AN4515
应用笔记

使用 STM32F411xx 批采集模式 (BAM) 实现电源效率最大化

前言

STM32F411 产品线属于 STM32 Dynamic Efficiency™ 微控制器的一部分。这些器件是高性能 F4 系列的入门级产品，具有动态功耗和处理性能的最佳平衡，并能够在运行、睡眠和停止模式下实现极低功耗。

STM32F411xx 微控制器具有新型的批采集模式 （BAM），为数据批处理进行了功耗优化，将动态效率提升到了一个新的水平。

本应用笔记通过一个用例，指导您在 STM32F411xx 微控制器上实现预期的功耗。还提供了关于如何实现 BAM 的示例。

本应用笔记中采用 X-CUBE-BAM 固件包。

目录

表格索引

图片索引

1 应用概述

本章给出了应用笔记用例概览，详细说明了用户如何实现 BAM，并描述了用例工作中涉及的每个低功耗模式。

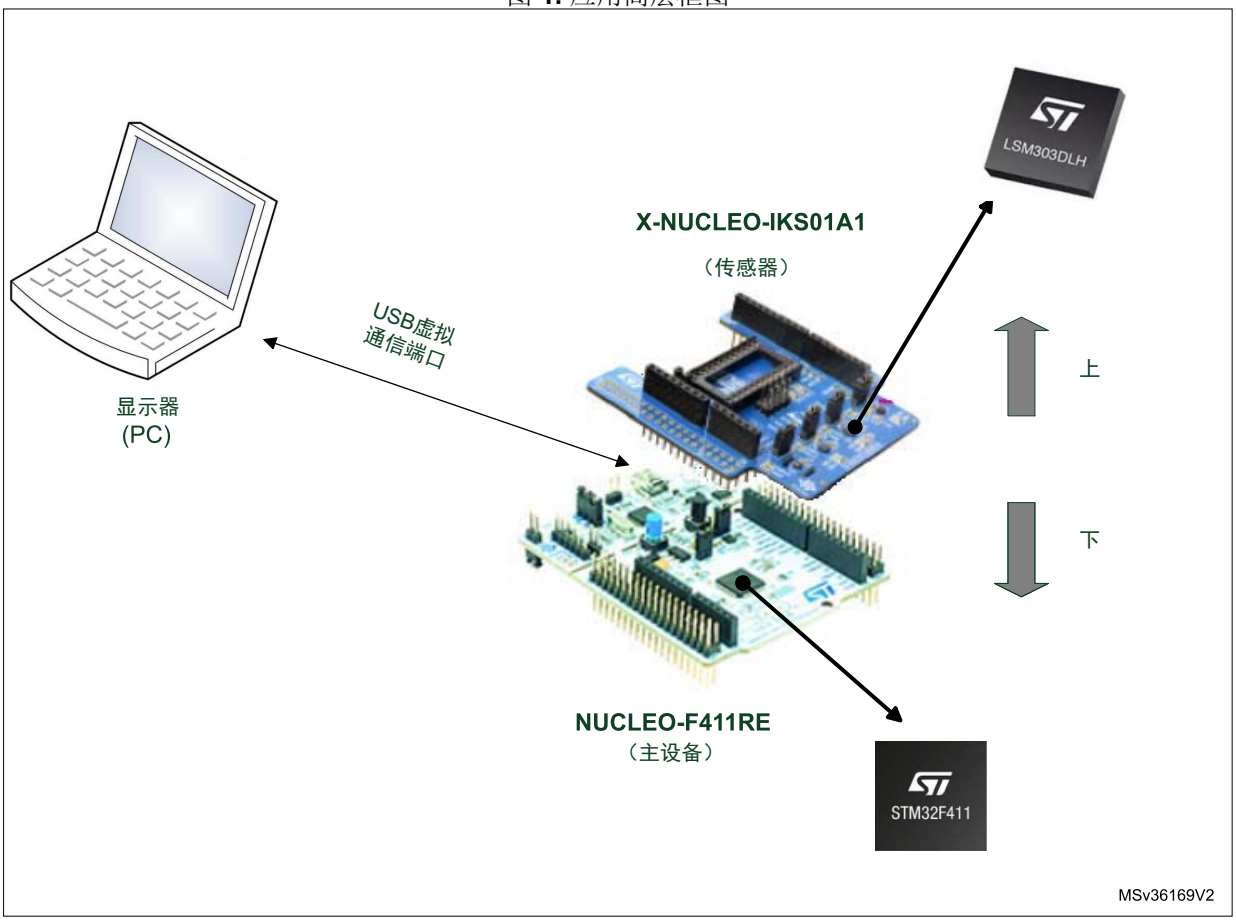
1.1 硬件高层描述

本文档首先说明如何评价不同低功耗模式 （待机、睡眠、停止和低功耗运行）中的功耗，然后重点说明通过使用 BAM 将传感器 （LSM6DS0 MEMS）数据流向 （UP 或 DOWN）传输至主 STM32F411xx 微控制器，来实现明显的功率降低。

传感器发送的数据通过 I2C 接口接收，利用 DMA 传输至 RAM。经过数据处理后， DMA 将数据流向传到 UART。然后通过 USB 虚拟串口显示出来。

1 显示了应用的高层框图。

图 **1.** 应用高层框图



□□ *X-CUBE-BAM□□□□□□□□LSM303DLH MEMS□32F401DISCOVERY□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□/NUCLEO-F411RE/LSM303DLHC□□□□*

1.2 低功耗模式

本章描述了应用中所使用的不同低功耗模式。

- 低功耗运行模式
  CPU 和一些外设处于运行中。为了进一步降低功耗，不用的 GPIO 被配置为模拟引脚并禁用外设。
- 睡眠模式
  这种模式下，仅有 CPU 是停止的，外设保持运行。为了降低功耗，可在进入睡眠模式前关闭闪存。发生中断时，外设能够唤醒 CPU。
- 停止模式
  CPU 处于深度睡眠模式 （DeepSleep mode）。除了 RTC，所有外设都禁用。通过发出中断来退出停止模式。
- 待机模式
  这种模式下，功率仅用于维持 RTC 寄存器。器件被 WKUP 脚上的上升沿 （产生系统复位）唤醒。

