AN0047
Application Note

AT32 ERTC User Guide

Introduction

The AT32 MCU enhanced real-time clock (ERTC) provides a calendar and clock function. This application note introduces ERTC basic functions and application examples.

Note: The corresponding code in this application note is developed on the basis of V2.x.x BSP provided by Artery. For other versions of BSP, please pay attention to the differences in usage.

Applicable products:

Part number	AT32F415xx
	AT32F421xx
	AT32F425xx
	AT32F435xx
	AT32F437xx
	AT32L021xx
	AT32F405xx
	AT32F423xx

Contents

List of tables

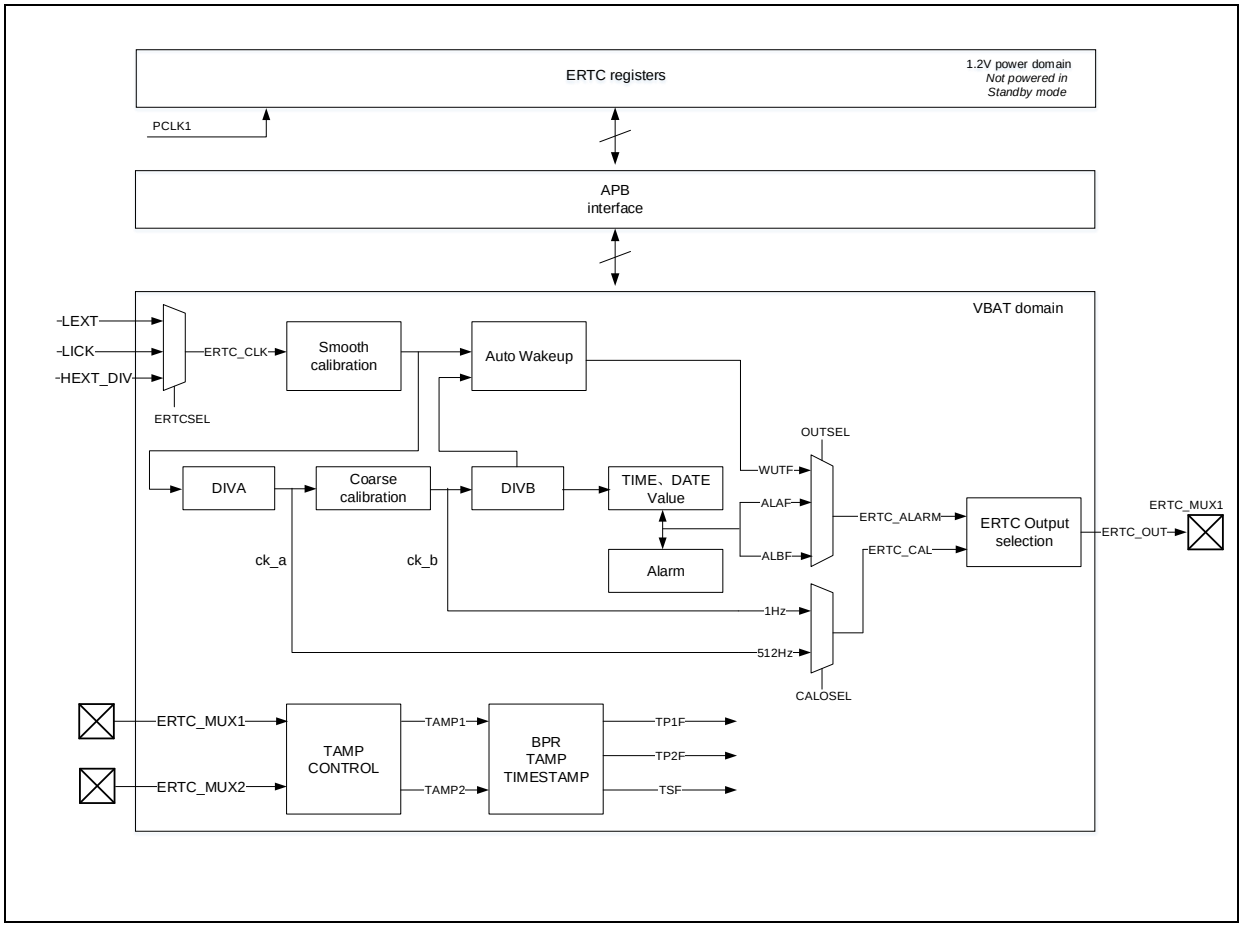
List of figures

1 ERTC introduction

The RTC module is in battery powered domain, which means that it keeps running and free from the influence of system reset and VDD power off as long as VBAT is powered. ERTC has the following features:

- Real-time calendar function: year, month, day, hour, minute, second
- Compensations for 28-, 29- (leap year), 30-, and 31-day months are performed (when the year register is a multiple of 4, it represents a leap year)
- Clock function: Clock A, Clock B
- Periodic auto-wakeup
- Tamper detection
- Supports smooth and coarse calibration

Figure 1. ERTC block diagram



2 ERTC function overview

2.1 ERTC functions of each series

ERTC functions of each series are basically the same with program compatibility, except that advanced functions are not available for some series.

Table 1. ERTC functions of each series

Series	CLK A	CLK B	Periodic wakeup	TAMP 1	TAMP2	Smooth CAL	Coarse CAL	Time stamp	Ref. clock detection	BPR register (Qty.)
AT32F415xx	√	√	√	√	×	√	√	√	√	20
AT32F421xx	√	×	×	√	×	√	×	√	√	5
AT32F425xx	√	×	√	√	×	√	×	√	√	5
AT32F435xx	√	√	√	√	√	√	√	√	√	20
AT32F437xx	√	√	√	√	√	√	√	√	√	20
AT32L021xx	√	×	√	√	×	√	×	√	√	5
AT32F405xx	√	√	√	√	√	√	×	√	√	20
AT32F423xx	√	√	√	√	√	√	×	√	√	20

√: This function is available and is the same for other series.

×: This function is not supported.

The clock cycles required for ERTC register read and write operations are listed below.

Table 2. Clock cycles required for ERTC register read and write operations

Series	Write	Unit	Read	Unit
AT32F415xx	2	SYSCLK	2	SYSCLK
AT32F421xx	4	ERTC CLK	11	PCLK1
AT32F425xx	4	ERTC CLK	11	PCLK1
AT32F435xx	4	ERTC CLK	11	PCLK1
AT32F437xx	4	ERTC CLK	11	PCLK1
AT32L021xx	15	PCLK1	15	PCLK1
AT32F405xx	15	PCLK1	15	PCLK1
AT32F423xx	15	PCLK1	15	PCLK1

2.2 Register

Register write protection

All ERTC registers are write-protected after a power-on reset. Write access to the ERTC registers (except the ERTC_STS[14:8], ERTC_TAMP and ERTC_BPRx registers) can be enabled by unlocking such protection mechanism.

Unlock procedures:

1) Enable PWC peripheral clock

```
crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);
```

2) Unlock write protection of battery powered domain

```
pwc_battery_powered_domain_access(TRUE);
```

3) Write 0xCA and 0x53 to ERTC_WP register in sequence to unlock write protection.

Writing an incorrect key will activate the write protection again.

```
ertc_write_protect_disable();
```

4) Configure ERTC registers

Table 3 lists the write protection status of ERTC registers and conditions for write operation.

Table 3. ERTC registers

Register	ERTC_WP enabled	Whether to enter initialization mode	Others
ERTC_TIME	Y	Y	-
ERTC_DATE	Y	Y	-
ERTC_CTRL	Y	Y (bit 7/6/4 only)	-
ERTC_STS	Y, except [14:8]	-	-
ERTC_DIV	Y	Y	-
ERTC_WAT	Y	N	Configurable when WATWF=1
ERTC_CCAL	Y	Y	-
ERTC_ALA	Y	N	Configurable when ALAWF=1
ERTC_ALB	Y	N	Configurable when ALBWF=1
ERTC_WP	-	-	-
ERTC_SBS	-	-	-
ERTC_TADJ	Y	N	Configurable when TADJF=0
ERTC_TSTM	-	-	-
ERTC_TSDT	-	-	-
ERTC_TSSBS	-	-	-
ERTC_SCAL	Y	N	Configurable when CALUPDF=0
ERTC_TAMP	N	N	-
ERTC_ALASBS	Y	N	Configurable when ALAWF=1
ERTC_ALBSBS	Y	N	Configurable when ALBWF=1
ERTC_BPRx	N	N	-

Register reset

ERTC registers are in the battery powered domain, and the battery powered domain software reset is performed by setting the BPDRST bit in the CRM_BPDC register. It is also possible to reset by writing the default value through the library function.

