Using Sleep and Hibernate Modes in PSoC® 3

AN66083

Author: Kantesh Kudapali
Associated Project: No
Associated Part Family: CY8C3xxxx
Software Version: PSoC® Creator™ 1.0
Associated Application Notes: None

Application Note Abstract

In Sleep and Hibernate modes, devices conserve power while being able to switch back quickly to active mode. This application note focuses on these two power-saving modes and the mechanisms available to enter and exit them in PSoC® 3 devices. It also discusses several different wakeup sources and ways to reduce power consumption.

Introduction

Most microcontrollers in today's marketplace provide fixed mechanisms for switching devices to low-power modes. As systems become more complex, requiring low-power operation, system designers seek more flexibility in controlling the system power.

PSoC 3 stands out from the competition by providing various clock and power gating mechanisms to allow precise control over operating frequency and power. This makes PSoC 3 ideal for low-power applications such as battery-operated or handheld devices where power consumption is crucial without compromising on performance.

With the state of the art PSoC Creator development platform; configuring PSoC 3 for various low power modes is relatively simple and easy. With the rich library APIs the user need not look at the various registers and their definitions to use many of the reconfiguration options available with the device.

In order to keep the application note simple for beginners, the clock and power gating mechanisms and the Alternate active mode of operation available with PSoC 3 devices are not discussed. These will be covered in a separate application note.

Power Modes in PSoC 3

PSoC 3 devices provide four power modes: Active, Alternative Active, Sleep, and Hibernate.

Active Mode

Upon reset or wakeup, PSoC 3 devices automatically enter Active mode. In this mode, you can enable or disable any of the subsystems. You can also enter any of the other three power modes by calling the appropriate API. For example, you can call `CyPmSleep()` to force the system into Sleep mode.

Alternate Active Mode

Alternative Active mode provides a quick way to enable a different set of subsystems on the device. You can enter this mode by calling `CyPmAltAct()`.

In Alternative Active mode, the CPU is in a halt state and other enabled subsystems continue to function. An interrupt from the active subsystems brings the system back to Active mode.

Sleep Mode

In Sleep mode, the CPU and all subsystems except supervisory modules such as CTW and WDT (if enabled) are disabled. The device consumes as low as 0.6 µA of current with 3.3 V V_{DD}. You can enter this mode by calling `CyPmSleep()`. Any configured pin, comparator, or supervisory interrupts can bring the system back to Active mode.

The following table lists the wakeup sources for Sleep mode along with typical power consumption and associated wakeup time.

Wakeup source	Wakeup time (µs)	Typical Sleep current (µA)
1PPS (RTC)	< 15	< 1.0
CTW	< 15	< 0.6
PICU	< 15	< 1.0
Comparator	< 15	< 6.0
LVD	< 15	< 1.0
I2C	< 15	< 1.0

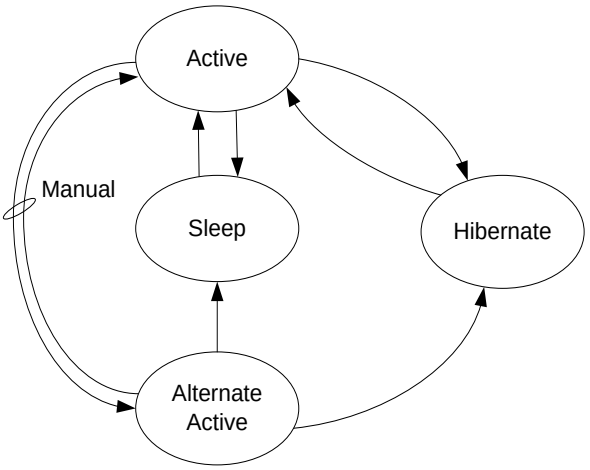
Hibernate Mode

In Hibernate mode, a PSoC 3 device consumes typically 200 nA. The CPU and all peripherals and supervisory modules are powered off. This mode can be entered by calling `CyPmHibernate()`. Any configured pin interrupt can bring the device back to Active mode.

