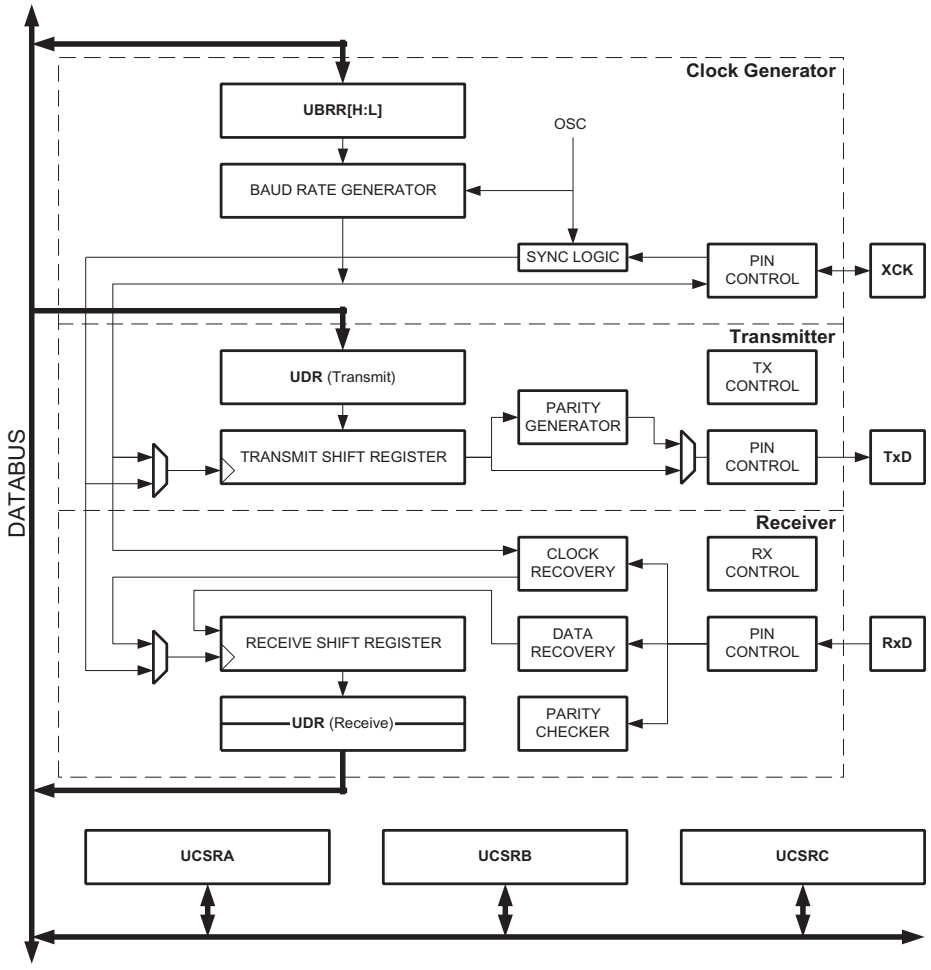
通用同步和异步串行接收器和转发器 (USART) 是一个高度灵活的串行通讯设备。主要特点为：

- **全双工操作 (独立的串行接收和发送寄存器)**
- **异步或同步操作**
- **主机或从机提供时钟的同步操作**
- **高精度的波特率发生器**
- **支持 5, 6, 7, 8, 或 9 个数据位和 1 个或 2 个停止位**
- **硬件支持的奇偶校验操作**
- **数据过速检测**
- **帧错误检测**
- **噪声滤波，包括错误的起始位检测，以及数字低通滤波器**
- **三个独立的中断：发送结束中断，发送数据寄存器空中断，以及接收结束中断**
- **多处理器通讯模式**
- **倍速异步通讯模式**

综述

Figure 69 为 USART 的简化框图。 CPU 可以访问的 I/O 寄存器和 I/O 引脚以粗体表示。

Figure 69. USART 方框图 (1)



Note: 1. 请参考 P2 Figure 1 、 P61 Table 33 与 P57 Table 27 了解 USART 的引脚分布。

虚线框将 USART 分为了三个主要部分：时钟发生器，发送器和接收器。控制寄存器由三个单元共享。时钟发生器包含同步逻辑，通过它将波特率发生器及为从机同步操作所使用的外部输入时钟同步起来。XCK (发送器时钟) 引脚只用于同步传输模式。发送器包括一

个写缓冲器，串行移位寄存器，奇偶发生器以及处理不同的帧格式所需的控制逻辑。写缓冲器可以保持连续发送数据而不会在数据帧之间引入延迟。由于接收器具有时钟和数据恢复单元，它是 USART 模块中最复杂的。恢复单元用于异步数据的接收。除了恢复单元，接收器还包括奇偶校验，控制逻辑，移位寄存器和一个两级接收缓冲器 UDR。接收器支持与发送器相同的帧格式，而且可以检测帧错误，数据过速和奇偶校验错误。

AVR USART 和 AVR UART 兼容性

USART 在如下方面与 AVR UART 完全兼容：

- 所有 USART 寄存器的位定义
- 波特率发生器
- 发送器操作
- 发送缓冲器的功能
- 接收器操作

然而，接收器缓冲器有两个方面的改进，在某些特殊情况下会影响兼容性：

- 增加了一个缓冲器。两个缓冲器的操作好象是一个循环的 FIFO。因此对于每个接收到的数据只能读一次！更重要的是错误标志 FE 和 DOR，以及第 9 个数据位 RXB8 与数据一起存放于接收缓冲器。因此必须在读取 UDR 寄存器之前访问状态标志位。否则将丢失错误状态。
- 接收移位寄存器可以作为第三级缓冲。在两个缓冲器都没有空的时候，数据可以保存于串行移位寄存器之中 (参见 Figure 69) ，直到检测到新的起始位。从而增强了 USART 抵抗数据过速 (DOR) 的能力。

下面的控制位的名称做了改动，但其功能和在寄存器中的位置并没有改变：

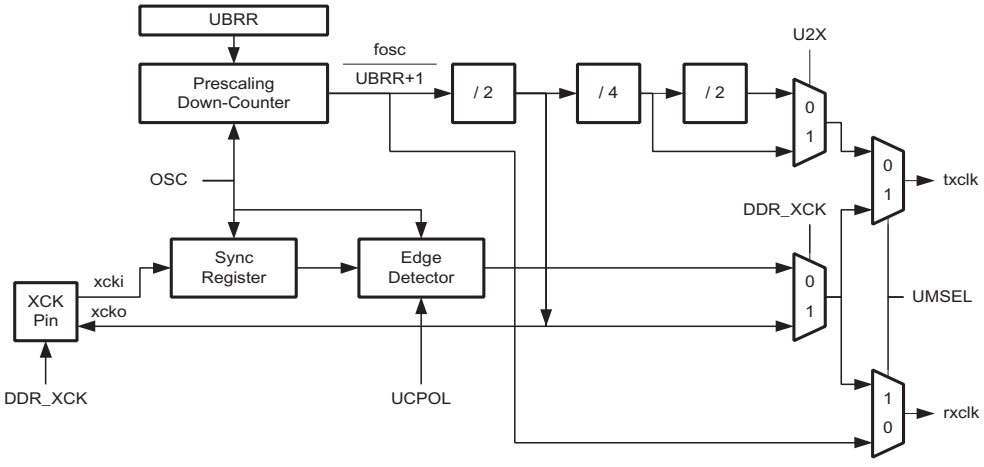
- CHR9 改为 UCSZ2
- OR 改为 DOR

时钟产生

时钟产生逻辑为发送器和接收器产生基础时钟。 USART 支持 4 种模式的时钟：正常的异步模式，倍速的异步模式，主机同步模式，以及从机同步模式。 USART 控制位 UMSEL 和状态寄存器 C (UCSRC) 用于选择异步模式和同步模式。倍速模式 (只适用于异步模式) 受控于 UCSRA 寄存器的 U2X。使用同步模式 (UMSEL = 1) 时 ,XCK 的数据方向寄存器 (DDR_XCK)决定时钟源是由内部产生(主机模式)还是由外部生产(从机模式)。仅在同步模式下 XCK 有效。

Figure 70 为时钟产生逻辑的框图。

Figure 70. 时钟产生逻辑框图



信号说明：

txclk 发送器时钟 (内部信号)

rxclk 接收器基础时钟 (内部信号)

xcki XCK 引脚输入 (内部信号) , 用于同步从机操作

xcko 输出到 XCK 引脚的时钟 (内部信号) , 用于同步主机操作

fosc XTAL 频率 (系统时钟)

片内时钟产生 - 波特率发生器

内部时钟用于异步模式与同步主机模式，请参见 Figure 70。

USART 的波特率寄存器 UBRR 和降序计数器相连接 , 一起构成可编程的预分频器或波特率发生器。降序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 f_{osc}/(UBRR+1)。发生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于接收器与数据恢复单元。数据恢复单元使用了一个有2、8或16个状态的状态机 , 具体状态数由UMSEL、U2X 与 DDR_XCK 位设定的工作模式决定。

Table 60给出了计算波特率(位/秒)以及计算每一种使用内部时钟源工作模式的UBRR值的公式。

Table 60. 波特率计算公式

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

Note: 1. 波特率定义为每秒的位传输速度 (bps)。

BAUD 波特率 (bps)

f_{osc} 系统时钟频率

UBRR UBRRH 与 UBRRL 的数值 (0-4095)

Table 68 给出了在某些系统时钟频率下对应的 UBRR 数值。

倍速工作模式 (U2X)

通过设定 UCSRA 寄存器的 U2X 可以使传输速率加倍。该位只对异步工作模式有效。当工作在同步模式时，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。此时接收器只使用一半的采样数对数据进行采样及时钟恢复，因此在该模式下需要更精确的系统时钟与更精确的波特率设置。发送器则没有这个要求。

外部时钟

同步从机操作模式由外部时钟驱动，如 Figure 70 所示。

输入到 XCK 引脚的外部时钟由同步寄存器进行采样，用以提高稳定性。同步寄存器的输出通过一个边沿检测器，然后应用于发送器与接收器。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：

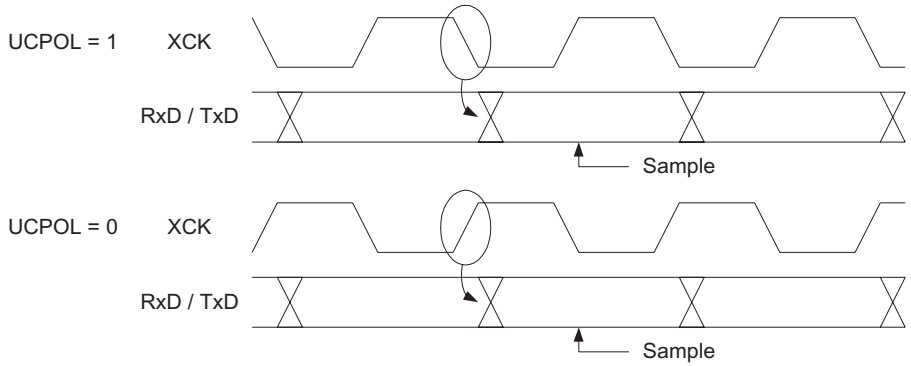
$$f_{XCK} < \frac{f_{OSC}}{4}$$

要注意 f_{osc} 由系统时钟的稳定性决定，为了防止因频率漂移而丢失数据，建议保留足够的裕量。

同步时钟操作

使用同步模式时 (UMSEL = 1)XCK 引脚被用于时钟输入 (从机模式) 或时钟输出 (主机模式)。时钟的边沿、数据的采样与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。

Figure 71. 同步模式时的 XCK 时序 .



UCRSC 寄存器的 UCPOL 位确定使用 XCK 时钟的哪个边沿对数据进行采样和改变输出数据。如 Figure 71 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。

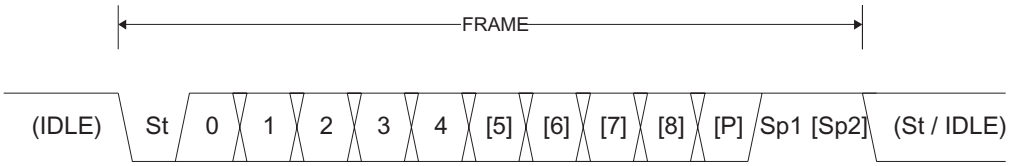
帧格式

串行数据帧由数据字加上同步位 (开始位与停止位) 以及用于纠错的奇偶校验位构成。USART 接受以下 30 种组合的数据帧格式：

- 1 个起始位
- 5、6、7、8 或 9 个数据位
- 无校验位、奇校验或偶校验位
- 1 或 2 个停止位

数据帧以起始位开始；紧接着是数据字的最低位，数据字最多可以有 9 个数据位，以数据的最高位结束。如果使能了校验位，校验位将紧接着数据位，最后是结束位。当一个完整的数据帧传输后，可以立即传输下一个新的数据帧，或使传输线处于空闲状态。 Figure 72 所示为可能的数据帧结构组合。括号中的位是可选的。

Figure 72. 帧格式



St 起始位，总是为低电平

(n) 数据位 (0 ~ 8)

P 校验位，可以为奇校验或偶校验

Sp 停止位，总是为高电平

IDLE 通讯线上没有数据传输 (RxD 或 TxD)，线路空闲时必须为高电平

数据帧的结构由 UCSRB 和 UCSRC 寄存器中的 UCSZ2:0、UPM1:0、 USBS 设定。接收与发送使用相同的设置。设置的任何改变都可能破坏正在进行的数据传送与接收。

USART的字长位UCSZ2:0确定了数据帧的数据位数；校验模式位UPM1:0用于使能与决定校验的类型； USBS 位设置帧有一位或两位结束位。接收器忽略第二个停止位，因此帧错误 (FE) 只在第一个结束位为 "0” 时被检测到。

校验位的计算

校验位的计算是对数据的各个位进行异或运算。如果选择了奇校验，则异或结果还需要取反。校验位与数据位的关系如下：

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

P_{even} 偶校验结果

P_{odd} 奇校验位结果

d_n 第 n 个数据位

校验位处于最后一个数据位与第一个停止位之间。

USART 的初始化

进行通信之前首先要对 USART 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，以及根据需要使能接收器或发送器。对于中断驱动的 USART 操作，在初始化时首先要清零全局中断标志位 (全局中断被屏蔽)。

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接收缓冲器中是否还有数据未读出。在每次发送数据之前 (在写发送数据寄存器 UDR 前)TXC 标志位必须清零。

以下是 USART 初始化程序示例。例程采用了轮询 (中断被禁用) 的异步操作，而且帧结构是固定的。波特率作为函数参数给出。在汇编程序里波特率参数保存于寄存器 r17:r16。当写入 UCSRC 寄存器时，由于 UBRRH 与 UCSRC 共用 I/O 地址，URSEL 位 (MSB) 必须置位。

汇编代码例程 (1)

```
USART_Init:
  ; 设置波特率
  out   UBRRH, r17
  out   UBRRL, r16
  ; 接收器与发送器使能
  ldi   r16, (1<<RXEN)|(1<<TXEN)
  out   UCSRB,r16
  ; 设置帧格式： 8 个数据位， 2 个停止位
  ldi   r16, (1<<URSEL)|(1<<USBS)|(3<<UCSZ0)
  out   UCSRC,r16
  ret
```

C 代码例程 (1)

```
void USART_Init( unsigned int baud )
{
  /* 设置波特率*/
  UBRRH = (unsigned char)(baud>>8);
  UBRRL = (unsigned char)baud;
  /* 接收器与发送器使能*/
  UCSRB = (1<<RXEN)|(1<<TXEN);
  /* 设置帧格式： 8 个数据位， 2 个停止位*/
  UCSRC = (1<<URSEL)|(1<<USBS)|(3<<UCSZ0);
}
```

Note: 1. 本代码假定已经包含了合适的头文件。

更高级的初始化程序可将帧格式作为参数、禁止中断等等。然而许多应用程序使用固定的波特率与控制寄存器。此时初始化代码可以直接放在主程序中，或与其它 I/O 模块的初始化代码组合到一起。

数据发送 - USART 发送器

置位 UCSRB 寄存器的发送允许位 TXEN 将使能 USART 的数据发送。使能后 TxD 引脚的通用 I/O 功能即被 USART 功能所取代，成为发送器的串行输出引脚。发送数据之前要设置好波特率、工作模式与帧结构。如果使用同步发送模式，施加于 XCK 引脚上的时钟信号即为数据发送的时钟。

发送 5 到 8 位数据位的帧

将需要发送的数据加载到发送缓存器将启动数据发送。加载过程即为 CPU 对 UDR 寄存器的写操作。当移位寄存器可以发送新一帧数据时，缓冲的数据将转移到移位寄存器。当移位寄存器处于空闲状态（没有正在进行的数据传输），或前一帧数据的最后一个停止位传送结束，它将加载新的数据。一旦移位寄存器加载了新的数据，就会按照设定的波特率完成数据的发送。

以下程序给出一个对 UDRE 标志采用轮询方式发送数据的例子。当发送的数据少于 8 位时，写入 UDR 相应位置的高几位将被忽略。当然，执行本段代码之前首先要初始化 USART。在汇编代码中要发送的数据存放于 R16。

汇编代码例程 (1)

```
USART_Transmit:
  ; 等待发送缓冲器为空
  sbis  UCSRA,UDRE
  rjmp  USART_Transmit
  ; 将数据放入缓冲器，发送数据
  out   UDR,r16
  ret
```

C 代码例程 (1)

```
void USART_Transmit( unsigned char data )
{
  /* 等待发送缓冲器为空 */
  while ( !( UCSRA & (1<<UDRE)) )
        ;
  /* 将数据放入缓冲器，发送数据 */
  UDR = data;
}
```

Note: 1. 本代码假定已经包含了合适的头文件。

这个程序只是在载入新的要发送的数据前，通过检测 UDRE 标志等待发送缓冲器为空。如果使用了数据寄存器空中断，则数据写入缓冲器的操作在中断程序中进行。

发送 9 位数据位的帧

如果发送 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，然后再将低8位数据写入发送数据寄存器UDR。以下程序给出发送9位数据的数据帧例子。在汇编代码中要发送的数据存放在 R17:R16 寄存器中。

汇编代码例程 (1)

```
USART_Transmit:
  ; 等待发送缓冲器为空
  sbis  UCSRA,UDRE
  rjmp  USART_Transmit
  ; 将第 9 位从 r17 中复制到 TXB8
  cbi   UCSRB,TXB8
  sbrc  r17,0
  sbi   UCSRB,TXB8
  ; 将低 8 位数据放入缓冲器，发送数据
  out   UDR,r16
  ret
```

C 代码例程 (1)

```
void USART_Transmit( unsigned int data )
{
  /* 等待发送缓冲器为空 */
  while ( !( UCSRA & (1<<UDRE)) )
        ;
  /* 将第 9 位复制到 TXB8 */
  UCSRB &= ~(1<<TXB8);
  if ( data & 0x0100 )
    UCSRB |= (1<<TXB8);
  /* 将数据放入缓冲器，发送数据 */
  UDR = data;
}
```

Note: 1. 这些函数均为通用函数。如果 UCSRB 的内容在应用中是固定的，函数可以进一步优化。例如，初始化后只使用 UCSRB 寄存器的 TXB8 位。

第 9 位数据在多机通信中用于表示地址帧，在同步通信中可以用于协议处理。

传送标志位与中断

USART 发送器有两个标志位：USART 数据寄存器空标志 UDRE 及传输结束标志 TXC，两个标志位都可以产生中断。

数据寄存器空 UDRE 标志位表示发送缓冲器是否可以接受一个新的数据。该位在发送缓冲器空时被置 "1"；当发送缓冲器包含需要发送的数据时清零。为与将来的器件兼容，写 UCSRA 寄存器时该位要写 "0"。

当 UCSRB 寄存器中的数据寄存器空中断使能位 UDRIE 为 "1" 时，只要 UDRE 被置位 (且全局中断使能)，就将产生 USART 数据寄存器空中断请求。对寄存器 UDR 执行写操作将清零 UDRE。当采用中断方式的传输数据时，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 以清零 UDRE ；或者是禁止数据寄存器空中断。否则一旦该中断程序结束，一个新的中断将再次产生。

当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，发送结束标志 TXC 置位。TXC 在传送结束中断执行时自动清零，也可在该位写 "1" 来清零。TXC 标志位对于采用如 RS-485 标准的半双工通信接口十分有用。在这些应用里，一旦传送完毕，应用程序必须释放通信总线并进入接收状态。

当 UCSRB 上的发送结束中断使能位 TXCIE 与全局中断使能位均被置为 "1” 时 , 随着 TXC 标志位的置位， USART 发送结束中断将被执行。一旦进入中断服务程序， TXC 标志位即被自动清零，中断处理程序不必执行 TXC 清零操作。

奇偶校验产生电路

奇偶校验产生电路为串行数据帧生成相应的校验位。校验位使能 (UPM1 = 1) 时，发送控制逻辑电路会在数据的最后一位与第一个停止位之间插入奇偶校验位。

禁止发送器

TXEN 清零后，只有等到所有的数据发送完成后发送器才能够真正禁止，即发送移位寄存器与发送缓冲寄存器中没有要传送的数据。发送器禁止后，TxD 引脚恢复其通用 I/O 功能。

数据接收 - USART 接收器

置位 UCSRB 寄存器的接收允许位 (RXEN) 即可启动 USART 接收器。接收器使能后 RxD 的普通引脚功能被 USART 功能所取代，成为接收器的串行输入口。进行数据接收之前首先要设置好波特率、操作模式及帧格式。如果使用同步操作， XCK 引脚上的时钟被用为传输时钟。

以 5 到 8 个数据位的方式接收数据帧

一旦接收器检测到一个有效的起始位，便开始接收数据。起始位后的每一位数据都将以所设定的波特率或 XCK 时钟进行接收，直到收到一帧数据的第一个停止位。接收到的数据被送入接收移位寄存器。第二个停止位会被接收器忽略。接收到第一个停止位后，接收移位寄存器就包含了一个完整的数据帧。这时移位寄存器中的内容将被转移到接收缓冲器中。通过读取 UDR 就可以获得接收缓冲器的内容的。

以下程序给出一个对 RXC 标志采用轮询方式接收数据的例子。当数据帧少于 8 位时，从 UDR 读取的相应的高几位为 0。当然，执行本段代码之前首先要初始化 USART。

汇编代码例子 (1)

```
USART_Receive:
  ; 等待接收数据
  sbis  UCSRA, RXC
  rjmp  USART_Receive
  ; 从缓冲器中获取并返回数据
  in    r16, UDR
  ret
```

C 代码例子 (1)

```
unsigned char USART_Receive( void )
{
  /* 等待接收数据 */
  while ( !(UCSRA & (1<<RXC)) )
        ;
  /* 从缓冲器中获取并返回数据 */
  return UDR;
}
```

Note: 1. 本代码假定已经包含了相应的头文件。

在读缓冲器并返回之前，函数通过检查 RXC 标志来等待数据送入接收缓冲器。

以 9 个数据位的方式接收帧

如果设定了 9 位数据的数据帧 (UCSZ=7)，在从 UDR 读取低 8 位**之前**必须首先读取寄存器 UCSRB 的 RXB8 以获得第 9 位数据。这个规则同样适用于状态标志位 FE、 DOR 及 UPE。状态通过读取UCSRA获得，数据通过UDR获得。读取UDR存储单元会改变接收缓冲器 FIFO 的状态，进而改变同样存储在 FIFO 中的 TXB8、FE、 DOR 及 UPE 位。

接下来的代码示例展示了一个简单的USART接收函数，说明如何处理9位数据及状态位。

汇编代码例程 (1)

```
USART_Receive:
  ; 等待接收数据
  sbis  UCSRA, RXC
  rjmp  USART_Receive
  ; 从缓冲器中获得状态、第 9 位及数据
  in    r18, UCSRA
  in    r17, UCSRB
  in    r16, UDR
  ; 如果出错，返回 -1
  andi  r18,(1<<FE)|(1<<DOR)|(1<<PE)
  breq  USART_ReceiveNoError
  ldi   r17, HIGH(-1)
  ldi   r16, LOW(-1)
USART_ReceiveNoError:
  ; 过滤第 9 位数据，然后返回
  lsr   r17
  andi  r17, 0x01
  ret
```

C 代码例程 (1)

```
unsigned int USART_Receive( void )
{
  unsigned char status, resh, resl;
  /* 等待接收数据*/
  while ( !(UCSRA & (1<<RXC)) )
        ;
  /* 从缓冲器中获得状态、第 9 位及数据*/
  /* from buffer */
  status = UCSRA;
  resh = UCSRB;
  resl = UDR;
  /* 如果出错，返回 -1 */
  if ( status & (1<<FE)|(1<<DOR)|(1<<PE) )
    return -1;
  /* 过滤第 9 位数据，然后返回*/
  resh = (resh >> 1) & 0x01;
  return ((resh << 8) | resl);
}
```

Note: 1. 本代码假定已经包含了相应的头文件。

上述例子在进行任何计算之前将所有的 I/O 寄存器的内容读到寄存器文件中。这种方法优化了对接收缓冲器的利用。它尽可能早地释放了缓冲器以接收新的数据。

接收结束标志及中断

USART 接收器有一个标志用来指明接收器的状态。

接收结束标志 (RXC) 用来说明接收缓冲器中是否有未读出的数据。当接收缓冲器中有未读出的数据时，此位为 1，当接收缓冲器空时为 0(即不包含未读出的数据)。如果接收器被禁止 (RXEN = 0)，接收缓冲器会被刷新，从而使 RXC 清零。

置位 UCSRB 的接收结束中断使能位 (RXCIE) 后，只要 RXC 标志置位 (且全局中断只能) 就会产生 USART 接收结束中断。使用中断方式进行数据接收时，数据接收结束中断服务程序程序必须从 UDR 读取数据以清 RXC 标志，否则只要中断处理程序一结束，一个新的中断就会产生。

接收器错误标志

USART 接收器有三个错误标志 : : 帧错误 (FE)、数据溢出 (DOR) 及奇偶校验错 (UPE)。它们都位于寄存器 UCSRA。错误标志与数据帧一起保存在接收缓冲器中。由于读取 UDR 会改变缓冲器，UCSRA 的内容必须在读接收缓冲器 (UDR) 之前读入。错误标志的另一个同一性是它们都不能通过软件写操作来修改。但是为了保证与将来产品的兼容性，对执行写操作是必须对这些错误标志所在的位置写 "0“。所有的错误标志都不能产生中断。

帧错误标志 (FE) 表明了存储在接收缓冲器中的下一个可读帧的第一个停止位的状态。停止位正确 (为 1) 则 FE 标志为 0，否则 FE 标志为 1。这个标志可用来检测同步丢失、传输中断，也可用于协议处理。UCSRC 中 USBS 位的设置不影响 FE 标志位，因为除了第一位，接收器忽略所有其他的停止位。为了与以后的器件相兼容，写 UCSRA 时这一位必须置 0。

数据溢出标志 (DOR) 表明由于接收缓冲器满造成了数据丢失。当接收缓冲器满 (包含了两个数据)，接收移位寄存器又有数据，若此时检测到一个新的起始位，数据溢出就产生了。DOR 标志位置位即表明在最近一次读取 UDR 和下一次读取 UDR 之间丢失了一个或更多的数据帧。为了与以后的器件相兼容，写 UCSRA 时这一位必须置 0。当数据帧成功地从移位寄存器转入接收缓冲器后，DOR 标志被清零。

奇偶校验错标志 (PE) 指出，接收缓冲器中的下一帧数据在接收时有奇偶错误。如果不使能奇偶校验，那么 UPE 位应清零。为了与以后的器件相兼容，写 UCSRA 时这一位必须置 0。细节请参照 P134 “ 校验位的计算 ” 与 P141 “ 奇偶校验器 ” 。

奇偶校验器

奇偶校验模式位UPM1置位将启动奇偶校验器。校验的模式(偶校验还是奇校验)由UPM0确定。奇偶校验使能后，校验器将计算输入数据的奇偶并把结果与数据帧的奇偶位进行比较。校验结果将与数据和停止位一起存储在接收缓冲器中。这样就可以通过读取奇偶校验错误标志位 (UPE) 来检查接收的帧中是否有奇偶错误。

如果下一个从接收缓冲器中读出的数据有奇偶错误，并且奇偶校验使能 (UPM1 = 1)，则 UPE 置位。直到接收缓冲器 (UDR) 被读取，这一位一直有效。

禁止接收器

与发送器对比，禁止接收器即刻起作用。正在接收的数据将丢失。禁止接收器 (RXEN 清零) 后，接收器将不再占用 RxD 引脚；接收缓冲器 FIFO 也会被刷新。缓冲器中的数据将丢失。

刷新接收缓冲器

禁止接收器时缓冲器 FIFO 被刷新，缓冲器被清空。导致未读出的数据丢失。如果由于出错而必须在正常操作下刷新缓冲器，则需要一直读取 UDR 直到 RXC 标志清零。下面的代码展示了如何刷新接收缓冲器。

汇编代码例程 (1)

```
USART_Flush:
  sbis  UCSRA, RXC
  ret
  in    r16, UDR
  rjmp  USART_Flush
```

C 代码例程 (1)

```
void USART_Flush( void )
{
  unsigned char dummy;
  while ( UCSRA & (1<<RXC) ) dummy = UDR;
}
```

Note: 1. 本代码假定已经包含了相应的头文件。

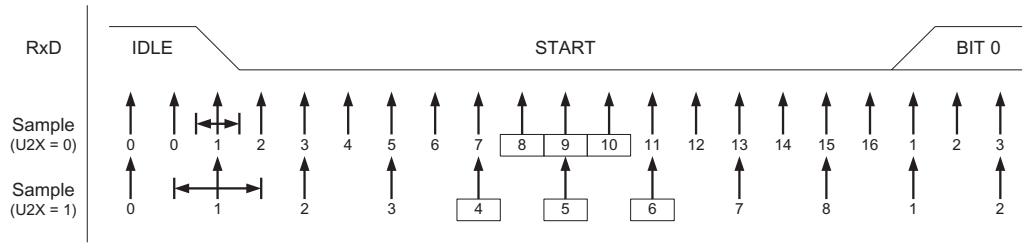
异步数据接收

USART 有一个时钟恢复单元和数据恢复单元用来处理异步数据接收。时钟恢复逻辑用于同步从 RxD 引脚输入的异步串行数据和内部的波特率时钟。数据恢复逻辑采集数据，并通过一低通滤波器过滤所输入的每一位数据，从而提高接收器的抗干扰性能。异步接收的工作范围依赖于内部波特率时钟的精度、帧输入的速率及一帧所包含的位数。

恢复异步时钟

时钟恢复逻辑将输入的串行数据帧与内部时钟同步起来。Figure 73 展示了对输入数据帧起始位的采样过程。普通工作模式下采样率是波特率的 16 倍，倍速工作模式下则为波特率的 8 倍。水平箭头表示由于采样而造成的同步的变化。使用倍速模式 (U2X = 1) 时同步变化时间更长。 RxD 线空闲 (即没有任何通讯活动) 时，采样值为 0。

Figure 73. 起始位采样

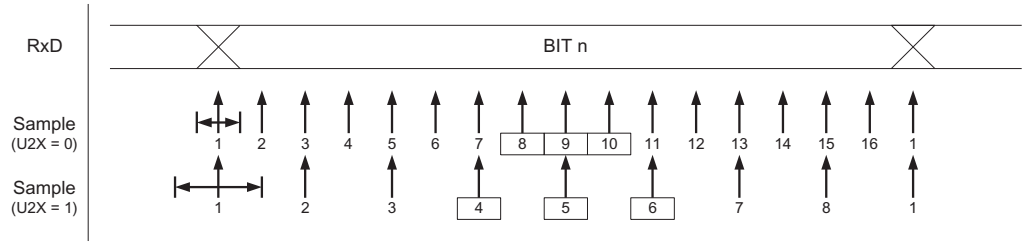


当时钟恢复电路检测到 RxD 线上一个由高 (空闲) 到低 (开始) 的电平跳变时，起始位检测序列即被启动。如图所示，我们用采样 1 表示第一个 0 采样。然后，时钟恢复逻辑用采样 8、 9、 10(普通模式)，或采样 4、 5、 6(倍速模式)，来判断是否接收到一个正确的起始位。如果这三个采样中的两个或更多个是逻辑高电平 (多数表决)，起始位会被视为毛刺噪声而被拒绝接受，接收器等待下一个由高到低的电平转换。如果检测到一个有效的起始位，时钟恢复逻辑即被同步并开始接收数据。每一个起始位都会引发同样的同步过程。

恢复异步数据

接收时钟与起始位同步之后，数据恢复工作可开始了。数据恢复单元使用一个状态机来接收每一个数据位。这个状态机在普通模式下具有 16 个状态，在倍速模式下具有 8 个状态。 Figure 74 演示了对数据位和奇偶位的采样。每个采样点都被赋予了一个数字，这个数字等于数据恢复单元当前的状态序号。

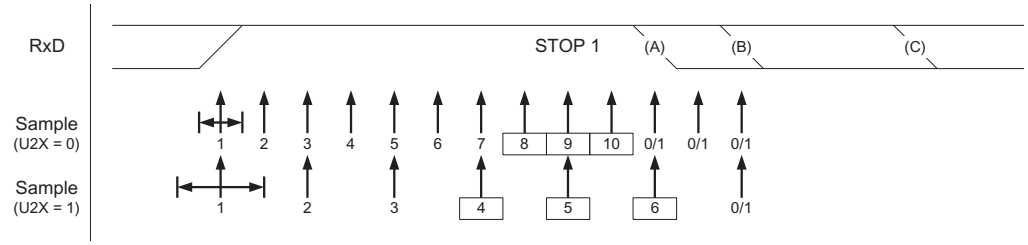
Figure 74. 数据及奇偶位的采样



确定接收到的数据位的逻辑电平的方法为多数表决法。表决对象即为三个在数据位中心获得的采样。为了强调这些采样，图中采样序号被包含小方框中。多数表决是这样工作的：如果有 2 个或所有 3 个采样值都是高电平，那么接收位就为逻辑 1。如果 2 个或所有3个采样值都是低电平，那么接收位就被为逻辑0。对从RxD引脚输入的信号来说，多数表决的作用就象是一个低通滤波。数据恢复过程重复进行，直到接收到一个完整的数据帧。其中也包含了第一个停止位。接收器将忽略其他的停止位。

Figure 75 说明了停止位的采样，以及下一帧信号起始位最早可能出现的情况。

Figure 75. 停止位及下一个起始位采样



多数表决对停止位同样有效。若停止位为逻辑 0，那么帧错误标志 FE 置位。

如果电平再一次出现了从高到低的跳变，说明紧接着上一个数据帧来了新的数据帧。在普通模式中，第一个低电平的采样点可以发生在 Figure 75 的 A 点。在倍速工作模式下第一个低电平采样点必须延迟到 B 点， C 点则为完整停止位的结束位置。对起始位的及早检测将影响接收器的工作范围。

异步工作范围

接收器的工作范围取决于接收到的数据速率及内部波特率之间的不匹配程度。如果发送器以过快或过慢的比特率传输数据帧，或者接收器内部产生的波特率没有相同的频率 (见 Table 61)，那么接收器就无法与起始位同步。

下面的公式可用来计算数据输入速率与内部接收器波特率的比值。

$$R_{slow} = \frac{(D+1)S}{S-1+D\cdot S+S_F}$$

$$R_{fast} = \frac{(D+2)S}{(D+1)S+S_M}$$

D 字符长度及奇偶位长度的总和 (D = 5 到 10 位)

S 每一位的采样数。普通模式下 S = 16，倍速模式下 S = 8

S_F 用于多数表决的第一个采样序号。普通模式下 S_F = 8 ，倍速模式下 S_F = 4

S_M 用于多数表决的中间采样序号。普通模式下 S_M = 9 ，倍速模式下 S_M = 5

R_{slow} 是可接受的、最慢的数据输入速率与接收器波特率的比值； R_{fast} 是可接受的、最快的数据输入速率与接收器波特率的比值。

Table 61 和Table 62列出了容许的最大接收器波特率误差。需要注意的是，普通模式下波特率允许有更大的变化范围。

Table 61. 普通模式下推荐的最大接收器波特率误差范围 (U2X = 0)

D # (数据 + 奇偶位)	R_{slow} (%)	R_{fast}(%)	最大的总误差 (%)	推荐的最大接收器误差 (%)
5	93.20	106.67	+6.67/-6.8	± 3.0
6	94.12	105.79	+5.79/-5.88	± 2.5
7	94.81	105.11	+5.11/-5.19	± 2.0
8	95.36	104.58	+4.58/-4.54	± 2.0
9	95.81	104.14	+4.14/-4.19	± 1.5
10	96.17	103.78	+3.78/-3.83	± 1.5

Table 62. 速率模式下推荐的最大接收器波特率误差范围 (U2X = 1)

D # (数据 + 奇偶位)	R_{slow} (%)	R_{fast} (%)	最大的总误差 (%)	推荐的最大接收器误差 (%)
5	94.12	105.66	+5.66/-5.88	± 2.5
6	94.92	104.92	+4.92/-5.08	± 2.0
7	95.52	104.35	+4.35/-4.48	± 1.5
8	96.00	103.90	+3.90/-4.00	± 1.5
9	96.39	103.53	+3.53/-3.61	± 1.5
10	96.70	103.23	+3.23/-3.30	± 1.0

上述推荐的最大接收波特率误差是在假定接收器和发送器对最大总误差具有同等贡献的前提下得出的。

产生接收器波特率误差的可能原因有两个。首先，接收器系统时钟 (XTAL) 的稳定性于电压范围及工作温度有关。使用晶振来产生系统时钟时一般不会有此问题，但对于谐振器而言，根据谐振器不同的误差容限，系统时钟可能有超过 2% 的偏差。第二个误差的原因就好控制多了。波特率发生器不一定能够通过对系统时钟的分频得到恰好的波特率。此时可以调整 UBRR 值，使得误差低至可以接受。

多处理器通讯模式

置位 UCSRA 的多处理器通信模式位 (MPCM) 可以对 USART 接收器接收到的数据帧进行过滤。那些没有地址信息的帧将被忽略，也不会存入接收缓冲器。在一个多处理器系统中，处理器通过同样的串行总线进行通信，这种过滤有效的减少了需要 CPU 处理的数据帧的数量。MPCM 位的设置不影响发送器的工作，但在使用多处理器通信模式的系统中，它的使用方法会有所不同。

如果接收器所接收的数据帧长度为 5 到 8 位，那么第一个停止位表示这一帧包含的是数据还是地址信息。如果接收器所接收的数据帧长度为 9 位，那么由第 9 位 (RXB8) 来确定是数据还是地址信息。如果确定帧类型的位（第一个停止位或第 9 个数据位）为 1，那么这是地址帧，否则为数据帧。

在多处理器通信模式下，多个从处理器可以从一个主处理器接收数据。首先要通过解码地址帧来确定所寻址的是哪一个处理器。如果寻址到某一个处理器，它将正常接收后续的数据，而其他的从处理器会忽略这些帧直到接收到另一个地址帧。

使用 MPCM

对于一个作为主机的处理器来说，它可以使用 9 位数据帧格式 (UCSZ = 7)。如果传输的是一个地址帧 (TXB8 = 1) 就将第 9 位 (TXB8) 置 1，如果是一个数据帧 (TXB = 0) 就将它清零。在这种帧格式下，从处理器必须工作于 9 位数据帧格式。

下面即为在多处理器通信模式下进行数据交换的步骤：

1. 所有从处理器都工作在多处理器通信模式 (UCSRA 寄存器的 MPCM 置位)。
2. 主处理器发送地址帧后，所有从处理器都会接收并读取此帧。从处理器 UCSRA 寄存器的 RXC 正常置位。
3. 每一个从处理器都会读取 UDR 寄存器的内容已确定自己是否被选中。如果选中，就清零 UCSRA 的 MPCM 位，否则它将等待下一个地址字节的到来，并保持 MPCM 为 1。
4. 被寻址的从处理器将接收所有的数据帧，直到收到一个新的地址帧。而那些保持 MPCM 位为 1 的从处理器将忽略这些数据。
5. 被寻址的处理器接收到最后一个数据帧后，它将置位 MPCM，并等待主处理器发送下一个地址帧。然后第 2 步之后的步骤重复进行。

使用 5 至 8 比特的帧格式是可以的，但是不实际，因为接收器必须在使用 n 和 n+1 帧格式之间进行切换。由于接收器和发送器使用相同的字符长度设置，这种设置使得全双工操作变得很困难。如果使用 5 至 8 比特的帧格式，发送器应该设置两个停止位 (USBS = 1)，其中的第一个停止位被用于判断帧类型。

不要使用读 - 修改 - 写指令 (SBI 和 CBI) 来操作 MPCM 位。 MPCM 和 TXC 标志使用相同的 I/O 单元，使用 SBI 或 CBI 指令可能会不小心将它清零。

访问 UBRRH/ UCSRC 寄存器

UBRRH 与寄存器 UCSRC 共用 I/O 地址。因此访问该地址时需注意以下问题。

写访问

当在该地址执行写访问时，USART 寄存器选择位 (URSEL) 控制被写入的寄存器。若 URSEL 为 0，对 UBRRH 值更新；若 URSEL 为 1，对 UCSRC 设置更新。

下面代码给出如何访问这两个寄存器。

汇编代码例程 (1)

```
...
; 设置UBRRH 为2
ldi r16,0x02
out UBRRH,r16
...
; 设置USBS 与UCSZ1 位为1 ，且其余位为0
ldi r16,(1<<URSEL)|(1<<USBS)|(1<<UCSZ1)
out UCSRC,r16
...
```

C 代码例程 (1)

```
...
/* 设置UBRRH 为2*/
UBRRH = 0x02;
...
/* 设置USBS 与UCSZ1 位为1 ，且其余位为0*/
UCSRC = (1<<URSEL)|(1<<USBS)|(1<<UCSZ1);
...
```

Note: 1. 本代码假定已经包含了相应的头文件。

如例中所示，对两寄存器的写访问不影响共用 I/O 地址。

读访问

对 UBRRH 或 UCSRC 寄存器的读访问则较为复杂。但在大多数应用中，基本不需要读这些寄存器。

读访问由时序控制。一旦返回 UBRRH 寄存器内容则读 I/O 地址。若寄存器地址在前一个系统时钟周期中读入，当前时钟下对寄存器的读入将返回 UCSRC 内容中。注意，读 UCSRC 的时钟序列为自动工作。在读操作中的中断 (例如禁止全局中断) 必须人为控制。

下面代码给出如何读 UCSRC 寄存器内容。

汇编代码例程 (1)

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

C 代码例程 (1)

```
unsigned char USART_ReadUCSRC( void )
{
  unsigned char ucsrc;
  /* 读 UCSRC */
  ucsrc = UBRRH;
  ucsrc = UCSRC;
  return ucsrc;
}
```

Note: 1. 本代码假定已经包含了相应的头文件。

汇编代码在 r16 中返回 UCSRC 值。

对 UBRRH 内容的读操作不是自动完成，且当前一条指令没有访问该寄存器地址时，该寄存器作为普通寄存器使用。

USART 寄存器描述

USART I/O 数据寄存器 - UDR

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (读)
	TXB[7:0]								UDR (写)
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

USART 发送数据缓冲寄存器和 USART 接收数据缓冲寄存器共享相同的 I/O 地址，称为 USART 数据寄存器或 UDR。将数据写入 UDR 时实际操作的是发送数据缓冲器存器 (TXB)，读 UDR 时实际返回的是接收数据缓冲寄存器 (RXB) 的内容。

在 5、6、7 比特字长模式下，未使用的高位被发送器忽略，而接收器则将它们设置为 0。

只有当 UCSRA 寄存器的 UDRE 标志置位后才可以对发送缓冲器进行写操作。如果 UDRE 没有置位，那么写入 UDR 的数据会被 USART 发送器忽略。当数据写入发送缓冲器后，若移位寄存器为空，发送器将把数据加载到发送移位寄存器。然后数据串行地从 TxD 引脚输出。

接收缓冲器包括一个两级 FIFO，一旦接收缓冲器被寻址 FIFO 就会改变它的状态。因此不要对这一存储单元使用读 - 修改 - 写指令 (SBI 和 CBI)。使用位查询指令 (SBIC 和 SBIS) 时也要小心，因为这也有可能改变 FIFO 的状态。

USART 控制和状态寄存器 A - UCSRA

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
读 / 写	R	R/W	R	R	R	R	R/W	R/W	
初始值	0	0	1	0	0	0	0	0	

- **Bit 7 – RXC: USART 接收结束**

接收缓冲器中有未读出的数据时 RXC 置位，否则清零。接收器禁止时，接收缓冲器被刷新，导致 RXC 清零。 RXC 标志可用来产生接收结束中断 (见对 RXCIE 位的描述)。

- **Bit 6 – TXC: USART 发送结束**

发送移位缓冲器中的数据被送出，且当发送缓冲器 (UDR) 为空时 TXC 置位。执行发送结束中断时 TXC 标志自动清零，也可以通过写 1 进行清除操作。TXC 标志可用来产生发送结束中断 (见对 TXCIE 位的描述)。

- **Bit 5 – UDRE: USART 数据寄存器空**

UDRE标志指出发送缓冲器(UDR)是否准备好接收新数据。UDRE为1说明缓冲器为空 ,已准备好进行数据接收。UDRE标志可用来产生数据寄存器空中断(见对UDRIE位的描述)。

复位后 UDRE 置位，表明发送器已经就绪。

- **Bit 4 – FE: 帧错误**

如果接收缓冲器接收到的下一个字符有帧错误，即接收缓冲器中的下一个字符的第一个停止位为 0，那么 FE 置位。这一位一直有效直到接收缓冲器 (UDR) 被读取。当接收到的停止位为 1 时， FE 标志为 0。对 UCSRA 进行写入时，这一位要写 0。

- **Bit 3 – DOR: 数据溢出**

数据溢出时 DOR 置位。当接收缓冲器满 (包含了两个数据) , 接收移位寄存器又有数据，若此时检测到一个新的起始位，数据溢出就产生了。这一位一直有效直到接收缓冲器 (UDR) 被读取。对 UCSRA 进行写入时，这一位要写 0。

- **Bit 2 – PE: 奇偶校验错误**

当奇偶校验使能 (UPM1 = 1)，且接收缓冲器中所接收到的下一个字符有奇偶校验错误时 UPE 置位。这一位一直有效直到接收缓冲器 (UDR) 被读取。对 UCSRA 进行写入时 ,这一位要写 0。

- **Bit 1 – U2X: 倍速发送**

这一位仅对异步操作有影响。使用同步操作时将此位清零。

此位置 1 可将波特率分频因子从 16 降到 8 ,从而有效的将异步通信模式的传输速率加倍。

- **Bit 0 – MPCM: 多处理器通信模式**

设置此位将启动多处理器通信模式。 MPCM 置位后， USART 接收器接收到的那些不包含地址信息的输入帧都将被忽略。发送器不受 MPCM 设置的影响。详细信息请参考 P145 “ 多处理器通讯模式 ” 。

USART 控制和状态寄存器 B - UCSRB

Bit	7	6	5	4	3	2	1	0	
	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	

初始值	0	0	0	0	0	0	0	0

- **Bit 7 – RXCIE: 接收结束中断使能**

置位后使能 RXC 中断。当 RXCIE 为 1，全局中断标志位 SREG 置位， UCSRA 寄存器的 RXC 亦为 1 时可以产生 USART 接收结束中断。

- **Bit 6 – TXCIE: 发送结束中断使能**

置位后使能 TXC 中断。当 TXCIE 为 1，全局中断标志位 SREG 置位，UCSRA 寄存器的 TXC 亦为 1 时可以产生 USART 发送结束中断。

- **Bit 5 – UDRIE: USART 数据寄存器空中断使能**

置位后使能 UDRE 中断。当 UDRIE 为 1，全局中断标志位 SREG 置位，UCSRA 寄存器的 UDRE 亦为 1 时可以产生 USART 数据寄存器空中断。

- **Bit 4 – RXEN: 接收使能**

置位后将启动 USART 接收器。 RxD 引脚的通用端口功能被 USART 功能所取代。禁止接收器将刷新接收缓冲器，并使 FE、 DOR 及 PE 标志无效。

- **Bit 3 – TXEN: 发送使能**

置位后将启动 USART 发送器。TxD 引脚的通用端口功能被 USART 功能所取代。TXEN 清零后，只有等到所有的数据发送完成后发送器才能够真正禁止，即发送移位寄存器与发送缓冲寄存器中没有要传送的数据。发送器禁止后， TxD 引脚恢复其通用 I/O 功能。

- **Bit 2 – UCSZ2: 字符长度**

UCSZ2与UCSRC寄存器的UCSZ1:0结合在一起可以设置数据帧所包含的数据位数(字符长度)。

- **Bit 1 – RXB8: 接收数据位 8**

对 9 位串行帧进行操作时， RXB8 是第 9 个数据位。读取 UDR 包含的低位数据之前首先要读取 RXB8。

- **Bit 0 – TXB8: 发送数据位 8**

对 9 位串行帧进行操作时，TXB8 是第 9 个数据位。写 UDR 之前首先要对它进行写操作。

USART 控制和状态寄存器 C - UCSRC

Bit	7	6	5	4	3	2	1	0	
	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	
初始值	1	0	0	0	0	1	1	0	

UCSRC 寄存器与 UBRRH 寄存器共用相同的 I/O 地址。对该寄存器的访问，请参见 P146 “ 访问 UBRRH/ UCSRC 寄存器 ” 。

- **Bit 7 – URSEL: 寄存器选择**

通过该位选择访问 UCSRC 寄存器或 UBRRH 寄存器。当读 UCSRC 时，该位为 1 ；当写 UCSRC 时，URSEL 为 1。

- **Bit 6 – UMSEL: USART 模式选择**

通过这一位来选择同步或异步工作模式。

Table 63. UMSEL Bit Settings

UMSEL	模式
0	异步操作
1	同步操作

- **Bit 5:4 – UPM1:0: 奇偶校验模式**

这两位设置奇偶校验的模式并使能奇偶校验。如果使能了奇偶校验，那么在发送数据，发送器都会自动产生并发送奇偶校验位。对每一个接收到的数据，接收器都会产生一奇偶值，并与 UPM0 所设置的值进行比较。如果不匹配，那么就将 UCSRA 中的 PE 置位。

Table 64. UPM 设置

UPM1	UPM0	奇偶模式
0	0	禁止
0	1	保留
1	0	偶校验
1	1	奇校验

- **Bit 3 – USBS: 停止位选择**

通过这一位可以设置停止位的位数。接收器忽略这一位的设置。

Table 65. USBS 设置

USBS	停止位位数
0	1
1	2

- **Bit 2:1 – UCSZ1:0: 字符长度**

UCSZ1:0与UCSRB寄存器的 UCSZ2结合在一起可以设置数据帧包含的数据位数(字符长度)。

Table 66. UCSZ 设置

UCSZ2	UCSZ1	UCSZ0	字符长度
0	0	0	5 位
0	0	1	6 位
0	1	0	7 位
0	1	1	8 位
1	0	0	保留
1	0	1	保留
1	1	0	保留
1	1	1	9 位

- **Bit 0 – UCPOL: 时钟极性**

这一位仅用于同步工作模式。使用异步模式时，将这一位清零。UCPOL 设置了输出数据的改变和输入数据采样，以及同步时钟 XCK 之间的关系。

Table 67. UCPOL 设置

UCPOL	发送数据的改变 (TxD 引脚的输出)	接收数据的采样 (RxD 引脚的输入)
0	XCK 上升沿	XCK 下降沿
1	XCK 下降沿	XCK 上升沿

USART 波特率寄存器 - UBRRL 和 UBRRH

Bit	15	14	13	12	11	10	9	8	
	URSEL	–	–	–	UBRR[11:8]				UBRRH
	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	
初始值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

UCSRC 寄存器与 UBRRH 寄存器共用相同的 I/O 地址。对该寄存器的访问，请参见 P146 “ 访问 UBRRH/ UCSRC 寄存器 ” 。

- **Bit 15 – URSEL: 寄存器选择**

通过该位选择访问 UCSRC 寄存器或 UBRRH 寄存器。当读 UBRRH 时，该位为 0 ；当写 UBRRH 时，URSEL 为 0。

- **Bit 14:12 – 保留位**

这些位是为以后的使用而保留的。为了与以后的器件兼容，写 UBRRH 时将这些位清零。

- **Bit 11:0 – UBRR11:0: USART 波特率寄存器**

这个 12 位的寄存器包含了 USART 的波特率信息。其中 UBRRH 包含了 USART 波特率高 4 位，UBRRL 包含了低 8 位。波特率的改变将造成正在进行的数据传输受到破坏。写 UBRRL 将立即更新波特率分频器。

波特率设置的例子

对标准晶振及谐振器频率来说，异步模式下最常用的波特率可通过 Table 68 中 UBRR 的设置来产生。表中的粗体数据表示由此产生的波特率与目标波特率的偏差不超过 0.5%。更高的误差也是可以接受的，但发送器的抗噪性会降低，特别是需要传输大量数据时 (参看 P143 “ 异步工作范围 ”)。误差可以通过如下公式计算：

$$Error[\%] = \left(\frac{BaudRate_{Closest\ Match}}{BaudRate} - 1\right) \bullet 100\%$$

Table 68. 通用振荡器频率下设置 UBRR 的例子

波特率 (bps)	f_{osc} = 1.0000 MHz				f_{osc} = 1.8432 MHz				f_{osc} = 2.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差
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%
最大 (1)	62.5 kbps		125 kbps		115.2 kbps		230.4 kbps		125 kbps		250 kbps	

1. UBRR = 0, 误差 = 0.0%

Table 69. 通用振荡器频率下设置 UBRR 的例子 (续)

波特率 (bps)	f_{osc} = 3.6864 MHz				f_{osc} = 4.0000 MHz				f_{osc} = 7.3728 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差
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%
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%
最大 (1)	230.4 kbps		460.8 kbps		250 kbps		0.5 Mbps		460.8 kbps		921.6 kbps	

1. UBRR = 0, 误差 = 0.0%

Table 70. 通用振荡器频率下设置 UBRR 的例子 (续)

波特率 (bps)	f_{osc} = 8.0000 MHz				f_{osc} = 11.0592 MHz				f_{osc} = 14.7456 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差
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%
最大 (1)	0.5 Mbps		1 Mbps		691.2 kbps		1.3824 Mbps		921.6 kbps		1.8432 Mbps	

1. UBRR = 0, 误差 = 0.0%

Table 71. 通用振荡器频率下设置 UBRR 的例子 (续)

波特率 (bps)	f_{osc} = 16.0000 MHz				f_{osc} = 18.4320 MHz				f_{osc} = 20.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差
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%	–	–	–	–	–	–	–	–
最大 (1)	1 Mbps		2 Mbps		1.152 Mbps		2.304 Mbps		1.25 Mbps		2.5 Mbps	

