AN219528

PSoC 6 MCU 的低功耗模式以及降低功耗技术

作者：Brian Lee

相关器件系列：所有 PSoC® 6 MCU 部件

相关文档：要想获取完整的列表，请点击此处。

更多代码示例？我们明白。

如需寻找包含上百 PSoC 代码示例并有不断更新的网上资源，请浏览我们的代码示例网页。您还可以在此处观看 PSoC 视频库。

AN219528 说明了如何使用 PSoC 6 MCU 功耗模式来优化功耗。主要内容包括 PSoC 6 MCU 器件中的低功耗模式，以及使用 PSoC 6 MCU 功能的电源管理技术。相关代码示例演示了不同的低功耗技术。

目录

1 简介

当采用其他节能特性和技术实现低功耗模式时，PSoC 6 MCU 可提供最佳的节能优势，同时不会明显牺牲性能。本应用笔记不仅介绍了一般的省电方法，还介绍了 PSoC 6 MCU 中独特的低功耗模式。它还讨论了其他低功耗注意事项。

本应用笔记需要具备 PSoC 架构的基本知识，以及使用 PSoC Creator™ 或 ModusToolbox™开发 PSoC 6 MCU 应用的能力。如果您不熟悉 PSoC 6 MCU，请参见 AN221774 - Getting Started with PSoC 6 MCU 或 AN210781 – Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity on PSoC Creator。如果您不熟悉 PSoC Creator，请参阅 PSoC Creator™ IDE。

如果您要设计物联网（IoT）系统，则还有其他与连接有关的功耗注意事项，这些因素会影响系统的总功耗。 尽管这些注意事项不在本应用笔记（AN）的讨论范围之内，但仍有一些针对低功耗 IoT 系统的应用笔记。 AN227910 – Low-Power System Design with CYW43012 and PSoC 6 MCU 讨论了 Wi-Fi，蓝牙和 PSoC 6 MCU 系统中的低功耗优化。

2 功耗模式

2.1 功耗模式转换

PSoC 6 MCU 具有 7 种功耗模式，分为影响整个器件的系统模式，以及仅影响一个 CPU 的标准 Arm® CPU 模式。系统电源模式为低功耗（LP）、超低功耗（ULP）、深度睡眠和休眠。Arm CPU 功耗模式为活动、睡眠和深度睡眠，可在系统 LP 和 ULP 电源模式下使用。表 1 列出了器件工作的功耗模式。

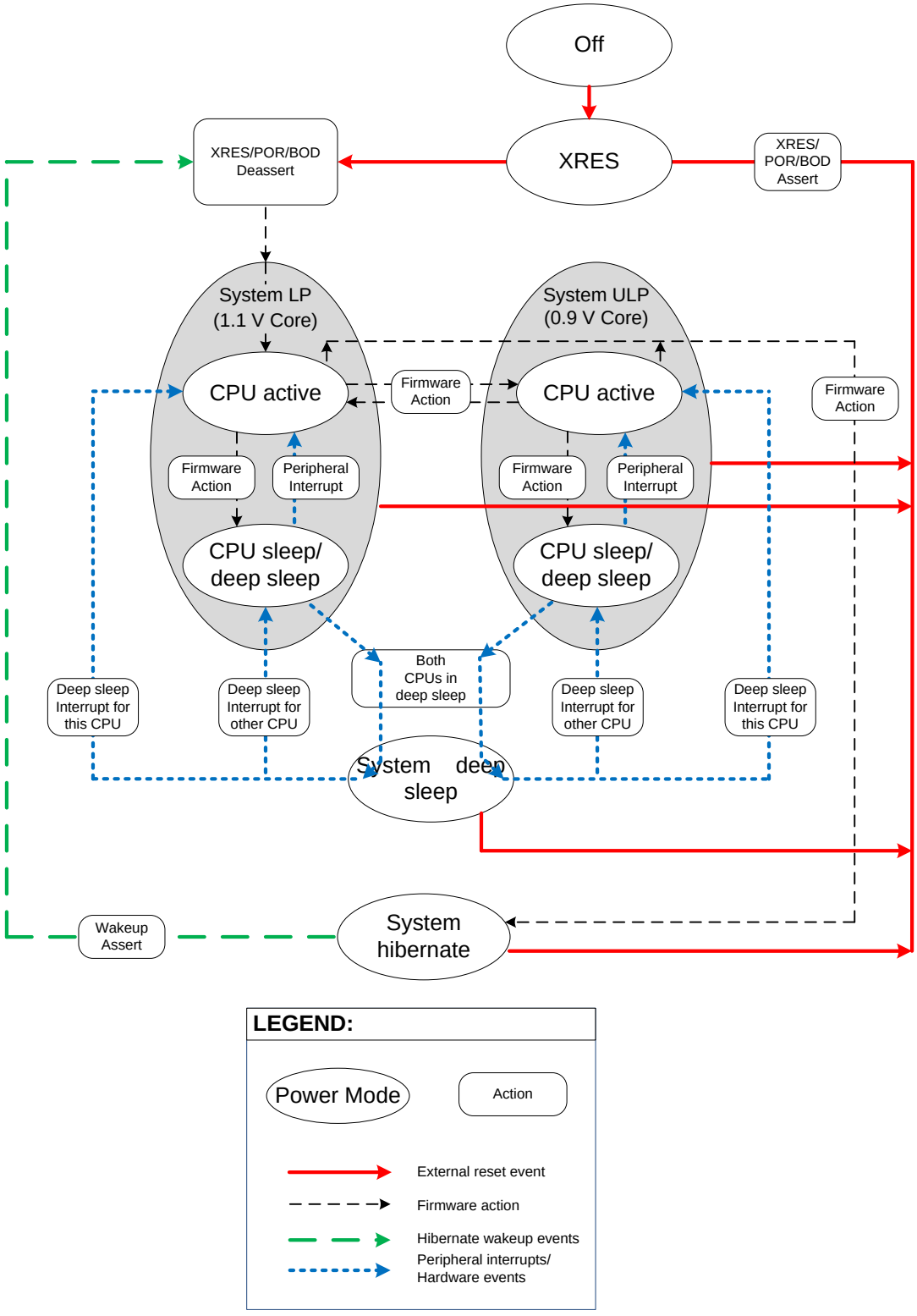
表 1. 功耗模式说明

功耗模式	说明
系统低功耗	• 所有资源均可以最大功率和速度使用 • 支持所有 CPU 功耗模式。.
系统超低功耗	• 所有模块均可用，但核心电压降低，导致高频时钟频率降低。 • 支持所有 CPU 功耗模式。
CPU 活动	• 正常的 CPU 代码执行 • 适用于系统 LP 和 ULP 功耗模式
CPU 睡眠	• CPU 暂停代码执行 • 适用于系统 LP 和 ULP 功耗模式
CPU 深度睡眠	• CPU 暂停代码执行。 • 请求系统进入深度睡眠 • 适用于系统 LP 和 ULP 功耗模式
系统深度睡眠	• 当所有 CPU 处于 CPU 深度睡眠状态时发生 • CPU，大多数外设和高频时钟均为 OFF 状态 • 低频时钟开启 • 低功耗模拟和一些数字外设可用于操作和唤醒源 • 保留 SRAM
系统休眠	• CPU 和时钟处于关闭状态 • GPIO 输出状态被冻结 • 低功耗比较器，RTC 报警和专用 WAKEUP 引脚可用于唤醒系统 • 备份域可用 • SRAM 未保留。

图 1 显示了功耗模式转换如何基于不同的事件和操作，包括中断、固件操作和复位事件。在某些情况下，模式转换通过多种模式完成。

有关更多详细信息，请参见 PSoC 6 MCU Architecture Technical Reference Manual 和 Appendix A（PSoC 6 MCU 架构技术参考手册和附录 A）。

图 1. PSoC 6 MCU 器件功耗模式转换示意图

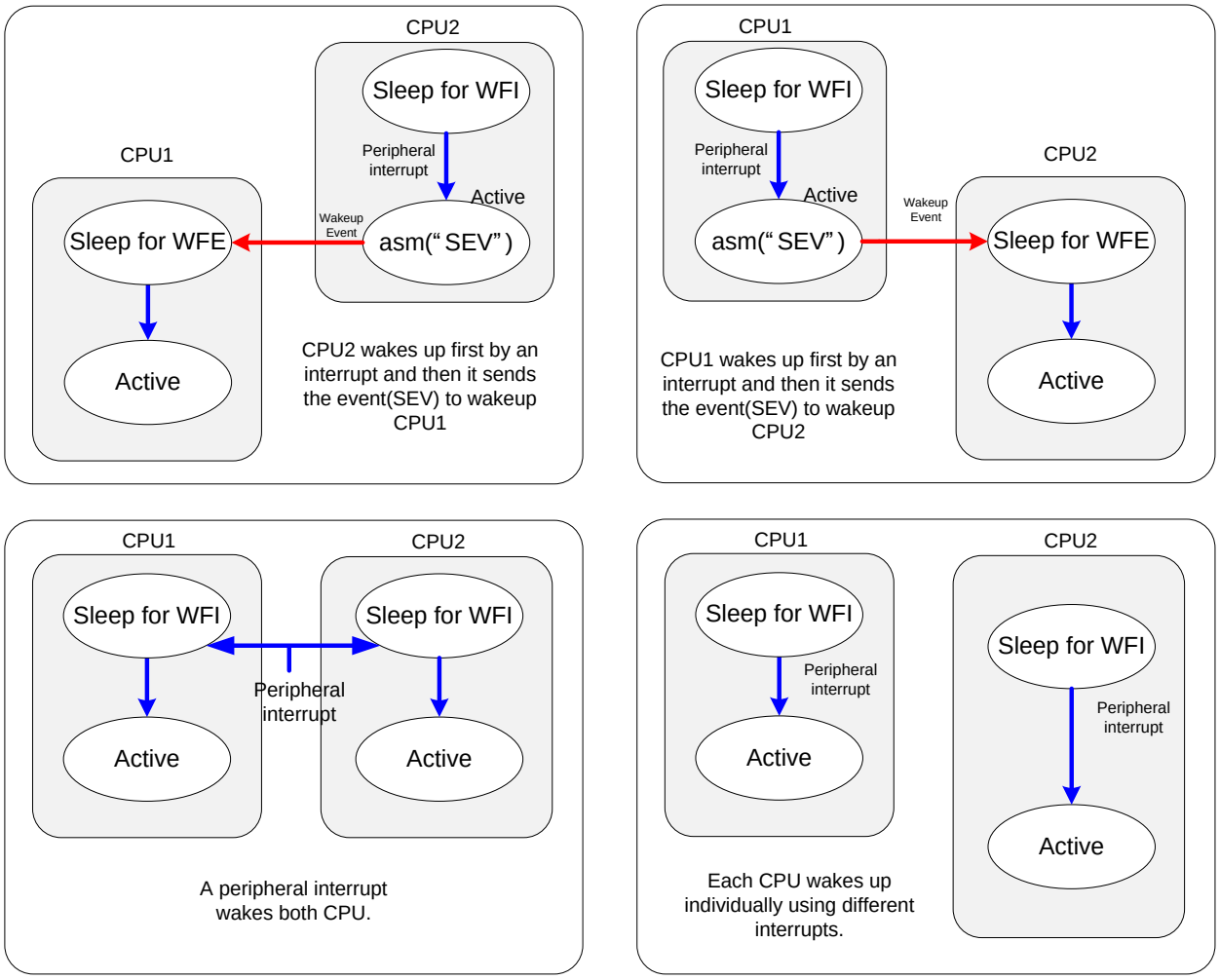


2.2 CPU 睡眠和唤醒指令

Arm Cortex® CPU 于睡眠和唤醒之间独立转换。图 2 显示了从睡眠中唤醒的几种情况。

Wait-for-Interrupt (__WFI) 是核心睡眠指令。CPU 执行__WFI 后，CPU 进入睡眠状态并保持睡眠状态，直到任何中断被置位。Wait-for-Event (__WFE) 类似于__WFI，但是当接收到唤醒事件而不是中断时它会唤醒。由于__WFE，Set Event (__SEV)用于在睡眠模式下唤醒其他 CPU。CPU 深度睡眠使用相同的睡眠和唤醒指令，但是在睡眠指令之前设置了 Arm 系统控制寄存器（SCR）的 SLEEPDEEP 位[2]。有关 SCR 的更多信息，请参阅 Arm System Control Register User Guide。此进程在 SysPm PDL 库 API 中执行。

