16-Bit Hardware Density Modulated PWM Datasheet DMM16HW V 1.0

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

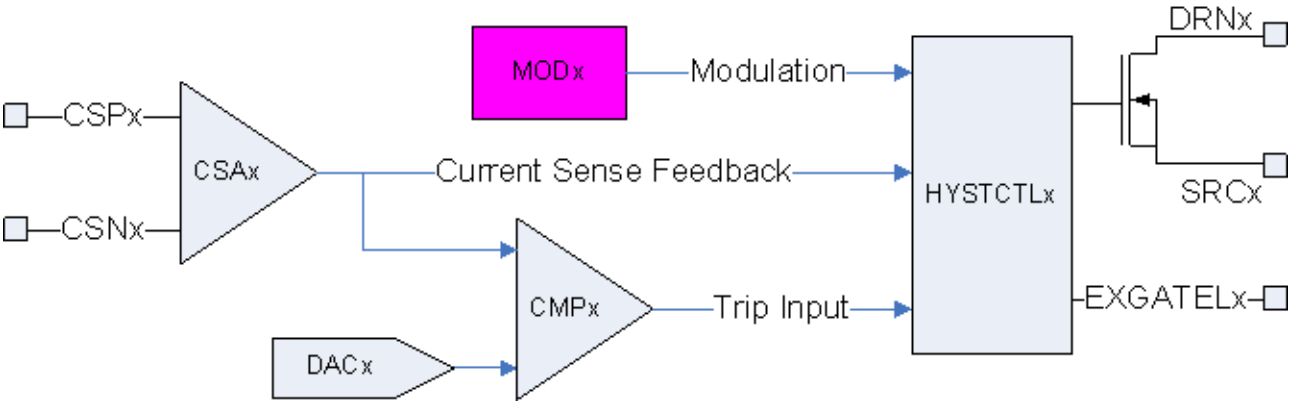
Resources	PSoC® Blocks	API Memory (Bytes)		Pins (per External I/O)
	DRAM	Flash	RAM	
CY8CLED0xD, CY8CLED0xG	1	186	0	1

Features and Overview

- Programmable dimming resolution
- Programmable output frequencies
- Dedicated DMM module frees PSoC core digital blocks for other uses

The DMM16HW User Module produces a density modulated PWM signal.

Figure 1. DMM16HW Block Diagram



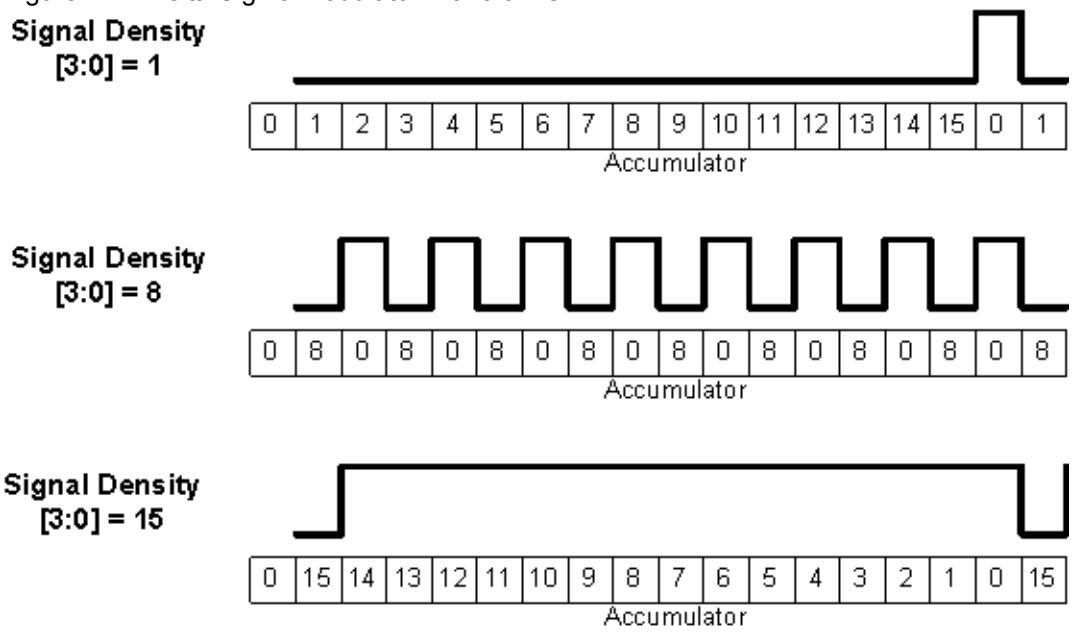
Functional Description

The DMM16HW is a modification of the Pulse Width Modulator (PWM) with dithered output frequency.

