16-Bit PWM 死区发生器基本介绍 PWMDB16 V 2.5

资源	PSoC® 模块			API 存储空间 （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, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8CTMA140, CY8C21x45, CY8C22x45, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx						
16-bit	3	0	0	44	0	1

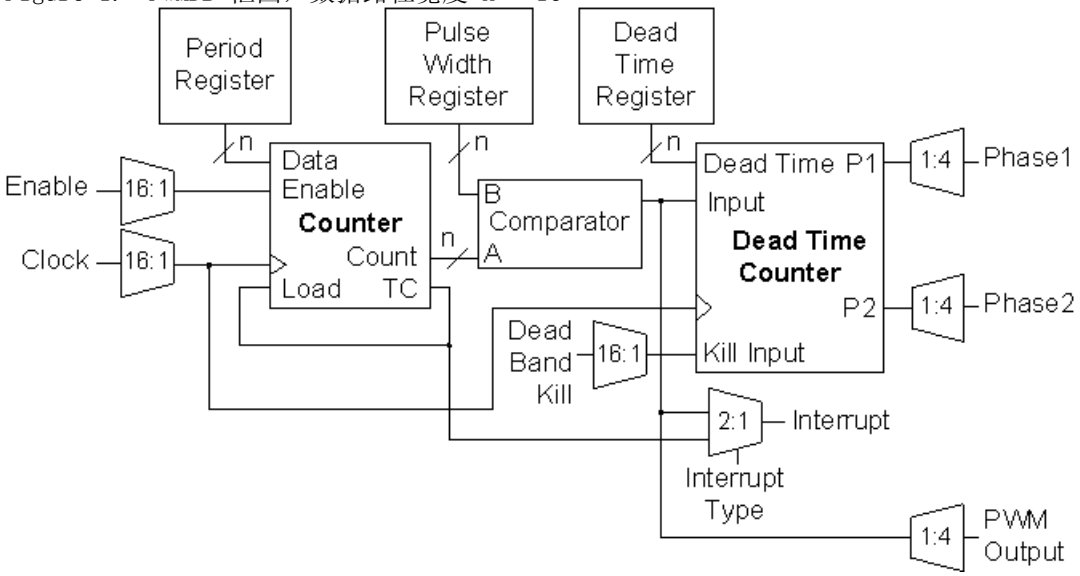
如需一个或多个使用此用户模块且完全配置的功能性示范工程，请转到 www.cypress.com/psocexampleprojects.

功能和概述

- 16-bit 带 8-bit 死区发生器的通用脉冲宽度调制器 (PWM)，分别使用两个或三个 PSoC 模块
- 相位 1 和相位 2 负遮盖输出跟踪 PWM 生成信号的频率
- 可编程占空比
- 可编程死时间
- 死区非同步停止输入将相位 1 和相位 2 输出置为低电平
- 计数器时钟频率高达 48 MHz
- 在 PWM 生成信号的上升沿触发或者基于计数器终端计数的中断选项

16-bit PWMDB 用户模块是组合了一个 8-bit 死区发生器的脉冲宽度调制器。 脉冲宽度调制器给死区发生器提供一个可编程周期和脉冲宽度输入信号。 死区发生器用和输入信号相同频率的可编程停滞时间输出两个负覆盖信号。 置位死区非同步停止输入后，它会将相位 1 和相位 2 输出信号置为低电平。 可以从多个源中选择时钟以及启用信号。 可以将相位 1 和相位 2 输出信号路由至外部引脚端口或者全局输出总线，以供其他用户模块内部使用。 可以对中断进行编程，使其在脉冲宽度调制器输出的上升沿和下降沿上有效触发。

