Freescale Semiconductor
应用笔记

Document Number: AN4503
Rev 2, 04/2015

Kinetis MCU 功耗管理

何时以及如何使用 Kinetis 低功耗模式

内容

1 简介.....1
2 功耗管理技术.....3
3 Kinetis SDK 快速入门.....17
4 快速启动.....23
5 复位管理.....25
6 动态和静态功耗管理.....29
7 低功耗模式下的时钟操作.....29
8 功耗模式转换.....33
9 功耗模式进入代码.....35
10 功耗模式退出转换.....47
11 不同功耗模式下的模块.....49
12 使用外部存储器和外设 - DDR 存储器控制器和 DDR 低功耗模式用例.....56
13 功率测量.....58
14 LLWU 引脚和模块唤醒.....61
15 参考文献和修订历史记录.....63

1 简介

各种应用都追求在有限的能耗预算内实现高性能。日益提高的系统级别要求不允许降低性能，而是推动其往低能耗预算方面发展，以延长产品的使用时间。Kinetis 微控制器系列产品具有内部功耗管理功能，该功能可用于控制微控制器的电量使用并帮助达成嵌入式设计的目标。

本应用笔记讨论了如何使用功耗管理系统，提供了相关用例并介绍了这些用例的实时电流测量结果。本版本的其中一个章节介绍了如何使用 DDR 存储器控制器维持应用的低功耗特性。

此外，本版本还讨论了不同微控制器上功耗管理功能之间的区别，以及用于展示这些低功耗状态的驱动程序。同时还提供了每种功耗模式的使用技巧。

此处讨论的功耗管理方法不包含与时钟发生器模块相关的详细信息。有关时钟模式的介绍以及多用途时钟发生器（MCG）缩略语的说明，请参阅 MCU 参考手册。本应用笔记重点介绍了功耗管理控制器（PMC）、低漏电唤醒单元（LLWU）、复位控制模块（RCM）以及系统模式控制器（SMC）。

以下网址提供了关于 Kinetis 上低功耗特性的所有信息，一切应有尽有：http://www.freescale.com/lowpower。

© 2015 Freescale Semiconductor, Inc.

1.1 系统模式控制器修订 - MC 和 SMC

模式控制通过 MCU 中的状态机实现，该状态机称为模式控制器或系统模式控制器（SMC）。

本应用笔记中的代码与 Kinetis 器件兼容，但存在一些差别。模式控制器模块已经随着时间的推移而不断发展演变，其中添加了新的低功耗模式和新的运行模式，并保持了模式进入和退出兼容。在本应用笔记中，MC1 是指 Kinetis Rev 1.x 100 MHz 器件上的模式控制器模块（MC）。术语 MC2 - MC4 是指后续 Kinetis MCU 上的系统模式控制器模块（SMC）。推出 Kinetis L 后，创造了 MC3 (SMC)。Kinetis E 中引入了模式控制器版本 4，其中低功耗模式数量有所减少。2014 年推出 Kinetis 器件（包括 K64、K22FN 和 KV）时，引入了具有高速运行功耗模式的 MC5(SMC)。请参见下表，了解概述。

表 1. Kinetis 模式控制器版本

模式控制器版本	Kinetis 系列	说明
模式控制器 v1	K40、K60- 100 MHz	Kinetis Rev 1.x 100 MHz 器件，带 VBAT 域和低功耗 OSC
SMC1 模式控制器 v2	Kinetis L 48 MHz	添加了 VLLS0 功耗模式，无单独的 VBAT 域
SMC2 模式控制器 v3	Kinetis K 72 MHz、K64/K24 120 MHz	存储器更大，M4 内核，添加了 LLS2、LLS3
SMC3 模式控制器 v4	Kinetis E	仅具有运行、等待、停止模式
SMC4 模式控制器 v5	K65、K22FN、KV40 - 部分高达 180MHz	Flash 和 RAM 更大，部分包含高速运行模式

1.2 操作序列化确保操作顺序正确

考虑以下代码段:

```
SCB_SCR |= SCB_SCR_SLEEPDEEP_MASK;
SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK;
SMC_PMCTRL |=  SMC_PMCTRL_STOPM(0x3);
asm("WFI");
```

在上个寄存器写入操作完成之前，ARM Cortex-M 架构不会延迟执行 WFI 指令。因此，在完成上个模式控制器寄存器写入操作之前，内核就可能向模式控制器设置 SLEEP 输出。如果出现此情况，MCU 可能会进入错误的低功耗模式。

Kinetis MCU 至少需要 3 个时钟，才能完成对大多数外设的写入操作。对于像 SMC 这样在低功耗模式下保持通电状态的外设，至少需要 4 个时钟才能将数据写入 SMC 寄存器。这里假定通过交叉开关矩阵时没有延迟。然而，WFI 只需 1 个时钟即可设置 SLEEP/SLEEPDEEP 输出。

```
volatile unsigned int dummyread;
SCB_SCR |= SCB_SCR_SLEEPDEEP_MASK;
SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK;
SMC_PMCTRL |=  SMC_PMCTRL_STOPM(0x3);
dummyread = SMC_PMCTRL;   /* read-after-write sequence */
dummyread++;   /* not required but added this statement to eliminate the compile
warning */
asm("WFI");
```

避免此问题的最简单（且最保险）方法是，在写入控制寄存器后紧跟着读取同一寄存器，此例中为 SMC_PMCTRL 寄存器。先写后读序列保证在读取之前完成写入操作，从而确保在执行 WFI 指令之前锁存正确的低功耗模式。

这种外设寄存器的先写后读序列化序列适用于所有实时序列控制。为了保持软件结构的一致性，对于所有控制 MCU 来说，外设寄存器先写后读序列是一种完全可接受的存储器序列化方法。

警告：如果启动编译优化功能以尝试改善代码，则此类代码序列可能会被优化。

2 功耗管理技术

2.1 应用设计简介

设计团队发明的术语“追求纳瓦”是指不懈追求降低芯片中每个电路的功耗以确保电路达到最低功耗的过程。

下面详细介绍了此应用设计过程中所关注的一些重要领域，通过该设计过程，Kinetis MCU 实现了超低功耗性能。

此图所示为“减小能耗曲线下方面积”的思路。

在低功耗至关重要的嵌入式系统中，完成应用所需的多项任务必须尽可能节能。这些任务通常可概括为 3 个阶段：

- 初始化阶段
- 控制阶段，其中可能包括数据收集、通信和控制
- 计算阶段

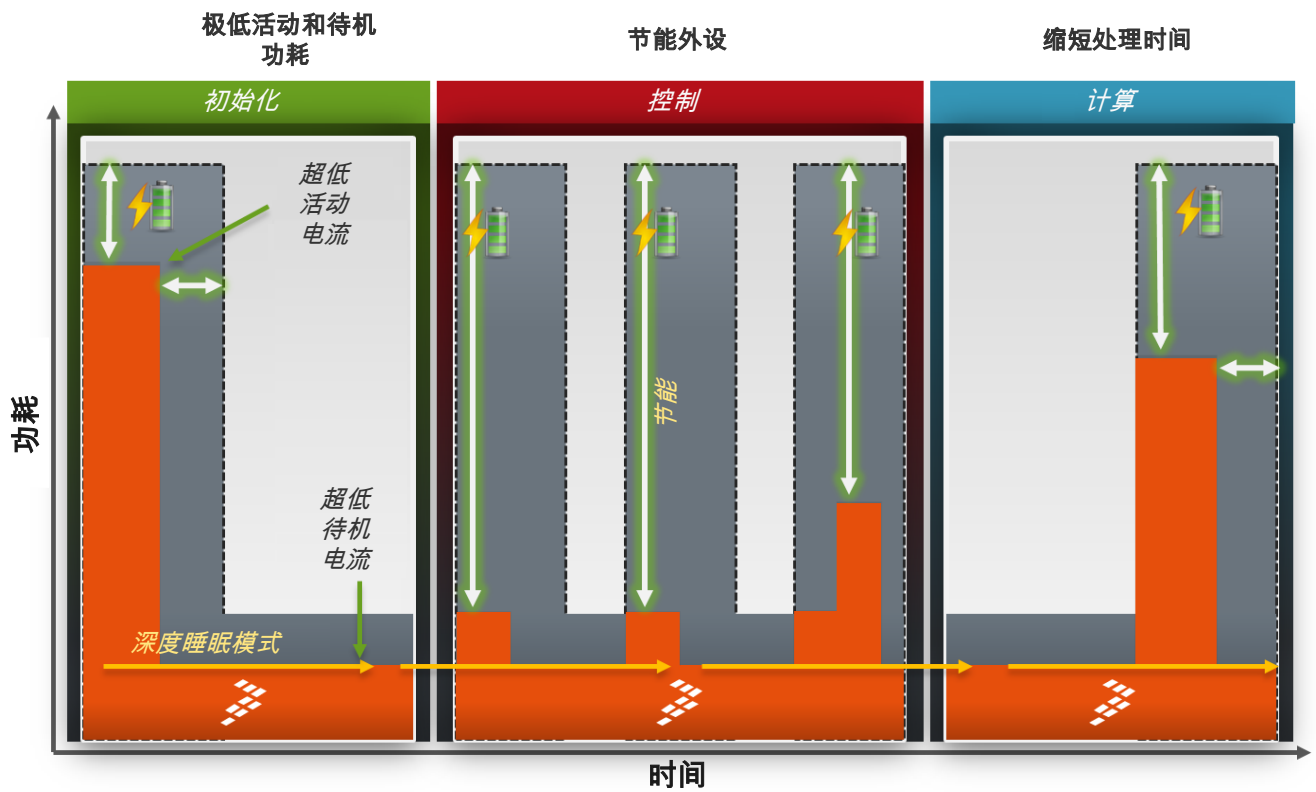


图 1. 能效：能耗 = 功率 x 时间

要降低这三个阶段中的能耗，等式非常简单；以较低的功率在更短时间内完成更多任务（能耗 = 功率 x 时间）。实现方式是在所有阶段实现低功耗，而不只是在其中一个阶段。您必须具有极低活动和待机功耗。

Kinetis MCU 可以在运行模式和速度降低的极低功耗运行（VLPR）模式下实现这两个目标。您可以查阅 MCU 数据手册（DS）的“功耗特性”章节来了解代表您的用例的规格。DS 中提供的测试描述介绍了使用控制因素降低应用功耗时的预期电流。此处仅列出了部分控制因素：

- 频率
- 时钟源
- 电压
- 外设时钟选通
- 从 Flash 或 RAM 执行
- 使用计算模式
- 深度睡眠模式

使用 DS 中的数据为您的低功耗设计工作提供指引。

您可以查看图 1 中的图形，了解 Kinetis L 和 K2 系列能耗曲线（以橙色表示）与其他竞争产品能耗曲线（以灰色表示）的节能效果比较。

其次，您必须拥有足够智能的节能外设，甚至无需唤醒 CPU，即可收集、处理和存储数据。其他产品可能需要唤醒进入完全运行模式，才能激活外设并在稍后完成数据收集阶段，再返回至深度睡眠模式。

对于 Kinetis MCU，数据收集阶段从深度睡眠模式开始，并显示由低功耗定时器触发的 3 个周期性事件。该定时器触发启动低功耗 ADC 转换，在此过程中会使用 ADC 中的内置比较功能将结果与预编程阈值进行比较。

借助此类功能，当值超出所需结果范围时，则无需存储结果。请注意，前两个事件不会触发结果存储。但是，最后一个事件会触发结果存储，但不会唤醒 CPU 存储数据，此时我们会看到一个小得多的能耗峰值，因为 Kinetis 节能外设支持异步 DMA 唤醒功能，该功能可以在 CPU 仍处于睡眠状态时将 ADC 结果存储到 SRAM 中，以供后续处理。DMA 传输完成之后，MCU 将自动返回至深度睡眠模式。使用低功耗 UART 收集到充足数据或传输完数据之后，您可以唤醒 CPU 并开始进入计算阶段。

这只是可用节能外设的一个使用示例。

最后，您必须缩短计算阶段的处理时间，以便返回深度睡眠模式并再次开始整个序列。在 Kinetis MCU 上可以实现这点，这得益于其超高效 Cortex 内核和节能架构的组合，节能架构包括允许 DMA 和 CPU 并行访问从机外设的交叉开关矩阵、可简化面向位的任务的位操作引擎，以及可消除 Flash 访问中的等待状态的 Flash 存储器控制器。以下几个段落将分别详细介绍这些概念。

2.2 节能外设

外设	低功耗功能
DMA	允许节能外设（例如ADC、UART和定时器/PWM）触发异步DMA请求（在STOP/VLPS模式下），以便在无CPU干预的情况下执行DMA传输和返回当前功耗模式
UART	低至STOP/VLPS模式下，支持由总线时钟支持的通信的异步传输和接收操作。可配置的接收器波特率过采样率为4x至32x，从而允许通过更低时钟源实现更高波特率
SPI	低至STOP/VLPS模式下，支持从机模式地址匹配唤醒和第一个消息捕获功能
I2C	低至STOP/VLPS模式下，支持多地址匹配唤醒功能
USB	低至STOP/VLPS模式下，支持通过恢复信号进行异步唤醒
LPTPM（定时器/PWM）	低至STOP/VLPS模式下，支持16位定时器输入捕获、输出比较和PWM功能
LPTMR（定时器/脉冲定时器）	在所有功耗模式下，支持16位定时器和脉冲计数器功能
RTC	在所有功耗模式下（包括温度和电压补偿），支持带秒钟中断和可编程闹钟的32位秒钟计数器

图 2. 节能外设 1

UART0 – 可在 VLPS 模式下接收和发送。可在内核关断时使用 DMA 接收数据并存储到缓冲区，反之亦然。地址匹配功能使得无需采取任何操作，即可接收和忽略字符。例如，在联网烟雾警报系统中，总线上可以存在多个节点，而中央系统可单独对每个探测器进行寻址以检查其状态。此外，UART 支持单线系统。另外还可在 VLPS 模式下接收第一个字符。

SPI 和 I2C 支持地址匹配唤醒功能。USB 支持在收到恢复信号时唤醒。

LPTPM – 低至 VLPS 模式时仍支持定时器、IC、OC 和 PWM。对自定义串行协议有用。在控制电压可从 ADC 读数不定期更新的应用（光传感器控制照明亮度）中，可在 VLPS 模式下通过 DMA 修改 PWM 输出。

低至 VLLS1 模式下 LPTMR 仍支持定时器工作 – 唤醒事件和系统“节拍”功能。常用于触发事件（ADC 读取操作等）或 DMA 传输。还可在低至 VLLS1 模式下提供脉冲计数功能 – 对流量计应用非常有用，通过传感器进行脉冲计数，唤醒几千个脉冲，否则始终处于 VLLS1 模式。

RTC – 对于不带专用 RTC 振荡器的器件，采用 32 KHz 极低功耗晶振时，这可在低至 VLLS1 模式下提供低于 1 uA 的低功耗计时功能和唤醒功能，而在使用外部方波时钟时则可在低至 VLLS0 模式下提供这些功能。

请注意，目前仅 Kinetis L 以及新款 K 系列器件可在低至 VLPS 模式下支持 DMA 以及部分外设的异步操作。

外设	低功耗功能
ADC	低至STOP/VLPS模式下，支持将单次转换存储到多个结果寄存器和硬件均值和自动比较模式
CMP（模拟比较器）	在所有功耗模式下，支持阈值交叉检测和触发式的比较模式以实现更低平均功耗比较
DAC	在所有功耗模式下支持静态参考
段式LCD	在所有功耗模式下支持交替显示和闪烁功能
TSI（电容触摸感应接口）	在所有功耗模式下支持单通道的唤醒电容触摸
LLWU（低漏电唤醒单元）	在LLS和VLLSx模式下支持8个唤醒引脚、REFET和NMI唤醒引脚以及某些节能外设

图 3. 节能外设 2

ADC 仍可通过节能外设在硬件均值和自动比较模式下支持单次转换。例如，在恒温器中，MCU 可以处于低功耗模式，由 LPTMR 定期触发 ADC 读取操作，并且仅当温度超过某些阈值时唤醒 MCU。

模拟比较器具有阈值交叉检测功能。此外，它还具有触发器模式，在该模式下，LPTMR 可以长时间打开 CMP 以进行比较，然后再次将其关断。

DAC 支持静态参考。这可在 VLPS 模式下通过 DMA 进行更新。

段式 LCD 具有内置闪烁模式和交替闪烁模式，内置闪烁模式无需 CPU 干预，交替闪烁模式可 CPU 干预频率降低至一半。这些可以降低功耗。低至 VLLS1 模式下仍可工作。

电容触摸感应接口支持单通道唤醒触摸。

2.3 功耗模式的架构概览

传统内核和其他嵌入式系统的典型功耗模式为运行、等待和停止。ARM® Cortex™-M4 和 M0+的功耗模式为运行、睡眠和深度睡眠。扩展功耗模式及其与典型和 ARM Cortex-M4 和 M0+内核之间的关系如图 4 所示。

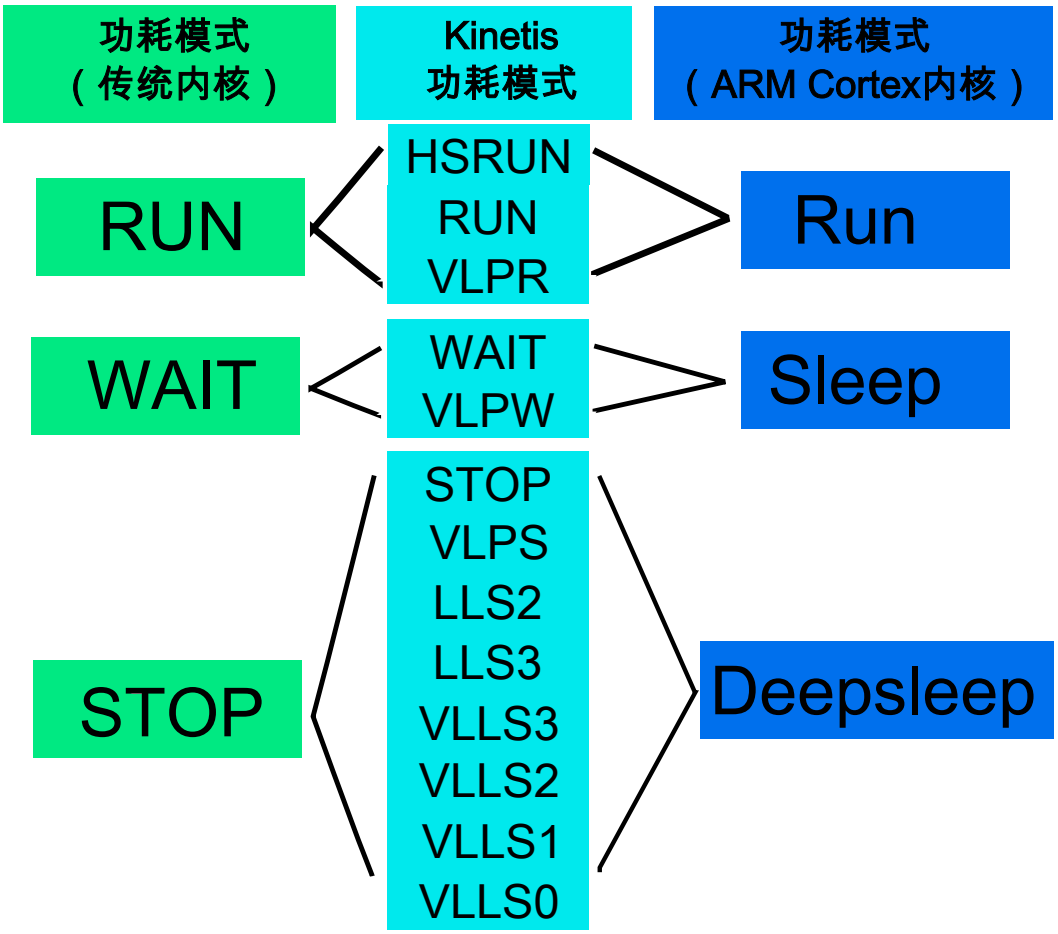


图 4. 功耗模式比较

2.3.1 ARM Cortex-M4 和 M0+低功耗的实现

ARM Cortex-M4 和 M0+内核具有三种主要的操作模式：运行、睡眠和深度睡眠。在 Kinetis 上，内核使用中断唤醒指令（WFI）来启用睡眠和深度睡眠模式。

图 5 介绍了 Cortex -M4 和 M0 的低功耗的实现。睡眠和深度睡眠状态构建在硬件层面上，用于控制内核和中断控制器的时钟。进入睡眠模式后，NVIC 逻辑保持活动状态，并且通过中断或复位将内核从睡眠模式唤醒。进入深度睡眠模式后，则利用异步唤醒中断控制器 (AWIC) 从一组挑选出来的源唤醒 MCU。这些源会在低漏电唤醒单元 (LLWU) 模块进行描述。

借助 Sleep-on-exit 功能，ARM Cortex 内核还可通过另外一种方式进入低功耗模式。在 Cortex 处理器的系统控制模块中有一个寄存器，称为系统控制寄存器（SCR），其中包含多个与睡眠操作相关的控制位。将控制位 SLEEPONEXIT 置 1 时将使能中断驱动的应用程序，用于避免在每个唤醒事件之后返回至主应用程序。如果 SLEEPONEXIT 已使能，MCU 可在完成所有挂起异常处理程序（中断服务程序）之后立即进入低功耗模式。这就像在异常退出后立即执行 WFI。Sleep-on-exit 功能减少了不必要的压栈和出栈操作。

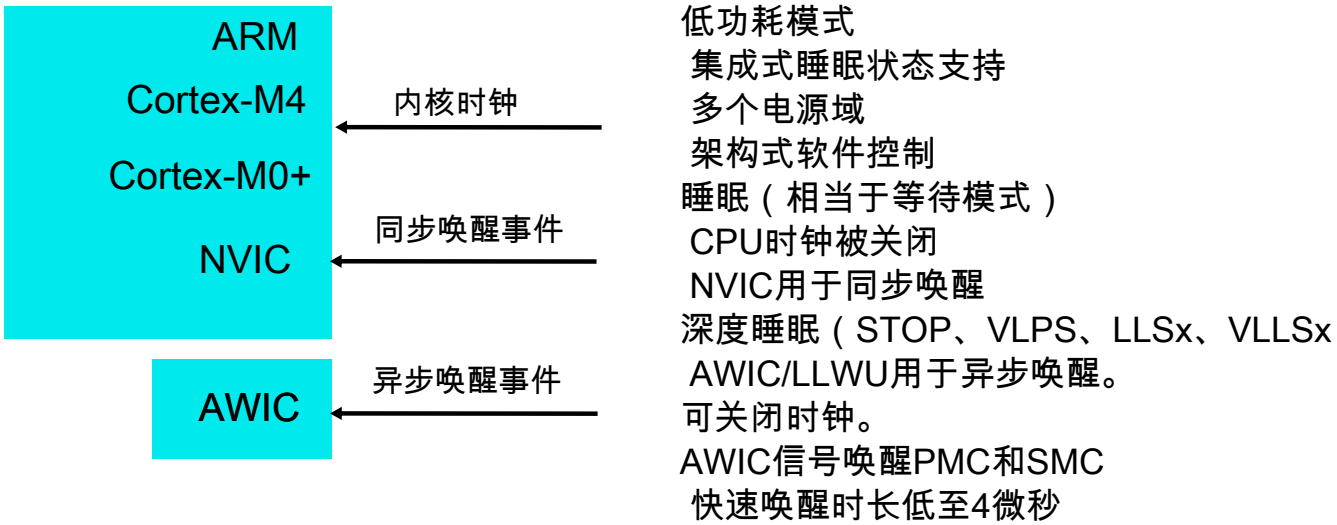


图 5. ARM Cortex-M4 和 M0+功耗模式

2.3.2 模式描述

表 2 介绍了每种功耗模式，其中包含模式的相关详细信息以及模式进入和退出的基本信息。Kinetis MCU 可能具有所有这些或其中部分低功耗模式。部分产品重提传统的运行、等待和停止低功耗模式。有些产品没有 LLWU 模块。有关受支持的具体模式列表，请参见《MCU 参考手册》。有关更多信息，包括阻止进入低功耗模式的因素，请参阅参考手册中的功耗模式转换章节。以下给出的测量数据、频率和其他极限数据仅供参考。请确保使用 MCU 数据手册中的正式值。

下表介绍了大多数 Kinetis 器件可用的功耗模式。有关芯片特定模式的详情，请参见各器件的参考手册。

表 2. 芯片功耗模式

芯片模式	说明
正常运行	MCU 可全速运行且内部电源充分调节，也就是说，处于运行调节状态。此模式也称为正常运行模式。
高速运行	允许芯片实现最佳性能。与运行模式相比，MCU 的运行频率更高且内部电源充分调节。有关最大允许频率的详情，请参见“功耗管理”章节。
通过 WFI 实现正常等待	内核处于睡眠模式时允许外设运行以降低功耗。NVIC 依然监测中断；继续提供外设时钟。维持运行调节。
通过 WFI 实现正常停止	将芯片置于静态。是既能保持所有寄存器，同时还能维持 LVD 保护的最低功耗模式。NVIC 禁用；AWIC 用于从中断唤醒；外设时钟停止。来自支持外设的所有停止确认信号均有效之后，其他主机的系统时钟以及总线时钟将关闭。内部电压调节器处于运行调节状态。
VLPR（超低功耗运行）	片内电压调节器处于低功耗模式，仅提供芯片低频运行所需的电压。降频 Flash 访问模式（1 MHz）；LVD 关闭；内部振荡器为内核、总线以及外设时钟提供低功耗 4 MHz 源。
通过 WFI 实现 VLPW（超低功耗等待）	与 VLPR 相同，但内核处于睡眠模式下，以便进一步降低功耗；NVIC 仍然对中断敏感（FCLK 开启）。片内电压调节器处于低功耗模式，仅提供芯片低频运行所需的电压。
通过 WFI 实现 VLPS（超低功耗停止）	将芯片置于静态，关闭 LVD 操作。ADC 和引脚中断可以工作的最低功耗模式。外设时钟停止，但 LPTimer、RTC、CMP、TSI、DAC 可以使用。NVIC 禁用（FCLK 关闭）；AWIC 用来从中断唤醒。片内电压调节器处于低功耗模式，仅提供芯片低频运行所需的电压。所有 SRAM 均处于工作状态（保留内容，保持 I/O 状态）。
LLS（低漏电停止）	状态保持功耗模式。大多数外设都处于状态保持模式（时钟停止），但 LLWU、LPTimer、RTC、CMP、TSI、DAC 可以使用。NVIC 禁用；LLWU 用来唤醒。

下一页继续介绍此表...

