TRAVEO™ T2G ファミリの RTC 使用方法

本書について

適用範囲と目的

このアプリケーションノートでは、TRAVEO™ T2G ファミリ MCU のリアルタイムクロック (RTC) 機能の使用方法を説明します。

対象者

本書は、TRAVEO™ T2G ファミリの RTC を使用するすべての人を対象とします。

目次

1 はじめに

TRAVEO™ T2G ファミリの RTC 機能には、RTC, アラーム, 校正, およびバックアップレジスタの 4 つの機能があります。

RTC は、現在時刻 (年, 月, 日, 曜日, 時, 分, 秒) を正確に刻みます。アラーム機能は、プログラム可能な設定で CPU への割込みを生成できます。校正機能では、WCO および低電力外部水晶発振器 (LPECO) の周波数誤差を修正できます。バックアップレジスタは、どのパワーモードでもユーザデータを保持できます。

このアプリケーションノートでは、下記を説明します。

- RTC レジスタの時刻を更新する方法
- RTC レジスタから時刻を読み出す方法
- シリーズの RTC 機能
- RTC 機能の設定方法および WCO の校正方法

このアプリケーションノートに記載されている機能説明や用語について理解いただくためには、architecture reference manual の Real-Time Clock 章を参照してください。

1.1 RTC 機能

以下は RTC の機能です。

- 充実した RTC 機能
  - 年/月/日, 曜日, 時: 分: 秒 (すべての領域は整数値)
  - 12 時制と 24 時制の両方に対応
  - 2400 年までの「うるう年修正」に対応
- 設定可能なアラーム機能
  - 月/日, 曜日, 時: 分: 秒に対応したアラーム
  - 2 つの独立したアラーム
- 32.768 kHz WCO および 4 MHz～8 MHz LPECO の校正機能
  - 校正波形の出力
  - 512 Hz, 1 Hz, および 2 Hz に対応
- バックアップレジスタ

***注**: 使用されるデバイスが LPECO に対応しているかどうかは、デバイスのデータシートを確認してください。*

2 動作概要

RTC 機能は、現在時刻を正確に刻むため、自動車アプリケーションの正確な時刻源として使用できます。このアプリケーションノートでは、使用例として自動車内の時計表示について説明します。この時計表示は、正確かつ定期的に更新されます。

図 1 に、RTC 機能ブロックダイヤグラムとユーザシステムの例を示します。

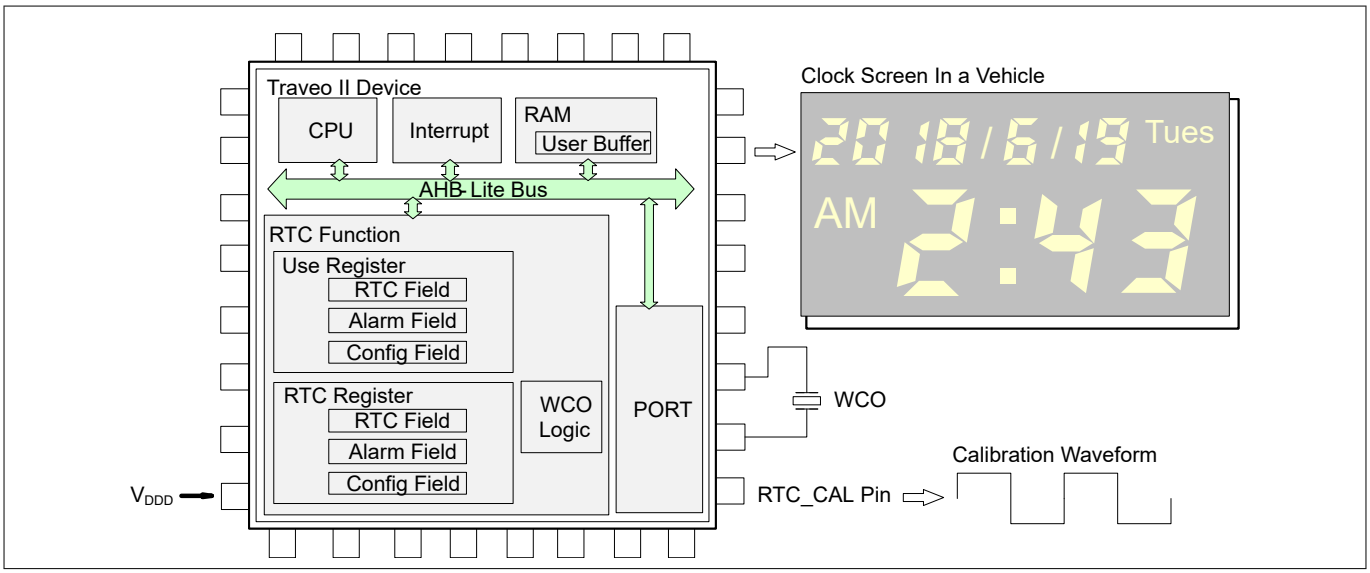

図 1 RTC 機能を使用したユーザシステムの構造例

RTC 機能は、ユーザレジスタ, RTC レジスタ, および WCO と LPECO で構成されます。RTC 機能は、CPU や他のサブシステムと、AHB-Lite バスインタフェースを介して接続されます。外部水晶発振子や外部クロック入力によって、WCO または LPECO は必要なクロックを生成します。RTC 機能はプログラマブルなアラーム機能を持っており、CPU に割込みを発生させられます。

自動車内の時計表示に現在時刻を更新および表示するために、周期的な割込みを伴う RTC 機能が一般的に使用されます。CPU と RTC 機能は、AHB-Lite バスインタフェースで接続され、ファームウェアアクセスインタフェースを提供します。

CPU は RTC 機能から現在の時間を読み出して、RAM 上に割り当てられたローカル変数であるユーザバッファにリアルタイムデータを格納し、外部時計画面にリアルタイムデータを出力します。さらに RTC 機能により校正波形の出力も可能です。

ユーザレジスタと RTC レジスタは、それぞれ RTC フィールド, Alarm フィールド, および Config フィールドを持ちます。ソフトウェアは、ユーザレジスタにアクセス可能です。決められた書き込み手順により、ユーザレジスタで RTC レジスタを更新します。決められた読み出し手順により、RTC レジスタからユーザレジスタへデータをコピーします。RTC フィールド, Alarm フィールド, および Config フィールドの詳細は、architecture reference manual と register reference manual を参照してください。

WCO デバイスとロジック, LPECO とロジック, および RTC 機能は、連続電源である VDDD で動作します。したがって、RTC 機能はすべての電力モードで実行され、現在時刻を継続的に刻み続けます。

現在時刻を刻み続ける操作は 4 つあります。

1. 割込みを含む RTC 機能の初期化
2. 時間と日付の読み出し
3. 時間と日付の更新
4. WCO の校正

RTC 機能を使用する前に、初期化ルーチンを実行する必要があります。初期化処理にはアラーム割込みの設定も含みます。割込みは、RTC フィールドレジスタに対応するフィールドを持つアラーム機能によって引き起こされ

ます。ベーシック RTC の設定でアラーム機能を 2 つ使用して、30 秒ごとに割込みを発生させる設定例を説明します。

現在時刻を自動車内の時計画面に表示させるためには、CPU は現在時刻を RTC フィールドのレジスタから読み出します。時間と日付の読出しで現在時刻を RTC フィールドから読み出す例を説明します。

夏時間 (DST) などのために、RTC フィールドレジスタを更新する必要がある場合があります。DST では、時間フィールドを更新する必要があります。時間と日付の更新で RTC フィールドの現在時刻を更新する手順例を説明します。

RTC 機能のソースクロックとして、ILO, CLK_LF, WCO, または LPECO を使用できます。しかし、より正確であるためには、WCO または LPECO を使用することを推奨します。WCO の校正で RTC 機能の時間を校正する例を説明します。

2.1 ベーシック RTC の設定

ここでは、下記の前提のもとで RTC 機能の動作について説明します。

ここでは、インフィニオンが提供するサンプルドライバライブラリ (SDL) を使用して、使用例に基づいて RTC を設定する方法についても説明します。このアプリケーションノートのプログラムコードは SDL の一部です。SDL についてはその他の参考資料 を参照してください。

SDL には基本的に、設定部とドライバ部があります。設定部は、主に目的の操作のためのパラメータ値を設定します。ドライバ部は、設定部のパラメータを各レジスタに設定します。

2.1.1 使用例

ここでは、次の使用例によって示される RTC 機能を使った例を説明します。この使用例は、RTC 機能を有効にするために、入力クロック, 時刻, および日付などの項目を初期化する方法を示します。さらに、この使用例は ALARM 機能も有効にします。ALARM 設定時刻と RTC 時刻が一致すると割込みが生成されます。

使用例:

- ソースクロック: 内部低速発振器 (ILO)
- 設定年: 2019
- 設定月: 8 月
- 設定日: 21 日
- 曜日: 月曜日
- 設定時間: 12:00:00
- HR モードの設定: 24HR
- ALARM1: 0 秒ごと
- ALARM1 の IRQ 番号: 3

2.1.2 RTC 機能の初期化

以下に、パワーオンリセット後に RTC 機能を初期化する例を説明します。RTC 機能は一度初期化をした後は、デバイスがいかなるパワーモードに遷移しても、RTC 機能を再初期化をする必要はありません。

