16-Bit Hardware Pulse Width Modulator Datasheet PWM16HW V 1.0

Copyright © 2009-2012 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	PSoC® Blocks	API Memory (Bytes)		Pins (per External I/O)
	DPWM	Flash	RAM	
CY8CLED0xD, CY8CLED0xG	1	167	0	1

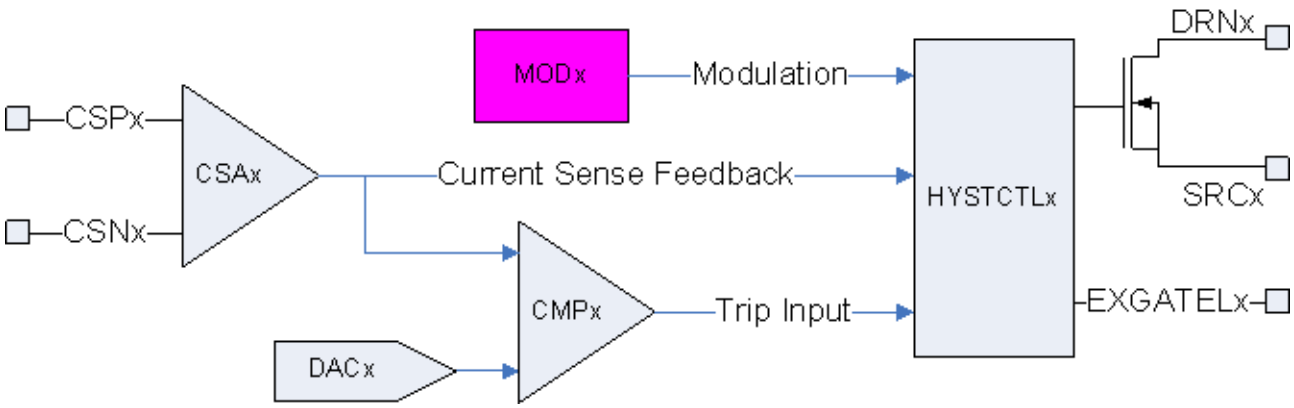
For one or more fully configured, functional example projects that use this user module go to www.cypress.com/psocexampleprojects.

Features and Overview

- Programmable resolution up to 16 bits.
- Programmable output frequencies up to 48 MHz.
- Dedicated PWM UM frees PSoC core digital blocks for other uses.
- Automatic reload of period for each pulse cycle.
- Interrupt option on rising edge of the output or terminal count.
- Precise PWM phase control to reduce system current edges.

The PWM16HW User Module (UM) features a Counter and a Pulse Width register. A comparator output asserts when the count value is less than or equal to the value in the Pulse Width register.

Figure 1. PWM16HW Block Diagram



Functional Description

The PWM16HW User Module employs a dedicated DPWM hardware resource.

The PWM16HW asserts its output high when stopped. While running, a comparator controls the duty cycle of the output signal. During every clock cycle, this comparator tests the values of the Counter register against that of the Pulse Width register, performing a "less than" or "less than or equal to" test depending on the CompareType parameter. The PWM16HW asserts the active high truth value of the comparison at the rising edge of the clock following the period in which the comparison is made. The ratio between the pulse width and the period sets the duty cycle of the output waveform.

