AN14122

How to use RTC on KW45-EVK

Rev. 1.0 — 26 December 2023 Application note

Document information

Information	Content
Keywords	AN14122, KW45-EVK, RTC, low power, K32W1, clocking, interrupt
Abstract	This document explains the process for integrating RTC feature into a wireless low-power demo.

1 Introduction

A real-time clock (RTC) is a powered block that remains active in all low-power modes and is powered by the battery power supply (VBAT). The battery power supply ensures that the RTC registers retain their state during chip power down and the RTC time counter remains operational.

The time counter within the RTC is clocked by default from a 32.768 kHz clock and can supply this clock to other peripherals. The 32.768 kHz clock can be sourced from an external crystal using the oscillator that is part of the RTC module.

The RTC includes an analog power-on reset (POR) block, which generates a VBAT power-on reset signal whenever the RTC module is powered up and initializes all RTC registers to their default state. Software reset bit can also initialize all RTC registers. The RTC also monitors the chip power supply and electrically isolates itself when the rest of the chip is powered down.

2 Acronyms

Table 1 lists the acronyms used in this document.

Table 1. Acronyms

Acronym	Meaning
RTC	Real-time clock
VBAT	Voltage battery
POR	Power-on reset
CCM32K	32 kHz clock control module
TAR	Time alarm register

3 Functional description

The RTC remains functional in all low-power modes and generates an interrupt for the application processor to exit any low-power mode.

During chip power down, the RTC is powered from the VBAT and is electrically isolated from the rest of the chip. However, the RTC continues to increment the time counter (if enabled) and retain the state of the RTC registers. The RTC registers are not accessible.

During chip power up, RTC remains powered from the VBAT. All RTC registers are accessible by software and all functions are operational. If enabled, the 32.768 kHz clock can supply the rest of the chip.

4 RTC signal

The `RTC_CLKOUT` signal can output either a square wave prescaler output or the RTC 32.768 kHz clock. The square wave prescaler output is configurable to 1, 2, 4, 8, 16, 32, 64, or 128 Hz.

The RTC wake-up pin is an open drain, active low output that allows the RTC to wake up the chip via an external component. The wake-up pin asserts when the wake-up pin enabler is set. Either the RTC interrupt is asserted, or the wake-up pin is turned on via a register bit. The wake-up pin does not assert from the RTC interrupt in seconds.

Table 2. RTC signal descriptions

Signal	Description
`EXTAL32`	32.768 kHz oscillator input
`XTAL32`	32.768 kHz oscillator output
`RTC_CLKOUT`	Prescaler square-wave output or RTC 32.768 kHz clock
`RTC_WAKEUP_b`	Active low wake for external device
`RTC_TAMPER[3:0]`	Tamper pin input

5 Clocking

The `FRO-32K` and the `OSC-32K` clocks are generated in a separate 32 kHz Clock Control Module (`CCM32K`) within a different power domain. The different power domains are powered independently, allowing one of these clock sources (`FRO-32K` or `OSC-32K`) to clock the RTC module (also in this separate power domain). The input to the SCG from this separate power domain is called `32K_CLK`.