The previous block diagram can be divided into two parts; the PWM Block and the Delta-Sigma Modulator (DSM). The output of the PWM block produces the dimming frequency. The adder with the feedback path forms the Delta-Sigma modulator. The carry signal of the adder produces the dither frequency.

The 12 most significant bits of the signal density value are used as a compare value for the PWM block. If the low nibble of the signal density value is zero there is a constant low logical level on the DSM block output and entire module works as a standard 12-bit PWM. If the low nibble is not zero then with each modulator clock pulse this value is added to the contents of the accumulator and the sum is again stored in the accumulator. Sometimes this operation causes the assertion of the carry line which changes the compare type of the PWM block. The following figure depicts the DSM output signal for different low nibble values of signal density (at ModResolution = 4):

Figure 2. Delta-Sigma Modulator Waveforms.



For instance, if the low nibble is one the carry occurs after 16 additions. In this case, during 15 periods the PWM block generates output pulses based on a "less than" compare type and during 1 period it compares using "less than or equal." When the compare type changes the output pulse length is dithered and the average duty ratio is also changed.

In general, if Delta is the low nibble value of the signal density and Compare is the value of the remaining 12 bits then the average duty cycle ratio can be computed using the following equation:

Equation 1

$$DutyCycle = \frac{Compare}{(Period+1)} + \frac{Delta}{(Period+1)2^{DimmingResolution}}$$

The $\frac{Compare}{(Period+1)}$ in this formula is the same as a standard PWM.

The $\frac{Delta}{(Period+1)2^{DimmingResolution}}$ portion can be considered fractional because the resulting duty cycle value is between these two values:

Equation 2

$$\frac{Compare}{(Period+1)} < DutyCycle < \frac{Compare+1}{(Period+1)}$$

The DSM output frequency depends on the SignalDensity [3:0] (delta) value. If this frequency is too low a flicker might be visible. The flicker frequency can be estimated using this formula:

Equation 3

$$F_{flicker} = \frac{Clock}{Period+1}\left(\frac{Delta}{2^{DimResolution}}, Delta < 2^{DimResolution-1}\right)$$

Equation 4

$$F_{flicker} = \frac{Clock}{Period+1}\left(\frac{Delta-(2^{DimResolution-1})}{2^{DimResolution}}, Delta > 2^{DimResolution-1}\right)$$

Clock is 24 MHz or 48 MHz.

Alignment

The DMM16HW User Module provides three options for output pulse alignment:

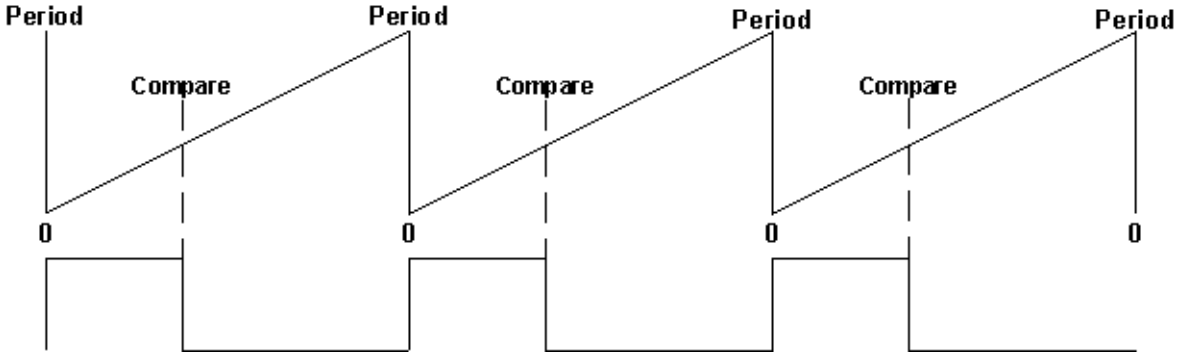
- Right aligned
- Left aligned
- Center aligned

Phase shift allows you to stagger the DMM16HW 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 DMM16HW output, the internal counter of the DPWM counts up from zero to the period value. Every time the counter hits the period value it reloads to zero. The output pulse asserts when the counter is "less than" (or "less than or equal to" depending on the current output of the DSM) the SignalDensity[15:4] (Compare). The period register can be modified with a new value anytime. If the counter reloads to zero, it becomes greater than the period after a new period value is written.