図 2 にベーシック RTC 設定のフロー例を示します。

2 動作概要

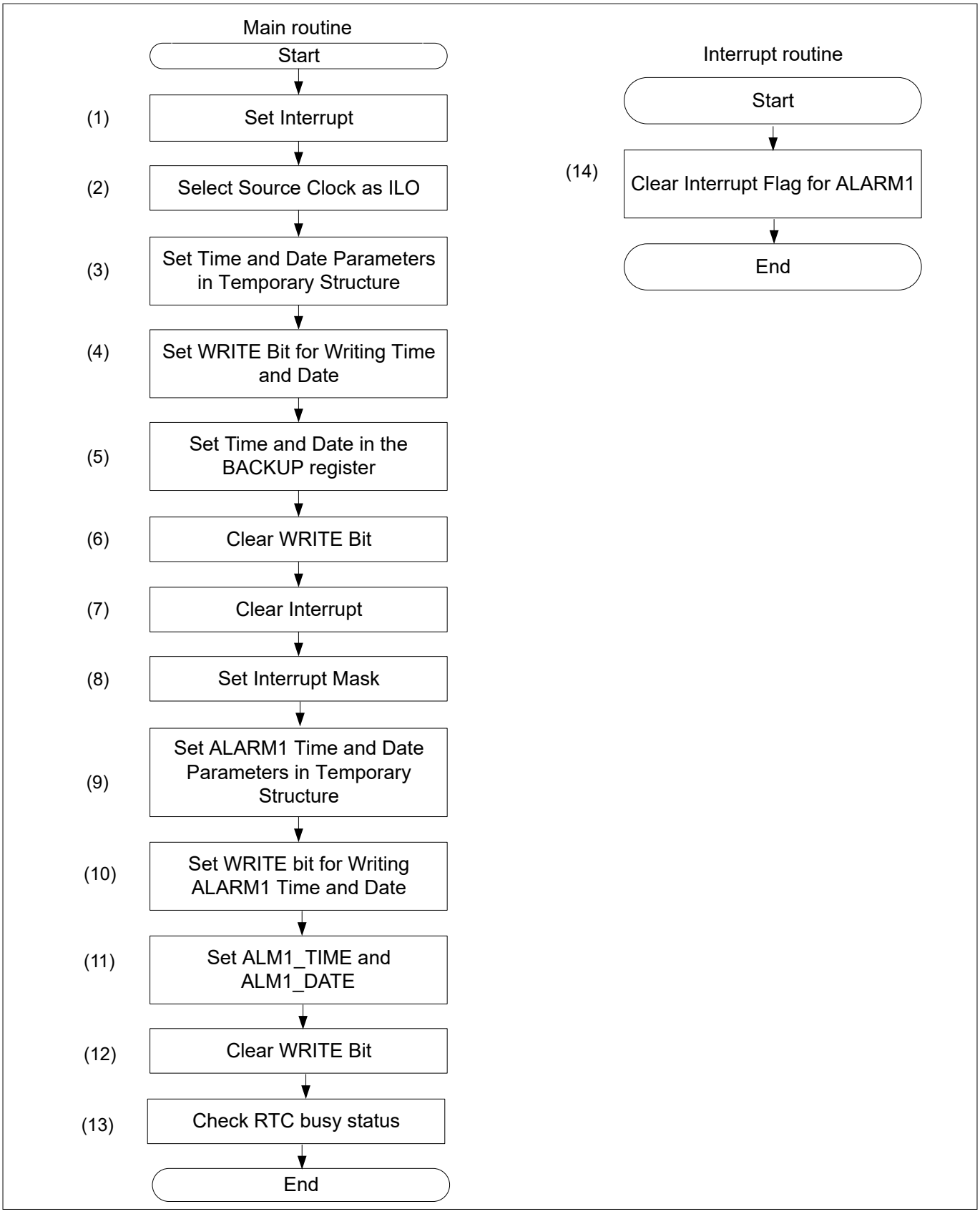

図 2　　ベーシック RTC 設定と割込みルーチンを設定するフロー例

以下の手順は、RTC 機能の基本的なセットアップ手順を示します。

1.　割込みをセットしてください。

2. ソースクロックを ILO として選択してください。BACKUP_CTL.CLK_SEL = CY_RTC_CLK_SRC_ILO_0 (ILO) を書き込んでください。
3. 時刻と日付パラメータを設定してください。TIME パラメータ (秒=0, 分=0, 時=12, HR モード=0(24HR), 曜日=1(月曜日))DATE パラメータ (日付=21, 月=8, 年=19 (2019 年))
4. 時刻と日付を書き込むために WRITE ビットを設定してください。BACKUP_RTC_RW.WRITE = '1'を書き込んでください。
5. BACKUP_RTC_TIME および BACKUP_RTC_DATE レジスタに時刻と日付を設定してください: BACKUP_RTC_TIME = 時刻パラメータに書き込む BACKUP_RTC_DATE = 日付パラメータに書き込む
6. RTC レジスタの時間と日付を更新するために WRITE ビットをクリアしてください。BACKUP_RTC_RW.WRITE = '0'を書き込んでください。
7. 割込みをクリアしてください。
8. 割込みマスクを設定してください。
9. ALARM1 の日時パラメータを暫定構造に設定してください。
10. 時間と日付を ALARM1 に書き込むために、WRITE ビットを設定してください。
11. BACKUP_ALM1_TIME および BACKUP_ALM1_DATE レジスタに ALARM1 の時刻と日付を設定してください。RTC フィールドとアラームフィールドが一致すると、割込みが発生します。BACKUP_ALM1_TIME = ALARM1 時刻パラメータ (0 秒) を書き込んでください。BACKUP_ALM1_DATE = ALARM1 日付パラメータを書き込んでください。(Date パラメータに ALARM はありません)
12. 時間と日付を更新するために WRITE ビットをクリアしてください。BACKUP_RTC_RW.WRITE = '0'を書き込んでください。
13. RTC レジスタ更新のために RTC ビジーステータスを確認してください。
14. ALARM1 割込みが発生したら、割込みフラグをクリアしてください。

アラーム機能は、RTC フィールドの値とアラームフィールドの値が一致した時に、割込みを生成します。

30 秒ごとにアラームを設定したい場合は、2 つのアラームを設定する必要があります。

割込み設定手順の詳細については、AN219842 の Interrupt and Fault Report Structure 章を参照してください。

2.1.3 RTC 機能の設定

SDL における RTC と ALARM の設定部分について、2 つのセクションに分けて説明します。

- 表 1 パラメータの一覧
- 表 2 関数の一覧

表 1 RTC および ALARM パラメータ

パラメータ	説明	値
RTC_config.sec	カレンダー 秒、0～59	RTC_INITIAL_DATE_SEC = 0ul
RTC_config.min	カレンダー 分、0～59	RTC_INITIAL_DATE_MIN = 0ul
RTC_config.hour	カレンダー 時間、12/24HR モードによる値	RTC_INITIAL_DATE_HOUR = 12ul
RTC_config.hrMode	12/24HR モードを選択する: 1=12HR, 0=24HR CY_RTC_12_HOURS = 1ul CY_RTC_24_HOURS = 0ul	RTC_INITIAL_DATE_HOUR_FORMAT = CY_RTC_24_HOURS

(続く)

2 動作概要

表 1 (続き) RTC および ALARM パラメータ

パラメータ	説明	値
RTC_config.dayOfWeek	カレンダー 曜日、1～7 値は自由に設定できますが、1=月曜日とすることを推奨します。	RTC_INITIAL_DATE_DOW = 1ul
RTC_config.date	カレンダー 日付、1～31 うるう年の自動修正	RTC_INITIAL_DATE_DOM = 21ul
RTC_config.month	カレンダー 月、1～12	RTC_INITIAL_DATE_MONTH = 8ul
RTC_config.year	カレンダー 年、0～99	RTC_INITIAL_DATE_YEAR = 19ul
alarm.sec	アラーム 秒、0～59	0ul
alarm.sec_en	アラーム 秒を有効にする: 0=無視、1=一致	CY_RTC_ALARM_ENABLE = 1ul
alarm.min	アラーム 分、0～59	0ul
alarm.min_en	アラーム 分を有効にする: 0=無視、1=一致	CY_RTC_ALARM_DISABLE = 0ul
alarm.hour	アラーム 時間、12/24HR モードによる値	0ul
alarm.hour_en	アラーム 時間を有効にする: 0=無視、1=一致	CY_RTC_ALARM_DISABLE = 0ul
alarm.dayOfWeek	カレンダー 曜日、1～7 値は自由に設定できますが、1=月曜日とすることを推奨します。	1ul
alarm.dayOfWeek_en	アラーム 曜日を有効にする: 0=無視、1=一致	CY_RTC_ALARM_DISABLE = 0ul
alarm.date	カレンダー 日付、1～31 うるう年の自動修正	1ul
alarm.date_en	アラーム 日付を有効にする: 0=無視、1=一致	CY_RTC_ALARM_DISABLE = 0ul
alarm.month	アラーム 月、1～12	1ul
alarm.month_en	アラーム 月を有効にする: 0=無視、1=一致	CY_RTC_ALARM_DISABLE = 0ul
alarm.alm_en	アラーム 1 のマスターイネーブル。 0: アラーム 1 が無効。日付と時刻のフィールドは無視される。 1: アラーム 1 が有効。日時フィールドがいずれも有効でない場合、このアラームは 1 秒に 1 回作動する。	CY_RTC_ALARM_ENABLE = 0ul

2 動作概要

表 2　　RTC および ALARM 機能

関数	説明	値
Cy_Rtc_clock_source (clock_source)	RTC 入力クロックソースを設定 Clock source: 入力クロックソース	CY_RTC_CLK_SRC_ILO_0
Cy_Rtc_Init (*config)	RTC ドライバを初期化し、RTC レジスタアドレスを返します。 config: RTC コンフィギュレーション構造アドレス	&RTC_config
Cy_Rtc_SetDateAndTime (*dateTime)	時刻と日付の値を RTC_TIME と RTC_DATE レジスタに設定します。 dateTime: RTC コンフィギュレーション構造アドレス	config
Cy_Rtc_ConstructTimeDate (*timeDate, time, date)	渡された個々の要素から、API で使用される形式で整数の時刻と整数の日付を返します。 timeDate: 時刻と日付の構造アドレス time: RTC_TIME レジスタ設定の時間設定構造アドレス date: RTC_TIME レジスタ設定の日付設定構造アドレス	dateTime, &tmpTime, &tmpDate
Cy_Rtc_ClearInterrupt (interruptMask)	RTC 割込みをクリアします interruptMask: クリアする割込みのビットマスク	CY_RTC_INTR_ALARM1=0x1ul
Cy_Rtc_SetInterruptMask (interruptMask)	RTC 割込みを設定します interruptMask: 設定する割込みのビットマスク	CY_RTC_INTR_ALARM1=0x1ul
Cy_Rtc_SetAlarmDateAndTime (alarmDateTime, alarmIndex)	ALMx_TIME と ALMx_DATE レジスタにアラーム時刻と日付の値を設定する alarmDateTime: アラーム設定構造 alarmIndex: 設定するアラームインデックス	&alarm, CY_RTC_ALARM_1=0ul

(続く)

2 動作概要

表 2 (続き) RTC および ALARM 機能

関数	説明	値
Cy_Rtc_ConstructAlarmTimeDate (*alarmDateTime, *alarmTime, *alarmDate)	アラームに渡された個々の要素から、API で使用される形式で 整数の時刻と整数の日付を返します。 alarmDateTime: アラーム時刻と日付の構造アドレス alarmTime: ALMx_TIME レジスタ設定のアラーム時刻設定構造アドレス alarmDate: ALMx_DATE レジスタ設定のアラーム日付設定構造アドレス	alarmDateTime, &tmpAlarmTime, &tmpAlarmDate
_VAL2FLD (field, value)	レジスタのビット範囲で使用するビットフィールド値をマスクおよびシフトします。 Field: レジスタビットフィールド名 値ビットフィールドの値。このパラメータは uint32_t 型として解釈されます。	-