图 2. 多 CPU 睡眠和唤醒情况



CPU 电源模式与系统电源模式不同。图 2 显示每个 CPU 都支持自己的睡眠模式，与其他 CPU 的状态无关。当两个 CPU 处于深度睡眠状态时，设备处于系统深度睡眠模式。有关更多详细信息，请参见 AN215656 – PSoC 6 MCU Dual-CPU System Design (AN215656 - PSoC 6 MCU 双 CPU 系统设计)。

2.3 低功耗助手

2.3.1 低功耗助手特性

低功耗助手（LPA）提供易于使用的 GUI，用于设置设备和系统电源选项。图 3A LPA 的目标是帮助在现实世界的用户应用中快速获得数据表的电源编号。 对于设备中支持的每个电源选项，ModusToolbox 设备配置器都有相应的部分，以与配置任何其他外设相同的方法配置电源。

要启动助手，对于 PSoC 6 MCU 器件，从 PSoC 6 MCU 的部件号选项卡和 System 子选项卡下，选择 Power resource，如图 3A 所示。 对于连接设备，从 Wi-Fi 或 Bluetooth 设备部件号页签下，在 Power 部分中选择 BT 或 Wi-Fi 资源，如图 3B 所示。 请参阅电源配置器顶部的“文档”部分中的 LPA 文档。

图 3A. PSoC LPA 选择

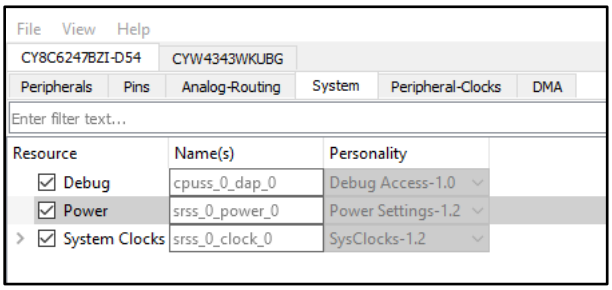
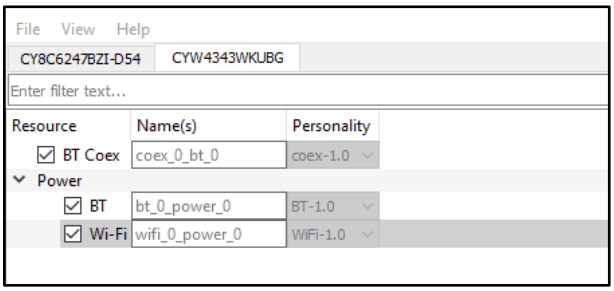


图 3B. 连接设备 LPA 选择

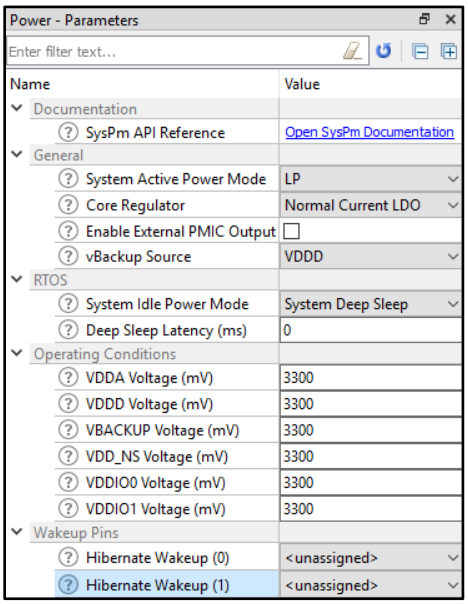


2.3.2 PSoC 6 MCU 设备 LPA 设置

使用 LPA 的 PSoC 6 MCU 设备部分来配置设备的初始功耗配置，如图 4 所示。使用 LPA 的 General 部分来配置设计中系统功耗模式，核心稳压器，和 Vbackup 域功能。配置最高功耗模式后，请使用 RTOS 部分配置系统自动转换到的最低功耗模式。在许多设计中，这是所有需要的功耗模式配置，因为 RTOS 通常会处理所有动态模式转换。对于更高级的用例，可以随时调用 HAL 或 SysPm PDL 库功率功能，以动态修改任何功耗设置或启动功耗模式转换。

触发任何已配置的外设中断时，CPU 和系统会从睡眠和深度睡眠模式唤醒。您可以通过在 ModusToolbox 设备配置器中的外设配置器中启用所需的唤醒外设的中断，用于配置睡眠和深度睡眠唤醒中断。如果您的系统在运行时使用 HAL 或 SysPm 调用进入了休眠模式，则 LPA Wakeup pins 部分将简化对休眠唤醒源的选择。

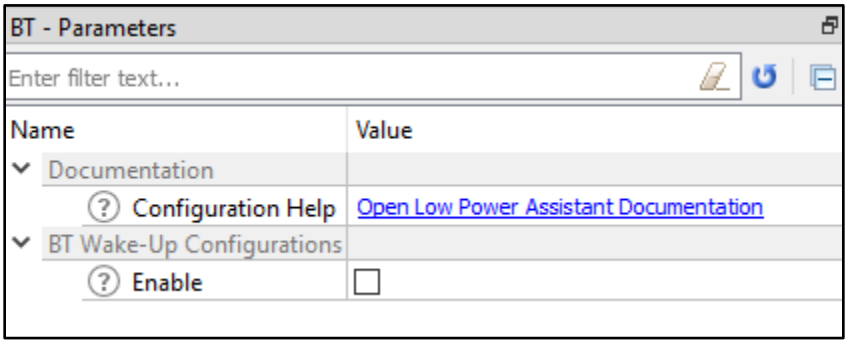
图 4. PSoC LPA 功耗参数



2.3.3 连接设备 LPA 设置

使用 LPA 的“连接设备”部分来配置 Wi-Fi 和蓝牙电源选项。 赛普拉斯的蓝牙连接设备旨在自动以最低功耗模式运行。主要电源选项是启用 BT 设备主机唤醒触发器，如图 5 所示。BT 主机唤醒允许 PSoC 6 MCU 主机设备进入系统深度睡眠或休眠模式，并等待 BT 连接设备唤醒 MCU，这一操作通过在需要处理 BT 操作时生成 GPIO 引脚中断来实现。

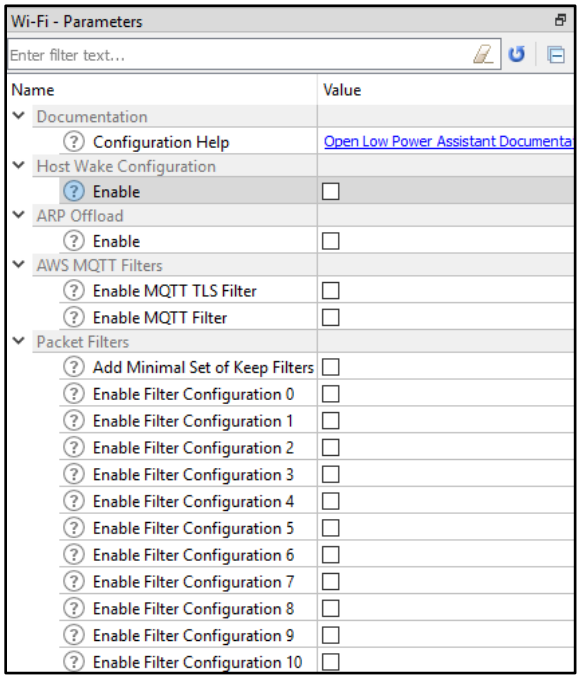
图 5 蓝牙 LPA 功耗参数



赛普拉斯的 Wi-Fi 连接设备支持多种电源优化技术，您可以根据应用程序的需求和所支持的网络特性进行配置，如图 6 所示。Wi-Fi 的三个主要低功耗功能是主机唤醒、卸载和过滤。

- **主机唤醒**允许 PSoC 6 MCU 主机进入系统深度睡眠或休眠模式，直到 Wi-Fi 设备需要 MCU 处理为止。当主机 MCU 必须唤醒时，Wi-Fi 设备会触发 GPIO 引脚中断。要使用大多数其他低功耗 Wi-Fi 功能，需要唤醒主机。
- **卸载**允许将 MCU 的设备堆栈处理转移（卸载）到 Wi-Fi 设备内部处理器。这增加了 MCU 主机可用于其他任务或在低功耗模式下花费的时间。
- **筛选器**在 Wi-Fi 设备上运行，并避免在满足筛选器条件之前唤醒主机 MCU。一个示例是仅需要响应 MQTT 数据包的 IoT 设备。该设备可使 MQTT 过滤器忽略所有其他网络流量，从而使主机 MCU 设备在低功耗模式下停留更长时间。

图 6. Wi-Fi LPA 功率参数



2.4 子系统可用性和功耗

2.4.1 子系统可用性

每个子系统资源在系统功耗模式下的工作不一样。例如，CPU 可以处于开、关和保留模式。选择合适的外设以使电源模式正常工作非常重要。表 6 列出了不同功耗模式下可用的资源。

2.4.2 近似功耗

器件数据表提供给定条件的功耗数据。由于存在不同的组合以实现最佳功耗，因此应用的实际功耗可能与数据表不同。

2.4.3 功耗评估器

ModusToolbox IDE 2.0 版本提供了 Cypress Power Estimator (CyPE)工具，它可以在运行时动态估计目标设备或平台消耗的功耗。这个工具可以帮助你确定不同工作模式下的功耗，以及实际系统条件下的功耗变化情况。

图 7. CyPE 工具

2.5 示例场景

适当的电源模式选择可降低功耗，而不会降低性能。表 2 列出了功率模式的示例情况。在一些示例中，仅少数功率模式可有效使用。

表 2. 功率模式的示例场景

功耗模式	可穿戴设备	空调	遥控器	温度计
系统低功耗 CPU 活动	用户 GUI 互动	马达运行	–	通过 BLE 通信
系统超低功耗 CPU 活动	处理心跳	–	发送命令	读取温度 在 LCD 上更新结果
系统低功耗 CPU 睡眠	–	–	–	–

功耗模式	可穿戴设备	空调	遥控器	温度计
系统超低功耗 CPU 睡眠	模拟模块检测心跳	–	–	–
系统深度睡眠	当设备 30 秒未检测到心跳时进入深度睡眠状态（设备未使用）	等待命令 无马达运行 由红外触发唤醒	–	使用看门狗定时器(WDT)每秒唤醒一次
系统休眠	低电量-不做任何操作 充电器插入时复位设备	–	等待按压按钮	–

2.6 系统电源管理（SysPm）库

2.6.1 概述

赛普拉斯外设驱动程序库（PDL）是一个完整的软件工具，包括用于配置外设和系统寄存器以实现所需功能的 API。PDL 可以直接访问目标器件的几乎所有硬件资源。它减少了理解和直接访问寄存器和位结构的需要。

在 PDL 中，SysPm API 提供了更改电源模式的功能，如图 1 所示。API 还可以注册回调函数，以便在电源模式转换之前或之后执行外设功能，如图 9 所示。

根据您使用的开发平台，PDL 有两个来源。这两个版本的 PDL 共享相同的 API，但与其他版本的工具库不兼容。

- - cypress.com 的 PSoC Creator Peripheral Driver Library。PSoC Creator 的下载和安装包括 PDL，因此不需要额外的用户操作。
- - 来自 github.com 的 ModusToolbox Peripheral Driver Library (psoc6pdl)。ModusToolbox 安装时会自动安装 PDL，但安装过程中需要直接从 GitHub 下载。与第三方开发环境一起使用时，需要手动下载并安装。