表 2. 芯片功耗模式 (继续)

芯片模式	说明
	附注： LLWU 中断不可通过中断控制器屏蔽，以避免系统无法在 LLS 恢复时完全退出停止模式的现象。所有 SRAM 均处于工作状态（保留内容，保持 I/O 状态）。
LLS3（低漏电停止 3）	状态保持功耗模式。大多数外设都处于状态保持模式（时钟停止），但 LLWU、LPTimer、RTC、CMP、TSI、DAC 可以使用。NVIC 禁用；LLWU 用来唤醒。 附注： LLWU 中断不可通过中断控制器屏蔽，以避免系统无法在 LLS 恢复时完全退出停止模式的现象。所有 SRAM 均处于工作状态（保留内容，保持 I/O 状态）。
LLS2（低漏电停止 2）	状态保持功耗模式。大多数外设都处于状态保持模式（时钟停止），但 LLWU、LPTimer、RTC、CMP、TSI、DAC 可以使用。NVIC 禁用；LLWU 用来唤醒。 附注： LLWU 中断不可通过中断控制器屏蔽，以避免系统无法在 LLS 恢复时完全退出停止模式的现象。SRAM_U 的一部分保持上电状态（保留内容，保持 I/O 状态）。RAM2 分区。 通过降低内部逻辑的电压，同时关断系统 RAM3 分区，MCU 进入低漏电模式。可使用 STOPCTRL[RAM2PO]选择性地保留系统 RAM2 分区。系统 RAM1 分区、内部逻辑和 I/O 状态保留。
VLLS3（极低漏电停止 3）	大多数外设禁用（时钟停止），但 LLWU、LPTimer、RTC、CMP、TSI、DAC 可以使用。NVIC 禁用；LLWU 用来唤醒。SRAM_U 和 SRAM_L 保持上电状态（内容保留，保持 I/O 状态）。
VLLS2（极低漏电停止 2）	大多数外设禁用（时钟停止），但 LLWU、LPTimer、RTC、CMP、TSI、DAC 可以使用。NVIC 禁用；LLWU 用来唤醒。SRAM_L 断电。SRAM_U 的一部分保持上电状态（保留内容，保持 I/O 状态）。可使用 STOPCTRL[RAM2PO]选择性地保留系统 RAM2 分区。在此模式下，会保留系统 RAM1 分区内容。内部逻辑状态不会保留。
VLLS1（极低漏电停止 1）	大多数外设禁用（时钟停止），但 LLWU、LPTimer、RTC、CMP、TSI、DAC 可以使用。NVIC 禁用；LLWU 用来唤醒。所有 SRAM_U 和 SRAM_L 均断电。32 字节系统寄存器文件和 128 字节 VBAT 寄存器文件保持上电，以便保存客户关键型数据。保持 I/O 状态。内部逻辑状态不会保留。
VLLS0（极低漏电停止 0）	大多数外设禁用（时钟停止），但 LLWU 和 RTC 可以使用。NVIC 禁用；LLWU 用来唤醒。所有 SRAM_U 和 SRAM_L 均断电。32 字节系统寄存器文件和 128 字节 VBAT 寄存器文件保持上电，以便保存客户关键型数据。保持 I/O 状态。内部逻辑状态不会保留。1kHz LPO 时钟禁用，并且可使用 CTRL[PORPO]选择性地使能上电复位（POR）电路。
BAT（仅备用电池）	除了 VBAT 电源外，此芯片将掉电。RTC 和 128 字节 VBAT 寄存器文件保持上电，以便保存客户关键型数据。

2.3.3 运行模式

- 任何复位之后均会选择此模式，除非由 FOPT 控制选择 LPRUN。
- 片内电压调节器开启，全功能，除非由 FOPT 控制选择 LPRUN。
- 堆栈指针（SP）、程序计数器（PC）和链路寄存器均被设置。
- ARM Cortex-M4 和 M0+处理器退出复位并读取起始 SP。
- ARM Cortex-M4 和 M0+处理器从向量表读取起始 PC。
- 如果从 VLLSx 恢复通过复位退出，则 OSC 和输出引脚会保持状态，直到对 ACKISO 执行写操作。
- 如果在复位失效时 NMI 为低电平，则在执行起始 PC 处的指令之前，将对 NMI 向量执行异常处理。

2.3.4 高速运行模式

- 片内电压调节器处于高速运行模式。
- 通过写入 RUNM 控制位来选择。
- 不允许进行 Flash 编程和擦除操作。
- 写入或复位 RUNM 位时，将退出 HSRUN 并进入运行模式。
- 如未先进入运行模式，则无法进入低功耗模式。

2.3.5 超低功耗运行模式（VLPR）

- 可从运行模式进入 VLPR，或 FOPT 中的 LPBOOT 位置位时由复位进入 VLPR。
- 片内电压调节器处于仅提供 MCU 低频运行所需电压的模式。
- 内核和总线频率限制在 4 MHz 以内。
- Flash 频率限制在 800 kHz 和 1 MHz 之间。
- 不允许更改时钟。
- 不允许进行 Flash 编程和擦除操作。
- 不允许进行 Flex 存储器（EEPROM）编程操作。
- 可使用内部快速 IRC 或 LIRC（使用 MCG-Lite 时）进入 VLPR。
- 可使用外部时钟进入 VLPR。
- 可直接进入除了正常停止和等待之外的所有低功耗模式。
- 可通过写入 RUNM 控制位或（在某些器件上）通过中断退出 VLPR。

2.3.6 等待模式

- ARM Cortex-M4 和 M0+内核进入睡眠模式。
- ARM Cortex-M4 和 M0+内核时钟被关闭。
- NVIC 仍然对中断敏感。
- 继续为外设提供时钟。
- 清零 SCGCx 寄存器中的时钟关闭位以降低功耗。
- 不允许进行 Flex 存储器（EEPROM）编程操作。
- 中断时，ARM Cortex-M4 和 M0+内核退出睡眠模式，恢复运行模式处理。
- 退出等待模式后的 MCG 模式与等待模式进入前的 MCG 模式相同。

2.3.7 超低功耗等待模式（VLPW）

- 仅可从 VLPR 进入 VLPW。
- 片内电压调节器处于仅提供 MCU 低频运行所需电压的模式。
- ARM Cortex-M4 和 M0+内核进入睡眠模式。
- ARM Cortex-M4 和 M0+内核时钟被关闭。
- NVIC 仍然对中断敏感。
- 继续为外设提供时钟。
- 清零 SCGCx 寄存器中的时钟关闭位以降低功耗并允许进行单个模块时钟控制。
- 不允许进行 Flex 存储器（EEPROM）编程操作。
- 不允许进行 Flash 存储器编程和擦除操作。
- 退出 VLPW 后的 MCG 模式与 VLPW 进入前的 MCG 模式相同。
- 可选择性地保持外部参考时钟使能（最大 16 MHz）。
- 中断时，ARM Cortex-M4 和 M0+内核退出睡眠模式，恢复 VLPR 或运行模式处理。

2.3.8 正常停止模式

- ARM Cortex-M4 和 M0+内核进入深度睡眠模式。
- ARM Cortex-M4 和 M0+内核时钟被关闭。
- INTC/NVIC 禁用。
- 唤醒中断控制器（WIC）用于从中断唤醒。
- 除非调试模块处于活动状态，否则系统和外设时钟停止。在某些 MCU 上，低功耗模块和 DMA 可使用异步时钟，以便在停止模式下操作。
- 低电压检测（LVD）和低电压警告（LVW）在停止模式下完全工作。
- 所有 SRAM 内容均保留。保持寄存器文件、I/O 和振荡器状态。

- 可从任意 MCG 模式进入停止模式。
- MCU 可配置为使参考时钟保持运行状态。
- 可选择性地保持 PLL 使能，但是输出会关闭—MCG_C5[PLLSTEN]。
- 可选择性地保持内部参考时钟使能 —MCG_C1[IREFSTEN]。
- 可选择性地保持外部参考时钟使能—OSC_CR[EREFSTEN]。
- 将在进入停止模式时的相同时钟模式下退出停止模式，除了在处于 PLL 外部启用（PEE）模式且 PLLSTEN = 0 时进入停止模式；退出停止模式时， MCG 将处于 PLL 外部旁路（PBE）模式。

使用调试器时，MCU 电流更高，这是因为使能调试器时 ARM Cortex-M4 内核需要在停止模式下使时钟处于活动状态。

2.3.9 超低功耗停止模式（VLPS）

- ARM Cortex-M4 和 M0+内核进入深度睡眠模式。
- ARM Cortex-M4 和 M0+内核时钟被关闭。
- INTC/NVIC 禁用。
- WIC 用于从中断唤醒。
- 除非调试模块处于活动状态，否则系统和外设时钟停止。在某些 MCU 上，低功耗模块和 DMA 可使用异步时钟，以便在 VLPS 模式下操作。
- 所有 SRAM 内容均保留。保持寄存器文件、I/O 和振荡器状态。
- 可从任意时钟模式进入 VLPS 模式。
- MCU 可配置为使参考时钟保持运行状态。
- 可选择性地保持内部参考时钟使能 —MCG_C1[IREFSTEN]。
- 可选择性地保持外部参考时钟使能—OSC_CR[EREFSTEN]。
- 将在进入 VLPS 模式时的相同时钟模式下退出 VLPS 模式，除了在处于 PLL 外部启用（PEE）模式且 PLLSTEN = 0 时进入 VLPS 模式；退出 VLPS 模式时， MCG 将处于 PLL 外部旁路（PBE）模式。

使用调试器时，MCU 电流更高，这是因为使能调试器时 ARM Cortex-M4 内核需要在 VLPS 模式下使时钟处于活动状态。

2.3.10 低漏电停止模式（LLS、LLS2 和 LLS3）

- ARM Cortex-M4 和 M0+内核进入深度睡眠模式。
- ARM Cortex-M4 和 M0+内核时钟被关闭。
- INTC/NVIC 禁用且无需使能中断。
- 必须对 LLWU 进行配置，以使能所需的唤醒源。
- 系统和外设时钟停止。OSCERCLK 可选择性地用于某些外设。
- 所有 SRAM（LLS3）或部分 SRAM（LLS2）内容保留。保持寄存器文件、I/O 和振荡器状态。
- 大多数外设都处于状态保持模式（无法工作）。
- 可从任意时钟模式进入 LLS、LLS2 或 LLS3 模式。
- MCU 为静态，无时钟激活（IREFSTEN 和 PLLSTEN 不起作用）。
- 可选择性地保持外部参考时钟使能—OSC_CR[EREFSTEN]。
- 将在进入 LLSx 模式时的相同时钟模式下退出 LLSx 模式，除了在处于 PLL 外部启用（PEE）模式且 PLLSTEN = 0 时进入 LLSx 模式；退出 LLSx 模式时， MCG 将处于 PLL 外部旁路（PBE）模式。
- 发生唤醒事件时，SRS 寄存器中的 WAKEUP 位将置位。
- 完成 LLWU 中断代码后，会继续执行 LLS 模式进入指令的下一条指令。

2.3.11 极低漏电停止 3 模式（VLLS3）

- ARM Cortex-M4 和 M0+内核进入深度睡眠模式。
- ARM Cortex-M4 和 M0+内核时钟被关闭。
- INTC/NVIC 禁用且无需使能中断。

- 必须对 LLWU 进行配置，以使能所需的唤醒源。
- 在 VLL3 中，系统和外设时钟停止。OSCERCLK 可选择性地用于某些外设。
- 大多数模块禁用。
- 所有 SRAM 内容均保留。保留寄存器文件、I/O 和振荡器状态。
- 可从任意时钟模式进入 VLLS3 模式。
- 发生唤醒事件时，将通过复位流程退出 VLLS3。MCU 进入 FLL 内部启用（FEI）模式。
- 发生唤醒事件时，SRS 寄存器中的 WAKEUP 位将置位，并且 MCU 执行来自复位向量的代码。
- 对于 Kinetis MCU，将执行具有最高优先级的首个已使能中断。
- 可选择性地保持外部参考时钟使能—OSC_CR[EREFSTEN]。

2.3.12 极低漏电停止 2 模式（VLLS2）

- ARM Cortex-M4 和 M0+内核进入深度睡眠模式。
- ARM Cortex-M4 和 M0+内核时钟被关闭。
- INTC/NVIC 禁用。
- 必须对 LLWU 进行配置，以使能所需的唤醒源。
- 在 VLLS2 中，系统和外设时钟停止。
- 仅部分模块可以运行。
- 保留 4–8K SRAM 内容，保持寄存器文件、I/O 和振荡器状态。
- 可从任意时钟模式进入 VLLS2 模式。
- 可选择性地保持外部参考时钟使能。
- MCG 关闭，无时钟激活（IREFSTEN 和 PLLSTEN 不起作用）。
- 发生唤醒事件时，将通过复位流程退出 VLLS2。MCU 进入 FEI 模式。
- 发生唤醒事件时，SRS 寄存器中的 WAKEUP 位将置位，并且 MCU 执行来自复位的代码，除非由 NMI 中断唤醒。
- 对于 Kinetis MCU，将执行具有最高优先级的首个已使能中断。
- 可选择性地保持外部参考时钟使能—OSC_CR[EREFSTEN]。

2.3.13 极低漏电停止 1 模式（VLLS1）

- ARM Cortex-M4 和 M0+内核进入深度睡眠模式。
- ARM Cortex-M4 和 M0+内核时钟被关闭。
- INTC/NVIC 禁用。
- 必须对 LLWU 进行配置，以使能所需的唤醒源。
- 在 VLLS1 中，系统和外设时钟停止。
- 仅部分模块可以运行。
- 不保留 SRAM 内容，保持寄存器文件、I/O 和振荡器状态。
- 可从任意时钟模式进入 VLLS1 模式。
- 可选择性地保持外部参考时钟使能。
- MCG 关闭，无时钟激活（IREFSTEN 和 PLLSTEN 不起作用）。
- 发生唤醒事件时，将通过复位流程退出 VLLS1。MCU 进入 FEI 模式。
- 发生唤醒事件时，SRS 寄存器中的 WAKEUP 位将置位，并且 MCU 执行来自复位的代码。
- 对于 Kinetis MCU，将执行具有最高优先级的首个已使能中断。
- 可选择性地保持外部参考时钟使能—OSC_CR[EREFSTEN]。

2.3.14 极低漏电停止 0 模式（VLLS0）

- ARM Cortex-M4 和 M0+内核进入深度睡眠模式。
- ARM Cortex-M4 和 M0+内核时钟被关闭。
- INTC/NVIC 禁用。
- 必须对 LLWU 进行配置，以使能所需的唤醒源。

- 在 VLLS0 中，系统和外设时钟停止。
- 可选择性地使能上电保护（POR）。
- 不保留 SRAM 内容，保持寄存器文件、I/O 和振荡器状态。
- 可从任意时钟模式进入 VLLS0 模式。
- 低功耗振荡器（1 KHz）时钟关闭。
- 外部参考时钟禁用。
- MCG 关闭，无时钟激活（IREFSTEN 和 PLLSTEN 不起作用）。
- 向 MCU 提供 32,768 Hz 方波输入时，RTC 可保持活动状态。
- 发生唤醒事件时，将通过复位流程退出 VLLS0。MCU 进入 FEI 模式。
- 发生唤醒事件时，SRS 寄存器中的 WAKEUP 位将置位，并且 MCU 执行来自复位的代码。
- 将执行具有最高优先级的首个已使能中断。

2.4 能效 – 典型功耗

模式	*K70 - 150 MHz*	*K20 - 50 MHz*	*KL46z- 48 MHz*
运行	89.9 mA	13.9 mA	6.3 mA*
VLP运行 (VLPR)	1.4 mA	867uA - 1.1 mA	292** - 522 uA***
等待	40.9 / 19.6^ mA	7.5 mA	3.3 - 4.4mA
VLP等待 (VLPW)	926 uA	509 uA	261 uA
停止	1.3 mA	310 uA	212 uA
VLP停止 (VLPS)	250 uA	3.5 uA	2.8 uA
LL停止 (LLS)	250 uA	2.1 uA	2.0 uA
VLL停止3 (VLLS3)	5.6 uA	2.9 uA	1.5 uA
VLL停止1 (VLLS1)	2.8 uA	1.5 uA	650 nA
VLL停止0 (VLLS0)	-	176 - 367 nA	130 - 330 nA

Cortex-M0+

*已使能计算操作：4.1 mA（48MHz内核/ 24MHz总线）
** 已使能计算操作：188uA（4MHz内核/ 800kHz总线）
*** 运行Coremark算法，KEIL 4.54针对速度进行优化

^降频等待

图 6. 能效 – 典型功耗

请注意，这些都是 3V 和 25C 条件下的典型电流估计值。运行模式和 VLPR 模式所对应的数值基于所有模块均关闭（计算操作）且已使能时钟选项。在运行模式下，当内核频率为最大值 48 MHz 时，功耗为 4 mA。在 VLPR 模式下，当内核频率为最大值 4 MHz 时，功耗为 200 uA。在这两种模式下，所有模块均关闭且已使能时钟选项时，节能外设仍可运行。

2.5 超低功耗模式检查表

2.5.1 进入低功耗模式之前要执行的操作项目

- SMC_PMPROT 必须设置为允许使用您希望进入的停止模式和低功耗运行模式。复位后，该寄存器只能写入一次。如果不允许使用任何低功耗模式且 SLEEPDEEP = 1，则执行 WFI 指令将进入正常停止低功耗模式。
- 如果使用外部时钟源，则在进入除等待模式之外的任何低功耗模式之前，必须先禁用时钟监视器。
- 进入低功耗模式之前，应使能唤醒源。对于 VLPS、等待或 VLPW 模式，为中断，而对于 LLSx、VLLSx 模式，则为 LLWU 源。如果未使能任何源，则复位和 NMI 仍会将 MCU 从所有低功耗模式唤醒。
- 在 SMC_PMCTRL 和 SMC_STOPCTRL 寄存器中选择所需的低功耗模式。
- 针对停止/等待模式，置位/清除 ARM SLEEPDEEP 位。
- 以上所有步骤均可在初始化阶段完成。如果还存在其他操作并需要至少 12 个内核时钟周期，则在置位停止模式和执行 WFI 指令之间无需序列化。
- 除非满足唤醒条件，否则等待中断指令（WFI）将使器件立即进入睡眠模式。
- 如果在进入睡眠模式前一刻切换低功耗模式，则在执行 WFI 之前，必须确保对 SMC_PMCTRL 或 SMC_STOPCTRL 寄存器的写入操作已经完成。这可通过在执行 WFI 指令之前对 SMC_PMCTRL 寄存器执行读取操作来完成。这就是序列化。

2.5.2 使用 sleep-on-exit 进入低功耗模式:

- 置位 SCR 寄存器的 Sleep-On-Exit 模式并使 SLEEPDEEP=1。
- 执行 WFI 以首次进入。
- 正常停止、VLPS 模式或 LLS 可通过 Sleep-On-Exit 使用。
- 如果 SCR 的 SLEEPONEXIT 位置 1，则在完成所有异常句柄执行之后，处理器将立即重新进入睡眠模式。
- 置位 Sleep-On-Exit 模式后，低功耗外设和异步 DMA 可以在 VLPS 下工作。
- 此机制可用在仅需在发生唤醒异常时运行内核的应用中。

2.5.2 要进入 VLPR，您必须:

- 处于 BLPI（对于 MCG 使用快速 IRC，对于 MCG_Lite 使用 LIRC）或 BLPE 时钟模式。
- 将内核频率设为 4 MHz 或更低并将 Flash 时钟设为 1 MHz 或更低。
- 禁用慢速 IRC。
- 禁用时钟监视器。
- 使用 MCG_Lite 的器件要禁用 FIRC。
- 写入 SMC_PMCTRL RUNM 位以进入 VLPR。

2.5.4 退出 VLPR

- 通过写入 SMC_PMCTRL RUNM 位，退出 VLPR
- 在 K 系列上，可通过中断来实现。
- 通过写入 RUNM 位来退出 VLPR 不会更改代码流。不会产生任何异常。

2.5.5 退出 LLS、LLS2 或 LLS3

- 发生低漏电唤醒（LLWU）事件时，LLWU 中断将在唤醒之后挂起并获得优先级。

- 可在 LLWU 异常处理程序（ISR）中清除内部模块标志，或者如果您保留模块标志置位，则完成 LLWU ISR 之后，您可进入模块 ISR 以清除该标志。如果清除该模块标志且不想进入模块 ISR，则必须清除此模块的 NVIC 清除挂起中断标志位。此时必须执行序列化操作，以确保在从 LLWU 或模块 ISR 返回之前已清除该标志。
- 执行完所有异常处理程序之后，代码将从 WFI 指令之后的指令重新开始。

2.5.6 退出 VLLS3、VLLS2、VLLS1 或 VLLS0

- VLLSx 恢复通过复位流程实现。
- 退出任何 VLLSx 模式之后，IO 保持锁定，直到通过向 PMC 中的 ACKISO 位写入“1”将其释放。
- 写入 ACKISO 之前，必须重新配置 GPIO。
- 如果无需使用任何外部 IO，则并未写入 ACKISO 位时，重新进入 VLLSx 之前无需重新配置 IO。
- 如果振荡器在 VLLS1、VLLS2 或 VLLS3 下继续运行，则写入 ACKISO 之前，必须重新配置该振荡器（除非已在 RTC OSC 中配置）。
- 如果使用 VLLS1 或 VLLS0，则需要重新初始化 RAM 内容。
- 对于 VLLS2 中未上电的 RAM，需要重新初始化 RAM 内容。
- LLWU 中断将挂起。
- 如果由模块唤醒，则关联的模块中断也将挂起。
- 所执行的中断为复位流程中使能的第一个中断。
- 如果使能的第一个中断是 LLWU 中断，则必须清除 LLWU ISR 中的模块中断标志。
- 如果使能的第一个中断是模块中断，则模块 ISR 将清除中断标志且 LLWU ISR 不再挂起。
- 必须清除 LLWU 中的引脚唤醒标志。
- 此时 VLLS 唤醒将在 RCM 中报告。

注

VLLS0 模式仅适用于部分 Kinetis 器件。

2.6 Kinetis 节能 – 提示和技巧

1. 绝对的最低功耗模式为“VDD 关闭”。此模式使用 RTC_WAKEUP 控制 MCU VDD。在部分具有隔离 VBAT 电源域的新款 Kinetis 器件上，如 K22F、K64、K24 和 K65，可使用称为 RTC_WAKEUP_B 的控制输出来管理使用外部组件的 MCU VDD。
2. 使用 FOPT 选项：存在多个 FOPT 寄存器位（例如 LPBOOT），有助于降低启动时的电流或禁用 EZPORT 模式进入等不需要的功能。
3. 从技术角度来说，将异步 DMA 与低功耗外设结合使用是另一种低功耗模式。在低功耗模式下，外设（如 LPUART）可以发送和接收数据，直到缓冲器需要管理。在已禁用内核时钟的停止或 VLPS 模式下：
   - 仍可进行 UART 数据传输
   - 可通过 DMA 更改 PWM 输出
   - 可通过 DMA 读取结果寄存器进行 ADC 数据采样
4. UART0/LPUART – 低功耗 UART:
   - 与异步 DMA 结合使用时，可进行过采样、地址匹配、异步操作。
5. 在除最低功耗模式 VLLS0 之外的所有模式下，采用 LPO 的低功耗定时器（LPTMR）提供低分辨率（通常为 1 ms）唤醒定时器。
   - 所有功耗模式下都只增加极少电流。
   - 通过除 POR 和 LVD 外的全部复位进行操作。
   - 在除 VLLS0 外的所有功耗模式下，均可采用内部 1 KHz 低功耗振荡器运行。
6. 对于定期唤醒源，采用外部方波时钟源的实时时钟（RTC）为最低功耗选项。
   - 唤醒源增加电流最小。
   - 可在所有功耗模式下运行。
   - 通过除 POR 和 LVD 外的全部复位进行操作。
   - 在除 VLLS0 外的所有功耗模式下，可采用外部 32,768 Hz 低功耗晶振工作。
7. 未使用的引脚应配置为禁用状态（mux(0)），以防止意外漏电（可能因浮动输入所致）。

8. 关闭 SIM 中不必要的模块时钟门控。
9. 使用编译器选项减少 Flash 访问文字池产生的毛刺电流。文字池用作模块寄存器组的基地址。编译器可优化文字池地址。
10. 尽可能减小循环，以便将代码访问保持在缓存大小内。
11. 从 RAM 而非 FLASH 中执行常用循环或函数。
12. 显式地写入寄存器，而不是进行“读取-修改-写入”。
13. ADC：如果不需要 32x，则使用 4x h/w 均值。
14. 如有可能，以较低时钟速度使用低功耗模式。
15. 对于 GPIO 访问，尽可能使用 IOPORT。它可在更短的时钟周期内将写入内容传播到输出引脚。
16. 对于具有 USB 功能的电池应用，连接至 USB 总线（可使用片内调压器）时，请在电源线路中添加一个隔离二极管并使用 USB 为应用供电。
17. 此外，对于具有 USB 功能的电池应用，可通过使用大小合适的分压电阻将 VBUS 连接至 LLWU 唤醒输入引脚，以便在低功耗模式下使用 VBUS 唤醒 MCU。
18. 编译器优化等级可影响 IDD。
19. 最低功耗时的 IDD 并不一定最低。图 7 所示为 Kinetis L MCU 上 IDD 与电压的典型关系曲线。图 8 所示为总功率曲线（功率 = IDD * VDD）。请注意，在较低 VDD 值下运行 MCU 的确可降低功耗，但是会导致 IDD 较高。