1. UBRR = 0, 误差 = 0.0%

两线串行接口 TWI

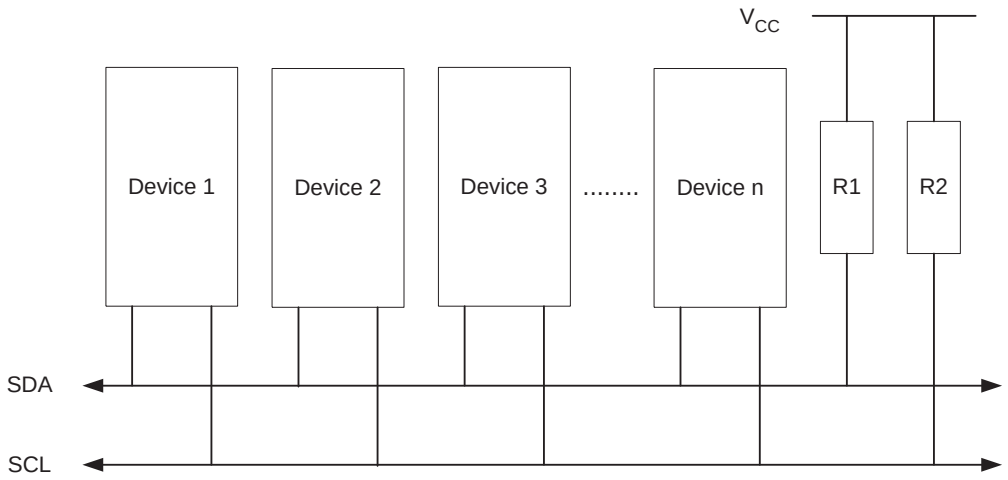
特点

- 简单，但是强大而灵活的通讯接口，只需要两根线
- 支持主机和从机操作
- 器件可以工作于发送器模式或接收器模式
- 7 位地址空间允许有 128 个从机
- 支持多主机仲裁
- 高达 400 kHz 的数据传输率
- 斜率受控的输出驱动器
- 可以抑制总线尖峰的噪声抑制器
- 完全可编程的从机地址以及公共地址
- 睡眠时地址匹配可以唤醒 AVR

两线串行接口总线定义

两线接口 TWI 很适合于典型的处理器应用。 TWI 协议允许系统设计者只用两根双向传输线就可以将 128 个不同的设备互连到一起。这两根线一是时钟 SCL，一是数据 SDA。外部硬件只需要两个上拉电阻，每根线上一个。所有连接到总线上的设备都有自己的地址。TWI 协议解决了总线仲裁的问题。

Figure 76. TWI 总线的连接



TWI 词汇

以下定义将在本节频繁出现。

Table 72. TWI 词汇

单词	说明
主机	启动和停止传输的设备。主机同时要产生 SCL 时钟
从机	被主机寻址的设备
发送器	将数据放到总线上的设备
接收器	从总线读取数据的设备

电气连接

从 Figure 76 可以看出，两根线都通过上拉电阻与正电源连接。所有 TWI 兼容的器件的总线驱动都是漏极开路或集电极开路的。这样就实现了对接口操作非常关键的线与功能。TWI器件输出为"0”时，TWI总线会产生低电平。当所有的TWI器件输出为三态时，总线会输出高电平，允许上拉电阻将电压拉高。注意，为保证所有的总线操作，凡是与 TWI 总线连接的 AVR 器件必须上电。

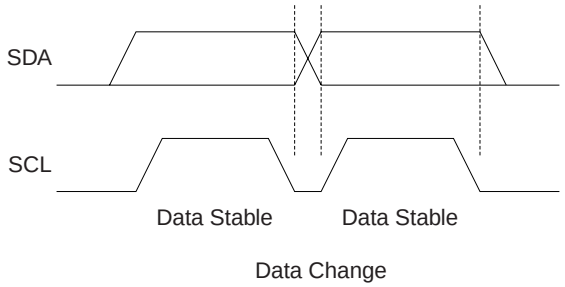
与总线连接的器件数目受如下条件限制：总线电容要低于 400 pF，而且可以用 7 位从机地址进行寻址。 TWI 详细的电气特性说明请见 P272 “ 两线串行接口特性 ” 。这儿给出了两个不同的规范，一种是总线速度低于 100 kHz，而另外一种是总线速度高达 400 kHz。

数据传输和帧格式

传输数据 (位)

TWI 总线上数据位的传送与时钟脉冲同步。时钟线为高时，数据线电压必须保持稳定，除非在启动与停止的状态下。

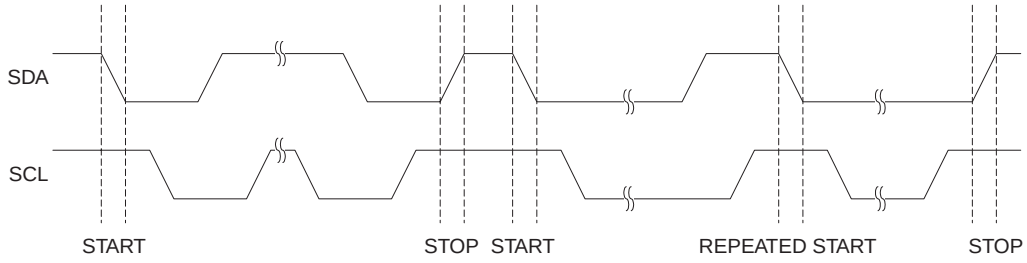
Figure 77. 数据有效性



START/STOP 状态

主机启动与停止数据传输。主机在总线上发出 START 信号以启动数据传输；在总线上发出 STOP 信号以停止数据传输。在 START 与 STOP 状态之间，需要假定总线忙，不允许其它主机控制总线。特例是在 START 与 STOP 状态之间发出一个新的 START 状态。这被称为 REPEATED START 状态，适用于主机在不放弃总线控制的情况下启动新的传送。在 REPEATED START 之后，直到下一个 STOP，需要假定总线处于忙的状态。这与 START 是完全一样的，因此在本手册中，如果没有特殊说明，START 与 REPEATED START 均用 START 表述。如下所示，START 与 STOP 状态是在 SCL 线为高时，通过改变 SDA 电平来实现的。

Figure 78. START、 REPEATED START 与 STOP 状态



地址数据包格式

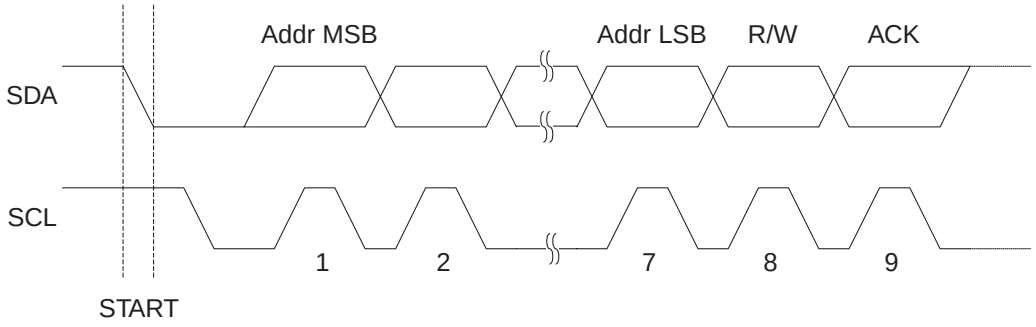
所有在 TWI 总线上传送的地址包均为 9 位，包括 7 位地址位、 1 位 READ/WRITE 控制位与 1 位应答位。如果 READ/WRITE 为 1，则执行读操作；否则执行写操作。从机被寻址后，必须在第九个 SCL (ACK) 周期通过拉低 SDA 作出应答。若该从机忙或有其它原因无法响应主机，则应该在 ACK 周期保持 SDA 为高。然后主机可以发出 STOP 状态或 REPEATED START 状态重新开始发送。地址包包括从机地址与分别称为 SLA+R 或 SLA+W 的 READ 或 WRITE 位。

地址字节的 MSB 首先被发送。从机地址由设计者自由分配，但需要保留地址 0000 000 作为广播地址。

当发送广播呼叫时，所有的从机应在 ACK 周期通过拉低 SDA 作出应答。当主机需要发送相同的信息给多个从机时可以使用广播功能。当 Write 位在广播呼叫之后发送，所有的从机通过在 ACK 周期通过拉低 SDA 作出响应。所有的从机接收到紧跟的数据包。注意在整体访问中发送 Read 位没有意义，因为如果几个从机发送不同的数据会带来总线冲突。

所有形如 1111 xxx 格式的地址都需要保留，以便将来使用。

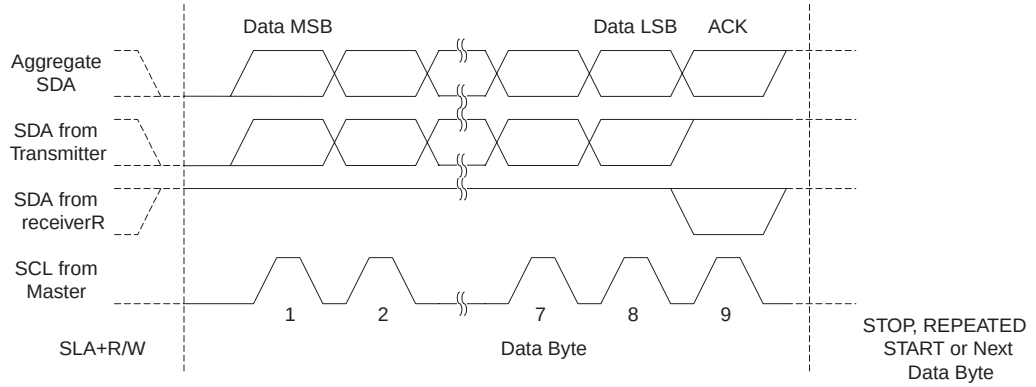
Figure 79. 地址包格式



数据包格式

所有在 TWI 总线上传送的数据包为 9 位长，包括 8 位数据位及 1 位应答位。在数据传送中，主机产生时钟及 START 与 STOP 状态，而接收器响应接收。应答是由从机在第 9 个 SCL 周期拉低 SDA 实现的。如果接收器使 SDA 为高，则发出 NACK 信号。接收器完成接收，或者由于某些原因无法接收更多的数据，应该在收到最后的字节后发出 NACK 来告知发送器。数据的 MSB 首先发送。

Figure 80. 数据包格式

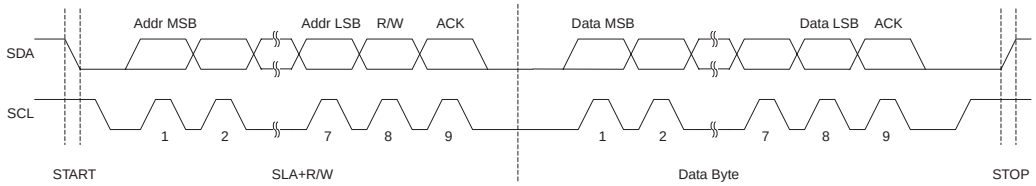


将地址包和数据包组合为一个完整的传输过程

发送主要由 START 状态、SLA+R/W、至少一个数据包及 STOP 状态组成。只有 START 与 STOP 状态的空信息是非法的。可以利用 SCL 的线与功能来实现主机与从机的握手。从机可通过拉低 SCL 来延长 SCL 低电平的时间。当主机设定的时钟速度相对于从机太快，或从机需要额外的时间来处理数据时，这一特性是非常有用的。从机延长 SCL 低电平的时间不会影响 SCL 高电平的时间，因为 SCL 高电平时间是由主机决定的。由上述可知，通过改变 SCL 的占空比可降低 TWI 数据传送速度。

Figure 81 说明了典型的数据传送。注意 SLA+R/W 与 STOP 之间传送的字节数由应用程序的协议决定。

Figure 81. 典型的数据传送



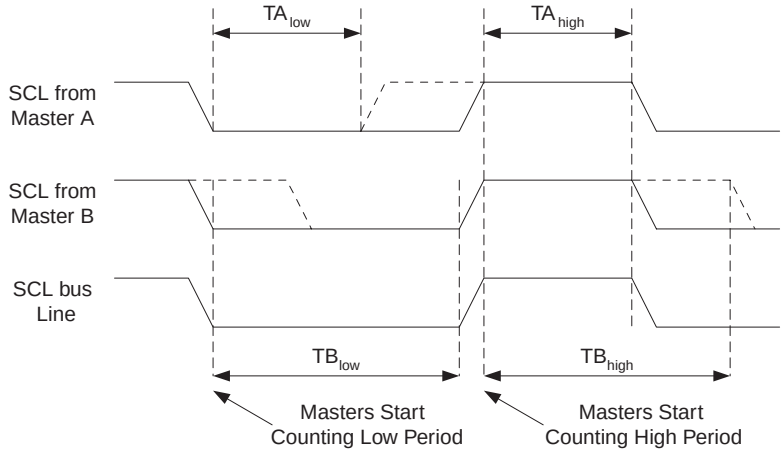
多主机总线系统，仲裁和同步

TWI 协议允许总线上由多个主机。特别要注意的是即使有多个主机同时开始发生数据，也要保证发送正常进行。多主机系统中有两个问题：

- 算法必须只能允许一个主机完成传送。当其余主机发现它们失去选择权后应停止传送。这个选择过程称为仲裁。当竞争中的主机发现其仲裁失败，应立即转换到从机模式检测是否被获得总线控制权的的主机寻址。事实上多主机同时传送时不应该让从机检测到，即不许破坏数据在总线上的传送。
- 不同的主机可能使用不同的 SCL 频率。为保证传送的一致性，必须设计一种同步主机时钟的方案。这会简化仲裁过程。

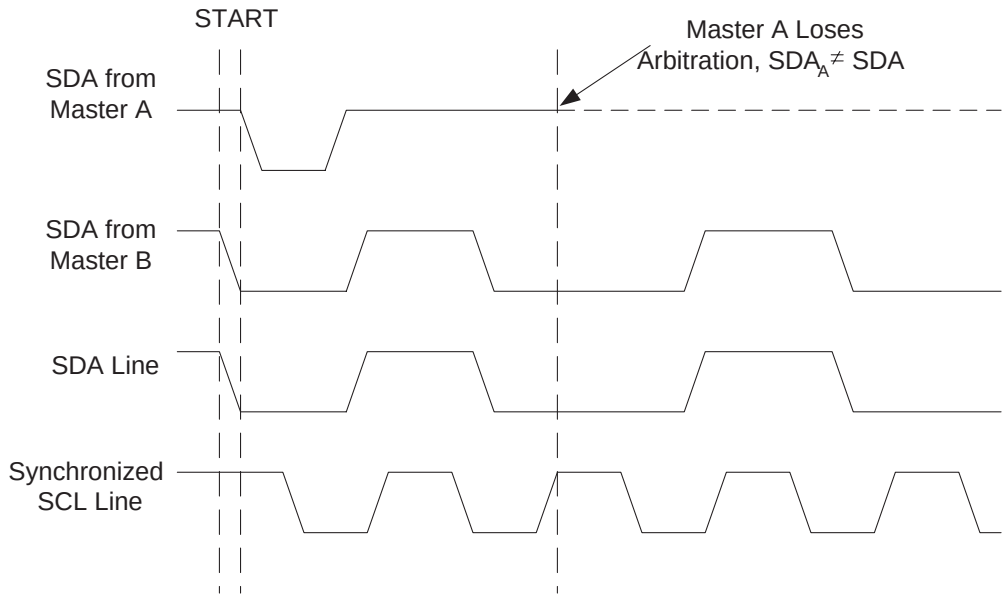
总线的线与功能用来解决上述问题。将所有的主机时钟进行与操作，会生成组合的时钟，其高电平时间等于所有主机中最短的一个；低电平时间则等于所有主机中最长的一个。所有的主机都监听 SCL，使其可以有效地计算本身高 / 低电平与组合 SCL 信号高 / 低电平的时间差异。

Figure 82. 多主机 SCL 的同步



输出数据之后所有的主机都持续监听 SDA 来实现仲裁。如果从 SDA 读回的数值与主机输出的数值不匹配，该主机即失去仲裁。要注意只有当一个主机输出高电平的 SDA，而其它主机输出为低，该主机才会失去仲裁，并立即转为从机模式，检测是否被胜出的主机寻址。失去仲裁的主机必须将 SDA 置高，但在当前的数据或地址包结束之前还可以产生时钟信号。仲裁将会持续到系统只有一个主机。这可能会占用许多比特。如果几个主机对相同的从机寻址，仲裁将会持续到数据包。

Figure 83. 两主机之间的仲裁



注意不允许在以下情况进行仲裁：

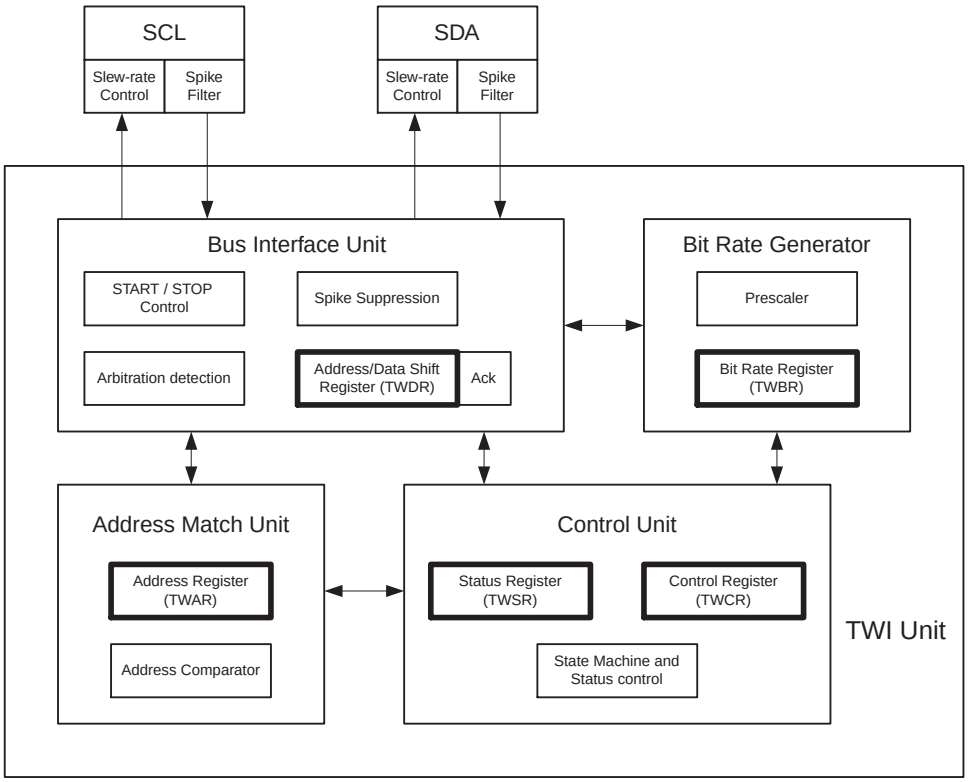
- 一个 REPEATED START 状态与一个数据位
- 一个 STOP 状态与一个数据位
- 一个 REPEATED START 状态与一个 STOP 状态

应用软件应考虑上述情况，保证不会出现这些非法仲裁状态。这意味着在多主机系统中，所有的数据传输必须由相同的 SLA+R/W 与数据包组合组成。换句话说：所有的传送必须包含相同数目的数据包，否则仲裁结果无法定义。

TWI 模块综述

TWI模块由几个子模块组成，如Figure 84所示。所有位于粗线之中的寄存器可以通过AVR数据总线进行访问。

Figure 84. TWI 模块概述



SCL 和 SDA 引脚

SCL 与SDA为MCU的 TWI接口引脚。引脚的输出驱动器包含一个波形斜率限制器以满足 TWI 规范。引脚的输入部分包括尖峰抑制单元以去除小于 50 ns 的毛刺。当相应的端口设置为 SCL 与 SDA 引脚时，可以使能 I/O 口内部的上拉电阻，这样可省掉外部的上拉电阻。

比特率发生器单元

TWI工作于主机模式时，比特率发生器控制时钟信号SCL的周期。具体由TWI状态寄存器TWSR的预分频系数以及比特率寄存器TWBR设定。当TWI工作在从机模式时，不需要对比特率或预分频进行设定，但从机的 CPU 时钟频率必须大于 TWI 时钟线 SCL 频率的 16 倍。注意，从机可能会延长 SCL 低电平的时间，从而降低 TWI 总线的平均时钟周期。SCL 的频率根据以下的公式产生：

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

- TWBR = TWI 比特率寄存器的数值
- TWPS = TWI 状态寄存器预分频的数值

Note: TWI 工作在主机模式时，TWBR 值应该不小于 10。否则主机会在 SDA 与 SCL 产生错误输出作为提示信号。问题出现于 TWI 工作在主机模式下，向从机发送 Start + SLA + R/W 的时候 (不需要真的有从机与总线连接)。

总线接口单元

该单元包括数据与地址移位寄存器 TWDR，START/STOP 控制器和总线仲裁判定硬件电路。 TWDR 寄存器用于存放发送或接收的数据或地址。除了 8 位的 TWDR，总线接口单元还有一个寄存器，包含了用于发送或接收应答的 (N)ACK。这个 (N)ACK 寄存器不能由程序直接访问。当接收数据时，它可以通过 TWI 控制寄存器 TWCR 来置位或清零；在发送数据时， (N)ACK 值由 TWCR 的设置决定。

START/STOP 控制器负责产生和检测 TWI 总线上的 START、REPEATED START 与 STOP 状态。即使在 MCU 处于休眠状态时，START/STOP 控制器仍然能够检测 TWI 总线上的 START/STOP 条件，当检测到自己被 TWI 总线上的主机寻址时，将 MCU 从休眠状态唤醒。

如果 TWI 以主机模式启动了数据传输，仲裁检测电路将持续监听总线，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了总线控制权，则通知 TWI 控制单元执行正确的动作，并产生合适的状态码。

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 寄存器的 TWI 广播应答识别使能位 TWGCE 为 "1”，从总线接收到的地址也会与广播地址进行比较。一旦地址匹配成功，控制单元将得到通知以进行正确地响应。TWI 可以响应，也可以不响应主机的寻址，这取决于 TWCR 寄存器的设置。即使 MCU 处于休眠状态时，地址匹配单元仍可继续工作。一旦主机寻址到这个器件，就可以将 MCU 从休眠状态唤醒。

控制单元

控制单元监听 TWI 总线，并根据 TWI 控制寄存器 TWCR 的设置作出相应的响应。当 TWI 总线上产生需要应用程序干预处理的事件时，TWI 中断标志位 TWINT 置位。在下一个时钟周期， TWI 状态寄存器 TWSR 被表示这个事件的状态码字所更新。在其它时间里，TWSR 的内容为一个表示无事件发生的特殊状态字。一旦 TWINT 标志位置 "1”，时钟线 SCL 即被拉低，暂停 TWI 总线上的数据传输，让用户程序处理事件。

在下列状况出现时， TWINT 标志位置位：

- 在 TWI 传送完 START/REPEATED START 信号之后
- 在 TWI 传送完 SLA+R/W 数据之后
- 在 TWI 传送完地址字节之后
- 在 TWI 总线仲裁失败之后
- 在 TWI 被主机寻址之后 (广播方式或从机地址匹配)
- 在 TWI 接收到一个数据字节之后
- 作为从机工作时， TWI 接收到 STOP 或 REPEATED START 信号之后
- 由于非法的 START 或 STOP 信号造成总线错误时

TWI 寄存器说明

TWI 比特率寄存器 - TWBR

Bit	7	6	5	4	3	2	1	0	
	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	
初始值	0	0	0	0	0	0	0	0	

- **Bits 7..0 – TWI 比特率寄存器**

TWBR 为比特率发生器分频因子。比特率发生器是一个分频器 ,在主机模式下产生SCL时钟频率。比特率计算公式请见 P162 “ 比特率发生器单元 ” 。

TWI 控制寄存器 - TWCR

Bit	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
读 / 写	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
初始值	0	0	0	0	0	0	0	0	

TWCR 用来控制TWI操作。它用来使能TWI ,通过施加START到总线上来启动主机访问，产生接收器应答，产生 STOP 状态，以及在写入数据到 TWDR 寄存器时控制总线的暂停等。这个寄存器还可以给出在 TWDR 无法访问期间 ,试图将数据写入到 TWDR 而引起的写入冲突信息。

- **Bit 7 – TWINT: TWI 中断标志**

当 TWI 完成当前工作 ,希望应用程序介入时 TWINT 置位。若 SREG 的 I 标志以及 TWCR 寄存器的 TWIE 标志也置位，则 MCU 执行 TWI 中断例程。当 TWINT 置位时， SCL 信号的低电平被延长。 TWINT 标志的清零必须通过软件写 "1” 来完成。执行中断时硬件不会自动将其改写为 "0”。要注意的是，只要这一位被清零， TWI 立即开始工作。因此，在清零 TWINT 之前一定要首先完成对地址寄存器 TWAR ,状态寄存器 TWSR ,以及数据寄存器 TWDR 的访问。

- **Bit 6 – TWEA: TWI 使能应答**

TWEA 标志控制应答脉冲的产生。若 TWEA 置位，出现如下条件时接口发出 ACK 脉冲 :

1. 芯片的从机地址与主机发出的地址相符合
2. TWAR 的 TWGCE 置位时接收到广播呼叫
3. 在主机 / 从机接收模式下接收到一个字节的数据

将 TWEA 清零可以使器件暂时脱离总线。置位后器件重新恢复地址识别。

- **Bit 5 – TWSTA: TWI START 状态标志**

当 CPU 希望自己成为总线上的主机时需要置位 TWSTA。 TWI 硬件检测总线是否可用。若总线空闲，接口就在总线上产生 START 状态。若总线忙，接口就一直等待，直到检测到一个 STOP 状态 ,然后产生 START 以声明自己希望成为主机。发送 START 之后软件必须清零 TWSTA。

- **Bit 4 – TWSTO: TWI STOP 状态标志**

在主机模式下 ,如果置位 TWSTO ,TWI 接口将在总线上产生 STOP 状态 ,然后 TWSTO 自动清零。在从机模式下 ,置位 TWSTO 可以使接口从错误状态恢复到未被寻址的状态。此时总线上不会有 STOP 状态产生，但 TWI 返回一个定义好的未被寻址的从机模式且释放 SCL 与 SDA 为高阻态。

- **Bit 3 – TWWC: TWI 写冲突标志**

当 TWINT 为低时写数据寄存器 TWDR 将置位 TWWC。当 TWINT 为高时，每一次对 TWDR 的写访问都将更新此标志。

- **Bit 2 – TWEN: TWI 使能**

TWEN 位用于使能TWI操作与激活TWI接口。当TWEN位被写为"1”时，TWI引脚将I/O引脚切换到 SCL 与 SDA 引脚，使能波形斜率限制器与尖峰滤波器。如果该位清零，TWI 接口模块将被关闭，所有 TWI 传输将被终止。

- **Bit 1 – Res: 保留**

保留，读返回值为 ”0”。

- **Bit 0 – TWIE: TWI 中断使能**

当 SREG 的 I 以及 TWIE 置位时，只要 TWINT 为 "1”，TWI 中断就激活。

TWI 状态寄存器 - TWSR

Bit	7	6	5	4	3	2	1	0	
	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
读 / 写	R	R	R	R	R	R	R/W	R/W	
初始值	1	1	1	1	1	0	0	0	

- **Bits 7..3 – TWS: TWI 状态**

这 5 位用来反映 TWI 逻辑和总线的状态。不同的状态代码将会在后面的部分描述。注意从 TWSR 读出的值包括 5 位状态值与 2 位预分频值。检测状态位时设计者应屏蔽预分频位为 "0”。这使状态检测独立于预分频器设置。在无特殊声明的情况下，在手册中使用该方法。

- **Bit 2 – Res: 保留**

保留，读返回值为 "0”。

- **Bits 1..0 – TWPS: TWI 预分频位**

这两位可读 / 写，用于控制比特率预分频因子。

Table 73. TWI 比特率预分频器

TWPS1	TWPS0	预分频器值
0	0	1
0	1	4
1	0	16
1	1	64

如何计算比特率请见 P162 “ 比特率发生器单元 ”。 TWPS1..0 值在该公式中使用。

TWI 数据寄存器 - TWDR

Bit	7	6	5	4	3	2	1	0	
	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	
初始值	1	1	1	1	1	1	1	1	

在发送模式，TWDR 包含了要发送的字节；在接收模式，TWDR 包含了接收到的数据。当 TWI 接口没有进行移位工作 (TWINT 置位) 时这个寄存器是可写的。在第一次中断发生之前用户不能够初始化数据寄存器。只要 TWINT 置位，TWDR 的数据就是稳定的。在数据移出时，总线上的数据同时移入寄存器。 TWDR 总是包含了总线上出现的最后一个字节，除非 MCU 是从掉电或省电模式被 TWI 中断唤醒。此时 TWDR 的内容没有定义。总线仲裁失败时，主机将切换为从机，但总线上出现的数据不会丢失。 ACK 的处理由 TWI 逻辑自动管理， CPU 不能直接访问 ACK。

- **Bits 7..0 – TWD: TWI 数据寄存器**

根据状态的不同，其内容为要发送的下一个字节，或是接收到的数据。

TWI(从机) 地址寄存器 - TWAR

Bit	7	6	5	4	3	2	1	0	
	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	
初始值	1	1	1	1	1	1	1	0	

TWAR 的高 7 位为从机地址。工作于从机模式时，TWI 将根据这个地址进行响应。主机模式不需要此地址。在多主机系统中， TWAR 需要进行设置以便其他主机访问自己。

TWAR 的 LSB 用于识别广播地址 (0x00)。芯片内有一个地址比较器。一旦接收到的地址和本机地址一致，芯片就请求中断。

- **Bits 7..1 – TWA: TWI 从机地址寄存器**

其值为从机地址。

- **Bit 0 – TWGCE: TWI 广播识别使能**

置位后 MCU 可以识别 TWI 总线广播。

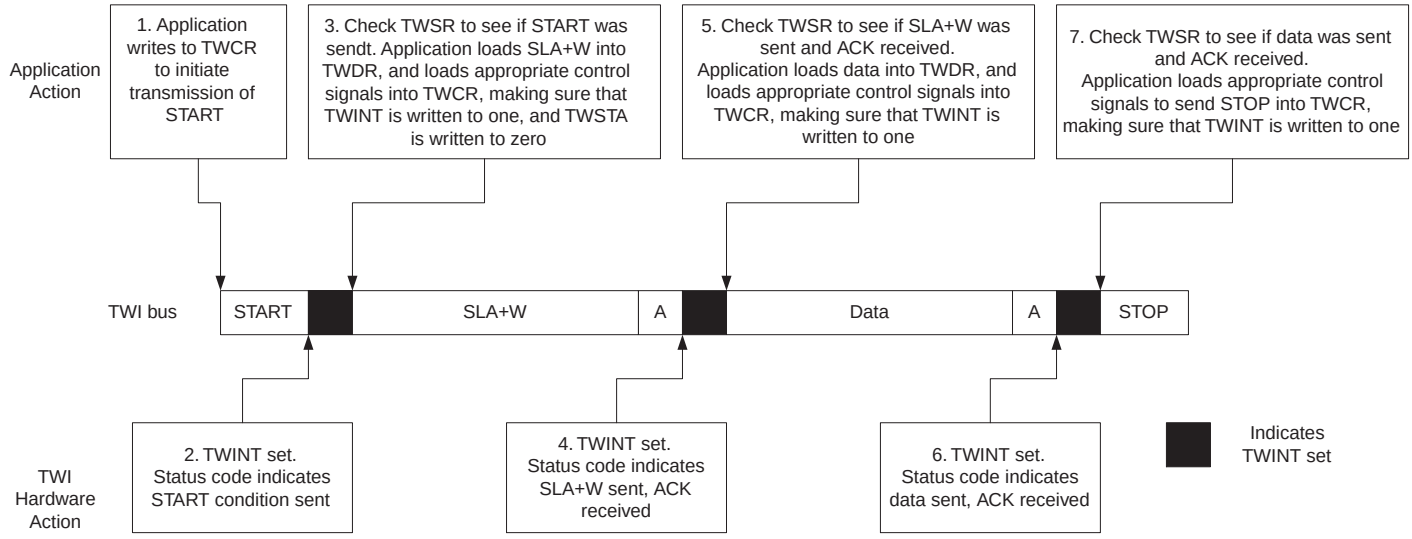
使用 TWI

AVR的 TWI接口是面向字节和基于中断的。所有的总线事件，如接收到一个字节或发送了一个 START 信号等，都会产生一个 TWI 中断。由于 TWI 接口是基于中断的，因此 TWI 接口在字节发送和接收过程中，不需要应用程序的干预。 TWCR 寄存器的 TWI 中断允许 TWIE位和SREG寄存器的全局中断允许位一起决定了应用程序是否响应TWINT标志位产生的中断请求。如果 TWIE 被清零，应用程序只能采用轮询 TWINT 标志位的方法来检测 TWI 总线状态。

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 包含了表明当前 TWI 总线状态的值。应用程序可以读取 TWCR 的状态码，判别此时的状态是否正确，并通过设置 TWCR 与 TWDR 寄存器，决定在下一个 TWI 总线周期 TWI 接口应该如何工作。

Figure 85 给出应用程序与 TWI 接口连接的例子。该例中，主机发送一个数据字节给从机。这里只是简述，本节的后面会有更多的解释，还有简单的代码例程。

Figure 85. 典型数据传输中应用程序与 TWI 的接口



1. TWI 传输的第一步是发送 START 信号。通过对 TWCR 写入特定值，指示 TWI 硬件发送 START 信号。写入的值将在后面说明。在写入值时 TWINT 位要置位，这非常重要。给 TWINT 写 "1" 清除此标志。 TWCR 寄存器的 TWINT 置位期间 TWI 不会启动任何操作。一旦 TWINT 清零， TWI 由 START 信号启动数据传输。
2. START 信号被发送后，TWCR 寄存器的 TWINT 标志位置位，TWCR 更新为新的状态码，表示 START 信号成功发送。
3. 应用程序应检验 TWSR，确定 START 信号已成功发送。如果 TWSR 显示为其它，应用程序可以执行一些指定操作，比如调用错误处理程序。如果状态码与预期一致，应用程序必须将 SLA+W 载入 TWDR。TWDR 可同时在地址与数据中使用。TWDR 载入 SLA+W 后，TWCR 必须写入特定值指示 TWI 硬件发送 SLA+W 信号。写入的值将在后面说明。在写入值时 TWINT 位要置位，这非常重要。给 TWINT 写 "1" 清除此标志。 TWCR 寄存器的 TWINT 置位期间 TWI 不会启动任何操作。一旦 TWINT 清零， TWI 启动地址包的传送。
4. 地址包发送后，TWCR 寄存器的 TWINT 标志位置位，TWDR 更新为新的状态码，表示地址包成功发送。状态代码还会反映从机是否响应包。
5. 应用程序应检验 TWSR，确定地址包已成功发送、ACK 为期望值。如果 TWSR 显示为其它，应用程序可能执行一些指定操作，比如调用错误处理程序。如果状态码与预期一致，应用程序必须将数据包载入 TWDR。随后， TWCR 必须写入特定值指示 TWI 硬件发送 TWDR 中的数据包。写入的值将在后面说明。在写入值时 TWINT 位要置位，这非常重要。 TWCR 寄存器中的 TWINT 置位期间 TWI 不会启动任何操作。一旦 TWINT 清零， TWI 启动数据包的传输。
6. 数据包发送后，TWCR 寄存器的 TWINT 标志位置位，TWSR 更新为新的状态码，表示数据包成功发送。状态代码还会反映从机是否响应包。
7. 应用程序应检验 TWSR，确定地址包已成功发送、ACK 为期望值。如果 TWSR 显示为其它，应用程序可能执行一些指定操作，比如调用错误处理程序。如果状态码与预期一致， TWCR 必须写入特定值指示 TWI 硬件发送 STOP 信号。写入的值将在后面说明。在写入值时 TWINT 位要置位，这非常重要。给 TWINT 写 "1" 清除此标志。 TWCR 寄存器中的 TWINT 置位期间 TWI 不会启动任何操作。一旦 TWINT 清零， TWI 启动 STOP 信号的传送。注意 TWINT 在 STOP 状态发送后不会置位。

尽管示例比较简单，但它包含了 TWI 数据传输过程中的所有规则。总结如下：

- 当 TWI 完成一次操作并等待反馈时， TWINT 标志置位。直到 TWINT 清零，时钟线 SCL 才会拉低。

- TWINT标志置位时，用户必须用与下一个TWI总线周期相关的值更新TWI寄存器。例如， TWDR 寄存器必须载入下一个总线周期中要发送的值。
- 当所有的 TWI 寄存器得到更新，而且其它挂起的应用程序也已经结束， TWCR 被写入数据。写 TWCR 时， TWINT 位应置位。对 TWINT 写 "1" 清除此标志。 TWI 将开始执行由 TWCR 设定的操作。

下面给出了汇编与 C 语言例程。注意假设下面代码均已给出定义。

	汇编代码例程	C 代码例程	说明
1	`ldi r16, (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)` `out TWCR, r16`	`TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)`	发出 START 信号

2	`wait1:`
	`in r16,TWCR`
	`sbrs r16,TWINT`
	`rjmp wait1`
		`while (!(TWCR & (1<<TWINT)))`
		`;`
			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`
		`if ((TWSR & 0xF8) != START)`
		`ERROR();`
			检验 TWI 状态寄存器，屏蔽预分频位，如果状态字不是 START 转出错处理

	`ldi r16, SLA_W`
	`out TWDR, r16`
	`ldi r16, (1<<TWINT) | (1<<TWEN)`
	`out TWCR, r16`
		`TWDR = SLA_W;`
		`TWCR = (1<<TWINT) | (1<<TWEN);`
			将 SLA_W 载入 TWDR 寄存器， TWINT 位清零，启动发送地址

4	`wait2:`
	`in r16,TWCR`
	`sbrs r16,TWINT`
	`rjmp wait2`
		`while (!(TWCR & (1<<TWINT)))`
		`;`
			等待 TWINT 置位， TWINT 置位表示总线命令 SLA+W 已发出，及收到应答信号 ACK/NACK

5	`in r16,TWSR`
	`andi r16, 0xF8`
	`cpi r16, MT_SLA_ACK`
	`brne ERROR`
		`if ((TWSR & 0xF8) != MT_SLA_ACK)`
		`ERROR();`
			检验 TWI 状态寄存器，屏蔽预分频位，如果状态字不是 MT_SLA_ACK 转出错处理

	`ldi r16, DATA`
	`out TWDR, r16`
	`ldi r16, (1<<TWINT) | (1<<TWEN)`
	`out TWCR, r16`
		`TWDR = DATA;`
		`TWCR = (1<<TWINT) | (1<<TWEN);`
			将数据载入 TWDR 寄存器， TWINT 清零，启动发送数据

6	`wait3:`
	`in r16,TWCR`
	`sbrs r16,TWINT`
	`rjmp wait3`
		`while (!(TWCR & (1<<TWINT)))`
		`;`
			等待 TWINT 置位， TWINT 置位表示总线数据 DATA 已发送，及收到应答信号 ACK/NACK

7	`in r16,TWSR`
	`andi r16, 0xF8`
	`cpi r16, MT_DATA_ACK`
	`brne ERROR`
		`if ((TWSR & 0xF8) != MT_DATA_ACK)`
		`ERROR();`
			检验 TWI 状态寄存器，屏蔽预分频器，如果状态字不是 MT_DATA_ACK 转出错处理

	`ldi r16, (1<<TWINT)|(1<<TWEN)|(1<<TWSTO)` `out TWCR, r16`	`TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);`	发送 STOP 信号

传输模式

TWI 可以工作于 4 个不同的模式 : 主机发送器 (MT)、主机接收器 (MR)、从机发送器 (ST) 及从机接收器 (SR)。同一应用程序可以使用几种模式。例如， TWI 可用 MT 模式给 TWI EEPROM 写入数据 ,用 MR 模式从 EEPROM 读取数据。如果系统中有其它主机存在 ,它们可能给 TWI 发送数据，此时就可以用 SR 模式。应用程序决定采用何种模式。

下面对每种模式进行具体说明。每种模式的状态码在详细说明数据发送的图中进行描述。这些图包含了如下的缩写：

S：START 状态

Rs：REPEATED START 状态

R：读一个比特 (SDA 为高电平)

W：写一个比特 (SDA 为低电平)

A：应答位 (SDA 为低电平)

$\overline{A}$：无应答位 (SDA 为高电平)

Data：8 位数据

P：STOP 状态

SLA：从机地址

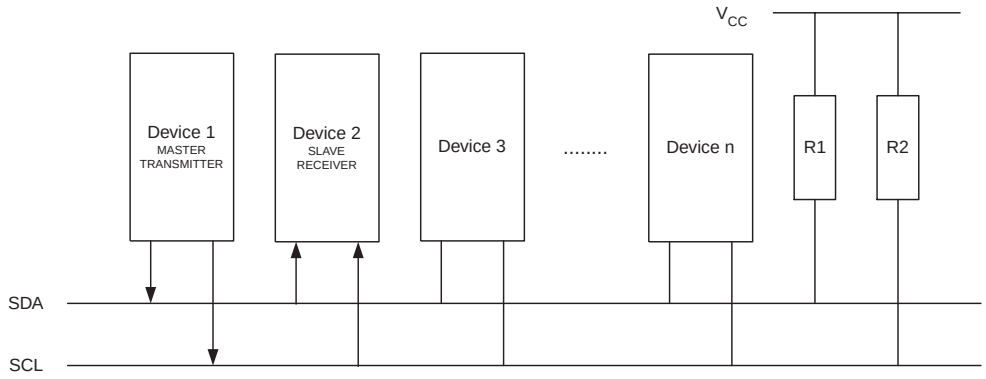
在 Figure 87 到 Figure 93 中 ,圆圈用来说明 TWINT 标志已经置位。圆圈中的数字用来表示预 TWSR 的数值，其中分频位屏蔽为 0。在这些地方应用程序必须执行合适的工作以继续 / 完成 TWI 传输。 TWI 传输被挂起，一直到 TWINT 标志被软件清零。

TWINT 标志置位后 ,TWSR 的状态码用来决定适当的软件操作。Table 74 到 Table 77 给出了每一个状态码所需的软件工作和后续串行传输的细节。注意在这些表中预分频位屏蔽为 0。

主机发送模式

在主机发送模式，主机可以向从机发送数据，如 Figure 86 所示。为进入主机模式，必须发送 START 信号。紧接着的地址包格式决定进入 MT 或 MR 模式。如果发送 SLA+W 进入 MT 模式；如果发送 SLA+R 则进入 MR 模式。本节所提到的状态字均假设其预分频位为 "0”。

Figure 86. 主机发送模式下的数据传输



通过在 TWCR 寄存器中写入下列数值发出 START 信号：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	1	0	X	1	0	X

TWEN必须置位以使能两线接口 ,TWSTA必须置"1”来发出START 信号且 TWINT必须置"1”来对 TWINT 标志清零。TWI 逻辑开始检测串行总线 ,一旦总线空闲就发送 START。接着中断标志 TWINT 置位，TWSR 的状态码为 0x08 (见 Table 74)。为进入 MT 模式 ，必

须发送 SLA+W。这可通过对 TWDR 写入 SLA+W 来实现。完成此操作后软件清零 TWINT 标志， TWI 传输继续进行。这通过在 TWCR 寄存器中写入下述值完成：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	0	0	X	1	0	X

当SLA+W 发送完毕并接收到确认信号，主机的TWINT标志再次置位。此时主机的TWSR 状态码可能是 0x18、 0x20 或 0x38。对各状态码的正确响应列于 Table 74。

SLA+W 发送成功后可以开始发送数据包。这通过对 TWDR 写入数据实现。TWDR 只有在 TWINT 为高时方可写入。否则，访问被忽略，寄存器 TWCR 的写碰撞位 TWWC 置位。TWDR 更新后，TWINT 位应清零来继续传送。这通过在 TWCR 寄存器中写入下述值完成

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	0	0	X	1	0	X

这过程会一直重复下去，直到最后的字节发送完且发送器产生 STOP 或 REPEATED START 信号 。STOP 信号通过在 TWCR 中写入下述值实现：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	0	1	X	1	0	X

REPEATED START 信号通过在 TWCR 中写入下述值实现：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	1	0	X	1	0	X

在 REPEATED START (状态 0x10) 后，两线接口可以再次访问相同的从机，或不发送 STOP 信号来访问新的从机。REPEATED START 使得主机可以在不丢失总线控制的条件下在从机、主机发送器及主机接收器模式间进行切换。

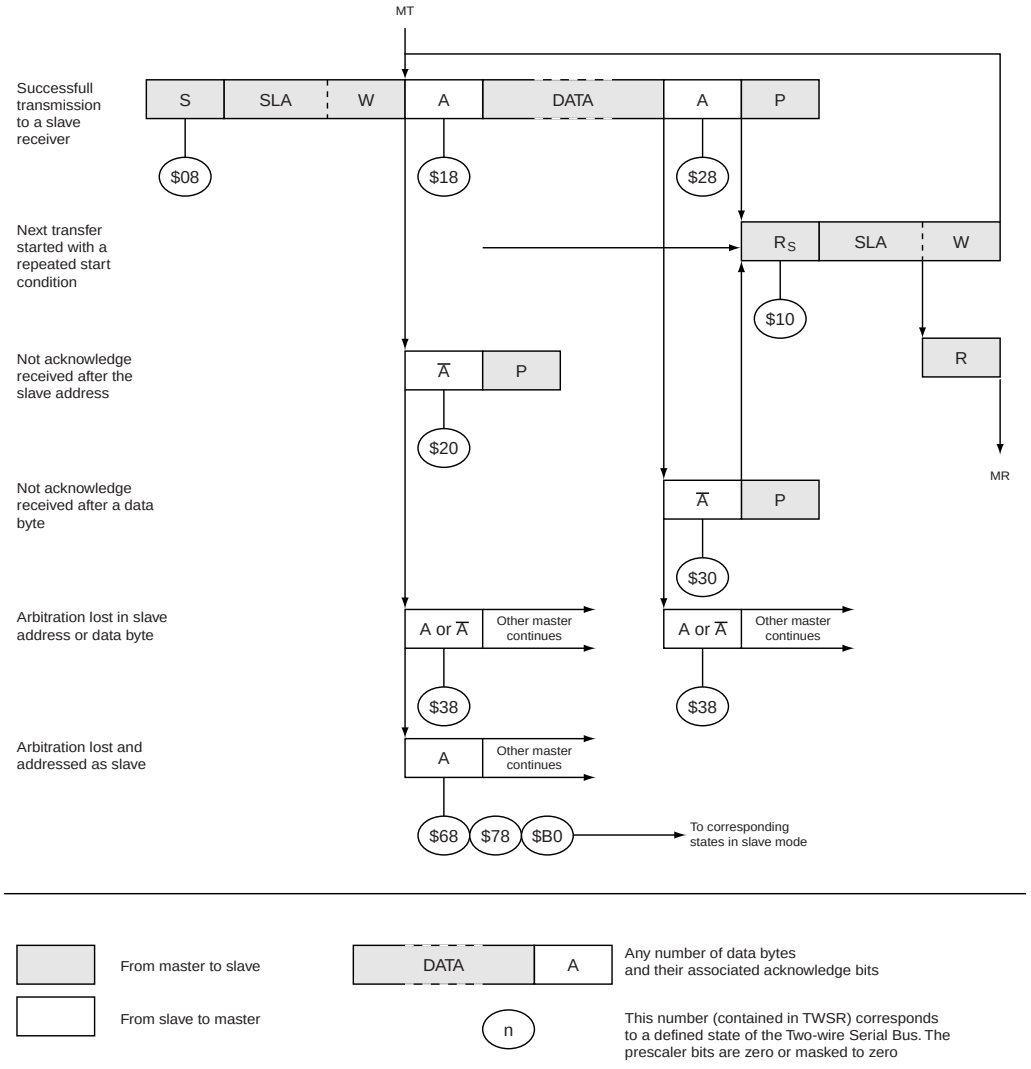
Table 74. 主机发送模式的状态码

状态码 (TWSR) 预分频位为 "0"	2线串行总线和2线串行硬件的状态	应用软件的响应					2 线串行硬件下一步应采取的动作
			对 TWCR 的操作				
		读 / 写 TWDR	STA	STO	TWINT	TWEA	
$08	START 已发送	加载 SLA+W	0	0	1	X	将发送 SLA+W 将接收到 ACK 或 NOT ACK
$10	重复 START 已发送	加载 SLA+W 或 加载 SLA+R	0 0	0 0	1 1	X X	将发送 SLA+W 将接收到 ACK 或 NOT ACK 将发送 SLA+R 切换到主机接收模式
$18	SLA+W 已发送； 接收到 ACK	加载数据 (字节) 或 不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR	0 1 0 1	0 0 1 1	1 1 1 1	X X X X	将发送数据，接收 ACK 或 NOT ACK 将发送重复 START 将发送 STOP， TWSTO 将复位 将发送 STOP，然后发送 START， TWSTO 将复位
$20	SLA+W 已发送 接收到 NOT ACK	加载数据 (字节) 或 不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR	0 1 0 1	0 0 1 1	1 1 1 1	X X X X	将发送数据，接收 ACK 或 NOT ACK 将发送重复 START 将发送 STOP， TWSTO 将复位 将发送 STOP，然后发送 START， TWSTO 将复位

Table 74. 主机发送模式的状态码

$28	数据已发送 接收到 ACK	加载数据 (字节) 或 不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR	0 1 0 1	0 0 1 1	1 1 1 1	X X X X	将发送数据，接收 ACK 或 NOT ACK 将发送重复 START 将发送 STOP， TWSTO 将复位 将发送 STOP，然后发送 START， TWSTO 将复位
$30	数据已发送 接收到 NOT ACK	加载数据 (字节) 或 不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR	0 1 0 1	0 0 1 1	1 1 1 1	X X X X	将发送数据，接收 ACK 或 NOT ACK 将发送重复 START 将发送 STOP， TWSTO 将复位 将发送 STOP，然后发送 START， TWSTO 将复位
$38	SLA+W 或数据的仲裁失败	不操作 TWDR 或 不操作 TWDR	0 1	0 0	1 1	X X	2 线串行总线将被释放，并进入未寻址从机模式 总线空闲后将发送 START

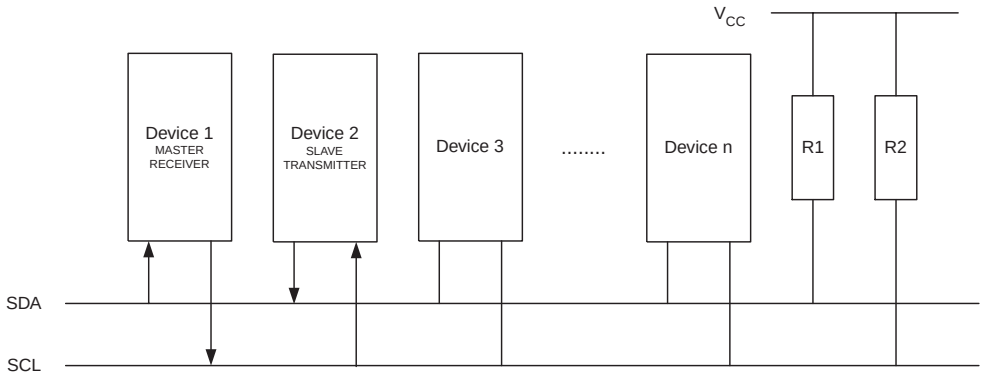
Figure 87. 主机发送模式的格式和状态



主机接收模式

在主机接收模式，主机可以从从机接收数据，如 Figure 88 所示。为进入主机模式，必须发送 START 信号。紧接着的地址包格式决定进入 MT 或 MR 模式。如果发送 SLA+W 进入 MT 模式；如果发送 SLA+R 则进入 MR 模式。本节所提到的状态字均假设其预分频位为 "0”。

Figure 88. 主机接收模式下的数据传输



通过在 TWCR 寄存器中写入下列数值发出 START 信号：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	1	0	X	1	0	X

TWEN必须置位以使能两线接口，TWSTA必须置"1”来发出START 信号且 TWINT必须置"1”来对 TWINT 标志清零。TWI 逻辑开始检测串行总线，一旦总线空闲就发送 START。接着中断标志 TWINT 置位，TWSR 的状态码为 0x08 (见 Table 74)。为进入 MR 模式 ，必须发送 SLA+R。这可通过对 TWDR 写入 SLA+R 来实现。完成此操作后软件清零 TWINT 标志， TWI 传输继续进行。这通过在 TWCR 寄存器中写入下述值完成：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	0	0	X	1	0	X

当 SLA+R 发送完毕并接收到确认信号，主机的 TWINT 标志再次置位。此时主机的 TWSR 状态码可能是 0x38、0x40 或 0x48。对各状态码的正确响应列于 Table 75。TWDR 只有在 TWINT 为高时才能读收到的数据。这过程会一直重复下去，直到最后的字节接收结束。接收完成后， MR 应通过在接收到最后的字节后发送 NACK 信号。发送器产生 STOP 或 REPEATED START 信号结束传送。STOP 信号通过在 TWCR 中写入下述值实现：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	0	1	X	1	0	X

REPEATED START 信号通过在 TWCR 中写入下述值实现：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	1	0	X	1	0	X

在 REPEATED START (状态 0x10) 后，两线接口可以再次访问相同的从机，或不发送 STOP 信号来访问新的从机。REPEATED START 使得主机可以在不丢失总线控制的条件下在从机、主机发送器及主机接收器模式间进行切换。

