

Low-Frequency Subsystem and VBAT Feature in MSPM0L222X



Hoa Tang

Mixed-Signal Processors

ABSTRACT

The Low-Frequency Subsystem (LFSS), is a new feature set in the MSPM0Lx22x device family that features a real-time clock (RTC), low-frequency crystal (LFXT), low-frequency oscillator (LFOSC), and scratchpad memory (SPM). The LFSS is powered by a separate power domain named *VBAT Power Domain* which allows the peripherals in the LFSS to continue running when the main VDD power is lost. This application note discusses how the LFSS and VBAT domain is used in an application. Find used examples under SDK with this address: C:\ti\mspm0_sdk_x_xx_xx_xx_internal\examples\nortos\LP_MSPM0L2228\driverlib.

Table of Contents

1 Introduction	2
2 Low-Frequency Subsystem Introduction	2
2.1 Resetting LFSS IP Using VBAT.....	2
2.2 Power Domain Supply Detection.....	2
2.3 LFXT, LFOSC.....	4
2.4 Independent Watchdog Timer (IWDT).....	4
2.5 Tamper I/O.....	5
2.6 Scratchpad Memory (SPM).....	7
2.7 Real-Time Clock (RTC).....	8
2.8 VBAT Charging Mode.....	9
3 Application Examples	12
3.1 Tamper I/O Heartbeat Example.....	12
3.2 RTC Tamper I/O Timestamp Event Example.....	12
3.3 Supercapacitor Charging Example.....	13
3.4 LFOSC Transition Back to LFXT Example.....	13
3.5 RTC_A Calibration.....	14

List of Figures

Figure 2-1. LFSS Block Diagram.....	2
Figure 2-2. Start-Up Sequence When VBAT Comes First.....	3
Figure 2-3. Start-Up Sequence When VDD Comes First.....	3
Figure 2-4. LFXT, LFOSC Flow Chart.....	4
Figure 2-5. IWDT Flow Chart.....	5
Figure 2-6. IOMUX Mode Flow Chart.....	6
Figure 2-7. Heartbeat Generator Diagram.....	7
Figure 2-8. SPM Diagram.....	8
Figure 2-9. RTC Flow Diagram.....	9
Figure 2-10. Supercapacitor Charging Circuit.....	11

List of Tables

Table 2-1. Specifications.....	10
--------------------------------	----

Trademarks

Microsoft® and Windows® are registered trademarks of Microsoft Corporation. All trademarks are the property of their respective owners.

1 Introduction

The Low-Frequency Sub-System (LFSS) combines several functional peripherals within the subsystem. In the LFSS intellectual property (IP), all the functional peripherals are clocked by low-frequency clock (LFCLK) which activates during low-power mode. The battery backup domain is initially powered by the power input VDD and compensates power loss by drawing power from the battery to keep running the functional peripherals at a frequency rate of 32kHz with the intention of long-term time keeping. The following sections provide brief descriptions for each of the LFSS peripherals. See also the [MSPM0 L-Series 32MHz Microcontrollers Technical Reference Manual](#).

2 Low-Frequency Subsystem Introduction

Figure 2-1 shows the block diagram for the LFSS. Each part of the LFSS is thoroughly explained in this document.

