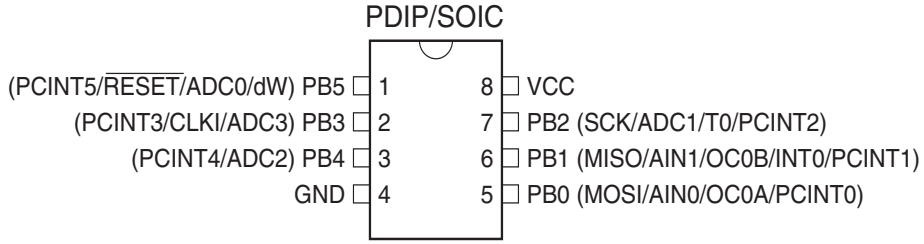
产品特性

- 高性能、低功耗的 8 位 AVR® 微处理器
- 高级 RISC 结构
  - 120 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 20 MHz 时性能高达 20 MIPS
- 非易失性程序和数据存储器
  - 1K 字节的系统内可编程 Flash
    擦写寿命 : 10,000 次
  - 64 字节的系统内可编程 EEPROM
    擦写寿命 : 100,000 次
  - 64 字节的片内 SRAM
  - 可以对锁定位进行编程以及实现 EEPROM 数据的加密
- 外设特点
  - 一个具有独立预分频器的 8 位定时器 / 计数器及两条 PWM 通道
  - 含有片内参考电压的 4 路 10 位 ADC
  - 具有独立片内振荡器的可编程看门狗定时器
  - 片内模拟比较器
- 特殊的处理器特点
  - 片内调试系统
  - 通过 SPI 端口在系统内可编程
  - 片内 / 片外中断源
  - 低功耗空闲模式、噪声抑制模式、省电模式
  - 增强型上电复位
  - 可编程的掉电检测
  - 片内标定振荡器
- I/O 和封装
  - 8 引脚 PDIP/SOIC: 6 可编程 I/O 线
- 工作电压 :
  - ATtiny13V : 1.8 - 5.5V
  - ATtiny13 : 2.7 - 5.5V
- 速度等级
  - ATtiny13V: 0 - 4 MHz @ 1.8 - 5.5V, 0 - 10 MHz @ 2.7 - 5.5V
  - ATtiny13: 0 - 10 MHz @ 2.7 - 5.5V, 0 - 20 MHz @ 4.5 - 5.5V
- 工业级温度范围
- 低功耗
  - 正常模式 :
    1 MHz, 1.8V: 240µA
  - 掉电模式 :
    < 0.1µA at 1.8V

引脚配置

Figure 1. ATtiny13 芯片引脚



具有 1KB 系统内可编程 Flash 的 8 位 AVR® 微控制器

ATtiny13

初稿

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

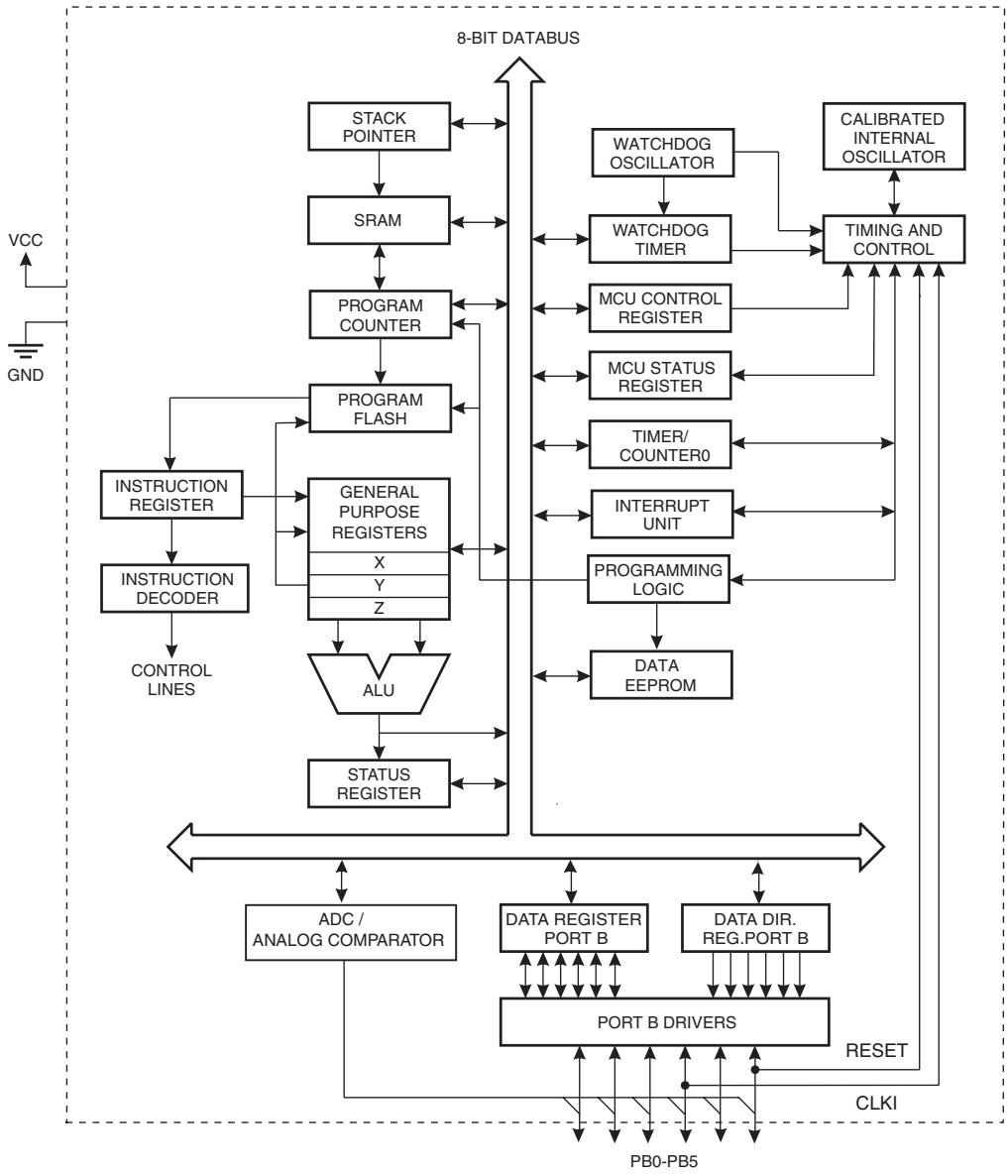
Rev. 2535D–AVR–04/04

综述

ATtiny13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， ATtiny13 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. ATtiny13 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。

ATtiny13 有 1K 字节 Flash ,64 字节 EEPROM ,64 字节 SRAM ,6 个通用 I/O 口线，32 个通用工作寄存器， 1 个具有比较模式的 8 位定时器 / 计数器，片内 / 外中断， 4 路 10 位 ADC ,具有片内振荡器的可编程看门狗定时器 ,以及三种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、 ADC、模拟比较器以及中断系统继续工作；掉电模式时保存寄存器中值，停止除中断和硬件复位之外所有功能工作；ADC 噪声抑制模式时终止 CPU 及 ADC 以外所有 I/O 模块的工作以降低 ADC 转换噪声。

本芯片是以 Atmel 高密度非易失性存储器技术生产的。通过 SPI 串行接口可对程序存储器进行系统内编程。

ATtiny13 AVR 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 B (PB5..PB0) 端口 B 为 6 位双向 I/O 口 ,具有可编程的内部上拉电阻。其输出缓冲器具有对称的驱动特性，可以输出和吸收大电流。作为输入使用时，若内部上拉电阻使能，端口被外部电路拉低时将输出电流。在复位过程中，即使系统时钟还未起振，端口 B 处于高阻状态。

端口 B 也可以用做其他不同的特殊功能，请参见 P48。

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 P30Table 12。持续时间小于门限时间的脉冲不能保证可靠复位。

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。

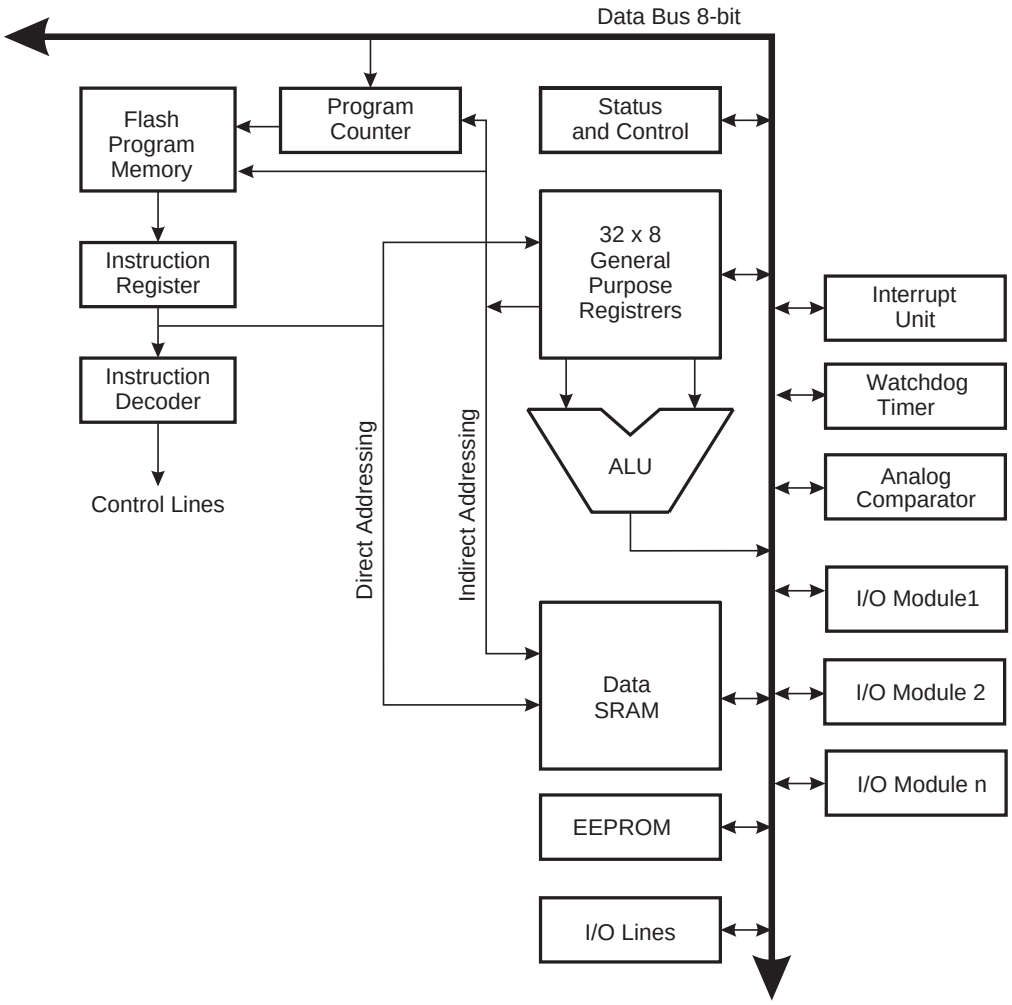
AVR CPU 内核

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。

在中断和调用子程序时返回地址的程序计数器 (PC) 保存于堆栈之中。堆栈位于通用数据 SRAM，因此其深度仅受限于 SRAM 的大小。在复位例程里用户首先要初始化堆栈指针 SP。这个指针位于 I/O 空间，可以进行读写访问。数据 SRAM 可以通过 5 种不同的寻址模式进行访问。

AVR 存储器空间为线性的平面结构。

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

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

ALU - 算术逻辑单元

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

状态寄存器

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

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

AVR 中断寄存器 SREG 定义如下：

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

- **Bit 7 – I: 全局中断使能**

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

- **Bit 6 – T: 位拷贝存储**

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

- **Bit 5 – H: 半进位标志**

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

- **Bit 4 – S: 符号位 , S = N ⊕ V**

S 为负数标志 N 与 2 的补码溢出标志 V 的异或。详见指令集的说明。

- **Bit 3 – V: 2 的补码溢出标志**

支持 2 的补码运算。详见指令集的说明。

- **Bit 2 – N: 负数标志**

表明算术或逻辑操作结果为负。详见指令集的说明。

- **Bit 1 – Z: 零标志**

表明算术或逻辑操作结果为零。详见指令集的说明。

- **Bit 0 – C: 进位标志**

表明算术或逻辑操作发生了进位。详见指令集的说明。

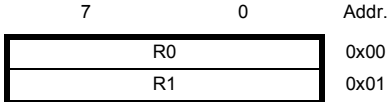
通用寄存器文件

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

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

Figure 4 为 CPU 32 个通用工作寄存器的结构。

Figure 4. AVR CPU 通用工作寄存器



	R2	0x02	
	…		
	R13	0x0D	
通用	R14	0x0E	
工作	R15	0x0F	
寄存器	R16	0x10	
	R17	0x11	
	…		
	R26	0x1A	X 寄存器，低字节
	R27	0x1B	X 寄存器，高字节
	R28	0x1C	Y 寄存器，低字节
	R29	0x1D	Y 寄存器，高字节
	R30	0x1E	Z 寄存器，低字节
	R31	0x1F	Z 寄存器，高字节

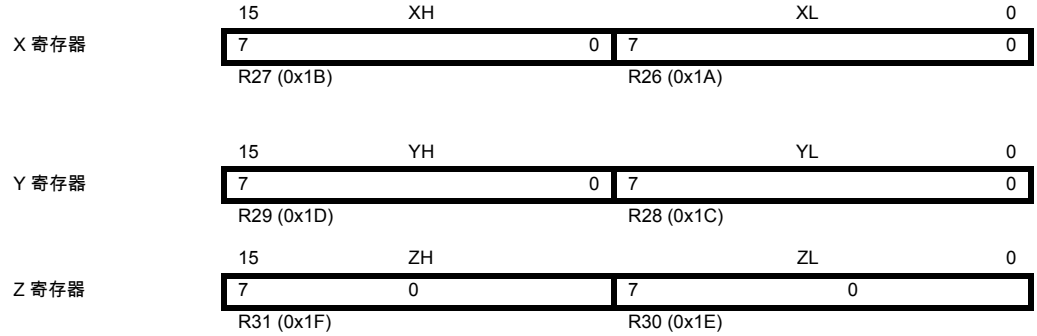
大多数操作寄存器文件的指令都可以直接访问所有的寄存器，而且多数这样的指令的执行时间为单个时钟周期。

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

X、Y、Z 寄存器

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

Figure 5. X、 Y、 Z 寄存器



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

堆栈指针

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

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

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

Bit	15	14	13	12	11	10	9	8	
	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	
初始值	1	0	0	1	1	1	1	1	

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

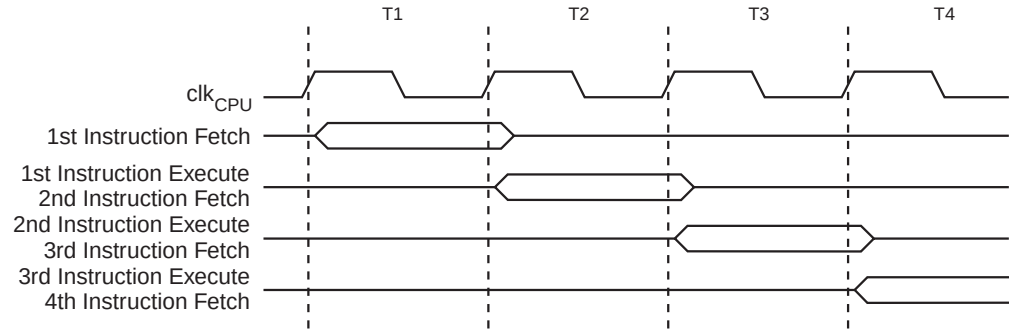
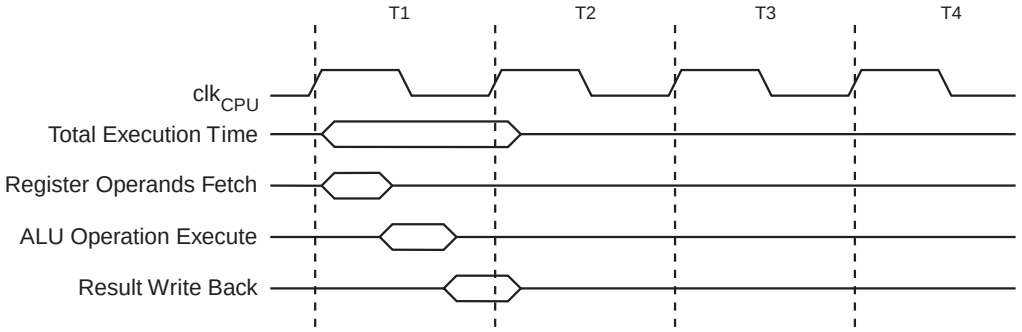


Figure 7 演示的是寄存器文件内部访问时序。在一个时钟周期里，ALU 可以同时对两个寄存器操作数进行操作，同时将结果保存到目的寄存器中去。

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。

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 P40“ 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。

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 值
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 值 */
/* 禁止中断 */
__disable_interrupt();
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 ATtiny13 的存储器

本节讲述 ATtiny13 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， ATtiny13 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

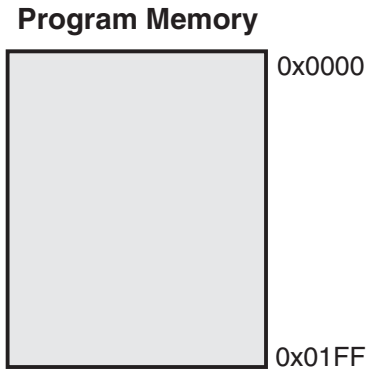
ATtiny13具有1K字节的在线编程Flash ,用于存放程序指令代码。因为所有的AVR指令为 16 位或 32 位，故而 Flash 组织成 512 x 16 位的形式。

Flash存储器至少可以擦写10,000次。ATtiny13的程序计数器(PC)为9位 ,因此可以寻址512字的程序存储器空间。P97“存储器编程” 详述了如何用SPI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 P9“ 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 ATtiny13 SRAM 空间的组织结构。

前 160 个数据存储器包括了寄存器文件、 I/O 存储器及内部数据 SRAM。起始的 32 个地址为寄存器文件与 64 个标准 I/O 存储器，接着是 64 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。

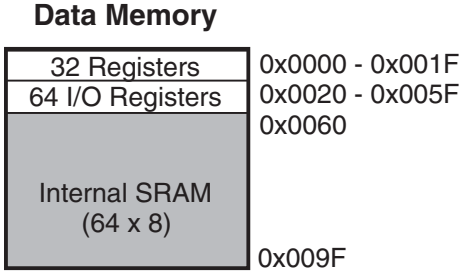
直接寻址范围可达整个数据区。

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。

ATtiny13的全部32个通用寄存器、64个I/O寄存器及64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 P6“ 通用寄存器文件 ”。

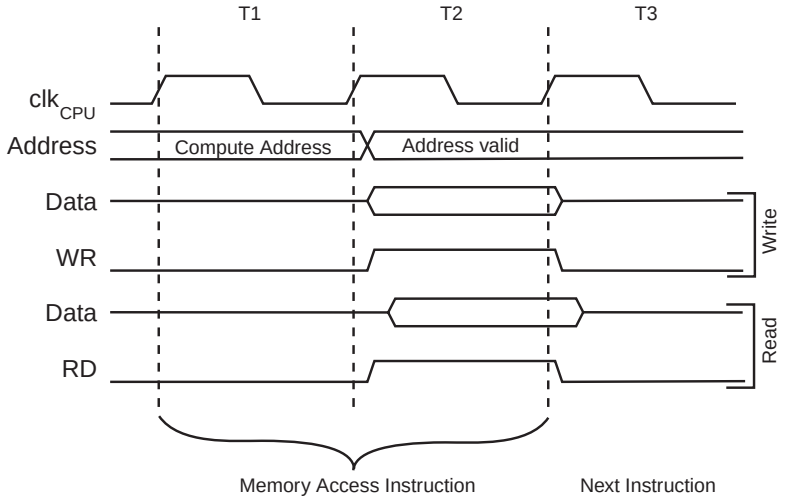
Figure 9. 数据存储器映像



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

ATtiny13 包含 64 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。详见 P100 中 EEPROM 的串行数据下载。

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 P18“ 防止 EEPROM 数据丢失 ” 以避免出现 EEPROM 数据丢失的问题。

为了防止无意识的 EEPROM 写操作，需要执行一个特定的写时序。具体参看 P15“ 基本字节编程 ” 及 P16“ 分离字节编程 ”。

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 - EEARL

Bit	7	6	5	4	3	2	1	0	
	–	–	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
读 / 写	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	X	X	X	X	X	X	

- **Bits 7..6 – Res: 保留**

保留位，读操作返回值为零。

- **Bits 5..0 – EEAR5..0: EEPROM 地址**

EEPROM地址寄存器EEARL指定了64字节的EEPROM空间。EEPROM地址是线性的，从 0 到 63。EEAR 的初始值没有定义。在访问 EEPROM 之前必须为其赋予正确的数据。

EEPROM 数据寄存器 - EEDR

Bit	7	6	5	4	3	2	1	0	
	EEDR7	EEDR6	EEDR5	EEDR4	EEDR3	EEDR2	EEDR1	EEDR0	EEDR
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	

- **Bits 7..0 – EEDR7..0: EEPROM 数据**

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

EEPROM 控制寄存器 - EECR

Bit	7	6	5	4	3	2	1	0	
	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	EECR
读 / 写	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	X	X	0	0	X	0	

- **Bit 7 – Res: 保留**

保留位，读操作返回值为零。读完后，屏蔽该位。

- **Bit 6 – Res: 保留**

保留位，读操作返回值为零。

- **Bits 5, 4 – EEPM1 与 EEPM0: EEPROM 编程模式位**

设置编程模式位定义当对 EEPE 写入时触发哪种编程方式。可能在一个时钟周期中数据编程 (擦除旧值写入新值) 或将擦除与写操作分为两步。不同模式的编程时间见 Table 1。当 EEPE 置位，对 EEPMn 的写操作将忽略。复位时，除非 EEPROM 处于编程状态，否则 EEPMn 位将复位为 0b00。

Table 1. EEPROM 模式位

EEPM1	EEPM0	编程时间	操作
0	0	3.4 ms	擦除与写入操作在一个时钟周期中完成
0	1	1.8 ms	只擦除
1	0	1.8 ms	只写入
1	1	–	保留

- **Bit 3 – EERIE: EEPROM 准备好中断使能**

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

- **Bit 2 – EEMPE: EEPROM 主机编程使能**

EEMPE 位决定 EEPE 写入 "1" 是否有效。

当 EEMPE 为 "1" 时，在四个时钟周期内设置 EEPE 将会在 EEPROM 指定的位置编程；若 EEMPE 为 "0“ , 设置 EEPE 无效。当 EEMPE 由软件写入 "1" , 则在四个时钟周期后由硬件清零。

- **Bit 1 – EEPE: EEPROM 编程使能**

EEPE 为 EEPROM 的编程使能信号。当 EEPE 为 "1" , 通过 EEPMn 位的设置，将会对 EEPROM编程。在EEPE写入逻辑"1"前 ,EEMPE位必须写入"1" ,否则不会出现EEPROM 写操作。当写访问时间结束， EEPE 位由硬件清零。当 EEPE 置位，CPU 在执行指令前终止两个时钟周期。

- **Bit 0 – EERE: EEPROM 读使能**

EERE为EEPROM读操作的使能信号。当EEPROM地址设置好之后 ,需置位EERE以便将数据读入 EEARL。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 EEPE。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 EEARL。

基本字节编程

使用基本字节编程是最简单的模式。当对 EEPROM 写入一个字节，用户必须将地址写入 EEARL 寄存器 ,将数据写入 EEDR 寄存器。若 EEPMn 位为零 ,对 EEPE 的写操作 (在对 EEMPE 写完后的四个时钟周期内) 将触发擦除 / 写入操作。擦除与写入操作在一个时钟周期内完成，整个编程时间见 Table 1。 EEPE 位会保持置位，直到擦除与写入操作完成。而当芯片处于编程状态时，不会进行其他 EEPROM 操作。

分离字节编程

可以将擦除与写入操作分为两个周期。若系统需要对一些有限的时间缩短访问时间 (尤其若电源电压下降) 该方式有效。使用该方式时，必须在写入操作前先进行擦除操作。但由于擦除与写入操作是分离的，有可能当系统允许进行时间临界操作时 (尤其在掉电后) 进行擦除操作。

擦除

擦除一个字节 ,地址必须写入 EEARL。若 EEPMn 为 0b01 ,对 EEPE 写入 (在对 EEMPE 写完后的四个时钟周期内) 将只触发擦除操作 (编程时间见 Table 1)。 EEPE 位会保持到擦除操作完成。而当芯片处于编程状态时，不会进行其他 EEPROM 操作。

写入

写入时 ,用户必须将地址写入 EEARL ,将数据写入 EEDR。若 EEPMn 为 0b10 ,对 EEPE 写入 (在对 EEMPE 写完后的四个时钟周期内) 将只触发写入操作 (编程时间见 Table 1)。 EEPE 位会保持到擦除操作完成。若在写入前数据没有擦除，则认为写入数据丢失。当芯片处于编程状态时，不会进行其他 EEPROM 操作。

EEPROM访问使用标定振荡器定时。振荡器频率见P22“振荡器标定寄存器 – OSCCAL”。

下面的代码分别用汇编和 C 函数说明如何实现 EEPROM 的擦除、写入或基本写入。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_write:
  ; 等待上一次写操作结束
  sbic EECR,EEPE
  rjmp EEPROM_write
  ; 设置编程模式
  ldi  r16, (0<<EEPM1)|(0<<EEPM0)
  out  EECR, r16
  ; 设置地址寄存器 r17
  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 char ucAddress, unsigned char ucData)
{
  /* 等待上一次写操作结束 */
  while(EECR & (1<<EEPE))
    ;
  /* 设置编程模式 */
  EECR = (0<<EEPM1)|(0>>EEPM0)
  /* 设置地址与数据寄存器 */
  EEARL = ucAddress;
  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,EEPE
  rjmp EEPROM_read
  ; 设置地址寄存器 r17
  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 char ucAddress)
{
  /* 等待上一次写操作结束 */
  while(EECR & (1<<EEPE))
    ;
  /* 设置地址寄存器 */
  EEARL = ucAddress;
  /* 设置 EERE 以启动读操作 */
  EECR |= (1<<EERE);
  /* 自数据寄存器返回数据 */
  return EEDR;
}
```

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

ATtiny13 的 I/O 空间定义见 P150“ 寄存器概述 ”。

ATtiny13 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。

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

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

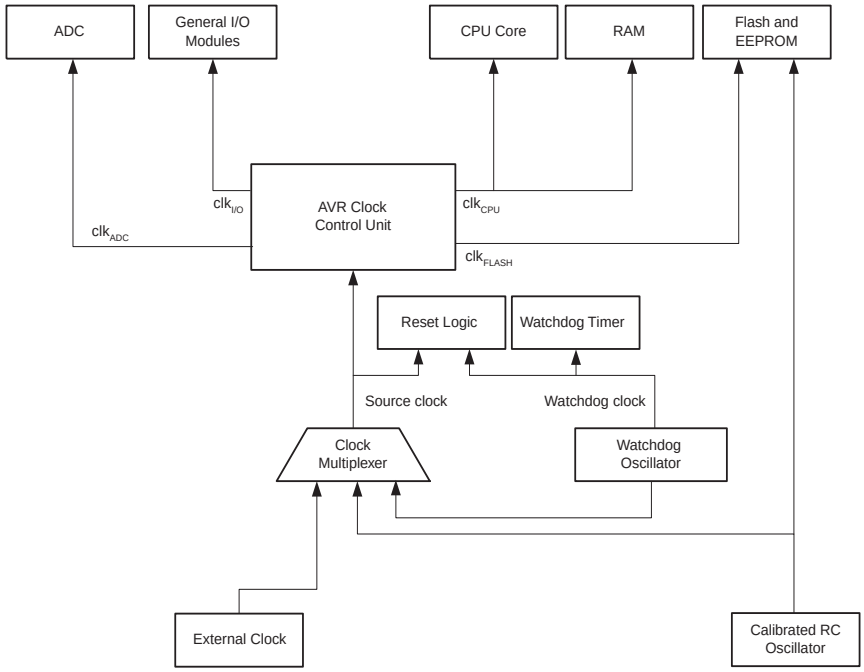
I/O 和外设控制寄存器在后续其他章节进行介绍。

系统时钟及时钟选项

时钟系统及其分布

Figure 11为AVR的主要时钟系统及其分布。这些时钟并不需要同时工作。为了降低功耗，可以通过使用不同的睡眠模式来禁止无需工作的模块的时钟，详见 P26“ 电源管理及睡眠模式 ”。时钟系统详见 Figure 11。

Figure 11. 时钟分布



CPU 时钟 - clk_{CPU}

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

I/O 时钟 - $clk_{I/O}$

I/O 时钟用于主要的 I/O 模块，如定时器 / 计数器。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。

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。

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

ATtiny13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	CKSEL1..0
标定的内部 RC 振荡器	01, 10
外部时钟	00
128 kHz 内部振荡器	11

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

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

Table 3. 看门狗振荡器周期数

典型的溢出时间	时钟周期数
4 ms	512
64 ms	8K (8,192)

缺省时钟源

器件出厂时 CKSEL = “10”， SUT = “10”，且 CKDIV8 编程。这个缺省设置的时钟源是9.6MHz 的内部 RC 振荡器，初始系统时钟预分频为 8，启动时间为最长。这种设置保证用户可以通过 ISP 或并行编程器得到所需的时钟源。

标定的片内 RC 振荡器

标定的片内 RC 振荡器提供了固定的 9.6 MHz 或 4.8 MHz 的时钟。这些频率都是 3V、25°C 下的标称数值。若频率超出器件标称值 ,必须对 CKDIV8 熔丝位编程 ,以在启动阶段对内部频率 8 分频，详见 P24“ 系统时钟预分频器 ”。这个时钟也可以作为系统时钟，只要按照 Table 4 对熔丝位 CKSEL 进行编程即可。选择这个时钟之后就无需外部器件了。复位时硬件将标定字节加载到 OSCCAL 寄存器，自动完成对 RC 振荡器的标定。在 3V、25°C 时 ,这种标定可以提供标称频率 ± 10% 的精度。使用在 www.atmel.com/avr 中所述的标定方法，可能会在任何电压及任何温度下使精度达到 ± 3%。当使用这个振荡器作为系统时钟时 ,看门狗仍然使用自己的看门狗定时器作为溢出复位的依据。更多的有关标定数据的信息请参见 P99“ 校准字节 ”。

Table 4. 片内标定的 RC 振荡器工作模式

CKSEL1..0	标称频率
10(1)	9.6 MHz
01	4.8 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 5 所示。 .

Table 5. 内部标定 RC 振荡器的启动时间

SUT1..0	掉电模式的启动时间	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 ms	电源快速上升
10(1)	6 CK	14CK + 64 ms	电源缓慢上升
11	保留		

Note: 1. 出厂时的设置。

振荡器标定寄存器 - OSCCAL

Bit	7	6	5	4	3	2	1	0	
	–	CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0	OSCCAL
读 / 写	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	标定数据							

- **Bit 7 – Res: 保留**

保留位，读操作返回值为零。

- **Bits 6..0 – CAL6..0: 振荡器标定值**

将标定数据写入这个地址可以对内部振荡器进行调节以消除由于生产工艺所带来的振荡器频率偏差。这在复位时自动完成。当 OSCCAL 为零时振荡器以最低频率工作。当对其写如不为零的数据时内部振荡器的频率将增长。写入 0x7F 即得到最高频率。标定的振荡器用来为访问 EEPROM 和 Flash 定时。有写 EEPROM 和 Flash 的操作时不要将频率标定到超过标称频率的 10%，否则写操作有可能失败。要注意振荡器只对 9.6 MHz 和 4.8 MHz 这两种频率进行了标定，其他频率则无法保证。

为保证 MCU 稳定工作，当标定内部 RC 振荡器时避免大幅度改变标称值。工作频率突变超过 2% 将会产生异常现象。每次对 OSCCAL 寄存器中值的改变不应超过 0x20。

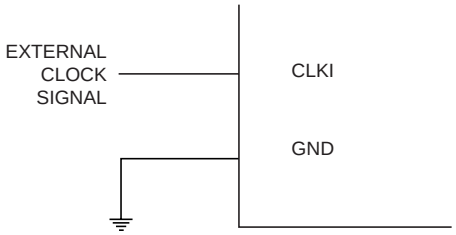
Table 6. 内部 RC 振荡器频率范围

OSCCAL 数值	最小频率，标称频率的百分比	最大频率，标称频率的百分比
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， CLKI 必须如 Figure 12 所示的进行连接。同时，熔丝位 CKSEL 必须编程为 “00”。

Figure 12. 外部时钟配置图



选择了这个振荡器之后，启动时间由熔丝位 SUT 确定，如 Table 7 所示。

Table 7. 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	推荐用法
00	6 CK	14CK	BOD 使能
01	6 CK	14CK + 4 ms	电源快速上升
10	6 CK	14CK + 64 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。

注意，系统时钟预分频器可用来实现内部时钟频率运行时间改变且保证稳定工作，详见 P24“ 系统时钟预分频器 ”。

128 kHz 片内振荡器

128 kHz片内振荡器为提供时钟频率为128 kHz的低功耗振荡器。该频率为在3V、25°C下的标定值。通过将 CKSEL 熔丝位编程为 “11”，该时钟作为系统时钟。

当选择该时钟源，启动时间由 Table 8 中所示的 SUT 熔丝位决定。

Table 8. 128 kHz 片内振荡器启动时间

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	推荐用法
00	6 CK	14CK	BOD 使能
01	6 CK	14CK + 4 ms	电源快速上升
10	6 CK	14CK + 64 ms	电源缓慢上升
11	保留		

系统时钟预分频器

ATtiny13 系统时钟可通过设置时钟预分频寄存器 CLKPR 来分频。该特性可用来降低功耗。该分频器对所有时钟源都有效，且可影响 CPU 时钟频率及所有同步外设。 $clk_{I/O}$、clk_{ADC}、 clk_{CPU} 及 clk_{FLASH} 分频因子见 Table 9。

时钟预分频寄存器 - CLKPR

Bit	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				

- **Bit 7 – CLKPCE: 时钟预分频器变化使能**

CLKPCE 位必须置"1”使能 CLKPS位。只有当CLKPR寄存器的其他位同时写"0”时，CLKPCE 位改变。CLKPCE 在写入四个周期后或当 CLKPS 位写入后由硬件清零。在暂停周期中重新写 CLKPCE 位，既不扩展暂停周期，也不清除 CLKPCE 位。

- **Bits 6..4 – Res: 保留**

保留位，读操作返回值为零。

- **Bits 3..0 – CLKPS3..0: 时钟预分频器选择位 3 - 0**

这几位定义所选时钟源与内部系统时钟所分频因子。这几位写入运行时间来改变时钟频率以适应运行需要。当作为 MCU 主时钟输入分频器，使用分频因子时，所有同步外设速度将会下降。分频因子见 Table 9。

为避免时钟频率的无意改变，对 CLKPS 位的写入必须按照如下步骤进行：

1. 将 CLKPCE 位写 "1”，而 CLKPR 寄存器的其他位写 "0”。
2. 在四个时钟周期内，将期望值写入 CLKPS，并在 CLKPCE 位写 "0”。

在改变预分频器设置时必须禁止中断，以保证在写入过程中不会出现中断。

CKDIV8熔丝位决定CLKPS位的初始值。若CKDIV8未编程，CLKPS位复位为“0000”；若CKDIV8 已编程，CLKPS 位复位为 “0011”，给出启动时分频因子为 8。若所选时钟源频率大于当前工作状态下器件最大频率时，应利用该特性分频。注意， CLKPS 位写入值不受

CKDIV8 熔丝位设置影响。若所选时钟源频率大于当前工作状态下器件最大频率，应用程序必须保证选择一个足够大的分频因子。芯片出厂时 CKDIV8 熔丝位已编程。

Table 9. 时钟预分频器选择

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
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

转换时间

当预分频器转换时，系统时钟预分频器保证时钟系统中不会出现毛刺，且频率低于转换前后的频率。

脉动计数器使预分频器在未分频时钟频率下运行，这可能比 CPU 时钟频率快。因此即使预分频器可读，我们也无法确定其状态，所以我们也无法得到准确的转换时间。

CLKPS 值的写入时间介于 T1 + T2 与 T1 + 2*T2 之间。在此间隔中，产生 2 个时钟边沿。其中 T1 为前一个时钟周期， T2 为新设置后相应的时钟周期。

电源管理及睡眠模式

AVR 微控制器的高性能与针对工业级的有效代码使其成为低功耗器件的最佳选择。

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

进入睡眠模式的条件是置位寄存器 MCUCR 的 SE，然后执行 SLEEP 指令。具体哪一种模式 (空闲模式、 ADC 噪声抑制模式、掉电模式) 由 MCUCR 的 SM1 和 SM0 决定，如 Table 10 所示。使能的中断可以将进入睡眠模式的 MCU 唤醒。经过启动时间，外加 4 个时钟周期后， MCU 就可以运行中断例程了。然后返回到 SLEEP 的下一条指令。唤醒时不会改变寄存器文件和 SRAM 的内容。如果在睡眠过程中发生了复位，则 MCU 唤醒后从中断向量开始执行。

P20Figure 11 介绍了 ATtiny13 不同的时钟系统及其分布。此图在选择合适的睡眠模式时非常有用。

MCU 控制寄存器 - MCUCR

MCU 控制寄存器包含了电源管理的控制位。

Bit	7	6	5	4	3	2	1	0	
	–	PUD	SE	SM1	SM0	—	ISC01	ISC00	MCUCR
读 / 写	R	R/W	R/W	R/W	R/W	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 5 – SE: 休眠使能**

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

- **Bits 4, 3 – SM1..0: 休眠模式选择位 2..0**

如 Table 10 所示，这些位用于选择具体的休眠模式。

Table 10. 休眠模式选择

SM1	SM0	休眠模式
0	0	空闲模式
0	1	ADC 噪声抑制模式
1	0	掉电模式
1	1	保留

- **Bit 2 – Res: 保留**

保留位，读操作返回值为零。

空闲模式

当 SM1..0 为 00 时， SLEEP 指令将使 MCU 进入空闲模式。在此模式下， CPU 停止运行，而模拟比较器、 ADC、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 SM1..0 为 01 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、外部中断和看门狗继续工作。这个睡眠模式只停止了 $clk_{I/O}$、 clk_{CPU} 和 clk_{FLASH}，其他时钟则继续工作。

此模式提高了 ADC 的噪声环境，使得转换精度更高。 ADC 使能的时候，进入此模式将自动启动一次 AD 转换。 ADC 转换结束中断、外部复位、看门狗复位、 BOD 复位、SPM/EEPROM准备好中断、外部中断INT0 或引脚变化中断可以将MCU从ADC噪声抑制模式唤醒。

掉电模式

当 SM1..0 为 10 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、外部电平中断 INT0 或引脚变化中断可以使 MCU 脱离掉电模式。这个睡眠模式停止了所有的时钟，只有异步模块可以继续工作。

当使用外部电平中断方式将 MCU 从掉电模式唤醒时，必须保持外部电平一定的时间。具体请参见 P51“ 外部中断 ”。

Table 11. 在不同睡眠模式下活动的时钟以及唤醒源

	工作的时钟				振荡器	唤醒源				
睡眠模式	**clk_{CPU}**	**clk_{FLASH}**	**clk_{IO}**	**clk_{ADC}**	使能的主时钟	**INT0 与引脚变化**	**SPM/ EEPROM 准备好**	**ADC**	**Other I/O**	**看门狗中断**
空闲模式			X	X	X	X	X	X	X	X
ADC 噪声抑制模式				X	X	X(1)	X	X		X
掉电模式						X(1)				X

Note: 1. INT0 只有电平中断

最小化功耗

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

模数转换器

使能时， ADC 在睡眠模式下继续工作。为了降低功耗，在进入睡眠模式之前需要禁止 ADC。重新启动后的第一次转换为扩展的转换。具体请参照 P76“ 模数转换器 ”。

模拟比较器

在空闲模式时，如果没有使用模拟比较器，可以将其关闭。在 ADC 噪声抑制模式下也是如此。在其他睡眠模式模拟比较器是自动关闭的。如果模拟比较器使用了内部电压基准源，则不论在什么睡眠模式下都需要关闭它。否则内部电压基准源将一直使能。请参见 P73“ 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 BODLEVEL 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 P32“ 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 P34“ 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 P40“ 中断 ” 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 P45“ 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。

对于模拟输入引脚，数字输入缓冲应始终禁用。即使在工作模式下，在输入引脚的接近 $V_{CC}/2$ 的模拟信号电平会带来明显的电流 。数字输入缓冲可通过写 DIDR0 来禁用，参见 P75“ 数字输入禁用寄存器 0 – DIDR0” 。

系统控制和复位

复位 AVR

复位时所有的 I/O 寄存器都被设置为初始值，程序从复位向量处开始执行。复位向量处的指令必须是绝对跳转 JMP 指令，以使程序跳转到复位处理例程。如果程序永远不利用中断功能，中断向量可以由一般的程序代码所覆盖。Figure 13 为复位逻辑的电路图。Table 12 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 P20“ 时钟源 ” 。

复位源

The ATtiny13 有 4 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。

Figure 13. 复位逻辑

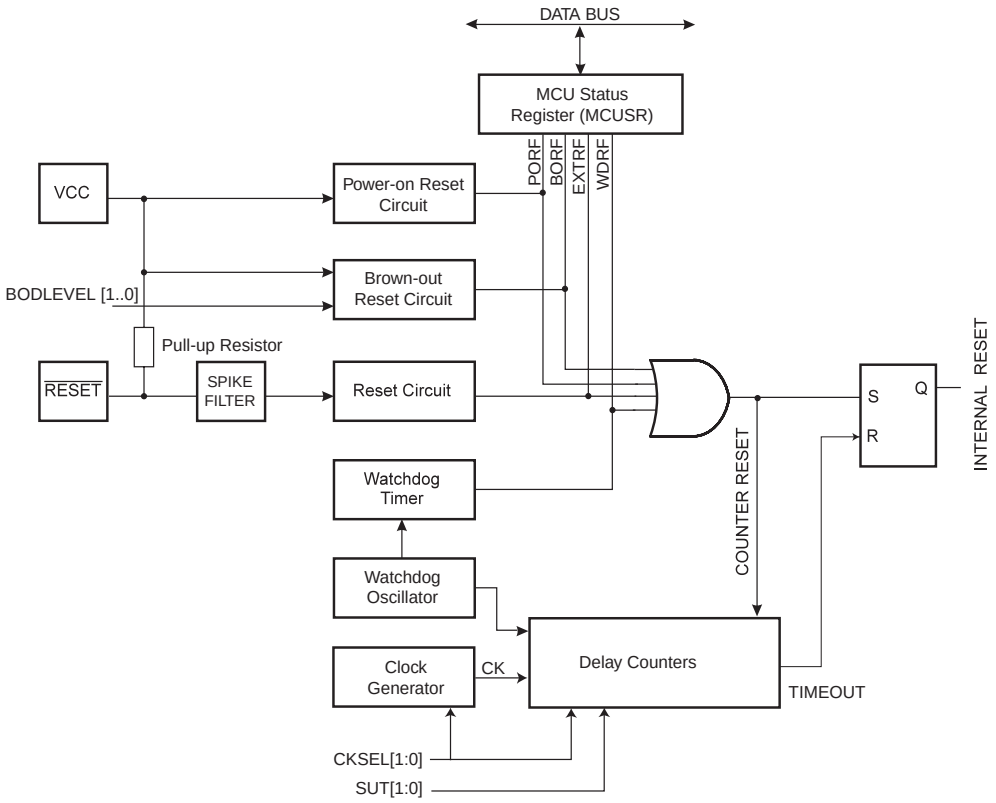


Table 12. 复位特性 (1)

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 - 85°C		1.2		V
	上电复位门限电压 (电压由高到低跌落)(2)	T_A = -40 - 85°C		1.1		V
V_{RST}	$\overline{\text{RESET}}$ 门限电压	V_{CC} = 1.8V - 5.5V	0.1 V_{CC}		0.9 V_{CC}	V
t_{RST}	$\overline{\text{RESET}}$ 最小脉冲宽度	V_{CC} = 1.8V - 5.5V			2.5	µs

Notes: 1. 这些值仅作为参考，实际值待测试。
2. 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 12。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 14. MCU 启动过程，$\overline{\text{RESET}}$ 连接到 V_{CC}

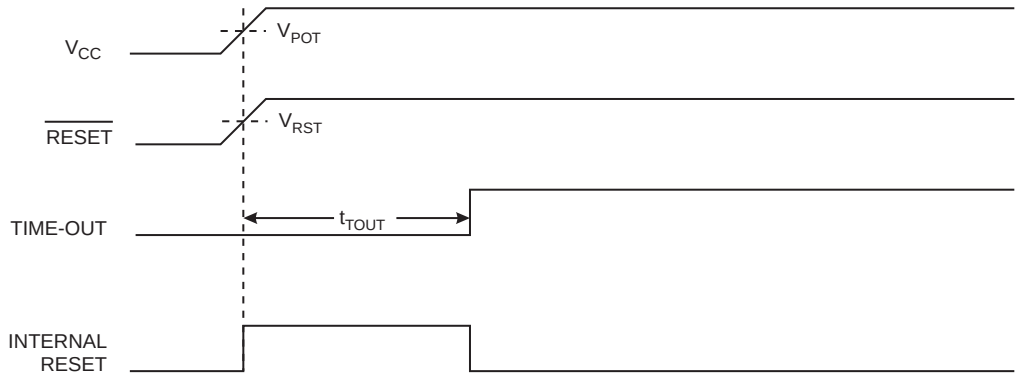
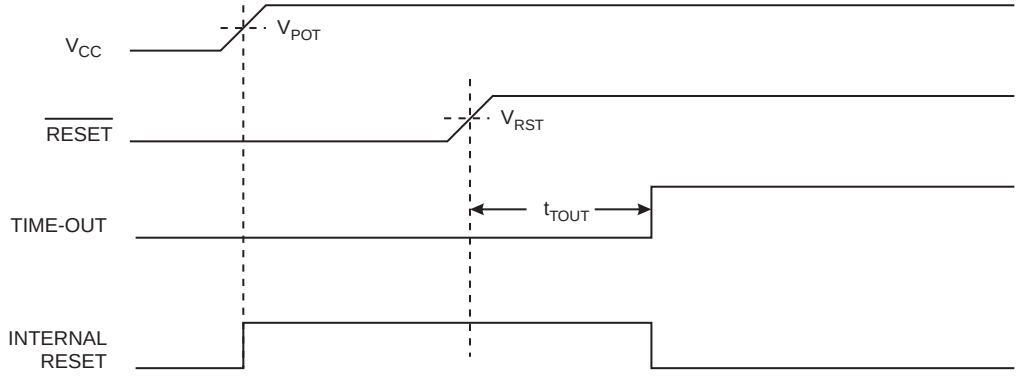


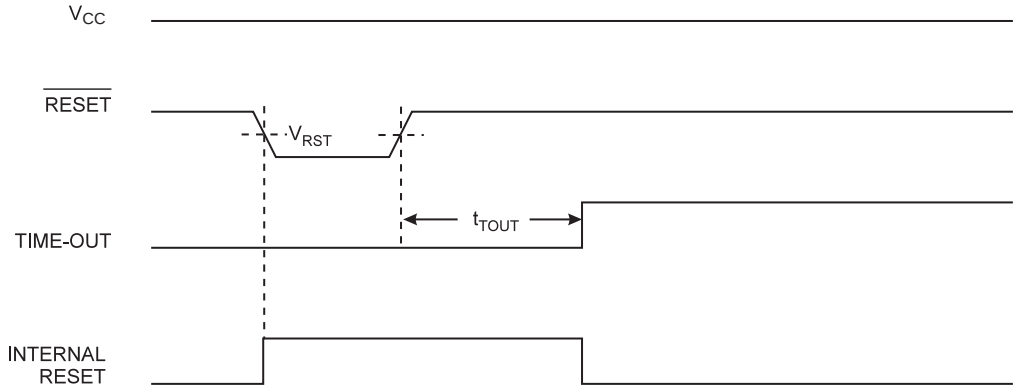
Figure 15. MCU 启动过程，$\overline{\text{RESET}}$ 由外电路控制



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 12) 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 16. 工作过程中发生外部复位



掉电检测

ATtiny13 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 13. BODLEVEL 熔丝位编码 (1)

BODLEVEL [1..0] 熔丝位	最小 V_{BOT}	典型 V_{BOT}	最大 V_{BOT}	单位
11	BOD 禁用			
10		1.8		V
01		2.7		
00		4.3		

Note: 1. 对某些芯片 V_{BOT} 可能低于其标称的最小电压。对这些芯片，在产品测试时，必须有 $V_{CC} = V_{BOT}$ ，以保证电压低于正常工作电压 V_{CC} 时会有掉电复位。

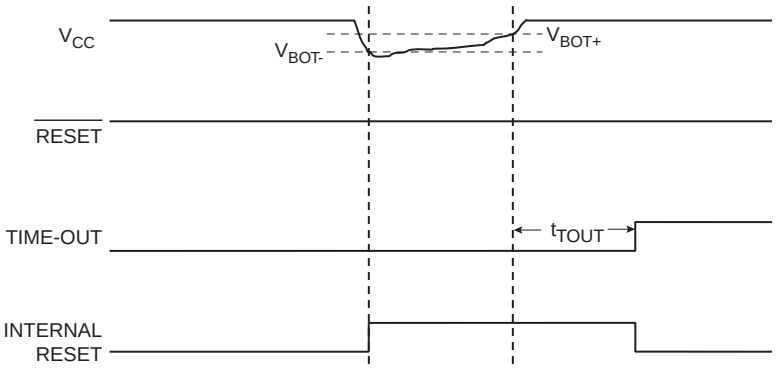
Table 14. 掉电特性

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

当 BOD 使能，一旦 V_{CC} 下降到触发电平以下 (V_{BOT-}， Figure 17)， BOD 复位立即被激发。当 V_{CC} 上升到触发电平以上时 (V_{BOT+}，Figure 17)，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 12 所示的时间 t_{BOD}， BOD 电路将只检测电压跌落。

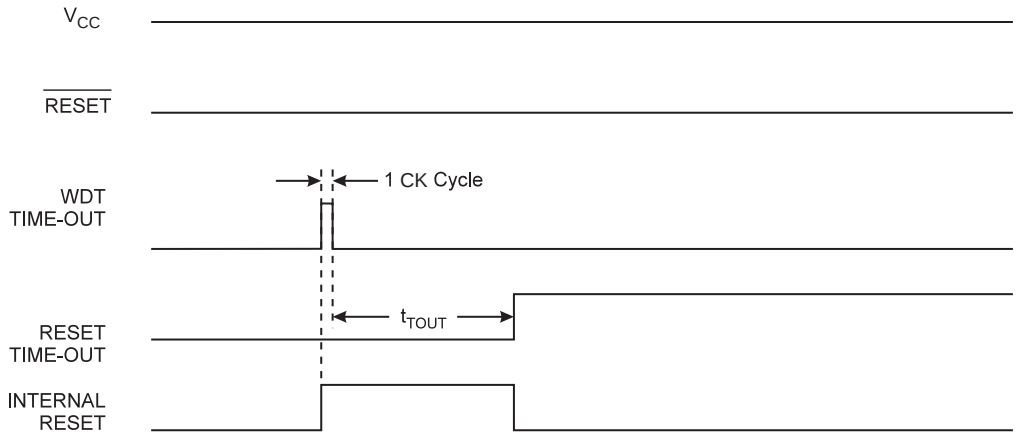
Figure 17. 工作过程中的掉电复位



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 P40 以了解看门狗定时器的具体操作过程。

Figure 18. 工作过程中发生看门狗复位



MCU 状态寄存器 - MCUSR

MCU 状态寄存器提供了有关引起 MCU 复位的复位源的信息。

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

- **Bits 7..4 – Res: 保留**

保留位，读操作返回值为零。

- **Bit 3 – WDRF: 看门狗复位标志**

看门狗复位发生时置位。上电复位将使其清零，也可以通过写 ”0” 来清除。

- **Bit 2 – BORF: 掉电检测复位标志**

掉电检测复位发生时置位。上电复位将使其清零，也可以通过写 ”0” 来清除。

- **Bit 1 – EXTRF: 外部复位标志**

外部复位发生时置位。上电复位将使其清零，也可以通过写 ”0” 来清除。

- **Bit 0 – PORF: 上电复位标志**

上电复位发生时置位。只能通过写 ”0” 来清除。

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

片内基准电压

ATtiny13 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 15。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 [1..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 后要启动基准源。为了降低掉电模式的功耗，用户可以禁止上述三种条件，并在进入掉电模式之前关闭基准源。

Table 15. 内部电压基准源的特性 (1)

符号	参数	最小值	典型值	最大值	单位
V_{BG}	能隙基准源电压	1.0	1.1	1.2	V
t_{BG}	能隙基准源启动时间		40	70	µs
I_{BG}	能隙基准源功耗		15		µA

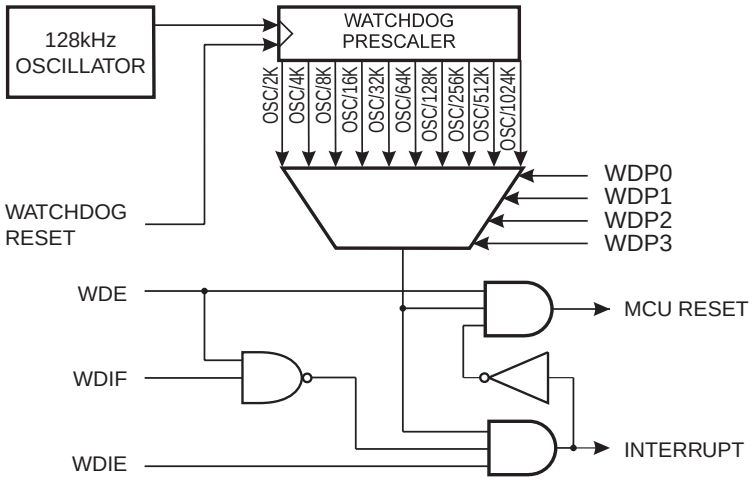
Note: 1. 这些值仅作为参考，实际值待测试。

看门狗定时器

ATtiny13 有一个增强型的看门狗定时器 (WDT)，其主要特征为：

- **独立的片上振荡器提供时钟**
- **3 种工作模式**
  - **– 中断**
  - **– 系统复位**
  - **– 中断与系统复位**
- **暂停时间从 16ms 到 8s 可选**
- **看门狗熔丝始终处于故障保险模式。**

Figure 19. 看门狗定时器



看门狗定时器由独立的 128 kHz 片内振荡器驱动。当计数器达到给定的溢出值时，WDT 发出中断或系统复位。在正常工作模式下，在计数器达到溢出值前，它需要系统使用看门狗定时器复位指令来重启计数器。若系统没有重启计数器，则会出现中断或系统复位。

在中断模式下，当定时器结束 WDT 发出一个中断。该中断可将芯片从休眠状态中唤醒，也可作为一个通用系统定时器。例如限制最大工作时间，当工作时间超出期望值时发出中断。在系统复位模式下，当定时器结束 WDT 发出复位信号。这是为防止由于错误代码所引起的系统挂起的典型使用。第三种模式，中断与系统复位模式，结合两种模式，首先给出中断，然后转换到系统复位模式。使用该模式，可在系统复位前通过保存临界参数来安全关闭。

WDTON 熔丝位编程将使看门狗定时器进入系统复位模式。对其编程时，系统复位模式位(WDE)与中断模式位(WDTIE)分别为1和0。为保证编程安全，必须按照下面顺序来改变看门狗设置：

1. 在一步操作中，同时对 WDCE 位与 WDE 写"1”。无论 WDE 的初始值是多少，在此必须对其写逻辑 ”1”。
2. 在接着的四个时钟周期内，在 WDE 与 WDP 中写入期望值，但同时要清除 WDCE 位。

下面的例子分别用汇编和 C 语言实现了关闭 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 (1)