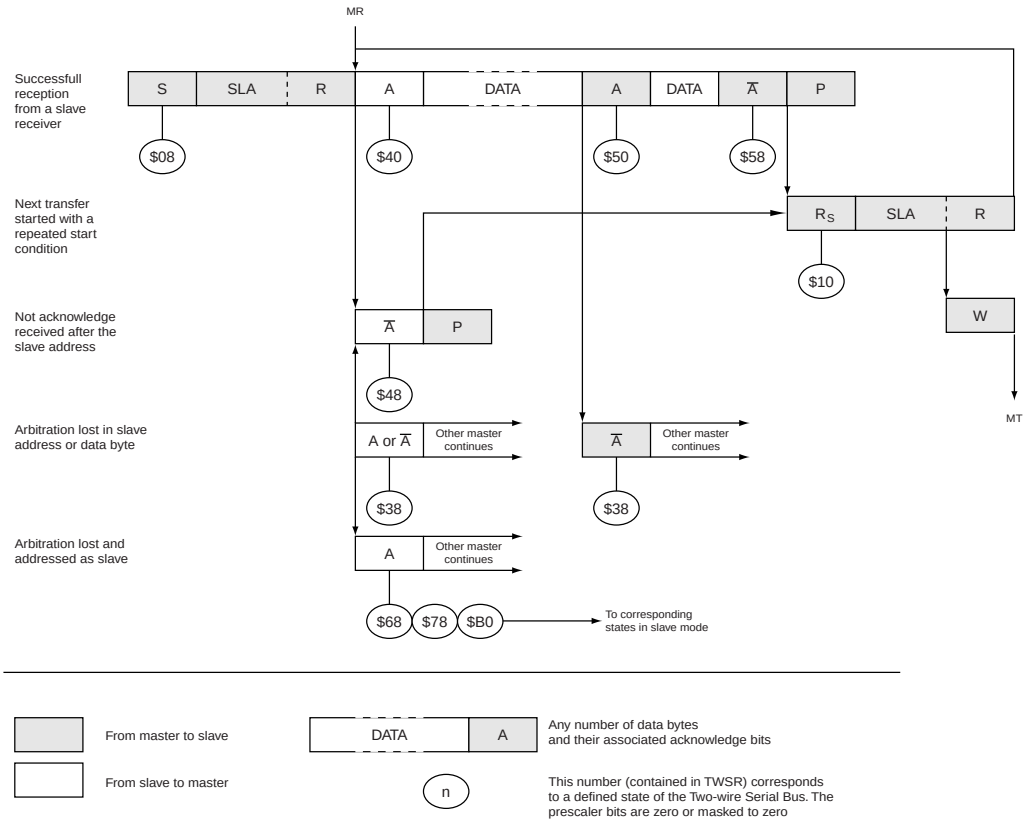
Table 75. 主机接收模式的状态码

状态码 (TWSR) 预分频位为 "0”	2线串行总线和2线串行硬件的状态	应用软件的响应					2 线串行硬件下一步应采取的动作
		读 / 写 TWDR	对 TWCR 的操作				
			STA	STO	TWINT	TWEA	

Table 75. 主机接收模式的状态码 (Continued)

$08	START 已发送	加载 SLA+R	0	0	1	X	将发送 SLA+R 将接收到 ACK 或 NOT ACK
$10	重复 START 已发送	加载 SLA+R 或 加载 SLA+W	0 0	0 0	1 1	X X	将发送 SLA+R 将接收到 ACK 或 NOT ACK 将发送 SLA+W 逻辑切换到主机发送模式
$38	SLA+R 或 NOT ACK 的仲裁失败	不操作 TWDR 或 不操作 TWDR	0 1	0 0	1 1	X X	2 线串行总线将被释放，并进入未寻址从机模式 总线空闲后将发送 START
$40	SLA+R 已发送 接收到 ACK	不操作 TWDR 或 不操作 TWDR	0 0	0 0	1 1	0 1	接收数据，返回 NOT ACK 接收数据，返回 ACK
$48	SLA+R 已发送 接收到 NOT ACK	不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR	1 0 1	0 1 1	1 1 1	X X X	将发送重复 START 将发送 STOP， TWSTO 将复位 将发送 STOP，然后发送 START， TWSTO 将复位
$50	接收到数据 ACK 已返回	读数据或 读数据	0 0	0 0	1 1	0 1	接收数据，返回 NOT ACK 接收数据，返回 ACK
$58	接收到数据 NOT ACK 已返回	读数据或 读数据或 读数据	1 0 1	0 1 1	1 1 1	X X X	将发送重复 START 将发送 STOP， TWSTO 将复位 将发送 STOP，然后发送 START， TWSTO 将复位

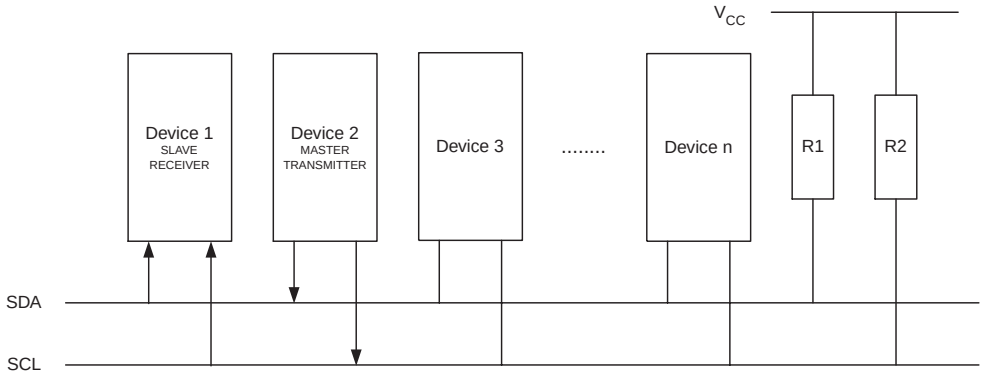
Figure 89. 主机接收模式的格式和状态



从机接收模式

在从机接收模式，从机自主机接收数据，如 Figure 90 所示。本节所提到的状态字均假设其预分频位为 "0”。

Figure 90. 从机接收模式下的数据传输



为启动从机接收模式， TWAR 与 TWCR 设置如下：

TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
值	芯片本身从机地址							

前 7 位是主机寻址时从机响应的 TWI 接口地址。若 LSB 置位，则 TWI 接口响应广播地址 0x00。否则忽略广播地址。

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	0	1	0	0	0	1	0	X

TWEN必须置位以使能TWI接口。TWEA也要置位以使主机寻址到自己(从机地址或广播)时返回确认信息 ACK。 TWSTA 和 TWSTO 必须清零。

初始化 TWAR 和 TWCR 之后， TWI 接口即开始等待，直到自己的从机地址（或广播地址，如果 TWAR 的 TWGCE 置位的话）出现在主机寻址地址当中，并且数据方向位为 0（写）。然后 TWINT 标志置位， TWSR 则包含了相应的状态码。对各状态码的正确响应列于 Table 76。当 TWI 接口处于主机模式（状态 0x68 或 0x78）并发生仲裁失败时 CPU 将进入从机接收模式。

如果在传输过程中 TWEA 复位， TWI 接口在接收到下一个字节后将向 SDA 返回“ 无应答” 。 TWEA 复位时 TWI 接口不再响应自己的从机地址，但是会继续监视总线。一旦TWEA置位就可以恢复地址识别和响应。也就是说，可以利用TWEA暂时将TWI接口从总线中隔离出来。

在除空闲模式外的其它休眠模式时， TWI 接口的时钟被关闭。若使能了从机接收模式，接口将利用总线时钟继续响应广播地址 / 从机地址。地址匹配将唤醒 CPU。在唤醒期间，TWI接口将保持SCL为低电平，直至TWCINT标志清零。当AVR时钟恢复正常运行后TWI可以接收更多的数据。显然如果 AVR 设置为长启动时间，时钟线 SCL 可能会长时间保持低，阻塞其它数据的传送。

当 MCU 从这些休眠模式唤醒时，和正常工作模式不同的是，数据寄存器 TWDR 的数据并不反映总线上出现的最后一个字节。

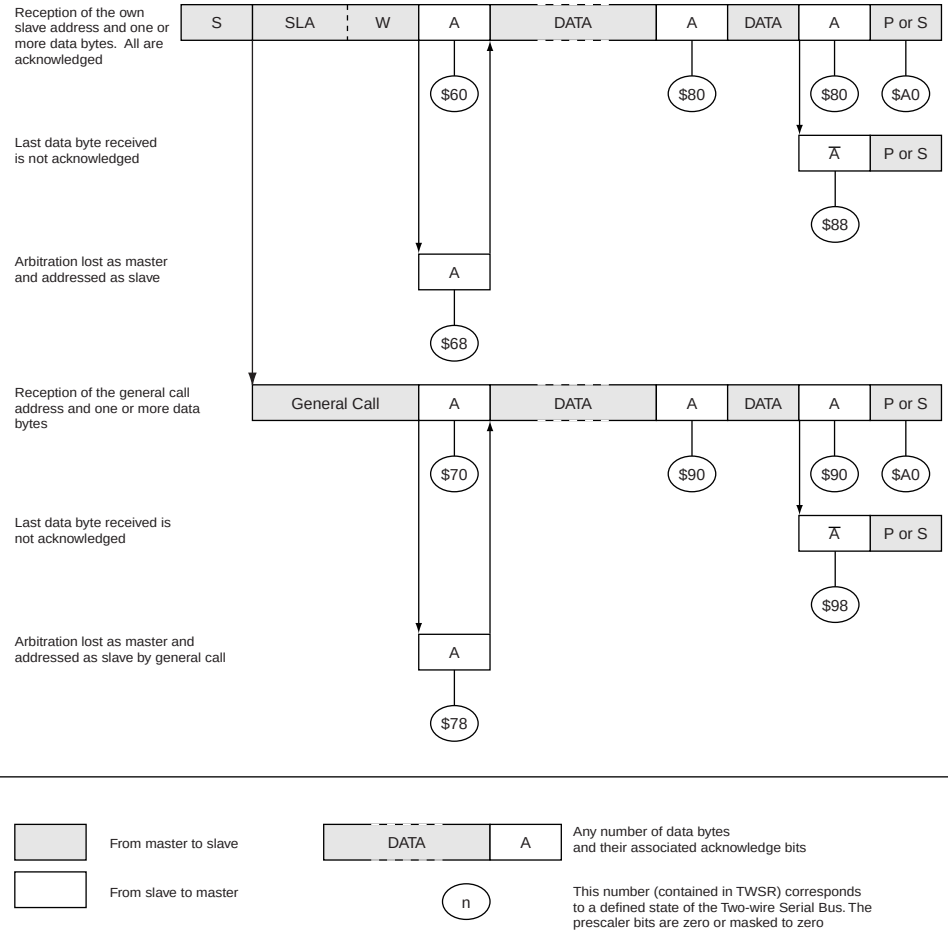
Table 76. 从机接收模式的状态码

状态码 (TWSR) 预分频位为 "0"	2线串行总线和2线串行硬件的状态	应用软件的响应					2 线串行硬件下一步应采取的动作
		读 / 写 TWDR	对 TWCR 的操作				
			STA	STO	TWINT	TWEA	
$60	自己的 SLA+W 已经被接收 ACK 已返回	不操作 TWDR 或	X	0	1	0	接收数据，返回 NOT ACK
		不操作 TWDR	X	0	1	1	接收数据，返回 ACK
$68	SLA+R/W 作为主机的仲裁失败；自己的 SLA+W 已经被接收 ACK 已返回	不操作 TWDR 或	X	0	1	0	接收数据，返回 NOT ACK
		不操作 TWDR	X	0	1	1	接收数据，返回 ACK
$70	接收到广播地址 ACK 已返回	不操作 TWDR 或	X	0	1	0	接收数据，返回 NOT ACK
		不操作 TWDR	X	0	1	1	接收数据，返回 ACK
$78	SLA+R/W 作为主机的仲裁失败；接收到广播地址 ACK 已返回	不操作 TWDR 或	X	0	1	0	接收数据，返回 NOT ACK
		不操作 TWDR	X	0	1	1	接收数据，返回 ACK
$80	以前以自己的 SLA+W 被寻址；数据已经被接收 ACK 已返回	不操作 TWDR 或	X	0	1	0	接收数据，返回 NOT ACK
		不操作 TWDR	X	0	1	1	接收数据，返回 ACK
$88	以前以自己的 SLA+W 被寻址；数据已经被接收 NOT ACK 已返回	读数据或	0	0	1	0	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA
		读数据或	0	0	1	1	切换到未寻址从机模式；能够识别自己的 SLA ；若 TWGCE = “1”， GCA 也可以识别
		读数据或	1	0	1	0	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA ；总线空闲时发送 START
		读数据	1	0	1	1	切换到未寻址从机模式；能够识别自己的 SLA ；若 TWGCE = “1”， GCA 也可以识别；总线空闲时发送 START

Table 76. 从机接收模式的状态码 (Continued)

$90	以前以广播方式被寻址；数据已经被接收 ACK 已返回	读数据或 读数据	X X	0 0	1 1	0 1	接收数据，返回 NOT ACK 接收数据，返回 ACK
$98	以前以广播方式被寻址；数据已经被接收 NOT ACK 已返回	读数据或 读数据或 r 读数据或 读数据	0 0 1 1	0 0 0 0	1 1 1 1	0 1 0 1	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA 切换到未寻址从机模式；能够识别自己的 SLA ；若 TWGCE = “1”， GCA 也可以识别 切换到未寻址从机模式；不再识别自己的 SLA 或 GCA ；总线空闲时发送 START 切换到未寻址从机模式；能够识别自己的 SLA ；若 TWGCE = “1”， GCA 也可以识别；总线空闲时发送 START
$A0	在以从机工作时接收到 STOP 或重复 START	无操作	0 0 1 1	0 0 0 0	1 1 1 1	0 1 0 1	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA 切换到未寻址从机模式；能够识别自己的 SLA ；若 TWGCE = “1”， GCA 也可以识别 切换到未寻址从机模式；不再识别自己的 SLA 或 GCA ；总线空闲时发送 START 切换到未寻址从机模式；能够识别自己的 SLA ；若 TWGCE = “1”， GCA 也可以识别；总线空闲时发送 START

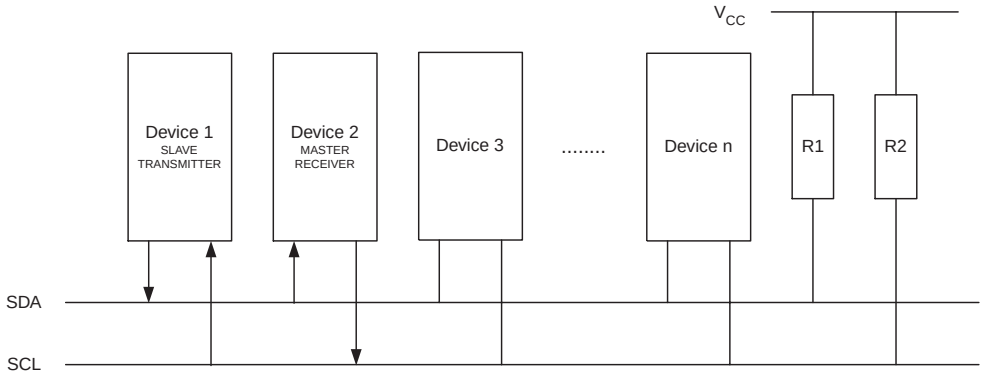
Figure 91. 从机接收模式的格式和状态



从机发送模式

在从机发送模式，从机可以向主机发送数据，如 Figure 92 所示。本节所提到的状态字均假设其预分频位为 "0”。

Figure 92. 从机发送模式下的数据传输



为启动从机发送模式， TWAR 与 TWCR 设置如下：

TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
值	芯片本身从机地址							

前 7 位是主机寻址时从机响应的 TWI 接口地址。若 LSB 置位，则 TWI 接口响应广播地址 0x00。否则忽略广播地址。

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	0	1	0	0	0	1	0	X

TWEN必须置位以使能TWI接口。TWEA也要置位以便主机寻址到自己(从机地址或广播)时返回确认信息 ACK。 TWSTA 和 TWSTO 必须清零。

初始化 TWAR 和 TWCR 之后， TWI 接口即开始等待，直到自己的从机地址（或广播地址，如果 TWAR 的 TWGCE 置位的话）出现在主机寻址地址当中，并且数据方向位为 "1”（读）。然后 TWI 中断标志置位，TWSR 则包含了相应的状态码。对各状态码的正确响应列于 Table 77。当 TWI 接口处于主机模式（状态 0xB0) 并发生仲裁失败时 CPU 将进入从机发送模式。

如果在传输过程中 TWEA 复位，TWI 接口发送完数据之后进入状态 0xC0 或 0xC8。接口也切换到未寻址从机模式，忽略任何后续总线传输。从而主机接收到的数据全为 “1”。如果主机需要附加数据位（通过发送 ACK)，即使从机已经传送结束，也进入状态 0xC8。

TWEA复位时TWI接口不再响应自己的从机地址，但是会继续监视总线。一旦TWEA置位就可以恢复地址识别和响应。也就是说，可以利用 TWEA 暂时将 TWI 接口从总线中隔离出来。

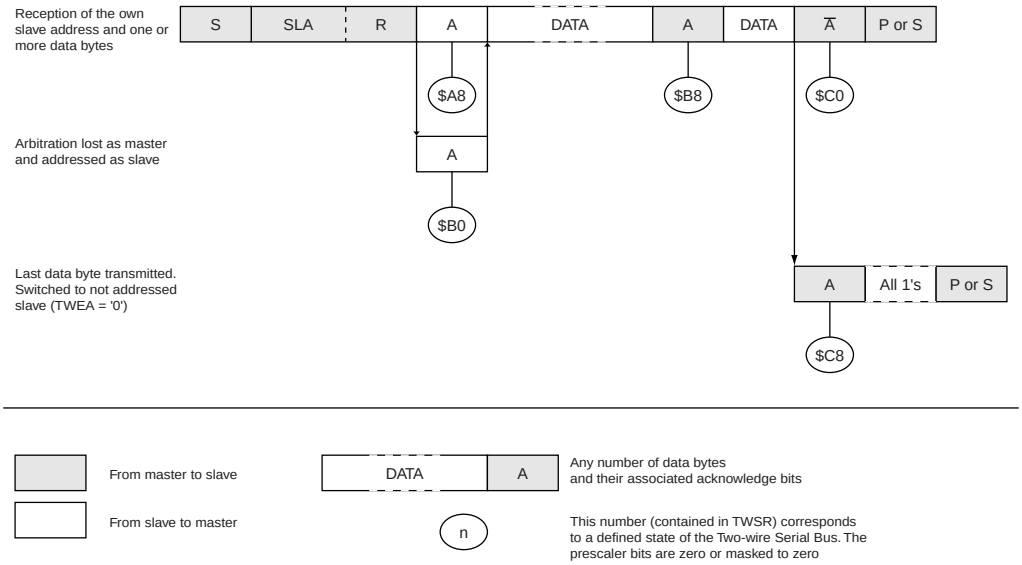
在除空闲模式外的其它休眠模式时， TWI 接口的时钟被关闭。若使能了从机接收模式，接口将利用总线时钟继续响应广播地址 / 从机地址。地址匹配将唤醒 CPU。在唤醒期间，TWI接口将保持SCL为低电平，直至TWCINT标志清零。当AVR时钟恢复正常运行后可以发送更多的数据。显然如果 AVR 设置为长启动时间，时钟线 SCL 可能会长时间保持低，阻塞其它数据的传送。

当 MCU 从这些休眠模式唤醒时，和正常工作模式不同的是，数据寄存器 TWDR 的数据并不反映总线上出现的最后一个字节。

Table 77. 从机发送模式的状态码

状态码 (TWSR) 预分频位为 "0"	2线串行总线和2线串行硬件的状态	应用软件的响应					2 线串行硬件下一步应采取的动作
		读 / 写 TWDR	对 TWCR 的操作				
			STA	STO	TWINT	TWEA	
$A8	自己的 SLA+R 已经被接收 ACK 已返回	加载一字节的数据或	X	0	1	0	发送一字节的数据，接收 NOT ACK
		加载一字节的数据	X	0	1	1	发送数据，接收 ACK
$B0	SLA+R/W 作为主机的仲裁失败；自己的 SLA+R 已经被接收 ACK 已返回	加载一字节的数据或	X	0	1	0	发送一字节的数据，接收 NOT ACK
		加载一字节的数据	X	0	1	1	发送数据，接收 ACK
$B8	TWDR 里数据已经发送 接收到 ACK	加载一字节的数据或	X	0	1	0	发送一字节的数据，接收 NOT ACK
		加载一字节的数据	X	0	1	1	发送数据，接收 ACK
$C0	TWDR 里数据已经发送 接收到 NOT ACK	不操作 TWDR 或	0	0	1	0	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA
		不操作 TWDR 或	0	0	1	1	切换到未寻址从机模式；能够识别自己的 SLA；若 TWGCE = “1”，GCA 也可以识别
		不操作 TWDR 或	1	0	1	0	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA ；总线空闲时发送 START
		不操作 TWDR	1	0	1	1	切换到未寻址从机模式；能够识别自己的 SLA；若 TWGCE = “1”，GCA 也可以识别；总线空闲时发送 START
$C8	TWDR 的一字节数据已经发送 (TWAE = “0”); 接收到 ACK	不操作 TWDR 或	0	0	1	0	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA
		不操作 TWDR 或	0	0	1	1	切换到未寻址从机模式；能够识别自己的 SLA；若 TWGCE = “1”，GCA 也可以识别
		不操作 TWDR 或	1	0	1	0	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA ；总线空闲时发送 START
		不操作 TWDR	1	0	1	1	切换到未寻址从机模式；能够识别自己的 SLA；若 TWGCE = “1”，GCA 也可以识别；总线空闲时发送 START

Figure 93. 从机发送模式的格式和状态



其他状态

有两个状态码没有相应的 TWI 状态定义，见 Table 78。

状态 0xF8 表明当前没有相关信息，因为中断标志 TWINT 为 "0"。这种状态可能发生在自己的 TWI 接口没有参与串行传输的时候。

状态 0x00 表示在串行传输过程中发生了总线错误。当 START 或 STOP 出现在错误的位置时总线错误就会发生。比如说在地址和数据、地址和ACK之间出现了START或STOP。总线错误将导致 TWINT 置位。为了从错误中恢复出来，必须置位标志 TWSTO，并通过写 "1" 以清零 TWINT。这将导致 TWI 接口进入未寻址从机模式、标志 TWSTO 被清零 (TWCR 的其他位不受影响)，以及 SDA 和 SCL 被释放，但是不会产生 STOP。

Table 78. 其它状态码

状态码 (TWSR) 预分频位为 "0"	2线串行总线和2线串行硬件的状态	应用软件的响应					2 线串行硬件下一步应采取的动作
		读 / 写 TWDR	对 TWCR 的操作				
			STA	STO	TWINT	TWEA	
$F8	没有相关的状态信息；TWINT = "0"	不操作 TWDR	No TWCR action				等待或进行当前传输
$00	由于非法的 START 或 STOP 引起的总线错误	不操作 TWDR	0	1	1	X	只影响内部硬件；不会发送 STOP 到总线上。总线将释放并清零 TWSTO

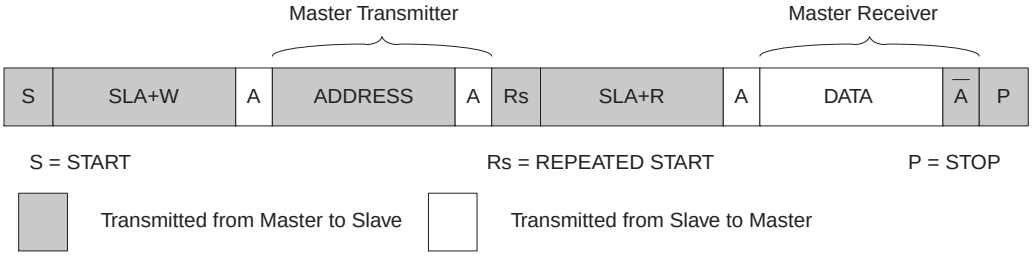
将几个 TWI 模式组合到一起

在某些情况下，为完成期望的工作，必须将几种TWI模式组合起来。例如从串行EEPROM读取数据。典型的这种传输包括以下步骤：

1. 传输必须启动
2. 必须告诉 EEPROM 读取的位置
3. 必须完成读操作
4. 传送必须结束

注意数据可从主机传到从机，反之也可。首先主机必须告诉从机读取实际的位置，因此需要使用 MT 模式；然后数据必须由从机读出，需要使用 MR 模式，但传送方向必须改变。在上述步骤中，主机必须保持对总线的控制，且以上各步骤应该自动进行。如果在多主机系统中违反这一规则，即在第二步与第三步之间其它主机改变 EEPROM 中的数据指针，则主机读取的数据位置是错误的。传送方向改变是通过在发送地址字节与接收数据之间发送 REPEATED START 信号来实现的。在发送 REPEATED START 信号后，主机继续保持总线的控制权。下图给出传送的流程图。

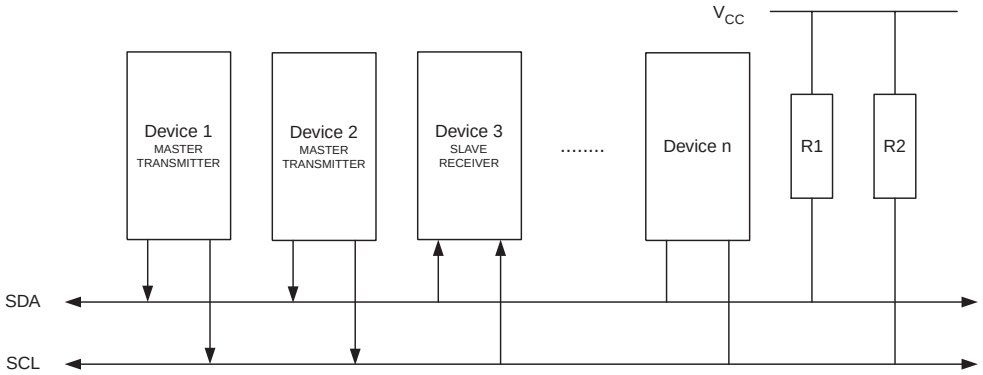
Figure 94. 几种 TWI 模式联合访问串行 EEPROM



多主机系统和仲裁

如果有多个主机连接在同一总线上，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，允许其中的一个主机进行传送而不会丢失数据。总线仲裁的例子如下所述，该例中有两个主机试图向从接收器发送数据。

Figure 95. 仲裁示例



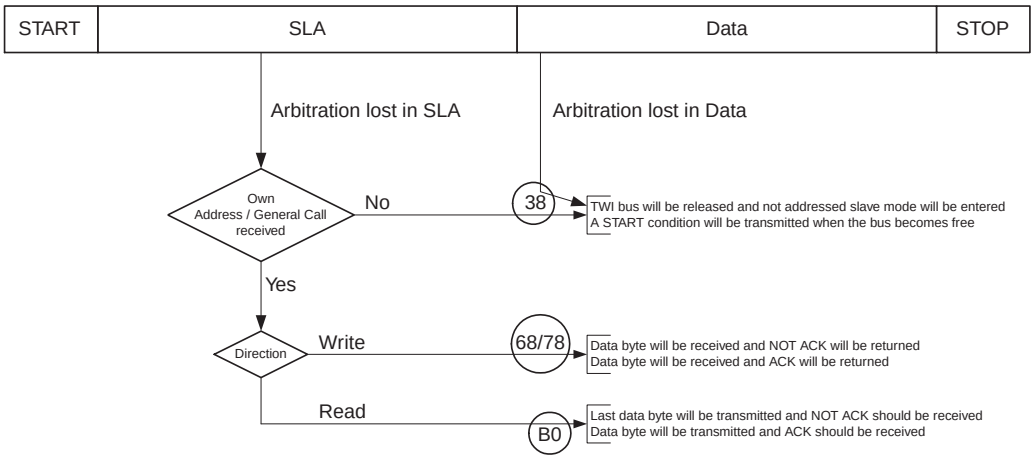
有几种不同的情况会产生总线仲裁过程：

- 两个或更多的主机同时与一个从机进行通信。在这种情况下，无论主机或从机都不知道有总线的竞争。
- 两个或更多的主机同时对同一个从机进行不同的数据或方向的访问。在这种情况下，会在 READ/WRITE 位或数据间发生仲裁。主机试图在 SDA 线上输出一个高电平时，如果其他主机已经输出 "0”，则该主机在总线仲裁中失败。失败的主机将转换成未被寻址的从机模式，或等待总线空闲后发送一个新的 START 信号，这由应用程序决定。

- 两个或更多的主机访问不同的从机。在这种情况下，总线仲裁在 SLA 发生。主机试图在 SDA 线上输出一个高电平时，如有其它主机已经输出 "0”，则该主机将在总线仲裁中失败。在 SLA 总线仲裁失败的主机将切换到从机模式，并检查自己是否被获得总线控制权的主机寻址。如果被寻址，它将进入 SR 或 ST 模式，这取决于 SLA 的 READ/WRITE 位的值。如果它未被寻址，将转换到未被寻址的从机模式或等待总线空闲，发送一个新的 START 信号，这由应用程序决定。

Figure 96 描述了总线仲裁的过程，图中的数字为 TWI 的状态值。

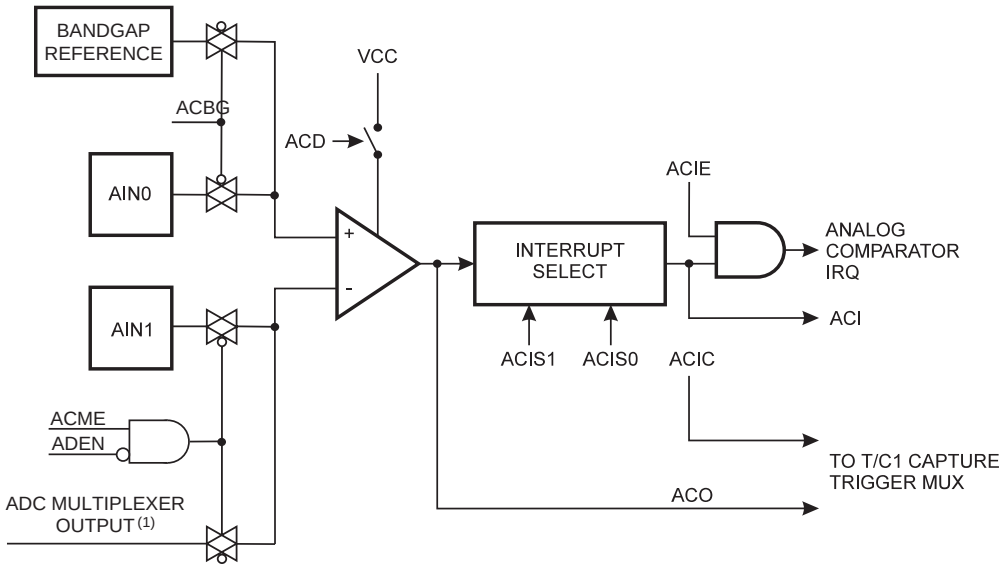
Figure 96. 总线仲裁过程



模拟比较器

模拟比较器对正极 AIN0 的值与负极 AIN1 的值进行比较。当 AIN0 上的电压比负极 AIN1 上的电压要高时，模拟比较器的输出 ACO 即置位。比较器的输出可用来触发定时器 / 计数器 1 的输入捕捉功能。此外，比较器还可触发自己专有的、独立的中断。用户可以选择比较器是以上升沿、下降沿还是交替变化的边沿来触发中断。 Figure 97 为比较器及其外围逻辑电路的框图。

Figure 97. 模拟比较器框图 (2)



Notes: 1. 见 P185 Table 80 。
2. 模拟比较器的管脚分布见 P2 Figure 1 及 P55 Table 25 。

特殊功能 IO 寄存器 - SFIOR

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
读 / 写	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 3 – ACME: 模拟比较器多路复用器使能**

当此位为逻辑 "1”，且 ADC 处于关闭状态 (ADCSRA 寄存器的 ADEN 为 "0”) 时，ADC 多路复用器为模拟比较器选择负极输入。当此位为 "0” 时， AIN1 连接到比较器的负极输入端。更详细描述的请参见 P186 “ 模拟比较器多工输入 ” 。

模拟比较器控制和状态寄存器 - ACSR

Bit	7	6	5	4	3	2	1	0	
	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	
初始值	0	0	N/A	0	0	0	0	0	

- **Bit 7 – ACD: 模拟比较器禁用**

ACD 置位时，模拟比较器的电源被切断。可以在任何时候设置此位来关掉模拟比较器。这可以减少器件工作模式及空闲模式下的功耗。改变 ACD 位时，必须清零 ACSR 寄存器的 ACIE 位来禁止模拟比较器中断。否则 ACD 改变时可能会产生中断。

- **Bit 6 – ACBG: 选择模拟比较器的能隙基准源**

ACBG 置位后，模拟比较器的正极输入由能隙基准源所取代。否则， AIN0 连接到模拟比较器的正极输入。见 P39 “ 片内基准电压 ” 。

- **Bit 5 – ACO: 模拟比较器输出**

模拟比较器的输出经过同步后直接连到 ACO。同步机制引入了 1-2 个时钟周期的延时。

- **Bit 4 – ACI: 模拟比较器中断标志**

当比较器的输出事件触发了由 ACIS1 及 ACIS0 定义的中断模式时，ACI 置位。如果 ACIE 和 SREG 寄存器的全局中断标志 I 也置位，那么模拟比较器中断服务程序即得以执行，同时 ACI 被硬件清零。 ACI 也可以通过写 "1” 来清除。

- **Bit 3 – ACIE: 模拟比较器中断使能**

当 ACIE 位被置 "1” 且状态寄存器中的全局中断标志 I 也被置位时，模拟比较器中断被激活。否则中断被禁止。

- **Bit 2 – ACIC: 模拟比较器输入捕捉使能**

ACIC 置位后允许通过模拟比较器来触发 T/C1 的输入捕捉功能。此时比较器的输出被直接连接到输入捕捉的前端逻辑，从而使得比较器可以利用 T/C1 输入捕捉中断逻辑的噪声抑制器及触发沿选择功能。 ACIC 为 "0” 时模拟比较器及输入捕捉功能之间没有任何联系。为了使比较器可以触发 T/C1 的输入捕捉中断，定时器中断屏蔽寄存器 TIMSK 的 TICIE1 必须置位。

- **Bits 1, 0 – ACIS1, ACIS0: 模拟比较器中断模式选择**

这两位确定触发模拟比较器中断的事件。 Table 79 给出了不同的设置。

Table 79. ACIS1/ACIS0 设置

ACIS1	ACIS0	中断模式
0	0	比较器输出变化即可触发中断
0	1	保留
1	0	比较器输出的下降沿产生中断
1	1	比较器输出的上升沿产生中断

需要改变 ACIS1/ACIS0 时，必须清零 ACSR 寄存器的中断使能位来禁止模拟比较器中断。否则有可能在改变这两位时产生中断。

模拟比较器多工输入

可以选择 ADC7..0 之中的任意一个来代替模拟比较器的负极输入端。ADC 复用器可用来完成这个功能。当然，为了使用这个功能首先必须关掉 ADC。如果模拟比较器复用器使能位 (SFIOR 中的 ACME) 被置位，且 ADC 也已经关掉 (ADCSRA 寄存器的 ADEN 为 0)，则可以通过 ADMUX 寄存器的 MUX2..0 来选择替代模拟比较器负极输入的管脚，详见 Table 80。如果 ACME 清零或 ADEN 置位，则模拟比较器的负极输入为 AIN1。

Table 80. 模拟比较器复用输入

ACME	ADEN	MUX2..0	模拟比较器负极输入
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0
1	0	001	ADC1
1	0	010	ADC2
1	0	011	ADC3
1	0	100	ADC4
1	0	101	ADC5
1	0	110	ADC6
1	0	111	ADC7

模数转换器

特性

- **10 位 精度**
- **0.5 LSB 的非线性度**
- **± 2 LSB 的绝对精度**
- **65 - 260 µs 的转换时间**
- **最高分辨率时采样率高达 15 kSPS**
- **8 路复用的单端输入通道**
- **7 路差分输入通道**
- **2 路可选增益为 10x 与 200x 的差分输入通道** (1)
- **可选的左对齐 ADC 读数**
- **0 - V_{CC} 的 ADC 输入电压范围**
- **可选的 2.56V ADC 参考电压**
- **连续转换或单次转换模式**
- **通过自动触发中断源启动 ADC 转换**
- **ADC 转换结束中断**
- **基于睡眠模式的噪声抑制器**

Note: 1. 在PDIP封装下的差分输入通道器件未经测试。只保证器件在TQFP 与MLF封装下正常工作。

ATmega32有一个10位的逐次逼近型ADC。ADC与一个8通道的模拟多路复用器连接，能对来自端口 A 的 8 路单端输入电压进行采样。单端电压输入以 0V (GND) 为基准。

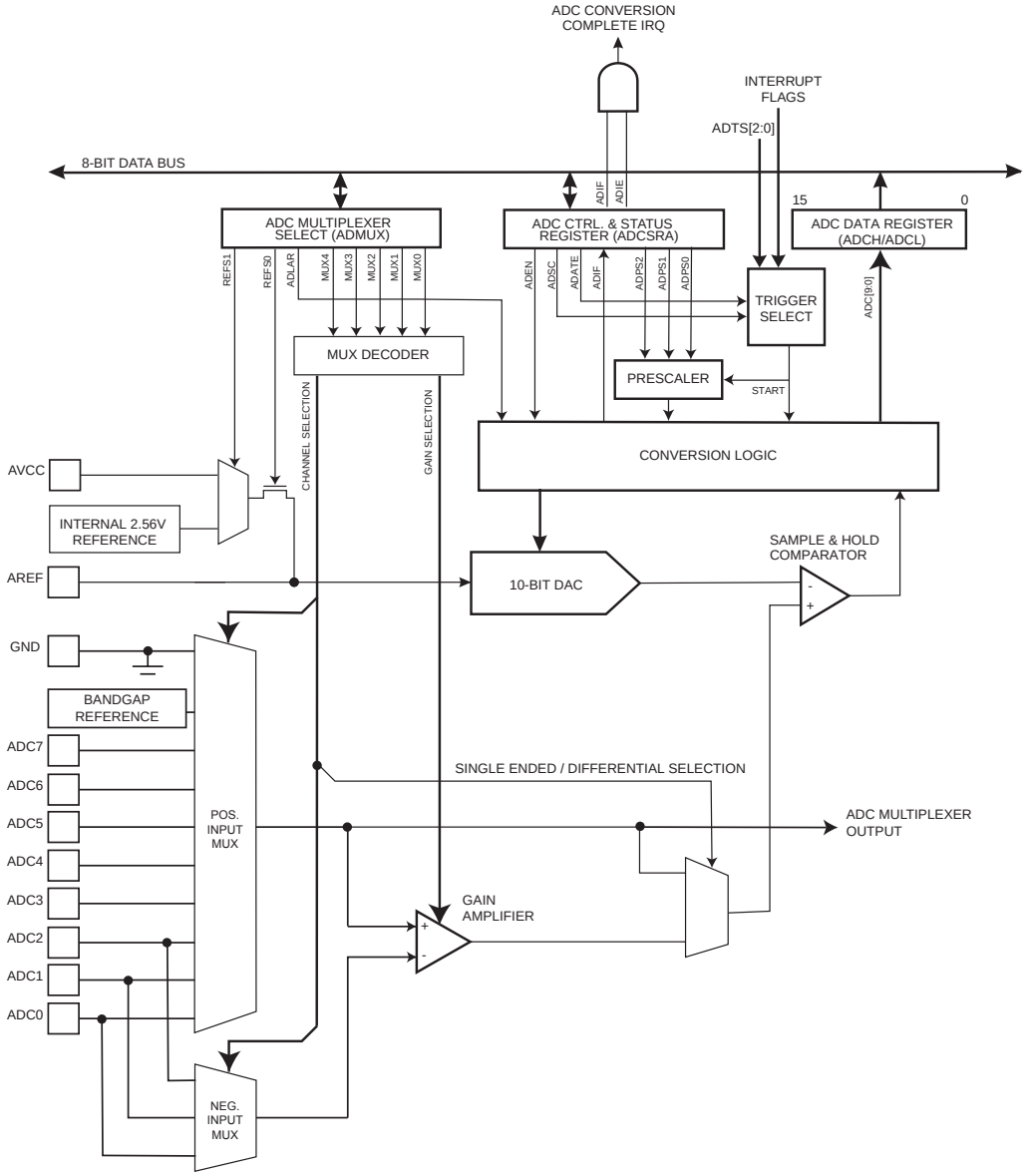
芯片还支持 16 路差分电压输入组合。两路差分输入 (ADC1、 ADC0 与 ADC3、 ADC2) 有可编程增益级，在 A/D 转换前给差分输入电压提供 0 dB (1x)、 20 dB (10x) 或 46 dB (200x) 的放大级。七路差分模拟输入通道共享一个通用负端 (ADC1)，而其他任何 ADC 输入可做为正输入端。如果使用 1x 或 10x 增益，可得到 8 位分辨率。如果使用 200x 增益，可得到 7 位分辨率。

ADC 包括一个采样保持电路，以确保在转换过程中输入到 ADC 的电压保持恒定。ADC 的框图如 Figure 98 所示。

ADC由AVCC引脚单独提供电源。AVCC与V_{CC}之间的偏差不能超过± 0.3V。请参考 P193 “ADC 噪声抑制器 ” 来了解如何连接这个引脚。

标称值为 2.56V 的基准电压，以及 AVCC，都位于器件之内。基准电压可以通过在 AREF 引脚上加一个电容进行解耦，以更好地抑制噪声。

Figure 98. 模数转换器方框图



操作

ADC 通过逐次逼近的方法将输入的模拟电压转换成一个 10 位的数字量。最小值代表 GND，最大值代表 AREF 引脚上的电压再减去 1 LSB。通过写 ADMUX 寄存器的 REFSn 位可以把 AVCC 或内部 2.56V 的参考电压连接到 AREF 引脚。在 AREF 上外加电容可以对片内参考电压进行解耦以提高噪声抑制性能。

模拟输入通道与差分增益可以通过写 ADMUX 寄存器的 MUX 位来选择。任何 ADC 输入引脚，像 GND 及固定能隙参考电压，都可以作为 ADC 的单端输入。ADC 输入引脚可选做差分增益放大器的正或负输入。

如果选择差分通道，通过选择被选输入信号对的增益因子得到电压差分放大级。然后放大值成为 ADC 的模拟输入。如果使用单端通道，将绕过增益放大器。

通过设置 ADCSRA 寄存器的 ADEN 即可启动 ADC。只有当 ADEN 置位时参考电压及输入通道选择才生效。ADEN 清零时 ADC 并不耗电，因此建议在进入节能睡眠模式之前关闭 ADC。

ADC转换结果为10位，存放于ADC数据寄存器ADCH及ADCL中。默认情况下转换结果为右对齐，但可通过设置 ADMUX 寄存器的 ADLAR 变为左对齐。

如果要求转换结果左对齐，且最高只需 8 位的转换精度，那么只要读取 ADCH 就足够了。否则要先读 ADCL，再读 ADCH，以保证数据寄存器中的内容是同一次转换的结果。一旦读出 ADCL， ADC 对数据寄存器的寻址就被阻止了。也就是说，读取 ADCL 之后，即使在读 ADCH 之前又有一次 ADC 转换结束，数据寄存器的数据也不会更新，从而保证了转换结果不丢失。 ADCH 被读出后， ADC 即可再次访问 ADCH 及 ADCL 寄存器。

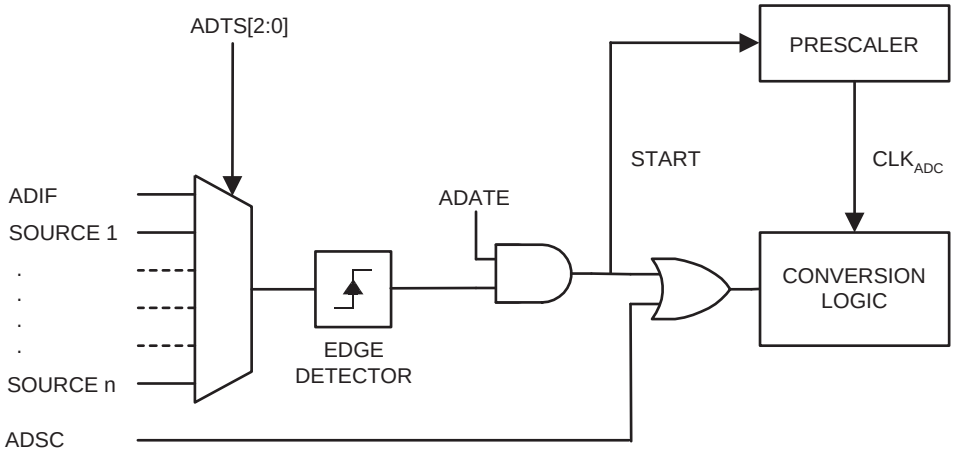
ADC转换结束可以触发中断。即使由于转换发生在读取ADCH与ADCL之间而造成ADC无法访问数据寄存器，并因此丢失了转换数据，中断仍将触发。

启动一次转换

向 ADC 启动转换位 ADSC 位写 "1" 可以启动单次转换。在转换过程中此位保持为高，直到转换结束，然后被硬件清零。如果在转换过程中选择了另一个通道，那么 ADC 会在改变通道前完成这一次转换。

ADC转换有不同的触发源。设置ADCSRA寄存器的ADC自动触发允许位ADATE可以使能自动触发。设置 ADCSRB 寄存器的 ADC 触发选择位 ADTS 可以选择触发源（见触发源列表中对 ADTS 的描述）。当所选的触发信号产生上跳沿时， ADC 预分频器复位并开始转换。这提供了一个在固定时间间隔下启动转换的方法。转换结束后即使触发信号仍然存在，也不会启动一次新的转换。如果在转换过程中触发信号中又产生了一个上跳沿，这个上跳沿将被忽略。即使特定的中断被禁止或全局中断使能位为 0，中断标志仍将置位。这样可以在不产生中断的情况下触发一次转换。但是为了在下次中断事件发生时触发新的转换，必须将中断标志清零。

Figure 99. ADC 自动触发逻辑

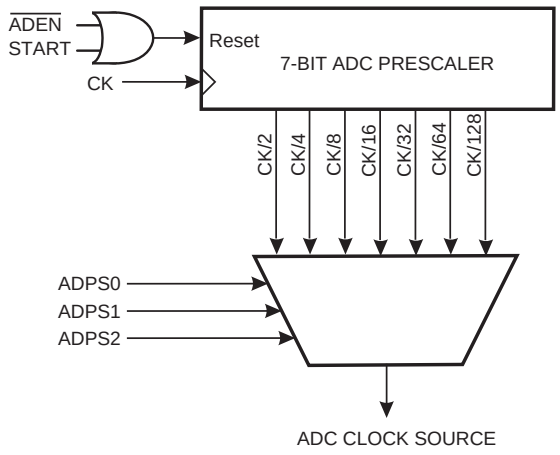


使用 ADC 中断标志作为触发源，可以在正在进行的转换结束后即开始下一次 ADC 转换。之后 ADC 便工作在连续转换模式，持续地进行采样并对 ADC 数据寄存器进行更新。第一次转换通过向 ADCSRA 寄存器的 ADSC 写 1 来启动。在此模式下，后续的 ADC 转换不依赖于 ADC 中断标志 ADIF 是否置位。

如果使能了自动触发，置位 ADCSRA 寄存器的 ADSC 将启动单次转换。 ADSC 标志还可用来检测转换是否在进行之中。不论转换是如何启动的，在转换进行过程中 ADSC 一直为 1。

预分频及 ADC 转换时序

Figure 100. ADC 预分频器



在默认条件下，逐次逼近电路需要一个从50 kHz到200 kHz的输入时钟以获得最大精度。如果所需的转换精度低于 10 比特，那么输入时钟频率可以高于 200 kHz，以达到更高的采样率。

ADC 模块包括一个预分频器，它可以由任何超过 100 kHz 的 CPU 时钟来产生可接受的 ADC 时钟。预分频器通过 ADCSRA 寄存器的 ADPS 进行设置。置位 ADCSRA 寄存器的 ADEN 将使能 ADC，预分频器开始计数。只要 ADEN 为 1，预分频器就持续计数，直到 ADEN 清零。

ADCSRA寄存器的ADSC置位后，单端转换在下一个ADC时钟周期的上升沿开始启动。差分转换时序见 P192 “ 差分增益信道 ” 。

正常转换需要 13 个 ADC 时钟周期。为了初始化模拟电路， ADC 使能 (ADCSRA 寄存器的 ADEN 置位) 后的第一次转换需要 25 个 ADC 时钟周期。

在普通的 ADC 转换过程中，采样保持在转换启动之后的 1.5 个 ADC 时钟开始；而第一次 ADC 转换的采样保持则发生在转换启动之后的 13.5 个 ADC 时钟。转换结束后，ADC 结果被送入 ADC 数据寄存器，且 ADIF 标志置位。ADSC 同时清零 (单次转换模式)。之后软件可以再次置位 ADSC 标志，从而在 ADC 的第一个上升沿启动一次新的转换。

使用自动触发时，触发事件发生将复位预分频器。这保证了触发事件和转换启动之间的延时是固定的。在此模式下，采样保持在触发信号上升沿之后的 2 个 ADC 时钟发生。为了实现同步逻辑需要额外的 3 个 CPU 时钟周期。

使用差分模式，且自动触发源为除 ADC 转换完外时，每次转换要 25 ADC 时钟。这是由于每次转换后 ADC 必须禁用再重新使能。

在连续转换模式下，当 ADSC 为 1 时，只要转换一结束，下一次转换马上开始。转换时间请见 Table 81。

Figure 101. ADC 时序图，第一次转换 (单次转换模式)

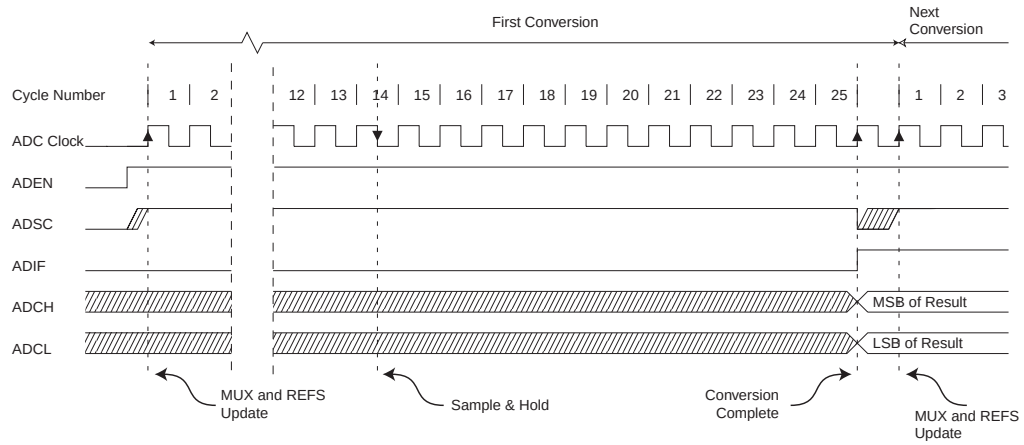


Figure 102. ADC 时序图，单次转换

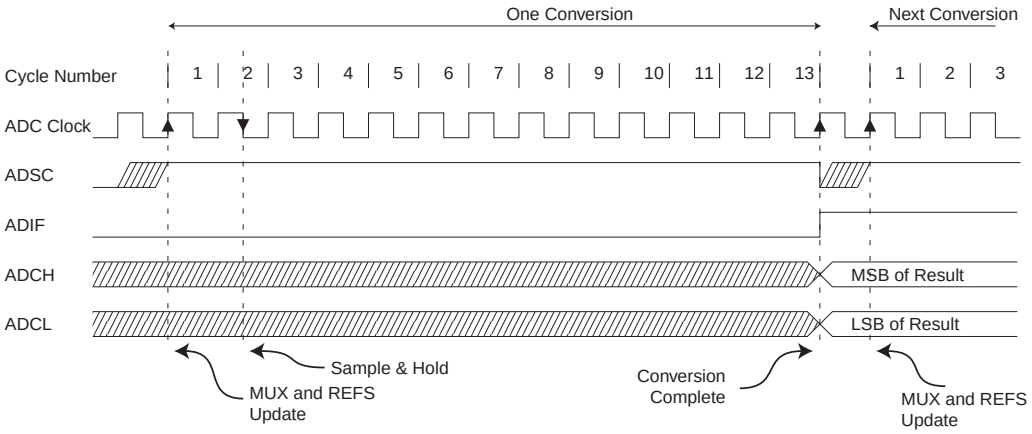


Figure 103. ADC 时序图，自动触发转换

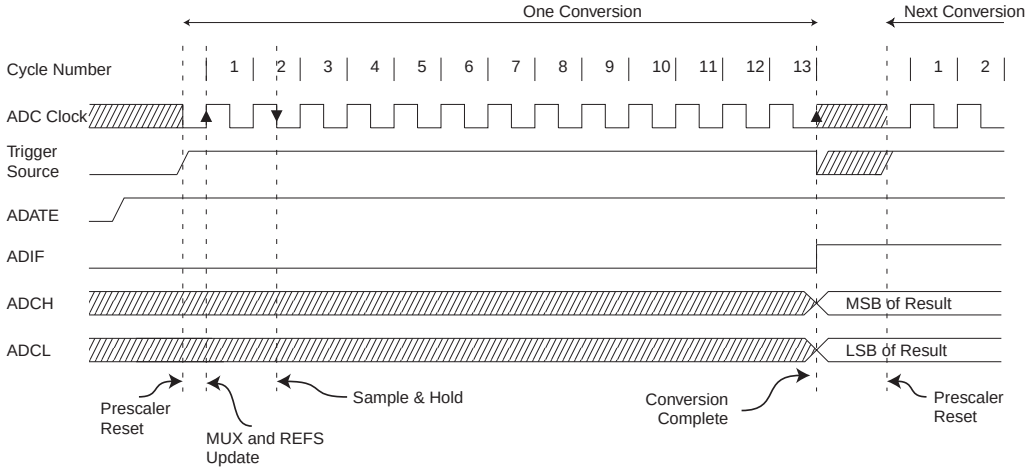


Figure 104. ADC 时序图，连续转换

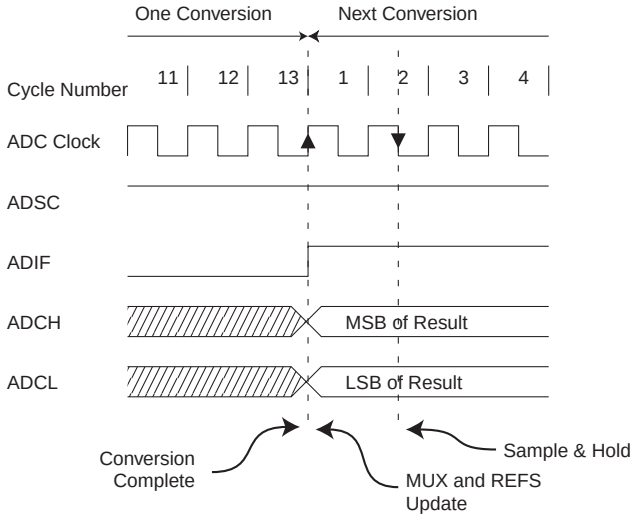


Table 81. ADC 转换时间

条件	采样 & 保持 (启动转换后的时钟周期数)	转换时间 (周期)
第一次转换	14.5	25
正常转换，单端	1.5	13
自动触发转换	2	13.5
正常转换，差分	1.5/2.5	13/14

