32-Bit 定时器基本介绍 Timer32 V 2.6

Copyright © 2012 Cypress Semiconductor Corporation. All Rights Reserved.

资源	PSoC® 模块			API 存储器 （字节）		引脚 （每个外部 I/O）
	数字	模拟 CT	模拟 SC	Flash （闪存）	RAM	
CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8CTMA140, CY8C21x45, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx						
32-bit	4	0	0	154	0	1

如需一个或多个使用此用户模块且完全配置的功能性示范项目，请转到 www.cypress.com/psocexampleprojects.

功能和概述

- 32-bit 通用定时器使用四个 PSoC 块
- 源时钟频率高达 48 MHz
- 计数结束后自动重新加载计数周期
- 时钟捕获速率高达 24 MHz
- 终止计数输出脉冲可用作其他模拟和数字功能的输入时钟
- 中断触发选项包括：终止计数、捕获 （部分器件中可用）或当计数器达到某个预先设定的值

32-bit 定时器用户模块提供了具有周期可编程并带捕获功能的递减计数器。 可从任何系统基准时钟或外部源选择时钟和使能信号。 一旦启动，定时器便持续运行，每当计数结束时，再重新从周期寄存器加载其内部值开始计数。 在终止计数之后的时钟周期中，输出将为高电平。 事件能够获得当前定时器计数值，通过捕获输入信号的沿。 在每个时钟周期中，定时器都会将计数与比较寄存器的值进行对比测试，测试两数为 “ 小于 ”(Less Than) 还是 “ 小于或等于 ”(Less Than or Equal To) 关系。 可以基于计数结束或比较信号生成中断。 一些器件系列提供了两个额外的功能。 即中断选项包括 “ 捕获中断方式 ”(interrupt on capture)，以及可将比较信号路由到行总线上。 如果您选择的器件上提供了这些选项，则它们会显示在器件编辑器中。

Figure 1. 定时器框图 （对于大多数 PSoC 器件），数据路径宽度 n = 32

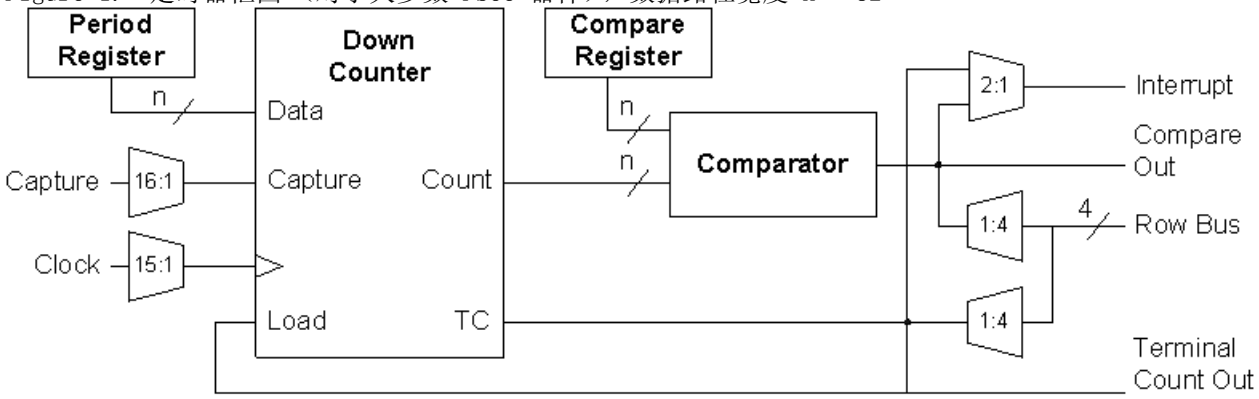
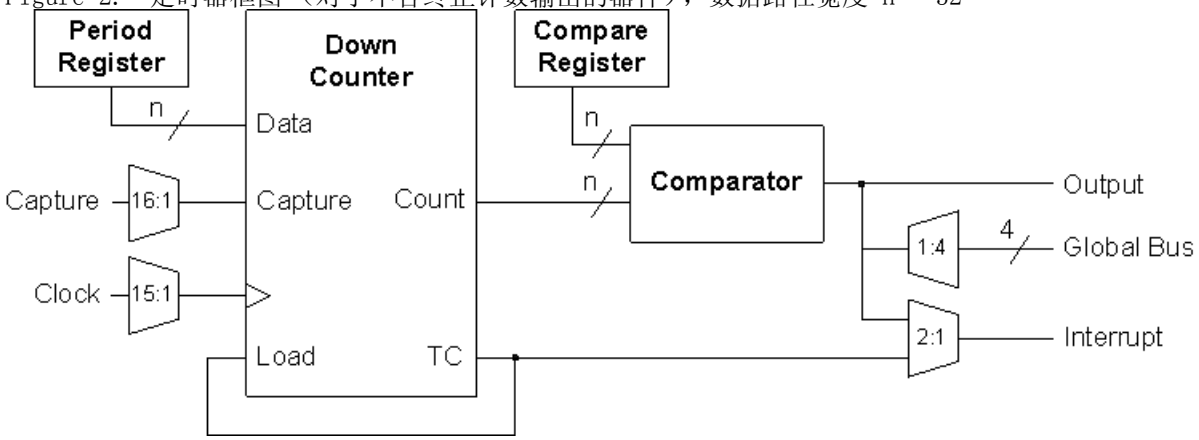


