Real-time clock (RTC) usage in XMC7000 family

About this document

Scope and purpose

This application note describes how to use the real-time clock (RTC) in XMC7000 family MCUs.

Intended audience

This document is intended for anyone who uses the RTC in XMC7000 family MCUs

Associated part family

XMC7000 family of XMC™ industrial microcontrollers

Table of contents

1 Introduction

The RTC in the XMC7000 family has four features: RTC, Alarm, Calibration, and Backup registers.

The RTC feature keeps track of the current time, year, month, date, day-of-week, hour, minute, and second accurately. The alarm feature generates interrupts to the CPU with a programable setting. The calibration feature corrects a frequency error of the WCO. The backup registers keep user data in any power mode.

This application note:

- Explains how to update the time of the RTC registers
- Describes how to read the time from the RTC registers
- Explains the functions of RTC in series
- Shows how to set up the RTC function and calibrate the WCO

To understand the functionality and terminology used in this application note, see the “Real-Time Clock” chapter of the architecture technical reference manual (TRM).

1.1 Features of RTC

The following are the features of RTC:

- Fully-featured RTC function
  - Year/Month/Date, Day-of-Week, Hour: Minute: Second fields (all fields are integer values)
  - Supports both 12-hour and 24-hour formats
  - Automatic leap-year correction until 2400
- Configurable alarm function
  - Alarm on Month/Date, Day-of-Week, Hour: Minute: Second fields
  - Two independent alarms
- Calibration for 32.768-kHz WCO
  - Calibration waveform output
  - Supports 512 Hz, 1 Hz, and 2 Hz
- Backup registers

2 Operation overview

The RTC function keeps track of the current time accurately up to seconds. Optional alarms can be supported as well. The time information and alarm features can be used in a system that requires real-time and alarm functions. This application note describes how to get and set the RTC time.

The RTC function block consists of user registers, RTC registers, and WCO. The RTC function interfaces with the CPU and other sub-systems via the AHB-Lite bus interface. The WCO can generate the required clock with the help of an external crystal or external clock inputs.

User registers and the RTC registers have RTC fields, Alarm fields, and Config fields. The software can access the user registers. A specific writing operation updates the RTC registers with the user registers. A specific read operation copies data from the RTC registers to the user registers. See the architecture TRM and registers TRM for the details on RTC fields, Alarm fields, and Config fields.

For the WCO device and logic, the RTC function runs on VDDD, which is a continuous power supply. Therefore, the RTC function runs in all power modes, and the function continuously keeps track of the current time.

There are four operations that keep track of the current time:

1. Initializing the RTC function including interrupts
2. Reading the time and date
3. Updating the time and date
4. Calibrating the WCO

Before using the RTC function, an initialization routine must be executed. The initializing operation includes all hardware and system source initialize. The Basic RTC settings section provides the example setting for generating an interrupt when the alarm time matches the setting time.

To display the current time through UART, the CPU reads a current time from the RTC field registers. See Reading time and updating the RTC time for an example of reading the current time from the RTC field.

You can use the ILO, CLK_LF, and WCO as a source clock for the RTC function. However, the WCO is recommended as it is more accurate. See Calibrating the WCO for an example of calibrating the time of the RTC function.

2.1 Basic RTC settings

This section describes the operation of the RTC function and explains how to configure the RTC based on a use case using the Hardware Abstraction Layer (HAL) provided by Infineon. The code snippets in this application note are part of the HAL.

The HAL has a configuration part and a driver part. The configuration part mainly configures the parameter values for the desired operation. The driver part configures each register based on the parameter values in the configuration part.

2.1.1 Use case

This use case shows how to initialize items such as input clock, time, and date to enable the RTC function, and enable the ALARM function. When the ALARM setting time and RTC time match, an interrupt is generated.

2.1.2 Initializing the RTC function

The following example initializes the RTC function after the power-on reset (POR). After the RTC function is initialized, there is no need to reinitialize the RTC function even if the device has switched power modes.

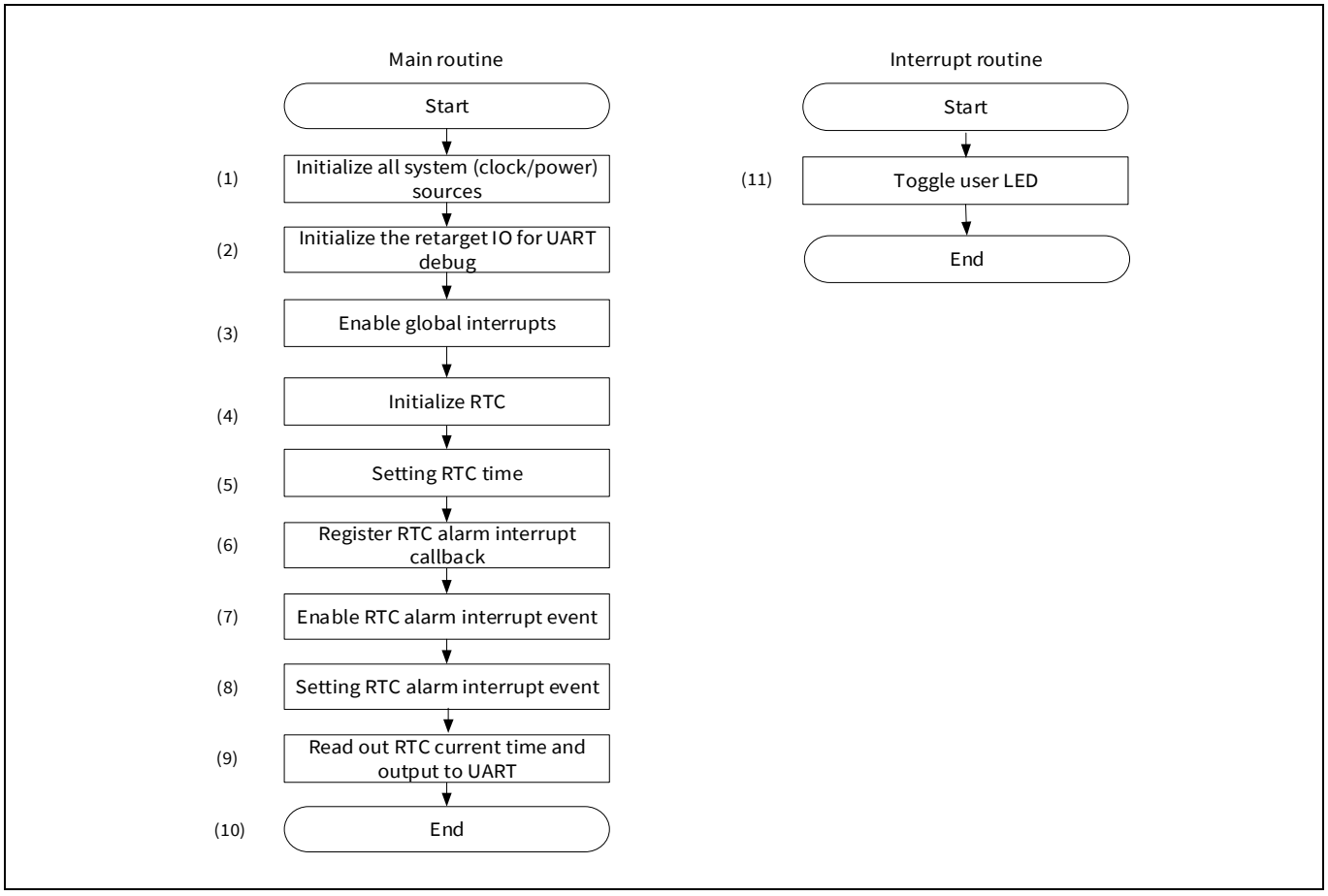

Figure 1 Configuring basic RTC settings and interrupt routine

The basic setup for the RTC function has the following steps:

1. Initialize all system resources and peripherals on the board.
2. Initialize the retarget standard I/O to the debug UART port.
3. Enable the global interrupt.
4. Initialize the RTC with the specified data struct.
5. Update the RTC current time and date.
6. Register the callback function to handle the alarm interrupt event.
7. Set the alarm interrupt for the specified time and date.
8. Get the RTC current time and date.
9. Read out the RTC current time, then output to the UART terminal.
   The alarm function generates an interrupt when the RTC fields and the alarm fields match.

If you want to set the alarm every 30 seconds, you need to set two alarms.

For details on the interrupt setting procedure, see the “Interrupt and Fault Report Structure” section in AN233226 - Interrupt usage in XMC7000 family.

2.1.3 Configuring the RTC function

Table 1 lists the parameters and Table 2 lists the functions of the configuration part in HAL for RTC and ALARM settings.

Table 1 RTC and ALARM setting parameters

Parameters	Description	Value
`time.tm_sec`	Calendar seconds, 0-59	INITIAL_DATE_SEC
`time.tm_min`	Calendar minutes, 0-59	INITIAL_DATE_MIN
`time.tm_hour`	Calendar hours, value depending on the 12/24HR mode	INITIAL_DATE_HOUR
`time.tm_mday`	Calendar days of the month, 1-31	INITIAL_DATE_MDAY
`time.tm_mon`	Calendar month, 1-12	INITIAL_DATE_MON
`time.tm_year`	Calendar year, 0-99	INITIAL_DATE_YEAR
`time.tm_wday`	Calendar day of the week, 1-7. You can define the values, but it's recommended to set 1=Monday	INITIAL_DATE_WDAY
`time.tm_yday`	Calendar day of the year, 1-365	INITIAL_DATE_YDAY
`time.tm_isdst`	Enables the DST function: 0 = disable 1 = enable	ENABLE_DST_FUNCTION
`cyhal_alarm_active_t.en_sec`	Alarm second enable: 0 = ignore 1 = match	1
`cyhal_alarm_active_t.en_min`	Alarm min enable: 0 = ignore 1 = match	1
`cyhal_alarm_active_t.en_hour`	Alarm hour enable: 0 = ignore 1 = match	1
`cyhal_alarm_active_t.en_day`	Alarm day of the week enable: 0 = ignore 1 = match	1
`cyhal_alarm_active_t.en_date`	Alarm day of the month enable: 0 = ignore 1 = match	1
`cyhal_alarm_active_t.en_month`	Alarm month enable: 0 = ignore 1 = match	1

Operation overview

Table 2 RTC and ALARM setting functions

Functions	Description	Value
`cybsp_init()`	Initializes the device and board peripherals	None
`__enable_irq()`	Enables the global interrupt	None
`cyhal_rtc_init(&rtc_obj)`	Initializes the RTC function	`&rtc_obj`
`cyhal_rtc_write(&rtc_obj, &new_date_time)`	Updates the RTC current date and time	`&rtc_obj, &new_date_time`
`cyhal_rtc_register_callback( &rtc_obj,(cyhal_rtc_event_callback_t) cyhal_rtc_alarm_interrupt_handler, RTC_CALLBACK_ARG)`	Registers an RTC event callback handler	`&rtc_obj, cyhal_rtc_alarm_interrupt_handler, RTC_CALLBACK_ARG`
`cyhal_rtc_enable_event(&rtc_obj, CYHAL_RTC_ALARM, RTC_INTERRUPT_PRIORITY, TRUE)`	Configures RTC event enablement	`&rtc_obj, CYHAL_RTC_ALARM, RTC_NTERRUPT_PRIORITY, TRUE`
`cyhal_rtc_set_alarm(&rtc_obj, &alarm_date_time, alarm_active)`	Sets an alarm (interrupt) for the specified time and date	`&rtc_obj, &alarm_date_time, alarm_active`
`cyhal_rtc_read(&rtc_obj, &new_date_time)`	Gets the RTC date and time to save into `struct new_date_time`	`&rtc_obj, &new_date_time`

Code Listing 1 shows an example program of the configuration part for the RTC function.

