How to use RTC in TRAVEO™ T2G family

About this document

Scope and purpose

This application note describes how to use the real-time clock (RTC) in TRAVEO™ T2G family MCUs.

Intended audience

This document is intended for anyone who uses the RTC of the TRAVEO™ T2G family.

Table of contents

1 Introduction

The RTC in TRAVEO™T2G 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, hours, minutes, and seconds accurately. The alarm feature can generate interrupts to the CPU with a programmable setting. The calibration feature can correct a frequency error of the WCO and low-power external crystal oscillator (LPECO). The backup registers can 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 described and terminology used in this application note, see the Real-Time Clock chapter of the architecture reference manual.

1.1 RTC features

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 and 4 MHz to 8 MHz LPECO
  - Calibration waveforms output
  - Supports 512 Hz, 1 Hz, and 2 Hz
- Backup registers

***Note**: See the device-specific datasheet to confirm whether LPECO is present.*

2 Operation overview

The RTC function keeps track of the current time accurately up to seconds and therefore, can be used as an accurate time source in automotive applications. This application note describes clock screens in vehicles as a sample use case. The time on the clock screen is updated accurately and periodically.

Figure 1 shows the RTC function block diagram and an example of the user system.

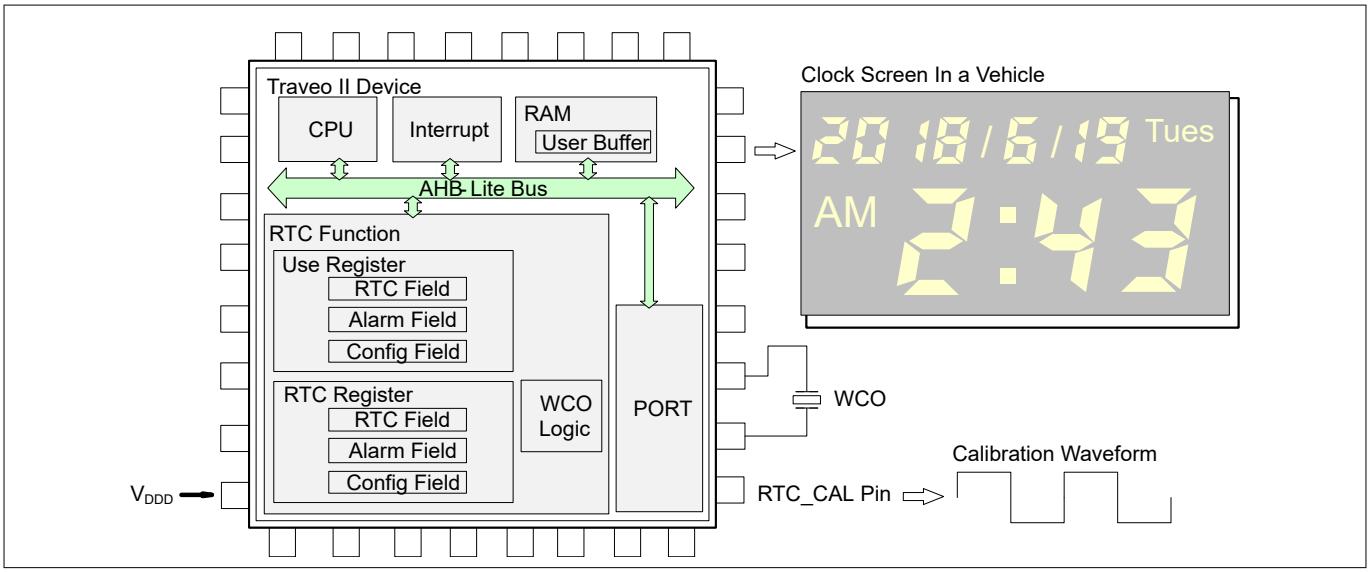

Figure 1 Example structure of the user system using the RTC function

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

For updating and displaying the current time on the clock screen of vehicles, the RTC function is generally used with periodic interrupts. The CPU and the RTC function are connected via an AHB-Lite interface, which provides a firmware access interface.

The CPU reads the current time from the RTC function, stores the real-time data in the user buffers, which are local variables assigned in RAM, and outputs the real-time data to the external clock screen. Moreover, the RTC function can also output a calibration waveform.

User registers and the RTC registers have RTC fields, alarm fields, and configuration fields. 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 reference manual and register reference manual for the details on RTC fields, Alarm fields, and Config fields.

The WCO device and logic, the LPECO and logic, and the RTC function run 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 RTC function including interrupts
2. Reading time and date
3. Updating time and date
4. Calibrating WCO

Before using the RTC function, an initialization routine needs to be executed. The initializing operation includes the alarm interrupt setting. The interrupt is caused by the alarm function which has fields corresponding to the RTC field registers. Basic RTC settings provides the example setting for generating interrupts every 30 seconds with two alarm functions.

To display the current time on the clock screen of a vehicle, the CPU reads a current time from the RTC field registers. Reading TIME and DATE provides an example to read the current time from the RTC field.

There are instances where you might have to update the RTC fields registers, for example for Daylight Saving Time (DST). For DST, the Hour field needs to be updated. Updating TIME and DATE provides an example to the update the current time of the RTC field.

You can use ILO, CLK_LF, WCO, or LPECO as a source clock for the RTC function. However, WCO or LPECO is recommended as it is more accurate. Calibrating the WCO provides an example to calibrate the time of the RTC function.

2.1 Basic RTC settings

This section describes the operation of the RTC function with the following assumptions.

This section also explains how to configure the RTC based on a use case using the Sample Driver Library (SDL) provided by Infineon. The code snippets in this application note are part of SDL. See Other references for the SDL.

SDL basically 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 section provides an example of using the RTC function, as demonstrated by the following use case. This use case shows how to initialize items such as input clock, time, and date to enable the RTC function. Additionally, this use case also enables the ALARM function. When the ALARM setting time and RTC time match, an interrupt is generated.

Use case:

- Source clock: Internal Low-speed oscillator (ILO)
- Setting year: 2019
- Setting month: August
- Setting date: 21
- Setting day of week: Monday
- Setting time: 12:00:00
- Setting HR mode: 24HR
- ALARM1: Every 0 seconds
- IRQ number for ALARM1: 3

2.1.2 Initializing the RTC function

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

Figure 2 shows an example flow to configure the basic RTC settings.

2 Operation overview

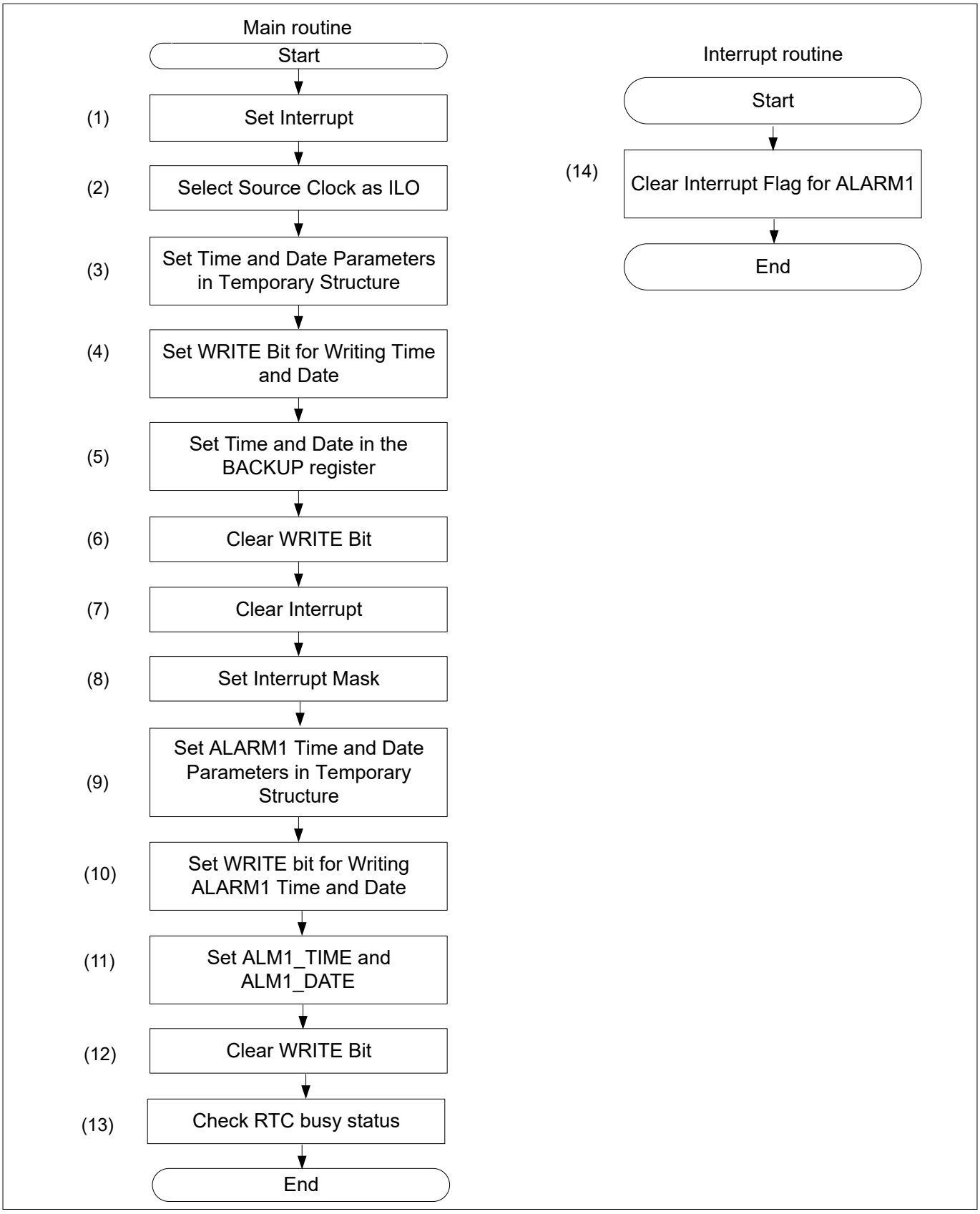

Figure 2 Example flow to configure basic RTC settings and interrupt routine

The following procedure shows the basic setup procedure for the RTC function:

1. Set interrupt

2. Select the source clock as ILO: Write BACKUP_CTL.CLK_SEL = CY_RTC_CLK_SRC_ILO_0 (ILO)
3. Set the time and date parameters in temporary structure: TIME parameter (Second=0, minute=0, hour=12, HR mode=0(24HR), day of week=1(Monday))DATE parameter (date=21, month=8, year=19 (2019))
4. Set the WRITE bit for writing the time and date: Write BACKUP_RTC_RW.WRITE = '1'
5. Set the time and date for the BACKUP_RTC_TIME and BACKUP_RTC_DATE registers: Write BACKUP_RTC_TIME = time parameter Write BACKUP_RTC_DATE = date parameter
6. Clear the WRITE bit to update the date and time in the RTC registers: Write BACKUP_RTC_RW.WRITE = '0'
7. Clear the interrupt
8. Set the interrupt mask
9. Set the ALARM1 time and date parameters in temporary structure
10. Set the WRITE bit for writing the ALARM1 time and the date
11. Set the ALARM1 time and date for the BACKUP_ALM1_TIME and BACKUP_ALM1_DATE registers: When the RTC fields and the alarm fields match, an interrupt occurs. Write BACKUP_ALM1_TIME = ALARM1 time parameter (0 seconds) Write BACKUP_ALM1_DATE = ALARM1 date parameter (No ALARM in Date parameter)
12. Clear the WRITE bit to update the date and time in the RTC registers: Write BACKUP_RTC_RW.WRITE = '0'
13. Check the RTC busy status for RTC register update
14. When the ALARM1 interrupt occurs, clear the interrupt flag

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 AN219842.

2.1.3 Configuring the RTC function

The configuration part in SDL for RTC and ALARM settings is described in two sections:

- Table 1, which lists the parameters
- Table 2 which lists the functions

Table 1 RTC and ALARM parameters

Parameters	Description	Value
RTC_config.sec	Calendar seconds, 0-59	RTC_INITIAL_DATE_SEC = 0ul
RTC_config.min	Calendar minutes, 0-59	RTC_INITIAL_DATE_MIN = 0ul
RTC_config.hour	Calendar hours, value depending on 12/24HR mode	RTC_INITIAL_DATE_HOUR = 12ul
RTC_config.hrMode	Select 12/24HR mode: 1=12HR, 0=24HR CY_RTC_12_HOURS = 1ul, CY_RTC_24_HOURS = 0ul	RTC_INITIAL_DATE_HOUR_FORMAT = CY_RTC_24_HOURS
RTC_config.dayOfWeek	Calendar Day of the week, 1-7 You can define the values, but it is recommended to set 1=Monday.	RTC_INITIAL_DATE_DOW = 1ul

(table continues...)

Table 1 (continued) RTC and ALARM parameters

Parameters	Description	Value
RTC_config.date	Calendar Day of the Month, 1-31 Automatic Leap Year Correction	RTC_INITIAL_DATE_DOM = 21ul
RTC_config.month	Calendar Month, 1-12	RTC_INITIAL_DATE_MONTH = 8ul
RTC_config.year	Calendar year, 0-99	RTC_INITIAL_DATE_YEAR = 19ul
alarm.sec	Alarm seconds, 0-59	0ul
alarm.sec_en	Alarm second enable: 0=ignore, 1=match	CY_RTC_ALARM_ENABLE = 1ul
alarm.min	Alarm minutes, 0-59	0ul
alarm.min_en	Alarm minutes enable: 0=ignore, 1=match	CY_RTC_ALARM_DISABLE = 0ul
alarm.hour	Alarm hours, value depending on 12/24HR mode	0ul
alarm.hour_en	Alarm hour enable: 0=ignore, 1=match	CY_RTC_ALARM_DISABLE = 0ul
alarm.dayOfWeek	Calendar Day of the week, 1-7 You can define the values, but it is recommended to set 1=Monday	1ul
alarm.dayOfWeek_en	Alarm Day of the Week enable: 0=ignore, 1=match	CY_RTC_ALARM_DISABLE = 0ul
alarm.date	Calendar Day of the Month, 1-31 Automatic Leap Year Correction	1ul
alarm.date_en	Alarm Day of the Month enable: 0=ignore, 1=match	CY_RTC_ALARM_DISABLE = 0ul
alarm.month	Alarm Month, 1-12	1ul
alarm.month_en	Alarm Month enable: 0=ignore, 1=match	CY_RTC_ALARM_DISABLE = 0ul
alarm.alm_en	Master enable for alarm 1. 0: Alarm 1 is disabled. Fields for date and time are ignored. 1: Alarm 1 is enabled. If none of the date and time fields are enabled, this alarm triggers once every second.	CY_RTC_ALARM_ENABLE = 0ul

Table 2 RTC and ALARM functions

Functions	Description	Value
Cy_Rtc_clock_source (clock_source)	Set the RTC input clock source Clock source: Input clock source	CY_RTC_CLK_SRC_ILO_0
Cy_Rtc_Init (*config)	Initialize the RTC driver and return the RTC register address config: RTC configuration structure address	&RTC_config
Cy_Rtc_SetDateAndTime (*dateTime)	Set the time and date values to the RTC_TIME and RTC_DATE registers. dateTime: RTC configuration structure address	config
Cy_Rtc_ConstructTimeDate (*timeDate, time, date)	Returns Integer time and Integer date in the format used in APIs from individual elements passed. timeDate: Structure address of time and date time: Time configuration structure address for the RTC_TIME register set date: Date configuration structure address for the RTC_TIME register set	dateTime, &tmpTime, &tmpDate
Cy_Rtc_ClearInterrupt (interruptMask)	Clear the RTC interrupt interruptMask: The bit mask of interrupt to clear	CY_RTC_INTR_ALARM1=0x1ul
Cy_Rtc_SetInterruptMask (interruptMask)	Set the RTC interrupt interruptMask: The bit mask of interrupt to set	CY_RTC_INTR_ALARM1=0x1ul
Cy_Rtc_SetAlarmDateAndTime (alarmDateTime, alarmIndex)	Sets alarm time and date values into the ALMx_TIME and ALMx_DATE registers alarmDateTime: The alarm configuration structure alarmIndex: The alarm index to be configured	&alarm, CY_RTC_ALARM_1=0ul

(table continues...)