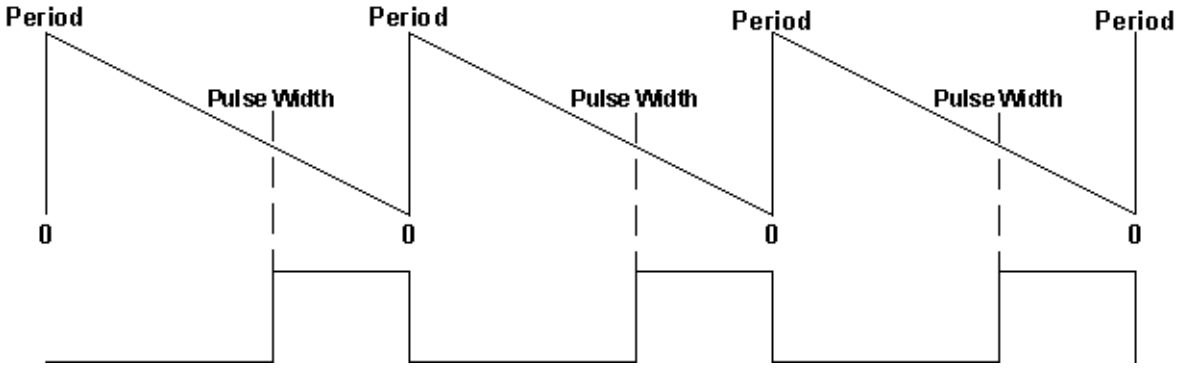
Figure 3. DMM16HW Left Alignment



Right Alignment

For right alignment, the DMM16HW uses a down counter from the period value to zero. Every time the counter hits zero, it reloads to the period value. The output asserts when the counter is "less than" (or "less than or equal to" depending on the current output of the DSM) the Compare. The Period register can be modified with a new value anytime. When the DMM16HW User Module is stopped, writing a value to the period register also changes the value in the counter register. While the DMM16HW User Module is running, writing to the period register does not update the counter register with the new period value until the next reload occurs, following a terminal count.

Figure 4. DMM16HW Right Alignment



Because the terminal count is reached when the count is zero, the period of operation and the period of the output signal is one greater than the value stored in the period register. For the left and right pulse alignment, the duration can be computed with the following equation:

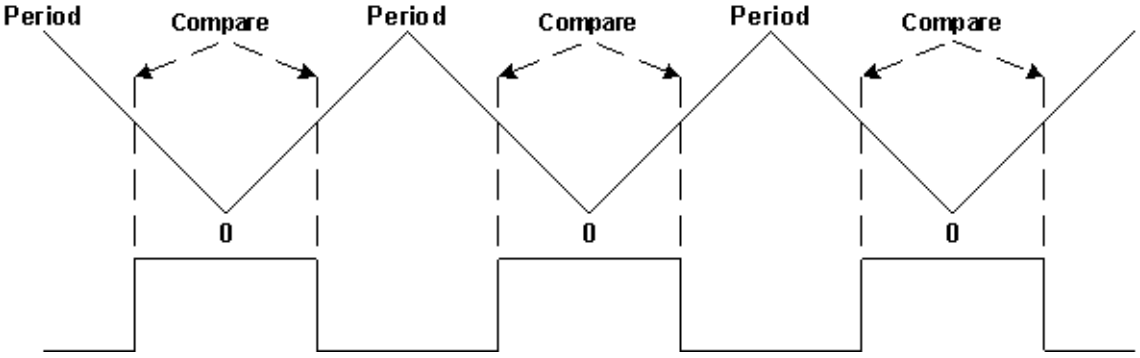
Equation 5

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

Center Alignment

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

Figure 5. DMM16HW Center Alignment



The counter sweep in center alignment is very different from that of left or right alignment. For a given period value, the periodic time of the counter sweep is almost double the period of either left or right alignment. Also, for a given compare value, the width of the generated output pulse in center alignment mode is almost double that of either left or right alignment modes.

