AN3289

tinyAVR® 1 系列的计时功能

特性

- 提供了两种不同的时间和日期计时算法
- 对两种计时方法进行了比较和基准测试
- 介绍了一种用于在 8 位架构上实现快速分频的倒数相乘方法
- 可从 Atmel START 获取四个代码示例
- 用于测试示例功能的 USART 命令接口代码
- 用于自动计算闰年和闹钟功能的代码
- 时序基本信息和定时器/计数器概述

简介

作者：Henrik Nyholm，Microchip Technology Inc.

与 32 位和 64 位架构相比，8 位系统上的计时功能通常更为复杂。在 32 位或 64 位系统上，计时功能通常使用单个 32 位或 64 位变量来实现，此类变量可跟踪从某个纪元或时间戳开始的单个时间单位。通常，这些系统会在指令集中包含快速乘法和除法程序。因此，就指令周期或存储器使用而言，不需要在单变量时间与人类可读的时间之间进行转换。但对于 8 位架构，单变量法有时并非最佳选择。还可以选用另一种方法，也就是使用多个变量来跟踪单独的时间单位，如秒、分钟和小时。有关这两种方法的优缺点，请参见 1. 带时间和日期的计时章节。此外，该章节还介绍了这两种方法的实现方式。

本文还在执行时间和存储器使用方面对单变量法和多变量法的性能进行了比较。为了提高单变量法的性能，我们提出了一种与某个常量的倒数相乘的方法。这种方法无需对 32 位变量执行除法程序，因此可以缩短执行时间并减少闪存使用量。如果经常需要人类可读的格式，多变量法与单变量法相比可以明显减少闪存使用率，并且执行时间更短。采用这种方法时，也无需将单个时间单位转换为人类可读的格式。

Atmel START 中提供了四个用于 ATtiny817 Xplained Pro 的示例项目，它们实现了本应用笔记中讨论的各种计时方法。*仅时间*及*时间和日期*示例中均实现了单变量法和多变量法。在多变量和单变量示例项目中，均可使用万年历（自动闰年计算）。此外还加入了通用同步/异步收发器（Universal Synchronous Asynchronous Receiver Transmitter，USART）命令接口，以便测试各示例的功能。通过该接口，用户可以设置及获取时间和/或日期。此外，多变量示例具有闹钟功能，并且可以通过命令接口连续显示当前时间。各示例中引入的概念适用于任何带 RTC 和 USART 外设的 AVR®单片机。这些示例项目在 4. 从 Atmel | START 获取源代码部分列出。

3. 时序基本信息部分简要介绍了 tinyAVR® 1 系列上的定时器/计数器。2. 硬件要求部分说明了如何将外部 32 kHz 晶振连接到 ATtiny817 Xplained Pro 上的器件。需要使用外部晶振才能在按原样运行示例并准确跟踪时间。第 2 节说明了在连接晶振后如何通过 Micro-USB 端口使用 USART 命令接口。此外，还介绍了一种替代方法，这种方法无需修改工具包即可运行示例，但计时精度将明显降低。

目录

特性 1

简介 1

1. 带时间和日期的计时 3
   - 1.1. 单变量时间 3
   - 1.2. 将时间拆分为多个变量 5
   - 1.3. 闹钟 6
   - 1.4. 比较两种方法 7
   - 1.5. 使用 AVR Libc 和 time.h 7

2. 硬件要求 9
   - 2.1. 连接 32 kHz 晶振 9
   - 2.2. 通过 Micro-USB 使用 USART 连接外部晶振 9
   - 2.3. 在不修改工具包的情况下运行示例 10

3. 时序基本信息 12
   - 3.1. RTC 12
   - 3.2. TCA 12
   - 3.3. TCB 12
   - 3.4. TCD 12

4. 从 Atmel | START 获取源代码 13

5. 附录 A——倒数乘法检查 14

6. 附录 B——代码大小测试详细信息 15

7. 版本历史 16

Microchip 网站 17

产品变更通知服务 17

客户支持 17

Microchip 器件代码保护功能 17

法律声明 17

商标 18

质量管理体系 18

全球销售及服务网点 19

1. 带时间和日期的计时

实时时钟（Real-Time Clock，RTC）是最适合用于计时的定时器/计数器（Timer/Counter，TC）。这是因为 RTC 可以使用 32,768 Hz 晶振作为其时钟源，并且 RTC 预分频器最多可对输入频率进行 32,768 分频，从而提供 1 Hz 节拍。此外，RTC 可在待机休眠模式下运行，并且可用作器件的唤醒源。因此，对于低功耗应用而言，RTC 是一种绝佳选择。在 Atmel START 提供的示例中，RTC 配置为每秒产生一个中断。

可以使用周期性 1 秒中断节拍来递增软件变量，以便跟踪时间。为此，最有效的方式是使用中断服务程序（Interrupt Service Routine，ISR），每次 RTC 寄存器溢出时都会调用此程序。在将此软件变量转换为人类可读的时间和日期时，必须考虑到日历的复杂性。

日期和日历

在跟踪日期和时间时，必须考虑到公历的一些复杂特征。公历使用闰年来补偿地球绕太阳公转需要 365.256 天的事实。闰年的年份数字是 4 的倍数，且不是 100 的倍数，但是 400 的倍数除外。例如，1900 年不是闰年，而 2000 年是闰年。每个闰年的二月份天数都会从 28 天增加到 29 天。

闰秒是无规律插入的秒，用于修正因地球自转变慢而产生的时间差。由于何时插入闰秒没有规律，因此没有公式可以用来计算闰秒。惟一的方式是通过手动调整时间或从全球定位系统（Global Positioning System，GPS）或 Internet 等时间源获取时间更新来自动补偿闰秒。

跟踪实时时间时，应同时考虑到闰年和闰秒。但本应用笔记中未对闰秒进行说明。

振荡器精度

所有时钟源都存在一定的频率误差。与器件中的内部 RC 振荡器相比，晶振在频率输出和稳定性方面通常更为精确，因此它一般用于计时功能。内部 RC 振荡器更适用于对精度要求较低的低功耗应用，并且适合作为脉宽调制（Pulse-Width Modulation，PWM）或串行通信模块等外设的时钟源。

晶振的精度取决于多个因素。最主要的因素如下：

- 总负载电容，包括：
  - PCB 寄生电容
  - 增加的负载电容及其制造商容差
  - 器件中的电容
- 晶振本身的制造商容差
- 由温度引起的标称频率漂移

通常，使用成本相对较低的 32,768 Hz 晶振即可在 25°C 的条件下实现精度为±20 PPM 的设计，但不同晶振之间的温度变化有所不同。应用笔记 AN2711——AVR®单片机的实时时钟校准和补偿中说明了如何补偿这些偏差。有关 tinyAVR 1 系列上时钟和时钟系统的更多信息，请参见相关器件数据手册中的“CLKCTRL——时钟控制器”一章。