The following description will help you understand the driver part of the HAL:

Code Listing 1 Configuring the RTC

```
/* Set the RTC initial time and date parameter in temporary structure*/
struct tm new_date_time =
    {
        .tm_sec   = INITIAL_DATE_SEC,
        .tm_min   = INITIAL_DATE_MIN,
        .tm_hour  = INITIAL_DATE_HOUR,
        .tm_mday  = INITIAL_DATE_MDAY,
        .tm_mon   = INITIAL_DATE_MONTH,
        .tm_year  = INITIAL_DATE_YEAR,
        .tm_wday  = INITIAL_DATE_WDAY,
        .tm_yday  = INITIAL_DATE_YDAY,
```

Operation overview

Code Listing 1 Configuring the RTC

```
        .tm_isdst = ENABLE_DST_FUNCTION,
    };

/* Set the RTC alarm time and date parameter in temporary structure*/
struct tm alarm_date_time =
{
       .tm_sec = INITIAL_ALARM_SEC,
       .tm_min = INITIAL_ALARM_MIN,
       .tm_hour = INITIAL_ALARM_HOUR,
       .tm_mday = INITIAL_ALARM_MDAY,
       .tm_mon = INITIAL_ALARM_MONTH,
       .tm_year = INITIAL_ALARM_YEAR,
       .tm_wday = INITIAL_ALARM_WDAY,
       .tm_yday = INITIAL_ALARM_YDAY,
       .tm_isdst = ENABLE_DST_FUNCTION,

};
/* Set the RTC alarm enable functions parameter in temporary structure*/
 cyhal_alarm_active_t alarm_active =
    {
        .en_sec   = 1,
        .en_min   = 1,
        .en_hour  = 1,
        .en_day   =  1,
        .en_date  = 1,
        .en_month = 1,
    };
```

Code Listing 2 shows an example program of the RTC interrupts routine for alarm1.

Code Listing 2 RTC interrupt routine for alarm1

```
void cyhal_rtc_alarm_interrupt_handler(void* arg, cyhal_rtc_event_t event)
{
    (void)arg;
    if (event == CYHAL_RTC_ALARM)
{
    /* Toggle user LED when RTC alarm interrupt generated*/
        cyhal_gpio_toggle(CYBSP_USER_LED);
    }
}
```

2.1.4 Configuring the RTC function in the driver part

Code Listing 3 shows an example program to configure the RTC and alarm1 function.

Code Listing 3 Configuring the RTC input clock source set in the driver part

```
int main(void)
{
    cy_rslt_t rslt;

    /* Initialize the device and board peripherals */
    rslt = cybsp_init();
    if (CY_RSLT_SUCCESS != rslt)
    {
        handle_error();
    }

    /* Initialize retargeting standard IO to the debug UART port */
    cy_retarget_io_init(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX,
                        CY_RETARGET_IO_BAUDRATE);

    /* Enable global interrupts */
    __enable_irq();

    /* \x1b[2J\x1b[;H - ANSI ESC sequence for clear screen */
    printf("\x1b[2J\x1b[;H");
    printf("****************** XMC7200 MCU RTC Basics ******************\r\n\n");

    /*Initialize the USER LED pin*/
    rslt = cyhal_gpio_init(CYBSP_USER_LED, CYHAL_GPIO_DIR_OUTPUT,
CYHAL_GPIO_DRIVE_STRONG, TRUE);

    /* Initialize RTC */
    rslt = cyhal_rtc_init(&rtc_obj);
    if (CY_RSLT_SUCCESS != rslt)
    {
        handle_error();
    }

    /*write RTC new time*/
    cyhal_rtc_write(&rtc_obj, &new_date_time);
    if (CY_RSLT_SUCCESS != rslt)
              {
```

Operation overview

Code Listing 3 Configuring the RTC input clock source set in the driver part

```
                handle_error();
            }


    /*register RTC alarm interrupt callback*/

cyhal_rtc_register_callback(&rtc_obj,(cyhal_rtc_event_callback_t)cyhal_rtc_alarm_in
terrupt_handler, RTC_CALLBACK_ARG);


    /*enable RTC alarm interrupt event*/
    cyhal_rtc_enable_event(&rtc_obj, CYHAL_RTC_ALARM, RTC_INTERRUPT_PRIORITY,
TRUE);


    /*set the RTC alarm event*/
    rslt = cyhal_rtc_set_alarm(&rtc_obj, &alarm_date_time, alarm_active);
    if (CY_RSLT_SUCCESS != rslt)
       {
          handle_error();
       }



    for (;;)
    {
       rslt = cyhal_rtc_read(&rtc_obj, &new_date_time);
       if(rslt == CY_RSLT_SUCCESS)
       {
        printf("%d:%d:%d %d/%d/%d \r\n",new_date_time.tm_hour,
new_date_time.tm_min, new_date_time.tm_sec,new_date_time.tm_year,
new_date_time.tm_mon, new_date_time.tm_mday );
       }
    }
}
```

2.2 Reading time and updating the RTC time

To output the RTC date and time to the UART, the CPU reads the current time from the RTC field registers.

The RTC function of the XMC7000 device has a READ bit in the BACKUP_RTC_RW register that will be used to copy the current RTC field values in the RTC registers to the RTC field in user registers in real-time. The RTC field values in user registers are frozen and are not updated even if the RTC field values in the RTC registers are updated. Then, the user firmware can copy the frozen RTC values to a user buffer, which are local variables assigned in the RAM.

Figure 2 describes the use case of reading and updating the RTC time. It also can set the RTC DST function.

This section explains how to read and update the RTC value using the HAL. The code snippets in this application note are part of the HAL.

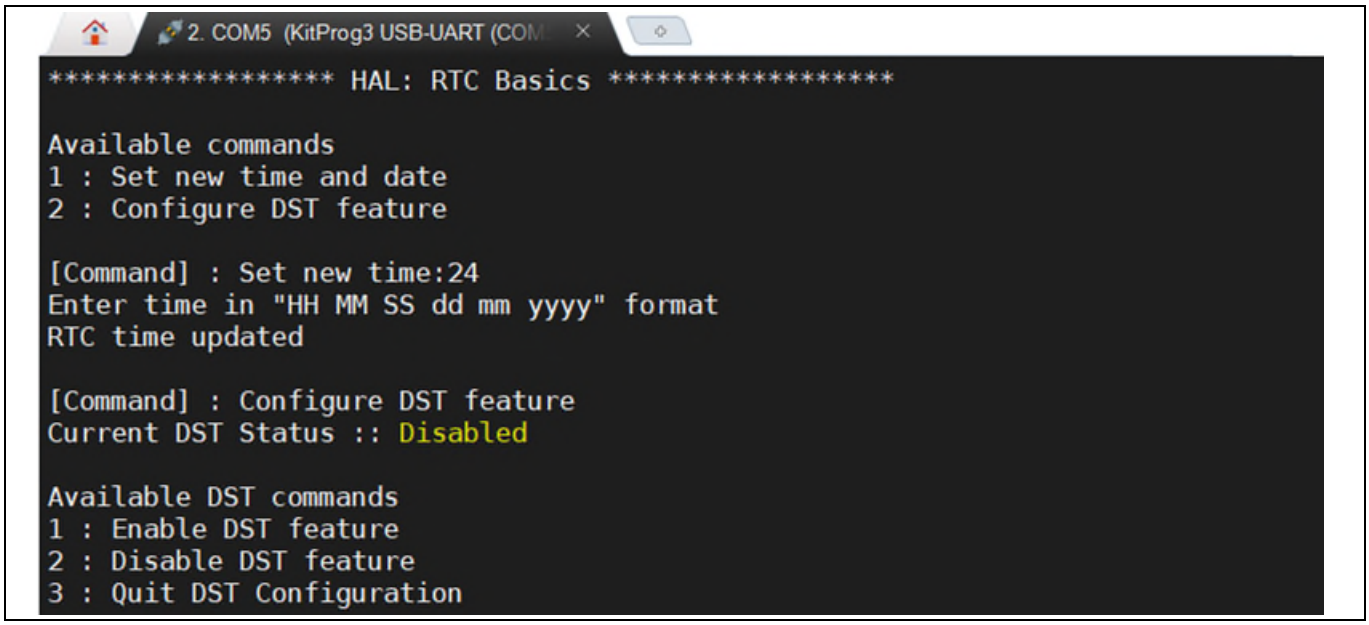

Figure 2 Readout the RTC date and time

2.2.1 Use case

This section explains an example to read and update the RTC value in the following use case. In this example, the following items are read regularly. Therefore, these have no fixed values.

Figure 3 shows an example flow to read and update the RTC value.

