8-Bit Counter Datasheet Counter8 V 2.60

Copyright © 2002-2012 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8CTMA140, CY8C21x45, CY8C22x45, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C21x12						
8-bit	1	0	0	71	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

- The 8-bit general purpose counter uses one PSoC block
- Source clock rates up to 48 MHz
- Automatic reload of period on terminal count
- Programmable pulse width
- Input enables/disables continuous counter operation
- Interrupt option on compare output *or* terminal count

The 8-bit Counter User Module provides a down counter with a programmable period and pulse width. The clock and enable signals can be selected from any system time base or external source. Once it starts, the counter operates continuously and reloads its internal value from the period register upon reaching terminal count. During each clock cycle, the counter compares the current count to the value stored in the compare register. During each clock cycle, the counter also tests the count against the value of the compare register for either a “less than" or “less than or equal to" condition. The comparator output provides a logic level that may be routed to pins and to other user modules. Most PSoC device families also permit the terminal count output to be routed in the same manner. If your device has this ability, it is shown in the device editor. An interrupt can be programmed to trigger when the counter reaches the terminal count or when the comparator (primary) output is asserted.

Figure 1. Counter Block Diagram (Most PSoC Devices), Data Path Width n = 8 Bits

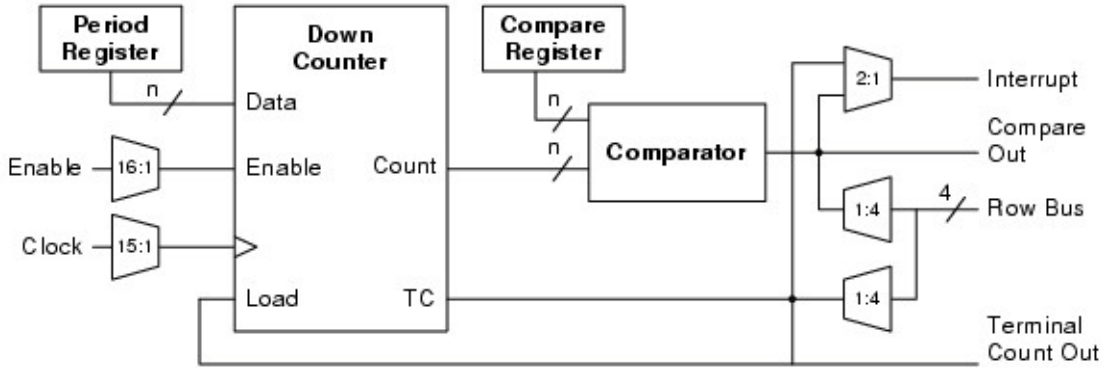
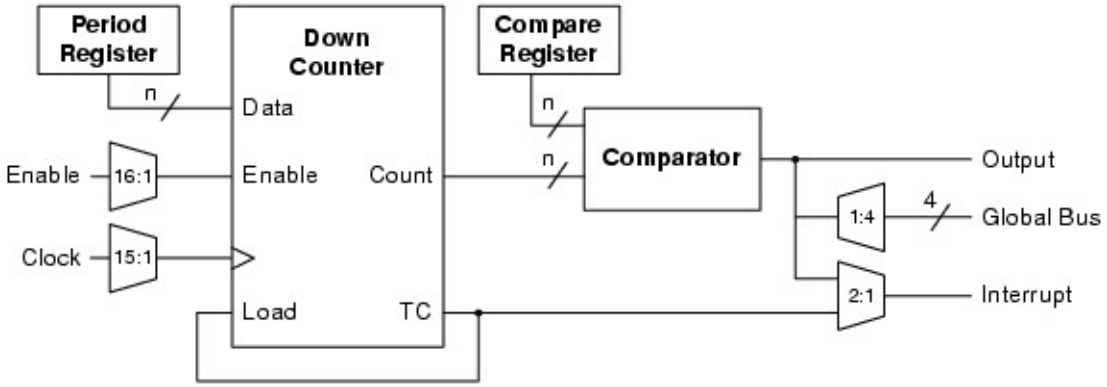