1.1 单变量时间

信息： 本手册提供了两个使用单个变量跟踪时间的示例项目，一个项目转换为秒、分钟和小时；另一个项目转换为秒、分钟、小时和日期。后者还考虑到了上一节中所述的闰年计算。有关这些示例的链接，请参见 4. 从 Atmel | START 获取源代码部分。

使用单个变量跟踪时间是最常用的方法。在 32 位架构（甚至是 64 位环境）中，此方法是实现计时的最简单方法，在某种程度上也是最有效的方法。而对于 8 位架构，是否选择使用此方法取决于多个因素，以下部分将对此进行探讨。

使用单变量法跟踪时间非常简单，只需向单个 32 位变量添加一个时间单位（在本例中为 1 秒）即可。如果应用需要包含在特定时间设置闹钟的功能，则可以通过将计数器值与单个 32 位比较值进行比较来轻松实现此目的。然而，由于 8 位器件一次只能操作 8 个数据位，因此对 32 位值执行加法和比较指令需要多个时钟周期。即使 C 代码很简单，编译后的机器代码也将更加耗时耗力。

为了长期跟踪时间，需要对 32 位数字执行除法运算，以将单变量计数器转换为秒、分钟和小时。在 8 位架构上，当涉及到执行周期数和编译后代码的大小时，除法运算需要大量资源。在将从某个时间戳开始的当前秒数转换为人类可读的格式时，必须将变量除以几个不同的常量。通过计算这些常量的倒数，可以在代码中利用乘法和位移来代替除法。

除以常量的倒数

假设在某个程序执行期间，N 位变量 A 需要不通过除法运算来除以常量 C。由于$\frac{A}{C} = A \cdot \frac{1}{C}$，因此可以先计算$\frac{1}{C}$，即$C$的倒数，然后再编译代码并执行乘法运算。问题在于，为了准确地表示倒数，必须使用浮点数或定点数表示法。由于使用浮点数或定点数进行计算时需要大量资源，因此这种方法并不可取。下面的步骤列出了一种更高效的方法，此方法在执行期间仅使用整数乘法和位移。列表中的每个点都使用 $C = 3600$ 和 $N = 32$来说明。使用这些值的原因在于，*单变量时间*示例中跟踪秒数的变量是一个 32 位变量，需要除以 3600 以将秒转换为小时。 表 1-1 给出了对用于时间转换的不同常量应用算法后得出的结果值。

1. 将$\frac{1}{C}$ 转换为 Q 数字格式为 2N 的定点二进制表示，以实现更准确的定点数表示。

   示例： 1/3600 的浮点值表示为 0.00027777777777777777。将此值乘以 $2^{2N} = 2^{64}$，并忽略结果中的十进制值，得到 5,124,095,576,030,431。

   通过将该数字转换为有符号的二进制补码，可以得到 64 位数字：

   `0b0000000000001001000110100010101100111100010011010101111100110111111`

2. 左移此二进制数，直到最高有效位（Most Significant bit，MSb）为 1，并将 S 设为移位数。换句话说，从左向右计算 2N 位数中第一个 1 之前 0 的个数。

   示例： 二进制数中第一个 1 之前有 11 个 0，因此，S = 11。

3. 取左移 S 位后二进制数的 N+1 个最高有效位，并将值 1 加到此数上。通过将二进制数截断为 N 位，对该数进行舍入。

   示例： 移位后二进制数的 33 个最高有效位为：`0b100100011010001010110011110001001`。

   将数字 `0b1` 加到其中，得到 `0b1001000110100010101100111100010010`。

   截断为 32 位，得到 `0b10010001101000101011001111000101`。

4. 将 N 位二进制数转换为 N/4 位十六进制数，表示为 K。

   示例： 将 32 位二进制数转换为十六进制数，得到 K = 0x91A2B3C5。

5. 在 C 语言编程中，计算 `uintN_t R = A/C;`等同于

   ```
   uintN_t R = (((uint2N_t)A * (uint2N_t)K) >> N) >> S;
   ```

   此过程无需除法程序。

   示例： 在本示例中，我们希望将 `seconds` 除以常量 C = 3600 得到 `hours`。由于 `seconds` 是一个 32 位变量， N = 32，我们发现 S = 11 且 K = 0x91A2B3C5，因此结果代码变为：

   ```
   uint32_t hours = (((uint64)seconds * (uint64_t)0x91A2B3C5) >> 32) >> 11;
   ```

5. 附录 A——倒数乘法检查部分提供了一个测试程序，用来验证是否正确使用了该方法。测试器函数使用 C、S 和 K 值作为输入参数，并检查定点乘法的结果与对应整数除法的结果是否相同。变量 A 在整个正整数范围内递增，以测试所有可能的值。 表 1-1 列出了在*单变量日期和时间*示例中用于时间转换的有用常量所对应的 K 值和 S 值。这些值可以直接插入到下面信息框中的代码行内，无需使用除法程序即可实现将 32 位变量或 64 位变量除以其中一个常量。

表 1-1. 用于时间转换的常量所对应的 S 值和 K 值（不使用除法）

说明	C	S	K	N 位除法
一分钟的秒数	60	5	0x8889	16
一小时的秒数	3600	11	0x91A2B3C5	32
一天的小时数	24	4	0xAAAAAAAB	32

.....（续）				
说明	C	S	K	N 位除法
一年的天数	365	8	0xB38D	16

信息： `uintN_t R = A/C` 等同于

```
uintN_t R = (((uint2N_t)A * (uint2N_t)K) >> N) >> S;
```

单变量法中的闰年

尽管闰年可以很好地确保一年四季不发生偏差，但会增加日期计算的复杂性。在撰写本文档时，4 年规则的第一个例外是 2100 年。对于当今的许多应用，人们可能会认为 100 年和 400 年的例外情况无关紧要。在*单变量日期和时间*示例中，转换函数不会处理此类例外情况，而是发出编译器警告。如果代码处理 100 年和 400 年规则，必然需要额外的倒数乘法运算。在单变量法中加入闰年计算会增加一些额外的计算和逻辑，这会使程序变得复杂，导致在每次转换单个变量时都需要额外的执行周期。

单变量代码的基准测试

下表列出了*单变量时间*和*单变量日期和时间*示例的编译大小和执行时间。编译大小测量是通过仅编译必要的 Atmel START 设置（不包括任何 USART 设置）和计时所需的最基本代码实现的，这意味着其工作方式与实际示例并不完全相同。为了进行比较，表中还列出了具有相同设置，但没有计时功能且 `main()` 为空的程序的编译大小。有关如何进行大小测量的更多信息，请参见 6. 附录 B——代码大小测试详细信息部分。

表 1-2. 编译大小和执行时间

程序	GCC -O3 优化		GCC -Os 优化		周期
	程序存储器	数据存储器	程序存储器	数据存储器	
配置为空 ISR 的 RTC 和休眠程序	444 字节	0 字节	262 字节	0 字节	不适用
单个变量时间	1184 字节	11 字节	1000 字节	11 字节	589
单个变量日期和时间	1598 字节	27 字节	1372 字节	27 字节	1370~1880

