产品特性

- 高性能、低功耗的 8 位 AVR® 微处理器
- 先进的 RISC 结构
  - 130 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
  - 16K 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
  - 512 字节的 EEPROM
    擦写寿命 : 100,000 次
  - 1K 字节的片内 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
  - 4 x 25 段的 LCD 驱动器
  - 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
  - 可编程的串行 USART
  - 可工作于主机 / 从机模式的 SPI 串行接口
  - 有开始状态检测器的通用串行接口 USI
  - 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
  - 引脚电平变化可引发中断及唤醒 MCU
- 特殊的微控制器特点
  - 上电复位 (POR) 以及可编程的掉电检测 (BOD)
  - 经过校准的片内 RC 振荡器
  - 片内、片外中断源
  - 五种休眠模式：空闲模式、 ADC 噪声抑制模式、省电模式、掉电模式和 Standby 模式
- I/O 口与封装
  - 53 个可编程的 I/O 口
  - 64 引脚 TQFP 封装 与 64 引脚 MLF 封装
- 工作电压 :
  - ATmega169V：1.8 - 5.5V
  - ATmega169L：2.7 - 5.5V
  - ATmega169：4.5 - 5.5V
- 工作温度范围 :
  - -40°C 至 85°C，工业级

具有 16KB 系统内可编程 Flash 的 8 位 AVR® 微控制器

ATmega169V
ATmega169L
ATmega169

初稿

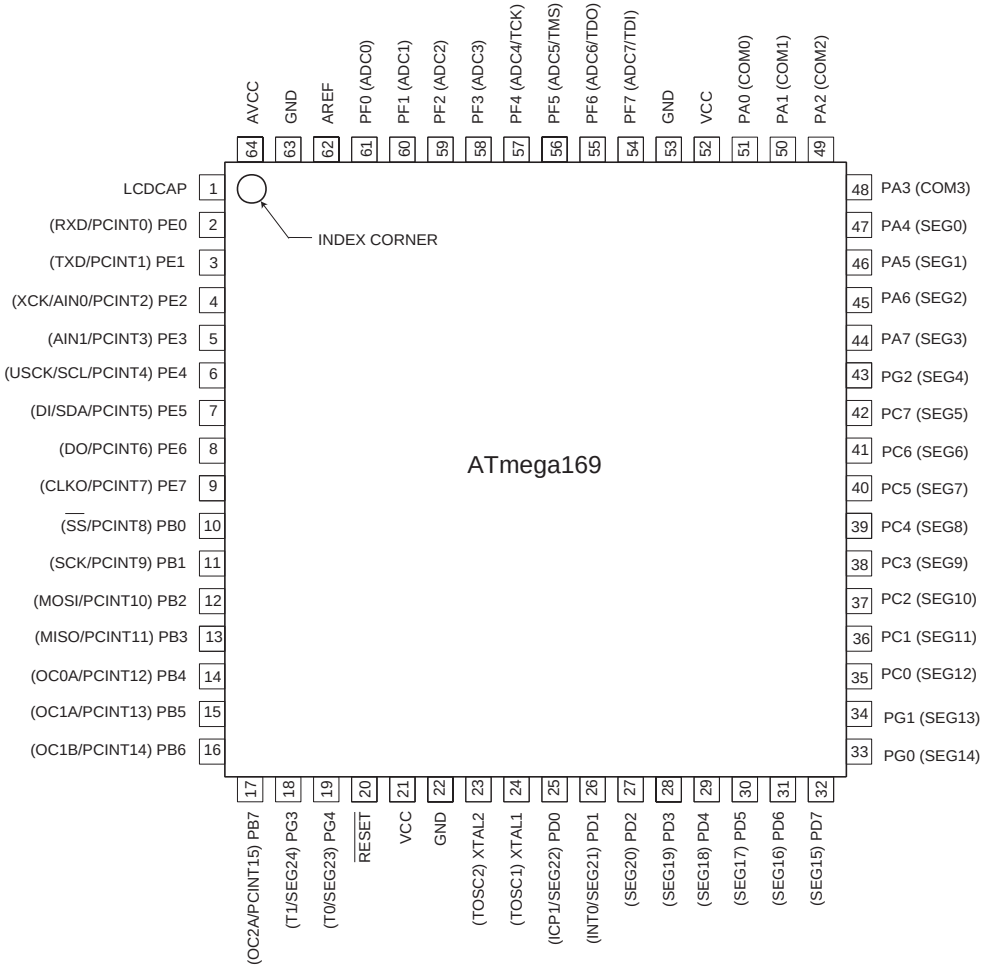
2514I–AVR–10/03

特性 (续)

- 工作速度 :
  - ATmega169V 0 - 1 MHz
  - ATmega169L 0 - 8 MHz
  - ATmega169 0 - 16 MHz
- 极低功耗
  - 正常模式 :
    - 1 MHz, 1.8V: 400µA
    - 32 kHz, 1.8V: 20µA (包括振荡器)
    - 32 kHz, 1.8V: 40µA (包括振荡器与 LCD)
  - 掉电模式 :
    - 1.8V ， 0.5µA

引脚配置

Figure 1. ATmega169 引脚排列



声明

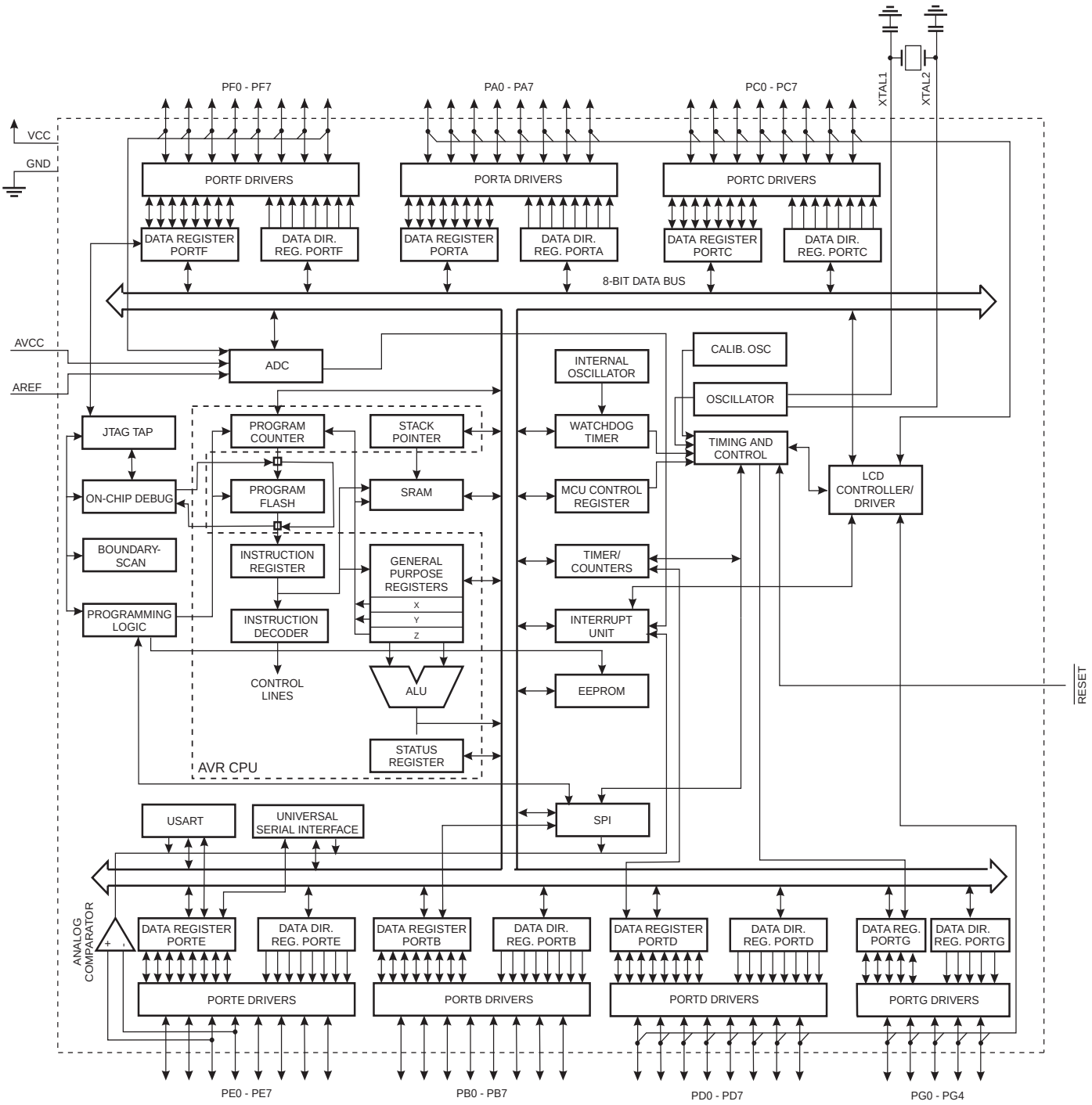
本数据手册的典型值来源于对器件的仿真，以及其他基于相同产生工艺的 AVR 微控制器的标定特性。本器件经过特性化之后将给出实际的最大值和最小值。

综述

ATmega169 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， ATmega169 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。

ATmega169 有如下特点 :16K 字节的系统内可编程 Flash(具有同时读写的能力，即 RWW) ,512 字节 EEPROM ,1K 字节 SRAM ,53 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，内含升压电路的 LCD 控制器，三个有比较模式灵活的定时器 / 计数器 (T/C), 片内 / 外中断，可编程串行 USART，有起始条件检测器的通用串行接口，8 路 10 位 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，以及五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 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，寄存器的内容则一直保持；在省电模式下，异步定时器与 LCD 控制器继续运行，允许用户保持一个时间基准及对 LCD 显示器进行操作，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、 LCD 控制器与 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。

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，ATmega169 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。

ATmega169 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 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 A 也可以用做其他不同的特殊功能，请参见 P57。

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 比其余端口的驱动性要好。

端口 B 也可以用做其他不同的特殊功能，请参见 P58。

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。

端口 C 也可以用做其他不同的特殊功能，请参见 P61。

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 P63。

端口 E (PE7..PE0) 端口 E 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 E 处于高阻状态。

端口 E 也可以用做其他不同的特殊功能，请参见 P65。

端口 F (PF7..PF0) 端口 F 是 ADC 的模拟输入引脚。

如果不作为 ADC 的模拟输入，端口 F 可以作为 8 位双向 I/O 口，并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 F 呈现为三态。如果使能了 JTAG 接口，则复位发生时引脚 PF7(TDI)、PF5(TMS) 与 PF4(TCK) 的上拉电阻使能。

端口 F 也可以作为 JTAG 接口。

端口 G (PG4..PG0) 端口 G 为 5 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 G 仍呈现为三态。

端口 G 也可以用做其他不同的特殊功能，请参见 P67。

$\overline{\text{RESET}}$ 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 P37 Table 16 。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F与ADC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 ADC 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。

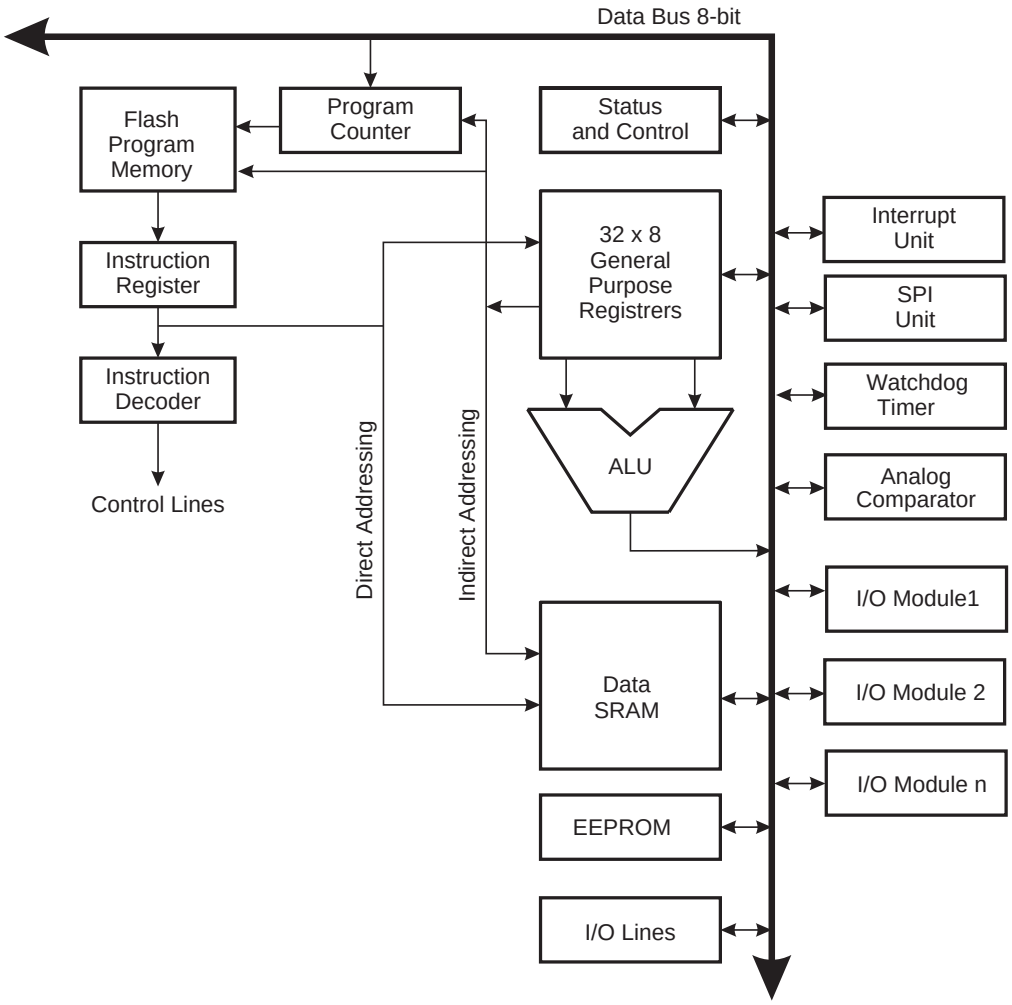
AVR CPU 内核

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 具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。此外，ATmega169 还有位于 SRAM 地址 0x60 - 0xFF 的扩展 I/O 空间。扩展 I/O 空间只能使用 ST/STS/STD 和 LD/LDS/LDD 指令来访问。

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：

位	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	

- **位 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。

- **位 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。

- **位 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。

- **位 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。

- **位 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。

- **位 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。

- **位 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。

- **位 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。

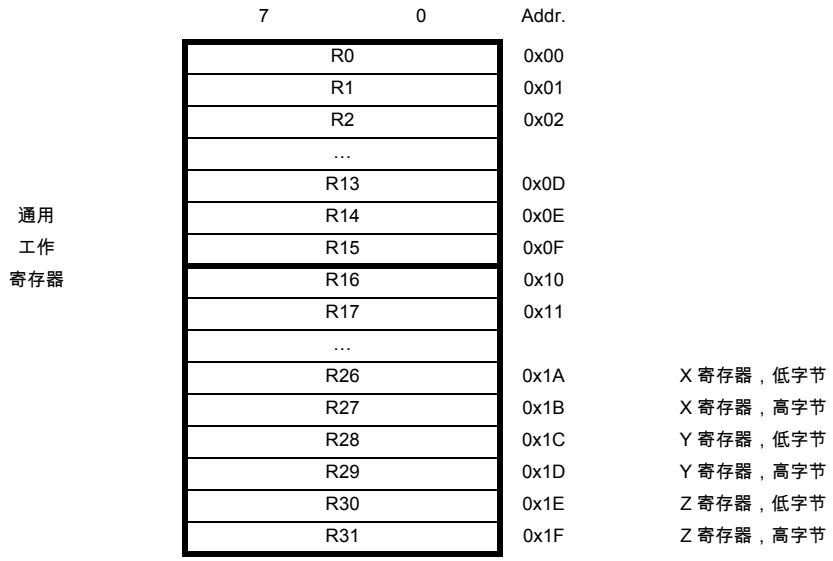
通用寄存器文件

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 32 x 8 通用工作寄存器



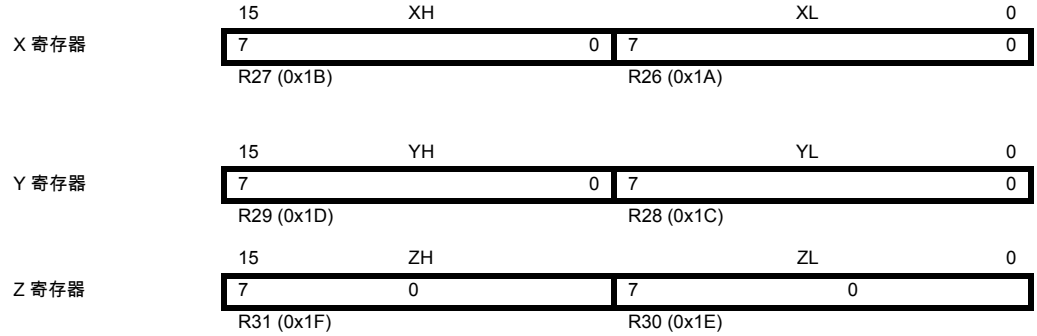
大多数操作寄存器文件的指令都可以直接访问所有的寄存器，而且多数这样的指令的执行时间为单个时钟周期。

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 0xFF 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。

位	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

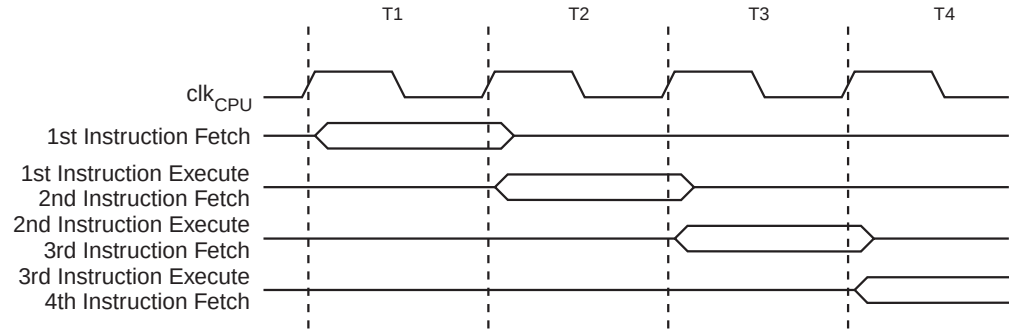
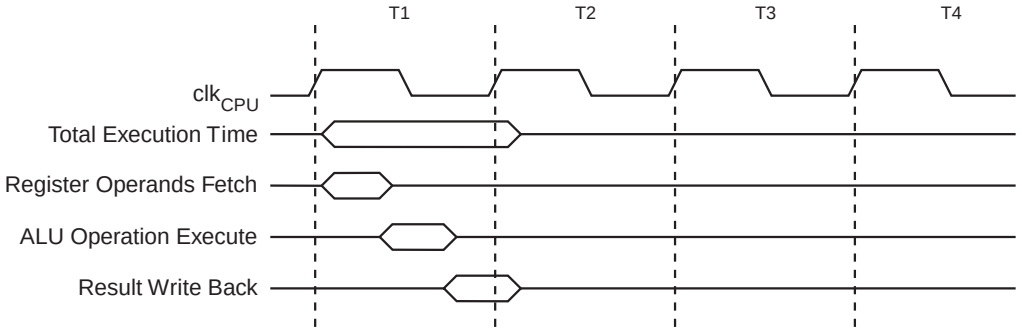


Figure 7 演示的是寄存器文件内部访问时序。在一个时钟周期里 ,ALU 可以同时对两个寄存器操作数进行操作，同时将结果保存到目的寄存器中去。

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 P256 “ 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P45 “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 MCU 控制寄存器 (MCUCR) 的 IVSEL ,中断向量可以移至引导Flash的起始处。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 P243 “Boot Loader 支持 – RWW 自编程 ” 。

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 内容的可能破坏。

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。这个时间不包括在前面提到的时钟周期里。

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 ATmega169 的存储器

本节讲述 ATmega169 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， ATmega169 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

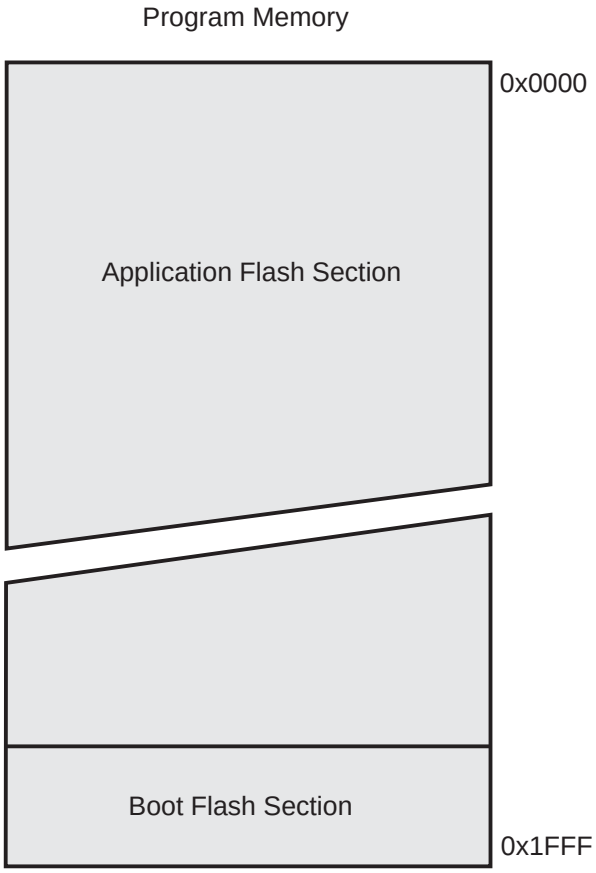
ATmega169具有16K字节的在线编程Flash ,用于存放程序指令代码。因为所有的AVR指令为 16 位或 32 位 ,故而 Flash 组织成 8K x 16 位的形式。用户程序的安全性要根据 Flash 程序存储器的两个区：引导 (Boot) 程序区和应用程序区，分开来考虑。

Flash 存储器至少可以擦写 10,000 次。ATmega169 的程序计数器 (PC) 为 13 位 ,因此可以寻址8K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见 P243 “Boot Loader 支持 – RWW 自编程 ” ,而 P256 “ 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 P12 “ 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 ATmega169 SRAM 空间的组织结构。

ATmega169是一个复杂的微控制器 ,其支持的外设要比预留的64个I/O(通过IN/OUT指令访问) 所能支持的要多。对于扩展的 I/O 空间段 0x60 - 0xFF，只能使用 ST/STS/STD 和 LD/LDS/LDD 指令进行访问。

前 1,280 个数据存储器包括了寄存器文件、 I/O 存储器、扩展的 I/O 存储器以及内部数据 SRAM。起始的32个地址为寄存器文件，然后是64个I/O存储器，接着是160个扩展I/O存储器。最后是 1024 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。

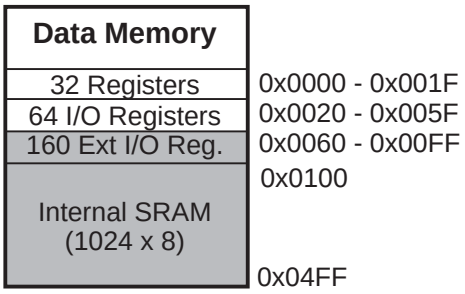
直接寻址范围可达整个数据区。

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。

ATmega169的全部32个通用寄存器、64个I/O寄存器、160个扩展I/O寄存器及1024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 P9 “通用寄存器文件”。

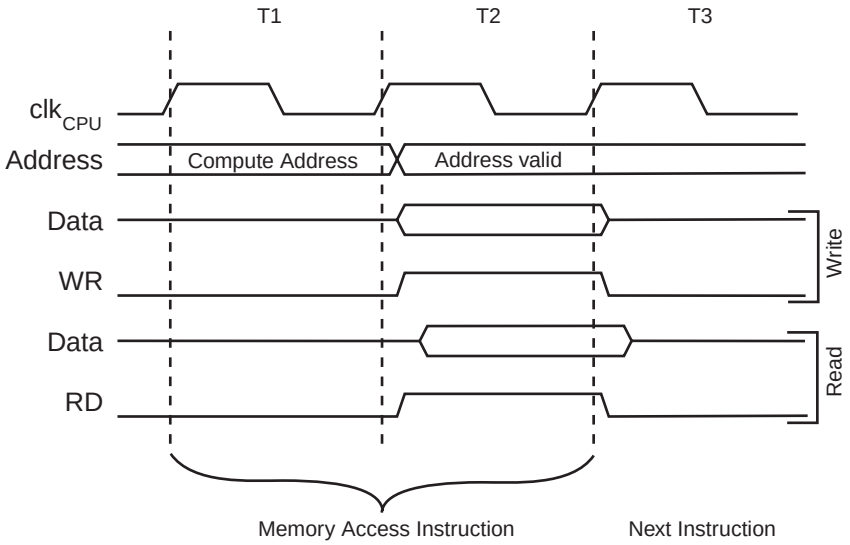
Figure 9. 数据存储器映像



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

ATmega169包含512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。

通过SPI和JTAG及并行电缆下载EEPROM数据的操作请分别参见P269、P274及 P259。

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 P21 “ 防止 EEPROM 出错 ” 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

位	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	–	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	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	

- **位 15..9 – Res: 保留**

保留位，读操作返回值为零。

- **位 8..0 – EEAR8..0: EEPROM 地址**

EEPROM地址寄存器– EEARH和EEARL指定了512字节的EEPROM空间。EEPROM地址是线性的，从 0 到 511。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

位	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	

- **位 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

位	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	

- **位 7..4 – Res: 保留**

保留位，读操作返回值为零。

- **位 3 – EERIE: 使能 EEPROM 准备好中断**

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。

- **位 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。

- **位 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 P243 “Boot Loader 支持 – RWW 自编程 ”。

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。

- **位 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)	67 584	8.5 ms

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:
  ; W 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，单片机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

.ATmega169 的 I/O 空间定义见 P325 “ 寄存器概述 ” 。

ATmega169 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 LD/LDS/LDD 与 ST/STS/STD 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。ATmega169 是一个复杂的微控制器，其支持的外设要比预留的 64 个 I/O(通过 IN/OUT 指令访问) 所能支持的要多。对于扩展的 I/O 空间 0x60 - 0xFF，只能使用 ST/STS/STD 和 LD/LDS/LDD 指令进行寻址。

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。

通用 I/O 寄存器

ATmega169 包含 3 个通用 I/O 寄存器。这些寄存器可以用来存储信息，尤其适合于存储全局变量与状态标志。位于 0x00 - 0x1F 的通用 I/O 寄存器可以通过 SBI、 CBI、 SBIS 与 SBIC 指令直接进行位寻址。

通用 I/O 寄存器 2—GPIOR2

位	7	6	5	4	3	2	1	0	
	MSB							LSB	GPIOR2
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	

通用 I/O 寄存器 1—GPIOR1

位	7	6	5	4	3	2	1	0	
	MSB							LSB	GPIOR1
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	

通用 I/O 寄存器 0—GPIOR0

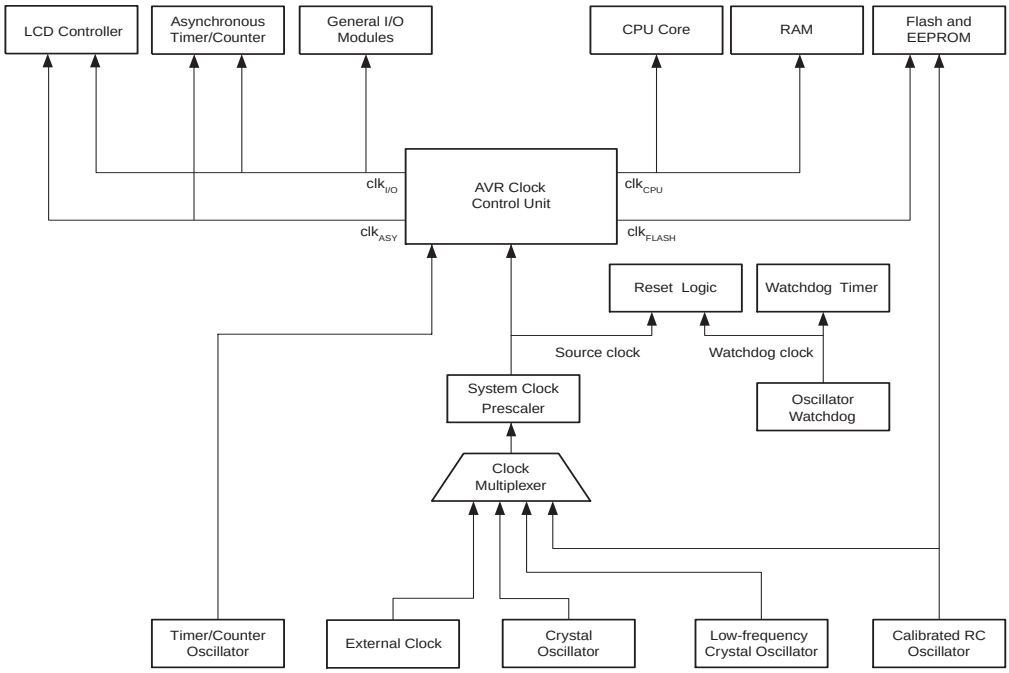
位	7	6	5	4	3	2	1	0	
	MSB							LSB	GPIOR0
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

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 P32 “ 电源管理及休眠模式 ”。

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 / 计数器与 LCD 控制器直接由外部 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。且当系统处于休眠状态时， LCD 控制器仍可继续输出。

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

ATmega169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)

器件时钟选项	熔丝位 CKSEL3..0
外部晶体 / 陶瓷振荡器	1111 - 1000
外部低频晶振	0111 - 0100
标定的内部 RC 振荡器	0010
外部时钟	0000
保留	0011, 0001

注：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 P292 “ATmega169 典型特性 – 初始数据 ”。

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

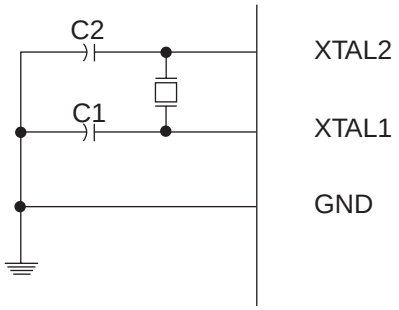
器件出厂时 CKSEL = “0010”， SUT = “10”，并且 CKDIV8 被编程。这个缺省设置的时钟源是内部 RC 振荡器，启动时间为最长，并且对时钟进行 8 分频。这种设置保证用户可以通过 ISP 或并行编程器得到所需的时钟源。

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。

无论外接的是石英晶体还是陶瓷振荡器，电容 C1、 C2 的值总是相等的。具体电容值的选择取决于使用的是石英晶体还是陶瓷振荡器，及总的杂散电容与环境电磁噪声等。Table 4 给出了采用石英晶体时的电容选择范围。使用陶瓷振荡器时，电容值应采用生产商给出的值。

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

CKSEL3..1	频率范围 (MHz)	使用晶体时电容 C1 和 C2 的推荐范围 (pF)
100(1)	0.4 - 0.9	–
101	0.9 - 3.0	12 - 22
110	3.0 - 8.0	12 - 22
111	8.0 -	12 - 22

Notes: 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)	14CK + 4.1 ms	陶瓷谐振器，电源快速上升
0	01	258 CK(1)	14CK + 65 ms	陶瓷谐振器，电源缓慢上升
0	10	1K CK(2)	14CK	陶瓷谐振器， BOD 使能
0	11	1K CK(2)	14CK + 4.1 ms	陶瓷谐振器，电源快速上升
1	00	1K CK(2)	14CK + 65 ms	陶瓷谐振器，电源缓慢上升
1	01	16K CK	14CK	陶瓷谐振器， BOD 使能
1	10	16K CK	14CK + 4.1 ms	石英振荡器，电源快速上升
1	11	16K CK	14CK + 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 “0100”、“0101”、“0110” 或 “0111”，以选择低频晶体振荡器。晶体的连接方式如 Figure 12 所示。选定钟表晶体振荡器后，启动时间由熔丝位 SUT(Table 6 所示) 与 CKSEL1..0(Table 7 所示) 确定。

Table 6. 低频晶体振荡器的启动时间

SUT1..0	复位时的额外延迟时间 (V_{CC} = 5.0V)	推荐用法
00	14CK	电源快速上升，或是 BOD 使能
01	14CK + 4.1 ms	电源缓慢上升
10	14CK + 65 ms	启动时频率已经稳定
11	保留	

Table 7. 低频晶振启动时的时钟选择

CKSEL3..0	掉电模式和省电模式下的启动时间	推荐用法
0100(1)	1K CK	
0101	32K CK	启动时频率已经稳定
0110(1)	1K CK	
0111	32K CK	启动时频率已经稳定

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。

校准的片内 RC 振荡器

经过校准的片内 RC 振荡器提供了固定的 8.0 MHz 时钟，这是在 3V、 25°C 下的标称数值。如果器件的工作条件无法达到8 MHz 的频率 (由 V_{CC} 决定) ，必须编程熔丝位 CKDIV8 实现在启动时对内部频率进行 8 分频。CKDIV8 在器件出厂时已经被编程，如 P30 “ 系统时钟预分频器 ” 所示。按照 Table 8 对熔丝位 CKSEL 进行编程即可将其作为系统时钟。选择这个时钟之后就无需外部器件了。复位时硬件将校准字节加载到 OSCCAL 寄存器，自动完成对 RC 振荡器的标定。在 3V、25°C 时，这种标定可以提供标称频率 ± 1% 的精度。使用这个振荡器作为系统时钟时，看门狗振荡器继续为看门狗定时器和溢出复位提供时钟。更多的有关校准数据的信息请参见 P259 “ 校准字节 ” 。

Table 8. I 校准的片内 RC 振荡器工作模式

CKSEL3..0	标称频率
0010	8.0 MHz

Note: 1. 出厂时的设置

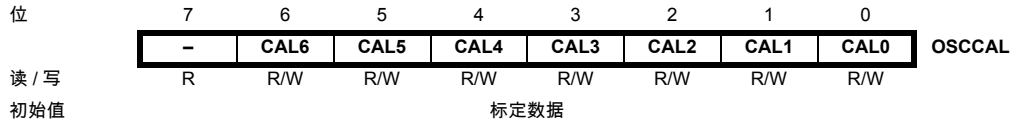
选择了这个振荡器之后，启动时间由熔丝位 SUT 确定，如 Table 9 所示。选择内部 RC 振荡器后引脚 XTAL1/TOSC1 与 XTAL2/TOSC2 可以用作定时器振荡器的引脚。 .

Table 9. 内部标定 RC 振荡器的启动时间

SUT1..0	掉电与省电模式启动时间	复位延时启动时间 (V_{CC} = 5.0V)	推荐用法
00	6 CK	14CK	BOD 使能
01	6 CK	14CK + 4.1 ms	电源快速上升
10(1)	6 CK	14CK + 65 ms	电源缓慢上升
11	保留		

Note: 1. 出厂时的设置

振荡器校准寄存器—OSCCAL



- **位 6..0 – CAL6..0: 振荡器校准数据**

将校准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。这在芯片复位时自动完成。 OSCCAL 为零时振荡器以最低频率工作。对其写入不为零的数据时内部振荡器的频率将增加。写入 0x7F 即得到最高频率。标定的振荡器用来为访问 EEPROM 和 Flash 定时。当应用有写 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。要注意只有 8.0 MHz 这个频率得到校准，其他频率则没有。详见 Table 10 的说明。 .

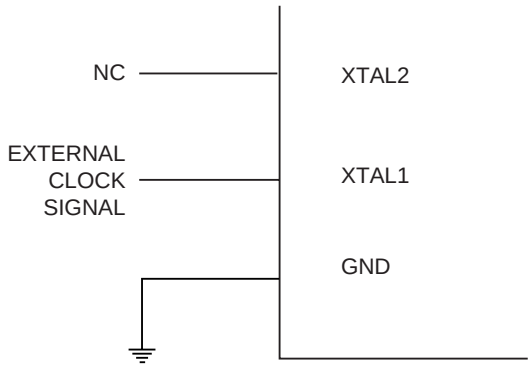
Table 10. 内部 RC 振荡器频率范围

OSCCAL 的值	最小频率 (标称频率的百分比)	最大频率 (标称频率的百分比)
0x00	50%	100%
0x3F	75%	150%
0x7F	100%	200%

外部时钟

为了从外部时钟源驱动芯片， XTAL1 必须如 Figure 13 所示那样进行连接。同时，熔丝位 CKSEL 必须设为 “0000”。

Figure 13. 外部时钟配置图



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

Table 11. 晶体振荡器时钟频率

CKSEL3..0	频率范围
0000	0 - 16 MHz

Table 12. 外部时钟的启动时间

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	6 CK	14CK	BOD 使能
01	6 CK	14CK + 4.1 ms	电源快速上升
10	6 CK	14CK + 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。

要注意的是，系统时钟预分频可以实现在运行期间改变内部时钟频率而保持系统稳定运行。请参见 P30 “ 系统时钟预分频器 ” 。

时钟输出缓冲器

CKOUT 熔丝位编程后，系统时钟将从 CLKO 输出。这种模式适用于需要芯片时钟驱动系统内其它电路的情况。即使芯片处于复位状态，此时钟还会被输出。对 CKOUT 熔丝位编程后 I/O 口的正常操作被切换为时钟输出。当 CLKO 作为时钟输出时，系统时钟可以为包括片内 RC 振荡器在内的所有时钟源。如果使用系统时钟预分频，输出的是被分频后的系统时钟频率。

定时器 / 计时器振荡器

ATmega169 的定时器/计数器引脚TOSC1/TOSC2和XTAL1 /XTAL2是共用的。这意味着只有当校准的内部 RC 振荡器作为系统时钟源时定时器 / 计数器振荡器才能使用。此振荡器针对 32.768 kHz 的钟表晶体作了优化。参见 P25 Figure 12 的晶体振荡器连接图。

如果 ASSR 寄存器的 EXTCLK 为逻辑 ”1”，则可以对 TOSC1 使用外部时钟源。如何想使用外部时钟而不是 32 kHz 的晶体，请参见 P131 “ 定时器 / 计数器的异步操作 ” 。

系统时钟预分频器

ATmega169可以通过设置时钟预分频寄存器CLKPR来得到分频的系统时钟。当需要的系统处理能力比较低时可以利用这个特性来降低功耗。预分频对所有时钟源都适用，并且影响 CPU 及所有同步外设的时钟频率。$clk_{I/O}$、clk_{ADC}、clk_{CPU} 和 clk_{FLASH} 则根据 Table 13 所列出的系数进一步被分频。

时钟预分频寄存器—CLKPR

位	7	6	5	4	3	2	1	0	
	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0	CLKPR
读 / 写	R/W	R	R	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	见位说明				

- **位 7 – CLKPCE: 时钟预分频器更新使能**

如果要改变 CLKPS 的内容， CLKPCE 必须写入逻辑 "1”。只有当 CLKPR 的其它位同时写入 "0” 时， CLKPCE 位才会更新。在 CLKPCE 被写入逻辑 "1” 之后 4 个时钟周期，或是 CLKPS 位被执行写操作之后 4 个时钟周期， CLKPCE 将被硬件清除。在这 4 个时钟周期的计时时间内修改 CLKPCE 既不会延长此计时时间，也不会清零 CLKPCE。

- **位 3..0 – CLKPS3..0: 时钟预分频选择位 3 - 0**

这几位确定了被选定的时钟源与内部系统时钟间的分频因子。它们在系统运行期间可以进行修改来改变时钟频率以适应实际需要。由于分频器分频的对象是 输入到 MCU 的主时钟，使用分频因子将改变所有同步外设的运行速度。分频因子在 Table 13 中给出。

为避免无意中改变时钟频率，在改变 CLKPS 时后必须执行一个特殊的写程序：

1. 将时钟预分频器更新使能 (CLKPCE) 置位，同时将 CLKPR 的其余位清零。
2. 在随后的 4 个时钟周期内，将需要的数值写入 CLKPS 并同时将 CLKPCE 清零。

为保证写操作不被中断，在改变预分频设置时必须禁止中断。

熔丝位 CKDIV8 决定了 CLKPS 的初始值。如果 CKDIV8 未编程， CLKPS 将被设为 “0000”。如果CKDIV8已编程，CLKPS将被设为“0011”，在芯片启动时将8作为分频因子。如果选择的时钟源超出了器件工作条件所能允许的最大频率值，则一定要编程这个熔丝位。不论 CKDIV8 熔丝如何设定， CLKPS 的值都可以任意进行修改。因此当选择的时钟源超出了器件工作条件所能允许的最大频率值时，应用软件必须使用合适的分频因子。CKDIV8 熔丝出厂时已编程。

Table 13. 时钟预分频选择

CLKPS3	CLKPS2	CLKPS1	CLKPS0	时钟分频因子
0	0	0	0	1
0	0	0	1	2
0	0	1	0	4
0	0	1	1	8
0	1	0	0	16
0	1	0	1	32
0	1	1	0	64

Table 13. 时钟预分频选择

CLKPS3	CLKPS2	CLKPS1	CLKPS0	时钟分频因子
0	1	1	1	128
1	0	0	0	256
1	0	0	1	保留
1	0	1	0	保留
1	0	1	1	保留
1	1	0	0	保留
1	1	0	1	保留
1	1	1	0	保留
1	1	1	1	保留

时钟切换时间

在预分频设置转换过程中，系统时钟预分频器确保时钟系统内不会产生干扰脉冲，并且在切换过程中产生的中间频率不会高于分频设置前后对应的时钟频率。

实现预分频器的纹波计数器的操作对象为未分频的时钟，此时钟可能比CPU的时钟要高。因此，即使预分频器可读，也无法确定其状态，并且由一个时钟转换到另一个时钟的切换时间也无法准确预测。

从写入 CLKPS 的值开始，新的时钟频率需要 T1 + T2 到 T1 + 2*T2 的时间才能生效。在此期间将产生 2 个有效的时钟沿。 T1 为前一种时钟的时钟周期， T2 为新的预分频器设置的时钟周期。

电源管理及休眠模式

休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 具有不同的休眠模式，允许用户根据自己的应用要求实施剪裁。

进入 5 个休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、省电模式和 Standby 模式）由 MCUCR 的 SM2、SM1 和 SM0 决定，如 Table 14 所示。使能的中断可以将进入休眠模式的 MCU 唤醒。经过启动时间，外加 4 个时钟周期（此时 MCU 停止）后，MCU 就可以运行中断服务程序了。然后 MCU 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。如果在休眠过程中发生了复位，则 MCU 从复位向量开始执行。

P23 Figure 11 介绍了 ATmega169 不同的时钟系统及其分布。此图在选择合适的休眠模式时非常有用。

休眠模式控制寄存器—SMCR

休眠模式控制寄存器包含了电源管理的控制位。 .

位	7	6	5	4	3	2	1	0	
	–	–	–	–	SM2	SM1	SM0	SE	SMCR
读 / 写	R	R	R	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 3、 2、 1 – SM2..0: 休眠模式选择位 2、 1 和 0**

如 Table 14 所示，这些位用于选择具体的休眠模式。

Table 14. 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 模式
1	1	1	保留

Note: 1. 仅在使用外部晶体或谐振器时 Standby 模式才可用。

- **位 1 – 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。

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 LCD 控制器、SPI、USART、模拟比较器、ADC、USI、定时器 / 计数器、看门狗和中断系统继续工作。这个休眠模式只停止了 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、USI 起始条件检测中断、定时器 / 计数器 2、LCD 控制器和和看门狗继续工作 (如果已经使能)。这个休眠模式只停止了 $clk_{I/O}$、clk_{CPU} 和 clk_{FLASH}，其他时钟则继续工作。

此模式改善了 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、LCD 控制器中断、USI 起始条件检测中断、定时器 / 计数器 2 中断、SPM/EEPROM 准备好中断、外部中断 INT0 或引脚电平变化中断可以将 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、USI 起始条件检测中断及看门狗 (如果使能的话) 继续工作。只有外部复位、看门狗复位、BOD 复位、USI 起始条件检测中断、外部电平中断 INT0，以及引脚电平变化中断可以使 MCU 脱离掉电模式。这个休眠模式基本停止了所有的时钟，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，必须使外部电平保持一定的时间。具体请参见 P74 “ 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。唤醒时间与由熔丝位 CKSEL 定义的复位时间是一样的，具体描述参见 P24 “ 时钟源 ”。

省电模式

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 及 / 或 LCD 控制器是使能的，在器件休眠期间它们继续运行。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 TIMSK2 使能了这些中断，而且 SREG 的全局中断使能位 I 置位。LCD 控制器中断也可唤醒 MCU。

如果定时器 / 计数器 2 及 LCD 控制器均无需运行，建议使用掉电模式而不是省电模式。

定时器 / 计数器 2 及 LCD 控制器在省电模式下可采用同步或异步时钟驱动。两模块的时钟源可独立选择。如果 LCD 控制器与定时器 / 计数器 2 均未采用异步时钟，休眠期间定时 / 计数振荡器将停止；如果 LCD 控制器与定时器 / 计数器 2 均未采用同步时钟，休眠期间时钟源将停止。要注意的是，在省电模式下同步时钟只对 LCD 控制器与定时器 / 计数器 2 有效。

Standby 模式

当 SM2..0 为 110 ，且选择了外部晶体振荡器或陶瓷谐振器作为时钟源， 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。

Table 15. 在不同睡眠模式下活动的时钟以及唤醒 MCU 的来源

	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	**INT0 与引脚电平变化**	**USI 起始条件**	**LCD 控制器**	**定时器 2**	**SPM/ EEPROM 准备好**	**ADC**	**其它 I/O**
空闲模式			X	X	X	X	X(2)	X	X	X	X	X	X	X
ADC 噪声抑制模式				X	X	X	X(2)	X(3)	X	X(2)	X(2)	X	X	
掉电模式								X(3)	X					
省电模式					X		X	X(3)	X	X	X			
Standby 模式						X		X(3)	X					

Notes: 1. 时钟源为外部晶体或谐振器
2. LCD 控制器或定时器 / 计数器 2 工作在异步模式下
3. 电平类型的 INT0 中断

功耗最小化

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，要尽可能利用休眠模式，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 在所有休眠模式下都继续工作。为了降低功耗，在进入休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。详见 P184 “ 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。在其他休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，则不论在什么休眠模式下都需要通过程序来关闭它。否则内部电压基准源将一直使能。请参见 P181 “ 模拟比较器 ” 以了解如何配置模拟比较器。

掉电检测 BOD

如果系统没有利用掉电检测器 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 P39“ 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 P41“ 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

如果系统无需利用看门狗，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 P42 “ 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 P54 “ 数字输入使能和休眠模式 ” 。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。

模拟输入引脚的数字输入缓冲器应一直禁用。否则，即使当输入引脚工作于模拟输入状态，当模拟信号电压接近 $V_{CC}/2$ 时输入缓冲器需要消耗很大的电流。可以通过操作数字输入禁止寄存器 (DIDR1 与 DIDR0) 来禁止数字输入缓冲器。具体请参见 P183 “ 数字输入禁止寄存器 1 – DIDR1” 与 P201 “ 数字输入禁止寄存器 0 – DIDR0” 。

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。

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 14 为复位逻辑的电路图。 Table 16 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 P24 “ 时钟源 ” 。

复位源

ATmega169 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 P224 “IEEE 1149.1 (JTAG) 边界扫描 ” 。

Figure 14. 复位逻辑

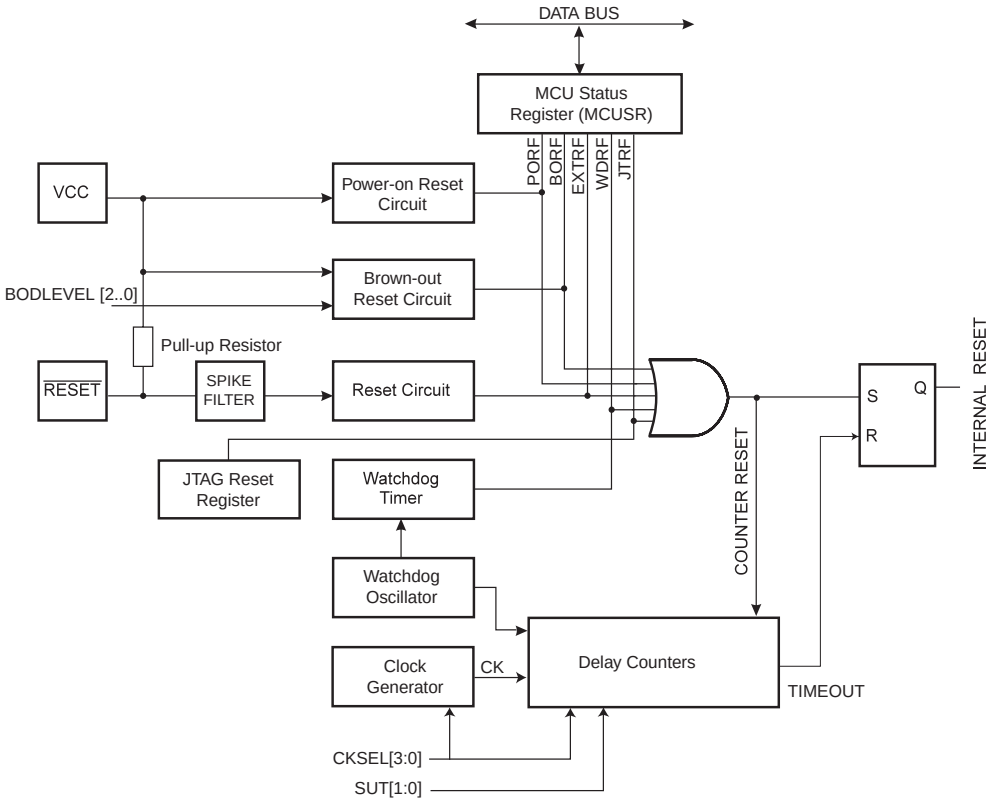


Table 16. 复位特性

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)	T_A = -40°C 到 85°C	0.7	1.0	1.4	V
	上电复位门限电压 (电压由高到低跌落)(1)	T_A = -40°C 到 85°C	0.6	0.9	1.3	V
V_{RST}	$\overline{\text{RESET}}$ 门限电压	V_{CC} = 3V	0.1 V_{CC}		0.9 V_{CC}	V
t_{RST}	$\overline{\text{RESET}}$ 最小脉冲宽度	V_{CC} = 3V			2.5	µs

Notes: 1. 电压下降时，只有电压低于 V_{POT} 时复位才会发生。

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

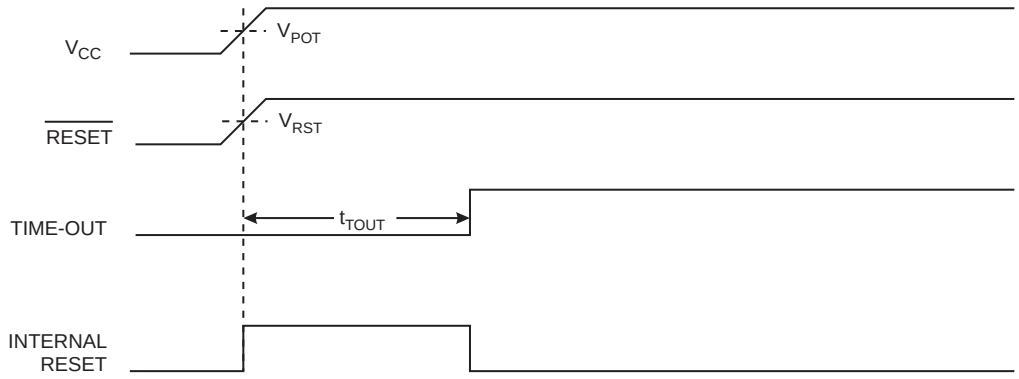
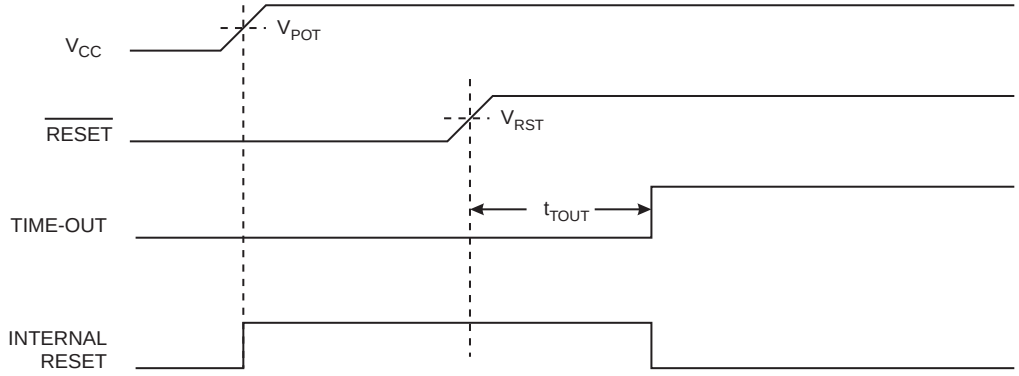


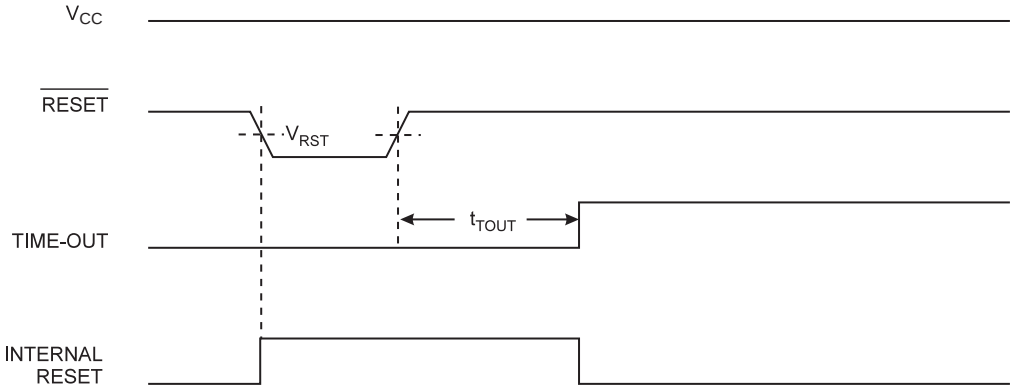
Figure 16. MCU 启动过程，$\overline{\text{RESET}}$ 由外电路控制



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 16) 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 17. 工作过程中发生外部复位



掉电检测

ATmega169 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。 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 $V_{BOT-} = V_{BOT} - V_{HYST}/2$。

Table 17. BODLEVEL 熔丝位编码 (1)

BODLEVEL 2..0	最小 V_{BOT}	典型 V_{BOT}	最大 V_{BOT}	单位
111	BOD 被禁用			
110	1.5	1.8	2.1	V
101	2.4	2.7	3.0	
100	4.0	4.5	4.6	
011	保留			
010				
001				
000				

Note: 1. V_{BOT} 可能低于某些器件的最小标称工作电压。对于有这种情形的器件，在产品测试时将做 $V_{CC} = V_{BOT}$ 的实验。这保证了在芯片工作电压 V_{CC} 降至微处理器已经无法正常工作之前 ,掉电复位必定发生。ATmega169V用BODLEVEL = 110做检测 ,ATmega169L用 BODLEVEL = 101 做检测。

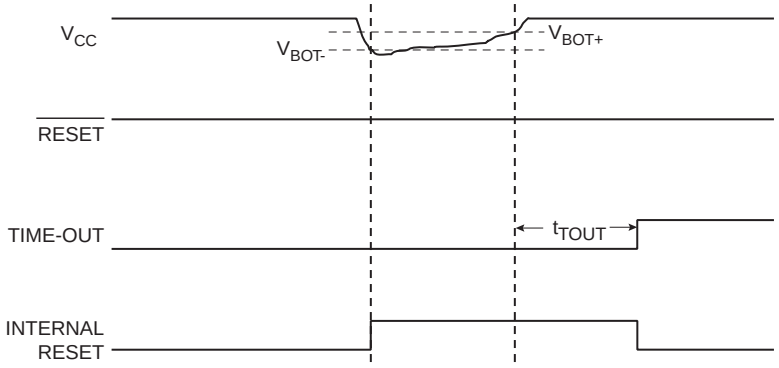
Table 18. 掉电检测特性

符号	参数	最小值	典型值	最大值	单位
V_{HYST}	掉电检测迟滞		50		mV
t_{BOD}	掉电复位最小脉宽		2		µs

当 BOD 使能后，一旦 V_{CC} 下降到触发电平以下 (V_{BOT-} ， Figure 18)， BOD 复位立即被激发。当 V_{CC} 上升到触发电平以上时 (V_{BOT+}，Figure 18)，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 16 所示的时间 t_{BOD}，BOD 电路将只检测电压跌落。

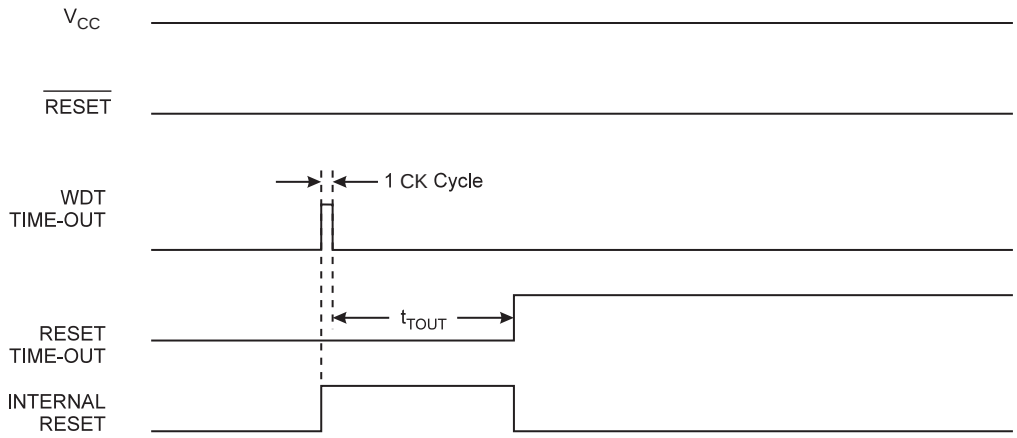
Figure 18. 工作过程中发生掉电检测复位



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 P42 以了解看门狗定时器的具体操作过程。

Figure 19. 工作过程中发生看门狗复位



MCU 状态寄存器—MCUCSR

MCU 状态寄存器提供了有关引起 MCU 复位的复位源的信息。

位	7	6	5	4	3	2	1	0	
	–	–	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUSR
读 / 写	R	R	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	参见各个位的说明					

- **位 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。

- **位 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。

- **位 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。

- **位 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。

- **位 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 MCUSR 的数据，然后将其清零。如果在其他复位发生之前将此寄存器清零，则后续复位源可以通过检查复位标志来识别。

片内基准电压

ATmega169 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。

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 19。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 BODLEVEL [2..0] 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 后要等待基准源启动之后才能使用这些功能。为了降低掉电模式的功耗，在进入掉电模式之前用户可以禁止上述三种条件以关闭基准源。

Table 19. 内部电压基准源的特性 (1)

符号	参数	条件	最小值	典型值	最大值	单位
V_{BG}	能隙基准源电压	V_{CC} = 2.7V, T_A = 25°C	1.05	1.1	1.35	V
t_{BG}	能隙基准源启动时间	V_{CC} = 2.7V, T_A = 25°C		40	70	µs
I_{BG}	能隙基准源功耗	V_{CC} = 2.7V, T_A = 25°C		15		µA

Note: 1. 上述只是参考值，实际值待定。

看门狗定时器

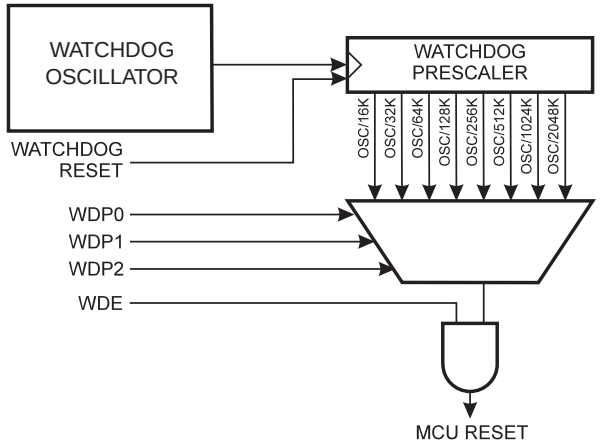
看门狗定时器由独立的 1 Mhz 片内振荡器驱动。这是 V_{CC} = 5V 时的典型值。请参见特性数据以了解其他 V_{CC} 电平下的典型值。通过设置看门狗定时器的预分频器可以调节看门狗复位的时间间隔，如 P43 Table 21 所示。看门狗复位指令 WDR 用来复位看门狗定时器。此外，禁止看门狗定时器或发生复位时它也被复位。复位时间有 8 个选项。如果没有及时复位定时器，一旦时间超过复位周期，ATmega169 就复位，并执行复位向量指向的程序。具体的看门狗复位时序在 P43 Table 21 有说明。

为了防止无意之间禁止看门狗定时器或改变了复位时间，熔丝位 WDTON 为此提供了 2 个不同的保护级别，如 Table 20. 所示。请参见 P44 “ 改变看门狗定时器配置的时间序列 ”

Table 20. 由 WDTON 熔丝位设定的 WDT 配置表

WDTON	安全等级	WDT 初始状态	如何禁止 WDT	如何改变复位时间
未编程	1	禁止	时间序列	时间序列
已编程	2	使能	总是使能	时间序列

Figure 20. 看门狗定时器



看门狗定时器控制寄存器— WDTCR

位	7	6	5	4	3	2	1	0	
	–	–	–	WDCE	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	

- **位 7..5 – Res: 保留位**

ATmega169 保留位，读操作返回值为零。

- **位 4 – WDCE: 看门狗修改使能**

清零 WDE 时必须置位 WDCE，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。修改预分频器也必须置位 WDCE，如 P44 “ 改变看门狗定时器配置的时间序列 ” 所示。

- **位 3 – WDE: 使能看门狗**

WDE为"1“时，看门狗使能，否则看门狗将被禁止。只有在WDCE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 WDCE 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”

工作于安全级别 2 时，即使使用了上述的算法，也无法禁止看门狗定时器。如 P44 “ 改变看门狗定时器配置的时间序列 ” 所示。

- **位 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，如 Table 21 所示。 .

Table 21. 看门狗定时器预分频器选项

WDP2	WDP1	WDP0	看门狗振荡器周期	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	17.1 ms	16.3 ms
0	0	1	32K	34.3 ms	32.5 ms
0	1	0	64K	68.5 ms	65 ms
0	1	1	128K	0.14 s	0.13 s
1	0	0	256K	0.27 s	0.26 s
1	0	1	512K	0.55 s	0.52 s
1	1	0	1,024K	1.1 s	1.0 s
1	1	1	2,048K	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:
  ; WDT 复位
  wdr
  ; 置位 WDCE 和 WDE
  in   r16, WDTCR
  ori  r16, (1<<WDCE)|(1<<WDE)
  out  WDTCR, r16
  ; 关闭 WDT
  ldi  r16, (0<<WDE)
  out  WDTCR, r16
  ret
```

C 代码例程 (1)

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

Note: 1. 例程假设含有所有必须的头文件。当I/O寄存器位于扩展I/O寄存器地址时 ,必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、 “OUT”、 “SBIS”、“SBIC”、 “CBI” 与 “SBI” 指令。

改变看门狗定时器配置的时间序列

改变配置的序列根据不同的安全级别略有不同。下面将分别加以描述：

安全级别 1

在此模式下看门狗定时器的初始状态是禁止的，可以通过置位 WDE 来使能它。改变定时器溢出时间及禁止（已经使能的）看门狗定时器需要执行一个特定的时间序列：

1. 在同一个指令内对 WDCE 和 WDE 写 "1“，即使 WDE 已经为 "1“
2. 在紧接的 4 个时钟周期之内将 WDE 和 WDP 设置为合适的值，而 WDCE 写 "0”

安全级别 2

在此模式下看门狗定时器总是使能的， WDE 的读返回值总是为 "1”。改变定时器溢出时间需要执行一个特定的时间序列：

1. 在同一个指令内对WDCE和WDE写"1“。虽然WDE总是为置位状态，也必须写"1“以启动时序。
2. 在紧接的 4 个时钟周期之内同时对 WDCE 写 "0”，以及为 WDP 写入合适的数据。WDE 的数值可以任意。

中断

本节描述ATmega169的中断处理。更一般的AVR中断处理请参见P12“复位与中断处理”。

ATmega169 的中断向量

Table 22. 复位和中断向量

向量号	程序地址 (2)	中断源	中断定义
1	0x0000(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	0x0002	INT0	外部中断请求 0
3	0x0004	PCINT0	引脚电平变化中断请求 0
4	0x0006	PCINT1	引脚电平变化中断请求 1
5	0x0008	TIMER2 COMP	定时器 / 计数器 2 比较匹配
6	0x000A	TIMER2 OVF	定时器 / 计数器 2 溢出
7	0x000C	TIMER1 CAPT	定时器 / 计数器 1 事件捕捉
8	0x000E	TIMER1 COMPA	定时器 / 计数器 1 比较匹配 A
9	0x0010	TIMER1 COMPB	定时器 / 计数器 1 比较匹配 B
10	0x0012	TIMER1 OVF	定时器 / 计数器 1 溢出
11	0x0014	TIMER0 COMP	定时器 / 计数器 0 比较匹配
12	0x0016	TIMER0 OVF	定时器 / 计数器 0 溢出
13	0x0018	SPI, STC	SPI 串行传输结束
14	0x001A	USART, RX	USART ， Rx 结束
15	0x001C	USART, UDRE	USART 数据寄存器空
16	0x001E	USART, TX	USART ， Tx 结束
17	0x0020	USI START	USI 起始条件
18	0x0022	USI OVERFLOW	USI 溢出
19	0x0024	ANALOG COMP	模拟比较器
20	0x0026	ADC	ADC 转换结束
21	0x0028	EE READY	EEPROM 就绪
22	0x002A	SPM READY	保存程序存储器内容就绪
23	0x002C	LCD	LCD 帧起始 (SOF)

Notes: 1. 熔丝位BOOTRST被编程时，MCU复位后程序跳转到Boot Loader。请参见P243 “Boot Loader 支持 – RWW 自编程”。
2. 寄存器MCUCR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 23 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 23. 复位和中断向量位置的确定

BOOTRST	IVSEL	复位地址	中断向量起始地址
1	0	0x0000	0x0002
1	1	0x0000	Boot 区复位地址 + 0x0002
0	0	Boot 区复位地址	0x0002
0	1	Boot 区复位地址	Boot 区复位地址 + 0x0002

Note: 1. Boot 区复位地址列于 P253 Table 113 。对于熔丝位 BOOTRST ,“1” 表示未编程，“0” 表示已编程。

ATmega169 典型的复位和中断设置如下：

```
地址              符号     代码                  说明
0x0000            jmp      RESET              ; 复位中断向量
0x0002            jmp      EXT_INT0           ; IRQ0 中断向量
0x0004            jmp      PCINT0             ; PCINT0 中断向量
0x0006            jmp      PCINT1             ; PCINT0 中断向量
0x0008            jmp      TIM2_COMP          ; 定时器 2 比较中断向量
0x000A            jmp      TIM2_OVF           ; 定时器 2 溢出中断向量
0x000C            jmp      TIM1_CAPT          ; 定时器 1 捕捉中断向量
0x000E            jmp      TIM1_COMPA         ; 定时器 1 比较 A 中断向量
0x0010            jmp      TIM1_COMPB         ; 定时器 1 比较 B 中断向量
0x0012            jmp      TIM1_OVF           ; 定时器 1 溢出中断向量
0x0014            jmp      TIM0_COMP          ; 定时器 0 比较中断向量
0x0016            jmp      TIM0_OVF           ; 定时器 0 溢出中断向量
0x0018            jmp      SPI_STC            ; SPI 传输结束中断向量
0x001A            jmp      USART_RXC          ; USART RX 结束中断向量
0x001C            jmp      USART_DRE          ; USART,UDR 空中断向量
0x001E            jmp      USART_TXC          ; USART TX 结束中断向量
0x0020            jmp      USI_STRT           ; USI 开始状态中断向量
0x0022            jmp      USI_OVFL           ; USI 溢出中断向量
0x0024            jmp      ANA_COMP           ; 模拟比较器中断向量
0x0026            jmp      ADC                ; ADC 转换结束中断向量
0x0028            jmp      EE_RDY             ; EEPROM 就绪中断向量
0x002A            jmp      SPM_RDY            ; SPM 就绪中断向量
0x002C            jmp      LCD_SOF            ; LCD 帧起始中断向量
;
0x002E   RESET:   ldi      r16, high(RAMEND); 主程序
0x002F            out      SPH,r16              设置堆栈指针为 RAM 的顶部
0x0030            ldi      r16, low(RAMEND)
0x0031            out      SPL,r16
0x0032            sei                         ;使能中断
0x0033            <instr>  xxx
  ...     ...     ...    ...
```

当熔丝位 BOOTRST 未编程，Boot 区为 2K 字节，且寄存器 MCUCR 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
0x0000   RESET:  ldi     r16,high(RAMEND) ; 主程序
0x0001           out     SPH,r16          ; 设置堆栈指针为 RAM 的顶部
0x0002           ldi     r16,low(RAMEND)
0x0003           out     SPL,r16
0x0004           sei                      ; 使能中断
0x0005           <instr>  xxx
;
.org 0x1C02
0x1C02           jmp     EXT_INT0         ; IRQ0 中断向量
0x1C04           jmp     PCINT0           ; PCINT0 中断向量
...              ...     ...              ;
0x1C2C           jmp     SPM_RDY          ; SPM 就绪中断向量
```

当熔丝位 BOOTRST 已编程，且 Boot 区为 2K 字节时，典型的复位和中断设置如下：

```
地址              符号     代码                 说明
.org 0x0002
0x0002           jmp     EXT_INT0         ; IRQ0 中断向量
0x0004           jmp     PCINT0           ; PCINT0 中断向量
...              ...     ...              ;
0x002C           jmp     SPM_RDY          ; SPM 就绪中断向量
;
.org 0x1C00
0x1C00   RESET:  ldi     r16,high(RAMEND) ; 主程序
0x1C01           out     SPH,r16          ; 设置堆栈指针为 RAM 的顶部
0x1C02           ldi     r16,low(RAMEND)
0x1C03           out     SPL,r16
0x1C04           sei                      ; 使能中断
0x1C05           <instr>  xxx
```

当熔丝位 BOOTRST 已编程，Boot 区为 2K 字节，且寄存器 MCUCR 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 0x1C00
0x1C00            jmp      RESET             ; Reset 中断向量
0x1C02            jmp      EXT_INT0          ; IRQ0 中断向量
0x1C04            jmp      PCINT0            ; PCINT0 中断向量
...               ...      ...               ;
0x1C2C            jmp      SPM_RDY           ; SPM 就绪中断向量
;
0x1C2E    RESET:  ldi      r16,high(RAMEND)  ; 主程序
0x1C2F            out      SPH,r16           ; 设置堆栈指针为 RAM 的顶部
0x1C30            ldi      r16,low(RAMEND)
0x1C31            out      SPL,r16
0x1C32            sei                        ; 使能中断
0x1C33            <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。

MCU 控制寄存器—MCUCR

位	7	6	5	4	3	2	1	0	
	JTD	–	–	PUD	–	–	IVSEL	IVCE	MCUCR
读 / 写	R/W	R	R	R/W	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 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 P243 “Boot Loader 支持 – RWW 自编程 ”。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 P243 “Boot Loader 支持 – RWW 自编程 ”。

- **位 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  MCUCR, r16
  ; 将中断向量转移到 boot 区
  ldi  r16, (1<<IVSEL)
  out  MCUCR, r16
  ret
```

C 代码例程

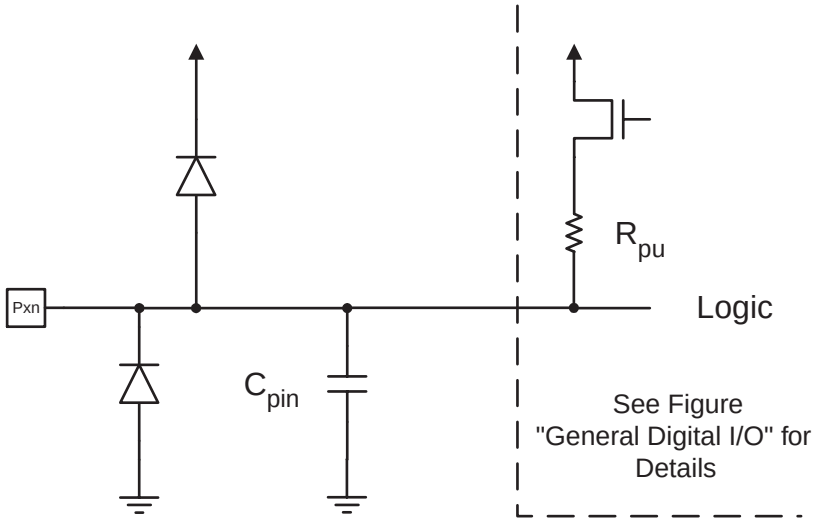
```
void Move_interrupts(void)
{
  /* 使能中断向量的修改 */
  MCUCR = (1<<IVCE);
  /* 将中断向量转移到 boot 区 */
  MCUCR = (1<<IVSEL);
}
```

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 21 所示。请参见 P286 “ 电气特性 ” 以了解完整的参数列表。

Figure 21. 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 P71 “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 MCUCR 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。

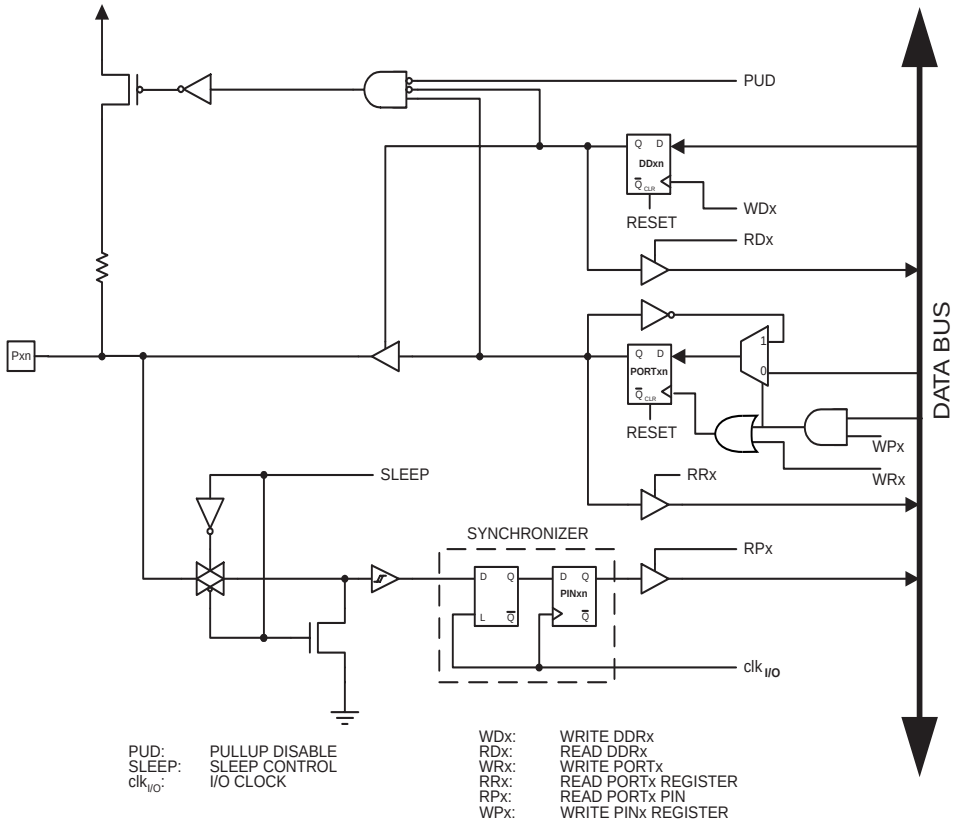
作为通用数字 I/O 时的端口请参见 P50 “ 作为通用数字 I/O 的端口 ” 。多数端口引脚是与第二功能复用的，如 P55 “ 端口的第二功能 ” 所示。请参见各个模块的具体说明以了解引脚的第二功能。

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 22 为一个 I/O 端口引脚的说明。

Figure 22. 通用数字 I/O[1]



Note: 1. WRx, 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 P71 “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。

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“)。

改变引脚电平

不论 DDRxn 的数值是多少，向 PINxn 中写逻辑 “ 1” 将使 PORTxn 的值在 “0“ 和 "1“ 之间来回变化。注意 SBI 指令能够用来改变端口的单个位。

输入与输出之间的切换

在高阻态 (三态)({DDxn, PORTxn} = 0b00) 和输出高电平 ({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，因为高阻环境并不不区分强高电平输出和上拉输出。如果实际应用环境不允许这样，则可以通过置位 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} = 0b10) 作为中间步骤。

Table 24 总结了引脚的控制信号。

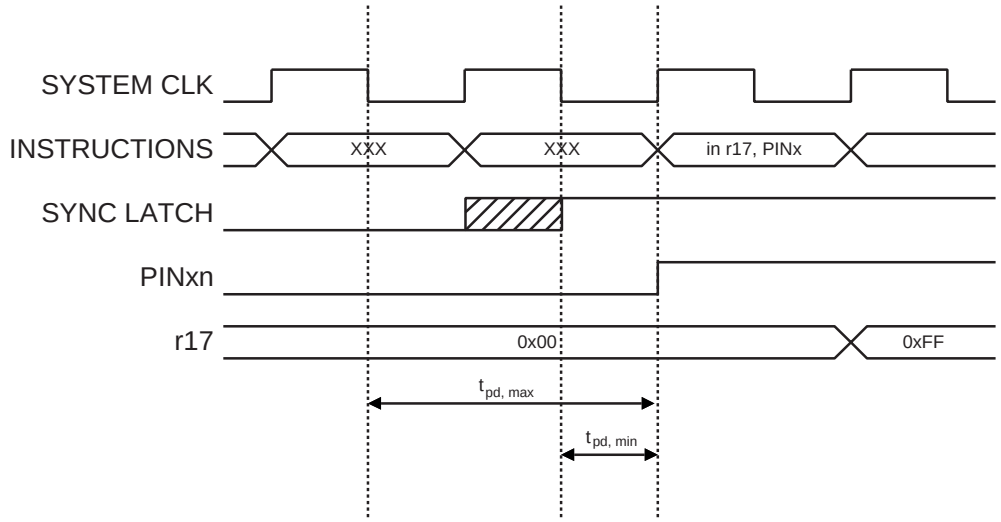
Table 24. 端口引脚配置

DDxn	PORTxn	PUD(位于 MCUCR)	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 (吸收电流)
1	1	X	输出	No	输出高电平 (输出电流)

读取引脚上的数据

不论 DDxn 如何配置，引脚电平的信息都可以通过 PINxn 寄存器来获得。如 Figure 22 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。这样就可以避免由于引脚电平在内部时钟边沿发生变化而造成的亚稳定。其缺点是引入了延迟。 Figure 23 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 $t_{pd,min}$。

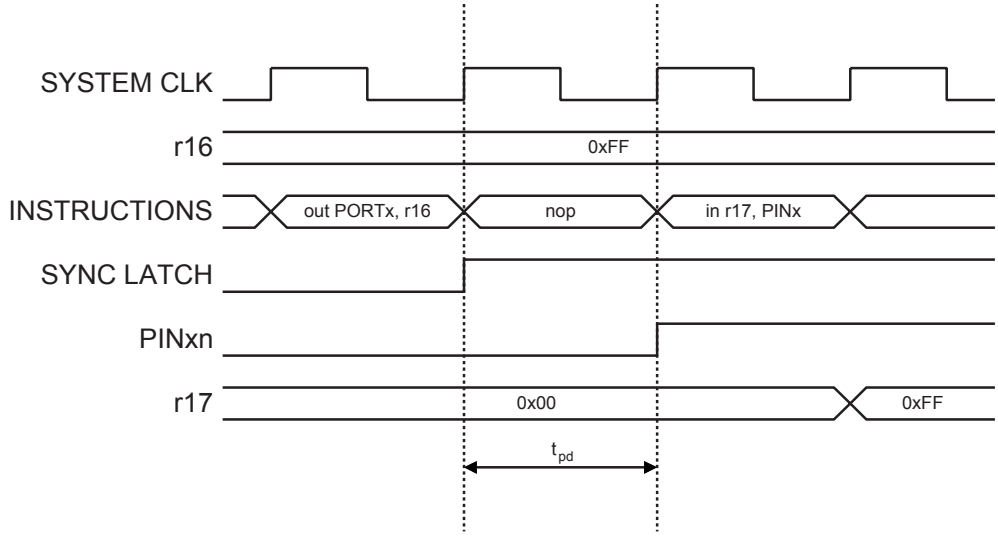
Figure 23. 读取引脚数据时的同步



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，根据信号施加时间的不同，引脚上信号转换的延迟时间界于 ½ 到 1½ 个系统时钟。

如 Figure 24. 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* 之间至少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} 为一个系统时钟周期。

Figure 24. 读取软件赋予的引脚电平的同步情况



下面的例程演示了如何置位端口 B 的引脚 0 和 1，并使引脚 2 和 3 输出低电平，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。然后程序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

```
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。

数字输入使能和休眠模式

如 Figure 22 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 P55 “ 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。

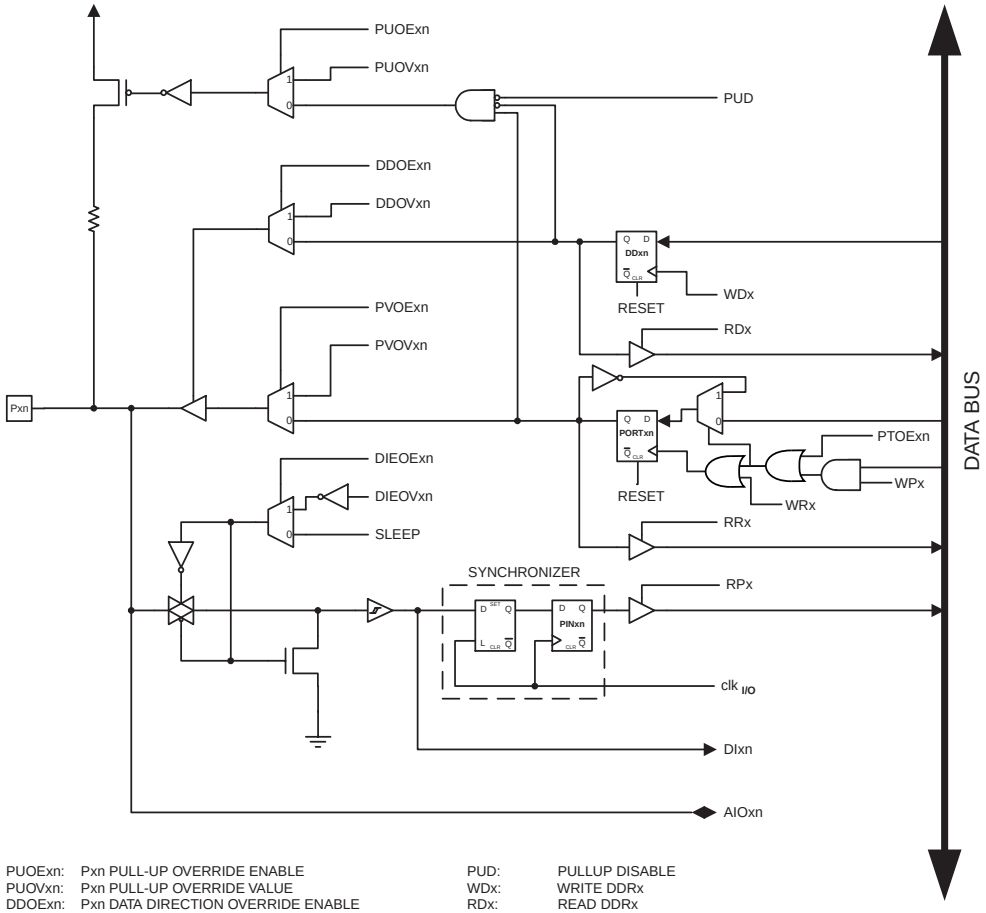
最简单的保证未用引脚具有确定电平的方法是使能内部上拉电阻。但要注意的是复位时上拉电阻将被禁用。如果复位时的功耗也有严格要求则建议使用外部上拉或下拉电阻。不

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 25 说明了由Figure 22 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 25. 端口的第二功能 (1)



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

Note: 1. WPx, WDx, RLx, 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 25 为重载信号的简介。表中没有给出 Figure 25 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 25. 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 重载使能	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 重载使能	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 重载使能	若 PVOE 置位，端口值设置为 PVOV，而不管寄存器 PORTxn 如何设置。
PTOE	端口信号切换 重载使能	若 PTOE 置位，则端口寄存器位取反。
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 使能覆盖使能	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。

MCU 控制寄存器—MCUCR

位	7	6	5	4	3	2	1	0	
	JTD	–	–	PUD	–	–	IVSEL	IVCE	MCUCR
读 / 写	R/W	R	R	R/W	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 4 – 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 P51 “ 配置引脚 ” 。

端口 A 的第二功能

端口 A 的第二功能为 LCD 控制器的 COM0:3 与 SEG0:3 。 .