Figure 2. 定时器框图 （对于不含终止计数输出的器件），数据路径宽度 n = 32



功能描述

Timer32 用户模块利用了四个数字 PSoC 模块，每个模块提供 8 位分辨率。 连续模块彼此关联，因此能够同时连接内部进位位、终止计数和比较信号。 这样可使模块间的 8-bit 计数、周期和比较寄存器 （分别对应数据寄存器 DR0、DR1 和 DR2）相互连结，以提供所需的分辨率。 这样，Timer32 可作为单片同步定时器运行。

定时器 API 提供了可用 C 语言和汇编语言调用的多种函数，以便停止或启动定时器操作以及读写各种数据寄存器。 提供了一个控制寄存器，用于启动和停止定时器用户模块。 当定时器停止时，对周期寄存器的写入操作会导致将周期寄存器的值复制到计数寄存器中。 当定时器停止时，输出被置于低电平。

定时器启动时，计数寄存器会在每个时钟上升沿出现时递减 1。 零计数之后，再次出现时钟上升沿时，计数寄存器将从周期寄存器重新加载值。 在下一个下降沿出现时，将触发终止计数事件，并将输出置为高电平，持续半个时钟周期，或者， CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx 在器件系列中可选择持续整个时钟周期。 通过这种方法，定时器可充当时钟分频器。 其周期和频率与源时钟的周期和频率相关，系数等于定时器周期寄存器的值加 1。

Equation 1

$$OutputPeriod = SourceClockPeriod \times (PeriodRegisterValue + 1)$$

周期值为 0 时，将输出移位半个时钟周期的输入源时钟，从而产生一分频的时钟。 在 CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx 器件系列中，终止计数脉冲宽度必须设置为半个周期。

终止计数输出的工作周期如下所示。

Equation 2

$$DutyCycle = \frac{0.5}{PeriodValue + 1}$$

此外，当终止计数脉冲宽度设置为整个周期时，工作周期将延长为两倍。

Equation 3

$$DutyCycle = \frac{1}{PeriodValue + 1}$$

周期寄存器值参数可以使用器件编辑器进行指定。 此外，也可以在运行时使用 API 对其进行修改。 在计数寄存器的值达到 0（终止计数）之后的周期中，周期寄存器将自动复制到计数寄存器中。 因此，如果通过 API 更改周期，新值将不会立刻生效。 要在运行时做出立即生效的更改，正确的步骤是：停止定时器，写入新周期值，再重新启动定时器。

在每个输入时钟上，将计数寄存器中的计数与存储在比较寄存器中的值进行比较。 根据器件编辑器中为 CompareType 参数所指定的选项，将执行 “ 小于 ”(Less Than) 或 “ 小于或等于 ”(Less Than or Equal To) 对比测试。 当满足比较条件时，将在下一时钟上触发比较事件。

在 CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx 在器件系列中，定时器用户模块提供了比较输出信号作为辅助输出。 在满足比较条件周期之后的时钟周期中，出现上升沿时，将置位此高电平有效信号。 辅助输出不能直接连接到相邻的数字 PSoC 模块；但是，它能连接到其他数字 PSoC 模块，或者通过本地行输出总线连接到 GPIO 引脚。

当捕获输入被置于高电平时，系统时钟也将同步跃变，且计数寄存器中的值将传输到比较寄存器中。 在 CY8C29/27/24/22/21xxx 及 CY8CLED04/08/16 系列中，如果中断类型设置为 “ 捕获 ”(capture)，则会在捕获事件后发生中断。 然后，可使用 ReadTimer API 函数读取计数值。 如果满足以下条件，则会在出现 “ 比较真值 ”(compare true) 事件时发生中断。

1. 在 CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx 系列中，中断类型必须设置为 “ 捕获 ”(capture) 触发方式。
2. 必须启用定时器中断
3. 必须启用全局中断

按以下方式计算已用时间。

Equation 4

$$ElapsedTime = ClockPeriod \times (PeriodValue - CounterValue)$$

触发中断的方式可为终止计数触发或比较事件触发， CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx 在 PSoC 器件的系列中，基于捕获信号本身触发。 在 CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx 系列中，在使用外部捕获信号执行事件时序时，将中断设置为捕获事件触发。 在执行已用时间测量时，将中断设置为终止计数触发。

