16-Bit 计数器数据手册 Counter16 V 2.5

Copyright © 2002-2011 Cypress Semiconductor Corporation. All Rights Reserved.

资源	PSoC® 模块			API 内存 （字节）		引脚 （每个外部 I/O）
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY8C29/27/24/22/21xxx、CY8C23x33、CYWUSB6953、CY7C64215、CY8CLED02/04/08/16、CY8CLED0xD、CY8CLED0xG、CY8CTST110、CY8CTMG110、CY8CTST120、CY8CTMG120、CY8CTMA120、CY8C21x45、CY8C22x45、CY8CTMA30xx、CY8C28x45、CY8CPLC20、 CY8CLED16P01						
16-bit	2	0	0	93	0	1
CY8C26/25xxx						
16-bit	2	0	0	142	0	1

如需一个或多个使用此用户模块的完全配置的功能性示例工程，请转到 www.cypress.com/psocexampleprojects.

功能和概述

- 16-bit 通用计数器使用两个 PSoC 模块
- 源时钟频率高达 48 MHz
- 基于终端计数自动重新加载周期
- 可编程脉冲宽度
- 输入启用 / 禁用计数器连续操作
- 比较输出*或*终端计数触发中断选项

16-bit 计数器用户模块可提供一个递减计数器，并配有可编程周期和脉冲宽度。 可从任何系统时基或外部源选择时钟和使能信号。 一旦启动，计数器便持续运行，并从周期寄存器重新加载其内部值，直至达到终端计数。 在每个时钟周期中，计数器都会将当前计数与比较寄存器中存储的值进行比较。 在每个时钟周期中，计数器都会将计数与比较寄存器中的值进行对比测试，测试两数为 “ 小于 ”(less than) 还是 “ 小于或等于 ”(less than or equal to) 关系。 比较器输出所提供的逻辑电平可路由至引脚或其他用户模块。 大多数 PSoC 器件系列的终端计数输出都可以这样进行路由。 如果您的器件有此功能，则会显示在器件编辑器中。 可设计为当计数器达到终端计数或比较器 （主要）输出激活时，则触发中断。

Figure 1. 计数器框图 （对于大多数 PSoC 器件），数据路径宽度 n = 16 位

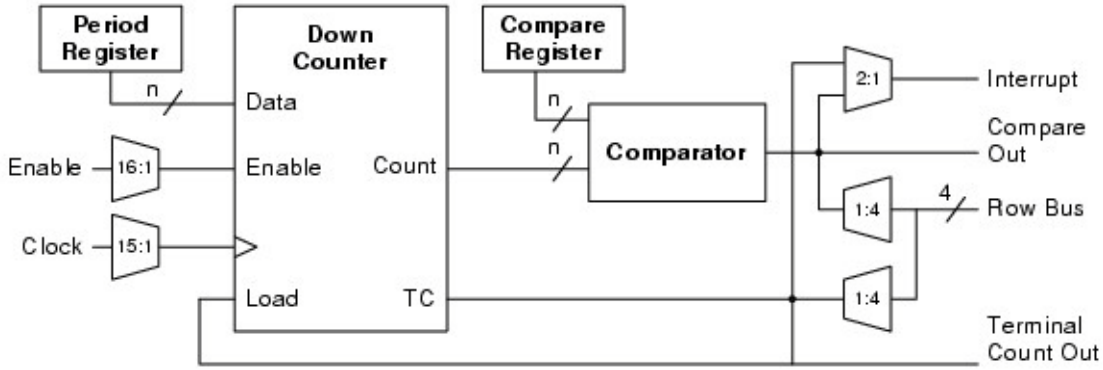
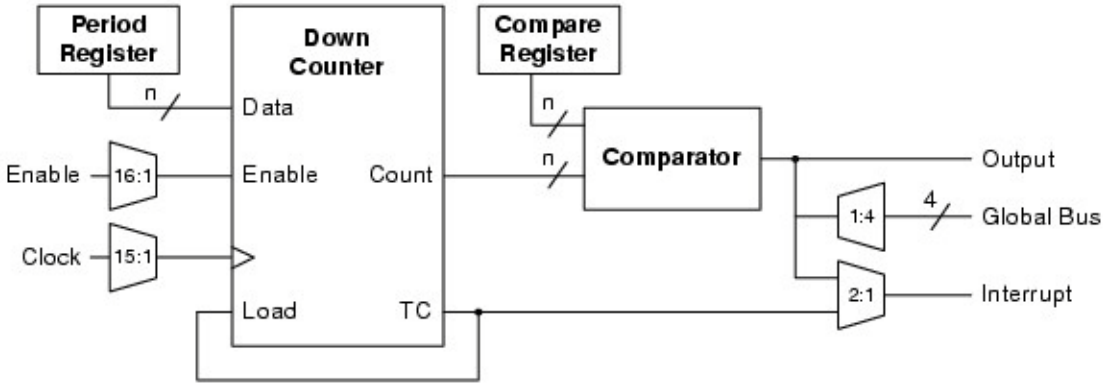