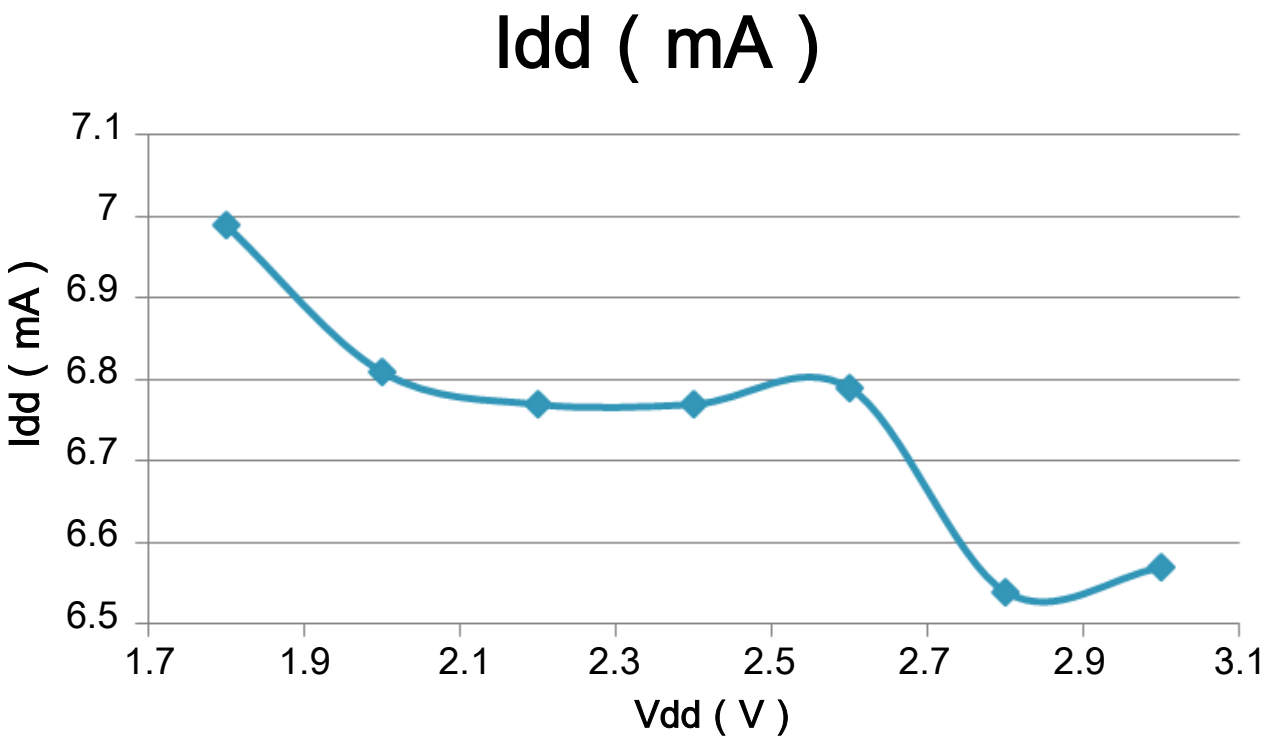


图 7. Kinetis MCU 上 IDD 与 VDD 之间的典型关系

在最低电压实现最低功耗

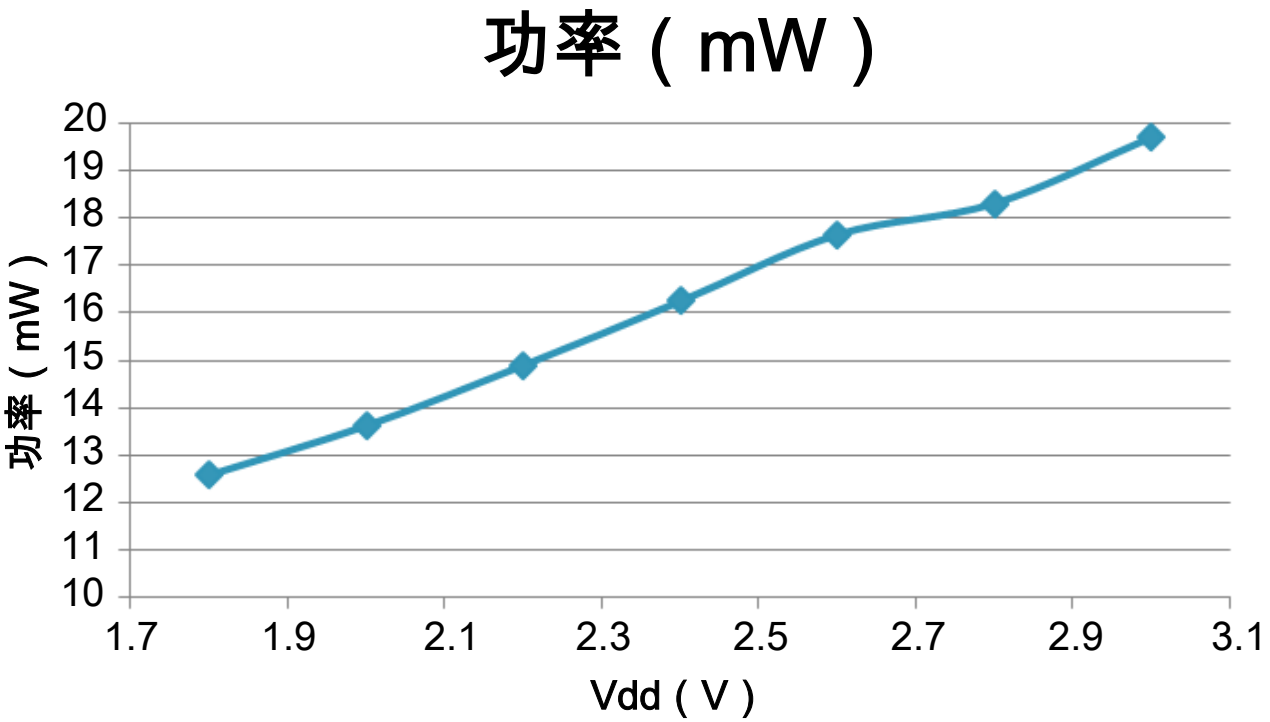


图 8. Kinetis MCU 功率与 VDD 之间的关系

3 Kinetis SDK 快速入门

Kinetis SDK 推出之后，越来越多的 Kinetis MCU 可以借助低层软件支持来设置和操作 MCU。SDK 中的最低层代码为硬件抽象层(HAL)。HAL 的上一层为参考外设驱动程序和系统服务驱动程序。功耗模式控制是系统服务驱动程序集的一部分，并且允许在进入和退出功耗模式时进行高级设备管理，而 HAL 函数则允许您创建自己的自定义驱动程序并且可以提供更多灵活性。

本 SDK 版本包含演示工程，提供了基于用例的示例，您可以在自己的应用中重复使用这些示例。其中一个演示为 power_manager_demo。此代码提供了一个工作功耗模式转换示例，供您学习和重复使用。

本快速入门章节介绍了如何使用 HAL 或系统服务将 Kinetis MCU 从运行模式置于 LLS 低功耗模式。如果您的应用从基于非 SDK 的项目开始或作为基于 SDK HAL 的项目开始，则您可能希望继续使用 HAL（并且 HAL 示例的假设条件亦是如此）。但是，如果您的项目从基于 SDK 驱动程序或系统服务的应用程序开始，则您可能希望继续使用参考外设驱动程序（并且功耗管理系统服务示例的假设条件亦是如此）。首先，我们重点关注 HAL 层的使用，随后我们将着重介绍预建驱动程序的使用。

3.1 LLS 模式进入和退出 - Kinetis 软件开发套件（KSDK）代码示例

3.1.1 使用 SDK HAL 函数准备 MCU

进入低功耗模式的第一步为让 MCU 做好进入该低功耗模式的准备。这包括关闭所需模式中不需要的所有外设模块并且将任何未使用的 GPIO 引脚置于禁用状态。SDK HAL 中提供了多个易于使用的函数，可用于实现此操作。此类语句的示例如下所示。

```
/* Disable PTC7 */
PORT_HAL_SetMuxMode(PORTC,7u,kPortPinDisabled );
```

应针对应用中使用的每个引脚执行此操作。

3.1.2 唤醒源配置

接着, 必须配置退出方法。此示例使用与 GPIO 引脚相连的按钮进行退出。配置退出方法的第一步为初始化与该按钮相连的 GPIO 引脚。以下代码演示了如何使用 KSDK HAL 函数实现此目的。具体来说，以下代码将执行以下任务:

- 配置用于从 **LLS** 模式唤醒的 **GPIO** 引脚。
- 将引脚配置为数字输入
- 将 **LLWU** 模块引脚 **PORTC6 (LLWU_P10)**配置为有效唤醒源。

```
void main (void){
  /* Enable Port C6 to be a digital pin.   */
  SIM_HAL_EnablePortClock(SIM_BASE, HW_PORTC);

  /* PORTC_PCR6 */
  PORT_HAL_SetMuxMode(PORTC_BASE,6u,kPortMuxAsGpio);

  /* Configure GPIO input features. */
  PORT_HAL_SetPullCmd(PORTC_BASE, 6u, true);
  PORT_HAL_SetPullMode(PORTC_BASE, 6u, kPortPullUp);

  /* Set the LLWU pin enable bits to enable the PORTC6 input
  * to be a wake-up source.
  * WUPE10 is used in this case since it is associated with PTC6.
  * This information is in the Chip Configuration chapter of the Reference Manual.
  * 0b10: External input pin enabled with falling edge detection
  */
  SIM_HAL_EnableLlwuClock(SIM_BASE, HW_LLWU); // this bit may not exist on some MCUs
  LLWU_HAL_ClearExternalPinWakeupFlag(LLWU_BASE, 10u);

  //falling edge detection
  LLWU_HAL_SetExternalInputPinMode(LLWU_BASE, kLlwuExternalPinFallingEdge, kLlwuWakeupPin10);
```

3.1.3 使用 SDK HAL 函数配置 SMC

使 MCU 外设准备好进入低功耗模式并完成唤醒源配置之后，就可以配置 SMC（或 MC，具体取决于特定的 MCU）了。请遵循以下步骤:

- 首先，必须配置功耗模式保护寄存器，以允许进入所需模式。这是只能写入一次的寄存器，仅可在每次复位之后写入一次。建议在您的启动代码中对此寄存器执行写入操作。

  ```
  /* Power mode protection initialization */
  SMC_HAL_SetProtection(SMC, kAllowPowerModeLls);
  ```

注

只需在函数调用通过或运算将相应的屏蔽位合并在一起，即可启用多个功耗模式。例如，如果还要允许进入 VLPS 模式，则函数调用将变为 SMC_HAL_SetProtection(SMC, kAllowPowerModeLls | kAllowPowerModeVlps);

- 然后，必须配置停止模式和子模式。配置完成后，应置位 ARM 系统控制块（SCB）中的睡眠位，然后执行等待中断（WFI）指令。借助 SDK HAL，只需单次函数调用，即可便利地完成这些任务。这是 SMC_HAL_SetMode 函数，其使用示例如下所示：

```
/* First, setup an smc_power_mode_config_t variable to be passed to the SMC
 * HAL function
 */
smc_power_mode_config_t smc_power_config = {
    .powerModeName = kPowerModeLls,          // Set the desired power mode
    .stopSubMode = kSmcStopSub3,             // Set the sub-mode if necessary
#if FSL_FEATURE_SMC_HAS_LPWUI
    .lpwuiOption = false,                   // Set the LPWUI option if available
    .lpwuiOptionValue = kSmcLpwuiDisabled,
#endif
#if FSL_FEATURE_SMC_HAS_PORPO
    .porOption = false,                      // Set the PORPO option.  Only needed
    .porOptionValue = kSmcPorEnabled,    //  if entering VLLS0
#endif
#if FSL_FEATURE_SMC_HAS_PSTOPO
    .pstopOption = false,                // Only needed if entering a PSTOP mode
    .pstopOptionValue = kSmcPstopStop,
#endif
};

/* Disable clock monitor before moving to a STOP mode */
CLOCK_HAL_DisableOsc0Monitor(MCG_BASE);

/* Now call the SMC HAL SetMode function to move to the desired power mode. */
SMC_HAL_SetMode(SMC_BASE, &smc_power_config);
```

注

SMC_HAL_SetMode 函数可以简单地配置 SMC 控制寄存器并执行 WFI 指令，并且每次执行相同操作，无论是否需要对这些寄存器执行写入操作。如果您的应用需要更高级功能，例如更短的低功耗模式进入时间或使用 Sleep-on-exit，可通过几种方法完成这些任务。这些方法如下所示：

- 更短的低功耗模式进入时间：调用 SMC_HAL_SetMode 函数之后，SMC 的控制寄存器在该函数退出之后仍处于相同的配置。因此，如果后续需要进入相同的低功耗模式，则您只需执行如下所示的 WFI 指令：

```
__WFI();
```

- Sleep-on-exit：要使用 Sleep-on-exit 功能，只需在调用 SMC_HAL_SetMode 函数之前将系统控制寄存器（SCB->SCR）中的 sleep-on-exit 位置位。SMC_HAL_SetMode 函数仅对 SCB->SCR 寄存器执行“读取-修改-写入”操作，因此 sleep-on-exit 位的配置将保留不变。操作示例方法如下所示：

```
/* Set the Sleep-on-Exit bit */
SCB->SCR |= SCB_SCR_SLEEPONEXIT_Msk;

/* Then call the SMC_HAL_SetMode function */
SMC_HAL_SetMode(SMC_BASE, &smc_power_config);
```

3.1.4 退出低漏电停止（LLS）低功耗模式

MCU 目前处于 LLS 模式。必须使唤醒引脚电平变为有效，才能将 MCU 从 LLS 模式唤醒（在此情况下为端口 C 引脚 6 上出现下降沿）。至少必须执行以下代码，以清除 LLWU 寄存器中的唤醒事件标志（请注意，对于 LLS 或 VLLSx 模式，无需使能中断，但其可以简化 LLS 或 VLLS 模式退出程序）。

```
  /* after exiting LLS (can be in LLWU interrupt service Routine)
  * clear the wake-up flag in the LLWU-write one to clear the flag
  */
  if (LLWU_HAL_GetExternalPinWakeupFlag(LLWU_BASE, 10u)) {
      LLWU_HAL_ClearExternalPinWakeupFlag(LLWU_BASE, 10u);
  }
} /* end of main */
```

注

如果中断已使能，则会恢复执行相应的中断服务程序。如果中断已禁用，则会恢复执行 WFI 或 STOP 指令后的指令。

3.1.5 使用功耗管理系统服务准备 MCU

使用功耗管理系统服务时，MCU 进入低功耗模式的准备工作与使用 HAL 时相同，但存在一个区别。仍应关闭所需模式下不需要的所有外设模块，并且应将任何未使用的 GPIO 引脚置于禁用状态。区别在于可以使用回调函数。使用回调函数进行此操作的示例如下：

```
/* Structure for callback data */
typedef struct {
    uint32_t counter;
    uint32_t status;
    uint32_t err;
} callback_data_t;

/* Callback structure definition to be used by the Power Manager variables */
typedef struct {
    callback_data_t none;
    callback_data_t before;
    callback_data_t after;
    power_manager_callback_type_t lastType;
    uint32_t err;
} user_callback_data_t;

/* This is the declaration of the callback function that can be used
     * in power mode transitions
     */
    user_callback_data_t callbackData0;

    power_manager_static_callback_user_config_t callbackCfg0 = { callback0,
            kPowerManagerCallbackBeforeAfter,
            (power_manager_callback_data_t*) &callbackData0 };

    power_manager_static_callback_user_config_t *callbacks[] =
            { &callbackCfg0 };

/*
 * Power manager callback implementation code
 *
 * This section defines what the power manager functions do before
 * and after the power manager mode transitions occur.  It also defines
 * what to do if no callback is provided.
 *
```

```
 */
power_manager_error_code_t callback0(power_manager_callback_type_t type,
        power_manager_user_config_t * configPtr,
        power_manager_callback_data_t * dataPtr) {

    user_callback_data_t * userData = (user_callback_data_t*) dataPtr;
    power_manager_error_code_t ret = kPowerManagerError;

    switch (type) {

    case kPowerManagerCallbackNone:

        userData->none.counter++;
        ret = kPowerManagerSuccess;
        break;
    case kPowerManagerCallbackBefore:

        /* Disable PTC7 and any other pins that are not used */
        PORT_HAL_SetMuxMode(PORTC,7u,kPortPinDisabled );

        userData->before.counter++;
        ret = kPowerManagerSuccess;
        break;
    case kPowerManagerCallbackAfter:

        userData->after.counter++;
        ret = kPowerManagerSuccess;
        break;
    default:
        userData->err++;
        break;
    }

    userData->lastType = type;

    return ret;
}
```

3.1.6 使用功耗管理系统服务配置唤醒源

配置唤醒源的第一步为将所需引脚添加到_gpio_pins 枚举中。以下为示例代码:

```
/*! @brief gpio pin names.*/
/*!*/
/*! This should be defined according to board setting.*/
enum _gpio_pins
{
    kGpioSW1         = GPIO_MAKE_PIN(GPIOC_IDX, 6),  /* TWR-K22F120M SW1 */
};
```

接着，必须将该引脚包括在输入引脚结构体中，该结构体定义了该引脚在应用中工作所必需的所有参数。此结构体供 GPIO 驱动程序使用，以正确初始化所需引脚:

```
  /* Declare Switch pins */
gpio_input_pin_user_config_t switchPins[] = {
    {
        .pinName = kGpioSW1,
        .config.isPullEnable = true,
        .config.pullSelect = kPortPullUp,
        .config.isPassiveFilterEnabled = false,
        .config.interrupt = kPortIntDisabled,
    },
    {
        .pinName = GPIO_PINS_OUT_OF_RANGE,
    }
};
```

将 GPIO 引脚使能为低功耗唤醒源的另一项重要操作为正确配置端口 MUX 域。这可通过 hardware_init.c 中的 hardware_init 函数来实现。另外，必须仅使能要使用的引脚（以便尽可能降低功耗）。以下示例显示了如何使能引脚实现 GPIO 功能。

```
/* enable clock for PORTC */
CLOCK_SYS_EnablePortClock(2);

/* enable PORTC pin 6 as a GPIO */
PORT_HAL_SetMuxMode(PORTC,6u,kPortMuxAsGpio);
```

现在，可以调用 GPIO 驱动程序以初始化所需引脚。

```
 /* Initialize the pins used for switches and LEDs only */
    GPIO_DRV_Init(switchPins, NULL);
```

接下来，由于低漏电功耗模式（LLS、VLLSx）使用低漏电唤醒单元（LLWU）作为恢复方法，因此，必须配置 LLWU。为此，我们使用如下所示的 LLWU HAL 函数。

```
/*************************************************************
     * LLWU pin initialization
     *
     * First, clear the associated flag just to be sure the device
     * doesn't immediately enter the LLWU interrupt service routine
     * (ISR). Then enable the interrupt
     *************************************************************/
    LLWU_HAL_ClearExternalPinWakeupFlag(LLWU_BASE, pinEn);

    LLWU_HAL_SetExternalInputPinMode(LLWU_BASE, riseFall, pinEn);
```

在上述函数调用中，LLWU_BASE 为 LLWU 的寄存器位置（在特定器件的头文件中定义），riseFall 为 llwu_external_pin_modes_t 型变量（在 fsl_llwu_hal.h 中定义），pinEn 为 uint8_t 变量，用于指定 LLWU 中要使能的引脚。

3.1.7 使用功耗管理系统服务进入低漏电停止（LLS）低功耗模式

Kinetis SDK 功耗管理系统服务使用一组电源配置与应用进行交互。用户需要创建一组应用中要使用的所有电源配置。以下所示为功耗管理器使用的一组定义示例。

```
    /* Power manager configuration variables that configure
     * the named low power mode
     */
    power_manager_user_config_t llsConfig;
    power_manager_user_config_t runConfig;

    /* This is the power manager configuration variable.  The order of this
     * list determines the index that should be passed to the Setmode
     * function to enter the desired power mode.
     */
    power_manager_user_config_t const *powerConfigs[] = { &llsConfig, &runConfig };
```

声明功耗管理器用户配置并定义功耗管理器函数调用阵列之后，必须配置功耗管理器配置变量。示例如下所示。

```
    /* Define the power mode configurations */
    /* Define all of the parameters for the LLS configuration */
    llsConfig.mode = kPowerManagerLls;
    llsConfig.policy = kPowerManagerPolicyAgreement;
#if FSL_FEATURE_SMC_HAS_LPWUI
    llsConfig.lowPowerWakeUpOnInterruptOption = true;
    llsConfig.lowPowerWakeUpOnInterruptValue = kSmcLpwuiEnabled;
#endif
    llsConfig.sleepOnExitValue = false;
    llsConfig.sleepOnExitOption = false;
#if FSL_FEATURE_SMC_HAS_PORPO
    llsConfig.powerOnResetDetectionOption = true;
```

```
    llsConfig.powerOnResetDetectionValue = kSmcPorEnabled;
#endif
#if FSL_FEATURE_SMC_HAS_LPOPO
    llsConfig.lowPowerOscillatorOption = true;
    llsConfig.lowPowerOscillatorValue = kSmcLpoEnabled;
#endif
```

注

如果使用的器件具有 LLS 子模式（即 LL2 和 LL3），则您可以使用*.mode 变量 kPowerManagerLls2 和 kPowerManagerLls3 对其执行指定。

配置功耗管理器用户配置变量之后，需要初始化功耗管理器并注册回调函数。以下为操作代码示例。

```
/* Initialize the power manager module */
POWER_SYS_Init(&powerConfigs,
     sizeof(powerConfigs)/sizeof(power_manager_user_config_t *),
     &callbacks,
     sizeof(callbacks)/sizeof(power_manager_callback_user_config_t *));
```

上述函数可以简单地告知功耗管理系统服务哪些配置可用。要使用驱动程序进入低功耗模式，请使用 POWER_SYS_SetMode 函数并将索引传递给以上代码中定义的 powerConfigs 阵列中的配置。此函数将返回一个状态变量，用于确认是否已成功更改模式（请注意，如果进入 VLLSx 模式，则不会执行此代码，因为这些模式通过复位序列退出）。以下是一个示例。

```
/* Now Issue power mode change */
ret = POWER_SYS_SetMode(powerModeLls1Index);

/* The code execution should not get here on successful entries.
 * However, unsuccessful entries should make it to this code and
 * report errors
 */
if (ret != kPowerManagerSuccess)
{
    printf("POWER_SYS_SetMode(powerModeLls1Index) returns : %u\n\r", ret);
}
```

要进一步探索这些驱动程序，请参见 SDK 安装中的 power_manager_hal_demo，路径为<SDK root>/apps/<board name>/demos/power_manager_hal_demo。

4 快速启动

4.1 LLS 模式进入和退出 - 裸机代码示例

对于希望快速看到低功耗操作的用户，只需添加以下代码行，即可进入低漏电停止（LLS）模式。要使能唤醒引脚，请遵循以下初始化部分中的步骤。初始化部分还会禁用时钟监视器，仅当应用采用外部时钟源运行时才需要该监视器。

此处选择演示 LLS 模式，因为它是其中一种比较有用的低功耗模式。退出低功耗模式时执行 WFI 指令（该指令将 MCU 置于低功耗模式）的下一条指令。LLS 的唤醒时间非常短，当处于 FLL 内部启用（FEI）时钟模式且频率为 100 MHz 时快达 4 us。它是低功耗模式序列中需要 LLWU 将 MCU 从低功耗模式唤醒的第一种模式。LLS 模式在低功耗模式下维持所有寄存器和 SRAM。

4.1.1 初始化

以下代码将执行以下任务:

- 配置用于从 **LLS** 模式唤醒的 **GPIO** 引脚。

- 将引脚配置为数字输入（所示为 Kinetis 器件）
- 将 **LLWU** 模块引脚 **PORTE1 (LLWU_P0)**配置为有效唤醒源。

```
void main (void){
  unsigned int dummyread;
  /* Enable Port E1 to be a digital pin.   */
  SIM_SCGC5 = SIM_SCGC5_PORTE_MASK;
  PORTE_PCR1 =(PORT_PCR_ISF_MASK |  //clear flag if there
              PORT_PCR_MUX(01) |   //set pin functionality -GPIO
              PORT_PCR_IRQC(0x0A)| //falling edge interrupt enable
              PORT_PCR_PE_MASK |   // pull enable
              PORT_PCR_PS_MASK);   // pullup enable

  /* Set the LLWU pin enable bits to enable the PORTE1 input
  * to be a wake-up source.
  * WUPE0 is used in this case since it is associated with PTE1.
  * This information is in the Chip Configuration chapter of the Reference Manual.
  * 0b10: External input pin enabled with falling edge detection
  */
  SIM_SCGC4 = SIM_SCGC4_LLWU_MASK; // this bit may not exist on some MCUs
  LLWU_PE1 = LLWU_PE1_WUPE0(2); //falling edge detection
```

- 如果使用外部时钟源，则禁用时钟监视器

```
  /* Need to make sure the clock monitor(s) is disabled if using
  * an external clock source. This assumes OSC0 is used as the
  * external clock source and that the clock gate for MCG has
  * already been enabled.
  */
  MCG_C6 &= ~MCG_C6_CME0_MASK; //CME=0 clock monitor disable
```

4.1.2 进入低漏电停止（LLS）低功耗模式

以下代码将执行以下任务:

- 允许 **MCU** 进入 **LLS** 模式。**PMPROT** 寄存器是只能写入一次的寄存器，并且应对其执行写入操作以允许所有需要的低功耗模式。如果仅需 **LLS** 和 **VLLS1** 模式，请仅使能这些模式。

```
/* Write to PMPROT to allow LLS power modes */
  #ifdef MC1
  MC_PMPROT = MC_PMPROT_ALLS_MASK; // (MC1)
  #else
  SMC_PMPROT = SMC_PMPROT_ALLS_MASK;   // (MC2)
  #endif
```

