AN2046

PSoC® 1 Task Scheduler

Author: Edward Nova, Arvind Krishnan
Associated Project: Yes
Associated Part Family: All PSoC® 1 Families
Software Version: PSoC Designer™ 5.4 SP1
Related Application Notes: None

AN2046 explains the implementation of a simple task scheduler in the PSoC® 1 family of devices. A task scheduler can be used in applications where you need to execute multiple tasks within respective periods of time.

Contents

1 Introduction....1
2 PSoC Resources....2
2.1 PSoC Designer....2
2.2 Code Examples....3
2.3 Technical Support....4
3 Task Scheduler....5
3.1 What is a Task?....5
3.2 Time Slots....5
4 Implementation....6
5 Summary....8
Appendix A – Considerations when task blocking is used.9
Worldwide Sales and Design Support....12

1 Introduction

In this application note, we illustrate how a task scheduler can be used to schedule multiple tasks that execute periodically. An example project is provided in which each task executes during a specific time slot. Eight such tasks are associated with PSoC 1 GPIOs, which are toggled to indicate execution of the tasks. The example project can be used as template to organize your own functions into the structure of the task scheduler.

Full-featured commercial RTOS for other PSoC families are available; visit the third-party websites listed below for more details-

PSoC 4:

- Segger (https://www.segger.com/embos_cm3_cypress.html)
- FreeRTOS (http://interactive.freertos.org/entries/46569687-FreeRTOS-8-0-0-for-PSoC-4-Pioneer-kit-using-ARM-MDK-compiler)

PSoC 5:

- Segger (https://www.segger.com/embos_cm3_cypress.html)
- Micrium (http://micrium.com/page/downloads/ports)
- FreeRTOS (http://www.freertos.org/FreeRTOS-port-and-demo-for-Cypress-PSoC5-CY8C5588-Cortex-M3.html)

2 PSoC Resources

Cypress provides a wealth of data at www.cypress.com to help you to select the right PSoC device for your design, and quickly and effectively integrate the device into your design. In this document, PSoC refers to the PSoC 1 family of devices. To learn more about PSoC 1, refer to the application note AN75320 - *Getting Started with PSoC 1*.

The following is an abbreviated list for PSoC 1:

- **Overview:** PSoC Portfolio, PSoC Roadmap
- **Product Selectors:** PSoC 1, PSoC 3, PSoC 4, or PSoC 5LP. In addition, PSoC Designer includes a device selection tool.
- **Datasheets:** Describe and provide electrical specifications for the PSoC 1 device family.
- **Application Notes and Code Examples:** Cover a broad range of topics, from basic to advanced level. Many of the application notes include code examples.
- **Technical Reference Manuals (TRM):** Provide detailed descriptions of the internal architecture of the PSoC 1 devices.
- **Development Kits:**
  - CY3215A-DK In-Circuit Emulation Lite Development Kit includes an in-circuit emulator (ICE). While the ICE-Cube is primarily used to debug PSoC 1 devices, it can also program them using ISSP.
  - CY3210-PSOCEVAL1 Kit enables you to evaluate and experiment Cypress's PSoC 1 programmable system-on-chip design methodology and architecture.
  - CY8CKIT-001 is a common development platform for all PSoC family devices.
- The MiniProg1 and MiniProg3 devices provide an interface for flash programming.

