AN0163
应用笔记

AT32F423 PWC应用笔记

前言

AT32 的 PWC 是实现电源管理部分的外设。本文主要以低功耗为基础进行讲解和案例解析。

注：本应用笔记对应的代码是基于雅特力提供的 V2.x.x 板级支持包（BSP）而开发，对于其他版本 BSP，需要注意使用上的区别。

支持型号列表：

支持型号	AT32F423xx

目录

表目录

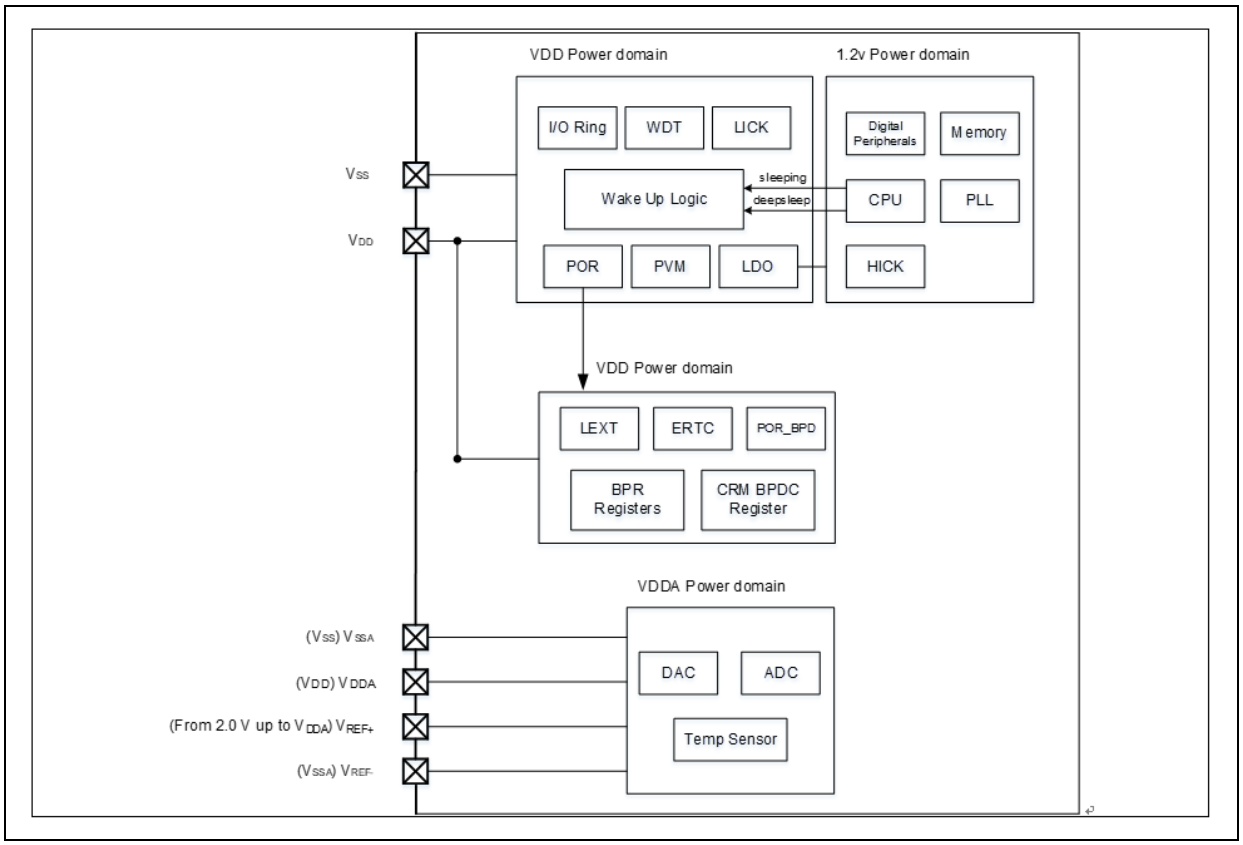
图目录

1 PWC 简介

电源控制的功能主要包含以下内容

- 供电方案，包括 VDD、VDDA 的供电
- 电源域，由 VDD/VDDA 域，1.2V 域组成
- 上电低电压复位，由上电复位和低电压复位组成
- 电压监测器，监测供电电压与设定临界值关系
- 电压调节器，电压调节器的几个工作状态
- 省电模式，包括睡眠模式、深度睡眠模式、待机模式

图 1. 电源域框图



2 PWC 基本功能解析

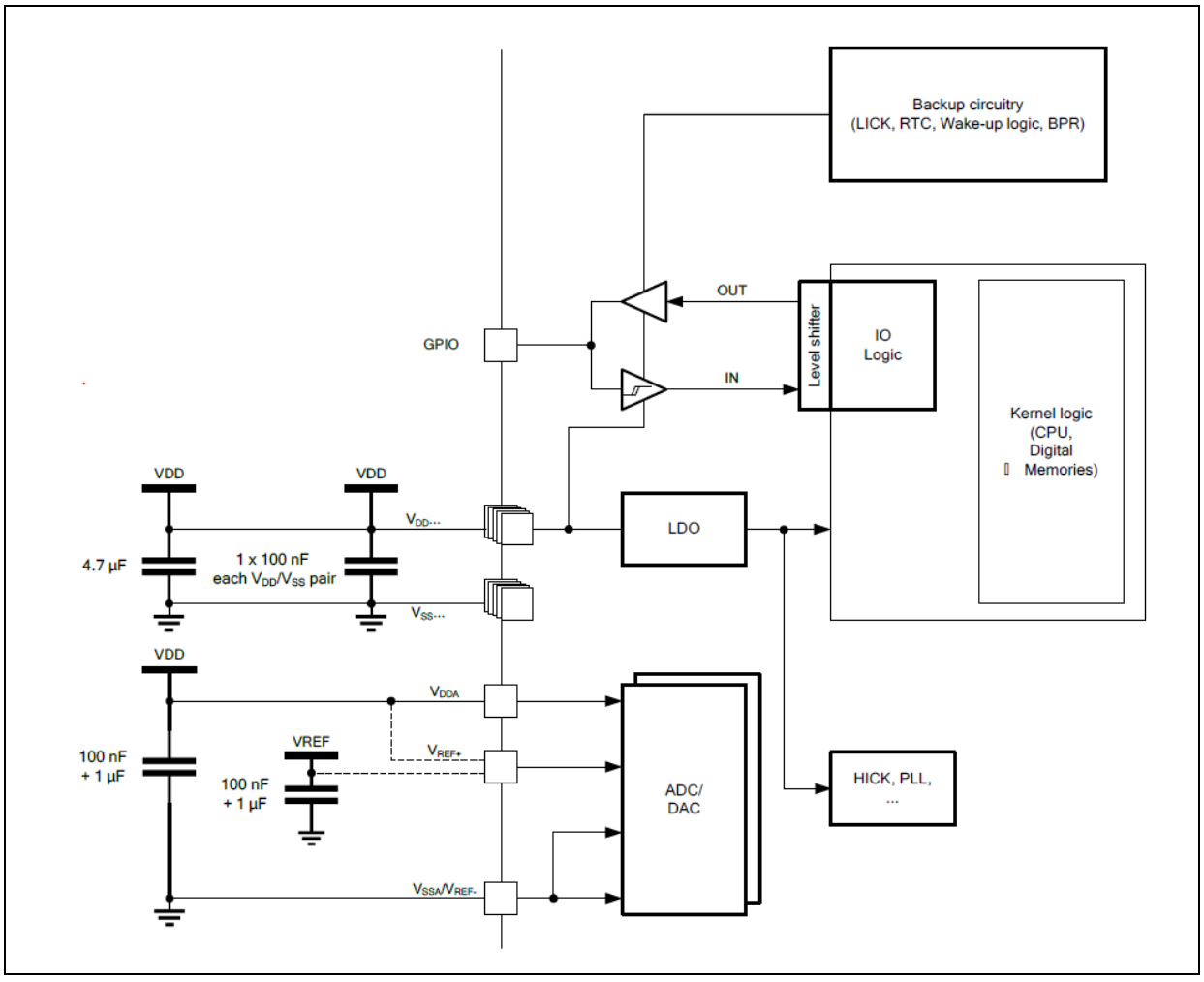
2.1 供电方案

2.1.1 功能介绍

AT32F423 MCU 的供电主包括 VDD、VDDA 及 V_{REF} 几个部分，其设计要求如下：

- V_{DD} = 2.4 ~ 3.6 V，为 GPIO 引脚和内部模块（如电压调节器）供电；
- V_{DDA} = 2.4 ~ 3.6 V，为 ADC 和 DAC 供电；
- V_{REF} = 2.0 ~ V_{DDA} V，为 ADC 提供输入参考电压；

图 2. 供电方案图



供电设计注意事项：

1）不同型号供电方案存在差异，本图仅适用于 AT32F423xx，其他型号请以实际 Datasheet 为准；

2）为保障 ADC 的有效工作，V_{DDA} 和 V_{SSA} 必须与 V_{DD} 和 V_{SS} 等电位；

3）部分型号 V_{REF} 未独立出 pin，在芯片内部将 V_{REF} 与 V_{DDA} 连接在一起，请忽略对应部分供电设计。

2.1.2 软件接口

不涉及。

2.2 电源域

2.2.1 功能介绍

AT32F423 MCU 的电源根据作用范围，可分为 VDD/VDDA 域，1.2V 域两个部分。

VDD/VDDA 域

VDD 域包括 I/O 电路、省电模式唤醒电路、看门狗 WDT、上电/低电压复位（POR/LVR）、电压调节器 LDO 以及除 PC13、PC14 和 PC15 之外的所有 PAD 电路等。

VDDA 域包括 DAC/ADC（DA/AD 转换器）、温度传感器 Temp Sensor 等。

1.2V 域

1.2V 内核域包括 CPU 内核、存储器 SRAM、内嵌数字外设以及时钟锁相环 PLL 等，其由电压调节器（LDO）供电。

2.2.2 软件接口

不涉及。

2.3 上电低电压复位

2.3.1 功能介绍

VDD/VDDA 域内置一个 POR 模拟模块用于产生电源复位。

- 上电复位：当 VDD 由 0V 上升至工作电压过程中，电源复位信号在 V_{POR} 时刻被上电释放；
- 低电压复位：当 VDD 由工作电压下降至 0V 过程中，电源复位信号在 V_{LVR} 时刻被低电压复位。

上电复位过程，复位信号的释放相较于 VDD 升压过程存在一定的时间延迟。同时为避免电源电压在合理范围内的波动造成芯片误复位，上电复位与低电压复位间具有一定迟滞。

图 3. 上电/低电压复位波形图

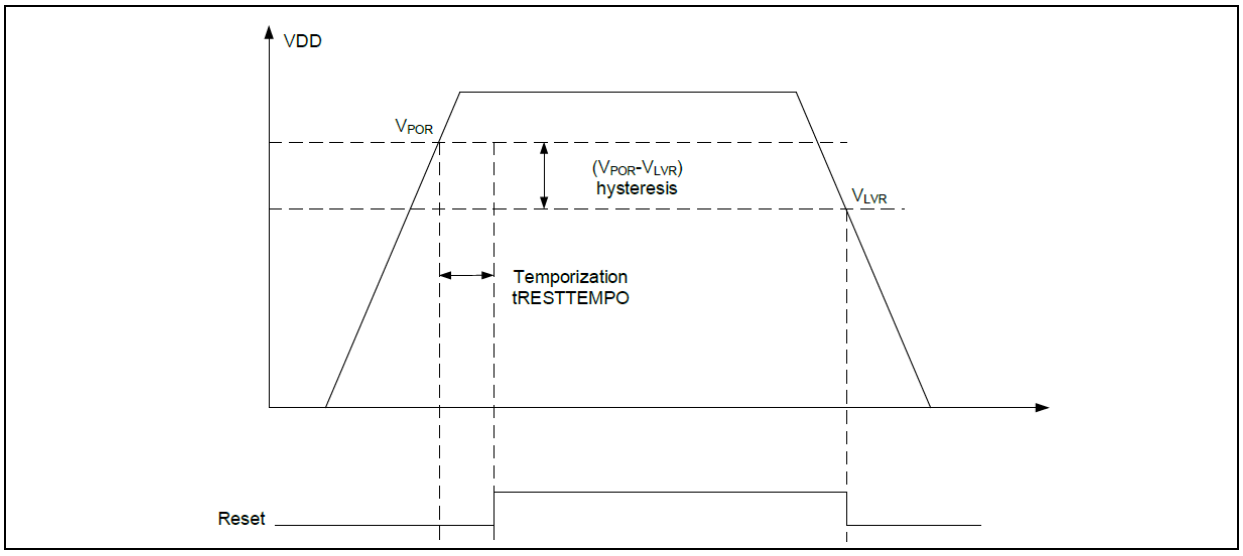


表 1. 上电/低电压复位特性表

符号	参数	条件	最小值	典型值	最大值	单位

V_{POR}(1)	上电复位阈值	-	1.81	2.10	2.40	V
V_{LVR}(1)	低电压复位阈值	-	1.68(2)	1.90	2.08	V
$V_{LVRhyst}$(1)	LVR迟滞	-	-	180	-	mV
$T_{RESTTEMPO}$(1)	复位持续时间：V_{DD}高于V_{POR}且持续时间超过$T_{RSTTEMPO}$后CPU开始运行	-	-	3.5	-	ms