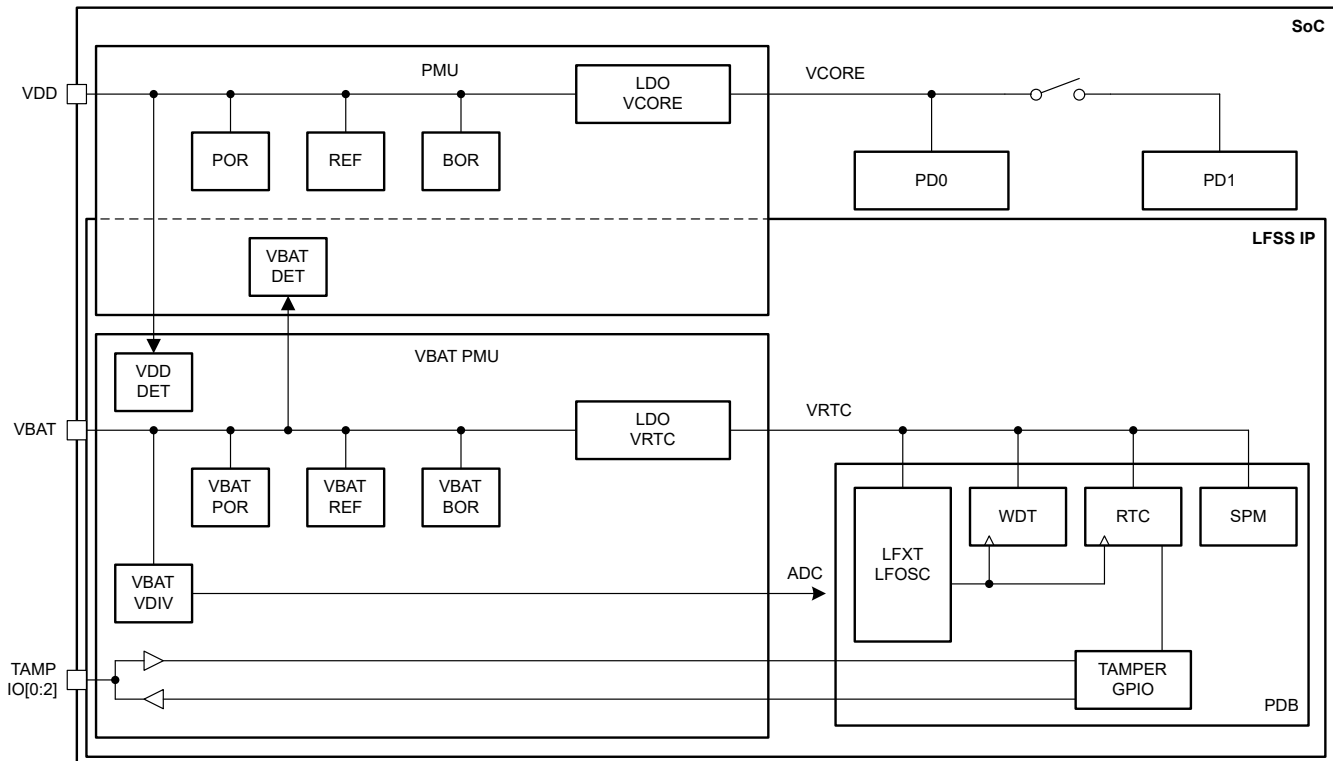


Figure 2-1. LFSS Block Diagram

2.1 Resetting LFSS IP Using VBAT

The LFSS IP has a reset generation circuit, which is implemented in the VBAT domain. The two related subcircuits that are related to the reset of LFSS are the power-on reset (POR) and the brownout reset (BOR). The POR is a V_{th} based supply voltage detector on the VBAT domain that is used to reset the power management unit (PMU) and start-up the cold boot sequence. The BOR is a reference-based voltage monitor that enables the LDO when the VBAT supply is large enough to operate the LDO safely. On initial VBAT supply power up and the enable of VRTC LDO, the VRTC detects an initial reset. Once the reset is deasserted, the VRTC domain does not detect another reset unless the power supply drops below the BOR level. The POR and BOR does not reset VRTC domain.

2.2 Power Domain Supply Detection

The LFSS IP needs to monitor the presence of the VBAT supply and the VDD supply independently as there was no dedicated always-on domain that can indicate which domain is powered and which is not. The VDD and VBAT can be powered with different voltage levels. For example, the system on chip (SoC) can be powered by a

1.8V LDO from the system LDO on the PCB, while VBAT is supplied from a 3V coin cell battery. Alternatively, in the other direction the SoC can be powered by the LDO system with 3.3V.

Figure 2-1 shows that the power domain sensing, isolation, and reset generation are very similar for both power domains. Each domain has a POR and BOR circuit. The isolation and reset signal for the sub-power domain (VCORE, VRTC) are generated by a reset latch - set latch (RS-latch). The isolation and reset signal are actively asserted (SET) with the detection of a POR or BOR event. The RS-latch is cleared once the subdomain voltage comparator gives an OK signal for the corresponding domain.

2.2.1 Start-Up Sequences

When the VBAT domain is supplied first on a unpowered SoC, the LFSS IP starts with the asynchronous sequence of the VBAT PMU. The release of the VBAT POR is around 0.9V which starts the VBAT REF circuit. When the OK signal is asserted, the signal enables the BOR circuit. Once the BOR indicates the VBAT has surpassed the minimum operating supply voltage at 1.62V, the LDO-VRTC is enabled. Once the VRTC domain has reached 1.35V, the comparator indicates a good status which releases the reset from the VRTC domain. After the reset, the LFOSC starts a 32kHz clock to operate the VBAT REF and BOR circuit in sampled mode. The LFSS IP power consumption in this state is expected to be below the specification limit to provide a 10-year lifetime from a coin cell battery.

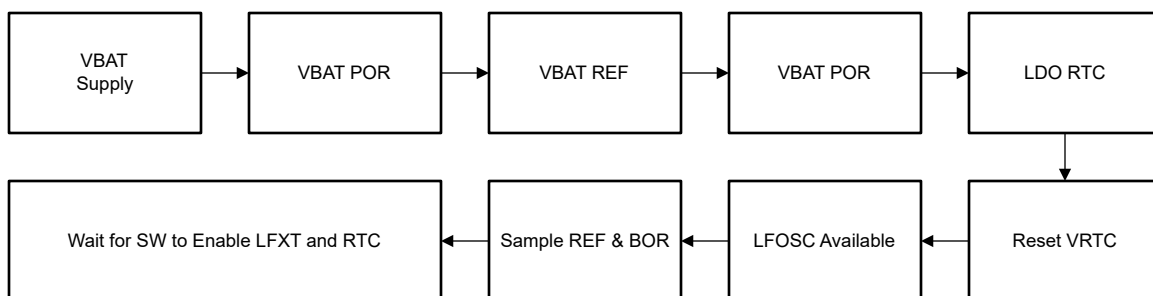


Figure 2-2. Start-Up Sequence When VBAT Comes First

When the VDD domain is supplied first, then the SoC starts with the asynchronous sequence of the PMU. This is the same start-up sequence as the SoC without LFSS IP. The difference is the unavailability of the LFCLK for use in the SoC. The VBAT-detector indicates VBAT is not present yet. In this case, the SoC cannot go into STANDBY mode as 32kHz is required for the STANDBY mode.

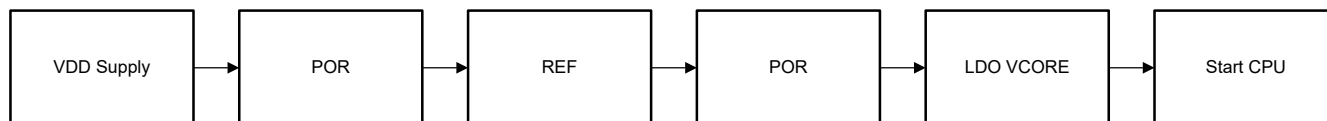


Figure 2-3. Start-Up Sequence When VDD Comes First

2.2.2 LFSS IP Behavior

The power loss of VDD is not going to happen instantaneously due to the required system-level decoupling capacitors. As VDD supply ramps down and reaches the BOR level (1.62V), the VCORE LDO gets disabled and VCORE_ISO latch gets set. Eventually the VDD level drops below the POR level and resets the VDD domain of the SoC. This is detected by the VDD detect circuit which is powered by the VBAT domain.

