SleepTimer Datasheet SleepTimer V 1.0

Copyright © 2001-2013 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C29/27/24/22/21/20xxx, CY8C23x33, CY8CLED02/04/08/16, CY7C64215, CY7C64343, CY7C60413, CY8CLED0xD, CY8CLED0xG, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8CTMA140, CY8CTST200, CY8CTMG2xx, CY8C21x45, CY8C22x45, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C20336AN, CY8C20436AN, CY8C20636AN, CY8C20xx6AS, CY8C20XX6L, CY8C21x12, CY8C20xx7/7S, CYRF89x35, CY8C20045, CY8C20055, CY8C24x93, CY8C20065, CY7C69xxx						
	0	0	0	150	7	0

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

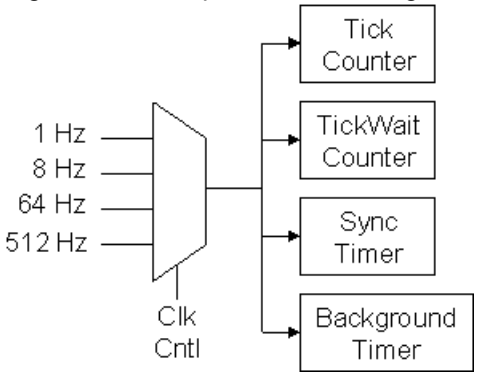
Features and Overview

- Does not require digital blocks
- Selectable 8, 16, or 32-bit tick counter
- Three types of timer functions.

The SleepTimer User Module provides basic timing functions without the use of valuable digital blocks. This user module makes use of the standard sleep timer to create a variety of timing functions that are often useful in a project. These functions include:

- Background tick counter may operate at 1, 8, 64, or 512 Hz.
- Simple wait function to delay program execution.
- Setable countdown timer.
- Loop governor, controls loop time to be consistent

Figure 1. SleepTimer Block Diagram



Functional Description

The SleepTimer User Module is based on the standard PSoC system sleep timer. The interrupt rate is user selectable to 1, 8, 64, or 512 Hz. Each time the sleep timer interrupt occurs, the tick counter increments by one count. The length of the tick counter is also user selectable (8, 16, or 32 bits) to optimize timing needs, interrupt efficiency, and RAM usage. The tick counter can be read at anytime with the supplied API.

All timer functions are based in increments of ticks. The period of the tick is inversely proportional to the interrupt rate. For interrupt rates of 1, 8, 64, and 512 Hz, the period is 1000, 128, 15.6, and 1.9 msec respectively.

For information about sleep timer accuracy, see the device datasheet.

Placement

Only one instance of the SleepTimer User Module may be placed in a given project. Although it does not require any PSoC analog or digital blocks, it uses the PSoC’s sleep timer hardware and interrupt vector.

Parameters and Resources

TickCounterSize

This parameter selects the size of the tick counter. Available options are 1, 2 and, 4 bytes. The user selects this parameter based on the interval rate, timing resolution, and maximum period that needs to be measured.

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.

SleepTimer_EnableInt

Description:

Enables the interrupt mode operation. Note, however, that global interrupts must also be enabled before interrupts will actually be serviced. This function must be called in order for the SleepTimer User Module to operate.

C Prototype:

```
void  SleepTimer_EnableInt(void);
```

Assembly:

```
lcall  SleepTimer_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 (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

SleepTimer_DisableInt

Description:

Disables the interrupt mode operation.

C Prototype:

```
void  SleepTimer_DisableInt(void);
```

Assembly:

```
lcall  SleepTimer_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 (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

SleepTimer_Start

Description:

Starts the SleepTimer operation. Global interrupts and SleepTimer interrupts must be enabled to allow the tick counter to advance.

C Prototype:

```
void  SleepTimer_Start(void);
```

Assembly:

```
lcall  SleepTimer_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 (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

SleepTimer_Stop

Description:

This function does not affect the operation of the sleep timer. Calling SleepTimer_DisableInt() will cease timer operation or by disabling global interrupts.

C Prototype:

```
void  SleepTimer_Stop(void);
```

Assembly:

```
lcall  SleepTimer_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 (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

Sleeptimer_SetInterval

Description:

This function sets the interrupt rate and therefore the tick period for all the SleepTimer User Module timing functions.

C Prototype:

```
void  SleepTimer_SetInterval(BYTE bInterval);
```

Assembly:

```
mov   A, [bInterval]           ; Place interval in A
lcall  SleepTimer_SetInterval
```

Parameters:

bInterval: This parameter allows the user to set one of four interrupt rates, 1, 8, 64, or 512 Hz.

Constant	Value	Interrupt rate
SleepTimer_1_HZ	0x18	1 Hz
SleepTimer_8_HZ	0x10	8 Hz
SleepTimer_64_HZ	0x08	64 Hz
SleepTimer_512_HZ	0x00	512 Hz

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 (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

SleepTimer_SyncWait

Description:

This function is used to synchronize events. It can be used to force a loop with different paths to always operate at the same period. This function has two modes. In the first mode, the timer is forced to the new value and the function is immediately exited. In the second mode, the program flow waits for the previous delay to lapse then sets the new delay and exits.

C Prototype:

```
void  SleepTimer_SyncWait(BYTE bTicks, BYTE bMode);
```

Assembly:

```
mov   A, [bTicks]                ; Load tick wait period in A
mov   X, SleepTimer_WAIT_RELOAD  ; Load mode in X
lcall  SleepTimer_SyncWait
```

Parameters:

bTicks: The amount of time in ticks that the period is set. If interval is set to 64Hz by the SleepTimer_SetInterval function, the sync delay will be bTicks times 1/64 seconds. If bTicks is set to 8 the sync time will be 125 mSec. bMode: This parameter sets the mode for this function. If set to 1, the new time will be set and program flow will exit the function immediately. If bMode = 0, program execution will delay until the remainder of the previous delay time has expired. The new tick count will be loaded and the function will be exited.

bMode	Value	Comment
SleepTimer_FORCE_RELOAD	0x01	Reload Timer immediately.
SleepTimer_WAIT_RELOAD	0x00	Wait for timer to expire then reload timer.