Table 26. 端口 A 的第二功能

端口引脚	第二功能
PA7	SEG3 (LCD 前平面 3)
PA6	SEG2 (LCD 前平面 2)
PA5	SEG1 (LCD 前平面 1)
PA4	SEG0 (LCD 前平面 0)
PA3	COM3 (LCD 后平面 3)
PA2	COM2 (LCD 后平面 2)
PA1	COM1 (LCD 后平面 1)
PA0	COM0 (LCD 后平面 0)

Table 27 和 Table 28 将端口 A 的第二功能与 P55 Figure 25 的重载信号关联在了一起。

Table 27. PA7..PA4 的第二功能重载信号

信号名称	PA7/SEG3	PA6/SEG2	PA5/SEG1	PA4/SEG0
PUOE	LCDEN	LCDEN	LCDEN	LCDEN
PUOV	0	0	0	0
DDOE	LCDEN	LCDEN	LCDEN	LCDEN
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
PTOE	–	–	–	–
DIEOE	LCDEN	LCDEN	LCDEN	LCDEN
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	SEG3	SEG2	SEG1	SEG0

Table 28. PA3..PA0 的第二功能重载信号

信号名称	PA3/COM3	PA2/COM2	PA1/COM1	PA0/COM0
PUOE	LCDEN • (LCDMUX>2)	LCDEN • (LCDMUX>1)	LCDEN • (LCDMUX>0)	LCDEN
PUOV	0	0	0	0
DDOE	LCDEN • (LCDMUX>2)	LCDEN • (LCDMUX>1)	LCDEN • (LCDMUX>0)	LCDEN
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
PTOE	–	–	–	–
DIEOE	LCDEN • (LCDMUX>2)	LCDEN • (LCDMUX>1)	LCDEN • (LCDMUX>0)	LCDEN
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	COM3	COM2	COM1	COM0

端口 B 的第二功能

端口 B 的第二功能列于 Table 29。

Table 29. 端口 B 的第二功能

端口引脚	第二功能
PB7	OC2A/PCINT15 (T/C2 的输出比较和 PWM 输出 A，或引脚电平变化中断 15).
PB6	OC1B/PCINT14 (T/C1 的输出比较和 PWM 输出 B，或引脚电平变化中断 14).
PB5	OC1A/PCINT13 (T/C1 的输出比较和 PWM 输出 A ，或引脚电平变化中断 13).
PB4	OC0A/PCINT12 (T/C0 的输出比较和 PWM 输出 A ，或引脚电平变化中断 12).
PB3	MISO/PCINT11 (SPI 总线的 MISO 信号，或引脚电平变化中断 11).
PB2	MOSI/PCINT10 (SPI 总线的 MOSI 信号，或引脚电平变化中断 10).
PB1	SCK/PCINT9 (SPI 总线的的串行时钟，或引脚电平变化中断 9).
PB0	$\overline{SS}$/PCINT8 (SPI 从机选择引脚，或引脚电平变化中断 8).

引脚配置如下：

- **OC2A/PCINT15, 位 7**

OC2，输出比较匹配模块 A 的输出：PB7 可以作为 T/C2 输出比较模块 A 的输出。此时引脚必须配置为输出 (DDB7 设置为 “1”)。 OC2 A 引脚也是 PWM 模式的输出引脚。

PCINT15，引脚电平变化中断源 15：PB7 可以作为外部中断源。

- **OC1B/PCINT14, 位 6**

OC1B，输出比较匹配B模块的输出: PB6可以作为T/C1输出比较B模块的输出。此时引脚必须配置为输出 (DDB6 设置为 “1”)。 OC1B 也是 PWM 模式的输出引脚。

PCINT14，引脚电平变化中断源 14：PB6 可以作为外部中断源。

- **OC1A/PCINT13, 位 5**

OC1A ,输出比较匹配A模块的输出: PB5可以作为T/C1输出比较A的输出。此时引脚必须配置为输出 (DDB5 设置为 “1”)。 OC1A 也是 PWM 模式的输出引脚。

PCINT13，引脚电平变化中断源 13：PB5 可以作为外部中断源。

- **OC0A/PCINT12, 位 4**

OC0A ,输出比较匹配模块 A 的输出:PB4 可以作为 T/C0 输出比较模块 A 的输出。此时引脚必须配置为输出 (DDB4 设置为 “1”)。 OC0A 也是 PWM 模式的输出引脚。

PCINT12，引脚电平变化中断源 12：PB4 可以作为外部中断源。

- **MISO/PCINT11 – 端口 B, 位 3**

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 DDB3 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 DDB3 控制。设置为输入后，上拉电阻由 PORTB3 控制。

PCINT11，引脚电平变化中断源 11：PB3 可以作为外部中断源。

- **MOSI/PCINT10 – 端口 B, 位 2**

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 DDB2 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 DDB2 控制。设置为输入后，上拉电阻由 PORTB2 控制。

PCINT10，引脚电平变化中断源 10：PB2 可以作为外部中断源。

- **SCK/PCINT9 – 端口 B, 位 1**

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 DDB1 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 DDB1 控制。设置为输入后，上拉电阻由 PORTB1 控制。

PCINT9，引脚电平变化中断源 9：PB1 可以作为外部中断源。

- **$\overline{SS}$/PCINT8 – 端口 B, 位 0**

$\overline{SS}$: 从机选择输入。工作于从机模式时，不论DDB0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 DDB0 控制。设置为输入后，上拉电阻由 PORTB0 控制。

PCINT8，引脚电平变化中断源 8：PB0 可以作为外部中断源。

Table 30 和 Table 31 将端口 B 的第二功能与 P55 Figure 25 的重载信号关联在了一起。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 30. PB7..PB4 的第二功能重载信号

信号名称	PB7/OC2A/ PCINT15	PB6/OC1B/ PCINT14	PB5/OC1A/ PCINT13	PB4/OC0A/ PCINT12
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	OC2A ENABLE	OC1BENABLE	OC1A ENABLE	OC0A ENABLE
PVOV	OC2A	OC1B	OC1A	OC0A
PTOE	–	–	–	–
DIEOE	PCINT15 •PCIE1	PCINT14 • PCIE1	PCINT13 • PCIE1	PCINT12 •PCIE1
DIEOV	1	1	1	1
DI	PCINT15 INPUT	PCINT14 INPUT	PCINT13 INPUT	PCINT12 INPUT
AIO	–	–	–	–

Table 31. PB3..PB0 的第二功能重载信号

信号名称	PB3/MISO/PCINT11	PB2/MOSI/PCINT10	PB1/SCK/PCINT9	PB0/$\overline{SS}$/PCINT8
PUOE	SPE • MSTR	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$
PUOV	PORTB3 • $\overline{PUD}$	PORTB2 • $\overline{PUD}$	PORTB1 • $\overline{PUD}$	PORTB0 • $\overline{PUD}$
DDOE	SPE • MSTR	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$
DDOV	0	0	0	0
PVOE	SPE • $\overline{MSTR}$	SPE • MSTR	SPE • MSTR	0
PVOV	SPI SLAVE OUTPUT	SPI MSTR OUTPUT	SCK OUTPUT	0
PTOE	–	–	–	–
DIEOE	PCINT11 • PCIE1	PCINT10 • PCIE1	PCINT9 • PCIE1	PCINT8 •PCIE1
DIEOV	1	1	1	1
DI	PCINT11 INPUT SPI MSTR INPUT	PCINT10 INPUT SPI SLAVE INPUT	PCINT9 INPUT SCK INPUT	PCINT8 INPUT SPI $\overline{SS}$
AIO	–	–	–	–

端口 C 的第二功能

端口 C 的第二功能为 LCD 控制器的 SEG5:12。

Table 32. 端口 C 的第二功能

端口引脚	第二功能
PC7	SEG5 (LCD 前平面 5)
PC6	SEG6 (LCD 前平面 6)
PC5	SEG7 (LCD 前平面 7)
PC4	SEG8 (LCD 前平面 8)
PC3	SEG9 (LCD 前平面 9)
PC2	SEG10 (LCD 前平面 10)
PC1	SEG11 (LCD 前平面 11)
PC0	SEG12 (LCD 前平面 12)

Table 33 和 Table 34 将端口 C 的第二功能与 P55 Figure 25 的重载信号关联在了一起。.

Table 33. PC7..PC4 的第二功能重载信号

信号名称	PC7/SEG5	PC6/SEG6	PC5/SEG7	PC4/SEG8
PUOE	LCDEN	LCDEN	LCDEN	LCDEN
PUOV	0	0	0	0
DDOE	LCDEN	LCDEN	LCDEN	LCDEN
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
PTOE	–	–	–	–
DIEOE	LCDEN	LCDEN	LCDEN	LCDEN
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	SEG5	SEG6	SEG7	SEG8

Table 34. PC3..PC0 的第二功能重载信号

信号名称	PC3/SEG9	PC2/SEG10	PC1/SEG11	PC0/SEG12
PUOE	LCDEN	LCDEN	LCDEN	LCDEN
PUOV	0	0	0	0
DDOE	LCDEN	LCDEN	LCDEN	LCDEN
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
PTOE	–	–	–	–
DIEOE	LCDEN	LCDEN	LCDEN	LCDEN
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	SEG9	SEG10	SEG11	SEG12

端口 D 的第二功能

端口 D 的第二功能列于 Table 35.

Table 35. 端口 D 的第二功能

端口引脚	第二功能
PD7	SEG15 (LCD 前平面 15)
PD6	SEG16 (LCD 前平面 16)
PD5	SEG17 (LCD 前平面 17)
PD4	SEG18 (LCD 前平面 18)
PD3	SEG19 (LCD 前平面 19)
PD2	SEG20 (LCD 前平面 20)
PD1	INT0/SEG21 (外部中断 0 输入，或 LCD 前平面 21)
PD0	ICP1/SEG22 (T/C1 输入捕捉的触发引脚，或 LCD 前平面 22)

第二功能配置如下：

- **SEG15 - SEG20 – 端口 D, 位 7:2**

SEG15-SEG20, LCD 前平面 15-20。

- **INT0/SEG21 – 端口 D, 位 1**

INT0，外部中断 0。 PD1 引脚作为 MCU 的外部中断源。

SEG21， LCD 前平面 21。

- **ICP1/SEG22 – 端口 D, 位 0**

ICP1 – 输入捕捉引脚 1：PD0 作为 T/C1 的输入捕捉引脚。

SEG22， LCD 前平面 22。

Table 36 和 Table 37 将端口 D 的第二功能与 P55 Figure 25 的重载信号关联在了一起。

Table 36. PD7..PD4 第二功能重载信号

信号名称	PD7/SEG15	PD6/SEG16	PD5/SEG17	PD4/SEG18
PUOE	LCDEN • (LCDPM>1)	LCDEN • (LCDPM>1)	LCDEN • (LCDPM>2)	LCDEN • (LCDPM>2)
PUOV	0	0	0	0
DDOE	LCDEN • (LCDPM>1)	LCDEN • (LCDPM>1)	LCDEN • (LCDPM>2)	LCDEN • (LCDPM>2)
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
PTOE	–	–	–	–
DIEOE	LCDEN • (LCDPM>1)	LCDEN • (LCDPM>1)	LCDEN • (LCDPM>2)	LCDEN • (LCDPM>2)
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	SEG15	SEG16	SEG17	SEG18

Table 37. PD3..PD0 第二功能重载信号

信号名称	PD3/SEG19	PD2/SEG20	PD1/INT0/SEG21	PD0/ICP1/SEG22
PUOE	LCDEN • (LCDPM>3)	LCDEN • (LCDPM>3)	LCDEN • (LCDPM>4)	LCDEN • (LCDPM>4)
PUOV	0	0	0	0
DDOE	LCDEN • (LCDPM>3)	LCDEN • (LCDPM>3)	LCDEN • (LCDPM>4)	LCDEN • (LCDPM>4)
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
PTOE	–	–	–	–
DIEOE	LCDEN • (LCDPM>3)	LCDEN • (LCDPM>3)	LCDEN + (INT0 ENABLE)	LCDEN • (LCDPM>4)
DIEOV	0	0	$\overline{\text{LCDEN}}$ • (INT0 ENABLE)	0
DI	–	–	INT0 INPUT	ICP1 INPUT
AIO	–	–		

端口 E 的第二功能

端口 E 的第二功能列于 Table 38.

Table 38. 端口 E 的第二功能

引脚	第二功能
PE7	PCINT7 (引脚电平变化中断 7) CLKO (分频后的系统时钟)
PE6	DO/PCINT6 (USI 数据输出或引脚电平变化中断 6)
PE5	DI/SDA/PCINT5 (USI 数据输入，或 TWI 串行数据，或引脚电平变化中断 5)
PE4	USCK/SCL/PCINT4 (USART 外部时钟输入 / 输出 ,或 TWI 串行时钟 ,或引脚电平变化中断 4)
PE3	AIN1/PCINT3 (模拟比较器负输入端，或引脚电平变化中断 3)
PE2	XCK/AIN0/ PCINT2 (USART 外部时钟，或模拟比较器正输入端，或引脚电平变化中断 2)
PE1	TXD/PCINT1 (USART 发送引脚，或引脚电平变化中断 1)
PE0	RXD/PCINT0 (USART 接收引脚或 引脚电平变化中断 0)

- **PCINT7 – 端口 E, 位 7**

PCINT7，引脚电平变化中断源 7：PE7 可以作为外部中断源。

CLKO，分频后的系统时钟，可以从 PE7 引脚输出。一旦熔丝位 CKOUT 被编程，则不论 PORTE7 和 DDE7 的设置如何，分频后的系统时钟都将从 PE7 引脚输出。即使在复位期间此也是如此。

- **DO/PCINT6 – 端口 E, 位 6**

DO，通用串行接口 USI 的数据输出端口。

PCINT6，引脚电平变化中断源 6：PE6 可以作为外部中断源。

- **DI/SDA/PCINT5 – 端口 E, 位 5**

DI，通用串行接口 USI 的数据输入端口。

SDA，两线串行接口数据。

PCINT5，引脚电平变化中断源 5：PE5 可以作为外部中断源。

- **USCK/SCL/PCINT4 – 端口 E, 位 4**

USCK，通用串行接口 USI 的时钟。

SCL，两线串行接口时钟。

PCINT4，引脚电平变化中断源 4：PE4 可以作为外部中断源。

- **AIN1/PCINT3 – 端口 E, 位 3**

AIN1 – 模拟比较器负输入端。该引脚直接与模拟比较器负输入端相连接。

PCINT3，引脚电平变化中断源 3：PE3 可以作为外部中断源。

- **XCK/AIN0/PCINT2 – 端口 E, 位 2**

XCK ,USART外部时钟。数据方向寄存器的DDE2控制这个时钟是输入时钟 (DDE2 为”0”) 还是输出时钟 (DDE2 为 ”1”)。只有当 USART 工作于同步模式时 XCK 才会生效。

AIN0 – 模拟比较器正输入端。该引脚直接与模拟比较器正输入端相连接。

PCINT2，引脚电平变化中断源 2：PE2 可以作为外部中断源。

- **TXD/PCINT1 – 端口 E, 位 1**

TXD0， UART0 发送引脚。

PCINT1，引脚电平变化中断源 1：PE1 可以作为外部中断源。

- **RXD/PCINT0 – 端口 E, 位 0**

RXD ,USART 接收引脚。接收数据 (USART 数据输入引脚)。使能 USART 接收器后这个引脚配置为输入而不管 DDE0 的设置如何。 PORTE0 仍然控制着上拉电阻的使能。

PCINT0，引脚电平变化中断源 0：PE0 可以作为外部中断源。

Table 39 和 Table 40 将端口 E 的第二功能与 P55 Figure 25 的重载信号关联在了一起。

Table 39. PE7..PE4 的第二功能重载信号

信号名称	PE7/PCINT7	PE6/DO/ PCINT6	PE5/DI/SDA/ PCINT5	PE4/USCK/SCL/ PCINT4
PUOE	0	0	USI_TWO-WIRE	0
PUOV	0	0	0	0
DDOE	CKOUT(1)	0	USI_TWO-WIRE	USI_TWO-WIRE
DDOV	1	0	($\overline{SDA}$ + $\overline{PORTE5}$) • DDE5	(USI_SCL_HOLD + $\overline{PORTE4}$) + DDE4
PVOE	CKOUT(1)	USI_THREE-WIRE	USI_TWO-WIRE • DDE5	USI_TWO-WIRE • DDE4
PVOV	$clk_{I/O}$	DO	0	0
PTOE	–	–	–	USITC
DIEOE	PCINT7 • PCIE0	PCINT6 • PCIE0	(PCINT5 • PCIE0) + USISIE	(PCINT4 • PCIE0) + USISIE
DIEOV	1	1	1	1
DI	PCINT7 INPUT	PCINT6 INPUT	DI/SDA INPUT PCINT5 INPUT	USCKL/SCL INPUT PCINT4 INPUT
AIO	–	–	–	–

Note: 1. 如果 CKOUT 熔丝被编程，则 CKOUT 为 "1”。

Table 40. PE3..PE0 的第二功能重载信号

信号名称	PE3/AIN1/ PCINT3	PE2/XCK/AIN0/ PCINT2	PE1/TXD/ PCINT1	PE0/RXD/PCINT0
PUOE	0	0	TXEN	RXEN
PUOV	0	0	0	PORTE0 • $\overline{PUD}$
DDOE	0	0	TXEN	RXEN
DDOV	0	0	1	0
PVOE	0	XCK 输出使能	TXEN	0
PVOV	0	XCK	TXD	0
PTOE	–	–	–	–
DIEOE	(PCINT3 • PCIE0) + AIN1D(1)	(PCINT2 • PCIE0) + AIN0D(1)	PCINT1 • PCIE0	PCINT0 • PCIE0
DIEOV	PCINT3 • PCIE0	PCINT2 • PCIE0	1	1
DI	PCINT3 INPUT	XCK/PCINT2 INPUT	PCINT1 INPUT	RXD/PCINT0 INPUT
AIO	AIN1 INPUT	AIN0 INPUT	–	–

Note: 1. AIN0D 与 AIN1D 在 P183 “ 数字输入禁止寄存器 1 – DIDR1” 有描述。

端口 F 的第二功能

如 Table 41 所示 ,端口 F 的第二功能是 ADC 输入。如果端口 F 的一些引脚配置为输出，很重要的一点是在 AD 转换过程中不要改变输出引脚的电平，否则会造成转换结果不正确。如果使能了 JTAG 接口，则即使在复位阶段 PF7(TDI)、 PF5(TMS) 和 PF4(TCK) 的上拉电阻仍然有效。

Table 41. 端口 F 的第二功能

端口引脚	第二功能
PF7	ADC7/TDI (ADC 输入通道 7，或是 JTAG 测试数据输入引脚)
PF6	ADC6/TDO (ADC 输入通道 6，或是 JTAG 测试数据输出引脚)
PF5	ADC5/TMS (ADC 输入通道 5 ，或是 JTAG 测试模式选择引脚)
PF4	ADC4/TCK (ADC 输入通道 4，或是 JTAG 测试时钟)
PF3	ADC3 (ADC 输入通道 3)
PF2	ADC2 (ADC 输入通道 2)
PF1	ADC1 (ADC 输入通道 1)
PF0	ADC0 (ADC 输入通道 0)

- **TDI, ADC7 – 端口 F, 位 7**

ADC7，模数转换器通道 7。

TDI , JTAG 测试数据输入引脚 : 将要移入指令寄存器或数据寄存器 (扫描链) 的串行输入数据。使能 JTAG 接口之后这个引脚不能再用作普通 I/O 口。

- **TDO, ADC6 – 端口 F, 位 6**

ADC6，模数转换器通道 6。

TDO，JTAG 测试数据输出引脚：将要移出指令寄存器或数据寄存器的串行输出数据。使能 JTAG 接口之后这个引脚不能再用作普通 I/O 口。在移出数据的 TAP 状态下 TDO 引脚是活动的。其它状态下该引脚被拉高。

- **TMS, ADC5 – 端口 F, 位 5**

ADC5，模数转换器通道 5。

TMS ,JTAG 测试模式选择引脚。这个引脚用于 TAP 控制器状态机 的定位。使能 JTAG 接口之后这个引脚不能再用作普通 I/O 口。

- **TCK, ADC4 – 端口 F, 位 4**

ADC4，模数转换器通道 4。

TCK ,JTAG 测试时钟:JTAG 的操作相对 TCK 是同步的。使能 JTAG 接口之后这个引脚不能再用作普通 I/O 口。

- **ADC3 - ADC0 – 端口 F, 位 3:0**

模数转换器通道 3..0。

Table 42. PF7..PF4 的第二功能重载信号

信号名称	PF7/ADC7/TDI	PF6/ADC6/TDO	PF5/ADC5/TMS	PF4/ADC4/TCK
PUOE	JTAGEN	JTAGEN	JTAGEN	JTAGEN
PUOV	1	1	1	1
DDOE	JTAGEN	JTAGEN	JTAGEN	JTAGEN
DDOV	0	SHIFT_IR + SHIFT_DR	0	0
PVOE	0	JTAGEN	0	0
PVOV	0	TDO	0	0
PTOE	–	–	–	–
DIEOE	JTAGEN	JTAGEN	JTAGEN	JTAGEN
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	TDI ADC7 INPUT	ADC6 INPUT	TMS ADC5 INPUT	TCK ADC4 INPUT

Table 43. PF3..PF0 第二功能的重载信号

信号名称	PF3/ADC3	PF2/ADC2	PF1/ADC1	PF0/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
PTOE	–	–	–	–
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	ADC3 INPUT	ADC2 INPUT	ADC1 INPUT	ADC0 INPUT

端口 G 的第二功能

第二功能介绍如下：

Table 44. 端口 G 的第二功能

端口引脚	第二功能
PG4	T0/SEG23 (T/C0 时钟输入，或 LCD 前平面 23)
PG3	T1/SEG24 (T/C1 时钟输入，或 LCD 前平面 24)
PG2	SEG4 (LCD 前平面 4)
PG1	SEG13 (LCD 前平面 13)
PG0	SEG14 (LCD 前平面 14)

第二功能介绍如下：

- **T0/SEG23 – 端口 G, 位 4**

T0， T/C0 计数器时钟源。

SEG23， LCD 前平面 23。

- **T1/SEG24 – 端口 G, 位 3**

T1， T/C1 计数器时钟源。

SEG24， LCD 前平面 24。

- **SEG4 – 端口 G, 位 2**

SEG4， LCD 前平面 4。

- **SEG13 – 端口 G, 位 1**

SEG13，字段驱动 13。

- **SEG14 – 端口 G, 位 0**

SEG14，LCD 前平面 14。

Table 44 和 Table 45 将端口 G 的第二功能与 P55 Figure 25 的重载信号关联在了一起。.

Table 45. PG4 的第二功能重载信号

信号名称				PG4/T0/SEG23
PUOE				LCDEN • (LCDPM>5)
PUOV				0
DDOE				LCDEN • (LCDPM>5)
DDOV				1
PVOE				0
PVOV				0
PTOE	–	–	–	–
DIEOE				LCDEN • (LCDPM>5)
DIEOV				0
DI				T0 INPUT
AIO				SEG23

Table 46. PG3:0 的第二功能重载信号

信号名称	PG3/T1/SEG24	PG2/SEG4	PG1/SEG13	PG0/SEG14
PUOE	LCDEN • (LCDPM>6)	LCDEN	LCDEN • (LCDPM>0)	LCDEN • (LCDPM>0)
PUOV	0	0	0	0
DDOE	LCDEN • (LCDPM>6)	LCDEN	LCDEN • (LCDPM>0)	LCDEN • (LCDPM>0)
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
PTOE	–	–	–	–
DIEOE	LCDEN • (LCDPM>6)	LCDEN	LCDEN • (LCDPM>0)	LCDEN • (LCDPM>0)
DIEOV	0	0	0	0
DI	T1 INPUT	–	–	–
AIO	SEG24	SEG4	SEG13	SEG14

I/O 端口寄存器的说明

端口 A 数据寄存器—PORTA

位	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

位	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

位	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
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

位	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

位	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

位	7	6	5	4	3	2	1	0	
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
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

位	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

位	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

位	7	6	5	4	3	2	1	0	
	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
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

位	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

位	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

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

端口 E 数据寄存器—PORTE

位	7	6	5	4	3	2	1	0	
	PORTE7	PORTE6	PORTE5	PORTE4	PORTE3	PORTE2	PORTE1	PORTE0	PORTE
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 E 数据方向寄存器—DDRE

位	7	6	5	4	3	2	1	0	
	DDE7	DDE6	DDE5	DDE4	DDE3	DDE2	DDE1	DDE0	DDRE
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 E 输入引脚地址—PINE

位	7	6	5	4	3	2	1	0	
	PINE7	PINE6	PINE5	PINE4	PINE3	PINE2	PINE1	PINE0	PINE
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

端口 F 数据寄存器—PORTF

位	7	6	5	4	3	2	1	0	
	PORTF7	PORTF6	PORTF5	PORTF4	PORTF3	PORTF2	PORTF1	PORTF0	PORTF
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 F 数据方向寄存器—DDRF

位	7	6	5	4	3	2	1	0	
	DDF7	DDF6	DDF5	DDF4	DDF3	DDF2	DDF1	DDF0	DDRF
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 F 输入引脚地址—PINF

位	7	6	5	4	3	2	1	0	
	PINF7	PINF6	PINF5	PINF4	PINF3	PINF2	PINF1	PINF0	PINF
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

端口 G 数据寄存器—PORTG

位	7	6	5	4	3	2	1	0	
	–	–	–	PORTG4	PORTG3	PORTG2	PORTG1	PORTG0	PORTG
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	

端口 G 数据方向寄存器—DDRG

位	7	6	5	4	3	2	1	0	
	–	–	–	DDG4	DDG3	DDG2	DDG1	DDG0	DDRG
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	

端口 G 输入引脚地址—PING

位	7	6	5	4	3	2	1	0	
	–	–	–	PING4	PING3	PING2	PING1	PING0	PING
读 / 写	R	R	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	N/A	N/A	N/A	N/A	N/A	

外部中断

外部中断通过引脚 INT0 或 PCINT15..0 触发。只要使能了中断，即使引脚 INT0 或 PCINT15..0 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。只要使能， PCINT15..8 引脚上的电平变化将触发外部中断 PCI1，PCINT7..0 将触发外部中断 PCI0。PCMSK1 与 PCMSK0 寄存器则用来检测是哪个引脚上的电平发生了变化。 PCINT15..0 外部中断的检测是异步的。也就是说，和其他中断方式一样，这些中断也可以用来将器件从休眠模式唤醒。

INT0 中断可以由下降沿、上升沿，或者是低电平触发。具体由外部中断控制寄存器 A – EICRA 的设置来确定。当 INT0 中断使能且设定为电平触发时，只要引脚电平被拉低，中断就会产生。若要求 INT0 在信号下降沿或上升沿触发中断，则 I/O 时钟必须工作 (请参见 P23 “ 时钟系统及其分布 ” 了解更多信息)。INT0 的低电平中断检测是异步的。也就是说它可以用来将器件从休眠模式唤醒。在休眠过程 (除了空闲模式) 中 I/O 时钟是停止的。

通过电平中断将 MCU 从掉电模式唤醒时，要保证低电平保持一定的时间以使 MCU 完成唤醒过程并触发中断。如果触发电平在启动时间结束前就消失， MCU 将被唤醒，但中断不会被触发。启动时间由熔丝位 SUT 与 CKSEL 决定。详见 P23“ 系统时钟与时钟选择 ”。

外部中断控制寄存器 A—EICRA

外部中断控制寄存器 A 包括决定中断触发方式的控制位。

位	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	ISC01	ISC00	EICRA
读 / 写	R	R	R	R	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 1, 0 – ISC01, ISC00: 中断触发方式控制 0 之位 1 与位 0**

外部中断 0 由引脚 INT0 激发，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 47 所示。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 47. 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

外部中断屏蔽寄存器—EIMSK

位	7	6	5	4	3	2	1	0	
	PCIE1	PCIE0	–	–	–	–	–	INT0	EIMSK
读 / 写	R/W	R/W	R	R	R	R	R	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7 – PCIE1: 引脚电平变化中断使能 1**

当 PCIE1 为 ”1” , 而且状态寄存器 SREG 的 I 置位 , 引脚电平变化中断 1 就使能了。被使能的引脚 PCINT15..8 上的任何电平变化都会触发中断。相应的外部中断向量为 PCI1 中断向量。 PCINT15..8 由 PCMSK1 寄存器控制是否使能。

- **位 6 – PCIE0: 引脚电平变化中断使能 0**

当 PCIE0 为 ”1” , 而且状态寄存器 SREG 的 I 置位 , 引脚电平变化中断 0 就使能了。被使能的引脚 PCINT7..0 上的任何电平变化都会触发中断。相应的外部中断向量为 PCI0 中断向量。 PCINT7..0 由 PCMSK0 寄存器控制是否使能。

- **位 0 – 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。外部中断控制寄存器 – EICRA 的中断触发方式控制位决定中断是由 INT0 上升沿、下降沿，还是电平触发的。使能之后，即使引脚 INT0 被配置为输出，只要引脚电平发生了相应的变化，中断就会产生。相应的中断向量为 INT0 中断向量。

外部中断标志寄存器—EIFR

位	7	6	5	4	3	2	1	0	
	PCIF1	PCIF0	–	–	–	–	–	INTF0	EIFR
读 / 写	R/W	R/W	R	R	R	R	R	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7 – PCIF1: 引脚电平变化中断标志 1**

PCINT15..8 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 PCIF1。如果SREG的位I以及EIMSK寄存器相应的中断使能位PCIE1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。

- **位 6 – PCIF0: 引脚电平变化中断标志 0**

PCINT7..0 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 PCIF0。如果SREG的位I以及EIMSK寄存器相应的中断使能位PCIE0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。

- **位 0 – 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 EIMSK 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。若 INTF0 配置为电平触发，则 INTF0 一直为零。

引脚电平变化屏蔽寄存器 1—PCMSK1

位	7	6	5	4	3	2	1	0	
	PCINT15	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
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	

- **位 7..0 – PCINT15..8: 引脚电平变化使能屏蔽 15..8**

PCINT15..8 中的每一位决定相应的 I/O 引脚电平变化中断是否使能。如果 PCINT15..8 与 EIMSK 上的 PCIE1 位置位 ,则相应的引脚电平变化中断使能。如果 PCINT15..8 清零 ,相应的引脚电平变化中断禁用。

引脚变化屏蔽寄存器 0— PCMSK0

位	7	6	5	4	3	2	1	0	
	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
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	

- **位 7..0 – PCINT7..0: 引脚电平变化使能屏蔽 7..0**

PCINT7..0 中的每一位决定相应的 I/O 引脚电平变化中断是否使能。如果 PCINT7..0 与 EIMSK 上的 PCIE0 位置位 ,则相应的引脚电平变化中断使能。如果 PCINT7..0 清零 ,相应引脚电平变化中断禁用。

具备 PWM 功能的 8 位定时器 / 计时器 0

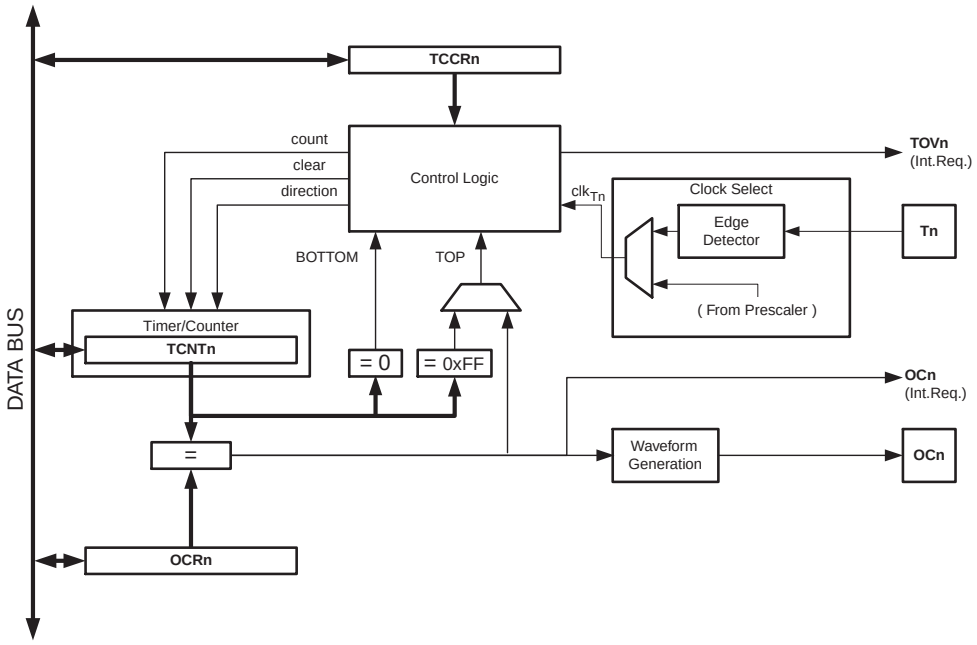
T/C0 是一个通用的单通道 8 位定时器 / 计数器模块。其主要特点如下：

- **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 位时钟预分频器**
- **溢出和比较匹配中断源 (TOV0 和 OCF0A)**

综述

Figure 26为8位定时器/计数器的简化框图。实际引脚排列请参考P2 “ATmega169引脚排列 ” 。 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 P86 “8 位定时器 / 计数器寄存器的说明 ” 。

Figure 26. 8 位 T/C 方框图



寄存器

T/C(TCNT2)和输出比较寄存器(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 TIFR0 都有反映。所有中断都可以通过定时器中断屏蔽寄存器 TIMSK0 单独进行屏蔽。 TIFR0 和 TIMSK0 在图中没有 表示。

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 OCR0A 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OC0A上产生变化频率的输出，如P79“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OCF0A。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。小写的 “x” 取代了输出比较单元通道，在此即为通道 A。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 48 的定义适用于全文。

Table 48. 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 OCR0A 里的数值，具体由工作模式确定

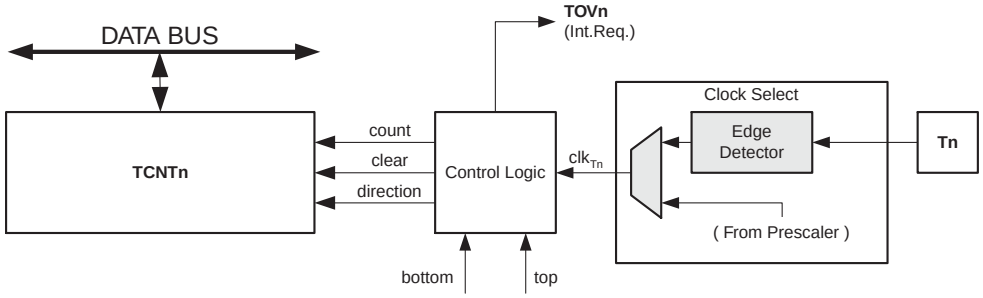
T/C 的时钟源

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 TCCR0A 的时钟选择位 CS02:0 控制的。 P90 “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 27 即为计数器和它周边电路的方框图。

Figure 27. 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 (TCCR0A) 的 WGM01 和 WGM00 决定。计数器计数行为与输出比较 OC0A 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 P82 “工作模式 ”。

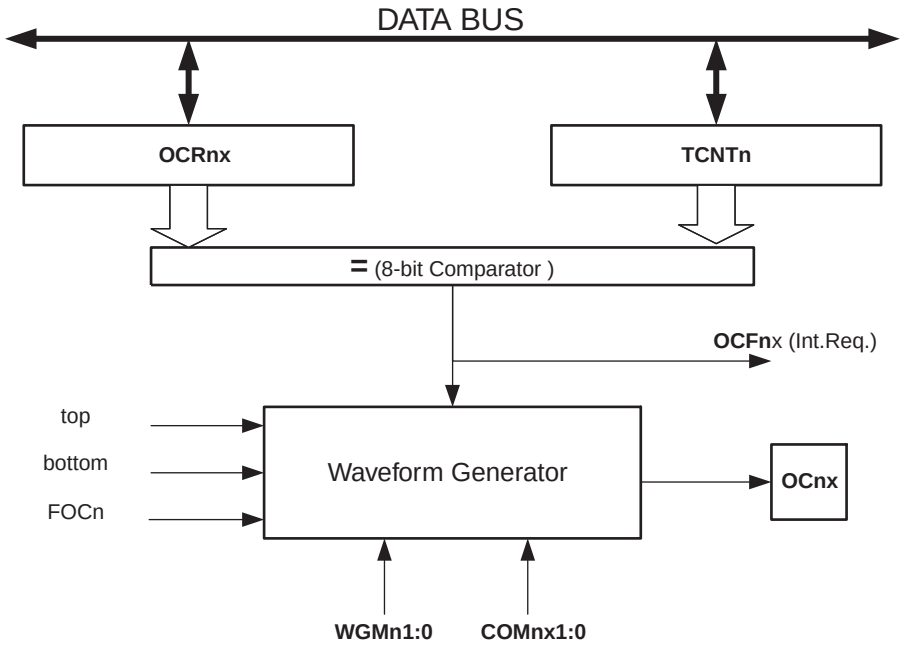
T/C溢出中断标志TOV0根据WGM01:0 设定的工作模式来设置。TOV0可以用于产生CPU中断。

输出比较单元

8位比较器持续对TCNT0和输出比较寄存器OCR0A进行比较。一旦TCNT0等于OCR0A，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 OCF0A 置位。若此时 OCIE0A = 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 OCF0A 自动清零，或者通过软件写 ”1” 的方式来清零。根据由 WGM21:0 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 (P82 “工作模式 ”)。

Figure 28 为输出比较单元的方框图。

Figure 28. 输出比较单元方框图



使用 PWM 模式时 OCR0A 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OCR0A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OCR0A缓冲寄存器；禁止双缓冲功能时 CPU 访问的则是 OCR0A 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 FOC0A 写 ”1” 的方式来产生比较匹配。强制比较匹配不会置位 OCF0A 标志，也不会重载 / 清零定时器，但是 OC0A 引脚将被更新，好象真的发生了比较匹配一样 (COM0A1:0 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 OCR0A 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 OCR0A，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。

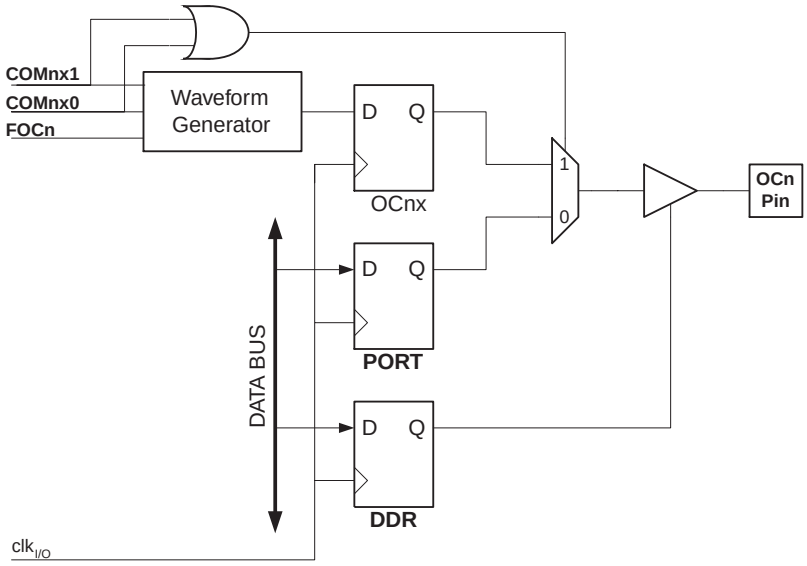
OC0A 的设置应该在设置数据方向寄存器之前完成。最简单的设置 OC0A 的方法是在普通模式下利用强制输出比较 FOC0A。即使在改变波形发生模式时 OC0A 寄存器也会一直保持它的数值。

注意 COM0A1:0 和比较数据都不是双缓冲的。 COM0A1:0 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 COM0A1:0 具有双重功能。波形发生器利用 COM0A1:0 来确定下一次比较匹配发生时的输出比较状态 (OC0A) ； COM0A1:0 还控制 OC0A 引脚输出信号的来源。 Figure 29 为受 COM0A1:0 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 COM0A1:0 影响的通用 I/O 端口控制寄存器 (DDR 和 PORT)。谈及 OC0A 状态时指的是内部 OC0A 寄存器，而不是 OC0A 引脚。系统复位时 OC0A 寄存器清零。

Figure 29. 比较匹配输出单元原理图



如果 COM0A1:0 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OC0A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OC0A功能之前首先要通过数据方向寄存器的 DDR_OC0A 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OC0A状态在输出之前首先进行初始化。要注意某些COM0A1:0设置保留给了其他操作类型，详见 P86 “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 COM0A1:0 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 COM0A1:0 = 0 表明比较匹配发生时波形发生器不会操作 OC0A 寄存器。非 PWM 模式的比较输出请参见 P87 Table 50 ；快速 PWM 的比较输出示于 P87 Table 51 ；相位修正 PWM 的比较输出在 P87 Table 52 有描述。

改变 COM0A1:0 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 FOC0A 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 (COM0A1:0) 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。 COM0A1:0 控制 PWM 输出是否为反极性。非 PWM 模式时 COM0A1:0 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 (P81“ 比较匹配输出单元 ”)。

具体的时序信息请参考 P85“T/C 时序图 ” 之 Figure 33，Figure 34，Figure 35 与 Figure 36。

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。

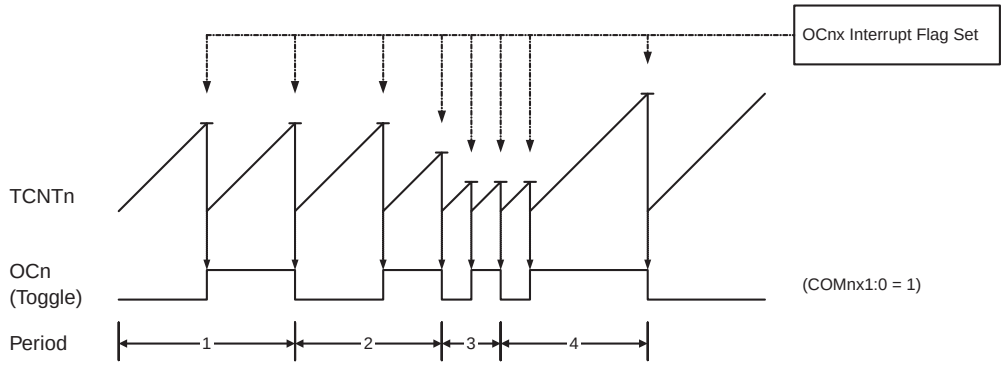
输出比较单元可以用来产生中断。但是不推荐在普通模式下利用输出比较来产生波形，因为这会占用太多的 CPU 时间。

CTC(比较匹配时清零定时器) 模式

在 CTC 模式 (WGM01:0 = 2) 下 OCR0A 寄存器用于调节计数器的分辨率。当计数器的数值 TCNT0 等于 OCR0A 时计数器清零。OCR0A 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 30。计数器数值TCNT0一直累加到TCNT0与OCR0A匹配，然后 TCNT0 清零。

Figure 30. CTC 模式的时序图



利用 OCF0A 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 OCR0A 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 OCF0A。

为了在CTC模式下得到波形输出，可以设置OC0A在每次比较匹配发生时改变逻辑电平。这可以通过设置 COM0A1:0 = 1 来完成。在期望获得 OC0A 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：

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

变量 N 代表预分频因子 (1、 8、 64、或 1024)。

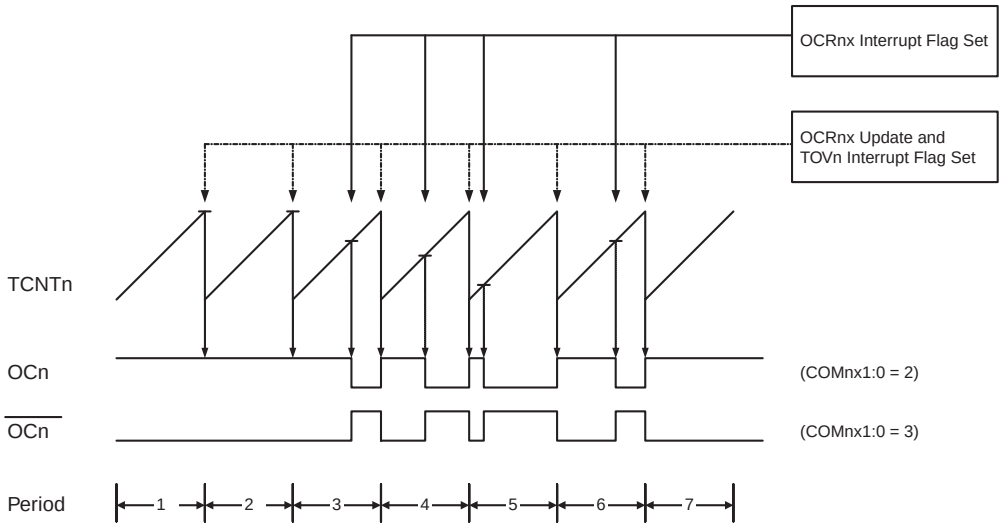
在普通模式下，TOV0 标志的置位发生在计数器从 MAX 变为 0x00 的定时器时钟周期。

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OC0A在TCNT0与OCR0A 匹配时清零，在 BOTTOM 时置位；对于反向比较输出模式，OC0A 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 31。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 OCR0A 和 TCNT0 的比较匹配。

Figure 31. 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OC0A引脚上输出PWM波形。设置COM0A1:0 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 P87 Table 51)。要想在引脚上得到输出信号还必须将 OC0A 的数据方向设置为输出。产生 PWM 波形的机理是 OC0A 寄存器在 OCR0A 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：

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

变量 N 代表分频因子 (1、8、64、256 或 1024)。

OCR0A 寄存器为极限值时表示快速 PWM 模式的一些特殊情况。若 OCR0A 等于 BOTTOM，输出为出现在第MAX+1个定时器时钟周期的窄脉冲；OCR0A为MAX时，根据 COM0A1:0 的设定，输出恒为高电平或低电平。

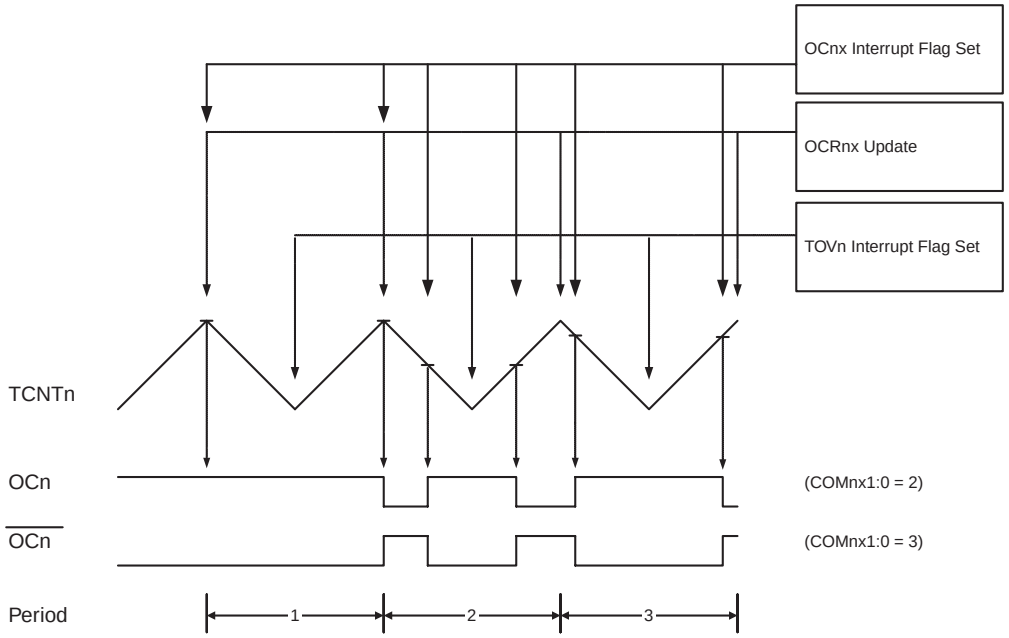
通过设定 OC0A 在比较匹配时进行逻辑电平取反 (COM0A1:0 = 1)，可以得到占空比为 50%的周期信号。OCR0A为0时信号有最高频率$f_{oc2} = f_{clk_I/O}/2$。这个特性类似于CTC模式下的 OC0A 取反操作，不同之处在于快速 PWM 模式具有双缓冲。

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 OCR0A 的匹配， OC0A 将清零为低电平；而在计时器往 BOTTOM 计数时若发生了TCNT0 与 OCR0A 的匹配，OC0A 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 32。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 OCR0A 与 TCNT0 的比较匹配。

Figure 32. 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OC0A引脚产生PWM波形：将COM0A1:0设置为 2 产生普通相位的 PWM，设置 COM0A1:0 为 3 产生反向 PWM 信号 (参见 P87 Table 52)。要想在引脚上得到输出信号还必须将 OC0A 的数据方向设置为输出。OCR0A 和 TCNT0 比较匹配发生时 OC0A 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：

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

变量 N 表示预分频因子 (1、8、64、 256 或 1024)。

OCR0A 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 OCR0A 等于 BOTTOM，输出一直保持为低电平；若 OCR0A 等于 MAX，则输出保持为高电平。反向 PWM 模式则正好相反。

在 Figure 32 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 32 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。为了保证波形在 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 OCR0A 高的值开始记数，并因而丢失了一次比较匹配。为了保证波形在 BOTTOM 两侧的对称，此时 OCn 必须立即切换到合适的电平。系统因此引入了第二个虽然没有比较匹配发生 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_{T2} 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 33 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 33. T/C 时序图，无预分频器

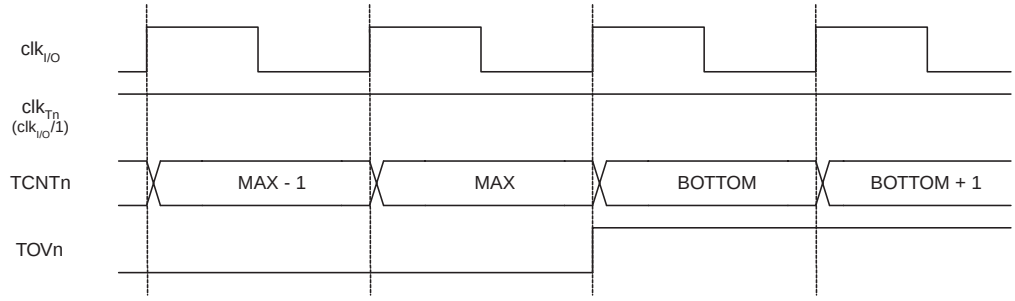


Figure 34 所示为相同的工作时序，但有预分频。

Figure 34. T/C 时序图，预分频器为 $f_{clk\_I/O}/8$

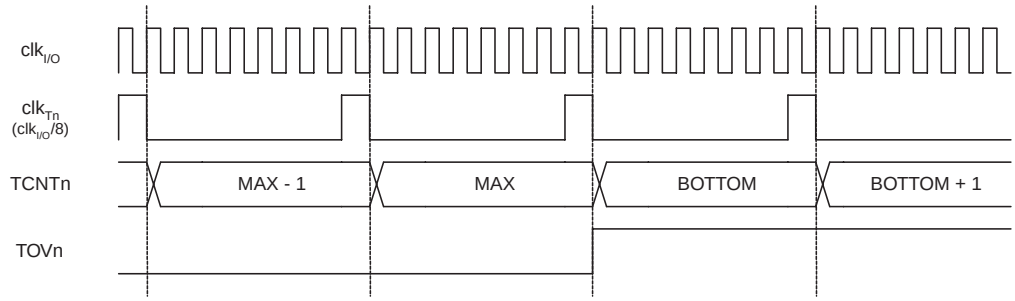


Figure 35 给出了各种模式下 (除了 CTC 模式)OCF0A 的置位情况。

Figure 35. T/C 时序图， OCF0A 置位，预分频器为 $f_{clk\_I/O}/8$

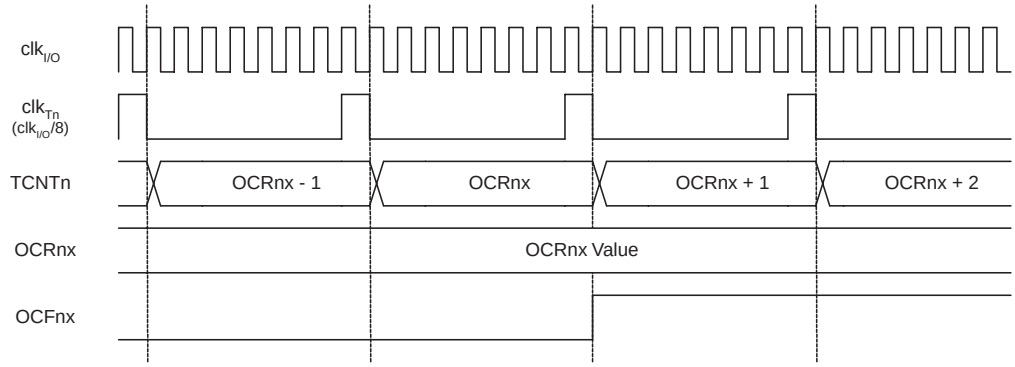
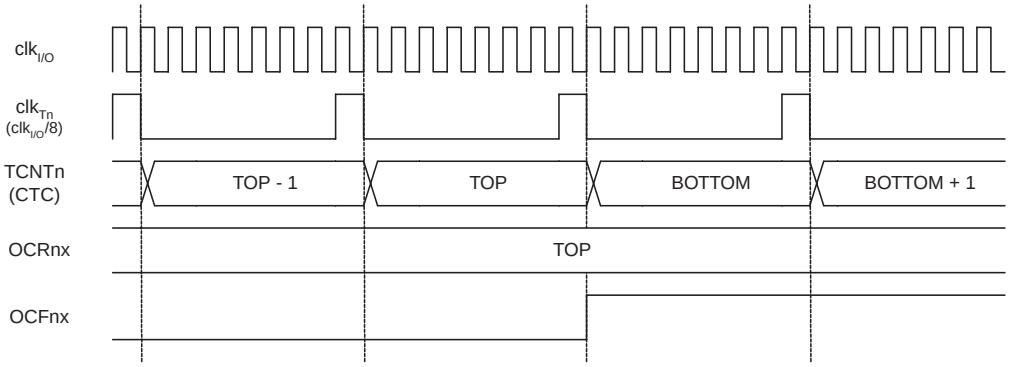


Figure 36 给出了 CTC 模式下 OCF0A 置位和 TCNT0 清除的情况。

Figure 36. 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 A—TCCR0A

位	7	6	5	4	3	2	1	0	
	FOC0A	WGM00	COM0A1	COM0A0	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	

- **位 7 – FOC0A: 强制输出比较 A**

FOC0A 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 OC0A 将按照 COM0A1:0 的设置输出相应的电平。要注意 FOC0A 类似一个锁存信号，真正对强制输出比较起作用的是 COM0A1:0 的设置。

FOC0A不会引发任何中断，也不会在利用OCR0A作为TOP的CTC模式下对定时器进行清零的操作。

读 FOC0A 的返回值永远为 0。

- **位 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 49 与 P82 “ 工作模式 ” 。

Table 49. 波形产生模式的位定义

模式	WGM01 (CTC0)	WGM00 (PWM0)	T/C 的工作模式	TOP	OCR0A 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	OCR0A	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。

- **位 5:4 – COM0A1:0: 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 OC0A 的电平。如果 COM0A1:0 中的一位或全部都置位， OC0A 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 以使能输出驱动器。

当 OC0A 连接到物理引脚上时， COM0A1:0 的功能依赖于 WGM01:0 的设置。 Table 50 给出了当 WGM01:0 设置为普通模式或 CTC 模式时 COM0A1:0 的功能。 .

Table 50. 比较输出模式，非 PWM 模式

COM0A1	COM0A0	说明
0	0	正常的端口操作，不与 OC0A 相连接
0	1	比较匹配发生时 OC0A 取反
1	0	比较匹配发生时 OC0A 清零
1	1	比较匹配发生时 OC0A 置位

Table 51 给出了当 WGM01:0 设置为快速 PWM 模式时 COM0A1:0 的功能。 .

Table 51. 比较输出模式，快速 PWM 模式 (1)

COM0A1	COM0A0	说明
0	0	正常的端口操作，不与 OC0A 相连接
0	1	保留
1	0	比较匹配发生时 OC0A 清零，计数到 TOP 时 OC0A 置位
1	1	比较匹配发生时 OC0A 置位，计数到 TOP 时 OC0A 清零

Note: 1. 一个特殊情况是OCR0A等于TOP，且COM0A1置位。此时比较匹配将被忽略，而计数到 TOP 时 OC0A 的动作继续有效。详细信息请参见 P83“ 快速 PWM 模式 ”。

Table 52 给出了当 WGM01:0 设置为相位修正 PWM 模式时 COM0A1:0 的功能。

Table 52. 比较输出模式，相位修正 PWM 模式 (1)

COM0A1	COM0A0	说明
0	0	正常的端口操作，不与 OC0A 相连接

Table 52. 比较输出模式，相位修正 PWM 模式 (1)

COM0A1	COM0A0	说明
0	1	保留
1	0	在升序计数时发生比较匹配将清零 OC0A ；降序计数时发生比较匹配将置位 OC0A
1	1	在升序计数时发生比较匹配将置位 OC0A ；降序计数时发生比较匹配将清零 OC0A

Note: 1. 一个特殊情况是OCR0A等于TOP ,且COM0A1置位。此时比较匹配将被忽略 ,而计数到 TOP 时 OC0A 的动作继续有效。详细信息请参见 P84 “ 相位修正 PWM 模式 ” 。

- **位 2:0 – CS02:0: 时钟选择**

用于选择 T/C 的时钟源。

Table 53. 时钟选择位定义

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

位	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 OCR0A 的比较匹配。

输出比较寄存器 A—OCR0A

位	7	6	5	4	3	2	1	0	
	OCR0A[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 A 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 OC0A 引脚上产生波形。

T/C0 中断屏蔽寄存器—TIMSK0

位	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	OCIE0A	TOIE0	TIMSK0
读 / 写	R	R	R	R	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 1 – OCIE0A: T/C0 输出比较匹配 A 中断使能**

当 OCIE0A 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 A 中断使能。当 T/C0 的比较匹配发生，即 TIFR0 中的 OCF0A 置位时，中断服务程序得以执行。

- **位 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 TIFR0 中的 TOV0 位置位时，中断服务程序得以执行。

T/C0 中断标志寄存器—TIFR0

位	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	OCF0A	TOV0	TIFR0
读 / 写	R	R	R	R	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 1 – OCF0A: 输出比较标志 0 A**

当 T/C0 与 OCR0A(输出比较寄存器 0) 的值匹配时， OCR0A 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、 OCIE0A(T/C0 比较匹配中断使能) 和 OCF0A 都置位时，中断服务程序得到执行。

- **位 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)。

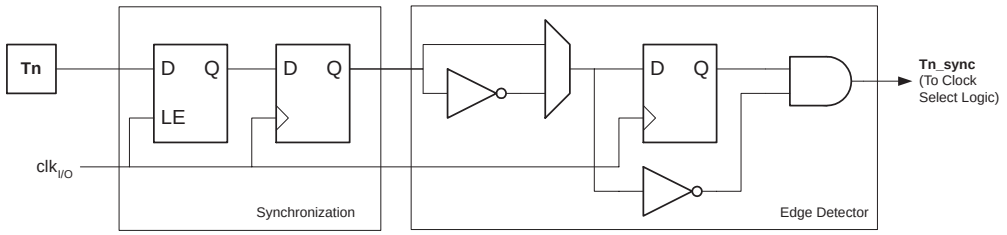
通过复位预分频器来同步 T/C 与程序运行是可能的。但是必须注意另一个 T/C 是否也在使用这一预分频器，因为预分频器复位将会影响所有与其连接的 T/C。

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 37 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 37. T1/T0 引脚采样



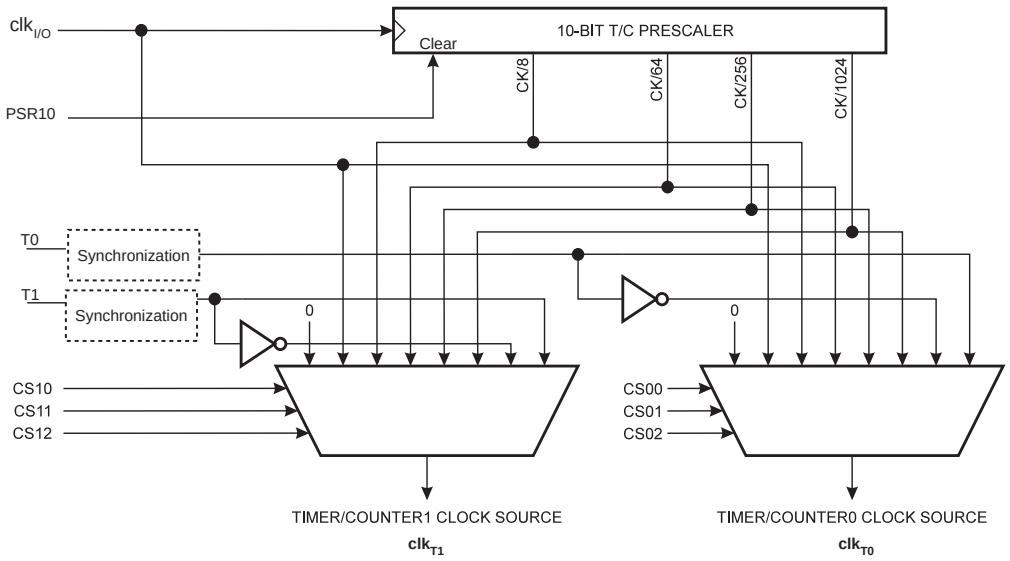
由于引脚上同步与边沿监测电路的存在，引脚 T1/T0 上的电平变化需要延时 2.5 到 3.5 个系统时钟周期才能使计数器进行更新。

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 38. T/C0 与 T/C1 预分频器



Note: 1. 输入引脚 (T1/T0) 的同步逻辑见 Figure 37。

通用 T/C 控制寄存器—GTCCR

位	7	6	5	4	3	2	1	0	
	TSM	–	–	–	–	–	PSR2	PSR10	GTCCR
读 / 写	R/W	R	R	R	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 7 – TSM: T/C 同步模式**

TSM置位激活T/C同步模式。只要TSM置位，PSR2 与PSR10的数值将保持不变，使得相关的定时器 / 计数器预分频器处于持续复位状态。这样相关的 T/C 将停止工作。用户可以为它们赋予相同的数值而不会出现在配置一个定时器 / 计数器时另一个 T/C 在运行的现象。一旦TSM清零，PSR2 与PSR10位被硬件清零，相关的定时器/计数器同时开始计数。

- **位 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。操作完成后这一位通常由硬件立即清零，除非 TSM 置位。要注意的是 T/C1 与 T/C0 共用一个预分频器，复位对两个计时器都有影响。

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:

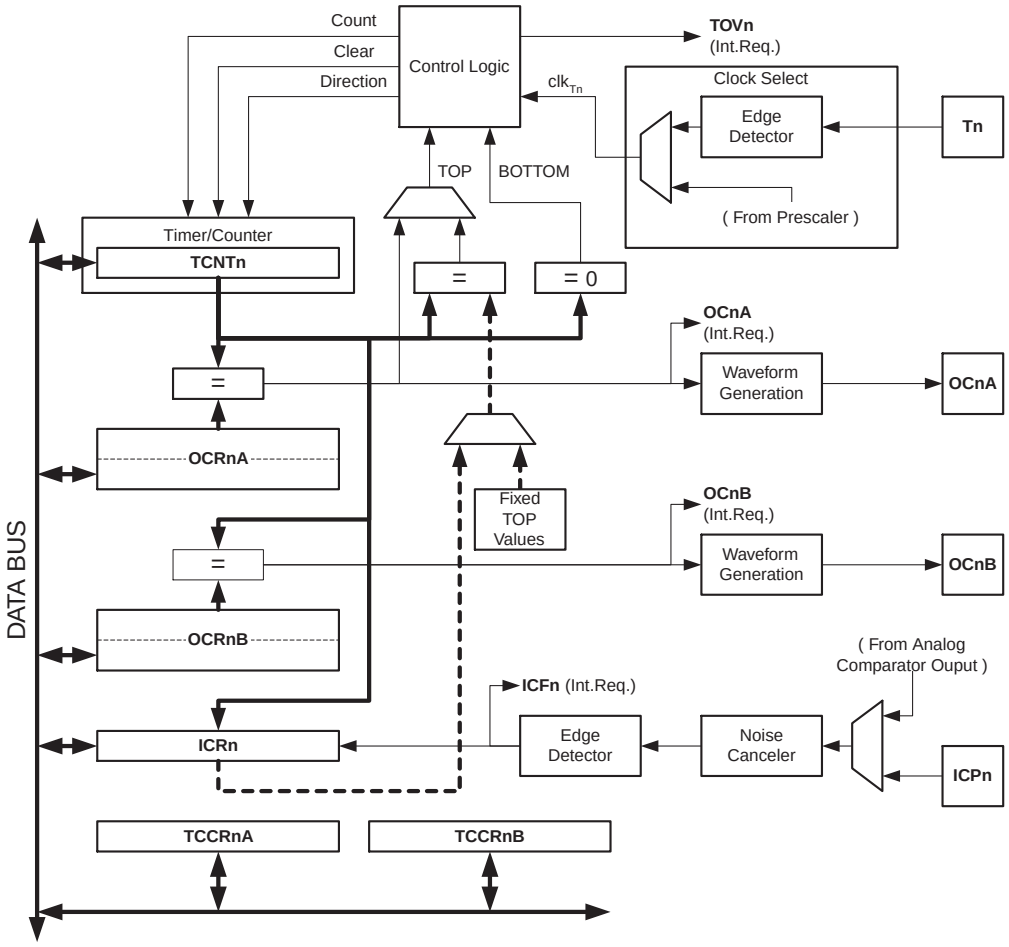
- **真正的 16 位设计 (即允许 16 位的 PWM)**
- **2 个独立的输出比较单元**
- **双缓冲的输出比较寄存器**
- **一个输入捕捉单元**
- **输入捕捉噪声抑制器**
- **比较匹配发生时清除寄存器 (自动重载)**
- **无干扰脉冲，相位正确的 PWM**
- **可变的 PWM 周期**
- **频率发生器**
- **外部事件计数器**
- **4 个独立的中断源 (TOV1、OCF1A、 OCF1B 与 ICF1)**

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 39。I/O引脚的实际位置请参见P2 “ATmega169引脚排列”。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 P112 “16 位定时器 / 计数器寄存器的说明”。

Figure 39. 16 位 T/C 框图 (1)



Note: 1. 请参考 P2 Figure 1 ， P58Table 28 和 P62 Table 34 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 P94 “ 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P99 “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 P181 “ 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 54. 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：

- TCCR1C 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。当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、 “OUT”、 “SBIS”、“SBIC”、 “CBI” 与 “SBI” 指令。

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原子操作是非常重要的。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 寄存器内容的原子操作。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。当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、 “OUT”、 “SBIS”、“SBIC”、 “CBI” 与 “SBI” 指令。

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 寄存器的原子操作。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。当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、 “OUT”、 “SBIS”、“SBIC”、 “CBI” 与 “SBI” 指令。