2.6.2 模式转换功能

图 1 显示了电源模式的固件转换。SysPm 为 CPU 睡眠、CPU 深度睡眠，系统休眠、系统 LP 和系统 ULP 提供默认的五种转换功能。

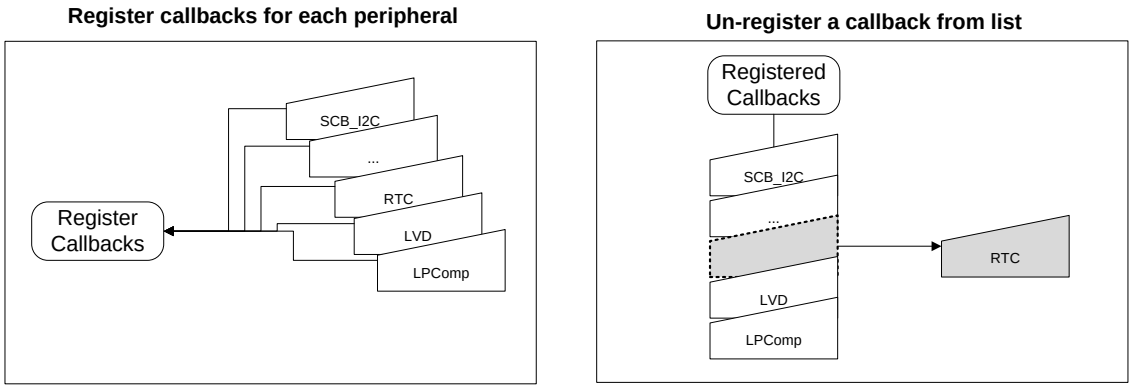
电源模式更改功能提供四种不同的回调操作，以便为每个外设执行必要的操作：

- `CY_SYS_PM_CHECK_READY`: 检查就绪状态以转换到其他模式。如果返回 `CY_SYSPM_FAIL`，则退出而不进行转换。
- `CY_SYSPM_BEFORE_TRANSITION`: 回调在模式转换之前执行并配置所需的操作。
- `CY_SYSPM_AFTER_TRANSITION`: 在模式转换或配置后执行回调。
- `CY_SYS_CHECK_FAIL`: 仅当 `CY_SYSPM_CHECK_READY` 失败时才执行回调。它执行回滚操作。

SysPm 驱动程序提供三种回调函数：注册、取消注册，和执行。这些功能不仅有助于功耗优化，还有助于防止模式转换后的异常外设状态。PDL 期望用户为每种电源模式注册回调，如图 8 所示。大多数外设驱动程序都具有与每种电源模式相关的预定义回调。您可以选择注册定义的外设回调，也可以进行自定义回调。SysPm 转换函数按顺序执行已注册的回调。首先执行第一个注册的功能。

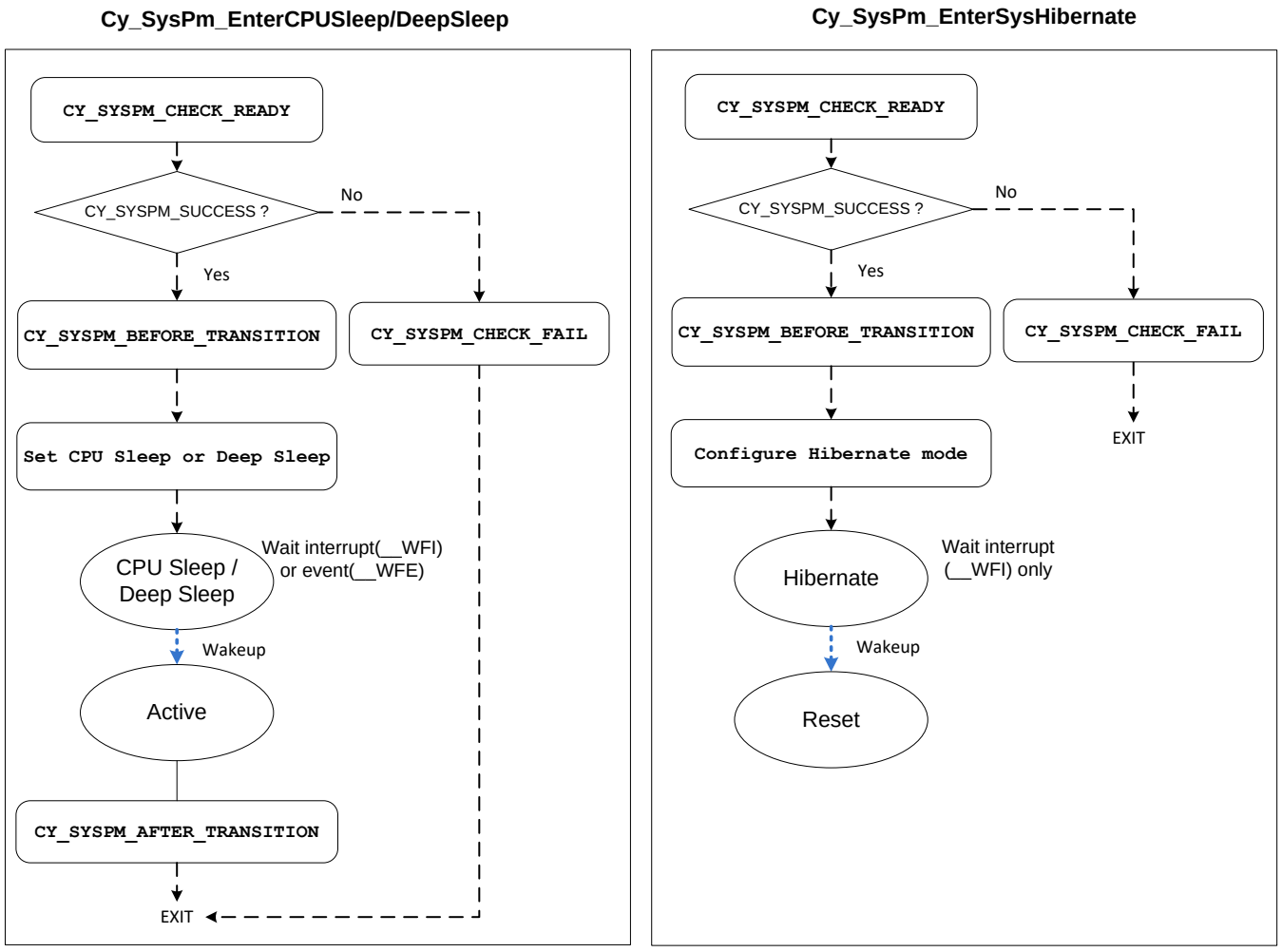
有关回调注册和实现的更多信息，请参阅 Appendix C 和 D.1 CE219881 - PSoC 6 MCU 切换功耗模式，这是 CPU 活动、CPU 睡眠、系统低功耗、系统超低功耗、和系统深度睡眠的模式转换示例。

图 8. 电源模式回调注册和取消注册



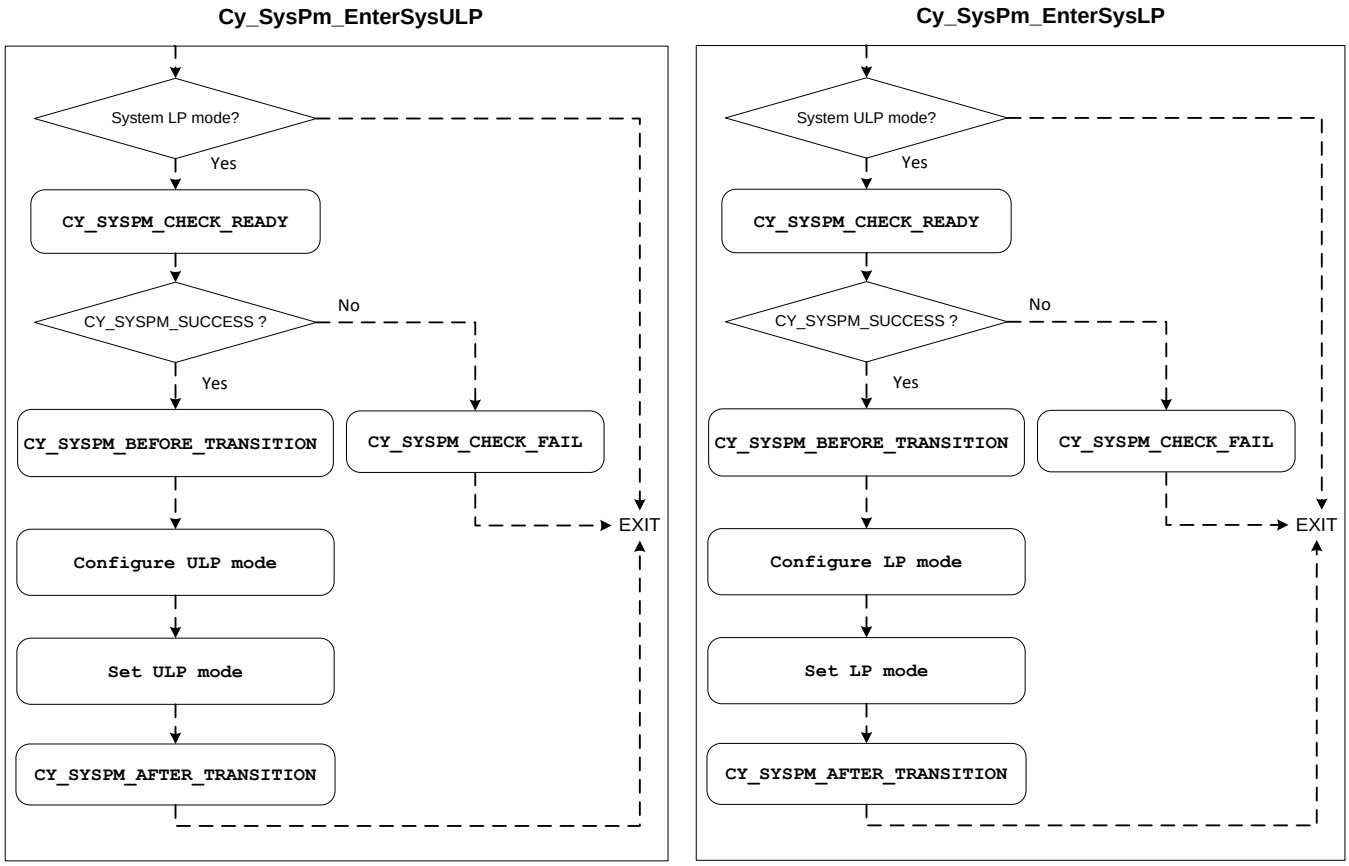
通过调用模式转换功能，设备开始通过四次回调操作进行转换。CPU 睡眠和 CPU 深度睡眠模式使用 Arm 睡眠指令。在 CPU 睡眠功耗模式下，代码执行停止并等待中断。图 9 显示了 CPU 通过调用睡眠指令等待唤醒源：`WFI()`或`__WFE()`。因此，在执行睡眠指令之后，实际模式转换到系统深度睡眠。唤醒后，设备会自动转换为 CPU 活动状态。

图 9. 睡眠/深度睡眠/休眠模式转换流程图



在系统 LP 和系统 ULP 模式中，所有系统资源都在继续运行。因此，通过配置功耗模式控制寄存器来进入 LP 和 ULP 模式，并且没有延迟或等待中断。SysPm PDL 提供相关的驱动程序功能，如图 10 所示。为获得最佳功耗效率，必须配置核心电压调节器和系统时钟。更多详细信息，请参见 3.1 核心电压选择, 3.2 ULP 模式时钟，和外设驱动程序库文档（**PSoC Creator > Help > Documentation > Peripheral Driver Library**）。