Figure 2. Counter Block Diagram (Devices Without Terminal Count Output), Data Path Width n = 8 Bits



Functional Description

The Counter User Module employs from one to four digital PSoC blocks, each contributing 8 bits to the total resolution. To form counters that exceed 8 bits, consecutive blocks are linked so that their internal carry, terminal count, and compare signals are synchronously chained. This concatenates the 8-bit Count, Period, and Compare registers (data registers DR0, DR1 and DR2, respectively) from block to block to provide the required resolution. In this way, Counters wider than 8 bits operate as a single monolithic synchronous counter.

The Counter 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 it starts, the Count register is decremented on the rising edge of each clock cycle at which the active-high enable input signal is asserted. On the rising clock edge following the terminal count when the Count register reaches zero, it is reloaded from the Period register.

The Period register can be modified with a new value at anytime. When the Counter is stopped, writing a value to the Period register also changes the value in the Count register. While the Counter 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 1 count greater than the value stored in the Period register. The duration in terms of the period of the input clock is given by the following equation.

Equation 1

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

The Counter 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 Compare register. The comparator performs a "less than" or "less than or equal" test depending on an option selected using the Device Editor. The Counter asserts 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 compare value and the period sets the duty cycle of the output waveform. The duty cycle ratio can be computed using the following equation.

Equation 2

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

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

Table 1. Counter Special Output Signal Conditions

Period Register Value	Compare Type	Compare Register Value	Ratio of Pulse-Width High Timer 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 = Compare	≤	Period = Compare	1.0
Period = Compare	<	Period = Compare	Period/(Period+1)
Compare Value > Period	Don’t Care	Compare Value > Period	1.0

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

In the CY8C29/27/24/22/21xxx device families, the Counter 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 should be taken when modifying the Compare register since its value, in conjunction with the current count value, determines the Counter’s output state. To prevent a possible premature low assertion of the

output signal and potential glitches, the Compare register should 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 Counter can be routed to a pin where its state is polled. Upon the detection of the output transition from high to low, the Compare can then be updated. Note that if the Compare causes the compare true condition, then the output is asserted high on the next clock.

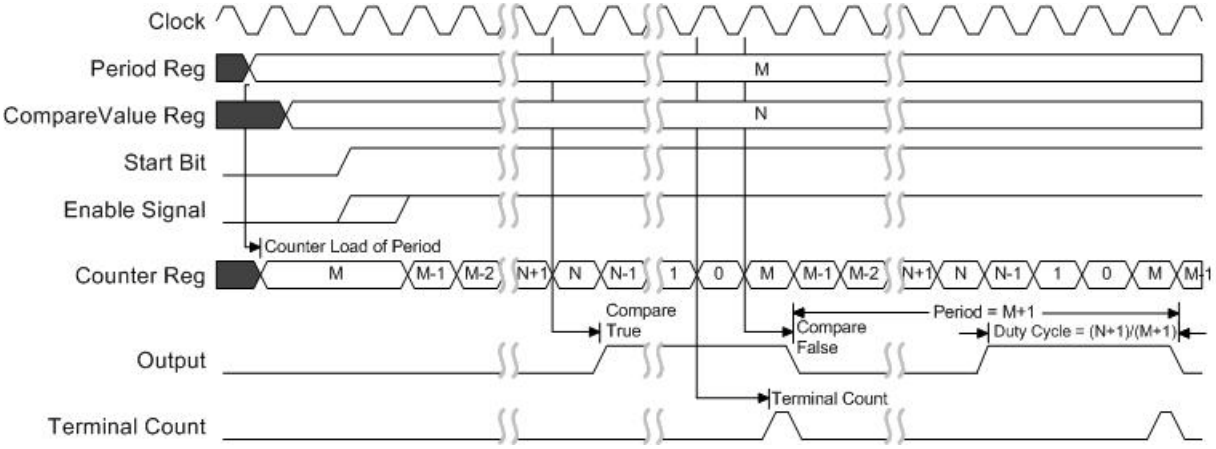
Acquiring the Count register value should be done very carefully. Reading the Count register causes its contents to latch into the Compare register. This causes the output duty cycle to change.

