# 增强型 16 位死区 PWM 数据表 PWMDB16L V 1.0

Copyright © 2009-2010 Cypress Semiconductor Corporation. All Rights Reserved.

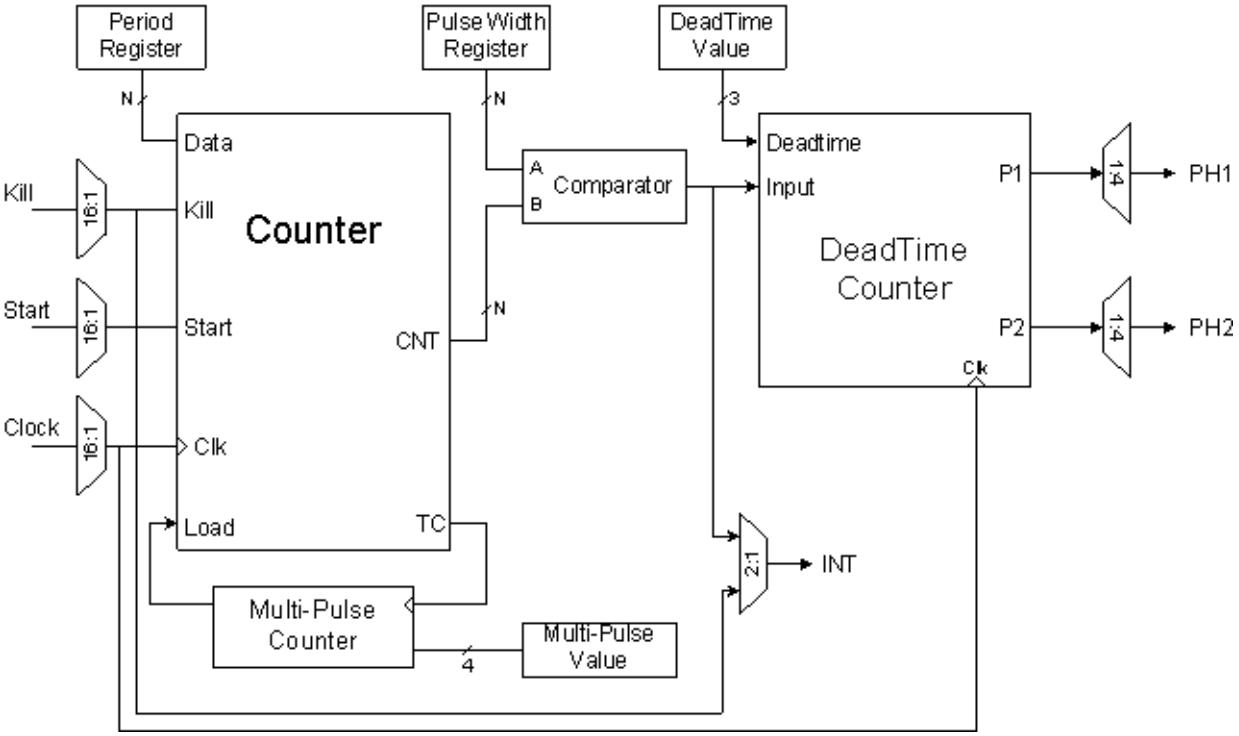
| 资源 | PSoC® 模块 | | | API 存储器 （字节） | | 引脚 （每个外部 I/O） |
|---|---|---|---|---|---|---|
| | 数字 | 模拟 CT | 模拟 SC | 闪存 | RAM | |
| CY8C21/22x45、CY8C28x45、CY8C28xxx、CY8CTMA30xx | | | | | | |
| 16-bit | 2 | 0 | 0 | 116 | 0 | 2 |

## 功能和概述

- 带有 8 位死区发生器的 16 位通用脉冲宽度调制器 (PWM)，占两个 PSoC 模块
- 将计数器和死区功能结合在一个数字模块中，提供有限度死区宽度调整
- 不互相遮盖的相位 1 (PH1) 和相位 2 (PH2) 输出对 PWM 生成信号的频率进行跟踪
- 可编程的占空比
- 可编程的死时间
- 死区非同步停止输入将相位 1 (PH1) 和相位 2 (PH2) 输出控制在低水平
- 拥有单触发 / 多触发功能
- 计数器时钟达 48 MHz
- 在 PWM 相位 1 信号或非同步停止输入的上升沿触发中断选项

