AN0003
Application Note

AT32F403 Battery Powered Domain Application

Introduction

The battery powered domain of AT32F403 series MCUs contains RTC, battery powered data register and other features. The RTC interface is used to realize calendar and clock features. This application note introduces basic functions of RTC and some application cases.

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 product:

Part number	AT32F403 series

Contents

List of Tables

List of Figures

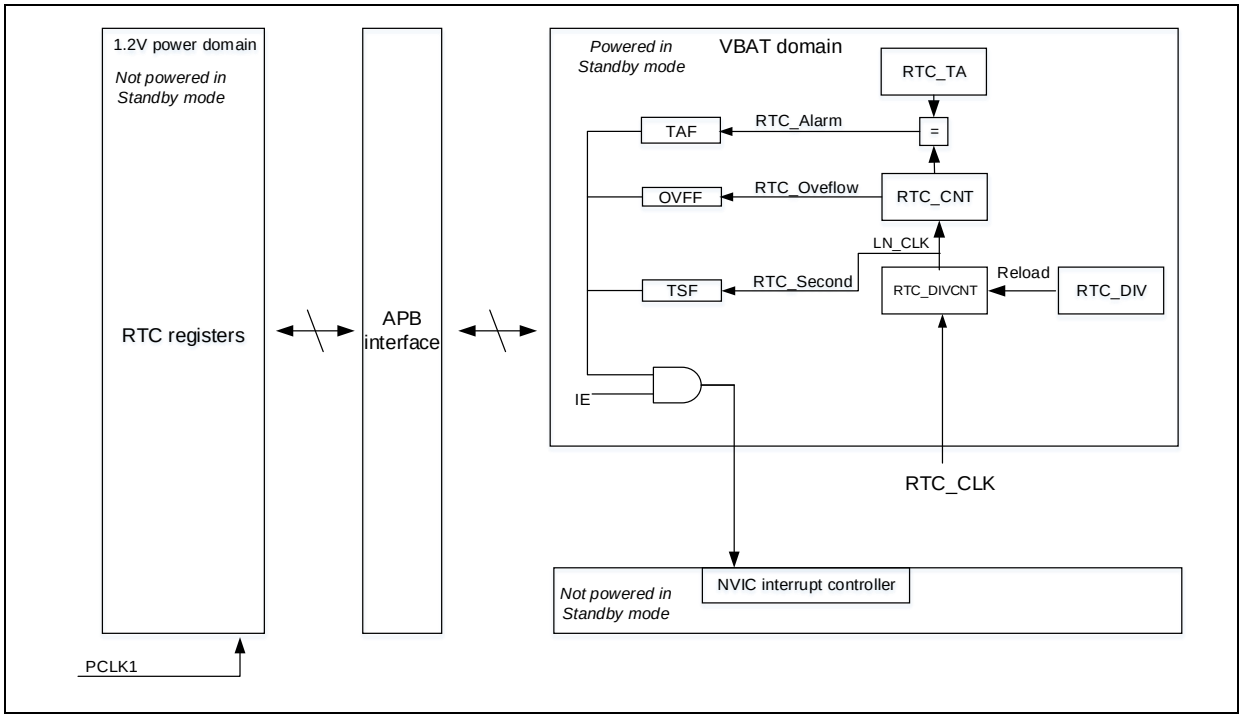
1 RTC introduction

The real-time clock has an internal 32-bit incremental counter. 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 V_{BAT} is powered.

It supports the following features:

- Calendar: 32-bit counter, with year, month, day, hour, minute and second through conversion
- Clock
- Tamper detection
- Calibration

Figure 1. RTC block diagram



2 RTC functions

2.1 Register access

Register write protection

RTC registers are write-protected after power-on reset. It can be configured only when the write protection is disabled.

The following procedures are recommended:

1) Enable PWC interface clock

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

2) Enable BPR interface clock

```
crm_periph_clock_enable(CRM_BPR_PERIPH_CLOCK, TRUE);
```

3) Disable battery powered domain write protection

```
pwc_battery_powered_domain_access(TRUE);
```

RTC register synchronization

RTC consists of a RTC counter logic in battery powered domain and an APB1 interface; therefore, there is a synchronization circuit for write/read access to RTC registers.

