16-Bit Pulse Width Modulator Datasheet PWM16 V 2.5

Copyright © 2000-2014 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C29/27/24/22/21xxx, CY8C23x33, CY7C64215/603xx, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8CTMA140, CY8C21x45, CY8C22x45, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx						
16-bit	2	0	0	89	0	1

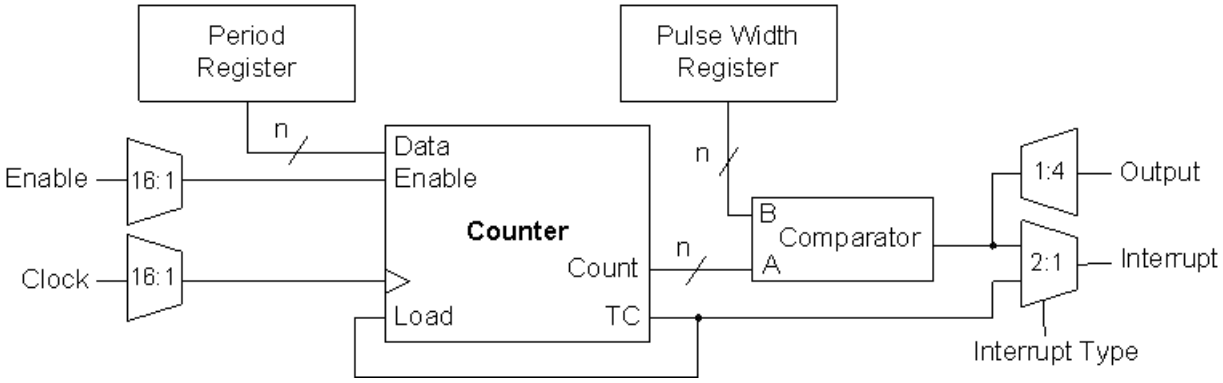
For one or more fully configured, functional example projects that use this user module go to www.cypress.com/psocexampleprojects.

Features and Overview

- 16-bit general purpose pulse width modulator uses two PSoC blocks
- Source clock rates up to 48 MHz
- Automatic reload of period for each pulse cycle
- Programmable pulse width
- Input enables/disables continuous counter operation
- Interrupt option on rising edge of the output or terminal count

The 16-bit PWM User Module is a pulse width modulator with programmable period and pulse width. The clock and enable signals can be selected from several sources. The output signal can be routed to a pin or to one of the global output buses, for internal use by other user modules. An interrupt can be programmed to trigger on the rising edge of the output or when the counter reaches the terminal count condition.

Figure 1. PWM Block Diagram, Data Path width n = 16



Functional Description

The PWM User Module employs two digital PSoC blocks, each contributing 8 bits to the total resolution. To form a 16-bit pulse width modulator, the two consecutive blocks are linked so their internal carry, terminal count and compare signals are synchronously chained. This concatenates the individual Count, Period and Compare registers (data registers DR0, DR1 and DR2, respectively) to provide the required 16-bit resolution.

The PWM API provides functions that may be called from C and assembly to stop and start operation of the Counter and to read and write the various data registers. The data register values may also be established by using the Device Editor. Once started, the Count register is decremented on the rising edge of each clock cycle at which the active-high enable input signal is asserted. The Count register is reloaded with the value in the Period register on the risking clock edge following a terminal count (when the count register reaches zero).

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 1 than the value stored in the Period register. The following equations relate the output of the PWM to the input clock and the value in the Period register.

TOUT = (PeriodValue+1)/FCLOCK

$$FOUT = FCLOCK/(PeriodValue+1) \quad \textbf{Equation 1}$$

Where *F*OUT is The output frequency of the PWM, *T*OUT is the output period of the PWM, *F*CLOCK is frequency of the input clock, and *PeriodValue* is the value entered for the period.

The PWM asserts its output low 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 Count register against that of the PulseWidth register, performing a "Less Than" or "Less Than Or Equal" test depending on an option selected using the Device Editor. The PWM 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 PulseWidth value and the period sets the duty cycle of the output waveform. The duty cycle ratio can be computed using this equation.