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

Equation 6

$$OutputPeriod = 2 \cdot (Period + 1)t_{CLK}$$

The average duty cycle in center alignment mode is:

Equation 7

$$DutyCycle = \frac{2 \cdot (Compare - 1)}{2 \cdot Period} + \frac{2 \cdot Delta}{2 \cdot Period \cdot 2^{DimmingResolution}}$$

In the last equation, Delta is the low nibble value of the signal density value and compare is the value of the remaining 12 bits.

The signal density value may be set using the Device Editor or during run time using the SetSignalDensity API. The signal density register is not buffered the same way the counter register is, so changes to the signal density value while the user module is running affect the compare output on the next clock cycle.

Interrupts

The DMM16HW User Module uses slightly different interrupt logic than standard PSoC digital blocks. All the modulator blocks share two interrupt vectors. These shared interrupt vectors are the high priority (HP) interrupt (22h) and the low priority (LP) interrupt (23h). Each modulator block can generate either an LP or an HP interrupt or both. If several user modules use the same interrupt vector then an interrupt dispatcher is used to call the interrupt service routine of each of the user modules. 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 to trigger, check the DPWMINTFLAG register.

There are several user module parameters and API functions that affect interrupts. To use DMM16HW interrupts:

- Enable the LowISR parameter, the HighISR parameter, or both parameters in the Device Editor. This enables the interrupt and includes this module's ISR calls in the interrupt dispatcher.
- Call the DMM16HW_EnableLPIntGlobal() function, the DMM16HW_EnableHPIntGlobal() function, or both, to enable the corresponding interrupt vectors.
- 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 with the InterruptType parameter using the Device Editor.

Enable or disable the interrupt at run time using the DMM16HW_EnableHPInt(), DMM16HW_DisableHPInt(), DMM16HW_EnableLPInt(), and DMM16HW_DisableLPInt() APIs. You can only use these functions if the corresponding LowISR or HighISR parameter is enabled in the Device Editor.

Sync Mode

Sync Mode is a scheme where two or more DMM16HW user modules can operate synchronously. The presence of periodic dithering on the output pulses of the DMM16HW blocks is not guaranteed to be coincident in the Sync Mode.

Four, three, or two of the DMM16HW blocks can participate in Sync Mode. Which DMM16HW blocks are participating in Sync Mode is defined by the SyncMode parameter settings and can be changed at run time using the corresponding API functions. In Sync Mode, one of the participating DMM16HW modules is specified as the master. The remaining participating blocks are slaves.