Figure 2. 计数器框图 （对于不含终端计数输出的器件），数据路径宽度 n = 16 位



功能说明

计数器用户模块利用了两个数字 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、终端计数和比较信号。 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，宽度大于 8 位的计数器即可作为一个完整的单片同步计数器进行操作。

计数器 API 提供了可用 C 语言和汇编语言调用的多种函数，以便停止或启动计数器操作以及读写各种数据寄存器。 数据寄存器的值也可以利用器件编辑器进行设置。 一旦启动，计数寄存器将沿着每个时钟周期的上升沿进行递减，且高电平有效使能输入信号将被激活。 当计数寄存器达到零即终端计数之后出现时钟上升沿时，计数寄存器将从周期寄存器中重新加载值。

随时可以用新的值来修改周期寄存器。 计数器停止运行后，在周期寄存器中写入值也会更改计数寄存器中的值。 计数器仍在运行时，在周期寄存器中写入值，不会将新的周期值更新到计数寄存器中，而是在达到终端计数后下一次重新加载时才会更新。 由于计数为 0 时即达到终端计数，因此操作周期和输出信号周期比周期寄存器中存储的值大 1。 输入时钟周期的持续时间是通过以下公式得出的。

Equation 1

$$OutputPeriod = (PeriodValue + 1)t_{CLK}$$

计数器在停止运行时将把输出置为低电平。 比较器在运行时，将控制输出信号的占空比。 在每个时钟周期中，比较器都会将计数寄存器的值和比较寄存器的值进行对比测试。 比较器将根据利用器件编辑器选定的选项，测试两数为 “ 小于 ”(less than) 还是 “ 小于或等于 ”(less than or equal to) 关系。 计数器将在比较发生周期后的时钟上升沿激活比较的高电平有效真实值。 比较值和周期的比率将设定输出波形的占空比。 占空比可通过以下公式计算得出。

[+] Feedback

Equation 2

$$DutyCycle = \begin{cases} \dfrac{CompareValue}{PeriodValue + 1}, \text{For } Less\ Than \text{ comparison} \\ \dfrac{CompareValue + 1}{PeriodValue + 1}, \text{For } Less\ Than\ Or\ Equal\ To \text{ comparison} \end{cases}$$

下表根据周期寄存器、比较寄存器和比较操作的设置，总结了一些特殊的输出信号条件。

Table 1. 计数器的特殊输出信号条件

周期寄存器的值	比较类型	比较寄存器的值	脉冲宽度高电平定时器与周期的比率
0	无需关注	> 0	1.0
0	≤	0	1.0
0	<	0	0.0
> 0	≤	0	1/ （周期 +1）
> 0	<	0	0.0
周期 = 比较	≤	周期 = 比较	1.0
周期 = 比较	<	周期 = 比较	周期 / （周期 +1）
比较值 > 周期	无需关注	比较值 > 周期	1.0

比较寄存器的值可利用器件编辑器进行设置，或在运行时利用 API 进行设置。 周期寄存器在达到终端计数前会为计数寄存器提供缓冲，但比较寄存器则不会以这种方式进行缓冲。 因此，对比较寄存器所做的更改在下一个时钟周期才会影响比较输出，而不是在达到终端计数后立即生效。 这可能会产生多个不同脉冲的周期。

在 CY8C29/27/24/22/21xxx 器件系列中，计数器用户模块会将终端计数信号作为辅助输出提供。 在达到终端计数后的时钟周期上升沿出现时，此高电平有效信号将被激活，此时计数寄存器将从周期寄存器加载值。

