AN0047
入门指南

AT32 ERTC入门指南

前言

AT32 的 ERTC 接口用于实现日历、时钟功能，本文主要就 ERTC 的基本功能进行讲解和案列解析。

注：本应用笔记对应的代码是基于雅特力提供的V2.x.x 板级支持包（BSP）而开发，对于其他版本BSP，需要注意使用上的区别。

支持型号列表：

支持型号	具备 ERTC 的型号

目录

1 ERTC 接口简介 6

2 ERTC 功能 7

2.1 各型号 ERTC 功能差异 7

2.2 寄存器访问 8

2.3 时钟设置 9

2.4 日历 11

2.5 闹钟 13

2.6 周期性自动唤醒 15

2.7 入侵检测 16

2.8 时间戳 18

2.9 参考时钟检测 19

2.10 校准 20

2.11 事件输出功能 22

2.12 电池供电数据寄存器 22

2.13 中断 23

3 案例 读写电池供电数据寄存器 26

3.1 功能简介 26

3.2 资源准备 26

3.3 软件设计 26

3.4 实验效果 27

4 案例 使用日历以及闹钟功能 29

4.1 功能简介 29

4.2 资源准备 29

4.3 软件设计 29

4.4 实验效果 33

5 案例 使用 LICK 时钟并校准 34

5.1 功能简介 34

5.2 资源准备 34

5.3 软件设计 34

5.4 实验效果 35

6 案例 入侵检测 36

6.1 功能简介 36

6.2 资源准备 36

6.3 软件设计 36

6.4 实验效果 37

7 案例 时间戳 38

7.1 功能简介 38

7.2 资源准备 38

7.3 软件设计 38

7.4 实验效果 39

8 案例 周期唤醒定时器 40

8.1 功能简介 40

8.2 资源准备 40

8.3 软件设计 40

8.4 实验效果 41

9 文档版本历史 42

表目录

表 1. 各型号 ERTC 差异 7
表 2. 各型号 ERTC 寄存器读写时间差异 7
表 3. ERTC 寄存器 8
表 4. 各型号 HEXT 的预分频值 9
表 5. 各时钟源优缺点对比 10
表 6. 分频设置举例 10
表 7. 屏蔽设置举例 14
表 8. 亚秒屏蔽设置举例 14
表 9. ERTC 唤醒低功耗模式 23
表 10. 中断控制 23
表 11. 各型号中断对应 EXINT 线 23
表 12. 各型号中断对应中断向量号 24
表 13. 中断向量对应中断函数 24
表 14. 文档版本历史 42

图目录

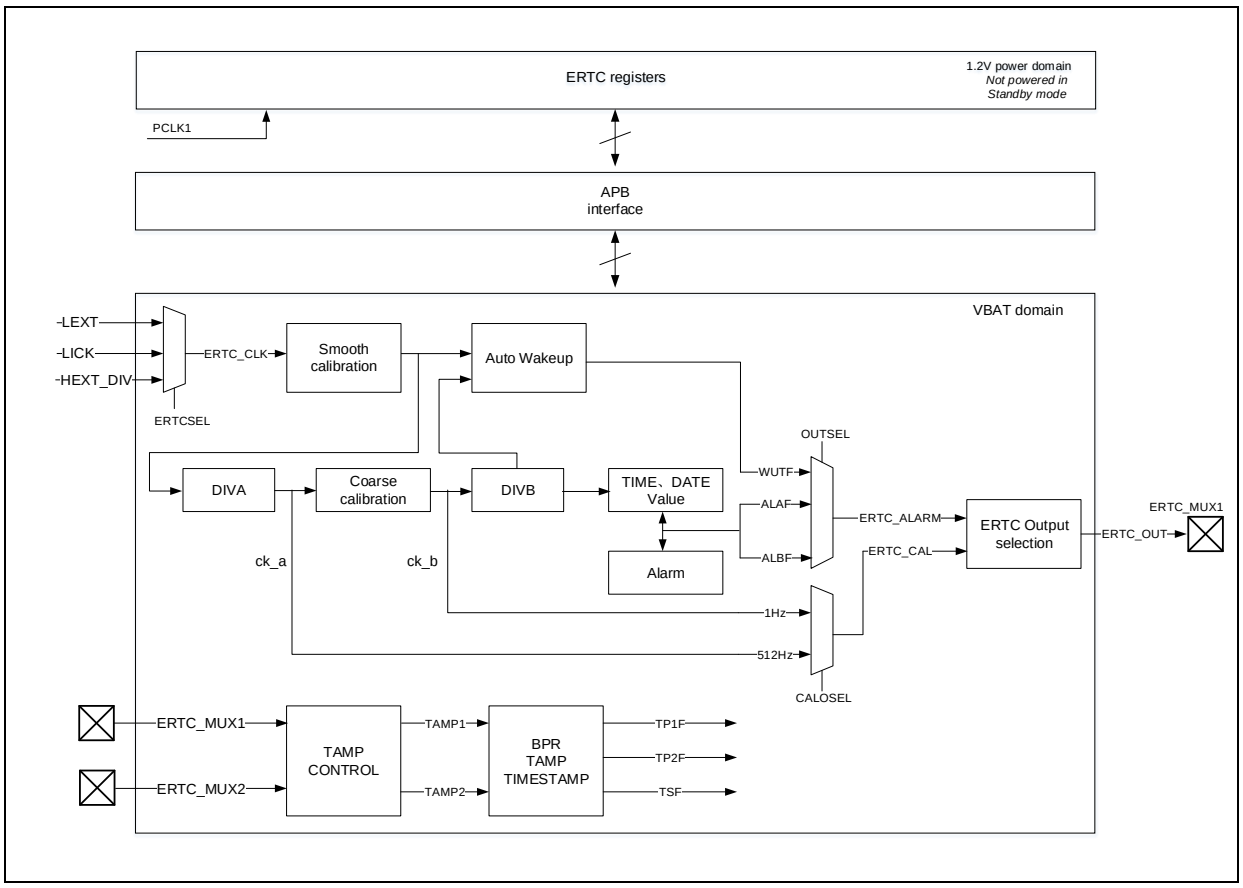
图 1. ERTC 框图 6
图 2. ERTC 时钟结构 9
图 3. 日历格式 11
图 4. 日闹钟匹配 13
图 5. 唤醒定时器时钟选择 16
图 6. 入侵检测 16
图 7. 预充电时间为 4 个 ERTC CLK 示意图 17
图 8. 时间戳检测 19
图 9. 参考时钟检测 20
图 10. 粗略校准 20
图 11. 精密校准 21
图 12. 事件输出 22

1 ERTC 接口简介

ERTC 计数逻辑位于电池供电域，只要电池供电域有电，ERTC 便会一直运行，不受系统复位以及 VDD 掉电影响，ERTC 主要具有以下功能：

- 日历功能：年、月、日、时、分、秒
- 自动处理月份天数 28（平年 2 月）、29（闰年 2 月）、30（小月）、31（大月），其中当年份寄存器是 4 的倍数时为闰年
- 闹钟功能：闹钟 A、闹钟 B
- 周期性唤醒功能
- 入侵检测功能
- 校准功能：精密校准、粗略校准

图 1. ERTC 框图



2 ERTC 功能

2.1 各型号 ERTC 功能差异

各型号 ERTC 基本功能相同，只是各个型号之间，可能去掉了一些更高级的功能，所有保留的功能程序兼容。

表 1. 各型号 ERTC 差异

型号	闹钟 A	闹钟 B	周期性唤醒	入侵检测 1	入侵检测 2	精密校准	粗略校准	时间戳	参考时钟检测	电池供电数据寄存器（个）
AT32F415xx	√	√	√	√	×	√	√	√	√	20
AT32F421xx	√	×	×	√	×	√	×	√	√	5
AT32F425xx	√	×	√	√	×	√	×	√	√	5
AT32F435xx	√	√	√	√	√	√	√	√	√	20
AT32F437xx	√	√	√	√	√	√	√	√	√	20
AT32L021xx	√	×	√	√	×	√	×	√	√	5
AT32F405xx	√	√	√	√	√	√	×	√	√	20
AT32F402xx	√	√	√	√	√	√	×	√	√	20
AT32F423xx	√	√	√	√	√	√	×	√	√	20
AT32A423xx	√	√	√	√	√	√	×	√	√	20
AT32M412xx	√	√	√	√	√	√	×	√	√	20
AT32M416xx	√	√	√	√	√	√	×	√	√	20
AT32F455xx	√	√	√	√	√	√	√	√	√	20
AT32F456xx	√	√	√	√	√	√	√	√	√	20
AT32F457xx	√	√	√	√	√	√	√	√	√	20
AT32F490xx	√	√	√	√	√	√	×	√	√	20
AT32WB415xx	√	√	√	√	×	√	√	√	√	20

