Application Note

MSPM0L222X 中的低频子系统和 VBAT 特性

Hoa Tang *Mixed-Signal Processors*

摘要

低频子系统 (LFSS) 是 MSPM0Lx22x 器件系列的一项新功能集，具有实时时钟 (RTC)、低频晶体 (LFXT)、低频振荡器 (LFOSC) 和便笺式存储器 (SPM)。LFSS 由名为 *VBAT 电源域* 的单独电源域供电，因此允许 LFSS 中的外设在主 VDD 电源丢失时继续运行。本应用手册将讨论如何在应用中使用 LFSS 和 VBAT 域。请按以下地址在 SDK 下查找所用的示例：
`C:\ti\mspm0_sdk_x_xx_xx_xx_internal\examples\nortos\LP_MSPM0L2228\driverlib`。

内容

插图清单

表格清单

商标

1 引言

低频子系统 (LFSS) 在子系统中将多个功能外设组合在一起。在 LFSS 知识产权 (IP) 中，所有功能外设均由低频时钟 (LFCLK) 计时，该时钟在低功耗模式期间激活。电池备份域最初由电源输入 VDD 供电，并通过从电池获取电力来补偿功率损耗，以便保持功能外设以 32kHz 的频率运行，旨在实现长期计时。以下各节简要介绍了每个 LFSS 外设。另请参阅 *MSPM0 L 系列 32MHz 微控制器技术参考手册*。

2 低频子系统简介

图 2-1 展示了 LFSS 的方框图。本文档将全面介绍 LFSS 的每个部分。

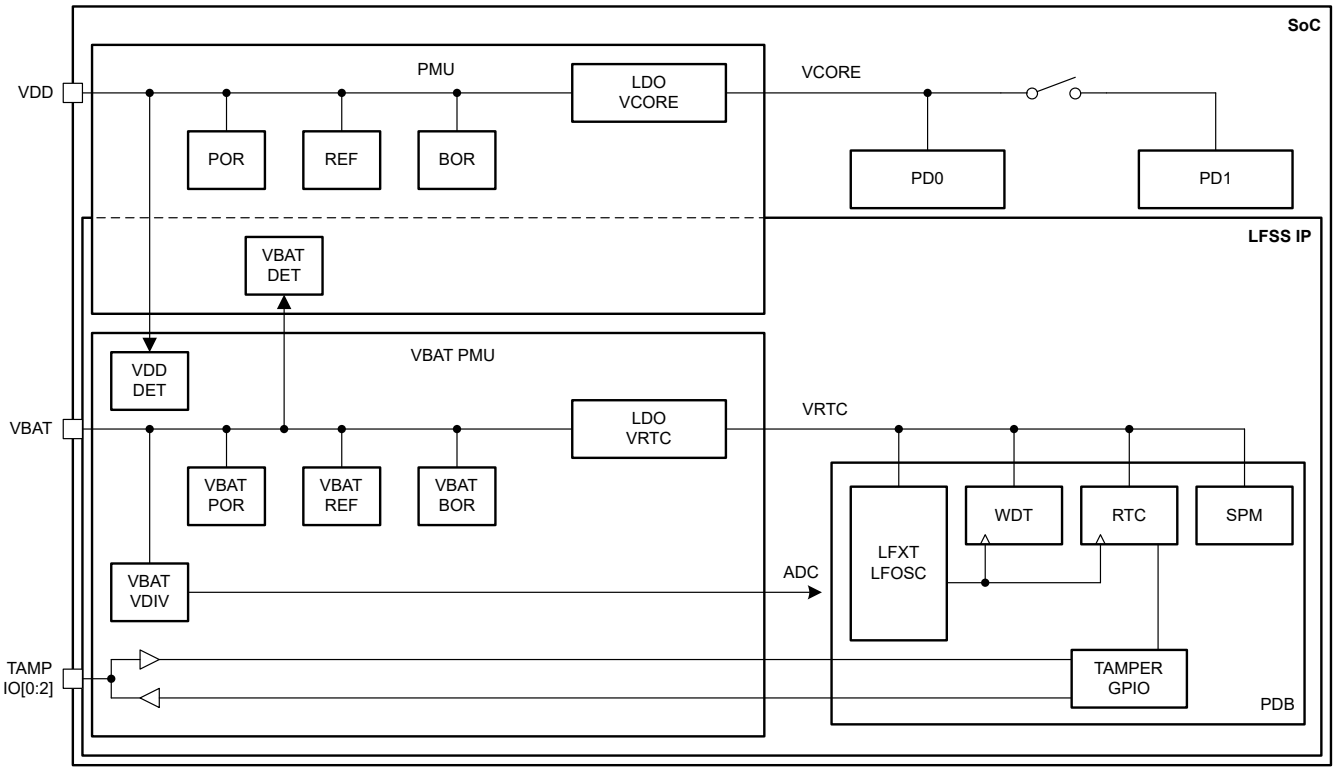

图 2-1. LFSS 方框图

2.1 使用 VBAT 对 LFSS IP 进行复位

LFSS IP 有一个在 VBAT 域中实现的复位信号生成电路。与 LFSS 复位相关的两个子电路是上电复位 (POR) 和欠压复位 (BOR)。POR 是 VBAT 域上基于 V_{th} 的电源电压检测器，用于复位电源管理单元 (PMU) 和启动冷启动序列。BOR 是基于基准的电压监测器，可在 VBAT 电源足以使 LDO 安全运行时启用 LDO。在 VBAT 电源初始上电和 VRTC LDO 启用时，VRTC 会检测到初始复位。一旦将复位置为无效，除非电源电压降至 BOR 电平以下，否则 VRTC 域不会检测到另一个复位。POR 和 BOR 不会复位 VRTC 域。

2.2 电源域电源检测

LFSS IP 需要独立监控 VBAT 电源和 VDD 电源是否存在，因为没有专用的常开域可以指示哪个域已通电、哪个域未通电。VDD 和 VBAT 可以由不同的电压电平供电。例如，片上系统 (SoC) 可由 PCB 上系统 LDO 的 1.8V LDO 供电，而 VBAT 则由 3V 纽扣电池供电。或者，在另一个方向上，SoC 可由 LDO 系统提供 3.3V 电源。

图 2-1 表明，两个电源域的电源域检测、隔离和复位信号生成功能都非常相似。每个域都有一个 POR 和 BOR 电路。子电源域 (VCORE、VRTC) 的隔离和复位信号由复位锁存器-设置锁存器 (RS-latch) 生成。检测到 POR 或 BOR 事件时，隔离和复位信号将主动置为有效 (SET)。一旦子域电压比较器为相应域提供了 OK 信号，便会清除 RS-latch。

2.2.1 启动序列

在未上电的 SoC 上首先为 VBAT 域供电时，LFSS IP 会使用 VBAT PMU 的异步序列启动。VBAT POR 释放电压约为 0.9V，这可以启动 VBAT REF 电路。当 OK 信号被置为有效时，该信号会启用 BOR 电路。一旦 BOR 指示 VBAT 已超过 1.62V 的最小工作电源电压，便会启用 LDO-VRTC。一旦 VRTC 域达到 1.35V，比较器就会指示良好状态，从而从 VRTC 域释放复位。复位后，LFOSC 会启动一个 32kHz 时钟，以便在采样模式下运行 VBAT REF 和 BOR 电路。此状态下的 LFSS IP 功耗应该低于规格限值，使纽扣电池使用寿命可以长达 10 年。