Table 2 (continued) RTC and ALARM functions

Functions	Description	Value
Cy_Rtc_ConstructAlarmTimeDate (*alarmDateTime, *alarmTime, *alarmDate)	Returns the Integer time and Integer date in the format used in APIs from individual elements passed for alarm alarmDateTime: Structure address of alarm time and date alarmTime: Alarm time configuration structure address for the ALMx_TIME register set alarmDate: Alarm date configuration structure address for the ALMx_DATE register set	alarmDateTime, &tmpAlarmTime, &tmpAlarmDate
_VAL2FLD (field, value)	Mask and shift a bit field value for use in a register bit range. Field: Name of the register bit field. Value: Value of the bit field. This parameter is interpreted as a uint32_t type.	–

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

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

- **BACKUP**->un**RTC_TIME** register is the BACKUP_RTC_TIME register mentioned in the register reference manual. Other registers are also described in the same manner

See `cyip_backup_v3.h` under `hdr/rev_x/ip` for more information on the union and structure representation of registers.

2 Operation overview

Code Listing 1 Example program to configure the RTC

```
cy_stc_rtc_config_t const RTC_config =
{
    /* Initiate time and date */
    /* Configure initial time and date parameters. See Table 1*/
    .sec       = RTC_INITIAL_DATE_SEC,
    .min       = RTC_INITIAL_DATE_MIN,
    .hour      = RTC_INITIAL_DATE_HOUR,
    .hrMode    = RTC_INITIAL_DATE_HOUR_FORMAT,
    .dayOfWeek = RTC_INITIAL_DATE_DOW,
    .date      = RTC_INITIAL_DATE_DOM,
    .month     = RTC_INITIAL_DATE_MONTH,
    .year      = RTC_INITIAL_DATE_YEAR,
};

/*Configure ALARM1 time and date parameters. See Table 1*/
cy_stc_rtc_alarm_t const alarm =
{
    .sec          = 0ul,
    .sec_en       = CY_RTC_ALARM_ENABLE,
    .min          = 0ul,
    .min_en       = CY_RTC_ALARM_DISABLE,
    .hour         = 0ul,
    .hour_en      = CY_RTC_ALARM_DISABLE,
    .dayOfWeek    = 1ul,
    .dayOfWeek_en = CY_RTC_ALARM_DISABLE,
    .date         = 1ul,
    .date_en      = CY_RTC_ALARM_DISABLE,
    .month        = 1ul,
    .month_en     = CY_RTC_ALARM_DISABLE,
    .alm_en       = CY_RTC_ALARM_ENABLE
};

int main(void)
{
    SystemInit();

    __enable_irq(); /* Enable global interrupts. */

    Cy_GPIO_Pin_Init(USER_LED_PORT, USER_LED_PIN, &user_led_port_pin_cfg);

/*(1) Set interrupt.*/
    cy_stc_sysint_irq_t irq_cfg = (cy_stc_sysint_irq_t)
    {
        .sysIntSrc  = srss_interrupt_backup_IRQn,
        .intIdx     = CPUIntIdx0_IRQn,
        .isEnabled  = true,
    };
    Cy_SysInt_InitIRQ(&irq_cfg);
    Cy_SysInt_SetSystemIrqVector(irq_cfg.sysIntSrc, RTC_Handler);
    NVIC_SetPriority(CPUIntIdx0_IRQn, 3ul);
    NVIC_ClearPendingIRQ(CPUIntIdx0_IRQn);
```

```
    NVIC_EnableIRQ(CPUIntIdx0_IRQn);

    /* Set the ILO_0 as the clock source to the RTC block */
    /* Select the source clock as ILO. See Code Listing 3.*/

    Cy_Rtc_clock_source(CY_RTC_CLK_SRC_ILO_0);

    /* Wait for alarm to be set */
    /*Set the RTC initial time, date parameter. See Code Listing 4 */

    while(Cy_Rtc_Init(&RTC_config) != CY_RET_SUCCESS);

    /* Clear any pending interrupts */
    /*Clear interrupt. See Code Listing 8. */
    Cy_Rtc_ClearInterrupt(CY_RTC_INTR_ALARM1);

    /*Configures the source (Alarm1) that trigger the interrupts */
    /*Set the interrupt mask. See Code Listing 9. */

    Cy_Rtc_SetInterruptMask(CY_RTC_INTR_ALARM1);

    /* Wait for alarm to be set */
    /*Set the ALARM1 time and date parameters. See Code Listing 10. */

    while(Cy_Rtc_SetAlarmDateAndTime(&alarm,CY_RTC_ALARM_1) != CY_RET_SUCCESS);


    for(;;)
    {
        Cy_Rtc_GetDateAndTime(&Read_DateTime);
    }
}
```

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

Code Listing 2 Example program of RTC interrupt routine for alarm1

```
void Cy_Rtc_Alarm1Interrupt(void)   /* Interrupt routine for alarm1 function */
{
    /* Clear any pending interrupts */
    /* (14) Clear the Interrupt flag for alarm1.  See Code Listing 8.
    Cy_Rtc_ClearInterrupt(CY_RTC_INTR_ALARM1);
}
```

2.1.4 Example program to configure the RTC function in the driver part

Code Listing 3 to Code Listing 11 shows an example program to configure the RTC in the driver part.

Code Listing 3 Example program to configure RTC input clock source set in the driver part

```
void Cy_Rtc_clock_source(cy_en_rtc_clock_src_t clock_source)
{
    /* (2) Select the source clock as ILO */
    BACKUP->unCTL.stcField.u2CLK_SEL = clock_source;
}
```

Code Listing 4 Example program to initialize the RTC in the driver part

```
cy_en_rtc_status_t Cy_Rtc_Init(cy_stc_rtc_config_t const *config)
{
    cy_en_rtc_status_t retVal;

    if(NULL != config)
    {
    /* This function sets the time, date parameters, WRITE bit. See Code Listing 5. */
       retVal = Cy_Rtc_SetDateAndTime(config);
    }
    else
    {
        retVal = CY_RTC_INVALID_STATE;
    }
    return(retVal);
}
```

Code Listing 5 Example program to set date and time registers in the driver part

```
cy_en_rtc_status_t Cy_Rtc_SetDateAndTime(cy_stc_rtc_config_t const *dateTime)
{
    uint32_t tmpTime;
    uint32_t tmpDate;
    uint32_t tmpDaysInMonth;
    uint32_t interruptState;

    cy_en_rtc_status_t retVal = CY_RTC_BAD_PARAM;

    /* Check the input parameters valid ranges */
    /* Check if configuration parameter values are valid */
    if((dateTime->month > 0u) && (dateTime->month <= CY_RTC_MONTHS_PER_YEAR) && (dateTime->year
<=
     CY_RTC_MAX_YEAR))
    {
        tmpDaysInMonth = Cy_Rtc_DaysInMonth(dateTime->month, (dateTime->year +
        CY_RTC_TWO_THOUSAND_YEARS));

        /* Check if the date is in the valid range */
        if((dateTime->date > 0u) && (dateTime->date <= tmpDaysInMonth))
        {
        /* Set the initial time and date parameter in temporary structure. See Code Listing 6.
*/
            Cy_Rtc_ConstructTimeDate(dateTime, &tmpTime, &tmpDate);

            /* The RTC AHB register can be updated only under condition that the
            *  Write bit is set and the RTC busy bit is cleared (CY_RTC_BUSY = 0).
            */
            interruptState = Cy_SysLib_EnterCriticalSection();
            /* Set the WRITE bit for writing the time and date. See Code Listing 7. */
            retVal = Cy_Rtc_WriteEnable(CY_RTC_WRITE_ENABLED);
            if(retVal == CY_RTC_SUCCESS)
            {
                /* (5) Set for BACKUP_RTC_TIME and BACKUP_RTC_DATE registers  */
                BACKUP->unRTC_TIME.u32Register = tmpTime;
                BACKUP->unRTC_DATE.u32Register = tmpDate;

                /* Clear the RTC Write bit to finish RTC register update */
                /* Clear the WRITE bit. See Code Listing 7. */
                retVal = Cy_Rtc_WriteEnable(CY_RTC_WRITE_DISABLED);
            }
            Cy_SysLib_ExitCriticalSection(interruptState);
        }
    }
    return(retVal);
}
```

2 Operation overview

Code Listing 6 Example program to construct the time and date in the driver part

```
static void Cy_Rtc_ConstructTimeDate(cy_stc_rtc_config_t const *timeDate, uint32_t *time,
                                                                          uint32_t *date)
{
    uint32_t tmpTime;
    uint32_t tmpDate;

    /* Prepare the RTC TIME value based on the structure obtained */
    /* (3) Set the initial time and date parameters in temporary structure. */
    /* Set to Time parameters */
    tmpTime = (_VAL2FLD(BACKUP_RTC_TIME_RTC_SEC, (timeDate->sec)));
    tmpTime |= (_VAL2FLD(BACKUP_RTC_TIME_RTC_MIN, (timeDate->min)));

    /* Read the current hour mode to know how many hour bits to convert.
    *  In the 24-hour mode, the hour value is presented in [21:16] bits in the
    *  Integer format.
    *  In the 12-hour mode, the hour value is presented in [20:16] bits in the
    *  Integer format and
    *  bit [21] is present: 0 - AM; 1 - PM.
    */
    if(timeDate->hrMode != CY_RTC_24_HOURS)
    {
        if(CY_RTC_AM != timeDate->amPm)
        {
            /* Set the PM bit */
            tmpTime |= CY_RTC_BACKUP_RTC_TIME_RTC_PM;
        }
        else
        {
            /* Set the AM bit */
            tmpTime &= ((uint32_t) ~CY_RTC_BACKUP_RTC_TIME_RTC_PM);
        }
        tmpTime |= BACKUP_RTC_TIME_CTRL_12HR_Msk;
        tmpTime |=
        (_VAL2FLD(BACKUP_RTC_TIME_RTC_HOUR,
        ((timeDate->hour) & ((uint32_t) ~CY_RTC_12HRS_PM_BIT))));
    }
    else
    {
        tmpTime &= ((uint32_t) ~BACKUP_RTC_TIME_CTRL_12HR_Msk);
        tmpTime |= (_VAL2FLD(BACKUP_RTC_TIME_RTC_HOUR, (timeDate->hour)));
    }
    tmpTime |= (_VAL2FLD(BACKUP_RTC_TIME_RTC_DAY, (timeDate->dayOfWeek)));

    /* Prepare the RTC Date value based on the structure obtained */
    /* Set to Date parameters */
    tmpDate  = (_VAL2FLD(BACKUP_RTC_DATE_RTC_DATE,(timeDate->date)));
    tmpDate |= (_VAL2FLD(BACKUP_RTC_DATE_RTC_MON, (timeDate->month)));
    tmpDate |= (_VAL2FLD(BACKUP_RTC_DATE_RTC_YEAR, (timeDate->year)));

    /* Update the parameter values with prepared values */
    *time = tmpTime;
```

```
    *date = tmpDate;
}
```

Code Listing 7 Example program to write enable in the driver part

```
cy_en_rtc_status_t Cy_Rtc_WriteEnable(uint32_t writeEnable)
{
    cy_en_rtc_status_t retVal = CY_RTC_INVALID_STATE;

    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
        */
        if((CY_RTC_BUSY != Cy_Rtc_GetSyncStatus()) && (!_FLD2BOOL(BACKUP_RTC_RW_READ, BACKUP->
            unRTC_RW.u32Register)))
        {
            /* (4) Set the WRITE bit for writing the time and date. */
            /* (10) Set the WRITE bit for writing ALARM1 time and date. */
            BACKUP->unRTC_RW.u32Register |= BACKUP_RTC_RW_WRITE_Msk;
            retVal = CY_RTC_SUCCESS;
        }
    }
    else
    {
        /* Clearing Write Bit to complete write procedure */
        /* (6) Clear the WRITE bit. */
        /* (12) Clear the WRITE bit. */
        BACKUP->unRTC_RW.u32Register &= ((uint32_t) ~BACKUP_RTC_RW_WRITE_Msk);

        /* wait until CY_RTC_BUSY bit is cleared */
        /* (7) Check the RTC busy status. */
        /* (13) Check the RTC busy status. */
        while(CY_RTC_BUSY == Cy_Rtc_GetSyncStatus());

        retVal = CY_RTC_SUCCESS;
    }

    return(retVal);
}
```

Code Listing 8 Example program to clear the interrupt in the driver part

```
void Cy_Rtc_ClearInterrupt(uint32_t interruptMask)
{
    /* (7) Clear interrupt */
    BACKUP->unINTR.u32Register = interruptMask;

    (void) BACKUP->unINTR.u32Register;
}
```

Code Listing 9 Example program to set interrupt mask in driver part

```
void Cy_Rtc_SetInterruptMask(uint32_t interruptMask)
{
    /* (8) Set interrupt mask */
    BACKUP->unINTR_MASK.u32Register = interruptMask;
}
```