对于需要在不影响计数或比较寄存器的情况下读取传输中的定时器递减值的算法，可调用 ReadTimerSaveCV() API。 此函数可在保留比较寄存器内容的同时，读取计数寄存器的值。 此函数有可能产生延迟的副作用，在本用户模块的 API 章节中有所说明。

捕获机制允许在某操作发生前，使用最大时间临界值对外部事件进行定时。 执行步骤如下。

1. 将周期寄存器设置为等于该最大值的周期值。
2. 将比较寄存器设置为最大时间限制计数，此计数的计算方式如下。

Equation 5

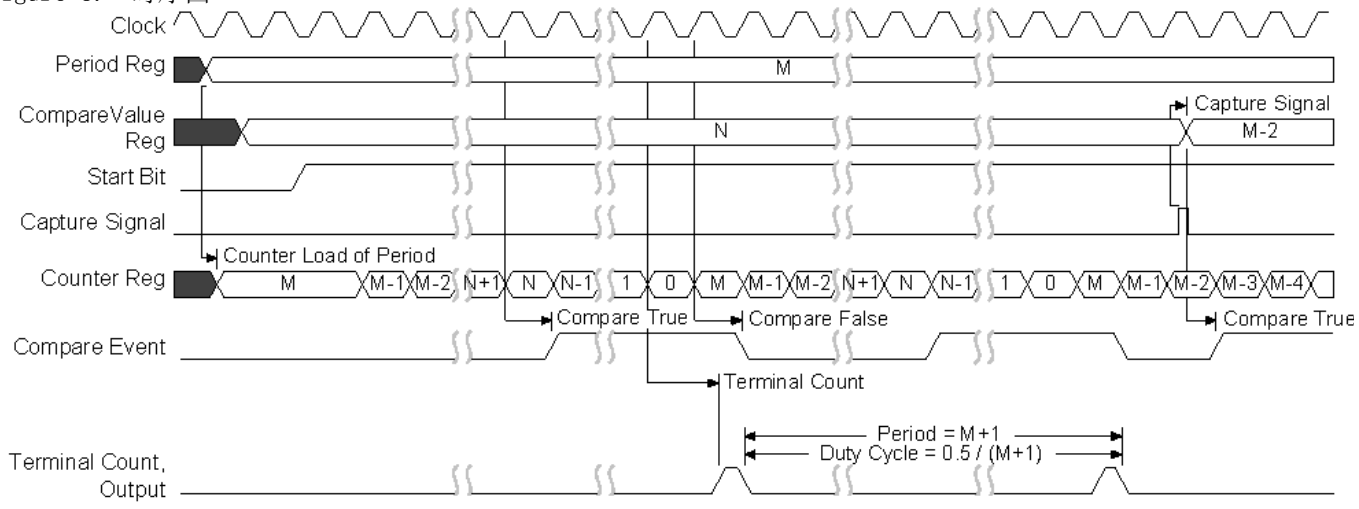
$$MaxTimeLimitCount = PeriodValue - \frac{MaxTimeLimit}{ClockPeriod}$$

3. 在 CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx 系列中，中断类型必须设置为 “ 捕获 ”(capture) 触发方式。
4. 在适当的时候启动定时器。
5. 中断触发时，读取比较寄存器。

时序

通过 PSoC 器件的全局总线功能将外部引脚路由至计数器，这些外部引脚即可为定时器计时。 下图说明了定时器用户模块的时序。

Figure 3. 时序图



直流和交流电气特性

Table 1. 定时器交流电气特性

参数	典型值	限制	单位	条件和注释
最大输入频率	--	24[a]	MHz	24 或 32-bit 宽
最大输出频率	--	24[a]	MHz	Vdd=5.0V 及 48 MHz 输入时钟
	--	12[b]	MHz	Vdd=3.3V 及 24 MHz 输入时钟

a. 如果输入或输出通过全局总线路由，则频率限制为不超过 12 MHz。

b. PSoC 模块在 3.3V 电压下运行时，可用的最快时钟频率为 24 MHz。

放置

Timer32 使用四个数字 PSoC 块。 器件编辑器将这四个模块连续放置，并按模块数递增从最低有效位 (LSB) 到最高有效位 (MSB) 进行排序。 每个模块都有一个给定的符号名，器件编辑器会在放置模块的过程中以及放置之后显示该名称。 API 使用用户指定的实例名称和模块名称来分配所有寄存器名称，以便通过 API 包含文件直接访问定时器寄存器。 下表中提供了由各种宽度使用的模块名称。