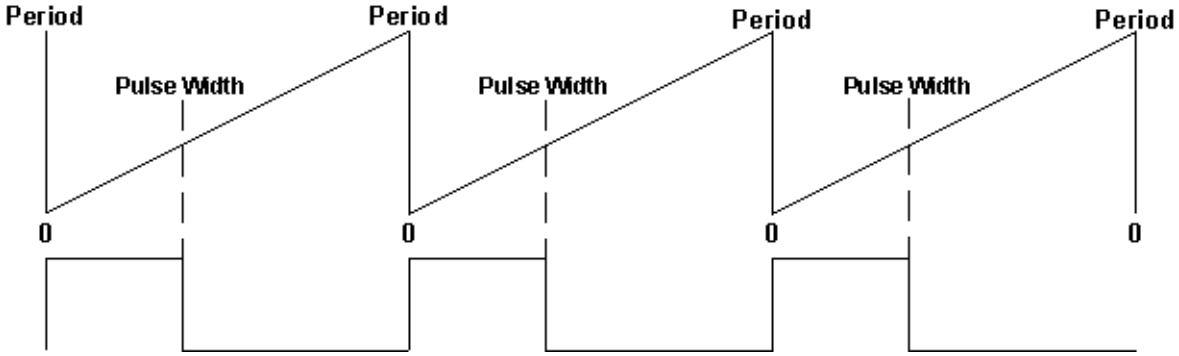
Alignment

PWM16HW provides three options for output pulse alignment: right aligned, left aligned, and center aligned. Phase shift enables staggering of the PWM16HW phase in the left and right aligned configurations when multiple modulators are working in SyncMode. For each of these three alignments, the internal counter of the DPWM block counts in a different direction.

Left Alignment

To achieve a left aligned PWM16HW output, the internal counter of the DPWM counts up from zero to Period. Every time the counter hits period, it reloads to zero. The output pulse asserts when the counter is "less than" (or "less than or equal to") the pulse width. The Period register can be modified with a new value anytime. If the counter reloads to zero, it becomes greater than the period after new period value is written.

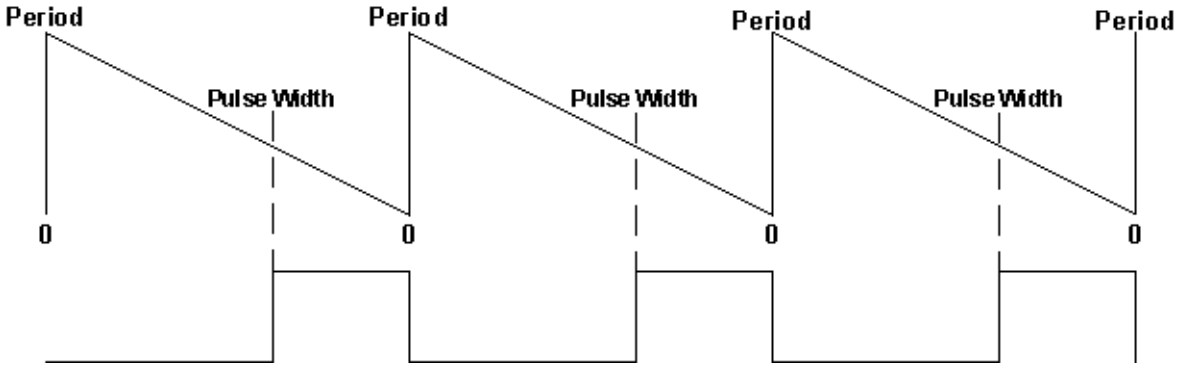
Figure 2. PWM16HW Left Alignment



Right Alignment

For right alignment, the PWM16HW uses a down counter from Period to zero. Every time the counter hits zero, it reloads to Period. The output asserts when the counter is "less than" (or "less than or equal to") the pulse width.

Figure 3. PWM16HW Right Alignment



The Period register can be modified with a new value at anytime. When the PWM is stopped, writing a value to the Period register also changes the value in the Count register. While the PWM is running, writing the Period register does not update the Count register with the new Period value until the next reload occurs, following terminal count. Because the terminal count is reached when the count is zero, the period of operation and of the output signal is greater by one than the value stored in the Period register. For the left and right pulse alignment, the duration in terms of the Period of the input clock is given by the following equation:

Equation 1

$$OutputPeriod = (Period + 1)t_{CLK}$$

The duty cycle ratio can be computed using this equation:

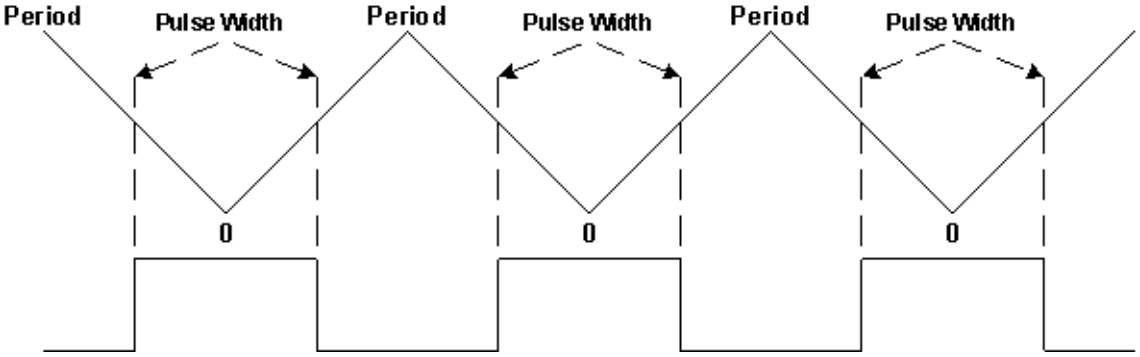
Equation 2

$$DutyCycle = \begin{cases} \frac{PulseWidth}{Period + 1} & \text{for less than comparison} \\ \frac{PulseWidth + 1}{Period + 1} & \text{for less than or equal to comparison} \end{cases}$$

Center Alignment

For center alignment there is an up-down counter. First, it counts down from Period to zero; then it counts up from zero to period. This means that every time the counter hits zero or Period, the direction of the counter toggles. The output asserts when the counter is "less than" (or "less than or equal to") the pulse width.

Figure 4. PWM16HW Center Alignment



As shown in the previous figure, the counter sweep in center alignment is different from that of left or right alignment. For a given Period value, the periodic time of the counter sweep is almost double that for either left or right alignment. For a given pulse width value, the width of the output pulse in center alignment mode is almost double that of either left or right alignment mode.

The expression for the effective period in center alignment mode is:

Equation 3

$$OutputPeriod = 2\ Period\ t_{CLK}$$

The duty cycle ratio can be computed using the following equation:

Equation 4

$$DutyCycle = \begin{cases} \frac{2 \cdot PulseWidth - 1}{2 \cdot Period} & \text{for less than comparison} \\ \frac{2 \cdot PulseWidth + 1}{2 \cdot Period} & \text{for less than or equal to comparison} \end{cases}$$

The value of the Pulse Width register may be set using the Device Editor or during run time using the PWM16HW_SetPulseWidth() API. Because the pulse width register is not buffered the same way the counter register is, changes to the pulse width register while the user module is running affect the compare output on the next clock cycle.

Interrupts

The PWM16HW User Module has a slightly different interrupt logic relative to standard PSoC digital blocks. The peculiarity is that all the modulator blocks share two available interrupt vectors. These interrupt vectors are the High Priority (HP) Interrupt (22h) and the Low Priority (LP) Interrupt (23h). Each modulator block can generate either LP or HP interrupt or both. If several user modules use the same interrupt vector, then an interrupt dispatcher is used to call all modules interrupt service routine. This dispatcher sequentially calls LowISR() or HighISR() functions of all user modules that have the corresponding interrupt enabled. To find out which block caused the interrupt trigger, the DPWMINTFLAG register may be checked.

There are several UM parameters and API functions concerning interrupts. To use PWM16HW interrupt, do the following:

- Enable LowISR or/and HighISR parameter in the Device Editor. This also includes this module ISR call into interrupt dispatcher.
- Call PWM16HW_EnableLPIntGlobal() or PWM16HW_EnableHPIntGlobal() API functions to enable corresponding interrupt vectors. It is enough to call these functions once.
- Enable global interrupts using M8C_EnableGInt.

An interrupt can be programmed to occur on terminal count or when the compare becomes true. This option is set by the InterruptType parameter using the Device Editor.