ERTC reset function:

Battery powered domain reset

```
void crm_battery_powered_domain_reset(confirm_state new_state)
```

Set all ERTC registers to default value

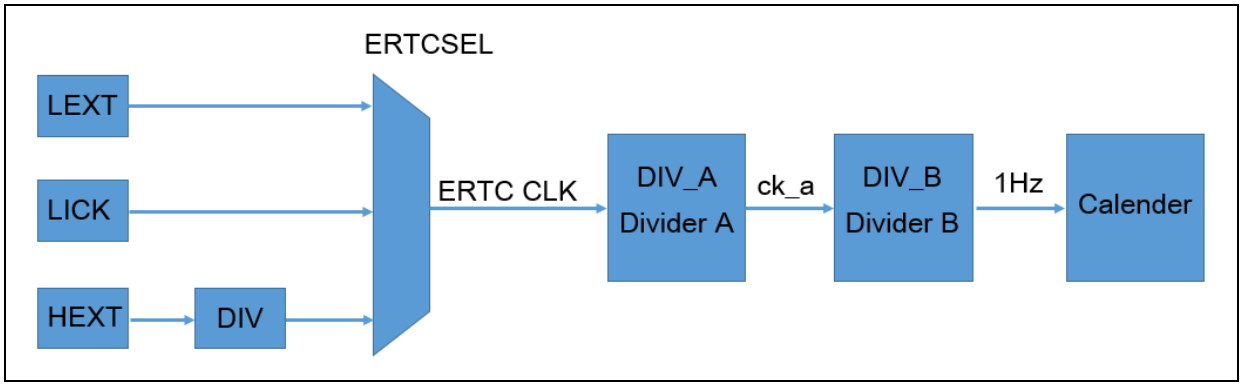
```
error_status ertc_reset(void)
```

2.3 Clock

Clock source selection

The selected ERTC clock source is output to divider A and divider B, and finally generates a 1 Hz clock to update the calendar.

Figure 2. ERTC clock structure



ERTC clock source is selected from:

- LEXT: External low-speed crystal oscillator (generally 32.768 kHz)
- LICK: Internal low-speed crystal oscillator (typical value: 40 kHz, range: 30~60 kHz; refer to the datasheet of each series for details)
- HEXT_DIV: Divided external high-speed crystal oscillator (dividers of each series are listed below)

Table 4. HEXT pre-scaler dividers

Series	Pre-scaler divider
AT32F415xx	Fixed 128
AT32F421xx	Fixed 32
AT32F425xx	Fixed 32
AT32F435xx	Configurable, 2~31
AT32F437xx	Configurable, 2~31
AT32L021xx	Fixed 32

Table 5. Comparison of clock sources

Clock source	Frequency	Advantages	Shortage
LEXT	32.768kHz	Maximum accuracy, working in battery powered mode and low-power mode	One crystal oscillator is required, with a higher cost and more PCB wiring areas
LICK	Typical 40kHz Range:30kHz~60kHz	Working in low-power mode, saving one crystal oscillator and reducing PCB wiring area	Low time accuracy
HEXT	Main crystal oscillator frequency	Relatively high accuracy (depending on the main crystal oscillator), saving one crystal oscillator and reducing PCB wiring area	It does not work in battery powered mode and low-power mode

ERTC clock source setting function:

Enable the selected clock source

```
void crm_clock_source_enable(crm_clock_source_type source, confirm_state new_state)
```

Select ERTC clock

```
void crm_ertc_clock_select(crm_ertc_clock_type value)
```

Enable ERTC clock

```
void crm_ertc_clock_enable(confirm_state new_state)
```

Prescaler

After being divided by prescaler A and B, a 1 Hz clock is generated according the following formula:

$$f_{1Hz} = \frac{f_{ERTC\_CLK}}{(DIVB + 1) \times (DIVA + 1)}$$

It is recommended to set DIVA=127 to minimize power consumption.

Table 6. Example of divider setting

Clock source	Frequency	DIV_A	DIV_B	Calendar clock
LEXT	32.768kHz	127	256	1Hz
LICK	Typical 40kHz	124	319	1Hz
HEXT	8MHz, divided by 32 (for AT32F421)	124	1999	1Hz

Set ERTC prescaler divider

```
error_status ertc_divider_set(uint16_t div_a, uint16_t div_b)
```

Example of ERTC clock initialization:

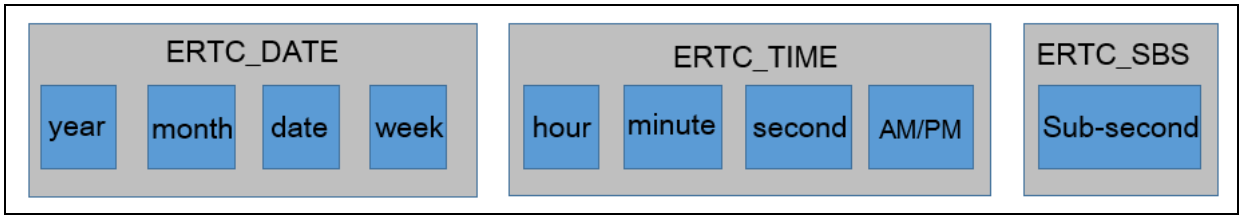
```
/* Enable LEXT */
crm_clock_source_enable(CRM_CLOCK_SOURCE_LEXT, TRUE);

/* Wait LEXT to become stable */
```

```
while(crm_flag_get(CRM_LEXT_STABLE_FLAG) == RESET)
{
}

/* Select LEXT as ERTC clock source */
crm_ertc_clock_select(CRM_ERTC_CLOCK_LEXT);

/* Enable ERTC clock */
crm_ertc_clock_enable(TRUE);

/* Set ERTC prescaler DIVA=127 and DIV_B=255 */
/* ertc second(1hz) = ertc_clk / (div_a + 1) * (div_b + 1) */
ertc_divider_set(127, 255);
```

2.4 Calendar

ERTC calendar format includes the year, month, date, weekday, hour, minute, second and subsecond, as shown below.

Figure 3. Calendar format



Time format

Both 12-hour and 24-hour formats are available. If the 24-hour format is set, AM/PM field is meaningless. Set the time format before calendar initialization.

Time format setting function:

```
error_status ertc_hour_mode_set(ertc_hour_mode_set_type mode)
```

For example, set 24-hour format:

```
ertc_hour_mode_set(ERTC_HOUR_MODE_24);
```

Calendar initialization

Set the calendar time by setting ERTC_DATE and ERTC_TIME registers.

Calendar time setting function:

```
error_status ertc_date_set(uint8_t year, uint8_t month, uint8_t date, uint8_t week)
error_status ertc_time_set(uint8_t hour, uint8_t min, uint8_t sec, ertc_am_pm_type ampm)
```

For example, set the time to 2020-05-01 12:30:00 Saturday

```
ertc_date_set(21, 5, 1, 6);
ertc_time_set(12, 30, 0, ERTC_24H);
```

Read calendar

Read the calendar by reading ERTC_DATE, ERTC_TIME and ERTC_SBS registers. The ERTC offers two different ways to read the calendar, that is, synchronous read (DREN=0) and asynchronous read (DREN=1).

- DREN=0: ERTC synchronizes calendar values to shadow registers (ERTC_DATE, ERTC_TIME and ERTC_SBS) every two ERTC_CLK cycles, and set UPDF=1 after synchronization is completed. Reading the low-order register will lock the high-order register value until the ERTC_DATE register is read, which guarantees values read from ERTC_SBS, ERTC_TIME and ERTC_DATE registers are at the same time.

  For example, reading ERTC_SBS register will lock values in ERTC_TIME and ERTC_DATE registers.
- DREN=1: ERTC perform direct read access to the ERTC clock and calendar located in the battery powered domain, avoiding the occurrence of errors caused by time synchronization. In this mode, UPDF flag is cleared by hardware. To ensure the data is correct when reading clock and calendar, the software must read the clock and calendar registers twice, and compare the results of two read operations. If the result is not aligned, read again until that the results of two read accesses are consistent.

The synchronous read mode is recommended for most applications to simplify the program execution.

Wait synchronous function (wait for UPDF=1)

```
error_status ertc_wait_update(void)
```

Read mode setting function:

```
void ertc_direct_read_enable(confirm_state new_state)
```

For example, set synchronous read mode:

```
void ertc_direct_read_enable(DISABLE);
```

Calendar read function:

```
void ertc_calendar_get(ertc_time_type* time)
```

The struct “ertc_time_type” contains the following parameters:

- year
- month
- day
- hour
- min
- sec
- week
- ampm: AM/PM, effective in 12-hour format

Read subsecond

The subsecond value is the prescaler DIV_B downcounter value.

When DIV_B=255, one subsecond value represents 1/(255 + 1) second(s).