Figure 1. PWMDB 框图，数据路径宽度 n = 16



功能描述

PWMDB 用户模块利用了三个数字 PSoC 模块，前两个模块提供 PWM 总分辨率中的 8 位。 PWM16_LSB 和 PWM16_MSB 这两个模块采用了 16-bit 脉冲宽度调制器，并配有可编程周期和脉冲宽度。 脉冲宽度调制输出信号载入到第三个 PSoC 模块 DB8。 DB8 实现带有可编程死时间的死区发生器。 相位 1 和相位 2 这两个输出信号会提供 PWMDB16 输出。

控制寄存器启动并停止 PWMDB 的 PWM 和 DB8 组件。 停止时写入周期寄存器会导致将新的周期寄存器值复制到计数器寄存器中。 停止时写入死时间寄存器会导致将新的死时间寄存器值载入到死时间计数器寄存器中。 PWMDB 停止时，PWM 输出和 DB8 的相位 1 和相位 2 输出均置于低电平。

高电平有效使能信号关断 PWMDB。 置于低电平时，PWM 和 DB8 PSoC 模块实际上被禁用，因而不可操作。 PWM 输出保持为当前状态，因而阻止 DB8 修改其输出。 不修改当前寄存器的内容，通过置位使能信号可以继续进行操作。

PWMDB 的 16-bit PWM 和 DB8 组件使用相同的输入时钟。

脉冲宽度调制器

启动并启用时，PWM 在时钟的每个上升沿递减计数器寄存器。 在计数器寄存器终端计数之后的时钟沿上，从周期寄存器中重新加载计数器寄存器。 随时可以将周期寄存器修改为新的周期值。 周期寄存器值为参数，可以通过器件编辑器或在运行时使用 API 设置。

PWM 的输出周期实际为编入周期寄存器中的周期值加一。

Equation 1

$$OutputPeriod = PeriodValue + 1$$

生成波形的占空比取决于周期以及脉冲宽度值的关系。 脉冲宽度寄存器中的值可以确定在周期中的哪些时候，输出会被设置为高电平。 在每个时钟上，PWM 将计数器和脉冲宽度寄存器中的值进行比较。 当计数值 “ 等于或小于 ” 周期值时，在下一个时钟中，输出将会被设置为高电平。 当出现自动重新载入周期时，意味着计数器寄存器和脉冲宽度寄存器比较失败，在下一时钟中，输出将会被设置成低电平。

可以通过下列方式计算占空比。

Equation 2

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

如果周期与脉冲宽度数值相等，则输出将永远保持高电平。 脉冲宽度值的范围是：零至周期寄存器中载入的周期值。 脉冲宽度寄存器值为参数，可以通过器件编辑器或在运行时使用 API 设置。

可以对中断进行编程，使其在 PWM 输出的上升沿或者计数器寄存器的终端计数条件下出现。 终端计数条件位于输出信号下降沿之前的半个时钟周期。 可使用器件编辑器设置中断选项。 启用或禁用中断使用 API 在运行时完成。

死区发生器 (Dead Band Generator)

在输入信号 （PWM8 或 PWM16 输出）的每个沿中，将重复下列操作：

Rising Edge （上升沿）

- 在下一个时钟周期的上升沿中，将相位 2 信号复位成低电平。
- 使用死时间寄存器值加载死时间计数器寄存器。
- 在输入时钟的每个上升沿中，递减死时间计数器寄存器，直至其达到终端计数。 相位 1 会随后在时钟的下一个下降沿中被设置成高电平。

Falling Edge （下降沿）

- 在下一时钟周期的上升沿中，将相位 1 信号复位成低电平。
- 使用死时间寄存器值加载死时间计数器寄存器。
- 在输入时钟的每个上升沿中，递减死时间计数器寄存器，直至其达到终端计数。 相位 2 会随后在时钟的下一个下降沿中被设置成高电平。

相位 1 和相位 2 跟踪从 PWM 中接收到的输入信号的频率。 相位 1 跟踪输入信号的占空比，再减去死时间。 相位 2 跟踪输入信号的反相周期，再减去死时间。

