8-Bit 脉冲宽度调制器数据表 PWM8 V 2.60

资源	PSoC® 模块			API 内存 （字节）		引脚 （根据外部 I/O）
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY8C29/27/24/22/21xxx、CY8C23x33、CY7C64215/603xx、CYWUSB6953、CY8CLED02/04/08/16、CY8CLED0xD、CY8CLED0xG、CY8CNP102、CY8CTST110、CY8CTMG110、CY8CTST120、CY8CTMG120、CY8CTMA120、CY8C21x45、CY8C22x45、CY8CTMA30xx、CY8C28x45、CY8CPLC20、CY8CLED16P01、CY8C28xxx、 CY8C21x12						
8-bit	1	0	0	67	0	1
CY8C26/25xxx						
8-bit	1	0	0	103	0	1

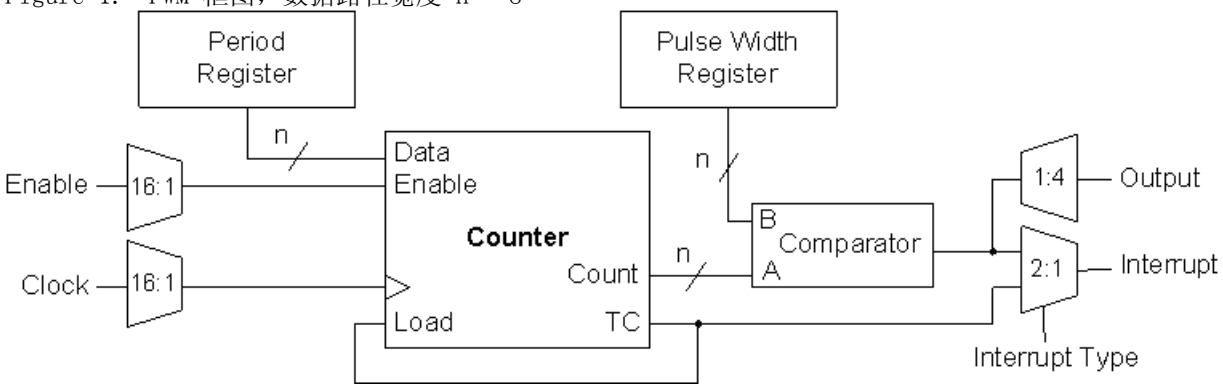
如需一个或多个使用此用户模块的完全配置的功能性示例项目，请转到 www.cypress.com/psocexampleprojects.

功能和概述

- 8 位通用脉冲宽度调制器使用一个 PSoC 模块
- 源时钟频率高达 48 MHz
- 每个脉冲周期的自动重新加载周期
- 可编程脉冲宽度
- 输入启用 / 禁用连续计数器操作
- 输出或终端计数的上升沿上的中断选项

8-bit PWM 用户模块为一个具有可编程周期和脉冲宽度的脉冲宽度调制器。 可以从多个源中选择时钟以及启用信号。 其他用户模块可以把输出信号路由至某一引脚或全局输出总线其中某一总线，以供内部使用。 可以对中断进行编程，使其在输出的上升沿或当计数器达到终端计数条件时触发。

Figure 1. PWM 框图，数据路径宽度 n = 8



功能说明

PWM 用户模块采用一个数字 PSoC 模块，以提供 8 位总分辨率。

PWM API 提供可用 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。 也可以使用器件编辑器建立数据寄存器值。 一旦启动，计数寄存器会在每个时钟周期上升沿出现时递减，在该上升沿上，将置位高电平有效启用输入信号。 终端计数之后，再次出现时钟上升沿时，计数寄存器重新加载周期寄存器中的值 （计数寄存器的值达到 0）。

随时可以通过新值修改周期寄存器。 当 PWM 停止时，将某个值写入到周期寄存器中也可更改计数寄存器中的值。 当 PWM 正在运行时，在下一个重新加载发生之前，写入周期寄存器不会使计数寄存器更新为新的周期值 （在终端计数之后）。 由于在计数为 0 时已达到终端计数，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。 以下等式将 PWM 的输出与周期寄存器中的输入时钟和周期值相关联。