2.1 PSoC Designer

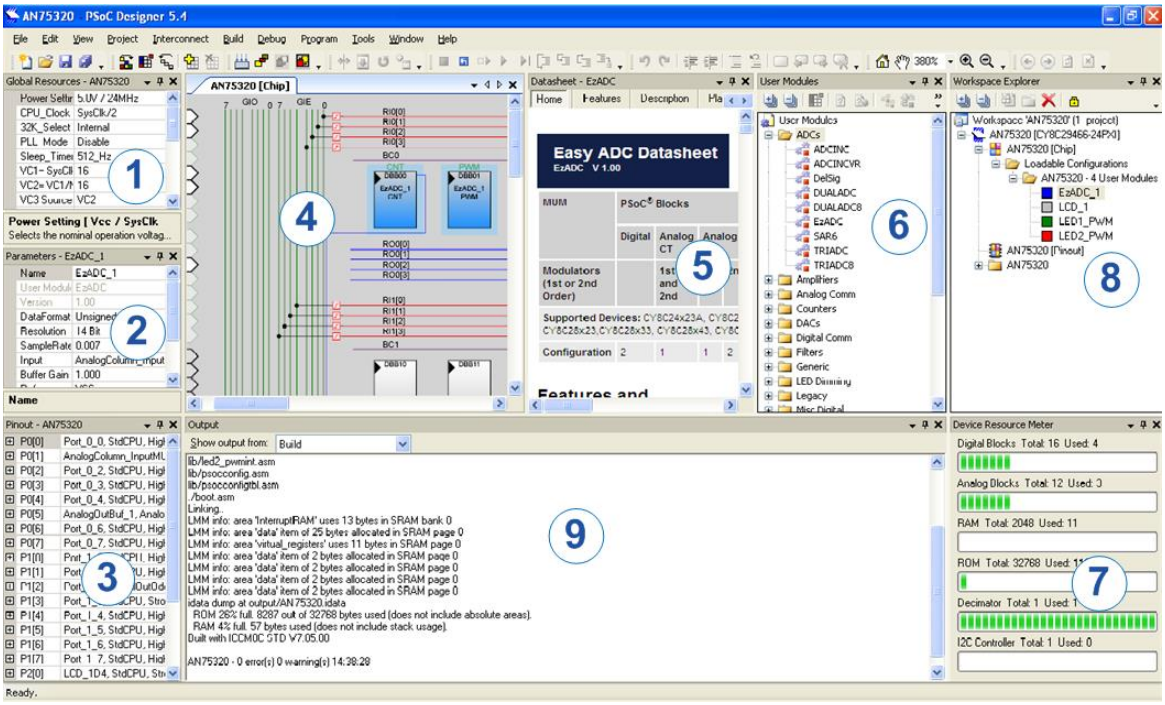
PSoC Designer is a free Windows-based Integrated Design Environment (IDE). Develop your applications using a library of pre-characterized analog and digital peripherals in a drag-and-drop design environment. Then, customize your design leveraging the dynamically generated API libraries of code. Figure 1 shows PSoC Designer windows.

Note: This is not the default view.

1. **Global Resources –** all device hardware settings.
2. **Parameters –** the parameters of the currently selected User Modules.
3. **Pinout –** information related to device pins.
4. **Chip-Level Editor –** a diagram of the resources available on the selected chip.
5. **Datasheet –** the datasheet for the currently selected UM
6. **User Modules –** all available User Modules for the selected device.
7. **Device Resource Meter –** device resource usage for the current project configuration.
8. **Workspace –** a tree level diagram of files associated with the project.
9. **Output –** output from project build and debug operations.

Note: For detailed information on PSoC Designer, go to **PSoC® Designer** > **Help > Documentation** > **Designer Specific Documents** > **IDE User Guide**.

Figure 1. PSoC Designer Layout



2.2 Code Examples

The following webpage lists the PSoC Designer based Code Examples. These Code Examples can speed up your design process by starting you off with a complete design, instead of a blank page and also show how PSoC Designer User modules can be used for various applications.

http://www.cypress.com/go/PSoC1Code Examples

To access the Code Examples integrated with PSoC Designer, follow the path **Start Page** > **Design Catalog** > **Launch Example Browser** as shown in Figure 2.

