AN13593

在 LPC5500 上使用 FreeRTOS 的无滴答模式

第 0 版 — 2022 年 3 月 21 日

应用笔记

目录

1 介绍

在开发裸跑的 MCU 时，用户可以直接配置相关的功能，使 MCU 进入低功耗模式。运行实时操作系统（RTOS）的 MCU 也需要低功耗模式。无滴答机制是当前各类小型 RTOS（如 FreeRTOS、EmbedOS、RTX）采用的一种通用的低功耗方法。本应用笔记介绍了如何在恩智浦 LPC5500 系列 MCU 上实现 FreeRTOS 的无滴答模式。

2 FreeRTOS 无滴答模式的工作原理

FreeRTOS 无滴答模式的原理是，在 MCU 执行空闲任务时，使 MCU 进入低功耗模式以节省系统功耗。调用中断等待（WFI）指令可使 MCU 进入低功耗模式，而使用中断可唤醒 MCU。

因此，FreeRTOS 无滴答模式的核心思想是：

- 当执行空闲任务时，使 MCU 进入低功耗模式
- 在适当条件下，通过中断或外部事件唤醒 MCU。

由于 FreeRTOS 是使用 Systick 定时器来产生系统滴答节拍并生成中断的，所以此中断也会唤醒 MCU，因此在进入低功耗模式之前必须禁用系统滴答中断。一旦系统滴答节拍停止，在退出低功耗模式后，系统滴答计数器将丢失一些计数值，这是不允许的。针对这种情况，FreeRTOS 也提供了相应的解决方案，即使用一个定时器记录系统处于低功耗模式的时间，并在退出低功耗模式后将此时间补偿到系统滴答计数器。

实现无滴答模式的关键步骤包括：

- 计算当前空闲任务的执行时间
- 配置系统参数并进入低功耗模式
- 退出低功耗模式并补偿系统滴答计数器

`vPortSuppressTicksAndSleep()` 函数是在 FreeRTOS 中定义的，可用于实现无滴答模式。用户可以将 `configUSE_TICKLESS_IDLE` 宏设置为 1 以实现无滴答模式。下面用 `vPortSuppressTicksAndSleep()`函数的源代码来解释这三个步骤。

2.1 计算低功耗模式的持续时间

当启用无滴答模式（`configUSE_TICKLESS_IDLE !=0`）时，一旦系统是空闲的，则在空闲任务中计算当前空闲任务的执行时间。

```
xExpectedIdleTime = prvGetExpectedIdleTime();
```

这个时间也是低功耗模式的最长持续时间，因为低功耗模式也可能被随机的外部事件唤醒，例如外部按键中断。

低功耗模式的持续时间也有一个上限。这是因为在进入低功耗模式后，用户必须使用定时器记录处于低功耗模式的时间。定时器的计数值是有限的，这决定了 MCU 能够进入到低功耗模式的最长时间长度。定时器可以是 Systick 或其他低功耗定时器之一。例如，系统时钟频率为 48 M，系统滴答节拍频率为 1 K，用户使用 Systick 的 24 位计数器记录低功耗状态的持续时间，Systick 可以记录的最长系统滴答节拍持续时间 `xMaximumPossibleSuppressedTicks` 计算如下：

```
ulTimerCountsForOneTick = ( configSYSTICK_CLOCK_HZ / configTICK_RATE_HZ );
xMaximumPossibleSuppressedTicks = portMAX_24_BIT_NUMBER / ulTimerCountsForOneTick;
                                = 0XFFFFFF/ (48000000/1000) = 349;
```

如果空闲任务 xExpectIdleTime 的持续时间超过了 `xMaximumPossibleSuppressedTicks`，则低功耗模式的最长时间应为 `xMaximumPossibleSuppressedTicks`。

然后根据有效的 `xExpectedIdleTime` 计算低功耗定时器的初始值。

```
ulReloadValue = portNVIC_SYSTICK_CURRENT_VALUE_REG + ( ulTimerCountsForOneTick * ( xExpectedIdleTime
- 1UL ) );
```

```
/* Make sure the SysTick reload value does not overflow the counter. */
if( xExpectedIdleTime > xMaximumPossibleSuppressedTicks )
{
    xExpectedIdleTime = xMaximumPossibleSuppressedTicks;
}

/* Stop the SysTick momentarily. The time the SysTick is stopped for is
 * accounted for as best it can be, but using the tickless mode will
 * inevitably result in some tiny drift of the time maintained by the
 * kernel with respect to calendar time. */
portNVIC_SYSTICK_CTRL_REG &= ~portNVIC_SYSTICK_ENABLE_BIT;

/* Calculate the reload value required to wait xExpectedIdleTime
 * tick periods. -1 is used because this code will execute part way
 * through one of the tick periods. */
ulReloadValue = portNVIC_SYSTICK_CURRENT_VALUE_REG + ( ulTimerCountsForOneTick * ( xExpectedIdleTime - 1UL ) );

if( ulReloadValue > ulStoppedTimerCompensation )
{
    ulReloadValue -= ulStoppedTimerCompensation;
}
```

图 1. 计算低功耗定时器的重载值

`ulReloadValue` 的值用于配置低功耗定时器。当计数器值减小到 0 时，MCU 会被唤醒。

2.2 进入低功耗模式

在进入低功耗模式之前，必须做一些准备工作，例如关闭 SysTick 滴答中断、配置唤醒源等。

从 2.1 节的描述来看，关闭 Systick 的滴答中断后，用户必须采用一个低功耗定时器（如 Systick 或 RTC）来记录低功耗状态的持续时间并唤醒 MCU。

在 FreeRTOS 源代码中，SysTick 被用作 MCU 的一个唤醒源，SysTick 的重载值被配置为 2.1 节中计算的 `ulReloadValue` 值。当 SysTick 的计数值减小到 0 时，MCU 会被唤醒。

```
/* Set the new reload value. */
portNVIC_SYSTICK_LOAD_REG = ulReloadValue;
```

```
/* Clear the SysTick count flag and set the count value back to zero. */
portNVIC_SYSTICK_CURRENT_VALUE_REG = 0UL;

/* Restart SysTick. */
portNVIC_SYSTICK_CTRL_REG |= portNVIC_SYSTICK_ENABLE_BIT;
```