注： 更多关于低功耗模式的详细信息，请参考 STM32F411xx 数据手册。

1.3 批采集模式 （**BAM**）

1.3.1 原理

批采集模式 （BAM）可优化数据批处理的功耗。

它可通过通信外设交换大量数据，同时器件的其它部分 （包括 CPU）可处于省电模式：

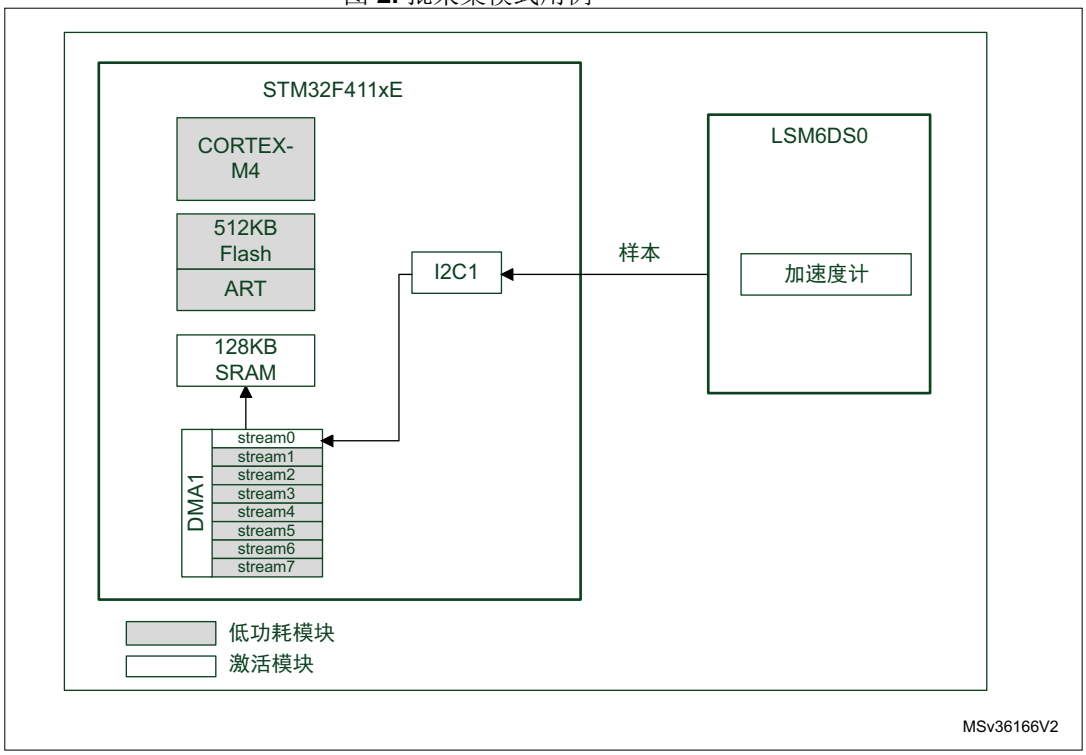
- 只使能所需要的 DMA 数据流，并将来自通信接口的数据传输到内部 RAM。
- 从 RAM 执行程序允许关闭闪存和停止 Flash 接口时钟。
- MCU 内核处于睡眠模式，等待中断 / 事件将其唤醒。

1.3.2 **BAM** 用例

□ *2* 显示了应用笔记用例高层框图的一部分。它描述了 BAM 涉及的不同外设及其功率状态（低功耗或激活的）。

事实上，数据接收过程中，只有 DMA stream0、 I2C1 和 SRAM 是激活的。 MCU 内核处于睡眠模式且闪存停止，直至完成 DMA 传输。

图 **2.** 批采集模式用例



1.3.3 如何实现 **BAM**

当 BAM 使能时，通过设置 PWR_CR 寄存器的 FMSSR 和 FISR 位来关闭闪存。从闪存执行程序时不能设置这些位。这可以通过从RAM执行一个特殊程序来实现（具体描述请参见 □ *3*，□ *4* 和 □ *5*）。

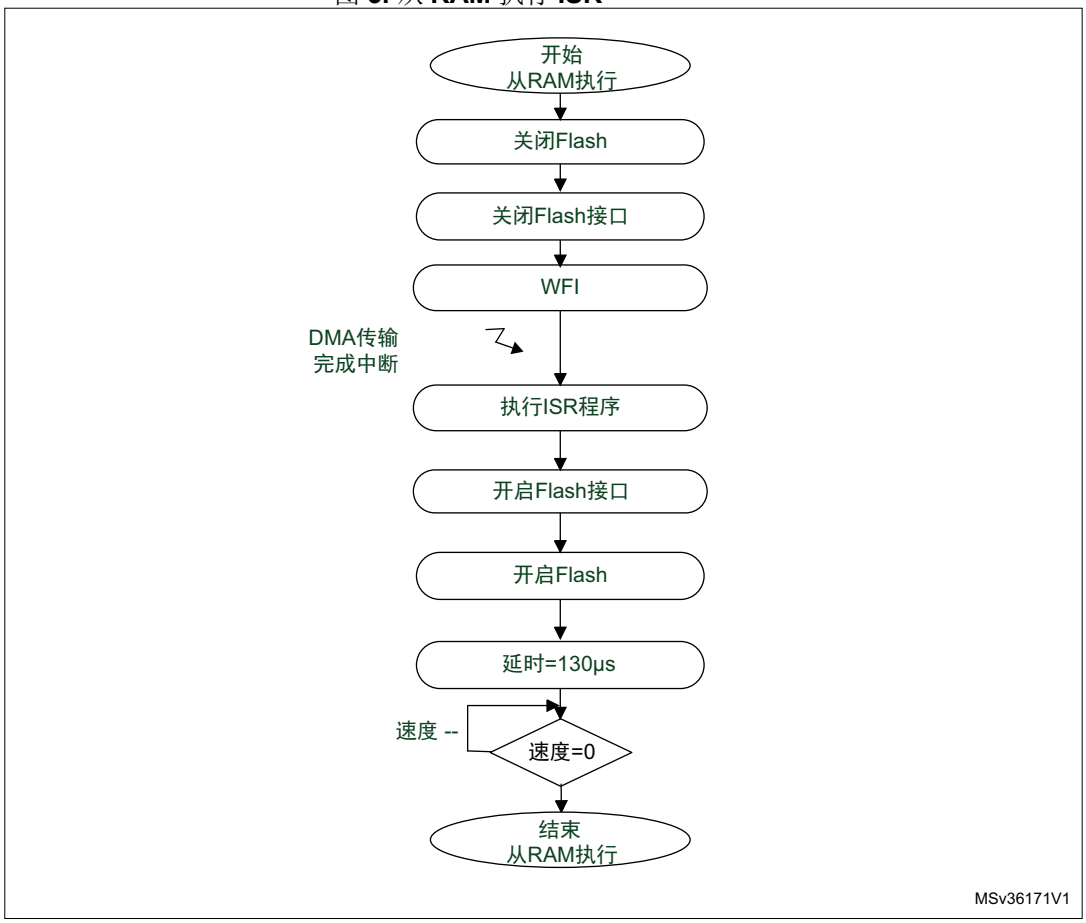
可采用 3 种方法来实现 BAM。每种方法都取决于用户应用：

- 从 RAM 执行中断
- 从闪存执行中断
- 利用事件来唤醒 CPU

从 **RAM** 执行中断

应用笔记用例中采用此方法 （参见 □ *3*）。中断将器件从睡眠模式中唤醒。因此用户应用必须将所需 ISR 和向量表存储在 RAM 中，以便在中断发生时能够立即从 RAM 中执行。