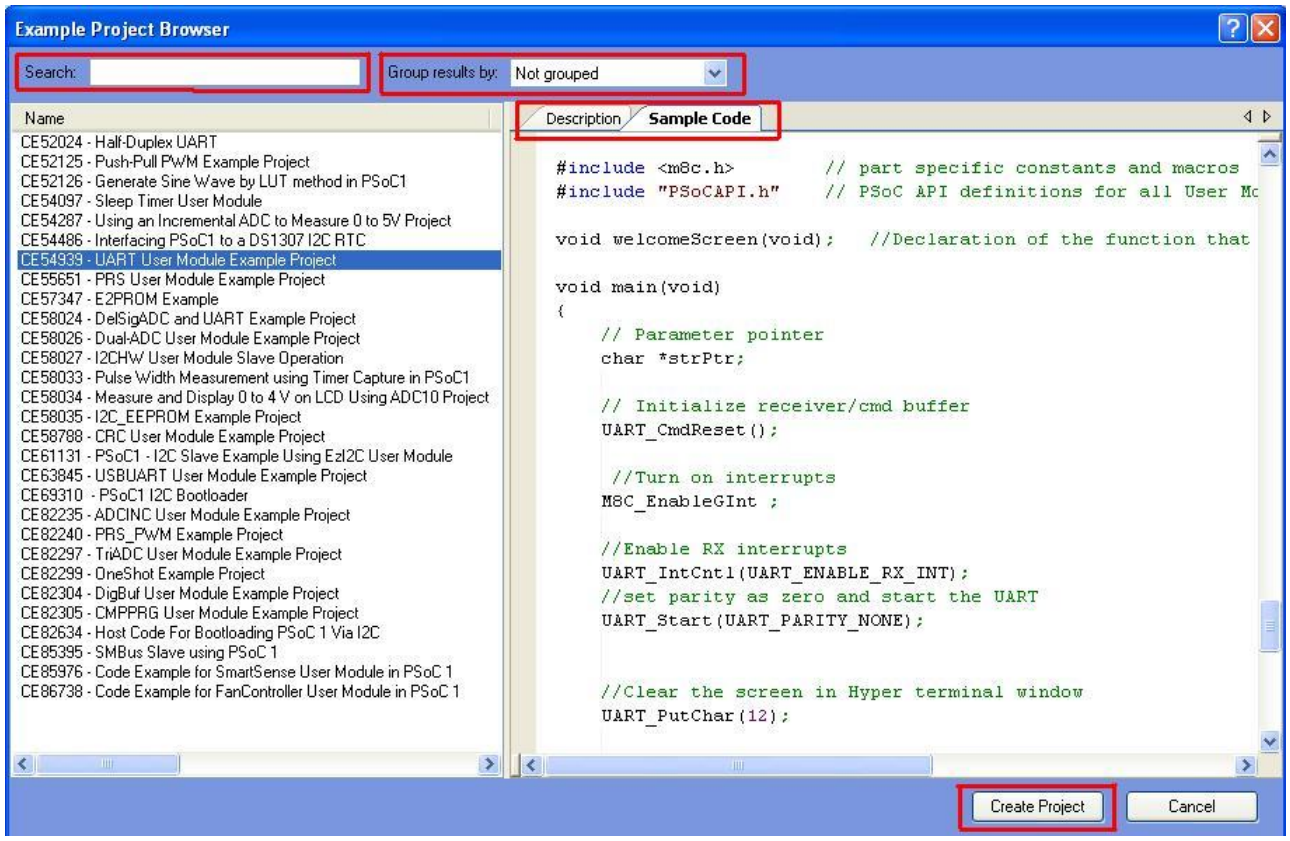
Figure 2. Code Examples in PSoC Designer



In the Example Projects Browser shown in Figure 3, you have the following options.

- Keyword search to filter the projects.
- Listing the projects based on Category.
- Review the datasheet for the selection (on the Description tab).
- Review the code example for the selection. You can copy and paste code from this window to your project, which can help speed up code development, or
- Create a new project (and a new workspace if needed) based on the selection. This can speed up your design process by starting you off with a complete, basic design. You can then adapt that design to your application.