```
WDT_off:
  ; 关闭全局中断
  cli
  ; WDT 复位
  wdr
  ; 清除 MCUSR 寄存器中 WDRF
  in    r16, MCUSR
  andi  r16, (0xff & (0<<WDRF))
  out   MCUSR, r16
  ; 在 WDCE 与 WDE 中写逻辑 1
  ; 保持旧预分频器设置防止无意暂停
  in    r16, WDTCR
  ori   r16, (1<<WDCE) | (1<<WDE)
  out   WDTCR, r16
  ; 关闭 WDT
  ldi   r16, (0<<WDE)
  out   WDTCR, r16
  ;开启全局中断
  sei
  ret
```

C 代码例程 (1)

```
void WDT_off(void)
{
  __disable_interrupt();
  __watchdog_reset();
  /* 清除 MCUSR 寄存器中 WDRF*/
  MCUSR &= ~(1<<WDRF);
  /* 在 WDCE 与 WDE 中写逻辑 1*/
  /* 保持旧预分频器设置防止无意暂停 */
  WDTCR |= (1<<WDCE) | (1<<WDE);
  /* 关闭 WDT */
  WDTCR = 0x00;
  __enable_interrupt();
}
```

Note: 1. 代码例程假设包括所需头文件。

注意：若看门狗由于错误指针或掉电状态等使看门狗出现意外使能，芯片将复位看门狗定时器将保持使能。如果编码没有设置处理看门狗，则可能导致溢出复位出现死循环。为避免出现这种状况，即使没有使用看门狗，应用程序在初始化时应对 WDRF 与 WDE 控制位清零。

下面的例子分别用汇编和 C 语言实现了看门狗定时器溢出值的改变。

汇编代码例程 (1)

```
WDT_Prescaler_Change:
  ; 关闭全局中断
  cli
  ; 复位看门狗定时器
  wdr
  ; 启动时序
  in    r16, WDTCR
  ori   r16, (1<<WDCE) | (1<<WDE)
  out   WDTCR, r16
  ; --  从此时在四个周期中设置新值 -
  ; 设置新预分频器值 = 64K 周期 (~0.5 s)
  ldi   r16, (1<<WDE) | (1<<WDP2) | (1<<WDP0)
  out   WDTCR, r16
  ; --  完成新值设置，使用 2 周期 -
  ; 开启全局中断
  sei
  ret
```

C 代码例程 (1)

```
void WDT_Prescaler_Change(void)
{
  __disable_interrupt();
  __watchdog_reset();
  /* 启动时序 */
  WDTCR |= (1<<WDCE) | (1<<WDE);
  /* 设置新预分频器值 = 64K 周期 (~0.5 s) */
  WDTCR = (1<<WDE) | (1<<WDP2) | (1<<WDP0);
  __enable_interrupt();
}
```

Note: 1. 代码例程假设包括所需头文件。

注意：看门狗定时器应在 WDP 位改变前复位，因为当改变 WDP 转换到一个短溢出周期时可能会导致暂停。

看门狗定时器控制寄存器 - WDTCR

Bit	7	6	5	4	3	2	1	0	
	WDTIF	WDTIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	WDTCR
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	X	0	0	0	

- **Bit 7 - WDTIF: 看门狗定时器中断标志**

当看门狗定时器出现溢出且看门狗定时器配置为中断时，该位置位。当执行相应的中断程序时， WDTIF 由硬件清除；或者在该位写入逻辑 "1" 来清除。当 SREG 寄存器的 I 位与 WDTIE 置位，执行看门狗溢出中断 。

- **Bit 6 - WDTIE: 看门狗定时器中断使能**

当该位与 SREG 寄存器的 I 位置位，看门狗中断使能。如果同时也将 WDE 清除，看门狗定时器进入中断模式，一旦看门狗定时器程序暂停，则执行相应的中断。

若 WDE 置位，则看门狗定时器处于中断与系统复位模式。看门狗定时器的第一次溢出将设置 WDTIF。执行相应的中断向量将会由硬件最大清除 WDTIE 与 WDTIF(看门狗进入系统复位模式)。这种方式会保证使用中断时看门狗定时器的安全性。在中断与系统复位模式下， WDTIE 在每次中断后必须设置。然而它不能在中断服务子程序中执行，因为这可能会损害看门狗系统复位模式的安全性。如果在下一次溢出前没有程序中断，则进入系统复位模式。

Table 16. 看门狗定时器配置

WDTON	WDE	WDTIE	模式	暂停活动
0	0	0	停止	无
0	0	1	中断模式	中断
0	1	0	系统复位模式	复位
0	1	1	中断与系统复位模式	中断，然后进入系统复位模式
1	x	x	系统复位模式	复位

- **Bit 4 - WDCE: 看门狗修改使能**

该位用在改变 WDE 与预分频位的时序中。WDCE 置位来清除 WDE 位，与 / 或改变预分频位。

一旦置 "1"，硬件将在四个时钟周期后对 WDCE 清零。

- **Bit 3 - WDE: 看门狗系统复位使能**

WDE 由 MCUSR 寄存器的 WDRF 决定。这就是说当 WDRF 设置时 ,WDE 也设置。要清除 WDE ,必须先清除 WDRF。这一特性保证状态出错时的多重复位 ,及出错后的安全启动。

- **Bit 5, 2..0 - WDP3..0: 看门狗定时器预分频器 3, 2, 1 和 0**

WDP3..0 决定看门狗定时器的预分频器。如 P39Table 17 所示。

Table 17. 看门狗定时器预分频器选项

WDP3	WDP2	WDP1	WDP0	看门狗振荡器周期	V_{CC} = 5.0V 时典型的溢出周期
0	0	0	0	2K (2048) 周期	16 ms
0	0	0	1	4K (4096) 周期	32 ms
0	0	1	0	8K (8192) 周期	64 ms
0	0	1	1	16K (16384) 周期	0.125 s
0	1	0	0	32K (32768) 周期	0.25 s
0	1	0	1	64K (65536) 周期	0.5 s
0	1	1	0	128K (131072) 周期	1.0 s
0	1	1	1	256K (262144) 周期	2.0 s
1	0	0	0	512K (524288) 周期	4.0 s
1	0	0	1	1024K (1048576) 周期	8.0 s
1	0	1	0	保留	
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

中断

本节描述 ATtiny13 的中断处理。更一般的 AVR 中断处理请参见 P9“ 复位与中断处理 ”。

ATtiny13 的中断向量

Table 18. 复位和中断向量

向量号	程序地址	中断源	中断定义
1	0x0000	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
2	0x0001	INT0	外部中断请求 0
3	0x0002	PCINT0	外部中断请求 1
4	0x0003	TIM0_OVF	定时器 / 计数器溢出
5	0x0004	EE_RDY	EEPROM 准备好
6	0x0005	ANA_COMP	模拟比较器
7	0x0006	TIM0_COMPA	定时器 / 计数器比较匹配 A
8	0x0007	TIM0_COMPB	定时器 / 计数器比较匹配 B
9	0x0008	WDT	看门狗暂停
10	0x0009	ADC	ADC 转换结束

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。 ATtiny13 复位与中断向量地址典型设置为：

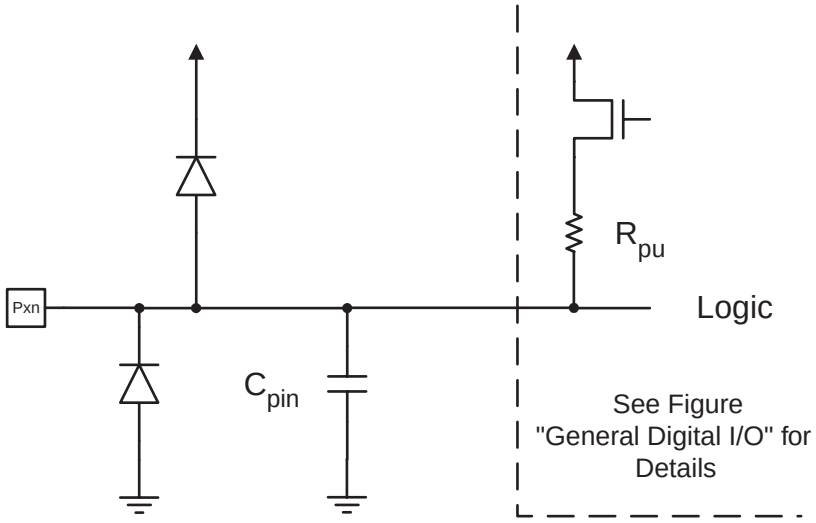
```
地址      符号    代码                    说明
0x0000            rjmp   RESET           ; 复位中断向量
0x0001            rjmp   EXT_INT0        ; IRQ0 中断向量
0x0002            rjmp   PCINT0          ; PCINT0 中断向量
0x0003            rjmp   TIM0_OVF        ; Timer0 溢出中断向量
0x0004            rjmp   EE_RDY          ; EEPROM 准备好中断向量
0x0005            rjmp   ANA_COMP        ; 模拟比较器中断向量
0x0006            rjmp   TIM0_COMPA      ; Timer0 比较 A 中断向量
0x0007            rjmp   TIM0_COMPB      ; Timer0 比较 B 中断向量
0x0008            rjmp   WATCHDOG        ; 看门狗中断向量
0x0009            rjmp   ADC             ; ADC 转换中断向量
;
0x000A   RESET: ldi    r16, low(RAMEND); 主程序
0x000B    out    SPL,r16                ; 设置堆栈指针为 RAM 的顶部
0x000C          sei                     ; 使能中断
0x000D    <instr>  xxx
   ...     ...     ...   ...
```

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时不会无意地改变其他管脚的方向。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 20 所示。请参见 P112“电气特性 ” 以了解完整的参数列表。

Figure 20. 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 P50“I/O 端口寄存器说明 ” 。

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。

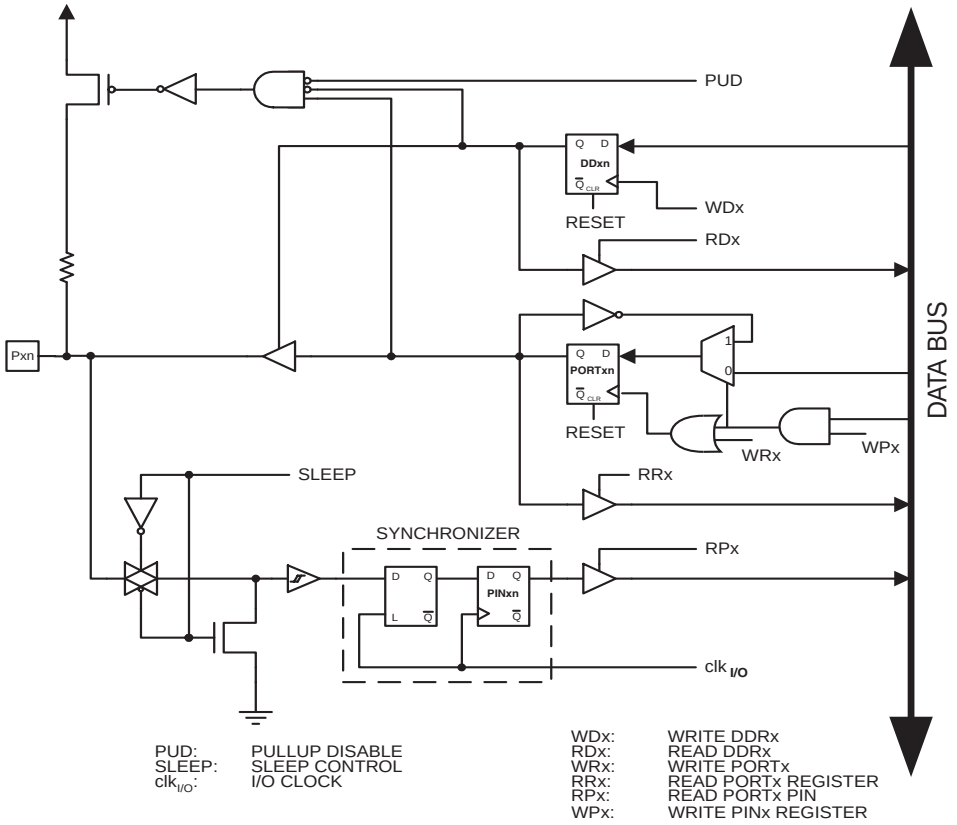
作为通用数字 I/O 时的端口请参见 P41“ 作为通用数字 I/O 的端口 ” 。多数端口引脚是与第二功能复用的，如 P46“ 端口的第二功能 ” 所示。请参见各个模块的具体说明以了解引脚的第二功能。

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 21 为一个 I/O 端口引脚的说明。

Figure 21. 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 P50“I/O 端口寄存器说明 ” 所示。DDxn 位于 DDRx 寄存器，PORTxn 位于 PORTx 寄存器，PINxn 位于 PINx 寄存器。

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

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

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

转换引脚

在 PINxn 写逻辑 "1” 转换 PORTxn 值，与 DDRxn 值无关。注意，SBI 指令可用来端口的一位。

输入输出间转换

在 (高阻态) 三态 ({DDxn, PORTxn} = 0b00) 输出高电平 ({DDxn, PORTxn} = 0b11) 两种状态之间进行切换时，上拉电阻使能 ({DDxn, PORTxn} = 0b01) 或输出低电平 ({DDxn, PORTxn} = 0b10) 这两种模式必然会有一个发生。通常，上拉电阻使能是完全可以接受的，因为高阻环境不在意是强高电平输出还是上拉输出。如果使用情况不是这样子，可以通过置位 MCUCR 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 19 总结了引脚的控制信号。

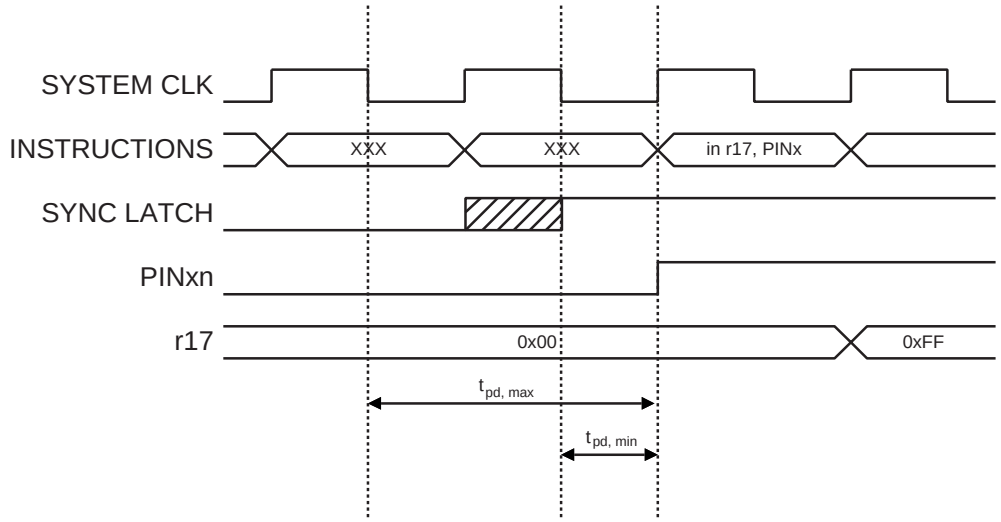
Table 19. 端口引脚配置

DDxn	PORTxn	PUD (in 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	被外部电路拉低时将输出电流
0	1	1	输入	No	高阻态 (Hi-Z)
1	0	X	输出	No	输出低电平 (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

不论如何配置 DDxn ,都可以通过读取 PINxn 寄存器来获得引脚电平。如 Figure 21 所示，PINxn 寄存器的各个位与其前面的锁存器组成了一个同步器。这样就可以避免在内部时钟状态发生改变的短时间范围内由于引脚电平变化而造成的信号不稳定。其缺点是引入了延迟。 Figure 22 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 $t_{pd,min}$。

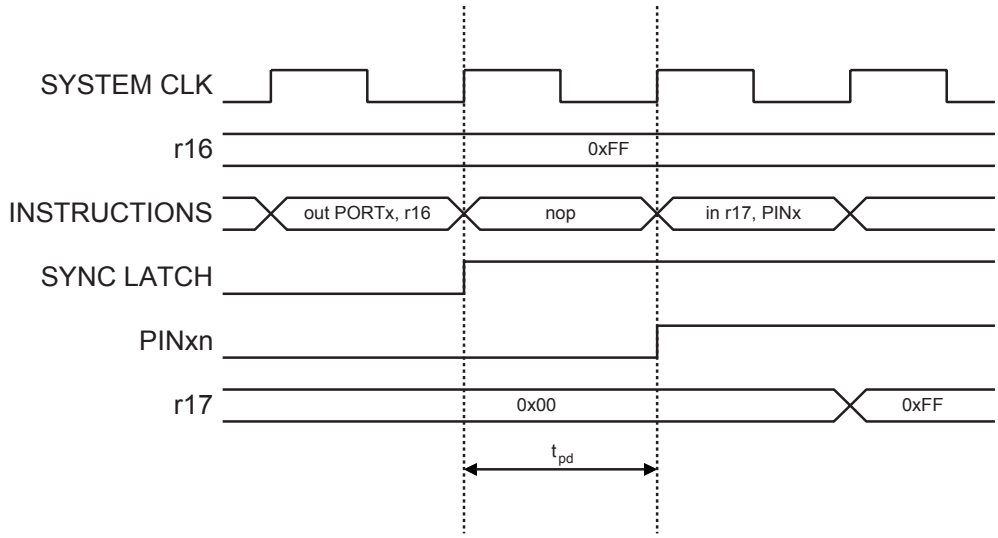
Figure 22. 读取引脚数据时的同步



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

如 Figure 23 所示，读取软件赋予的引脚电平时需要在赋值指令 *out* 和读取指令 *in* 之间有一个时钟周期的间隔，如 *nop* 指令。*out* 指令在时钟的上升沿置位 SYNC LATCH 信号。此时同步器的延迟时间 t_{pd} 为一个系统时钟。

Figure 23. 读取软件赋予的引脚电平的同步



下面的例子演示了如何置位端口 B 的引脚 0 和 1，清零引脚 2 和 3，以及将引脚 4 到 5 设置为输入，并且为引脚 4 设置上拉电阻。然后将各个引脚的数据读回来。如前面讨论的那样，我们在输出和输入语句之间插入了一个 *nop* 指令。

汇编代码例程 (1)