The shutdown mode sequence is different since the VDD domain stays powered. The shutdown mode is initiated by software and is stored in the shutdown mode register in the VDD domain. This disables the LDO and sets the VCORE_ISO latch. On a wakeup from shutdown mode, the SoC PMU reenables the REF system, BOR, and the LDO.

There are certain minimum decoupling capacitors required on the VBAT supply pin, mainly for noise immunity and to supply peak currents during switching activity on a high-impedance power source. Due to the required decoupling capacitor, the loss of the VBAT supply happens in a ramp. Due to the ramp, the VBAT-BOR circuit detects the loss of the VBAT domain first and then sets the VRTC_ISO latch. Meanwhile the circuit disables the

VBAT-PMU, the VRTC_ISO isolation signal transverses through the VDD domain and isolates all signals from the VRTC domain.

2.3 LFXT, LFOSC

LFXT and LFOSC are external and internal low-frequency crystal oscillators with a typical frequency of 32kHz. The LFCLK control bits are located inside SYSCCTL registers and user control signals are latched in a shadow register which allows the LFCLK configuration to be active during VDD, VCORE power loss. In all modes, LFCLK only functions while the VBAT domain is powered. Figure 2-4 shows the flow of how LFXT and LFOSC function with other peripherals.

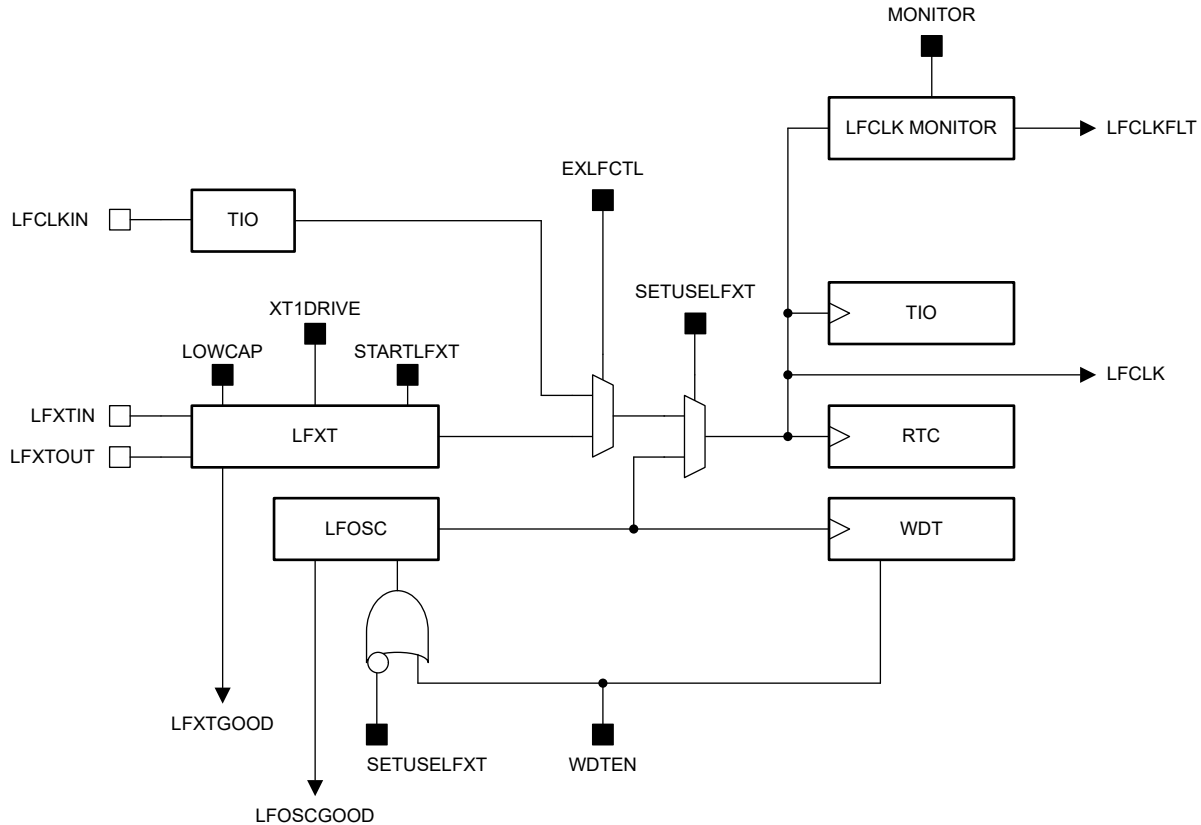


Figure 2-4. LFXT, LFOSC Flow Chart

2.4 Independent Watchdog Timer (IWDT)

The independent watchdog timer (IWDT) within LFSS is a device-independent supervisor which monitors the code execution and overall hang up scenarios. The IWDT is a 25-bit counter with closed and open windows and driven from the LFOSC with a clock path of 32kHz as shown in the Figure 2-5. The IWDT has eight selectable watchdog timer periods. The purpose of IWDT is to replace an external watchdog timer in the system and some safety applications require monitoring by an IWDT. The IWDT primary function is to initiate a full power reset when the timer detects operation failure of the device due to an unexpected software or system delay.