- 置位 **LPLLSM**（对于 **MC1**）或 **STOPM**（对于 **MC2**）位，以选择 **LLS** 低功耗模式。如果需要 **VLLS1** 模式，则需要对 **MC2 MCU** 再执行一次写入操作。
- 将 **VLLSM** 位设为 **0b11**（对于 **LLS3**）。如果需要 **LLS2** 模式并保留经过减少的 **SRAM**，请选择 **0b10**。

```
  #ifdef MC1
  MC_PMCTRL  =  MC_PMCTRL_LPLLSM(3); // (MC1) set LPLLSM = 0b11
  #else
  SMC_PMCTRL =  SMC_PMCTRL_STOPM(3); // (MC2) Set STOPM = 0b11
  #endif

  #ifdef MC4
  SMC_VLLSCTRL =  SMC_VLLSCTRL_VLLSM(3);  // (MC4) Set VLLSM = 0b11 for LLS3
  #endif
```

- 对于 **Kinetis** 器件，请置位内核中的 **SLEEPDEEP** 位，然后执行中断唤醒（**WFI**）指令以进入 **LLS** 模式。
- 需要对 **PMCTRL** 寄存器执行序列化读取，以确保在完成写入操作以设置低功耗模式之前内核不会停止工作。有关序列化的详情，请参见功耗模式转换代码 章节。

```
/*wait for write to complete to SMC before stopping core */
dummyread = SMC_PMCTRL;

/* Set the SLEEPDEEP bit to enable deep sleep mode */
SCB_SCR |= SCB_SCR_SLEEPDEEP_MASK;

/* WFI instruction will start entry into low-power mode */
asm("WFI");
```

- 需要对 **PMCTRL** 寄存器执行序列化读取，以确保在完成写入操作以设置低功耗模式之前内核不会停止工作。有关序列化的详情，请参见功耗模式转换代码 章节。

```
/*wait for write to complete to SMC before stopping core */
dummyread = SMC_PMCTRL;

SIM_SOPT4 = SIM_SOPT4_STOPE_MASK;  //enable stop mode entry
asm (  stop #0x2000; )
```

4.1.3 退出低漏电停止（LLS）低功耗模式

- **MCU** 目前处于 **LLS** 模式。
- **LLWU** 唤醒电路独立于中断工作。
- 无需使能中断，也无需使能 **LLWU** 中断向量，即可从 **LLS** 或 **VLLSx** 低功耗模式唤醒 **MCU**。
- 如果 **PORTE1** 上发生下降沿事件，则 **MCU** 将退出 **LLS** 模式并返回运行模式。
- 从 **LLS** 模式退出后，以下代码将清除 **LLWU** 标志寄存器中的唤醒事件标志。
- 如果中断和 **LLWU** 中断已使能，则会恢复执行 **LLWU** 中断服务程序。然后，代码执行将返回至 **WFI** 或 **STOP** 指令后的指令。
- 如果中断已禁用，则会恢复执行 **WFI** 或 **STOP** 指令后的指令。

```
  /* after exiting LLS (can be in LLWU interrupt service Routine)
  * clear the wake-up flag in the LLWU-write one to clear the flag
  */
  if (LLWU_F1 & LLWU_F1_WUF0_MASK) {
      LLWU_F1 |= LLWU_F1_WUF0_MASK;
  }
} /* end of main */
```

5 复位管理

5.1 低功耗模式下 I/O 上的复位和保留功能

在引入可通过复位流程恢复的新低功耗模式之后，Kinetis MCU 中的复位管理变得比以前的微控制器更加复杂。复位不再是“将所有模块返回至默认状态”的简单功能。在某些复位类型中，许多用于将 MCU 从低功耗模式唤醒的模块不会清除寄存器状态。

“复位”章节详细介绍了所有复位类型。以下是一个简短的复位类型列表：

1. 上电复位（POR）和低电压检测（LVD）复位。在 MCU 首次通电时发生。存在多个电源域时，会针对 VDD 电源域和 VBAT 电源域执行 POR 复位。仅当存在 RTC 无效标志时，才可检测到 VBAT 中发生 POR 复位。
2. 外部引脚复位（PIN）。此引脚为开漏引脚，具有内部上拉电阻。RESET 的电平变为有效值可将器件从任意模式唤醒。如果是从 VLLSx 低功耗模式唤醒，I/O 引脚的保持值会无效。

3. 计算机正常运行（COP）看门狗定时器
4. 从技术角度来说，LLWU 并非是一个复位源，但如果是从 VLLSx 低功耗模式唤醒，则 LLWU 低漏电唤醒单元确实会导致复位。(WAKEUP)位已置位。
5. 多用途时钟发生器时钟丢失 (LOC)。如果在进入停止或低功耗模式时时钟监视器已使能，则会进行 LOC 复位。这是由内部慢速 IRC 在低功耗模式下停止所导致的。慢速 IRC 用作时钟监视器的参考时钟。
6. MCG 锁定丢失（LOL）复位
7. 停止模式应答错误（SACKERR）
8. 软件复位（SW）
9. 内核死锁复位（LOCKUP）
10. EzPort 复位
11. MDM-AP 系统复位请求

在从 VLLSx 低功耗模式恢复而发生的复位流程中，SRS 寄存器中的 WAKEUP 位将置位，并且以下模块不会复位：LPTMR、RTC、CMP、TSI、PMC、SMC。此详细信息源自于 MCU 参考手册中每个寄存器位描述之前的注释。具体措辞如下：

“附注：此寄存器在收到非 **VLLS** 型的芯片 **POR** 时复位且其复位类型会触发非 **VLLS** 型的芯片 **POR**。不触发非 **VLLS** 型的芯片 **POR** 的复位类型对其无影响。有关更多信息，请参见‘复位’章节。”

如上 LLWU 复位中所述，LPTMR 寄存器仅可通过 POR 和 LVD 类型的复位进行复位。因此，该模块可以在所有功耗模式下以及除 POR 和 LVD 复位之外的所有复位类型持续工作。还有其他许多此类模块。

LLWU 复位在从 VLLS0、VLLS1、VLLS2 或 VLLS3 恢复时发生。借助 SRS 状态寄存器和模式控制器控制寄存器，您可以识别复位是来自 LVD、POR 还是 LLWU 唤醒事件。如果为 LLWU 复位，则 SRS 寄存器中的 WAKEUP 位会置位，并且可从模式控制器控制寄存器中的值识别 MCU 是从哪种模式恢复的。

管理所有这些不同类型的复位还需要考虑一些其他因素。某些模块在复位后不会复位至默认状态。部分或所有 RAM 都可保留并且无需通过启动代码进行初始化。

5.1.1 I/O 和振荡器的保留功能

如果处于 VLLSx 和 LLS 低功耗模式，当未通过复位引脚退出 VLLSx 低功耗模式时，所有 I/O 状态和系统振荡器都将保留。保留将在退出 LLS 模式时自动释放，但大多数 I/O 引脚都必须在退出 VLLSx 模式时手动释放。

GPIO：非 VLLSx 复位恢复之后，所有 Kinetis MCU 上大部分数字 I/O 引脚都默认处于高阻抗模式。但是，在 VLLSx (LLWU) 复位恢复时，引脚状态将保留。例如，如果端口引脚已配置为高电平驱动输出，则在释放保留之前，它将继续驱动高电平。如果未能在释放保留之前重新初始化输出端口，则该引脚将暂时变为高阻抗引脚。在软件设置该引脚以驱动正确的输出状态之前，可能会在短时间内出现低电平干扰脉冲。

OSC：如果振荡器已使能为可在 VLLSx 或 LLS 模式下工作，则当 MCU 处于这些模式以及退出 VLLSx 低功耗模式时，它将继续工作。在这些低功耗模式下使能 OSC 的唯一理由是，通过振荡器的 EPCLK 输出为模块提供时钟。退出 VLLSx 模式时，控制该振荡器的寄存器会置位。在 VLLSx 模式下，实际控制信号会被锁存。释放保留之后，振荡器控制信号将使用寄存器值进行更新。如果未能在释放保留之前正确配置振荡器，则它将停止运行。

```
PMC_REGSC |= PMC_REGSC_ACKISO_MASK; //write to release hold on I/O
```

5.1.1 释放保留：条件或时间

在 LLS 模式下，I/O 和 OSC 上的保留会维持，并且无需软件干预，即可在退出模式时自动释放。

VLLSx 模式：在 VLLSx 模式下，I/O 和 OSC 上的保留会维持。在复位恢复过程中，调试引脚上的保留会自动释放，而其他引脚则会在软件对 PMC 模块中的 ACKISO 位执行写入操作时自动释放。

条件问题：

有时候，您可能并不希望释放保留。您可能想要检查内部条件，如实时时钟，然后返回至其中一种 VLLSx 低功耗模式。

时间问题：

当 MCU 从 VLLSx 模式唤醒时，如果您不希望干扰 I/O 或临时停止振荡器，则代码需要按顺序完成初始化流程，以便在释放保留之前重新初始化所有 I/O 和振荡器。当然，这取决于具体的应用。

1. 在此示例中，初始化振荡器模块和 MCG 振荡器控制位

```
// OSC_CR must enable external clock and enable in stop
OSC_CR = OSC_CR_EREFSTEN_MASK | OSC_CR_ERCLKEN_MASK;
```

2. 初始化所有数字 I/O（包括 GPIO）、串行接口、比较器驱动输出、定时器等。
3. 通过写入 ACKISO，释放保留。

由于所需顺序取决于具体的应用，因此示例代码或裸机工程并没有给出这种顺序示例。

5.2 识别复位类型

有多种方法可以识别先前章节中所示的新复位类型。SRS 寄存器（现在位于复位控制模块（RCM）中）以及 SMC 控制寄存器中的设置可以识别 MCU 是从哪种模式恢复的。

SRS 寄存器中的上电状态指示 POR 和 LVD 复位。如果 SRS 寄存器中的 POR 和 LVD 位已置位，则可以确定所有 MCU 寄存器均已设置为其默认状态，而 RAM 和寄存器文件寄存器未初始化。

如果在 SRS 寄存器中识别出其他复位源，您可以选择如何处理复位恢复。从 VLLS0、VLLS1、VLLS2 或 VLLS3 中恢复之后的复位流程会使 SRS 寄存器中的 WAKEUP 位置位。WAKEUP 位指示 MCU 通过复位从低功耗模式唤醒。获悉 WAKEUP 位已置位后，您可以解读 SMC 中的控制寄存器，如 PMPROT、PMCTRL、STOPCTRL 以及 VLLSCTRL（在某些 MCU 上），以识别所退出的低功耗模式。这些寄存器反映进入 MCU 低功耗模式时的状态，并且不会因 LLWU 复位清零。

5.3 以 C 语言初始化变量堆和栈空间

低功耗模式 LLS、LLS2、VLLS2 和 VLLS3 会在进入、处于低功耗模式期间以及恢复过程中保留部分或全部 RAM。寄存器文件模块、系统寄存器文件以及 RTC 寄存器文件会在所有休眠模式中保留，包括 VLLS0 和 VLLS1。MCU 还具有内置 Flash 或 EEPROM 存储器，可用于非易失性数据存储。

在管理这些存储器时，可以发挥创意。如果变量空间、堆和/或栈已保留，则在复位流程中无需进行重新初始化。

5.3.1 RAM 和寄存器文件保留

并非在所有低功耗模式下均会保留整个 RAM，但始终会保留寄存器文件内容。为此，我们提供了一个简表，其中显示了所有低功耗模式中每种模式的状态。该表为参考手册“功耗管理”章节中的“低功耗模式下的模块操作”表。参阅此表后，您将了解到在 VLLS2 模式下会保留 RAM，但在 VLLS1、VLLS0 或 MCU 断电时不会保留 RAM。

5.3.2 低功耗模式恢复检测

以下 C 语言代码用于检测每个复位源的 SRS 寄存器位。假设 SRS 寄存器中的唤醒位已置位，则会读取 SMC 模块寄存器，以确定 MCU 从哪种低功耗模式恢复。借助此信息，恢复 MCU 可以选择绕过部分惯用的复位初始化。

注

本示例所示为版本 2.x Kinetis 硅片，请参阅参考手册，了解所选 MCU 的位和寄存器定义。

```
if (RCM_SRS1 & RCM_SRS1_SW_MASK)
  printf("Software Reset\n");)
if (RCM_SRS1 & RCM_SRS1_LOCKUP_MASK)
  printf("Core Lockup Event Reset\n");
if (RCM_SRS1 & RCM_SRS1_JTAG_MASK)
```

```
  printf("JTAG Reset\n");
if (RCM_SRS0 & RCM_SRS0_POR_MASK)
  printf("Power-on Reset\n");
if (RCM_SRS0 & RCM_SRS0_PIN_MASK)
  printf("External Pin Reset\n");
if (RCM_SRS0 & RCM_SRS0_WDOG_MASK)
  printf("Watchdog(COP) Reset\n");
if (RCM_SRS0 & RCM_SRS0_LOC_MASK)
  printf("Loss of Clock Reset\n");
if (RCM_SRS0 & RCM_SRS0_LVD_MASK)
  printf("Low-voltage Detect Reset\n");
if (RCM_SRS0 & RCM_SRS0_WAKEUP_MASK)
{
  printf("[outSRS]Wakeup bit set from low power mode exit\n");
  printf("[outSRS]SMC_PMPROT    = %#02X \r\n", (SMC_PMPROT))   ;
  printf("[outSRS]SMC_PMCTRL    = %#02X \r\n", (SMC_PMCTRL))   ;
  printf("[outSRS]SMC_VLLSCTRL = %#02X \r\n", (SMC_VLLSCTRL));
  printf("[outSRS]SMC_PMSTAT    = %#02X \r\n", (SMC_PMSTAT))   ;

if ((SMC_PMCTRL & SMC_PMCTRL_STOPM_MASK)== 3)
          printf("[outSRS] LLS exit \n") ;
if (((SMC_PMCTRL & SMC_PMCTRL_STOPM_MASK)== 4) &&
    ((SMC_VLLSCTRL & SMC_VLLSCTRL_VLLSM_MASK)== 1))
          printf("[outSRS] VLLS1 exit \n") ;
if (((SMC_PMCTRL & SMC_PMCTRL_STOPM_MASK)== 4) &&
    ((SMC_VLLSCTRL & SMC_VLLSCTRL_VLLSM_MASK)== 2))
          printf("[outSRS] VLLS2 exit \n") ;
if (((SMC_PMCTRL & SMC_PMCTRL_STOPM_MASK)== 4) &&
   ((SMC_VLLSCTRL & SMC_VLLSCTRL_VLLSM_MASK)== 3))
          printf("[outSRS] VLLS3 exit \n") ;

}
```

5.4 启动顺序

4 种基本启动顺序如下：从 Flash 存储器正常启动、从 ROM 启动、在 EZPORT 模式下启动以及在已使能调试器的情况下复位。有关 Kinetis 器件的启动详情，请参阅参考手册。

从 Flash 启动有两个选项：正常启动和低功耗启动。两者的主要区别在于启动速度以及启动时消耗的电流。要控制此启动选项，需要不设置 FOPT 寄存器中的 LPBOOT 位以选择正常启动，或者配置 LPBOOT 位以选择低功耗启动模式。

对于具有 EZPORT 的 Kinetis 器件，如果在 POR 或 LVD 复位时 EZPORT 片选输入保持低电平，则会发生 EZPORT 启动。如不希望采用此模式，可以通过将 FOPT Flash 寄存器中对应的 EZPORT 禁用位清零来不选择使用 EZPORT 启动模式。该位清零之后，就不会发生意外进入 EZPORT 启动模式的情况。如果以后需要采用 EZPORT 模式，可以整片擦除 MCU 并对 FOPT 寄存器进行编程来指定新的设置。

从技术角度来说，调试模式启动并非一种单独的启动模式，此处引用的目的在于参考其对低功耗模式的影响。如果在调试器已连接并处于活动状态时复位 MCU，则部分 MCU 低功耗模式可能会作出与其在应用中正常操作模式下不同的响应。MCU 在 STOP 和 VLPS 模式下的电流读数高于其在应用任务模式（调试器断开）的读数，这是因为此时调试器模块时钟保持激活，以允许在 STOP 和 VLPS 模式下进行调试。

要确保应用处于任务模式，请断开调试器并重启 VDD 以产生 POR 复位。如果需要保持 RAM 的状态，则 POR 是做不到的，因此会进入 VLLS3 低功耗模式并恢复，且调试器会禁用。

6 动态和静态功耗管理

6.1 通过时钟控制进行功耗管理

为 MCU 电路提供时钟会产生功耗。谨慎控制系统时钟可对应用的平均电流消耗产生重大影响。

时钟频率越快，电路功耗越大。MCU 具有多个时钟和时钟控制，以便通过时钟控制进行 MCU 功耗管理。MCU 可采用内部或外部时钟源工作，并且可使用内部参考时钟、FLL 或 PLL 为 CPU、总线、外部总线、Flash 存储器和外设（在某些 Kinetis MCU 上）设置时钟速度。某些 MCU 上的 RTC 模块以及 RTC_OSC 为独立于内核、平台和其他外设的时钟域。这意味着，即便是在 VDD 或 MCU 关闭的情况下，RTC 仍可自主运行来计时或为 MCU 提供唤醒事件。

可以进行内部模块功耗管理控制。外设模块在 SIM 中具有时钟选通控制，SIM 可禁用未使用或闲置的模块时钟。这是一种模块级别的有效功耗管理控制。

6.2 使用低功耗模式进行功耗管理

Kinetis MCU 具有多达 11 种不同的功耗模式，其中包括 4 种动态功耗模式和多达 7 种静态功耗模式。为了使用这些功耗模式进行最有效的功耗管理，必须了解每种模式的权衡利弊：可用模块、唤醒时间长短、保留的存储器、关断的 MCU 部分以及 I/O 引脚的状态等。

MCU 参考手册中可以找到了解这些权衡利弊所需的资源。

7 低功耗模式下的时钟操作

MCU 中具有多个时钟。所有这些时钟均可在运行和等待功耗模式下运行。在部分低功耗模式下，可选择性地打开其中部分时钟。

MCU 的时钟由以下其中一种类型的时钟模块生成。有关特定时钟的详情，请参见 Kinetis 器件的参考手册。

1. 多用途时钟生成器模块（MCG）
2. 精简版多用途时钟生成器模块（MCG-Lite）
3. 内部时钟源（ICS）

也可在其他模块中生成时钟，包括但不限于：

1. 系统振荡器（OSC）
2. 实时时钟（RTC）
3. 功耗管理控制器（PMC）
4. USB 无晶体时钟源（48MHz IRC）

这些与 SIM 一起控制这些时钟在各种功耗模式下的时钟选择和分布。图 9 所示为这些模块互相配合的示例。如需进一步研究以及了解时钟模式转换的严格规定，请参见参考手册。

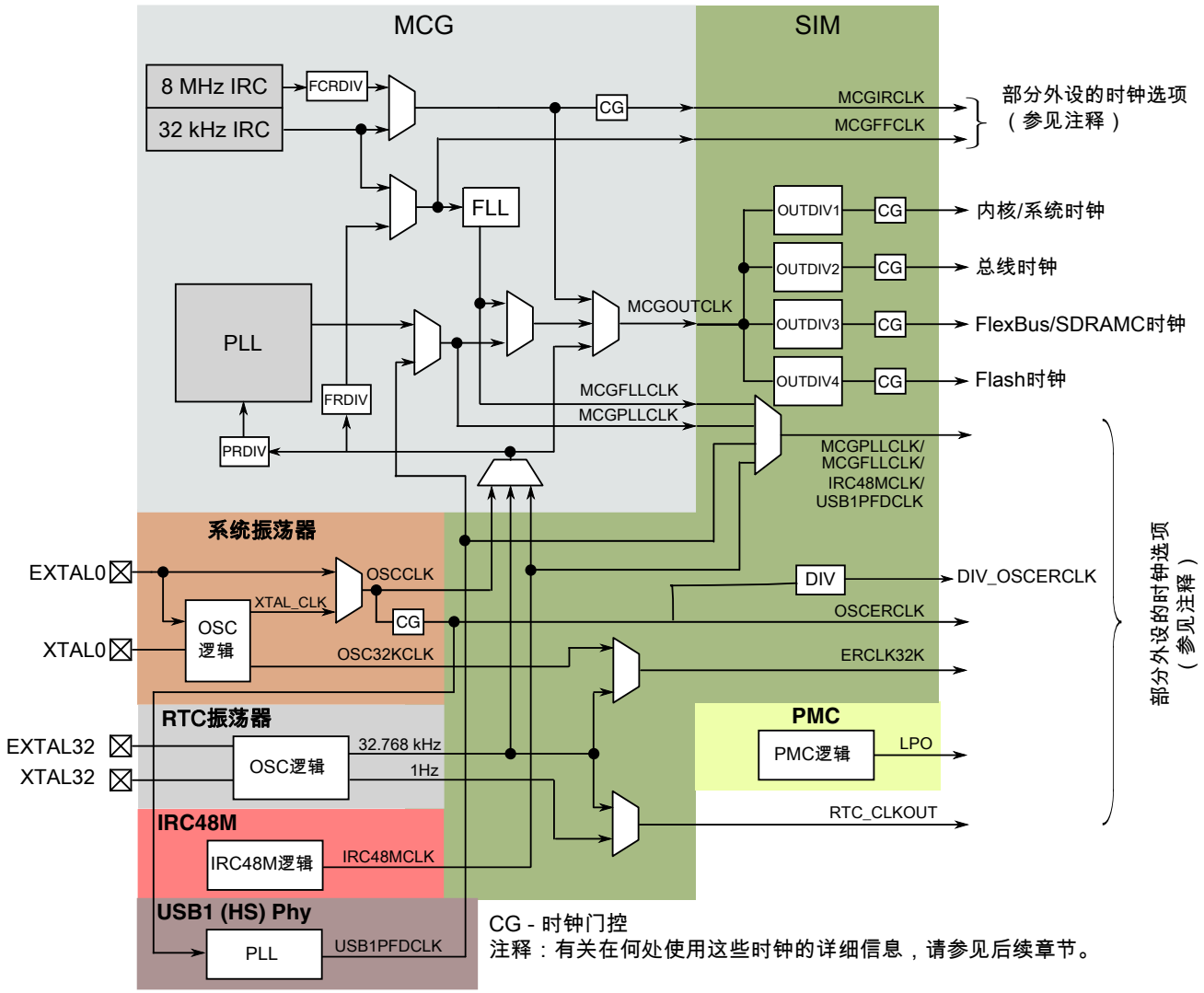


图 9. SOC 时钟图示例 - Kinetis K66

7.1 多用途时钟生成器（MCG）

MCG 是主时钟模块。它与振荡器模块一起工作，以满足 MCU 和外设模块的时钟需求。MCG 包含 1 个 FLL 和多达 2 个 PLL 时钟生成器。FLL 不太精确，用于在无重要接口计时需求时为 MCU 提供时钟。PLL 用于在需要此类准确时钟时为 CPU 和外设提供准确、低抖动时钟。PLL 仅可在以外部晶体或时钟输入源作为参考时钟时运行。

MCG 主系统时钟源的频率范围为 0 至 150 MHz。USB、I2S 和以太网模块还具有单独的 PLL 时钟路径。其他模块具有单独的 FLL 时钟路径。MCGIRCLK 为内部生成的时钟，可供段式 LCD、LPT 和 TSI 模块使用。MCGFFCLK 可供 Flex Timer（FTM）使用。

某些 Kinetis MCU 上提供称为 IRC48 的高速 IRC。此时钟可以为 MCU 提供时钟，并且还能为处于设备模式的 USB 模块提供时钟，而不需外部晶体。

7.2 MCG Lite

所示为 MCG_Lite 框图。在该模块中，第一个搭载此模块的器件为 Kinetis KL03

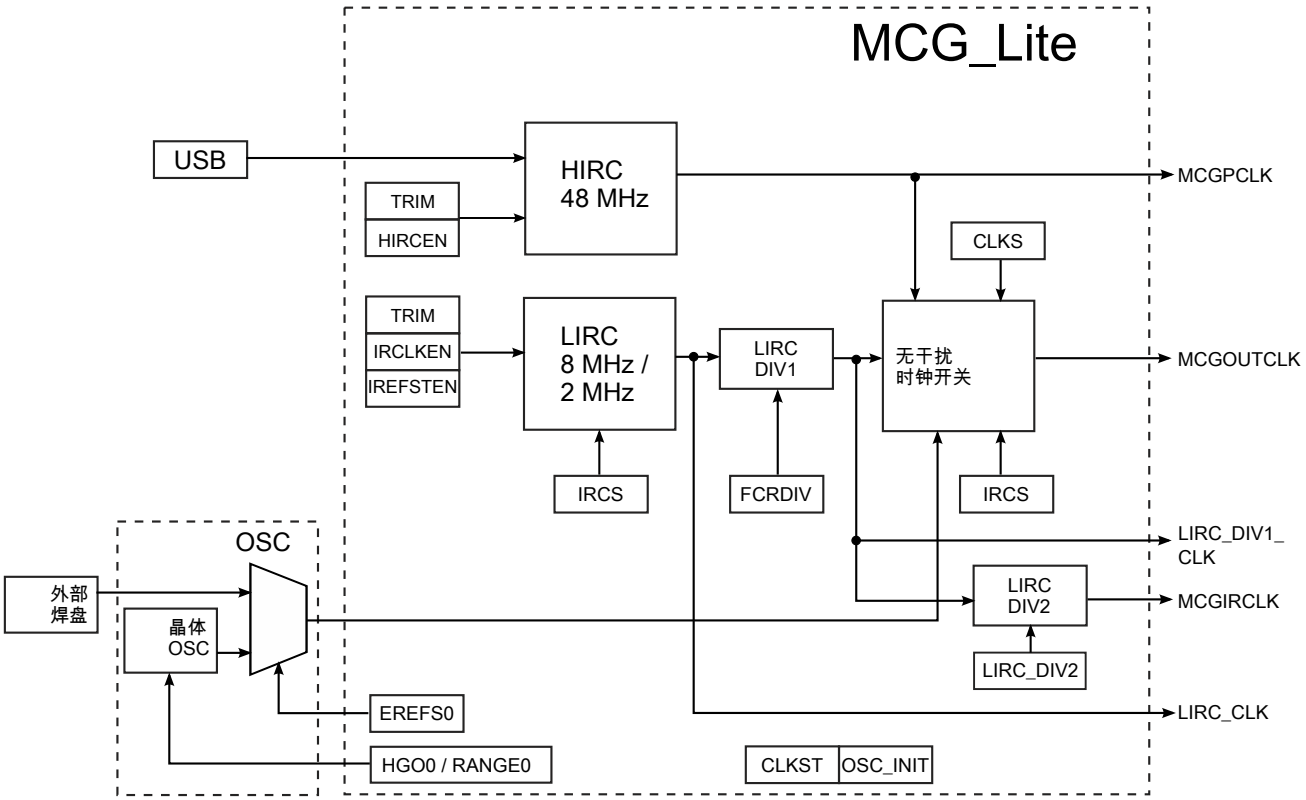


图 10. SOC 时钟图 MCG LITE - Kinetis KL03

7.3 系统振荡器（OSC）

从 OSC 生成的 OSCERCLK 可供 FlexCAN、LPT 和 ADC 使用。内部 ERCLK32K 可供段式 LCD、LPT 和 TSI 使用。

7.4 RTC 振荡器（RTC）

OSC 逻辑的输出会发送至实时时钟（RTC），并且可通过 ERCLK32K 用于其他模块。RTC 振荡器还可用作 FLL（但不是 PLL）的时钟源，并且可通过 MCGOUTCLK 路由至系统其余部分。

