AN219528

PSoC 6 MCU Low-Power Modes and Power Reduction Techniques

Author: Brian Lee

Associated Part Family: All PSoC® 6 MCU parts

Related Documents: For a complete list, click here.

More code examples? We heard you.

To access an ever-growing list of hundreds of PSoC code examples, please visit our code examples web page. You can also explore the PSoC video library here.

AN219528 describes how to use PSoC 6 MCU power modes to optimize power consumption. Major topics include low-power modes in PSoC 6 MCU devices, and power management techniques using PSoC 6 MCU features. Associated code examples demonstrate different low-power techniques.

Contents

1 Introduction

PSoC 6 MCU gives the best power-saving benefit when low-power modes are implemented with other power-saving features and techniques, without significantly sacrificing the performance. This application note describes not only general power-saving methods but also the unique low-power modes in PSoC 6 MCU. It also discusses other low-power considerations.

This application note requires a basic knowledge of the PSoC architecture, and the ability to develop a PSoC 6 MCU application using PSoC Creator™ or ModusToolbox™. If you are new to PSoC 6 MCU, see AN221774 - Getting Started with PSoC 6 MCU or AN210781 – Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity on PSoC Creator. If you are new to PSoC Creator, see PSoC Creator™ IDE.

If you are designing an Internet of Things (IoT) system, there are additional connectivity-related power considerations that impact total system power. While these considerations are outside the scope of this application note (AN), there are specific application notes focused on low-power IoT systems. AN227910 – Low-Power System Design with CYW43012 and PSoC 6 MCU discusses optimizations for low power in Wi-Fi, Bluetooth, and PSoC 6 MCU systems.

2 Power Modes

2.1 Power Mode Transitions

PSoC 6 MCU features seven power modes that are divided into system modes that affect the whole device, and standard Arm® CPU modes that affect only one CPU. The system power modes are Low-Power (LP), Ultra-Low-Power (ULP), deep sleep, and hibernate. The Arm CPU power modes are active, sleep, and deep sleep; these are available in system LP and ULP power modes. Table 1 lists the power modes in which the devices operate.

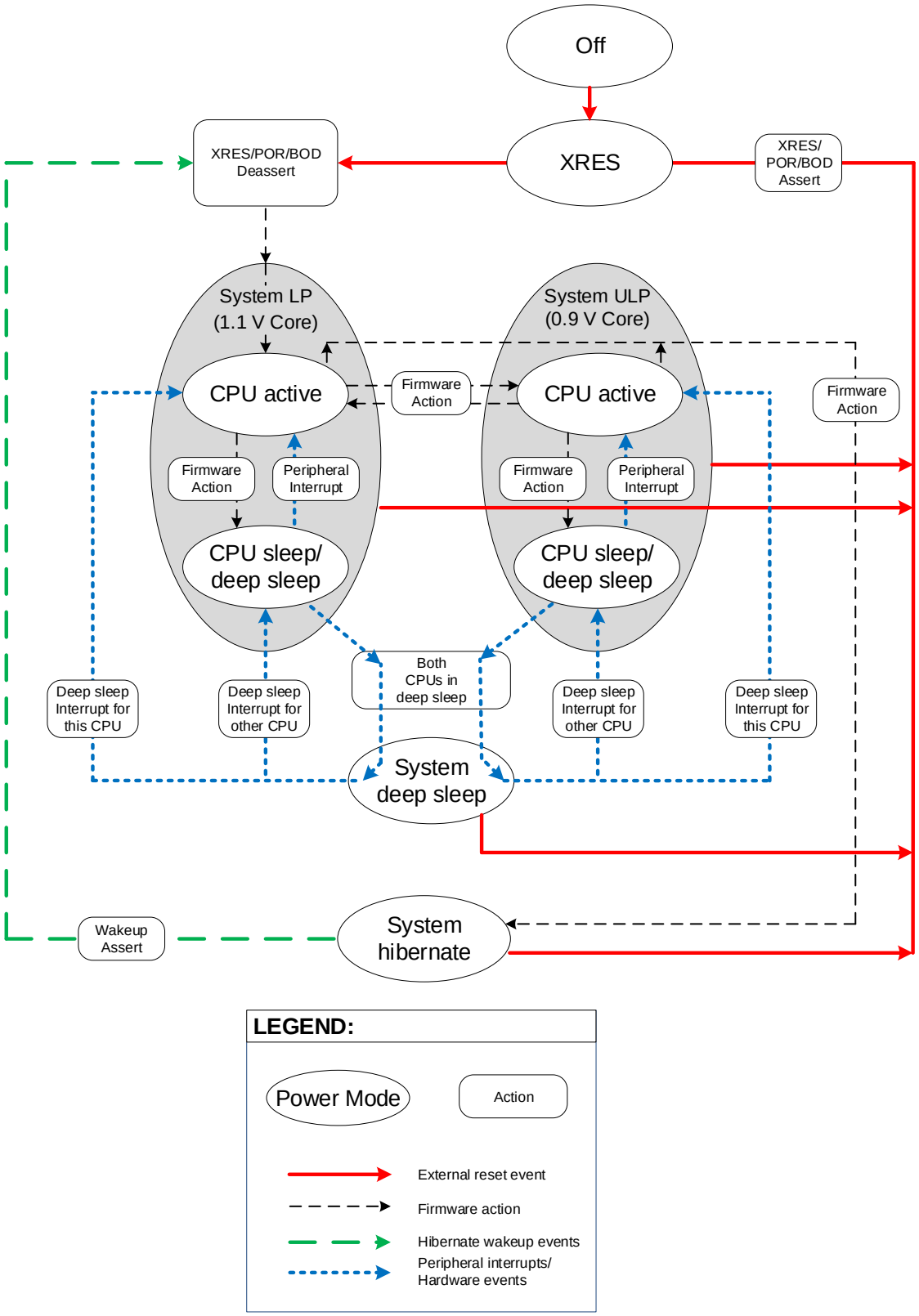
Table 1. PSoC 6 MCU Power Modes

Power Mode	Description
System LP	• All resources are available with maximum power and speed • All CPU power modes supported
System ULP	• All blocks are available, but the core voltage is lowered resulting in reduced high-frequency clock frequencies • All CPU power modes supported
CPU active	• Normal CPU code execution • Available in system LP and ULP power modes
CPU sleep	• CPU halts code execution • Available in system LP and ULP power modes
CPU deep sleep	• CPU halts code execution • Requests system deep sleep entry • Available in system LP and ULP power modes
System deep sleep	• Occurs when all CPUs are in CPU deep sleep • CPUs, most peripherals, and high-frequency clocks are OFF • Low frequency clock is ON • Low-power analog and some digital peripherals are available for operation and as wakeup sources • SRAM is retained
System hibernate	• CPUs and clocks are OFF • GPIO output states are frozen • Low-power comparator, RTC alarm, and dedicated WAKEUP pins are available to wake up the system • Backup domain is available • SRAM is not retained

Figure 1 shows how power mode transitions are based on different events and actions, including interrupts, firmware actions, and reset events. In some cases, mode transitions are done through multiple modes.

For more detailed information, see PSoC 6 MCU Architecture Technical Reference Manual and Appendix A.

Figure 1. PSoC 6 MCU Device Power Mode Transition

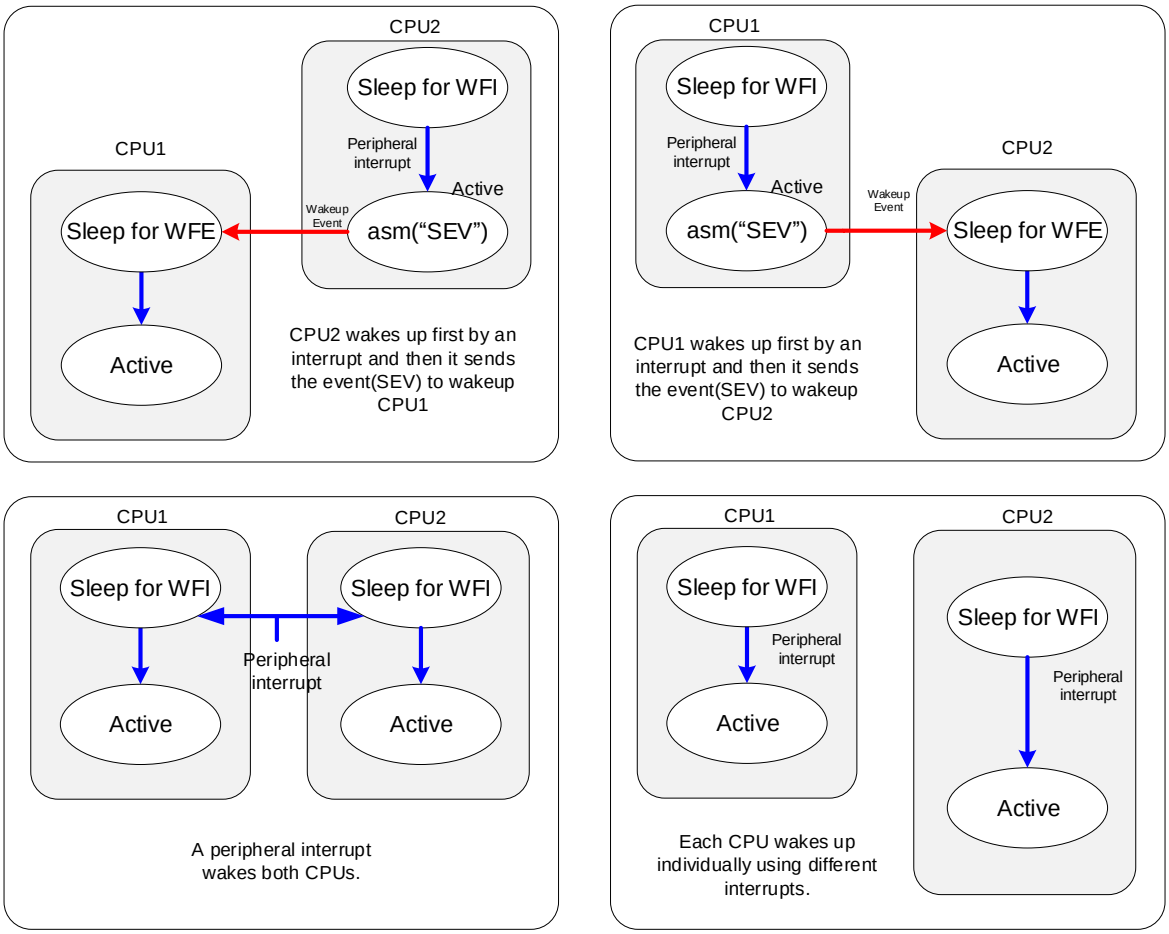


2.2 CPU Sleep and Wakeup Instructions

Arm Cortex® CPUs transition between sleep and wakeup independently. Figure 2 shows several scenarios of wakeup from sleep.

Wait-for-Interrupt (__WFI) is the core sleep instruction. After a CPU executes __WFI, the CPU goes to sleep and stays in sleep until any interrupt is asserted. Wait-for-Event (__WFE) is similar to __WFI, but it wakes up when the wakeup event is received instead of an interrupt. Set Event (__SEV) is used for waking up other CPUs in sleep mode because of a __WFE. CPU deep sleep uses the same instructions for sleep and wakeup, but the SLEEPDEEP bit[2] of the Arm System Control Register (SCR) is set before a sleep instruction. For more information on SCR, see Arm System Control Register User Guide. This process is implemented in SysPm PDL library APIs.

Figure 2. Multi CPU Sleep and Wakeup Cases



CPU power modes are different from system power modes. Figure 2 shows that each CPU supports its own sleep modes, independent of the state of the other CPU. The device is in system deep sleep mode when both CPUs are in deep sleep. For more detailed information, see AN215656 – PSoC 6 MCU Dual-CPU System Design.