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 Compare register contents, reads the Count register, reads the Compare register, restores the Compare register, and then restores the clock. See the description for the ReadCounter() function in the Application Programming Interface section for possible side effects.

Timing

The Counter User Module’s operation may be gated on and off or clocked by external pins routed to the Counter by the global bus feature of the PSoC device. The following figure illustrates the timing for the Counter User Module.

Figure 3. Counter Timing Diagram



DC and AC Electrical Characteristics

Table 2. Counter AC Electrical Characteristics for the CY8C29/27/24/22/21xxx Device Family

Parameter	Typical	Limit	Units	Conditions and Notes
Maximum input frequency	--	48[1]	MHz	Vdd = 5.0 V[2]
Maximum output frequency	--	24[1]	MHz	Vdd = 5.0 V and 48 MHz input clock
	--	12[3]	MHz	Vdd = 3.3 V and 24 MHz input clock

Electrical Characteristics Notes

1. If the input or output is routed through the global buses, then the frequency is limited to a maximum of 12 MHz.
2. The provided enable signal is always high; otherwise, the limit is 24 MHz.
3. Fastest clock available to PSoC blocks is 24 MHz at 3.3 V operation.

Placement

The Counter consumes one digital PSoC block for every 8 bits of resolution. When more than one block is allocated, all the blocks are placed consecutively by the Device Editor in order of increasing block number from least significant byte (LSB) to most significant byte (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 names and block names to provide direct access to the Counter registers through the API include files. The block names assigned by the various Counter User Modules are given in the following table.

Table 3. Symbolic Names of the Mapped PSoC Blocks

PSoC Block Number	8-Bit Counter
1	CNTR8

Parameters and Resources

After a Counter User Module has been selected and placed using the Device Editor, values may be selected and altered for the following parameters.

Clock

The Clock parameter is selected from one of 16 sources. These sources include the 48 MHz oscillator (5.0 V 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 the available 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 family 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 busses. This parameter appears only for members of the CY8C29/27/24/22/21xxx family of PSoC devices.

Period

This parameter sets the period of the counter. Allowed values are between 0 and 2^n-1 where n is the width of the counter in bits. The period is loaded into the Period register. The effective output waveform period of Counter is the period count + 1. The value may be modified using the API.

CompareValue

This parameter sets the Compare register with the compare value. Allowed values are between zero and the period value. The value may be modified using the API.

CompareType

This parameter sets the compare function type “less than" or “less than or equal" as described in the functional description, above.

InterruptType

The Counter generates an interrupt on either the comparator true or on terminal count. A separate register independently enables the interrupt.

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 PSoC device families because of various data-path optimizations, particularly those applied to the system busses. This parameter may be used to control clock skew and ensure proper operation when reading and writing PSoC block register values. Appropriate values for this parameter should be determined from the following table:

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

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 family 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:

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 prior to the call if those values are required after the call. This “registers are volatile" policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must ensure their code observes the policy, too. Though some user module API function 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.

The following are the variables are provided for Counter8:

Counter8_PERIOD

Description:

Represents the value chosen for the Period field of the Counter8 in the Device Editor. The value can have a range between 0 and 255.

Counter8_COMPARE_VALUE

Description:

Represents the value chose for the PulseWidth field of the Counter8 in the Device Editor. The value can have a range between 0 and 255.

The following are the API programming routines provided for Counter8:

Counter8_Start

Description:

Starts the Counter8 User Module. If the enable input is high, the counter begins to down count.