TOUT = （周期值 +1）/FCLOCK

$$FOUT = FCLOCK / (周期值 + 1)$$ **Equation 1**

其中，*F*OUT 为 PWM 的输出频率，*T*OUT 为 PWM 的输出周期，*F*CLOCK 为输入时钟的频率，*周期值*为输入的周期的值。

PWM 在停止时，将其输出置为低。 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，此比较器会针对 PulseWidth 寄存器的值来测试计数寄存器的值，从而执行 “ 小于 ”(Less Than) 或 “ 小于或等于 ”(Less Than Or Equal) 测试 （具体取决于使用器件编辑器选定的选项）。 在比较过程完成之后的时钟上升沿上，PWM 会将比较置为高电平有效真值。 PulseWidth 值和周期之间的比率会设置输出波形的占空比。 占空比可使用此等式进行计算。

对于脉冲宽度值 (PulseWidthValue) < 周期值 (PeriodValue)：

Equation 2

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

对于脉冲宽度值 (PulseWidthValue) >= 周期值 (PeriodValue)

占空比 (DutyCycle) = 100%

下表根据周期、脉冲宽度 (PulseWidth) 和比较操作的设置总结了某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	脉冲宽度 (PulseWidth) 寄存器值	脉冲宽度高电平时间与周期的比率
0	无需关注	> 0	1.0
0	≤	0	1.0
0	<	0	0.0
> 0	≤	0	1/ （周期 +1）

周期寄存器值	Compare Type	脉冲宽度 (PulseWidth) 寄存器值	脉冲宽度高电平时间与周期的比率
> 0	<	0	0.0
周期 = 脉冲宽度 (PulseWidth)	≤	周期 = 脉冲宽度 (PulseWidth)	1.0
周期 = 脉冲宽度 (PulseWidth)	<	周期 = 脉冲宽度 (PulseWidth)	周期 / （周期 +1）
脉冲宽度值 (PulseWidthValue) > 周期	无需关注	脉冲宽度值 (PulseWidthValue) > 周期	1.0

可以通过器件编辑器或在运行时使用 API 来设置脉冲宽度 (PulseWidth) 寄存器的值。 在终端计数之前，未以周期寄存器缓冲计数寄存器的方式提供脉冲宽度 (PulseWidth) 寄存器的任何缓冲。 因此，更改脉冲宽度 (PulseWidth) 寄存器会影响在下一个时钟周期上的比较输出，而不是遵循终端计数。 这样可产生具有多个脉冲的周期。

在 CY8C29/27/24/22/21xxx, CY8C23x33, CY7C64215/603xx, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8CTMG300, CY8CTST300, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx 器件系列中，PWM 用户模块提供终端计数信号作为辅助输出。 在用于从周期寄存器加载计数寄存器的终端计数之后的时钟周期中，出现上升沿时，将置位此高电平有效信号。

可以对中断进行编程，使其发生在终端计数上或比较变为真时。 在输出信号出现上升沿时比较器输出触发中断，而在输出信号的下降沿之前的半个时钟周期时，终端计数会触发中断。 通过使用器件编辑器设置此选项。 在运行时使用计数器 API 完成对中断的启用或禁用。 必须在计数器的中断激发之前启用全局中断。

当修改脉冲宽度 (PulseWidth) 寄存器时务必要小心，因为其值（与当前计数值结合使用）决定着 PWM 的输出状态。 要阻止可能发生提前将输出信号置于低电平和潜在短时脉冲，必须在使用中断检测到终端计数条件之后，修改脉冲宽度 (PulseWidth) 寄存器。

对于需要更快占空比更新间隔的应用程序，可将 PWM 的输出路由到其状态被轮询的引脚中。 在检测到输出从高电平切换到低电平之后，即可更新脉冲宽度 (PulseWidth)。 请注意，如果脉冲宽度 (PulseWidth) 可导致 “ 比较真值条件 ”(compare true condition)，则会在下一时钟上输出置于高电平。

