Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.



THIS SPEC IS OBSOLETE

Spec No: 001-25439

Spec Title: AN2354 - PRACTICAL APPLICATION OF THE PSOC(R) 1 SLEEP TIMER

Sunset Owner: Meenakshi Sundaram Ravindran (msur)

Replaced By: 001-13619

AN2354 - Practical Application of the PSoC® 1 Sleep Timer

AN2354

Author: Darrin Vallis
Associated Project: Yes
Associated Part Family: CY8C21xxx, 24xxxA, 24x94, 27x43, 29xxx
Software Version: PSoC® Designer™ 5.1 SP1
Associated Application Notes: None

OBSOLETE

Application Note Abstract

AN2354 describes document a simple application of the Sleep Timer with PSoC® 1. People new to PSoC can enjoy the introduction to one of the PSoC's simple yet very useful resources, the Sleep Timer. An example is provided, which carries the new user through the setup and implementation of a PSoC project using the Sleep Timer.

Introduction

The Cypress Programmable System-on-Chip™ (PSoC) is a sophisticated, configurable mixed-signal device. It allows engineers to solve design problems with innovative, cost-effective solutions.

In fact, PSoC is such a feature rich device that engineers often overlook some simple yet very useful resources. The hardware multiply and accumulate system resource, analog modulators or even built-in programmable logic gates are good examples. They allow PSoC to implement hardware solutions impossible with traditional microcontroller architectures.

Another often overlooked resource is the sleep timer. It can be used to create long duration timer intervals that would otherwise take large hardware counters or require the CPU to remain active as a software counter. In some cases this is perfectly acceptable. However, if you have a battery-powered application, or want a more efficient implementation, the sleep timer is extremely useful.

More Than a "Sleep Timer"

First, let's get a few details straight. Even though it is referred to as the "sleep timer" in the PSoC Designer™ and in the Technical Reference Manual (TRM), this hardware counter does **not** have to be used in conjunction with the PSoC sleep mechanism. When configured properly, the sleep timer generates a periodic interrupt and wake-up the CPU from sleep. However, any interrupt source can wake the CPU, even from analog or digital blocks. So, the sleep timer may be used with PSoC sleep, but is also very useful as a standalone long duration counter.

To avoid further confusion, the sleep timer is also closely tied to watchdog timer reset (WDT) functionality in PSoC. When enabled, the watchdog timer is designed to generate a hardware reset in PSoC, after three cycles of the sleep timer. This is not intuitively obvious, so if you are using the sleep timer, be aware of its implications on the watchdog reset.

It is strongly recommended to investigate this further in the TRM on http://www.cypress.com and fully understand the relationship between sleep timer, sleep mechanism, and watchdog timer reset.

Configuring the Device

With that out of the way, let's proceed with actually using the sleep timer. Start a new project in the PSoC Designer with the device CY8C27443-24SXI. Make sure 'C' is checked to generate your main. Navigate to the Device Editor Interconnect View and select a frequency for the sleep timer under Global Resources. In this case, I chose 1 Hz. Make sure you have the watchdog timer disabled then click the Generate Application icon.

Figure 1. Configuring Global Resources

Global Resources	Value
CPU_Clock	3_MHz (SysClk/8)
32K_Select	Internal
PLL_Mode	Disable
Sleep_Timer	1_Hz
VC1= SysClk/N	1
VC2= VC1/N	1
VC3 Source	SysClk/1
VC3 Divider	1
SysClk Source	Internal 24_MHz
SysClk*2 Disable	No
Analog Power	SC On/Ref Low
Ref Mux	(Vdd/2)+/-BandGap
AGndBypass	Disable
Op-Amp Bias	Low
A_Buff_Power	Low
SwitchModePump	OFF
Trip Voltage [LVD (SMP)]	4.81V (5.00V)
LVDThrottleBack	Disable
Supply Voltage	5.0V
Watchdog Enable	Disable

We need some way to see the sleep timer working. LEDs are always a good debug tool. The CY3210-Eval1 board comes with a CY3210-MiniProg1 and a development board, providing a few LEDs and a socket for the included DIP samples. Perfect for this example. We are going to use pins P2[0] and P2[1] to show what's happening while your code is running, without the full ICE debugger. Place jumper wires from P2[0] and P2[1] to the LEDs of your choosing on the Eval1 board.