- Write access to registers: A new value can be written to the RTC registers only when the previous RTC configuration is completed (CFGF = 1).
- Read access to registers: When the value is uploaded from the battery powered domain to the APB1 interface, set UPDF=1. It is possible that the RTC registers are not fully synchronized if a system reset or power reset has occurred or the microcontroller has woken up from Standby or Deepsleep mode. At this time, clear UPDF flag by software and wait until UPDF=1; otherwise, an error value is returned.

RTC synchronization correlation functions

Wait until the previous RTC register configuration is completed (before write access to the register)

```
void rtc_wait_config_finish(void);
```

Wait until the RTC register update is completed (before read access to the register)

```
void rtc_wait_update_finish(void);
```

Write access to RTC registers

To enable write operation to RTC_DIV, RTC_TA and RTC_CNT registers, the first step is to enter configuration mode (CFGEN = 1). Setting CFGEN = 0 to exit configuration mode, and values in these registers are actually written to the battery powered domain, which takes at least three RTCCLK cycle to complete.

The write protection states and conditions for write operation to RTC registers are listed below.

Table 1. RTC registers

Register	Write-protected	Enter configuration mode
RTC_CTRLH, RTC_CTRLL	Yes	No
RTC_DIVH, RTC_DIVL	Yes	Yes

RTC_DIVCNTH, RTC_DIVCNTL	-	-
RTC_CNTH, RTC_CNTL	Yes	Yes
RTC_TAH, RTC_TAL	Yes	Yes

Reset of registers

RTC registers are in the battery powered domain. Set the bit BPDRST in CRM_BPDC register to trigger battery powered domain reset, or write the default value to each register through library functions to perform a reset.

RTC reset correlation functions

Battery powered domain reset

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

or function:

```
void bpr_reset(void);
```

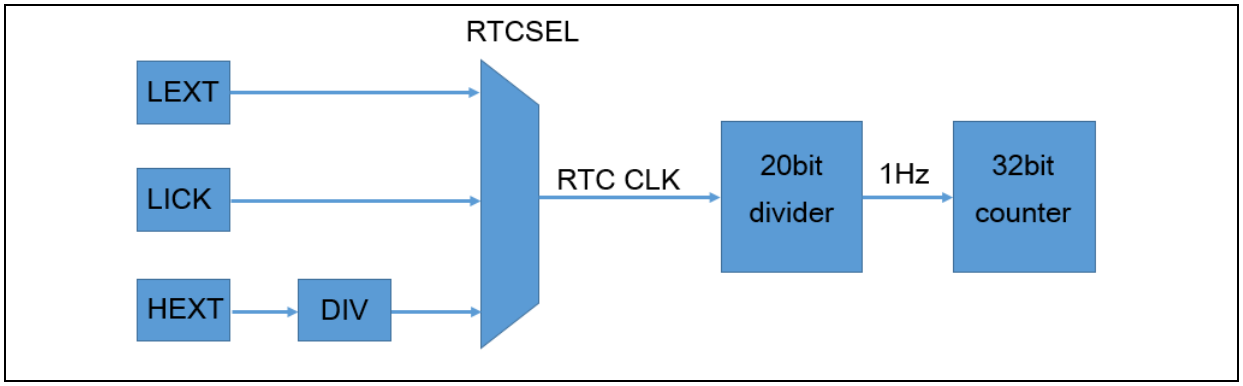
Both functions have the same features, except that *bpr_reset()* encapsulates the previous function.

2.2 Clock settings

Clock source select

RTC clock source is selected and input to the divider, and finally a 1 Hz clock is obtained to update the calendar.

