# AN12736

## K32L2B 功耗模式开关应用

版本 0 — 2021 年 2 月

应用笔记

目录



## 1 导言

K32L2B 微控制器系列为功耗敏感市场提供超低功耗功能。 为了满足这一要求，本系列微控制器实现了几种低功耗模式。本应用笔记向用户展示了每个功耗模式的详细信息，并在 SDK 功耗模式切换演示示例中提供了用户案例示例。本文给出了使用每种功耗模式的提示。

该 MCUXpressoSDK 为用户提供了健壮的外部驱动程序、堆栈、中间件和示例应用程序，旨在简化和加速任何恩智浦 MCU 的应用程序开发。MCUXpresso SDK 是免费的，包含了所有硬件抽象和外部驱动软件的允许开源许可下的完整源代码。

本应用笔记的重点是电源管理控制器（PMC）、系统模式控制器（SMC）、多用途时钟发生器 Lite 版本（MCG-Lite）和低泄漏唤醒单元（LLWU）。

## 2 K32L2B 微控制器的功耗模式

### 2.1 Cortex-M0+核中的基本功耗模式

Arm® Cortex-M0+使用 Arm Cortex-M 架构的基本功耗模式：RUN、Sleep 和 Deep Sleep。Cortex-M0+处理器 Sleep 模式降低了功耗。

- Sleep 模式：它停止处理器时钟。
- Deep Sleep 模式：它停止系统时钟，关闭 PLL 和闪存。

系统可以生成虚假的唤醒事件，例如，调试操作唤醒处理器。 因此，软件必须能够在这样的事件发生后使处理器返回 Sleep 模式。程序可能有一个空闲循环，使处理器回到 Sleep 模式。要进入低功耗模式（Sleep/DeepSleep），有三个指令通知处理器：

- 等待中断（WFI）：WFI 指令导致立即进入 Sleep 模式。 处理器执行 WFI 指令时，停止执行指令，进入 Sleep 模式。
- 等待事件（WFE）：WFE 指令根据一位事件寄存器的值（由 SEV 指令设置）导致进入 Sleep 模式。 当处理器执行 WFE 指令时，它检查事件寄存器的值如下：
  - 0=处理器停止执行指令并进入 Sleep 模式。
  - 1=处理器将寄存器设置为零并继续执行指令而不进入 Sleep 模式。
- 发送 EVent（SEV）：SEV-指令导致事件被发送给多处理器系统中的所有处理器。 它还设置本地事件寄存器。

在 Cortex-M0+核中，SCB 寄存器控制 WFI/WFE 指令后进入低功耗模式的行为。

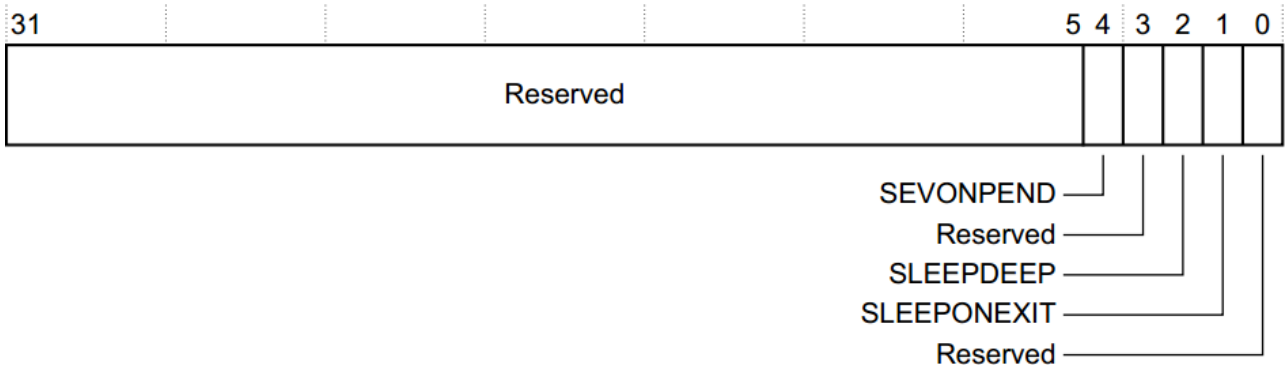


**图 1. 在 Arm 核中的 SCB 寄存器**

- SCB[SLEEPDEEP]位控制处理器是使用 Sleep 模式还是 DeepSleep 模式作为其低功耗模式：
  - 0=Sleep
  - 1=DeepSleep
- SCB[SLEEPONEXIT]位表示从 Handler 模式（中断服务例程）返回到线程模式（`main()`函数）时的 sleep-on-exit：
  - 0=返回线程模式时不进入 Sleep，直接返回 `main()`函数。
  - 1=从 ISR 返回到 Thread 模式时，再次进入 Sleep 或 DeepSleep，再也不要返回到 `main()`函数。将此位设置为 1 可使能一个中断驱动的应用程序以避免返回空主应用程序。
- 在挂起位上的 SCB[SEVONPEND]位发送事件：
  - 0=只有使能的中断或事件才能唤醒处理器，禁用的中断被排除在外。
  - 1=使能的事件和所有中断，包括禁用中断，都可以唤醒处理器。当事件或中断挂起时，事件信号从 WFE 唤醒处理器。 如果处理器没有等待事件，则该事件被注册并影响下一个 WFE。处理器在执行 SEV 指令或外部事件时也会唤醒。

这是说 WFE 作为 WFI 的一个轻量级版本来挂起 CPU 的执行，但不恢复和恢复上下文，这节省了更多的周期来进入和退出低功耗模式。

WFE 指令根据单比特事件寄存器的值条件性进入 Sleep 模式。当处理器执行 WFE 指令时，它检查事件寄存器的值：

- 0=处理器停止执行指令并进入 Sleep 模式。
- 1=处理器将寄存器设置为零并继续执行指令而不进入 Sleep 模式。

如果事件寄存器为 0，则 WFE 将暂停执行指令，直到发生以下事件之一：

- 异常，除非被异常掩码寄存器或当前优先级级别屏蔽。
- 如果设置了系统控制寄存器中的 SEVONPEND，则异常进入挂起状态。
- 一个调试输入请求，如果启用了调试。
- 使用 SEV 指令的多处理器系统中由外部设备或其他处理器发出信号的事件。