Code Listing 1 に、RTC 機能の設定部のプログラム例を示します。

以下の説明は、SDL ドライバ部のレジスタ表記を理解するのに役立ちます。

- **BACKUP**->un**RTC_TIME** レジスタは、register reference manual に記載されている BACKUP_RTC_TIME レジスタです。他のレジスタも同様に記述されます。

レジスタ表記の結合と構造の詳細については、`hdr/rev_x/ip` の `cyip_backup_v3.h` を参照してください。

2 動作概要

Code Listing 1 RTC を設定するプログラム例

```
cy_stc_rtc_config_t const RTC_config =
{
    /* Initiate time and date */
    /* Configure initial time and date parameters. See 表 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 表 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 に、アラーム 1 の RTC 割込みルーチンのプログラム例を示します。

Code Listing 2 アラーム 1 の RTC 割込みルーチンのプログラム例

```
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 ドライバ部に RTC 機能を設定するサンプルプログラム

Code Listing 3～Code Listing 11 に、ドライバ部に RTC を設定するプログラム例を示します。

2 動作概要

Code Listing 3 ドライバ部に RTC 入力クロックソースを設定するプログラム例

```
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 ドライバ部の RTC を初期化するプログラム例

```
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);
}
```

2 動作概要

Code Listing 5 ドライバ部の日時レジスタを設定するプログラム例

```
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 動作概要

Code Listing 6 ドライバ部に時刻と日付を作成するプログラム例

```
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 ドライバ部にイネーブルを書き込むプログラム例

```
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 ドライバ部の割込みをクリアするプログラム例

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

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

2 動作概要

Code Listing 9 ドライバ部に割込みマスクを設定するプログラム例

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

2 動作概要

Code Listing 10 ドライバ部にアラーム時刻と日付を設定するプログラム例

```
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);
}
```

2 動作概要

Code Listing 11 ドライバ部にアラーム時刻と日付を作成するプログラム例

```
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 時間と日付の読出し

現在時刻を自動車内の時計表示に表示させるために、CPU は現在時刻を RTC フィールドのレジスタから読み出します。

TRAVEO™ T2G デバイスの RTC 機能は、BACKUP_RTC_RW レジスタの READ ビットを使用して、現在の RTC フィールド値を RTC レジスタからユーザーレジスタの RTC フィールドにリアルタイムでコピーします。ユーザーファームウェアは、RTC レジスタ内の RTC フィールド値が更新されても、ユーザーレジスタ内の RTC フィールド値を固定します。その後、ユーザファームウェアは、固定された RTC 値をユーザーバッファにコピーし、それを RAM 内のローカル変数に割り当てます。

図 3 に、現在時刻を読み出す使用例を示します。この例では、4 月 30 日 午後 11 時 59 分 59 秒に現在時刻を読み出します。初めに、BACKUP_RTC_RW レジスタの READ ビットに”1”を設定します。ハードウェアはただちに、RTC レジスタの RTC フィールドデータをユーザレジスタの RTC フィールドにコピーし、ユーザファームウェアは READ ビットを‘0’にクリアします。その後、現在の日付と時刻のユーザー登録を読み出せます。

ここでは、SDL を使用した使用例に基づいて RTC 値を読み出す方法を説明します。このアプリケーションノートのプログラムコードは SDL の一部です。SDL については、その他の参考資料を参照してください。

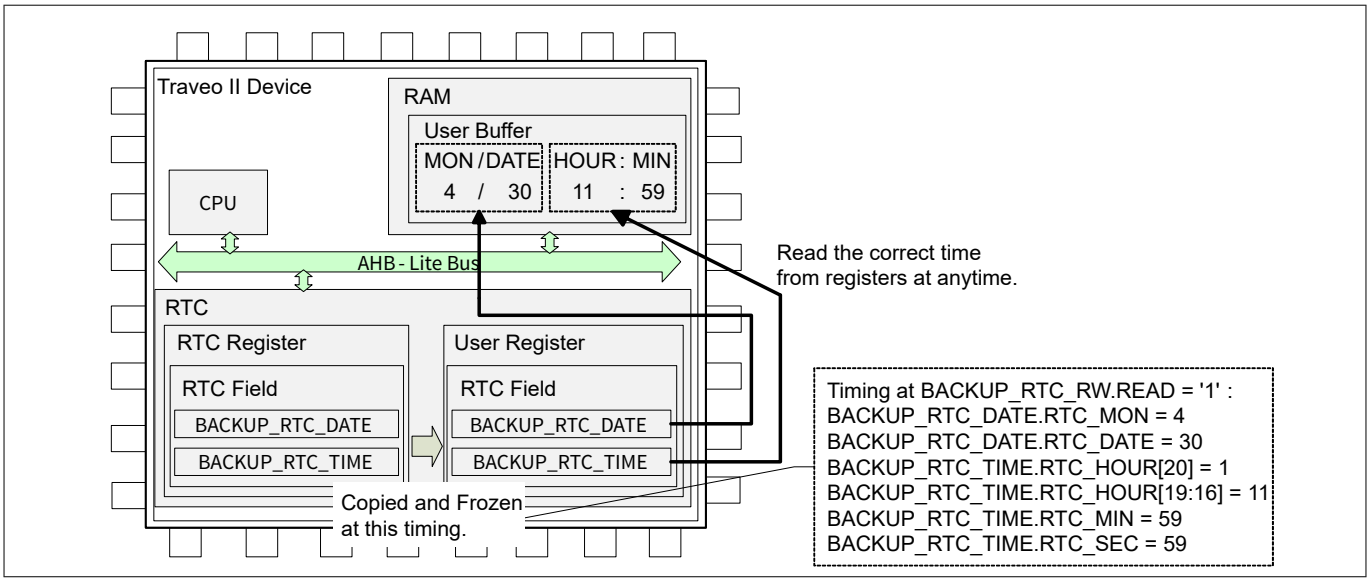

図 3　　READ ビットを使った正しい時刻の読み出し

2.2.1　使用例

ここでは、次の使用例によって RTC 値の読み出し例を説明します。この例では、以下の項目が定期的に読み出されます。したがって、これらに固定値はありません。

使用例: RTC 値の読み出し

- Read Data: Read_DateTime

図 4 に、RTC 値を読み出すフロー例を示します。

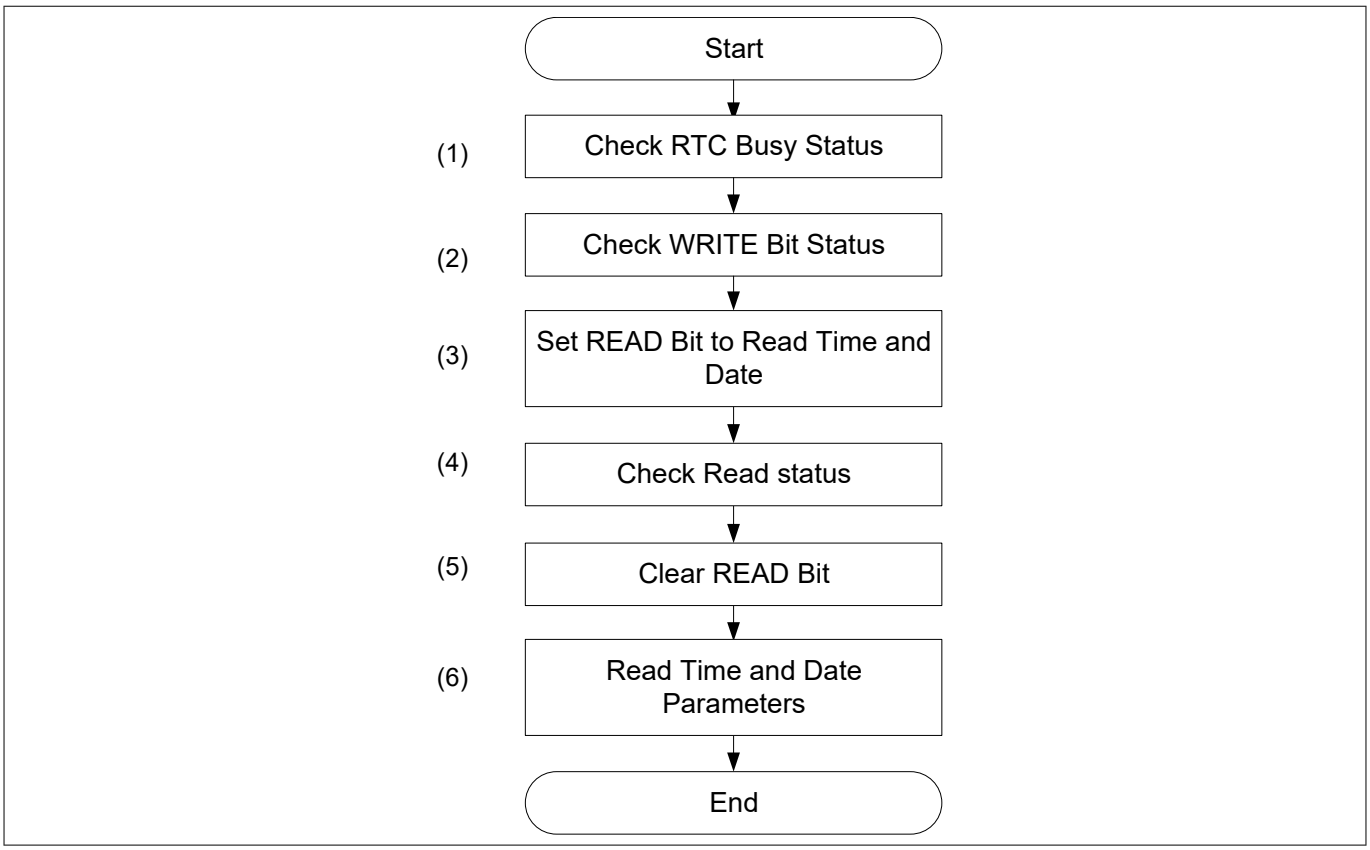

図 4　　RTC 値を読み出すまでのフロー例

2.2.2 RTC 値の読み出し

以下の例は RTC 値を読み出す動作を記述します。

1. RTC busy ステータスを確認してください。安全な設計を確保するために、BACKUP_STATUS.RTC_BUSY = '0'になるまで待ってください [1]。
2. 書き込みステータスを確認してください。安全な設計を確保するために、BACKUP_RTC_RW.WRITE = '0'になるまで待ってください [2]。
3. READ ビットを設定して、RTC 値の読み出し操作を準備してください。

   BACKUP_RTC_RW.READ = '1' (読み出し操作開始)を書き込んでください。

   これにより、RTC レジスタの現在の RTC フィールド値がユーザーレジスタの RTC フィールドにコピーされ、ハードウェアがその値を固定します。
4. 読み出しステータスを確認してください。
5. READ ビットをクリアしてください。

   BACKUP_RTC_RW.READ = '0' (読み出し操作終了) を書き込んでください。
6. ユーザーレジスタから時刻と日付のパラメータを読み出してください。

   時刻パラメータを取得 = BACKUP_RTC_TIME

   日付パラメータを取得 = BACKUP_RTC_DATE

2.2.3 RTC 値の読み出し項目

表 3 に、RTC データの読み出し値を保存するパラメータの一覧を示します。これらの項目は、表 1 および 表 2 に記載されている初期化項目と同じです。

表 3 RTC 値の読み出し

パラメータ	説明
Read_DateTime.sec	保存された秒の値 (カレンダー 秒、0～59)
Read_DateTime.min	保存された分の値 (カレンダー 分、0～59)
Read_DateTime.hour	保存された時間の値 (カレンダー 時間、12/24HR モードによる値)
Read_DateTime.hrMode	保存された時間の HR モード値 (12/24HR モード、0 または 1)
Read_DateTime.dayOfWeek	保存された曜日の値 (カレンダー曜日、1～7) 値は自由に設定できますが、1=月曜日とすることを推奨します。
Read_DateTime.date	保存された月日の値 (カレンダー 月日、1～31) うるう年の自動修正
Read_DateTime.month	保存された月の値 (カレンダー 月、1～12)
Read_DateTime.year	保存された年の値 (カレンダー 年、0～99)

Code Listing 12 に、RTC 値読み出しのプログラム例を示します。例では、メインルーチンの初期化部分のプログラムコードを省略しています。

1 直前の操作が完了するまで、次の読み出し操作は実行できません。
2 WRITE ビットが設定されている場合、次の読み出し操作は実行できません。

Code Listing 12 RTC 値読み出しのプログラム例