图 10. 低功耗模式转换流程图



3 PSoC 6 MCU 功耗管理技术

3.1 核心电压选择

3.1.1 线性稳压器和降压稳压器

PSoC 6 MCU 支持多个片上稳压器包括低压差（LDO）和单输入多输出（SIMO）或单输入单输出（SISO）降压器，用于为核心电源生成 V_{CC}，如表 3 所列。LDO 可提供多达高电流模式（正常）时为 300 mA，低电流模式下为 25 mA。降压稳压器可为一个输出和 30 mA 组合输出和 SIMO 降压器提供高达 20 mA 的电流。SIMO 降压稳压器在正常负载条件下提供更高的效率。切换到 SIMO 降压稳压器后，如果不重置设备，则无法切换回 LDO。

表 3. 用于低功耗配置文件的核心稳压器的不同选择

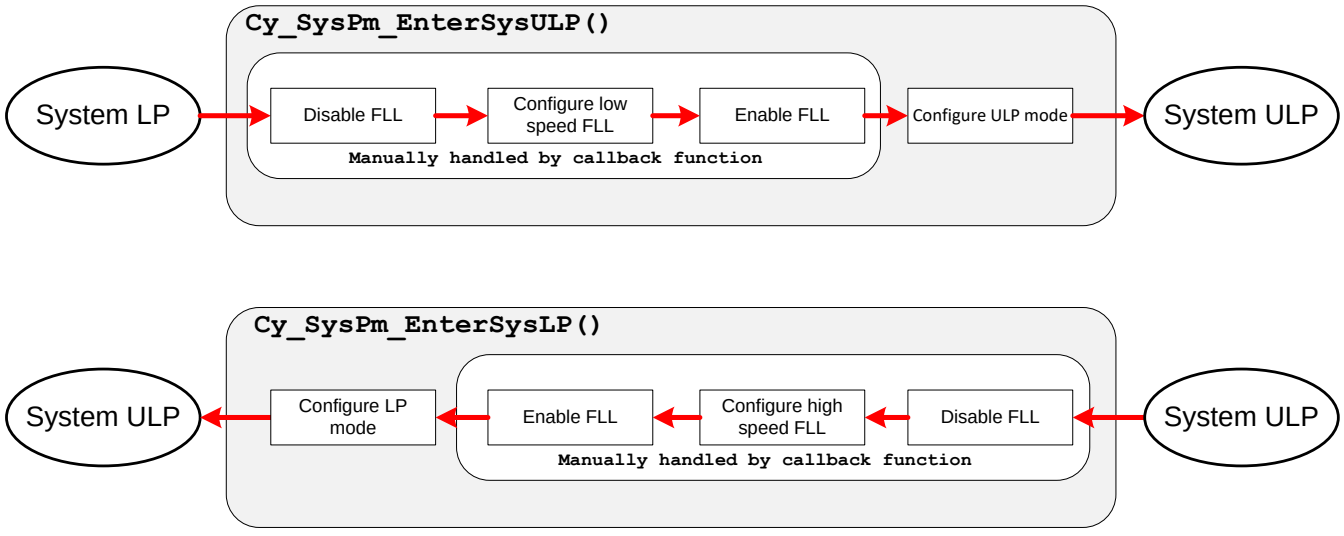
	输出	最大负载	最大时钟频率
LDO	0.9 V	25/300 mA（低电流模式/正常模式）	Cortex-M4 (CM4) 为 50 MHz Cortex-M0+ (CM0+) 为 25 MHz
	1.1 V	25/300 mA（低电流模式/正常模式）	运行最大可支持时钟频率
Buck	0.9 V	20 mA	CM4 为 50MHz CM0+ 为 25MHz
	1.1 V	20 mA	运行最大可支持时钟频率

3.2 ULP 模式时钟

可以通过配置功耗模式控制寄存器来完成到 ULP 模式的转换。如前所述，ULP 模式中存在最大时钟速度限制，因此在进入或退出 ULP 模式时，应根据稳压器输出调整时钟配置。PDL 提供配置 PWR_CTL 寄存器的相关功能。有关更多信息，请参见 PSoC 6 MCU Registers Technical Reference Manual。

图 11 显示了如何使用 PDL 函数和注册的回调函数在 LP 和 ULP 模式之间进行转换。由于 ULP 模式的时钟限制，应在模式转换之前调整锁频环（FLL）时钟速度或 HFClk 为有效频率。更改 FLL 频率会影响使用 FLL 衍生时钟的模块，因此所有活动外设都应注册自己的回调以处理变化的频率。CE219881 – PSoC 6 MCU Switching between Power Modes 提供了时钟调整回调的示例。

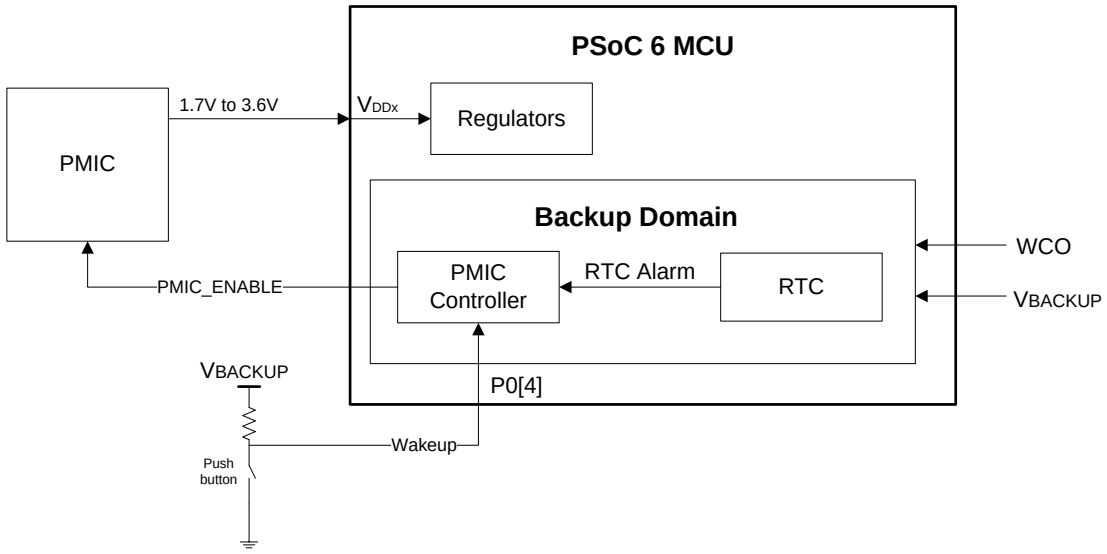
图 11. LP 模式进入/退出转换



3.3 外部 PMIC 控制

PSoC 6 MCU 备用电源域提供电源管理 IC（PMIC）控制功能。图 12 显示了外部 PMIC 供电系统电源（V_{DDD}）。PMIC 可以完全关闭 PSoC 6 MCU，但是用于控制 PMIC 的备用电源可以使备份域保持活动状态。备份域包括一个 RTC、有限的内存保留和 PMIC 唤醒功能，允许 PMIC 重新启动整个 PSoC 6 MCU 设备。

图 12. 外部 PMIC 控制模块示意图



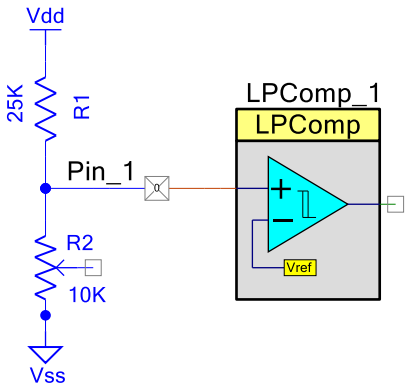
4 其它降低功耗技术

4.1 使用 PSoC 6 MCU 控制电流路径

您的 PCB 板上可能包含其他耗电组件。PSoC 6 MCU 可用于通过它们控制电源，通过 GPIO 引脚提供电源，这些引脚可在固件中打开和关闭。请注意，不能超过数据表中列出的最大引脚拉电流和灌电流能力。如果需要比 GPIO 能够直接提供的电流更大，可以使用外部电源设备，这些设备由 GPIO 控制。

图 13 显示的低功耗比较器(LPComp)应用是属于这种情况的良好示例。在这种情况下，PSoC 设备会比较模拟引脚上的电压（随电位计电阻的变化而变化）。

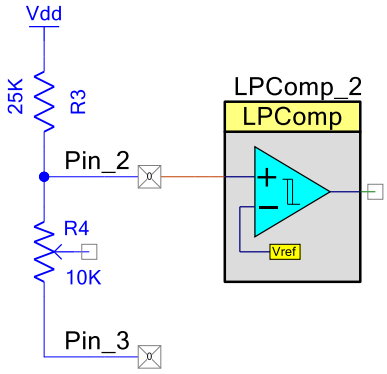
图 13. 典型的 LPComp 应用



LPComp 可以在不使用时关闭，但外部元件仍会消耗功率，因为通过电阻和电位计的电流路径仍然存在。PSoC 6 MCU 的一个简单解决方案是使用第二个引脚作为接地开关，如图 14 所示。

在这种配置中，可以通过向 Pin_3 写入'1'并允许引脚浮动来停止电流流动。这可以通过将两个电阻上的电压差减小到 0 V 来消除电流消耗。写入“0”可以恢复电流。这种省电功能只使用有一个引脚和几行代码。

图 14. 使用 GPIO 作为接地引脚



4.2 禁用未使用的模块

您可以通过禁用未使用的模块来节省不必要的电流消耗。节省的功耗有赖于禁用的模块。

4.3 使用 DMA 移动数据

每次从 CPU 卸载任务时都可以节省电量，并且可以暂停 CPU 或让它同时执行其他操作。DMA 引擎，可用于系统 LP 和 ULP 模式，无需 CPU 使用即可传输数据。如果 CPU 可以停止工作，节省的功率是指则 CPU 主动模式和 CPU 停止电源模式的差值；如果 CPU 可以以较慢的频率计时，仍然可以完成同样的工作，则为低 CPU 活动电流。

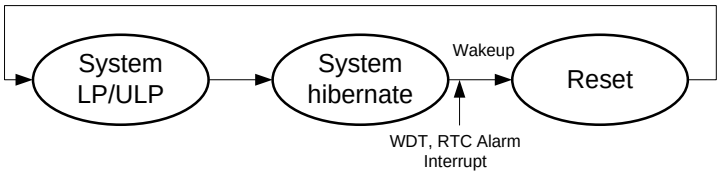
4.4 周期性唤醒定时器

CPU 休眠模式的定期唤醒是降低功耗的最普遍方法。平均功耗由 CPU 活动周期功耗比例和 CPU 休眠周期功耗决定。为了达到最佳效果，睡眠时间应尽可能长，活动期应尽可能短。CY8C61x6 数据表中 CPU 睡眠模式节约功耗的例子是 CM4 主动=1.7 mA (SIDF5)，CM4 睡眠=0.76 mA (SIDS7)。

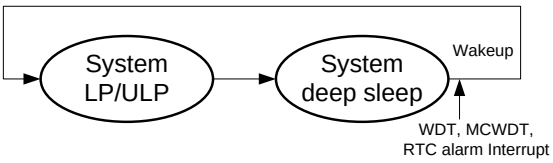
在系统深度睡眠和系统休眠模式下，WDT 和多计数器 WDT（MCWDT）是有效的周期性唤醒源。如果您的应用需要更长或更精确的唤醒周期，RTC 报警可以是一个很好的周期性唤醒源。请参阅 D.3 CE218542 -，了解使用 RTC 周期性定时器进行唤醒的代码示例。CY8C61x6 数据表中的系统深度睡眠模式节电示例是 CM4 活动=1.7 mA (SIDF5)，系统深度睡眠=7 µA (SIDDS1)。