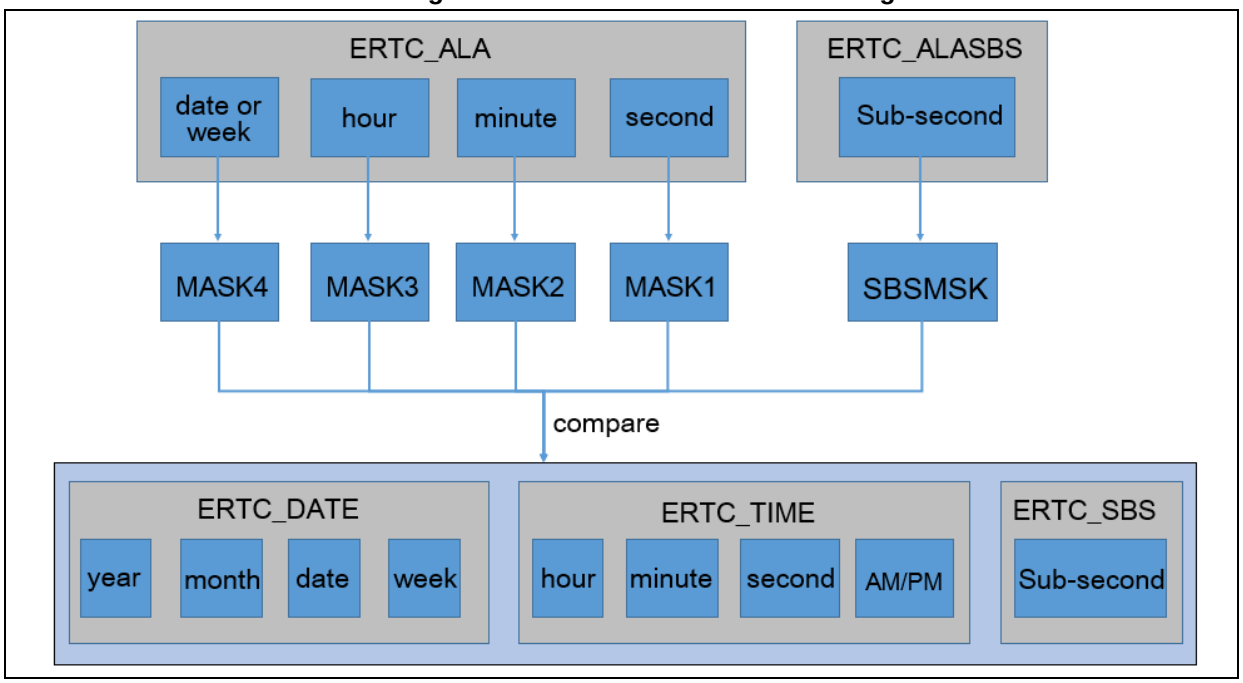
Subsecond read function:

uint32_t ertc_sub_second_get(void)

2.5 Alarm

ERTC contains identical alarm A and alarm B, and the alarm value is programmed by ERTC_ALASBS/ERTC_ALA (ERTC_ALBSBS/ERTC_ALB). When the programmed alarm value matches the calendar value, an alarm event is generated if an alarm clock is enabled. The MASKx bit can be used to selectively mask calendar fields. The calendar fields, which are masked, are not allocated with an alarm clock.

Figure 4. Calendar and alarm matching



Alarm A is used for demonstration in the following content.

Alarm format selection:

- WKSEL=0 (in the ERTC_ALA register): date, hour, minute, second, subsecond
- WKSEL=1 (in the ERTC_ALA register): week, hour, minute, second, subsecond

Alarm fields (date/week, hour, minute, second, subsecond) can be masked by setting the MASK bit.

- MASK4 = 1: alarm irrelevant to date/week
- MASK3 = 1: alarm irrelevant to hour
- MASK2 = 1: alarm irrelevant to minute
- MASK1 = 1: alarm irrelevant to second

When WKSEL=0, set the alarm to 12:30:10 on the 15th.

Table 7. Example of mask setting

	MASK4	MASK3	MASK2	MASK1	Result
1	1	0	0	0	Alarm triggered at xx-12:30:10 xx: date
2	0	1	0	0	Alarm triggered at 15-xx:30:10 xx: hour
3	0	0	1	0	Alarm triggered at 15-12:xx:10 xx: minute
4	0	0	0	1	Alarm triggered at 15-12:30:xx xx: second

Set the SBSMSK bit in the ERTC_ALASBS register to mask subsecond:

- SBSMSK = 0: Not match subsecond, alarm irrelevant to subsecond
- SBSMSK = 1: Match SBS[0] only
- SBSMSK = 2: Match SBS[1:0] only
- SBSMSK = 3: Match SBS[2:0] only
- ...
- SBSMSK = 14: Match SBS[13:0] only
- SBSMSK = 15: Match SBS[14:0]

When DIV_A = 127 and DIV_B=255 (subsecond), the alarm is only triggered by subsecond.

Table 8. Example of subsecond mask setting

SBSMSK	Result
0	Alarm irrelevant to subsecond
1	Match SBS[0] only, alarm triggered in 1/128 second
2	Match SBS[1:0] only, alarm triggered in 1/64 second
3	Match SBS[2:0] only, alarm triggered in 1/32 second
4	Match SBS[3:0] only, alarm triggered in 1/16 second
5	Match SBS[4:0] only, alarm triggered in 1/8 second
6	Match SBS[5:0] only, alarm triggered in 1/4 second
7	Match SBS[6:0] only, alarm triggered in 1/2 second
8	Match SBS[7:0] only, alarm triggered in 1 second
9	Match SBS[8:0] only, alarm triggered in 1 second
10	Match SBS[9:0] only, alarm triggered in 1 second
11	Match SBS[10:0] only, alarm triggered in 1 second
12	Match SBS[11:0] only, alarm triggered in 1 second
13	Match SBS[12:0] only, alarm triggered in 1 second
14	Match SBS[13:0] only, alarm triggered in 1 second
15	Match SBS[14:0] only, alarm triggered in 1 second

Alarm related functions

Mask date/week, hour, minute, second

```
void ertc_alarm_mask_set(ertc_alarm_type alarm_x, uint32_t mask)
```

Select date/week format

```
void ertc_alarm_week_date_select(ertc_alarm_type alarm_x, ertc_week_date_select_type wk)
```

Set alarm value: date/week, hour, minute, second, AM/PM

```
void ertc_alarm_set(ertc_alarm_type alarm_x, uint8_t week_date, uint8_t hour, uint8_t min,
uint8_t sec, ertc_am_pm_type ampm)
```

Set alarm subsecond and mask

```
void ertc_alarm_sub_second_set(ertc_alarm_type alarm_x, uint32_t value,
ertc_alarm_sbs_mask_type mask)
```

Alarm interrupt enable

```
error_status ertc_alarm_enable(ertc_alarm_type alarm_x, confirm_state new_state)
```

Get current alarm value

```
void ertc_alarm_get(ertc_alarm_type alarm_x, ertc_alarm_value_type* alarm)
```

Get current alarm subsecond

```
uint32_t ertc_alarm_sub_second_get(ertc_alarm_type alarm_x)
```

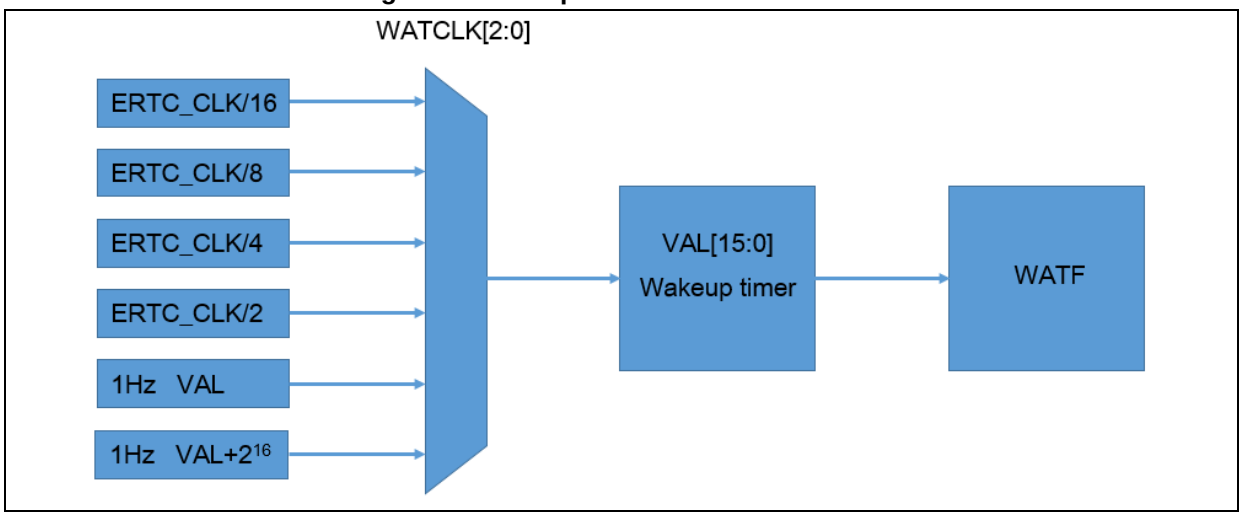
2.6 Periodic automatic wakeup

Periodic automatic wakeup unit is used to wake up ERTC from low power consumption modes automatically. The period is programmed with the VAL[15:0] bit. When the wakeup counter value drops from the VAL to zero, the WATF bit is set to 1, and a wakeup event is generated, with the wakeup counter being reloaded with the VAL value.

The clock source can be set through the WATCLK[2:0] bit:

- 000: ERTC_CLK/16
- 001: ERTC_CLK/8
- 010: ERTC_CLK/4
- 011: ERTC_CLK/2
- 10x: 1Hz
- 11x: 1Hz, wakeup counter value increases 2^{16}, wakeup time = WAT+2^{16} + 1.