```
...
; 定义上拉电阻和设置高电平输出
; 为端口引脚定义方向
ldi  r16,(1<<PB4)|(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<<PB4)|(1<<PB1)|(1<<PB0);
DDRB = (1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0);
/* 为了同步插入 nop 指令 */
__no_operation();
/* 读取端口引脚 */
i = PINB;
...
```

Note: 1. 在汇编程序里使用了两个暂存器。其目的是为了使整个操作过程的时间最短。通过拉高引脚 0、1 与 4，直到方向位设置正确，定义位 2、3 为低，且重新定义为 0 与 1 为强驱动。

数字输入使能和休眠模式

如 Figure 21 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 P46“ 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

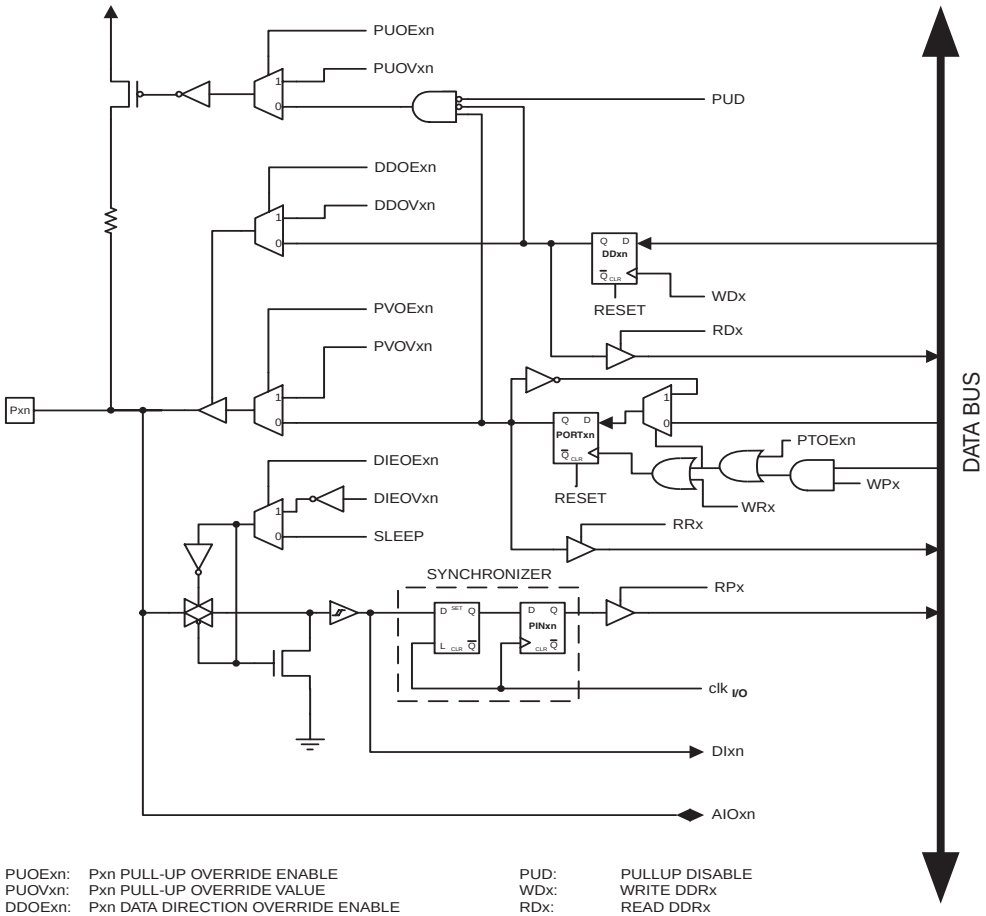
如果有引脚未被使用，建议给这些引脚赋予一个确定电平。虽然如上文所述，在深层休眠模式下大多数数字输入被禁用，但还是需要避免因引脚没有确定的电平而造成悬空引脚在其它数字输入使能模式 (复位、工作模式、空闲模式) 消耗电流。

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 24 说明了由Figure 21 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 24. 端口的第二功能 (1)



PUOExn:	Pxn PULL-UP OVERRIDE ENABLE	PUD:	PULLUP DISABLE
PUOVxn:	Pxn PULL-UP OVERRIDE VALUE	WDx:	WRITE DDRx
DDOExn:	Pxn DATA DIRECTION OVERRIDE ENABLE	RDx:	READ DDRx
DDOVxn:	Pxn DATA DIRECTION OVERRIDE VALUE	RRx:	READ PORTx REGISTER
PVOExn:	Pxn PORT VALUE OVERRIDE ENABLE	WRx:	WRITE PORTx
PVOVxn:	Pxn PORT VALUE OVERRIDE VALUE	RPx:	READ PORTx PIN
DIEOExn:	Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE	WPx:	WRITE PINx
DIEOVxn:	Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE	$clk_{I/O}$:	I/O CLOCK
SLEEP:	SLEEP CONTROL	DIxn:	DIGITAL INPUT PIN n ON PORTx
PTOExn:	Pxn, PORT TOGGLE OVERRIDE ENABLE	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 20 为重载信号的简介。表中没有给出 Figure 24 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 20. 第二功能重载信号的一般说明

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

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

MCU 控制寄存器 - MCUCR

Bit	7	6	5	4	3	2	1	0	
	–	PUD	SE	SM1	SM0	–	ISC01	ISC00	MCUCR
读 / 写	R	R/W	R/W	R/W	R/W	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bits 7, 2– Res: 保留**

保留位，读操作返回值为零。

- **Bit 6 – PUD: 上拉禁用**

当该位为 1，即使 DDxn 与 PORTxn 寄存器配置为使能上拉电阻 ({DDxn, PORTxn} = 0b01)， I/O 端口上拉电阻禁用，详见 P42“ 配置引脚 ”。

端口 B 的第二功能

端口 B 的第二功能列于 Table 21。

Table 21. 端口 B 的第二功能

端口引脚	第二功能
PB5	$\overline{\text{RESET}}$/dW/ADC0/PCINT5(1)
PB4	ADC2/PCINT4(2)
PB3	ADC3/CLKI/PCINT3(3)
PB2	SCK/ADC1/T0/PCINT2(4)
PB1	MISO/AIN1/OC0B/INT0/PCINT1/RXD(5)
PB0	MOSI/AIN0/OC0A/PCINT0/TXD(6)

Notes:
1. 复位引脚，调试线 I/O， ADC 输入通道或引脚变化中断。
2. ADC 输入通道或引脚变化中断。
3. ADC 输入通道，时钟输入或引脚变化中断。
4. 串行时钟输入，定时器 / 计数器时钟输入， ADC 输入通道 0 或引脚变化中断
5. 串行数据输入，模拟比较器反向输入，输出比较与定时器 / 计数器的 PWM 输出 B，外部中断 0 或引脚变化中断
6. 串行数据输入，模拟比较器正向输入，输出比较与定时器 / 计数器的 PWM 输出 A 或引脚变化中断

Table 22 与 P49Table 23 端口 B 的第二功能的重载信号见 P46Figure 24。

Table 22. PB5..PB3 重载信号

信号名称	PB5/RESET/ ADC0/PCINT5	PB4/ADC2/PCINT4	PB3/ADC3/CLKI/PCINT3
PUOE	$\overline{\text{RSTDISBL}}$[1] • DWEN[1]	0	0
PUOV	1	0	0
DDOE	RSTDISBL[1] • DWEN[1]	0	0
DDOV	调试线传送	0	0
PVOE	0	0	0
PVOV	0	0	0
PTOE	0	0	0
DIEOE	$\overline{\text{RSTDISBL}}$[1] + (PCINT5 • PCIE + ADC0D)	PCINT4 • PCIE + ADC2D	PCINT3 • PCIE + ADC3D
DIEOV	$\overline{\text{ADC0D}}$	$\overline{\text{ADC2D}}$	$\overline{\text{ADC3D}}$
DI	PCINT5 输入	PCINT4 输入	PCINT3 输入
AIO	RESET 输入，ADC0 输入	ADC2 输入	ADC3 输入

Note: 1. 1 当熔丝位为 “0” (已编程)。

Table 23. PB2..PB0 重载信号

信号名称	PB2/SCK/ADC1/ T0/PCINT2	PB1/MISO/AIN1/ OC0B/INT0/PCINT1	PB0/MOSI/AIN0/AREF/ OC0A/PCINT0
PUOE	0	0	0
PUOV	0	0	0
DDOE	0	0	0
DDOV	0	0	0
PVOE	0	OC0B 使能	OC0A 使能
PVOV	0	OC0B	OC0A
PTOE	0	0	0
DIEOE	PCINT2 • PCIE + ADC1D	PCINT1 • PCIE + AIN1D	PCINT0 • PCIE + AIN0D
DIEOV	$\overline{\text{ADC1D}}$	$\overline{\text{AIN1D}}$	$\overline{\text{AIN0D}}$
DI	T0/INT0/ PCINT2 输入	PCINT1 输入	PCINT0 输入
AIO	ADC1 输入	模拟比较器反向输入	模拟比较器正向输入

I/O 端口寄存器说明

端口 B 数据寄存器 - PORTB

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

端口 B 数据方向寄存器 - DDRB

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

端口 B 输入引脚地址 - PINB

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

外部中断

外部中断通过引脚INT0或PCINT5..0触发。只要使能了中断，即使引脚INT0或PCINT5..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。使能PCINT5..0引脚转换将触发引脚变化PCI。哪个引脚作为引脚变化中断由PCMSK寄存器控制。PCINT5..0 的引脚变化中断是异步检测的。也就是说，这些中断可以用来将器件从睡眠模式唤醒。

通过设置 MCU 控制寄存器 MCUCR，中断可以由下降沿、上升沿，或者是低电平触发。当外部中断使能并且配置为电平触发，只要引脚电平为低，中断就会产生。若要求 INT0 在信号下降沿或上升沿触发， I/O 时钟必须工作，如 P20“ 时钟系统及其分布 ” 说明的那样。 INT0 的中断条件检测是异步的。也就是说，这些中断可以用来将器件从睡眠模式唤醒。在睡眠过程 (除了空闲模式) 中 I/O 时钟是停止的。

注意，通过电平方式触发中断，从而将 MCU 从掉电模式唤醒时，要保证电平保持一定的时间。若电平在启动完成前结束， MCU 被唤醒，但不会产生中断。启动时间由 P20“ 系统时钟及时钟选项 ” 所示的 SUT 与 CKSEL 熔丝位定义。

MCU 控制寄存器 - MCUCR

外部中断控制寄存器 A 包含中断敏感控制的控制位。

Bit	7	6	5	4	3	2	1	0	
	–	PUD	SE	SM1	SM0	–	ISC01	ISC00	MCUCR
读 / 写	R	R/W	R/W	R/W	R/W	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bits 1, 0 – ISC01, ISC00: 中断敏感方式控制 0 Bit1 与 Bit0**

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 24 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 24. 中断 0 敏感控制

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

通用中断屏蔽寄存器 - GIMSK

Bit	7	6	5	4	3	2	1	0	
	–	INT0	PCIE	–	–	–	–	–	GIMSK
读 / 写	R	R/W	R/W	R	R	R	R	R	
初始值	0	0	0	0	0	0	0	0	

- **Bits 7, 4..0 – Res: 保留**

保留位，该操作返回值为零。

- **Bit 6 – INT0: 外部中断请求 0 使能**

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

- **Bit 5 – PCIE: 引脚变化中断使能**

当 PCIE 位为 '1'，而且状态寄存器 SREG 的 I 标志置位，引脚变化中断使能。使能的 PCINT5..0 引脚上电平变化将造成中断。执行相应的 PCI 中断程序。PCINT5..0 各引脚由 PCMSK0 寄存器单独使能。

通用中断标志寄存器 - GIFR

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

- **Bits 7, 4..0 – Res: 保留**

保留位，该操作返回值为零。

- **Bit 6 – INTF0: 外部中断标志 0**

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

- **Bit 5 – PCIF: 引脚变化中断标志**

当 PCINT5..0 任意引脚电平变化触发中断请求，PCIF 置 "1"。若 SREG 的位 I 与 GIMSK 寄存器的 PCIE 位为 "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。

引脚变化屏蔽寄存器 - PCMSK

Bit	7	6	5	4	3	2	1	0	
	–	–	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK
读 / 写	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	1	1	1	1	1	1	

- **Bits 7, 6 – Res: 保留**

保留位，该操作返回值为零。

- **Bits 5..0 – PCINT5..0: 引脚变化使能屏蔽 5..0**

每个 PCINT5..0 位选择是否使能相应的 I/O 引脚变化中断。若 PCINT5..0 与 PCIE 位设置，使能相应的引脚变化中断。若 PCINT5..0 清除，则禁用相应的引脚变化中断。

具有 PWM 功能的 8 位定时器 / 计时器 0

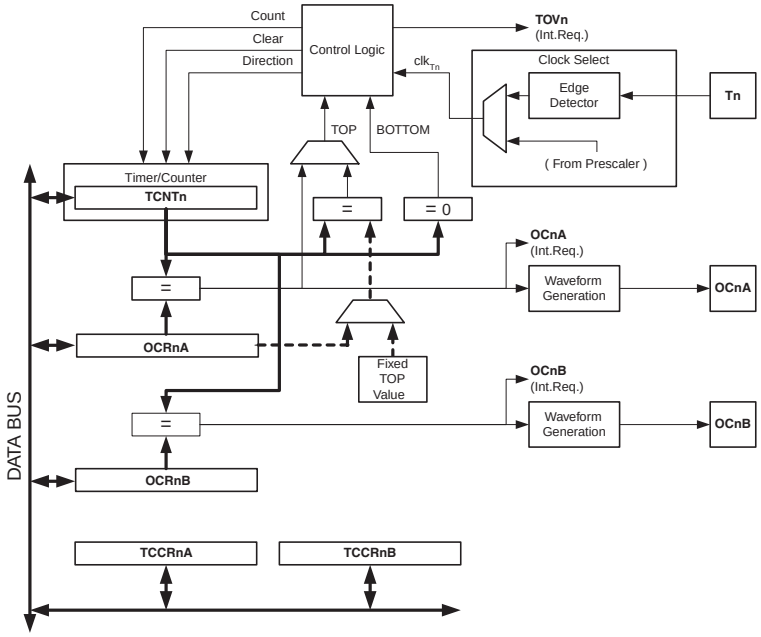
T/C0 是通用 8 位定时器 / 计数器模块，有两个独立的输出比较单元，且支持 PWM 功能。它提供精确的执行时序与波形产生。其主要特点如下：

- **两个独立输出比较单元**
- **双缓冲输出比较寄存器**
- **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**
- **可变 PWM 周期**
- **频率发生器**
- **三个独立中断源 (TOV0, OCF0A 及 OCF0B)**

综述

Figure 25为8位定时器/计数器的简化框图。实际引脚排列请参考P1“ATtiny13芯片引脚”。CPU可以访问的I/O寄存器，包括位和引脚，以粗体显示。I/O寄存器和位的位置列于P65“8位 定时器 / 计数器寄存器的说明”。

Figure 25. 8 位 T/C 方框图



寄存器

T/C(TCNT0)和输出比较寄存器(OCR0A 与OCR0B)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。图中没有给出 TIFR0 和 TIMSK0。

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 与 OCR0B) 一直与 T/C 的数值进行比较。比较的结果可用来产生 PWM 波，或在输出比较引脚 OC0 上产生变化频率的输出，如 P56“ 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 或 OCF0B)。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 或 B。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 25 的定义适用于全文。

Table 25. 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

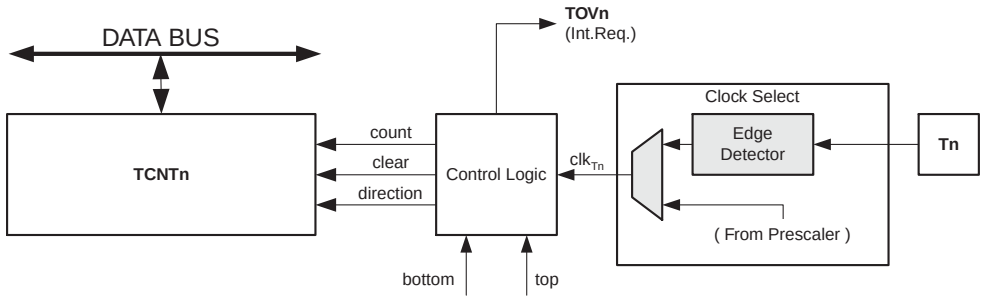
T/C 的时钟源

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 TCCR0B 的时钟选择位 CS02:0 控制的。 P71“T/C 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 26 即为计数器和周边电路的框图。

Figure 26. 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 及 (TCCR0B) 的 WGM02 位决定。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 P59“ 工作模式 ” 。

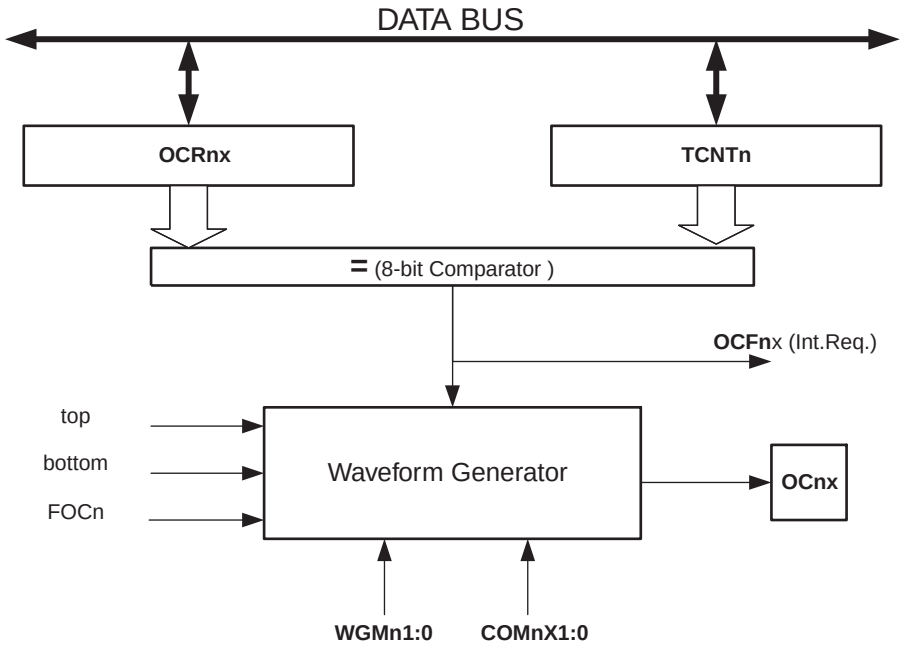
T/C溢出中断标志TOV0根据WGM01:0 设定的工作模式来设置。TOV0可以用于产生CPU中断。

输出比较单元

8位比较器持续对TCNT0和输出比较寄存器OCR0A(与OCR0B)进行比较。一旦TCNT0等于 OCR0A 或 OCR0B，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(或 OCF0B) 置位。若此时相应中断使能， CPU 将产生输出比较中断。执行中断服务程序时 OCF0A(或 OCF0B) 自动清零，或者通过软件写 ”1” 的方式来清零。根据由 WGM2:0 和 COM0x1:0 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 (P59“ 工作模式 ”)。

Figure 27 为输出比较单元的方框图。

Figure 27. 输出比较单元方框图



使用 PWM 模式时 OCR0x 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 OCR0x 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 OCR0x 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 OCR0x 缓冲寄存器；禁止双缓冲功能时 CPU 访问的则是 OCR0x 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 FOC0x 写 ”1” 的方式来产生比较匹配。强制比较匹配不会置位 OCF0x 标志，也不会重载 / 清零定时器，但是 OC0x 引脚将被更新，好象真的发生了比较匹配一样 (COM01:0 决定 OC0x 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 OCR0x 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 OCR0x，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。

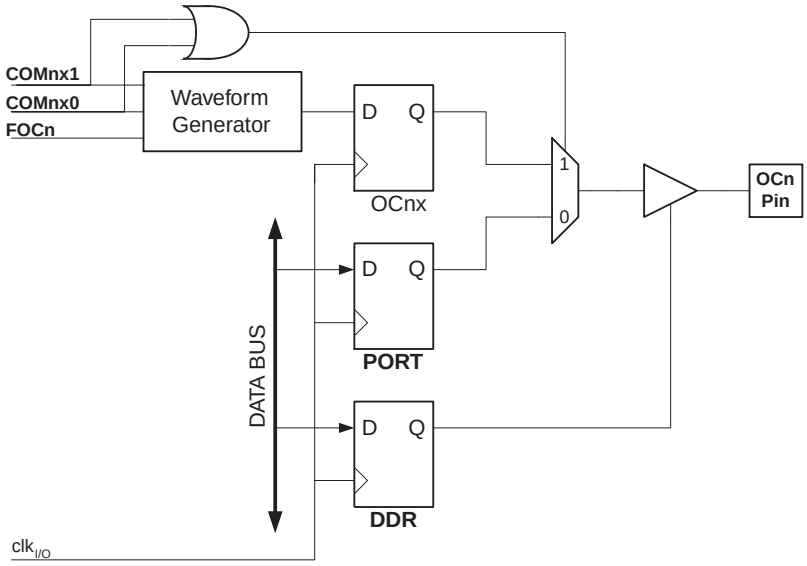
OC0x 的设置应该在设置数据方向寄存器之前完成。最简单的设置 OC0x 的方法是在普通模式下利用强制输出比较 FOC0x。即使在改变波形发生模式时 OC0x 寄存器也会一直保持它的数值。

注意 COM0x1:0 和比较数据都不是双缓冲的。 COM0x1:0 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 COM0x1:0 具有双重功能。波形发生器利用 COM0x1:0 来确定下一次比较匹配发生时的输出比较状态 (OC0x) ；COM0x1:0 还控制 OC0x 引脚输出信号的来源。 Figure 28 为受 COM0x1:0 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 COM0x1:0 影响的通用 I/O 端口控制寄存器 (DDR 和 PORT)。谈及OC0x状态时指的是内部OC0x寄存器，而不是OC0x引脚。系统复位时OC0x寄存器清零。

Figure 28. 比较匹配输出单元原理图



如果COM0x1:0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OC0x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 OC0x 功能之前首先要通过数据方向寄存器的 DDR_OC0x 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 OC0x 状态在输出之前首先进行初始化。要注意某些 COM0x1:0 设置保留给了其他操作类型，详见 P65“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 COM0x1:0 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 COM0x1:0 = 0 表明比较匹配发生时波形发生器不会操作 OC0x 寄存器。非PWM模式的比较输出请参见 P65Table 26 ；快速PWM的比较输出示于 P65Table 27 ；相位修正 PWM 的比较输出在 P65Table 28 有描述。

改变 COM0x1:0 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 FOC0x 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 (WGM02:0) 及比较输出模式 (COM0x1:0) 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。 COM0x1:0 控制 PWM 输出是否为反极性。非 PWM 模式时 COM0x1:0 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 (P58“ 比较匹配输出单元 ”)。

具体的时序信息请参考 P63“T/C 时序图 ” 之 Figure 32、Figure 33 、Figure 34 与 Figure 35。

普通模式

普通模式 (WGM02:0 = 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。

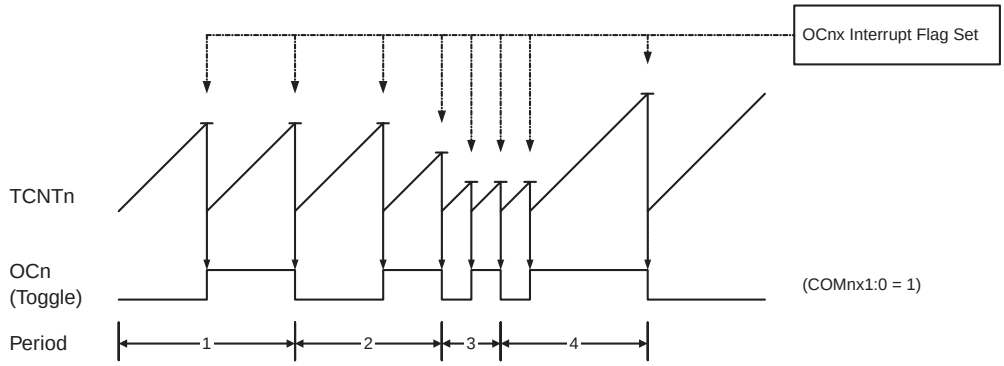
输出比较单元可以用来产生中断。但是不推荐在普通模式下利用输出比较来产生波形，因为这会占用太多的 CPU 时间。

CTC(比较匹配时清零定时器) 模式

在 CTC 模式 (WGM02:0 = 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 29。计数器数值 TCNT0 一直累加到 TCNT0 与 OCR0A 匹配，然后 TCNT0 清零。

Figure 29. 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 A= 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、 256 或 1024)。

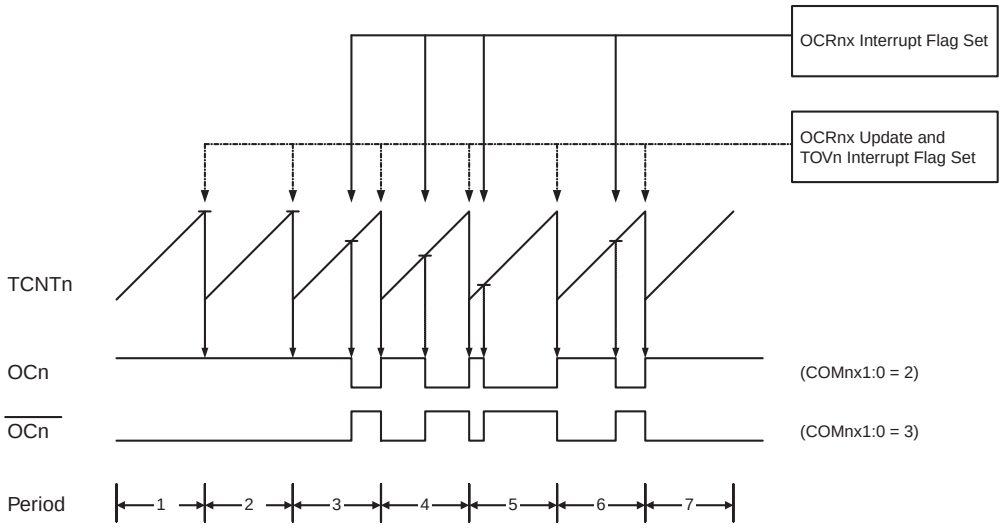
在普通模式下， TOV0 标志的置位发生在计数器从 MAX 变为 0x00 的定时器时钟周期。

快速 PWM 模式

快速 PWM 模式 (WGM02:0 = 3 或 7) 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 TOP，然后立即回到 BOTTOM 重新开始。当 WGM2:0 = 3 时，TOP 值为 0xFF ；当 WGM2:0 = 7 时，TOP值为OCR0A。对于普通的比较输出模式，输出比较引脚OC0x在TCNT0与OCR0x匹配时清零，在 BOTTOM 时置位；对于反向比较输出模式， OC0x 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 TOP，然后在后面的一个时钟周期清零。具体的时序图为 Figure 30。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 OCR0x 和 TCNT0 的比较匹配。

Figure 30. 快速 PWM 模式时序图



计时器数值达到 TOP 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OC0x引脚上输出PWM波形。设置COM0x1:0为 2 可以产生普通的 PWM 信号；为 3 则可以产生反向 PWM 波形：若 WGM02 位设置，设置 COM0A1:0 位为 "1" 允许 AC0A 引脚转换到比较匹配。该选项对 OC0B 引脚无效 (参见 P65Table 27)。要想在引脚上得到输出信号还必须将 OC0x 的数据方向设置为输出。产生 PWM 波形的机理是 OC0x 寄存器在 OCR0x 与 TCNT0 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：

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

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。

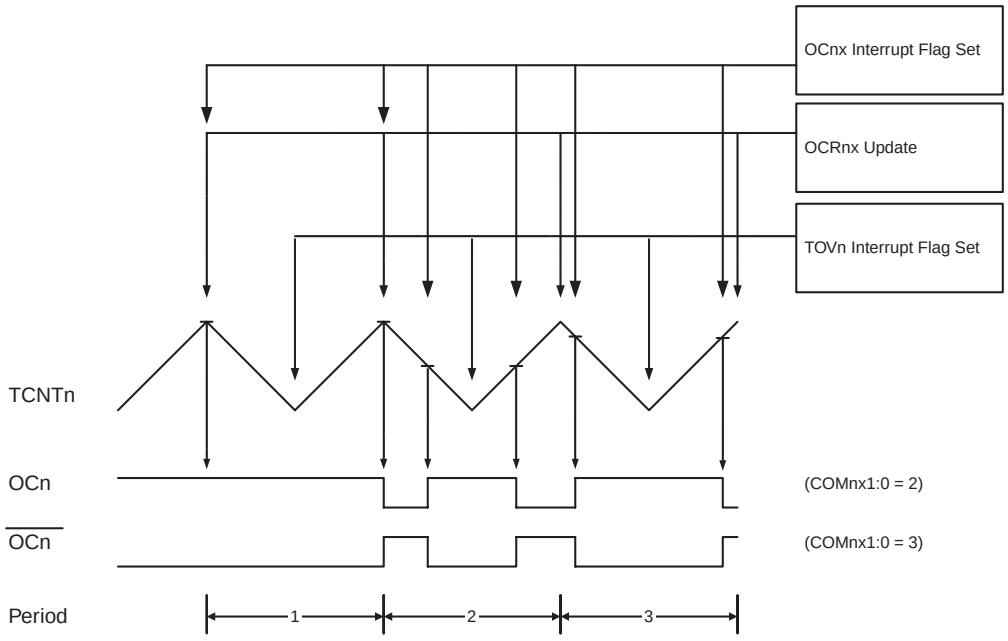
通过设定OC0x在比较匹配时进行逻辑电平取反(COM0x1:0 = 1)，可以得到占空比为50%的周期信号。 OCR0A 为 0 时信号有最高频率 $f_{oc0} = f_{clk\_I/O}/2$。这个特性类似于 CTC 模式下的 OC0A 取反操作，不同之处在于快速 PWM 模式具有双缓冲。

相位修正 PWM 模式

相位修正 PWM 模式 (WGM02:0 = 1 或 5) 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 TOP，然后又从 TOP 倒退回到 BOTTOM。当 WGM2:0 = 1 时， TOP 值为 0xFF ；当 WGM2:0 = 5 时， TOP 值为 OCR0A 。在一般的比较输出模式下，当计时器往 TOP 计数时若发生了 TCNT0 与 OCR0x 的匹配，OC0x 将清零为低电平；而在计时器往 BOTTOM 计数时若发生了 TCNT0 与 OCR0x 的匹配， OC0x 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。

计时器不断地累加直到 TOP，然后开始减计数。在一个定时器时钟周期里 TCNT0 的值等于 TOP。时序图可参见 Figure 31。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 OCR0x 与 TCNT0 的比较匹配。

Figure 31. 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 OC0x 引脚产生 PWM 波形：将 COM0x1:0 设置为 2 产生普通相位的 PWM，设置 COM0x1:0 为 3 产生反向 PWM 信号 ：若 WGM02 位设置，设置 COM0A0 位为 "1" 允许 OC0A 引脚转换到比较匹配。该选项对 OC0B 引脚无效 (参见 P65Table 28)。要想在引脚上得到输出信号还必须将 OC0x 的数据方向设置为输出。 OCR0x 和 TCNT0 比较匹配发生时 OC0x 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 31 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 31 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。
- 定时器从一个比 OCR0A 高的值开始记数，并因而丢失了一次比较匹配。系统因此引入发生 OCn 却仍然有跳变的现象。