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 (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

SleepTimer_TickWait

Description:

This function is used to delay program execution by n number of ticks.

C Prototype:

```
void  SleepTimer_TickWait(BYTE bTicks);
```

Assembly:

```
mov   A, [bTicks]               ; Load tick wait period in A
lcall  SleepTimer_TickWait
```

Parameters:

bTicks: The amount of time in ticks to delay program execution. This function will not complete until the tick counter counts down to zero.

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 (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

SleepTimer_SetTimer

Description:

This function sets the timer to a value that will decrement by one (tick) each time the SleepTimer interrupt is executed. This rate is set by the SleepTimer_SetInterval() function. This function will set the countdown timer and exit immediately.

C Prototype:

```
void  SleepTimer_SetTimer(BYTE bTicks);
```

Assembly:

```
mov   A, [bTicks]               ; Load timer period in A
lcall  SleepTimer_TickWait
```

Parameters:

bTicks: The amount of time in ticks until the timer counts down to zero.

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 (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

SleepTimer_bGetTimer

Description:

Reads remaining time, in ticks, left on the countdown timer.

C Prototype:

```
WORD  SleepTimer_bGetTimer(void);
```

Assembly:

```
lcall  SleepTimer_bGetTimer
mov   [bTickRemaining], A         ; Returns remain tim in A
```

Parameters:

None

Return Value:

bTimeLeft: The value in ticks remaining until the countdown timer expires. If the return value is zero, the timer has expired prior to this function call.

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 (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

SleepTimer_bGetTickCntr

Description:

Returns current least significant 8 bits of the background tick counter.

C Prototype:

```
BYTE  SleepTimer_bGetTickCntr(void);
```

Assembly:

```
lcall  SleepTimer_bGetTickCntr
mov   [bTickCount], A         ; Returns LSB of tick counter in A
```

Parameters:

None

SleepTimer_iGetTickCntr

Description:

Returns current least significant 16bits of the background tick counter.

C Prototype:

```
unsigned int SleepTimer_iGetTickCntr(void);
```

Assembly:

```
lcall  SleepTimer_iGetTickCntr
mov   [iTickCounter], X          ; MSB returned in X
mov   [iTickCounter+1], A        ; LSB returned in A
```

Parameters:

None

Return Value:

bTicks: Returns the current 16 LSBs of the tick counter.

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 (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

SleepTimer_lGetTickCntr

Description:

Returns current value of the 32-bit background tick counter.

C Prototype:

```
unsigned long * SleepTimer_lGetTickCntr(unsigned long * lPtr);
```

Assembly:

```
mov   A, >lTickCount
mov   X, <lTickCount             ; A:X points to the return buffer
lcall  SleepTimer_lGetTickCntr
```

Parameters:

lPtr: Pointer to 32-bit tick counter variable to be returned.

Return Value:

lTimeLeft: The value in ticks remaining until the countdown timer expires. If the return value is zero, the timer has expired prior to this function call.

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 (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

Sample Firmware Source Code

Here is a simple C example for controlling a loop speed:

```
//--------------------------------------------------------------------
// Sample Code for the SleepTimer User Module
// In this example, the sleep timer interrupt rate is set to 64Hz.
// The loop is controlled to loop no faster than 8Hz.
//--------------------------------------------------------------------


#include "m8c.h"
#include "PSoCAPI.h"

void main(void)
{
   M8C_EnableGInt ;                              // Turn on interrupts
   SleepTimer_Start();
   SleepTimer_SetInterval(SleepTimer_64_HZ);    // Set interrupt to a
   SleepTimer_EnableInt();                       // 64 Hz rate


   // The following loop should loop no faster than 8 Hz (64/8)
   while(1) {
      SleepTimer_SyncWait(8, SleepTimer_WAIT_RELOAD);


   // …Other code in loop
   }
}
```

Here is a simple ASM example for controlling a loop speed:

```
;-------------------------------------------------------------------------
; Sample Code for the SleepTimer User Module
; In this example, the sleep timer interrupt rate is set to 64Hz.
; The loop is controlled to loop no faster than 8Hz.
;-------------------------------------------------------------------------

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

_main:

   M8C_EnableGInt               ; Turn on interrupts
lcall SleepTimer_Start        ; Start SleepTimer
mov   A,SleepTimer_64_HZ      ; Use 64 Hz interrupt rate
lcall SleepTimer_SetInterval  ;
lcall SleepTimer_EnableInt    ; Enable SleepTimer Interrupt

.Loop:
   mov   A,8                     ; Set tick delay to 8 ticks
   mov   X,SleepTimer_WAIT_RELOAD ; Wait for delay then reload timer
```

```
   lcall SleepTimer_SyncWait      ;

;  Put main loop user code here

   jmp .Loop
```

Configuration Registers

This register is configured by the initialization and API library. You do not have to change or read this register directly. This section is supplied as a reference.

Table 1. Register OSC_CR

Bit	7	6	5	4	3	2	1	0
Value	32k Select	PLL mode	No Buzz	Sleep[1]	Sleep[0]	CPU Speed[2]	CPU Speed[1]	CPU Speed[0]

Only bits 3 and 4 (Sleep[1:0]) are altered in the SleepTimer User Module.

Version History

Version	Originator	Description
1.0	DHA	Initial version
1.0.b	DHA	Updated constants for SleepTimer_SetInterval API function in 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 © 2001-2013 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.