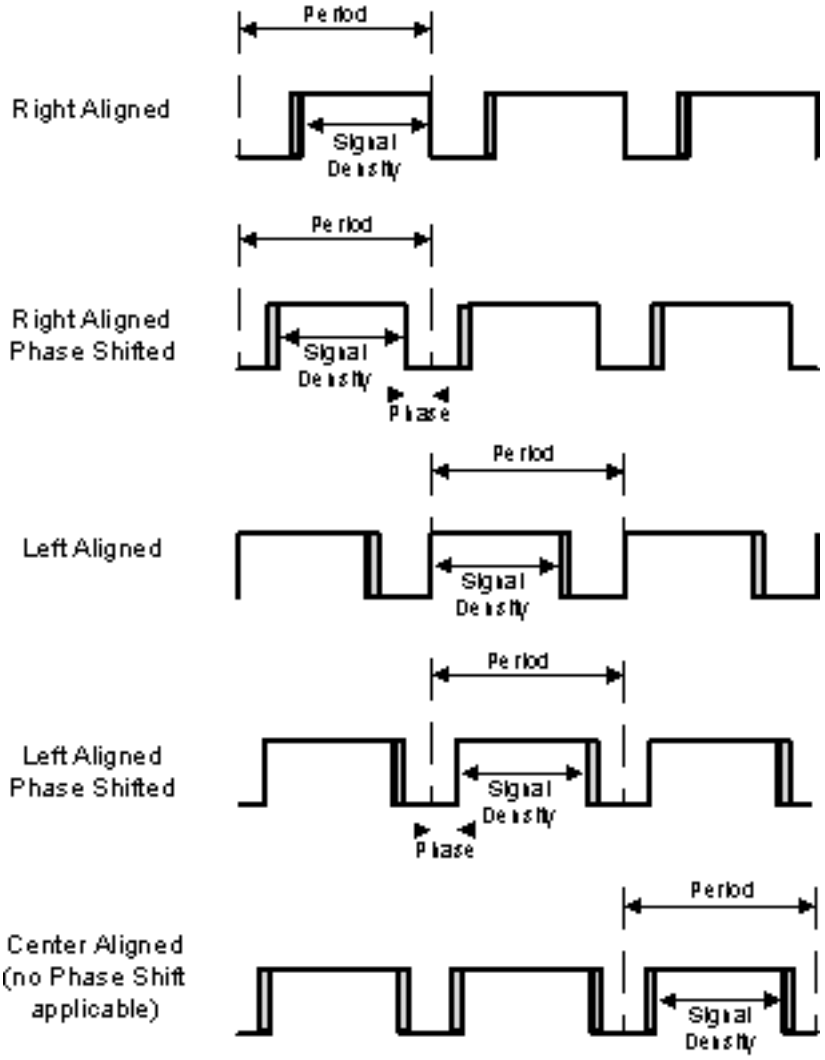
In Sync Mode, the following parameter settings must be the same for each participating DMM16HW block:

- 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 6. DMM16HW Alignment and Phase Shift Waveforms



To enable Sync Mode:

- Enable SyncMode parameter for all participating blocks or use DMM16HW_EnableSyncMode() API function;
- Select the master block using the SyncMater parameter or DMM16HW_SetAsSyncMaster() function;
- Call DMM16HW_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
			Max	
Clock	Input frequency.		48	MHz
ClockScaler	Input frequency scaler.	1	256	
Res_{DSM}	DSM Modulator resolution.	1	4	
F_{DSM}	Delta-Sigma Modulator input frequency (terminal count).	$\frac{Clock_{min}}{ClockScaler_{max} \cdot Period}$	$\frac{Clock_{min}}{ClockScaler_{max}}$	MHz
F_{DITHER}	Dither frequency.	$\frac{1}{16}F_{dsm_{min}}$	$\frac{1}{2}F_{dsm_{max}}$	MHz
F_{out}	Fout is the DPWM block output frequency for DMM mode. For these calculations, ClockScalerMIN value is taken as 1. ClockScalerMAX = 256 for minimum estimate.	$\frac{Clock_{min}}{ClockScaler_{max} \cdot Period}$	$\frac{Clock_{max}}{ClockScaler_{min} \cdot 2}$	MHz

Placement

The DMM16HW User Module can be placed in one of the available DPWM blocks: MOD0, MOD1, MOD2, or MOD3.

Parameters and Resources

ClockScaler

The ClockScaler parameter allows the input clock (24 MHz or 48 MHz) to be scaled down by a factor of ClockScaler. Enter a value between 1 and 256.

Period

The Period parameter is a 12-bit number that configures the period (resolution) of the base PWM. Enter a value from 0 to 4095. This parameter, along with the ClockScaler and SignalDensity parameters, defines the output frequency of the DMM.

SignalDensity

The SignalDensity parameter is a 16-bit value that configures the signal density of the DMM. Enter a value from 0 to 65535. The lower nibble of the LSB (lower 4 bits) of the 16-bit value sets the dither signal density and the remaining 12 bits determine the DMM16HW signal density. This parameter also affects the flicker frequency according to Equations 3 and 4.

ModResolution

The ModResolution parameter is a 1, 2, 3, or 4-bit value for the resolution of the DSM Modulator. If the ModResolution = 4 then the low nibble of the SignalDensity value is used as dither signal density for the DSM. If ModResolution < 4 then unused high order bits of that nibble are ignored. The average signal density also depends on this parameter value according to Equations 1 or 7. This parameter affects the flicker frequency according to Equations 3 and 4.

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 Count register. The Terminal Count Low Point and Terminal Count High Point parameter choices are not available unless center alignment mode is chosen in the Alignment parameter.