```
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～Code Listing 15 に、ドライバ部で RTC 値読み出しのプログラム例を示します。

2 動作概要

Code Listing 13 ドライバ部の RTC の時刻と日付の値を読み出すプログラム例

```
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 ドライバ部の同期 RTC レジスタのプログラム例

```
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 ドライバ部の同期ステータスを取得するプログラム例

```
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 時間と日付の更新

RTC フィールドは、独立して更新できます。ユーザレジスタの各 RTC フィールドは、独立した更新フラグを持ちます。これらの更新フラグは内部フラグで、ユーザファームウェアがアクセスできません。ユーザレジスタの RTC フィールドに値を書き込むことで、これらの更新フラグがセットされます。書込み命令 (BACKUP_RTC_RW.WRITE = '0') により、更新フラグがセットされている RTC フィールドのみが更新されます。

ここでは、インフィニオンが提供するサンプルドライバライブラリ (SDL) を使用して、使用例に基づいて RTC 値を更新する方法について説明します。このアプリケーションノートのプログラムコードは SDL の一部です。SDL については、その他の参考資料を参照してください。

2.3.1 使用例

この使用例は、夏時間 (DST) を調整する方法を示します。DST を調整するために、RTC_HOUR ビットフィールドのみを更新する必要があります。ほかのフィールドは変更されません。

使用例:

- RTC 初期値を設定
  - 2018 年, 6 月, 19 日, 火曜日, 2 時 (12HR モード), 43 分, 0 秒
- RTC 値のカレンダー 時間を'2'から'3'に更新

図 5 に、RTC 値を更新するフロー例を示します。

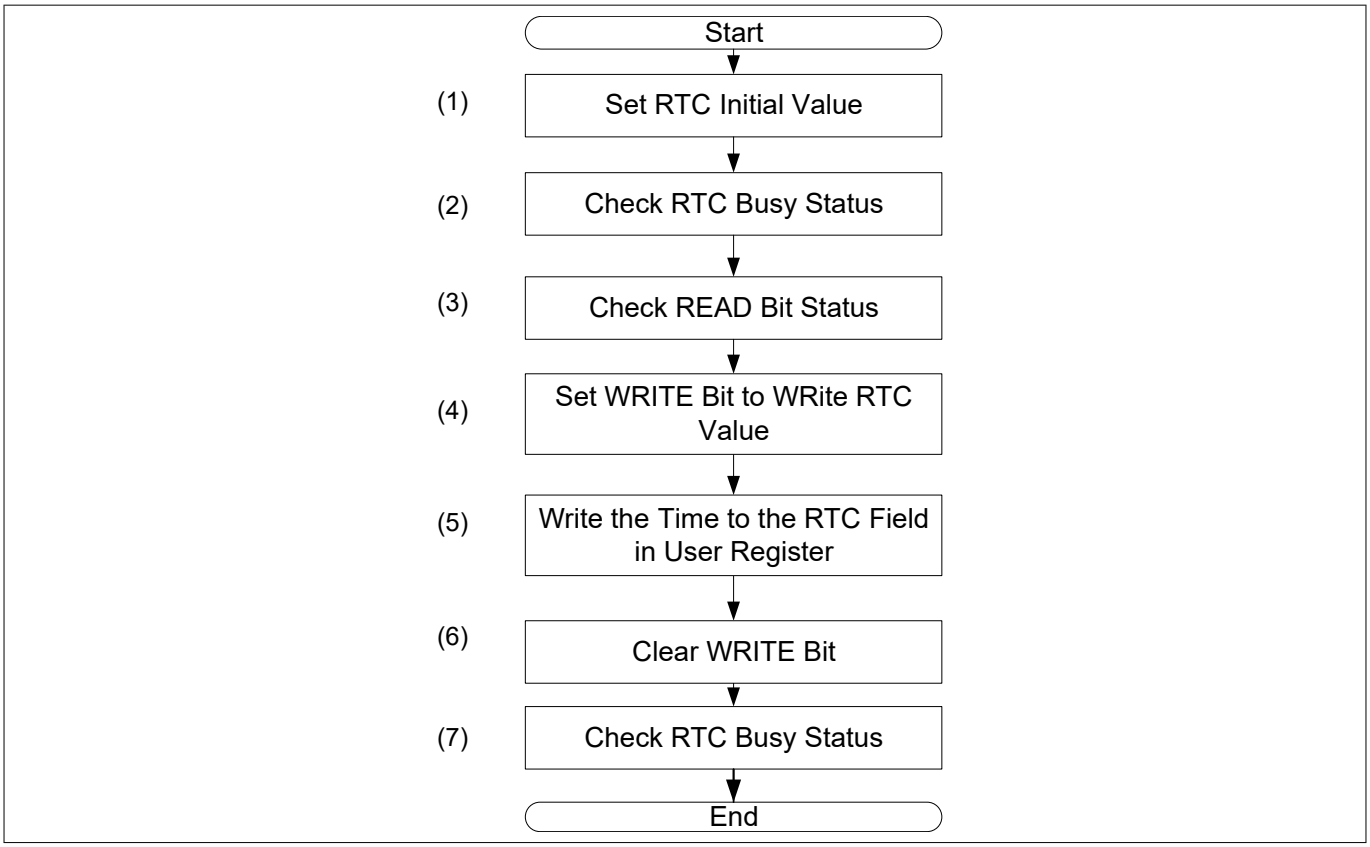

図 5 RTC 値を更新するフロー例

2.3.2 RTC 値の更新 (例: DST の調整)

1. RTC 初期値を設定してください。初期設定方法については、RTC 機能の初期化を参照してください。
2. RTC busy ステータスを確認してください。BACKUP_STATUS.RTC_BUSY = ‘0’になるまで待ってください。
3. 読み出しステータスを確認してください。BACKUP_RTC_RW.READ が‘0’でない場合は、READ ビットをクリアしてください (BACKUP_RTC_RW.READ = ‘0’)。
   - その後、BACKUP_RTC_RW.READ = '0'になるまで待ってください。
4. RTC 値の書き込み動作のために、WRITE ビットの準備の設定をしてください。
   - BACKUP_RTC_RW.WRITE = ‘1’ (書き込み操作開始)を書き込んでください。
5. 更新する時間をユーザレジスタの RTC フィールドに書き込んでください。
   - BACKUP_RTC_TIME を書き込んでください。RTC_HOUR = '3'
   - 例えば、DST を調整するために、米国で 4 月の初めの日曜日の午前 2 時に、RTC_HOUR ビットフィールドに'2'の代わりに'3'を書き込みます。

- 例えば、DST を調整するために、米国で 4 月の初めの日曜日の午前 2 時に、RTC_HOUR ビットフィールドに'2'の代わりに'3'を書き込みます。
- BACKUP_RTC_TIME.RTC_SEC, RTC_MIN, 12HR, DAY, DATE, MON および YEAR などの他の RTC フィールドは変更しないままです。

6. WRITE ビットをクリアしてください。
   - BACKUP_RTC_RW.WRITE = '0' (書き込み操作終了)を書き込んでください。
7. RTC レジスタ更新のために RTC busy ステータスを確認してください。

表 3 に SDL の設定部の RTC 初期パラメータを示します。表 3 の値で初期化した後、RTC 値を更新してください。

表 4 RTC 初期値パラメータ

パラメータ	説明	値
RTC_config.sec	カレンダー 秒、0～59	RTC_INITIAL_DATE_SEC = 0ul
RTC_config.min	カレンダー 分、0～59	RTC_INITIAL_DATE_MIN = 43ul
RTC_config.hour	カレンダー 時間、12/24HR モードによる値	RTC_INITIAL_DATE_HOUR = 2ul
RTC_config.hrMode	12/24HR モードを選択します。 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	カレンダー 曜日、1～7 値の意味の定義はユーザー次第ですが、1=月曜日とすることを推奨します	RTC_INITIAL_DATE_DOW = 2ul
RTC_config.date	カレンダー 日付、1～31 うるう年の自動修正	RTC_INITIAL_DATE_DOM = 19ul
RTC_config.month	カレンダー 月、1～12	RTC_INITIAL_DATE_MONTH = 6ul
RTC_config.year	カレンダー 年、0～99	RTC_INITIAL_DATE_YEAR = 18ul

表 5 に更新された RTC 値を含む SDL の設定部のパラメータを、表 6 に関数を示します。

表 5 更新された RTC 値

パラメータ	説明	値
RTC_UPDATE_HOUR	カレンダー 時間、12/24HR モードによる値	3ul

表 6 更新された RTC 値関数

関数	説明	値
Cy_Rtc_IsReadBitSet (void)	READ ビットステータスを返します。 1: READ ビットが設定されます。 0: READ ビットがクリアされます。	-
Cy_Rtc_ClearReadBit (void)	READ ビットがクリアされます。	-
Cy_Rtc_SetHourBit (uint8_t hour)	RTC 時間値のみ設定	RTC_UPDATE_HOUR = 3ul

2 動作概要

Code Listing 16 に、更新された RTC 値のプログラム例を示します。
以下の説明は、SDL ドライバ部のレジスタ表記を理解するのに役立ちます。

- **BACKUP**->un**RTC_TIME** レジスタは、register reference manual で言及されている BACKUP_RTC_TIME レジスタです。他のレジスタも同様です。

Code Listing 16 RTC 値を更新するプログラム例