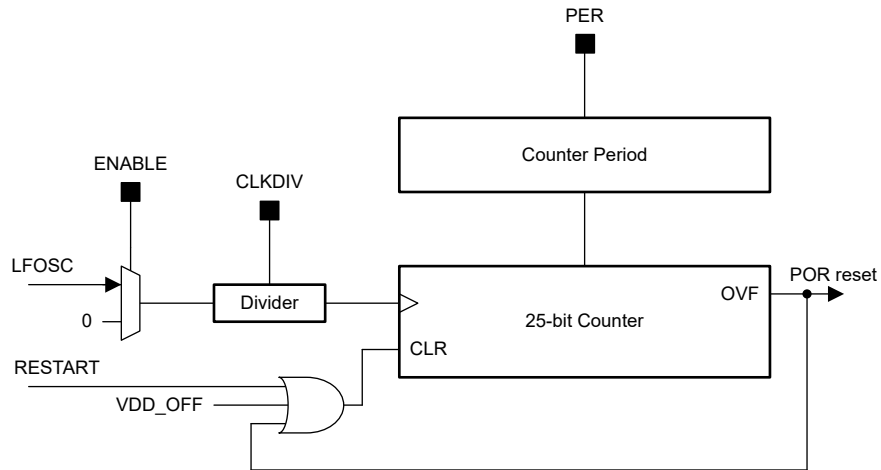


Figure 2-5. IWDT Flow Chart

2.5 Tamper I/O

Tamper I/O is a general purpose input/output (GPIO) sourced using the VBAT domain and with two modes of operation in LFSS. The output of the I/O pin is able to drive a low-current LED from the VBAT domain. The tamper input and output are used to detect tamper events such as a voltage transition on the input pin or an output that can enable or disable peripherals. Tamper I/O is often used in Alarm control systems, Microsoft® Windows® security, and so forth.

2.5.1 IOMUX Mode

Different peripherals have different signals and IOMUX mode allows the user to set the peripheral to a specific signal. [Figure 2-6](#) shows the IOMUX mode is also the default mode for LFSS and is mainly useful when both VDD and VBAT are shorted or when LFSS is used as a secondary supply to operate on a secondary function at a different voltage level. One example of IOMUX mode is that using the device which is powered by 3V for the full analog performance and use LFSS act as a internal level shifter to operate serial-peripheral interface (SPI) connection to a CPU that requires 1.8V. Some limitations with IOMUX mode include the following:

- I/O not functional when losing the main supply or when the device is in SHUTDOWN mode
- Does not allow a wakeup of the device from SHUTDOWN mode
- When VCORE is not available; therefore, Tamper I/O is not available either

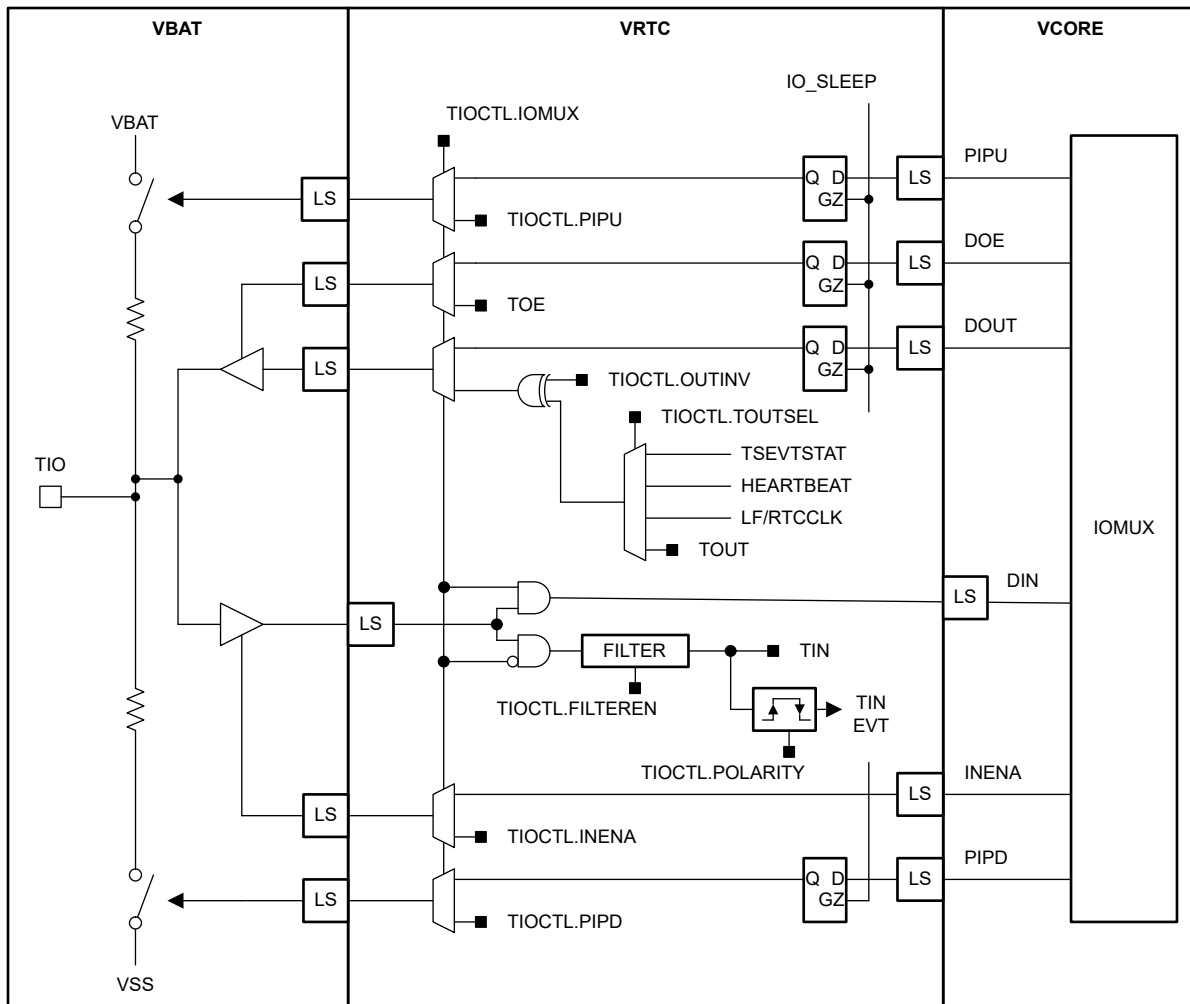


Figure 2-6. IOMUX Mode Flow Chart

2.5.2 Tamper Mode

In Tamper mode, all controls and circuits are powered by the internal VRTC domain and the controls and circuits remain functional when power is lost or when the device is in SHUTDOWN mode. Users are able to configure Tamper I/O as input or output mode. Tamper Mode also supports several internal secondary functions.