可设计为当达到终端计数或比较值为真时则触发中断。 比较器输出会在输出信号的上升沿触发中断，而终端计数会在输出信号下降沿之前的半个时钟周期触发中断。 此选项可利用器件编辑器进行设置。 在运行时可利用计数器 API 来启用或禁用中断。 在触发计数器中断前必须启用全局中断。

在修改比较寄存器时应格外小心，因为其值将与当前计数值共同决定计数器的输出状态。 为避免过早将输出信号置为低电平，以及避免发生潜在的短时脉冲，应在利用中断检测出终端计数条件后再修改比较寄存器的值。

对于需要更快占空比更新间隔的应用， 一旦检测到输出从高到低的跃变，则可更新比较。 请注意，如果比较使比较条件变为真，则输出将在下一个时钟周期被置为高电平。

在获取计数寄存器的值时应格外小心。 在读取计数寄存器时，会将其内容锁存入比较寄存器。 这会使输出占空比发生变化。

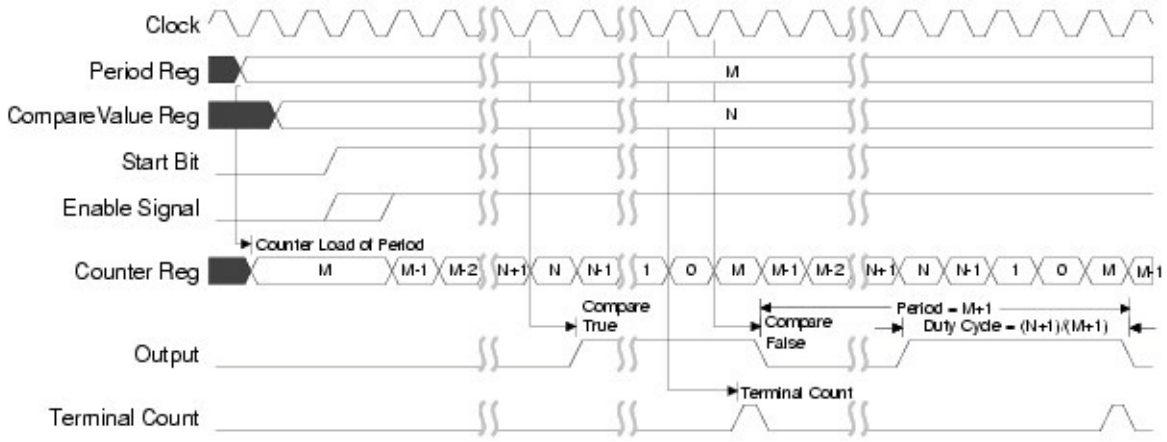
如需实时读取计数寄存器的值，可调用 ReadCounter() API 函数。 此函数将暂时禁用时钟，保存比较寄存器的内容、读取计数寄存器的值、读取比较寄存器的值、恢复比较寄存器，然后恢复时钟运作。 请参见 “ 应用程序编程接口 ” 部分对 ReadCounter() 函数的说明，以了解可能发生的副作用。

[+] Feedback

时序

计数器用户模块的操作可以被置为打通和关断，或由 PSoC 器件全局总线特性路由至计数器的外部引脚来计时。 下图说明了计数器用户模块的时序。

Figure 3. 计数器时序图



直流和交流电气特性

Table 2. CY8C26/25xxx 器件系列的计数器交流电气特性

参数	典型值	极限值	单位	条件和注释
最大输入频率	--	48[1]	MHz	16-bit 宽度，Vdd=5.0V[2]
最大输出频率	--	24[1]	MHz	Vdd=5.0V 及 48 MHz 输入时钟
	--	12[3]	MHz	Vdd=3.3V 及 24 MHz 输入时钟

Table 3. CY8C29/27/24/22/21xxx 器件系列的计数器交流电气特性

参数	典型值	极限值	单位	条件和注释
最大输入频率	--	48[1]	MHz	Vdd=5.0V[2]
最大输出频率	--	24[1]	MHz	Vdd=5.0V 及 48 MHz 输入时钟
	--	12[3]	MHz	Vdd=3.3V 及 24 MHz 输入时钟