```
/* RTC Hour value for update */
            /* RTC Hour value for update.  See 表 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 表 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 ドライバ部で RTC 値を更新するプログラム例

Code Listing 17～Code Listing 20 に、ドライバ部の RTC 値を更新するプログラム例を示します。

Code Listing 17 ドライバ部の同期状態を取得するプログラム例

```
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 ドライバ部の READ ビットの状態を確認するプログラム例

```
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 ドライバ部の READ ビットをクリアするプログラム例

```
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 ドライバ部の RTC 時間値を設定するプログラム例

```
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 WCO の校正

注: WCO 校正は、RTC_PIN を持つデバイスにのみ実施が可能です。

たとえ水晶発振器を WCO に使用したとしても、周波数の誤差があります。この誤差により、RTC 値がお住まいの地域の標準時間と一致しない可能性があります。このような場合は、ユーザが WCO を校正してリアルタイムクロックの精度を向上させることができます。

ここでは、インフィニオンが提供するサンプルドライバーライブラリ (SDL) を使用して、使用例に基づいて WCO を校正する方法について説明します。このアプリケーションノートのプログラムコードは SDL の一部です。SDL については、その他の参考資料を参照してください。

2.4.1 使用例

この使用例は、校正波形の出力、校正値の設定、校正の確認方法を示します。

使用例:

- 512 Hz の校正波形を RTC_CAL ピンに出力する。

- 27.125 ppm のスローダウン校正波形に基づき、レジスタに校正値を設定する。
- 1 Hz 出力で校正波形を確認する。

この使用例には次の 3 つのステップがあります。

1. WCO 波形を RTC_CAL ピンへルーティングして波形をキャプチャ
2. 理想的な波形と実際の WCO 出力との差 (周波数) を測定して、校正値を設定
3. 再測定により WCO 周波数の精度を確認

2.4.2 WCO 波形のキャプチャ

オシロスコープで WCO 波形をキャプチャするためには、CAL_OUT ビットを設定して、デバイスの RTC_CAL ピンに配線します。この動作は、WCO から派生した 512 Hz のクロック波形を RTC_CAL 端子に出力します。次に、この出力の理想的な 512 Hz クロックからの偏差を ppm 値に変換します。その後、WCO の周波数誤差を修正するために必要な校正設定値を計算してください。この操作を行う前に、表 3 で指定した設定で RTC 機能が正常に動作していることを確認してください。

図 6 に、WCO 波形出力を校正するフロー例を示します。

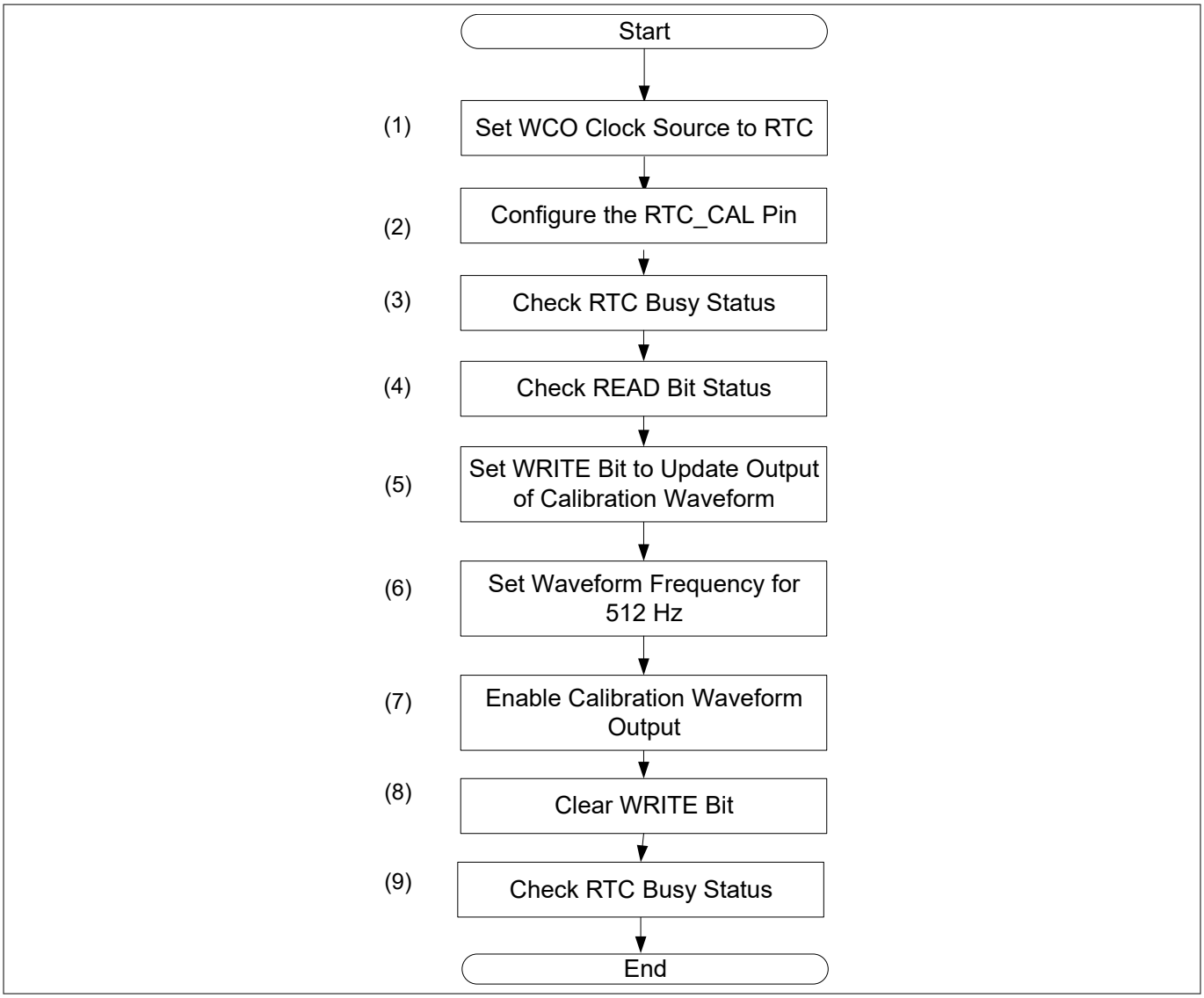

図 6 WCO 波形出力の校正フロー例

WCO 波形を出力する方法を以下の手順で示します。

2 動作概要

1. RTC に WCO クロックソースを設定します。
2. RTC_CAL ピンの設定をしてください。詳細は architecture reference manual の I/O System 章を参照してください。
3. RTC busy ステータスを確認してください。安全な設計を確保するために、BACKUP_STATUS.RTC_BUSY = '0'になるまで待ってください [3]。
4. 読み出しステータスを確認してください。BACKUP_RTC_RW.READ が"0"でない場合は、READ ビットをクリアしてください (BACKUP_RTC_RW.READ = '0')。その後、安全な設計を確保するために、BACKUP_RTC_RW.READ = '0'になるまで待ってください [4]。
5. 校正波形の出力を更新するためには、WRITE ビットを設定してください。BACKUP_RTC_RW.WRITE = '1' (書き込み操作開始)を書き込んでください。
6. 波形周波数を 512Hz に設定してください: BACKUP_CAL_CTL.CAL_SEL = '0'を書き込む
7. 校正波形を有効にしてください: BACKUP_CAL_CTL.CAL_OUT = '1' (校正波形出力) を書き込むと、RTC_CAL ピンに WCO 波形が出力されます。
8. WRITE ビットをクリアしてください: BACKUP_RTC_RW.WRITE = '0'を書き込む (書き込み操作を終了)。
9. RTC レジスタ更新のために RTC busy ステータスを確認してください。

表 7 に、GPIO の RTC_CAL ピン出力に対する SDL の設定部のパラメータを示します。

GPIO 設定については、architecture reference manual の I/O System 章を参照してください。

表 7　　GPIO の RTC_CAL ピン出力

パラメータ	説明	値
user_led_port_pin_cfg.outVal	端子の出力状態を選択します。 0: 出力状態に影響なし、1: 出力状態を'0'に設定	0ul

(続く)

[3] 直前の操作を実行中は、次の校正操作は実行できません。
[4] READ ビットが設定されている間は、以降の書き込み操作は実行できません

表 7 (続き) GPIO の RTC_CAL ピン出力

パラメータ	説明	値
user_led_port_pin_cfg.driveMode	I/O ピンの GPIO 駆動モードを選択します。 0: アナログ High インピーダンス 1: 予約されているため使用しないでください 2: 抵抗プルアップ 3: 抵抗プルダウン 4: オープンドレイン、LOW 駆動 5: オープンドレイン、HIGH 駆動 6: ストロング駆動 7: 抵抗プルアップ/プルダウン 8: デジタル High-Z。入力バッファ ON。 9: 抵抗プルアップ入力バッファ ON。 10: 抵抗プルダウン入力バッファ ON。 11: オープンドレイン、LOW 駆動入力バッファ ON。 12: オープンドレイン、HIGH 駆動入力バッファ ON 13: ストロング駆動入力バッファ ON 14: 抵抗プルアップ/プルダウン入力バッファ ON	CY_GPIO_DM_STRONG_IN_OFF =6ul
user_led_port_pin_cfg.hsiom	RTC_CAL ピン経路の接続を設定します。	P1_2_SRSS_CAL_WAVE
user_led_port_pin_cfg.intEdge	IO ピンの IRQ をトリガするエッジを設定します。 0: ディセーブル、1: 立ち上りエッジ、2: 立ち下りエッジ、3: 両方	0ul
user_led_port_pin_cfg.intMask	IO ピンエッジ割込みをマスクします。 0: ピン割込み転送を無効にします 1: ピン割込み転送を有効にします	0ul
user_led_port_pin_cfg.vtrip	IO ピン入力バッファモードを選択します。 0: CMOS、1: TTL	0ul

(続く)