Figure 2. RTC clock structure



RTC clock source can be selected from:

- LEXT: low-speed external oscillator, generally 32.768 kHz;
- LICK: low-speed internal RC oscillator, typical frequency of 40 kHz (within the range of 30~60 kHz); see the datasheet for details;
- HEXT_DIV: divided high-speed external oscillator; prescaler values are listed below.

Table 2. HEXT prescaler values

Part number	Prescaler value
AT32F403Axx	Fixed 128
AT32F407xx	Fixed 128

AT32F413xx	Fixed 128

Table 3. Comparisons of clock sources

Clock source	Frequency	Advantages	Disadvantages
LEXT	32.768 kHz	It has the highest accuracy, and can work when it is battery powered or in low-power modes.	One crystal oscillator is required, thus increasing the cost and PCB wiring area.
LICK	Typical: 40 kHz Range: 30 kHz~60 kHz	It can work when it is battery powered or in low-power modes, without the need for one crystal oscillator and reducing PCB wiring area.	Low time accuracy
HEXT	Main crystal oscillator frequency	It has a high accuracy (depending on the accuracy of the main crystal oscillator), without the need for one additional crystal oscillator and reducing PCB wiring area.	Not able to work when it is battery powered or in low-power modes

RTC clock source setting correlation functions

Select and enable the corresponding clock

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

Select RTC clock

```
void crm_rtc_clock_select(crm_rtc_clock_type value)
```

Enable RTC clock

```
void crm_rtc_clock_enable(confirm_state new_state)
```

Prescaler settings

The RTC_CLK is divided by a 20-bit prescaler to obtain a 1 Hz clock. The formula is as follows:

$$f_{1Hz} = \frac{f_{RTC\_CLK}}{DIV + 1}$$

Table 4. Example of prescaler settings

Clock source	Frequency	DIV	Calendar clock
LEXT	32.768 kHz	32767	1 Hz
LICK	Typical 40 kHz	39999	1 Hz
HEXT	8 MHz, divided by 128	62499	1 Hz

RTC divider setting correlation functions

Set RTC prescaler

```
void rtc_divider_set(uint32_t div_value);
```

Obtain RTC prescaler value

```
uint32_t rtc_divider_get(void);
```

Example of RTC clock initialization

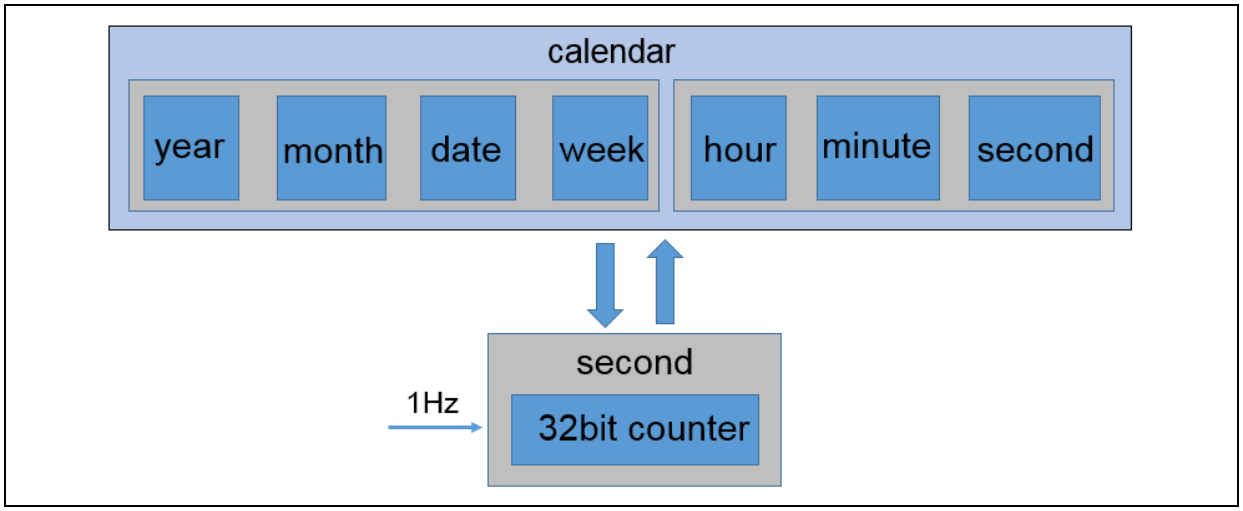
```
/* Enable LEXT clock */
crm_clock_source_enable(CRM_CLOCK_SOURCE_LEXT, TRUE);

/* Wait until the LEXT clock is stable */
while(crm_flag_get(CRM_LEXT_STABLE_FLAG) == RESET)
{
}

/* Select LEXT clock as the RTC clock */
crm_rtc_clock_select(CRM_RTC_CLOCK_LEXT);

/* Enable RTC clock */
crm_rtc_clock_enable(TRUE);

/* Configure RTC divider: DIV=32767 */
rtc_divider_set(32767);
```