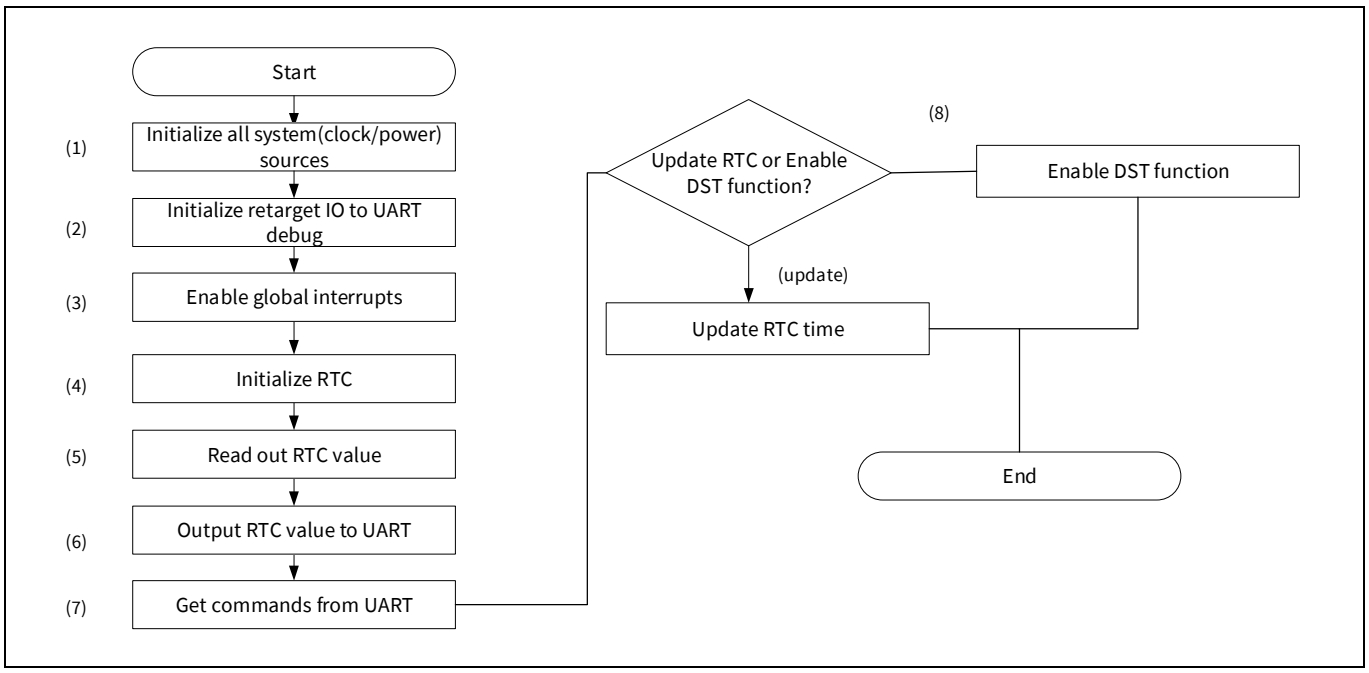

Figure 3 Reading and updating the RTC value

2.2.2 Reading the RTC values

The following example describes reading the RTC values:

1. Initialize the device system and peripherals sources.
2. Enable the global interrupt.
3. Initialize the RTC peripheral and enable it.
4. Read out the RTC current time, then output to the UART terminal.
5. Get the commands from the UART to update the RTC time or enable the DST function.

2.2.3 Items read for the RTC value

Table 3 lists the parameters that store the RTC data readings. These items are the same as the initialization items listed in Table 1 and Table 2.

Table 3 Read RTC values

Parameters	Description
`date_time.tm_sec`	Stored second value (Calendar seconds, 0-59)
`date_time.tm_min`	Stored minute value (Calendar minutes, 0-59)
`date_time.tm_hour`	Stored hour value (Calendar hours, value depending on 12/24HR mode)
`date_time.tm_mday`	Stored day of the month value (Calendar day of the month, 1-31) Automatic leap year correction
`date_time.tm_mon`	Stored month value (Calendar month, 1-12)
`date_time.tm_year`	Stored year value (Calendar year, 0-99)
`date_time.tm_wday`	Stored day value (Calendar day of the week, 1-7)

Operation overview

Parameters	Description
	You can define the values, but it is recommended to set 1=Monday.
`date_time.tm_yday`	Stored day value (Calendar day of the year, 1-365)
`date_time.tm_isdst`	Stored DST value (0-disable, 1-enable)

Code Listing 4 shows an example program of reading the RTC value and the main routine.

Code Listing 4 Reading the RTC value and main routine

```
int main(void)
{
    cy_rslt_t rslt;
    uint8_t cmd;
    char buffer[STRING_BUFFER_SIZE];
    struct tm date_time;

    /* Initialize the device and board peripherals */
    rslt = cybsp_init();
    if (CY_RSLT_SUCCESS != rslt)
    {
        handle_error();
    }

    /* Initialize retargeting standard IO to the debug UART port */
    cy_retarget_io_init(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX,
                        CY_RETARGET_IO_BAUDRATE);

    /* Enable global interrupts */
    __enable_irq();

    /* \x1b[2J\x1b[;H - ANSI ESC sequence for clear screen */
    printf("\x1b[2J\x1b[;H");
    printf("****************** HAL: RTC Basics ******************\r\n\n");

    /* Initialize RTC */
    rslt = cyhal_rtc_init(&rtc_obj);
    if (CY_RSLT_SUCCESS != rslt)
    {
        handle_error();
    }

    /* Display available commands */
```

Code Listing 4 Reading the RTC value and main routine

```
    printf("Available commands \r\n");
    printf("1 : Set new time and date\r\n");
    printf("2 : Configure DST feature\r\n\n");

    for (;;)
{
    /*Read out the RTC time to uart*/
        rslt = cyhal_rtc_read(&rtc_obj, &date_time);
        if (CY_RSLT_SUCCESS == rslt)
        {
            strftime(buffer, sizeof(buffer), "%c", &date_time);
            printf("\r%s", buffer);
            memset(buffer, '\0', sizeof(buffer));
        }

        /*Get commands from uart*/
        rslt = cyhal_uart_getc(&cy_retarget_io_uart_obj, &cmd, UART_TIMEOUT_MS);
        if (rslt != CY_RSLT_ERR_CSP_UART_GETC_TIMEOUT)
        {
            /*Update RTC time command*/
            if (RTC_CMD_SET_DATE_TIME == cmd)
            {
                printf("\r[Command] : Set new time\r\n");
                set_new_time(INPUT_TIMEOUT_MS);        /*update RTC time value*/
            }
             /*Get command to enable DST function*/
            else if (RTC_CMD_CONFIG_DST == cmd)
            {
                printf("\r[Command] : Configure DST feature\r\n");
                set_dst_feature(INPUT_TIMEOUT_MS);    /*operate the RTC DST
function*/
            }
        }
    }
}
```

Code Listing 5 shows an example program to update the RTC value.

Code Listing 5 Updating the RTC value

```
static void set_new_time(uint32_t timeout_ms)
{
```

Operation overview

Code Listing 5 Updating the RTC value

```
    cy_rslt_t rslt;
    char buffer[STRING_BUFFER_SIZE] = {0};
    uint32_t space_count = 0;

    /* Variables used to store date and time information */
    int mday, month, year, sec, min, hour;
    struct tm new_time = {0};

    printf("\rEnter time in \"HH MM SS dd mm yyyy\" format \r\n");
    rslt = fetch_time_data(buffer, timeout_ms, &space_count);
    if (rslt != CY_RSLT_ERR_CSP_UART_GETC_TIMEOUT)
    {
        if (space_count != MIN_SPACE_KEY_COUNT)
        {
            printf("\rInvalid values! Please enter the values in specified
format\r\n");
        }
        else
        {
            sscanf(buffer, "%d %d %d %d %d %d",
                   &hour, &min, &sec,
                   &mday, &month, &year);

            if (validate_date_time(sec, min, hour, mday, month, year))
            {
                new_time.tm_sec = sec;
                new_time.tm_min = min;
                new_time.tm_hour = hour;
                new_time.tm_mday = mday;
                new_time.tm_mon = month - 1;
                new_time.tm_year = year - TM_YEAR_BASE;
                new_time.tm_wday = get_day_of_week(mday, month, year);

                rslt = cyhal_rtc_write(&rtc_obj, &new_time);   /*updating RTC
value*/
                if (CY_RSLT_SUCCESS == rslt)
                {
                    printf("\rRTC time updated\r\n\n");
                }
                else
                {
```

Operation overview

Code Listing 5 Updating the RTC value

```
                    handle_error();
                }
            }
            else
            {
                printf("\rInvalid values! Please enter the values in specified"
                       " format\r\n");
            }
        }
    }
    else
    {
        printf("\rTimeout \r\n");
    }
}
```

Code Listing 5 shows an example program to operate the RTC DST function.

Code Listing 6 Operating the RTC DST function

```
static void set_dst_feature(uint32_t timeout_ms)
{
    cy_rslt_t rslt;
    uint8_t dst_cmd;
    char dst_start_buffer[STRING_BUFFER_SIZE] = {0};
    char dst_end_buffer[STRING_BUFFER_SIZE] = {0};
    uint32_t space_count = 0;

    /* Variables used to store DST start and end time information */
    cyhal_rtc_dst_t dst_start_time, dst_end_time;

    /* Variables used to store date and time information */
    int mday = 0, month = 0, year = 0, sec = 0, min = 0, hour = 0;
    uint8_t fmt = 0;
    if (DST_ENABLED_FLAG == dst_data_flag)
    {
        if (cyhal_rtc_is_dst(&rtc_obj))
        {
            printf("\rCurrent DST Status :: Active\r\n\n");
        }
        else
```

Code Listing 6 Operating the RTC DST function

```
        {
            printf("\rCurrent DST Status :: Inactive\r\n\n");
        }
    }
    else
    {
        printf("\rCurrent DST Status :: Disabled\r\n\n");
    }

    /* Display available commands */
    printf("Available DST commands \r\n");
    printf("1 : Enable DST feature\r\n");
    printf("2 : Disable DST feature\r\n");
    printf("3 : Quit DST Configuration\r\n\n");

    rslt = cyhal_uart_getc(&cy_retarget_io_uart_obj, &dst_cmd, timeout_ms);
    if (rslt != CY_RSLT_ERR_CSP_UART_GETC_TIMEOUT)
    {
        if (RTC_CMD_ENABLE_DST == dst_cmd)
        {
            /* Get DST start time information */
            printf("Enter DST format \r\n");
            printf("1 : Fixed DST format\r\n");
            printf("2 : Relative DST format\r\n\n");

            rslt = cyhal_uart_getc(&cy_retarget_io_uart_obj, &fmt, timeout_ms);
            if (rslt != CY_RSLT_ERR_CSP_UART_GETC_TIMEOUT)
            {
                printf("Enter DST start time in \"HH MM SS dd mm yyyy\"
format\r\n");
                rslt = fetch_time_data(dst_start_buffer, timeout_ms, &space_count);
                if (rslt != CY_RSLT_ERR_CSP_UART_GETC_TIMEOUT)
                {
                    if (space_count != MIN_SPACE_KEY_COUNT)
                    {
                        printf("\rInvalid values! Please enter the values in
specified format\r\n");
                    }
                    else
                    {
                        sscanf(dst_start_buffer, "%d %d %d %d %d %d",
```

