32-Bit Timer Datasheet Timer32 V 2.6

Copyright © 2000-2012 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, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx						
32-bit	4	0	0	154	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

- 32-bit general purpose timer uses four PSoC blocks
- Source clock rates up to 48 MHz
- Automatic reload of period on terminal count
- Capture for clocks up to 24 MHz.
- Terminal count output pulse may be used as input clock for other analog and digital functions
- Interrupt option on terminal count, capture (on some devices), or when counter reaches a preset value

The 32-bit Timer User Module provides a down counter with programmable period and capture ability. The clock and enable signals can be selected from any system time base or external source. Once started, the timer operates continuously and reloads its internal value from the period register upon reaching terminal count. The output pulses high in the clock cycle following terminal count. Events can capture the current Timer count value by asserting the edge-sensitive capture input signal. Each clock cycle, the Timer tests the count against the value of the compare register for either a “Less Than" or “Less Than or Equal To" condition. Interrupts may be generated based on terminal count and compare signals. Some device families offer two additional features. The interrupt options include “interrupt on capture" and, in addition, the compare signal may be routed onto the row buses. If these options are available on your chosen device they will be shown in the Device Editor.

Figure 1. Timer Block Diagram (Most PSoC Devices), Data Path width n = 32

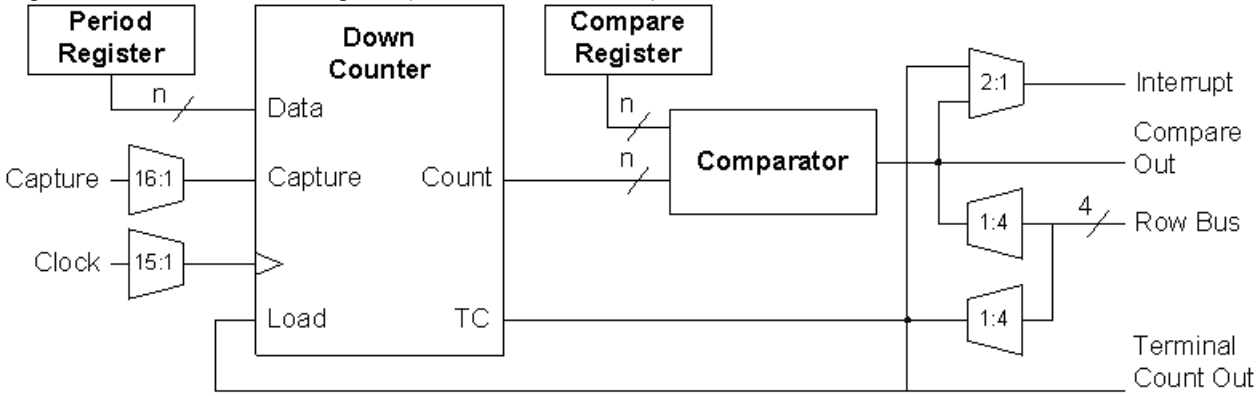
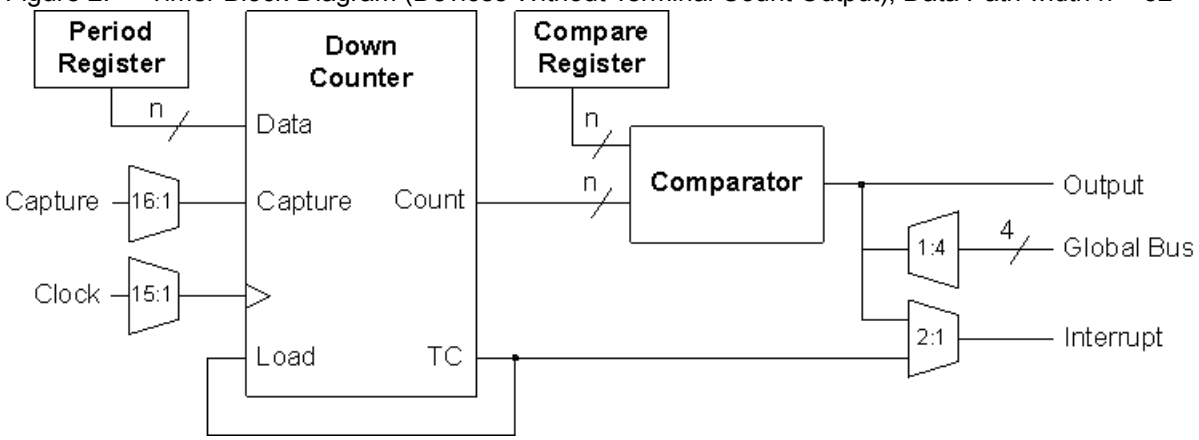