For PulseWidthValue < PeriodValue:

Equation 2

$$DutyCycle = \begin{cases} \frac{PulseWidthValue}{PeriodValue + 1}, \text{For } Less\ Than \text{ comparison} \\ \frac{PulseWidthValue + 1}{PeriodValue + 1}, \text{For } Less\ Than\ Or\ Equal\ To \text{ comparison} \end{cases}$$

For PulseWidthValue >= PeriodValue

DutyCycle = 100%

The following table summarizes some special output signal conditions based on the setting of the Period, the PulseWidth, and the comparison operation.

Table 1. Counter Special Output Signal Conditions

Period Register Value	Compare Type	PulseWidth Register Value	Ratio of Pulse-Width High Time to Period
0	Don’t Care	> 0	1.0
0	≤	0	1.0
0	<	0	0.0
> 0	≤	0	1/(Period+1)
> 0	<	0	0.0
Period = PulseWidth	≤	Period = PulseWidth	1.0
Period = PulseWidth	<	Period = PulseWidth	Period/(Period+1)
PulseWidthValue > Period	Don’t Care	PulseWidthValue > Period	1.0

The value of the PulseWidth register may be set using the Device Editor or during run time using the API. No buffering of the PulseWidth register is provided in the way the Period register buffers the Count register before terminal count. Therefore, changes to the PulseWidth register affect the compare output on the next clock cycle, rather than following terminal count. This can produce periods with multiple pulses.

In the CY8C29/27/24/22/21xxx, CY8C23x33, CY7C64215/603xx, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8CTMG300, CY8CTST300, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx device families, the PWM User Module provides the terminal count signal as an auxiliary output. This active-high signal is asserted on the rising edge of the clock cycle following terminal count in which the Count register is loaded from the Period register.

An interrupt can be programmed to occur on terminal count or when the compare becomes true. The comparator output triggers an interrupt on the rising edge of the output signal and the terminal count triggers an interrupt one-half clock cycle before the falling edge of the output signal. This option is set using the Device Editor. Enabling or disabling the interrupt is done at run time using the Counter API. Global interrupts must be enabled before the Counter’s interrupt fires.

Care must be taken when modifying the PulseWidth register since its value, in conjunction with the current count value, determines the PWM’s output state. To prevent a possible premature low assertion of the output signal and potential glitches, the PulseWidth register must be modified after the terminal count condition is detected using the interrupt.

For applications that require a faster duty cycle update interval, the output of the PWM can be routed to a pin where its state is polled. Upon the detection of the output transition from high to low, the PulseWidth can then be updated. Note that if the PulseWidth causes the compare true condition, then the output is asserted high on the next clock.

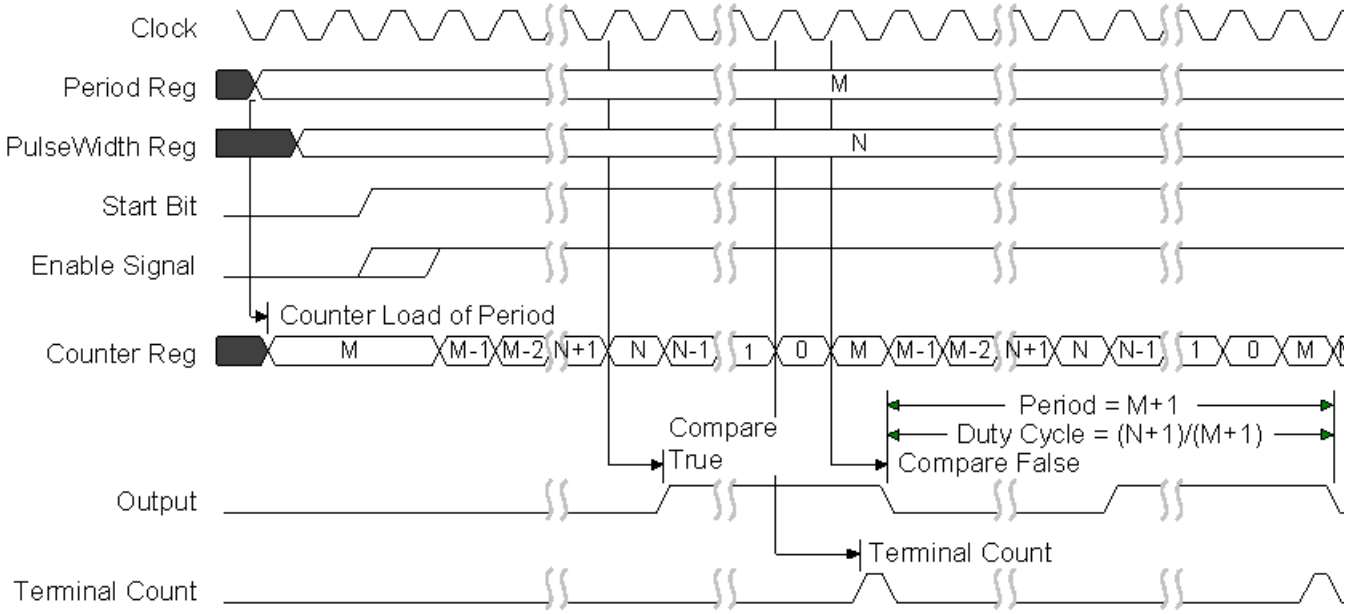
Acquiring the Count register value must be done very carefully. Reading the Count register causes its contents to latch into the PulseWidth register. This causes the output duty cycle to change.