如果用户想使用某个外部中断唤醒 MCU，还需要在系统进入低功耗状态之前配置外部中断唤醒源。在 MCU 被一个外部中断唤醒后，低功耗状态的持续时间可以通过 Systick 计数器的初始值和当前值计算得到。在 FreeRTOS 的源代码中没有将外部中断配置为唤醒源。

配置唤醒源后，MCU 可以进入睡眠模式。使用 WFI 指令可使 MCU 进入睡眠模式。此睡眠模式只会关闭核心时钟，不会影响外设的工作。

```
configPRE_SLEEP_PROCESSING( xModifiableIdleTime );

if( xModifiableIdleTime > 0 )
{
        _ asm volatile ( "dsb" ::: "memory" );
        _ asm volatile ( "wfi" );
        _ asm volatile ( "isb" );
}

configPOST_SLEEP_PROCESSING( xExpectedIdleTime );
```

除了 `vPortSuppressTicksAndSleep()` 函数之外，FreeRTOS 还为用户提供了另外两个接口函数：

- `configPRE_SLEEP_PROCESSING(xModifiableIdleTime);`
- `configPOST_SLEEP_PROCESSING(xExpectedIdleTime);`

FreeRTOS 仅提供了函数接口，用户要定义函数实体。在用户能够使 MCU 进入低功耗模式之前，必须调用 `configPRE_SLEEP_PROCESSING()` 来配置系统参数以降低系统功耗，例如关闭其他外设时钟，降低系统频率等。在退出低功耗模式后，调用 `configPOST_SLEEP_PROCESSING()` 函数以恢复系统主频和外设功能。

2.3 退出低功耗模式

当 MCU 被某个定时器中断或外部中断唤醒时，其在低功耗状态下的持续时间必须补偿到系统滴答计数器。`vPortSuppressTicksAndSleep()` 函数还提供了补偿时间的计算，如图 2 所示。

```
     * SysTick must have brought the system out of sleep mode). */
    if( ( portNVIC_SYSTICK_CTRL_REG & portNVIC_SYSTICK_COUNT_FLAG_BIT ) != 0 )
    {
        uint32_t ulCalculatedLoadValue;

        /* The tick interrupt is already pending, and the SysTick count
         * reloaded with ulReloadValue.  Reset the
         * portNVIC_SYSTICK_LOAD_REG with whatever remains of this tick
         * period. */
        ulCalculatedLoadValue = ( ulTimerCountsForOneTick - 1UL ) - ( ulReloadValue - portNVIC_SYSTICK_CURRENT_VALUE_REG );

        /* Don't allow a tiny value, or values that have somehow
         * underflowed because the post sleep hook did something
         * that took too long. */
        if( ( ulCalculatedLoadValue < ulStoppedTimerCompensation ) || ( ulCalculatedLoadValue > ulTimerCountsForOneTick ) )
        {
            ulCalculatedLoadValue = ( ulTimerCountsForOneTick - 1UL );
        }

        portNVIC_SYSTICK_LOAD_REG = ulCalculatedLoadValue;

        /* As the pending tick will be processed as soon as this
         * function exits, the tick value maintained by the tick is
         * stepped forward by one less than the time spent waiting. */
        ulCompleteTickPeriods = xExpectedIdleTime - 1UL;
    }
    else
    {
        /* Something other than the tick interrupt ended the sleep.
         * Work out how long the sleep lasted rounded to complete tick
         * periods (not the ulReload value which accounted for part
         * ticks). */
        ulCompletedSysTickDecrements = ( xExpectedIdleTime * ulTimerCountsForOneTick ) - portNVIC_SYSTICK_CURRENT_VALUE_REG;

        /* How many complete tick periods passed while the processor
         * was waiting? */
        ulCompleteTickPeriods = ulCompletedSysTickDecrements / ulTimerCountsForOneTick;

        /* The reload value is set to whatever fraction of a single tick
         * period remains. */
        portNVIC_SYSTICK_LOAD_REG = ( ( ulCompleteTickPeriods + 1UL ) * ulTimerCountsForOneTick ) - ulCompletedSysTickDecrements;
    }
```

图 2. 计算 SysTick 的补偿时间

在计算补偿时间时，首先要确定 MCU 是由低功耗定时器唤醒还是由外部中断唤醒的。

- 如果 MCU 是被 SysTick 唤醒的，则补偿时间为 `xExpectedIdleTime-1UL`。
- 如果 MCU 是被外部中断唤醒的，则意味着 SysTick 计数器的值尚未减少到 0。此时，需要根据 SysTick 计数器的初始值和当前值计算补偿时间。

然后调用 `vTaskStepTick()`函数将计算值 `ulCompleteTickPeriods` 补偿到 `xTickCount`。

```
vTaskStepTick( ulCompleteTickPeriods );

void vTaskStepTick( const TickType_t xTicksToJump )
{
     /* Correct the tick count value after a period during which the tick
      * was suppressed.  Note this does *not* call the tick hook function for
      * each stepped tick. */
     configASSERT( ( xTickCount + xTicksToJump ) <= xNextTaskUnblockTime );
     xTickCount += xTicksToJump;
     traceINCREASE_TICK_COUNT( xTicksToJump );
}
```

除了使用 SysTick 唤醒 MCU 外，用户还可以使用其他低功耗定时器，例如 RTC。用户也可以将 `configUSE_TICKLESS_IDLE` 宏的值设置为 2，从而可以自行定义 `vPortSuppressTicksAndSleep()` 函数来配置不同的唤醒源。