电气特性注释

1. 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。
2. 所提供的使能信号始终为高电平；否则，限制为 24 MHz。
3. PSoC 模块在 3.3V 电压下运行时，可用的最快时钟频率为 24 MHz。

[+] Feedback

放置

计数器每 8 位分辨率使用一个数字 PSoC 模块。 如果分配了多个模块，则器件编辑器将把所有模块连续放置，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 include 文件直接访问计数器寄存器。 多个计数器用户模块所指定的模块名称见下表所示。

Table 4. 映射 PSoC 模块的符号名

PSoC 模块数	16-Bit 计数器
1	CNTR16_LSB
2	CNTR16_MSB

参数和资源

使用器件编辑器选择和放置计数器用户模块后，就可以选择或更改下列参数的值。

时钟 (Clock)

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）、从 24 MHz 系统时钟细分出来的较低频率 （24V1 与 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。

使能 (Enable)

从一个可用源中选择 “ 使能 ”(Enable) 参数。 在无需重设计数器的情况下，高电平输入将启用连续计数，而低电平输入将禁用计数。

输出 (Output)

“ 输出 ”(Output) 参数可以设置为禁用，或将其路由至四个全局输出信号之一；此参数仅适用于 PSoC 器件中的 CY8C26/25xxx 系列。

比较输出 (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 系列成员中显示。

终端计数输出 (TerminalCountOut)

终端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 系列成员中显示。

周期 (Period)

此参数可以设置计数器的周期。 数值范围介于 0 到 2^n-1 之间，其中 n 是计数器的位宽。 周期已经载入周期寄存器。 计数器的有效输出波形周期为周期计数 + 1。 可使用 API 修改此值。

比较值 (CompareValue)

此参数可设置比较寄存器的比较值。 数值范围介于 0 到周期值之间。 可使用 API 修改此值。

比较类型 (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。

[+] Feedback

中断类型 (InterruptType)

当比较器为真或达到终端计数时，计数器均可生成中断。 单独的寄存器可以独立启用中断。

时钟同步 (ClockSync)

在 PSoC 器件中，数字模块可以在系统时钟以外还提供其他时钟源。 数字时钟源甚至可以用连锁方式串联起来。 这样就引入了与系统时钟相关的时滞现象。 由于各种数据路径优化，在 CY8C29/27/24/22/21xxx PSoC 器件系列中，这些时滞更具危害性，特别是应用于系统总线的部分。 此参数可用于控制时钟时滞，确保读取和写入 PSoC 模块寄存器的值时进行正确操作。 此参数的正确数值应当由下表决定。

时钟同步 (ClockSync) 数值	使用情况
与 SysClk 同步 (Sync to SysClk)	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 （当 VC3 由 SysClk 驱动时）、32KHz 和采用 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。
与 SysClk*2 同步 (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。
直接使用 SysClk (Use SysClk Direct)	在需要 24 MHz (SysClk/1) 时钟时使用。 虽然此选项并不实际执行同步，但提供了对系统时钟本身的低时滞访问方式。 如果选择此选项，上面的 “ 时钟 ”(Clock) 参数设置将被覆盖。 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，而不要使用 VC1、VC2、VC3 或数字模块。
不同步 (Unsynchronized)	在选定 48 MHz (SysClk*2) 输入时使用。 在需要不同步输入时使用。 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。 在睡眠期间依然保持活动状态的模块需要进行此设置。

反相使能 (InvertEnable)

此参数可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。 “ 反相使能 ”(InvertEnable) 仅适用于 PSoC 器件中的 CY8C29/27/24/22/21xxx 系列。

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，将有另外两个参数可供使用。 此选项可通过以下路径找到: **项目 (Project) > 设置 (Settings) > 芯片编辑器 (Chip Editor)**. 当使用多个外覆层并且外覆层上的多个用户模块共享中断时

- 中断 API (Interrupt API)
- 中断调度模式 (IntDispatchMode)

中断 API (InterruptAPI)

“ 中断 API”(InterruptAPI) 参数允许按条件生成用户模块的中断处理程序和中断矢量表入口。 选择 “ 启用 ”(Enable) 可生成中断处理程序和中断矢量表入口。 选择 “ 禁用 ”(Disable) 可避免生成中断处理程序和中断矢量表入口。 对于具有多个外覆层并且不同外覆层共享一个模块源的项目，强烈建议要正确选择是否生成中断 API。 仅在必要时选择生成中断 API，可能就避免了生成中断调度码的需求，从而降低开销。