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。前面讲到的原子操作在这种情况下同样适用。

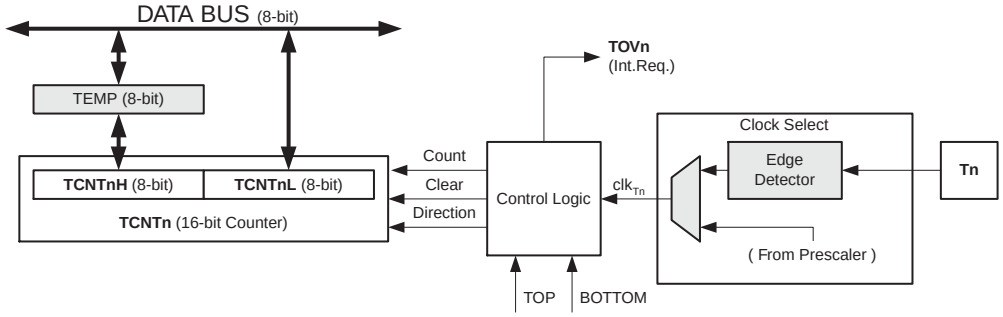
T/C 时钟源

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 P90 “T/C0 与 T/C1 的预分频器 ” 。

控制单元

16 位 T/C 的主要部分是可编程的 16 位双向计数器单元。 Figure 40 给出了计数器与其外围电路方框图。

Figure 40. 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 P102 “ 工作模式 ” 。

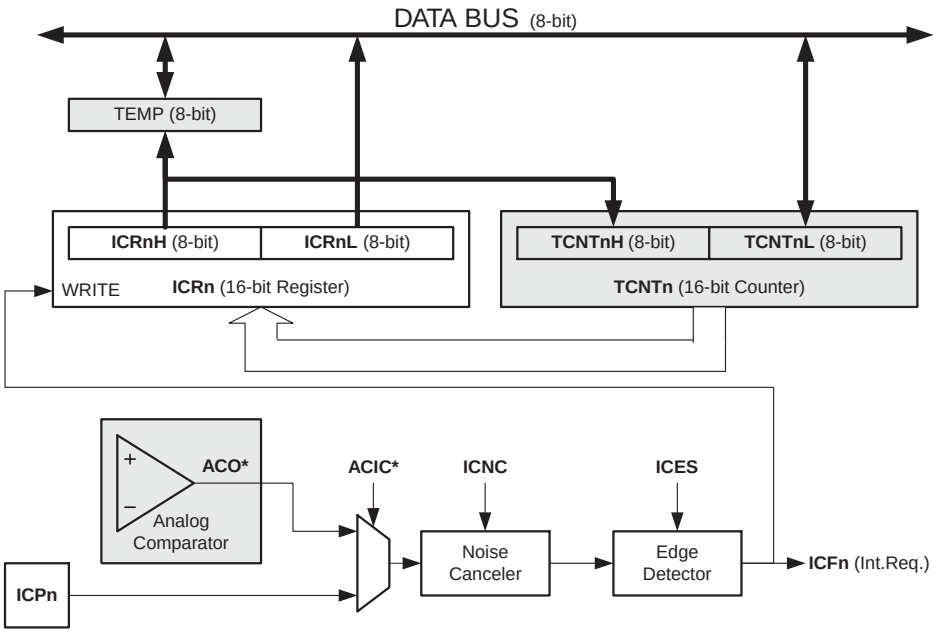
通过 WGM13:0 确定了计数器的工作模式之后，TOV1 的置位方式也就确定了。TOV1 可以用来产生 CPU 中断。

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 41。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 41. I 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 P94 “ 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 (P90 Figure 37), 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。

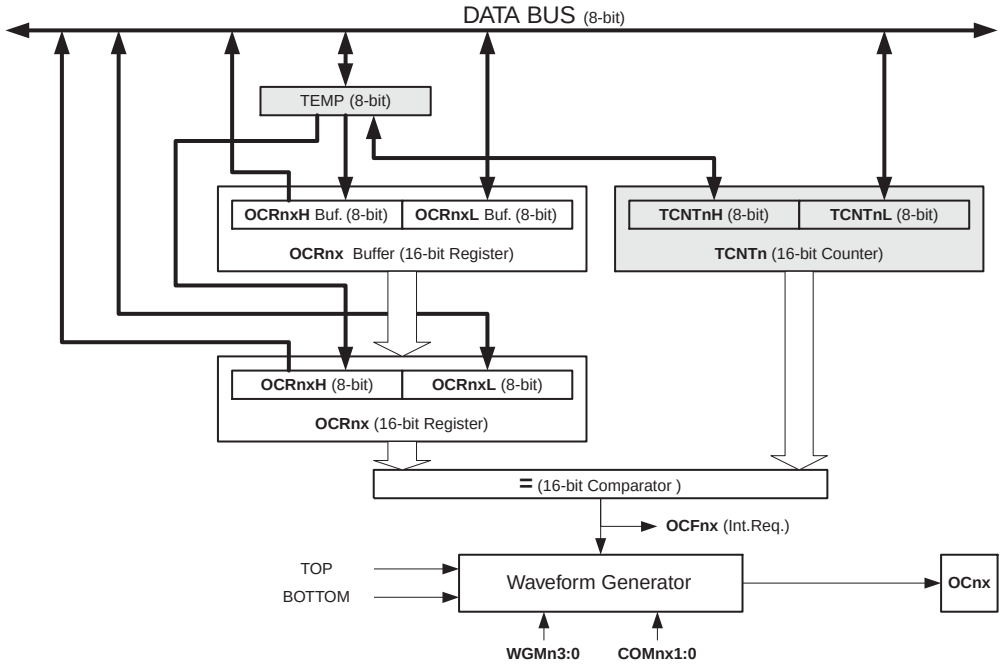
输出比较单元

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 (P102 “ 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 42 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 42. 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 P94 “ 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。

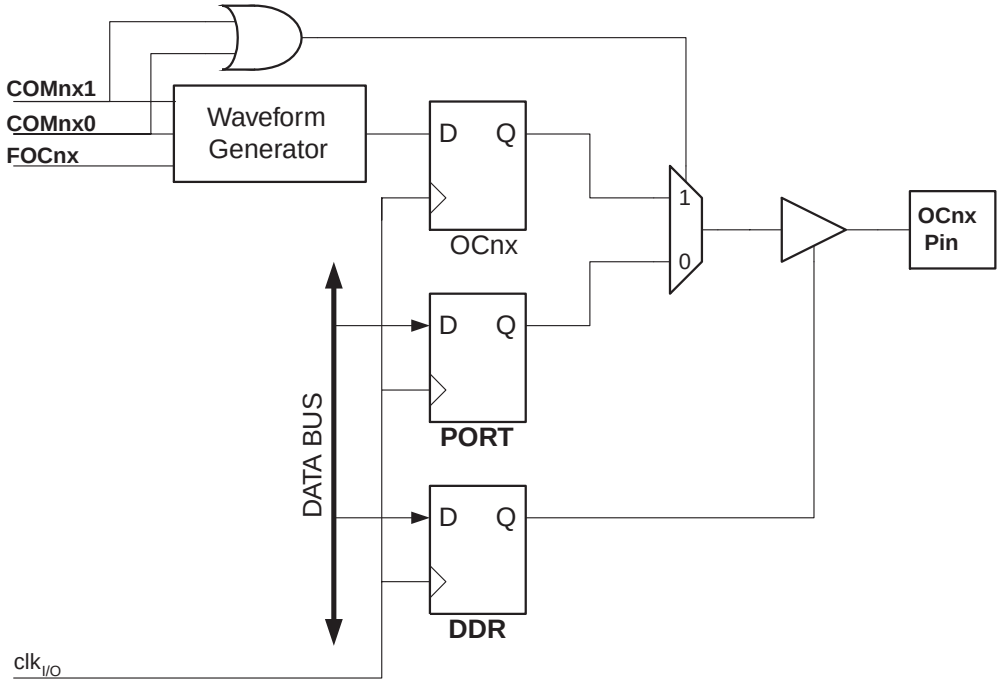
OC1x 的设置应该在设置数据方向寄存器之前完成。最简单的设置 OC1x 的方法是在普通模式下利用强制输出比较 FOC1x。即使在改变波形发生模式时 OC1x 寄存器也会一直保持它的数值。

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 43 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 43. 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 55、Table 56 与 Table 57。

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P112 “16位定时器/计数器寄存器的说明” 所示。

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 P112 Table 55 ，快速 PWM 的比较输出于 P112 Table 56 ，相位修正 PWM 的比较输出于 P113 Table 57 。

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 (见 P101 “ 比较匹配输出单元 ”)。

具体的时序信息请参考 P110 “ 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。

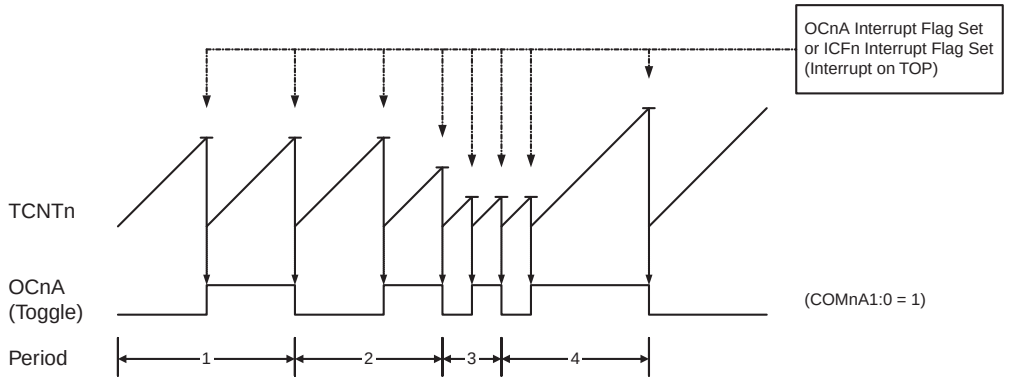
输出比较单元可以用来产生中断。但是不推荐在普通模式下利用输出比较来产生波形 , 因为会占用太多的 CPU 时间。

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 44。计数器数值TCNT1一直累加到TCNT1与OCR1A 或ICR1匹配，然后 TCNT1 清零。

Figure 44. 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。

为了在CTC模式下得到波形输出，可以设置OC0A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 $f_{OC2} = f_{clk\_I/O}/2$ (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

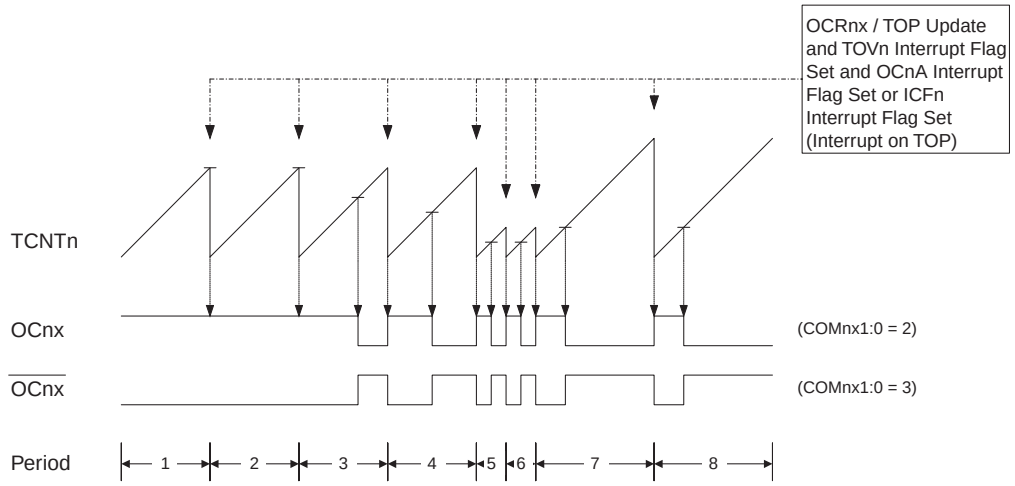
快速 PWM 模式 (WGM13:0 = 5、 6、7、 14 或 15) 可用来产生高频的 PWM 波形。快速 PWM 模式与其他 PWM 模式的不同之处是其单边斜坡工作方式。计数器从 BOTTOM 计到 MAX ,然后立即回到 BOTTOM 重新开始。对于普通的比较输出模式 ,输出比较引脚 OC1x 在 TCNT1 与 OCR1x 匹配时置位，在 TOP 时清零；对于反向比较输出模式， OCR1x 的动作正好相反。由于使用了单边斜坡模式 ,快速 PWM 模式的工作频率比使用双斜坡的相位修正 PWM 模式高一倍。此高频操作特性使得快速 PWM 模式十分适合于功率调节 ,整流和 DAC 应用。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 45。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 45. 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 P112 Table)。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 $f_{oc2} = f_{clk\_I/O}/2$。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

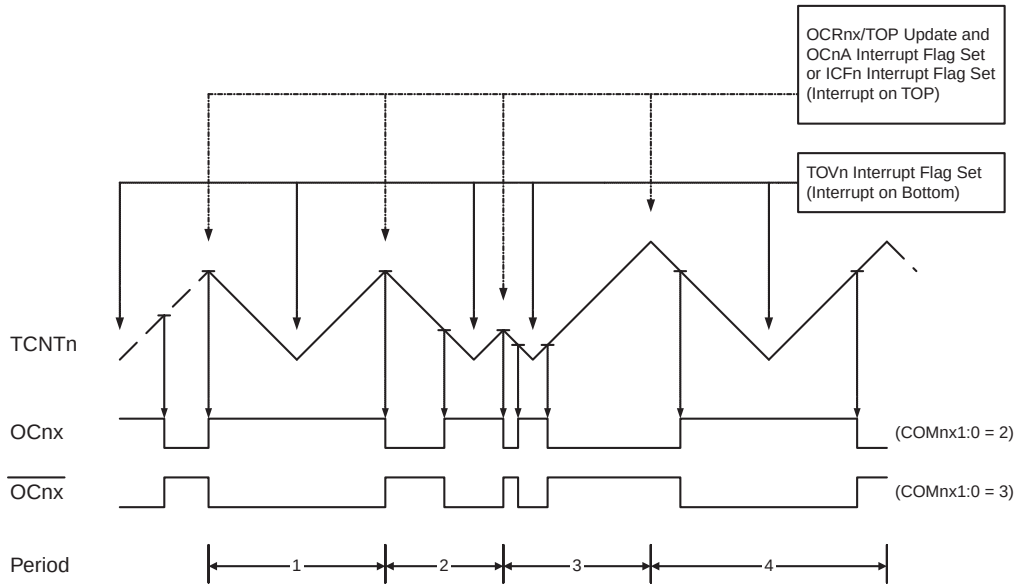
相位修正 PWM 模式 (WGM13:0 = 1、2、3、10 或 11) 为用户提供了一个获得高精度的、相位准确的 PWM 波形的方法。与相位和频率修正模式类似，此模式基于双斜坡操作。计时器重复地从 BOTTOM 计到 TOP，然后又从 TOP 倒退回到 BOTTOM。在一般的比较输出模式下，当计时器往 TOP 计数时若 TCNT1 与 OCR1x 匹配，OC1x 将清零为低电平；而在计时器往 BOTTOM 计数时若 TCNT1 与 OCR1x 匹配，OC1x 将置位为高电平。工作于反向比较输出时则正好相反。与单斜坡操作相比，双斜坡操作可获得的最大频率要小。但其对称特性十分适合于电机控制。

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 46。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 46. 相位修正 PWM 模式的时序图



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。

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 46 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P113 Table)。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。

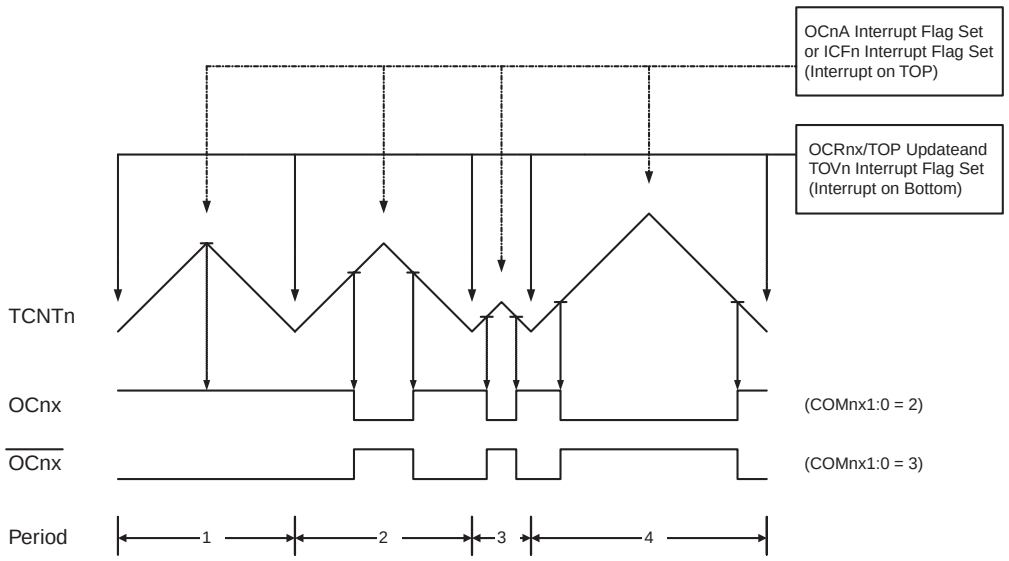
相频修正修正 PWM 模式与相位修正 PWM 模式的主要区别在于 OCR1x 寄存器的更新时间，详见 Figure 46 与 Figure 47。

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 47。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 47. 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 47 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。

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 P112 Table)。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)。

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P113 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 48 给出了置位 OCF1x 的时序图。

Figure 48. T/C 时序图 ， OCF1x 置位，无预分频器

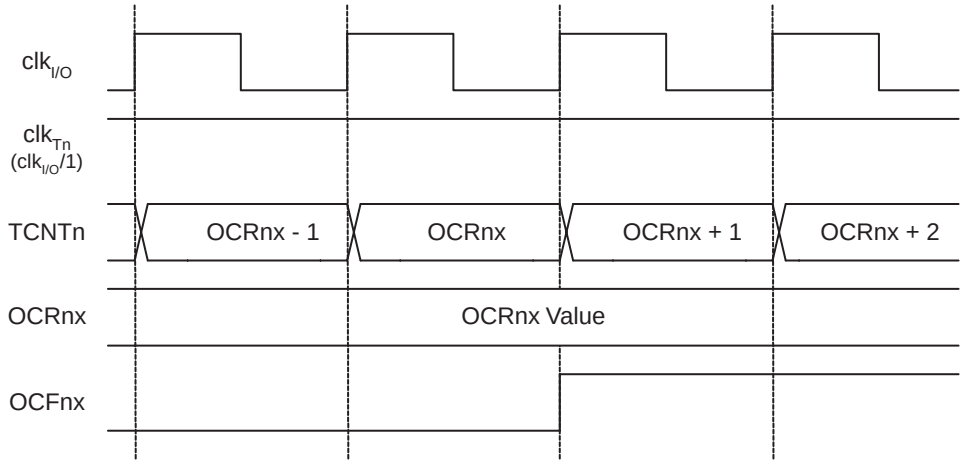


Figure 49 给出相同的时钟数据，但预分频使能。

Figure 49. T/C 时序图，置位 OCF1x，预分频器为 $f_{clk\_I/O}/8$

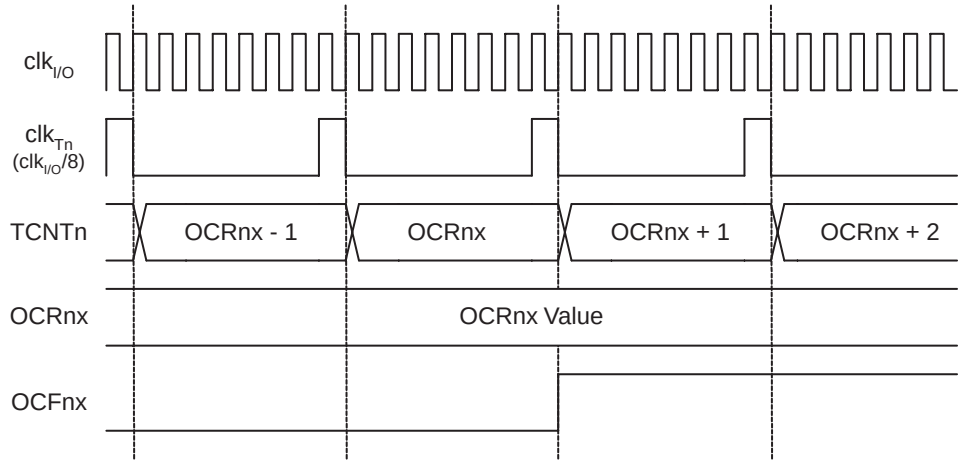


Figure 50 给出工作在不同模式下接近 TOP 值时的计数序列。工作于相频修正 PWM 模式时， OCR1x 寄存器在 BOTTOM 被更新。时序图相同，但 TOP 需要用 BOTTOM 代替，BOTTOM+1 代替 TOP-1 ,等等。同样的命名规则也适用于那些在 BOTTOM 置位 TOV1 标志的工作模式。

Figure 50. T/C 时序图，无预分频器

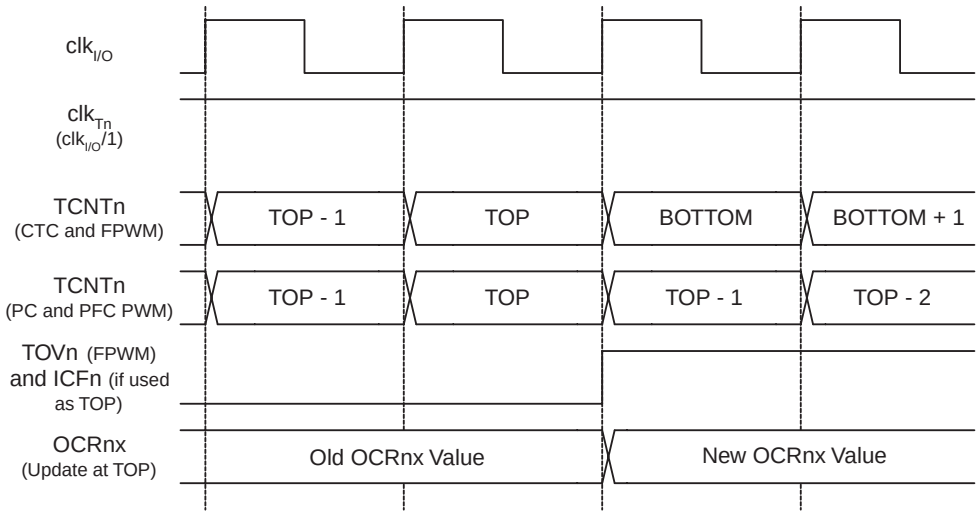
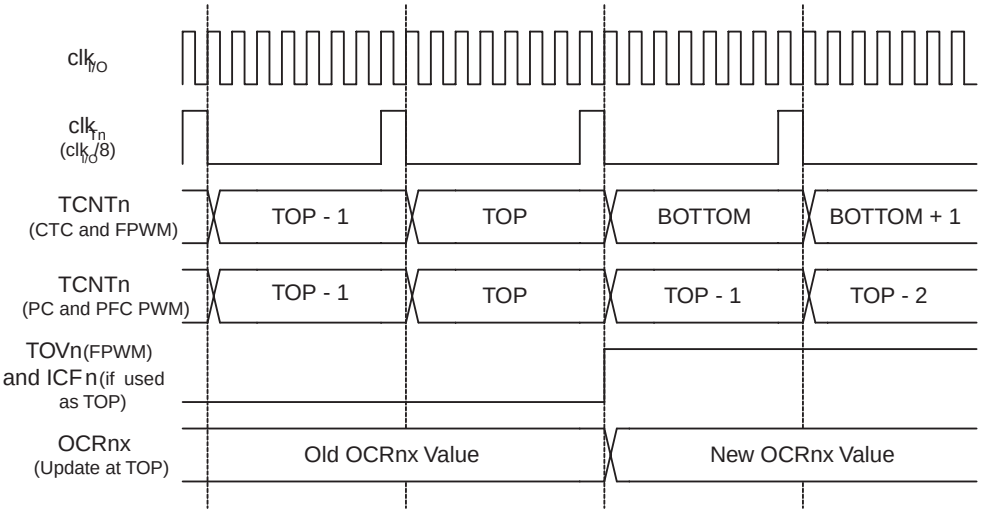


Figure 51 给出相同的时钟数据，但预分频使能。

Figure 51. 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

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

- **位 7:6 – COM1A1:0: 通道 A 的比较输出模式**

- **位 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 55 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 55. 比较输出模式，非 PWM

COM1A1/COM1B1	COM1A0/COM1B0	说明
0	0	普通端口操作，非 OC1A/OC1B 功能
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 56 给出 WGM13:0 设置为快速 PWM 模式时 COM1x1:0 的功能定义。

Table 56. 比较输出模式，快速 PWM(1)

COM1A1/COM1B1	COM1A0/COM1B0	说明
0	0	普通端口操作，非 OC1A/OC1B 功能
0	1	WGM13:0 = 15: 比较匹配时 OC1A 取反 ，OC1B 不占用物理引脚。 WGM1 为其它值时为普通端口操作，非 OC1A/OC1B 功能
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 P104 “ 快速 PWM 模式 ” 。

Table 57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 57. 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，非 OC1A/OC1B 功能
0	1	WGM13:0 = 9 或 14: 比较匹配时 OC1A 取反，OC1B 不占用物理引脚。WGM1 为其它值时为普通端口操作，非 OC1A/OC1B 功能
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 P106 “ 相位修正 PWM 模式 ” 。

- **位 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 58)。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 (P102 “ 工作模式 ”)。

Table 58. 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	计数上限值 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

位	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	

- **位 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。

- **位 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。

- **位 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"。

- **位 4:3 – WGM13:2: 波形发生模式**

见 TCCR1A 寄存器中的描述。

- **位 2:0 – CS12:0: 时钟选择**

这 3 位用于选择 T/C 的时钟源，见 Figure 48 与 Figure 49。

Table 59. 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，其 1 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 控制寄存器 C—TCCR1C

位	7	6	5	4	3	2	1	0	
	FOC1A	FOC1B	–	–	–	–	–	–	TCCR1C
读 / 写	R/W	R/W	R	R	R	R	R	R	
初始值	0	0	0	0	0	0	0	0	

- **位 7 – FOC1A: 强制输出比较 A**

- **位 6 – FOC1B: 强制输出比较 B**

FOC1A/FOC1B位只在WGM13:0位被设置为非PWM模式时才有效。为了保证同以后的器件兼容，在 PWM 模式下写 TCCR1A 寄存器时，该寄存器必须清零。对 FOC1A/FOC1B 写 "1" 将强制波形发生器产生一次成功的比较匹配，并使波形发生器依据 COM1x1:0 的设置而改变 OC1A/OC1B 的输出状态。FOC1A/FOC1B 的作用如同一个选通信号，COM1x1:0 的设置才是最终确定比较匹配结果的因素。

FOC1A/FOC1B 选通信号不会产生任何中断请求，也不会对计数器清零，象使用 OCR1A 为 TOP 值的 CTC 工作模式那样。

FOC1A/FOC1B 的读返回值总为零。

T/C1—TCNT1H 与 TCNT1L

位	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 P94 " 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

位	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

位	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 P94 “ 访问 16 位寄存器 ”。

输入捕捉寄存器 1—ICR1H 与 ICR1L

位	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 P94 “ 访问 16 位寄存器 ” 。

T/C1 中断屏蔽寄存器—TIMSK1

位	7	6	5	4	3	2	1	0	
	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
读 / 写	R	R	R/W	R	R	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 5 – ICIE1: 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 IFR1 的 ICF1 置位,CPU 即开始执行 T/C1 输入捕捉中断服务程序 (见 P45 “ 中断 ”)。

- **位 2 – 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 TIFR1 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 P45 “ 中断 ”)。

- **位 1 – 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 TIFR1 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 P45 “ 中断 ”)。

- **位 0 – 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 TIFR1 上的 TOV1 置位, CPU 即开始执行 T/C1 溢出中断服务程序 (见 P45 “ 中断 ”)。

T/C1 中断标志寄存器—TIFR1

位	7	6	5	4	3	2	1	0	
	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	TIFR1
读 / 写	R	R	R/W	R	R	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 5 – ICF1: T/C1 输入捕捉标志位**

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。

- **位 2 – OCF1B: T/C1 输出比较 B 匹配标志位**

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。

- **位 1 – OCF1A: T/C1 输出比较 A 匹配标志位**

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。

- **位 0 – 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 P114 Table 58 。

执行溢出中断服务程序时 OCF1A 自动清零。也可以对其写入逻辑 "1” 来清除该标志位。

8 位有 PWM 与异步操作的定时器 / 计数器 2

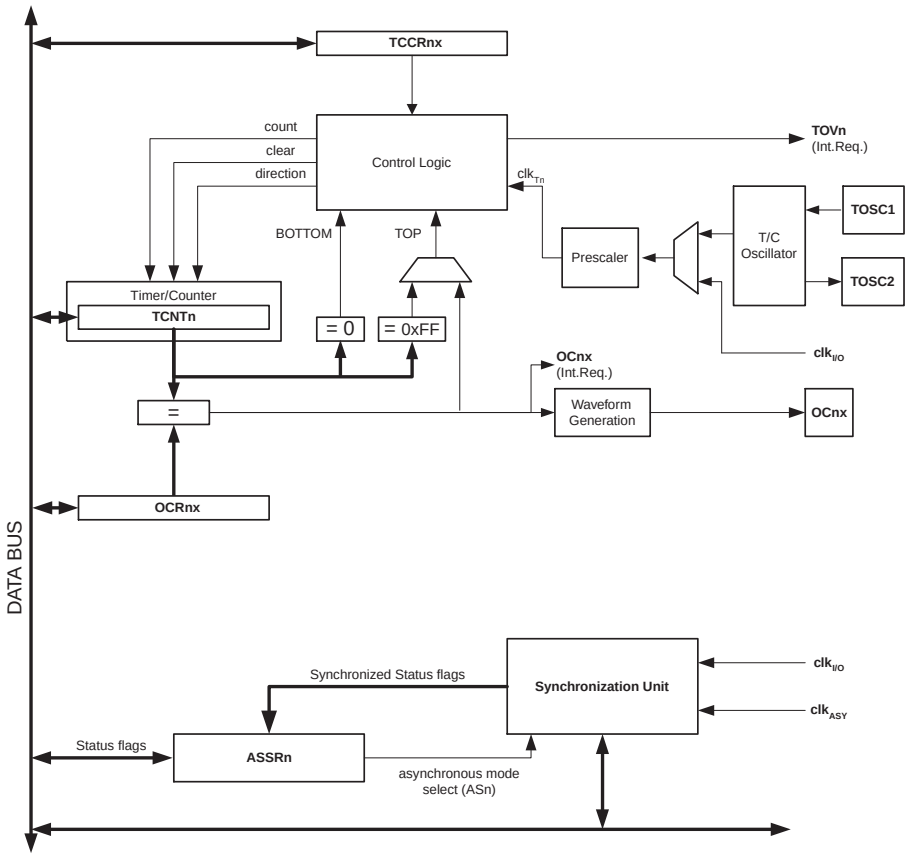
T/C2 是一个通用单通道 8 位定时 / 计数器，其主要特点如下：

- **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 OCF2A)**
- **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 52为8位T/C的方框图。实际的引脚图请参见P2 “ATmega169引脚排列”。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 P128 “8 位 T/C 寄存器说明”。

Figure 52. 8 位 T/C 方框图



寄存器

定时器 / 计数器 TCNT2、输出比较寄存器 OCR2A 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 TIFR2 都有反映。所有中断都可以通过定时器中断屏蔽寄存器 TIMSK2 单独进行屏蔽。图中未给出 TIFR2 与 TIMSK2。

T/C2 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 OCR2A 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 OC2A 输出可变频率的信号。参见 P122“ 输出比较单元 ”。比较匹配结果还会置位比较匹配标志 OCF2A，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 60 中定义适用于本节：

Table 60. 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 OCR2A 里的数值，具体由工作模式确定

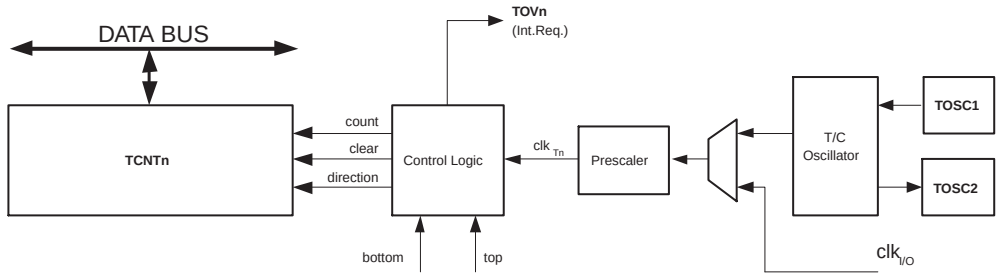
T/C 的时钟源

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 P131 “ 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 P134 “ 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 53 即为计数器和它周边电路的方框图。 8 位 T/C 的主要部分为可编程的双向计数单元。

Figure 53. 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 (TCCR2A) 的 WGM21 和 WGM20 决定。计数器计数行为与输出比较 OC2A 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 P122 “ 工作模式 ”。

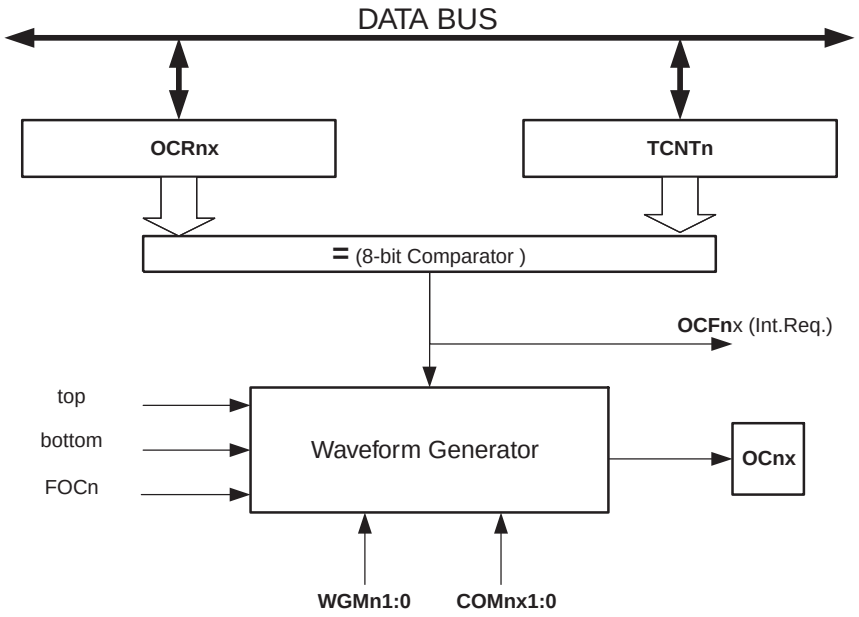
T/C溢出中断标志TOV2根据WGM21:0 设定的工作模式来设置。TOV2可以用于产生CPU中断。

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 OCR2A 进行比较。一旦 TCNT2 等于 OCR2A，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 OCF2A置位。若OCIE2A = 1还将引发输出比较中断。执行中断服务程序时OCF2A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 COM2A1:0 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 (P122 “ 工作模式 ”)。

Figure 54 为输出比较单元的方框图。

Figure 54. 输出比较单元方框图



使用 PWM 模式时 OCR2A 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 OCR2A 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OCR2A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OCR2A缓冲寄存器；禁止双缓冲功能时 CPU 访问的则是 OCR2A 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 FOC2A 写 ”1” 来产生比较匹配。强制比较匹配不会置位 OCF2A 标志，也不会重载 / 清零定时器，但是 OC2A 引脚将被更新，好象真的发生了比较匹配一样 (COM2A1:0 决定 OC2A 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 OCR2A 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 OCR2A，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。

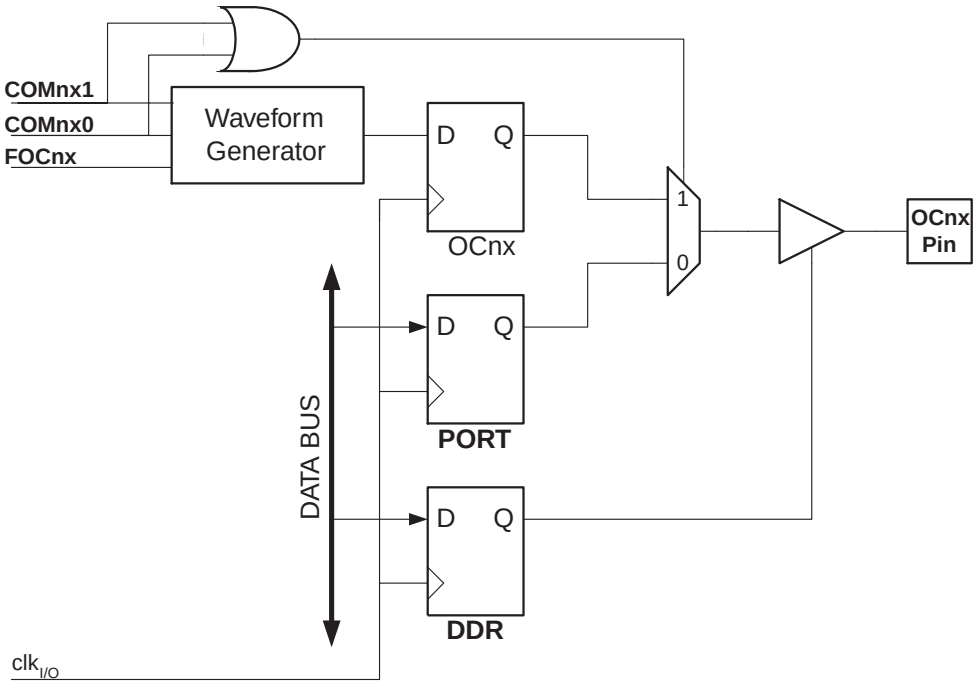
OC2A 的设置应该在设置数据方向寄存器之前完成。最简单的设置 OC2A 的方法是在普通模式下利用强制输出比较 FOC2A。即使在改变波形发生模式时 OC2A 寄存器也会一直保持它的数值。

注意 COM2A1:0 和比较数据都不是双缓冲的。 COM2A1:0 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 COM2A1:0 具有双重功能。波形发生器利用 COM2A1:0 来确定下一次比较匹配发生时的输出比较状态 (OC2A) ； COM2A1:0 还控制 OC2A 引脚输出信号的来源。 Figure 55 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 COM2A1:0 影响的通用 I/O 端口控制寄存器 (DDR 和 PORT)。谈及 OC2A 状态时指的是内部 OC2A 寄存器，而不是 OC2A 引脚。

Figure 55. 比较匹配输出单元原理图



只要 COM2A1:0 的一个或两个置位，波形发生器的输出比较功能 OC2A 就会取代通用 I/O 口功能。但是 OC2A 引脚的方向仍然受控于数据方向寄存器 (DDR)。在使用 OC2A 功能之前首先要通过数据方向寄存器的 DDR_OC2A 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 OC2A 状态在输出之前首先进行初始化。要注意某些 COM2A1:0 设置保留给了其他操作类型，详见 P128 “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 COM2A1:0 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， COM2A1:0 = 0 表明比较匹配发生时波形发生器不会操作 OC2A 寄存器。非 PWM 模式的比较输出请参见 P129 Table 62 ；快速 PWM 的比较输出于 P129 Table 63 ；相位修正 PWM 的比较输出于 P129 Table 64 。

改变 COM2A1:0 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 FOC2A 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 (COM2A1:0) 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。 COM2A1:0 控制 PWM 输出是否反极性。非 PWM 模式时 COM2A1:0 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 (P124“ 比较匹配输出单元 ”)。

具体的时序信息请参考 P126 “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。

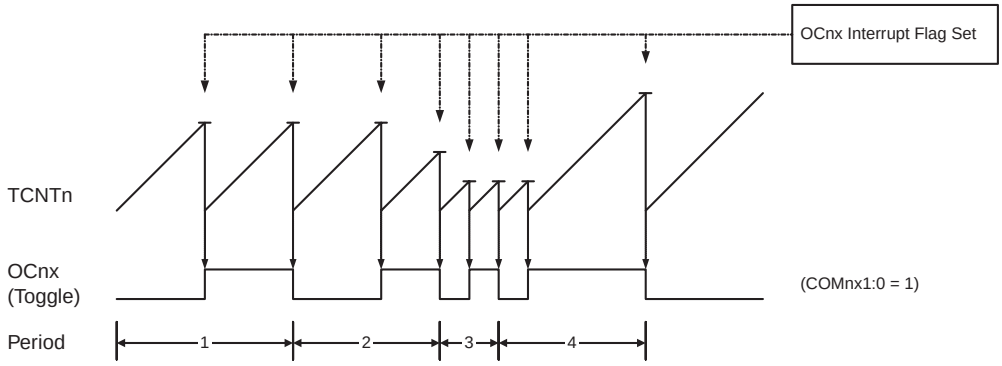
输出比较单元可以用来产生中断。但是不推荐在普通模式下利用输出比较产生波形，因为会占用太多的 CPU 时间。

CTC(比较匹配时清除定时器) 模式

在 CTC 模式 (WGM21:0 = 2) 里 OCR2A 寄存器用于调节计数器的分辨率。当计数器的数值 TCNT2 等于 OCR2A 时计数器清零。OCR2A 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 56。计数器数值TCNT2一直累加到TCNT2与OCR2A匹配，然后 TCNT2 清零。

Figure 56. CTC 模式的时序图



利用 OCF2A 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 OCR2A 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 OCR2A。

为了在CTC模式下得到波形输出，可以设置OC2A在每次比较匹配发生时改变逻辑电平。这可以通过设置 COM2A1:0 = 1 来完成。在期望获得 OC2A 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 $f_{OC2} = f_{clk\_I/O}/2$ (OC2A = 0x00)。频率由如下公式确定：

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

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。

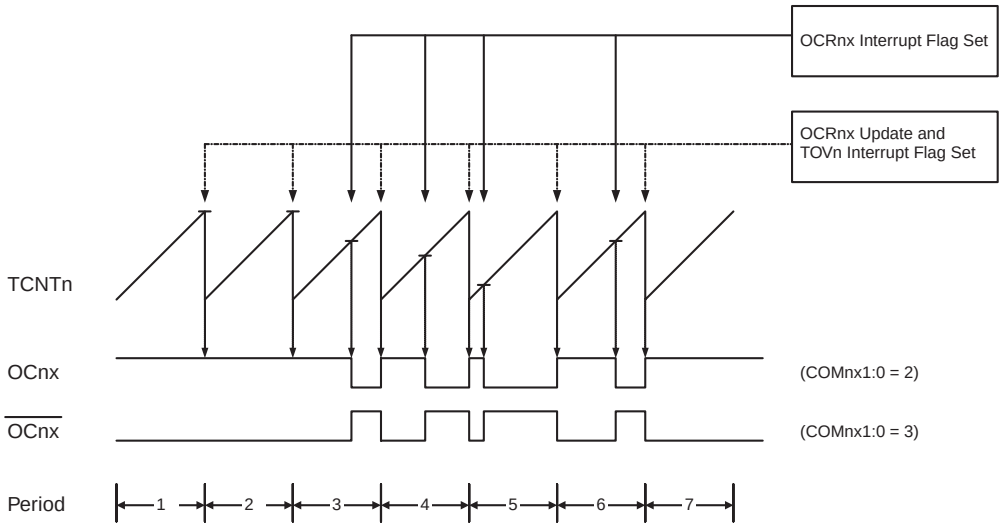
快速 PWM 模式

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 OC2A 在 TCNT2 与 OCR2A 匹配时清零，在 BOTTOM 时置位；对于反向比较输出模式，OC2A 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 57。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 OCR2A 和 TCNT2 的比较匹配。

Figure 57. 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 OC2A 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 P129 Table 63)。要想在引脚上得到输出信号还必须将 OC2A 的数据方向设置为输出。产生 PWM 波形的机理是 OC2A 寄存器在 OCR2A 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：

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

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

OCR2A 寄存器为极限值时表示快速 PWM 模式的一些特殊情况。若 OCR2A 等于 BOTTOM，输出为出现在第MAX+1个定时器时钟周期的窄脉冲；OCR2A为MAX时，根据 COM2A1:0 的设定，输出恒为高电平或低电平。

通过设定 OC2A 在比较匹配时进行逻辑电平取反 (COM2A1:0 = 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 OC2A 取反操作，不同之处在于快速 PWM 模式具有双缓冲。

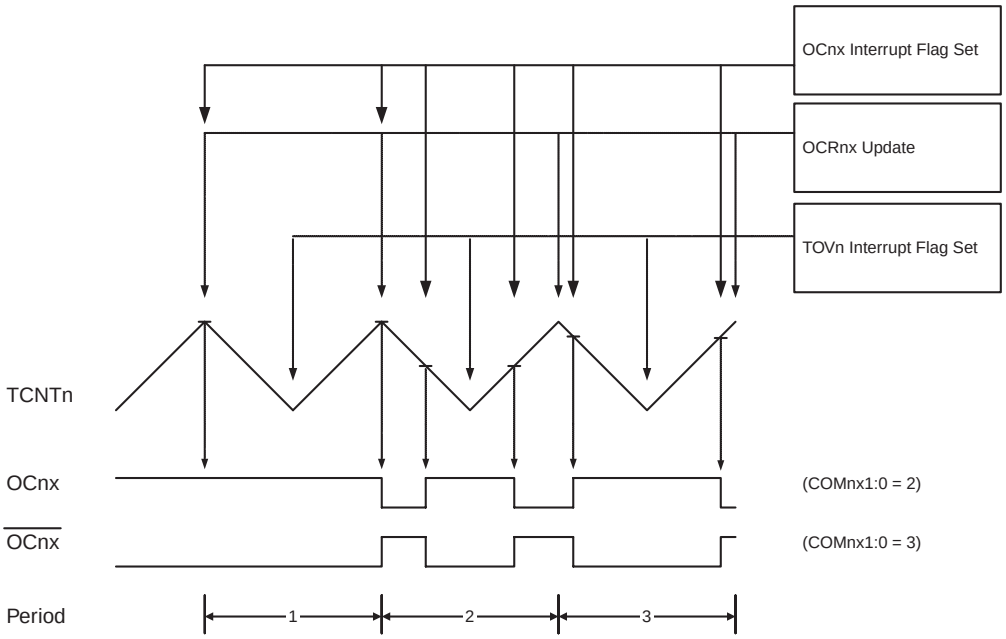
相位修正 PWM 模式

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 OCR2A 的匹配， OC2A 将清零为低电平；而在计时器往 BOTTOM 计数时若发生了 TCNT2 于 OCR2A 的匹配，OC2A 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 58。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 OCR2A 和 TCNT2 的比较匹配。

Figure 58. 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 OC2A 引脚产生 PWM 波形：将 COM2A1:0 设置为 2 产生普通相位的 PWM，设置 COM2A1:0 为 3 产生反向 PWM 信号（参见 P129 Table 64 ）。要想在引脚上得到输出信号还必须将 OC2A 的数据方向设置为输出。OCR2A 和 TCNT2 比较匹配发生时 OC2A 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：

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

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

OCR2A 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 OCR2A 等于 BOTTOM，输出一直保持为低电平；若 OCR2A 等于 MAX，则输出保持为高电平。反向 PWM 模式则正好相反。

在 Figure 58 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 58 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。为了保证波形在 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。为了保证波形在 BOTTOM 两侧的对称，此时 OCn 必须立即切换到合适的电平。系统因此引入了第二个虽然没有比较匹配发生 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 59 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 59. T/C 时序图，无预分频器

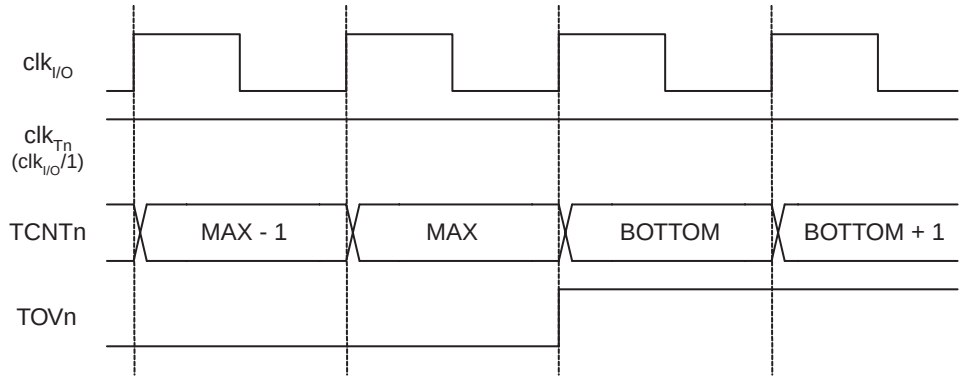


Figure 60 给出相同的定时器数据，但预分频器使能。

Figure 60. T/C 时序图，预分频器为 $f_{clk\_I/O}/8$

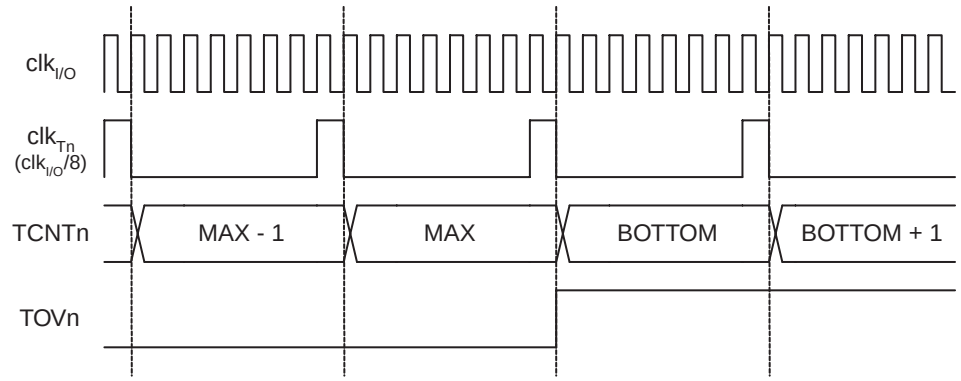


Figure 61 给出了各种模式下 (除了 CTC 模式)OCF2A 的置位情况。

Figure 61. T/C 时序图， OCF2A 置位，预分频器为 $f_{clk\_I/O}/8$

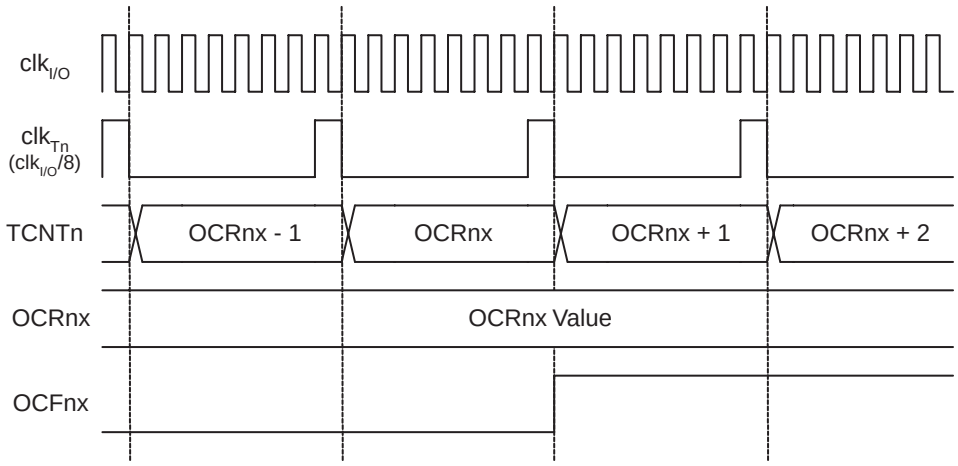
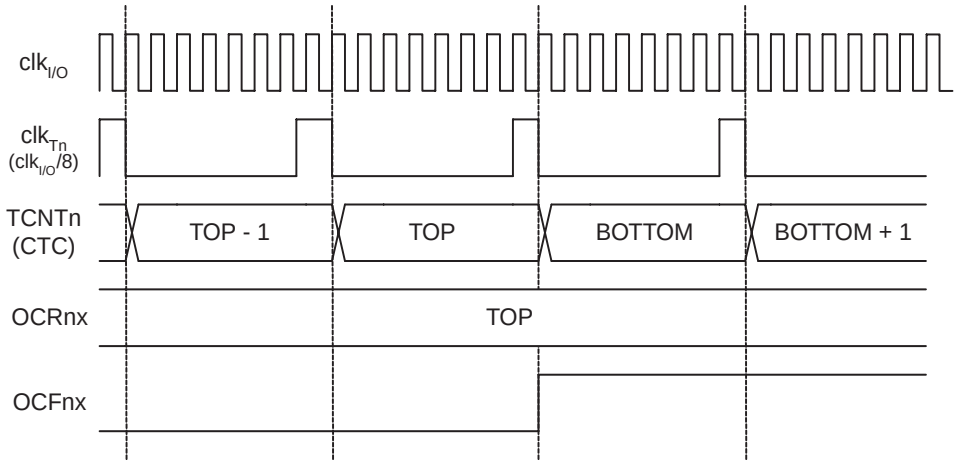


Figure 62 给出了 CTC 模式下 OCF2A 置位和 TCNT2 清除的情况。

Figure 62. 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 A—TCCR2A

位	7	6	5	4	3	2	1	0	
	FOC2A	WGM20	COM2A1	COM2A0	WGM21	CS22	CS21	CS20	TCCR2A
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	

- **位 7 – FOC2A: 强制输出比较**

FOC2A 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 TCCR2A 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 OC2A 将按照 COM2A1:0 的设置输出相应的电平。要注意 FOC2A 类似一个锁存信号，真正对强制输出比较起作用的是 COM2A1:0 的设置。

FOC2A不会引发任何中断 ,也不会在使用OCR2A作为TOP的CTC模式下对定时器进行清零。

读 FOC2A 的返回值永远为 0。

- **位 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 61 与 P122 “ 工作模式 ” 。

Table 61. 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	OCR2A 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	OCR2A	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。

- **位 5:4 – COM2A1:0: 比较匹配输出模式 A**

这些位决定了比较匹配发生时输出引脚 OC2A 的电平。如果 COM2A1:0 中的一位或全部都置位， OC2A 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 以使能当 OC2A 连接到物理引脚上时， COM2A1:0 的功能依赖于 WGM21:0 的设置。 Table 62 给出了当 WGM21:0 设置为普通模式或 CTC 模式时 COM2A1:0 的功能。

Table 62. 比较输出模式，非 PWM 模式

COM2A1	COM2A0	说明
0	0	正常的端口操作，不与 OC2A 相连接
0	1	比较匹配发生时 OC2A 取反
1	0	比较匹配发生时 OC2A 清零
1	1	比较匹配发生时 OC2A 置位

Table 63 给出了当 WGM21:0 设置为快速 PWM 模式时 COM2A1:0 的功能。

Table 63. 比较输出模式，快速 PWM 模式 (1)

COM2A1	COM2A0	说明
0	0	正常的端口操作，不与 OC2A 相连接
0	1	保留
1	0	比较匹配发生时 OC2A 清零 ，计数到 TOP 时 OC2A 置位
1	1	比较匹配发生时 OC2A 清零 ，计数到 TOP 时 OC2A 清零

Note: 1. 一个特殊情况是OCR2A等于TOP，且COM2A1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 P125“ 快速 PWM 模式 ”。

Table 64 给出了当 WGM21:0 设置为相位修正 PWM 模式时 COM2A1:0 的功能。

Table 64. 比较输出模式，相位修正 PWM 模式 (1)

COM2A1	COM2A0	说明
0	0	正常的端口操作，不与 OC2A 相连接
0	1	保留
1	0	在升序计数时发生比较匹配将清零 OC2A ；降序计数时发生比较匹配将置位 OC2A
1	1	在升序计数时发生比较匹配将置位 OC2A ；降序计数时发生比较匹配将清零 OC2A

Note: 1. 一个特殊情况是OCR2A等于TOP，且COM2A1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 P124 “ 相位修正 PWM 模式 ”。

- **位 2:0 – CS22:0: 时钟选择**

用于选择 T/C 的时钟源。参见 Table 65.

Table 65. 时钟选择位定义

CS22	CS21	CS20	说明
0	0	0	无时钟， T/C 不工作
0	0	1	clk_{T2S}/1 (没有预分频)
0	1	0	clk_{T2S}/8 (来自预分频器)

Table 65. 时钟选择位定义

CS22	CS21	CS20	说明
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)

T/C 寄存器—TCNT2

位	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 OCR2A 的比较匹配。

输出比较寄存器 A—OCR2A

位	7	6	5	4	3	2	1	0	
	OCR2A[7:0]								OCR2A
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 A 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 OC2A 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

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

- **位 4 – EXCLK: 外部时钟输入使能**

当 EXCLK 为 "1" 且选择了异步时钟，则外部时钟输入缓冲使能，可以从 TOSC1 引脚输入外部时钟，而不是 32 kHz 晶振。 EXCLK 的写操作应在选择异步操作之前完成。只有当该位为 "0" 时，晶振才能运行。

- **位 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、 OCR2A 与 TCCR2A 的内容。

- **位 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。

- **位 1 – OCR2UB: 输出比较寄存器 2 更新中**

T/C2 工作于异步模式时，写OCR2A 将引起 OCR2UB 置位。当 OCR2A 从暂存寄存器更新完毕后 OCR2UB 由硬件清零。 OCR2UB 为 0 表明 OCR2A 可以写入新值了。

- **位 0 – TCR2UB: T/C2 控制寄存器更新中**

T/C2工作于异步模式时，写TCCR2A将引起TCR2UB置位。当TCCR2A从暂存寄存器更新完毕后 TCR2UB 由硬件清零。 TCR2UB 为 0 表明 TCCR2A 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、 OCR2A 和 TCCR2A 的机制是不同的。读取 TCNT2 得到的是实际的值，而 OCR2A 和 TCCR2A 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、OCR2A 和 TCCR2A 数据的损毁。安全的步骤应该是：
  1. 清零 OCIE2A 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、 OCR2A 和 TCCR2A 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
- 系统主时钟必须比晶振高 4 倍以上。
- 写 TCNT2 ,OCR2A 和 TCCR2A 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 OCR2A 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 ADC 噪声抑制模式的唤醒条件 ,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 OCR2A 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 ADC 噪声抑制模式的唤醒条件 ,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 TCCR2A、 TCNT2 或 OCR2A 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 ADC 噪声抑制模式。
- 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 ADC 噪声抑制模式的唤醒过程 : 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 OCR2A 或 TCCR2A 。
  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 2 中断屏蔽寄存器—TIMSK2

位	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	OCIE2A	TOIE2	TIMSK2
读 / 写	R	R	R	R	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 1 – OCIE2A: T/C2 输出比较匹配 A 中断使能**

当 OCIE2A 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 TIFR2 中的 OCF2A 置位时 , 中断服务程序得以执行。

- **位 0 – 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 TIFR2 中的 TOV2 位置位时，中断服务程序得以执行。

定时器 / 计数器 2 中断标志寄存器—TIFR2

位	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	OCF2A	TOV2	TIFR2
读 / 写	R	R	R	R	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

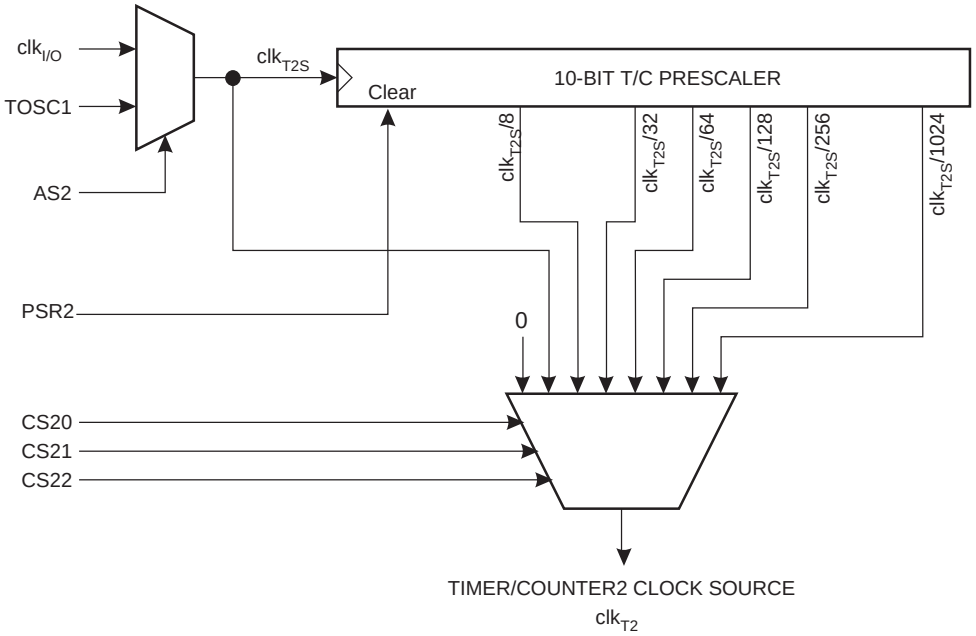
- **位 1 – OCF2A: 输出比较标志 2 A**

当 T/C2 与 OCR2A(输出比较寄存器 2) 的值匹配时 , OCF2A 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、OCIE2A 和 OCF2A 都置位时，中断服务程序得到执行。

- **位 0 – 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、TOIE2A 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。

Figure 63. 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_{T2} 与系统主时钟 $clk_{I/O}$ 连接。若置位 ASSR 的 AS0 , 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GTCCR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。

通用 T/C 控制寄存器—GTCCR

位	7	6	5	4	3	2	1	0	
	TSM	–	–	–	–	–	PSR2	PSR10	GTCCR
读 / 写	R/W	R	R	R	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 1 – PSR2:T/C2 预分频器复位**

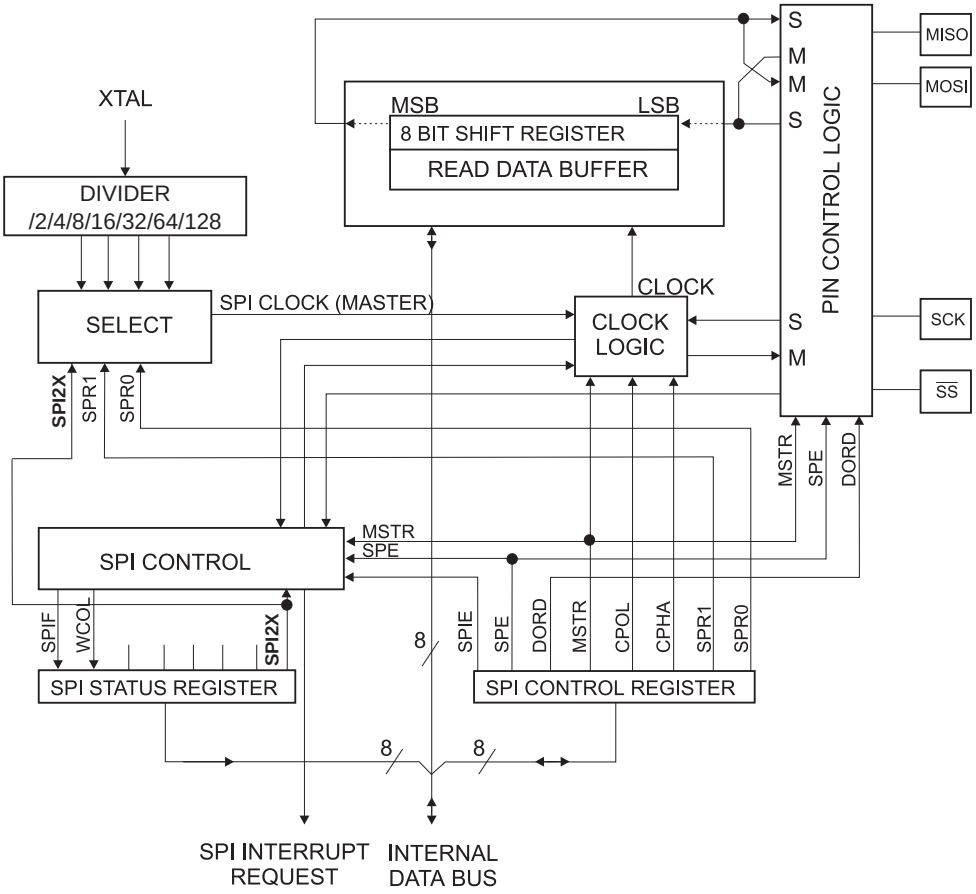
此位写 "1“ 时复位 T/C2 的预分频器。操作完成后通常立即由硬件清零。如果 T/C2 工作于异步模式，则这一位置位后一直保持到预分频器复位操作真正完成。如果 TSM 位置位，则该位不会被硬件清零。对T/C同步模式的描述请参见P91 “位 7 – TSM: T/C 同步模式” 。

串行外设接口—SPI

串行外设接口 SPI 允许 ATmega169 和外设或其他 AVR 器件进行高速的同步数据传输。ATmega169 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 64. SPI 方框图 (1)



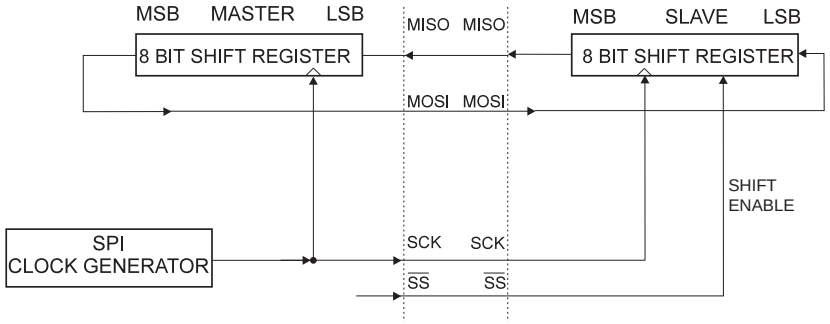
Note: 1. SPI 的引脚排列请参见 P2 Figure 1 与 P58 Table 29 。

主机和从机之间的 SPI 连接如 Figure 65 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；从机的数据从从机的 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 65. 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 66 所示自动进行配置。更多自动重载信息请参考 P55 “ 端口的第二功能 ” 。

Table 66. SPI 引脚重载 (1)

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: 1. 请参考 P58 “ 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 配置为主机时，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

位	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	

- **位 7 – SPIE: 使能 SPI 中断**

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。

- **位 6 – SPE: 使能 SPI**

SPE 置位将使能 SPI。进行任何 SPI 操作之前必须置位 SPE。

- **位 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。

- **位 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。

- **位 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 66 与 Figure 67。CPOL 功能总结如下：

Table 67. CPOL 功能

CPOL	起始沿	结束沿
0	上升沿	下降沿
1	下降沿	上升沿

- **位 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 66 与 Figure 67 。

Table 68. CPHA 功能

CPHA	起始沿	结束沿
0	采样	设置
1	设置	采样

- **位 1, 0 – SPR1, SPR0: SPI 时钟速率选择**

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 69. 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 状态寄存器—SPSR

位	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	

- **位 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。

- **位 6 – WCOL: 写碰撞标志**

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。

- **位 5..1 – Res: 保留位**

保留位，读操作返回值为零。

- **位 0 – SPI2X: SPI 倍速**

置位后 SPI 的速度加倍 (见 Table 69)。若为主机，则 SCK 频率可达 CPU 频率的一半。若为从机，必须保证此时钟不大于 $f_{osc}/4$ 以保证正常工作。

ATmega169 的 SPI 接口同时还用来实现程序和 EEPROM 的下载和上载。请参见 P269 的 SPI 串行编程和校验。

SPI 数据寄存器—SPDR

位	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	Undefined

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

There are four combinations of SCK phase and polarity with respect to serial data, which are determined by control bits CPHA and CPOL. The SPI data transfer formats are shown in Figure 66 and Figure 67. Data bits are shifted out and latched in on opposite edges of the SCK signal, ensuring sufficient time for data signals to stabilize. This is clearly seen by summarizing Table 67 and Table 68, as done

SCK 的相位、极性与数据间有 4 种组合。CPHA 和 CPOL 控制组合的方式。SPI 数据传输格式见 Figure 66 与 Figure 67。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 67 和 Table 68 有清楚的说明：

Table 70. CPOL 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 66. CPHA = 0 时 SPI 的传输格式

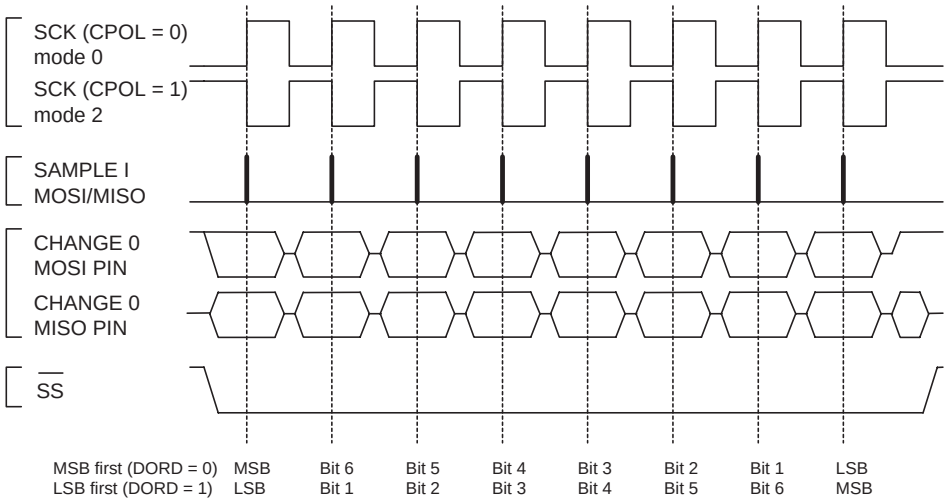
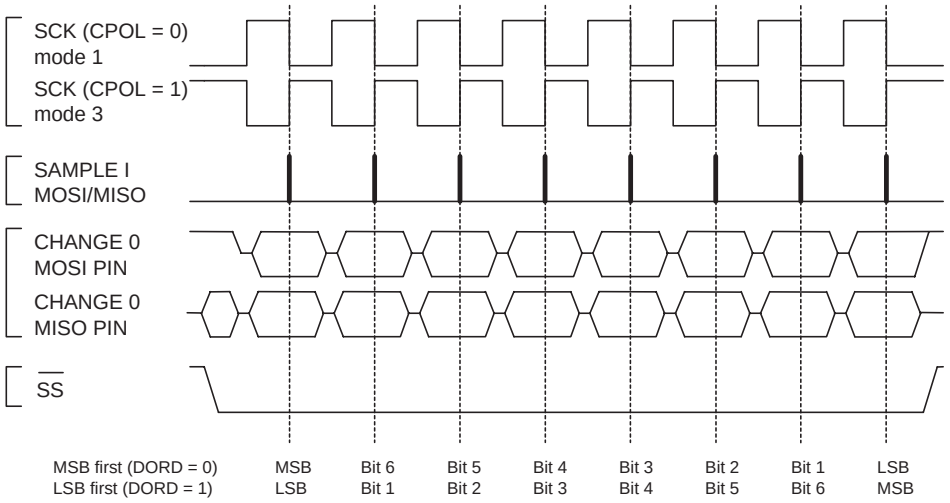


Figure 67. CPHA = 1 时 SPI 的传输格式



USART

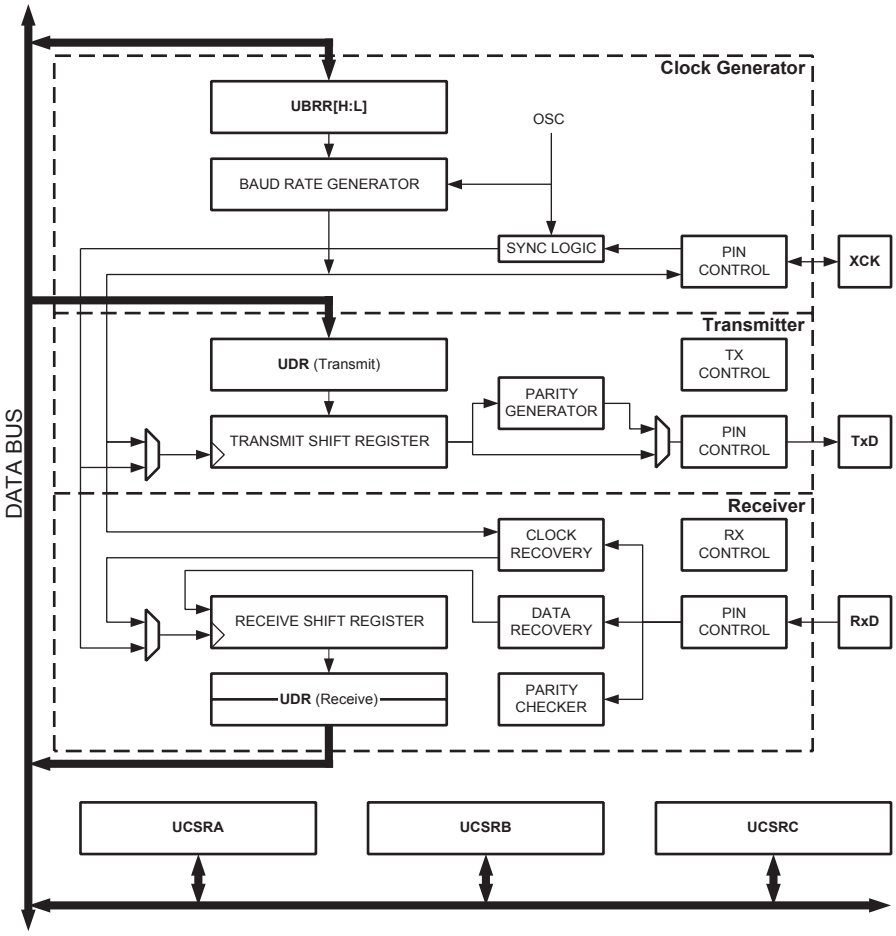
通用同步和异步串行接收器和转发器 (USART) 是一个高度灵活的串行通讯设备。主要特点为：

- **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 68 为 USART 转发器的简化框图。CPU 可以访问的 I/O 寄存器和 I/O 引脚以粗体表示。

Figure 68. USART 方框图 (1)



Note: 1. 请参考 P2 Figure 1 、 P64 Table 36 与 P60 Table 30 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 68) ，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：

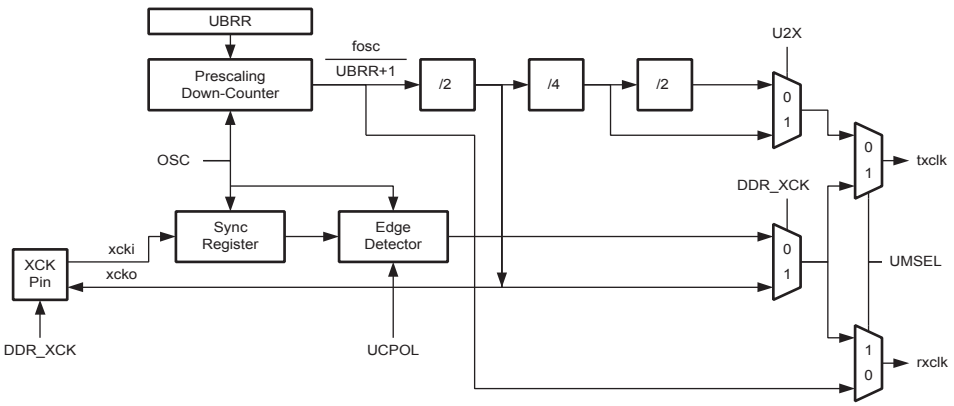
- CHR9 改为 UCSZ2
- OR 改为 DOR

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 69 为时钟产生逻辑的框图。

Figure 69. 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 69。

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 71 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 71. 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)

$\mathbf{f_{osc}}$ 系统时钟频率

UBRR UBRRH 与 UBRRL 的数值 (0-4095)

P169Table 79 给出了在某些系统时钟频率下对应的 UBRR 数值。

两倍速工作模式 (U2X)

通过设定 UCSRA 寄存器中的 U2X 位可以使传输速率加倍。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 69 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：

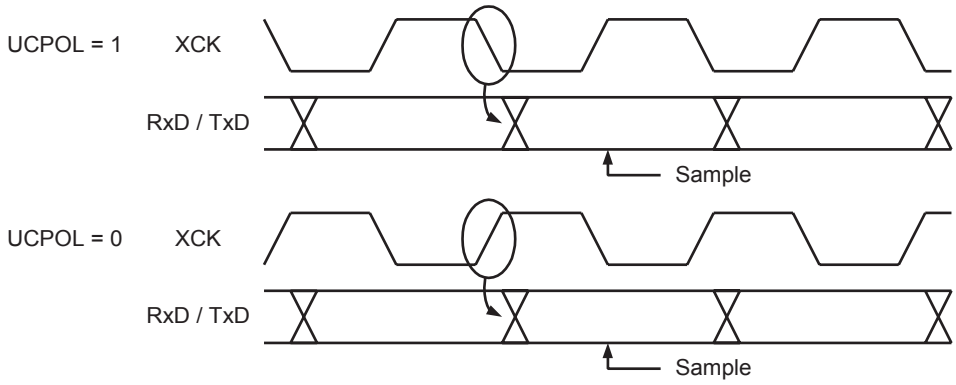
$$f_{XCK} < \frac{f_{OSC}}{4}$$

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 70. 同步模式时的 XCK 时序



The UCPOL bit UCRSC selects which XCK clock edge is used for data sampling and which is used for data change. As Figure 70 shows, when UCPOL is zero the data will be changed at rising XCK edge and sampled at falling XCK edge. If UCPOL is set, the data will be changed at falling XCK edge and sampled at rising XCK edge.