3 在 LPC5500 上实现 FreeRTOS 的无滴答模式

在第 2 章中，基于 FreeRTOS 的源代码解释了 FreeRTOS 无滴答模式的工作原理。本章将介绍如何在 LPC5500 平台上实现无滴答模式。恩智浦 SDK 为 LPC5500 系列 MCU 提供了一个 freertos_tickless 的示例。本应用笔记以 LPC55S16 SDK 中的 freertos_tickless 为例，描述在 LPC55S16 平台上实现无滴答模式的一些细节。在 LPC55S16 freertos_tickless 示例中，`configUSE_TICKLESS_IDLE` 宏的值为 2，且使用了用户自定义的 `vPortSuppressTicksAndSleep` 函数。进入低功耗模式后，RTC 时钟被用于记录系统进入低功耗模式的持续时间，同时用于唤醒 MCU，如图 4 所示。

freertos_tickless.c | FreeRTOSConfig.h

```
 * Application specific definitions.
 *
 * These definitions should be adjusted for your particular hardware and
 * application requirements.
 *
 * THESE PARAMETERS ARE DESCRIBED WITHIN THE 'CONFIGURATION' SECTION OF THE
 * FreeRTOS API DOCUMENTATION AVAILABLE ON THE FreeRTOS.org WEB SITE.
 *
 * See http://www.freertos.org/a00110.html.
 *----------------------------------------------------------*/

#define configUSE_PREEMPTION                    1
#define configUSE_TICKLESS_IDLE                 2
#define configCPU_CLOCK_HZ                      (SystemCoreClock)
#define configTICK_RATE_HZ                      ((TickType_t)1000)
#define configMAX_PRIORITIES                    5
#define configMINIMAL_STACK_SIZE                ((unsigned short)180)
#define configMAX_TASK_NAME_LEN                 20
#define configUSE_16_BIT_TICKS                  0
#define configIDLE_SHOULD_YIELD                 1
#define configUSE_TASK_NOTIFICATIONS            1
#define configUSE_MUTEXES                       1
#define configUSE_RECURSIVE_MUTEXES             1
#define configUSE_COUNTING_SEMAPHORES           1
#define configUSE_ALTERNATIVE_API               0 /* Deprecated! */
#define configQUEUE_REGISTRY_SIZE               8
#define configUSE_QUEUE_SETS                    0
#define configUSE_TIME_SLICING                  0
#define configUSE_NEWLIB_REENTRANT              0
#define configENABLE_BACKWARD_COMPATIBILITY     0
#define configNUM_THREAD_LOCAL_STORAGE_POINTERS 5
#define configOVERRIDE_DEFAULT_TICK_CONFIGURATION 1
```

图 3. 设置 configUSE_TICKLESS_IDLE 宏的值为 2

```
    else
    {
        /* Set the new reload value. */
        RTC_SetWakeupCount(pxRtcBase, ulReloadValue);

        /* Enable RTC. */
        RTC_StartTimer(pxRtcBase);

        /* Sleep until something happens.  configPRE_SLEEP_PROCESSING() can
        set its parameter to 0 to indicate that its implementation contains
        its own wait for interrupt or wait for event instruction, and so wfi
        should not be executed again.  However, the original expected idle
        time variable must remain unmodified, so a copy is taken. */
        xModifiableIdleTime = xExpectedIdleTime;
        configPRE_SLEEP_PROCESSING(xModifiableIdleTime);
        if (xModifiableIdleTime > 0)
        {
            __DSB();
            __WFI();
            __ISB();
        }
        configPOST_SLEEP_PROCESSING(xExpectedIdleTime);

        ulLPTimerInterruptFired = false;

        /* Re-enable interrupts - see comments above __disable_irq()
        call above. */
        __enable_irq();
        __NOP();
```

图 4. 设置 RTC 为唤醒源

在 FreeRTOS 开始调度之前，还定义了一个外部引脚中断来唤醒中断，如下所示。

```
/* Initialize PINT */
PINT_Init(PINT);
/* Setup Pin Interrupt 0 for falling edge */
PINT_PinInterruptConfig(PINT, kPINT_PinInt0, kPINT_PinIntEnableFallEdge, pint_intr_callback);
NVIC_SetPriority(BOARD_SW_IRQ, SW_NVIC_PRIO);
EnableIRQ(BOARD_SW_IRQ);
```

用户可以按下 LPC55S16-EVK 上的 SW3 按钮来唤醒 MCU。

在此示例中，定义了两个用户任务，即无滴答任务和切换任务，如下所示。

```
/*Create tickless task*/
if (xTaskCreate(Tickless_task, "Tickless_task", configMINIMAL_STACK_SIZE + 100, NULL,
tickless_task_PRIORITY, NULL) != pdPASS)
{
    PRINTF("Task creation failed!.\r\n");
    while (1)
    ;
}
if (xTaskCreate(SW_task, "Switch_task", configMINIMAL_STACK_SIZE + 100, NULL, SW_task_PRIORITY,
NULL) != pdPASS)
{
```

```
    PRINTF("Task creation failed!.\r\n");
    while (1)
    ;
}

/* Tickless Task */
static void Tickless_task(void *pvParameters)
{
    for (;;)
    {
        PRINTF("%d\r\n", xTaskGetTickCount());
        vTaskDelay(TIME_DELAY_SLEEP);
    }
}

/* Switch Task */
static void SW_task(void *pvParameters)
{
    xSWSemaphore = xSemaphoreCreateBinary();
    /* Enable callbacks for PINT */
    PINT_EnableCallback(PINT);
    for (;;)
    {
        if (xSemaphoreTake(xSWSemaphore, portMAX_DELAY) == pdTRUE)
        {
            PRINTF("CPU woken up by external interrupt\r\n");
        }
    }
}
```

无滴答任务会打印一次当前的滴答节拍值，然后调用 `vTaskDelay()`函数以延时 5 秒。此时会调用空闲任务，并在空闲任务中调用 `vPortSuppressTicksAndSleep()`，让 RTC 启动一个 5s 的定时器，并调用 WFI 指令将 MCU 置于睡眠模式。

当 RTC 计数器减至 0 或用户按下 SW3 按钮时，MCU 会被唤醒并退出睡眠模式。用户必须根据 RTC 的状态和计数器的值计算补偿时间，然后将计算出的 `ulCompleteTickPeriods` 值补偿到 xTickCount。如果无滴答任务的 5s 延时已经执行完毕，则打印一次 xTickCount，然后继续进入低功耗模式，等待下一次唤醒。