Figure 3. Code Example Projects, with Sample Codes



2.3 Technical Support

If you have any questions, our technical support team is happy to assist you. You can create a support request on the Cypress Technical Support page.

You can also use the following support resources if you need quick assistance.

- Self-help
- Local Sales Office Locations

3 Task Scheduler

A task scheduler is a piece of program that chooses which task to run next. In this section, the various terms and concepts related to scheduling are explained.

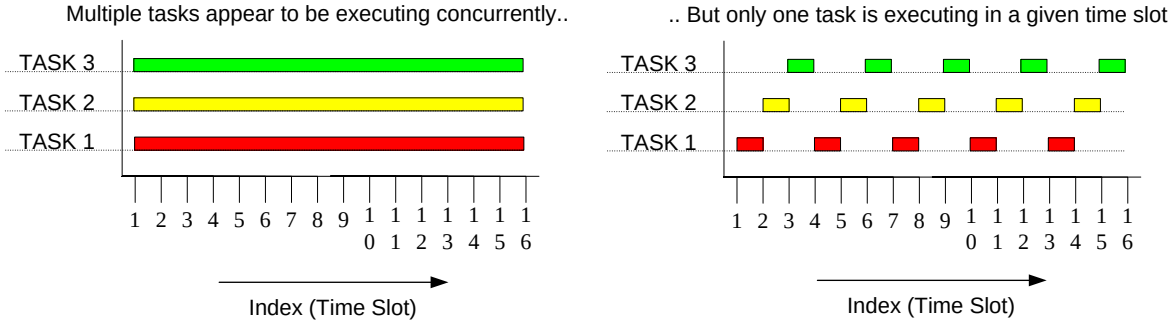
3.1 What is a Task?

A task is a small program section to do a job. It is a basic unit of resource allocation. Tasks are competing for CPU time, memory space, or other resources. A complex problem can be divided into a number of smaller and simpler tasks. Every task handles only a small portion of the problem. Sometimes a big task, in terms of execution time, can also be divided into several smaller time slots.

3.2 Time Slots

A time slot is a time interval that can be allocated to a task. A conventional CPU, such as the one in PSoC 1, can only execute a single task at a time. By rapidly switching between tasks in a program, the tasks appear to execute concurrently.

Figure 4. Execution of Tasks in Time Slots

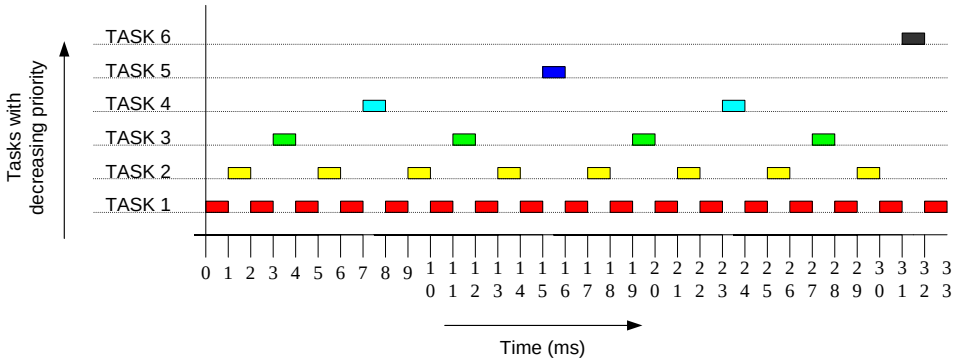


Tasks can run at regular time intervals. In an application, some tasks might be more important or time-critical than others. For example, monitoring a pin for a pulse signal of a very short duration may be more important than refreshing the display on an LCD. Each distinct task can be assigned a unique priority. Tasks that you need to execute more frequently are assigned a higher priority. Higher-priority tasks need shorter periodic rates.