输入信号的每个相位的实际死时间如下。

Equation 3

$$DeadTime = ClockPeriod \times (DeadTime + 1)$$

死时间寄存器必须加载为一个 8-bit 值。 此值的范围必须从零到 PWM 周期寄存器值减去二、PWM 脉冲宽度寄存器值减去二，或 255 三者中的最小值。

死时间寄存器值为参数，可以通过器件编辑器或在运行时使用 API 设置。

置为高电平时，死区非同步停止输入将相位 1 和相位 2 输出置为低电平。 此信号只影响相位 1 和相位 2 信号的输出关断，而并不影响死时间计数器寄存器。 当把死区非同步停止输入释放 （即设成低电平）时，第一个适用的相位输出会产生小于死时间时钟计数而大于等于一的时序抖动。 此时，死区发生器会与脉冲带宽调制输入同步。 即第一个输出脉冲会加长。

如果 PWMDB 输出必须与应用中生成的 PWM 输入同步，则当死区非同步停止输入解除激活时请采取以下步骤 （当您发现死区非同步停止输入被置于高电平时，可采用同样的操作）：

1. 通过调用 Stop() API 函数停止 PWMDB。
2. 调用 WriteDeadTime() API 函数重写死时间周期。
3. 检测到死区解除激活时，再启动 PWMDB 。

系列支持 CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C28x45, CY8CTMG300, CY8CTST300, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx 三种非同步停止输入模式。 在所有模式中，非同步停止输入信号总是会以异步方式把输出变为逻辑值 “0”。 各种模式的区别在于死区处理如何重启。

1. **同步重启模式**：当非同步停止输入被置为高电平时，内部状态为复位状态并且原始死区周期会重新载入至计数器。 当非同步停止输入保持在高电平时，输入的 PWM 参照沿将被忽略。 当非同步停止输入降为低电平时，下一个输入的 PWM 参照沿会重启死区处理。 使用占空比为 100% 的 PWM 时，则没有输入的参照沿，并且当非同步停止输入降为低电平时不会重启死区处理。 请参阅第 5 页的同步重启非同步停止输入模式示例图。
2. **异步重启模式**：当非同步停止输入被置为高电平时，内部状态不会受影响。 当非同步停止输入降为低电平时，输出就会被还原，但最少禁用时间会在 0.5 至 1.5 时钟周期之间。 请参阅第 5 页的异步重启非同步停止输入模式示例图。