差分增益信道

当使用差分增益通道，需要考虑转换的确定特征。

差分转换与内部时钟 CK_{ADC2} 同步等于 ADC 时钟的一半。同步是当 ADC 接口在 CK_{ADC2} 边沿出现采样与保持时自动实现的。当 CK_{ADC2} 为低时，通过用户启动转换 (即，所有的单次转换与第一次连续转换) 将与单端转换使用的时间 (接着的预分频后的 13 个 ADC 时钟周期)。当 CK_{ADC2} 为高时，由于同步机制，将会使用 14 个 ADC 时钟周期。在连续转换模式时，一次转换结束后立即启动新的转换，而由于 CK_{ADC2} 此时为高，所有的自动启动 (即除第一次外) 将使用 14 个 ADC 时钟周期。

在所有的增益设置中，当带宽为 4 kHz 时增益级最优。更高的频率可能会造成非线性放大。当输入信号包含高于增益级带宽的频率时，应在输入前加入低通滤波器。注意，ADC 时钟频率不受增益级带宽限制。比如，不管通道带宽是多少， ADC 时钟周期为 6 µs，允许通道采样率为 12 kSPS。

如果使用差分增益通道且通过自动触发启动转换，在转换时 ADC 必须关闭。当使用自动触发时， ADC 预分频器在转换启动前复位。由于在转换前的增益级依靠稳定的 ADC 时钟，该转换无效。在每次转换 (在寄存器 ADCSRA 的 ADEN 位中 写 “0” 接着为 “1”)，通过禁用然后重使能 ADC，只执行扩展转换。扩展转换结果有效。时序图见 P190 “ 预分频及 ADC 转换时序 ” 。

改变通道或基准源

ADMUX寄存器中的MUXn及REFS1:0通过临时寄存器实现了单缓冲。CPU可对此临时寄存器进行随机访问。这保证了在转换过程中通道和基准源的切换发生于安全的时刻。在转换启动之前通道及基准源的选择可随时进行。一旦转换开始就不允许再选择通道和基准源了 ,从而保证 ADC 有充足的采样时间。在转换完成 (ADCSRA 寄存器的 ADIF 置位) 之前的最后一个时钟周期 ,通道和基准源的选择又可以重新开始。转换的开始时刻为 ADSC

置位后的下一个时钟的上升沿。因此，建议用户在置位 ADSC 之后的一个 ADC 时钟周期里，不要操作 ADMUX 以选择新的通道及基准源。

使用自动触发时，触发事件发生的时间是不确定的。为了控制新设置对转换的影响，在更新 ADMUX 寄存器时一定要特别小心。

若ADATE及ADEN都置位，则中断事件可以在任意时刻发生。如果在此期间改变ADMUX寄存器的内容，那么用户就无法判别下一次转换是基于旧的设置还是最新的设置。在以下时刻可以安全地对 ADMUX 进行更新：

1. ADATE 或 ADEN 为 0
2. 在转换过程中，但是在触发事件发生后至少一个 ADC 时钟周期
3. 转换结束之后，但是在作为触发源的中断标志清零之前

如果在上面提到的任一种情况下更新 ADMUX，那么新设置将在下一次 ADC 时生效。

当改变差分通道时要特别注意。一旦选定差分通道，增益级要用 125 µs 来稳定该值。因此在选定新通道后的 125 µs 内不应启动转换。或舍弃该时间段内的转换结果。

当改变 ADC 参考值后 (通过改变 ADMUX 寄存器中的 REFS1:0 位) 的第一次转换也要遵守前面的说明。

ADC 输入通道

选择模拟通道时请注意以下指导方针：

工作于单次转换模式时，总是在启动转换之前选定通道。在 ADSC 置位后的一个 ADC 时钟周期就可以选择新的模拟输入通道了。但是最简单的办法是等待转换结束后再改变通道。

在连续转换模式下，总是在第一次转换开始之前选定通道。在 ADSC 置位后的一个 ADC 时钟周期就可以选择新的模拟输入通道了。但是最简单的办法是等待转换结束后再改变通道。然而，此时新一次转换已经自动开始了，下一次的转换结果反映的是以前选定的模拟输入通道。以后的转换才是针对新通道的。

当切换到差分增益通道，由于自动偏移抵消电路需要沉积时间，第一次转换结果准确率很低。用户最好舍弃第一次转换结果。

ADC 基准电压源

ADC的参考电压源(V_{REF})反映了ADC的转换范围。若单端通道电平超过了V_{REF}，其结果将接近 0x3FF。 V_{REF} 可以是 AVCC、内部 2.56V 基准或外接于 AREF 引脚的电压。

AVCC通过一个无源开关与ADC相连。片内的2.56V参考电压由能隙基准源(V_{BG})通过内部放大器产生。无论是哪种情况，AREF 都直接与 ADC 相连，通过在 AREF 与地之间外加电容可以提高参考电压的抗噪性。 V_{REF} 可通过高输入内阻的伏特表在 AREF 引脚测得。由于 V_{REF} 的阻抗很高，因此只能连接容性负载。

如果将一个固定电源接到 AREF 引脚，那么用户就不能选择其他的基准源了，因为这会导致片内基准源与外部参考源的短路。如果 AREF 引脚没有联接任何外部参考源，用户可以选择 AVCC 或 2.56V 作为基准源。参考源改变后的第一次 ADC 转换结果可能不准确，建议用户不要使用这一次的转换结果。

如果使用差分通道，选择参考电压不应接近如 P273 Table 122 中所示的 AVCC。

ADC 噪声抑制器

ADC的噪声抑制器使其可以在睡眠模式下进行转换，从而降低由于CPU及外围I/O设备噪声引入的影响。噪声抑制器可在 ADC 降噪模式及空闲模式下使用。为了使用这一特性，应采用如下步骤：

1. 确定 ADC 已经使能，且没有处于转换状态。工作模式应该为单次转换，并且 ADC 转换结束中断使能。
2. 进入 ADC 降噪模式 (或空闲模式)。一旦 CPU 被挂起， ADC 便开始转换。

3. 如果在ADC转换结束之前没有其他中断产生，那么ADC中断将唤醒CPU并执行 ADC 转换结束中断服务程序。如果在 ADC 转换结束之前有其他的中断源唤醒了 CPU，对应的中断服务程序得到执行。 ADC 转换结束后产生 ADC 转换结束中断请求。 CPU 将工作到新的休眠指令得到执行。

进入除空闲模式及 ADC 降噪模式之外的其他休眠模式时， ADC 不会自动关闭。在进入这些休眠模式时，建议将 ADEN 清零以降低功耗。如果 ADC 在该休眠模式下使能，且用户要完成差分转换，建议关闭 ADC 且在唤醒后促使外部转换得到有效值。

模拟输入电路

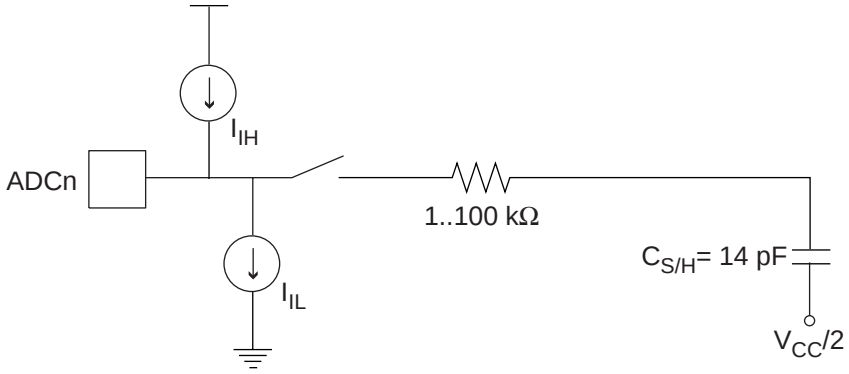
单端通道的模拟输入电路见 Figure 105. 。不论是否用作 ADC 的输入通道，输入到 ADCn 的模拟信号都受到引脚电容及输入泄露的影响。用作 ADC 的输入通道时，模拟信号源必须通过一个串联电阻 (输入通道的组合电阻) 驱动采样保持 (S/H) 电容。

ADC针对那些输出阻抗接近于10 kΩ或更小的模拟信号做了优化。对于这样的信号采样时间可以忽略不计。若信号具有更高的阻抗，那么采样时间就取决于对 S/H 电容充电的时间。这个时间可能变化很大。建议用户使用输出阻抗低且变化缓慢的模拟信号，因为这可以减少对 S/H 电容的电荷传输。

如果使用差分增益通道，输入电路有所不同，建议使用几百 kΩ 的源电阻。

频率高于奈奎斯特频率 ($f_{ADC}/2$) 的信号源不能用于任何一个通道，这样可以避免不可预知的信号卷积造成的失真。在把信号输入到 ADC 之前最好使用一个低通滤波器来滤掉高频信号。

Figure 105. 模拟输入电路

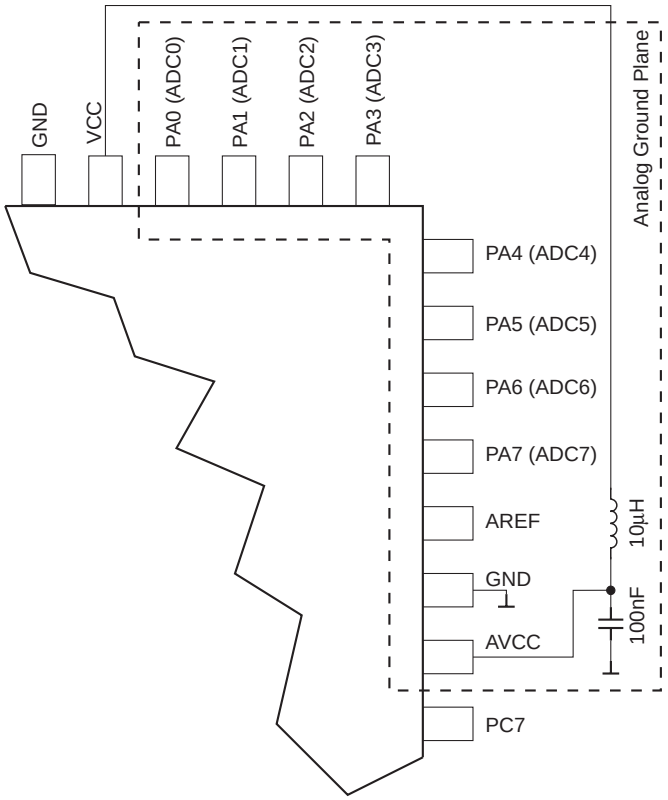


模拟噪声抑制技术

设备内部及外部的数字电路都会产生电磁干扰 (EMI)，从而影响模拟测量的精度。如果转换精度要求较高，那么可以通过以下方法来减少噪声：

1. 模拟通路越短越好。保证模拟信号线位于模拟地之上，并使它们与高速切换的数字信号线分开。
2. 如 Figure 106 所示， AVCC 应通过一个 LC 网络与数字电压源 V_{CC} 连接。
3. 使用 ADC 噪声抑制器来降低来自 CPU 的干扰噪声。
4. 如果有 ADC 端口被用作数字输出，那么必须保证在转换进行过程中它们不会有电平的切换。

Figure 106. ADC 电源连接图



偏置补偿方案

增益级有固定的偏置补偿电路，尽可能降低差分度量偏差。模拟电路中的残余偏差可直接由选择的均为差分输入的相同的通道测得。该值可由软件从测量结果中减去。使用这类基于偏置修正的软件，通道的偏置可降到 1LSB 以下。

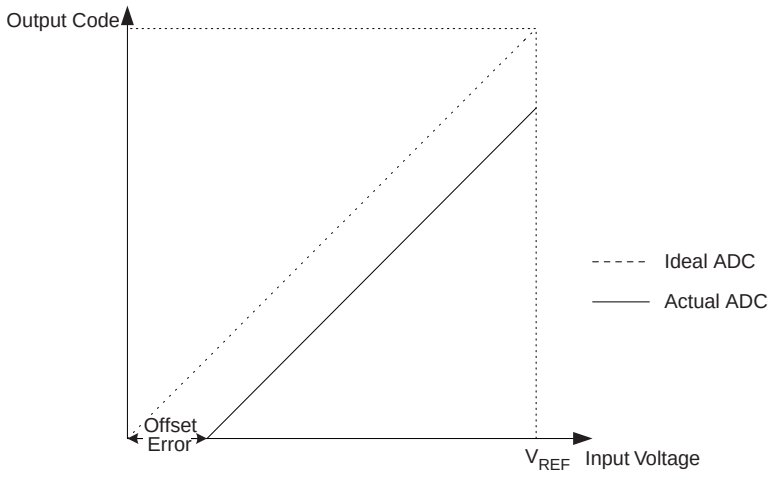
ADC 精度定义

一个 n 位的单端 ADC 将 GND 与 V_{REF} 之间的线性电压转换成 2^n 个 (LSBs) 不同的数字量。最小的转换码为 0，最大的转换码为 2^n-1。

以下几个参数描述了与理想情况之间的偏差：

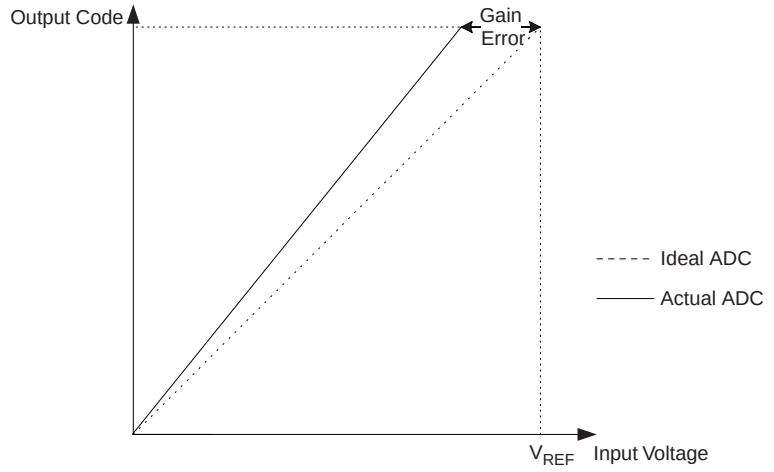
- 偏移：第一次转换 (0x000 到 0x001) 与理想转换 (0.5 LSB) 之间的偏差。理想情况：0 LSB。

Figure 107. 偏移误差



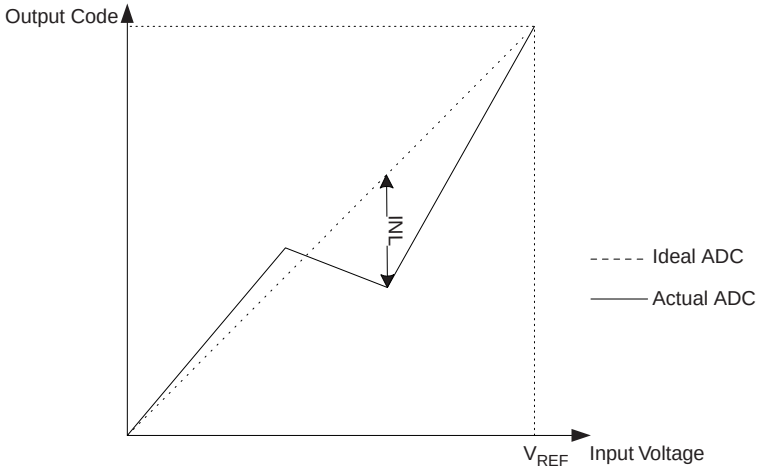
- 增益误差 : 调整偏差之后，最后一次转换 (0x3FE 到 0x3FF) 与理想情况 (最大值以下 1.5 LSB) 之间的偏差即为增益误差。理想值为 0 LSB。

Figure 108. 增益误差



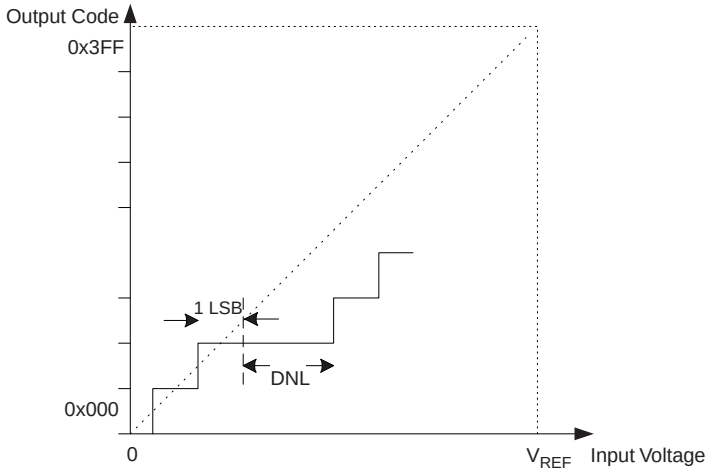
- 整体非线性 (INL)：调整偏移及增益误差之后，所有实际转换与理想转换之间的最大误差即为 INL。理想值：0 LSB。

Figure 109. 整体非线性 (INL)



- 差分非线性(DNL):实际码宽(两个邻近转换之间的码间距)与理论码宽(1 LSB)之间的偏差。理论值：0 LSB。

Figure 110. 差分非线性 (DNL)



- 量化误差：由于输入电压被量化成有限位的数码，某个范围的输入电压 (1 LSB) 被转换为相同的数码。量化误差总是为 ± 0.5 LSB。
- 绝对精度：所有实际转换 (未经调整) 与理论转换之间的最大偏差。由偏移、增益误差、差分误差、非线性及量化误差构成。理想值为 ± 0.5 LSB。

ADC 转换结果

转换结束后 (ADIF 为高)，转换结果被存入 ADC 结果寄存器 (ADCL, ADCH)。

单次转换的结果如下：

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

式中， V_{IN} 为被选中引脚的输入电压， V_{REF} 为参考电压 (参见 P198 Table 83 与 P199 Table 84)。0x000 代表模拟地电平， 0x3FF 代表所选参考电压的数值减去 1LSB。

如果使用差分通道，结果是：

$$ADC = \frac{(V_{POS} - V_{NEG}) \cdot GAIN \cdot 512}{V_{REF}}$$

式中，V_{POS} 为输入引脚正电压，V_{NEG} 为输入引脚负电压，GAIN 为选定的增益因子，且 V_{REF} 为参考电压。结果用 2 的补码形式表示 ,从 0x200 (-512d) 到 0x1FF (+511d)。如果用户希望对结果执行快速极性检测，它充分读结果的 MSB(ADCH 中 ADC9)。如果该位为 1，结果为负；该位为 0，结果为正。 Figure 111 给出差分输入域的解码。

Table 82 给出当选定的增益为 GAIN 且参考电压为 V_{REF} 的差分输入对 (ADCn - ADCm) 的输入码结果。

Figure 111. 差分测量范围

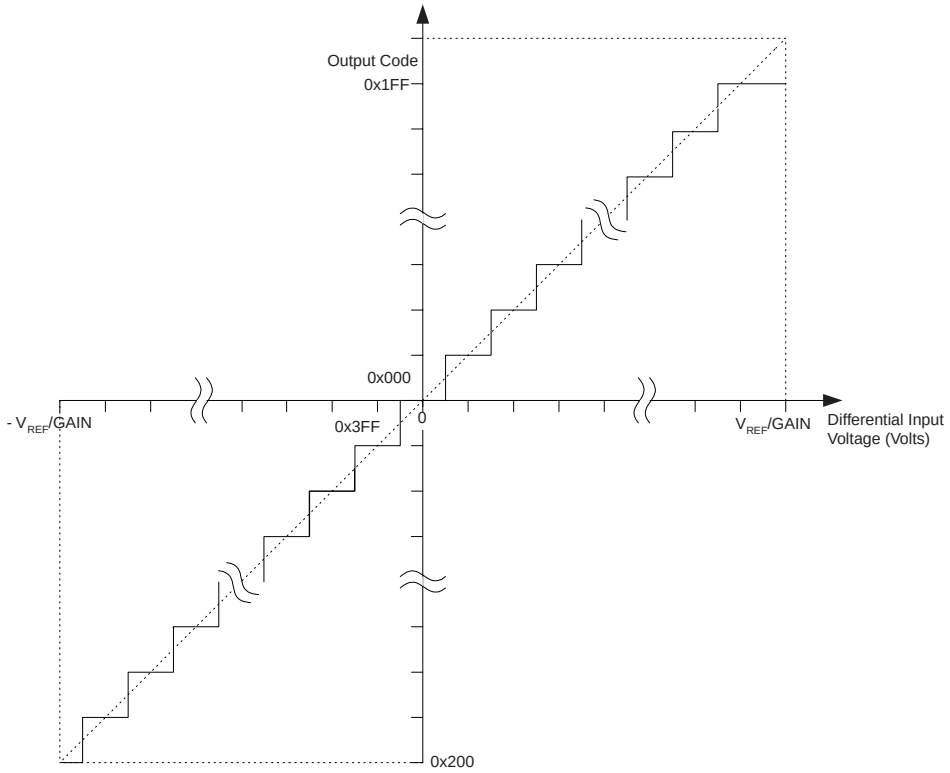


Table 82. 输入电压域输出码的相互关系

V_{ADCn}	读出码	相应十进制值
V_{ADCm} + V_{REF}/GAIN	0x1FF	511
V_{ADCm} + 0.999 V_{REF}/GAIN	0x1FF	511
V_{ADCm} + 0.998 V_{REF}/GAIN	0x1FE	510
...	...	...
V_{ADCm} + 0.001 V_{REF}/GAIN	0x001	1
V_{ADCm}	0x000	0
V_{ADCm} - 0.001 V_{REF}/GAIN	0x3FF	-1
...	...	...
V_{ADCm} - 0.999 V_{REF}/GAIN	0x201	-511
V_{ADCm} - V_{REF}/GAIN	0x200	-512

例：

ADMUX = 0xED (ADC3 - ADC2，10x 增益，2.56V 参考电压，左对齐)。

ADC3 上电压为 300 mV， ADC2 电压为 500 mV。

ADCR = 512 * 10 * (300 - 500) / 2560 = -400 = 0x270。

ADCL 将读为 0x00 ,且 ADCH 读为 0x9C。给 ADLAR 写 0 右对齐:ADCL = 0x70，ADCH = 0x02。

ADC 多工选择寄存器 - ADMUX

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7:6 – REFS1:0: 参考电压选择**

如 Table 83 所示 ,通过这几位可以选择参考电压。如果在转换过程中改变了它们的设置，只有等到当前转换结束 (ADCSRA 寄存器的 ADIF 置位) 之后改变才会起作用。如果在 AREF 引脚上施加了外部参考电压，内部参考电压就不能被选用了。

Table 83. ADC 参考电压选择

REFS1	REFS0	参考电压选择
0	0	AREF，内部 Vref 关闭
0	1	AVCC， AREF 引脚外加滤波电容
1	0	保留
1	1	2.56V 的片内基准电压源， AREF 引脚外加滤波电容

- **Bit 5 – ADLAR: ADC 转换结果左对齐**

ADLAR影响ADC转换结果在ADC数据寄存器中的存放形式。ADLAR置位时转换结果为左对齐，否则为右对齐。 ADLAR 的改变将立即影响 ADC 数据寄存器的内容，不论是否有转换正在进行。关于这一位的完整描述请见 P202 “ADC 数据寄存器 –ADCL 及 ADCH” 。

- **Bits 4:0 – MUX4:0: 模拟通道与增益选择位**

通过这几位的设置，可以对连接到 ADC 的模拟输入进行选择。也可对差分通道增益进行选择。细节见 Table 84。如果在转换过程中改变这几位的值，那么只有到转换结束 (ADCSRA 寄存器的 ADIF 置位) 后新的设置才有效。

Table 84. 输入通道与增益选择

MUX4..0	单端输入	正差分输入	负差分输入	增益
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	N/A	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010(1)		ADC0	ADC0	200x
01011(1)		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110(1)		ADC2	ADC2	200x
01111(1)		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	1.22 V (V_{BG})	N/A		
11111	0 V (GND)			

Note: 1. PDIP 封装器件的差分输入通道未检测。该特性保证 TQFP 与 MLF 封装器件正常工作。

ADC 控制和状态寄存器 A - ADCSRA

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – ADEN: ADC 使能**

ADEN置位即启动ADC ,否则ADC功能关闭。在转换过程中关闭ADC将立即中止正在进行的转换。

- **Bit 6 – ADSC: ADC 开始转换**

在单次转换模式下，ADSC 置位将启动一次 ADC 转换。在连续转换模式下，ADSC 置位将启动首次转换。第一次转换 (在 ADC 启动之后置位 ADSC , 或者在使能 ADC 的同时置位 ADSC) 需要 25 个 ADC 时钟周期，而不是正常情况下的 13 个。第一次转换执行 ADC 初始化的工作。

在转换进行过程中读取 ADSC 的返回值为 "1”, 直到转换结束。ADSC 清零不产生任何动作。

- **Bit 5 – ADATE: ADC 自动触发使能**

ADATE置位将启动ADC自动触发功能。触发信号的上跳沿启动ADC转换。触发信号源通过 SFIOR 寄存器的 ADC 触发信号源选择位 ADTS 设置。

- **Bit 4 – ADIF: ADC 中断标志**

在 ADC 转换结束，且数据寄存器被更新后， ADIF 置位。如果 ADIE 及 SREG 中的全局中断使能位 I 也置位，ADC 转换结束中断服务程序即得以执行，同时 ADIF 硬件清零。此外，还可以通过向此标志写 1 来清 ADIF。要注意的是，如果对 ADCSRA 进行读 - 修改 - 写操作，那么待处理的中断会被禁止。这也适用于 SBI 及 CBI 指令。

- **Bit 3 – ADIE: ADC 中断使能**

若 ADIE 及 SREG 的位 I 置位， ADC 转换结束中断即被使能。

- **Bits 2:0 – ADPS2:0: ADC 预分频器选择位**

由这几位来确定 XTAL 与 ADC 输入时钟之间的分频因子。

Table 85. ADC 预分频选择

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

ADC 数据寄存器 - ADCL 及 ADCH

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
读 / 写	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
初始值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
读 / 写	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
初始值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADC 转换结束后，转换结果存于这两个寄存器之中。如果采用差分通道，结果由 2 的补码形式表示。

读取 ADCL 之后，ADC 数据寄存器一直要等到 ADCH 也被读出才可以进行数据更新。因此，如果转换结果为左对齐，且要求的精度不高于 8 比特，那么仅需读取 ADCH 就足够了。否则必须先读出 ADCL 再读 ADCH。

ADMUX 寄存器的 ADLAR 及 MUXn 会影响转换结果在数据寄存器中的表示方式。如果 ADLAR 为 1，那么结果为左对齐；反之 (系统缺省设置)，结果为右对齐。

- **ADC9:0: ADC 转换结果**

ADC 转换的结果，细节见 P198 “ADC 转换结果 ” 。

特殊功能 IO 寄存器 - SFIOR

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
读 / 写	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7:5 – ADTS2:0: ADC 自动触发源**

若 ADCSRA 寄存器的 ADATE 置位，ADTS 的值将确定触发 ADC 转换的触发源；否则，ADTS 的设置没有意义。被选中的中断标志在其上升沿触发 ADC 转换。从一个中断标志清零的触发源切换到中断标志置位的触发源会使触发信号产生一个上升沿。如果此时

ADCSRA 寄存器的 ADEN 为 1 ,ADC 转换即被启动。切换到连续运行模式 (ADTS[2:0]=0) 时，即使 ADC 中断标志已经置位也不会产生触发事件。

Table 86. ADC 自动触发源选择

ADTS2	ADTS1	ADTS0	触发源
0	0	0	连续转换模式
0	0	1	模拟比较器
0	1	0	外部中断请求 0
0	1	1	定时器 / 计数器 0 比较匹配
1	0	0	定时器 / 计数器 0 溢出
1	0	1	定时器 / 计数器比较匹配 B
1	1	0	定时器 / 计数器 1 溢出
1	1	1	定时器 / 计数器 1 捕捉事件

- **Bit 4 – Res: 保留位**

这一位保留。写操作时时这一位应写 0。

JTAG 接口和片上调试系统 OCD(On-chip Debug)

特点

- 与 IEEE 1149.1 标准兼容的 JTAG 接口
- 遵从 IEEE 1149.1 (JTAG) 标准的边界扫描功能
- 调试器可以访问：
  - 所有的片内外设
  - 内部和外部 SRAM
  - 寄存器文件
  - 程序计数器
  - EEPROM 和 Flash 存储器
  - 扩展 OCD 支持各种断点，包括
  - AVR *Break* 指令
  - 程序流程改变
  - 单步
  - 单个地址或一个地址范围的程序断点
  - 单个地址或一个地址范围的数据断点
- 通过 JTAG 接口对 Flash、 EEPROM、熔丝位和锁定位进行编程
- AVR Studio® 支持 OCD

综述

与 IEEE 1149.1 标准兼容的 AVR JTAG 接口可用于

- 通过 JTAG 边界扫描功能测试 PCB
- 对非易失性存储器、熔丝位和锁定位进行编程
- 片上调试 OCD

下面为 JTAG 接口的简短描述。有关通过 JTAG 接口进行编程、使用边界扫描链的具体说明请分别参见 P256 “ 通过 JTAG 接口进行编程 ” ,以及 P210 “IEEE 1149.1 (JTAG) 边界扫描 ” 。片内调试功能 OCD 以专用的 JTAG 指令实现，只在 ATMEL 内部分发。ATMEL 将有选择地支持第三方 JTAGICE 生产商。

Figure 112为JTAG接口及OCD系统的框图。TAP控制器为受TCK和TMS信号控制的状态机。 TAP 控制器或者选择 JTAG 指令寄存器，或者选择数据寄存器作为 TDI(输入) 和 TDO(输出) 之间的扫描链 (移位寄存器)。指令寄存器保存了控制数据寄存器行为的 JTAG 指令。

ID 寄存器、旁路 (Bypass) 寄存器和边界扫描链组成了用于板级测试的数据寄存器。JTAG 编程接口 (包括几个物理的和虚拟的数据寄存器) 用于串行编程。片内扫描链和断点扫描链只用于 OCD 功能。

测试访问端口 - TAP

JTAG接口有四个引脚。以JTAG的术语来说 ,这些引脚组成了测试访问端口TAP。这些引脚是：

- TMS：测试模式选择。此引脚用来实现 TAP 控制器各个状态之间的切换。
- TCK：测试时钟。 JTAG 操作是与 TCK 同步的。
- TDI:测试数据输入--需要移位到指令寄存器或数据寄存器 (扫描链) 的串行输入数据。
- TDO：测试数据输出 -- 自指令寄存器或数据寄存器串行移出的数据。

ATmega32 没有实现 IEEE 1149.1 标准指定的可选 TAP 信号 TRST - Test ReSeT。

在 JTAGEN 熔丝位没有被编程的情况下，四个 TAP 引脚为正常的端口引脚， TAP 控制器处于复位状态。一旦 JTAGEN 被编程，且 MCUCSR 寄存器的 JTD 清零，TAP 输入信号即被拉高， JTAG 边界扫描和编程功能使能。此时 TAP 输出 (TDO) 处于悬空状态，JTAG TAP 控制器不移位数据，因此必须连接一个上拉电阻或有上拉电阻的硬件 (如扫描链中下一个器件的 TDI 输入)。芯片付运时这个熔丝位被编程。

对于片上调试系统，除了 JTAG 接口引脚外，调试器还监控 $\overline{\text{RESET}}$ 引脚，以便检测外部复位源。如果复位引脚是以漏极 (集电极) 开路的方式驱动的，调试器也可以拉低 $\overline{\text{RESET}}$ 引脚来复位整个系统。

Figure 112. 方框图

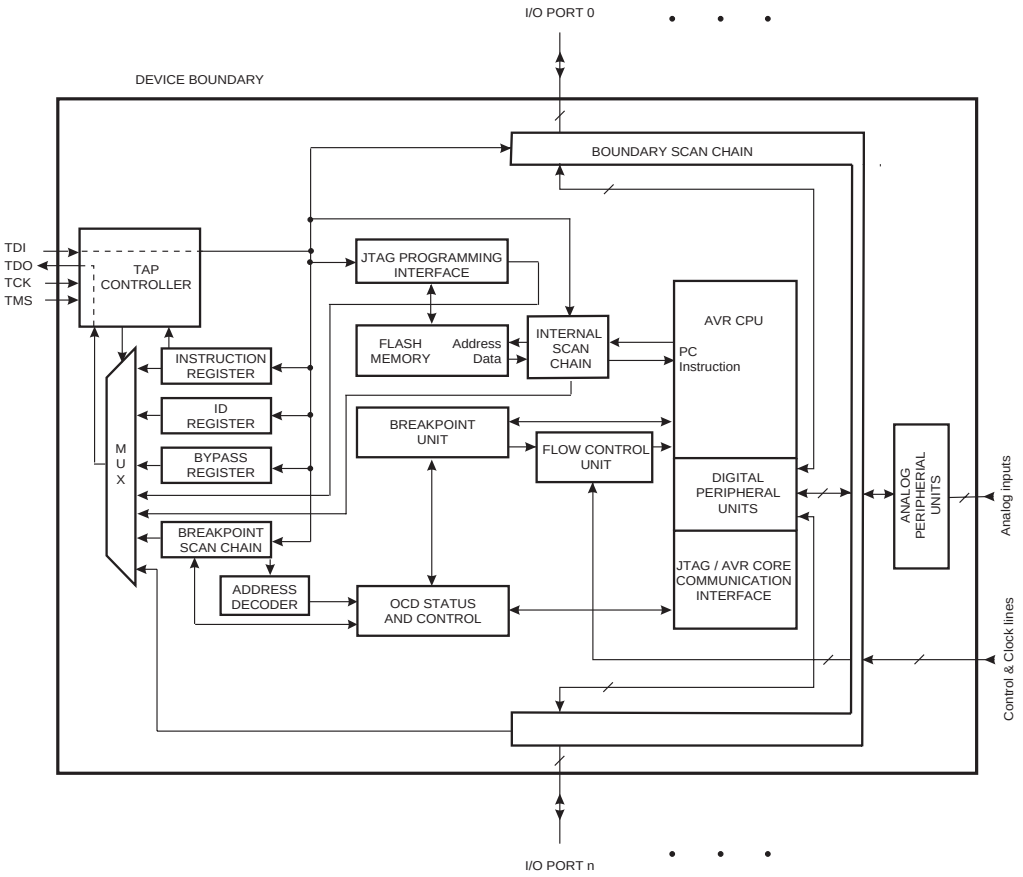
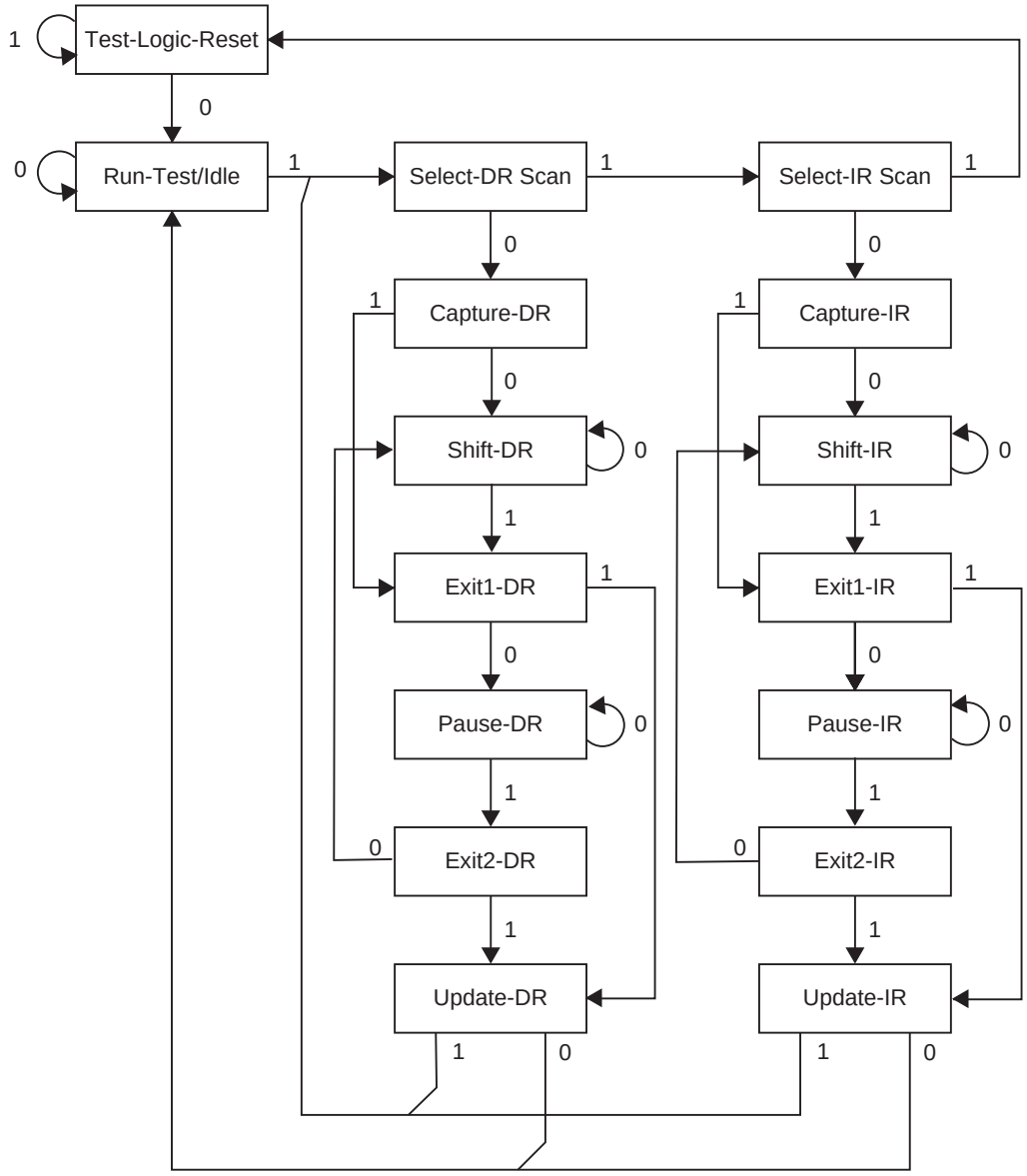


Figure 113. TAP 控制器状态图



TAP 控制器

TAP 控制器是具有 16 个状态的有限状态机 , 它控制着边界扫描电路、JTAG 编程电路 , 即片上调试系统。 Figure 113 中描述的状态转换取决于 TCK 上升沿时的 TMS(显示于靠近状态转换的地方) 信号。上电复位后的初始状态是 Test-Logic-Reset：测试逻辑复位。

对于所有的移位寄存器， LSB 是第一个进行移出和移入的比特位。

假设当前状态为 Run-Test/Idle(运行 - 测试 / 空闲) , 典型的 JTAG 接口使用过程可以描述如下：

- 在 TCK 的上升沿通过 TMS 引脚顺序输入 1, 1, 0, 0 , 从而进入移位指令寄存器 – Shift-IR 状态。然后在 TCK 的上升沿通过 TDI 输入的 4 比特的 JTAG 指令到 JTAG 指令寄存器。在输入 3 个 LSB 的过程中， TMS 必须保持低电平以保持 Shift-IR 状态。指令的 MSB 位则是在拉高 TMS 离开 Shift-IR 状态的时候移入 JTAG 指令寄存器。该指令从 TDI 引脚移入时，捕获的 IR 状态 0x01 则通过 TDO 引脚串行移出。 JTAG 指令选择一个特定的数据寄存器作为 TDI 引脚到 TDO 引脚的通路，并且控制所选数据寄存器的外围电路。

- 对 TMS引脚输入序列 1, 1, 0以再次进入Run-Test/Idle状态。指令在Update-IR状态通过移位寄存器锁存到并行输出。Exit-IR, Pause-IR, 和 Exit2-IR 状态只用来操纵状态机。
- 在 TCK 的上升沿对 TMS 施加信号 1, 0, 0 以进入移位数据寄存器 – Shift-DR 状态。在此状态下， TDI 输入数据在 TCK 的上升沿加载到被选定的数据寄存器 (通过 JTAG 指令寄存器的 JTAG 指令选定)。为了保持 Shift-DR 状态， TMS 输 入必须在除 MSB 以外的所有比特输入过程中保持为低电平。在 TMS 设置成高电平以离开 Shift-IR 状态时，输入数据的 MSB 进入数据寄存器。当数据从 TDI 引脚移入数据寄存器时，在 Capture-DR 状态下捕获的、以并行方式输入到数据寄存器的数据从 TDO 引脚移出。
- 对 TMS 引脚输入序列 1, 1, 0 以再次进入 Run-Test/Idle 状态。如果所选的数据寄存器有被锁存的并行输出，那么锁存动作发生于 Update-DR 状态。 Exit-IR, Pause-IR, 和 Exit2-IR 状态用来操纵状态机。

如状态表所示，在选择 JTAG 指令和使用数据寄存器之间不必进入 Run-Test/Idle 状态。一些 JTAG 指令可以在 Run-Test/Idle 状态选择一定的功能并运行。当然此时的状态就不适合称作空闲状态了。

Note: TMS保持五个TCK时钟周期的高电平可以使TAP控制器进入Test-Logic-Reset状态。此操作与 TAP 控制器的初始状态无关。

JTAG 规范的详细说明清参见 P209 “ 参考文献 ” 。

使用边界扫描链

边界扫描特性的完整描述在 P210 “IEEE 1149.1 (JTAG) 边界扫描 ” 中给出。

使用片上调试系统

如 Figure 112 所示，支持片上调试的硬件主要由以下几部分组成：

- AVR CPU 和内部外围单元接口上的扫描链
- 断点单元
- CPU 和 JTAG 系统之间的通信接口

实现调试器的所有读操作和修改 / 写操作都要通过内部 AVR CPU 扫描链运行 AVR 指令得以实现。CPU 再将结果送入 I/O 存储器。此存储器是 CPU 和 JTAG 系统间通信接口的一部分。

断点单元可以实现程序流程改变断点、单步断点、两个程序存储器断点以及两个混合断点。将它们组合在一起可以得到如下的配置：

- 四个程序存储器断点
- 三个程序存储器断点加上一个数据存储器断点
- 两个程序存储器断点加上两个数据存储器断点
- 两个程序存储器断点加上一个可以屏蔽的程序存储器断点 (区间断点)
- 两个程序存储器断点加上一个可以屏蔽的数据存储器断点 (区间断点)

然而，像 AVR Studio 这样的调试器为了其内部目的，可能要使用一个或更多的资源。这样就减少了使最终用户的自由度。

片上调试的专有 JTAG 指令在 P208 “ 片上调试专用的 JTAG 指令 ” 给出。

必须对 JTAGEN 熔丝位进行编程才能使能 JTAG 测试访问端口。此外还必须编程熔丝位 OCDEN，并保持所有的锁定位处于非锁定状态，才能真正使片上调试系统工作。作为片上调试系统的安全特性，在设置了任一个锁定位时片上调试系统被禁止。否则，片上调试系统就会给安全器件留下后门。

Atmel的AVR JTAG ICE是使用IEEE 1149.1协议 ,兼容 JTAG接口的针对全系列AVR8位微控制器的片上调试开发工具。 JTAG ICE 与 AVR Studio 用户接口向用户提供完整的微

控制器内部资源控制权，简化调试以缩短调试时间。JTAG ICE 在目标系统中运行时实现实时仿真。

完整的 AVR JTEG ICE 资料请访问 www.atmel.com AVR 主页的 Support Tools 部分。同时还可从 Software 部分中免费下载 AVR Studio。

AVR Studio包含了所有必须的运行命令，不论是源代码级还是汇编级。用户可以运行诸如单步、跟踪进入、运行到光标处、停止、复位等各种与其他调试器完全相同的操作。此外，用户还可以通过 BREAK 指令设置无限多个程序断点，以及设置两个数据存储器断点，或组成一个屏蔽 (范围) 断点。

片上调试专用的 JTAG 指令

支持片上调试的 JTAG 指令在规范中并没有指定，是 ATMEL 的单独实现，仅在 ATMEL 内部和选定的第三方分发。指令操作码如下。

PRIVATE0; $8 访问片上调试系统的非规范 JTAG 指令。

PRIVATE1; $9 访问片上调试系统的非规范 JTAG 指令。

PRIVATE2; $A 访问片上调试系统的非规范 JTAG 指令。

PRIVATE3; $B 访问片上调试系统的非规范 JTAG 指令。

I/O 存储器里与片上调试相关的寄存器

片上调试寄存器 - OCDR

Bit	7	6	5	4	3	2	1	0	
	MSB/IDRD							LSB	OCDR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

OCDR 寄存器为运行于微控制器的程序和调试器提供了一个通信信道。CPU 可以通对该字节进行写操作将一个字节传输到调试器。与此同时， I/O 调试寄存器脏位 – IDRD – 被置位，以此来告知调试器寄存器已经被进行了写操作。CPU 读取 OCDR 寄存器时，7 个 LSB 来自于 OCDR 寄存器，而 MSB 是 IDRD 位。调试器读完信息后清除 IDRD 位。

在一些 AVR 器件中，这个寄存器与标准 I/O 存储单元共享。此时，只有在熔丝位 OCDEN 被编程，并且调试器访问 OCDR 寄存器功能被使能， MCU 访问的才是 OCDR 寄存器。在其他情况下， MCU 访问的则是标准的 I/O 存储单元。

关于如何使用这个寄存器的信息请参见调试器文档。

利用 JTAG 的可编程能力

通过 JTAG 对 AVR 器件进行编程仅需要使用 JTAG 端口的四个引脚—TCK, TMS, TDI 和 TDO(除了电源引脚)。编程不需要额外的12V电压。使能JTAG测试访问端口需要首先编程 JTAGEN 熔丝位，同时保证 MCUCR 寄存器的 JTD 被清零。

JTAG 编程支持：

- Flash 编程和校验
- EEPROM 编程和校验
- 熔丝位编程和校验
- 锁定位编程和校验

锁定位的安全性和并行编程模式的完全一样。如果锁定位 LB1 或 LB2 被编程，则 OCDEN 熔丝位不能被编程，除非首先进行芯片擦除。这个安全特性确保在读取加密的器件时没有后门。

通过JTAG接口编程和专用JTAG编程指令的详细内容在P256 “通过JTAG接口进行编程” 。

参考文献

更多关于边界扫描的知识可以参考下面的文献：

- IEEE: IEEE Std 1149.1-1990. IEEE Standard Test Access Port and Boundary-scan Architecture, IEEE, 1993
- Colin Maunder: The Board Designers Guide to Testable Logic Circuits, Addison-Wesley, 1992

IEEE 1149.1 (JTAG) 边界扫描

特点

- **JTAG 接口 (与 IEEE std. 1149.1 兼容标准)**
- **根据 JTAG 标准实现的边界扫描功能**
- **可以对所有端口功能以及具有片外连接的模拟电路进行完整的扫描**
- **支持可选的 IDCODE 指令**
- **用于复位 AVR 的 AVR_RESET 指令**

系统概述

边界扫描链可以驱动和观察数字 I/O 引脚的逻辑电平，以及具有片外连接的模拟电路的数字逻辑和模拟逻辑的边界。在系统一级，所有具有 JTAG 功能的 IC 都通过 TDI/TDO 信号串行连接，以形成一个长的移位寄存器。外部控制器设置这些器件去驱动输出引脚，并观察从其他芯片接收到的输入值。控制器对接收到的数据和期望值进行比较。通过这种方式，边界扫描只使用四个 TAP 信号就实现了印刷电路板上元器件的互连及完整性测试。

四个 IEEE 1149.1 定义的必须实现的 JTAG 指令 IDCODE， BYPASS， SAMPLE/PRELOAD 和 EXTEST，以及只有 AVR 拥有的 JTAG 指令 AVR_RESET，可以用来测试印刷电路板。由于 IDCODE 是 JTAG 的默认指令，数据寄存器通路的初始扫描将会显示该芯片的 ID。在测试模式期间可能希望 AVR 处于复位状态。如果没有复位，该器件的输入将由扫描操作决定。退出测试模式时，内部软件可能处于不确定状态。进入复位时，任何端口引脚的输出将会立刻进入高阻态，从而实现了 HIGHZ 指令的冗余。必要的话，可以发出旁路指令 BYPASS，使芯片的扫描链尽可能的短。通过拉低 $\overline{\text{RESET}}$ 引脚或通过对复位数据寄存器的适当设置发出 AVR_RESET 指令可以将芯片设置成复位状态。

EXTEST 指令用来抽样引脚信号以及对输出引脚加载数据。一旦 EXTEST 被加载到 JTAG IR 寄存器，来自输出贮存器的数据就会被输出到这些引脚上。因此必须使用 SAMPLE/PRELOAD 指令来设置扫描环路的初始值以避免首次使用 EXTEST 指令时破坏电路板。 SAMPLE/PRELOAD 还用来在芯片正常工作时提取外部引脚信号的快照。

必须编程 TAGEN 熔丝位并清零 I/O 寄存器 MCUCR 的 JTD 位，从而使能 JTAG 测试访问端口。

使用 JTAG 接口进行边界扫描时， JTAG TCK 时钟频率不可以高于芯片内部频率。但是并不要求芯片时钟出于工作状态。

数据寄存器

与边界扫描操作相关的数据寄存器是：

- 旁路 (Bypass) 寄存器
- 器件识别寄存器
- 复位寄存器
- 边界扫描链

旁路寄存器

旁路寄存器由移位寄存器组成。当旁路寄存器被选择作为 TDI 和 TDO 之间的通路时，离开 Capture-DR 控制器状态将使其复位为 0。测试其他器件时，旁路寄存器可以用来缩短系统的扫描链。

器件识别寄存器

Figure 114 给出了器件识别寄存器的结构。

Figure 114. 器件识别寄存器的格式

- **版本**

版本号是一个四位的数字，用来鉴别芯片的修订版本。相关版本号见 Table 87。

Table 87. JTAG 版本号

版本	JTAG 版本号 (Hex)
ATmega32 修订版 A	0x0

- **芯片型号**

芯片型号是用来识别器件的 16 位代码。Table 88 列出了 ATmega32 的 JTAG 器件型号。

Table 88. AVR JTAG 器件型号

芯片型号	JTAG 芯片型号 (Hex)
ATmega32	0x9502

- **制造商 ID**

制造商 ID 是用来区别厂商的 11 位代码。Table 89 中列出了 ATMEL 的 JTAG 制造商 ID。

Table 89. 制造商 ID

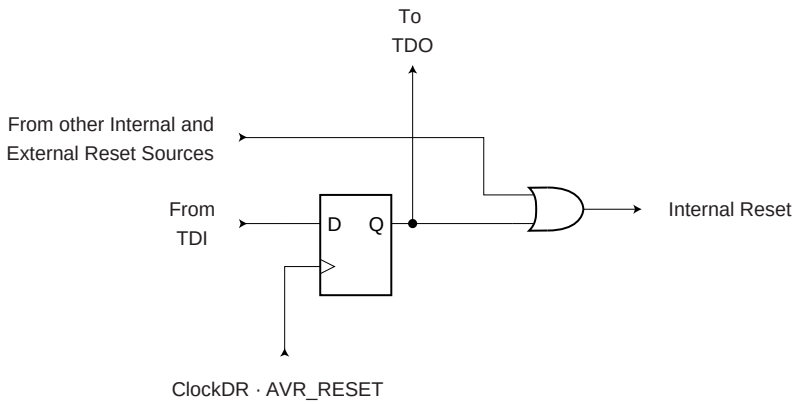
制造商	JTAG 制造商 ID (Hex)
ATMEL	0x01F

复位寄存器

复位寄存器是用来复位芯片的测试数据寄存器。由于复位时 AVR 端口引脚为三态，复位寄存器还可以代替未实现的可选 JTAG 指令 HIGHZ 的功能。

复位寄存器不为零时相当于将外部复位引脚拉低。根据熔丝位对于时钟选择的设置，释放复位寄存器后器件会保持复位状态一个复位溢出时间 (参见 P23 “ 时钟源 ”)。这个数据寄存器的输出没有锁存，所以复位可以立刻发生，如 Figure 115 所示。

Figure 115. 复位寄存器



边界扫描链

边界扫描链可以驱动和获知数字 I/O 引脚的逻辑电平，以及具有片外连接的模拟电路的数字逻辑和模拟逻辑的边界。

完整地描述参见 P214 “ 边界扫描链 ” 。

用于边界扫描的 JTAG 指令

指令寄存器为 4 比特，支持多达 16 条指令。下面列出的是与边界扫描操作相关的 JTAG 指令。 AVR 没有实现可选的 HIGHZ 指令，但是 AVR_RESET 指令可以使所有端口引脚成为高阻态。

对于所有的移位寄存器，数据的 LSB 首先被移入或移出。

每条指令的 OPCODE 都以 hex 格式显示在指令名称的下面。文本则描述了哪个数据寄存器被选作每条指令的 TDI 和 TDO 之间的路径。

EXTEST; $0

EXTEST 是必须实现的 JTAG 指令，用来选择边界扫描链作为数据寄存器，为 AVR 外部的测试电路提供数据。通过扫描链可以实现端口引脚的禁止上拉电阻、输出控制、输出数据及输入数据功能。对于具有片外连接的模拟电路来说，模拟与数字逻辑之间的接口也位于扫描链之中。一旦 JTAG IR 寄存器通过 EXTEST 指令被加载，边界扫描链锁存输出的内容立即被驱动到输出引脚。

工作状态有：

- Capture-DR：取样外部引脚的数据并输入扫描链
- Shift-DR：通过 TCK 移位内部扫描链
- Update-DR：来自扫描链的数据应用到输出引脚上

IDCODE; $1

IDCODE 是可选的 JTAG 指令，用来选择 32 位 ID 寄存器作为数据寄存器。ID 寄存器由版本号、芯片型号备号和由 JEDEC 确定的制造商号组成。 IDCODE 是上电后的默认指令。

工作状态有：

- Capture-DR：获取 IDCODE 寄存器的数据并输入到边界扫描链
- Shift-DR：通过 TCK 移位 IDCODE 扫描链

SAMPLE_PRELOAD; $2

SAMPLE_PRELOAD 是必须实现的 JTAG 指令，用来在不影响系统工作的前提下，预加载输出锁存以及为输入 / 输出引脚获取快照。但是输出锁存并没有连接到引脚上。边界扫描链被选作数据寄存器。

工作状态有：

- Capture-DR：取样外部引脚的数据并输入扫描链
- Shift-DR：通过 TCK 移位内部扫描链
- Update-DR：扫描链的数据应用到输出锁存。但是输出锁存并没有连接到引脚上。

AVR_RESET; $C

AVR_RESET 是 AVR 特有 JTAG 指令，用来强制 AVR 芯片进入复位模式或释放 JTAG 复位源。 TAP 控制器不会被这条指令复位。这个只有一个比特有效数据的字节被选作数据寄存器。只要复位链为中逻辑 1，复位即被激活。该扫描链的输出不被锁存。

工作状态有：

- Shift-DR：通过 TCK 移位复位寄存器

BYPASS; $F

BYPASS 是必须实现的 JTAG 指令，用来选择旁路寄存器作为数据寄存器。

工作状态有：

- Capture-DR：在旁路寄存器中载入逻辑 0
- Shift-DR：TDI 和 TDO 之间的旁路寄存器单元被移位

I/O 存储器中与边界扫描相关的寄存器

MCU 控制与状态寄存器 - MCUCSR

MCU 控制与状态寄存器包含控制通用 MCU 功能的控制位。还提供 MCU 复位的复位源信息。

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	**–**	**JTRF**	**WDRF**	**BORF**	**EXTRF**	**PORF**	**MCUCSR**
读 / 写	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	见位说明					

- **Bit 7 – JTD: 禁止 JTAG 接口**

此位为 0 时，如果 JTAGEN 熔丝位被编程则 JTAG 接口使能。如果这位为 1， JTAG 接口禁止。为了避免无意的禁止或使能 JTAG 接口，必须通过一个时间序列来改变 JTD 位。应用软件必须在四个时钟周期内将期望的数值两次写入 JTD。

如果 JTAG 接口没有与其他 JTAG 电路连接， JTD 应该置位。这样做的原因是为了避免 JTAG 接口 TDO 引脚的静态电流。

- **Bit 4 – JTRF: JTAG 复位标志位**

通过 JTAG 指令 AVR_RESET 为 JTAG 复位寄存器置 1 可以复位芯片。芯片复位后 JTRF 置位。上电复位或写入逻辑 0 将清零 JTRF。

边界扫描链

边界扫描链可以驱动和获知数字 I/O 引脚的逻辑电平，以及具有片外连接的模拟电路的数字逻辑和模拟逻辑的边界。

扫描数字端口引脚

Figure 116 显示了具有上拉功能的双向引脚的边界扫描单元。该单元包括一个服务于上拉使能 – PUExn – 功能的标准边界扫描单元，和一个将三个信号：输出控制 – OCxn、输出数据 – ODxn 和输入数据 – IDxn 组合为一个两位移位寄存器的双向引脚单元构成。在下面的叙述中不使用端口和引脚索引。

数据手册的插图没有给出边界扫描逻辑。 Figure 117 给出了一个如 P47 “I/O 端口 ” 所述的简单数字端口引脚。 Figure 117 的虚线框包含了 Figure 116 的边界扫描细节。

当端口没有第二功能时，输入数据 – ID – 对应于 PINxn 寄存器的值 (但 ID 没有同步器)，输出数据对应于端口寄存器 PORT，输出控制对应于数据方向 – DD 寄存器，上拉使能 – PUExn – 对应于逻辑表达式 $\overline{PUD} \cdot \overline{DDxn} \cdot PORTxn$。

端口第二功能在 Figure 117 中连接到虚线框之外，这样可以使扫描链读到实际的引脚值。对于模拟功能，外部引脚和模拟电路有直接连接，扫描链可以嵌入到数字逻辑和模拟电路之间的接口上。

Figure 116. 具有上拉功能双向端口引脚的边界扫描 .