表 7 (続き) GPIO の RTC_CAL ピン出力

パラメータ	説明	値
user_led_port_pin_cfg.slewRate	IO ピンのスルーレートを選択します。 0: 高速スルーレート、1: 低速スルーレート	0ul
user_led_port_pin_cfg.driveSel	IO ピンの GPIO 駆動強度を設定します。 0: フル駆動強度 1: フル駆動強度 2: 1/2 駆動強度 3: 1/4 駆動強度	0ul

表 8 に RTC 校正波形出力への SDL の設定部のパラメータを、表 9 に関数を示します。

表 8 RTC 校正波形出力パラメータ

パラメータ	説明	値
CALIB_VALUE	絶対周波数の校正値。各ステップでは、毎時 128 tick が追加または削除されます。	0ul
CY_EN_RTC_CALIB_SIGN_NEGATIVE	0: 負の符号: パルスを除去します (1 秒をカウントするために、より多くのクロック tick が必要です) 1: 正の符号: パルスを追加します (1 秒をカウントするために必要なクロック tick 数が減ります)	0
CY_EN_RTC_CAL_SEL_CAL512	校正波出力信号を選択します 0: 512 Hz 波、校正設定に影響されません。 1: 予約済み 2: 2 Hz 波、校正設定の影響を含みます。 3: 1 Hz 波、校正設定の影響を含みます。	0

表 9 RTC 校正波形出力関数

関数	説明	値
Cy_Rtc_CalibrationControlEnable (uint8_t calib_val、cy_en_rtc_calib_sign_t calib_sign、cy_en_rtc_cal_sel_t cal_sel)	校正制御レジスタを設定し、校正波形出力を有効にします calib_val: 絶対周波数の校正値。 calib_sign: 校正符号 cal_sel: 校正波出力信号を選択します	CALIB_VALUE = 0ul, CY_EN_RTC_CALIB_SIGN_NEGATIVE=0, CY_EN_RTC_CAL_SEL_CAL512 = 0

2 動作概要

Code Listing 18 に、RTC 校正波形出力のプログラム例を示します。

以下の説明は、SDL のドライバ部のレジスタ表記を理解するのに役立ちます。

- **BACKUP**->un**CAL_CTL** レジスタは、register reference manual に記載されている BACKUP_CAL_CTL レジスタです。他のレジスタも同様に記述されます。

RTC_CAL 端子設定は CYT4DN シリーズの場合を示します。これらの RTC_CAL ピンの設定は、例えば Code Listing 21、Code Listing 28、および Code Listing 34 と同じです。

2 動作概要

Code Listing 21 RTC 校正波形出力のプログラム例

```
/* 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 表 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 表 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 ドライバ部の WCO 波形をキャプチャするプログラム例

Code Listing 22～Code Listing 27 に、ドライバ部の WCO 波形をキャプチャするプログラム例を示します。

Code Listing 22 ドライバ部の RTC 入力クロックソース設定のプログラム例

```
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 ドライバ部の同期ステータスを取得するプログラム例

```
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 ドライバ部の READ ビットの状態を確認するプログラム例

```
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 ドライバ部の READ ビットをクリアするプログラム例

```
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);
}
```

2 動作概要

Code Listing 26 ドライバ部にイネーブルを書き込むプログラム例

```
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 ドライバ部の校正制御イネーブルを設定するプログラム例

```
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 WCO と理想波形の誤差の測定

WCO 波形 (512Hz) を RTC_CAL の外部端子に出力してください。次に、外部の校正済み実験装置を用いて、校正波形と原子時計や校正済み実験装置などの理想波形との差を測定します。この装置は TRAVEO™ T2G デバイスとは別に設置する必要があります。2 つのクロック信号の差に基づいて ppm 値を計算してください。図 7 に、校正波形の測定プロセスを示します。

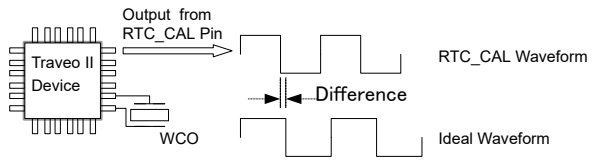

図 7 2 つの波形の差を測定

正の較正値'1'を設定して 1 時間ごとに 2×64 = 128 クロック tick を追加し、その時間をカウントするために必要な WCO クロック tick 数を減らします。校正値が'1'の時は、(2 × 64) / (32,768 × 60 × 60) = 1.085 ppm の補正を行います。6 ビットの校正値フィールドには最大 63 の値を保持できますが、較正カウンターは 1 時間ごとにリロードされるため、実際に使用できるのは最大 60 の値だけです。これにより、校正範囲は、± 60 × 1.085 ppm = ± 65.1 ppm になります。

CALIB_SIGN ビットで、修正の方向を指定できます。値'0'は、負の校正を表し、パルスが削除されるため、RTC 機能によるリアルタイムトラッキングが遅くなります。逆に、正の校正を適用する場合は'1'を使用します。

図 8 に、WCO を校正するフロー例を示します。

2 動作概要

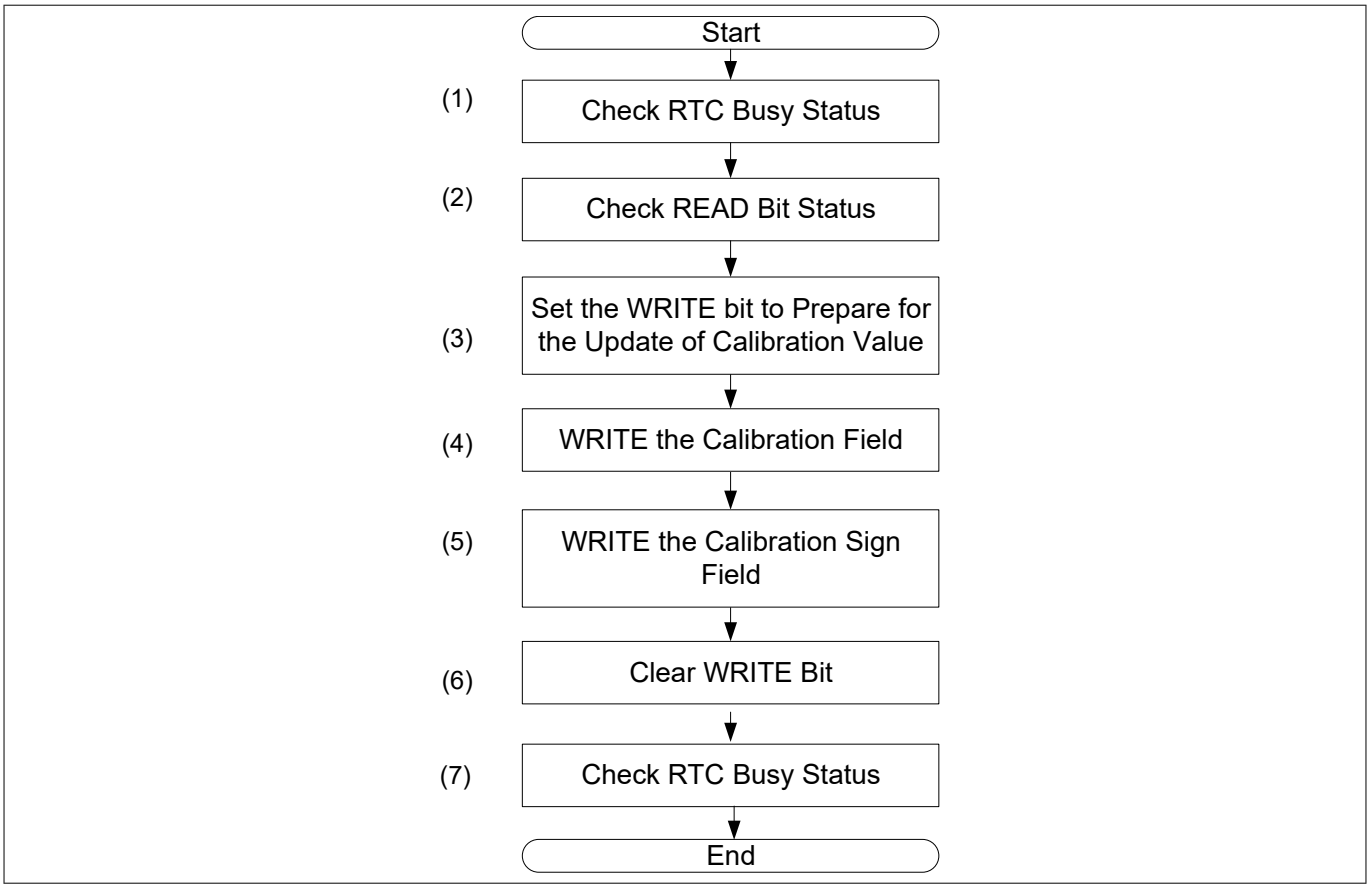

図 8　　WCO 校正のフロー例

以下の手順を実行する前に、WCO 波形のキャプチャで説明されているように、WCO 入力クロックと RTC の RTC_CAL ピン出力を設定する必要があります。

1. RTC busy ステータスを確認してください。安全な設計を確保するために、BACKUP_STATUS.RTC_BUSY = ‘0’になるまで待ってください [5]。
2. 読み出しステータスを確認してください。BACKUP_RTC_RW.READ が”0”でない場合は、読み出しビットをクリアしてください (BACKUP_RTC_RW.READ = ‘0’)。
   - その後、安全な設計を確保するため、BACKUP_RTC_RW.READ = ‘0’になるまで待ってください [6]。
3. 校正値の更新を準備するには、WRITE ビットを設定してください。
   - BACKUP_RTC_RW.WRITE = ‘1’ (書き込み操作開始)を書き込んでください。
4. 校正値フィールドを書き込んでください。
   - BACKUP_CAL_CTL.CALIB_VAL = "value"を書き込んでください。
   - この例では、+27.125 ppm の WCO 周波数誤差を調整されます。
   - 27.125 ppm/1.085 = 25
   - 校正値は‘25’です。
   - BACKUP_CAL_CTL.CALIB_VAL = ‘25’を書き込んでください。
5. 校正記号フィールドを書き込んでください。
   - BACKUP_CAL_CTL.CALIB_SIGN = ‘1’を書き込んでください。
   - この例では RTC クロックを早める必要があるため、’1’を書き込んでください。

5 直前の操作が実行されている間は、次の校正操作は実行できません。
6 READ ビットが設定されている間は、次の書き込み操作は実行できません。

6. WRITE ビットをクリアしてください。
   - BACKUP_RTC_RW.WRITE = '0' (書き込み操作終了) を書き込んでください。
7. RTC レジスタ更新のために RTC busy ステータスを確認してください。

校正補正は、発振器分周器のパルス数を 1 時間ごとに追加または削除することによって (この例のように) 実行され、それぞれクロックが速くなったり遅くなったりします。校正開始後、校正補正は 59 分 59 秒から 1 時間ごとに実行され、2 × BACKUP_CAL_CTL.CALIB_VAL の調整が完了するまで、30 秒ごとに 64 tick として補正が適用されます。