4 测试

4.1 运行 freertos_tickless 示例

编译 SDK 包中的 freertos_tickless 工程，并将程序下载到 LPC55S16-EVK 板上，测量 MCU_VBAT 引脚上的电流消耗，如图 6 所示。

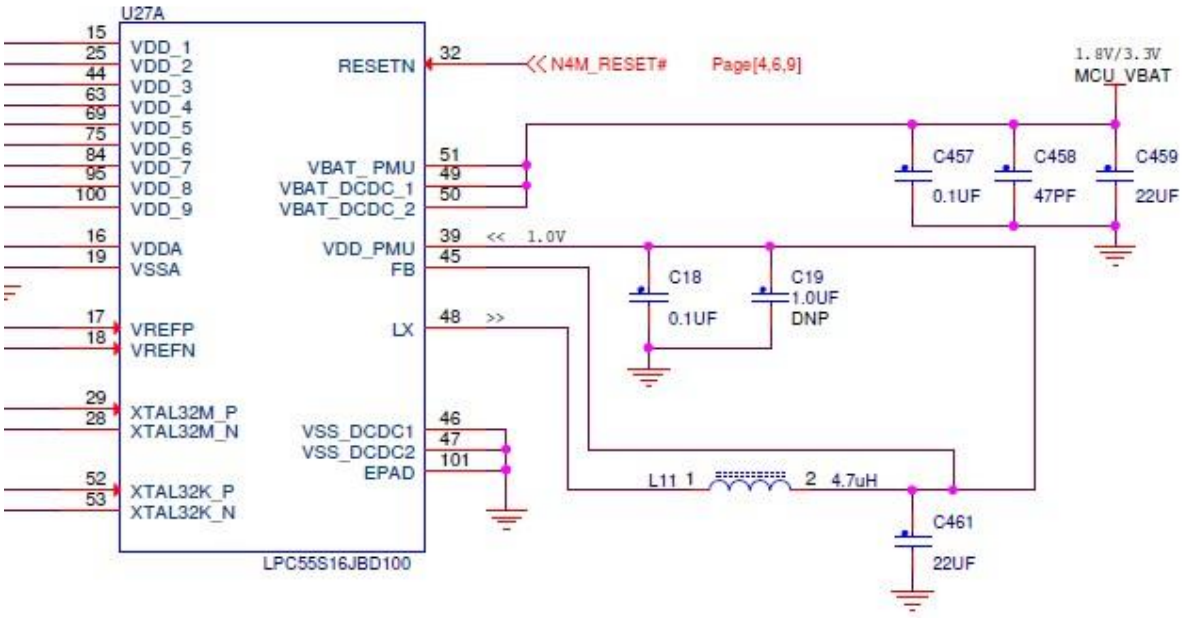

图 5. LPC55S16-EVK 原理图

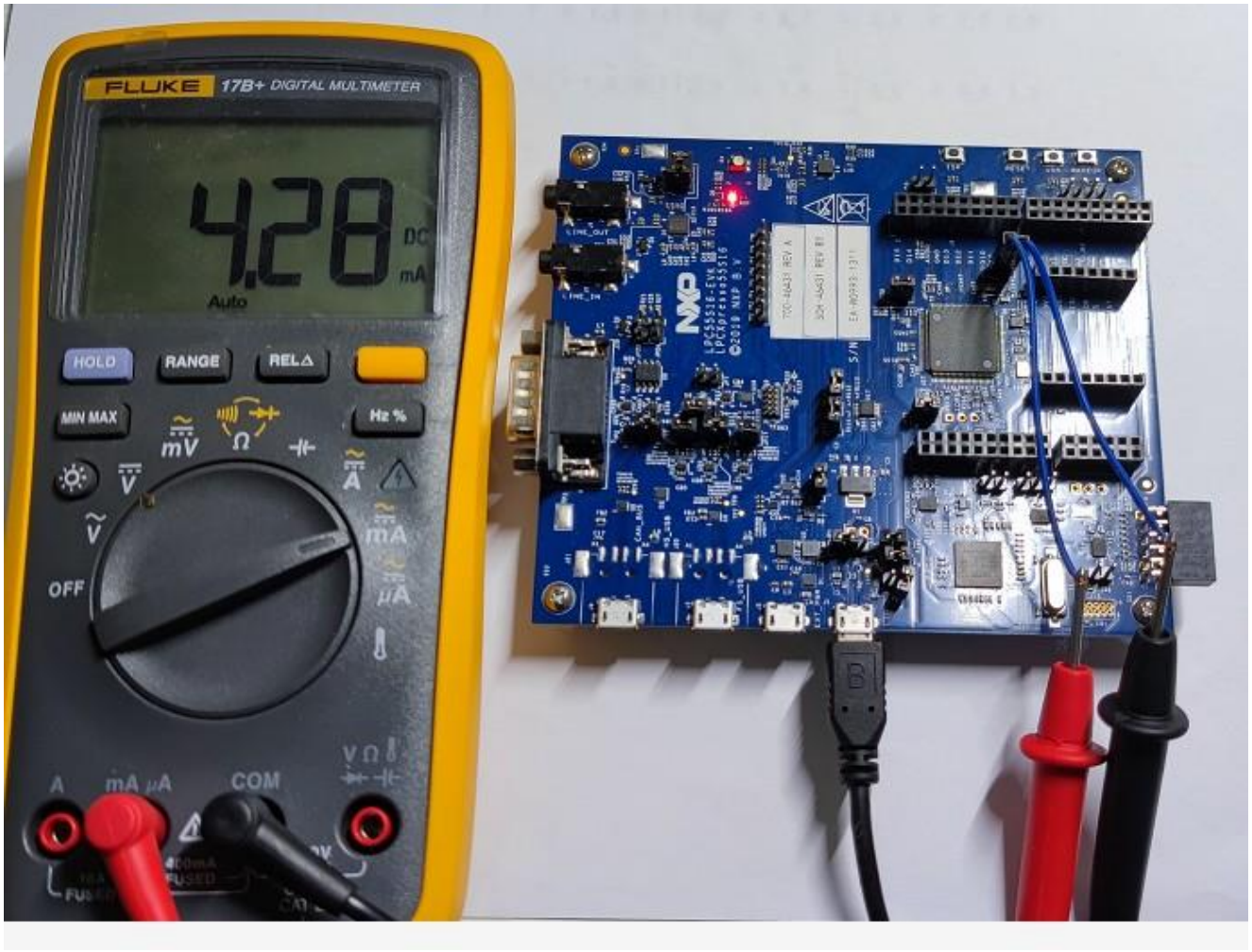

图 6. 当启用无滴答模式时，MCU_VBAT 引脚的电流消耗

当 LPC55S16 进入低功耗模式时，电流消耗为 4.28 mA。

将 `configUSE_TICKLESS_IDLE` 宏的值修改为 0，即禁用无滴答模式，重新编译工程并将程序下载到 LPC55S16-EVK 板上，之后测试 `MCU_VBAT` 引脚上的电流，如图 7 所示。