Table 2. PSoC 模块符号名称

PSoC 模块	32-Bit 定时器
1	TIMER32_LSB
2	TIMER32_ISB1
3	TIMER32_ISB2
4	TIMER32_MSB

参数和资源

使用器件编辑器选择或放置定时器用户模块后，就可以选择或更改下列参数的值。

时钟

从一个可用源中选择 “ 时钟 ”(Clock) 参数。 这些源包括 48 MHz 振荡器（仅适用于 5.0V 运行）、24V1、24V2、其他 PSoC 模块以及通过全局输入和输出路由的外部输入。 当模块使用外部数字时钟时，应将行输入同步关闭，以获得最佳精度以及进行睡眠操作。

捕获

从一个可用源中选择此参数。 此输入的上升沿将导致计数寄存器的值传输到比较寄存器中。 如果将此参数设置为 1 或外部保持高电平，则软件捕获机制将无法正常运行。

Output

可以禁用该输出参数，或者将其路由到四个全局输出信号之一。 此参数仅适用于 PSoC 器件的 CY8C26/25xxx 系列。

TerminalCountOut

终止计数输出是辅助计数器输出。 通过此参数可以禁用计数器输出，或将该输出连接到任意行输出总线。 此参数仅在 PSoC 器件的 CY8C29/27/24/22/21xxx 和 CY8CLED04/08/16 系列成员中显示。

CompareOut

比较输出可以设置为禁用（在不干扰中断操作的情况下），或将其连接到任意行输出总线。 无论设置如何，此参数均可作为下一个更高的数字 PSoC 模块以及模拟列时钟选择复用器的输入使用。 此参数仅在 PSoC 器件的 CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx 系列成员中显示。

Period（周期）

此参数设置定时器的周期。 容许值介于 0 到 2^{32}-1 之间。 此值将加载到周期寄存器中。 当计数器达到 0 或定时器从禁用状态切换到使能状态时，将自动重新加载周期。 可使用 API 修改此值。

比较值 (CompareValue)

此参数设置比较事件触发时定时器周期的计数点。 此值将加载到比较寄存器中。 容许值介于 0 到周期值之间。 可使用 API 修改此值。

CompareType

此参数可将比较函数类型设置为 “ 小于 ”(less than) 或 “ 小于或等于 ”(less than or equal)，如之前功能说明中所述。

InterruptType

此参数指定基于终止计数事件或比较事件触发中断。 使用 API 启用中断。

ClockSync

在 PSoC 器件中，数字模块可以在系统时钟以外提供时钟源。 数字时钟源甚至可以用串行方式串联起来。 这样就会引起与系统时钟相关的时滞。 这些时滞对于 CY8C29/27/24/22/21xxx 及 CY8CLED04/08/16 PSoC 器件系列尤为重要，因为它们对数据路径（尤其是系统总线）进行了各种优化。 此参数可用于控制时钟时滞并确保其在读取和写入 PSoC 模块寄存器值时的正确运行。 此参数的正确数值应当由下表决定。

ClockSync 值	使用说明
Sync to SysClk（同步到系统时钟）	此设置值适用于任何由 24 MHz (SysClk) 经过二分频或更多分频所衍生出来的时钟源。 示例包括 VC1、VC2、VC3（在 VC3 由 SysClk 驱动时）、32KHz 和采用基于 SysClk 时钟源的数字 PSoC 模块。 外部生成的时钟源也应使用此值来确保执行正确的同步操作。
Sync to SysClk*2	除非生成的频率为 48 MHz（换句话说，在所有分频器的乘积为 1 时），此设置值可以适用于任何基于 48 MHz (SysClk*2) 的时钟。
Use SysClk Direct	在需要 24 MHz (SysClk/1) 时钟时使用。 此选项并不真正执行同步，但提供了对系统时钟本身的低时滞访问方式。 如果选择此项，则此选项将覆盖上述时钟参数的设置。 在所有分频器组合起来最终生成了 24 MHz 的输出时，一定要使用此项，而不使用 VC1、VC2、VC3 或数字模块。
Unsynchronized	在选定 48 MHz (SysClk*2) 输入时使用。 在需要未同步输入时使用。 一般来说，只有在中断生成是计数器的唯一应用时才推荐使用此选项。 在睡眠时仍然保持活动状态的模块需要此设置。

TC_PulseWidth

使用此参数，可指定终止计数输出脉冲为一个时钟周期宽还是半个时钟周期宽。

InvertCapture

此参数确定使能输入信号的意义。 当选中 “ 正常 ”(Normal) 时，使能输入为高电平有效。 选择 “ 反相 ”(Invert) 则解释为低电平有效。 InvertCapture 仅适用于 PSoC 器件的 CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx 系列成员。

中断生成控制

当选中 PSoC Designer 中的 “ 启用中断生成控制 ” 复选框时，有两个附加参数变为可用。 **启用中断生成控制** 复选框时，会有一个附加参数变为可用。 可以在以下菜单下找到此复选框： **项目 > 设置 > 芯片编辑器** . 当外覆层的多个用户模块所共享的中断用于多个外覆层时，中断生成控制非常重要：