Enabling or disabling the interrupt is done at run time using the PWM16HW_EnableHPInt(), PWM16HW_DisableHPInt(), PWM16HW_EnableLPInt(), and PWM16HW_DisableLPInt() APIs. Use these functions if corresponding LowISR / HighISR parameter is enabled.

Sync Mode

In Sync Mode, two or more PWM16HW User Modules can operate synchronously. Four, three, or two of the PWM16HW blocks can participate in Sync Mode. The SyncMode parameter settings defines which PWM16HW blocks are participating in Sync Mode; this can be changed at run time using corresponding API functions. In Sync Mode, one of the participating PWM16HW modules is specified as the master. The remaining participating blocks are slaves.

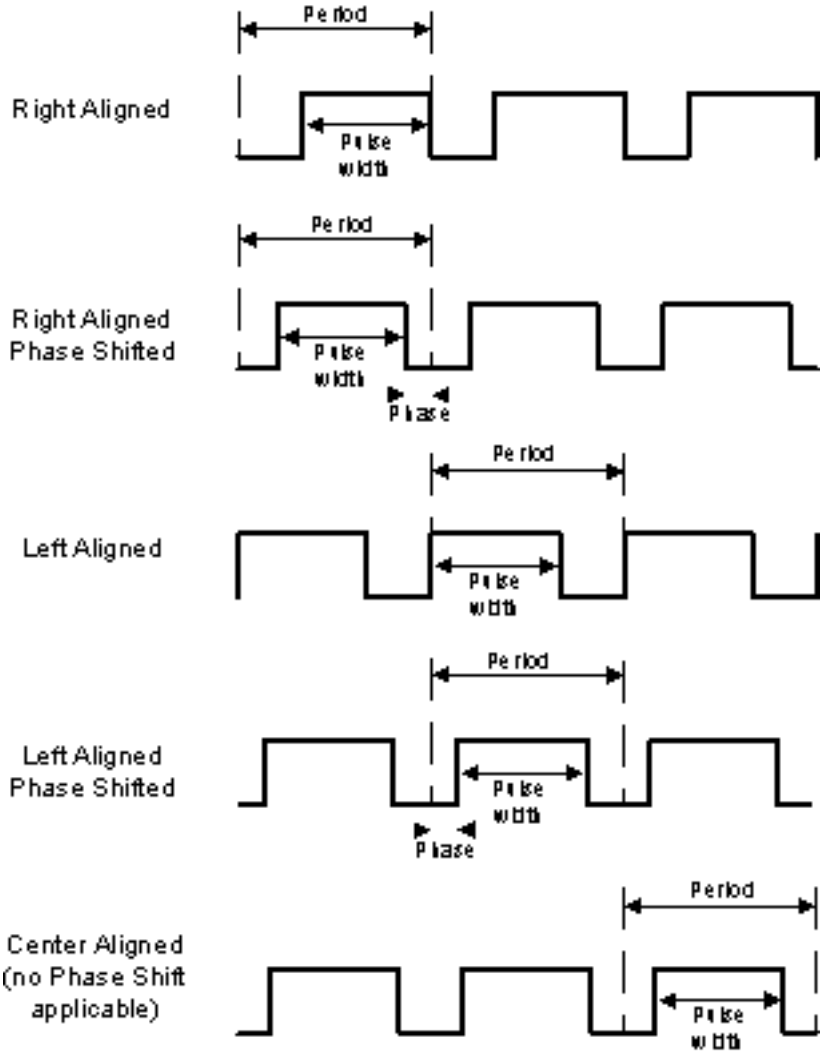
In Sync Mode, the following parameter settings must be the same for each participating PWM16HW block:

- Period
- Alignment
- ClockScaler

In Sync Mode, the outputs of slave blocks have phase shift with respect to the master block:

- In left alignment, the start point of the slave's internal counter is phase shifted to the right with respect to that of master.
- In right alignment, the start point of the slave's internal counter is phase shifted to the left with respect to that of the master.
- In center alignment, the trough of the slave's internal counter is phase shifted to the left with respect to that of the master.