If you need to read the Count register “on-the-fly,” then the ReadCounter() API function can be called. This function temporarily disables the clock, saves the PulseWidth register contents, reads the Count register, reads the PulseWidth register, restores the PulseWidth register, and then restores the clock. See the description for the ReadCounter() function in the Application Programming Interface section for possible side effects.

Timing

PWM operation may be gated On and Off, or clocked by external pins routed to the PWM by the global bus feature of the device.

Figure 2. PWM Timing Diagram



DC and AC Electrical Characteristics

Table 2. PWM DC and AC Electrical Characteristics

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

Electrical Characteristics Notes

1. If the output is routed through 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 PWM consumes one digital PSoC block per 8 bits of resolution. The blocks are all placed consecutively by the Device Editor in order of increasing block number from least-significant byte (LSB) to most significant (the MSB). 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 PWM registers through the API include files. The block names used by the various widths are given in the following table.

Table 3. PWM Symbolic PSoC Block Names

PSoC Blocks	16-Bit PWM
1	PWM16_LSB
2	PWM16_MSB

Parameters and Resources

Clock

The Clock parameter is selected from one of 16 sources. These sources include the 48 MHz oscillator (5.0V operation only), lower frequencies (VC1, VC2, and VC3) divided down from the 24 MHz system clock, other PSoC blocks, and external inputs routed through global inputs and outputs. 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 16 sources. A high input enables continuous count, while a low enable disables count without resetting the counter.

CompareOut

The compare output may be disabled (without interfering with interrupt operations) or connected to any of the row output busses. It is always available as an input to the next higher digital PSoC block and to the analog column clock selection multiplexors, regardless of the setting of this parameter. This parameter appears only for members of the CY8C29/27/24/22/21xxx, CY8C23x33, CY7C64215/603xx, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8CTMG300, CY8CTST300, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx families of PSoC devices.

TerminalCountOut

The terminal count output is an auxiliary Counter output. This parameter allows it to be disabled or connected to any of the row output buses. This parameter appears only for members of the CY8C29/27/24/22/21xxx, CY8C23x33, CY7C64215/603xx, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8CTMG300, CY8CTST300, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx families of PSoC devices.

Period

This parameter sets the period of the counter. Allowed values for PWM8 are between zero and 255. Allowed values for PWM16 are between zero and 2^{16}-1. The period is loaded into the Period register. The effective output waveform period of the PWM16 is the period count + 1. The value may be modified using the API.

PulseWidth

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 so that it triggers on the rising edge of the output signal or on the terminal count of the Counter register. A separate register independently enables the interrupt.

CompareType

This parameter sets the compare function type “Less Than” or “Less Than or Equal To.”

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, CY8C23x33, CY7C64215/603xx, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8CTMG300, CY8CTST300, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx PSoC device families because of various data-path optimizations, particularly those applied to the system buses. 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 must be determined from the following table.