增强型 16 位死区 PWM 用户模块 (PWMDB16L) 是 PWM16 用户模块的增强版。 PWMDB16L 支持每个数字模块中的死区， 并改进了一些功能，例如单触发和多触发功能。 脉冲宽度调制器为死区发生器提供可编程周期以及脉冲宽度输入信号。 死区发生器输出两个不互相遮盖信号以及与输入信号频率相同的可编程死时间。 添加死区非同步停止输入后，死区非同步停止输入会将相位 1 和相位 2 输出信号控制在低水平。 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相位 1 和相位 2 输出信号路由至外部引脚端口以及内部用户的全局输出总线。 可以对中断进行编程，使其在 PWM 相位 1 信号的上升沿或 PWM 模块非同步停止输入的上升沿有效触发。

Figure 1. PWMDB16L 框图




## 功能说明

PWMDB16L 用户模块占用两个数字 PSoC 模块，由一个周期寄存器、一个同步递减数器、一个脉冲宽度寄存器以及一个死时间计数器组成。

PWM16_LSB 和 PWM16_MSB 这两个模块采用了 16 位脉冲宽度调制器，并配有可编程周期和脉冲宽度。 脉冲宽度调制输出信号会载入到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。

控制寄存器可启动并终止 PWMDB16L。 如果终止时正在向周期寄存器写入值，则新周期寄存器的值将复制到计数器寄存器中。 如果终止时正在向 Control 寄存器的 “ 死时间 ” 位写入值，则将会加载新的死时间值。 当 PWMDB16L 被终止时，相位 1 和相位 2 输出将被置为低电平。

PWMDB16L 由高电平有效的开始信号终止。 被置为低电平时，PWMDB16L PSoC 模块实际上被禁用。 激活开始信号可让操作继续进行，并且无需修改当前寄存器内容。

在启动并启用后，每当出现时钟上升沿，PWM 便减少计数器寄存器。 在计数器寄存器最终计数后的时钟沿上，计数器寄存器会从周期寄存器中重新载入。 随时可以通过新周期值修改周期寄存器。 周期寄存器值是一个参数，可以使用器件编辑器或在运行时使用 API，来指定此参数。

PWM 的输出周期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，使其发生在 PWM 相位 1 信号的上升沿或 PWM 模块非同步停止输入的上升沿。 可通过使用器件编辑器设置中断选项。 在运行时，使用 API 禁用或启用中断。

在输入信号的每个沿中，将重复下列操作：

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。
- 从 Control1 寄存器的 “ 死时间 ” 位域中载入死时间值。
- 在输入时钟的每个上升沿中，死时间值都会减少，直至其达到最终计数。 相位 1 会随后在时钟的下一个下降沿中被设置成高电平。

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。
- 从 Control1 寄存器的 “ 死时间 ” 位域中载入死时间值。
- 在输入时钟的每个上升沿中，死时间值都会减少，直至其达到最终计数。 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循环，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**

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

使用 “ 死时间 ” 位时，必须载入一个 3 位的值。 因此死时间值的大小可以是 0、1、2、4、8、16 或 32 模块时钟。

死时间值为参数，可以通过器件编辑器或在运行时使用 API 设置。

死区非同步停止输入被置为高电平时，会使相位 1 和相位 2 输出保持在低电平。 此信号只影响相位 1 和相位 2 信号的输出关断，而不会影响死时间值。 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 函数终止 PWMDB16L。
2. 调用 WriteDeadTime() API 函数，重新编写死时间周期。
3. 在检测到死区解除激活时，启动 PWMDB16L。

支持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”。 三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。 请参阅下文同步重启非同步停止输入模式的示例图。
2. **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。
3. **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。 请参阅下文异步重启非同步停止输入模式的示例图。