Let us consider an example. In Figure 5, assume the time slot is 1 ms so Task 1 is running at a periodic rate of every 2 ms for a length of 1 ms. Task 2 executes at every 4 ms for a length of 1 ms, and so on. Here, every Task '*n*' executes half as slow (or at twice the periodic rate) as Task '*n – 1*'. The task period for every task '*n*' is in this example can be calculated using Equation 1. In this scheme, Task 1 (considered to execute more frequently than others) is considered to have the highest priority or priority 1. Task 2 has priority 2 and so on.

$$Task\ n\ Period\ =\ Time\ Slot\ Period\ x\ 2^n \qquad \text{Equation 1}$$

Figure 5. Example of Periodic Tasks

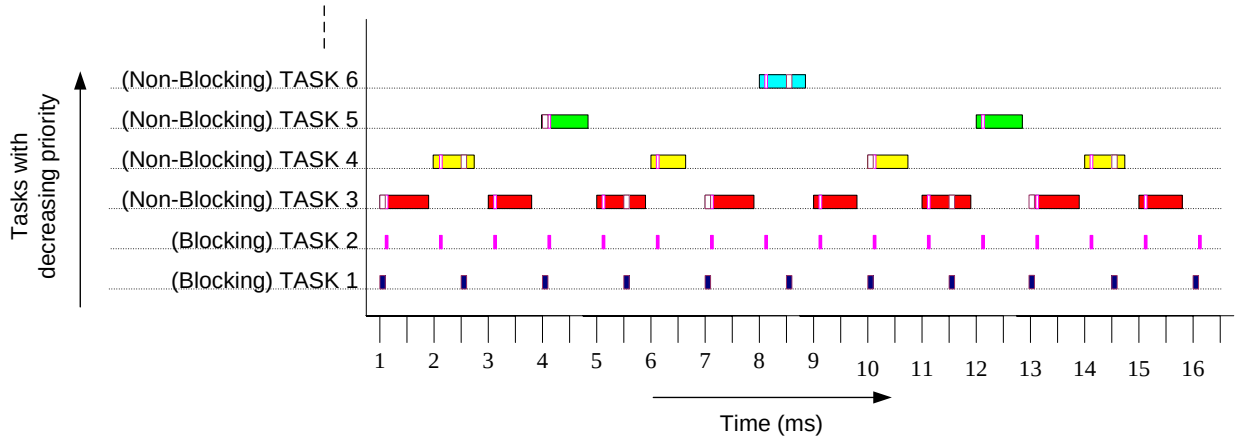


4 Implementation

The following section explains the implementation of the task scheduler shown in this application note. In this implementation, periodic time slots are used to run a number of tasks. Each task has a unique priority. Higher-priority tasks are allowed to block others; that is, a task that is blocked can continue execution only after the task which has blocked it is completed. In such implementations, ensure that the task blocking time is not excessive enough to cause problems in scheduling successive tasks. For more details, refer to Appendix A – Considerations when task blocking is used.

Let us consider an example application in which there are two types of tasks – periodic tasks and interrupts. Interrupts are high-priority tasks and are allowed to block lower-priority tasks. The periodic tasks are designed such that they do not block each other. Task 4 does not start unless Task 3 is complete and Task 5 does not start unless Task 4 is complete (see Figure 6). The only way to block other tasks is to use the interrupt services. In Figure 6, since Tasks 1 and 2 are interrupt routines, they can block any task in the periodic time slots. Usually most of the peripherals in PSoC, such as analog-to-digital converters, are handled by the Interrupt Service Routines (ISRs). Because task blocking is performed only in ISRs, switching is needed only in the ISRs.

Figure 6. Example Task Scheduler with Six Tasks



In the project implemented in this application note, there is one interrupt routine and 16 periodic tasks. Eight of these tasks are to toggle a pin corresponding to the task. Task 1 to Task 8 are dummy slots to which you can assign your own tasks. Task 9 to Task 16 toggle pins Port_2_0 to Port_2_7 respectively. The state of the pin changes each time the task executes. The time slot for the particular task can be measured by measuring the time between successive toggles on the pin associated with the task.