ClockSync Value	Use
Sync to SysClk	Use this setting for any 24 MHz (SysClk) derived input clock source less than 24 MHz. Examples include VC1, VC2, VC3 (when VC3 is driven by SysClk), 32KHz, and digital PSoC blocks with SysClk-based sources. Externally generated clock sources must also use this value to ensure that proper synchronization occurs.
Sync to SysClk*2	Use this setting for any 48 MHz (SysClk*2) based input clock less than 48 MHz.
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, above. It must 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.

InvertEnable

This parameter determines the sense of the enable input signal. When “Normal” is selected, the enable input is active-high. Selecting “Invert” causes the sense to be interpreted as active-low. InvertEnable applies only to the CY8C29/27/24/22/21xxx, CY8C23x33, CY7C64215/603xx, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8CTMG300, CY8CTST300, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx families of PSoC devices.

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 sections 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 functions may leave A and X unchanged, there is no guarantee they will do so in the future.

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

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. The following are the API programming routines provided for PWM16.

PWM16_PERIOD

Description:

Represents the value chosen for the Period field of the PWM16 in the Device Editor. The value can have a range between 0 and 65535.

PWM16_PULSE_WIDTH

Description:

Represents the value chose for the PulseWidth field of the PWM16 in the Device Editor. The value can have a range between 0 and 65535.

PWM16_EnableInt

Description:

Enables the interrupt mode operation.

C Prototype:

```
void  PWM16_EnableInt(void);
```

Assembly:

```
lcall  PWM16_EnableInt
```

Parameters:

None

Return Value:

None

Side Effects:

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

PWM16_DisableInt

Description:

Disables the interrupt mode operation.

C Prototype:

```
void  PWM16_DisableInt(void);
```

Assembly:

```
lcall  PWM16_DisableInt
```

Parameters:

None

Return Value:

None

Side Effects:

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

PWM16_Start

Description:

Starts the PWM16 User Module. If the enable input is high, the counter begins to down count.

C Prototype:

```
void  PWM16_Start(void);
```

Assembly:

```
lcall  PWM16_Start
```

Parameters:

None

Return Value:

None

Side Effects:

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

PWM16_Stop

Description:

Stops the counter operation.

C Prototype:

```
void  PWM16_Stop(void);
```

Assembly:

```
lcall  PWM16_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

The output is reset low and writing to the Period register causes the Counter register to update with the new period value. The A and X registers may be altered by this function.

PWM16_WritePeriod

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 PWM16 is stopped or when the counter reaches the zero count.

C Prototype:

```
void  PWM16_WritePeriod(WORD wPeriod);
```

Assembly:

```
mov   X, [wPeriod]
```

```
mov   A, [wPeriod+1]
lcall  PWM16_WritePeriod
```

Parameters:

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

Return Value:

None

Side Effects:

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

PWM16_WritePulseWidth

Description:

Writes the PulseWidth register with the pulse width value.

C Prototype:

```
void  PWM16_WritePulseWidth(WORD wPulseWidth);
```

Assembly:

```
mov   X, [wPulseWidth]
mov   A, [wPulseWidth+1]
lcall  PWM16_WritePulseWidth
```

Parameters:

wPulseWidth: wPulseWidth value is the value from 0 to the period value. MSB is passed in the X register and LSB 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.

PWM16_wReadPulseWidth

Description:

Reads the PulseWidth register.

C Prototype:

```
WORD  PWM16_wReadPulseWidth();
```

Assembly:

```
lcall  PWM16_wReadPulseWidth
mov   [wPulseWidth], X
mov   [wPulseWidth+1], A
```

Parameters:

None

Return Value:

The Pulse width value is stored in the PulseWidth register. MSB is passed in the X register and LSB is passed in the Accumulator.

Side Effects:

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

PWM16_wReadCounter

Description:

Reads the Counter register.

Note that this function is for applications that must read the Counter register on-the-fly, creating some side effects.

C Prototype:

```
BYTE  PWM16_wReadCounter();
```

Assembly:

```
lcall  PWM16_wReadCounter
mov    [wCounter], X
mov    [wCounter+1], A
```

Parameters:

None

Return Value:

Returns the Counter register value. MSB is passed in the X register and LSB is passed in the Accumulator.

Side Effects:

To read the PWM16 Counter register, the PulseWidth register must be temporarily modified. This could cause the PWM16 Counter register operation to be postponed by one or more counts. In addition, this could result in an inadvertent interrupt condition. 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 PWM16.h file.

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