T/C 时序图

T/C 是同步电路 ,因此其时钟 clk_{T0} 可以表示为时钟使能信号 ,如下图所示。图中还说明了中断标志设置的时间。 Figure 32 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 32. T/C 时序图，无预分频器

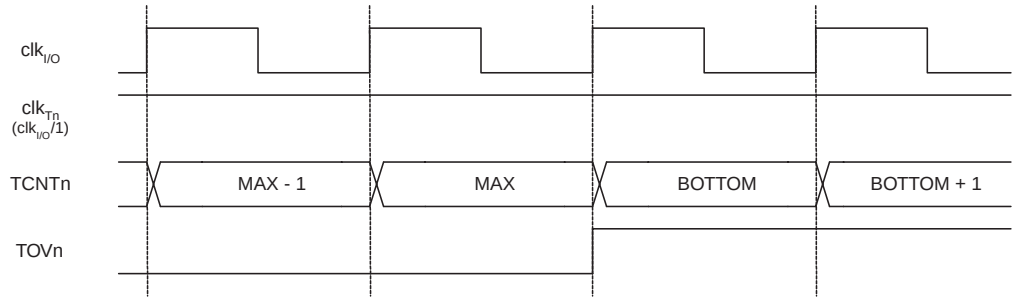


Figure 33 所示为相同的工作时序，但有预分频。

Figure 33. T/C 时序图，预分频器为 $f_{clk\_I/O}/8$

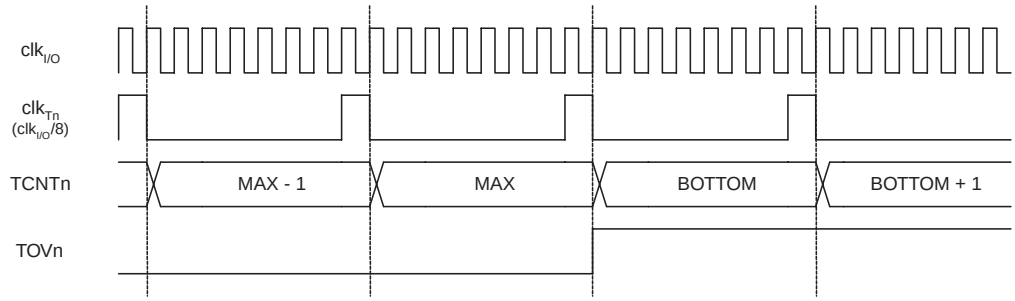


Figure 34 给出了各种模式下 (除了 CTC 模式)OCF0x 的置位情况 ,其中 OCR0A 为 TOP。

Figure 34. T/C 时序图， OCF0x 置位，预分频器为 $f_{clk\_I/O}/8$

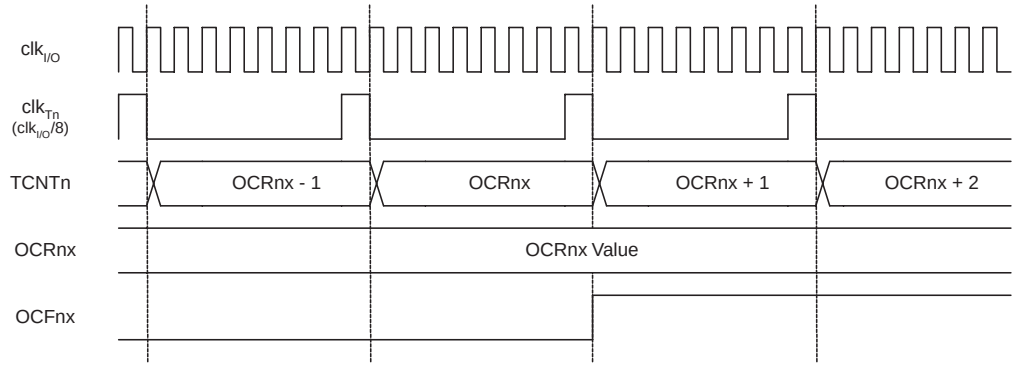
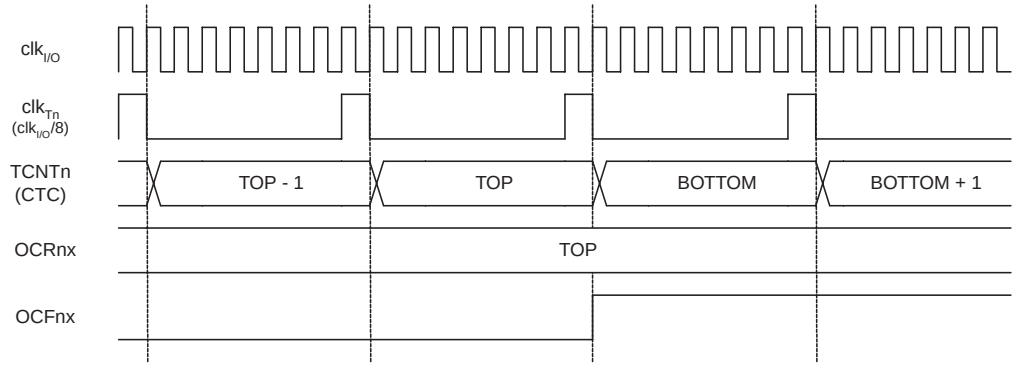


Figure 35 给出了 CTC 模式与快速 PWM 模式下 OCF0A 置位和 TCNT0 清除的情况，其中 OCR0A 为 TOP。

Figure 35. 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

Bit	7	6	5	4	3	2	1	0	
	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
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	

- **Bits 7:6 – COM01A:0: 强制输出比较 A**

这两位控制输出比较引脚 (OC0A) 状态。如果 COM0A1:0 中至少有一位置位，OC0A 输出替代与其相联的 I/O 口的普通端口功能。但要注意必须设置与 OC0A 相应的 DDR 位，来使能输出驱动。

当 OC0A 与引脚相联，COM0A1:0 位的功能由 WGM02:0 位的设置决定。Table 26 给出当 WGM02:0 位设为 CTC 模式 (非 PWM) 时 COM0A1:0 位的功能。

Table 26. 比较输出模式，非 PWM 模式

COM01	COM00	说明
0	0	正常的端口操作，不与 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 27 给出了当 WGM01:0 设置为快速 PWM 模式时 COM01:0 的功能。

Table 27. 比较输出模式，快速 PWM 模式 (1)

COM01	COM00	说明
0	0	正常的端口操作，不与 OC0A 相连接
0	1	WGM02 = 0: 正常的端口操作，不与 OC0A 相连接 WGM02 = 1: 比较匹配发生时 OC0A 取反
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 COM01 置位。此时比较匹配将被忽略 ,而计数到 TOP 时 OC0 的动作继续有效。详细信息请参见 P60“ 快速 PWM 模式 ” 。

Table 28 给出了当 WGM02:0 设置为相位修正 PWM 模式时 COM0A1:0 的功能。

Table 28. 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
0	1	WGM02 = 0: 正常的端口操作，不与 OC0A 相连接 WGM02 = 1: 比较匹配发生时 OC0A 取反
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 P62“ 相位修正 PWM 模式 ”。

- **Bits 5:4 – COM0B1:0: 比较匹配输出模式 B**

这些位决定了比较匹配发生时输出引脚 OC0B 的电平。如果 COM0B1:0 中的一位或全部都置位， OC0B 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 OC0B 连接到物理引脚上时， COM0B1:0 的功能依赖于 WGM01:0 的设置。 Table 26 给出了当 WGM02:0 设置为普通模式或 CTC 模式时 COM0B1:0 的功能。

Table 29. 比较输出模式，非 PWM 模式

COM01	COM00	说明
0	0	正常的端口操作，不与 OC0B 相连接
0	1	比较匹配发生时 OC0B 取反
1	0	比较匹配发生时 OC0B 清零
1	1	比较匹配发生时 OC0B 置位

Table 27 给出了当 WGM02:0 设置为快速 PWM 模式时 COM0B1:0 的功能。

Table 30. 比较输出模式，快速 PWM 模式 (1)

COM01	COM00	说明
0	0	正常的端口操作，不与 OC0B 相连接
0	1	保留
1	0	比较匹配发生时 OC0A 清零，计数到 TOP 时 OC0B 置位
1	1	比较匹配发生时 OC0A 置位，计数到 TOP 时 OC0B 清零

Note: 1. 一个特殊情况是OCR0B等于TOP,且COM0B1置位。此时比较匹配将被忽略,而计数到 TOP 时 OC0B 的动作继续有效。详细信息请参见 P60“ 快速 PWM 模式 ”

Table 28 给出了当 WGM02:0 设置为相位修正 PWM 模式时 COM0B1:0 的功能。

Table 31. 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 OC0B 相连接
0	1	保留
1	0	在升序计数时发生比较匹配将清零 OC0B ；降序计数时发生比较匹配将置位 OC0B
1	1	在升序计数时发生比较匹配将置位 OC0B ；降序计数时发生比较匹配将清零 OC0B

Note: 1. 一个特殊情况是OCR0B等于TOP,且COM0B1置位。此时比较匹配将被忽略,而计数到 TOP 时 OC0B 的动作继续有效。详细信息请参见 P62“ 相位修正 PWM 模式 ”。

- **Bits 3, 2 – Res: 保留**

保留位，该操作返回值为零。

- **Bits 1:0 – WGM01:0: 波形产生模式**

这两位与 TCCR0B 寄存器中的 WGM02 位一起控制计数器的计数顺序,计数器的 TOP 值及产生哪种波形，见 Table 32。T/C 单元支持的工作模式有：正常模式 (计数器)， CTC 模式，及两种 PWM 模式 (见 P59“ 工作模式 ”)。

Table 32. 波形产生位说明

Mode	WGM2	WGM1	WGM0	T/C 工作模式	TOP	OCRx 更新	TOV 标志设置 (1)(2)
0	0	0	0	正常	0xFF	立即	MAX
1	0	0	1	相位修正 PWM	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	立即	MAX
3	0	1	1	快速 PWM	0xFF	TOP	MAX
4	1	0	0	保留	–	–	–
5	1	0	1	相位修正 PWM	OCRA	TOP	BOTTOM
6	1	1	0	保留	–	–	–
7	1	1	1	快速 PWM	OCRA	TOP	TOP

Notes: 1. MAX = 0xFF
2. BOTTOM = 0x00

T/C 控制寄存器 B - TCCR0B

Bit	7	6	5	4	3	2	1	0	
	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
读 / 写	W	W	R	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC0A: 强制输出比较 A**

FOC0A 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，在使用 PWM 时，写 TCCR0B 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。

- **Bit 6 – FOC0B: 强制输出比较 B**

FOC0B 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，在使用 PWM 时，写 TCCR0B 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 OC0B 将按照 COM0B1:0 的设置输出相应的电平。要注意 FOC0B 类似一个锁存信号，真正对强制输出比较起作用的是 COM0B1:0 的设置。

FOC0B不会引发任何中断，也不会在利用OCR0B作为TOP的CTC模式下对定时器进行清零的操作。

读 FOC0B 的返回值永远为 0。

- **Bits 5:4 – Res: 保留**

保留位，返回值为零。

- **Bit 3 – WGM02: 波形产生模式**

见 P65“T/C 控制寄存器 A – TCCR0A” 中说明。

- **Bits 2:0 – CS02:0: 时钟选择**

这三位用于选择 T/C 的时钟源。

Table 33. 时钟选择位说明

CS02	CS01	CS00	说明
0	0	0	无时钟， T/C 不工作
0	0	1	$clk_{I/O}$/1 (没有预分频)
0	1	0	$clk_{I/O}$/8 (来自预分频器)
0	1	1	$clk_{I/O}$/64 (来自预分频器)
1	0	0	$clk_{I/O}$/256 (来自预分频器)
1	0	1	$clk_{I/O}$/1024 (来自预分频器)
1	1	0	时钟由 T0 引脚输入，下降沿触发
1	1	1	时钟由 T0 引脚输入，上升沿触发

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

T/C 寄存器 - TCNT0

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

通过 T/C 寄存器可以直接对计数器的 8 位数据进行读写访问。对 TCNT0 寄存器的写访问将在下一个时钟阻止比较匹配。在计数器运行的过程中修改 TCNT0 的数值有可能丢失一次 TCNT0 和 OCR0x 的比较匹配。

输出比较寄存器 A - OCR0A

Bit	7	6	5	4	3	2	1	0	
	OCR0A[7:0]								OCR0A
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

输出比较寄存器包含一个 8 位的数据，不间断地与计数器数值 TCNT0 进行比较。匹配事件可以用来产生输出比较中断，或者用来在 OC0A 引脚上产生波形。

输出比较寄存器 B - OCR0B

Bit	7	6	5	4	3	2	1	0	
	OCR0B[7:0]								OCR0B
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

输出比较寄存器包含一个 8 位的数据，不间断地与计数器数值 TCNT0 进行比较。匹配事件可以用来产生输出比较中断，或者用来在 OC0B 引脚上产生波形。

T/C 中断屏蔽寄存器 - TIMSK0

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	OCIE0B	OCIE0A	TOIE0	–	TIMSK0
读 / 写	R	R	R	R	R/W	R/W	R/W	R	
初始值	0	0	0	0	0	0	0	0	

- **Bits 7..4, 0 – Res: 保留**

保留位，返回值为零。

- **Bit 3 – OCIE0B: T/C 输出比较匹配 B 中断使能**

当 OCIE0B 和状态寄存器的全局中断使能位 I 都为 ”1” 时，T/C 的输出比较匹配 B 中断使能。当 T/C 的比较匹配发生，即 TIFR0 中的 OCF0B 置位时，中断服务程序得以执行。

- **Bit 2 – 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。

- **Bit 1 – TOIE0: T/C0 溢出中断使能**

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

T/C0 中断标志寄存器 - TIFR0

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	OCF0B	OCF0A	TOV0	–	TIFR0
读 / 写	R	R	R	R	R/W	R/W	R/W	R	
初始值	0	0	0	0	0	0	0	0	

- **Bits 7..4, 0 – Res: 保留**

保留位，返回值为零。

- **Bit 3 – OCF0B: 输出比较标志 0 B**

当 T/C 与 OCR0B(输出比较寄存器 0B) 的值匹配时， OCF0B 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、 OCIE0B(T/C0 比较 B 匹配中断使能) 和 OCF0B 都置位时，中断服务程序得到执行。

- **Bit 2 – 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，OCF0A 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。

- **Bit 1 – 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。

该标志的设置决定于 WGM02:0 位的设置，参见 Table 32， P67“ 波形产生位说明 ”。

T/C 预分频器

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 $f_{CLK\_I/O}/1024$。

预分频器复位

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。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)。

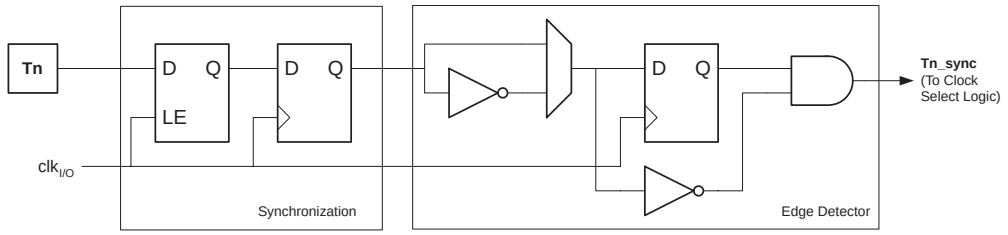
通过复位预分频器来同步 T/C 与程序运行是可能的。

外部时钟源

由 T0 引脚提供的外部时钟源可以用作 T/C 时钟 clk_{T0}。引脚同步逻辑在每个系统时钟周期对引脚 T0 进行采样。然后将同步 (采样) 信号送到边沿检测器。 Figure 36 给出了 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 36. T0 引脚采样



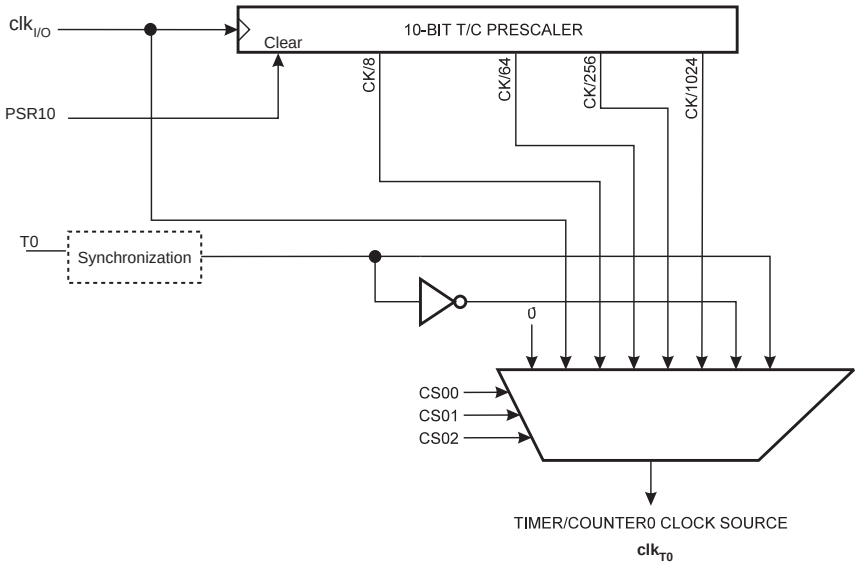
由于引脚上同步与边沿监测电路的存在，引脚 T1/T0 上的电平变化需要延时 2.5 到 3.5 个系统时钟周期才能使计数器进行更新。

禁止或使能时钟输入必须在 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 37. T/C0 预分频器



Note: 1. 输入引脚 (T0) 的同步逻辑见 Figure 36。

通用 T/C 控制寄存器 - GTCCR

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

- **Bit 7 – TSM: T/C 同步模式**

TSM位置"1”激活T/C同步模式。在该模式下，保存写入PSR10位的值，从而保存未用的预分频器复位信号。这保证 T/C 挂起，且配置时不会有提前启动的风险。当 TSM 位写 "0”，PSR10 位由硬件清零，且 T/C 开始计数。

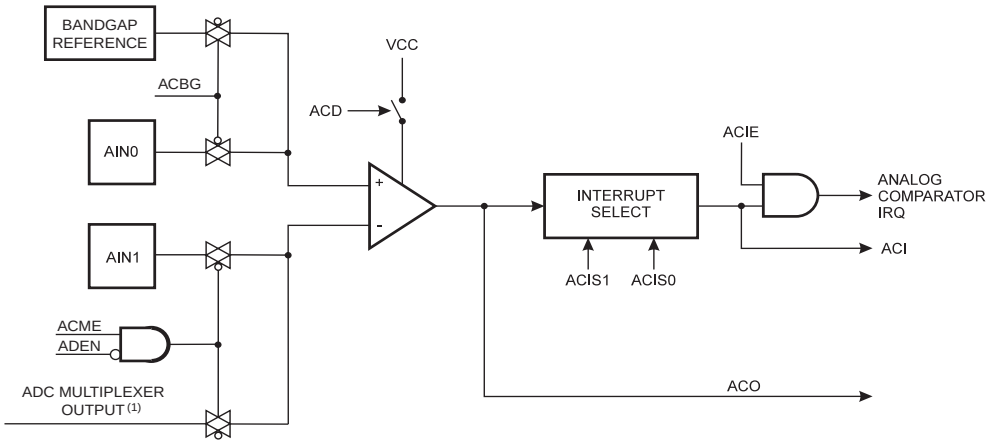
- **Bit 0 – PSR10: T/C0 预分频器复位**

置位时 T/C0 的预分频器复位。在 TSM 未置位时，操作完成后这一位由硬件自动清零。

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 38 为比较器及其外围逻辑电路的框图。

Figure 38. 模拟比较器框图 (2)



Notes: 1. 见 P75Table 35.
2. 模拟比较器引脚配置图见 P1Figure 1 与 P49Table 23。

ADC 控制及状态寄存器 B - ADCSRB

Bit	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	

- **Bit 6 – ACME: 模拟比较器复用器使能**

当该位写 "1" 且 ADC 关闭 (ADCSRA 中 ADEN 为 0)， ADC 复用器选择模拟比较器的负输入。当该位写 "0"，AIN1 作为模拟比较器的负输入。详见 P75“ 模拟比较器多路输入 ”。

模拟比较器控制及状态寄存器 - ACSR

Bit	7	6	5	4	3	2	1	0	
	ACD	ACBG	ACO	ACI	ACIE	–	ACIS1	ACIS0	ACSR
读 / 写	R/W	R/W	R	R/W	R/W	R	R/W	R/W	
初始值	0	0	N/A	0	0	0	0	0	

- **Bit 7 – ACD: 模拟比较器禁用**

ACD 置位时，模拟比较器的电源被切断。可以在任何时候设置此位来关掉模拟比较器。这可以减少器件工作模式及空闲模式下的功耗。改变 ACD 位时，必须清零 ACSR 寄存器的 ACIE 位来禁止模拟比较器中断。否则 ACD 改变时可能会产生中断。

- **Bit 6 – ACBG: 选择模拟比较器的能隙基准源**

ACBG 置 "1" 后，模拟比较器的正极输入由固定能隙基准源所取代。ACBG 清零，AIN0 作为模拟比较器的正极输入。

- **Bit 5 – ACO: 模拟比较器输出**

模拟比较器的输出为同步信号，直接连到 ACO。同步加入 1 - 2 时钟周期的延迟。

- **Bit 4 – ACI: 模拟比较器中断标志**

当比较器的输出事件触发了由 ACIS1 及 ACIS0 定义的中断模式时，ACI 由硬件置位。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。

- **Bit 3 – ACIE: 模拟比较器中断使能**

当 ACIE 位被置 1 且状态寄存器中的全局中断标志 I 也被置位时，模拟比较器中断被激活。否则中断被禁止。

- **Bit 2 – Res: 保留**

保留位，返回值为零。

- **Bits 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 34 给出了不同的设置。

Table 34. 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

当改变 ACIS1/ACIS0 位时，必须通过清除 ACSR 寄存器中断使能位禁用模拟比较中断。否则当位变化时可能会出现中断。

模拟比较器多路输入

可选择 ADC3..0 中任意引脚作为模拟比较器的负极输入。ADC 复用器选择输入端，使用该功能时， ADC 必须关闭。若模拟比较器复用器使能位 (ADCSRB 寄存器中 ACME 位) 设置且 ADC 关闭 (ADCSRA 寄存器中 ADEN 为零)，ADMUX 中 MUX1..0 选择模拟比较器负极输入引脚，如 Table 35 所示。若 ACME 清零或 ADEN 置位， AIN1 作为比较器的负极输入。

Table 35. 模拟比较输入选择

ACME	ADEN	MUX1..0	模拟比较负极输入
0	x	xx	AIN1
1	1	xx	AIN1
1	0	00	ADC0
1	0	01	ADC1
1	0	10	ADC2
1	0	11	ADC3

数字输入禁用寄存器 0 - DIDR0

Bit	7	6	5	4	3	2	1	0	
	–	–	ADC0D	ADC2D	ADC3D	ADC1D	AIN1D	AIN0D	DIDR0
读 / 写	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bits 1, 0 – AIN1D, AIN0D: AIN1, AIN0 数字输入禁用**

当该位写 "1”， AIN1/0 引脚的数字输入缓冲禁用。相应的引脚寄存器位为零。当 AIN1/0 引脚输入为模拟信号且不使用数字输入，该位写 "1” 以降低数字输入缓冲的功耗。

模数转换器

特点

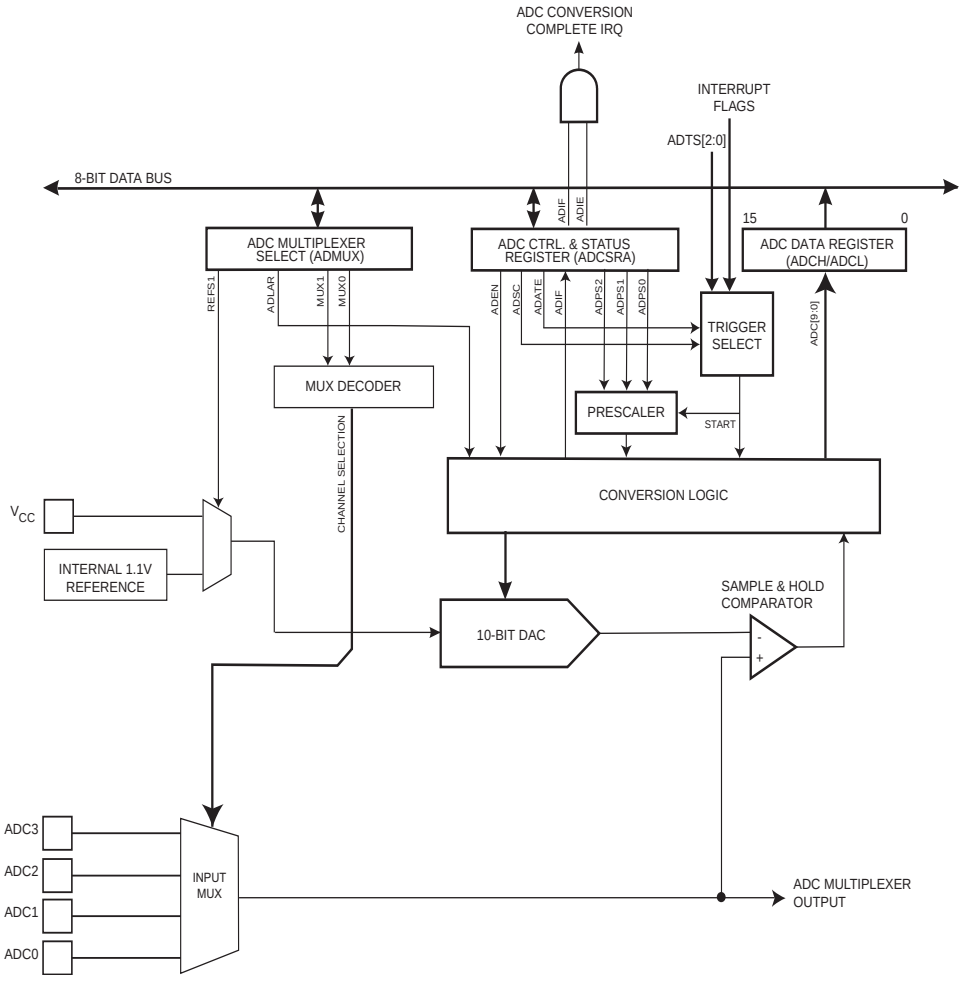
- **10 位精度**
- **0.5 LSB 的非线性度**
- **± 2 LSB 的绝对精度**
- **13 - 260 µs 的转换时间**
- **最大精度达到 15 kSPS**
- **四路复用单端输入通道**
- **可选的向左调整 ADC 读数**
- **0 - V_{CC} 的 ADC 输入电压范围**
- **可选的 1.1V ADC 参考电压**
- **连续转换或单次转换模式**
- **通过中断源自动触发的 ADC 转换启动**
- **ADC 转换结束中断**
- **基于睡眠模式的噪声抑制器**

ATtiny13 有一个 10 位的逐次逼近型 ADC。ADC 与一个 4 通道的模拟多路复用器连接，能对来自端口 B 的四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 39 所示。

标称值为 1.1V 的基准电压或 V_{CC} 位于器件之内。

Figure 39. 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 V_{CC} 或 1.1V 参考电压。

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，都可以作为 ADC 的单端输入。

ADC 由 ADCSR 寄存器的 ADEN 位使能。在 ADEN 设置前，参考电压与输入通道无效。当 ADEN 清零时， ADC 没有功耗，因此建议在进入省电模式前关闭 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。

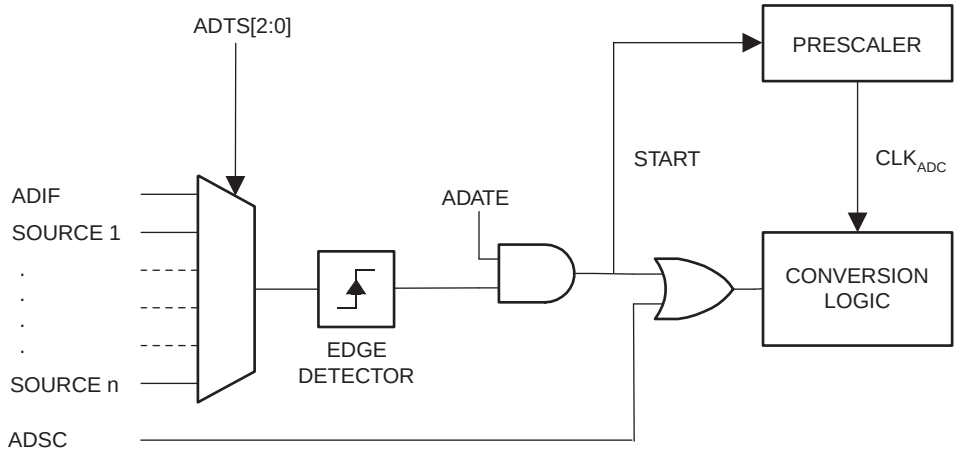
ADC转换结束可以触发中断。即使由于转换发生在读取ADCH与ADCL之间而造成ADC无法访问数据寄存器，并因此丢失了转换数据，中断仍将触发。

启动转换

通过在 ADSC 位写入逻辑 1 来启动转换。该位在转换过程中始终为 1，当转换完成后该位置 0。若在转换过程中选择差分数据通道， ADC 将在执行通道变换前结束转换。

转换可以由不同的源自动触发。设置 ADCSRA 寄存器的 ADATE 位使能自动触发。触发源由 ADCSRB 寄存器的 ADTS 位的设定来决定 (见有关 ADTS 位的说明)。当选择的触发信号正边沿出现， ADC 预分频器复位，转换启动。这样我们可以实现固定转换启动时间间隔。若转换完成后触发信号仍然置位，将不会启动新的转换。如果在转换过程中触发信号出现其他正边沿，则应将其忽略。注意，即使相应的中断禁用或 SREG 中 I 位清零，中断标志仍会设置。这样可在没有中断的情况下触发转换。然而必须清除中断标志以便下一次中断可以触发新的转换。

Figure 40. ADC 自动触发逻辑

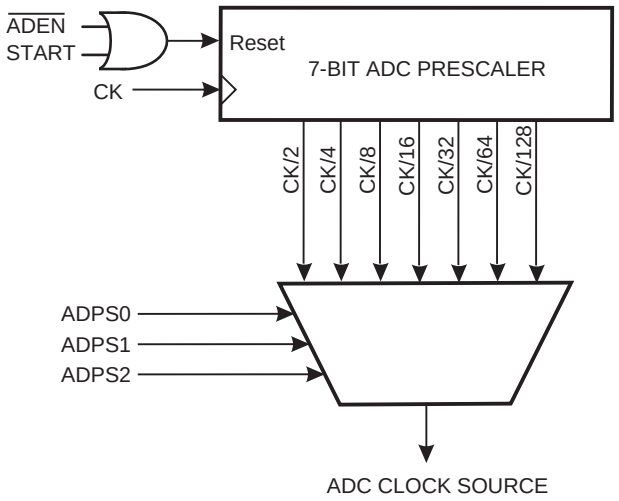


使用 ADC 中断标志作为触发源，可使系统在一次转换结束后立即开始新的转换。ADC 进入连续转换模式，不断采样与更新 ADC 数据寄存器。第一次转换必须通过在 ADCSRA 寄存器的 ADSC 位写 "1" 来启动。在该模式下，转换成功与否取决于 ADC 中断标志与 ADIF 是否清零。

如果使能自动触发，通过在 ADCSRA 寄存器的 ADSC 写 "1" 启动单次转换。ADSC 还可用来检测是否正在进行转换。无论转换是通过何种方式启动，在转换进行时， ADSC 值为 "1" 。

预分频与转换时序

Figure 41. ADC 预分频器



默认情况下，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。若需要低于 10 位的精度， ADC 输入时钟频率要高于 200 kHz，以达到高采样率。

ADC模块包括一个预分频器，它可以产生可接受的ADC时钟。ADCSR寄存器的ADPS位用于从片内产生一个超过 100 kHz 的适当的 ADC 时钟输入信号。预分频器从 ADCSR 寄存器的 ADEN 位置位启动 ADC 起开始计数。ADEN 置位时预分频器保持运转，当 ADEN 为低时预分频器复位。

转换在 ADCSR 的 ADSC 位设置后的上升沿开始。如果使用差分通道，转换在 ADEN 设置后的第二个上升沿启动。

正常转换需要 13 个 ADC 时钟周期。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 14.5 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。

当使用自动触发时，当触发事件出现时，预分频器复位。这保证从触发到转换启动有一个给定的延时。在该模式下，触发源信号上升沿后有两个 ADC 时钟周期的采样与保持时间。为同步还需要三个额外的 CPU 时钟周期。

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。转换时间见 Table 36。

Figure 42. ADC 时序图，第一次转换（单次转换模式）

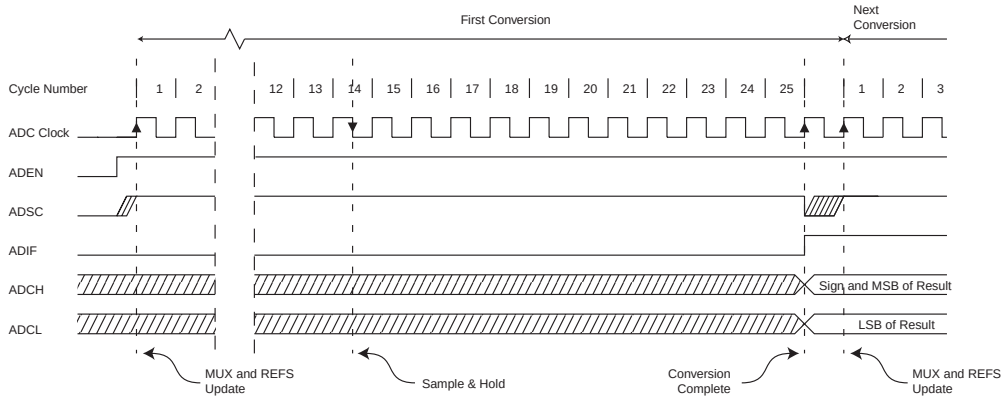


Figure 43. ADC 时序图，单次转换

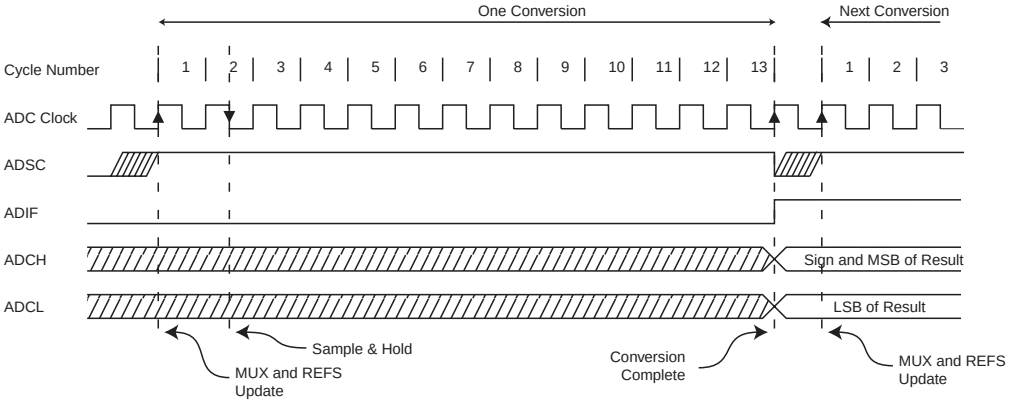


Figure 44. ADC 时序图，自动触发转换

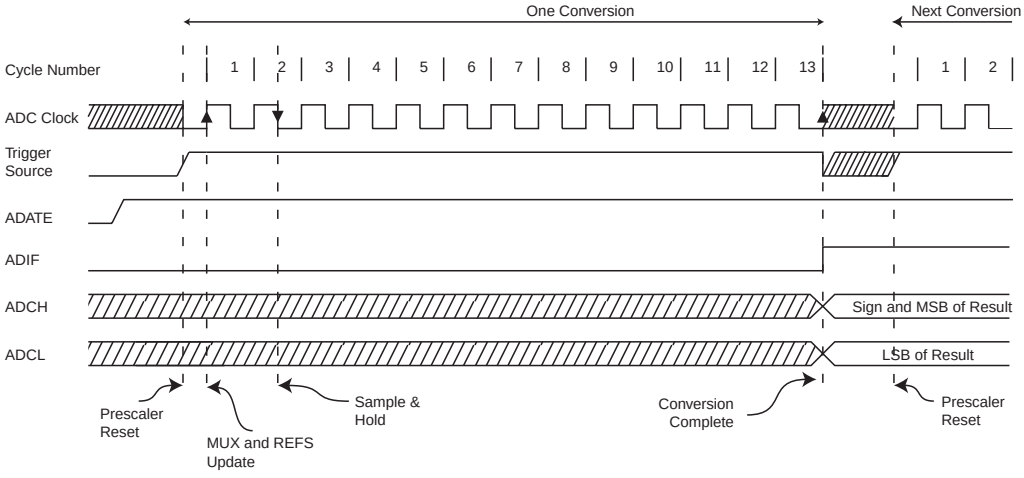


Figure 45. ADC 时序图，连续转换

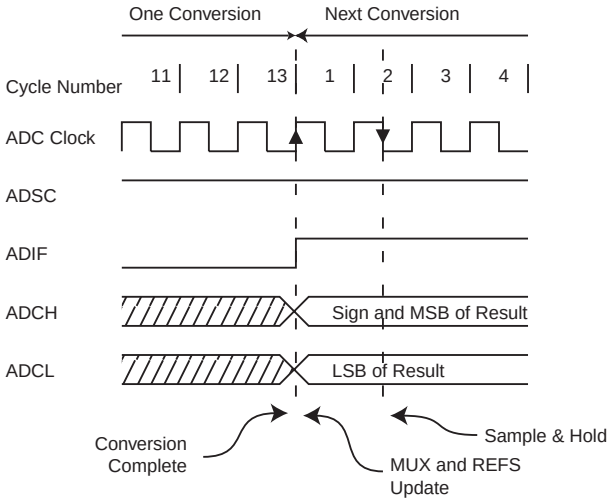


Table 36. ADC 转换时间

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	13.5	25
正常转换	1.5	13
自动触发转换	2	13.5

变化通道或基准选择

ADMUX寄存器的MUXn与REFS1:0位的单缓冲为CPU可随机访问的临时寄存器。这确保通道及基准选择只能在转换稳定时才进行。通道与基准选择在转换启动前一直更新。一旦转换启动，通道及基准选择锁定，以保证 ADC 有效采样时间。在转换结束前的最后一个ADC时钟周期前恢复更新。注意,在ADSC写入后的第一个ADC时钟上升沿转换启动。建议用户在 ADSC 写入一个 ADC 时钟周期后再对 ADMUX 写入新的通道或基准选择值。

如果使用自动触发模式,无法确定触发事件的准确时间。当更新 ADMUX 寄存器时要特别当心，以便控制由新的设置所影响的转换。

若 ADATE 与 ADEN 均为零，中断会随时发生。若此时改变 ADMUX 寄存器值，则无法确定转换是基于哪种设置。在以下情况下可实现 ADMUX 安全更新：

1. 当 ADATE 或 ADEN 清零
2. 转换时，触发完成至少一个 ADC 时钟周期后
3. 转换完成后，在中断标志作为触发源被清除前。

当 ADMUX 在以上任一种情况下被更新后，新设置将在下一次 ADC 转换中有效。