A device can wake up from Hibernate mode with a PICU interrupt or by resetting the device; all other resources on the device are turned off to save power. A device can take as long as 100 µs to wake up from Hibernate mode.

Figure 1 shows the state transition of PSoC 3 power modes.

Figure 1. Power Modes



Switching To Sleep or Hibernate Mode

Use this procedure to switch a device to Sleep or Hibernate mode:

1. Call the `component`[1]`_Sleep()` function. This function disables the power, clock, or both depending on the component and keeps the component in lowest power consumption mode. You should define one global sleep function and call all of the component sleep functions within that global function.
2. Call the `CyPmSaveClocks()` function. This function saves the system clock configuration and other clock configuration data. This data may be required to prepare the device for entering sleep. This function also turns off PLL along with digital and analog clock dividers.
3. Make sure that you clear the interrupt flags of all wakeup sources, and enable them if using them as wakeup sources.
4. Call one of the following functions:
   - CyPmSleep(wakeupTime, wakeupSource) : Puts the device into Sleep mode after configuring the wakeup time and source.
   - CyPmHibernate() : Puts the device into Hibernate mode after configuring the wakeup source.

After wake up from Sleep or Hibernate, use the following procedure:

1. Clear the interrupt request flag that caused the system to wake up either in the ISR service routine or immediately after the Sleep or Hibernate function call. This can be done by calling the function `component`[1]`_Status()` in the interrupt service routine if the interrupts are enabled. Otherwise, call this function immediately after the Sleep or Hibernate function call.
2. Read the status register of the component that caused the wakeup to clear the interrupt status. Incase of CTW or 1PPS event call CyPmReadStatus() function to clear the interrupt request.
3. Call the `CyPmRestoreClocks()` function. This function restores the analog and digital divider clock configuration, enables PLL (if it was enabled before entering Sleep or Hibernate mode), and enables the crystal oscillator if that was enabled before switching modes.
4. Call the `component`[1]`_Wakeup()` function. This function restores the previously stored configurations and sets up the module for normal functioning. It is always a good practice to define one global wakeup function and call all of the component functions within that global function.
5. In order to reduce the wakeup time, check for the wakeup source which caused the device to wake up, and enable components which are required to run the application for that wakeup source. This will reduce the wakeup time considerably when system is expected to wake up from multiple sources and carry out a specific task on each respective wakeups.

For more information on APIs related to Sleep and Hibernate modes, see the Appendix.

Note 1: The `component` refers to the instance name of component placed on the schematic, please replace this with component's actual instance name

Reducing Power Consumption during Sleep Mode

Use the following tips to minimize power consumption during Sleep mode:

- Keep all unused I/O pins in an analog high impedance state (default state).
- To further reduce the radiated EMI susceptibility, tie all unused I/O pins to ground with a pull-down resistor of around 1 kΩ. (Example for automotive ACS100 or Medical FDA qualification)
- If possible, drive all I/O pins that are connected to load to their inactive state.
- Maintain idle state on communication lines by configuring them digital output and driving "Ideal state" level on those pins. For example, I2C and SPI normally have a high ideal state. Before going to sleep, maintain the high state on these lines to avoid generating spurious transitions during wakeup.
- Configure special input/output (SIO) pins as output and drive to high or low state.
- Configure SIO pins to use the Vddio drive level to avoid Vref being on during sleep.

  The following is example code to set port12 to use Vddio as the drive level.