Figure 2. Timer Block Diagram (Devices Without Terminal Count Output), Data Path width n = 32



Functional Description

The Timer32 User Module employs four digital PSoC blocks, each contributing 8 bits to the total resolution. The consecutive blocks are linked so 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, Timer32 operates as a single monolithic synchronous timer.

The Timer API provides functions callable from C and assembly to stop and start operation of the Timer and to read and write the various data registers. A Control register starts and stops the Timer User Module. Writing the Period register, while the Timer is stopped, causes the Period register value to be copied into the Count register. While the Timer is stopped, the output is asserted low.

When a Timer is started, the Count register is decremented by 1 on each rising edge of the clock. On the rising clock edge following the zero count, the Count register is reloaded from the Period register. On the next falling edge, the terminal count event is triggered and the output is asserted high for one-half clock cycle or, optionally, in the CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxxdevice families, for one full clock cycle. In this way, the Timer acts as a clock divider. Its period and frequency are related to the period and frequency of the source clock by a factor equal to the value of the Timer's Period register plus 1.

Equation 1

$$OutputPeriod = SourceClockPeriod \times (PeriodRegisterValue + 1)$$

A period value of zero will output the input source clock shifted by one-half clock cycle, producing a divide-by-one clock. In the CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxxdevice families, the terminal count pulse width must be set to one-half cycle.

The duty cycle of the terminal count output is:

Equation 2

$$DutyCycle = \frac{0.5}{PeriodValue + 1}$$

Alternatively, when the terminal count pulse width is set to a full cycle, the duty cycle will be twice as long.

Equation 3

$$DutyCycle = \frac{1}{PeriodValue + 1}$$

The Period register value is a parameter that may be assigned using the Device Editor. In addition, it can be modified at run-time using the API. The Period register will be copied into the Count register automatically in the cycle after the value of the Count register reaches zero (terminal count). Thus, if the period is changed by means of the API, the new value does not take effect immediately. To make an immediate change at run time, the correct procedure is to stop the Timer, write a new period value and then restart the Timer.

On every input clock, the count in the Count register is compared to the value stored in the Compare register. The comparison performs a "Less Than" or "Less Than or Equal To" test, according to an option assigned to the CompareType parameter in the Device Editor. When the comparison condition is met, a compare event is triggered on the next clock.

In the CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxxdevice families, the Timer User Module provides the Compare output signal as an auxiliary output. This active-high signal is asserted on the rising edge of the clock cycle following the cycle in which compare condition is satisfied. This auxiliary output cannot be directly connected to the adjacent digital PSoC block; however, it can be connected to other digital PSoC blocks or to the GPIO pins through the local row output buses.

When the capture input is asserted high, the transition is synchronized to the system clock and the value in the Count register will be transferred to the Compare register. In the CY8C29/27/24/22/21xxx and CY8CLED04/08/16 families, if the interrupt type is set to “capture", an interrupt will occur following the capture event. The Count value can then be read using the ReadTimer API function. An interrupt will occur on a “compare true" event if the following conditions are met.

1. In the CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxxfamilies the interrupt type must be set to trigger on “capture".
2. The Timer interrupt must be enabled
3. Global interrupts must be enabled

The elapsed time is computed as follows.

Equation 4

$$ElapsedTime = ClockPeriod \times (PeriodValue - CounterValue)$$

An interrupt can be enabled to trigger on either the terminal count or on compare events and, in the CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxxfamilies of PSoC devices, on the capture signal itself. In the CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx, set the interrupt to trigger on the capture event when using the external capture signal to perform event timing. Set the interrupt to trigger on the terminal count, when performing elapsed timing measurements.