To implement this, a variable 'Time_Slot_Counter' is defined. This variable is 16-bits wide and each bit is reserved for each of the 16 tasks. An asserted bit indicates if a particular task needs to be executed. For example, if bit 0 of Time_Slot_Counter is '1', Task 1 is executed; if bit 1 is '1', Task 2 is executed and so on. The macro TIME_SLOT_COUNTER_BIT_*n* indicates the bit mask for Task *n*.

Table 1. Time_Slot_Counter bits

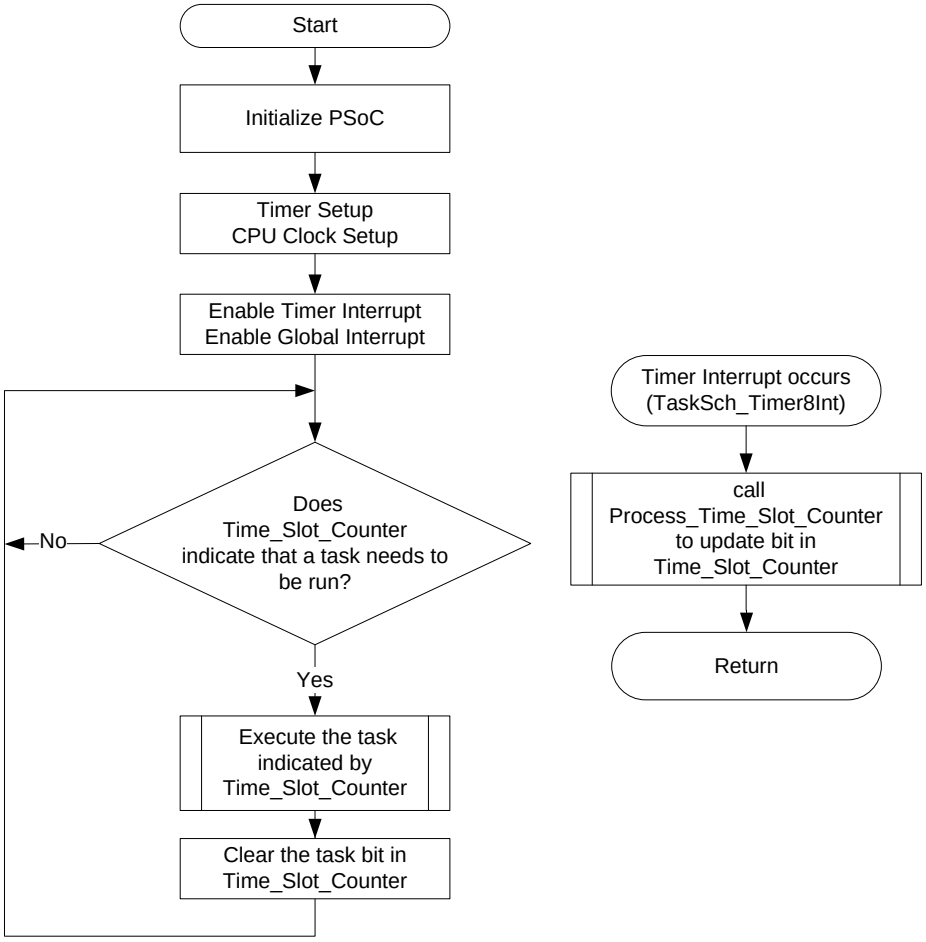
Index of Task to be run	Masked bit which should be asserted in 'Time_Slot_Counter' for task to run	Task	Time Slot Interval [ms]
Task 1	TIME_SLOT_COUNTER_BIT_00	Not assigned	0.25
Task 2	TIME_SLOT_COUNTER_BIT_01	Not assigned	0.5
Task 3	TIME_SLOT_COUNTER_BIT_02	Not assigned	1
Task 4	TIME_SLOT_COUNTER_BIT_03	Not assigned	2
Task 5	TIME_SLOT_COUNTER_BIT_04	Not assigned	4
Task 6	TIME_SLOT_COUNTER_BIT_05	Not assigned	8