Figure 2. 同步重启非同步停止输入模式

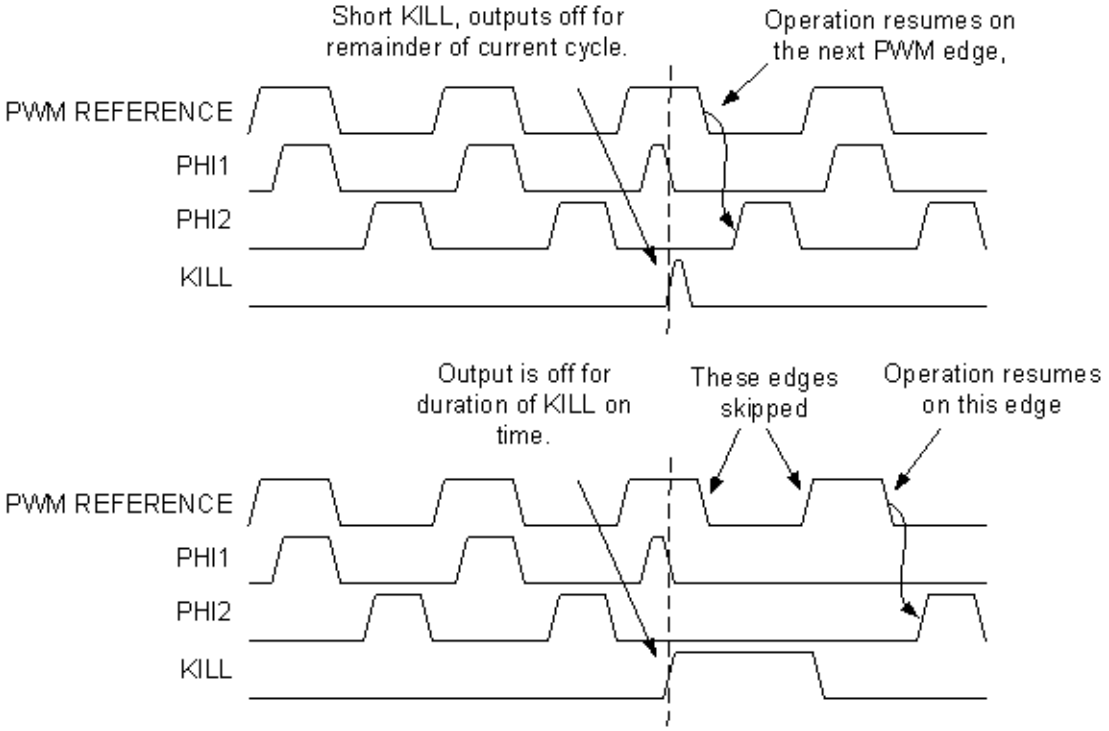
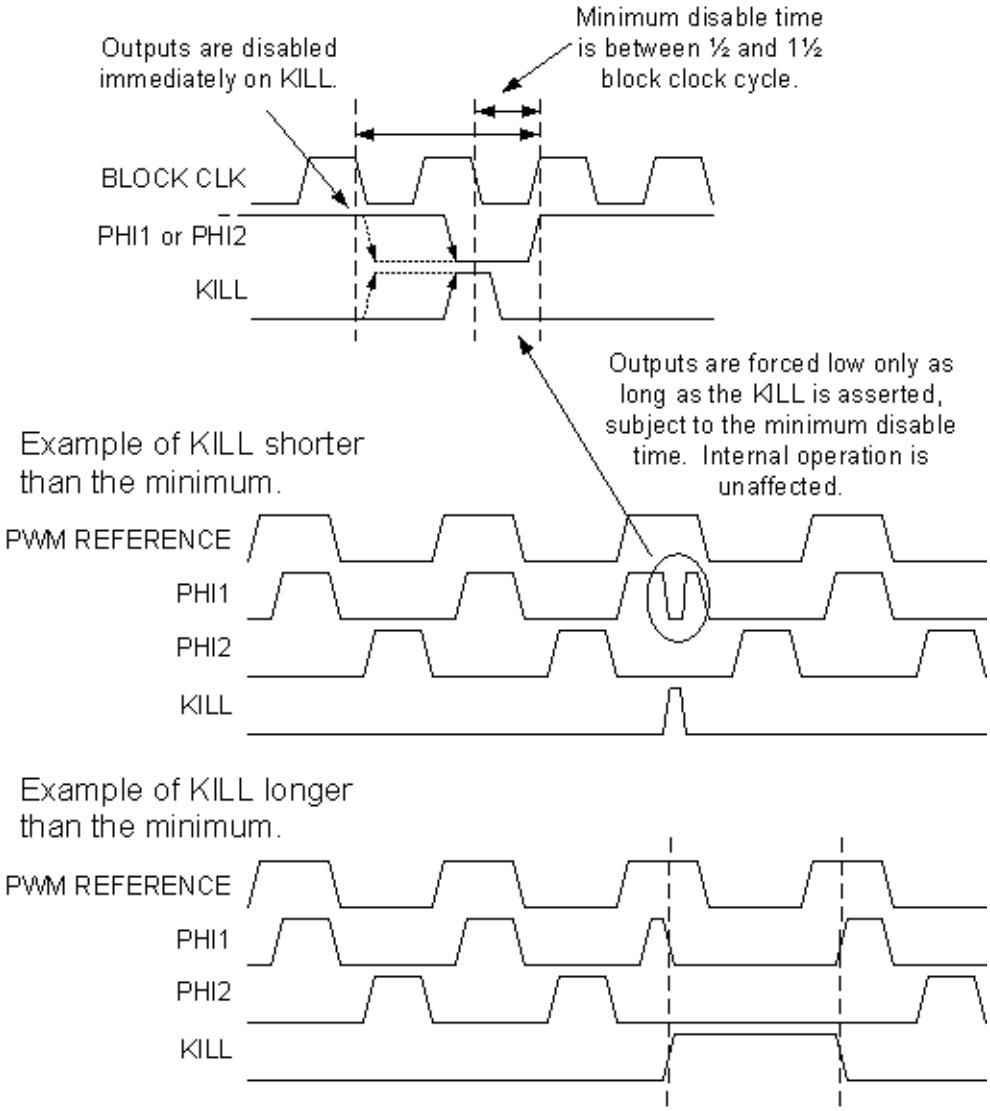