√：表示支持该功能，且功能相同。

×：表示不支持该功能。

各型号 ERTC 读写寄存器所需时间有所不同，使用时应注意差异。

表 2. 各型号 ERTC 寄存器读写时间差异

型号	写寄存器时钟个数	单位	读寄存器时钟个数	单位
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
AT32F402xx	15	PCLK1	15	PCLK1
AT32F423xx	15	PCLK1	15	PCLK1
AT32A423xx	15	PCLK1	15	PCLK1

AT32M412xx	15	PCLK1	15	PCLK1
AT32M416xx	15	PCLK1	15	PCLK1
AT32F455xx	15	PCLK1	15	PCLK1
AT32F456xx	15	PCLK1	15	PCLK1
AT32F457xx	15	PCLK1	15	PCLK1
AT32F490xx	15	PCLK1	15	PCLK1
AT32WB415xx	2	SYSCLK	2	SYSCLK

2.2 寄存器访问

寄存器写保护

上电复位后 ERTC 寄存器处于写保护状态，需要先解除写保护，才能写配置 ERTC 寄存器。

需要注意的是 ERTC_STS[14：8]、ERTC_TAMP 和 ERTC_BPRx 寄存器不受写保护。

解锁步骤：

1) 使能 PWC 接口时钟

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

2) 解锁电池供电域写保护

```
pwc_battery_powered_domain_access(TRUE);
```

3) 依次向 ERTC_WP 寄存器写入 0xCA，0x53，解锁写保护

若向 ERTC_WP 寄存器写入错误的值，将重新激活写保护

```
ertc_write_protect_disable();
```

4) 配置 ERTC 寄存器

下表列举了 ERTC 寄存器受写保护状态，以及写入的条件：

表 3. ERTC 寄存器

寄存器简称	是否受 ERTC_WP 写保护	是否需要进入初始化模式	其它
ERTC_TIME	是	是	-
ERTC_DATE	是	是	-
ERTC_CTRL	是	位 7、6、4 需要	-
ERTC_STS	除位[14：8]外	-	-
ERTC_DIV	是	是	-
ERTC_WAT	是	否	WATWF 为 1 时可配置
ERTC_CCAL	是	是	-
ERTC_ALA	是	否	ALAWF 为 1 时可配置
ERTC_ALB	是	否	ALBWF 为 1 时可配置
ERTC_WP	-	-	-
ERTC_SBS	-	-	-
ERTC_TADJ	是	否	TADJF 为 0 时可配置
ERTC_TSTM	-	-	-
ERTC_TSDT	-	-	-

ERTC_TSSBS	-	-	-
ERTC_SCAL	是	否	CALUPDF 为 0 时可配置
ERTC_TAMP	否	否	-
ERTC_ALASBS	是	否	ALAWF 为 1 时可配置
ERTC_ALBSBS	是	否	ALBWF 为 1 时可配置
ERTC_BPRx	否	否	-

寄存器复位

ERTC 寄存器处于电池供电域，可以 CRM_BPDC 的 BPDRST 进行电池供电域复位，也可以由提供的库函数对每个寄存器写默认值进行复位。

ERTC 复位相关函数：

电池供电域复位

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

将 ERTC 所有寄存器恢复成默认值

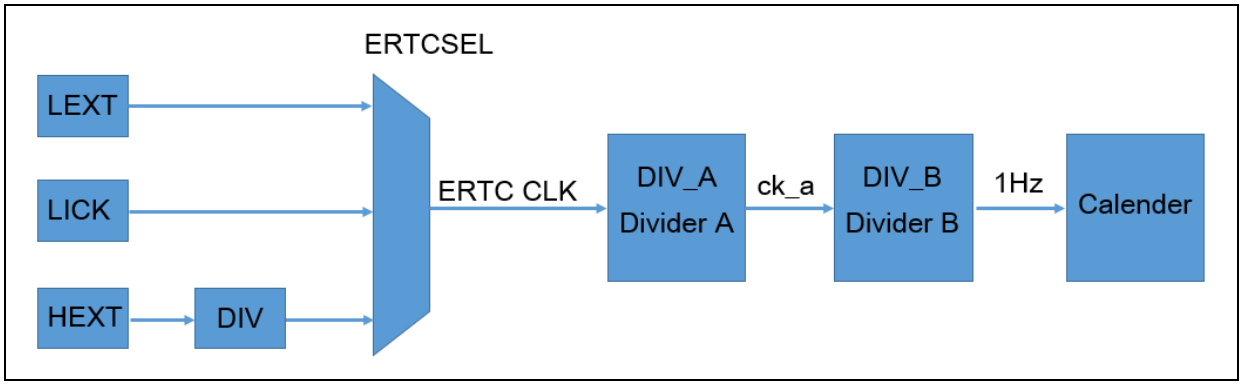
```
error_status ertc_reset(void)
```

2.3 时钟设置

时钟源选择

ERTC 时钟源经过选择后输入到分频器 A 和分频器 B，最终得到 1Hz 的时钟用来更新日历。

图 2. ERTC 时钟结构



ERTC 的时钟源共有 3 种可以选择：

- LEXT：外部低速晶振，通常为 32.768kHz
- LICK：内部低速晶振，通常典型值为 40kHz 范围（30~60kHz），详情请见各型号的 datasheet
- HEXT_DIV：外部高速晶振分频后得到的时钟，不同的型号下，分频值不一样，请见表 3

表 4. 各型号 HEXT 的预分频值

型号	预分频值
AT32F415xx	固定 128
AT32F421xx	固定 32

AT32F425xx	固定 32
AT32F435xx	可变 2~31
AT32F437xx	可变 2~31
AT32L021xx	固定 32
AT32F405xx	可变 2~31
AT32F402xx	可变 2~31
AT32F423xx	可变 2~31
AT32A423xx	可变 2~31
AT32M412xx	可变 2~31
AT32M416xx	可变 2~31

表 5. 各时钟源优缺点对比

时钟源	频率	优点	缺点
LEXT	32.768kHz	精度最高，能在电池供电下以及低功耗模式下工作	需要一颗晶振，增加元件成本，增大 PCB 布线面积
LICK	典型值 40kHz 范围(30kHz~60kHz)	能在低功耗模式下工作，节省一颗晶振，降低了 PCB 布线面积	时间精度低，时间不准
HEXT	主晶振频率	精度也比较高（取决于主晶振精度），节省一颗晶振，降低了 PCB 布线面积	不能在电池供电和低功耗模式下工作

ERTC 时钟源设置相关函数：

选择对应时钟使能

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

选择 ERTC 时钟

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

使能 ERTC 时钟

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

预分频器设置

通过预分频器 A 和预分频器 B 将获得 1Hz 时钟，计算公式如下：

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

推荐在应用中将预分频器 A 设置成最大值（127）这样可以最大程度降低功耗。

表 6. 分频设置举例

时钟源	频率	DIV_A	DIV_B	日历时钟
LEXT	32.768kHz	127	256	1Hz
LICK	典型值 40kHz	124	319	1Hz

HEXT	8MHz，32 分频 （AT32F421 为例）	124	1999	1Hz

设置 ERTC 预分频器

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

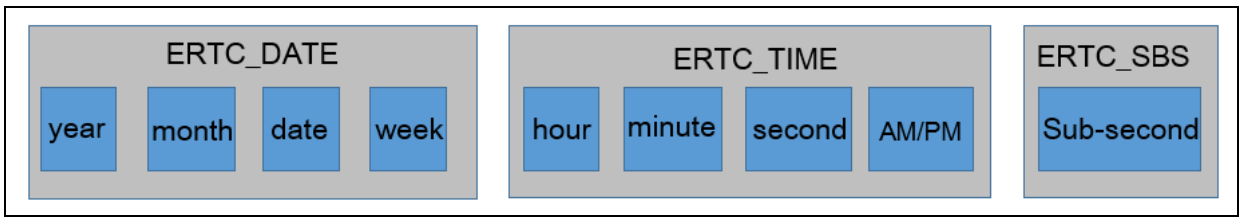
ERTC 时钟初始化举例：

```
/* 使能 LEXT 时钟 */
crm_clock_source_enable(CRM_CLOCK_SOURCE_LEXT, TRUE);

/* 等待 LEXT 时钟稳定 */
while(crm_flag_get(CRM_LEXT_STABLE_FLAG) == RESET)
{
}

/* 选择 LEXT 时钟作为 ERTC 时钟 */
crm_ertc_clock_select(CRM_ERTC_CLOCK_LEXT);

/* 使能 ERTC 时钟 */
crm_ertc_clock_enable(TRUE);

/* 配置 ERTC 分频器 DIVA=127，DIV_B=255 */
/* ertc second(1hz) = ertc_clk / (div_a + 1) * (div_b + 1) */
ertc_divider_set(127, 255);
```