2.5.2.1 Tamper Event Detection

Tamper Event Detection allows the developer to configure one or more Tamper I/Os to trigger a timestamp to generate an interrupt to the CPU. The configured Tamper I/O needs to be an input to act as a tamper event trigger. To prevent false triggers, a digital filter circuit can detect whether the filter width is the same as configured. The input filter width can be selected from none, 30µs, 100µs, and 200µs.

2.5.2.2 Timestamp Event Output

Timestamp and event output triggers by edge detection or power loss detection of main power supply. This feature captures the RTC state as timestamp of first and last occurrence of the event. TSCTL register controls able to identify whether the first or last event is captured. For a better visibility, users able to configure Tamper I/O to display the status outside the world via I/O pin. It stays active until timestamp is cleared by Software register TSCTL.TSCLR. Timestamp applications can be found in real life applications such as Network Security Log, Database Management and etc.

2.5.2.3 Heatbeat Generator

The heartbeat generator allows the signaling of certain LFSS operating states to be visible for the users. The generator can be used as a trigger for an external watchdog to indicate whether RTC or another application

is still functioning by flashing the LED. E-metering is an example of utilizing Tamper I/O to trigger an event which is captured by the Timestamp. The event that is captured by Timestamp is signaling through the heartbeat generator to notify the user because an event was detected. In the diagram, RT2PS, RT1PS, and RT0PS are the counters that propagate through to divide down and set the heartbeat speed for the Heartbeat Generator. The users are able to set the HBINTERVAL in between 125ms, 250ms, 500ms, 1s, 2s, 4s, 8s, and 16s. The user is also able to set the HBWIDTH in between 1ms to 128ms, in binary steps. [Figure 2-7](#) shows the full diagram flow.

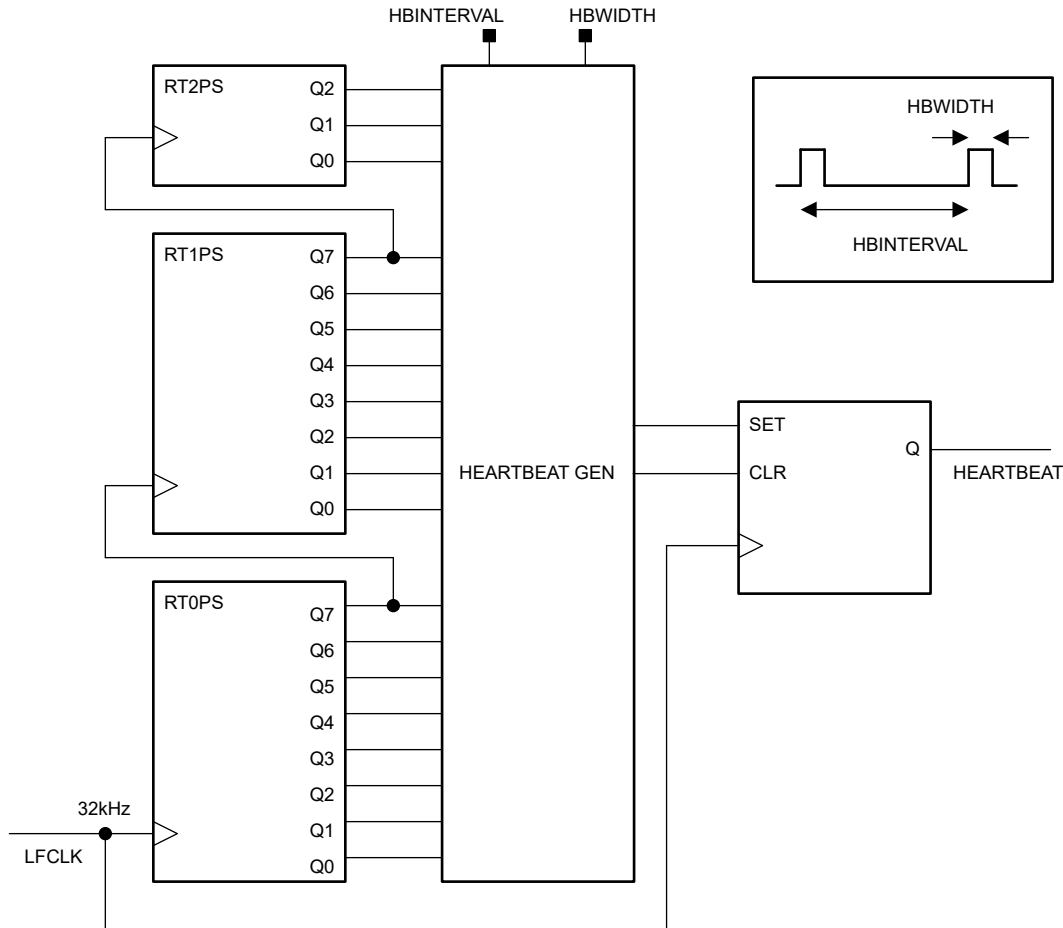


Figure 2-7. Heartbeat Generator Diagram

2.6 Scatchpad Memory (SPM)

SPM contains the 16 to 256-byte register-based memory and retains data like flash as long as VBAT is supplied. SPM retains memory when the main VDD power is lost or during shutdown mode. SPM does not retain any memory content if VBAT power is lost. Memory size is from 4 to 64 words of 32-bit size and each bit is addressable for read and write. [Figure 2-8](#) shows the LFSS write enable register is implemented to protect the memory from overwrite.





