32-Bit 计数器数据手册 Counter32 V 2.5

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	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, CY8C22x45, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8CLED16P01						
32-bit	4	0	0	146	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 通用计数器占用四个 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 / 禁用连续计数器操作
- 比较输出*或*终端计数中断触发选择

32-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输出。 计数器可从任何系统基准时钟或外部信号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值与比较寄存器中存储的值进行比较。 每个时钟周期，计数器都会将计数值与比较寄存器中的值进行比较测试，测试两数为 “ 小于 ” 还是 “ 小于或等于 ” 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。 在大多数 PSoC 器件中终端计数信号（TC） 也可以像这样路由出去。 如果您的器件有此功能，则会显示在器件编辑器中。 当计数器计数结束或比较器 （主要）输出激活时，可设置触发中断。

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

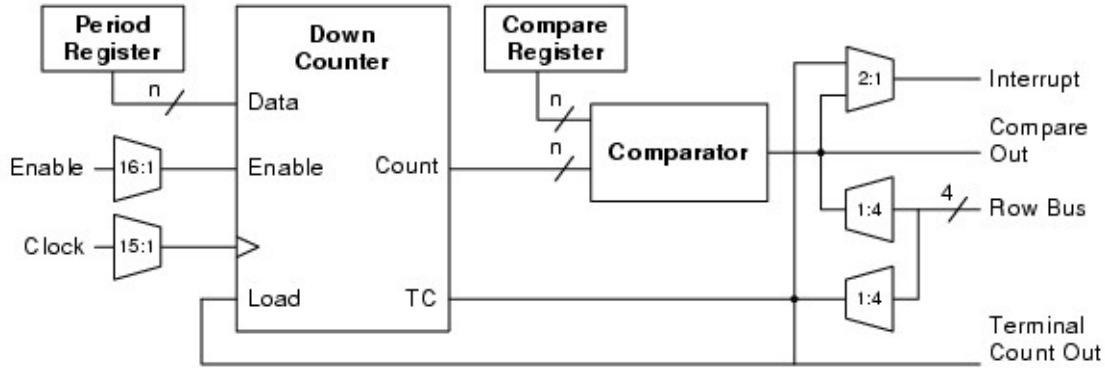
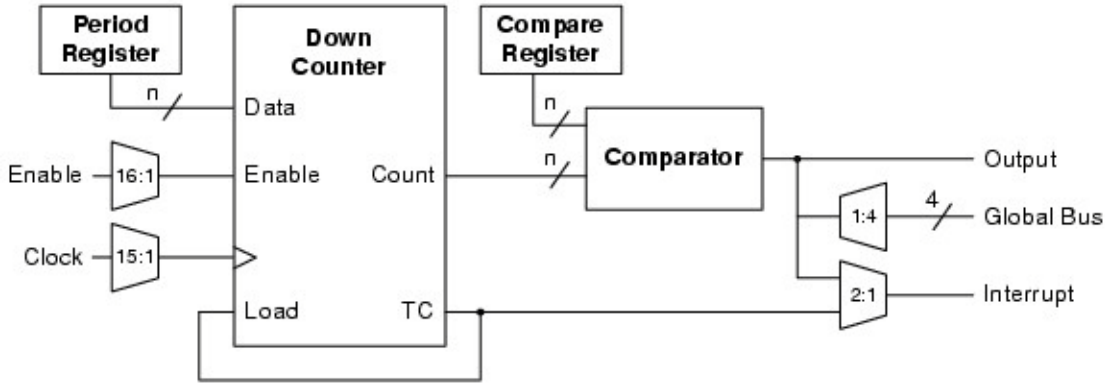


Figure 2. 计数器框图 （对于不含终端计数输出的器件），数据路径宽度 n = 32 位



功能描述

32-bit 计数器用户模块占用了四个数字 PSoC 模块，每个模块提供 8 位分辨率。 这些连续的模块彼此链接，因此能够同时链接内部进位位、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）相互链接，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 提供了多种可用 C 语言和汇编语言调用的函数，以便停止或启动计数器操作以及读写各种数据寄存器。 也可以使用器件编辑器确定数据寄存器值。 在计数器的使能输出信号为高的情况下，一旦启动，计数寄存器会在每个时钟周期上升沿递减。 当计数寄存器达到零即结束计数之后，下一个时钟的上升沿，计数寄存器将从周期寄存器中重新加载值。