Code Listing 6 Operating the RTC DST function

```
                                &hour, &min, &sec,
                                &mday, &month, &year);


                        if ((validate_date_time(sec, min, hour, mday, month, year))
&& ((fmt == FIXED_DST_FORMAT) || (fmt == RELATIVE_DST_FORMAT)))
                            {
                                dst_start_time.format = (fmt == FIXED_DST_FORMAT) ?
CYHAL_RTC_DST_FIXED : CYHAL_RTC_DST_RELATIVE;
                                dst_start_time.hour = hour;
                                dst_start_time.month = month;
                                dst_start_time.dayOfWeek = (fmt == FIXED_DST_FORMAT) ?
1 : get_day_of_week(mday, month, year);
                                dst_start_time.dayOfMonth = (fmt == FIXED_DST_FORMAT) ?
mday : 1;
                                dst_start_time.weekOfMonth = (fmt == FIXED_DST_FORMAT)
? 1 : get_week_of_month(mday, month, year);
            /* Update flag value to indicate that a valid DST start time information
has been received*/
                                dst_data_flag = DST_VALID_START_TIME_FLAG;
                            }
                            else
                            {
                                printf("\rInvalid values! Please enter the values in
specified"
                                       " format\r\n");
                            }
                        }
                    }
                    else
                    {
                        printf("\rTimeout \r\n");
                    }

                    if (DST_VALID_START_TIME_FLAG == dst_data_flag)
                    {
                    /* Get DST end time information, iff a valid DST start time
information is received */
                        printf("Enter DST end time in \"HH MM SS dd mm yyyy\"
format\r\n");
                        rslt = fetch_time_data(dst_end_buffer, timeout_ms,
&space_count);
                        if (rslt != CY_RSLT_ERR_CSP_UART_GETC_TIMEOUT)
                        {
```

Code Listing 6 Operating the RTC DST function

```
                        if (space_count != MIN_SPACE_KEY_COUNT)
                        {
                         printf("\rInvalid values! Please enter the values in
specified format\r\n");
                        }
                        else
                        {
                            sscanf(dst_end_buffer, "%d %d %d %d %d %d",
                                   &hour, &min, &sec,
                                   &mday, &month, &year);

                            if ((validate_date_time(sec, min, hour, mday, month,
year)) && ((fmt == FIXED_DST_FORMAT) || (fmt == RELATIVE_DST_FORMAT)))
                            {
                                dst_end_time.format = (fmt == FIXED_DST_FORMAT) ?
CYHAL_RTC_DST_FIXED : CYHAL_RTC_DST_RELATIVE;
                                dst_end_time.hour = hour;
                                dst_end_time.month = month;
                                dst_end_time.dayOfWeek = (fmt == FIXED_DST_FORMAT)
? 1 : get_day_of_week(mday, month, year);
                                dst_end_time.dayOfMonth = (fmt == FIXED_DST_FORMAT)
? mday : 1;
                                dst_end_time.weekOfMonth = (fmt ==
FIXED_DST_FORMAT) ? 1 : get_week_of_month(mday, month, year);
            /* Update flag value to indicate that a valid DST end time information
has been received*/
                                dst_data_flag = DST_VALID_END_TIME_FLAG;
                            }
                            else
                            {
                                printf("\rInvalid values! Please enter the values
in specified"
                                       " format\r\n");
                            }
                        }
                    }
                    else
                    {
                        printf("\rTimeout \r\n");
                    }
                }

                if (DST_VALID_END_TIME_FLAG == dst_data_flag)
```

Operation overview

Code Listing 6 Operating the RTC DST function

```
                {
                    rslt = cyhal_rtc_set_dst(&rtc_obj, &dst_start_time,
&dst_end_time);
                    if (CY_RSLT_SUCCESS == rslt)
                    {
                        dst_data_flag = DST_ENABLED_FLAG;
                        printf("\rDST time updated\r\n\n");
                    }
                    else
                    {
                        handle_error();
                    }
                }
            }
            else
            {
                printf("\rTimeout \r\n");
            }
        }
        else if (RTC_CMD_DISABLE_DST == dst_cmd)
        {
            dst_end_time.format = CYHAL_RTC_DST_FIXED;
            dst_end_time.hour = 0;
            dst_end_time.month = 1;
            dst_end_time.dayOfWeek = 1;
            dst_end_time.dayOfMonth = 1;
            dst_end_time.weekOfMonth = 1;
            dst_start_time = dst_end_time;
            rslt = cyhal_rtc_set_dst(&rtc_obj, &dst_start_time, &dst_end_time);
            if (CY_RSLT_SUCCESS == rslt)
            {
                dst_data_flag = DST_DISABLED_FLAG;
                printf("\rDST feature disabled\r\n\n");
            }
            else
            {
                handle_error();
            }
        }
        else if (RTC_CMD_QUIT_CONFIG_DST == dst_cmd)
        {
```

Code Listing 6 Operating the RTC DST function

```
            printf("\rExit from DST Configuration \r\n\n");
        }
    }
    else
    {
        printf("\rTimeout \r\n");
    }
}
```

2.3 Calibrating the WCO

Note: Calibration of the WCO is possible only on the device having the RTC_CAL pin.

Even if the WCO uses a crystal oscillator, it is not immune to frequency errors. The RTC values may not correspond to the standard time in your region because of such errors. In such cases, you can calibrate the WCO and improve the accuracy of the real-time clock.

This section explains how to calibrate the WCO using the Peripherals Driver Library (PDL) provided by Infineon. The code snippets in this application note are part of PDL.

2.3.1 Use case

This use case shows how to output the calibration waveform, set the calibration value, and confirm the calibration.

Use case:

- Outputs the 512-Hz calibration waveform to the RTC_CAL pin.
- Sets the calibration value in the register based on the 27.125-ppm slow down calibration waveform.
- Confirms the calibration waveform in the 1-Hz output.

This use case has the following three steps:

1. Routing the WCO waveform to the RTC_CAL pin and capturing the waveform.
2. Measuring the difference (frequency) between the ideal waveform and the actual WCO output, and setting a calibration value.
3. Confirming the accuracy of the WCO frequency by measuring the WCO frequency again.

2.3.2 Capturing the WCO waveform

The WCO waveform can be captured in an oscilloscope by routing it to the RTC_CAL pin of the device. This can be achieved by setting the CAL_OUT bit. A 512-Hz clock waveform, derived from the WCO, will be output on the RTC_CAL pin. You must measure the deviation of this output from an ideal 512-Hz clock, and convert the deviation to a ppm value. Then, you can calculate the calibration settings to be used to correct the WCO frequency error. Before executing this operation, the RTC function must be run normally with the configuration specified in Table 3.

Operation overview

Figure 4 shows an example flow to calibrate the WCO waveform output.

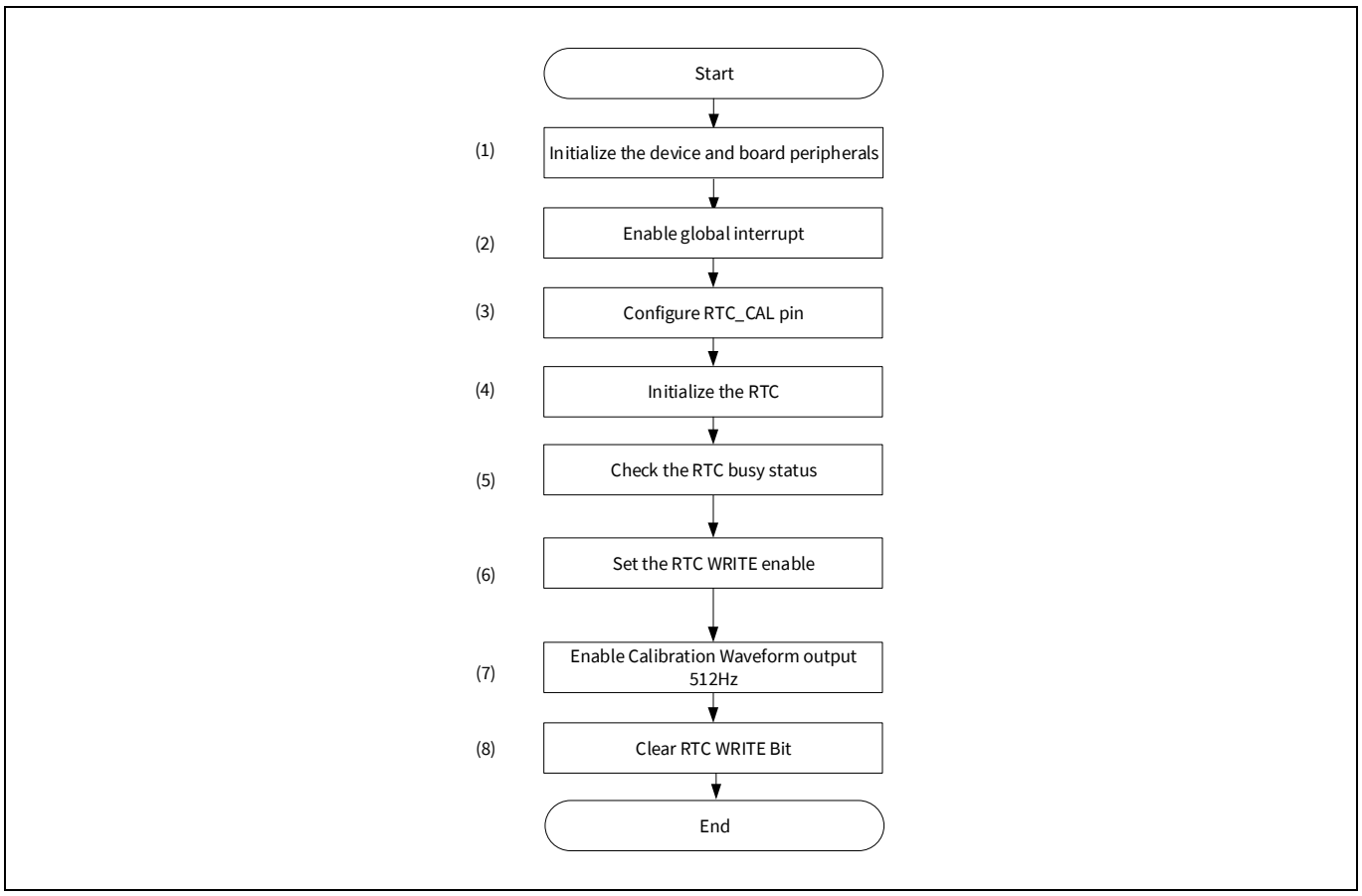

Figure 4 Calibrating the WCO waveform output

Do the following to output the WCO waveform:

1. Initialize the device and board peripherals.
2. Enable Global Interrupt (CPU Interrupt Enable). For details, see the CPU interrupt handling sections in the architecture TRM.
3. Configure the RTC_CAL pin; see the “I/O System” chapter in the architecture TRM.
4. Initialize the RTC driver.
5. Check the RTC busy status.
6. Set the WRITE bit to update the output of the calibration waveform:
   BACKUP_RTC_RW |= BACKUP_RTC_RW_WRITE_Msk (Begin the Write operation)
7. Set the waveform frequency for 512 Hz:
   Write BACKUP_CAL_CTL.CAL_SEL = ‘0’
   Enable the calibration waveform:
   Write BACKUP_CAL_CTL.CAL_OUT = ‘1’ (Calibration waveform output).
   Then, the WCO waveform is output on the RTC_CAL pin.
8. Clear the WRITE bit:
   BACKUP_RTC_RW &= ((uint32_t) ~BACKUP_RTC_RW_WRITE_Msk) (End the Write operation).