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

PhaseShift

This parameter configures the phase shift of the pulse. Enter a value from 0 to 65535. PhaseShift allows staggering of the DMM16HW phase in the left and right alignment configurations. SyncMode must be enabled to use PhaseShift.

LowISR

This parameter enables the DMM16HW to generate a low priority MOD interrupt. The low priority MOD interrupt is shared by all DPWM blocks.

HighISR

This parameter enables the DMM16HW to generate a high priority MOD interrupt. The high priority MOD interrupt is shared by all DPWM blocks. All high interrupts have the same priority, therefore, if there are four MOD blocks each with a high ISR, the first interrupt in is serviced first.

Alignment

DMM16HW provides three options for output pulse alignment: right aligned, left aligned, and center aligned. For each of these alignments, the internal counter of the UM block counts in a different direction.The following options are provided:

Parameter	Description
Left	Left alignment of the DMM16HW output signal to the period clock.
Center	Center alignment of the DMM16HW output signal to the period clock.
Right	Right alignment of the DMM16HW output signal to the period clock.

SyncMode

All of the DPWM blocks can be synchronized so that the independent modulated signals can be phase locked. When you enable SyncMode, all DMM16HW modulators must be configured with the same frequency, alignment, and period.

There are four DPWM blocks in the logic core. Two or more of the DPWM blocks can participate in SyncMode. Which DPWM blocks participate in SYNC is user defined by setting DPWMSYNC[7:4]. In SyncMode, one of the participating DPWM blocks is specified as the master. The remaining participating DPWM blocks are slaves. Use the SYNC_MASTER_SEL bits in the DPWMSYNC register to designate the master. If less than four DPWM blocks are participating in SyncMode, the remaining DPWM blocks can be operated independently.

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 you 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 DMM16HW_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 DMM16HW 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 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 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.

DMM16HW_Start

Description:

Starts DMM16HW operation. Until started, the output of the DMM16HW asserts high.

C Prototype:

```
void DMM16HW_Start(void);
```

Assembler:

```
lcall DMM16HW_Start
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

DMM16HW_Stop

Description:

Stops DMM16HW operation. The output of the DMM16HW is high when the user module is stopped.

C Prototype:

```
void DMM16HW_Stop(void);
```

Assembler:

```
lcall DMM16HW_Stop
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

DMM16HW_EnableHPInt

Description:

Enables the DMM16HW to generate a high priority MOD interrupt. The high priority MOD interrupt is shared by all DPWM blocks.

C prototype:

```
void DMM16HW_EnableHPInt(void);
```

Assembler:

```
lcall DMM16HW_EnableHPInt
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

DMM16HW_DisableHPInt

Description:

Disables the high priority MOD interrupt. The high priority MOD interrupt is shared by all DPWM blocks.

C prototype:

```
void DMM16HW_DisableHPInt(void);
```

Assembler:

```
lcall DMM16HW_DisableHPInt
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

DMM16HW_EnableLPInt

Description:

Enables the DMM16HW to generate a low priority MOD interrupt. The low priority MOD interrupt is shared by all DPWM blocks.

C prototype:

```
void DMM16HW_EnableLPInt(void);
```

Assembler:

```
lcall DMM16HW_EnableLPInt
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

DMM16HW_DisableLPInt

Description:

Disables the DMM16HW low priority MOD interrupt. The low priority MOD interrupt is shared by all DPWM blocks.

C prototype:

```
void DMM16HW_DisableLPInt(void);
```

Assembler:

```
lcall DMM16HW_DisableLPInt
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

DMM16HW_EnableHPIntGlobal

Description:

Enables the high priority MOD interrupt operation. The high priority MOD interrupt is shared between all four PSoC DPWM blocks.

C prototype:

```
void DMM16HW_EnableHPIntGlobal(void);
```

Assembler:

```
lcall DMM16HW_EnableHPIntGlobal
```

Parameters:

None.

Return Value:

None.

Side Effects:

Calling this function takes control over interrupt that is related to all four PSoC DPWM blocks.

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

DMM16HW_DisableHPIntGlobal

Description:

Disables the high priority MOD interrupt operation. The high priority MOD interrupt is shared between all four PSoC DPWM blocks.