(1) 由综合评估得出，不在生产中测试；
(2) 产品的特性由设计保证至最小的数值V_{LVR}；
(3) 不同型号产品对应的特性参数存在区别，本表摘自AT32F423xx，其他型号请以实际Datasheet为准；

2.3.2 软件接口

不涉及。

2.4 电压监测器

2.4.1 功能介绍

电压监测器主要用来监控供电电源的跳变，以响应一些紧急任务。

电压监测器开启后，PVMOF 将会实时的指示 VDD 与设定阈值比较的结果。当 VDD 越过设定的 PVM 阈值边界时，产生的 PVMOF 位电平变化可以通过外部中断第 16 号线产生 PVM 中断。

图 4. PVM 的阈值与输出

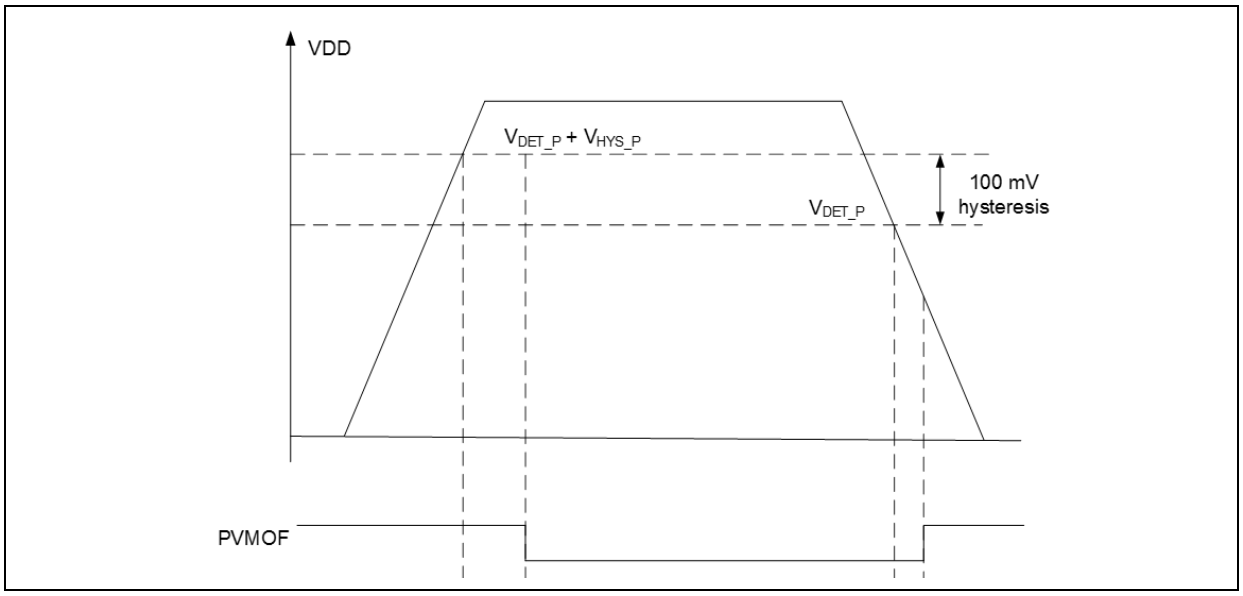


表 2. 电压监测电平选择

符号	参数	条件	最小值	典型值	最大值	单位
V_{PVM1}	PVM阈值1（PVMSEL[2:0] = 001）	上升沿	2.19	2.28	2.37	V
		下降沿	2.09	2.18	2.27	V
V_{PVM2}	PVM阈值2（PVMSEL[2:0] = 010）	上升沿	2.28	2.38	2.48	V
		下降沿	2.18	2.28	2.38	V
V_{PVM3}	PVM阈值3（PVMSEL[2:0] = 011）	上升沿	2.38	2.48	2.58	V
		下降沿	2.28	2.38	2.48	V
V_{PVM4}	PVM阈值4（PVMSEL[2:0] = 100）	上升沿	2.47	2.58	2.69	V
		下降沿	2.37	2.48	2.59	V

V_{PVM5}	PVM阈值5（PVMSEL[2:0] = 101）	上升沿	2.57	2.68	2.79	V
		下降沿	2.47	2.58	2.69	V
V_{PVM6}	PVM阈值6（PVMSEL[2:0] = 110）	上升沿	2.66	2.78	2.9	V
		下降沿	2.56	2.68	2.8	V
V_{PVM7}	PVM阈值7（PVMSEL[2:0] = 111）	上升沿	2.76	2.88	3	V
		下降沿	2.66	2.78	2.9	V
V_{HYS_P}	PVM迟滞	-	-	100	-	mV
$I_{DD\ (PVM)}$	PVM电流消耗	-	-	20	30	μA

(1) 由综合评估得出，不在生产中测试；
(2) 不同型号产品对应的特性参数存在区别，本表摘自AT32F423xx，其他型号请以实际Datasheet为准。

2.4.2 软件接口

电压监测临界值的选择，软件由独立的函数接口实现，其软件实例如下：

```
/* set the threshold voltage to 2.9v */
pwc_pvm_level_select(PWC_PVM_VOLTAGE_2V9);
```

电压监测功能使能，软件由单独的函数接口实现，其软件实例如下：

```
/* enable power voltage monitor */
pwc_power_voltage_monitor_enable(TRUE);
```

电压监测功能通常需结合外部中断使用。故需对外部中断线 16 进行初始化，其软件实例如下：

```
exint_init_type exint_init_struct;

/* config the exint line of the power voltage monitor */
exint_init_struct.line_select = EXINT_LINE_16;
exint_init_struct.line_enable = TRUE;
exint_init_struct.line_mode = EXINT_LINE_INTERRUPUT;
exint_init_struct.line_polarity = EXINT_TRIGGER_BOTH_EDGE;
exint_init(&exint_init_struct);

/* enable power voltage monitor interrupt */
nvic_irq_enable(PVM_IRQn, 0, 0);
```

注意：

1) 通过电压监测功能来实现的软件任务需安排在 EXTI_LINE_16 的中断函数内；

2) 电源电压高于临界值及低于临界值均具备产生 EXTI_LINE_16 中断的能力，应用需根据实际需求，通过 EXTI 的边沿检测配置来过滤掉不期望的中断事件。

2.5 电压调节器

2.5.1 功能介绍

AT32F423 MCU 内置电压调节器 LDO，其主要用于 MCU 的 1.2V 域部分的供电。

LDO 有四个工作模式：正常模式、低功耗模式、额外低功耗模式和关断模式。

- 正常模式：用于 CPU 的正常运行模式、睡眠模式、深度睡眠模式；
- 低功耗模式：用于 CPU 的深度睡眠模式；
- 额外低功耗模式：用于 CPU 的深度睡眠模式；

- 关断模式：用于 CPU 的待机模式。LDO 的输出为高阻状态，内核电路的供电切断，寄存器和 SRAM 的内容将丢失

其中在 MCU 复位后 LDO 保持在正常工作模式状态。

表 3. 深度睡眠模式下的典型电流消耗表

符号	参数	条件	典型值(1)		最大值(2)			单位
			V_{DD} = 2.4 V	V_{DD} = 3.3 V	T_A = 25 °C	T_A = 85 °C	T_A = 105 °C	
I_{DD}	深睡眠模式的供应电流	LDO 处于运行模式，HICK 和 HEXT 关闭，WDT 关闭	281	286	330	910	1540	μA
		LDO 处于低功耗模式，HICK 和 HEXT 关闭，WDT 关闭	141	143	160	550	980	

(1) 典型值是在T_A = 25 °C下测试得到；
(2) 由综合评估得出，不在生产中测试；
(3) 不同型号产品对应的特性参数存在区别，本表摘自AT32F423xx，其他型号请以实际Datasheet为准。

用户还可根据实际需求调整 AT32F423 MCU 内置电压调节器 LDO 的功耗等级来节省整机功耗。LDO 有节能和正常两种功耗等级供用户选择。选择功耗等级时需要遵照如下限定：

图 5. 电压调节器不同功耗等级的使用限定

符号	参数	条件		最小值	最大值	单位
f_{HCLK}	内部AHB时钟频率	LDO电压	1.3 V	0	150	MHz
			1.2 V	0	120	
			1.0 V	0	64	
f_{PCLK1}	内部APB1时钟频率	LDO电压	1.3 V	0	120	MHz
			1.2 V，1.0 V	0	f_{HCLK}	

电压调节器功耗等级调整需要严格按照如下步骤进行：

1) 系统时钟切换至 HICK 或 HEXT
2) 修改 LDO 输出电压（PWC_LDOOV 寄存器的 LDOOVSEL）
3) 设置闪存性能选择寄存器（FLASH_PSR）
4) 设置 PLL 相关寄存器至目标频率，开启 PLL，等待 PLL_STBL
5) 设置 AHB 及 APB 预除频系数
6) 若 PLL 频率大于 108MHz， 打开顺滑切换
7) 切换系统时钟至 PLL

2.5.2 软件接口

深度睡眠模式下的 LDO 工作模式选择，其软件实例如下：

```
/* congfig the voltage regulator mode */
pwc_voltage_regulate_set(PWC_REGULATOR_LOW_POWER);
```

注意：仅 CPU 的深度睡眠模式下才可配置 LDO 的工作模式。

3 PWC 省电模式解析

MCU 的工作不可避免的会产生一定的功耗，对于应用实际而言，降低功耗的考量十分重要。结合 MCU 特性及应用条件，以下罗列部分典型降低功耗的方法。

- CPU 运行状态下，适当降低系统时钟；
- CPU 运行状态下，关闭 AHB 和 APB 总线上未被使用的外设时钟；
- CPU 无需运行时，MCU 进入省电模式（睡眠模式、深度睡眠模式、待机模式）。

3.1 睡眠模式

3.1.1 功能介绍

在睡眠模式下，CPU 时钟关闭，其他时钟保持正常工作，电压调节器正常工作，所有的 I/O 管脚都保持它们在运行模式时的状态，LDO 以正常功耗模式提供 1.2V 电源（CPU 内核、内存和内嵌外设）。

Cortex™-M4F 内核设计控制位 SLEEPONEXIT，其功能如下：

图 6. SLEEPONEXIT 功能说明图

[1]	SLEEPONEXIT	Indicates sleep-on-exit when returning from Handler mode to Thread mode: 0 = do not sleep when returning to Thread mode. 1 = enter sleep, or deep sleep, on return from an ISR. Setting this bit to 1 enables an interrupt driven application to avoid returning to an empty main application.

结合 SLEEPONEXIT 位的设定，MCU 支持两种睡眠机制：

- SLEEPONEXIT = 0，执行睡眠指令，此时可立即进入睡眠模式；
- SLEEPONEXIT = 1，执行睡眠指令，此时每当系统从最低优先级的中断处理程序中退出时，会立即进入睡眠模式。

睡眠模式进入及退出

WFI

进入条件：SLEEPDEEP = 0，再执行 WFI 命令行；

唤醒条件：任意外设中断（该外设的中断使能位及 NVIC 使能位均被使能）的响应；

WFE

进入条件：SLEEPDEEP = 0，再执行 WFE 命令行；

唤醒条件：

- 任意外设中断（该外设的中断使能位及 NVIC 使能位均被使能）的响应；
- 任意 EXINT 线（该 EXINT 线必须配置为事件模式）上产生的唤醒事件；
- SEVONPEND = 1，任意外设中断（该外设的 NVIC 使能位未使能）的产生。在进入睡眠之前要确保外设中断挂起位和 NVIC 通道挂起位均未处于置位状态。且此方式唤醒后，软件需清除外设中断挂起位和 NVIC 通道挂起位。

其中， SLEEPDEEP、SEVONPEND 均为 Cortex™-M4F 内设计核控制位。其功能介绍如下（详细的说明可参考 Cortex™-M4F 手册）：

图 7. SLEEPDEEP/SEVONPEND 功能说明图

[4]	SEVONPEND	Send Event on Pending bit: 0 = only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded 1 = enabled events and all interrupts, including disabled interrupts, can wakeup the processor. When an event or interrupt enters pending state, the event signal wakes up the processor from WFE. If the processor is not waiting for an event, the event is registered and affects the next WFE. The processor also wakes up on execution of an SEV instruction or an external event.
[3]	-	Reserved.
[2]	SLEEPDEEP	Controls whether the processor uses sleep or deep sleep as its low power mode: 0 = sleep 1 = deep sleep.