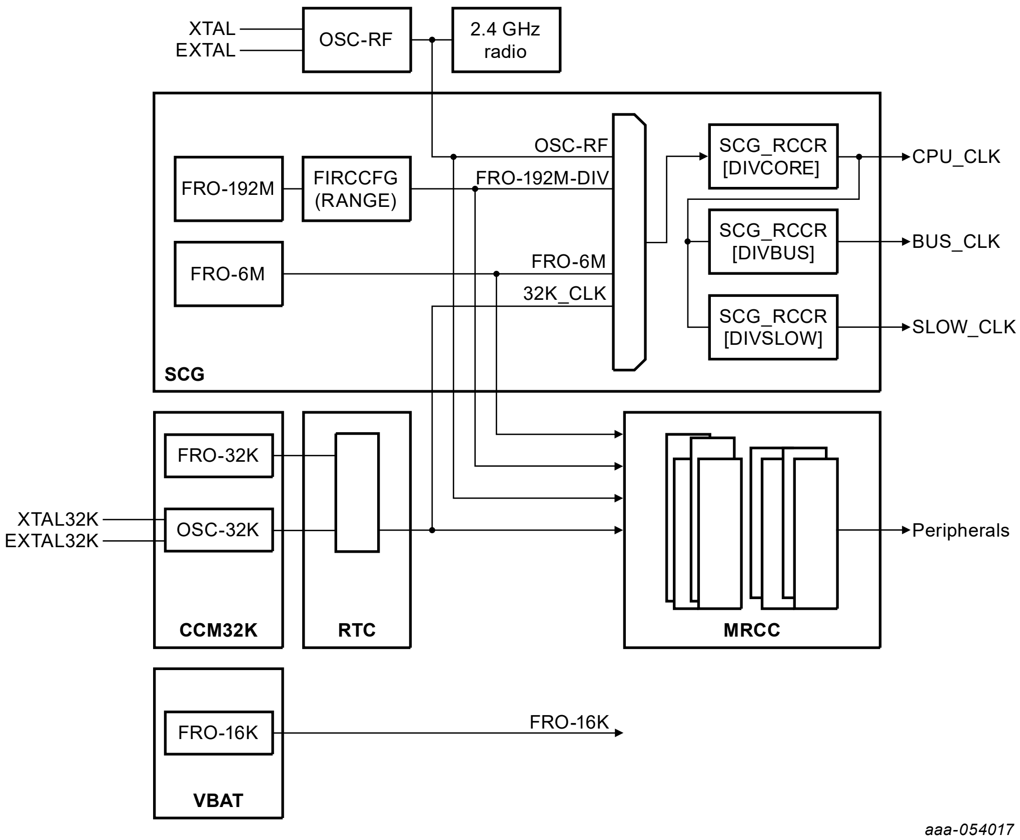

Figure 1. Device clock sources

The `CCM32K` module powers up, loads the default trims from flash, and starts the FRO after POR is complete. To configure the module further and clock the RTC module, software must perform the following steps:

```
ccm32k_osc_config_t osc32kConfig = {
        .enableInternalCapBank = true,
        .xtalCap               = kCCM32K_OscXtal0pFCap,
        .extalCap              = kCCM32K_OscExtal16pFCap,
        .coarseAdjustment      = kCCM32K_OscCoarseAdjustmentRange0,
    };
```

```
    CCM32K_Set32kOscConfig(CCM32K, kCCM32K_Enable32kHzCrystalOsc, &osc32kConfig);
    CCM32K_SelectClockSource(CCM32K, kCCM32K_ClockSourceSelectOsc32k);
```

6 Time alarm and interrupts

The time alarm register (TAR), SR[TAF], and IER[TAIE] allow the RTC to generate an interrupt at a predefined time. The RTC interrupt is asserted whenever a status flag and the corresponding interrupt enable bit are set.

Note: *The RTC interrupt is always asserted even when on VBAT POR or if there is a software reset, or when the VBAT power supply is powered down.*

The RTC interrupt is enabled at the chip level by enabling the chip-specific RTC clock gate control bit. The RTC interrupt can be used to wake up the chip from any low-power mode. To configure the time alarm and the interrupt further, software must perform the following steps:

```
/* Enable RTC alarm interrupt */
    RTC_EnableInterrupts(RTC, kRTC_AlarmInterruptEnable);

    /* Enable at the NVIC */
    WUU_SetInternalWakeUpModulesConfig(APP_WUU, 0x6, kWUU_InternalModuleInterrupt);
    EnableIRQ(RTC_IRQn);
After the alarm occurs it necessary to write the IER register to enable the software
 interrupt
        RTC->IER = RTC_IER_TAIE(0x01);
```

7 Integrating the RTC to a low-power application

This section explains how to integrate the RTC feature to the low-power demo to wake up the chip from low power using only the RTC interrupt.

7.1 Prerequisites

This document includes a functional demo using the RTC in low power. The example is based on the Power mode switch project. This project is available in the KW45B41Z/K32W148 SDK package and developed on the MCUXpresso IDE platform. To complete the implementation of the RTC low-power integration demo, the following prerequisites are required:

- MCUXpresso SDK Builder v11.7.0 or later
- KW45B41Z/K32W148 SDK v.2_12_5
- Low-power reference design demo package
- KW45B41Z/K32W148 board

7.2 Downloading and installing the software development kit

This section provides the steps required to download the KW45B41Z-EVK SDK package to begin with the process. For more details, refer to the Getting Started with the KW45B41Z Evaluation Kit.

To download and install the SDK package for the KW45B148-EVK, perform the following steps:

1. Navigate to the MCUXpresso website.
2. Click **Select Development Board**.
3. Log in with your registered account.
4. In the **Search for Hardware** field, search for "KW45B41-EVK/K32W148-EVK".
5. Select the suggested board and click **Build MCUXpresso SDK**.