通过计算在调用倒数时间转换器函数后使用的周期数，就可以得出表中*单变量时间*示例的周期测量值。相比之下，除法时间转换器函数使用了 954 个周期，并且使用了更多闪存。由此可以推断，如果执行时间至关重要，则应避免使用普通的大数除法。

*单变量日期和时间*示例中使用的周期数是通过调用倒数时间和日期转换函数测得的。日期转换中使用的周期数将取决于当前日期。这是因为代码会遍历各月份，同时减去该月的天数以找到正确的日期。此外，根据当前年份是否为闰年，进行闰年计算时的执行时间将有所不同。

1.2 将时间拆分为多个变量

在 8 位架构上，将时间拆分成多个变量具有诸多好处。与单变量法相比，使用多变量法可以减少编译的代码量并缩短执行时间。其原因在于，使用多变量法时，无需将单个变量转换为更便于人们阅读的多变量表示形式。因此，不需要对 32 位变量进行除法或乘法运算，并且闰年计算也简单了很多。

在这种方法中，RTC ISR 每 1 秒节拍将标志置 1 一次。当主函数中存在 while 无限循环时，会检查标志是否置 1。根据标志是否置 1，CPU 会进入休眠模式或调用将 `seconds` 变量递增 1 的函数。然后，将 `seconds` 变量与值 60 进行比较。当 `seconds` 变量等于 60 时，将其设置为 0，`minutes` 变量递增。对于 `hours`，方法相同。

对于月份和年份，该过程略有不同。月份和年份并不总是一样长。为解决此问题，一种简单的方法是使用月份查找表（Look-up Table，LUT），并在表中附加闰年的例外情况。这可以通过数组来实现，其中索引对应月份，索引上的值对应该月的天数。

将一秒加到 8 位值时所需的平均周期数并不高。大多数时候，只需进行一次 8 位加法和一次 8 位比较操作。仅在比较结果为真时，才需要更多周期。每当 `seconds` 变量变为 60 时，平均只需要额外执行一次 8 位加法和比较操作，依此类推。即使在最坏情况下的执行时间很长，平均执行时间仍然非常短。如果添加了闹钟功能，则需要额外进行多次比较。有关更多信息，请参见 1.3 闹钟部分。

闰年

采用这种方法时，可以轻松计算闰年。在代码中添加闰年不会占用大量闪存或 RAM。只需每年计算一次当前年份是否为闰年，并且计算本身不需要很多周期。因此，多变量示例中包含两个用于闰年计算的宏。第一个宏默认处于使能状态，执行完整的闰年计算。另一个宏已被注释掉，并且只在 2100 年之前有效。

多变量代码的基准测试

下表列出了*多变量时间*和*多变量日期和时间*示例的编译大小和执行时间。编译大小测量是通过仅编译必要的 Atmel START 设置（不包括任何 USART 设置）和计时所需的最基本代码实现的，这意味着其工作方式与实际示例并不完全相同。为了进行比较，表中还列出了具有相同设置，但没有计时功能且 `main()` 为空的程序的编译大小。有关如何进行大小测量的更多信息，请参见 6. 附录 B——代码大小测试详细信息部分。

表 1-3. 编译大小和执行时间

程序	GCC -O3 优化		GCC -Os 优化		平均执行时间
	程序存储器	数据存储器	程序存储器	数据存储器	
配置为空 ISR 的 RTC 和休眠程序	444 字节	0 字节	262 字节	0 字节	不适用
多个变量时间	600 字节	6 字节	416 字节	6 字节	约 8 个周期/秒
多个变量日期和时间	1598 字节	27 字节	1372 字节	27 字节	约 8 个周期/秒

由于上述基准测试代码中无需转换，因此执行时间是“为当前时间增加一秒”操作的平均执行时间。

1.3 闹钟

在跟踪时间以发出事件信号的应用中可能需要闹钟功能。

在多变量法中实现此功能的一种简单方法为，在嵌套 `if` 语句中，将 `seconds` 变量与 `alarm_seconds` 变量每秒进行一次比较，将 `minutes` 变量与 `alarm_minutes` 变量每秒进行一次比较，依此类推。也可能需要每秒进行多次比较，但比较只在两个 8 位值之间进行，而不是像单变量法那样进行 32 位比较。

每秒检查闹钟时，用户可能不清楚是按 `hours`、`minutes` 和 `seconds` 比较，还是按相反的顺序比较。由于 `seconds` 比较每分钟将有一次为真，因此用户可能趋向于从 `hours` 开始比较，以减少每天执行的比较次数。但是，当 `hours` 比较结果为真时，将始终每秒执行一次额外的比较，每天持续一小时。事实证明，如果先比较秒，则每天可以减少 2000 次比较。因此，平均执行时间不仅将缩短，而且还有更高的可预测性，因为它不依赖于当天的时间。第一次比较需要四个周期。因此，平均周期数将接近四个。最坏情况下的执行时间将变为 4x3 或 4x6 个周期，具体取决于闹钟是仅针对时钟还是针对时钟和日期设置。如果用户将闹钟设置为一天中的特定时间，则即使用户通过设置时钟来更改当天时间，也无需转换为秒或进行特殊考虑。

在多变量示例中，为简单起见，将每秒调用一次闹钟功能，并且仅在当天而非某个日期的时间触发该功能。如果用户仅需要在 `hours` 的值递增时将 `alarm_hours` 与 `hours` 进行一次比较，以及在递增函数内部找到匹配时将标志置 1，则可以使用更高级的解决方案。然后，用户可以对 minutes 和 seconds 执行相同的操作，但会调用不同的递增函数来大幅减少比较次数，具体取决于将哪个标志置 1。缺点是增加了闪存使用量，并且在代码中将递增时间与闹钟功能混杂在一起。

如 1.1 单变量时间部分所述，为单个变量创建闹钟似乎非常简单，因为用户只需每秒将 seconds 变量与另一个 32 位 alarm_seconds 变量进行一次比较。与 8 位比较相比，这种比较方法的缺点在于需要多个额外的周期。使用 32 位单变量法时，每次都必须考虑到整个变量，因此运行时间将保持不变。换句话说，avr-gcc 不会优化 32 位比较，因此，只要 4 个字节中有任何一个字节的比较结果为 false，比较就会停止。32 位相等性检查在 AVR 上使用 13 个周期。

但是，用户可能不知道如何设置 alarm_seconds 变量的值。一种可行的方法是，将闹钟设置为当前时间经过已知秒数后的时间，这种方法比较直接。要在某一天的特定时间设置闹钟，用户必须每秒将当前 seconds 变量的值转换为人类可读的时间，或者将人类可读的闹钟时间转换为秒数，并且每秒进行比较。这两种方案都存在一些问题。与多变量法一样，对于第一种方案，在转换完成后，可以将 seconds 与 alarm_seconds 进行比较，将 minutes 与 alarm_minutes 进行比较（依此类推）来查看闹钟是否触发。但是，根据表 1-2 得知，使用单变量法时，每秒将 seconds 变量转换为人类可读的时间所需的周期数约为使用多变量法时的周期数的 70-200 倍。因此，尽管比较两个 32 位值会带来一些问题，但将人类可读的闹钟时间转换为秒数似乎仍然是最佳方法。