[+] Feedback

中断调度模式 (IntDispatchMode)

外覆层 外覆层 每次请求共享中断时都会执行此检测。 这样会增加延迟，还会产生一个处理共享中断请求的非决定性程序，但不需要任何 RAM。 外覆层 这种计算可减少中断延迟，并产生一个处理共享中断请求的决定性程序，但会占用 RAM 空间。

应用程序编程接口

应用程序编程接口 (API) 例程作为用户模块的一部分提供，从而使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 “include” 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。 选择这种 “寄存器易失 ” 策略是为了提高效率，并且从 PsoC Designer 的 1.0 版本起已开始使用。 C 编译器自动遵循此要求。 汇编语言编程人员也必须确保自己的代码遵守这一策略。 虽然有些用户模块的 API 函数可能保留 A 和 X 不变，但并不保证在未来也将如此。

以下是为 Counter16: 提供的变量：

Counter16_PERIOD

说明：

表示器件编辑器中为 Counter16 的 “ 周期 ” 字段选择的值。 该值的范围介于 0 到 65535 之间。

Counter16_COMPARE_VALUE

说明：

表示器件编辑器中为 Counter16 的 “ 脉冲宽度 ” 字段选择的值。 该值的范围介于 0 到 65535 之间。

以下是为 Counter16: 提供的 API 编程例程：

Counter16_Start

说明：

启动 Counter16 用户模块。 若使能输入为高电平，则计数器会开始递减计数。

C 原型：

```
void Counter16_Start(void);
```

汇编：

```
lcall  Counter16_Start
```

参数：

无

返回值：

无

副作用：

A 和 X 寄存器均有可能在此函数的当前实现或后续实现中被修改。 对于大型存储器模型 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

[+] Feedback

Counter16_Stop

说明：

停止计数器操作。

C 原型：

```
void Counter16_Stop(void);
```

汇编：

```
lcall  Counter16_Stop
```

参数：

无

返回值：

无

副作用：

输出将被置为低电平，且对周期寄存器的写入操作会使计数寄存器更新为新的周期值。 A 和 X 寄存器均有可能在此函数的当前实现或后续实现中被修改。 对于大型存储器模型 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

Counter16_EnableInt

说明：

启用中断模式运行。

C 原型：

```
void Counter16_EnableInt(void);
```

汇编：

```
lcall  Counter16_EnableInt
```

参数：

无

返回值：

无

副作用：

A 和 X 寄存器均有可能在此函数的当前实现或后续实现中被修改。 对于大型存储器模型 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

Counter16_DisableInt

说明：

禁用中断模式运行。

C 原型：

```
void Counter16_DisableInt(void);
```

汇编：

```
lcall  Counter16_DisableInt
```


[+] Feedback

参数：

无

返回值：

无

副作用：

A 和 X 寄存器均有可能在此函数的当前实现或后续实现中被修改。 对于大型存储器模型 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

Counter16_WritePeriod

说明：

将周期值写入周期寄存器。 如果 Counter16 停止运行或计数器达到零计数，则周期值将立即从周期寄存器转入计数寄存器中。

C 原型：

```
void Counter16_WritePeriod(WORD wPeriod);
```

汇编：

```
mov   X, [wPeriod]                  ;load the MSB, e.g. from RAM
mov   A, [wPeriod+1]                ;load the LSB
lcall  Counter16_WritePeriod
```

参数：

wPeriod: wPeriod 是一个介于 0 和 65535 之间的值。 MSB 传递到 X 寄存器，LSB 传递到 A 寄存器。

返回值：

无

副作用：

A 和 X 寄存器均有可能在此函数的当前实现或后续实现中被修改。 对于大型存储器模型 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

Counter16_WriteCompareValue

说明：

此参数可将比较值写入比较值 (CompareValue) 寄存器。

C 原型：

```
void Counter16_WriteCompareValue(WORD wCompareValue);
```

汇编：

```
mov   X, [wCompareValue]
mov   A, [wCompareValue+1]
lcall  Counter16_WriteCompareValue
```