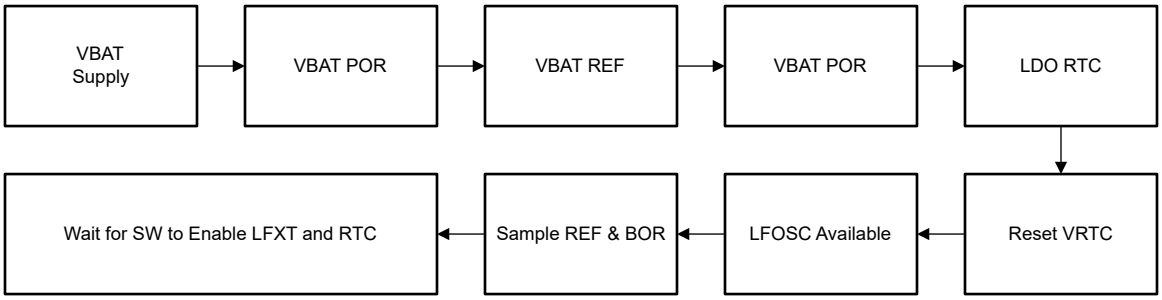

图 2-2. VBAT 优先时的启动序列

当首先为 VDD 域供电时，SoC 会使用 PMU 的异步序列启动。这种情况与不带 LFSS IP 的 SoC 的启动序列相同。区别在于 LFCLK 不可在 SoC 中使用。VBAT 检测器会指示 VBAT 尚不存在。在这种情况下，SoC 无法进入 STANDBY 模式，因为 STANDBY 模式需要 32kHz 的频率。

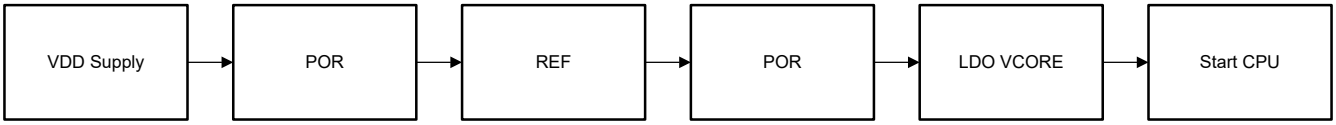

图 2-3. VDD 优先时的启动序列

2.2.2 LFSS IP 行为

由于有所需的系统级去耦电容器，VDD 的功率损耗不会瞬时发生。随着 VDD 电源电压下降并达到 BOR 电平 (1.62V)，VCORE LDO 将被禁用并会设置 VCORE_ISO 锁存器。最终，VDD 电平降至 POR 电平以下并复位 SoC 的 VDD 域。该状态由 VDD 检测电路检测，而这一电路由 VBAT 域供电。

关断模式序列不同，因为 VDD 域保持通电状态。关断模式由软件启动并存储在 VDD 域中的关断模式寄存器内。因此会禁用 LDO 并设置 VCORE_ISO 锁存器。在从关断模式唤醒时，SoC PMU 会重新启用 REF 系统、BOR 和 LDO。

VBAT 电源引脚上需要一些最小去耦电容器，主要用于抗噪和在高阻抗电源的开关活动期间提供峰值电流。由于需要去耦电容器，VBAT 电源会在斜坡期间发生损耗。由于斜坡，VBAT-BOR 电路会首先检测到 VBAT 域的损耗，然后设置 VRTC_ISO 锁存器。同时，该电路会禁用 VBAT-PMU，VRTC_ISO 隔离信号穿过 VDD 域，并隔离来自 VRTC 域的所有信号。

2.3 LFXT、LFOSC

LFXT 和 LFOSC 是外部和内部低频晶体振荡器，典型频率为 32kHz。LFCLK 控制位位于 SYSCTL 寄存器内，用户控制信号锁存在影子寄存器中，允许在 VDD、VCORE 断电期间激活 LFCLK 配置。在所有模式下，LFCLK 仅在 VBAT 域通电时工作。图 2-4 展示了 LFXT 和 LFOSC 与其他外设一起运行的流程。

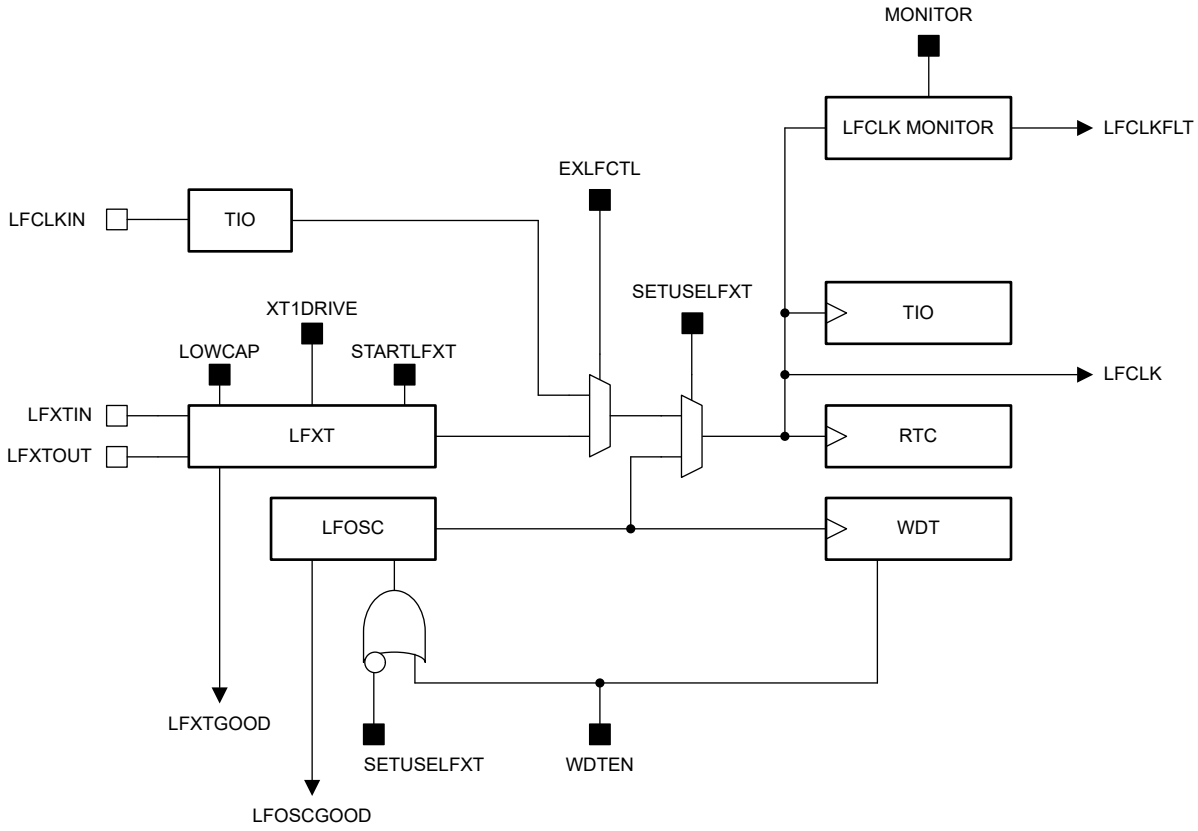

图 2-4. LFXT、LFOSC 流程图

2.4 独立看门狗计时器 (IWDT)

