8-Bit PWM Dead Band Generator Datasheet PWMDB8 V 2.5

Copyright © 2002-2015 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8CTMA140, CY8C28x45, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx						
8-bit	2	0	0	35	0	1
16-bit	3	0	0	44	0	1

For one or more fully configured, functional example projects that use this user module go to www.cypress.com/psocexampleprojects.

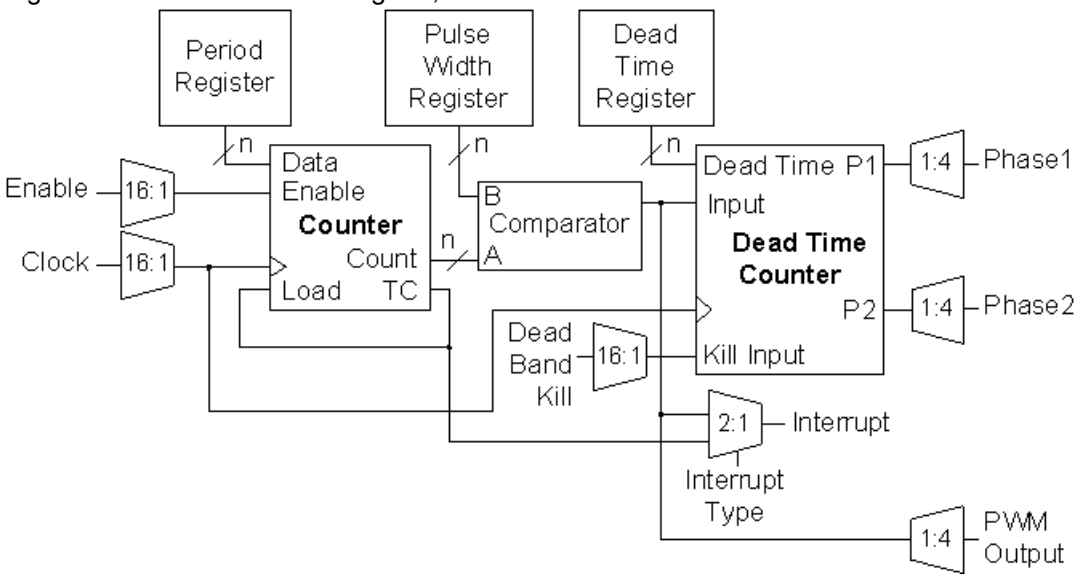
Features and Overview

- 8-bit general purpose pulse width modulator (PWM) with an 8-bit dead band generator consumes two PSoC blocks
- Phase1 and Phase2 underlapped outputs track the frequency of the generated PWM signal
- Programmable duty cycle
- Programmable dead time
- Dead Band Kill input drives Phase1 and Phase2 outputs low
- Counter clocking up to 48 MHz
- Interrupt option triggered on rising edge of the PWM generated signal or counter terminal count

The 8-bit PWMDB User Module is a pulse width modulator combined with an 8-bit dead band generator. The pulse width modulator provides a programmable period and pulse width input signal to the dead band generator. The dead band generator outputs two under-lapped signals, with programmable dead time at the same frequency as the input signal. When asserted, the Dead Band Kill input drives 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 both edges of the pulse width modulator output.

Figure 1. PWMDB Block Diagram, Data Path width n = 8



Functional Description

The PWMDB User Module employs two digital PSoC blocks. The first block, PWM8, implements a pulse width modulator with programmable period and pulse width. The pulse width modulated output signal is fed into the second PSoC block, DB8. DB8 implements a dead band generator with programmable dead time. The two output signals, Phase1 and Phase2, provide the PWMDB8 outputs.

The Control registers start and stop both the PWM and the DB8 components of the PWMDB. Writing the Period register while stopped, causes the new Period register value to be copied to the Counter register. Writing the DeadTime register while stopped, causes the new DeadTime register value to be loaded into the DeadTimeCounter register. While the PWMDB is stopped, the PWM output and the DB8 Phase1 and Phase2 outputs are asserted low.