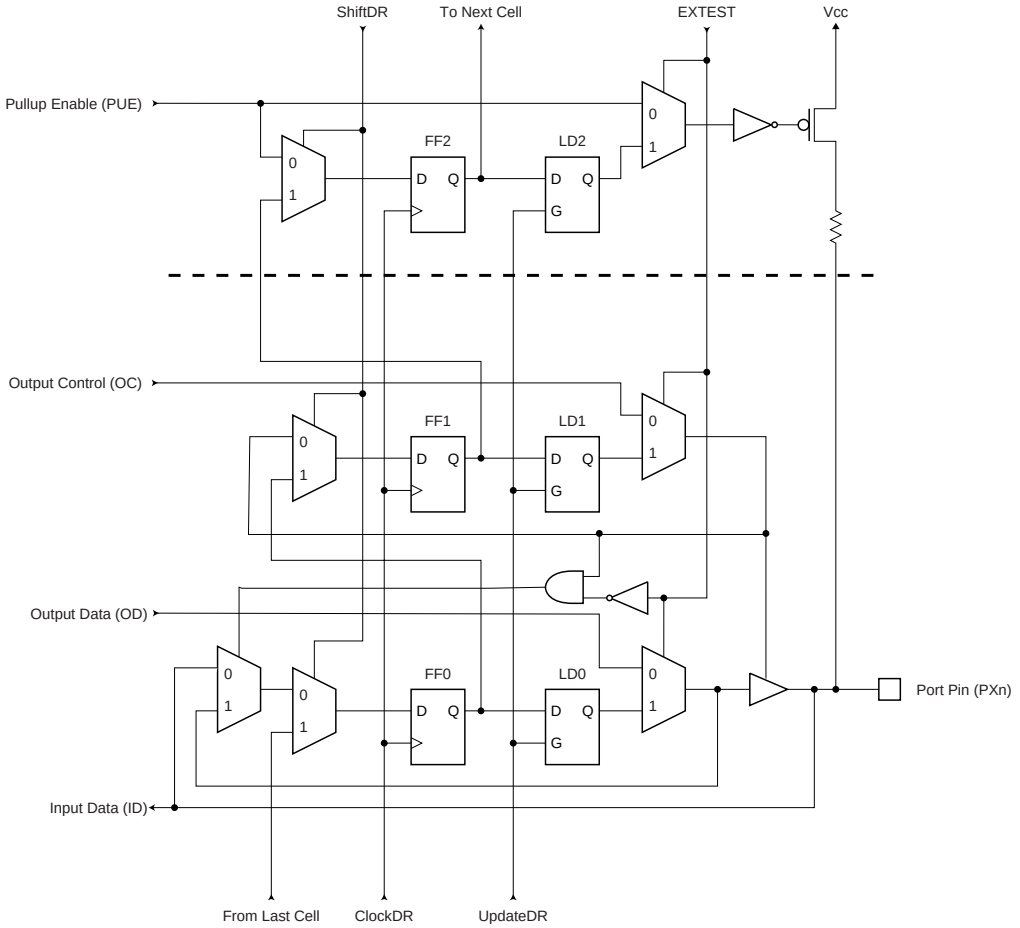
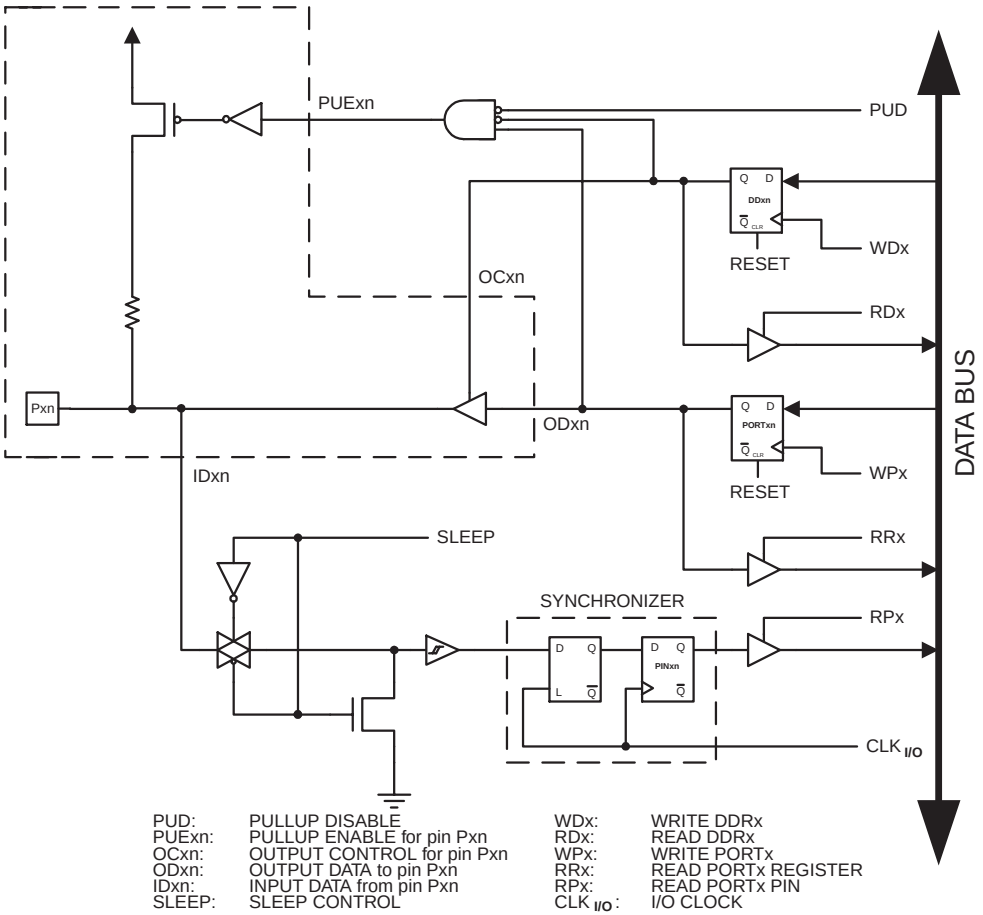


Figure 117. 通用端口引脚原理图 (1)



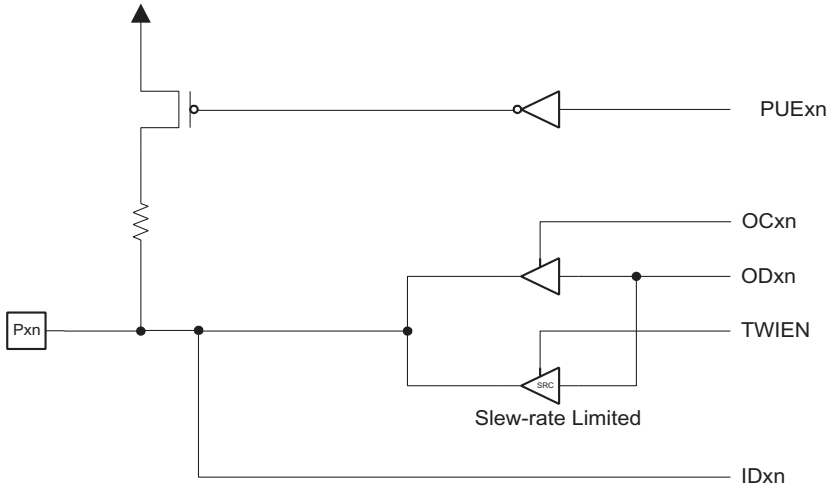
Note: 1. 详见边界扫描的说明。

边界扫描和 TWI

两线接口引脚 SCL 和 SDA 在扫描链里有一个额外的控制信号 : 两线接口使能 – TWIEN。如 Figure 118 所示，TWIEN 信号可以使能与普通数字端口并行的、具有斜率控制功能的三态缓冲器，并与 Figure 122 给出的普通扫描链单元相连。

Notes: 1. 芯片没有为输入引脚的 50ns 尖峰滤波器提供扫描链。数字端口的扫描支持已经足够用于连接性测试了。在扫描链路里加入 TWIEN 信号的唯一原因是在进行边界扫描时可以关闭斜率控制缓冲器。
2. 不要同时使能 OC 和 TWIEN，否则会引起驱动冲突。

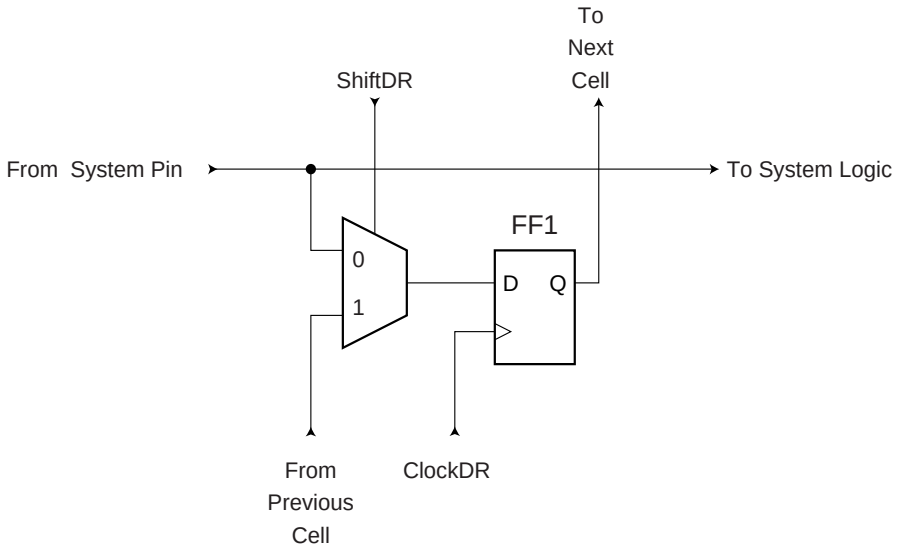
Figure 118. 为两线接口添加的额外扫描信号



扫描 RESET 引脚

RESET引脚接受5V电平的低有效逻辑电平以实现标准的复位操作；以及12V的高有效电平以实现高电压并行编程。Figure 119 所示的只能观测的扫描单元既适用于 5V 复位信号 RSTT，也适用于 12V 的复位信号 RSTHV。

Figure 119. 只能观测的单元



扫描时钟引脚

AVR器件可以通过熔丝位选择多种时钟选项，如片内RC振荡器、外部RC振荡器、外部时钟、（高频）晶体振荡器、低频晶体振荡器，以及陶瓷振荡器。

Figure 120 说明扫描链是如何支持每一种振荡器的。使能信号为普通的边界扫描单元，而振荡器 / 时钟的输出则连接到只可观测的单元。实时时钟振荡器的扫描方式与主时钟的一样。片内 RC 振荡器的输出是不能扫描的，因为这个振荡器没有外部连接。

Figure 120. 振荡器和时钟的边界扫描单元

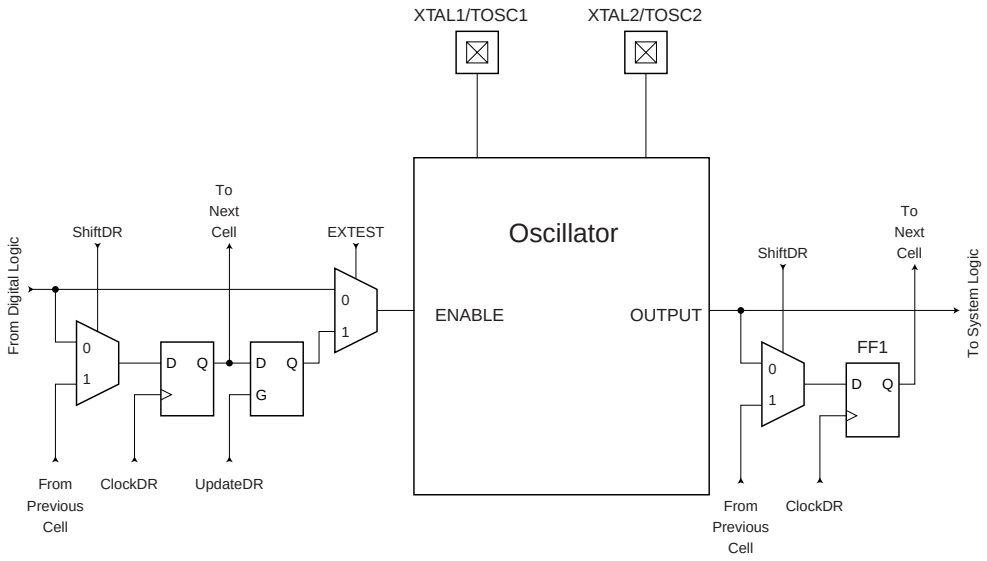


Table 90 简单总结了外部时钟引脚 XTAL1、与 XTAL1/XTAL2 连接的振荡器，以及 32 kHz 时钟振荡器的扫描寄存器。

Table 90. 振荡器的扫描信号 (1)(2)(3)

使能信号	扫描的时钟	时钟选项	未使用时扫描的时钟
EXTCLKEN	EXTCLK (XTAL1)	外部时钟	0
OSCON	OSCCK	外部晶体 外部陶瓷振荡器	0
RCOSCEN	RCCK	外部 RC	1
OSC32EN	OSC32CK	外部低频晶体振荡器	0
TOSKON	TOSCK	32 kHz 时钟振荡器	0

Notes:
1. 不要同时使能多于一个的时钟源作为主时钟
2. 扫描振荡器输出可能得到不可预期的结果。这是因为振荡器和 JTAG TCK 时钟之间有频率漂移。扫描外部时钟更可行。
3. 时钟配置通过熔丝位进行。由于运行时不能改变熔丝位，因此对于一个应用可以认为时钟配置是固定的。建议用户在扫描时钟时使用与最终系统一样的选项。由于在睡眠模式下系统有可能禁止时钟，扫描链也支持时钟使能信号，从而将振荡器引脚从扫描链中断开。扫描链不支持熔丝位 INTCAP。所以边界扫描链无法使要求内部电容的 XTAL 振荡器运行，除非此熔丝位已经得到正确编程。

扫描模拟比较器

与边界扫描相关的比较器信号示于 Figure 121。 Figure 122 给出的边界扫描单元与这些信号相连接。信号在 Table 91 中说明。

对于纯粹的连接性测试可以避开比较器，因为所有的模拟输入与数字输入共享端口引脚。

Figure 121. 模拟比较器

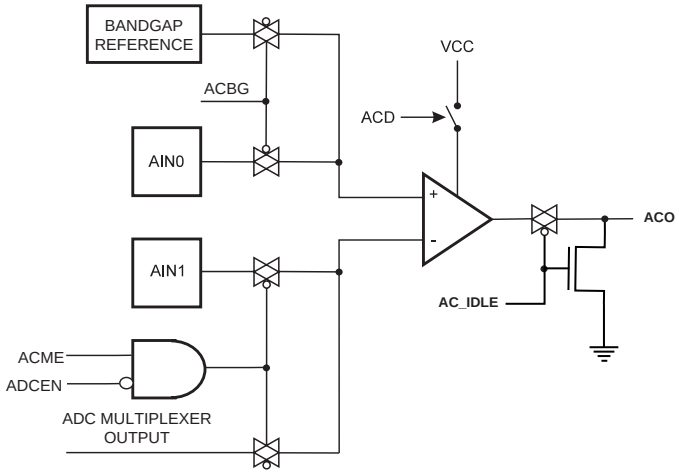


Figure 122. 适用于比较器和 ADC 的通用边界扫描单元

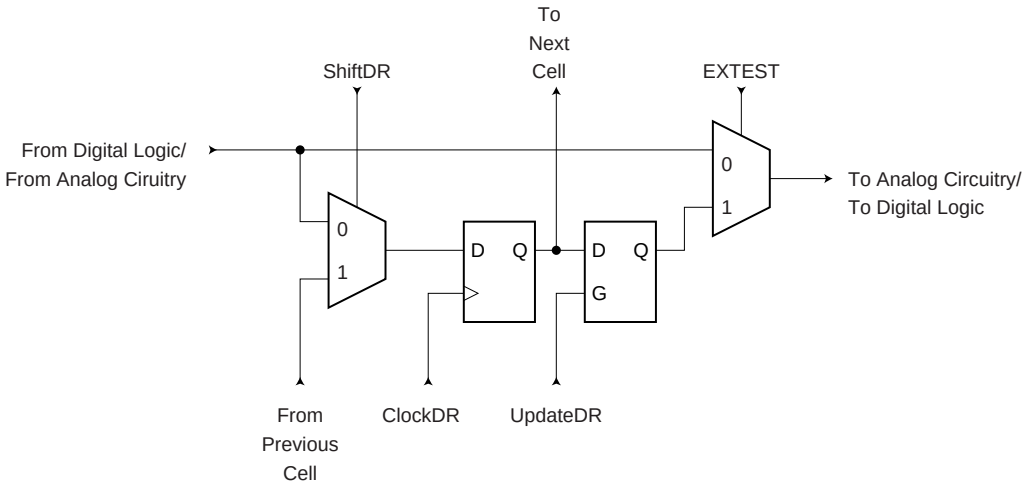


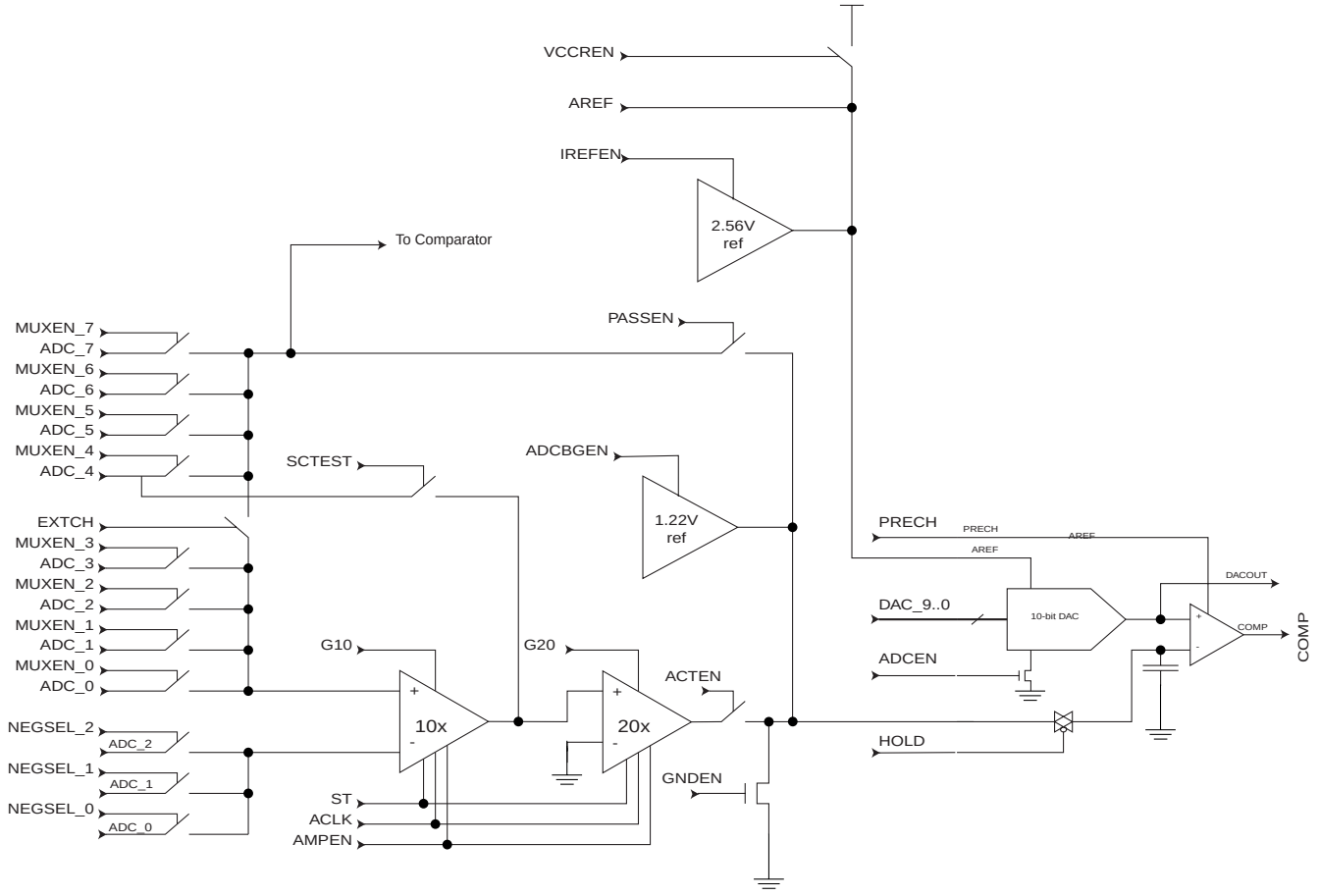
Table 91. 模拟比较器的边界扫描单元

信号名称	从比较器一侧看的方向	说明	不使用时推荐的输入	使用推荐的输入时的输出数值
AC_IDLE	输入	为 '1' 时关闭模拟比较器	1	依赖于正在执行的 µC 代码
ACO	输出	模拟比较器的输出	将成为正在执行的 µC 代码的输入	0
ACME	输入	为 '1' 时使用来自 ADC 多路器的输出	0	依赖于正在执行的 µC 代码
ACBG	输入	使能能隙基准源	0	依赖于正在执行的 µC 代码

扫描 ADC

Figure 123 给出了具有所有相关的控制和观测信号的 ADC 原理图。Figure 122 所示的边界扫描单元与这些信号相连。对于纯粹的连接性测试可以避开 ADC，因为所有的模拟输入与数字输入共享端口引脚。

Figure 123. 模数转换器



信号在 Table 92 有简短描述。

Table 92. ADC 的边界扫描信号

信号名称	从 ADC 一侧看的方向	说明	不使用时推荐的输入	当推荐的输入正在使用，且 CPU 没有使用 ADC 时的输出数值
COMP	输出	比较器输出	0	0
ACLK	输入	以开关电容滤波器方式实现的增益的时钟信号	0	0
ACTEN	输入	使能从增益到比较器的通路	0	0
ADCBGEN	输入	使能能隙基准源连接到比较器的负输入端	0	0
ADCEN	输入	ADC 的上电信号	0	0
AMPEN	输入	增益的的上电信号	0	0
DAC_9	输入	DAC 数字数值的第 9 位	1	1
DAC_8	输入	DAC 数字数值的第 8 位	0	0
DAC_7	输入	DAC 数字数值的第 7 位	0	0
DAC_6	输入	DAC 数字数值的第 6 位	0	0
DAC_5	输入	DAC 数字数值的第 5 位	0	0
DAC_4	输入	DAC 数字数值的第 4 位	0	0
DAC_3	输入	DAC 数字数值的第 3 位	0	0
DAC_2	输入	DAC 数字数值的第 2 位	0	0
DAC_1	输入	DAC 数字数值的第 1 位	0	0
DAC_0	输入	DAC 数字数值的第 0 位	0	0
EXTCH	输入	将 ADC 的 0 - 3 通道连接到旁路通道	1	1
G10	输入	使能 10x 增益	0	0
G20	输入	使能 20x 增益	0	0
GNDEN	输入	为 '1' 时将比较器的负输入端连接到地	0	0
HOLD	输入	采样 & 保持信号。为 '0' 时采样模拟信号，为 '1' 时保持信号。如果使能增益，在 ACLK 为高时这个信号必须有效	1	1
IREFEN	输入	使能能隙基准源作为 DAC 的 AREF 信号	0	0
MUXEN_7	输入	输入多路器的位 7	0	0
MUXEN_6	输入	输入多路器的位 6	0	0
MUXEN_5	输入	输入多路器的位 5	0	0
MUXEN_4	输入	输入多路器的位 4	0	0
MUXEN_3	输入	输入多路器的位 3	0	0
MUXEN_2	输入	输入多路器的位 2	0	0
MUXEN_1	输入	输入多路器的位 1	0	0

Table 92. ADC 的边界扫描信号 (Continued)

信号名称	从 ADC 一侧看的方向	说明	不使用时推荐的输入	当推荐的输入正在使用，且 CPU 没有使用 ADC 时的输出数值
MUXEN_0	输入	输入多路器的位 0	1	1
NEGSEL_2	输入	差分信号负输入端的输入多路器的位 2	0	0
NEGSEL_1	输入	差分信号负输入端的输入多路器的位 1	0	0
NEGSEL_0	输入	差分信号负输入端的输入多路器的位 0	0	0
PASSEN	输入	使能增益的传输门	1	1
PRECH	输入	比较器输出锁存器预充电 (低有效)	1	1
SCTEST	输入	开关电容 TEST 使能。 10x 增益输出发送到 ADC_4 引脚	0	0
ST	输入	如果此信号在 AMPEN 变高之后的头两个 ACLK 周期里为高，增益的输出将以更快的速度稳定下来	0	0
VCCREN	输入	选择 Vcc 为 ADC 的基准源	0	0

Note: 若不正确地设置 Figure 123 中的选项将造成信号冲突 , 并有可能毁坏器件。Figure 123 中输出比较器负输入端的 S&H 电路的输出有多种选择。要保证在 ADC 引脚、能隙基准源和地三者之一选择一个。

若在扫描过程中没有使用 ADC，则推荐使用 Table 92 给出的输入值。不推荐用户在扫描过程中使用差分增益。基于开关电容的增益需要快速的操作和精确的定时，这在使用扫描链时是很难保证的。详细的关于差分增益级操作没有给出。

AVR ADC 的模拟电路示于 Figure 123，是以数字逻辑实现的逐次逼近算法。使用边界扫描的问题是保证施加的模拟电压在某些限制下得以测量。这可以方便地通过不运行逐次逼近算法来实现：在数字 DAC[9:0] 上施加下限值，确保比较器的输出为低；然后在数字 DAC[9:0] 上施加上限值，确保比较器的输出为高。

对于纯粹的连接性测试可以避开 ADC，因为所有的模拟输入与数字输入共享端口引脚。

使用 ADC 时请记住

- 使用 ADC 时必须将引脚配置为输入，而且要禁止上拉电阻，以防止信号冲突。
- 在正常模式下 , 使能ADC时将启动一次"哑"模数转换(包括10次比较)。建议用户在使能 ADC 后等待至少 200ns，然后再开始操作；或者是执行一次 " 哑 " 模数转换。
- HOLD 信号为低时 (采样模式)， DAC 数值必须稳定于中间值 0x200。

作为例子，考虑电源电压为 5.0V、 AREF 连接到 V_{CC} 时在 ADC 通道 3 施加 1.5V ± 5% 的输入信号 。

$$\text{The lower limit is:} \quad \lceil 1024 \cdot 1{,}5V \cdot 0{,}95 / 5V \rceil = 291 = 0x123$$
$$\text{The upper limit is:} \quad \lceil 1024 \cdot 1{,}5V \cdot 1{,}05 / 5V \rceil = 323 = 0x143$$

除非使用 Table 93 所示算法的数值 , 建议使用 Table 92 的推荐数据。Table 93 只给出了扫描链中 DAC 和端口引脚的数值。" 动作 " 一列说明了在填充后面几列边界扫描寄存器之前要使用的 JTAG 指令。在移入数据的时候要对移出的数据进行校验。

Table 93. 使用 ADC 的算法

步骤	动作	ADCEN	DAC	MUXEN	HOLD	PRECH	PA3. 数据	PA3. 控制	PA3. 上拉使能
1	SAMPLE_PRELOAD	1	0x200	0x08	1	1	0	0	0
2	EXTEST	1	0x200	0x08	0	1	0	0	0
3		1	0x200	0x08	1	1	0	0	0
4		1	0x123	0x08	1	1	0	0	0
5		1	0x123	0x08	1	0	0	0	0
6	校验 COMP 是否为 0	1	0x200	0x08	1	1	0	0	0
7		1	0x200	0x08	0	1	0	0	0
8		1	0x200	0x08	1	1	0	0	0
9		1	0x143	0x08	1	1	0	0	0
10		1	0x143	0x08	1	0	0	0	0
11	校验 COMP 是否为 1	1	0x200	0x08	1	1	0	0	0

使用这个算法时对 HOLD 信号的时序约束将对 TCK 时钟产生约束。由于算法在 5 个步骤里将 HOLD 保持为高， TCK 时钟必须至少 5 倍于扫描位数除以最大保持时间 $t_{hold,max}$。

ATmega32 边界扫描次序

选择边界扫描链为数据通路时 TDI 和 TDO 之间的扫描次序如 Table 94 所示。 Bit 0 为 LSB ,是第一个被扫描 (输出 / 输入) 的位。按照引脚次序进行扫描是不可能的。因此 ,端口 A 的扫描次序与其他端口是相反的。模拟电路的扫描链是一个例外 , 它构成了扫描链的最高位 , 而不管连接到哪一个物理引脚。在 Figure 116 里 , PXn.Data 相应于 FF0 ; PXn.Control 相应于 FF1 ; PXn. Pullup_enable 相应于 FF2。端口 C 的 2、3、4、5 位不在扫描链之中 , 因为 JTAG 使能时这些引脚是 TAP 引脚。

Table 94. ATmega32 边界扫描次序

各个位的序号	信号名称	模块
140	AC_IDLE	比较器
139	ACO	
138	ACME	
137	ACBG	
136	COMP	ADC
135	PRIVATE_SIGNAL1(1)	
134	ACLK	
133	ACTEN	
132	PRIVATE_SIGNAL2(2)	
131	ADCBGEN	
130	ADCEN	
129	AMPEN	
128	DAC_9	
127	DAC_8	
126	DAC_7	
125	DAC_6	
124	DAC_5	
123	DAC_4	
122	DAC_3	
121	DAC_2	
120	DAC_1	
119	DAC_0	
118	EXTCH	
117	G10	
116	G20	
115	GNDEN	
114	HOLD	
113	IREFEN	

Table 94. ATmega32 边界扫描次序 (Continued)

各个位的序号	信号名称	模块
112	MUXEN_7	ADC
111	MUXEN_6	
110	MUXEN_5	
109	MUXEN_4	
108	MUXEN_3	
107	MUXEN_2	
106	MUXEN_1	
105	MUXEN_0	
104	NEGSEL_2	
103	NEGSEL_1	
102	NEGSEL_0	
101	PASSEN	
100	PRECH	
99	SCTEST	
98	ST	
97	VCCREN	
96	PB0.Data	端口 B
95	PB0.Control	
94	PB0.Pullup_Enable	
93	PB1.Data	
92	PB1.Control	
91	PB1.Pullup_Enable	
90	PB2.Data	
89	PB2.Control	
88	PB2.Pullup_Enable	
87	PB3.Data	
86	PB3.Control	
85	PB3.Pullup_Enable	
84	PB4.Data	
83	PB4.Control	
82	PB4.Pullup_Enable	

Table 94. ATmega32 边界扫描次序 (Continued)

各个位的序号	信号名称	模块
81	PB5.Data	端口 B
80	PB5.Control	
79	PB5.Pullup_Enable	
78	PB6.Data	
77	PB6.Control	
76	PB6.Pullup_Enable	
75	PB7.Data	
74	PB7.Control	
73	PB7.Pullup_Enable	
72	RSTT	复位逻辑 (只可观测)
71	RSTHV	
70	EXTCLKEN	使能主时钟 / 振荡器信号
69	OSCON	
68	RCOSCEN	
67	OSC32EN	
66	EXTCLK (XTAL1)	主时钟的时钟输入与振荡器 (只可观测)
65	OSCCK	
64	RCCK	
63	OSC32CK	
62	TWIEN	TWI
61	PD0.Data	端口 D
60	PD0.Control	
59	PD0.Pullup_Enable	
58	PD1.Data	
57	PD1.Control	
56	PD1.Pullup_Enable	
55	PD2.Data	
54	PD2.Control	
53	PD2.Pullup_Enable	
52	PD3.Data	
51	PD3.Control	
50	PD3.Pullup_Enable	
49	PD4.Data	
48	PD4.Control	
47	PD4.Pullup_Enable	

Table 94. ATmega32 边界扫描次序 (Continued)

各个位的序号	信号名称	模块
46	PD5.Data	端口 D
45	PD5.Control	
44	PD5.Pullup_Enable	
43	PD6.Data	
42	PD6.Control	
41	PD6.Pullup_Enable	
40	PD7.Data	
39	PD7.Control	
38	PD7.Pullup_Enable	
37	PC0.Data	端口 C
36	PC0.Control	
35	PC0.Pullup_Enable	
34	PC1.Data	
33	PC1.Control	
32	PC1.Pullup_Enable	
31	PC6.Data	
30	PC6.Control	
29	PC6.Pullup_Enable	
28	PC7.Data	
27	PC7.Control	
26	PC7.Pullup_Enable	
25	TOSC	32 kHz 定时振荡器
24	TOSCON	
23	PA7.Data	端口 A
22	PA7.Control	
21	PA7.Pullup_Enable	
20	PA6.Data	
19	PA6.Control	
18	PA6.Pullup_Enable	
17	PA5.Data	
16	PA5.Control	
15	PA5.Pullup_Enable	
14	PA4.Data	
13	PA4.Control	
12	PA4.Pullup_Enable	

Table 94. ATmega32 边界扫描次序 (Continued)

各个位的序号	信号名称	模块
11	PA3.Data	端口 A
10	PA3.Control	
9	PA3.Pullup_Enable	
8	PA2.Data	
7	PA2.Control	
6	PA2.Pullup_Enable	
5	PA1.Data	
4	PA1.Control	
3	PA1.Pullup_Enable	
2	PA0.Data	
1	PA0.Control	
0	PA0.Pullup_Enable	

Notes: 1. PRIVATE_SIGNAL1 总是扫描为 0
2. PRIVATE_SIGNAL2 总是扫描为 0

边界扫描描述语言文件

边界扫描描述语言 (BSDL) 文件以一种为自动化测试生成软件使用的标准格式描述具有边界扫描功能的器件。这种文件的使用对象为自动化的测试生成软件。边界扫描数据寄存器各个位的次序和功能都包含于此文件中。 BSDL 文件在 ATmega32 中有效。

支持引导装入程序 - 在写的同时可以读 (RWW, Read-While-Write) 的自我编程能力

Boot Loader为通过MCU本身来下载和上载程序代码提供了一个真正的同时读-写(Read-While-Write，以下简称 RWW) 自编程机制。这一特点使得系统可以在 MCU 的控制下，通过驻留于程序 Flash 的 Boot Loader，灵活地进行应用软件升级。 Boot Loader 可以使用任何器件具有的数据接口和相关的协议获得代码并把代码（程序）写入 Flash，或者从程序存储器读取代码。Boot Loader 区的程序可以写整个 Flash，包括 Boot Loader 区本身。因而 Boot Loader 可以对其自身进行修改，甚至将自己擦除。Boot Loader 存储器空间的大小可以通过熔丝位进行配置。 Boot Loader 具有两套程序加密位，各自可以独立设置，给用户提供了选择保护级的灵活性。

特点

- **RWW 自编程**
- **灵活的 Boot Loader 存储区配置**
- **高度的安全性 (有单独的 Boot 锁定位实现灵活的程序保护)**
- **有独立的熔丝位用于选择复位向量**
- **优化的页 (1) 大小**
- **代码优化的算法**
- **有效的 RWW 支持**

Note: 1. 页是 Flash 的一个部分，由数个字节组成 (见 P243 Table 111)，在编程过程中使用。页的组织结构不影响正常的操作

应用程序 Flash 区以及引导程序 Flash 区

Flash 由两个区构成，应用区和 Boot Loader 区 (见 Figure 125)。两个区的存储空间大小由 BOOTSZ 熔丝位配置，如 P237 Table 100 与 Figure 125 所示。由于两个区使用不同的锁定位，所以可以具有不同的加密级别。

应用程序区

应用区是 Flash 用来存储应用代码的区域。应用区的保护级别通过应用 Boot 锁定位 (Boot 锁定位 0) 确定，详见 P230 Table 96 。由于 SPM 指令在应用区执行时是无效的，所以应用区不能用来存储 Boot Loader 代码。

引导程序区 (Boot Loader Section) - BLS

应用区用来存储应用代码，而 Boot Loader 软件必须保存在 BLS。这是因为只有在 BLS 运行时 SPM 指令才有效。SPM 指令可以访问整个 Flash，包括 BLS 本身。Boot Loader 区的保护级别通过 Boot Loader 锁定位 (Boot 锁定位 1) 确定，详见 P230 Table 97 。

RWW Flash 区及非 RWW Flash 区

CPU 是否支持 RWW，或者 CPU 是否在利用 Boot Loader 软件进行代码更新时停止，取决于被编程的是哪个地址。除了前面所述的通过 BOOTSZ 熔丝位配置的两个区之外，Flash 还可以分成两个固定的区——同时读 - 写 (RWW) 区和非同时读 - 写 (NRWW) 区。RWW- 和 NRWW 的分界在 P237 Table 101 和 P229 Figure 125 给出。两个区的主要区别是：

- 对 RWW 区内的页进行擦除或写操作时可以读 NRWW 区

对 NRWW 区内的页进行擦除或写操作时， CPU 停止

注意，Boot Loader 软件工作时，用户软件不能读取位于 RWW 区内的任何代码。"RWW 区 " 指的是被编程 (擦除或写) 的那个存储区，而不是利用 Boot Loader 软件进行代码更新过程中实际被读取的那部分。

RWW 区

如果 Boot Loader 软件是对 RWW 区内的某一页进行编程，则可以从 Flash 中读取代码，但只限于 NRWW 区内的代码。在 Flash 编程期间，用户软件必须保证没有对 RWW 区的读访问。如果用户软件在编程过程中试图读取位于 RWW 区的代码 (如通过 call/jmp/lpm 指令或中断)，软件可能会终止于一个未知状态。为了避免这种情况的发生，需要禁止中断或将其转移到 Boot Loader 区。 Boot Loader 总是位于 NRWW 存储区。只要 RWW 区处于不能读访问的状态，存储程序存储器控制和状态寄存器 (SPMCSR) 的 RWW 区忙标志位 RWWSB 置位。编程结束后，要在读取 位于 RWW 区的代码之前通过软件清除 RWWSB。具体如何清除RWWSB请参见 P232 "保存程序存储器控制寄存器–SPMCR" 。

非 RWW 区 - NRWW

在 Boot Loader 软件更新 RWW 区的某一页时，可以读取位于 NRWW 区的代码。当 Boot Loader 代码更新 NRWW 区时，在整个页擦除或写操作过程中 CPU 被挂起。

Table 95. RWW 的特点

编程过程中 Z 指针寻址哪个区？	编程过程中可以读取哪个区？	CPU 挂起吗？	支持 RWW 吗？
RWW 区	NRWW 区	不	是
NRWW 区	无	是	不

Figure 124. RWW 与 NRWW

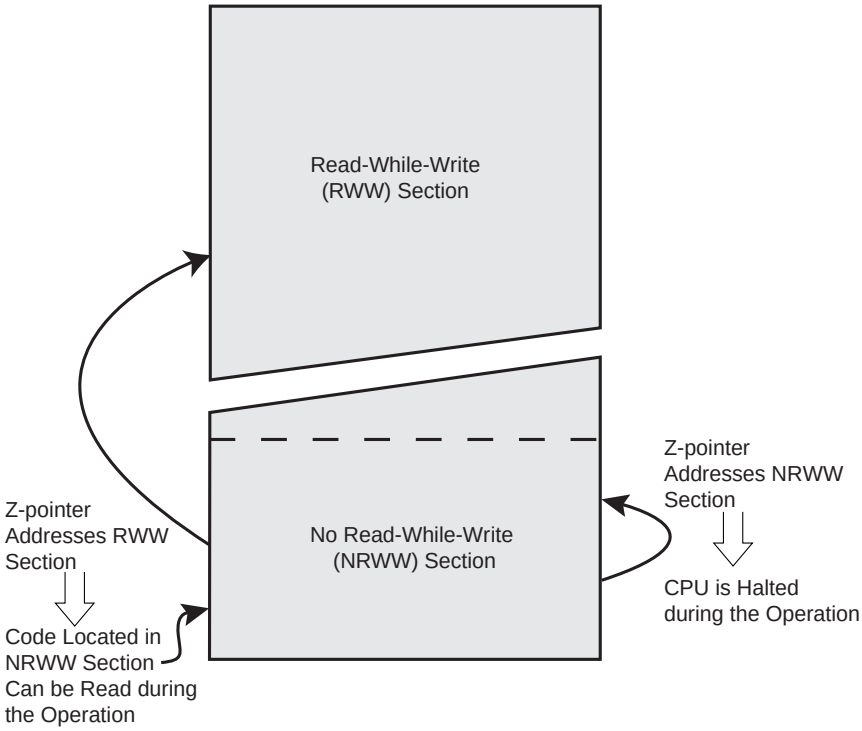
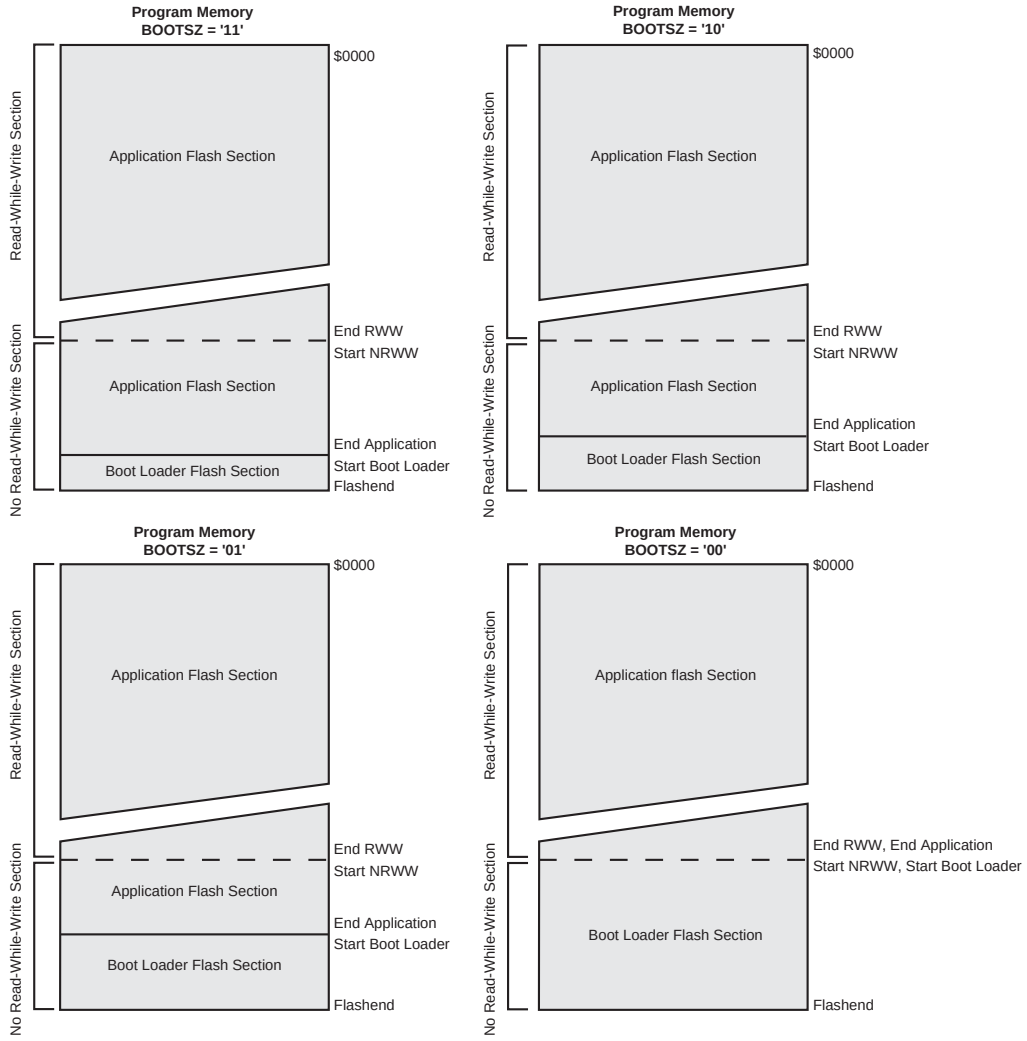


Figure 125. 存储器区 (1)



Note: 1. 上图中的参数在 P237 Table 100 中给出。

引导程序区锁定位

如果不需要 Boot Loader 功能，则整个 Flash 都可以为应用代码所用。Boot Loader 具有两套可以独立设置的 Boot 锁定位。用户可以灵活地选择不同的代码保护方式。

用户可以选择：

- 保护整个 Flash 区，不让 MCU 进行软件升级
- 不允许 MCU 升级 Boot Loader Flash 区
- 不允许 MCU 升级应用 Flash 区
- 允许 MCU 升级整个 Flash 区

详细内容请参见 Table 96 与 Table 97。Boot 锁定位可以通过软件、串行或并行编程进行设置，但只能通过芯片擦除命令清除。通用的写锁定位 (锁定位模式 2) 不限制通过 SPM 指令对 Flash 进行编程。与此类似，通用的读 / 写锁定位 (锁定位模式 1) 也不限制通过 LPM/SPM 指令对 Flash 进行读 / 写访问。

Table 96. Boot 锁定位 0 保护模式 (应用区)[1]

BLB0 Mode	BLB02	BLB01	保护
1	1	1	允许 SPM/LPM 指令访问应用区
2	1	0	不允许 SPM 指令对应用区进行写操作
3	0	0	不允许 SPM 指令对应用区进行写操作，也不允许运行于 Boot Loader 区的 LPM 指令从应用区读取数据。若中断向量位于 Boot Loader 区，那么执行应用区代码时中断是禁止的。
4	0	1	不允许运行于 Boot Loader 区的 LPM 指令从应用区读取数据。若中断向量位于 Boot Loader 区，那么执行应用区代码时中断是禁止的。

Note: 1. “1” 表示未被编程，“0” 表示已编程。

Table 97. Boot 锁定位 1 保护模式 (Boot Loader 区)[1]

BLB1 mode	BLB12	BLB11	保护
1	1	1	允许 SPM/LPM 指令访问 Boot Loader 区
2	1	0	不允许 SPM 指令对 Boot Loader 区进行写操作
3	0	0	不允许 SPM 指令对 Boot Loader 区进行写操作，也不允许运行于应用区的 LPM 指令从 Boot Loader 区读取数据。若中断向量位于应用区，那么执行 Boot Loader 区代码时中断是禁止的。
4	0	1	不允许运行于应用区的 LPM 指令从 Boot Loader 区读取数据。若中断向量位于应用区，那么执行 Boot Loader 区代码时中断是禁止的。

Note: 1. “1” 表示未被编程，“0” 表示已编程。

进入引导程序

通过跳转指令或从应用区调用的方式可以进入 Boot Loader。这些操作可以由一些触发信号启动，比如通过 USART 或 SPI 接口接收到了相关的命令。另外，可以通过编程 Boot 复位熔丝位使得复位向量指向 Boot 区的起始地址。这样，复位后 Boot Loader 立即就启动了。加载了应用代码后，程序开始执行应用代码。 MCU 本身不能改变熔丝位的设置。也就是说，一旦 Boot 复位熔丝位被编程，复位向量将一直指向 Boot 区的起始地址。熔丝位只能通过串行或并行编程的方法来改变。

Table 98. Boot 复位熔丝位 (1)

BOOTRST	复位地址
1	复位向量 = 应用区复位 (地址 0x0000)
0	复位向量 =Boot Loader 复位 (见 P237 Table 100)

Note: 1. “1” 表示未编程 , “0” 表示已编程。

保存程序存储器控制寄存器 - SPMCR

SPMCR 包含控制 Boot Loader 操作的位。

Bit	7	6	5	4	3	2	1	0	
	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	SPMCR
读 / 写	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – SPMIE: SPM 中断使能**

SPMIE 置位后 , 如果状态寄存器的 I 位也置位 , SPM 中断即被使能。只要 SPMCSR 寄存器的 SPMEN 清零， SPM 中断将被执行。

- **Bit 6 – RWWSB: RWW 区忙标志**

启动对 RWW 区的自编程 (页擦除或页写入) 操作时， RWWSB 被硬件置 1。 RWWSB 置位时不能访问 RWW 区。自编程操作完成后，如果 RWWSRE 位为 1， RWWSB 位将被清除。另外，启动页加载操作将使 RWWSB 位自动清零。

- **Bit 5 – Res: 保留位**

在 ATmega32 中为保留位，读返回值为 "0”。

- **Bit 4 – RWWSRE: RWW 区读使能**

RWW区处于编程(页擦除或页写入)状态时 ,RWW区的读操作(RWWSB被硬件置"1”)将被阻塞。用户软件必须等到编程结束(SPMEN清零)才能重新使能RWW区。如果 RWWSRE 位和SPMEN同时被写入"1” ,则在紧接着的四个时钟周期内的SPM指令将再次使能RWW区。如果 Flash 忙于页擦除或页写入 (SPMEN 置位),RWW 区不能被使能。如果 Flash 加载与 RWWSRE 写操作同时发生，则 Flash 加载操作终止，加载的数据亦将丢失。

- **Bit 3 – BLBSET: Boot 锁定位设置**

如果这一位和 SPMEN 同时置位 , 发生于紧接着的四个时钟周期内的 SPM 指令会根据 R0 中的数据设置 Boot 锁定位。R1 中的数据和 Z 指针的地址信息被忽略。锁定位设置完成，或在四个时钟周期内没有 SPM 指令被执行时， BLBSET 自动清零。

在 SPMCSR 寄存器的 BLBSET 和 SPMEN 置位后的三个周期内运行的 LPM 指令将读取锁定位或熔丝位 (取决于 Z 指针的 Z0) 并送到目的寄存器。详见 P236 “ 以软件方式读取熔丝位和锁定位 ” 。

- **Bit 2 – PGWRT: 页写入**

如果这一位和 SPMEN 同时置位，发生于紧接着的四个时钟周期内的 SPM 指令执行页写功能 , 将临时缓冲器中存储的数据写入 Flash。页地址取自 Z 指针的高位部分。R1 和 R0 的数据则被忽略。页写操作完成 , 或在四个时钟周期内没有 SPM 指令被执行时 , PGWRT 自动清零。如果页写对象为 NRWW 区，在整个页写操作过程中 CPU 停止。

- **Bit 1 – PGERS: 页擦除**

如果这一位和 SPMEN 同时置位，发生于紧接着的四个时钟周期内的 SPM 指令执行页擦除功能。页地址取自 Z 指针的高位部分。 R1 和 R0 的数据则被忽略。页擦除操作完成，或在四个时钟周期内没有SPM指令被执行时，PGERS自动清零。如果页写对象为NRWW区，在整个页擦除操作过程中 CPU 停止。

- **Bit 0 – SPMEN: 存贮程序存储器使能**

这一位在紧接着的四个时钟周期内使能 SPM 指令。如果将这一位和 RWWSRE、BLBSET、PGWRT 或 PGERS 之一同时置位，则如上所述，接下来的 SPM 指令将有特殊的含义。如果只有 SPMEN 置位，那么接下来的 SPM 指令将把 R1:R0 中的数据存储到由 Z 指针确定的临时页缓冲器。Z 指针的 LSB 被忽略。SPM 指令完成，或在四个时钟周期内没有 SPM 指令被执行时， SPMEN 自动清零。在页擦除和页写过程中 SPMEN 保持为 1 直到操作完成。