Example code to setup Vddio as drive level for PORT12

```
/* set PRT12_AG register to use Vddio for sleep */
CY_SET_REG8(CYDEV_IO_PRT_PRT12_AG, (CY_GET_REG8(CYDEV_IO_PRT_PRT12_AG) & 0x3F));
/* set PRT12_SIO_CFG register to use Vddio for sleep */
CY_SET_REG8(CYDEV_IO_PRT_PRT12_SIO_CFG, (CY_GET_REG8(CYDEV_IO_PRT_PRT12_SIO_CFG) & 0x3F));
/* set PRT12_SIO_DIFF register to use Vddio for sleep */
CY_SET_REG8(CYDEV_IO_PRT_PRT12_SIO_DIFF, (CY_GET_REG8(CYDEV_IO_PRT_PRT12_SIO_DIFF) & 0x3F));
```

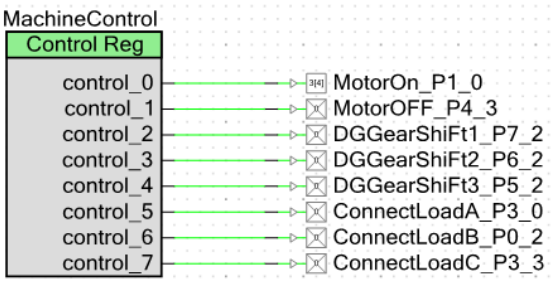
Entering Sleep or Hibernate Mode

Note the following when a device enters Sleep or Hibernate mode:

- Allow the component to complete ongoing operations before calling its associated sleep function.
- The component's sleep function suspends all ongoing operations immediately and puts the component into low power mode. Check the status of the corresponding component before calling a component's sleep function. For example calling UART_Sleep() function will suspends the ongoing data transmission immediately, check for the transmission complete status before calling the UART_Sleep() function.
- Do not use control registers to set the I/O pin state if pin state needs to be retained during Sleep mode. The control registers are non retention type; that is the value of control register is not retained during sleep. When device enters sleep mode the contents of control registers become '0' because of this the output pin state also becomes low even though the port pin was set to high before sleep. This may create a problem if port pin value need to be maintained high during low power mode.

  Consider the following schematic in which there are 8 different IO port pins connected to a control register. Though this arrangement provide a flexibility to write all IO port pins connected to control register with a single instruction; during Sleep mode as the control register loses the value IO ports driven through this control registers will become low even though if they are set before the Sleep mode. This may change the system behavior.

Figure 2 Driving IO Ports with A Control Register



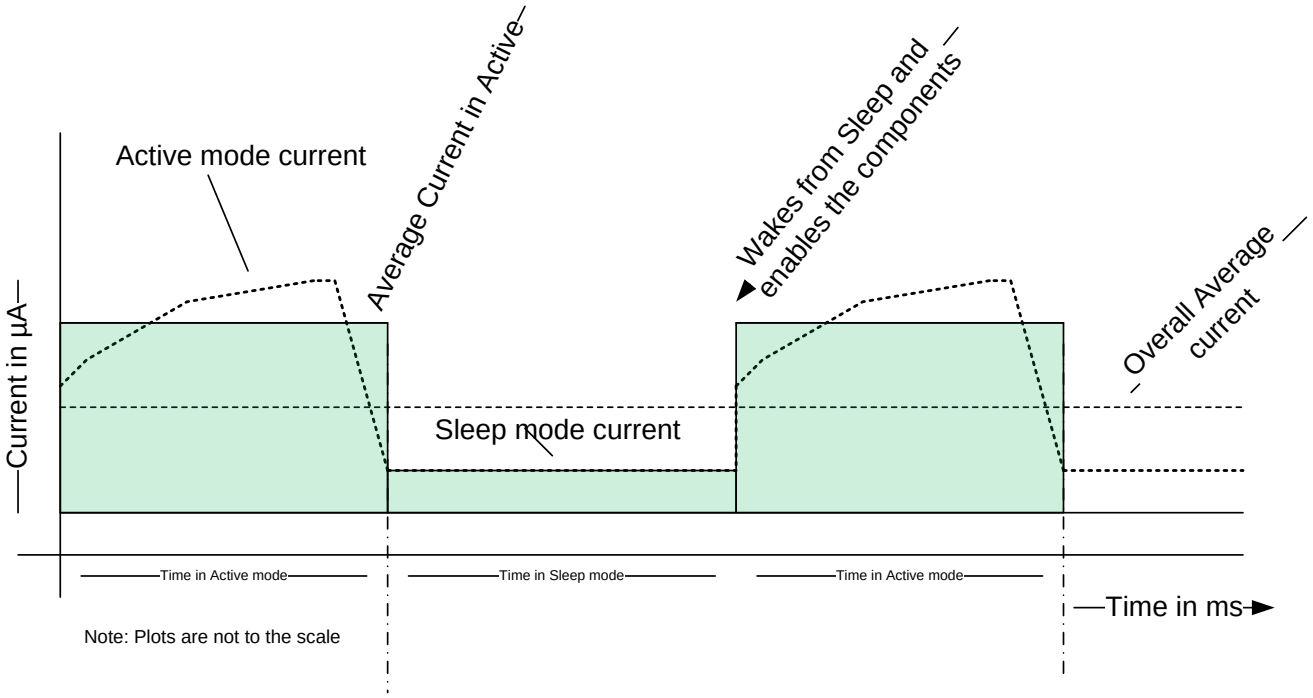
Reducing Power Consumption in Active Mode