2 Operation overview

Code Listing 10 Example program to set the alarm time and date in the driver part

```
cy_en_rtc_status_t Cy_Rtc_SetAlarmDateAndTime(cy_stc_rtc_alarm_t const *alarmDateTime,
cy_en_rtc_alarm_t alarmIndex)
{
    uint32_t tmpAlarmTime;
    uint32_t tmpAlarmDate;
    uint32_t tmpYear;
    uint32_t tmpDaysInMonth;
    uint32_t interruptState;
    cy_en_rtc_status_t retVal = CY_RTC_BAD_PARAM;

    /* Read the current RTC time and date to validate the input parameters */
    Cy_Rtc_SyncRegisters();

    tmpYear = CY_RTC_TWO_THOUSAND_YEARS + (_FLD2VAL(BACKUP_RTC_DATE_RTC_YEAR,
                                                    BACKUP->unRTC_DATE.u32Register));

    /* Parameters validation */
    /* Check if configuration parameter values are valid */
    if((alarmDateTime->month > 0u) && (alarmDateTime->month <= CY_RTC_MONTHS_PER_YEAR))
    {
        tmpDaysInMonth = Cy_Rtc_DaysInMonth(alarmDateTime->month, tmpYear);

        if((alarmDateTime->date > 0u) && (alarmDateTime->date <= tmpDaysInMonth))
        {
        /* Set the ALARM1 time and date parameters in temporary structure. See Code Listing 11.
*/
            Cy_Rtc_ConstructAlarmTimeDate(alarmDateTime, &tmpAlarmTime, &tmpAlarmDate);

            /* The RTC AHB register 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();

            /* Set the WRITE bit for writing ALARM1 time and date. See Code Listing 7. */
            retVal = Cy_Rtc_WriteEnable(CY_RTC_WRITE_ENABLED);
            if(CY_RTC_SUCCESS == retVal)
            {
                /* Update the AHB RTC registers with formed values */
                if(alarmIndex != CY_RTC_ALARM_2)
                {
                    /* (11) Set the ALM1_TIME and ALM1_DATE registers */
                    BACKUP->unALM1_TIME.u32Register = tmpAlarmTime;
                    BACKUP->unALM1_DATE.u32Register = tmpAlarmDate;
                }
                else
                {
                    BACKUP->unALM2_TIME.u32Register = tmpAlarmTime;
                    BACKUP->unALM2_DATE.u32Register = tmpAlarmDate;
                }

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

```
                /* Clear the WRITE bit. See Code Listing 7.
                retVal = Cy_Rtc_WriteEnable(CY_RTC_WRITE_DISABLED);
            }
            Cy_SysLib_ExitCriticalSection(interruptState);
        }
    }
    return(retVal);
}
```

Code Listing 11 Example program to construct the alarm time and date in the driver part

```
static void Cy_Rtc_ConstructAlarmTimeDate(cy_stc_rtc_alarm_t const *alarmDateTime,
                                          uint32_t *alarmTime,
                                          uint32_t *alarmDate)
{
    uint32_t tmpAlarmTime;
    uint32_t tmpAlarmDate;
    uint32_t hourValue;

    /* Prepare the RTC ALARM value based on the structure obtained */
    /* (9) Set the ALARM1 time and date parameter in temporary structure. */
    /* Set to ALARM Time parameters */
    tmpAlarmTime  = (_VAL2FLD(BACKUP_ALM1_TIME_ALM_SEC, (alarmDateTime->sec)));
    tmpAlarmTime |= (_VAL2FLD(BACKUP_ALM1_TIME_ALM_SEC_EN, alarmDateTime->sec_en));
    tmpAlarmTime |= (_VAL2FLD(BACKUP_ALM1_TIME_ALM_MIN, (alarmDateTime->min)));
    tmpAlarmTime |= (_VAL2FLD(BACKUP_ALM1_TIME_ALM_MIN_EN, alarmDateTime->min_en));

    /* Read the current hour mode to know how many hour bits to convert.
    *  In the 24-hour mode, the hour value is presented in [21:16] bits in the
    *  Integer format.
    *  In the 12-hour mode, the hour value is presented in [20:16] bits in the
    *  Integer format and bit [21] is present: 0 - AM; 1 - PM
    */
    Cy_Rtc_SyncRegisters();
    if(CY_RTC_24_HOURS != Cy_Rtc_GetHoursFormat())
    {
        /* Convert the hour from the 24-hour mode into the 12-hour mode */
        if(alarmDateTime->hour >= CY_RTC_HOURS_PER_HALF_DAY)
        {
            /* The current hour is more than 12 in the 24-hour mode. Set the PM
            *  bit and converting hour: hour = hour - 12
            */
            hourValue = (uint32_t) alarmDateTime->hour - CY_RTC_HOURS_PER_HALF_DAY;
            hourValue = ((0u != hourValue) ? hourValue : CY_RTC_HOURS_PER_HALF_DAY);
            tmpAlarmTime |=
            CY_RTC_BACKUP_RTC_TIME_RTC_PM | (_VAL2FLD(BACKUP_ALM1_TIME_ALM_HOUR, (hourValue)));
        }
        else if(alarmDateTime->hour < 1u)
        {
            /* The current hour in the 24-hour mode is 0 which is equal to 12:00 AM */
            tmpAlarmTime = (tmpAlarmTime & ((uint32_t) ~CY_RTC_BACKUP_RTC_TIME_RTC_PM)) |
            (_VAL2FLD(BACKUP_ALM1_TIME_ALM_HOUR, CY_RTC_HOURS_PER_HALF_DAY));
        }
        else
        {
            /* The current hour is less than 12. Set the AM bit */
            tmpAlarmTime = (tmpAlarmTime & ((uint32_t) ~CY_RTC_BACKUP_RTC_TIME_RTC_PM)) |
            (_VAL2FLD(BACKUP_ALM1_TIME_ALM_HOUR, (alarmDateTime->hour)));
        }
        tmpAlarmTime |= BACKUP_RTC_TIME_CTRL_12HR_Msk;
    }
    else
```

```
    {
        tmpAlarmTime |= (_VAL2FLD(BACKUP_ALM1_TIME_ALM_HOUR, (alarmDateTime->hour)));
        tmpAlarmTime &= ((uint32_t) ~BACKUP_RTC_TIME_CTRL_12HR_Msk);
    }
    tmpAlarmTime |= (_VAL2FLD(BACKUP_ALM1_TIME_ALM_HOUR_EN, alarmDateTime->hour_en));
    tmpAlarmTime |= (_VAL2FLD(BACKUP_ALM1_TIME_ALM_DAY, (alarmDateTime->dayOfWeek)));
    tmpAlarmTime |= (_VAL2FLD(BACKUP_ALM1_TIME_ALM_DAY_EN, alarmDateTime->dayOfWeek_en));

    /* Prepare the RTC ALARM DATE value based on the obtained structure */
    /* Set  ALARM date parameters */
    tmpAlarmDate  = (_VAL2FLD(BACKUP_ALM1_DATE_ALM_DATE, (alarmDateTime->date)));
    tmpAlarmDate |= (_VAL2FLD(BACKUP_ALM1_DATE_ALM_DATE_EN, alarmDateTime->date_en));
    tmpAlarmDate |= (_VAL2FLD(BACKUP_ALM1_DATE_ALM_MON, (alarmDateTime->month)));
    tmpAlarmDate |= (_VAL2FLD(BACKUP_ALM1_DATE_ALM_MON_EN, alarmDateTime->month_en));
    tmpAlarmDate |= (_VAL2FLD(BACKUP_ALM1_DATE_ALM_EN, alarmDateTime->alm_en));
    /* Update the parameter values with prepared values */
    *alarmTime = tmpAlarmTime;
    *alarmDate = tmpAlarmDate;
}
```

2.2 Reading TIME and DATE

To display the current time on the clock screen of a vehicle, the CPU reads the current time for the RTC field registers.

The RTC function of the TRAVEO™T2G device uses the READ bit in the BACKUP_RTC_RW register to copy the current RTC field values from the RTC registers to the RTC field in the user registers in real time. The user firmware freezes the RTC field values in the user registers, preventing them from being updated even when the RTC field values in the RTC registers are updated. Next, the user firmware copies the frozen RTC values to a user buffer, which is then assigned to a local variable in RAM.

Figure 3 describes the use case of reading the current time. The example uses the current time as April 30th 11:59:59 PM (11 hours 59 minutes 59 seconds). First, set the READ bit in the BACKUP_RTC_RW register to '1'. Immediately, the hardware copies the current RTC field data in the RTC register to the RTC field in the user register, then the user firmware clears the READ bit to '0'. After that you can read the user register for the current date and time.

This section explains how to read the RTC value based on a use case using the SDL. The code snippets in this application note are part of SDL. See Other references for the SDL.

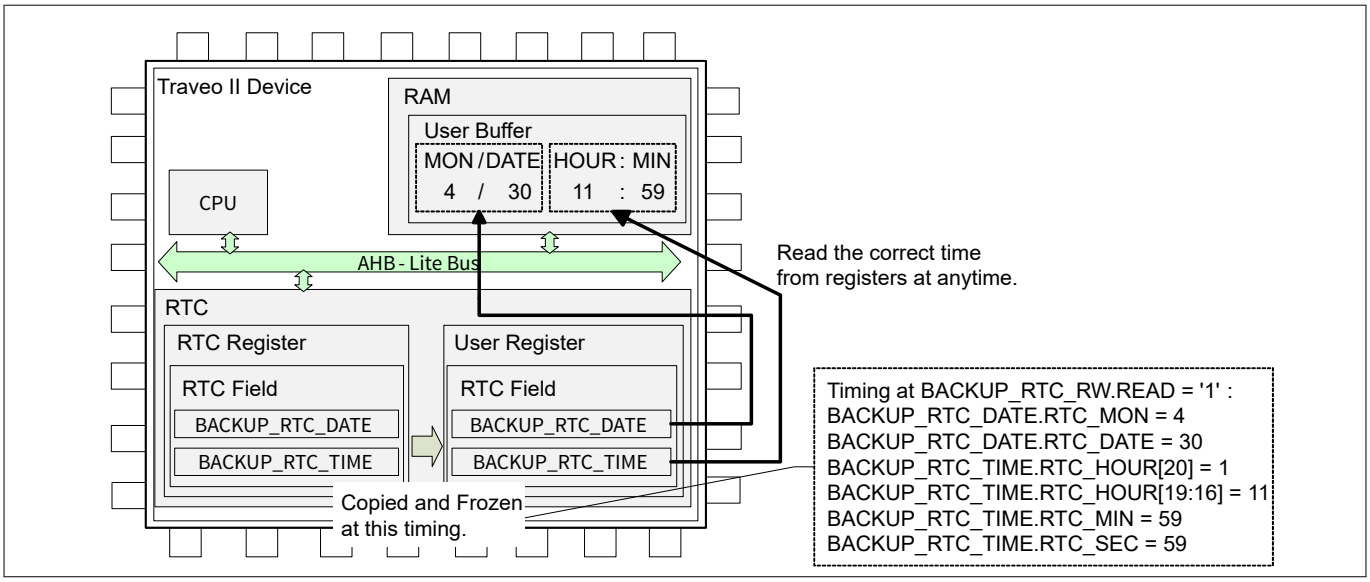

Figure 3 Correct time is read with the READ bit

2.2.1 Use case

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

Use case: Reading RTC values

- Read Data: Read_DateTime

Figure 4 shows an example flow to read the RTC value.

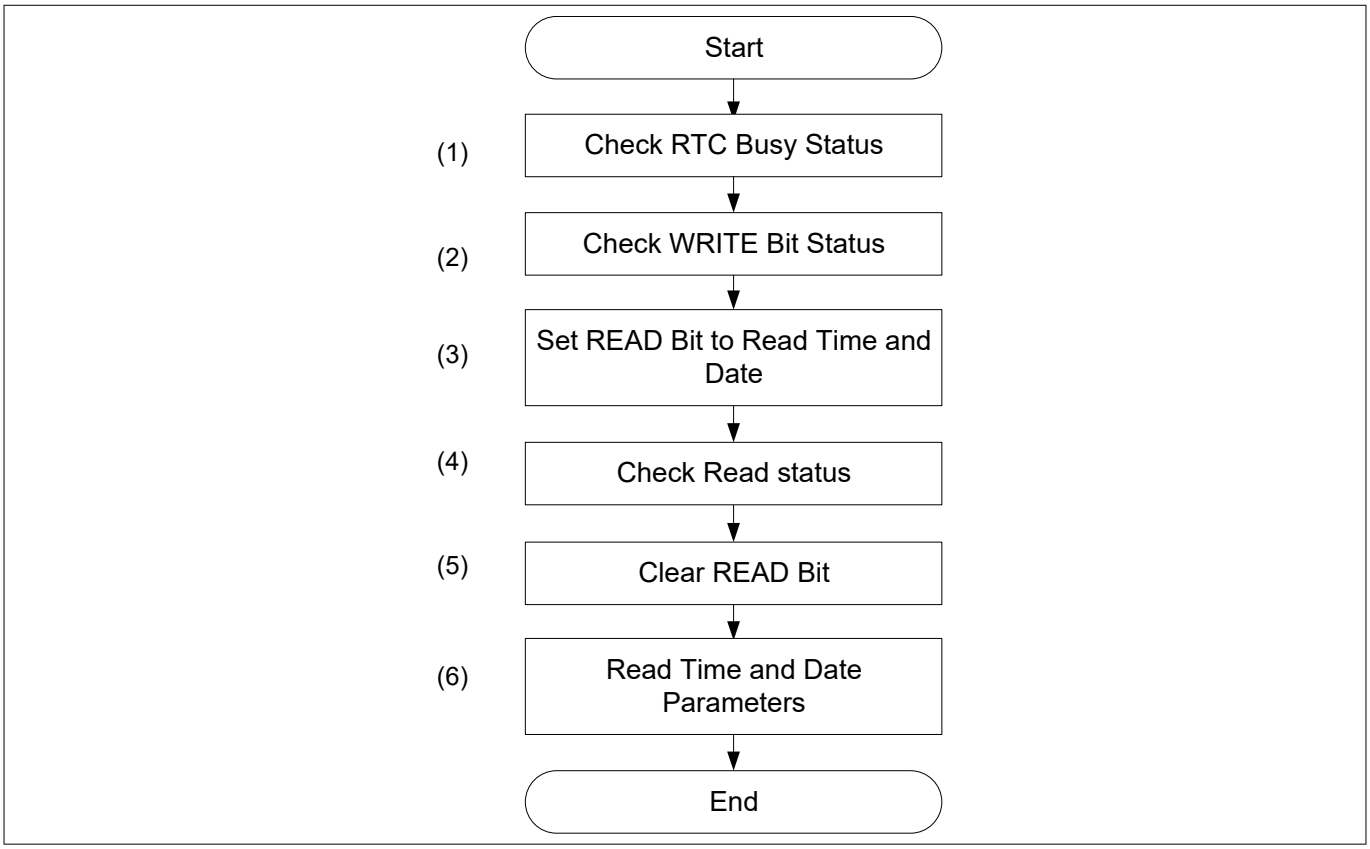

Figure 4 Example flow to read then RTC value

2.2.2 Reading the RTC values

The following example describes the operation of reading RTC values:

1. Check the RTC busy status; wait until BACKUP_STATUS.RTC_BUSY = '0' to ensure safe design[1)]
2. Check the write status; wait until BACKUP_RTC_RW.WRITE = '0' to ensure safe design[2)]
3. Set the READ bit to prepare for the read operation of the RTC values:
   Write BACKUP_RTC_RW.READ = '1' (Begin the Read operation)
   Then, the current RTC field values in the RTC registers are copied to the RTC field in the user registers, and then the hardware freezes the values
4. Check the Read status
5. Clear the READ bit:
   Write BACKUP_RTC_RW.READ = '0' (End the Read operation)
6. Read the time and date parameters from the user register:
   Get time parameter = BACKUP_RTC_TIME
   Get date parameter = BACKUP_RTC_DATE

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
Read_DateTime.sec	Stored second value (Calendar seconds, 0-59)
Read_DateTime.min	Stored minute value (Calendar minutes, 0-59)
Read_DateTime.hour	Stored hour value (Calendar hours, value depending on 12/24HR mode)
Read_DateTime.hrMode	Stored hour HR mode value (12/24HR mode, 0 or 1)
Read_DateTime.dayOfWeek	Stored day value (Calendar Day of the week, 1-7) You can define the values, but it is recommended to set 1=Monday.
Read_DateTime.date	Stored day of the month value (Calendar Day of the Month, 1-31) Automatic Leap Year Correction
Read_DateTime.month	Stored month value (Calendar Month, 1-12)
Read_DateTime.year	Stored year value (Calendar year, 0-99)

Code Listing 12 shows an example program of the reading RTC value. In the example, the program code of the initialization part of the main routine was omitted.

1 The subsequent reading operation cannot be executed until the previous operation is completed.
2 The subsequent reading operation cannot be executed when the WRITE bit is set.

Code Listing 12 Example program to read the RTC value