Figure 5. PWM16HW Alignment and Phase Waveforms



To enable Sync Mode:

- Enable SyncMode parameter for all participating blocks or use PWM16HW_EnableSyncMode() API function.
- Select the master block using the SyncMaster parameter or PWM16HW_SetAsSyncMaster() function.
- Call PWM16HW_EnableSyncGlobal() API function.

DC and AC Electrical Characteristics

The following values are indicative of expected performance and based on initial characterization data.

Table 1. AC Parameters and Specifications

AC Parameter	Description	Specification		Units
		Min	Max	
Clock	Input frequency.	24	48	MHz
ClockScaler	Input frequency scaler.	1	256	
Period	Period value.	0	65535	
PulseWidth	Pulse width value.	0	65535	
Fout	Fout is the PWM16HW User Module output frequency. For these calculations, ClockScalerMIN value is taken as 1 for maximum estimation and ClockScalerMAX = 256 for minimum estimation.	$\frac{Clock_{min}}{ClockScaler_{max} \cdot Period}$	$\frac{Clock_{max}}{ClockScaler_{min} \cdot Period}$	MHz

Placement

The PWM16HW User Module occupies one of the possible modulator blocks: MOD0, MOD1, MOD2, or MOD3.

Parameters and Resources

This parameter allows the input clock (48 MHz or 24 MHz) to be scaled down by a factor of ClockScaler. Allowed values are between 1 and 256.

Period

This parameter sets the period of the counter. Allowed values are between 0 and 216-1. The period is loaded into the Period register. The value can be later modified using the PWM16HW_SetPeriod() API function.

PulseWidth

This parameter sets the pulse width of the PWM16HW output. Allowed values are between 0 and 216-1. The value can be modified using the API.

InterruptType

This parameter sets the interrupt trigger type. The interrupt can be set so that it triggers on the rising edge of the output signal or on the terminal count of the Counter register. Some choices are unavailable depending on the selected user module alignment parameter.

Parameter	Description
Compare True	CPU interrupt enabled for edge of the output.
Terminal Count	CPU interrupt enabled for end of the period (valid for right and left alignment only).
Terminal Count Low Point	CPU interrupt enabled for end of the period at lowest point (valid for center alignment only).
Terminal Count High Point	CPU interrupt enabled for end of the period at highest point (valid for center alignment only)

PhaseShift

This field configures the phase shift of the pulse. Allowed values are between 0 and 65535. This allows for staggering of the PWM16HW phase in the left and right aligned configurations. The phase shift is intended for use together with SyncMode.

LowISR

This parameter enables or disables the PWM16HW to generate a low priority DPWM block interrupt.

HighISR

This parameter enables or disables the PWM16HW to generate a high priority DPWM block interrupt.

CompareType

This parameter sets the compare function type to "less than" or "less than or equal."

Parameter	Description
Less Than	PWM16HW output goes high when counter value is less than pulse width value.
Less Than or Equal	PWM16HW output goes high when counter value is less than or equal to pulse width value.

Alignment

The following options are provided.

Parameter	Description
Left	Left alignment PWM16HW output signal to period clock.
Center	Center alignment PWM16HW output signal to period clock.
Right	Right alignment PWM16HW output signal to period clock.

SyncMode

This parameter sets the individual PWM16HW modulators to be synchronous with other DPWM blocks to enable synchronization of the independent modulated signals to control the phases with

respect to one another. In this mode, all of the PWM16HW modulators are required to be configured with the same frequency, alignment, and period. The following options are provided:

Parameter	Description
Disable	Disables synchronization of the independent modulated signals.
Enable	Enables synchronization of the independent modulated signals.

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 PWM16HW_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 PWM16HW 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 before the call if those values are required after the call. This "registers are volatile" policy was selected for efficiency 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 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.