7.5 功耗管理控制器（PMC）时钟

PMC 为低功耗振荡器（LPO）的来源，LPO 可用作多个模块的时钟源。LPO 的标称工作频率为 1 KHz +/– 30%。它温度和电压的波动使其不十分准确，但可在所有功耗模式下运行。此时钟可以作为 MCU 中多个模块的时钟源，包括 LPTMR、复位滤波器、LLWU 引脚滤波器和 LCD 控制器，以便这些模块可以在最低功耗模式下具有时钟。请参见参考手册中各模块对应的配置章节。可用时钟选项如表中详述。

7.6 SIM 中的时钟控制

SIM 可控制系统时钟分频器和各种时钟源多路复用器。SIM 还可对器件中各模块进行时钟选通控制。有关详情，请参见各器件参考手册中的 SIM 和时钟分配章节。

7.6.1 SIM 中外设模块的时钟选通控制

SIM 中存在多达 7 个时钟选通控制寄存器。大多数时钟选通使能位可通过复位清零，少数除外。其中一个是用于控制 FTFx 的位，即 Flash 存储器的时钟使能位。

在通过读取或写入操作访问任意模块之前，您需要使能该模块的时钟选通，否则将会产生硬故障复位。如果不再需要某个模块，或者该模块将闲置较长时间，则可将对应的时钟选通位清零。这将降低该模块的功耗。

为了最大程度地降低 MCU 的总体功耗，通常仅使能当前操作所需模块的时钟选通。这样一来，您就可以与当时所需任务的功耗预算保持一致。

如果模块需要时钟才能工作（例如 UART），则禁用时钟选通将导致该模块停止工作。

注

FTFx 时钟选通位需要特殊处理。如果在已清除 FTFx 的情况下进入 LLS 模式，则 LLWU 唤醒服务程序和中断向量必须位于内部 RAM 中，才可正确唤醒 MCU。如果从 LLWU 服务程序中退出时需要访问 Flash，则在退出 ISR 之前必须先重新使能 FTFx 时钟选通。

7.7 高速运行、运行和等待功耗模式下的时钟操作

高速运行模式可以实现芯片最高性能。与正常运行模式相比，在此状态下，MCU 可以更高频率运行

SIM 具有多个时钟控制。其中一个最强大的时钟控制为 SIMCLKDIV 寄存器，该寄存器允许在运行模式下进行时钟分频控制，以便实现动态频率调整，从而可根据现有性能需求调整系统功耗。运行和等待模式下的最大时钟速度由 MCU 规格决定。

7.8 VLPR 时钟注意事项

要从运行模式进入 VLPR 模式，必须确保满足 VLPR 时钟要求。由于时钟速度受限，串行波特率和定时器时基也会受限。

VLPW 仅可从 VLPR 进入并且具有完全相同的时钟限制。尽管可直接从运行模式进入 VLPS，但在 VLPS 中获得时钟的任何外设均不得超过 VLPR 中规定的最大频率。有关特定器件的信息，请参见各器件数据手册。

在 VLPR 模式下，不得更改 SIMCLKDIV 系统时钟分频器。必须在进入 VLPR 之前配置所需的系统时钟频率。进入 VLPR 后，无法像在运行模式下那样抑制时钟速率。

可用于 VLPR 和 VLPW 模式的时钟模式为 BLPE、BLPI 或 LIRC。BLPE 模式采用外部时钟源。

在某些器件上，可对 FOPT 寄存器中的 LPBOOT 位进行配置，以便在从复位退出后直接进入 VLPR 模式。SIMCLKDIV 分频器的配置应确保不会超过最大 VLPR 系统时钟频率。

8 功耗模式转换

8.1 进入低功耗模式

图 11 所示为允许的功耗模式转换。任何复位均会使芯片返回至 LPBOOT 位所配置的默认运行状态。在运行、等待和停止模式下，均已使能有效功率调节。VLPR 和 VLPW 模式下频率受限，但可提供低于正常模式下功耗的低功耗操作模式。LLS 和 VLLSx 模式为最低功耗停止模式，并应根据应用要保留的逻辑或存储器数量进行选择。

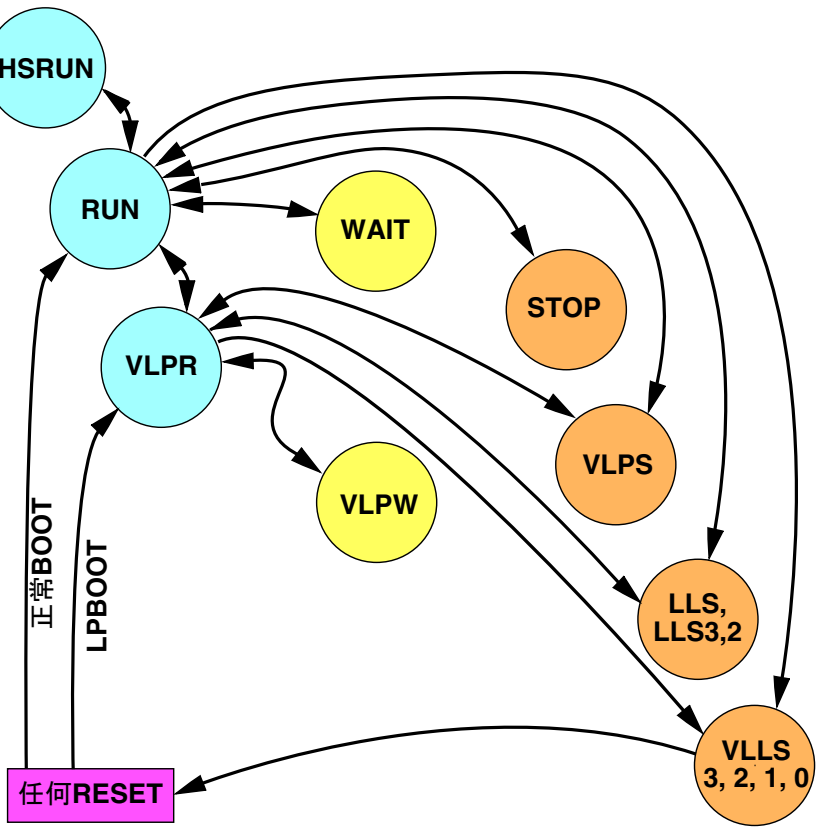


图 11. 功耗模式状态转换示意图

8.1.1 每种低功耗模式的进入和退出条件

表 3. 功率模式转换触发

转换编号	从	到	触发条件
1	RUN	WAIT	Sleep-now 模式或 Sleep-on-exit 模式可在 SLEEPDEEP 清零时进入，在 ARM 内核的系统控制寄存器中进行控制。 参见注释。[1]
	WAIT	RUN	中断或复位
2	RUN	STOP	PMCTRL[RUNM]=00，PMCTRL[STOPM]=000

下一页继续介绍此表...

表 3. 功率模式转换触发 (继续)

转换编号	从	到	触发条件
			[2] Sleep-now 模式或 Sleep-on-exit 模式可在 SLEEPDEEP 置位时进入，在 ARM 内核的系统控制寄存器中进行控制。
	STOP	RUN	中断或复位
3	RUN	VLPR	内核、系统、总线和 Flash 时钟频率和 MCG、MCG_Lite 时钟模式在此模式下受限。有关支持的最大允许频率以及 MCG、MCG_Lite 模式的详情，请参见“功耗管理”章节。 设置 PMPROT[AVLP]=1，PMCTRL[RUNM]=10。 注：要限制峰值电流，可通过 Flash IFR 设置在任意复位时设定 PMPROT[AVLP]和 PMCTRL[RUNM]位，从而使 SMC 在复位恢复过程中将 MCU 从 RUN 转换到 VLPR 模式。
	VLPR	RUN	设置 PMCTRL[RUNM]=00 或 PMCTRL[LPWUI] =1 时中断，或 复位。 注：并非所有器件都具有 PMCTRL[LPWUI]位。
4	VLPR	VLPW	Sleep-now 模式或 Sleep-on-exit 模式可在 SLEEPDEEP 清零时进入，在 ARM 内核的系统控制寄存器中进行控制。
	VLPW	VLPR	PMCTRL[LPWUI]=0 时中断 注：并非所有器件都具有 PMCTRL[LPWUI]位。
5	VLPW	RUN	PMCTRL[LPWUI]=1 时中断，或 复位 注：并非所有器件都具有 PMCTRL[LPWUI]位。这些器件不存在 LPWUI = 1 的情况。
6	VLPR	VLPS	PMCTRL[STOPM]=000 [3] 或 010， Sleep-now 模式或 Sleep-on-exit 模式可在 SLEEPDEEP 置位时进入，在 ARM 内核的系统控制寄存器中进行控制。
	VLPS	VLPR	PMCTRL[LPWUI]=0 时中断 注：如果 VLPS 是从 RUN 直接进入的（转换编号为 7），则硬件将强制退回到 RUN，并不允许转换到 VLPR。并非所有器件都具有 PMCTRL[LPWUI]位。
7	RUN	VLPS	PMPROT[AVLP]=1，PMCTRL[STOPM]=010 Sleep-now 模式或 Sleep-on-exit 模式可在 SLEEPDEEP 置位时进入，在 ARM 内核的系统控制寄存器中进行控制。
	VLPS	RUN	PMCTRL[LPWUI]=1 时中断，或 PMCTRL[LPWUI]=0 时中断，并且直接从 RUN 进入 VLPS 模式，或 复位 注：并非所有器件都具有 PMCTRL[LPWUI]位。
8	RUN	VLLSx	PMPROT[AVLLS]=1，PMCTRL[STOPM]=100，STOPCTRL[VLLSM]=x (VLLSx)，STOPE=1，WAITE=0

下一页继续介绍此表...

表 3. 功率模式转换触发 (继续)

转换编号	从	到	触发条件
			PMPROT[AVLLS]=1，PMCTRL[STOPM]=100，STOPCTRL[VLLSM]=x (VLLSx)，Sleep-now 模式或 Sleep-on-exit 模式可在 SLEEPDEEP 置位时进入，在 ARM 内核的系统控制寄存器中进行控制。
	VLLSx	RUN	从已使能的 LLWU 输入源或 RESET 引脚唤醒
9	VLPR	VLLSx	PMPROT[AVLLS]=1，PMCTRL[STOPM]=100，STOPCTRL[VLLSM]=x (VLLSx)，STOPE=1，WAITE=0 PMPROT[AVLLS]=1，PMCTRL[STOPM]=100，STOPCTRL[VLLSM]=x (VLLSx)，Sleep-now 模式或 Sleep-on-exit 模式可在 SLEEPDEEP 置位时进入，在 ARM 内核的系统控制寄存器中进行控制。
10	RUN	LLS	PMPROT[ALLS]=1，PMCTRL[STOPM]=011，STOPE=1，WAITE=0 Sleep-now 模式或 Sleep-on-exit 模式可在 SLEEPDEEP 置位时进入，在 ARM 内核的系统控制寄存器中进行控制。
11	LLS	RUN	从已使能的 LLWU 输入源唤醒且从 RUN 直接进入 LLS 模式，或 RESET 引脚。
12	VLPR	LLS	PMPROT[ALLS]=1，PMCTRL[STOPM]=011，STOPE=1，WAITE=0 Sleep-now 模式或 Sleep-on-exit 模式可在 SLEEPDEEP 置位时进入，在 ARM 内核的系统控制寄存器中进行控制。
	LLS	VLPR	在部分器件上实施。从已使能的 LLWU 输入源唤醒且从 VLPR 直接进入 LLS 模式 注：如果 LLS 是从 RUN 直接进入的，则硬件将不允许这次转换，并将强制返回 RUN
13	RUN	HSRUN	设置 PMPROT[AHSRUN]=1，PMCTRL[RUNM]=11。
	HSRUN	RUN	设置 PMCTRL[RUNM]=00 复位

1. 如果已使能调试，则内核时钟保持支持调试。
2. 如果 PMCTRL[STOPM]=000 并且 STOPCTRL[PSTOPO]=01 或 10，则仅进入局部停止模式，而非进入 STOP 模式
3. 如果 PMCTRL[STOPM]=000 且 STOPCTRL[PSTOPO]=00，则会进入 VLPS 模式而非 STOP 模式。如果 PMCTRL[STOPM]=000 且 STOPCTRL[PSTOPO]=01 或 10，则仅进入局部停止模式，而非进入 VLPS 模式

9 功耗模式进入代码

9.1 功耗模式转换代码

9.1.1 所含内容和假定条件

后续几个章节给出了进入不同低功耗模式的示例代码。简短的代码片段展示了进入低功耗模式的简易性。

此处提供了进入模式所需的函数。虽然提供的注释说明了执行此代码时应预先完成的操作，但却省略了进入相关模式之前所需的任何初始化工作。

9.1.2 功耗模式进入驱动程序的函数原型

```
/*****************************************************************/
// function prototypes
void sleep(void);
void deepsleep(void);
void enter_wait(void);
void enter_stop(void);
int enter_vlpr(char lpwui_value);
void exit_vlpr(void);
void enter_vlps(void);
void enter_lls(void);
void enter_lls2(void);
void enter_lls3(void);
void enter_vlls3(void);
void enter_vlls2(void);
void enter_vlls1(void);
void enter_vlls0(unsigned char PORPO_value);
/*****************************************************************/
```

9.1.3 进入睡眠模式 — 创建用于进入睡眠模式的函数

ARM Cortex-M4 和 M0+内核使用 SCR 中 SLEEPDEEP 位的状态来控制执行 WFI 指令时内核平台进入的状态。如果 SLEEPDEEP 位清零，则会进入睡眠或等待模式。

```
/*****************************************************************/
/*
 * Configures the ARM system control register for WAIT(sleep)mode
 * and then executes the WFI instruction to enter the mode.
 *
 * Parameters:
 * none
 *
 */

void sleep (void)
{
/* Clear the SLEEPDEEP bit to make sure we go into WAIT (sleep)
 * mode instead of deep sleep.
 */
    SCB_SCR &=  ~SCB_SCR_SLEEPDEEP_MASK;
#ifdef CMSIS
    __wfi();
#else
/* WFI instruction will start entry into WAIT mode */
asm("WFI");
#endif
}
/*****************************************************************/
```

9.1.4 进入深度睡眠模式 — 创建用于进入深度睡眠模式的函数

ARM Cortex-M4 和 M0+内核使用 SCR 中 SLEEPDEEP 位的状态来控制执行 WFI 指令时内核平台进入的状态。如果 SLEEPDEEP 位置位，则会进入深度睡眠或内核停止模式。

```
/*****************************************************************/
/*
 * Configures the ARM system control register for STOP
 * (deepsleep) mode and then executes the WFI instruction
 * to enter the mode.
 *
 * Parameters:
 * none
 *
 */

void deepsleep (void)
{
  /* Set the SLEEPDEEP bit to enable deep sleep mode (STOP) */
  SCB_SCR |= SCB_SCR_SLEEPDEEP_MASK;

#ifdef CMSIS
  __wfi();
#else
  /* WFI instruction will start entry into STOP mode */
  asm("WFI");
#endif
}
/*****************************************************************/
```

9.1.5 进入等待模式或 VLPW

MCU 可通过以下代码进入等待或 VLPW 模式。进入的模式取决于当时所使用的运行模式。如果为运行模式，则进入等待模式；如果为 VLPR 模式，则进入 VLPW 模式。

如果看门狗已使能，则必须通过某种方式唤醒器件以为看门狗服务，否则将会通过看门狗复位唤醒。

前提条件：

- 如果要进入 VLPW 模式，则在调用此函数之前，必须禁用 MCG 中的时钟监视器。例如，如果 MCG 处于 BLPE 模式，那么在进入 VLPR 或 VLPW 功耗模式之前，还应向 CME0 写入逻辑 0。
- 调用此函数之前，必须设置要用于将 MCU 从等待或 VLPW 模式唤醒的唤醒事件。

```
/*****************************************************************/
/* WAIT mode entry routine. Puts the processor into Wait mode.
 * In this mode the core clock is disabled (no code executing),
 * but bus clocks are enabled (peripheral modules are
 * operational)
 *
 * Mode transitions:
 * RUN to WAIT
 * VLPR to VLPW
 *
 * This function can be used to enter normal wait mode or VLPW
 * mode. If you are executing in normal run mode when calling
 * this function, then you will enter normal wait mode.
 * If you are in VLPR mode when calling this function,
 * then you will enter VLPW mode instead.
 *
 * NOTE: Some modules include a programmable option to disable
 * them in wait mode. If those modules are programmed to disable
 * in wait mode, they will not be able to generate interrupts to
 * wake the core.
 *
 * WAIT mode is exited using any enabled interrupt or RESET,
 * so no exit_wait routine is needed.
 * For Kinetis K:
 * If in VLPW mode, the statue of the SMC_PMCTRL[LPWUI] bit
 * determines if the processor exits to VLPR (LPWUI cleared)
```

```
 * or normal run mode (LPWUI set). The enable_lpwui()
 * and disable_lpwui()functions can be used to set this bit
 * to the desired option prior to calling enter_wait().
 * For Kinetis L:
 * LPWUI does not exist.
 * Exits with an interrupt from VLPW will always be back to VLPR.
 * Exits from an interrupt from Wait will always be back to Run.
 *
 * Parameters:
 * none
 */
void enter_wait(void)
{
    sleep();
}
/*****************************************************************/
```

9.1.6 进入正常停止模式

MCU 可通过以下代码从运行模式进入正常停止模式。

对于 Kinetis L，如果从 VLPR 模式开始，则 MCU 会进入 VLPS 而非停止模式。

如果看门狗已使能，则必须通过某种方式唤醒器件以为看门狗服务，否则将会通过看门狗复位唤醒。

此时 MCU 始终会从停止模式退出到运行模式。

前提条件:

- 调用此函数之前，必须禁用 MCG 中的时钟监视器。
- 必须处理不会对停止模式进入指令作出应答的所有模块。有关需要处理的模块，请参见参考手册。
- 要进入停止模式，请设置 PMCTRL[RUNM]=00，PMCTRL[STOPM]=02。
- 如果 STOPCTRL 寄存器中的 PSTOPO 位设为'b01 或'b10，则会进入局部停止模式而非 STOP 模式
- 调用此函数之前，必须设置要用于将 MCU 从停止或 VLPS 模式唤醒的唤醒事件。

```
/*****************************************************************/
/* STOP mode entry routine.
 * if in Run mode puts the processor into normal stop mode.
 * If in VLPR mode puts the processor into VLPS mode.
 * In this mode core, bus and peripheral clocks are disabled.
 *
 * Mode transitions:
 * RUN to STOP
 * VLPR to VLPS
 *
 * This function can be used to enter normal stop mode.
 * If you are executing in normal run mode when calling this
 * function and AVLP = 0, then you will enter normal stop mode.
 * If AVLP = 1 with previous write to PMPROT
 * then you will enter VLPS mode instead.
 *
 * STOP mode is exited using any enabled interrupt or RESET,
 * so no exit_stop routine is needed.
 *
 * Parameters:
 * none
 */
void enter_stop(void)
{
  volatile unsigned int dummyread;
  /*The PMPROT register may have already been written by init
     code. If so, then this next write is not done since
     PMPROT is write once after RESET
     this write-once bit allows the MCU to enter the
     normal STOP mode.
```

```
     If AVLP is already a 1, VLPS mode is entered
     instead of normal STOP
     is SMC_PMPROT = 0  */

    /* Set the STOPM field to 0b000 for normal STOP mode
       For Kinetis L: if trying to enter Stop from VLPR user
       forced to VLPS low power mode */
    SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK;
    SMC_PMCTRL |=  SMC_PMCTRL_STOPM(0);
    /*wait for write to complete to SMC before stopping core */
    dummyread = SMC_PMCTRL;
    deepsleep();
}
/*******************************************************************/
```

9.1.7 进入 VLPR 模式

MCU 可通过以下代码进入 VLPR 模式。在 Kinetis K 上，LPWUI 的值会作为参数传递给此函数。LPWUI 在 VLPR、VLPW 和 VLPS 模式下为静态，并且在 VLPR 模式下不得进行修改。对于 Kinetis L，传递给此程序的值无关重要。

前提条件:

- 必须降低时钟频率。某些 MCU 可以在系统、总线和 FlexBus 时钟频率为 4 MHz 且 Flash 时钟频率为 1 MHz 时工作。

```
//core = /1 bus = /2 flexbus = /2 flash clk =/4
SIM_CLKDIV1  =  (SIM_CLKDIV1_OUTDIV1(1) |
                 SIM_CLKDIV1_OUTDIV2(1) |
                 SIM_CLKDIV1_OUTDIV3(1) |
                 SIM_CLKDIV1_OUTDIV4(3));
```

- 调用此函数之前，必须禁用 MCG 中的时钟监视器。例如，如果 MCG 处于 BLPE 模式，那么在进入 VLPR 或 VLPW 功耗模式之前，还应向 CME0 写入逻辑 0。
- 必须禁用不会对停止模式进入指令作出应答的所有模式。

```
/*******************************************************************/
/* VLPR mode entry routine.Puts the processor into Very Low Power
 * Run Mode. In this mode, all clocks are enabled,
 * but the core, bus, and peripheral clocks are limited
 * to 2 or 4 MHz or less.
 * The flash clock is limited to 1MHz or less.
 *
 * Mode transitions:
 * RUN to VLPR
 *
 * For Kinetis K:
 * While in VLPR, VLPW or VLPS the exit to VLPR is determined by
 * the value passed in from the calling program.
 * LPWUI is static during VLPR mode and
 * should not be written to while in VLPR mode.
 *
 * For Kinetis L:
 * LPWUI does not exist. the parameter pass is a don't care
 * Exits with an interrupt from VLPW will always be back to VLPR.
 * Exits from an interrupt from Wait will always be back to Run.
*
 * Parameters:
 * lpwui_value - The input determines what is written to the
 *               LPWUI bit in the PMCTRL register
 *               Clear LPWUI and interrupts keep you in VLPR
 *               Set LPWUI and interrupts return you to Run mode
 * Return value : PMSTAT value or error code
 *                PMSTAT = 000_0100 Current power mode is VLPR
 *                ERROR Code =  0x14 - already in VLPR mode
 *                           =  0x24 - REGONS never clears
 *                              indicating stop regulation
```

```
 */
int enter_vlpr(char lpwui_value)
{
  int i;
  if ((SMC_PMSTAT & SMC_PMSTAT_PMSTAT_MASK)== 4){
         return 0x14;
         }

  /* The PMPROT register may have been written by init code
     If so, then this next write is not done
     PMPROT is write once after RESET
     This write-once bit allows the MCU to enter the
     very low power modes: VLPR, VLPW, and VLPS   */
  SMC_PMPROT = SMC_PMPROT_AVLP_MASK;

  /* Set the (for MC1)LPLLSM or (for MC2)STOPM field
     to 0b010 for VLPS mode -
     and RUNM bits to 0b010 for VLPR mode
     Need to set state of LPWUI bit */
  lpwui_value &= 1;
  SMC_PMCTRL &= ~SMC_PMCTRL_RUNM_MASK;
  SMC_PMCTRL |= SMC_PMCTRL_RUNM(0x2) |
                (lpwui_value << SMC_PMCTRL_LPWUI_SHIFT);
  /* OPTIONAL Wait for VLPS regulator mode to be confirmed */
  for (i = 0 ; i < 10 ; i++)
    {     /* Check that the value of REGONS bit is not 0
             When it is a zero, you can stop checking */
      if ((PMC_REGSC & PMC_REGSC_REGONS_MASK) ==0x04){
       /* 0 Regulator is in stop regulation or in transition
             to/from it
          1 MCU is in Run regulation mode */
      }
      else  break;
    }
  if ((PMC_REGSC & PMC_REGSC_REGONS_MASK) ==0x04) {
       return 0x24;
     }
  /* SMC_PMSTAT register only exist in Mode Controller 2 */
  if ((SMC_PMSTAT & SMC_PMSTAT_PMSTAT_MASK) == 4) {
       return 0x04;
     }
}
/*****************************************************************/

/*  Use this statement in-line with your code to put
 *  the MCU into VLPR
 *  This assumes
 *  1)that the clocks are setup properly
 *  2)that you are entering VLPR from Run mode */

/*  add the next line if not already written. */
//  SMC_PMPROT = SMC_PMPROT_AVLP_MASK;

  SMC_PMCTRL = SMC_PMCTRL_RUNM(0x2);
```

9.1.8 进入 VLPS 模式

MCU 可通过以下代码从运行或 VLPR 模式进入 VLPS 模式。

如果看门狗已使能，则必须通过某种方式喚醒器件以为看门狗服务，否则将会通过看门狗复位喚醒。

前提条件:

- 调用此函数之前，必须禁用 MCG 中的时钟监视器。例如，在 MCG 进入任何停止模式之前，应向 CME0 位写入逻辑 0。否则，可能会在停止模式下发生复位请求。
- 要进入 VLPS 模式，则必须置位 PMPROT 寄存器中的 ALVP 位，且 PMCTRL 寄存器中的 STOPM 位必须为 'b10。

```
/******************************************************************/
/* VLPS mode entry routine. Puts the processor into VLPS mode
 * directly from run or VLPR modes.
 *
 * Mode transitions:
 * RUN  to VLPS
 * VLPR to VLPS
 *
 * Kinetis K:
 * when VLPS is entered directly from RUN mode,
 * exit to VLPR is disabled by hardware and the system will
 * always exit back to RUN.
 *
 * If however VLPS mode is entered from VLPR the state of
 * the LPWUI bit determines the state the MCU will return
 * to upon exit from VLPS.If LPWUI is 1 and an interrupt
 * occurs you will exit to normal run mode instead of VLPR.
 * If LPWUI is 0 and an interrupt occurs you will exit to VLPR.
 *
 * For Kinetis L:
 * when VLPS is entered from run an interrupt will exit to run.
 * When VLPS is entered from VLPR an interrupt will exit to VLPS
 * Parameters:
 * none
 */
/******************************************************************/

void enter_vlps(void)
{
  volatile unsigned int dummyread;
  /*The PMPROT register may have already been written by init
     code. If so then this next write is not done since
     PMPROT is write once after RESET.
     This Write allows the MCU to enter the VLPR, VLPW,
     and VLPS modes. If AVLP is already written to 0
     Stop is entered instead of VLPS*/
  SMC_PMPROT = SMC_PMPROT_AVLP_MASK;
  /* Set the STOPM field to 0b010 for VLPS mode */
  SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK;
  SMC_PMCTRL |=  SMC_PMCTRL_STOPM(0x2);
  /*wait for write to complete to SMC before stopping core */
  dummyread = SMC_PMCTRL;
  /* Now execute the stop instruction to go into VLPS */
  deepsleep();
}
/******************************************************************/
```

9.1.9 进入 LLS 模式

MCU 可通过以下代码从运行或 VLPR 模式进入 LLS 模式。

前提条件：

- 必须禁用不会对停止模式进入指令作出应答的所有模式。
- 调用此函数之前，必须禁用 MCG 中的时钟监视器。
- 要进入 LLS 模式，必须置位 PMPROT 寄存器中的 ALLS 位。
- 如果 PMPROT 寄存器中的 ALLS 和 AVLP 位为 0，则会忽略对 PMCTRL 中位的写入操作。

- 所有 I/O 都保留 MCU 处于 LLS 模式时的状态，并在唤醒时自动释放。
- 调用此函数之前，必须在 LLWU 和唤醒模块中设置要用于将 MCU 从 LLS 模式唤醒的唤醒事件。有关设置 LLWU 以通过 PTE1 引脚唤醒的示例，请参见本应用笔记的 LLS 模式进入和退出 - 裸机代码示例 章节。

```
/******************************************************************/
/* LLS mode entry routine. Puts the processor into LLS mode from
 * normal Run mode or VLPR.
 *
 * Mode transitions:
 * RUN to LLS
 * VLPR to LLS
 *
 * Wake-up from LLS mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in LLS mode, so make
 * sure to set up the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * none
 */
void enter_lls(void)
{
   volatile unsigned int dummyread;
   /* Write to PMPROT to allow LLS power modes this write-once
      bit allows the MCU to enter the LLS low power mode*/
   SMC_PMPROT = SMC_PMPROT_ALLS_MASK;
   /* Set the (for MC1) LPLLSM or
     (for MC2)STOPM field to 0b011 for LLS mode
      Retains LPWUI and RUNM values */
   SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK ;
   SMC_PMCTRL |= SMC_PMCTRL_STOPM(0x3) ;
    /* set LLSM = 0b00 in SMC_VLLSCTRL (for MC4) */
     SMC_VLLSCTRL =  SMC_VLLSCTRL_LLSM(0);
   /*wait for write to complete to SMC before stopping core */
   dummyread = SMC_PMCTRL;
   /* Now execute the stop instruction to go into LLS */
   deepsleep();
}
/******************************************************************/
```

9.1.10 进入 LLS2 模式

MCU 可通过以下代码从运行或 VLPR 模式进入 LLS2 模式。

MCU 从 LLS2 退出后始终会进入运行模式。

前提条件:

- 必须禁用不会对停止模式进入指令作出应答的所有模式。
- 调用此函数之前，必须禁用 MCG 中的时钟监视器。
- 要进入 LLS2 模式，必须置位 PMPROT 寄存器中的 ALLS 位。
- 如果 PMPROT 寄存器中的 ALLS 和 AVLP 位为 0，则会忽略对 PMCTRL 中位的写入操作。
- 所有 I/O 都保留 MCU 处于 LLS 模式时的状态，并在唤醒时自动释放。
- 调用此函数之前，必须在 LLWU 和唤醒模块中设置要用于将 MCU 从 LLS 模式唤醒的唤醒事件。有关设置 LLWU 以通过 PTE1 引脚唤醒的示例，请参见本应用笔记的 LLS 模式进入和退出 - 裸机代码示例 章节。

```
/******************************************************************/
/* LLS mode entry routine. Puts the processor into LLS mode from
 * normal Run mode or VLPR.
 *
 * Mode transitions:
 * RUN to LLS2
 * VLPR to LLS2
```

```
 *
 * NOTE: LLS2 mode will always exit to Run mode even if you were
 * in VLPR mode before entering LLS2.
 *
 * Wake-up from LLS2 mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in LLS mode, so make
 * sure to set up the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * none
 */
void enter_lls2(void)
{
   volatile unsigned int dummyread;
   /* Write to PMPROT to allow LLS power modes this write-once
      bit allows the MCU to enter the LLS low power mode*/
   SMC_PMPROT = SMC_PMPROT_ALLS_MASK;
   /* Set the (for MC1) LPLLSM or
     (for MC2)STOPM field to 0b011 for LLS3 mode
      Retains LPWUI and RUNM values */
   SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK ;
   SMC_PMCTRL |= SMC_PMCTRL_STOPM(0x3) ;
    /* set LLSM = 0b10 in SMC_VLLSCTRL (for MC4) */
     SMC_VLLSCTRL =  SMC_VLLSCTRL_LLSM(2);

   /*wait for write to complete to SMC before stopping core */
   dummyread = SMC_PMCTRL;
   /* Now execute the stop instruction to go into LLS */
   deepsleep();
}
/*****************************************************************/
```

9.1.11 进入 LLS3 模式

MCU 可通过以下代码从运行或 VLPR 模式进入 LLS3 模式。

前提条件:

- 必须禁用不会对停止模式进入指令作出应答的所有模式。
- 调用此函数之前，必须禁用 MCG 中的时钟监视器。
- 要进入 LLS3 模式，必须置位 PMPROT 寄存器中的 ALLS 位。
- 如果 PMPROT 寄存器中的 ALLS 和 AVLP 位为 0，则会忽略对 PMCTRL 中位的写入操作。
- 所有 I/O 都保留 MCU 处于 LLS3 模式时的状态，并在唤醒时自动释放。
- 调用此函数之前，必须在 LLWU 和唤醒模块中设置要用于将 MCU 从 LLS3 模式唤醒的唤醒事件。有关设置 LLWU 以通过 PTE1 引脚唤醒的示例，请参见本应用笔记的 LLS 模式进入和退出 - 裸机代码示例 章节。

```
/*****************************************************************/
/* LLS3 mode entry routine. Puts the processor into LLS3 mode from
 * normal Run mode or VLPR.
 *
 * Mode transitions:
 * RUN to LLS3
 * VLPR to LLS3
 *
 * Wake-up from LLS3 mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in LLS3 mode, so make
 * sure to set up the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * none
 */
void enter_lls3(void)
```

```
{
   volatile unsigned int dummyread;
   /* Write to PMPROT to allow LLS power modes this write-once
      bit allows the MCU to enter the LLS low power mode*/
   SMC_PMPROT = SMC_PMPROT_ALLS_MASK;
   /* Set the (for MC1) LPLLSM or
     (for MC2)STOPM field to 0b011 for LLS3 mode
      Retains LPWUI and RUNM values */
   SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK ;
   SMC_PMCTRL |= SMC_PMCTRL_STOPM(0x3) ;
    /* set LLSM = 0b11 in SMC_VLLSCTRL (for MC4) */
     SMC_VLLSCTRL =  SMC_VLLSCTRL_LLSM(3);
   /*wait for write to complete to SMC before stopping core */
   dummyread = SMC_PMCTRL;
   /* Now execute the stop instruction to go into LLS */
   deepsleep();
}
/*****************************************************************/
```

9.1.12 进入 VLLS3 模式

MCU 可通过以下代码从运行或 VLPR 模式进入 VLLS3 模式。

MCU 从 VLLS3 退出后始终会进入复位模式。

前提条件:

- 要进入 VLLS3 模式，必须置位 PMPROT 寄存器中的 AVLLS 位。
- 如果 PMPROT 寄存器中的 AVLLS 和 AVLP 位为 0，则会进入正常停止模式而非 VLLS3。
- 调用此函数之前，必须禁用 MCG 中的时钟监视器。
- 所有 I/O 都保留 MCU 处于 VLLS3 模式时的状态，并在复位之后通过代码对 ACKISO 位执行写入操作时释放。
- 调用此函数之前，必须在 LLWU 和唤醒模块中设置要用于将 MCU 从 VLLS3 模式唤醒的唤醒事件。有关设置 LLWU 以通过 PTE1 引脚唤醒的示例，请参见本应用笔记的 LLS 模式进入和退出 - 裸机代码示例 章节。

```
/*****************************************************************/
/* VLLS3 mode entry routine. Puts the processor into
 * VLLS3 mode from normal Run mode or VLPR.
 *
 * Mode transitions:
 * RUN to VLLS3
 * VLPR to VLLS3
 *
 * NOTE: VLLSx modes will always exit to Run mode even if you were
 *       in VLPR mode before entering VLLSx.
 *
 * Wake-up from VLLSx mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in VLLSx mode, so make
 * sure to setup the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * none
 */
void enter_vlls3(void)
{
   volatile unsigned int dummyread;
    /* Write to PMPROT to allow VLLS3 power modes */
    SMC_PMPROT = SMC_PMPROT_AVLLS_MASK;

    /* Set the VLLSM field to 0b100 for VLLSx(for MC1)
      or STOPM field to 0b100 for VLLSx (for MC2)
    - Retain state of LPWUI and RUNM */
     SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK ;
     SMC_PMCTRL |= SMC_PMCTRL_STOPM(0x4) ;
 //    MC_PMCTRL &= ~MC_PMCTRL_VLLSM_MASK ; //(for MC1)
```

```
 //     MC_PMCTRL |= MC_PMCTRL_VLLSM(0x4) ; //(for MC1)
    /* set VLLSM = 0b11 in SMC_VLLSCTRL (for MC2) */
     SMC_VLLSCTRL =  SMC_VLLSCTRL_VLLSM(3);
    /*wait for write to complete to SMC before stopping core */
    dummyread = SMC_CLLSCTRL;
    /* Now execute the stop instruction to go into VLLS3 */
    deepsleep();
}
/*****************************************************************/
```

9.1.13 进入 VLLS2 模式

MCU 可通过以下代码从运行或 VLPR 模式进入 VLLS2 模式。

MCU 从 VLLS 退出后始终会进入复位模式。

对于 Kinetis L，MCU 进入 VLLS1 模式；Kinetis L 不支持 VLLS2 模式。

前提条件:

- 要进入 VLLS2 模式，必须置位 PMPROT 寄存器中的 AVLLS 位。
- 如果 PMPROT 寄存器中的 AVLLS 和 AVLP 位为 0，则会进入正常停止模式而非 VLLS2。
- 调用此函数之前，必须禁用 MCG 中的时钟监视器。
- 所有 I/O 都保留 MCU 处于 VLLS2 模式时的状态，并在复位之后通过代码对 ACKISO 位执行写入操作时释放。
- 调用此函数之前，必须在 LLWU 和唤醒模块中设置要用于将 MCU 从 VLLS2 模式唤醒的唤醒事件。有关设置 LLWU 以通过 PTE1 引脚唤醒的示例，请参见本应用笔记的 LLS 模式进入和退出 - 裸机代码示例 章节。

```
/*****************************************************************/
/* VLLS2 mode entry routine. Puts the processor into
 * VLLS2 mode from normal Run mode or VLPR.
 *
 * Mode transitions:
 * RUN to VLLS2
 * VLPR to VLLS2
 *
 * NOTE: VLLSx modes will always exit to Run mode even
 *       if you were in VLPR mode before entering VLLSx.
 *
 * Wake-up from VLLSx mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in VLLSx mode, so make
 * sure to setup the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * none
 */
void enter_vlls2(void)
{
    volatile unsigned int dummyread;
     /* Write to PMPROT to allow VLLS2 power modes */
    SMC_PMPROT = SMC_PMPROT_AVLLS_MASK;

    /* Set the VLLSM field to 0b100 for VLLSx(for MC1)
      or STOPM field to 0b100 for VLLSx (for MC2)
    - Retain state of LPWUI and RUNM */
     SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK ;
     SMC_PMCTRL |= SMC_PMCTRL_STOPM(0x4) ;
 //    MC_PMCTRL &= ~MC_PMCTRL_VLLSM_MASK ;
 //    MC_PMCTRL |= MC_PMCTRL_VLLSM(0x4) ;
   /* set VLLSM = 0b10 in SMC_VLLSCTRL (for MC2) */
   SMC_VLLSCTRL =  SMC_VLLSCTRL_VLLSM(2);
   /*wait for write to complete to SMC before stopping core */
   dummyread = SMC_VLLSCTRL;
   /* Now execute the stop instruction to go into VLLS2 */
   deepsleep();
```

```
}
/*****************************************************************/
```

9.1.14 进入 VLLS1 模式

MCU 可通过以下代码从运行或 VLPR 模式进入 VLLS1 模式。

MCU 从 VLLS1 退出后始终会进入复位模式。

前提条件:

- 要进入 VLLS1 模式，必须置位 PMPROT 寄存器中的 AVLLS 位。
- 如果 PMPROT 寄存器中的 AVLLS 和 AVLP 位为 0，则会进入正常停止模式而非 VLLS1。
- 调用此函数之前，必须禁用 MCG 中的时钟监视器。
- 所有 I/O 都保留 MCU 处于 VLLS1 模式时的状态，并在复位之后通过代码对 ACKISO 位执行写入操作时释放。
- 调用此函数之前，必须在 LLWU 和唤醒模块中设置要用于将 MCU 从 VLLS1 模式唤醒的唤醒事件。有关设置 LLWU 以通过 PTE1 引脚唤醒的示例，请参见本应用笔记的 LLS 模式进入和退出 - 裸机代码示例 章节。

```
/*****************************************************************/
/* VLLS1 mode entry routine. Puts the processor into
 * VLLS1 mode from normal Run mode or VLPR.
 *
 * Mode transitions:
 * RUN to VLLS1
 * VLPR to VLLS1
 *
 * NOTE:VLLSx modes will always exit to Run mode even if you were
 * in VLPR mode before entering VLLSx.
 *
 * Wake-up from VLLSx mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in VLLSx mode, so make
 * sure to setup the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * none
 */
void enter_vlls1(void)
{
   volatile unsigned int dummyread;
   /* Write to PMPROT to allow all possible power modes */
   SMC_PMPROT = SMC_PMPROT_AVLLS_MASK;
   /* Set the VLLSM field to 0b100 for VLLSx(for MC1)
     or STOPM field to 0b100 for VLLSx (for MC2)
   - Retain state of LPWUI and RUNM */
   SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK ;
   SMC_PMCTRL |= SMC_PMCTRL_STOPM(0x4) ;
 //    MC_PMCTRL &= ~MC_PMCTRL_VLLSM_MASK ;
 //    MC_PMCTRL |= MC_PMCTRL_VLLSM(0x4) ;
   /* set VLLSM = 0b01 in SMC_VLLSCTRL (for MC2) */
   SMC_VLLSCTRL =  SMC_VLLSCTRL_VLLSM(1);
   /*wait for write to complete to SMC before stopping core */
   dummyread = SMC_VLLSCTRL;
   /* Now execute the stop instruction to go into VLLS1 */
   deepsleep();
}
/*****************************************************************/
```

9.1.15 进入 VLLS0 模式

MCU 可通过以下代码从运行或 VLPR 模式进入 VLLS0 模式。

MCU 从 VLLS0 退出后始终会进入复位模式。

前提条件:

- 要进入 VLLS0 模式，必须置位 PMPROT 寄存器中的 AVLLS 位。
- 如果 PMPROT 寄存器中的 AVLLS 和 AVLP 位为 0，则会进入正常停止模式而非 VLLS1。
- 调用此函数之前，必须禁用 MCG 中的时钟监视器。
- 所有 I/O 都保留 MCU 处于 VLLS0 模式时的状态，并在复位之后通过代码对 ACKISO 位执行写入操作时释放。
- 调用此函数之前，必须在 LLWU 和唤醒模块中设置要用于将 MCU 从 VLLS0 模式唤醒的唤醒事件。有关设置 LLWU 以通过 PTE1 引脚唤醒的示例，请参见本应用笔记的 LLS 模式进入和退出 - 裸机代码示例 章节。

```
/******************************************************************/
/* VLLS0 mode entry routine. Puts the processor into
 * VLLS0 mode from normal run mode or VLPR.
 *
 * Mode transitions:
 * RUN to VLLS0
 * VLPR to VLLS0
 *
 * NOTE: VLLSx modes will always exit to RUN mode even if you were
 * in VLPR mode before entering VLLSx.
 *
 * Wake-up from VLLSx mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in VLLSx mode, so make
 * sure to setup the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * PORPO_value - 0 POR detect circuit is enabled in VLLS0
 *               1 POR detect circuit is disabled in VLLS0
 */
 /******************************************************************/

void enter_vlls0(unsigned char PORPO_value )
{
    volatile unsigned int dummyread;
    /* Write to PMPROT to allow all possible power modes */
    SMC_PMPROT = SMC_PMPROT_AVLLS_MASK;
    /* Set the STOPM field to 0b100 for VLLS0 mode */
    SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK;
    SMC_PMCTRL |=  SMC_PMCTRL_STOPM(0x4);
    /* set VLLSM = 0b00 */
    SMC_VLLSCTRL =  (PORPO_value <<SMC_VLLSCTRL_PORPO_SHIFT)
                  | SMC_VLLSCTRL_VLLSM(3);
    /*wait for write to complete to SMC before stopping core */
    dummyread = SMC_VLLSCTRL;
    stop();
}
```

10 功耗模式退出转换

10.1 关于退出低功耗模式的更多注意事项

正常的退出流程是使能和检测可发起模式更改的事件。

对于 WAIT、STOP、VLPS、VLPR 和 VLPW，唤醒事件通常为来自引脚状态更改或模块的中断。LLWU 不使能且不用于退出这些模式。

对于 LLS、LLS2、LLS3、VLLS3、VLLS2、VLLS1 和 VLLS0，唤醒事件限制为已使能的 LLWU 唤醒引脚或模块，以及复位和 NMI 引脚。只要已在引脚控制寄存器中将该引脚“使能作为数字输入源”，则该引脚可作为唤醒源。对于无 LLWU 的 MCU，如果已包括在 MCU 参考手册中的唤醒源表，则模块或引脚唤醒为唤醒源且被设置为中断唤醒源。

10.1.1 引脚中断和 LLWU 唤醒功能集成

对于 LLWU 唤醒输入引脚，建议也使能该引脚上的引脚中断功能。若未使能引脚中断，则当 MCU 转换到低漏电模式时，会存在一个可能丢失边沿跳变的时间窗口。如果已使能引脚中断功能且在这段短暂的转换时间内出现边沿，则会停止进入低功耗模式，且 MCU 为引脚中断服务。

在 Kinetis K 器件中，所有 LLWU 唤醒引脚还可用作引脚中断源。在 Kinetis L 器件中，部分 LLWU 唤醒引脚没有此双重功能。

如果引脚还可用作唤醒 MCU 的引脚中断，则建议同时使用中断和 LLWU 唤醒功能。这是因为，若未为该引脚使能中断功能，则在 MCU 转换到低漏电功耗模式（如 LLSx 或 VLLSx）时，会出现一段非常短的时间窗口（~4 ns），在此期间可能会丢失边沿跳变。如果这段短暂的转换时间内出现边沿，则会停止进入低功耗模式，MCU 将使用引脚中断。

如果该引脚配置为引脚中断和 LLWU 唤醒引脚，则端口控制寄存器中相应的 ISF 标志位可能会置位，并且该标志位可在 LLWU 中清零。如果 ISF 标志已设置且未在 LLWU ISR 中清除，则 LLWU ISR 完成后将进入端口 ISR。

10.1.2 复位作为 LLWU 唤醒

在所有模式下，唤醒事件都可以为因任何复位源导致的复位。这些复位源如 Kinetis MCU 的参考手册中所列。

所有对 LPWUI 位的引用通常都适合于 Kinetis K MCU。Kinetis L 系列没有 LPWUI 位。

10.1.3 功耗模式转换时间

在基于 M4 内核的 Kinetis 器件上，中断延迟为 12 个周期。在基于 M0+内核的 Kinetis 器件上，中断延迟为 15 个周期。

表 4. 功耗模式转换时间

转换编号	转换	转换时间
1	WAIT - RUN	中断延迟
2	STOP - RUN	2 us + 中断延迟
3	VLPW - VLPR	中断延迟
4	VLPW - RUN	2 - 4 us + 中断延迟 [1]
5	VLPS - VLPR	2 - 4 us + 中断延迟
6	VLPS - RUN	2 - 4 us + 中断延迟 [2]
7	VLLSx - RUN	BOOT(LP) + 53 us 至 115 us[3]
8	LLS - RUN	2 us + 中断延迟

1. VLPW 到 RUN 转换仅可发生在具有 LPWUI 位的 Kinetis 器件上。
2. VLPS 到 RUN 转换仅可发生在具有 LPWUI 位的 Kinetis 器件或 Kinetis K22F MCU 上。
3. 通过唤醒复位流程退出 VLLSx 模式

10.1.4 可减慢从低功耗模式退出的因素

- MCG 时钟模式未使用 FLL。若使用 MCG 时钟模式 PEE，会导致仅在启动外部时钟源之后才可启动唤醒。此时钟源通常比 PEE 时钟频率慢得多。
- 在输入触发器上使用引脚滤波。
- 输入触发器的上升时间缓慢（在 VIH 处检测到上升沿，在 VIL 处检测到下降沿）。
- MCU 时钟缓慢或停止。
- 唤醒源为 NMI 引脚并在一段时间内保持低电平。NMI 对电平敏感并将保持处于 NMI ISR，直到 NMI 输入返回高电平。
- 调试器在模式转换过程中处于活动状态。调试模块交互可能会使处理器中止或停顿。
- 外部时钟源为晶体，并且在低功耗模式下停止运行。晶体启动时间会增加到模式恢复时间中。

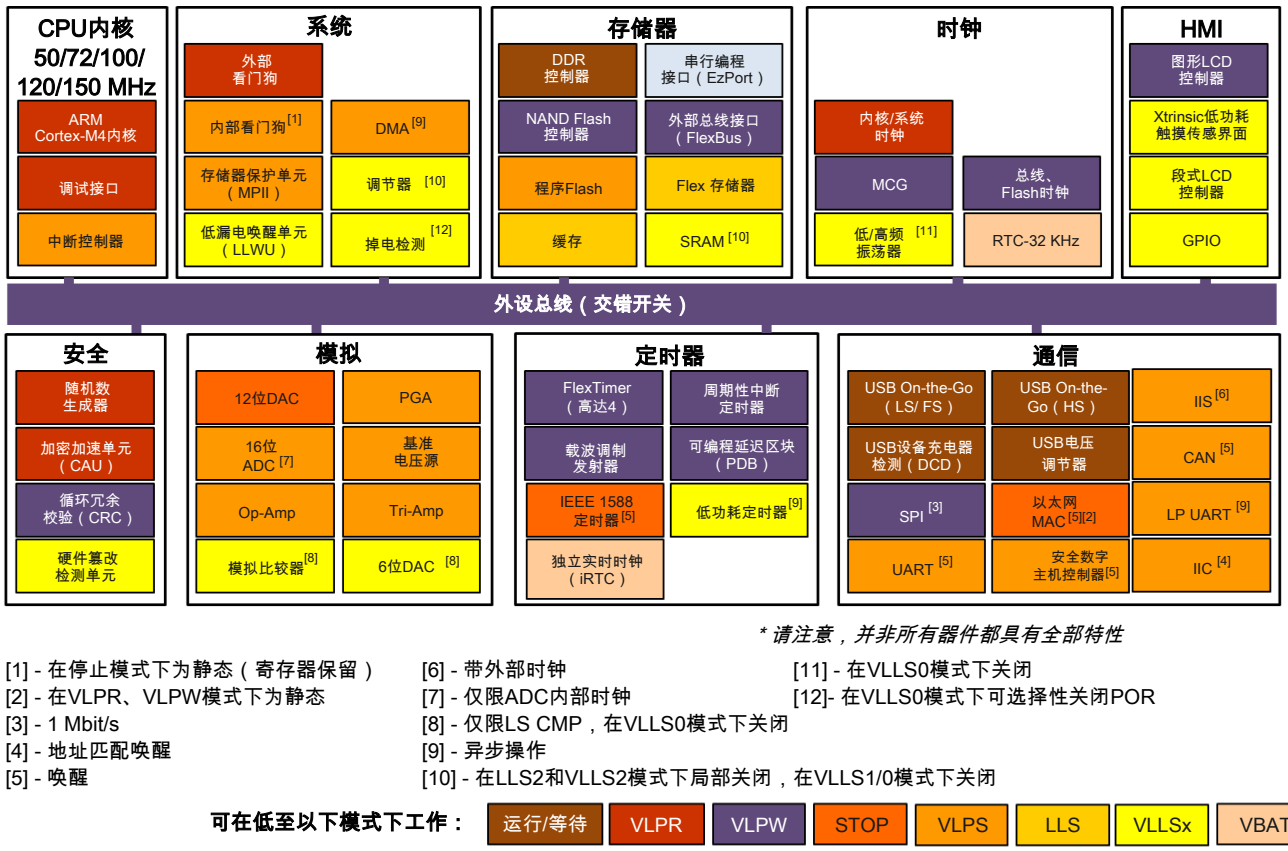
10.1.5 导致 MCU 无法识别中断或 LLWU 输入的因素

- MCU 处于 RUN、WAIT、VLPR 或 VLPW 模式且输入脉冲小于 1.5 倍总线时钟长度（例如，总线时钟为 1 MHz 时，250 ns 输入脉冲无法工作）。
- 在输入触发器上使用引脚滤波并且输入长度不够。如果 LPO 用于为滤波器提供时钟，则脉冲小于 3 毫秒
- 输入触发器的上升时间缓慢（在 VIH 处检测到上升沿，在 VIL 处检测到下降沿）。
- MCU 时钟缓慢或停止。
- 处于 WAIT、VLPW、STOP、VLPS 模式且全局中断已禁用，同时 RESET 和 NMI 未用于唤醒 MCU。
- 如果引脚不是数字输入，设置为禁止或输出，则 LLWU 输入功能不会唤醒 MCU。