```
int main(void)
{
    SystemInit();

    /* Configuration structure for reading RTC values */
    cy_stc_rtc_config_t Read_DateTime;

      /* The initialization part is omitted. */
    __enable_irq(); /* Enable global interrupts. */
    .
    .

    for(;;)
    {
    /* Read the time and date. See Code Listing 13. */
        Cy_Rtc_GetDateAndTime(&Read_DateTime);
    }
}
```

Code Listing 13 to Code Listing 15 shows an example program to read RTC value in the driver part.

2 Operation overview

Code Listing 13 Example program to read the RTC time and date values in the driver part

```
void   Cy_Rtc_GetDateAndTime(cy_stc_rtc_config_t* dateTime)
{
    uint32_t tmpTime;
    uint32_t tmpDate;

    /* Read the current RTC time and date to validate the input parameters */
    /* Check the status for reading and setting the READ bit. See Code Listing 14 */
    Cy_Rtc_SyncRegisters();

    /* Write the AHB RTC registers date and time into the local variables and
    * updating the dateTime structure elements
    */
    /* (6) Read the time and date parameters from user register */
    tmpTime = BACKUP->unRTC_TIME.u32Register;
    tmpDate = BACKUP->unRTC_DATE.u32Register;

    dateTime->sec    = (_FLD2VAL(BACKUP_RTC_TIME_RTC_SEC, tmpTime));
    dateTime->min    = (_FLD2VAL(BACKUP_RTC_TIME_RTC_MIN, tmpTime));
    dateTime->hrMode = (_FLD2VAL(BACKUP_RTC_TIME_CTRL_12HR, tmpTime));

    /* Read the current hour mode to know how many hour bits should be converted
    * In the 24-hour mode, the hour value is presented in [21:16] bits in the
    * Integer format.
    * In the 12-hour mode the hour value is presented in [20:16] bits in
    *  the Integer format and bit [21] is present: 0 - AM; 1 - PM.
    */
    if(dateTime->hrMode != CY_RTC_24_HOURS)
    {
        dateTime->hour =
        ((tmpTime & CY_RTC_BACKUP_RTC_TIME_RTC_12HOUR) >> BACKUP_RTC_TIME_RTC_HOUR_Pos);

        dateTime->amPm =
        ((0u != (tmpTime & CY_RTC_BACKUP_RTC_TIME_RTC_PM)) ? CY_RTC_PM : CY_RTC_AM);
    }
    else
    {
        dateTime->hour = (_FLD2VAL(BACKUP_RTC_TIME_RTC_HOUR, tmpTime));
    }
    dateTime->dayOfWeek =(_FLD2VAL(BACKUP_RTC_TIME_RTC_DAY, tmpTime));
    dateTime->date  = (_FLD2VAL(BACKUP_RTC_DATE_RTC_DATE, tmpDate));
    dateTime->month = (_FLD2VAL(BACKUP_RTC_DATE_RTC_MON, tmpDate));
    dateTime->year  = (_FLD2VAL(BACKUP_RTC_DATE_RTC_YEAR, tmpDate));
}
```

Code Listing 14 Example program to sync RTC register in the driver part

```
void Cy_Rtc_SyncRegisters(void)
{
    uint32_t interruptState;

    interruptState = Cy_SysLib_EnterCriticalSection();

    /* RTC Write is possible only in the condition that CY_RTC_BUSY bit = 0
    *  or RTC Write bit is not set.
    */
    /* Check RTC busy status. See Code Listing 15 */    /* (2) Check WRITE bit status */
    if((CY_RTC_BUSY != Cy_Rtc_GetSyncStatus()) && (!_FLD2BOOL(BACKUP_RTC_RW_WRITE, BACKUP->

unRTC_RW.u32Register)))
    {
        /* Setting RTC Read bit */
        /* (3) Set the READ bit for read the time and date */
        BACKUP->unRTC_RW.u32Register = BACKUP_RTC_RW_READ_Msk;

        /* Poll till the read bit is set */
        /* (4) Check the read status. */
        while(BACKUP->unRTC_RW.u32Register !=  BACKUP_RTC_RW_READ_Msk);

        /* Clearing RTC Read bit */
        /* (5) Clear the READ bit */
        BACKUP->unRTC_RW.u32Register = 0u;
    }
    Cy_SysLib_ExitCriticalSection(interruptState);
}
```

Code Listing 15 Example program to get the sync status in the driver part

```
uint32_t Cy_Rtc_GetSyncStatus(void)
{
            /* (1) Check RTC busy status */
    return((_FLD2BOOL(BACKUP_STATUS_RTC_BUSY, BACKUP->unSTATUS.u32Register)) ? CY_RTC_BUSY :

CY_RTC_AVAILABLE);
}
```

2.3 Updating TIME and DATE

The RTC fields can be updated independently. Each RTC field in the user register has an independent update flag. These are internal flags which are not accessed by a user firmware. These flags are set by writing values to the RTC fields in the user register. The writing instruction (BACKUP_RTC_RW.WRITE = '0') writes to only those fields where the update flag is set.

This section explains how to update the RTC value based on a use case using the Sample Driver Library (SDL) provided by Infineon. The code snippets in this application note are part of SDL. See Other references for the SDL.

2.3.1 Use case

This use case shows how to adjust daylight saving time (DST). For adjusting DST, only the RTC_HOUR bit field needs to be updated. The other fields remain unchanged.

Use case:

- Set the RTC initial value
  - 2018 year, June, 19 day, Tuesday, 2 hour (12HR mode), 43 minutes, 0 seconds
- Update the calendar hour in the RTC value from '2' to '3'

Figure 5 shows an example flow to update RTC value.

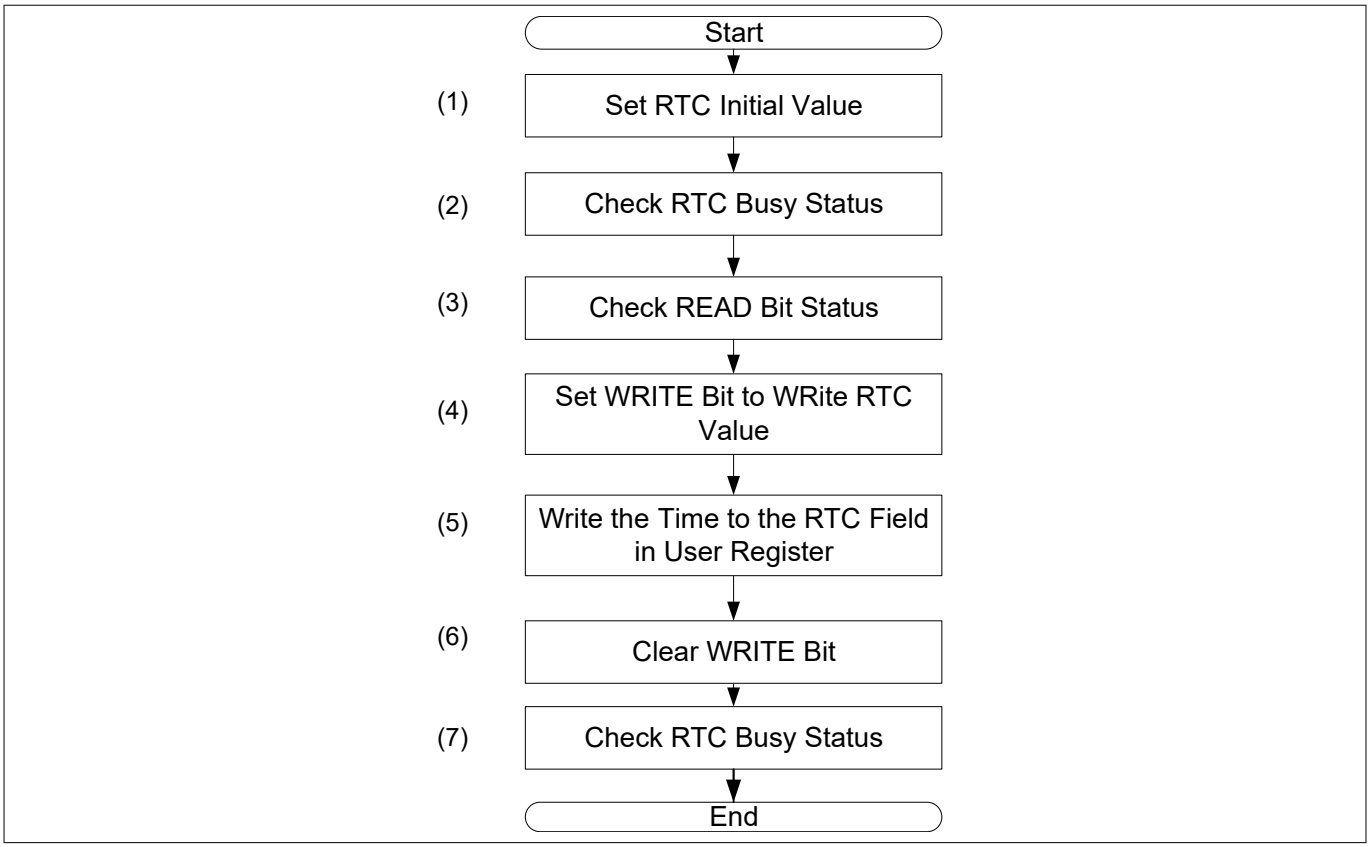

Figure 5 Example flow to update the RTC value

2.3.2 Updating the RTC values (e.g, adjusting DST)

1. Set the RTC initial value. For the initial setting method, see Initializing the RTC function
2. Check the RTC busy status; wait until BACKUP_STATUS.RTC_BUSY = ‘0’
3. Check the read status; if BACKUP_RTC_RW.READ is not ‘0’, clear the READ bit (BACKUP_RTC_RW.READ = ‘0’):
   - Then, wait until BACKUP_RTC_RW.READ = ‘0’
4. Set the WRITE bit to prepare for the write operation of the RTC values:
   - Write BACKUP_RTC_RW.WRITE = ‘1’ (Begin the Write operation)
5. Write the time to the RTC field in the user register:
   - Write BACKUP_RTC_TIME. RTC_HOUR = ‘3’
   - For adjusting DST, ‘3’ is written to the RTC_HOUR bit field instead of ‘2’ to adjust the summer time at 2 AM on the first Sunday of April in the United States

- For adjusting DST, '3' is written to the RTC_HOUR bit field instead of '2' to adjust the summer time at 2 AM on the first Sunday of April in the United States
- The other RTC fields, BACKUP_RTC_TIME.RTC_SEC, RTC_MIN, 12HR, DAY, DATE, MON, and YEAR, are left unchanged

6. Clear the WRITE bit:

- Write BACKUP_RTC_RW.WRITE = '0' (End the Write operation)

7. Check the RTC busy status for RTC register update

Table 3 lists the RTC initial parameters of the configuration part of SDL. After initializing with the values in Table 3, update the RTC value.

Table 4 Initial RTC value parameters

Parameters	Description	Value
RTC_config.sec	Calendar seconds, 0-59	RTC_INITIAL_DATE_SEC = 0ul
RTC_config.min	Calendar minutes, 0-59	RTC_INITIAL_DATE_MIN = 43ul
RTC_config.hour	Calendar hours, value depending on 12/24HR mode	RTC_INITIAL_DATE_HOUR = 2ul
RTC_config.hrMode	Select 12/24HR mode: 1=12HR, 0=24HR CY_RTC_12_HOURS = 1ul, CY_RTC_24_HOURS = 0ul	RTC_INITIAL_DATE_HOUR_FORMAT = CY_RTC_12_HOURS
RTC_config.dayOfWeek	Calendar Day of the week, 1-7 It is up to the user to define the meaning of the values, but 1=Monday is recommended	RTC_INITIAL_DATE_DOW = 2ul
RTC_config.date	Calendar Day of the Month, 1-31 Automatic Leap Year Correction	RTC_INITIAL_DATE_DOM = 19ul
RTC_config.month	Calendar Month, 1-12	RTC_INITIAL_DATE_MONTH = 6ul
RTC_config.year	Calendar year, 0-99	RTC_INITIAL_DATE_YEAR = 18ul

Table 5 lists the parameter and Table 6 lists the functions of the configuration part of SDL with updated RTC values.

Table 5 Updated RTC values

Parameter	Description	Value
RTC_UPDATE_HOUR	Calendar hours, value depending on 12/24HR mode	3ul

Table 6 Updated RTC value functions

Functions	Description	Value
Cy_Rtc_IsReadBitSet (void)	Returns READ bit status. 1: READ bit is set. 0: READ bit is cleared	-
Cy_Rtc_ClearReadBit (void)	Clears READ bit.	-
Cy_Rtc_SetHourBit (uint8_t hour)	Only sets the RTC hour value	RTC_UPDATE_HOUR = 3ul

Code Listing 16 shows an example program of the updated RTC value.

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

- **BACKUP**->un**RTC_TIME** register is the BACKUP_RTC_TIME register mentioned in the register reference manual. Other registers are also described in the same manner

Code Listing 16 Example program to update the RTC value