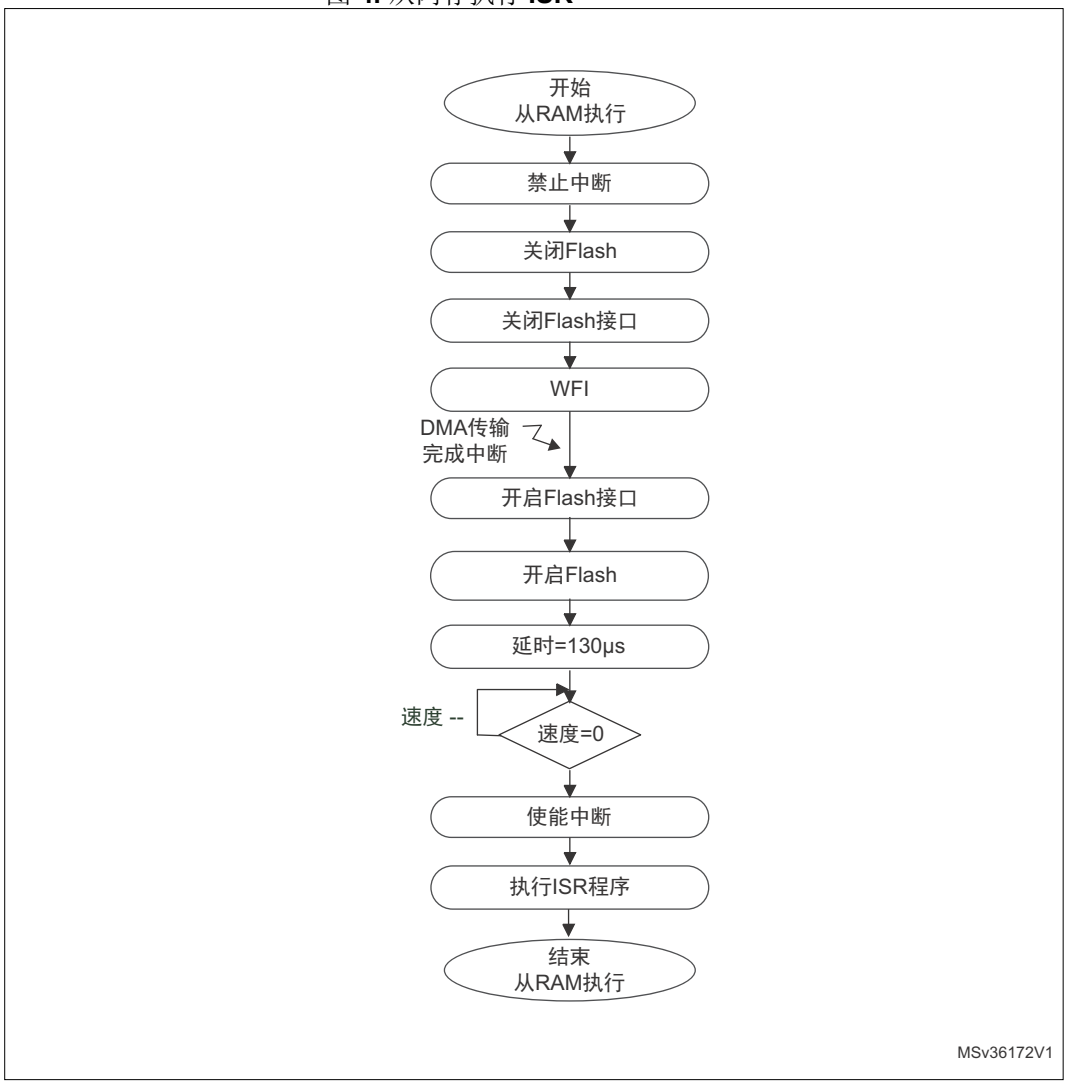
图 **3.** 从 **RAM** 执行 **ISR**



从闪存执行中断

这种方法里，闪存停止，必须禁用所有中断，直至闪存再次使能 （参见 *4*）。

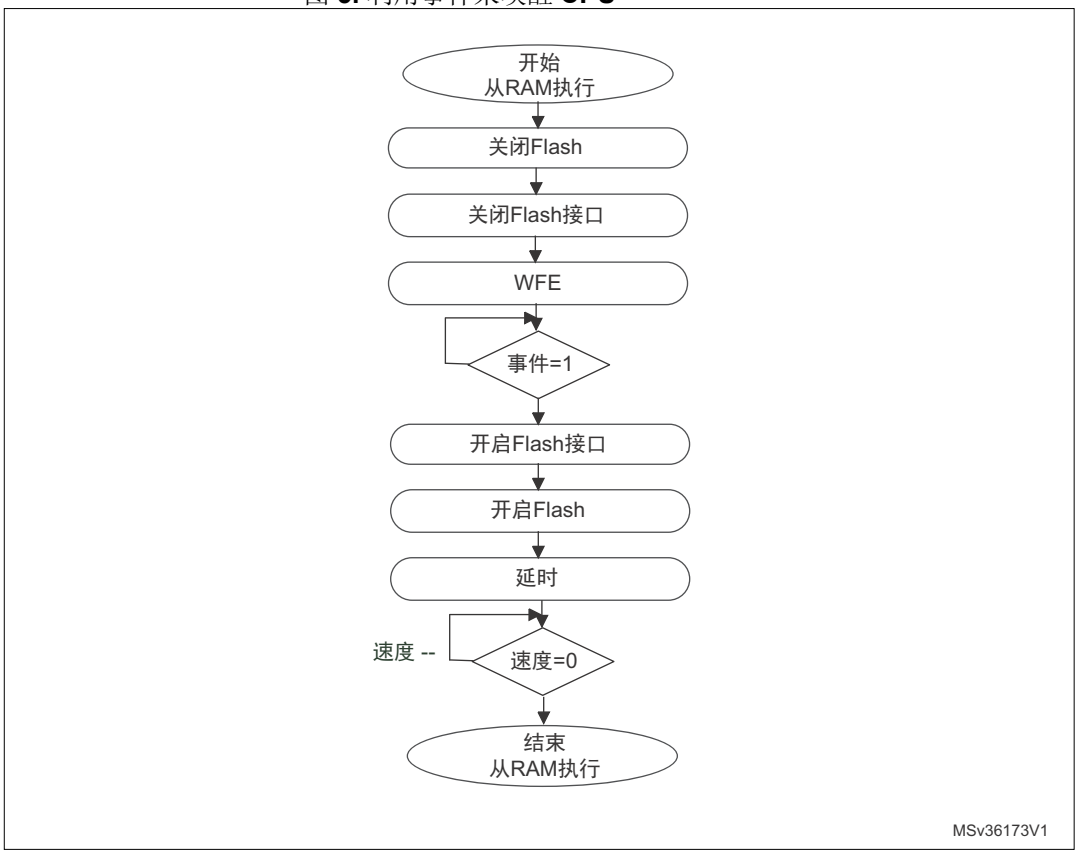
图 **4.** 从闪存执行 **ISR**



利用事件唤醒 CPU

此方法利用事件而不是中断将 CPU 从睡眠模式中唤醒 （参见 □ *5*）。

图 **5.** 利用事件来唤醒 **CPU**



BAM 实现方法总结

表 **1.** 方法总结

方法	优势	劣势
从 **RAM** 执行 **ISR**	中断执行中无延迟	固件实现略复杂
从闪存执行 **ISR**	简单的固件实现	中断执行中有延迟
利用事件来唤醒 **CPU**		由于闪存停止，因此不执行中断