- 中断 API
- IntDispatchMode

InterruptAPI

InterruptAPI 参数允许有条件生成用户模块的中断处理程序和中断矢量表条目。 选择 “ 启用 ”(Enable) 可生成中断处理程序和中断向量入口。 选择 “ 禁用 ”(Disable) 可避免生成中断处理程序和中断向量入口。 在那些拥有多个外覆层而且有多个外覆层使用同一模块资源的项目中，特别推荐要正确地选择是否要生成中断 API。 仅在必要时选择生成中断 API，这样可以避免生成中断调度代码，从而减少开销。

IntDispatchMode

IntDispatchMode 参数用于指定中断请求的处理方式，这些中断由同一模块不同外覆层中的多个用户模块共享。 选择 “ActiveStatus” 会导致固件在为共享的中断请求提供服务之前测试哪一个外覆层正处于活动状态。 每次请求共享中断时，都会进行此测试。 这会增加延迟，还会产生为共享中断请求提供服务的不确定过程，但是不需要任何 RAM。 选择 “OffsetPreCalc” 参数会导致固件仅在最初加载重叠层时计算共享中断请求的来源。 这种计算可减少中断延迟，并产生为共享中断请求提供服务的确定过程，但会占用一个字节的 RAM 空间。

应用程序编程接口

提供的应用程序编程接口 (API) 子程序作为用户模块的一部分，允许设计人员能够采用更高级的方式处理模块。 本节指定每个函数的接口，以及 “ 引用 ” 文件所提供的相关常量。

Note

在这里，如同所有用户模块 API 中的一样，A 和 X 寄存器的值可能通过调用 API 函数发生更改。 如果在调用后需要 A 和 X 的值，则调用函数负责在调用前保留 A 和 X 的值。 选择此 “ 寄存器易失 ” 策略是为了提高效率，自 PSoC Designer 1.0 版起已强制使用此策略。 C 编译器自动遵循此要求。 汇编语言程序员也必须确保其代码遵守这一策略。 虽然一些用户模块 API 函数可以保留 A 和 X 不变，但是无法保证它们将来也会如此。

对于大型储存器模块驱动， 保存 CUR_PP、IDX_PP、MVR_PP 以及 MVW_PP 寄存器中的所有值也是调用程序的职责。 尽管部分寄存器现在可能不可修改，但是无法保证在将来的版本中也会如此。

应用程序编程接口 (API) 子程序作为用户模块的一部分提供，使设计人员能够在较高的层级处理模块。 以下是为 Timer32 提供的 API 编程子程序。

Timer32_PERIOD

说明：

表示器件编辑器中为 Timer32 的 “ 周期 ” 字段选择的值。 该值的范围介于 0 到 4294967295 之间。

Timer24_COMPARE_VALUE

说明：

表示器件编辑器中为 Timer32 的 “ 脉冲宽度 ” 字段选择的值。 该值的范围介于 0 到 4294967295 之间。

Timer32_EnableInt

说明：

启用中断模式运行。 但是请注意，在实际处理中断之前，还必须启用全局中断。

C 原型：

```
void  Timer32_EnableInt(void);
```

汇编：

```
lcall  Timer32_EnableInt
```

参数：

None

返回值：

None

副作用：

此子程序修改 IO 空间中相应的中断使能寄存器。 A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在扩展内存模式中，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

Timer32_DisableInt

说明：

禁用中断模式运行。

C 原型：

```
void  Timer32_DisableInt(void);
```

汇编：

```
lcall  Timer32_DisableInt
```

参数：

None

返回值：

None

副作用：

此子程序修改 IO 空间中相应的中断使能寄存器。 A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在扩展内存模式中，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

Timer32_Start

说明：

开始运行 Timer32。 计数寄存器将在下一个时钟周期上减少。

C 原型：

```
void  Timer32_Start(void);
```

汇编：

```
lcall  Timer32_Start
```

参数：

None

返回值：

None

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在扩展内存模式中，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

Timer32_Stop

说明：

停止运行 Timer32。

C 原型：

```
void  Timer32_Stop(void);
```

汇编：

```
lcall  Timer32_Stop
```

参数：

None

返回值：

None

副作用：

输出将被置为低电平，且后续对周期寄存器的写入操作将会使计数寄存器更新为新的周期值。 A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在扩展内存模式中，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

Timer32_WritePeriod

说明：

将周期值写入周期寄存器。 在达到零计数条件时将周期载入到计数寄存器中，或者当 Timer32 处于停止状态时立即执行载入。

C 原型：

```
void  Timer32_WritePeriod(DWORD dwPeriod);
```

汇编：

```
mov A,[dwPeriod]
push A
mov A,[dwPeriod+1]
push A
mov A,[dwPeriod+2]
push A
mov A,[dwPeriod+3]
push A
lcall _Timer32_WritePeriod
```