3.1.2 软件接口

睡眠模式的进入由独立的软件接口实现，其软件实例如下：

```
/* enter sleep mode */
pwc_sleep_mode_enter(PWC_SLEEP_ENTER_WFI);
```

注意：

1) *WFE 进入的睡眠模式唤醒所需的时间最短，因为没有时间损失在中断的进入或退出上；*
2) *SLEEPONEXIT 规则可结合 WFI 或 WFE 使用，但应用设计时需注意其与唤醒条件的配合；*
3) *应用设计时不开 PWC 接口时钟条件下，执行睡眠模式进入函数同样会实现 CPU 暂停并等待中断或事件的效果，只是其功耗不会被明显降低。*

3.2 深度睡眠模式

3.2.1 功能介绍

在深度睡眠模式下，所有 1.2V 时钟关闭，HICK 和 HEXT 振荡器都被关闭，电压调节器以正常工作或低功耗工作状态给 1.2V 域供电，所有 I/O 管脚都保持它们在运行模式时的状态，SRAM 和寄存器内容保持。

深度睡眠模式可与 LDO 的正常模式、低功耗模式、额外低功耗模式配合使用以进一步节省功耗。

深度睡眠模式进入及退出

WFI

进入条件：SLEEPDEEP = 1，LPSEL = 0，再执行 WFI 命令行；

唤醒条件：任意 EXINT 线（该 EXINT 线需配置为中断模式且 NVIC 使能位被使能）上的中断响应。

WFE

进入条件：SLEEPDEEP = 1，LPSEL = 0，再执行 WFE 命令行；

唤醒条件：任意 EXINT 线（该 EXINT 线需配置为事件模式）上产生的唤醒事件。

其中， SLEEPDEEP 为 Cortex™-M4F 内设计核控制位。相关介绍请参考 3.1.1 节说明。

系统从深度睡眠模式退出时，HICK RC 振荡器被自动开启并在稳定后被选为系统时钟。

3.2.2 软件接口

深度睡眠模式的进入由独立的软件接口实现，其软件实例如下：

```
/* congfig the voltage regulator mode */
pwc_voltage_regulate_set(PWC_REGULATOR_LOW_POWER);
/* enter deep sleep mode */
pwc_deep_sleep_mode_enter(PWC_DEEP_SLEEP_ENTER_WFI);
```

注意：

1) *退出深度睡眠模式后，HICK RC 振荡器被选为系统时钟，软件需根据需求对系统时钟重新设定；*
2) *退出深度睡眠模式时，LDO 会保持正常模式，因此若进深睡眠前配置为了低功耗模式的话，LDO 的模式切换需要一定耗时，从而会增加额外的唤醒时间。*

3.3 待机模式

3.3.1 功能介绍

待机模式可最大限度的降低系统功耗，在该模式下，电压调节器关闭，只有电池供电的寄存器和待机电路维持供电，其他的 1.2V 供电区域，PLL、HICK 和 HEXT 振荡器都被断电。寄存器和 SRAM 中的内容也会丢失。

在待机模式下，除了复位管脚、被设置为防侵入或校准输出时的 TAMPER 管脚和被使能的唤醒管脚之外，所有的 I/O 管脚处于高阻态。

待机模式进入及退出

进入条件：SLEEPDEEP = 1，LPSEL = 1，再执行 WFI/WFE 命令行；

退出条件：

- WKUP 管脚的上升沿；发生唤醒时会置位 SEF、SWEF 标志
- NRST 管脚上外部复位；发生复位时会置位 SEF、NRSTF 标志
- WDT 复位；发生复位时会置位 SEF、WDTRSTF、NRSTF 标志
- 实时时钟事件的上升沿；发生唤醒时会置位 SEF、SWEF、及实时时钟事件对应标志

  实时时钟事件为 ERTC 闹钟事件、ERTC 入侵事件、ERTC 时间戳、ERTC 周期性自动唤醒事件。

实时时钟在部分型号为 RTC，部分型号为 ERTC，部分 ERTC 型号不支持周期性自动唤醒，部分型号支持双闹钟。且部分型号具备多个 WKUP 管脚等，这些差异部分请以实际芯片手册为准。

3.3.2 软件接口

待机模式的进入由独立的软件接口实现，其软件实例如下：

```
/* enter standby mode */
pwc_standby_mode_enter();
```

用于待机模式唤醒的 WKUP 管脚使能由独立的软件接口实现，其软件实例如下：

```
/* enable wakeup pin */
pwc_wakeup_pin_enable(PWC_WAKEUP_PIN_1, TRUE);
```

注意：

1) *SWEF 标志为待机唤醒事件标志，其处于置位状态下执行进入待机模式命令，会立即产生复位。故在进入待机模式前，软件需确保 SWEF 标志已被清除；*
2) *部分型号具备多个 WKUP 管脚，具体请以实际芯片手册为准*
3) *实时时钟在部分型号为 RTC，部分型号为 ERTC，具体请以实际芯片手册为准；*
4) *部分 ERTC 型号不支持周期性自动唤醒，部分型号支持双闹钟，具体请以实际芯片手册为准。*

3.4 省电模式特性

3.4.1 省电模式电流消耗

省电模式下的电流消耗会被明显降低， Datasheet 都有经过详细测试后的数据记录。如下表示例记录：

表 4. 深度睡眠和待机模式下的典型电流消耗表

符号	参数	条件	典型值(1)		最大值(2)			单位
			V_{DD} = 2.4 V	V_{DD} = 3.3 V	T_A = 25 °C	T_A = 85 °C	T_A = 105 °C	
I_{DD}	深睡眠模式的供应电流	LDO 处于运行模式，HICK 和 HEXT 关闭，WDT 关闭	281	286	330	910	1540	μA
		LDO 处于低功耗模式，HICK 和 HEXT 关闭，WDT 关闭	141	143	160	550	980	
	待机模式的供应电流	LEXT和ERTC关闭	2.6	3.9	4.6	6.8	8.1	μA
		LEXT和ERTC开启	3.6	5.4	6.5	8.8	12.9	

(1) 典型值是在TA = 25 °C下测试得到；
(2) 由综合评估得出，不在生产中测试；
(3) 睡眠模式下的电流消耗与运行模式间差异不是特别大，本表未做罗列，具体请参考Datasheet；
(4) 不同型号产品对应的特性参数存在区别，本表摘自AT32F423xx，其他型号请以实际Datasheet为准。

3.4.2 省电模式唤醒时间

省电模式下的唤醒均需要等待及稳定时间， Datasheet 都有经过详细测试后的数据记录。如下表记录：

表 5. 省电模式的唤醒时间表

符号	参数	条件	典型值	单位
$t_{WUSLEEP}$	从睡眠模式唤醒	-	3.7	μs
$t_{WUDEEPSLEEP}$	从深睡眠模式唤醒	LDO处于运行模式	450	μs
		LDO处于低功耗模式	500	
$t_{WUSTDBY}$	从待机模式唤醒	-	800	μs

(1) 不同型号产品对应的特性参数存在区别，本表摘自AT32F423xx，其他型号请以实际Datasheet为准。

4 案例 PWC 电压监测

4.1 功能简介

PWC 的电压监测器主要用来监控供电电源的跳变，以响应一些紧急任务。

本例将演示如何配置使用电压监测功能。

4.2 资源准备

1) 硬件环境

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_f423\examples \pwc\power_voltage_monitor

4.3 软件设计

1) 配置流程

- 开启 PWC 时钟
- 设置电压监测临界值
- 使能电压监测功能
- 配置 EXINT 线 16 为中断模式并使能
- 电压监测对应 NVIC 中断使能

2) 代码介绍

- EXINT 线 16 配置函数代码

```
void pvm_exint_config(void)
{
  exint_init_type exint_init_struct;

  /* config the exint line of the power voltage monitor */
  exint_init_struct.line_select = EXINT_LINE_16;
  exint_init_struct.line_enable = TRUE;
  exint_init_struct.line_mode = EXINT_LINE_INTERRUPUT;
  exint_init_struct.line_polarity = EXINT_TRIGGER_BOTH_EDGE;
  exint_init(&exint_init_struct);
}
```

- 中断服务函数代码

```
/* 获取普通通道数据传输完成状态 */
void PVM_IRQHandler(void)
{
  if(exint_flag_get(EXINT_LINE_16) != RESET)
  {
    /* clear exint line flag */
    exint_flag_clear(EXINT_LINE_16);

    /* toggle led */
```

```
    at32_led_toggle(LED2);
    at32_led_toggle(LED3);
    at32_led_toggle(LED4);
    /* user code add */
  }
}
```

■ main 函数代码

```
int main(void)
{
  __IO uint32_t index = 0;

  /* congfig the system clock */
  system_clock_config();

  /* init at start board */
  at32_board_init();

  /* config priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* turn on the led light */
  at32_led_on(LED2);
  at32_led_on(LED3);
  at32_led_on(LED4);

  /* enable pwc clock */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

  /* set the threshold voltage to 2.9v */
  pwc_pvm_level_select(PWC_PVM_VOLTAGE_2V9);

  /* enable power voltage monitor */
  pwc_power_voltage_monitor_enable(TRUE);

  /* config the exint line of the power voltage monitor */
  pvm_exint_config();

 /* when power voltage monitor enable,exint line16 flag will be set */
  while(exint_flag_get(EXINT_LINE_16) == RESET);

  /* clear the unexpected exint line flag */
  exint_flag_clear(EXINT_LINE_16);

  /* clear the unexpected nvic pending flag */
  NVIC_ClearPendingIRQ(PVM_IRQn);
```

```
  /* enable power voltage monitor interrupt */
  nvic_irq_enable(PVM_IRQn, 0, 0);

  while(1)
  {
  }
}
```

4.4 实验效果

可通过 AT-START BOARD 上的 LED 翻转查看实现效果。

每次供电电源电压越过阈值边界时，LED2、LED3、LED4 的状态都会发生翻转。

5 案例 TMR2 溢出中断唤醒 PWC 睡眠模式

5.1 功能简介

本例将演示 PWC 睡眠模式的使用。其中，唤醒源使用 TMR2 的溢出中断。

5.2 资源准备

1) 硬件环境

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_f423\examples\pwc\sleep_tmr2

5.3 软件设计

1) 配置流程

- 开启 PWC 时钟
- 初始化 TMR2 并使能溢出中断
- 使能 TMR2 中断的 NVIC 中断
- 使能 TMR2
- 循环执行进睡眠模式命令并等待唤醒

2) 代码介绍

- TMR2 配置函数代码

```
void tmr2_config(void)
{
  crm_clocks_freq_type crm_clocks_freq_struct = {0};

  /* enable tmr 2 clock */
  crm_periph_clock_enable(CRM_TMR2_PERIPH_CLOCK, TRUE);

  /* get system clock */
  crm_clocks_freq_get(&crm_clocks_freq_struct);

  /* (systemclock/(systemclock/10000))/10000 = 1Hz(1s) */
  tmr_base_init(TMR2, 9999, (crm_clocks_freq_struct.sclk_freq/10000 - 1));

  /* config the counting direction */
  tmr_cnt_dir_set(TMR2, TMR_COUNT_UP);

  /* config the clock divider value */
  tmr_clock_source_div_set(TMR2, TMR_CLOCK_DIV1);

  /* enable tmr 2 interrupt */
  tmr_interrupt_enable(TMR2, TMR_OVF_INT, TRUE);

  /* config tmr 2 nvic */
```

```
  nvic_irq_enable(TMR2_GLOBAL_IRQn, 0, 0);

  /* enable tmr 2 */
  tmr_counter_enable(TMR2, TRUE);
}
```

■ 中断服务函数代码

```
void TMR2_GLOBAL_IRQHandler(void)
{
  if(tmr_flag_get(TMR2, TMR_OVF_FLAG) != RESET)
  {
    /* clear timer 2 ovf flag */
    tmr_flag_clear(TMR2, TMR_OVF_FLAG);

    /* toggle led */
    at32_led_toggle(LED4);
  }
}
```

■ main 函数代码

```
int main(void)
{
  __IO uint32_t index = 0;
  __IO uint32_t systick_index = 0;

  /* congfig the system clock */
  system_clock_config();

  /* init at start board */
  at32_board_init();

  /* config priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* turn on the led light */
  at32_led_on(LED2);
  at32_led_on(LED3);
  at32_led_on(LED4);

  /* enable pwc clock */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

  /* config tmr 2 */
  tmr2_config();

  while(1)
  {
```

```
      at32_led_off(LED2);

      /* save systick register configuration */
      systick_index = SysTick->CTRL;
      systick_index &= ~((uint32_t)0xFFFFFFFE);

      /* disable systick */
      SysTick->CTRL &= (uint32_t)0xFFFFFFFE;

      /* enter sleep mode */
      pwc_sleep_mode_enter(PWC_SLEEP_ENTER_WFI);

      /* restore systick register configuration */
      SysTick->CTRL |= systick_index;

      /* wake up from sleep mode */
      at32_led_on(LED2);
      delay_ms(500);
    }
}
```

5.4 实验效果

可通过 AT-START BOARD 上的 LED 翻转查看实现效果。

LED2 亮：MCU 处于运行模式；

LED2 灭：MCU 处于睡眠模式；

LED4 状态翻转： TMR2 溢出事件发生并唤醒了睡眠模式。

6 案例 USART1 接收中断唤醒 PWC 睡眠模式

6.1 功能简介

本例将演示 PWC 睡眠模式的使用。其中，唤醒源使用 USART1 的接收中断。

6.2 资源准备

3) 硬件环境

对应产品型号的 AT-START BOARD

4) 软件环境

project\at_start_f423\examples\pwc\sleep_usart1

6.3 软件设计

3) 配置流程

- 开启 PWC 时钟
- 初始化 USART1 并使能接收数据缓冲器满中断
- 使能 USART1 中断的 NVIC 中断
- 使能 USART1
- 循环执行进睡眠模式命令并等待唤醒