Figure 5. Wakeup timer clock source selection



When WATCLK[2:0]= 11x, the maximum wakeup time=65535+2^{16}+1=131072 seconds if the calendar clock is 1 Hz.

If the calendar clock is <1Hz (increase the value of prescaler DIV_B), a longer wakeup time can be obtained.

Periodic automatic wakeup related functions

Wakeup counter clock source selection

```
void ertc_wakeup_clock_set(ertc_wakeup_clock_type clock)
```

Set wakeup counter value

```
void ertc_wakeup_counter_set(uint32_t counter)
```

Get wakeup counter value

```
uint16_t ertc_wakeup_counter_get(void)
```

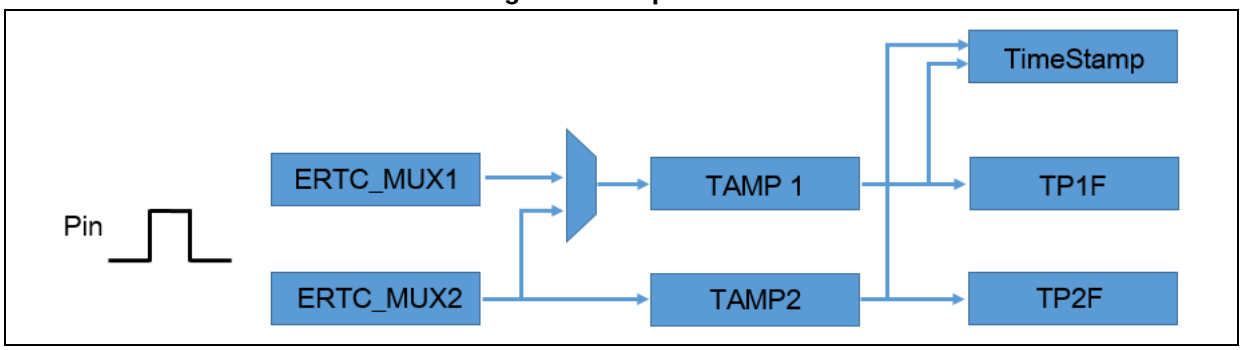
Wakeup timer enable

```
error_status ertc_wakeup_enable(confirm_state new_state)
```

2.7 Tamper detection

ERTC have two tamper detection modes: TAMP1 and TAMP2. When a tamper event occurs, the ERTC_BPRx register will be cleared; a time stamp is generated when a tamper event occurs.

Figure 6. Tamper detection



TAMP1 is selected from:

- ERTC_MUX1: Pin 1, usually PC13
- ERTC_MUX2: Pin 2, usually PA0

TAMP2 is the fixed ERTC_MUX2 (usually PA0).

Tamper detection includes edge detection and level detection.

- Edge detection: Tamper detection is triggered when a valid edge is detected, including rising edge and falling edge.
- Level detection: Tamper detection is triggered when the valid level reaches the programmed time, including high level and low level.

For the edge detection, configure and enable the valid edge; for the level detection, configure the following parameters:

Sampling frequency

Set the TPFREQ register to set the tamper detection frequency as below:

- ERTC_CLK/32768
- ERTC_CLK/16384
- ERTC_CLK/8192
- ERTC_CLK/4096
- ERTC_CLK/2048
- ERTC_CLK/1024
- ERTC_CLK/512
- ERTC_CLK/256

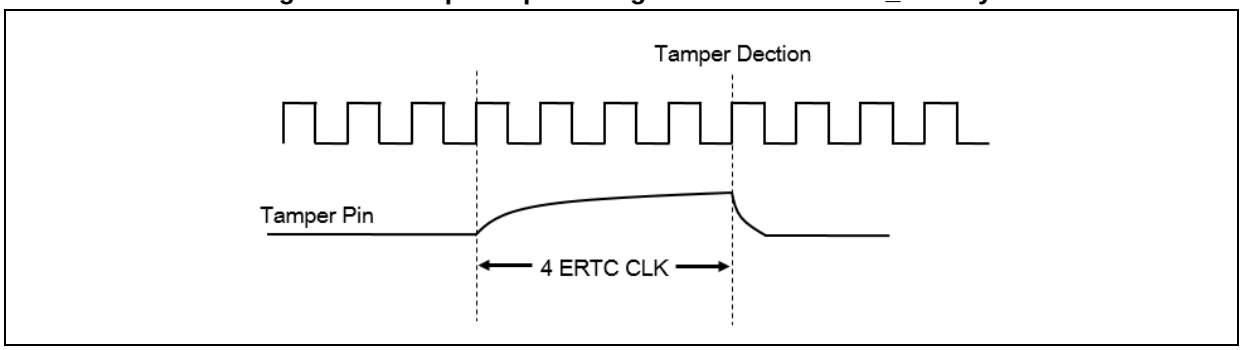
For example, when ERTC_CLK=32768 Hz and the detection frequency= ERTC_CLK/256, the tamper detection frequency is 32768/256 = 128Hz.

Tamper detection pull-up

Enable or disable tamper pull-up function by setting the TPPU bit. When the tamper pull-up resistor is enabled, the tamper detection pre-charge time is set through the TPPR bit, as shown below:

- 1 x ERTC_CLK cycle
- 2 x ERTC_CLK cycles
- 4 x ERTC_CLK cycles
- 8 x ERTC_CLK cycles

Figure 7. Example of pre-charge time = four ERTC_CLK cycles



Tamper detection filter

Set the tamper detection filter time through the TPFLT bit, as shown below:

- No filter; tamper is detected after 1 effective sample
- Tamper is detected after 2 consecutive samples
- Tamper is detected after 4 consecutive samples
- Tamper is detected after 8 consecutive samples

Tamper detection related functions

TAMP1 pin selection

```
void ertc_tamper_1_pin_select(ertc_pin_select_type pin)
```

Tamper detection pull-up enable

```
void ertc_tamper_pull_up_enable(confirm_state new_state)
```

Pull-up pre-charge time set

```
void ertc_tamper_precharge_set(ertc_tamper_precharge_type precharge)
```

Filter time set

```
void ertc_tamper_filter_set(ertc_tamper_filter_type filter)
```

Tamper detection frequency set

```
void ertc_tamper_detect_freq_set(ertc_tamper_detect_freq_type freq)
```

Tamper detection valid edge set

```
void ertc_tamper_valid_edge_set(ertc_tamper_select_type tamper_x,
ertc_tamper_valid_edge_type trigger)
```

Save time stamp in case of a tamper event

```
void ertc_tamper_timestamp_enable(confirm_state new_state)
```

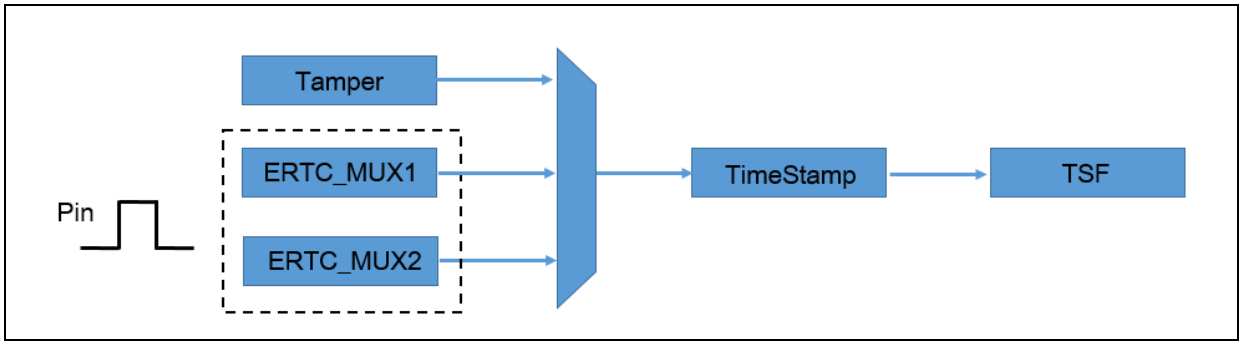
Tamper detection enable

```
void ertc_tamper_enable(ertc_tamper_select_type tamper_x, confirm_state new_state)
```

2.8 Time stamp

When time stamp event is detected on the tamper pin (valid edge is detected), the current calendar value will be stored to the time stamp register.

Figure 8. Time stamp detection



Usage of time stamp:

- Independent usage: select one of the following pins for detection

  ERTC_MUX1: Pin 1, usually PC13

  ERTC_MUX2: Pin 2, usually PA0

- Save time stamp when a tamper event occurs (in this mode, the tamper detection should be configured properly in advance)

When the time stamp is used independently, set rising edge or falling edge detection, which determines the trigger of tamper detection.

Time stamp overflow

When the time stamp is detected, set TSF=1; at this point, TSOF=1 when a time stamp event occurs again, while the time stamp register is not updated and remains the value triggered at the first time.