To drive LEDs, the PSoC I/Os need to be set to "Strong" drive mode. Do this in the Device Editor port/pin configuration window as shown in Figure 2.

Figure 2. Configuring Pins for LED Drive

Name	Port	Select	Drive	Interrupt
Port_0_0	P0[0]	StdCPU	High Z Analog	DisableInt
Port_0_1	P0[1]	StdCPU	High Z Analog	DisableInt
Port_0_2	P0[2]	StdCPU	High Z Analog	DisableInt
Port_0_3	P0[3]	StdCPU	High Z Analog	DisableInt
Port_0_4	P0[4]	StdCPU	High Z Analog	DisableInt
Port_0_5	P0[5]	StdCPU	High Z Analog	DisableInt
Port_0_6	P0[6]	StdCPU	High Z Analog	DisableInt
Port_0_7	P0[7]	StdCPU	High Z Analog	DisableInt
Port_1_0	P1[0]	StdCPU	High Z Analog	DisableInt
Port_1_1	P1[1]	StdCPU	High Z Analog	DisableInt
Port_1_2	P1[2]	StdCPU	High Z Analog	DisableInt
Port_1_3	P1[3]	StdCPU	High Z Analog	DisableInt
Port_1_4	P1[4]	StdCPU	High Z Analog	DisableInt
Port_1_5	P1[5]	StdCPU	High Z Analog	DisableInt
Port_1_6	P1[6]	StdCPU	High Z Analog	DisableInt
Port_1_7	P1[7]	StdCPU	High Z Analog	DisableInt
LED1	P2[0]	StdCPU	Strong	DisableInt
LED2	P2[1]	StdCPU	Strong	DisableInt
Port_2_2	P2[2]	StdCPU	High Z Analog	DisableInt
Port_2_3	P2[3]	StdCPU	High Z Analog	DisableInt
Port_2_4	P2[4]	StdCPU	High Z Analog	DisableInt
Port_2_5	P2[5]	StdCPU	High Z Analog	DisableInt
Port_2_6	P2[6]	StdCPU	High Z Analog	DisableInt
Port_2_7	P2[7]	StdCPU	High Z Analog	DisableInt

That's it; you are done configuring the hardware. Move to the Application Editor subsystem and we continue with the design.

Modifying the Software/Application

Even though it might have sounded complicated, using the sleep timer is actually very easy. The basic steps are:

1. Clear the watchdog and sleep timer.
2. Enable the sleep timer interrupt.
3. Enable global interrupts.

Really, that's all there is to it. Here's the actual PSoC code used to do this:

Code 1. PSoC Code to Modify Software

```
M8C_ClearWDTAndSleep;


M8C_EnableIntMask(INT_MSK0,
INT_MSK0_SLEEP);


M8C_EnableGInt;
```

OBSOLETE

After this is set up, the sleep timer generates an interrupt at one-second intervals. Remember, this can have up to 20% deviation, because the sleep timer is derived from the PSoC's internal low speed oscillator. Accuracy can be increased by the addition of a cheap external watch crystal, which can be selected as an alternate clock for the sleep timer. It may also be used as a reference for the PSoC phase-locked loop (PLL).

Now that the sleep timer is generating interrupts, we must do something with them. This is where the interrupt service routine (ISR) comes into play. We want the PSoC to jump to a particular piece of code every time the interrupt fires. Do this by opening *boot.tpl*, and making the change shown below. Replace the text `'@INTERRUPT_25'` with `"ljmp _SleepISR"` (no quotes). This tells the CPU to jump to a label called "SleepISR" whenever the sleep interrupt is triggered. Save and close *boot.tpl*, re-generate the project and check your *boot.asm* for the changes.