Figure 3. 异步重启非同步停止输入模式




除以下所列情况，PWMDB16L 功能与计数器功能相同：

- 无计数器门输入。 计数功能由不同子模式控制。
- PWMDB16L 的多触发模式被称为 PPG 模式，全称为可编程脉冲发生器 (Programmable Pulse Generator)。 要在 PPG 模式下启动 PWMDB16L，多触发寄存器应设置为非零值。 在最后触发后，此功能不会自动关闭， 只会停止计数。 启动硬件或软件 （向 “EN” 位写入 1）可恢复计数功能。 “ 开始 ” 处在高电平时进行最后触发不会停止计数。 将 “ 开始 ” 保持在高电平不会影响计数。
- 比较结果为 DR0>DR2，而不是 DR0<=DR2 或 DR0<DR2。 因此比较输出的波形会变为反向。
- 当正在运行 PWMDB16L 用户模式时，总会缓冲向 DR2 执行的写入操作。
- 无需像使用计数器功能时那样设置寄存器。
- 不可输出 TC。 比较输出不可直接输出，只能通过死区功能输出。

- 非同步停止输入模式将继续使用死区功能设置。
  - 同步重启非同步停止输入
  - 禁用非同步停止输入
  - 异步非同步停止输入
- 除非在禁用非同步停止输入模式下，否则非同步停止输入不会影响计数。 当在禁用非同步停止输入模式下激活非同步停止输入时，则将禁用该功能。

除以下所列情况，PWMDB16L 死区功能与死区功能相同：

- 无需从之前模块设置参照时钟输入。 参照时钟数据来自当前模块中计数器功能的比较输出。
- 死区宽度选择受限。 因此死区只可是 0、1、2、4、8、16、32 或 64 个模块时钟周期。 将死区设置为 0 个时钟周期，将禁用死区保护功能。
- 死区功能使用模块时钟。 计数器功能使用相同的时钟。
- 从非同步停止输入信号中，用户模块可以得到另一个中断源。

## 直流和交流电气特性

Table 1. PWMDB16L 直流和交流电气特性

| 参数 | 条件和注释 | 典型值 | 限值 | 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 模块的最快时钟为 24 MHz、3.3V 时钟。

## 放置

PWMDB16L 用户模块占用两块增强型数字模块。

## 参数和资源

### 时钟

PWMDB16L UM 的时钟可以由多个时钟源中的一个提供。 全局 I/O 总线可用于将时钟输入连接至外部引脚或其他 PSoC 模块生成的时钟功能。 48 MHz 时钟、CPU_32 kHz 时钟、一个分频时钟 （24V1 或者 24V2）、另一 PSoC 模块输出都可指定为时钟输入。 在模块中使用外部数字时钟时，为达到最高精度并使用睡眠操作应关闭行输入同步。

### 开始

“ 开始 ” 参数从多个源中选择开始信号源。 高电平信号输入将启用数字模块中的继续计数功能，而低电平信号输入则禁用计数功能且无需复位计数器。 输出不会受到输入信号的状态的影响。 例如，如果在解除激活信号时，输出处于高电平，则输出值仍会保持在高电平状态。