PWM16HW_Start

Description:

Starts PWM16HW operation. This starts the user module, but if the user module is used in an LED application, it does not turn on the LEDs. Until started, PWM16HW output asserts high.

C Prototype:

```
void PWM16HW_Start(void);
```

Assembler:

```
lcall PWM16HW_Start
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

PWM16HW_Stop

Description:

Stops PWM16HW operation. The output of the PWM16HW is high when the user module is stopped.

C Prototype:

```
void PWM16HW_Stop(void);
```

Assembler:

```
lcall PWM16HW_Stop
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

PWM16HW_EnableHPInt

Description:

Enables the PWM16HW to generate a high priority MOD interrupt. The high priority MOD interrupt is shared between all four modulators.

C prototype:

```
void PWM16HW_EnableHPInt(void);
```

Assembler:

```
lcall PWM16HW_EnableHPInt
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

PWM16HW_DisableHPInt

Description:

Disables the PWM16HW to generate a high priority MOD interrupt. The high priority MOD interrupt is shared between all four modulators.

C prototype:

```
void PWM16HW_DisableHPInt(void);
```

Assembler:

```
lcall PWM16HW_DisableHPInt
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

PWM16HW_EnableLPInt

Description:

Enables the PWM16HW to generate a low priority MOD interrupt. The low priority MOD interrupt is shared between all four modulators.

C prototype:

```
void PWM16HW_EnableLPInt(void);
```

Assembler:

```
lcall PWM16HW_EnableLPInt
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

PWM16HW_DisableLPInt

Description:

Disables the PWM16HW to generate a low priority MOD interrupt. The low priority MOD interrupt is shared between all four modulators.

C prototype:

```
void PWM16HW_DisableLPInt(void);
```

Assembler:

```
lcall PWM16HW_DisableLPInt
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

PWM16HW_EnableHPIntGlobal

Description:

Enables the high priority MOD interrupt operation. The high priority MOD interrupt is shared between all four modulators.

C prototype:

```
void PWM16HW_EnableHPIntGlobal(void);
```

Assembler:

```
lcall PWM16HW_EnableHPIntGlobal
```

Parameters:

None.

Return Value:

None.

Side Effects:

Calling this function takes control over the interrupt related to all four modulators.

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

PWM16HW_DisableHPIntGlobal

Description:

Disables the high priority MOD interrupt operation. The high priority MOD interrupt is shared between all four modulators.

C prototype:

```
void PWM16HW_DisableHPIntGlobal(void);
```

Assembler:

```
lcall PWM16HW_DisableHPIntGlobal
```

Parameters:

None.

Return Value:

None.

Side Effects:

Calling this function takes control over the interrupt related to all four modulators.

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

PWM16HW_EnableLPIntGlobal

Description:

Enables the low priority MOD interrupt operation. The low priority MOD interrupt is shared between all four modulators.

C prototype:

```
void PWM16HW_EnableLPIntGlobal(void);
```

Assembler:

```
lcall PWM16HW_EnableLPIntGlobal
```

Parameters:

None.

Return Value:

None.

Side Effects:

Calling this function takes control over the interrupt related to all four modulators.

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

PWM16HW_DisableLPIntGlobal

Description:

Disables the low priority MOD interrupt operation. The low priority MOD interrupt is shared between all four modulators.

C prototype:

```
void PWM16HW_DisableLPIntGlobal(void);
```

Assembler:

```
lcall PWM16HW_DisableLPIntGlobal
```

Parameters:

None.

Return Value:

None.

Side Effects:

Calling this function takes control over the interrupt related to all four modulators.

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

PWM16HW_EnableSyncMode

Description:

Enables the PWM16HW to participate in SyncMode operation.