参数：

wCompareValue：wCompareValue 是一个介于零和周期值之间的值。 MSB 传递到 X 寄存器，LSB 传递到 A 寄存器。

返回值：

无

[+] Feedback

副作用：

当计数器为活动状态时，写入比较值 (CompareValue) 寄存器将改变输出占空比。 这可能会使输出发生短时脉冲或意外改变。 A 和 X 寄存器均有可能在此函数的当前实现或后续实现中被修改。 对于大型存储器模型 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

Counter16_wReadCompareValue

说明：

读取比较值 (CompareValue) 寄存器。

C 原型：

```
WORD  Counter16_wReadCompareValue(void);
```

汇编：

```
lcall  Counter16_wReadCompareValue
mov   [wCompareValue], X
mov   [wCompareValue+1], A
```

参数：

wReadCompareValue: 返回从比较寄存器中得到的 16-bit 值。 MSB 传递到 X 寄存器，LSB 传递到 A 寄存器。

返回值：

无

副作用：

A 和 X 寄存器均有可能在此函数的当前实现或后续实现中被修改。 对于大型存储器模型在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

过期别名：

wCounter16_ReadCompareValue - 此名称可能在以后的 PSoC Designer 版本中被取消。

Counter16_wReadCounter

说明：

读取计数寄存器的值 （硬件 DR0 寄存器），并保留比较寄存器的值。 无论在计数器运行还是停止时，均可调用此函数读取计数寄存器。 即使比较寄存器和计数寄存器的值暂时相等，也可以防止意外中断。 但是，会有一些严重的副作用 （如下所示）。

C 原型：

```
WORD  Counter16_wReadCounter(void);
```

汇编：

```
lcall  Counter16_wReadCounter
mov   [wCount], X
mov   [wCount+1], A
```

参数：

无

返回：

wReadCounter：计数寄存器中的 wReadCounter 值。 MSB 传递到 X 寄存器，LSB 传递到 A 寄存器。

[+] Feedback

副作用：

如果已启用计数器用户模块，并在调用此函数时计数，由于必须暂时停止用户模块，有些时钟可能会被忽略（相应的计数寄存器减少量也会被忽略）。A 和 X 寄存器均有可能在此函数的当前实现或后续实现中被修改。对于大型存储器模型 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

过期别名：

wCounter16_ReadCounter - 此名称可能在以后的 PSoC Designer 版本中被取消。

固件源代码示例

在以下示例中，C 语言代码和汇编语言代码之间的对应比较简单。通过 A 寄存器中的 LSB 和 X 寄存器中的 MSB 来传递一个简单的双字节 “INT” 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。当在 *Counter16.h* 文件中遇到 #pragma 快速调用声明时，C 语言编译器会对 “INT” 类型应用此机制，而不会将参数推入堆栈。

以下源代码说明了 API 在汇编语言中是如何使用的。

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;    This sample shows how to create a clock divider.  This specific
;    example divides the clock by 1000.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

include "m8c.inc"                       ; include the device interface
include "PSoCAPI.inc"                   ; include the API interface file

export _main

_main:

   mov   X, 03h                         ; set the period to 1000
   mov   A, E7h                         ;    i.e., 1000-1 = 999 = 0x03e7
   call  Counter16_WritePeriod

   mov   X, 01h                         ; Generate a 50% duty cycle
   mov   A, F3h                         ;    i.e., 500-1 = 499 = 0x01F3
   call  Counter16_WriteCompareValue

   call  Counter16_EnableInt            ; enable the Counter Interrupt
   M8C_EnableGInt                       ; enable global interrupts
   call  Counter16_Start                ; start to count when the enable
                                        ;    input is asserted
.terminate:
        jmp .terminate
```


[+] Feedback

同一编码用 C 语言表示如下。

```
/*****************************************************************************
* Description:
*    This sample shows how to create a clock divider.  This specific
*    example divides the clock by 1000.
*****************************************************************************/