本信号选择参数相当于其他用户模块中占用数字模块的 “ 启用 ” 信号选择参数。 可以将开始信号理解为模块中的一个启用信号。 不应将 “ 开始 ” 参数与开始 API 功能混为一谈，后者用来以应用程序代码方式启用用户模块操作。

### 反向启动

此参数可以让你反置传输进来的开始信号。

### 周期

此参数可以设置 PWM 计数器的周期。 16 位的 PWM 可以选择从 0 至 $2^{16}$ - 1 的值。 周期已经载入周期寄存器。 PWM 有效周期为周期数加 1。 可以在运行时使用 API 修改其值。

### 脉冲宽度

此参数可以设置 PWM 输出的脉冲宽度。 数值范围为零至周期数。 可以在运行时使用 API 修改其值。 注意： 相位 2 的输出为直接输出。 相位 1 的输出已反转。

### 中断类型

此参数可以设置中断触发类型。 可以设置中断，使其在 PWM 相位 1 信号的上升沿或 PWM 模块非同步停止输入的上升沿中触发。 单独的寄存器可以独自启用中断。

### 死时间

使用此参数控制死区宽度。 死区可以是 0、1、2、4、8、16、32 或 64 个模块时钟周期。 将死区设置为 0 个时钟周期，将禁用死区保护功能。

### 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。

### 反转死区非同步停止输入

此参数可以让你反转传输进来的死区非同步停止输入信号。

### 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。 查看死区发生器的前一部分，获取更多信息。

### 时钟同步

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。 本参数的正确值应根据以下内容确定：

| 时钟同步值 | 说明 |
|---|---|
| 与 SysClk 同步 | 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。 例如：VC1, VC2, VC3 （当 VC3 由 SysClk 驱动时）、32 KHz，以及基于 SysClk 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 | 除非所导致的频率为 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 SysClk | 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 | 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，只有在生成中断是计数器的唯一应用时才推荐使用此选项。 应对在睡眠时仍然保持活动状态的模块进行此设置。 |

### PWM 边沿对齐

设置此参数后，对比输出会延迟半个时钟周期。 当使用 48 MHz 时钟作为模块时钟时，可以运用此参数取得更高的分辨率。

### 软件触发

此参数为软件触发启用控制位。 设置完成后，只有向软件中 PWMDB16L_CONTROL0_REG 的 Bit[0] 写入 1 时，才会触发 PWM 。 若 MultiShot[3:0] 不等于零，则 PWMDB16L 将在多触发模式下运行。 复位此参数后，PWM 模块只可以通过硬件触发。

## 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，以及内置文件所提供的相关常量。

每次放置用户模块时，都会为其分配一个实例名称。 默认情况下，PSoC Designer 会为指定项目中此用户模块的第一个实例分配 PWMDB16L_1。 可将该值更改为符合标识符语法规则的任意唯一值。 分配的实例名称成为每个全局函数名称、变量和常量符号的前缀。 为简便起见，在以下说明中将实例名称缩写为 PWMDB16L。

**Note** ** 在这里，如同所有用户模块 API 中的一样，A 和 X 寄存器的值可以通过调用 API 函数来更改。 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。 此 “ 寄存器易失 ” 策略是针对提高效率的目的选择，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。

### PWMDB16L_Start

**说明：**

同时启用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。 若输入开始值为高电平，则计数器会开始递减计数。