Table 4 lists the parameters of the configuration part in PDL for RTC_CAL pin output in the GPIO.

For the GPIO setting, see the “I/O System” chapter in the architecture TRM.

Operation overview

Table 4 RTC_CAL pin output in GPIO

Parameters	Description	Value
.outVal	Selects the pin output state: 0 = Output state not affected 1 = Output state set to '0'	0ul
.driveMode	Selects GPIO drive mode for I/O pin: 0 = Analog High-Z. Input buffer OFF. 1 = Invalid mode. Do not use. 2 = Resistive Pull-Up. Input buffer OFF. 3 = Resistive Pull-Down. Input buffer OFF. 4 = Open Drain, Drives Low. Input buffer OFF. 5 = Open Drain, Drives High. Input buffer OFF. 6 = Strong Drive. Input buffer OFF. 7 = Resistive Pull-Up/Down. Input buffer OFF. 8 = Digital High-Z. Input buffer ON. 9 = Invalid mode. Do not use. 10 = Resistive Pull-Up. Input buffer ON. 11 = Resistive Pull-Down. Input buffer ON. 12 = Open Drain, Drives Low. Input buffer ON. 13 = Open Drain, Drives High. Input buffer ON. 14 = Strong Drive. Input buffer ON. 15 = Resistive Pull-Up/Down. Input buffer ON.	CY_GPIO_DM_STRONG_IN_OFF =6ul
.hsiom	Sets the connection for the RTC_CAL pin route.	P21_7_SRSS_CAL_WAVE
.intEdge	Sets the edge which will trigger an IRQ for the I/O pin: 0 = Disabled 1 = Rising edge 2 = Falling edge 3 = Both	0ul
.intMask	Masks edge interrupt on IO pin: 0 = Pin interrupt forwarding disabled 1 = Pin interrupt forwarding enabled	0ul
.vtrip	Selects the IO pin input buffer mode: 0 = CMOS 1 = TTL	0ul
.slewRate	Selects the slew rate for the I/O pin: 0 = Fast slew rate 1 = Slow slew rate	0ul
.driveSel	Sets the GPIO drive strength for the I/O pin: 0 = Full drive strength 1 = Full drive strength 2 = 1/2 drive strength 3 = 1/4 drive strength	0ul

Operation overview

Parameters	Description	Value
.vregEn	SIO pair output buffer mode	0ul
.ibufMode	SIO pair input buffer mode	0ul
.vtripSel	SIO pair input buffer trip point	0ul
.vrefSel	SIO pair reference voltage for the input buffer trip point	0ul
.vohSel	SIO pair regulated voltage output level	0ul

Table 5 lists the parameters and Table 6 lists the function of the configuration part in the PDL to output the RTC calibration waveform.

Table 5 RTC calibration waveform output parameters

Parameters	Description	Value
CALIB_VALUE	Calibration value for absolute frequency. Each step causes 128 ticks to be added or removed each hour.	0ul
CY_EN_RTC_CALIB_SIGN_NEGATIVE	0 = Negative sign: Removes pulses (it takes more clock ticks to count one second) 1 = Positive sign: Adds pulses (it takes less clock ticks to count one second)	0
CY_EN_RTC_CAL_SEL_CAL512	Selects the calibration wave output signal 0 = 512-Hz wave, not affected by calibration setting. 1 = reserved 2 = 2-Hz wave, includes the effect of the calibration setting. (not supported for 50/60Hz input clock: CTL.PRESCALER!=0) 3 = 1-Hz wave, includes the effect of the calibration setting. supported for all input clocks	0

Table 6 RTC calibration waveform output functions

Function	Description	Value
Cy_RTC_CalibrationControlEnable (uint8_t calib_val, cy_en_rtc_calib_sign_t calib_sign, cy_en_rtc_cal_sel_t cal_sel)	Sets the calibration control register and enables the calibration waveform output: calib_val = Calibration value for absolute frequency calib_sign = Calibration sign cal_sel = Selects calibration waveform output signal	CALIB_VALUE = 0ul, CY_EN_RTC_CALIB_SIGN_NEGATIVE=0, CY_EN_RTC_CAL_SEL_CAL512 = 0

Code Listing 7 shows an example program to output the RTC calibration waveform.

The following description will help you understand the register notation of the driver part of PDL:

Operation overview

- **BACKUP > CAL_CTL** register is the BACKUP_CAL_CTL register mentioned in the registers TRM. Other registers are also described in the same manner.

The RTC_CAL pin configuration shows the case of the XMC7200 series. These RTC_CAL pin settings are the same for example Code Listing 7 and Code Listing 12.

Code Listing 7 Outputting the RTC calibration waveform

```
/* Configure the RTC_CAL pin definition (P21_7)pin */
#define RTC_CAL_PIN                     P21_7_SRSS_CAL_WAVE
#define RTC_CAL_PIN_PORT                GPIO_PRT21
#define RTC_CAL_PIN_NUM                 P21_7_NUM

/* Setting value for calibration definition */
#define CALIB_VALUE                     0u

/*Define the RTC initial parameters*/
#define CY_RTC_DATE                     22u
#define CY_RTC_YEAR                     22u
#define CY_RTC_INTINAL_SEC              0u
#define CY_RTC_INTIAL_MIN               0u
#define CY_RTC_INITIAL_HOUR             15u

/*******************************************************************************
* Global Variables
*******************************************************************************/
/*configure the RTC initialize time*/
cy_stc_rtc_config_t const RTC_Config =
{
        .sec = CY_RTC_INTINAL_SEC,
        .min = CY_RTC_INTIAL_MIN,
        .hour = CY_RTC_INITIAL_HOUR,
        .dayOfWeek = CY_RTC_FRIDAY,
        .hrFormat = CY_RTC_24_HOURS,
        .date = CY_RTC_DATE,
        .month = CY_RTC_APRIL,
        .year = CY_RTC_YEAR
};

/*configure RTC_CAL Pin*/
cy_stc_gpio_pin_config_t rtc_cal_pin_cfg =
{
        .outVal = 0,
        .driveMode = CY_GPIO_DM_STRONG_IN_OFF,
```

Code Listing 7 Outputting the RTC calibration waveform

```
        .hsiom = RTC_CAL_PIN,
        .intEdge = CY_GPIO_INTR_DISABLE,
        .intMask = 0UL,
        .vtrip = CY_GPIO_VTRIP_CMOS,
        .slewRate = CY_GPIO_SLEW_FAST,
        .driveSel = CY_GPIO_DRIVE_FULL,
        .vregEn = 0UL,
        .ibufMode = 0UL,
        .vtripSel = 0UL,
        .vrefSel = 0UL,
        .vohSel = 0UL,
};

/*******************************************************************************
* Function Name: main
********************************************************************************
* Summary:
* Return: int
*
*******************************************************************************/
int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init();

    /* Board initialize failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Enable global interrupts */
    __enable_irq();

    /*RTC_CAL pin initialize*/
    Cy_GPIO_Pin_Init(RTC_CAL_PIN_PORT, RTC_CAL_PIN_NUM, &rtc_cal_pin_cfg);

    /*RTC initialize and wait for finish it*/
```

Code Listing 7 Outputting the RTC calibration waveform

```
    while(Cy_RTC_Init(&RTC_Config) != CY_RTC_SUCCESS );

    /*Check the RTC busy status*/
    while(Cy_RTC_GetSyncStatus() == CY_RTC_BUSY);

    /*Calibration waveform output enable */
    Cy_RTC_CalibrationControlEnable(CALIB_VALUE, CY_RTC_CALIB_SIGN_NEGATIVE,
CY_RTC_CAL_SEL_CAL512);

    for (;;)
    {

    }
}
```

2.3.3 Capturing the WCO waveform in the driver part

Code Listing 8 to Code Listing 10 shows an example program to capture the WCO waveform in the driver part.