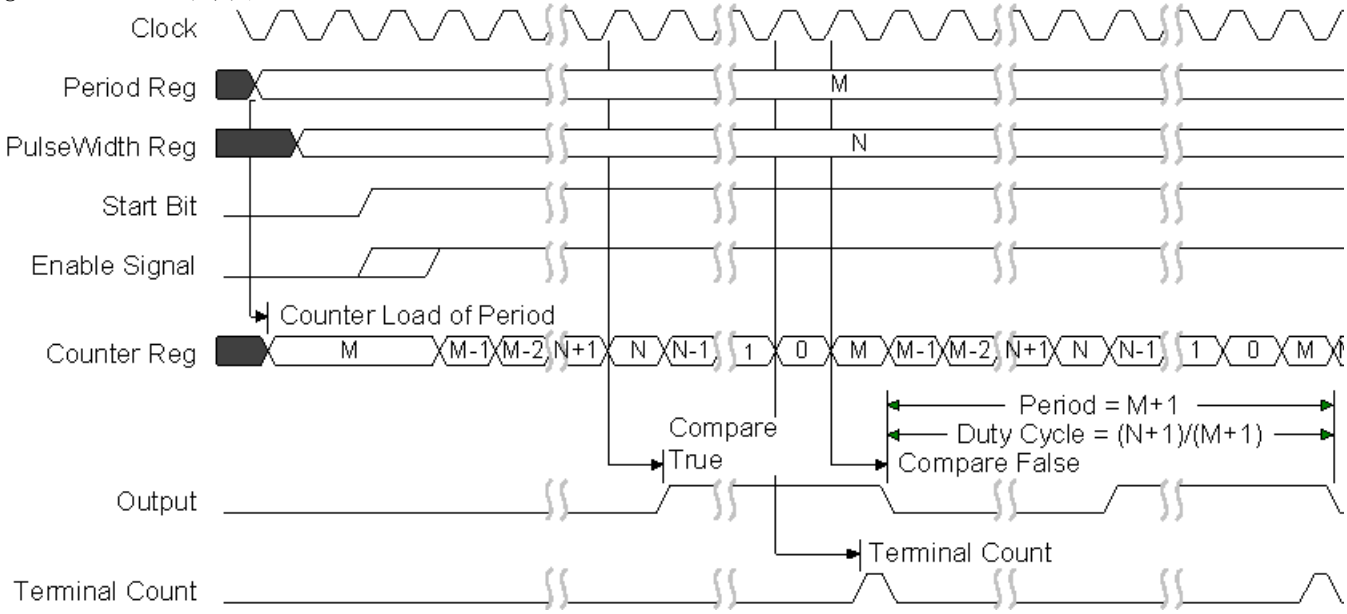
必须在十分小心的情况下获取计数寄存器值。 读取计数寄存器会导致其内容栓锁入脉冲宽度 (PulseWidth) 寄存器中。 这样会导致输出占空比更改。

如果您需要 “ 动态 ”(on-the-fly) 读取计数寄存器，则可以调用 ReadCounter() API 函数。 此函数会临时禁用时钟，保存脉冲宽度 (PulseWidth) 寄存器内容，读取计数寄存器，读取脉冲宽度 (PulseWidth) 寄存器，恢复脉冲宽度 (PulseWidth) 寄存器，然后恢复时钟。 请参见 “ 应用程序编程接口 ” 一节中有关 ReadCounter() 函数的说明，以获取可能的副作用信息。

时序

PWM 操作可以被置为打通和关断，或由器件全局总线特性路由到 PWM 的外部引脚来计时。

Figure 2. PWM 时序图



直流和交流电气特性

Table 2. PWM 直流和交流电气特性

参数	典型值	限制	单位	条件和注释
$FOutput_{max}$	--	24[1]	MHz	5.0V 和 48 MHz 输入时钟
	--	12[2]	MHz	3.3V 和 24 MHz 输入时钟