UCRSC 寄存器中的 UCPOL 确定使用 XCK 时钟的哪个边沿对数据采样和改变输出数据。如 Figure 70 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。

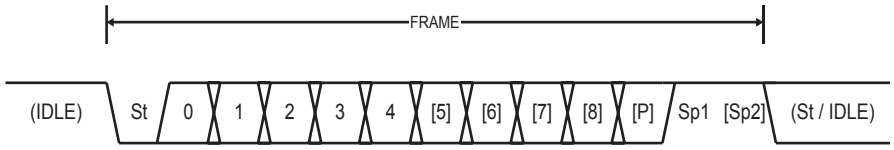
帧格式

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 71 所示为可能的数据帧结构组合。括号中的位是可选的。

Figure 71. 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。

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<<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<<USBS)|(3<<UCSZ0);
}
```

Note: 1. 本代码假定已经包含了合适的头文件
当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、“OUT”、“SBIS”、“SBIC”、“CBI” 与 “SBI” 指令。

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 )
{
  /* W 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
当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、“OUT”、“SBIS”、“SBIC”、“CBI” 与 “SBI” 指令。

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)(2)

```
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)(2)

```
void USART_Transmit( unsigned int data )
{
  /* W 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;
}
```

Notes:
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。
2. 本代码假定已经包含了合适的头文件
   当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、“OUT”、“SBIS”、“SBIC”、“CBI” 与 “SBI” 指令。

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
当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、“OUT”、“SBIS”、“SBIC”、“CBI” 与 “SBI” 指令。

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

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

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。

汇编代码例子 (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<<UPE)
  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 代码例子 (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 */
  status = UCSRA;
  resh = UCSRB;
  resl = UDR;
  /* 如果出错，返回 -1*/
  if ( status & (1<<FE)|(1<<DOR)|(1<<UPE) )
    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
当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、

“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、“OUT”、“SBIS”、“SBIC”、“CBI” 与 “SBI” 指令。

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 (UPE) 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 P148 “ 校验位的计算 ” 及 P156 “ 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。

汇编代码例子 (1)

```
USART_Flush:
  sbis  UCSRA, RXC
  ret
  in    r16, UDR
  rjmp  USART_Flush
```

C 代码例子 (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
当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、“OUT”、“SBIS”、“SBIC”、“CBI” 与 “SBI” 指令。

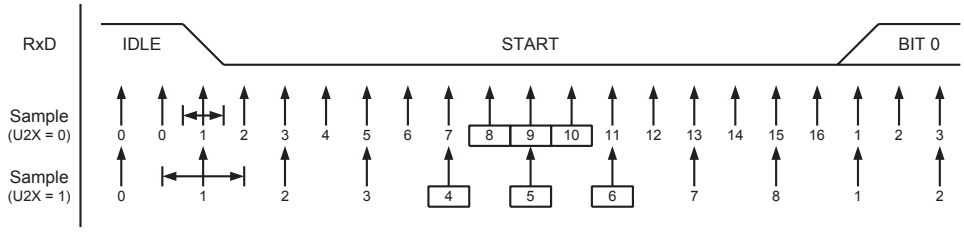
异步数据接收

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 72 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 72. 起始位采样

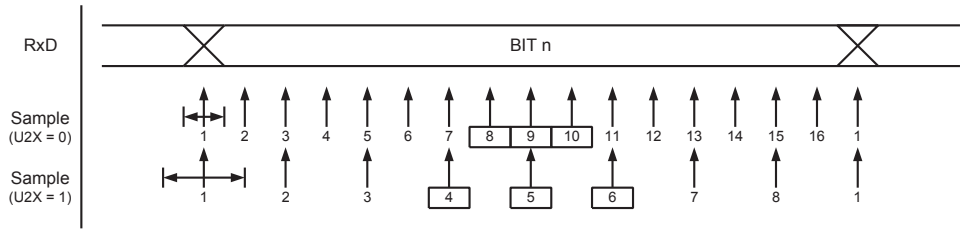


当时钟恢复电路检测到 RxD 线上一个由高 (空闲) 到低 (开始) 的电平跳变时，起始位检测序列即被启动。如图所示，我们用采样 1 表示第一个 0 采样。然后，时钟恢复逻辑用采样 8、 9、 10(普通模式)，或采样 4、 5、 6(倍速模式)，来判断是否接收到一个正确的起始位。如果这三个采样中的两个或更多个是逻辑高电平 (多数表决)，起始位会被视为毛刺噪声而被拒绝接受，接收器等待下一个由高到低的电平转换。如果检测到一个有效的起始位，时钟恢复逻辑即被同步并开始接收数据。每一个起始位都会引发同样的同步过程。

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 73 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。

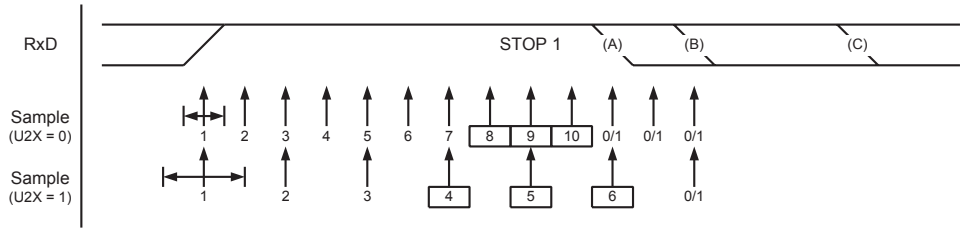
Figure 73. 数据及奇偶位的采样



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 74 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 74. 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 74 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 72)，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。

$$R_{slow} = \frac{(D+1)S}{S-1+D\cdot S+S_F} \qquad R_{fast} = \frac{(D+2)S}{(D+1)S+S_M}$$

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 72 和 Table 73 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 72. 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 73. 倍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。

多处理器通信模式

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。

USART 寄存器描述

USART I/O 数据寄存器—UDR

位	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

位	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	UPE	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	

- **位 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)。

- **位 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)。

- **位 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。

- **位 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。

- **位 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。

- **位 2 – UPE: USART 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。

- **位 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。

- **位 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 P160 “ 多处理器通信模式 ”。

USART 控制和状态寄存器 B — UCSRB

位	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	

- **位 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。

- **位 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。

- **位 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。

- **位 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 UPE 标志无效。

- **位 3 – TXEN: 发送使能**

置位后将启动将启动 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。

- **位 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)。

- **位 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。

- **位 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

位	7	6	5	4	3	2	1	0	
	–	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
读 / 写	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	1	1	0	

- **位 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 74. UMSEL 设置

UMSEL	模式
0	异步操作
1	同步操作

- **位 5:4 – UPM1:0: 奇偶校验模式**

这几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 UPE 置位。

Table 75. 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

- **位 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 76. USBS 设置

USBS	停止位位数
0	1
1	2

- **位 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 77. 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

- **位 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 78. 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

位	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
读 / 写	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

- **位 15:12 – 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。

- **位 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 79 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 P158 “ 异步工作范围 ”)。误差可以通过如下公式计算：

$$\text{Error}[\%] = \left(\frac{\text{BaudRate}_{\text{Closest Match}}}{\text{BaudRate}} - 1\right) \bullet 100\%$$

Table 79. 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 80. 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 81. 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 82. 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%

通用串行接口—USI

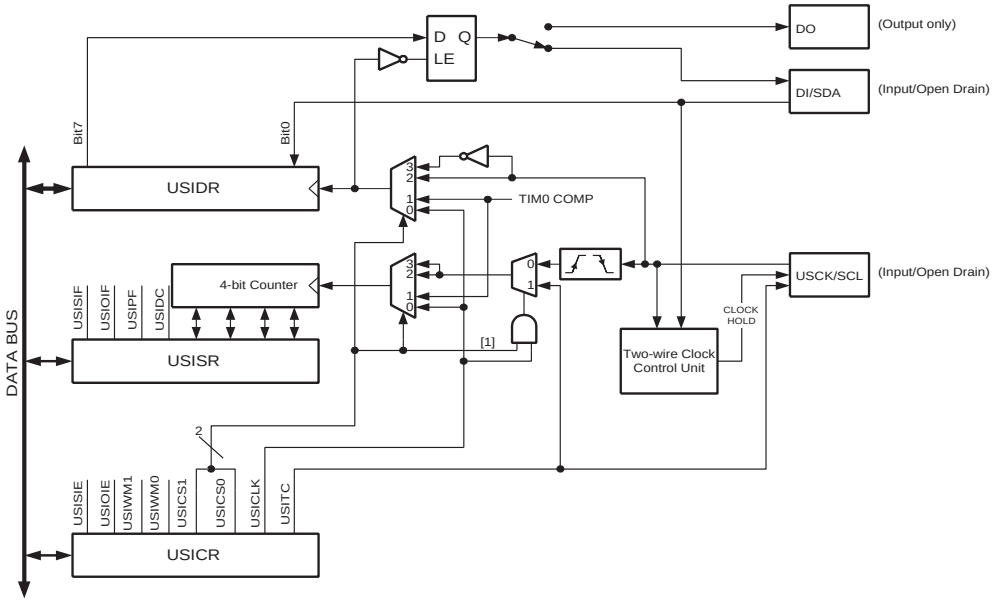
T 通用串行接口，或 USI，提供了进行串行通信所需的最基本的硬件资源。和最小化的控制软件结合之后，USI 可以提供比使用纯软件高得多的传输速率并使用更少的代码空间。可以使用中断来最小化处理器的工作量。 USI 的主要特性是：

- **两线同步数据传输 (主机或从机，$f_{SCLmax} = f_{CK}/16$)**
- **三线同步数据传输 (主机或从机，$f_{SCKmax} = f_{CK}/4$)**
- **数据接收中断**
- **可以从空闲模式唤醒**
- **两线模式下可从所有的睡眠模式唤醒，包括掉电模式**
- **两线启动条件检测器有中断能力**

概述

Figure 75.展示了一个简化的USI框图。实际的I/O管脚请参看P2 “ATmega169引脚排列” 。CPU 可寻址的 I/O 寄存器，包括位和引脚，以粗体显示。器件专用 I/O 寄存器及位说明在 P176 “USI 寄存器描述 ” 中列出。

Figure 75. 通用串行接口框图



通过数据总线可直接访问 8 位的移位寄存器，这个寄存器包含正在发送和接收的数据。由于这个寄存器没有缓冲，因此要尽快读出其中的数据以保证数据不丢失。依据不同的工作模式，最高位与两个输出引脚之一相连。在串行寄存器输出和输出引脚之间有一个透明的锁存器，它的作用是将数据输出延迟到和用于数据输入采样的时钟沿相反的时钟沿。串行输入总是经过数据输入引脚 (DI) 进行采样，而与配置无关。

通过数据总线还可以访问 4 比特的计数器，并产生溢出中断。串行寄存器和计数器由相同的时钟驱动，从而定时器可以对接收或发送的比特数进行计数，并在传输结束时产生中断。当选择了外部时钟时，计数器将对时钟的上下两个沿进行计数。此时计数器统计沿的数目而不是比特数。有三种可选的时钟源：USCK 引脚、定时器 / 计数器 0 比较匹配或通过软件产生。

检测到起始条件后，两线时钟控制单元可以产生中断。它也可以在检测到起始条件或计数器溢出之后通过拉低时钟引脚来产生等待状态。

功能描述

三线模式

USI的三线工作模式与串行外设接口(SPI)模式0和模式1兼容，但没有从机选择(SS)功能。不过这可以通过软件来实现。这种模式下的引脚名称为 DI、 DO 和 USCK。

Figure 76. 三线模式简化框图

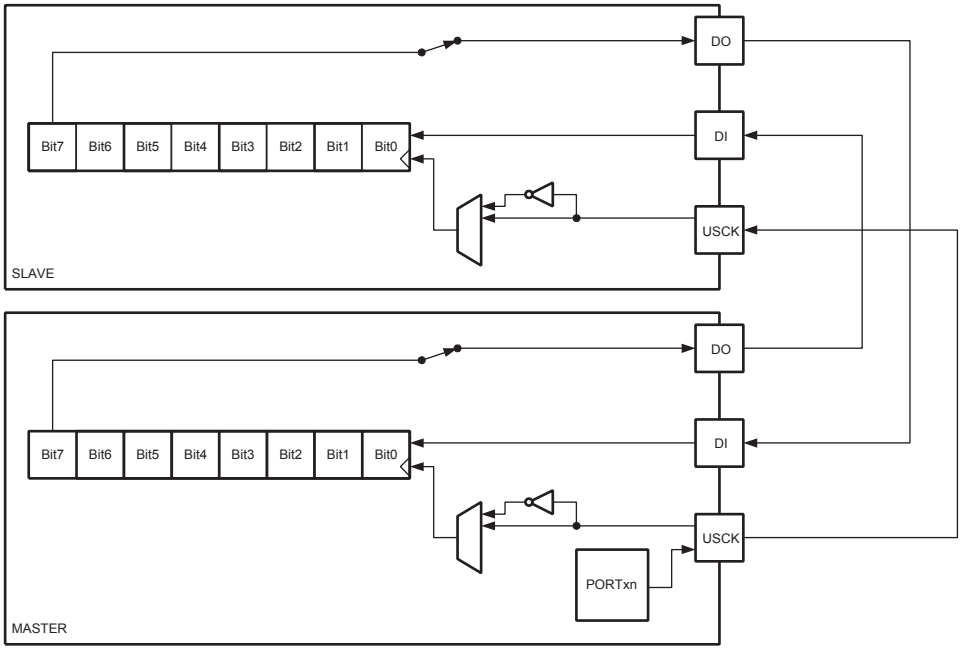


Figure 76 给出了两个工作于三线模式的 USI 单元，一个为主机，另一个为从机。两个移位寄存器的连接方式使得 8 个 USCK 时钟之后，两个寄存器中的数据相互交换。同样的时钟还驱动 USI 的 4 位计数器。因此计数器溢出 (中断) 标志 USIOI 可用来判断传输何时完成。这个时钟可以由两种方式产生 : 一是由主机软件通过操作端口寄存器来操作 USCK 引脚，二是置位 USICR 寄存器的 USITC。

Figure 77. 三线模式时序图

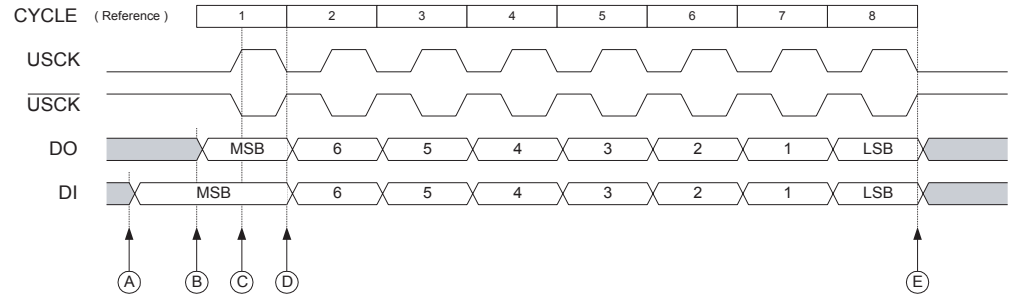


Figure 77. 给出了三线模式的时序。图的顶端是 USCK 的参考周期。在每一个这样的周期里都有一个比特的数据转移到 USI 移位寄存器 (USIDR) 中。 USCK 时序展示了两种外部时钟模式。工作于外部时钟模式 0(USICS0 = 0) 时， DI 在时钟的上升沿采样，输出 DO 在下降沿改变 (数据寄存器移动一位)，外部时钟模式 1(USICS0 = 1) 使用与模式 0 相反的时钟沿，即在在下降沿进行数据采样，在上升沿改变输出。USI 时钟模式对应于 SPI 数据模式 0 和模式 1。

由 Figure 77. 时序图可以看出，总线传输包括以下步骤 :

1. 通过向串行数据寄存器中写入需要发送的数据来准备数据输出。通过设置数据方向寄存器中的对应位来启动数据输出。注意， A 与 B 点之间并没有什么特殊的顺

序，但是它们都必须早于数据采样点 C 点之前至少半个 USCK 周期。这点必须得到保障，以保证数据准备所需条件得到满足。 4 位计数器被重置为 0。

2. 主机通过软件改变USCK两次(C与D)来产生一个时钟脉冲。主从设备的输入引脚(DI)上的值由USI在第一个沿(C)进行采样；数据输出则在其相对的沿(D)改变。4位计数器将对两个沿进行计数。
3. 在一个完整的寄存器 (字节) 传输过程中，第 2 步将重复 8 次。
4. 8 个时钟脉冲 (16 个时钟沿) 之后 ,计数器溢出 ,表明传输完成。传输的数据必须在下一次传输开始之前得到处理。如果处理器处于空闲模式，那么溢出中断会将它唤醒。依据通讯协议，从机现在可以将它的输出置为高阻状态。

SPI 主机工作例子

接下来的代码说明了如何将 USI 模块当作 SPI 主机来使用：

```
SPITransfer:
    sts     USIDR,r16
    ldi     r16,(1<<USIOIF)
    sts     USISR,r16
    ldi     r16,(1<<USIWM0)|(1<<USICS1)|(1<<USICLK)|(1<<USITC)
SPITransfer_loop:
    sts     USICR,r16
    lds     r16, USISR
    sbrs    r16, USIOIF
    rjmp    SPITransfer_loop
    lds     r16,USIDR
    ret
```

这段代码仅使用了 8 条指令 (+ ret)，非常优化。示例代码假定 DO 及 USCK 引脚已经通过设置 DDRE 寄存器成为输出引脚。调用函数之前， r16 寄存器包含了要送到从机的数据，传输结束之后， r16 寄存器包含了从从机接收回来的数据。

第二和第三条指令是清 USI 计数器溢出标志及 USI 计数器。第四第五条指令设置三线模式、上升沿移位寄存器时钟、在 USITC 选通时进行计数以及触发 USCK。循环将重复运行 16 次。

下面的代码则演示了在最大速率 (fsck = fck/4) 下如何将 USI 模块作为 SPI 主机来使用：

```
SPITransfer_Fast:

    sts     USIDR,r16
    ldi     r16,(1<<USIWM0)|(0<<USICS0)|(1<<USITC)
    ldi     r17,(1<<USIWM0)|(0<<USICS0)|(1<<USITC)|(1<<USICLK)

    sts     USICR,r16 ; MSB
    sts     USICR,r17
    sts     USICR,r16
    sts     USICR,r17
    sts     USICR,r16
    sts     USICR,r17
    sts     USICR,r16
    sts     USICR,r17
    sts     USICR,r16
    sts     USICR,r17
    sts     USICR,r16
    sts     USICR,r17
    sts     USICR,r16
    sts     USICR,r17
    sts     USICR,r16 ; LSB
    sts     USICR,r17

    lds     r16,USIDR
ret
```

SPI 从机工作例子

接下来的代码展示了如何将 USI 模块作为 SPI 从机来使用：

```
init:
    ldi     r16,(1<<USIWM0)|(1<<USICS1)
    sts     USICR,r16
...
SlaveSPITransfer:
    sts     USIDR,r16
    ldi     r16,(1<<USIOIF)
    sts     USISR,r16
SlaveSPITransfer_loop:
    lds     r16, USISR
    sbrs    r16, USIOIF
    rjmp    SlaveSPITransfer_loop
    lds     r16,USIDR
    ret
```

这段代码仅使用了 8 条指令 (+ ret)，非常优化。示例代码假定 DO 及 USCK 引脚已经通过设置 DDRE 寄存器分别成为输出和输入引脚。调用函数之前，r16 寄存器包含了要送到主机的数据，传输结束之后，r16 寄存器包含了从主机接收回来的数据。

开头的两条指令为初始化指令，仅需执行一次。这些指令用来设置三线模式及上升沿沿移位寄存器时钟。循环一直重复到 USI 计数器溢出标志位置位。

两线模式

USI 两线模式兼容 IC 间 (TWI) 总线协议，但没有输出转换速率限制及输入噪声滤波器。这种模式下的引脚名为 SCL 和 SDA。

Figure 78. 两线模式框图

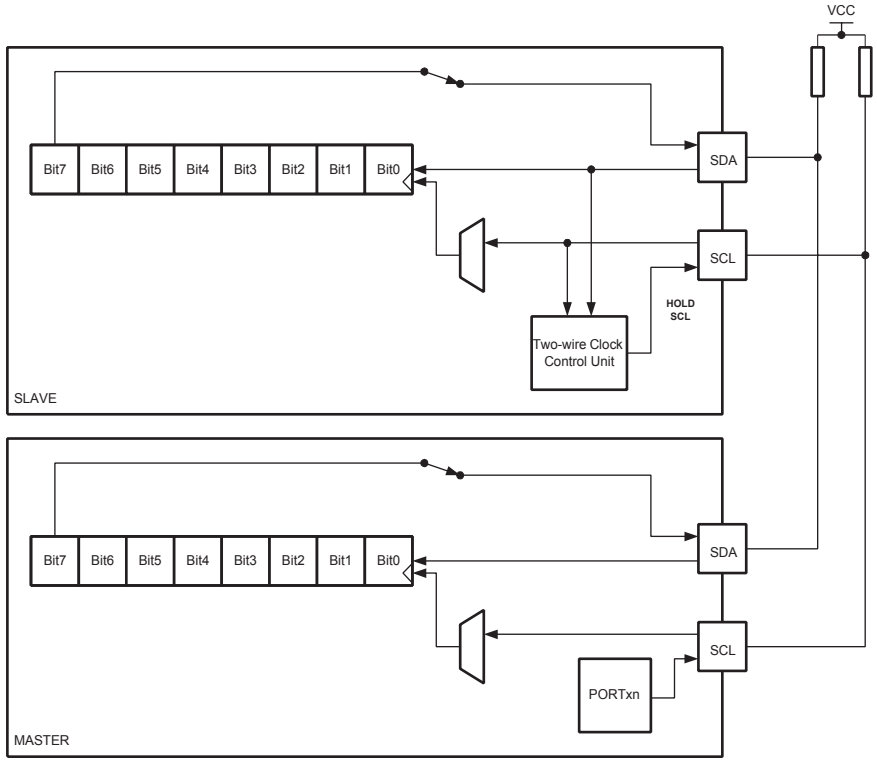
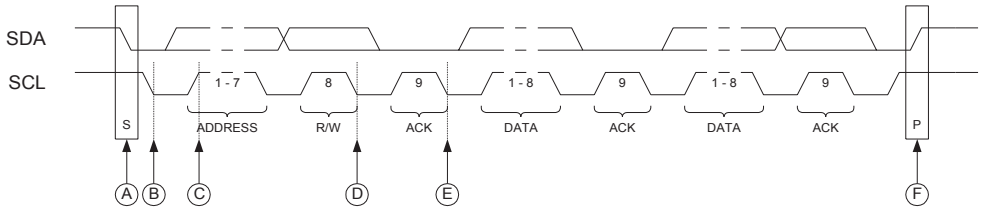


Figure 78 展示了两个工作在两线模式下的 USI 单元，一个为主机，另一个为从机。由于系统的工作在很大程度上取决于所使用的通信方案，这里仅仅给出了物理层。在这一层中，主机和从机的主要区别是：串行时钟由主机产生，只有从机使用时钟控制单元。时钟的产生必须由软件实现，但移位操作是自动完成的。数据传输过程中只有下降沿触发数据移位。通过强制 SCL 时钟为低，从机可以在传输启动或结束时插入等待状态。这说明，产生上升沿之后，主机必须检查 SCL 线是否已经释放。

由于时钟同时驱动计数器计数，计数器溢出可用来判断传输是否结束。主机通过 PORT 寄存器使 USCK 引脚产生时钟。

物理层不确定数据的方向。数据流控制由协议决定，如 TWI 总线协议。

Figure 79. 两线模式典型时序图



根据此时序图 (Figure 79.)，总线传输包括以下步骤：

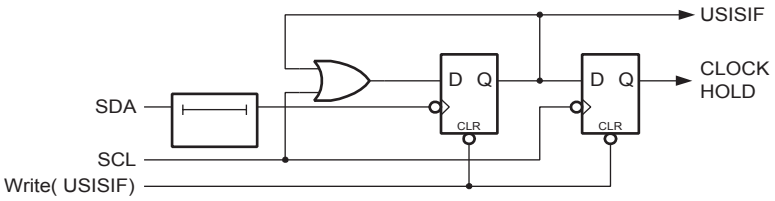
1. 当SCL为高时(A)，主机可通过将SDA强制拉低来产生一个起始条件。有两种方法可使 SDA 强制为低：一种是对移位寄存器的第 7 位写 0，另一种是将 PORT 寄存器中的对应位置 0。在此之前还需要通过数据方向寄存器将相关引脚设置为输

出。从机起始条件监测逻辑 (Figure 80.) 检测到起始条件后置位 USISIF 标志。如果有必要，可通过此标志产生中断。

2. 此外，当主机强制在 SCL 线产生下降沿 (B) 后，起始条件检测器将保持 SCL 为低。这可将从机从睡眠状态唤醒，或在设置移位寄存器接收地址之前完成其他任务。这个操作通过清起始条件标志及复位计数器来实现。
3. 主机设置第一个需要传输的位，并释放 SCL 线 (C)。从机在 SCL 时钟上升沿对数据进行采样并将数据转移到串行寄存器中。
4. 当包含从机地址及数据方向 (读或写) 的 8 位数据全都传输完完毕之后，从机计数器溢出， SCL 被强制置为低 (D)。如果这个从机不是主机所寻址的设备，它将释放 SCL 线并等待下一个起始条件。
5. 如果从机被寻址，那么在 SCL 被再次拉低之前 (在释放 SCL(D) 之前计数器要达到 14)，在应答期间它将保持 SDA 线为低。 R/W 位决定是主机还是从机输出数据。若 R/W 为 1，主机执行读操作 (即从设备驱动 SDA 线)。在应答 (E) 之后从机可以保持 SCL 线为低。
6. 现在可以在同一方向传输多个字节的数据了，直到主机发出停止条件 (F)，或者产生一个新的起始条件。

如果从机无法接收更多的数据，那么它不要应答最后接收到的数据。主机进行读操作时，接收到最后一个字节后，它必须强制应答位为低来结束读操作。

Figure 80. 起始条件监测器逻辑电路图



起始条件监测器

Figure 80. 为起始条件监测器。SDA 被延迟 (50 到 300 ns) , 以保证 SCL 的有效采样。起始条件监测器仅在两线模式下使能。

起始条件监测器工作在异步模式，因此可以将处理器从掉电睡眠模式唤醒。但是，通讯协议可能对 SCL 的保持时间有限制。在这种情况下就必须考虑由 CKSEL 熔丝位确定的晶振启动时间 (见 P23 “ 时钟系统及其分布 ”)。更多细节请参见 176 页对 USISIF 位的描述。

其他的 USI 用法

如果 USI 不用于串行通讯，则由于其设计上的灵活性，可以用来完成其他工作。

半双工异步数据传输

在三线模式下使用移位寄存器可以实现比软件方案更紧凑、更高效的 UART 功能。

4 位计数器

这个 4 比特计数器可作为独立的计数器来使用，并可产生溢出中断。要注意的是，如果计数器由外部时钟源提供时钟，那么两个时钟边沿都会使其计数。

12 位定时器 / 计数器

将 USI 的 4 比特计数器与定时器 / 计数器 0 结合起来使用可得到一个 12 位的计数器。

边沿触发的外部中断

把计数器的值设到最大 (F)，可实现一个额外的外部中断。溢出标志位及中断使能位都为此外部中断服务。可通过 USICS1 来选择这一特性。

软件中断

计数器溢出中断还可用作软件中断。

USI 寄存器描述

USI 数据寄存器—USIDR

位	7	6	5	4	3	2	1	0	
	MSB							LSB	USIDR
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	

USI 串行寄存器没有缓冲，寻址数据寄存器 USIDR 时，实际操作的是串行寄存器。若在写寄存器的同一个周期产生了一个串行时钟，寄存器将被更新，但不会进行移位操作。移位操作依赖于 USICS1..0 的设置，可由外部时钟沿、定时器 / 计数器 0 比较匹配或直接通过软件使用 USICLK 来控制。即使没有选择任何连接模式 (USIWM1..0 = 0)，移位寄存器仍然可以使用外部数据输入 (DI/SDA) 及外部时钟输入 (USCK/SCL)。

输出引脚 (DO 或 SDA，由连接模式确定) 通过输出锁存器与数据寄存器的最高位 (位 7) 相连。选择了外部时钟源时 (USICS1 = 1)，输出锁存器在串行时钟的前半个周期打开 (透明)，如果使用的是内部时钟源 (USICS1 = 0)，它将一直打开。锁存器打开时写入新的 MSB 会立即在输出引脚反映出来。锁存器保证输入数据的采样时间与输出数据的改变发生在相反的时钟沿。

为了输出移位寄存器的数据，必须利用方向寄存器将对应的引脚设置为输出。

USI 状态寄存器—USISR

位	7	6	5	4	3	2	1	0	
	USISIF	USIOIF	USIPF	USIDC	USICNT3	USICNT2	USICNT1	USICNT0	USISR
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	

状态寄存器包括中断标志、线状态标志及计数器值。

- **位 7 – USISIF: 起始状态中断标志**

在两线模式下，检测到起始条件将置位 USISIF 标志。当使用输出禁止模式或三线模式，并且 (USICSx = 0b11 & USICLK = 0) 或 (USICS = 0b10 & USICLK = 0) 时， SCK 引脚的任意电平变化都会使此标志置位。

USISIF 置位时，若 USICR 寄存器的 USISIE，以及全局中断使能标志也置位，则会产生中断。 USISIF 标志位清零的唯一方式是对其写入逻辑 1。在两线模式下清除这一标志会释放由于检测到起始条件而保持的 USCL。

起始条件中断可把处理器从所有睡眠状态唤醒。

- **位 6 – USIOIF: 计数器溢出中断标志**

4位计数器溢出(即从15跳变到0)时此标志置位。若USICR寄存器的USIOIE，以及全局中断使能标志也置位，则会产生中断。 USIOIF 标志位清零的唯一方式是对其写入逻辑 1。在两线模式下清这一标志会释放由于计数器溢出而保持的 SCL。

溢出中断可以将处理器从所有睡眠模式唤醒。

- **位 5 – USIPF: 停止状态标志**

在两线模式下，如果检测到停止状态， USIPF 标志置位。 USIOIF 标志位清零的方法是对其写入逻辑 1。注意这不是一个中断标志位。在进行总线主机仲裁时，可使用这个标志。

- **位 4 – USIDC: 数据输出冲突**

如果移位寄存器中的位 7 与物理引脚所对应的值不同， USIDC 置位。此标志仅在两线模式下有效。在进行总线主机仲裁时，可使用这个标志。

- **位 3..0 – USICNT3..0: 计数器值**

反映的是 4 位计数器的当前值。 CPU 可以直接读写这几位。

使计数器计数的时钟源有外部时钟边沿检测器、定时器 / 计数器 0 比较匹配及通过软件使用 USICLK 或 USITC 产生的时钟源。时钟源的确定由 USICS1..0 设置。外部时钟操作另有一个特性，即可以通过写 USITC 来产生时钟。方法是设置外部时钟源 (USICS1 = 1) 时将 USICLK 置 1。

即使没有选择任何连接模式 (USIWM1..0 = 0)，计数器仍然可以使用外部时钟输入 (USCK/SCL)。

USI 控制寄存器—USICR

位	7	6	5	4	3	2	1	0	
	USISIE	USIOIE	USIWM1	USIWM0	USICS1	USICS0	USICLK	USITC	USICR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	W	W	
初始值	0	0	0	0	0	0	0	0	

控制寄存器包括中断使能控制、连接模式设置、时钟选择设置及时钟选择信号。

- **位 7 – USISIE: 起始条件中断使能**

将此位置 1 可使能起始条件检测中断。如果 USISIE 及全局中断使能标志位置位时产生了一个这样的中断，那么中断将立即得到处理。更多的细节请参见 176 页对 USISIF 的描述。

- **位 6 – USIOIE: 计数器溢出中断使能**

. 此标志为 1 可产生计数器溢出中断。如果 USIOIE 及全局中断使能标志位置位时产生了一个这样的中断，那么中断将立即得到处理。更多的细节请参见 177 页对 USIOIF 位的描述。

- **位 5..4 – USIWM1..0: 连接模式**

这几位用来设置连接模式。基本上只有输出功能受这几位的影响。数据及时钟输入不受所选模式的影响，总是保持同样的功能。因此，即使输出被禁止，计数器和移位寄存器照样可由外部提供时钟，也可以进行数据输入采样。 USIWM1..0 与 USI 操作的关系在 Table 83 中有简要介绍。

Table 83. USIWM1..0 与 USI 操作之间的关系

USIWM1	USIWM0	描述
0	0	输出，时钟保持，起始检测器禁止。引脚以普通端口方式工作。.
0	1	三线模式。使用 DO、 DI 及 USCK 引脚 在这种模式下*数据输出* (DO) 功能取代了普通端口 IO 功能，但对应的 DDR 仍然控制这数据方向。端口引脚设置为输入时，引脚的上拉电阻由 PORT 位来控制。 *数据输入* (DI) 及*串行时钟* (USCK) 功能不影响正常的端口功能。作为主机工作时，软件通过操作 PORT 寄存器来产生时钟脉冲，同时数据方向设为输出。 USICR 寄存器中的 USITC 位可用作这一目的。
1	0	两线模式。使用 SDA (DI) 及 SCL (USCK) 引脚 (1)。 *串行数据* (SDA) 及*串行时钟* (SCL) 引脚是双向的，且使用集电极开路输出驱动器。通过设置 DDR 寄存器中相应的位来启动输出驱动器。 SDA 引脚的驱动器使能时，如果移位寄存器的输出或 PORT 寄存器对应的位为 0，那么输出驱动器会把 SDA 线强制拉低。否则 SDA 线将不被驱动 (即将它释放)。 SCL 引脚输出驱动器使能时，如果 PORT 寄存器中的对应位为 0，或者由于起始检测器的作用， SCL 线被强制置低。否则 SCL 线将不被驱动。 当起始检测器检测到起始条件且输出允许时， SCL 被拉低。清起始条件标志 (USISIF) 将释放此口线。启动这一模式不会影响 SDA 及 SCL 引脚的输入。在两线模式下 SDA 及 SCL 引脚的上拉无效。
1	1	两线模式。使用 SDA 及 SCL 引脚 两线模式下的一些操作在上面已有所描述，除了以下这点：当发生计数器溢出时 SCL 线保持为低，并一直保持到计数器溢出标志位 (USIOIF) 被清零。

Note: 1. 为了避免混淆，DI 及 USCK 引脚分别被重命名为*串行数据* (SDA) 及*串行时钟* (SCL)。

- **位 3..2 – USICS1..0: 时钟源选择**

通过这几位可以设置移位寄存器及计数器的时钟源。数据输出锁存器保证在使用外部时钟 (USCK/SCL) 时，输出数据的改变与输入数据 (DI/SDA) 的采样发生在相反的时钟沿。如果选择了软件方式或定时器 / 计数器 0 比较匹配作为时钟，输出锁存器即成为是透明的，输出可以立即改变。 USICS1..0 为 0 时软件方式使能。此时，向 USICLK 写 1 就可以同时给移位寄存器和计数器提供时钟。对于外部时钟 (USICS1 = 1)， USICLK 位不再用作选通信号，而是通过 USITC 在外部时钟及软件时钟之间进行选择。

Table 84 给出了 USICS1..0 及 USICLK 的设置与移位寄存器及 4 位计数器所使用的时钟之间的关系。

Table 84. USICS1..0 及 USICLK 的设置

USICS1	USICS0	USICLK	移位寄存器时钟源	4 位计数器时钟源
0	0	0	无时钟	无时钟
0	0	1	软件时钟 (USICLK)	软件时钟 (USICLK)
0	1	X	定时器 / 计数器 0 比较匹配	定时器 / 计数器 0 比较匹配
1	0	0	外部时钟，上升沿	外部时钟，上升及下降沿
1	1	0	外部时钟，下降沿	外部时钟，上升及下降沿
1	0	1	外部时钟，上升沿	软件时钟 (USITC)
1	1	1	外部时钟，下降沿	软件时钟 (USITC)

- **位 1 – USICLK: 时钟选通**

若 USICS1..0 为 0，置位 USICLK 将使移位寄存器进行一次移位，计数器累加一。此即为软件时钟。在同一指令周期里输出立即得到更新。移入移位寄存器的值在上一个指令周期得以采样。这一位的读出值为 0。

使用外部时钟时 (USICS1 = 1)， USICLK 的功能由时钟选通变为时钟选择寄存器。在这种情况下设置 USICLK 将选择 USITC 作为软件时钟源来驱动 4 位计数器 (见 Table 84)。

- **位 0 – USITC: 交替变换时钟端口引脚**

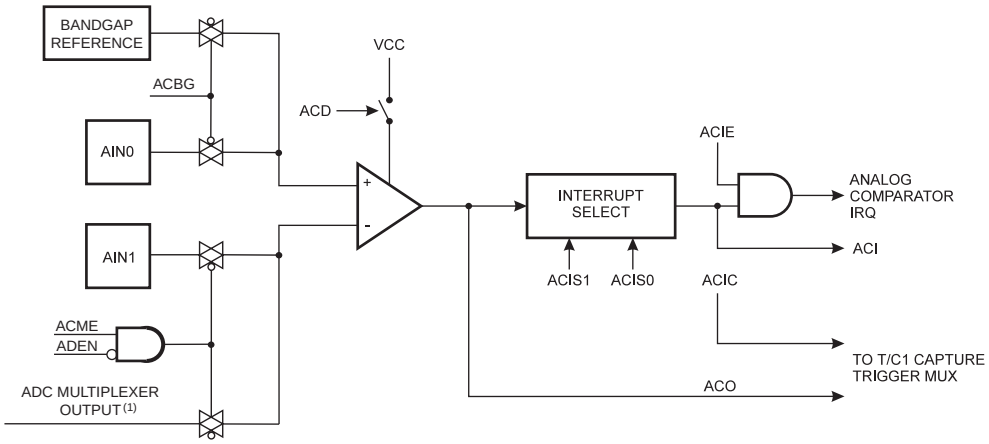
USITC 置位将使 USCK/SCL 出现 0、1 的交替变换。方向寄存器的设置不影响触发，但需要置位 DDRE4 使相应的端口成为输出。这个特性为主机实现提供了一个产生时钟的简单办法。这一位的读返回值为 0。

选用外部时钟 (USICS1 = 1) 并且 USICLK 置 1 时，对 USITC 进行写入将直接驱动 4 位计数器。作为主机工作时，这样可以更早地知道传输何时结束。

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 81 为比较器及其外围逻辑电路的框图。

Figure 81. 模拟比较器框图 (2)



Notes: 1. 见 P182 Table 86 。
2. 模拟比较器的管脚分布见 P2 Figure 1 及 P58Table 28 。

ADC 控制及状态寄存器 B—ADCSRB

位	7	6	5	4	3	2	1	0	
	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
读 / 写	R	R/W	R	R	R	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 6 – 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 P183 “ 模拟比较器多路输入 ”。

模拟比较器控制及状态寄存器—ACSR

位	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	

- **位 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。

- **位 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固定能隙基准源所取代。否则， AIN0 连接到模拟比较器的正极输入。见 P41 “ 片内基准电压 ”。

- **位 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。

- **位 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 来清零。

- **位 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。

- **位 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定时器 / 计数器 1 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 TIMSK1 的 ICIE1 必须置位。

- **位 1, 0 – ACIS1, ACIS0: 模拟比较器中断模式选择**

这两位确定哪个事件可以触发模拟比较器中断。 Table 85 给出了不同的设置。

Table 85. 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。

模拟比较器多路输入

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 (ADCSRB 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 86。如果 ACME 为 0 或 ADEN 置位，则模拟比较器的负极输入为 AIN1。

Table 86. 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

数字输入禁止寄存器 1—DIDR1

位	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	AIN1D	AIN0D	DIDR1
读 / 写	R	R	R	R	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **位 1, 0 – AIN1D, AIN0D: AIN1, AIN0 数字输入禁止**

AIN1D 和 AIN0D 置 1 后 ,AIN1/0 引脚的数字输入缓冲器被禁止 ,PIN 寄存器的读返回值为 0。当 AIN1/0 引脚加载了模拟信号 ,且当前应用不需要 AIN1/0 引脚的数字输入缓冲器时，AIN1D 和 AIN0D 应该置位以降低数字输入缓冲的功耗。

模数转换器

特点

- **10 位 精度**
- **0.5 LSB 的非线性度**
- **± 2 LSB 的绝对精度**
- **13 µs - 260 µs 的转换时间 (50kHz 到 1MHz 的 ADC 时钟)**
- **最高分辨率 (200kHz 的 ADC 时钟) 时采样率高达 15 kSPS**
- **8 路复用的单端输入通道**
- **可选的向左调整 ADC 读数**
- **0 - V_{CC} 的 ADC 输入电压范围**
- **可选的 1.1V 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**

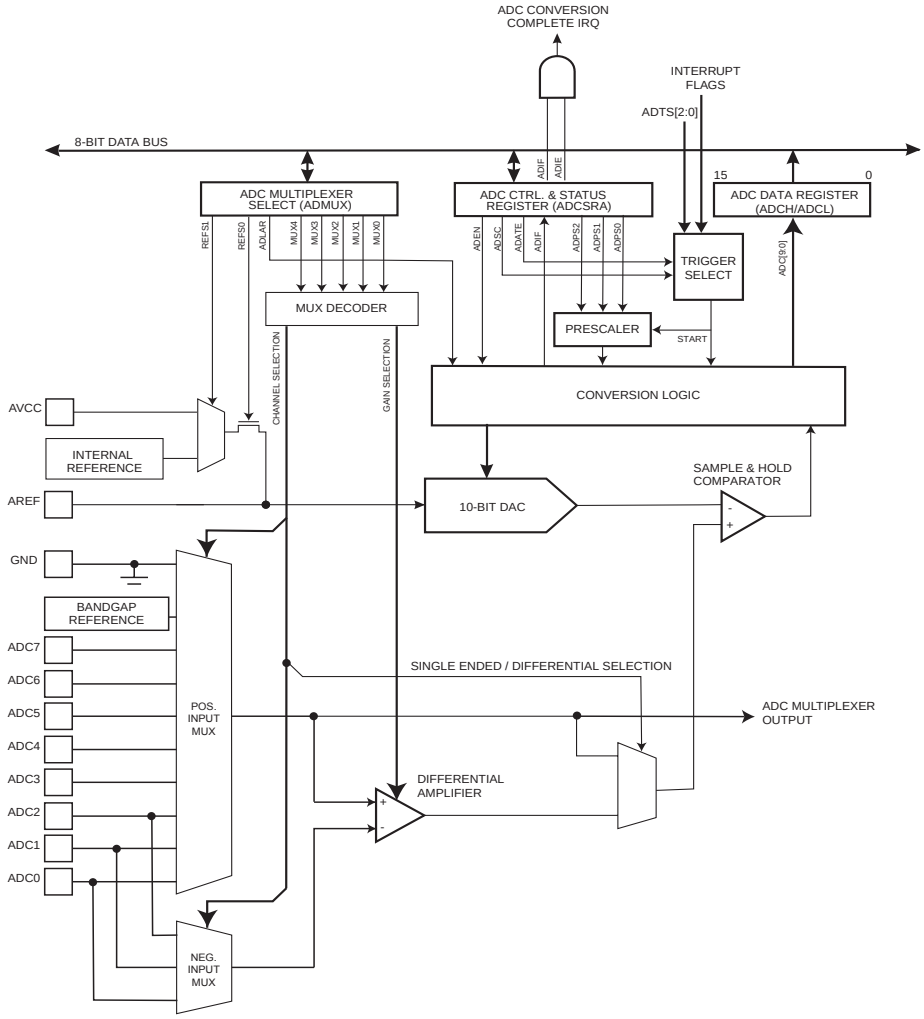
ATmega169有一个10位的逐次逼近型ADC。ADC与一个8通道的模拟多路复用器连接，能对来自端口 A 的 8 路单端输入电压进行采样。单端电压输入以 0V (GND) 作为基准。

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 82 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P190 “ADC 噪声抑制器 ” 来了解如何连接这个引脚。

标称值为 1.1V 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 82. 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 1.1V 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。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。

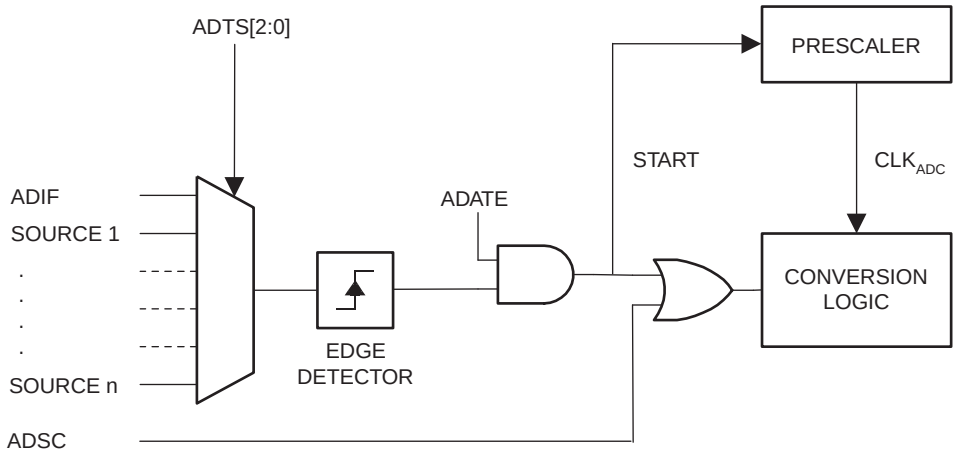
ADC转换结束可以触发中断。即使由于转换发生在读取ADCH与ADCL之间而造成ADC无法访问数据寄存器，并因此丢失了转换数据，中断仍将触发。

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 83. ADC 自动触发逻辑

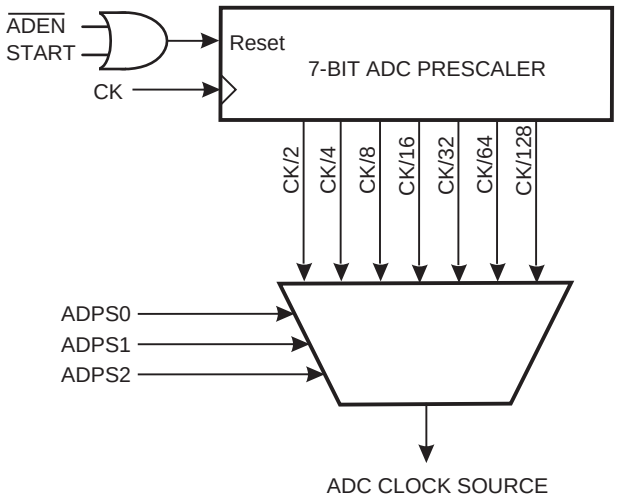


使用 ADC 中断标志作为触发源，可以在正在进行的转换结束后即开始下一次 ADC 转换。之后 ADC 便工作在连续转换模式，持续地进行采样并对 ADC 数据寄存器进行更新。第一次转换通过向 ADCSRA 寄存器的 ADSC 写 1 来启动。在此模式下，后续的 ADC 转换不依赖于 ADC 中断标志 ADIF 是否置位。

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 84. 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。

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。如果使用差分模式，加上不是由 ADC 转换结束实现的自动触发，每次转换需要 25 个 ADC 时钟周期。因为每次转换结束后都要关闭 ADC 然后又启动它。

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 87。

Figure 85. ADC 时序图，第一次转换（单次转换模式）

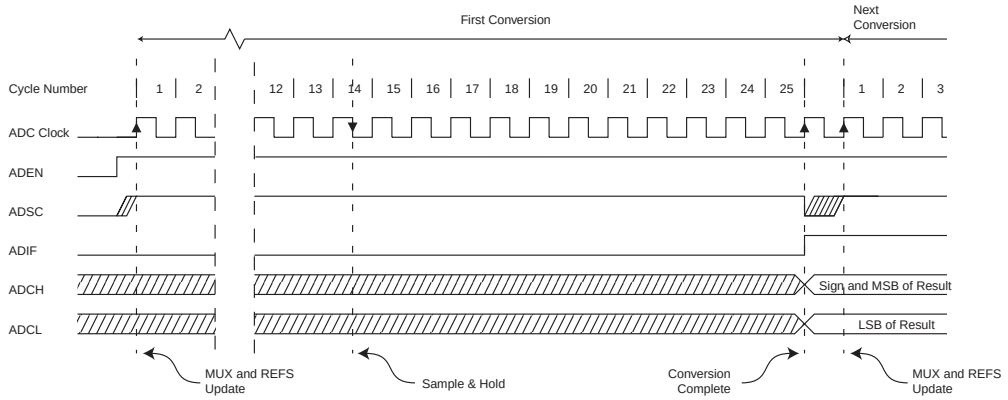


Figure 86. ADC 时序图，单次转换

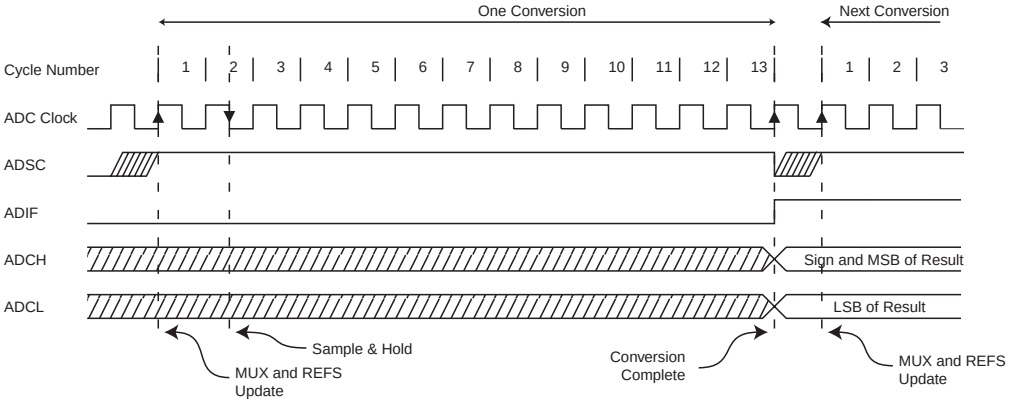


Figure 87. ADC 时序图，自动触发的转换

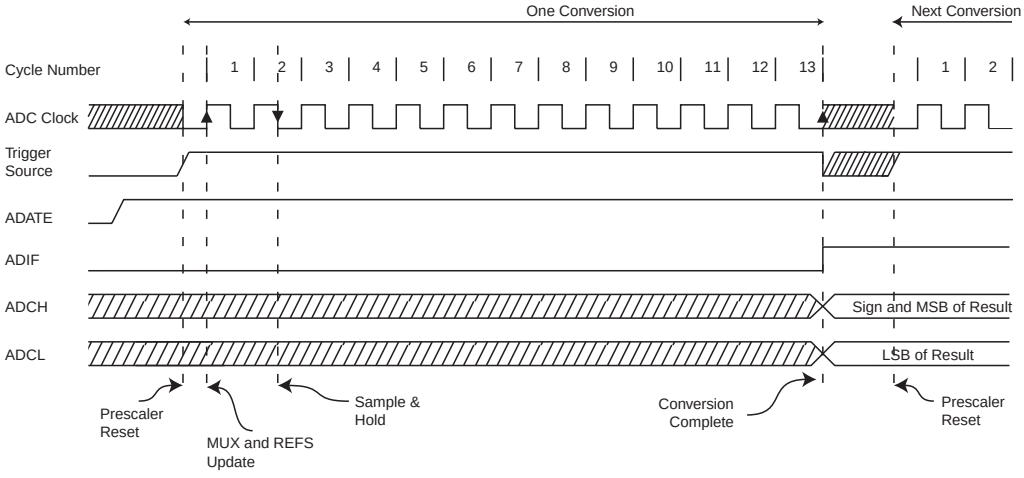


Figure 88. ADC 时序图，连续转换

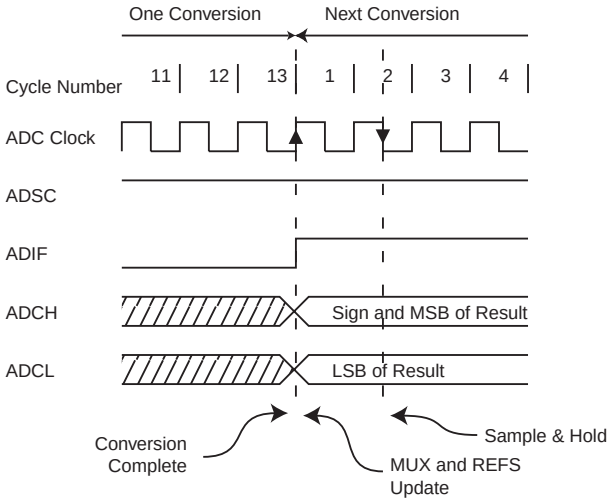


Table 87. ADC 转换时间

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
自动触发的转换	2	13.5

通道改变及基准源选择

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。

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。

ADC 参考电压源

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 1.1V 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1.1V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 1.1V 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。

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。

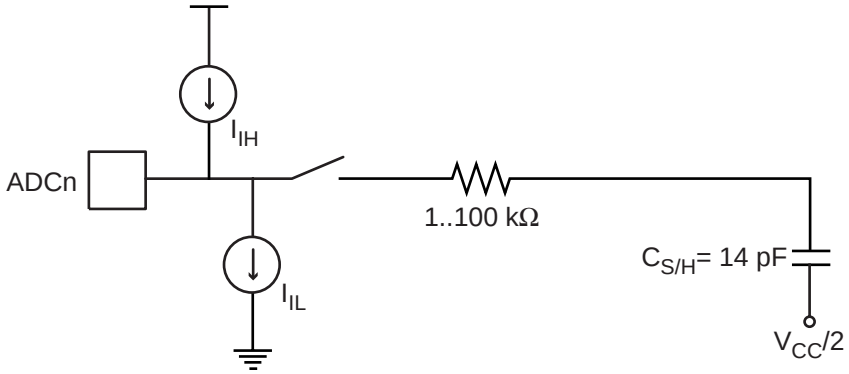
模拟输入电路

单端通道的模拟输入电路见 Figure 89.。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。

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 89. 模拟输入电路

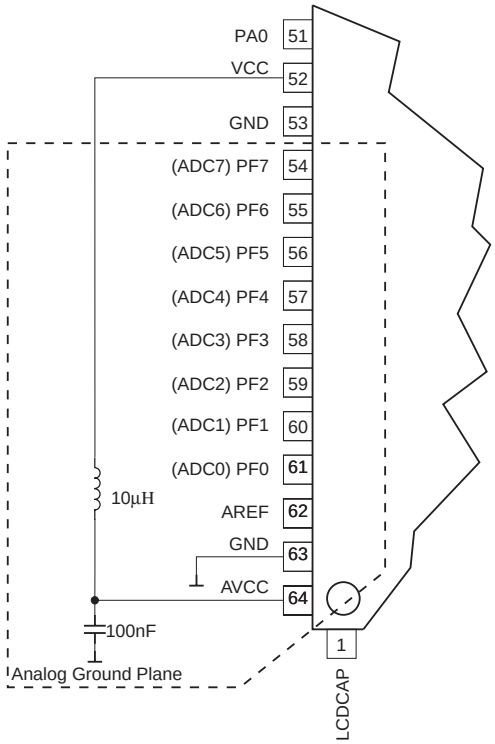


模拟噪声抑制技术

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，并使它们与高速转换切换的数字信号线分开。
2. 如 Figure 90 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 90. ADC 电源连接图



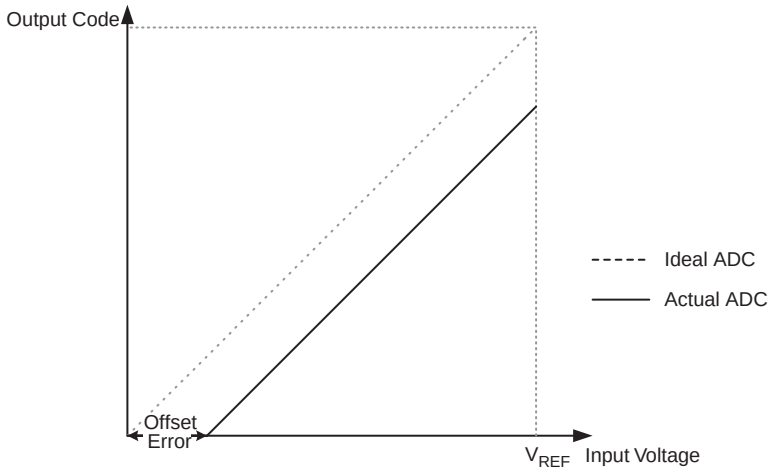
ADC 精度定义

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：

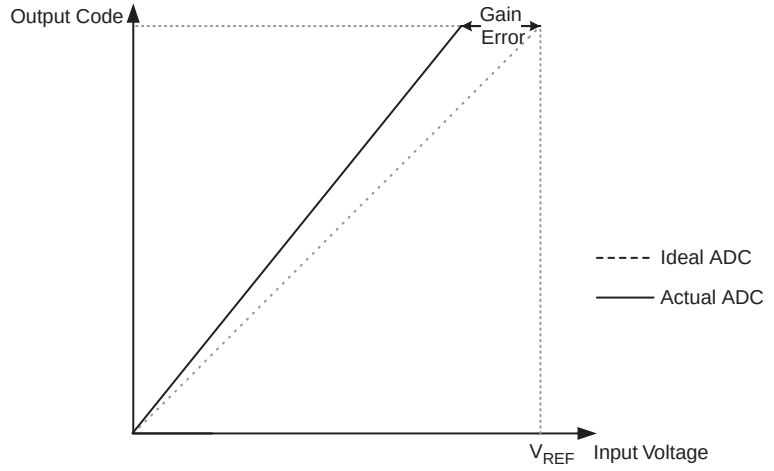
- 偏移：第一次转换 (0x000 到 0x001) 与理想转换 (0.5 LSB) 之间的偏差。理想情况：0 LSB。

Figure 91. 偏移误差



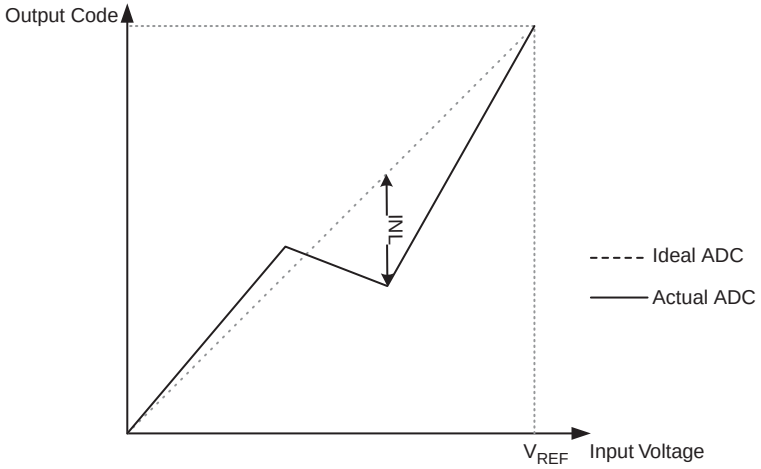
- 增益误差：调整偏差之后，最后一次转换 (0x3FE 到 0x3FF) 与理想情况 (最大值以下 1.5 LSB) 之间的偏差即为增益误差。理想值为 0 LSB。

Figure 92. 增益误差



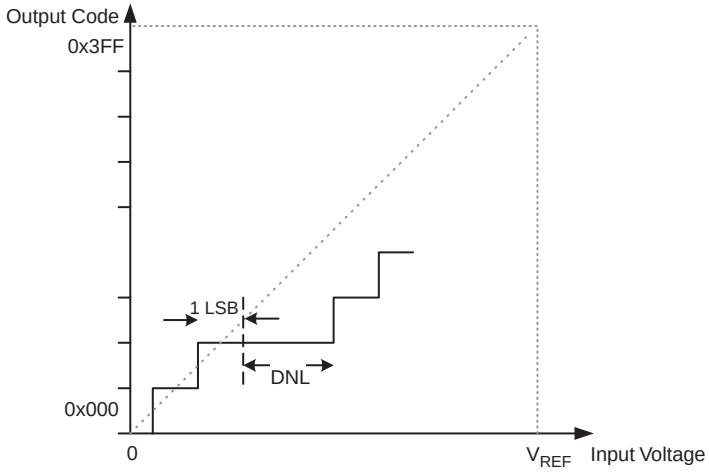
- 整体非线性 (INL)：调整偏移及增益误差之后，所有实际转换与理想转换之间的最大误差即为 INL。理想值：0 LSB。

Figure 93. 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 94. 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 P196 Table 89 及 P197 Table 90)。 0x000 代表模拟地电平， 0x3FF 代表所选参考电压的数值减去 1LSB。

$$ADC = \frac{(V_{POS} - V_{NEG}) \cdot 512}{V_{REF}}$$

Figure 95. 差分测量范围

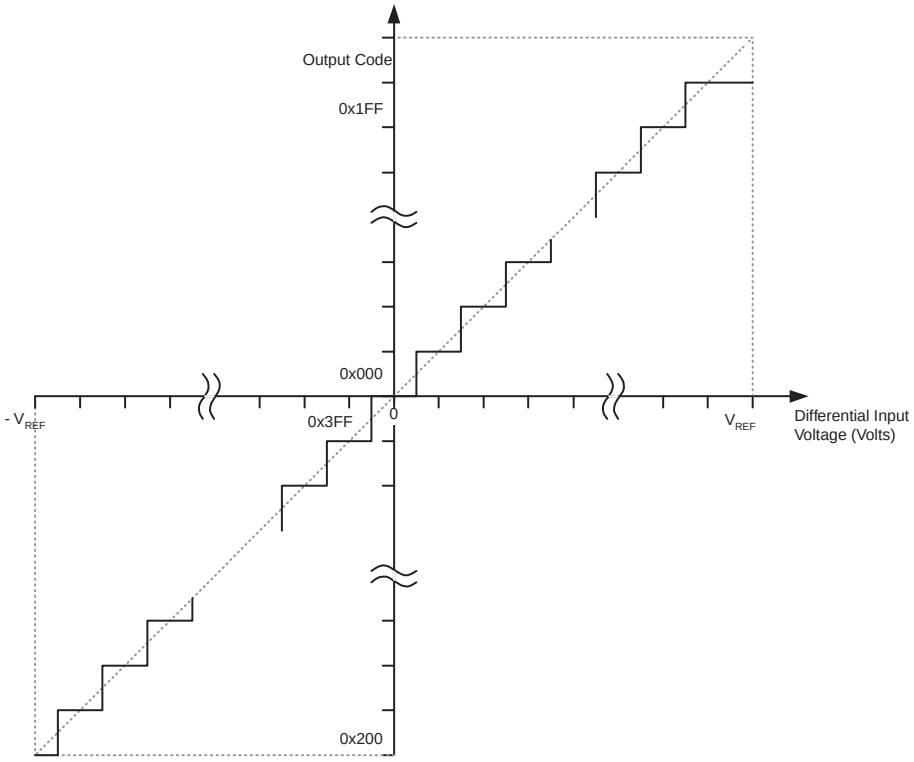


Table 88. 输入电压及输出码之间的关联关系

V_{ADCn}	读出的代码	对应的十进制数值
$V_{ADCm} + V_{REF}$	0x1FF	511
$V_{ADCm} + {}^{511}/_{512}\ V_{REF}$	0x1FF	511
$V_{ADCm} + {}^{510}/_{512}\ V_{REF}$	0x1FE	510
...	...	...
$V_{ADCm} + {}^{1}/_{512}\ V_{REF}$	0x001	1
V_{ADCm}	0x000	0
$V_{ADCm} - {}^{1}/_{512}\ V_{REF}$	0x3FF	-1
...	...	...
$V_{ADCm} - {}^{511}/_{512}\ V_{REF}$	0x201	-511
$V_{ADCm} - V_{REF}$	0x200	-512

ADMUX = 0xFB (ADC3 - ADC2, 1.1V 参考电压，左对齐)

ADC3 上的电压为 300 mV， ADC2 上的电压为 500 mV.

ADCR = 512 * (300 - 500) / 1100 = -93 = 0x3A3.

ADCL 的内容为 0xC0 ,ADCH 的内容为 0xD8。向 ADLAR 写 0 ,对结果进行右对齐之后得到 ADCL = 0xA3， ADCH = 0x03。

ADC 多路复用选择寄存器— ADMUX

位	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	

- **位 7:6 – REFS1:0: 参考电压选择**

如 Table 89 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 89. 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	1.1V 的片内基准电压源， AREF 引脚外加滤波电容

- **位 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，不论是否有转换在进行。关于这一位的完整描述请见 P200 “ADC 数据寄存器 – ADCL 及 ADCH” 。

- **位 4:0 – MUX4:0: 模拟通道选择位**

通过这几位的设置，可以对连接到 ADC 的模拟输入进行选择。细节见 Table 90。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 90. 输入通道选择

MUX4..0	单端输入	差分输入正极	差分输入负极
00000	ADC0	N/A	
00001	ADC1		
00010	ADC2		
00011	ADC3		
00100	ADC4		
00101	ADC5		
00110	ADC6		
00111	ADC7		
01000	N/A		
01001			
01010			
01011			
01100			
01101			
01110			
01111			
10000		ADC0	ADC1
10001		ADC1	ADC1
10010		ADC2	ADC1
10011		ADC3	ADC1
10100		ADC4	ADC1
10101		ADC5	ADC1
10110		ADC6	ADC1
10111		ADC7	ADC1
11000		ADC0	ADC2
11001		ADC1	ADC2
11010		ADC2	ADC2
11011		ADC3	ADC2
11100		ADC4	ADC2
11101		ADC5	ADC2
11110	1.1V (V_{BG})	N/A	
11111	0V (GND)		

ADC 控制及状态寄存器 A—ADCSRA

位	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	

- **位 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。

- **位 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。

- **位 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 ADCSRB 寄存器的 ADC 触发信号源选择位 ADTS 设置。

- **位 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。

- **位 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。

- **位 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 91. 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

位	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

位	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。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 P196 “ADC 转换结果 ” 。

ADC 控制及状态寄存器 B—ADCSRB

位	7	6	5	4	3	2	1	0	
	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB

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

- **位 7 – Res: 保留位**

. 这一位保留。为了与以后的器件相兼容，在写 ADCSRB 时这一位应写 0。

- **位 2:0 – 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 92. 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

数字输入禁止寄存器 0—DIDR0

位	7	6	5	4	3	2	1	0	
	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	DIDR0
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
e 初始值	0	0	0	0	0	0	0	0	

- **位 7..0 – ADC7D..ADC0D: ADC7..0 数字输入禁止**

如果这几位为 1，那么对应 ADC 引脚的数字输入缓冲器被禁止， PIN 寄存器的对应位将为 0。如果 ADC7..0 引脚施加了模拟信号，且当前应用不需要这些管脚提供数字输入缓冲器时，应向这几位写 1 来降低数字输入缓冲器的功耗。

LCD 控制器

LCD 控制器 / 驱动器适用于单色无源液晶显示器 (LCD) , 可以支持四个公共端和 25 个段。

特点

- **25 段和 4 个公共端的显示能力**
- **显示数据可以锁存，从而给寄存器更新充分的自主性**
- **支持静态、 1/2、1/3 和 1/4 的占空比**
- **支持静态、 1/2、1/3 的偏置**
- **低功耗省电模式下仍然可以显示**
- **可选的低功耗波形软件**
- **灵活的帧频选项**
- **可以通过软件选择系统时钟或一个外部异步时钟源作为 LCD 时钟**
- **片内 LCD 电源 , 只需要一个外部电容**
- **在所有 V_{CC} 范围之内， LCD 输出电压 (对比度) 可以通过软件进行选择，范围为 2.6 到 3.35V**
- **相同的电流输出和吸收能力提高了 LCD 的使用寿命**
- **LCD 中断可用于显示数据更新或将芯片从睡眠模式唤醒**
- **LCD 显示不需要的段和公共引脚可以用作普通的 I/O 引脚**

综述

Figure 96 为简化的 LCD 控制器 / 驱动器的方框图。 I/O 引脚的实际排列位置参见 P2 “ATmega169 引脚排列 ” 。

LCD 由段组成 (像素或完整的符号) , 这些段可以是可见的 , 也可以不可见。一个段 有两个电极，两者之间填充有液晶。在液晶上施加超过门限的电压即可以使此段变为可见。

电压必须交替变换以避免液晶出现电泳现象。这种现象会使显示性能下降。因此施加于段上的波形一定不能有直流分量。

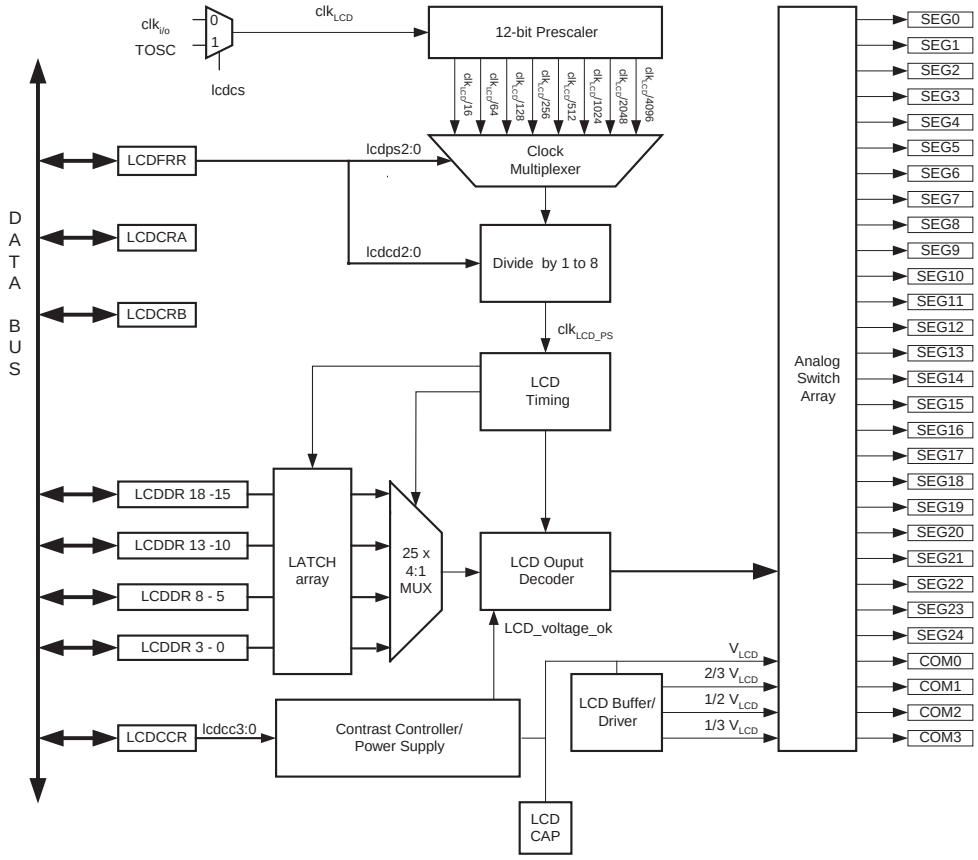
定义

在描述 LCD 时有几个术语。 Table 93 的定义适用于整个文档。

Table 93. 定义

LCD	端点直接与段相连的无源显示板
段	可以打开或关闭的最小元素 (像素)
公用极	表明有多少个段连接到段端点
占空比	1/(实际使用的 LCD 公共端点的数量)
偏置	1/(用于驱动 LCD 的电平的数量 -1)
帧速率	每秒钟激活 LCD 段的次数

Figure 96. LCD 模块块图



LCD 时钟源

LCD 控制器的时钟可以是片内同步时钟或外部异步时钟。时钟源 clk_{LCD} 默认为系统时钟 $clk_{I/O}$。寄存器 LCDCRB 的 LCDCS 位为逻辑 1 时，时钟源将从引脚 TOSC1 获取。

时钟源必须稳定以得到精确的 LCD 时序，并使 LCD 段间直流偏压达到最小。

LCD 预分频器

预分频器由 12 位的波纹计数器和 8 分频器组成。LCDPS2:0 用来选择 clk_{LCD} 是 16、64、128、 256、 512、 1024、 2048 或 4096 分频。

如果需要更高的分辨率，设置 LCDCD2:0 可以进一步将时钟进行 1 到 8 分频。

时钟分频器的输出 clk_{LCD_PS} 即为 LCD 的时钟源，用于确定 LCD 时序。

LCD 存储器

显示存储器以 I/O 寄存器的方式出现，每个公共端都有一组这样的 I/O 寄存器。当显存中的某位为 1 时，相应的段被激活，反之这一段不会激活。

激活一个段必须要提供超过一定门限的绝对电压。通过将相应的 COM 引脚和 SEG 引脚的输出电压设置成相反相位就可以实现该功能。对于有多个公用端的 LCD，必须使用一个 (1/2 偏置) 或两个 (1/3 偏置) 额外的电平。否则，选中 COM0 时没有激活的段会在选中其他公共端时被激活。

对 COM0 的寻址标志着一个帧的开始。寻址的具体动作是驱动 COM0 输出一个大振幅的、相对其他未被寻址的 COM 端相位相反的电压。未被激活的段的电压与 COM0 的电压同相，被激活的段则具有相反相位以及大的电压振幅。波形图请参见 P204 “ 操作模式 ”。来源于 LCDDR3 - LCDDR0 的锁存数据通过多路器送入译码器。该译码器由 LCD 的定时信号控制，并生产控制模拟开关的相关信号以产生输出波形。接下来， COM1 被寻址。来自 LCDDR8 - LCDDR5 的锁存数据输入译码器。寻址操作持续进行，直到所有的公共端都得到寻址。在一个新的帧开始之前要锁存需要显示的数据。

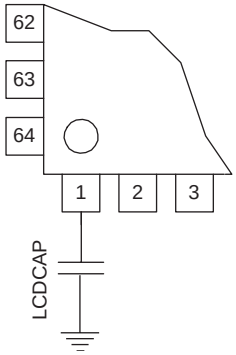
LCD 对比度控制器 / 电源

输出波形的峰值 (V_{LCD}) 决定 LCD 的对比度。V_{LCD} 由软件控制，其值可以在 2.6V 到 3.35V 之间，且与 V_{CC} 无关。在 V_{LCD} 达到目标值之前，一个内部信号会禁止其输出到 LCD。

LCDCAP

如 Figure 97 所示，外部电容 (典型情况下 > 470 nF) 必须连到 LCDCAP 引脚。此电容为 LCD 电压 (V_{LCD}) 的贮能器。大电容值可以减小 V_{LCD} 上的脉动，但是会增加 V_{LCD} 达到其目标值的时间。

Figure 97. LCDCAP 的连接



LCD 缓冲驱动器

中间电平由缓冲器 / 驱动器产生。为了减小功耗，在输出电压值达到 ± 10 mV 范围之前缓冲器一直工作。然后 LCD 输出引脚变成三态，缓冲器关闭。

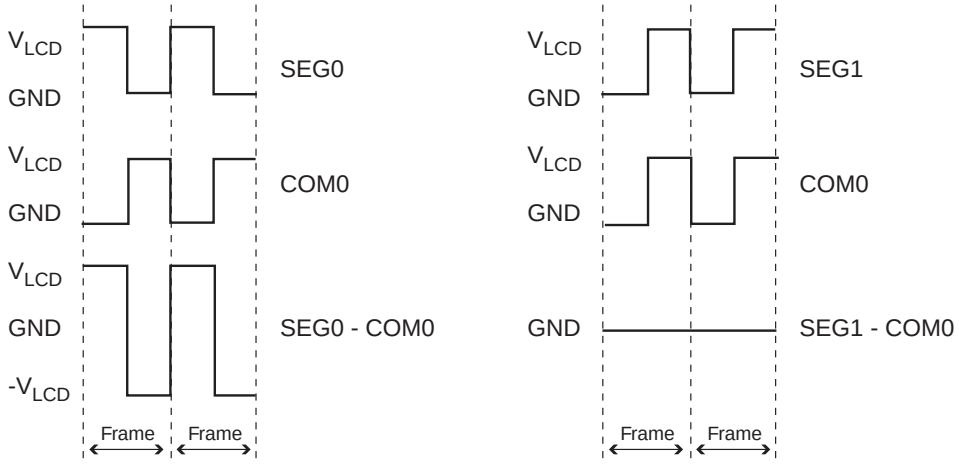
操作模式

静态占空比和偏置

如果 LCD 的所有段只有一个公共电极，则每个段都必须有单独的端口。

这种显示方式的驱动波形如 Figure 98 所示。SEG0 - COM0 是开启段的段间电压，而 SEG1 - COM0 是关闭段的段间电压。

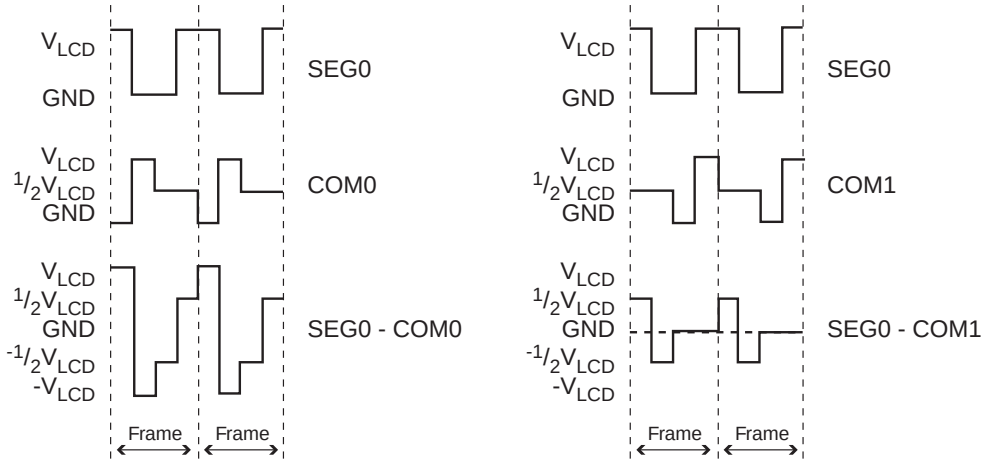
Figure 98. 驱动只有一个公共端的 LCDI



1/2 占空比和 1/2 偏置

对于有两个公共端的 LCD(1/2 占空比) 必须使用更复杂的波形来控制每一个段。尽管可以选择 1/3 偏置，但对这类 LCD 来说，1/2 偏置是最常用的。波形如 Figure 99 所示。SEG0 - COM0 是开启段的段间电压，而 SEG1 - COM0 是关闭段的段间电压。

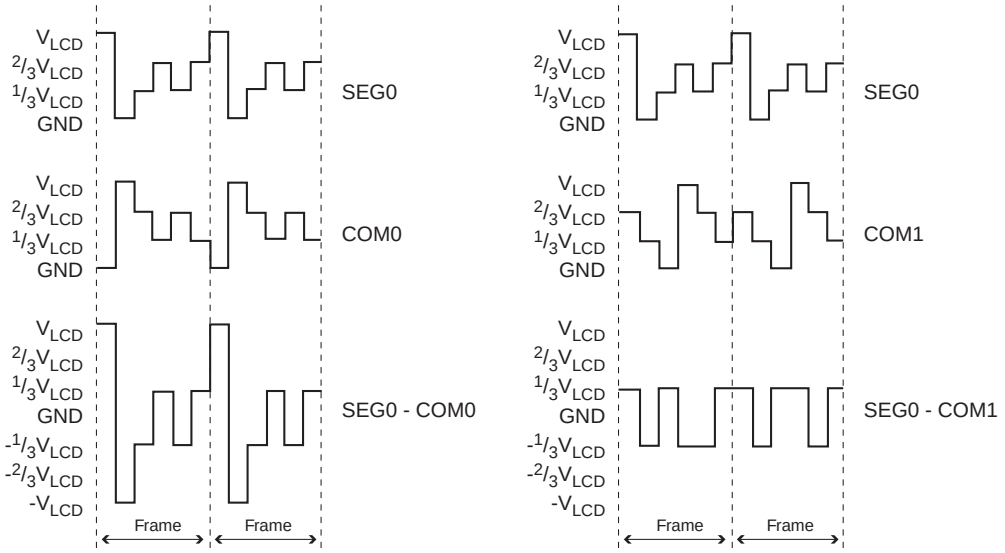
Figure 99. 驱动有两个公共端的 LCD



1/3 占空比和 1/3 偏置

有三个公共端的 LCD(1/3 占空比) 通常建议使用 1/3 偏置。波形见 Figure 100。 SEG0 - COM0 是开启段的段间电压，而 SEG1 - COM0 是关闭段间电压。

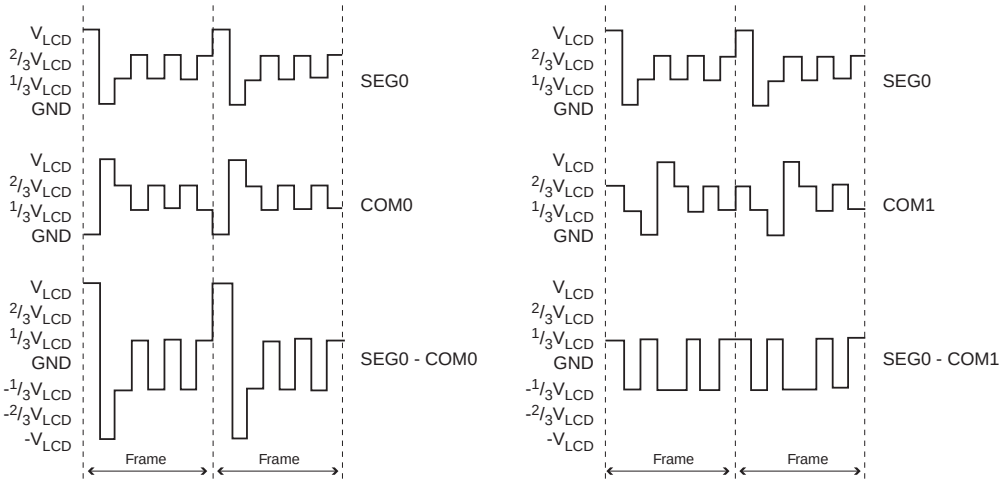
Figure 100. 驱动有 3 个公共端的 LCD



1/4 占空比和 1/3 偏置

1/3 偏置最适合于驱动有四个公共端的 LCD(1/4 占空比)。波形见 Figure 101。SSEG0 - COM0 是开启段的段间电压，而 SEG1 - COM0 是关闭段的段间电压。

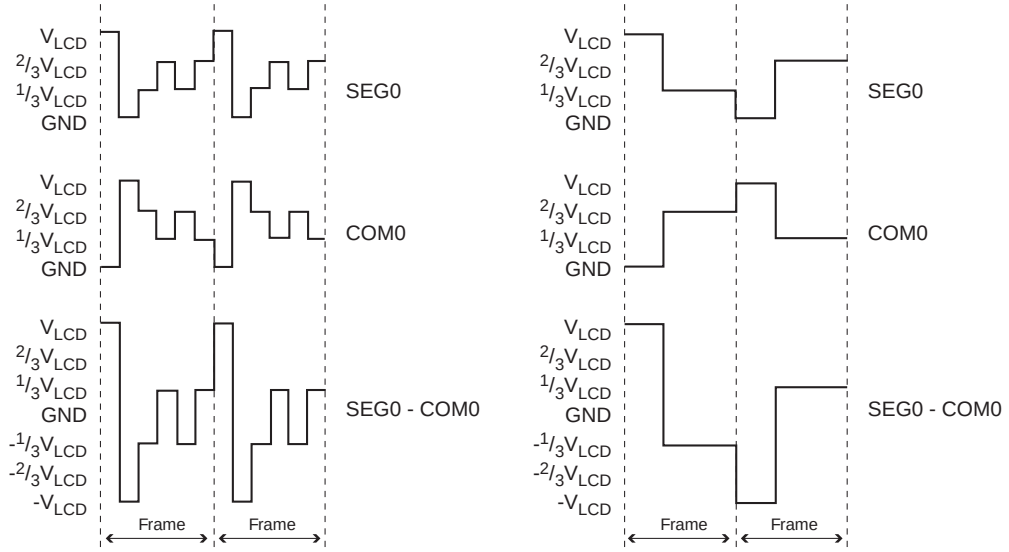
Figure 101. 驱动有 4 个公共端的 LCD



低功耗波形

为了减少电平的变化从而降低功耗，可以通过将 LCDAB 置 1 的方式选用低功耗波形。低功耗波形要求后续两个帧具有相同的显示数据，从而消除直流电压。因此，每隔一帧进行数据锁存和中断标志置位。 1/3 占空比和 1/3 偏置的缺省低功耗波形见 Figure 102。对于其他的占空比和偏置效果是类似的。

Figure 102. 缺省低功耗波形



睡眠模式下的操作

选择了同步 LCD 时钟后 (LCDCS = 0)，不论使用何种时钟源，在空闲模式和省电模式下 LCD 将继续工作。

当选择了校准的内部 RC 振荡器作为系统时钟源时，通过将 LCDCS 位置 1，来自 TOSC1 的异步时钟成为LCD时钟。在空闲模式、ADC噪声抑制模式和省电模式下LCD继续工作。

寄存器 ASSR 的 EXCLK 置位将选择异步时钟。这时，外部时钟输入缓冲器使能，并且外部时钟可以是定时振荡器 1(TOSC1) 引脚的输入，而不是 32kHz 的晶振。详细内容参见 P131 “ 定时器 / 计数器的异步操作 ” 。

在进入掉电模式、选择了同步 LCD 时钟的 Standby 模式或 ADC 噪声抑制模式时，用户必须禁止 LCD。参见 P210 “ 禁止 LCD” 。

显示屏蔽

当 LCDBL 置位时， LCD 在显示完当前帧之后被关闭。所有的段和公共端引脚都接到 GND， LCD 上的电荷被释放。显存的内容则被保留。禁止显示之前首先要关闭显示，以防止有直流电压施加于 LCD，并出现图像逐渐淡化的效果。

端口屏蔽

对于段终端少于 25 的 LCD 来说，可以屏蔽没有使用的引脚并把它们作为普通的端口引脚来使用。详细内容参见 Table 95。未使用的公共端引脚自动配置为普通端口引脚。

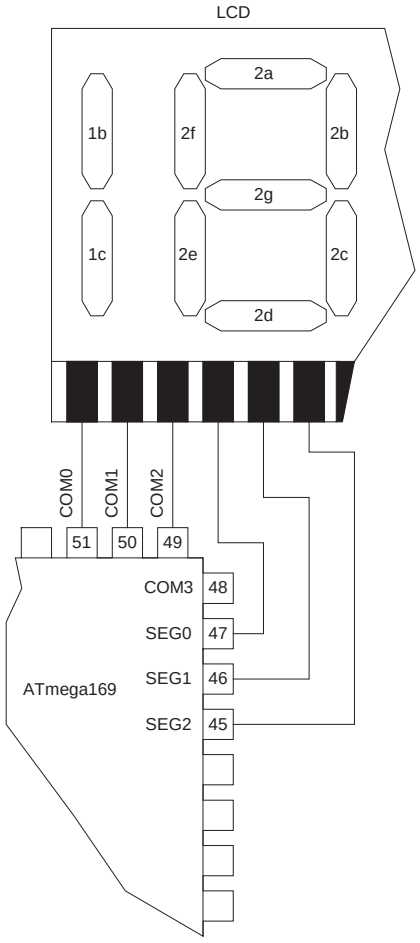
LCD 的用法

LCD 初始化

LCD 使能之前必须进行初始化。初始化过程通常包括帧速率设置、占空比、偏置及端口屏蔽。 LCD 对比度可以在初始阶段进行设置，也可以在运行过程中进行调整。

以下面的 LCD 为例：

Figure 103.