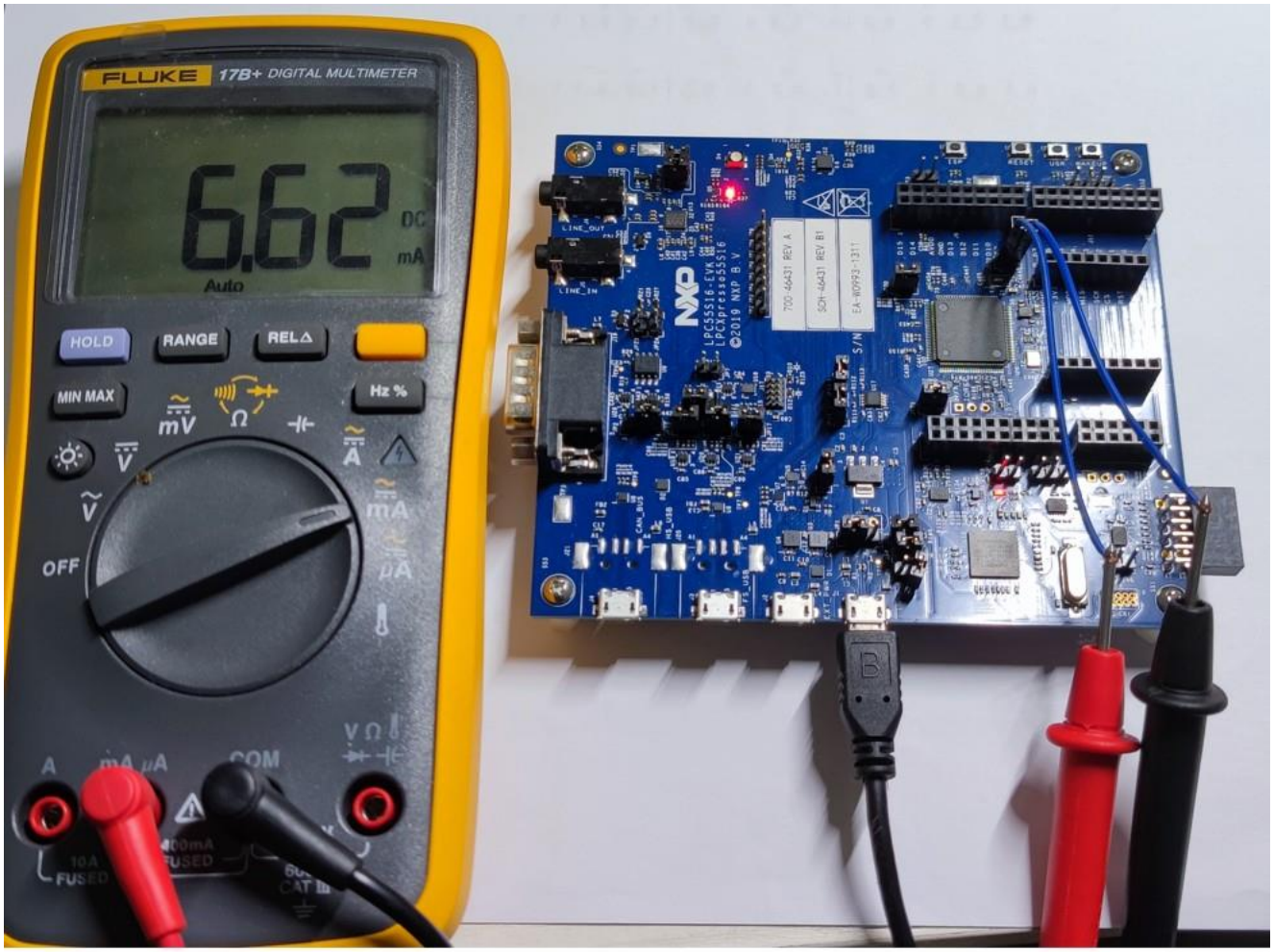

图 7. 禁用无滴答模式时 MCU_VBAT 引脚的电流消耗

从图 6 和图 7 可以看出，在启用无滴答模式后，`MCU_VBAT` 引脚上的电流消耗降低了 2.34 mA。

4.2 减少系统功耗的更多方法

如果用户想要更低的电流消耗，可以在 MCU 空闲时使 MCU 进入深度睡眠模式或断电模式。用户还可以使用 `configPRE_SLEEP_PROCESSING` 和 `configPOST_SLEEP_PROCESSING` 宏相关函数来执行更多的降低功耗的操作，例如关闭外设时钟和降低系统频率等。

在 LPC55S16 SDK 的 `freertos_tickless` 示例中，并没有为 `configPRE_SLEEP_PROCESSING` 和 `configPOST_SLEEP_PROCESSING` 宏所定义的函数。本节将介绍如何实现更低的电流消耗。