Code Listing 8 Initializing the RTC in the driver part

```
cy_en_rtc_status_t Cy_RTC_Init(cy_stc_rtc_config_t const *config)
{
    /* Set the RTC configuration structure*/
    return(Cy_RTC_SetDateAndTime(config));
}
```

Code Listing 9 Getting the sync status in the driver part

```
__STATIC_INLINE uint32_t Cy_RTC_GetSyncStatus(void)
{
    /* Check RTC busy status*/
    return((_FLD2BOOL(BACKUP_STATUS_RTC_BUSY, BACKUP_STATUS)) ? CY_RTC_BUSY :
CY_RTC_AVAILABLE);
}
```

Code Listing 10 Write enable in the driver part

```
__STATIC_INLINE cy_en_rtc_status_t Cy_RTC_WriteEnable(cy_en_rtc_write_status_t
writeEnable)
{
    cy_en_rtc_status_t retVal = CY_RTC_INVALID_STATE;
    uint32_t rtcAccessRetry = CY_RTC_ACCESS_BUSY_RETRY_COUNT;

    CY_ASSERT_L3(CY_RTC_IS_WRITE_VALID(writeEnable));

    if (writeEnable == CY_RTC_WRITE_ENABLED)
    {
        /* RTC Write bit set is possible only in condition that CY_RTC_BUSY bit = 0
        * or RTC Read bit is not set
        */
        while((Cy_RTC_GetSyncStatus() == CY_RTC_BUSY) && (rtcAccessRetry != 0U))
        {
            rtcAccessRetry--;
            Cy_SysLib_Delay(CY_RTC_BUSY_RETRY_DELAY_MS);
        }

        if((rtcAccessRetry != 0U) && (!_FLD2BOOL(BACKUP_RTC_RW_READ,
BACKUP_RTC_RW)))
        {
            /* Set the WRITE bit for write the CAL_OUT field*/
            BACKUP_RTC_RW |= BACKUP_RTC_RW_WRITE_Msk;
            retVal = CY_RTC_SUCCESS;
        }
    }
    else
    {
        /* Clearing Write Bit to complete write procedure */
        BACKUP_RTC_RW &= ((uint32_t) ~BACKUP_RTC_RW_WRITE_Msk);

        /* Delay to guarantee data write after clearing write bit */
        Cy_SysLib_DelayUs(CY_RTC_DELAY_WRITE_US);
        retVal = CY_RTC_SUCCESS;
    }

    return(retVal);
}
```

Code Listing 11 Setting the calibration control enable in the driver part

```
cy_en_rtc_status_t Cy_RTC_CalibrationControlEnable(uint8_t calib_val,
cy_en_rtc_calib_sign_t calib_sign, cy_en_rtc_calib_sel_t calib_sel)
{
    cy_en_rtc_status_t retVal = CY_RTC_INVALID_STATE;
    uint32_t interruptState;

    CY_UNUSED_PARAMETER(calib_sel);

    /* The RTC AHB registers can be updated only under condition that the
    *  Write bit is set and the RTC busy bit is cleared (RTC_BUSY = 0).
    */
    interruptState = Cy_SysLib_EnterCriticalSection();
    retVal = Cy_RTC_WriteEnable(CY_RTC_WRITE_ENABLED);
    if (CY_RTC_SUCCESS == retVal)
    {
        BACKUP_CAL_CTL =
        _VAL2FLD(BACKUP_CAL_CTL_CALIB_VAL, calib_val)  |
        _VAL2FLD(BACKUP_CAL_CTL_CALIB_SIGN, (uint32_t)calib_sign) |
#if defined (CY_IP_MXS40SSRSS) || (CY_IP_MXS40SRSS_RTC_VERSION >= 2)
        _VAL2FLD(BACKUP_CAL_CTL_CAL_SEL, (uint32_t)calib_sel) |
#endif
        /* Enable the calibration waveform output*/
        _VAL2FLD(BACKUP_CAL_CTL_CAL_OUT, 1U);

        /* Clear the RTC Write bit to finish RTC update */
        retVal = Cy_RTC_WriteEnable(CY_RTC_WRITE_DISABLED);
    }
    Cy_SysLib_ExitCriticalSection(interruptState);

    return(retVal);
}
```

2.3.4 Measuring the difference between the WCO and an ideal waveform

After outputting the WCO waveform (512 Hz) onto the external pin of RTC_CAL, measure the difference between the calibration waveform and the ideal waveform such as an atomic clock or calibrated lab equipment. Measure the difference using external calibrated lab equipment, and not using the same XMC7200 device. This external equipment calculates the ppm value based on the difference between the two clock signals. An example of measuring the calibration waveform is shown in Figure 5.

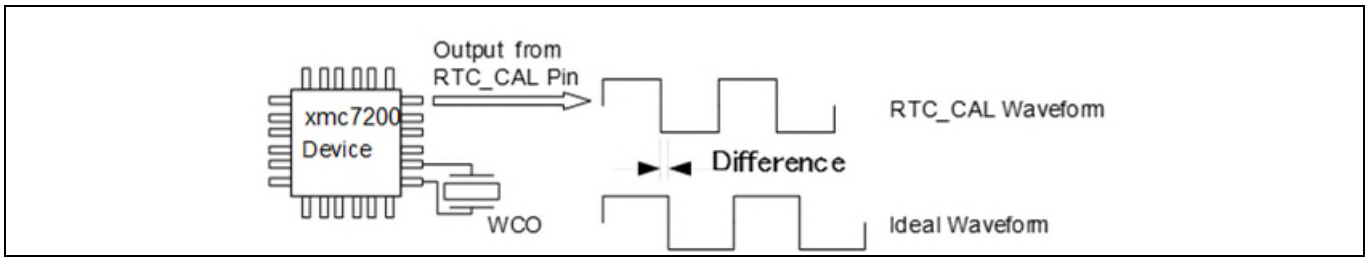

Figure 5 Measuring the difference between two waveforms

A positive calibration value of '1' will add 2 × 64 = 128 clock ticks hourly, thus reducing the number of WCO clock ticks needed to count that hour. Therefore, a calibration value of '1' represents a correction of (2 × 64) / (32,768 × 60 × 60) = 1.085 ppm. The 6-bit calibration value field can hold values up to 63, but because the calibration counter is reloaded every hour, only values up to 60 can effectively be used. This gives a calibration range of ± 60 × 1.085 ppm = ± 65.1 ppm.

The CALIB_SIGN bit determines the direction of correction. A value of '0' represents negative calibration, which will remove pulses and thus slow down real-time tracking by the RTC function. A value of '1' represents positive calibration, which will add pulses and speed up real-time tracking by the RTC function.

Figure 6 shows an example flow to calibrate the WCO.

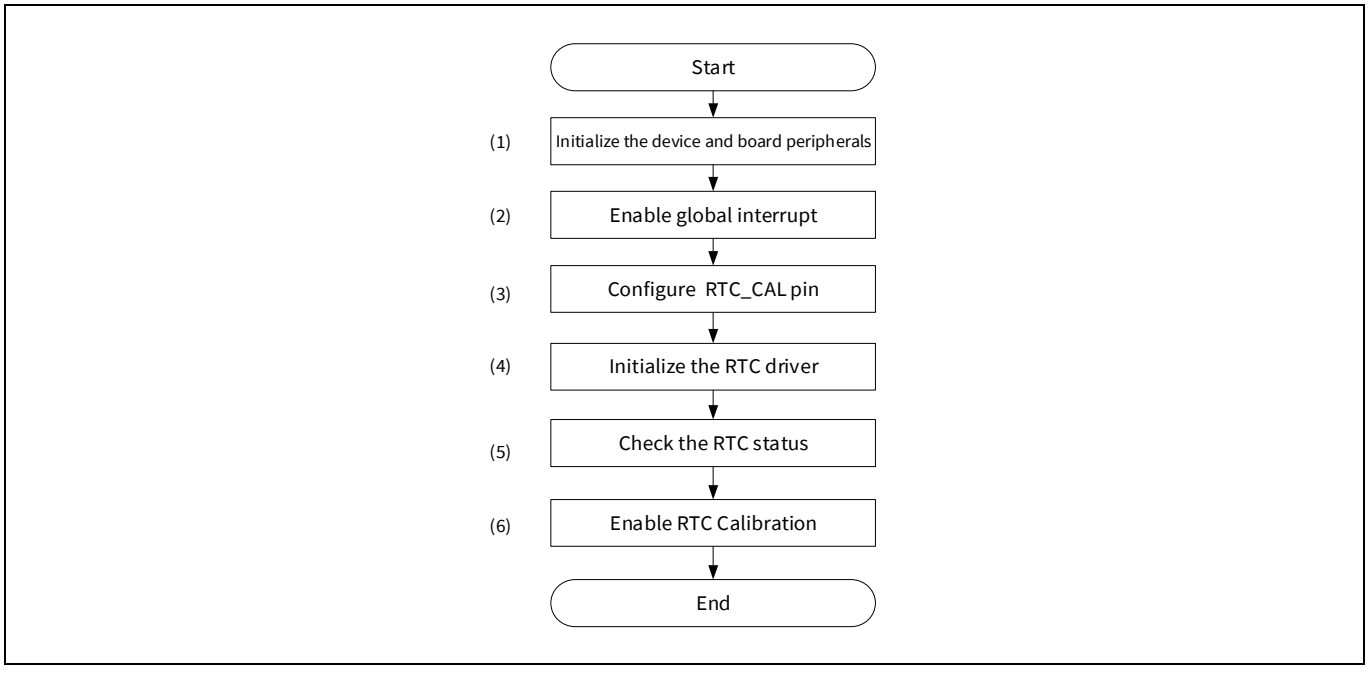

Figure 6 Calibrating the WCO

Operation overview

Before performing the following steps, it is necessary to set the WCO input clock and RTC_CAL pin output of RTC as mentioned in Capturing the WCO waveform.

1. Initialize the device and board peripherals.
2. Enable Global Interrupt (CPU Interrupt Enable). For details, see the CPU interrupt handling sections in the architecture TRM.
3. Configure the RTC_CAL pin; see the I/O System chapter in the architecture TRM.
4. Initialize the RTC driver.
5. Enable the RTC calibration.
   5-1. Set the WRITE bit to prepare for the update of the calibration value:
   Write BACKUP_RTC_RW.WRITE = '1' (Begin the Write operation).
   5-2. Write the calibration value field:
   Write BACKUP_CAL_CTL.CALIB_VAL = "value".
   In this example, +27.125 ppm of the WCO frequency error is adjusted.
   27.125 ppm / 1.085 = 25
   The calibration value is '25'.
   Write BACKUP_CAL_CTL.CALIB_VAL = '25'.
   5-3 Write the calibration sign field:
   Write BACKUP_CAL_CTL.CALIB_SIGN = '1'.
   '1' because the RTC clock needs to be speeded up in this example.
   5-4. Clear the WRITE bit:
   Write BACKUP_RTC_RW.WRITE = '0' (End the Write operation).

The calibration correction is done by either adding (such as this example) or removing pulse counts from the oscillator divider each hour, which speeds up or slows down the clock respectively. After calibration starts, the calibration correction is performed hourly, starting at 59 minutes and 59 seconds the correction is applied as 64 ticks every 30 seconds until there have been 2 × BACKUP_CAL_CTL.CALIB_VAL adjustments.

Table 7 lists the parameters and Table 8 lists the function of the configuration part in PDL to set the calibration value of RTC.