SPMEM0	B3	B2	B1	B0	0x000
SPMEM1	B7	B6	B5	B4	0x004
SPMEM2	B11	B10	B9	B8	0x008
SPMEM3	B15	B14	B13	B12	0x00C
SPMEM4	B19	B18	B17	B16	0x010
SPMEM5	B23	B22	B21	B20	0x014
SPMEM6	B27	B26	B25	B24	0x018
SPMEM7	B31	B30	B29	B28	0x01C
SPMWPROT0	KEY				0x100
SPMWPROT1	KEY				0x104
SPMTERASE0	KEY				0x140
SPMTERASE1	KEY				0x144

Figure 2-8. SPM Diagram

2.7 Real-Time Clock (RTC)

LFSS features an RTC module to provide time-keeping applications. With the counters, users are able to find seconds, minutes, hours, days of the week, day of the month, and year. The values can be displayed in binary or binary-coded decimal (BCD) in the LFSS registers. The RTC can also be used to set alarms based on calendar events or intervals. In the case of a tamper event or loss of VDD, the RTC performs a timestamp capture. The RTC control and calendar registers can be write-protected to prevent accidental updates. [Figure 2-9](#) shows the full RTC flow diagram.

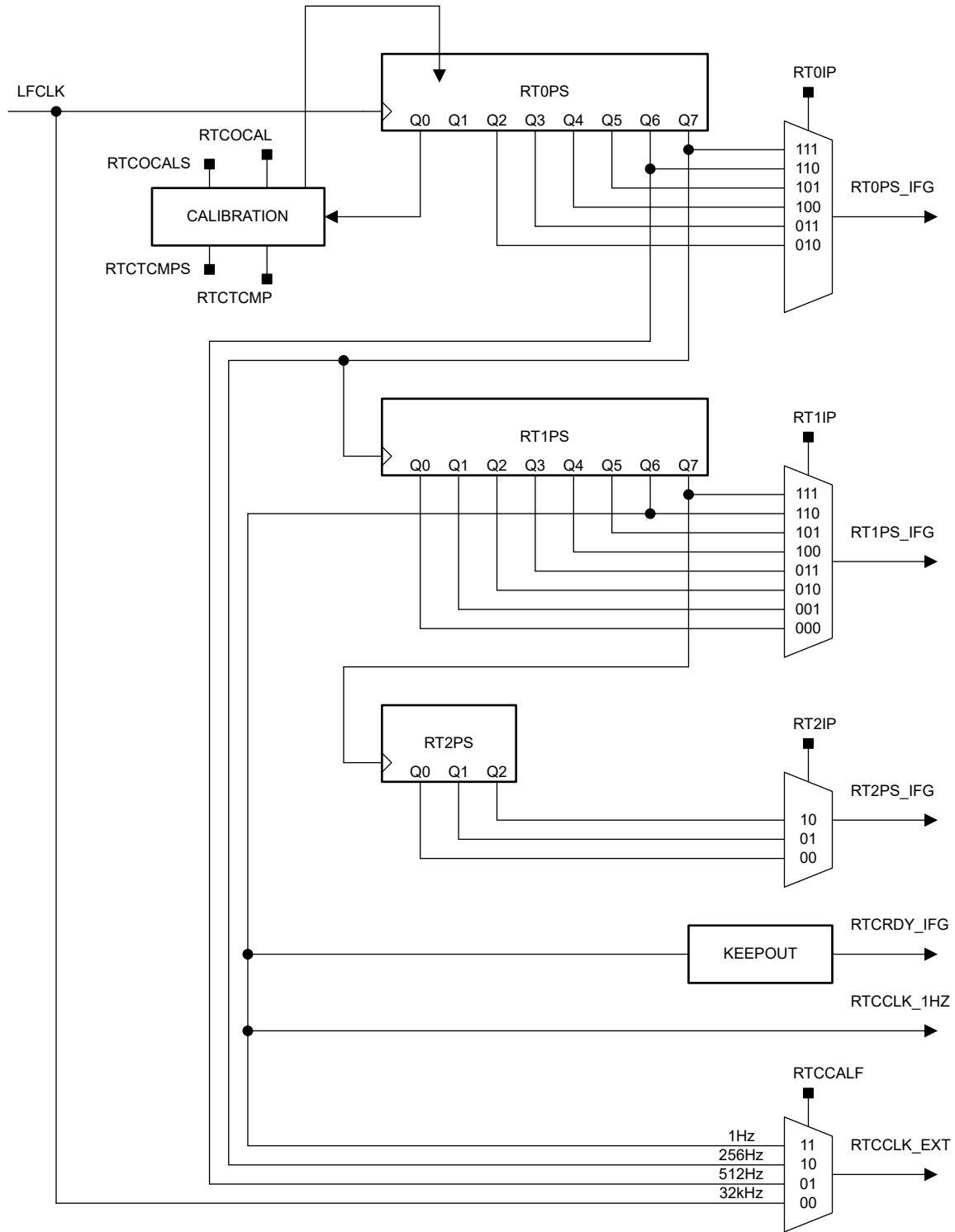


Figure 2-9. RTC Flow Diagram

2.8 VBAT Charging Mode

Table 2-1 shows the simple charging circuit can provide current-limited charging of a supercapacitor at VBAT from the primary VDD supply when VDD is greater than VBAT, while also allowing VBAT to start at time zero when VBAT is at 0V. The charging circuit needs to be enabled by software when the SUPERCAPEN bit inside the SYSTEMCFG register is set. Make sure the VBATGOOD bit in the SYSSTATUS register is set, this bit is set only when VBAT Power Domain is valid.