电气特性说明

1. 如果通过全局总线路由输出，则频率会限制在最大 12 MHz 以内。
2. 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

PWM 使用一个数字 PSoC 模块。该模块具有符号名称，在放置后由器件编辑器显示该名称。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 PWM 寄存器。下表中提供了由各种宽度使用的模块名称。

Table 3. PWM 符号 PSoC 模块名称

PSoC 模块	8-Bit PWM
1	PWM8

参数和资源

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。

启用

可以从 16 个源其中的某个源中选择 “ 启用 ” 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。

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, CY7C64215/603xx, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8CTMG300, CY8CTST300, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx 系列成员中显示。

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, CY8C23x33, CY7C64215/603xx, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8CTMG300, CY8CTST300, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx 系列成员中显示。

周期

此参数设置计数器的周期。 PWM8 的容许值介于 0 到 255 之间。 PWM16 的容许值介于 0 到 $2^{16}-1$ 之间。 此周期将加载到周期寄存器中。 PWM16 的有效输出波形周期为周期计数加 + 1。 可使用 API 修改此值。

脉冲宽度

设置 PWM 输出的脉冲宽度。 容许值介于 0 到周期值之间。 可使用 API 修改此值。

InterruptType

此参数用于设置中断触发类型。 可以设置中断，使其在输出信号的上升沿或计数寄存器的终端计数上触发。 单独的寄存器可以独自启用中断。

CompareType

此参数设置比较函数类型 “ 小于 ”(Less Than) 或 “ 小于或等于 ”(Less Than or Equal To)。

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, CY8C23x33, CY7C64215/603xx, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8CTMG300, CY8CTST300, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx 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。 此参数的正确数值必须由下表决定。

ClockSync 值	使用说明
Sync to SysClk	此设置值适用于小于 24 MHz 的任何 24 MHz (SysClk) 衍生输入时钟源。 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必须使用此值来确保发生正确的同步。
Sync to SysClk*2	此设置可以适用于小于 48 MHz 的任何基于 48 MHz (SysClk*2) 的输入时钟。
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 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。

InvertEnable

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。 InvertEnable 仅适用于 PSoC 器件的 CY8C29/27/24/22/21xxx, CY8C23x33, CY7C64215/603xx, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8CTMG300, CY8CTST300, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx 系列。

中断生成控制

以下两项参数 InterruptAPI 和 IntDispatchMode 只能在 PSoC Designer 内将 “ 启用中断生成控制 ” 复选框选中后进行访问。 可以在以下菜单下找到此复选框： **项目 > 设置 > 此复选框位于 “ 项目 ”>“ 设置 ”>“ 器件编辑器 ”。.**

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 ” 以生成中断处理程序和中断矢量表条目。 选择 “ 禁用 ” 可不生成中断处理程序和中断矢量表条目。 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” 参数会导致固件只在最初载入一个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。 本部分指明了每个函数的接口以及由 “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。 选择此 “ 寄存器易失 ” 策略旨在提高效率，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。