LFSS 中的独立看门狗计时器 (IWDT) 是与器件无关的监控器，可监控代码执行和整体挂起场景。IWDT 是一个具有闭合和开放窗口的 25 位计数器，由 LFSOC 驱动，时钟路径为 32kHz，如图 2-5 所示。IWDT 具有八个可选的看门狗计时器周期。IWDT 旨在取代系统中的外部看门狗计时器，并且某些安全应用需要由 IWDT 进行监控。IWDT 的主要功能是在计时器检测到器件因软件或系统意外延迟而导致运行故障时启动完全电源复位。

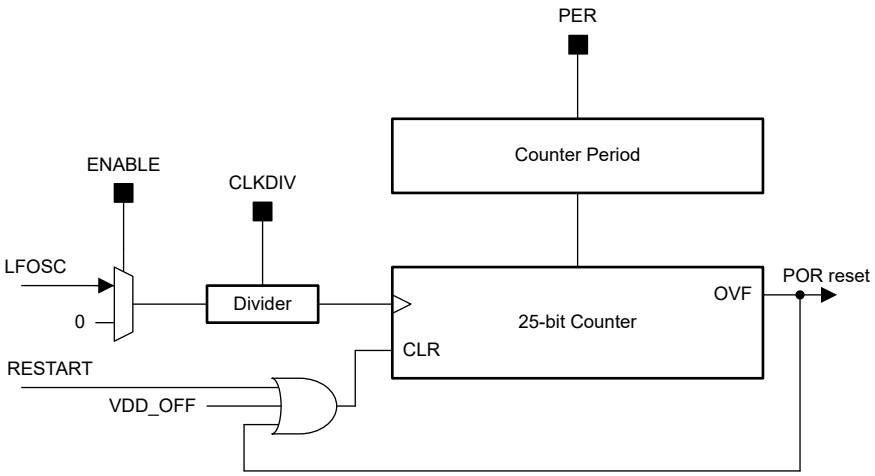

图 2-5. IWDT 流程图

2.5 防篡改 I/O

防篡改 I/O 是一种使用 VBAT 域供电的通用输入/输出 (GPIO)，在 LFSS 中具有两种运行模式。I/O 引脚的输出能够驱动来自 VBAT 域的低电流 LED。防篡改输入和输出用于检测篡改事件，例如输入引脚上的电压转换或者可以启用或禁用外设的输出。防篡改 I/O 通常用于警报控制系统、Microsoft® Windows® 安全等。

2.5.1 IOMUX 模式

不同的外设具有不同的信号，IOMUX 模式允许用户将外设设置为特定的信号。图 2-6 展示了 IOMUX 模式也是 LFSS 的默认模式，主要用于 VDD 和 VBAT 均短路时，或 LFSS 用作辅助电源以在不同的电压电平下运行辅助功能时。IOMUX 模式的一个示例是使用由 3V 供电的器件来实现完整的模拟性能，并使用 LFSS 作为内部电平转换器，以支持与需要 1.8V 电压的 CPU 的串行外设接口 (SPI) 连接。IOMUX 模式的一些限制包括：

- 当失去主电源或器件处于 SHUTDOWN 模式时，I/O 不起作用
- 不允许将器件从 SHUTDOWN 模式唤醒
- 当 VCORE 不可用时，防篡改 I/O 也不可用

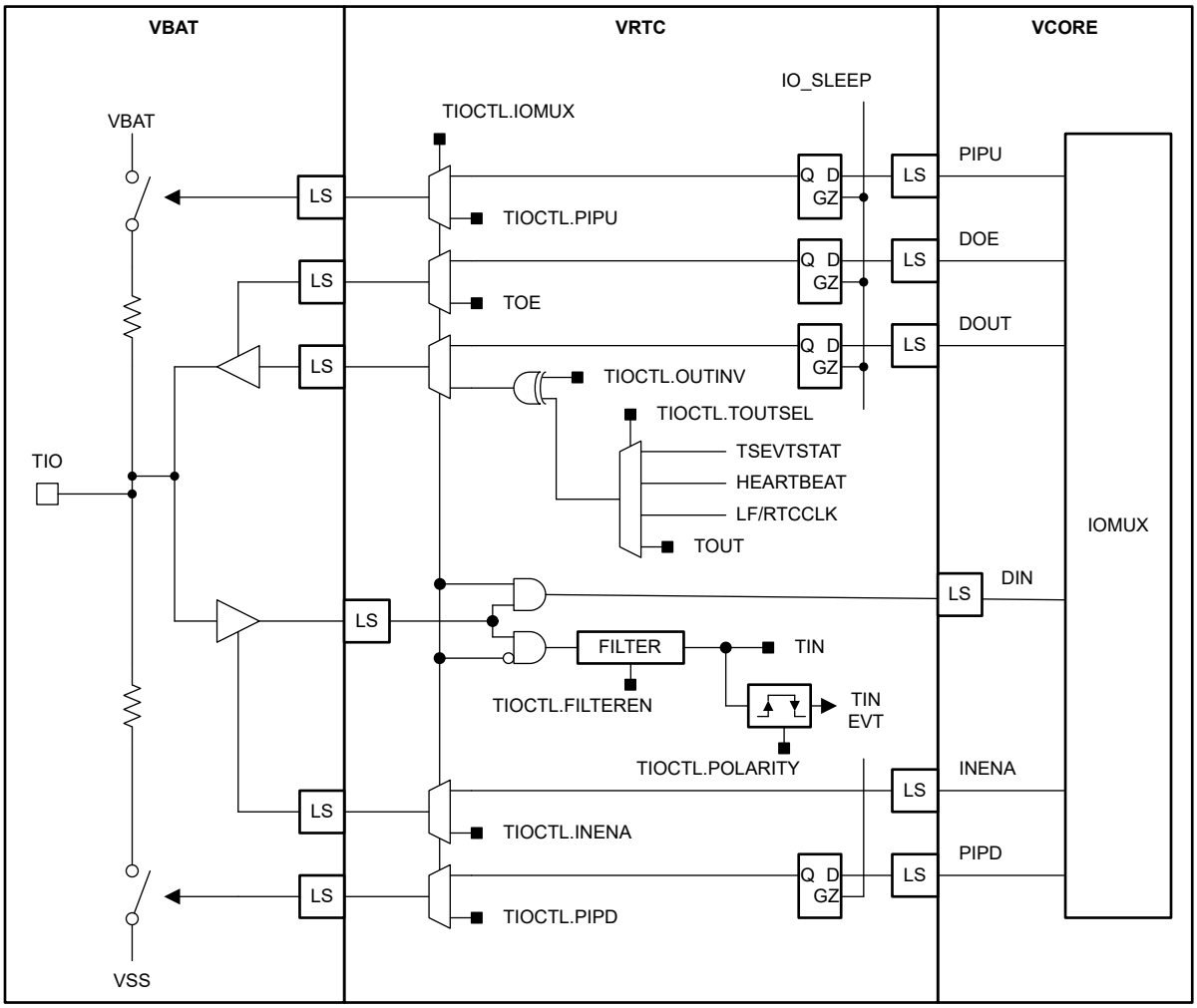

图 2-6. IOMUX 模式流程图

2.5.2 防篡改模式

在防篡改模式下，所有控件和电路都由内部 VRTC 域供电，并且当断电或器件处于 SHUTDOWN 模式时，控件和电路将保持正常工作。用户能够将防篡改 I/O 配置为输入或输出模式。防篡改模式还支持多种内部辅助功能。