4) 代码介绍

- USART1 配置函数代码

```
void usart1_config(uint32_t baudrate)
{
  gpio_init_type gpio_init_struct;

  /* enable the uart1 and gpio clock */
  crm_periph_clock_enable(CRM_USART1_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);

  gpio_default_para_init(&gpio_init_struct);

  /* configure the uart1 tx pin */
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type  = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init_struct.gpio_pins = GPIO_PINS_9;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init(GPIOA, &gpio_init_struct);
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE9, GPIO_MUX_7);

  /* configure the uart1 rx pin */
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type  = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
```

```
  gpio_init_struct.gpio_pins = GPIO_PINS_10;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init(GPIOA, &gpio_init_struct);
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE10, GPIO_MUX_7);

  nvic_irq_enable(USART1_IRQn, 0, 0);

  /* configure uart param */
  usart_init(USART1, baudrate, USART_DATA_8BITS, USART_STOP_1_BIT);
  usart_parity_selection_config(USART1, USART_PARITY_NONE);
  usart_transmitter_enable(USART1, TRUE);
  usart_receiver_enable(USART1, TRUE);
  usart_hardware_flow_control_set(USART1,USART_HARDWARE_FLOW_NONE);
  usart_interrupt_enable(USART1, USART_RDBF_INT, TRUE);
  usart_enable(USART1, TRUE);
}
```

■ 中断服务函数代码

```
void USART1_IRQHandler(void)
{
  if(usart_flag_get(USART1, USART_RDBF_FLAG) != RESET)
  {
    /* clear rdbf flag */
    usart1_index = usart_data_receive(USART1);

    /* toggle led */
    at32_led_toggle(LED4);
  }
}
```

■ main 函数代码

```
int main(void)
{
  __IO uint32_t index = 0;
  __IO uint32_t systick_index = 0;

  /* congfig the system clock */
  system_clock_config();

  /* init at start board */
  at32_board_init();

  /* config priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* turn on the led light */
  at32_led_on(LED2);
```

```
  at32_led_on(LED3);
  at32_led_on(LED4);

  /* enable pwc clock */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

  /* config usart1 */
  usart1_config(115200);

  printf("exit sleep mode by usart1 rdbf interrupt \r\n");

  while(1)
  {
    at32_led_off(LED2);
    at32_led_off(LED3);
    printf("now enter sleep mode \r\n");

    /* save systick register configuration */
    systick_index = SysTick->CTRL;
    systick_index &= ~((uint32_t)0xFFFFFFFE);

    /* disable systick */
    SysTick->CTRL &= (uint32_t)0xFFFFFFFE;

    /* enter sleep mode */
    pwc_sleep_mode_enter(PWC_SLEEP_ENTER_WFI);

    /* restore systick register configuration */
    SysTick->CTRL |= systick_index;

    /* wake up from sleep mode */
    printf("now exit sleep mode by usart1 rdbf interrupt \r\n");
    printf("\r\n");
    at32_led_on(LED2);
    delay_ms(500);
  }
}
```

6.4 实验效果

可通过 AT-START BOARD 上的 LED 翻转查看实现效果。

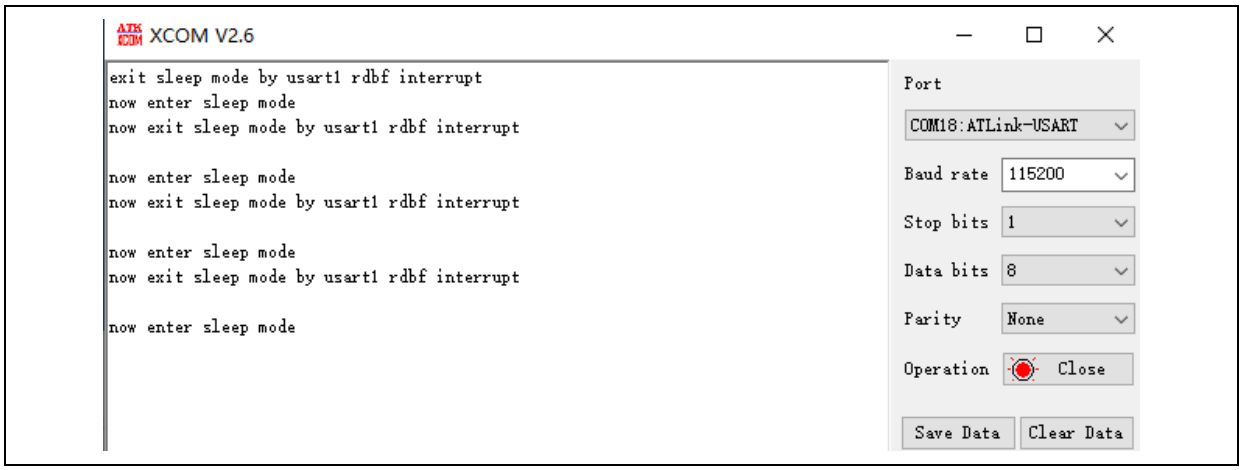
LED2 亮：MCU 处于运行模式；

LED2 灭：MCU 处于睡眠模式；

LED4 状态翻转： USART1 接收数据缓冲器满终端发生并唤醒了睡眠模式。

也可通过 USART1 的串口打印查看实验效果，如下：

图 8. 案例 6 测试输出


XCOM V2.6
exit sleep mode by usart1 rdbf interrupt
now enter sleep mode
now exit sleep mode by usart1 rdbf interrupt
now enter sleep mode
now exit sleep mode by usart1 rdbf interrupt
now enter sleep mode
now exit sleep mode by usart1 rdbf interrupt
now enter sleep mode
Port
COM18:ATLink-USART
Baud rate
115200
Stop bits
1
Data bits
8
Parity
None
Operation
Close
Save Data
Clear Data

7 案例 ERTC 闹钟唤醒 PWC 深度睡眠模式

7.1 功能简介

本例将演示 PWC 深度睡眠模式的使用。其中，唤醒源使用 ERTC 闹钟中断。

7.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_f423\examples\pwc\deepsleep_ertc_alarm

7.3 软件设计

1) 配置流程

- 开启 PWC 时钟
- 开启电池供电区域的写入使能
- 等待 VDD 上升到 2.57V 后执行 ERTC 初始化
- 配置 EXINT 线 17 为中断模式并使能
- ERTC 闹钟初始化并使能闹钟中断及其 NVIC 中断
- 设定 ERTC 闹钟值
- 设定电压调节器为节能等级
- 执行进深度睡眠模式命令并等待唤醒
- 唤醒后设定电压调节器为正常等级
- 重新进行系统时钟的恢复设定
- 从“设定 ERTC 闹钟值”开始循环，实现深度睡眠模式的循环唤醒

2) 代码介绍

- ERTC 配置函数代码

```
void ertc_config(void)
{
  /* allow access to ertc */
  pwc_battery_powered_domain_access(TRUE);

  /* reset ertc domain */
  crm_battery_powered_domain_reset(TRUE);
  crm_battery_powered_domain_reset(FALSE);

  /* enable the lext osc */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_LEXT, TRUE);

  /* wait till lext is ready */
  while(crm_flag_get(CRM_LEXT_STABLE_FLAG) == RESET);
```

```
  /* select the ertc clock source */
  crm_ertc_clock_select(CRM_ERTC_CLOCK_LEXT);

  /* enable the ertc clock */
  crm_ertc_clock_enable(TRUE);

  /* deinitializes the ertc registers */
  ertc_reset();

  /* wait for ertc apb registers synchronisation */
  ertc_wait_update();

  /* configure the ertc data register and ertc prescaler
     ck_spre(1hz) = ertcclk(lext) /(ertc_clk_div_a + 1)*(ertc_clk_div_b + 1)*/
  ertc_divider_set(127, 255);

  /* configure the hour format is 24-hour format*/
  ertc_hour_mode_set(ERTC_HOUR_MODE_24);

  /* set the date: friday june 11th 2021 */
  ertc_date_set(21, 6, 11, 5);

  /* set the time to 06h 20mn 00s am */
  ertc_time_set(6, 20, 0, ERTC_AM);
}
```

■ ERTC 闹钟配置函数代码

```
void ertc_alarm_config(void)
{
  exint_init_type exint_init_struct;

  /* config the exint line of the ertc alarm */
  exint_init_struct.line_select    = EXINT_LINE_17;
  exint_init_struct.line_enable    = TRUE;
  exint_init_struct.line_mode      = EXINT_LINE_INTERRUPUT;
  exint_init_struct.line_polarity = EXINT_TRIGGER_RISING_EDGE;
  exint_init(&exint_init_struct);

  /* set the alarm 05h:20min:10s */
  ertc_alarm_mask_set(ERTC_ALA, ERTC_ALARM_MASK_DATE_WEEK | ERTC_ALARM_MASK_HOUR
| ERTC_ALARM_MASK_MIN);
  ertc_alarm_week_date_select(ERTC_ALA, ERTC_SLECT_DATE);
  ertc_alarm_set(ERTC_ALA, 31, 6, 20, 5, ERTC_AM);

  /* enable the ertc interrupt */
  nvic_irq_enable(ERTCAlarm_IRQn, 0, 0);
```

```
  /* enable ertc alarm a interrupt */
  ertc_interrupt_enable(ERTC_ALA_INT, TRUE);

  /* enable the alarm */
  ertc_alarm_enable(ERTC_ALA, TRUE);
}
```

■ ERTC 闹钟值设定函数代码

```
void ertc_alarm_value_set(uint32_t alam_index)
{
  ertc_time_type ertc_time_struct;

  /* disable the alarm */
  ertc_alarm_enable(ERTC_ALA, FALSE);
  ertc_calendar_get(&ertc_time_struct);
  ertc_alarm_set(ERTC_ALA, ertc_time_struct.day, ertc_time_struct.hour, ertc_time_struct.min,
(ertc_time_struct.sec + alam_index)% 60, ertc_time_struct.ampm);

  /* disable the alarm */
  ertc_alarm_enable(ERTC_ALA, TRUE);
}
```

■ 系统时钟恢复函数代码

```
void system_clock_recover(void)
{
  /* enable external high-speed crystal oscillator - hext */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_HEXT, TRUE);

  /* wait till hext is ready */
  while(crm_hext_stable_wait() == ERROR);

  /* enable pll */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_PLL, TRUE);

  /* wait till pll is ready */
  while(crm_flag_get(CRM_PLL_STABLE_FLAG) == RESET);

  /* enable auto step mode */
  crm_auto_step_mode_enable(TRUE);

  /* select pll as system clock source */
  crm_sysclk_switch(CRM_SCLK_PLL);

  /* wait till pll is used as system clock source */
  while(crm_sysclk_switch_status_get() != CRM_SCLK_PLL);
}
```

■ 中断服务函数代码

```
void ERTCAlarm_IRQHandler(void)
{
  if(ertc_flag_get(ERTC_ALAF_FLAG) != RESET)
  {
    /* clear ertc alarm flag */
    ertc_flag_clear(ERTC_ALAF_FLAG);

    /* clear exint line flag */
    exint_flag_clear(EXINT_LINE_17);

    /* toggle led */
    at32_led_toggle(LED4);
  }
}
```

■ main 函数代码

```
int main(void)
{
  __IO uint32_t systick_index = 0;
  __IO uint32_t delay_index = 0;

  /* congfig the system clock */
  system_clock_config();

  /* init at start board */
  at32_board_init();

  at32_led_on(LED2);
  at32_led_on(LED3);
  at32_led_on(LED4);

  /* config priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* enable pwc and bpr clock */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

  /* congfig the voltage regulator mode.only used with deep sleep mode */
  pwc_voltage_regulate_set(PWC_REGULATOR_EXTRA_LOW_POWER);

  /* add a necessary delay to ensure that Vdd is higher than the operating
     voltage of battery powered domain (2.57V) when the battery powered
     domain is powered on for the first time and being operated. */
  delay_ms(60);
```

```
/* config ertc or other operations of battery powered domain */
ertc_config();

/* set the wakeup time: 06h:20min:5s */
ertc_alarm_config();
while(1)
{
  /* turn off the led light */
  at32_led_off(LED2);

  /* save systick register configuration */
  systick_index = SysTick->CTRL;
  systick_index &= ~((uint32_t)0xFFFFFFFE);

  /* disable systick */
  SysTick->CTRL &= (uint32_t)0xFFFFFFFE;

  ertc_alarm_value_set(1);

  /* select system clock source as hick before ldo set */
  crm_sysclk_switch(CRM_SCLK_HICK);

  /* wait till hick is used as system clock source */
  while(crm_sysclk_switch_status_get() != CRM_SCLK_HICK)
  {
  }

  pwc_optimization_level_set(PWC_ECONOMIZE_LEVEL);

  /* congfig the voltage regulator mode */
  pwc_voltage_regulate_set(PWC_REGULATOR_LOW_POWER);

  /* enter deep sleep mode */
  pwc_deep_sleep_mode_enter(PWC_DEEP_SLEEP_ENTER_WFI);

  /* wake up from deep sleep mode, restore systick register configuration */
  SysTick->CTRL |= systick_index;

  /* turn on the led light */
  at32_led_on(LED2);

  /* wait clock stable */
  delay_us(120);

  /* resume pwc optimization level to normal level before system clock source enhance */
  pwc_optimization_level_set(PWC_NORMAL_LEVEL);
```

```
    /* congfig the system clock */
    system_clock_recover();

    delay_ms(500);
  }
}
```