2.4 日历

ERTC 日历格式如下图所示，包含有年、月、日、星期、时、分、秒、亚秒。

图 3. 日历格式



时间格式设置

时间格式可以选择 24 小时或者 12 小时格式，如果为 24 小时格式时，AM/PM 字段无意义，在初始化日历之前，应该首先选择时间格式。

时间格式设置函数：

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

例如设置日历格式为 24 小时模式

```
ertc_hour_mode_set(ERTC_HOUR_MODE_24);
```

日历初始化

通过配置 ERTC_DATE 和 ERTC_TIME 寄存器可以设置日历时间：

日历值设置函数：

```
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)
```

例如设置时间为 2020-05-01 12:30:00 星期六

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

日历读取

通过读取 ERTC_DATE、ERTC_TIME 和 ERTC_SBS 寄存器可以读取日历时间，日历读取有两种模式分别为同步读取（DREN=0）和异步读取（DREN=1）。

- 同步读取：ERTC 每两个 ERTC_CLK 周期，将日历值同步到影子寄存器 ERTC_DATE、ERTC_TIME 和 ERTC_SBS，同步完成后 UPDF 将置 1。读取低阶寄存器时会将高阶寄存器值锁定，直到读取 ERTC_DATE 寄存器，这保证读取的 ERTC_SBS、ERTC_TIME、ERTC_DATE 寄存器值来自同一时刻。

  例如读取 ERTC_SBS，会将 ERTC_TIME、ERTC_DATE 寄存器值锁定。
- 异步读取：ERTC 直接读取电池供电域的 ERTC 时钟和日历值，这样避免了由于同步时间带来的误差。异步读取时，UPDF 标志将由硬件清 0。为保证异步读取时钟和日历值来自同一时刻，软件必须连续两次读取时钟和日历值，并比较两次结果是否一致，如果不一致应该再读，直到两次结果一致。

在大多数应用下，都推荐使用同步读取模式，因为这样可以简化程序。

等待同步函数（等待 UPDF 置 1）

```
error_status ertc_wait_update(void)
```

读取模式设置函数

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

例如设置读取模式为同步读取

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

日历读取函数：

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

结构体 ertc_time_type 里面参数含义如下：

- year：年
- month：月
- day：日
- hour：时
- min：分
- sec：秒
- week：星期几

— ampm：上午/下午（只有在 12 小时制时，数据有效）

亚秒读取

亚秒值为预分频器 DIV_B 的计数值，预分频器 DIV_B 是一个递减计数器，例如当 DIV_B=255 时，1 个亚秒值代表的时间为 1/(255 + 1)秒。

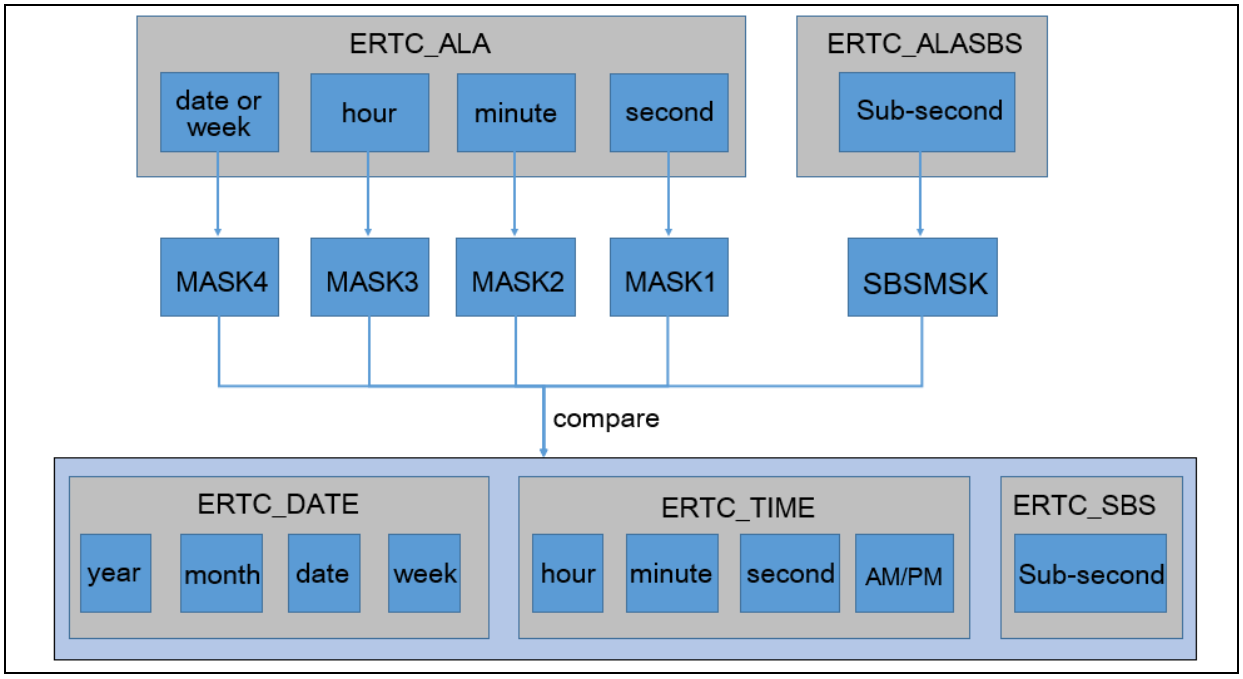
亚秒读取函数：

```
uint32_t ertc_sub_second_get(void)
```

2.5 闹钟

ERTC 包含两个完全相同的闹钟 A、闹钟 B，闹钟值由 ERTC_ALASBS/ERTC_ALA（ERTC_ALBSBS/ERTC_ALB）设定，开启闹钟后，当设定的闹钟值匹配日历值时 ALAF/ALBF 置 1，发生闹钟事件。通过 MASKx 位，可选择性的屏蔽日历字段，被屏蔽的字段不参与闹钟匹配。

图 4. 日闹钟匹配



因为闹钟 A 和闹钟 B 完全一样，所以后面的举例都用闹钟 A 举例

闹钟格式选择：

— ERTC_ALA 的 WKSEL=0 时：日期、时、分、秒、亚秒

— ERTC_ALA 的 WKSEL=1 时：星期、时、分、秒、亚秒

闹钟的各个字段日期/星期、时、分、秒、亚秒均可通过 MASK 位屏蔽，使闹钟的产生更加的灵活

— MASK4 = 1：闹钟和日/星期无关

— MASK3 = 1：闹钟和小时无关

— MASK2 = 1：闹钟和分钟无关

— MASK1 = 1：闹钟和秒钟无关

例如在 WKSEL=0 时，将闹钟设置为 15 号 12:30:10

表 7. 屏蔽设置举例

	MASK4	MASK3	MASK2	MASK1	现象
1	1	0	0	0	xx-12:30:10 触发闹钟 xx 表示任意日期
2	0	1	0	0	15-xx:30:10 触发闹钟 xx 表示任意小时
3	0	0	1	0	15-12:xx:10 触发闹钟 xx 表示任意分钟
4	0	0	0	1	15-12:30:xx 触发闹钟 xx 表示任意秒钟

通过设置 ERTC_ALASBS 的 SBSMSK 可以对亚秒进行屏蔽：

- SBSMSK = 0：不匹配亚秒，闹钟与亚秒无关；
- SBSMSK = 1：只匹配 SBS[0]；
- SBSMSK = 2：只匹配 SBS[1：0]；
- SBSMSK = 3：只匹配 SBS[2：0]；
- ...
- SBSMSK = 14：只匹配 SBS[13：0]；
- SBSMSK = 15：匹配 SBS[14：0]。
-

