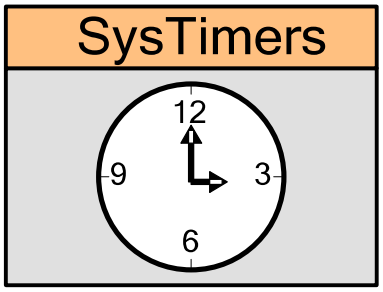
SysTimers
1.0

Features



- Uses PSoC 4/5LP SysTick Timer
- Requires no internal hardware or GPIO pins.
- Up to 16 parallel timers
- Timer updates in a single ISR
- Timer resolution of 100uS, 1mS, 10mS, and 100mS
- Two modes of operation

General Description

The SysTimers component makes use of the Cortex M0/M3 SysTick timer to create 2 to 16 non-blocking timers. The SysTick interrupt period is set by the component and increments the timer/s at each interrupt. These timers provide a way to time or schedule parallel periodic events without consuming valuable hardware or making use of blocking functions such as CyDelay().

When to Use a SysTimers Component

Use the SysTimers component when events or delays need to be timed, but without using blocking code such as with CyDelay(). Timer resolution is between 100u Sec and 100 mSec.

Quick Start Guide

Below is an example of how to setup a timer to cause a section of code to be executed every 50 mSec. Additional timers could be added as well that expire at different rates. As you can see, there is no need for blocking code such as CyDelay(), so your code does not waste time executing timing loops. The timer will automatically retrigger until the SysTimers component is stopped, or the timer is released.

```
uint32 update_timerID;

CyGlobalIntEnable;  // Make sure Global interrupt are enabled

SysTimers_Start();  // Enable the SysTick interrupt and initialize the timers.

// Start a timer to expire at a 20 Hz rate, every 50 mSec.
```

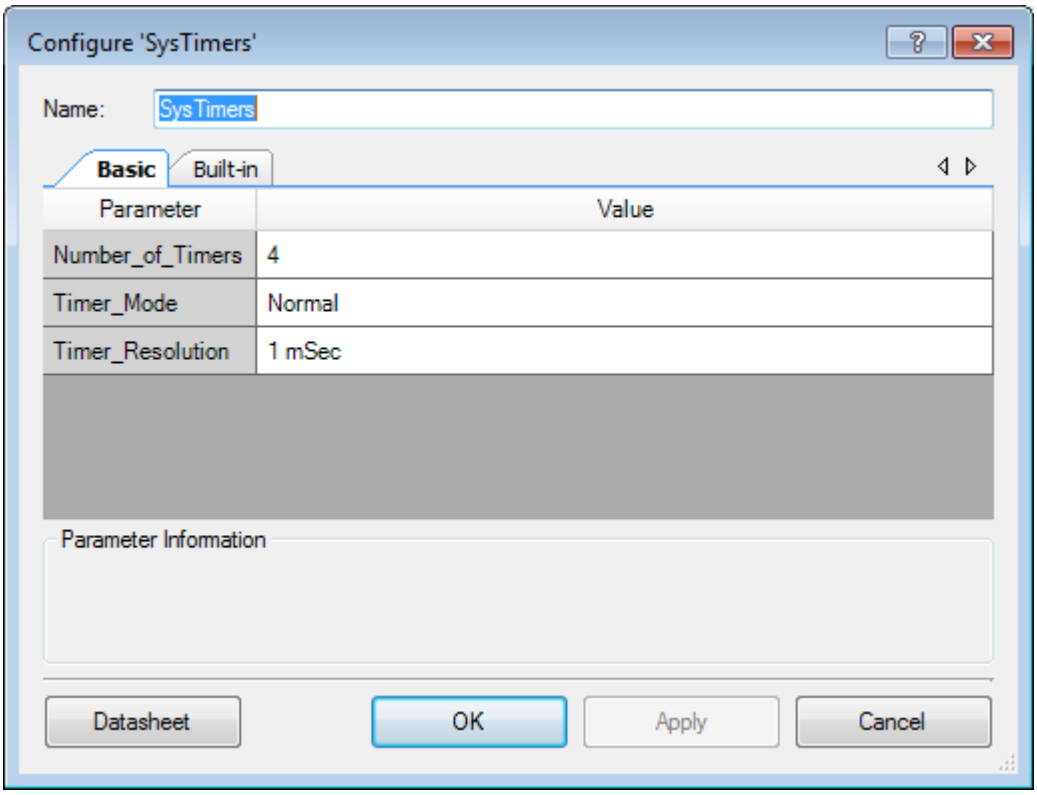
```
// Using the constant SysTimers_TICKS_PER_SECOND, makes your code independent
// of the timer resolution.  The update_timerID is now the ID of the timer.
update_timerID = SysTimers_GetTimer(SysTimers_TICKS_PER_SECOND/20);

for(;;)  // Main Loop
{
    if( SysTimers_GetTimerStatus(update_timerID))
    {
        // Put code here to execute every 50 mSec
    }
    // Other User code
}
```

Input/Output Connections

There are no Input or Output pins on this component.