2.5.2.1 篡改事件检测

篡改事件检测功能允许开发人员配置一个或多个防篡改 I/O 来触发时间戳以向 CPU 产生中断。配置的防篡改 I/O 必须是输入以用作篡改事件触发器。为了防止误触发，数字滤波器电路可以检测滤波器宽度是否与配置相同。可以选择输入滤波器宽度为无、30 µs、100 µs 或 200 µs。

2.5.2.2 时间戳事件输出

时间戳和事件输出由主电源的边沿检测或功率损耗检测进行触发。此功能将 RTC 状态捕获为第一次和最后一次发生事件的时间戳。TSCTL 寄存器控件能够识别捕获的是第一个事件还是最后一个事件。为了提高可见性，用户能够配置防篡改 I/O 以通过 I/O 引脚显示外界状态。它会保持有效状态，直到软件寄存器 TSCTL.TSCLR 清除时间戳。时间戳应用程序可以在网络安全日志、数据库管理等实际应用程序中找到。

2.5.2.3 检测信号发生器

检测信号发生器允许用户查看某些 LFSS 工作状态的信令。该发生器可用作外部看门狗的触发器，通过闪烁 LED 来指示 RTC 或其他应用是否仍在运行。利用防篡改 I/O 触发由时间戳捕获的事件的一个示例是电表计量。时间戳捕获的事件通过检测信号发生器发出信号以通知用户有事件被检测到。在图中，RT2PS、RT1PS 和 RT0PS 计数器逐级传递进行不断分频，最终设置检测信号发生器的检测信号速度。用户能够将 HBINTERVAL 设置为 125ms、250ms、500ms、1s、2s、4s、8s 和 16s 之间的值。用户还能够以二进制步长在 1ms 至 128ms 的范围内设置 HBWIDTH。图 2-7 展示了完整的流程图。

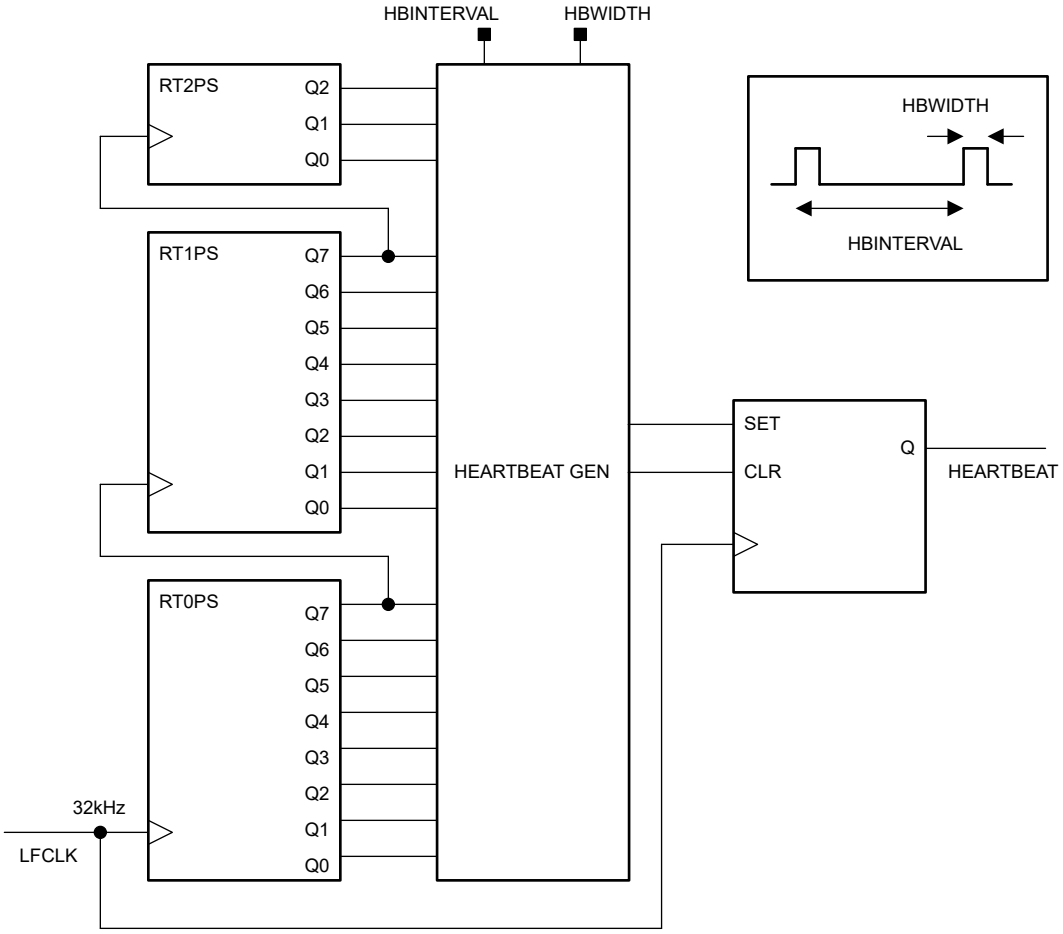

图 2-7. 检测信号发生器图

2.6 便笺式存储器 (SPM)

SPM 包含 16 到 256 字节基于寄存器的存储器，并且只要向 VBAT 供电，就能像闪存一样一直保留数据。在主 VDD 电源丢失或处于关断模式时，SPM 可以保留存储器内容。如果 VBAT 断电，SPM 不能保留任何存储器内

容。存储器大小为 4 至 64 个 32 位字，每一位均可寻址以进行读写。图 2-8 展示了实现 LFSS 写入使能寄存器以防止覆盖存储器。

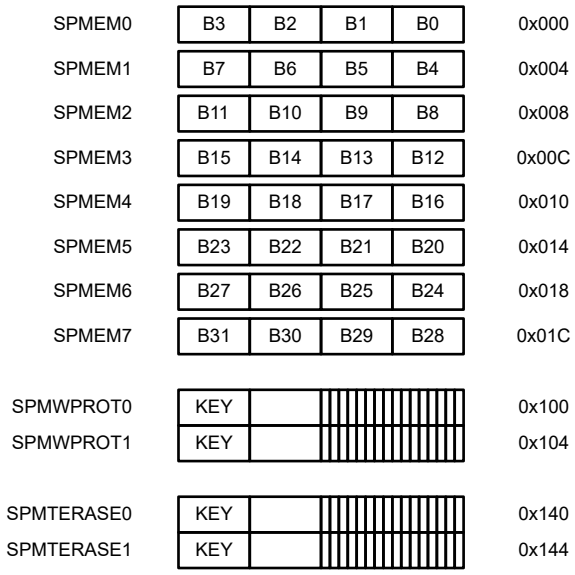

图 2-8. SPM 图

2.7 实时时钟 (RTC)

LFSS 有一个 RTC 模块可提供计时应用。通过计数器，用户可以查找秒、分钟、小时、星期几、月份日期和年份。这些值可以在 LFSS 寄存器中以二进制或二进制编码小数 (BCD) 形式显示。RTC 还可用于设置基于日历事件或时间间隔的警报。如果发生篡改事件或 VDD 丢失，RTC 将执行时间戳捕获。RTC 控制和日历寄存器可实施写保护，以防止意外更新。图 2-9 展示了完整的 RTC 流程图。