2.3 Calendar

The RTC has an internal 32-bit counter that is increased by one at each second. In other words, this counter serves as a second clock. The current second value can be converted into time and date (year, month, date, week, hour, minute and second) to provide a calendar function. The time and date can be modified by modifying the counter value.

It can generate a second interrupt as required. If second interrupt is enabled (TSIEN=1), a second interrupt is generated every second.

Figure 3. Calendar conversion



Counter correlation functions

Set RTC counter value

```
void rtc_counter_set(uint32_t counter_value);
```

Obtain RTC counter value

```
uint32_t rtc_counter_get(void);
```

Convert second clock to calendar

First, set a starting time. For example,1970-1-1 00:00:00 corresponds to “0” of the counter. When the counter value is 200000, convert as follows:

- Days: 200000 / 86400 = 2
- Hours: (200000 % 86400) / 3600= 7
- Minutes: (200000 % 3600) / 60= 33
- Seconds: 200000 % 60 = 20

Therefore, the converted date is 1970-1-3 07:33:20. The calendar can be converted to second clock in a similar way.

In project\at_start_f403a\examples\rtc\calendar in BSP, there are available functions for conversion between second clock and calendar.

Set calendar value (convert calendar to second clock)

```
uint8_t rtc_time_set(calendar_type *calendar);
```

Parameters in the calendar_type structure are:

- year
- month
- day
- hour
- min
- sec
- week

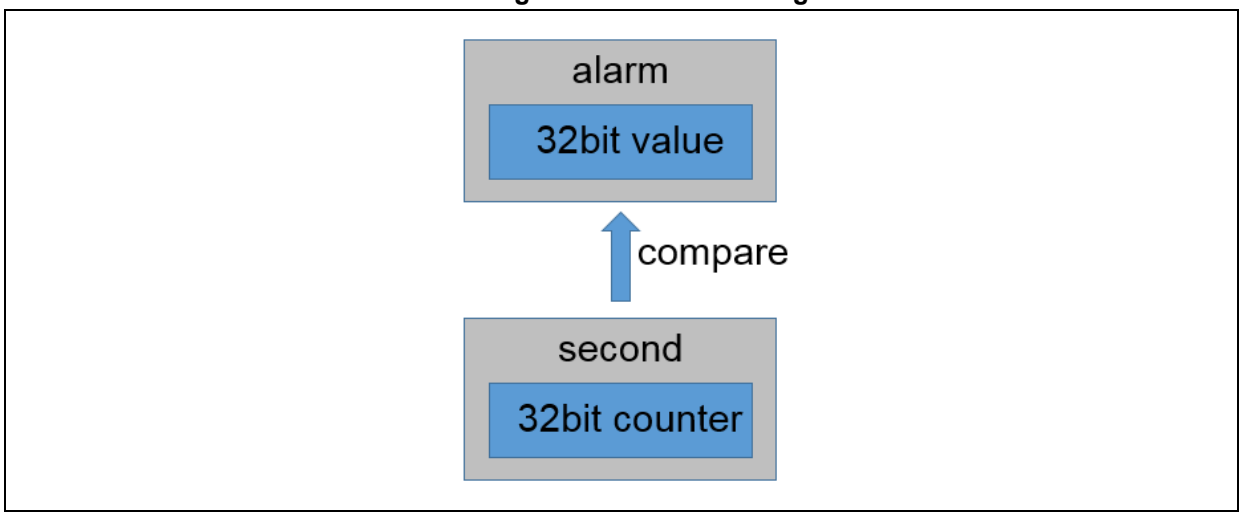
Read calendar value (convert second clock to calendar)

```
void rtc_time_get(void);
```