以下方案显示了如何更改模式状态：

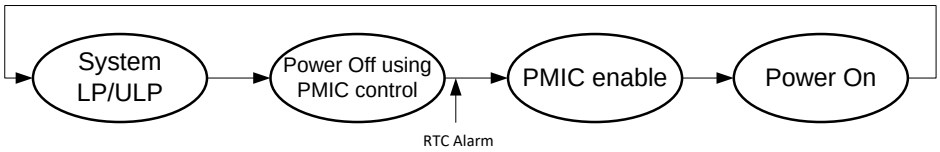
- 系统休眠



- 系统深度睡眠



- 用外部 PMIC 关闭电源



4.5 禁用 PSoC CPU

如果在应用中没有使用 CM4 CPU，可以通过调用 CM0+固件中的 `Cy_SysDisableCM4()`函数禁用它。你也可以将 Clk_Fast 分频器设置为 256，以减少该时钟的扇出泄漏。

如果在应用中没有使用 CM0+ CPU，将其置于系统深度睡眠电源模式，并将 Clk_Slow 分频器设置为 256，以最小化该时钟的任何扇出泄漏。请注意，Clk_Slow 也会影响 DMA 引擎的运行速度。

4.6 在 CPU 之间分割任务

选择 CM4 或 CM0+来完成某些任务有几个原因。比如：

1. 需要浮点运算或 DSP 运算的任务在 CM4 上运行效率更高。
2. 需要最大时钟速度(150 MHz)的时间紧要的任务应该只在 CM4 上运行。
3. 如果一个独立于所有其他任务的单一任务需要确定性的时序，那么将该任务移到自己的 CPU 上简化时序往往是有益的。通常使用 CM0+CPU 来实现这一目的。
4. 当如 4.7 时钟一节所讨论的最大化系统空闲状态时间时，将任务分散在 CM0+和 CM4 之间可能是有益的，这样由于同时进行任务处理，可以更快地进入低功耗的空闲状态。

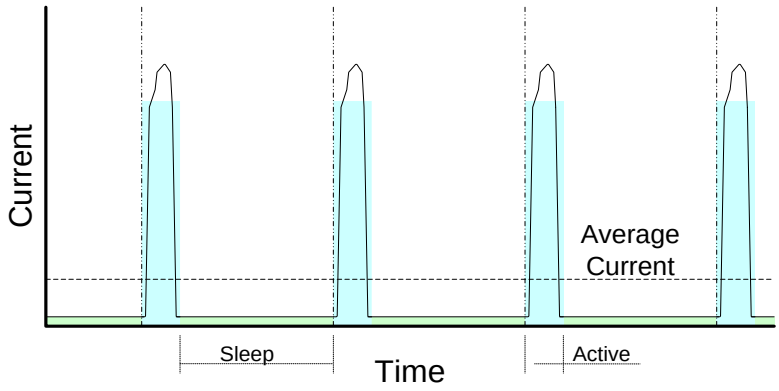
5. 如果由于实时硬件交互限制了在低功率状态下所花费的时间，单个 CPU 的处理必须分散在很长一段时间内进行，那么在相同的活动时间间隔内，CM0+往往会比 CM4 更低功率。
6. 如果需要的处理能力超过了一个 CPU 所能提供的能力，则应将任务分散在两个核心之间。由于每个核心都有自己的总线控制器，并且可以访问独立的内存块，因此由于总线或内存争用造成的等待状态极少。因争用而导致的性能下降通常小于 2%，而第二个核心带来的性能提升则大于 50%（假设 CM0+加到 CM4）。
7. 在保持活动模式的应用中，在单核上可以达到所需的处理能力，大多数情况下，在核心之间共享所需的任务并没有任何功耗上的好处。

4.7 时钟

在某些情况下，更快地运行 CPU 时钟可以降低平均电流消耗。例如，考虑设计，每秒钟从传感器读取一次，执行多次计算，然后将结果传输到另一台设备。

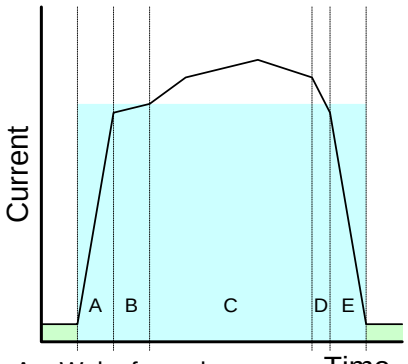
您可以使用 CPU 休眠或系统深度睡眠模式来降低 PSoC 器件空闲时的功耗，但由于在 CPU 活动模式下花费的时间，平均电流消耗可能较高。图 15 表示该示例的电流消耗，系统时钟设置为 3 MHz。

图 15. 时钟频率为 3 MHz 的示例电流分析



根据 PSoC 器件唤醒时正在执行的任务或计算，可以通过更快地运行 CPU 时钟来更快地完成它们。这可以降低平均电流消耗，因为 PSoC 器件处于 CPU 活动模式的时间更短，或处于 CPU 睡眠或系统深度睡眠时间更长。图 16 显示了 CPU 活动模式时序被分解为任务。

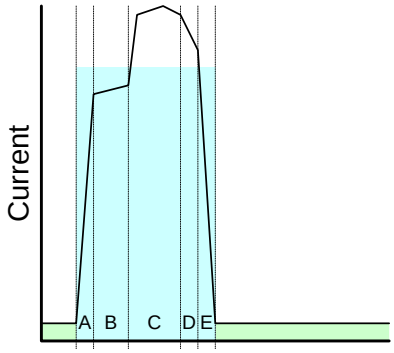
图 16. 3 MHz 时 CPU 活动模式下的任务分析



即使系统时钟频率增加，某些任务所需的时间也不会改变。如传感器读取和数据传输。但是，如果 CPU 以更快的频率运行，则其他任务如数据处理需要的时间更少。

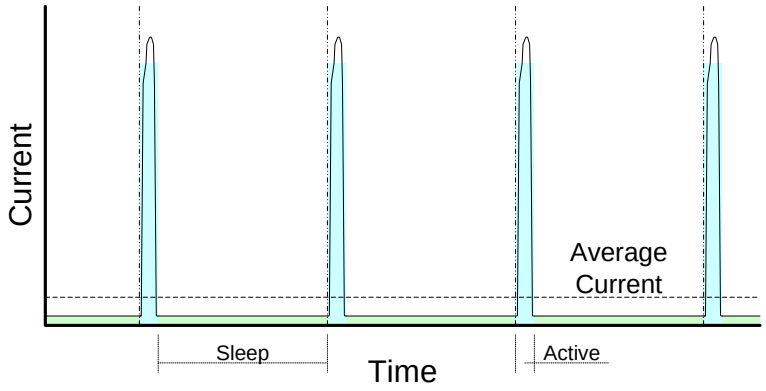
有时，虽然在活动模式下工作的时间短是一个优势，但是这会导致在高频率驱动时钟所消耗的功耗变得更大。假设最佳速度为 48 MHz，如图 17 所示。对于 48 MHz 时钟，在活动模式下的运行时间大概等于 8 MHz 时钟所需的时间的一半。图 18 显示当时钟频率更高时，峰值电流消耗更大，但由于在 CPU 睡眠或系统深度睡眠中花费的时间较长，总体平均值却更低。在 PSoC 6 MCU 设备上，一般情况下，CPU 完成处理工作时运行速度越快越好，这样在低功耗模式下花费更多时间。

图 17. 工作频率为 48 MHz 时，在活动模式下执行的任务分析



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

图 18. 时钟频率为 48 MHz 的示例电流分析



4.8 GPIOs

当 PSoC 器件处于低功耗模式时，GPIO 可以继续驱动外部电路。当您需要在某个固定电平上保持外部逻辑时，该功能非常有用。但如果引脚不需要源或灌电流，这样便浪费功耗。这种技术的具体节电效果取决于连接到具体 GPIO 引脚的电路。

您应该分析系统设计，并确定 GPIO 在低功耗模式下的最佳状态。如果保持数字输出引脚为逻辑 1 或逻辑 0 导致最低电流，那么可以使用 `Cy_GPIO_Write()`函数来匹配该电平。

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

将所有未使用的 GPIO 配置为模拟 High-Z，除非有特殊原因使用不同的驱动模式。High-Z 驱动模式使 GPIO 引脚的电流最小。可以使用 `Cy_GPIO_SetDrivemode()`函数来设置引脚的驱动模式。

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

PSoC 的灵活性使其易于管理 GPIO 驱动模式，以防止不必要的电流泄漏。在系统休眠模式下，GPIO 驱动模式和数据寄存器会自动"冻结"。在被 "解冻 "之前，它们必须被重新配置到一个已知的状态，以避免唤醒复位后的 GPIO 输出转换，并允许在运行时改变它们的状态。在配置 GPIO 引脚并设置其输出驱动状态后，调用 `Cy_SysPm_IoUnfreeze()`。关于休眠和 I/O 控制的代码示例，请参见请参见 D.2 CE218129 - 使用低功耗比较器从休眠状态唤醒 PSoC 6 MCU。

4.9 SRAM

PSoC 6 MCU 器件允许关闭单个 SRAM 组或组内页面的电源。一个组内页数的大小取决于具体的器件和组，详见器件数据手册。特定器件将有一个或多个 SRAM 库。大多数器件都有一个具有较小页面大小（通常为 32 KB）的库，用于细粒度控制启用的 SRAM 数量。任何未使用的页面可以通过写入 CPUSS 电源控制寄存器来禁用。这种技术在系统深度睡眠模式下最有用，在这种模式下，保留 64K SRAM = 7 µA (SIDDS1)，同时保留 256K SRAM = 9 µA (SIDDS2)，如 CY8C61x6 数据表中所列。

```
/* 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;
```

请参考器件的寄存器 TRM，了解所有功率控制寄存器。

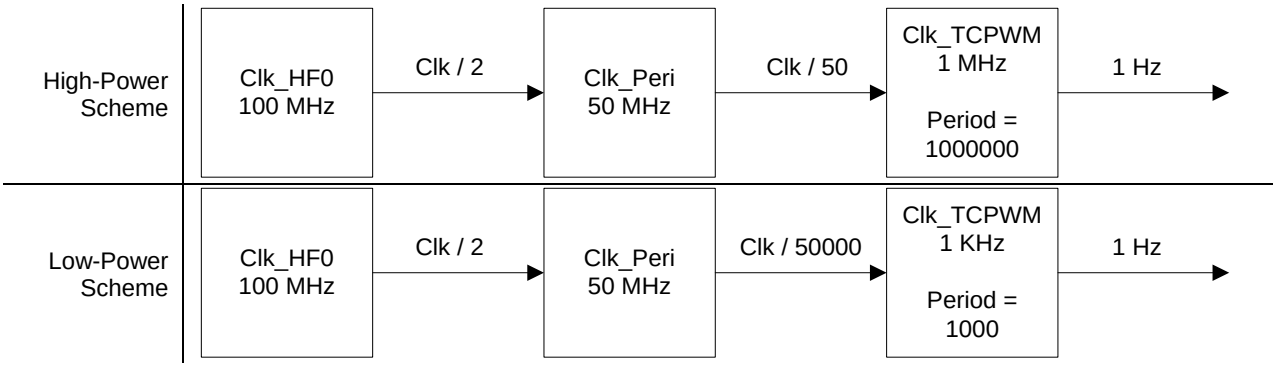
作为高级用例，修改项目的链接器脚本可用于 SRAM 断电，以优化所需的 SRAM 总量，或允许自定义数据放置，以支持动态 SRAM 供电。