The PWMDB is gated by an active high enable signal. While asserted low, the PWM and the DB8 PSoC blocks are effectively disabled from operating. The PWM output is held constant at its current state, thus preventing the DB8 from modifying its outputs. Asserting the enable signal continues operation without modifying current register contents.

The same input clock is used by both the 8-bit PWM and the DB8 components of the PWMDB.

Pulse Width Modulator

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 PWM is effectively the period value programmed in the Period register, plus one.

Equation 1

$$OutputPeriod = PeriodValue + 1$$

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 is set high. On every clock, 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 set equal, the output remains high, indefinitely. The pulse width value may have the 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 the PWM output or on the Counter register’s terminal count condition. The terminal count condition is one-half clock period before the falling edge of the output signal. The interrupt option can be set using the Device Editor. Enabling or disabling the interrupt is done at run time using the API.

Dead Band Generator

For each edge of the input signal (PWM8 output), the following is repeated:

Rising Edge

- Phase2 signal is reset low on the rising edge of the next clock cycle.
- DeadTimeCounter register is loaded with the DeadTime register value.
- DeadTimeCounter register 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

- Phase1 signal is reset low on the rising edge of the following clock cycle.
- DeadTimeCounter register is loaded with the DeadTime register value.
- DeadTimeCounter register 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 as follows.

Equation 3

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

The DeadTime register must be loaded with an 8-bit value. It must range from zero to the minimum of the PWM Period register value minus two and PWM PulseWidth register value minus two, or 255.

The DeadTime register 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 drives the Phase1 and Phase2 outputs low. This signal only affects the output gating of Phase1 and Phase2 signals and not the DeadTimeCounter register. 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 is synced with the pulse width modulated input. This means that the first output pulse is lengthened.

If it is required that the PWMDB output always be synchronized to the generated PWM input, upon the de-assertion of the Dead Band Kill input, then upon the detection of the Dead Band Kill high assertion:

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

The 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, CY8C28xxxfamilies support three KILL modes. In all cases, The KILL signal asynchronously forces the outputs to logic '0'. The differences in the modes come from how dead band processing is restarted.

1. **Synchronous Restart Mode**: When KILL is asserted high 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. When using a PWM with 100% duty cycle, there is no incoming reference edge, and the dead band processing does not restart when KILL is negated. Refer Synchronous Restart Kill Mode Figure on page 5.
2. **Asynchronous Restart Mode**: When KILL is asserted high, 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 cycle. Refer Asynchronous Restart Kill Mode Figure on page 5.