周期寄存器的值可以随时修改 计数器停止运行后，向周期寄存器中写入值也会更改计数寄存器中的值。 计数器运行时，向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，而是当前计数周期结束后下一次重新加载时才会更新。 由于在计数为 0 时才会到达终端计数，因而操作和输出信号的周期会比存储在周期寄存器中的值大 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。 在运行时，比较器会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。

Equation 2

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

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. 计数器特殊输出信号条件

周期寄存器值	Compare Type（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。 周期寄存器在达到计数结束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就会影响比较输出，而不是当前计数周期结束后。 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，计数器用户模块将结束计数信号作为辅助输出提供。 在达到结束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。

可设置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结束计数条件后再修改比较寄存器的值。

对于需要频繁更新占空比的应用，计数器的输出可路由至某个对其状态进行轮询的引脚。 一旦检测到输出从高到低的跃变，即可更新比较。 请注意，如果比较寄存器引起比较条件为真，则输出将在下一个时钟周期被置为高电平。

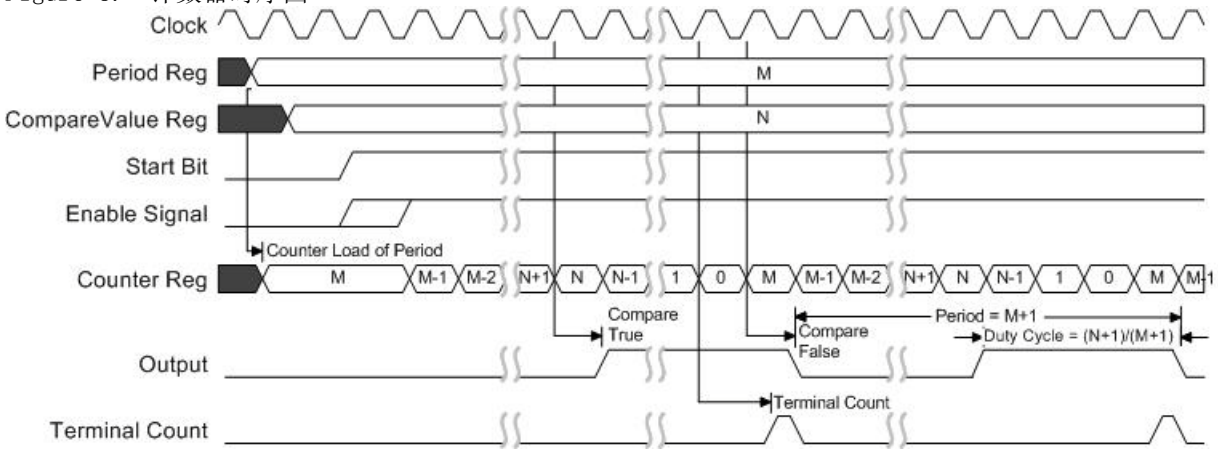
在获取计数寄存器的值时应格外小心。 在读取计数寄存器时，会将其内容锁存入比较寄存器。 这会使输出占空比发生变化。

如需即刻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 ” 一节中有关 ReadCounter() 函数的说明，以了解可能产生的副作用。

时序

计数器用户模块的操作可以被置为打开和关断，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. CY8C29/27/24/22/21xxx 器件系列的计数器交流电气特性

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

电气特性说明

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。

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 3. 映射 PSoC 模块的符号名

PSoC 模块数	32-Bit 计数器
1	CNTR32_LSB
2	CNTR32_ISB1
3	CNTR32_ISB2
4	CNTR32_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

从 16 个源其中的某个源中选择 “ 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 系统时钟向下分频的较低频率 （VC1、VC2 和 VC3）、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

从一个可用源中选择 “ 启用 ”(Enable) 参数。 高电平输入将启用继续计数功能，而低电平输入则禁用计数功能且无需复位计数器。

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。 该参数仅在 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。 该参数仅在 PSoC 器件的 CY8C29/27/24/22/21xxx 系列成员中显示。

Period （周期）

此参数设置计数器的周期。 数值范围介于 0 到 2^n-1 之间，其中 n 是计数器的位宽。 此周期将加载到周期寄存器中。 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。 容许值介于 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。

中断类型 (InterruptType)

当比较器为真或达到计数终止时，计数器均可触发中断。 单独的寄存器可以独自启用中断。

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 PSoC 器件系列具有更为关键的意义，因为其中存在各种数据路径优化，特别是那些应用于系统总线的部分。 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。