2.3 Low-Power Assistant

2.3.1 Low-Power Assistant Features

The Low-Power Assistant (LPA) provides an easy-to-use GUI for setup of both device and system power options. The LPA's goal is to aid in quickly attaining datasheet power numbers in real-world user applications. For each power option supported in the device, the ModusToolbox Device Configurator has corresponding sections to configure power resources in the same method as configuring any other peripheral.

To launch the assistant, for PSoC 6 MCU devices, from under the PSoC 6 MCU part number tab and **System** sub-tab, select **Power** resource, as shown in Figure 3A. For connectivity devices, from under the Wi-Fi or Bluetooth device part number tab, select the **BT** or **Wi-Fi** Resource in the **Power** section, as shown in Figure 3B. See the LPA documentation in the Documentation section located at the top of the Power Configurator.

Figure 3A. PSoC LPA Selection

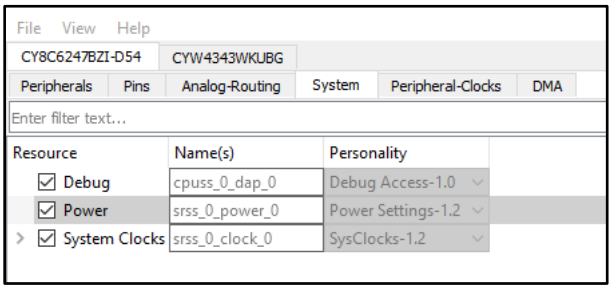
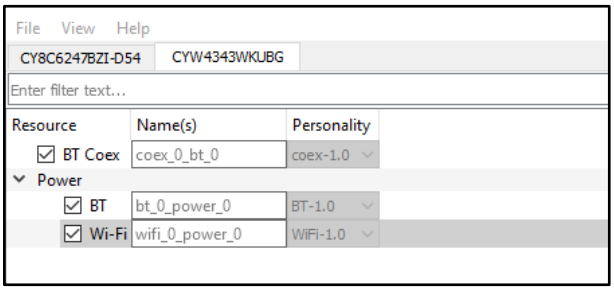


Figure 3B. Connectivity device LPA Selection

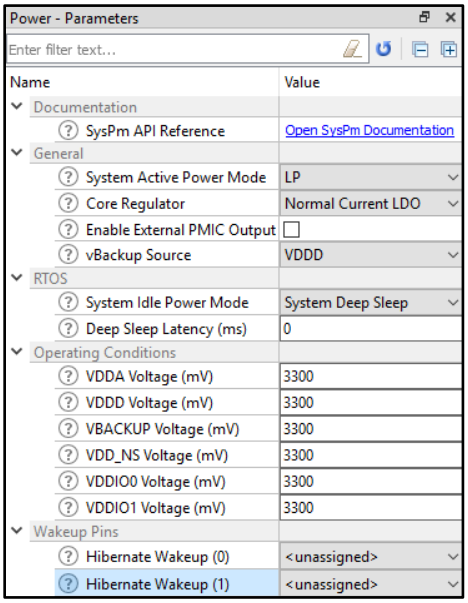


2.3.2 PSoC 6 MCU Device LPA Settings

Use the PSoC 6 MCU device section of the LPA to configure the initial power configuration of the device as shown in Figure 4. Use the **General** section of the LPA to configure the highest power settings required in the design for system power mode, core regulators, and Vbackup domain features. After the highest power mode is configured, use the **RTOS** section to configure the lowest power mode the system automatically transitions into. In many designs, this is all the power mode configuration required, because the RTOS typically handles all the dynamic mode transitions. For more advanced use cases, the HAL or SysPm PDL library power functions can be called at any time to dynamically modify any of the power settings or initiate a power mode transition.

CPU and system wakeup from sleep and deep sleep power modes occur when any configured peripheral interrupt is triggered. You can configure the sleep and deep sleep wakeup interrupts by enabling the desired wakeup peripheral's interrupt in that peripheral's configurator within ModusToolbox Device Configurator. If your system enters hibernate mode at runtime using HAL or SysPm calls, the LPA **Wakeup pins** section simplifies the selection of hibernate wakeup sources.

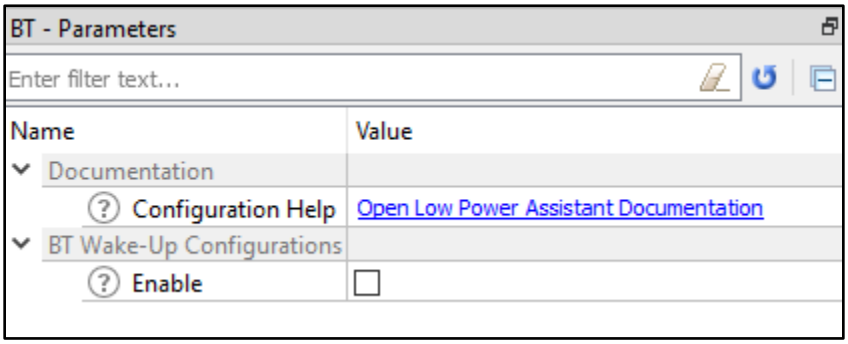
Figure 4. PSoC LPA Power Parameters



2.3.3 Connectivity Device LPA Settings

Use the “Connectivity device” section of the LPA to configure Wi-Fi and Bluetooth power options. Cypress Bluetooth connectivity devices are designed to automatically operate in their lowest power modes. The primary power option is to enable the BT device host wakeup trigger as shown in Figure 5. BT host wakeup allows the PSoC 6 MCU host device to enter a system deep sleep or hibernate mode and wait for the BT connectivity device to wake up the MCU by generating a GPIO pin interrupt when it is required to process BT operations.

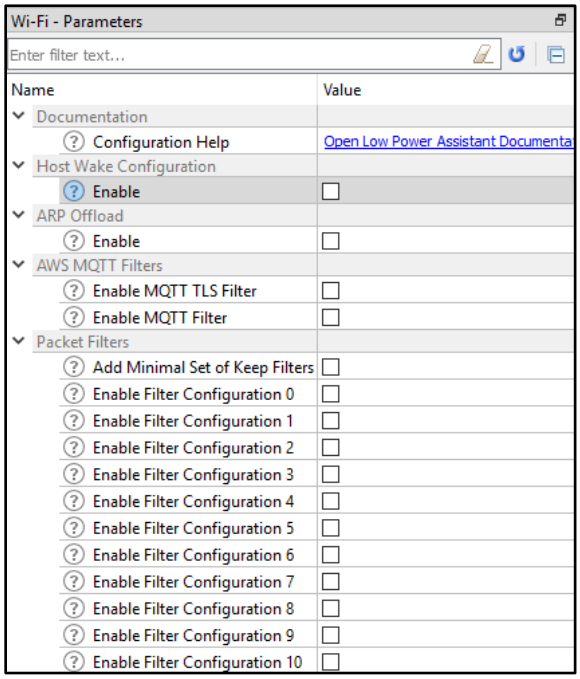
Figure 5. Bluetooth LPA Power Parameters



Cypress Wi-Fi connectivity devices support a wide range of power optimization techniques that you can configure based on the needs of the application and network features supported, as shown in Figure 6. The three primary low-power features for Wi-Fi are host wakeup, offloads, and filters.

- **Host wakeup** allows the PSoC 6 MCU host to enter system deep sleep or hibernate mode until the Wi-Fi device requires MCU processing. When the host MCU must wake up, the Wi-Fi device triggers a GPIO pin interrupt. A host wakeup is required to use most of the other low-power Wi-Fi features.
- **Offloads** allow the MCU’s device stack processing to be moved (offloaded) to the Wi-Fi device internal processor. This increases the time the MCU host can devote to other tasks or spend in a low-power mode.
- **Filters** run on the Wi-Fi device and avoid waking the host MCU until the filter conditions are met. An example is an IoT device that needs to respond only to MQTT packets. The device can enable MQTT filters to ignore all other network traffic allowing the host MCU device to stay longer in a low-power mode.

Figure 6. Wi-Fi LPA Power Parameters



2.4 Subsystem Availability and Power Consumption

2.4.1 Subsystem Availability

Each subsystem resource works differently in various system power modes. For example, the CPU can be in ON, OFF, and Retention modes. It is important to select proper peripherals for the power mode to work correctly. Table 5 lists the resources available in different power modes.

2.4.2 Approximating Power Consumption

See the device datasheet for power consumption data for given conditions. Because there are different combinations to achieve the best power consumption, the actual power consumption of the application can be different from the datasheet.

2.4.3 Power Estimator

ModusToolbox IDE version 2.0 provides the Cypress Power Estimator (CyPE) tool, which estimates the power consumed by a target device or platform dynamically at runtime. This tool helps you determine the power in different operating modes and how power changes under actual system conditions.

Figure 7. CyPE Tool

2.5 Example Case Scenarios

Proper power mode selection reduces power consumption without performance degradation. Table 2 lists sample scenarios of power modes. In some examples, only a few power modes are used effectively.

Table 2. Sample Case Scenarios of Power Modes

Power Modes	Wearable Device	Air Conditioner	Remote Controller	Thermometer
System LP CPU active	GUI interaction by user	Motor run	–	Communicates over BLE
System ULP CPU active	Processes heartbeat	–	Sends command	Reads temperature Updates result on LCD
System LP CPU sleep	–	–	–	–
System ULP CPU sleep	Analog block detects heartbeat	–	–	–

Power Modes	Wearable Device	Air Conditioner	Remote Controller	Thermometer
System deep sleep	Goes to deep sleep when the device does not detect heartbeat for 30 seconds (device is not in use)	Waits for command No motor run Wakeup by infra-red (IR) triggering	–	Wakes up every 1 second using watchdog timer (WDT)
System hibernate	Low battery – Does nothing Resets device when charger is plugged in	–	Waits for button press	–

2.6 System Power Management (SysPm) Library

2.6.1 Overview

The Cypress Peripheral Driver Library (PDL) is a complete software tool that includes APIs for configuring peripherals and system registers to implement the desired functionality. PDL provides direct access to almost all hardware resources of the target device. It reduces the need to understand and directly access registers and bit structures.

Within the PDL, the System Power Management (SysPm) API provides functions to change power modes as shown in Figure 1. The API can also register callback functions to execute a peripheral function before or after power mode transitions as shown in Figure 9.

PDL is available from two sources depending on the development platform you are using. The two versions of PDL share the same APIs but are not compatible with the other versions tool base.

- PSoC Creator Peripheral Driver Library from cypress.com. The PSoC Creator download and installation includes PDL so no additional user actions are required.
- ModusToolbox Peripheral Driver Library (psoc6pdl) from github.com. The ModusToolbox installation automatically installs PDL but requires a download during install directly from GitHub. Use with 3rd party development environments requires manual download and install.

2.6.2 Mode Transition Functions

Figure 1 shows firmware transitions for power modes. SysPm provides the default five transition functions for CPU sleep, CPU deep sleep, system hibernate, system LP, and system ULP.