For algorithms that require reading the Timer countdown value on-the-fly without affecting the Count or Compare registers, the ReadTimerSaveCV() API can be called. This function will read the Count register value while preserving the Compare register contents. This function has some potential latency side effects as noted in the API section of this user module.

The capture mechanism allows an external event to be timed with a limiting bound on the maximum time, before an action should occur. This is performed as follows.

1. Set the Period register with a period value equal to the maximum value.
2. Set the Compare register with the maximum time limit count computed as follows.

Equation 5

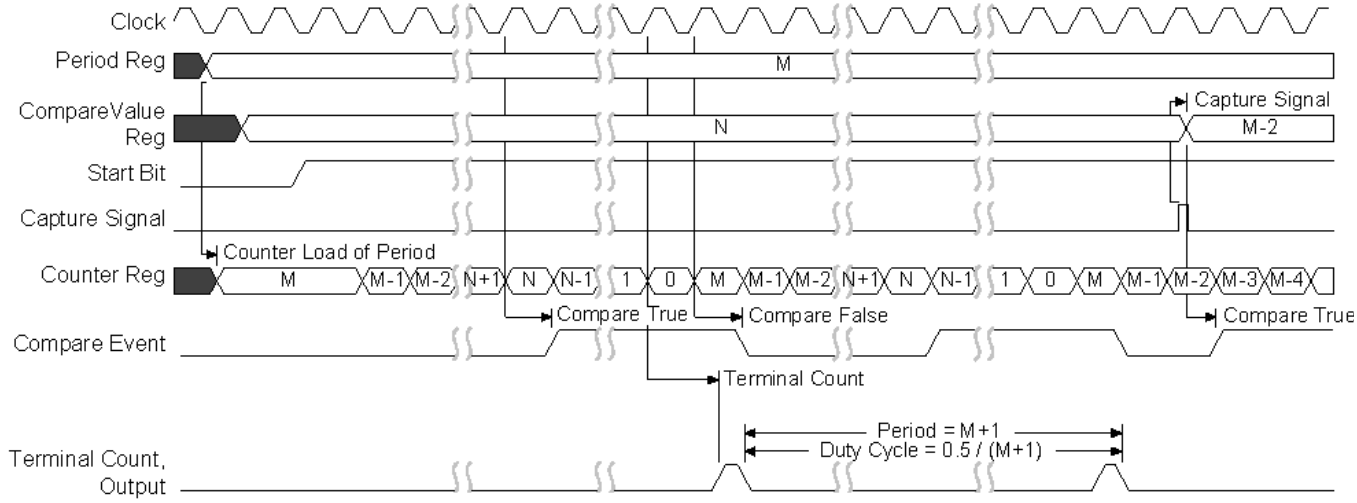
$$MaxTimeLimitCount = PeriodValue - \frac{MaxTimeLimit}{ClockPeriod}$$

3. In the CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxxfamilies set the interrupt to trigger on “capture".
4. Start the timer when appropriate.
5. Read the Compare register when the interrupt is triggered.

Timing

External pins, routed to the counter by the global bus feature of the PSoC device, can clock the Timer. The following figure illustrates the timing for the Timer User Modules.

Figure 3. Timing Diagram



DC and AC Electrical Characteristics

Table 1. Timer AC Electrical Characteristics

Parameter	Typical	Limit	Units	Conditions and Notes
Maximum input frequency	--	24[a]	MHz	24 or 32-bit width
Maximum output frequency	--	24[a]	MHz	Vdd=5.0V and 48 MHz input clock
	--	12[b]	MHz	Vdd=3.3V and 24 MHz input clock

a. If the input or output is routed through the global buses, then the frequency is limited to a maximum of 12 MHz.

b. Fastest clock available to PSoC blocks is 24 MHz at 3.3V operation.

Placement

The Timer32 consumes four digital PSoC blocks. All are 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 Timer registers through the API include files. The block names used by the various widths are given in the following table.