时钟同步 (ClockSync) 值	使用说明
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。 这样的时钟源包括 VC1、VC2、VC3 （在 VC3 由 SysClk 驱动时）、32 KHz 和以 SysClk 作为时钟源的数字 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。 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。 如果选择此选项，上面的 “ 时钟 ”(Clock) 参数设置将被覆盖。 在所有分频器联合起来的净结果生成了 24 Mhz 的输出时，一定要使用此选项，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。 在需要非同步输入时使用。 一般来说，只有在中断生成是计数器的唯一用途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。

反向使能 (InvertEnable)

此参数确定使能输入信号的类型。 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。 反向使能 (InvertEnable) 仅适用于 PSoC 器件的 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， **有两个附加参数可供使用** 。 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，中断生成控制非常重要：

中断 API

中断 API 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。 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。

中断调度模式 (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” 会使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

应用程序编程接口 (API) 子程序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，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。

对于大型储存器模型器件， 保存 CUR_PP、IDX_PP、MVR_PP 以及 MVW_PP 寄存器中的所有值也是调用者的职责。 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。

以下是为 Counter32: 提供的变量：

Counter32_PERIOD

说明：

表示器件编辑器中为 Counter32 的 “ 周期 ” 字段选择的值。 该值的范围介于 0 到 4294967295 之间。

Counter32_COMPARE_VALUE

说明：

表示器件编辑器中为 Counter32 的 “ 脉冲宽度 ” 字段选择的值。 该值的范围介于 0 到 4294967295 之间。

以下是为 Counter32 提供的 API 编程子程序。

Counter32_Start

说明：

启动 Counter32 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 Counter32_Start(void);
```

汇编：

```
lcall  Counter32_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。 对于大型储存器模型 (CY8C29xxx) 中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

Counter32_Stop

说明：

停止计数器操作。

C 原型：

```
void Counter32_Stop(void);
```

汇编：

```
lcall  Counter32_Stop
```

参数：

None

返回值：

None

副作用：

输出将被重置为低电平，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 寄存器可以由此函数的本次执行或以后执行而被修改。对于大型储存器模型 (CY8C29xxx)中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

Counter32_EnableInt

说明：

启用中断模式运行。

C 原型：

```
void Counter32_EnableInt(void);
```

汇编：

```
lcall  Counter32_EnableInt
```

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。 对于大型储存器模型 (CY8C29xxx) 中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

Counter32_DisableInt

说明：

禁用中断模式运行。

C 原型：

```
void Counter32_DisableInt(void);
```

汇编：

```
lcall  Counter32_DisableInt
```

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。 对于大型储存器模型 (CY8C29xxx) 中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

Counter32_WritePeriod

说明：

将周期值写入周期寄存器。 如果 Counter32 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 Counter32_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 _Counter32_WritePeriod
```

参数：

dwPeriod： 提供一个介于 0 和 2^{32}-1 之间的数值。 X 寄存器中装载 dwPeriod 的 MSB 的地址。

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。 对于大型储存器模型 (CY8C29xxx) 中的所有 RAM 页指针寄存器，也是如此。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

Counter32_WriteCompareValue

说明：

此参数可将比较值写入比较值 (CompareValue) 寄存器。

C 原型：

```
void Counter32_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 _Counter32_WriteCompareValue
```

参数：

dwCompareValue: 介于 0 和期间值之间的数值。寄存器 X 中加载 dwCompareValue 的 MSB 的地址。

返回值：

None

副作用：

当计数器运行时写入比较值 (CompareValue) 寄存器将改变输出的占空比。这样可能导致输出短时脉冲或无意中更改占空比。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。对于大型储存器模型 (CY8C29xxx) 中的所有 RAM 页指针寄存器，也是如此。在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

Counter32_ReadCompareValue

说明：

读取比较值 (CompareValue) 寄存器。

C 原型：

```
void Counter32_ReadCompareValue(DWORD * pdwCompareValue);
```

汇编：

```
mov   A,[pdwCompareValue]
mov   X,[pdwCompareValue+1]
lcall _Counter32_ReadCompareValue
```

参数：

pdwCompareValue: 指向 DWORD 的指针，其中保留着 CompareValue 寄存器数据。X 寄存器中加载 pdwCompareValue 的 MSB 的地址。

返回值：

在指定缓冲器中返回 CompareValue 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。对于大型储存器模型 (CY8C29xxx) 中的所有 RAM 页指针寄存器，也是如此。在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。当前，仅修改 IDX_PP 页面指针寄存器。

Counter32_ReadCounter

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，会有一些重大的副作用 （见副作用）。

C 原型：