1.3.4 如何利用 Keil® MDK-ARM™ 工具链从 RAM 执行代码

本章对利用 Keil MDK-ARM 工具链从 RAM 执行部分应用代码所需的步骤进行了概述。

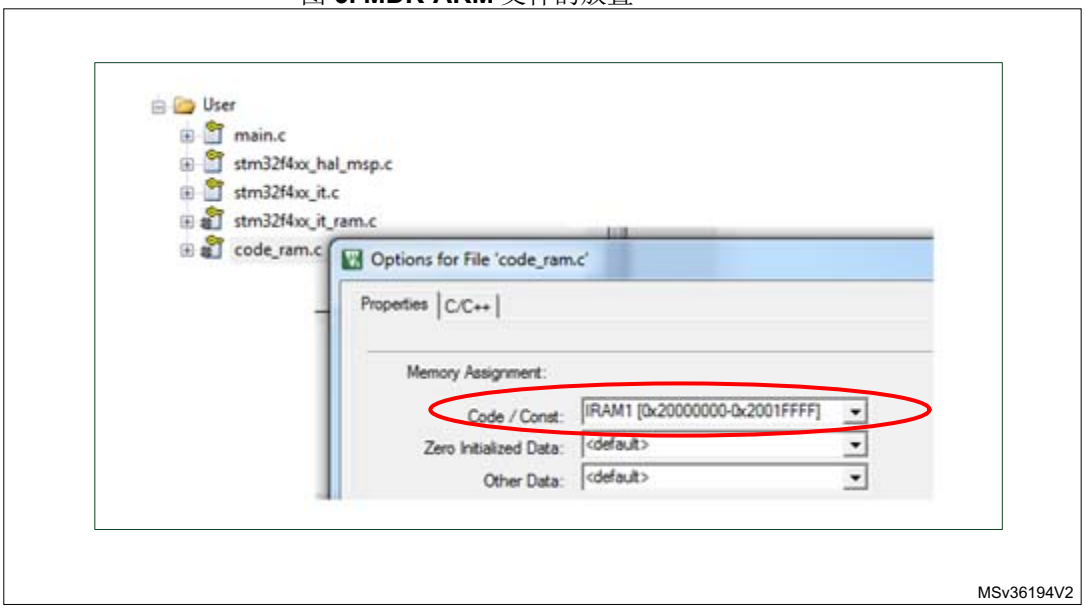
从 RAM 执行源文件

从 RAM 执行源文件意味着该文件所声明的全部函数均从 RAM 执行。

请按照以下步骤从 RAM 执行文件 （参见 □ 6）：

1. 右键点击文件，将其放入 RAM，然后选择选项。
2. 选择内存分配菜单中的 RAM 区域。

图 6. MDK-ARM 文件的放置



从 RAM 执行中断

从 RAM 执行中断处理程序需要执行下述步骤：

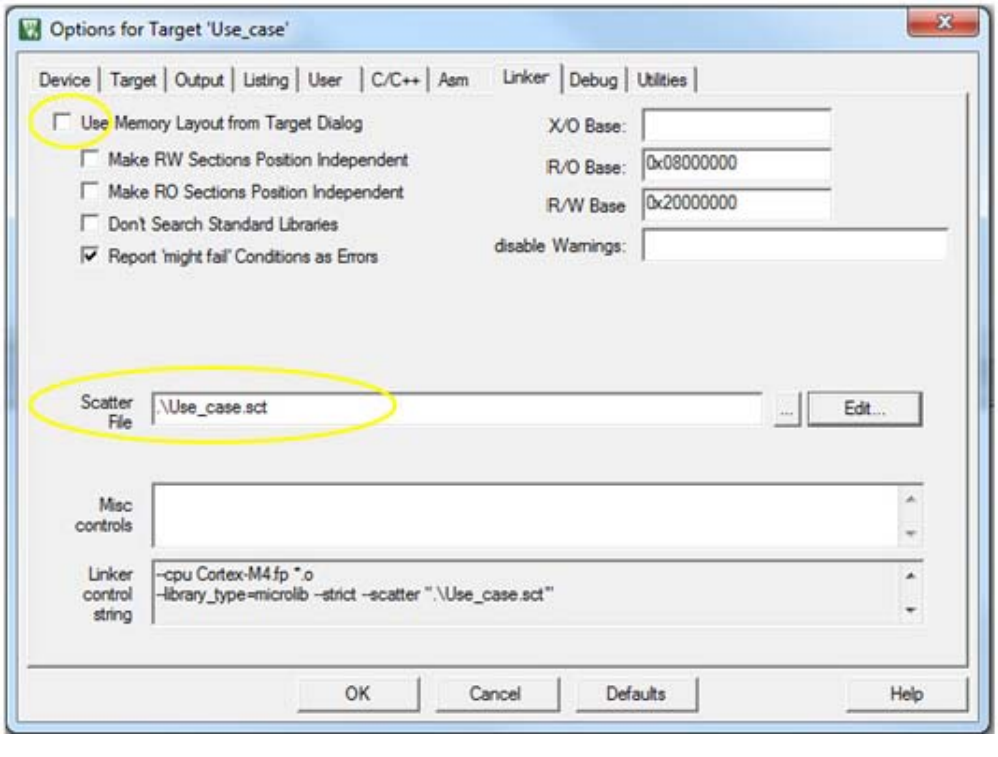
1. 建立第二个向量表，并将其保存为一个新文件 （startup_stm32f411xe_ram.s），该文件将存储在 RAM 中。
2. 将要从 RAM 执行的中断处理程序放入一个新的文件 （文件名为 stm32f4xx_it_ram.c）中。
3. 在分散文件中，将所需文件放入正确的 RAM 区 （参见 □ 7）。

图 **7. MDK-ARM** 分散文件

```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

LR_IROM1 0x08000000 0x00080000  {    ; load region size_region
  ER_IROM1 0x08000000 0x00080000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x20000000 0x00020000  {  ; RW data
  *.o (RESET_ram, +First)
    startup_stm32f411xe_ram.o (+RO)
    code_ram.o (+RO +RW)
    stm32f4xx_it_ram.o (+RO +RW)
   .ANY (+RW +ZI)
  }
}
```