```
/* RTC Hour value for update */
            /* RTC Hour value for update.  See Table 5 */
#define RTC_UPDATE_HOUR             (3ul) /**< Calendar hours for update */

/* RTC configuration */
cy_stc_rtc_config_t const RTC_config =
{
    /* Initiate time and date */
    /* Configure initial time and date parameters. See Table 4 */
    .sec       = RTC_INITIAL_DATE_SEC,
    .min       = RTC_INITIAL_DATE_MIN,
    .hour      = RTC_INITIAL_DATE_HOUR,
    .hrMode    = RTC_INITIAL_DATE_HOUR_FORMAT,
    .dayOfWeek = RTC_INITIAL_DATE_DOW,
    .date      = RTC_INITIAL_DATE_DOM,
    .month     = RTC_INITIAL_DATE_MONTH,
    .year      = RTC_INITIAL_DATE_YEAR,
};

int main(void)
{
    cy_en_rtc_status_t retVal;
    uint8_t retStatus;
    uint32_t interruptState;

    SystemInit();

    __enable_irq(); /* Enable global interrupts. */

    /* Set the WCO as the clock source to the RTC block */
    retStatus = Cy_SysClk_WcoEnable(100ul);
    if(retStatus == CY_SYSCLK_SUCCESS)
    {
    Cy_Rtc_clock_source(CY_RTC_CLK_SRC_WCO);  /* Set the WCO clock source to RTC. See Code
Listing 29 */
    }

    /* Wait for initial setting */
    while(Cy_Rtc_Init(&RTC_config) != CY_RET_SUCCESS);  /* (1) Set the RTC initial value. The
initialization part is omitted. See Code Listing 4 */



    /* Check for RTC busy status */
    while(Cy_Rtc_GetSyncStatus()==CY_RTC_BUSY);  /* Check RTC busy status. See Code Listing 17
*/

    /* Check for RTC read bit clear */
    /* Check READ bit status. See Code Listing 18 and Code Listing 19. */
    if(Cy_Rtc_IsReadBitSet()==1)
    {
```

```
            Cy_Rtc_ClearReadBit();
        }
        /* Check for read bit status */
        while(Cy_Rtc_IsReadBitSet()==1);



        /* Set to RTC write bit */
        /* Set the WRITE bit to write the RTC value. See Code Listing 7 */
        interruptState = Cy_SysLib_EnterCriticalSection();
        retVal = Cy_Rtc_WriteEnable(CY_RTC_WRITE_ENABLED);
        if(retVal == CY_RTC_SUCCESS)
        {
            /* Set hour value for updating */
            Cy_Rtc_SetHourBit(RTC_UPDATE_HOUR);  /* Write the time to the RTC field in the user
register. See Code Listing 20 */

            /* Clear the RTC Write bit */
            Cy_Rtc_WriteEnable(CY_RTC_WRITE_DISABLED);  /* Clear the WRITE bit. See Code Listing 7
*/
        }
        Cy_SysLib_ExitCriticalSection(interruptState);

        for(;;)
        {

        }
    }
```

2.3.3 Example program to update RTC values in the driver part

Code Listing 17 to Code Listing 20 shows an example program to update the RTC values in the driver part.

Code Listing 17 Example program to get the sync status in the driver part

```
uint32_t Cy_Rtc_GetSyncStatus(void)
{
    /* (2) Check RTC busy status */
    return((_FLD2BOOL(BACKUP_STATUS_RTC_BUSY, BACKUP->unSTATUS.u32Register)) ? CY_RTC_BUSY :

CY_RTC_AVAILABLE);
}
```

Code Listing 18 Example program to check the READ bit status in the driver part

```
bool Cy_Rtc_IsReadBitSet(void)
{
    if(BACKUP->unRTC_RW.stcField.u1READ == 0ul)  /* (3) Check READ bit status. */
    {
        return false;
    }
    else
    {
        return true;
    }
}
```

Code Listing 19 Example program to clear the READ bit in the driver part

```
cy_en_rtc_status_t Cy_Rtc_ClearReadBit(void)
{
BACKUP->unRTC_RW.stcField.u1READ = 0ul;  /* This function clears the READ bit to 0 */

    return(CY_RTC_SUCCESS);
}
```

Code Listing 20 Example program to set the RTC hour value in the driver part

```
cy_en_rtc_status_t Cy_Rtc_SetHourBit(uint8_t hour)
{
BACKUP->unRTC_TIME.stcField.u5RTC_HOUR = hour;   /* (5) Write the time to the RTC field in the
user register. */
    return CY_RTC_SUCCESS;
}
```

2.4 Calibrating the WCO

***Note**: Calibration of WCO is possible only on the device having the RTC_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, the user has the capability to calibrate the WCO and improve the accuracy of the real-time clock.

This section explains how to calibrate the WCO based on a use case using the Sample Driver Library (SDL) provided by Infineon. The code snippets in this application note are part of SDL. See Other references for the SDL.

2.4.1 Use case

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

Use case:

- Output the 512 Hz calibration waveform to the RTC_CAL pin

- Set the calibration value in the register based on 27.125 ppm slow down calibration waveform
- Confirm 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 re-measuring

2.4.2 Capturing the WCO waveform

To capture the WCO waveform in an oscilloscope, route it to the RTC_CAL pin of the device by setting the CAL_OUT bit. This action outputs a 512 Hz clock waveform, derived from the WCO, on the RTC_CAL pin. Next, measure the deviation of this output from an ideal 512 Hz clock and convert the deviation to a ppm value. Then, calculate the calibration settings needed to correct the WCO frequency error. Before proceeding with this operation, ensure that the RTC function is running normally with the configuration specified in Table 3.

Figure 6 shows an example flow to calibrate WCO waveform output.

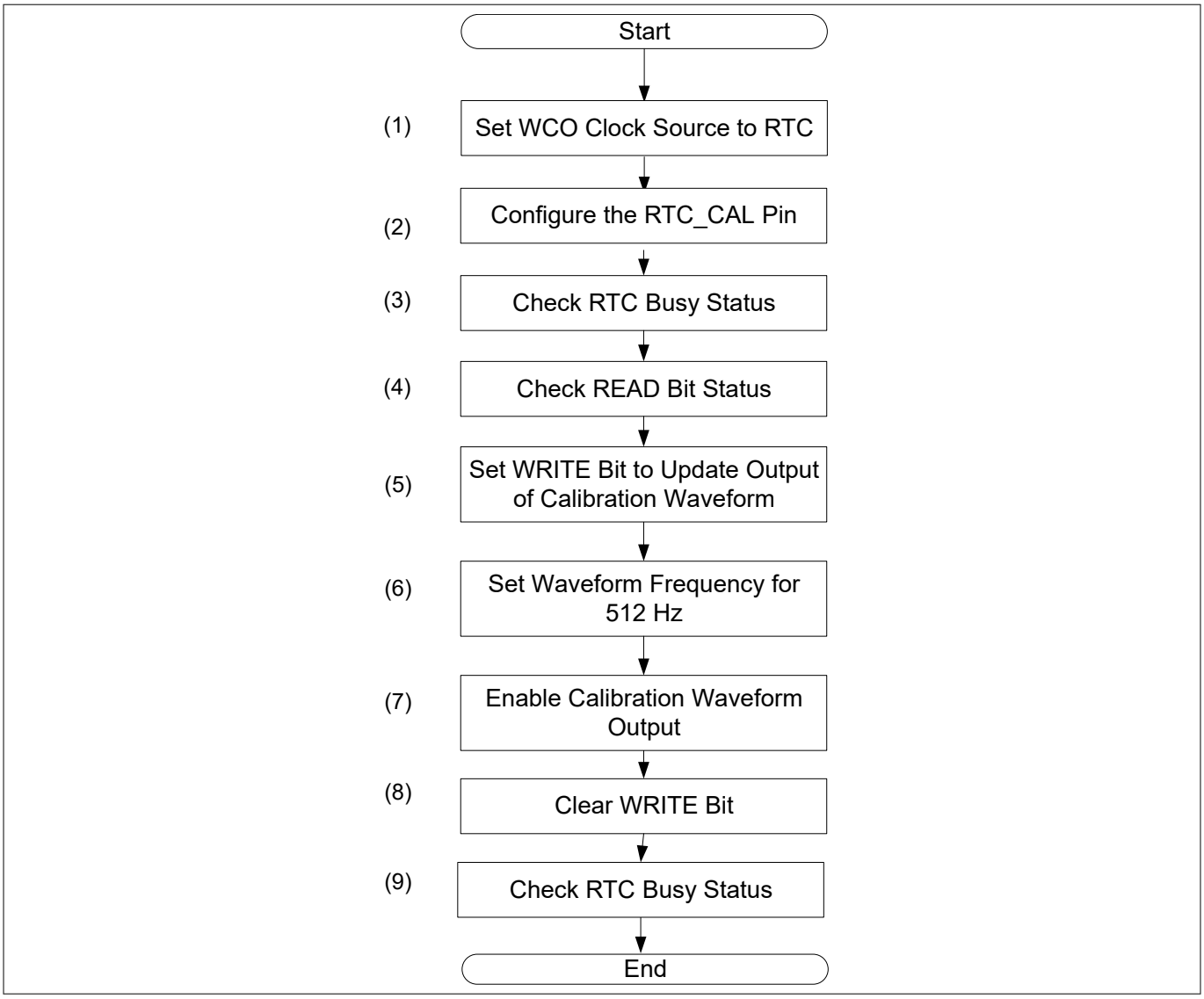

Figure 6 Example flow to calibrate the WCO waveform output

The following steps show how to output the WCO waveform:

1. Set the WCO clock source to the RTC
2. Configure the RTC_CAL pin; see the I/O System Chapter in the architecture reference manual
3. Check the RTC busy status; wait until BACKUP_STATUS.RTC_BUSY = '0' to ensure safe design[3)]
4. Check the read status; if BACKUP_RTC_RW.READ is not '0', clear the READ bit (BACKUP_RTC_RW.READ = '0'). Then, wait until BACKUP_RTC_RW.READ = '0' to ensure safe design[4)]
5. Set the WRITE bit to update the output of the calibration waveform: Write BACKUP_RTC_RW.WRITE = '1' (Begin the Write operation)
6. Set waveform frequency for 512 Hz: Write BACKUP_CAL_CTL.CAL_SEL = '0'
7. 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: Write BACKUP_RTC_RW.WRITE = '0' (End the Write operation)
9. Check the RTC busy status for RTC register update

Table 7 lists the parameters of the configuration part in SDL to RTC_CAL pin output in GPIO.

For GPIO setting, see the I/O System Chapter in the architecture reference manual.

Table 7 RTC_CAL pin outputs in GPIO

Parameters	Description	Value
user_led_port_pin_cfg.outVal	Selects pin output state. 0: Output state not affected, 1: Output state set to '0'	0ul
user_led_port_pin_cfg.driveMode	Selects GPIO drive mode for IO pin. 0: Analog High Impedance 1: Reserved and should not be used 2: Resistive pull-up 3: Resistive pull-down 4: Open drain, drives LOW 5: Open drain, drives HIGH 6: Strong drive 7: Resistive pull-up/down 8: Digital High-Z. Input buffer ON. 9: Resistive Pull-Up. Input buffer ON. 10: Resistive Pull-Down. Input buffer ON. 11: Open Drain, Drives LOW. Input buffer ON. 12: Open Drain, Drives HIGH. Input buffer ON. 13: Strong Drive. Input buffer ON. 14: Resistive Pull-Up/Down. Input buffer ON.	CY_GPIO_DM_STRONG_IN_OFF =6ul

(table continues...)

3 The subsequent calibration operation cannot be executed while executing the previous operation.
4 The subsequent writing operation cannot be executed while the READ bit is set.

Table 7 (continued) RTC_CAL pin outputs in GPIO

Parameters	Description	Value
user_led_port_pin_cfg.hsiom	Sets connection for RTC_CAL pin route.	P1_2_SRSS_CAL_WAVE
user_led_port_pin_cfg.intEdge	Sets the edge which will trigger an IRQ for IO pin. 0: Disabled, 1: Rising edge, 2: Falling edge, 3: Both	0ul
user_led_port_pin_cfg.intMask	Masks edge interrupt on IO pin. 0: Pin interrupt forwarding disabled 1: Pin interrupt forwarding enabled	0ul
user_led_port_pin_cfg.vtrip	Selects the IO pin input buffer mode. 0: CMOS, 1: TTL	0ul
user_led_port_pin_cfg.slewRate	Selects slew rate for IO pin. 0: Fast slew rate, 1: Slow slew rate	0ul
user_led_port_pin_cfg.driveSel	Sets the GPIO drive strength for IO pin. 0: Full drive strength 1: Full drive strength 2: 1/2 drive strength 3: 1/4 drive strength	0ul

Table 8 lists the parameters and Table 9 lists the function of the configuration part in SDL to RTC calibration waveform output.

Table 8 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

(table continues...)

Table 8 (continued) RTC calibration waveform output parameters

Parameters	Description	Value
CY_EN_RTC_CAL_SEL_CAL512	Select 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 9 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)	Set calibration control register and enable to calibration waveform output calib_val: Calibration value for absolute frequency. calib_sign: Calibration sign cal_sel: Select calibration wave output signal	CALIB_VALUE = 0ul, CY_EN_RTC_CALIB_SIGN_NEGATIVE=0, CY_EN_RTC_CAL_SEL_CAL512 = 0

Code Listing 18 shows an example program of the RTC calibration waveform output.

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

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

The RTC_CAL pin configuration shows the case of the CYT4DN series. These RTC_CAL pin settings are the same, for example Code Listing 21, Code Listing 28, and Code Listing 34.

Code Listing 21 Example program of the RTC calibration waveform output