4.10 TCPWM

当使用计数器、定时器或 PWM 时，你应该在满足你的频率和精度要求的同时，将通道的时钟源配置得尽可能低。例如，如果你需要用定时器产生 1 秒的中断，那么使用周期等于 1,000 个计数的 1 kHz 的时钟频率比使用周期等于 1,000,000 个计数的 1 MHz 的时钟频率更好。降低 TCPWM 时钟所节省的功率主要是基于时钟频率的线性变化。根据 CY8C61x6 数据表，TCPWM 工作时电流为 100 MHz=540 µA(SID.TCPWM.2B)，而 8 MHz=70 µA(SID.TCPWM.1)。

当使用 PWM 使 LED 变暗时，同样的想法也适用。尽可能使用不会让人眼察觉到 LED 正在闪烁的最小时钟频率。对于大多数应用来说，60 Hz 是一个不错的频率。图 19 显示了 TCPWM 模块的时钟设置比较。

图 19. TCPWM 时钟设置比较



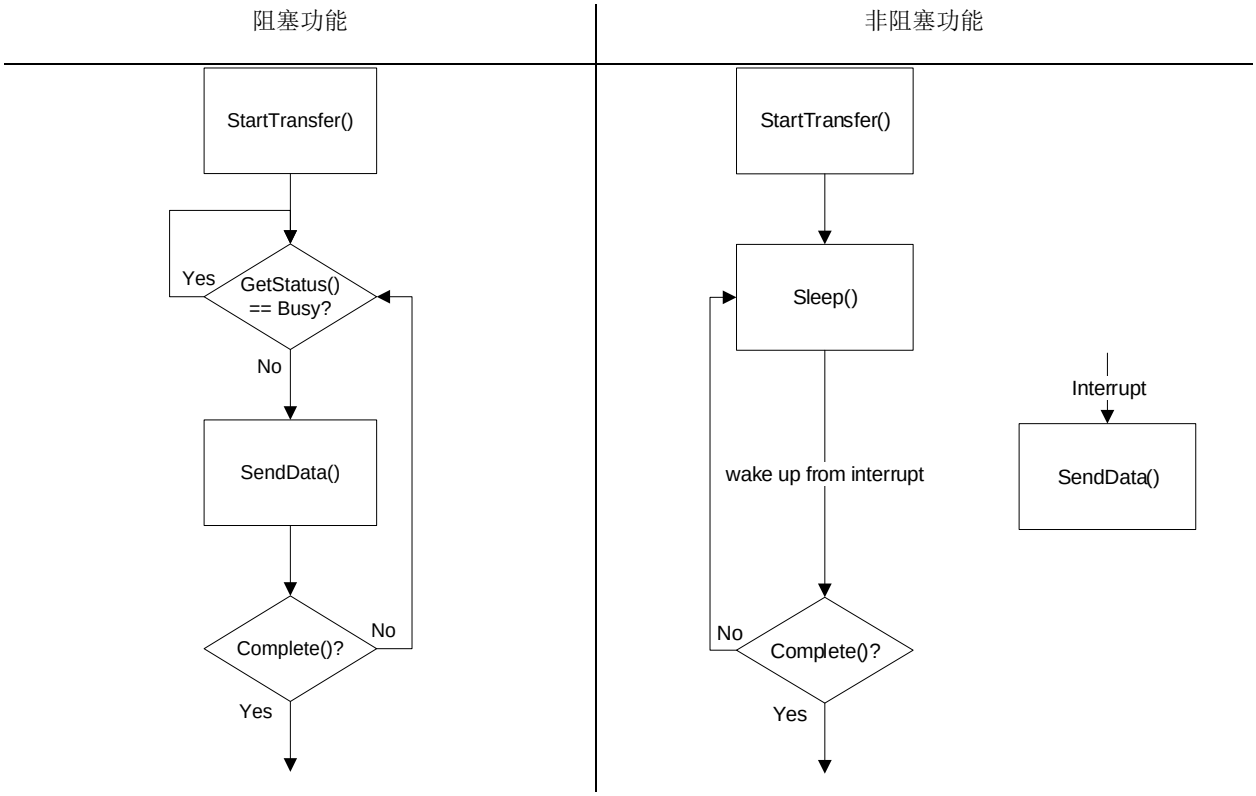
TCPWM 模块有一个时钟预分频器功能。为了最小的功耗，在使用 TCPWM 时钟预分频器之前，先将外设时钟分频器最大化。

如果 TCPWM 模块使用任何引脚连接，则在禁用该模块时将引脚设置为 Analog High-Z。如果 PSoC 器件支持数字系统互连 (DSI)，请选择与所需引脚分配直接连接的 TCPWM 通道，避免使用 DSI。如果 PSoC 设备支持 UDB（通用数字块），请先使用所有固定功能的 TCPWM 通道，而不是基于 UDB 的 TCPWM。

4.11 SCB

发送或接收数据时避免使用阻塞功能。使用基于中断的事件或 RTOS 来传输数据，同时将 CPU 让给其他任务。这种策略背后的想法是让 CPU 处于睡眠状态的时间更长，而不是轮询传输的状态。图 20 显示了一个阻塞和非阻塞功能的例子。

图 20. 阻塞和非阻塞功能示例



不要在固件中使用 SCB 中断来访问其 FIFO，要使用由 FIFO 级别控制的 DMA 来减少应用中所需要的 CPU 循环量；因此，CPU 可以在睡眠状态下停留更长时间或执行其他任务。

如果 SCB 模块使用任何引脚连接，当该模块被禁用时，将引脚设置为 Analog High-Z。如果 PSoC 器件支持 DSI，请选择与所需引脚分配直接连接的 SCB 通道，以避免使用 DSI。如果 PSoC 设备支持 UDB，请在使用基于 UDB 的通信接口之前使用所有固定功能的 SCB 通道。

以系统所能支持的最低数据速率运行也有助于降低功耗，因为 SCB 的功率随时钟频率线性增加。例如在以 100 kbps = 30 µA (SID149)运行时，以 1Mbps = 180 µA (SID152)运行 I2C，如 CY8C61x6 数据表所示。

4.12 音频子系统

I2S 和 PDM/PCM 模块通常由 PLL 产生的高频时钟提供源。对于高精度的帧速率，ECO 为 PLL 提供源。在为 ECO 选择频率时，请考虑所需的音频采样率是多少。如果你需要支持所有标准的音频采样率(8/16/22.05/32/44.1/48 kHz)，你可以使用的最小 ECO 频率是 17.2032 MHz。如果您只需要 8/16/48 kHz 的采样率，请将 PLL 配置为 12.288 MHz。如果您只需要 22.05/44.1 kHz 的采样率，请将 PLL 配置为 22.5792 MHz。

如果不使用音频子系统，不仅要禁用 I2S 和/或 PDM/PCM 模块，还要禁用 PLL 和 ECO，这是造成高电流消耗的主要原因。如果帧速率的准确性对应用不重要，就不要使用 ECO 和/或 PLL。例如，如果您的应用实现了声音检测器，并且可以容忍较高的误差，则使用 FLL 作为音频子系统的源。一旦检测到声音且系统需要完全运行，则启用 PLL 和 ECO 以更高的精度进行采样。

在固件中不使用音频子系统中断来访问其 FIFO，而是使用由 FIFO 级触发器控制的 DMA 来减少应用中所需要的 CPU 周期数，这样 CPU 可以在睡眠状态下停留更长时间或执行其他任务。

4.13 USB

当 USB 总线上没有活动时，您可以暂停 USB 模块以消耗较少的电流。参考 CE223305 – PSoC 6 MCU: USB Suspend and Resume，了解如何设置器件进入和退出低功耗状态。

4.14 低功耗比较器

如果低功耗比较器需要引脚连接和系统深度睡眠工作，请始终选择比较器的专用引脚连接。这样可以让比较器在系统深度睡眠状态下工作，避免使用全局模拟多路复用器，因为全局模拟多路复用器会消耗额外的电流来保持活跃。

有两种方法可以验证比较器的状态。调用 `Cy_LPComp_GetCompare()`函数，或者设置一个中断。在大多数情况下，最好使用中断的方法，减少 CPU 的使用。

比较器支持三种影响其速度和功率的操作模式：

1. **超低功耗/低速**：ICMP1 = 0.85 µA (SID259)- 将响应时间减慢至 20 µs (SID92)。仅在系统深度睡眠或休眠时使用此模式。
2. **低功耗/低**：I_{CMP2} = 10 µA (SID248)- 当不需要系统深度睡眠或休眠，并且可以接受较慢的 1 µs (SID258) 响应时间时的首选该模式。将输入偏移电压提高到 25 mV (SID85A)。
3. **正常功率/快速**：ICMP1 = 150 µA (SID89) - 当需要 100 ns (SID91)的最快响应时间和/或 10 mV(SID84)的低偏移电压时使用。

另一个减少比较器消耗电流的选项是禁用磁滞。可以在 ModusToolbox 配置器、PSoC Creator 定制器或 PDL 配置结构中禁用磁滞。

4.15 SAR ADC

在选择要连接到 SAR ADC 的引脚时，最好先选择连接到 SAR MUX 的专用端口，只有在该端口的引脚用完后，才选择其他端口的引脚。只有在该端口的引脚用完后，再选择其他端口的引脚。只使用专用端口，就不需要使用全局模拟多路复用器，因为它会消耗额外的电流。由于输入电容较低，您还可以通过专用端口实现更高的采样率。

如果不需要 ADC 结果的满级精度，请使用较低的分辨率，并且不要使用平均化，这样可以减少相同采样率所需的 ADC 时钟数。

如果不需要最大采样率，可以考虑使用单镜头模式而不是连续模式。这样可以避免 SAR ADC 一直工作。在单枪模式下，ADC 仅在软件或硬件触发时采样，具体取决于应用。

4.16 电压 DAC

如果 DAC 电压输出需要定期用预设值改变，则使用 DMA 更新电压值。这样可以避免使用 CPU 周期向 DAC 寄存器写入数据。

如果只周期性地使用需要 DAC 输出电压的外部设备，则在不需要其输出时，保持禁用 DAC。

当必须在系统深度睡眠时提供 DAC 输出电压时，使用采样和保持策略，在系统深度睡眠时禁用电压 DAC 时保持电压输出。更多细节请参考代码示例 CE220925 – PSoC 6 MCU VDAC Sample and Hold。根据 CY8C61x6 数据手册，DAC 工作电流=125 µA (SID100D)，而 DAC 停止电流=1 µA (SID101D)。

4.17 Opamp

运算放大器支持三种不同的工作模式--低、中、高。这些工作模式受系统功率模式--系统 LP/ULP 和深度睡眠的影响。在系统深度睡眠中，还有两种模式可供选择。表 4 显示了如何选择合适的工作模式。

表 4.. Opamp 操作模式条件

规格	高功率运算模式		中等功率运算模式		低功率运算模式	
	LP / ULP	深度睡眠	LP / ULP	深度睡眠	LP / ULP	深度睡眠
最小增益-带宽	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
最大输出电流(无负载)	1500 µA	M1: 1500 µA M2: 120 µA	600 µA	M1: 600 µA M2: 60 µA	350 µA	M1: 350 µA M2: 15 µA
响应时间	150 ns		400 ns		2000 ns	

M1: 模式 1 有较高的 GBW 和最大的电流。

M2: 模式 2 的 GBW 较低，电流最小。在选择连接到运算放大器的引脚时，最好选择连接到运算放大器的专用引脚。只使用专用引脚，就不需要使用全局模拟多路复用器，因为它会消耗额外的电流。

5 功率测量

5.1 用 DMM 测量电流

当使用数字万用表(DMM)测量设备电流时，了解 DMM 中分流电阻的值非常重要。DMM 在电流输入之间有一个或多个（分流）电阻。这些电阻器的范围可以从不到 1 欧姆到超过 10kΩ。不同品牌甚至同一供应商的不同型号之间的分流电阻没有标准值。查看您的仪表手册并了解分流电阻的值非常重要，因为该分流电阻上总会有一个电压降。这意味着 PSoC 设备将不会看到您认为您提供的相同电压。如果您仪表中的分流电阻为 1 或更小，您在测量 PSoC 电流时将只看到几毫伏的压降，可以忽略。如果分流电阻为 1 kΩ，有些厂商将其用于低电流测量，那么 1 mA 的电流将导致 1 V 的压降！分流电阻值过高会导致器件在较大电流下因电压过低而复位。另外，在改变量程时，要注意 DMM 不要做先断后续，否则电源会被循环，你的项目会被复位。