例如在 DIV_A = 127，DIV_B=255（亚秒）时，只考虑亚秒的触发闹钟

表 8. 亚秒屏蔽设置举例

SBSMSK	现象
0	闹钟和亚秒无关
1	只匹配 SBS[0]，1/128 秒发生一次闹钟
2	只匹配 SBS[1:0]，1/64 秒发生一次闹钟
3	只匹配 SBS[2:0]，1/32 秒发生一次闹钟
4	只匹配 SBS[3:0]，1/16 秒发生一次闹钟
5	只匹配 SBS[4:0]，1/8 秒发生一次闹钟
6	只匹配 SBS[5:0]，1/4 秒发生一次闹钟
7	只匹配 SBS[6:0]，1/2 秒发生一次闹钟
8	只匹配 SBS[7:0]，1 秒发生一次闹钟
9	只匹配 SBS[8:0]，1 秒发生一次闹钟
10	只匹配 SBS[9:0]，1 秒发生一次闹钟
11	只匹配 SBS[10:0]，1 秒发生一次闹钟
12	只匹配 SBS[11:0]，1 秒发生一次闹钟
13	只匹配 SBS[12:0]，1 秒发生一次闹钟
14	只匹配 SBS[13:0]，1 秒发生一次闹钟

15	只配 SBS[14:0]，1 秒发生一次闹钟

闹钟相关函数

日期/星期、时、分、秒屏蔽

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

选择日期或者星期格式

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

设置闹钟值：日期/星期、时、分、秒、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)
```

设置闹钟亚秒值以及屏蔽

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

闹钟中断使能

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

获取当前配置的闹钟值

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

获取当前配置的闹钟亚秒值

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

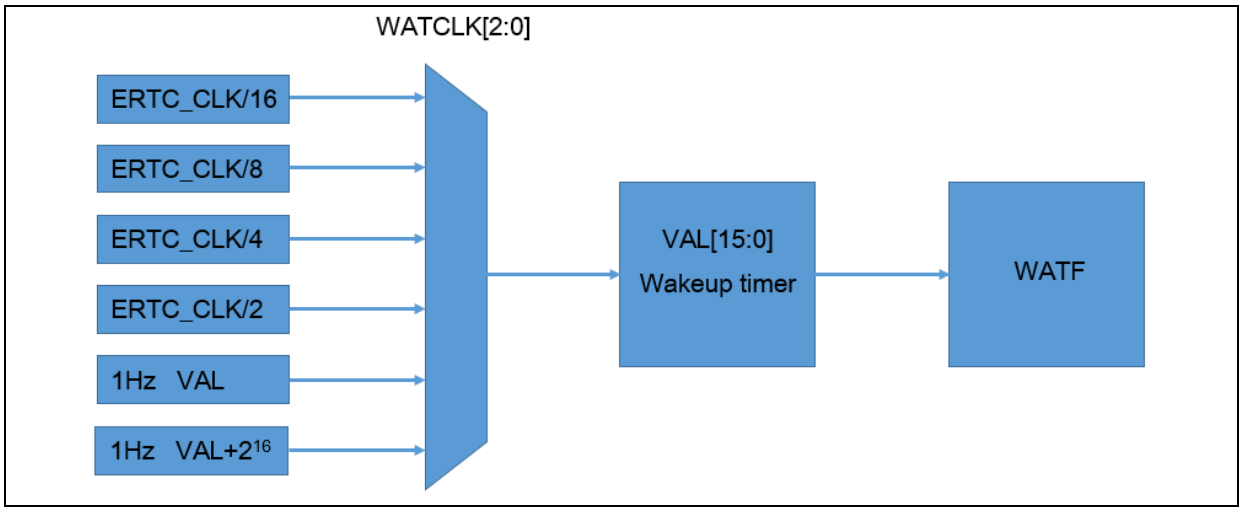
2.6 周期性自动唤醒

周期性唤醒功能用于周期性自动唤醒低功耗模式，唤醒周期由 VAL[15：0]设定。当唤醒计数器值由 VAL 值递减至 0 时，WATF 标志置 1，产生唤醒事件，同时唤醒计数器值重载 VAL 值。

可以根据需要选择不同的时钟源，通过寄存器 WATCLK[2：0]配置

- 000：ERTC_CLK/16；
- 001：ERTC_CLK/8；
- 010：ERTC_CLK/4；
- 011：ERTC_CLK/2；
- 10x：1Hz；
- 11x：1Hz，唤醒计数值增加 2^{16}，唤醒时间= WAT+2^{16} + 1。

图 5. 唤醒定时器时钟选择



当 WATCLK[2：0]= 11x 时，如果日历时钟为 1Hz，可获得最长的唤醒时间=65535+2^{16}+1=131072 秒。

如果日历时钟调整为<1Hz（增大预分频器 DIV_B 的值），还可以获得更长的唤醒时间。

周期自动唤醒相关函数

唤醒计数器时钟源选择

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

设置唤醒计数器值

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

获取唤醒计数器值