通过这种方法，如果闹钟应在每天的同一时间触发，应用程序必须在闹钟每次触发时向 alarm_seconds 变量添加一天的秒数值。此外，应用程序还可以使用标志来使能或禁止闹钟。这样便可避免 32 位比较，从而来降低功耗，并加强对触发的控制。另一方面，假设用户先禁止闹钟，然后在一天后重新使能，在这种情况下如果要求应用程序记住一天中触发闹钟的时间，情况就会变得更复杂，因为在禁止的情况下代码不再每天自动更新 alarm_seconds。最后，如果用户更改了时钟值，则代码还必须更新 alarm_seconds 变量。虽然可以完整实现上述功能，但为了明确起见以及提高代码的可读性，我们在单变量示例中省略了闹钟功能。

1.4 比较两种方法

通过比较这两种方法的代码大小，会发现多变量法占用的闪存空间较小。请注意，如果代码的另一部分仍然需要 32 位乘法，则采用常数倒数乘法的单变量法可能不会占用太多闪存，因为编译器可以共享乘法子程序。

单变量法所用的周期数多于多变量法所用的周期数。因此，在递增时间单位变量（因为它是 32 位），尤其是在将该变量转换为人类可读的格式（在多变量法中无需此操作）时，单变量法的平均速度更慢。多变量法的执行时间不是恒定的，但每秒的平均周期数略微超过 8。在 8 位架构上，当考虑到加载和存储指令时，递增 32 位变量将导致周期数至少为递增 8 位变量时的两倍。单变量法的主要优势在于，递增时间的执行时间是固定的，而使用多变量法时，最坏情况下的执行时间（Worst-case Execution Time，WCET）会显著变长。多变量法的 WCET 在一天或一年中很少发生，但如果基于 WCET 调度对时限有严格要求的实时应用，很可能会发生问题。请注意，在单个变量示例中，转换为人类可读的时间操作的执行时间不固定。

通常，单变量法涉及到的内容更多（尤其是在使用倒数乘法时），并且实现起来更为复杂（考虑到日历和闰年时），而多变量法比较容易实现。当用户希望以人类可读的格式设置时钟和日期，这一差别就更加突出，因为需要加入将时间和日期转换为秒的功能。如 1.3 闹钟章节所述，如果添加闹钟，情况就会变得更复杂。单变量法的最佳用例是应用仅需要跟踪从某个事件（如启动或外部中断等）开始流逝的时间单位。这种情况下，无需在秒数和人类可读的时间之间进行转换，并且只要将计时变量与另一闹钟变量进行比较就可以实现闹钟功能。请注意，如果时间单位很小且计时间隔较短，更有效的方法是使用器件中具有适当频率和中断的定时器/计数器的计数（CNT）寄存器，具体可参考 3. 时序基本信息章节。此外，如上文所述，当涉及到实时应用时，单变量法可能更有用。

总而言之，多变量法更容易实现，通常更适用于 8 位架构。这种方法的平均速度更快，且无需使用除法，即使需要将人类可读的格式转换为秒时也是如此。主要缺点是用于递增时钟的执行时间不固定，因此不能在中断程序中执行。如果主 while 循环较大，且执行时间很长，则当前时间将不会每秒即时递增。如果主循环的执行时间长于一秒，标志就会置 1 两次，但递增时间函数只调用了一次，这样就会错过 1 秒，因此需要额外的逻辑来确保不会发生这种情况。闹钟功能很容易实现，但其执行时间同样不固定。

1.5 使用 AVR Libc 和 time.h

AVR Libc 是一个免费的 C 库，与 AVR 单片机和 GCC 兼容。它包含一个标准 C 库的子集，该库专为 Microchip AVR 8 位 RISC 单片机定制，并且包括标准 time.h 库。与标准 C 库一样，计时功能是使用单个 32 位变量实现的，此变量可跟踪从 UNIX 或 Y2K 纪元开始流逝的时间单位。可通过在每个时间单位（文档中指定为秒）调用一次 system_tick() 函数来递增变量，并且随后可将当前时间转换为多种格式，包括人类可读的时间。除了用于时间和日期的标准获取和设置函数外，该库还具有针对夏令时、时区以及基于特定地理位置的一般太阳和月球事件的功能。显而易见，该库比本应用笔记中讨论的示例更高级，很适合需要简单快速地跟踪时间的应用。但是，它在实现方面与 1.4 比较两种方法部分中讨论的单变量法具有相同的优缺点。此外，这种方法使用的是常规除法程序而非倒数乘法程序，

这可能会增加程序存储器的使用量（除非必须在应用的其他地方使用其他 32 位变量除以常量），并导致执行速度变慢。如果存储器和能效至关重要，则建议根据本文档中讨论的示例编写计时功能。

2. 硬件要求

如 1. 振荡器精度部分所述，若要实现精确计时，需要使用外部晶振而非内部 RC 定时器/计数器。ATtiny817 Xplained Pro 工具包中提供了一个板上 32.768 kHz 晶振，该晶振可用作 ATtiny817 器件上计时示例的时钟源。请注意，默认情况下晶振未连接到器件，因为器件与 USART 模块共享晶振引脚。为了按原样使用示例并实现精确计时（振荡器除外，需要校准才能确保精确度），需要对工具包进行一些修改。下文介绍了修改步骤以及如何通过 Micro-USB 使用 USART。如果只是为了测试（这种情况下计时精度不重要），可在不修改工具包的情况下运行示例，具体步骤参见 2.3 在不修改工具包的情况下运行示例。