Code 2. Code to Add SleepISR

```
Org 64h          ;Sleep    Timer    Interrupt
Vector

ljmp _SleepISR

reti
```

Now you need to do something in the ISR. The following example code shows how to toggle an LED, decrement a software counter in the ISR, check if the counter has expired, and return to main. Notice that after "`NotDone:`" we call `M8C_ClearWDTAndSleep`. This is critical, as it resets the sleep timer and allows a new interrupt to be generated.

Code 3. Code to Reset SleepISR

```
_SleepISR:

xor reg[PRT2DR],0x02 // Toggle LED2, P2_1

cec [_sleepTimer]    // Dec timer

jnz notDone          // Not done. Keep
                     counting down

M8C_DisableIntMask INT_MSK0,INT_MSK0_SLEEP

                     // Done, so disable
                     interrupt

and reg[PRT2DR],~0x01      // Turn off P2_0

reti                 // And return to main

notDone:

M8C_ClearWDTAndSleep // Clear WDT and sleep
                     timer

reti                 // And return to main
```

Putting it All Together

Now it's time to put everything together. The following main code defines a global variable for the countdown timer and initializes it to five sleep timer ticks. Both LEDs are turned on, the sleep timer interrupt is cleared then enabled, and finally, the global interrupts are enabled. The "while" loop calls the PSoC sleep function.

Code 4. Code to Define Global Variable for the Countdown Timer

```
BYTE sleepTimer=5;

void main()

{

    PRT2DR |= LED1 | LED2;

    M8C_ClearWDTAndSleep;

    M8C_EnableIntMask(INT_MSK0,
    INT_MSK0_SLEEP);

    M8C_EnableGInt;


    while(1)

    {
            M8C_Sleep;


    }

}
```

At this point, the CPU is actually asleep, doing nothing and saving a lot of power. PSoC can get down to single digit micro-amp sleep current in this mode, provided its global resources are configured correctly. Conversely, you can use the main loop for processing data or other tasks in your application.

After the sleep timers interrupt fires, it wakes up the CPU, which jumps to your ISR. LED2 is toggled, the counter is decremented, and end of count is checked. The sleep timer gets re-enabled and execution returns to main. The "while" loop once again calls the PSoC sleep function and the cycle repeats.

So, now you know how to use the PSoC sleep timer to generate periodic interrupts how to put the PSoC into sleep mode and how to wake it up. The attached PSoC project contains the complete source code and shows all the device configurations discussed earlier. Download the project HEX file to a CY3210-Eval1 board with a CY8C27443-24SXI in the socket. Have the proper jumper wires to the LEDs on the board, and you should see flashing LEDs.

Summary

PSoC is an extremely versatile engineering platform. I guarantee the further you investigate, the more things you discover it can do in your designs.

OBSOLETE

About the Author

Name: Darrin Vallis
Title: Principal Field Applications Engineer
Contact: Cypress Semiconductor
Austin, Texas

OBSOLETE

Document History Page

Document Title: AN2354 - Practical Application of the PSoC® 1 Sleep Timer
Document Number: 001-25439

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1444223	YJI	10/01/2007	Original Document
*A	3196047	YJI	03/15/2011	Document title updated. Document History Page added. The Abstract content has been updated.
*B	4311210	MSUR	03/17/2014	Obsolete document – SleepTimer is already a user module

OBSOLETE

In March of 2007, Cypress recataloged all of its Application Notes using a new documentation number and revision code. This new documentation number and revision code (001-xxxxx, beginning with rev. **), located in the footer of the document, will be used in all subsequent revisions.

PSoC is a registered trademark of Cypress Semiconductor Corp. "Programmable System-on-Chip," PSoC Designer, and PSoC Express are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.com/

© Cypress Semiconductor Corporation, 2007-2011. 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.

This 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.0