## 2.2 K32L2B 扩展功耗模式

在 K32L2B 中，该核使用 WFI 指令调用 Sleep 和 DeepSleep 模式，但也扩展了功耗模式及其关系，如 表 1 所示。

**表 1. K32L2B 单片机上的功耗模式**

| Arm CM0+功耗模式 | K32L2B MCU 功耗模式 | 唤醒模块 | 是否复位？ |
|---|---|---|---|
| RUN | RUN，VLPR | — | — |

*下页继续...*

**表 1. K32L2B 单片机上的功耗模式（续上页）**

| Arm CM0+功耗模式 | K32L2B MCU 功耗模式 | 唤醒模块 | 是否复位？ |
|---|---|---|---|
| RUN | CPO | AWIC/NVIC | 否 |
| SLEEP | WAIT，VLPW | NVIC | 否 |
| DEEP SLEEP | STOP，VLPS | WIC | 否 |
| DEEP SLEEP | PSTOP1 | AWIC | 否 |
| DEEP SLEEP | PSTOP2 | AWIC/NVIC | 否 |
| DEEP SLEEP | LLS | LLWU | 否 |
| DEEP SLEEP | VLLSx(x=0/1/3) | LLWU | 是 |

NVIC 意味着任何中断源都可以从 WAIT/VLPW 模式唤醒 MCU。 AWIC 意味着只有参考手册中的 AWIC 唤醒源才能从停止/VLPS 模式唤醒 MCU。 LLWU 意味着只有参考手册中的 LLWU 唤醒源才能从 LLS/VLLSx 模式唤醒 MCU。 若要从 VLLSx 模式中唤醒，请通过复位流程并调用 LLWU 复位。 对于计算操作模式（CPO），Arm 核处于运行模式。任何异步中断和 ARM 核同步中断都可以唤醒 MCU 到运行模式。表 2 显示关于每个功耗模式的详细描述。

**表 2. 功耗模式说明**

| 模式 | 说明 |
|---|---|
| RUN | MCU 可以全速运行，内部电源完全调节，即在运行调节中。 这种模式也被称为正常运行模式。 |
| WAIT | 核时钟被关闭。 系统时钟继续运行。 总线时钟，如果启用，则继续操作。 运行调节保持。 |
| STOP | 核时钟被关闭。在来自支持外设的所有停止确认信号均有效后，关闭通往其他主设备的系统时钟和总线时钟。 |
| VLPR | 在这种模式下，核、系统、总线和闪存时钟的最大频率受到限制。有关最大允许频率的详细信息，参阅 *KL43 子系列参考手册*（文档 KL43P64M48SF6RM）中的**功耗管理**章节。 |
| VLPW | 核时钟被关闭。系统、总线和闪存时钟继续运行，尽管它们的最大频率受到限制。有关最大允许频率的详细信息，参阅 *KL43 子系列参考手册*（文档 KL43P64M48SF6RM）中的**功耗管理**章节。 |
| VLPS | 核时钟被关闭。在来自支持外设的所有停止确认信号均有效后，关闭通往其他主设备的系统时钟和总线时钟。 |
| LLS | 核时钟被关闭。在来自支持外设的所有停止确认信号均有效后，关闭通往其他主设备的系统时钟和总线时钟。通过降低内部逻辑电压，将 MCU 置于低泄漏模式。 保留所有系统 RAM 内容，内部逻辑和 I/O 状态。 |
| VLLS3 | 核时钟被关闭。在来自支持外设的所有停止确认信号均有效后，关闭通往其他主设备的系统时钟和总线时钟。 通过降低内部逻辑电压，MCU 被放置在低泄漏模式下。 保留所有系统 RAM 内容并保持 I/O 状态。 内部逻辑状态不保留。 |
| VLLS1 | 核时钟被关闭。在来自支持外设的所有停止确认信号均有效后，关闭通往其他主设备的系统时钟和总线时钟。 通过关闭内部逻辑和所有系统 RAM 的电源，将 MCU 置于低泄漏模式。I/O 状态被保留。内部逻辑状态不保留。 |

*下页继续...*

**表 2. 功耗模式说明（续上页）**

| 模式 | 说明 |
| --- | --- |
| VLLS0 | 核时钟被关闭。在来自支持外设的所有停止确认信号均有效后，关闭通往其他主设备的系统时钟和总线时钟。 通过关闭内部逻辑和所有系统 RAM 的电源，将 MCU 置于低泄漏模式。I/O 状态被保留。内部逻辑状态不保留。禁用 1 kHz LPO 时钟，可以使用 STOPCTRL[PORPO]选择性地使能电源复位（POR）电路]。 |