Component Parameters

Drag a SysTimers component onto your design and double-click it to open the Configure dialog.



Parameters

Number of Timers

This parameter allows you to select the maximum timers that you require in your design. The options are 2, 4, 8, and 16, with 4 being the default.

Time Mode

There are two modes “Counting” and “FastIRQ”. For most applications, they are identical, except the FastIRQ has minimal code in the ISR. The Counting mode returns the number of periods that have expired since the timer was started or the last time the GetTimerStatus() function was called. The FastIRQ mode will just return a non-zero if the timer period has expired.

Timer Resolution

This parameter sets the timer resolution. It is basically how often the SysTick ISR is invoked to update the counter/s. The default period is 1 mSec.

- 100 mSec
- 10 mSec
- **1 mSec**
- 100 uSec

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function together with related constants provided by the "include" files. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name "SysTimers_1" to the first instance of a component in a given project. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "SysTimers."

Functions	Description
SysTimers_Start()	Initializes timers, sets the SysTick period and enables the interrupt.
SysTimers_Stop()	Disables timers by disabling the interrupt.
SysTimers_GetTimer()	Get a free timer and set the period
SysTimers_GetTimerStatus()	Check to see if timer has expired

Functions	Description
SysTimers_GetTimerValue()	Get remaining timer period
SysTimers_GetSysTickValue()	Return value of SysTick timer.
SysTimers_ResetTimer()	Reset the period of a specific timer.
SysTimers_ReleaseTimer()	Release a timer for use so that it can be used by another part of the firmware.
SysTimers_ReleaseAllTimers()	Release all timers and reset their period back to default.

void SysTimers_Start(void)

Description: This function sets the period of the SysTick interrupt to the selected value in the customizer, initializes the timers, and enables the SysTick interrupt.

Parameters: None

Return Value: None

Side Effects: None

void SysTimers_Stop(void)

Description: Disables the SysTick interrupt so that timers will not update.

Parameters: None

Return Value: None

Side Effects: None

uint32 SysTimers_GetTimer(uint32 timerPeriod)

Description: Returns a timer ID from the timer set to the requested period.

Parameters: uint32 timerPeriod: The period in SysTick interrupt periods For example, if the timer resolution is set to 1mSec and the timer Value is 100, the timer will expire in 100 mSec or at a 10 Hz rate.

Return Value: uint32: The timer ID of the requested timer. Use this ID when calling the functions GetTimerStatus(), GetTimerValue(), ResetTimer(), or ReleaseTimer();

Side Effects: None

uint32 SysTimers_GetTimerValue(uint32 timerID)

Description: Returns how many SysTicks until the counter expires in the Counting mode and the amount of SysTicks since the timer started in the FastIRQ mode .

Parameters: uint32 timerID: Timer ID for the specific timer, returned from GetTimer().

Return Value: None

Side Effects: None

uint32 SysTimers_GetSysTickValue(void)

Description: Returns the value of the SysTick counter or "SysTimers_SysTickCount". This 32-bit counter is set to zero when the Start() command is executed. If the timer resolution is set to 1mSec, the timer will roll over in about 49 days.

Parameters: None

Return Value: uint32: Value of the 32-bit SysTick counter, "SysTimers_SysTickCount".

Side Effects: None

void SysTimers_ResetTimer(uint32 timerID, uint32 timerPeriod)

Description: Resets the timer with a new or original period, starting at the call of this function.

Parameters: uint32 timerID: ID of the timer that will be reset.

uint32 timerPeriod: New timer period. If this value is zero, the old period will be reloaded.

Return Value: None

Side Effects: None

void SysTimers_ReleaseTimer(uint32 timerID)

Description: Releases the specified timer so it can be used elsewhere.

Parameters: uint32 timerID: ID of the specific timer to release.

Return Value: None

Side Effects: None

void SysTimers_ReleaseAllTimers(void)

Description: Releases all timers. User must use GetTimer() function to resume using any timers.

Parameters: None.

Return Value: None

Side Effects: All timer IDs will become invalid.

Functional Description

The SysTimers component provides up to 16 non-blocking timers without adding any hardware resources. It takes advantage of the Cortex M0 and M3 SysTick timer in the PSoC 4 and PSoC 5LP processors.

Resources

The SysTick interrupt is used by the SysTimers component.

API Memory Usage

SRAM usage is 12 bytes per timer plus 4 bytes overhead. For typical usage, Flash will be less than 500 bytes. Flash usage is not dependent on the amount of channels selected.

DC and AC Electrical Characteristics

N/A

Component Changes

This section lists the major changes in the component from the previous versions.

Version	Description of Changes	Reason for Changes / Impact
1.00	Initial Release	NA

© Cypress Semiconductor Corporation, 2010-2013. 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® is a registered trademark, and PSoC Creator™ and Programmable System-on-Chip™ are trademarks 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.