**C 原型：**

```
void PWMDB16L_Start(void);
```

**汇编：**

```
lcall PWMDB16L_Start
```

**参数：**

无

**返回值：**

无

**副作用：**

请参阅 API 一节开头的注意 **。

### PWMDB16L_Stop

**说明：**

禁用 PWMDB16L PSoC 模块。

**C 原型：**

```
void PWMDB16L_Stop(void);
```

**汇编程序：**

```
lcall PWMDB16L_Stop
```

**参数：**

无

**返回值：**

无

**副作用：**

请参阅 API 一节开头的注意 **。

### PWMDB16L_EnableInt

**说明：**

启用中断模式操作。

**C 原型：**

```
void PWMDB16L_EnableInt(void);
```

**汇编程序：**

```
lcall PWMDB16L_EnableInt
```

**参数：**

无

**返回值：**

无

**副作用：**

请参阅 API 一节开头的注意 **。

### PWMDB16L_DisableInt

**说明：**

禁用中断模式操作。

**C 原型：**

```
void PWMDB16L_DisableInt(void);
```

**汇编程序：**

```
lcall PWMDB16L_DisableInt
```

**参数：**

无

**返回值：**

无

**副作用：**

请参阅 API 一节开头的注意 **。

### PWMDB16L_WritePeriod

**说明：**

把周期值写入 PWMDB16L 周期寄存器。

**C 原型：**

```
void PWMDB16L_WritePeriod(WORD wPeriod);
```

**汇编程序：**

```
mov   A, [wPeriod+1]
mov   X, [wPeriod]
lcall PWMDB16L_WritePeriod
```

**参数：**

wPeriod 值的范围为 0 至 $2^{16}$ -1。 MSB 传递到 X 寄存器，LSB 传递到累加器。

**返回值：**

无

**副作用：**

请参阅 API 一节开头的注意 **。

### PWMDB16L_WritePulseWidth

**说明：**

把脉冲宽度值写入 PWM 脉冲宽度寄存器。

**C 原型：**

```
void PWMDB16L_WritePulseWidth(WORD wPulseWidth);
```

**汇编程序：**

```
mov   A, [wPulseWidth+1]
mov   X, [wPulseWidth]
lcall PWMDB16L_WritePulseWidth
```

**参数：**

wPulseWidth 为脉冲宽度值。 有效值范围为零至周期值。 MSB 传递到 X 寄存器，LSB 传递到累加器。

**返回值：**

无

**副作用：**

请参阅 API 一节开头的注意 **。

### PWMDB16L_WriteDeadTime

**说明：**

把死时间计数值 DeadTime[2:0] 写入 Control 寄存器的 “ 死时间 ” 位。

**C 原型：**

```
void PWMDB16L_WriteDeadTime(BYTE bDeadTime);
```

**汇编程序：**

```
mov A,[bDeadTime]
lcall PWMDB16L_WriteDeadTime
```

**参数：**

bDeadTime 为可以设置死时间的值。 有效值范围为 0 至 7。 值传递到累加器。

| 值 | 说明 |
|---|---|
| 0 | 无死时间 |
| 1 | 死时间为 1 个模块时钟 |
| 2 | 死时间为 2 个模块时钟 |
| 3 | 死时间为 4 个模块时钟 |
| 4 | 死时间为 8 个模块时钟 |
| 5 | 死时间为 16 个模块时钟 |
| 6 | 死时间为 32 个模块时钟 |
| 7 | 死时间为 64 个模块时钟 |

**返回值：**

无

**副作用：**

请参阅 API 一节开头的注意 **。

### PWMDB16L_TriggerMultiShot

**说明：**

设置 PWMDB16L“ 启用 ” 位或者设置 PWMDB16L_CONTROL0_REG. 中的 bit[0]

Note 将多触发寄存器设置为非零值时，PWMDB16L 用户模块会进入 PPG（可编程脉冲发生器）模式。 在最后触发时，会停止输出脉冲。 只有启动硬件或软件 （再次向 “EN” 位写入 1），才能恢复脉冲输出。

**C 原型：**

```
void PWMDB16L_TriggerMultiShot(void);
```

**汇编程序：**

```
lcall PWMDB16L_TriggerMultiShot
```

**参数：**

无

**返回值：**

无

**副作用：**

请参阅 API 一节开头的注意 **。

### PWMDB16L_SetTrigMode

**说明：**

设置 PWM 软件触发模式：SWT 或 PWMDB16L_CONTROL0_REG. 中的 Bit[1]

**C 原型：**

```
void PWMDB16L_SetTrigMode(BYTE bSWTMode);
```

**汇编程序：**

```
mov A,[bSWTMode]
lcall PWMDB16L_SetTrigMode
```

**参数：**

| 符号名 | 值 | 说明 |
|---|---|---|
| TRIGMODE_SOFTWARE_DISABLE | 0 | 禁用软件触发模式 |
| TRIGMODE_SOFTWARE_ENABLE | 1 | 启用软件触发模式 |

**返回值：**

无

**副作用：**

请参阅 API 一节开头的注意 **。

### PWMDB16L_SetEdgeAlign

**说明：**

设置 PWM 沿对齐模式：NPS 或 PWMDB16L_CONTROL0_REG. 中的 Bit[3]