SEG2	2f	2g	..
SEG1	2c	2d	2e
SEG0	1b,1c	2a	2b
	COM0	COM1	COM2

Connection table

显示：	TN 正，反射型
公用端口数：	3
段终端数：	21
偏置系统：	1/3 偏置
驱动系统：	1/3 占空比
工作电压：	3.3 ± 0.9 V

汇编程序代码示例 (1)

```
LCD_Init:

  ; 使用 32 kHz 的晶体振荡器
  ; 1/3 偏置和 1/3 占空比，SEG21:SEG24 用作端口引脚

  ldi   r16, (1<<LCDCS) | (1<<LCDMUX1)| (1<<LCDPM2)
  sts   LCDCRB, r16

  ; 选择 16 为预分频因子，7 为 LCD 时钟分频数
  ; 得到帧速率为 49 Hz

  ldi   r16, (1<<LCDCD2) | (1<<LCDCD1)
  sts   LCDFRR, r16

  ;设置输出电压为 3.3 V

  ldi   r16, (1<<LCDCC3) | (1<<LCDCC2) | (1<<LCDCC1)
  sts   LCDCCR, r16

  ; 使能 LCD，缺省波形并禁止所有中断

  ldi   r16, (1<<LCDEN)
  sts   LCDCRA, r16
  ret
```

C 程序代码示例 (1)

```
Void LCD_Init(void);
{
  /* 使用 32 kHz 的晶体振荡器 */
  /* 1/3 偏置和 1/3 占空比，SEG21:SEG24 用作端口引脚 */

LCDCRB = (1<<LCDCS) | (1<<LCDMUX1)| (1<<LCDPM2);

  /* 选择 16 为预分频因子，7 为 LCD 时钟分频数 */
  /* 得到帧速率为 49 Hz */

  LCDFRR = (1<<LCDCD2) | (1<<LCDCD1);

  /* 设置输出电压为 3.3 V */

  LCDCCR = (1<<LCDCC3) | (1<<LCDCC2) | (1<<LCDCC1);

  /* 使能 LCD，缺省波形并禁止所有中断 */

  LCDCRA = (1<<LCDEN);
}
```

Note: 1. 该示例代码假定包括了合适的头文件。

在重新初始化之前要禁止 LCD 控制器 / 驱动器。

更新 LCD

显示每一帧之前首先要锁存显示存储器 (LCDDR0、 LCDDR1...)、 LCD 屏蔽 (LCDBL)、低功耗波形 (LCDAB) 及对比度控制 (LCDCCR)。对这些 LCD 寄存器的寻址并没有什么限制，但如果在数据更新过程中有新的数据被锁存， LCD 数据的更新将被分割到两个帧。为了避免这个问题，可以利用中断来更新显示存储器、 LCD 屏蔽、低功耗波形及对比度控制。

在下面这个例子中，我们假设在帧之间只有 COM1 的 SEG10 段及 COM0 的 SEG4 段发生变化。数据则存储于 r20 和 r21。

汇编程序代码示例 (1)

```
LCD_update:

  ; 不改变 LCD 屏蔽及低功耗波形
  ;更新显示存储器

  sts   LCDDR0, r20
  sts   LCDDR6, r21
  ret
```

C 程序代码示例 (1)

```
Void LCD_update(unsigned char data1, data2);
{
  /* 不改变 LCD 屏蔽及低功耗波形 */
  /* 更新显示存储器 */

  LCDDR0 = data1;
  LCDDR6 = data2;
}
```

Note: 1. 该示例代码假设包括了合适的头文件。

禁止 LCD

有些应用需要禁止 LCD。例如 MCU 进入掉电模式，同时时钟源也被撤消。

禁止 LCD 之前必须进行彻底地放电，使得每一个段都没有残留的不直流电压。达到这种效果的最佳方式就是使用 LCD 屏蔽特性，使所有的段引脚和公共端引脚接地。

当 LCD 被禁止时，引脚的端口功能将被再次激活。因此，用户必须确保连接到 LCD 引脚的端口为高阻态，或者输出低电平。

汇编程序代码示例 (1)

```
LCD_disable:

  ; 在新的一帧开始之前一直等待

Wait_1:
  lds   r16, LCDCRA
  sbrs  r16, LCDIF
  rjmp  Wait_1

  ; 通过标志位写 1 的方式设置 LCD 屏蔽并清除中断标志位
  ;

  ldi   r16, (1<<LCDEN)|(1<<LCDIF)|(1<<LCDBL)
  sts   LCDCRA, r16

  ; 在 LCD 屏蔽生效前一直等待
Wait_2:
  lds   r16, LCDCRA
  sbrs  r16, LCDIF
  rjmp  Wait_2

  ; 禁止 LCD
  ldi   r16, (0<<LCDEN)
  sts   LCDCRA, r16
  ret
```

C 程序代码示例 (1)

```
Void LCD_disable(void);
{
  /* 在新的一帧开始之前一直等待 */

while ( !(LCDCRA & (1<<LCDIF)) )
  ;
  /* 通过标志位写 1 的方式设置 LCD 屏蔽并清除中断标志位 */
  /* */

  LCDCRA = (1<<LCDEN)|(1<<LCDIF)|(1<<LCDBL);

  /* 在 LCD 屏蔽生效前一直等待 */

while ( !(LCDCRA & (1<<LCDIF)) )
  ;
  /* 禁止 LCD */

  LCDCRA = (0<<LCDEN);
}
```

Note: 1. 该示例代码假设包括了合适的头文件。

LCD 控制及状态寄存器 A—LCDCRA

位	7	6	5	4	3	2	1	0	
	LCDEN	LCDAB	–	LCDIF	LCDIE	–	–	LCDBL	LCDCRA
读 / 写	R/W	R/W	R	R/W	R/W	R	R	R/W	
初始化值	0	0	0	0	0	0	0	0	

- **位 7 – LCDEN: LCD 使能位**

LCDEN置位可以使能LCD控制器/驱动器。此位清零将立即关闭LCD。在显示过程时关闭 LCD 控制器 / 驱动器将使能普通的端口功能，而且如果端口配置为输出，则直流电压将施加到 LCD。 LCD 控制器 / 驱动器被禁止时，建议使这些驱动 LCD 的引脚输出低电平。

- **位 6– LCDAB: LCD 低功耗波形位**

LCDAB 为 0 时，默认的波形从 LCD 引脚输出。LCDAB 置位时，LCD 的引脚输出低功耗波形。如果在显示过程中这一位被修改，则在新的一帧开始时输出发生变化。

- **位 5– Res: 保留位**

在 ATmega169 中这一位是保留位，读返回值为 0。

- **位 4 – LCDIF: LCD 中断标志位**

LCDIF 在每一帧开始的时候被硬件置位，与此同时，要显示的数据得到更新。如果 LCDIE 和 SREG 的 I 位也置位，那么 LCD 的 SOF(帧的起始) 中断将得到执行。执行中断处理程序时 LCDIF 被硬件清除。另外，对 LCDIF 写 1 也可以清零 LCDIF。要注意的是，如果对 LCDCRA 进行读 - 修改 - 写操作，正在等待处理的中断将被禁止。如果选选择了低功耗波形，那么中断标志位每两帧设置一次。

- **位 3– LCDIE: LCD 中断使能位**

如果 LCDIE 及 SREG 的 I 置位，那么 LCD 的 SOF 中断即被使能。

- **位 2:1 – Res: 保留位**

在 ATmega169 中这些位是保留位，读返回值为 0。

- **位 0 – LCDBL: LCD 屏蔽位**

LCDBL 置位时，在当前帧结束后显示将被屏蔽。所有的段和公共端引脚输出低电平。

LCD 控制及状态寄存器 B—LCDCRB

位	7	6	5	4	3	2	1	0	
	LCDCS	LCD2B	LCDMUX1	LCDMUX0	–	LCDPM2	LCDPM1	LCDPM0	LCDCRB
读 / 写	R/W	R/W	R/W	R/W	R	R/W	R/W	R/W	
I 初始化值	0	0	0	0	0	0	0	0	

- **位 7 – LCDCS: LCD 时钟选择位**

LCDCS为0时LCD使用系统时钟。否则LCD使用外部异步时钟源作为自己的时钟源。取决于 ASSR 寄存器 EXCLK 的设置，这个异步时钟源可以是定时器 / 计数器振荡器，也可以是外部时钟。详细内容请参见 P131 “ 定时器 / 计数器的异步操作 ” 。

- **位 6 – LCD2B: LCD 1/2 偏置选择位**

LCD2B为0时LCD将使用1/3偏置。否则LCD使用1/2偏置。请参考 LCD产品推荐的偏置数据。

- **位 5:4 – LCDMUX1:0: LCD Mux 选择位**

LCDMUX1:0 决定了LCD的占空比。没有用于LCD显示的公共端引脚为普通的端口引脚。不同的占空比选择请参见 Table 94。

Table 94. LCD 占空比选择

LCDMUX1	LCDMUX0	占空比	偏置	COM 引脚	I/O 端口引脚
0	0	静态	静态	COM0	COM1:3
0	1	1/2	1/2 or 1/3(1)	COM0:1	COM2:3
1	0	1/3	1/2 or 1/3(1)	COM0:2	COM3
1	1	1/4	1/2 or 1/3(1)	COM0:3	无

Note: 1. LCD2B 为 1 时偏置选 1/2，否则选 1/3。

- **位 3 – Res: 保留位**

在 ATmega169 中这一位被保留，读返回值为零。

- **位 2:0 – LCDPM2:0: LCD 端口屏蔽位**

LCDPM2:0决定了用作段驱动的端口引脚数。Table 95中列出了不同的选择方案。未使用的引脚可以用作普通的端口引脚。

Table 95. LCD 端口屏蔽

LCDPM2	LCDPM1	LCDPM0	用于段驱动的 I/O 端口	最大的段数
0	0	0	SEG0:12	13
0	0	1	SEG0:14	15
0	1	0	SEG0:16	17
0	1	1	SEG0:18	19
1	0	0	SEG0:20	21
1	0	1	SEG0:22	23
1	1	0	SEG0:23	24
1	1	1	SEG0:24	25

LCD 帧频寄存器—LCDFRR

位 t	7	6	5	4	3	2	1	0	
	–	LCDPS2	LCDPS1	LCDPS0	–	LCDCD2	LCDCD1	LCDCD0	LCDFRR
读 / 写	R	R/W	R/W	R/W	R	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

- **位 7 – Res: 保留位**

在 ATmega169 中这一位是保留位，读返回值为零。

- **位 6:4 – LCDPS2:0: LCD 预分频器选择位**

LCDPS2:0 确定从预分频器的哪一点引出信号。预分频器的输出可以进一步通过设置时钟驱动位(LCDCD2:0)进行分频。Table 96中列出了各种不同的选择。它们共同决定了预分频的 LCD 时钟 (clk_{LCD_PS})，为 LCD 模块产生时钟信号。

Table 96. LCD 预分频器选择

LCDPS2	LCDPS1	LCDPS0	预分频器输出 clk_{LCD}/N	LCDCD2:0 = 0, 占空比 = 1/4, 以及帧频 = 64 Hz 时预分频的 LCD 时钟频率
0	0	0	$clk_{LCD}/16$	8.1 kHz
0	0	1	$clk_{LCD}/64$	33 kHz
0	1	0	$clk_{LCD}/128$	66 kHz
0	1	1	$clk_{LCD}/256$	130 kHz
1	0	0	$clk_{LCD}/512$	260 kHz
1	0	1	$clk_{LCD}/1024$	520 kHz
1	1	0	$clk_{LCD}/2048$	1 MHz
1	1	1	$clk_{LCD}/4096$	2 MHz

- **位 3 – Res: 保留位**

在 ATmega169 中这一位是保留位，读返回值为零。

- **位 2:0 – LCDCD2:0: LCD 时钟分频**

LCDCD2:0决定了时钟分频器的分频比率。各种参数选择参见Table 97。时钟分频器为帧频的选择提供了更大的灵活性。

Table 97. LCD 时钟分频

LCDCD2	LCDCD1	LCDCD0	预分频器输出除以：	ck_{LCD} = 32.768 kHz, N = 16, 占空比 = 1/4, 得到如下帧频：
0	0	0	1	256 Hz
0	0	1	2	128 Hz
0	1	0	3	85.3 Hz
0	1	1	4	64 Hz
1	0	0	5	51.2 Hz
1	0	1	6	42.7 Hz
1	1	0	7	36.6 Hz
1	1	1	8	32 Hz

帧频可以通过以下方程计算

$$f_{frame} = \frac{f_{clk_{LCD}}}{(K \cdot N \cdot (1 + LCDCD))}$$

其中：

N = 预分频器分频数 (16， 64， 128， 256， 512， 1024， 2048 或 4096)

K = 8，适用于占空比为 1/4， 1/2 和静态的情况

K = 6，适用于占空比为 1/3 的情况

这种方案很灵活，用户可以根据上面的公式得到自己的数据表。当帧频寄存器保持不变时，使用 1/3 的占空比可使帧速率增加 33%。帧速率的计算实例参见 Table 98。

Table 98. 帧频计算实例

ckl_{LCD}	占空比	K	LCDPS2:0	N	LCDCS2:0	帧频
4 MHz	1/4	8	6	2048	3	4000000/(8*2048*(1+3)) = 61 Hz
4 MHz	1/3	6	6	2048	3	4000000/(6*2048*(1+3)) = 81 Hz
32.768 kHz	Static	8	0	16	0	32768/(8*16*(1+0)) = 256 Hz
32.768 kHz	1/2	8	0	16	4	32768/(8*16*(1+4)) = 51 Hz

LCD 对比度控制寄存器— LCDCCR

位	7	6	5	4	3	2	1	0	
	–	–	–	–	LCDCC3	LCDCC2	LCDCC1	LCDCC0	LCDCCR
读 / 写	R	R	R	R	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

- **位 7:4 – Res: 保留位**

在 ATmega169 中这些位是保留位，读返回值为零。

- **位 3:0 – LCDCC3:0: LCD 对比度控制位**

LCDCC3:0决定了施加于段和公共端引脚的最大电压 VLCD。Table 99中列出了不同的参数选择。新的数值在帧开始的时候生效。

Table 99. LCD 对比度控制

LCDCC3	LCDCC2	LCDCC1	LCDCC0	最大电压值 V_{LDC}
0	0	0	0	2.60
0	0	0	1	2.65
0	0	1	0	2.70
0	0	1	1	2.75
0	1	0	0	2.80
0	1	0	1	2.85
0	1	1	0	2.90
0	1	1	1	2.95
1	0	0	0	3.00
1	0	0	1	3.05
1	0	1	0	3.10
1	0	1	1	3.15
1	1	0	0	3.20
1	1	0	1	3.25
1	1	1	0	3.30
1	1	1	1	3.35

LCD 存储器映像

LCD 存储器的某一位置 1 将使则相应的段被激活 (使之可见)。在实际应用中没有用到的 LCD 存储器可以用于其他存储用途。

位	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	–	–	LCDDR19
COM3	–	–	–	–	–	–	–	SEG324	LCDDR18
COM3	SEG323	SEG322	SEG321	SEG320	SEG319	SEG318	SEG317	SEG316	LCDDR17
COM3	SEG315	SEG314	SEG313	SEG312	SEG311	SEG310	SEG309	SEG308	LCDDR16
COM3	SEG307	SEG306	SEG305	SEG304	SEG303	SEG302	SEG301	SEG300	LCDDR15
	–	–	–	–	–	–	–	–	LCDDR14
COM2	–	–	–	–	–	–	–	SEG224	LCDDR13
COM2	SEG223	SEG222	SEG221	SEG220	SEG219	SEG218	SEG217	SEG216	LCDDR12
COM2	SEG215	SEG214	SEG213	SEG212	SEG211	SEG210	SEG209	SEG208	LCDDR11
COM2	SEG207	SEG206	SEG205	SEG204	SEG203	SEG202	SEG201	SEG200	LCDDR10
	–	–	–	–	–	–	–	–	LCDDR9
COM1	–	–	–	–	–	–	–	SEG124	LCDDR8
COM1	SEG123	SEG122	SEG121	SEG120	SEG119	SEG118	SEG117	SEG116	LCDDR7
COM1	SEG115	SEG114	SEG113	SEG112	SEG111	SEG110	SEG109	SEG108	LCDDR6
COM1	SEG107	SEG106	SEG105	SEG104	SEG103	SEG102	SEG101	SEG100	LCDDR5
	–	–	–	–	–	–	–	–	LCDDR4
COM0	–	–	–	–	–	–	–	SEG024	LCDDR3
COM0	SEG023	SEG022	SEG021	SEG020	SEG019	SEG018	SEG017	SEG016	LCDDR2
COM0	SEG015	SEG014	SEG013	SEG012	SEG011	SEG010	SEG009	SEG008	LCDDR1
COM0	SEG007	SEG006	SEG005	SEG004	SEG003	SEG002	SEG001	SEG000	LCDDR0
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	

JTAG 接口和片上调试系统

特点

- **JTAG (IEEE std. 1149.1 标准兼容) 接口**
- **遵循 IEEE std. 1149.1 (JTAG) 标准的边界扫描功能**
- **调试器可以访问：**
  - **所有的外设**
  - **片内及片外的 RAM**
  - **内部寄存器文件**
  - **程序计数器**
  - **EEPROM 及 Flash 存储器**
- **扩展的片上调试功能，支持多种断点条件：**
  - **AVR 断点指令**
  - **程序流程变化断点**
  - **单步断点**
  - **单个地址或地址范围程序断点**
  - **单个地址或地址范围数据断点**
- **通过 JTAG 接口实现 Flash、 EEPROM、熔丝位和锁定位的编程**
- **AVR Studio® 支持片上调试功能**

概述

AVR 器件中与 IEEE std. 1149.1 兼容的 JTAG 接口可以用于：

- 通过 JTAG 边界扫描特性测试 PCB
- 对非易失性存储器、熔丝位和锁定位进行编程
- 片上调试

以下部分给出了简单的描述。通过 JTAG 接口进行编程和边界扫描链的详细描述可分别在 P274 “通过JTAG接口进行编程” 和P224 “IEEE 1149.1 (JTAG)边界扫描” 中查到。片上调试支持不属于 JTAG 公共指令，仅对 ATMEL 内部和经过挑选的第三方开放。

Figure 104 为JTAG接口和片上调试系统的方框图。TAP控制器是由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之一作为 TDI (输入) 和 TDO(输出) 之间的扫描链 (移位寄存器)。指令寄存器保存的是控制数据寄存器操作的 JTAG 指令。

ID寄存器 , 旁路寄存器和边界扫描链是用于板级测试的数据寄存器。JTAG编程接口 (由几个物理数据寄存器和虚拟数据寄存器组成) 通过 JTAG 接口进行串行编程。内部的扫描链和断点扫描链只用于片上调试。

测试访问端口—TAP

JTAG接口由AVR的四个引脚组成。以JTAG专有名词来说 , 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。此引脚通过 TAP 控制器状态机执行导航功能。
- TCK: 测试时钟。 JTAG 操作和 TCK 是同步的。
- TDI: 测试数据输入。需要移至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。从指令寄存器或数据寄存器输出的串行数据。

1149.1 规范还规定了一个可选的 TAP 信号 : TRST – 测试复位。AVR 没有给出这个信号。

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) 处于悬空挂态，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 104. 方框图

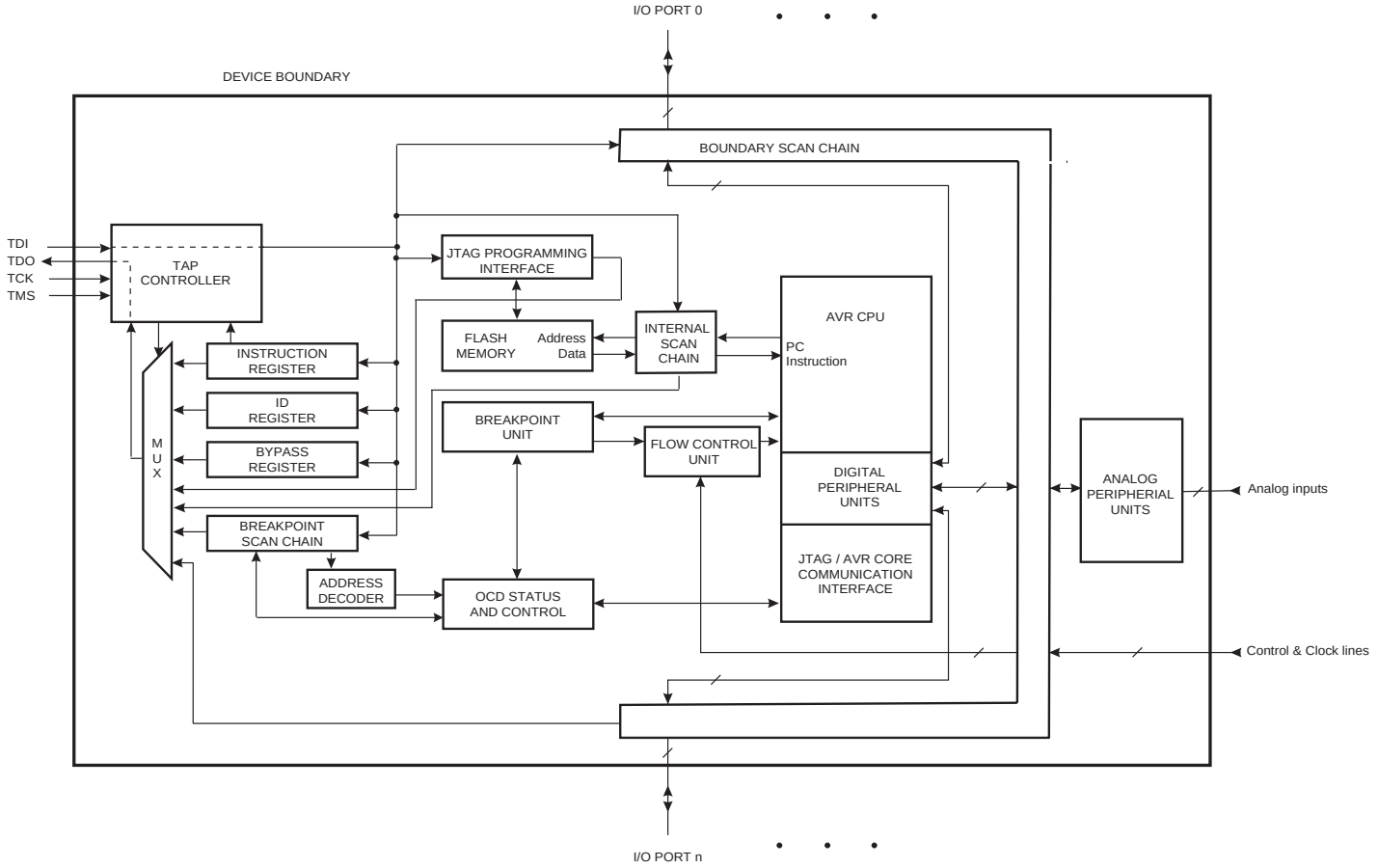
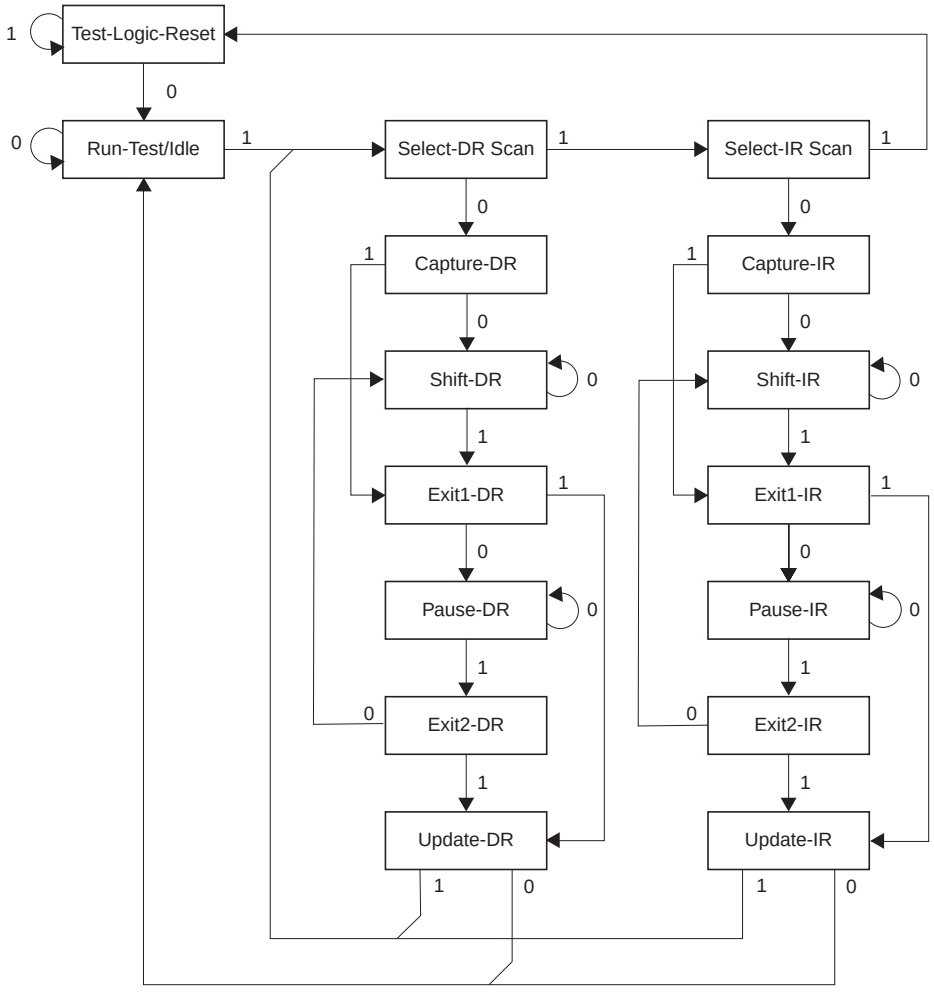


Figure 105. TAP 控制器状态图



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 105 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 P223 “ 参考文献 ” 。

使用边界扫描链

边界扫描特性的完整描述在 P224 “IEEE 1149.1 (JTAG) 边界扫描 ” 中给出。

使用片上调试系统

如 Figure 104 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 P222 “ 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 LB1 或 LB2 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。

AVR Studio 使用户能完全控制那些具有片上调试能力的 AVR 器件、AVR 仿真器以及 AVR 指令集仿真器。 AVR Studio® 支持由 Atme 公司的 AVR 汇编编译器编译的汇编程序以及经过第三方编译器编译的 C 程序源代码级的调试。

AVR Studio 可在 Microsoft® Windows® 95/98/2000, Windows NT® 和 Windows XP® 系统下运行。

AVR Studio 的全面描述请参见 AVR Studio 用户指南。本文只介绍一些要点。

在 AVR Studio 中，无论源码级还是反汇编级，所有的指令都可以执行。用户可以执行程序、通过跟踪 (Trace) 或跳过 (Step Over) 功能来单步运行程序、跳出函数、执行程序直到光标所在语句、结束执行程序以及复位程序。此外，用户可以设置无限多个代码断点 (使用 BREAK 指令) 和至多两个数据存储断点，或者组合的可屏蔽 (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; 0x8 访问片上调试系统的非规范 JTAG 指令。

PRIVATE1; 0x9 访问片上调试系统的非规范 JTAG 指令。

PRIVATE2; 0xA 访问片上调试系统的非规范 JTAG 指令。

PRIVATE3; 0xB 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

位 t	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	
初始化值 I	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 P274 “ 通过 JTAG 接口进行编程 ” 部分给出。

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

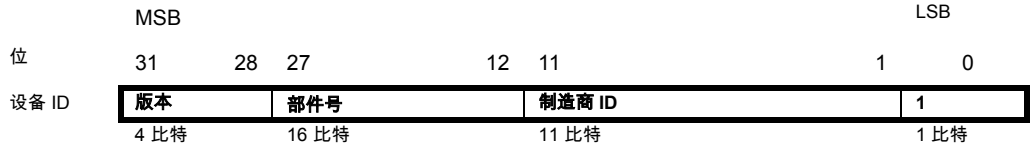
旁路寄存器

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 106 给出了器件识别寄存器的结构。

Figure 106. 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 100。

Table 100. JTAG 版本号

版本	JTAG 版本号 (Hex)
ATmega169 修订版 A	0x0
ATmega169 修订版 B	0x1
ATmega169 修订版 C	0x2
ATmega169 修订版 D	0x3

芯片型号

芯片型号是用来识别器件的16位代码。Table 101列出了ATmega169的JTAG器件型号。

Table 101. AVR JTAG 器件型号

芯片型号	JTAG 芯片型号 (Hex)
ATmega169	0x9405

制造商 ID

制造商 ID 是用来区别厂商的 11 位代码。Table 102 中列出了 ATMEL 的 JTAG 制造商 ID。

Table 102. 制造商 ID

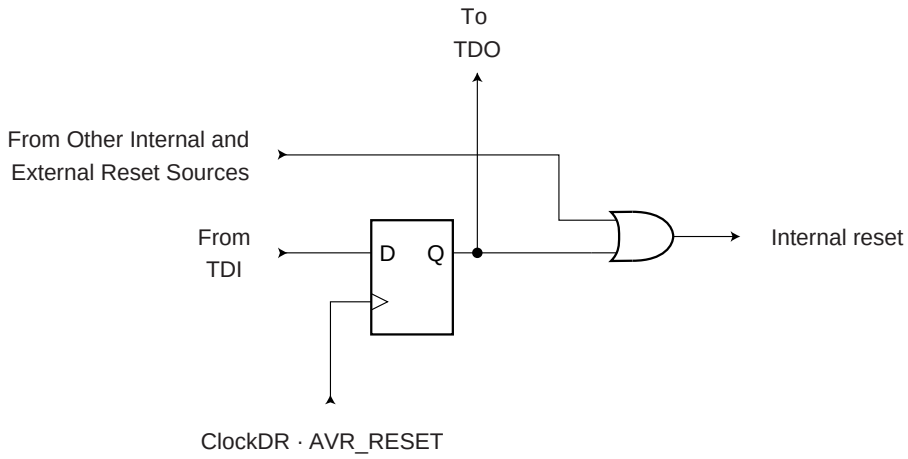
制造商	JTAG 制造商 ID (Hex)
ATMEL	0x01F

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 P24 “ 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 107 所示。

Figure 107. 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 P228 “ 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; 0x0

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; 0x1

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; 0x2

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; 0xC

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; 0xF

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 控制寄存器—MCUCR

MCU 控制寄存器包含控制通用 MCU 功能的控制位。 .

位	7	6	5	4	3	2	1	0	
	JTD	–	–	PUD	–	–	IVSEL	IVCE	MCUCR
读 / 写	R/W	R	R	R/W	R	R	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

- **位 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。使用片上调试系统时一定不要改变这一位。

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。

MCU 状态寄存器—MCUSR

MCU 状态寄存器可以提供哪个复位源致使 MCU 复位的信息。

位	7	6	5	4	3	2	1	0	
	–	–	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUSR
读 / 写	R	R	R	R/W	R/W	R/W	R/W	R/W	
初始化值 I	0	0	0	参见位描述					

- **位 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 108 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 109 给出了一个如 P50 “I/O 端口 ” 所述的简单数字端口引脚。 Figure 109 的虚线框包含了 Figure 108 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 109 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 108. 具有上拉功能双向端口引脚的边界扫描 .

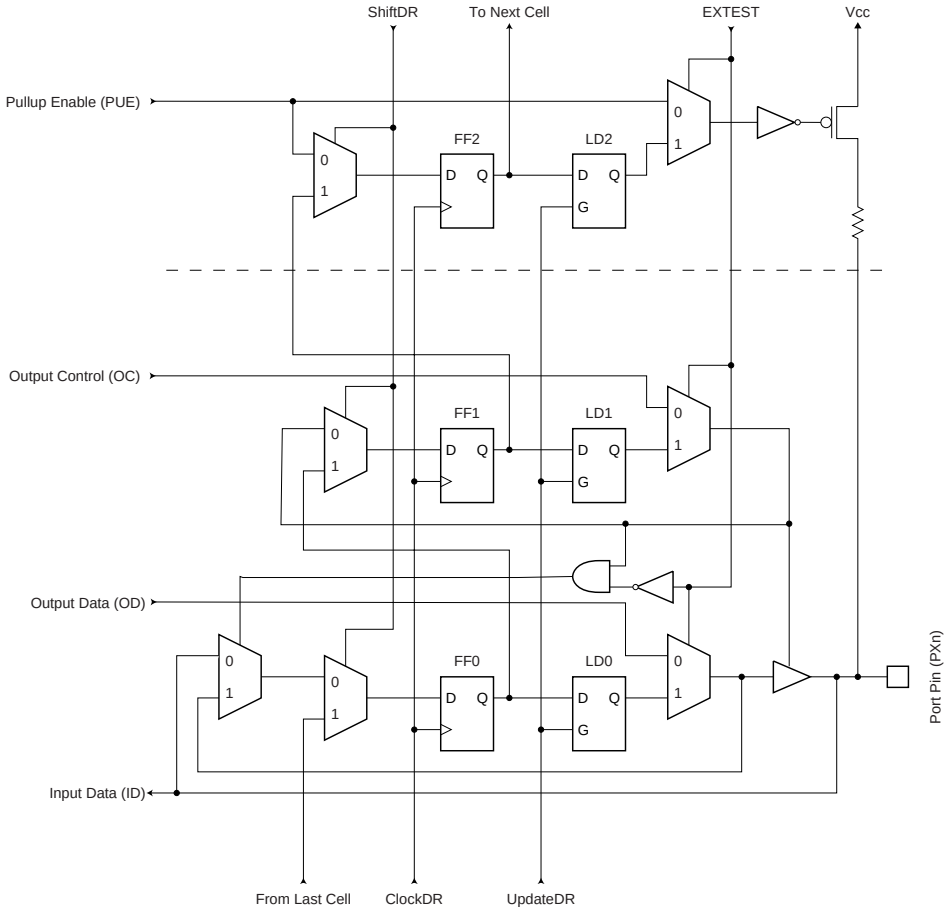
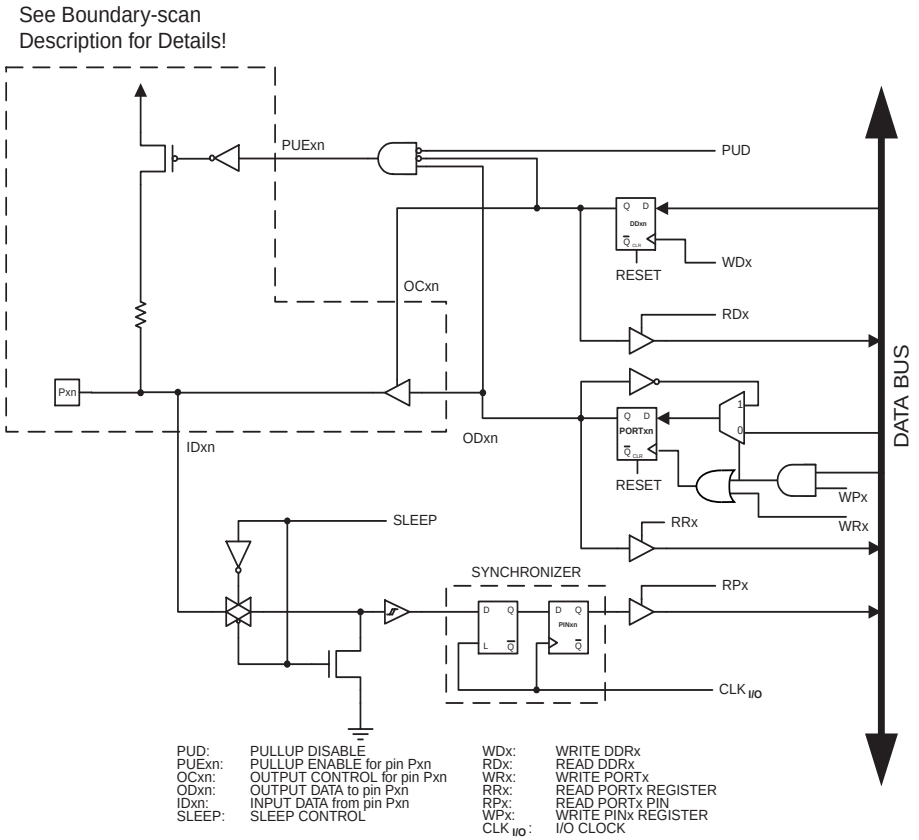


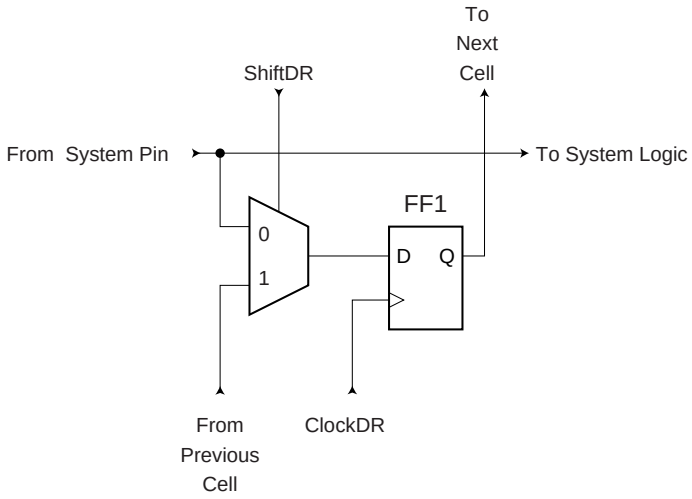
Figure 109. 通用端口引脚原理图



扫描复位引脚

对于标准复位操作， RESET 引脚接受 5V 系统的有效低电平，对于高电压并行编程则接受 12V 的高电平。复位引脚嵌入了如 Figure 110 所示的观测单元，它同时适用于 5V 复位信号 RSTT 和 12V 复位信号 RSTHV。

Figure 110. 观测单元 I



扫描时钟引脚

AVR 有很多通过熔丝位设置的时钟选项。具体有内部 RC 振荡器、外部时钟、(高频) 晶体振荡器、低频晶体振荡器和陶瓷谐振器。

Figure 111 示出了在扫描链中每个具有外部连接的振荡器是如何得到支持的。使能信号由通用扫描单元支持，而振荡器 / 时钟输出则附属于一个观测单元。和主时钟一样，定时器振荡器的扫描也是如此。由于片内 RC 振荡器没有外部连接，因此不会被扫描。

Figure 111. 振荡器和时钟选项的边界扫描单元

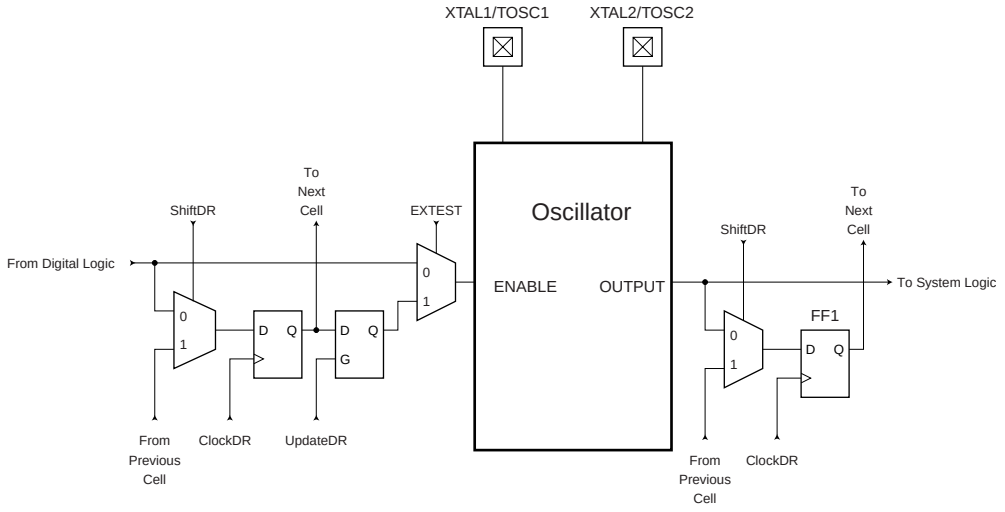


Table 103 简单概括了用于外部时钟引脚 XTAL1、具有 XTAL1/XTAL2 连接的振荡器及定时器振荡器的扫描寄存器。

Table 103. 振荡器 (1)(2)(3) 的扫描信号

使能信号	扫描的时钟线	时钟选择	不使用时扫描的时钟线
EXTCLKEN	EXTCLK (XTAL1)	外部时钟	0
OSCON	OSCCK	外部晶体 外部陶瓷共振器	1
OSC32EN	OSC32CK	低频外部晶体	1

Notes:
1. 一次最多只能使能一个时钟源作为主时钟。
2. 由于内部振荡器和 JTAG TCK 时钟之间有频率漂移，所以扫描振荡器输出会出现不确定的结果。可能的话扫描外部时钟更好一些。
3. 时钟配置通过熔丝位确定。由于熔丝位在运行时间不会改变，所以对于给定的应用可以认为时钟配置是固定的。建议用户扫描与最终系统时钟相同的时钟。由于在休眠模式下系统逻辑可以禁止时钟，所以扫描链支持使能信号。如果没有提供这样的使能信号，则需要在扫描路径中将其断开。

扫描模拟比较器

Figure 112 给出了与边界扫描相关的比较器信号。Figure 113 的边界扫描单元与所有这些信号相连接。 Table 104 描述了这些信号。

由于所有的模拟输入都与相应的数字端口引脚共享，纯连接测试不必使用比较器。

Figure 112. 模拟比较器

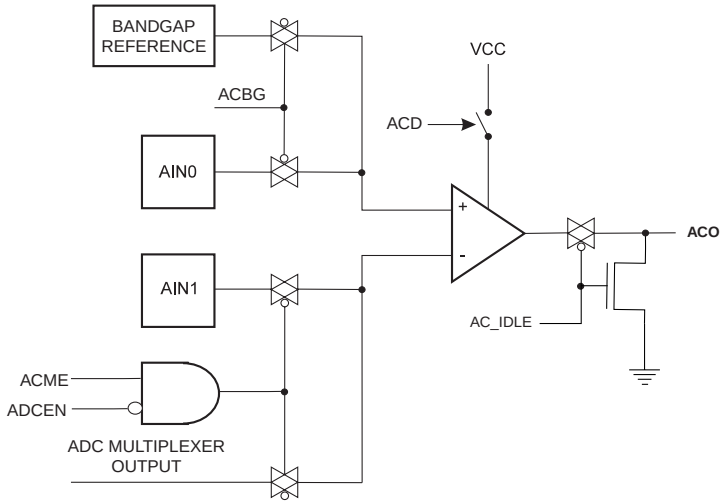


Figure 113. 用于比较器和 ADC 信号的一般边界扫描单元

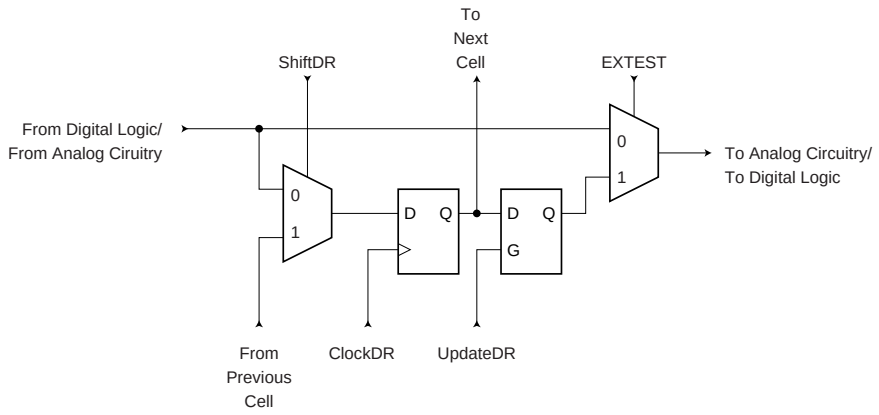


Table 104. 模拟比较器边界扫描信号

信号名称	相对于比较器的方向	描述	不使用时的推荐输入	使用推荐输入时的输出值
AC_IDLE	输入	为真时关闭模拟比较器	1	取决于正在执行的 µC 代码
ACO	输出	模拟比较器输出	将变成正在执行的 µC 代码的输入	0
ACME	输入	为真实使用 ADC mux 的输出信号	0	取决于正在执行的 µC 代码
ACBG	输入	能带隙参考使能	0	取决于正在执行的 µC 代码

扫描 ADC

Figure 114为具有相关控制和观测信号的ADC块图。Figure 110的边界扫描单元与每一个信号相连。由于所有的模拟输入都与相应的数字端口引脚共享，纯连接测试不必使用ADC。

Figure 114. AD 转换器

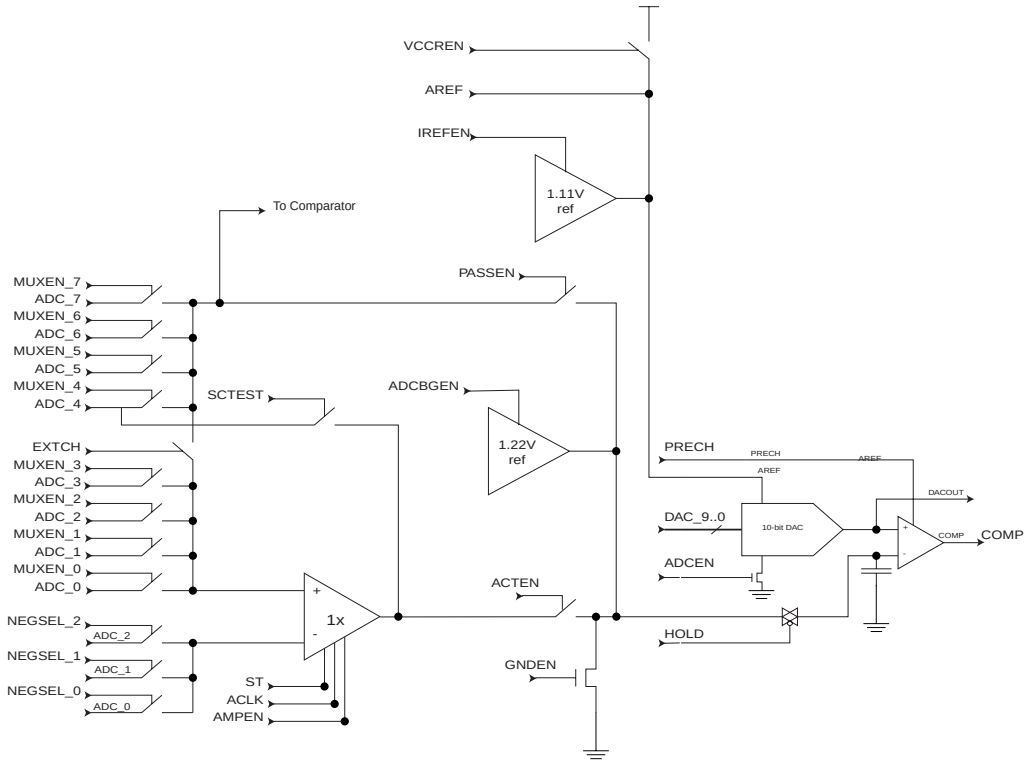


Table 105 对信号进行了简单的描述。

Table 105. ADC[1] 边界扫描信号

信号名称	相对 ADC 的方向	描述	不使用时的推荐输入	使用了推荐输入并且 CPU 不使用 ADC 时的输出值
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级的时钟信号	0	0
ACTEN	输入	使能增益级与比较器间的通路	0	0
ADCBGEN	输入	使能能隙基准作为比较器的负极输入	0	0
ADCEN	输入	ADC 的上电信号	0	0
AMPEN	输入	增益级的上电信号	0	0
DAC_9	输入	DAC 的第 9 位数据	1	1
DAC_8	输入	DAC 的第 8 位数据	0	0
DAC_7	输入	DAC 的第 7 位数据	0	0
DAC_6	输入	DAC 的第 6 位数据	0	0
DAC_5	输入	DAC 的第 5 位数据	0	0
DAC_4	输入	DAC 的第 4 位数据	0	0
DAC_3	输入	DAC 的第 3 位数据	0	0
DAC_2	输入	DAC 的第 2 位数据	0	0
DAC_1	输入	DAC 的第 1 位数据	0	0
DAC_0	输入	DAC 的第 0 位数据	0	0
EXTCH	输入	连接 ADC 通道 0 - 3 到增益级周围的旁路通道	1	1
GNDEN	输入	为真时将比较器的负极输入接地	0	0
HOLD	输入	采样保持信号。为低时采样信号，为高时保持信号抽样模拟信号。如果使用了增益级，那么 ACLK 为高时必须使该信号有效	1	1
IREFEN	输入	使能隙基准作为 DAC 的 AREF	0	0
MUXEN_7	输入	多工输入第 7 位	0	0
MUXEN_6	输入	多工输入第 6 位	0	0
MUXEN_5	输入	多工输入第 5 位	0	0
MUXEN_4	输入	多工输入第 4 位	0	0
MUXEN_3	输入	多工输入第 3 位	0	0
MUXEN_2	输入	多工输入第 2 位	0	0
MUXEN_1	输入	多工输入第 1 位	0	0

Table 105. ADC[1] 边界扫描信号 (Continued)

信号名称	相对 ADC 的方向	描述	不使用时的推荐输入	使用了推荐输入并且 CPU 不使用 ADC 时的输出值
MUXEN_0	输入	多工输入第 0 位	1	1
NEGSEL_2	输入	差分信号负极输入端的多工输入，第 2 位	0	0
NEGSEL_1	输入	差分信号负极输入端的多工输入，第 1 位	0	0
NEGSEL_0	输入	差分信号负极输入端的多工输入，第 0 位	0	0
PASSEN	输入	使能增益级的开关	1	1
PRECH	输入	比较器预充电输出锁存 (低有效)	1	1
SCTEST	输入	使能开关电容的 TEST 功能。 x10 增益级的输出送到 ADC_4 端口	0	0
ST	输入	如果信号在 AMPEN 变高后的起始两个 ACLK 周期内保持为高，则增益级输出将更快地稳定	0	0
VCCREN	输入	选择 Vcc 作为参考电压	0	0

Note: 1. 错误地设置 Figure 114 中的开关将会导致信号冲突 , 毁坏芯片。对于 Figure 114 , 比较器负极输入的 S&H 电路有数种输入选择。要确保只能在 ADC 引脚、能隙基准源或地之间选择一个。

如果不打算在扫描期间使用 ADC，则应该使用 Table 105 的推荐输入值。在扫描期间建议用户**不要**使用差分增益级。以开关电容方式实现的增益级要求快速的操作和准确的定时，这在扫描链中很难办到。因此关于差分增益级的详细操作在此不作详细介绍。

AVR ADC 基于 Figure 114 所示的模拟电路 , 该模拟电路通过数字逻辑实现了逐次逼近算法。进行边界扫描时，问题通常是确保施加的模拟电压在某些极限值范围内得到测量。通过下面不使用逐次逼近算法的方法可以很容易地实现这一目标：在数字 DAC[9:0] 线施加最小值，看比较器的输出是否为低；接着在数字 DAC[9:0] 线上施加最大值，并验证比较器的输出是否为高。

由于所有的模拟输入都与相应的数字端口引脚共享，纯连接测试不必使用 ADC。

使用 ADC 时要记住以下内容

- ADC 通道所用的端口引脚必须配置为禁止上拉功能的输入，以避免信号冲突
- 在正常模式下，使能 ADC 将启动一次空转换 (有 10 次比较操作)。建议用户在使能 ADC 后 , 在控制 / 观察任何 ADC 信号之前 , 等待至少 200ns , 或在使用第一个结果之前执行一次空转换。
- HOLD 信号为低时 (采样模式)， DAC 值必须稳定于中间值 0x200

作为例子 , 假设电源为 5.0V , AREF 连接到 V_{CC} , ADC 通道 3 中有 1.5V ± 5% 的输入信号。

$$\text{The lower limit is:} \quad \lceil 1024 \cdot 1{,}5V \cdot 0{,}95 / 5V \rceil = 291 = 0x123$$
$$\text{The upper limit is:} \quad \lceil 1024 \cdot 1{,}5V \cdot 1{,}05 / 5V \rceil = 323 = 0x143$$

没有在 Table 106 给出具体数值的参数使用 Table 105 的推荐值。Table 106 只列出了扫描链中的 DAC 值和端口引脚数据。“ 动作 ” 一栏描述了在填充后面所述扫描链寄存器之

前应该使用的JTAG指令。验证则是在被扫描入芯片的数据和扫描出芯片的数据之间进行的，它们位于表中的同一行。

Table 106. 使用 ADC 的算法

步骤	动作	ADCEN	DAC	MUXEN	HOLD	PRECH	PA3. Data	PA3. Control	PA3. Pull-up_ Enable
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	验证 COMP 的扫描值为 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	验证 COMP 的扫描值为 1	1	0x200	0x08	1	1	0	0	0

使用这种算法时，HOLD 信号的时间约束将限制 TCK 时钟频率的选择。由于这个算法要在 5 个步骤里保持 HOLD 为高，因此 TCK 时钟频率至少是扫描位数除以最大保持时间 $t_{hold,max}$ 的结果的 5 倍。

ATmega169 边界扫描顺序

Table 107显示了在边界扫描链选为数据通路时TDI和TDO之间的扫描顺序。位0是LSB，最先被扫描入芯片的位也是最先被扫描出芯片的位。***扫描次序依从输出引脚的顺序。因此端口 A 的位在扫描链中位于其他端口的后面。***此规则的例外是模拟电路的扫描链。模拟电路的内容占据了扫描链的最高位，而不管其连接到哪个物理引脚。在 Figure 108 中，PXn. Data 对应于 FF0，PXn. Control 对应于 FF1，PXn. Pull-up_enable 对应于 FF2。端口C的位2、3、4和5不存在于扫描链之内，因为这些引脚构成了JTAG使能时的TAP引脚。

Table 107. ATmega169 边界扫描顺序

位次序号	信号名称	信号所属模块
197	AC_IDLE	比较器
196	ACO	
195	ACME	
194	AINBG	
193	COMP	ADC
192	ACLK	
191	ACTEN	
190	PRIVATE_SIGNAL1[1]	
189	ADCBGEN	
188	ADCEN	
187	AMPEN	
186	DAC_9	
185	DAC_8	
184	DAC_7	
183	DAC_6	
182	DAC_5	
181	DAC_4	
180	DAC_3	
179	DAC_2	
178	DAC_1	
177	DAC_0	
176	EXTCH	
175	GNDEN	
174	HOLD	
173	IREFEN	
172	MUXEN_7	
171	MUXEN_6	
170	MUXEN_5	
169	MUXEN_4	

Table 107. ATmega169 边界扫描顺序 (Continued)

位次序号	信号名称	信号所属模块
168	MUXEN_3	ADC
167	MUXEN_2	
166	MUXEN_1	
165	MUXEN_0	
164	NEGSEL_2	
163	NEGSEL_1	
162	NEGSEL_0	
161	PASSEN	
160	PRECH	
159	ST	
158	VCCREN	
157	PE0.Data	端口 E
156	PE0.Control	
155	PE0.Pull-up_Enable	
154	PE1.Data	
153	PE1.Control	
152	PE1.Pull-up_Enable	
151	PE2.Data	
150	PE2.Control	
149	PE2.Pull-up_Enable	
148	PE3.Data	
147	PE3.Control	
146	PE3.Pull-up_Enable	
145	PE4.Data	
144	PE4.Control	
143	PE4.Pull-up_Enable	
142	PE5.Data	
141	PE5.Control	
140	PE5.Pull-up_Enable	
139	PE6.Data	
138	PE6.Control	
137	PE6.Pull-up_Enable	
136	PE7.Data	
135	PE7.Control	
134	PE7.Pull-up_Enable	
133	PB0.Data	端口 B

Table 107. ATmega169 边界扫描顺序 (Continued)

位次序号	信号名称	信号所属模块
132	PB0.Control	端口 B
131	PB0.Pull-up_Enable	
130	PB1.Data	
129	PB1.Control	
128	PB1.Pull-up_Enable	
127	PB2.Data	
126	PB2.Control	
125	PB2.Pull-up_Enable	
124	PB3.Data	
123	PB3.Control	
122	PB3.Pull-up_Enable	
121	PB4.Data	
120	PB4.Control	
119	PB4.Pull-up_Enable	
118	PB5.Data	
117	PB5.Control	
116	PB5.Pull-up_Enable	
115	PB6.Data	
114	PB6.Control	
113	PB6.Pull-up_Enable	
112	PB7.Data	
111	PB7.Control	
110	PB7.Pull-up_Enable	
109	PG3.Data	端口 G
108	PG3.Control	
107	PG3.Pull-up_Enable	
106	PG4.Data	
105	PG4.Control	
104	PG4.Pull-up_Enable	
103	PG5	(只可观测)
102	RSTT	复位逻辑 (只可观测)
101	RSTHV	
100	EXTCLKEN	主时钟 / 振荡器的使能信号
99	OSCON	
98	RCOSCEN	
97	OSC32EN	

Table 107. ATmega169 边界扫描顺序 (Continued)

位次序号	信号名称	信号所属模块
96	EXTCLK (XTAL1)	主时钟的时钟输入和振荡器输入 (只可观测)
95	OSCCK	
94	RCCK	
93	OSC32CK	
92	PD0.Data	端口 D
91	PD0.Control	
90	PD0.Pull-up_Enable	
89	PD1.Data	
88	PD1.Control	
87	PD1.Pull-up_Enable	
86	PD2.Data	
85	PD2.Control	
84	PD2.Pull-up_Enable	
83	PD3.Data	
82	PD3.Control	
81	PD3.Pull-up_Enable	
80	PD4.Data	
79	PD4.Control	
78	PD4.Pull-up_Enable	
77	PD5.Data	
76	PD5.Control	
75	PD5.Pull-up_Enable	
74	PD6.Data	
73	PD6.Control	
72	PD6.Pull-up_Enable	
71	PD7.Data	
70	PD7.Control	
69	PD7.Pull-up_Enable	
68	PG0.Data	端口 G
67	PG0.Control	
66	PG0.Pull-up_Enable	
65	PG1.Data	
64	PG1.Control	
63	PG1.Pull-up_Enable	
62	PC0.Data	端口 C
61	PC0.Control	

Table 107. ATmega169 边界扫描顺序 (Continued)

位次序号	信号名称	信号所属模块
60	PC0.Pull-up_Enable	端口 C
59	PC1.Data	
58	PC1.Control	
57	PC1.Pull-up_Enable	
56	PC2.Data	
55	PC2.Control	
54	PC2.Pull-up_Enable	
53	PC3.Data	
52	PC3.Control	
51	PC3.Pull-up_Enable	
50	PC4.Data	
49	PC4.Control	
48	PC4.Pull-up_Enable	
47	PC5.Data	
46	PC5.Control	
45	PC5.Pull-up_Enable	
44	PC6.Data	
43	PC6.Control	
42	PC6.Pull-up_Enable	
41	PC7.Data	
40	PC7.Control	
39	PC7.Pull-up_Enable	
38	PG2.Data	端口 G
37	PG2.Control	
36	PG2.Pull-up_Enable	
35	PA7.Data	端口 A
34	PA7.Control	
33	PA7.Pull-up_Enable	
32	PA6.Data	
31	PA6.Control	
30	PA6.Pull-up_Enable	
29	PA5.Data	
28	PA5.Control	
27	PA5.Pull-up_Enable	
26	PA4.Data	
25	PA4.Control	

Table 107. ATmega169 边界扫描顺序 (Continued)

位次序号	信号名称	信号所属模块
24	PA4.Pull-up_Enable	端口 A
23	PA3.Data	
22	PA3.Control	
21	PA3.Pull-up_Enable	
20	PA2.Data	
19	PA2.Control	
18	PA2.Pull-up_Enable	
17	PA1.Data	
16	PA1.Control	
15	PA1.Pull-up_Enable	
14	PA0.Data	
13	PA0.Control	
12	PA0.Pull-up_Enable	
11	PF3.Data	端口 F
10	PF3.Control	
9	PF3.Pull-up_Enable	
8	PF2.Data	
7	PF2.Control	
6	PF2.Pull-up_Enable	
5	PF1.Data	
4	PF1.Control	
3	PF1.Pull-up_Enable	
2	PF0.Data	
1	PF0.Control	
0	PF0.Pull-up_Enable	

Note: 1. PRIVATE_SIGNAL1 扫描入芯片时应该为 0。

边界扫描描述语言文件

边界扫描描述语言 (BSDL) 文件以标准的格式描述有边界扫描功能的器件，使之可被自动化的测试生成软件所使用。描述包括了边界扫描数据寄存器各个位的次序和功能。

Boot Loader 支持—RWW 自编程

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) 写入闪存，或者是从程序存储器读取代码。 Boot Loader 区的程序可以操作整个闪存， 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，如果系统已经不再需要 Boot Loader 这一特性了。 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。

Boot Loader 的特点

- **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 Flash 页 (1) 大小**
- **有效算法代码**
- **有效 RWW 支持**

Note: 1. 页是闪存的一个部分，由数个字节组成 (见 P260 Table 125) , 在编程过程中使用。页的组织构造不影响正常的操作。

应用及 Boot Loader Flash 区

Flash 由两个区构成 ,应用区和 Boot Loader 区 (见 Figure 116)。两个区的存储空间大小由 BOOTSZ 熔丝位配置 ,如 P253 Table 113 和 Figure 116 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。

应用区

应用区是闪存中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 P245 Table 109 。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。

BLS—Boot Loader 区

应用区用来存储应用代码，而 Boot Loader 程序必须保存在 BLS。这是因为只有在 BLS 运行时 SPM 指令才有效。 SPM 指令可以访问整个闪存，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 P245 Table 110 。

RWW 和非 RWW Flash 区

CPU 是否支持 RWW ,或者 CPU 是否在升级 Boot Loader 软件时停止 ,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，闪存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 P253 Table 114 和 P244 Figure 116 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

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，而不是在升级 Boot Loader 软件过程中实际被读取的那区。

RWW—同时读—写存储区

如果 Boot Loader 软件更新是对 RWW 区内的某一页进行编程，则可以从闪存中读取代码，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 P246 " 存贮程序存储控制器和状态寄存器 – SPMCSR" 。

NRWW—非同时读—写存储区

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 108. 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 115. RWW 与 NRWW

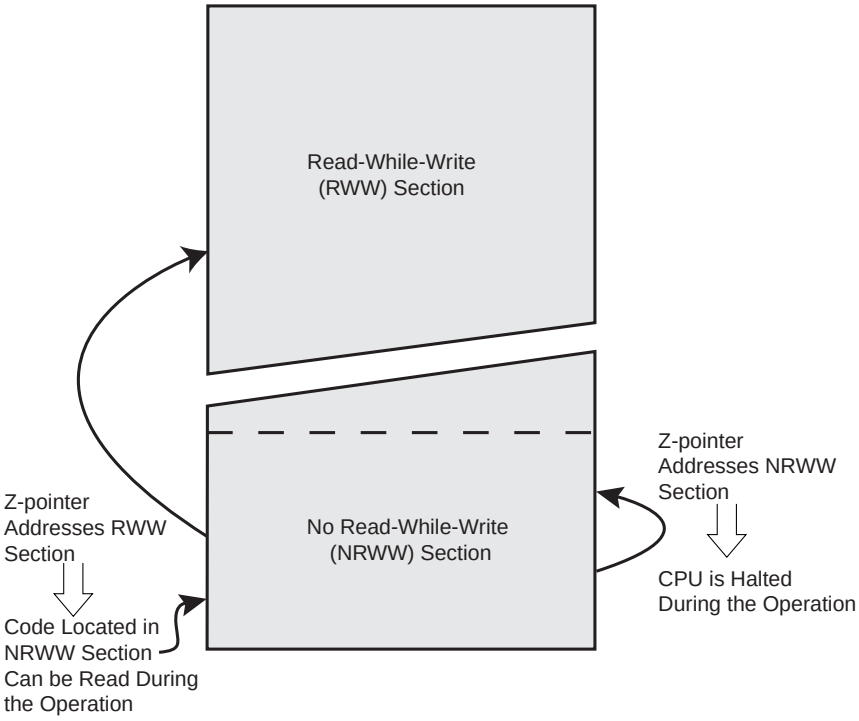
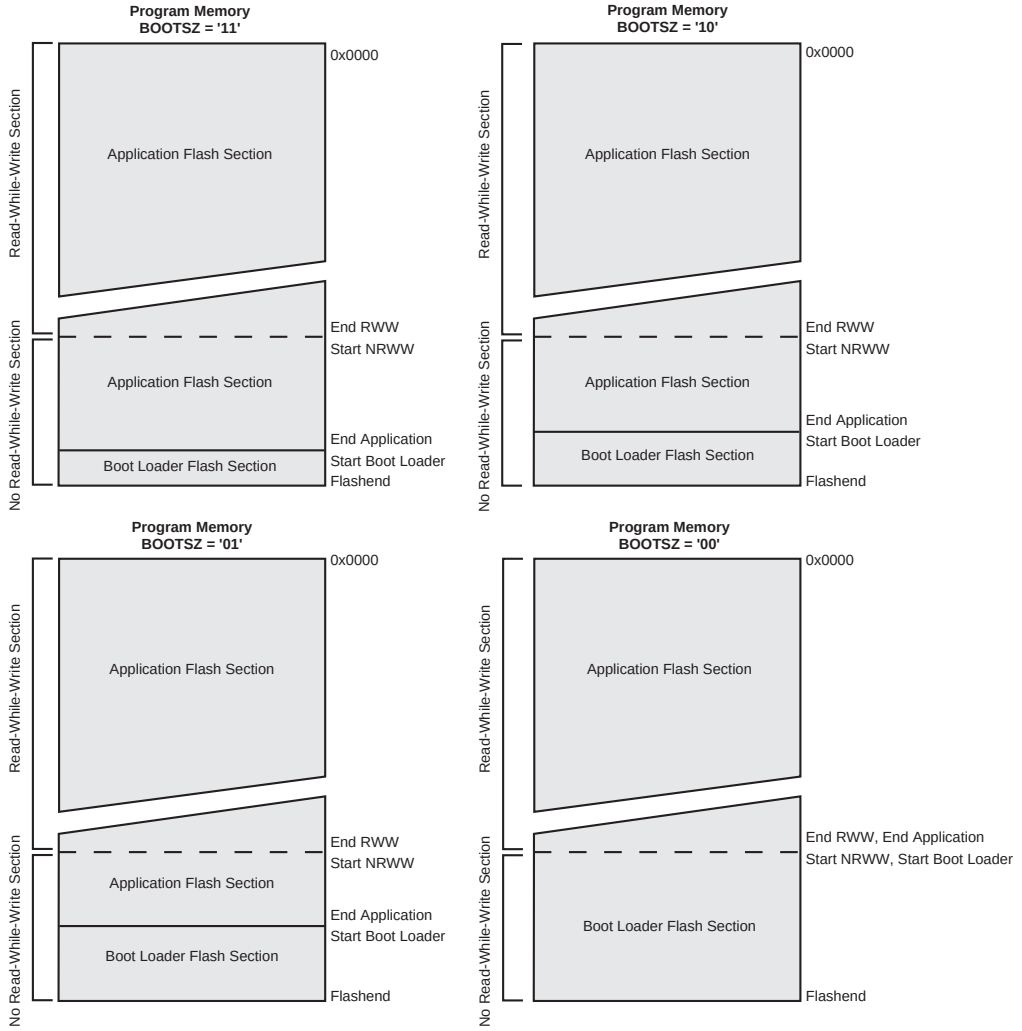


Figure 116. 存储区



Note: 1. 上图中的参数在 P253 Table 113 中给出。

引导加载锁定位

如果不需要 Boot Loader 功能，则整个闪存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 升级整个闪存区

详细内容请参见 Table 109 和 Table 110。Boot 锁定位可以通过软件、串行下载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 指令对闪存进行编程。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 指令对闪存进行读 / 写访问。

Table 109. Boot 锁定位 0 保护模式 (应用区)(1)

BLB0 模式	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 110. Boot 锁定位 1 保护模式 (Boot Loader 区)(1)

BLB1 模式	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 Boot Loader 程序

通过跳转指令或活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 111. 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 P253 Table 113)

Note: 1. “1” 意味着没被编程 , “0” 意味着已编程。

存贮程序存储控制器和状态寄存器—SPMCSR

存贮程序存储控制器和状态寄存器包括了控制 Boot Loader 操作所需的控制位。

位	7	6	5	4	3	2	1	0	
	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	SPMCSR
读 / 写	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
初始化值	0	0	0	0	0	0	0	0	

- **位 7 – 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。

- **位 6 – RWWSB:RWW 区忙标志**

启动对 RWW 区中自编程（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。

- **位 5 – Res: 保留位**

在 ATmega169 中为保留位，读返回值为 0。

- **位 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。如果闪存忙于页擦除或页写入 (SPMEN 置位)，RWW 区不能被使能。如果闪存加载与 RWWSRE 写操作同时发生，则闪存加载操作终止，加载的数据亦将丢失。

- **位 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 位和 SPMEN 位 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 P251 “ 通过软件读取熔丝位和锁定位 ”。

- **位 2 – PGWRT: 页写位**

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。

- **位 1 – PGERS: 页擦除位**

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。

- **位 0 – SPMEN: 存贮程序存储器使能位**

这一位使能紧接着的四个时钟周期内的 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。

自编程过程中寻址闪存

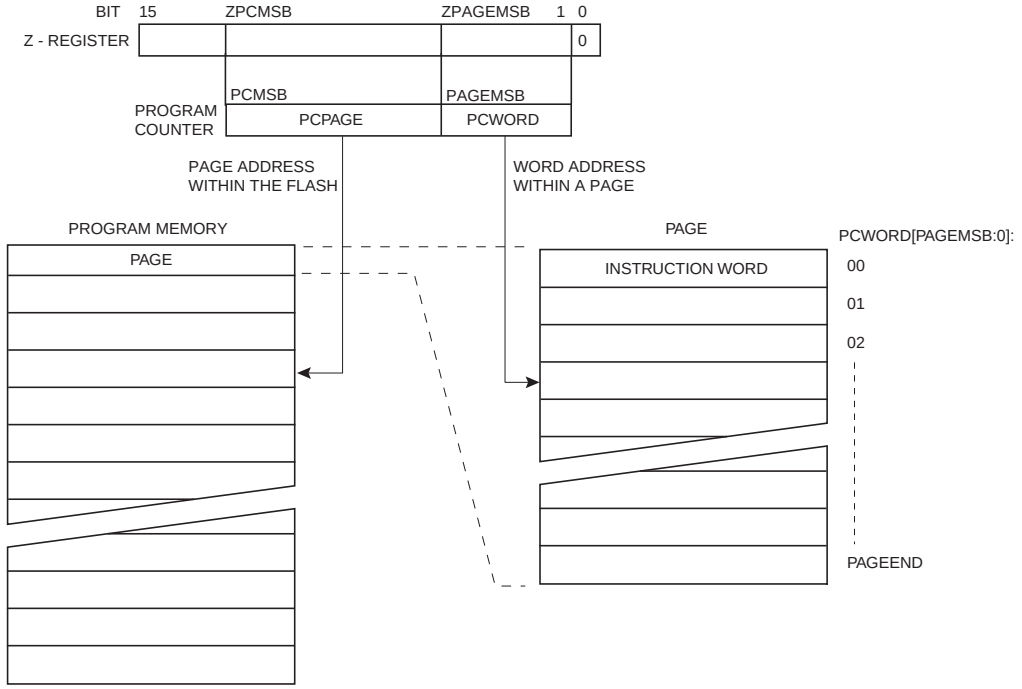
Z 指针用于 SPM 命令的寻址。

位	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 (P260 Table 125) 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 117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 指针的最低位 (位 Z0) 也使用到了 。

Figure 117. SPM(1) 的寻址



Note: 1. Figure 117 中所用的不同的变量在 P254 Table 115 列出。
2. PCPAGE 和 PCWORD 在 P260 Table 125 中列出。

自编程闪存

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

选择方案 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（如存在临时页缓冲区中），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 P253 “ 简单的 Boot Loader 汇编代码例子 ”。

利用 SPM 进行页擦除

执行页擦除操作首先需要设置 Z 指针的地址信息，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

写临时缓冲区 (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。

如果在 SPM 页加载操作过程中对 EEPROM 执行了写操作 ,则所有加载的数据都将丢失。

执行页写

执行页写操作首先需要设置 Z 指针的地址信息，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：在操作过程中 CPU 停止

使用 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 P45 “ 中断 ” 。

更新 BLS 时要考虑的问题

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 本身进行的误写操作会破坏整个 Boot Loader ,造成软件无法更新。如果不需要改变 Boot Loader，建议编程 Boot 锁定位 11，以防止不小心改变了 Boot Loader。

防止自编程过程中访问 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 P45 “ 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 P253 “ 简单的 Boot Loader 汇编代码例子 ” 。

通过 SPM 设置 Boot Loader 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。.