```
uint16_t ertc_wakeup_counter_get(void)
```

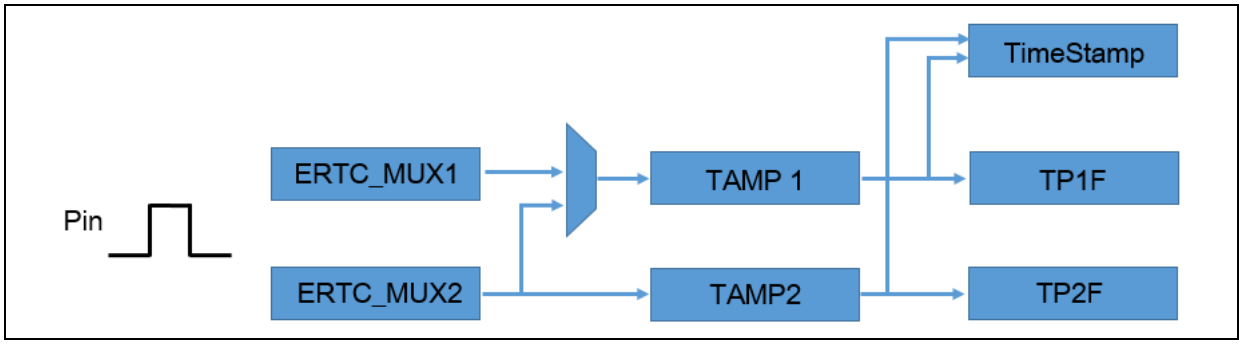
唤醒定时器使能

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

2.7 入侵检测

ERTC 提供了两组入侵检测 TAMP1 和 TAMP2，当在发生入侵事件时，将自动清除 ERTC 电池供电数据寄存器(ERTC_BPRx)的值，也支持发生入侵事件时产生事件戳。

图 6. 入侵检测



入侵检测 1 引脚可以选择

- ERTC_MUX1：引脚 1，通常为 PC13
- ERTC_MUX2：引脚 2，通常为 PA0

入侵检测 2 引脚为固定 ERTC_MUX2（通常为 PA0）

入侵检测模式分为边沿检测和电平检测

- 边沿检测：当检测到了有效的边沿触发入侵检测，分为上升沿触发、下降沿触发
- 电平检测：当有效电平长度达到了设定的时间，触发发入侵检测，分为高电平触发、低电平触发。

其中边沿检测比较简单，只需要配置有效边沿并使能就行了，电平检测需要配置的参数会比较多，需要配置以下参数：

采样频率

通过配置 TPFREQ 寄存器，可以配置的入侵检测频率为

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

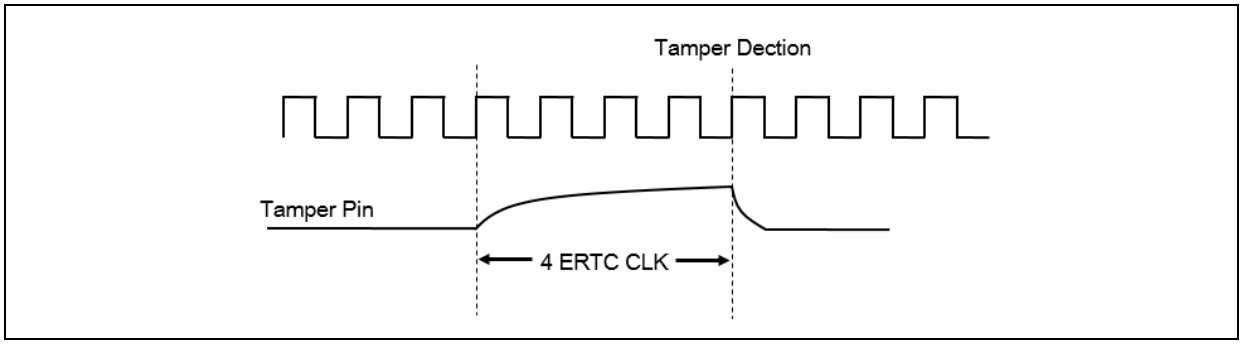
例如当 ERTC_CLK=32768Hz 时，配置检测频率为 ERTC_CLK/256 时，此时入侵检测频率为 32768/256 = 128Hz

入侵上拉

通过 TPPU 配置可以打开或者关闭入侵上拉功能，当使能了入侵上拉电阻时，可以通过 TPPR 设置在入侵检测前的上拉预充电时间，时间可配置为如下：

- 1 个 ERTC_CLK；
- 2 个 ERTC_CLK；
- 4 个 ERTC_CLK；
- 8 个 ERTC_CLK。

图 7. 预充电时间为 4 个 ERTC CLK 示意图



入侵滤波

通过 TPFLT 设置入侵检测的滤波时间，可以配置下面 4 种模式

- 无滤波，当 1 次采样有效，判定入侵事件发生；
- 连续 2 次采样有效，判定入侵事件发生；
- 连续 4 次采样有效，判定入侵事件发生；
- 连续 8 次采样有效，判定入侵事件发生。

入侵检测相关函数

入侵检测 1 引脚选择

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

入侵检测上拉使能配置

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

上拉预充电时间设置

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

滤波时间设置

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

入侵检测频率设置

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

入侵检测有效边沿设置

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

发生入侵事件时，保存时间戳

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

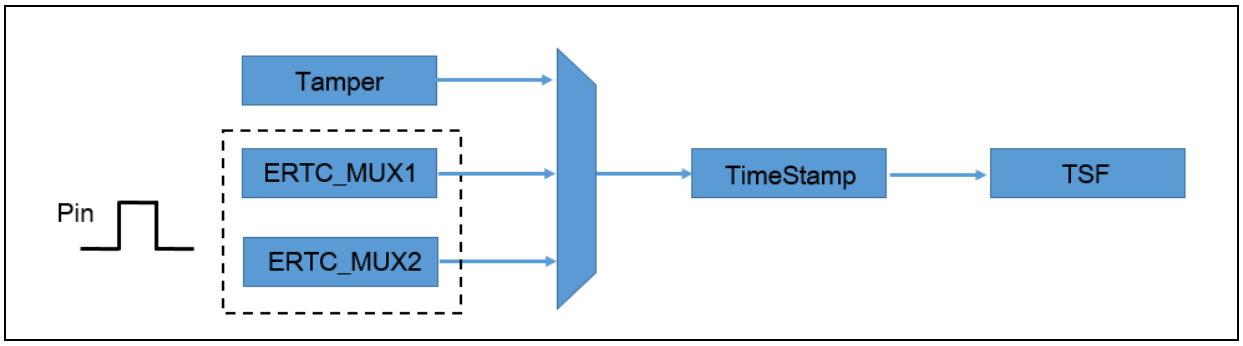
入侵检测使能

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

2.8 时间戳

时间戳功能用于在发生时间戳事件时（入侵引脚检测到有效边沿），将当前的日历值保存到时间戳寄存器中。

图 8. 时间戳检测



时间戳使用

— 单独使用：可以选择引脚进行检测

ERTC_MUX1：引脚 1，通常为 PC13

ERTC_MUX2：引脚 2，通常为 PA0

— 在发生入侵事件时保存时间戳，在这种模式下，需要先将入侵检测功能正确配置好

时间戳在单独使用时可以配置为上升沿检测或者下降沿检测，在入侵检测触发时，取决于入侵检测的配置

时间戳溢出

当发生时间戳时，TSF 位置 1，此时若再次发生时间戳事件，TSOF 标志位将置 1，但时间戳寄存器并不会更新，仍保留第一次触发的值。

时间戳相关函数

时间戳引脚选择

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

检测边沿设置

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

时间戳使能

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

获得时间戳时间

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

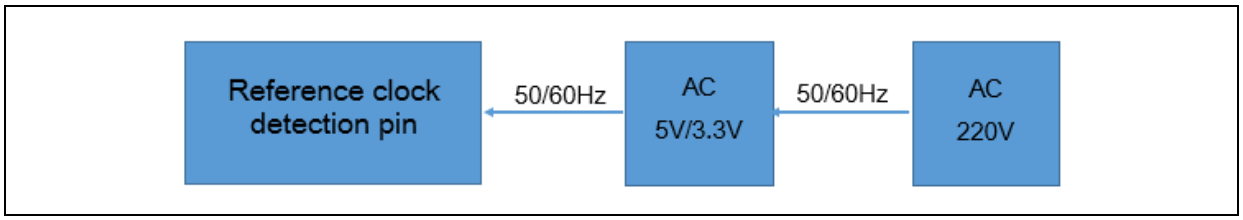
获得时间戳亚秒