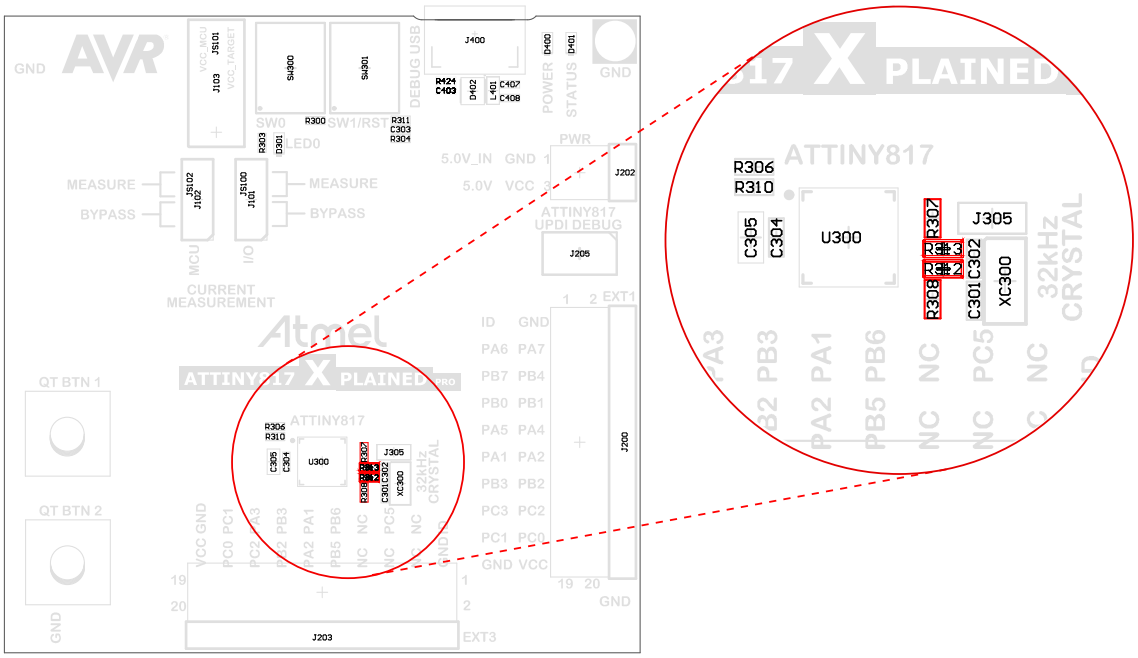
2.1 连接 32 kHz 晶振

ATtiny817 Xplained Pro 板的工具包上安装了一个 32.768 kHz 晶振。默认情况下，晶振未连接，因为器件上的 TOSC 引脚用于与扩展连接头和嵌入式调试器（EDBG）虚拟 COM 端口进行 USART 通信。要使用晶振，必须将器件上的引脚连接到晶振电路，并断开其他所有连接。

要连接晶振，请移除 0Ω 电阻 R307 和 R308，然后放在 R312 和 R313 所在的位置。请参见下面的装配图纸，以了解各个元件的具体位置。

信息： 要通过晶振操作 ATtiny817，需要使用烙铁对工具包进行物理修改。

图 2-1. 装配图纸（顶部）



2.2 通过 Micro-USB 使用 USART 连接外部晶振

在示例应用中，由于使用了晶振，因此 USART 与 USB 虚拟 COM 端口之间的连接断开。此外，它还将 USART 作为命令接口来发送命令，以设置时间或日期并接收当前信息（例如当前时间或日期）。为了在连接晶振的情况下正常执行该功能，此处介绍了一种解决方法。

ATtiny817 上的默认 USART 引脚为 PB3 和 PB2，分别用于接收（RX）数据和发送（TX）数据。如 ATtiny817 Xplained Pro 原理图所示，ATtiny817 器件上的 PB2 和 PB3 连接到电路板上的扩展连接头 PB2 和 PB3（Ext1）。引脚和扩展连接头均直接连接到 EDBG 的虚拟 COM 端口上。利用此连接，可以通过 Micro-USB 线缆发送 USART 数据。移除 0Ω 电阻后，PB2 和 PB3 引脚与 EDBG CDC 之间的连接会断开，但扩展连接头仍连接到 EDBG CDC。

ATtiny817 具有用于 USART 通信的备用引脚；PA2 用于 RX，PA1 用于 TX。这些引脚会依次连接到自己的板上扩展连接头 PA1 和 PA2（Ext3）。问题在于，这些引脚未连接到 EDBG 的虚拟 COM 端口。但是，PB2 和 PB3 的扩展连接头连接到了 EDBG 的虚拟 COM 端口，因此可以使用两条跳线电缆将备用 USART 引脚连接到默认 USART 引脚的扩展连接头。

换句话说，为了按原样运行示例，必须移动两个 0Ω 电阻以使能外部 32 kHz 晶振。要通过 Micro-USB 使用 USART 接口发送命令和接收数据以验证计时功能，请连接以下扩展连接头引脚：

- PA1（Ext3）连接到 PB2（Ext1）
- PA2（Ext3）连接到 PB3（Ext1）

例如，可使用两根母头跳线进行连接。或者，可以使用其他方式通过 PA2 和 PA3 扩展连接头直接读取和发送 RX 和 TX 数据。

2.3 在不修改工具包的情况下运行示例

可以在不对工具包进行任何修改的情况下运行示例，但是必须对代码本身进行微小改动。注意，将不使用外部 32 kHz 晶振，而是使用内部 RC 定时器/计数器作为备选时钟源。随着时间的推移，此时钟将与实时时间产生偏差，从而导致计时不准确。由于使用 Atmel START 设置了四个示例，因此可以在下载示例之前直接在 Atmel START 中进行必要的更改。另外，还可以选择在其中一个示例的主文件中取消注释函数调用。不推荐后一种方法，因为系统先执行通过 Atmel START 创建的设置函数，然后再调用该函数。换句话说，将首先执行常规设置，然后特殊函数会通过覆盖一些寄存器来重做某些设置。但是，它肯定是最快的实现方法。

选项 A——通过 Atmel START 修改示例

1. 在 Atmel START 中选择四个示例之一，然后单击 OPEN SELECTED EXAMPLE（打开所选示例）。
2. 单击 USART_0，然后找到 COMPONENT SIGNALS（元件信号）部分。在 RXD:和 TDX:旁的下拉菜单中，选择 PB3 和 PB2。
3. 单击 RTC_0，然后找到 CLOCK CONFIGURATION（时钟配置）部分。在 RTC Clock Source Selection:（RTC 时钟源选择：）旁的下拉菜单中，选择 32 kHz Internal Ultra Low Power Oscillator (OSCULP32K)（32 kHz 内部超低功耗振荡器（OSCULP32K））。
4. 单击 CLKCTRL，然后找到 32.786 kHz CRYSTAL OSCILLATOR CONFIGURATION（32.786 kHz 晶振配置）部分。取消选中以下两项旁的复选框：ENABLE: 32.768 kHz Enable Crystal Oscillator:（ENABLE：32.768 kHz 使能晶振：）和 RUNSTDBY: Run Standby:（RUNSTDBY：以待机模式运行：）。
5. 在 CLKCTRL 内部找到 32 kHz INTERNAL ULP OSCILLATOR CONFIGURATION（32 kHz 内部 ULP 振荡器配置）部分。确保选中 RUNSTDBY: Run standby:（RUNSTDBY：以待机模式运行：）旁的复选框。
6. 单击 *EXPORT PROJECT（导出项目）>DOWNLOAD PACK（下载数据包）*，然后保存.atzip 文件。
7. 通过单击 *File（文件）>Import（导入）>Atmel Start Project（Atmel Start 项目）*将.atzip 文件导入 Atmel Studio 7。

选项 B——在示例代码中取消注释函数调用

1. 打开四个示例之一。
2. 打开文件 `main.c`。
3. 取消注释对 `alternate_START_setup();` 函数的调用

3. 时序基本信息

我们为只需要定期维护的应用提供了多种选项。可以在 CPU 中使用延时函数，但这种方法也存在一些缺点。如果使用这种方法，将强制 CPU 执行延时直至完成。此外，如果在 CPU 执行延时时产生中断，则会由于延时函数未考虑到执行 ISR 的周期而导致总延时增加。通常，最好是使用定时器/计数器（TC）产生中断来实现延时和时序。这样，CPU 可以自由执行程序的其他部分，甚至可以进入休眠状态以降低功耗，同时通常还可以提高时序的精度。