```
void Counter32_ReadCounter(DWORD * pdwCount);
```

汇编：

```
mov   A,[pdwCount]
mov   X,[pdwCount+1]
lcall _Counter32_ReadCounter
```

参数：

pdwCount：指向 DWORD 的指针，其中保留着 Counter 寄存器数据。X 寄存器中装载 pdwCount 的 MSB 的地址。

返回值：

在指定缓冲器中返回 Count 值。

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（丢失计数寄存器的相应减少量也会被忽略）。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。对于大型储存器模型下 (CY8C29xxx) 中的所有 RAM 页指针寄存器，也是如此。在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。当前，仅修改 IDX_PP 页面指针寄存器。

固件源代码示例

以下源代码说明了 Counter32 API 在汇编语言中是如何使用的。

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;    This sample shows how to create a 1 Hz interrupt and 1Hz output
;    with a 50% duty cycle. A Counter32 User Module must have been
;    configured with a 24 MHz input clock. Interrupts may be configured
;    either for triggering on the terminal count or compare true
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

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

STACKFRAME_SIZE:         equ 4        ; Nor bytes for a 32-bit parameter

export _main

_main:
   mov   X, SP                        ; create a stack frame pointer
   add   SP, STACKFRAME_SIZE          ;   and allocate space for a DWORD

   mov   [X+0], 01h                   ; load the Period onto the stack
   mov   [X+1], 6eh                   ;    = 24,000,000 -1 = 0x16e35ff
   mov   [X+2], 35h                   ;    ...
   mov   [X+3], ffh                   ;    and...
   call  Counter32_WritePeriod        ; Update the Period register
   add   SP, -STACKFRAME_SIZE         ; deallocate the stack frame

   mov   X, SP                        ; create a stack frame pointer
   add   SP, STACKFRAME_SIZE          ;   and allocate space for a DWORD
   mov   [X+0], 00h                   ; load the Compare Value on the stack
   mov   [X+1], b7h                   ;    = (24,000,000-1)/2 = 0x00b71aff
   mov   [X+2], 1ah                   ;    ...
```

```
   mov   [X+3], ffh                    ;    and...
   call  Counter32_WriteCompareValue   ; Update the Compare register

   add   SP, -STACKFRAME_SIZE          ; deallocate the stack frame
   call  Counter32_EnableInt           ; enable Counter interrupts
   M8C_EnableGInt                      ; enable global interrupts
   call  Counter32_Start               ; start to count when the enable
                                       ;    input is asserted
.terminate:
        jmp .terminate
```

同一代码用 C 语言表示如下：

```
#include <m8c.h>// part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
void main(void)
{
    /* Create a 1 Hz square wave */
    Counter32_WritePeriod(24000000-1);              /* set the period    */
    Counter32_WriteCompareValue((24000000-1)/2);    /* set the duty cycle*/
    Counter32_EnableInt();                  /* enable Counter interrupts  */
    M8C_EnableGInt;                         /* enable global interrupts   */
    Counter32_Start();                      /* start the counter!         */
}
```

配置寄存器

32-bit 计数器占用四个数字 PSoC 模块。 按照从左到右的放置次序，这些模块分别名为 CNTR32_LSB、CNTR32_ISB1、CNTR32_ISB2 和 CNTR32_MSB.。每个模块都通过七个寄存器进行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 4. 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	Compare Type （比较类型）	Interrupt Type （中断类型）	0	0	1
ISB2	0	0	0	Compare Type （比较类型）	0	0	0	1
ISB1	0	0	0	Compare Type （比较类型）	0	0	0	1
LSB	0	BCEN	0	Compare Type （比较类型）	0	0	0	1

BCEN 将比较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。 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 5. 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	启用				时钟			

使能在 16 个源其中的某个源中选择数据输入。 “ 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 6. 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	
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，设置 OutEnable。OutputSelect 控制着将从比较输出中驱动哪条总线。

Table 7. 计数寄存器 (DR0)，组 0

模块 / 位	7	6	5	4	3	2	1	0
MSB	Count(MSB)							
ISB2	Count(ISB2)							
ISB1	Count(ISB1)							
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 Counter32 API 读取此值。

Table 8. 周期寄存器 (DR1)，组 0

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

周期寄存器为只写寄存器，可通过器件编辑器和 Counter32 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 9. 比较寄存器 (DR2)，组 0

模块 / 位	7	6	5	4	3	2	1	0
MSB	Compare Value (MSB)							
ISB2	Compare Value (ISB2)							
ISB1	Compare Value (ISB1)							
LSB	Compare Value (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 Counter32 API 对其进行设置。

Table 10. 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	Start/Stop

“ 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 Counter32 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。
2.5.b	DHA	更新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。