```
uint32_t ertc_timestamp_sub_second_get(void)
```

2.9 参考时钟检测

为保证日历长时间运行的精确性，ERTC 提供了时钟同步功能（低功耗模式不可用），用精度更高的参考时钟（一般用 50Hz 或者 60Hz 的市电）校准更新日历的 1Hz 时钟。

图 9. 参考时钟检测



参考时钟检测功能开启后，在每次更新日历值的前 7 个 ck_a 周期检测参考时钟边沿，若检测到边沿，将使用此边沿更新日历值（更新秒钟），后续采用 3 个 ck_a 周期检测参考时钟边沿。每一次检测到参考时钟边沿时，都会将分频器 A 的值进行重载，这会使得内部 1Hz 的日历时钟与参考时钟边沿刚好对齐，当内部 1Hz 时钟出现微小偏移时，利用更精确的参考时钟，将 1Hz 时钟微调至与参考时钟边沿对齐。当没有检测到参考时钟边沿时，ERTC 会利用原来的时钟源更新日历。

需要注意的是，使能参考时钟功能后，需要将 DIVA、DIVB 设置为复位值（0x7F、0xFF），并且时钟同步功能不能与粗校准功能同时开启。

参考时钟检测使能函数

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

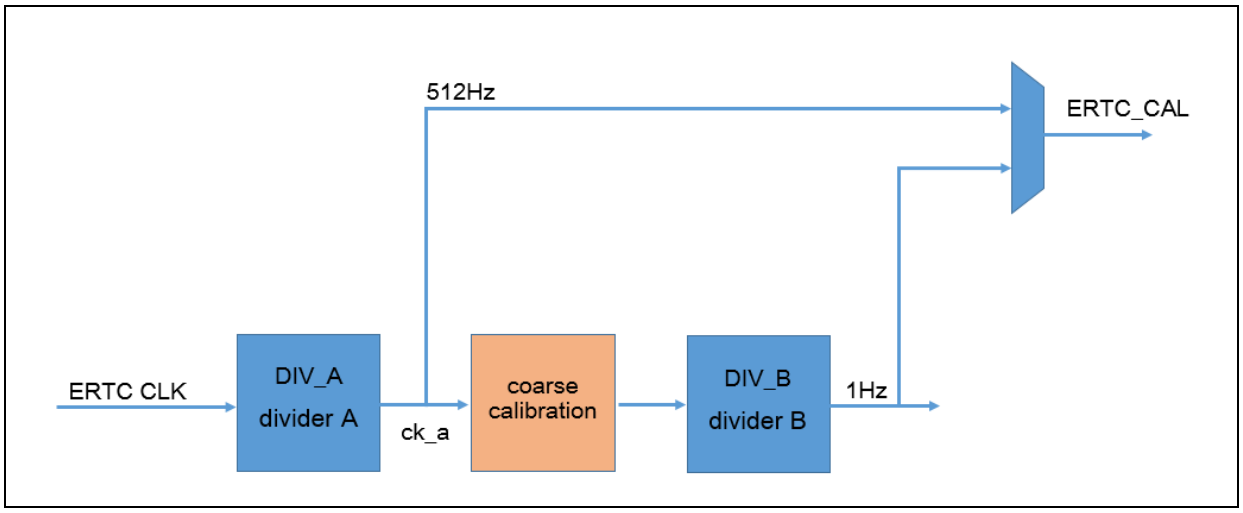
2.10 校准

ERTC 提供了两种校准方法：粗略校准和精密校准。但两种校准方法不能同时使用。

粗略校准

粗略数字校准通过增加或减少 ck_a 周期值来实现提前或推迟更新日历值的功能。

图 10. 粗略校准



正校准时（CALDIR=0）：在 64 分钟的前 2xCALVAL 分钟时间内，每分钟（约 15360 个 ck_a 周期）插入 2 个 ck_a 周期，相当于提前更新日历。

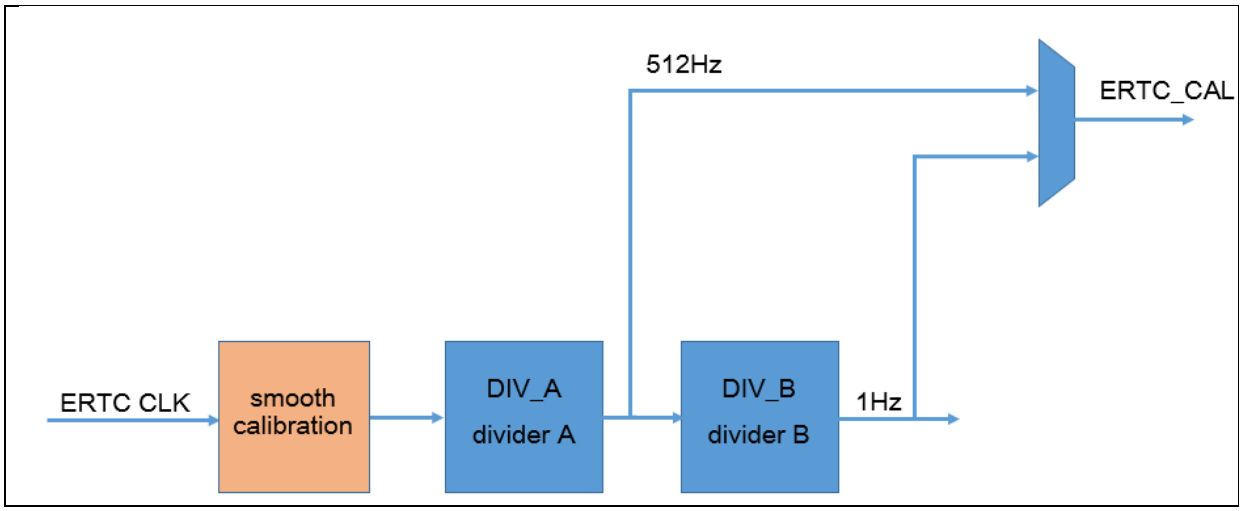
负校准时（CALDIR=1）：在 64 分钟前的 2xCALVAL 分钟时间内，每分钟（约 15360 个 ck_a 周期）忽略 1 个 ck_a 周期，相当于推迟更新日历。

注：粗略数字校准至少要将 DIVA 值设置为 6。

精密校准

区别于粗略数字校准，精密校准的校准效果更好且校准更加均匀。开启精密校准校准功能后，将均匀增加或减少 ERTC_CLK 来达到校准的目的。

图 11. 精密校准



当 ERTC_CLK 为 32.768kHz 时，精密校准周期约为 2^{20} 个 ERTC_CLK(32 秒)。DEC[8：0]值指定了 2^{20} 个 ERTC_CLK 中忽略的脉冲数，最多可忽略 511 个脉冲；将 ADD 置 1，可在 2^{20} 个 ERTC_CLK 中插入 512 个脉冲。两者搭配使用，可在 2^{20} 个 ERTC_CLK 周期进行-511~+512 的调整。

有效校准频率 F_{SCAL}：

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

当分频器 A 值小于 3 时，会按照 ADD 等于 0 校准。此时应降低分频器 B 值来实现每秒增加 8 个 ERTC_CLK，也就是 32 秒增加 256 个 ERTC_CLK 搭配 DEC[8：0]位，可在 2^{20} 个 ERTC_CLK 周期进行-255~+256 的调整。

此时有效校准频率 F_{SCAL}：

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

精密数字校准的校准周期还可选择 8 秒或 16 秒（由 CAL8 和 CAL16 配置），8 秒校准周期的优先级更高，同时使能 8 秒和 16 秒校准周期，将优先选择 8 秒校准周期。

校准相关函数

精密校准配置并使能

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

粗略校准配置

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

粗略校准使能

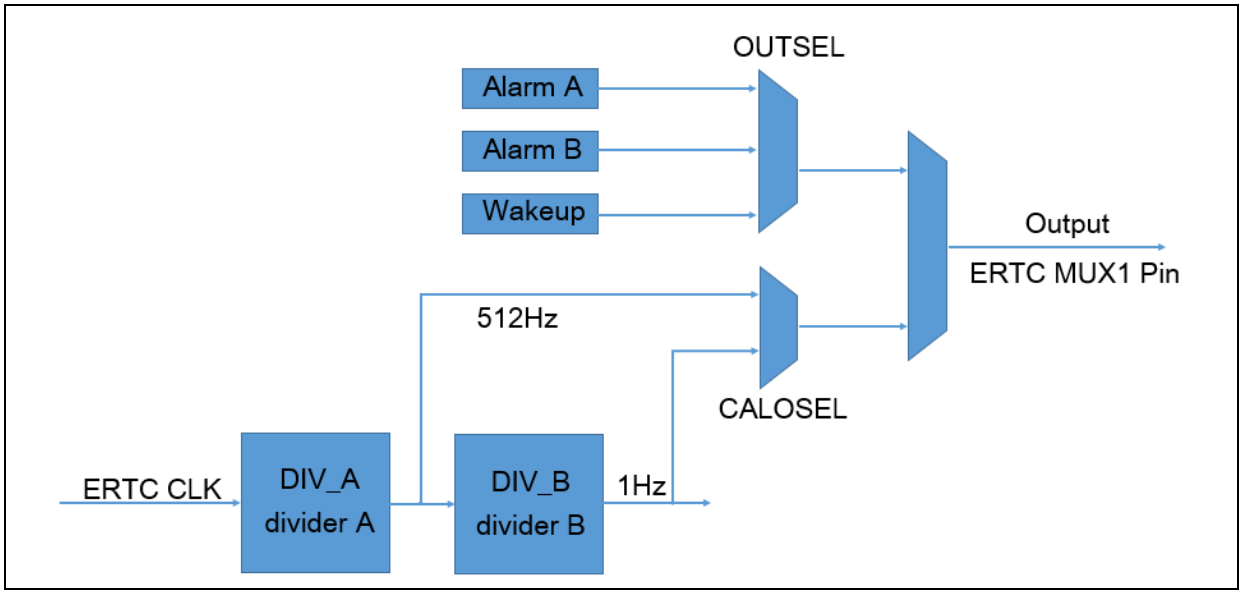
```
error_status ertc_coarse_calibration_enable(confirm_state new_state)
```

2.11 事件输出功能

ERTC 提供了一组复用功能输出，可以输出以下事件：

- 校准输出：512Hz、1Hz
- 事件输出：闹钟 A、闹钟 B、唤醒事件

图 12. 事件输出



通常 ERTC MUX1 引脚为 PC13

当输出模式为事件输出时（闹钟 A、闹钟 B、唤醒事件），可以通过 OUTTYPE 选择输出类型为开漏或是推挽，通过 OUTP 配置输出极性。

事件输出相关函数

事件输出设置（闹钟 A、闹钟 B、唤醒事件）

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

校准输出选择（512Hz、1Hz）

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

校准输出使能

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

2.12 电池供电数据寄存器

ERTC 一共提供了 20 个 32 位电池供电数据寄存器，可以在只由电池供电下保存数据，不会被系统复位所复位，只能通过电池供电域复位或入侵事件进行复位

相关函数

写电池供电数据寄存器

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

读电池供电数据寄存器

uint32_t ertc_bpr_data_read(ertc_dt_type dt)

2.13 中断

当发生闹钟 A、闹钟 B、周期性唤醒事件时，ERTC 可产生中断。要使能 ERTC 中断可按以下操作配置：

- 将 ERTC 对应中断的 EXINT 线配置为中断模式并使能，有效沿选择上升沿。
- 使能 ERTC 中断对应的 NVIC 通道。
- 使能对应的 ERTC 中断控制位。

下表说明了 ERTC 时钟源、事件以及中断对唤醒低功耗模式的影响：

表 9. ERTC 唤醒低功耗模式

时钟源	事件	唤醒 SLEEP	唤醒 DEEPSLEEP	唤醒 STANDBY
HEXT	闹钟 A	√	×	×
	闹钟 B	√	×	×
	周期性唤醒	√	×	×
	时间戳	√	×	×
	入侵事件	√	×	×
LICK	闹钟 A	√	√	√
	闹钟 B	√	√	√
	周期性唤醒	√	√	√
	时间戳	√	√	√
	入侵事件	√	√	√
LEXT	闹钟 A	√	√	√
	闹钟 B	√	√	√
	周期性唤醒	√	√	√
	时间戳	√	√	√
	入侵事件	√	√	√

表 10. 中断控制

中断事件	事件标志	中断使能位
闹钟 A	ALAF	ALAIEN
闹钟 B	ALBF	ALBIEN
周期性唤醒	WATF	WATIEN
时间戳	TSF	TSIEN
入侵事件	TP1F/TP2F	TPIEN

表 11. 各型号中断对应 EXINT 线

型号	闹钟 A	闹钟 B	周期性唤醒	时间戳	入侵检测 1	入侵检测 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	-

表 12. 各型号中断对应中断向量号

型号	闹钟 A 闹钟 B	周期性唤醒	时间戳 入侵检测 1 入侵检测 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

表 13. 中断向量对应中断函数

中断向量	中断函数
ERTCAlarm_IRQn	ERTCAlarm_IRQHandler
ERTC_IRQn	ERTC_IRQHandler
ERTC_WKUP_IRQn	ERTC_IRQHandler
TAMP_STAMP_IRQn	ERTCAlarm_IRQHandler

中断、事件相关函数

事件中断使能

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

获取相应中断是否使能

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

标志获取

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

标志清除

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

中断配置示例：以 AT32F435 的闹钟 A 为例