Time stamp related functions

Time stamp pin selection

```
void ertc_timestamp_pin_select(ertc_pin_select_type pin)
```

Edge detection set

```
void ertc_timestamp_valid_edge_set(ertc_timestamp_valid_edge_type edge)
```

Time stamp enable

```
void ertc_timestamp_enable(confirm_state new_state)
```

Get time stamp time

```
void ertc_timestamp_get(ertc_time_type* time)
```

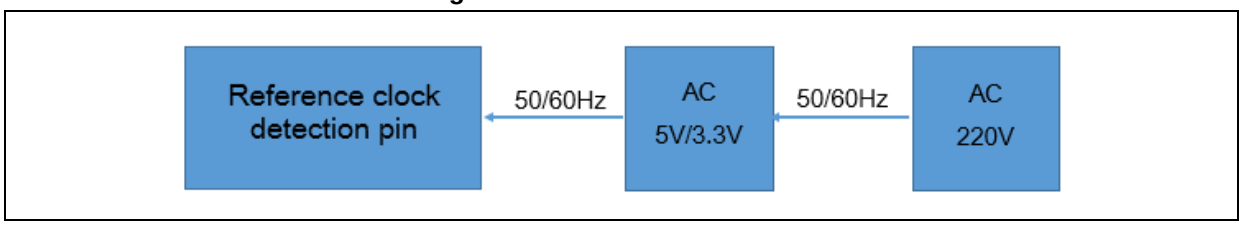
Get time stamp subsecond

```
uint32_t ertc_timestamp_sub_second_get(void)
```

2.9 Reference clock detection

To guarantee the accuracy of calendar for long-time operation, ERTC supports clock synchronization function (not available in low-power mode) to use a more accurate reference clock (50 Hz or 60 Hz mains supply) for the 1 Hz calendar clock calibration.

Figure 9. Reference clock detection



After the reference clock detection is enabled, detect the reference clock edge in the first seven ck_a cycles every time updating the calendar value. If a valid edge is detected, this edge is used to update calendar value (update second clock), and then take three ck_a cycles to detect the reference clock edge. The value of prescaler A is reloaded every time the reference clock edge is detected, which aligns the internal 1Hz calendar clock with the reference clock edge. In case of any slight offset of the 1 Hz clock, it is recommended to use a more accurate reference clock to tune the 1 Hz clock until it is aligned with the reference clock edge. If no reference clock edge is detected, ERTC updates calendar with the original clock source.

Note: After the reference clock is enabled, set DIVA and DIVB to their reset value (0x7F and 0xFF, respectively); the synchronization function cannot be enabled at the same time as the coarse calibration function.

Reference clock detection enable function

```
error_status ertc_refer_clock_detect_enable(confirm_state new_state)
```

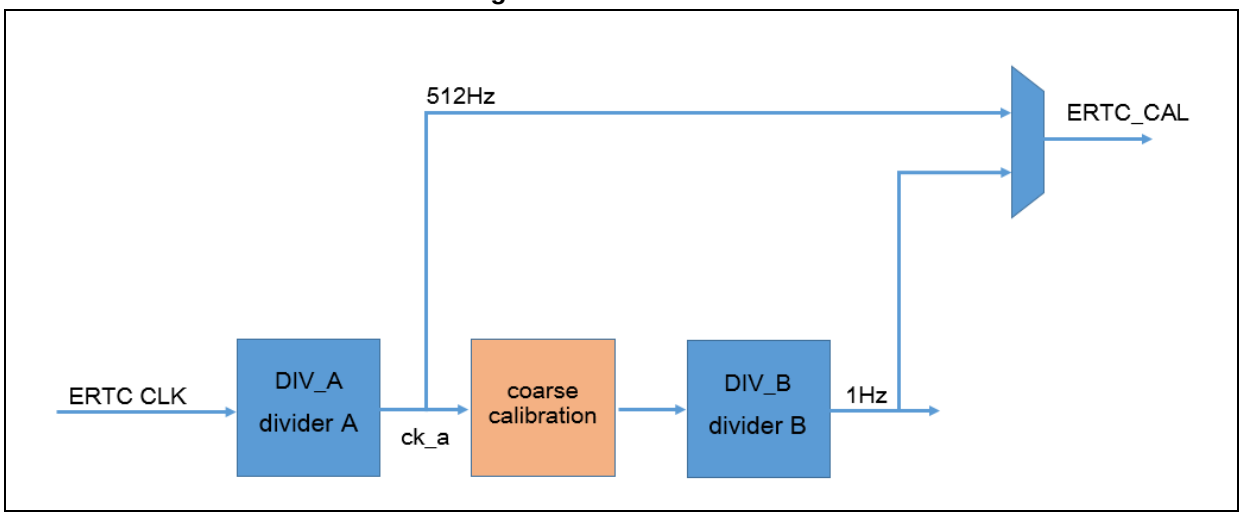
2.10 Calibration

ERTC supports smooth calibration and coarse calibration that cannot be enabled at the same time.

Coarse calibration

Coarse calibration advances or delays calendar value updating by increasing or reducing ck_a cycles.

Figure 10. Coarse calibration



CALDIR=0: In the first 2xCALVAL minutes of 64 minutes, insert two ck_a cycles every minute (about 15360 x ck_a cycles), that is, updating calendar in advance.

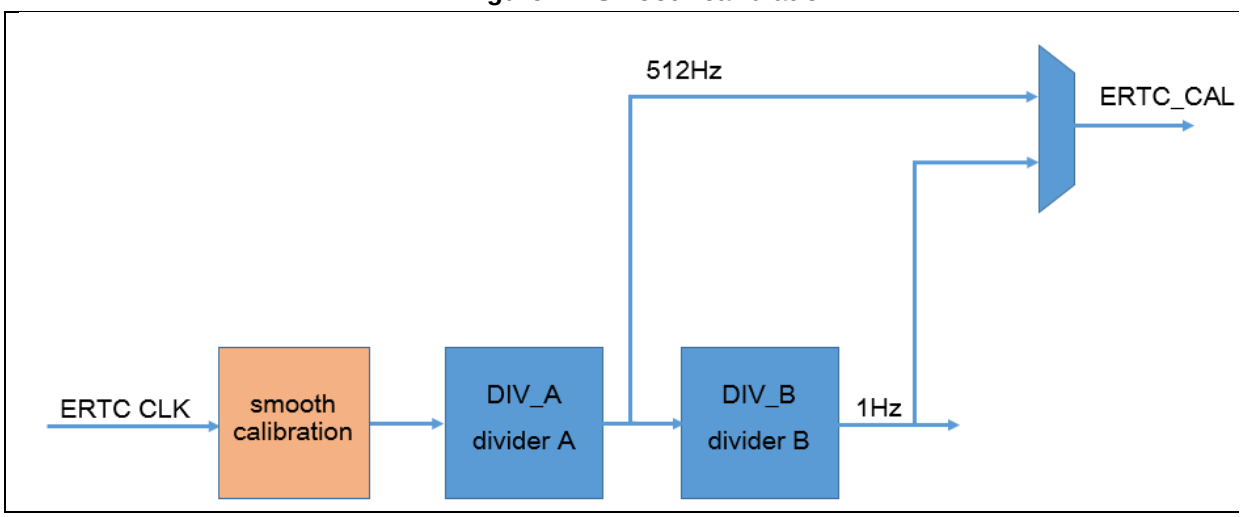
CALDIR=1: In the first 2xCALVAL minutes of 64 minutes, ignore one ck_a cycle every minute (about 15360 x ck_a cycles), that is, delaying calendar updating.

Note: Set DIVA=6 (at least) in coarse calibration mode.

Smooth calibration

Smooth digital calibration has a higher and well-distributed performance than the coarse digital calibration. The calibration is performed by increasing or decreasing ERTC_CLK in an evenly manner.

Figure 11. Smooth calibration



The smooth calibration period is about 2^{20}xERTC_CLK cycles (32s) when ERTC_CLK is 32.768 kHz. The value of DEC[8:0] specifies the number of pulses to be masked during 2^{20}xERTC_CLK cycles. A maximum of 511 pulses can be removed. When ADD=1, 512 pulses can be inserted during the 2^{20}xERTC_CLK cycles. When DEC[8: 0] and ADD are used together, a deviation ranging from -511 to +512 ERTC_CLK cycles can be added during the 2^{20}xERTC_CLK cycles.

Valid calibration frequency F_{SCAL}:

$$F_{SCAL} = F_{ERTC\_CLK} \times [\,1 + \frac{ADD \times 512 - DEC}{2^{20} + DEC - ADD \times 512}\,]$$

When the divider A is less than 3, the calibration operates as if ADD was equal to 0. The divider B value should be reduced so that each second is accelerated by 8 ERTC_CLK cycles, which means that 256 ERTC_CLK cycles are added every 32 seconds. When DEC[8: 0] and ADD are sued together, a deviation ranging from -255 to +256 ERTC_CLK cycles can be added during the 2^{20} ERTC_CLK cycles.

At this point, the effective calibrated frequency F_{SCAL}:

$$F_{SCAL} = F_{ERTC\_CLK} \times [\,1 + \frac{256 - DEC}{2^{20} + DEC - 256}\,]$$

It is also possible to select 8 or 16-second digital calibration period through the CAL8 and CAL16 bits. The 8-second period takes priority over 16-second. In other words, when both 8-second and 16-second are enabled, 8-second calibration period prevails.