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

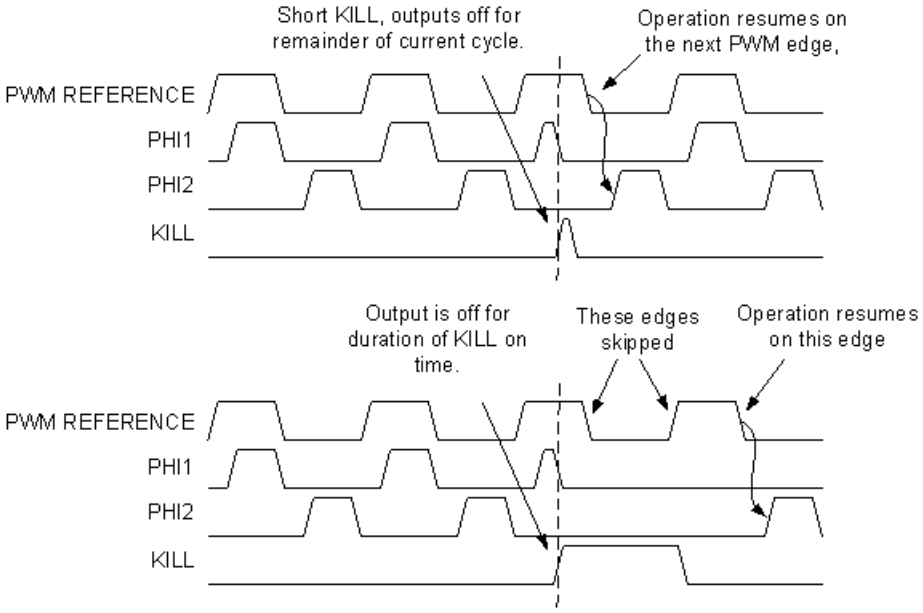
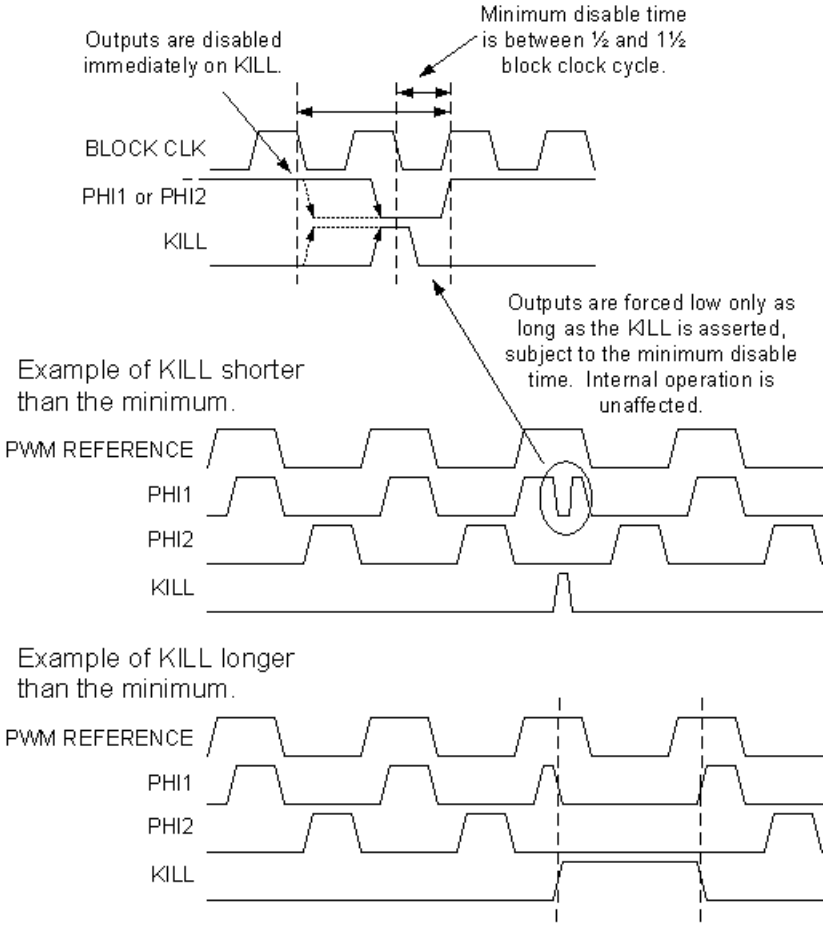