```
/* Configure the RTC_CAL pin definition (P1_2)pin */
/* Definition for RTC_CAL */
#define RTC_CAL_PIN                    P1_2_SRSS_CAL_WAVE
#define USER_PORT                      GPIO_PRT1
#define USER_PIN                       2

/* Setting value for calibration definition */
/* Definition for calibration value */
#define CALIB_VALUE                    0ul

/* RTC configuration */
cy_stc_rtc_config_t const RTC_config =
{
    /* Initiate time and date */
    /* Configure initial time and date parameters. See Table 4. */
    .sec       = RTC_INITIAL_DATE_SEC,
    .min       = RTC_INITIAL_DATE_MIN,
    .hour      = RTC_INITIAL_DATE_HOUR,
    .hrMode    = RTC_INITIAL_DATE_HOUR_FORMAT,
    .dayOfWeek = RTC_INITIAL_DATE_DOW,
    .date      = RTC_INITIAL_DATE_DOM,
    .month     = RTC_INITIAL_DATE_MONTH,
    .year      = RTC_INITIAL_DATE_YEAR,
};

/* RTC_CAL pin configuration */
cy_stc_gpio_pin_config_t user_led_port_pin_cfg =   /* Configuration for RTC_CAL in the GPIO.
See Table 7 */
{
    .outVal    = 0ul,
    .driveMode = CY_GPIO_DM_STRONG_IN_OFF,
    .hsiom     = RTC_CAL_PIN,
    .intEdge   = 0ul,
    .intMask   = 0ul,
    .vtrip     = 0ul,
    .slewRate  = 0ul,
    .driveSel  = 0ul,

};

int main(void)
{
    cy_en_rtc_status_t retVal;
    uint8_t retStatus;
    uint32_t interruptState;

    SystemInit();

    __enable_irq(); /* Enable global interrupts. */

    /* Set the WCO as the clock source to the RTC block */
```

```
    retStatus = Cy_SysClk_WcoEnable(100ul);
    if(retStatus == CY_SYSCLK_SUCCESS)
    {
    Cy_Rtc_clock_source(CY_RTC_CLK_SRC_WCO);   /* Set the WCO clock source to the RTC. See Code
Listing 22 */

    }

    /* Configure the RTC_CAL pin */
    Cy_GPIO_Pin_Init(USER_PORT, USER_PIN, &user_led_port_pin_cfg);  /* (2) Configure the
RTC_CAL pin */

    /* Wait for initial setting */
    while(Cy_Rtc_Init(&RTC_config) != CY_RET_SUCCESS);  /* Set the RTC initial value. The
initialization part is omitted. See Code Listing 4. */

    /* Check for RTC busy status */
    while(Cy_Rtc_GetSyncStatus()==CY_RTC_BUSY);  /* Check RTC busy status. See Code Listing 23.
*/


    /* Check for RTC read bit clear */
    /* Check READ bit status. See Code Listing 24, Code Listing 25. */

    if(Cy_Rtc_IsReadBitSet()==1)
    {
         Cy_Rtc_ClearReadBit();
    }

    /* Check for read bit status */
    while(Cy_Rtc_IsReadBitSet()==1);

    /* Set to RTC write bit */
    interruptState = Cy_SysLib_EnterCriticalSection();
    retVal = Cy_Rtc_WriteEnable(CY_RTC_WRITE_ENABLED);   /* Set the WRITE bit for write the
CAL_OUT field.
    See Code Listing 26. */
    if(retVal == CY_RTC_SUCCESS)
    {
        /* Calibration waveform output enable */
        /* Set waveform frequency and enable the calibration waveform output.  See Code Listing
27. */
        Cy_Rtc_CalibrationControlEnable(CALIB_VALUE, CY_EN_RTC_CALIB_SIGN_NEGATIVE,
CY_EN_RTC_CAL_SEL_CAL512);
        /* Clear the RTC Write bit */
        Cy_Rtc_WriteEnable(CY_RTC_WRITE_DISABLED);   /* Clear the WRITE bit. See Code Listing
26. */
    }
    Cy_SysLib_ExitCriticalSection(interruptState);

    for(;;)
    {
```

```
    }
}
```

2.4.3 Example program to capture the WCO waveform in the driver part

Code Listing 22 to Code Listing 27 shows an example program to capture the WCO waveform in the driver part.

Code Listing 22 Example program to the RTC input clock source set in the driver part

```
void Cy_Rtc_clock_source(cy_en_rtc_clock_src_t clock_source)
{
BACKUP->unCTL.stcField.u2CLK_SEL = clock_source;   /* (1) Set the WCO clock source to the RTC.
*/
}
```

Code Listing 23 Example program to get sync status in the driver part

```
uint32_t Cy_Rtc_GetSyncStatus(void)
{
    /* (3) Check RTC busy status. */
    return((_FLD2BOOL(BACKUP_STATUS_RTC_BUSY, BACKUP->unSTATUS.u32Register)) ? CY_RTC_BUSY :

CY_RTC_AVAILABLE);
}
```

Code Listing 24 Example program to check the READ bit status in the driver part

```
bool Cy_Rtc_IsReadBitSet(void)
{
    if(BACKUP->unRTC_RW.stcField.u1READ == 0ul)   /* (4) Check READ bit status. */
    {
        return false;
    }
    else
    {
        return true;
    }
}
```

Code Listing 25 Example program to clear the READ bit in the driver part

```
cy_en_rtc_status_t Cy_Rtc_ClearReadBit(void)
{
    BACKUP->unRTC_RW.stcField.u1READ = 0ul;  /* This function clears the READ bit to 0. */

    return(CY_RTC_SUCCESS);
}
```

Code Listing 26 Example program to write enable in the driver part

```
cy_en_rtc_status_t Cy_Rtc_WriteEnable(uint32_t writeEnable)
{
    cy_en_rtc_status_t retVal = CY_RTC_INVALID_STATE;

    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
        */
        if((CY_RTC_BUSY != Cy_Rtc_GetSyncStatus()) && (!_FLD2BOOL(BACKUP_RTC_RW_READ, BACKUP->
            unRTC_RW.u32Register)))
        {
        BACKUP->unRTC_RW.u32Register |= BACKUP_RTC_RW_WRITE_Msk;  /* (5) Set the WRITE bit for
write the CAL_OUT field. */
            retVal = CY_RTC_SUCCESS;
        }
    }
    else
    {
        /* Clearing Write Bit to complete write procedure */
        BACKUP->unRTC_RW.u32Register &= ((uint32_t) ~BACKUP_RTC_RW_WRITE_Msk);  /* (8) Clear
the WRITE bit. */

        /* wait until CY_RTC_BUSY bit is cleared */
        while(CY_RTC_BUSY == Cy_Rtc_GetSyncStatus());  /*  (9) Check the RTC busy status. */

        retVal = CY_RTC_SUCCESS;
    }

    return(retVal);
}
```

Code Listing 27 Example program to set 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_cal_sel_t cal_sel)
{
    if(BACKUP->unRTC_RW.stcField.u1WRITE == 0)
    {
        // Writes are ignored unless Write bit is set
        return CY_RTC_INVALID_STATE;
    }

    un_BACKUP_CAL_CTL_t tmpBackupCalCtl = {0ul};
    tmpBackupCalCtl.stcField.u6CALIB_VAL  = calib_val;
    tmpBackupCalCtl.stcField.u1CALIB_SIGN = (uint32_t)calib_sign;
    tmpBackupCalCtl.stcField.u2CAL_SEL    = (uint32_t)cal_sel;   /* (6) Set waveform frequency
for 512Hz. */

    /* (7) Enable the calibration waveform output. */
    tmpBackupCalCtl.stcField.u1CAL_OUT    = 1ul; // Output enable for wave signal for
calibration and
    allow CALIB_VAL to be written.
    BACKUP->unCAL_CTL.u32Register = tmpBackupCalCtl.u32Register;

    return CY_RTC_SUCCESS;
}
```

2.4.4 Measuring the difference between the WCO and an ideal waveform

Output the WCO waveform (512 Hz) onto the external pin of RTC_CAL. Then, measure the difference between the calibration waveform and the ideal waveform, such as an atomic clock or calibrated lab equipment, using an external calibrated lab equipment. This equipment should be separate from the TRAVEO™ T2G device. Calculate the ppm value based on the difference between the two clock signals. The process of measuring the calibration waveform is demonstrated in Figure 7.

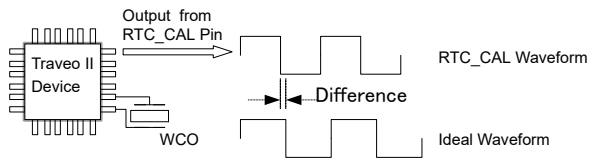

Figure 7 Measuring the difference between two waveforms

Add 2 × 64 = 128 clock ticks hourly by setting a positive calibration value of '1', reducing the number of WCO clock ticks needed to count that hour. This means that a calibration value of '1' represents a correction of (2 × 64)/(32,768 × 60 × 60) = 1.085 ppm. We can hold values up to 63 in the 6-bit calibration value field, but because we reload the calibration counter every hour, we can only effectively use values up to 60. This gives us a calibration range of ± 60 × 1.085 ppm = ± 65.1 ppm.

The CALIB_SIGN bit determines the direction of correction. Use a value of '0' to apply negative calibration, which removes pulses and slows down real-time tracking by the RTC function. Conversely, we use a value of '1' to apply positive calibration.

Figure 8 shows an example flow to calibrate WCO.

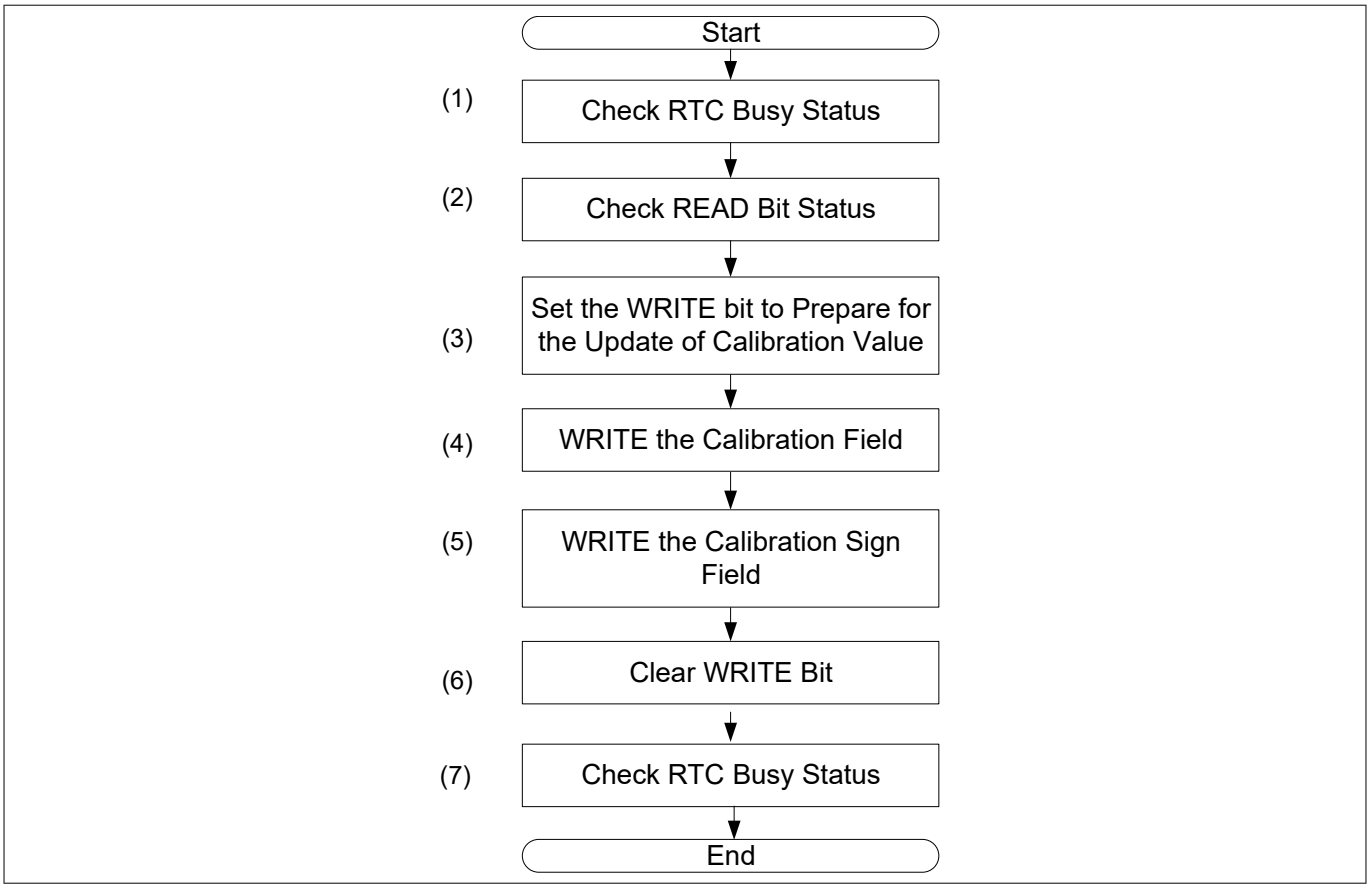

Figure 8 Example flow to calibrate the WCO

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. Check the RTC busy status; wait until BACKUP_STATUS.RTC_BUSY = '0' to ensure safe design[5)]
2. Check the read status; if BACKUP_RTC_RW.READ is not '0', clear the READ bit (BACKUP_RTC_RW.READ = '0'):
   - Then, wait until BACKUP_RTC_RW.READ = '0' to ensure safe design[6)]
3. Set the WRITE bit to prepare for the update of the calibration value:
   - Write BACKUP_RTC_RW.WRITE = '1' (Begin the Write operation)
4. 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. Write the calibration sign field:
   - Write BACKUP_CAL_CTL.CALIB_SIGN = '1'
   - Use a value of '1' because the RTC clock needs to be sped up in this example

5 The subsequent calibration operation cannot be executed while executing the previous operation.
6 The subsequent writing operation cannot be executed while the READ bit is set.

6. Clear the WRITE bit:
   - Write BACKUP_RTC_RW.WRITE = '0' (End the Write operation)
7. Check the RTC busy status for RTC register update

The calibration correction is done by either adding (like in 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 10 lists the parameters and Table 11 lists function of the configuration part in SDL to set the calibration value of RTC.