7.4 实验效果

可通过 AT-START BOARD 上的 LED 翻转查看实现效果。

LED2 亮：MCU 处于运行模式；

LED2 灭：MCU 处于深度睡眠模式；

LED4 状态翻转： ERTC 闹钟中断发生并唤醒了深度睡眠模式。

8 案例 ERTC 入侵事件唤醒 PWC 深度睡眠模式

8.1 功能简介

本例将演示 PWC 深度睡眠模式的使用。其中，唤醒源使用 ERTC 入侵事件中断。

8.2 资源准备

3) 硬件环境:

对应产品型号的 AT-START BOARD

4) 软件环境

project\at_start_f423\examples\pwc\deepsleep_ertc_tamper

8.3 软件设计

3) 配置流程

- 开启 PWC 时钟
- 开启电池供电区域的写入使能
- 等待 VDD 上升到 2.57V 后执行 ERTC 初始化
- 配置 EXINT 线 21 为中断模式并使能
- ERTC 入侵事件检测初始化并使能入侵事件检测中断及其 NVIC 中断
- 设定电压调节器为节能等级
- 执行进深度睡眠模式命令并等待唤醒
- 唤醒后设定电压调节器为正常等级
- 重新进行系统时钟的恢复设定
- 从“设定电压调节器为节能等级”开始循环，实现深度睡眠模式的循环唤醒

4) 代码介绍

- ERTC 配置函数代码

```
void ertc_config(void)
{
  /* enable the pwc clock interface */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

  /* allow access to ertc */
  pwc_battery_powered_domain_access(TRUE);

  /* reset ertc domain */
  crm_battery_powered_domain_reset(TRUE);
  crm_battery_powered_domain_reset(FALSE);

  /* enable the lext osc */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_LEXT, TRUE);

  /* wait till lext is ready */
  while(crm_flag_get(CRM_LEXT_STABLE_FLAG) == RESET);
```

```
  /* select the ertc clock source */
  crm_ertc_clock_select(CRM_ERTC_CLOCK_LEXT);

  /* enable the ertc clock */
  crm_ertc_clock_enable(TRUE);

  /* deinitializes the ertc registers */
  ertc_reset();
}
```

■ ERTC 入侵事件配置函数代码

```
void ertc_tamper_config (void)
{
  exint_init_type exint_init_struct;

  /* config the exint line of the ertc tamper */
  exint_init_struct.line_select     = EXINT_LINE_21;
  exint_init_struct.line_enable     = TRUE;
  exint_init_struct.line_mode       = EXINT_LINE_INTERRUPUT;
  exint_init_struct.line_polarity = EXINT_TRIGGER_RISING_EDGE;
  exint_init(&exint_init_struct);

  /* enable tamper irqchannel */
  nvic_irq_enable(TAMP_STAMP_IRQn, 0, 0);

  /* disable the tamper 1 detection */
  ertc_tamper_enable(ERTC_TAMPER_1, FALSE);

  /* clear tamper 1 pin event(tamp1f) pending flag */
  ertc_flag_clear(ERTC_TP1F_FLAG);

  /* configure the tamper 1 trigger */
  ertc_tamper_valid_edge_set(ERTC_TAMPER_1, ERTC_TAMPER_EDGE_RISING);

  /* enable the tamper interrupt */
  ertc_interrupt_enable(ERTC_TP_INT, TRUE);

  /* enable the tamper 1 detection */
  ertc_tamper_enable(ERTC_TAMPER_1, TRUE);
}
```

■ 系统时钟恢复函数代码

```
void system_clock_recover(void)
{
  /* enable external high-speed crystal oscillator - hext */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_HEXT, TRUE);
```

```
  /* wait till hext is ready */
  while(crm_hext_stable_wait() == ERROR);

  /* enable pll */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_PLL, TRUE);

  /* wait till pll is ready */
  while(crm_flag_get(CRM_PLL_STABLE_FLAG) == RESET);

  /* enable auto step mode */
  crm_auto_step_mode_enable(TRUE);

  /* select pll as system clock source */
  crm_sysclk_switch(CRM_SCLK_PLL);

  /* wait till pll is used as system clock source */
  while(crm_sysclk_switch_status_get() != CRM_SCLK_PLL);
}
```

■ 中断服务函数代码

```
void TAMP_STAMP_IRQHandler(void)
{
  if(ertc_flag_get(ERTC_TP1F_FLAG) != RESET)
  {
    /* clear ertc alarm flag */
    ertc_flag_clear(ERTC_TP1F_FLAG);

    /* clear exint line flag */
    exint_flag_clear(EXINT_LINE_21);

    /* toggle led */
    at32_led_toggle(LED4);
  }
}
```

■ main 函数代码

```
int main(void)
{
  __IO uint32_t systick_index = 0;
  __IO uint32_t delay_index = 0;

  /* congfig the system clock */
  system_clock_config();

  /* init at start board */
  at32_board_init();
```

```
/* config priority group */
nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

/* enable pwc and bpr clock */
crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

/* congfig the voltage regulator mode.only used with deep sleep mode */
pwc_voltage_regulate_set(PWC_REGULATOR_EXTRA_LOW_POWER);

/* turn on the led light */
at32_led_on(LED2);
at32_led_on(LED3);
at32_led_on(LED4);

/* add a necessary delay to ensure that Vdd is higher than the operating
    voltage of battery powered domain (2.57V) when the battery powered
    domain is powered on for the first time and being operated. */
delay_ms(60);

/* config ertc or other operations of battery powered domain */
ertc_config();

ertc_tamper_config();

while(1)
{
  /* turn off the led light */
  at32_led_off(LED2);

  /* save systick register configuration */
  systick_index = SysTick->CTRL;
  systick_index &= ~((uint32_t)0xFFFFFFFE);

  /* disable systick */
  SysTick->CTRL &= (uint32_t)0xFFFFFFFE;

  /* select system clock source as hick before ldo set */
  crm_sysclk_switch(CRM_SCLK_HICK);

  /* wait till hick is used as system clock source */
  while(crm_sysclk_switch_status_get() != CRM_SCLK_HICK)
  {
  }

  pwc_optimization_level_set(PWC_ECONOMIZE_LEVEL);
```

```
    /* congfig the voltage regulator mode */
    pwc_voltage_regulate_set(PWC_REGULATOR_LOW_POWER);

    /* enter deep sleep mode */
    pwc_deep_sleep_mode_enter(PWC_DEEP_SLEEP_ENTER_WFI);

    /* wake up from deep sleep mode, restore systick register configuration */
    SysTick->CTRL |= systick_index;

    /* turn on the led light */
    at32_led_on(LED2);

    /* wait clock stable */
    delay_us(120);

    /* resume pwc optimization level to normal level before system clock source enhance */
    pwc_optimization_level_set(PWC_NORMAL_LEVEL);

    /* congfig the system clock */
    system_clock_recover();

    delay_ms(500);
  }
}
```

8.4 实验效果

可通过 AT-START BOARD 上的 LED 翻转查看实现效果。

LED2 亮：MCU 处于运行模式；

LED2 灭：MCU 处于深度睡眠模式；

LED4 状态翻转： ERTC 入侵事件中断发生并唤醒了深度睡眠模式。

9 案例 ERTC 周期性唤醒事件唤醒 PWC 深度睡眠模式

9.1 功能简介

本例将演示 PWC 深度睡眠模式的使用。其中，唤醒源使用 ERTC 周期性唤醒事件中断。

9.2 资源准备

5) 硬件环境:

对应产品型号的 AT-START BOARD

6) 软件环境

project\at_start_f423\examples\pwc\deepsleep_ertc_wakeup

9.3 软件设计

5) 配置流程

- 开启 PWC 时钟
- 开启电池供电区域的写入使能
- 等待 VDD 上升到 2.57V 后执行 ERTC 初始化
- 配置 EXINT 线 22 为中断模式并使能
- ERTC 唤醒定时器初始化并使能唤醒定时器中断及其 NVIC 中断
- 设定 ERTC 唤醒定时器值
- 设定电压调节器为节能等级
- 执行进深度睡眠模式命令并等待唤醒
- 唤醒后设定电压调节器为正常等级
- 重新进行系统时钟的恢复设定
- 从“设定 ERTC 唤醒定时器值”开始循环，实现深度睡眠模式的循环唤醒

6) 代码介绍

- ERTC 配置函数代码

```
void ertc_config(void)
{
  /* enable the pwc clock interface */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

  /* allow access to ertc */
  pwc_battery_powered_domain_access(TRUE);

  /* reset ertc domain */
  crm_battery_powered_domain_reset(TRUE);
  crm_battery_powered_domain_reset(FALSE);

  /* enable the lext osc */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_LEXT, TRUE);
```

```
  /* wait till lext is ready */
  while(crm_flag_get(CRM_LEXT_STABLE_FLAG) == RESET);

  /* select the ertc clock source */
  crm_ertc_clock_select(CRM_ERTC_CLOCK_LEXT);

  /* enable the ertc clock */
  crm_ertc_clock_enable(TRUE);

  /* deinitializes the ertc registers */
  ertc_reset();

  /* wait for ertc apb registers synchronisation */
  ertc_wait_update();

  /* configure the ertc data register and ertc prescaler
     ck_spre(1hz) = ertcclk(lext) /(ertc_clk_div_a + 1)*(ertc_clk_div_b + 1)*/
  ertc_divider_set(127, 255);

  /* configure the hour format is 24-hour format*/
  ertc_hour_mode_set(ERTC_HOUR_MODE_24);

  /* set the date: friday june 11th 2021 */
  ertc_date_set(21, 6, 11, 5);

  /* set the time to 06h 20mn 00s am */
  ertc_time_set(6, 20, 0, ERTC_AM);
}
```

■ ERTC 唤醒定时器配置函数代码

```
void ertc_alarm_config(void)
{
  exint_init_type exint_init_struct;

  /* config the exint line of the ertc wakeup timer */
  exint_init_struct.line_select    = EXINT_LINE_22;
  exint_init_struct.line_enable    = TRUE;
  exint_init_struct.line_mode      = EXINT_LINE_INTERRUPUT;
  exint_init_struct.line_polarity = EXINT_TRIGGER_RISING_EDGE;
  exint_init(&exint_init_struct);

  /* set wakeup timer clock 1hz */
  ertc_wakeup_clock_set(ERTC_WAT_CLK_CK_B_16BITS);

  /* enable the ertc interrupt */
  nvic_irq_enable(ERTC_WKUP_IRQn, 0, 0);
```

```
  /* enable ertc wakeup timer a interrupt */
  ertc_interrupt_enable(ERTC_WAT_INT, TRUE);
}
```

■ ERTC 唤醒定时器值设定函数代码

```
void ertc_alarm_value_set(uint32_t alam_index)
{
  /* enable the wakeup timer */
  ertc_wakeup_enable(FALSE);

  while(ertc_flag_get(ERTC_WATWF_FLAG) == RESET);

  /* set the wakeup time */
  ertc_wakeup_counter_set(wakeup_index - 1);

  /* enable the wakeup timer */
  ertc_wakeup_enable(TRUE);
}
```

■ 系统时钟恢复函数代码

```
void system_clock_recover(void)
{
  /* enable external high-speed crystal oscillator - hext */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_HEXT, TRUE);

  /* wait till hext is ready */
  while(crm_hext_stable_wait() == ERROR);

  /* enable pll */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_PLL, TRUE);

  /* wait till pll is ready */
  while(crm_flag_get(CRM_PLL_STABLE_FLAG) == RESET);

  /* enable auto step mode */
  crm_auto_step_mode_enable(TRUE);

  /* select pll as system clock source */
  crm_sysclk_switch(CRM_SCLK_PLL);

  /* wait till pll is used as system clock source */
  while(crm_sysclk_switch_status_get() != CRM_SCLK_PLL);
}
```

■ 中断服务函数代码

```
void ERTC_WKUP_IRQHandler(void)
{
```

```
  if(ertc_flag_get(ERTC_WATF_FLAG) != RESET)
  {
    /* clear ertc wakeup timer flag */
    ertc_flag_clear(ERTC_WATF_FLAG);

    /* clear exint line flag */
    exint_flag_clear(EXINT_LINE_22);

    /* toggle led */
    at32_led_toggle(LED4);
  }
}
```