Calibration related functions

Smooth calibration configuration and enable

```
error_status ertc_smooth_calibration_config(ertc_smooth_cal_period_type period,
ertc_smooth_cal_clk_add_type clk_add, uint32_t clk_dec)
```

Coarse calibration set

```
error_status ertc_coarse_calibration_set(ertc_cal_direction_type dir, uint32_t value)
```

Coarse calibration enable

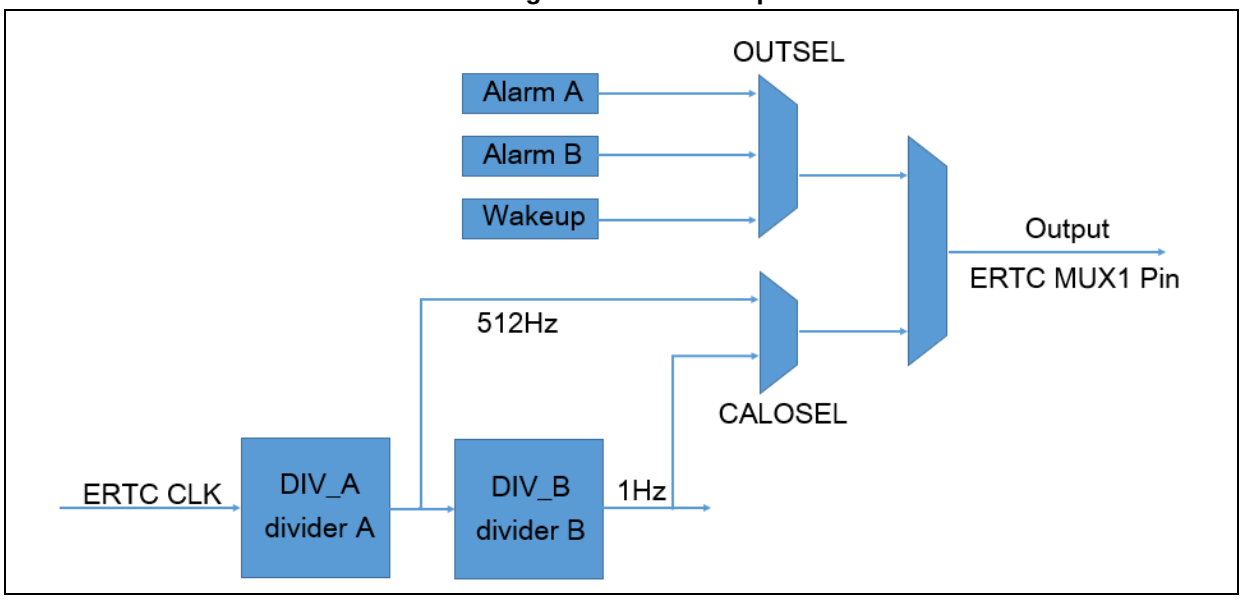
```
error_status ertc_coarse_calibration_enable(confirm_state new_state)
```

2.11 Multiplexed function output

ERTC provides a set of multiplexed function output for the following events:

- Calibration: output 512 Hz and output 1 Hz
- Event: clock A, clock B, wakeup event

Figure 12. Event output



Usually, ERTC MUX1 is PC13.

In event output mode (alarm clock A, alarm clock B and wakeup event), it is possible to select output type (open-drain or push-pull) with the OUTTYPE bit, and output polarity with the OUTP bit.

Event output related functions

Event output set (alarm clock A, alarm clock B and wakeup event)

```
void ertc_output_set(ertc_output_source_type source, ertc_output_polarity_type polarity,
ertc_output_type type)
```

Calibration output select (512 Hz, 1 Hz)

```
void ertc_cal_output_select(ertc_cal_output_select_type output)
```

Calibration output enable

```
void ertc_cal_output_enable(confirm_state new_state)
```

2.12 Battery powered domain data register

ERTC provides 20 x 32-bit battery powered domain data registers that can save data in battery powered mode and reset by battery powered domain reset or tamper event instead of system reset.

Related functions

Write battery powered domain data register

```
void ertc_bpr_data_write(ertc_dt_type dt, uint32_t data)
```

Read battery powered domain data register

```
uint32_t ertc_bpr_data_read(ertc_dt_type dt)
```

2.13 Interrupt

ERTC can be woken up by alarm clock A/B or periodic wakeup event. To enable an ERTC interrupt, configure as follows:

- Configure the EXINT line corresponding to ERTC interrupts as an interrupt mode and enable it, and select a rising edge
- Enable a NVIC channel corresponding to ERTC interrupts
- Enable the corresponding ERTC interrupt control bit

Table 9 lists the ERTC clock sources, events and interrupts that are able to wakeup low-power modes.

Table 9. ERTC low-power mode wakeup

Clock source	Event	Wake up Sleep	Wake up Deepsleep	Wake up Standby
HEXT	Alarm clock A	√	×	×
	Alarm clock B	√	×	×
	Periodic automatic wakeup	√	×	×
	Time stamp	√	×	×

	Tamper event	√	×	×
LICK	Alarm clock A	√	√	√
	Alarm clock B	√	√	√
	Periodic automatic wakeup	√	√	√
	Time stamp	√	√	√
	Tamper event	√	√	√
LEXT	Alarm clock A	√	√	√
	Alarm clock B	√	√	√
	Periodic automatic wakeup	√	√	√
	Time stamp	√	√	√
	Tamper event	√	√	√

Table 10. Interrupt control bits

Interrupt event	Event flag	Interrupt enable bit
Alarm clock A	ALAF	ALAIEN
Alarm clock B	ALBF	ALBIEN
Periodic automatic wakeup	WATF	WATIEN
Time stamp	TSF	TSIEN
Tamper event	TP1F/TP2F	TPIEN

Table 11. EXINT lines corresponding to ERTC interrupts

MCU Series	Alarm clock A	Alarm clock B	Periodic automatic wakeup	Time stamp	Tamper detection 1	Tamper detection 2
AT32F415xx	17	17	22	21	21	-
AT32F421xx	17	-	-	19	19	-
AT32F425xx	17	-	20	19	19	-
AT32F435xx	17	17	22	21	21	21
AT32F437xx	17	17	22	21	21	21
AT32L021xx	17	-	20	19	19	-

Table 12. Interrupt vector number

MCU Series	Alarm clock A Alarm clock B	Periodic automatic wakeup	Time stamp Tamper detection 1 Tamper detection 2
AT32F415xx	ERTCAlarm_IRQn	ERTC_WKUP_IRQn	TAMP_STAMP_IRQn
AT32F421xx	ERTC_IRQn	-	ERTC_IRQn
AT32F425xx	ERTC_IRQn	ERTC_IRQn	ERTC_IRQn
AT32F435xx	ERTCAlarm_IRQn	ERTC_WKUP_IRQn	TAMP_STAMP_IRQn
AT32F437xx	ERTCAlarm_IRQn	ERTC_WKUP_IRQn	TAMP_STAMP_IRQn
AT32L021xx	ERTC_IRQn	ERTC_IRQn	ERTC_IRQn

Table 13. Interrupt vector and function

Interrupt vector	Interrupt function
ERTCAlarm_IRQn	ERTCAlarm_IRQHandler
ERTC_IRQn	ERTC_IRQHandler
ERTC_WKUP_IRQn	ERTC_IRQHandler
TAMP_STAMP_IRQn	ERTCAlarm_IRQHandler

Interrupt and event related functions

Interrupt enable

```
void ertc_interrupt_enable(uint32_t source, confirm_state new_state)
```

Get interrupt status

```
flag_status ertc_interrupt_get(uint32_t source)
```

Get flag

```
flag_status ertc_flag_get(uint32_t flag);
```

Flag clear

```
void ertc_flag_clear(uint32_t flag);
```

Example of interrupt configuration (alarm clock A of AT32F435 series)

```
/* Configure EXINT line */
exint_default_para_init(&exint_init_struct);
exint_init_struct.line_enable = TRUE; // enable
exint_init_struct.line_mode = EXINT_LINE_INTERRUPUT; // interrupt mode
exint_init_struct.line_select = EXINT_LINE_17;//EXINT line 17
exint_init_struct.line_polarity = EXINT_TRIGGER_RISING_EDGE;// trigger rising edge
exint_init(&exint_init_struct);

/* Enable alarm interrupt NVIC vector: ERTCAlarm_IRQn */
nvic_irq_enable(ERTCAlarm_IRQn, 0, 1);

/* Enable alarm interrupt */
ertc_interrupt_enable(ERTC_ALA_INT, TRUE);

/* Enable alarm */
ertc_alarm_enable(ERTC_ALA, TRUE);
```

3 Application case 1: Read and write battery powered domain data register

3.1 Introduction

Write and read battery powered domain data registers (ERTC_BPRx).

3.2 Resources

1) Hardware:

   AT-START BOARD of the corresponding model

2) Software:

   project\at_start_f4xx\examples\ertc\bpr_domain

Note: All projects are built around keil 5. If users want to use them in other compiling environments, please refer to AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates (such as IAR6/7, keil 4/5) for a simple change.