4. 关于项目选项，请参考已修改的分散文件 （参见 □ *8*）：

图 **8. MDK-ARM** 选项菜单



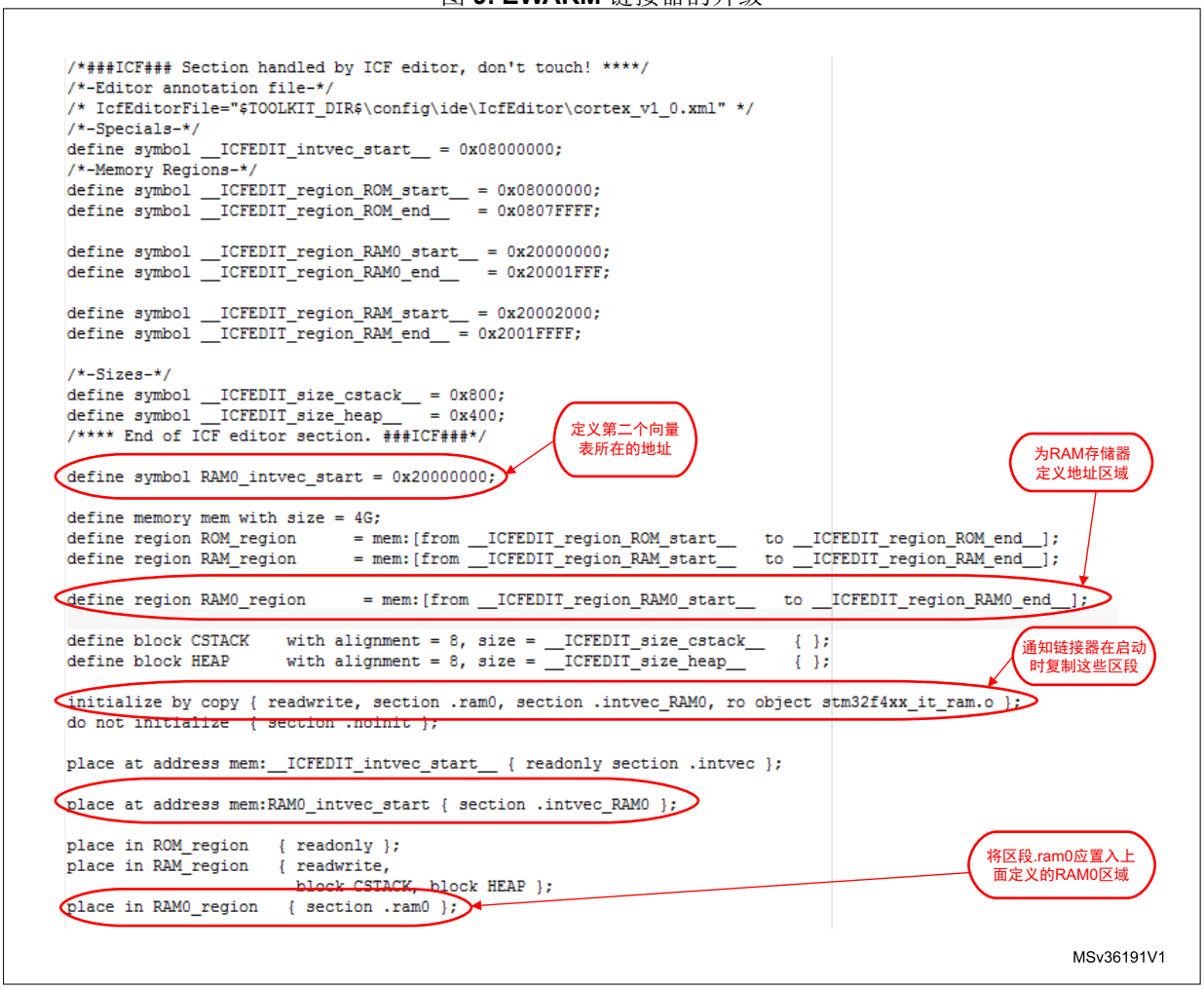
1.3.5 如何利用 IAR-EWARM 工具链从 RAM 执行代码

从 RAM 执行代码

该操作需要一个新的 RAM0 扇区，在链接器文件 （.icf）中定义 （参见 □ 9）。以下是所需的步骤：

1. 通过设定起始地址和结束地址，定义 RAM0 的地址区。
2. 通知链接器在启动时将名为 “ram0” 的区段从闪存复制到 RAM0。
3. 向链接器说明，代码区段 ram0 应置入 RAM0 区域。

图 **9. EWARM** 链接器的升级



从 RAM 执行源文件

如需执行来自 RAM 的源文件，请使用文件选项窗口：

1. 在出现的菜单上选择选项。
2. 选择 **C/C++** 编译器
3. 勾选窗口中的覆盖继承设定。
4. 选择输出选项卡，然后在代码段名称字段中输入链接器文件中已设定的区段名称 （本例中为 **.ram0**）。

从 **RAM** 执行中断

从 RAM0 执行中断处理程序的步骤如下所述：

1. 更新链接器文件 (.icf) （参见 □ *9*）。
   a) 定义第二个向量表所在的地址：0x2000 0000。
   b) 通知链接器在启动时复制名为 .intvec_RAM0 的扇区。
2. 更新启动文件 （参见 □ *10*）。
3. 将要在 RAM0 中执行的中断处理程序放到新的 **stm32F4xx_it_ram.c** 中，如 □□ □□ *RAM*□□□□□ 中所述。
4. 将向量表重新映射至 RAM0。要实现它，将 SystemInit 函数中的 VTOR 寄存器如下修改：
   SCB->VTOR = 0x2000 0000 | VECT_TAB_OFFSET

图 **10.** 更新 **EWARM** 启动文件来处理中断

```
        MODULE  ?cstartup

        ;; Forward declaration of sections.
        SECTION CSTACK:DATA:NOROOT(3)

        SECTION .intvec:CODE:NOROOT(2)

        EXTERN  __iar_program_start
        EXTERN  SystemInit
        PUBLIC  __vector_table

        DATA

__vector_table
        DCD     sfe(CSTACK)
        DCD     Reset_Handler             ; Reset Handler

        SECTION .intvec_RAM0:CODE:ROOT(2)

        PUBLIC  __vector_table_RAM0

        DATA

__vector_table_RAM0
        DCD     sfe(CSTACK)
        DCD     Reset_Handler             ; Reset Handler

        DCD     NMI_Handler               ; NMI Handler
        DCD     HardFault_Handler         ; Hard Fault Handler
        DCD     MemManage_Handler         ; MPU Fault Handler
        DCD     BusFault_Handler          ; Bus Fault Handler
        DCD     UsageFault_Handler        ; Usage Fault Handler
        DCD     0                         ; Reserved
        DCD     0                         ; Reserved
```