对于深度睡眠、休眠和停止模式下的极低电流，一个好的技术是使用零或低电阻分流，直到器件进入低功耗模式。进入低功耗模式后，代码应使器件保持在该模式下，并切换到高电阻分流器进行电流测量。

作为依赖 DMM 分流器的替代方案，大多数 PSoC 6 MCU 套件都包含一个分流电阻或电流测量头的位置。请参阅各套件用户指南以确定具体的电流测量功能。可以替换这些电阻或将其连接到一个小电阻上，以便用电压表测量分流器上的电压。然后，您可以轻松地确定电流。在 1 Ω和 100 Ω之间的分流器应该可以满足大多数应用的要求。

5.2 近似功耗

器件数据表提供了用于估计项目特定用户配置的 PSoC 6 MCU 功耗的信息。为了简化这一过程，我们提供了一个电子表格，其中包括各种内部元件和模式的典型功率要求。该电子表格 *PSoC_6_Power_Estimator.xlsx* 位于 AN219528 页面。由于每个项目都不一样，因此该电子表格提供的功率计算只是对典型值的估计，但应该足够接近，以便在您的设计完成之前提供反馈。

电子表格中有几个选项卡；在输入数据之前，请确保阅读 " Instructions "选项卡。五个 " Config "选项卡允许配置不同的硬件设置和电源模式，器件可以在运行时过渡。" Summary "选项卡显示每个配置的电流，以及基于在每个配置中花费的时间的所有模式的峰值和平均电流。摘要选项卡上的可选电池寿命估计部分根据平均电流估计电池寿命。

6 电源保护系统

6.1 硬件控制

6.1.1 掉电检测（BOD）

掉电检测（BOD）可以在 V_{DDD} 和 V_{CCD} 电源丢失时，逻辑崩溃之前重置系统。掉电系统保证在 V_{DDD} 达到最小系统工作电压之前复位，该电压适用于所有逻辑、SRAM、和闪存。BOD 由硬件控制，没有可配置的寄存器。

6.1.2 低压检测（LVD）

低压检测（LVD）与 BOD 类似，但它的阈值电压是可配置的。当 VDDD 低于配置的跳闸电压时，LVD 会产生中断。该中断允许应用程序在在触发 BOD 复位之前处理重要数据。LVD 提供 15 种可选的跳闸电压。LVD 在系统深度睡眠和休眠模式下不可用。

6.1.3 过压检测（OVD）

过压检测（OVD）与 BOD 相反。当检测到 V_{DDD} 和 V_{CCD} 上的不安全过压电源条件时，这些电路会产生复位。无需固件控制。OVD 有助于保护系统免受高压损坏。

7 总结

PSoC 6 MCU 中可以使用许多电源管理选项。通过遵循适当的方法，您可以优化设计并确保 PSoC 6 MCU 的功耗模式和功能为最低功耗提供最佳选择，而不会降低电池供电设备所需的性能。

8 相关文档

- 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. 功耗模式总结

A.1 功耗模式和唤醒源

表 5. 功耗模式和唤醒源

系统功耗模式	MCU 功耗模式	说明	进入条件	唤醒源	唤醒操作
LP	活动	主要运作模式。1.1V 核心电压。所有外围设备均可用（可编程）。时钟的最大频率。	从外部复位，掉电，上电复位系统和休眠模式复位。从系统 ULP 模式手动寄存器写入。在系统 LP 模式下从 CPU 睡眠或 CPU 深度睡眠唤醒。从 LP 模式进入后从系统深度睡眠唤醒。	不适用	N/A
	睡眠	1.1V 核心电压。处于睡眠模式的一个或多个 CPU（执行暂停）。所有外围设备均可用（可编程）。时钟处于最大频率。	在系统 LP 模式下，CPU 执行禁用深度睡眠的 WFI/WFE 指令	任意 CPU 中断	中断
	深度睡眠	1.1V 核心电压。一个 CPU 处于深度睡眠模式（执行暂停）。其他 CPU 处于活动或睡眠模式。所有外围设备均可用（可编程）。时钟处于最大频率。	在系统 LP 模式下，CPU 在启用深度睡眠的情况下执行 WFI/WFE 指令	任意 CPU 中断	中断
ULP	活动	0.9V 核心电压。所有外围设备均可用（可编程）。时钟频率有限。	从系统 LP 模式手动寄存器写入。在系统 ULP 模式下从 CPU 休眠或 CPU 深度休眠唤醒。从 ULP 模式进入后从系统深度睡眠唤醒。	不适用	N/A
	睡眠	0.9V 核心电压。处于睡眠模式的一个或多个 CPU（执行暂停）。所有外围设备均可用（可编程）。时钟频率有限。	在系统 ULP 模式下，CPU 执行禁用深度睡眠的 WFI/WFE 指令	任意 CPU 中断	中断
	深度睡眠	0.9V 核心电压。一个 CPU 处于深度睡眠模式（执行暂停）。处于活动或睡眠模式的其他 CPU。所有外围设备均可用（可编程）。时钟频率有限。	在系统 ULP 模式下，CPU 在启用深度睡眠的情况下执行 WFI/WFE 指令	任意 CPU 中断	中断
深度睡眠	深度睡眠	所有高频时钟和外围设备均已关闭。低频时钟（32 kHz）和低功耗模拟和数字外设可用于操作和唤醒源。SRAM 保留（可编程）。	两个 CPU 同时处于 CPU 深度睡眠模式	GPIO 中断，低功耗比较器，SCB，CTBm，看门狗定时器和 RTC 报警	中断
休眠	N/A	GPIO 状态被冻结。除低功耗比较器和备用域外，器件中的所有外设和时钟均完全关闭。通过 WAKEUP 引脚，XRES，低功耗比较器（可编程）和 RTC 报警（可编程）可以唤醒。设备在唤醒时重置。	从 LP 或 ULP 模式手动寄存器写入	WAKEUP 引脚，低功耗比较器，看门狗定时器[1]和 RTC[2]报警	复位

[1] 看门狗定时器能够产生休眠唤醒。

[2] RTC（包括 WCO）是备份域的一部分，无论设备电源模式如何，它都可用。 RTC 警报能够从任何电源模式唤醒设备。

Appendix B. 子系统可用性

B.1 不同功耗模式下可用资源

表 6 显示了不同功耗模式下资源可用性的信息。

表 6. 不同功耗模式下可用资源

组件	系统功耗模式							
	LP		ULP		深度睡眠	休眠	XRES	备份关闭电源
	CPU 活动	CPU 睡眠/深度睡眠	CPU 活动	CPU 睡眠/深度睡眠				
核心功能								
CPU	打开	睡眠	打开	睡眠	保留	关闭	关闭	关闭
SRAM	打开	打开	打开	打开	保留	关闭	关闭	关闭
Flash	读/写	读/写	只读	只读	关闭	关闭	关闭	关闭
高速时钟 (IMO, ECO, PLL, FLL)	打开	打开	打开	打开	关闭	关闭	关闭	关闭
LVD	打开	打开	打开	打开	关闭	关闭	关闭	关闭
ILO	打开	打开	打开	打开	打开	打开	关闭	关闭
外设								
SMIF	打开	打开	打开	打开	保留	关闭	关闭	关闭
UDB	打开	打开	打开	打开	关闭	关闭	关闭	关闭
SAR ADC	打开	打开	打开	打开	关闭	关闭	关闭	关闭
CTBm	打开	打开	打开	打开	打开 (低 GBW)[3]	关闭	关闭	关闭
LPCMP	打开	打开	打开	打开	打开 [3]	打开[4]	关闭	关闭
TCPWM	打开	打开	打开	打开	关闭	关闭	关闭	关闭
CSD	打开	打开	打开	打开	保留	关闭	关闭	关闭
BLE	打开	打开	打开	打开	保留	关闭	关闭	关闭
LCD	打开	打开	打开	打开	打开	关闭	关闭	关闭
SCB	打开	打开	打开	打开	保留(I^2C/SPI 唤醒可用)[5]	关闭	关闭	关闭
GPIO	打开	打开	打开	打开	打开	冻结	关闭	关闭
看门狗定时器	打开	打开	打开	打开	打开	打开	关闭	关闭
多计数器 WDT	打开	打开	打开	打开	打开	关闭	关闭	关闭
复位								
XRES	打开	打开	打开	打开	打开	打开	打开	关闭
POR	打开	打开	打开	打开	打开	打开	关闭	关闭
BOD	打开	打开	打开	打开	打开	关闭	关闭	关闭
看门狗复位	打开	打开	打开	打开	打开	打开[6]	关闭	关闭
备份域								
WCO, RTC, 告警	打开	打开	打开	打开	打开	打开	打开	打开

[3] 可以选择在系统深度睡眠模式下启用低功耗比较器和 CTBm 以产生唤醒。

[4] 可以选择在休眠模式下启用低功耗比较器以产生唤醒。

[5] 在系统深度睡眠功耗模式下，只有一个支持深度睡眠的 SCB 可用; 其他 SCB 在系统深度睡眠功耗模式下不可用。

[6] 看门狗中断可以产生休眠唤醒。 有关详细信息，请参见技术参考手册的“看门狗定时器”一章。

Appendix C. 回调函数示例

C.1 寄存器回调函数

```
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 执行自定义回调函数

```
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. 代码示例

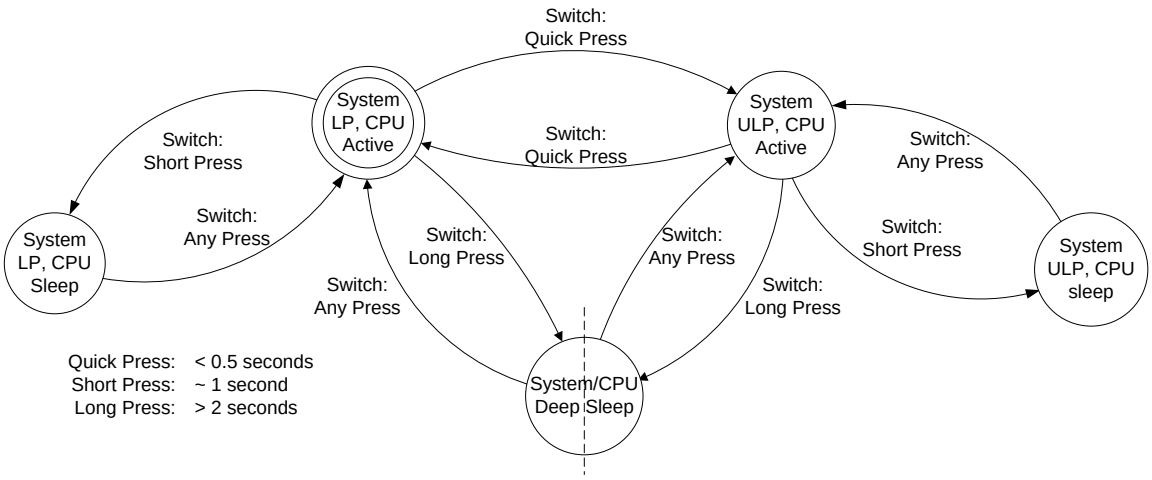
本应用笔记中包含以下代码示例，用以演示 PSoC 6 MCU 功耗模式和功耗降低技术。

D.1 CE219881 - PSoC 6 MCU 切换功耗模式

此代码示例演示如何进入和退出系统 LP 和 ULP 功耗模式，以及如何将 CPU 从 CPU 激活转换为睡眠或深度睡眠。一旦处于任一模式，该示例还显示如何唤醒并返回到系统 LP 或 ULP 模式和 CPU 活动模式之一。