11 不同功耗模式下的模块

11.1 低功耗模式下的模块操作

以下流程图所示为 Kinetis K 系列模块在各种低功耗模式下的工作情况。

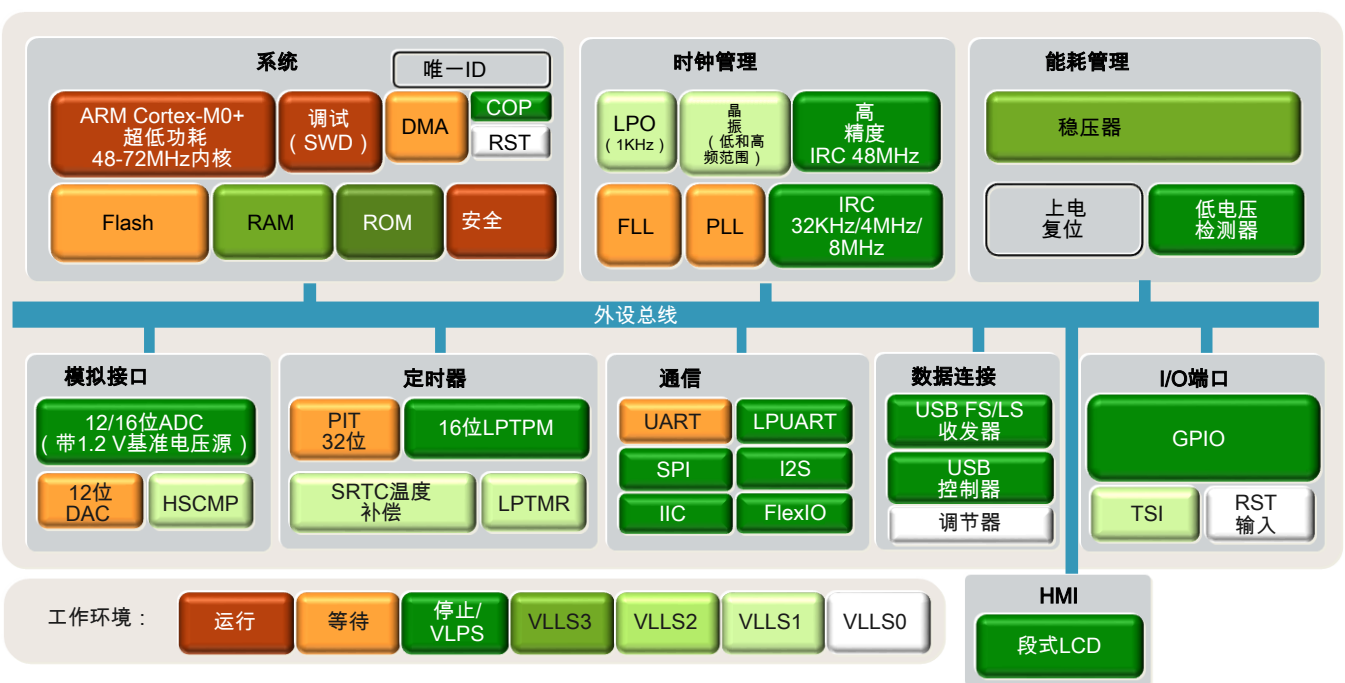


** 请注意，并非所有器件都具有全部特性*

[1] - 在停止模式下为静态（寄存器保留）
[2] - 在VLPR、VLPW模式下为静态
[3] - 1 Mbit/s
[4] - 地址匹配唤醒
[5] - 唤醒
[6] - 带外部时钟
[7] - 仅限ADC内部时钟
[8] - 仅限LS CMP，在VLLS0模式下关闭
[9] - 异步操作
[10] - 在LLS2和VLLS2模式下局部关闭，在VLLS1/0模式下关闭
[11] - 在VLLS0模式下关闭
[12]- 在VLLS0模式下可选择性关闭POR

可在低至以下模式下工作： 运行/等待 | VLPR | VLPW | STOP | VLPS | LLS | VLLSx | VBAT

图 12. Kinetis K 模块功耗模式图



- FF — 完整功能。在 VLPR 和 VLPW 下，系统频率受限，但如果模块的功能未受限，则仍列为 FF。
- 异步操作 = Kinetis L 以及后续的 Kinetis K
- 静态 — 保留模块寄存器状态和相关存储器。
- 通电 — 储存器通电以保留内容。

- 低功耗 — Flash 具有低功耗状态，能够保留配置寄存器以支持更加快速的唤醒。
- OFF = 模块断电；唤醒后模块处于复位状态。
- 唤醒 — 模块可用作芯片的唤醒源。
- CPO - 仅计算模式 - Kinetis L 以及后续的 Kinetis K
- PSTOPx - 局部停止模式 1 或 2 - Kinetis L 以及后续的 Kinetis K
- IOPORT - 单周期 IOPORT - Kinetis L 以及后续的 Kinetis K

表 5. 低功耗模式下的模块操作

模块	停止	VLPR	VLPW	VLPS	LLSx	VLLSx
内核平台模块						
内核	静态	FF	静态	静态	静态	关断
NVIC	静态	FF	FF	静态	静态	关断
浮点处理单元 (FPU)	静态	FF	静态	静态	静态	关断
HDQR	静态	FF	静态	静态	静态	关断
系统模块						
模式控制器	FF	FF	FF	FF	FF	FF
LLWU[1]	静态	静态	静态	静态	FF	FF[2]
调节器	开	低功耗	低功耗	低功耗	低功耗	VLLS2/3 时为低功耗，VLLS0/1 时为关断
LVD	开	禁用	禁用	禁用	禁用	禁用
掉电检测	开	开	开	开	开	VLLS1/2/3 时为开，VLLS0 时为可选禁用 [3]
DMA	异步操作	FF CPO 中为异步操作	FF	异步操作	静态	关断
看门狗	静态 PSTOP2 中为 FF	FF CPO 中为静态	FF	FF	静态	关断
EWM	静态 PSTOP2 中为 FF	FF CPO 中为静态	静态	静态	静态	关断
时钟						
1 kHz LPO	开	开	开	开	开	VLLS1/2/3 时为开，VLLS0 时为关断
系统振荡器 (OSC)	OSCERCLK 可选	OSCERCLK（最大为 16 MHz 晶体）	OSCERCLK（最大为 16 MHz 晶体）	OSCERCLK（最大为 16 MHz 晶体）	限定为低范围/低功耗	VLLS1/2/3 时限定为低范围/低功耗，VLLS0 时为关断
MCG PLL	PLL 可选择性开启但不选通	静态	静态	静态	静态	关断
MCG FLL	静态 — MCGIRCLK 可选	静态	静态	静态	静态 — 无时钟输出	关断

下一页继续介绍此表...

表 5. 低功耗模式下的模块操作 (继续)

模块	停止	VLPR	VLPW	VLPS	LLSx	VLLSx
MCG 慢速 IRC	静态 - MCGIRCLK 可选	静态	静态	静态 - MCGIRCLK 可选	静态 — 无时钟输出	关断
MCG 快速 IRC	MCGIRCLK 可选	2 或 4 MHz IRC	2 或 4 MHz IRC	MCGIRCLK 可选	静态 — 无时钟输出	关断
内核时钟	关断 在调试中除外	2 或 4 MHz IRC	关断	关断	关断	关断
系统时钟	关断 在调试中除外	2 或 4 MHz（最大值） CPO 为关断	2 或 4 MHz IRC	关断	关断	关断
总线时钟	关断 在调试中除外	1 MHz L 系列，2 或 4 MHz IRC CPO 为关断	2 或 4 MHz IRC	关断	关断	关断
存储器和存储器接口						
Flash	通电	1 MHz（最大值）访问 - 无编程	低功耗	低功耗	关断	关断
部分 SRAM_U[4]	低功耗	低功耗	低功耗	低功耗	低功耗	VLLS3,2 时为低功耗
剩余的 SRAM_U 和所有 SRAM_L	低功耗	低功耗	低功耗	低功耗	低功耗	VLLS3 中为低功耗
高速缓存	低功耗	低功耗	低功耗	低功耗	低功耗	关断
FlexMemory[5]	低功耗	低功耗 [6]	低功耗	低功耗	低功耗	在 VLLS3 下为低功耗，在 VLLS2 和 VLLS1 下关闭
VBAT 寄存器文件 [7]	通电	通电	通电	通电	通电	通电
系统寄存器文件	通电	通电	通电	通电	通电	通电
DDR 控制器	低功耗	低功耗	低功耗	低功耗	低功耗	关断
SDRAM 控制器	低功耗	FF 在 CPO 中禁用	FF	低功耗	低功耗	关断
NFC	静态	FF	FF	静态	静态	关断
FlexBus	静态	FF 在 CPO 中禁用	FF	静态	静态	关断
EzPort	禁用	禁用	禁用	禁用	禁用	禁用
通信接口						
高速 USB 物理层	静态	静态	静态	静态	关断	关断
高速 USB 控制器	静态，在恢复时唤醒	静态，在恢复时唤醒	静态，在恢复时唤醒	静态	静态	关断
全速/低速 USB	静态，在恢复时唤醒	静态，在恢复时唤醒	静态，在恢复时唤醒	静态	静态	关断
USB DCD	静态	FF	FF	静态	静态	关断

下一页继续介绍此表...

表 5. 低功耗模式下的模块操作 (继续)

模块	停止	VLPR	VLPW	VLPS	LLSx	VLLSx
USB 稳压器	可选	可选	可选	可选	可选	可选
以太网	唤醒	静态	静态	静态	静态	关断
UART0, UART1	静态，在边沿唤醒	高达 250kbits/s 静态，在 CPO 边沿唤醒	高达 250kbits/s	静态，在边沿唤醒	静态	关断
LPUART Kinetis L 以及后续的 Kinetis K	异步操作 PSTOP2 中为 FF	1-4 Mbps CPO 中为异步操作	1-4 Mbps	异步操作	静态	关断
FLEXIO Kinetis K2	异步操作 PSTOP2 中为 FF	1-4 Mbps CPO 中为异步操作	1-4 Mbps	异步操作	静态	关断
UART（其他）	静态，在边沿唤醒 PSTOP2 中为 FF	125-250 kbit/s 静态，在 CPO 边沿唤醒	125-250 kbit/s	静态，在边沿唤醒	静态	关断
串行外设接口 (SPI) Kinetis L K2	静态 PSTOP2 中为 FF	静态，从机模式接收 CPO 中为静态	主机模式，500 kbps， 从机模式，250 kbps 静态，在 CPO 中以从机模式接收	静态，从机模式接收	静态	关断
SPI	静态 PSTOP2 中为 FF	1 Mbit/s（从机） 2 Mbit/s（主机） CPO 中为静态	1 Mbit/s（从机） 2 Mbit/s（主机）	静态	静态	关断
I^2C	静态，地址匹配唤醒	100 kbit/s	100 kbit/s	静态，地址匹配唤醒	静态	关断
CAN	唤醒 PSTOP2 中为 FF	250-500 kbit/s	250-500 kbit/s	唤醒	静态	关断
I^2C Kinetis L	50 kbps	静态，地址匹配唤醒 PSTOP2 中为 FF	50 kbps 静态，CPO 中为地址匹配唤醒	静态，地址匹配唤醒	静态	关断
I^2C Kinetis K2	静态，地址匹配唤醒 PSTOP2 中为 FF	200 kbit/s，静态，CPO 中为地址匹配唤醒	200 kbps	静态，地址匹配唤醒	静态	关断
I2S	采用外部时钟的异步操作 [8]	FF CPO 中为异步操作	FF	带外部时钟的 FF[9]	静态	关断
SDHC	唤醒	FF	FF	唤醒	静态	关断
安全						
CRC	静态	FF	FF	静态	静态	关断

下一页继续介绍此表...

表 5. 低功耗模式下的模块操作 (继续)

模块	停止	VLPR	VLPW	VLPS	LLSx	VLLSx
RNG	静态	FF	FF	静态	静态	关断
DryIce [7]	FF	FF	FF	FF	FF	FF
MMCAU	静态	FF	FF	静态	静态	关断
定时器						
TPM Kinetis L K2	异步操作 PSTOP2 中为 FF	FF CPO 中为异步操作	FF	异步操作	静态	关断
FTM	静态 PSTOP2 中为 FF	FF	FF	静态	静态	关断
PIT	静态 PSTOP2 中为 FF	FF CPO 中为静态	FF	静态	静态	关断
PDB	静态 PSTOP2 中为 FF	FF CPO 中为静态	FF	静态	静态	关断
LPTMR Kinetis L K2	异步操作 PSTOP2 中为 FF	FF	FF	异步操作	异步操作	异步操作 [10]
LPTMR	FF	FF	FF	FF	FF[11]	异步操作 [10]
RTC Kinetis L K2	异步操作 PSTOP2 中为 FF	FF CPO 中为异步操作	FF	异步操作	异步操作	FF[12]
RTC - 32kHz OSC	FF	FF	FF	FF	FF[13]	FF
CMT	静态	FF	FF	静态	静态	关断
模拟						
16 位 ADC	仅限 ADC 内部时钟	FF	FF	仅限 ADC 内部时钟	静态	关断
CMP - Kinetis L[14]	高速或低速比较 PSTOP2 中为 FF	FF CPO 中为高速或低速比较	FF	高速或低速比较	低速比较	VLLS1/3 时为低速比较，VLLS0 时为关断
CMP[14]	高速或低速比较	FF	FF	高速或低速比较	低速比较	VLLS1/2/3 时为低速比较，VLLS0 时为关断
6 位 DAC	静态 PSTOP2 中为 FF	FF CPO 中为静态	FF	静态	静态	静态，VLLS0 时为关断
VREF	FF	FF	FF	FF	静态	关断
OPAMP	FF	FF	FF	FF	静态	关断
TRIAMP	FF	FF	FF	FF	静态	关断
PGA	FF	FF	FF	FF	静态	关断

下一页继续介绍此表...

表 5. 低功耗模式下的模块操作 (继续)

模块	停止	VLPR	VLPW	VLPS	LLSx	VLLSx
12 位 DAC	静态 PSTOP2 中为 FF	FF CPO 中为静态	FF	静态	静态	静态
人机接口						
GPIO	静态输出，唤醒输入 PSTOP2 中为 FF	FF 仅在 CPO 中写入 IOPORT	FF	静态输出，唤醒输入	静态输出，引脚锁定，唤醒输入	静态输出，引脚锁定，唤醒输入
段式 LCD	FF	FF	FF	FF	FF[15]	FF[16]
图形 LCD	静态	FF	FF	静态	静态	关断
TSI Kinetis L	异步操作 PSTOP2 中为 FF	FF CPO 中为异步操作	异步操作	异步操作	异步操作	异步操作
TSI	唤醒	FF	FF	唤醒	唤醒 [16]	唤醒 [11]

1. 使用 LLWU 模块，则用于此芯片的外部引脚不需要使能相关的外设功能。它只需将控制该引脚（GPIO 或外设）的功能配置为数字输入，以便允许将转换传递至 LLWU。
2. 由于 LPO 时钟源禁用，VLLS0 期间将旁路滤波器。
3. [MC2]SMC 模块中的 VLLSCTRL[PORPO]位控制该选项。
4. SRAM_U 以及 32KB SRAM L（具有 PORPO 的器件）模块的 4、8、16、32 KB 部分在低功耗模式 VLLS2 中保持通电状态。Kinetis L 不提供 VLLS2 模式。
5. 在 VLLS3 下，FlexRAM 始终处于通电状态。将 FlexRAM 配置为传统 RAM 时，可选择性地在 VLLS2 模式下通电。将 FlexRAM 配置为 EEPROM 时，则在 VLLS2 模式下断电。
6. 使能为 EEPROM 的 FlexRAM 在 VLPR 下不可写入，且会忽略写操作。允许对 VLPR 下使能为 EEPROM 的 FlexRAM 进行读访问。FlexRAM 配置为传统 RAM 后，对其无访问限制。
7. 这些元件在 BAT 功耗模式下仍然保持通电状态。
8. 使用外部生成的位时钟或外部生成的音频主机时钟（包括 EXTAL）。

   PSTOP2 中为 FF
9. 使用外部生成的位时钟或外部生成的音频主机时钟（包括 EXTAL）。
10. VLLS0 中不提供 LPO 时钟源。此外，要在 VLLS0 中使用系统 OSC，则必须对其进行配置以实现旁路（外部时钟）操作。所有模式均提供脉冲计数。
11. VLLS0 中不提供系统 OSC 和 LPO 时钟源。
12. 在 VLLS0 中，唯一的时钟选项来自 RTC_CLKIN。
13. RTC_CLKOUT 不可用。
14. 停止模式或 VLPS 时的 CMP 支持高速或低速外部引脚间或外部引脚与 DAC 之间的比较。LLS 或 VLLSx 时的 CMP 仅支持低速外部引脚间或外部引脚与 DAC 之间的比较。停止、VLPS、LLS、或 VLLSx 模式下不提供窗口、采样和滤波工作模式。
15. LLS 和 VLLSx 不支持帧结束唤醒。
16. 从 LLS 和 VLLSx 模式进行 TSI 唤醒仅限于单个可选引脚。

12 使用外部存储器和外设 - DDR 存储器控制器和 DDR 低功耗模式用例

12.1 控制 DDR 存储器和接口代码示例

Kinetis K70/K61 集成了一个 DRAM（DDR）存储器控制器，该控制器具有额外的低功耗模式，称为 DDR 低功耗模式。DDR 控制有助于实现低功耗操作，即便是在执行 DDR 存储器中的代码时也是如此。使用 DDR2 或 LPDDR1 存储器工作时，系统实现最低功耗的关键在于管理 MCU 运行模式之间、存储器以及 MCU 低功耗模式之间的转换。

本应用笔记章节介绍了一些使用技巧，可使集成 DDR 存储器控制器来驱动 DDR2 或 LPDDR 存储器的 Kinetis K70/K61 实现低功耗操作。以下章节介绍了 DDR 存储器控制以及在 DDR 和 MCU 低功耗模式以及运行模式之间转换所使用的进入和退出步骤。

请参考这些说明以及应用笔记演示代码，了解这些控制并评估 MCU 和存储器操作的不同情形。

在 VLPR 模式下使用 DDR，情况将会怎样。请注意，DDR 采用 PLL 运行。因此，在进入会关闭 PLL（例如 VLPR、VLPS、STOP、LLS、VLLSx）的任何 MCU 模式之前，必须先将 DDR 置于低功耗模式。

对于 SDRAM、串行 Flash 和传感器之类的其他外部存储器和外设，可使用相同的低功耗模式进入方法。在大多数此类器件上，在使 MCU 进入其低功耗模式之前，都需要先设置低功耗模式。

注释：SDK 低功耗模式进入函数具有前后回调函数，是此类代码的理想之选。对前面的 LPDDR 存储器或外设进行提前设置让 MCU 可以进入低功耗模式，而进行后续设置则允许执行恢复操作。更多详情，请参见《SDK API 参考手册》。

12.1.1 DDR 功耗模式控制和使用

外部 DDR 存储器和控制器具有 5 种低功耗模式，其中 LP 模式 1 和 2 不会保留存储器内容。

表 6. 通过寄存器 DDR_CR16_LPCTRL 实现 DDR 存储器低功耗模式控制

模式	存储器类型	值	DDR 低功耗模式
1	DDR2 和 LPDDR	0x10	DDR 存储器掉电
2	DDR2 和 LPDDR	0x08	DDR 存储器掉电（存储器时钟关闭）
3	DDR2 和 LPDDR	0x04	DDR 存储器自动刷新
4	DDR2 和 LPDDR	0x02	DDR 存储器自动刷新（存储器时钟关闭）
5	LPDDR	0x01	DDR 存储器自动刷新（存储器和控制器时钟关闭）。无法手动进入 LP 模式 5

12.1.2 进入 MCU 低功耗模式前

一般情况下，软件流将调用函数以将 MCU 置于其中一种低功耗模式中，如 STOP、VLPS、LLS 或 VLLS3。如果使用允许低功耗操作的外部存储器，则在进入所请求的低功耗模式之前，该函数会设置存储器和存储器控制器状态。将外部 DDR 存储器与 MCU 低功耗模式结合使用时，则在进入低功耗模式之前，会将 DDR 存储器置于低功耗模式，禁用 DDR 存储器控制器且将 MCU DDR 引脚设为更低的功耗状态。

以下 LLS 低功耗模式进入代码包括 DDR 控制器以及 SIM_MCR 寄存器写入操作，以将 DDR 存储器置于模式 4 – DDR 自动刷新模式 (时钟关闭)。这是 DDR2 类型存储器的最低功耗模式。

```
    /* send command to memory to enter ddr low power mode 4 */
    DDR_CR16 = DDR_CR16_LPCTRL(0x02);      //bit 17 set of DDR_CR16
    SIM_MCR =   SIM_MCR_DDRDQSDIS_MASK | SIM_MCR_DDRCFG(1)
              | SIM_MCR_DDRS_MASK ;
    for (i = 0;i < 100;i++){
         if ( SIM_MCR & SIM_MCR_DDRS_MASK == 0x02)
         break;
         }
    SIM_SCGC3 &= ~SIM_SCGC3_DDR_MASK;
   // turned off the clock gate for the DDR Controller
```

12.1.3 退出 MCU 低功耗模式后

MCU 目前处于 LLS 模式。

如果 PORTE1 上发生下降沿事件，MCU 将退出 LLS 模式并返回运行模式。

从 LLS 模式退出后，代码会清除 LLWU 标志寄存器中的唤醒事件标志

如果中断和 LLWU 中断已使能，则会恢复执行 LLWU 中断服务程序。然后，代码执行将返回至 WFI 或 STOP 指令后的指令。

如果中断已禁用，则会恢复执行 WFI 或 STOP 指令后的指令。

此时，DDR 存储器会禁用。在 PLL 重启以及 DDR 存储器退出自动刷新模式之前，以下任何代码均不能访问 DDR。请注意，在 PLL 重启且此代码已完成之前，切勿启用任何可能会尝试访问 DDR 的中断。

```
  /* after exiting LLS (can be in LLWU interrupt service Routine)
  * clear the wake-up flag in the LLWU-write one to clear the flag
  */
  if (LLWU_F1 & LLWU_F1_WUF0_MASK) {
      LLWU_F1 |= LLWU_F1_WUF0_MASK;

 /* Enable DDR controller clock gate */
      SIM_SCGC3 |= SIM_SCGC3_DDR_MASK; /* Enable DDR controller clock gate */
     DDR_RCR = DDR_RCR_RST_MASK;      // reset DDR PHY

     DDR_RCR = 0;
     /* soft reset to DDR RCR, DDR_DQS analog circuit enabled,
        lpddr half strength, DDRPEN = 1 */
     SIM_MCR &= ~SIM_MCR_RCRRSTEN_MASK;
     SIM_MCR &= ~SIM_MCR_DDRSREN_MASK;
     SIM_MCR &= ~SIM_MCR_DDRDQSDIS_MASK;
     DDR_CR50 |= DDR_CR50_CLKSTATUS_MASK;
     /* disable the DDR memory low power mode  */
     DDR_CR16 &= ~DDR_CR16_LPCTRL_MASK;  // all bits are zero
     DDR_CR16 |= DDR_CR16_QKREF_MASK;    // since set by lpddr_init code
     DDR_CR15 &= ~DDR_CR15_SREF_MASK;    // Disable self-refresh mode
     SIM_MCR |=  SIM_MCR_DDRPEN_MASK;
    /*wait for write to complete to before continuing */
    dummyread = SIM_MCR;
 }
```

13 功率测量

13.1 功率测量结果和技巧

13.1.1 实时功率测量

测量应用中电流消耗的典型方法是通过数字万用表。要测量 Kinetis MCU 在这些最低功耗模式下的电流，需要在宽范围内达到+/– 10 nA 的精度。

如果测量与 MCU 的 VDD 串联的电阻或 10 匝电流环路上的压降，则可以获得更佳的实时电流消耗数据。

图 13 和图 14 所示为使用有源差分探头和示波器在 10 欧姆电阻上测得的实时电流数据。

测量所使用的软件从一组演示代码移植而来，它们会对单个输入反复执行 ADC 测量，并将结果存储在一个表中。每获得 256 个样本，便会使用 Kinetis CMSIS DSP 库执行一次 FFT 计算。ADC 测量和 FFT 数字计算均在运行或 VLPR 模式下完成。当不处于运行模式时，MCU 会被置于 VLPS 模式。