3. **禁用模式：** 禁用模式无特定时序。 模块已被禁用，用户必须在固件中重新启用此功能，以继续处理。

Figure 2. 同步重启非同步停止输入模式

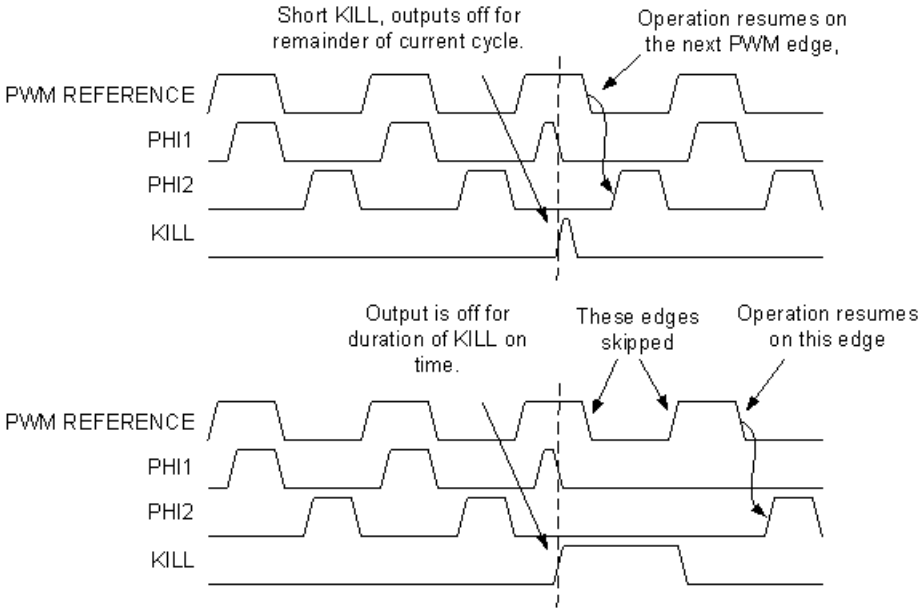
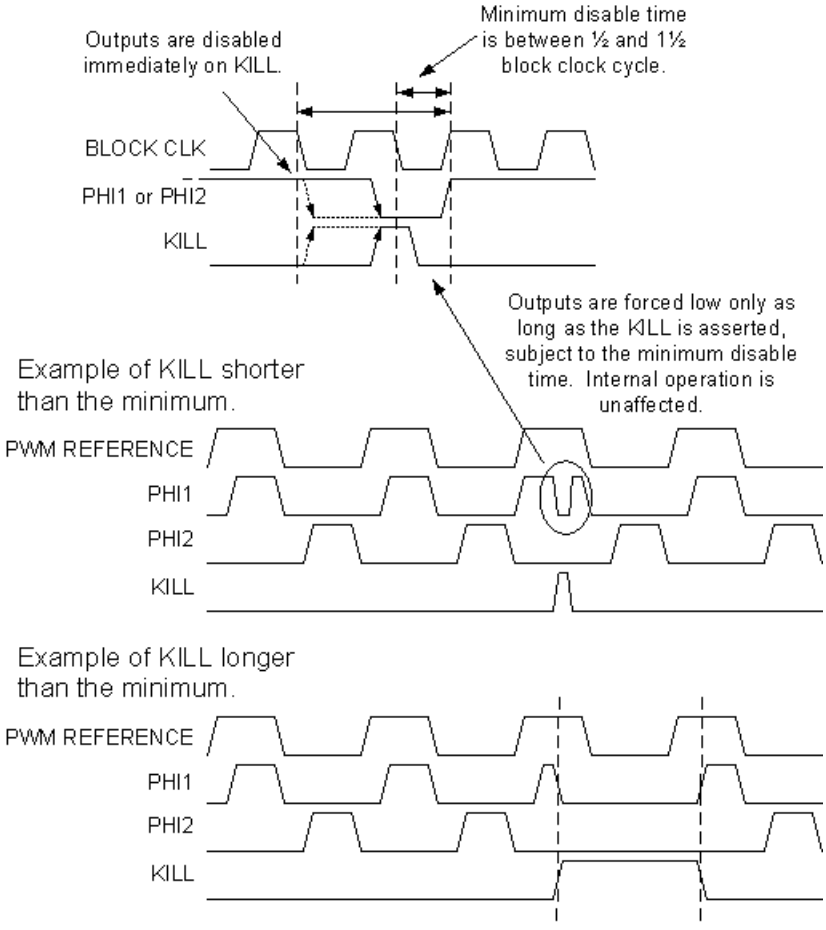