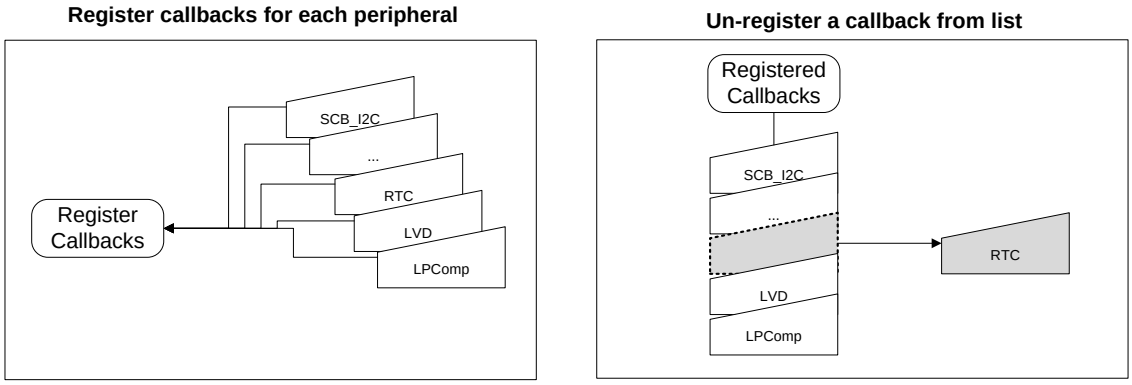
The power mode changing functions provide four different callback operations to execute a necessary action for each peripheral:

- `CY_SYS_PM_CHECK_READY`: Checks the ready state to transition to other mode. Exits without transition if it returns `CY_SYSPM_FAIL`.
- `CY_SYSPM_BEFORE_TRANSITION`: Callbacks execute and configure required actions before mode transition.
- `CY_SYSPM_AFTER_TRANSITION`: Callbacks execute after mode transition or configuration.
- `CY_SYS_CHECK_FAIL`: Callbacks execute only when `CY_SYSPM_CHECK_READY` fails. It executes the rollback action.

The SysPm driver provides three functions for callback: registration, de-registration, and execution. These functions not only help in power optimization, but also in preventing an abnormal peripheral state after mode transition. The PDL expects the user to register callbacks for each power mode, as shown in Figure 8. Most peripheral drivers have predefined callbacks associated with each power mode. You can choose to register the defined peripheral callback or can make a custom callback. The SysPm transition function executes the registered callbacks sequentially. The first registered function is executed first.

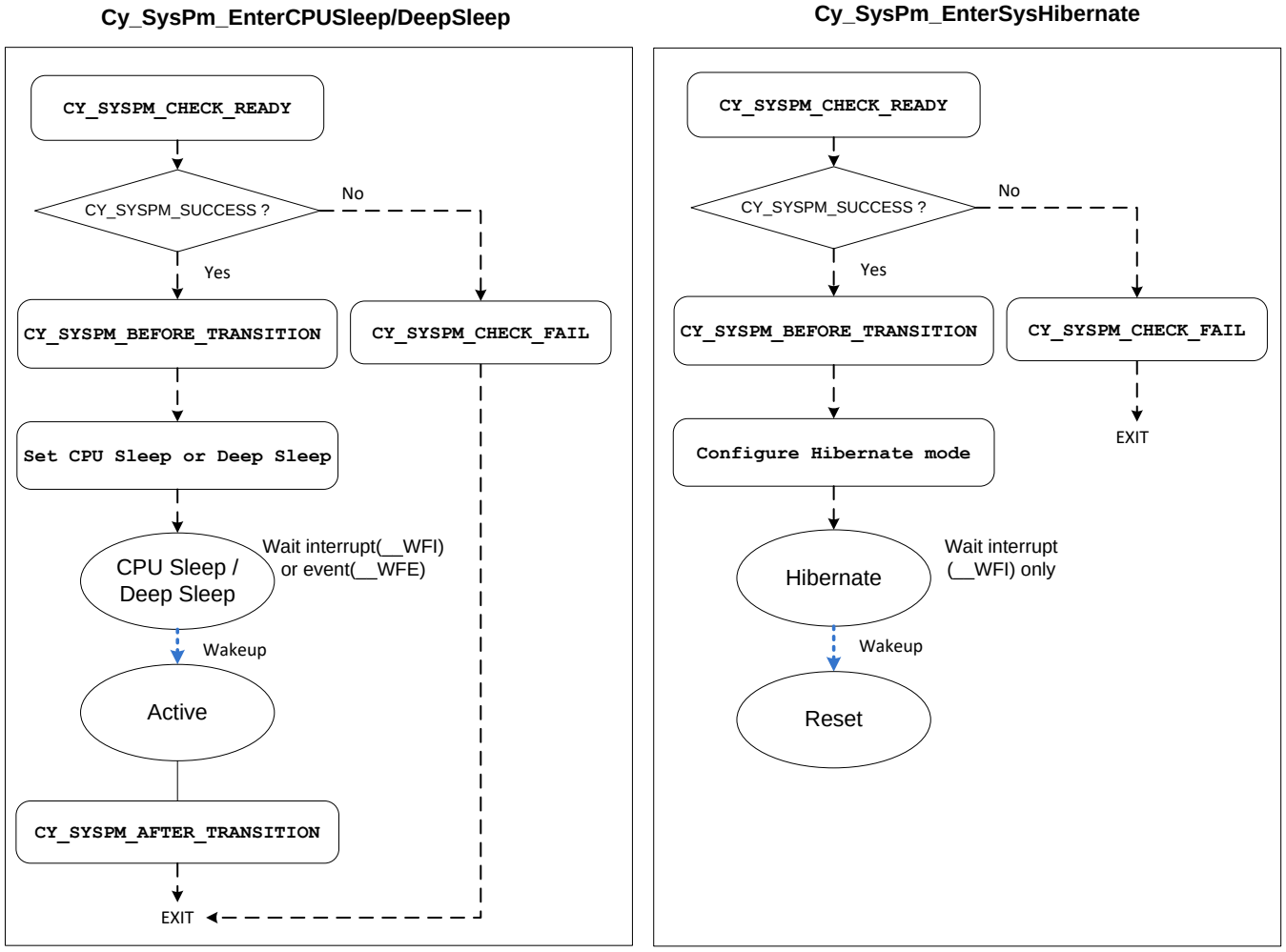
For more information on callback registration and implementation, see Appendix C, and D.1 of the code example CE219881 - PSoC 6 MCU Switching Between Power Modes, which is the mode transition example for CPU active, CPU sleep, system LP, system ULP, and system deep sleep.

Figure 8. Power Mode Callback Registration and Un-registration



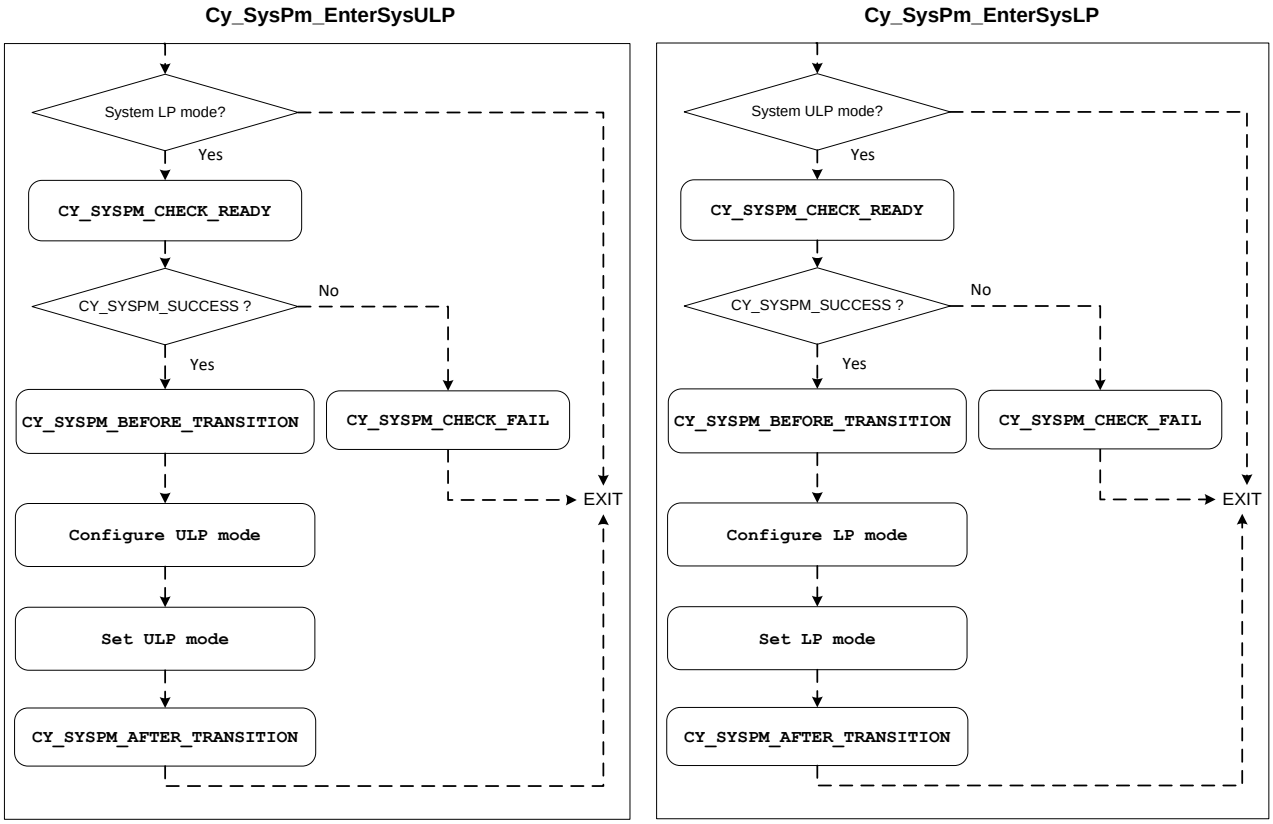
By calling the mode transition function, the device starts to transition with four callback operations. The CPU sleep and CPU deep sleep modes use Arm sleep instructions. Code execution stops and waits for an interrupt during the CPU sleep power mode. Figure 9 shows the CPU waiting for a wakeup source after calling the sleep instruction: `__WFI()` or `__WFE()`. After wakeup, the device automatically transitions to CPU active.

Figure 9. Sleep/ Deep Sleep/ Hibernate Mode Transition



In the system LP and system ULP modes, all system resources keep running. Entering system LP and ULP modes is done by configuring the power mode control register; the transition occurs without delay. SysPm PDL provides the associated driver functions, as shown in Figure 10 . For the best power efficiency, it is necessary to configure the core voltage regulator and the system clock. For more detailed information, see 3.1 Core Voltage Selection, 3.2 ULP Mode Clock and Peripheral Driver Library Document (**PSoC Creator** > **Help** > **Documentation** > **Peripheral Driver Library**).

Figure 10. Low-Power Mode Transition



3 PSoC 6 MCU Power Management Techniques

3.1 Core Voltage Selection

3.1.1 Linear Regulator and Buck Regulator

PSoC 6 MCU supports multiple on-chip regulators including low drop out (LDO) and single input multiple output (SIMO), or single input single output (SISO) buck to generate V_{CC} for core power, as listed in Table 3. The LDO can provide up to 300 mA in high-current mode (normal) and 25 mA in low-current mode. The buck regulator can provide up to 20 mA for one output and 30 mA combined for both outputs of the SIMO buck. The SIMO buck regulator provides better efficiency under normal load conditions. Once switched to the SIMO buck regulator, it is not possible to switch back to the LDO without resetting the device.