C prototype:

```
void DMM16HW_DisableHPIntGlobal(void);
```

Assembler:

```
lcall DMM16HW_DisableHPIntGlobal
```

Parameters:

None.

Return Value:

None.

Side Effects:

Calling this function takes control over interrupt that is related to all four PSoC DPWM blocks.

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

DMM16HW_EnableLPIntGlobal

Description:

Enables the low priority MOD interrupt operation. The low priority MOD interrupt is shared between all four PSoC DPWM blocks.

C prototype:

```
void DMM16HW_EnableLPIntGlobal(void);
```

Assembler:

```
lcall DMM16HW_EnableLPIntGlobal
```

Parameters:

None.

Return Value:

None.

Side Effects:

Calling this function takes control over interrupt that is related to all four PSoC DPWM blocks.

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

DMM16HW_DisableLPIntGlobal

Description:

Disables the low priority MOD interrupt operation. The low priority MOD interrupt is shared between all four PSoC DPWM blocks.

C prototype:

```
void DMM16HW_DisableLPIntGlobal(void);
```

Assembler:

```
lcall DMM16HW_DisableLPIntGlobal
```

Parameters:

None.

Return Value:

None.

Side Effects:

Calling this function takes control over interrupt that is related to all four PSoC DPWM blocks.

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

DMM16HW_EnableSyncMode

Description:

Enables the DMM16HW participation in SyncMode operation. All of the DPWM blocks can be synchronized so that the independent modulated signals can be phase locked. When you enable SyncMode, all DMM16HW modulators must be configured with the same frequency and period. Use the SYNC_MASTER_SEL bits in the DPWMSYNC register to designate the master.

C prototype:

```
void DMM16HW_EnableSyncMode(void);
```

Assembler:

```
lcall DMM16HW_EnableSyncMode
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

DMM16HW_DisableSyncMode

Description:

Disables the DMM16HW participation in SyncMode operation.

C prototype:

```
void DMM16HW_DisableSyncMode(void);
```

Assembler:

```
lcall DMM16HW_DisableSyncMode
```

Parameters:

None.

Return Value:

None.

Side Effects:

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

DMM16HW_SetAsSyncMaster

Description:

Sets this DMM16HW block as the master. For correct synchronous operation, all synchronized MOD blocks must have the same period and clock scaler.

C prototype:

```
void DMM16HW_SetAsSyncMaster(void);
```

Assembler:

```
lcall DMM16HW_SetAsSyncMaster
```

Parameters:

None.

Return Value:

None.

Side Effects:

This function affects settings shared by all four MOD blocks.

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

DMM16HW_EnableSyncGlobal

Description:

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

C prototype:

```
void DMM16HW_EnableSyncGlobal(void);
```

Assembler:

```
lcall DMM16HW_EnableSyncGlobal
```

Parameters:

None.

Return Value:

None.

Side Effects:

This function affects settings shared by all four MOD blocks.

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

DMM16HW_DisableSyncGlobal

Description:

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

C prototype:

```
void DMM16HW_DisableSyncGlobal(void);
```

Assembler:

```
lcall DMM16HW_DisableSyncGlobal
```

Parameters:

None.

Return Value:

None.

Side Effects:

This function affects settings shared by all four MOD blocks.

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

DMM16HW_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 DMM16HW_SetClockScaler(WORD wClockScaler);
```

Assembly:

```
mov X, [wClockScaler] ; MSB
mov A, [wClockScaler+1] ; LSB
lcall DMM16HW_SetClockScaler
```

Parameters:

wClockScaler: Is a value from 1 to 256.

Return Value:

None.

Side Effects:

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

DMM16HW_SetAlignment

Description:

Selects an alignment value.

C Prototype:

```
void DMM16HW_SetAlignment(BYTE bAlignment);
```

Assembly:

```
mov A, [bAlignment]
lcall DMM16HW_SetAlignment
```

Parameters:

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

Parameter	Value	Description
DMM16HW_LEFT_ALIGNMENT	0x00	Left alignment to period clock.
DMM16HW_CENTER_ALIGNMENT	0x04	Center (with even period and even duty cycles) alignment to period clock.
DMM16HW_RIGHT_ALIGNMENT	0x08	Right alignment to period clock.

Return Value:

None.

