Enhanced 16-Bit Dead Band PWM Datasheet PWMDB16L V 1.10

Copyright © 2009-2014 Cypress Semiconductor Corporation. All Rights Reserved.

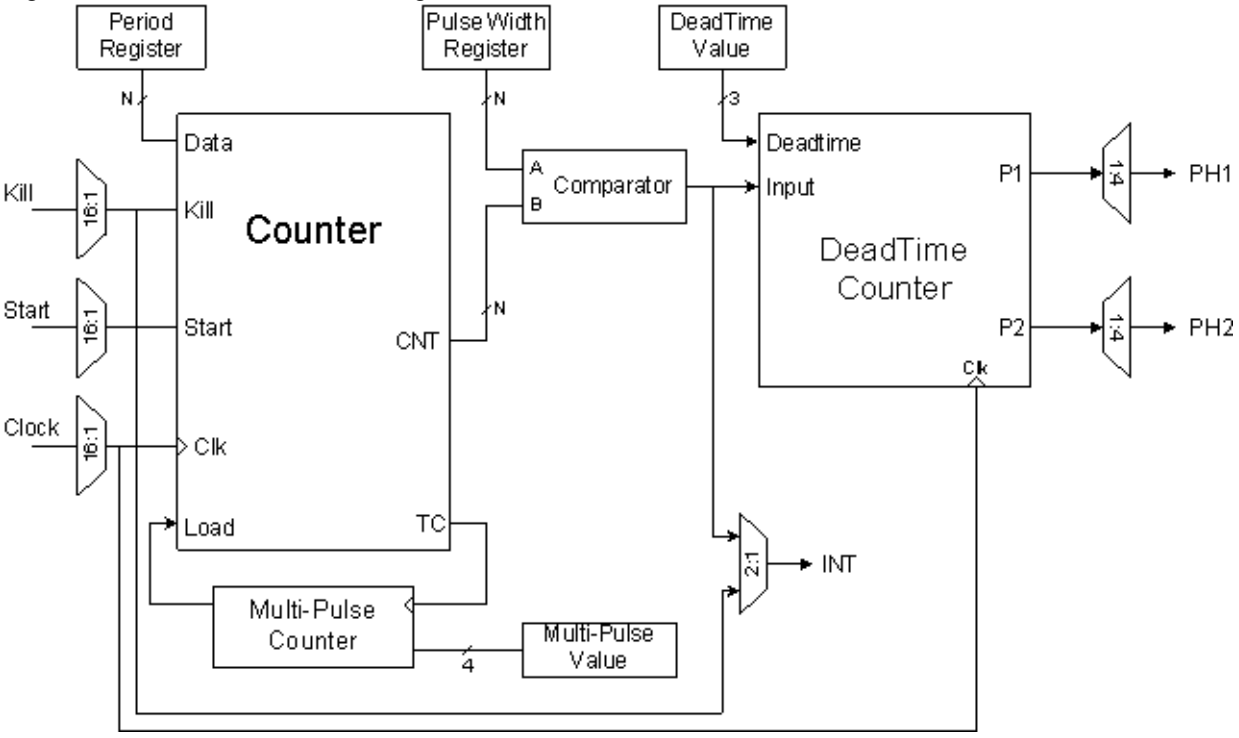
Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C21/22x45, CY8C28x45, CY8C28xxx, CY8CTMA140, CY8CTMA30xx						
16-bit	2	0	0	116	0	2

Features and Overview

- 16-bit general purpose pulse width modulator (PWM) with 8-bit dead band generator, occupies two PSoC blocks
- Combines the counter and deadband functions in one digital block with limited deadband width selection
- Phase1 (PH1) and Phase2 (PH2) under-lapped outputs track the frequency of the generated PWM signal
- Programmable duty cycle
- Programmable dead time
- Dead Band Kill input drives Phase1 (PH1) and Phase2 (PH2) outputs low
- Has one-shot/multi-shot feature
- Counter clocking up to 48 MHz
- Interrupt option triggered on rising edge of the PWM Phase1 signal or Kill input

The Enhanced 16-Bit Dead Band PWM User Module (PWMDB16L) is an enhanced version of the PWM16 User Module. The PWMDB16L supports the deadband in every digital block. It has improved features like one-shot and multi-shot. The pulse width modulator provides a programmable period and pulse width input signal to the dead band generator. The dead band generator outputs two underlapped signals, with programmable dead time at the same frequency as the input signal. When asserted, the Dead Band Kill input will drive the Phase1 and Phase2 output signals low. The clock and enable signals can be selected from several sources. The Phase1 and Phase2 output signals can be routed to the external pin ports or to the global output buses for internal use by other user modules. An interrupt can be programmed to effectively trigger on the rising edge of the PWM Phase1 signal or Kill input of the PWM module.

Figure 1. PWMDB16L Block Diagram



Functional Description

The PWMDB16L User Module employs two digital PSoC blocks and consists of a period register, a synchronous down counter, a pulse width register, and a dead time counter.