**C 原型：**

```
void PWMDB16L_SetEdgeAlign(BYTE bEdgeAlign);
```

**汇编程序：**

```
mov A,[bEdgeAlign]
lcall PWMDB16L_SetEdgeAlign
```

**参数：**

| 符号名 | 值 | 说明 |
|---|---|---|
| EDGEALIGN_DISABLE | 0 | 正常模式 |
| EDGEALIGN_ENABLE | 1 | 完成时，对比输出将延迟半个时钟周期。 当使用 48 MHz 时钟作为模块时钟时，可以运用此方法取得更高的分辨率。 |

**返回值：**

无

**副作用：**

请参阅 API 一节开头的注意 **。

### PWMDB16L_WriteMultiShotCounter

**说明：**

把多触发计数值 ——MULTI_SHOT[3:0] 或 PWMDB16L_CONTROL1_REG. 中的 Bit[7:4]—— 写入多触发计数器寄存器

**C 原型：**

```
void PWMDB16L_WriteMultiShotCounter(BYTE bMultiShotCounter);
```

**汇编程序：**

```
mov A,[bMultiShotCounter]
lcall PWMDB16L_WriteMultiShotCounter
```

**参数：**

多触发计数器值范围为 0 至 15 并且会传递到累加器中

**返回值：**

无

**副作用：**

请参阅 API 一节开头的注意 **。

### PWMDB16L_wReadPulseWidth

**说明：**

读取 PWM 脉冲宽度寄存器。

**C 原型：**

```
BYTE PWMDB16L_wReadPulseWidth(void);
```

**汇编程序：**

```
lcall PWMDB16L_wReadPulseWidth
mov   [wPulseWidth], X
mov   [wPulseWidth+1], A
```

**参数：**

无

**返回值：**

脉冲宽度值储存在脉冲宽度寄存器中，并返回至累加器和 X 寄存器中。

**副作用：**

请参阅 API 一节开头的注意 **。

### PWMDB16L_ClearInt

**说明：**

为 UM 清除待处理中断。

**C 原型：**

```
void PWMDB16L_ClearInt(void);
```

**汇编：**

```
lcall PWMDB16L_ClearInt
```

**参数：**

无

**返回值：**

无

**副作用：**

请参阅 API 一节开头的注意 **。

## 固件源代码示例

在下例中，C 和汇编代码之间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一，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。 汇编程序和 C 语言编译器均通过将简单的单字节参数传输到 A 寄存器（而不是堆栈）来优化用户模块 API 的性能。 当在 PWMDBL16.h 文件中遇到 #pragma fascall 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。

下文为展示 API 使用方法的汇编源语言。

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  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 "PWMDB16L.inc"          ; include the PWMDB16L API include file

GenerateFetDrive:
    mov   A, 29               ; set the period to be 30 counts of the clock
    mov   X, 0
    lcall  PWMDB16L_WritePeriod
     mov   A, 14               ; set the pulse width to create 50% duty cycle
    mov   X, 0
    lcall  PWMDB16L_WritePulseWidth
    mov   A, 2                ; set the dead time to 20% ->  (15*0.2)-1
    lcall  PWMDB16L_WriteDeadTime
    lcall  PWMDB16L_DisableInt  ; ensure that interrupts are disabled
     lcall  PWMDB16L_Start       ; start the PWMDB16L - counter will start to
    ret
```

同一代码用 C 语言表示为：