Table 2. Symbolic PSoC Block Names

PSoC Blocks	32-Bit Timer
1	TIMER32_LSB
2	TIMER32_ISB1
3	TIMER32_ISB2
4	TIMER32_MSB

Parameters and Resources

Once a Timer 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 the available sources. These sources include the 48 MHz oscillator (5.0V operation only), 24V1, 24V2, 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.

Capture

This parameter is selected from one of the available sources. A rising edge on this input causes the Count register to be transferred to the Compare register. The software capture mechanism will not operate correctly if this parameter is set to a value of one or is held high externally.

Output

The Output parameter may be disabled or routed to one of four global output signals. This parameter applies only to the CY8C26/25xxx 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 buses. This parameter appears only for members of the CY8C29/27/24/22/21xxx and CY8CLED04/08/16 families of PSoC devices.

CompareOut

The compare output may be disabled (without interfering with interrupt operations) or connected to any of the row output buses. It is always available as an input to the next higher digital PSoC block and to the analog column clock selection multiplexers, regardless of the setting of this parameter. This parameter appears only for members of the CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxxfamilies of PSoC devices.

Period

This parameter sets the period of the timer. Allowed values are between 0 and 2^{32}-1. This value is loaded into the Period register. The period is automatically reloaded when the counter reaches zero or the timer is enabled from the disabled state. This value may be modified using the API.

CompareValue

This parameter sets the count point in the timer period when a compare event is triggered. This value is loaded into the Compare register. Allowed values are between zero and the period value. This 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

This parameter specifies whether the terminal count event or the compare event triggers the interrupt. The interrupt is enabled using the API.

ClockSync

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

ClockSync Value	Use
Sync to SysClk	Use this setting for any 24 MHz (SysClk) derived clock source that is divided by two or more. Examples include VC1, VC2, VC3 (when VC3 is driven by SysClk), 32KHz, and digital PSoC blocks with SysClk-based sources. Externally generated clock sources should also use this value to ensure that proper synchronization occurs.
Sync to SysClk*2	Use this setting for any 48 MHz (SysClk*2) based clock unless the resulting frequency is 48 MHz (in other words, when the product of all divisors is 1).
Use SysClk Direct	Use when a 24 MHz (SysClk/1) clock is desired. This does not actually perform synchronization but provides low-skew access to the system clock itself. If selected, this option overrides the setting of the Clock parameter, 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 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.

TC_PulseWidth

This parameter provides the means of specifying whether the terminal count output pulse is one clock cycle wide or one half clock cycle wide.

InvertCapture

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. Invert-Capture applies only to the CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY7C64215, CY8CLED02/04/08/16, CY8CLED03D/04D, CY8CTST110, CY8CTMG110, CY8CTST120,

CY8CTMG120, CY8CTMA120, CY8C21x45, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxxfamilies 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 section specifies the interface to each function together with related constants provided by the "include" files.

Note

In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X 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.

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 Timer32.

Timer32_PERIOD

Description:

Represents the value chosen for the Period field of the Timer32 in the Device Editor. The value can have a range between 0 and 4294967295.

Timer24_COMPARE_VALUE

Description:

Represents the value chose for the PulseWidth field of the Timer32 in the Device Editor. The value can have a range between 0 and 4294967295.

Timer32_EnableInt

Description:

Enables the interrupt mode operation. Note, however, that global interrupts must also be enabled before interrupts will actually be serviced.

C Prototype:

```
void  Timer32_EnableInt(void);
```

Assembly:

```
lcall  Timer32_EnableInt
```

Parameters:

None

Return Value:

None

Side Effects:

This routine modifies the appropriate interrupt enable register in IO space. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

Timer32_DisableInt

Description:

Disables the interrupt mode operation.

C Prototype:

```
void  Timer32_DisableInt(void);
```

Assembly:

```
lcall  Timer32_DisableInt
```

Parameters:

None

Return Value:

None

Side Effects:

This routine modifies the appropriate interrupt enable register in IO space. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

Timer32_Start

Description:

Starts the Timer32 operation. The Count register will be decremented on the next clock cycle.