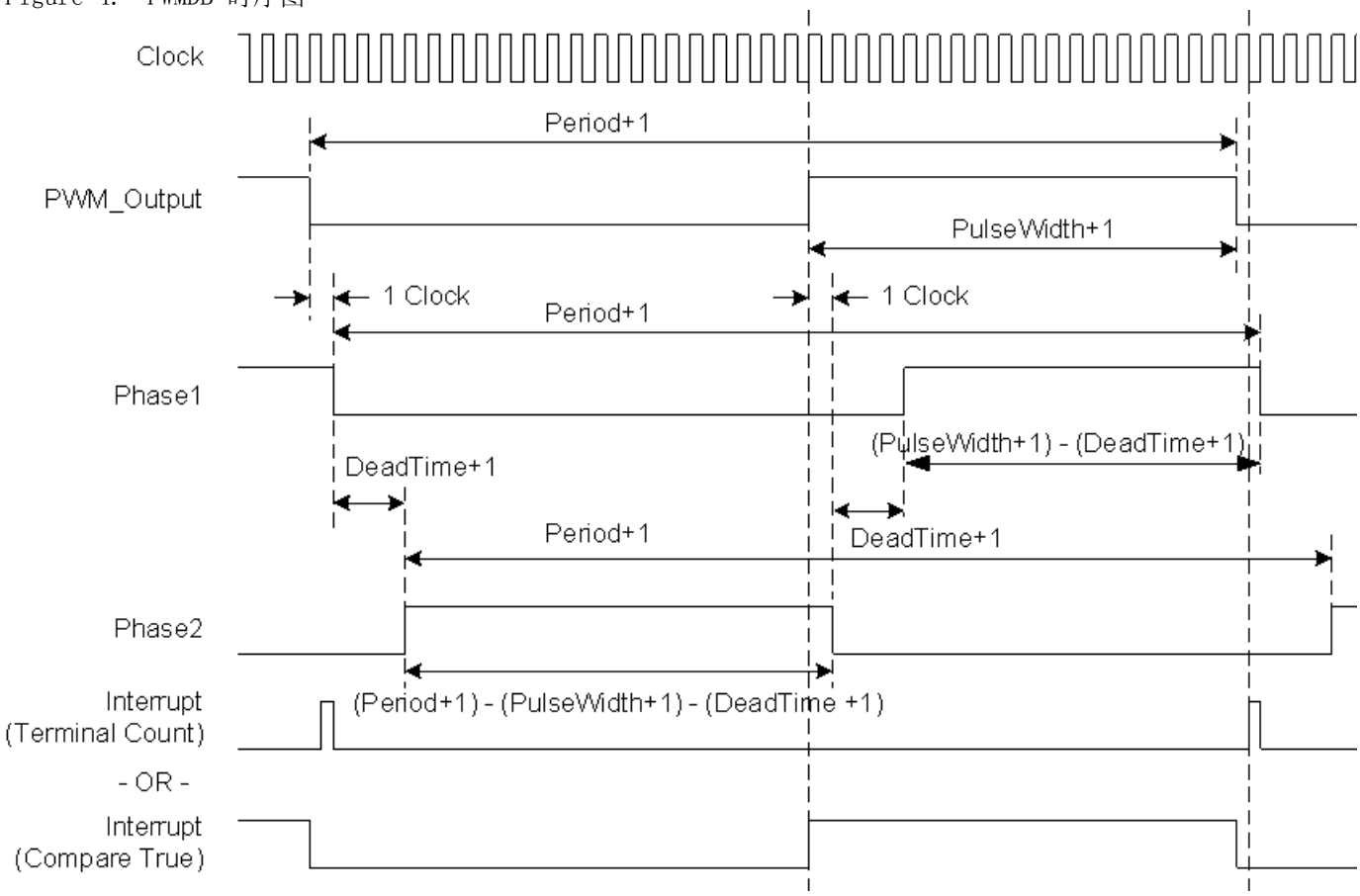
Figure 3. 异步重启非同步停止输入模式



时序

PWMDB 操作可以打开或关断，或由路由至 PWMDB 的外部引脚来计时，路由通过器件全局总线特性来完成。全局总线频率限制为最大 12 MHz。

Figure 4. PWMDB 时序图



直流和交流电气特性

Table 1. PWMDB 直流和交流电气特性

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

电气特性说明

1. 若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

8-bit PWMDB 耗用两个 PSoC 数字模块，16 位耗用三个。 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。 每个模块都具有符号名称，在放置期间和放置后由器件编辑器显示该名称。 API 使用用户分配的实例名称和模块名称来限定所有寄存器的名称，以便于通过 API 包含文件直接访问 PWMDB 寄存器。 下表中提供了由各种宽度使用的模块名称。