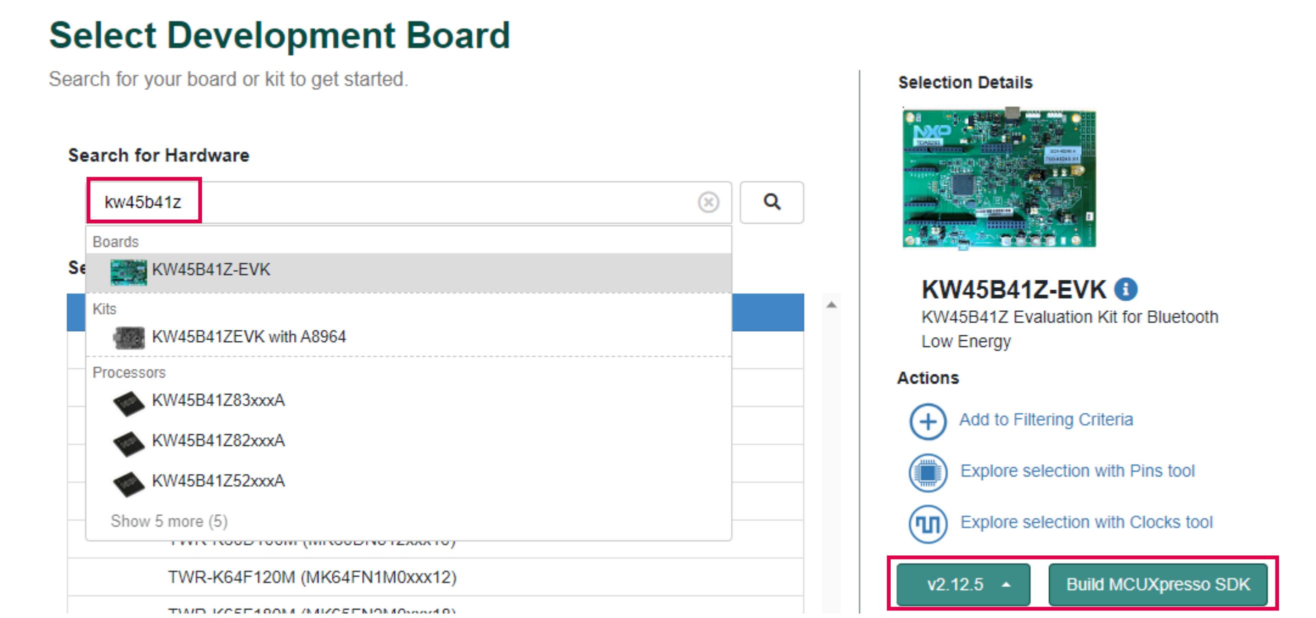

Figure 2. Select Development Board

6. Select "MCUXpresso IDE" in the Toolchain/IDE combo box. Select the supported OS. Click **Download SDK** and the system takes a few minutes to get the package into your account on the MCUXpresso webpage. Read and accept the license agreement. The SDK download starts automatically on your PC.