Two blocks, PWM16_LSB and PWM16_MSB, implement a 16-bit pulse width modulator with programmable period and pulse width. The pulse width modulated output signal is fed into the dead band generator with programmable dead time. The two output signals, Phase1 and Phase2, provide the PWMDB16 outputs.

The Control registers start and stop the PWMDB16L. Writing the Period register while stopped, causes the new Period register value to be copied to the Counter register. Writing the DeadTime bits of the Control1 register while stopped, causes the new DeadTime value to be loaded. While the PWMDB16L is stopped the Phase1 and Phase2 outputs are asserted low.

The PWMDB16L is gated by an active high Start signal. While asserted low, the PWMDB16L PSoC block is effectively disabled. Asserting the Start signal continues operation without modifying current register contents.

When started and enabled, PWM decrements the Counter register on each rising edge of the clock. On the clock edge that follows the Counter register's terminal count, the Counter register is reloaded from the Period register. The Period register can be modified with a new period value at anytime. The Period register value is a parameter that may be assigned using the Device Editor or at run time using the API.

The output period of the PWM is the period value programmed in the Period register, plus one.

Equation 1

$$\mathrm{OutputPeriod} = \mathrm{PeriodValue} + 1$$

The duty cycle of the generated waveform is defined by the relationship of the period and the pulse width values. The value in the PulseWidth register defines at what count within the period the output will be set high. On every clock, the PWM compares the values in the Counter and PulseWidth registers. When the count value is “Equal To or Less Than” the period value the output is set high on the following clock. When the automatic reload of the period occurs, the Counter and PulseWidth register comparison fails and the output is set low on the following clock.

The duty cycle can be computed as follows.

Equation 2

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

If the period and the pulse width values are equal, the output remains high indefinitely. The pulse width value may have a value from zero to the period value loaded in the Period register. The PulseWidth register value is a parameter that may be set using the Device Editor or at run time using the API.

An interrupt can be programmed to occur on the rising edge of PWM Phase1 signal or Kill input of the PWM module. The interrupt option can be set using the Device Editor. Enabling or disabling interrupts is done at run time with the API.

For each edge of the input signal, the following is repeated:

Rising Edge

- The Phase2 signal is reset low on the rising edge of the next clock cycle.
- The DeadTime value is loaded from the DeadTime bit field of the Control1 register.
- The DeadTime value is decremented on each rising edge of the input clock until it reaches the terminal count. Phase1 is then set high on the next falling edge of the clock.

Falling Edge

- The Phase1 signal is reset low on the rising edge of the following clock cycle.
- The DeadTime value is loaded from the DeadTime bit field of the Control1 register.
- The DeadTime value is decremented on each rising edge of the input clock until it reaches the terminal count. Phase 2 is then set high on the next falling edge of the clock.

Phase1 and Phase2 track the frequency of the input signal received from PWM. Phase1 tracks the duty cycle of the input signal, minus the dead time. Phase2 tracks the inverted cycle of the input signal, minus the dead time.

The effective dead time for each phase of the input signal is:

Equation 3

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

The DeadTime bits must be loaded with a 3-bit value. So the DeadTime value can be 0, 1, 2, 4, 8, 16, or 32 block clocks in size.

The DeadTime value is a parameter that may be assigned using the Device Editor or at run time using the API.

When asserted high, the Dead Band Kill input will drive the Phase1 and Phase2 outputs low. This signal only affects the output gating of Phase1 and Phase2 signals and not the DeadTime value. When the Dead Band Kill input is released (asserted low), the first applicable phase output may incur a jitter less than the DeadTime clock counts, but at least one. At this time, the dead band generator will be synced with the pulse width modulated input. This means that the first output pulse will be lengthened.

If the PWMDB output must be synchronized to the generated PWM input in your application, do the following when the Dead Band Kill input deasserts (repeat the same actions when you detect Dead Band Kill high assertion):

1. Stop the PWMDB16L by calling the Stop() API function.
2. Call the WriteDeadTime() API function to rewrite the Dead Time period.
3. Start the PWMDB16L upon the detection of the Dead Band de-assertion.

Three KILL modes are supported. In all cases, The KILL signal asynchronously forces the outputs to logic '0'. The difference between the three modes lies in how dead band processing is restarted.

1. **Synchronous Restart Mode**: When KILL is asserted high the internal state is held in reset and the initial dead band period is reloaded into the counter. While KILL is held high, incoming PWM reference edges are ignored. When KILL is negated, the next incoming PWM reference edge restarts dead band processing. See the Synchronous Restart Kill Mode Figure below.
2. **Asynchronous Restart Mode**: When KILL is asserted high the internal state is not affected. When KILL is negated, outputs are restored subject to a minimum disable time between one-half and one and one-half clock cycles. See the Asynchronous Restart Kill Mode figure (Figure 3).

3. **Disable Mode**: There is no specific timing associated with Disable Mode. The block is disabled and the user must re-enable the function in firmware to continue processing.