```
;-------------------------------------------------------------------------
; FILENAME: main.asm
;
; DESCRIPTION:
;
;  This sample shows how to create a 33% duty cycle output pulse.
;
; OVERVIEW:
;
;  The PWM output can be routed to any pin.
;  In this example the PWM output is routed to P0[4].
;  The pin P0[4] has the 33% duty cycle output pulse with frequency = 1,5 kHz.
;
;The following changes need to be made to the default settings in the Device Editor:
;
;   1. Select PWM16 user module.
;   2. The User Module will occupy the space in dedicated system resources.
;   3. Rename User Module's instance name to PWM16.
;   4. Set PWM16's Clock Parameter to VC1.
;   5. Set PWM16's Enable Parameter to High.
;   6. Set PWM16's CompareOut Parameter to Row_0_Output_0.
;   7. Set PWM16's CompareType Parameter to Less Than Or Equal.
;   8. Set PWM16's ClockSync Parameter to SyncSysClk.
;   9. Click on Row_0_Output_0 and connect Row_0_Output_0 to GlobalOutEven_4.
;   10.Select GlobalOutEven_4 for P0[4] in the Pinout.
;
; CONFIGURATION DETAILS:
;
;  1. The clock selected should be 1000 times the required period.
;  2. The UM's instance name must be shortened to PWM16.
;
; 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
;  ------------------------------------------------------------------------
;  PWM16      Name              PWM16                 UM's instance name
;             Clock             VC1
;             Enable            High
```

```
;             CompareOut        Row_0_Output_0
;             TerminalCountOut  Row_0_Output_1
;             Period            0                    The Code changes it.
;             PulseWidth        0                    The Code changes it.
;             CompareType       Less Than Or Equal
;             InterruptType     Terminal Count
;             ClockSync         SyncSysClk
;
;  ------------------------------------------------------------------------
;-------------------------------------------------------------------------
; 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

PWM16_LSB_PERIOD:     equ E7h
PWM16_MSB_PERIOD:     equ 03h
PWM16_LSB_PULSEWIDTH: equ 4ch
PWM16_MSB_PULSEWIDTH: equ 01h

_main:

    ; M8C_EnableGInt               ; Uncomment this line to enable Global Interrupts
    mov   A, PWM16_LSB_PERIOD      ; set period to be 1000 counts of the clock
    mov   X, PWM16_MSB_PERIOD      ; Period is set to 1000 - 1 = 999 (0x3E7).
    lcall PWM16_WritePeriod
    mov   A, PWM16_LSB_PULSEWIDTH  ; set PulseWidth to generate a 33% duty cycle
    mov   X, PWM16_MSB_PULSEWIDTH  ; Pulse Width = 1000/3 - 1 = 332 (0x14C).
    lcall PWM16_WritePulseWidth
    lcall PWM16_Start              ; start the PWM16 - counter will start to
                                   ; count when the enable input is asserted
     ; Insert your main assembly code here.

.terminate:
    jmp .terminate
```

The same code in C is:

```
//-------------------------------------------------------------------------
// FILENAME: main.c
//
// DESCRIPTION:
//
//  This sample shows how to create a 33% duty cycle output pulse.
//
// OVERVIEW:
//
//  The PWM output can be routed to any pin.
//  In this example the PWM output is routed to P0[4].
//  The pin P0[4] has the 33% duty cycle output pulse with frequency = 1,5 kHz.
//
//The following changes need to be made to the default settings in the Device Editor:
```

```
//
//   1. Select PWM16 user module.
//   2. The User Module will occupy the space in dedicated system resources.
//   3. Rename User Module's instance name to PWM16.
//   4. Set PWM16's Clock Parameter to VC1.
//   5. Set PWM16's Enable Parameter to High.
//   6. Set PWM16's CompareOut Parameter to Row_0_Output_0.
//   7. Set PWM16's CompareType Parameter to Less Than Or Equal.
//   8. Set PWM16's ClockSync Parameter to SyncSysClk.
//   9. Click on Row_0_Output_0 and connect Row_0_Output_0 to GlobalOutEven_4.
//   10.Select GlobalOutEven_4 for P0[4] in the Pinout.
//
// CONFIGURATION DETAILS:
//
//  1. The clock selected should be 1000 times the required period.
//  2. The UM's instance name must be shortened to PWM16.
//
// 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
//  ------------------------------------------------------------------------------
//  PWM16      Name              PWM16                  UM's instance name
//             Clock             VC1
//             Enable            High
//             CompareOut        Row_0_Output_0
//             TerminalCountOut  Row_0_Output_1
//             Period            0                     The Code changes it.
//             PulseWidth        0                     The Code changes it.
//             CompareType       Less Than Or Equal
//             InterruptType     Terminal Count
//             ClockSync         SyncSysClk
//
//  ------------------------------------------------------------------------------