定时器/计数器 A（TCA）、定时器/计数器 B（TCB）、定时器/计数器 D（TCD）和实时时钟（RTC）均可用于时序和延时。对于这些外设，要考虑的一些主要问题包括 TC 的时钟选择、时钟的预分频功能以及计数器的位数。

定时器的最大周期是时钟频率、设定的预分频比与定时器大小的乘积。随着时钟频率的增加，时间分辨率将提高，但是最大周期长度将减小。周期长度将随着计数器的大小而增加。但是，即使使用可达到最大 TC 周期的设置，仍可能无法满足特定应用的要求。在此类情况下，使用软件计数器变量是一个不错的选择。将 TC 与软件计数器结合使用时，来自 TC 的定期中断可用于递增 ISR 中的软件变量。然后，可以将软件变量与用于跟踪何时应采取其他操作的变量或常量进行比较。这样，时序长度便可超过给定定时器/计数器的最大周期。

3.1 RTC

RTC 可以通过 32.768 kHz 晶振、内部 32 kHz RC 振荡器或其他外部时钟源运行。RTC 预分频器最多可对输入频率进行 32768 分频。假定输入为 32.768 kHz，结果将为 1 Hz 节拍。计数器大小为 16 位。通过低频时钟和大型分频器，可以轻松实现长时间超时。

此外，还可以配置周期性中断定时器（Periodic Interrupt Timer，PIT）以从 RTC 的预分频器运行。使用 PIT 不会更改其余的 RTC 功能，并且 PIT 具有单独的中断向量。PIT 提供了周期性中断，其中所有可能的周期值都是 2 的幂，从 4 个、8 个、16 个一直到 32,768 个时钟周期。此功能可用于需要定期维护的应用，此时无需使用更高级的定时器/计数器来完成此任务。此外，它也是可在掉电休眠模式下运行的惟一定时器/计数器，在此模式下，可实现最低功耗并且可在产生中断时唤醒器件。

3.2 TCA

TCA 使用外设时钟（CLK_PER），其最大预分频比为 1024，计数器大小为 16 位。默认时钟设置为 3.3 MHz，最大超时周期为 2^{16}/(3.3 MHz/1024) = 23s。

除了对 CLK_PER 的时钟周期进行计数之外，还可能发生 TCA 计数事件。利用此功能，用户可以通过向某个事件输入引脚提供时钟信号的外部电路运行 TCA。此外，还可以使用来自其他定时器/计数器和外设的内部事件。

与 TCA 结合时的一个外部事件示例是使用来自 GPS 模块的一秒脉冲。通过计数此节拍，可以实现非常精确的时序。此外，也可以将 TCA 设置为计数内部事件，例如 RTC 或 TCB 溢出事件。

3.3 TCB

TCB 使用外设时钟（CLK_PER），并且其最大预分频比为 2，计数器大小为 16 位。默认时钟设置为 3.3 MHz，最大超时周期为 2^{16}/(3.3 MHz/1024) = 39.7 ms。

TCB 通常用于执行事件和信号捕捉。与 TCA 不同，TCB 无法计数事件。

3.4 TCD

TCD 可以从外设时钟（CLK_PER）或外部时钟运行，也可以直接从 20 MHz 振荡器运行。TCD 的最大预分频比为 32，计数器大小为 12 位。如果 TCD 在默认时钟设置为 3.3 MHz 时从 CLK_PER 运行，则最大超时周期为 2^{12}/(3.3 MHz/32) = 636 ms。

TCD 专为电源应用设计，例如电机控制和驱动 LED。但是，该定时器也可用于基本定时应用。与 TCB 一样，TCD 无法计数事件。

4. 从 Atmel | START 获取源代码

示例代码可通过 Atmel | START 获得，Atmel | START 是一种基于 Web 的工具，可通过图形用户界面（Graphical User Interface，GUI）配置应用程序代码。可以通过下面提供的直接示例代码链接或 Atmel | START 起始页上的 *Browse examples*（浏览示例）按钮，下载 Atmel Studio 和 IAR Embedded Workbench®对应的代码。

Atmel | START 网页： Atmel START

示例代码

- 单个变量时间示例：
  - http://start.atmel.com/#examples/single/variable/time
- 单个变量时间和日期示例：
  - http://start.atmel.com/#examples/single/variable/date/time/
- 多个变量时间示例：
  - http://start.atmel.com/#examples/multivariable/time
- 多个变量时间和日期示例：
  - http://start.atmel.com/#example/multivariable/date/time
- 多个变量时间、日期和闹钟示例：
  - http://start.atmel.com/#examples/multivariable/alarm

有关详细信息和示例项目的相关信息，请单击 Atmel | START 中的 *User guide*（用户指南）。*User guide* 按钮可以在该网页中找到，方法是在 Atmel | START 项目配置器中的仪表板视图中单击项目名称。

Atmel Studio

在 Atmel | START 的示例浏览器中单击 *Download selected example*（下载所选示例），下载 Atmel Studio 对应的代码并保存为`.atzip` 文件。要从 Atmel | START 下载文件，单击 *Export project*（导出项目），然后单击 *Download pack*（下载数据包）。

双击下载的`.atzip` 文件，将项目导入到 Atmel Studio 7.0。

IAR Embedded Workbench

有关如何在 IAR Embedded Workbench 中导入项目的信息，请打开 Atmel | START User Guide（Atmel | START 用户指南），选择 *Using Atmel Start Output in External Tools*（使用外部工具中的 Atmel Start 输出），然后选择 *IAR Embedded Workbench*。单击 Atmel | START 起始页右上角的 *Help*（帮助）或项目配置器中右上角的 *Help And Support*（帮助和支持），均可找到 Atmel | START 用户指南的链接。

5. 附录 A——倒数乘法检查

以下程序可用于检查所使用的倒数乘法值的有效性。如果常量与表 1-1 中的不同，就有必要验证计算步骤是否与 1.1 单变量时间章节中介绍的方法完全一致，也就是说需要在计算机上执行这段程序，而不是在 AVR 上。该程序包含两个可在整个整数范围（对于 32 位为 0—2^{32}，对于 16 位为 0—2^{16}）内比较倒数乘法与相应除法的函数。

```
     #include <stdint.h>
#include <stdint.h>
/* 测试 32 位除法 */
voidverify_reciprocal_mult_32(intC,intS,uint32_tK){
printf("Reciprocal multiplication verification started for C = %i.\n", C);
for(uint32_tA = 0; A < 0xFFFFFFFF; A++) {
if((((A * (uint64_t)K) >> 32) >> S) != (uint32_t)(A/(C))) {
printf("Division failed with A = %u.\n", A);
return;
}
}
printf("Test passed!\n");
}
/* 测试 16 位除法 */
voidverify_reciprocal_mult_16(intC,intS,uint16_tK)
{
printf("Reciprocal multiplication verification started for C = %i.\n", C);
for(uint16_tA = 0; A < 0xFFFF; A++) {
if((((A * (uint32_t)K) >> 16) >> S) != (uint16_t)(A/(C))) {
printf("Division failed with A = %u.\n", A);
return;
}
}
printf("Test passed!\n");
}
intmain() {
verify_reciprocal_mult_32(3600, 11, 0x91A2B3C5);//一小时秒数
verify_reciprocal_mult_32(24, 4, 0xAAAAAAAB);//一天小时数
verify_reciprocal_mult_16(60, 5, 0x8889);//一分钟秒数
verify_reciprocal_mult_16(365, 8, 0xB38D);//一年天数
return0;
}
```