Index of Task to be run	Masked bit which should be asserted in 'Time_Slot_Counter' for task to run	Task	Time Slot Interval [ms]
Task 7	TIME_SLOT_COUNTER_BIT_06	Not assigned	16
Task 8	TIME_SLOT_COUNTER_BIT_07	Not assigned	32
Task 9	TIME_SLOT_COUNTER_BIT_08	Port_2_0 toggled	64
Task 10	TIME_SLOT_COUNTER_BIT_09	Port_2_1 toggled	128
Task 11	TIME_SLOT_COUNTER_BIT_10	Port_2_2 toggled	256
Task 12	TIME_SLOT_COUNTER_BIT_11	Port_2_3 toggled	512
Task 13	TIME_SLOT_COUNTER_BIT_12	Port_2_4 toggled	1024
Task 14	TIME_SLOT_COUNTER_BIT_13	Port_2_5 toggled	2048
Task 15	TIME_SLOT_COUNTER_BIT_14	Port_2_6 toggled	4096
Task 16	TIME_SLOT_COUNTER_BIT_15	Port_2_7 toggled	8192

In the main loop, the Time_Slot_Counter is being tested for the specific 'task' bits. According to the bit position in Time_Slot_Counter, a Time_Slot_*n* function (specific task) will be called. Before execution of each Time_Slot_*n* function, the corresponding bit will be cleared to prevent an unnecessary execution in the next pass.

The Process_Time_Slot_Counter function is responsible for updating Time_Slot_Counter with the bit associated with the next task to be executed. An 8-bit timer annotated as 'TaskSch_Timer8Int', generates a periodic interrupt every 0.25 ms, which is the base time-slot duration. This interrupt has the highest priority and thus is used to invoke the Process_Time_Slot_Counter function which is called every time the interrupt triggers. Figure 7 shows the flow of the task scheduler.

Figure 7. Implementation Flowchart



5 Summary

This application note demonstrates the implementation of a task scheduler in PSoC 1 to run a number of tasks in the presence of higher-priority tasks such as interrupts.

A. Appendix A – Considerations when task blocking is used

In a scheduling scheme where higher-priority tasks are allowed to block lower-priority ones, the task scheduler must ensure that while switching, tasks are processed in a timely manner to meet their deadlines (a deadline is the time after which a task computation is not just late but also considered invalid). To ensure that the scheduler has enough time to execute all the tasks, the following condition can be used:

$$\sum_{j=1}^{n}\left(\frac{E_j}{P_j}\right) + max_j \sum_{j=1}^{n}\left(\frac{B_j}{P_j}\right) \leq n(2^{\frac{1}{n}} - 1) \qquad \text{Equation 2}$$

In this equation, “n” is the maximum number of tasks. “E” is the execution time of task “j” and “P” is the period of task “j.” “B” is the blocking time for Task “j.” The left side of this equation is called Computed Utilization and the right side is called Utilization Bound. All tasks meet their deadline as long as the Computed Utilization is less than Utilization Bound.

Table 2. Increasing Number of Tasks Reduces Utilization Bound

Number of Tasks $= n$	Utilization Bound $= n(2^{\frac{1}{n}} - 1)$
1	1.000
2	0.828
3	0.780
4	0.757
5	0.743
6	0.735
7	0.729
8	0.724
9	0.721
10	0.718
11	0.715
12	0.714
13	0.712
14	0.711
15	0.709
16	0.708

To analyze the blocking time for each task, it is only necessary to add the higher-priority tasks that block it.

Figure 8 illustrates a scheduler with six tasks. These tasks run at different periods and have different execution times.