ADC 输入通道

当改变通道选择时，用户应注意以下原则，以保证选择正确的通道：

单次转换模式下，在启动转换前选择通道。通道选择在 ADSC 写入 "1” 后一个 ADC 时钟周期后可能会改变。最简单的方法是等到转换结束后改变通道设置。

连续转换模式下，在启动转换前选择通道。通道选择在 ADSC 写入 "1” 后一个 ADC 时钟周期后可能会改变。最简单的方法是等到转换结束后改变通道设置。由于下一次转换已经自动启动，其结果也反映前面的通道选择。其后的转换将反映新的通道选择。

ADC 电压基准

ADC 参考电压(V_{REF})给出ADC转换范围。单端通道超过V_{REF} 其结果接近0x3FF。V_{REF} 可为 V_{CC} ,或内部基准电压 1.1V ,或外部 AREF 引脚电压。改变基准电压源后的第一次 ADC 转换结果可能不准确，建议用户舍弃。

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。若在 ADC 转换结束前其他中断将 CPU 唤醒，则会执行中断，当 ADC 转换结束后产生 ADC 转换结束中断请求。 CPU 在下一个休眠指令执行前将保持正常模式。

注意，当进入休眠模式时 (除空闲模式与 ADC 噪声抑制模式)ADC 不会自动关闭。建议用户在进入休眠模式前对 ADEN 写 "0”，以降低功耗。

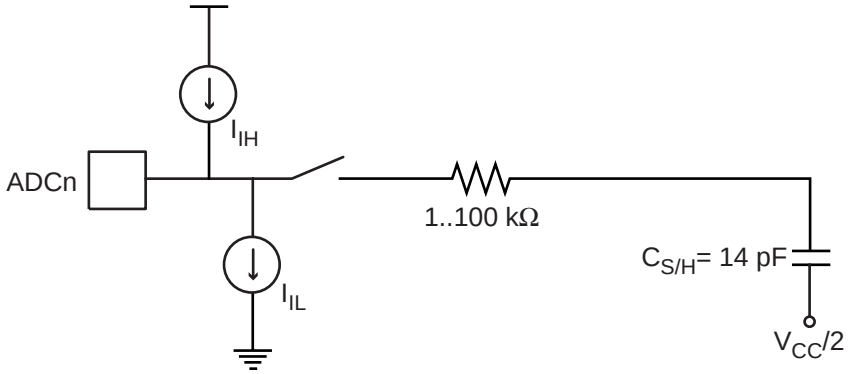
模拟输入电路

单端通道的模拟输入电路见 Figure 46.。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 46. 模拟输入电路



模拟噪声抑制技术

设备内部及外部的数字电路都会产生电磁干扰 (EMI)，从而影响模拟测量的精度。如果转换精度要求较高，那么可以通过以下方法来减少噪声：

1. 模拟通路越短越好。保证模拟信号线位于模拟地之上，并使它们与高速切换的数字信号线分开。
2. 使用 ADC 噪声抑制器来降低来自 CPU 的干扰噪声。
3. 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。

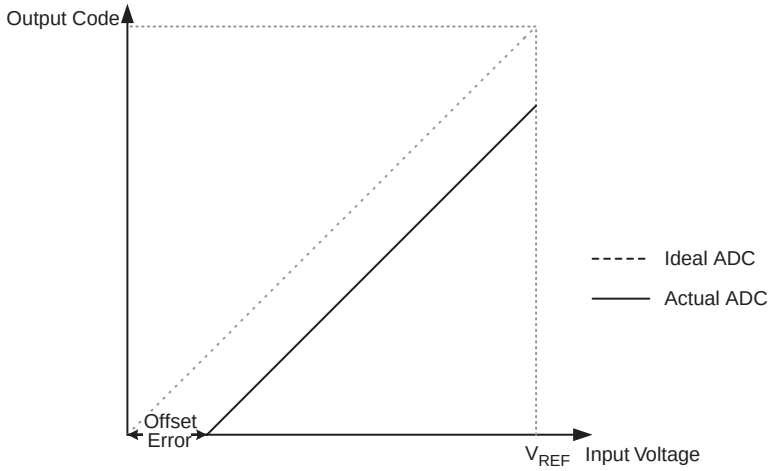
ADC 精度定义

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：

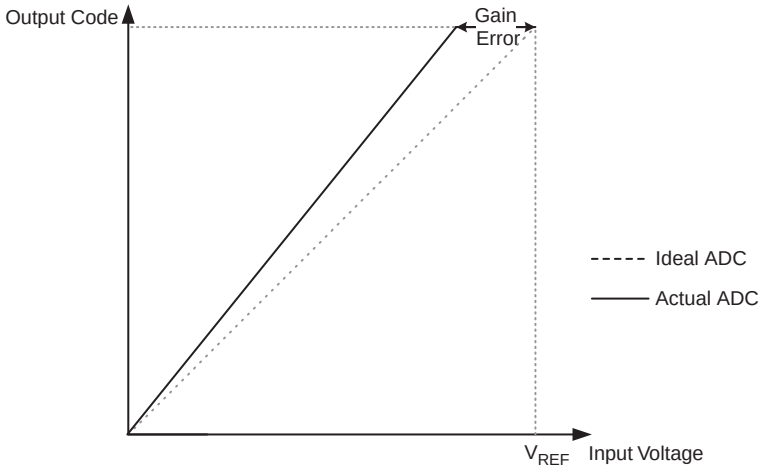
- 偏移：第一次转换 (0x000 到 0x001) 与理想转换 (0.5 LSB) 之间的偏差。理想情况：0 LSB。

Figure 47. 偏移误差



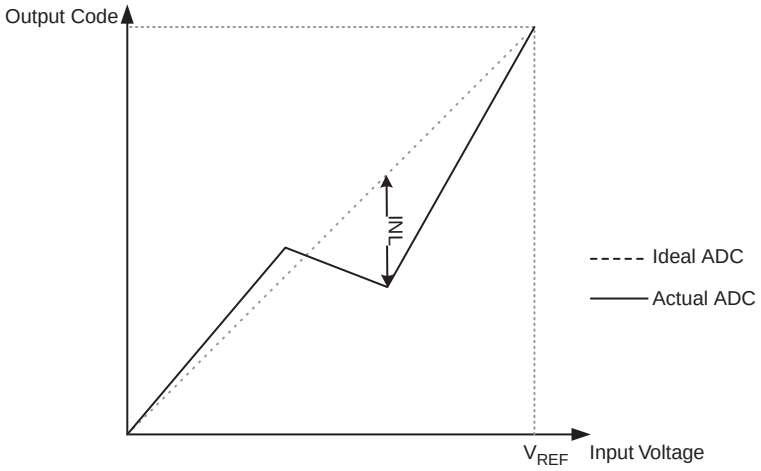
- 增益误差：调整偏差之后，最后一次转换 (0x3FE 到 0x3FF) 与理想情况 (最大值以下 1.5 LSB) 之间的偏差即为增益误差。理想值为 0 LSB。

Figure 48. 增益误差



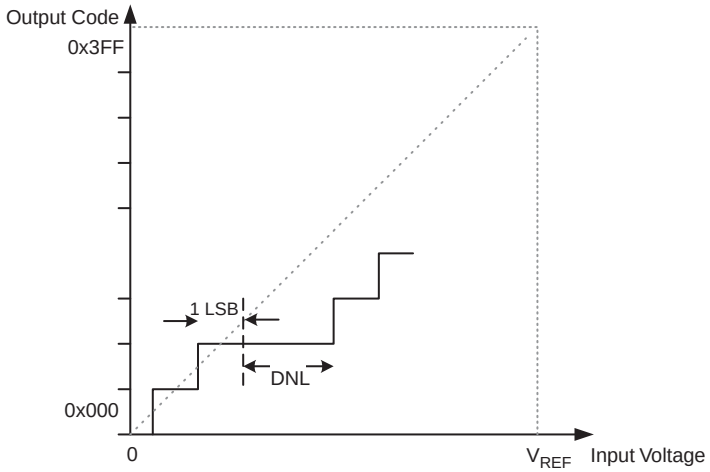
- 整体非线性 (INL)：调整偏移及增益误差之后，所有实际转换与理想转换之间的最大误差即为 INL。理想值：0 LSB。

Figure 49. 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 50. 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 P86Table 37 与 P86Table 38)。0x000 代表模拟地电平， 0x3FF 代表所选参考电压的数值减去 1LSB。

ADC 多工选择寄存器 - ADMUX

Bit	7	6	5	4	3	2	1	0	
	–	REFS0	ADLAR	–	–	–	MUX1	MUX0	ADMUX
读 / 写	R	R/W	R/W	R	R	R	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – Res: 保留**

保留位，返回值为零。

- **Bit 6 – REFS0: 参考电压选择**

如 Table 37 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。

Table 37. ADC 参考电压选择

REFS0	参考电压选择
0	V_{CC} 作为模拟参考电压
1	片内基准电压

- **Bit 5 – ADLAR: ADC 转换结果 左对齐**

ADLAR影响ADC转换结果在ADC数据寄存器中的存放形式。ADLAR置位时转换结果为左对齐，否则为右对齐。 ADLAR 的改变将立即影响 ADC 数据寄存器的内容，不论是否有转换正在进行。关于这一位的完整描述请见 P87“ADC 数据寄存器 – ADCL 及 ADCH” 。

- **Bits 4:2 – Res: 保留**

保留位，返回值为零。

- **Bits 1:0 – MUX1:0: 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，详见 Table 38。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 38. 输入通道选择

MUX1..0	单端输入
00	ADC0 (PB5)
01	ADC1 (PB2)
10	ADC2 (PB4)
11	ADC3 (PB3)

ADC 控制和状态寄存器 A - ADCSRA

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – ADEN: ADC 使能**

ADEN置位即启动ADC ,否则ADC功能关闭。在转换过程中关闭ADC将立即中止正在进行的转换。

- **Bit 6 – ADSC: ADC 开始转换**

在单次转换模式下，ADSC 置位将启动一次 ADC 转换。在连续转换模式下，ADSC 置位将启动首次转换。第一次转换 (在 ADC 启动之后置位 ADSC , 或者在使能 ADC 的同时置位 ADSC) 需要 25 个 ADC 时钟周期，而不是正常情况下的 13 个。第一次转换执行 ADC 初始化的工作。

在转换进行过程中读取 ADSC 的返回值为 "1”，直到转换结束。ADSC 清零不产生任何动作。

- **Bit 5 – ADATE: ADC 自动触发使能**

ADATE 置位将启动 ADC 自动触发功能。触发信号的上跳沿启动 ADC 转换。触发信号源通过 ADCSRB 寄存器的 ADC 触发信号源选择位 ADTS 设置。

- **Bit 4 – ADIF: ADC 中断标志**

在 ADC 转换结束，且数据寄存器被更新后， ADIF 置位。如果 ADIE 及 SREG 中的全局中断使能位 I 也置位，ADC 转换结束中断服务程序即得以执行，同时 ADIF 硬件清零。此外，还可以通过向此标志写 1 来清 ADIF。要注意的是，如果对 ADCSRA 进行读 - 修改 - 写操作，那么待处理的中断会被禁止。这也适用于 SBI 及 CBI 指令。

- **Bit 3 – ADIE: ADC 中断使能**

若 ADIE 及 SREG 的位 I 置位， ADC 转换结束中断即被激活。

- **Bits 2:0 – ADPS2:0: ADC 预分频器选择位**

由这几位来确定系统时钟与 ADC 输入时钟之间的分频因子。

Table 39. ADC 预分频选择

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

ADC 数据寄存器 - ADCL 及 ADCH

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
读 / 写	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
初始值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH

	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
读 / 写	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
初始值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADC 转换结束后，转换结果存于这两个寄存器之中。

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，详见 P85“ADC 转换结果 ”。

ADC 控制与状态寄存器 B - ADCSRB

Bit	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	

- **Bits 7, 5..3 – Res: 保留**

保留位，返回值为零。

- **Bits 2:0 – ADTS2:0: ADC 自动触发源**

若 ADCSRA 寄存器的 ADATE 位写入 "1”,ADTS2:0 选择 ADC 转换的触发源 。若 ADATE 清零， ADTS2:0 的设置无效。被选中断标志的上升沿触发转换。注意，从一个触发源的清除到另一个触发源的设置过程中，将产生一个下跳沿。一旦设置 ADCSRA 寄存器的 ADEN 位，将会启动一次转换。即使 ADC 中断标志设置，变化到连续模式 (ADTS[2:0]=0) 时不会触发转换。

Table 40. 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	T/C 比较匹配 A
1	0	0	T/C 溢出
1	0	1	T/C 比较匹配 B
1	1	0	引脚变化中断请求

数字输入禁用寄存器 0 - DIDR0

Bit	7	6	5	4	3	2	1	0	
	–	–	ADC0D	ADC2D	ADC3D	ADC1D	AIN1D	AIN0D	DIDR0
读 / 写	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bits 5..2 – ADC3D..ADC0D: ADC3..0 数字输入禁用**

当该位写 "1”,禁用相应的 ADC 引脚数字输入缓冲。此时相应的引脚寄存器始终读为 "0”。当 ADC3..0 输入为模拟信号，且不需要数字信号时，该位应置位，以降低数字输入缓冲器的功耗。

片上调试系统

特点

- **完全的程序流控制**
- **仿真芯片上所有的模拟和数字功能，除了 RESET 引脚**
- **实时操作**
- **支持符号调试 (C 与汇编级，或其它 HLL)**
- **没有限制的程序断点数 (使用软件断点)**
- **非插入式操作**
- **与实际器件相同的电气特性**
- **自动配置系统**
- **高速操作**
- **编程非易失性存储器**

概述

debugWIRE 片上调试系统使用单线双向接口来控制程序流，在 CPU 中执行 AVR 指令，对不同的非易失性存储器进行编程。

物理接口

当 debugWIRE 使能熔丝位 DWEN 被编程且锁定位未编程时，目标器件中的 debugWIRE 系统被激活。RESET 端口引脚配置为上拉使能的线与 (开漏) 双向 I/O，成为目标与仿真器间的联系通路。

Figure 51. The debugWIRE 设置

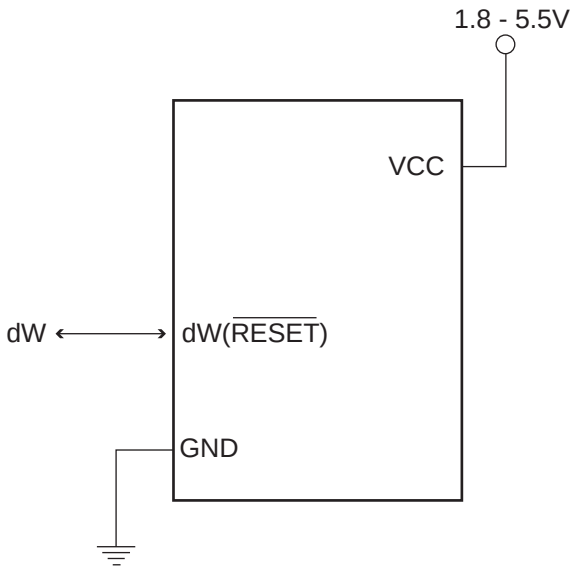


Figure 51 给出 debugWIRE 使能的目标 MCU 及仿真连接器的示意图。系统时钟不受 debugWIRE 的影响，只由 CKSEL 熔丝位决定。

设计使用 debugWIRE 的系统时，必须进行下面的检查：

- dW/(RESET) 的上拉电阻不得小于 10kΩ。 debugWIRE 并不需要上拉电阻
- 将 RESET 引脚与 V_{CC} 直接连接将无法工作
- 使用 debugWIRE 时必须断开与 RESET 引脚连接的电容
- 必须断开所有的外部复位源

软件断点

debugWIRE 通过 AVR 断点指令来设置程序存储器断点。在 AVR Studio® 设置一个断点将在程序存储器中插入 BREAK 指令。被 BREAK 指令所替代的指令将被保存。程序继续运

行时，保存的指令得到执行，然后继续执行其他指令。断点也可以通过在程序中插入 BREAK 指令进行手工设置。

每次断点改变后 Flash 必须要重新编程。这由 AVR Studio® 通过 debugWIRE 接口自动处理。断点的使用会降低 Flash 数据记忆时间。调试用的器件不能发给最终客户。

debugWIRE 的局限

debugWIRE 通讯引脚 (dW) 与外部复位 (RESET) 共用同一引脚。因此使能 debugWIRE 之后，系统不支持外部复位源。

当程序在 CPU 中全速运行时， debugWIRE 系统精确的仿真所有的 I/O 口功能；当 CPU 停止工作时，通过调试器访问某些 I/O 寄存器时要注意。详见 debugWIRE 文档。

DWEN 熔丝位的编程使部分时钟系统在所有的休眠模式下都保持运行。这会增加器件休眠模式的功耗。因此不使用 debugWire 时应该禁用 DWEN 熔丝位。

I/O 存储器中与 debugWIRE 相关的寄存器

下面说明在 debugWire 中用到的寄存器。

debugWire 数据寄存器 - DWDR

Bit	7	6	5	4	3	2	1	0	
	DWDR[7:0]								DWDR
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	

DWDR 寄存器为在 MCU 中运行的程序与调试器提供了通信通路。该寄存器只能由 debugWIRE 访问且不能在通常操作中作为通用寄存器使用。

Flash 自编程

器件为通过 MCU 本身来下载和上载程序代码提供了一个自编程机制。自编程可以使用任何器件可用的数据接口和相关的协议来获得代码并把代码 (程序) 写入程序存储器。

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之前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，在页擦除前填充缓冲器

- 填充临时页缓冲器
- 执行页擦除操作
- 执行页写操作

方案 2，在页擦除之后填充缓冲器

- 执行页擦除操作
- 填充临时页缓冲器
- 执行页写操作

如果只需要改变页中的一部分，擦除前必须保存页中的其它部分 (例如保存于临时页缓冲器之中)，再重新写入。使用方案 1 时， Boot Loader 提供了有效的读 - 该 - 写操作，允许用户先读页，做必要的改变，再写回修改后的数据；若使用方案 2，由于页已被擦除，因此不可能在加载数据时读取旧的数据。临时页缓冲器可以进行随机访问。进行页擦除与页写操作时要确保地址是相同的。

通过 SPM 完成页擦除

执行页擦除操作首先需要设置 Z 指针的地址信息，然后将 “00000011” 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。

- 在擦除操作过程中 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。

执行页写操作

执行页写操作首先需要设置 Z 指针的地址信息，然后将 “00000101” 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。

- 页写过程中 CPU 停止

在自编程过程中寻址 Flash

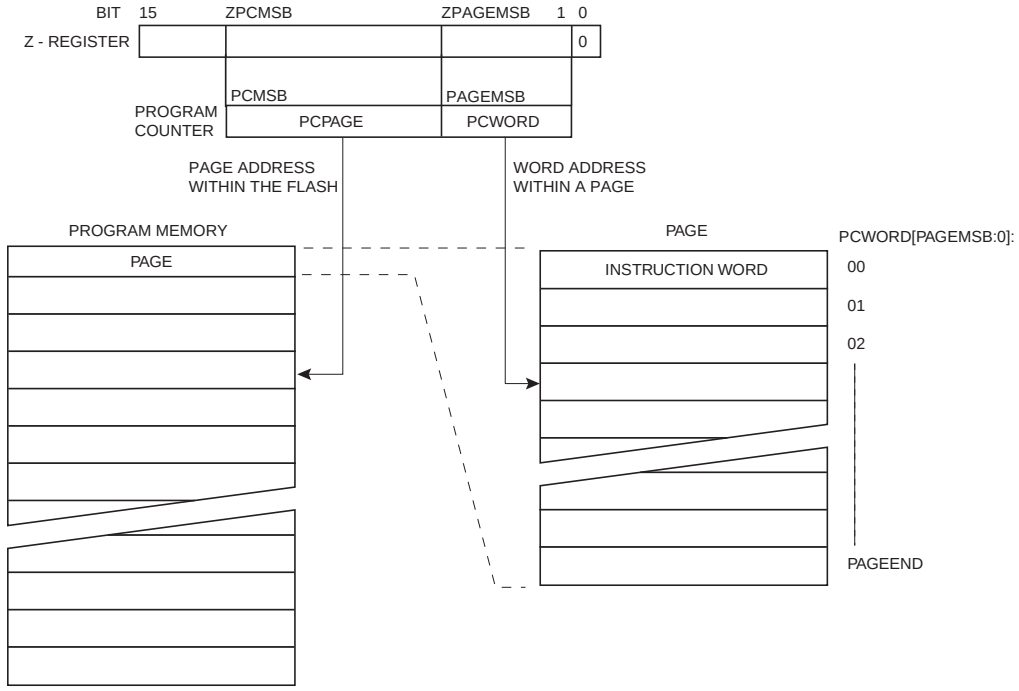
Z 指针用来寻址 SPM 命令。

Bit	15	14	13	12	11	10	9	8
ZH (R31)	Z15	Z14	Z13	Z12	Z11	Z10	Z9	Z8
ZL (R30)	Z7	Z6	Z5	Z4	Z3	Z2	Z1	Z0
	7	6	5	4	3	2	1	0

由于 Flash 存储器是以页的形式组织 (P99Table 46) 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 52 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。

LPM指令也使用Z指针来保存地址。由于这个指令的寻址逐字节地进行，所以Z指针的LSB位 (位 Z0) 也使用到了。

Figure 52. SPM(1) 的寻址



Note: 1. Figure 52 中所用的不同的变量在 P99Table 46 列出。

存贮程序存储器 (SPM) 控制和状态寄存器 - SPMCSR

SPMCSR 包括了控制 Boot Loader 操作所需的控制位。

Bit	7	6	5	4	3	2	1	0	
	–	–	–	CTPB	RFLB	PGWRT	PGERS	SELFPRGEN	SPMCSR
读 / 写	R	R	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bits 7..5 – Res: 保留**

保留位，返回值为零。

- **Bit 4 – CTPB: 清除临时页缓冲**

当填充临时页缓冲时，若写入 CTPB 位，临时页缓冲将被清除，数据将会丢失。

- **Bit 3 – RFLB: 读熔丝与锁定位**

在 SPMCSR 寄存器中的 RFLB 与 SELFPRGEN 位设置后的三个时钟周期内，LPM 将锁定位或熔丝位 (取决于 Z 指针的 Z0) 读入目的寄存器 ,详见 P95“EEPROM 写操作阻止对 SPMCSR 寄存器的写操作 ” 。

- **Bit 2 – PGWRT: 页写入**

如果这一位和 SELFPRGEN 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，将数据存入临时缓冲器中。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。在整个页写操作过程中 CPU 停止。

- **Bit 1 – PGERS: 页擦除**

如果这一位和 SELFPRGEN 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。在整个页擦除操作过程中 CPU 停止。

- **Bit 0 – SELFPRGEN: 自编程使能**

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 CTPB、 RFLB、 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 SELFPRGEN 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， SELFPRGEN 自动清零。在页擦除和页写过程中 SELFPRGEN 保持为高直到操作完成。

在低五位中写入除 “10001”、“01001”、“00101”、 “00011” 或 “00001” 之外的任何组合都无效。

EEPROM 写操作阻止对 SPMCSR 寄存器的写操作

EEPROM 写操作会阻塞对 Flash 的编程，也会阻塞对熔丝位和锁定位的读操作。建议用户在对 SPMCSR 寄存器进行写操作之前首先检查 EECR 寄存器的状态位 EEWE，确保此位已被清除。

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 RFLB 和 SELFPRGEN。在 RFLB 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，RFLB 和 SELFPRGEN 位将自动硬件清零。RFLB 和 SELFPRGEN 清零后，LPM 将按照指令手册中所描述的那样工作。

Bit	7	6	5	4	3	2	1	0
Rd	–	–	–	–	–	–	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RFLB 和 SELFPRGEN。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 P98Table 45 。

Bit	7	6	5	4	3	2	1	0
Rd	FLB7	FLB6	FLB5	FLB4	FLB3	FLB2	FLB1	FLB0

类似的，读取熔丝位高位字节时，需要将 0x0003 赋予给 Z 指针并且置位 SPMCSR 寄存器的 RFLB 和 SELFPRGEN。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 P98Table 44 。

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

被编程的熔丝位 / 锁定位的读返回值为 "0”。未被编程的熔丝位 / 锁定位的读返回值为 "1”。

防止 Flash 损毁

V_{CC} 低于工作电压时 ,CPU 和 Flash 正常工作无法保证 ,Flash 的内容可能受到破坏。这个问题在板级系统的独立 Flash 中一样存在。所以也要采用同样的解决方案。

电压太低时有两种情况可以破坏 Flash 内容。第一，Flash 写过程需要一个最低电压。第二，电压太低时 CPU 本身会错误地执行指令。

通过遵循以下设计建议可以避免 Flash 被破坏 (采用其中之一就足够了) :

1. 电源电压不足期间，保持 AVR RESET 为低：采用的方式为：如果工作电压与检测电平相匹配，可以使能 BOD 功能；否则可以使用外部复位保护电路。如果在写操作进行中发生了复位，只要电源电压足够，写操作还会完成。
2. 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 Flash 被无意识得修改掉。

使用 SPM 时的 Flash 编程时间

片内校准的 RC 振荡器用于 Flash 寻址时序控制。Table 41 给出了 CPU 访问 Flash 的典型编程时间。

Table 41. SPM 编程时间

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

存储器编程

本节说明 ATtiny13 存储器的不同编程方法。

程序存储器和数据存储器锁定位

The ATtiny13 提供了 2 个锁定位，根据其已编程 (“0”) 还是未编程 (“1”) 的情况可以获得 Table 43 列出的附加性能。锁定位只能通过芯片擦除命令擦写为 “1”。

当 DWEN 熔丝位编程后 ,即使设置锁定位 ,程序存储器也可通过 debugWIRE 接口读出 。但当锁定位有安全要求时，应将 DWEN 熔丝位清除以禁用 debugWIRE。

Table 42. 锁定位字节 (1)

锁定位字节	位号	说明	默认值
	7	–	1 (未编程)
	6	–	1 (未编程)
	5	–	1 (未编程)
	4	–	1 (未编程)
	3	–	1 (未编程)
	2	–	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 43. 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 Flash 和 EEPROM 的进一步编程被禁止，熔丝位被锁定。(1) debugWire 禁用
3	0	0	在并行和串行编程模式中 Flash 和 EEPROM 的进一步编程及验证被禁止，锁定位和熔丝位被锁定 (1) debugWire 禁用

Notes: 1. 在编程 LB1 和 LB2 前先编程熔丝位
2. “1” 表示未编程 , “0” 表示已编程。

熔丝位

ATtiny13 有两个熔丝位字节。Table 44 与 Table 45 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 44. 熔丝位高位字节

熔丝位高位字节	位号	说明	默认值
–	7	–	1 (未编程)
–	6	–	1 (未编程)
–	5	–	1 (未编程)
SELFPRGEN	4	自编程使能	1 (未编程)
DWEN(3)	3	debugWire 使能	1 (未编程)
BODLEVEL1(1)	2	BOD 触发电平	1 (未编程)
BODLEVEL0(1)	1	BOD 触发电平	1 (未编程)
RSTDISBL(4)	0	外部复位禁用	1 (未编程)

Notes: 1. BODLEVEL 熔丝位解码，见 P32Table 13
2. RSTDISBL 与 DWEN 熔丝位的说明，请见 P48“ 端口 B 的第二功能 ” 。
3. 当锁定位有安全要求时 ,DWEN 不许编程 ,见 P97“ 程序存储器和数据存储器锁定位 ” 。
4. 当对 RSTDISBL 熔丝位编程，必须使用高电压串行编程来改变熔丝位以执行更深层的编程。

Table 45. 熔丝位低位字节

熔丝位低位字节	位号	说明	默认值
SPIEN(1)	7	使能串行编程与数据下载	0 (已编程，SPI 编程使能)
EESAVE	6	在芯片擦除时保存 EEPROM 存储器值	1 (未编程，EEPROM 不保存)
WDTON(2)	5	打开看门狗定时器	1 (未编程)
CKDIV8(5)	4	时钟 8 分频	0 (已编程)
SUT1	3	选择启动时间	1 (未编程)(3)
SUT0	2	选择启动时间	0 (已编程)(3)
CKSEL1	1	选择时钟源	1 (未编程)(4)
CKSEL0	0	选择时钟源	0 (已编程)(4)

Notes: 1. 在 SPI 编程模式下， SPIEN 熔丝位不可访问。
2. 详见 P37“ 看门狗定时器控制寄存器 - WDTCR” 。
3. 对于默认时钟源， SUT1..0 的默认值给出最大的启动时间。详细内容见 P22Table 5。
4. CKSEL1..0 的默认设置导致了片内 RC 振荡器运行于 9.6 MHz。详细内容见 P22Table 4。
5. 详见 P24“ 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。在正常工作模式中器件上电时熔丝位也被锁存。

标识字节

所有的 Atmel 微控制器都具有一个三字节的标识代码用来区分器件型号。这个代码可以通过串行和并行模式读取，也可以在芯片被锁定时读取。这三个字节分别存储于三个独立的地址空间。

ATtiny13 的表示字节为：

1. 0x000: 0x1E (表示由 Atmel 公司生产)
2. 0x001: 0x90 (表示芯片包含 1KB Flash 存储器)
3. 0x002: 0x07 (当 0x001 字节的内容为 0x90 时表示这是 ATtiny13)

校准字节

ATtiny13 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。

页尺寸

Table 46. Flash 中的页号及页中的字号

Flash 尺寸	页尺寸	PCWORD	页号	PCPAGE	PCMSB
512 字 (1K 字节)	16 字	PC[3:0]	32	PC[8:4]	8

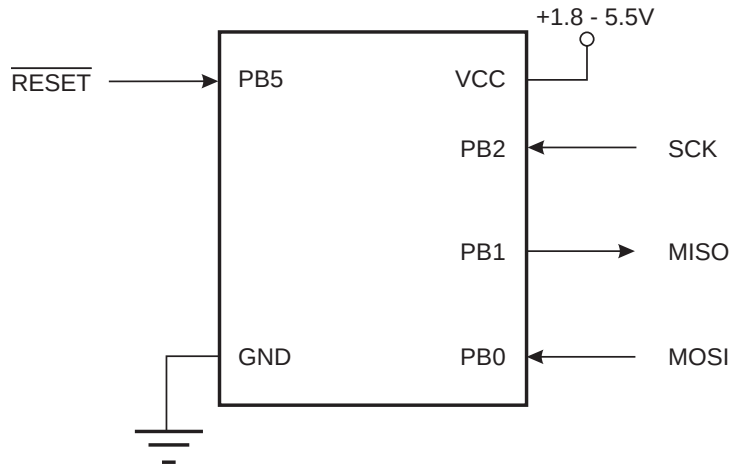
Table 47. EEPROM 中的页号及页中的字号

EEPROM 尺寸	页尺寸	PCWORD	页号	PCPAGE	EEAMSB
64 字节	4 字节	EEA[1:0]	16	EEA[5:2]	5

串行下载