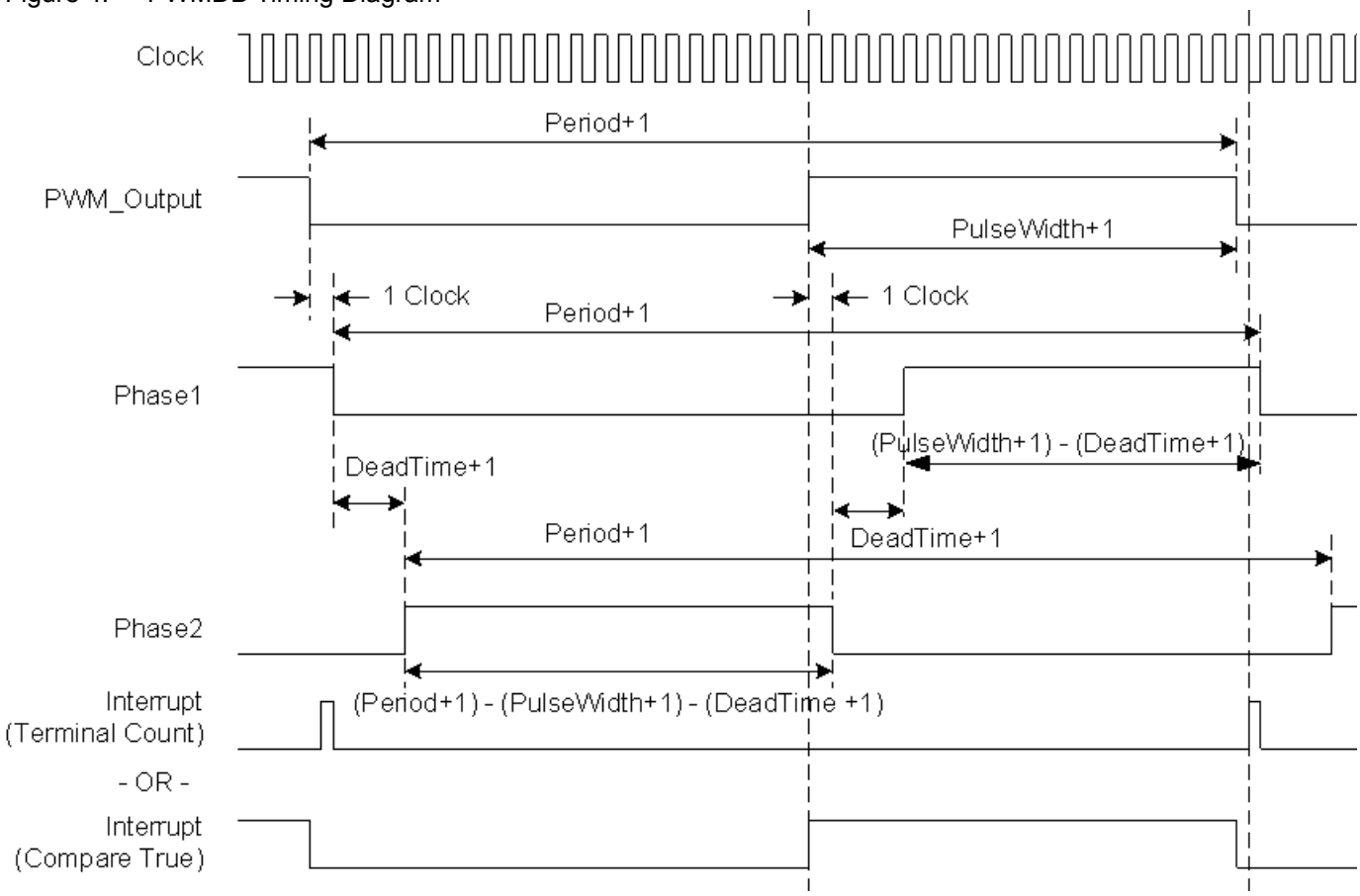
Figure 3. Asynchronous Restart Kill Mode



Timing

PWMDB operation may be gated On and Off or clocked by external pins routed to the PWMDB by the global bus feature of the device. The global buses have a frequency limitation of 12 MHz.

Figure 4. PWMDB Timing Diagram



DC and AC Electrical Characteristics

Table 1. PWMDB 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 8-bit PWMDB consumes two digital PSoC blocks. Each block is given a symbolic name displayed by the Device Editor during and after placement. The API qualifies all register names with user assigned instance name and block name to provide direct access to the PWMDB registers through the API include files. The block names used by the various widths are given in the following table.