Table 3. Different Options of Core Voltage Regulators for Low-Power Profile

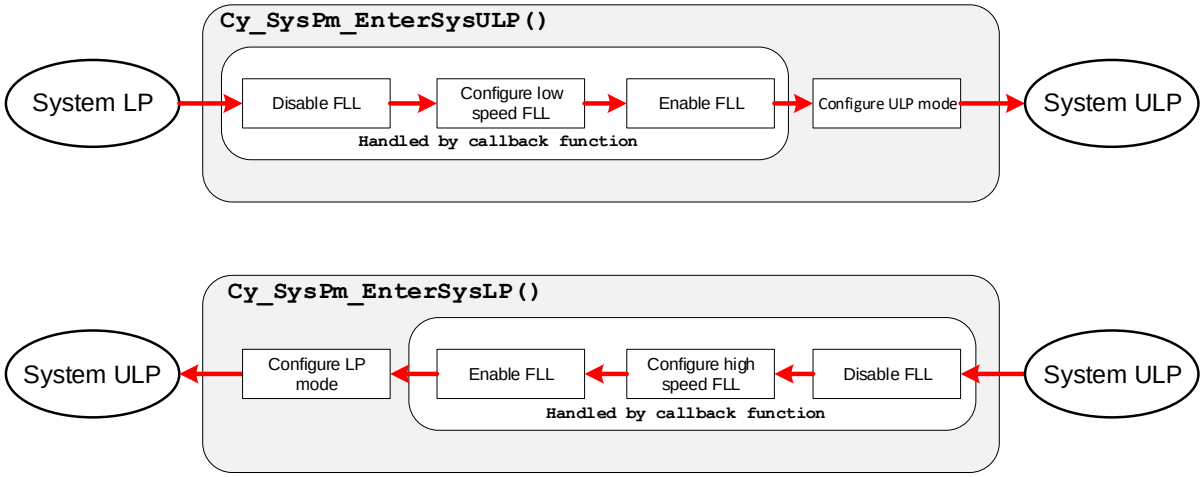
	Output	Max Load	Max Clock Frequency
LDO	0.9 V	25/300 mA (low-current mode/ normal mode)	50 MHz for Cortex-M4 (CM4) 25 MHz for Cortex-M0+ (CM0+)
	1.1 V	25/300 mA (low-current mode/ normal mode)	Allow maximum supportable clock frequency
Buck	0.9 V	20 mA	50 MHz for CM4 25 MHz for CM0+
	1.1 V	20 mA	Allow maximum supportable clock frequency

3.2 ULP Mode Clock

Transition to ULP mode can be done by configuring the power mode control register. There is a maximum clock speed limitation in ULP mode as described earlier, so the clock configuration should be adjusted based on the regulator output when entering or exiting ULP mode. PDL provides associated functions to configure the PWR_CTL register. For more information, see PSoC 6 MCU Registers Technical Reference Manual.

Figure 11 shows how to transition between LP and ULP modes using PDL functions with the registered callback function. Because of the ULP mode clock limitation, either the frequency-locked loop (FLL) clock speed or HFClk should be adjusted to a valid frequency before the mode transition. Changing the FLL frequency impacts the blocks that use FLL-derived clocks; therefore, all active peripherals should register their own callbacks to handle the changing frequency. CE219881 – PSoC 6 MCU Switching between Power Modes provides an example of clock adjustment using callbacks.

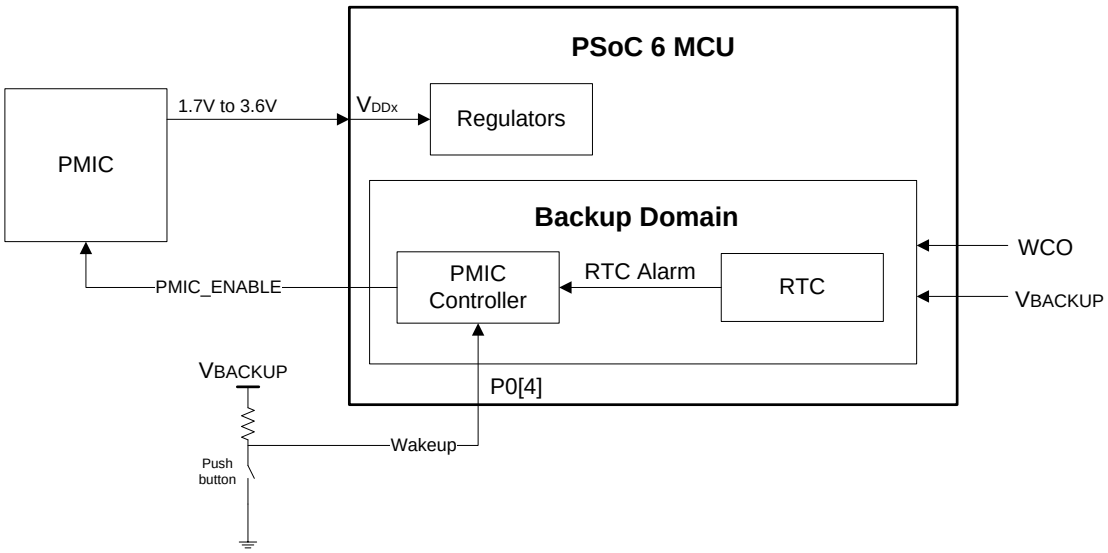
Figure 11. LP Mode Enter/Exit Transition



3.3 External PMIC Control

The PSoC 6 MCU backup power domain provides Power Management IC (PMIC) control functions. Figure 12 shows the external PMIC supplying system power (V_{DDD}). PSoC 6 MCU can be shut down completely by the PMIC, but a backup supply to control the PMIC can keep the backup domain alive. The backup domain includes an RTC, limited memory retention, and PMIC wakeup functions allowing the PMIC to restart the full PSoC 6 MCU device.

Figure 12. An External PMIC Control Block Diagram



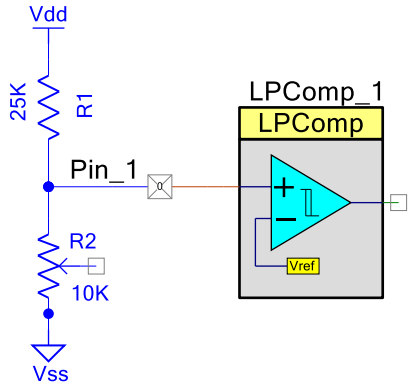
4 Other Power Saving Techniques

4.1 Use PSoC 6 MCU to Gate Current Paths

Your PCB may contain other components that draw power; PSoC 6 MCUs can be used to control the power through them by supplying the power with GPIO pins that can be turned ON and OFF in firmware. Note that the maximum pin source and sink capabilities listed in the datasheet must not be exceeded. If a higher current is required than the GPIO can directly supply, external power devices can be used, which are controlled by the GPIO.

A good example of this scenario is a low-power comparator (LPComp) application, as shown in Figure 13. In this case, the PSoC device compares the voltage on an analog pin, which changes as the potentiometer resistance changes.

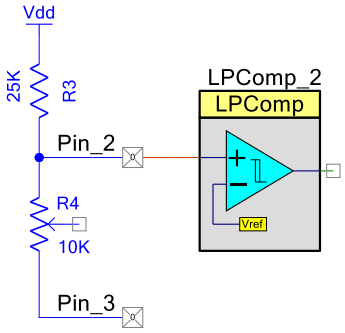
Figure 13. Typical LPComp Application



LPComp can be turned OFF when not in use, but external components will still consume power because the current path through the resistor and potentiometer remains. A simple solution with PSoC 6 MCU is to use a second pin as a switch to ground, as shown in Figure 14.

In this configuration, the current flow can be stopped by writing a '1' to Pin_3 and allowing the pin to float. This removes the current consumption by reducing the voltage differential across the two resistors to 0 V. Writing a '0' resumes the current flow. Only one pin and a few lines of code are required to implement this power-saving feature.

Figure 14. Using a GPIO as a Ground Switch



4.2 Disable Unused Blocks

You can save unnecessary current consumption by disabling unused blocks. The power saved depends on the block disabled.

4.3 Use DMA to Move Data

You can save power any time you offload a task from the CPU and either halt the CPU or let it do something else in parallel. The DMA engine can be used in both system LP and ULP modes to transfer data with no CPU use. The power saved is either the difference between CPU active and CPU stop power modes if the CPU can be halted, or lower CPU active current if the CPU can be clocked at a slower frequency and still get the same work done.

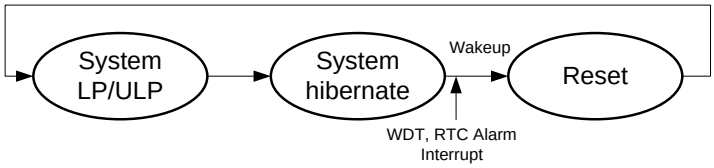
4.4 Periodic Wakeup Timers

A periodic wakeup from CPU sleep mode is the most common way to reduce power consumption. The average power consumption is determined by the ratio of CPU active period power consumption and CPU sleep period power consumption. To achieve the best result, the sleep period should be as long as possible and the active period should be as short as possible. An example of CPU sleep mode power savings from the CY8C61x6 datasheet is CM4 active = 1.7 mA (SIDF5) and CM4 sleep = 0.76 mA (SIDS7).

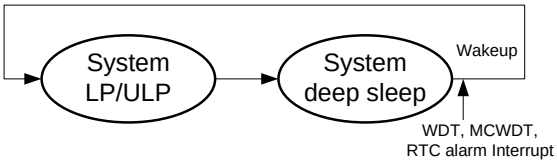
The WDT and multi-counter WDT (MCWDT) are effective periodic wakeup sources in system deep sleep and system hibernate modes. If your application needs longer, or more precise wakeup periods, an RTC alarm can be a good periodic wakeup source. See D.3 CE218542 - PSoC 6 MCU Custom Tick Timer Using RTC Alarm Interrupt for a code example using the RTC periodic timer for wakeup. An example of system deep sleep mode power savings from the CY8C61x6 datasheet is CM4 active = 1.7 mA (SIDF5) and system deep sleep = 7 µA (SIDDS1).

The following scenarios show how to change the mode states:

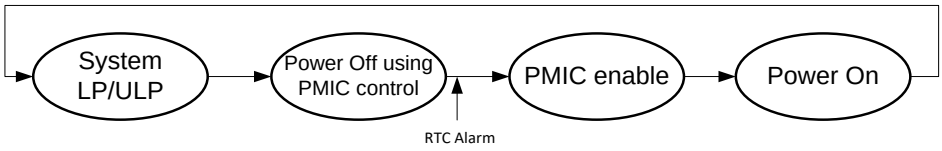
- System hibernate



- System deep sleep



- Power OFF with an external PMIC



4.5 Disabling PSoC CPUs

If the CM4 CPU is not used in the application, disable it by calling the `Cy_SysDisableCM4()` function from the CM0+ firmware. You can also set the Clk_Fast divider to 256 to minimize any fan-out leakage from this clock.

If the CM0+ CPU is not used in the application, place it into system deep sleep power mode and set the Clk_Slow divider to 256 to minimize any fan-out leakage from this clock. Note that Clk_Slow also affects how fast the DMA engine runs.

4.6 Splitting Tasks Between the CPUs

There are several reasons to choose either CM4 or CM0+ to do certain tasks. For example:

1. Tasks that require floating point or DSP operations are more efficiently run on CM4.
2. Tasks that are time-critical requiring the maximum clock speed (150 MHz) should run only on CM4.
3. If deterministic timing is required by a single task independent of all other tasks, it is often beneficial to move that task to its own CPU simplifying timing. Usually the CM0+ CPU is used for this purpose.
4. When maximizing the system idle state time as discussed in Section 4.7 Clocks, it may be beneficial to spread tasks between CM0+ and CM4 so that the lower-power idle state can be entered faster due to simultaneous task processing.