Table 7 RTC parameters to set calibration values

Parameters	Description	Value
CALIB_VALUE	Calibration value for absolute frequency. Each step causes 128 ticks to be added or removed each hour.	25ul
CY_EN_RTC_CALIB_SIGN_POSITIVE	0 = Negative sign. Removes pulses (it takes more clock ticks to count one second) 1 = Positive sign. Adds pulses (it takes less clock ticks to count one second)	1

Operation overview

Parameters	Description	Value
CY_EN_RTC_CAL_SEL_CAL512	Selects calibration wave output signal: 0 = 512-Hz wave, not affected by calibration setting. 1 = Reserved 2 = 2-Hz wave, includes the effect of the calibration setting. 3 = 1-Hz wave, includes the effect of the calibration setting.	0

Table 8 Functions to set the RTC calibration value

Function	Description	Value
Cy_RTC_CalibrationControlEnable (uint8_t calib_val, cy_en_rtc_calib_sign_t calib_sign, cy_en_rtc_cal_sel_t cal_sel)	Sets the calibration control register and enables the calibration wave form output: calib_val = Calibration value for absolute frequency. calib_sign = Calibration sign cal_sel = Selects the calibration wave output signal	CALIB_VALUE = 25ul, CY_EN_RTC_CALIB_SIGN_POSITIVE= 1, CY_EN_RTC_CAL_SEL_CAL512 = 0

Code Listing 12 shows an example program to set the calibration value of the RTC.

Code Listing 12 Setting the calibration value

```
/* Configure the RTC_CAL pin definition (P21_7) pin */
#define RTC_CAL_PIN                     P21_7_SRSS_CAL_WAVE
#define RTC_CAL_PIN_PORT                GPIO_PRT21
#define RTC_CAL_PIN_NUM                 P21_7_NUM

/* Setting value for calibration definition */
#define CALIB_VALUE                     25u

/*configure the RTC initialize time*/
cy_stc_rtc_config_t const RTC_Config =
{
        .sec = CY_RTC_INTINAL_SEC,
        .min = CY_RTC_INTIAL_MIN,
        .hour = CY_RTC_INITIAL_HOUR,
        .dayOfWeek = CY_RTC_FRIDAY,
        .hrFormat = CY_RTC_24_HOURS,
        .date = CY_RTC_DATE,
```

Code Listing 12 Setting the calibration value

```
        .month = CY_RTC_APRIL,
        .year = CY_RTC_YEAR
};

/*configure RTC_CAL Pin*/
cy_stc_gpio_pin_config_t rtc_cal_pin_cfg =
{
        .outVal = 0,
        .driveMode = CY_GPIO_DM_STRONG_IN_OFF,
        .hsiom = RTC_CAL_PIN,
        .intEdge = CY_GPIO_INTR_DISABLE,
        .intMask = 0UL,
        .vtrip = CY_GPIO_VTRIP_CMOS,
        .slewRate = CY_GPIO_SLEW_FAST,
        .driveSel = CY_GPIO_DRIVE_FULL,
        .vregEn = 0UL,
        .ibufMode = 0UL,
        .vtripSel = 0UL,
        .vrefSel = 0UL,
        .vohSel = 0UL,
};

/*******************************************************************************
* Function Name: main
********************************************************************************
* Summary:
* Return: int
*
*******************************************************************************/
int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init();

    /* Board initialize failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
```

Code Listing 12 Setting the calibration value

```
    }

    /* Enable global interrupts */
    __enable_irq();

    /*RTC_CAL pin initialize*/
    Cy_GPIO_Pin_Init(RTC_CAL_PIN_PORT, RTC_CAL_PIN_NUM, &rtc_cal_pin_cfg);

    /*RTC initialize and wait for finish it*/
    while(Cy_RTC_Init(&RTC_Config) != CY_RTC_SUCCESS );

    /*Check the RTC busy status*/
    while(Cy_RTC_GetSyncStatus() == CY_RTC_BUSY);

    /*Calibration waveform output enable */
    Cy_RTC_CalibrationControlEnable(CALIB_VALUE, CY_RTC_CALIB_SIGN_POSITIVE,
CY_RTC_CAL_SEL_CAL512);

    for (;;)
    {

    }
}
```

2.3.5 Measuring the WCO waveform in the driver part

The example program to measure the WCO waveform in the driver is the same as in Section 2.3.3.

2.3.6 Confirming the accuracy of the WCO frequency after calibration

The WCO waveform (512 Hz) is not affected even if the calibration values (`BACKUP_CAL_CTL.CALIB_VAL` and `BACKUP_CAL_CTL.CALIB_SIGN`) are changed. However, 1-Hz and 2-Hz calibration waveforms are affected by the current calibration value. As Figure 7 shows, the 512-Hz divider is not routed through the calibration logic. You can verify the accuracy of the WCO frequency by measuring the 1-Hz or the 2-Hz waveform.

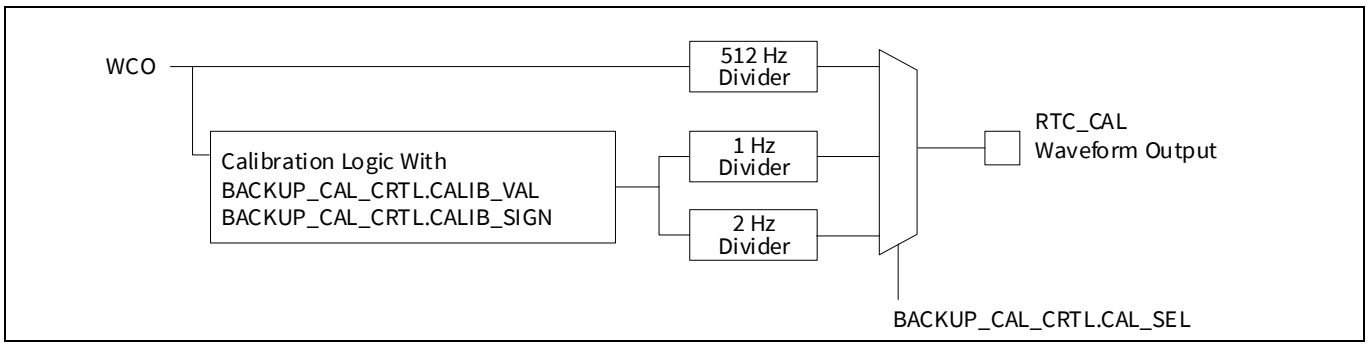

Figure 7 Clock select block diagram

Figure 8 shows an example flow to confirm the accuracy of the WCO frequency after calibration.

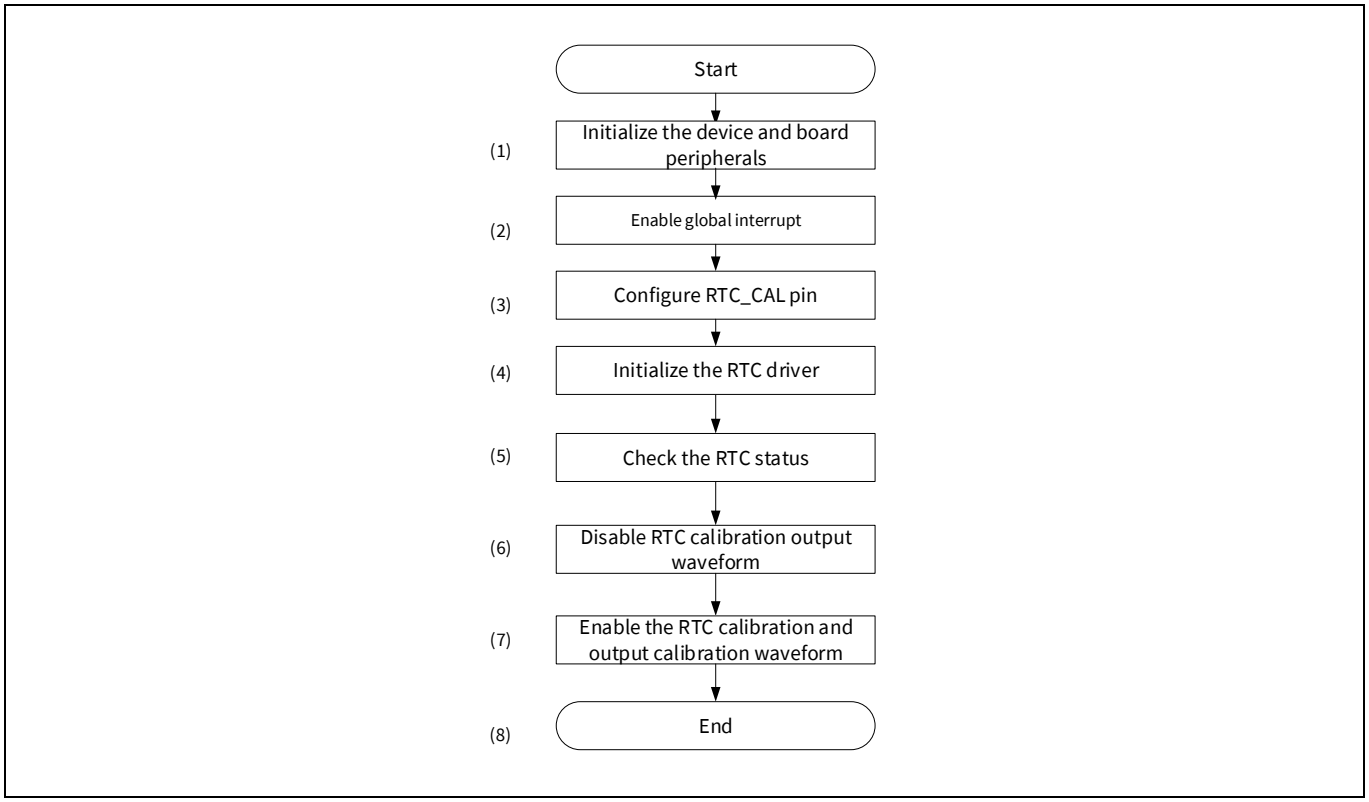

Figure 8 Confirming the accuracy of the WCO frequency after calibration

Outputting the WCO frequency of 1-Hz waveform to confirm the accuracy post-calibration involves the following steps:

1. Initialize the device and board peripherals.
2. Enable Global Interrupt (CPU Interrupt Enable). For details, see the CPU interrupt handling sections in the architecture TRM.
3. Configure the RTC_CAL pin.
4. Initialize the RTC driver.
5. Disable the RTC calibrate output waveform.
6. Enable the RTC to calibrate and enable the output waveform.

Operation overview

6-1. Set the WRITE bit to prepare to change the calibration frequency:
Write BACKUP_RTC_RW.WRITE = '1' (Begin the Write operation).

6-2. Set the calibration value.
Write BACKUP_CAL_CTL. CALIB_VAL = '25'.

6-3. Select the waveform frequency:
Write BACKUP_CAL_CTL.CAL_SEL = '3'(1-Hz).

6-4. Enable the calibration waveform:
Write BACKUP_CAL_CTL.CAL_OUT = '1' (Enable Calibration waveform output).

6-5. Clear the WRITE bit:
Write BACKUP_RTC_RW.WRITE = '0' (End the Write operation).