C prototype:

```
void PWM16HW_EnableSyncMode(void);
```

Assembler:

```
lcall PWM16HW_EnableSyncMode
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

PWM16HW_DisableSyncMode

Description:

Disables the PWM16HW from participating in SyncMode operation.

C prototype:

```
void PWM16HW_DisableSyncMode(void);
```

Assembler:

```
lcall PWM16HW_DisableSyncMode
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

PWM16HW_SetAsSyncMaster

Description:

Sets the PWM16HW block to be master. For correct SyncMode operation, two or more MOD blocks must have the same period and clock scaler.

C prototype:

```
void PWM16HW_SetAsSyncMaster(void);
```

Assembler:

```
lcall PWM16HW_SetAsSyncMaster
```

Parameters:

None.

Return Value:

None.

Side Effects:

This function affects settings shared by all four modulators.

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

PWM16HW_EnableSyncGlobal

Description:

Enables global synchronous operation. Note that changing the period and phase shift is allowed only when global synchronization is disabled.

C prototype:

```
void PWM16HW_EnableSyncGlobal(void);
```

Assembler:

```
lcall PWM16HW_EnableSyncGlobal
```

Parameters:

None.

Return Value:

None.

Side Effects:

This function affects settings shared by all four modulators.

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

PWM16HW_DisableSyncGlobal

Description:

Disables global synchronous operation. Note that changing the period and phase shift is allowed only when global synchronization is disabled.

C prototype:

```
void PWM16HW_DisableSyncGlobal(void);
```

Assembler:

```
lcall PWM16HW_DisableSyncGlobal
```

Parameters:

None.

Return Value:

None.

Side Effects:

This function affects settings shared by all four modulators.

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

PWM16HW_SetClockScaler

Description:

Writes the Clock Scaler with a value from 1 to 256. The input clock (48 MHz or 24 MHz) is scaled down by a factor of ClockScaler.

C Prototype:

```
void PWM16HW_SetClockScaler(WORD wClockScaler);
```

Assembly:

```
mov X, [wClockScaler] ; MSB
mov A, [wClockScaler+1] ; LSB
lcall PWM16HW_SetClockScaler
```

Parameters:

wClockScaler: A value from 1 to 256.

Return Value:

None.

Side Effects:

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

PWM16HW_SetAlignment

Description:

Selects an alignment value.

C Prototype:

```
void PWM16HW_SetAlignment(BYTE bAlignment);
```

Assembly:

```
mov A, [bAlignment]
lcall PWM16HW_SetAlignment
```

Parameters:

Parameter	Value	Description
PWM16HW_LEFT_ALIGNMENT	0x00	Left alignment to period clock.
PWM16HW_CENTER_ALIGNMENT	0x04	Center alignment (with even period and even duty cycles) to period clock.
PWM16HW_RIGHT_ALIGNMENT	0x08	Right alignment to period clock.

bAlignment: This parameter lets the user select PWM output waveform alignment. The following options are provided:

Return Value:

None.

Side Effects:

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

PWM16HW_SetPhaseShift

Description:

This API configures the phase of the pulse. This phase control allows for staggering of the PWM phase in the left and right aligned configurations.

C Prototype:

```
void PWM16HW_SetPhaseShift(WORD wPhaseShift);
```

Assembly:

```
mov X, [wPhaseShift]
mov A, [wPhaseShift+1]
lcall PWM16HW_SetPhaseShift
```

Parameters:

wPhaseShift: Allowed values for this field are between zero and 216-1.

Return Value:

None.

Side Effects:

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

PWM16HW_SetPeriod

Description:

Writes the Period register with the period value. The period value is transferred from the Period register to the Counter register immediately, if the PWM16HW is stopped.

C Prototype:

```
void PWM16HW_SetPeriod(WORD wPeriod);
```

Assembly:

```
mov X, [wPeriod]
mov A, [wPeriod+1]
lcall PWM16HW_SetPeriod
```

Parameters:

wPeriod: Period value is a value from 0 to 216-1.

Return Value:

None.

Side Effects:

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

PWM16HW_SetPulseWidth

Description:

Writes the Pulse Width register with the pulse width value. Writing the Pulse Width register while the counter is active changes the duty cycle of the output. This may cause the output to glitch or change inadvertently.