PWM8_PERIOD

说明：

表示器件编辑器中为 PWM8 的 Period 字段选择的值。 该值的范围介于 0 到 255 之间。

PWM8_PULSE_WIDTH

说明：

表示器件编辑器中为 PWM8 的 PulseWidth 字段选择的值。 该值的范围介于 0 到 255 之间。

PWM8_EnableInt

说明：

启用中断模式运行。

C 原型：

```
void  PWM8_EnableInt(void);
```

汇编：

```
lcall  PWM8_EnableInt
```

参数：

无

返回值：

无

副作用：

此函数可能会更改 A 和 X 寄存器。

PWM8_DisableInt

说明：

禁用中断模式运行。

C 原型：

```
void  PWM8_DisableInt(void);
```

汇编：

```
lcall  PWM8_DisableInt
```

参数：

无

返回值：

无

副作用：

此函数可能会更改 A 和 X 寄存器。

PWM8_Start

说明：

启动 PWM8 用户模块。 如果使能输入为高电平，则计数器会开始递减计数。

C 原型：

```
void  PWM8_Start(void);
```

汇编：

```
lcall  PWM8_Start
```

参数：

无

返回值：

无

副作用：

此函数可能会更改 A 和 X 寄存器。

PWM8_Stop

说明：

停止计数器运行。

C 原型：

```
void  PWM8_Stop(void);
```

汇编：

```
lcall  PWM8_Stop
```

参数：

无

返回值：

无

副作用：

输出复位成低电平，且写入到周期寄存器会导致计数器寄存器更新为新的周期值。 此函数可能会更改 A 和 X 寄存器。

PWM8_WritePeriod

说明：

将周期值写入周期寄存器。 如果 PWM8 停止或当计数器达到零计数，则立即将周期值从周期寄存器传输到计数器寄存器。

C 原型：

```
void  PWM8_WritePeriod(BYTE bPeriod);
```

汇编：

```
mov   A, [bPeriod]
lcall  PWM8_WritePeriod
```

参数：

bPeriod：bPeriod 值的范围为 0 至 255，并且会传递到累加器中。

返回值：

无

副作用：

此函数可能会更改 A 和 X 寄存器。

PWM8_WritePulseWidth

说明：

将脉冲宽度值写入脉冲宽度 (PulseWidth) 寄存器。

C 原型：

```
void  PWM8_WritePulseWidth(BYTE bPulseWidth);
```

汇编：

```
mov   A, [bPulseWidth]
lcall  PWM8_WritePulseWidth
```

参数：

bPulseWidth：bPulseWidth 值的范围为 0 至周期值，并且会传递到累加器中。

返回值：

无

副作用：

在计数器处于活动状态时写入脉冲宽度 (PulseWidth) 寄存器，从而更改输出的占空比。 这样可能导致输出短时脉冲或无意中更改。 此函数可能会更改 A 和 X 寄存器。

PWM8_bReadPulseWidth

说明：

读取脉冲宽度 (PulseWidth) 寄存器。

C 原型：

```
BYTE  PWM8_bReadPulseWidth();
```

汇编：

```
lcall  PWM8_bReadPulseWidth
mov   [bPulseWidth], A
```

参数：

无

返回值：

脉冲宽度值存储在脉冲宽度 (PulseWidth) 寄存器，并且会在累加器内返回。

副作用：

此函数可能会更改 A 和 X 寄存器。

PWM8_bReadCounter

说明：

读取计数器寄存器。

请注意，此函数适用于必须 “ 动态 ” 读取计数器寄存器的应用程序，会产生一些副作用。

C 原型：

```
BYTE  PWM8_bReadCounter();
```

汇编：

```
lcall  PWM8_bReadCounter
mov   [bCounter], A
```

参数：

无

返回值：

返回计数器寄存器值，并且会在累加器内返回。

副作用：

要读取 PWM8 计数器寄存器，则必须临时修改脉冲宽度 (PulseWidth) 寄存器。这样可能会导致 PWM8 计数器寄存器运行受一个或多个计数影响而延迟。 此外，这样可能会导致无意中断的情况发生。 此函数可能会更改 A 和 X 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。 显示的周期值和比较值都与基本值 “ 相差 1”(off-by-1)，因为寄存器是从零开始的，即零是递减计数循环的终端计数。 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。 当在 PWM8.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。

以下汇编语言源代码说明了 API 的使用。

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  Function:  GenerateOneThirdDutyCycle
;  Description:
;    This sample shows how to create a 33% duty cycle output pulse.
;    The clock selected should be 24 times the required period.  The
;    comparator operation is specified to be "Less than or Equal".
;
;  Parameters: none
;  Returns:    none
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

include "PWM8.inc" ; include the PWM8 API include file

GenerateOneThirdDutyCycle:
    mov   A, 23                ; set the period to be 24 counts of the clock
    call  PWM8_WritePeriod
    mov   A, 7                 ; set Pulse Width to generate a 33% duty cycle
    call  PWM8_WritePulseWidth
    call  PWM8_DisableInt      ; ensure that interrupts are disabled
    call  PWM8_Start           ; start the PWM8 - counter will start to
    ret                        ; count when the enable input is asserted high
```

同一代码用 C 语言表示如下：