当 RESET 为低电平时，可以通过串行 SPI 总线对 Flash 及 EEPROM 进行编程。串行接口包括 SCK、 MOSI(输入) 及 MISO(输出)。 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。P100Table 48 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 53. 串行编程及校验 (1)



Notes: 1. 如果芯片由片内振荡器提供时钟，那么就不用在 CLKI 引脚上连接时钟源。

Table 48. 引脚映射串行编程

符号	引脚	I/O	说明
MOSI	PB0	I	串行数据输入
MISO	PB1	O	串行数据输出
SCK	PB2	I	串行时钟

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 (仅在串行模式下) , 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 ATtiny13 串行写入数据时，数据在 SCK 的上升沿得以锁定。

从ATtiny13读取数据时，数据在SCK的下降沿输出。时序细节见 Figure 54 与 Figure 55。

在串行编程模式下对 ATtiny13 进行编程及校验时，应遵循以下的步骤 (见 Table 50 中的 4 字节指令格式) :

1. 上电顺序：
   在 RESET 及 SCK 为 "0" 时，向 V_{CC} 及 GND 供电。在一些系统中，编程器不能保证在上电时 SCK 保持为低。在这种情况下，SCK 拉低之后应在 RESET 加一正脉冲，而且这个脉冲至少要维持 2 个 CPU 时钟周期。
2. 上电之后等待至少 20 ms，然后向 MOSI 引脚输入串行编程使能指令以使能串行编程。
3. 通信不同步将造成串行编程指令不工作。同步之后，在发送编程使能指令的第三个字节时，第二个字节的内容 (0x53) 将被反馈回来。不论反馈的内容正确与否，指令的 4 个字节必须全部传输。如果 0x53 未被反馈，则需要向 RESET 提供一个正脉冲以开始新的编程使能指令。
4. Flash 的编程以一次一页的方式进行。在执行加载程序存储页指令时，通过 5LSB 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。如果不使用轮询的方式，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 49.)。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. **A:** 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。如果不使用轮询的方式，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 49) 。对于全片擦除之后的芯片，数据为 0xFF 的不需要编程。
   B: EEPROM 编程是以页为单位的。存储器页通过载入 EEPROM 存储器页指令同时得到 2 LSB 的地址与数据载入一个字节。 EEPROM 存储器页保存是通过载入写 EEPROM 存储器页指令及 4 MSB 地址来实现。当使用 EEPROM 页访问时，只有位于载入 EEPROM 页指令处的字节改变。其余部分不变。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 47.) 。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}。

Table 49. 写下一个 Flash 或 EEPROM 单元之前的最小等待时间

符号	最小等待时间
t_{WD_FLASH}	4.5 ms
t_{WD_EEPROM}	4.0 ms
t_{WD_ERASE}	4.0 ms
t_{WD_FUSE}	4.5 ms

Figure 54. 串行编程波形图

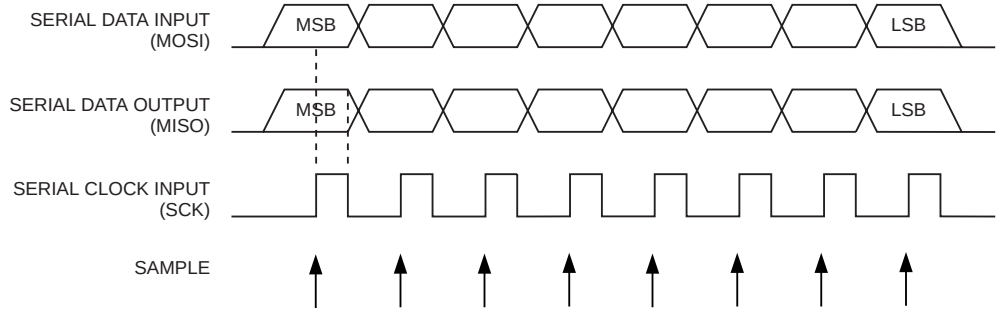


Table 50. 串行编程指令集

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	0000 000**a**	**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	xxxx **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	0000 000**a**	**bbbb** xxxx	xxxx xxxx	在地址 **a**:**b** 加载程序存储页
读 EEPROM 存储器	1010 0000	000x xxxx	xx**bb bbbb**	**oooo oooo**	从 EEPROM 的地址 **b** 处读出数据
写 EEPROM 存储器	1100 0000	000x xxxx	xx**bb bbbb**	**iiii iiii**	向 EEPROM 地址 **b** 处中写入数据 **i**
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 xxxx	xx**bb bb**00	xxxx xxxx	对地址为 **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 P97Table 42
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 P97Table 42
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**
写熔丝位	1010 1100	1010 0000	xxxx xxxx	**iiii iiii**	“0” 表示已编程，“1” 表示未编程
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 P80Table 36
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
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 P80Table 36
读校准字节	0011 1000	000x xxxx	0000 0000	**oooo oooo**	读校准字节
轮询 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，**i** = 数据输入，x = 任意值

串行编程特性参数

Figure 55. 串行编程时序

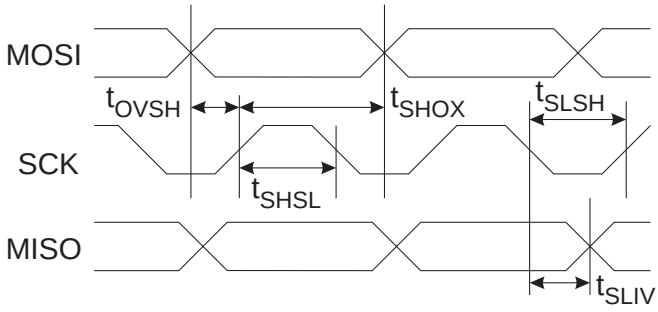


Table 51. 串行编程参数，T_A = -40°C-85°C, V_{CC} = 1.8 - 5.5V（如无特殊说明）

符号	参数	最小值	典型值	最大值	单位
$1/t_{CLCL}$	振荡器频率 (ATtiny13V)	0		1	MHz
t_{CLCL}	振荡器周期 (ATtiny13V)	1,000			ns
$1/t_{CLCL}$	振荡器频率 (ATtiny13L, VCC = 2.7 - 5.5V)	0		9.6	MHz
t_{CLCL}	振荡器周期 (ATtiny13L, VCC = 2.7 - 5.5V)	104			ns
$1/t_{CLCL}$	振荡器频率 (ATtiny13, V_{CC} = 4.5V - 5.5V)	0		16	MHz
t_{CLCL}	振荡器周期 (ATtiny13, V_{CC} = 4.5V - 5.5V)	67			ns
t_{SHSL}	SCK 脉宽高	2 t_{CLCL}*			ns
t_{SLSH}	SCK 脉宽低	2 t_{CLCL}*			ns
t_{OVSH}	当 SCK 为高 MOSI 建立	t_{CLCL}			ns
t_{SHOX}	SCK 为高后 MOSI 保持	2 t_{CLCL}			ns
t_{SLIV}	MISO 有效 SCK 为低	TBD	TBD	TBD	ns

Note: 1. f_{ck} < 12 MHz 为 2 t_{CLCL}，f_{ck} >= 12 MHz 为 3 t_{CLCL}

高电压串行编程

本节说明 ATtiny13 中对 Flash 程序存储器、 EEPROM 数据存储器锁定位及熔丝位的编程与检验。

Figure 56. 高电压串行编程

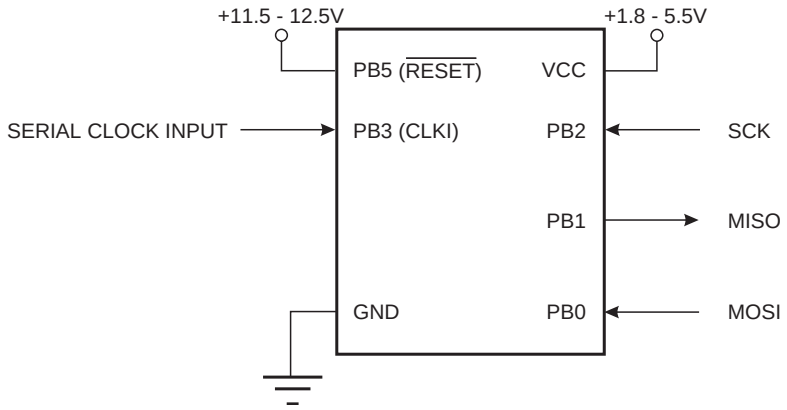


Table 52. 对应引脚名称

在高电压串行编程模式下的信号名称	引脚名称	I/O	功能
SDI	PB0	I	串行数据输入
SII	PB1	I	串行指令输入
SDO	PB2	O	串行数据输出
SCI	PB3	I	串行时钟输入 (最小周期 220ns)

在高电压串行编程时的串行时钟输入 (SCI) 最小周期为 220 ns。

Table 53. 进入编程模式时所用引脚值

引脚	符号	值
SDI	Prog_enable[0]	0
SII	Prog_enable[1]	0
SDO	Prog_enable[2]	0

高电压串行编程算法

在高电压串行编程模式下 ATtiny13 的编程与检验，推荐次序如下 (指令格式见 Table 55):

进入高电压串行指令模式

按照如下算法，芯片进入高电压串行编程模式：

1. V_{CC} 与 GND 间电压为 4.5 - 5.5V
2. 将 RESET 引脚置 “0”， SCI 至少取反 6 次
3. 将 Table 53 中列出的 Prog_enable 引脚设为 “000”，再等待至少 100 ns
4. RESET 电压为 V_{HVRST} - 5.5V ，在提供高电压后，Prog_enable 引脚电压至少保持 t_{HVRST} ，以保证 Prog_enable 信号锁存。
5. 锁存Prog_enable信号后，芯片将从Prog_enable[2]/SDO引脚输出数据，而因此导致的驱动争用将会增加功耗。为最小化驱动争用，在 t_{HVRST} 后释放 Prog_enable[2] 。
6. 至少在 50 µs 后，再在 SDI/SII 上使用串行指令。

Table 54. 高电压复位特性

提供电压	RESET 引脚 高电压阈值	锁存 Prog_enable 的最小高电压周期
V_{CC}	V_{HVRST}	t_{HVRST}
4.5V	11.5V	100 ns
5.5V	11.5V	100 ns

有效编程的几点考虑

在编程过程中，载入的命令与地址还保存在芯片中，为使编程有效，应只有以下几点：

- 当对复用存储器地址写入或读取时，命令只需载入一次。
- 当数据值为0xFF时可不必写入，因为整个EEPROM (除非EESAVE熔丝位已编程) 与 Flash 在芯片擦除后的值为 0xFF。
- 只有在Flash出现256字窗口或EEPROM有256字节时才需载入地址高字节。对信号字节的读取也一样。

芯片擦除

芯片擦除将擦去 Flash 与 EEPROM(1) 内容及锁定位。直到程序存储器完全擦除后锁定位才会复位。熔丝位不会改变。芯片擦除必须在 Flash 与 / 或 EEPROM 重新编程前执行。

Note: 1. 若 EESAVE 熔丝位已编程， EEPROM 存储器在芯片擦除期间保存。

1. 载入命令 “ 芯片擦除 ” (见 Table 55)。
2. Instr. 3 后等待，直到 SDO 变高。
3. 载入命令 “ 无操作 ”。

Flash 编程

Flash 按页组织 ,见 P103Table 50。当对 Flash 编程 ,编程数据存入页缓冲器。这样可实现一页数据同时编程。下面给出对整个 Flash 编程的步骤：

1. 载入命令 “ 写 Flash” (见 Table 55)。
2. 载入 Flash 页缓冲器。
3. 载入 Flash 高地址与编程页。Instr. 3 后等待，直到 SDO 变高。
4. 重复步骤 2 到 3，直到对整个 Flash 编程或所有数据已编程。
5. 通过载入命令 “ 无操作 ” 结束页编程。

ATtiny13 中串行数据的写入或读出，是在串行时钟的上升沿进行，见 Figure 58、Figure 59 及 Table 56。

Figure 57. 对按页组织的 Flash 寻址

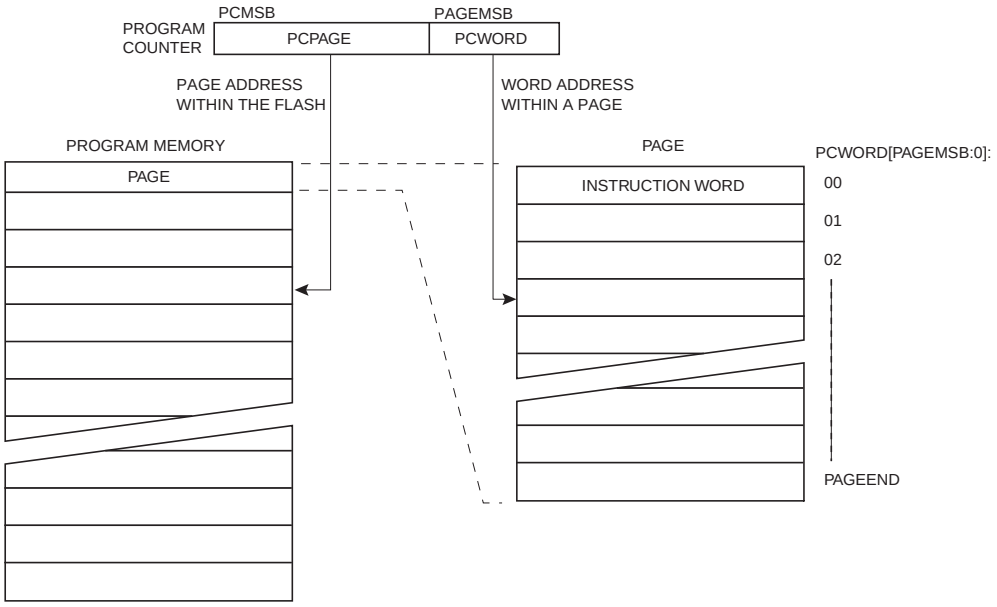
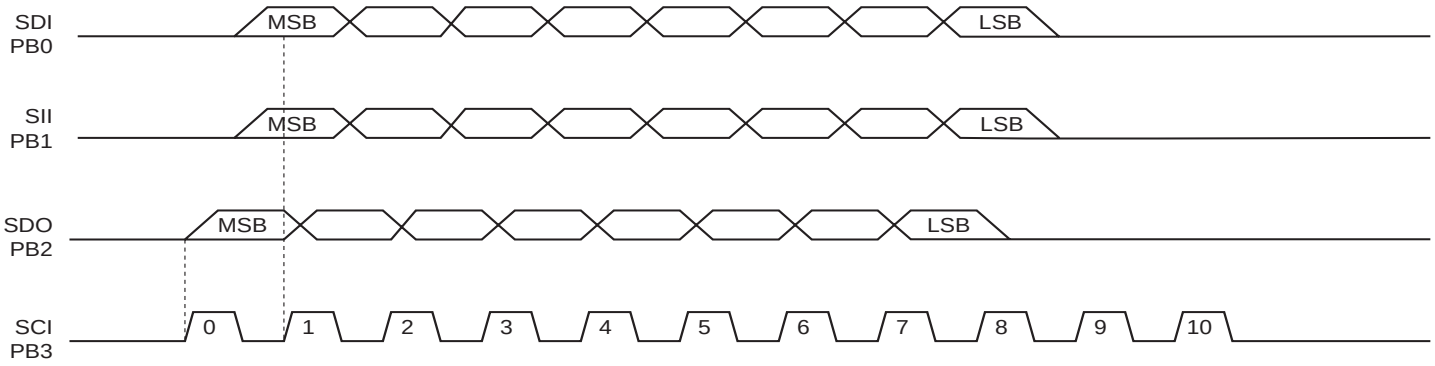


Figure 58. 高电压串行编程波形



EEPROM 编程

EEPROM 按页组织 ,见 P104Table 51。当对 EEPROM 编程 ,编程数据存入页缓冲器。这样可实现一页数据同时编程。下面给出对整个 EEPROM 编程的步骤 (参见 Table 55) :

1. 载入命令 “ 写 EEPROM”
2. 载入 EEPROM 页缓冲器
3. 载入 EEPROM 高地址与编程页。Instr. 2 后等待，直到 SDO 变高。
4. 重复步骤 2 到 3，直到对整个 EEPROM 编程或所有数据已编程。
5. 通过载入命令 “ 无操作 ” 结束页编程。

读取 Flash

读取 Flash 存储器步骤如下 (参见 Table 55) :

1. 载入命令 " 读取 Flash"。
2. 读 Flash 低、高字节。所选地址内容在串行输出 SDO 有效。

读取 EEPROM

读取 EEPROM 存储器步骤如下 (参见 Table 55) :

1. 载入命令 " 读取 EEPROM"。
2. 读 EEPROM 字节。所选地址内容在串行输出 SDO 有效。

对熔丝与锁定位编程与读取

对熔丝位低 / 高位及锁定位的编程与读取见 Table 55。

读取标识字节与标定字节

对标识字节与标定字节的读取见 Table 55。

掉电顺序

设置 SCI 为 “0”。设置 RESET 为 “1”。关闭 V_{CC} 。

Table 55. ATtiny13 高电压串行编程指令集

指令		指令格式				操作说明
		Instr.1/5	**Instr.2/6**	**Instr.3**	**Instr.4**	
Chip Erase	SDI SII SDO	0_1000_0000_00 0_0100_1100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_0100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_1100_00 x_xxxx_xxxx_xx		Instr. 3 后等待，直到 SDO 变高
Load “Write Flash” Command	SDI SII SDO	0_0001_0000_00 0_0100_1100_00 x_xxxx_xxxx_xx				进入 Flash 编程码
Load Flash Page Buffer	SDI SII SDO	0_ **bbbb_bbbb** _00 0_0000_1100_00 x_xxxx_xxxx_xx	0_**eeee_eeee**_00 0_0010_1100_00 x_xxxx_xxxx_xx	0_**dddd_dddd**_00 0_0011_1100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0111_1101_00 x_xxxx_xxxx_xx	重复 Instr. 1 - 5 直到填满整个页缓冲器或填完所有数据，见 Note 1。
	SDI SII SDO	0_0000_0000_00 0_0111_1100_00 x_xxxx_xxxx_xx				Instr 5.
Load Flash High Address and Program Page	SDI SII SDO	0_0000_000**a**_00 0_0001_1100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_0100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_1100_00 x_xxxx_xxxx_xx		Instr. 3 后等待，直到 SDO 变高。在载入 Flash 页时重复 Instr. 2 - 3，直到整个 Flash 或所有数据已编程。对新的 256 字节，重复 Instr. 1，见 Note 1。
Load “Read Flash” Command	SDI SII SDO	0_0000_0010_00 0_0100_1100_00 x_xxxx_xxxx_xx				进入 Flash 读取模式
Read Flash Low and High Bytes	SDI SII SDO	0_**bbbb_bbbb**_00 0_0000_1100_00 x_xxxx_xxxx_xx	0_0000_000**a**_00 0_0001_1100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_1000_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_1100_00 **q_qqqq_qqq**x_xx	每个新地址重复 Instr. 1, 3 - 6，对新的 256 字节，重复 Instr. 2 。
	SDI SII SDO	0_0000_0000_00 0_0111_1000_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0111_1100_00 **p_pppp_ppp**x_xx			Instr 5 - 6.
Load “Write EEPROM” Command	SDI SII SDO	0_0001_0001_00 0_0100_1100_00 x_xxxx_xxxx_xx				进入 EEPROM 编程模式
Load EEPROM Page Buffer	SDI SII SDO	0_00**bb_bbbb**_00 0_0000_1100_00 x_xxxx_xxxx_xx	0_**eeee_eeee**_00 0_0010_1100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_1101_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_1100_00 x_xxxx_xxxx_xx	重复 Instr. 1 - 4 直到填满整个页缓冲器或填完所有数据，见 Note 2。
Program EEPROM Page	SDI SII SDO	0_0000_0000_00 0_0110_0100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_1100_00 x_xxxx_xxxx_xx			Instr. 2 后等待，直到 SDO 变高。每次载入 EEPROM 页重复 Instr. 1 - 2 直到整个 EEPROM 或所有数据编程。
Write EEPROM Byte	SDI SII SDO	0_00**bb_bbbb**_00 0_0000_1100_00 x_xxxx_xxxx_xx	0_**eeee_eeee**_00 0_0010_1100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_1101_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_0100_00 x_xxxx_xxxx_xx	对每个新地址重复 Instr. 1 - 5。Instr. 5 后等待，直到 SDO 变高，见 Note 3。
	SDI SII SDO	0_0000_0000_00 0_0110_1100_00 x_xxxx_xxxx_xx				Instr. 5
Load “Read EEPROM” Command	SDI SII SDO	0_0000_0011_00 0_0100_1100_00 x_xxxx_xxxx_xx				进入 EEPROM 读取模式

Table 55. ATtiny13 高电压串行编程指令集 (Continued)

指令		指令格式 Instr.1/5	Instr.2/6	Instr.3	Instr.4	操作说明
Read EEPROM Byte	SDI SII SDO	0_**bbbb_bbbb**_00 0_0000_1100_00 x_xxxx_xxxx_xx	0_**aaaa_aaaa**_00 0_0001_1100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_1000_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_1100_00 **q_qqqq_qqq**0_00	对每个新地址重复 Instr. 1, 3 - 4 。对新的 256 字节页重复 Instr. 2。
Write Fuse Low Bits	SDI SII SDO	0_0100_0100_00 0_0100_1100_00 x_xxxx_xxxx_xx	0_**A987_6543**_00 0_0010_1100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_0100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_1100_00 x_xxxx_xxxx_xx	Instr. 4 后等待 ,直到 SDO 变高。写 **A - 3** = “0” 对熔丝位编程。
Write Fuse High Bits	SDI SII SDO	0_0100_0000_00 0_0100_1100_00 x_xxxx_xxxx_xx	0_000**F_EDCB**_00 0_0010_1100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0111_0100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0111_1100_00 x_xxxx_xxxx_xx	Instr. 4 后等待 ,直到 SDO 变高。写 **F - B** = “0” 对熔丝位编程。
Write Lock Bits	SDI SII SDO	0_0010_0000_00 0_0100_1100_00 x_xxxx_xxxx_xx	0_0000_00**21**_00 0_0010_1100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_0100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_1100_00 x_xxxx_xxxx_xx	Instr. 4 后等待 ,直到 SDO 变高。写 **2 - 1** = “0” 对锁定位编程
Read Fuse Low Bits	SDI SII SDO	0_0000_0100_00 0_0100_1100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_1000_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_1100_00 **A_9876_543**x_xx		读取 **A - 3** = “0” 表示熔丝位已编程
Read Fuse High Bits	SDI SII SDO	0_0000_0100_00 0_0100_1100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0111_1010_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0111_1110_00 x_xx**FE_DCB**x_xx		读取 **F - B** = “0” 表示熔丝位已编程
Read Lock Bits	SDI SII SDO	0_0000_0100_00 0_0100_1100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0111_1000_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0111_1100_00 x_xxxx_x**21**x_xx		读取 **2, 1** = “0” 表示锁定位已编程
Read Signature Bytes	SDI SII SDO	0_0000_1000_00 0_0100_1100_00 x_xxxx_xxxx_xx	0_0000_00**bb**_00 0_0000_1100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_1000_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0110_1100_00 **q_qqqq_qqq**x_xx	每个表示字节地址，重复 Instr 2 4
Read Calibration Byte	SDI SII SDO	0_0000_1000_00 0_0100_1100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0000_1100_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0111_1000_00 x_xxxx_xxxx_xx	0_0000_0000_00 0_0111_1100_00 **p_pppp_ppp**x_xx	
Load “No Operation” Command	SDI SII SDO	0_0000_0000_00 0_0100_1100_00 x_xxxx_xxxx_xx				

Note: **a** = 地址高字节， **b** = 地址低字节， **d** = 数据高位， **e** = 数据低位，**p** = 数据输出高位， **q** = 数据输出低位
x = 任意值，***1*** = 锁定位 1，***2*** = 锁定位 2，**3** = CKSEL0 熔丝位，**4** = CKSEL1 熔丝位，**5** = SUT0 熔丝位，**6** = SUT1 熔丝位，**7** = CKDIV8 熔丝位，**8** = WDTON 熔丝位，**9** = EESAVE 熔丝位，**A** = SPIEN 熔丝位，**B** = RSTDISBL 熔丝位，**C** = BODLEVEL0 熔丝位， **D**= BODLEVEL1 熔丝位， **E** = MONEN 熔丝位， **F** = SELFPRGEN 熔丝位，

Notes:
1. 页尺寸低于 256 字，部分地址 (bbbb_bbbb) 将作为页地址。
2. 页尺寸低于 256 字节，部分地址 (bbbb_bbbb) 将作为页地址。
3. EEPROM 以页方式写入。但只有将字节载入页，才能真正写入 EEPROM。当多字节写入同一页时，页方式 EEPROM 访问效率更高 。注意 EEPROM 的自动擦除在高电压串行编程中无效。

高电压串行编程特性

Figure 59. 高电压串行编程时序

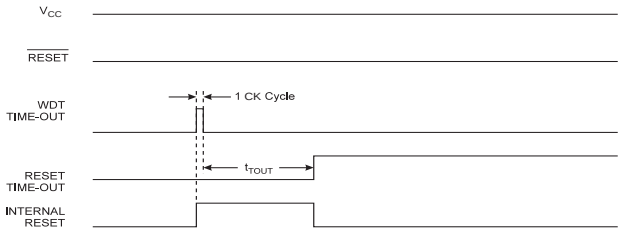


Table 56. 高电压串行编程特性 V_{CC} = 5.0V ± 10% (无特殊说明)

符号	参数 r	最小值	典型值	最大值	单位
t_{SHSL}	SCI (PB3) 脉宽高	110			ns
t_{SLSH}	SCI (PB3) 脉宽低	110			ns
t_{IVSH}	SDI (PB0)、 SII (PB1) 有效到 SCI (PB3) 高	50			ns