该程序仅使用 C 标准库中的函数，并且是在测试时使用 GCC 进行编译的，但其编译和运行效果可媲美任何 C 编译器。

6. 附录 B——代码大小测试详细信息

单个变量示例

下表列出了*单变量时间*和*单变量日期和时间*示例的编译大小和执行时间。通过仅编译必要的 Atmel START 设置（无任何 USART 设置）和 RTC ISR（每秒将变量递增一次，并在每次调用标志时将其置 1），可确定编译大小。此外，主函数将检查标志是否已置 1。如果标志已置 1，则主函数会将标志清零，然后调用倒数转换函数，该函数可将秒数转换为人类可读的时间，并将该时间存储在时钟或时钟和日期结构体中。当没有标志时，`main()`函数会将 CPU 设为待机休眠模式。为了进行比较，下表还列出了只具有 RTC 设置且 RTC ISR 为空的程序的编译大小。在该程序中，主函数会在 ISR 完成后将 CPU 设为待机休眠模式。

表 6-1. 编译大小和执行时间

程序	GCC -O3 优化		GCC -Os 优化		周期
	程序存储器	数据存储器	程序存储器	数据存储器	
配置为空 ISR 的 RTC 和休眠程序	444 字节	0 字节	262 字节	0 字节	不适用
单个变量时间	1184 字节	11 字节	1000 字节	11 字节	589
单个变量日期和时间	1598 字节	27 字节	1372 字节	27 字节	1370~1880

多变量示例

下表列出了*多变量时间*和*多变量日期和时间*示例的编译大小和执行时间。通过仅编译必要的 Atmel START 设置（无任何 USART 设置）和 RTC ISR（每秒将标志置 1 一次），可确定编译大小。此外，`main()`函数会将器件设为待机休眠模式，并检查标志是否已置 1。如果标志已置 1，则 `main()`函数会将标志清零、调用递增时间函数并执行必要的比较，同时根据需要递增其他时间相关变量。这些变量将存储在时钟和日期结构体中。

表 6-2. 编译大小和执行时间

程序	GCC -O3 优化		GCC -Os 优化		平均执行时间
	程序存储器	数据存储器	程序存储器	数据存储器	
配置为空 ISR 的 RTC 和休眠程序	444 字节	0 字节	262 字节	0 字节	不适用
多个变量时间	600 字节	6 字节	416 字节	6 字节	约 8 个周期/秒
多个变量日期和时间	1598 字节	27 字节	1372 字节	27 字节	约 8 个周期/秒

7. 版本历史

版本	日期	说明
A	2019 年 12 月	文档初始版本

Microchip 网站

Microchip 网站（www.microchip.com/）为客户提供在线支持。客户可通过该网站方便地获取文件和信息。我们的网站提供以下内容：

- **产品支持**——数据手册和勘误表、应用笔记和示例程序、设计资源、用户指南以及硬件支持文档、最新的软件版本以及归档软件
- **一般技术支持**——常见问题解答（FAQ）、技术支持请求、在线讨论组以及 Microchip 设计伙伴计划成员名单
- **Microchip 业务**——产品选型和订购指南、最新 Microchip 新闻稿、研讨会和活动安排表、Microchip 销售办事处、代理商以及工厂代表列表

产品变更通知服务

Microchip 的产品变更通知服务有助于客户了解 Microchip 产品的最新信息。注册客户可在他们感兴趣的某个产品系列或开发工具发生变更、更新、发布新版本或勘误表时，收到电子邮件通知。

欲注册，请访问 www.microchip.com/pcn，然后按照注册说明进行操作。

客户支持

Microchip 产品的用户可通过以下渠道获得帮助：

- 代理商或代表
- 当地销售办事处
- 应用工程师（ESE）
- 技术支持

客户应联系其代理商、代表或 ESE 寻求支持。当地销售办事处也可为客户提供帮助。本文档后附有销售办事处的联系方式。

也可通过 www.microchip.com/support 获得网上技术支持。

Microchip 器件代码保护功能

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术指标。
- Microchip 确信：在正常使用的情况下，Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿意与关心代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。

代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了《数字器件千年版权法案（Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

法律声明

提供本文档的中文版本仅为了便于理解。请勿忽视文档中包含的英文部分，因为其中提供了有关 Microchip 产品性能和使用情况的有用信息。Microchip Technology Inc.及其分公司和相关公司、各级主管与员工及事务代理机构对译文中可能存在的任何差错不承担任何责任。建议参考 Microchip Technology Inc.的英文原版文档。

本出版物中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。Microchip 对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担

保，包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。Microchip 对因这些信息及使用这些信息而引起的后果不承担任何责任。如果将 Microchip 器件用于生命维持和/或生命安全应用，一切风险由买方自负。买方同意在由此引发任何一切伤害、索赔、诉讼或费用时，会维护和保障 Microchip 免于承担法律责任，并加以赔偿。除非另外声明，否则在 Microchip 知识产权保护下，不得暗中或以其他方式转让任何许可证。

商标

Microchip 的名称和徽标组合、Microchip 徽标、Adaptec、AnyRate、AVR、AVR 徽标、AVR Freaks、BesTime、BitCloud、chipKIT、chipKIT 徽标、CryptoMemory、CryptoRF、dsPIC、FlashFlex、flexPWR、HELDO、IGLOO、JukeBlox、KeeLoq、Kleer、LANCheck、LinkMD、maXStylus、maXTouch、MediaLB、megaAVR、Microsemi、Microsemi 徽标、MOST、MOST 徽标、MPLAB、OptoLyzer、PackeTime、PIC、picoPower、PICSTART、PIC32 徽标、PolarFire、Prochip Designer、QTouch、SAM-BA、SenGenuity、SpyNIC、SST、SST 徽标、SuperFlash、Symmetricom、SyncServer、Tachyon、TempTrackr、TimeSource、tinyAVR、UNI/O、Vectron 及 XMEGA 均为 Microchip Technology Incorporated 在美国和其他国家或地区的注册商标。

APT、ClockWorks、The Embedded Control Solutions Company、EtherSynch、FlashTec、Hyper Speed Control、HyperLight Load、IntelliMOS、Libero、motorBench、mTouch、Powermite 3、Precision Edge、ProASIC、ProASIC Plus、ProASIC Plus 徽标、Quiet-Wire、SmartFusion、SyncWorld、Temux、TimeCesium、TimeHub、TimePictra、TimeProvider、Vite、WinPath 和 ZL 均为 Microchip Technology Incorporated 在美国的注册商标。