该项目使用开关在功耗模式之间转换，并显示不同的 LED 颜色以指示当前的功耗模式。图 21 显示了在固件中实现以执行转换的状态机。有关更多信息，请参阅 CE219881 - PSoC 6 MCU Switching between Power Modes。

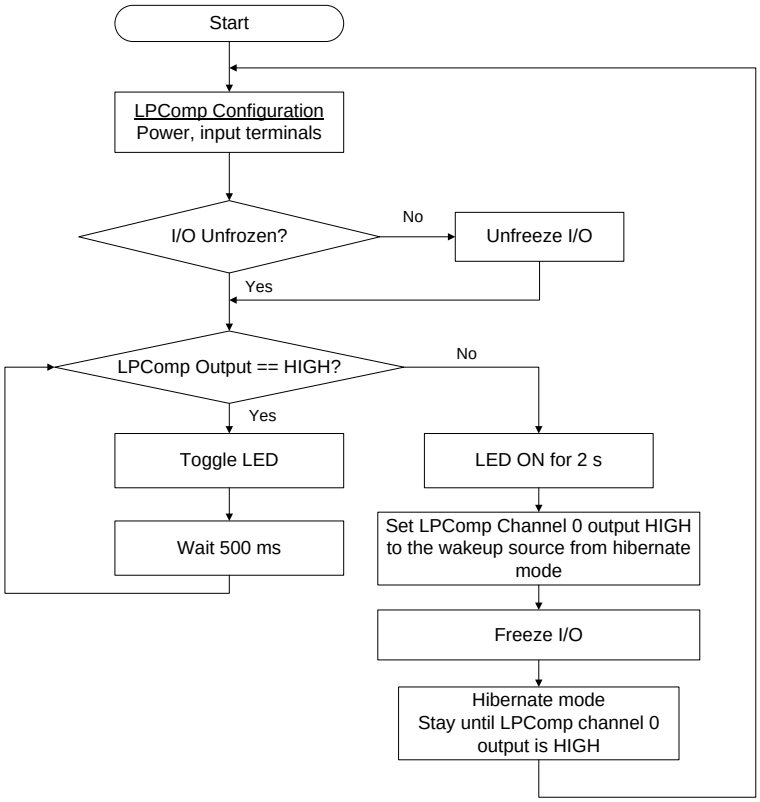
图 21.功耗模式状态机



D.2 CE218129 - 使用低功耗比较器从休眠状态唤醒 PSoC 6 MCU

此代码示例演示如何为 LPComp 内部参考电压设置 Component 选项，以及如何使用 LPComp 驱动程序设置 GPIO 的外部输入。该项目是系统休眠功耗模式转换的一个很好的例子。它教你如何在系统休眠之前和之后处理 GPIO，并且展示了如何为休眠注册唤醒源。图 22 显示了项目的基本流程。有关更多信息，请参阅 CE218129 - PSoC 6 MCU Wakeup from Hibernate Using a Low-Power Comparator。

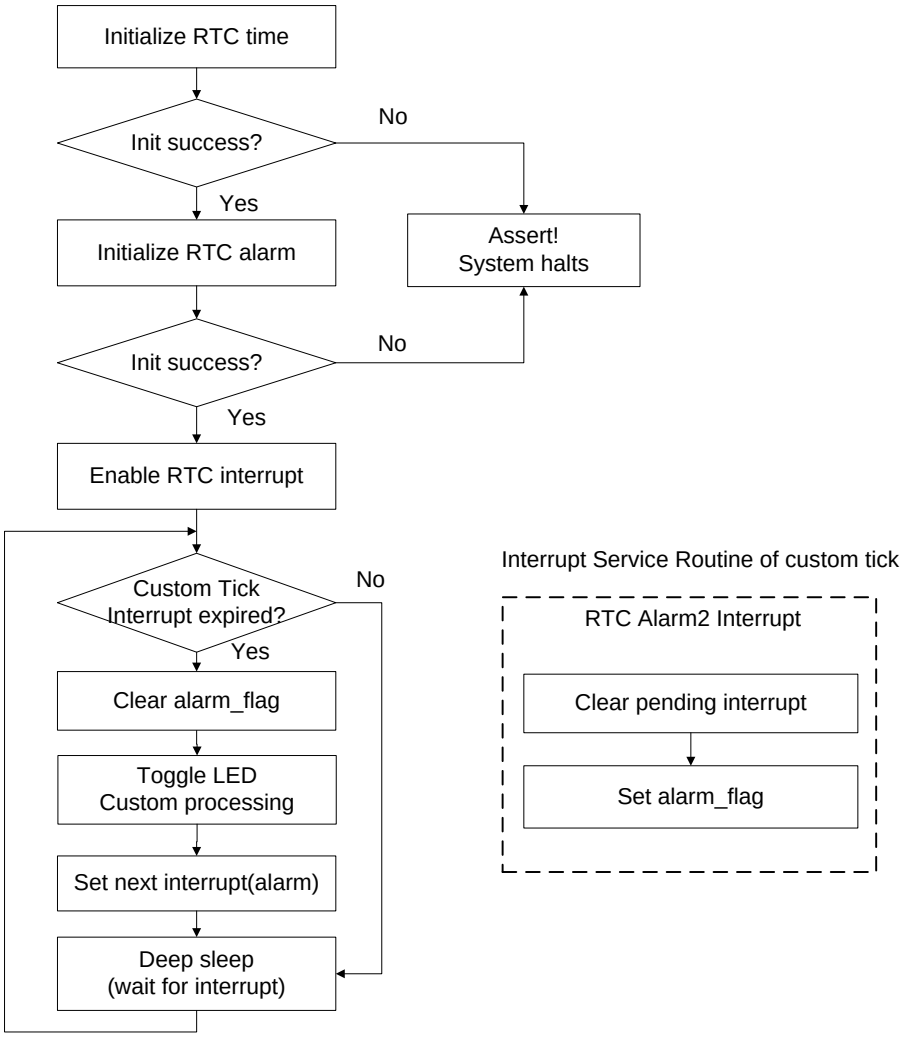
图 22.使用 LPComp 输入从系统休眠模式唤醒



D.3 CE218542 - 使用 RTC 报警中断的 PSoC 6 MCU 自定义滴答定时器

此代码示例演示如何使用 PDL RTC 驱动程序 API 为定期警报中断配置 RTC 寄存器。该项目使用系统 LP 和系统深度睡眠模式来节省功耗。包含 GPIO 输出以切换 LED 以显示中断周期。有关更多信息，请参阅 CE218542 - PSoC 6 MCU Custom Tick Timer Using RTC Alarm Interrupt.

图 23. 使用报警中断的 RTC 周期性唤醒定时器



D.4 CE226306 - PSoC 6 MCU 功率测量

本示例展示了如何配置 PSoC 6 MCU 器件，使其运行 PSoC 6 MCU 数据表中 "器件级别规格 "表中指定的时钟频率和系统模式。在 PSoC 6 MCU 器件中构建和编程应用程序后，您可以测量 PSoC 6 MCU 器件消耗的电流，并将其与数据表中指定的值进行比较。您可以通过更改固件中的#define 来选择功率配置。

表 7. 配置选项

配置	选项	说明

系统模式	SYSTEM_LP
	SYSTEM_ULP
		定义固件进入的系统模式--系统低功耗或系统超低功耗模式。使用以下函数：
		`Cy_SysPm_SystemEnterLp(); // Enter System Low Power mode`
		`Cy_SysPm_SystemEnterUlp(); // Enter System Ultra-Low Power mode`

核心电压供电	VCCD_1V1
	VCCD_0V9
		定义 BUCK 或 LDO 的核心电压电源--0.9 V 或 1.1 V。
		`Cy_SysPm_SwitchToSimoBuck();`
		`Cy_SysPm_SimoBuckSetVoltage1(CY_SYSPM_SIMO_BUCK_OUT1_VOLTAGE_1_1V);`

缓存支持	RUN_FROM_FLASH
	RUN_FROM_CACHE
		定义是否使用缓存。如果禁用缓存，则使用以下指令。
		`//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);`
		对 `Cy_SysLib_SetWaitStates()` 函数的调用是基于对 CACHE 的支持。如果禁用 CACHE，则设置给定系统模式的最大允许频率。如果启用了 CACHE，则根据 CM4 的 CPU 频率进行设置。

CM4 CPU 模式	CM4_WHILE_LOOP
	CM4_DHRYSTONE
	CM4_SLEEP
	CM4_DEEP_SLEEP
		定义 CM4 CPU 运行什么- While(1)循环，Dhrystone 算法，进入 CPU 睡眠，或进入 CPU 深度睡眠。使用以下函数:
		`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 频率 [CM4_FREQ_MHZ]	FREQ_8_MHZ
	FREQ_25_MHZ
	FREQ_50_MHZ
	FREQ_100_MHZ
		定义 HFCLK0 的时钟频率，它与 CM4 CPU 时钟相连。使用以下函数。
		`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 模式	CM0P_WHILE_LOOP CM0P_DHRYSTONE CM0P_SLEEP CM0P_DEEP_SLEEP CM0P_HIBERNATE	定义 CM0+ CPU 运行什么- While(1)循环，Dhrystone 算法，进入 CPU 睡眠，进入 CPU 深度睡眠或休眠 使用与 CM4 CPU 模式相同的函数： `Cy_SysPm_SystemEnterHibernate(); // Go to Hibernate`

CM0+ CPU 频率 [CM0P_FREQ_MHZ]	FREQ_8_MHZ
	FREQ_25_MHZ
	FREQ_50_MHZ
	FREQ_100_MHZ
		定义 CM0+CPU 时钟的时钟频率。使用以下函数：
		`/* Set the PERI Clock Divider */`
		`Cy_SysClk_ClkPeriSetDivider((CM4_FREQ_MHZ/CM0P_FREQ_MHZ)-1);`

时钟源	USE_IMO
	USE_FLL
		定义时钟源的 HCLK0 - IMO 或 FLL。如果使用 IMO, 旁路 FLL。如果使用了 FLL, IMO 仍然用作 FLL 的源。
		使用以下函数:
		`Cy_SysClk_ClkPathSetSource(0UL, CY_SYSCLK_CLKPATH_IN_IMO);`
		`Cy_SysClk_ClkHfSetSource(0UL, CY_SYSCLK_CLKHF_IN_CLKPATH0);`

最小稳压器电流模式	MIN_CURRENT	决定固件是否调用了 `Cy_SysPm_SystemSetMinRegulatorCurrent()` 函数。

文档修订记录

文档标题: AN219528 - PSoC 6 MCU 的低功耗模式以及降低功耗技术

文档编号: 002-28820

版本	ECN	提交日期	变更说明
**	6725654	11/06/2019	本文档版本号为 Rev. **，译自英文版 002-19528 Rev. *A。
*A	7019359	11/06/2020	本文档版本号为 Rev. *A，译自英文版 002-19528 Rev. *B。

全球销售和设计支持

赛普拉斯公司具有一个由办事处、解决方案中心、厂商代表和经销商组成的全球性网络。要想查找离您最近的办事处，请访问赛普拉斯所在地。

产品

Arm® Cortex®微控制器	cypress.com/arm
汽车级产品	cypress.com/automotive
时钟与缓冲器	cypress.com/clocks
接口	cypress.com/interface
物联网	cypress.com/iot
存储器	cypress.com/memory
微控制器	cypress.com/mcu
PSoC	cypress.com/psoc
电源管理 IC	cypress.com/pmic
触摸感应	cypress.com/touch
USB 控制器	cypress.com/usb
无线连接	cypress.com/wireless

PSoC®解决方案

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

赛普拉斯开发者社区

社区 | 代码示例 | 项目 | 视频 | 博客 | 培训 | 组件

技术支持

cypress.com/support

Cypress Semiconductor
An Infineon Technologies Company
198 Champion Court
San Jose, CA 95134-1709