t_{SHIX}	SCI (PB3) 高后 SDI (PB0)、 SII (PB1) 保持	50			ns
t_{SHOV}	SCI (PB3) 高到 SDO (PB2) 有效		16		ns
t_{WLWH_PFB}	Instr. 3 后等待写熔丝位		2.5		ms

电气特性

绝对极限值 *

工作温度	-55°C 到 +125°C
存储温度	-65°C 到 +150°C
各个引脚对地的电压，除了 $\overline{\text{RESET}}$	-0.5V 到 V_{CC}+0.5V
$\overline{\text{RESET}}$ 引脚对地的电压	-0.5V 到 +13.0V
最大工作电压	6.0V
每个 I/O 引脚的 DC 电流	40.0 mA
V_{CC} 与 GND 引脚的 DC 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

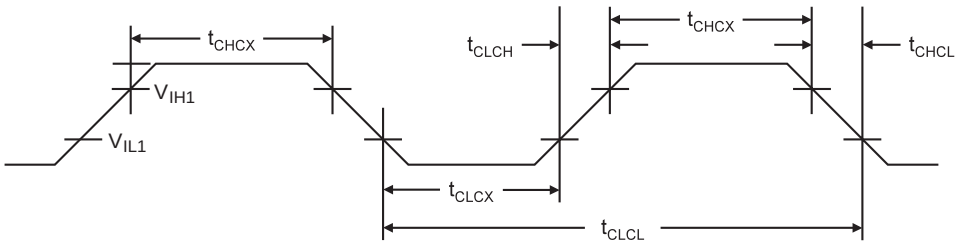
T_A = -40°C -85°C, V_{CC} = 1.8V -5.5V (如无特别说明)(1)

符号	参数	条件	最小值	典型值	最大值	单位
V_{IL}	输入低电压		-0.5		$0.2V_{CC}$	V
V_{IH}	输入高电压	除 $\overline{\text{RESET}}$ 引脚	$0.6V_{CC}$(3)		V_{CC} +0.5	V
V_{IH2}	输入高电压	$\overline{\text{RESET}}$ 引脚	$0.9V_{CC}$(3)		V_{CC} +0.5	V
V_{OL}	输出低电压 (4) (PB5、 PB1 与 PB0)	I_{OL} = 20 mA, V_{CC} = 5V I_{OL} = 10 mA, V_{CC} = 3V			0.7 0.5	V V
V_{OL1}	输出低电压 (4) (PB4、 PB3 与 PB2)	I_{OL} = 10 mA, V_{CC} = 5V I_{OL} = 5 mA, V_{CC} = 3V			0.7 0.5	V V
V_{OH}	输出高电压 (5) (PB5、 PB1 与 PB0)	I_{OH} = -20 mA, V_{CC} = 5V I_{OH} = -10 mA, V_{CC} = 3V	4.2 2.5			V V
V_{OH1}	输出高电压 (5) (PB4、 PB3 与 PB2)	I_{OH} = -10 mA, V_{CC} = 5V I_{OH} = -5 mA, V_{CC} = 3V	4.2 2.5			V V
I_{IL}	输入漏电流， I/O 引脚	Vcc = 5.5V, 引脚低 (绝对值)			1	µA
I_{IH}	输入漏电流， I/O 引脚	Vcc = 5.5V, 引脚高 (绝对值)			1	µA
R_{RST}	Reset 引脚上拉电阻		30		80	kΩ
R_{pu}	I/O 引脚上拉电阻		20		50	kΩ
I_{CC}	电源电流	正常 1MHz, V_{CC} = 2V			0.55	mA
		正常 4MHz, V_{CC} = 3V			3.5	mA
		正常 8MHz, V_{CC} = 5V			12	mA
		空闲 1MHz, V_{CC} = 2V		0.08	0.25	mA
		空闲 4MHz, V_{CC} = 3V		0.41	1.5	mA
		空闲 8MHz, V_{CC} = 5V		1.6	5.5	mA
	掉电模式	WDT 使能 , V_{CC} = 3V		< 5	16	µA
		WDT 使能 , V_{CC} = 3V		< 0.5	1	µA

Notes: 1. 本手册中所有的直流特性数据与其它 AVR 微控制器手册一样，均是基于仿真的，这些值仅是设计目标，实际值在对实际芯片的测试后会更新。
2. “ 最大值” 表示保证引脚读取数值为低时的最高值
3. “ 最小值” 表示保证引脚读取数值为高时的最低值
4. 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 (对 PB5、PB1:0：20 mA , V_{CC} = 5V ； 10 mA，V_{CC} = 3V。对 PB4:2：10 mA ， V_{CC} = 5V ； 5 mA，V_{CC} = 3V) ，但是仍需要遵循以下要求：
1] 所有端口的 IOL 总和不能超过 60 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。
5. 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 (对 PB5、PB1:0：20 mA , V_{CC} = 5V ； 10 mA，V_{CC} = 3V。对 PB4:2：10 mA ， V_{CC} = 5V ； 5 mA，V_{CC} = 3V) ，但是仍需要遵循以下要求：：
1] 所有端口的 IOL 总和不能超过 60 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。

外部时钟驱动波形

Figure 60. 外部时钟驱动波形



外部时钟驱动

Table 57. 外部时钟驱动

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	
$1/t_{CLCL}$	时钟频率	0	1	0	9.6	0	16	MHz
t_{CLCL}	时钟周期	1000		104		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	%

最大速度与 V_{CC} 的关系

最高频率取决于 $V_{CC.}$，如图 Figure 61 与 Figure 62 所示， 1.8V < V_{CC} < 2.7V 及 2.7V < V_{CC} < 4.5V 时，最高频率与 V_{CC} 为线性关系

Figure 61. ATtiny13V 中最高频率与 V_{CC} 的关系

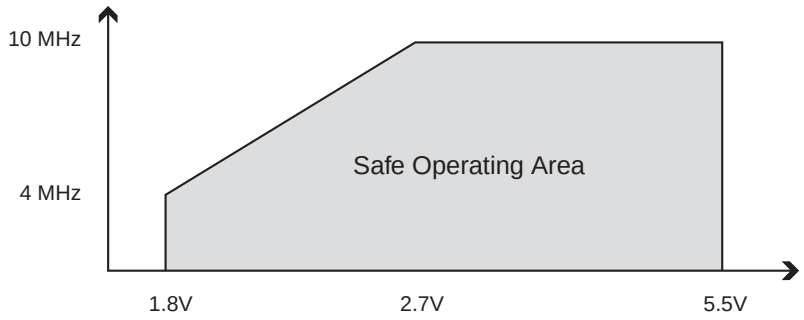
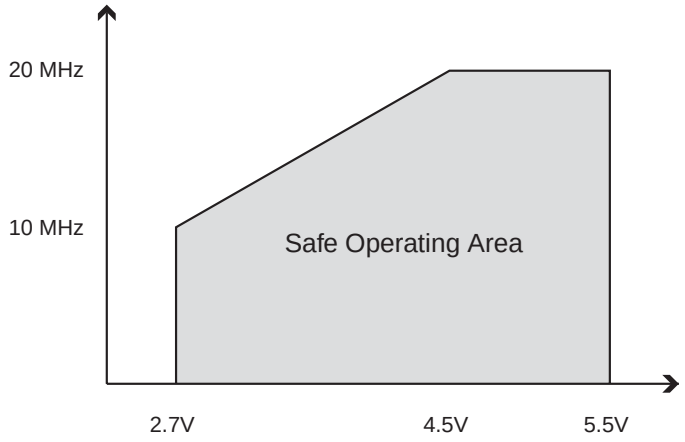


Figure 62. ATtiny13 中最高频率与 V_{CC} 的关系



ADC 特性 - 初始参数

Table 58. ADC 特性参数，单端通道。-40°C - 85°C

符号	参数	条件	最小值 (1)	典型值 (1)	最大值 (1)	单位
	分辨率	单端转换			10	Bits
	绝对精度 (包括 INL、 DNL、量化误差、增益及偏置误差)	单端转换 V_{REF} = 4V，V_{CC} = 4V ADC 时钟 = 200 kHz		2		LSB
		单端转换 V_{REF} = 4V，V_{CC} = 4V ADC 时钟 = 1 MHz		3		LSB
		单端转换 V_{REF} = 4V，V_{CC} = 4V ADC 时钟 = 200 kHz 噪声抑制模式		1.5		LSB
		单端转换 V_{REF} = 4V，V_{CC} = 4V ADC 时钟 = 1 MHz 噪声抑制模式		2.5		LSB
	整体非线性 (INL)	单端转换 V_{REF} = 4V，V_{CC} = 4V ADC 时钟 = 200 kHz		1		LSB
	差分非线性 (DNL)	单端转换 V_{REF} = 4V，V_{CC} = 4V ADC 时钟 = 200 kHz		0.5		LSB
	增益误差	单端转换 V_{REF} = 4V，V_{CC} = 4V ADC 时钟 = 200 kHz		2.5		LSB
	偏移误差	单端转换 V_{REF} = 4V，V_{CC} = 4V ADC 时钟 = 200 kHz		1.5		LSB
	转换时间	连续转换	13		260	µs
	时钟频率		50		1000	kHz
V_{IN}	输入电压		GND		V_{REF}	V
	输入带宽			38.5		kHz
V_{INT}	片内参考电压		1.0	1.1	1.2	V
R_{AIN}	模拟输入阻抗			100		MΩ

Notes: 1. 值为初始值

ATtiny13 典型特征参数

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 63. 工作电流和工作频率 (0.1 - 1.0 MHz) 的关系

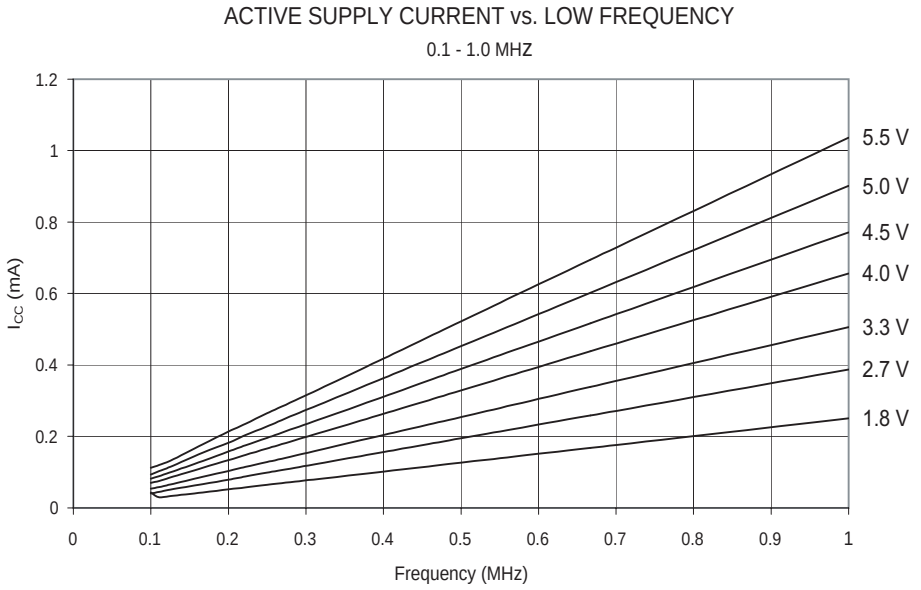


Figure 64. 工作电流和工作频率 (1 - 24 MHz) 的关系

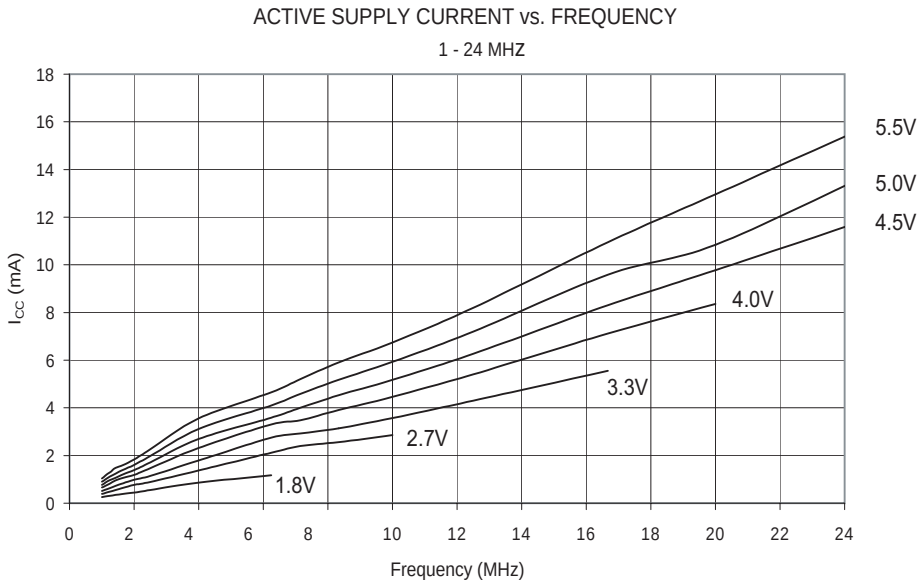


Figure 65. 工作电流和 V_{CC} 的关系 (内部 RC 振荡器，9.6MHz)

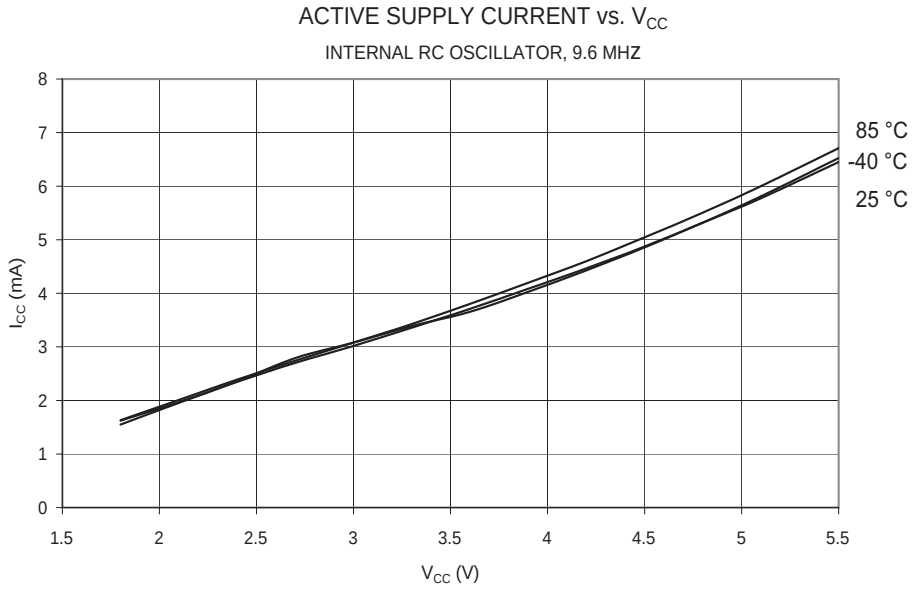


Figure 66. 工作电流和 V_{CC} 的关系 (内部 RC 振荡器， 4.8 MHz)

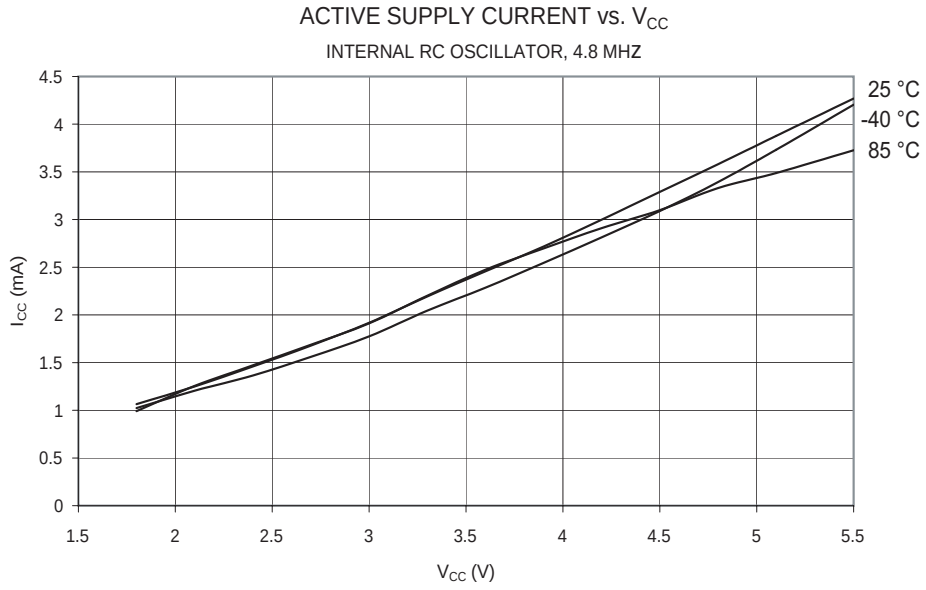


Figure 67. 工作电流和 V_{CC} 的关系 (内部 WDT 振荡器， 128 kHz)

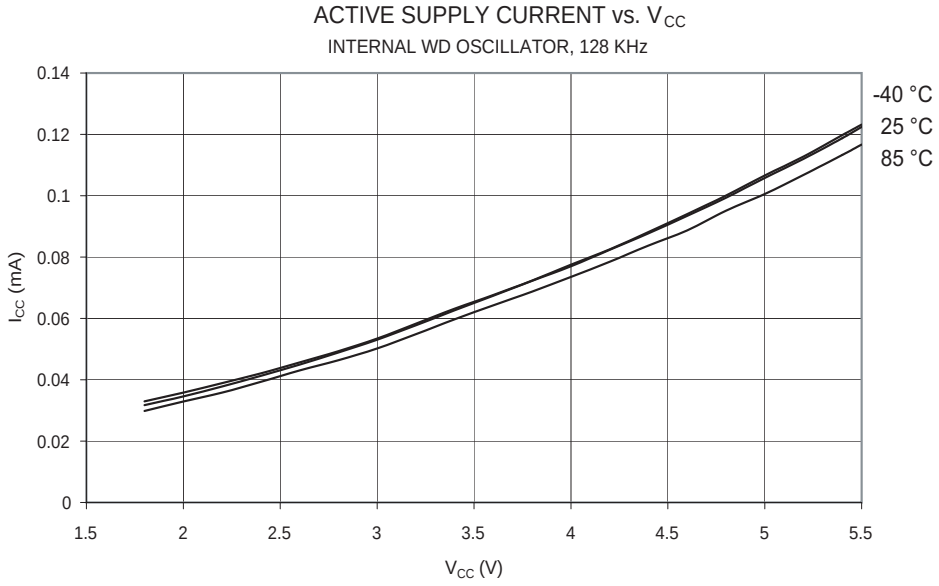
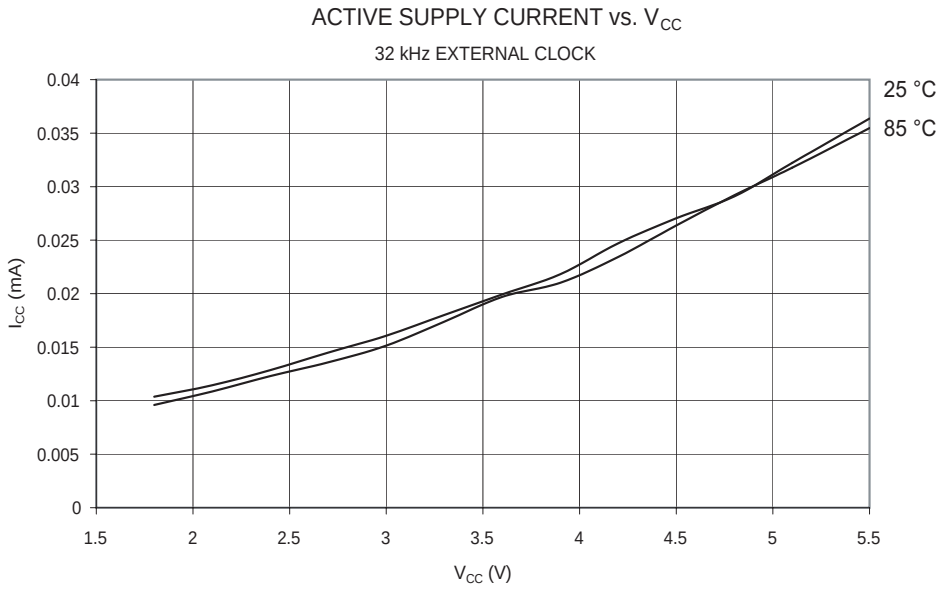


Figure 68. 工作电流和 V_{CC} 的关系 (32 kHz 外部时钟)



空闲模式电流

Figure 69. 空闲模式电流和工作频率 (0.1 - 1.0 MHz) 的关系

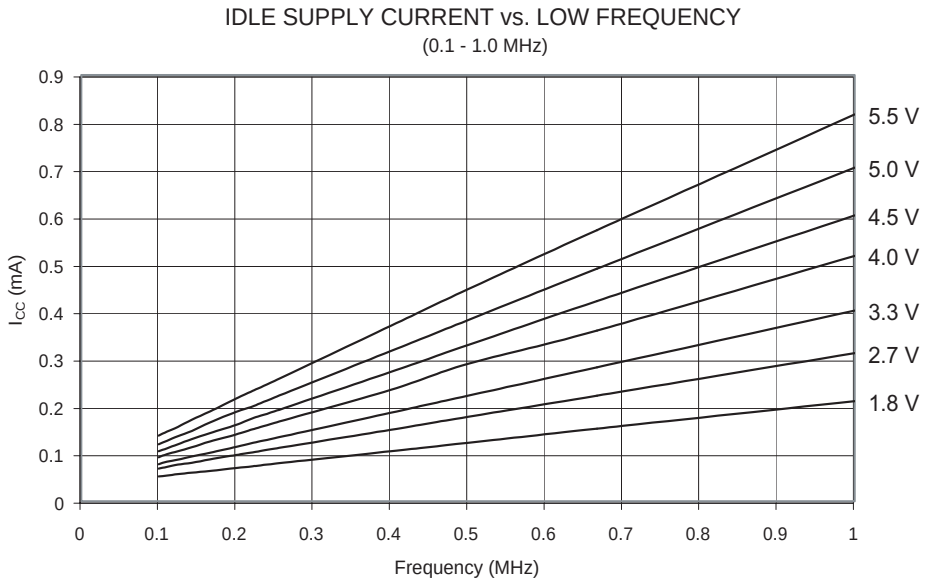


Figure 70. 空闲模式电流和工作频率 (1 - 24 MHz) 的关系

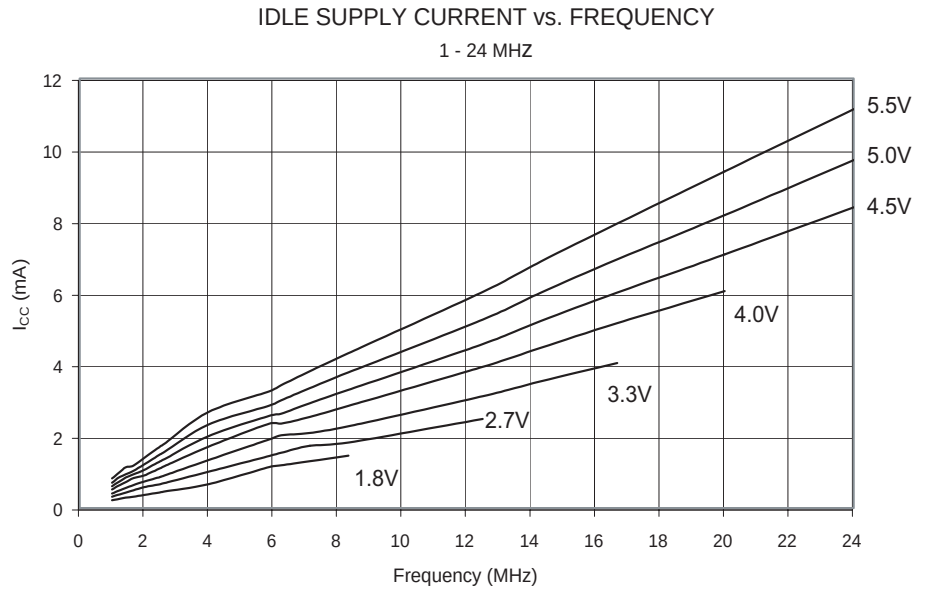


Figure 71. 空闲模式电流和 V_{CC} 的关系 (内部 RC 振荡器 ， 9.6 MHz)

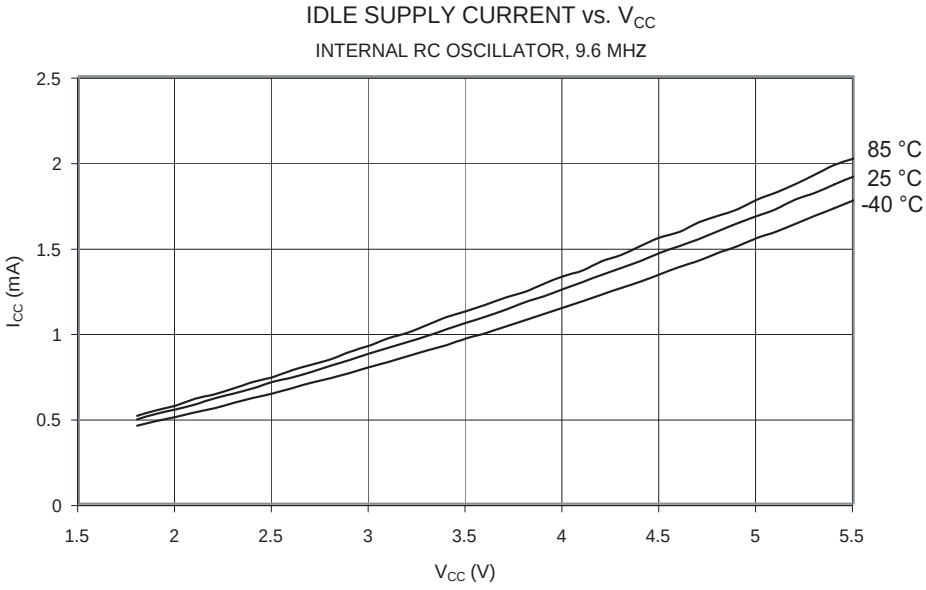


Figure 72. 空闲模式电流和 V_{CC} 的关系 (内部 RC 振荡器 ， 4.8 MHz)

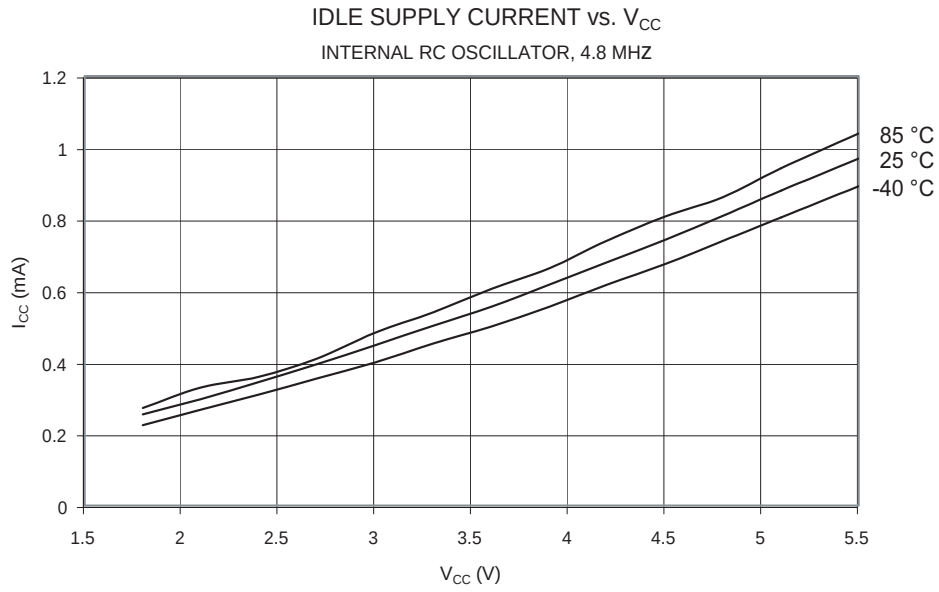


Figure 73. 空闲模式电流和 V_{CC} 的关系 (内部 RC 振荡器 ， 128 kHz)

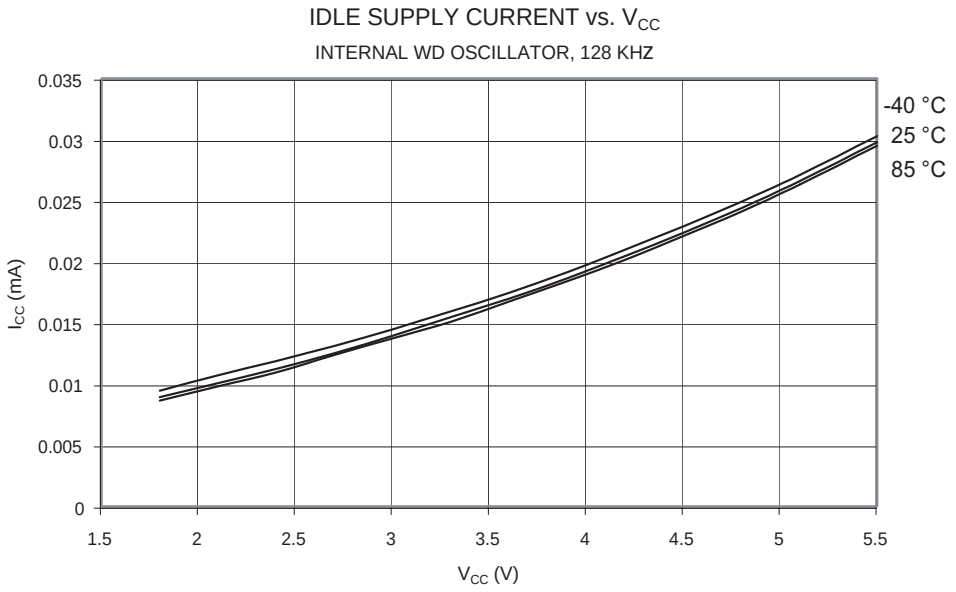
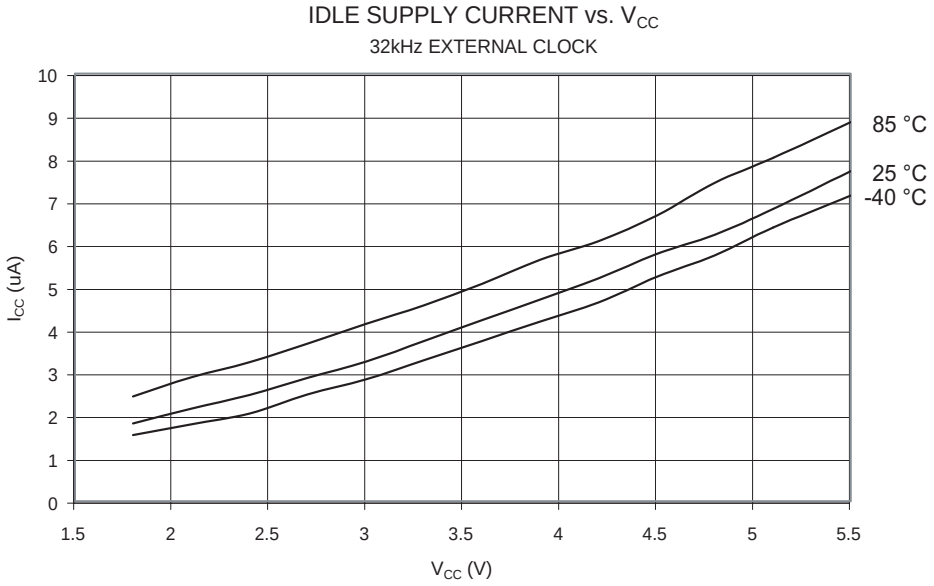


Figure 74. 空闲模式电流和 V_{CC} 的关系 (32 kHz 外部时钟)



掉电模式电流

Figure 75. 掉电模式电流和 V_{CC} 的关系 (看门狗定时器禁用)

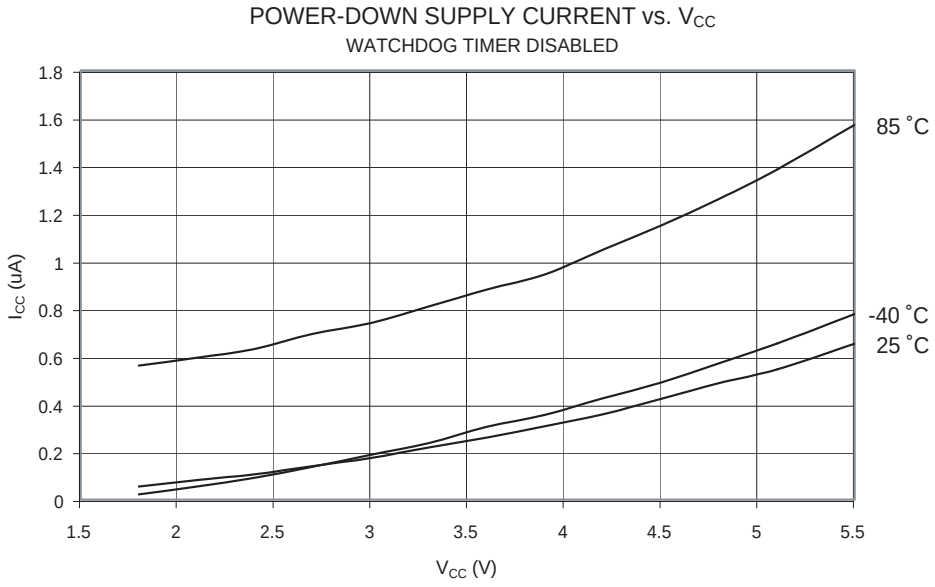
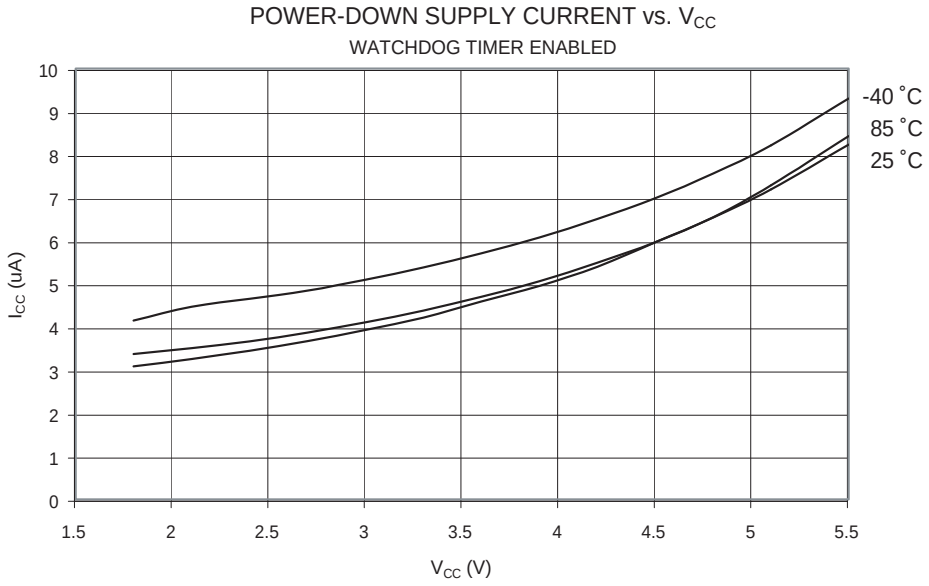


Figure 76. 掉电模式电流和 V_{CC} 的关系 (看门狗定时器使能)



引脚上拉

Figure 77. I/O 引脚上拉电阻电流和输入电压的关系 (V_{CC} = 5V)

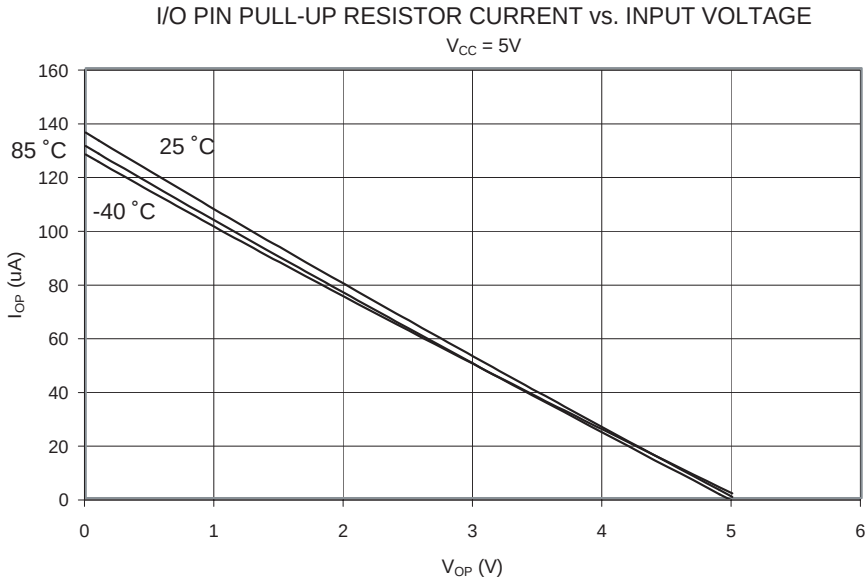


Figure 78. I/O 引脚上拉电阻电流和输入电压的关系 (V_{CC} = 2.7V)

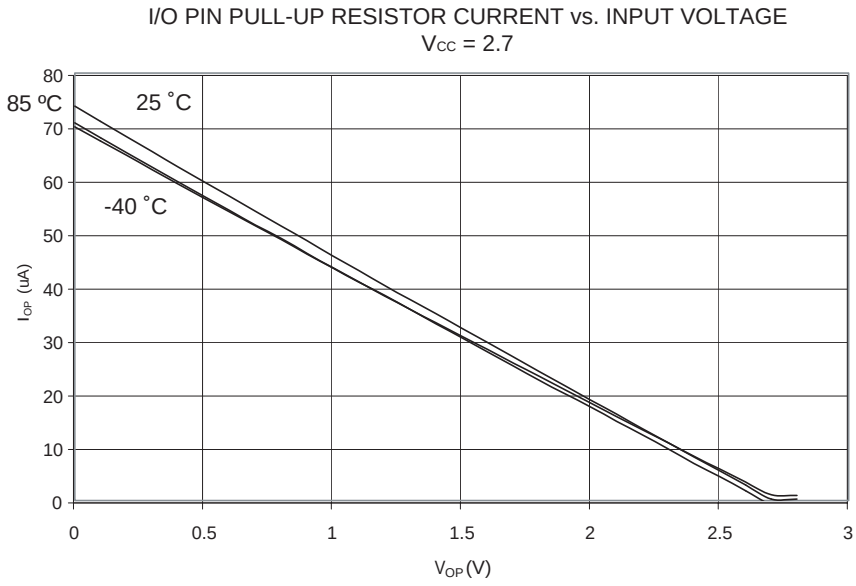


Figure 79. 复位 (Reset) 引脚上拉电阻电流和 Reset 引脚电压的关系 (V_{CC} = 5V)

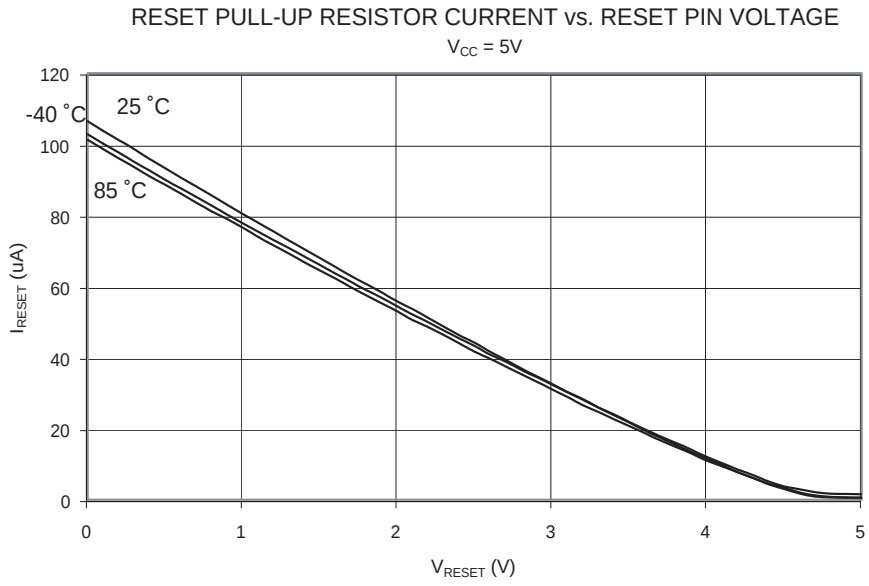
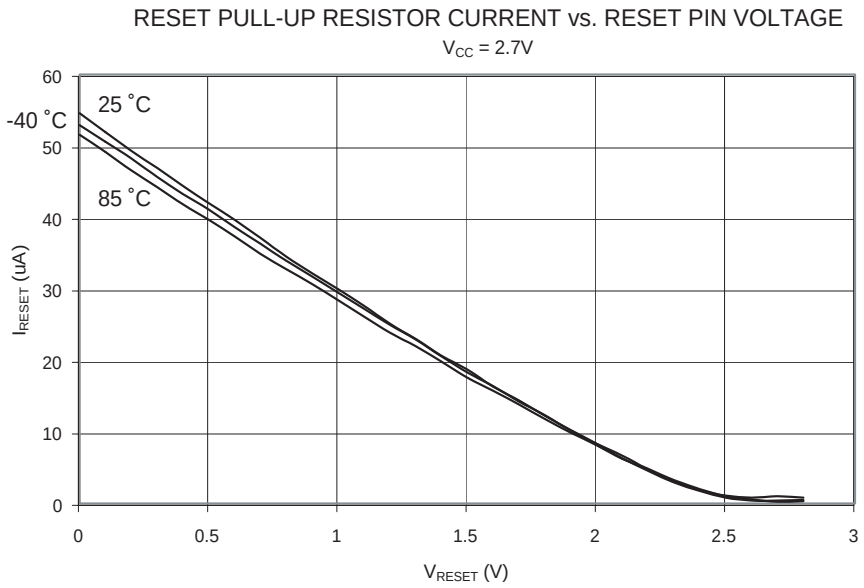


Figure 80. 复位 (Reset) 引脚上拉电阻电流和 Reset 引脚电压的关系 (V_{CC} = 2.7V)



驱动能力

Figure 81. I/O 引脚源电流和输出电压的关系 (低功端口， V_{CC} = 5V)

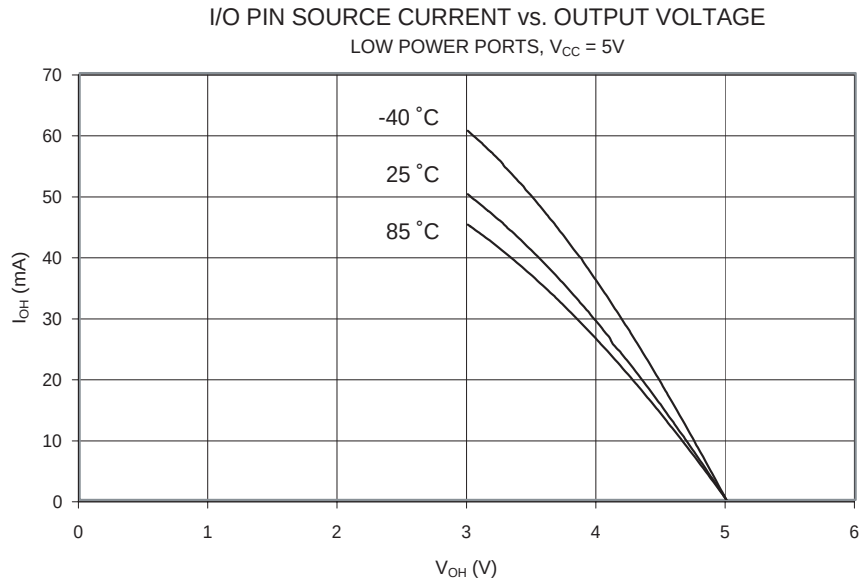


Figure 82. I/O 引脚源电流和输出电压的关系 (低功端口， V_{CC} = 2.7V)

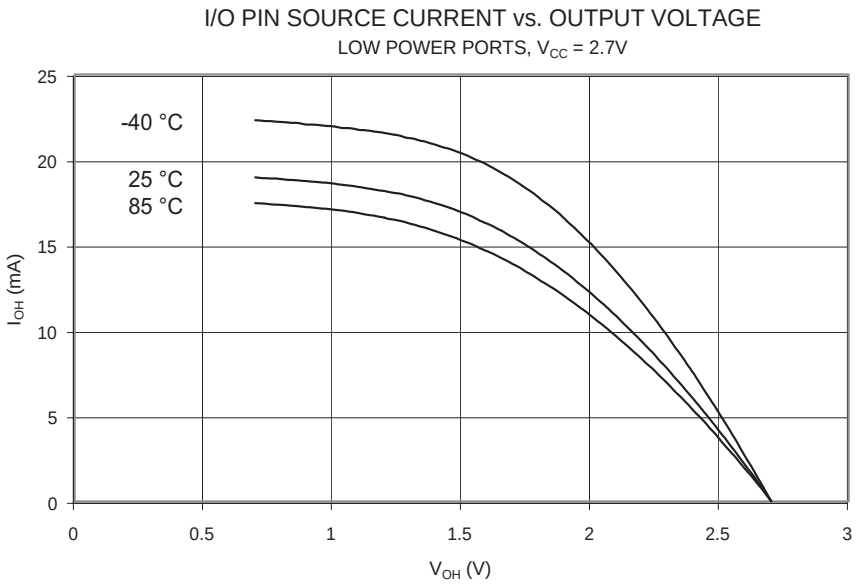


Figure 83. I/O 引脚源电流和输出电压的关系 (低功端口， V_{CC} = 1.8V)

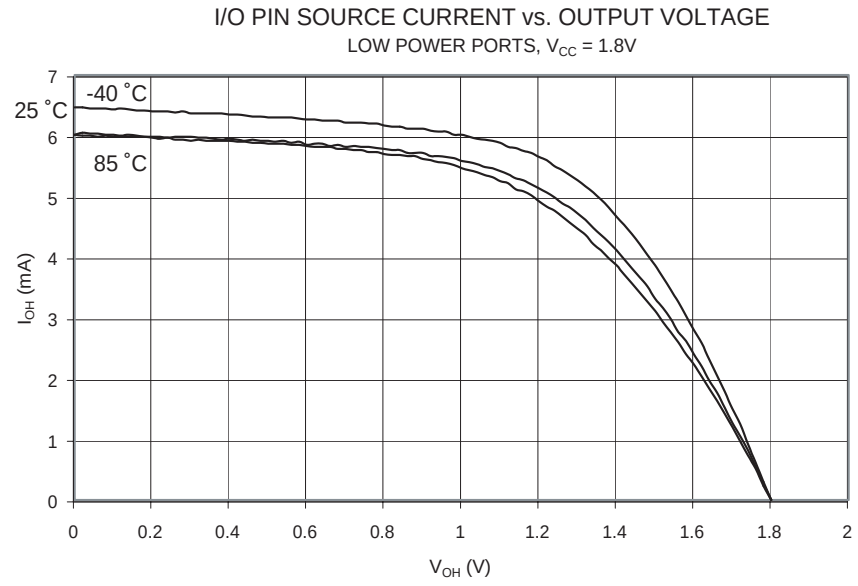


Figure 84. I/O 引脚吸收电流和输出电压的关系 (低功端口， V_{CC} = 5V)

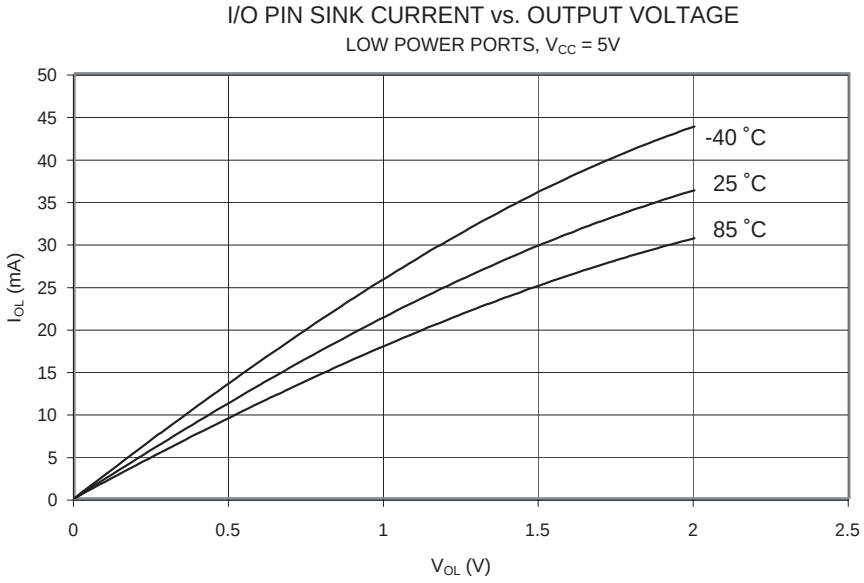


Figure 85. I/O 引脚吸收电流和输出电压的关系 (低功端口， V_{CC} = 2.7V)

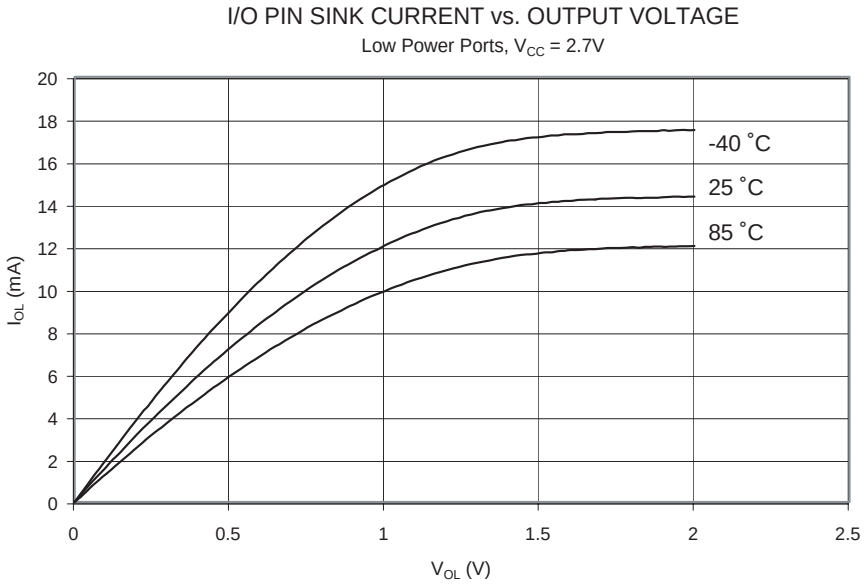


Figure 86. I/O 引脚吸收电流和输出电压的关系 (低功端口，V_{CC} = 1.8V)

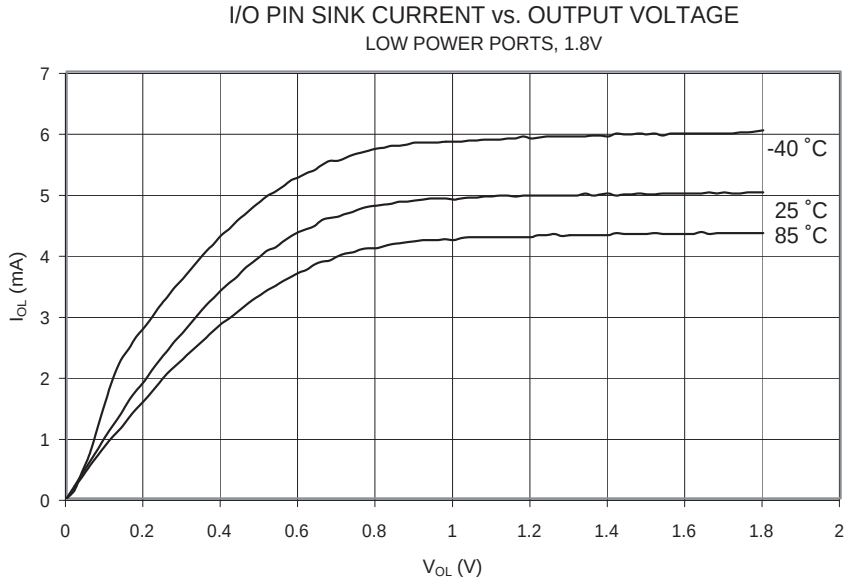


Figure 87. I/O 引脚源电流和输出电压的关系 (V_{CC} = 5V)

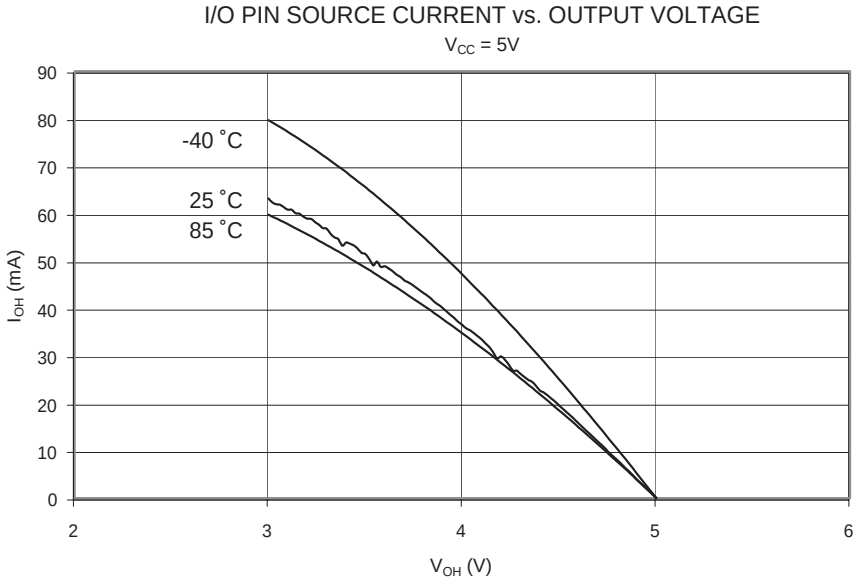


Figure 88. I/O 引脚源电流和输出电压的关系 (V_{CC} = 2.7V)

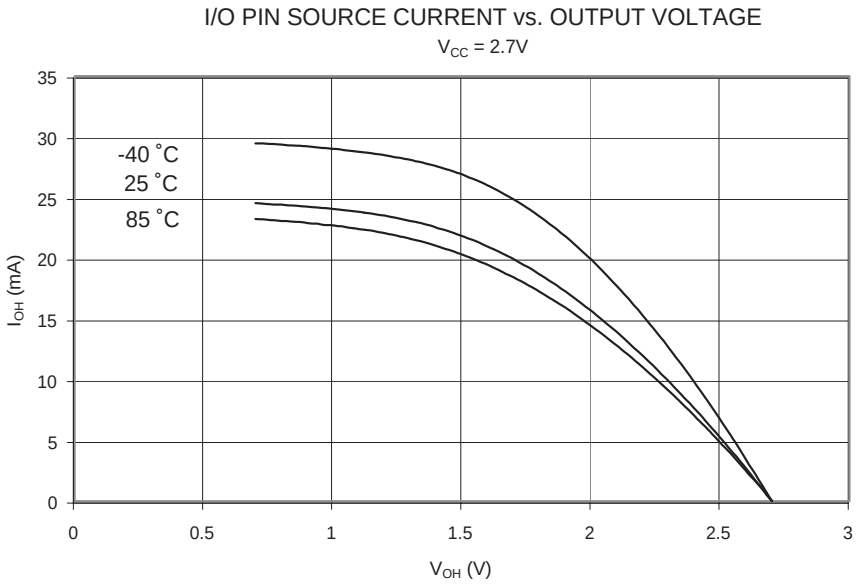


Figure 89. I/O 引脚源电流和输出电压的关系 (V_{CC} = 1.8V)

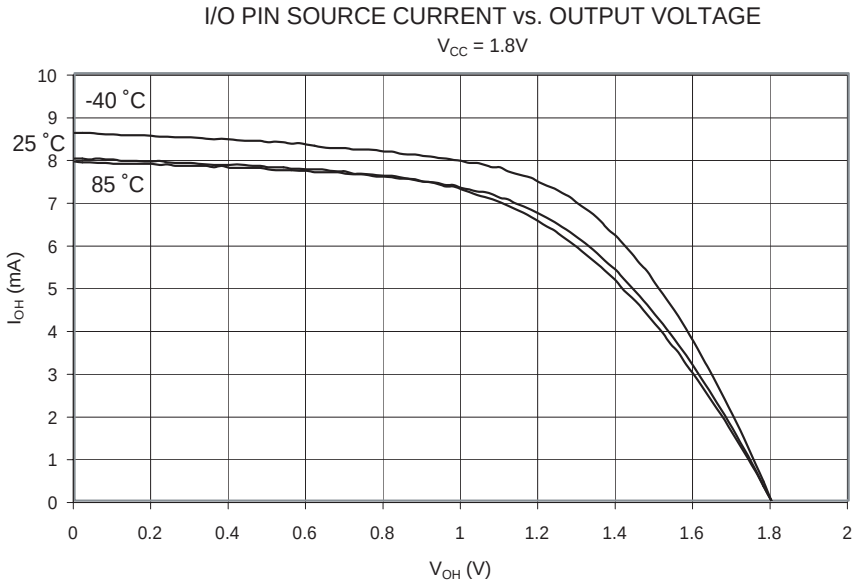


Figure 90. I/O 引脚吸收电流和输出电压的关系 (V_{CC} = 5V)

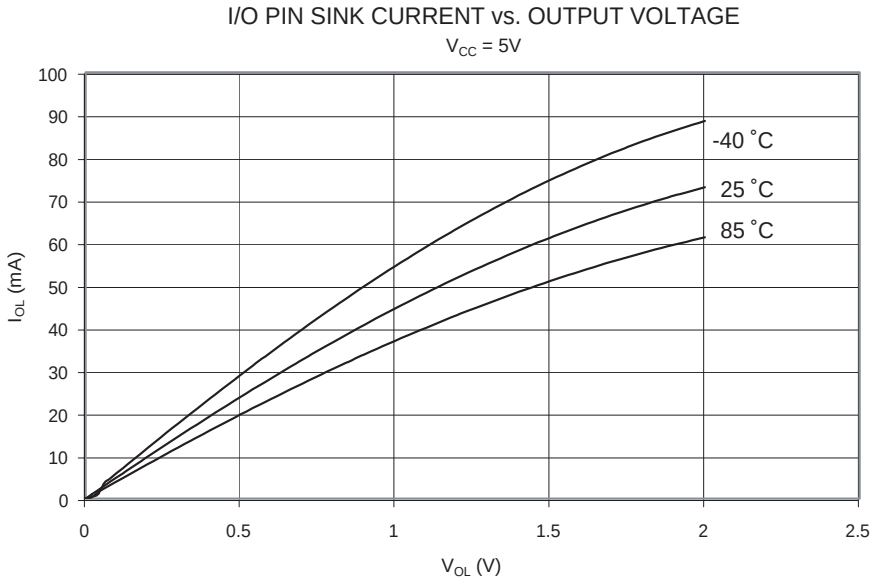


Figure 91. I/O 引脚吸收电流和输出电压的关系 (V_{CC} = 2.7V)

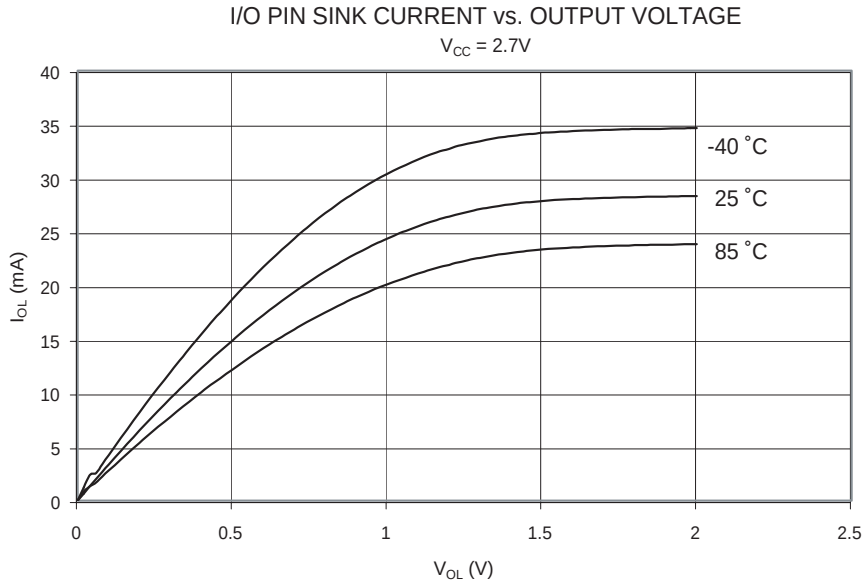


Figure 92. I/O 引脚吸收电流和输出电压的关系 (V_{CC} = 1.8V)

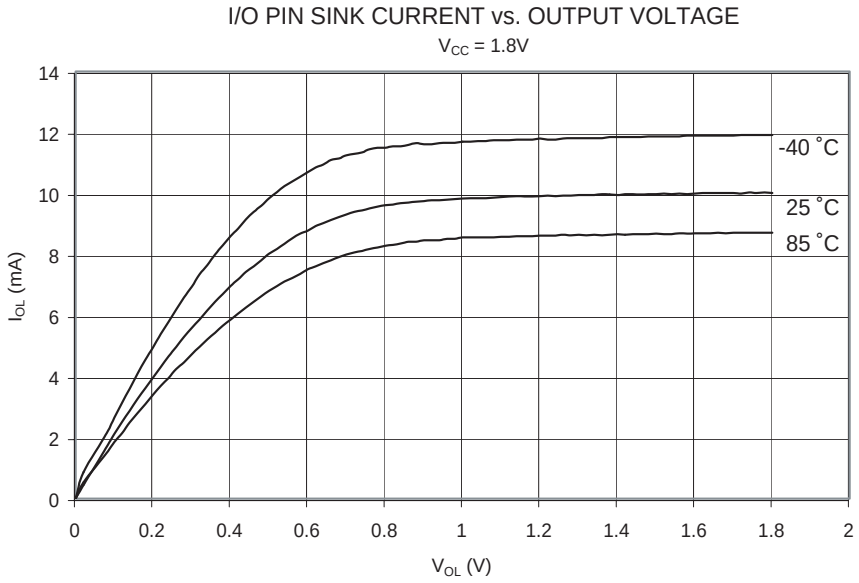


Figure 93. Reset 引脚作为 I/O – 源电流和输出电压的关系 (V_{CC} = 5V)

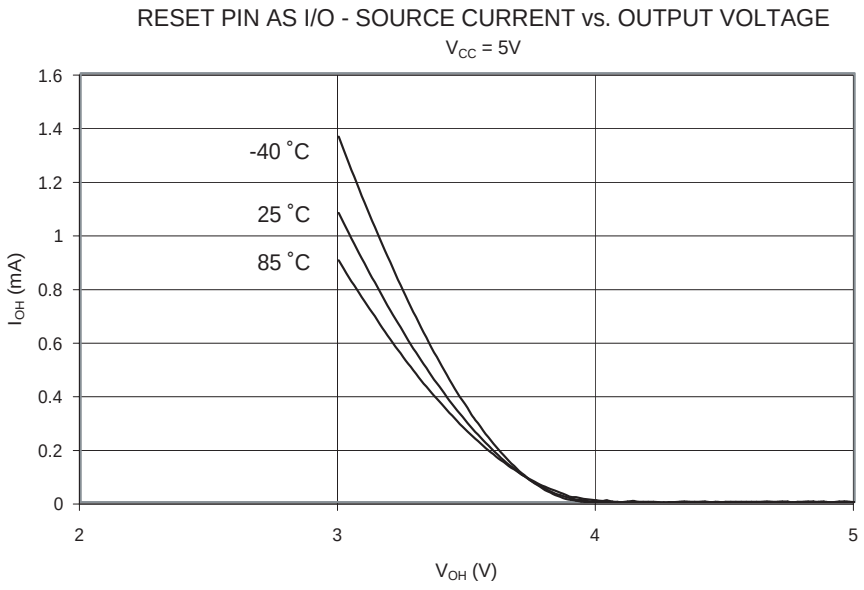


Figure 94. Reset 引脚作为 I/O – 源电流和输出电压的关系 (V_{CC} = 2.7V)

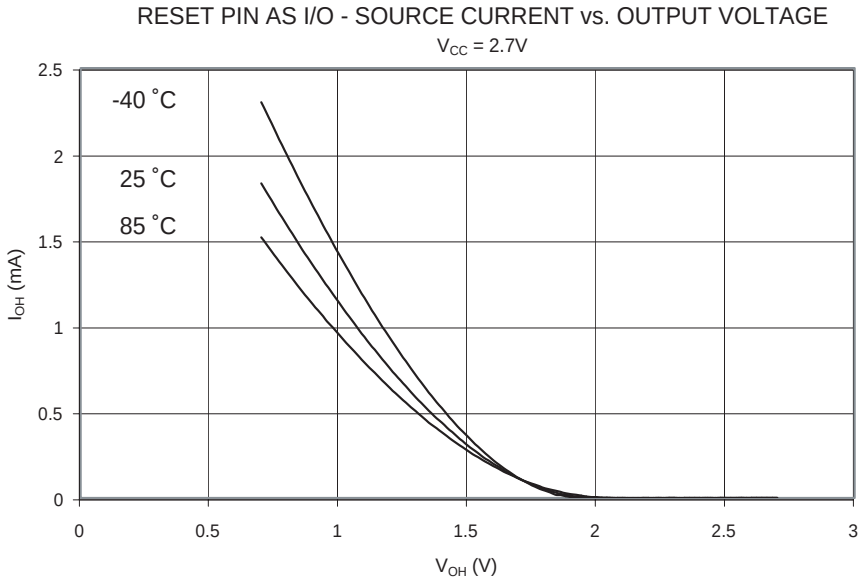


Figure 95. Reset 引脚作为 I/O – 源电流和输出电压的关系 (V_{CC} = 1.8V)

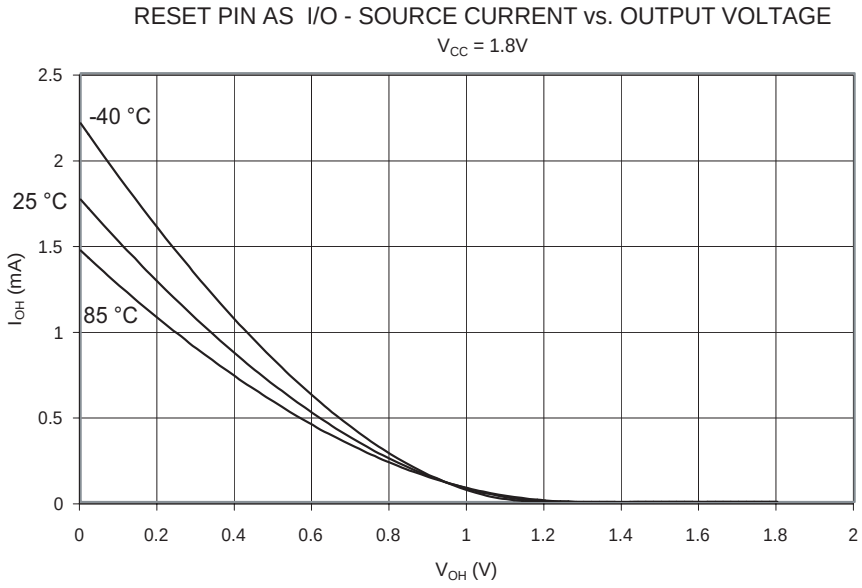


Figure 96. Reset 引脚作为 I/O – 吸收电流和输出电压的关系 (V_{CC} = 5V)

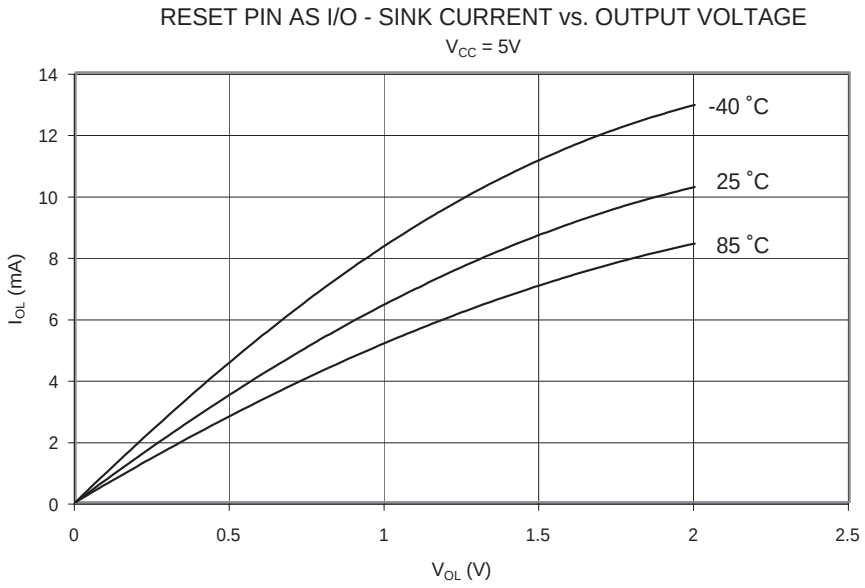


Figure 97. Reset 引脚作为 I/O – 吸收电流和输出电压的关系 (V_{CC} = 2.7V)

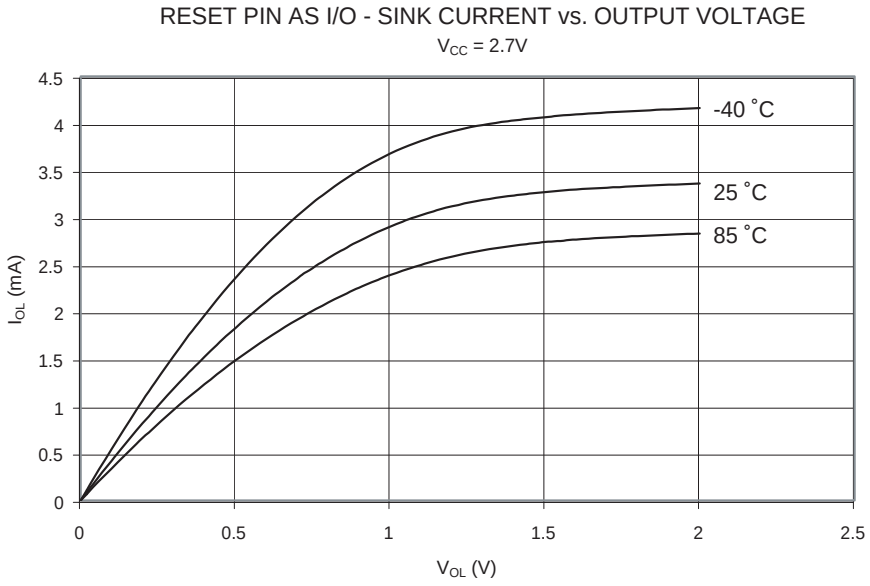
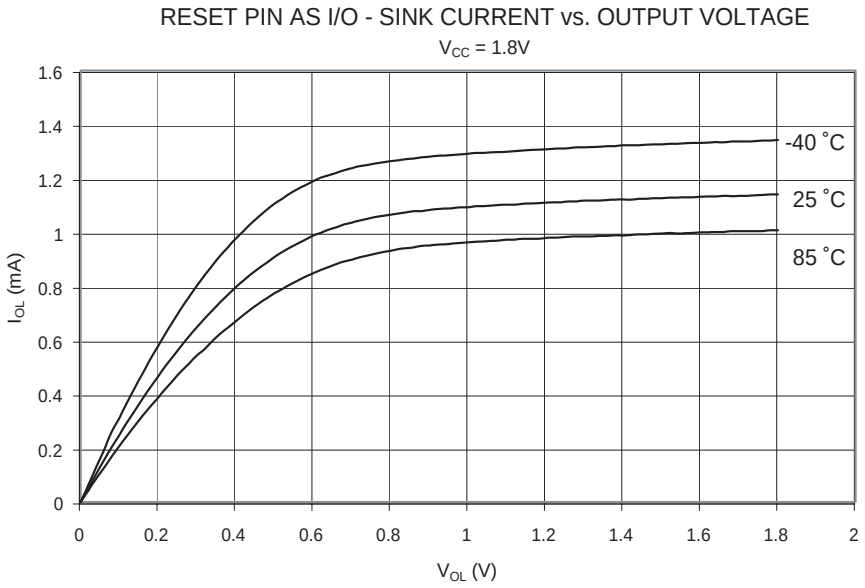


Figure 98. Reset 引脚作为 I/O – 吸收电流和输出电压的关系 (V_{CC} = 1.8V)



引脚门限及滞后

Figure 99. 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')

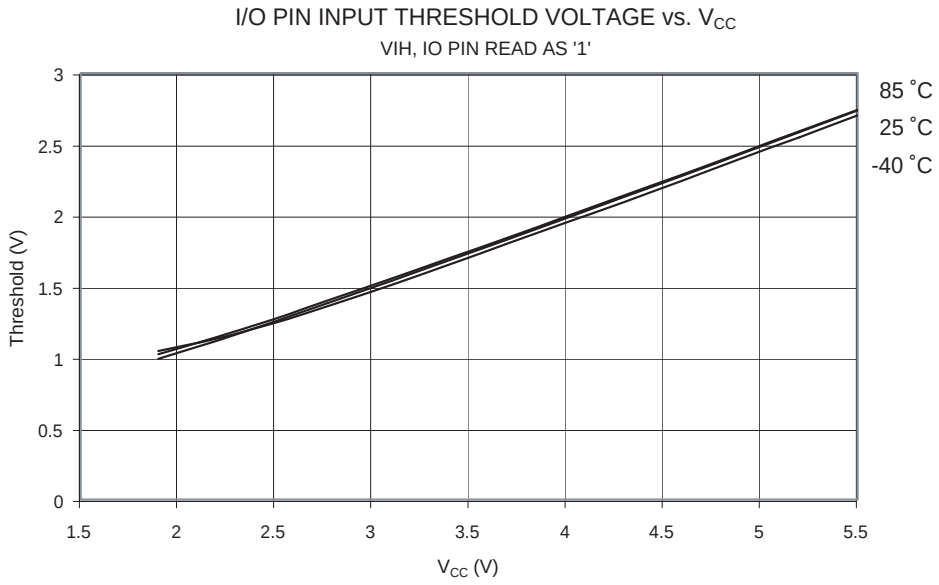


Figure 100. I/O 引脚输入门限电压和 V_{CC} 的关系 (V_{IL}, I/O 引脚读出值为 '0')

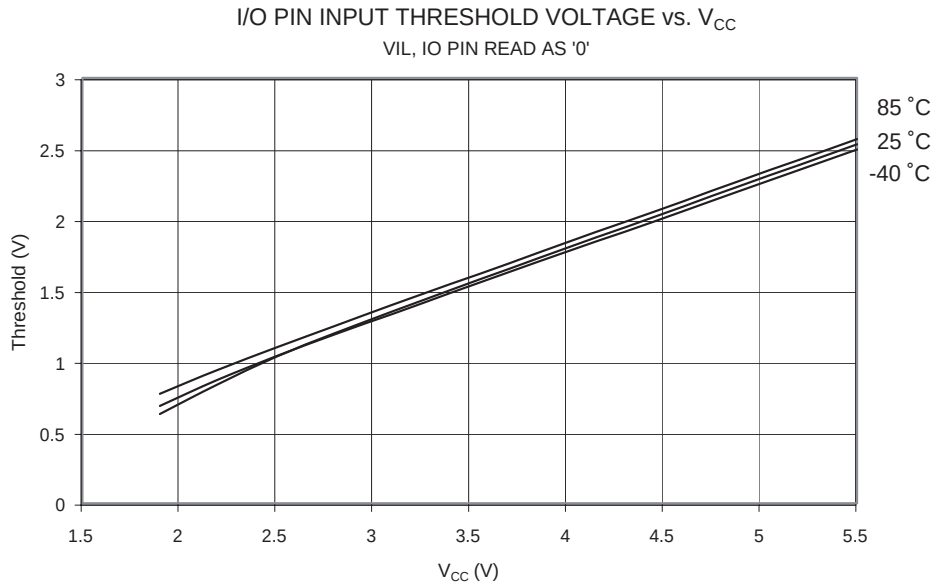


Figure 101. I/O 引脚输入迟滞和 V_{CC} 的关系

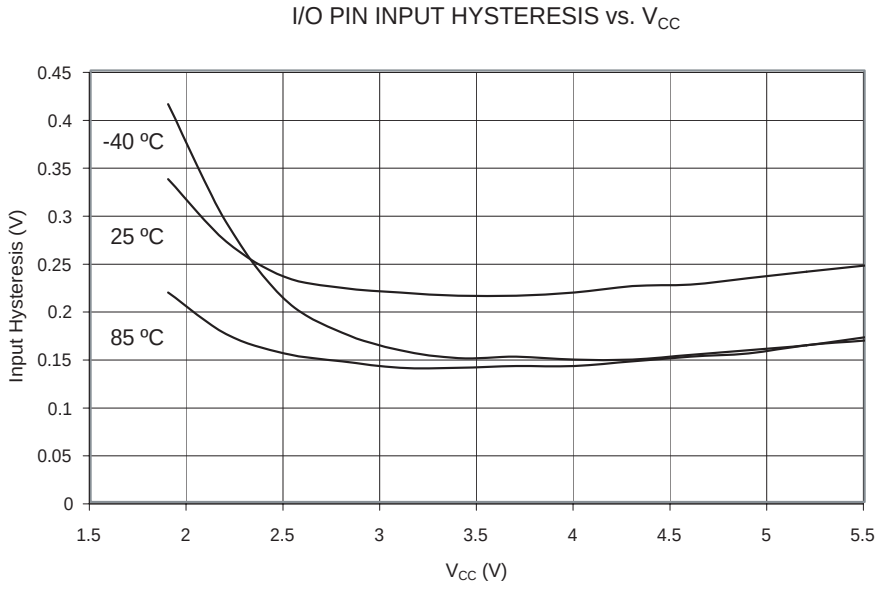


Figure 102. Reset 作为普通 I/O 输入门限电压和 V_{CC} 的关系 (V_{IH},Reset 引脚读出值为 '1')

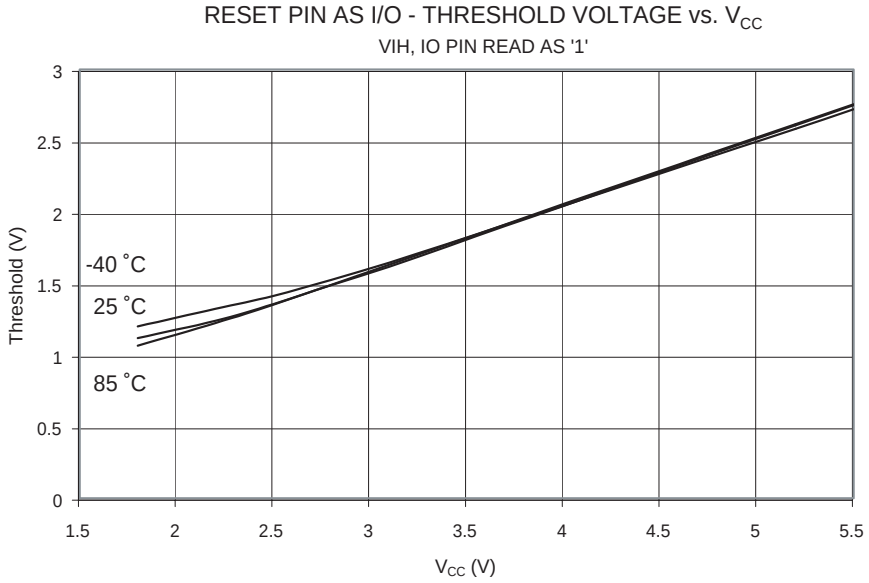


Figure 103. Reset 作为普通 I/O 输入门限电压和 V_{CC} 的关系 (V_{IL},Reset 引脚读出值为 '0')

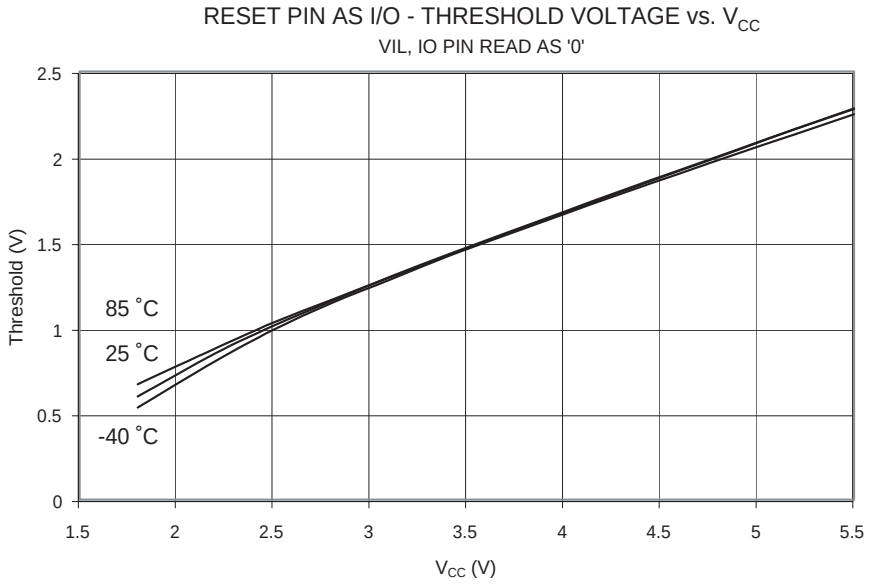


Figure 104. Reset 输入迟滞和 V_{CC} 的关系

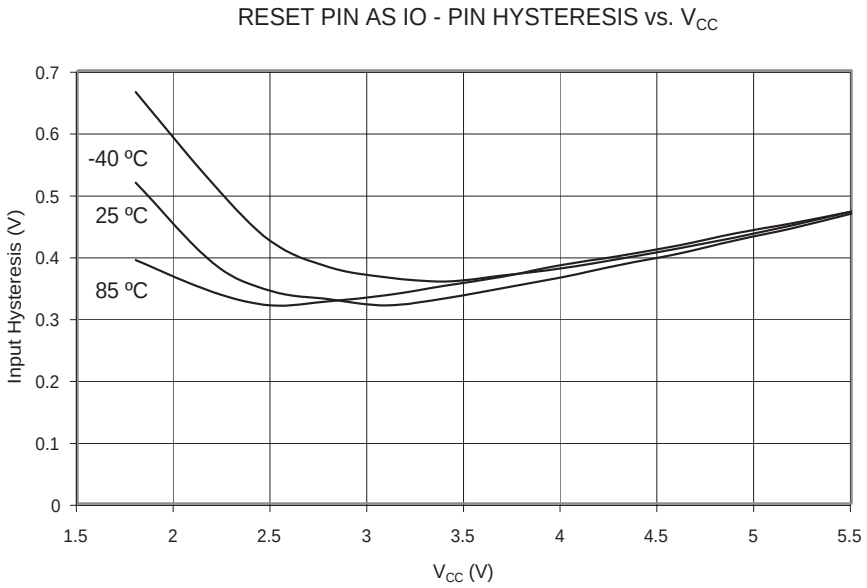


Figure 105. Reset 输入门限电压和 V_{CC} 的关系 (V_{IH},Reset 引脚读出值为 '1')

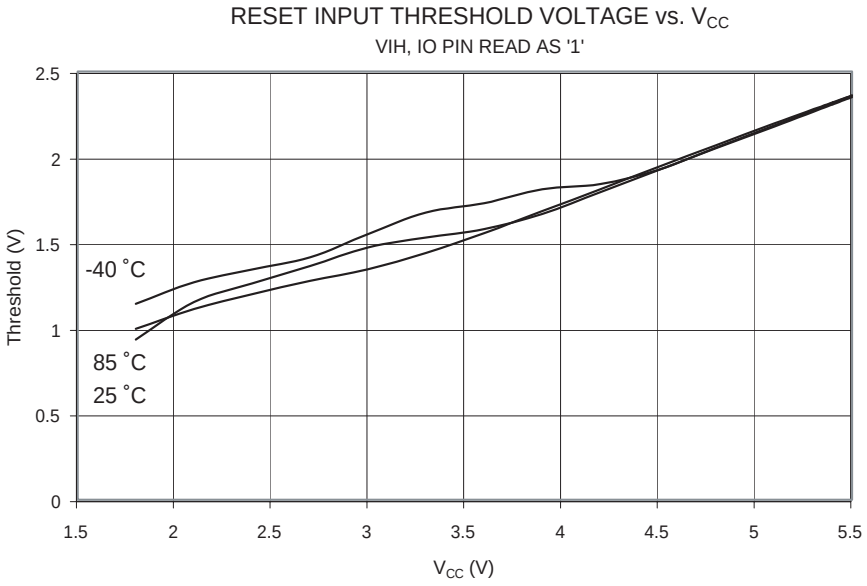


Figure 106. Reset 输入门限电压和 V_{CC} 的关系 (V_{IL},Reset 引脚读出值为 '0')

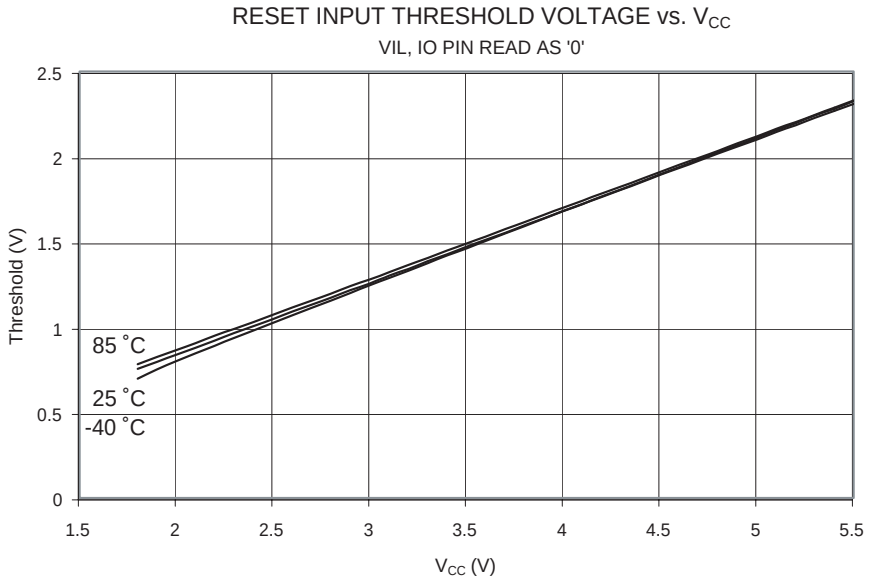
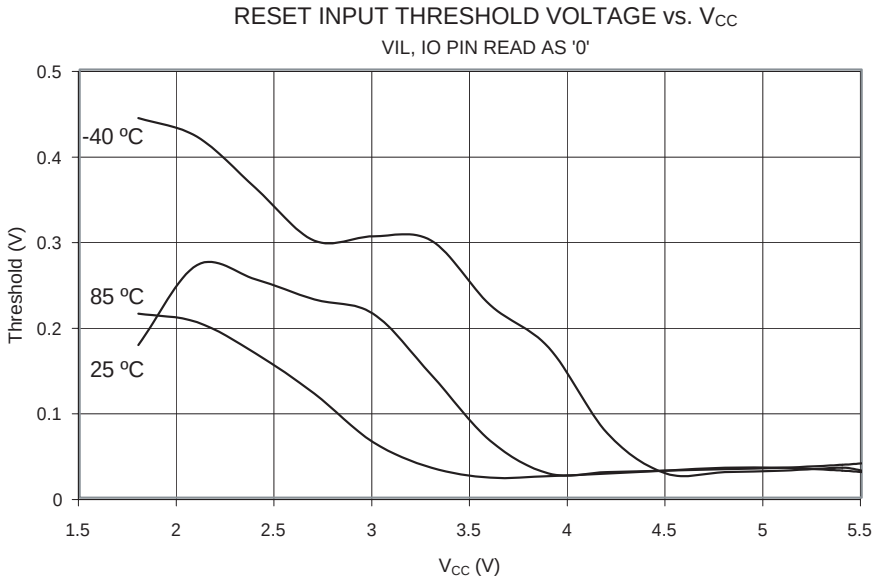


Figure 107. Reset 输入迟滞和 V_{CC} 的关系



BOD 门限值与模拟比较器偏移量

Figure 108. BOD 门限值和温度的关系 (BOD 电平为 4.3V)

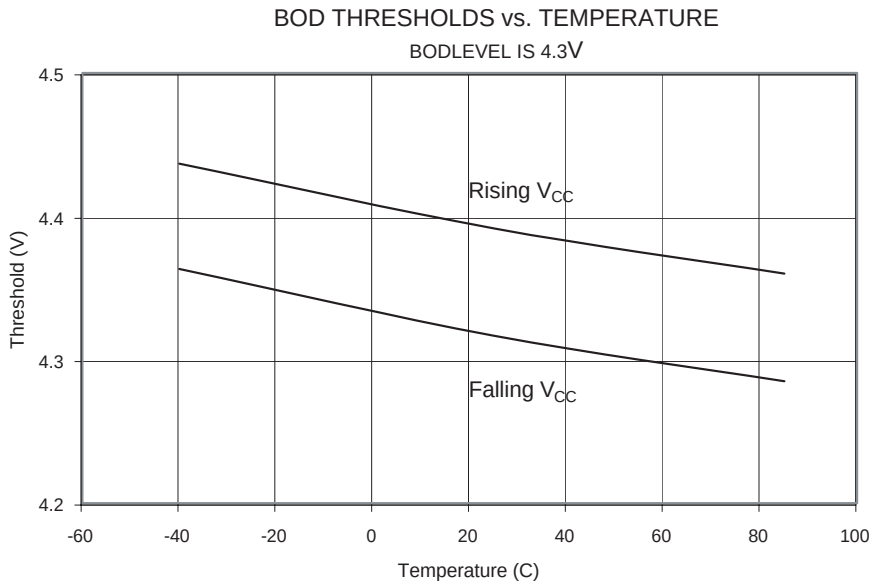


Figure 109. BOD 门限值和温度的关系 (BOD 电平为 2.7V)

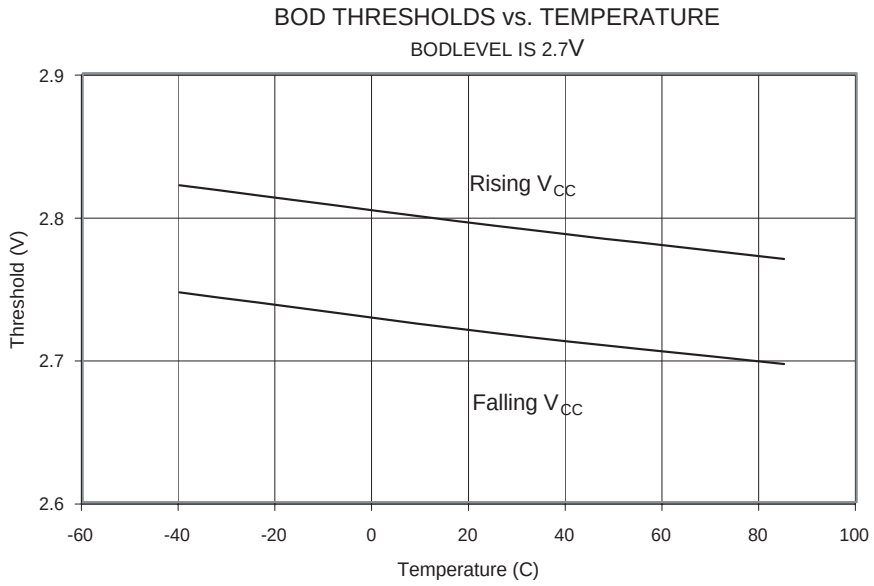


Figure 110. BOD 门限值和温度的关系 (BOD 电平为 1.8V)

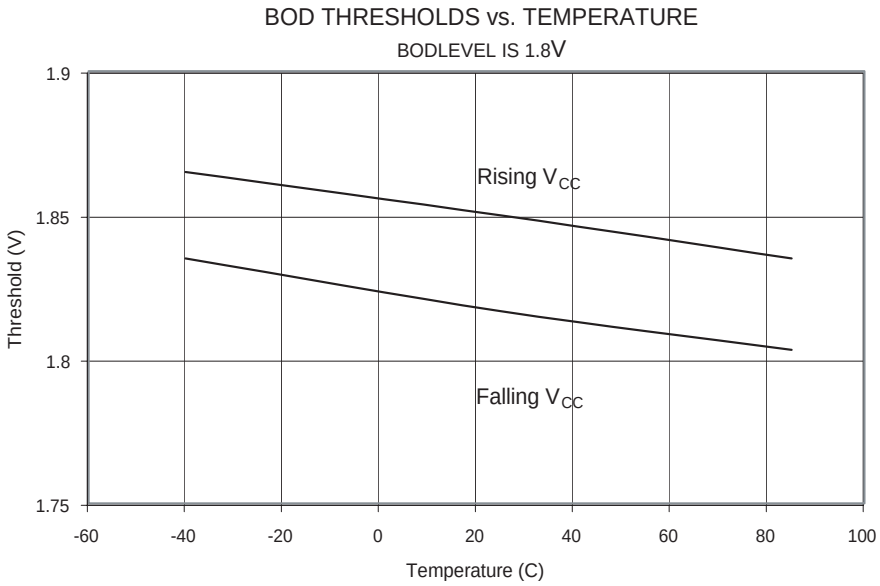


Figure 111. 能隙电压和 V_{CC} 的关系

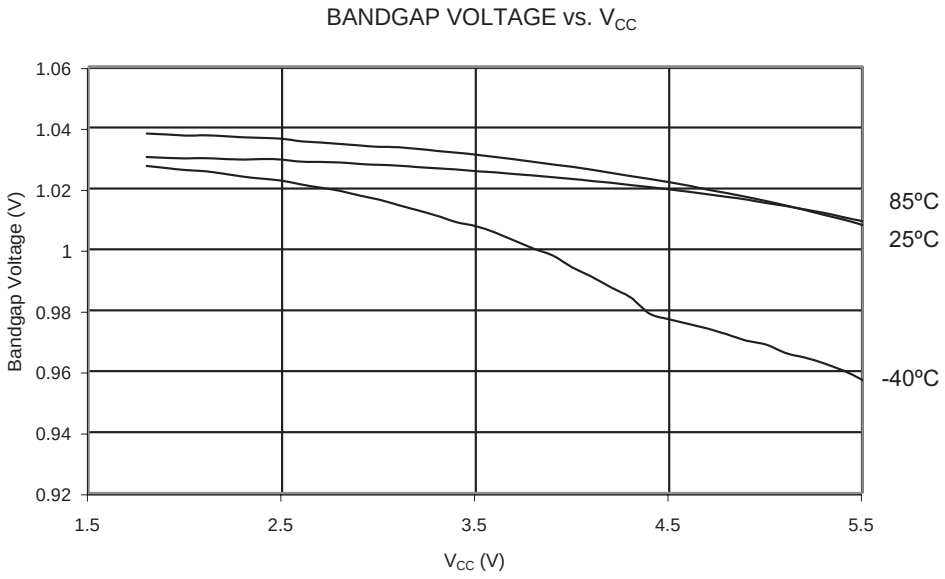


Figure 112. 模拟比较器偏置电压和共模电压的关系 (V_{CC} = 5V)

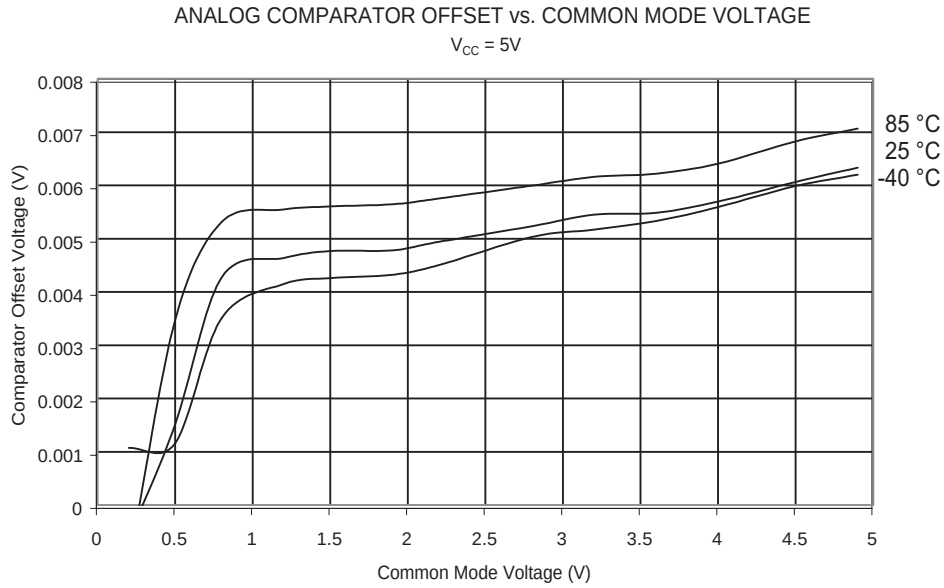
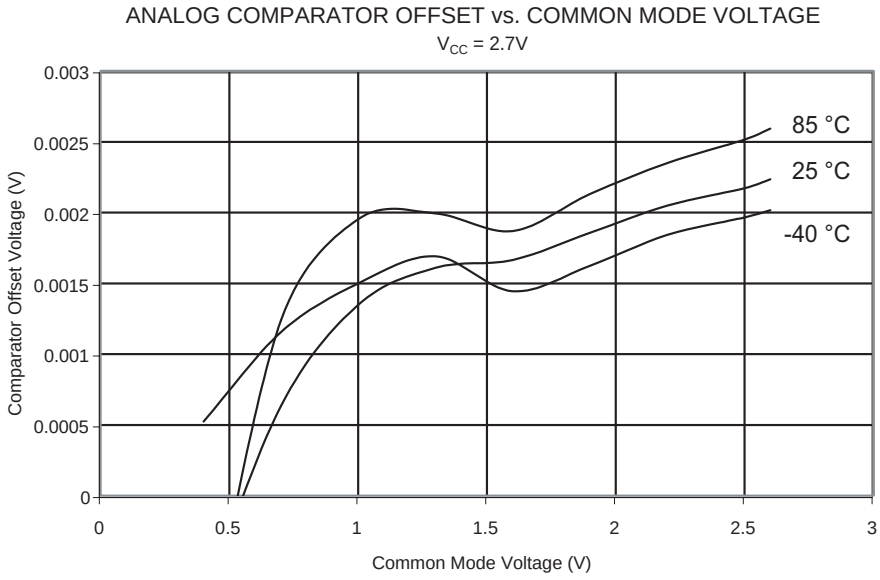


Figure 113. 模拟比较器偏置电压和共模电压的关系 (V_{CC}= 2.7V)



内部振荡器速度

Figure 114. 标定 9.6MHz RC 振荡器频率与温度的关系

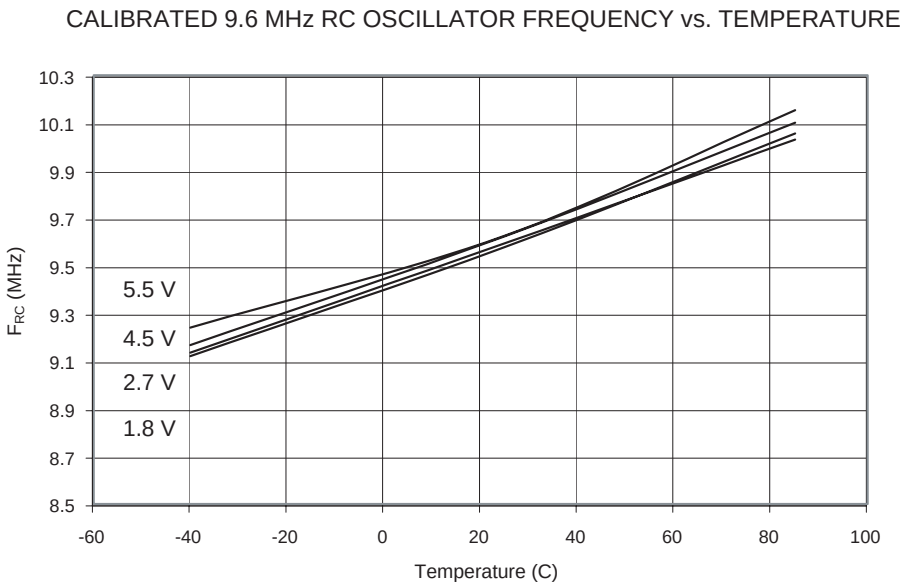


Figure 115. 标定 9.6MHz RC 振荡器频率与 V_{CC} 的关系

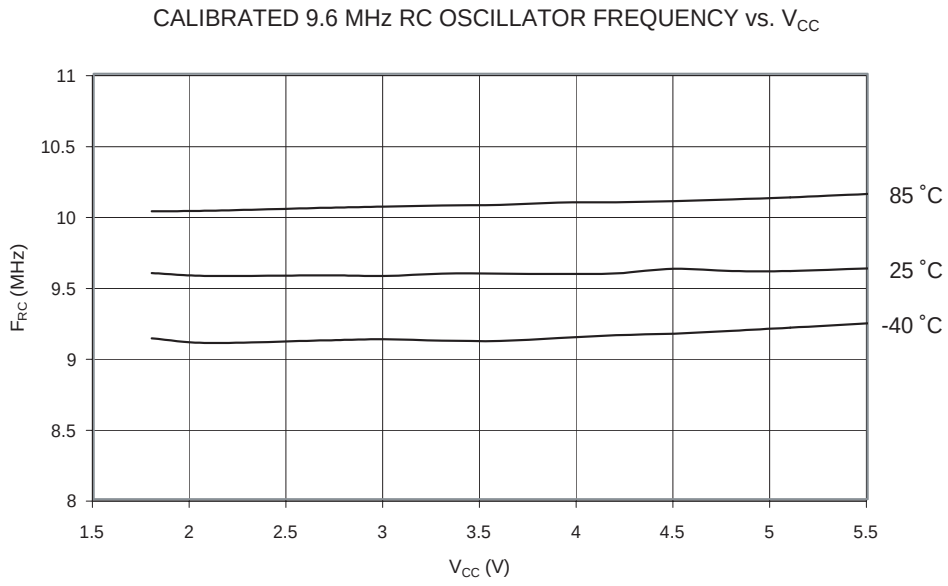


Figure 116. 标定 9.6MHz RC 振荡器频率与 Osccal 值的关系

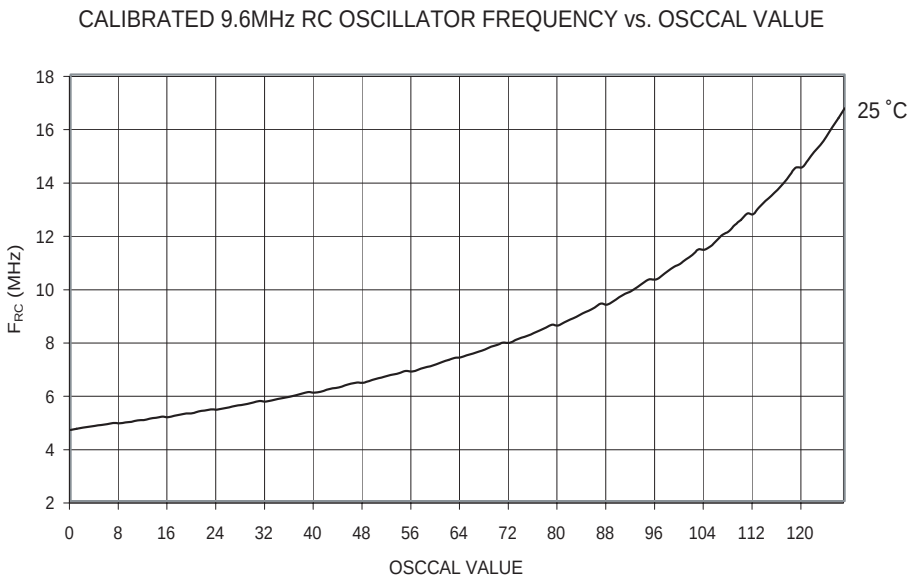


Figure 117. 标定 4.8 MHz RC 振荡器频率与温度的关系

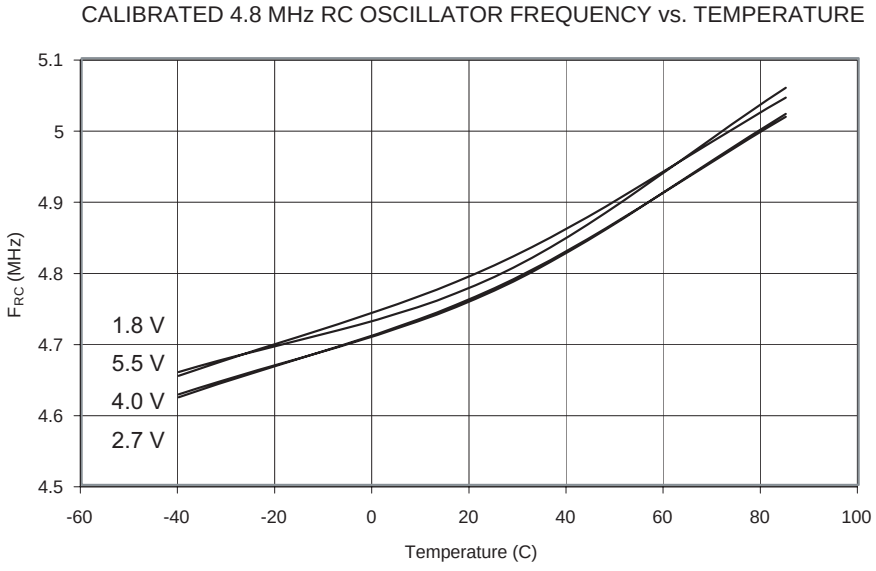


Figure 118. 标定 4.8 MHz RC 振荡器频率与 V_{CC} 的关系

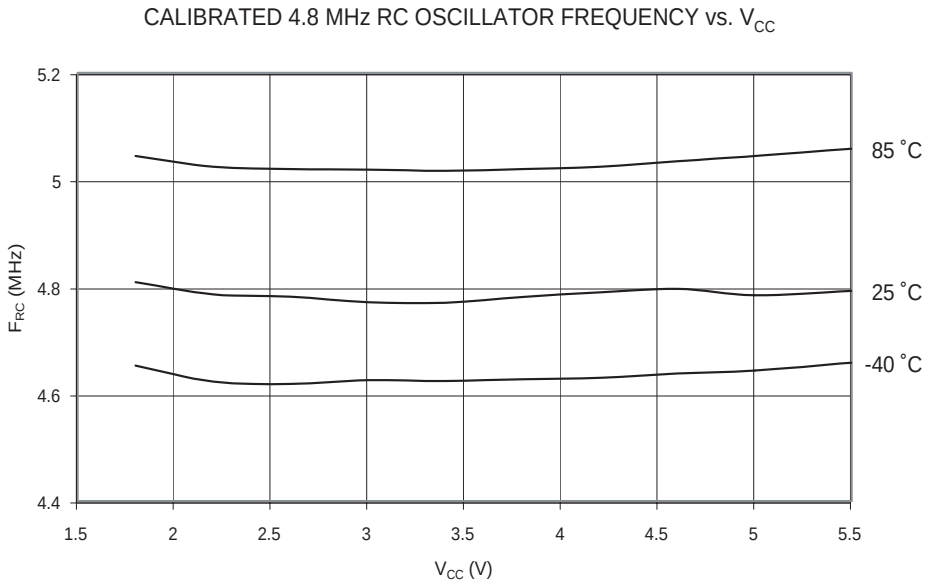


Figure 119. 标定 4.8 MHz RC 振荡器频率与 Osccal 值的关系

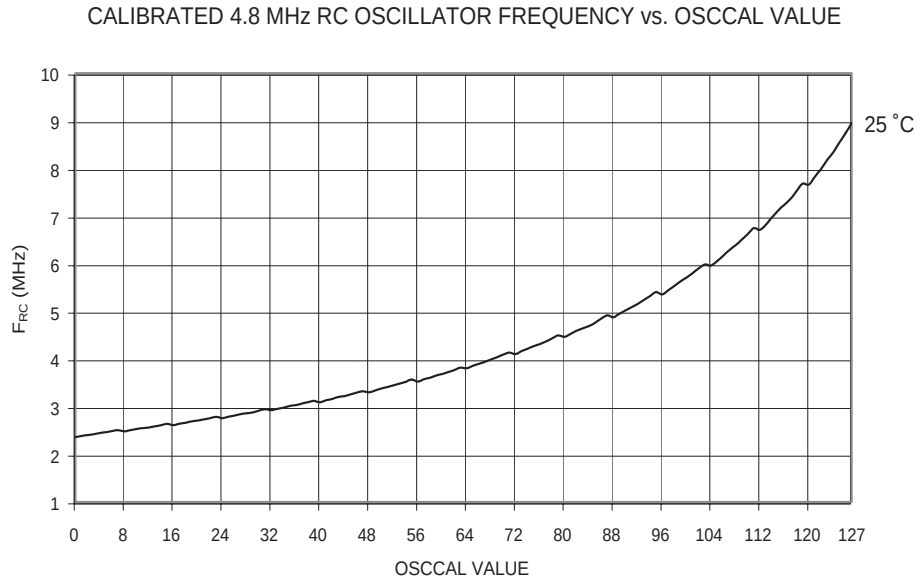


Figure 120. 128 kHz 看门狗振荡器频率与 V_{CC} 的关系

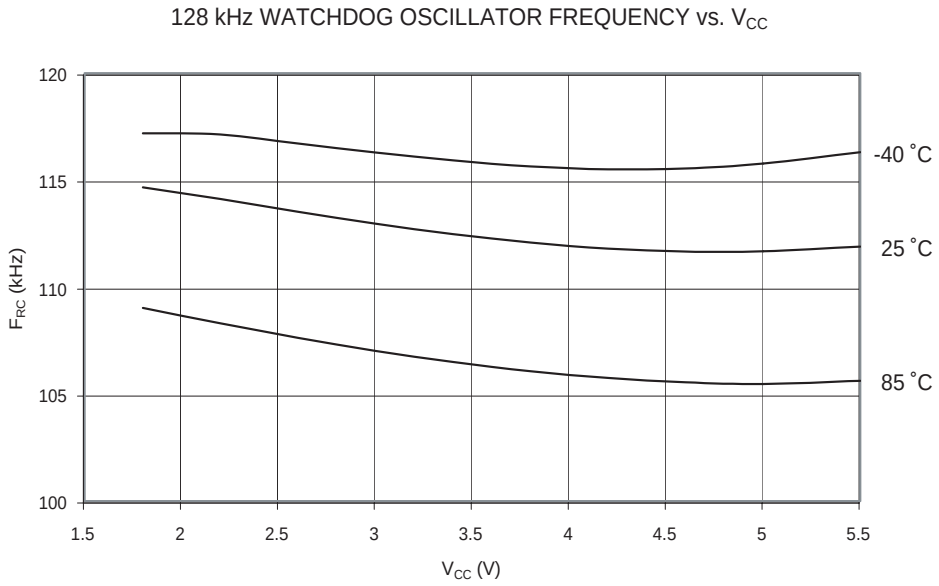
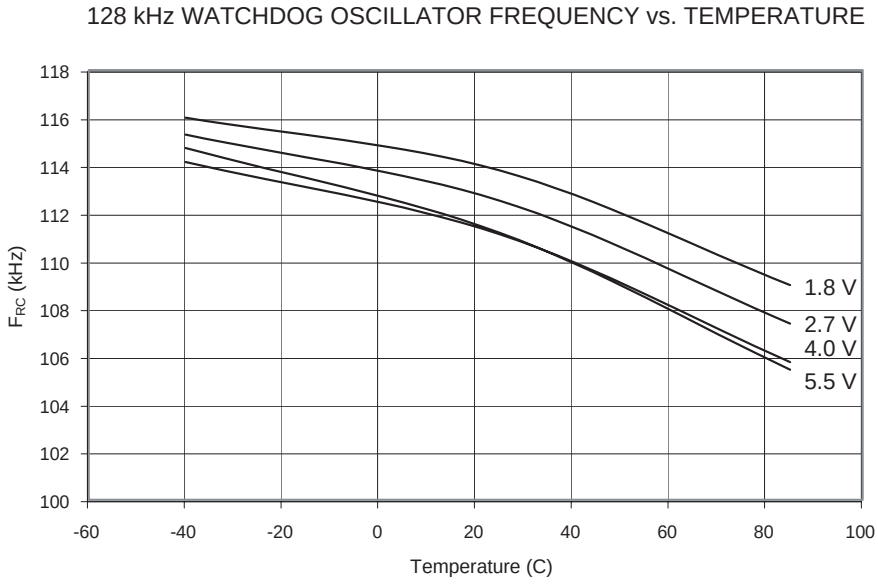


Figure 121. 128 kHz 看门狗振荡器频率与温度的关系



外设电流

Figure 122. BOD 电流和 V_{CC} 的关系

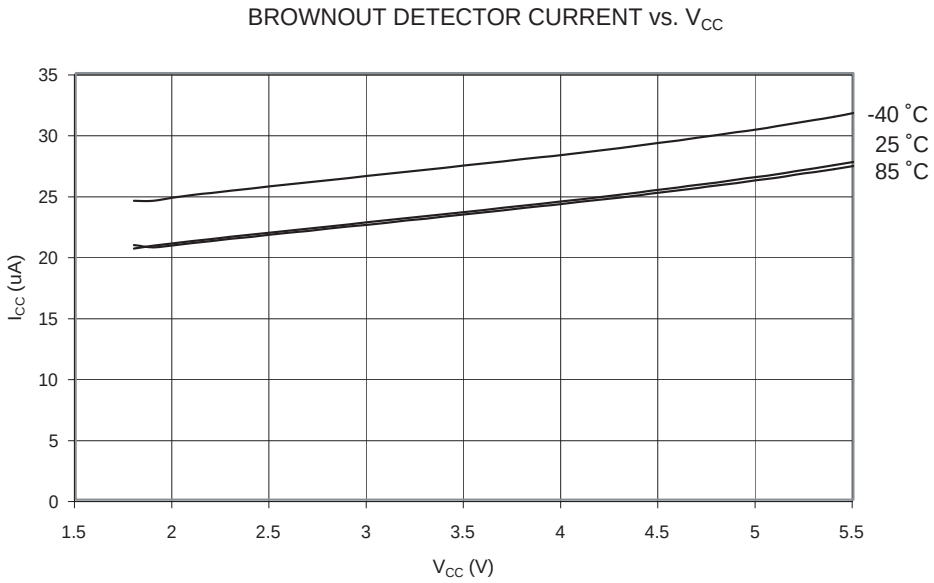


Figure 123. ADC 电流和 V_{CC} 的关系

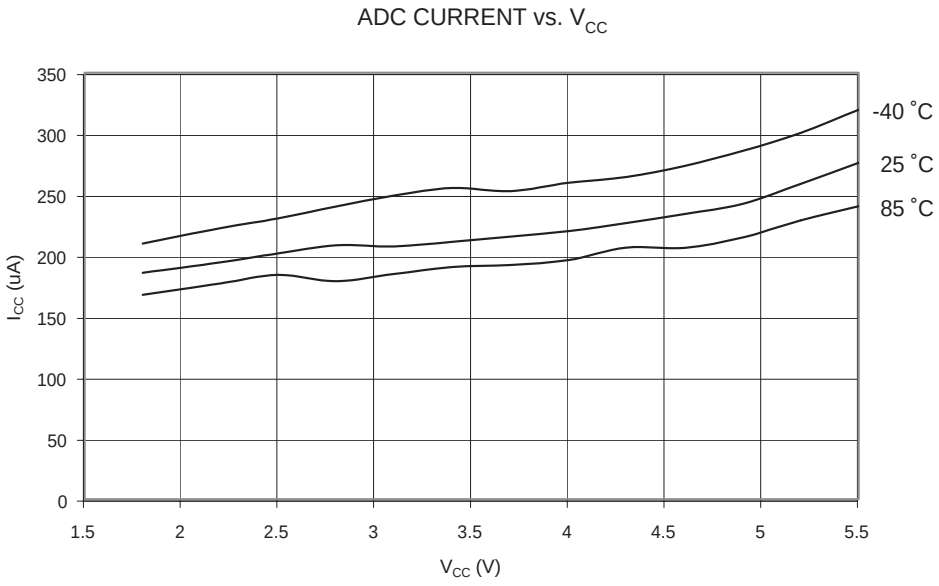


Figure 124. 模拟比较器电流和 V_{CC} 的关系

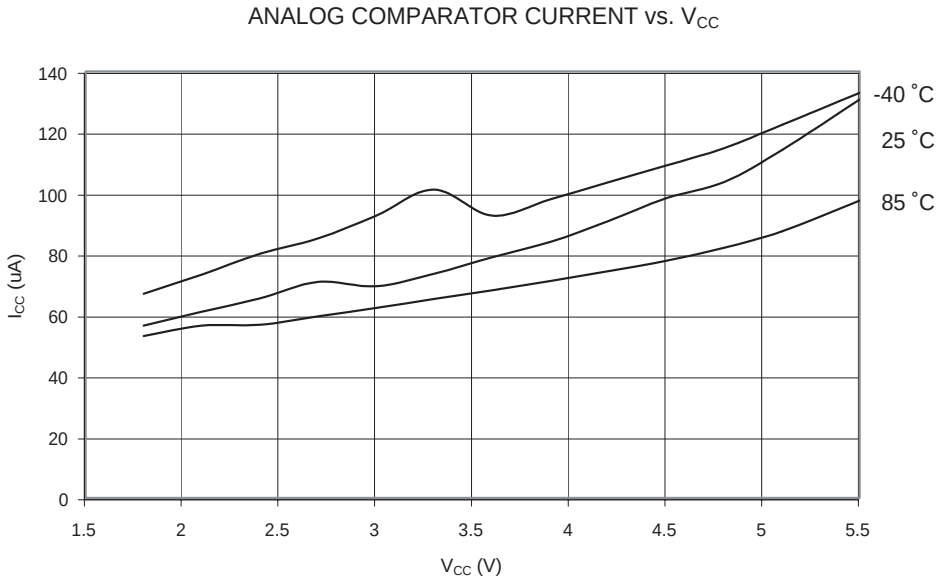
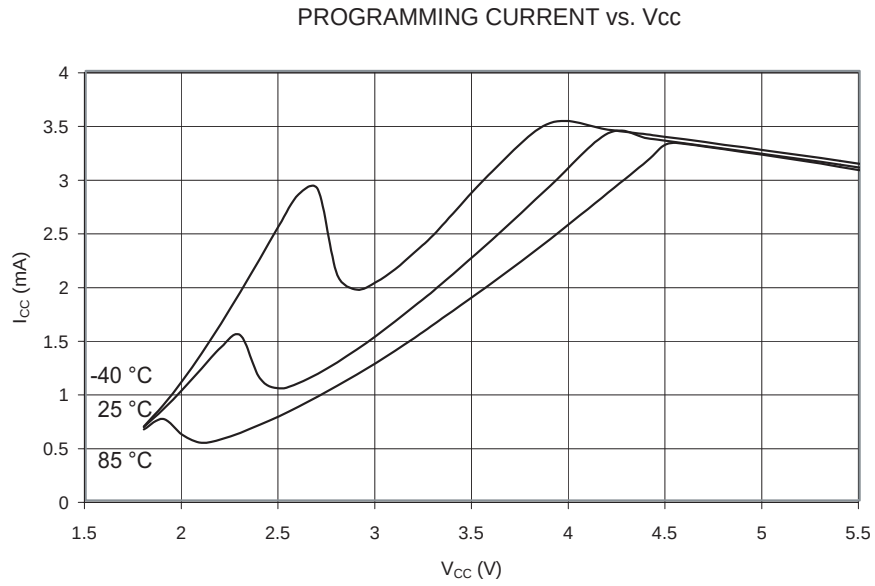


Figure 125. 编程电流与 V_{CC} 的关系



复位电流消耗及复位脉宽

Figure 126. 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)

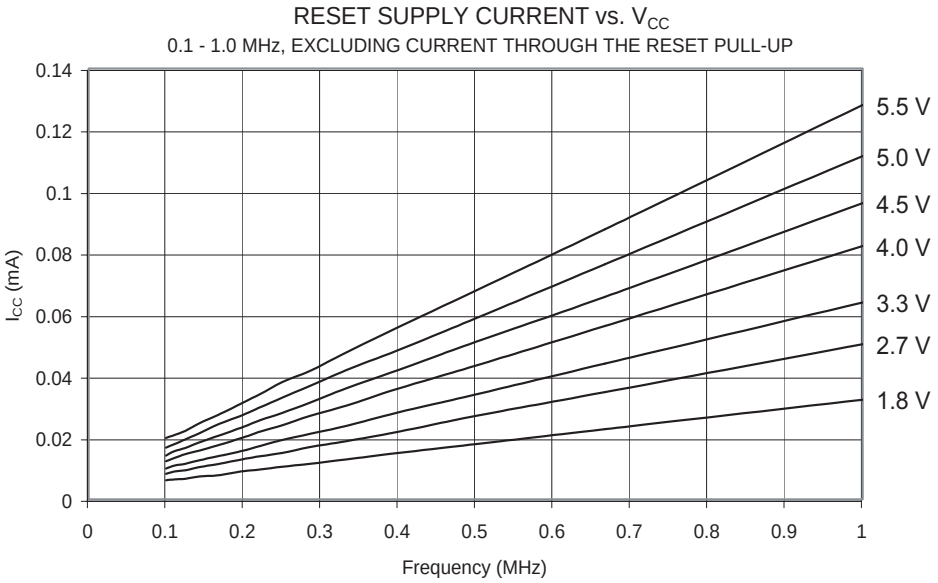


Figure 127. 复位电流和 V_{CC} 的关系 (1 - 24MHz，包括流经复位上拉电阻的电流)

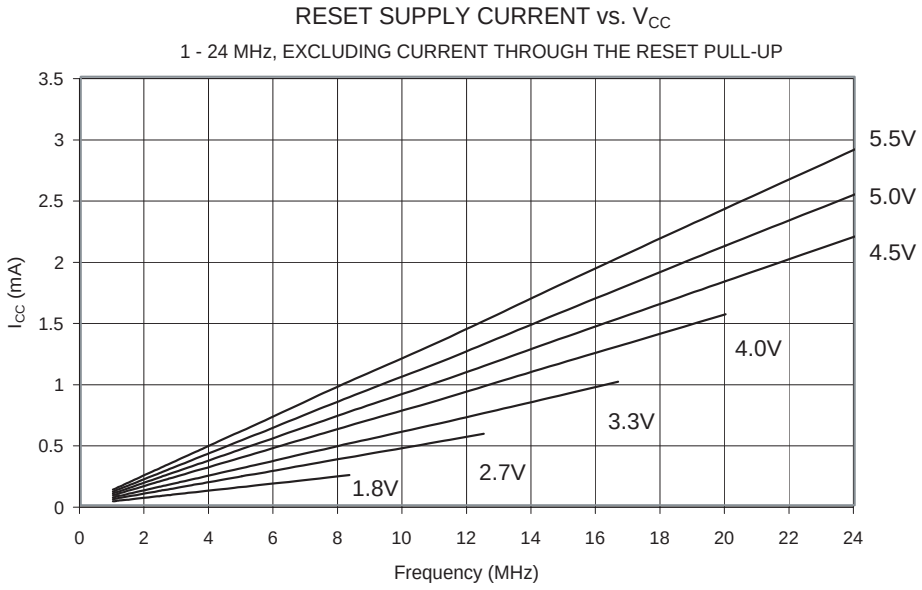
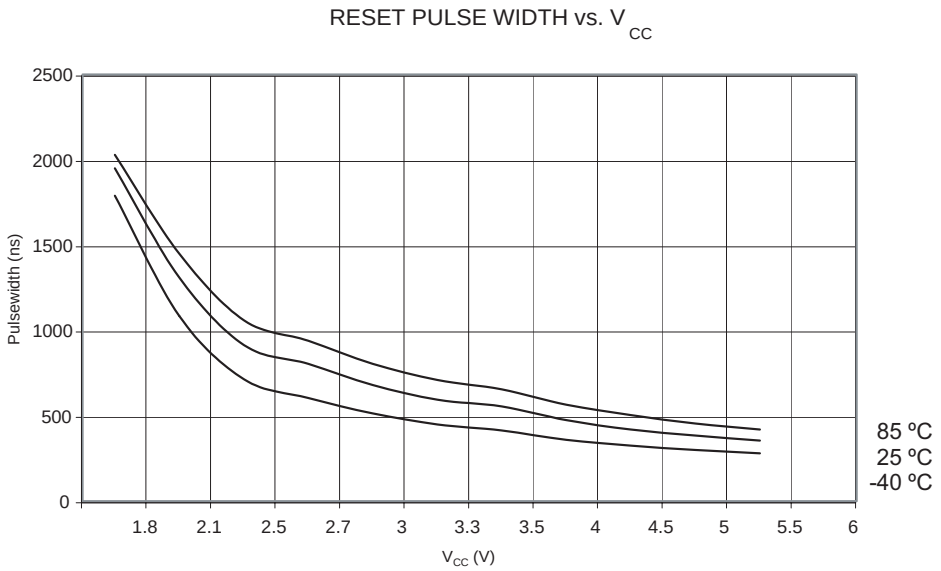


Figure 128. 复位脉宽和 V_{CC} 的关系



寄存器概述

地址	名称	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	页码
0x3F	SREG	I	T	H	S	V	N	Z	C	P6
0x3E	保留	–	–	–	–	–	–	–	–	
0x3D	SPL	SP[7:0]								P8
0x3C	保留	–								
0x3B	GIMSK	–	INT0	PCIE	–	–	–	–	–	P52
0x3A	GIFR	–	INTF0	PCIF	–	–	–	–	–	P52
0x39	TIMSK0	–	–	–	–	OCIE0B	OCIE0A	TOIE0	–	P69
0x38	TIFR0	–	–	–	–	OCF0B	OCF0A	TOV0	–	P69
0x37	SPMCSR	–	–	–	CTPB	RFLB	PGWRT	PGERS	SELFPRGEN	P94
0x36	OCR0A	T/C – 输出比较寄存器 A								P69
0x35	MCUCR	–	PUD	SE	SM1	SM0	–	ISC01	ISC00	P48
0x34	MCUSR	–	–	–	–	WDRF	BORF	EXTRF	PORF	P33
0x33	TCCR0B	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	P65
0x32	TCNT0	T/C0 (8 位)								P69
0x31	OSCCAL	振荡器标定寄存器								P22
0x30	保留	–								
0x2F	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	P68
0x2E	DWDR	DWDR[7:0]								P91
0x2D	保留	–								
0x2C	保留	–								
0x2B	保留	–								
0x2A	保留	–								
0x29	OCR0B	T/C– 输出比较寄存器 B								P69
0x28	GTCCR	TSM	–	–	–	–	–	–	PSR10	P72
0x27	保留	–								
0x26	CLKPR	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0	P24
0x25	保留	–								
0x24	保留	–								
0x23	保留	–								
0x22	保留	–								
0x21	WDTCR	WDTIF	WDTIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	P37
0x20	保留	–								
0x1F	保留	–								
0x1E	EEARL	–	–	EEPROM 地址寄存器						P14
0x1D	EEDR	EEPROM 数据寄存器								P14
0x1C	EECR	–	–	EEPM1	EEPM0	EERIE	EEMWE	EEWE	EERE	P15
0x1B	保留	–								
0x1A	保留	–								
0x19	保留	–								
0x18	PORTB	–	–	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	P50
0x17	DDRB	–	–	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	P50
0x16	PINB	–	–	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	P50
0x15	PCMSK	–	–	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	P53
0x14	DIDR0	–	–	ADC0D	ADC2D	ADC3D	ADC1D	EIN1D	AIN0D	P75, P88
0x13	保留	–								
0x12	保留	–								
0x11	保留	–								
0x10	保留	–								
0x0F	保留	–								
0x0E	保留	–								
0x0D	保留	–								
0x0C	保留	–								
0x0B	保留	–								
0x0A	保留	–								
0x09	保留	–								
0x08	ACSR	ACD	ACBG	ACO	ACI	ACIE	–	ACIS1	ACIS0	P73
0x07	ADMUX	–	REFS0	ADLAR	–	–	–	MUX1	MUX0	P86
0x06	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	P86
0x05	ADCH	ADC 数据寄存器高字节								P87
0x04	ADCL	ADC 数据寄存器低字节								P87
0x03	ADCSRB	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	P88
0x02	保留	–								
0x01	保留	–								
0x00	保留	–								

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. CBI 和 SBI 指令可使用的范围只能是地址为 0x00 - 0x1F 的寄存器。在这些寄存器中，单独位值由 SBIS 与 SBIC 指令检查。
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。