Side Effects:

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

DMM16HW_SetPhaseShift

Description:

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

C Prototype:

```
void DMM16HW_SetPhaseShift(WORD wPhaseShift);
```

Assembly:

```
mov X, [wPhaseShift]
mov A, [wPhaseShift+1]
lcall DMM16HW_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.

DMM16HW_SetPeriod

Description:

Writes the Period register with the period value. When the DMM16HW is stopped, writing a value to the Period register also changes the value in the Counter register. While the DMM16HW is running, writing a new value to the Period register does not update the Counter register with the new period value until the next reload occurs, following the terminal count.

C Prototype:

```
void DMM16HW_SetPeriod(WORD wPeriod);
```

Assembly:

```
mov X, [wPeriod]
mov A, [wPeriod+1]
lcall DMM16HW_SetPeriod
```

Parameters:

wPeriod: Period value is a value from 0 to 4095.

Return Value:

None.

Side Effects:

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

DMM16HW_SetSignalDensity

Description:

Writes the Signal Density register with the signal density value.

C Prototype:

```
void DMM16HW_SetSignalDensity(WORD wSignalDensity);
```

Assembly:

```
mov X, [wSignalDensity]
mov A, [wSignalDensity +1]
lcall DMM16HW_SetSignalDensity
```

Parameters:

wSignalDensity: Signal density contains values from 0 to 65535.

Return Value:

None.

Side Effects:

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

DMM16HW_SetModResolution

Description:

Sets the resolution of the modulator.

C Prototype:

```
void DMM16HW_SetModResolution(BYTE bModResolution);
```

Assembly:

```
mov A, [bModResolution]
lcall DMM16HW_SetModResolution
```

Parameters:

bModResolution: One of the following four values is provided:

Parameter	Value	Modulator Resolution
DMM16HW_RESOLUTION_1BIT	0x30	1 bit
DMM16HW_RESOLUTION_2BIT	0x20	2 bit
DMM16HW_RESOLUTION_3BIT	0x10	3 bit
DMM16HW_RESOLUTION_4BIT	0x00	4 bit

The resolution value is passed in the A register.

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 DMM16HW User Module:

```
DMM16HW_SetPeriod(2047);
DMM16HW_SetSignalDensity((1024<<4)+3);
DMM16HW_EnableHPInt();
DMM16HW_EnableHPIntGlobal();
DMM16HW_Start();
```

The same code in assembly is:

```
mov   X, 07h
mov   A, FFh
call  DMM16HW_SetPeriod
mov   X, >((1024<<4)+3)
mov   A, <((1024<<4)+3)
call  DMM16HW_SetSignalDensity
call  DMM16HW_EnableHPInt
call  DMM16HW_EnableHPIntGlobal
call  DMM16HW_Start
```

Configuration Registers

Table 2. DMM16HW_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 DMM 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 be changed by the DMM16HW_SetClockScaler() API.

Table 3. DMM16HW_PDH_REG (MSB), DMM16HW_PDL_REG (LSB)

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

Period determines the period value for the DMM16HW 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 be changed by the DMM16HW_SetPeriod() API.

Table 4. DMM16HW_PWH_REG (MSB), DMM16HW_PWL_REG (LSB)

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

Signal Density determines the pulse width value for the DMM16HW block. The value of this register is determined by the choice made for the SignalDensity parameter in the user module parameters of the Device Editor. This value can be changed by the DMM16HW_SetSignalDensity() API.

Table 5. DMM16HW_PCH_REG (MSB), DMM16HW_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 DMM16HW 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 be changed by the DMM16HW_SetPhaseShift() API.

Table 6. DMM16HW_PCFG_REG

Bit	7	6	5	4	3	2	1	0
Value	0	IntMode	ModResolution		Align		0	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.

ModResolution determines the DMM16HW pulse alignment configuration. The value of these bits is determined by the choice made for the parameter with the same name in the user module parameters of the Device Editor. This value can be changed by the DMM16HW_SetModResolution() API.

Align determines the DMM16HW 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 be changed by the DMM16HW_SelectAlignment() API.

Table 7. DMM16HW_GCFG_REG

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

Enable turns on the DMM16HW block. The value of this bit can be changed by the DMM16HW_Start() and DMM16HW_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.