2.4 Alarm

The RTC alarm value is 32-bit. When the alarm value is equal to the counter value, an alarm event occurs (TAF=1). When the alarm interrupt is enabled, an alarm interrupt is generated.

Figure 4. Alarm matching



Alarm correlation functions

Alarm value setting function

```
void rtc_alarm_set(uint32_t alarm_value);
```

Interrupt enable function

```
void rtc_interrupt_enable(uint16_t source, confirm_state new_state);
```

Flag get function

```
flag_status rtc_flag_get(uint16_t flag);
```

Flag clear function

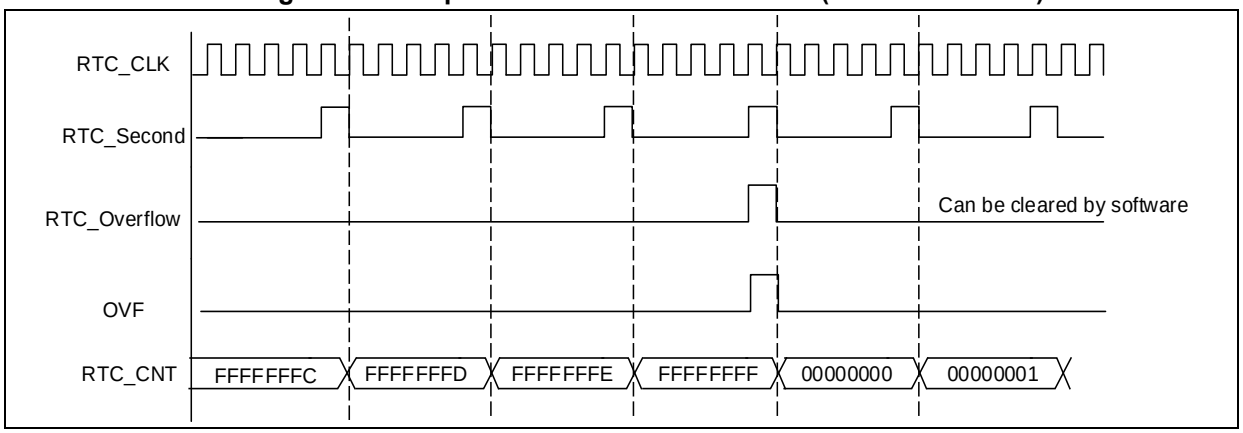
```
void rtc_flag_clear(uint16_t flag);
```

2.5 Counter value overflow

Since the counter value is 32-bit, value overflow may occur. When the counter value 0xFFFFFFFF reaches the value 0x00000000, an overflow event occurs and OVFF=1. When the alarm is enabled, the conversion between second clock and calendar is incorrect due to the overflow event. In this case, users need to handle value overflow properly.

The maximum time that 0xFFFFFFFF can represent is 136 years. For example, if the starting time is the year of 1975, the value does not overflow until the year of 2106.

Figure 5. Example of counter value overflow (division value=4)



2.6 Interrupt

If alarm, second or overflow interrupt occurs, the RTC interrupt is generated. The following two configuration modes are supported:

- RTC_IRQn interrupt vector without EXINT line: not able to wake up from DEEPSLEEP and STANDBY modes
- RTCAlarm_IRQn interrupt vector with EXINT line: able to wake up from DEEPSLEEP and STANDBY mode

The RTC alarm (no need to wake up from low-power modes), second and overflow interrupts can be enabled by the following steps:

- Enable the NVIC channel corresponding to RTC interrupt;
- Enable the corresponding RTC interrupt control bit.