在低五位中写入除 “10001”、“01001”、“00101”、 “00011” 或 “00001” 之外的任何组合都无效。

在自编程时访问 Flash

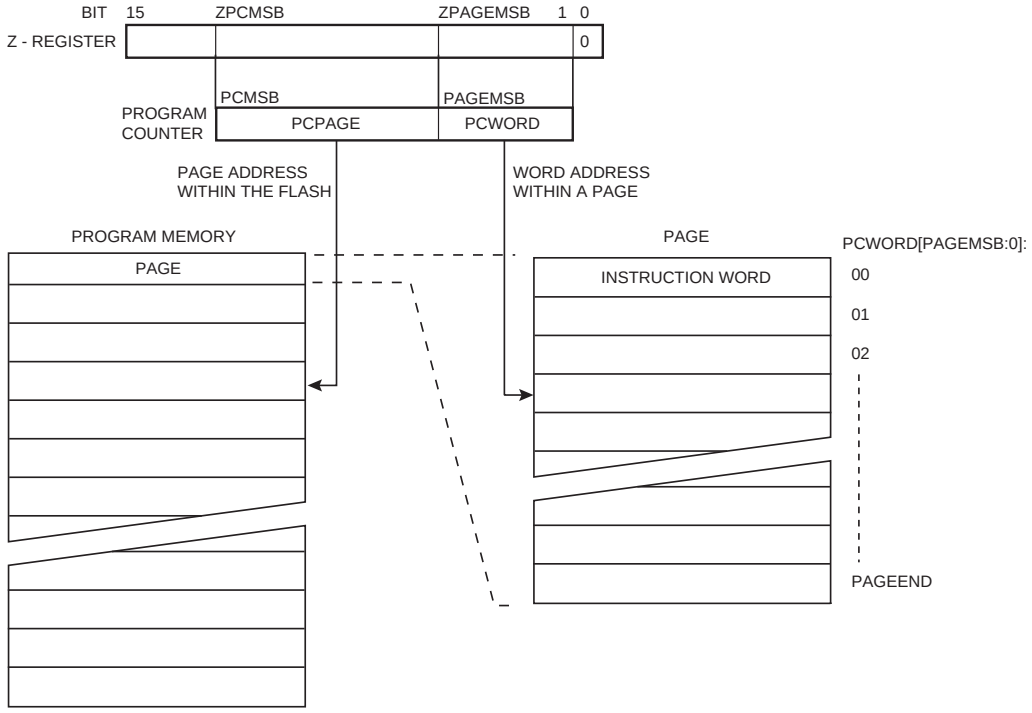
Z 指针用于 SPM 命令的寻址。

Bit	15	14	13	12	11	10	9	8
ZH (R31)	**Z15**	**Z14**	**Z13**	**Z12**	**Z11**	**Z10**	**Z9**	**Z8**
ZL (R30)	**Z7**	**Z6**	**Z5**	**Z4**	**Z3**	**Z2**	**Z1**	**Z0**
	7	6	5	4	3	2	1	0

由于 Flash 存储器是以页的形式组织 (see P243 Table 111) 起来的，程序计数器可看作由两个部分构成：其一为实现页内寻址的低位部分；其次为实现页寻址的高位部分，如 Figure 126 所示。由于页擦除和页写操作的寻址是相互独立的，因此保证 Boot Loader 软件在页擦除和页写操作时寻址相同的页是最重要的。一旦编程操作开始启动，地址就被锁存，然后 Z 指针可以用作其他用途了。

唯一不使用 Z 指针的 SPM 操作是设置 Boot Loader 锁定位。Z 指针的内容被忽略。LPM 指令也使用 Z 指针来保存地址。由于这个指令的寻址逐字节地进行，所以 Z 指针的 LSB 位 (位 Z0) 也使用到了。

Figure 126. SPM 的寻址 (1)



Note: 1. Figure 126 中所用的不同的变量在 P238 Table 102 中列出。

Flash 的自编程

程序存储器的更新以页的方式进行。在用临时页缓冲器存储的数据对一页存储器进行编程时，首先要将这一页擦除。 SPM 指令以一次一个字的方式将数据写入临时页缓冲器。临时页缓冲器的写入可以在页擦除命令之前完成，也可以在页擦除和页写操作之间完成。

方案 1，在页擦除前写缓冲器

- 写临时页缓冲器
- 执行页擦除操作
- 执行页写操作

方案 2，在页擦除后写缓冲器

- 执行页擦除操作
- 写临时页缓冲器
- 执行页写操作

如果只需要改变页的一部分，则在页擦除之前必须将页中其他部分存储起来 (如保存于临时页缓冲区中)，然后再写回 Flash。使用方案 1 时， Boot Loader 提供了一个有效的读 - 修改 - 写特性，允许用户软件首先读取页中的内容，然后对内容做必要的改变，接着把修改后的数据写回 Flash。如果使用方案 2，则无法读取旧数据，因为页已经被擦除了。临时页缓冲区可以随机寻址。保证在页擦除和页写操作中寻址相同的页是很关键的。汇编代码的例子请参见 P237 “ 一个简单的引导程序汇编代码 ” 。

通过 SPM 执行页擦除

执行页擦除操作首先需要设置 Z 指针与 RAMPZ 的地址信息，然后将 “X0000011” 写入 SPMCSR，最后在其后的四个时钟周期内执行 SPM。R1 和 R0 中的数据被忽略。页地址必须写入 Z 寄存器的 PCPAGE。 Z 指针的其他位被忽略。

- 擦除 RWW 区的页：在页擦除过程中可以读取 NRWW 区
- 擦除 NRWW 区的页：在操作过程中 CPU 停止

装载临时缓冲器 (页加载)

写一个指令字首先需要设置 Z 指针的地址信息，以及将指令字写入 R1:R0，然后将 “00000001” 写入 SPMCSR，最后在其后的四个时钟周期内执行 SPM。Z 寄存器中 PCWORD 的内容用来寻址临时缓冲区。页写操作完成，或置位 SPMCSR 寄存器的 RWWSRE 将使临时缓冲区自动擦除。系统复位也会擦除临时缓冲区。但是如果不清除临时缓冲区就只能对每个地址进行一次写操作。

Note: 如果 EEPROM 在 SPM 页下载中间写操作，所有下载数据丢失。

执行页写操作

执行页写操作首先需要设置 Z 指针与 RAMPZ 的地址信息，然后将 “X0000101” 写入 SPMCSR，最后在其后的四个时钟周期内执行 SPM。R1 和 R0 中的数据被忽略。页地址必须写入 Z 寄存器的 PCPAGE。Z 指针的其他位被忽略。

- 擦除 RWW 区的页：在页擦除过程中可以读取 NRWW 区
- 擦除 NRWW 区的页：在页写过程中 CPU 停止

利用 SPM 中断

如果 SPM 中断使能，则 SPMCSR 寄存器的 SPMEN 清零将产生中断。这意味着软件可以利用中断来代替对 SPMCSR 寄存器的查询。使用 SPM 中断时，要将中断向量移到 BLS，以避免 RWW 区读禁止时中断程序却访问它。如何移动中断向量请见 P42 “ 中断 ” 。

在更新 BLS 时需要考虑的问题

通过不编程 Boot 锁定位 11 的方式来更新 Boot Loader 区时需要给予格外关注。对 Boot Loader 本身进行的误操作会破坏整个 Boot Loader ，造成软件无法更新。如果程序不需要改变 Boot Loader，建议对 Boot 锁定位 11 编程，以防止不小心改变了 Boot Loader。

在自编程时防止读取 RWW 区

在自编程过程中 (页擦除或页写)，对 RWW 区的访问被阻塞。用户软件要避免此情况发生。RWW 区忙将使 SPMCSR 寄存器的 RWWSB 置位。在自编程时，如 P42 “ 中断 ” 所述，中断向量表应该移到 BLS 中，或者禁止中断。编程结束后，在寻址 RWW 区之前用户软件必须对 RWWSRE 写 1 来清零 RWWSB。例子请见 P237 “ 一个简单的引导程序汇编代码 ” 。

通过 SPM 设置引导程序锁定位

设置Boot Loader 锁定位首先要给R0赋予期望的数据，然后将“X0001001”写入SPMCSR 寄存器，并在紧接着的四个时钟周期内执行 SPM 指令。唯一可访问的锁定位是 Boot Loader 锁定位。利用这个锁定位可以阻止 MCU 对应用程序和 Boot Loader 软件的更新。

Bit	7	6	5	4	3	2	1	0
R0	1	1	BLB12	BLB11	BLB02	BLB01	1	1

不同的 Boot Loader 锁定位设置对 Flash 访问的影响请参见 Table 96 与 Table 97 。

如果 R0 的 5..2 位为 0 ,并且在 SPMCSR 寄存器的 BLBSET 和 SPMEN 置位之后的四个周期内执行了 SPM 指令，相应的 Boot 锁定位将被编程。此操作不使用 Z 指针，但出于兼容性的考虑，建议将 Z 指针赋值为 0x0001(与读 IO_{ck} 位的操作相同)。同样出于兼容性的考虑，建议在写锁定位时将 R0 中的 7、6、 1 和第 0 位置 "1”。在编程锁定位的过程中可以自由访问整个 Flash 区。

写 EEPROM 将阻止写 SPMCR

EEPROM 写操作会阻塞对 Flash 的编程 ,也会阻塞对熔丝位和锁定位的读操作。建议用户在对 SPMCR 寄存器进行写操作之前首先检查 EECR 寄存器的状态位 EEWE，确保此位以被清除。

以软件方式读取熔丝位和锁定位

熔丝位和锁定位可以通过软件读取。读锁定位时，需要将 0x0001 赋予给 Z 指针并且置位 SPMCSR寄存器的BLBSET 和 SPMEN。在SPMCSR操作之后的三个CPU周期内执行的 LPM 指令将把锁定位的值将加载到目的寄存器。读锁定位操作结束 ,或者在三个 CPU 周期内没有 执行LPM指令 ,或在四个CPU周期内没有执行SPM指令 ,BLBSET 和 SPMEN 位将自动硬件清零。 BLBSET 和 SPMEN 清零后，LPM 将按照指令手册中所描述的那样工作。

Bit	7	6	5	4	3	2	1	0
Rd	–	–	BLB12	BLB11	BLB02	BLB01	LB2	LB1

读取熔丝位低字节的算法和上述读取锁定位的算法类似。要读取熔丝位低字节，需要将 0x0000赋予给Z指针并且置位SPMCR寄存器的BLBSET 和 SPMEN。在SPMCSR操作之后的三个 CPU 周期内执行的 LPM 指令将把熔丝位低位字节的值 (FLB) 加载到目的寄存器。更详细的说明及熔丝位低位字节映射的细节请参见 P240 Table 106 。

Bit	7	6	5	4	3	2	1	0
Rd	FLB7	FLB6	FLB5	FLB4	FLB3	FLB2	FLB1	FLB0

类似的 ,读取熔丝位高位字节时 ,需要将 0x0003 赋予给 Z 指针并且置位 SPMCR 寄存器的 BLBSET 和 SPMEN。在 SPMCSR 操作之后的三个 CPU 周期内执行的 LPM 指令将把熔丝位高位字节的值 (FHB) 加载到目的寄存器。更详细的说明及熔丝位高位字节映射的细节请参见 P240 Table 105 。

Bit	7	6	5	4	3	2	1	0
Rd	FHB7	FHB6	FHB5	FHB4	FHB3	FHB2	FHB1	FHB0

被编程的熔丝位和锁定位的读返回值为 "0”。未被编程的熔丝位和锁定位的读返回值为 "1”。

防止 Flash 的内容损毁

V_{CC} 低于工作电压时 ,CPU 和 Flash 正常工作无法保证 ,Flash 的内容可能受到破坏。这个问题对于应用于板级系统的独立 Flash 一样存在。所以也要采用同样的解决方案。

电压太低时有两种情况可以破坏 Flash 内容。第一， Flash 写过程需要一个最低电压。第二，电压太低时 CPU 本身会错误地执行指令。

遵循以下设计建议可以避免 Flash 被破坏 (采用其中之一就足够了) :

1. 如果系统不需要更新 Boot Loader，建议编程 Boot Loader 锁定位以防止 Boot Loader 软件更新
2. 电源电压不足期间，保持 AVR RESET 为低：采用的方式为：如果工作电压与检测电平相匹配，可以使能 BOD 功能；否则可以使用外部复位保护电路。如果在写操作进行中发生了复位，只要电源电压足够，写操作还会完成。
3. 低电压期间保持 AVR 内核处于掉电休眠模式。这样可以防止 CPU 解码并执行指令，有效地保护 SPMCR 寄存器，从而防止 Flash 被无意识得修改掉。

使用 SPM 时的 Flash 编程时间

片内校准的 RC 振荡器用于 Flash 寻址时序控制。Table 99 给出了 CPU 访问 Flash 的典型编程时间。

Table 99. SPM 编程时间

符号	最小编程时间	最大编程时间
Flash 写操作 (通过 SPM 实现页擦除、页写、及写锁定位)	3.7 ms	4.5 ms

一个简单的引导程序汇编代码

```
    ;- 本例程将 RAM 中的一页数据写入 Flash
  ; Y 指针指向 RAM 的第一个数据单元
  ;Z 指针指向 Flash 的第一个数据单元
  ;- 本例程没有包括错误处理
  ;- 该程序必须放置于 Boot 区 ( 至少 Do_spm 子程序是如此 )
  ; 在自编程过程中 ( 页擦除和页写操作 ) 只能读访问 NRWW 区的代码
  ;- 使用的寄存器：r0、 r1、 temp1 (r16)、 temp2 (r17)、 looplo (r24)、
  ; loophi (r25)、 spmcrval (r20)
  ; 在程序中不包括寄存器内容的保护和恢复
  ; 在牺牲代码大小的情况下可以优化寄存器的使用
  ;- 假设中断向量表位于 Boot loader 区，或者中断被禁止。
.equ  PAGESIZEB = PAGESIZE*2        ; PAGESIZEB 是以字节为单位的页大小，不是以字为单位
.org SMALLBOOTSTART
Write_page:
  ; 页擦除
  ldi   spmcrval, (1<<PGERS) | (1<<SPMEN)
  call  Do_spm

  ; 重新使能 RWW 区
  ldi   spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  call  Do_spm

  ; 将数据从 RAM 转移到 Flash 页缓冲区
  ldi   looplo, low(PAGESIZEB)      ; 初始化循环变量
  ldi   loophi, high(PAGESIZEB)     ;PAGESIZEB<=256 时不需要此操作
Wrloop:
  ld    r0, Y+
  ld    r1, Y+
  ldi   spmcrval, (1<<SPMEN)
  call  Do_spm
  adiw  ZH:ZL, 2
  sbiw  loophi:looplo, 2            ;PAGESIZEB<=256 时请使用 subi
  brne  Wrloop

  ; 执行页写
  subi  ZL, low(PAGESIZEB)          ; 恢复指针
  sbci  ZH, high(PAGESIZEB)         ;PAGESIZEB<=256 时不需要此操作
  ldi   spmcrval, (1<<PGWRT) | (1<<SPMEN)
  call  Do_spm

  ; 重新使能 RWW 区
  ldi   spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  call  Do_spm

  ; 读回数据并检查，为可选操作
  ldi   looplo, low(PAGESIZEB)      ; 初始化循环变量
  ldi   loophi, high(PAGESIZEB)     ;PAGESIZEB<=256 时不需要此操作
  subi  YL, low(PAGESIZEB)          ; 恢复指针
  sbci  YH, high(PAGESIZEB)
Rdloop:
  lpm   r0, Z+
  ld    r1, Y+
  cpse  r0, r1
```

```
  jmp   Error
  sbiw  loophi:looplo, 1            ;PAGESIZEB<=256 时请使用 subi
  brne  Rdloop

  ; 返回到 RWW 区
  ; 确保 RWW 区已经可以安全读取
Return:
  in    temp1, SPMCR
  sbrs  temp1, RWWSB               ; 若 RWWSB 为 "1"，说明 RWW 区还没有准备好
  ret
  ; 重新使能 RWW 区
  ldi   spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  call  Do_spm
  rjmp  Return

Do_spm:
  ; 检查先前的 SPM 操作是否已经完成
Wait_spm:
  in    temp1, SPMCR
  sbrc  temp1, SPMEN
  rjmp  Wait_spm
  ; 输入：spmcrval 决定了 SPM 操作
  ; 禁止中断，保存状态标志
  in    temp2, SREG
  cli
  ; 确保没有 EEPROM 写操作
Wait_ee:
  sbic  EECR, EEWE
  rjmp  Wait_ee
  ; SPM 时间序列
  out   SPMCR, spmcrval
  spm
  ; 恢复 SREG （如果中断原本是使能的，则使能中断）
  out   SREG, temp2
  ret
```

ATmega32 引导程序参数

自编程描述中所用的参数在 Table 100 到 Table 102 中给出。

Table 100. Boot 区大小配置 (1)

BOOTSZ1	BOOTSZ0	Boot 区大小	页数	应用 Flash 区	Boot Loader Flash 区	应用区结束地址	Boot 复位地址 (Boot Loader 起始地址)
1	1	256 字	4	$0000 - $3EFF	$3F00 - $3FFF	$3EFF	$3F00
1	0	512 字	8	$0000 - $3DFF	$3E00 - $3FFF	$3DFF	$3E00
0	1	1024 字	16	$0000 - $3BFF	$3C00 - $3FFF	$3BFF	$3C00
0	0	2048 字	32	$0000 - $37FF	$3800 - $3FFF	$37FF	$3800

Note: 1. 不同的 BOOTSZ 熔丝位配置请参见 Figure 125。

Table 101. RWW 界限 (1)

Flash 区	页数	寻址范围
同时读 - 写区 (RWW)	224	$0000 - $37FF
非同时读 - 写区 (NRWW)	32	$3800 - $3FFF

Note: 1. 关于两个区的详细说明请见 P229 “ 非 RWW 区 –NRWW” 与 P228 “RWW 区 ” 。

Table 102. Figure 126 中所用变量的说明及 Z 指针的映射

变量		相应的 Z 指针数据 (1)	描述
PCMSB	13		程序计数器的最高位 (程序计数器为 14 位 PC[13:0])
PAGEMSB	5		用于页内字寻址的最高位 (一页有 64 个字，需要 6 位 PC [5:0])
ZPCMSB		Z14	Z 寄存器与 PCMSB 对应的位。由于没有使用 Z0 ， ZPCMSB 等于 PCMSB + 1
ZPAGEMSB		Z6	Z 寄存器与 PAGEMSB 对应的位。由于没有使用 Z0 ， ZPAGEMSB 等于 PAGEMSB + 1
PCPAGE	PC[13:6]	Z14:Z7	程序计数器页地址：在页擦除和页写操作中进行页选择
PCWORD	PC[5:0]	Z6:Z1	程序计数器字地址：为填充临时缓冲区进行字选择 (在页写过程中必须为 0)

Note: 1. Z15: 不计
Z0：对所有的 SPM 命令都为 "0”，对 LPM 指令的位选择。
关于自编程过程中 Z 指针的使用请参见 P233 “ 在自编程时访问 Flash” 。

存储器编程

程序及数据存储器锁定位

ATmega32 提供了 6 个锁定位，根据其被编程 (“0”) 还是没有被编程 (“1”) 的情况可以获得 Table 104 列出的附加性能。锁定位只能通过芯片擦除命令擦写为 “1”。

Table 103. 锁定位字节 (1)

锁定位字节	位号	描述	默认值
	7	–	1 (未编程)
	6	–	1 (未编程)
BLB12	5	Boot 锁定位	1 (未编程)
BLB11	4	Boot 锁定位	1 (未编程)
BLB02	3	Boot 锁定位	1 (未编程)
BLB01	2	Boot 锁定位	1 (未编程)
LB2	1	锁定位	1 (未编程)
LB1	0	锁定位	1 (未编程)

Note: 1. “1” 表示未编程，“0” 表示被编程。

Table 104. 锁定位保护模式

存储器锁定位 (2)			保护类型
LB 模式	**LB2**	**LB1**	
1	1	1	没有使能存储器保护特性
2	1	0	在并行和SPI/JTAG串行编程模式中Flash和EEPROM的进一步编程被禁止，熔丝位被锁定。(1)
3	0	0	在并行和SPI/JTAG串行编程模式中Flash和EEPROM的进一步编程及验证被禁止，锁定位和熔丝位被锁定 (1)
BLB0 模式	**BLB02**	**BLB01**	
1	1	1	SPM 和 LPM 对应用区的访问没有限制
2	1	0	不允许 SPM 对应用区进行写操作
3	0	0	不允许 SPM 指令对应用区进行写操作，也不允许运行于 Boot Loader 区的 LPM 指令从应用区读取数据。若中断向量位于 Boot Loader 区，那么执行应用区代码时中断是禁止的。
4	0	1	不允许运行于 Boot Loader 区的 LPM 指令从应用区读取数据。若中断向量位于 Boot Loader 区，那么执行应用区代码时中断是禁止的。
BLB1 模式	**BLB12**	**BLB11**	
1	1	1	允许 SPM/LPM 指令访问 Boot Loader 区

Table 104. 锁定位保护模式 (Continued)

存储器锁定位 (2)			保护类型
2	1	0	不允许 SPM 指令对 Boot Loader 区进行写操作
3	0	0	不允许 SPM 指令对 Boot Loader 区进行写操作，也不允许运行于应用区的 LPM 指令从 Boot Loader 区读取数据。若中断向量位于应用区，那么执行 Boot Loader 区代码时中断是禁止的。
4	0	1	不允许运行于应用区的 LPM 指令从 Boot Loader 区读取数据。若中断向量位于应用区，那么执行 Boot Loader 区代码时中断是禁止的。

Notes: 1. 在编程锁定位前先编程熔丝位。
2. “1” 表示未被编程 , “0” 表示被编程。

熔丝位

ATmega32 有两个熔丝位字节。Table 105 - Table 106 简单地描述了所有熔丝位的功能以及他们是如何映射到熔丝字节的。如果熔丝位被编程则读返回值为 “0”。

Table 105. 熔丝位高字节

熔丝高字节	位号	描述	默认值
OCDEN(4)	7	使能 OCD	1 (未编程， OCD 禁用)
JTAGEN(5)	6	使能 JTAG	0 (已编程，JTAG 使能)
SPIEN(1)	5	使能串行程序和数据下载	0 (已编程，SPI 编程使能)
CKOPT(2)	4	振荡器选项	1 (未编程)
EESAVE	3	执行芯片擦除时 EEPROM 的内容保留	1 (未编程)，EEPROM 内容不保留
BOOTSZ1	2	选择 Boot 区大小 (详见 Table 100)	0 (被编程)(3)
BOOTSZ0	1	选择 Boot 区大小 (详见 Table 100)	0 (被编程)(3)
BOOTRST	0	选择复位向量	1 (未被编程)

Notes: 1. 在 SPI 串行编程模式下 SPIEN 熔丝位不可访问。
2. CKOPT 熔丝位功能由 CKSEL 位设置决定，详见 P23 “ 时钟源 ” 。
3. BOOTSZ1..0 默认值为最大 Boot 大小，详见 P237 Table 100 。
4. 不论锁位与 JTAGEN 熔丝位设置为什么 ,产品出厂时不对 OCDEN 编程。对 OCDEN 熔丝位编程后会使能系统时钟的某些部分在所有的休眠模式下运行。这会增加功耗。
5. 如果没有连接 JTAG 接口，应尽可能取消 JTAGEN 熔丝位的编程状态，以消除存在于 JTAG 接口之 TDO 引脚的静态电流。

Table 106. 熔丝位低位字节

熔丝位低位字节	位号	描述	默认值
BODLEVEL	7	BOD 触发电平	1 (未编程)
BODEN	6	BOD 使能	1 (未编程，BOD 禁用)
SUT1	5	选择启动时间	1 (未编程)(1)
SUT0	4	选择启动时间	0 (已编程)(1)
CKSEL3	3	选择时钟源	0 (已编程)(2)

Table 106. 熔丝位低位字节

熔丝位低位字节	位号	描述	默认值
CKSEL2	2	选择时钟源	0 (已编程)(2)
CKSEL1	1	选择时钟源	0 (已编程)(2)
CKSEL0	0	选择时钟源	1 (未编程)(2)

Notes: 1. 对于默认时钟源 ,SUT1..0 的默认值给出最大的启动时间。详细内容见 P28 Table 10 。
2. CKSEL3..0的默认设置导致了片内RC振荡器运行于1 MHz。详细内容见 P23 Table 2 。

熔丝位的状态不受芯片擦除命令的影响。如果锁定位 1(LB1) 被编程则熔丝位被锁定。在编程锁定位前先编程熔丝位。

锁存熔丝位的数据

芯片进入编程模式时熔丝位的值被锁存。其间熔丝位的改变不会生效 , 直到器件退出编程模式。不过这不适用于 EESAVE 熔丝位。它一旦被编程立即起作用。在正常工作模式中器件上电时熔丝位也被锁存。

标识字节

所有的 Atmel 微控制器都具有一个三字节的标识代码用来区分器件型号。这个代码可以通过串行和并行模式读取 , 也可以在芯片被锁定时读取。这三个字节分别存储于三个独立的地址空间。

ATmega32 标识字节为 :

1. 0x000: 0x1E (表示由 Atmel 公司生产)
2. 0x001: 0x95 (表示芯片包含 32 KB Flash 存储器)
3. 0x002: 0x02 (当 0x001 字节的内容为 0x95 时表示这是 ATmega32)

标定字节

ATmega32 内部 RC 振荡器的有四个不同的校准值保存于校准字节。这个字节位于标识地址空间 0x000、 0x0001、 0x0002 及 0x0003 的高位字节 , 分别标定 1、2、 4、8 MHz。在复位期间 , 1 MHz 的标定值被自动写入 OSCCAL 寄存器。若需要其他频率标定值 , 则需手动完成 , 详见 P28 “ 振荡器标定寄存器 –OSCCAL” 。

并行编程参数，引脚映射及命令

这部分描述了如何对 ATmega32 的 Flash 程序存储器，EEPROM 数据存储器，存储锁定位及熔丝位进行并行编程和校验。除非另有说明，脉冲宽度至少为 250 ns。

信号名称

在这一节 ATmega32 的相关引脚以并行编程信号的名称进行引用，如 Figure 127 与 Table 107 所示。表中没有描述的引脚沿用原来的称谓。

XA1/XA0 决定了给 XTAL1 引脚一个正脉冲时所执行的操作。具体编码请见 Table 109。

给 $\overline{WR}$ 或 $\overline{OE}$ 输入脉冲时所加载的命令决定了要执行的操作。具体命令请参见 Table 110。

Figure 127. 并行编程

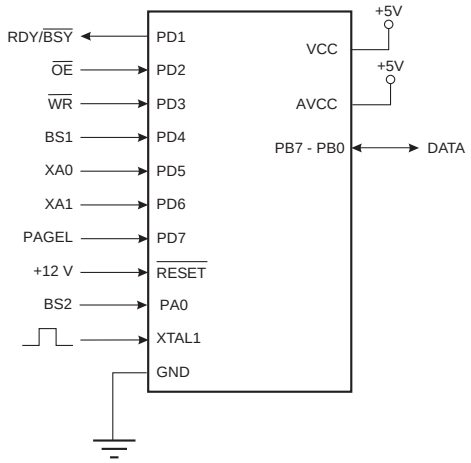


Table 107. 引脚名称映射

编程模式信号的名称	引脚名称	I/O	功能
RDY/$\overline{BSY}$	PD1	O	0: 芯片忙于编程，1: 芯片等待新的命令。
$\overline{OE}$	PD2	I	输出使能 (低电平有效).
$\overline{WR}$	PD3	I	写脉冲 (低电平有效).
BS1	PD4	I	字节选择 1(“0” 选择低位字节，“1” 选择高位字节).
XA0	PD5	I	XTAL 动作位 0
XA1	PD6	I	XTAL 动作位 1
PAGEL	PD7	I	加载程序存储器和 EEPROM 数据页
BS2	PA0	I	字节选择 2(“0” 选择低位字节，“1” 选择第二个高位字节)
DATA	PB7-0	I/O	双向数据总线 ($\overline{OE}$ 为低时输出)

Table 108. 进入编程模式所需要的引脚数据

引脚	符号	数值
PAGEL	Prog_enable[3]	0
XA1	Prog_enable[2]	0
XA0	Prog_enable[1]	0
BS1	Prog_enable[0]	0

Table 109. XA1 和 XA0 的编码

XA1	XA0	给 XTAL1 施加脉冲激发的动作
0	0	加载 Flash 或 EEPROM 地址 (通过 BS1 确定是高位还是低位字节)
0	1	加载数据 (通过 BS1 决定是高位还是低位闪存数据字节)
1	0	加载命令
1	1	无操作，空闲

Table 110. 命令字节编码

命令字节	执行的命令
1000 0000	芯片擦除
0100 0000	写熔丝位
0010 0000	写锁定位
0001 0000	写 Flash
0001 0001	写 EEPROM
0000 1000	读标识字节和校准字节
0000 0100	读熔丝位和锁定位
0000 0010	读 Flash
0000 0011	读 EEPROM

Table 111. 一页包含的字和 Flash 中的页数

Flash 大小	页大小	PCWORD	页号	PCPAGE	PCMSB
16K 字 (32K 字节)	64 字	PC[5:0]	256	PC[13:6]	13

Table 112. 一页包含的字和 EEPROM 中的页数

EEPROM 大小	页大小	PCWORD	页数	PCPAGE	EEAMSB
1024 字节	4 字节	EEA[1:0]	256	EEA[9:2]	9

并行编程

进入编程模式

通过下面的算法进入并行编程模式：

1. 在 V_{CC} 及 GND 之间提供 4.5 - 5.5V 的电压
2. 将 $\overline{\text{RESET}}$ 拉低，并至少改变 XTAL1 电平 6 次
3. 将 P242 Table 108 中列出的的 Prog_enable 引脚置为 "0000" , 并等待至少 100 ns
4. 给 $\overline{\text{RESET}}$ 提供 11.5 - 12.5V 的电压。在向 $\overline{\text{RESET}}$ 提供 +12V 电压后的 100 ns 内，Prog_enable 引脚的任何行为都会导致芯片无法进入编程模式

如果选择外部晶体或外部 RC，它不可能提供合格的 XTAL1 脉冲。在这种情况下，应采取如下算法：

1. 设置列于 P242 Table 108 的 Prog_enable 引脚为 “0000”。
2. 在 V_{CC} 与 GND 间提供电压 4.5 - 5.5V 同时在 $\overline{\text{RESET}}$ 上提供 11.5 - 12.5V 电压。
3. 等待 100 ns。
4. 对熔丝位重编程，保证外部时钟源作为系统时钟 (CKSEL3:0 = 0b0000)。如果锁定位已编程，在改变熔丝前必须执行芯片擦除指令。
5. 通过降低器件功率或置 $\overline{\text{RESET}}$ 引脚为 0b0 来退出编程模式。
6. 用前面讲到的算法进入编程模式。

进行高效编程需要考虑的问题

在编程过程中，加载的命令及地址保持不变。为了实现高效的编程应考虑以下因素：

- 对多个存储单元进行读或写操作时，命令仅需加载一次
- 当需要写入的数据为 0xFF 时可以跳过，因为这就是执行全片擦除命令后 Flash 及 EEPROM(除非 EESAVE 熔丝位被编程) 的内容。
- 只有在编程或读取 Flash 及 EEPROM 中新的 256 字时才需要用到地址高位字节。在读标识字节时也需考虑这一点。

芯片擦除

芯片擦除操作会擦除 Flash 及 EEPROM(1) 存储器以及锁定位。程序存储器没有擦除结束之前锁定位不会复位。全片擦除不影响熔丝位。芯片擦除命令必须在编程 Flash 与 / 或 EEPROM 之前完成。

Note: 1. 如果 EESAVE 熔丝位被编程，那么在芯片擦除时 EEPRPOM 不受影响。

加载 " 芯片擦除 ” 命令的过程：

1. 将 XA1、 XA0 置为 "10” 以启动命令加载。
2. 将 BS1 置为 "0”。
3. DATA 赋值为 “1000 0000”。这是芯片擦除命令。
4. 给 XTAL1 提供一个正脉冲，进行命令加载。
5. 给 $\overline{\text{WR}}$ 提供一个负脉冲，启动芯片擦除。 RDY/$\overline{\text{BSY}}$ 变低。
6. 等待 RDY/$\overline{\text{BSY}}$ 变高，然后才能加载新的命令。

对 Flash 进行编程

Flash 是以页的形式组织起来的 , 如 P243 Table 111 所示。编程 Flash 时 , 程序数据被锁存到页缓冲区中。这样一整页的程序数据可以同时得到编程。下面的步骤描述了如何对 Flash 进行编程：

A. 加载 " 写 Flash" 命令

1. 将 XA1、 XA0 置为 "10"，启动命令加载
2. 将 BS1 置 "0”
3. DATA 赋值为 “0001 0000”，这是写 Flash 命令
4. 给 XTAL1 提供一个正脉冲以加载命令

B. 加载地址低位字节

1. 将 XA1、 XA0 置为 "00"，启动地址加载
2. 将 BS1 置 "0”，选择低位地址
3. DATA 赋值为地址低位字节 (0x00 - 0xFF)
4. 给 XTAL1 提供一个正脉冲，加载地址低位字节

C. 加载数据低位字节

1. 将 XA1、 XA0 置为 "01"，启动数据加载
2. DATA 赋值为数据低位字节 (0x00 - 0xFF)
3. 给 XTAL1 提供一个正脉冲，加载数据字节

D. 加载数据高位字节

1. 将 BS1 置为 "1”，选择数据高位字节
2. 将 XA1、 XA0 置为 "01"，启动数据加载
3. DATA 赋值为数据高位字节 (0x00 - 0xFF)
4. 给 XTAL1 提供一个正脉冲，进行数据字节加载

E. 锁存数据

1. 将 BS1 置为 "1”，选择数据高位字节
2. 给 PAGEL 提供一个正脉冲，锁存数据 (见 Figure 129 信号波形)

F. 重复 B 到 E 操作，直到整个缓冲区填满或此页中所有的数据都已加载

地址信息中的低位用于页内寻址，高位用于 FLASH 页的寻址，详见 P246 Figure 128 。如果页内寻址少于 8 位 (页地址 < 256) ,那么进行页写操作时地址低字节中的高位用于页寻址。

G. 加载地址高位字节

1. 将 XA1、 XA0 置为 "00”，启动地址加载操作
2. 将 BS1 置为 "1”，选择高位地址
3. DATA 赋值为地址高位字节 (0x00 - 0xFF)
4. 给 XTAL1 提供一个正脉冲，加载地址高位字节

H. 编程一页数据

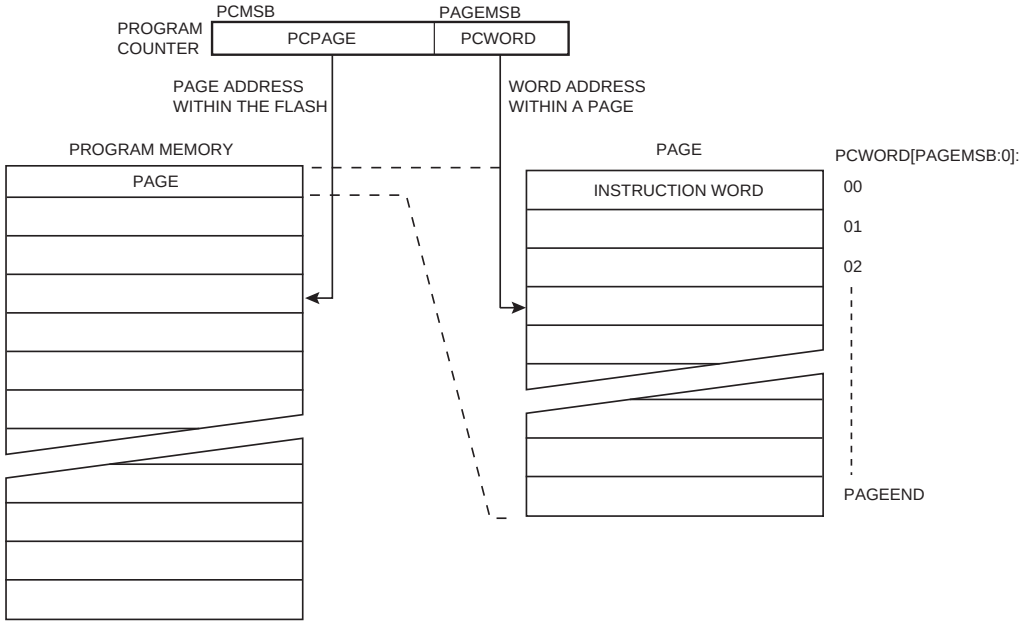
1. 置 BS1 = “0”
2. 给 $\overline{WR}$ 提供一个负脉冲，对整页数据进行编程， RDY/$\overline{BSY}$ 变低
3. 等待 RDY/$\overline{BSY}$ 变高 (见 Figure 129 的信号波形)

I. 重复 B 到 H 的操作，直到整个 Flash 编程结束或者所有的数据都被编程

J. 结束页编程

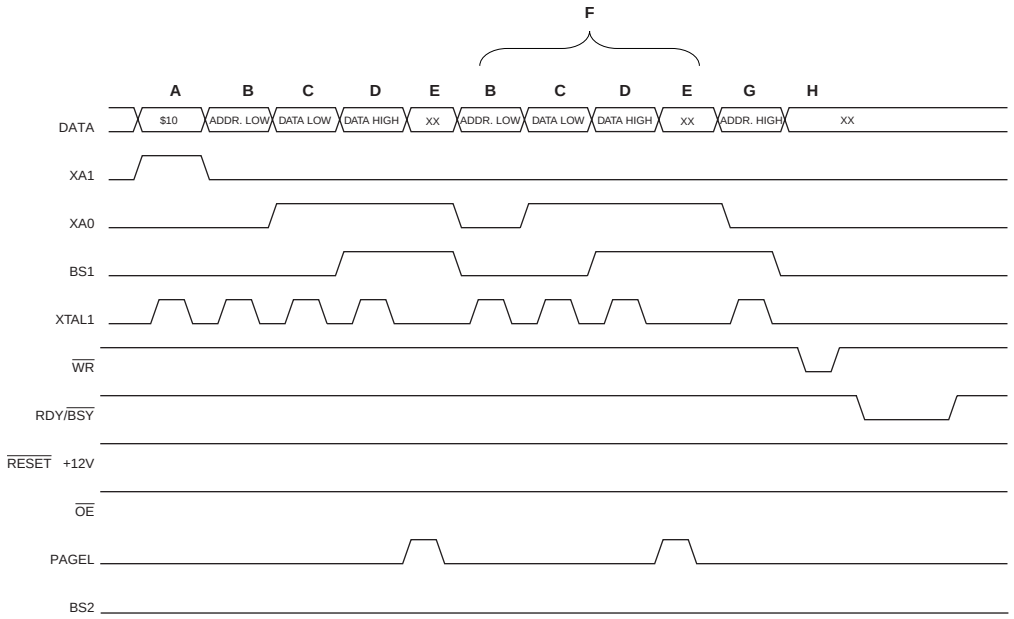
1. 1. 将 XA1、 XA0 置为 "10”，启动命令加载操作
2. DATA 赋值为 "0000 0000"，这是不操作指令
3. 给 XTAL1 提供一个正脉冲，加载命令，内部写信号复位

Figure 128. 对以页为组织单位的 Flash 进行寻址



Note: 1. PCPAGE 及 PCWORD 列于 P243 Table 111 。

Figure 129. Flash 编程波形 (1)



Note: 1. 不用考虑 "XX"，各个大写字母对应于前面描述的 Flash 编程阶段。

对 EEPROM 进行编程

如 P243 Table 112 所示， EEPROM 也以页为单位。编程 EEPROM 时，编程数据锁存于页缓冲区中。这样可以同时对一页数据进行编程。EEPROM 数据存储器编程算法如下 (命令、地址及数据加载的细节请参见 P245 “ 对 Flash 进行编程 ”) :

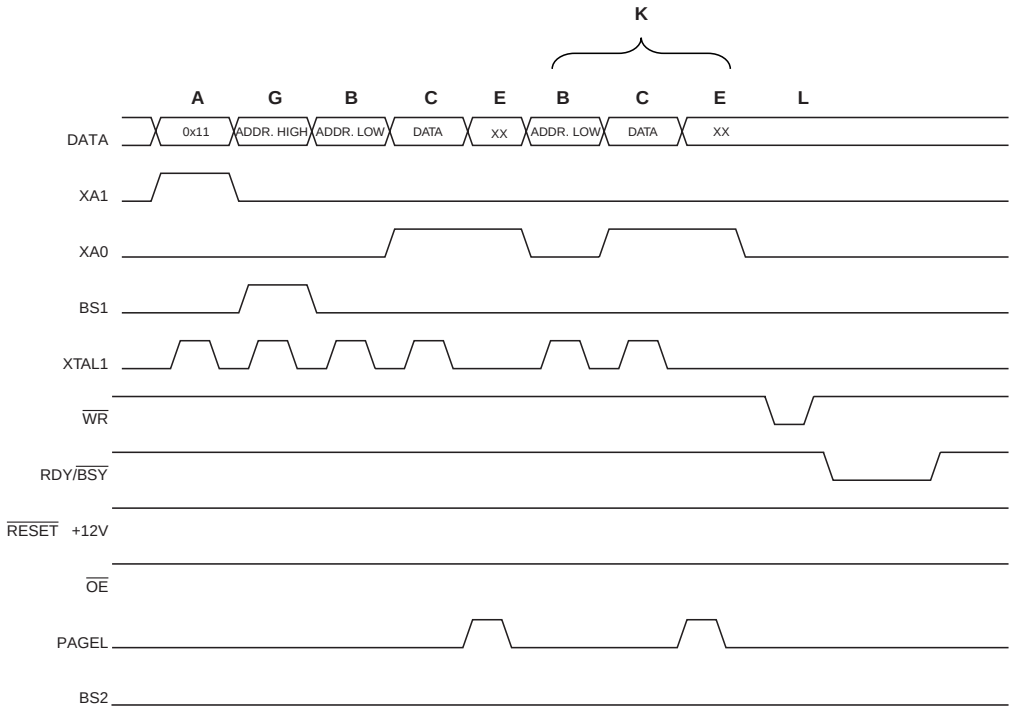
1. A：加载命令 “0001 0001”
2. G：加载地址高位字节 (0x00 - 0xFF)
3. B：加载地址低位字节 (0x00 - 0xFF)
4. C：加载数据 (0x00 - 0xFF)
5. E：锁存数据 (给 PAGEL 提供一个正脉冲)

K：重复步骤 3 到 5，直到整个缓冲区填满

L：对 EEPROM 页进行编程

1. 将 BS1 置 “0”
2. 给 $\overline{WR}$ 提供一个负脉冲，开始对 EEPROM 页进行编程， RDY/$\overline{BSY}$ 变低
3. 等到 RDY/$\overline{BSY}$ 变高再对下一页进行编程 (信号波形见 Figure 130)

Figure 130. EEPROM 编程波形



读取 Flash

读 Flash 存储器的过程如下 (命令及地址加载细节见 P245 “ 对 Flash 进行编程 ”) :

1. A：加载命令 “0000 0010”
2. G：加载地址高位字节 (0x00 - 0xFF)
3. B：加载地址低位字节 (0x00 - 0xFF)
4. 将 $\overline{OE}$ 置 “0”， BS1 置 “0”，然后从 DATA 读出 Flash 字的低位字节
5. 将 BS1 置 “1”，然后从 DATA 读出 Flash 字的高位字节
6. 将 $\overline{OE}$ 置 “1”

读取 EEPROM

读存储器的步骤如下 (命令及地址加载细节见 P245 “ 对 Flash 进行编程 ”) :

1. A：加载命令 “0000 0011”
2. G：加载地址高位字节 (0x00 - 0xFF)
3. B：加载地址低位字节 (0x00 - 0xFF)
4. 将 $\overline{OE}$ 置 "0”，BS1 置 "0”，然后从 DATA 读出 EEPROM 数据字节
5. 将 $\overline{OE}$ 置 ”1”

对熔丝位的低位进行编程

对熔丝低位的编程步骤如下 (命令及数据装在细节见 P245 “ 对 Flash 进行编程 ”) :

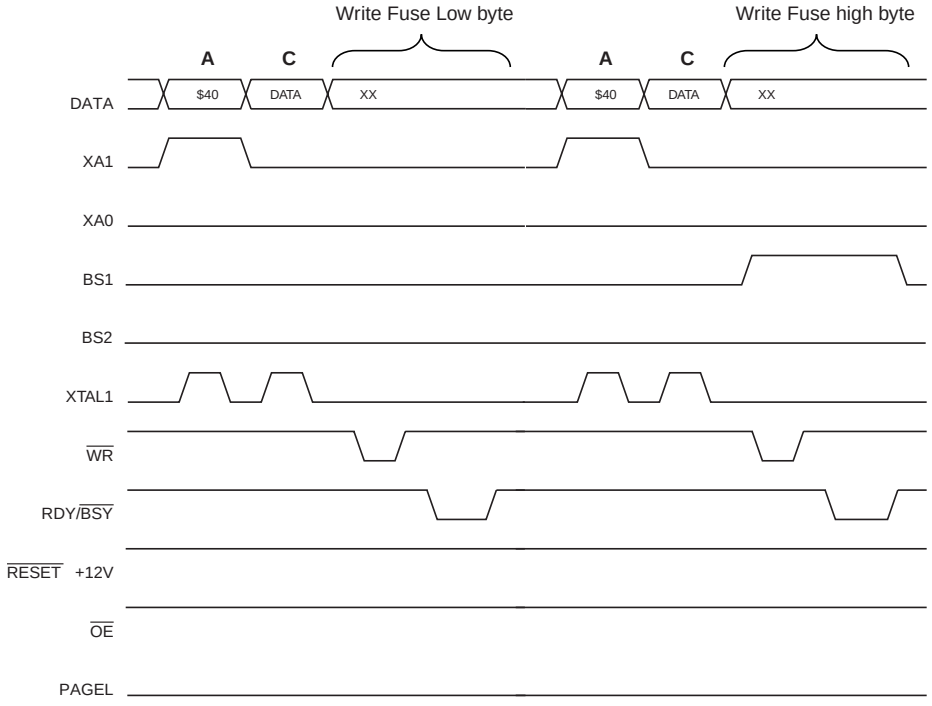
1. A：加载命令 “0100 0000”
2. C：加载数据低字节，若某一位为 "0” 表示需要进行编程，否则需要擦除
3. 置 BS1 为 “0”，BS2 为 “0”。
4. 给 $\overline{WR}$ 提供一个负脉冲，并等待 RDY/$\overline{BSY}$ 变高

对熔丝位的高位进行编程

对熔丝高位的编程步骤如下 (命令及数据装在细节见 P245 “ 对 Flash 进行编程 ”) :

1. A：加载命令 “0100 0000”
2. C：加载数据高字节，若某一位为 "0” 表示需要进行编程，否则需要擦除
3. 将 BS1 置 "1”、 BS2 置 "0”，选择高位数据字节
4. 给 $\overline{WR}$ 提供一个负脉冲并等待 RDY/$\overline{BSY}$ 变高
5. 将 BS1 置 "0”，选择低位字节

Figure 131. 熔丝位编程波形



对锁定位进行编程

锁定位编程步骤如下 (命令及数据装在细节见 P245 “ 对 Flash 进行编程 ”) :

1. A：加载命令 “0010 0000”
2. C：加载数据低字节，位 n 为 "0” 表示此锁定位需要编程。
3. 给 $\overline{WR}$ 提供一个负脉冲并等待 RDY/$\overline{BSY}$ 变高

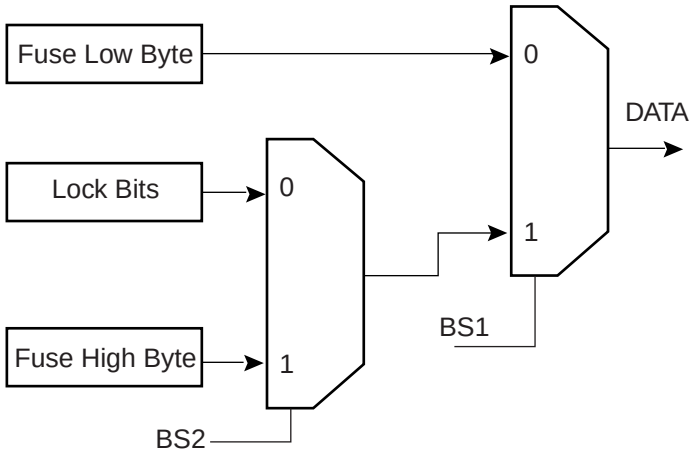
锁定位只能通过芯片擦除命令来清除。

读取熔丝位和锁定位

读取熔丝位及锁定位的步骤如下 (命令加载细节见 P245 “ 对 Flash 进行编程 ”) :

1. A：加载命令 “0000 0100”
2. 将 $\overline{OE}$、BS2 和 BS1 置 "0”，然后从 DATA 读取熔丝低位的状态 ("0” 表示已编程)
3. 将 $\overline{OE}$ 置 "0” ,BS2 和 BS1 置 “1” ,然后从 DATA 读取熔丝高位的状态 ("0” 表示已编程)
4. 将 $\overline{OE}$ 置 "0” ,BS2 置 "0” ,BS1 置 “1” ,然后从 DATA 读取锁定位的状态 ("0” 表示已编程)
5. 将 $\overline{OE}$ 置 “1”。

Figure 132. 读操作过程中 BS1、 BS2 与熔丝位及锁定位的对应关系



读取标识字节

读取标识字节的算法如下 (命令与地址加载参考 P245 “ 对 Flash 进行编程 ”) :

1. A：加载命令 “0000 1000”
2. B：加载地址低字节 0x00 - 0x02
3. 将 $\overline{OE}$ 、 BS1 置 "0”，然后从 DATA 读取标识字节
4. 将 $\overline{OE}$ 置 "1”

读取标定字节

读取校准字节的算法如下 (命令与地址加载参考 P245 “ 对 Flash 进行编程 ”) :

1. A：加载命令 “0000 1000”
2. B：加载地址低字节
3. 将 $\overline{OE}$ 置 "0”，BS1 置 “1”，然后从 DATA 读取校准字节
4. 将 $\overline{OE}$ 置 “1”

并行编程特性

Figure 133. 并行编程时序，包括一些常规的时序要求

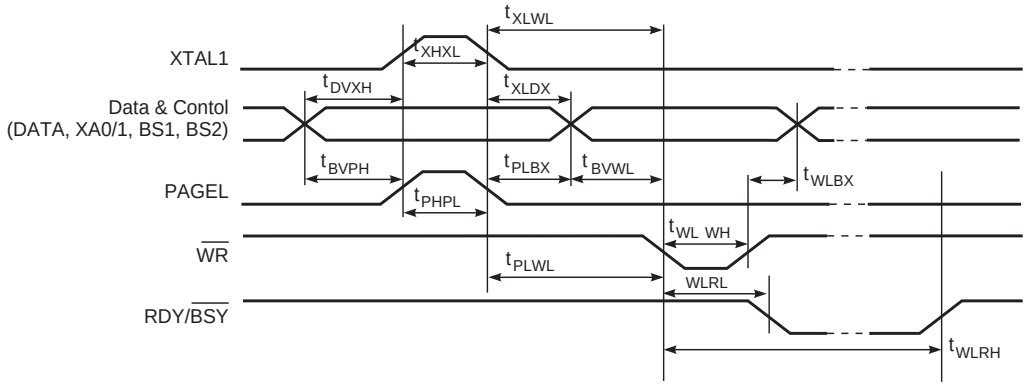
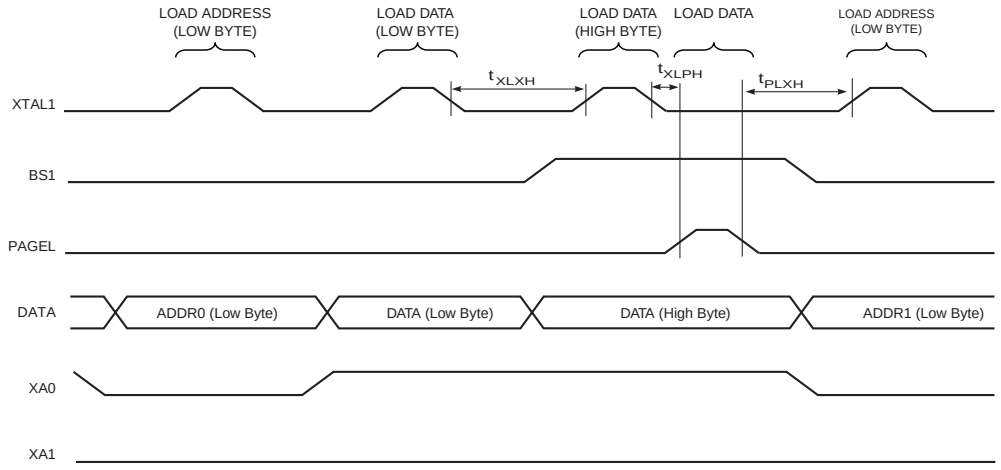
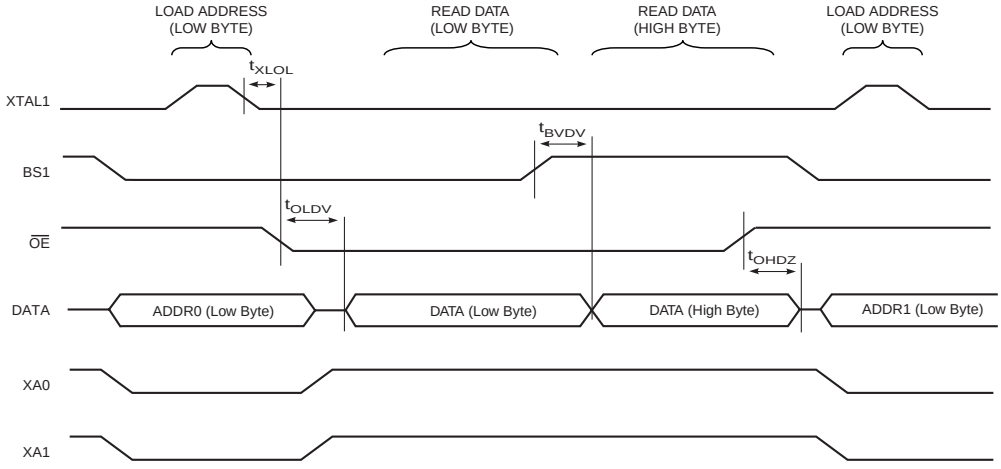


Figure 134. 并行编程时序，有时序要求的加载序列 (1)



Note: 1. Figure 133 给出的时序要求 (t_{DVXH}、 t_{XHXL} 及 t_{XLDX}) 也适用于加载操作。

Figure 135. 并行编程时序，有时序要求的读序列 (同一页)(1)



Note: 1. Figure 133 给出的时序要求 (即 t_{DVXH}、 t_{XHXL} 及 t_{XLDX}) 也适用于读操作。

Table 113. 并行编程参数， V_{CC} = 5 V ± 10%

符号	参数	最小值	典型值	最大值	单位
V_{PP}	编程使能电压	11.5		12.5	V
I_{PP}	编程使能电流			250	µA
t_{DVXH}	在 XTAL1 为高之前数据及控制有效	67			ns
t_{XLXH}	从 XTAL1 低到 XTAL1 高	200			ns
t_{XHXL}	XTAL1 为高时的脉宽	150			ns
t_{XLDX}	XTAL1 为低之后数据及控制保持	67			ns
t_{XLWL}	从 XTAL1 低到 $\overline{WR}$ 低	0			ns
t_{XLPH}	从 XTAL1 低到 PAGEL 高	0			ns
t_{PLXH}	从 XTAL1 低到 PAGEL 高	150			ns

Table 113. 并行编程参数， V_{CC} = 5 V ± 10% (Continued)