■ main 函数代码

```
int main(void)
{
  __IO uint32_t systick_index = 0;
  __IO uint32_t delay_index = 0;

  /* congfig the system clock */
  system_clock_config();

  /* init at start board */
  at32_board_init();

  /* config priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* enable pwc and bpr clock */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

  /* congfig the voltage regulator mode.only used with deep sleep mode */
  pwc_voltage_regulate_set(PWC_REGULATOR_EXTRA_LOW_POWER);

  /* turn on the led light */
  at32_led_on(LED2);
  at32_led_on(LED3);
  at32_led_on(LED4);

  /* add a necessary delay to ensure that Vdd is higher than the operating
     voltage of battery powered domain (2.57V) when the battery powered
     domain is powered on for the first time and being operated. */
  delay_ms(60);

  /* config ertc or other operations of battery powered domain */
  ertc_config();
```

```
/* config the wakeup timer   */
ertc_wakeup_timer_config();

while(1)
{
  /* turn off the led light */
  at32_led_off(LED2);

  ertc_wakeup_value_set(5);

  /* save systick register configuration */
  systick_index = SysTick->CTRL;
  systick_index &= ~((uint32_t)0xFFFFFFFE);

  /* disable systick */
  SysTick->CTRL &= (uint32_t)0xFFFFFFFE;

  /* select system clock source as hick before ldo set */
  crm_sysclk_switch(CRM_SCLK_HICK);

  /* wait till hick is used as system clock source */
  while(crm_sysclk_switch_status_get() != CRM_SCLK_HICK)
  {
  }

  pwc_optimization_level_set(PWC_ECONOMIZE_LEVEL);

  /* congfig the voltage regulator mode */
  pwc_voltage_regulate_set(PWC_REGULATOR_LOW_POWER);

  /* enter deep sleep mode */
  pwc_deep_sleep_mode_enter(PWC_DEEP_SLEEP_ENTER_WFI);

  /* wake up from deep sleep mode, restore systick register configuration */
  SysTick->CTRL |= systick_index;

  /* turn on the led light */
  at32_led_on(LED2);

  /* wait clock stable */
  delay_us(120);

  /* resume pwc optimization level to normal level before system clock source enhance */
  pwc_optimization_level_set(PWC_NORMAL_LEVEL);
```

```
    /* congfig the system clock */
    system_clock_recover();

    delay_ms(500);
  }
}
```

9.4 实验效果

可通过 AT-START BOARD 上的 LED 翻转查看实现效果。

LED2 亮：MCU 处于运行模式；

LED2 灭：MCU 处于深度睡眠模式；

LED4 状态翻转： ERTC 唤醒定时器中断发生并唤醒了深度睡眠模式。

10 案例 USART 接收数据唤醒 PWC 深度睡眠模式

10.1 功能简介

本例将演示 PWC 深度睡眠模式的使用。其中，唤醒源使用 USART1 接收数据缓冲器满中断。

10.2 资源准备

7) 硬件环境:

对应产品型号的 AT-START BOARD

8) 软件环境

project\at_start_f423\examples\pwc\deepsleep_usart1

10.3 软件设计

7) 配置流程

- 开启 PWC 时钟
- 等待 VDD 上升到 2.57V 后执行 USART1 基础配置并使能接收数据缓冲器满中断
- 开启 deepsleep 模式下的 USART 使能位
- 选择低功耗唤醒方式为 RDBF
- 使能 USART 的低功耗唤醒中断
- 配置 EXINT 线 25 为中断模式并使能
- 等待 USART 接收器空闲
- 设定电压调节器为节能等级
- 执行进深度睡眠模式命令并等待唤醒
- 唤醒后设定电压调节器为正常等级
- 重新进行系统时钟的恢复设定
- 从“等待 USART 接收器空闲”开始循环，实现深度睡眠模式的循环唤醒

8) 代码介绍

- USART1 基础配置函数代码

```
void usart1_config(uint32_t baudrate)
{
  gpio_init_type gpio_init_struct;

  /* allow access to ertc */
  pwc_battery_powered_domain_access(TRUE);

  /* reset ertc domain */
  crm_battery_powered_domain_reset(TRUE);
  crm_battery_powered_domain_reset(FALSE);

  /* enable the lext osc */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_LEXT, TRUE);
```

```
  /* wait till lext is ready */
  while(crm_flag_get(CRM_LEXT_STABLE_FLAG) == RESET);

  crm_usart_clock_select(CRM_USART_INDEX_1, CRM_USART_CLOCK_SOURCE_LEXT);

  /* enable the uart1 and gpio clock */
  crm_periph_clock_enable(CRM_USART1_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_SCFG_PERIPH_CLOCK, TRUE);

  gpio_default_para_init(&gpio_init_struct);

  /* configure the uart1 tx pin */
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type       = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_mode           = GPIO_MODE_MUX;
  gpio_init_struct.gpio_pull           = GPIO_PULL_NONE;
  gpio_init_struct.gpio_pins           = GPIO_PINS_9;
  gpio_init(GPIOA, &gpio_init_struct);

  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type       = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_mode           = GPIO_MODE_MUX;
  gpio_init_struct.gpio_pull           = GPIO_PULL_NONE;
  gpio_init_struct.gpio_pins           = GPIO_PINS_10;
  gpio_init(GPIOA, &gpio_init_struct);

  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE9, GPIO_MUX_7);
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE10, GPIO_MUX_7);

  /* configure uart param */
  nvic_irq_enable(USART1_IRQn, 0, 0);

  usart_init(USART1, baudrate, USART_DATA_8BITS, USART_STOP_1_BIT);
  usart_parity_selection_config(USART1, USART_PARITY_NONE);
  usart_transmitter_enable(USART1, TRUE);
  usart_receiver_enable(USART1, TRUE);
  usart_hardware_flow_control_set(USART1,USART_HARDWARE_FLOW_NONE);
  usart_interrupt_enable(USART1, USART_RDBF_INT, TRUE);
  usart_enable(USART1, TRUE);

  /* polling usart initialisation */
  while((!(usart_flag_get(USART1, USART_TXON_FLAG))) || (!(usart_flag_get(USART1,
USART_RXON_FLAG))))
  {
  }
```

```
}
```

■ USART1 低功耗唤醒相关函数代码

```
void usart1_wakeup_config(void)
{
  exint_init_type exint_init_struct;

  /* keep usart work when deepsleep */
  usart_deep_sleep_mode_enable(USART1, TRUE);

  /* low power wakeup method is receive data buffer full */
  usart_low_power_wakeup_set(USART1, USART_WAKEUP_METHOD_RDBF);

  usart_interrupt_enable(USART1, USART_LPWUF_INT, TRUE);

  /* config the exint line of the usart1 */
  exint_init_struct.line_select    = EXINT_LINE_25;
  exint_init_struct.line_enable    = TRUE;
  exint_init_struct.line_mode      = EXINT_LINE_INTERRUPUT;
  exint_init_struct.line_polarity = EXINT_TRIGGER_RISING_EDGE;
  exint_init(&exint_init_struct);
}
```

■ 系统时钟恢复函数代码

```
void system_clock_recover(void)
{
  /* enable external high-speed crystal oscillator - hext */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_HEXT, TRUE);

  /* wait till hext is ready */
  while(crm_hext_stable_wait() == ERROR);

  /* enable pll */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_PLL, TRUE);

  /* wait till pll is ready */
  while(crm_flag_get(CRM_PLL_STABLE_FLAG) == RESET);

  /* enable auto step mode */
  crm_auto_step_mode_enable(TRUE);

  /* select pll as system clock source */
  crm_sysclk_switch(CRM_SCLK_PLL);

  /* wait till pll is used as system clock source */
  while(crm_sysclk_switch_status_get() != CRM_SCLK_PLL);
}
```

■ 中断服务函数代码

```
void USART1_IRQHandler (void)
{
  if(usart_flag_get(USART1, USART_RDBF_FLAG) != RESET)
  {
    /* clear rdbf flag */
    usart1_index = usart_data_receive(USART1);

    /* toggle led */
    at32_led_toggle(LED4);
  }

  if(usart_flag_get(USART1, USART_LPWUF_FLAG) != RESET)
  {
    usart_flag_clear(USART1, USART_LPWUF_FLAG);
  }

  if(exint_flag_get(EXINT_LINE_25) != RESET)
  {
    exint_flag_clear(EXINT_LINE_25);
  }
}
```

■ main 函数代码

```
int main(void)
{
  __IO uint32_t index = 0;
  __IO uint32_t systick_index = 0;

  /* enable pwc clock */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

  /* congfig the voltage regulator mode.only used with deep sleep mode */
  pwc_voltage_regulate_set(PWC_REGULATOR_EXTRA_LOW_POWER);

  /* congfig the system clock */
  system_clock_config();

  /* init at start board */
  at32_board_init();

  /* turn on the led light */
  at32_led_on(LED2);
  at32_led_on(LED3);
  at32_led_on(LED4);
```

```
/* add a necessary delay to ensure that Vdd is higher than the operating
    voltage of battery powered domain (2.57V) when the battery powered
    domain is powered on for the first time and being operated. */
delay_ms(60);

/* config usart or other operations of battery powered domain */
usart1_config(2400);
usart1_wakeup_config();

printf("exit deepsleep mode by usart1 rdbf interrupt \r\n");
while(1)
{
  at32_led_off(LED2);
  printf("now enter deepsleep mode \r\n");

  /* make sure that no usart receiver is ongoing */
  while(usart_flag_get(USART1, USART_OCCUPY_FLAG) == SET)
  {
  }

  /* select system clock source as hick before ldo set */
  crm_sysclk_switch(CRM_SCLK_HICK);

  /* wait till hick is used as system clock source */
  while(crm_sysclk_switch_status_get() != CRM_SCLK_HICK)
  {
  }

  /* reduce pwc optimization level to economize level before enter deep sleep */
  pwc_optimization_level_set(PWC_ECONOMIZE_LEVEL);

  while(usart_flag_get(USART1, USART_TDC_FLAG) == RESET)
  {
  }

  /* enter deep sleep mode */
  pwc_deep_sleep_mode_enter(PWC_DEEP_SLEEP_ENTER_WFI);

  /* turn on the led light */
  at32_led_on(LED2);

  /* wait clock stable */
  delay_us(120);

  /* resume pwc optimization level to normal level before system clock source enhance */
  pwc_optimization_level_set(PWC_NORMAL_LEVEL);
```

```
    /* wake up from deep sleep mode, congfig the system clock */
    system_clock_recover();

    /* wake up from sleep mode */
    printf("\r\nnow exit deepsleep mode by usart1 rdbf interrupt \r\n");
    printf("usart1_rdne_data = 0x%x\r\n", usart1_index);
    delay_ms(300);
  }
}
```

10.4 实验效果

可通过 AT-START BOARD 上的 LED 翻转查看实现效果。

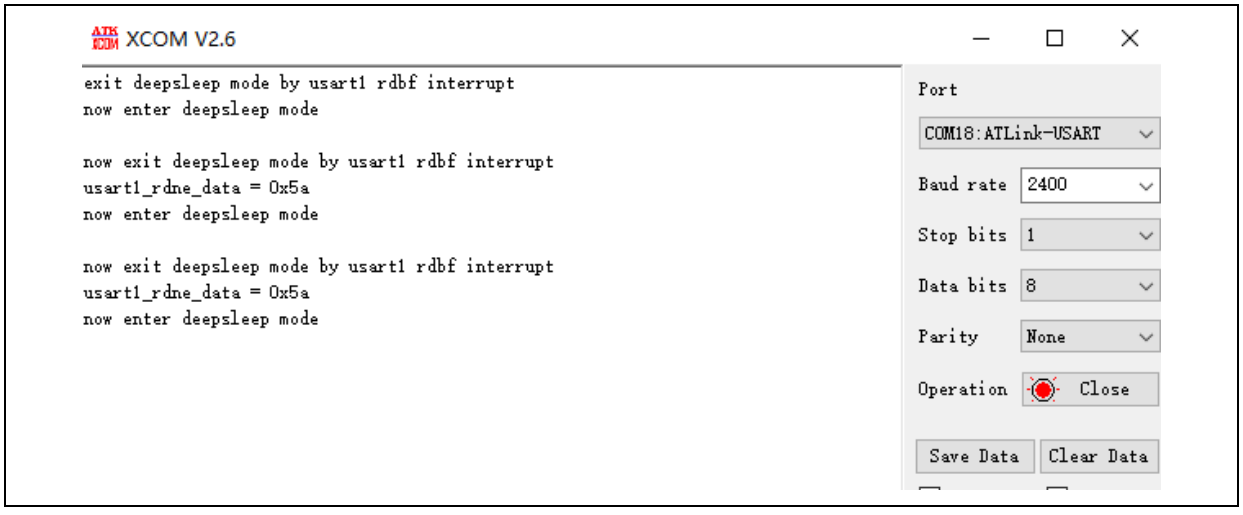
LED2 亮：MCU 处于运行模式；

LED2 灭：MCU 处于深度睡眠模式；

LED4 状态翻转： USART1 接收缓冲器满中断发生并唤醒了深度睡眠模式。

也可通过 USART1 的串口打印查看实验效果，如下：

图 9. 案例 10 测试输出



11 案例 I2C 地址匹配事件唤醒 PWC 深度睡眠模式

11.1 功能简介

本例将演示 PWC 深度睡眠模式的使用。其中，唤醒源使用 I2C 地址匹配事件。

11.2 资源准备

9) 硬件环境:

对应产品型号的 AT-START BOARD

10) 软件环境

project\at_start_f423\examples\pwc\deepsleep_i2c

11.3 软件设计

9) 配置流程

- I2C1 相关 IO 初始化及 I2C1 基础配置
- I2C1 低功耗相关配置
- 配置 EXINT 线 23 为中断模式并使能
- 使能 I2C1
- 选择 I2C1 时钟源
- I2C 从机等待接收数据
- 设定电压调节器为节能等级
- 执行进深度睡眠模式命令并等待唤醒（由 I2C1 主机发送匹配地址唤醒）
- 唤醒后设定电压调节器为正常等级
- 重新进行系统时钟的恢复设定
- I2C1 从机等待数据接收完成
- I2C1 从机等待发送数据
- 设定电压调节器为节能等级
- 执行进深度睡眠模式命令并等待唤醒（由 I2C1 主机发送匹配地址唤醒）
- 唤醒后设定电压调节器为正常等级
- 重新进行系统时钟的恢复设定
- I2C1 从机等待数据发送完成
- 从“选择 I2C1 时钟源”开始循环，实现深度睡眠模式的循环唤醒

10) 代码介绍

- I2C1 配置函数代码

```
void i2c_lowlevel_init(i2c_handle_type* hi2c)
{
  gpio_init_type gpio_init_structure;
  exint_init_type exint_init_struct;

  if(hi2c->i2cx == I2C1_PORT)
  {
```

```
    exint_init_struct.line_select          = EXINT_LINE_23;
    exint_init_struct.line_enable          = TRUE;
    exint_init_struct.line_polarity        = EXINT_TRIGGER_RISING_EDGE;
    exint_init_struct.line_mode            = EXINT_LINE_INTERRUPUT;
    exint_init(&exint_init_struct);

    /* pwc periph clock enable */
    crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

    /* i2c periph clock enable */
    crm_periph_clock_enable(I2Cx_CLK, TRUE);
    crm_periph_clock_enable(I2Cx_SCL_GPIO_CLK, TRUE);
    crm_periph_clock_enable(I2Cx_SDA_GPIO_CLK, TRUE);

    /* gpio configuration */
    gpio_pin_mux_config(I2Cx_SCL_GPIO_PORT, I2Cx_SCL_GPIO_PinsSource,
I2Cx_SCL_GPIO_MUX);

    gpio_pin_mux_config(I2Cx_SDA_GPIO_PORT, I2Cx_SDA_GPIO_PinsSource,
I2Cx_SDA_GPIO_MUX);

    /* configure i2c pins: scl */
    gpio_init_structure.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
    gpio_init_structure.gpio_mode          = GPIO_MODE_MUX;
    gpio_init_structure.gpio_out_type      = GPIO_OUTPUT_OPEN_DRAIN;
    gpio_init_structure.gpio_pull          = GPIO_PULL_UP;

    gpio_init_structure.gpio_pins          = I2Cx_SCL_GPIO_PIN;
    gpio_init(I2Cx_SCL_GPIO_PORT, &gpio_init_structure);

    /* configure i2c pins: sda */
    gpio_init_structure.gpio_pins          = I2Cx_SDA_GPIO_PIN;
    gpio_init(I2Cx_SDA_GPIO_PORT, &gpio_init_structure);

    /* configure and enable i2c interrupt */
    nvic_irq_enable(I2C1_EVT_IRQn, 0, 0);

    /* config i2c, the digital filter must be 0 in deepsleep mode */
    i2c_init(hi2c->i2cx, 0x00, I2Cx_CLKCTRL);

    i2c_own_address1_set(hi2c->i2cx, I2C_ADDRESS_MODE_7BIT, I2Cx_ADDRESS);

    i2c_wakeup_enable(hi2c->i2cx, TRUE);
  }
}
```

■ 通讯错误处理函数代码

```
void error_handler(uint32_t error_code)
{
  while(1)
  {
    at32_led_toggle(LED2);
    delay_ms(500);
  }
}
```

■ 接收数据比较函数代码

```
uint32_t buffer_compare(uint8_t* buffer1, uint8_t* buffer2, uint32_t len)
{
  uint32_t i;

  for(i = 0; i < len; i++)
  {
    if(buffer1[i] != buffer2[i])
    {
      return 1;
    }
  }

  return 0;
}
```

■ 系统时钟恢复函数代码

```
void system_clock_recover(void)
{
  /* enable external high-speed crystal oscillator - hext */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_HEXT, TRUE);

  /* wait till hext is ready */
  while(crm_hext_stable_wait() == ERROR);

  /* enable pll */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_PLL, TRUE);

  /* wait till pll is ready */
  while(crm_flag_get(CRM_PLL_STABLE_FLAG) == RESET);

  /* enable auto step mode */
  crm_auto_step_mode_enable(TRUE);

  /* select pll as system clock source */
  crm_sysclk_switch(CRM_SCLK_PLL);

  /* wait till pll is used as system clock source */
```

```
  while(crm_sysclk_switch_status_get() != CRM_SCLK_PLL);
}
```

■ 中断服务函数代码

```
void I2C1_EVT_IRQHandler(void)
{
  if(exint_flag_get(EXINT_LINE_23) != RESET)
  {
    exint_flag_clear(EXINT_LINE_23);
  }

  i2c_evt_irq_handler(&hi2cx);
}

void I2C1_ERR_IRQHandler(void)
{
  i2c_err_irq_handler(&hi2cx);
}
```

■ main 函数代码

```
#define I2C_TIMEOUT                     0xFFFFFFFF

/* setting i2c clock frequency */
//#define I2Cx_CLKCTRL                  0x7170F7F7    //10K
//#define I2Cx_CLKCTRL                  0xA0F06767    //50K
#define I2Cx_CLKCTRL                    0xA0F03131    //100K
//#define I2Cx_CLKCTRL                  0x30D03259    //200K
#define I2Cx_ADDRESS                    0xA0

#define I2Cx_PORT                       I2C1
#define I2Cx_CLK                        CRM_I2C1_PERIPH_CLOCK

#define I2Cx_SCL_GPIO_CLK               CRM_GPIOB_PERIPH_CLOCK
#define I2Cx_SCL_GPIO_PIN               GPIO_PINS_6
#define I2Cx_SCL_GPIO_PinsSource        GPIO_PINS_SOURCE6
#define I2Cx_SCL_GPIO_PORT              GPIOB
#define I2Cx_SCL_GPIO_MUX               GPIO_MUX_1

#define I2Cx_SDA_GPIO_CLK               CRM_GPIOB_PERIPH_CLOCK
#define I2Cx_SDA_GPIO_PIN               GPIO_PINS_7
#define I2Cx_SDA_GPIO_PinsSource        GPIO_PINS_SOURCE7
#define I2Cx_SDA_GPIO_PORT              GPIOB
#define I2Cx_SDA_GPIO_MUX               GPIO_MUX_1

#define I2Cx_IRQn                       I2C1_IRQn

#define BUF_SIZE                        8
```

```
//#define MASTER_BOARD

uint8_t tx_buf[BUF_SIZE] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
uint8_t rx_buf[BUF_SIZE] = {0};

i2c_handle_type hi2cx;

int main(void)
{
  i2c_status_type i2c_status;

  /* initial system clock */
  system_clock_config();

  /* at board initial */
  at32_board_init();

  hi2cx.i2cx = I2Cx_PORT;

  /* i2c config */
  i2c_config(&hi2cx);

  while(1)
  {

  #if defined (MASTER_BOARD)

    /* wait for key USER_BUTTON press before starting the communication */
    while(at32_button_press() != USER_BUTTON)
    {
    }

    crm_i2c1_clock_select(CRM_I2C_CLOCK_SOURCE_PCLK);

    /* start the request reception process */
    if((i2c_status = i2c_master_transmit_int(&hi2cx, I2Cx_ADDRESS, tx_buf, BUF_SIZE,
I2C_TIMEOUT)) != I2C_OK)
    {
      error_handler(i2c_status);
    }

    /* wait for the communication to end */
    if(i2c_wait_end(&hi2cx, I2C_TIMEOUT) != I2C_OK)
    {
      error_handler(i2c_status);
    }
```

```
    delay_ms(10);

    /* start the request reception process */
    if((i2c_status = i2c_master_receive_int(&hi2cx, I2Cx_ADDRESS, rx_buf, BUF_SIZE,
I2C_TIMEOUT)) != I2C_OK)
    {
      error_handler(i2c_status);
    }

    /* wait for the communication to end */
    if(i2c_wait_end(&hi2cx, I2C_TIMEOUT) != I2C_OK)
    {
      error_handler(i2c_status);
    }

    if(buffer_compare(tx_buf, rx_buf, BUF_SIZE) == 0)
    {
      at32_led_on(LED3);
    }
    else
    {
      error_handler(i2c_status);
    }

  #else

    /* wait for key USER_BUTTON press before starting the communication */
    while(at32_button_press() != USER_BUTTON)
    {
    }

    crm_hick_sclk_frequency_select(CRM_HICK_SCLK_48MHZ);
    crm_i2c1_clock_select(CRM_I2C_CLOCK_SOURCE_HICK);

    /* start the transmission process */
    if((i2c_status = i2c_slave_receive_int(&hi2cx, rx_buf, BUF_SIZE, I2C_TIMEOUT)) != I2C_OK)
    {
      error_handler(i2c_status);
    }

    /* select system clock source as hick before ldo set */
    crm_sysclk_switch(CRM_SCLK_HICK);

    /* wait till hick is used as system clock source */
    while(crm_sysclk_switch_status_get() != CRM_SCLK_HICK)
```

```
{
}

/* reduce pwc optimization level to economize level before enter deep sleep */
pwc_optimization_level_set(PWC_ECONOMIZE_LEVEL);

/* enter deep sleep mode */
pwc_deep_sleep_mode_enter(PWC_DEEP_SLEEP_ENTER_WFI);

/* wait clock stable */
delay_us(120);

/* resume pwc optimization level to normal level before system clock source enhance */
pwc_optimization_level_set(PWC_NORMAL_LEVEL);

/* wake up from deep sleep mode, congfig the system clock */
system_clock_recover();

/* wait for the communication to end */
if(i2c_wait_end(&hi2cx, I2C_TIMEOUT) != I2C_OK)
{
  error_handler(i2c_status);
}

if((i2c_status = i2c_slave_transmit_int(&hi2cx, tx_buf, BUF_SIZE, I2C_TIMEOUT)) != I2C_OK)
{
  error_handler(i2c_status);
}

/* select system clock source as hick before ldo set */
crm_sysclk_switch(CRM_SCLK_HICK);

/* wait till hick is used as system clock source */
while(crm_sysclk_switch_status_get() != CRM_SCLK_HICK)
{
}

/* reduce pwc optimization level to economize level before enter deep sleep */
pwc_optimization_level_set(PWC_ECONOMIZE_LEVEL);

/* enter deep sleep mode */
pwc_deep_sleep_mode_enter(PWC_DEEP_SLEEP_ENTER_WFI);

/* wait clock stable */
delay_us(120);
```

```
    /* resume pwc optimization level to normal level before system clock source enhance */
    pwc_optimization_level_set(PWC_NORMAL_LEVEL);

    /* wake up from deep sleep mode, congfig the system clock */
    system_clock_recover();

    /* wait for the communication to end */
    if(i2c_wait_end(&hi2cx, I2C_TIMEOUT) != I2C_OK)
    {
      error_handler(i2c_status);
    }

    if(buffer_compare(tx_buf, rx_buf, BUF_SIZE) == 0)
    {
      at32_led_on(LED3);
    }
    else
    {
      error_handler(i2c_status);
    }
  #endif
  }
}
```

11.4 实验效果

可通过 AT-START BOARD 上的 LED 翻转查看实现效果。

LED3 亮： I2C1 通讯正常且能唤醒深度睡眠模式。

LED2 闪烁：I2C1 通讯异常。

12 案例 设定内部电压调节器功耗等级

12.1 功能简介

本例将演示 PWC 内部电压调节器不同功耗等级的使用。其中，用户需要遵照图 5 的“电压调节器不同功耗等级的使用限定”来进行应用设计。

12.2 资源准备

11) 硬件环境:

对应产品型号的 AT-START BOARD

12) 软件环境

project\at_start_f423\examples\pwc\power_optimization_level_set

12.3 软件设计

11) 配置流程

- 开启 PWC 时钟
- 开启电池供电区域的写入使能
- 等待 VDD 上升到 2.57V 后执行 ERTC 初始化
- 配置 EXINT 线 21 为中断模式并使能
- ERTC 入侵事件检测初始化并使能入侵事件检测中断及其 NVIC 中断
- 设定电压调节器为节能等级
- 执行进深度睡眠模式命令并等待唤醒
- 唤醒后设定电压调节器为正常等级
- 重新进行系统时钟的恢复设定
- 从“设定电压调节器为节能等级”开始循环，实现深度睡眠模式的循环唤醒

12) 代码介绍

- ERTC 配置函数代码

```
void ertc_config(void)
{
  /* enable the pwc clock interface */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

  /* allow access to ertc */
  pwc_battery_powered_domain_access(TRUE);

  /* reset ertc domain */
  crm_battery_powered_domain_reset(TRUE);
  crm_battery_powered_domain_reset(FALSE);

  /* enable the lext osc */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_LEXT, TRUE);

  /* wait till lext is ready */
```

```
  while(crm_flag_get(CRM_LEXT_STABLE_FLAG) == RESET);

  /* select the ertc clock source */
  crm_ertc_clock_select(CRM_ERTC_CLOCK_LEXT);

  /* enable the ertc clock */
  crm_ertc_clock_enable(TRUE);

  /* deinitializes the ertc registers */
  ertc_reset();
}
```