对于 K32L2B 系列器件，NMI 引脚可以唤醒所有功耗模式，而如果复位引脚未被总线时钟过滤，则复位引脚会将 MCU 的功耗模式复位为默认的 RUN 模式。

图 2 显示芯片上可用的功耗模式状态转换。任何复位都会使 MCU 恢复到正常的运行状态。

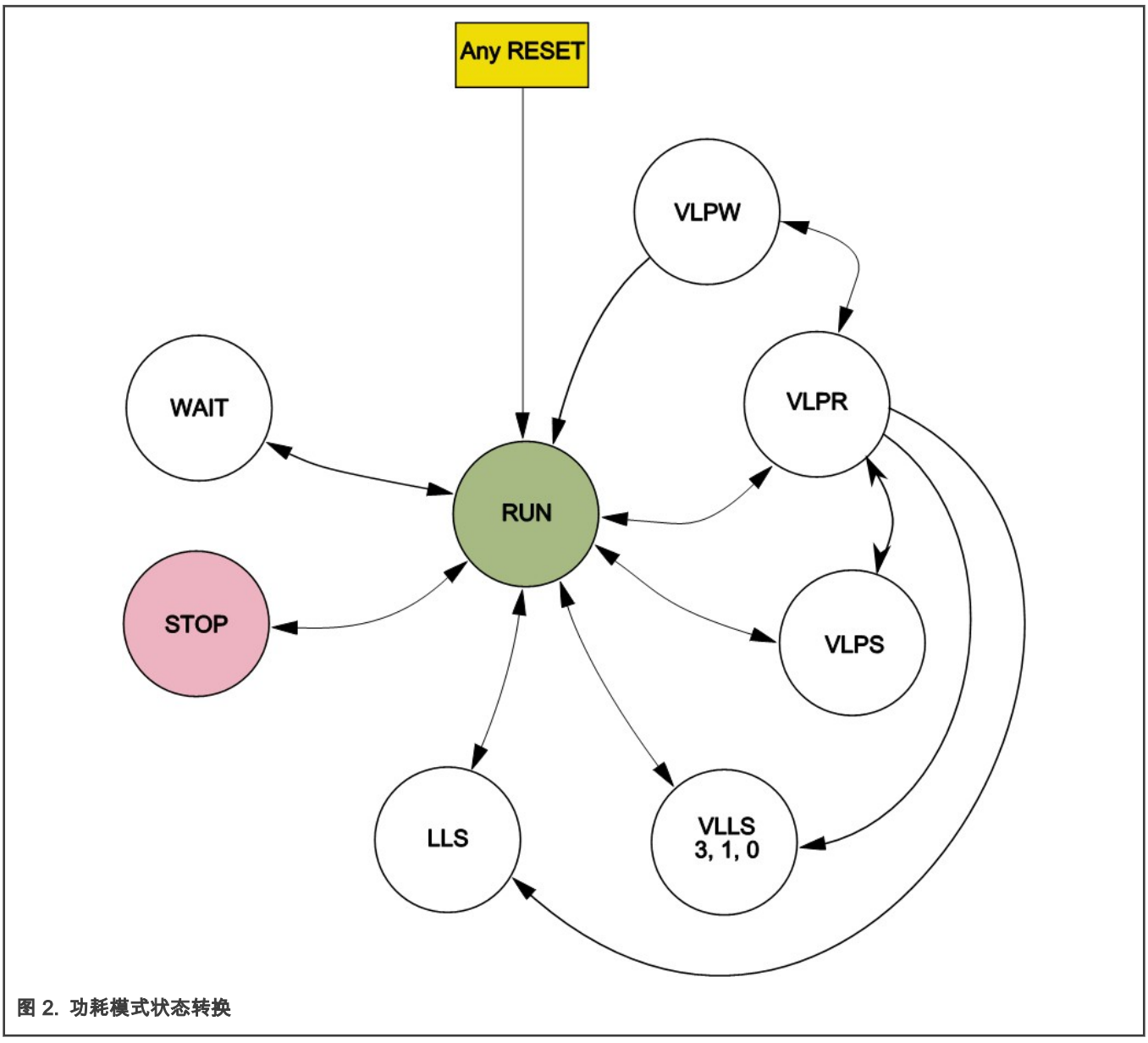


**图 2. 功耗模式状态转换**

从 RUN/VLPR 模式进入目标功耗模式：

- 禁用电源管理控制器（PMC）模块中的低电压检测（LVD）功能可以忽略在一些超低功耗模式下 LDO 降低电源电压时的警告。

- 在目标低功耗模式下禁用不必要的模块/引脚以节省功耗，并设置时钟唤醒源模块以触发低功耗退出事件。
- 为目标功耗模式设置时钟源，使现存模块仍然可以在可用时钟源下工作。
- 在 SMC->PMPROT（电源模式保护）寄存器中解锁所指示的功耗模式，从而可以进入目标功耗模式。
- 在 SMC->PMCTRL（电源模式控制）寄存器和 SMC->停止控制（停止控制)中设置目标模式。
- 调用 WFI 或 WFE 进入到低功耗模式。 退出低功耗模式：
- 等待预先设置的唤醒源被唤醒事件触发。
- 对于 VLLSx 模式，LLWU 特别设计为唤醒模块，以收集所有可用的唤醒源。

对于来自复位的一些唤醒例程，还需要清除 PMC->REGSC[ACKISO]位来解锁端口引脚，该引脚在一些超低功耗模式下被锁定并保持稳定。

关于唤醒源，表 3 显示超低功耗模式（LLS 和 VLLSx）中的现存模块。

**表 3. 超低功耗模式下的现存模块**

| Modules | VLPR | VLPW | Stop | VLPS | LLS | VLLSx |
|---|---|---|---|---|---|---|
| 核模块 | | | | | | |
| NVIC | FF | FF | static | static | static | OFF |
| 系统模块 | | | | | | |
| Mode controller | FF | FF | FF | FF | FF | FF |
| LLWU | static | static | static | static | FF | FF |
| Regulator | Low power | Low power | ON | Low power | Low power | 在 VLLS3 低功率，在 VLLS0/1 关闭 |
| Brown-out Detection | ON | ON | ON | ON | ON | 在 VLLS1/3 启用，可选地在 VLLS0 中禁用 |
| 时钟 | | | | | | |
| 1 kHz LPO | ON | ON | ON | ON | ON | 在 VLLS1/3 启用，在 VLLS0 关闭 |
| 系统振荡器（OSC） | OSCERCLK<br>最大 16 MHz 结晶 | OSCERCLK<br>最大 16 MHz 结晶 | OSCERCLK<br>可选的 | OSCERCLK<br>最大 16 MHz 结晶 | OSCERCLK<br>最大 16 MHz 结晶 | OSCERCLK<br>在 VLLS1/3 最大 16 MHz 结晶，在 VLLS0 关闭 |
| 内存和内存接口系统模块 | | | | | | |
| SRAM_U 和 SRAM_L | Low power | Low power | Low power | Low power | Low power | 在 VLLS3 低功率，在 VLLS0/1 关闭 |
| 系统注册文件 | Powered | Powered | Powered | Powered | Powered | Powered |
| 计时器 | | | | | | |

*下页继续...*

表 3. 超低功耗模式下的现存模块（续上页）

| Modules | VLPR | VLPW | Stop | VLPS | LLS | VLLSx |
|---|---|---|---|---|---|---|
| LPTMR | FF | FF | 异步操作在 PSTOP2 中的 FF | 异步操作 | 异步操作 | 异步操作 |
| RTC | FF 在 CPO 中的异步操作 | FF | 异步操作 FF 在 PSTOP2 | 异步操作 | 异步操作 | 异步操作 |
| 人机界面 | | | | | | |
| RTC | FF 在 CPO 中的异步操作 | FF | 异步操作 FF 在 PSTOP2 | 异步操作 | 异步操作 | 在 VLLS0 中关闭异步操作 |

# 3 应用-测量各种功耗模式下的电流

本文设计了一种应用软件，用于测量 K32L2BMCU 在各种功耗模式下的电流。采用 FRDM-K32L2B 板作为主要硬件平台。板上的两个按钮用于切换 SLCD 屏幕上的目标功耗模式选择。