Table 2. 映射 PSoC 模块的符号名

PSoC 模块数	16-Bit PWMDB
1	PWM16_LSB
2	PWM16_MSB
3	DB8

参数和资源

时钟

可以从多个源中选得 “ 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。 脉冲宽度调制器和死区发生器使用同一时钟源。 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

可以从多个源中选得 “ 使能 ”(Enable) 参数。 来自全局输入和输出的外部输入会自动同步到器件的内部 24 MHz 振荡器。

Period (周期)

此参数可以设置 PWM 计数器的周期。 8-bit PWM 允许的数值范围为 0 至 255，16 位 PWM 允许的数值范围为 0 至 2^{16}-1。 此周期将加载到周期寄存器中。 PWM 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

此参数设置 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 PWM 信号的上升沿或在 PWM 计数器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。

PWMOutput

可以把此输出参数路由至四个全局输出总线之一。 如非必须，则建议避免路由此信号 (为其他输出节省全局资源)。

DeadTime

此参数设置 DB8 输出的死时间计数。 8-bit 数值，范围为零至以下值中的最小值： “PWM 周期 ”(PWM Period) 参数减 2、“PWM 脉冲宽度 ”(PWM Pulse Width) 参数值减 2 或 255。

相位 1

可以把此输出参数路由至四个全局输出总线之一。

相位 2

可以把此输出参数路由至四个全局输出总线之一。

死区非同步停止输入

可以从多个源中选得此参数。 当该参数被置为高电平时，相位 1 和相位 2 输出则被控制在低电平。

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。 由于各种数据路径优化， CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C28x45, CY8CTMG300, CY8CTST300, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx 在 PSoC 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 ”(Clock) 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。

死区非同步停止输入模式

此参数来自三个非同步停止输入模式之一，这三个非同步停止输入模式为同步重启非同步停止输入、禁用非同步停止输入以及异步非同步停止输入。 有关详细信息，请参见 “ 死区发生器 ” 一节。

反转死区非同步停止输入

此参数可以让用户反转传输进来的死区非同步停止输入信号。

反转使能

此参数可以让用户反转传输进来的 “ 使能 ”(Enable) 信号。

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

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。 本部分具体说明了每个函数对应的接口以及 “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。

对于大型储存器模式的器件， 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。

PWMDB16_PERIOD

说明：

表示器件编辑器中为 PWMDB16 的 “ 周期 ” 字段选择的值。 该值的范围介于 0 到 65535 之间。

PWMDB16_PULSE_WIDTH

说明：

表示器件编辑器中为 PWMDB16 的 “ 脉冲宽度 ” 字段选择的值。 该值的范围介于 0 到 65535 之间。

PWMDB16_EnableInt

说明：

启用中断模式运行。

C 原型：

```
void  PWMDB16_EnableInt(void);
```

汇编：

```
lcall  PWMDB16_EnableInt
```

参数：

None

返回值：

None

副作用：

此函数可能会更改 A 和 X 寄存器。

PWMDB16_DisableInt

说明：

禁用中断模式运行。

C 原型：

```
void  PWMDB16_DisableInt(void);
```

汇编：

```
lcall  PWMDB16_DisableInt
```

参数：

None

返回值：

None

副作用：

此函数可能会更改 A 和 X 寄存器。

PWMDB16_Start

说明：

同时启动脉冲宽度调制器和死区发生器 PSoC 模块。PWM 周期寄存器被加载到计数器寄存器，并启动 PWM16 时钟。 若输入使能处于高电平，则计数器会开始递减计数。