/* Code begins here */

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

#define PWM_PERIOD     999
#define PWM_PULSEWIDTH 332

void main(void)
{
    // M8C_EnableGInt ;        // Uncomment this line to enable Global Interrupts
    PWM16_WritePeriod(PWM_PERIOD);          // Set period
    PWM16_WritePulseWidth(PWM_PULSEWIDTH);  // Set pulse width to generate a 33%
                                            //duty cycle
    PWM16_Start();
```

```
    // Insert your main routine code here.
}
```

Configuration Registers

Except where noted, the register specifications given in this section apply to all PSoC device families.

The 16-bit PWM uses two digital PSoC blocks. In placement order from left to right, they are named PWM16_LSB and PWM16_MSB. 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).

Table 4. Function Register, Bank 1 CY8C29/27/24/22/21xxx and CY8CLED04/08/16

Block/Bit	7	6	5	4	3	2	1	0
MSB	Data Invert	0	1	Compare Type	Interrupt Type	0	0	1
LSB	0	BCEN	1	Compare Type	0	0	0	1

BCEN gates the compare output onto the row broadcast bus line. This bitfield is set in the Device Editor by directly configuring the broadcast line. The Data Invert flag, set through a user module parameter displayed in the Device Editor, controls the sense of the enable input signal. The CompareType flag indicates whether the compare function is set to “Less Than or Equal” or “Less Than.” The InterruptType flag determines whether to trigger the interrupt on the compare event or on the terminal count. Both CompareType and InterruptType are set in the Device Editor directly through user module parameters described in the earlier section on the topic.

Table 5. Input Register, Bank 1

Block/Bit	7	6	5	4	3	2	1	0
MSB	0	0	1	1	Clock			
LSB	Enable				Clock			

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

Table 6. Output Register, Bank 1 CY8C29/27/24/22/21xxx and CY8CLED04/08/16

Block/Bit	7	6	5	4	3	2	1	0
MSB	AuxClk		AuxEnable	AuxSelect		OutEnable	OutputSelect	
LSB	AuxClk		0	0	0	0	0	0

The user module “ClockSync” parameter in the Device Editor determines the value of the AuxClk bits. Though similarly named, the AuxEnable and AuxSelect bits are related, instead, to the OutEnable and OutSelect bit fields. AuxEnable and AuxSelect permit driving the terminal count output signal onto one of the row output busses and are controlled by manipulating the row bus graphically in the Device Editor placement view. OutEnable is set when the compare output is driven onto one of the row or global output busses. OutputSelect controls which of the busses are driven from the compare output.

Table 7. Count Register (DR0), Bank 0

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

Count is the PWM16 MSB and LSB down PWM. Both can be read using the PWM16 API.

Table 8. Period Register (DR1), Bank 0

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

Period holds the MSB and LSB of the period value that is loaded into the Counter register upon enable or terminal count condition. Both can be set in the Device Editor and the PWM16 API.

Table 9. Pulse Width Register (DR2), Bank 0

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

PulseWidth holds the MSB and LSB of the pulse width value used to generate the compare event. Both are set in the Device Editor and the PWM16 API.

Table 10. Control Register (CR0), Bank 0

Block/Bit	7	6	5	4	3	2	1	0
MSB	0	0	0	0	0	0	0	0[1]
LSB	0	0	0	0	0	0	0	Start/Stop

Start/Stop indicates that the PWM16 is enabled when set. It is modified by using the PWM16 API.

Start/Stop is controlled by the LSB Control register in chained PSoC blocks and is set to zero.

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 © 2000-2014 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.