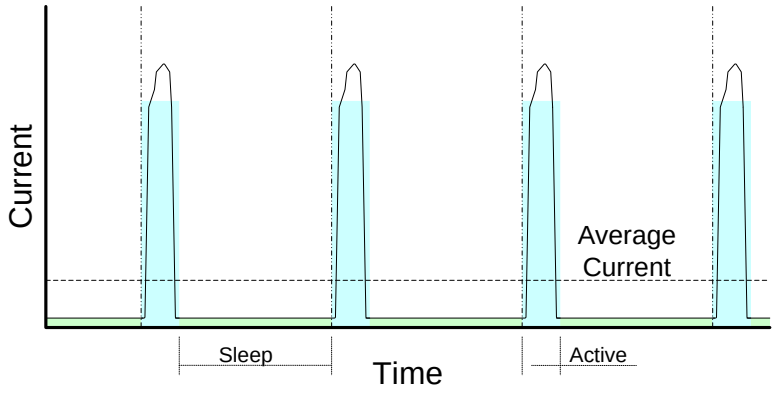
5. If single CPU processing must be spread out over a long period due to real-time hardware interactions that limit time spent in lower-power states, CM0+ will often be lower-power than CM4 over the same active time interval.
6. If processing power is required that exceeds what one CPU can provide, the tasks should be spread between both cores. Because each core has its own bus controller and can access separate memory blocks, there are minimal wait states due to bus or memory contention. The performance decrease due to contention is typically less than 2% while the performance increase from the second core is greater than 50% (assuming CM0+ added to CM4).
7. In applications that stay in active mode and can achieve the required processing power on a single core, there is most often no power benefit to sharing the required tasks between cores.

4.7 Clocks

In some cases, running CPU clocks faster can result in a lower average current consumption. For example, consider a design that takes a reading from a sensor once every second, performs several calculations, and then transmits the results to another device.

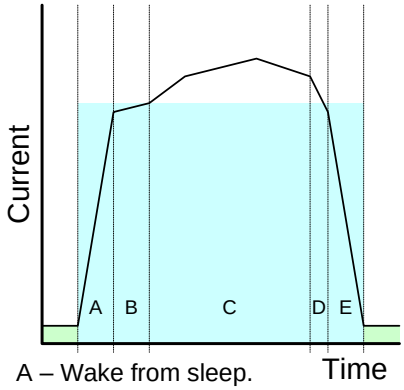
You can use CPU sleep or system deep sleep mode to reduce power when the PSoC device is idle, but the average current consumption can be higher because of the additional time spent in CPU active mode. Figure 15 is a representation of the current consumption of this example with the system clocks set at 8 MHz.

Figure 15. Example Current Profile with 8-MHz Clocks



Depending on the tasks or calculations that are being performed when the PSoC device is awake, it may be possible to complete them sooner by running the CPU clocks faster. This can reduce the average current consumption because the PSoC device is in CPU active mode for less time and in CPU sleep or system deep sleep longer. Figure 16 is a representation of CPU active mode timing, broken up into tasks.

Figure 16. Analysis of Tasks in CPU Active Mode at 8 MHz

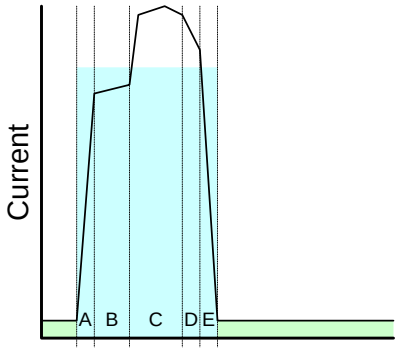


A – Wake from sleep.
B – Read sensor data.
C – Manipulate data.
D – Transmit result.
E – Go back to sleep.

The time required for some tasks does not change even if the system clock frequency increases. Sensor reading and data transmitting fall into this category. Other tasks like data processing, require less time if the CPU operates at a faster frequency.

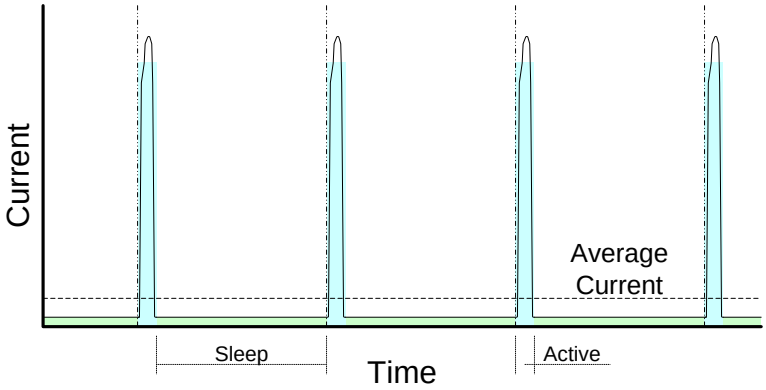
At some point, the benefit of a shorter active time is overcome by the energy required to drive the clocks at a higher rate. Assume that the optimal speed is 48 MHz, as Figure 17 shows. With a 48-MHz clock, the time spent in active mode is about half the time spent with an 8-MHz clock. Figure 18 shows that the peak current consumption is greater when the clocks are faster, but the overall average is lower due to the longer time spent in CPU sleep or system deep sleep. On PSoC 6 MCU devices, it is generally better to run the CPU as fast as possible to complete processing and spend more time in the low-power mode.

Figure 17. Analysis of Tasks in Active Mode at 48 MHz



A – Wake from sleep.
B – Read sensor data.
C – Manipulate data.
D – Transmit result.
E – Go back to sleep.

Figure 18. Example Current Profile with 48-MHz Clocks



4.8 GPIOs

GPIOs can continue to drive external circuitry while the PSoC device is in any low-power mode. This is helpful when you need to hold external logic at a fixed level, but it can lead to wasted power if the pins needlessly source or sink current. The specific power savings of this technique depend on the circuit attached to the specific GPIO pin.

You should analyze your design and determine the best state for your GPIOs during low-power operation. If holding a digital output pin at logic '1' or '0' results in the lowest current, match that digital level using `Cy_GPIO_Write()`.

```
/* Set MyPin to '0' for low power. */
Cy_GPIO_Write(MYPIN_0_PORT, MYPIN_0_NUM, 0u);
```

Configure all unused GPIOs to Analog High-Z unless there is a specific reason to use a different drive mode. The High-Z drive mode results in the lowest current for a GPIO pin. A pin's drive mode may be set using the `Cy_GPIO_SetDrivemode()` function.

```
/* Set MyPin to Alg HI-Z for low power. */
Cy_GPIO_SetDrivemode(MYPIN_0_PORT, MYPIN_0_NUM, CY_GPIO_DM_HIGHZ);
```

The flexibility of PSoC makes it easy to manage GPIO drive modes to prevent unwanted current leakage. In system hibernate mode, the GPIO drive modes and data registers are automatically "frozen." They must be reconfigured to a known state before being "unfrozen" to avoid GPIO output transitions after a wakeup reset and to allow their states to be changed at runtime. Call `Cy_SysPm_IoUnfreeze()` after configuring the GPIO pins and setting their output drive state. See Section D.2 CE218129 - PSoC 6 MCU Wakeup from Hibernate Using a Low-Power Comparator for a code example on hibernate and I/O control.

4.9 SRAM

PSoC 6 MCU devices allow powering off individual SRAM banks or pages within a bank. The size of pages within a bank depends on the specific device and bank as detailed in the device datasheet. Specific devices will have one or more SRAM banks. Most devices have one bank with a smaller page size (typically 32 KB) for fine-grained control of the amount of SRAM enabled. Any unused page can be disabled by writing to the CPUSS power control registers. This technique is most useful in system deep sleep mode where retaining 64K SRAM = 7 µA (SIDDS1) while retaining 256K SRAM = 9 µA (SIDDS2) as listed in the CY8C61x6 datasheet.

```
/* Power off all except the first two pages of RAM0 */
for (uint32_t i = 2; i < NUM_RAM_PAGES; i++)
{
    CPUSS->RAM0_PWR_MACRO_CTL[i] = 0x05FA0000;
}

/* Additional SRAM banks */
CPUSS->RAM1_PWR_CTL = 0x05FA0000;
CPUSS->RAM2_PWR_CTL = 0x05FA0000;
```

Consult the Register TRM of the device for all power control registers.

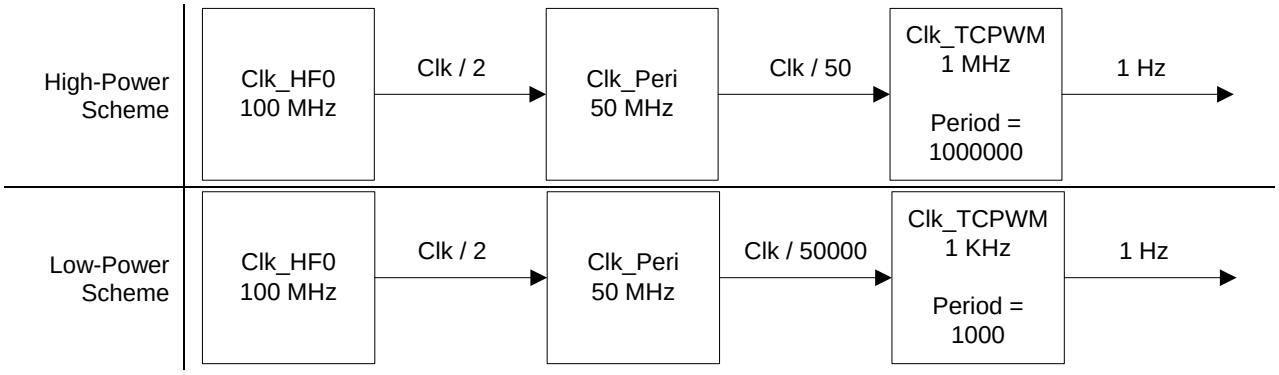
As an advanced use case, modification of the project's linker script can be used with SRAM power down to optimize the total amount of SRAM required or allow custom data placement in support of dynamic SRAM powering.

4.10 TCPWM

When using a counter, timer or PWM, you should configure the clock sourcing the channel as low as possible while still meeting your frequency and accuracy requirements. For example, if you need to generate a 1-second interrupt with a timer, it is better to use a clock frequency of 1 kHz with the period equaling 1,000 counts than a clock frequency of 1 MHz with a period equal to 1,000,000 counts. The power savings from reducing the TCPWM clock is mostly linear based on clock frequency. TCPWM operating current at 100 MHz = 540 µA (SID.TCPWM.2B) while at 8 MHz = 70 µA (SID.TCPWM.1) per the CY8C61x6 datasheet.

The same idea applies when using a PWM to dim an LED. Use the minimum clock frequency possible that does not let the human eye perceive that the LED is blinking. 60 Hz is a good frequency for most applications. Figure 19 shows a comparison between the clock settings for the TCPWM block.