Then, the 1-Hz waveform can now be observed at the RTC_CAL pin. These waveforms are calibrated with BACKUP_CAL_CTL.CALIB_VAL. By measuring the waveform with external equipment (for example, a calibrated oscilloscope), you can check the effectiveness of the calibration.

Table 9 lists the parameters and Table 10 lists the function of the configuration part in the PDL to change the output frequency of the calibration waveform.

Table 9 RTC parameters to change the output frequency of the calibration waveform

Parameters	Description	Value
`CALIB_VALUE`	Calibration value for the absolute frequency. Each step causes 128 ticks to be added or removed each hour.	25ul
`CY_EN_RTC_CALIB_SIGN_POSITIVE`	0 = Negative sign. Removes pulses (it takes more clock ticks to count one second) 1 =Positive sign. Add pulses (it takes less clock ticks to count one second)	1
`CY_EN_RTC_CAL_SEL_CAL1`	Selects the calibration wave output signal: 0 = 512-Hz wave, not affected by calibration setting. 1 = Reserved 2 = 2-Hz wave, includes the effect of the calibration setting. 3 = 1-Hz wave, includes the effect of the calibration setting.	3

Table 10 RTC functions to change the output frequency of the calibration waveform

Functions	Description	Value
`Cy_RTC_CalibrationControlDisable (void)`	Disables the calibration waveform output	-

Operation overview

Functions	Description	Value
`Cy_RTC_CalibrationControlEnable (uint8_t calib_val, cy_en_rtc_calib_sign_t calib_sign, cy_en_rtc_cal_sel_t cal_sel)`	Sets the calibration control register and enables calibration wave form output: calib_val = Calibration value for absolute frequency. calib_sign = Calibration sign cal_sel = Selects calibration wave output signal	`CALIB_VALUE = 25ul, CY_EN_RTC_CALIB_SIGN_POSITIVE= 1, CY_EN_RTC_CAL_SEL_CAL1 = 3`

Code Listing 13 shows an example program to change the output frequency of the calibration waveform.

Code Listing 13 Changing the output frequency of the calibration waveform

```
/* Configure the RTC_CAL pin definition (P21_7)pin */
#define RTC_CAL_PIN                    P21_7_SRSS_CAL_WAVE
#define RTC_CAL_PIN_PORT               GPIO_PRT21
#define RTC_CAL_PIN_NUM                P21_7_NUM

/* Setting value for calibration definition */
#define CALIB_VALUE                    25u

/*configure the RTC initialize time*/
cy_stc_rtc_config_t const RTC_Config =
{
       .sec = CY_RTC_INTINAL_SEC,
       .min = CY_RTC_INTIAL_MIN,
       .hour = CY_RTC_INITIAL_HOUR,
       .dayOfWeek = CY_RTC_FRIDAY,
       .hrFormat = CY_RTC_24_HOURS,
       .date = CY_RTC_DATE,
       .month = CY_RTC_APRIL,
       .year = CY_RTC_YEAR
};

/*configure RTC_CAL Pin*/
cy_stc_gpio_pin_config_t rtc_cal_pin_cfg =
{
       .outVal = 0,
       .driveMode = CY_GPIO_DM_STRONG_IN_OFF,
```

Code Listing 13 Changing the output frequency of the calibration waveform

```
        .hsiom = RTC_CAL_PIN,
        .intEdge = CY_GPIO_INTR_DISABLE,
        .intMask = 0UL,
        .vtrip = CY_GPIO_VTRIP_CMOS,
        .slewRate = CY_GPIO_SLEW_FAST,
        .driveSel = CY_GPIO_DRIVE_FULL,
        .vregEn = 0UL,
        .ibufMode = 0UL,
        .vtripSel = 0UL,
        .vrefSel = 0UL,
        .vohSel = 0UL,
};

/*******************************************************************************
* Function Name: main
********************************************************************************
* Summary:
* Return: int
*
*******************************************************************************/
int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init();

    /* Board initialize failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Enable global interrupts */
    __enable_irq();

    /*RTC_CAL pin initialize*/
    Cy_GPIO_Pin_Init(RTC_CAL_PIN_PORT, RTC_CAL_PIN_NUM, &rtc_cal_pin_cfg);

    /*RTC initialize and wait for finish it*/
```

Code Listing 13 Changing the output frequency of the calibration waveform

```
    while(Cy_RTC_Init(&RTC_Config) != CY_RTC_SUCCESS );

    /*Check the RTC busy status*/
    while(Cy_RTC_GetSyncStatus() == CY_RTC_BUSY);

    /*Disable RTC Calibration output waveform*/
    Cy_RTC_CalibrationControlDisable();

    /*Calibration waveform output enable */
    Cy_RTC_CalibrationControlEnable(CALIB_VALUE, CY_RTC_CALIB_SIGN_POSITIVE,
CY_RTC_CAL_SEL_CAL1);

    for (;;)
    {

    }
}
```

2.3.7 Confirming the accuracy of the WCO frequency after calibration in the driver part

Code Listing 14 shows the driver part of disabling the RTC calibration output. The driver part has the same example as in Section 2.3.3 to confirm the accuracy of WCO frequency after calibration.

Code Listing 14 Disabling RTC calibration in the driver part

```
cy_en_rtc_status_t Cy_RTC_CalibrationControlDisable(void)
{
    cy_en_rtc_status_t retVal = CY_RTC_INVALID_STATE;
    uint32_t interruptState;

    /* The RTC AHB registers can be updated only under condition that the
    *  Write bit is set and the RTC busy bit is cleared (RTC_BUSY = 0).*/
    interruptState = Cy_SysLib_EnterCriticalSection();
    retVal = Cy_RTC_WriteEnable(CY_RTC_WRITE_ENABLED);
    if (CY_RTC_SUCCESS == retVal)
    {
        BACKUP_CAL_CTL =
        _VAL2FLD(BACKUP_CAL_CTL_CAL_OUT, 0U);

        /* Clear the RTC Write bit to finish RTC update */
```

Code Listing 14 Disabling RTC calibration in the driver part

```
        retVal = Cy_RTC_WriteEnable(CY_RTC_WRITE_DISABLED);
    }
    Cy_SysLib_ExitCriticalSection(interruptState);

    return(retVal);
}
```

Glossary

RTC function

Function of the RTC logic.

RTC value

Time tracked by the RTC function. See the *Real-Time Clock* section in the “Real-Time Clock” chapter of the architecture TRM for details.

Calibration feature

Feature to correct any frequency error of the RTC values.

Integer values

Whole numbers and not Binary Coded Decimal (BCD).

Leap-year correction

The RTC implements automatic leap year correction for the Date field (day of the month). See the *Real-Time Clock* section in the “Real-Time Clock” chapter of the architecture TRM for details.

ILO

Internal Low-speed Oscillator is the embedded oscillator inside XMC7000 device. See the “Clocking System” chapter of the architecture TRM for details.

CLK_LF

Low-Frequency Clock is selectable. See the “Clocking System” chapter of the architecture TRM for details.

WCO

Watch Crystal Oscillator is the external crystal that is input from the external WCO_IN and WCO_OUT pins. See the “Clocking System” chapter of the architecture TRM for details.

Calibration waveform

The signal outputs from the RTC_CAL pin for adjusting the RTC. See the *WCO Calibration* section in the “Real-Time Clock” chapter of the architecture TRM for details.

Ideal waveform

Waveform that has an accurate frequency

Clock screen

The dot-matrix display or the segment display that shows the actual time in a vehicle.

User system

User-developed electronic board/system where XMC7000 device is mounted.

Glossary

User firmware

User-developed software that runs on the CPU.

Local variables

Variables used by user firmware.

User buffer

The local variables assigned by user firmware in RAM.

AHB-Lite bus interface

The bus interface between CPU Subsystem and Peripheral interconnect. See the *Top-Level Architecture* section in the “Introduction” chapter of the architecture TRM for details.

User registers

Resisters that can be accessed by user firmware. Read and write operations are possible. However, values do not always reflect the latest data. See the *Real-Time Clock* section in the “Real-Time Clock” chapter of the architecture TRM for details.

RTC registers

Resisters that can be accessed by hardware. Read and write operations are not possible by user firmware. Values always reflect the latest data. See the *Real-Time Clock* section in the “Real-Time Clock” chapter of the architecture TRM for details.

RTC fields

Bit fields of RTC values in user registers and RTC registers. See the *Real-Time Clock* section in the “Real-Time Clock” chapter of the architecture TRM for details.

Alarm fields

Bit fields of alarm values in user registers and RTC registers. See the *Real-Time Clock* section in the “Real-Time Clock” chapter of the architecture TRM for details.

Config fields

Bit fields of configuration bits in user registers and RTC registers. See the *Real-Time Clock* section in the “Real-Time Clock” chapter of the architecture TRM for details.

WCO device and logic

WCO device is an external crystal oscillator. WCO logic is a circuit inside XMC7000 device.

Power mode / Device power mode

XMC7000 family has several power modes in the order of decreasing power consumption. See the “Device Power Modes” chapter of the architecture TRM for details.

Glossary

Alarm function

An interrupt is generated when the RTC fields and the alarm fields match. See the *Alarm Feature* section in the "Real-Time Clock" chapter of the architecture TRM for details.

Interval time

A time of periodic interrupt.

Power-on reset (POR)

Reset operation after power-on sequence. See the "Rest System" chapter of the architecture TRM for details.

Waking up

Restarting user firmware from stand-by mode, Sleep, Low-Power Sleep, DeepSleep, Hibernate. See the "Device Power Modes" chapter of the architecture TRM for details.

References

Contact Infineon Technical support to obtain more XMC7000 family documents.

[1] Device datasheet

- XMC7100 series 32-bit Arm® Cortex®-M7 microcontroller datasheet
- XMC7200 series 32-bit Arm® Cortex®-M7 microcontroller datasheet

[2] Technical reference manuals

- XMC7000 MCU family architecture technical reference manual
- XMC7100 registers technical reference manual
- XMC7200 registers technical reference manual

[3] Application note

- AN234226 - Interrupt usage in XMC7000 family

Revision history

Document revision	Date	Description of changes
**	2021-12-02	Initial release
*A	2022-05-11	Updated sections 2.1, 2.2, and 2.3 code snippets and PDL driver APIs and description contents
*B	2023-04-13	Removed other reference section and updated relevant contents Updated links in References section Updated code snippets comments

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2023-04-13
Published by

Infineon Technologies AG
81726 Munich, Germany

Do you have a question about this document?
Email: erratum@infineon.com

Document reference
002-33882 Rev. *B

Important notice
The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.