Figure 2. Synchronous Restart KILL Mode

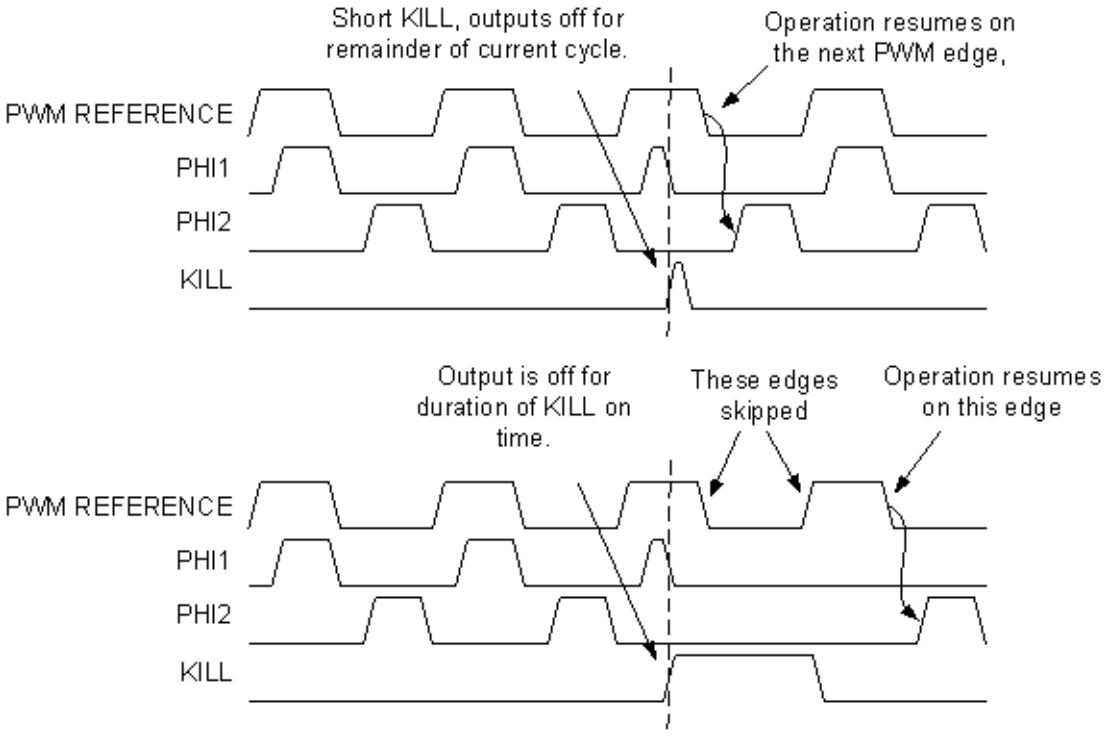
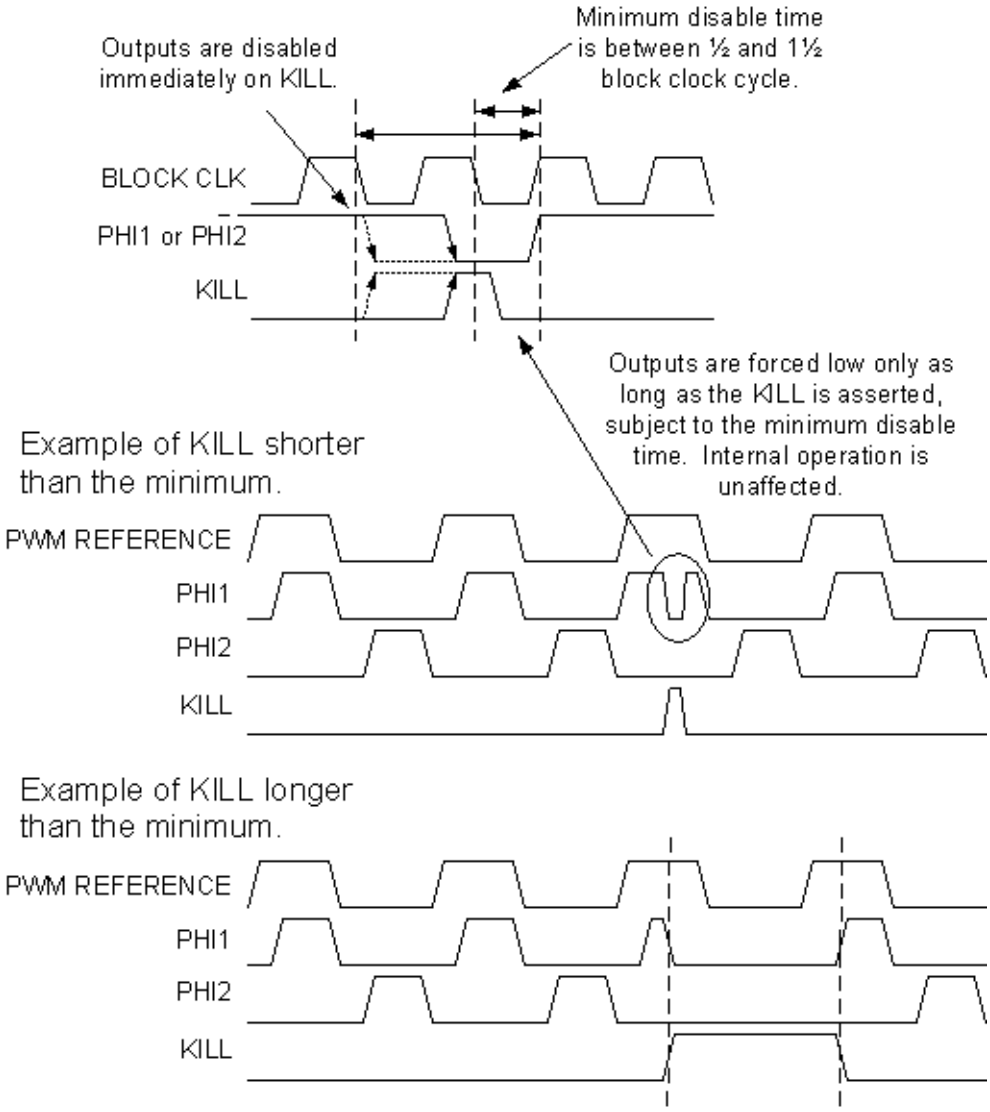