## 3.1 板设置

FRDM-K32L2B 板有测量插座，但需要一些额外的硬件工作，才能使其在应用中可用。在原理图中，J20 是预期的测量插座，如图 3 所示。

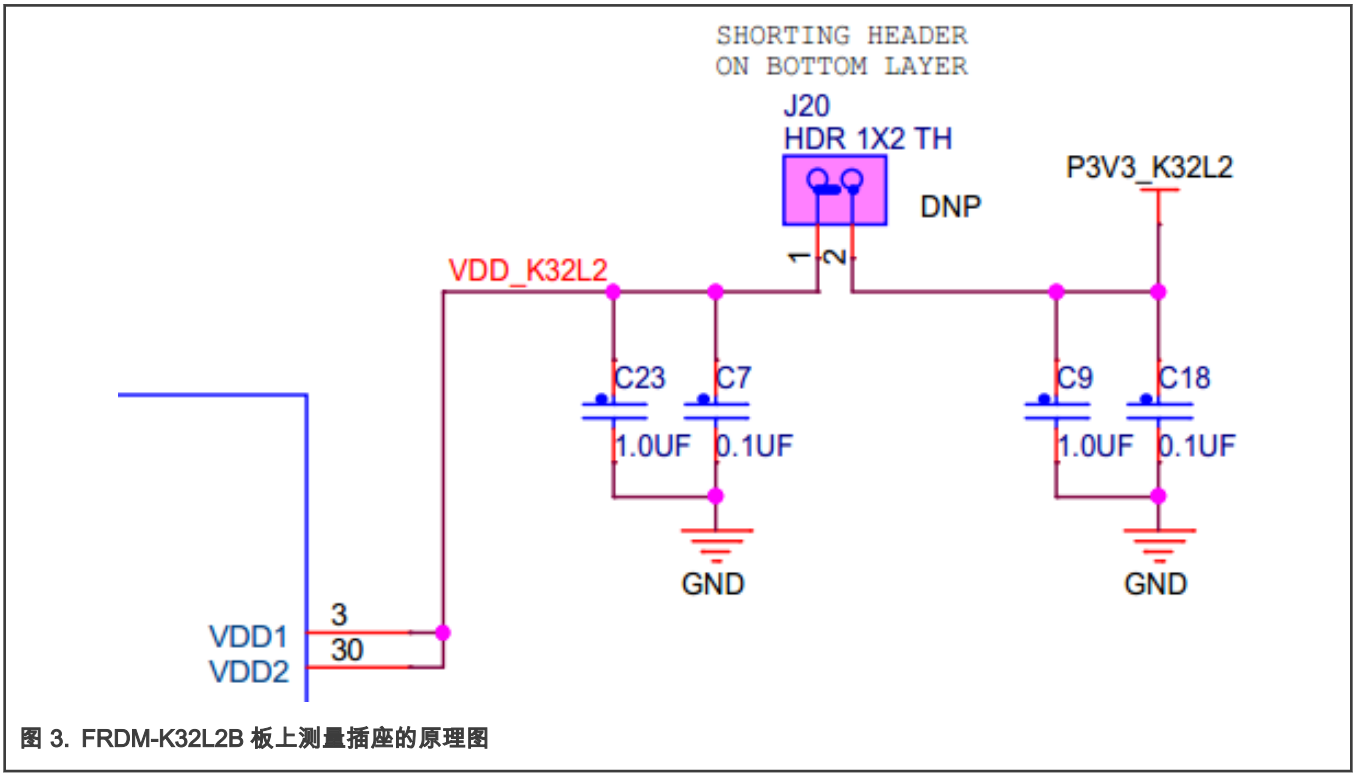


图 3. FRDM-K32L2B 板上测量插座的原理图

然而，在默认情况下，J20 在板上是短路的。要测量进入 VDD 的电流，我们需要切断这个连接，并将万用表（在电流测量模式下）放入串联连接中，如 图 4 所示。

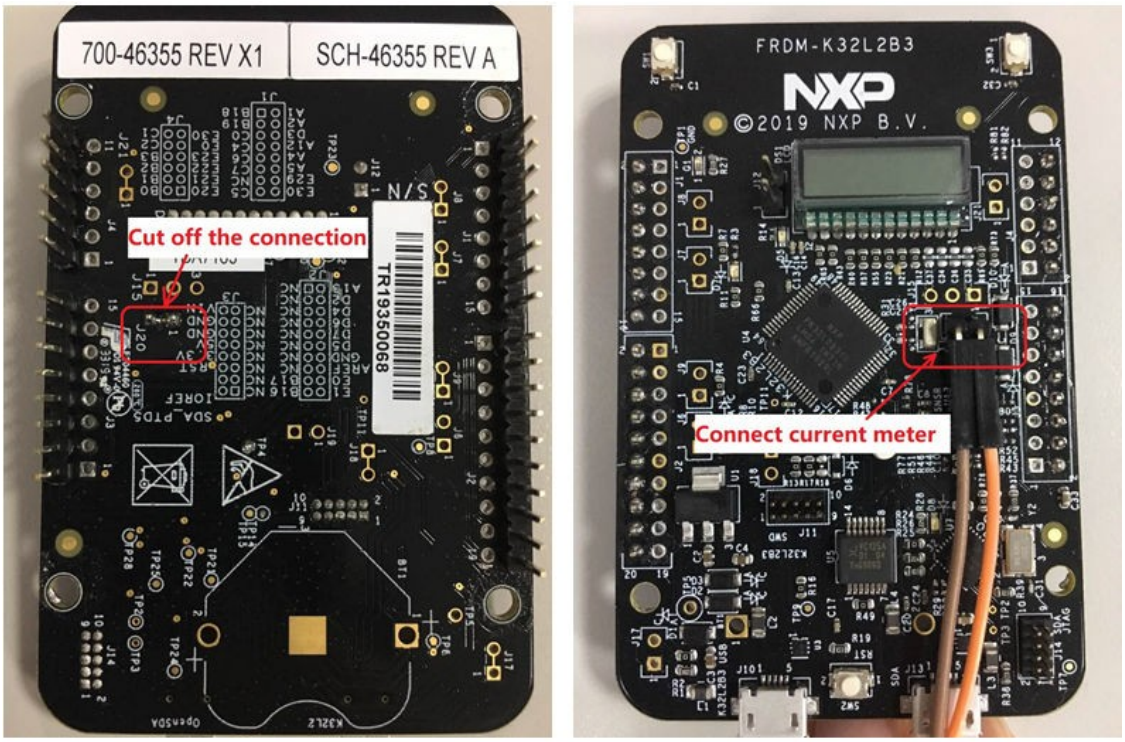


**图 4. FRDM-K32L2B 测量插座**

若要在低功耗模式期间启用 SLCD 显示，请使用板上按钮（SW1 和 SW3）和 SLCD 屏幕切换功耗模式并显示状态，如 图 5所示。

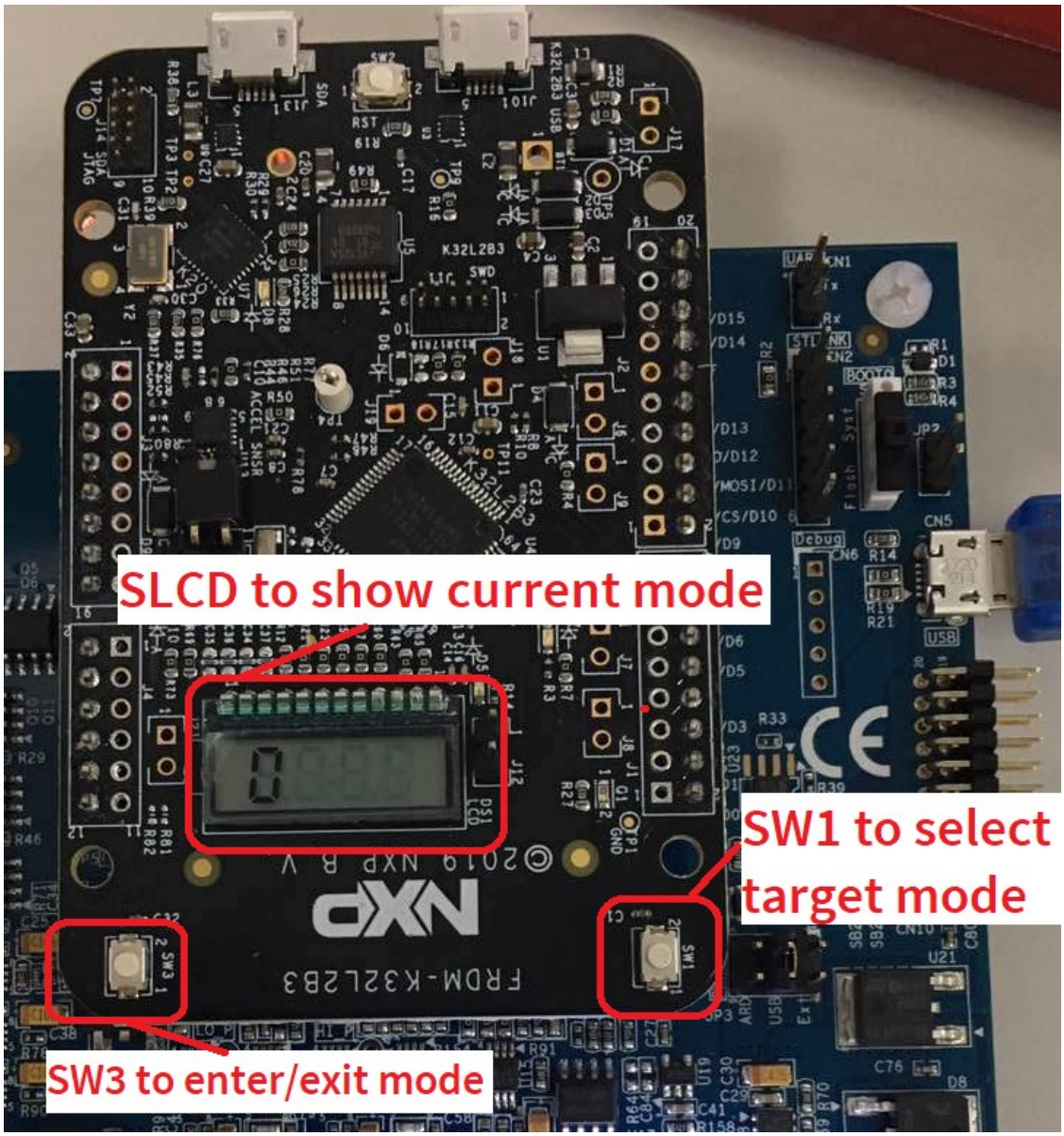


**图 5. 使用按钮切换电源模式**

## 3.2 软件设计

MCUXpresso SDK 软件包为 SMC、PMC、LLWU 和时钟模块提供驱动程序。在应用软件中，我们可以使用这些驱动程序 API 与其他外部驱动程序一起操作功耗模式。

### 3.2.1 用软件切换功耗模式

本应用程序演示包括 10 种功耗模式：

```
/* Power mode definition used in application. */
typedef enum _app_power_mode
{
    kAPP_PowerModeRun, /* Normal RUN mode */
    kAPP_PowerModeWait, /* WAIT mode. */
    kAPP_PowerModeStop, /* STOP mode. */
```

```
    kAPP_PowerModeVlpr, /* VLPR mode. */
    kAPP_PowerModeVlpw, /* VLPW mode. */
    kAPP_PowerModeVlps, /* VLPS mode. */
    kAPP_PowerModeLls, /* LLS mode. */
    kAPP_PowerModeVlls0, /* VLLS0 mode. */
    kAPP_PowerModeVlls1, /* VLLS1 mode. */
    kAPP_PowerModeVlls3, /* VLLS3 mode. */
    kAPP_PowerModeMax
} app_power_mode_t;
```