符号	参数	最小值	典型值	最大值	单位
t_{BVPH}	从 PAGEL 低到 XTAL1 高	67			ns
t_{PHPL}	PAGEL 为高之前 BS1 有效	150			ns
t_{PLBX}	PAGEL 为高时的脉宽	67			ns
t_{WLBX}	PAGEL 为低之后 BS1 保持	67			ns
t_{PLWL}	$\overline{WR}$ 为低之后 BS2/1 保持	67			ns
t_{BVWL}	从 PAGEL 低到 $\overline{WR}$ 为低	67			ns
t_{WLWH}	BS1 有效至 $\overline{WR}$ 为低	150			ns
t_{WLRL}	从 $\overline{WR}$ 低到 RDY/$\overline{BSY}$ 为低	0		1	µs
t_{WLRH}	从 $\overline{WR}$ 低到 RDY/$\overline{BSY}$ 为高 (1)	3.7		4.5	ms
t_{WLRH_CE}	从 $\overline{WR}$ 低到 RDY/$\overline{BSY}$ 为高，芯片擦除操作 (2)	7.5		9	ms
t_{XLOL}	从 XTAL1 低到 $\overline{OE}$ 为低	0			ns
t_{BVDV}	BS1 有效至 DATA 有效	0		250	ns
t_{OLDV}	从 $\overline{OE}$ 低到 DATA 有效			250	ns
t_{OHDZ}	从 $\overline{OE}$ 低到 DATA 为高阻态			250	ns

Notes: 1. 在进行 Flash、 EEPROM、熔丝位及锁定位写操作时 t_{WLRH} 有效。
2. 在执行芯片擦除操作时 t_{WLRH_CE} 有效。

SPI 串行下载

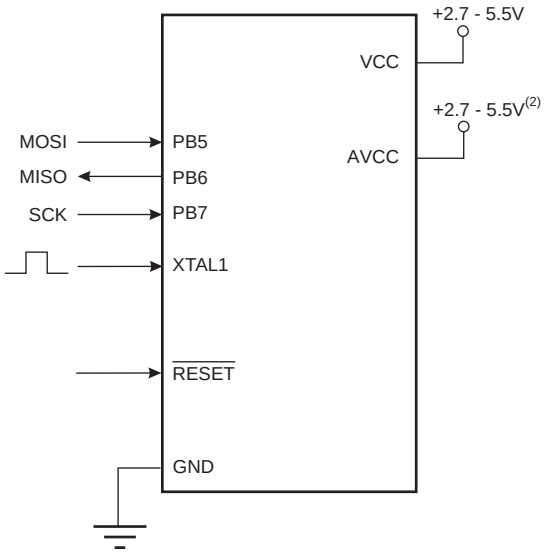
当 $\overline{RESET}$ 为低电平时，可以通过串行 SPI 总线对 Flash 及 EEPROM 进行编程。串行接口包括 SCK、 MOSI(输入) 及 MISO(输出)。 $\overline{RESET}$ 为低之后，应在执行编程 / 擦除操作之前执行编程允许指令。P251 Table 114 列出了 SPI 编程所需引脚的映射。不是所有的器件都使用 SPI 引脚专用于内部 SPI 接口。

SPI 串行编程引脚映射

Table 114. 串行编程映射

符号	引脚	I/O	说明
MOSI	PB5	I	连续数据输入
MISO	PB6	O	连续数据输出
SCK	PB7	I	连续时钟

Figure 136. SPI 串行编程及校验 (1)



Notes: 1. 如果芯片由片内振荡器提供时钟，那么就不用在 XTAL1 引脚上连接时钟源。
2. V_{CC} -0.3V < AVCC < V_{CC} +0.3V，但是 AVCC 必须在 2.7 - 5.5V 范围内。

编程 EEPROM 时， MCU 在自定时的编程操作中会插入一个自动擦除周期，从而无需执行芯片擦除命令。芯片擦除操作将程序存储器及 EEPROM 的内容都擦除为 0xFF。

时钟通过 CKSEL 熔丝位确定。串行时钟 (SCK) 的最小低电平时间和最小高电平时间要满足如下要求：

低 :> f_{ck} < 12 MHz 时为 2 个 CPU 时钟周期，f_{ck} ≥12 MHz 时为 3 个 CPU 时钟周期。

高 :> f_{ck} < 12 MHz 时为 2 个 CPU 时钟周期，f_{ck} ≥12 MHz 时为 3 个 CPU 时钟周期。

SPI 串行编程算法

向 ATmega32 串行写入数据时，数据在 SCK 的上升沿得以锁存。

从 ATmega32 读取数据时，数据在 SCK 的下降沿输出。时序细节见 Figure 137 。

在串行编程模式下对 ATmega32 进行编程及校验时，应遵循以下的步骤 (见 Table 116 中的 4 字节指令格式) :

1. 上电顺序 :
在 RESET 及 SCK 为 "0" 时，向 V_{CC} 及 GND 供电。在一些系统中，编程器不能保证在上电时 SCK 保持为低。在这种情况下， SCK 拉低之后应在 RESET 加一正脉冲，而且这个脉冲至少要维持 2 个 CPU 时钟周期。
2. 上电之后等待至少 20 ms，然后向 MOSI 引脚输入串行编程使能指令以使能串行编程。
3. 通信不同步将造成串行编程指令不工作。同步之后，在发送编程使能指令的第三个字节时，第二个字节的内容 (0x53) 将被反馈回来。不论反馈的内容正确与否，指令的 4 个字节必须全部传输。如果 0x53 未被反馈，则需要向 RESET 提供一个正脉冲以开始新的编程使能指令。
4. Flash 的编程以一次一页的方式进行。在执行加载程序存储页指令时，通过 6 LSB 的地址信息，数据以字节为单位加载到存储页。为保证加载的正确性，应先向给定地址传送数据低字节，之后是高字节。程序存储页通过地址的高 7 位以及写程序存储器页指令获得数据。如果不使用查询的方式，那么在操作下一页数据之前应等待至少 t_{WD_FLASH} 的时间 (见 Table 115)。在 Flash 写操作完成之前访问串行编程接口会导致编程错误。

5. 提供了地址及数据信息之后，适合的写指令将以字节为单位对 EEPROM 编程。EEPROM 存储单元总是在写入新数据之前自动擦除。如果不使用查询的方式，那么在操作下一页数据之前应等待至少 t_{WD_EEPROM} 的时间 (见 Table 115)。对于全片擦除之后的芯片，数据为 0xFF 的不需要编程。
6. 可通过读指令来校验任何一个存储单元的内容。数据从串行输出口 MISO 输出。
7. 编程结束后可以将 $\overline{\text{RESET}}$ 拉高开始正常操作。
8. 下电序列 (如果需要)：
   将 $\overline{\text{RESET}}$ 置 “1”。
   切断 V_{CC}。

Flash 的数据轮询

当 Flash 正处于某一页的编程状态时，读取此页中的内容将得到 0xFF。编程结束后，被编程的数据即可以正确读出。通过这种方法可以确定何时可以写下一页。由于整个页是同时编程的，这一页中的任何一个地址都可以用来查询。 Flash 数据查询不适用于数据 0xFF。因此，在编程 0xFF 时，用户至少要等待 t_{WD_FLASH} 才能进行下一页的编程。由于全片擦除将所有的单元擦为 0xFF，所以编程数据为 0xFF 时可以跳过这个操作。 t_{WD_FLASH} 的值见 Table 115。

EEPROM 的数据轮询

当 EEPROM 正在处理一个字节的编程操作时，读取此地址将返回 0xFF。编程结束后，被编程的数据即可以正确读出。这一方法可用来判断何时可以写下一个字节。数据查询对数据 0xFF 无效。但用户应该考虑到，全片擦除将所有的单元擦为 0xFF，所以编程数据为 0xFF 时可以跳过这个操作。不过这不适用于全片擦除时 EEPROM 内容被保留的情况。用户若在此时编程 0xFF，在进行下一字节编程之前至少等待 t_{WD_EEPROM} 的时间。t_{WD_EEPROM} 的值见 Table 115。

Table 115. 写下一个 Flash 或 EEPROM 单元之前的最小等待时间

符号	最小等待时间
t_{WD_FLASH}	4.5 ms
t_{WD_EEPROM}	9.0 ms
t_{WD_ERASE}	9.0 ms

Figure 137. SPI 串行编程波形图

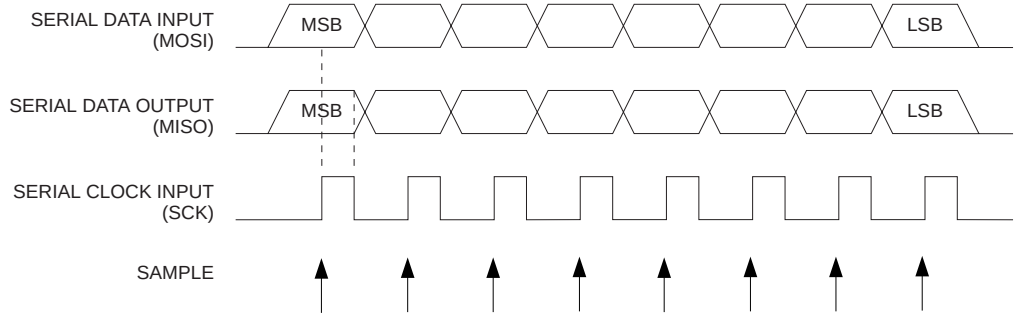


Table 116. SPI 串行编程指令集

指令	指令格式				操作
	字节 1	字节 2	字节 3	字节 4	
编程使能	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	$\overline{\text{RESET}}$ 拉低后使能串行编程
芯片擦除	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	擦除 EEPROM 及 Flash
读程序存储器	0010 **H**000	00**aa aaaa**	**bbbb bbbb**	**oooo oooo**	从字地址为 **a**:**b** 的程序存储器读取 **H**(高或低字节) 数据的 **o**
加载程序存储器页	0100 **H**000	00xx xxxx	xx**bb bbbb**	**iiii iiii**	向字地址为 **b** 的程序存储页 **H**(高或低字节) 写入数据 **i**。应先写低字节再写高字节
写程序存储器页	0100 1100	00**aa aaaa**	**bb**xx xxxx	xxxx xxxx	在地址 **a**:**b** 加载程序存储页
读 EEPROM 存储器	1010 0000	00xx xx**aa**	**bbbb bbbb**	**oooo oooo**	从 EEPROM 的地址 **a**:**b** 处读出数据 **o**
写 EEPROM 存储器	1100 0000	00xx xx**aa**	**bbbb bbbb**	**iiii iiii**	向 EEPROM 地址 **a**:**b** 处中写入数据 **i**
读锁定位	0101 1000	0000 0000	xxxx xxxx	xx**oo oooo**	读锁定位。”0” 为已编程，"1” 为未编程。细节见 P239 Table 103 。
写锁定位	1010 1100	111x xxxx	xxxx xxxx	11**ii iiii**	写锁定位。写 “0” 表示编程锁定位。细节见 P239 Table 103 。
读标识字节	0011 0000	00xx xxxx	xxxx xx**bb**	**oooo oooo**	从地址 **b** 读取标识字节 **o**.
写熔丝位	1010 1100	1010 0000	xxxx xxxx	**iiii iiii**	“0” 表示已编程，“1” 表示未编程。见 P240 Table 106 。
写高熔丝位	1010 1100	1010 1000	xxxx xxxx	**iiii iiii**	“0” 表示已编程，“1” 表示未编程。见 P240 Table 105 。
读熔丝位	0101 0000	0000 0000	xxxx xxxx	**oooo oooo**	读熔丝位。“0” 表示已编程，“1” 表示未编程。细节见 P240 Table 106 。
读高熔丝位	0101 1000	0000 1000	xxxx xxxx	**oooo oooo**	读熔丝高位。“0” 表示已编程，“1” 表示未编程。细节见 P240 Table 105 。
读校准字节	0011 1000	00xx xxxx	0000 0000	**oooo oooo**	读校准字节

Note: **a** = 地址高位，**b** = 地址低位， **H** =0 - 低字节， 1 - 高字节， **o** = 数据输出，**i** = 数据输入，x = 任意值

SPI 串行编程特性

对 SPI 模块特性，请参见 P273 “SPI 时序特性 ” 。

通过 JTAG 接口进行编程

通过 JTAG 接口进行编程需要控制 4 个 JTAG 专用引脚 :TCK、TMS、TDI 及 TDO。reset 及时钟引脚不用控制。

使用 JTAG 接口之前首先要编程 JTAGEN 熔丝位。芯片出厂时这个熔丝位缺省为编程状态。此外 ,MCUCSR 寄存器的 JTD 位必须清零。如果 JTD 已被置 1 ,则可以将外部 reset 强制拉低。经过两个时钟周期之后 JTD 位就清零了。 JTAG 引脚即可用于编程功能。因此，JTAG 引脚除了可以用作通用 I/O 之外还可以用于 ISP 功能。但要注意，当 JTAG 用于边界扫描或片内调试时，不可以使用这个技术。在这种情况下，JTAG 引脚只能用作上述用途。

在本手册定义中， LSB 是第一个移入 / 移出移位寄存器的位。

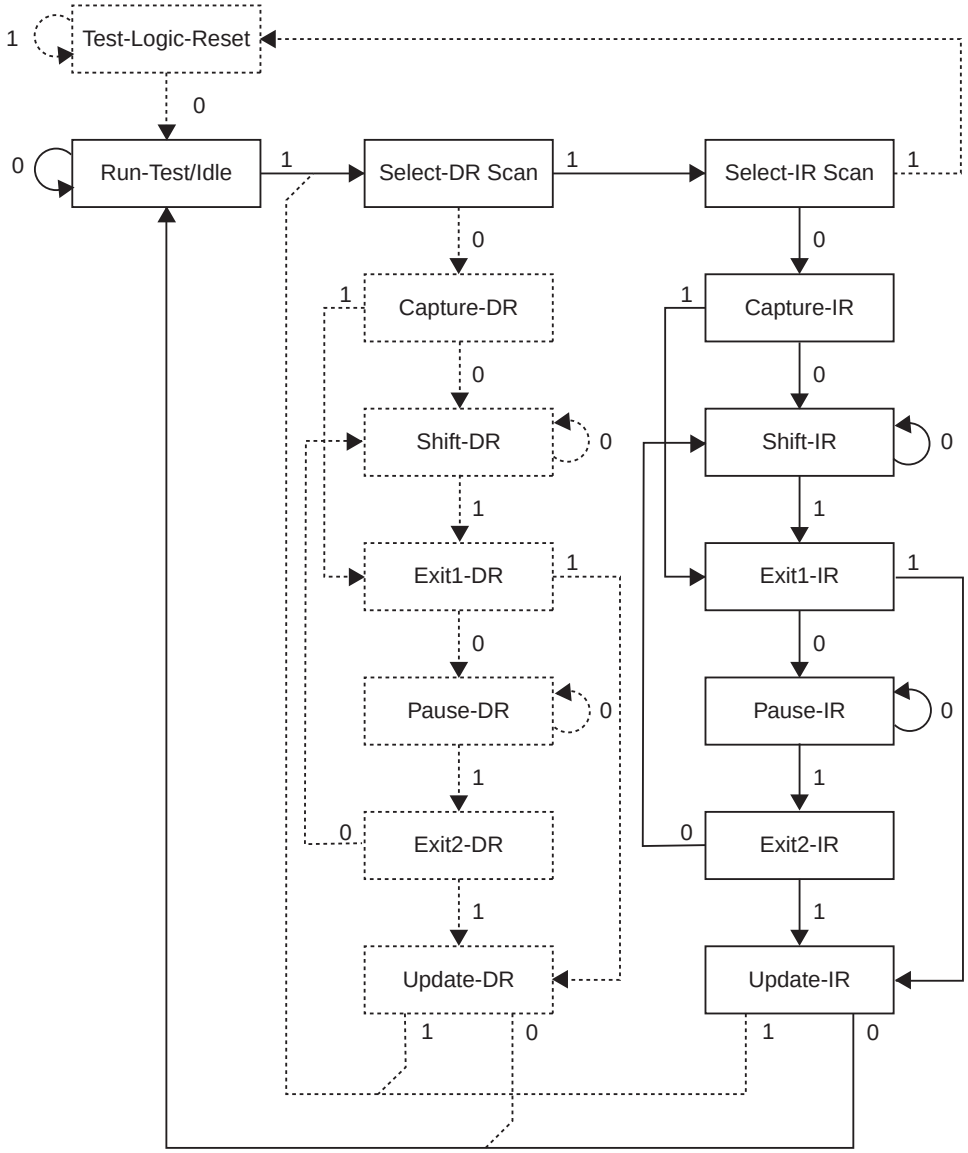
与编程相关的 JTAG 指令

指令寄存器为 4 位，可支持 16 种指令。用于编程的 JTAG 指令在下面列出。

每一条指令的执行代码 OPCODE 都在指令名后以 16 进制形式列出。文字则描述了每条指令中 TDI 和 TDO 之间以哪个被数据寄存器作为数据通路。

TAP控制器的Run-Test/Idle状态用来产生内部时钟。它也可用作JTAG序列之间的空闲状态。改变指令字的状态机序列见 Figure 138。

Figure 138. 改变指令字的状态机序列



AVR_RESET ($C)

AVR_RESET为AVR特有的JTAG指令，用于使AVR进入复位模式或退出复位模式。这条指令不能进行 TAP 的重置。字长只有 1 位的复位寄存器被用作数据寄存器。一旦复位链中有一个逻辑 1，复位状态立刻被激活。这条链的输出不锁存。

活动状态是：

- Shift-DR：复位寄存器通过输入的 TCK 时钟进行移位。

PROG_ENABLE ($4)

PROG_ENABLE是AVR专用的JTAG指令，用来使能JTAG编程。16位的编程使能寄存器被选作数据寄存器。活动状态为：

- Shift-DR：编程使能标志被移入数据寄存器。
- Update-DR：编程使能标志与正确的数值进行比较，如果标志有效即进入编程模式。

PROG_COMMANDS ($5)

AVR 专用的 JTAG 指令。用于通过 JTAG 接口输入编程命令。15 位的编程命令寄存器被用作数据寄存器。活动状态有：

- Capture-DR：上一条指令的结果加载到数据寄存器。
- Shift-DR : 数据寄存器的内容通过输入的 TCK 时钟进行移位，将上一条指令的结果移出数据寄存器，并移入新的命令。
- Update-DR：编程命令已应用到 Flash 输入。
- Run-Test/Idle：产生一个时钟来执行加载的命令 (不总是需要，见 Table 117)。

PROG_PAGELOAD ($6)

AVR专用的JTAG指令。其功能是通过JTAG接口直接将数据加载到Flash数据页。1024位的数据字节寄存器被用作数据寄存器。这是长度等于一页 Flash 的虚拟扫描链。内部移位寄存器为 8 位。与大多数 JTAG 指令不同， Update-DR 状态不是用来从移位寄存器中传送数据。在 Shift-DR 状态中，通过内部状态工具，数据自动以字节为单位传入 Flash 页缓冲器，活动状态为：

- Shift-DR : Flash 数据字节寄存器的内容被通过输入的 TCK 时钟从 TDI 中移入 , 每次载入 1 字节。

Note: JTAG 的PROG_PAGELOAD 指令只有当 AVR器件是JTAG扫描链的第一个器件时才能使用。如果器件不满足这一条件，则必须使用字节法则编程算法。

PROG_PAGEREAD ($7)

AVR专用的JTAG指令。其功能是通过JTAG接口读取Flash的内容。1032位虚拟 Flash 页读寄存器作为数据寄存器。这是长度等于一页 Flash 加 8 的虚拟扫描链。内部移位寄存器为 8 位。与大多数 JTAG 指令不同，Capture-DR 状态不是用来传送数据到移位寄存器 。在 Shift-DR 状态中，通过内部状态工具，数据自动以字节为单位从 Flash 页缓冲器传出，活动状态为：

- Shift-DR : Flash 数据每次自动读 1 字节 , 且通过 TCK 时钟从 TDO 中移出。TDI 输入忽略。

Note: JTAG 的PROG_PAGEREAD 指令只有当 AVR器件是JTAG扫描链的第一个器件时才能使用。如果器件不满足这一条件，则必须使用字节法则编程算法。

数据寄存器

数据寄存器由 JTAG 指令寄存器选取，如 P256 “ 与编程相关的 JTAG 指令 ” 所述。与编程操作相关的数据寄存器为：

- 复位寄存器
- 编程使能寄存器
- 编程命令寄存器
- Flash 页载入寄存器
- Flash 页读出寄存器

复位寄存器

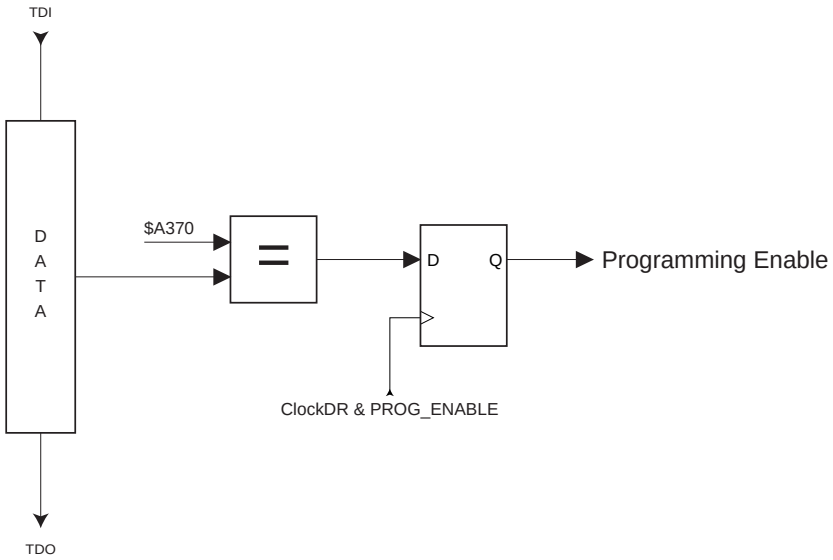
复位寄存器是一个测试数据寄存器，用来在编程期间复位芯片。进入编程模式之前首先要复位器件。

复位寄存器的值大于 0 相当于将外部复位引脚拉低。只要复位寄存器的值大于 0 芯片就处于复位状态。根据时钟项熔丝位的设置，在释放复位寄存器之后，设备仍保持复位状态直到复位定时时间溢出 (见 P23 “ 时钟源 ”)。从数据寄存器输出的数据不锁存，因此复位会立即发生，如 P210 Figure 115 。

编程使能寄存器

编程使能寄存器是一个 16 位的寄存器。这个寄存器的内容将与编程使能信号 (二进制 1010_0011_0111_0000) 进行比较。如果寄存器的内容与编程使能信号相同，就可以通过 JTAG 进行编程。此寄存器在上电复位时复位，并且在退出编程模式时也应将它复位。

Figure 139. 编程使能寄存器



编程命令寄存器

编程命令寄存器是一个 15 位的寄存器。这个寄存器用来串行地移入编程命令，串行地移出上一个命令的执行结果。 JTAG 编程指令集见 Table 117。编程命令移入的状态序列请参见 Figure 141。

Figure 140. 编程命令寄存器

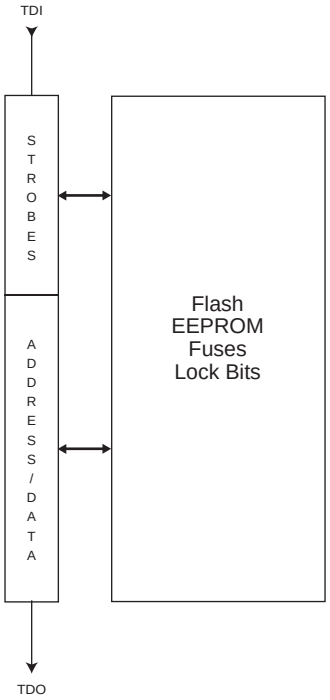


Table 117. JTAG 编程指令集

a = 高位地址 , **b** = 低位地址 , **H** = 0 - 高字节 , 1 - 低字节 , **o** = 数据输出 , **i** = 数据输入 , x = 无意义

指令	TDI 序列	TDO 序列	注意
1a. 芯片擦除	0100011_10000000 0110001_10000000 0110011_10000000 0110011_10000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	
1b. 芯片擦除结束检测	0110011_10000000	xxxxx**o**x_xxxxxxxx	(2)
2a. 进入 Flash 写操作	0100011_00010000	xxxxxxx_xxxxxxxx	
2b. 加载高位地址	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)
2c. 加载低位地址	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
2d. 加载数据低字节	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	
2e. 加载数据高字节	0010111_**iiiiiiii**	xxxxxxx_xxxxxxxx	
2f. 锁存数据	0110111_00000000 1110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
2g. 写 Flash 页	0110111_00000000 0110101_00000000 0110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
2h. 检测页写是否结束	0110111_00000000	xxxxx**o**x_xxxxxxxx	(2)
3a. 进入 Flash 读操作	0100011_00000010	xxxxxxx_xxxxxxxx	
3b. 加载高位地址	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)
3c. 装载低位地址	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
3d. 读数据低、高字节	0110010_00000000 0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo** xxxxxxx_**oooooooo**	 低字节 高字节
4a. 进入 EEPROM 写操作	0100011_00010001	xxxxxxx_xxxxxxxx	
4b. 加载高位地址	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)
4c. 加载低位地址	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
4d. 加载数据字节	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	
4e. 数据锁存	0110111_00000000 1110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
4f. 写 EEPROM 页	0110011_00000000 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
4g. 检测页写是否结束	0110011_00000000	xxxxx**o**x_xxxxxxxx	(2)
5a. 进入 EEPROM 读操作	0100011_00000011	xxxxxxx_xxxxxxxx	
5b. 加载高位地址	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)

Table 117. JTAG 编程指令集 (Continued)

a = 高位地址 , **b** = 低位地址 , **H** = 0 - 高字节 , 1 - 低字节 , **o** = 数据输出 , **i** = 数据输入 , x = 无意义

指令	TDI 序列	TDO 序列	注意
5c. 加载低位地址	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
5d. 读数据字节	0110011_**bbbbbbbb** 0110010_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
6a. 进入写熔丝位操作	0100011_01000000	xxxxxxx_xxxxxxxx	
6b. 加载数据低字节 (6)	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	(3)
6c. 写扩展熔丝位字节	0110111_00000000 0110101_00000000 0110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
6d. 检测写熔丝位是否结束	0110111_00000000	xxxxx**o**x_xxxxxxxx	(2)
6e. 加载数据低字节 (7)	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	(3)
6f. 写熔丝位高字节	0110011_00000000 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
6g. 检测写熔丝位是否结束	0110011_00000000	xxxxx**o**x_xxxxxxxx	(2)
7a. 进入写锁定位操作	0100011_00100000	xxxxxxx_xxxxxxxx	
7b. 加载数据字节 (9)	0010011_11**iiiiii**	xxxxxxx_xxxxxxxx	(4)
7c. 写锁定位	0110011_00000000 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
7d. 检测写锁定位是否结束	0110011_00000000	xxxxx**o**x_xxxxxxxx	(2)
8a. 进入熔丝位 / 锁定位读操作	0100011_00000100	xxxxxxx_xxxxxxxx	
8b. 读熔丝位高字节 (6)	0111110_00000000 0111111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
8c. 读熔丝位低字节 (7)	0110010_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
8d. 读锁定位 (8)	0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xx**oooooo**	(5)
8e. 读熔丝位及锁定位	0111110_00000000 0110010_00000000 0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo** xxxxxxx_**oooooooo** xxxxxxx_**oooooooo**	(5) 熔丝位高字节 熔丝位低字节 锁定位
9a. 进入标识字节读操作	0100011_00001000	xxxxxxx_xxxxxxxx	
9b. 加载地址字节	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
9c. 读标识字节	0110010_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	

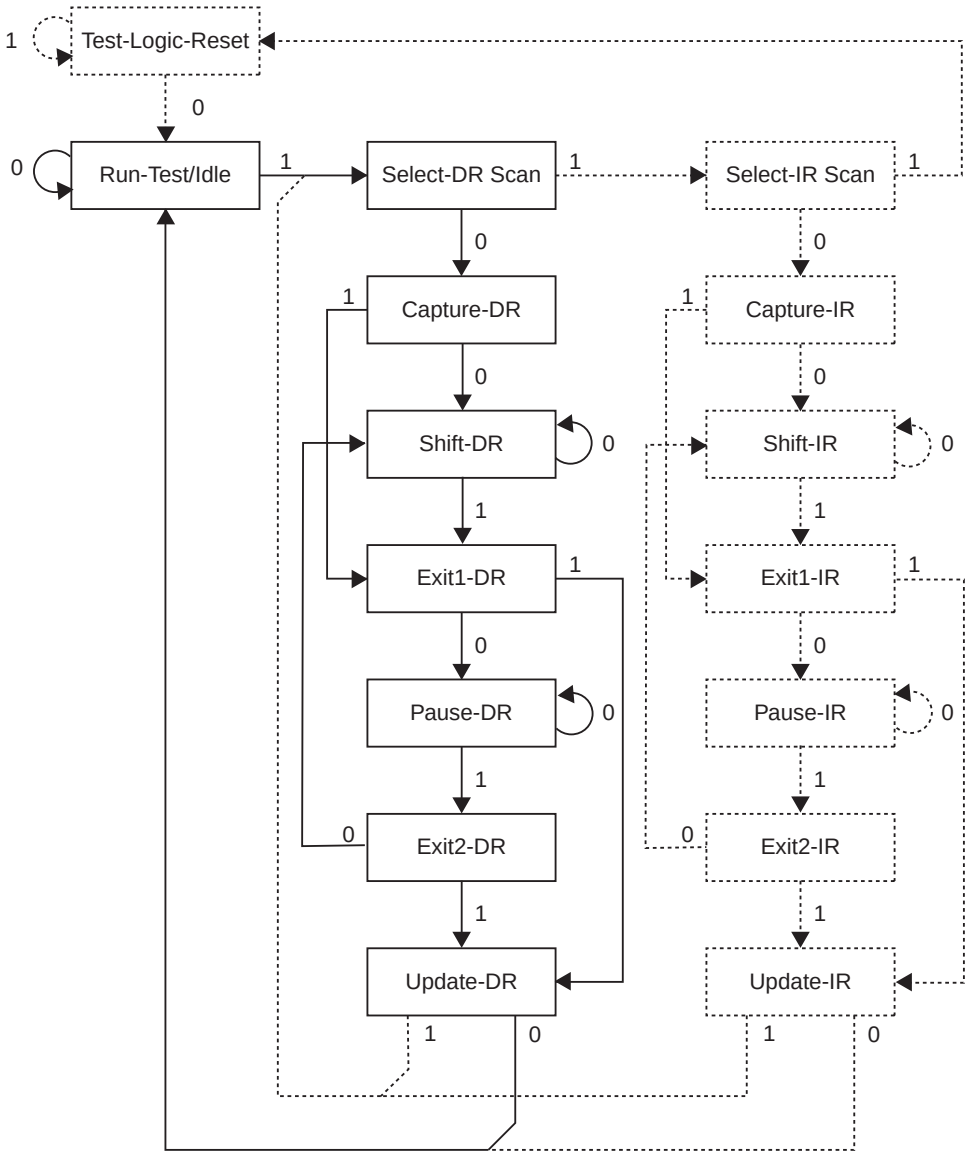
Table 117. JTAG 编程指令集 (Continued)

a = 高位地址 , **b** = 低位地址 , **H** = 0 - 高字节 , 1 - 低字节 , **o** = 数据输出 , **i** = 数据输入 , x = 无意义

指令	TDI 序列	TDO 序列	注意
10a. 进入校验字节读操作	0100011_00001000	xxxxxxx_xxxxxxxx	
10b. 加载地址字节	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
10c. 读校验字节	0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
11a. 加载无操作命令	0100011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	

Notes:
1. 如果在前一个命令序列 (正常情况) 中已正确设置了 7 个 MSB，那么就不需要这个命令序列。
2. 重复到 **o** = “1”.
3. 相应的熔丝位 “0” = 已编程 “1” = 未编程
4. 相应的锁定位 “0” = 已 编程 “1” = 未编程
5. “0” = 已编程 “1” = 未编程
6. 对应的熔丝位高字节的位映射列于 P240 Table 105
7. 对应的熔丝位低字节的位映射列于 P240 Table 106
8. 对应的锁定位字节的位映射列于 P239 Table 103
9. 忽略超过 PCMSB 及 EEAMSB(Table 111 及 Table 112) 的地址

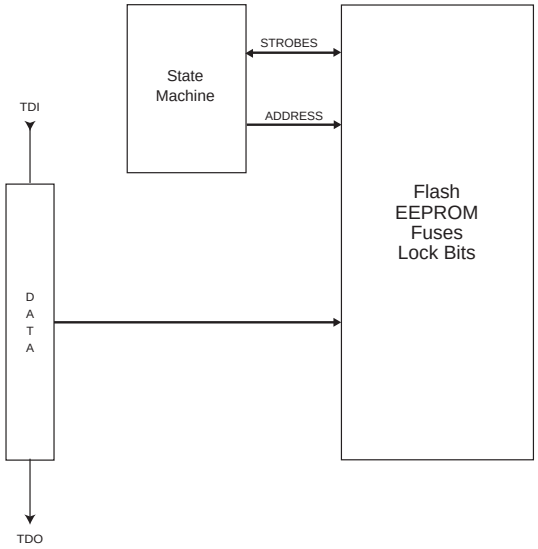
Figure 141. 改变 / 读取数据字的状态机序列



虚拟 Flash 页面加载寄存器

虚拟 Flash 页加载寄存器是长度等于一页 Flash 的虚拟扫描链。内部移位寄存器为 8 位，数据自动以字节为单位传入 Flash 页缓冲器。在页中移入所有指令字，由页内第一条指令的 LSB 开始，到最后一条指令的 MSB 结束。这为在执行页写操作前加载整个 Flash 页缓冲器提供了一条有效的措施。

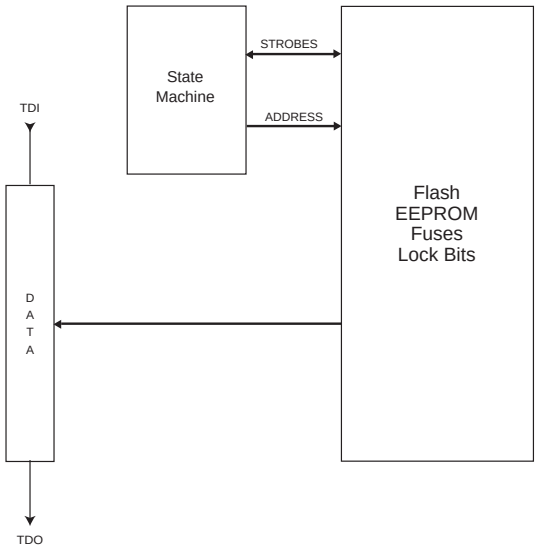
Figure 142. 虚拟 Flash 页加载寄存器



虚拟 Flash 页面读取寄存器

虚拟 Flash 页读取寄存器是长度等于一页 Flash 加 8 的虚拟扫描链。内部移位寄存器为 8 位，数据自动以字节为单位从 Flash 数据页传出。第一个 8 周期用来传送内部移位寄存器的第一个字节，且该字节应被忽略。在这一初始化后，数据移出由页内第一条指令的 LSB 开始，到最后一条指令的 MSB 结束。这为读取整个 Flash 页校验编程提供了一条有效的措施。

Figure 143. 虚拟 Flash 页读取寄存器



编程算法

所有类似 “1a”，“1b” 这样的参考请参见 Table 117。

进入编程模式

1. 进入 JTAG 指令 AVR_RESET，并把 1 移入 Reset 寄存器。
2. 进入 PROG_ENABLE 指令，并把 1010_0011_0111_0000 送入编程使能寄存器。

退出编程模式

1. 进入 JTAG 指令 PROG_COMMANDS
2. 通过无操作指令 11a 来禁止所有编程指令
3. 进入 PROG_ENABLE 指令，并将 0000_0000_0000_0000 送入编程使能寄存器
4. 进入 JTAG 指令 AVR_RESET，并把 0 送入 Reset 寄存器

执行芯片擦除操作

1. 进入 JTAG 指令 PROG_COMMANDS
2. 使用编程指令 1a 进行芯片擦除
3. 使用 1b 指令检查芯片擦除是否完成，或者等待 t_{WLRH_CE}(见 P250 Table 113)。

对 Flash 进行编程

在对 Flash 进行编程前，必须先执行芯片擦除，见 P266 “ 执行芯片擦除操作 ” 。

1. 进入 JTAG 指令 PROG_COMMANDS
2. 使用编程指令 2a 启动 Flash 写操作
3. 使用编程指令 2b 加载地址高字节
4. 使用编程指令 2c 加载地址低字节
5. 使用编程指令 2d、2e 及 2f 加载数据
6. 对这一页的所有程序字重复操作 4 和操作 5
7. 使用编程指令 2g 进行页写操作
8. 使用编程指令 2h 检查 Flash 编程是否结束，或等待 t_{WLRH}(见 P250 Table 113)
9. 重复操作 3 到 7，直到所有的数据都被编程

可以通过使用 PROG_PAGELOAD 指令来得到更高的数据传输率：

1. 进入 JTAG 指令 PROG_COMMANDS
2. 通过使用编程指令 2a 启动 Flash 写操作
3. 使用编程指令 2b 及 2c 加载页地址。PCWORD(见 P243 Table 111) 用于页内寻址，它必须为 0。
4. 进入 JTAG 指令 PROG_PAGELOAD
5. 一个字节一个字节地将页内的程序字加载到整个页，由第一条指令的 LSB 开始，结束于最后一条指令的 MSB。
6. 进入 JTAG 指令 PROG_COMMANDS
7. 使用编程指令 2g 执行页写操作
8. 使用编程指令 2h 检查 Flash 写操作是否完成，或等待 t_{WLRH} (见 P250 Table 113).
9. 重复步骤 3 到 8 ，直到所有的数据都被编程

读取 Flash

1. 进入 JTAG 指令 PROG_COMMANDS
2. 使用编程指令 3a 启动 Flash 读操作
3. 使用编程指令 3b 及 3c 加载地址
4. 使用编程指令 3d 读数据
5. 重复操作 3 和 4，直到所有数据都被读出

通过使用 PROG_PAGEREAD 指令可以更高效地传输数据：

1. 进入 JTAG 指令 PROG_COMMANDS
2. 使用编程指令 3a 启动 Flash 读操作
3. 使用编程指令3b及3c来加载页地址。PCWORD (见 P243 Table 111)用于页内寻址，必须为 0。
4. 进入 JTAG 指令 PROG_PAGEREAD
5. 通过将一页之中的所有程序字移出来读取一整页，由第一条指令的 LSB 开始，由终止于最后一条指令的 MSB。第一个移出的字节要忽略。
6. 使用 JTAG 指令 PROG_COMMANDS
7. 重复步骤 3 到 6，直到所有数据都被读出

对 EEPROM 进行编程

在对 EEPROM 进行编程前，必须先执行芯片擦除，见 P266 “ 执行芯片擦除操作 ” 。

1. 进入 JTAG 指令 PROG_COMMANDS
2. 使用编程指令 4a 启动 EEPROM 写操作
3. 使用编程指令 4b 来加载地址高字节
4. 使用编程指令 4c 来加载地址低字节
5. 使用 4d 及 4e 加载数据
6. 对页内所有数据字节执行操作 4 和 5
7. 使用编程指令 4f 写数据
8. 使用编程指令4g检查EEPROM写操作是否已经完成，或等待t_{WLRH}(见P250 Table 113)。
9. 重复步骤 3 到 8，直到所有的数据都被编程

编程 EEPROM 时不能使用 PROG_PAGELOAD 指令。

读取 EEPROM

1. 进入 JTAG 指令 PROG_COMMANDS
2. 使用编程指令 5a 启动 EEPROM 读操作
3. 使用编程指令 5b 及 5c 加载地址
4. 使用编程指令 5d 读数据
5. 重复操作 3 和 4，直到所有数据都被读出

注意，读 EEPROM 时不能使用 PROG_PAGEREAD 指令。

对熔丝位进行编程

1. 进入 JTAG 指令 PROG_COMMANDS
2. 使用编程指令 6a 启动熔丝位写操作
3. 使用编程指令 6b 加载数据高字节。位值为 0 表示相应熔丝位需要编程，否则不编程。
4. 使用编程指令 6c 写熔丝位高字节
5. 使用编程指令 6d 检查熔丝位写操作是否完成，或等待 t_{WLRH}（见 P250 Table 113）
6. 使用编程指令 6e 加载数据字节，写 0 表示熔丝位编程，写 1 表示未编程。
7. 使用编程指令 6f 写熔丝位高字节
8. 使用编程指令 6g 检查熔丝位写操作是否完成，或等待 t_{WLRH}（见 P250 Table 113）

对锁定位进行编程

1. 进入 JTAG 指令 PROG_COMMANDS
2. 使用编程指令 7a 进入锁定位写操作
3. 使用编程指令 7b 进行数据加载。位值为 0 表示相应熔丝位需要编程，否则不编程。
4. 使用编程指令 7c 写锁定位
5. 使用编程指令 7d 检验锁定位写操作是否完成，或等待 t_{WLRH}（见 P250 Table 113）

读取熔丝位和锁定位

1. 进入 JTAG 指令 PROG_COMMANDS
2. 使用编程指令 8a 进入熔丝位 / 锁定位读操作
3. 使用编程指令 8e 来读取所有熔丝位及锁定位
   使用编程指令 8b 来读取熔丝位高字节
   使用编程指令 8c 来读取熔丝位低字节
   使用编程指令 8d 读取锁定位

读取标识字节

1. 进入 JTAG 指令 PROG_COMMANDS
2. 使用指令 9a 启动标识字节读操作
3. 使用编程指令 9b 加载地址 0x00
4. 使用编程指令 9c 读第一个标识字节
5. 在地址 0x01 及 0x02 处重复操作 3 和 4，分别读第二及第三个标识字节

读取标定字节

1. 进入 JTAG 指令 PROG_COMMANDS
2. 使用 10a 指令启动检验字节读操作
3. 使用编程指令 10b 加载地址 0x00
4. 使用编程指令 10c 读取校验字节

电气特性

绝对极限值 *

工作温度	-55°C ~ +125°C
存储温度	-65°C ~ +150°C
各个引脚对地的电压，除了 $\overline{\text{RESET}}$	-0.5V ~ V_{CC}+0.5V
$\overline{\text{RESET}}$ 引脚对地的电压	-0.5V ~ +13.0V
最大工作电压	6.0V
每个 I/O 引脚上的直流电流	40.0 mA
V_{CC} 与 GND 引脚上的直流电流	200.0 mA

*NOTICE: 如果强制芯片在超出“ 绝对极限值” 表中所列的条件之下工作可能造成器件的永久损坏。这仅是工作应力的极限。并不表示器件可以工作于表中所列条件之下，或是那些超越工作范围明确规定的其他条件之下。长时间工作于绝对极限值可能会影响器件的寿命。

直流特性

T_A = -40°C ~ 85°C, V_{CC} = 2.7V ~ 5.5V (除非另外说明)

符号	参数	条件	最小值	典型值	最大值	单位
V_{IL}	输入低电压	除 XTAL1 引脚	-0.5		0.2 V_{CC}(1)	V
V_{IL1}	输入低电压	XTAL1 引脚，外部时钟	-0.5		0.1 V_{CC}(1)	V
V_{IH}	输入高电压	除了 XTAL1 和 $\overline{\text{RESET}}$ 引脚	0.6 V_{CC}(2)		V_{CC} + 0.5	V
V_{IH1}	输入高电压	XTAL1 引脚，外部时钟	0.7 V_{CC}(2)		V_{CC} + 0.5	V
V_{IH2}	输入高电压	$\overline{\text{RESET}}$ 引脚	0.9 V_{CC}(2)		V_{CC} + 0.5	V
V_{OL}	输出低电压 (3) (端口 A,B,C,D)	I_{OL} = 20 mA, V_{CC} = 5V I_{OL} = 10 mA, V_{CC} = 3V			0.7 0.5	V V
V_{OH}	输出高电压 (4) (端口 A,B,C,D)	I_{OH} = -20 mA, V_{CC} = 5V I_{OH} = -10 mA, V_{CC} = 3V	4.0 2.2			V V
I_{IL}	输入漏电流 I/O 引脚	V_{CC} = 5.5V, 引脚为低电平 (绝对值)			1	µA
I_{IH}	输入漏电流 I/O 引脚	V_{CC} = 5.5V, 引脚为高电平 (绝对值)			1	µA
R_{RST}	Reset 引脚上拉电阻		30		60	kΩ
R_{pu}	I/O 引脚上拉电阻		20		50	kΩ

T_A = -40°C ~ 85°C, V_{CC} = 2.7V ~ 5.5V (除非另外说明)

符号	参数	条件	最小值	典型值	最大值	单位
I_{CC}	工作电流	正常 1 MHz, V_{CC} = 3V (ATmega32L)		1.1		mA
		正常 4 MHz, V_{CC} = 3V (ATmega32L)		3.8	5	mA
		正常 8 MHz, V_{CC} = 5V (ATmega32)		12	15	mA
		空闲 1 MHz, V_{CC} = 3V (ATmega32L)		0.35		mA
		空闲 4 MHz, V_{CC} = 3V (ATmega32L)		1.2	2.5	mA
		空闲 8 MHz, V_{CC} = 5V (ATmega32)		5.5	8	mA
	掉电模式 (5)	WDT 使能 , V_{CC} = 3V		< 25	20	µA
		WDT 禁止 , V_{CC} = 3V		< 1	10	µA
V_{ACIO}	模拟比较器 输入偏置电压	V_{CC} = 5V V_{in} = V_{CC}/2			40	mV
I_{ACLK}	模拟比较器 输入泄漏电流	V_{CC} = 5V V_{in} = V_{CC}/2	-50		50	nA
t_{ACID}	模拟比较器 传输延迟	V_{CC} = 2.7V V_{CC} = 4.0V		750 500		ns

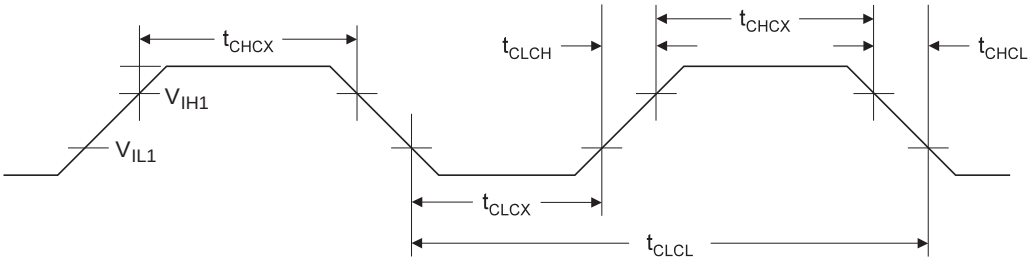
Notes:
1. " 最大值 " 表示保证引脚读取数值为低时的最高值
2. " 最小值 " 表示保证引脚读取数值为高时的最低值
3. 虽然在稳定状态条件(非瞬态)下每个I/O端口都可以吸收比测试条件下更多的电流(20 mA ,V_{CC} = 5V 以及10 mA，V_{CC} = 3V) ，但是需要遵循以下要求
   PDIP 封装：
   1] 所有端口的 IOL 总和不能超 400 mA
   2] 端口 A0 - A7 的 IOL 总和不能超过 200 mA
   3] 端口 B0 - B7,C0 - C7, D0 - D7 与 XTAL2 的 IOL 总和不能超过 300 mA
   TQFP 与 MLF 封装：
   1] 所有端口的 IOL 总和不能超过 400 mA
   2] 端口 A0 - A7 的 IOL 总和不能超过 200 mA
   3] 端口 B0 - B4 的 IOL 总和不能超过 200 mA
   4] 端口 B3 - B7, XTAL2, D0 - D2 的 IOL 总和不能超过 200 mA
   5] 端口 D3 - D7 的 IOL 总和不能超过 200 mA
   6] 端口 C0 - C7 的 IOL 总和不能超过 200 mA
   如果 IOL 超出了测试条件， VOL 可能超过指标。不保证引脚可以吸收比列于此处的测试条件更大的电流。
4. 虽然在稳定状态条件(非瞬态)下每个I/O端口都可以输出比测试条件下更多的电流(20 mA ,V_{CC} = 5V 以及10 mA，V_{CC} = 3V) ，但是需要遵循以下要求：
   PDIP 封装：
   1] 所有端口的 IOH 总和不能超过 400 mA
   2] 端口 A0 - A7 的 IOH 总和不能超过 200 mA
   3] 端口 B0 - B7,C0 - C7, D0 - D7 及 XTAL2 的 IOH 总和不能超过 300 mA
   TQFP 与 MLF 封装：
   1] 所有端口的 IOH 总和不能超过 400 mA
   2] 端口 A0 - A7 的 IOH 总和不能超过 200 mA
   3] 端口 B0 - B4 的 IOH 总和不能超过 200 mA
   4] 端口 B3 - B7, XTAL2, D0 - D2 的 IOH 总和不能超过 200 mA
   5] 端口 D3 - D7 的 IOH 总和不能超过 200 mA

6] 端口 C0 - C7 的 IOH 总和不能超过 200 mA。如果 IOH 超出了测试条件 ,VOH 可能超过指标。不保证引脚可以输出比列于此处的测试条件更大的电流。

5. 掉电模式下的最小 V_{CC} 为 2.5V。

外部时钟驱动波形

Figure 144. 外部时钟驱动波形



外部时钟驱动

Table 118. 外部时钟驱动

符号	参数	V_{CC} = 2.7V - 5.5V		V_{CC} = 4.5V - 5.5V		单位
		最小值	最大值	最小值	最大值	
$1/t_{CLCL}$	振荡器频率	0	8	0	16	MHz
t_{CLCL}	时钟周期	125		62.5		ns
t_{CHCX}	高电平时间	50		25		ns
t_{CLCX}	低电平时间	50		25		ns
t_{CLCH}	上升时间		1.6		0.5	µs
t_{CHCL}	下降时间		1.6		0.5	µs
Δt_{CLCL}	时钟周期的变化		2		2	%

Table 119. 外部 RC 振荡器，典型频率 (V_{CC} = 5V)

R [kΩ](1)	C [pF]	f(2)
100	47	87 kHz
33	22	650 kHz
10	22	2.0 MHz

Notes: 1. R 的取值范围为 3 kΩ - 100 kΩ ,C 至少应该为 20 pF。表中 C 值包括引脚电容 ,C 值随封装形式而变化

2. 频率因封装形式与板层的不同而不同

两线串行接口特性

Table 120 描述了连接到两线串行总线上的器件的要求。 ATmega32 的两线接口满足或超出此处列出的要求。时序符号请参考 Figure 145。

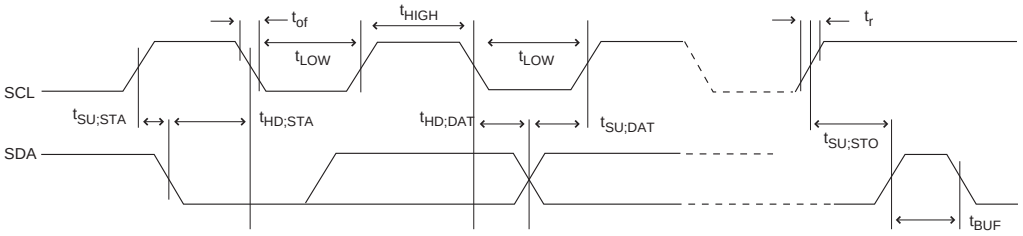
Table 120. 两线串行总线要求

符号	参数	条件	最小值	最大值	单位
V_{IL}	输入低电压		-0.5	0.3 V_{CC}	V
V_{IH}	输入高电压		0.7 V_{CC}	V_{CC} + 0.5	V
V_{hys}(1)	施密特触发器输入的迟滞电压		0.05 V_{CC}(2)	–	V
V_{OL}(1)	输出低电压	3 mA sink current	0	0.4	V
t_r(1)	SDA 和 SCL 的上升时间		20 + 0.1C_b(3)(2)	300	ns
t_{of}(1)	由 V_{IHmin} 到 V_{ILmax} 的输出下降时间	10 pF < C_b < 400 pF(3)	20 + 0.1C_b(3)(2)	250	ns
t_{SP}(1)	输入滤波器抑制的尖峰时间		0	50(2)	ns
I_i	每个 I/O 引脚的输入电流	0.1V_{CC} < V_i < 0.9V_{CC}	-10	10	µA
C_i(1)	每个 I/O 引脚的电容		–	10	pF
f_{SCL}	SCL 时钟频率	f_{CK}(4) > max(16f_{SCL}, 250kHz)(5)	0	400	kHz
Rp	上拉电阻值	f_{SCL} ≤ 100 kHz	$\frac{V_{CC}-0,4V}{3mA}$	$\frac{1000ns}{C_b}$	Ω
		f_{SCL} > 100 kHz	$\frac{V_{CC}-0,4V}{3mA}$	$\frac{300ns}{C_b}$	Ω
$t_{HD;STA}$	START 条件的保持时间 (重复)	f_{SCL} ≤ 100 kHz	4.0	–	µs
		f_{SCL} > 100 kHz	0.6	–	µs
t_{LOW}	SCL 时钟的低电平时间	f_{SCL} ≤ 100 kHz(6)	4.7	–	µs
		f_{SCL} > 100 kHz(7)	1.3	–	µs
t_{HIGH}	SCL 时钟的高电平时间	f_{SCL} ≤ 100 kHz	4.0	–	µs
		f_{SCL} > 100 kHz	0.6	–	µs
$t_{SU;STA}$	重复 STARTS 条件的建立时间	f_{SCL} ≤ 100 kHz	4.7	–	µs
		f_{SCL} > 100 kHz	0.6	–	µs
$t_{HD;DAT}$	数据保持时间	f_{SCL} ≤ 100 kHz	0	3.45	µs
		f_{SCL} > 100 kHz	0	0.9	µs
$t_{SU;DAT}$	数据建立时间	f_{SCL} ≤ 100 kHz	250	–	ns
		f_{SCL} > 100 kHz	100	–	ns
$t_{SU;STO}$	STOP 条件的建立时间	f_{SCL} ≤ 100 kHz	4.0	–	µs
		f_{SCL} > 100 kHz	0.6	–	µs
t_{BUF}	STOP 和 START 之间的总线空闲时间	f_{SCL} ≤ 100 kHz	4.7	–	µs
		f_{SCL} > 100 kHz	1.3	–	µs

Notes:
1. 对于 ATmega32，此参数是特性参数，没有经过 100% 的测试
2. 只有当 f_{SCL} > 100 kHz 时才需要
3. C_b = 总线的一条线的电容
4. f_{CK} = CPU 时钟频率

5. 此要求适用于 ATmega32 所有的两线串行接口的操作。其他连接到两线串行总线的器件只需要满足一般的 f_{SCL} 要求即可。
6. ATmega32 两线串行接口实际产生的低电平时间为 ($1/f_{SCL} - 2/f_{CK}$)。因此为了严格满足 f_{SCL} = 100 kHz 时低电平时间的要求 f_{CK} 必须大于 6 MHz。
7. ATmega32 两线串行接口实际产生的低电平时间为 ($1/f_{SCL} - 2/f_{CK}$)。因此在 f_{CK} = 8 MHz , 且 f_{SCL} > 308 kHz 时低电平时间无法严格满足要求。然而 , ATmega32 可以与其他 ATmega32 以全速 (400 kHz) 进行通讯。若其他器件具有合适的 t_{LOW} 接受裕量也可以做到这一点。

Figure 145. 两线串行总线时序



SPI 时序特性

具体信息请参见 Figure 146 和 Figure 147。

Table 121. SPI 时序参数

	说明	模式	最小值	典型值	最大值	
1	SCK 周期	主机		见 Table 58		
2	SCK 高 / 低电平	主机		占空比 50%		
3	上升 / 下降时间	主机		3.6		
4	建立时间	主机		10		
5	保持时间	主机		10		
6	输出到 SCK	主机		$0.5 \cdot t_{SCK}$		ns
7	SCK 到输出	主机		10		
8	SCK 到输出高电平	主机		10		
9	$\overline{SS}$ 低到输出	从机		15		
10	SCK 周期	从机	$4 \cdot t_{ck}$			
11	SCK 高 / 低电平	从机	$2 \cdot t_{ck}$			
12	上升 / 下降时间	从机			1.6	µs
13	建立时间	从机	10			
14	保持时间	从机	t_{ck}			
15	SCK 到输出	从机		15		
16	SCK 到 $\overline{SS}$ 高	从机	20			ns
17	$\overline{SS}$ 高到三态	从机		10		
18	SS 低到 SCK	从机	$2 \cdot t_{ck}$			

Figure 146. SPI 接口时序要求 (主机模式)

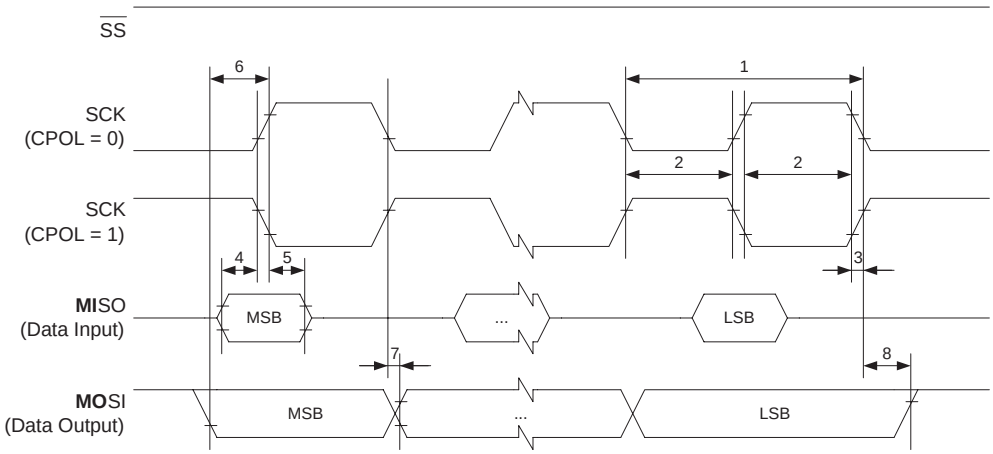
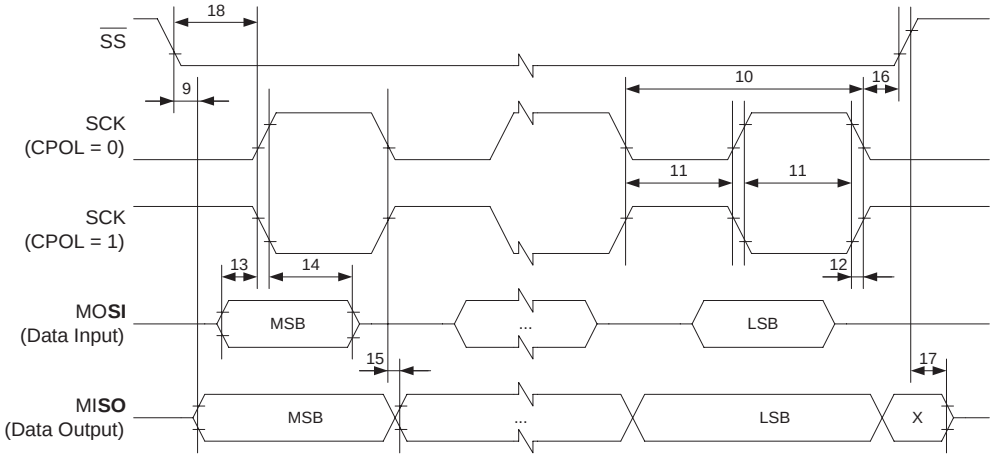


Figure 147. SPI 接口时序要求 (从机模式)



交流特性

Table 122. ADC 特性参数，单端通道 T_A = -40°C ~ 85°C

符号	参数	条件	最小值	典型值	最大值	单位
	分辨率	单极性转换		10		Bits
	绝对精度 (包括 INL, DNL, 量化误差 , Gain, 与偏置误差)	单极性转换 V_{REF} = 4V, V_{CC} = 4V ADC 时钟 = 200 kHz		1.5		LSB
		单极性转换 V_{REF} = 4V, V_{CC} = 4V ADC 时钟 = 1 MHz		3		LSB
		单极性转换 V_{REF} = 4V, V_{CC} = 4V ADC 时钟 = 200 kHz 噪声抑制模式		1.5		LSB
		单极性转换 V_{REF} = 4V, V_{CC} = 4V ADC 时钟 = 1 MHz 噪声抑制模式		3		LSB
	积分非线性	单极性转换 V_{REF} = 4V, V_{CC} = 4V ADC 时钟 = 200 kHz		0.75		LSB
	差分非线性	单极性转换 V_{REF} = 4V, V_{CC} = 4V ADC 时钟 = 200 kHz		0.25		LSB
	增益误差	单极性转换 V_{REF} = 4V, V_{CC} = 4V ADC 时钟 = 200 kHz		0.75		LSB
	偏置误差	单极性转换 V_{REF} = 4V, V_{CC} = 4V ADC 时钟 = 200 kHz		0.75		LSB
	时钟频率		50		1000	kHz
	转换时间		13		260	µs
AVCC	模拟电压		V_{CC} - 0.3(1)		V_{CC} + 0.3(2)	V
V_{REF}	参考电压		2.0		AVCC	V
V_{IN}	输入电压		GND		V_{REF}	V
	ADC 转换输出		0		1023	LSB
	输入带宽			38.5		kHz
V_{INT}	内部电压基准		2.3	2.56	2.7	V
R_{REF}	参考输入端电阻			32		kΩ
R_{AIN}	模拟输入电阻			100		MΩ

Notes: 1. AVCC 的最小值为 2.7V.
2. AVCC 的最大值为 5.5V.