C Prototype:

```
void  Timer32_Start(void);
```

Assembly:

```
lcall  Timer32_Start
```

Parameters:

None

Return Value:

None

Side Effects:

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

Timer32_Stop

Description:

Stops the Timer32 operation.

C Prototype:

```
void  Timer32_Stop(void);
```

Assembly:

```
lcall  Timer32_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

The output will be set low and subsequent writes to the Period register will cause the Count register to update with the new period value. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

Timer32_WritePeriod

Description:

Writes the Period register with the period value. The period will be loaded into the Count register, when the zero-count condition is reached or immediately if the Timer32 is currently stopped.

C Prototype:

```
void  Timer32_WritePeriod(DWORD dwPeriod);
```

Assembly:

```
mov A,[dwPeriod]
push A
mov A,[dwPeriod+1]
push A
mov A,[dwPeriod+2]
push A
mov A,[dwPeriod+3]
push A
lcall _Timer32_WritePeriod
```

Parameters:

dwPeriod: The value is from 0 to 2^{32}-1.

Return Value:

None

Side Effects:

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

Timer32_WriteCompareValue

Description:

Modifies the value of the Timer’s Compare register. In order to avoid unexpected side effects, the Timer should be disabled (not yet enabled via the Start API function or by first calling the Stop API function).

C Prototype:

```
void  Timer32_WriteCompareValue(DWORD dwCompareValue);
```

Assembly:

```
mov A,[dwCompareValue]
push A
mov A,[dwCompareValue+1]
push A
```

```
mov A,[dwCompareValue+2]
push A
mov A,[dwCompareValue+3]
push A
lcall _Timer32_WriteCompareValue
```

Parameters:

dwCompareValue: The value is from 0 to the period value.

Return Value:

None

Side Effects:

If this function is called while the Timer is running and the compare value is equal to or greater than the current value of the Count register, then a compare event can occur. The value of the compare register may vary somewhat unpredictably as the Compare register is distributed across multiple PSoC blocks and written one byte at a time. The order in which the bytes are written is not specified and subject to change. This could cause an interrupt, if both the interrupt type is set to trigger on the compare event and the Timer interrupt is enabled. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

Timer32_ReadCompareValue

Description:

Reads the Timer32 Compare registers using “pass-by-reference" parameter.

C Prototype:

```
void  Timer32_ReadCompareValue(DWORD * pdwCompareValue);
```

Assembly:

```
mov   A,[pdwCompareValue]
mov   X,[pdwCompareValue+1]
lcall _Timer32_ReadCompareValue
```

Parameters:

pdwCompareValue: Pointer to a buffer to hold the Compare register data. The X register is loaded with the ram address where the return value is to be stored.

Return Value:

The value of the Compare register is returned in specified buffer.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the IDX_PP page pointer register is modified.

Timer32_ReadTimerSaveCV

Description:

Reads the current Timer32 Count register value, while preserving the Compare registers. This performs a software-solicited, hardware-synchronous counter capture operation. This function should only be used if the contents of the Compare register must be preserved. If the Compare register contents do not need to be preserved, then using the ReadTimer() function is preferred. Note that this API routine used to be called ReadCounter.

C Prototype:

```
void  Timer32_ReadTimerSaveCV(DWORD * pdwCount);
```

Assembly:

```
mov   A,[pdwCount]
mov   X,[pdwCount+1]
lcall _Timer32_ReadTimerSaveCV
```

Parameters:

None

Return Value:

pdwCount: Count register contents. The X register is loaded with the address of the return buffer.

Side Effects:

In order to read the value of the Count register, its value must be momentarily transferred to the Compare register before it can be returned. This causes the compare condition to become true immediately or on the next Timer input clock cycle depending on whether the CompareType parameter is set to “Less than or Equal to," or “Less Than," respectively. If (or when) the user module and global interrupts are enabled, the interrupt will be serviced, quite possibly before this API function has returned to the caller and even before it has restored the Compare register to its previous state. Interrupts are momentarily disabled. Finally, in order to restore the Compare register, the user module itself is temporarily disabled . This may cause the Count register to miss one or more counts. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the IDX_PP page pointer register is modified.

Timer32_ReadTimer

Description:

Reads the current Timer32 Count register value. This performs a software-solicited, hardware-synchronous counter capture operation. This is the preferred method of reading the Count registers, providing that the Compare registers are not required to be preserved. Note that this API routine used to be called CaptureCounter.

C Prototype:

```
void  Timer32_ReadTimer(DWORD * pdwCount);
```

Assembly:

```
mov   A,[pdwCount]
mov   X,[pdwCount+1]
```

```
lcall _Timer32_ReadTimer
```

Parameters:

None

Returns:

pdwCount: Pointer to a buffer to hold the Count register data. The X register is loaded with the address of the return buffer.

Side Effects:

Compare register contents are lost. The compare condition becomes true immediately or on the next Timer input clock cycle depending on whether the CompareType parameter is set to “Less than or Equal to," or “Less Than," respectively. If (or when) the user module and global interrupts are enabled, the interrupt will be serviced, quite possibly before this API function has returned control to its caller. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the IDX_PP page pointer register is modified.

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; i.e., 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 Timer32.h file.

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

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  Description:
;    This sample shows how to capture an event with a bounded time limit.
;    The count resolution is 1 us, with a bounded time limit of 16 seconds.
;
;    The interrupt should be set to interrupt on the Compare - Less than
;    equal. The capture input should be connected to the event that is
;    being measured. The clock should be connected to 24V2(VC2). The 24V1
;    (VC1) divider should be set to 8 and the 24V2(VC2) divider set to 3.
;
;    Computed time lapse is:  (0xFFFFFFFF - dwElapsedTime ) / 1 MHz
;
;    The foreground routine sets and starts the timer. The interrupt
;    level routine captures the value.
;
;  Parameters: none
;  Returns:    none
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
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
```

```
area bss (RAM,REL)
_dwElapsedTime::
dwElapsedTime::    BLK    4

area text(ROM, REL, CON)
_main:

CapturePulse::
RAM_X_POINTS_TO_STACKPAGE
RAM_SETPAGE_CUR >dwElapsedTime
mov   X, SP               ; create a stack frame for arguments
add   SP, 4

mov   [dwElapsedTime],   0
mov   [dwElapsedTime+1], 0
mov   [dwElapsedTime+2], 0
mov   [dwElapsedTime+3], 0
mov   [X],   FFh          ; set the period to a Max count
mov   [X+1], FFh
mov   [X+2], FFh
mov   [X+3], FFh
lcall  Timer32_WritePeriod
mov   [X],   FFh          ; set the compare value to trigger at 16 secs
mov   [X+1], 0Bh          ; 4,294,967,295 - 16,000,000 = 4,278,967,295
mov   [X+2], DCh          ;  -> 0xFF0BDC00
mov   [X+3], 00h
lcall  Timer32_WriteCompareValue
lcall  Timer32_EnableInt    ; enable the timer interrupt mask
M8C_EnableGInt             ; enable global interrupts
RAM_X_POINTS_TO_INDEXPAGE
RAM_SETPAGE_IDX >dwElapsedTime
    mov   X, dwElapsedTime     ; point X to dwElapsedTime
lcall  Timer32_Start        ; start the timer - timer will start to
.WaitForCapture:
mov   A, [X+0]
or    A, [X+1]
or    A, [X+2]
or    A, [X+3]
jz    .WaitForCapture
add   SP, -4

.TimerDone:
;Evaluate captured value here!
;If dwElapsedTime is not > 1 then compute elapsed time.
;else if wElapsedTime is 1 or 0 then event did not occur within
;time limit.
; return to caller when complete

.terminate:
    jmp .terminate
```

The interrupt level routine, located in the file *Timer32int.asm*, is:

```
_Timer32_ISR:
```

```
   ;@PSoC_UserCode_BODY@ (Do not change this line.)
   ;---------------------------------------------------
   ; Insert your custom code below this banner
   ;---------------------------------------------------
   ;   NOTE: interrupt service routines must preserve
   ;   the values of the A and X CPU registers.
    push  X
push  A
mov   X, _dwElapsedTime
call  Timer32_ReadTimer
call  Timer32_Stop
pop   A
pop   X
;---------------------------------------------------
   ; Insert your custom code above this banner
   ;---------------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)

   reti
```

The same code in C is as follows. Note that the interrupt routine must be written in assembly.