APP_PowerModeSwitch()功能是执行功耗模式切换最重要的函数。它使用 SMC 驱动程序的 API 来控制目标功耗模式：

```
void APP_PowerModeSwitch(smc_power_state_t curPowerState, app_power_mode_t
targetPowerMode)
{
    smc_power_mode_vlls_config_t vlls_config;
    vlls_config.enablePorDetectInVlls0 = true;

    switch (targetPowerMode)
    {
        case kAPP_PowerModeVlpr:
            APP_SetClockVlpr(); /* setup the lower clock source for VLPR. */
            SMC_SetPowerModeVlpr(SMC);
            while (kSMC_PowerStateVlpr != SMC_GetPowerModeState(SMC))
            {
             }
            break;

        case kAPP_PowerModeRun:
             /* Power mode change. */
             SMC_SetPowerModeRun(SMC);
             while (kSMC_PowerStateRun != SMC_GetPowerModeState(SMC))
             {
              }
             /* If enter RUN from VLPR, change clock after the power mode change.
*/
             if (kSMC_PowerStateVlpr == curPowerState)
             {
                APP_SetClockRunFromVlpr(); /* setup the higher clock source for
RUN. */
              }
              break;

          case kAPP_PowerModeWait:
              SMC_PreEnterWaitModes();
              SMC_SetPowerModeWait(SMC);
              SMC_PostExitWaitModes();
              break;

          case kAPP_PowerModeStop:
             SMC_PreEnterStopModes();
             SMC_SetPowerModeStop(SMC, kSMC_PartialStop);
             SMC_PostExitStopModes();
             break;

          case kAPP_PowerModeVlpw:
             SMC_PreEnterWaitModes();
             SMC_SetPowerModeVlpw(SMC);
             SMC_PostExitWaitModes();
             break;
```

```
          case kAPP_PowerModeVlps:
              SMC_PreEnterStopModes();
              SMC_SetPowerModeVlps(SMC);
              SMC_PostExitStopModes();
              break;

          case kAPP_PowerModeLls:
              SMC_PreEnterStopModes();
              SMC_SetPowerModeLls(SMC);
              SMC_PostExitStopModes();
              break;

          case kAPP_PowerModeVlls0:
              vlls_config.subMode = kSMC_StopSub0;
              SMC_PreEnterStopModes();
              SMC_SetPowerModeVlls(SMC, &vlls_config);
              SMC_PostExitStopModes();
              break;

          case kAPP_PowerModeVlls1:
              vlls_config.subMode = kSMC_StopSub1;
              SMC_PreEnterStopModes();
              SMC_SetPowerModeVlls(SMC, &vlls_config);
              SMC_PostExitStopModes();
              break;

          case kAPP_PowerModeVlls3:
              vlls_config.subMode = kSMC_StopSub3;
              SMC_PreEnterStopModes();
              SMC_SetPowerModeVlls(SMC, &vlls_config);
              SMC_PostExitStopModes();
              break;

          default:
              break;
       }
}
```

进入一种新的功耗模式，就像在操作系统中切换到一种具有新的工作条件的新任务一样。一些操作应该在进入之前完成，以关闭当前模式，并为新模式做准备。当进入一种新的模式时，也应该做一些初步的工作。因此，在应用程序演示代码中，创建了 APP_PowerPreSwitchHook()和 APP_PowerPostSwitchHookAPP_PowerPreSwitchHook()的功能来封装这些操作。为了使 MCU 的成本尽可能低，在进入等待/停止模式之前禁用不必要的外部设备，并在唤醒后恢复。在应用程序演示中，用于终端交互的 UART 外部设备和输出引脚在低功耗模式下被禁用。在 MCU 休眠期间，只有由 OSC32 时钟驱动的 SLCD 和 NVIC/AWIC 仍然开启。

### 3.2.2 使用板载按钮选择模式

使用 SW1（PTA4）和 SW3（PTC3）进入选定的目标模式并退出。他们完全由 ISR 驱动：

```
/* PTC3. */
void APP_WAKEUP_BUTTON_IRQ_HANDLER(void)
{
    if ((1U <<APP_WAKEUP_BUTTON_GPIO_PIN) &
PORT_GetPinsInterruptFlags(APP_WAKEUP_BUTTON_PORT))
    {
         /* Disable interrupt. */
         PORT_SetPinInterruptConfig(APP_WAKEUP_BUTTON_PORT,
APP_WAKEUP_BUTTON_GPIO_PIN, kPORT_InterruptOrDMADisabled);
          PORT_ClearPinsInterruptFlags(APP_WAKEUP_BUTTON_PORT, (1U <<
```

```
APP_WAKEUP_BUTTON_GPIO_PIN));
           sw3_irq_done = true;
      }
}

/* PTA4. */
void PORTA_IRQHandler(void)
{
    uint32_t flags = PORT_GetPinsInterruptFlags(PORTA);
    PORT_ClearPinsInterruptFlags(PORTA, flags); /* clear flags. */

     /* PTA4. */
     if ( 0u != ((1u << 4u) & flags) )
     {
         sw1_irq_counter = (sw1_irq_counter+1u)% 10u;

         slcd_set_number(0u, sw1_irq_counter, false);
         PRINTF(".");
      }
}

app_power_mode_t APP_GetTargetPowerMode(void)
{
    sw3_irq_done = false;

     /* enable the pin detection. */
     PORT_SetPinInterruptConfig(PORTA, 4u, kPORT_InterruptFallingEdge);

     /* wait for a new sw3_irq. */
     PORT_SetPinInterruptConfig(PORTC, 3u, kPORT_InterruptFallingEdge);
     while (!sw3_irq_done)
     {}
      /* disable the pin detection. */
      PORT_SetPinInterruptConfig(PORTA, 4u, kPORT_InterruptOrDMADisabled);

      PRINTF("%d\r\n", sw1_irq_counter);

      return (app_power_mode_t)(sw1_irq_counter);
}
```

只在 APP_GetTargetPowerMode()函数运行期间激活 SW1。在此函数中按下 SW3 后，SW1 再次锁定。然后，SW1 先前增加的值将作为选择返回。

也可使用 SW3 作为唤醒源：

- 当 NVIC 仍然有效时，SW3 使用端口中断唤醒 MCU 并退出低功耗模式。
- 当 NVIC 在某些超低功耗模式下关闭时，SW3 被路由到 LLWU，它可以通过 AWIC 唤醒 MCU。