Table 2. Symbolic Names of the Mapped PSoC Blocks

PSoC Block Number	8-bit PWMDB
1	PWM8
2	DB8

Parameters and Resources

Clock

The Clock parameter is selected from one of a number of sources. These sources include the 48 MHz oscillator (5.0V operation only), lower frequencies (24V1 and 24V2) divided down from the 24 MHz system clock, other PSoC blocks, and external inputs routed through global inputs and outputs. Both the pulse width modulator and the dead band generator use the same clock source. When using an external digital clock for the block, the row input synchronization should be turned off for best accuracy, and sleep operation.

Enable

The Enable parameter is selected from one of a number of sources. External inputs, from global inputs and outputs, are automatically synchronized to the internal 24 MHz oscillator of the device.

Period

This parameter sets the period of the PWM counter. Allowed values are between 0 and 255 for an 8-bit PWM and between 0 and 2^{16}-1 for 16 bits. The period is loaded into the Period register. The effective output waveform period of the PWM is the period count + 1. The value may be modified using the API.

PulseWidth

This parameter sets the pulse width of the PWM output. Allowed values are between zero and the period value. The value may be modified using the API.

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 signal or on the terminal count of the PWM Counter register. A separate register independently enables the interrupt.

PWMOutput

This output parameter may be routed to one of four global output buses. If it is not required, you are advised to avoid routing this signal (to save the global resources for other outputs).

DeadTime

This parameter sets the dead time count of the DB8 output. An 8-bit value in the range of zero to the minimum of the following: The PWM Period parameter minus two, PWM Pulse Width parameter value minus two, or 255.

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.

ClockSync

In the PSoC devices, digital blocks may provide clock sources in addition to the system clocks. Digital clock sources may even be chained in ripple fashion. This introduces skew with respect to the system clocks. These skews are more critical in the CY8C29/27/24/22/21xxx and CY8CLED04/08/16 PSoC device families because of various data-path optimizations, particularly those applied to the system busses. 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 table.

ClockSync Value	Use
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), 32KHz, 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).
Use SysClk Direct	Use when a 24 MHz (SysClk/1) clock is desired. 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. 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 when the 48 MHz (SysClk*2) input is selected. Use when unsynchronized inputs are desired. 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.

DeadBandKill Mode

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

Invert DeadBandKill

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

Invert Enable

This parameter allows the user to invert the incoming Enable signal.

Interrupt Generation Control

There are two additional parameters that become available when the **Enable interrupt generation control** check box in PSoC Designer is checked. This is available under **Project > Settings > Chip Editor**. Interrupt Generation Control is important when multiple overlays are used with interrupts shared by multiple user modules across overlays:

- Interrupt API
- IntDispatchMode

InterruptAPI

The InterruptAPI parameter allows conditional generation of a user module's interrupt handler and interrupt vector table entry. Select "Enable" to generate the interrupt handler and interrupt vector table entry. Select "Disable" to bypass the generation of the interrupt handler and interrupt vector table entry. Properly selecting whether an Interrupt API is to be generated is recommended particularly with projects that have multiple overlays where a single block resource is used by the different overlays. By selecting only Interrupt API generation when it is necessary the need to generate an interrupt dispatch code might be eliminated, thereby reducing overhead.

IntDispatchMode

The IntDispatchMode parameter is used to specify how an interrupt request is handled for interrupts shared by multiple user modules existing in the same block but in different overlays. Selecting "ActiveStatus" causes firmware to test which overlay is active before servicing the shared interrupt request. This test occurs every time the shared interrupt is requested. This adds latency and also produces a nondeterministic procedure of servicing shared interrupt requests, but does not require any RAM. Selecting "OffsetPreCalc" causes firmware to calculate the source of a shared interrupt request only when an overlay is initially loaded. This calculation decreases interrupt latency and produces a deterministic procedure for servicing shared interrupt requests, but at the expense of a byte of RAM.

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.