参数：

dwPeriod： 该值介于 0 到 2^{32}-1 之间。

返回值：

None

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在扩展内存模式中，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

Timer32_WriteCompareValue

说明：

修改定时器的比较寄存器的值。 为避免产生意外的副作用，应当禁用定时器 （未通过 Start API 函数启用或者首先调用 Stop API 函数）。

C 原型：

```
void  Timer32_WriteCompareValue(DWORD dwCompareValue);
```

汇编：

```
mov A,[dwCompareValue]
push A
mov A,[dwCompareValue+1]
push A
mov A,[dwCompareValue+2]
push A
mov A,[dwCompareValue+3]
push A
lcall _Timer32_WriteCompareValue
```

参数：

dwCompareValue： 值介于 0 到周期值之间。

返回值：

None

副作用：

当定时器正在运行并且比较值等于或大于计数寄存器的当前值时，如果调用此函数，将可能发生比较事件。 当比较寄存器分布在多个 PSoC 模块中并且一次被写入一个字节，比较寄存器的值可能会发生意外变化。 写入字节的顺序未指定且可能会随时更改。 如果中断类型均设置为比较事件触发方式且定时器中断已启用，这可能会导致中断。 A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在扩展内存模式中，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

Timer32_ReadCompareValue

说明：

使用 “pass-by-reference” 参数读取 Timer32 Compare 寄存器。

C 原型：

```
void  Timer32_ReadCompareValue(DWORD * pdwCompareValue);
```

汇编：

```
mov   A,[pdwCompareValue]
mov   X,[pdwCompareValue+1]
lcall _Timer32_ReadCompareValue
```

参数：

pdwCompareValue： 指向缓冲器的指针，其中保留着 Compare 寄存器数据。 X 寄存器中装载 ram 地址，其中要保存返回值。

返回值：

在特定缓冲器中返回 Compare 寄存器的值。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在扩展内存模式中，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 IDX_PP 页面指针寄存器。

Timer32_ReadTimerSaveCV

说明：

读取当前 Timer32 计数寄存器的值，并保留比较寄存器的值。 这将执行软件请求、硬件同步的计数器捕获操作。 应当仅在必须保留比较寄存器内容的情况下，才使用此函数。 如果不需要保留比较寄存器内容，则首选使用 ReadTimer() 函数。 请注意，此 API 子程序过去被称为 ReadCounter。

C 原型：

```
void  Timer32_ReadTimerSaveCV(DWORD * pdwCount);
```

汇编：

```
mov   A,[pdwCount]
mov   X,[pdwCount+1]
lcall _Timer32_ReadTimerSaveCV
```

参数：

None

返回值：

pdwCount： 计数寄存器的内容。 X 寄存器中装载返回缓冲器的地址。

副作用：

要读取计数寄存器的值，必须在其值返回之前暂时将其传输到比较寄存器中。 此操作将导致比较条件立即变为真或在下一个定时器输入时钟周期上变为真，具体取决于 CompareType 参数的设置为 “ 小于或等于 ”(Less than or Equal to) 或 “ 小于 ”(Less Than)。 如果 （或当）用户模块和全局中断已启用，则很可能在此 API 函数返回调用程序之前，甚至在其将比较寄存器恢复到原先状态之前，中断将得到处理。 中断将暂时禁用。 最后，为了恢复比较寄存器，用户模块本身将被暂时禁用。 这将导致计数寄存器丢失一个或多个计数。 A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在扩展内存模式中，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 IDX_PP 页面指针寄存器。

Timer32_ReadTimer

说明：

读取当前 Timer32 计数寄存器的值。 这将执行软件请求、硬件同步的计数器捕获操作。 如果不要求保留比较寄存器，则首选使用此方法读取计数寄存器。 请注意，此 API 子程序被称为 CaptureCounter。

C 原型：

```
void  Timer32_ReadTimer(DWORD * pdwCount);
```

汇编：

```
mov   A,[pdwCount]
mov   X,[pdwCount+1]
lcall _Timer32_ReadTimer
```

参数：

None

返回：

pdwCount： 指向缓冲器的指针，其中保留着 Count 寄存器数据。 X 寄存器中装载返回缓冲器的地址。

副作用：

比较寄存器内容丢失。 比较条件立即变为真，或在下一个定时器输入时钟周期上变为真，具体取决于 CompareType 参数的设置为 “ 小于或等于 ”(Less than or Equal to) 或 “ 小于 ”(Less Than)。 如果 （或当）用户模块和全局中断已启用，则很可能在此 API 函数将控制权返回到其调用程序之前，中断将得到处理。 A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在扩展内存模式中，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 IDX_PP 页面指针寄存器。