```
/* include the PWMDB16L API header file  */
#include "PWMDB16L.h"
/* function prototype */
void GenerateFetDrive(void);
/* Generate Fet drive function*/
void GenerateFetDrive(void)
{
    /* set period to 30 clocks */
    PWMDB16L_WritePeriod(29);
    /* set pulse width to generate a 50% duty cycle */
    PWMDB16L_WritePulseWidth(14);
    /* set dead time to 20% -> (15*0.2)-1 */
    PWMDB16L_WriteDeadTime(2);
    /* start the PWM16L! */
    PWMDB16L_Start();
```

## 配置寄存器

通过寄存器对 PWMDB16L 用户模块进行个性化和参数化设置。 下文中的表格显示了常量寄存器值，以及已命名位域形式的参数。表格附带有简要描述。 寄存器符号名在用户模块的 C 语言和汇编语言接口文件（后缀名为 “.h” 和 “.inc” 的文件）中定义。

Table 2. PWMDB16L_FUNC_REG

| 模块 / 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| MSB | 0 | 0 | 1 | 死区非同步停止输入模式 | | 0 | 1 | 1 |
| LSB | 反转死区非同步停止输入 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

“ 反转死区非同步停止输入 ” 标志允许反转输入的死区非同步停止输入信号。 死区非同步停止输入模式可以从以下三个非同步停止输入模式中选择：同步重启非同步停止输入、禁用非同步停止输入或异步非同步停止输入。 在器件编辑器中设置参数。

Table 3. PWMDB16L_INPUT_REG

| 模块 / 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| MSB | 0 | 0 | 1 | 1 | Clock[3:0] | | | |
| LSB | DeadBandKill[3:0] | | | | Clock[3:0] | | | |

死区非同步停止输入从源中选择具有同一名称的输入信号，共有 16 个源可供选择。 器件编辑器中的用户模块 “ 死区非同步停止输入 ” 参数设置决定了它的值。 类似地，用户模块 “ 时钟 ” 参数设置决定了时钟的值。

Table 4. PWMDB16L_OUTPUT_REG

| 模块 / 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| MSB | 时钟同步 | | 1 | Phase2[1:0] | | 1 | Phase1[1:0] | |
| LSB | 时钟同步 | | 0 | 0 | 0 | 0 | 0 | 0 |

器件编辑器中的用户模块 “ 时钟同步 ” 参数决定了 “ 时钟同步 ” 位的值。 器件编辑器里的 “ 相位 1” 和 “ 相位 2” 参数可以路由到四个全局输出总线之一。

Table 5. PWMDB16L_COUNT_REG

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

“ 计数 ” 是指 PWMDB16L MSB 和 LSB 递减计数器。 可以使用 PWMDB16L API 读取。

Table 6. PWMDB16L_PERIOD_REG

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

“ 周期 ” 包含周期值的 MSB 和 LSB，此周期值在启动时或终端计数时加载到计数器寄存器中。 可在器件编辑器和 PWMDB16L API 里对其进行设置。

Table 7. PWMDB16L_PULSE_WIDTH_REG

| 模块 / 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| MSB | 脉冲宽度 (MSB) | | | | | | | |
| LSB | 脉冲宽度 (LSB) | | | | | | | |

“ 脉冲宽度 ” 包含可用来生成比较事件的脉冲宽度值的 MSB 和 LSB。 可在器件编辑器和 PWMDB16L API 里对其进行设置。

Table 8. PWMDB16L_CONTROL0_REG

| 模块 / 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| MSB | 0 | 0 | 0 | 0 | PWM 边沿对齐 | 中断类型 | 0 | 0 |
| LSB | 开始 [3:0] | | | | 0 | 中断类型 | 软件触发 | 启用 |

“ 开始 [3:0]” 是 “ 开始 ” 触发源输入选择的一个选项，是来自 16 个可能数字输入源的一个 16:1 复用器。 “ 中断类型 ” 位可设置中断触发类型。 这些参数只能在器件编辑器中设置。 “PWM 沿对齐 ” 位是 PWM 输出沿对齐选择位。 “ 软件触发 ” 位是软件触发启用位。 这些参数可以在器件编辑器和 PWMDB16L API 中设置。 设置 “ 启用 ” 表明已启用 PWMDB16L，清除 “ 启用 ” 表明已禁用 PWMDB8L。 可使用 PWMDB16L API 对其进行修改。

Table 9. PWMDB16L_CONTROL1_REG

| 模块 / 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| MSB | MULTI_SHOT[3:0] | | | | 0 | DeadTime[2:0] | | |
| LSB | 0 | 0 | 0 | 0 | 反向启动 | 0 | 0 | 0 |

“MULTI_SHOT[3:0]” 值是多触发计数器寄存器。 可使用 PWMDB16L API 对其进行修改。 “ 反转开始 ” 位允许反转输入的死区非同步停止输入信号。 可在器件编辑器中进行设置。 “ 死时间 ” 是指死区宽度控制。 它可以是 0、1、2、4、8、16、32，或 64 个模块时钟周期。 将 “ 死时间 ” 设置为零意味着没有死区保护。 可在器件编辑器和 PWMDB16L API 里对其进行设置。

# 版本历史记录

| 版本 | 创作者 | 说明 |
|---|---|---|
| 1.0 | DHA | 初始版本 |

**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。



Copyright © 2009-2010 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.