3.3 Software design

1) Configuration process
   - Enable PWC clock
   - Enable write protection of battery powered domain
   - Check whether the battery powered domain data is correct. If it is correct, skip initialization operation; otherwise, initialize ERTC and write data to the battery powered domain.

2) Code
   - Main function code

```
int main(void)
{
  uint32_t temp = 0;
  ertc_time_type time;

  /* Initialize system clock */
  system_clock_config();

  /* Configure NVIC priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* Initialize serial port */
  uart_init(115200);

  printf("\r\nertc bpr domain example\r\n\r\n");

  /* Enable PWC clock */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

  /* Enable battery powered domain write protection */
  pwc_battery_powered_domain_access(TRUE);
```

```
  /* Check whether the battery powered domain data is correct. If it is incorrect, perform initialization.
*/
  if(bpr_reg_check() == FALSE)
  {
    printf("bpr reg => reset\r\n\r\n");

    /* Initialize ERTC */
    ertc_config();

    /* Write data to battery powered domain */
    bpr_reg_write();
  }
  else
  {
    printf("bpr reg => none reset\r\n\r\n");

    /* Wait for ERTC register synchronization, requiring maximum two ERTC_CLK cycles */
    ertc_wait_update();
  }

  while(1)
  {
    /* Get current calendar value */
    ertc_calendar_get(&time);

    /* Time is updated if the second clock value is different from the previous */
    if(temp != time.sec)
    {
      temp = time.sec;

      /* Print: Y-M-D */
      printf("%02d-%02d-%02d ",time.year, time.month, time.day);

      /* Print: H:M:S */
      printf("%02d:%02d:%02d\r\n",time.hour, time.min, time.sec);
    }
  }
}
```

3.4 Test result

- Check the print information through the serial port assistant on PC;
- Print “bpr reg => none reset” if the register data is correct;
- Print “bpr reg => reset” if the register data is correct;
- Print the calendar information in the main function every second.

4 Application case 2: Calendar and alarm clock

4.1 Introduction

Demonstrate the application of calendar and alarm clock.

4.2 Resources

1) Hardware:

   AT-START BOARD of the corresponding model

2) Software:

   project\at_start_f4xx\examples\ertc\calendar

Note: All projects are built around keil 5. If users want to use them in other compiling environments, please refer to AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates (such as IAR6/7, keil 4/5) for a simple change.

4.3 Software design

1) Configuration process
   - Enable PWC clock
   - Enable write protection of battery powered domain
   - Check whether the calendar is initialized. If it is initialized properly, skip initialization operation; otherwise, initialize the calendar and alarm clock.
   - Print the calendar information in the main function every second.
   - Alarm clock event is generated at 21-05-01 12:00:10.
2) Code
   - Main function code

```
int main(void)
{
  exint_init_type exint_init_struct;
  ertc_time_type time;
  uint32_t temp = 0;

  /*Initialize system clock */
  system_clock_config();

  /* Configure NVIC priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* Initialize ATSTART board */
  at32_board_init();

  /* Initialize serial port */
  uart_init(115200);

  /* Enable PWC clock */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);
```

```
/* Enable battery powered domain write protection */
pwc_battery_powered_domain_access(TRUE);

if (ertc_bpr_data_read(ERTC_DT1) != 0x1234)
{
  /* Print information */
  printf("ertc has not been initialized\r\n\r\n");

  /* ERTC initialization */
  ertc_config();
}
else
{
  /* Print information */
  printf("ertc has been initialized\r\n\r\n");

  /* Wait for ERTC register synchronization, requiring maximum two ERTC_CLK cycles */
  ertc_wait_update();

  /* Clear alarm clock flag */
  ertc_flag_clear(ERTC_ALAF_FLAG);

  /* Clear exint set flag */
  exint_flag_clear(EXINT_LINE_17);
}

/* Display the current ertc time and alarm clock time */
ertc_time_show();
ertc_alarm_show();

printf("\r\n");

/* Alarm clock initialization */
exint_default_para_init(&exint_init_struct);
exint_init_struct.line_enable     = TRUE;
exint_init_struct.line_mode       = EXINT_LINE_INTERRUPUT;
exint_init_struct.line_select     = EXINT_LINE_17;
exint_init_struct.line_polarity = EXINT_TRIGGER_RISING_EDGE;
exint_init(&exint_init_struct);

/* Enable alarm clock interrupt */
nvic_irq_enable(ERTCAlarm_IRQn, 0, 1);

while(1)
{
```

```
      /* Get current calendar */
      ertc_calendar_get(&time);

      if(temp != time.sec)
      {
        temp = time.sec;

        /* Print: Y-M-D */
        printf("%02d-%02d-%02d ",time.year, time.month, time.day);

        /* Print: H:M:S */
        printf("%02d:%02d:%02d\r\n",time.hour, time.min, time.sec);
      }
    }
}
```

■ ERTC initialization ertc_config function code

```
void ertc_config(void)
{
  /* Enable PWC clock */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

  /* Enable battery powered domain write protection */
  pwc_battery_powered_domain_access(TRUE);

  /* Reset battery powered domain registers */
  crm_battery_powered_domain_reset(TRUE);
  crm_battery_powered_domain_reset(FALSE);

#if defined (ERTC_CLOCK_SOURCE_LICK)
  /* Enable LICK clock */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_LICK, TRUE);

  /* Wait for LICK clock to become stable */
  while(crm_flag_get(CRM_LICK_STABLE_FLAG) == RESET)
  {
  }

  /* Select ERTC clock source */
  crm_ertc_clock_select(CRM_ERTC_CLOCK_LICK);

  /* ertc second(1hz) = ertc_clk(lick) / (ertc_clk_div_a + 1) * (ertc_clk_div_b + 1) */
  ertc_clk_div_b = 255;
  ertc_clk_div_a = 127;
#elif defined (ERTC_CLOCK_SOURCE_LEXT)
  /* Enable LEXT clock */
```

```
  crm_clock_source_enable(CRM_CLOCK_SOURCE_LEXT, TRUE);

  /* Wait for LEXT clock to become stable */
  while(crm_flag_get(CRM_LEXT_STABLE_FLAG) == RESET)
  {
  }

  /* Select ERTC clock source */
  crm_ertc_clock_select(CRM_ERTC_CLOCK_LEXT);

  /* ertc second(1hz) = ertc_clk / (ertc_clk_div_a + 1) * (ertc_clk_div_b + 1) */
  ertc_clk_div_b = 255;
  ertc_clk_div_a = 127;
#endif

  /* Enable ERTC clock */
  crm_ertc_clock_enable(TRUE);

  /* Reset all ERTC registers */
  ertc_reset();

  /* Wait for ERTC register synchronization, requiring maximum two ERTC_CLK cycles */
  ertc_wait_update();

  /* Configure ERTC divider */
  ertc_divider_set(ertc_clk_div_a, ertc_clk_div_b);

  /* Set ERTC clock in 24-hour format */
  ertc_hour_mode_set(ERTC_HOUR_MODE_24);

  /* Set date: 2021-05-01 */
  ertc_date_set(21, 5, 1, 5);

  /* Set time: 12:00:00 */
  ertc_time_set(12, 0, 0, ERTC_AM);

  /* Set alarm clock 12:00:10 */
  ertc_alarm_mask_set(ERTC_ALA, ERTC_ALARM_MASK_DATE_WEEK);
  ertc_alarm_week_date_select(ERTC_ALA, ERTC_SLECT_DATE);
  ertc_alarm_set(ERTC_ALA, 1, 12, 0, 10, ERTC_AM);

  /* Enable alarm clock interrupt */
  ertc_interrupt_enable(ERTC_ALA_INT, TRUE);

  /* Enable alarm clock */
  ertc_alarm_enable(ERTC_ALA, TRUE);
```

```
    ertc_flag_clear(ERTC_ALAF_FLAG);

    /* Indicate ERTC initialization */
    ertc_bpr_data_write(ERTC_DT1, 0x1234);
}
```

- Alarm clock interrupt function code

```
void ERTCAlarm_IRQHandler(void)
{
    if(ertc_flag_get(ERTC_ALAF_FLAG) != RESET)
    {
        /* Display alarm clock */
        ertc_alarm_show();

        at32_led_on(LED2);

        /* Clear alarm clock flag */
        ertc_flag_clear(ERTC_ALAF_FLAG);

        /* Clear EXINT flag */
        exint_flag_clear(EXINT_LINE_17);
    }
}
```

4.4 Test result

- Check the print information through the serial port assistant on PC;
- Print the calendar information in the main function every second;
- Alarm clock event is generated at 21-05-01 12:00:10.

5 Application case 3: Use LICK clock and calibration

5.1 Introduction

Select LICK as ERTC clock, and use the timer to measure LICK frequency, which is used to adjust ERTC divider for time calibration within a specified period.

5.2 Resources

1) Hardware:

   AT-START BOARD of the corresponding model

2) Software:

   project\at_start_f4xx\examples\ertc\lick_calibration

Note: All projects are built around keil 5. If users want to use them in other compiling environments, please refer to AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates (such as IAR6/7, keil 4/5) for a simple change.