Figure 3. Asynchronous Restart Kill Mode



The PWMDB16L function is identical to the to the Counter function except for the following points:

- There is no counter gate input. Counting down is controlled by different sub modes.
- The multi-shot mode in PWMDB16L is called PPG mode (stands for Programmable Pulse Generator). To start PWMDB16L in the PPG mode the multi-shot register should be set to a non-zero value. The function will not go to disable mode at last-shot. It simply stops counting. Hardware or software start (writing 1 to the ‘EN’ bit) resumes counting. The last-shot with START held high will not stop counting. Holding START high will not affect the count.
- The comparison is DR0>DR2 instead of DR0<=DR2 or DR0<DR2. So the compare out waveform is reversed.
- Writing to DR2 is always buffered when the PWMDB16L User Module is running.
- You do not need to set the register as you do in the counter function.
- You can not output TC. Compare out can not be output directly (it must be output through the dead-band function).

- The KILL modes will follow the deadband function setting.
  - SyncRestartKill
  - DisableKill
  - AsyncKill
- KILL will not affect the counter except in the DisableKill mode. The whole function is disabled when KILL is asserted in DisableKill mode.

The PWMDB16L deadband function is identical to the DeadBand function except for the following points:

- You do not need to set the reference clock input from the previous block. The reference clock comes from the counter function's compare out in the current block.
- Deadband width selections are limited. The deadband can only be 0, 1, 2, 4, 8, 16, 32, or 64 block clock cycles. Setting the deadband to 0 clock cycles disables deadband protection.
- The Deadband function uses the block clock. The Counter function runs off of the same clock.
- The user module has an additional interrupt source from the KILL signal.

DC and AC Electrical Characteristics

Table 1. PWMDB16L DC and AC Electrical Characteristics

Parameter	Conditions and Notes	Typical	Limit	Units
$FOutput_{max}$	5.0V and 48 MHz input clock	--	24[1]	MHz
	3.3V and 24 MHz input clock	--	12[2]	MHz

Electrical Characteristics Notes

1. If the output is routed via the global buses, then the frequency is constrained to a maximum of 12 MHz.
2. Fastest clock available to PSoC blocks is 24 MHz at 3.3V operation.

Placement

The PWMDB16L User Module occupies two enhanced digital blocks.

Parameters and Resources

Clock

The PWMDB16L UM is clocked by one of a number of sources. The Global I/O buses may be used to connect the clock input to an external pin or a clock function generated by a different PSoC block. The 48 MHz clock, the CPU_32 kHz clock, one of the divided clocks (24V1 or 24V2), or another PSoC block output can be specified as the clock input. When using an external digital clock for the block, the row input synchronization should be turned off for best accuracy, and sleep operation.

Start

The Start parameter selects a Start signal source from a number of sources. A high signal input enables continuous counting in the digital block, while a low signal input disables counting without resetting the counter. The output is not affected by the state of the input signal. For example, if the output is high when the signal is de-asserted, the output remains in a high state.

This signal selection parameter is equivalent to the Enable signal selection parameter of many other user modules that occupy digital blocks. The Start signal can be thought of as an Enable signal of the block. This Start parameter should not be confused with the Start API function, which is used to start the user module operation in application code.

InvertStart

This parameter allows you to invert the incoming Start signal.

Period

This parameter sets the period of the PWM counter. The 16 bit PWM allows you to select values between 0 and $2^{16} – 1$. The period is loaded into the Period register. The effective period of the PWM is the period count + 1. You can modify the value at run time using the API.