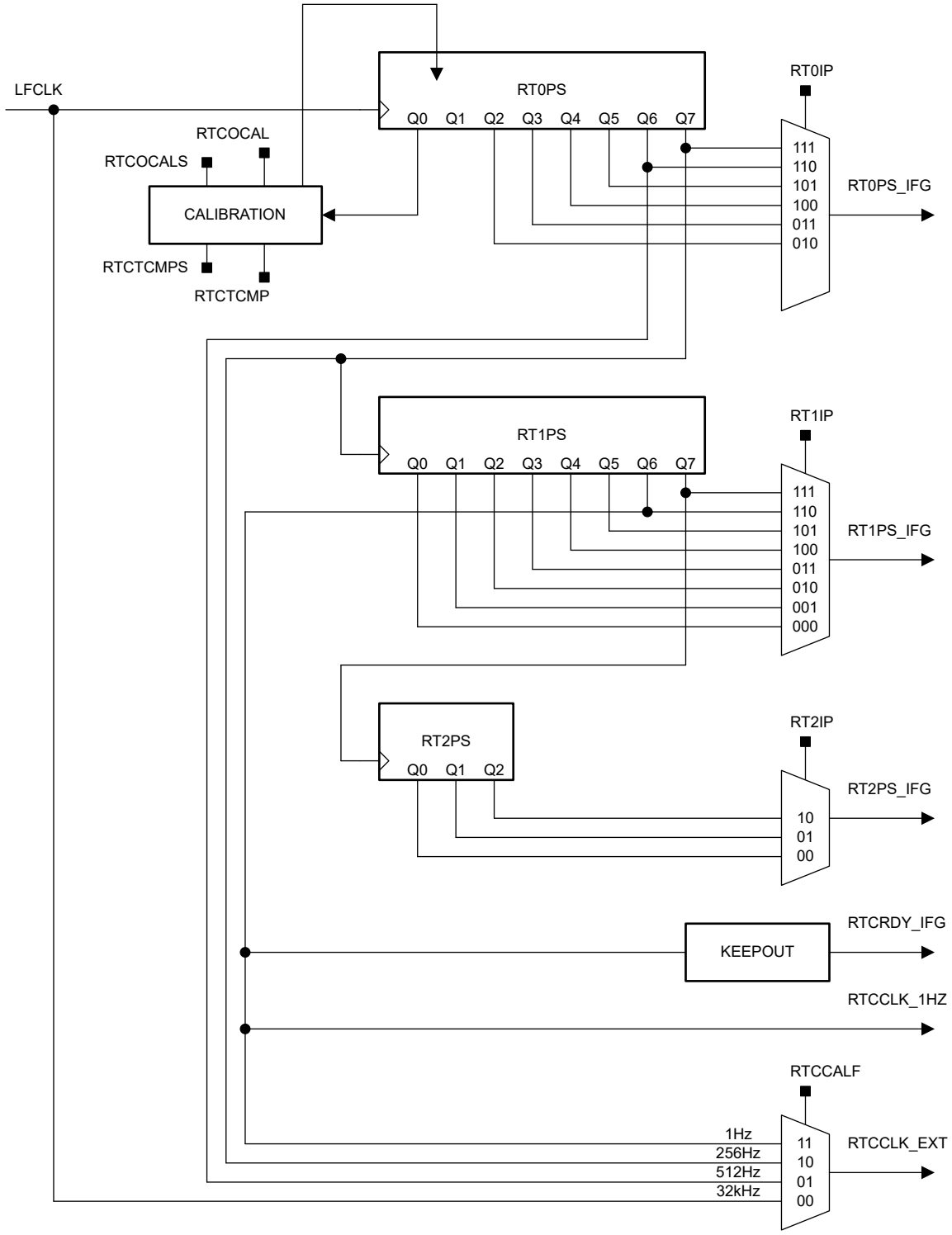

图 2-9. RTC 流程图

2.8 VBAT 充电模式

如表 2-1 所示，当 VDD 大于 VBAT 时，简单充电电路可以通过主 VDD 电源在 VBAT 上为超级电容器提供限流充电，同时在 VBAT 为 0V 时还允许 VBAT 从时间零点启动。当设置了 SYSTEMCFG 寄存器中的 SUPERCAPEN 位后，需要通过软件启用充电电路。确保设置了 SYSSTATUS 寄存器中的 VBATGOOD 位，仅当 VBAT 电源域有效时才会设置该位。

表 2-1. 规格

参数		值	单位
VDD	典型值	3.3	V
R_{SWITCH}	典型值	1400	Ω
R_{SWITCH}	最大值	2700	Ω
C_{SCAP}	典型值	0.33	F
V_{BAT}	最小值	1.62	V
最大 ICHARGE（最小 REXT）			
参数		**值**	**单位**
R_{EXT}		**3300**	Ω
$I_{CHARGE(T0, MAX)}$		**1.6**	mA
充电时间			
参数		**值**	**单位**
Tau		1980	s
完全充电时间 (5T)		9900	s
		165	分钟
		2.8	小时
运行时			
参数		**值**	**单位**
V_{RANGE} (VDD-$V_{BAT,MIN}$)		1.68	V
I_{DIS}		1.5	μA
T_{DIS}（放电寿命）		369600	s
		6160	分钟
		102	小时
		4.2	天

R_{EXT} 和 R_{SWITCH} 一起设置时间零点最大充电电流 $I_{CHARGE(T0,MAX)}$。可以通过增加 R_{EXT} 来降低 $I_{CHARGE(T0,MAX)}$，但要牺牲充电时间，不过 R_{EXT} 必须达到 $R_{EXT(MIN)}$ 的最小值，以确保 VBAT 引脚检测到 VBOR+；因此，备用岛在 T0 处启动，不需要等待充电。

$$\mathrm{VBAT}\left(\mathrm{T0}\right) = \left(\mathrm{VDD} - \mathrm{VCAP}\right)\frac{\mathrm{REXT}}{(\mathrm{REXT} + \mathrm{RSWITCH})} \quad (1)$$

$$\mathrm{ICHARGE} = \frac{(\mathrm{VDD} - \mathrm{VCAP})}{(\mathrm{REXT} + \mathrm{RSWITCH})} + \mathrm{ILEAK} \quad (2)$$

$$\mathrm{REXT} = \frac{(\mathrm{VBAT})(\mathrm{RSWITCH})}{(\mathrm{VDD} - \mathrm{VBAT})} \quad (3)$$

$$\mathrm{ICHARGE}\left(\mathrm{T0, MAX}\right) = \frac{\mathrm{VDD}}{\frac{(\mathrm{VBAT})(\mathrm{RSWITCH})}{(\mathrm{VDD} - \mathrm{VBAT})} + \mathrm{RSWITCH}} \quad (4)$$