Figure 19. TCPWM Clock Settings Comparison



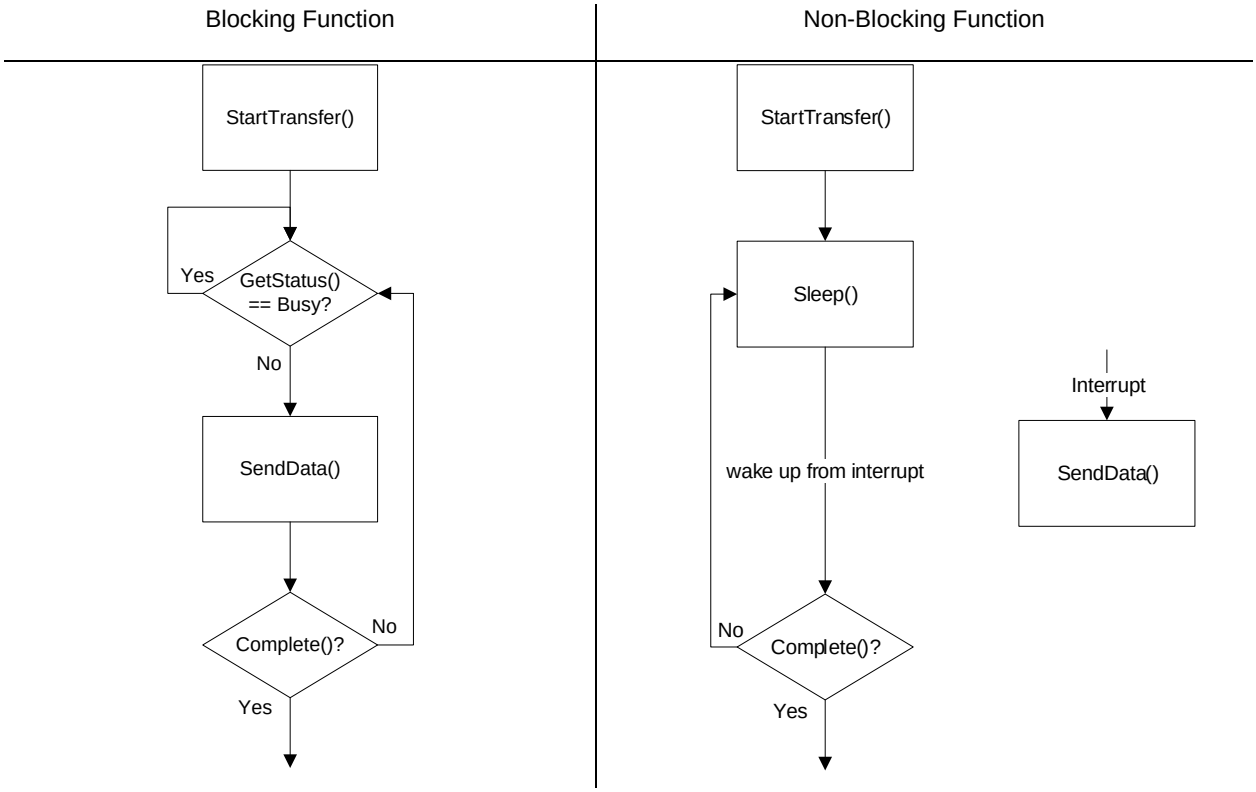
The TCPWM block has a Clock Prescaler feature. For minimum power consumption, maximize the peripheral clock divider first before using the TCPWM Clock Prescaler.

If the TCPWM block uses any pin connections, set the pins to Analog High-Z when the block is disabled. If the PSoC device supports Digital System Interconnect (DSI), avoid using DSI by choosing the TCPWM channels with direct connection to the desired pin assignment. If the PSoC device supports UDBs (Universal Digital Block), use all fixed-function TCPWM channels first instead of UDB-based TCPWMs.

4.11 SCB

Avoid using blocking functions when sending or receiving data. Use interrupt-based events or an RTOS to transfer the data while yielding the CPU to other tasks. The idea behind this strategy is to keep the CPU in a sleep state longer, instead of polling the status of the transmission. Figure 20 shows an example of blocking and non-blocking functions.

Figure 20. Example of Blocking and Non-Blocking Functions



Instead of using the SCB interrupt to access its FIFO in firmware, use DMAs controlled by the FIFO level to reduce the amount of CPU cycles required in the application; therefore, the CPU can stay in a sleep state longer or execute other tasks.

If the SCB block uses any pin connections, set the pins to Analog High-Z when the block is disabled. If the PSoC device supports DSI, avoid using it by choosing SCB channels with direct connections to the desired pin assignment. If the PSoC device supports UDBs, use all fixed-function SCB channels before using UDB-based communication interfaces.

Operating at the lowest data rate the system can support also helps in reducing power because the SCB's power increases linearly with clock frequency. For example running I2C at 1Mbps = 180 µA (SID152) while running at 100 kbps = 30 µA (SID149) as shown in the CY8C61x6 datasheet.

4.12 Audio Subsystem

The I2S and PDM/PCM blocks are generally sourced by a high-frequency clock generated from the PLL. For high-accuracy frame rates, an ECO sources the PLL. When picking a frequency for the ECO, consider what the desired audio sample rates are. If you need to support all standard audio sample rates (8/16/22.05/32/44.1/48 kHz), the minimum ECO frequency you can use is 17.2032 MHz. If you only need the sample rates 8/16/48 kHz, configure the PLL for 12.288 MHz. If you only need the sample rates 22.05/44.1 kHz, configure the PLL for 22.5792 MHz.

If the Audio Subsystem is not in use, not only disable the I2S and/or PDM/PCM blocks, but also the PLL and ECO, which are the main reasons for high current consumption. If the accuracy of the frame rates is not important for the application, do not use the ECO and/or PLL. For example, if your application implements a sound detector and can tolerate a higher error, use the FLL as the source for the Audio Subsystem. Once a sound is detected and the system needs to be fully operational, enable the PLL and ECO to sample at a higher accuracy.

Instead of using Audio Subsystem interrupts to access its FIFO in firmware, use DMAs controlled by the FIFO level triggers to reduce the number of CPU cycles required in the application so that the CPU can stay in a sleep state longer or execute other tasks.

4.13 USB

When there is no activity on the USB bus, you can suspend the USB block to consume less current. Refer to code example CE223305 – PSoC 6 MCU: USB Suspend and Resume on how to set up the device to enter and exit low-power states.

4.14 Low-Power Comparator

If the Low-Power Comparator requires pin connections and system deep sleep operation, always choose the dedicated pin connections to the comparator. This allows the comparator to work in system deep sleep and avoid using the global analog muxes, which consume additional current to stay active.

There are two ways to verify the status of the comparator: Call the `Cy_LPComp_GetCompare()` function, or set up an interrupt. In most cases, it is preferred to use the interrupt method, reducing CPU usage.

The comparator supports three operation modes that affect its speed and power:

1. **Ultra-low-power/Slow:** I_{CMP1} = 0.85 µA (SID259) – Slows the response time to 20 µs (SID92). Use this mode when in system deep sleep or hibernate only.
2. **Low-power/Low:** I_{CMP2} = 10 µA (SID248) – Preferred mode when system deep sleep or hibernate are not required and the slower response time of 1 µs (SID258) is acceptable. Increases input offset voltage to 25 mV (SID85A).
3. **Normal Power/Fast:** I_{CMP1} = 150 µA (SID89) - Use it when the fastest response time of 100 ns (SID91) and/or low offset voltage of 10 mV (SID84) are required.

Another option to reduce the amount of current consumed by the comparator is to disable hysteresis. Hysteresis can be disabled in the ModusToolbox configurator, PSoC Creator customizer, or PDL configuration structure.

4.15 SAR ADC

When selecting the pins to be connected to the SAR ADC, preferably choose the dedicated port connected to the SAR MUX first. Only after running out of pins in that port, choose pins from other ports. By only using the dedicated port, there is no need to use the global analog muxes, which consume extra current. You can also achieve higher sample rates with the dedicated port due to lower input capacitance.

If the full-rated accuracy of ADC results is not required, use a lower resolution and do not use averaging, which reduces the number of ADC clocks required for the same sample rate.

If the maximum sample rate is not required, consider using the single-shot mode instead of continuous mode. This avoids the SAR ADC operating all the time. In single shot mode, The ADC samples only when triggered by software or hardware, depending on the application.

4.16 Voltage DAC

If the DAC voltage output needs to be periodically changed with pre-determined values, use a DMA to update the voltage value. This avoids using CPU cycles to write to DAC registers.

If the external device that requires the DAC output voltage is used only periodically, keep the DAC disabled when its output is not required.

When the DAC output voltage must be provided while in system deep sleep, use a Sample and Hold strategy to maintain the voltage output while the Voltage DAC is disabled in system deep sleep. Refer to the code example CE220925 – PSoC 6 MCU VDAC Sample and Hold for more details. The DAC operating current = 125 µA (SID100D) while the DAC current stopped = 1 µA (SID101D) per the CY8C61x6 datasheet.

4.17 Opamp

The opamp supports three different operating modes – Low, Medium, and High. These operation modes are affected by the system power modes – system LP/ULP and deep sleep. In system deep sleep, there are two additional modes to choose from. Table 4 shows how to choose the appropriate operation mode.

Table 4. Opamp Operational Mode Conditions

Specification	High-Power Operation Mode		Medium-Power Operation Mode		Low-Power Operation Mode	
	LP / ULP	Deep sleep	LP / ULP	Deep sleep	LP / ULP	Deep sleep
Minimum Gain-Bandwidth	6 MHz	M1: 4 MHz M2: 0.5 MHz	4 MHz	M1: 2 MHz M2: 0.2 MHz	1 MHz	M1: 0.5 MHz M2: 0.1 MHz
Maximum Output Current (no load)	1500 µA	M1: 1500 µA M2: 120 µA	600 µA	M1: 600 µA M2: 60 µA	350 µA	M1: 350 µA M2: 15 µA
Response time	150 ns		400 ns		2000 ns	

M1: Mode 1 has higher GBW and highest current.
M2: Mode 2 has lower GBW and lowest current. When selecting the pins to be connected to the opamp, preferably choose the dedicated pins connected to the opamp. By using only the dedicated pins, there is no need to use the global analog muxes, which consume extra current.

5 Power Measurement

5.1 Measuring Current with a DMM

When using a digital multi meter (DMM) to measure device current, it is important to know the value of the shunt resistor in the DMM. DMMs have one or more (shunt) resistors between the current inputs. These resistors can range from less than an ohm to more than 10 kΩ. There is no standard value for the shunt resistor between brands or even models from the same vendor. It is important to review your meter's manual and learn the value of the shunt resistor because there will always be a voltage drop across this shunt. This means that the PSoC device will not see the same voltage you think you are supplying. If the shunt resistor in your meter is 1 Ω or less, you will see only a few millivolts of drop when measuring PSoC current, which can be ignored. If the shunt resistor is 1 kΩ, which some vendors use for low-current measurements, a 1-mA current will result in a drop of 1 V! High shunt resistor values can cause the device to reset due to low voltage at higher currents. Also, when changing ranges, be careful that the DMM does not do a break-before-make, or the power will be cycled and your project will be reset.

For extremely low currents in deep sleep, hibernate, and stop modes, a good technique is to use a zero or low-resistance shunt until the device enters low-power mode. After entering low-power mode, the code should keep the device in that mode and switch to a high-resistance shunt for current measurement.

As an alternative to relying on the DMM shunt, most PSoC 6 MCU kits include a place for a shunt resistor or current measurement header. See the respective kits user guide to determine specific current measurement features. These can be replaced or connected to a small resistor allowing measurement of the voltage across the shunt with a voltmeter. You can then easily determine the current. A shunt between 1 Ω and 100 Ω should work well for most applications.

5.2 Approximating Power Consumption

The device datasheet provides information to estimate PSoC 6 MCU power consumption for project-specific user configurations. To simplify this process, a spreadsheet has been provided that includes typical power requirements for a wide range of internal components and modes. This spreadsheet, *PSoC_6_Power_Estimator.xlsx*, is located on the AN219528 page. Because every project is different, the power calculation provided by this spreadsheet is only an estimate of typical values but should be close enough to provide feedback before your design is complete.

There are several tabs in the spreadsheet; make sure that you read the “Instructions” tab before entering your data. Five “Config” tabs allow configuration of different hardware settings and power modes that the device can transition through at run time. The “Summary” tab displays the current for each configuration as well as the peak and average current across all modes based on the time spent in each configuration. An optional battery life estimation section on the “Summary” tab estimates battery life based on the average current.

6 Power Supply Protection System

6.1 Hardware Control

6.1.1 Brownout Detect (BOD)

Brownout detect (BOD) can reset the system before the logic state is lost when V_{DDD} and V_{CCD} power are lost. The brownout system guarantees a reset before V_{DDD} reaches the minimum system operating voltage, and works for all logic, SRAM, and flash. BOD is controlled by hardware; there is no configurable register.