### 3.2.3 在低功耗模式下保存内存

为了显示在不同的功耗模式内存是否仍然可以将数据保持，里面写有软件令牌的变量用来表示内容丢失与否。 在应用程序演示中，在进入低功耗 Sleep 模式之前，将令牌 APP_LOW_POWER_MEM_TOKEN 写入 app_always_keep_value 变量，然后在 MCU 再次唤醒后读取它。对于需要复位例程唤醒的低功耗模式，软件还将在编写新令牌之前读取令牌变量。 每次当 MCU 被唤醒时，无论是从复位还是就地，软件都会读取令牌变量并与期望值进行比较，然后在 SLCD 屏幕和 UART 终端上告诉用户。

```
#define APP_LOW_POWER_MEM_TOKEN 0x55555555

volatile uint32_t app_always_keep_value; /* keep the token value during in low
power modes. */
```

```
int main(void)
{
    ...

    /* Unlock all the power modes of chip. */
    SMC_SetPowerModeProtection(SMC, kSMC_AllowPowerModeAll);

    /* Clear the low power lock bit. */
    if (kRCM_SourceWakeup & RCM_GetPreviousResetSources(RCM)) /* Wakeup from
VLLS. */
    {
        PMC_ClearPeriphIOIsolationFlag(PMC);
        NVIC_ClearPendingIRQ(LLWU_IRQn);
        PRINTF("\r\nMCU wakeup from VLLS modes...\r\n");

        if (APP_LOW_POWER_MEM_TOKEN == app_always_keep_value)
        {
            slcd_set_number(2, 1, false);
            PRINTF("memory value is kept.\r\n");
         }
         else
         {
            slcd_set_number(2, 0, false);
            PRINTF("memory value is missed.\r\n");
          }
    }
    else
    {
         PRINTF("\r\npower mode switch example.\r\n");
    }

    while (1)
    {
          ...
          /* Wait for user response */
          targetPowerMode = APP_GetTargetPowerMode();

          /* only go next with avaiable and right input. */
          if (1u == APP_CheckPowerMode(curPowerState, targetPowerMode))
          {
              APP_PowerPreSwitchHook(curPowerState, targetPowerMode);
               /* setup wakeup source. */
               if ( (kAPP_PowerModeRun != targetPowerMode)
                   && (kAPP_PowerModeVlpr != targetPowerMode) )
               {
                   APP_SetWakeupConfig(targetPowerMode);
                }

              APP_PowerModeSwitch(curPowerState, targetPowerMode);
              APP_PowerPostSwitchHook(curPowerState, targetPowerMode);
           }
   }

static void APP_PowerPreSwitchHook(smc_power_state_t originPowerState,
app_power_mode_t targetMode)
{
    /* setup a token in memory. */
    app_always_keep_value = APP_LOW_POWER_MEM_TOKEN;
    slcd_set_number(2, SLCD_ON_SHOW_NUMBER_NONE, false);
```

```
    PRINTF("Set a token in memory\r\n");

    ...
}
static void APP_PowerPostSwitchHook(smc_power_state_t originPowerState,
app_power_mode_t targetMode)
{
     /* check the token. */
     if (APP_LOW_POWER_MEM_TOKEN == app_always_keep_value)
     {
         slcd_set_number(2, 1, false);
         PRINTF("token value is kept.\r\n");
      }
      else
      {
         slcd_set_number(2, 0, false);
         PRINTF("token value is missed.\r\n");
       }

       ...
}
```

## 3.3 运行应用程序演示项目

最后，在编译项目并将镜像下载到 FRDM-K32L2B 板后，应用演示可以运行以测量不同功率模式下的工作电流。UART 终端可以输出日志信息。

让我们先试试万用表。如前所述，将万用表（在电流测量模式下）放入 J20 的串联连接中。将板载调试器连接到 PC 上，打开 UART 通信的终端工具（9600，8，N），如 图 6 所示。