充电由软件启用，并在 VDD 降至 VBAT 以下时自动禁用。放电寿命取决于 VBAT 烧毁 (IDIS)，R_{EXT} 为最小值。

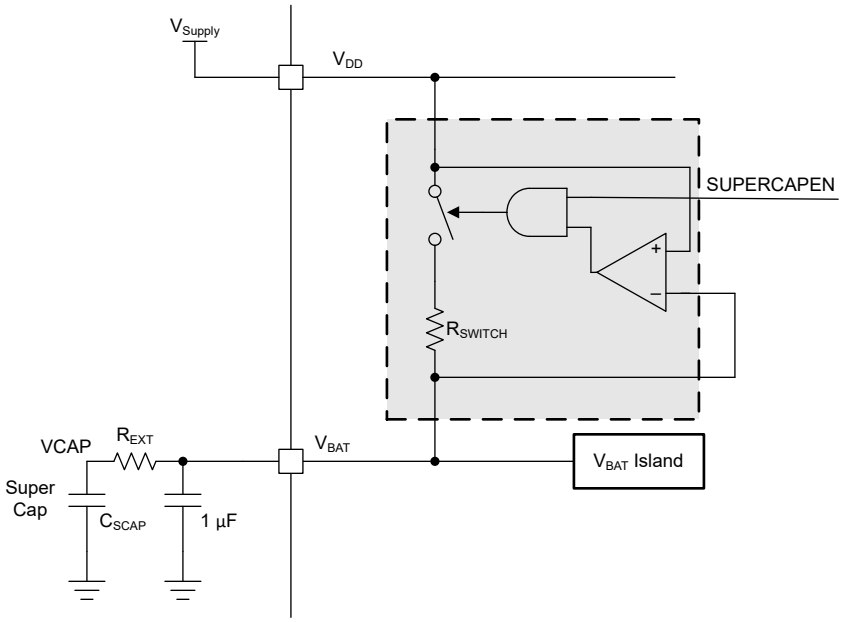

图 2-10. 超级电容器充电电路

3 应用示例

本节通过 Code Composer Studio 示例演示 LFSS。共有五个不同的示例展示了如何在 LFSS 中使用不同的外设。

3.1 防篡改 I/O 检测信号示例

此示例展示了当主电源 VDD 发生断电时检测信号发生器仍正常工作的情况。此示例是一个真实示例，例如之前在 VDD 仍然通电时设置的警报系统在 VDD 断电后仍能正常工作。为了让此示例正确运行，可以使用电池或外部电源为 VBAT 域供电。将防篡改 I/O PA7 配置为输出以驱动 LED。设置完成后，运行示例代码以切换相应的 LED PB2。通过断开主电源 VDD，只要 VBAT 域通电，LED 就会继续闪烁。

```
#include "ti_msp_dl_config.h"

#define DELAY (16000000)

int main(void)
{
    /* Initialization */
    SYSCFG_DL_init();
    while (1) {
        delay_cycles(DELAY);
        DL_GPIO_togglePins(GPIO_LEDS_USER_LED2_PORT,
           GPIO_LEDS_USER_LED2_PIN |GPIO_LEDS_USER_TEST_PIN);
    }

    return 0;
}
```

3.2 RTC 防篡改 I/O 时间戳事件示例

此示例需要两种防篡改 I/O 配置。将防篡改 I/O *0* 配置为输入以用作篡改事件触发器。时间戳事件输出可捕获事件发生情况，RTC 则记录捕获事件的时间。接下来，将防篡改 I/O *1* 配置为输出以切换检测信号模式的 LED。在防篡改 I/O 触发上升沿时，TSCTL 寄存器将捕获发生的第一个或最后一个事件。配置的 LED 会在 VDD 断电时一直闪烁，表明检测信号发生器仍在运行并由 VBAT 供电。

```
#include "ti_msp_dl_config.h"

volatile bool gCheckTSEVT = false;
volatile uint32_t counter = 0;
volatile uint8_t gBlink;

int main(void)
{
    /* Initialization */
    SYSCFG_DL_init();

    /* Enable the RTC interrupt at NVIC */
    NVIC_EnableIRQ(RTC_A_INT_IRQn);

    /* Start RTC clock */
    DL_RTC_A_enableClockControl(RTC_A);

    while (1) {
        __WFI();

        /* Wait in a while() loop until the time stamp interrupt triggers - Trigger a tamper event
externally */
        while (gCheckTSEVT == false)
            ;

        /* Blink LED a number of times equal to amount of time stamp events detected */
        if (DL_RTC_A_getTimeStampEventCause(
                RTC_A, DL_RTC_A_TIME_STAMP_EVENT_CAUSE_TIO_0) ==
            DL_RTC_A_TIME_STAMP_EVENT_CAUSE_TIO_0) {
            for (gBlink = 0; gBlink < (2 * counter); gBlink++) {
                DL_GPIO_togglePins(GPIO_LEDS_LED_2_PORT, GPIO_LEDS_LED_2_PIN);
                delay_cycles(16000000);
            }
        }
    }
}
```

```
void LFSS_IRQHandler()
{
    switch (DL_RTC_A_getPendingInterrupt(RTC_A)) {
        case DL_RTC_A_IIDX_TSEVT:
            counter++;
            gCheckTSEVT = true;
        default:
            break;
    }
}
```

3.3 超级电容器充电示例

此示例使用 CCS 防篡改 I/O 检测信号代码示例来演示超级电容器充电电路的使用情况。要开始使用此示例，请更改从 VDD 到 CAP 的 VBAT 跳线。然后，在软件中启用超级电容器寄存器，使超级电容器充电电路可以正常工作。将所需的防篡改 I/O 配置为输出，并将防篡改 IO 连接到 PB2 LED 以执行检测信号发生器。超级电容器充满电后，断开主电源 VDD。超级电容器开始向 VBAT 域进行回充。当 VBAT 域上电时，LED 在主电源断电后继续闪烁，表示超级电容器充电电路正常工作。当超级电容器完全放电后，LED 最终停止闪烁。

```
#include "ti_msp_dl_config.h"

#define DELAY (16000000)

int main(void)
{
    /* Initialization */
    SYSCFG_DL_init();
    DL_SYSCTL_enableSuperCapacitor(); /*enables SuperCap register to perform supercap charging
circuit example */
    while (1) {
        delay_cycles(DELAY);
        DL_GPIO_togglePins(GPIO_LEDS_USER_LED2_PORT,
           GPIO_LEDS_USER_LED2_PIN |GPIO_LEDS_USER_TEST_PIN);
    }

    return 0;
}
```

3.4 LFOSC 转换回 LFXT 的示例

此示例展示了水表计量中存在的一个问题，即 LFXT 由于湿度而被禁用。LFXT 自动禁用后，会转换为使用 LFOSC 来保持低频时钟以 32kHz 的频率运行。当 LFXTGOOD 位为 true 时，许多用户在转换回 LFXT 时会遇到问题。此示例说明了如何转换回 LFXT。