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 before 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 function may leave A and X unchanged, there is no guarantee they may 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.

PWMDB8_PERIOD

Description:

Represents the value chosen for the Period field of the PWMDB8 in the Device Editor. The value can have a range between 0 and 255.

PWMDB8_PULSE_WIDTH

Description:

Represents the value chose for the PulseWidth field of the PWMDB8 in the Device Editor. The value can have a range between 0 and 255.

PWMDB8_EnableInt

Description:

Enables the interrupt mode operation.

C Prototype:

```
void  PWMDB8_EnableInt(void);
```

Assembly:

```
lcall  PWMDB8_EnableInt
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

PWMDB8_DisableInt

Description:

Disables the interrupt mode operation.

C Prototype:

```
void  PWMDB8_DisableInt(void);
```

Assembly:

```
lcall  PWMDB8_DisableInt
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

PWMDB8_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 PWM8 clock is started. If the input enable is high, the counter begins to down count.

C Prototype:

```
void  PWMDB8_Start(void);
```

Assembly:

```
lcall  PWMDB8_Start
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

PWMDB8_Stop

Description:

Disables the PWM8 and DB8 PSoC blocks.

C Prototype:

```
void  PWMDB8_Stop(void);
```

Assembly:

```
lcall  PWMDB8_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

PWMDB8_WritePeriod

Description:

Writes the PWM Period register with the period value.

C Prototype:

```
void  PWMDB8_WritePeriod(BYTE bPeriod);
```

Assembly:

```
mov   A, [bPeriod]
lcall  PWMDB8_WritePeriod
```

Parameters:

Period value is a value from 0 to 255 and is passed in the Accumulator.

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

PWMDB8_WritePulseWidth

Description:

Writes the PWM PulseWidth register with the pulse width value.

C Prototype:

```
void  PWMDB8_WritePulseWidth(BYTE bPulseWidth);
```

Assembly:

```
mov   A, [bPulseWidth]
lcall  PWMDB8_WritePulseWidth
```

Parameters:

Pulse width value is the value from zero to the period value and is passed in the Accumulator.

Return Value:

None

Side Effects:

Writing the PulseWidth register, while the counter is active, changes the duty cycle of the output. This may cause the output to glitch or change inadvertently. The A and X registers may be altered by this function.

PWMDB8_WriteDeadTime

Description:

Writes the DB8 DeadTime register with the dead time count value.

C Prototype:

```
void  PWMDB8_WriteDeadTime(BYTE bDeadTime);
```

Assembly:

```
mov   A, [bDeadTime]
lcall  PWMDB8_WriteDeadTime
```

Parameters:

Pulse width value is a value from 0 to the period value and is passed in the Accumulator.

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

PWMDB8_bReadPulseWidth

Description:

Reads the PWM PulseWidth register.

C Prototype:

```
BYTE  PWMDB8_bReadPulseWidth();
```

Assembly:

```
lcall  PWMDB8_bReadPulseWidth
mov   [bPulseWidth], A
```

Parameters:

None

Return Value:

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

Side Effects:

The A and X registers may be altered by this function.

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 PWMDB8.h file.