定义 `LOW_POWER_MODE` 宏以表示不同的低功耗模式：

LOW_POWER_MODE = 1 : 睡眠模式

LOW_POWER_MODE = 2 : 深度睡眠模式

LOW_POWER_MODE = 3 : 断电模式

在 LPC55S16-EVK 板上测得的这些模式的电流消耗如表 1 所示。

表 1. 不同模式下的电流消耗

低功耗模式	电流
普通模式	6.62 mA
睡眠/启用 USART	4.28 mA
睡眠/禁用 USART	4.15 mA
深度睡眠模式	55.8 uA
断电模式	3.6 uA

下面介绍实现这些低功耗模式的方法。

4.2.1 睡眠模式

如第 2 章所述，`configUSE_TICKLESS_IDLE` 宏可以设置为 1 或 2 以启用无滴答模式，从而使 MCU 进入睡眠模式。但是，在睡眠模式下，只有内核是停止运行的，其他外设可能仍在工作，因此用户也可以使用 `configPRE_SLEEP_PROCESSING` 和 `configPOST_SLEEP_PROCESSING` 宏在进入睡眠模式之前关闭一些外设时钟，例如 USART 外设。

实现方法如下：

1. 定义与 `configPRE_SLEEP_PROCESSING` 和 `configPOST_SLEEP_PROCESSING` 宏相对应的函数以禁用/启用 USART 外设。

```
#ifndef configPRE_SLEEP_PROCESSING
    #define configPRE_SLEEP_PROCESSING( x )   vPortConfigPreSleepProcessing( x )
#endif

#ifndef configPOST_SLEEP_PROCESSING
    #define configPOST_SLEEP_PROCESSING( x )  vPortConfigPostSleepProcessing( x )
#endif

void vPortConfigPreSleepProcessing(TickType_t xExpectedIdleTime)
{
#if LOW_POWER_MODE == 1
    /* Disable USART0 peripheral */
    DbgConsole_Deinit();
    CLOCK_DisableClock(kCLOCK_FlexComm0);
#endif
}

void vPortConfigPostSleepProcessing(TickType_t xExpectedIdleTime)
{
#if (LOW_POWER_MODE == 1) || (LOW_POWER_MODE == 3)
    /* Enable USART0 peripheral */
    BOARD_InitDebugConsole();
#endif
}
```

2. 调用与 `configPRE_SLEEP_PROCESSING` 和 `configPOST_SLEEP_PROCESSING` 宏相对应的函数，来启用/禁用 USART 外设。

```
xModifiableIdleTime = xExpectedIdleTime;
configPRE_SLEEP_PROCESSING(xModifiableIdleTime);
if (xModifiableIdleTime > 0)
```

```
{
#if LOW_POWER_MODE == 1
    _ DSB();
    _ WFI();
    _ ISB();
#elif LOW_POWER_MODE == 2
    POWER_EnterDeepSleep(APP_EXCLUDE_FROM_DEEPSLEEP, 0x7FFF, WAKEUP_RTC_LITE_ALARM_WAKEUP,
0x0);
#elif LOW_POWER_MODE == 3
    POWER_EnterPowerDown(APP_EXCLUDE_FROM_POWERDOWN, 0x7FFF, WAKEUP_RTC_LITE_ALARM_WAKEUP, 1);
#endif
}
configPOST_SLEEP_PROCESSING(xExpectedIdleTime);
ulLPTimerInterruptFired = false;
```

当 LOW_POWER_MODE 宏为 1 且 MCU 处于空闲状态时，MCU 会进入睡眠模式。在进入睡眠模式之前，USART 外设被禁用。在这种情况下，电流消耗为 4.15 mA，如图 8 所示。禁用 USART0 外设后，电流会降低 130 uA。