5.3 Software design

1) Configuration process
   - Initialize ERTC
   - Configure the timer used for LICK frequency measurement
   - Re-configure ERTC divider according to the measured LICK frequency

2) Code
   - Main function code

```
int main(void)
{
  ertc_time_type time;
  uint32_t temp = 0;

  /* Initialize system clock    */
  system_clock_config();

  /* Configure NVIC priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* Initialize AT START board */
  at32_board_init();

  /* Initialize serial port */
  uart_init(115200);

  /* ERTC initialization */
  ertc_config();

  /* Measure LICK frequency */
  lick_freq = lick_frequency_get();
```

```
  /* Configure ERTC divider */
  /* ertc second(1hz) = ertc_clk(lick) / (div_a + 1) * (div_b + 1) */
  ertc_divider_set(127, (lick_freq / 128) - 1);

  printf("lick_freq = %d\r\n", lick_freq);
  printf("div_a     = %d\r\n", 127);
  printf("div_b     = %d\r\n", (lick_freq / 128) - 1);
  printf("\r\n");

  while(1)
  {
    /* Get current calendar */
    ertc_calendar_get(&time);

    if(temp != time.sec)
    {
      temp = time.sec;

      /* Print: Y-M-D */
      printf("%02d-%02d-%02d ",time.year, time.month, time.day);

      /* Print: H:M:S */
      printf("%02d:%02d:%02d\r\n",time.hour, time.min, time.sec);
    }
  }
}
```

5.4 Test result

- Check the print information through the serial port assistant on PC;
- Print the measured LICK frequency and values of DIV_A and DIV_B through the serial port assistant;
- Print the calendar information every second.

6 Application case 4: Tamper detection

6.1 Introduction

Demonstrate the tamper detection function. Tamper detection is triggered when a rising edge is detected on PC13. In case of a tamper event, the battery powered domain data registers are cleared.

6.2 Resources

1) Hardware:

   AT-START BOARD of the corresponding model

2) Software:

   project\at_start_f4xx\examples\ertc\tamper

Note: All projects are built around keil 5. If users want to use them in other compiling environments, please refer to AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates (such as IAR6/7, keil 4/5) for a simple change.

6.3 Software design

1) Configuration process
   - Initialize ERTC
   - Initialize tamper detection
   - Initialize battery powered registers

2) Code
   - Main function code

```
int main(void)
{
  /* Initialize system clock */
  system_clock_config();

  /* Configure NVIC priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* Initialize AT START board */
  at32_board_init();

  /* Initialize serial port */
  uart_init(115200);

  /* ERTC initialization */
  ertc_config();

  /* Initialize tamper detection */
  ertc_tamper_config();

  /* Write data to battery powered registers */
  bpr_reg_write();
```

```
  /* Check whether battery powered registers are correct */
  if(bpr_reg_check() == TRUE)
  {
    printf("init: bpr data registers are not reset\r\n");
  }
  else
  {
    printf("init: bpr data registers are cleared\r\n");
  }

  while(1)
  {
  }
}
```

- Tamper detection interrupt handler code

```
void TAMP_STAMP_IRQHandler(void)
{
  if(ertc_flag_get(ERTC_TP1F_FLAG) != RESET)
  {
    /* Check whether battery powered registers are cleared */
    if(is_bpr_reg_reset() == 0)
    {
      printf("tamper: bpr data registers are cleared\r\n");
    }
    else
    {
      printf("tamper: bpr data registers are not reset\r\n");
    }

    /* Clear tamper detection flag */
    ertc_flag_clear(ERTC_TP1F_FLAG);

    /* Clear EXINT flag */
    exint_flag_clear(EXINT_LINE_21);
  }
}
```

6.4 Test result

- Check the print information through the serial port assistant on PC;
- When a tamper event occurs (riding edge on PC13), print the information of battery powered registers cleared in the tamper interrupt function.

7 Application case 4: Time stamp

7.1 Introduction

Demonstrate the function of time stamp. When a rising edge is detected on PC13, the time stamp is triggered, and print the time when this event occurs in the time stamp interrupt function.

7.2 Resources

1) Hardware:

   AT-START BOARD of the corresponding model

2) Software:

   project\at_start_f4xx\examples\ertc\time_stamp

Note: All projects are built around keil 5. If users want to use them in other compiling environments, please refer to AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates (such as IAR6/7, keil 4/5) for a simple change.

7.3 Software design

1) Configuration process
   - Initialize ERTC
   - Initialize time stamp

2) Code
   - Main function code

```
int main(void)
{
  /* Initialize system clock */
  system_clock_config();

  /* Configure NVIC priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* Initialize AT START board */
  at32_board_init();

  /* Initialize serial port */
  uart_init(115200);

  /* ERTC initialization */
  ertc_config();

  /* Initialize time stamp */
  ertc_timestamp_config();

  /* Display the current time */
  ertc_time_show();

  while(1)
```

```
  {

  }
}
```

- Time stamp interrupt handler code

```
void TAMP_STAMP_IRQHandler(void)
{
  if(ertc_flag_get(ERTC_TSF_FLAG))
  {
    /* Display time stamp */
    ertc_timestamp_show();

    /* display the date / time */
    ertc_time_show();

    /* Clear EXINT flag */
    exint_flag_clear(EXINT_LINE_21);

    /* Clear time stamp flag */
    ertc_flag_clear(ERTC_TSF_FLAG);
  }
}
```

7.4 Test result

- Check the print information through the serial port assistant on PC;
- When time stamp event occurs (rising edge on PC13), print the current time stamp in the interrupt function.

8 Application case 5: Periodic wakeup timer

8.1 Introduction

Demonstrate how to use periodic wakeup timer.

8.2 Resources

1) Hardware:

   AT-START BOARD of the corresponding model

2) Software:

   project\at_start_f4xx\examples\ertc\wakeup_timer

Note: All projects are built around keil 5. If users want to use them in other compiling environments, please refer to AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates (such as IAR6/7, keil 4/5) for a simple change.

8.3 Software design

1) Configuration process
   - Initialize ERTC
   - Initialize periodic wakeup timer
2) Code
   - Main function code

```
int main(void)
{
  ertc_time_type time;
  uint32_t temp = 0;

  /* Initialize system clock */
  system_clock_config();

  /* Configure NVIC priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* Initialize AT START board */
  at32_board_init();

  /* Initialize serial port */
  uart_init(115200);

  /* ERTC initialization */
  ertc_config();

  /* Initialization of wakeup timer */
  wakeup_timer_config();

  while(1)
  {
```

```
      /* Get current calendar */
      ertc_calendar_get(&time);

      if(temp != time.sec)
      {
        temp = time.sec;

        /* Print: Y-M-D */
        printf("%02d-%02d-%02d ",time.year, time.month, time.day);

        /* Print: H:M:S */
        printf("%02d:%02d:%02d\r\n",time.hour, time.min, time.sec);
      }
    }
}
```

- Periodic wakeup interrupt handler code

```
void ERTC_WKUP_IRQHandler(void)
{
  if(ertc_flag_get(ERTC_WATF_FLAG) != RESET)
  {
    printf("wakeup\r\n");

    at32_led_on(LED2);

    /* Clear wakeup timer flag */
    ertc_flag_clear(ERTC_WATF_FLAG);

    /* Clear EXINT flag */
    exint_flag_clear(EXINT_LINE_22);
  }
}
```

8.4 Test result

- Check the print information through the serial port assistant on PC;
- Periodic wakeup event occurs every five seconds, and print the information in the interrupt function.
- Print calendar every second.

9 Revision history

Table 14. Document revision history

Date	Version	Revision note
2021.10.19	2.0.0	Initial release.
2022.08.29	2.0.1	Added Table 2; Added the description of leap year in ERTC introduction.
2022.09.30	2.0.2	Modified Table 5.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

Purchasers are solely responsible for the selection and use of ARTERY's products and services, and ARTERY assumes no liability whatsoever relating to the choice, selection or use of the ARTERY products and services described herein.

No license, express or implied, to any intellectual property rights is granted under this document. If any part of this document deals with any third party products or services, it shall not be deemed a license grant by ARTERY for the use of such third party products or services, or any intellectual property contained therein, or considered as a warranty regarding the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

Unless otherwise specified in ARTERY's terms and conditions of sale, ARTERY provides no warranties, express or implied, regarding the use and/or sale of ARTERY products, including but not limited to any implied warranties of merchantability, fitness for a particular purpose (and their equivalents under the laws of any jurisdiction), or infringement of any patent, copyright or other intellectual property right.

Purchasers hereby agrees that ARTERY's products are not designed or authorized for use in: (A) any application with special requirements of safety such as life support and active implantable device, or system with functional safety requirements; (B) any air craft application; (C) any automotive application or environment; (D) any space application or environment, and/or (E) any weapon application. Purchasers' unauthorized use of them in the aforementioned applications, even if with a written notice, is solely at purchasers' risk, and is solely responsible for meeting all legal and regulatory requirement in such use.

Resale of ARTERY products with provisions different from the statements and/or technical features stated in this document shall immediately void any warranty grant by ARTERY for ARTERY products or services described herein and shall not create or expand in any manner whatsoever, any liability of ARTERY.