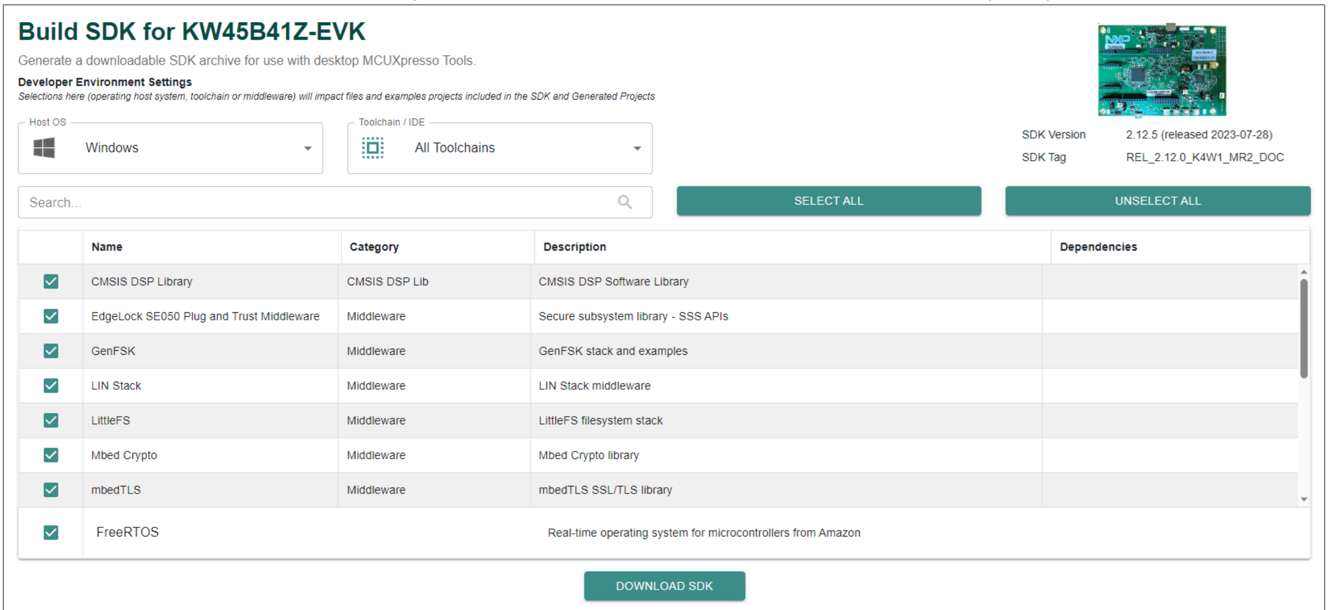

Figure 3. Downloading SDK for KW45B41Z-EVK

7. Open MCUXpresso IDE. Drag and drop the "KW45B41-EVK SDK zip" in the **Installed SDKs** list.

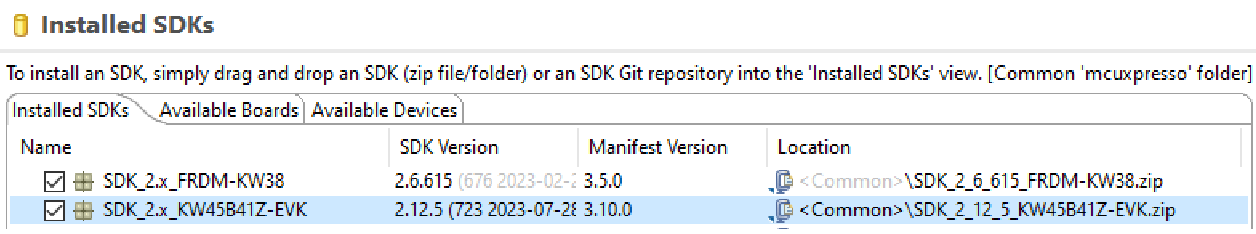

Figure 4. MCUXpresso Installed SDKs

Now, the SDK package for the KW45B148-EVK is downloaded and installed.

7.3 Import the power mode switch demo

To import the power mode switch demo, perform the following steps:

1. Select the demo that you want to use.
2. Select **demo_apps > power_mode_switch_k4**.
3. Click the **Finish** button.

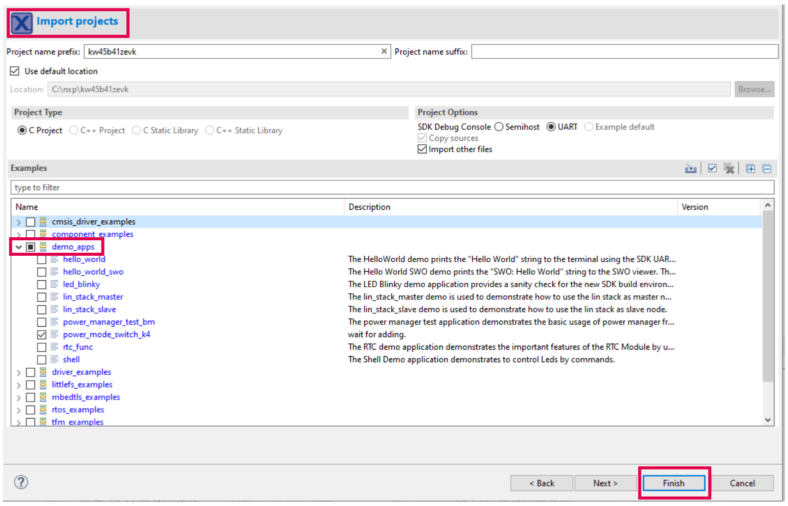

Figure 5. Import project from the SDK package

7.4 Main modifications in the source files

Once the RTC drivers files are included in the custom project, perform the following steps:

1. Right-click the **Project folder > SDK Management > Manage SDK components**.

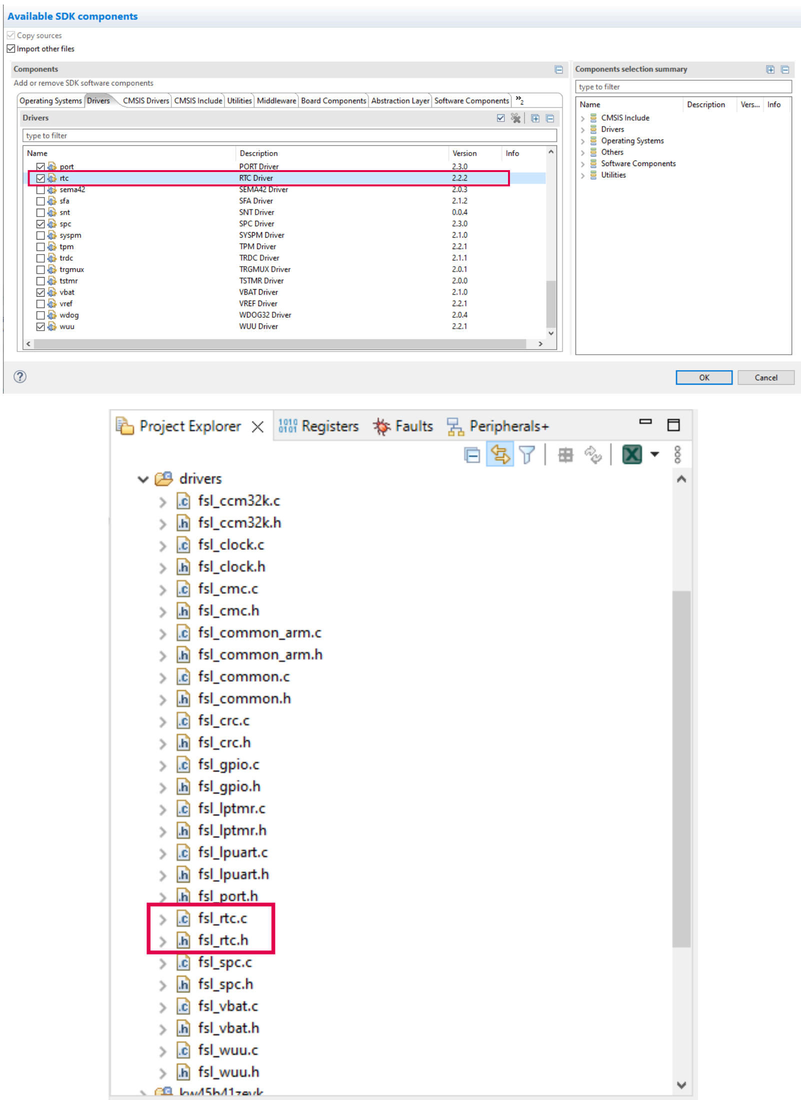

Figure 6. RTC drivers

2. To enable the RTC in low-power modes, add the required configurations.

The following sections explain the main aspects that the user must focus on.

7.4.1 pin_mux.c

To obtain the desired RTC output signal, set the right pins. For example, PTD2, PTD3, and PTD4. This project uses the PTD3 as TAMPER1.

Open the `pin_mux.h` file located in the board folder.

To set the necessary pin, add the function as follows:

```
void BOARD_InitPinsRTC(void)
{
        const port_pin_config_t portd3_pin26_config = {/* Internal pull-up/down resistor
 is disabled */
                                                (uint16_t)kPORT_PullUp,
/* Low internal pull resistor value is selected. */
                                                (uint16_t)kPORT_LowPullResistor,
```

```
/* Fast slew rate is configured */
                                                (uint16_t)kPORT_FastSlewRate,
/* Passive input filter is disabled */
                                                (uint16_t)kPORT_PassiveFilterDisable,
/* Open drain output is disabled */
                                                (uint16_t)kPORT_OpenDrainDisable,
/* Low drive strength is configured */
                                                (uint16_t)kPORT_LowDriveStrength,
/* Normal drive strength is configured */
                                                (uint16_t)kPORT_NormalDriveStrength,
/* Pin is configured as TAMPER1 */
                                                   (uint16_t)kPORT_MuxAlt3,
/* Pin Control Register fields [15:0] are not locked */
                                                   (uint16_t)kPORT_UnlockRegister};
/* PORTD3 (pin 26) is configured as TAMPER1 */
    PORT_SetPinConfig(PORTD, 3U, &portd3_pin26_config);
}
```