C 原型：

```
void  PWMDB16_Start(void);
```

汇编：

```
lcall  PWMDB16_Start
```

参数：

None

返回值：

None

副作用：

此函数可能会更改 A 和 X 寄存器。

PWMDB16_Stop

说明：

禁用 PWM16 和 DB8 PSoC 块。

C 原型：

```
void  PWMDB16_Stop(void);
```

汇编：

```
lcall  PWMDB16_Stop
```

参数：

None

返回值：

None

副作用：

此函数可能会更改 A 和 X 寄存器。

PWMDB16_WritePeriod

说明：

把周期值写入 PWM 周期寄存器。

C 原型：

```
void  PWMDB16_WritePeriod(WORD wPeriod);
```

汇编：

```
mov   A, [wPeriod+1]
mov   X, [wPeriod]
lcall  PWMDB16_WritePeriod
```

参数：

周期值介于 0 到 2^{16}-1 之间。 最高有效位在 X 寄存器中传递，而最低有效位在累加器中传递。

返回值：

None

副作用：

此函数可能会更改 A 和 X 寄存器。

PWMDB16_WritePulseWidth

说明：

把脉冲宽度值写入 PWM 脉冲宽度寄存器。

C 原型：

```
void  PWMDB16_WritePulseWidth(WORD wPulseWidth);
```

汇编：

```
mov   A, [wPulseWidth+1]
mov   X, [wPulseWidth]
lcall  PWMDB16_WritePulseWidth
```

参数：

脉冲宽度值介于零到周期值之间。 最高有效位在 X 寄存器中传递，而最低有效位在累加器中传递。

返回值：

None

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。

PWMDB16_WriteDeadTime

说明：

把死时间计数值写入 DB8 死时间寄存器。

C 原型：

```
void  PWMDB16_WriteDeadTime(BYTE bDeadTime);
```

汇编：

```
mov   A, [bDeadTime]
lcall  PWMDB16_WriteDeadTime
```

参数：

脉冲宽度值的范围是 0 至周期值，在累加器内传递。

返回值：

None

副作用：

此函数可能会更改 A 和 X 寄存器。

PWMDB16_wReadPulseWidth

说明：

读取 PWM 脉冲宽度寄存器。

C 原型：

```
WORD  PWMDB16_wReadPulseWidth();
```

汇编：

```
lcall  PWMDB16_wReadPulseWidth
mov   [wPulseWidth], X
mov   [wPulseWidth+1], A
```

参数：

None

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。

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 PWMDB16.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:  GenerateFetDrive
;  Description:
;       This sample shows how to generate 20% under-lapped output signals.
;       The clock selected should be 30 times the required period.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

include "PWMDB16.inc"          ; include the PWMDB16 API include file

GenerateFetDrive:
    mov   A, 29                ; set the period to be 30 counts of the clock
    mov   X, 0
    call  PWMDB16_WritePeriod
    mov   A, 14                ; set the pulse width to create 50% duty cycle
    mov   X, 0
    call  PWMDB16_WritePulseWidth
    mov   A, 2                 ; set the dead time to 20% ->  (15*0.2)-1
    call  PWMDB16_WriteDeadTime
    call  PWMDB16_DisableInt   ; ensure that interrupts are disabled
    call  PWMDB16_Start        ; start the PWMDB16 - 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  "PWMDB16.h"

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

/* Generate Fet drive function*/
void GenerateFetDrive(void)
{
    /* set period to 30 clocks */
    PWMDB16_WritePeriod(29);
    /* set pulse width to generate a 50% duty cycle */
    PWMDB16_WritePulseWidth(14);
    /* set dead time to 20% -> (15*0.2)-1 */
    PWMDB16_WriteDeadTime(2);
    /* ensure interrupt is disabled */
    PWMDB16_DisableInt();