位	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 109 和 Table 110。

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闪存区。

EEPROM 写操作将阻止对 SPMCSR 寄存器的写操作

EEPROM 写操作会阻塞对闪存的编程，也会阻塞对熔丝位和锁定位的读操作。建议用户在对 SPMCSR 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。

通过软件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。

位	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 P257 Table 120 。

位	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 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 P257 Table 119 。

位	7	6	5	4	3	2	1	0
Rd	FHB7	FHB6	FHB5	FHB4	FHB3	FHB2	FHB1	FHB0

When reading the Extended Fuse byte, load in the Z-pointer. When an LPM instruction is executed within three cycles after the BLBSET and SPMEN bits are set in the SPMCSR, the value of the Extended Fuse byte (EFB) will be loaded in the destination register as shown below. Refer to P256 Table 118 for detailed description and mapping of the Extended Fuse byte

读取熔丝位扩展字节时，需要将 0x0002 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指令将把熔丝位扩展字节的值 (EFB) 加载到目的寄存器。更详细的说明及熔丝位扩展字节映射的细节请参见 P256 Table 118 。

位	7	6	5	4	3	2	1	0
Rd	–	–	–	–	EFB3	EFB2	EFB1	EFB0

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。

防止闪存损毁

V_{CC} 低于工作电压时，CPU 和闪存正常工作无法保证，闪存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闪存一样存在。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闪存内容。第一，闪存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。

通过遵循以下设计建议可以避免闪存被破坏 (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：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 SPMCSR 寄存器，从而保护闪存被无意识得修改掉。

使用 SPM 时的闪存编程时间

片内校准的 RC 振荡器用于闪存寻址时序控制。 Table 112 给出了 CPU 访问闪存的典型编程时间。

Table 112. SPM 编程时间

符号	最小编程时间	最大编程时间
闪存写操作 (通过 SPM 实现页擦除、页写、及写锁定位)	3.7 ms	4.5 ms

简单的 Boot Loader 汇编代码例子

```
  ;- 本例程将 RAM 中的一页数据到入闪存
  ;Y 指针指向 RAM 的第一个数据单元
  ;Z 指针指向闪存的第一个数据单元
  ;- 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 转移到闪存页缓冲区
  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

  ; 执行页写操作
  subi  ZL, low(PAGESIZEB)          ; 复位指针
  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)          ; 复位指针
  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, SPMCSR
  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, SPMCSR
  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   SPMCSR, spmcrval
  spm
  ; 恢复 SREG ( 如果中断原本是使能的，则使能中断 )
  out   SREG, temp2
  ret
```

ATmega169 Boot Loader 参数

In Table 113 through Table 115, the parameters used in the description of the Self-Programming are give

自编程描述中所用的参数在 Table 113 到 Table 115 中给出。

Table 113. Boot 区大小配置 (1)

BOOTSZ1	BOOTSZ0	Boot 尺寸	页数	应用闪存区	Boot Loader 闪存区	应用区结束地址	Boot 复位寻址 (Boot Loader 起始地址)
1	1	128 字	2	0x0000 - 0x1F7F	0x1F80 - 0x1FFF	0x1F7F	0x1F80
1	0	256 字	4	0x0000 - 0x1EFF	0x1F00 - 0x1FFF	0x1EFF	0x1F00
0	1	512 字	8	0x0000 - 0x1DFF	0x1E00 - 0x1FFF	0x1DFF	0x1E00
0	0	1024 字	16	0x0000 - 0x1BFF	0x1C00 - 0x1FFF	0x1BFF	0x1C00

Note: 1. 不同的 BOOTSZ 熔丝位配置请参见 Figure 116。

Table 114. RWW 界限 (1)

Flash 区	页数	寻址范围
同时读 - 写区 (RWW)	112	0x0000 - 0x1BFF
非同时读 - 写区 (NRWW)	16	0x1C00 - 0x1FFF

Note: 1. 关于两个区的详细说明请见 P244 “NRWW — 非同时读 - 写存储区 ” 和 P243 “RWW — 同时读 - 写存储区 ”。

Table 115. Figure 117 中所用变量的说明及 Z 指针的映射 (1)

变量		相应的 Z 指针数据	描述
PCMSB	12		程序计数器的最高位 (程序计数器为 13 位 PC[12:0])
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		Z13	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[12:6]	Z13: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:Z14: 始终忽略
Z0: 对所有的 SPM 命令都为 0，对 LPM 指令则用于字节选择。
关于自编程过程中 Z 指针的使用请参见 P248 “ 自编程过程中寻址闪存 ”

存储器编程

程序存储器和数据存储器锁定位

ATmega169 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 117 中列出的附加性能。锁定位只能通过芯片擦除命令擦写为 “1”。

Table 116. 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 117. 锁定位保护模式 (1)(2)

存储器锁定位			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	在并行和串行编程模式中闪存和 EEPROM 的进一步编程被禁止，熔丝位被锁定。(1)
3	0	0	在并行和串行编程模式中闪存和 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 117. 锁定位保护模式 (1)(2) (Continued) (Continued)

存储器锁定位			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。

Notes: 1. 在编程 LB1 和 LB2 前先编程熔丝位和 Boot 锁定位。
2. “1” 表示未被编程 , “0” 表示被编程。

熔丝位

ATmega169 中有三个熔丝位字节。Table 118 到 Table 120 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 118. 熔丝位扩展字节

熔丝位低字节	位号	描述	默认值
–	7	–	1
–	6	–	1
–	5	–	1
–	4	–	1
BODLEVEL2(1)	3	BOD 触发电平	1 (未被编程)
BODLEVEL1(1)	2	BOD 触发电平	1 (未被编程)
BODLEVEL0(1)	1	BOD 触发电平	1 (未被编程)

Notes: 1. BODLEVEL 熔丝位对应的具体电平请参见 P39 Table 17 。

Table 119. 熔丝位高位字节

熔丝位高位字节	位号	描述	默认值
OCDEN(4)	7	使能 OCD	1 (未被编程，OCD 禁止)
JTAGEN(5)	6	使能 JTAG	0 (被编程，JTAG 使能)
SPIEN(1)	5	使能串行程序和数据下载	0 (被编程，SPI 编程使能)
WDTON(3)	4	看门狗定时器一直启用	1 (未被编程)
EESAVE	3	执行芯片擦除时 EEPROM 的内容不擦除	1 (未被编程, EEPROM 内容不保留)
BOOTSZ1	2	选择 Boot 区存储器大小 (详细内容见 Table 121)	0 (被编程)(2)
BOOTSZ0	1	选择 Boot 区存储器大小 (详细内容见 Table 121)	0 (被编程)(2)
BOOTRST	0	选择复位向量	1 (未被编程)

Note: 1. 在串行编程模式下 SPIEN 熔丝位不可访问。
2. BOOTSZ1..0 的默认值导致最大的 Boot 区。详细内容见 P259 Table 121 。
3. 详细内容见 P42 “ 看门狗定时器控制寄存器 – WDTCR” 。
4. 芯片出厂时OCDEN熔丝位不编程。这是因为 ,不管锁定位和JTAGEN熔丝位的设置怎样，OCDEN 熔丝位编程使时钟系统的某些部分在所有的睡眠模式下都继续运行，这将增加功耗。
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 120. 熔丝位低位字节

熔丝位低位字节	位号	描述	默认值
CKDIV8(4)	7	时钟 8 分频	0 (被编程)
CKOUT(3)	6	时钟输出	1 (未被编程)
SUT1	5	选择起动时间	1 (未被编程)(1)
SUT0	4	选择起动时间	0 (被编程)(1)
CKSEL3	3	选择时钟源	0 (被编程)(2)
CKSEL2	2	选择时钟源	0 (被编程)(2)
CKSEL1	1	选择时钟源	1 (未被编程)(2)
CKSEL0	0	选择时钟源	0 (被编程)(2)

Note: 1. 对于默认时钟源 ,SUT1..0 的默认值导致最大的起动时间。详细内容见 P37 Table 16 。
2. CKSEL3..0的默认设置导致了片内RC振荡器运行于8 MHz。详细内容见P26 Table 6 。
3. CKOUT 熔丝位允许系统时钟从 PORTE7 输出。详细内容见 P29 “ 时钟输出缓冲器 ” 。
4. 详细内容见 P30 “ 系统时钟预分频器 ” 。

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。

熔丝位的锁存

器件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。

对于 ATmega169 这三个标志字节是：

1. 0x000: 0x1E (表示由 Atmel 公司生产)
2. 0x001: 0x94 (表示芯片包含 16K 闪存)
3. 0x002: 0x05 (当 0x001 字节的内容为 0x94 时表示这是 ATmega169)

校准字节

ATmega169 内部 RC 振荡器的校准值保存于校准字节。这个字节位于标识地址空间 0x000 的高位字节。在复位期间，该字节被自动写入 OSCCAL 寄存器以确保校准的 RC 振荡器频率的正确性。

并行编程参数，引脚映射及命令

这部分描述了如何对 ATmega169 的闪速程序存储器，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 ATmega169 的相关引脚以并行编程信号的名称进行引用，如 Figure 118 和 Table 121 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 123。

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 124。

Figure 118. 并行编程

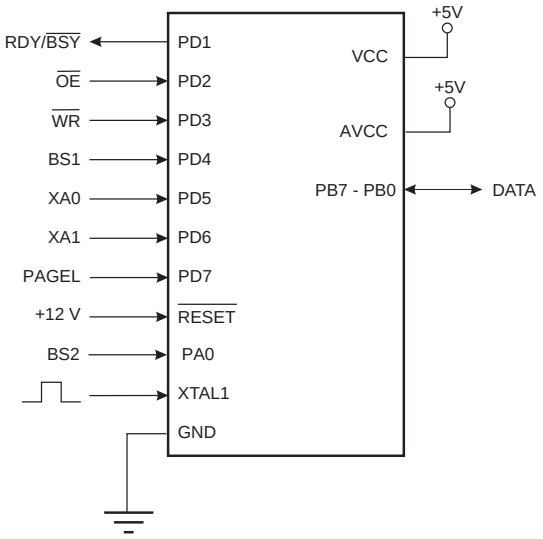


Table 121. 引脚名称映射

编程模式信号的名称	引脚名称	I/O	功能
RDY/$\overline{BSY}$	PD1	O	0: 设备忙于编程 , 1: 设备等待新的命令。
$\overline{OE}$	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 122. 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 123. XA1 和 XA0 的编码

XA1	XA0	给 XTAL1 施加脉冲激发的动作
0	0	加载闪存或 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 124. 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 125. 一页包含的字和闪存中的页数

闪存大小	页大小	PCWORD	页号	PCPAGE	PCMSB
8K 字 (16K 字节)	64 字	PC[5:0]	128	PC[12:6]	12

Table 126. 一页包含的字和 EEPROM 中的页数

EEPROM 大小	页大小	PCWORD	页数	PCPAGE	EEAMSB
512 字节	4 字节	EEA[1:0]	128	EEA[8:2]	8

串行编程引脚映射

Table 127. 串行编程时的引脚映射

符号	引脚	I/O	描述
MOSI	PB2	I	串行数据输入
MISO	PB3	O	串行数据输出 t
SCK	PB1	I	串行时钟

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 P259 Table 122 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
5. 在发送新命令之前至少等待 50 µs

高效编程的几点考虑

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被擦除。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 P260 Table 125 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 120 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 P263 Figure 119 。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

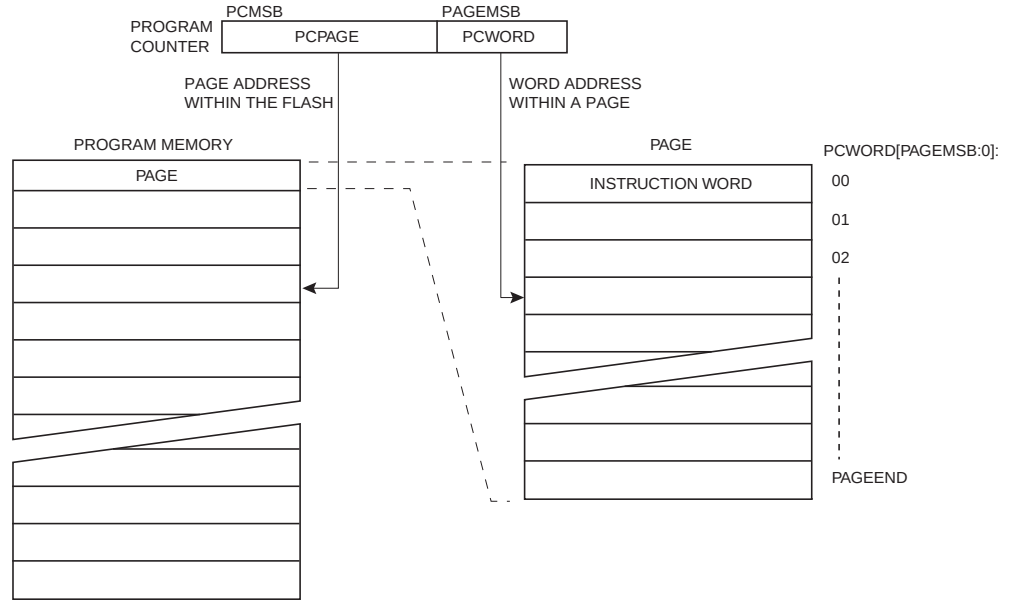
1. 给 $\overline{WR}$ 提供一个负脉冲，对整页数据进行编程， RDY/$\overline{BSY}$ 变低
2. 等待 RDY/$\overline{BSY}$ 变高 (见 Figure 120 的信号波形)

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

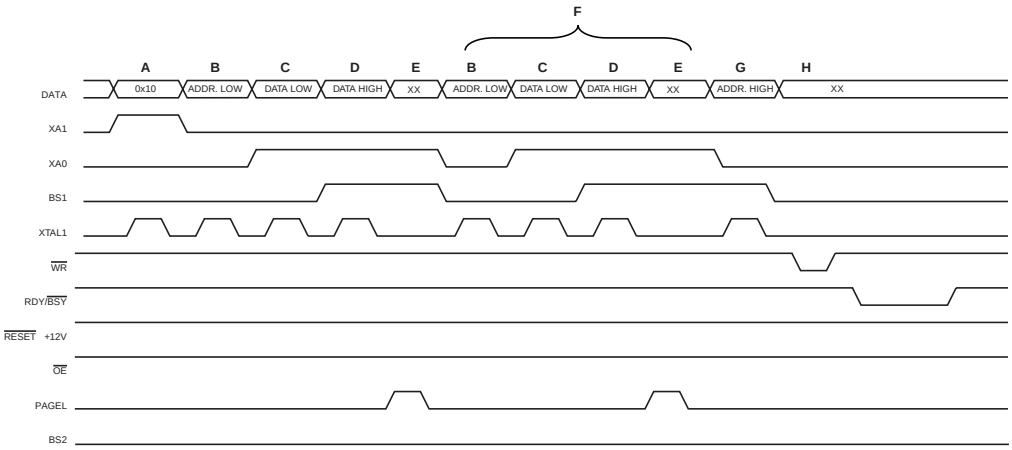
1. 1. 将 XA1、 XA0 置为 10，启动命令加载操作
2. DATA 赋值为 “ 0000 0000” ，这是不操作指令
3. 给 XTAL1 提供一个正脉冲，加载命令，内部写信号复位

Figure 119. 对以页为组织单位的 Flash 进行寻址 (1)



Note: 1. PCPAGE 及 PCWORD 列于 P260 Table 125 。

Figure 120. 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 Table 126 on page 271 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 P262 “ 对 Flash 进行编程 ”) :

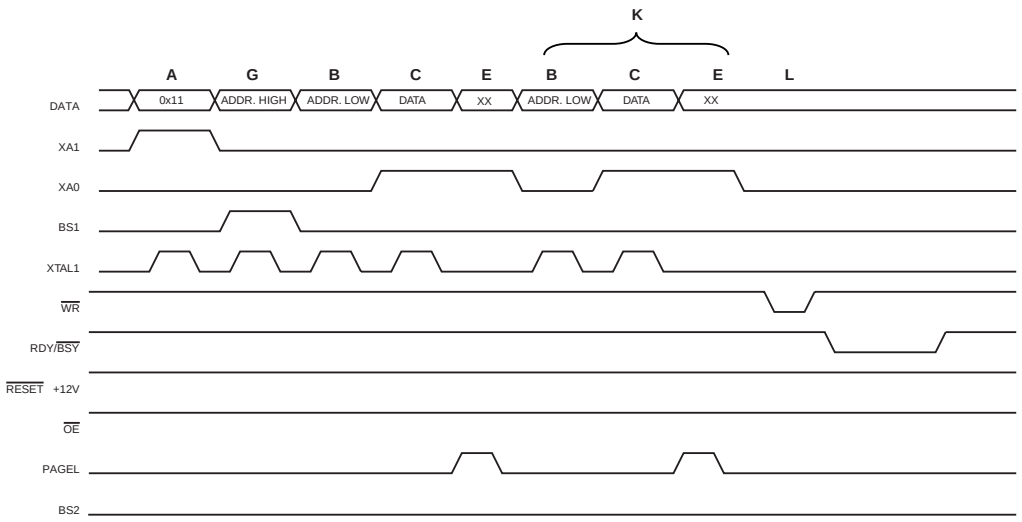
1. A：加载命令 “0001 0001”
2. G：加载地址高位字节 (0x00 - 0xFF)
3. B：加载地址低位字节 (0x00 - 0xFF)
4. C：加载数据 (0x00 - 0xFF)
5. E：锁存数据 (给 PAGEL 提供一个正脉冲)

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 BS 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 121)

Figure 121. EEPROM 编程波形



读取 Flash 的内容

读 Flash 存储器的过程如下 (命令及地址加载细节见 P262 “ 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 1， BS1 置 0，然后从 DATA 读出 Flash 字的低位字节
5. 将 BS 置 1，然后从 DATA 读出 Flash 字的高位字节

将 $\overline{OE}$ 置 1

读取 EEPROM 的内容

读存储器的步骤如下 (命令及地址加载细节见 P262 “ 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 P262 “ 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. 给 $\overline{WR}$ 提供一个负脉冲 , 并等待 RDY/$\overline{BSY}$ 变高

对熔丝高位进行编程

对熔丝高位的编程步骤如下 (命令及数据装在细节见 P262 “ 对 Flash 进行编程 ”) :

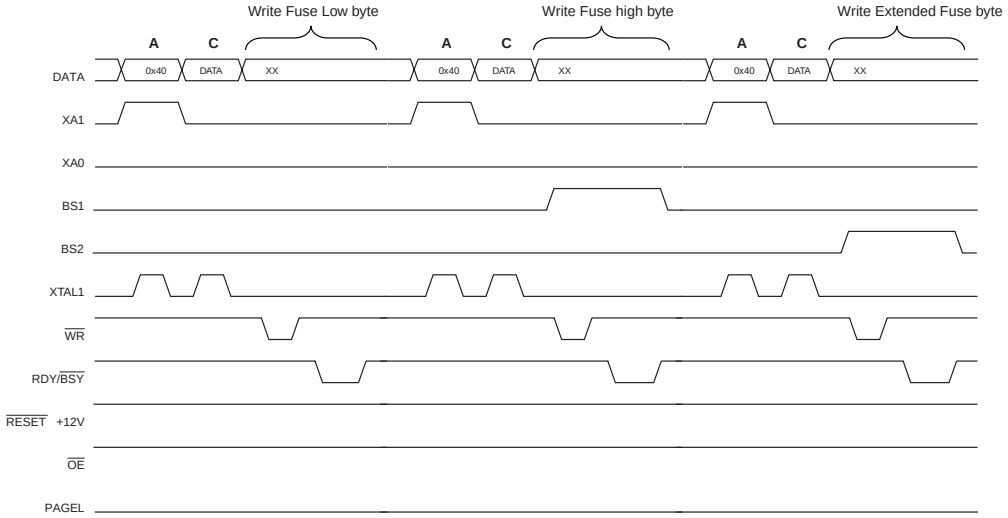
1. A : 加载命令 “0100 0000”
2. C : 加载数据高字节 , 若某一位为 0 表示需要进行编程 , 否则需要擦除
3. 将 BS1 置 1、 BS2 置 0 , 选择高位数据字节
4. 给 $\overline{WR}$ 提供一个负脉冲并等待 RDY/$\overline{BSY}$ 变高
5. 将 BS1 置 0 , 选择低位字节

对扩展熔丝位进行编程

对扩展熔丝位的编程步骤如下 (命令及数据装在细节见 P262 “ 对 Flash 进行编程 ”) :

1. 1. A : 加载命令 “0100 0000”
2. 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. 3. 将 BS1 置 1、 BS2 置 1 , 选择扩展数据字节
4. 4. 给 $\overline{WR}$ 提供一个负脉冲并等待 RDY/$\overline{BSY}$ 变高
5. 5. 将 BS2 置 0 , 选择低位字节

Figure 122. 熔丝位编程波形



锁定位编程

锁定位编程步骤如下 (命令及数据装在细节见 P262 “ 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。如果芯片已经处于 LB 模式 3(LB1 及 LB2 被编程) , 那么就不可能通过外部编程命令对锁定位进行编程
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。

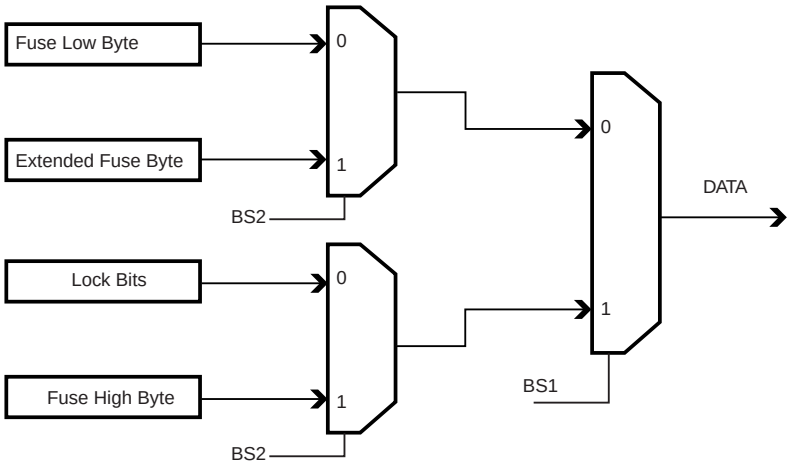
读取熔丝位及锁定位

读取熔丝位及锁定位的步骤如下 (命令加载细节见 P262 “ 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 $\overline{OE}$ 和 BS1 置 0 ,BS2 置 1 ,然后从 DATA 读取扩展熔丝位的状态 (0 表示已编程)
5. 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)
6. 将 $\overline{OE}$ 置 1。

Figure 123. 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 (命令及地址加载参考 P262 “ 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 $\overline{OE}$ 置 0，BS1 置 1，然后从 DATA 读取标识字节
4. 将 $\overline{OE}$ 置 1

读取校准字节

读取校准字节的算法如下 (命令及地址加载参考 P262 “ 对 Flash 进行编程 ”) :

1. A : 加载命令 “0000 1000”
2. B : 加载地址低字节 0x00
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 124. 并行编程时序，包括一些常规的时序要求

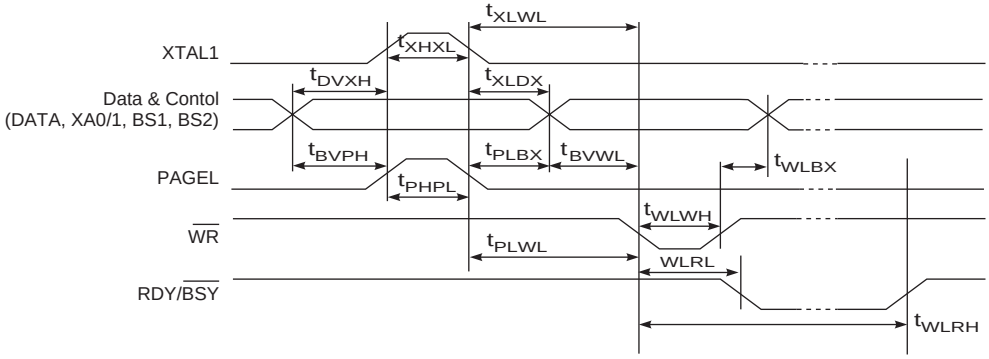
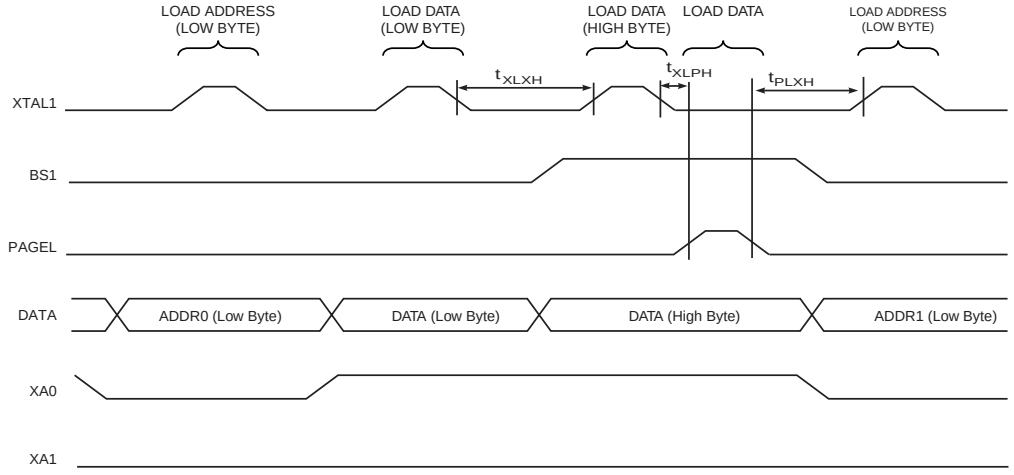
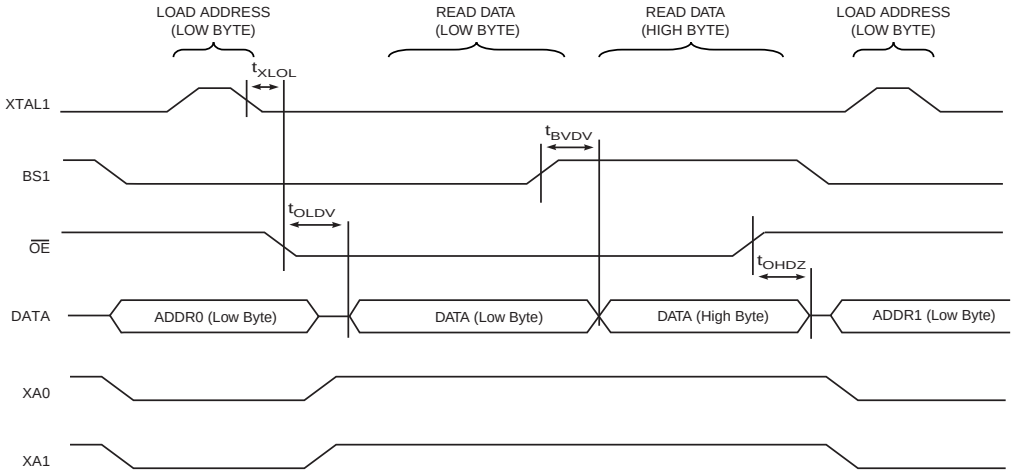


Figure 125. 并行编程时序，有时序要求的加载序列 (1)



Note: 1. Figure 124 给出的时序要求 (t_{DVXH}、 t_{XHXL} 及 t_{XLDX}) 也适用于加载操作。

Figure 126. 并行编程时序，有时序要求的读序列 (同一页)(1)



Note: 1. Figure 124 给出的时序要求 (即 t_{DVXH}、 t_{XHXL} 及 t_{XLDX}) 也适用于读操作。

Table 128. 并行编程参数 V_{CC} = 5V ± 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

Table 128. 并行编程参数 V_{CC} = 5V ± 10% (Continued)

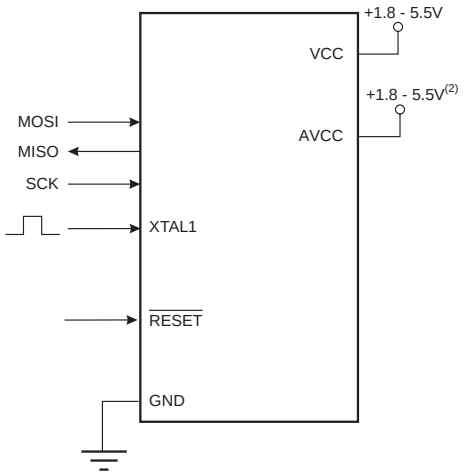
符号	参数	最小值	典型值	最大值	单位
t_{PLXH}	从 PAGEL 低到 XTAL1 高	150			ns
t_{BVPH}	PAGEL 为高之前 BS1 有效	67			ns
t_{PHPL}	PAGEL 为高时的脉宽	150			ns
t_{PLBX}	PAGEL 为低之后 BS1 保持	67			ns
t_{WLBX}	$\overline{WR}$ 为低之后 BS2/1 保持	67			ns
t_{PLWL}	从 PAGEL 低到 $\overline{WR}$ 为低	67			ns
t_{BVWL}	BS1 有效至 $\overline{WR}$ 为低	67			ns
t_{WLWH}	$\overline{WR}$ 为低时的脉宽	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 为三态			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。

串行下载

当 $\overline{RESET}$ 位低电平时，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。 P260 Table 127 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。

Figure 127. 串行编程及校验 (1)



Notes: 1. I 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 1.8 - 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

串行编程算法

向 ATmega169 串行写入数据时，数据在 SCK 的上升沿得以锁存。

从 ATmega169 读取数据时，数据在 SCK 的下降沿输出。时序细节见 Figure 128。

在串行编程模式下对 ATmega169 进行编程及校验时，应遵循以下的步骤 (见 Table 130 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。页的大小见 P260 Table 125 。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 8 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 129)。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 129)。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 RESET 拉高开始正常操作。
8. 下电序列 (如果需要) :
   将 RESET 置 “1”。
   切断 V_{CC}。

Flash 数据查询

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 129。

EEPROM 数据查询

As a chip-erased device contains 0xFF in all locations, programming of addresses that are meant to contain 0xFF, can be skipped. This does not apply if the EEPROM is re-programmed without chip erasing the device. In this case, data polling cannot be used for the value 0xFF, and the user will have to wait at least t_{WD_EEPROM} before programming the next byte. See Table 129 for t_{WD_EEPROM} va

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 129。 .

Table 129. 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_FUSE}	4.5 ms
t_{WD_FLASH}	4.5 ms
t_{WD_EEPROM}	9.0 ms
t_{WD_ERASE}	9.0 ms

Figure 128. 串行编程波形图

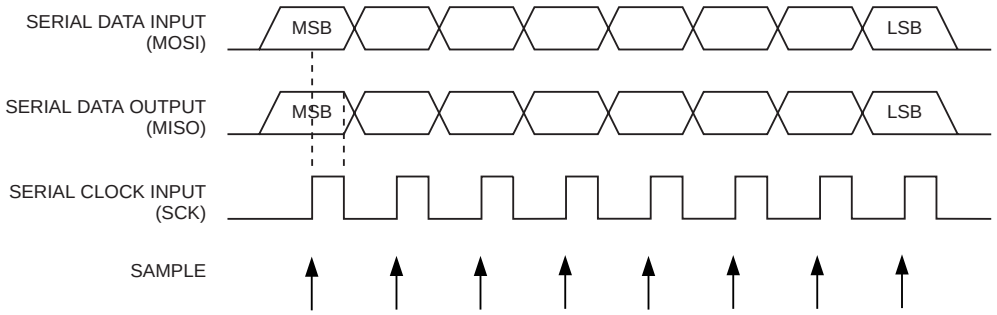


Table 130. 串行编程指令集

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
全片擦除	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	擦除 EEPROM 及 Flash
读程序存储器	0010 **H**000	000**a 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	000x 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	000**a 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	000x 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	000x xx**aa**	**bbbb bbbb**	**iiii iiii**	向 EEPROM 地址 **a**:**b** 处中写入数据 **o**
加载 EEPROM 存储器页 (页寻址)	1100 0001	0000 0000	0000 00**bb**	**iiii iiii**	将数据 **i** 加载到 EEPROM 存储器页缓冲区。数据加载完毕后对 EEPROM 页进行编程
写EEPROM存储器页(页寻址)	1100 0010	00xx xx**aa**	**bbbb bb**00	xxxx xxxx	对地址为 **a**:**b** 的 EEPROM 执行页写操作
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 P255 Table 116 。 .
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 P255 Table 116 。
读标识字节	0011 0000	000x xxxx	xxxx xx**bb**	**oooo oooo**	从地址 **b** 读取标识字节 **o**

Table 130. 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	
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 P196 Table 88 。
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 P188 Table 87 。
写扩展熔丝位	1010 1100	1010 0100	xxxx xxxx	**xxxx xxii**	0 表示已编程，1 表示未编程。见 P256 Table 118 。
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 P196 Table 88 。
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 P188 Table 87 。
读扩展熔丝位	0101 0000	0000 1000	xxxx xxxx	**oooo oooo**	读扩展熔丝位。0 表示已编程，1 表示未编程。细节见 P256 Table 118 。
读校准字节	0011 1000	000x xxxx	0000 0000	**oooo oooo**	读校准字节
查询 RDY/$\overline{\text{BSY}}$	1111 0000	0000 0000	xxxx xxxx	xxxx xxx**o**	**o** = 1 表示编程操作正在进行。等到它为 0 后可以执行其他命令。

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，4 **i** = 数据输入，x = 无用途

SPI 串行编程特性参数

SPI 模块的特性参数请见 P288 “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。

在编程过程中， TCK 输入的时钟必须小于芯片的最大时钟频率。不能通过系统时钟预分频器将 TCK 时钟频率降低。

LSB 是第一个移入 / 移出移位寄存器的位。

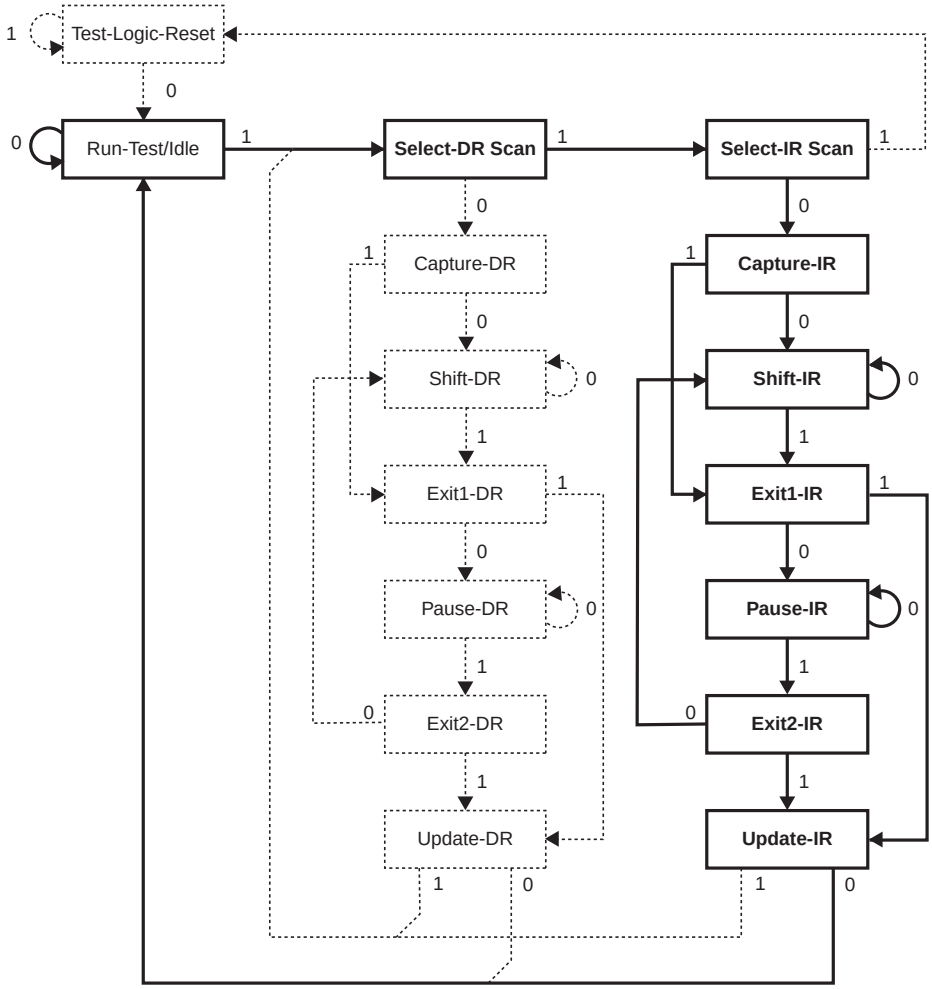
JTAG 专用编程指令

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 129。

Figure 129. 改变指令字的状态机序列



AVR_RESET (0xC)

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 (0x4)

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 (0x5)

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 131）

PROG_PAGELOAD (0x6)

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。8位的Flash数据字节寄存器被用作数据寄存器。在物理上这是编程命令寄存器的8个LSB。活动状态有：

- Shift-DR：Flash 数据字节寄存器的内容被通过输入的 TCK 时钟进行移位
- Update-DR：.Flash 数据字节寄存器的内容被复制到一个临时寄存器。在其后的 11 个 TCK 时钟里写序列被启动，将临时寄存器中的内容加载到 Flash 页缓冲区。对于每一个 Update-DR 状态 AVR 自动在写低字节和高字节之间进行切换。对于进入 PROG_PAGELOAD 命令之后的第一个 Update-DR 状态，AVR 首先写低字节。除了第一个字节，在写低字节之前程序计数器提前加 1。这样可以保证第一个数据正确写入由 PROG_COMMANDS 所设置的地址，同时在加载页缓冲区最后一个数据时不会使程序计数器指向下一页。

PROG_PAGEREAD (0x7)

AVR专用的JTAG指令。其功能是通过JTAG接口读取Flash的内容。8位的Flash数据字节寄存器用作数据寄存器。在物理上这是编程命令寄存器的 8 个 LSB。活动状态有：

- Capture-DR：选中的 Flash 存储单元的内容被捕捉到 Flash 数据字节寄存器中。对于每一个 Capture-DR 状态 AVR 自动在读低字节和读字节之间进行切换。对于进入 PROG_PAGEREAD 命令之后的第一个 Capture-DR 状态，AVR 首先读低字节。程序计数器在读每个高字节之后加 1，包括第一个字节。这样可以保证正确获取由 PROG_COMMANDS 设置好地址的第一个字节的内容，并保证读此页最后一个字节时程序计数器进入下一页。
- Shift-DR：Flash 数据字节寄存器的内容通过输入的 TCK 时钟进行移位操作。

数据寄存器

数据寄存器由 JTAG 指令寄存器选取，如 P274 “JTAG 专用编程指令 ” 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 数据字节寄存器

复位寄存器

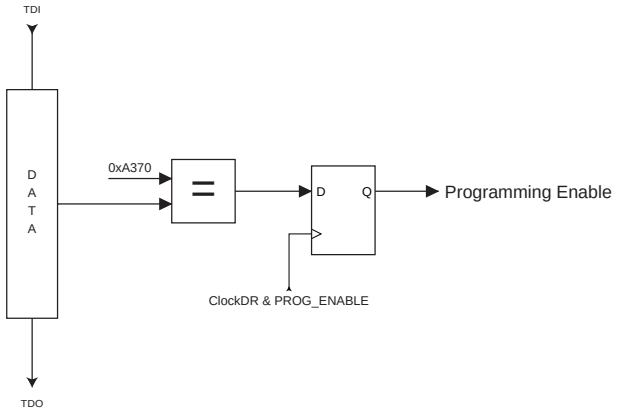
复位寄存器是一个测试数据寄存器，用来在编程期间复位芯片。进入编程模式之前首先要复位器件。

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 P24 “ 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 P225 Figure 107 所示。

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 0b1010_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 130. 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 131。编程命令移入的状态序列请参见 Figure 132。

Figure 131. 编程命令寄存器

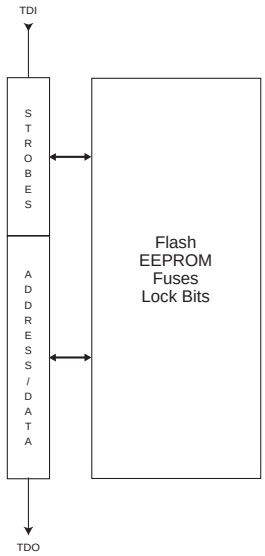


Table 131. JTAG 编程指令集

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 = 无用途

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. 装在低位地址	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)
5c. 加载低位地址	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	

Table 131. JTAG 编程指令集 (Continued)

(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 = 无用途

指令	TDI 序列	TDO 序列	注意
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	0111011_00000000 0111001_00000000 0111011_00000000 0111011_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	0110111_00000000 0110101_00000000 0110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
6g. 检测写熔丝位是否结束	0110111_00000000	xxxxx**o**x_xxxxxxxx	(2)
6h. 加载数据低位字节 (7)	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	(3)
6i. 写熔丝位低位字节	0110011_00000000 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
6j. 检测熔丝位写完	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. 读扩展熔丝位字节 (6)	0111010_00000000 0111011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
8c. 读熔丝位高字节 (7)	0111110_00000000 0111111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
8d. 读熔丝位低字节 (8)	0110010_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
8e. 读锁定位 (9)	0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xx**oooooo**	(5)

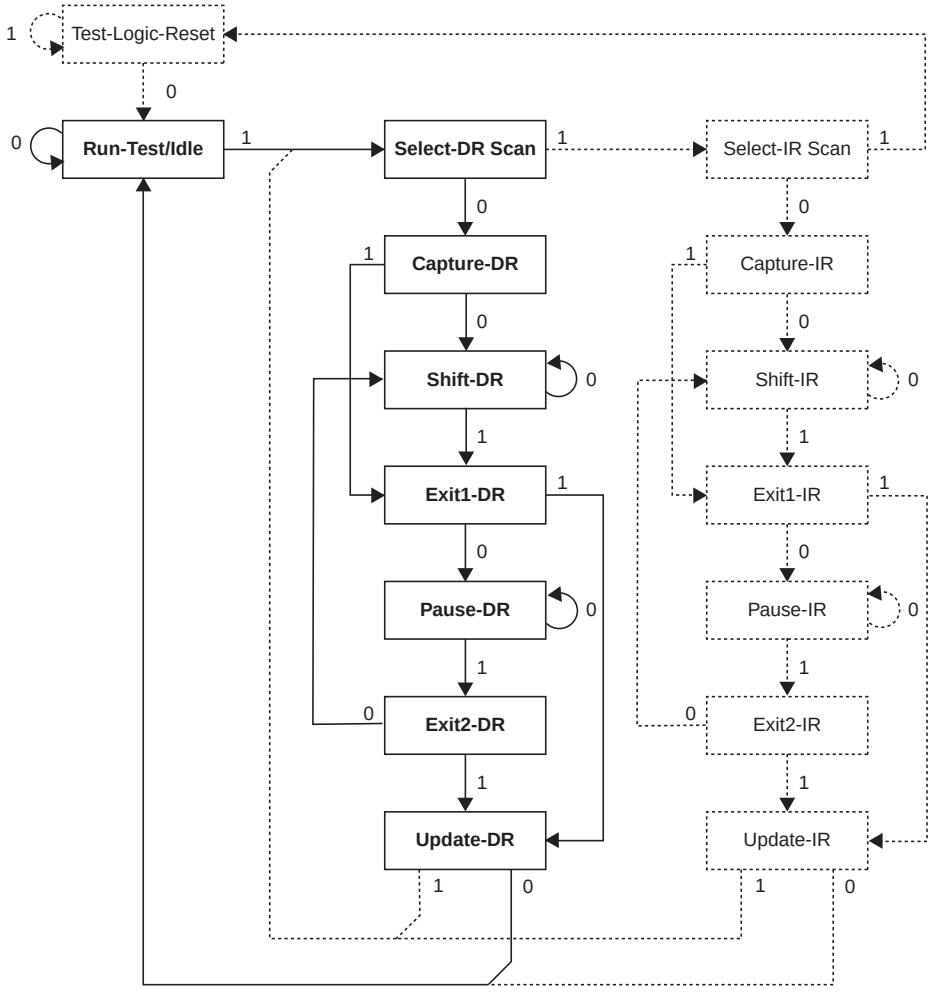
Table 131. JTAG 编程指令集 (Continued)

(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 = 无用途

指令	TDI 序列	TDO 序列	注意
8f. 读熔丝位及锁定位	0111010_00000000 0111110_00000000 0110010_00000000 0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo** xxxxxxx_**oooooooo** xxxxxxx_**oooooooo** xxxxxxx_**oooooooo**	(5) 熔丝位扩展字节 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**	
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. “0” = 编程 “1” = 不编程
4. “0” = 编程 “1” = 不编程
5. “0” = 编程 “1” = 未编程
6. 对应的扩展熔丝位字节的位映射列于 P256 Table 118
7. 对应的熔丝位高字节的位映射列于 P257 Table 119
8. 对应的熔丝位低字节的位映射列于 P257 Table 120
9. 对应的锁定位字节的位映射列于 P255 Table 116
10. 忽略超过 PCMSB 及 EEAMSB(Table 125 及 Table 126) 的地址
11. TDI 及 TDO 序列都以二进制表示

Figure 132. 改变 / 读取数据字的状态机序列



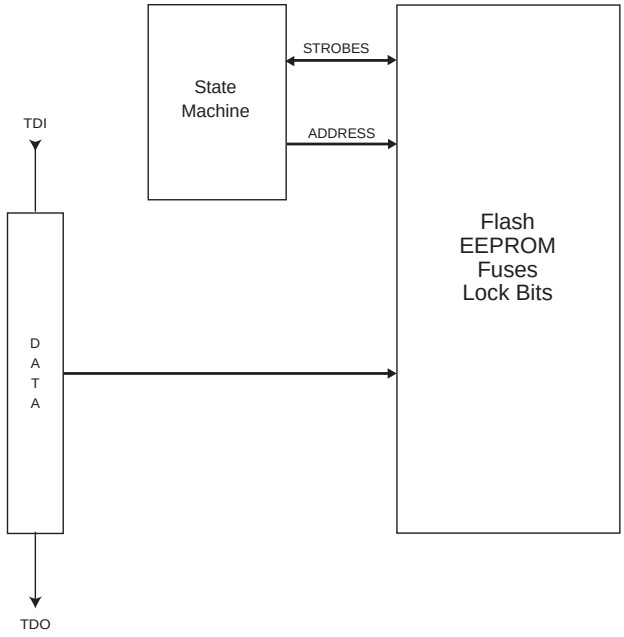
Flash 数据字节寄存器

Flash 数据字节寄存器提供了一个很有效的办法在进行页写操作之前加载整个 Flash 页缓冲区，或者读出 / 校验 Flash 的内容。片内有一个状态机设置 Flash 的控制信号，并感知 Flash 的锁存信号，因此只有数据字需要移入 / 移出。

Flash数据字节寄存器实际上包含了一个8位的扫描链及一个8位的临时寄存器。在页加载过程中， Update-DR 将扫描链的内容复制到临时寄存器，并在其后 11 个 TCK 时钟内将临时寄存器的内容加载到 Flash 页缓冲区。对于每一个 Update-DR 状态 AVR 自动在写低字节和高字节之间进行切换。对于进入 PROG_PAGELOAD 命令之后的第一个 Update-DR 状态，AVR 首先写低字节。除了第一个字节，在写低字节之前程序计数器提前加 1。这样可以保证第一个数据正确写入由 PROG_COMMANDS 所设置的地址，同时在加载页缓冲区最后一个数据时不会使程序计数器指向下一页。

在执行读操作的过程中，被选中的 Flash 字节的内容在 Capture-DR 状态被捕获到 Flash 数据字节寄存器中。对于每一个 Capture-DR 状态 AVR 自动在读低字节和读字节之间进行切换。对于进入 PROG_PAGEREAD 命令之后的第一个 Capture-DR 状态， AVR 首先读低字节。程序计数器在读每个高字节之后加 1，包括第一个字节。这样可以保证正确获取由 PROG_COMMANDS 设置好地址的第一个字节的内容，并保证读此页最后一个字节时程序计数器进入下一页。

Figure 133. Flash 数据字节寄存器



状态机控制 Flash 数据字节寄存器由 TCK 时钟驱动。在正常工作过程中，一个 Flash 字节需要移动 8 位。足够的时钟脉冲将通过 TAP 控制器驱动状态机完成这个操作。这个过程对用户是透明的。如果在进行页加载时 Update-DR 状态之间移动的位数太少，TAP 控制器应该保持 Run-Test/Idle 状态几个 TCK 时钟周期，确保 Update-DR 状态间隔至少有 11 个 TCK 周期。

编程算法

所有类似 “1a”，“1b” 这样的参考请参见 Table 131。

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 0b1010_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 0b0000_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

实施芯片擦除

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 P267 Table 128）

编程 Flash

在对 Flash 进行编程前必须先执行芯片擦除操作，见 P283 “ 实施芯片擦除 ” 。

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 加载地址位字节
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 P267 Table 128)
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 P260 Table 125) 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。通过 Update-DR 将 Flash 数据字节寄存器的内容装入 Flash 页存储单元，在对新的程序字进行操作之前程序计数器自动加 1。
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 P267 Table 128)
9. 重复步骤 3 到 8 ，直到所有的数据都被编程

读 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 P260 Table 125) 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。 Capture-DR 状态用来捕捉来自 Flash 的数据，并在每次读到一个字时将程序计数器加 1。 Capture-DR 状态发生于 shift-DR 状态之前。因此，第一个移出的字节是有效的数据。
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

编程 EEPROM

在对 EEPROM 进行编程之前首先要先进行芯片擦除操作，见 P283 “ 实施芯片擦除 ” 。

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}(参考 P267 Table 128)
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 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 Fuse 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P267 Table 128)。
6. 使用编程指令 6e 加载数据低字节，写 0 表示熔丝位编程，写 1 表示未编程。
7. 使用编程指令 6f 写熔丝位低字节
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P267 Table 128)。

编程锁定位

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P267 Table 128)。

读 熔丝位及锁定位

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

读取校验字节

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 to +125°C
存储温度	-65°C to +150°C
各个引脚对地的电压，除了 $\overline{\text{RESET}}$	-0.5V to V_{CC}+0.5V
$\overline{\text{RESET}}$ 引脚对地的电压	-0.5V to +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} = 1.8V-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，除了 XTAL1 引脚	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	-0.5 -0.5		$0.2V_{CC}$(1) $0.3V_{CC}$(1)	V
V_{IL1}	输入低电压， XTAL1 引脚	V_{CC} = 1.8V - 5.5V	-0.5		$0.1V_{CC}$(1)	V
V_{IH}	输入高电压，除了 XTAL1 及 RESET 引脚	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	$0.7V_{CC}$(2) $0.6V_{CC}$(2)		V_{CC} + 0.5 V_{CC} + 0.5	V
V_{IH1}	输入高电压， XTAL1 引脚	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	$0.8V_{CC}$(2) $0.7V_{CC}$(2)		V_{CC} + 0.5 V_{CC} + 0.5	V
V_{IH2}	输入高电压， RESET 引脚	V_{CC} = 1.8V - 5.5V	$0.9V_{CC}$(2)		V_{CC} + 0.5	V
V_{OL}	输出低电压 (3)，端口 A, C, D, E, F, G	I_{OL} = 10mA, V_{CC} = 5V I_{OL} = 5mA, V_{CC} = 3V			0.7 0.5	V
V_{OL1}	输出低电压 (3)，端口 B	I_{OL} = 20mA, V_{CC} = 5V I_{OL} = 10mA, V_{CC} = 3V			0.7 0.5	V
V_{OH}	输出高电压 (4)，端口 A, C, D, E, F, G	I_{OH} = -10mA, V_{CC} = 5V I_{OH} = -5mA, V_{CC} = 3V	4.2 2.3			V
V_{OH1}	输出高电压 (4)，端口 B	I_{OH} = -20mA, V_{CC} = 5V I_{OH} = -10mA, V_{CC} = 3V	4.2 2.3			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}	I 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} = 1.8V-5.5V (除非另外说明) (Continued)

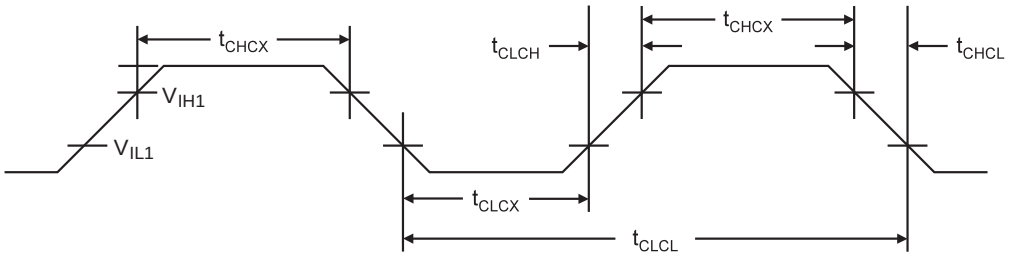
符号	参数	条件	最小值	典型值	最大值	单位
I_{CC}	电源电流	工作于 1MHz, V_{CC} = 2V (ATmega169V)			0.55	mA
		工作于 4MHz, V_{CC} = 3V (ATmega169L)			3.5	mA
		工作于 8MHz, V_{CC} = 5V (ATmega169)			12	mA
		空闲， 1MHz, V_{CC} = 2V (ATmega169V)		0.25	0.5	mA
		空闲， 4MHz, V_{CC} = 3V (ATmega169L)			1.5	mA
		空闲， 8MHz, V_{CC} = 5V (ATmega169)			5.5	mA
	掉电模式	WDT 使能， V_{CC} = 3V		<8	10	µA
		WDT 使能， V_{CC} = 3V		<1	2	µA
V_{ACIO}	模拟比较器输入偏置电压	V_{CC} = 5V V_{in} = V_{CC}/2		<10	40	mV
I_{ACLK}	模拟比较器输入漏电流	V_{CC} = 5V V_{in} = V_{CC}/2	-50		50	nA
t_{ACID}	模拟比较器传输延时	V_{CC} = 2.7V V_{CC} = 4.0V		750 500		ns

Note: 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 ； 10 mA , V_{CC} = 3V ； 10 mA， V_{CC} = 5V ；其他端口 5 mA ， V_{CC} = 3V) ，但是仍需要遵循以下要求：
   TQFP 及 MLF 封装：
   1] 所有端口的 IOL 总和不能超过 400 mA
   2] 端口 A0 - A7, C4 - C7, G2 的 IOL 总和不能超过 100 mA
   3] 端口 B0 - B7, E0 - E7, G3 - G5 的 IOL 总和不能超过 100 mA
   4] 端口 D0 - D7, C0 - C3, G0 - G1 的 IOL 总和不能超过 100 mA
   5] 端口 F0 - F7 的 IOL 总和不能超过 100 mA
   如果 IOL 超过了测试条件， VOL 可能超过相关指标。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 ； 10 mA , V_{CC} = 3V ； 10 mA， V_{CC} = 5V ；其他端口 5 mA ， V_{CC} = 3V) ，但是需要遵循以下要求：
   TQFP 及 MLF 封装：
   1] 所有端口的 IOL 总和不能超过 400 mA
   2] 端口 A0 - A7, C4 - C7, G2 的 IOL 总和不能超过 100 mA
   3] 端口 B0 - B7, E0 - E7, G3 - G5 的 IOL 总和不能超过 100 mA
   4] 端口 D0 - D7, C0 - C3, G0 - G1 的 IOL 总和不能超过 100 mA
   5] 端口 F0 - F7 的 IOL 不能超过 100 mA
   如果 IOH 超过了测试条件， VOH 可能超过相关指标。不保证引脚可以输出比列于此处的测试条件更大的电流。

外部时钟波形

Figure 134. 外部时钟波形



外部时钟

Table 132. 外部时钟

符号	参数	V_{CC}=1.8-5.5V		V_{CC}=2.7-5.5V		V_{CC}=4.5-5.5V		单位
		最小值	最大值	最小值	最大值	最小值	最大值	
$1/t_{CLCL}$	振荡器频率	0	1	0	8	0	16	MHz
t_{CLCL}	时钟周期	1000		125		62.5		ns
t_{CHCX}	高电平时间	400		50		25		ns
t_{CLCX}	低电平时间	400		50		25		ns
t_{CLCH}	上升时间		2.0		1.6		0.5	µs
t_{CHCL}	下降时间		2.0		1.6		0.5	µs
Δt_{CLCL}	时钟周期的变化		2		2		2	%

SPI 时序特性

细节见 Figure 135 及 Figure 136。

Table 133. SPI 时序参数

	说明	模式	最小值	典型值	最大值	
1	SCK 周期	主机		见表 6 9		
2	SCK 高 / 低电平	主机		50% 占空比		
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 (1)	从机	$2 \cdot t_{ck}$			
12	上升 / 下降时间	从机		1.6		µs

Table 133. SPI 时序参数

	说明	模式	最小值	典型值	最大值	
13	设置时间	从机	10			ns
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			
17	$\overline{SS}$ 高到三态	从机		10		
18	$\overline{SS}$ 低到 SCK	从机	20 • t_{ck}			

Note: 1. 在 SPI 编程模式，最小的 SCK 高 / 低电平时间为：
f_{CK} < 12 MHz 下 - 2 t_{CLCL}
f_{CK} > 12 MHz 下 - 3 t_{CLCL}

Figure 135. SPI 接口时序要求 (主机模式)

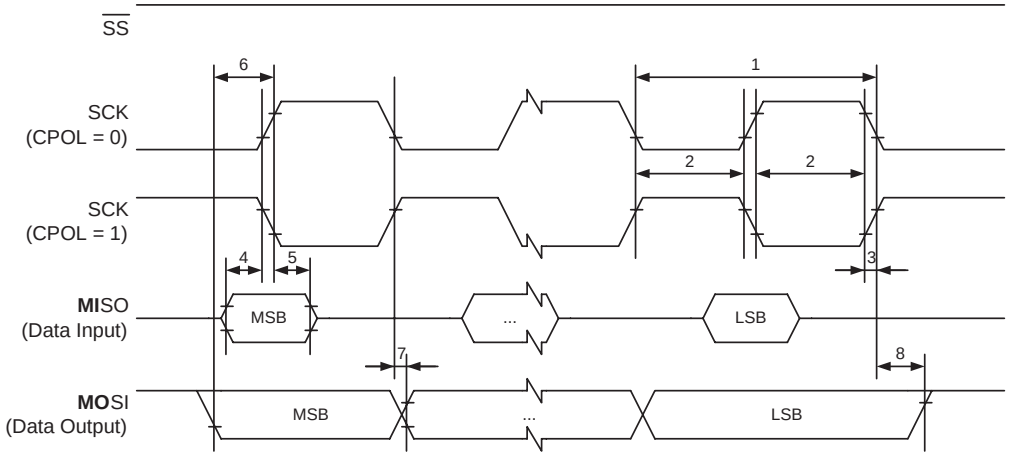
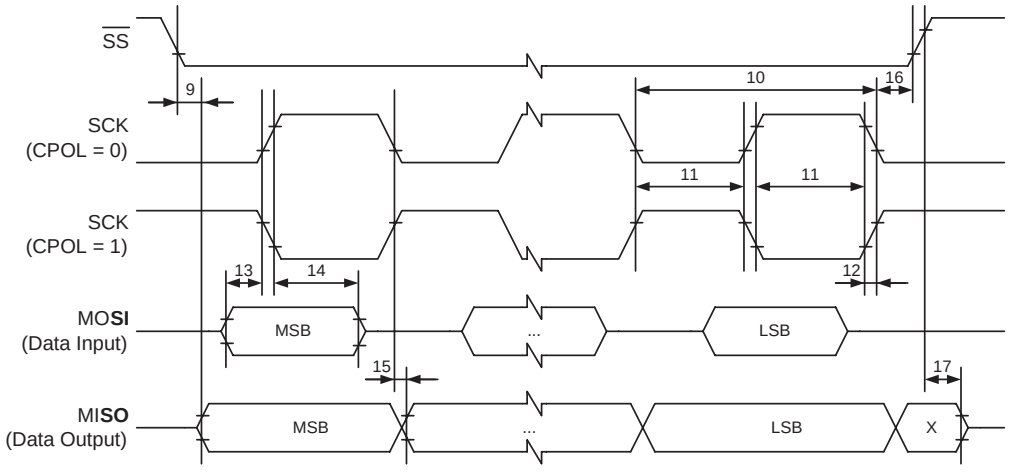


Figure 136. SPI 接口时序模式 (从机模式)



ADC 特性—初始参数

Table 134. ADC 特性参数

符号	参数	条件	最小值	典型值	最大值	单位
	分辨率	单端转换模式		10		Bits
		差分转换模式		8		Bits
	绝对精度 (包括 INL、 DNL、量化误差、增益及偏置误差)	单端转换模式 V_{REF} = 4V, V_{CC} = 4V, ADC 时钟 = 200 kHz		2	2.5	LSB
		单端转换模式 V_{REF} = 4V, V_{CC} = 4V, ADC 时钟 = 1 MHz		4.5		LSB
		单端转换模式 V_{REF} = 4V, V_{CC} = 4V, ADC 时钟 = 200 kHz 降噪模式		2		LSB
		单端转换模式 V_{REF} = 4V, V_{CC} = 4V, ADC 时钟 = 1 MHz 降噪模式		4.5		LSB
	整体非线性 (INL)	单端转换模式 V_{REF} = 4V, V_{CC} = 4V, ADC 时钟 = 200 kHz		0.5		LSB
	差分非线性 (DNL)	单端转换模式 V_{REF} = 4V, V_{CC} = 4V, ADC 时钟 = 200 kHz		0.25		LSB
	增益误差	单端模式 V_{REF} = 4V, V_{CC} = 4V, ADC 时钟 = 200 kHz		2		LSB
	偏移误差	单端模式 V_{REF} = 4V, V_{CC} = 4V, ADC 时钟 = 200 kHz		2		LSB
	转换时间	连续转换	13		260	µs
	时钟频率	单端模式	50		1000	kHz
AVCC	模拟电压		V_{CC} - 0.3		V_{CC} + 0.3	V
V_{REF}	参考电压	单端转换	1.0		AVCC	V
		差动转换	1.0		AVCC - 0.5	V
V_{IN}	输入电压	单端转换	GND		V_{REF}	V
		差动转换	0		AVCC(1)	V
	输入带宽	单端转换		38,5		kHz
		差动通道		4		kHz
V_{INT}	片内参考电压		1.0	1.1	1.2	V
R_{REF}	参考电压输入阻抗			32		kΩ
R_{AIN}	模拟输入阻抗			100		MΩ

Note: 1. V_{DIFF} 必须低于 V_{REF}

LCD 控制器特性参数—初始数据

Table 135. LCD 控制器特性参数

符号	参数	条件	最小值	典型值	最大值	单位
I_{LCD}	LCD 驱动器电流	所有的 COM 及 SEG 引脚		100		µA
R_{LCD}	LCD 驱动器输出阻抗	每一个 COM 或 SEG 引脚		10		kΩ

ATmega169 典型特性—初始数据

下面的图表给出了器件的典型性能。在产生过程中并不测试这些数据。全部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。

掉电模式下的功耗与选择的时钟无关。

耗电流与多个因素有关：工作电压、工作频率、 I/O 的负载、 I/O 引脚开关速率、执行的代码及环境温度。最主要的因素是工作电压和工作频率。

容性负载 I/O 的输出电流可通过公式 $C_L*V_{CC}*f$ 进行估算， C_L = 负载电容， V_{CC} = 工作电压， f = I/O 引脚平均开关频率。

器件的特性化是在比测试极限频率更高的频率进行的。但是不保证器件能够正常在高于订货信息表给出的工作频率。

看门狗使能的掉电模式和看门狗禁止的掉电模式之间的电流差值即为开关看门狗定时器所需的电流。

工作电流

Figure 137. 工作电流和工作频率 (0.1 - 1.0 MHz) 的关系

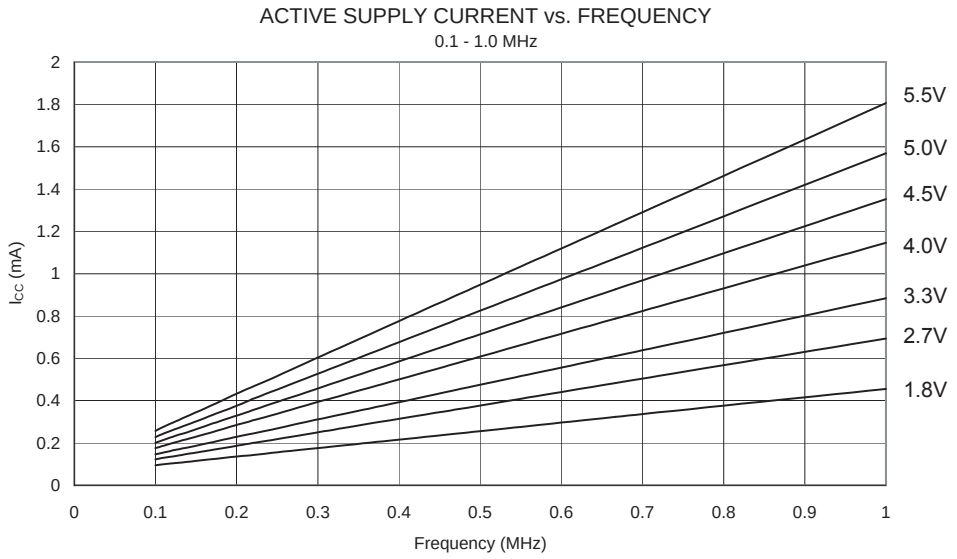


Figure 138. 工作电流和工作频率 (1 - 20 MHz) 的关系

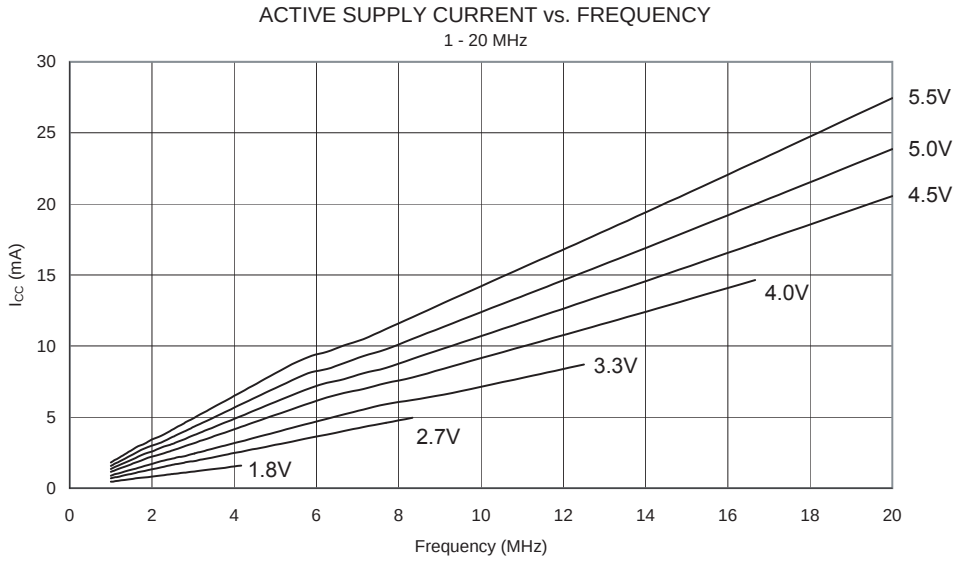


Figure 139. 工作电流和 V_{CC} 的关系 (内部 RC 振荡器，8 MHz)

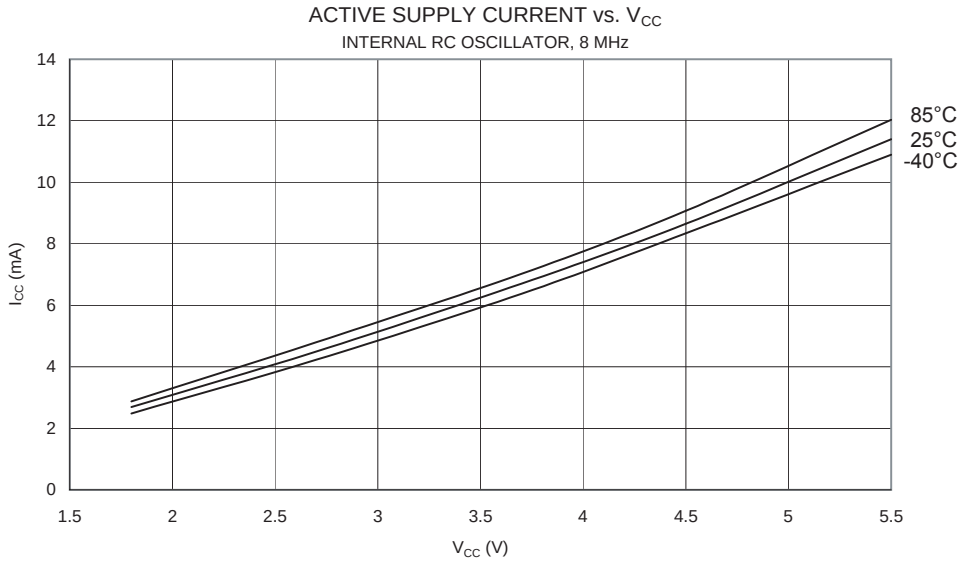


Figure 140. 工作电流和 V_{CC} 的关系 (内部 RC 振荡器 ， CKDIV8 编程， 1 MHz)