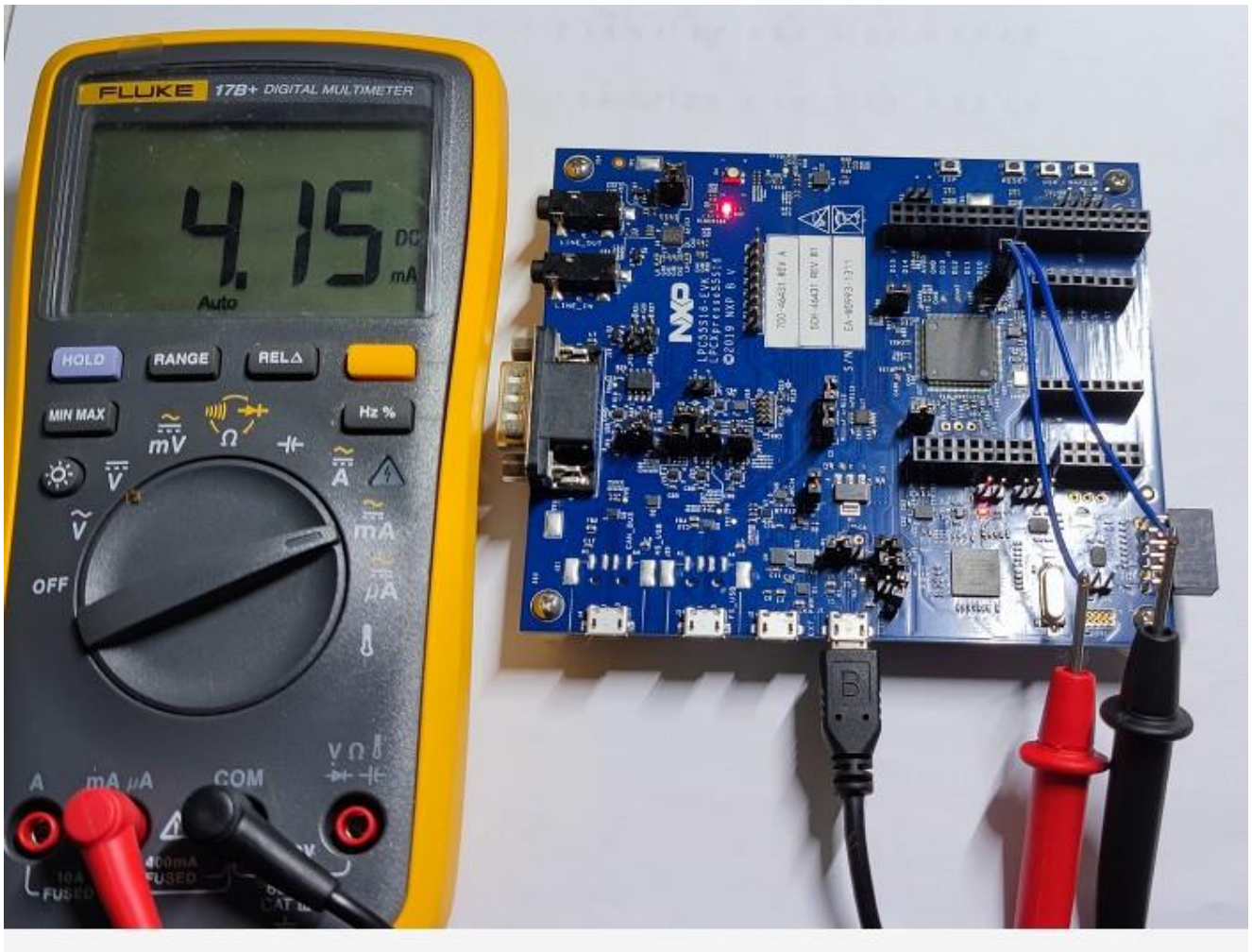

图 8. 禁用 USART0 外设后，MCU_BVAT 引脚的电流消耗

4.2.2 深度睡眠模式

除了让 LPC55S16 进入睡眠模式外，用户还可以使其在空闲状态时进入深度睡眠模式，如下图所示。

```
#define APP_EXCLUDE_FROM_DEEPSLEEP (kPDRUNCFG_PD_XTAL32K)

POWER_EnterDeepSleep(APP_EXCLUDE_FROM_DEEPSLEEP, 0x7FFF,
                     WAKEUP_RTC_LITE_ALARM_WAKEUP, 0x0);
```

深度睡眠模式下的电流消耗为 55.8 uA，如图 9 所示。

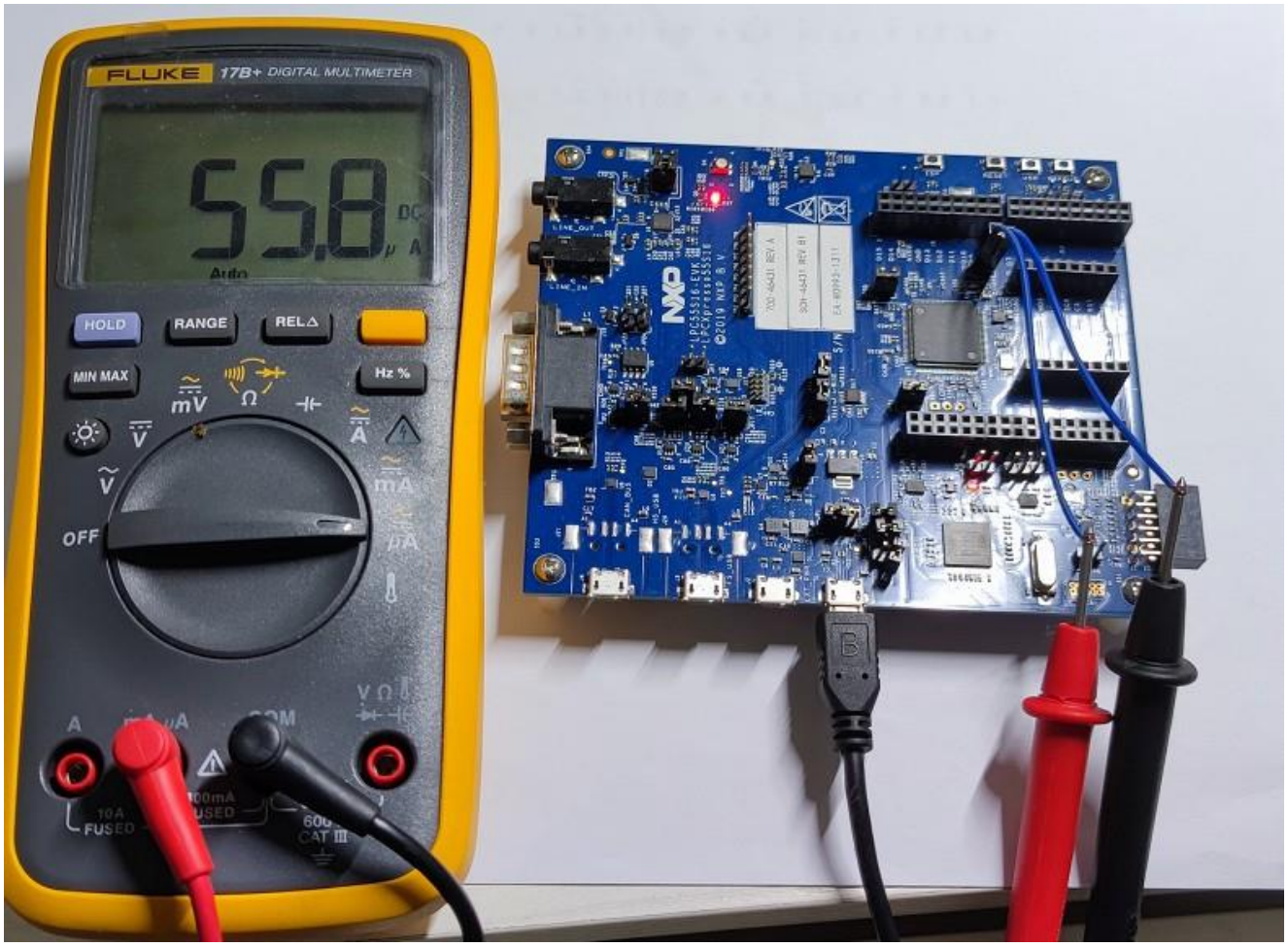

图 9. 深度睡眠模式的电流消耗

4.2.3 断电模式

用户还可以调用 `POWER_EnterPowerDown()` 函数使 LPC55S16 进入断电模式，如下所示。

```
#define APP_EXCLUDE_FROM_POWERDOWN (kPDRUNCFG_PD_XTAL32K)

POWER_EnterPowerDown(APP_EXCLUDE_FROM_POWERDOWN, 0x7FFF,
                     WAKEUP_RTC_LITE_ALARM_WAKEUP, 1);
```