Adjacent Key Suppression、AKS、Analog-for-the-Digital Age、Any Capacitor、AnyIn、AnyOut、BlueSky、BodyCom、CodeGuard、CryptoAuthentication、CryptoAutomotive、CryptoCompanion、CryptoController、dsPICDEM、dsPICDEM.net、Dynamic Average Matching、DAM、ECAN、EtherGREEN、In-Circuit Serial Programming、ICSP、INICnet、Inter-Chip Connectivity、JitterBlocker、KleerNet、KleerNet 徽标、memBrain、Mindi、MiWi、MPASM、MPF、MPLAB Certified 徽标、MPLIB、MPLINK、MultiTRAK、NetDetach、Omniscient Code Generation、PICDEM、PICDEM.net、PICkit、PICtail、PowerSmart、PureSilicon、QMatrix、REAL ICE、Ripple Blocker、SAM-ICE、Serial Quad I/O、SMART-I.S.、SQI、SuperSwitcher、SuperSwitcher II、Total Endurance、TSHARC、USBCheck、VariSense、ViewSpan、WiperLock、Wireless DNA 和 ZENA 均为 Microchip Technology Incorporated 在美国和其他国家或地区的商标。

SQTP 为 Microchip Technology Incorporated 在美国的服务标记。

Adaptec 徽标、Frequency on Demand、Silicon Storage Technology 和 Symmcom 均为 Microchip Technology Inc.在除美国外的国家或地区的注册商标。

GestIC 为 Microchip Technology Inc.的子公司 Microchip Technology Germany II GmbH & Co. KG 在除美国外的国家或地区的注册商标。

在此提及的所有其他商标均为各持有公司所有。

© 2020，Microchip Technology Incorporated 版权所有。

ISBN：978-1-5224-7284-1

质量管理体系

有关 Microchip 的质量管理体系的信息，请访问 www.microchip.com/quality。

全球销售及服务网点

美洲	亚太地区	亚太地区	欧洲
公司总部 2355 West Chandler Blvd. Chandler, AZ 85224-6199 电话：480-792-7200 传真：480-792-7277 技术支持： www.microchip.com/support 网址： www.microchip.com **亚特兰大** 德卢斯，佐治亚州 电话：678-957-9614 传真：678-957-1455 **奥斯汀，德克萨斯州** 电话：512-257-3370 **波士顿** 韦斯特伯鲁，马萨诸塞州 电话：774-760-0087 传真：774-760-0088 **芝加哥** 艾塔斯卡，伊利诺伊州 电话：630-285-0071 传真：630-285-0075 **达拉斯** 阿迪森，德克萨斯州 电话：972-818-7423 传真：972-818-2924 **底特律** 诺维，密歇根州 电话：248-848-4000 **休斯顿，德克萨斯州** 电话：281-894-5983 **印第安纳波利斯** 诺布尔斯维尔，印第安纳州 电话：317-773-8323 传真：317-773-5453 电话：317-536-2380 **洛杉矶** 米慎维荷，加利福尼亚州 电话：949-462-9523 传真：949-462-9608 电话：951-273-7800 **罗利，北卡罗来纳州** 电话：919-844-7510 **纽约，纽约州** 电话：631-435-6000 **圣何塞，加利福尼亚州** 电话：408-735-9110 电话：408-436-4270 **加拿大 - 多伦多** 电话：905-695-1980 传真：905-695-2078	**澳大利亚 - 悉尼** 电话：61-2-9868-6733 **中国 - 北京** 电话：86-10-8569-7000 **中国 - 成都** 电话：86-28-8665-5511 **中国 - 重庆** 电话：86-23-8980-9588 **中国 - 东莞** 电话：86-769-8702-9880 **中国 - 广州** 电话：86-20-8755-8029 **中国 - 杭州** 电话：86-571-8792-8115 **中国 - 香港特别行政区** 电话：852-2943-5100 **中国 - 南京** 电话：86-25-8473-2460 **中国 - 青岛** 电话：86-532-8502-7355 **中国 - 上海** 电话：86-21-3326-8000 **中国 - 沈阳** 电话：86-24-2334-2829 **中国 - 深圳** 电话：86-755-8864-2200 **中国 - 苏州** 电话：86-186-6233-1526 **中国 - 武汉** 电话：86-27-5980-5300 **中国 - 西安** 电话：86-29-8833-7252 **中国 - 厦门** 电话：86-592-2388138 **中国 - 珠海** 电话：86-756-3210040	**印度 - 班加罗尔** 电话：91-80-3090-4444 **印度 - 新德里** 电话：91-11-4160-8631 **印度 - 浦那** 电话：91-20-4121-0141 **日本 - 大阪** 电话：81-6-6152-7160 **日本 - 东京** 电话：81-3-6880-3770 **韩国 - 大邱** 电话：82-53-744-4301 **韩国 - 首尔** 电话：82-2-554-7200 **马来西亚 - 吉隆坡** 电话：60-3-7651-7906 **马来西亚 - 槟榔屿** 电话：60-4-227-8870 **菲律宾 - 马尼拉** 电话：63-2-634-9065 **新加坡** 电话：65-6334-8870 **台湾地区 - 新竹** 电话：886-3-577-8366 **台湾地区 - 高雄** 电话：886-7-213-7830 **台湾地区 - 台北** 电话：886-2-2508-8600 **泰国 - 曼谷** 电话：66-2-694-1351 **越南 - 胡志明市** 电话：84-28-5448-2100	**奥地利 - 韦尔斯** 电话：43-7242-2244-39 传真：43-7242-2244-393 **丹麦 - 哥本哈根** 电话：45-4485-5910 传真：45-4485-2829 **芬兰 - 埃斯波** 电话：358-9-4520-820 **法国 - 巴黎** 电话：33-1-69-53-63-20 传真：33-1-69-30-90-79 **德国 - 加兴** 电话：49-8931-9700 **德国 - 哈恩** 电话：49-2129-3766400 **德国 - 海尔布隆** 电话：49-7131-72400 **德国 - 卡尔斯鲁厄** 电话：49-721-625370 **德国 - 慕尼黑** 电话：49-89-627-144-0 传真：49-89-627-144-44 **德国 - 罗森海姆** 电话：49-8031-354-560 **以色列 - 若那那市** 电话：972-9-744-7705 **意大利 - 米兰** 电话：39-0331-742611 传真：39-0331-466781 **意大利 - 帕多瓦** 电话：39-049-7625286 **荷兰 - 德卢内市** 电话：31-416-690399 传真：31-416-690340 **挪威 - 特隆赫姆** 电话：47-72884388 **波兰 - 华沙** 电话：48-22-3325737 **罗马尼亚 - 布加勒斯特** 电话：40-21-407-87-50 **西班牙 - 马德里** 电话：34-91-708-08-90 传真：34-91-708-08-91 **瑞典 - 哥德堡** 电话：46-31-704-60-40 **瑞典 - 斯德哥尔摩** 电话：46-8-5090-4654 **英国 - 沃金厄姆** 电话：44-118-921-5800 传真：44-118-921-5820