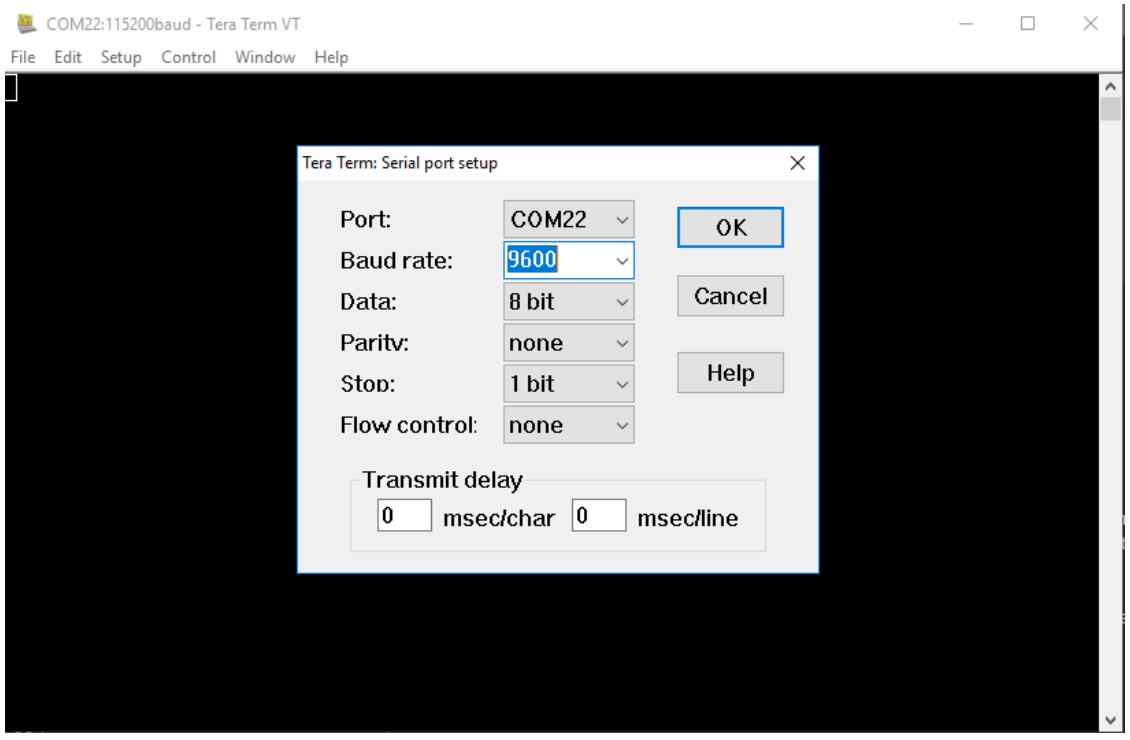


**图 6. UART 终端设置**

现在应用程序演示正在运行。

初始功耗模式为 RUN。 对于 RUN 模式，SLCD 显示 0，如 图 7 所示。


**图 7. 启动模式**

UART 终端还显示功耗模式选择的菜单，如 图 8 所示。

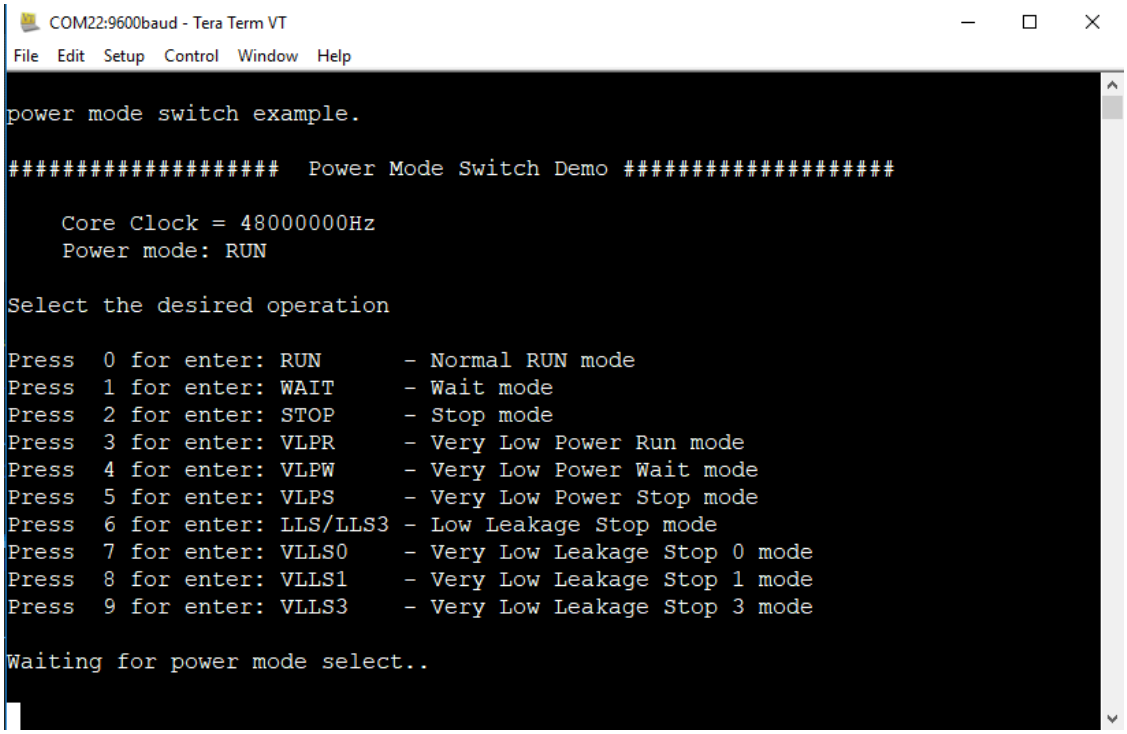


**图 8. 在 UART 终端上的电源模式选择**

然后按 SW1 增加数字，表示菜单中的功耗模式。每按一次，SLCD 上显示的数字逐个增加。此外，一个点显示在 UART 终端。例如，**2** 是用于停止模式。

一旦选择完成，按 SW3 进入目标模式。液晶显示器显示一个冒号来显示 MCU 正在睡觉，如 图 9 所示。在这种情况下，按 SW1 不会得到响应。

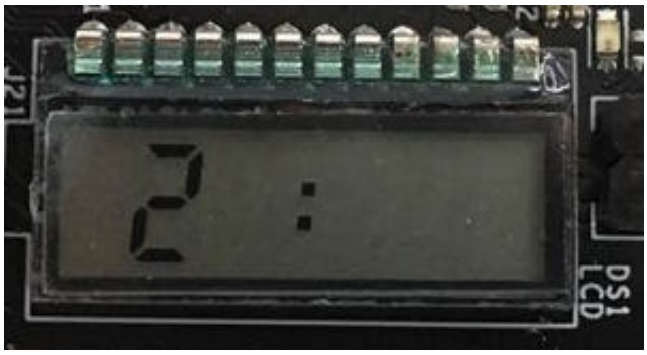


**图 9. 停止模式**

再次按 SW1 唤醒 MCU。则 SLCD 上的数字变为 0 和 1，如 图 10 所示。MCU 返回带有有效软件令牌的 RUN 状态。


**图 10. 运行模式与软件令牌可用**

注意

冒号消失了，这意味着 MCU 现在正在无睡眠的状态下运行。

1. 在左边意味着 MCU 仅返回运行模式。当 MCU 从 VLPS/VLPW 唤醒到 VLPR 时为 3。
2. 右边意味着变量中的令牌仍然保留。缺少令牌值时为 0。

对于 VLLS0（代码 7）只有一个例外。 当 MCU 处于休眠状态时，SLCD 什么也没有显示。此模式下关闭 SLCD 模式的电源。但在 SW3 被按下后，MCU 被唤醒，SLCD 再次上线。

按 SW1 选择其他功耗模式，按 SW3 进入或退出目标模式。

在切换-功耗模式的操作中，用户可以在万用表上读取电流值，这显示了实时功耗。

# 4 结论

在运行此应用程序演示时，我们测量了 FRDM-K32L2B 板上 K32L2B 的功耗条件。表 4 显示所有测量值。

注意

即使在超低功耗模式下，具有 OSC32 时钟源的 LLWU、SLCD 仍然处于活动状态，因为它们是专门为低功耗使用而设计的。

**表 4. 在 K32L2B 的各种功率模式下的功耗**

| 电源模式 | VDD_I | 记忆是否保存 | 描述 |
|---|---|---|---|
| RUN | 6.04 mA | 是 | 48 MHz 核时钟，启用 UART。 |

*下页继续...*

**表 4. 在 K32L2B 的各种功率模式下的功耗（续上页）**

| 电源模式 | VDD_I | 记忆是否保存 | 描述 |
|---|---|---|---|
| WAIT | 3.23 mA | 是 | 核休眠，禁用 UART，唤醒 NVIC。 |
| STOP | 0.16 mA | 是 | 核休眠，禁用 UART，唤醒 NVIC。 |
| VLPR | 0.21 mA | 是 | 4MHz 核时钟，启用 UART。 |
| VLPW | 0.10 mA | 是 | 核休眠，禁用 UART，唤醒 NVIC。 |
| VLPS | 8.7 uA | 是 | 核休眠，禁用 UART，唤醒 NVIC。 |
| LLS | 8.4 uA | 否 | 核休眠，禁用 UART，唤醒 LLWU，禁用 SLCD & OSC32。 |
| VLPS0 | 1.2 uA | 否 | 核休眠，禁用 UART，唤醒 LLWU，禁用 SLCD & OSC32。 |
| VLLS1 | 7.2uA | 否 | 核休眠，禁用 UART，唤醒 LLWU，禁用 SLCD & OSC32。 |
| VLLS3 | 7.7uA | 否 | 核休眠，禁用 UART，唤醒 LLWU，禁用 SLCD & OSC32。 |

*How To Reach Us*

**Home Page:**

nxp.com

**Web Support:**

nxp.com/support

**Limited warranty and liability** — Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

**Right to make changes** - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

**Security** — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.



For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

**Date of release: 2021 年 2 月**
**Document identifier: AN12736**

arm