C Prototype:

```
void PWM16HW_SetPulseWidth(WORD wPulseWidth);
```

Assembly:

```
mov X, [wPulseWidth]
mov A, [wPulseWidth+1]
lcall PWM16HW_SetPulseWidth
```

Parameters:

wPulseWidth: Pulse width value is the value from 0 to the period value.

Return Value:

None.

Side Effects:

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

Sample Firmware Source Code

The C code illustrated here shows you how to use the PWM16HW User Module:

```
PWM16HW_SetPeriod(4095);
PWM16HW_SetPulseWidth(2048);
PWM16HW_EnableHPInt();
PWM16HW_EnableHPIntGlobal();
PWM16HW_Start();
```

The same code in assembly is:

```
mov   X, 0Fh
mov   A, FFh
call  PWM16HW_SetPeriod
mov   X, 08h
mov   A, 00h
call  PWM16HW_SetPulseWidth
call  PWM16HW_EnableHPInt
call  PWM16HW_EnableHPIntGlobal
call  PWM16HW_Start
```

Configuration Registers

Table 2. PWM16HW_PCF_REG

Bit	7	6	5	4	3	2	1	0
Value	Programmable Clock Frequency Scaler							

Programmable Clock Frequency Scaler determines the clock scaler for the PWM block. The value of this register is determined by the choice made for the ClockScaler parameter in the user module parameters of the Device Editor as ClockScaler-1. This value can also be changed by the PWM16HW_SetClockScaler() API.

Table 3. PWM16HW_PDH_REG (MSB), PWM16HW_PDL_REG (LSB)

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

Period determines the period value for the PWM16HW block. The value of this register is determined by the choice made for the Period parameter in the user module parameters of the Device Editor. This value can also be changed by the PWM16HW_SetPeriod() API.

Table 4. PWM16HW_PWH_REG (MSB), PWM16HW_PWL_REG (LSB)

Bit	7	6	5	4	3	2	1	0
LSB	Pulse Width Register (LSB)							
MSB	Pulse Width Register (MSB)							

Pulse Width determines the pulse width value for the PWM16HW block. The value of this register is determined by the choice made for the PulseWidth parameter in the user module parameters of the Device Editor. This value can also be changed by the PWM16HW_SetPulseWidth() API.

Table 5. PWM16HW_PCH_REG (MSB), PWM16HW_PCL_REG (LSB)

Bit	7	6	5	4	3	2	1	0
LSB	Phase Shift Register (LSB)							
MSB	Phase Shift Register (MSB)							

Phase Shift determines the phase shift value for the PWM16HW block. The value of this register is determined by the choice made for the PhaseControl parameter in the user module parameters of the Device Editor. This value can also be changed by the PWM16HW_SetPhaseShift() API.

Table 6. PWM16HW_PCFG_REG

Bit	7	6	5	4	3	2	1	0
Value	0	IntMode	0	0	Align[1:0]		CompType	IntType

IntType selects interrupt on edge of the output or on the end of period. IntMode selects interrupt on the end of the period at lowest point or at highest point (valid for center alignment only). The value of these bits is determined by the choice made for the InterruptType parameter in the user module parameters of the Device Editor.

CompType determines the PWM16HW compare type. The value of this bit is determined by the choice made for the CompareType parameter in the user module parameters of the Device Editor.

Align[1:0] determines the PWM16HW pulse alignment configuration. The value of these bits is determined by the choice made for the Alignment parameter in the user module parameters of the Device Editor. This value can also be changed by the PWM16HW_SetAlignment() API.

Table 7. PWM16HW_GCFG_REG

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

Enable turns on the PWM16HW block. The value of this bit can be changed by the PWM16HW_Start() and PWM16HW_Stop() APIs.

Version History

Version	Originator	Description
1.0	DHA	Initial version

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 © 2009-2012 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.