Figure 8. Task Scheduler with Six Tasks

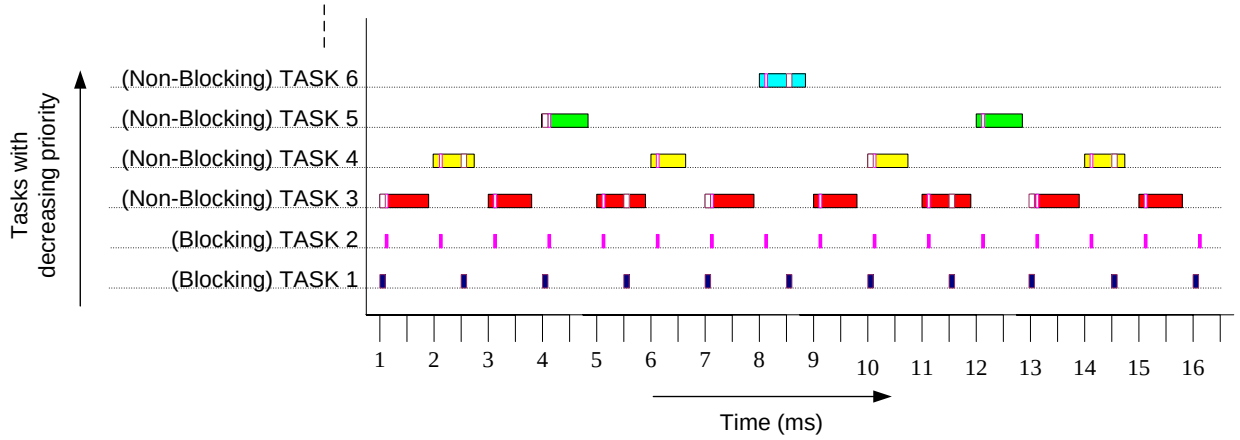


Table 3. Computed Utilization

Task	P = Period [ms]	E = Execution Time [ms]	E / P
1	2.5	0.1	0.04
2	1	0.05	0.05
3	2	0.25	0.125
4	4	0.6	0.15
5	8	0.85	0.10625
6	16	0.7	0.04375
Sum of E / P			0.515
Max of B / P			0.075
Computed Utilization			0.63375

Note that Task 1 and 2 have higher priority and can block the execution of Tasks 4, 5, and 6.

The Utilization Bound for six tasks is 0.735. (Refer to Table 2.)

To calculate maximum blocking time for Task 3, add execution time of Task 1 and 2 and divide it with the period time of Task 3. The result is (0.1+ 0.05) / 2 = 0.075 ms.

Because the Computed Utilization (0.633) is smaller than Utilization Bound (0.735), the scheduler meets all the deadlines.

Document History

Document Title: AN2046 – PSoC® 1 Task Scheduler

Document Number: 001-40921

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1536344	GYV	10/03/2007	New Application Note.
*A	2887416	RLRM	03/04/2010	Updated document and project files with PSoC Designer 5.0 content. Added Summary and Document History
*B	3173874	KANT	02/16/2011	Updated title. Updated abstract. Updated Introduction (To reflect project associated with this application note is a simple scheduler and is only for PSoC 1 devices). Updated RTOS and PSoC instances to reflect the title. Updated the associated project and compiled with latest PSoC Designer software.
*C	4330603	MSUR	04/02/2014	Update in new template. Completing Sunset Review.
*D	4650189	ASRI	02/03/2015	Updated document and project files with PSoC Designer 5.4 content Sunset review
*E	4852786	ARVI	07/30/2015	Sections re-written to demonstrate a Task Scheduler in PSoC 1. Updated title. Update template. Added PSoC Resources. Updated project files to PSoC Designer 5.4 SP1.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc
Memory	cypress.com/go/memory
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

PSoC is a registered trademark of Cypress Semiconductor Corp. "Programmable System-on-Chip" and PSoC Designer 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
Website : www.cypress.com

© Cypress Semiconductor Corporation, 2007-2015. 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.