PulseWidth

This parameter sets the pulse width of the PWM output. Allowed values are between zero and the period value. You can modify the value at run time using the API. Note: Phase2 output is direct. Phase1 output is inverted.

InterruptType

This parameter sets the interrupt trigger type. The interrupt can be set up so that it triggers on the rising edge of the PWM Phase1 signal or Kill input of the PWM module. A separate register independently enables the interrupt.

DeadTime

This parameter is used to control the deadband width. The deadband can be 0, 1, 2, 4, 8, 16, 32, or 64 block clock cycles. Setting the deadband to 0 clock cycles disables deadband protection.

Phase1

This output parameter may be routed to one of four global output buses.

Phase2

This output parameter may be routed to one of four global output buses.

DeadBandKill

This parameter is selected from one of a number of sources. When it is asserted high, Phase1 and Phase2 outputs are driven low.

InvertDeadBandKill

This parameter allows you to invert the incoming DeadBand Kill signal.

DeadBandKillMode

This parameter is selected from one of three Kill modes, SyncRestartKill, DisableKill, or AsyncKill. See the earlier section on the Dead Band Generator for more information.

ClockSync

This parameter may be used to control clock skew and ensure proper operation when reading and writing PSoC block register values. Appropriate values for this parameter should be determined from the following:

ClockSync value	Description
Sync to SysClk	Use this setting for any 24 MHz (SysClk) derived clock source that is divided by two or more. Examples include VC1, VC2, VC3 (when VC3 is driven by SysClk), 32 kHz, and digital PSoC blocks with SysClk-based sources. Externally generated clock sources should also use this value to ensure that proper synchronization occurs.
Sync to SysClk*2	Use this setting for any 48 MHz (SysClk*2) based clock unless the resulting frequency is 48 MHz (in other words, when the product of all divisors is 1).
SysClk Direct	Use this setting when a 24 MHz (SysClk/1) clock is wanted. This does not actually perform synchronization but provides low-skew access to the system clock itself. If selected, this option overrides the setting of the Clock parameter, above. It should always be used instead of VC1, VC2, VC3 or Digital Blocks where the net result of all dividers in combination produces a 24 MHz output.
Unsynchronized	Use this setting when the 48 MHz (SysClk*2) input is selected. Use when unsynchronized inputs are required. In general this use is advisable only when interrupt generation is the sole application of the Counter. This setting is required for blocks that remain active during sleep.

PWMEdgeAlign

When this parameter is set, the compare output will be delayed a half clock cycle. It is used to achieve a higher resolution when the 48 MHz clock is used as the block clock.

Trigger Mode

This parameter is used to control the software trigger enable control bit. When set to Software, the PWM will be triggered only by writing a 1 to Bit[0] of the PWMDB16L_CONTROL0_REG in software. If MultiShot[3:0] is not equal to zero, the PWMDB16L runs in the multi-shot mode. When this parameter is set to Hardware, the PWM module will be triggered only by the hardware.

Application Programming Interface

The Application Programming Interface (API) routines are provided as part of the user module to allow the designer to deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the include files.

Each time a user module is placed, it is assigned an instance name. By default, PSoC Designer assigns the PWMDB16L_1 to the first instance of this user module in a given project. It can be changed to any unique value that follows the syntactic rules for identifiers. The assigned instance name becomes the prefix of every global function name, variable, and constant symbol. In the following descriptions the instance name has been shortened to PWMDB16L for simplicity.

Note

** In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X prior to the call if those values are required after the call. This “registers are volatile” policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must ensure their code observes the policy, too. Though some user module API functions may leave A and X unchanged, there is no guarantee they will do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

PWMDB16L_Start

Description:

Starts both the pulse width modulator and the dead band generator PSoC blocks. The PWM Period register is loaded into the Counter register and the PWM16 clock is started. If the input Start is high, the counter will begin to down count.

C Prototype:

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

Assembly:

```
lcall PWMDB16L_Start
```

Parameters:

None

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

PWMDB16L_Stop

Description:

Disables the PWMDB16L PSoC blocks.

C Prototype:

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

Assembler:

```
lcall PWMDB16L_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

PWMDB16L_EnableInt

Description:

Enables the interrupt mode operation.

C Prototype:

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

Assembler:

```
lcall PWMDB16L_EnableInt
```

Parameters:

None

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

PWMDB16L_DisableInt

Description:

Disables the interrupt mode operation.

C Prototype:

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

Assembler:

```
lcall PWMDB16L_DisableInt
```

Parameters:

None

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

PWMDB16L_WritePeriod

Description:

Writes the PWMDB16L Period register with the period value.

C Prototype:

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

Assembler:

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

Parameters:

wPeriod value is a value from 0 to 2^{16} -1. The MSB is passed in the X register and the LSB is passed in the Accumulator.

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

PWMDB16L_WritePulseWidth

Description:

Writes the PWM PulseWidth register with the pulse width value.

C Prototype:

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

Assembler:

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

Parameters:

wPulseWidth is the pulse width value. Valid values are from zero to the period value. The MSB is passed in the X register and the LSB is passed in the Accumulator.

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

PWMDB16L_WriteDeadTime

Description:

Writes the DeadTime bits of the Control1 register with the dead time count value - DeadTime[2:0].

C Prototype:

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

Assembler:

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

Parameters:

bDeadTime is a value that sets the dead time. Valid values are from 0 to 7. The value is passed in the Accumulator.

Value	Description
0	No deadtime
1	Deadtime is 1 block clock
2	Deadtime is 2 block clocks
3	Deadtime is 4 block clocks
4	Deadtime is 8 block clocks
5	Deadtime is 16 block clocks
6	Deadtime is 32 block clocks
7	Deadtime is 64 block clocks

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

PWMDB16L_TriggerMultiShot

Description:

Set PWMDB16L Enable bit, or bit[0] in PWMDB16L_CONTROL0_REG.

Note When the multi-shot register is set to non-zero, PWMDB16L User Module will run into PPG (Programmable Pulse Generator) mode. And it will stop output pulse at last-shot. Only hardware and software start (writing 1 again to 'EN' bit) can resume the pulse output.

C Prototype:

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

Assembler:

```
lcall PWMDB16L_TriggerMultiShot
```

Parameters:

None

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

PWMDB16L_SetTrigMode

Description:

Set PWM software trigger mode, SWT, or Bit[1] in PWMDB16L_CONTROL0_REG.

C Prototype:

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

Assembler:

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

Parameters:

Symbolic Name	Value	Description
PWMDB16L_TRIGMODE_SOFTWARE_DISABLE	0	Disable software trigger mode
PWMDB16L_TRIGMODE_SOFTWARE_ENABLE	1	Enable software trigger mode

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

PWMDB16L_SetEdgeAlign

Description:

Sets the PWM edge-alignment mode, NPS, or Bit[3] in PWMDB16L_CONTROL0_REG.

C Prototype:

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

Assembler:

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

Parameters:

Symbolic name	Value	Description
PWMDB16L_EDGEALIGN_DISABLE	0	Normal mode
PWMDB16L_EDGEALIGN_ENABLE	1	Compare output will be delayed by half clock cycle to be finished. It is used to achieve a higher resolution when a 48 MHz clock is used as a block clock

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

PWMDB16L_WriteMultiShotCounter

Description:

Writes the multi-shot counter register with the multi-shot count value - MULTI_SHOT[3:0], or Bit[7:4] in PWMDB16L_CONTROL1_REG.

C Prototype:

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

Assembler:

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

Parameters:

The multi-shot counter value is a value from 0 to 15 and is passed in the Accumulator

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

PWMDB16L_wReadPulseWidth

Description:

Reads the PWM PulseWidth register.

C Prototype:

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

Assembler:

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

Parameters:

None

Return Value:

The PulseWidth value is stored in the PulseWidth register and returned in the Accumulator and X register.

Side Effects:

See Note** at the beginning of the API section.

PWMDB16L_ClearInt

Description:

Clears pending interrupts for the UM.

C Prototype:

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

Assembly:

```
lcall PWMDB16L_ClearInt
```

Parameters:

None

Return Value:

None

Side Effects:

See Note** at the beginning of the API section.

Sample Firmware Source Code

In the following examples, the correspondence between the C and assembly code is simple and direct. The values shown for period and compare value are each "off-by-1" from the cardinal values because the registers are zero-based; that is, zero is the terminal count in their down-count cycle. Passing a simple one byte parameter in the A register rather than on the stack is a performance optimization used by both the assembler and C compiler for user module APIs. The C compiler employs this mechanism for "INT" types instead of pushing the argument on the stack when it sees the #pragma fastcall declarations in the PWMDBL16.h file.

The following is the C sample code that illustrates the use of the APIs.

```
//
//  This sample shows how to generate 20% under-lapped output signals.
//
// OVERVIEW:
//
//  The PWMDB16L output can be routed to any pin.
//  In this example the PWMDB16L outputs is routed to P0[4] and P1[3] pins.
//  The pin P0[4] has the 40% duty cycle output pulse with frequency = 50 kHz.
//  The pin P1[3] has the 40% duty cycle output pulse with frequency = 50 kHz.
//  The second output is shifted to 50% (or 10 usec) relative to first one.
//
//The following changes need to be made to the default settings in the Device Editor:
//
//   1. Select PWMDB16L user module.
//   2. The User Module will occupy the space in dedicated system resources.
//   3. Rename User Module's instance name to PWMDB16L.
//   4. Set PWMDB16L's Clock Parameter to VC1.
//   5. Set PWMDB16L's Start Parameter to High.
//   6. Set PWMDB16L's DeadBandKill Parameter to Low.
//   7. Set PWMDB16L's ClockSync Parameter to Sync to SysClk.
//   8. Set PWMDB16L's Phase1 Parameter to Row_0_Output_0.
//   9. Set PWMDB16L's Phase2 Parameter to Row_0_Output_3.
//   10. Click on Row_0_Output_0 and connect Row_0_Output_0 to GlobalOutEven_4.
//   11. Select GlobalOutEven_4 for P0[4] in the Pinout.
//   12. Click on Row_0_Output_3 and connect Row_0_Output_3 to GlobalOutOdd_3.
//   13. Select GlobalOutOdd_3 for P1[3] in the Pinout.
//   14. Generate the project (Ctrl + F6).
//
// CONFIGURATION DETAILS:
//
//  1. The clock selected should be 30 times the required period.
```