```
/* include the Counter8 API header file  */
#include  "PWM8.h"

/* function prototype */
void GenerateOneThirdDutyCycle(void);

/* Divide by eight function */
void GenerateOneThirdDutyCycle(void)
{
    /* set period to eight clocks */
    PWM8_WritePeriod(23);

    /* set pulse width to generate a 33% duty cycle */
    PWM8_WritePulseWidth(7);

    /* ensure interrupt is disabled */
    PWM8_DisableInt();

    /* start the PWM8! */
    PWM8_Start();
}
```

配置寄存器

除非另有说明，否则本部分中提供的寄存器规范适用于所有 PSoC 器件系列。

8-bit PWM 使用名为 PWM8 的单个数字 PSoC 模块。 通过 7 个寄存器对每个模块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域。 这些寄存器的符号名称在用户模块实例的 C 语言和汇编语言接口文件 （“.h” 和 “.inc” 文件）中定义。

Table 4. 函数寄存器，组 1 CY8C26/25xxx

模块 / 位	7	6	5	4	3	2	1	0
PWM8	0	0	1	Compare Type	Interrupt Type	0	0	1

Table 5. 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
PWM8	数据反相	BCEN	1	Compare Type	Interrupt Type	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。 CompareType 标志指示比较函数是设置为 “ 小于或等于 ”(Less Than or Equal) 还是 “ 小于 ”(Less Than)。InterruptType 标志确定是通过比较事件还是终端计数触发中断。 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 6. 输入寄存器，组 1

模块 / 位	7	6	5	4	3	2	1	0
PWM8	启用				时钟			

使能在 16 个源其中的某个源中选择数据输入。 Clock 从 16 个源中的一个源选择时钟输入。 这两个参数都是在器件编辑器中设置的。

Table 7. 输出寄存器，组 1 CY8C26/25xxx

模块 / 位	7	6	5	4	3	2	1	0
CNTR8	0	0	0	0	0	OutEnable	OutputSelect	

Table 8. 输出寄存器，组 1 CY8C29/27/24/22/21xxx 和 CY8CLED04/08/16

模块 / 位	7	6	5	4	3	2	1	0
CNTR8	AuxClk		AuxEnable	AuxSelect		OutEnable	OutputSelect	

器件编辑器中用户模块 “ClockSync” 参数决定 AuxClk 位的值。 尽管命名类似，AuxEnable 和 AuxSelect 位却与 OutEnable 和 OutSelect 位域有关。 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
PWM8	计数							

“ 计数 ” 是指 PWM8 递减计数器。 它可以使用 PWM8 API 读取。

Table 10. 周期寄存器 (DR1)，组 0

模块 / 位	7	6	5	4	3	2	1	0
PWM8	周期							

“ 周期 ” 包含周期值，此周期值在启用时或满足终端计数条件时加载到计数器寄存器中。 可在器件编辑器和 PWM8 API 中对其进行设置。

Table 11. 比较寄存器 (DR2)，组 0

模块 / 位	7	6	5	4	3	2	1	0
PWM8	脉冲宽度							

PulseWidth 保留用于生成输出的脉冲宽度值。 可在器件编辑器和 PWM8 API 中对其进行设置。

Table 12. 控制寄存器 (CR0)，组 0

模块 / 位	7	6	5	4	3	2	1	0
PWM8	0	0	0	0	0	0	0	启动

设定 “ 启动 ” 时表明启用了 PWM8。 此参数使用 PWM8 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。
2.60	DHA	增加了对 CY8C21x12 器件的支持。

Note PSoC Designer 5.1 在所有的用户模块数据表中提供版本历史记录。 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。