6.1.2 Low-Voltage Detect (LVD)

Low-voltage detect (LVD) is similar to BOD but its threshold voltage is configurable. LVD generates an interrupt when V_{DDD} falls under a configured trip voltage. This interrupt allows the application time to handle important data before the BOD reset is triggered. LVD provides 15 selectable trip voltages. LVD is not available in system deep sleep and hibernate modes.

6.1.3 Overvoltage Detect (OVD)

Overvoltage detect (OVD) is the opposite of BOD. These circuits generate a reset when unsafe overvoltage supply conditions on V_{DDD} and V_{CCD} are detected. No firmware control is required. OVD helps to protect the system from high-voltage damage.

7 Summary

Many power managing options can be used in PSoC 6 MCU. By following proper methods, you can optimize your design and ensure that the power modes and features of PSoC 6 MCU give the best options for the lowest power consumption without degrading the required performance of battery powered devices.

8 Related Documents

- AN215656 – PSoC 6 MCU Dual-CPU System Design
- AN227910 – PSoC 6 Low-Power System Design with CYW43012 and PSoC 6 MCU
- CE219881 – PSoC 6 MCU Switching between Power Modes
- CE226306 – PSoC 6 MCU Power Measurements
- CE218542 – PSoC 6 MCU Custom Tick Timer Using RTC Alarm Interrupt
- CE218129 – PSoC 6 MCU Wakeup from Hibernate Using a Low-Power Comparator

Appendix A. Power Modes Summary

A.1 Power Modes and Wakeup Source

Table 5. Power Modes and Wakeup Source

System Power Mode	MCU Power Mode	Description	Entry Conditions	Wakeup Sources	Wakeup Action
LP	Active	Primary mode of operation. 1.1-V core voltage. All peripherals are available (programmable). Clocks at their maximum frequencies.	• Reset from external reset, brownout, power on reset system and hibernate mode. • Manual register write from system ULP mode. • Wakeup from CPU sleep or CPU deep sleep while in system LP mode. • Wakeup from system deep sleep after entered from LP mode.	Not applicable	N/A
	Sleep	1.1-V core voltage. One or more CPUs in sleep mode (execution halted). All peripherals are available (programmable). Clocks at their maximum frequencies.	In system LP mode, CPU executes WFI/WFE instruction with deep sleep disabled	Any interrupt to CPU	Interrupt
	Deep sleep	1.1-V core voltage. One CPU in deep sleep mode (execution halted). Other CPU in active or sleep mode. All peripherals are available (programmable). Clocks at their maximum frequencies.	In system LP mode, CPU executes WFI/WFE instruction with deep sleep enabled	Any interrupt to CPU	Interrupt
ULP	Active	0.9-V core voltage. All peripherals are available (programmable). Clock frequencies are limited.	Manual register write from system LP mode. Wakeup from CPU sleep or CPU deep sleep while in system ULP mode. Wakeup from system deep sleep after entered from ULP mode.	Not applicable	N/A
	Sleep	0.9-V core voltage. One or more CPUs in sleep mode (execution halted). All peripherals are available (programmable). Clock frequencies are limited.	In system ULP mode, CPU executes WFI/WFE instruction with deep sleep disabled	Any interrupt to CPU	Interrupt
	Deep sleep	0.9-V core voltage. One CPU in deep sleep mode (execution halted). Other CPU in active or sleep mode. All peripherals are available (programmable). Clock frequencies are limited.	In system ULP mode, CPU executes WFI/WFE instruction with deep sleep enabled	Any interrupt to CPU	Interrupt
Deep sleep	Deep sleep	All high-frequency clocks and peripherals are turned off. Low-frequency clock (32 kHz) and low-power analog and digital peripherals are available for operation and as wakeup sources. SRAM is retained (programmable).	Both CPUs simultaneously in CPU deep sleep mode	GPIO interrupt, low-power comparator, SCB, CTBm, watchdog timer, and RTC alarms	Interrupt
Hibernate	N/A	GPIO states are frozen. All peripherals and clocks in the device are completely turned OFF except low-power comparator and backup domain. Wakeup is possible through WAKEUP pins, XRES, low-power comparator (programmable), and RTC alarms (programmable). Device resets on wakeup.	Manual register write from LP or ULP modes	WAKEUP pin, low-power comparator, watchdog timer[b], and RTC[a] alarms	Reset

a. RTC (along with WCO) is a part of the backup domain and is available irrespective of the device power mode. RTC alarms are capable of waking up the device from any power mode.

b. Watchdog timer is capable of generating a hibernate wakeup.

Appendix B. Subsystem Availability

B.1 Resources Available in Different Power Modes

Table 6 shows information for the resource availability in different power modes.

Table 6. Resources Available in Different Power Modes

Component	System Power Modes							
	LP		ULP		Deep sleep	Hibernate	XRES	Power off with backup
	CPU active	CPU sleep/ deep sleep	CPU active	CPU sleep/ deep sleep				
Core functions								
CPU	On	Sleep	On	Sleep	Retention	Off	Off	Off
SRAM	On	On	On	On	Retention	Off	Off	Off
Flash	Read/Write	Read/Write	Read only	Read only	Off	Off	Off	Off
High-Speed Clock (IMO, ECO, PLL, FLL)	On	On	On	On	Off	Off	Off	Off
LVD	On	On	On	On	Off	Off	Off	Off
ILO	On	On	On	On	On	On	Off	Off
Peripherals								
SMIF	On	On	On	On	Retention	Off	Off	Off
UDB	On	On	On	On	Off	Off	Off	Off
SAR ADC	On	On	On	On	Off	Off	Off	Off
CTBm	On	On	On	On	On (lower GBW)[1]	Off	Off	Off
LPCMP	On	On	On	On	On[1]	On[2]	Off	Off
TCPWM	On	On	On	On	Off	Off	Off	Off
CSD	On	On	On	On	Retention	Off	Off	Off
BLE	On	On	On	On	Retention	Off	Off	Off
LCD	On	On	On	On	On	Off	Off	Off
SCB	On	On	On	On	Retention (I^2C/SPI wakeup available)[3]	Off	Off	Off
GPIO	On	On	On	On	On	Freeze	Off	Off
Watchdog timer	On	On	On	On	On	On	Off	Off
Multi-Counter WDT	On	On	On	On	On	Off	Off	Off
Resets								
XRES	On	On	On	On	On	On	On	Off
POR	On	On	On	On	On	On	Off	Off
BOD	On	On	On	On	On	Off	Off	Off
Watchdog reset	On	On	On	On	On	On[4]	Off	Off
Backup domain								
WCO, RTC, alarms	On	On	On	On	On	On	On	On

1. Low-power comparator and CTBm may be optionally enabled in system deep sleep mode to generate wakeup.
2. Low-power comparator may be optionally enabled in hibernate mode to generate wakeup.
3. Only one SCB with deep sleep support is available in system deep sleep power mode; other SCBs are not available in system deep sleep power mode.
4. Watchdog interrupt can generate a hibernate wakeup. See the “Watchdog Timer” chapter of the TRM for details.

Appendix C. Callback Function Examples

C.1 Register Callback Functions

```
cy_stc_syspm_callback_params_t myParams;
cy_stc_syspm_callback_t myAppSleep =

{
       &Application_Callback,              /* Callback function */
       CY_SYSPM_SLEEP,                     /* Select Power Mode */
       (CY_SYSPM_SKIP_CHECK_READY |
        CY_SYSPM_SKIP_CHECK_FAIL),         /* Skip CHECK_READY and CHECK FAIL */
       &myParams,                          /* Operation, contexts */
       NULL,                               /* Previous list callback */
       NULL                                /* Next list callback */

};

cy_stc_syspm_callback_t myAppHibernate =
{
       &Application_Callback,              /* Callback function */
       CY_SYSPM_HIBERNATE,                 /* Select Power Mode */
       0U,                                 /* Skip mode, no skip */
       &myParams,                         /* Operation, contexts */
       NULL,                               /* Previous list callback */
       NULL                                /* Next list callback */

};

/* Register Callback functions for each power mode */
Cy_SysPm_RegisterCallback(&myAppSleep);
Cy_SysPm_RegisterCallback(&myAppHibernate);
```

C.2 Implement Custom Callback Functions

```
cy_en_syspm_status_t Application_Callback(
cy_stc_syspm_callback_params_t *callbackParams)
{
    cy_en_syspm_status_t retVal = CY_SYSPM_SUCCESS;

     switch(callbackParams->mode)
       {
         case CY_SYSPM_CHECK_READY:
          {
            if(Check_HW())
             {
                    /* Hardware is ready */
             }
           else
             {
                    retVal = CY_SYSPM_FAIL;
             }
          }
        break;

         case CY_SYSPM_CHECK_FAIL:
          {
          /* Rollback any configuration during CHECK_READY */
          Rollback_HW();
          retVal = CY_SYSPM_SUCCESS;
          }
        break;
```

```
                case CY_SYSPM_BEFORE_TRANSITION:
                {
                     /* configure HW for new mode before transition */
                     ConfigureHW_BeforeMode();
                     retVal = CY_SYSPM_SUCCESS;
                }
                break;

                case CY_SYSPM_AFTER_TRANSITION:
                 {
                     /* configure HW after mode transition */
                     ConfigureHW_AfterMode();
                     retVal = CY_SYSPM_SUCCESS;
                }
                break;

                default:

                    break;
            }

    return (retVal);
}
```

Appendix D. Code Examples

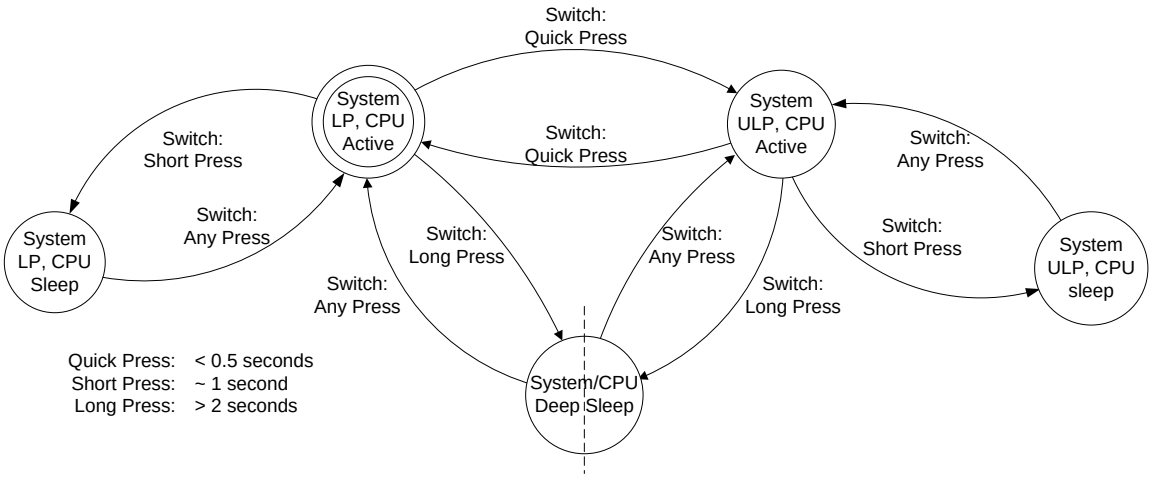
The following code example is included with this application note to demonstrate PSoC 6 MCU power modes and power reduction techniques.

D.1 CE219881 - PSoC 6 MCU Switching Between Power Modes

This code example shows how to enter and exit system LP and ULP power modes and transition the CPU from CPU active to sleep or deep sleep. Once in either mode, the example also shows how to wake up and return to one of the system LP or ULP modes and CPU active mode.

The project uses a switch to transition among power modes and shows different LED colors to indicate the current power mode. Figure 21 shows the state machine implemented in the firmware to execute the transitions. For more information, see CE219881 - PSoC 6 MCU Switching between Power Modes.