指令集概述

指令	操作数	说明	操作	标志	# 时钟数
算数和逻辑指令					
ADD	Rd, Rr	无进位加法	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	带进位加法	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	立即数与字相加	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	无进位减法	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	减立即数	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	带进位减法	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	带进位减立即数	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	从字中减立即数	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	逻辑与	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	与立即数的逻辑与操作	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	逻辑或	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	与立即数的逻辑或操作	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	异或	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	1 的补码	Rd ← 0xFF – Rd	Z,C,N,V	1
NEG	Rd	2 的补码	Rd ← 0x00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	设置寄存器的位	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	寄存器位清零	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	加一操作	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	减一操作	Rd ← Rd – 1	Z,N,V	1
TST	Rd	测试是否为零或负	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	寄存器清零	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	寄存器置位	Rd ← 0xFF	None	1
跳转指令					
RJMP	k	相对跳转	PC ← PC + k + 1	None	2
IJMP		间接跳转到 (Z)	PC ← Z	None	2
RCALL	k	相对子程序调用	PC ← PC + k + 1	None	3
ICALL		间接调用 (Z)	PC ← Z	None	3
RET		子程序返回	PC ← STACK	None	4
RETI		中断返回	PC ← STACK	I	4
CPSE	Rd,Rr	比较，相等则跳过下一条指令	if (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
CP	Rd,Rr	比较	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	带进位比较	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	与立即数比较	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	寄存器位为 "0" 则跳过下一条指令	if (Rr(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBRS	Rr, b	寄存器位为 "1" 则跳过下一条指令	if (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIC	P, b	I/O 寄存器位为 "0" 则跳过下一条指令	if (P(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBIS	P, b	I/O 寄存器位为 "1" 则跳过下一条指令	if (P(b)=1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	状态寄存器位为 "1" 则跳过下一条指令	if (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	状态寄存器位为 "0" 则跳过下一条指令	if (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	相等则跳转	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	不相等则跳转	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	进位位为 "1" 则跳转	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	进位位为 "0" 则跳转	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	大于或等于则跳转	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	小于则跳转	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	负则跳转	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	正则跳转	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	有符号数大于或等于则跳转	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2
BRLT	k	有符号数负则跳转	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	半进位位为 "1" 则跳转	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	半进位位为 "0" 则跳转	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	T 为 "1" 则跳转	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	T 为 "0" 则跳转	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	溢出标志为 "1" 则跳转	if (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	溢出标志为 "0" 则跳转	if (V = 0) then PC ← PC + k + 1	None	1/2
BRIE	k	中断使能则跳转	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	中断禁止则跳转	if (I = 0) then PC ← PC + k + 1	None	1/2
位和位测试指令					
SBI	P,b	I/O 寄存器位置位	I/O(P,b) ← 1	None	2
CBI	P,b	I/O 寄存器位清零	I/O(P,b) ← 0	None	2
LSL	Rd	逻辑左移	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	逻辑右移	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	带进位循环左移	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1

指令	操作数	说明	操作	标志	# 时钟数
ROR	Rd	带进位循环右移	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	算术右移	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	高低半字节交换	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	标志置位	SREG(s) ← 1	SREG(s)	1
BCLR	s	标志清零	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	从寄存器将位赋给 T	T ← Rr(b)	T	1
BLD	Rd, b	将 T 赋给寄存器位	Rd(b) ← T	None	1
SEC		进位位置位	C ← 1	C	1
CLC		进位位清零	C ← 0	C	1
SEN		负标志位置位	N ← 1	N	1
CLN		负标志位清零	N ← 0	N	1
SEZ		零标志位置位	Z ← 1	Z	1
CLZ		零标志位清零	Z ← 0	Z	1
SEI		全局中断使能	I ← 1	I	1
CLI		全局中断禁用	I ← 0	I	1
SES		符号测试标志位置位	S ← 1	S	1
CLS		符号测试标志位清零	S ← 0	S	1
SEV		2 的补码溢出标志置位	V ← 1	V	1
CLV		2 的补码溢出标志清零	V ← 0	V	1
SET		SREG 的 T 置位	T ← 1	T	1
CLT		SREG 的 T 清零	T ← 0	T	1
SEH		SREG 的半进位标志置位	H ← 1	H	1
CLH		SREG 的半进位标志清零	H ← 0	H	1
数据传送指令					
MOV	Rd, Rr	寄存器间复制	Rd ← Rr	None	1
MOVW	Rd, Rr	复制寄存器字	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	加载立即数	Rd ← K	None	1
LD	Rd, X	加载间接寻址数据	Rd ← (X)	None	2
LD	Rd, X+	加载间接寻址数据，然后地址加一	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	地址减一后加载间接寻址数据	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	加载间接寻址数据	Rd ← (Y)	None	2
LD	Rd, Y+	加载间接寻址数据，然后地址加一	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	地址减一后加载间接寻址数据	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	加载带偏移量的间接寻址数据	Rd ← (Y + q)	None	2
LD	Rd, Z	加载间接寻址数据	Rd ← (Z)	None	2
LD	Rd, Z+	加载间接寻址数据，然后地址加一	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	地址减一后加载间接寻址数据	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	加载带偏移量的间接寻址数据	Rd ← (Z + q)	None	2
LDS	Rd, k	从 SRAM 加载数据	Rd ← (k)	None	2
ST	X, Rr	以间接寻址方式存储数据	(X) ← Rr	None	2
ST	X+, Rr	以间接寻址方式存储数据，然后地址加一	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	地址减一后以间接寻址方式存储数据	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	加载间接寻址数据	(Y) ← Rr	None	2
ST	Y+, Rr	加载间接寻址数据，然后地址加一	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	地址减一后加载间接寻址数据	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	加载带偏移量的间接寻址数据	(Y + q) ← Rr	None	2
ST	Z, Rr	加载间接寻址数据	(Z) ← Rr	None	2
ST	Z+, Rr	加载间接寻址数据，然后地址加一	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	地址减一后加载间接寻址数据	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	加载带偏移量的间接寻址数据	(Z + q) ← Rr	None	2
STS	k, Rr	从 SRAM 加载数据	(k) ← Rr	None	2
LPM		加载程序空间的数据	R0 ← (Z)	None	3
LPM	Rd, Z	加载程序空间的数据	Rd ← (Z)	None	3
LPM	Rd, Z+	加载程序空间的数据，然后地址加一	Rd ← (Z), Z ← Z+1	None	3
SPM		保存程序空间的数据	(z) ← R1:R0	None	
IN	Rd, P	从 I/O 端口读数据	Rd ← P	None	1
OUT	P, Rr	想 I/O 端口输出数据	P ← Rr	None	1
PUSH	Rr	将寄存器推入堆栈	STACK ← Rr	None	2
POP	Rd	将寄存器弹出堆栈	Rd ← STACK	None	2
MCU 控制指令					
NOP		空操作		None	1
SLEEP		休眠	(see specific descr. for Sleep function)	None	1
WDR		复位看门狗	(see specific descr. for WDR/Timer)	None	1
BREAK		终止	For On-chip Debug Only	None	N/A

产品信息

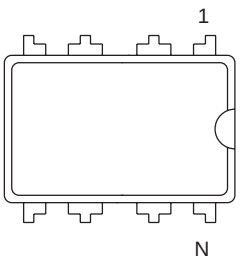
速度 (MHz)	所需电源	产品号	封装	工作范围
10[3]	1.8 - 5.5	ATtiny13V-10PI ATtiny13V-10PJ[2] ATtiny13V-10SI ATtiny13V-10SJ[2] ATtiny13V-10SSI ATtiny13V-10SSJ[2]	8P3 8P3 8S2 8S2 S8S1 S8S1	工业级 (-40°C - 85°C)
20[3]	2.7 - 5.5	ATtiny13-20PI ATtiny13-20PJ[2] ATtiny13-20SI ATtiny13-20SJ[2] ATtiny13-20SSI ATtiny13-20SSJ[2]	8P3 8P3 8S2 8S2 S8S1 S8S1	工业级 (-40°C - 85°C)

Notes: 1. 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。
2. 可选无铅封装。
3. 速度与 V_{CC} 间的关系，见 P114“ 最大速度与 V_{CC} 的关系 ”。

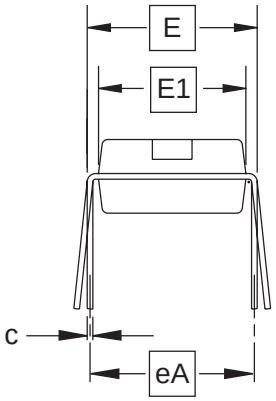
封装类型	
8P3	8- 引线，0.300" 宽，PDIP
8S2	8- 引线，0.209" 宽， EIAJ SOIC
S8S1	8- 引线，0.150" 宽， JEDEC SOIC

封装信息

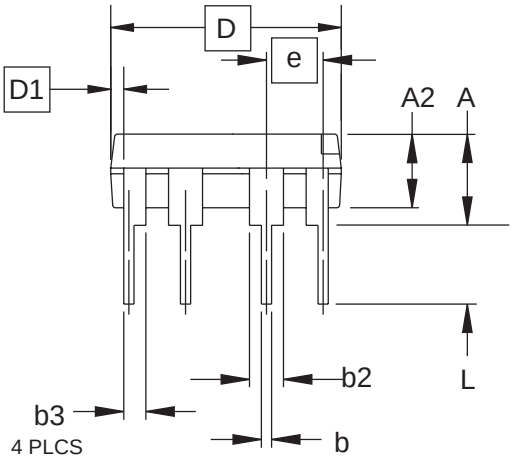
8P3



Top View



End View



Side View

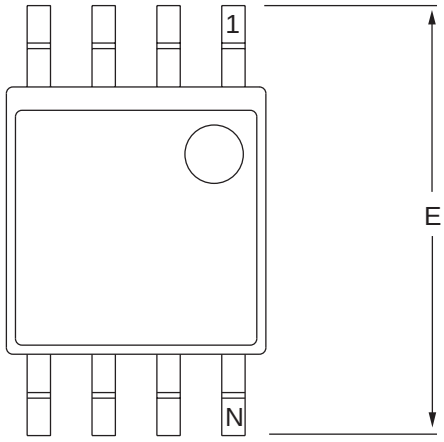
COMMON DIMENSIONS
(Unit of Measure = inches)

SYMBOL	MIN	NOM	MAX	NOTE
A			0.210	2
A2	0.115	0.130	0.195	
b	0.014	0.018	0.022	5
b2	0.045	0.060	0.070	6
b3	0.030	0.039	0.045	6
c	0.008	0.010	0.014	
D	0.355	0.365	0.400	3
D1	0.005			3
E	0.300	0.310	0.325	4
E1	0.240	0.250	0.280	3
e	0.100 BSC			
eA	0.300 BSC			4
L	0.115	0.130	0.150	2

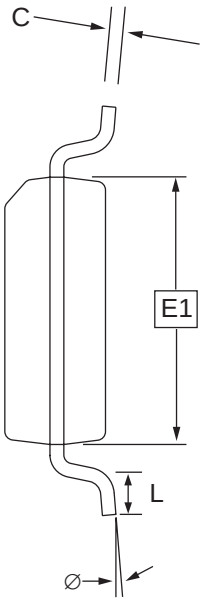
Notes:
1. This drawing is for general information only; refer to JEDEC Drawing MS-001, Variation BA for additional information.
2. Dimensions A and L are measured with the package seated in JEDEC seating plane Gauge GS-3.
3. D, D1 and E1 dimensions do not include mold Flash or protrusions. Mold Flash or protrusions shall not exceed 0.010 inch.
4. E and eA measured with the leads constrained to be perpendicular to datum.
5. Pointed or rounded lead tips are preferred to ease insertion.
6. b2 and b3 maximum dimensions do not include Dambar protrusions. Dambar protrusions shall not exceed 0.010 (0.25 mm).

01/09/02

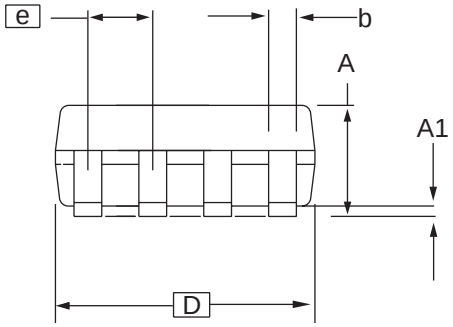
	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	**8P3**, 8-lead, 0.300" Wide Body, Plastic Dual In-line Package (PDIP)	8P3	B



Top View



End View



Side View

COMMON DIMENSIONS
(Unit of Measure = mm)

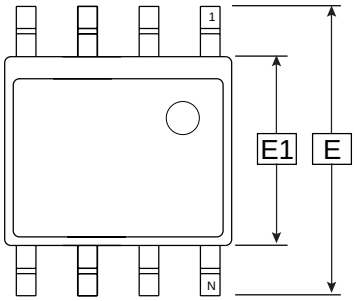
SYMBOL	MIN	NOM	MAX	NOTE
A	1.70		2.16	
A1	0.05		0.25	
b	0.35		0.48	5
C	0.15		0.35	5
D	5.13		5.35	
E1	5.18		5.40	2, 3
E	7.70		8.26	
L	0.51		0.85	
∅	0°		8°	
e	1.27 BSC			4

Notes: 1. This drawing is for general information only; refer to EIAJ Drawing EDR-7320 for additional information.
2. Mismatch of the upper and lower dies and resin burrs are not included.
3. It is recommended that upper and lower cavities be equal. If they are different, the larger dimension shall be regarded.
4. Determines the true geometric position.
5. Values b and C apply to pb/Sn solder plated terminal. The standard thickness of the solder layer shall be 0.010 +0.010/–0.005 mm.

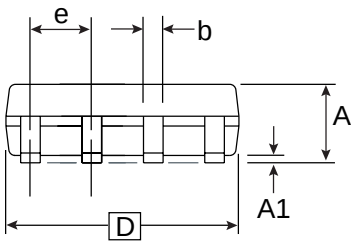
10/7/03

	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	**8S2**, 8-lead, 0.209" Body, Plastic Small Outline Package (EIAJ)	8S2	C

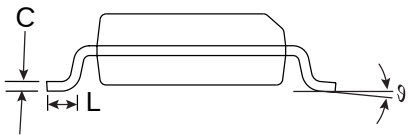
S8S1



Top View



Side View



End View

COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
E	5.79		6.20	
E1	3.81		3.99	
A	1.35		1.75	
A1	0.1		0.25	
D	4.80		4.98	
C	0.17		0.25	
b	0.31		0.51	
L	0.4		1.27	
e	1.27 BSC			
θ	0^o		8^o	

Notes: 1. This drawing is for general information only; refer to JEDEC Drawing MS-012 for proper dimensions, tolerances, datums,etc.

7/28/03

	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	**S8S1**, 8-lead, 0.150" Wide Body, Plastic Gull Wing Small Outline (JEDEC SOIC)	S8S1	A

勘误表

本节修订的为 ATtiny13 芯片。

ATtiny13 Rev. B

- **在擦除操作后读返回值错误**
- **对 Flash, EEPROM，熔丝位与锁定位的高电压串行编程可能失败**
- **为进一步编程，芯片可能锁定**
- **debugWIRE 通信不会被锁定位阻止**
- **看门狗定时器中断禁用**

1. 在擦除操作后读返回值错误

当电源电压低于 2.7 V 时，被擦除的 EEPROM 地址读返回值可能为 0x00，而不是 0xFF。

解决方法：

如果要在擦除操作之后进行读操作，请使用基本写操作，数据为 0xFF。这同样可以达到擦除的目的。在任何情况下写操作都是正确的。

2. 对 Flash, EEPROM, 熔丝位与锁定位的高电压串行编程可能失败

对其写入可能会出错。

解决方法：

在写入初始化后，始终检测 RDY/$\overline{\text{BSY}}$ 信号。若写入出错，就重新写入，直到 RDY/$\overline{\text{BSY}}$ 信号检测到正确的写入。这在修订版 D 中固定下来。

3. 为进一步编程，芯片可能锁定

特殊的熔丝位组合将锁定芯片，将其转换成 OTP 芯片。熔丝位按照下面的设定将会得到这种效果：

– 128 kHz 片内振荡器 (CKSEL[1..0] = 11)，最短的启动时间 (SUT[1..0] = 00)，Debugwire 使能 (DWEN = 0) 或复位禁用 RSTDISBL = 0。

– 9.6 MHz 片内振荡器 (CKSEL[1..0] = 10)，最短的启动时间 (SUT[1..0] = 00)，Debugwire 使能 (DWEN = 0) 或复位禁用 RSTDISBL = 0。

– 4.8 MHz 片内振荡器 (CKSEL[1..0] = 01)，最短的启动时间 (SUT[1..0] = 00)，Debugwire 使能 (DWEN = 0) 或复位禁用 RSTDISBL = 0。

解决方法：

避免出现上述熔丝位组合。选择长启动时间将会消除这一问题。

4. debugWIRE 通信不会被锁定位阻止

当 debugWIRE 片上调试使能 (DWEN = 0)，即使锁定位设置为阻止从芯片读取，还是能读取出程序存储器与 EEPROM 数据存储器中的内容。

解决方法：

产品出厂时，不要将片上调试使能。

5. 看门狗定时器中断禁用

在新的溢出没有出现前若看门狗定时器中断标志没有清除，看门狗将被禁用，且中断标志自动清除。这只在中断模式中出现。如果将看门狗配置为复位模式，芯片在看门狗溢出后有一个中断，芯片工作正常。

解决方法：

保证在新的看门狗溢出出现前有足够的时间处理前一次溢出。这就要选择一个足够长的溢出周期。

ATtiny13 Rev. A

没有抽取修订版 A。

ATtiny13 数据手册改变日志

本节所指的页号为本文的相关页码；修订号则为文档的修订号。

从版本 Rev. 2535D-04/04 到版本 Rev. 2535C-02/04 的改动

1. 最大速度等级改变
   - 从 12MHz 到 10MHz
   - 从 24MHz 到 20MHz
2. 更新 P103“ 串行编程指令集 ”。
3. 更新 P114“ 最大速度与 V_{CC} 的关系 ”。
4. 更新 P154“ 产品信息 ”。

从版本 Rev. 2535B-01/04 到版本 Rev. 2535C-02/04 的改动

1. C 代码例程更新使用合法的 IAR 语法。
2. 由 WDTIF 替代 WDIF ； 由 WDTIE 替代 WDIE。
3. 更新 P8“ 堆栈指针 ”。
4. 更新 P22“ 标定的片内 RC 振荡器 ”。
5. 更新 P22“ 振荡器标定寄存器 – OSCCAL”。
6. 更新 P35“ 看门狗定时器 ” 的介绍方式。
7. 更新 P80“ADC 转换时间 ”。
8. 更新 P100“ 串行下载 ”。
9. 更新 P112“ 电气特性 ”。
10. 更新 P154“ 产品信息 ”。
11. 在 P158“ 勘误表 ” 中删除 rev. C。

从版本 Rev. 2535A-06/03 到版本 Rev. 2535B-01/04 的改动

1. 更新 P2Figure 2。
2. 更新 P30Table 12, P39Table 17, P86Table 37 及 P113Table 57。
3. 更新 P22“ 标定的片内 RC 振荡器 ”。
4. 更新整个 P35“ 看门狗定时器 ”。
5. 更新 P100Figure 53 与 P105Figure 56。
6. 更新 P48“MCU 控制寄存器 – MCUCR” , P68“T/C 控制寄存器 B – TCCR0B” and P75“ 数字输入禁用寄存器 0 – DIDR0” 中的寄存器。
7. 更新 P112“ 电气特性 ” 中绝对最大速率及 DC 特性。
8. 加入 P114“ 最大速度与 V_{CC} 的关系 ”
9. 更新 P115“ADC 特性 - 初始参数 ”。
10. 更新 P116“ATtiny13 典型特征参数 ”。
11. 更新 P154“ 产品信息 ”。
12. 更新 P155“ 封装信息 ”。
13. 更新 P158“ 勘误表 ”。
14. 改变 EEAR 到 EEARL 的例子。