C Prototype:

```
void Counter8_Start(void);
```

Assembly:

```
lcall  Counter8_Start
```

Parameters:

None

Return Value:

None

Side Effects:

You can modify the A and X registers by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

Counter8_Stop

Description:

Stops counter operation.

C Prototype:

```
void Counter8_Stop(void);
```

Assembly:

```
lcall  Counter8_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. You can modify the A and X registers by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

Counter8_EnableInt

Description:

Enables interrupt mode operation.

C Prototype:

```
void Counter8_EnableInt(void);
```

Assembly:

```
lcall Counter8_EnableInt
```

Parameters:

None

Return Value:

None

Side Effects:

You can modify the A and X registers by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

Counter8_DisableInt

Description:

Disables interrupt mode operation.

C Prototype:

```
void Counter8_DisableInt(void);
```

Assembly:

```
lcall Counter8_DisableInt
```

Parameters:

None

Return Value:

None

Side Effects:

You can modify the A and X registers by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

Counter8_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 Counter8 is stopped or when the counter reaches the zero count.

C Prototype:

```
void Counter8_WritePeriod(BYTE bPeriod);
```

Assembly:

```
mov   A, [bPeriod]
lcall  Counter8_WritePeriod
```

Parameters:

Counter period value is a value from 0 to 255.

Return Value:

None

Side Effects:

You can modify the A and X registers by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

Counter8_WriteCompareValue

Description:

Writes the Compare register with the compare value.

C Prototype:

```
void Counter8_WriteCompareValue(BYTE bCompareValue);
```

Assembly:

```
mov   A, [bCompareValue]
lcall  Counter8_WriteCompareValue
```

Parameters:

Compare value is the value from 0 to the period value.

Return Value:

None

Side Effects:

Writing the CompareValue register, while the counter is active, changes the duty cycle of the output. This may cause the output to glitch or change inadvertently. You can modify the A and X registers by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

Counter8_bReadCompareValue

Description:

Reads the CompareValue register.

C Prototype:

```
BYTE  Counter8_bReadCompareValue(void);
```

Assembly:

```
lcall  Counter8_bReadCompareValue
mov   [bCompareValue], A            ; store the value in RAM (if desired)
```

Parameters:

None

Return Value:

Compare value is stored in the CompareValue register and returned in the Accumulator.

Side Effects:

You can modify the A and X registers by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

Deprecated Aliases:

bCounter8_ReadCompareValue–This name may be withdrawn in future versions of PSoC Designer.

Counter8_bReadCounter

Description:

Reads the Count register value (hardware DR0 registers) while preserving the Compare register. This function may be called to read the Count register while the counter is running or when it is stopped.

Inadvertent interrupts are prevented even though the Compare and Count registers briefly become equal. However, there are important side effects (see the following Side Effects section).

C Prototype:

```
BYTE  Counter8_bReadCounter(void);
```

Assembly:

```
lcall  Counter8_bReadCounter
mov   [bCounter], A
```

Parameters:

None

Returns:

Counter register value that is returned in the Accumulator.

Side Effects:

If the Counter User Module is enabled and counting at the time this function is called, some clocks may be ignored (and corresponding decrements of the Count register missed) because the user module must be stopped momentarily. You can modify the A and X registers by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

Deprecated Aliases:

bCounter8_ReadCounter–This name may be withdrawn in future versions of PSoC Designer.

Sample Firmware Source Code

In the following examples, the correspondence between the C and assembly code is simple and direct. The values shown for the 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 *Counter8.h* file.

The following source illustrates the use of the APIs in assembly language:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;    This sample shows how to create a clock divider.  This specific
;    example divides the clock by 8.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

include "m8c.inc"                    ; include the device interface
include "PSoCAPI.inc"                ; include the API interface file

export _main

_main:

   mov   A, 07h                      ; set the period to 8
   call  Counter8_WritePeriod
```

```
   mov   A, 03h                       ; generate a 50% duty cycle
   call  Counter8_WriteCompareValue
   call  Counter8_EnableInt           ; enable the Counter Interrupt
   M8C_EnableGInt                     ; enable global interrupts
   call  Counter8_Start               ; start to count when the enable
                                      ;    input is asserted
.terminate:
        jmp .terminate
```