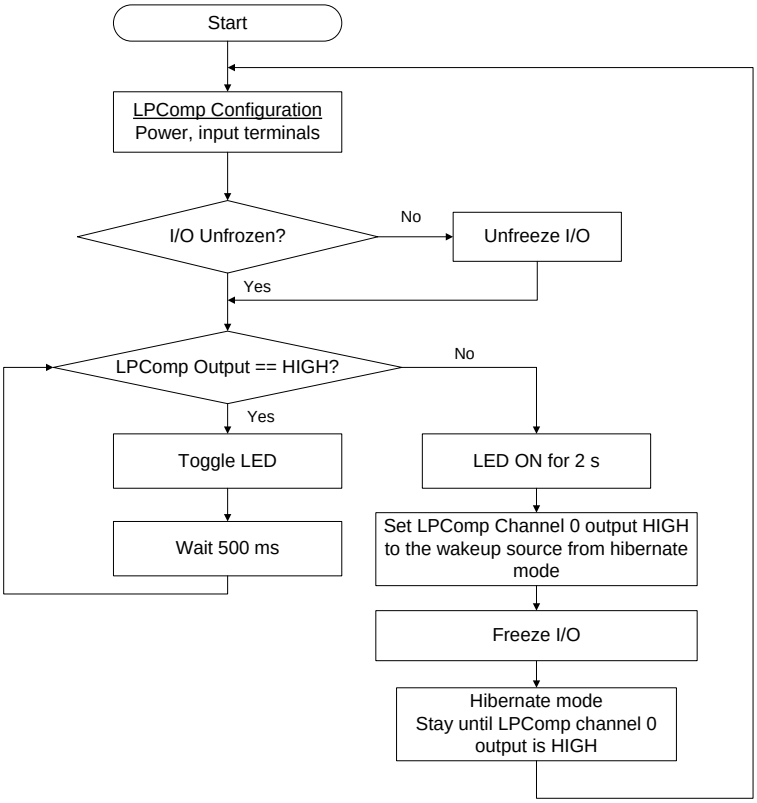
Figure 21. Power Mode State Machine



D.2 CE218129 - PSoC 6 MCU Wakeup from Hibernate Using a Low-Power Comparator

This code example demonstrates how to set the Component options for the LPComp internal reference voltage and how to set the external input from a GPIO using the LPComp driver. The project is a good example of the system hibernate power mode transition. It teaches you how to handle the GPIOs before and after system hibernate, and it shows how to register the wakeup source for hibernate. Figure 22 shows the basic flow of the project. For more information, see CE218129 - PSoC 6 MCU Wakeup from Hibernate Using a Low-Power Comparator.

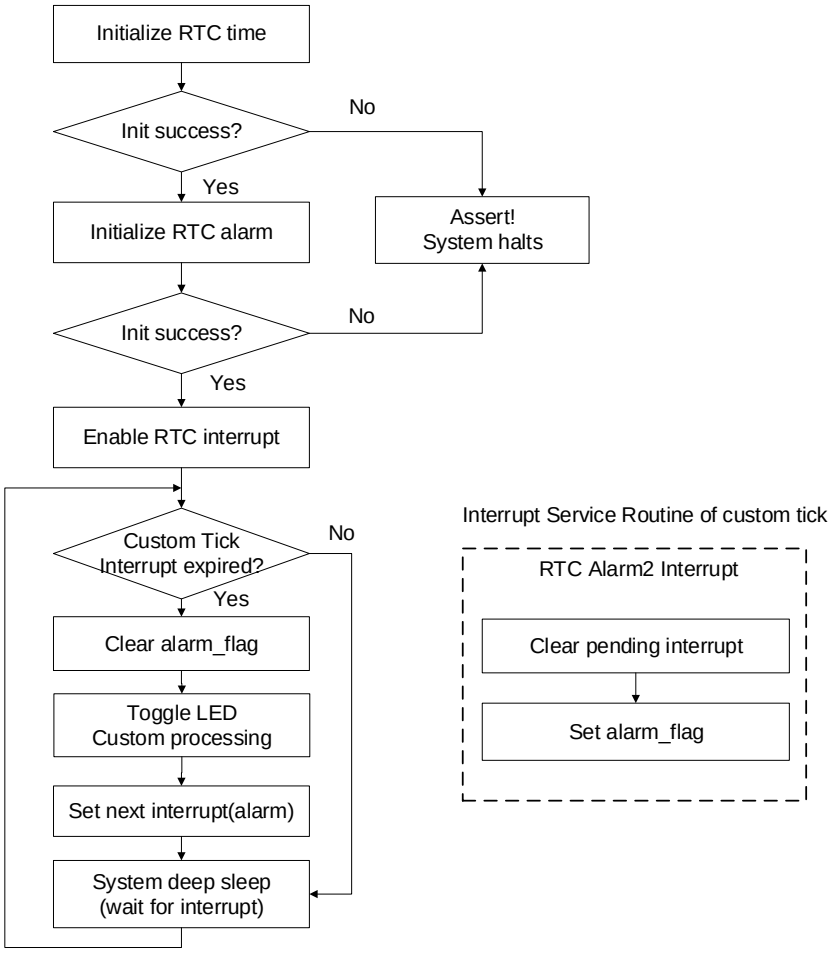
Figure 22. Wakeup from system hibernate Mode Using LPComp Input



D.3 CE218542 - PSoC 6 MCU Custom Tick Timer Using RTC Alarm Interrupt

This code example demonstrates how to configure the RTC registers for a periodic alarm interrupt using the PDL RTC driver API. The project uses system LP and system deep sleep modes to reduce power consumption. A GPIO output is included to toggle the LED to show the period of the interrupt. For more information, see CE218542 - PSoC 6 MCU Custom Tick Timer Using RTC Alarm Interrupt.

Figure 23. RTC Periodic Wakeup Timer using Alarm Interrupt



D.4 CE226306 - PSoC 6 MCU Power Measurements

This example shows how to configure PSoC 6 MCU devices to run the clock frequencies and system modes specified in the Device Level Specifications table in PSoC 6 MCU datasheets. After building and programming the application into PSoC 6 MCU devices, you can measure the current the PSoC 6 MCU device consumes and compare it against the values specified in the datasheet. You can select the power configuration by changing a #define in firmware.

Table 7. Configuration Options

Configuration	Options	Description

System Mode	SYSTEM_LP
	SYSTEM_ULP
		Defines the system mode the firmware enters – system low power or system ultra-low power modes. The following functions are used:
		`Cy_SysPm_SystemEnterLp(); // Enter System Low Power mode`
		`Cy_SysPm_SystemEnterUlp(); // Enter System Ultra-Low Power mode`

Core Voltage Supply	VCCD_1V1
	VCCD_0V9
		Defines the core voltage supply for the BUCK or LDO – 0.9 V or 1.1 V.
		`Cy_SysPm_SwitchToSimoBuck();`
		`Cy_SysPm_SimoBuckSetVoltage1(CY_SYSPM_SIMO_BUCK_OUT1_VOLTAGE_1_1V);`

Cache Support	RUN_FROM_FLASH
	RUN_FROM_CACHE
		Defines if cache is used. If cache is disabled, the following instructions are used:
		`//Disable CM4 cache`
		`CY_SET_REG32(CYREG_FLASHC_CM4_CA_CTL0,`
		`CY_GET_REG32(CYREG_FLASHC_CM4_CA_CTL0) & CACHE_DISABLE_MASK);`
		`//Disable CM0+ cache`
		`CY_SET_REG32(CYREG_FLASHC_CM0_CA_CTL0,`
		`CY_GET_REG32(CYREG_FLASHC_CM0_CA_CTL0) & CACHE_DISABLE_MASK);`
		The `Cy_SysLib_SetWaitStates()` function is called based on the CACHE support. If disabled, set the maximum allowed frequency for the given System Mode. If CACHE is enabled, set based on the CM4 CPU frequency.

CM4 CPU Mode	CM4_WHILE_LOOP
	CM4_DHRYSTONE
	CM4_SLEEP
	CM4_DEEP_SLEEP
		Defines what the CM4 CPU runs – While (1) loop, Dhrystone algorithm, go to CPU sleep, or go to CPU deep sleep. The following functions are used:
		`while (1); // Runs a forever while loop`
		`dhrystone(); // Runs the Dhrystone instructions`
		`Cy_SysPm_CpuEnterSleep(CY_SYSPM_WAIT_FOR_INTERRUPT); // Sleep`
		`Cy_SysPm_CpuEnterDeepSleep(CY_SYSPM_WAIT_FOR_INTERRUPT); // Deep-Sleep`

CM4 CPU Frequency [CM4_FREQ_MHZ]	FREQ_8_MHZ
	FREQ_25_MHZ
	FREQ_50_MHZ
	FREQ_100_MHZ
		Defines the clock frequency for HFCLK0, which is linked to the CM4 CPU clock. The following functions are used:
		`Cy_SysClk_FllConfigure(FREQ_8_MHZ, CM4_FREQ_MHZ,`
		`CY_SYSCLK_FLLPLL_OUTPUT_AUTO); // Configure the FLL`
		`Cy_SysClk_FllEnable(TIMEOUT_LOCK); // Enable the FLL`

CM0+ CPU Mode	CM0P_WHILE_LOOP CM0P_DHRYSTONE CM0P_SLEEP CM0P_DEEP_SLEEP CM0P_HIBERNATE	Defines what the CM0+ CPU runs – While (1) loop, Dhrystone algorithm, go to CPU sleep, go to CPU deep sleep or hibernate The same functions from the CM4 CPU Mode are used. `Cy_SysPm_SystemEnterHibernate(); // Go to Hibernate`

CM0+ CPU Frequency [CM0P_FREQ_MHZ]	FREQ_8_MHZ
	FREQ_25_MHZ
	FREQ_50_MHZ
	FREQ_100_MHZ
		Defines the clock frequency for the CM0+ CPU clock. The following function is used:
		`/* Set the PERI Clock Divider */`
		`Cy_SysClk_ClkPeriSetDivider((CM4_FREQ_MHZ/CM0P_FREQ_MHZ)-1);`

Clock Source	USE_IMO
	USE_FLL
		Defines the clock source for the HCLK0 – IMO or FLL. If IMO is used, the FLL is bypassed. If the FLL is used, the IMO is still used as the source for the FLL.
		The following functions are used:
		`Cy_SysClk_ClkPathSetSource(0UL, CY_SYSCLK_CLKPATH_IN_IMO);`
		`Cy_SysClk_ClkHfSetSource(0UL, CY_SYSCLK_CLKHF_IN_CLKPATH0);`

Minimum Regulator Current Mode	MIN_CURRENT	Determines whether the firmware calls the `Cy_SysPm_SystemSetMinRegulatorCurrent()` function.

Document History

Document Title: AN219528 - PSoC 6 MCU Low-Power Modes and Power Reduction Techniques

Document Number: 002-19528

Revision	ECN	Submission Date	Description of Change
**	5862341	09/12/2017	New application note.
*A	6166204	05/05/2018	Updated power mode names to match TRM names; CPU active, CPU, sleep, CPU deep sleep, system LP, system ULP, system deep sleep, system hibernate.
*B	6697788	12/19/2019	Added CE226306 PSoC 6 MCU Power Measurements Added reference to AN227910 Low-Power System Design with CYW43012 and PSoC 6 MCU Added sections on Low Power Assistant, Power Estimator, measuring current with a DMM, and approximating power consumption Added design details for the following IP blocks: Disabling PSoC CPUs, Splitting Tasks Between the CPUs, SRAM, TCPWM, SCBAudio Subsystem, USB, Low-Power Comparator, SAR ADC, Voltage DAC, Opamp. Added PSoC 6 Power Calculator Excel spreadsheet file download

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

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709