```
/* 配置 EXINT 线 */
exint_default_para_init(&exint_init_struct);
exint_init_struct.line_enable = TRUE; //使能
exint_init_struct.line_mode = EXINT_LINE_INTERRUPUT; //中断模式
exint_init_struct.line_select = EXINT_LINE_17;//EXINT 线 17
exint_init_struct.line_polarity = EXINT_TRIGGER_RISING_EDGE;//上升沿触发
exint_init(&exint_init_struct);

/* 使能闹钟中断 NVIC 向量：ERTCAlarm_IRQn */
```

```
nvic_irq_enable(ERTCAlarm_IRQn, 0, 1);

/* 使能闹钟中断 */
ertc_interrupt_enable(ERTC_ALA_INT, TRUE);

/* 使能闹钟 */
ertc_alarm_enable(ERTC_ALA, TRUE);
```

3 案例 读写电池供电数据寄存器

3.1 功能简介

对电池供电数据寄存器(ERTC_BPRx)进行读写访问。

3.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_f4xx\examples\ertc\bpr_domain

注：所有project都是基于keil 5而建立，若用户需要在其他编译环境上使用，请参考AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates中各种编译环境（例如IAR6/7,keil 4/5）进行简单修改即可。

3.3 软件设计

1) 配置流程

- 开启 PWC 时钟
- 使能电池供电域写保护
- 检查电池供电域数据是否正确，如果正确就跳过初始化，如果不正确就初始化 ERTC 并向电池供电域写上数据

2) 代码介绍

- main 函数代码描述

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

  /* 初始化系统时钟 */
  system_clock_config();

  /* 配置 NVIC 优先级组 */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* 初始化串口 */
  uart_init(115200);

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

  /* 开启 PWC 时钟 */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

  /* 使能电池供电域写保护 */
  pwc_battery_powered_domain_access(TRUE);
```

```
  /* 检查电池供电域数据是否正确，如果不正确就初始化，如果正确就初始化 */
  if(bpr_reg_check() == FALSE)
  {
    printf("bpr reg => reset\r\n\r\n");

    /* 初始化 ERTC */
    ertc_config();

    /* 向电池供电域写数据 */
    bpr_reg_write();
  }
  else
  {
    printf("bpr reg => none reset\r\n\r\n");

    /* 等待 ERTC 寄存器同步，最多需要 2 个 ERTC_CLK */
    ertc_wait_update();
  }

  while(1)
  {
    /* 获取当前的日历值 */
    ertc_calendar_get(&time);

    /* 如果秒钟和上一次不一样说明时间更新了 */
    if(temp != time.sec)
    {
      temp = time.sec;

      /* 打印日期：年-月-日 */
      printf("%02d-%02d-%02d ",time.year, time.month, time.day);

      /* 打印时间：时：分：秒 */
      printf("%02d:%02d:%02d\r\n",time.hour, time.min, time.sec);
    }
  }
}
```

3.4 实验效果

- 信息通过串口打印出来，在电脑上通过串口助手观看打印信息。
- 如果寄存器里数据正确打印 bpr reg => none reset。
- 如果寄存器里数据正确打印 bpr reg => reset。
- 主函数里每秒打印一次日历信息。

4 案例 使用日历以及闹钟功能

4.1 功能简介

演示日历功能、闹钟功能的使用。

4.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_f4xx\examples\ertc\calendar

注：所有project都是基于keil 5而建立，若用户需要在其他编译环境上使用，请参考AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates中各种编译环境（例如IAR6/7,keil 4/5）进行简单修改即可。

4.3 软件设计

1) 配置流程

- 开启 PWC 时钟
- 使能电池供电域写保护
- 检查日历是否已经初始化，如果正确就跳过初始化，如果不正确就初始化日历以及闹钟
- 主函数里每秒打印一次日历信息
- 在 21-05-01 12:00:10 时刻发生闹钟。

2) 代码介绍

- main 函数代码描述

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

  /*初始化系统时钟 */
  system_clock_config();

  /*配置 NVIC 优先级组 */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* 初始化 ATSTART 板子资源 */
  at32_board_init();

  /*初始化串口*/
  uart_init(115200);

  /* 开启 PWC 时钟 */
```

```
crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

/* 使能电池供电域写保护 */
pwc_battery_powered_domain_access(TRUE);

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

  /* ERTC 初始化 */
  ertc_config();
}
else
{
  /* 打印信息 */
  printf("ertc has been initialized\r\n\r\n");

  /* 等待 ERTC 寄存器同步，最多需要 2 个 ERTC_CLK */
  ertc_wait_update();

  /* 清除闹钟标志 */
  ertc_flag_clear(ERTC_ALAF_FLAG);

  /* 清除 exint 挂起标志 */
  exint_flag_clear(EXINT_LINE_17);
}

/* 显示当前的 ertc 时间和闹钟时间 */
ertc_time_show();
ertc_alarm_show();

printf("\r\n");

/* 闹钟初始化 */
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);

/* 使能闹钟中断 */
nvic_irq_enable(ERTCAlarm_IRQn, 0, 1);

while(1)
```

```
  {
    /* 获取当前日历 */
    ertc_calendar_get(&time);

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

      /* 打印日期：年-月-日 */
      printf("%02d-%02d-%02d ",time.year, time.month, time.day);

      /* 打印时间：时：分：秒 */
      printf("%02d:%02d:%02d\r\n",time.hour, time.min, time.sec);
    }
  }
}
```

■ ERTC 初始化 ertc_config 函数代码描述

```
void ertc_config(void)
{
  /* 开启 PWC 时钟 */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

  /* 使能电池供电域写保护 */
  pwc_battery_powered_domain_access(TRUE);

  /* 复位电池供电域寄存器 */
  crm_battery_powered_domain_reset(TRUE);
  crm_battery_powered_domain_reset(FALSE);

#if defined (ERTC_CLOCK_SOURCE_LICK)
  /* 使能 LICK 时钟 */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_LICK, TRUE);

  /* 等待 LICK 时钟稳定 */
  while(crm_flag_get(CRM_LICK_STABLE_FLAG) == RESET)
  {
  }

  /* 选择 ERTC 时钟源 */
  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)
  /* 使能 LEXT 时钟 */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_LEXT, TRUE);

  /* 等待 LEXT 时钟稳定 */
  while(crm_flag_get(CRM_LEXT_STABLE_FLAG) == RESET)
  {
  }

  /* 选择 ERTC 时钟源 */
  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

  /* 使能 ERTC 时钟 */
  crm_ertc_clock_enable(TRUE);

  /* 复位所有 ERTC 寄存器 */
  ertc_reset();

  /* 等待 ERTC 寄存器同步，最多需要 2 个 ERTC_CLK */
  ertc_wait_update();

  /* 配置 ERTC 分频器 */
  ertc_divider_set(ertc_clk_div_a, ertc_clk_div_b);

  /* 配置 ERTC 小时格式为 24 小时格式 */
  ertc_hour_mode_set(ERTC_HOUR_MODE_24);

  /* 设置日期: 2021-05-01 */
  ertc_date_set(21, 5, 1, 5);

  /* 设置时间: 12:00:00 */
  ertc_time_set(12, 0, 0, ERTC_AM);

  /* 设置闹钟 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);

  /* 使能闹钟中断 */
  ertc_interrupt_enable(ERTC_ALA_INT, TRUE);
```

```
  /* 开启闹钟 */
  ertc_alarm_enable(ERTC_ALA, TRUE);

  ertc_flag_clear(ERTC_ALAF_FLAG);

  /* 指示 ERTC 初始化 */
  ertc_bpr_data_write(ERTC_DT1, 0x1234);
}
```

- 闹钟中断函数代码描述

```
void ERTCAlarm_IRQHandler(void)
{
  if(ertc_flag_get(ERTC_ALAF_FLAG) != RESET)
  {
    /* 显示闹钟 */
    ertc_alarm_show();

    at32_led_on(LED2);