7.4.2 power_mode_switch.c

To configure the RTC, add the declarations and variables to the `power_mode_switch.c` file, as follows:

```
#include "fsl_rtc.h"    //include the driver in the main file
//add the necessary variables and prototypes
#define RTC_IRQn       RTC_Alarm_IRQn
#define RTC_IRQHandler RTC_Alarm_IRQHandler
#define EXAMPLE_OSC_WAIT_TIME_MS 1000UL
void config_RTC(void);
void set_time_RTC(void);
```

Also, if the application requires to call the RTC, it is necessary to declare and create the function to configure the RTC and the interruption:

```
void RTC_IRQHandler(void)
{
    uint32_t status = RTC_GetStatusFlags(RTC);

    if (status & kRTC_AlarmFlag)
    {
        busyWait = false;

        /* Clear alarm flag */
        RTC_ClearStatusFlags(RTC, kRTC_AlarmInterruptEnable);
    }
    else if (status & kRTC_TimeInvalidFlag)
    {
        /* Clear timer invalid flag */
        RTC_ClearStatusFlags(RTC, kRTC_TimeInvalidFlag);
    }
    else
    {
    }
    SDK_ISR_EXIT_BARRIER;
}
```

```
void config_RTC(void)
{

  rtc_config_t rtcConfig;

  PRINTF("RTC Init\r\n");
```

```
  BOARD_InitPinsRTC();

    ccm32k_osc_config_t osc32kConfig = {
        .enableInternalCapBank = true,
        .xtalCap               = kCCM32K_OscXtal0pFCap,
        .extalCap              = kCCM32K_OscExtal16pFCap,
        .coarseAdjustment      = kCCM32K_OscCoarseAdjustmentRange0,
    };
    CCM32K_Set32kOscConfig(CCM32K, kCCM32K_Enable32kHzCrystalOsc, &osc32kConfig);
    CCM32K_SelectClockSource(CCM32K, kCCM32K_ClockSourceSelectOsc32k);

    RTC_GetDefaultConfig(&rtcConfig);
    RTC_Init(RTC, &rtcConfig);


    RTC->CR |= RTC_CR_CPE(0x01);
    RTC->CR |= RTC_CR_CPS(0x1);

    /* Set a start date time and start RT */
    date.year   = 2014U;
    date.month  = 12U;
    date.day    = 25U;
    date.hour   = 19U;
    date.minute = 0;
    date.second = 0;

    /* RTC time counter has to be stopped before setting the date & time in the TSR
 register */
    RTC_StopTimer(RTC);

    /* Set RTC time to default */
    RTC_SetDatetime(RTC, &date);

    /* Enable RTC alarm interrupt */
    RTC_EnableInterrupts(RTC, kRTC_AlarmInterruptEnable);

    /* Enable at the NVIC */
    WUU_SetInternalWakeUpModulesConfig(APP_WUU, 0x6, kWUU_InternalModuleInterrupt);
    EnableIRQ(RTC_IRQn);

    /* Start the RTC time counter */
    RTC_StartTimer(RTC);

}
```

```
void set_time_RTC(void)
{

      uint32_t sec;
      uint32_t currSeconds;
      uint8_t index;

      rtc_datetime_t date;

        busyWait = true;
        index    = 0;
        sec      = 0;
        /* Get date time */
        RTC_GetDatetime(RTC, &date);

        /* Get alarm time from user */
        PRINTF("\n\nPlease input the number of second to wait for alarm \r\n");
        PRINTF("The second must be positive value\r\n");
        while (index != 0x0D)
        {
            index = GETCHAR();
```

```
            if ((index >= '0') && (index <= '9'))
            {
                PUTCHAR(index);
                sec = sec * 10 + (index - 0x30U);
            }
        }
        PRINTF("\r\n");

        /* Read the RTC seconds register to get current time in seconds */
        currSeconds = RTC->TSR;

        /* Add alarm seconds to current time, because RTC alarm will happen when RTC->TAR
 = RTC->TSR and RTC->TSR
        increments, thus there's possible 1 second maximum delay here. */
        currSeconds += sec;

        /* Set alarm time in seconds */
        RTC->TAR = currSeconds;

        /* Get alarm time */
        RTC_GetAlarm(RTC, &date);

        RTC->IER = RTC_IER_TAIE(0x01);
}
```