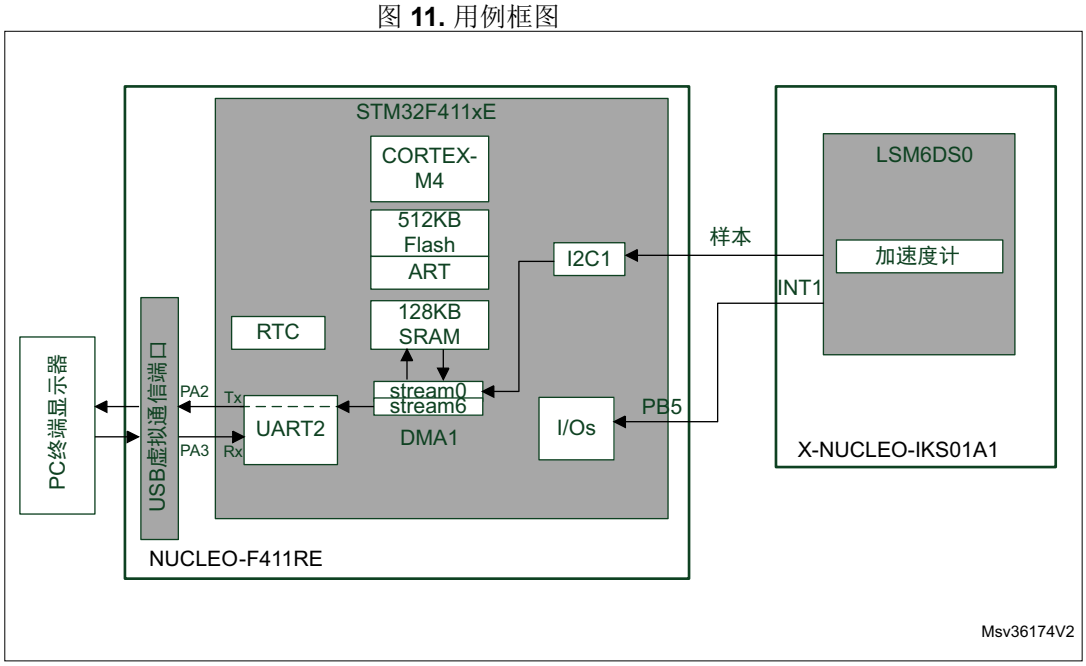
2 应用笔记用例

本章通过框图和状态机描述了此应用笔记中开发的用例，重点说明了不同外设及其所用的低功耗模式。

2.1 框图

11 描述了用例涉及的不同模块。

图 **11.** 用例框图



2.2 STM32F411xx 所用的外设

应用用例使用了下面的 STM32F411xx 外设：

- 时钟

  此应用中使用了 2 个时钟：
  - HSI：16 MHz 系统时钟源
  - LSE：32.768 kHz 低速外部晶振，驱动 RTC 时钟
- **SysTick** 定时器

  此定时器用于等待循环或用来产生超时 （延迟）。
- **GPIO**
  - PA0：用作用户按钮和唤醒引脚 （从待机模式唤醒）
  - PA3：设置为引脚，其 USART2_RX 输入上的中断 （可将器件）从停止模式唤醒
  - PB5：设置为输入，其中断功能连接至 X-NUCLEO-IKS01A1 上的 MEMS 中断引脚。该引脚用来将 CPU 从睡眠模式中唤醒。
  - PB8 （I2C1_SCL）和 PB9 （I2C1_SDA）连接到 MEMS 的 PB6 和 PB9
  - 当微控制器处于低功耗模式时，不用的 I/O 被置于模拟输入模式，以减少功耗。
- **DMA1**

  当 CPU 处于睡眠模式时， DMA1 用来传输数据。
  - Stream0：使能，用来接收来自 I2C1 的数据
  - Stream6：使能，用来将数据传输至 USART2
  - 不用的数据流被禁用
- **RTC**

  RTC 是一个独立的定时器 / 计数器，提供了具有亚秒、秒和分的日历，能够生成时间戳事件。它使用 LSE 时钟。
- **I2C1**

  I2C1 接口从传感器接收数据，速度可达到 400 KHz。
- **USART2**

  USART2 用来向 USB 虚拟串口传输信息 （通过 DMA）。

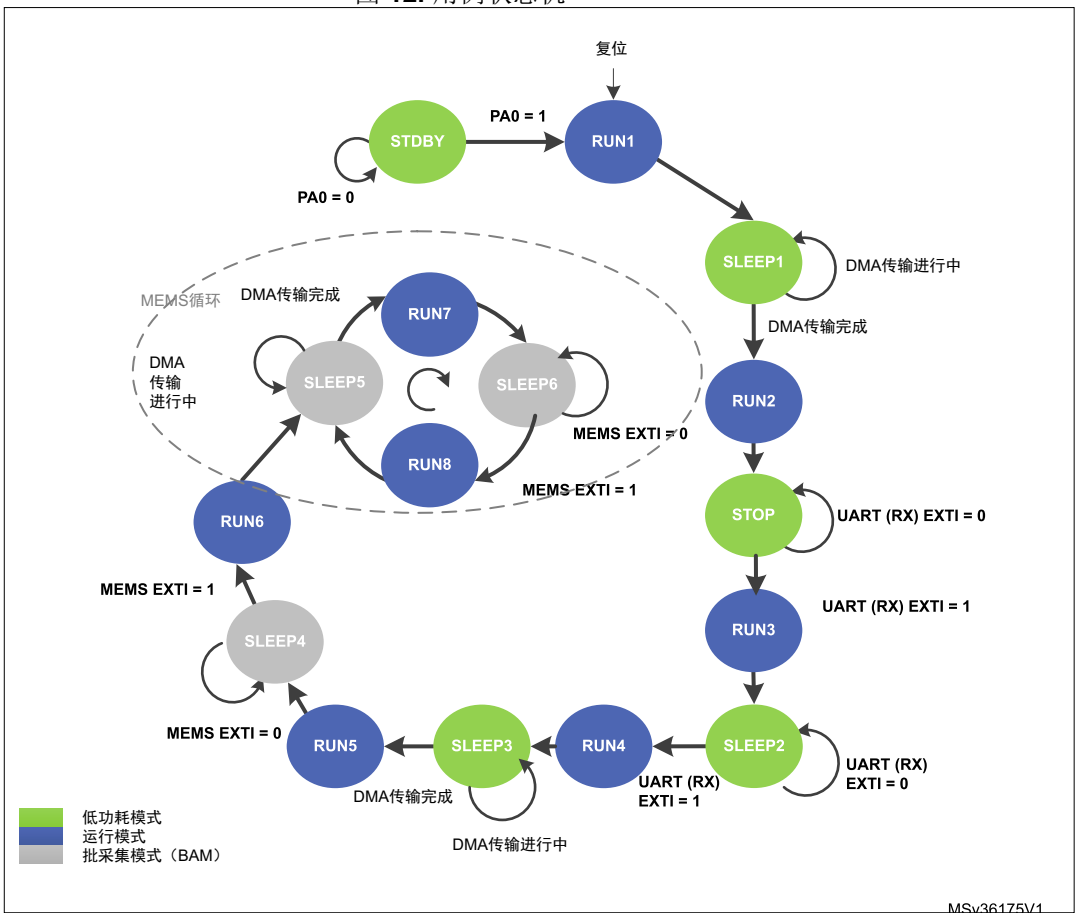
  USART2 配置如下：
  - 字长度 = 8 位
  - 停止位 = 一个停止位
  - 奇偶 = 无奇偶
  - 波特率 = 9600 波特
  - 硬件流控制禁用 （RTS 和 CTS 信号）
- **RAM**