The following is the assembly language source that illustrates the use of the APIs.

```
;-------------------------------------------------------------------------
; FILENAME: main.asm
;
; DESCRIPTION:
;
;  This sample shows how to generate 20% under-lapped output signals.
;
; OVERVIEW:
;
;  The PWMDB8 output can be routed to any pin.
;  In this example the PWMDB8 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 PWMDB8 user module.
;   2. The User Module will occupy the space in dedicated system resources.
;   3. Rename User Module's instance name to PWMDB8.
;   4. Set PWMDB8's Clock Parameter to VC1.
;   5. Set PWMDB8's Enable Parameter to High.
```

```
;   6. Set PWMDB8's Phase1 Parameter to Row_0_Output_0.
;   7. Set PWMDB8's CompareType Parameter to Less Than Or Equal.
;   8. Set PWMDB8's ClockSync Parameter to SyncSysClk.
;   9. Set PWMDB8'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 PWMDB8.
;
; 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
;  ------------------------------------------------------------------------------
;  PWMDB8     Name              PWMDB8               UM's instance name
;             Clock             VC1
;             Enable            High
;             Period            0                    The Code changes it.
;             PulseWidth        0                    The Code changes it.
;             InterruptType     Terminal Count
;             PWMOutput         Row_0_Output_1
;             DeadTime          0                    The Code changes it.
;             Phase1            Row_0_Output_0
;             Phase2            Row_0_Output_3
;             DeadBandKill      Low
;             DeadBandKill_Mode SyncRestartKill
;             ClockSync         SyncSysClk
;             InvertDeadBandKill Normal
;             InvertEnable      Normal
;
;  ------------------------------------------------------------------------------

;-------------------------------------------------------------------------------
; 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_PERIOD:     equ 29
PWMDB_PULSEWIDTH: equ 14
```

```
PWMDB_DEADTIME:   equ 2

_main:

    ; M8C_EnableGInt               ; Uncomment this line to enable Global Interrupts
    mov   A, PWMDB_PERIOD         ; set the period to be 30 counts of the clock
    lcall PWMDB8_WritePeriod
    mov   A, PWMDB_PULSEWIDTH     ; set the pulse width to create 50% duty cycle
    lcall PWMDB8_WritePulseWidth
    mov   A, PWMDB_DEADTIME       ; set the dead time to 20% ->  (15*0.2)-1
    lcall PWMDB8_WriteDeadTime
    lcall PWMDB8_Start            ; start the PWMDB8 - counter will start to
                                  ; count when the enable input is asserted high
    ; Insert your main assembly code here.

.terminate:
    jmp .terminate
```

The same code in C is:

```
//-------------------------------------------------------------------------
// FILENAME: main.c
//
// DESCRIPTION:

//  This sample shows how to generate 20% under-lapped output signals.
//
// OVERVIEW:
//
//  The PWMDB8 output can be routed to any pin.
//  In this example the PWMDB8 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 PWMDB8 user module.
//   2. The User Module will occupy the space in dedicated system resources.
//   3. Rename User Module's instance name to PWMDB8.
//   4. Set PWMDB8's Clock Parameter to VC1.
//   5. Set PWMDB8's Enable Parameter to High.
//   6. Set PWMDB8's Phase1 Parameter to Row_0_Output_0.
//   7. Set PWMDB8's CompareType Parameter to Less Than Or Equal.
//   8. Set PWMDB8's ClockSync Parameter to SyncSysClk.
//   9. Set PWMDB8'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 PWMDB8.
```

```
//
// 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
//  ------------------------------------------------------------------------------
//  PWMDB8     Name              PWMDB8               UM's instance name
//             Clock             VC1
//             Enable            High
//             Period            0                    The Code changes it.
//             PulseWidth        0                    The Code changes it.
//             InterruptType     Terminal Count
//             PWMOutput         Row_0_Output_1
//             DeadTime          0                    The Code changes it.
//             Phase1            Row_0_Output_0
//             Phase2            Row_0_Output_3
//             DeadBandKill      Low
//             DeadBandKill_Mode SyncRestartKill
//             ClockSync         SyncSysClk
//             InvertDeadBandKill Normal
//             InvertEnable      Normal
//
//  ------------------------------------------------------------------------------

/* 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
    PWMDB8_WritePeriod(PWM_PERIOD);  // Set period to 30 clocks
    PWMDB8_WritePulseWidth(PWM_PULSEWIDTH);     // Set pulse width to generate a 50%
                                                //duty cycle
    PWMDB8_WriteDeadTime(PWM_DEATHTIME);     // Set dead time to 20% -> (15*0.2)-1
    PWMDB8_Start();
    // Insert your main routine code here.
}
```

Configuration Registers

The 8-bit PWMDB uses two digital PSoC blocks named PWM8 and PWMDB8. Each block is personalized and parameterized through 7 registers. The following tables give the “personality” values as constants and the parameters as named bit-fields with brief descriptions. Symbolic names for these registers are defined in the user module instance’s C and assembly language interface files (the “.h” and “.inc” files).