The purpose of using the RTC is to start counting before the low power. Therefore, it is necessary to call the initialization function in the main() right before the low-power functionality begins:

```
void main(void)
{

        uint32_t freq;
        cmc_low_power_mode_t curmode;
        bool needSetWakeup = false;

        BOARD_InitPins();
        BOARD_BootClockRUN();
        BOARD_InitDebugConsole();
        BOARD_InitBootPeripherals();

        CLOCK_DeinitSys0sc();
        CLOCK_DeinitSirc();
        APP_SetSPCConfiguration();
        config_RTC();                        //RTC initialization
```

Call the counting function, where the RTC gets the desired time and the interrupt, before the KW45 goes to low-power mode:

```
static void APP_PowerModeSwitch(app_power_mode_t targetPowerMode)
{
    if (targetPowerMode != kAPP_PowerModeActive)
    {
        switch (targetPowerMode)
        {
            case kAPP_PowerModeSleep1:
             set_time_RTC();
                APP_EnterSleep1Mode();
                break;

            case kAPP_PowerModeDeepSleep1:
             set_time_RTC();
                APP_EnterDeepSleep1Mode();
                break;
```

```
            case kAPP_PowerModePowerDown1:
             set_time_RTC();
                APP_EnterPowerDown1Mode();
                break;

            case kAPP_PowerModeDeepPowerDown1:
             set_time_RTC();
                APP_EnterDeepPowerDown1Mode();
                break;

            case kAPP_PowerSwitchOff:
                SPC_PowerModeControlPowerSwitch(APP_SPC);
                APP_EnterDeepPowerDown1Mode();
                break;
            default:
                assert(false);
                break;
        }
    }
}
```

To obtain the RTC signal as an output in the PTD3, write this register:

```
void config_RTC(void)
{

  rtc_config_t rtcConfig;

  PRINTF("RTC Init\r\n");
  BOARD_InitPinsRTC();

    ccm32k_osc_config_t osc32kConfig = {
        .enableInternalCapBank = true,
        .xtalCap               = kCCM32K_OscXtal0pFCap,
        .extalCap              = kCCM32K_OscExtal16pFCap,
        .coarseAdjustment      = kCCM32K_OscCoarseAdjustmentRange0,
    };
    CCM32K_Set32kOscConfig(CCM32K, kCCM32K_Enable32kHzCrystalOsc, &osc32kConfig);
    CCM32K_SelectClockSource(CCM32K, kCCM32K_ClockSourceSelectOsc32k);

    RTC_GetDefaultConfig(&rtcConfig);
    RTC_Init(RTC, &rtcConfig);


    RTC->CR |= RTC_CR_CPE(0x01);
    RTC->CR |= RTC_CR_CPS(0x1);

    /* Set a start date time and start RT */
    date.year   = 2014U;
    date.month  = 12U;
    date.day    = 25U;
    date.hour   = 19U;
    date.minute = 0;
    date.second = 0;

    /* RTC time counter has to be stopped before setting the date & time in the TSR
 register */
    RTC_StopTimer(RTC);

    /* Set RTC time to default */
    RTC_SetDatetime(RTC, &date);

    /* Enable RTC alarm interrupt */
    RTC_EnableInterrupts(RTC, kRTC_AlarmInterruptEnable);
```

```
    /* Enable at the NVIC */
    WUU_SetInternalWakeUpModulesConfig(APP_WUU, 0x6, kWUU_InternalModuleInterrupt);
    EnableIRQ(RTC_IRQn);

    /* Start the RTC time counter */
    RTC_StartTimer(RTC);

}
```

8 RTC functional

The RTC remains functional in all low-power modes and can generate an interrupt to exit any low-power mode.