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

DWORD    dwElapsedTime;

void main(void)
{
    Timer32_WritePeriod(0xffffffff);
Timer32_WriteCompareValue(0xff0bdc00);
Timer32_EnableInt();
M8C_EnableGInt;
Timer32_Start();
while( dwElapsedTime == 0 );
}
```

Configuration Registers

The 32-bit Timer uses four digital PSoC blocks. In placement order from left to right they are named TIMER32_LSB, TIMER32_ISB1, TIMER32_ISB2 and TIMER32_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 3. Function Register, Bank 1, CY8C29/27/24/22/21xxx and CY8CLED04/08/16

Block/Bit	7	6	5	4	3	2	1	0
MSB	0	0	1	Compare Type	Interrupt Type	0	0	0
ISB2	0	0	0	Compare Type	0	0	0	0
ISB1	0	0	0	Compare Type	0	0	0	0
LSB	Data Invert	BCEN	0	Compare Type	0	0	0	0

BCEN gates the terminal count 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 capture input signal. The CompareType flag indicates whether the compare function is set to “Less Than or Equal To" or “Less Than." The InterruptType flag determines whether to trigger the interrupt on the compare event or on the terminal count (also see CaptureInt in the Control register). Both CompareType and InterruptType are set in the Device Editor directly through user module parameters described in the earlier section on the topic.

Table 4. Input Register, Bank 1

Block/Bit	7	6	5	4	3	2	1	0
MSB	0	0	1	1	Clock			
ISB2	0	0	1	1	Clock			
ISB1	0	0	1	1	Clock			
LSB	Capture				Clock			

Enable selects the data input from one of 16 sources. Clock selects the input clock from one of 16 sources. Both parameters are set in the Device Editor.

Table 5. 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	
ISB2	AuxClk		0	0	0	0	0	0
ISB1	AuxClk		0	0	0	0	0	0
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 bitfields. AuxEnable and AuxSelect permit driving the compare output signal onto one of the row output buses and are controlled by manipulating the row bus graphically in the Device Editor Interconnect View. OutEnable is set when the terminal count output is driven onto one of the row or global output buses. OutputSelect controls which of the buses will be driven from the compare output.

Table 6. Count Register (DR0), Bank 0

Block/Bit	7	6	5	4	3	2	1	0
MSB	Count(MSB)							
ISB2	Count(ISB1)							
ISB1	Count(ISB2)							
LSB	Count(LSB)							

The Count register is the 32-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 Timer32 API.

Table 7. Period Register (DR1), Bank 0

Block/Bit	7	6	5	4	3	2	1	0
MSB	Period(MSB)							
ISB2	Period(ISB1)							
ISB1	Period(ISB2)							
LSB	Period(LSB)							

The Period register is a write-only register that can be set through the Device Editor and by the Timer32 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 8. Compare Register (DR2), Bank 0

Block/Bit	7	6	5	4	3	2	1	0
MSB	Compare Val(MSB)							
ISB2	Compare Val(ISB1)							
ISB1	Compare Val(ISB2)							
LSB	Compare Val(LSB)							

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 Timer32 API.

Table 9. Control Register (CR0), Bank 0

Block/Bit	7	6	5	4	3	2	1	0
MSB	0	0	0	0	0	0	0	0
ISB2	0	0	0	0	0	0	0	0
ISB1	0	0	0	0	0	0	0	0
LSB	0	0	0	0	0	0	0	Enable

Enable indicates that the Timer32 is enabled when set and disabled when clear. It is modified by using the Timer32 API.

Version History

Version	Originator	Description
2.6	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.
2.6.b	DHA	Updated assembly prototypes for API functions in the user module datasheet.

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 © 2000-2012 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.