Table 10 RTC calibration value parameters

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
CY_EN_RTC_CAL_SEL_CAL512	Select 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 11 RTC calibration value 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)	Set calibration control register and enable to calibration wave form output calib_val: Calibration value for absolute frequency. calib_sign: Calibration sign cal_sel: Select calibration wave output signal	CALIB_VALUE = 25ul, CY_EN_RTC_CALIB_SIGN_POSITIVE= 0, CY_EN_RTC_CAL_SEL_CAL512 = 0

Code Listing 28 shows an example program to set the calibration value of RTC.

Code Listing 28 Example program to set the calibration value

```
/* Configure the RTC_CAL pin definition (P1_2)pin */
/* Definition for RTC_CAL. */
#define RTC_CAL_PIN                    P1_2_SRSS_CAL_WAVE
#define USER_PORT                      GPIO_PRT1
#define USER_PIN                       2

/* Setting value for calibration definition */
#define CALIB_VALUE                    25ul   /* Definition for calibration value */

/* RTC configuration */
cy_stc_rtc_config_t const RTC_config =
{
    /* Initiate time and date */
    /* Configure initial time and date parameters.See Table 4. */
    .sec       = RTC_INITIAL_DATE_SEC,
    .min       = RTC_INITIAL_DATE_MIN,
    .hour      = RTC_INITIAL_DATE_HOUR,
    .hrMode    = RTC_INITIAL_DATE_HOUR_FORMAT,
    .dayOfWeek = RTC_INITIAL_DATE_DOW,
    .date      = RTC_INITIAL_DATE_DOM,
    .month     = RTC_INITIAL_DATE_MONTH,
    .year      = RTC_INITIAL_DATE_YEAR,
};

/* RTC_CAL pin configuration */
cy_stc_gpio_pin_config_t user_led_port_pin_cfg =   /* Configuration for RTC_CAL in the GPIO.
See Table 7. */
{
    .outVal    = 0ul,
    .driveMode = CY_GPIO_DM_STRONG_IN_OFF,
    .hsiom     = RTC_CAL_PIN,
    .intEdge   = 0ul,
    .intMask   = 0ul,
    .vtrip     = 0ul,
    .slewRate  = 0ul,
    .driveSel  = 0ul,

};


int main(void)
{
    cy_en_rtc_status_t retVal;
    uint8_t retStatus;
    uint32_t interruptState;

    SystemInit();

    __enable_irq(); /* Enable global interrupts. */

    /* Set the WCO as the clock source to the RTC block */
```

```
    retStatus = Cy_SysClk_WcoEnable(100ul);
    if(retStatus == CY_SYSCLK_SUCCESS)
    {
    Cy_Rtc_clock_source(CY_RTC_CLK_SRC_WCO);  /* Set the WCO clock source to the See Code
Listing 29.

    }

    /* Configure the RTC_CAL pin */
    Cy_GPIO_Pin_Init(USER_PORT, USER_PIN, &user_led_port_pin_cfg);

    /* Wait for initial setting */
    while(Cy_Rtc_Init(&RTC_config) != CY_RET_SUCCESS); /* Set the RTC initial value. The
initialization part is omitted. See Code Listing 4.*/


    /* Check for RTC busy status */
    while(Cy_Rtc_GetSyncStatus()==CY_RTC_BUSY);  /* Check RTC busy status. See Code Listing 30.
*/



    /* Check for RTC read bit clear */
    /* Check READ bit status. See Code Listing 31, Code Listing 32 */.
    if(Cy_Rtc_IsReadBitSet()==1)
    {
         Cy_Rtc_ClearReadBit();
    }

    /* Check for read bit status */
    while(Cy_Rtc_IsReadBitSet()==1);


    /* Set to RTC write bit */
    interruptState = Cy_SysLib_EnterCriticalSection();
    retVal = Cy_Rtc_WriteEnable(CY_RTC_WRITE_ENABLED);  /* Set the WRITE bit to prepare for the
update of the calibration value. See Code Listing 33. */
    if(retVal == CY_RTC_SUCCESS)
    {

        /* Set calibartion value and sign */
        /* WRITE the calibration value and calibration sign filed. See Code Listing 34. */
        Cy_Rtc_CalibrationControlEnable(CALIB_VALUE, CY_EN_RTC_CALIB_SIGN_POSITIVE,
CY_EN_RTC_CAL_SEL_CAL512);
        /* Clear the RTC Write bit */
        Cy_Rtc_WriteEnable(CY_RTC_WRITE_DISABLED);  /* Clear the WRITE bit. See Code Listing
33. */
    }
    Cy_SysLib_ExitCriticalSection(interruptState);


    for(;;)
    {
```

```
    }
}
```

2.4.5 Example program to measure the WCO waveform in the driver part

Code Listing 29 to Code Listing 34 shows an example program to measure the WCO waveform in the driver part.

Code Listing 29 Example program measure the RTC input clock source set in the driver part

```
void Cy_Rtc_clock_source(cy_en_rtc_clock_src_t clock_source)
{
BACKUP->unCTL.stcField.u2CLK_SEL = clock_source;  /* Set the WCO clock source to the RTC. */
}
```

Code Listing 30 Example program to get the sync status in the driver part

```
uint32_t Cy_Rtc_GetSyncStatus(void)
{
      /* (1) Check RTC busy status. */
    return((_FLD2BOOL(BACKUP_STATUS_RTC_BUSY, BACKUP->unSTATUS.u32Register)) ? CY_RTC_BUSY :

CY_RTC_AVAILABLE);
}
```

Code Listing 31 Example program to check the READ bit status in the driver part

```
bool Cy_Rtc_IsReadBitSet(void)
{
    if(BACKUP->unRTC_RW.stcField.u1READ == 0ul)   /* (2) Check READ bit status */
    {
        return false;
    }
    else
    {
        return true;
    }
}
```

Code Listing 32 Example program to clear the READ bit in the driver part

```
cy_en_rtc_status_t Cy_Rtc_ClearReadBit(void)
{
    BACKUP->unRTC_RW.stcField.u1READ = 0ul;  /* This function clears the READ bit to 0. */

    return(CY_RTC_SUCCESS);
}
```

Code Listing 33 Example program to write enable in the driver part

```
cy_en_rtc_status_t Cy_Rtc_WriteEnable(uint32_t writeEnable)
{
    cy_en_rtc_status_t retVal = CY_RTC_INVALID_STATE;

    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
        */
        if((CY_RTC_BUSY != Cy_Rtc_GetSyncStatus()) && (!_FLD2BOOL(BACKUP_RTC_RW_READ, BACKUP->
            unRTC_RW.u32Register)))
        {
        BACKUP->unRTC_RW.u32Register |= BACKUP_RTC_RW_WRITE_Msk;  /* (3) Set the WRITE bit to
prepare for the update of the calibration value.  */
            retVal = CY_RTC_SUCCESS;
        }
    }
    else
    {
        /* Clearing Write Bit to complete write procedure */
        BACKUP->unRTC_RW.u32Register &= ((uint32_t) ~BACKUP_RTC_RW_WRITE_Msk);  /*(6) Clear the
WRITE bit. */

        /* wait until CY_RTC_BUSY bit is cleared */
        while(CY_RTC_BUSY == Cy_Rtc_GetSyncStatus());  /* (7) Check the RTC busy status. */

        retVal = CY_RTC_SUCCESS;
    }

    return(retVal);
}
```

Code Listing 34 Example program to set 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_cal_sel_t cal_sel)
{
    if(BACKUP->unRTC_RW.stcField.u1WRITE == 0)
    {
        // Writes are ignored unless Write bit is set
        return CY_RTC_INVALID_STATE;
    }

    un_BACKUP_CAL_CTL_t tmpBackupCalCtl = {0ul};
    tmpBackupCalCtl.stcField.u6CALIB_VAL  = calib_val;  /* (4) WRITE the calibration value
field */
    tmpBackupCalCtl.stcField.u1CALIB_SIGN = (uint32_t)calib_sign;  */ (5) WRITE the calibration
sign filed  */
    tmpBackupCalCtl.stcField.u2CAL_SEL    = (uint32_t)cal_sel;
    tmpBackupCalCtl.stcField.u1CAL_OUT    = 1ul; // Output enable for wave signal for
calibration and
    allow CALIB_VAL to be written.
    BACKUP->unCAL_CTL.u32Register = tmpBackupCalCtl.u32Register;

    return CY_RTC_SUCCESS;
}
```

2.4.6 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 shown in Figure 9, the 512 Hz divider is not routed through the calibration logic. Therefore, you can verify the accuracy of the WCO frequency by measuring the 1 Hz or the 2 Hz waveform.

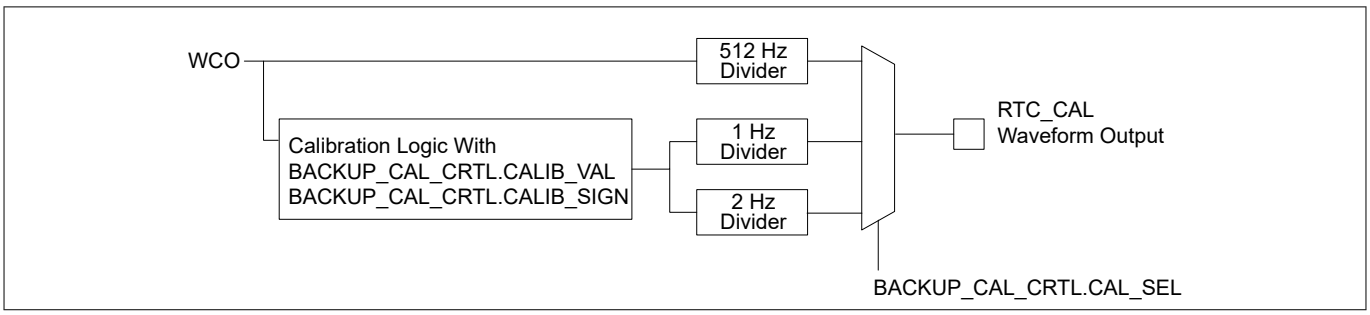

Figure 9 Clock select block diagram

Figure 10 shows an example flow to confirm the accuracy of the WCO frequency after calibration.

2 Operation overview

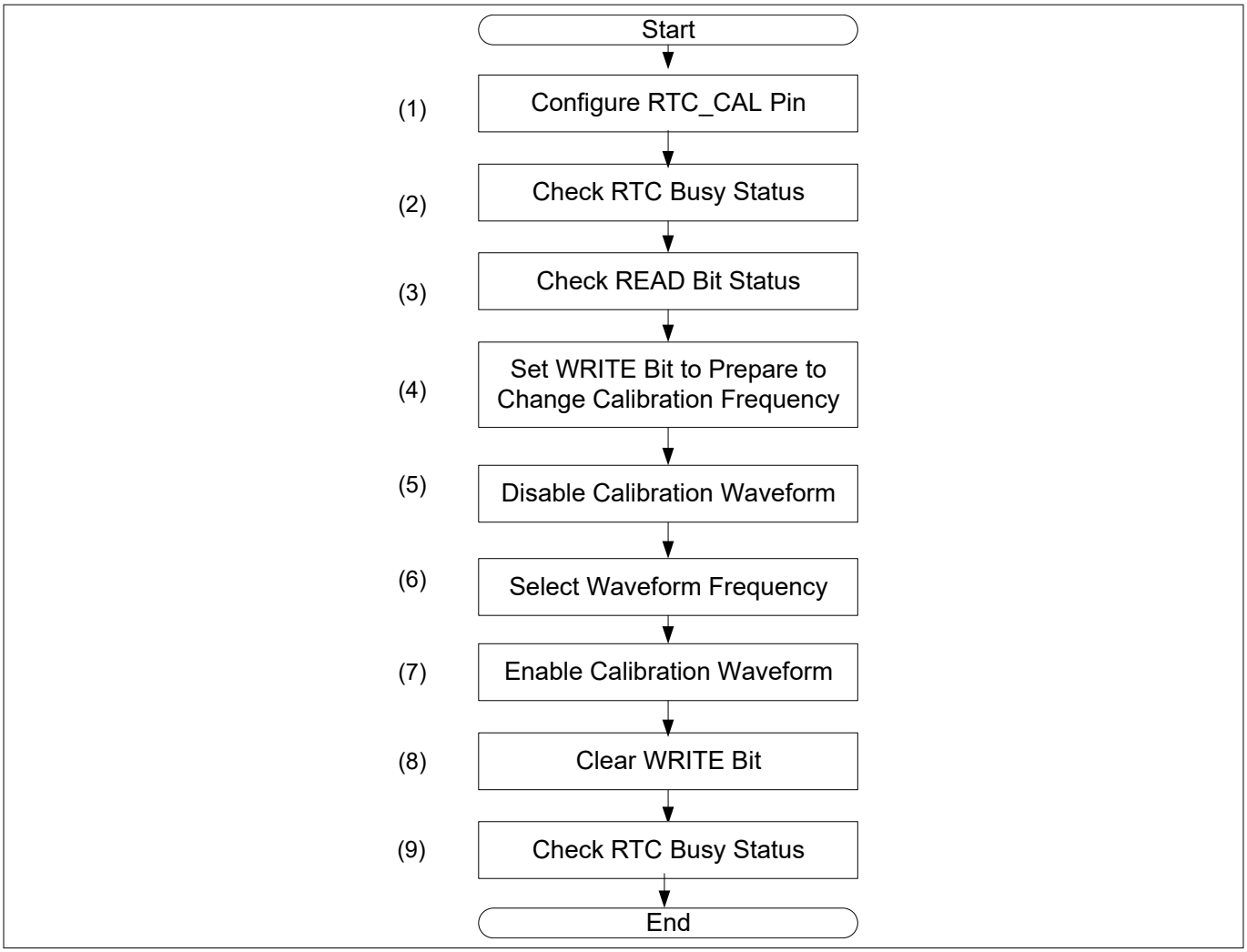

Figure 10 Example flow to confirm the accuracy of WCO frequency after calibration

The following steps explain how to output the WCO frequency of a 1 Hz waveform to confirm the accuracy post-calibration:

1. Configure the RTC_CAL pin; see the I/O Port chapter in the architecture reference manual
2. Check the RTC busy status; wait until BACKUP_STATUS.RTC_BUSY = '0' to ensure safe design[7)]
3. Check the read status; If BACKUP_RTC_RW.READ is not '0', clear the READ bit (BACKUP_RTC_RW.READ = '0'):
   - Then wait until BACKUP_RTC_RW.READ = '0' to ensure safe design[8)]
4. Set the WRITE bit to prepare to change the calibration frequency:
   - Write BACKUP_RTC_RW.WRITE = '1' (Begin the Write operation)
5. Disable the calibration waveform:
   - Write BACKUP_CAL_CTL.CAL_OUT = '0' (Disable calibration waveform output)
6. Select the waveform frequency:
   - Write BACKUP_CAL_CTL.CAL_SEL = '3' (1 Hz)
7. Enable the calibration waveform:
   - Write BACKUP_CAL_CTL.CAL_OUT = '1' (Enable Calibration waveform output)

7 The subsequent calibration operation cannot be executed while executing the previous operation.
8 The subsequent writing operation cannot be executed while the READ bit is set.

8. Clear the WRITE bit:
   - Write BACKUP_RTC_RW.WRITE = '0' (End the Write operation)
9. Check the RTC busy status for RTC register update

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 an external equipment (for example, a calibrated oscilloscope), you can check the effectiveness of the calibration.

Table 12 lists the parameters and Table 13 lists the function of the configuration part in SDL to change the output frequency of the calibration waveform.

Table 12 RTC to change the output frequency of the calibration waveform

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: Add pulses (it takes less clock ticks to count one second)	1
CY_EN_RTC_CAL_SEL_CAL1	Select 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 13 RTC to change output frequency of calibration waveform functions

Functions	Description	Value
Cy_Rtc_CalibrationControlDisable (void)	Disable calibration waveform output	-
Cy_Rtc_CalibrationControlEnable (uint8_t calib_val, cy_en_rtc_calib_sign_t calib_sign, cy_en_rtc_cal_sel_t cal_sel)	Set calibration control register and enable to calibration wave form output calib_val: Calibration value for absolute frequency. calib_sign: Calibration sign cal_sel: Select calibration wave output signal	CALIB_VALUE = 25ul, CY_EN_RTC_CALIB_SIGN_POSITIVE= 1, CY_EN_RTC_CAL_SEL_CAL1 = 3

Code Listing 35 shows an example program to change the output frequency of the calibration waveform.

Code Listing 35 Example program to change the output frequency of the calibration waveform