  当闪存关闭时，从 RAM 执行一个特殊程序。

2.3 功能描述

此用例的原理如 □ *12* 中状态机所概括。

图 **12.** 用例状态机



应用用例的主要步骤如下：

1. 主设备和传感器上电。默认情况下系统处于 **STDBY** 模式 （待机模式）。
2. 用户按下用户按钮 （在 NUCLEO-F411RE 板上），开始运行应用。 CPU 进入 **SLEEP1** 状态，同时 DMA 传输启动信息通过 USART2 和 USB 虚拟串口显示出来。
3. 当 DMA 传输完成时， CPU 从 **SLEEP1** 中唤醒并进入 **STOP**，等待用户读取下面的配置菜单 （参见 □ *13*）：

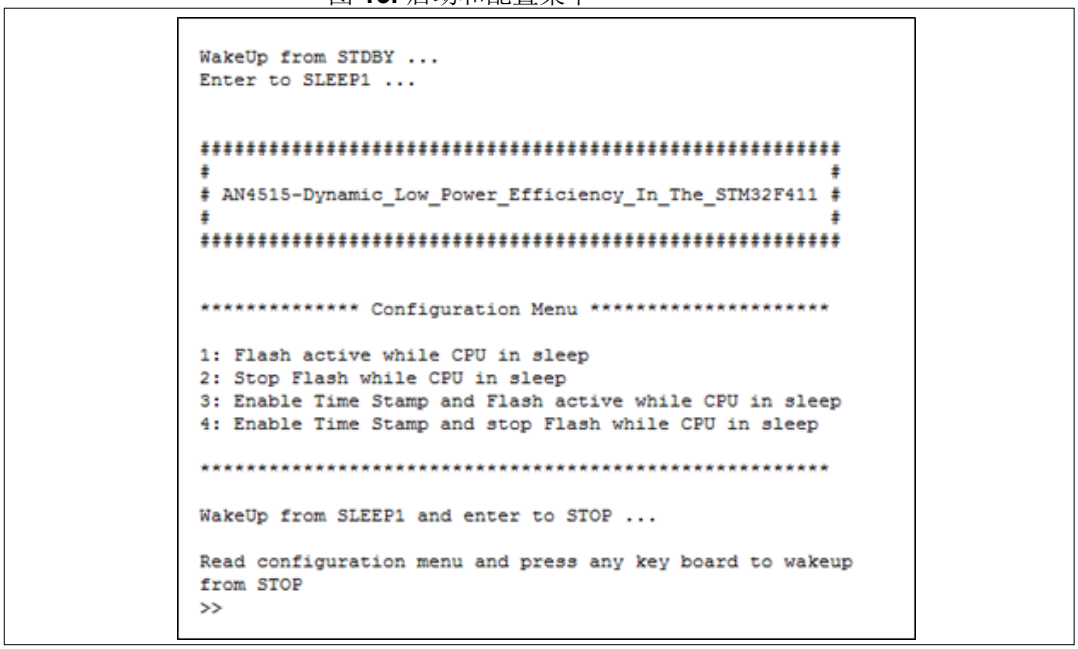
   1：**Flash** 激活同时 **CPU** 为睡眠模式：闪存运行

   2：停止 **Flash** 同时 **CPU** 为睡眠模式：此配置显示了 BAM 功能。

   3：使能时间戳， **Flash** 激活同时 **CPU** 为睡眠模式：此选项用来跟踪不同的应用状态功耗 （利用时间戳日志中显示的每个状态的周期）。

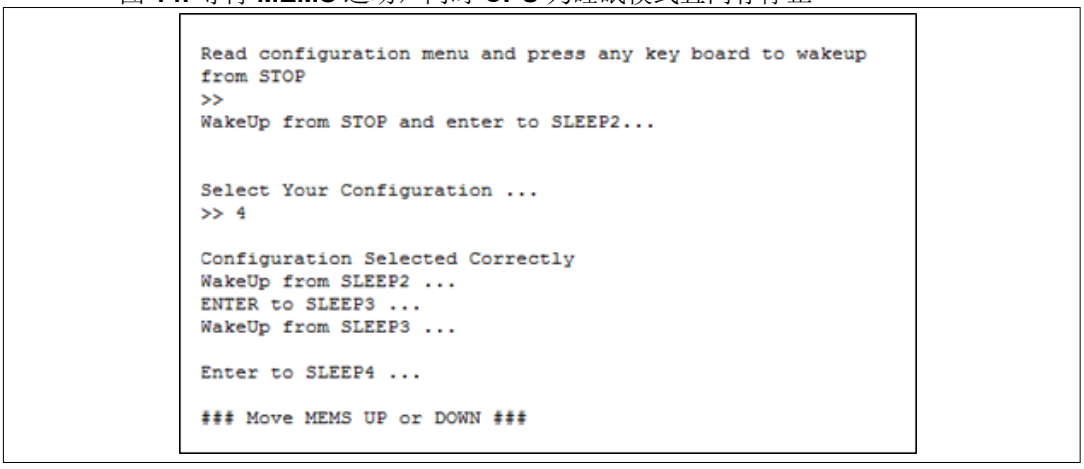
   4：使能时间戳，停止 **Flash** 同时 **CPU** 为睡眠模式：此选项允许利用第二和第三配置。

图 **13.** 启动和配置菜单



4. 系统从停止模式唤醒 并进入 **SLEEP2** 状态，等待用户选择所需的配置。
5. 一旦选择了配置， USART2_RX 上的外部中断将 CPU 从睡眠模式唤醒。
6. CPU 再次进入睡眠模式 （**SLEEP3** 状态），同时 DMA 传输信息通过 USART2 发送出去。
7. DMA 传输完成后，中断已经被接收， CPU 配置 MEMS 并进入 **SLEEP4** 状态，等待加速度计运动 （UP 或 DOWN 方向）。睡眠模式中 DMA 发送 ***### Move MEMS UP or DOWN ###*** 信息。

图 **14.** 等待 **MEMS** 运动，同时 **CPU** 为睡眠模式且闪存停止



8. MEMS 移动后， CPU 从睡眠模式唤醒，配置 I2C 接口和 DMA，传输来自 MEMS 的采样。然后 DMA 开始发送数据到 RAM，此时闪存停止， CPU 返回睡眠模式 （**SLEEP5** 状态）。

9. 一旦 DMA 数据传输完成，中断就唤醒 CPU，在其返回睡眠模式之前处理这些数据（**SLEEP6** 状态）。
10. 当 CPU 处于睡眠模式 （**SLEEP6** 状态）且闪存停止， DMA 发送一个完整日志到 USART2，包括 MEMS 方向和时间戳 （参见 *15*）。