Table 2-1. Specifications

PARAM		VALUE	UNIT
VDD	TYP	3.3	V
R _{SWITCH}	TYP	1400	Ω
R _{SWITCH}	MAX	2700	Ω
C _{SCAP}	TYP	0.33	F
V _{BAT}	MIN	1.62	V
Maximum ICHARGE (Smallest REXT)			
PARAM		VALUE	UNIT
R _{EXT}		3300	Ω
I _{CHARGE(T0, MAX)}		1.6	mA
Charging Times			
PARAM		VALUE	UNIT
Tau		1980	s
		9900	s
Full charge time (5T)		165	min
		2.8	hrs
Run time			
PARAM		VALUE	UNIT
V _{RANGE (VDD-V_{BAT,MIN})}		1.68	V
I _{DIS}		1.5	μA
T _{DIS (Discharge life)}		369600	s
		6160	min
		102	hrs
		4.2	days

R_{EXT} and R_{SWITCH} together set the time zero maximum charge current I_{CHARGE(T0,MAX)}. R_{EXT} can be increased to reduce I_{CHARGE(T0,MAX)} at the expense of charge time, but R_{EXT} must meet a minimum value of R_{EXT(MIN)} to make sure the V_{BAT} pin detects VBOR+; therefore, the backup island starts at T₀ and does not need to wait for charging.

$$V_{BAT}(T_0) = (V_{DD} - V_{CAP}) \frac{R_{EXT}}{(R_{EXT} + R_{SWITCH})} \quad (1)$$

$$I_{CHARGE} = \frac{(V_{DD} - V_{CAP})}{(R_{EXT} + R_{SWITCH})} + I_{LEAK} \quad (2)$$

$$R_{EXT} = \frac{(V_{BAT})(R_{SWITCH})}{(V_{DD} - V_{BAT})} \quad (3)$$

$$I_{CHARGE}(T_0, MAX) = \frac{V_{DD}}{\frac{(V_{BAT})(R_{SWITCH})}{(V_{DD} - V_{BAT})} + R_{SWITCH}} \quad (4)$$

Charging is enabled by software and is automatically disabled when VDD drops below V_{BAT}. The discharge life is driven by V_{BAT} burn (I_{DIS}), R_{EXT} is minimal.

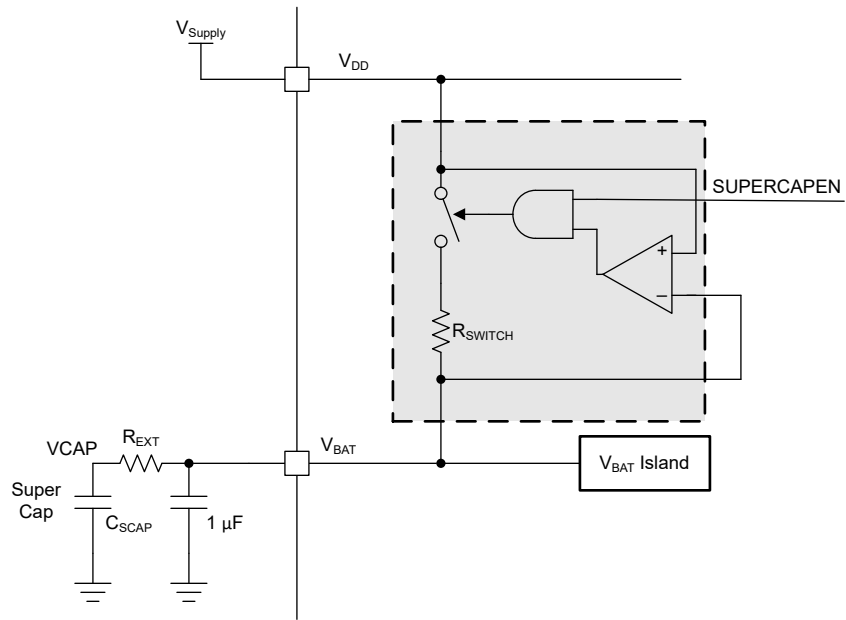


Figure 2-10. Supercapacitor Charging Circuit

3 Application Examples

This section demonstrates LFSS by using [Code Composer Studio](#) examples. There are five different examples showing how to use different peripherals within LFSS.

3.1 Tamper I/O Heartbeat Example

This example shows off the Heartbeat generator still functioning when the main supply VDD experiences power loss. This example represents a real-world example such as an alarm system that was previously set when VDD was still powered and is still functioning after VDD power is lost. For this example to function correctly, power on the VBAT domain with a cell battery or an external power supply. Configure the Tamper I/O PA7 as output to drive the LED. After the setup, run the example code to toggle the corresponding LED PB2. By disconnecting the main power supply, VDD, the LED continues flashing as long as VBAT domain is powered.

```
#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 Tamper I/O Timestamp Event Example

This example needs two Tamper I/O configurations. Configure Tamper I/O 0 as input to act as a tamper event trigger. The timestamp event output captures the event occurrence and RTC records when the event is captured. Next, configure Tamper I/O 1 as output to toggle LED for Heartbeat mode. As the Tamper I/O triggered the rising edge, the TSCTL register captures the first or last event that occurred. The configured LED remains flashing as VDD power loss to show that the heartbeat generator is still functioning and is supplied by VBAT.

```
#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 Supercapacitor Charging Example

This example uses CCS Tamper I/O heartbeat code example to demonstrate the use of a supercapacitor charging circuit. To begin with the example, change the VBAT jumper from VDD to CAP. Then in the software, enable the supercapacitor register for the supercapacitor charging circuit to function. Configure the desired Tamper I/O as an output and connect the Tamper IO to PB2 LED to perform the heartbeat generator. After the supercapacitor is fully charged up, disconnect the main power supply VDD. The supercapacitor starts charging back the VBAT domain. As the VBAT domain is powered up, the LED continues flashing after the main power supply loss to show that the supercapacitor charging circuit is working properly. The LED eventually stops flashing as the supercapacitor is completely discharged.

```

#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 Transition Back to LFXT Example

This example exhibits a problem existing in water metering where LFXT is disabled due to the humidity. The LFXT is automatically disabled and transitions to use LFOSC to keep running the low-frequency clock at 32kHz. Many users have issues transitioning back to LFXT when the LFXTGOOD bit is true. This example illustrates how to transition back to LFXT.

The Code Composer Studio provides the clock tree options allowing configuration of the desired clock path. Set LFXT as the source for LFCLK and make sure both the LFCLK start-up monitor and LFCLK monitor are enabled. This monitors the failure of LFXT to transition to LFOSC. Set the clock output to the desire pin to measure the frequency to make sure that the clock is working properly. Ground LFX_Out to disable LFXT, this action toggles the LED PA 16 to indicate that LFXT is failed and is using LFOSC. LFOSC cannot auto transition back to LFXT; therefore, the process of setting the key to enable the STARTLFXT bit to true then takes us to the IRQHandler to set the key to enable the SETUSELFXT bit to true and turn on the clock monitor again. There the processes allows a transition from LFOSC back to LFXT again when LFXT is cleared and good to use again.