```
/* Configure the RTC_CAL pin definition (P1_2)pin */
/* Definition for RTC_CAL. */
#define RTC_CAL_PIN                  P1_2_SRSS_CAL_WAVE
#define USER_PORT                    GPIO_PRT1
#define USER_PIN                     2

/* Setting value for calibration definition */
/* Definition for calibration value */
#define CALIB_VALUE                  25ul

/* RTC configuration */
cy_stc_rtc_config_t const RTC_config =
{
    /* Initiate time and date */
    /* Configure initial time and date parameters. See Table 4. */
    .sec       = RTC_INITIAL_DATE_SEC,
    .min       = RTC_INITIAL_DATE_MIN,
    .hour      = RTC_INITIAL_DATE_HOUR,
    .hrMode    = RTC_INITIAL_DATE_HOUR_FORMAT,
    .dayOfWeek = RTC_INITIAL_DATE_DOW,
    .date      = RTC_INITIAL_DATE_DOM,
    .month     = RTC_INITIAL_DATE_MONTH,
    .year      = RTC_INITIAL_DATE_YEAR,
};

/* RTC_CAL pin configuration */
/* Configuration for RTC_CAL in the GPIO. See Table 7. */
cy_stc_gpio_pin_config_t user_led_port_pin_cfg =
{
    .outVal    = 0ul,
    .driveMode = CY_GPIO_DM_STRONG_IN_OFF,
    .hsiom     = RTC_CAL_PIN,
    .intEdge   = 0ul,
    .intMask   = 0ul,
    .vtrip     = 0ul,
    .slewRate  = 0ul,
    .driveSel  = 0ul,

};

int main(void)
{
    cy_en_rtc_status_t retVal;
    uint8_t retStatus;
    uint32_t interruptState;

    SystemInit();

    __enable_irq(); /* Enable global interrupts. */
```

```
    /* Set the WCO as the clock source to the RTC block */
    retStatus = Cy_SysClk_WcoEnable(100ul);
    if(retStatus == CY_SYSCLK_SUCCESS)
    {
    Cy_Rtc_clock_source(CY_RTC_CLK_SRC_WCO);   /* Set the clock source to the RTC. See Code
Listing 29. */
    }

    /* Configure the RTC_CAL pin (P1_2) */
    /* (1) Configure the RTC_CAL pin */
    Cy_GPIO_Pin_Init(USER_PORT, USER_PIN, &user_led_port_pin_cfg);

    /* Wait for initial setting */
    /* Set the RTC initial value. The initialization part is omitted.  See Code Listing 4. */
    while(Cy_Rtc_Init(&RTC_config) != CY_RET_SUCCESS);



    /* Check for RTC busy status */
    /* Check RTC busy status. See Code Listing 36. */
    while(Cy_Rtc_GetSyncStatus()==CY_RTC_BUSY);


    /* Check for RTC read bit clear */
    /* Check READ bit status. See Code Listing 37, Code Listing 37. */
    if(Cy_Rtc_IsReadBitSet()==1)
    {
         Cy_Rtc_ClearReadBit();
    }

    /* Check for read bit status */
    while(Cy_Rtc_IsReadBitSet()==1);

    /* Set to RTC write bit */
    interruptState = Cy_SysLib_EnterCriticalSection();
    retVal = Cy_Rtc_WriteEnable(CY_RTC_WRITE_ENABLED);  /* Set the WRITE bit for changing the
calibration waveform. See Code Listing 39. */
    if(retVal == CY_RTC_SUCCESS)
    {


       /* Disable calibration waveform output */
       /* Disable the calibration waveform. See Code Listing 40.
        Cy_Rtc_CalibrationControlDisable();

        /* Change waveform frequency for 1Hz */
        /* Select the waveform frequency and enable the calibration waveform. See Code Listing
41. */
        Cy_Rtc_CalibrationControlEnable(CALIB_VALUE, CY_EN_RTC_CALIB_SIGN_POSITIVE,
CY_EN_RTC_CAL_SEL_CAL1);
        /* Clear the RTC Write bit */
```

```
        Cy_Rtc_WriteEnable(CY_RTC_WRITE_DISABLED);  /* Clear the WRITE bit. See Code Listing
39. */
    }
    Cy_SysLib_ExitCriticalSection(interruptState);

    for(;;)
    {

    }
}
```

2.4.7 Example program to confirm the accuracy of WCO frequency after calibration in the driver part

Code Listing 36 to Code Listing 41 shows an example program to confirm the accuracy of the WCO frequency after calibration in the driver part.

Code Listing 36 Example program to get the sync status in the driver part

```
uint32_t Cy_Rtc_GetSyncStatus(void)
{
    /* (2) Check RTC busy status. */
    return((_FLD2BOOL(BACKUP_STATUS_RTC_BUSY, BACKUP->unSTATUS.u32Register)) ? CY_RTC_BUSY :

CY_RTC_AVAILABLE);
}
```

Code Listing 37 Example program to check the READ bit status in the driver part

```
bool Cy_Rtc_IsReadBitSet(void)
{
    if(BACKUP->unRTC_RW.stcField.u1READ == 0ul)  /* (3) Check READ bit status.*/
    {
        return false;
    }
    else
    {
        return true;
    }
}
```

Code Listing 38 Example program to clear the READ bit in the driver part

```
cy_en_rtc_status_t Cy_Rtc_ClearReadBit(void)
{
    BACKUP->unRTC_RW.stcField.u1READ = 0ul;  /* This function clears the READ bit to 0. */

    return(CY_RTC_SUCCESS);
}
```

Code Listing 39 Example program to write enable in the driver part

```
cy_en_rtc_status_t Cy_Rtc_WriteEnable(uint32_t writeEnable)
{
    cy_en_rtc_status_t retVal = CY_RTC_INVALID_STATE;

    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
        */
        if((CY_RTC_BUSY != Cy_Rtc_GetSyncStatus()) && (!_FLD2BOOL(BACKUP_RTC_RW_READ, BACKUP->
            unRTC_RW.u32Register)))
        {
            BACKUP->unRTC_RW.u32Register |= BACKUP_RTC_RW_WRITE_Msk;  /* (4) Set the WRITE bit
for changing the calibration waveform */
            retVal = CY_RTC_SUCCESS;
        }
    }
    else
    {
        /* Clearing Write Bit to complete write procedure */
        BACKUP->unRTC_RW.u32Register &= ((uint32_t) ~BACKUP_RTC_RW_WRITE_Msk);  /* (8) Clear
the WRITE bit */

        /* wait until CY_RTC_BUSY bit is cleared */
        while(CY_RTC_BUSY == Cy_Rtc_GetSyncStatus());  /* (9) Check the RTC busy status. */

        retVal = CY_RTC_SUCCESS;
    }

    return(retVal);
}
```

Code Listing 40 Example program to disable the calibration waveform output in the driver part

```
cy_en_rtc_status_t Cy_Rtc_CalibrationControlDisable(void)
{
    if(BACKUP->unRTC_RW.stcField.u1WRITE == 0)
    {
        // Writes are ignored unless Write bit is set
        return CY_RTC_INVALID_STATE;
    }

    /* (5) Disable the calibration waveform */
    BACKUP->unCAL_CTL.stcField.u1CAL_OUT    = 0ul; // Output disable for wave signal for
calibration

    return CY_RTC_SUCCESS;
}
```

Code Listing 41 Example program to set 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_cal_sel_t cal_sel)
{
    if(BACKUP->unRTC_RW.stcField.u1WRITE == 0)
    {
        // Writes are ignored unless Write bit is set
        return CY_RTC_INVALID_STATE;
    }

    un_BACKUP_CAL_CTL_t tmpBackupCalCtl = {0ul};
    tmpBackupCalCtl.stcField.u6CALIB_VAL  = calib_val;
    tmpBackupCalCtl.stcField.u1CALIB_SIGN = (uint32_t)calib_sign;
    tmpBackupCalCtl.stcField.u2CAL_SEL    = (uint32_t)cal_sel;  /* (6) Select the waveform
frequency */
    tmpBackupCalCtl.stcField.u1CAL_OUT    = 1ul; /* (7) Enable the calibration waveform */ //
Output enable for wave signal for calibration and allow CALIB_VAL to be written.
    BACKUP->unCAL_CTL.u32Register = tmpBackupCalCtl.u32Register;

    return CY_RTC_SUCCESS;
}
```

3 Glossary

Terms	Description
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 reference manual 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 reference manual for details.
ILO	Internal Low-speed Oscillator is the embedded oscillator inside TRAVEO™ T2G device. See the Clocking System chapter of the architecture reference manual for details.
CLK_LF	Low-Frequency Clock is selectable. See the Clocking System chapter of the architecture reference manual 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 reference manual for details.
LPECO	Low-power external crystal oscillator is the external crystal that is input from the external LPECO_IN and LPECO_OUT pins. See the Clocking System chapter of the architecture reference manual for details.
Calibration waveform	The signal outputs from the RTC_CAL pin for adjusting the RTC. See the WCO/LPECO Calibration section in the Real-Time Clock chapter of the architecture reference manual 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 TRAVEO™ T2G device is mounted.
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 reference manual for details.

3 Glossary

Terms	Description
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 reference manual 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 reference manual 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 reference manual 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 reference manual 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 reference manual for details.
WCO device and logic	WCO device is an external crystal oscillator. WCO logic is a circuit inside TRAVEO™ T2G device.
LPECO device and logic	LPECO device is an external crystal oscillator. LPECO logic is a circuit inside TRAVEO™ T2G device.
Power mode/Device power mode	TRAVEO™ T2G family has several power modes in the order of decreasing power consumption. See the Device Power Modes chapter of the architecture TRM for details.
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 reference manual for details.
Interval time	A time of periodic interrupt.
Power on reset	Reset operation after power on sequence. See the Rest System chapter of the architecture reference manual 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 reference manual for details.

4 References

The following are the TRAVEO™ T2G family series datasheets and technical reference manuals. Contact Technical Support to obtain these documents.

[1] Device datasheet

- CYT2B6 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G Family
- CYT2B7 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G Family
- CYT2B9 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G Family
- CYT2BL datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
- CYT3BB/4BB datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G Family
- CYT4BF datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G Family
- CYT6BJ datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G Family (Doc No. 002-33466)
- CYT3DL datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G Family
- CYT4DN datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G Family
- CYT4EN datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-30842)
- CYT2CL datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family

[2] Technical reference manual

- Body controller entry family
  - TRAVEO™ T2G automotive body controller entry family architecture technical reference manual
  - TRAVEO™ T2G automotive body controller entry registers technical reference manual for CYT2B7
  - TRAVEO™ T2G automotive body controller entry registers technical reference manual for CYT2B9
  - TRAVEO™ T2G automotive body controller entry registers technical reference manual for CYT2BL (Doc No. 002-29852)
- Body controller high family
  - TRAVEO™ T2G automotive body controller high family architecture technical reference manual
  - TRAVEO™ T2G automotive body controller high registers technical reference manual for CYT3BB/4BB
  - TRAVEO™ T2G automotive body controller high registers technical reference manual for CYT4BF
  - TRAVEO™ T2G automotive body controller high registers technical reference manual for CYT6BJ (Doc No. 002-36068)
- Cluster 2D family
  - TRAVEO™ T2G automotive cluster 2D architecture technical reference manual
  - TRAVEO™ T2G automotive cluster 2D registers technical reference manual for CYT3DL
  - TRAVEO™ T2G automotive cluster 2D registers technical reference manual for CYT4DN
  - TRAVEO™ T2G automotive cluster 2D registers technical reference manual for CYT4EN (Doc No. 002-35181)
- Cluster entry family
  - TRAVEO™ T2G automotive cluster entry family architecture technical reference manual
  - TRAVEO™ T2G automotive cluster entry registers technical reference manual for CYT2CL

5 Other references

Infineon provides the Sample Driver Library (SDL) including startup as sample software to access various peripherals. SDL also serves as a reference for customers, providing drivers that are not covered by the official AUTOSAR products. The SDL is not intended for production use, as it does not meet any automotive standards. The code snippets in this application note are part of the SDL. Contact Technical Support to obtain the SDL.

Revision history

Document revision	Date	Description of changes
**	2019-01-15	New Application Note
*A	2019-03-15	Added target parts number (CYT4B series)
*B	2019-07-30	Added descriptions of LPECO and target parts number (CYT4D series).
*C	2020-03-10	Changed target parts number (CYT2/CYT4 series). Added target parts number (CYT3 series).
*D	2020-12-15	Added flowchart and example codes Moved to Infineon Template
*E	2022-09-14	Update code example in driver part. Added the procedure to check the RTC busy status for RTC register update.
*F	2023-11-22	Template update; no content update.
*G	2024-12-02	Added to CYT6BJ

Trademarks

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

Edition 2024-12-02

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-ibw1681382730311

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.