图 **15.** 选择配置 **4** 时的日志示例

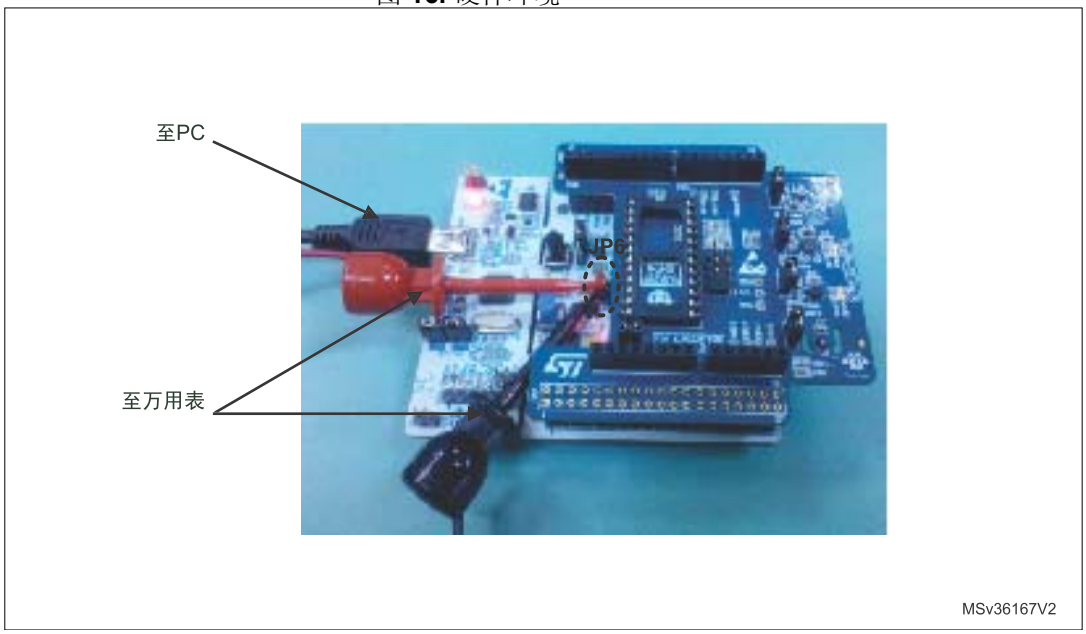
```
### Move MEMS UP or DOWN ###

************** Direction Display ***************
UP Direction

************** TimeStamp Display **************

00:00:00:000: Enter STDBY
00:00:02:145: WakeUp from STDBY
00:00:02:146: Enter to SLEEP1
00:00:02:904: WakeUp from SLEEP1
00:00:02:904: Enter to STOP
00:00:04:179: WakeUp from STOP
00:00:04:180: Enter to SLEEP2
00:00:04:982: WakeUp from SLEEP2
00:00:05:475: Enter to SLEEP3
00:00:05:552: WakeUp from SLEEP3
00:00:05:554: Enter to SLEEP4
00:00:09:078: WakeUp from SLEEP4
00:00:09:078: Enter to SLEEP5
00:00:09:079: Wakeup from SLEEP5
00:00:09:080: Enter to SLEEP6


## Move MEMS ##
```

3 应用电流消耗

3.1 硬件要求

运行应用和测量电流消耗前，将您的 PC 连接到 NUCLEO 板，如 □ *16* 所示。

图 **16.** 硬件环境



运行应用所需的硬件如下：

- Windows® PC
- NUCLEO-F411 和 X-NUCLEO-IKS01A1 板
- 利用一个微型USB数据线为NUCLEO板上电，并连接NUCLEO嵌入式ST-LINK进行调试和编程
- 使用万用表来测量电流消耗。

表 **2.** 主设备和传感器板连接

NUCLEO-F411RE	X-NUCLEO-IKS01A1
I2C SCL (PB8)	I2C SCL
I2C SDA (PB9)	I2C SDA
PB5	LSM6DS0_INT1
GND	GND

3.2 软件设置

使用一段代码 （利用 MDK-ARM 工具链 V5.14 和最优化 Level3(-O3) 编译）实现下面的测量。

3.3 测量的电流消耗

本章对下列条件下获得的电流消耗进行比较：

- 闪存停止且 CPU 为睡眠模式
- 闪存激活且 CPU 为睡眠模式。

□ *3* 描述了用例状态机中 MEMS 循环部分 （SLEEP5， RUN7， SLEEP6 和 RUN8）获得的平均测量值。

表 **3.** 电流消耗示例

配置菜单	激活的外设	电流消耗 **(uA)**
Flash 激活同时 CPU 处于睡眠	FLASH、 RTC、 USART2、 DMA1 （仅 stream0 和 stream 6 使能）、 I2C1	970
停止 Flash 同时 CPU 处于睡眠 （BAM）	RTC、 USART2、 DMA1 （仅 stream0 和 stream 6 使能）、 I2C1	710

□ *3* 显示明显降低了约 27 % 的电流消耗，利用新的 STM32F411xx □□*Flash*□□*CPU* □□□□ 功能 （BAM）而得到。

4 结论

此应用笔记给出了新型 RAM 实现 （由 STM32F411 产品线所提供）的用户指南，补充了数据手册和参考手册。

用例重点说明，通过使用 BAM （停止 flash 同时 CPU 为睡眠模式），可使电流消耗明显降低。它还允许评估不同的 STM32F411xx 低功耗模式，并且由于具有时间戳日志，因此能够确定所有的状态周期，计算应用平均功耗。

5 修订历史

表 **4.** 文档修订历史

日期	版本	变更
2014 年 12 月 19 日	1	初始版本。
2015 年 6 月 8 日	2	移除了表 1 可用产品和软件，因为文档描述了固件（STSW-STM32154）。 整个文档中，用 LSM6DS0 MEMS 替代了 LSM303DLHC，用 X-NUCLEO IKS01A1 替代了 32F401CDISCOVERY。 更新了 □ *2*□ □□□□□□□□ ， □ *6*□ *MDK-ARM*□□ □□□。 更新了 GPIO 目录并删除了 □ *2.2*□□*STM32F411xx*□ □□□□ 中的 I2C1。 更新了 □ *2.3*□□□□□□ 中用例的步骤 2。 更新了 □ *3.1*□□□□□□。 更新了 □ *3.2*□□□□□□ 中 MKD-ARM 发行版本。
2015 年 8 月 17 日	3	用 X-CUBE-BAM 替代了 STSW-STM32154。

表 **5.** 中文文档修订历史

日期	版本	变更
2017 年 6 月 30 日	1	中文初始版本。

重要通知 - 请仔细阅读

意法半导体公司及其子公司 （“ST”）保留随时对 ST 产品和 / 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于 ST 产品的最新信息。 ST 产品的销售依照订单确认时的相关 ST 销售条款。

买方自行负责对 ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的 ST 产品如有不同于此处提供的信息的规定，将导致 ST 针对该产品授予的任何保证失效。

ST 和 ST 徽标是 ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。