PWM8 Configuration Registers

Table 3. Block PWM8: Register Function

Bit	7	6	5	4	3	2	1	0
Value	0	0	1	0	Interrupt Type	0	0	1

Interrupt Type is a flag that indicates whether to trigger the interrupt on the rising edge of the output signal or on the terminal count condition. This parameter is set in the Device Editor.

Table 4. Block PWM8: Register Input

Bit	7	6	5	4	3	2	1	0
Value	Enable				Clock			

Enable selects the data input from one of a number of sources. Clock selects the input clock from one of a number of sources. Both parameters are set in the Device Editor.

Table 5. Block PWM8: Register Output

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	Out Enable	Output Select	

Output Enable is a flag that indicates the output is enabled. Output Select is a flag that indicates where the output of the PWM is routed. Both parameters are set in the Device Editor.

Table 6. Block PWM8: Counter Register DR0

Bit	7	6	5	4	3	2	1	0
Value	Count							

Count is the PWM8 down counter. It can be read using the PWM8 API.

Table 7. Block PWM8: Period Register DR1

Bit	7	6	5	4	3	2	1	0
Value	Period							

Period holds the period value that is loaded into the Counter register upon start or the terminal count condition. It can be set in the Device Editor and the PWM8 API.

Table 8. Block PWM8: Pulse Width Register DR2

Bit	7	6	5	4	3	2	1	0
Value	PulseWidth							

PulseWidth holds the pulse width value used to generate the output. It can be set in the Device Editor and the PWM8 API.

Table 9. Block PWM8: Control Register CR0

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	Start/Stop

Start/Stop indicates that the PWM8 is enabled when set. It is modified by using the PWM8 API.

DB8 Configuration Registers

Table 10. Block DB8: Register Function

Bit	7	6	5	4	3	2	1	0
Value	0	0	1	0	0	1	0	0

Table 11. Block DB8: Register Input

Bit	7	6	5	4	3	2	1	0
Value	Dead Band Kill				Clock			

Dead Band Kill selects the data input from one of a number of sources. Clock selects the input clock from one of a number of sources. Both parameters are set in the Device Editor.

Table 12. Block DB8: Register Output

Bit	7	6	5	4	3	2	1	0
Value	0	0	Phase2 Output Enable	Phase2 Output Select		Phase1 Output Enable	Phase1 Output Select	

Phase1 Output Enable is a flag that indicates that Phase 1 output is enabled. Phase1 Output Select specifies where the Phase 1 output of the DB8 is routed. Phase2 Output Enable is a flag that indicates that Phase 2 output is enabled. Phase2 Output Select specifies where the Phase 2 output of the DB8 is routed. All these parameters are set in the Device Editor.

Table 13. Block DB8: Dead Time Counter Register DR0

Bit	7	6	5	4	3	2	1	0
Value	Dead Time Counter							

Dead Time Counter is the DB8 down counter.

Table 14. Block DB8: Dead Time Register DR1

Bit	7	6	5	4	3	2	1	0
Value	Dead Time							

Dead Time holds the dead time count value. It is modified using the PWMDB8 API.

Table 15. Block DB8: Register DR2

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

This register is not used.

Table 16. Block DB8: Control Register CR0

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	Start/Stop

Start/Stop indicates that the DB8 is enabled when set. It is modified by using the PWMDB8 API.

Version History

Version	Originator	Description
2.5	TDU	Updated Clock description to include: When using an external digital clock for the block, the row input synchronization should be turned off for best accuracy, and sleep operation.

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 © 2002-2015 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.