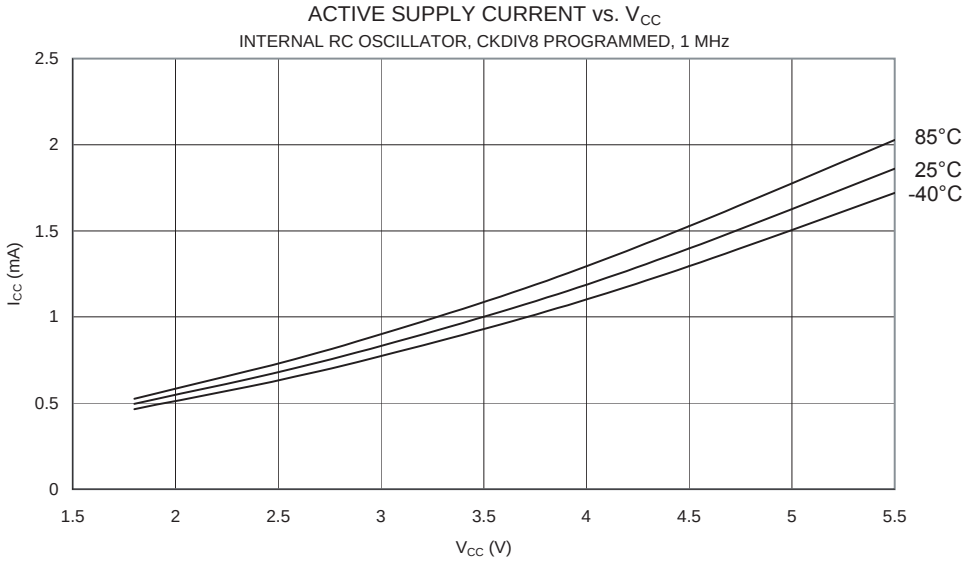
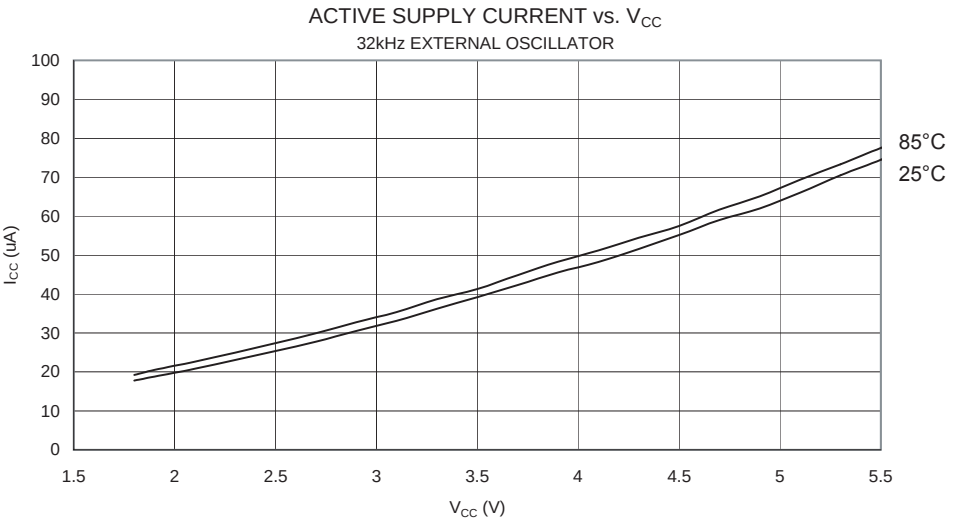


Figure 141. 工作电流和 V_{CC} 的关系 (32 kHz 外部晶振)



空闲模式电流

Figure 142. 空闲模式电流和工作频率 (0.1 - 1.0 MHz) 的关系

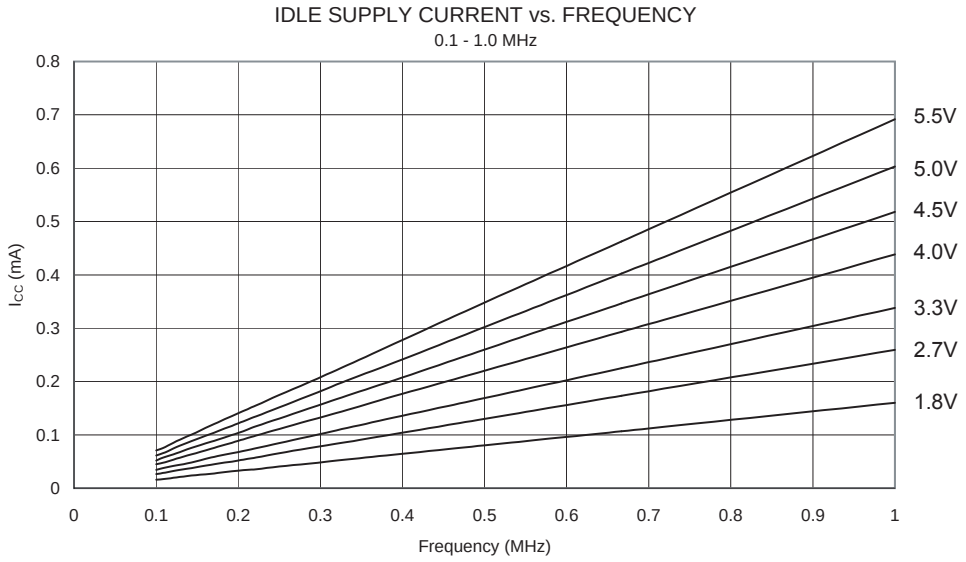


Figure 143. 空闲模式电流和工作频率 (1 - 20 MHz) 的关系

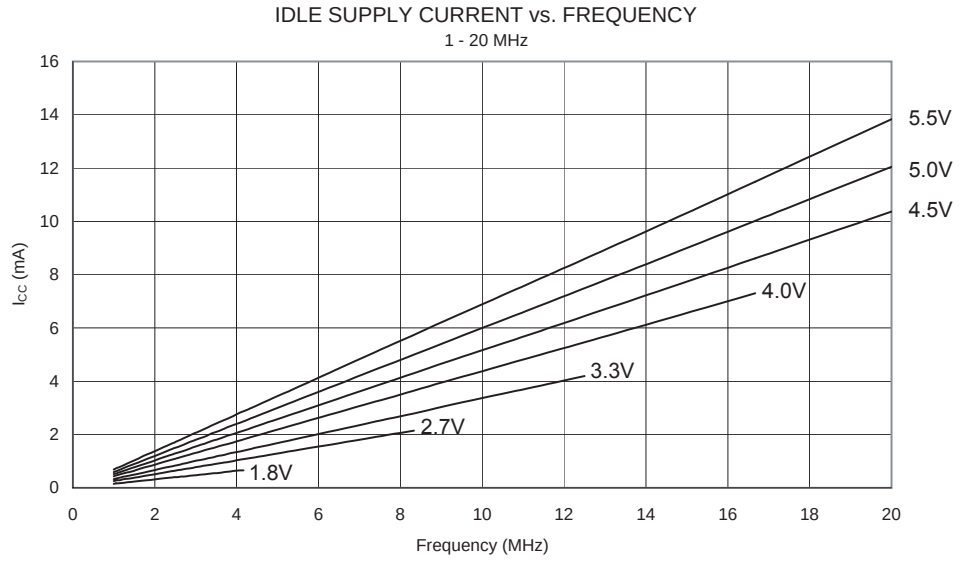


Figure 144. 空闲模式电流和 V_{CC} 的关系 (内部 RC 振荡器 ， 8 MHz)

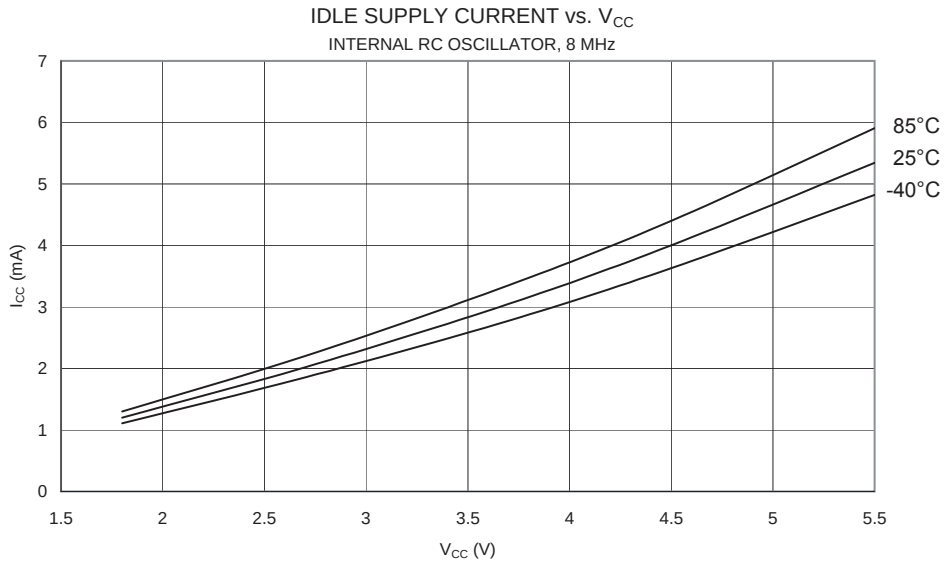


Figure 145. 空闲模式电流和 V_{CC} 的关系 (内部 RC 振荡器 ， CKDIV8 编程，1 MHz)

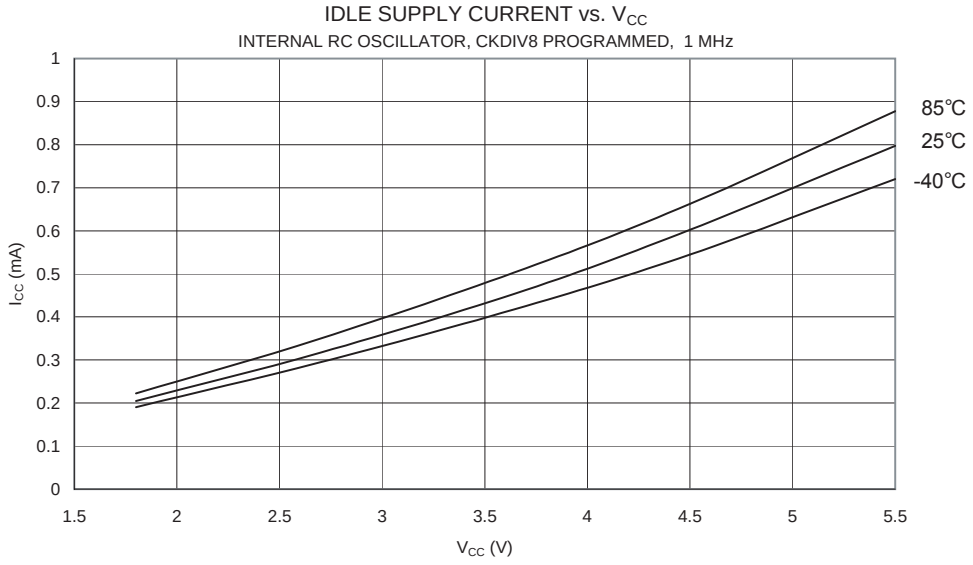
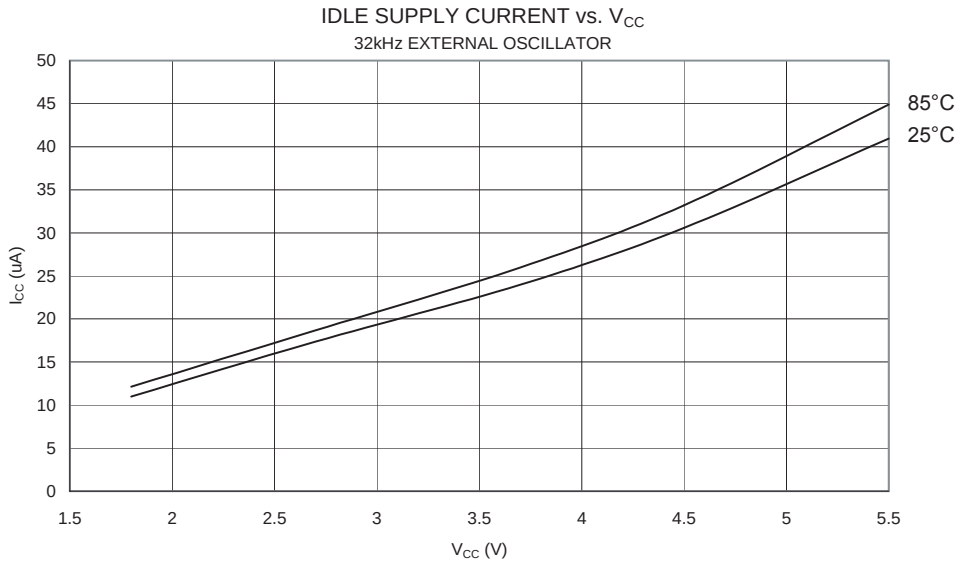


Figure 146. 空闲模式电流和 V_{CC} 的关系 (32 kHz 外部晶振)



掉电模式电流

Figure 147. 掉电模式电流和 V_{CC} 的关系 (看门狗定时器禁用)

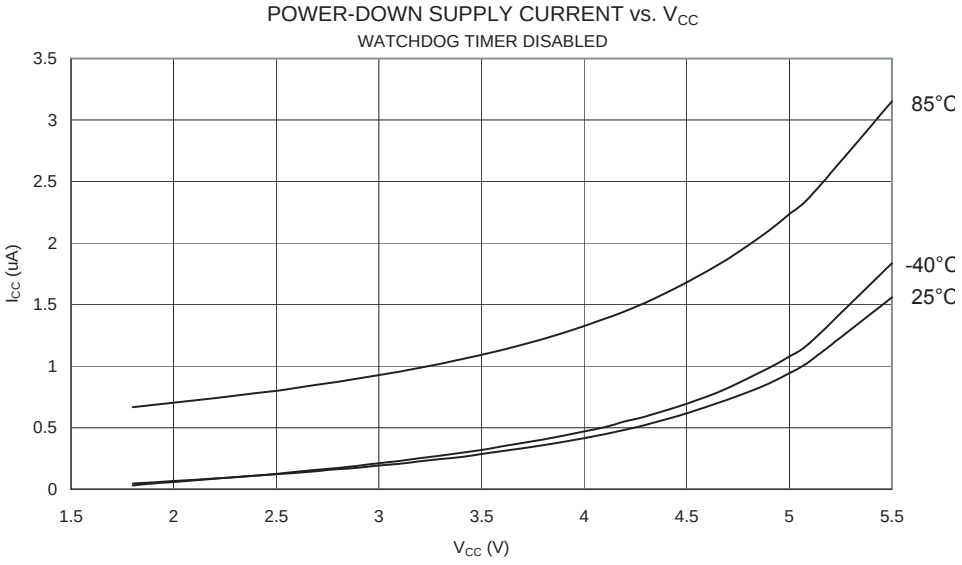
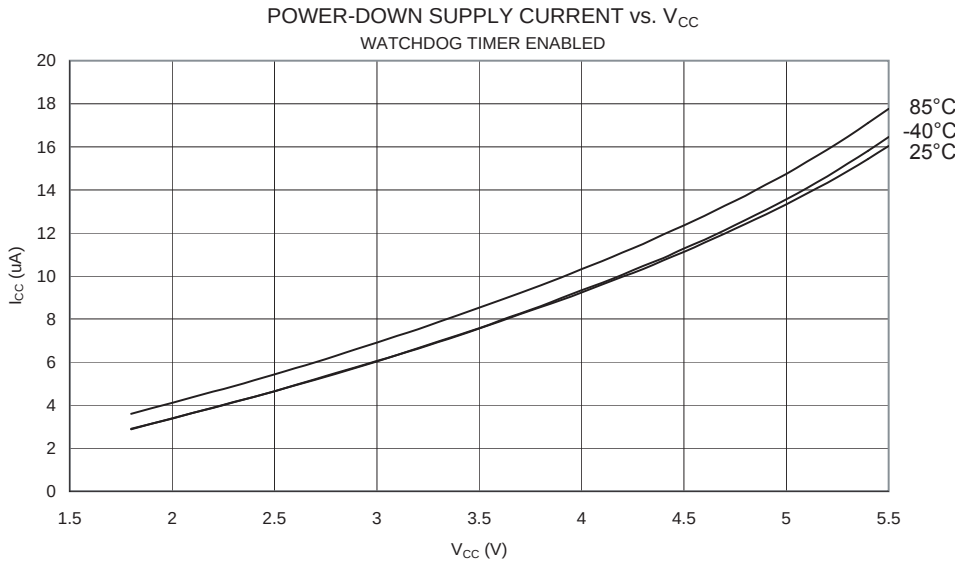
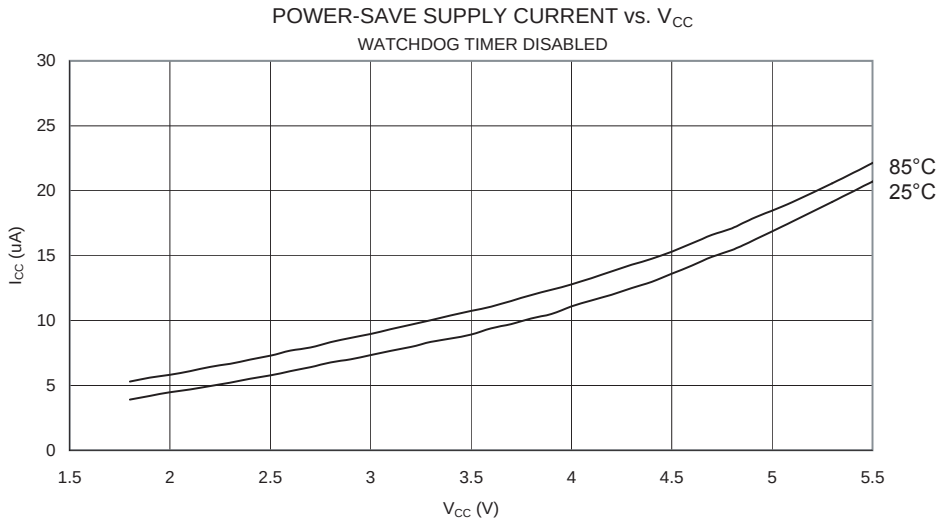


Figure 148. 掉电模式电流和 V_{CC} 的关系 (看门狗定时器使能)



省电模式电流

Figure 149. 省电模式电流和 V_{CC} 的关系 (看门狗定时器禁用)



Standby 模式电流

Figure 150. Standby 模式电流和 V_{CC} 的关系 (455 kHz 谐振器，看门狗定时器禁用)

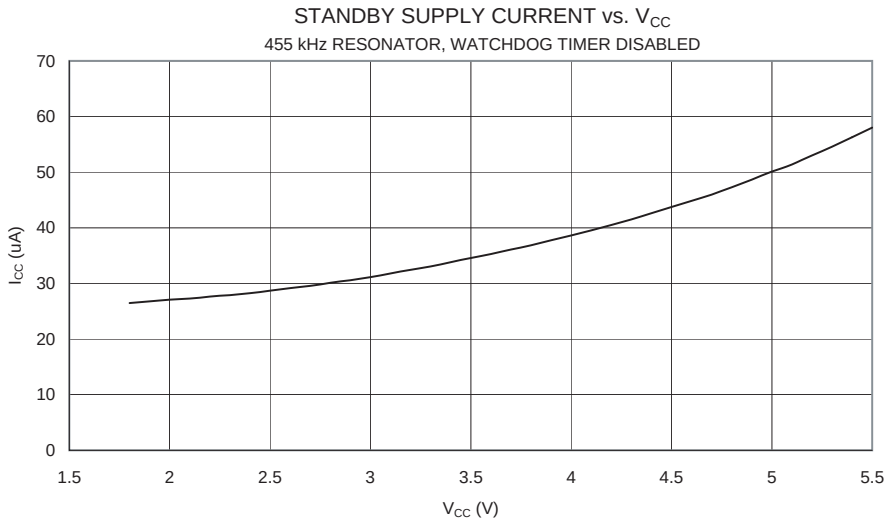


Figure 151. Standby 模式电流和 V_{CC} 的关系 (1 MHz 谐振器，看门狗定时器禁用)

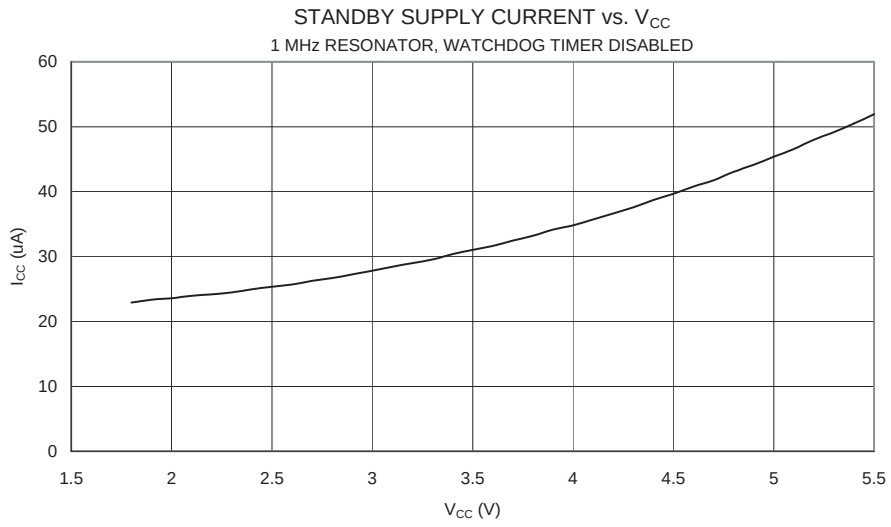


Figure 152. Standby 模式电流和 V_{CC} 的关系 (2 MHz 谐振器，看门狗定时器禁用)

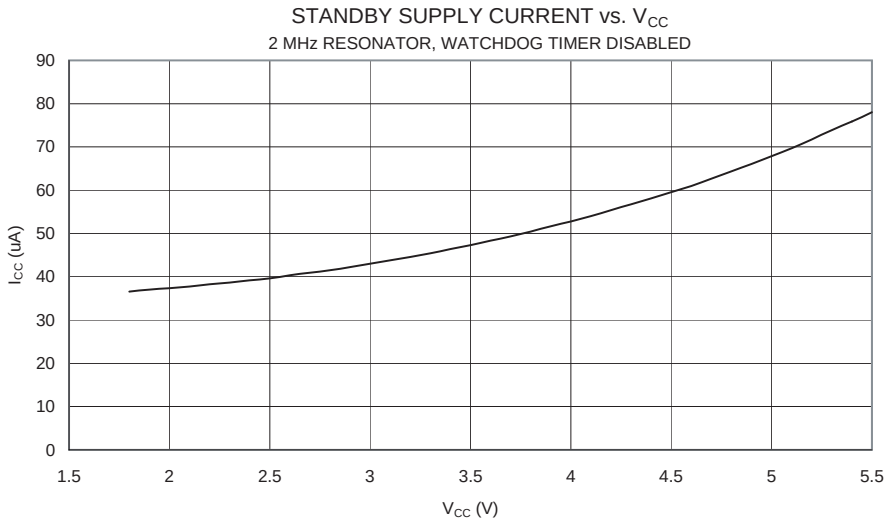


Figure 153. Standby 模式电流和 V_{CC} 的关系 (2 MHz Xtal，看门狗定时器禁用)

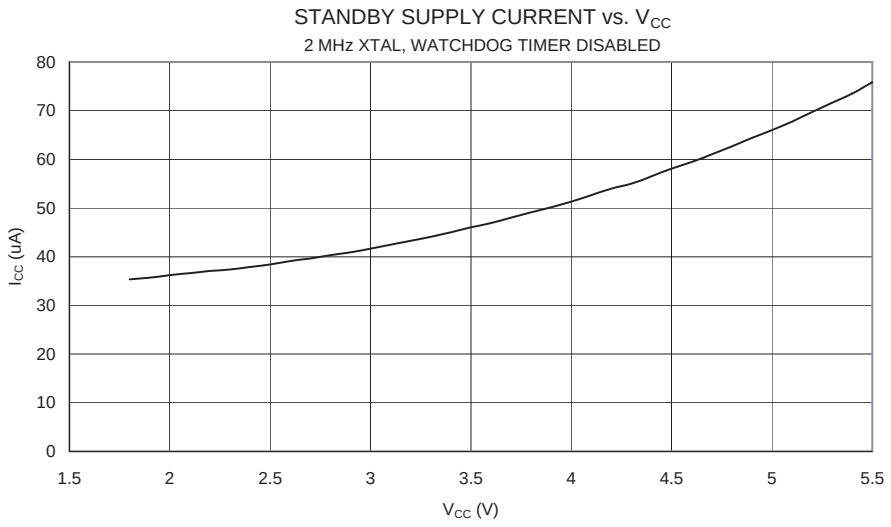


Figure 154. Standby 模式电流和 V_{CC} 的关系 (4 MHz 谐振器，看门狗定时器禁用)

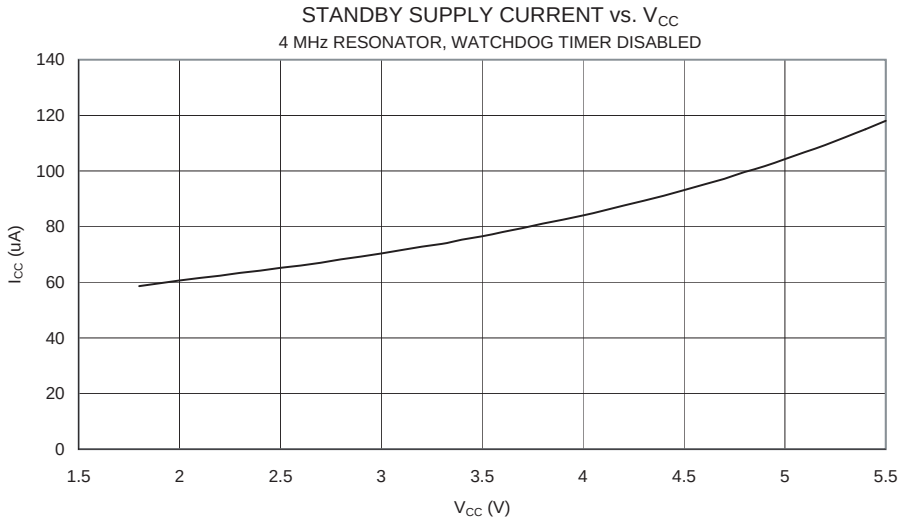


Figure 155. Standby 模式电流和 V_{CC} 的关系 (4 MHz Xtal，看门狗定时器禁用)

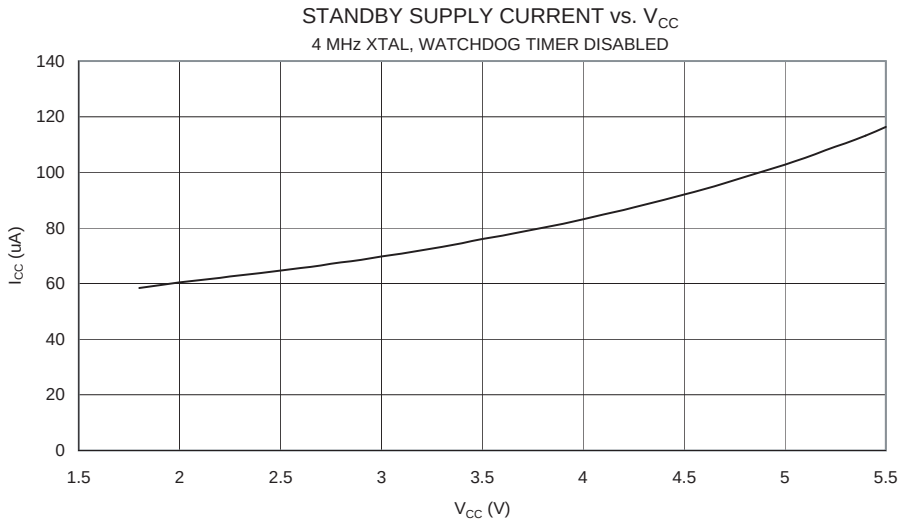


Figure 156. Standby 模式电流和 V_{CC} 的关系 (6 MHz 谐振器，看门狗定时器禁用)

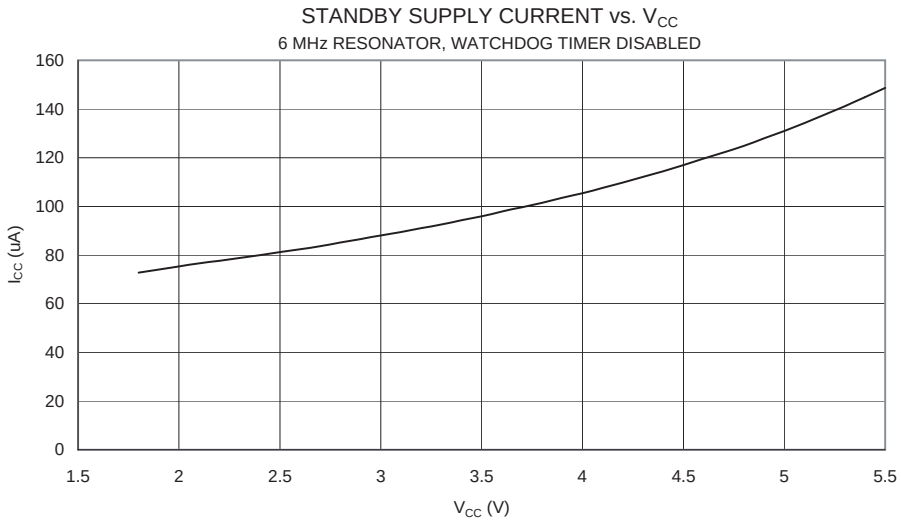
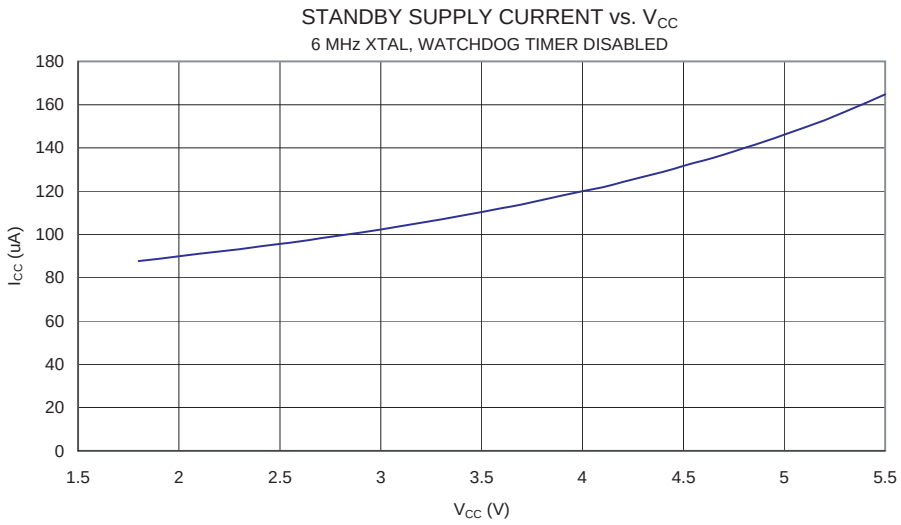


Figure 157. Standby 模式电流和 V_{CC} 的关系 (6 MHz 晶振，看门狗定时器禁用)



引脚上拉

Figure 158. I/O 引脚上拉电阻电流和输入电压的关系 (V_{CC} = 5V)

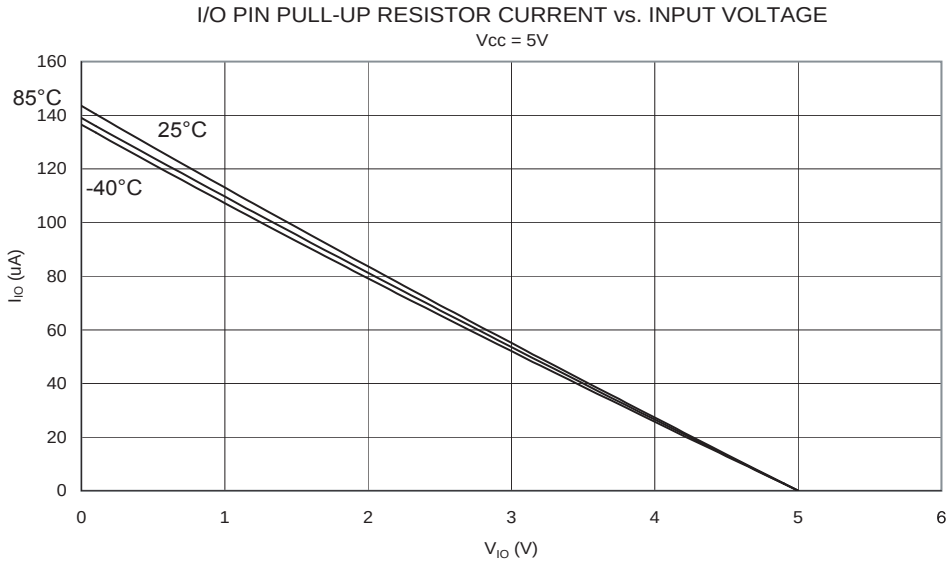


Figure 159. I/O 引脚上拉电阻电流和输入电压的关系 (V_{CC} = 2.7V)

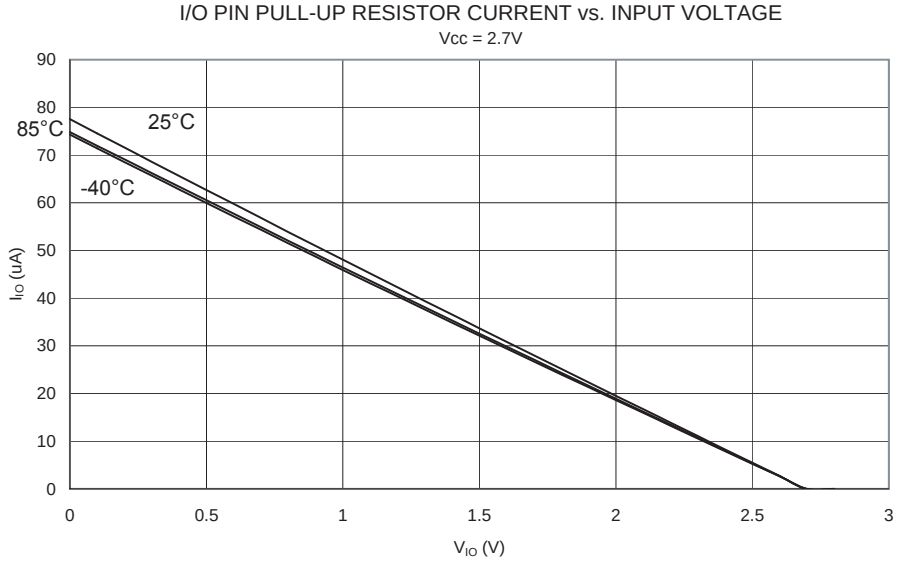


Figure 160. I/O 引脚上拉电阻电流和输入电压的关系 (V_{CC} = 1.8V)

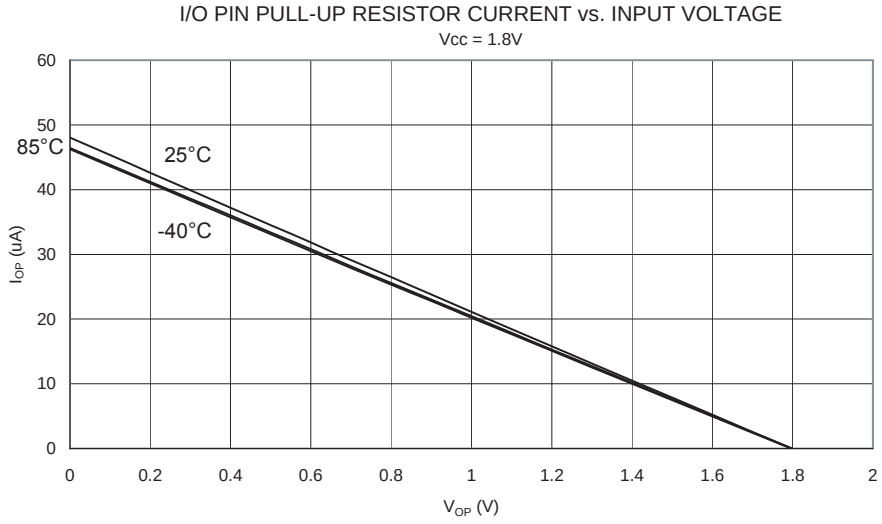


Figure 161. 复位 (Reset) 引脚上拉电阻电流和 Reset 引脚电压的关系 (V_{CC} = 5V)

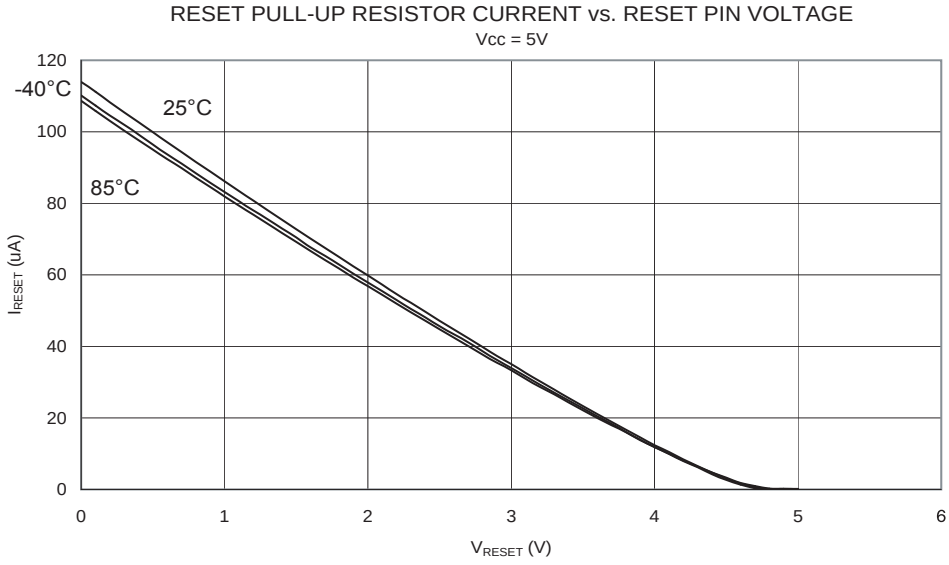


Figure 162. 复位 (Reset) 引脚上拉电阻电流和 Reset 引脚电压的关系 (V_{CC} = 2.7V)

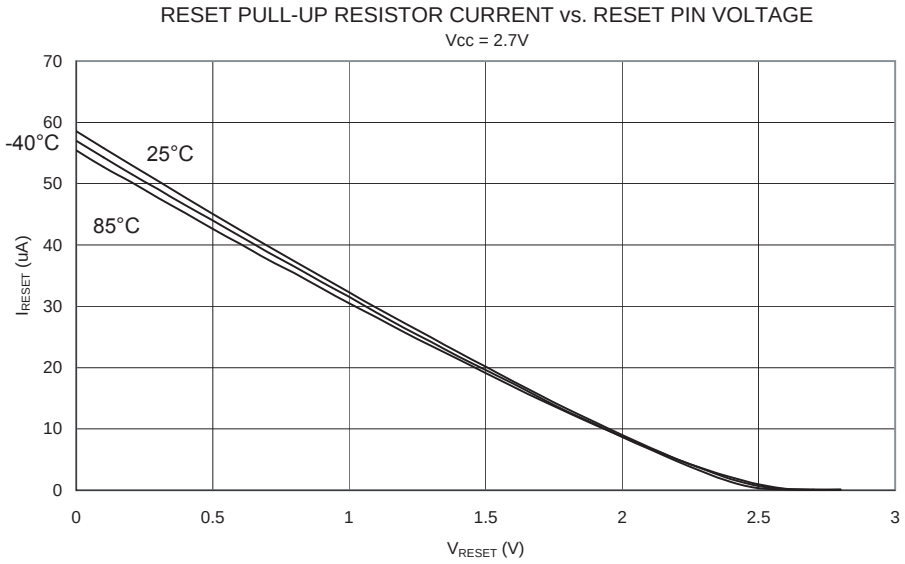
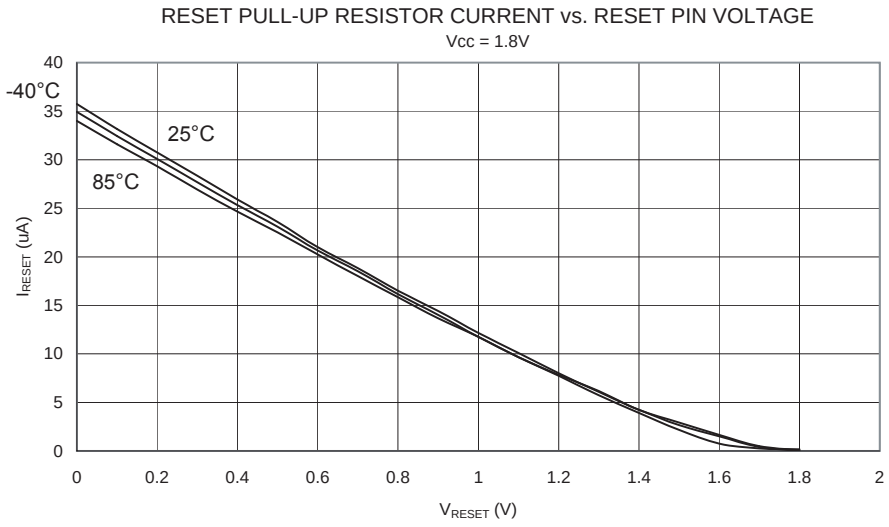


Figure 163. 复位 (Reset) 引脚上拉电阻电流和 Reset 引脚电压的关系 (V_{CC} = 1.8V)



驱动能力

Figure 164. I/O 引脚输出电流和输出电压的关系：端口 A, C, D, E, F, G (V_{CC} = 5V)

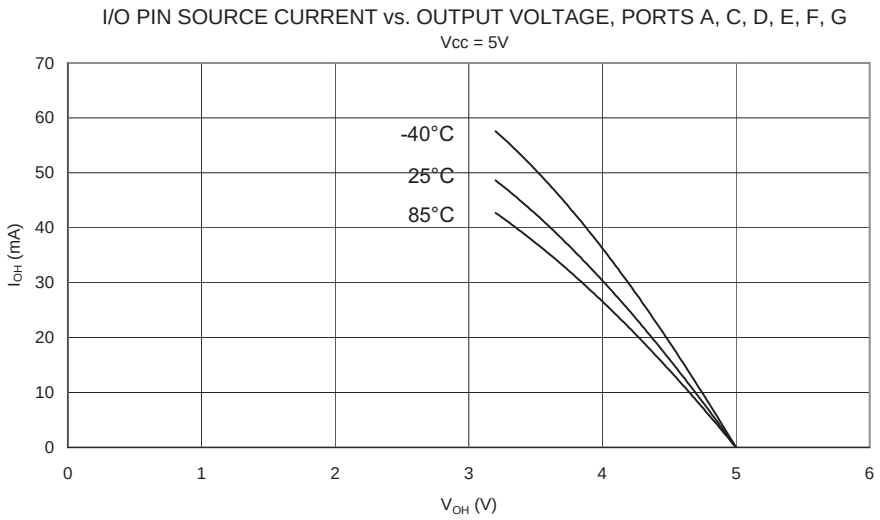


Figure 165. I/O 引脚输出电流和输出电压的关系：端口 A, C, D, E, F, G (V_{CC} = 2.7V)

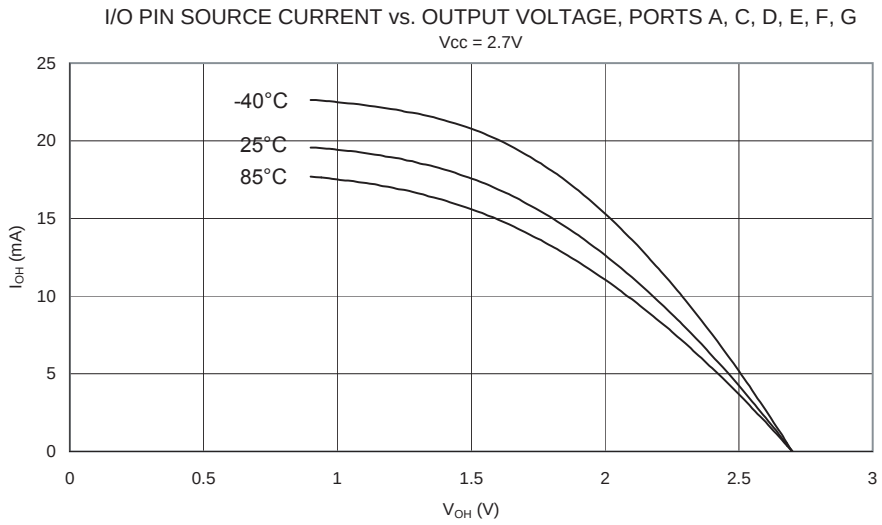


Figure 166. I/O 引脚输出电流和输出电压的关系：端口 A, C, D, E, F, G (V_{CC} = 1.8V)

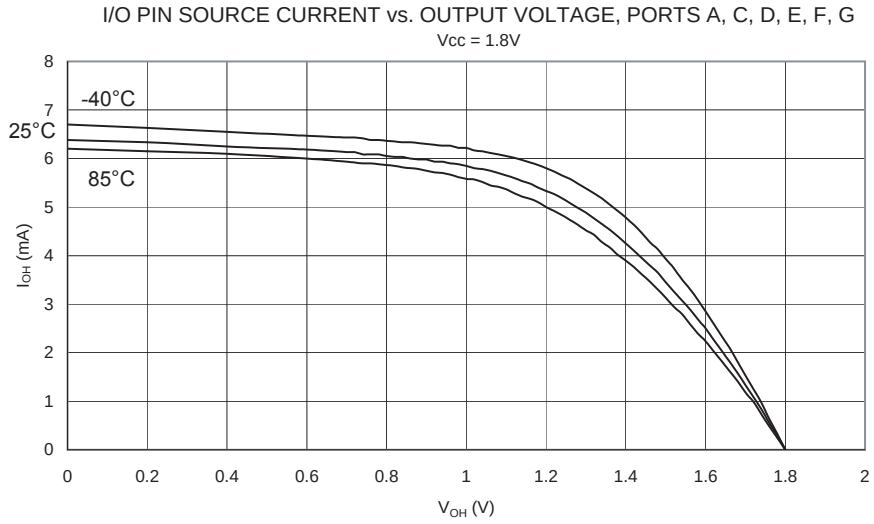


Figure 167. I/O 引脚输出电流和输出电压的关系：端口 B (V_{CC}= 5V)

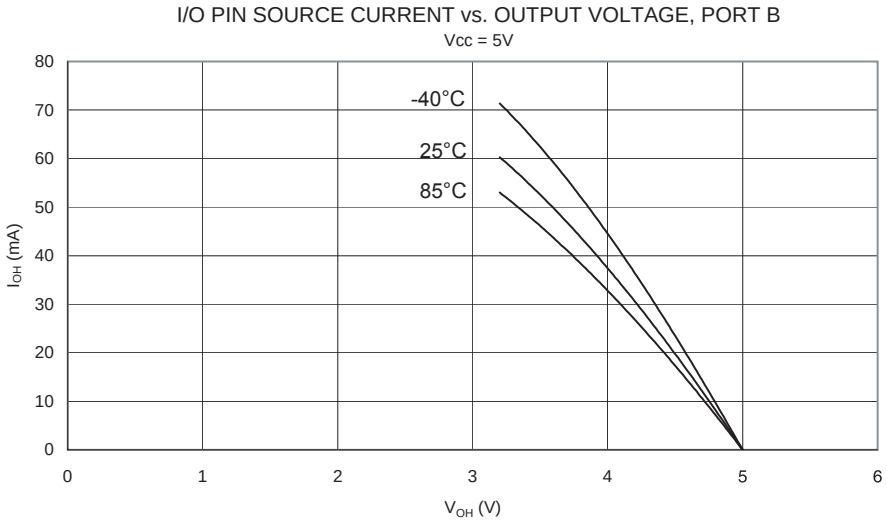


Figure 168. I/O 引脚输出电流和输出电压的关系：端口 B (V_{CC} = 2.7V)

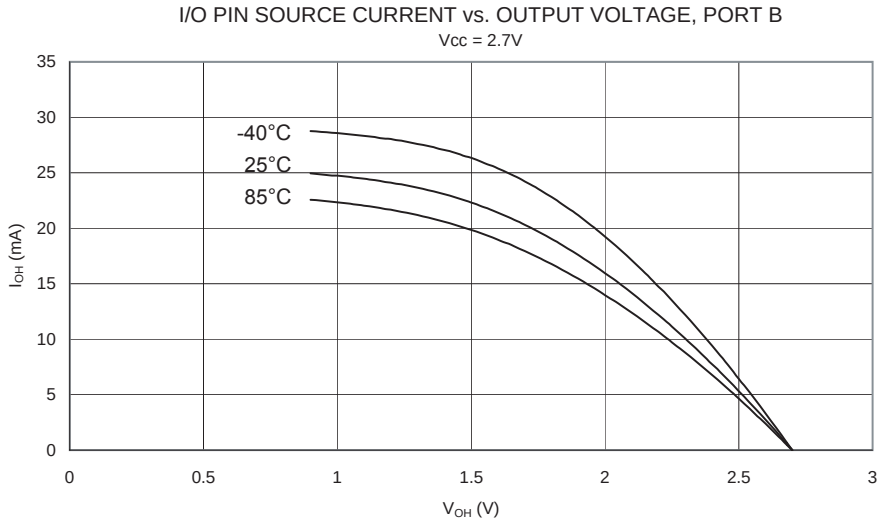


Figure 169. I/O 引脚输出电流和输出电压的关系：端口 B (V_{CC} = 1.8V)

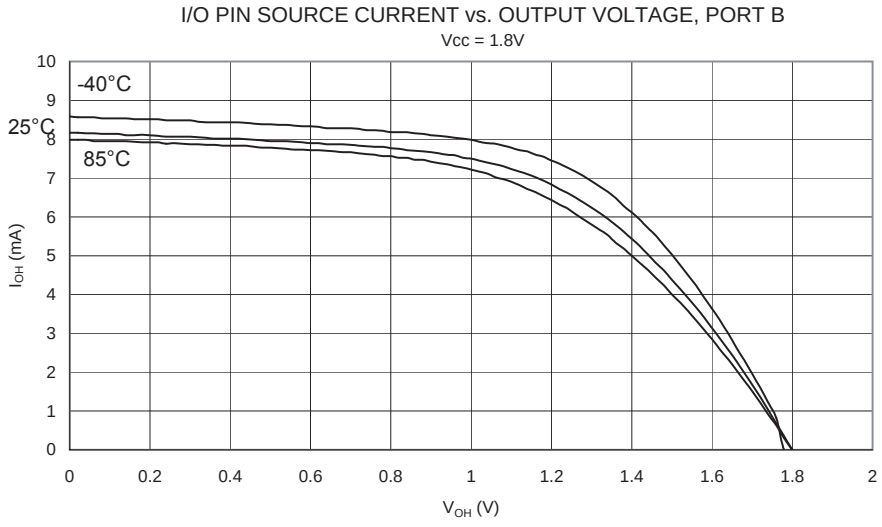


Figure 170. I/O 引脚吸收电流和输出电压的关系：端口 A, C, D, E, F, G (V_{CC} = 5V)

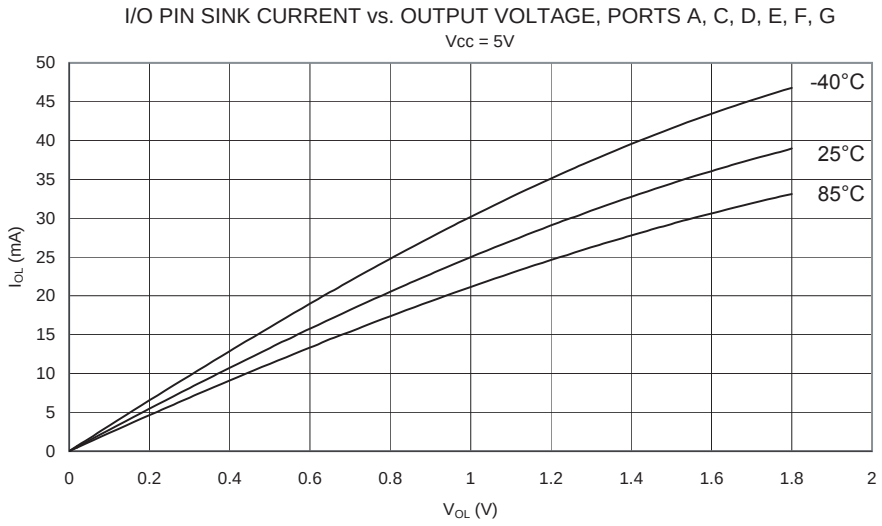


Figure 171. I/O 引脚吸收电流和输出电压的关系：端口 A, C, D, E, F, G (V_{CC} = 2.7V)

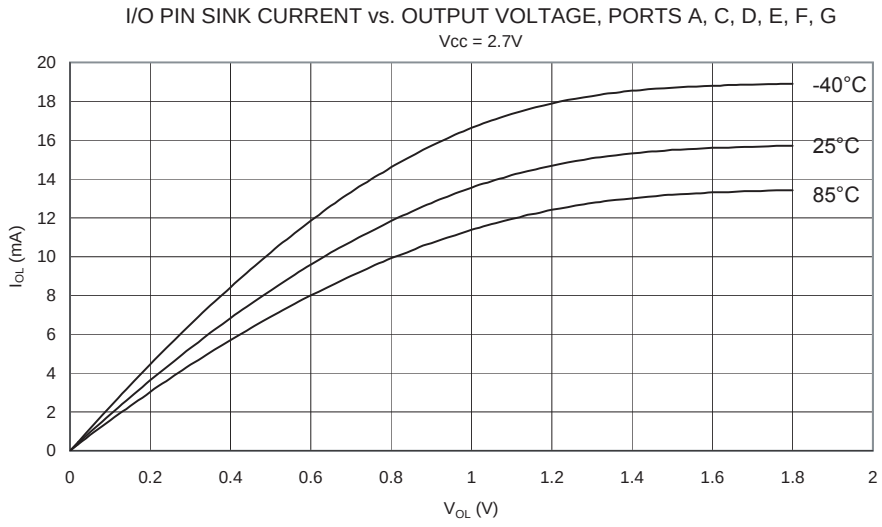


Figure 172. I/O 引脚吸收电流和输出电压的关系：端口 A, C, D, E, F, G (V_{CC} = 1.8V)

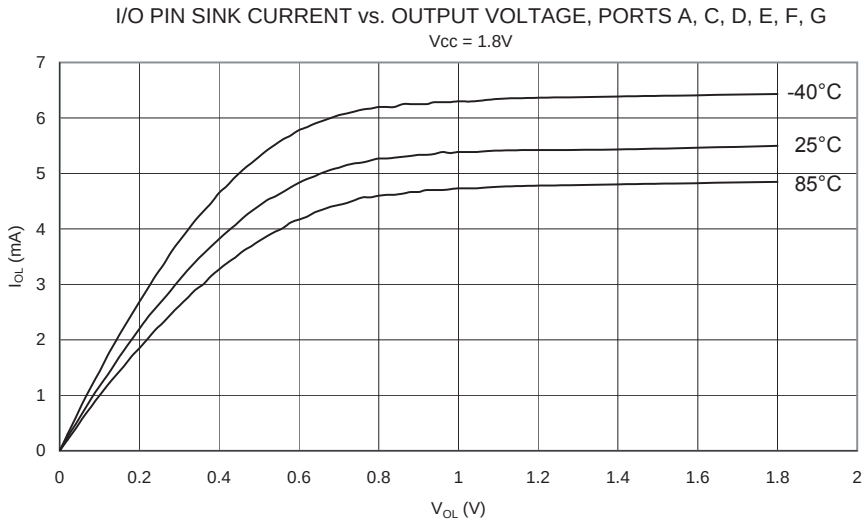


Figure 173. I/O 引脚吸收电流和输出电压的关系：端口 B (V_{CC} = 5V)

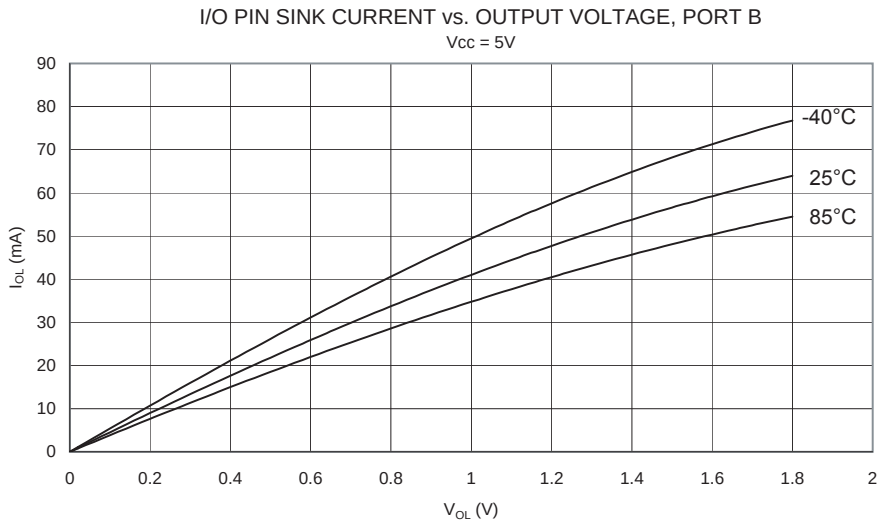


Figure 174. I/O 引脚吸收电流和输出电压的关系：端口 B (V_{CC} = 2.7V)

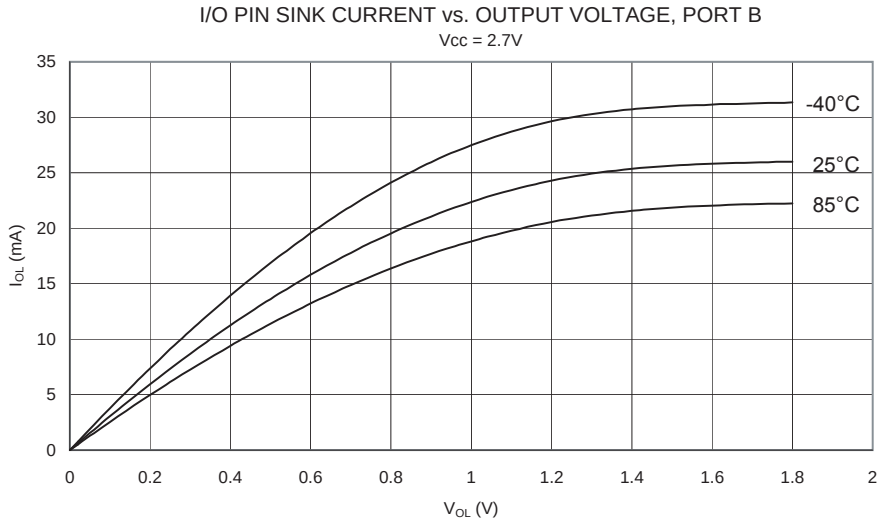
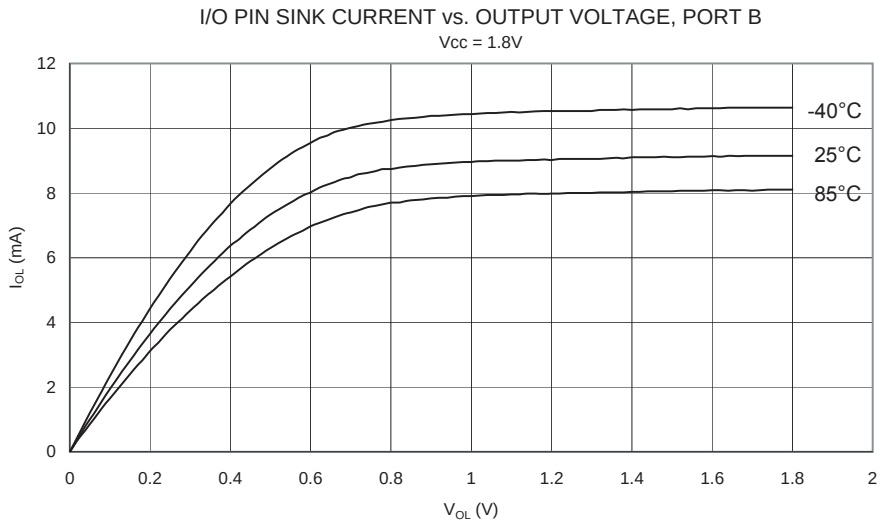


Figure 175. I/O 引脚吸收电流和输出电压的关系：端口 B (V_{CC} = 1.8V)



引脚门限及滞后

Figure 176. I/O 引脚输入门限电压和 V_{CC} 的关系 (V_{IH}, I/O 引脚读出值为 “1”)

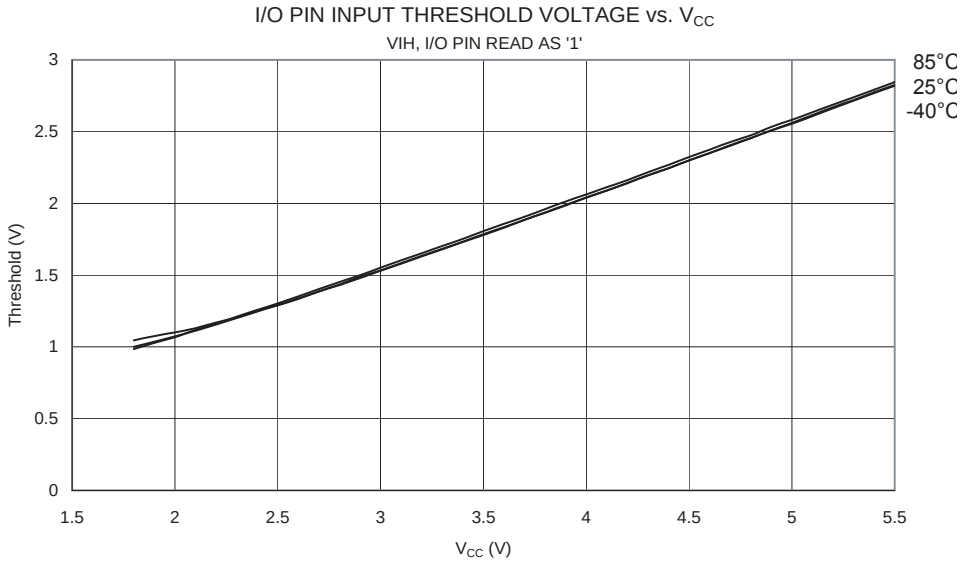


Figure 177. I/O 引脚输入门限电压和 V_{CC} 的关系 (V_{IL}, I/O 引脚读出值为 “0”)

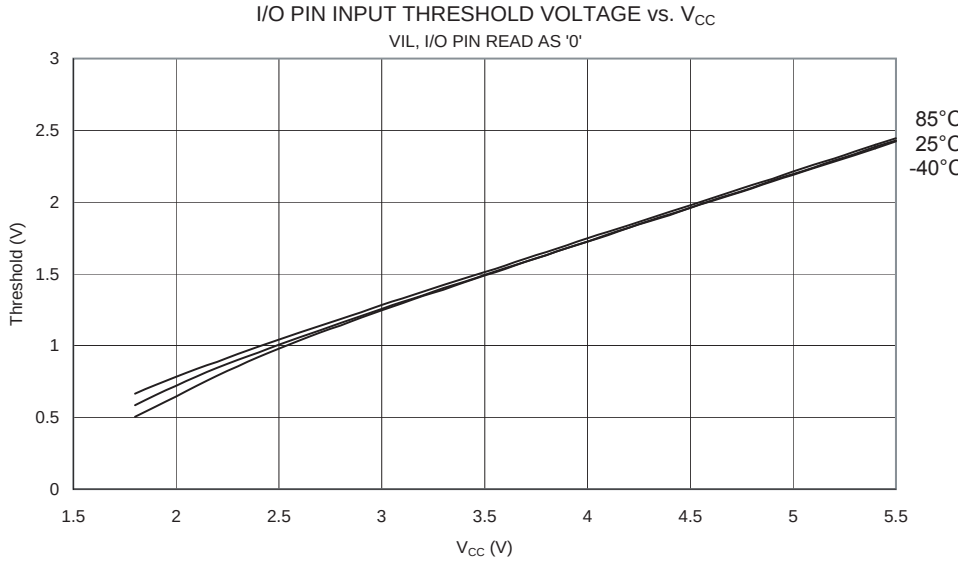


Figure 178. I/O 引脚输入迟滞和 V_{CC} 的关系

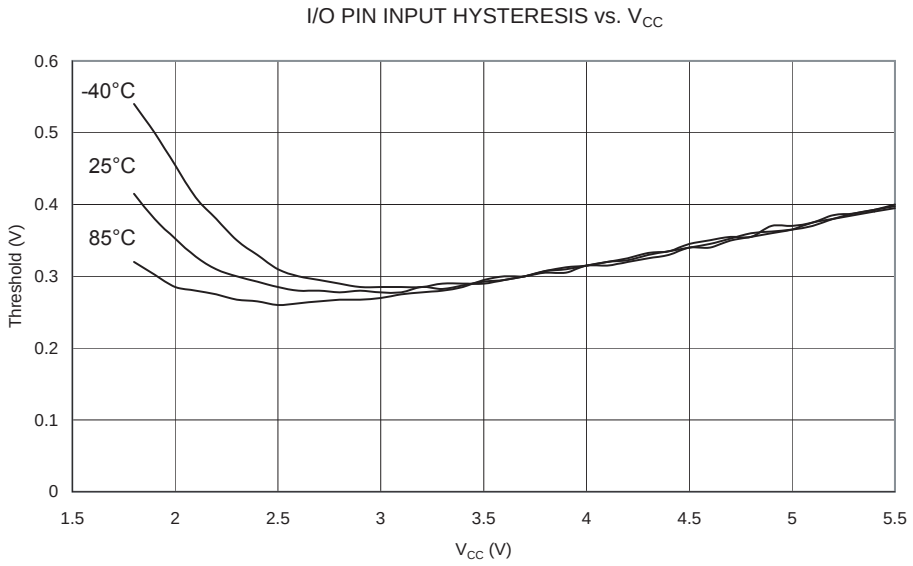


Figure 179. Reset 输入门限电压和 V_{CC} 的关系 (V_{IH},Reset 引脚读出值为 “1”)

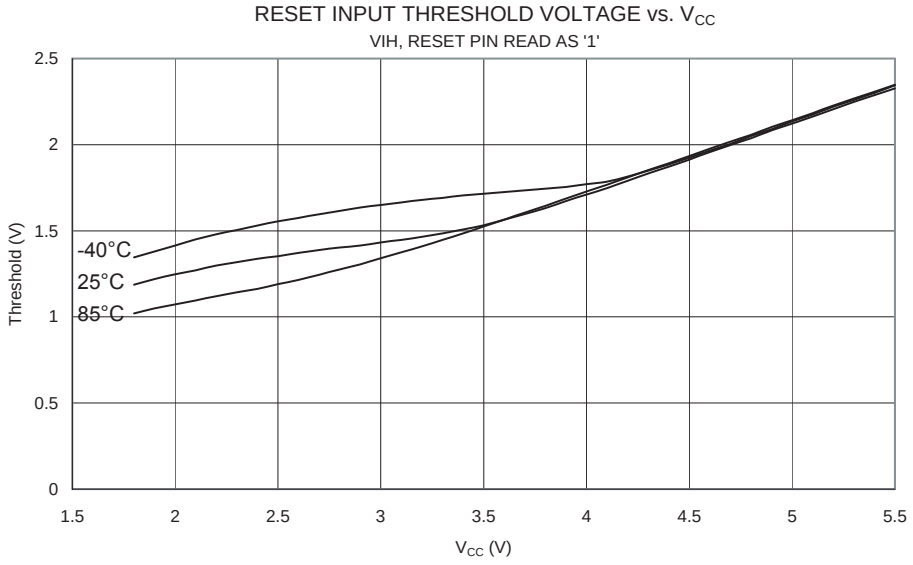


Figure 180. Reset 输入门限电压和 V_{CC} 的关系 (V_{IL},Reset 引脚读出值为 “0”)

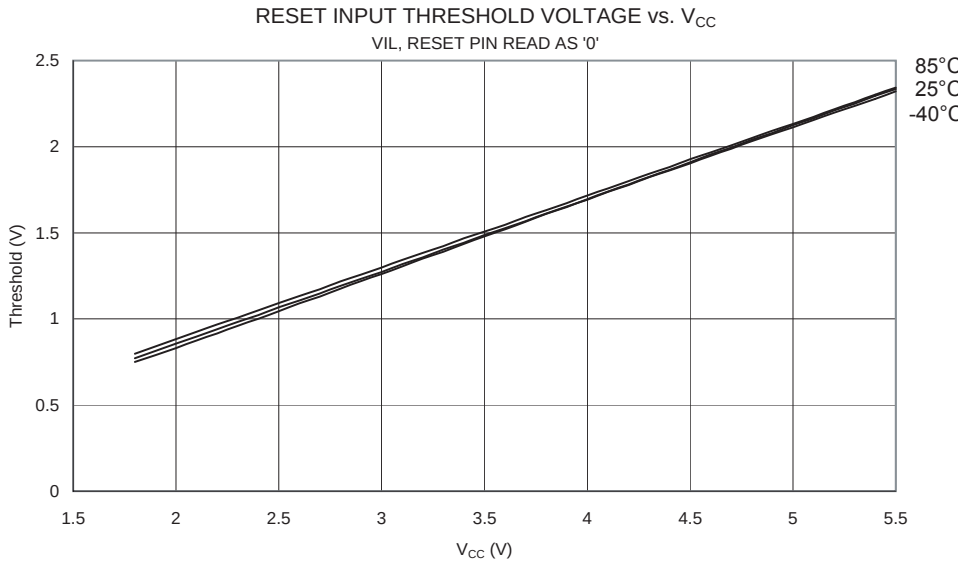
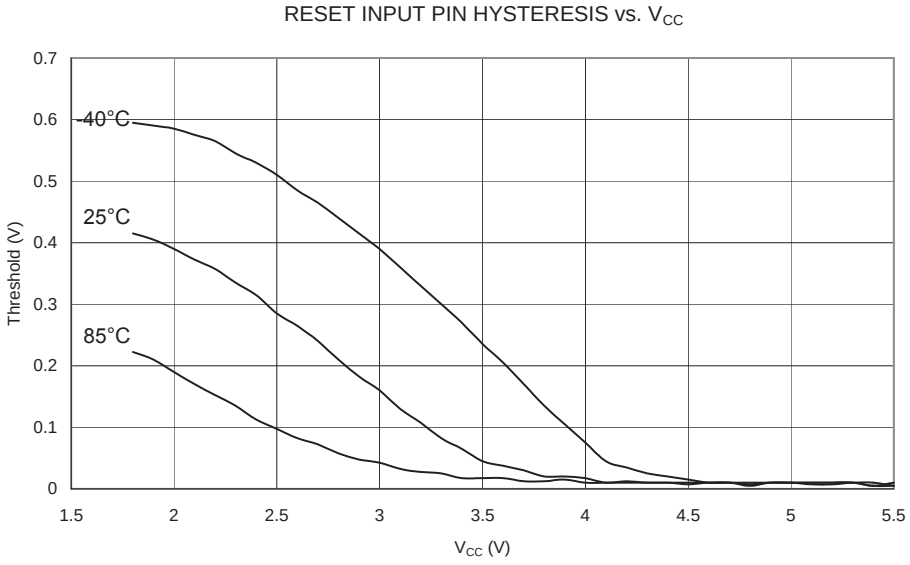


Figure 181. Reset 输入迟滞和 V_{CC} 的关系



BOD 门限值与模拟比较器偏移量

Figure 182. BOD 门限值和温度的关系 (BOD 电平为 4.3V)

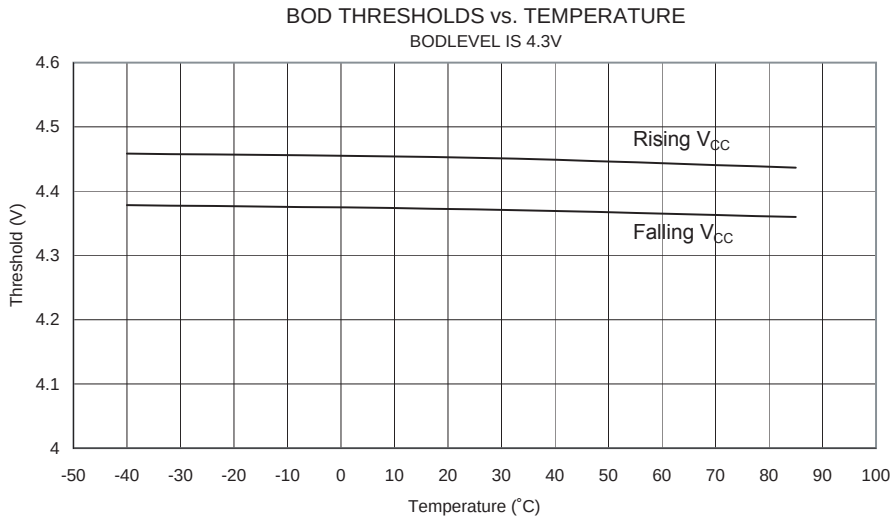


Figure 183. BOD 门限值和温度的关系 (BOD 电平为 2.7V)

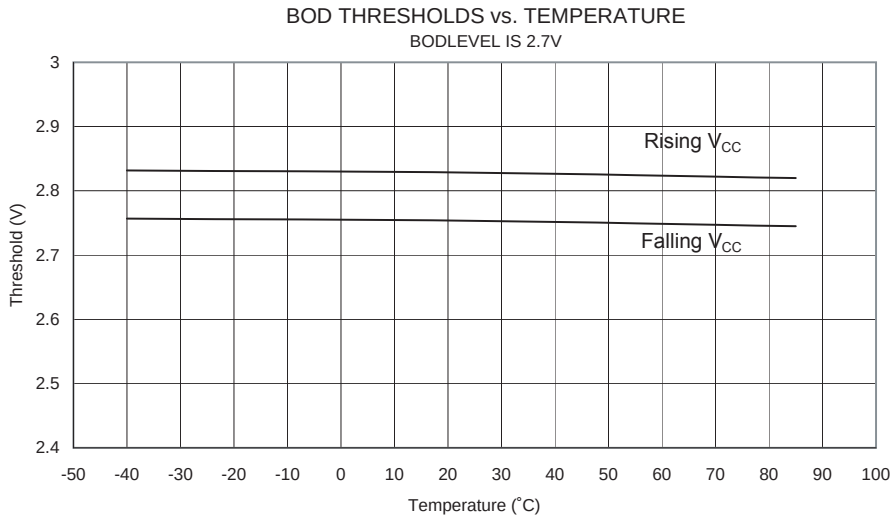


Figure 184. BOD 门限值和温度的关系 (BOD 电平为 1.8V)

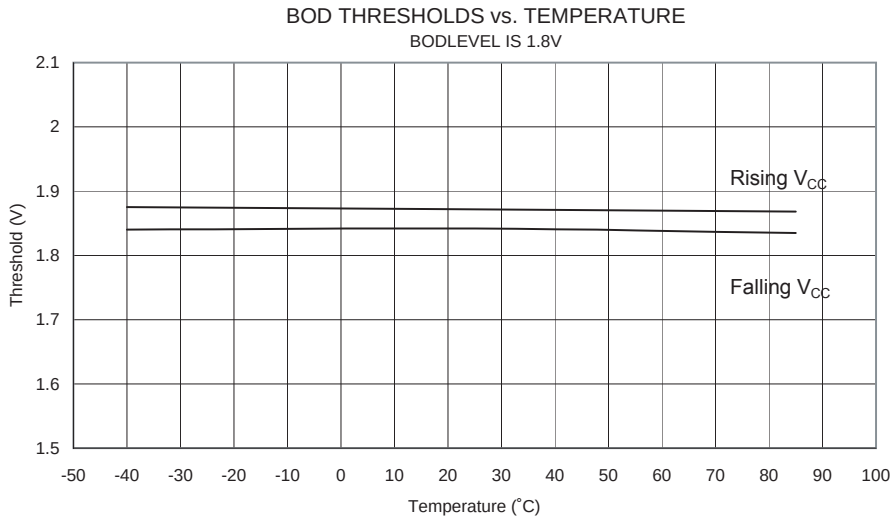


Figure 185. 能隙电压和 V_{CC} 的关系

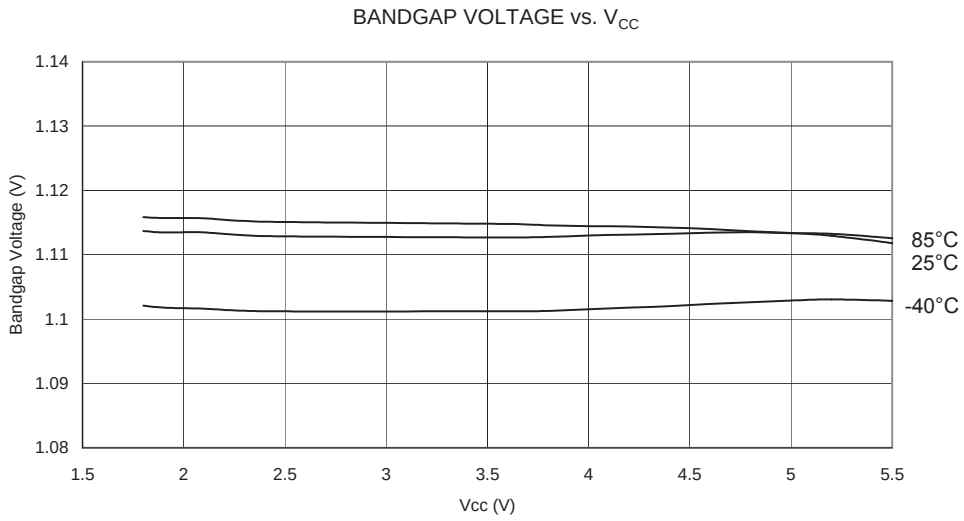


Figure 186. 模拟比较器偏置电压和共模电压的关系 (V_{CC} = 5V)