Code Composer Studio 提供了时钟树选项，可用于配置所需的时钟路径。将 LFXT 设置为 LFCLK 的源，并确保已启用 LFCLK 启动监视器和 LFCLK 监视器。这样就可以监视 LFXT 转换到 LFOSC 失败的情况。将时钟输出设置为所需的引脚以测量频率，从而确保时钟正常工作。将 LFX_Out 接地可以禁用 LFXT，该操作会切换 LED PA 16 以指示 LFXT 发生故障且正在使用 LFOSC。LFOSC 无法自动转换回 LFXT；因此，在将启用 STARTLFXT 位的键设置为 true 的过程中，我们随后会转到 IRQHandler 以将启用 SETUSELFXT 位的键设置为 true 并再次开启时钟监视器。当 LFXT 被清除并且适合再次使用时，这些过程允许从 LFOSC 再次转换回 LFXT。

```
#include "ti_msp_dl_config.h"

bool clockFailed = false;
static const DL_SYSCTL_LFCLKConfig gLFCLKConfig = {
    .lowCap   = true,
    .monitor  = true,
    .xt1Drive = DL_SYSCTL_LFXT_DRIVE_STRENGTH_HIGHEST,
};
int main(void)
{
    SYSCFG_DL_init();

    SYSCTL->SOCLOCK.LFCLKCFG |= SYSCTL_LFCLKCFG_MONITOR_ENABLE;  /*enable LFCLK monitor*/

    delay_cycles(32000);
    DL_SYSCTL_enableSleepOnExit();
    NVIC_EnableIRQ(SYSCTL_INT_IRQn);
```

```
    while (1) {
        if(clockFailed == 1){     //it wont hit this line of the code unless you pause the program


           SYSCTL->SOCLOCK.LFXTCTL = (SYSCTL_LFXTCTL_KEY_VALUE |
SYSCTL_LFXTCTL_STARTLFXT_TRUE);   //this starts the STARTLFXT bit
           delay_cycles(32000);
           clockFailed = false;
        }

    }
}

void NMI_Handler(void)
{

    switch (DL_SYSCTL_getPendingNonMaskableInterrupt()){
        case SYSCTL_NMIIIDX_STAT_LFCLKFAIL:  /*toggles LED as LFXT failed*/
            DL_GPIO_togglePins(GPIO_LEDS_PORT, GPIO_LEDS_User_LED_2_PIN |
GPIO_LEDS_USER_LED_1_PIN);  /*toggle LED PA16*/
            DL_SYSCTL_enableInterrupt(0x10);   //enable the bit in the IMASK of LFXTGOOD

            clockFailed = true;
            break;
        default:
            break;

    }

}
void GROUP0_IRQHandler(void){
    volatile uint32_t PendingInterrupt = SYSCTL->SOCLOCK.IIDX;
  switch(PendingInterrupt){
        case 5:
            DL_GPIO_togglePins(GPIO_LEDS_PORT, GPIO_LEDS_User_LED_2_PIN | GPIO_LEDS_USER_LED_1_PIN);
            clockFailed = false;
            SYSCTL->SOCLOCK.LFXTCTL = (SYSCTL_LFXTCTL_KEY_VALUE |
SYSCTL_LFXTCTL_SETUSELFXT_TRUE);    //this write the key and set it to use LFXT
            SYSCTL->SOCLOCK.LFCLKCFG = SYSCTL->SOCLOCK.LFCLKCFG |
SYSCTL_LFCLKCFG_MONITOR_MASK;         //this turns on the clock monitor
            break;
  }
}
```

3.5 RTC_A 校准

内部振荡器的精度以百分比为单位，而外部晶体以百万分率 (ppm) 为单位。晶体振荡器虽然功耗较高，但由于切割、形状和尺寸精确，因此精度很高。但是，振荡器的谐振频率仍然受外部因素的影响。温度、老化、机械冲击和振动以及重力都会影响晶体的精度。此示例显示了在测量偏移误差和基于温度的误差时如何校准 LFSS 的 RTC。

3.5.1 外设 ADC 12

ADC 与 MSP 的内部温度传感器搭配使用，可测量器件温度。具体实现流程如下：

1. 使用温度系数 (TS_C)、出厂修整温度 (TS_{TRIM}) 和特定于单位的单点修整值（在寄存器：TEMP_SENSE0 中）计算修正电压
2. 对内部温度传感器进行采样
3. 将原始 ADC 值转换为电压
4. 使用方程式 5 估算器件温度：

$$T_{Sample} = \left(\frac{1}{TS_C}\right) \times \left(ADC_{CODE} - TEMP_{SENSE0.DATA}\right) + TS_{TRIM} \quad (5)$$

TS_C 和 TS_{TRIM} 可以在器件数据表的*规格*部分中找到。$TEMP_{SENSE0}$ 位于存储器的出厂常量部分。温度传感器是器件相关的 ADC 通道上的输入，原始 ADC 值存储在 ADC 转换存储器 0 中。单次校准值是使用 1.4V 内部基准电压测量温度传感器的 ADC 结果。

3.5.2 RTC_A

使用 RT0PS 的 Q0 输出对 RTC 进行校准需要超过 60 秒的时间。通过每四分之一秒增加或减去 1 个时钟脉冲，可以近似得出 ±1ppm 的校正值。一个校准周期内最大可调节范围为 ±240ppm。每个周期都包含对偏移误差和温度误差的调整。超过 ±240ppm 界限的调整会达到饱和状态，但可以在多个周期内完成总补偿。如果未启用 RTC，校准将被禁用。

有 CAL 和 TCMP 两个寄存器用于校准。偏移调整值写入 CAL 寄存器，温度写入 TCMP。但是，TCMP 寄存器会存储两种误差类型之间的净补偿值。当设置 CAL 寄存器时，温度调整值复位为 0，并且 TCMP 仅存储偏移调整值。

进行校准以调整偏移误差时，RTC 提供一系列频率输出以用于外部测量。测试频率为 LFCLK (32kHz)、512Hz、256Hz 和 1Hz。选择结果存储在 CAL 寄存器的 RTCOCALFX 位字段中，如果设置为 *0*，则会禁用偏移校准。可以使用方程式 6 来计算偏移误差。

$$60 \times 16384 \times \left(\frac{1 - f_{\mathrm{RTCCLK}}}{32768}\right) \tag{6}$$

其中

- 60 是校准周期长度（以秒为单位）
- 16384 是 RT0PS 的 Q0 输出频率
- f_{RTCCLK} 是测得的输出频率乘以时钟分频因子
- 32768 是期望的 LFCLK 频率

计算出误差后，可以写入符号和幅度位字段。符号指示是向上还是向下调整。如果误差为正（向上校准），则设置 RTCOCALS，如果符号为负（向下校准），则清除 RTCOCALS。RTCOCALX 是幅度，限制为最大 240ppm。如果将有符号结果写入 RTCOCALX，负值用二进制补码表示，并饱和至 240ppm。

温度漂移校准方法是获取 ADC 的温度近似值，计算由相对于晶体周转温度的偏差引起的频率误差，然后将结果写入 TCMP 寄存器。获取温度近似值的过程如节 3.5.1 所述。使用方程式 7 可计算频率误差。

$$\mathrm{T_{COMP}} = (\mathrm{T_{APPROX}} - \mathrm{Ti}) \times \mathrm{B} \tag{7}$$