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

void main(void)
{
    Counter16_WritePeriod(999);            /* set the period to 1000   */
    Counter16_WriteCompareValue(499);      /* generate a 50 duty cycle */
    Counter16_EnableInt();                 /* disable the interrupt    */
    M8C_EnableGInt;                        /* enable global interrupts */
    Counter16_Start();                     /* start the counter        */
}
```

在上述两种情形下，只需要 include 文件 *Counter16.h*，以提供必要的声明（编译指令和原型）。 当应用需要直接访问寄存器时，汇编语言和 C 语言的 include 文件均可提供符号名。 在 CY8C29/27/24/22/21xxx 用户模块中，include 文件还可以提供宏定义，以支持某些较短的 API 函数源代码的嵌入式扩展。 可分别使用 *PSoCAPI.inc* 和 *PSoCAPI.h* 文件，而不再使用 *Counter16.inc* 和 *Counter16.h* 文件。 这些文件针对 PSoC Designer 项目中实例化和放置的每个用户模块，分别设置了 include 语句。

配置寄存器

16-bit 计数器使用两个数字 PSoC 模块。 按照从左到右的放置次序，这些模块分别名为 CNTR16_LSB 和 CNTR16_MSB.。每个模块都通过七个寄存器实现个性化和参数化设置。 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 5. 函数寄存器，组 1，CY8C26/25xxx

模块 / 位	7	6	5	4	3	2	1	0
MSB	0	0	1	比较类型	中断类型	0	0	1
LSB	0	0	0	比较类型	0	0	0	1

Table 6. 函数寄存器，组 1，CY8C29/27/24/22/21xxx

模块 / 位	7	6	5	4	3	2	1	0
MSB	数据反相	0	1	比较类型	中断类型	0	0	1
LSB	0	BCEN	0	比较类型	0	0	0	1

BCEN 会将比较输出传送至行广播总线。 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 ”（Data Invert）标志用于控制使能输入信号的意义，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。 “ 比较类型 ”（CompareType）标志表示将比较函数设置为 “ 小于或等于 ”（Less Than or Equal）还是 “ 小于 ”（Less Than）。 “ 中断类型 ”（InterruptType）标志决定在比较事件中还是在终端计数中触发中断。 “ 比较类型 ”（CompareType）与 “ 中断类型 ”（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。

[+] Feedback

Table 7. 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 (Clock)			
LSB	使能 (Enable)				时钟 (Clock)			

“ 使能 ”(Enable) 从 16 个源之一选择数据输入。 “ 时钟 ”(Clock) 从 15 个源之一选择输入时钟。 这两个位域的值均取决于对器件编辑器中同名的用户模块参数的设置。

Table 8. 输出寄存器，组 1，CY8C29/27/24/22/21xxx

模块 / 位	7	6	5	4	3	2	1	0
MSB	AuxClk		AuxEnable	AuxSelect		OutEnable	OutputSelect	
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。 尽管命名相似，AuxEnable 和 AuxSelect 位却与 OutEnable 和 OutSelect 位域相关。 AuxEnable 和 AuxSelect 允许将终端计数输出信号输出到其中一个行输出总线，这两个参数通过在 “ 器件编辑器互连视图 ” 中以图形方式操纵行总线来控制。 当比较输出被输出到某个行或全局输出总线时，将设置 OutEnable。 OutputSelect 控制哪条总线将从比较输出中输出。

Table 9. 计数寄存器 (DR0)，组 0

模块 / 位	7	6	5	4	3	2	1	0
MSB	计数 (MSB)							
LSB	计数 (LSB)							

计数寄存器是 16-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。 在终端计数 （零值）之后的时钟周期中，将从周期寄存器的内容加载其值。 可使用 Counter16 API 读取此值。

Table 10. 周期寄存器 (DR1)，组 0

模块 / 位	7	6	5	4	3	2	1	0
MSB	周期 (MSB)							
LSB	周期 (LSB)							

周期寄存器为只写寄存器，可通过器件编辑器和 Counter16 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。 在终端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 11. 比较寄存器 (DR2)，组 0

模块 / 位	7	6	5	4	3	2	1	0
MSB	比较值 (MSB)							
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。 可通过器件编辑器和 Counter16 API 对其进行设置。

[+] Feedback

Table 12. 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
LSB	0	0	0	0	0	0	0	启动 / 停止

“ 启动 / 停止 ”(Start/Stop) 设置时表示 Counter24 为启用状态，清除时为禁用状态。 此参数使用 Counter16 API 进行修改。

版本历史记录

版本	创作者	说明
2.5	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。

Note PSoC Designer 5.1 在所有的用户模块数据手册中提供了版本历史记录。 本数据手册详细介绍了当前和先前用户模块版本之间的区别。

Copyright © 2002-2011 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.