    /* 清除闹钟标志 */
    ertc_flag_clear(ERTC_ALAF_FLAG);

    /* 清除 EXINT 标志 */
    exint_flag_clear(EXINT_LINE_17);
  }
}
```

4.4 实验效果

- 信息通过串口打印出来，在电脑上通过串口助手观看打印信息。
- 主函数里每秒打印一次日历信息。
- 在 21-05-01 12:00:10 时刻发生闹钟。

5 案例 使用 LICK 时钟并校准

5.1 功能简介

使用 LICK 时钟作为 ERTC 时钟，并通过定时器测量出 LICK 时钟频率，通过得到的频率值，调整 ERTC 分频，达到在一定范围内校准时间的效果

5.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_f4xx\examples\ertc\lick_calibration

注：所有project都是基于keil 5而建立，若用户需要在其他编译环境上使用，请参考AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates中各种编译环境（例如IAR6/7,keil 4/5）进行简单修改即可。

5.3 软件设计

1) 配置流程

- ERTC 初始化
- 配置测量 LICK 频率定时器
- 根据测量到的频率重新配置 ERTC 分频

2) 代码介绍

- main 函数代码描述

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

  /* 初始化系统时钟 */
  system_clock_config();

  /* 配置 NVIC 优先级组 */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* 初始化 AT START 板子资源 */
  at32_board_init();

  /* 初始化串口 */
  uart_init(115200);

  /* ERTC 初始化 */
  ertc_config();

  /* 测量 LICK 频率 */
```

```
  lick_freq = lick_frequency_get();

  /* 配置 ERTC 分频器 */
  /* 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)
  {
    /* 获取当前日历 */
    ertc_calendar_get(&time);

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

      /* 打印日期：年-月-日 */
      printf("%02d-%02d-%02d ",time.year, time.month, time.day);

      /* 打印时间：时：分：秒 */
      printf("%02d:%02d:%02d\r\n",time.hour, time.min, time.sec);
    }
  }
}
```

5.4 实验效果

- 信息通过串口打印出来，在电脑上通过串口助手观看打印信息。
- 通串口打印出当前测量出的 LICK 的频率以及 DIV_A、DIV_B 的值。
- 每秒钟打印一次日历。

6 案例 入侵检测

6.1 功能简介

演示入侵检测功能使用，PC13 脚当检测到一个上升沿后将触发入侵检测，当入侵事件发生时，电池供电数据寄存器将会被清除。

6.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_f4xx\examples\ertc\tamper

注：所有project都是基于keil 5而建立，若用户需要在其他编译环境上使用，请参考AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates中各种编译环境（例如IAR6/7,keil 4/5）进行简单修改即可。

6.3 软件设计

1) 配置流程

- ERTC 初始化
- 初始化入侵检测
- 初始化电池供电寄存器

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  /* 初始化系统时钟 */
  system_clock_config();

  /* 配置 NVIC 优先级组 */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* 初始化 AT START 板子资源 */
  at32_board_init();

  /* 初始化串口 */
  uart_init(115200);

  /* ERTC 初始化 */
  ertc_config();

  /* 入侵检测初始化 */
  ertc_tamper_config();

  /* 写数据到电池供电寄存器 */
```

```
  bpr_reg_write();

  /* 检查电池供电寄存器是否正确 */
  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)
  {
  }
}
```

■ 入侵检测中断处理函数代码描述

```
void TAMP_STAMP_IRQHandler(void)
{
  if(ertc_flag_get(ERTC_TP1F_FLAG) != RESET)
  {
    /* 检查电池供电寄存器是否被清除掉 */
    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");
    }

    /* 清除入侵检测标志 */
    ertc_flag_clear(ERTC_TP1F_FLAG);

    /* 清除 EXINT 标志 */
    exint_flag_clear(EXINT_LINE_21);
  }
}
```

6.4 实验效果

- 信息通过串口打印出来，在电脑上通过串口助手观看打印信息。
- 当发生入侵事件时（PC13 出现上升沿），在入侵中断函数里打印电池供电寄存器被清除的信息。

7 案例 时间戳

7.1 功能简介

演示时间戳功能使用，PC13 脚当检测到一个上升沿后将触发时间戳，在时间戳中断里打印发生事件的时刻。

7.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_f4xx\examples\ertc\time_stamp

注：所有project都是基于keil 5而建立，若用户需要在其他编译环境上使用，请参考AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates中各种编译环境（例如IAR6/7,keil 4/5）进行简单修改即可。

7.3 软件设计

1) 配置流程

- ERTC 初始化
- 初始化时间戳

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  /* 初始化系统时钟 */
  system_clock_config();

  /* 配置 NVIC 优先级组 */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* 初始化 AT START 板子资源 */
  at32_board_init();

  /* 初始化串口 */
  uart_init(115200);

  /* ERTC 初始化 */
  ertc_config();

  /* 初始化时间戳 */
  ertc_timestamp_config();

  /* 显示当前时间 */
  ertc_time_show();
```

```
  while(1)
  {

  }
}
```

- 时间戳中断处理函数代码描述

```
void TAMP_STAMP_IRQHandler(void)
{
  if(ertc_flag_get(ERTC_TSF_FLAG))
  {
    /* 显示时间戳 */
    ertc_timestamp_show();

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

    /* 清除 EXINT 标志 */
    exint_flag_clear(EXINT_LINE_21);

    /* 清除时间戳标志 */
    ertc_flag_clear(ERTC_TSF_FLAG);
  }
}
```

7.4 实验效果

- 信息通过串口打印出来，在电脑上通过串口助手观看打印信息。
- 当发生时间戳事件时（PC13 出现上升沿），在中断里打印当前保存的时间戳。

8 案例 周期唤醒定时器

8.1 功能简介

演示周期唤醒定时器功能使用。

8.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_f4xx\examples\ertc\wakeup_timer

注：所有project都是基于keil 5而建立，若用户需要在其他编译环境上使用，请参考AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates中各种编译环境（例如IAR6/7,keil 4/5）进行简单修改即可。

8.3 软件设计

1) 配置流程

- ERTC 初始化
- 初始化周期唤醒定时器

2) 代码介绍

- main 函数代码描述

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

  /* 初始化系统时钟 */
  system_clock_config();

  /* 配置 NVIC 优先级组 */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* 初始化 AT START 板子资源 */
  at32_board_init();

  /* 初始化串口 */
  uart_init(115200);

  /* ERTC 初始化 */
  ertc_config();

  /* 唤醒定时器初始化 */
  wakeup_timer_config();
```

```
  while(1)
  {
    /* 获取当前日历 */
    ertc_calendar_get(&time);

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

      /* 打印日期：年-月-日 */
      printf("%02d-%02d-%02d ",time.year, time.month, time.day);

      /* 打印时间：时：分：秒 */
      printf("%02d:%02d:%02d\r\n",time.hour, time.min, time.sec);
    }
  }
}
```

- 周期唤醒中断处理函数代码描述

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

    at32_led_on(LED2);

    /* 清除唤醒定时器标志 */
    ertc_flag_clear(ERTC_WATF_FLAG);

    /* 清除 EXINT 标志 */
    exint_flag_clear(EXINT_LINE_22);
  }
}
```

8.4 实验效果

- 信息通过串口打印出来，在电脑上通过串口助手观看打印信息。
- 每个 5 秒发生一次周期性唤醒事件，在中断里打印出信息。
- 每秒钟打印一次日历。

9 文档版本历史

表 14. 文档版本历史

日期	版本	变更
2021.10.19	2.0.0	最初版本
2022.08.29	2.0.1	增加表2，各型号ERTC寄存器读写时间差异 增加简介里闰年描述
2022.09.30	2.0.2	修正表5 LICK作为时钟源优点描述错误
2024.06.14	2.0.3	新增M412、M416
2024.12.16	2.0.4	新增F455、F456、F457、F490、WB415

重要通知 - 请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和/或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途（及其依据任何司法管辖区的法律的对应情况），或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力产品并非设计或专门用于下列用途的产品：（A）对安全性有特别要求的应用，例如：生命支持、主动植入设备或对产品功能安全有要求的系统；（B）航空应用；（C）航天应用或航天环境；（D）武器，且/或（E）其他可能导致人身伤害、死亡及财产损害的应用。如果采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，风险及法律责任仍将由采购商单独承担，且采购商应独力负责在前述应用中满足所有法律和法规要求。

经销的雅特力产品如有不同于本文档中提出的声明和/或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。