■ ERTC 入侵事件配置函数代码

```
void ertc_tamper_config (void)
{
  exint_init_type exint_init_struct;

  /* config the exint line of the ertc tamper */
  exint_init_struct.line_select     = EXINT_LINE_21;
  exint_init_struct.line_enable     = TRUE;
  exint_init_struct.line_mode       = EXINT_LINE_INTERRUPUT;
  exint_init_struct.line_polarity = EXINT_TRIGGER_RISING_EDGE;
  exint_init(&exint_init_struct);

  /* enable tamper irqchannel */
  nvic_irq_enable(TAMP_STAMP_IRQn, 0, 0);

  /* disable the tamper 1 detection */
  ertc_tamper_enable(ERTC_TAMPER_1, FALSE);

  /* clear tamper 1 pin event(tamp1f) pending flag */
  ertc_flag_clear(ERTC_TP1F_FLAG);

  /* configure the tamper 1 trigger */
  ertc_tamper_valid_edge_set(ERTC_TAMPER_1, ERTC_TAMPER_EDGE_RISING);

  /* enable the tamper interrupt */
  ertc_interrupt_enable(ERTC_TP_INT, TRUE);

  /* enable the tamper 1 detection */
  ertc_tamper_enable(ERTC_TAMPER_1, TRUE);
}
```

■ 系统时钟恢复函数代码

```
void system_clock_recover(void)
{
  /* enable external high-speed crystal oscillator - hext */
```

```
  crm_clock_source_enable(CRM_CLOCK_SOURCE_HEXT, TRUE);

  /* wait till hext is ready */
  while(crm_hext_stable_wait() == ERROR);

  /* enable pll */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_PLL, TRUE);

  /* wait till pll is ready */
  while(crm_flag_get(CRM_PLL_STABLE_FLAG) == RESET);

  /* enable auto step mode */
  crm_auto_step_mode_enable(TRUE);

  /* select pll as system clock source */
  crm_sysclk_switch(CRM_SCLK_PLL);

  /* wait till pll is used as system clock source */
  while(crm_sysclk_switch_status_get() != CRM_SCLK_PLL);
}
```

■ 中断服务函数代码

```
void TAMP_STAMP_IRQHandler(void)
{
  if(ertc_flag_get(ERTC_TP1F_FLAG) != RESET)
  {
    /* clear ertc alarm flag */
    ertc_flag_clear(ERTC_TP1F_FLAG);

    /* clear exint line flag */
    exint_flag_clear(EXINT_LINE_21);

    /* toggle led */
    at32_led_toggle(LED4);
  }
}
```

■ main 函数代码

```
int main(void)
{
  __IO uint32_t systick_index = 0;
  __IO uint32_t delay_index = 0;

  /* congfig the system clock */
  system_clock_config();

  /* init at start board */
```

```
at32_board_init();

/* config priority group */
nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

/* turn on the led light */
at32_led_on(LED2);
at32_led_on(LED3);
at32_led_on(LED4);

/* enable pwc clock */
crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

/* congfig the voltage regulator mode.only used with deep sleep mode */
pwc_voltage_regulate_set(PWC_REGULATOR_EXTRA_LOW_POWER);

/* add a necessary delay to ensure that Vdd is higher than the operating
    voltage of battery powered domain (2.57V) when the battery powered
    domain is powered on for the first time and being operated. */
delay_ms(60);

/* config ertc or other operations of battery powered domain */   ertc_config();

/* set the wakeup time: 06h:20min:5s */
ertc_tamper_config();

while(1)
{
  /* turn off the led light */
  at32_led_off(LED2);

  /* save systick register configuration */
  systick_index = SysTick->CTRL;
  systick_index &= ~((uint32_t)0xFFFFFFFE);

  /* disable systick */
  SysTick->CTRL &= (uint32_t)0xFFFFFFFE;

  /* select system clock source as hick before ldo set */
  crm_sysclk_switch(CRM_SCLK_HICK);

  /* wait till hick is used as system clock source */
  while(crm_sysclk_switch_status_get() != CRM_SCLK_HICK)
  {
  }
```

```
    pwc_optimization_level_set(PWC_ECONOMIZE_LEVEL);

    /* congfig the voltage regulator mode */
    pwc_voltage_regulate_set(PWC_REGULATOR_LOW_POWER);

    /* enter deep sleep mode */
    pwc_deep_sleep_mode_enter(PWC_DEEP_SLEEP_ENTER_WFI);

    /* wake up from deep sleep mode, restore systick register configuration */
    SysTick->CTRL |= systick_index;

    /* turn on the led light */
    at32_led_on(LED2);

    /* wait clock stable */
    delay_us(120);

    /* resume pwc optimization level to normal level before system clock source enhance */
    pwc_optimization_level_set(PWC_NORMAL_LEVEL);

    /* congfig the system clock */
    system_clock_recover();

    delay_ms(500);
    }
}
```

12.4 实验效果

可通过 AT-START BOARD 上的 LED 翻转查看实现效果。

LED2 亮：MCU 处于运行模式，此时内部电压调节器处于正常等级；

LED2 灭：MCU 处于深度睡眠模式，此时内部电压调节器处于节能等级；

LED4 状态翻转： ERTC 入侵事件中断发生并唤醒了深度睡眠模式。

13 案例 ERTC 闹钟唤醒 PWC 待机模式

13.1 功能简介

本例将演示 PWC 待机模式的使用。其中，唤醒源使用 ERTC 闹钟中断。

13.2 资源准备

13) 硬件环境:

对应产品型号的 AT-START BOARD

14) 软件环境

project\at_start_f423\examples\pwc\standby_ertc_alarm

13.3 软件设计

13) 配置流程

- 开启 PWC 时钟
- 开启电池供电区域的写入使能
- 判断并清除已置位的“待机模式标志”及“待机唤醒事件标志”
- 等待 VDD 上升到 2.57V 后执行 ERTC 初始化
- ERTC 闹钟初始化并使能闹钟中断
- 设定 ERTC 闹钟值
- 执行进待机模式命令并等待唤醒

14) 代码介绍

- ERTC 配置函数代码

```
void ertc_config(void)
{
  /* enable the pwc clock interface */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

  /* allow access to ertc */
  pwc_battery_powered_domain_access(TRUE);

  /* reset ertc domain */
  crm_battery_powered_domain_reset(TRUE);
  crm_battery_powered_domain_reset(FALSE);

  /* enable the lext osc */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_LEXT, TRUE);

  /* wait till lext is ready */
  while(crm_flag_get(CRM_LEXT_STABLE_FLAG) == RESET);

  /* select the ertc clock source */
  crm_ertc_clock_select(CRM_ERTC_CLOCK_LEXT);
```

```
  /* enable the ertc clock */
  crm_ertc_clock_enable(TRUE);

  /* deinitializes the ertc registers */
  ertc_reset();

  /* wait for ertc apb registers synchronisation */
  ertc_wait_update();

  /* configure the ertc data register and ertc prescaler
     ck_spre(1hz) = ertcclk(lext) /(ertc_clk_div_a + 1)*(ertc_clk_div_b + 1)*/
  ertc_divider_set(127, 255);

  /* configure the hour format is 24-hour format*/
  ertc_hour_mode_set(ERTC_HOUR_MODE_24);

  /* set the date: friday june 11th 2021 */
  ertc_date_set(21, 6, 11, 5);

  /* set the time to 06h 20mn 00s am */
  ertc_time_set(6, 20, 0, ERTC_AM);
}
```

■ ERTC 闹钟配置函数代码

```
void ertc_alarm_config(void)
{
  /* set the alarm 05h:20min:10s */
  ertc_alarm_mask_set(ERTC_ALA, ERTC_ALARM_MASK_DATE_WEEK | ERTC_ALARM_MASK_HOUR
| ERTC_ALARM_MASK_MIN);
  ertc_alarm_week_date_select(ERTC_ALA, ERTC_SLECT_DATE);

  /* enable ertc alarm a interrupt */
  ertc_interrupt_enable(ERTC_ALA_INT, TRUE);
}
```

■ ERTC 闹钟值设定函数代码

```
void ertc_alarm_value_set(uint32_t alam_index)
{
  ertc_time_type ertc_time_struct;

  /* disable the alarm */
  ertc_alarm_enable(ERTC_ALA, FALSE);
  ertc_calendar_get(&ertc_time_struct);
  ertc_alarm_set(ERTC_ALA, ertc_time_struct.day, ertc_time_struct.hour, ertc_time_struct.min,
(ertc_time_struct.sec + alam_index)% 60, ertc_time_struct.ampm);
```

```
  /* disable the alarm */
  ertc_alarm_enable(ERTC_ALA, TRUE);
}
```

■ main 函数代码

```
int main(void)
{
  __IO uint32_t index = 0;

  /* congfig the system clock */
  system_clock_config();

  /* init at start board */
  at32_board_init();

  /* enable pwc and bpr clock */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

  if(pwc_flag_get(PWC_STANDBY_FLAG) != RESET)
  {
    /* wakeup from standby */
    pwc_flag_clear(PWC_STANDBY_FLAG);
    at32_led_on(LED2);
  }
  if(pwc_flag_get(PWC_WAKEUP_FLAG) != RESET)
  {
    /* wakeup event occurs */
    pwc_flag_clear(PWC_WAKEUP_FLAG);
    at32_led_on(LED3);
  }
  /* delay to check led state */
  delay_sec(1);

  /* add a necessary delay to ensure that Vdd is higher than the operating
     voltage of battery powered domain (2.57V) when the battery powered
     domain is powered on for the first time and being operated. */
  delay_ms(60);

  /* config ertc or other operations of battery powered domain */
  ertc_config();
  at32_led_on(LED4);
  ertc_alarm_config();

  /* delay to check led state */
  delay_sec(1);
```

```
  ertc_alarm_value_set(3);
  /* enter standby mode */
  pwc_standby_mode_enter();
  while(1)
  {
  }
}
```

13.4 实验效果

可通过 AT-START BOARD 上的 LED 翻转查看实现效果。

LED4 亮：MCU 处于运行模式；

LED4 灭：MCU 处于待机模式；

MCU 运行状态下的 LED2： 亮表示进入待机模式标志置位，反之未置位；

MCU 运行状态下的 LED3： 亮表示待机唤醒事件标志置位，反之未置位

14 案例 PWC 待机模式

14.1 功能简介

本例将演示 PWC 待机模式的使用。其中，唤醒源使用 WKUPx 管脚上升沿。

14.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

USER_KEY1——PA0

2) 软件环境

project\at_start_f423\examples\ pwc\standby_wakeup_pin

14.3 软件设计

1) 配置流程

- 开启 PWC 时钟
- 检测进入待机模式标志并清除
- 检测待机唤醒事件标志并清除
- 使能 WKUP1 引脚
- 执行进待机模式命令并等待唤醒

2) 代码介绍

- main 函数代码

```
int main(void)
{
  __IO uint32_t index = 0;

  /* congfig the system clock */
  system_clock_config();

  /* init at start board */
  at32_board_init();

  /* config priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* turn on the led light */
  at32_led_off(LED2);
  at32_led_off(LED3);
  at32_led_off(LED4);

  /* enable pwc clock */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);
```

```
  if(pwc_flag_get(PWC_STANDBY_FLAG) != RESET)
  {
    /* wakeup from standby */
    pwc_flag_clear(PWC_STANDBY_FLAG);
    at32_led_on(LED2);
  }

  if(pwc_flag_get(PWC_WAKEUP_FLAG) != RESET)
  {
    /* wakeup event occurs */
    pwc_flag_clear(PWC_WAKEUP_FLAG);
    at32_led_on(LED3);
  }

  at32_led_on(LED4);
  delay_ms(1000);

  /* enable wakeup pin1 */
  pwc_wakeup_pin_enable(PWC_WAKEUP_PIN_1, TRUE);

  /* enter standby mode */
  pwc_standby_mode_enter();

  while(1)
  {
  }
}
```

14.4 实验效果

可通过 AT-START BOARD 上的 USER_KEY 来唤醒，LED 翻转查看实现效果。

LED4 亮：MCU 处于运行模式；

LED4 灭：MCU 处于待机模式；

MCU 运行状态下的 LED2： 亮表示进入待机模式标志置位，反之未置位；

MCU 运行状态下的 LED3： 亮表示待机唤醒事件标志置位，反之未置位。

15 文档版本历史

表 6. 文档版本历史

日期	版本	变更
2023.03.23	2.0.0	最初版本
2023.04.21	2.0.1	更新“深度睡眠模式下的典型电流消耗表”及“深度睡眠和待机模式下的典型电流消耗表”
2023.08.21	2.0.2	更新“案例 PWC待机模式”的部分源码

重要通知 - 请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和/或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途(及其依据任何司法管辖区的法律的对应情况)，或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力产品并非设计或专门用于下列用途的产品：(A) 对安全性有特别要求的应用，例如：生命支持、主动植入设备或对产品功能安全有要求的系统；(B) 航空应用；(C) 航天应用或航天环境；(D) 武器，且/或(E)其他可能导致人身伤害、死亡及财产损害的应用。如果采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，风险及法律责任仍将由采购商单独承担，且采购商应独力负责在前述应用中满足所有法律和法规要求。

经销的雅特力产品如有不同于本文档中提出的声明和/或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。