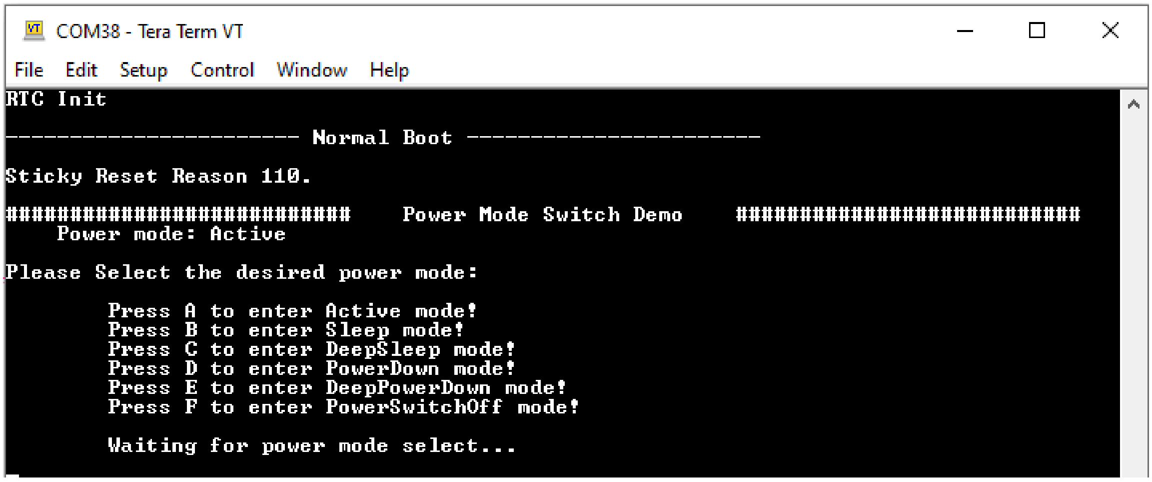

Figure 7. Demo interface

Appreciate the low-power functionality in Figure 8 indicating the changes in the current when the RTC interrupt occurs. When the RTC interrupts the MCU, it returns to the Active mode. The user can see the current behavior by measuring in the JP5 pin [3-4].

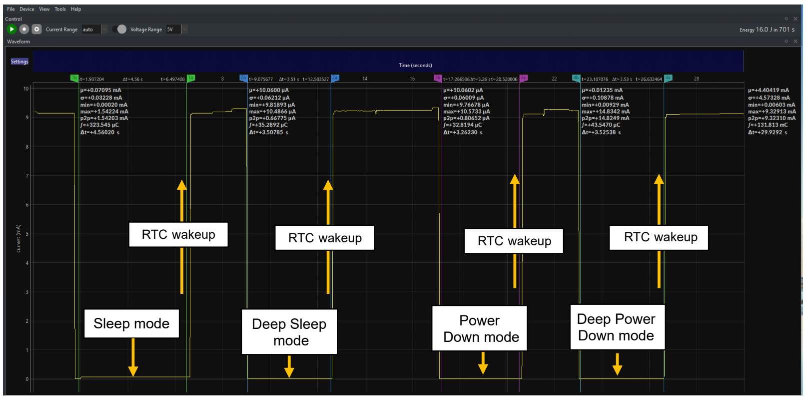

Figure 8. Current behavior

Figure 9 shows the RTC signal as an output using the TAMPER pin (J4 pin [3]).

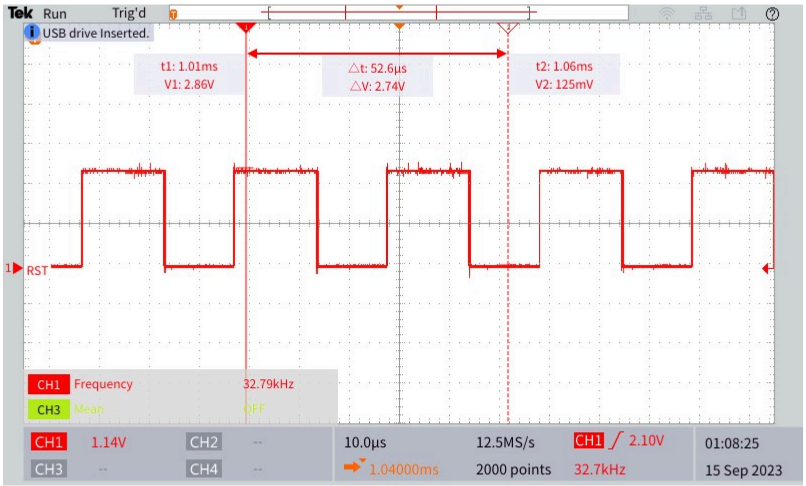

Figure 9. RTC frequency

9 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2023 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

10 Revision history

Table 3 summarizes the revisions to this document.

Table 3. Revision history

Document ID	Release date	Description
AN14122 v.1	26 December 2023	Initial public release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents

Date of release: 26 December 2023
Document identifier: AN14122