    /* start the PWM16! */
```

```
    PWMDB16_Start();
}
```

配置寄存器

16-bit PWMDB 使用三个数字 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的 “ 特性 ” 值和作为带简要描述的指定的位域。 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。

PWM16 配置寄存器

Table 3. 函数寄存器，组 1

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	1
LSB	0	0	1	Compare Type （比较类型）	0	0	0	1

Compare Type 标志指出比较功能是设置为 “ 小于等于 ” 或 “ 小于 ”。 Interrupt Type 标志指明是基于捕获事件还是基于终端条件来触发中断。 在器件编辑器中设置参数。

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			
LSB	启用				时钟			

“ 使能 ”(Enable) 从多个源中选择数据输入。 “ 时钟 ”(Clock) 从多个源中选择输入时钟。 在器件编辑器中设置参数。

Table 5. 输出寄存器，组 1

模块 / 位	7	6	5	4	3	2	1	0
MSB	0	0	0	0	0	输出使能 (Out Enable)	OutputSelect	
LSB	0	0	0	0	0	0	0	0

Output Enable 标志用于表示输出已启用。 Output Sel 标志用于表示输出 PWM16 经由的位置。 这两个参数都是在器件编辑器中设置的。

Table 6. 计数寄存器 (DR0)，组 0

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

“ 计数 ”(Count) 是 PWM16 MSB 和 LSB 递减 PWM。 可以使用 PWM16 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)							
LSB	Period(LSB)							

“ 周期 ”(Period) 保留周期值的 MSB 和 LSB，周期值根据使能或终端计数条件加载到计数器寄存器。 可在器件编辑器和 PWM16 API 里对其进行设置。

Table 8. 脉冲宽度寄存器 (DR2)，组 0

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

PulseWidth 保留用于生成比较事件的脉冲宽度值的 MSB 和 LSB。 可在器件编辑器和 PWM16 API 里对其进行设置。

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[1]
LSB	0	0	0	0	0	0	0	Start/Stop

设置 Start/Stop 表示启用 PWM16。 此参数使用 PWM16 API 进行修改。

“ 启动 / 停止 ”(Start/Stop) 由串联 PSoC 模块中的 LSB 控制寄存器控制，设置为零。

DB8 配置寄存器

Table 10. 模块 DB8: 寄存器函数

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	1	0	1

Table 11. 模块 DB8: 寄存器输入

位	7	6	5	4	3	2	1	0
值	死区非同步停止输入				时钟			

“ 死区非同步停止输入 ”(Dead Band Kill) 从多个源中选择数据输入。 “ 时钟 ”(Clock) 从多个源中选择输入时钟。 这两个参数都是在器件编辑器中设置的。

Table 12. 模块 DB8: 寄存器输出

位	7	6	5	4	3	2	1	0
值	0	0	相位 2 输出使能	相位 2 输出选择		相位 1 输出使能	相位 1 输出选择	

“ 相位 1 输出使能 ”(Phase1 Output Enable) 标志用于表示相位 1 输出已启用。 “ 相位 1 输出选择 ”(Phase1 Output Select) 指定 DB8 的相位 1 输出路由到的位置。 “ 相位 2 输出使能 ”(Phase2 Output Enable) 标志用于表示相位 2 输出已启用。 “ 相位 2 输出选择 ”(Phase2 Output Select) 指定 DB8 的相位 2 输出路由到的位置。 这些参数均在器件编辑器中设置。

Table 13. 模块 DB8: 死时间计数器寄存器 DR0

位	7	6	5	4	3	2	1	0
值	死时间计数器							

“死时间计数器”(Dead Time Counter) 是指 DB8 递减计数器。

Table 14. 模块 DB8: 死时间寄存器 DR1

位	7	6	5	4	3	2	1	0
值	死时间							

“死时间”(Dead Time) 保留死时间计数值。 可使用 PWMDB8 API 对其进行修改。

Table 15. 模块 DB8: 寄存器 DR2

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

未使用该寄存器。

Table 16. 模块 DB8: 控制寄存器 CR0

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	Start/Stop

若设置了“启动 / 停止”(Start/Stop)，则表示 DB8 已启用。 此参数使用 PWMDB8 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。