固件源代码示例

在以下示例中，C 语言和汇编语言的对应关系非常简单和直接。 周期和比较值的显示值是基数值中的各个 "off-by-1"，因为这些寄存器是基于 0 的，即在其递减计数周期中以 0 为终止计数。 在 A 寄存器中而非堆栈中传递单一字节参数，这是汇编程序和 C 语言编译器针对用户模块 API 使用的性能优化方式。 当 C

程序在 Timer32.h 文件中发现 #pragma 快速调用声明时，它将对 "INT" 类型运用此机制，而不是将参数推入堆栈。

以下汇编语言源代码说明了 API 的使用。

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  Description:
;    This sample shows how to capture an event with a bounded time limit.
;    The count resolution is 1 us, with a bounded time limit of 16 seconds.
;
;    The interrupt should be set to interrupt on the Compare - Less than
;    equal. The capture input should be connected to the event that is
;    being measured. The clock should be connected to 24V2(VC2). The 24V1
;    (VC1) divider should be set to 8 and the 24V2(VC2) divider set to 3.
;
;    Computed time lapse is:  (0xFFFFFFFF - dwElapsedTime ) / 1 MHz
;
;    The foreground routine sets and starts the timer. The interrupt
;    level routine captures the value.
;
;  Parameters: none
;  Returns:    none
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export _main

area bss (RAM,REL)
_dwElapsedTime::
dwElapsedTime::    BLK    4

area text(ROM, REL, CON)
_main:

CapturePulse::
RAM_X_POINTS_TO_STACKPAGE
RAM_SETPAGE_CUR >dwElapsedTime
mov   X, SP               ; create a stack frame for arguments
add   SP, 4

mov   [dwElapsedTime],   0
mov   [dwElapsedTime+1], 0
mov   [dwElapsedTime+2], 0
mov   [dwElapsedTime+3], 0
mov   [X],   FFh          ; set the period to a Max count
mov   [X+1], FFh
mov   [X+2], FFh
mov   [X+3], FFh
lcall  Timer32_WritePeriod
mov   [X],   FFh          ; set the compare value to trigger at 16 secs
mov   [X+1], 0Bh          ; 4,294,967,295 - 16,000,000 = 4,278,967,295
mov   [X+2], DCh          ;  -> 0xFF0BDC00
mov   [X+3], 00h
lcall  Timer32_WriteCompareValue
```

```
lcall  Timer32_EnableInt    ; enable the timer interrupt mask
M8C_EnableGInt             ; enable global interrupts
RAM_X_POINTS_TO_INDEXPAGE
RAM_SETPAGE_IDX >dwElapsedTime
    mov   X, dwElapsedTime     ; point X to dwElapsedTime
lcall  Timer32_Start        ; start the timer - timer will start to
.WaitForCapture:
mov   A, [X+0]
or    A, [X+1]
or    A, [X+2]
or    A, [X+3]
jz    .WaitForCapture
add   SP, -4

.TimerDone:
;Evaluate captured value here!
;If dwElapsedTime is not > 1 then compute elapsed time.
;else if wElapsedTime is 1 or 0 then event did not occur within
;time limit.
; return to caller when complete

.terminate:
    jmp .terminate
```

位于文件 *Timer32int.asm* 中的中断级别子程序如下所示。

```
_Timer32_ISR:

   ;@PSoC_UserCode_BODY@ (Do not change this line.)
   ;---------------------------------------------------
   ; Insert your custom code below this banner
   ;---------------------------------------------------
   ;   NOTE: interrupt service routines must preserve
   ;   the values of the A and X CPU registers.
    push  X
push  A
mov   X, _dwElapsedTime
call  Timer32_ReadTimer
call  Timer32_Stop
pop   A
pop   X
;---------------------------------------------------
   ; Insert your custom code above this banner
   ;---------------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)