The same code in C is:

```
/*****************************************************************************
* Description:
*    This sample shows how to create a clock divider.  This specific
*    example divides the clock by 8.
*****************************************************************************/

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
void main(void)
{
    Counter8_WritePeriod(0x07);          /* set period to eight clocks   */
    Counter8_WriteCompareValue(0x03);    /* generate a 50% duty cycle    */
    Counter8_EnableInt();                /* ensure interrupt is enabled  */
    M8C_EnableGInt;                      /* enable global interrupts     */
    Counter8_Start();                    /* start the counter!           */
}
```

In these two cases, the only required include file is Counter8.h which provides the necessary declarations (pragmas and prototypes). Both assembly and C include files provide symbolic names for use when direct register access is required by an application. In CY8C29/27/24/22/21xxx user modules, the include files also provide macro definitions for in-line expansion of the source code of some of the shorter API functions. The files *PSoCAPI.inc* and *PSoCAPI.h* may be used instead of *Counter8.inc* and *Counter8.h*, respectively. These files contain include statements for each of the user modules instantiated (and placed) in a PSoC Designer project.

Configuration Registers

The 8-bit Counter uses a single digital PSoC block named CNTR8. Each block is personalized and parameterized through seven 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

Block/Bit	7	6	5	4	3	2	1	0
CNTR8	Data Invert	BCEN	1	Compare Type	Interrupt Type	0	0	1

BCEN gates the compare output onto the row broadcast bus line. This bit field 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
CNTR8	Enable				Clock			

Enable selects the data input from one of 16 sources. Clock selects the input clock from one of 15 sources. The values of both bit fields are determined by the settings of user module parameters of the same name in the Device Editor.

Table 6. Output Register, Bank 1, CY8C29/27/24/22/21xxx

Block/Bit	7	6	5	4	3	2	1	0
CNTR8	AuxClk		AuxEnable	AuxSelect		OutEnable	OutputSelect	

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. Enable is set when the compare output is driven onto one of the two or global output busses. OutputSelect controls which of the busses will be driven from the compare output.

Table 7. Count Register (DR0), Bank 0

Block/Bit	7	6	5	4	3	2	1	0
CNTR8	Count							

The Count register is the 8-bit down count value decremented by 1 in every clock cycle that the enable input is active. Its value is loaded from the contents of the Period register in the clock cycle following the terminal count (zero value). It can be read using the Counter8 API.

Table 8. Period Register (DR1), Bank 0

Bit	7	6	5	4	3	2	1	0
CNTR8	Period							

The Period register is a write-only register that can be set through the Device Editor and by the Counter8 API. When written, the value is transferred to the Count register if the user module is disabled through the API. Its value is automatically copied into the Count register in the clock cycle following terminal count.

Table 9. Compare Register (DR2), Bank 0

Block/Bit	7	6	5	4	3	2	1	0
CNTR8	Compare Value							

The Compare register holds the value against which the Count register is tested in order to generate the compare output. It can be set by the Device Editor and the Counter8 API.

Table 10. Control Register (CR0), Bank 0

Block/Bit	7	6	5	4	3	2	1	0
CNTR8	0	0	0	0	0	0	0	Start/Stop

Start/Stop indicates that the Counter8 is enabled when set and disabled when clear. It is modified by using the Counter8 API.

Version History

Version	Originator	Description
2.5	TDU	Updated Clock description to include: When using an external digital clock for the block, the row input synchronization should be turned off for best accuracy, and sleep operation.
2.60	DHA	Added support for CY8C21x12 devices.

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

Copyright © 2002-2012 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.