```

#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 Calibration

The accuracy of internal oscillators is within percentages while external crystals are in parts per million (ppm). At the expense of higher power consumption, crystal oscillators are highly accurate due to the precise cut, shape, and size. However, the frequency the oscillators resonate at is still subject to external factors. Temperature, aging, mechanical shock and vibration, and gravity all have an effect on the accuracy of the crystal. This example shows how to calibrate the RTC of LFSS when offset and temperature-based errors are measured.

3.5.1 Peripheral ADC 12

Together with the internal temperature sensor of the MSP, the ADC is used to measure the device temperature. This is done with the following process:

1. Compute the trim voltage using the temperature coefficient (TS_C), factory trim temperature (TS_{TRIM}), and unit specific-single point trim value (in register: `TEMP_SENSE0`)
2. Sample the internal temperature sensor
3. Convert the raw ADC value to a voltage
4. Approximate the device temperature using [Equation 5](#):

$$T_{\text{Sample}} = \left(\frac{1}{T_{\text{SC}}} \right) \times (\text{ADC}_{\text{CODE}} - \text{TEMP}_{\text{SENSE0.DATA}}) + T_{\text{STRIM}} \quad (5)$$

T_{SC} and T_{STRIM} can be found within the device data sheet in the *Specifications* section. $\text{TEMP}_{\text{SENSE0}}$ is located in the factory constants section of memory. The temperature sensor is the input on a device-dependent ADC channel and the raw ADC value is stored in ADC Conversion Memory 0. The single-calibration value is the ADC result of the temperature sensor being measured with a 1.4V internal reference voltage.

3.5.2 RTC_A

Calibration of the RTC takes place over 60 seconds using the Q0 output of RT0PS. This approximates corrections of ± 1 ppm by adding or subtracting 1 clock pulse every quarter second. A maximum of ± 240 ppm can be adjusted for during one calibration cycle. Each cycle includes the adjustments for both offset and temperature errors. Adjustments exceeding the ± 240 ppm bounds saturates, but total compensation can be done over multiple cycles. Calibration is disabled if the RTC is not enabled.

There are two registers, CAL and TCMP, that are used for calibration. Offset adjustments are written to the CAL register and temperature written to TCMP. The TCMP register; however, stores the net compensation value between the two error types. When the CAL register is set, the temperature adjustment is reset to 0 and the TCMP only stores the offset adjustment.

When calibrating to adjust for offset error, the RTC offers a range of frequency outputs for external measurement. The test frequencies are LFCLK (32kHz), 512Hz, 256Hz, and 1Hz. The selection is stored in the RTCOCALFX bit field of the CAL register, and if set to 0, offset calibration is disabled. The offset error can be calculated using [Equation 6](#).

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

where

- 60 is the calibration cycle length in seconds
- 16384 is the frequency of the Q0 output of RT0PS
- f_{RTCCLK} is the measured output frequency multiplied by the clock division factor
- 32768 is the expected frequency of the LFCLK

Once the error has been calculated, the sign and magnitude bit fields can be written to. The sign indicates whether the adjustment is up or down. If the error is positive (Up calibration), set RTCOCALS, clear RTCOCALS if the sign is negative (Down calibration). RTCOCALX is the magnitude and limited to a maximum of 240ppm. If the signed result is written to RTCOCALX, negative values are represented in two's complement and saturate to 240ppm.

Temperature drift calibration is done by taking the temperature approximation from the ADC, calculating the frequency error caused by deviation from the turnover temperature of the crystal, and writing the result to the TCMP register. The temperature approximation process is described in [Section 3.5.1](#). Use [Equation 7](#) to calculate the frequency error.

$$T_{\text{COMP}} = (T_{\text{APPROX}} - T_i) \times B \quad (7)$$

where

- T_{COMP} is the temperature compensation value
- T_{APPROX} is the measured temperature approximation
- T_i is the crystal oscillator turnover temperature
- B is the crystal oscillator parabolic coefficient

The previous information is found in the Crystal Oscillator data sheet.

This equation changes based on what is generating the clock signal. Specifically, the turnover temperature and parabolic coefficient variables are dependent on the type of oscillator used. In this example a tuning fork external crystal is used and the turnover temperature and parabolic coefficient are found in the data sheet. A unique

property of a tuning fork crystal is that temperature deviation results in a negative frequency deviation. This means the direction of compensation is up. Once the error has been calculated, similar to the offset error, the sign (Up) is written to the RTCTCMPS bit field and the magnitude written to the RTCTCMPX bit field. After the write is successful, when reading the TCMP register, it has stored the net compensation between the offset error and temperature deviation. If the net compensation is too large, RTCTCMPX saturates to 240ppm. The compensation determined by temperature deviation is introduced in the next calibration cycle, not the current. Because each cycle is 60 seconds long, this example only measures the temperature once a minute and is triggered by the RTC_A interval interrupt. This can be changed to measure temperature multiple times and take the average.

```
#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_TCOMP_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 TCOMP register
    * should be updated.
    * The TCOMP 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 RTCCAL 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;
    }
}

```

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#) or other applicable terms available either on [ti.com](https://www.ti.com) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2024, Texas Instruments Incorporated