   reti
```

同一编码用 C 语言表示如下。 请注意，中断子程序必须用汇编语言编写。

```
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

DWORD    dwElapsedTime;
```

```
void main(void)
{
    Timer32_WritePeriod(0xffffffff);
Timer32_WriteCompareValue(0xff0bdc00);
Timer32_EnableInt();
M8C_EnableGInt;
Timer32_Start();
while( dwElapsedTime == 0 );
}
```

配置寄存器

32-bit 计数器使用四个数字 PSoC 块。 按照从左至右的次序放置，这些块名为 TIMER32_LSB、TIMER32_ISB1、TIMER32_ISB2 和 TIMER32_MSB.。每个块都通过 7 个寄存器实现个性化和参数化。 以下表格给出了作为常量和参数的 “ 特性 ” 值，命名为带有简要描述的位字段。 这些寄存器的符号名在用户模块实例的 C 语言和汇编语言接口文件 （“.h” 和 “.inc” 文件）中均有定义。

Table 3. 函数寄存器，组 1，CY8C29/27/24/22/21xxx 和 CY8CLED04/08/16

模块 / 位	7	6	5	4	3	2	1	0
MSB	0	0	1	Compare Type （比较类型）	Interrupt Type （中断类型）	0	0	0
ISB2	0	0	0	Compare Type （比较类型）	0	0	0	0
ISB1	0	0	0	Compare Type （比较类型）	0	0	0	0
LSB	数据反相	BCEN	0	Compare Type （比较类型）	0	0	0	0

BCEN 将终止计数输出导入到行广播总线中。 此位域在器件编辑器中通过直接配置广播线进行设置。 “ 数据反相 ” 标志用于控制捕获输入信号的意义，此参数通过显示在器件编辑器中的用户模块参数进行设置。 CompareType 标志表示比较函数的设置为 “ 小于或等于 ”(Less than or Equal to) 或 “ 小于 ”(Less Than)。 InterruptType 标志确定通过比较事件或终止计数触发中断 （另请参见控制寄存器中的 CaptureInt）。 CompareType 和 InterruptType 均在器件编辑器中直接通过用户模块参数进行设置，这些参数在之前相关主题部分中有所介绍。

Table 4. 输入寄存器，组 1

模块 / 位	7	6	5	4	3	2	1	0
MSB	0	0	1	1	时钟			
ISB2	0	0	1	1	时钟			
ISB1	0	0	1	1	时钟			
LSB	捕获				时钟			

使能在 16 个源其中的某个源中选择数据输入。 时钟从 16 个源之一选择输入时钟。 这两个参数都是在器件编辑器中设置的。

Table 5. Output 寄存器，组 1CY8C29/27/24/22/21xxx 和 CY8CLED04/08/16

模块 / 位	7	6	5	4	3	2	1	0
MSB	AuxClk		AuxEnable	AuxSelect		OutEnable	OutputSelect	
ISB2	AuxClk		0	0	0	0	0	0
ISB1	AuxClk		0	0	0	0	0	0
LSB	AuxClk		0	0	0	0	0	0

器件编辑器中的用户模块 “ClockSync” 参数决定 AuxClk 位的值。 虽然名称类似，但 AuxEnable 和 AuxSelect 位却关联 OutEnable 和 OutSelect 位域。AuxEnable 和 AuxSelect 允许将比较输出信号驱动至行输出总线之一，且通过在器件编辑器互连视图中操作行总线进行可视化控制。在将终止计数输出驱动至行总线之一或全局输出总线上时，设置 OutEnable。OutputSelect 控制着将从比较输出中驱动哪条总线。

Table 6. 计数寄存器 (DR0)，组 0

模块 / 位	7	6	5	4	3	2	1	0
MSB	Count(MSB)							
ISB2	Count(ISB1)							
ISB1	Count(ISB2)							
LSB	Count(LSB)							

计数寄存器是 32-bit 递减计数值，在每个使能输入为活动状态的时钟周期中递减 1。 在结束计数 （零值）之后的时钟周期中，将从周期寄存器的内容加载其值。 可以使用 Timer32 API 读取它。

Table 7. 周期寄存器 (DR1)，组 0

模块 / 位	7	6	5	4	3	2	1	0
MSB	Period(MSB)							
ISB2	Period(ISB1)							
ISB1	Period(ISB2)							
LSB	Period(LSB)							

周期寄存器为只写寄存器，可通过器件编辑器和 Timer32 API 进行设置。 在写入时，如果通过 API 禁用了用户模块，则值将被传输到计数寄存器中。 在结束计数之后的时钟周期中，其值将自动复制到计数寄存器中。

Table 8. 比较寄存器 (DR2)，组 0

模块 / 位	7	6	5	4	3	2	1	0
MSB	比较值 (MSB)							
ISB2	比较值 (ISB1)							
ISB1	比较值 (ISB2)							
LSB	比较值 (LSB)							

比较寄存器将保留此值，计数寄存器将测试此值以生成比较输出。 可以用器件编辑器和 Timer32 API 对其进行设置。

Table 9. 控制寄存器 (CR0)，组 0

模块 / 位	7	6	5	4	3	2	1	0
MSB	0	0	0	0	0	0	0	0
ISB2	0	0	0	0	0	0	0	0
ISB1	0	0	0	0	0	0	0	0
LSB	0	0	0	0	0	0	0	启用

设置 “ 启用 ” 表明已启用 Timer32，清除 “ 启用 ” 表明已禁用 PWMDB8L。 此参数使用 Timer32 API 进行修改。

版本历史记录

版本	创作者	说明
2.6	TDU	更新了时钟说明，内容包括： 当模块使用外部数字时钟时，应将行输入同步关闭，以获得最佳精度以及进行睡眠操作。
2.6.b	DHA	更新了用户模块数据手册中 API 函数的汇编原型。

Note PSoC Designer 5.1 在所有用户模块数据手册中都引入了 “ 版本历史 ”。 本数据表详细介绍了当前和先前用户模块版本之间的区别。

Copyright © 2012 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.