RTC の校正値を設定するためのパラメータを表 10 に示し、SDL 設定部の関数を表 11 に示します。

表 10 RTC 校正値パラメータ

パラメータ	説明	値
CALIB_VALUE	絶対周波数の校正値。各ステップでは、毎時 128 tick が追加または削除されます。	25ul
CY_EN_RTC_CALIB_SIGN_POSITIVE	0: 負の符号: パルスを除去します (1 秒をカウントするために、より多くのクロック tick が必要になります) 1: 正の符号: パルスを追加します (1 秒をカウントするために必要なクロック tick 数が減ります)	1
CY_EN_RTC_CAL_SEL_CAL512	校正波出力信号を選択します 0: 512 Hz 波、校正設定により影響されません。 1: 予約済み。 2: 2 Hz 波、校正設定による影響を含みます。 3: 1 Hz 波、校正設定による影響を含みます。	0

表 11 RTC 校正値関数

関数	説明	値
Cy_Rtc_CalibrationControlEnable (uint8_t calib_val、cy_en_rtc_calib_sign_t calib_sign、cy_en_rtc_cal_sel_t cal_sel)	校正制御レジスタを設定し、校正波形出力を有効にします calib_val: 絶対周波数の校正値。 calib_sign: 校正符号 cal_sel: 校正波出力信号の選択します	CALIB_VALUE = 25ul, CY_EN_RTC_CALIB_SIGN_POSITIVE= 0, CY_EN_RTC_CAL_SEL_CAL512 = 0

Code Listing 28 に、RTC の校正値を設定するプログラム例を示します。

Code Listing 28 校正値を設定するプログラム例

```
/* 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 表 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 表 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 ドライバ部の WCO 波形を測定するプログラム例

Code Listing 29～Code Listing 34 に、ドライバ部の WCO 波形を測定するプログラム例を示します。

Code Listing 29 ドライバ部に設定された RTC 入力クロックソースを測定するプログラム例

```
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 ドライバ部の同期状態を取得するプログラム例

```
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 ドライバ部の READ ビットの状態を確認するプログラム例

```
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 ドライバ部の READ ビットをクリアするプログラム例

```
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 ドライバ部にイネーブルを書き込むプログラム例

```
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 ドライバ部にキャリブレーション制御イネーブルを設定するプログラム例

```
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 校正後の WCO 周波数精度の確認

WCO 波形 (512 Hz) は、校正値(BACKUP_CAL_CTL.CALIB_VAL や BACKUP_CAL_CTL.CALIB_SIGN) が変更されても影響を受けません。しかし、1 Hz と 2 Hz の校正波形は、現在の校正値の影響を受けます。図 9 に示すように、512 Hz 分周器は校正ロジックを経由しません。したがって、1 Hz または 2 Hz の波形を測定することで、WCO 周波数の精度を確認できます。

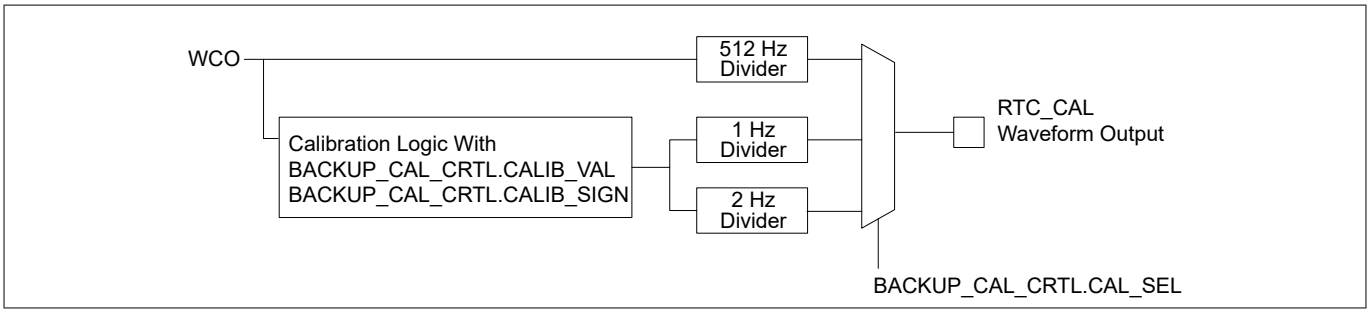

図 9 クロック選択ブロックダイヤグラム

図 10 に校正後の WCO 周波数精度を確認するフロー例を示します。

2 動作概要

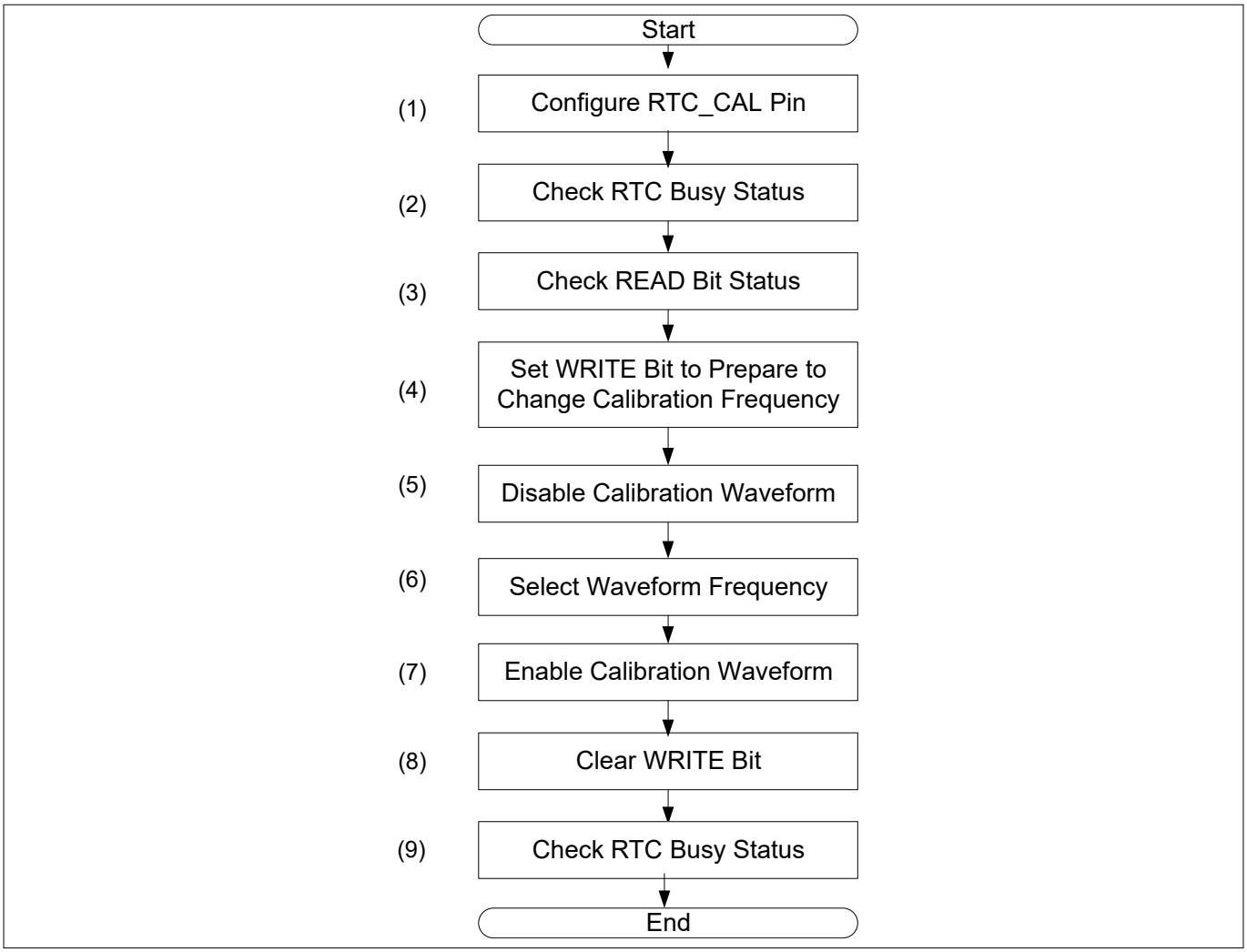

図 10　　校正後の WCO 周波数精度を確認するフロー例

下記の手順は、1Hz の波形の WCO 周波数を出力して、校正後の精度を確認する方法を説明します。

1. RTC_CAL ピンの設定をしてください。詳細は architecture reference manual の I/O Port 章を参照してください。
2. RTC busy ステータスを確認してください。安全な設計を確保するために、BACKUP_STATUS.RTC_BUSY = ‘0’になるまで待ってください [7]。
3. 読み出しステータスを確認してください。BACKUP_RTC_RW.READ が”0”でない場合は、READ ビットをクリアしてください (BACKUP_RTC_RW.READ = ‘0’)。
   - その後、安全な設計を確保するために、BACKUP_RTC_RW.READ = ‘0’になるまで待ってください [8]。
4. WRITE ビットを設定して、校正周波数を変更する準備をしてください。
   - BACKUP_RTC_RW.WRITE = ‘1’ (書き込み操作開始)を書き込んでください。
5. 校正波形を無効にしてください。
   - BACKUP_CAL_CTL.CAL_OUT = ‘0’ (校正波形出力を無効) を書き込んでください。
6. 波形周波数を設定してください。
   - BACKUP_CAL_CTL.CAL_SEL = ‘3’(1 Hz) を書き込んでください。
7. 校正波形を有効にしてください。
   - BACKUP_CAL_CTL.CAL_OUT = ‘1’ (校正波形出力を有効) を書き込んでください。

7 直前の動作が実行中には、次の校正動作を実行できません。
8 READ ビットが設定されている間は、次の書き込み動作は実行できません

8. WRITE ビットをクリアしてください。
   - BACKUP_RTC_RW.WRITE = '0' (書込み操作終了) を書き込んでください。
9. RTC レジスタ更新のために RTC busy ステータスを確認してください。

そうすると、RTC_CAL ピンで 1 Hz 波形が観測できるようになります。これらの波形は、BACKUP_CAL_CTL.CALIB_VAL によって校正されたものです。外部機器 (例えば、校正されたオシロスコープ) を使用して波形を測定して、校正の効果を確認してください。

校正波形の出力周波数を変更するため、表 12 にはパラメータを、表 13 には SDL 設定部の関数を示します。

表 12 校正波形の出力周波数を変更する RTC

パラメータ	説明	値
CALIB_VALUE	絶対周波数の校正値。各ステップでは、毎時 128 tick が追加または削除されます。	25ul
CY_EN_RTC_CALIB_SIGN_POSITIVE	0: 負の符号: パルスを除去 (1 秒をカウントするために、より多くのクロック tick が必要になります) 1: 正の符号: パルスを追加 (1 秒をカウントするために必要なクロック tick 数が減ります)	1
CY_EN_RTC_CAL_SEL_CAL1	校正波出力信号を選択 0: 512 Hz 波、校正設定に影響されません。 1: 予約済み。 2: 2 Hz 波、校正設定の影響を含みます。 3: 1 Hz 波、校正設定の影響を含みます。	3

表 13 校正波形関数の出力周波数を変更する RTC

関数	説明	値
Cy_Rtc_CalibrationControlDisable (void)	校正波形出力を無効にします	-
Cy_Rtc_CalibrationControlEnable (uint8_t calib_val、cy_en_rtc_calib_sign_t calib_sign、cy_en_rtc_cal_sel_t cal_sel)	校正制御レジスタを設定し、校正波形出力を有効にします calib_val: 絶対周波数の校正値。 calib_sign: 校正符号 cal_sel: 校正波出力信号を選択	CALIB_VALUE = 25ul、CY_EN_RTC_CALIB_SIGN_POSITIVE = 1、CY_EN_RTC_CAL_SEL_CAL1 = 3

Code Listing 35 に、校正波形の出力周波数を変更するプログラム例を示します。

Code Listing 35 校正波形の出力周波数を変更するプログラム例