断电模式下的电流消耗为 3.6 uA，如图 10 所示。

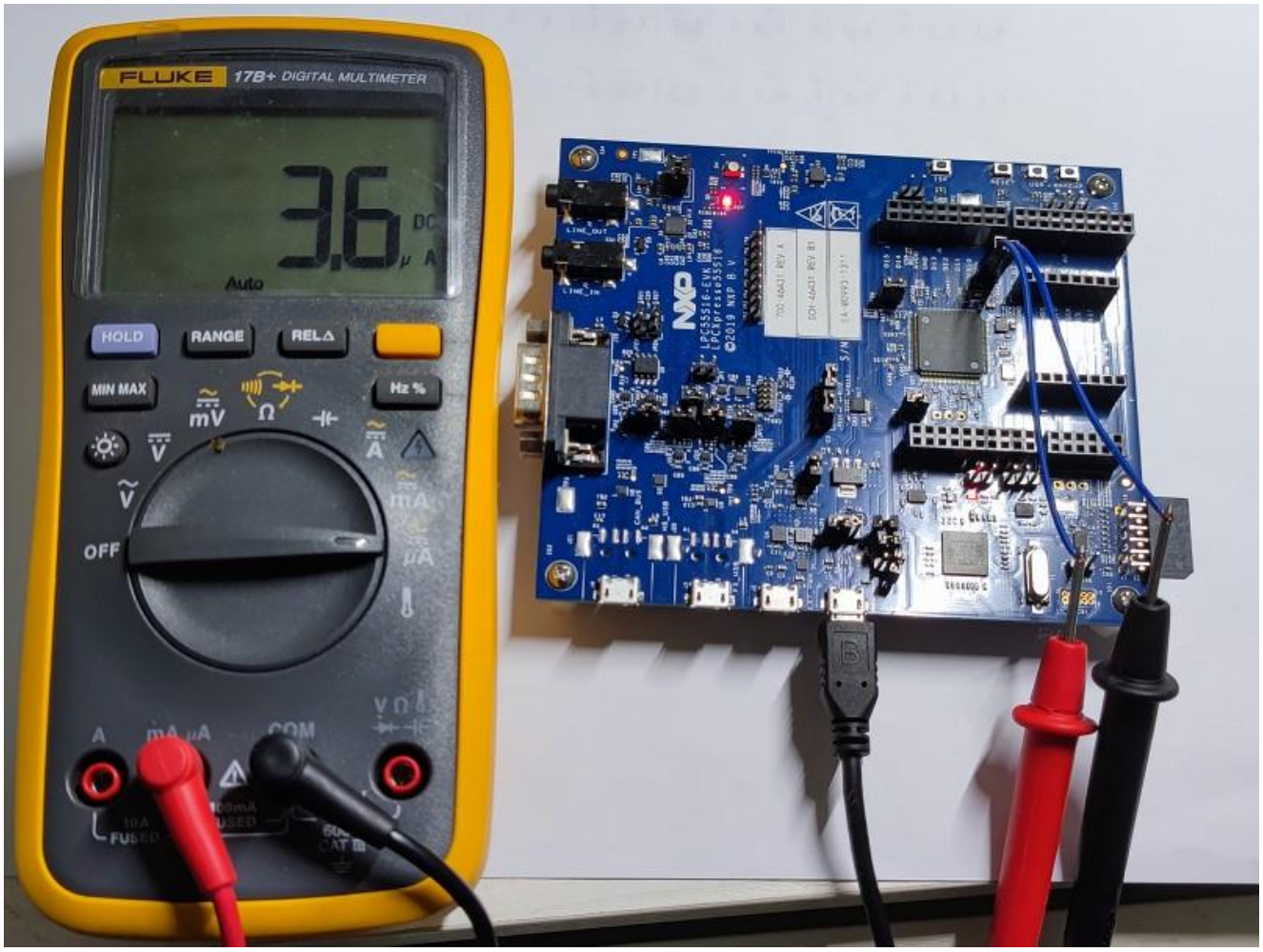

图 10. 断电模式的电流消耗

在 MCU 从断电模式被唤醒后，USART0（Flexcomm0）仍处于禁用状态，必须再次启用，如下所示。

```
void vPortConfigPostSleepProcessing(TickType_t xExpectedIdleTime)
{
#if (LOW_POWER_MODE == 1) || (LOW_POWER_MODE == 3)
    /* Enable USART0 peripheral */
    BOARD_InitDebugConsole();
#endif
}
```

第 4.2 节中描述的更改是在 `fsl_tickless_rtc.c` 文件中定义的，用户可以将这些更改复制到 `fsl_tickless_rtc.c` 文件的同一位置以实现这些函数。

5 结论

本应用笔记解释了 FreeRTOS 无滴答模式的实现原理，并介绍了在恩智浦 LPC5500 系列 MCU 上实现无滴答模式的细节。用户可以参考本应用笔记，在其应用中使用无滴答模式，以降低系统功耗。

6 参考资料

1). LPC55S1x/LPC551x User Manual

2). LPC55S3x UM.

7 修订历史

表 2. 修订历史

版本号	日期	实质性变更
第 0 版	2022 年 3 月 21 日	初版发布

How To Reach Us

Home Page:

nxp.com.cn

Web Support:

nxp.com.cn/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.cn/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.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX,EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE, VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org. M, M Mobileye and other Mobileye trademarks or logos appearing herein are trademarks of Mobileye Vision Technologies Ltd. in the United States, the EU and/or other jurisdictions.

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

Date of release: 21 March 2022
Document identifier: AN13593