其中

- T_{COMP} 是温度补偿值
- T_{APPROX} 是测量的温度近似值
- Ti 是晶体振荡器周转温度
- B 是晶体振荡器抛物线系数

前面的信息可以在晶体振荡器数据表中找到。

该公式会根据时钟信号的来源而变化。具体而言，周转温度和抛物线系数变量取决于所使用的振荡器类型。此示例中使用的是音叉外部晶体，周转温度和抛物线系数见数据表。音叉晶体的一个独特属性就是温度偏差会导致负频率偏差。这意味着补偿方向是向上。计算出误差后，与偏移误差类似，符号（向上）将写入 RTCTCMPS 位字段，而幅度将写入 RTCTCMPX 位字段。写入成功后，当读取 TCMP 寄存器时，该寄存器已存储偏移误差和温度偏差之间的净补偿值。如果净补偿值太大，RTCTCMPX 将饱和至 240ppm。由温度偏差确定的补偿将在下一个校准周期引入，而不是在当前校准周期引入。由于每个周期的长度为 60 秒，此示例每分钟只测量一次温度，并由 RTC_A 间隔中断进行触发。可以更改该值来多次测量温度并取平均值。

```
#include "ti_msp_dl_config.h"

/*
 * The following trim parameter is provided in the device datasheet in chapter
 * "Temperature Sensor"
 */
#define TEMP_TS_TRIM_C                                          ((uint32_t)30)

/*
 * Constant below is (1/TSc). Where TSc is Temperature Sensor coefficient
```

```
 * available in the device datasheet
 */
#define TEMP_TS_COEF_mV_C                                          (-555.55f)

#define ADC_VREF_VOLTAGE                                             (1.4f)
#define ADC_BIT_RESOLUTION                                 ((uint32_t)(1)<<12)

/*
 * The following turnover and  Parabolic Coefficient parameter is provided
 * in the crystal oscillator datasheet
 */
#define TEMP_CRYSTAL_Ti_C                                      ((uint32_t)25)
#define TEMP_CRYSTAL_B_PPM_C2                                        (-0.04f)

/* Offset measurements */
typedef struct measFreq {
    double freq;
    DL_RTC_COMMON_OFFSET_CALIBRATION_SIGN sign;
} measFreq;

volatile bool gStatus = false;
volatile bool gCheckADC;

void offset_error_correction(void) {
    measFreq slowCrystal;
    measFreq fastCrystal;

    /* Simulated frequency measurements (will need to be manually measured)*/
    slowCrystal.freq = 511.9658;
    fastCrystal.freq = 512.0241;

    /* Correction sign is adjusted up for slow measured frequencies and down for fast measured
frequencies */
    slowCrystal.sign = DL_RTC_COMMON_OFFSET_CALIBRATION_SIGN_UP;
    fastCrystal.sign = DL_RTC_COMMON_OFFSET_CALIBRATION_SIGN_DOWN;

    measFreq examples[2] = {slowCrystal, fastCrystal};

    /* Process for updating calibration register (for both slow and fast measurements)*/
    for(uint8_t i = 0; i < 2; i++) {
        /* Division Factor for 512Hz output frequency */
        uint8_t divFact = 64;

        /* Ideal Crystal Oscillation frequency*/
        uint16_t optOsc = 32768;

        /* Error Correction Formula:
         * Frtcclk = Frtcclk,meas * divider factor
         * Offset Value = Round(60 * 16384 * (1 - Frtcclk/32768))
         */

        double Frtcclk = examples[i].freq * divFact;
        uint8_t offVal = fabs(round(60 * 16384 * (1 - (Frtcclk / optOsc))));

        /* Wait until RTC is ready for compensation */
        while (!DL_RTC_A_isReadyToCalibrate(RTC_A)) {
            ;
        }

        /* Sets the offset error calibration adjustment value */
        DL_RTC_A_setOffsetCalibrationAdjValue(RTC_A, examples[i].sign, offVal);

        /* Check if write was successful */
        gStatus = DL_RTC_A_isCalibrationWriteResultOK(RTC_A);

        /* Stop the debugger to examine the output. At this point,
         * gStatus should be equal to "true" and the CAL register
         * should be updated.
         * slow crystal adjustment should be +66ppm
         * fast crystal adjustment should be -46ppm
         */
        __BKPT(0);
    }
}

void temp_drift_correction(float vTrim) {
    uint32_t adcResult, tComp;
    float vSample, tSample;
```

```
    DL_RTC_COMMON_TEMP_CALIBRATION tSign;

    gCheckADC = false;

    /* Read stored ADC value */
    adcResult = DL_ADC12_getMemResult(ADC12_0_INST, ADC12_0_ADCMEM_0);

    /*
    * Convert ADC result to equivalent voltage:
    * Vsample = (VREF_VOLTAGE_MV*(adcResult -0.5))/(2^ADC_BIT_RESOLUTION)
    */
    vSample = ADC_VREF_VOLTAGE / ADC_BIT_RESOLUTION * (adcResult -0.5);

    /*
    * Apply temperature sensor calibration data
    * TSAMPLE = (TEMP_TS_COEF_mV_C) * (vSample - vTrim) + TEMP_TS_TRIM_C
    */
    tSample = TEMP_TS_COEF_mV_C * (vSample - vTrim) + TEMP_TS_TRIM_C;

    /*
     * Compensation sign and value
     * TCOMP = (tSample - TEMP_CRYSTAL_Ti_C)^2 * TEMP_CRYSTAL_B_PPM_C2
     */
    tSign = RTC_TCMP_RTCTCMPS_UP;
    tComp = fabs(pow((int16_t)tSample - (int16_t)TEMP_CRYSTAL_Ti_C, 2) * TEMP_CRYSTAL_B_PPM_C2);

    /* Wait until RTC is ready for compensation */
    while (!DL_RTC_A_isReadyToCalibrate(RTC_A)) {
        ;
    }

    /* Sets the temperature error calibration value */
    DL_RTC_Common_setTemperatureCompensation(RTC_A, tSign, tComp);

    /* Check if write was successful */
    gStatus = DL_RTC_A_isCalibrationWriteResultOK(RTC_A);

    /* Stop the debugger to examine the output. At this point,
        * gStatus should be equal to "true" and the TCMP register
        * should be updated.
        * The TCMP register will show the net error
        * of the offset and temperature errors.
        */
    __BKPT(0);
}

int main(void)
{
    /* Output frequency for offset calculation initialized to 512Hz */
    SYSCFG_DL_init();

    /* Enable RTC interrupts on device */
    NVIC_EnableIRQ(RTC_A_INST_INT_IRQN);

    /* Start RTC clock */
    DL_RTC_A_enableClockControl(RTC_A);

    /* Disclaimers:
     * Writing to the RTCOCAL register resets the temperature
     * to zero.
     *
     * The maximum correction in one calibration cycle is +-240ppm.
     * Any error outside this range will be ignored.
     */

    /* Offset Error Correction Mechanism */
    offset_error_correction();

    /* Temperature Drift Correction Mechanism */

    /*
     * Convert TEMP_SENSE0 result to equivalent voltage:
     * Vtrim = (ADC_VREF_VOLTAGE*(TEMP_SENSE0 -0.5))/(2^12)
     */
    float vTrim;
    vTrim = ADC_VREF_VOLTAGE / ADC_BIT_RESOLUTION * (DL_SYSCTL_getTempCalibrationConstant() -0.5);

    gCheckADC = false;
```

```
    while (1) {
        if(gCheckADC) {
            temp_drift_correction(vTrim);
        }
        DL_ADC12_startConversion(ADC12_0_INST);
        __WFI();
        DL_ADC12_stopConversion(ADC12_0_INST);
    }
}

void RTC_A_INST_IRQHandler(void)
{
    switch (DL_RTC_A_getPendingInterrupt(RTC_A)) {
        case DL_RTC_A_IIDX_INTERVAL_TIMER:
            gCheckADC = true;
            break;
        default:
            break;
    }
}
```

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265