Consider these tips to reduce the overall device power consumption in Active mode:

- Stop all components that you are not using.
- Make sure that all the unused IO pins placed (if any) on top design are configured as analog HiZ to avoid driving any loads connected outside the part.
- Choose the appropriate operating frequency for the device:
  - If the system is expected to always operate in Active mode, run the system at the minimum possible frequency to reduce the average active current.
  - Configure the IMO for 3 MHz and use the PLL and master clock dividers in the clock configuration to achieve the required system frequency.
  - Please refer "AN60631 - PSoC® 3 Clocking Resources" for more details on selecting appropriate clocks
- Following are the advantages of using IMO at 3MHz along with PLL to get any desired operating frequency
  - The IMO is more precise at 3MHz than at higher frequencies.
  - The PLL consumes less power than the IMO at the same output frequency.
  - Using the IMO to source the PLL will result in less overall frequency error.
- Try to use IMO instead of an external crystal if the system can tolerate up to 1 percent frequency accuracy. The IMO takes less power compared to an external crystal at the same frequency.
- Overall the average energy consumption of the entire system depends on:
  - The average power consumption in Active mode
  - The average power consumption in Sleep mode
  - The time it stays in Active mode
  - The time it stays in Sleep mode

Figure 3 shows the average power consumption of the overall system when switching between Active and Sleep modes.

Figure 3. Average system power consumption versus time



Switching to Sleep Mode with RTC as a Wakeup Source

When a device is required to wake up and run certain functions with a periodicity of a second or seconds, and to optionally keep track of real time, you can use this configuration. The device uses the inexpensive external watch crystal as the time base for the RTC component. The internal low-frequency oscillator(ILO) can be turned off in Sleep mode if the watchdog timer is not enabled to further reduce the Sleep mode power consumption.

In Sleep mode, PSoC 3 consumes less than a microampere to keep the internal RAM intact and the external low-power watch crystal running, which is required for waking the controller.

Refer to the example project, "EP56186 - Enter Sleep Mode and Wake up Using RTC – PSoC 3 / PSoC 5" on the Cypress website to download the complete project and step-by-step setup.

Switching to Sleep Mode with CTW as a Wakeup Source

When the device is required to wake up periodically with a configurable time between 2 ms to 4 s and execute certain timely tasks then this mode can be used. The device will use internal ILO as the time base for the timer.

If the system does not have an external watch crystal and still requires a real time cock feature with lesser accuracy the CTW can be used by configuring the system to wakeup using the CTW every second. This will eliminate the cost of an external watch crystal and loading capacitors on crystal pins.

In Sleep mode, a PSoC 3 device takes less than a microampere to keep internal RAM intact and to keep the internal low-speed oscillator running, which is required for waking the controller. Active mode power consumption depends on the components used in the system, the loads connected to I/O pins, and the state of each I/O pin.