```
//  2. The UM's instance name must be shortened to PWMDB16L.
//
// PROJECT SETTINGS:
//
//     IMO setting (SysClk) = 24MHz     System clock is set to 24MHz
//     VC1=SysClk/1  = 16 (default)
//
// USER MODULE PARAMETER SETTINGS:
//
//  ------------------------------------------------------------------------------
//  UM         Parameter         Value                 Comments
//  ------------------------------------------------------------------------------
//  PWMDB16L   Name              PWMDB16L                UM's instance name
//             Clock             VC1
//             Start             High
//             InvertStart       Normal
//             Period            0                     The Code changes it.
//             PulseWidth        0                     The Code changes it.
//             InterruptType     Phase1
//             PWMOutput         None
//             DeadTime          None                  The Code changes it.
//             Phase1            Row_0_Output_0
//             Phase2            Row_0_Output_3
//             DeadBandKill      Low
//             DeadBandKill_Mode SyncRestartKill
//             ClockSync         Sync to SysClk
//             PWMEdgeAlign      Disable
//             Trigger Mode      Software
//
//  ------------------------------------------------------------------------------

/* Code begins here */

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

#define PWM_PERIOD     29
#define PWM_PULSEWIDTH 14
#define PWM_DEATHTIME  2

void main(void)
{
     // M8C_EnableGInt ;              // Uncomment this line to enable Global Interrupts
    PWMDB16L_WritePeriod(PWM_PERIOD);         // Set period to 30 clocks
    PWMDB16L_WritePulseWidth(PWM_PULSEWIDTH);
    // Set pulse width to generate a 50% duty cycle
    PWMDB16L_WriteDeadTime(PWM_DEATHTIME);    // Set dead time to 20% -> (15*0.2)-1
    PWMDB16L_Start();                         // start the PWMDB16L!
    // Insert your main routine code here.
}
```

The same code in Assembly is:

```
;
;  This sample shows how to generate 20% under-lapped output signals.
;
; OVERVIEW:
;
;  The PWMDB16L output can be routed to any pin.
;  In this example the PWMDB16L outputs is routed to P0[4] and P1[3] pins.
;  The pin P0[4] has the 40% duty cycle output pulse with frequency = 50 kHz.
;  The pin P1[3] has the 40% duty cycle output pulse with frequency = 50 kHz.
;  The second output is shifted to 50% (or 10 usec) relative to first one.
;
;The following changes need to be made to the default settings in the Device Editor:
;
;   1. Select PWMDB16L user module.
;   2. The User Module will occupy the space in dedicated system resources.
;   3. Rename User Module's instance name to PWMDB16L.

;   6. Set PWMDB16L's Phase1 Parameter to Row_0_Output_0.
;   7. Set PWMDB16L's DeadBandKill Parameter to Low.
;   8. Set PWMDB16L's ClockSync Parameter to SyncSysClk.
;   9. Set PWMDB16L's Phase2 Parameter to Row_0_Output_3.
;   10. Click on Row_0_Output_0 and connect Row_0_Output_0 to GlobalOutEven_4.
;   11. Select GlobalOutEven_4 for P0[4] in the Pinout.
;   12. Click on Row_0_Output_3 and connect Row_0_Output_3 to GlobalOutOdd_3.
;   13. Select GlobalOutOdd_3 for P1[3] in the Pinout.
;   14. Generate the project (Ctrl + F6).
;
; CONFIGURATION DETAILS:
;
;  1. The clock selected should be 30 times the required period.

;     VC1=SysClk/1  = 16 (default)
;
; USER MODULE PARAMETER SETTINGS:
;
;  ------------------------------------------------------------------------------
;  UM         Parameter         Value                 Comments

;-------------------------------------------------------------------------------
; Assembly code begins here
;-------------------------------------------------------------------------------

include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export _main

PWMDB_LSB_PERIOD:     equ 29
PWMDB_MSB_PERIOD:     equ 0
PWMDB_LSB_PULSEWIDTH: equ 14
PWMDB_MSB_PULSEWIDTH: equ 0
PWMDB_DEADTIME:       equ 2
```

```
_main:

    ; M8C_EnableGInt          ; Uncomment this line to enable Global Interrupts
    mov   A, PWMDB_LSB_PERIOD     ; set the period to be 30 counts of the clock
    mov   X, PWMDB_MSB_PERIOD
    call  PWMDB16L_WritePeriod
    mov   A, PWMDB_LSB_PULSEWIDTH ; set the pulse width to create 50% duty cycle
    mov   X, PWMDB_MSB_PULSEWIDTH
    call  PWMDB16L_WritePulseWidth
    mov   A, PWMDB_DEADTIME      ; set the dead time to 20% ->  (15*0.2)-1
    call  PWMDB16L_WriteDeadTime
    call  PWMDB16L_Start            ; start the PWMDB16L – counter will start to
                                    ; count when the enable input is asserted high

    ; Insert your main assembly code here.
.terminate:
    jmp .terminate
```

Configuration Registers

The PWMDB16L User Module is personalized and parameterized through the registers. The following tables show the register values as constants and the parameters as named bit-fields with brief descriptions. Symbolic names for these registers are defined in the user module's C and assembly language interface files (the ".h" and ".inc" files).

Table 2. PWMDB16L_FUNC_REG

Block/Bit	7	6	5	4	3	2	1	0
MSB	0	0	1	DeadBandKillMode		0	1	1
LSB	InvertDeadBandKill	0	0	0	0	0	1	1

The InvertDeadBandKill flag allows you to invert the incoming DeadBand Kill signal. DeadBandKillMode selects one of three Kill modes, SyncRestartKill, DisableKill, or AsyncKill. Parameters are set in the Device Editor.

Table 3. PWMDB16L_INPUT_REG

Block/Bit	7	6	5	4	3	2	1	0
MSB	0	0	1	1	Clock[3:0]			
LSB	DeadBandKill[3:0]				Clock[3:0]			

DeadBandKill selects the input signal of the same name from one of 16 sources. The user module "DeadBandKill" parameter setting in the Device Editor determines its value. Similarly, the user module "Clock" parameter setting determines the Clock value.

Table 4. PWMDB16L_OUTPUT_REG

Block/Bit	7	6	5	4	3	2	1	0
MSB	ClockSync		1	Phase2[1:0]		1	Phase1[1:0]	
LSB	ClockSync		0	0	0	0	0	0

The user module "ClockSync" parameter in the Device Editor determines the value of the ClockSync bits. Phase1 and Phase2 parameters in the Device Editor may be routed to one of four global output buses.

Table 5. PWMDB16L_COUNT_REG

Block/Bit	7	6	5	4	3	2	1	0
MSB	Count(MSB)							
LSB	Count(LSB)							

Count is the PWMDB16L MSB and LSB down counter. It can be read using the PWMDB16L API.

Table 6. PWMDB16L_PERIOD_REG

Block/Bit	7	6	5	4	3	2	1	0
MSB	Period(MSB)							
LSB	Period(LSB)							

Period holds the MSB and LSB of the period value that is loaded into the Counter register upon Start or terminal count condition. It can be set in the Device Editor and the PWMDB16L API.

Table 7. PWMDB16L_PULSE_WIDTH_REG

Block/Bit	7	6	5	4	3	2	1	0
MSB	PulseWidth(MSB)							
LSB	PulseWidth(LSB)							

PulseWidth holds the MSB and LSB of the PulseWidth value used to generate the compare event. It is set in the Device Editor and the PWMDB16L API.

Table 8. PWMDB16L_CONTROL0_REG

Block/Bit	7	6	5	4	3	2	1	0
MSB	0	0	0	0	PWMEdgeAlign	InterruptType	0	0
LSB	Start [3:0]				0	InterruptType	SWTrigger	Enable

Start [3:0] is the selection of “Start” trigger source input selection, a 16:1 mux from 16 possible digital input source. The InterruptType bit sets the interrupt trigger type. These parameters are set in the Device Editor only. The PWMEdgeAlign bit is the PWM output edge-alignment selection bit. The SWTrigger bit is the software trigger enable bit. These parameters are set in the Device Editor and the PWMDB16L API. Enable indicates that the PWMDB16L is enabled when set and disabled when clear. It is modified by using the PWMDB16L API.

Table 9. PWMDB16L_CONTROL1_REG

Block/Bit	7	6	5	4	3	2	1	0
MSB	MULTI_SHOT[3:0]				0	DeadTime[2:0]		
LSB	0	0	0	0	InvertStart	0	0	0

The MULTI_SHOT[3:0] value is the multi-shot counter register. It is modified by using the PWMDB16L API. The InvertStart bit allows you to invert the incoming DeadBand Kill signal. It is set in the Device Editor. The DeadTime is the deadband width control. It can be 0, 1, 2, 4, 8, 16, 32, or 64 block clock cycles. Setting DeadTime to zero means there is no deadband protection. It is set in the Device Editor and the PWMDB16L API.

Version History

Version	Originator	Description
1.0	DHA	Initial version
1.10	HPHA	Corrected method of clearing posted interrupts.

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.

Copyright © 2009-2014 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.