Table 123. ADC 特性参数，差分通道 T_A = -40°C ~ 85°C

符号	参数	条件	最小值	典型值	最大值	单位
	分辨率	Gain = 1x			10	Bits
		Gain = 10x			10	Bits
		Gain = 200x			10	Bits
	绝对精度	Gain = 1x V_{REF} = 4V, V_{CC} = 5V ADC 时钟 = 50 - 200 kHz		17		LSB
		Gain = 10x V_{REF} = 4V, V_{CC} = 5V ADC 时钟 = 50 - 200 kHz		16		LSB
		Gain = 200x V_{REF} = 4V, V_{CC} = 5V ADC 时钟 = 50 - 200 kHz		7		LSB
	INL(在补偿与增益误差标定后的精度)	Gain = 1x V_{REF} = 4V, V_{CC} = 5V ADC 时钟 = 50 - 200 kHz		0.75		LSB
		Gain = 10x V_{REF} = 4V, V_{CC} = 5V ADC 时钟 = 50 - 200 kHz		0.75		LSB
		Gain = 200x V_{REF} = 4V, V_{CC} = 5V ADC 时钟 = 50 - 200 kHz		2		LSB
	增益误差	Gain = 1x		1.6		%
		Gain = 10x		1.5		%
		Gain = 200x		0.2		%
	偏置误差	Gain = 1x V_{REF} = 4V, V_{CC} = 5V ADC 时钟 = 50 - 200 kHz		1		LSB
		Gain = 10x V_{REF} = 4V, V_{CC} = 5V ADC 时钟 = 50 - 200 kHz		1.5		LSB
		Gain = 200x V_{REF} = 4V, V_{CC} = 5V ADC 时钟 = 50 - 200 kHz		4.5		LSB
	时钟频率		50		200	kHz
	转换时间		65		260	µs
AVCC	模拟电压		V_{CC} - 0.3(1)		V_{CC} + 0.3(2)	V
V_{REF}	参考电压		2.0		AVCC - 0.5	V
V_{IN}	输入电压		GND		V_{CC}	V
V_{DIFF}	输入差分电压		$-V_{REF}$/Gain		V_{REF}/Gain/	V
	ADC 转换输出		-511		511	LSB
	输入带宽			4		kHz

Table 123. ADC 特性参数，差分通道 T_A = -40°C ~ 85°C (Continued)

符号	参数	条件	最小值	典型值	最大值	单位
V_{INT}	内部电压基准		2.3	2.56	2.7	V
R_{REF}	参考输入端电阻			32		kΩ
R_{AIN}	模拟输入电阻			100		MΩ

Notes: 1. AVCC 的最小值为 2.7V.
2. AVCC 的最大值为 5.5V.

5.

ATmega32 典型特性

下面图表给出了典型数据。这些数据在产生过程没有进行测试。所有的电流测量数据都是在所有的 I/O 引脚配置为输入且内部上拉电阻使能的条件下测得的。时钟源为外部正弦波发生器产生的满幅值正弦波。

掉电模式下的电流与时钟无关。

电流与多个因素有关，如：工作电压、工作频率、 I/O 引脚的负载及电平转换频率、执行的代码和环境温度。主要因素为工作电压和工作频率。

容性负载引脚的电流可以通过公式 C_L*V_{CC}*f 进行估计。式中，C_L 为负载电容，V_{CC} 为工作电压，f 为引脚的平均开关频率。

器件的特性参数为比测试上限更高的频率下的数据。但是不保证器件在比定货信息标明的工作频率更高的频率功能正常工作。

掉电模式下看门狗使能与看门狗禁止之间的电流差异即是看门狗定时器所需的工作电流。

Figure 148. RC 振荡器频率与温度的关系 (Vcc = 5V，T=25c，芯片频率 1 MHz)

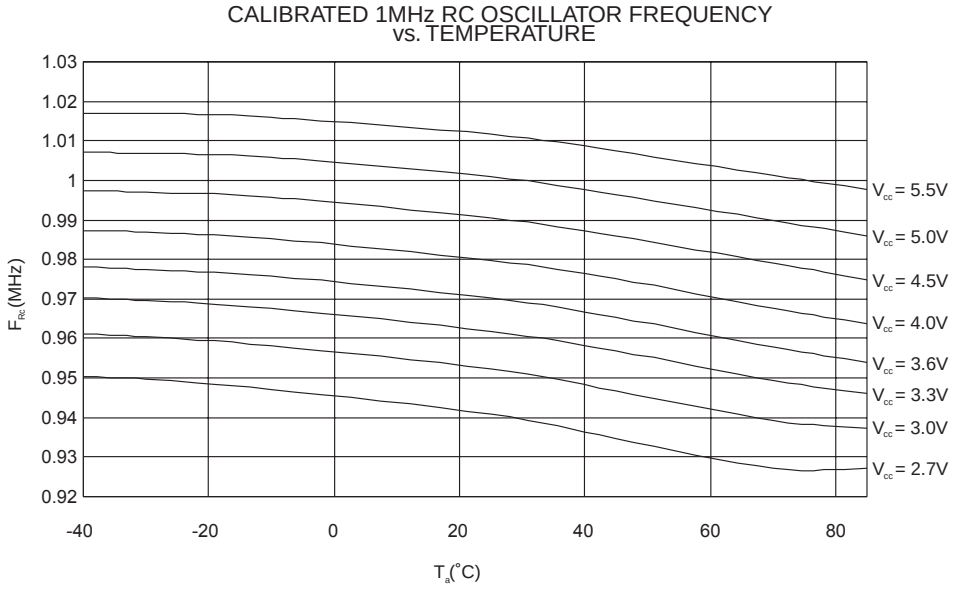


Figure 149. RC 振荡器频率与工作电压的关系 (Vcc = 5V，T=25c，芯片频率 1 MHz)

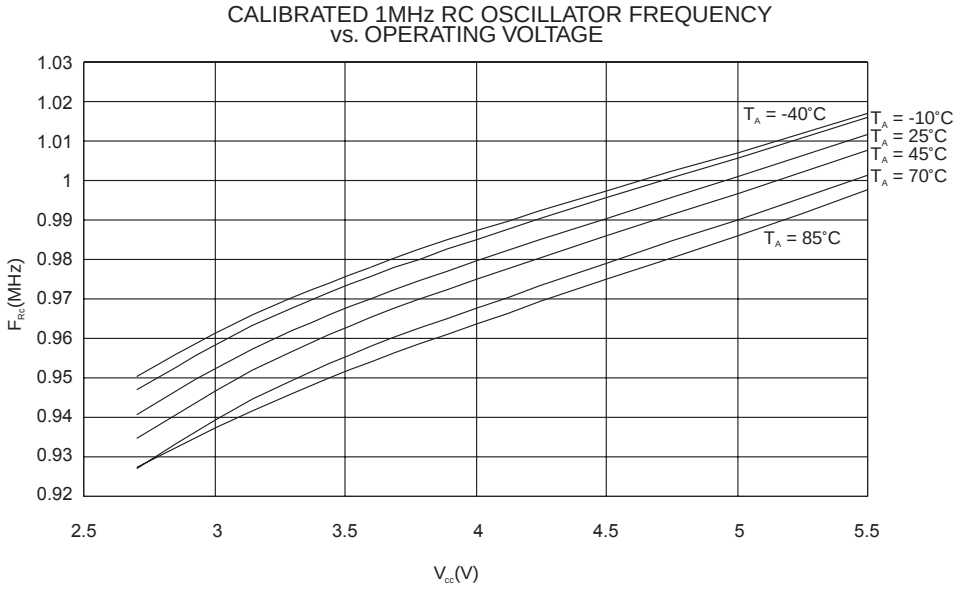


Figure 150. RC 振荡器频率与温度的关系 (Vcc = 5V，T=25c，芯片频率 2 MHz)

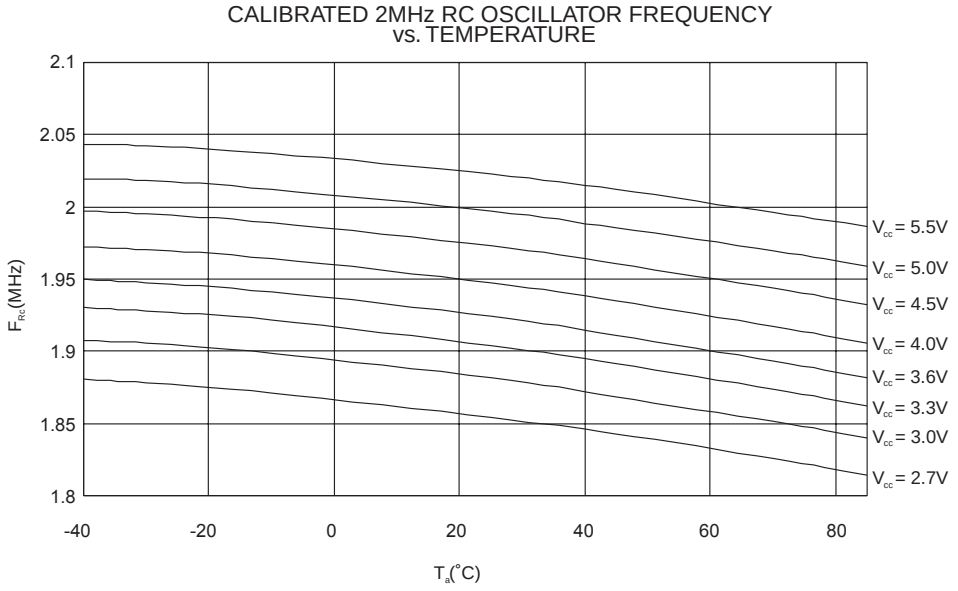


Figure 151. RC 振荡器频率与工作电压的关系 (Vcc = 5V，T=25c，芯片频率 2 MHz)

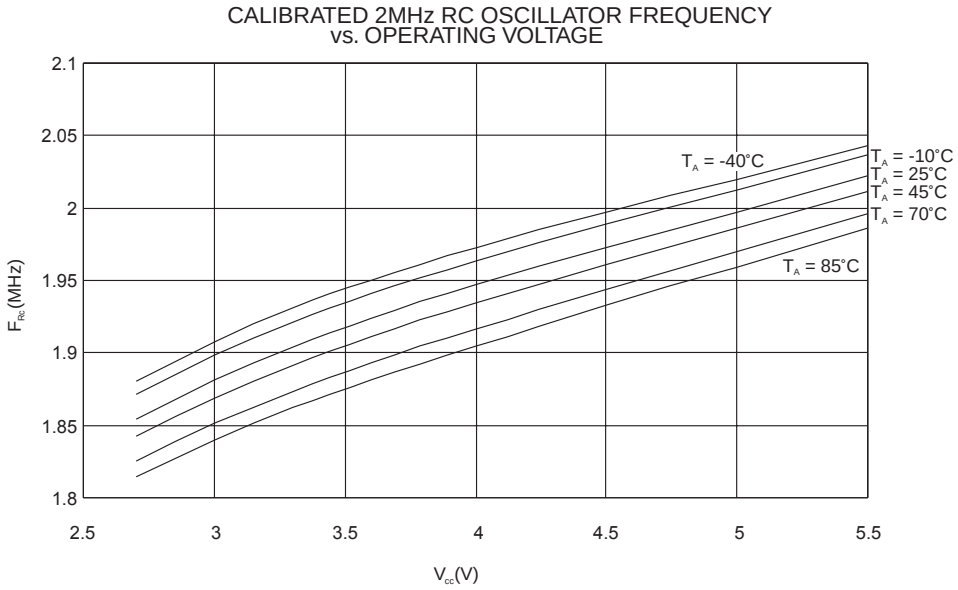


Figure 152. RC 振荡器频率与温度的关系 (Vcc = 5V，T=25c，芯片频率 4 MHz)

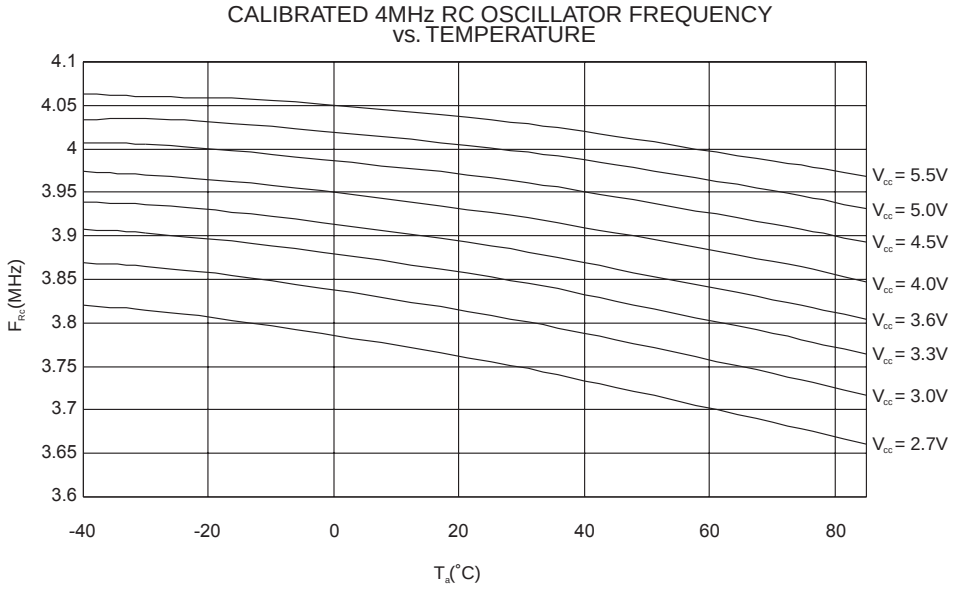


Figure 153. RC 振荡器频率与振荡频率的关系 (Vcc = 5V，T=25c，芯片频率 4 MHz)

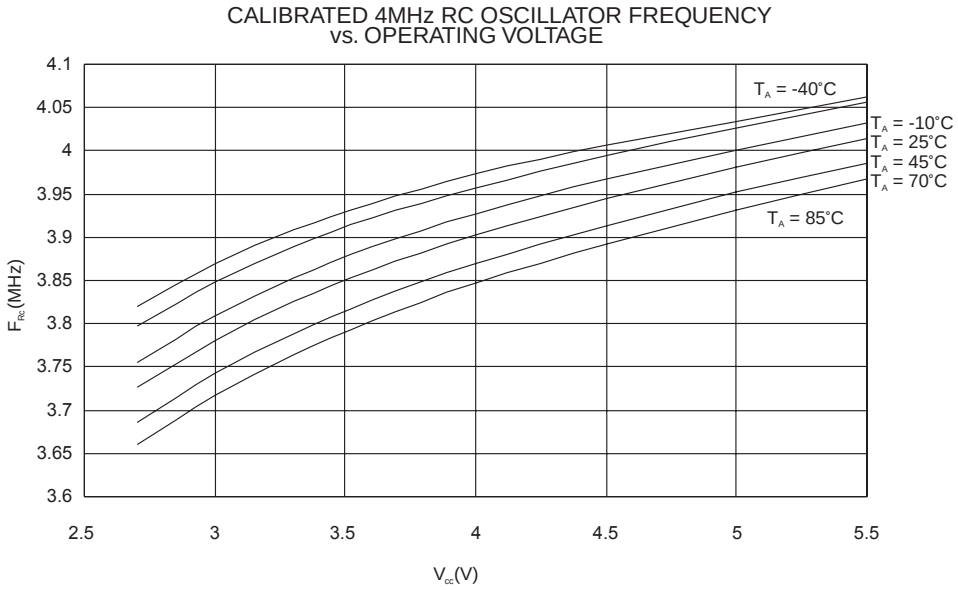


Figure 154. RC 振荡器频率与温度的关系 (Vcc = 5V，T=25c，芯片频率 8 MHz)

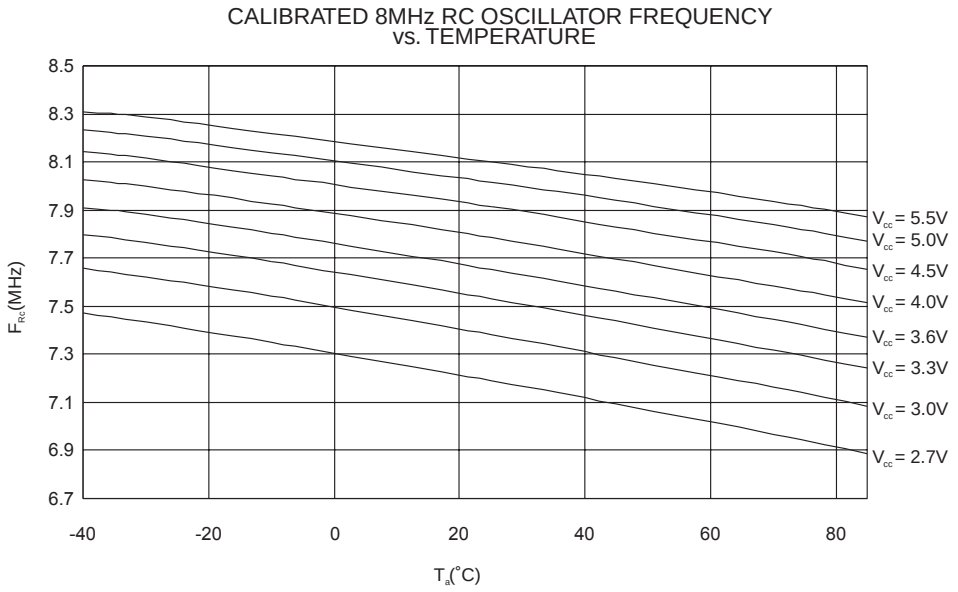
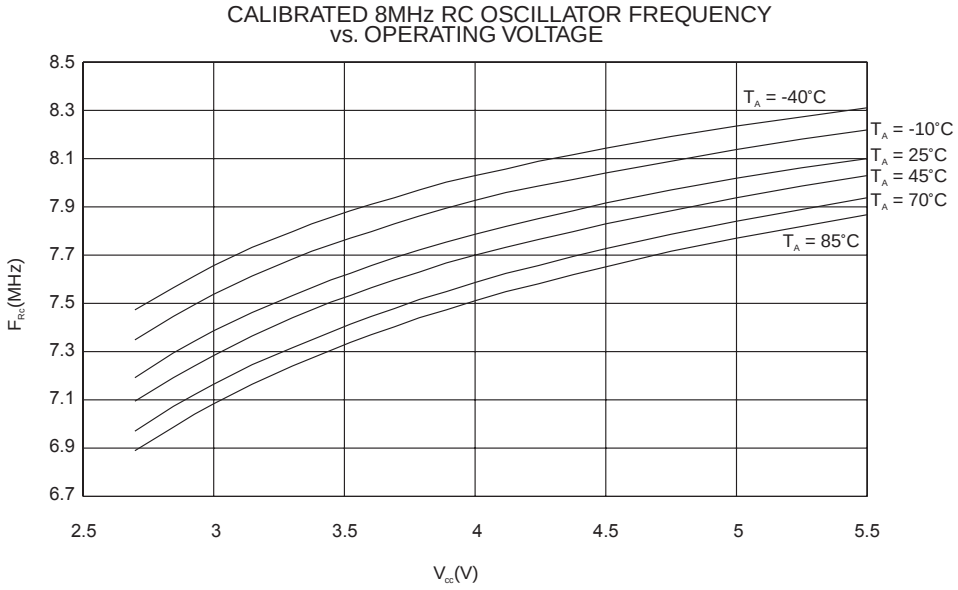


Figure 155. RC 振荡器频率与振荡频率的关系 (Vcc = 5V，T=25c，芯片频率 8 MHz)



寄存器概述

地址	名称	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	页码
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C	8
$3E ($5E)	SPH	–	–	–	–	SP11	SP10	SP9	SP8	10
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	10
$3C ($5C)	OCR0	T/C0 比较输出寄存器								78
$3B ($5B)	GICR	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	45, 65
$3A ($5A)	GIFR	INTF1	INTF0	INTF2	–	–	–	–	–	66
$39 ($59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	78, 104, 120
$38 ($58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	78, 105, 120
$37 ($57)	SPMCR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	232
$36 ($56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	164
$35 ($55)	MCUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	30, 64
$34 ($54)	MCUCSR	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	38, 65, 213
$33 ($53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	76
$32 ($52)	TCNT0	T/C0 (8 位)								77
$31[1] ($51)[1]	OSCCAL	振荡器校准寄存器								28
	OCDR	片上调试寄存器								209
$30 ($50)	SFIOR	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	54,80,121,184,202
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	100
$2E ($4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	103
$2D ($4D)	TCNT1H	T/C1 – 计数器寄存器高字节								103
$2C ($4C)	TCNT1L	T/C1 – 计数器寄存器低字节								103
$2B ($4B)	OCR1AH	T/C1 – 输出比较寄存器 A 高字节								104
$2A ($4A)	OCR1AL	T/C1 – 输出比较寄存器 A 低字节								104
$29 ($49)	OCR1BH	T/C1 – 输出比较寄存器 B 高字节								104
$28 ($48)	OCR1BL	T/C1 – 输出比较寄存器 B 低字节								104
$27 ($47)	ICR1H	T/C1 – 输入捕获寄存器高字节								104
$26 ($46)	ICR1L	T/C1 – 输入捕获寄存器低字节								104
$25 ($45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	116
$24 ($44)	TCNT2	T/C2 (8 位)								117
$23 ($43)	OCR2	T/C2 输出比较寄存器								118
$22 ($42)	ASSR	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB	119
$21 ($41)	WDTCR	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	39
$20[2] ($40)[2]	UBRRH	URSEL	–	–	–	UBRR[11:8]				152
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	149
$1F ($3F)	EEARH	–	–	–	–	–	–	EEAR9	EEAR8	17
$1E ($3E)	EEARL	EEPROM 地址寄存器低字节								17
$1D ($3D)	EEDR	EEPROM 数据寄存器								17
$1C ($3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE	17
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	61
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	61
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	62
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	62
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	62
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	62
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	62
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	62
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	62
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	62
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	62
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	63
$0F ($2F)	SPDR	SPI 数据寄存器								128
$0E ($2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	128
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	126
$0C ($2C)	UDR	USART I/O 数据寄存器								147
$0B ($2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	148
$0A ($2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	148
$09 ($29)	UBRRL	USART 波特率寄存器低字节								152
$08 ($28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	185
$07 ($27)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	199
$06 ($26)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	201
$05 ($25)	ADCH	ADC 数据寄存器高字节								202
$04 ($24)	ADCL	ADC 数据寄存器低字节								202
$03 ($23)	TWDR	两线串行接口数据寄存器								165
$02 ($22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	166

地址	名称	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	页码
$01 ($21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	165
$00 ($20)	TWBR	两线串行接口位率寄存器 r								164

Notes: 1. 当 OCDEN 熔丝位未编程， OSCCAL 寄存器总访问该地址，具体请参见 OCDR 寄存器使用说明。
2. 如何访问 UBRRH 与 UCSRC 请参见 USART 的说明。
3. 为了和将来器件兼容，访问保留位时应该写 0。保留的 I/O 地址不可以执行写操作。
4. 一些状态标志可以通过写入逻辑 1 来清除。需要注意的是，不同于大多数其他的 AVR，CBI 和 SBI 指令只对一些特殊位有效，因此可以对那些包含标志位的寄存器进行操作。CBI 和 SBI 指令可使用的范围只能是地址为 0x00 - 0x1F 的寄存器。

指令集概述

指令	操作数	说明	操作	标志	# 时钟数
算数和逻辑指令					
ADD	Rd, Rr	无进位加法	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	带进位加法	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	立即数与字相加	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	无进位减法	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	减立即数	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	带进位减法	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	带进位减立即数	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	从字中减立即数	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	逻辑与	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	与立即数的逻辑与操作	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	逻辑或	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	与立即数的逻辑或操作	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	异或	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	1 的补码	Rd ← $FF – Rd	Z,C,N,V	1
NEG	Rd	2 的补码	Rd ← $00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	设置寄存器的位	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	寄存器位清零	Rd ← Rd • ($FF - K)	Z,N,V	1
INC	Rd	加一操作	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	减一操作	Rd ← Rd – 1	Z,N,V	1
TST	Rd	测试是否为零或负	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	寄存器清零	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	寄存器置位	Rd ← $FF	None	1
MUL	Rd, Rr	无符号数乘法	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	有符号数乘法	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	有符号数与无符号数乘法	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	无符号小数乘法	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	有符号小数乘法	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	有符号小数与无符号小数乘法	R1:R0 ← (Rd x Rr) << 1	Z,C	2
跳转指令					
RJMP	k	相对跳转	PC ← PC + k + 1	None	2
IJMP		间接跳转到 (Z)	PC ← Z	None	2
JMP	k	直接跳转	PC ← k	None	3
RCALL	k	相对子程序调用	PC ← PC + k + 1	None	3
ICALL		间接调用 (Z)	PC ← Z	None	3
CALL	k	直接子程序调用	PC ← k	None	4
RET		子程序返回	PC ← Stack	None	4
RETI		中断返回	PC ← Stack	I	4
CPSE	Rd,Rr	比较，相等则跳过下一条指令	if (Rd = Rr) PC ← PC + 2 or 3	None	1 / 2 / 3
CP	Rd,Rr	比较	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	带进位比较	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	与立即数比较	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	寄存器位为 "0" 则跳过下一条指令	if (Rr(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBRS	Rr, b	寄存器位为 "1" 则跳过下一条指令	if (Rr(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIC	P, b	I/O 寄存器位为 "0" 则跳过下一条指令	if (P(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIS	P, b	I/O 寄存器位为 "1" 则跳过下一条指令	if (P(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
BRBS	s, k	状态寄存器位为 "1" 则跳过下一条指令	if (SREG(s) = 1) then PC←PC+k + 1	None	1 / 2
BRBC	s, k	状态寄存器位为 "0" 则跳过下一条指令	if (SREG(s) = 0) then PC←PC+k + 1	None	1 / 2
BREQ	k	相等则跳转	if (Z = 1) then PC ← PC + k + 1	None	1 / 2
BRNE	k	不相等则跳转	if (Z = 0) then PC ← PC + k + 1	None	1 / 2
BRCS	k	进位位为 "1" 则跳转	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRCC	k	进位位为 "0" 则跳转	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRSH	k	大于或等于则跳转	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRLO	k	小于则跳转	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRMI	k	负则跳转	if (N = 1) then PC ← PC + k + 1	None	1 / 2
BRPL	k	正则跳转	if (N = 0) then PC ← PC + k + 1	None	1 / 2
BRGE	k	有符号数大于或等于则跳转	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1 / 2
BRLT	k	有符号数负则跳转	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1 / 2
BRHS	k	半进位位为 "1" 则跳转	if (H = 1) then PC ← PC + k + 1	None	1 / 2
BRHC	k	半进位位为 "0" 则跳转	if (H = 0) then PC ← PC + k + 1	None	1 / 2
BRTS	k	T 为 "1" 则跳转	if (T = 1) then PC ← PC + k + 1	None	1 / 2
BRTC	k	T 为 "0" 则跳转	if (T = 0) then PC ← PC + k + 1	None	1 / 2
BRVS	k	溢出标志为 "1" 则跳转	if (V = 1) then PC ← PC + k + 1	None	1 / 2
BRVC	k	溢出标志为 "0" 则跳转	if (V = 0) then PC ← PC + k + 1	None	1 / 2

指令	操作数	说明	操作	标志	# 时钟数
BRIE	k	中断使能再跳转	if (I = 1) then PC ← PC + k + 1	None	1 / 2
BRID	k	中断禁用再跳转	if (I = 0) then PC ← PC + k + 1	None	1 / 2
数据传送指令					
MOV	Rd, Rr	寄存器间复制	Rd ← Rr	None	1
MOVW	Rd, Rr	复制寄存器字	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	加载立即数	Rd ← K	None	1
LD	Rd, X	加载间接寻址数据	Rd ← (X)	None	2
LD	Rd, X+	加载间接寻址数据，然后地址加一	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	地址减一后加载间接寻址数据	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	加载间接寻址数据	Rd ← (Y)	None	2
LD	Rd, Y+	加载间接寻址数据，然后地址加一	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	地址减一后加载间接寻址数据	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	加载带偏移量的间接寻址数据	Rd ← (Y + q)	None	2
LD	Rd, Z	加载间接寻址数据	Rd ← (Z)	None	2
LD	Rd, Z+	加载间接寻址数据，然后地址加一	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	地址减一后加载间接寻址数据	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	加载带偏移量的间接寻址数据	Rd ← (Z + q)	None	2
LDS	Rd, k	从 SRAM 加载数据	Rd ← (k)	None	2
ST	X, Rr	以间接寻址方式存储数据	(X) ← Rr	None	2
ST	X+, Rr	以间接寻址方式存储数据，然后地址加一	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	地址减一后以间接寻址方式存储数据	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	加载间接寻址数据	(Y) ← Rr	None	2
ST	Y+, Rr	加载间接寻址数据，然后地址加一	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	地址减一后加载间接寻址数据	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	加载带偏移量的间接寻址数据	(Y + q) ← Rr	None	2
ST	Z, Rr	加载间接寻址数据	(Z) ← Rr	None	2
ST	Z+, Rr	加载间接寻址数据，然后地址加一	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	地址减一后加载间接寻址数据	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	加载带偏移量的间接寻址数据	(Z + q) ← Rr	None	2
STS	k, Rr	从 SRAM 加载数据	(k) ← Rr	None	2
LPM		加载程序空间的数据	R0 ← (Z)	None	3
LPM	Rd, Z	加载程序空间的数据	Rd ← (Z)	None	3
LPM	Rd, Z+	加载程序空间的数据，然后地址加一	Rd ← (Z), Z ← Z+1	None	3
SPM		保存程序空间的数据	(Z) ← R1:R0	None	-
IN	Rd, P	从 I/O 端口读数据	Rd ← P	None	1
OUT	P, Rr	输出端口	P ← Rr	None	1
PUSH	Rr	将寄存器推入堆栈	Stack ← Rr	None	2
POP	Rd	将寄存器从堆栈中弹出	Rd ← Stack	None	2
位和位测试指令					
SBI	P,b	I/O 寄存器位置位	I/O(P,b) ← 1	None	2
CBI	P,b	I/O 寄存器位清零	I/O(P,b) ← 0	None	2
LSL	Rd	逻辑左移	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	逻辑右移	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	带进位循环左移	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	带进位循环右移	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	算术右移	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	高低半字节交换	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	标志置位	SREG(s) ← 1	SREG(s)	1
BCLR	s	标志清零	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	从寄存器将位赋给 T	T ← Rr(b)	T	1
BLD	Rd, b	将 T 赋给寄存器位	Rd(b) ← T	None	1
SEC		进位位置位	C ← 1	C	1
CLC		进位位清零	C ← 0	C	1
SEN		负标志位置位	N ← 1	N	1
CLN		负标志位清零	N ← 0	N	1
SEZ		零标志位置位	Z ← 1	Z	1
CLZ		零标志位清零	Z ← 0	Z	1
SEI		全局中断使能	I ← 1	I	1
CLI		全局中断禁用	I ← 0	I	1
SES		符号测试标志位置位	S ← 1	S	1
CLS		符号测试标志位清零	S ← 0	S	1
SEV		2 的补码溢出标志置位	V ← 1	V	1
CLV		2 的补码溢出标志清零	V ← 0	V	1
SET		SREG 的 T 置位	T ← 1	T	1
CLT		SREG 的 T 清零	T ← 0	T	1
SEH		SREG 的半进位标志置位	H ← 1	H	1

指令	操作数	说明	操作	标志	# 时钟数
CLH		SREG 的半进位标志清零	H ← 0	H	1
MCU **控制指令**					
NOP		空操作		None	1
SLEEP		休眠	(对休眠功能，见 specific descr.)	None	1
WDR		复位看门狗	(对 WDR/timer 见 specific descr.)	None	1
BREAK		暂停	只针对片内调试	None	N/A

产品信息

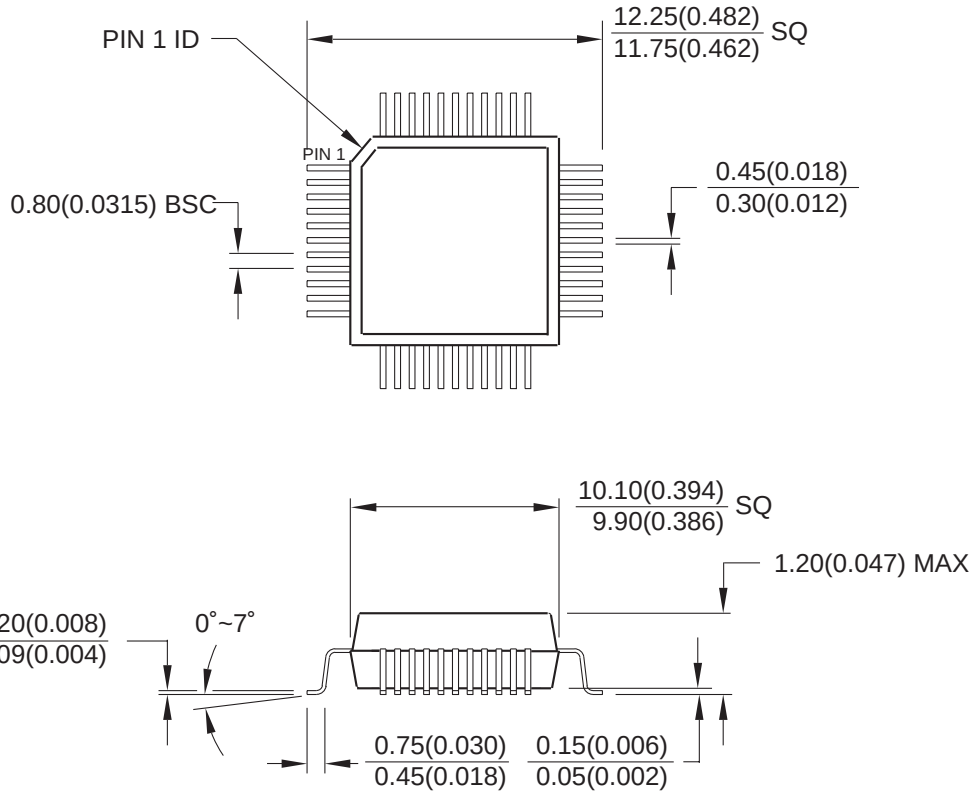
Speed (MHz)	Power Supply	Ordering Code	Package	Operation Range
8	2.7 - 5.5V	ATmega32L-8AC ATmega32L-8PC ATmega32L-8MC	44A 40P6 44M1	商业级 (0°C ~ 70°C)
		ATmega32L-8AI ATmega32L-8PI ATmega32L-8MI	44A 40P6 44M1	工业级 (-40°C ~ 85°C)
16	4.5 - 5.5V	ATmega32-16AC ATmega32-16PC ATmega32-16MI	44A 40P6 44M1	商业级 (0°C ~ 70°C)
		ATmega32-16AI ATmega32-16PI ATmega32-16MC	44A 40P6 44M1	工业级 (-40°C ~ 85°C)

封装类型	
44A	44- 引线，薄 (1.0 mm)TQFP
40P6	40- 引线，0.600” 宽，PDIP
44M1	44- 焊垫， 7 x 7 x 1.0 mm 大小，线距 0.50 mm， MLF

封装信息

44A

44-lead, Thin (1.0mm) Plastic Quad Flat Package
(TQFP), 10x10mm body, 2.0mm footprint, 0.8mm pitch.
Dimension in Millimeters and (Inches)*
JEDEC STANDARD MS-026 ACB

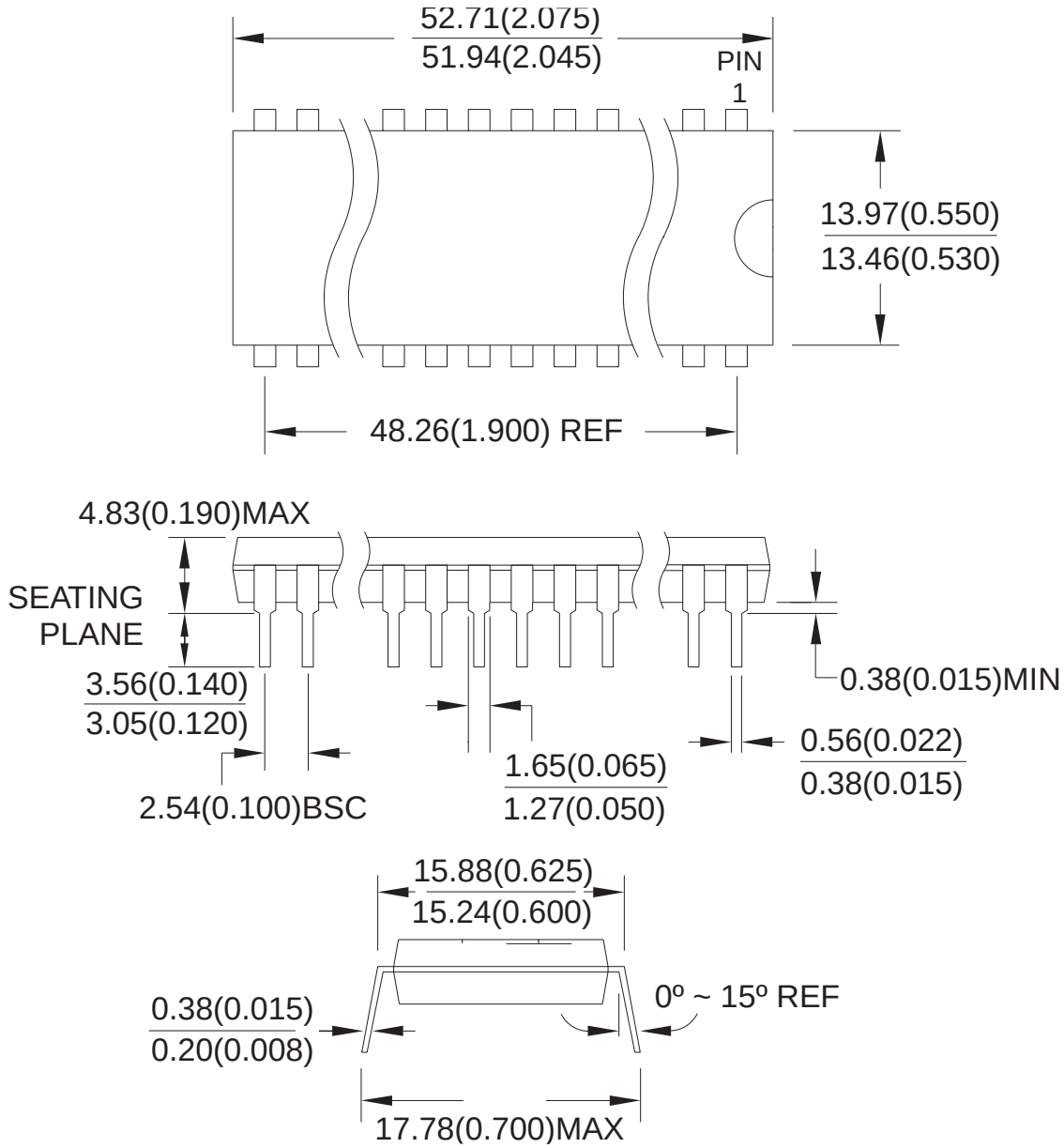


*Controlling dimension: millimeter

REV. A 04/11/2001

40P6

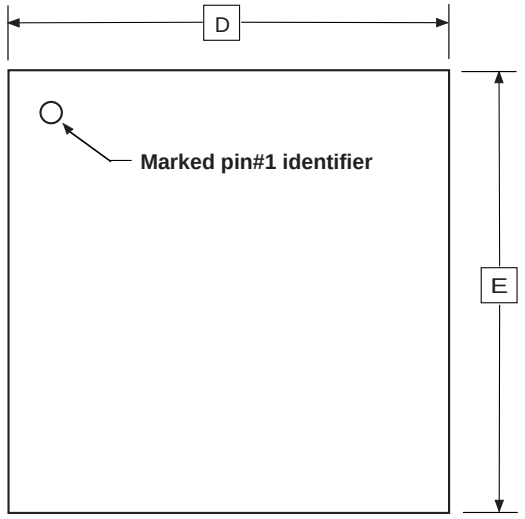
40-lead, Plastic Dual Inline
Package (PDIP), 0.600" wide
Dimension in Millimeters and (Inches)*
JEDEC STANDARD MS-011 AC



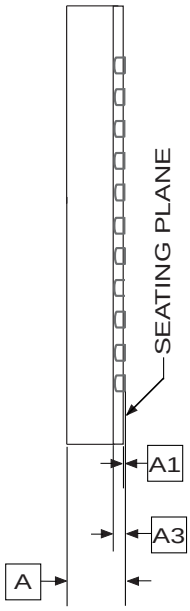
*Controlling dimension: Inches

REV. A 04/11/2001

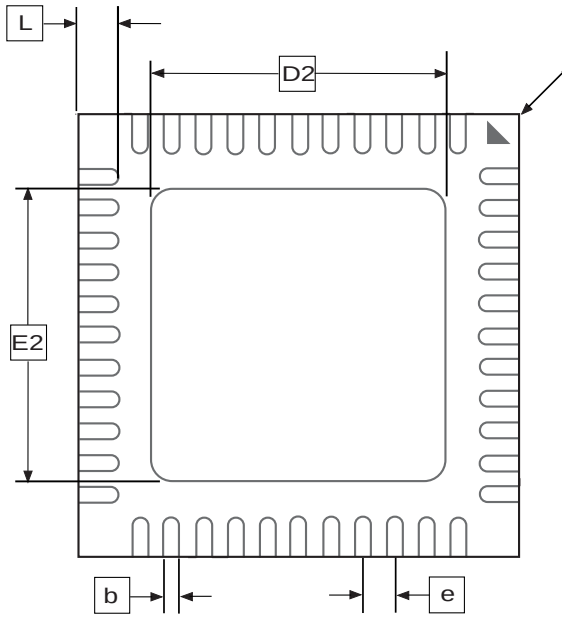
44M1



TOP VIEW



SIDE VIEW



BOTTOM VIEW

COMMON DIMENSIONS (*Unit of Measure = mm)				
SYMBOL	*MIN*	*NOM*	*MAX*	*NOTE*
A	0.80	0.90	1.00	
A1	0.00	0.02	0.05	
A3		0.25 REF		
b	0.18	0.23	0.30	
D		7.00 BSC		
D2	5.00	5.20	5.40	
E		7.00 BSC		
E2	5.00	5.20	5.40	
e		0.50 BSC		
L	0.35	0.55	0.75	

NOTE 1. JEDEC STANDARD MO-220, Fig 1 (Saw Singulation), VKKD-1

08/29/01

2325 Orchard Parkway San Jose, CA 95131	TITLE *44M1, 44-pad ,7 x 7 x 1.0 mm body, lead pitch 0.50mm Micro lead frame package (MLF)*		DRAWING NO. *44M1*	REV *B*

勘误表

ATmega32 Rev. A

该版本的 ATmega32 没有勘误表。下面为关于使用 JTAG 指令 IDCODEH 解决问题的建议。

IDCODE 屏蔽了从 TDI 输入的数据

按照 IEEE1149.1， JTAG 指令 IDCODE 工作不正确；当移位芯片 ID 寄存器时，输入不是 TDI 而是逻辑 "1"。从而，在边界扫描链时芯片运行所捕获的数据丢失，全部变为 "1"，送往后续器件的数据在 Update-DR 时全被 "1" 所代替。

如果 ATmega32 为扫描链上唯一的芯片，将无法看到问题。

解决方法

通过使用 IDCODE 命令或通过进入 TAP 控制器的 Test-Logic-Reset 状态来选择器件 ID 寄存器，然后读取其内容，以及后续器件的数据。读取扫描链中 ATmega32 前面器件的器件 ID 寄存器时对 ATmega32 发送 BYPASS 命令。当芯片 ID 寄存器选择 ATmega32 时，不要在扫描链中读数据或者更新数据。注意 TAP 控制器的 Test-Logic-Reset 状态的默认指令为 IDCODE 指令

其他 解决方法

如果必须同时获取扫描链中所有器件的 ID，那么 ATmega32 应该是扫描链中的第一个器件。 IDCODE 在 JTAG 指令寄存器中时， Update-DR 在选定芯片的边界扫描链中不工作，但芯片 ID 寄存器不能上传。

ATmega32 的数据手册变更日志

请注意本节中所提及的页码为手册中所对应的页码。请参看相应的版本号。

从版本 Rev. 2503E-09/03 到版本 Rev. 2503F-12/03 的变化

1. 更新 P27 “ 标定的片内 RC 振荡器 ” 。

从版本 Rev. 2503D-02/03 到版本 Rev. 2503E-09/03 的变化

1. 更新与修改 P33 “JTAG 接口与片上调试系统 ” 的 “ 片内调试系统 ”。
2. 更新 P35 Table 15 。
3. 更新 P204 “ 测试访问端口 –TAP” 中关于 JTAGEN 熔丝位部分。
4. 更新 Bit 7 – JTD: 禁止 JTAG 接口的说明。
5. 在 P240 Table 105 添加关于 JTAGEN 熔丝位的注释。
6. 更新 P269 “ 电气特性 ” 中绝对极限值 * , 直流特性与交流特性部分 。
7. 在 P292 “ 勘误表 ” 中加入一个关于 JTAG 指令 IDCODE 的解决问题的方法。

从版本 Rev. 2503C-10/02 到版本 Rev. 2503D-02/03 的变化

1. 在 P283 “ 寄存器概述 ” 中加入 EEARH 中的 EEAR9。
2. P266 “对Flash进行编程” 与 P267 “对EEPROM进行编程” 中添加第一步芯片擦除。
3. 删除 “ 多功能振荡器 ” 及 “32 kHz 晶振 ” 的说明部分。
4. 针对 Timer 0 与 Timer 2 添加关于 PWM 对称的内容。
5. P235 “ 装载临时缓冲器 (页加载)” 添加在 SPM 页加载时对 EEPROM 写。
6. 在 P1 “ 产品特性 ” 中加入功耗数据。
7. 添加 P20 “ 在掉电休眠模式下的 EEPROM 写操作 ” 部分。
8. 在 P190 “ 预分频及 ADC 转换时序 ” 中添加使用自动触发的差分模式的说明。
9. 更新 P216 Table 90 。
10. 添加更新 P289 “ 封装信息 ” 。

从版本 Rev. 2503B-10/02 到版本 Rev. 2503C-10/02 的变化

1. 更新 P269 “ 直流特性 ” 。

从版本 Rev. 2503A-03/02 到版本 Rev. 2503B-10/02 的变化

1. 将 Flash 的寿命改为 10,000 写入 / 擦除次。
2. 删除 SFIOR 寄存器的 Bit nr.4 – ADHSM 。
3. 添加 P23 “ 缺省时钟源 ” 部分。

4. 根据频率的变化在使用外部时钟时有一些限制，这部分说明见 P29 “ 外部时钟 ” 及 P269 Table 118 。

5. 在 P32 “ 最小化功耗 ” 中添加关于 OCD 系统及功耗的小节。

6. 校正类型 (WGM 位设置):
   – P71 “ 快速 PWM 模式 ” (T/C0)
   – P72 “ 相位修正 PWM 模式 ” (T/C0)
   – P111 “ 快速 PWM 模式 ” (T/C2)
   – P112 “ 相位修正 PWM 模式 ” (T/C2)

7. 修正 P150 Table 67 (USART)。

8. 更新 P269 “ 直流特性 ” 中参数 V_{IL}, I_{IL} 与 I_{IH} 值。

9. 更新关于 OSCCAL 标定字节的说明。
   在手册中，没有介绍如何使用标定字节为 2、 4、 8 MHz 的振荡器，这在下面的部分予以介绍：
   改变 P28 “ 振荡器标定寄存器 –OSCCAL” 及 P242 “ 标定字节 ” 中的说明。

10. Table 42 修正类型。

11. Table 45 与 Table 46 的修正说明。

12. 更新 Table 119, Table 121 与 Table 122。

13. 添加 P292 “ 勘误表 ” 。