Refer to the example project "EP58282 - System WakeUp Using Sleep Timer – PSoC 3 / PSoC 5" on the Cypress website to download the complete project and step-by-step setup.

Switching to Sleep Mode with Comparator as a Wakeup Source

When the device is required to wake up if an analog external signal rises above or falls below a set level, this mode can be used. The ILO can be turned off to further reduce the sleep power if the watchdog timer is not enabled.

The device can be configured to wake up from any number of its four comparators.

With the comparator as the wakeup source, the device requires approximately 3 to 5 µA of current to keep the Vref and comparator powered.

Switching to Sleep Mode with I2C Address Match as a Wakeup Source

When the PSoC device is used as a I2C slave and required to be in active mode only when a master requests to process some task this mode can be used.

In this mode, the internal I2C hardware continues to monitor the bus and wakes the device when the address transmitted via the I2C bus matches the address configured to wake up the device.

In Sleep mode, the PSoC 3 takes 1 µA of current to keep the I2C module active. The ILO can be turned off to further reduce the sleep power if the watchdog timer is not enabled. The Active mode power consumption depends on the components used in system, the loads connected to I/O pins, and the state of each I/O pin.

Switching to Hibernate Mode with PICU as a Wakeup Source

One of the most widely used applications for this setup is a soft power switch, in which the device is always in power-down mode and wakes up when user presses a key.

In Hibernate mode, the device requires as low as 200 nA of current to retain the internal RAM and requires approximately 100 µs to wake up the controller after the trigger event. Any of the I/O pins can be configured to wake up the device from Hibernate mode.

Refer to the example project "EP56317 - Enter Hibernate Mode and Wakeup Using PICU – PSoC 3 / PSoC 5" on Cypress website to download the complete project and step-by-step setup.

Summary

There are many different methods/features employed in designing the PSoC 3 device in order to reduce the overall power consumption of the device. This application note discussed the low power modes, switching between the Sleep, Hibernate and Active modes, By using the steps, methods provided in this application note; the user can achieve low power in Sleep and Hibernate modes. In order to ease the Sleep and Hibernate modes in end applications one can download the appropriate example projects using the reference links provided.

The more advanced topics on low power applications using PSoC devices will be discussed in separate application notes and the references will be updated in this document once the application notes are available.

Appendix

Function Reference Guide

component_Sleep(void): Stores all component configuration data to backup RAM, disables the clocks, and turns off the power to the component. Because the component ceases to work immediately after this function call, the user must ensure that the component is idle before calling the sleep function to avoid any unwanted system behavior.

CyPmSaveClocks(void): Saves all user clock settings into backup RAM and turns off the PLL. If the device is running on any oscillator other than IMO, this function disables the selected oscillator, enables IMO at 3 MHz and reconfigures the system clock settings to run the device after wakeup.

CyPmSleep(wakeupTime, wakeupSource) : Puts the part into sleep mode after configuring the wakeup time and wakeup source.

If wakeup time is configured through sleep or the RTC component on the top design, the user can call this function by passing the NONE parameter as follows:

CyPmSleep(PM_SLEEP_TIME_NONE, PM_SLEEP_SRC_ONE_PPS);

The previous function puts the device into Sleep mode but does not alter the sleep configuration set by the sleep component.

CyPmHibernate(void): Puts the component into Hibernate mode after configuring the PICU source to wake up the controller from Hibernate mode.

component **_Status(void):** Returns the status register value of the component and clears the interrupt request flag of the wakeup source.

CyPmRestoreClocks(void): Restores the clock sources and enables the user-configured clock source and frequency. This function also restores the flash read wait cycles.

component **_Wakeup(void):** Restores the respective component operation by reconfiguring the component with user settings and enabling the clock and power to the component.

Refer to the system_reference_guide.pdf and respective component data sheets for more information on all APIs related to low power.

Document History

Document Title: Using Sleep and Hibernate modes in PSoC® 3 – AN66083

Document Number: 001-66083

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3121920	KANT	12/28/10	New Application note.

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