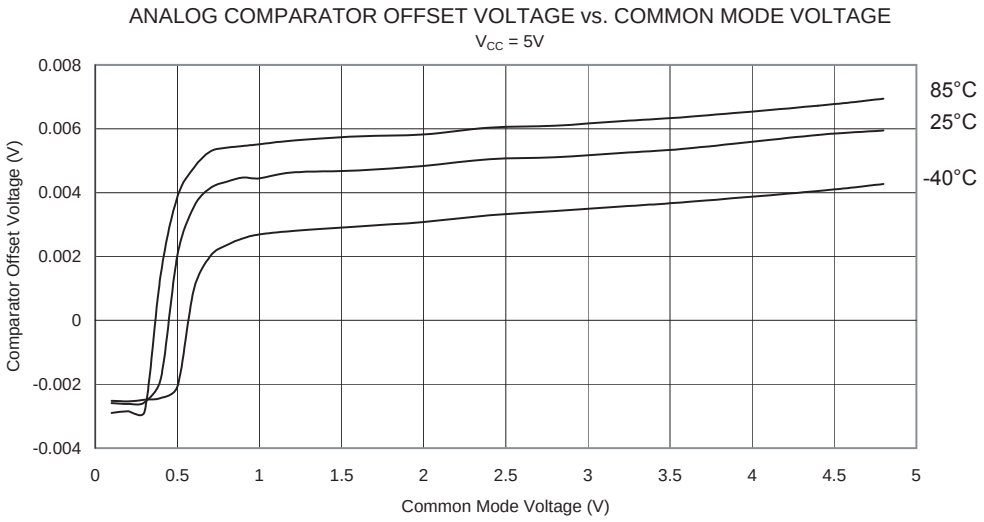
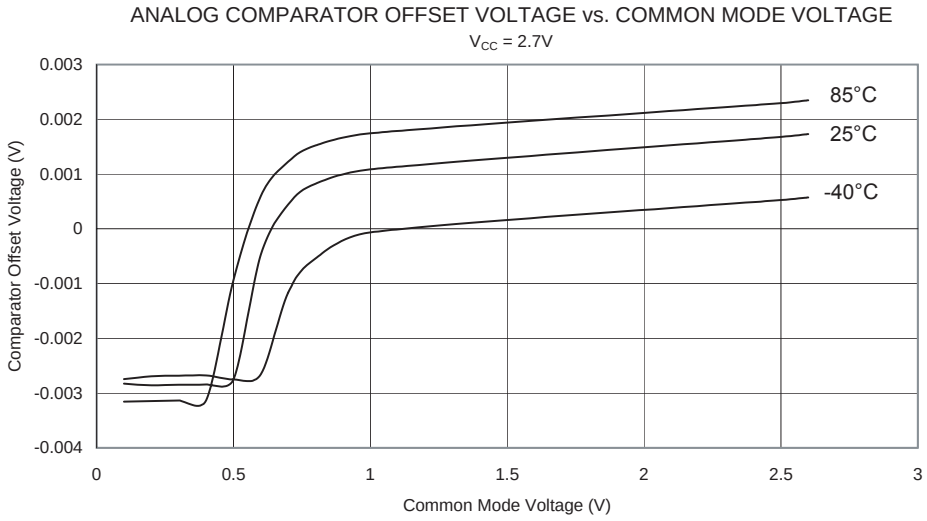


Figure 187. 模拟比较器偏置电压和共模电压的关系 (V_{CC} = 2.7V)



内部振荡器速率

Figure 188. 看门狗振荡器频率和 V_{CC} 的关系

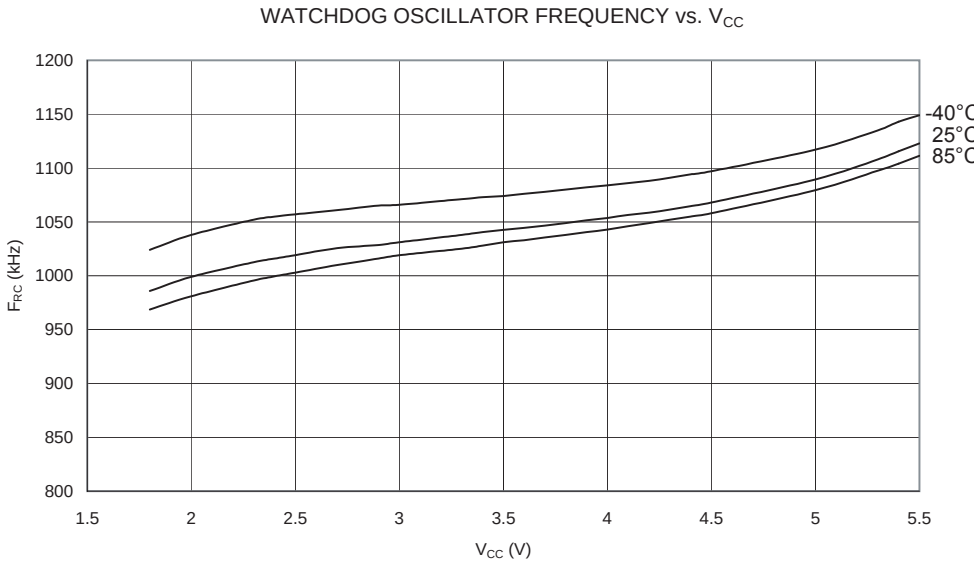


Figure 189. 校准的 8 MHz RC 振荡器频率和温度的关系

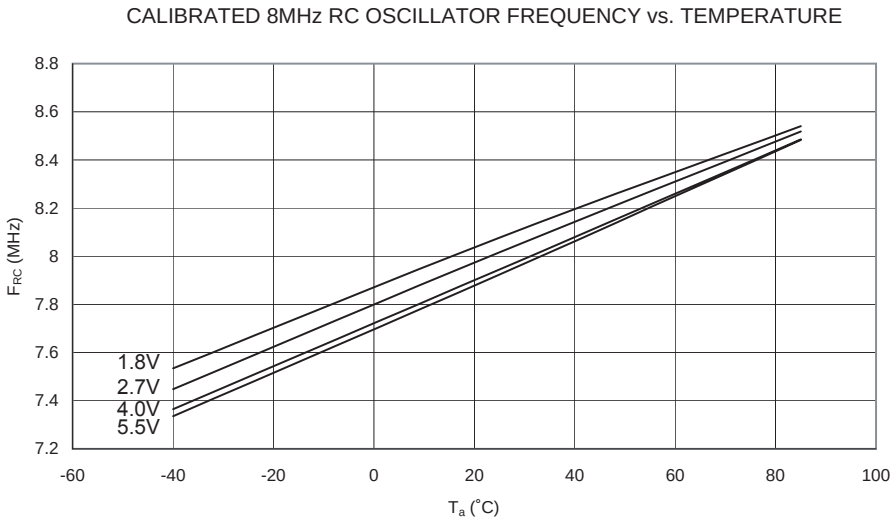


Figure 190. 校准的 8 MHz RC 振荡器频率和 V_{CC} 的关系

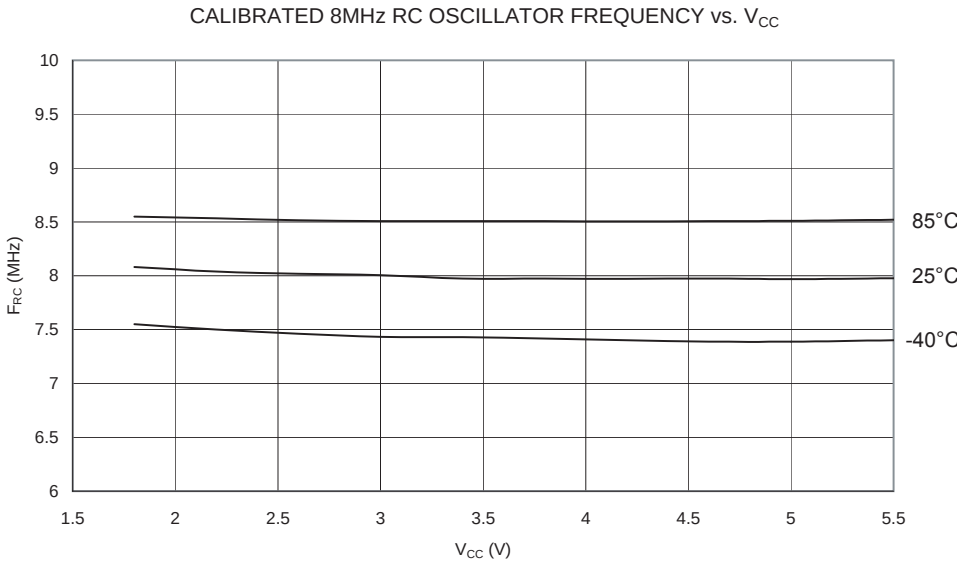
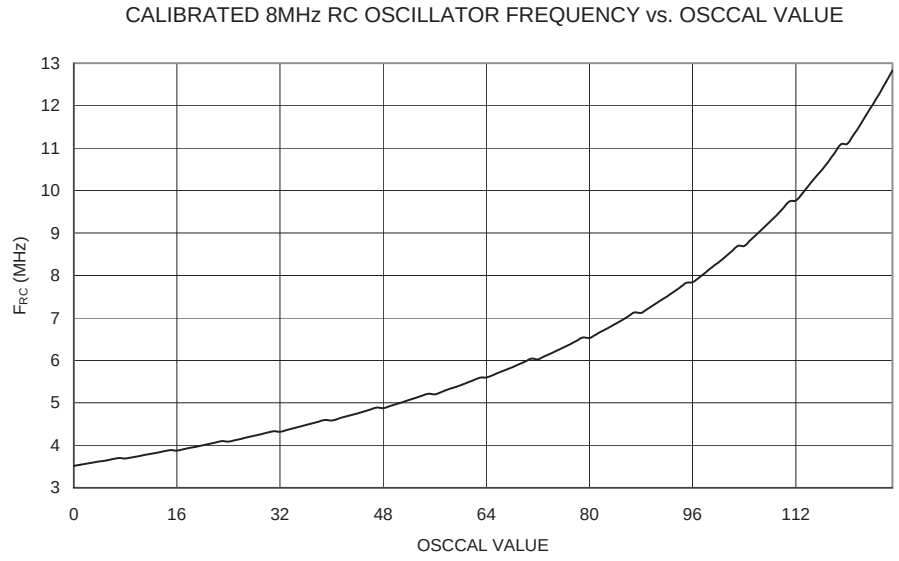


Figure 191. 校准的 8 MHz RC 振荡器频率和 Osccal 数值的关系



外围设备耗电流

Figure 192. BOD 电流和 V_{CC} 的关系

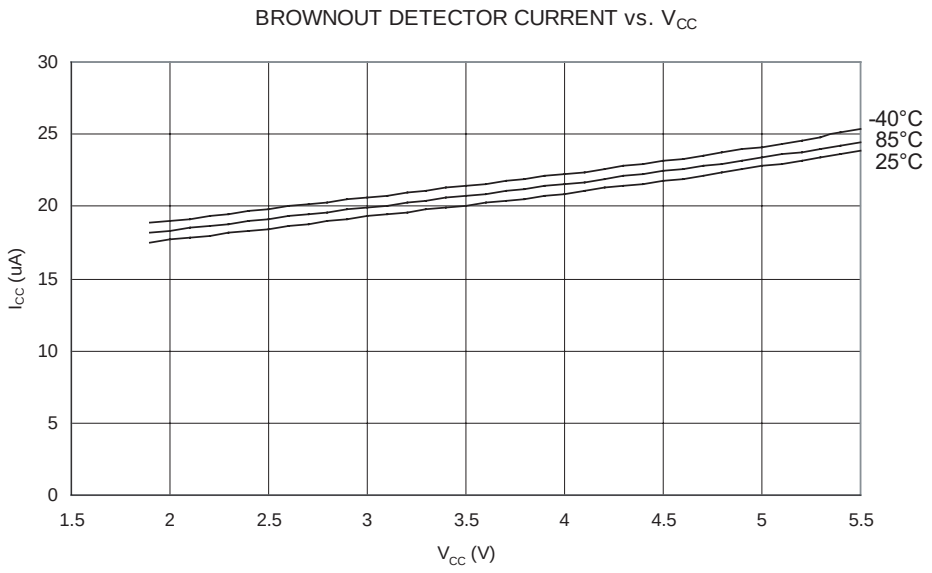


Figure 193. ADC 电流和 V_{CC} 的关系 (AREF = AVCC)

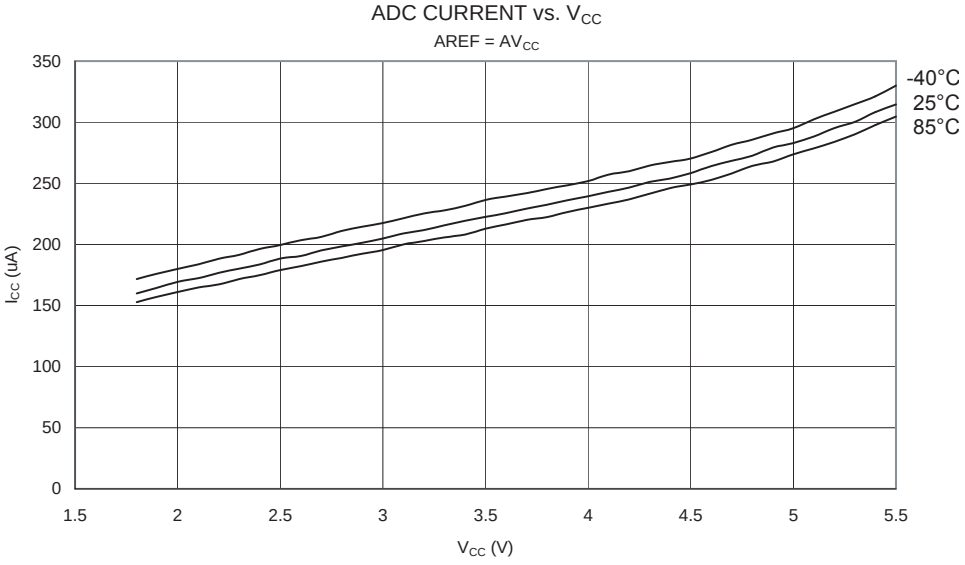


Figure 194. AREF 外部参考电流和 V_{CC} 的关系

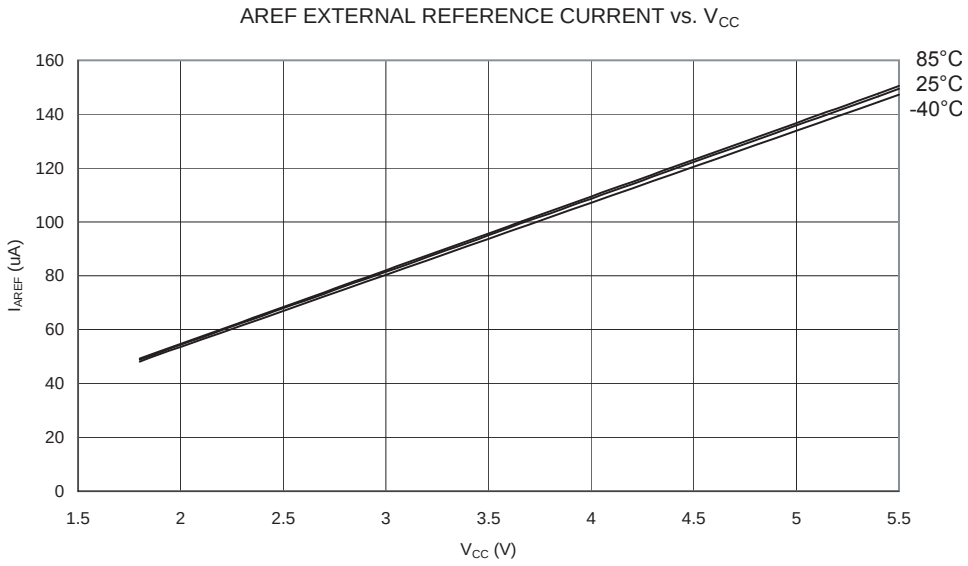


Figure 195. 32 kHZ TOSC 电流和 V_{CC} 的关系 (看门狗定时器禁止)

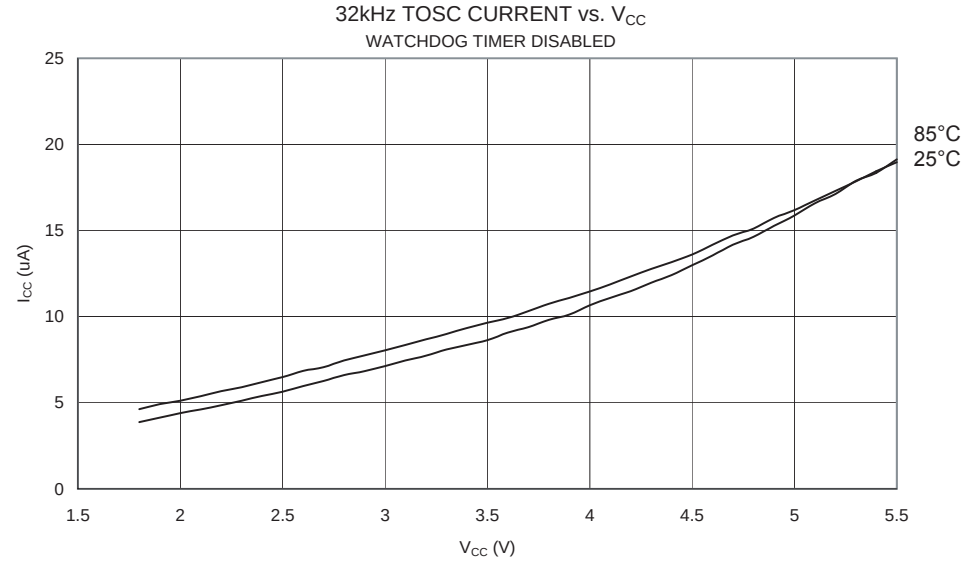


Figure 196. 看门狗定时器电流和 V_{CC} 的关系

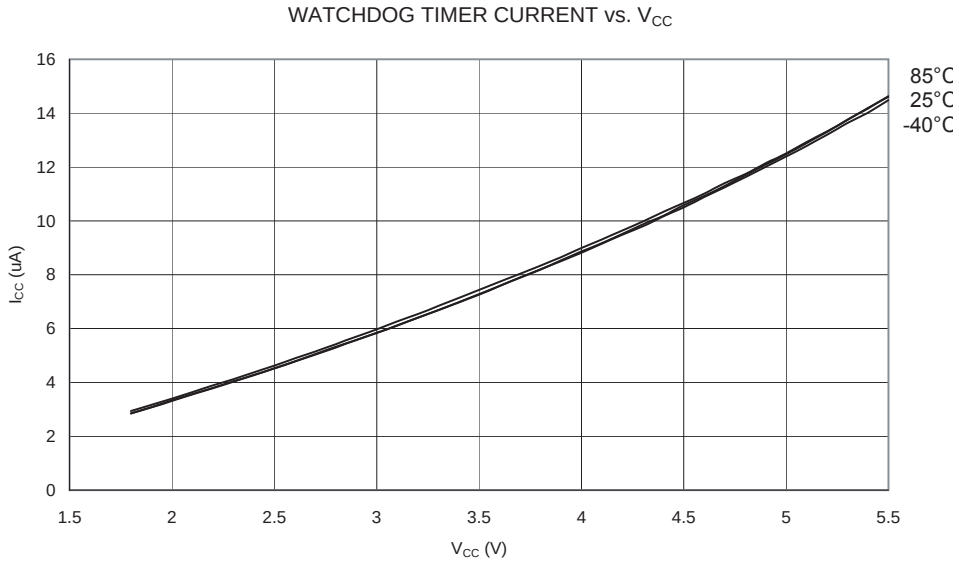


Figure 197. 模拟比较器电流和 V_{CC} 的关系

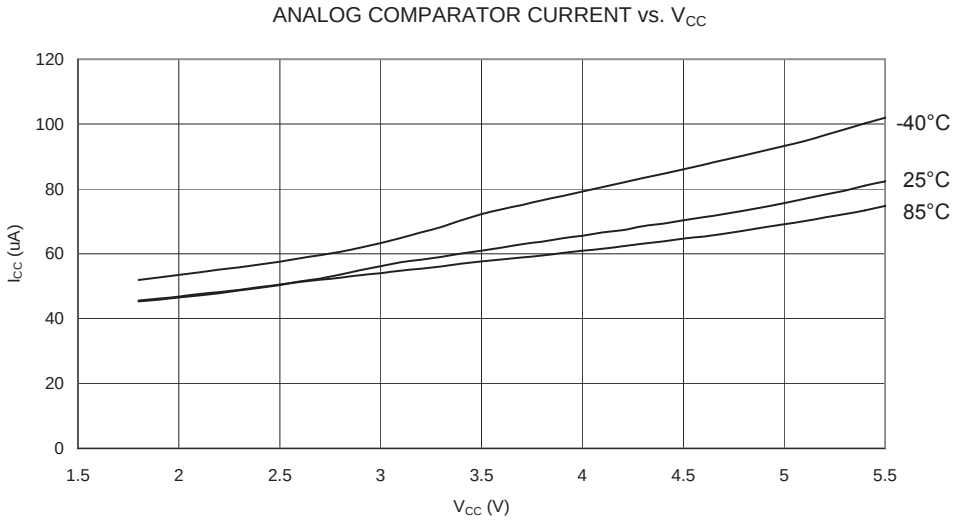
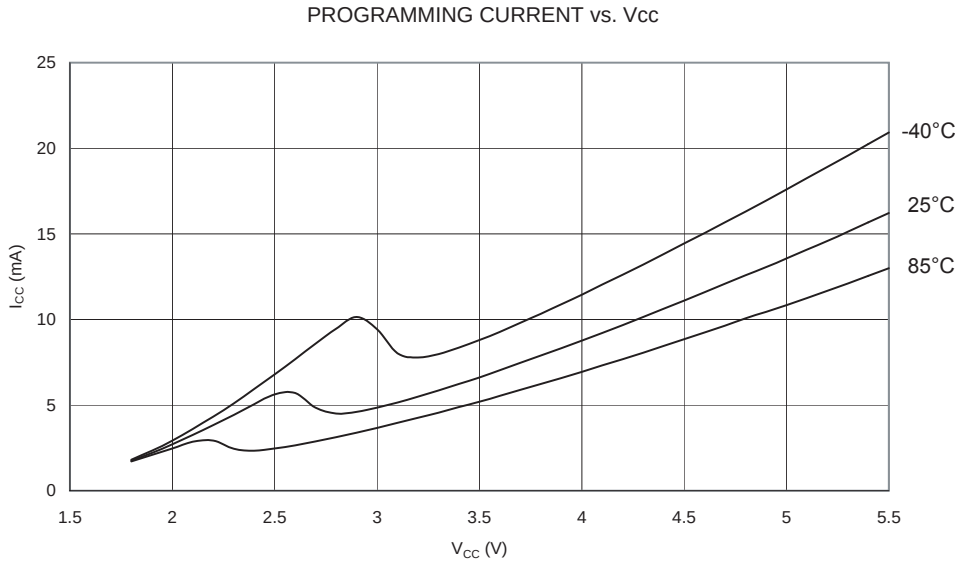


Figure 198. 编程电流和 V_{CC} 的关系



复位电流消耗及复位脉宽

Figure 199. 复位电流和 V_{CC} 的关系 (0.1 - 1.0 MHz，包括流经复位上拉电阻的电流)

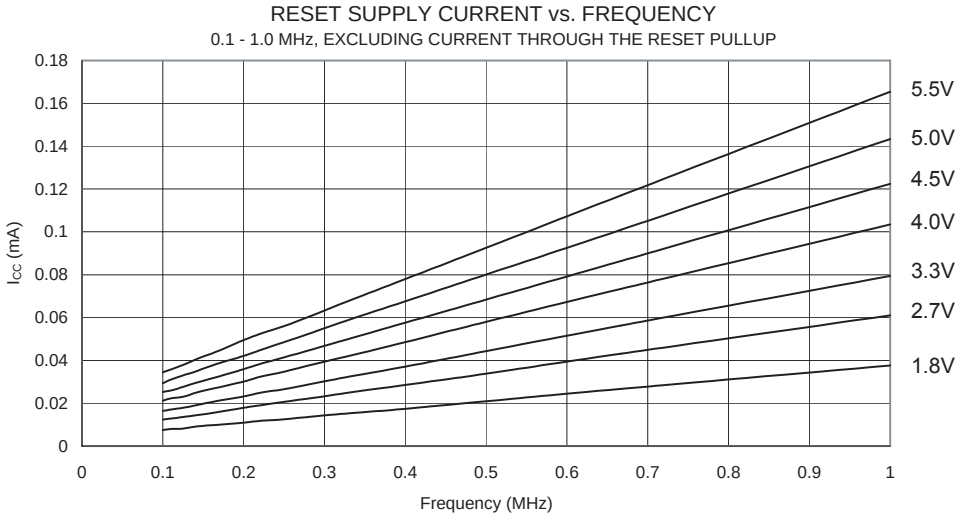


Figure 200. 复位电流和 V_{CC} 的关系 (1 - 20 MHz，包括流经复位上拉电阻的电流)

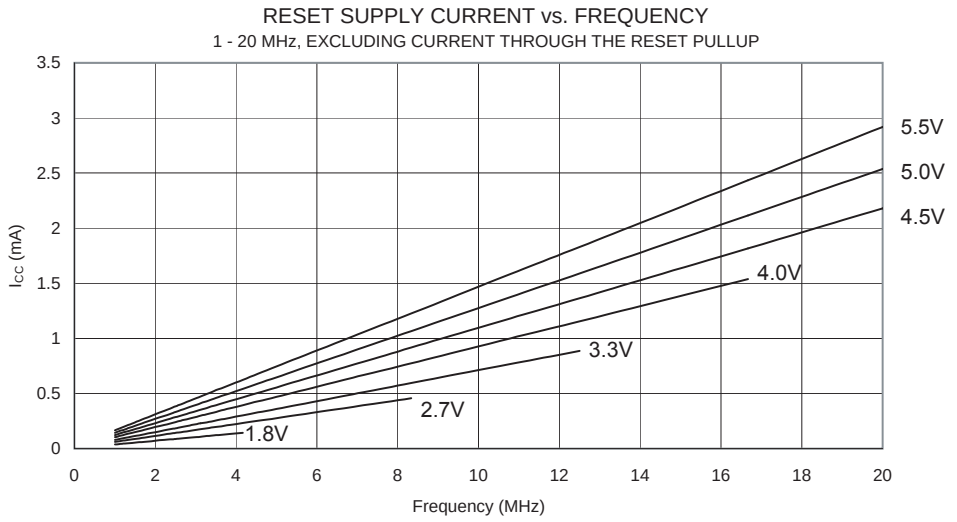
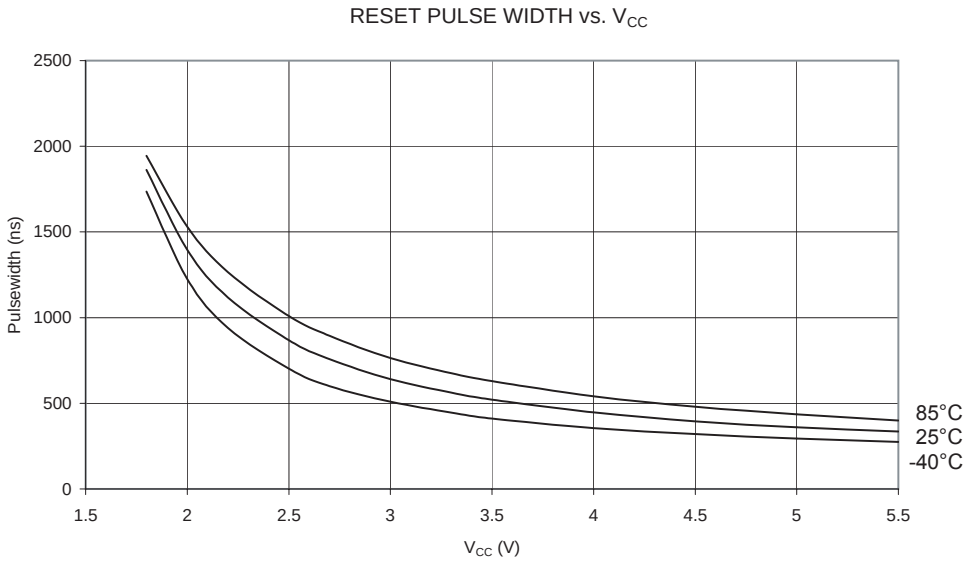


Figure 201. 复位脉宽和 V_{CC} 的关系



寄存器概述

地址	名称	位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	页码
(0xFF)	保留	–	–	–	–	–	–	–	–	
(0xFE)	LCDDR18	–	–	–	–	–	–	–	SEG324	216
(0xFD)	LCDDR17	SEG323	SEG322	SEG321	SEG320	SEG319	SEG318	SEG317	SEG316	216
(0xFC)	LCDDR16	SEG315	SEG314	SEG313	SEG312	SEG311	SEG310	SEG309	SEG308	216
(0xFB)	LCDDR15	SEG307	SEG306	SEG305	SEG304	SEG303	SEG302	SEG301	SEG300	216
(0xFA)	保留	–	–	–	–	–	–	–	–	
(0xF9)	LCDDR13	–	–	–	–	–	–	–	SEG224	216
(0xF8)	LCDDR12	SEG223	SEG222	SEG221	SEG220	SEG219	SEG218	SEG217	SEG216	216
(0xF7)	LCDDR11	SEG215	SEG214	SEG213	SEG212	SEG211	SEG210	SEG209	SEG208	216
(0xF6)	LCDDR10	SEG207	SEG206	SEG205	SEG204	SEG203	SEG202	SEG201	SEG200	216
(0xF5)	保留	–	–	–	–	–	–	–	–	
(0xF4)	LCDDR8	–	–	–	–	–	–	–	SEG124	216
(0xF3)	LCDDR7	SEG123	SEG122	SEG121	SEG120	SEG119	SEG118	SEG117	SEG116	216
(0xF2)	LCDDR6	SEG115	SEG114	SEG113	SEG112	SEG111	SEG110	SEG109	SEG108	216
(0xF1)	LCDDR5	SEG107	SEG106	SEG105	SEG104	SEG103	SEG102	SEG101	SEG100	216
(0xF0)	保留	–	–	–	–	–	–	–	–	
(0xEF)	LCDDR3	–	–	–	–	–	–	–	SEG024	216
(0xEE)	LCDDR2	SEG023	SEG022	SEG021	SEG020	SEG019	SEG018	SEG017	SEG016	216
(0xED)	LCDDR1	SEG015	SEG014	SEG013	SEG012	SEG011	SEG010	SEG09	SEG008	216
(0xEC)	LCDDR0	SEG007	SEG006	SEG005	SEG004	SEG003	SEG002	SEG001	SEG000	216
(0xEB)	保留	–	–	–	–	–	–	–	–	
(0xEA)	保留	–	–	–	–	–	–	–	–	
(0xE9)	保留	–	–	–	–	–	–	–	–	
(0xE8)	保留	–	–	–	–	–	–	–	–	
(0xE7)	LCDCCR	–	–	–	–	LCDCC3	LCDCC2	LCDCC1	LCDCC0	216
(0xE6)	LCDFRR	–	LCDPS2	LCDPS1	LCDPS0	–	LCDCD2	LCDCD1	LCDCD0	214
(0xE5)	LCDCRB	LCDCS	LCD2B	LCDMUX1	LCDMUX0	–	LCDPM2	LCDPM1	LCDPM0	212
(0xE4)	LCDCRA	LCDEN	LCDAB	–	LCDIF	LCDIE	–	–	LCDBL	212
(0xE3)	保留	–	–	–	–	–	–	–	–	
(0xE2)	保留	–	–	–	–	–	–	–	–	
(0xE1)	保留	–	–	–	–	–	–	–	–	
(0xE0)	保留	–	–	–	–	–	–	–	–	
(0xDF)	保留	–	–	–	–	–	–	–	–	
(0xDE)	保留	–	–	–	–	–	–	–	–	
(0xDD)	保留	–	–	–	–	–	–	–	–	
(0xDC)	保留	–	–	–	–	–	–	–	–	
(0xDB)	保留	–	–	–	–	–	–	–	–	
(0xDA)	保留	–	–	–	–	–	–	–	–	
(0xD9)	保留	–	–	–	–	–	–	–	–	
(0xD8)	保留	–	–	–	–	–	–	–	–	
(0xD7)	保留	–	–	–	–	–	–	–	–	
(0xD6)	保留	–	–	–	–	–	–	–	–	
(0xD5)	保留	–	–	–	–	–	–	–	–	
(0xD4)	保留	–	–	–	–	–	–	–	–	
(0xD3)	保留	–	–	–	–	–	–	–	–	
(0xD2)	保留	–	–	–	–	–	–	–	–	
(0xD1)	保留	–	–	–	–	–	–	–	–	
(0xD0)	保留	–	–	–	–	–	–	–	–	
(0xCF)	保留	–	–	–	–	–	–	–	–	
(0xCE)	保留	–	–	–	–	–	–	–	–	
(0xCD)	保留	–	–	–	–	–	–	–	–	
(0xCC)	保留	–	–	–	–	–	–	–	–	
(0xCB)	保留	–	–	–	–	–	–	–	–	
(0xCA)	保留	–	–	–	–	–	–	–	–	
(0xC9)	保留	–	–	–	–	–	–	–	–	
(0xC8)	保留	–	–	–	–	–	–	–	–	
(0xC7)	保留	–	–	–	–	–	–	–	–	
(0xC6)	UDR	USART I/O 数据寄存器								160
(0xC5)	UBRRH					USART 波特率寄存器高位				165
(0xC4)	UBRRL	USART 波特率寄存器低位								165
(0xC3)	保留	–	–	–	–	–	–	–	–	
(0xC2)	UCSRC	–	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	161
(0xC1)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	161
(0xC0)	UCSRA	RXC	TXC	UDRE	FE	DOR	UPE	U2X	MPCM	161

地址	名称	位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	页码
(0xBF)	保留	–	–	–	–	–	–	–	–	
(0xBE)	保留	–	–	–	–	–	–	–	–	
(0xBD)	保留	–	–	–	–	–	–	–	–	
(0xBC)	保留	–	–	–	–	–	–	–	–	
(0xBB)	保留	–	–	–	–	–	–	–	–	
(0xBA)	USIDR	USI 数据寄存器								176
(0xB9)	USISR	USISIF	USIOIF	USIPF	USIDC	USICNT3	USICNT2	USICNT1	USICNT0	176
(0xB8)	USICR	USISIE	USIOIE	USIWM1	USIWM0	USICS1	USICS0	USICLK	USITC	177
(0xB7)	保留	–		–	–	–	–	–	–	
(0xB6)	ASSR	–	–	–	EXCLK	AS2	TCN2UB	OCR2UB	TCR2UB	131
(0xB5)	保留	–	–	–	–	–	–	–	–	
(0xB4)	保留	–	–	–	–	–	–	–	–	
(0xB3)	OCR2A	定时器 / 计数器 2 输出比较寄存器 A								130
(0xB2)	TCNT2	定时器 / 计数器 2(8 位)								130
(0xB1)	保留	–	–	–	–	–	–	–	–	
(0xB0)	TCCR2A	FOC2A	WGM20	COM2A1	COM2A0	WGM21	CS22	CS21	CS20	128
(0xAF)	保留	–	–	–	–	–	–	–	–	
(0xAE)	保留	–	–	–	–	–	–	–	–	
(0xAD)	保留	–	–	–	–	–	–	–	–	
(0xAC)	保留	–	–	–	–	–	–	–	–	
(0xAB)	保留	–	–	–	–	–	–	–	–	
(0xAA)	保留	–	–	–	–	–	–	–	–	
(0xA9)	保留	–	–	–	–	–	–	–	–	
(0xA8)	保留	–	–	–	–	–	–	–	–	
(0xA7)	保留	–	–	–	–	–	–	–	–	
(0xA6)	保留	–	–	–	–	–	–	–	–	
(0xA5)	保留	–	–	–	–	–	–	–	–	
(0xA4)	保留	–	–	–	–	–	–	–	–	
(0xA3)	保留	–	–	–	–	–	–	–	–	
(0xA2)	保留	–	–	–	–	–	–	–	–	
(0xA1)	保留	–	–	–	–	–	–	–	–	
(0xA0)	保留	–	–	–	–	–	–	–	–	
(0x9F)	保留	–	–	–	–	–	–	–	–	
(0x9E)	保留	–	–	–	–	–	–	–	–	
(0x9D)	保留	–	–	–	–	–	–	–	–	
(0x9C)	保留	–	–	–	–	–	–	–	–	
(0x9B)	保留	–	–	–	–	–	–	–	–	
(0x9A)	保留	–	–	–	–	–	–	–	–	
(0x99)	保留	–	–	–	–	–	–	–	–	
(0x98)	保留	–	–	–	–	–	–	–	–	
(0x97)	保留	–	–	–	–	–	–	–	–	
(0x96)	保留	–	–	–	–	–	–	–	–	
(0x95)	保留	–	–	–	–	–	–	–	–	
(0x94)	保留	–	–	–	–	–	–	–	–	
(0x93)	保留	–	–	–	–	–	–	–	–	
(0x92)	保留	–	–	–	–	–	–	–	–	
(0x91)	保留	–	–	–	–	–	–	–	–	
(0x90)	保留	–	–	–	–	–	–	–	–	
(0x8F)	保留	–	–	–	–	–	–	–	–	
(0x8E)	保留	–	–	–	–	–	–	–	–	
(0x8D)	保留	–	–	–	–	–	–	–	–	
(0x8C)	保留	–	–	–	–	–	–	–	–	
(0x8B)	OCR1BH	定时器 / 计数器 1 输出比较寄存器 B 高字节								116
(0x8A)	OCR1BL	定时器 / 计数器 1 输出比较寄存器 B 低字节								116
(0x89)	OCR1AH	定时器 / 计数器 1 输出比较寄存器 A 高字节								116
(0x88)	OCR1AL	定时器 / 计数器 1 输出比较寄存器 A 低字节								116
(0x87)	ICR1H	定时器 / 计数器 1 输入捕捉寄存器高字节								122
(0x86)	ICR1L	定时器 / 计数器 1 输入捕捉寄存器低字节								122
(0x85)	TCNT1H	定时器 / 计数器 1 计数器寄存器高字节								115
(0x84)	TCNT1L	定时器 / 计数器 1 计数器寄存器低字节								115
(0x83)	保留	–	–	–	–	–	–	–	–	
(0x82)	TCCR1C	FOC1A	FOC1B	–	–	–	–	–	–	115
(0x81)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	114
(0x80)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	112
(0x7F)	DIDR1	–	–	–	–	–	–	AIN1D	AIN0D	183
(0x7E)	DIDR0	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	201

地址	名称	位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	页码
(0x7D)	保留	–	–	–	–	–	–	–	–	
(0x7C)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	197
(0x7B)	ADCSRB	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	181, 200
(0x7A)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	199
(0x79)	ADCH	ADC 数据寄存器高字节								200
(0x78)	ADCL	ADC 数据寄存器低字节								200
(0x77)	保留	–	–	–	–	–	–	–	–	
(0x76)	保留	–	–	–	–	–	–	–	–	
(0x75)	保留	–	–	–	–	–	–	–	–	
(0x74)	保留	–	–	–	–	–	–	–	–	
(0x73)	保留	–	–	–	–	–	–	–	–	
(0x72)	保留	–	–	–	–	–	–	–	–	
(0x71)	保留	–	–	–	–	–	–	–	–	
(0x70)	TIMSK2	–	–	–	–	–	–	OCIE2A	TOIE2	133
(0x6F)	TIMSK1	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	117
(0x6E)	TIMSK0	–	–	–	–	–	–	OCIE0A	TOIE0	89
(0x6D)	保留	–	–	–	–	–	–	–	–	
(0x6C)	PCMSK1	PCINT15	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	78
(0x6B)	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	78
(0x6A)	保留	–	–	–	–	–	–	–	–	
(0x69)	EICRA	–	–	–	–	–	–	ISC01	ISC00	76
(0x68)	保留	–	–	–	–	–	–	–	–	
(0x67)	保留	–	–	–	–	–	–	–	–	
(0x66)	OSCCAL	振荡器寄存器								28
(0x65)	保留	–	–	–	–	–	–	–	–	
(0x64)	保留	–	–	–	–	–	–	–	–	
(0x63)	保留	–	–	–	–	–	–	–	–	
(0x62)	保留	–	–	–	–	–	–	–	–	
(0x61)	CLKPR	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0	30
(0x60)	WDTCR	–	–	–	WDCE	WDE	WDP2	WDP1	WDP0	42
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C	9
0x3E (0x5E)	SPH	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	11
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	11
0x3C (0x5C)	保留									
0x3B (0x5B)	保留									
0x3A (0x5A)	保留									
0x39 (0x59)	保留									
0x38 (0x58)	保留									
0x37 (0x57)	SPMCSR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	246
0x36 (0x56)	保留	–	–	–	–	–	–	–	–	
0x35 (0x55)	MCUCR	JTD	–	–	PUD	–	–	IVSEL	IVCE	227
0x34 (0x54)	MCUSR	–	–	–	JTRF	WDRF	BORF	EXTRF	PORF	228
0x33 (0x53)	SMCR	–	–	–	–	SM2	SM1	SM0	SE	32
0x32 (0x52)	保留	–	–	–	–	–	–	–	–	
0x31 (0x51)	OCDR	IDRD/OCD	OCDR6	OCDR5	OCDR4	OCDR3	OCDR2	OCDR1	OCDR0	223
0x30 (0x50)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	181
0x2F (0x4F)	保留	–	–	–	–	–	–	–	–	
0x2E (0x4E)	SPDR	SPI 数据寄存器								141
0x2D (0x4D)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	141
0x2C (0x4C)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	139
0x2B (0x4B)	GPIOR2	通用 I/O 寄存器 2								22
0x2A (0x4A)	GPIOR1	通用 I/O 寄存器 1								22
0x29 (0x49)	保留	–	–	–	–	–	–	–	–	
0x28 (0x48)	保留	–	–	–	–	–	–	–	–	
0x27 (0x47)	OCR0A	定时器 / 计数器 0 输出比较寄存器 A								89
0x26 (0x46)	TCNT0	定时器 / 计数器 0 (8 位)								91
0x25 (0x45)	保留	–	–	–	–	–	–	–	–	
0x24 (0x44)	TCCR0A	FOC0A	WGM00	COM0A1	COM0A0	WGM01	CS02	CS01	CS00	89
0x23 (0x43)	GTCCR	TSM	–	–	–	–	–	PSR2	PSR10	91
0x22 (0x42)	EEARH	–	–	–	–	–	–	–	EEAR8	18
0x21 (0x41)	EEARL	EEPROM 地址寄存器低字节								18
0x20 (0x40)	EEDR	EEPROM 数据寄存器								18
0x1F (0x3F)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE	18
0x1E (0x3E)	GPIOR0	通用 I/O 寄存器 0								22
0x1D (0x3D)	EIMSK	PCIE1	PCIE0	–	–	–	–	–	INT0	77
0x1C (0x3C)	EIFR	PCIF1	PCIF0	–	–	–	–	–	INTF0	77

地址	名称	位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	页码
0x1B (0x3B)	保留	–	–	–	–	–	–	–	–	
0x1A (0x3A)	保留	–	–	–	–	–	–	–	–	
0x19 (0x39)	保留	–	–	–	–	–	–	–	–	
0x18 (0x38)	保留	–	–	–	–	–	–	–	–	
0x17 (0x37)	TIFR2	–	–	–	–	–	–	OCF2A	TOV2	141
0x16 (0x36)	TIFR1	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	118
0x15 (0x35)	TIFR0	–	–	–	–	–	–	OCF0A	TOV0	89
0x14 (0x34)	PORTG	–	–	–	PORTG4	PORTG3	PORTG2	PORTG1	PORTG0	75
0x13 (0x33)	DDRG	–	–	–	DDG4	DDG3	DDG2	DDG1	DDG0	73
0x12 (0x32)	PING	–	–	PING5	PING4	PING3	PING2	PING1	PING0	75
0x11 (0x31)	PORTF	PORTF7	PORTF6	PORTF5	PORTF4	PORTF3	PORTF2	PORTF1	PORTF0	74
0x10 (0x30)	DDRF	DDF7	DDF6	DDF5	DDF4	DDF3	DDF2	DDF1	DDF0	74
0x0F (0x2F)	PINF	PINF7	PINF6	PINF5	PINF4	PINF3	PINF2	PINF1	PINF0	75
0x0E (0x2E)	PORTE	PORTE7	PORTE6	PORTE5	PORTE4	PORTE3	PORTE2	PORTE1	PORTE0	72
0x0D (0x2D)	DDRE	DDE7	DDE6	DDE5	DDE4	DDE3	DDE2	DDE1	DDE0	74
0x0C (0x2C)	PINE	PINE7	PINE6	PINE5	PINE4	PINE3	PINE2	PINE1	PINE0	74
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	74
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	74
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	74
0x08 (0x28)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	73
0x07 (0x27)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	71
0x06 (0x26)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	72
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	73
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	73
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	73
0x02 (0x22)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	71
0x01 (0x21)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	71
0x00 (0x20)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	73

Note:
1. 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。
2. 通过 SBI 和 CBI 指令可直接对地址从 0x00 - 0x1F 的 I/O 寄存器进行位寻址。在这些寄存器中，单个位的值可以通过 SBIS 和 SBIC 指令进行查询。
3. 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。
4. 当使用特殊的I/O操作命令IN和OUT时，其寻址范围限定为0x00 - 0x3F。使用LD 和ST指令可以像操作普通数据空间一样对I/O寄存器进行寻址，这时必须在 I/O 地址上加 0x20 。ATmega169 是一款复杂的微处理器，拥有的外设的 I/O 地址超出了 IN/OUT 指令所能访问的 64 个地址。此时只能利用 ST/STS/STD 和 LD/LDS/LDD 指令来象访问 SRAM 一样访问位于 0x60 - 0xFF 的扩展 I/O 空间。

指令集概述

指令	操作数	说明	操作	标志	时钟数
		算术和逻辑指令			
ADD	Rd, Rr	无进位加法	Rd ← Rd + Rr	Z,C,N,V,S,H	1
ADC	Rd, Rr	带进位加法	Rd ← Rd + Rr + C	Z,C,N,V,S,H	1
ADIW	Rd, K	立即数与字相加	Rd+1:Rd ← Rd+1:Rd + K	Z,C,N,V,S	2
SUB	Rd, Rr	无进位减法	Rd ← Rd - Rr	Z,C,N,V,S,H	1
SUBI	Rd, K	减立即数	Rd ← Rd - K	Z,C,N,V,S,H	1
SBC	Rd, Rr	带进位减法	Rd ← Rd - Rr - C	Z,C,N,V,S,H	1
SBCI	Rd, K	带进位减立即数	Rd ← Rd - K - C	Z,C,N,V,S,H	1
SBIW	Rd, K	从字中减去立即数	Rd+1:Rd ← Rd+1:Rd - K	Z,C,N,V,S	2
AND	Rd, Rr	逻辑与	Rd ← Rd • Rr	Z,N,V,S	1
ANDI	Rd, K	与立即数的逻辑 与操作	Rd ← Rd • K	Z,N,V,S	1
OR	Rd, Rr	逻辑或	Rd ← Rd v Rr	Z,N,V,S	1
ORI	Rd, K	与立即数的逻辑 或操作	Rd ← Rd v K	Z,N,V,S	1
EOR	Rd, Rr	异或	Rd ← Rd ⊕ Rr	Z,N,V,S	1
COM	Rd	1 的补码	Rd ← $FF - Rd	Z,C,N,V,S	1
NEG	Rd	2 的补码	Rd ← $00 - Rd	Z,C,N,V,S,H	1
SBR	Rd, K	设置寄存器的位	Rd ← Rd v K	Z,N,V,S	1
CBR	Rd, K	寄存器位清零	Rd ← Rd • ($FFh - K)	Z,N,V,S	1
INC	Rd	加一操作	Rd ← Rd + 1	Z,N,V,S	1
DEC	Rd	减一操作	Rd ← Rd - 1	Z,N,V,S	1
TST	Rd	测试是否为零或负	Rd ← Rd • Rd	Z,N,V,S	1
CLR	Rd	寄存器赋零	Rd ← Rd ⊕ Rd	Z,N,V,S	1
SER	Rd	寄存器置位	Rd ← $FF	无	1
MUL	Rd, Rr	无符号数乘法	R1:R0 ← Rd × Rr (UU)	Z,C	2
MULS	Rd, Rr	有符号数乘法	R1:R0 ← Rd × Rr (SS)	Z,C	2
MULSU	Rd, Rr	有符号数与无符号数相乘	R1:R0 ← Rd × Rr (SU)	Z,C	2
FMUL	Rd, Rr	无符号小数乘法	R1:R0 ← (Rd × Rr)<<1 (UU)	Z,C	2
FMULS	Rd, Rr	有符号小数乘法	R1:R0 ← (Rd × Rr)<<1 (SS)	Z,C	2
FMULSU	Rd, Rr	有符号小数与无符号小数相乘	R1:R0 ← (Rd × Rr)<<1 (SU)	Z,C	2
		跳转指令			
RJMP	k	相对跳转	PC ← PC + k + 1	无	2
IJMP		间接跳转到 (Z)	PC(15:0) ← Z	无	2
JMP	k	跳转	PC ← k	无	3
RCALL	k	相对子程序调用	PC ← PC + k + 1	无	3

指令	操作数	说明	操作	标志	时钟数
ICALL		间接调用 (Z)	PC(15:0) ← Z	无	3
CALL	k	调用子程序	PC ← k	无	4
RET		子程序返回	PC ← STACK	无	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	无	1 / 2 / 3
CP	Rd, Rr	比较	Rd - Rr	Z,C,N,V,S,H	1
CPC	Rd, Rr	带进位比较	Rd - Rr - C	Z,C,N,V,S,H	1
CPI	Rd, K	与立即数比较	Rd - K	Z,C,N,V,S,H	1
SBRC	Rr, b	寄存器位 b 为 0 则跳过下一条指令	if (Rr(b) = 0) PC ← PC + 2 or 3	无	1 / 2 / 3
SBRS	Rr, b	寄存器位 b 为 1 则跳过下一条指令	if (Rr(b) = 1) PC ← PC + 2 or 3	无	1 / 2 / 3
SBIC	A, b	I/O 寄存器位 b 为 0 则跳过下一条指令	if(I/O(A,b) = 0) PC ← PC + 2 or 3	无	1 / 2 / 3
SBIS	A, b	I/O 寄存器位 b 为 1 则跳过下一条指令	If(I/O(A,b) = 1) PC ← PC + 2 or 3	无	1 / 2 / 3
BRBS	s, k	状态寄存器位 s 为 1 则跳转	if (SREG(s) = 1) then PC ←PC+k+1	无	1 / 2
BRBC	s, k	状态寄存器位 s 为 0 则跳转	if (SREG(s) = 0) then PC ←PC+k+1	无	1 / 2
BREQ	k	相等则跳转	if (Z = 1) then PC ← PC + k + 1	无	1 / 2
BRNE	k	不相等则跳转	if (Z = 0) then PC ← PC + k + 1	无	1 / 2
BRCS	k	进位位为 1 则跳转	if (C = 1) then PC ← PC + k + 1	无	1 / 2
BRCC	k	进位位为 0 则跳转	if (C = 0) then PC ← PC + k + 1	无	1 / 2
BRSH	k	大于或等于则跳转	if (C = 0) then PC ← PC + k + 1	无	1 / 2
BRLO	k	小于则跳转	if (C = 1) then PC ← PC + k + 1	无	1 / 2
BRMI	k	负则跳转	if (N = 1) then PC ← PC + k + 1	无	1 / 2
BRPL	k	正则跳转	if (N = 0) then PC ← PC + k + 1	无	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	无	1 / 2
BRLT	k	无符号数，小于则跳转	if (N ⊕ V= 1) then PC ← PC + k + 1	无	1 / 2
BRHS	k	半进位位为 1 则跳转	if (H = 1) then PC ← PC + k + 1	无	1 / 2
BRHC	k	半进位位为 0 则跳转	if (H = 0) then PC ← PC + k + 1	无	1 / 2
BRTS	k	T 为 1 则跳转	if (T = 1) then PC ← PC + k + 1	无	1 / 2
BRTC	k	T 为 0 则跳转	if (T = 0) then PC ← PC + k + 1	无	1 / 2
BRVS	k	溢出标志为 1 则跳转	if (V = 1) then PC ← PC + k + 1	无	1 / 2
BRVC	k	溢出标志为 0 则跳转	if (V = 0) then PC ← PC + k + 1	无	1 / 2
BRIE	k	中断使能则跳转	if (I = 1) then PC ← PC + k + 1	无	1 / 2
BRID	k	中断禁止则跳转	if (I = 0) then PC ← PC + k + 1	无	1 / 2
		位和位测试指令			
SBI	A, b	I/O 寄存器的位置位	I/O(A, b) ← 1	无	2
CBI	A, b	I/O 寄存器的位清零	I/O(A, b) ← 0	无	2

指令	操作数	说明	操作	标志	时钟数
LSL	Rd	逻辑左移	Rd(n+1)←Rd(n),Rd(0)←0,C←Rd(7)	Z,C,N,V,H	1
LSR	Rd	逻辑右移	Rd(n)←Rd(n+1),Rd(7)←0,C←Rd(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,H	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)	无	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 b 赋给 T	T ← Rr(b)	T	1
BLD	Rd, b	将 T 赋给寄存器位 b	Rd(b) ← T	无	1
SEC		置位进位位	C ← 1	C	1
CLC		进位位清零	C ← 0	C	1
SEN		置位负标志位	N ← 1	N	1
CLN		清除负标志位	N ← 0	N	1
SEZ		置位零标志位	Z ← 1	Z	1
CLZ		清除零标志位	Z ← 0	Z	1
SEI		全局中断使能	I ← 1	I	1
CLI		禁止全局中断	I ← 0	I	1
SES		置位符号测试标志位	S ← 1	S	1
CLS		清除符号测试标志位	S ← 0	S	1
SEV		置位 2 的补码溢出标志	V ← 1	V	1
CLV		清除 2 的补码溢出标志	V ← 0	V	1
SET		置位 SREG 的 T 标志	T ← 1	T	1
CLT		清除 SREG 的 T 标志	T ← 0	T	1
SEH		置位 SREG 的半进位标志	H ← 1	H	1
CLH		清除 SREG 的半进位标志	H ← 0	H	1
		数据传送指令			
MOV	Rd, Rr	寄存器之间拷贝	Rd ← Rr	无	1
MOVW	Rd, Rr	拷贝两个寄存器	Rd+1:Rd ← Rr+1:Rr	无	1
LDI	Rd, K	加载立即数	Rd ← K	无	1
LD	Rd, X	加载间接寻址的数据	Rd ← (X)	无	2
LD	Rd, X+	加载间接寻址数据，然后地址加一	Rd ← (X), X ← X + 1	无	2
LD	Rd, -X	地址减一后加载间接寻址数据	X ← X - 1, Rd ← (X)	无	2
LD	Rd, Y	加载间接寻址的数据	Rd ← (Y)	无	2
LD	Rd, Y+	加载间接寻址数据，然后地址加一	Rd ← (Y), Y ← Y + 1	无	2

指令	操作数	说明	操作	标志	时钟数
LD	Rd, -Y	地址减一后加载间接寻址数据	Y ← Y - 1, Rd ← (Y)	无	2
LDD	Rd, Y+q	加载带偏移量的间接寻址数据	Rd ← (Y + q)	无	2
LD	Rd, Z	加载间接寻址的数据	Rd ← (Z)	无	2
LD	Rd, Z+	加载间接寻址数据，然后地址加一	Rd ← (Z), Z ← Z+1	无	2
LD	Rd, -Z	地址减一后加载间接寻址数据	Z ← Z - 1, Rd ← (Z)	无	2
LDD	Rd, Z+q	加载带偏移量的间接寻址数据	Rd ← (Z + q)	无	2
LDS	Rd, k	从数据空间加载数据	Rd ← (k)	无	2
ST	X, Rr	以间接寻址方式存储数据	(X) ← Rr	无	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	无	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	无	2
ST	Y, Rr	以间接寻址方式存储数据	(Y) ← Rr	无	2
ST	Y+, Rr	以间接寻址方式存储数据，然后地址加一	(Y) ← Rr, Y ← Y + 1	无	2
ST	-Y, Rr	地址减一后以间接寻址方式存储数据	Y ← Y - 1, (Y) ← Rr	无	2
STD	Y+q, Rr	带偏移量的间接寻址方式存储数据	(Y + q) ← Rr	无	2
ST	Z, Rr	以间接寻址方式存储数据	(Z) ← Rr	无	2
ST	Z+, Rr	S 以间接寻址方式存储数据，然后地址加一	(Z) ← Rr, Z ← Z + 1	无	2
ST	-Z, Rr	地址减一后以间接寻址方式存储数据	Z ← Z - 1, (Z) ← Rr	无	2
STD	Z+q, Rr	以间接寻址方式存储数据	(Z + q) ← Rr	无	2
STS	k, Rr	以直接寻址方式存储数据	Rd ← (k)	无	2
LPM		加载程序空间的数据	R0 ← (Z)	无	3
LPM	Rd, Z	加载程序空间的数据	Rd ← (Z)	无	3
LPM	Rd, Z+	加载程序空间的数据，然后地址加一	Rd ← (Z), Z ← Z + 1	无	3
SPM		保存程序存储器的内容	(Z)← R1:R0	无	-
IN	Rd, A	从 I/O 地址读数据	Rd ← I/O(A)	无	1
OUT	A, Rr	向 I/O 地址输出数据	I/O(A) ← Rr	无	1
PUSH	Rr	将寄存器推入堆栈	STACK ← Rr	无	2
POP	Rd	将寄存器弹出堆栈	Rd ← STACK	无	2
		MCU 控制指令			
NOP		空操作		无	1
SLEEP		休眠	(参见数据手册)	无	1
WDR		复位看门狗	(参见数据手册 WDR 的说明)	无	1
BREAK		终止	只适用于片上调试	无	N/A

产品信息

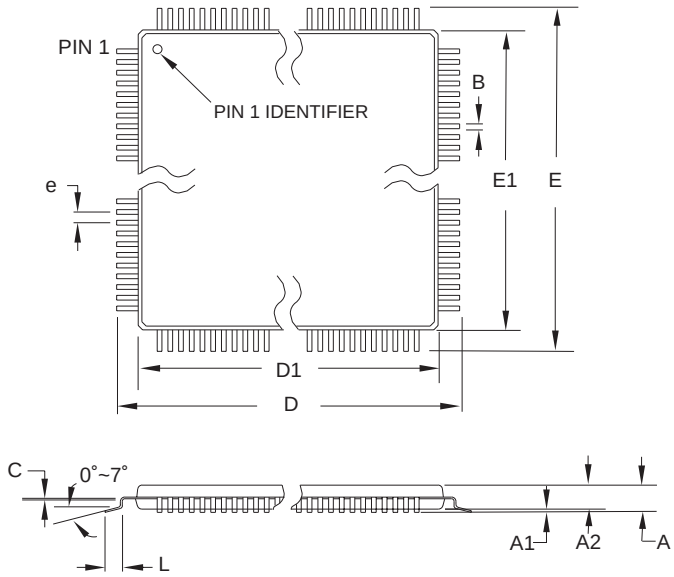
工作速度 (MHz)	电 源	定货号	封装	工作温度范围
1	1.8 - 5.5V	ATmega169V-1AI ATmega169V-1MI	64A 64M1	工业级 (-40°C - 85°C)
8	2.7 - 5.5V	ATmega169L-8AI ATmega169L-8MI	64A 64M1	工业级 (-40°C - 85°C)
16	4.5 - 5.5V	ATmega169-16AI ATmega169-16MI	64A 64M1	工业级 (-40°C - 85°C)

Note: 产品也可以 wafer 的形式提供，订货信息细节以及最小定货量请与 Atmel 当地机构联系。

封装类型	
64A	64- 引线，薄 (1.0 mm) 塑料鸥翼方形扁平式封装 (TQFP)
64M1	64- 焊垫 pad, 9 x 9 x 1.0 mm 大小，线距 0.50 mm，微导线框封装 (MLF)

封装信息

64A



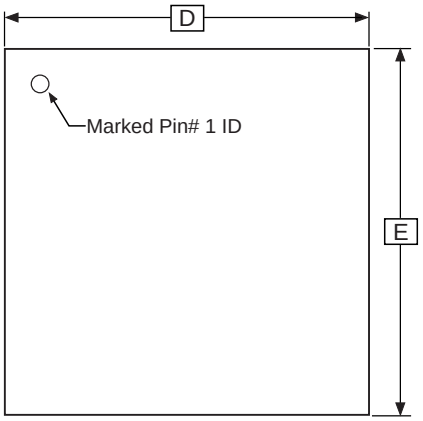
COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	1.20	
A1	0.05	–	0.15	
A2	0.95	1.00	1.05	
D	15.75	16.00	16.25	
D1	13.90	14.00	14.10	Note 2
E	15.75	16.00	16.25	
E1	13.90	14.00	14.10	Note 2
B	0.30	–	0.45	
C	0.09	–	0.20	
L	0.45	–	0.75	
e	0.80 TYP			

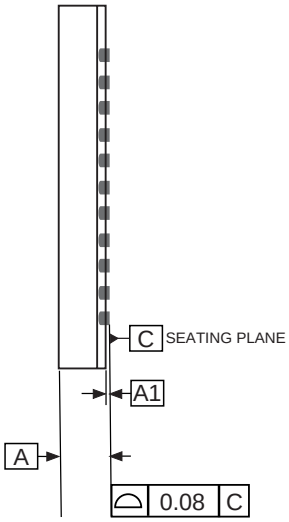
Notes:
1. This package conforms to JEDEC reference MS-026, Variation AEB.
2. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25 mm per side. Dimensions D1 and E1 are maximum plastic body size dimensions including mold mismatch.
3. Lead coplanarity is 0.10 mm maximum.

10/5/2001

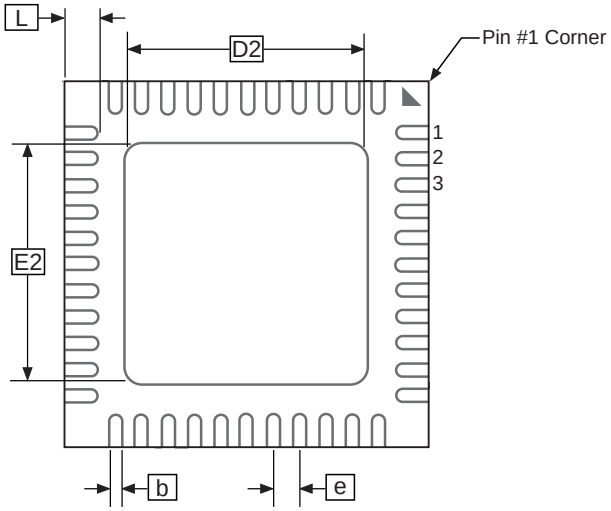
	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	**64A,** 64-lead, 14 x 14 mm Body Size, 1.0 mm Body Thickness, 0.8 mm Lead Pitch, Thin Profile Plastic Quad Flat Package (TQFP)	64A	B



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	–	0.02	0.05	
b	0.23	0.25	0.28	
D		9.00 BSC		
D2	5.20	5.40	5.60	
E		9.00 BSC		
E2	5.20	5.40	5.60	
e		0.50 BSC		
L	0.35	0.40	0.45	

Notes: 1. JEDEC Standard MO-220, Fig. 1, VMMD.

01/15/03

	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	**64M1**, 64-pad, 9 x 9 x 1.0 mm Body, Lead Pitch 0.50 mm Micro Lead Frame Package (MLF)	64M1	C

勘误表

ATmega169 Rev D

- **玻璃片中的高串联阻抗会导致 LCD 的段变暗**
- **IDCODE 屏蔽了从 TDI 输入的数据**

2. 玻璃片中的高串联阻抗会导致 LCD 的段变暗

一些显示模式中玻璃片的高串联阻抗 (>20 kΩ) 会导致 LCD 变暗。

解决方法

在每一个公共端和地之间加一个 1 nF(0.47 - 1.5 nF) 的电容。

1. IDCODE 屏蔽了从 TDI 输入的数据

JTAG指令IDCODE工作不正确。送往后续器件的数据在Update-DR时全被1所代替。

解决方法

– 如果 ATmega169 是扫描链中唯一的器件这个问题就不会出现。

– 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 ATmega169 前面器件的器件 ID 寄存器时对 ATmega169 发送 BYPASS 命令。

– 如果必须同时获取扫描链中所有器件的 ID，那么 ATmega169 应该是扫描链中的第一个器件。

ATmega169 Rev C

- **JTAGEN 编程后掉电模式的功耗将增加**
- **LCD 对比度控制**
- **一些数据组合会导致 LCD 变暗**
- **LCD 功耗**
- **IDCODE 屏蔽了从 TDI 输入的数据**

5. JTAGEN 编程后掉电模式的功耗增加

JTAG使能时TDO (PF6)的输入缓冲器使能，但上拉被禁止。这使引脚处于悬空状态。

解决方法

在 PF6 上加外部上拉电阻。

最终产品付运时不要编程 JTAGEN。

4. LCD 对比度控制

使用同步时钟 (片内时钟) 产生 LCD 的工作时钟时对比控制不能正常工作，此时片内时钟不小于 125 kHz。

解决方法

使用低的片内时钟 (32 kHz) 或给 LCDCAP 引脚提供一个外部电压。

3. 一些数据的组合会导致 LCD 的某些段变暗

当显示某个数据组合时所有与某个公共端相连的段都会变暗 (低对比度)。

解决方法

默认波形：如果有未使用的段管脚，给其中一个管脚连接一个 1 nF 的电容，并且通过给这一段写 "0" 来排除这个问题。

低功耗波形：给每个公共端加一个 1 nF 的电容。

2. LCD 功耗

当 V_{CC} 处于 VLCD -0.2V 到 VLCD + 0.4V 的范围内时， LCD 的电流将会达到预计的 3 倍。当器件工作于省电模式时这可能会带来问题。因为对于其他模式 LCD 的电流几乎可以忽略。

解决方法

无

1. IDCODE 屏蔽了从 TDI 输入的数据

JTAG命令IDCODE将会工作不正确。送往后续器件的数据在Update-DR时全被1所代替。

解决方法

– 如果 ATmega169 是扫描链中唯一的器件这个问题就不会出现。

– 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 ATmega169 前面器件的器件 ID 寄存器时对 ATmega169 发送 BYPASS 命令。

– 如果必须同时获取扫描链中所有器件的 ID，那么 ATmega169 应该是扫描链中的第一个器件。

ATmega169 Rev B

- **内部晶振工作在 4 MHz**
- **LCD 对比度电压不正确**
- **外部晶振不工作**
- **USART**
- **ADC 测量精度比指定的要低**
- **串行下载**
- **IDCODE 屏蔽了从 TDI 输入的数据**

7. 内部晶振工作在 4 MHz

内部晶振工作在 4 MHz 而不是指定的 8 MHz. 因此，所有的 Flash/EEPROM 的编程时间将会是指标的 2 倍。这包括片擦除、字节编程、页编程、熔丝编程、锁定位编程、CPU 对 EEPROM 写入、 Flash 自编程。

由于这个原因，rev-B 的样品在付运时 CKDIV8 熔丝位编程。

解决方法

如果需要工作在 8 MHz ，则要提供一个外部时钟 (这将会在 rev. C 中解决)

6. LCD 对比度电压不正确

LCD的对比度电压在1.8V 到3.1V 范围内是不正确的。当V_{CC} 在 1.8V 到3.1V之间时，LCD 的对比度电压将会降低大约 0.5V。这段时间内的功耗将会比预计的高。

解决方法

对比度将会不正确，但显示还是可见的。可部分的通过使用对比度控制寄存器来补偿 (这将会在 rev. C 中得到修正)。

5. 外部晶振不工作

在使用数据手册的设置时外部晶振不工作。

解决方法

使用其他时钟源 (这将会在 rev. C 中得到解决)。

其他的解决方法

在 XTAL1 加一个下拉电阻即可启动晶振。

4. USART

向 TXEN 写入 0 将使正在进行的传输立即中断。数据手册的描述为，应该在 USART 传输停止之后向 TXEN 写 0 来停止 USART。

解决方法

确保在 TXEN 被写 0 前传输已经结束 (这将会在 rev. C 得到修正)。

3. ADC 测量精度比指定的要低

ADC 将不会按预计的工作。测量结果中存在一个正的偏移量。

解决方法

rev. C 将修正这个问题。

2. 串行下载

当进入串行编程模式后，第二个字节不会像在串行编程算法中描述的那样回应。

解决方法

通过检查第三个字节是否有正确的回应来判断器件是否处于编程模式 (rev. C 将解决这个问题)。

1. IDCODE 屏蔽了从 TDI 输入的数据

JTAG指令IDCODE工作不正确。到后续器件的数据在Update-DR的过程中全被1所代替。

解决方法

– 如果 ATmega169 是扫描链中唯一的器件这个问题就不会出现。
– 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 ATmega169 前面器件的器件 ID 寄存器时对 ATmega169 发送 BYPASS 命令。
– 如果必须同时获取扫描链中所有器件的 ID，那么 ATmega169 应该是扫描链中的第一个器件。

ATmega169 的数据手册变更日志

从版本 Rev. 2514H-05/03 到版本 Rev. 2514I-09/03 的改动

1. 从数据手册中删除“ 预报” 部分
2. 删除了 P3 Figure 2 中的 AGND 并在 P23 Figure 11 中加入“ 系统时钟预分频器”。
3. 更新了 P37 Table 16 、 P39 Table 17 、 P41 Table 19 及 P67 Table 40 。
4. 将 P35 “JTAG 接口与片上调试系统 ” ” 重命名并修正为“ JTAG 接口及片上调试系统”。
5. 更新P89“T/C控制寄存器A – TCCR0A” 中的COM01:0为COM0A1:0及P128 “T/C控制寄存器 A– TCCR2A” 中的 COM21:0 为 COM2A1:0
6. 更新了 P218 “ 测试访问端口 – TAP” 中关于 JTAGEN 的部分。
7. 更新了 P226 中对 JTD 位的描述。
8. 在 P257 Table 119 中加入一条关于 JTAGEN 熔丝位的说明。
9. 更新了 P286 “ 电气特性 ” 中的最大绝对额定值及直流特性。
10. 更新了 P336 “ 勘误表 ” 并加入一条解决关于 JTAG 指令 IDCODE 的建议。

从版本 Rev. 2514G-04/03 到 版本 Rev. 2514H-05/03 的改动

1. 更了新 Figure 145, Figure 165, Figure 192 的排印错误。

从版本 Rev. 2514F-04/03 to 到版本 Rev. 2514G-04/03 的改动

1. 将文档中所有的 ICP 改为 ICP1
2. 去掉了注意事项 P25 “ 晶体振荡器工作模式 ”。
3. XTAL1/XTAL2 可以用作时钟振荡器引脚 ,P27“ 标定的片内 RC 振荡器 ” 章中有描述
4. 调整了 P31“ 转换时间 ” 的预分频器设置。
5. 更新了P286 “电气特性” 中的DC及ACD特性参数。删除了P37 Table 16 、P41 Table 19 及 P287 Table 133 中的 TBD 事项。
6. 更新了 P51 Figure 22 ， P55 Figure 25 ， P229 Figure 109 的 WRITE PINx REGISTER。
7. 更新了 P67 “ 端口 F 的第二功能 ” 中关于 JTAG 的内容。
8. 在 P170 “ 通用串行接口 – USI” 中用定时器 / 计数器 0 比较匹配替换了定时器 0 溢出。修改了 P176 “ 起始条件监测器 ” 及 P177 “USI 控制寄存器 – USICR” 。
9. 修改了 P184 “ 模数转换器 ” 及 P196 Table 88 中的特性。
10. 向 P247 Figure 117 及 P256 Table 118 中添加了注意事项。

从 版本 Rev. 2514D-01/03 到版本 Rev. 2514E-02/03 的改动

. 将 P1“ 性能 ” 中的部分改为关于 ATmega169 及 ATmega169L 的信息。

2. 删除 P2 Figure 1 , P3 Figure 2 , P5 “端口G (PG4..PG0)” , P69“端口G的第二功能” 及 P71 “I/O 端口寄存器的说明 ” 中所有关于 PG5 的内容。

3. 修改 P 256 Table 118, “ 熔丝位扩展字节 ,” 。

4. 添加 P339 “ATmega169 的数据手册变更日志 ” 的勘误表，包括 “ 重要数据表的改变” 。

5. 修改 P333 “ 产品信息 ” ,包括 ATmega169L 的新速率及新的 16 MHz ATmega169。

从版本 Rev. 2514C-11/02 到版本 Rev. 2514D-01/03 的改动

1. 在 P274 “ 通过 JTAG 接口进行编程 ” 中添加 TCK 频率限制。

2. 在 P283 “ 编程 Flash” 及 P284 “ 编程 EEPROM” 中添加芯片擦除作为第一步。

3. 添加 P54 “ 未连接引脚的处理 ” 。

4. 在 P35 “JTAG 接口与片上调试系统 ” 中添加如何禁止 OCD 系统。

5. 修正了中断地址。 ADC 与 ANA_COMP 作了交换。

6. 改进 P288 “SPI 时序特性 ” 中的表并删除 P274 “SPI 串行编程特性参数 ” 中的表。

7. 将 P250 “ 执行页写 ” 中的未用 Z 指针位的 “ 可忽略” 改为 “ 必须写为 0” 。

8. 将 P212 中对 LCDCRA 寄存器描述中的 “ LCD 帧完成” 改为 “ LCD 帧起始” 。

9. 在 USI 代码例子中将 OUT 改为 STS ,IN 改为 LDS ,并改正 f_{SCKmax}。USI I/O 在扩展的 I/O空间 ,因此不可以使用IN及OUT。执行LDS及STS时需要超过一个周期的时间 ,因此要进行修正。

10. 删除P230 Table 103 中的TOSKON及TOSCK ,P232 Figure 114 及P233 Table 105 中的 g10 及 g20，因为这些信号在边界扫描中不存在。

11. 在器件特性表中将 4MIPS 及 4MHz 改为 16 MIPS 及 16 MHz。

12. 在 P5 “ 引脚说明 ” 之后 P6 “AVCC” 中的端口 A 改为端口 F。

13. 将 P165 “ 波特率设置的例子 ” 中的 230.4 Mbps 改为 230.4 kbps。

14. 修正了 P 165 Table 78, “UCPOL 设置 ,” 中的 XCK 下降及上升沿。

15. 删除多功能振荡器应用笔记，因为没有此笔记。

16. 将 P19 Table 1 中的标准 RC 振荡器周期由 8,448 改为 67,584。

17. 主定时器 1 的各种修正。

18. 为定时器 0 及定时器 2 添加 PWM 对称性的信息。

19. 修正 DIDR0 及 DIDR1 的内容。

20. 为使得 LCDDR 寄存器的名字唯一 ,在 2 个已有数字前加上 COM 序号 ,如 SEG304。

21. 在 P132 “定时器/计数器2的异步操作” 中将扩展的Standby模式改为ADC噪声抑制模式。

22. 添加一条关于端口 B 的驱动能力比其他端口好的说明。作为其直接后果，P285 中的 “ 直流特性参数” 进行了修正。

23. 在 P250 “写临时缓冲区(页加载)” 之后添加有关在SPM页加载过程中对EEPROM进行写操作的注意事项。

24. 删除 ADHSM

25. 更新 了 P334 “ 封装信息 ” 。

从 版本 Rev. 2514B-09/02 到版本 Rev. 2514C-11/02 的改动

1. 添加 P336 “ 勘误表 ” 。
2. 在 P333 “ 产品信息 ” ， P334 “ 封装信息 ” 中添加 64 引脚的 MLF 信息。
3. 修改 P334 “ 封装信息 ” 中的温度范围并删除工业级定货代码。

从 版本 Rev. 2514A-08/02 到版本 Rev. 2514B-09/02 的改动

1. 将 Flash 的擦除寿命改为 10,000 次。