```
/* 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 表 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 表 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. */
```

2 動作概要

```
    /* 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 ドライバ部の校正後の WCO 周波数精度確認プログラム例

Code Listing 36 ～Code Listing 41 に、ドライバ部の校正後に WCO 周波数精度を確認するためのプログラム例を示します。

Code Listing 36 ドライバ部の同期ステータスを取得するプログラム例

```
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 ドライバ部の READ ビットの状態を確認するプログラム例

```
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 ドライバ部の READ ビットをクリアするプログラム例

```
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 ドライバ部にイネーブルを書き込むプログラム例

```
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 ドライバ部の校正波形出力を無効にするプログラム例

```
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 ドライバ部の校正制御イネーブルを設定するプログラム例

```
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 用語集

用語	説明
RTC 機能	RTC ロジックの機能です。
RTC 値	RTC 機能により刻まれている時刻です。詳細は architecture reference manual の Real-Time Clock 章の Real-Time Clock を参照してください。
校正機能	RTC 値の周波数誤差を補正する機能。
整数値	Binary Coded Decimal (BCD) ではない整数。
うるう年補正	RTC には、日付フィールド (月の日数) の自動的なうるう年補正機能が実装されています。詳細は architecture reference manual の Real-Time Clock 章の Real-Time Clock を参照してください。
ILO	Internal Low-speed Oscillator (内部低速発振器) は、TRAVEO™ T2G デバイスに組み込まれた発振器です。詳細については、architecture reference manual の Clocking System 章を参照してください。
CLK_LF	Low Frequency Clock (低周波クロック) をソースクロックとして選択できます。詳細については、architecture reference manual の Clocking System 章を参照してください。
WCO	Watch Crystal Oscillator (時計水晶発振器) は、WCO_IN と WCO_OUT 端子に接続した外部水晶発振器です。詳細については、architecture reference manual の Clocking System 章を参照してください。
LPECO	Low-power external crystal oscillator (低電力外部水晶発振器) は、LPECO_IN と LPECO_OUT 端子に接続した外部水晶発振器です。詳細については、architecture reference manual の Clocking System 章を参照してください。
校正波形	RTC_CAL 端子から出力される信号で、RTC を校正します。詳細は architecture reference manual の Real-Time Clock 章の WCO/LPECO Calibration を参照してください。
理想波形	正確な周波数の波形です。
時計表示	自動車内にあるドットマトリックスやセグメントディスプレイで現在時刻を表示する機器です。
ユーザシステム	ユーザが開発した TRAVEO™ T2G デバイスを搭載した電子基板/システムです。
ユーザファームウェア	ユーザが開発した CPU で実行されるソフトウェアです。
ローカル変数	ユーザファームウェアによって使用される変数です。
ユーザバッファ	ユーザファームウェアによって確保された RAM 上のローカル変数です。
AHB-Lite バスインタフェース	CPU サブシステムと周辺機能の間をつなぐバスインタフェースです。詳細は architecture reference manual の Introduction 章の Top-Level Architecture を参照してください。

3 用語集

用語	説明
ユーザレジスタ	ユーザファームウェアがアクセスできるレジスタ。読み書き動作が可能です。しかし、常に最新のデータを反映しているとは限りません。詳細は architecture reference manual の Real-Time Clock 章の Real-Time Clock を参照してください。
RTC レジスタ	ハードウェアからアクセス可能なレジスタ。ユーザーファームウェアによる読み書き動作はできません。常に最新のデータを反映しています。詳細は architecture reference manual の Real-Time Clock 章の Real-Time Clock を参照してください。
RTC フィールド	ユーザレジスタおよび RTC レジスタが持っている RTC 値のビットフィールドです。詳細は architecture reference manual の Real-Time Clock 章の Real-Time Clock を参照してください。
アラームフィールド	ユーザレジスタおよび RTC レジスタが持っているアラーム値のビットフィールドです。詳細は architecture reference manual の Real-Time Clock 章の Real-Time Clock を参照してください。
Config フィールド	ユーザレジスタおよび RTC レジスタが持っている設定ビットのビットフィールドです。詳細は architecture reference manual の Real-Time Clock 章の Real-Time Clock を参照してください。
WCO デバイスとロジック	WCO デバイスは外部水晶発振器です。WCO ロジックは、TRAVEO™ T2G デバイスに内蔵している回路です。
LPECO デバイスとロジック	LPECO デバイスは外部水晶発振器です。LPECO ロジックは、TRAVEO™ T2G デバイスに内蔵している回路です。
パワーモード/デバイスパワーモード	TRAVEO™ T2G ファミリは消費電流を抑えるために、いくつかのパワーモードを持っています。詳細は Architecture TRM の Device Power Modes 章を参照してください。
アラーム機能	RTC フィールドとアラームフィールドの値が一致した時に、割込みを発生させます。詳細は architecture reference manual の Real-Time Clock 章の Alarm Feature を参照してください。
インターバル時間	定期的な割込みを発生させる時間です。
パワーオンリセット	パワーオンシーケンス後のリセット動作です。詳細は architecture reference manual の Rest System 章を参照してください。
ウェイクアップ	スタンバイモード, Sleep, Low-Power Sleep, DeepSleep, Hibernate からユーザファームウェアを再スタートさせる動作です。詳細は architecture reference manual の Device Power Modes 章を参照してください。

4 関連資料

以下は TRAVEO™ T2G ファミリのデータシートおよびテクニカルリファレンスマニュアルです。これらのドキュメントの入手についてはテクニカルサポートに連絡してください。

[1] デバイスデータシート

- 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] テクニカルリファレンスマニュアル

- ボディコントローラ Entry ファミリ
  - 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)
- ボディコントローラ High ファミリ
  - 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 ファミリ
  - 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)
- クラスタ Entry ファミリ
  - TRAVEO™ T2G automotive cluster entry family architecture technical reference manual
  - TRAVEO™ T2G automotive cluster entry registers technical reference manual for CYT2CL

5 その他の参考資料

インフィニオンは、さまざまな周辺機器にアクセスするためのサンプルソフトウェアとして、スタートアップを含む Sample Driver Library (SDL) を提供しています。SDL は、公式の AUTOSAR 製品でカバーされていないドライバを提供するため、お客様にとってのリファレンスとしても機能します。SDL は自動車規格に適合していないため、製造目的では使用できません。このアプリケーションノートのプログラムコードは SDL の一部です。SDL の入手については、テクニカルサポートに連絡してください。

改訂履歴

版数	発行日	変更内容
**	2019-02-27	これは英語版 002-20190 Rev. **を翻訳した日本語版 002-26056 Rev. **です。
*A	2019-03-28	これは英語版 002-20190 Rev. *A を翻訳した日本語版 002-26056 Rev. *A です。
B	2019-09-13	これは英語版 002-20190 Rev. *B を翻訳した日本語版 002-26056 Rev. *B です。
C	2020-05-29	これは英語版 002-20190 Rev. *C を翻訳した日本語版 002-26056 Rev. *C です。英語版の改訂内容: Changed target parts number (CYT2/CYT4 series). Added target parts number (CYT3 series).
英語版*D	2020-12-15	本版は英語版のみの発行です。英語版の改訂内容: Added flowchart and example codes. Moved to Infineon Template
英語版*E	2022-09-14	本版は英語版のみの発行です。英語版の改訂内容: Update code example in driver part. Added the procedure to check the RTC busy status for RTC register update.
*D	2024-05-13	これは英語版 002-20190 Rev. *F を翻訳した日本語版 002-26056 Rev. *D です。英語版の改訂内容: Template update; no content update.
*E	2025-07-25	これは英語版 002-20190 Rev. *G を翻訳した日本語版 002-26056 Rev. *E です。英語版の改訂内容: Added to CYT6BJ

Trademarks

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

Edition 2025-07-25

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

重要事項

本手引書に記載された情報は、本製品の使用に関する手引きとして提供されるものであり、いかなる場合も、本製品における特定の機能性能や品質について保証するものではありません。本製品の使用の前に、当該手引書の受領者は実際の使用環境の下であらゆる本製品の機能及びその他本手引書に記された一切の技術的情報について確認する義務が有ります。インフィニオンテクノロジーズはここに当該手引書内で記される情報につき、第三者の知的所有権の不侵害の保証を含むがこれに限らず、あらゆる種類の一切の保証および責任を否定いたします。

本文書に含まれるデータは、技術的訓練を受けた従業員のみを対象としています。本製品の対象用途への適合性、およびこれら用途に関連して本文書に記載された製品情報の完全性についての評価は、お客様の技術部門の責任にて実施してください。

警告事項

技術的要件に伴い、製品には危険物質が含まれる可能性があります。当該種別の詳細については、インフィニオンの最寄りの営業所までお問い合わせください。

インフィニオンの正式代表者が署名した書面を通じ、インフィニオンによる明示の承認が存在する場合を除き、インフィニオンの製品は、当該製品の障害またはその使用に関する一切の結果が、合理的に人的傷害を招く恐れのある一切の用途に使用することはできないこと予めご了承ください。