图 13 所示为两款器件（运行模式下的 K40 100 MHz MCU 以及 VLPR 模式下的 K20 50 MHz MCU）的实时测量结果。

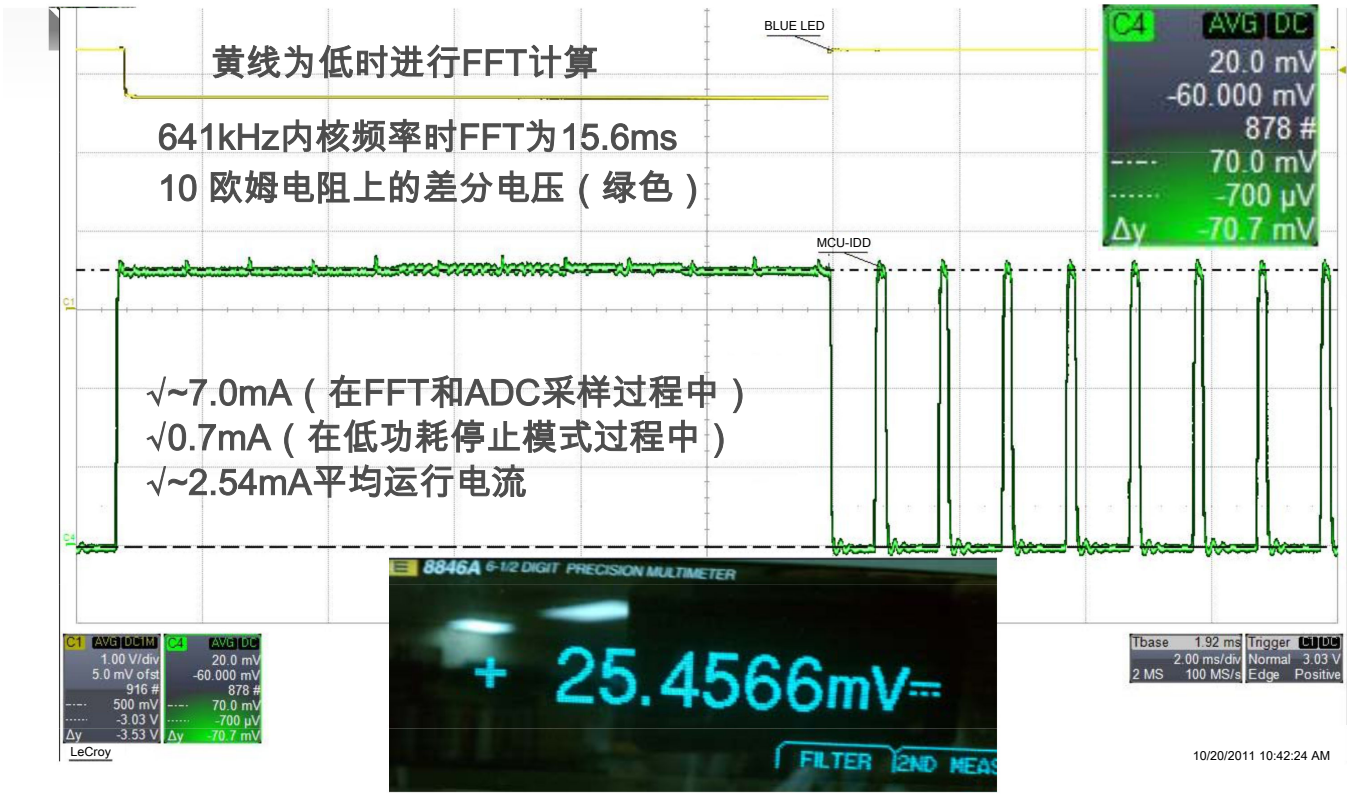


图 13. 运行和 VLPS 模式下的实时测量

上述测量数据均使用差分探头在 10 欧姆精密电阻上测得。将电压测量值除以 10 即可得到电流值。

运行模式下的电流约为 7 mA，而 VLPS 模式下的电流约为 700 µA。采用精密数字万用表测得的平均电流为 2.54 mA。

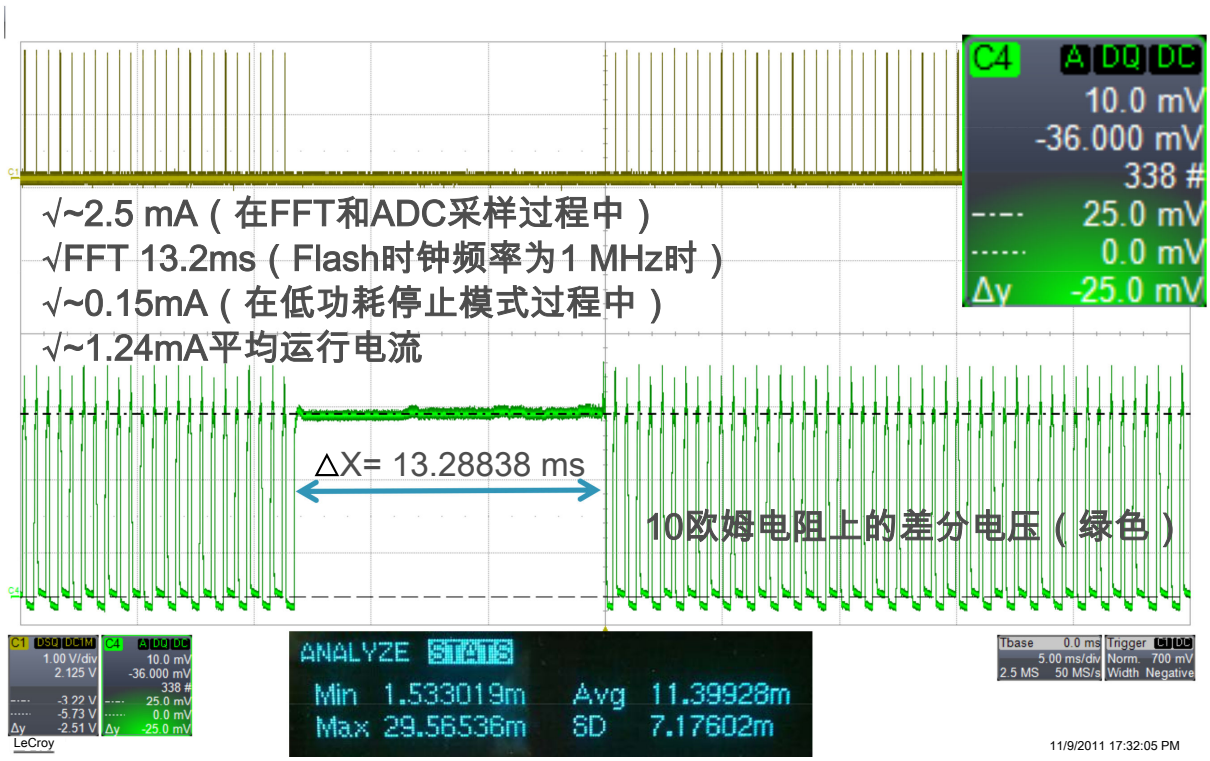


图 14. VLPR 和 VLPS 模式下的实时测量

上述测量数据均使用差分探头在 10 欧姆精密电阻上测得。将电压测量值除以 10 即可得到电流值。

使用带统计计算功能的精密数字万用表测量同一 10 欧姆电阻上的电压时，图形底部会叠加显示电流的最小值、最大值、平均值以及标准差。

极低功耗运行模式下的电流约为 2.5 mA，VLPS 模式下的电流约为 150 µA。采用精密数字万用表（DMM）测得的平均电流为 1.2 mA。

13.1.2 在测试台上进行低功耗测量的技巧

- 外部。 以下建议解答了您在重现数据手册中电流规格时遇到的常见问题。
  a. 使用数字万用表（**DMM**）时，请使用**"**手动量程模式**"**。 使用已启用自动量程功能的 DMM 时，可能会导致 LVD 和 POR 复位。这是您在从其中一种低功耗模式（如 LLS 或 VLPS 模式）返回至运行模式时遇到的最常见问题。在 MCU 处于低功耗模式期间，DMM 已将量程更改为微安或纳安，而突然涌入的电流要求 DMM 更改量程。量程更改不能很快实现，于是，MCU 会供电不足而将 VDD 电平拉至低于 LVD 或 POR 限值。
  b. 断开调试器并重启 **MCU**。 连接用于 Kinetis 的 JTAG 调试器之后，MCU 可能会使其内的调试器模块处于活动状态，为该模块提供时钟，因而会消耗功率。连接后，外部调试器硬件还可能会加载 JTAG 端口的 I/O。因此，低功耗测量值可能会高于预期值。
  c. 隔离 **MCU VDD**。 如果想要测量 MCU 消耗的电流，请移除由为 MCU 供电的电源供电的其他 IC 和元器件网络。例如，早期为 Kinetis 提供的塔式电路板在 MCU_VDD 和地线之间接有一个电位计。跨 3.6 V 电源连接一个 5 K 电位计可消耗 720 µA。与 MCU 在 VLLS1 模式下消耗的约 1.5 µA 电流相比，这个数字相当大。
  d. 自动测量 **VBAT** 或 **RTC VDD**。RTC 运行并计时时，RTC VBAT 域的供电电流通常远远小于 1 µA。这个非常小的 IDD 可被运行或 VLPR 模式下的电流覆盖。
  e. 匹配输入阻抗。 如果高速信号的阻抗(快速边沿跳变)不完全匹配，则信号可能"振铃"并超过器件的 VDD 电压。这可能会导致信号通过输入保护二极管为器件提供电流。如果是高速输入时钟，情况更是如此。处于最低功耗模式下时，此问题可能会导致 IDD 测量值为负数。

f. 匹配电压水平。 尽管某些器件上的 MCU 输入引脚兼容 5 V 电压，但在 MCU 进入低功耗模式后，MCU_VDD 上的电流测量值将会受任何高于 MCU_VDD 的输入影响。较高的输入引脚将通过此输入引脚为 MCU 反向供电，从而导致低功耗模式下 IDD 读数为负数。
g. 降低 **MCU** 的引脚负载。 当 MCU 通过输出引脚提供电流时，电源将源自 MCU_VDD。通过输出引脚输出高频信号时，这最为明显，如同您处理 FlexBus 时钟和地址/数据引脚一样。
h. 使用调试挂钩生成代码，它们无需会消耗大量电流的调试器或高速串行接口。

- **MCU** 内部。 以下是可能妨碍您获得最低数据手册电流规格的最常见问题列表。
  a. 看门狗未禁用，导致复位。禁用或服务看门狗。
  b. 时钟监视器未禁用，可能会导致复位。禁用所有时钟监视器。
  c. 已在低功耗模式下使能晶振。RTC 振荡器通常耗电低于 500 nA。
  d. CLKOUT 信号输出至引脚。不断改变状态的引脚将会产生功耗。
  e. PMPROT 寄存器中的对应位不允许所请求的低功耗模式。例如，如果未在 PMPROT 中设置 ALLS，而执行了 WFI 指令，则将进入停止模式，而不是 LLS 模式。
  f. 在读取或写入之前，未使能模块的时钟选通功能。这会导致在 MCU 尝试进入低功耗模式之前复位。
  g. 在进入低功耗模式之前，必须应答模式控制器模式进入请求的模块已被关闭时钟选通功能。这会导致停止模式应答复位。
  h. 如果在调试完成后未取消对停止或睡眠函数调用的注释，您将一直处于较高的功耗模式。
  i. 如果 MCU 软件正在轮询 RTC 寄存器或 VBAT 寄存器文件模块，则由于来自代码的不间断访问，VBAT 域电流将增大正常值的 2–5 倍。因此，要保持最低电流，不得轮询 RTC 寄存器。
  j. 唤醒事件的频率太高，这意味着 MCU 处于运行或 VLPR 模式的时间要长于处于低功耗模式的时间。如上所示，从低功耗模式到运行模式的转换时间非常短。如果 MCU 处于运行模式和低功耗模式下的时间分别仅为 9 ms 和 1 ms，则与每秒钟 MCU 仅运行 1 ms 相比，系统的平均电流将高出很多。
  k. MCU 的运行频率远高于完成工作所需的频率。使用 SIM CLK 分频器抑制时钟或降低时钟。显然，MCU 频率越高，MCU 的 IDD 就越高。减少时钟并降低运行或等待模式下的电流。然而，也需要进行权衡考虑。如果可以容忍运行模式下的电流，则与在运行模式下采用较慢时钟相比，尽快完成工作并立即返回低功耗模式将更为有利。
  l. 在无内部或外部拉动器件的情况下，输入引脚浮动。这可能导致每个引脚消耗 50-80 uA 的电流。这包括 JTAG 或 SWD 引脚。禁用端口 A 上的 JTAG 引脚，或正确端接输入。

14 LLWU 引脚和模块唤醒

14.1 LLWU 硬件考虑因素

14.1.1 引脚唤醒事件

MCU 唤醒逻辑设计用于使用中断将 MCU 从低功耗模式唤醒。建议屏蔽所用中断并使能 LLWU 中断。如果已屏蔽全局中断或未使能 LLWU 中断，则应用可能会出现意外操作。

LLWU_Px 为外部引脚输入。该引脚上多路复用的任何数字输入功能均可选为唤醒源。

您可以选择引脚上升沿、下降沿或者任何边沿引起唤醒事件。LLWU 控制着 LLWU 唤醒源表中所述每个引脚的操作。对于所有 Kinetis 系列器件，这组挑选出来的引脚基本保持不变。Kinetis L 器件可能仅具有所定义引脚的子集。

只要引脚复用主动选择数字输入引脚功能（例如 GPIO 输入、UART RX、 RTS 或从机模式下的 SPI 从机选择输入），下表中列出的引脚就可以用作唤醒源。如果引脚复用不是 LLWU 唤醒引脚，而是数字输出或选择模拟功能，则唤醒事件不会唤醒 MCU。如果引脚用作数字输入功能，则存在一个从管脚经过异步唤醒中断控制器（AWIC）到 LLWU 的异步路径。该异步路径允许将唤醒事件传递至 LLWU，而无需使用时钟和时钟同步器。有关数字输入信号选项和 LLWU 唤醒引脚列表，请参见芯片的信号多路复用表。

14.1.2 集成引脚中断和 LLWU 唤醒功能

对于 LLWU 唤醒输入引脚，建议使能 LLWU 唤醒引脚上的引脚中断功能。若未使能引脚中断，则当 MCU 转换到低漏电模式时，会存在一个可能丢失边沿跳变的时间窗口。如果已使能引脚中断功能且在这段短暂的转换时间内出现边沿，则会停止进入低功耗模式，且 MCU 为引脚中断服务。

在 Kinetis K 器件中，所有 LLWU 唤醒引脚还可用作引脚中断源。在 Kinetis L 器件中，部分 LLWU 唤醒引脚没有此双重功能。

14.1.3 模块唤醒事件

LLWU_M0IF–M7IF 是 LLWU 内部外设中断标志。您最多可以在 LLWU 中选择 8 个模块唤醒源。要完成唤醒源的初始化操作，必须使能该模块并进行设置以创建中断，并且必须使能 LLWU 模块标志以作为 LLWU 唤醒源。来自已使能模块的中断将会创建一个唤醒事件。

使用外设唤醒后，NVIC 挂起中断寄存器中的 LLWU 中断标志和唤醒模块的标志会被置位。如果从 LLS 中唤醒，则 LLWU 中断程序将先得到处理。如果 LLWU 服务程序代码未清除模块标志和 NVIC 中的模块挂起中断标志，则在完成 LLWU 中断服务程序时将调用模块中断。如果已清除模块标志，则建议回读该标志，以确保在退出中断服务程序之前将其清除。有关更多详细信息，请参阅“操作序列化”部分。

14.1.4 经过滤波的唤醒输入

Kinetis K 中为多达两个外部 GPIO 唤醒引脚提供了毛刺滤波器功能。在后续 MCU 中，毛刺滤波器基于内部 LPO 时钟，因此将会抑制任何短于 3 个 LPO 周期的脉冲。

您可以写入两个滤波器控制寄存器（LLWU_FILT1 和 LLWU_FILT2），以从 16 个输入引脚中选择要发送到两个滤波器的引脚。为引脚使能其中一个滤波器并同时为同一引脚使能引脚功能并没有多大意义。同时使能引脚和滤波器时，会绕过滤波器功能，因为 MCU 将始终在边沿事件中唤醒。如果为引脚使能了滤波器，请不要选择边沿标志唤醒。例如，要为滤波器 1 使用 LLWU_P7，请将 0b111 写入 FILTSEL 位，并在 LLWU_FILT1 寄存器中使能滤波器边沿（FILTE）。然后，将 0b00 写入 LLWU_PE2 寄存器的引脚使能位，以禁止任何边沿导致唤醒事件。

14.1.5 唤醒源

注

$\overline{\text{RESET}}$ 也是唤醒源，具体取决于 LLWU_CS 寄存器中的位设置（对于 MC1），或 LLWU_RST 寄存器中的位设置 (对于 MC2)。有关如何配置复位引脚输入滤波的更多信息，请参阅复位控制模块（RCM）中的复位滤波器设置。

表 7. LLWU 输入的示例唤醒源（有关特定 MCU 的实现，请参阅参考手册“芯片配置”部分）

输入	唤醒源	输入	唤醒源
LLWU_P0	PTE1/LLWU_P0 引脚	LLWU_P12	PTD0/LLWU_P12 引脚
LLWU_P1	PTE2/LLWU_P1 引脚	LLWU_P13	PTD2/LLWU_P13 引脚
LLWU_P2	PTE4/LLWU_P2 引脚	LLWU_P14	PTD4/LLWU_P14 引脚
LLWU_P3	PTA4/LLWU_P3 引脚 [1, 2]	LLWU_P15	PTD6/LLWU_P15 引脚
LLWU_P4	PTA13/LLWU_P4 引脚	LLWU_M0IF	LPTMR[3]
LLWU_P5	PTB0/LLWU_P5 引脚	LLWU_M1IF	CMP0
LLWU_P6	PTC1/LLWU_P6 引脚	LLWU_M2IF	CMP1

下一页继续介绍此表...

表 7. LLWU 输入的示例唤醒源（有关特定 MCU 的实现，请参阅参考手册“芯片配置”部分） (继续)

输入	唤醒源	输入	唤醒源
LLWU_P7	PTC3/LLWU_P7 引脚	LLWU_M3IF	CMP2
LLWU_P8	PTC4/LLWU_P8 引脚	LLWU_M4IF	TSI
LLWU_P9	PTC5/LLWU_P9 引脚	LLWU_M5IF	RTC 警报
LLWU_P10	PTC6/LLWU_P10 引脚	LLWU_M6IF	DryIce（篡改检测）
LLWU_P11	PTC11/LLWU_P11 引脚	LLWU_M7IF	RTC 秒钟

1. 对于 Kinetis Rev 1.x 器件，当从 VLLSx 模式唤醒 MCU（已使能 EzPort）时，在此引脚上保持低电平的下降沿输入将导致在复位序列中进入 EzPort 模式。从任何非 VLLSx 模式唤醒 MCU（已在其端口控制寄存器中选择 NMI 功能）时，在此引脚上保持低电平的下降沿输入将在低功耗模式恢复时响应 NMI 异常。在禁用 EzPort 的情况下从 VLLSx 模式恢复时，会出现相同现象；否则，会进入 EzPort 模式。
2. 对于 MC2 器件，仅在芯片复位而非 *VLLS* 时检查 $\overline{\text{EZP\_CS}}$ 信号，因此通过非复位源实现的 VLLS 唤醒不会导致进入 EzPort 模式。如果在进入 LLS/VLLS 时使能了 NMI，则置位 NMI 引脚会在退出低功耗模式时产生 NMI 中断。在 Kinetis 器件上，也可以通过 FOPT[NMI_DIS]位禁用 NMI。
3. 需要使能外设和外设中断。LLWU 的 WUME 位可使能内部模块标志作为唤醒输入。唤醒后，会根据外设清除机制清除这些标志。

15 参考文献和修订历史记录

15.1 参考文献

下列参考文献包含与 Kinetis 功耗管理相关的其他信息。您可以在 Kinetis（www.freescale.com/Kinetis）网页上选择一个器件，然后选择“文档”选项卡，来查找特定参考手册、数据手册或勘误表。如需查找以下应用笔记，请搜索文档编号。

15.1.1 Kinetis 微控制器（MCU）

Kinetis MCU 包含多个基于 ARM® Cortex®-M0+和-M4-的 MCU 系列。这些 MCU 软硬件互相兼容，具有出类拔萃的低功耗性能和扩展性，并集成了丰富的功能和特性。在编写本应用笔记时，共有以下 8 个系列

1. Kinetis K - 范围广泛且具有可扩展性和兼容性，50-180MHz，32 KB 至 2 M Flash，高达 256 KB RAM，浮点单元，安全、模拟、定时器和串行接口
2. Kinetis L - 世界上最节能的微控制器，高达 48 MHz，8KB-256KB Flash，高达 32 KB RAM，低功耗定时器和智能外设
3. Kinetis E - ESD/EMC 增强性能，高达 48 MHz，8 KB-128 KB Flash，高达 8K RAM，ADC，Flextimer，高电流输出。
4. Kinetis EA - 汽车级质量标准，高达 48 MHz，8 KB-128 KB Flash，高达 8K RAM，ADC，Flextimer，高电流输出。
5. Kinetis MINI - 世界上基于 ARM 的最小微控制器，采用芯片级封装。
6. Kinetis M - 仪表和测量应用，50 MHz，32 KB-128 KB Flash，高达 32 KB SRAM，高精度 Σ-Δ 模拟前端，安全和串行接口。
7. Kinetis V - 设计用于电机控制和数字电源转换，75-200 MHz，16 KB-2MB Flash，高达 256 KB SRAM，高速模拟和定时外设，数学加速器。
8. Kinetis W - 集成亚 1 GHz 和 2.4 GHz 射频收发器，48-50 MHz，32 KB-512 KB Flash，高达 64 KB SRAM，针对嵌入式无线解决方案优化。

请注意，这些 MCU 上的系统模式控制器并不相同。即便属于同一系列，例如 K20 系列，系统模式控制器将随着时间的推移出现一些显著改进。请勤于查阅所用 MCU 的每个参考手册和数据手册，以确保其具有应用所需的特性。

有关详细文档和更新，请访问 http://www.freescale.com/kinetis。

15.1.2 参考文档和链接

您可以通过以下参考资料访问正在增长的 MCU 应用设计人员社区。

- **MCU** 参考手册。 参考手册的“芯片配置”章节以及特定模块章节包含特定 MCU 的实现详细信息。请仔细了解每款 MCU 的时钟、复位和功耗管理功能说明。
- **MCU** 数据手册说明书。 数据手册包含所有 MCU 规格，包括时钟速率、低功耗模式功耗预期值。
- **MCU** 勘误表。 器件勘误表标识因 MCU 问题而未能实现的功能和/或规格。
- **Kinestis SDK**。 KSDK 下载文档文件夹中的软件 HAL 和驱动程序参考手册。
- **AN4447**，***Freescale MQX***™低功耗管理。 飞思卡尔提供 MQX，一款全功能的免费实时操作系统（RTOS）。从版本 3.8 开始，MQX 集成了低功耗管理（LPM）驱动程序，可以在 MQX 应用程序中利用低功耗工作模式。
- **AN4470**，在 ***Kinetis*** 系列产品上使用低功耗模式。包括可在带模式控制器 2 的 Kinetis 器件上使用的低功耗进入演示代码。
- **Kinetis Solutions Advisor**。访问并运行 Kinetis Solutions Advisor，以帮助选择所需的 MCU，网址：http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=SOLUTION_ADVISOR。
- **Kinetis** 社区。 请由此访问 Kinetis 社区：https://community.freescale.com/community/kinetis
- **MBED** 社区。 请由此访问 Kinetis MBED 社区：http://developer.mbed.org/teams/Freescale/，或者
- 您可以使用 Kinetis 提供的 Tower 和 Freedom 套件评估低功耗模式。

15.1.3 修订历史记录

下表概述了本文档自上次发布之后所做的修改。

表 8. 修订历史记录

修订版本号	日期	重大变更
1	2011 年 11 月	增加了新款 Kinetis 器件以及新低功耗模块（包括 Kinetis L 系列 MCU）的详细信息。
2	2015 年 3 月	增加了新款 Kinetis 器件（包括 K20、K61、K70、K22F、KVx、K65、K66、K24 等）的详细信息。 增加了“功耗管理技术”章节 删除了电源技术详细信息 增加了用于进入和退出低功耗模式的 Kinetis SDK 示例代码 删除了对 Coldfire+ MCU 的引用 增加了 MCU 时钟和示意图更新，包括 MCG Lite。 通篇更新了文档，以加入新功能和思想。 增加了关于进入低功耗模式之前所需操作的章节。用例包括 DDR 和 LPDDR 存储器。 增加了关于模式进入和退出的表格 增加了对 SDK API 参考手册的引用。 增加了 LLS2 和 LLS3 模式进入注释和代码。 更新了功耗模式状态示意图。

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

本文档中的信息仅供系统和软件实施方使用 Freescale 产品。 本文并未明示或者暗示授予利用本文档信息进行设计或者加工集成电路的版权许可。 Freescale 保留对此处任何产品进行更改的权利，恕不另行通知。

Freescale 对其产品在任何特定用途方面的适用性不做任何担保、表示或保证，也不承担因为应用程序或者使用产品或电路所产生的任何责任，明确拒绝承担包括但不局限于后果性的或附带性的损害在内的所有责任。 Freescale 的数据表和/或规格中所提供的“典型”参数在不同应用中可能并且确实不同，实际性能会随时间而有所变化。 所有运行参数，包括“经典值”在内，必须经由客户的技术专家对每个客户的应用程序进行验证。 Freescale 未转让与其专利权及其他权利相关的许可。 Freescale 销售产品时遵循以下网址中包含的标准销售条款和条件：freescale.com/SalesTermsandConditions.

Freescale, the Freescale logo, AltiVec, C-5, CodeTest, CodeWarrior, ColdFire, ColdFire+, C-Ware, Energy Efficient Solutions logo, Kinetis, mobileGT, PowerQUICC, Processor Expert, QorIQ, Qorivva, StarCore, Symphony, and VortiQa are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Airfast, BeeKit, BeeStack, CoreNet, Flexis, Layerscape, MagniV, MXC, Platform in a Package, QorIQ Qonverge, QUICC Engine, Ready Play, SafeAssure, SafeAssure logo, SMARTMOS, Tower, TurboLink, Vybrid, and Xtrinsic are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© 2015 Freescale Semiconductor, Inc.

© 2015 飞思卡尔半导体有限公司

Document Number AN4503
Revision 2, 04/2015