The RTC alarm (need to wake up from low-power modes) can be enabled by the following steps:

- Configure EXINT 17 as interrupt mode and enable it; set rising edge as the active edge;
- Enable the NVIC channel corresponding to RTC interrupt;
- Enable the corresponding RTC interrupt control bit.

The table below lists the influence of RTC clock source, event and interrupt on wakeup from low-power modes.

Table 5. RTC wakeup from low-power modes

Clock source	Event	Wakeup from SLEEP	Wakeup from DEEPSLEEP	Wakeup from STANDBY
HEXT	Alarm (RTCAlarm_IRQn)	√	×	×
	Alarm (RTC_IRQn)	√	×	×
	Overflow	√	×	×
	Second	√	×	×
LICK	Alarm (RTCAlarm_IRQn)	√	√	√
	Alarm (RTC_IRQn)	√	×	×
	Overflow	√	×	×
	Second	√	×	×
LEXT	Alarm (RTCAlarm_IRQn)	√	√	√
	Alarm (RTC_IRQn)	√	×	×
	Overflow	√	×	×
	Second	√	×	×

Table 6. Interrupt control

Interrupt	Event flag	Interrupt enable bit
Alarm	TAF	TAIEN
Second	TSF	TSIEN
Overflow	OVFF	OVFIEN

Table 7. Events and corresponding interrupt vectors

Event	Interrupt vector	Interrupt function
Alarm (RTCAlarm_IRQn)	RTCAlarm_IRQn	RTCAlarm_IRQHandler
Alarm (RTC_IRQn)	RTC_IRQn	RTC_IRQHandler
Overflow	RTC_IRQn	RTC_IRQHandler
Second	RTC_IRQn	RTC_IRQHandler

Interrupt and event correlation functions

Interrupt enable function

```
void rtc_interrupt_enable(uint16_t source, confirm_state new_state);
```

Flag get function

```
flag_status rtc_flag_get(uint16_t flag);
```

Flag clear function

```
void rtc_flag_clear(uint16_t flag);
```

Interrupt configuration example 1: For alarm, use the RTCAlarm_IRQn interrupt vector.

```
  /* 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: RTCAlarm_IRQn */
  nvic_irq_enable(RTCAlarm_IRQn, 0, 1);

  /* Set alarm value */
  rtc_alarm_set(seccount);

  /* Enable alarm interrupt */
  rtc_interrupt_enable(RTC_TA_INT, TRUE);
```

Interrupt handler

```
void RTCAlarm_IRQHandler(void)
{
  if(rtc_flag_get(RTC_TA_FLAG) != RESET)
  {
    /* Clear alarm flag */
    rtc_flag_clear(RTC_TA_FLAG);

    /* Clear EXINT line 17 flag */
    exint_flag_clear(EXINT_LINE_17);
  }
}
```

Interrupt configuration example 2: For alarm, use the RTC_IRQn interrupt vector.

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

  /* Set alarm value */
  rtc_alarm_set(seccount);

  /* Enable alarm interrupt */
  rtc_interrupt_enable(RTC_TA_INT, TRUE);
```

Interrupt handler

```
void RTC_IRQHandler(void)
{
  if(rtc_flag_get(RTC_TA_FLAG) != RESET)
  {
    /* Clear alarm flag */
    rtc_flag_clear(RTC_TA_FLAG);
  }
}
```

3 Battery powered domain

3.1 Battery powered registers

There are 42 x 16-bit registers in the battery powered domain. These registers can save data when they are powered only by the battery, and they are reset by battery powered domain reset or a tamper event, instead of a system reset. Read protection of these registers must be disabled before register reset, in the same way detailed in Section 2.1.

Battery powered domain data operation correlation functions

Write operation to battery powered data register

```
void bpr_data_write(bpr_data_type bpr_data, uint16_t data_value);
```

Read operation to battery powered data register

```
uint16_t bpr_data_read(bpr_data_type bpr_data);
```

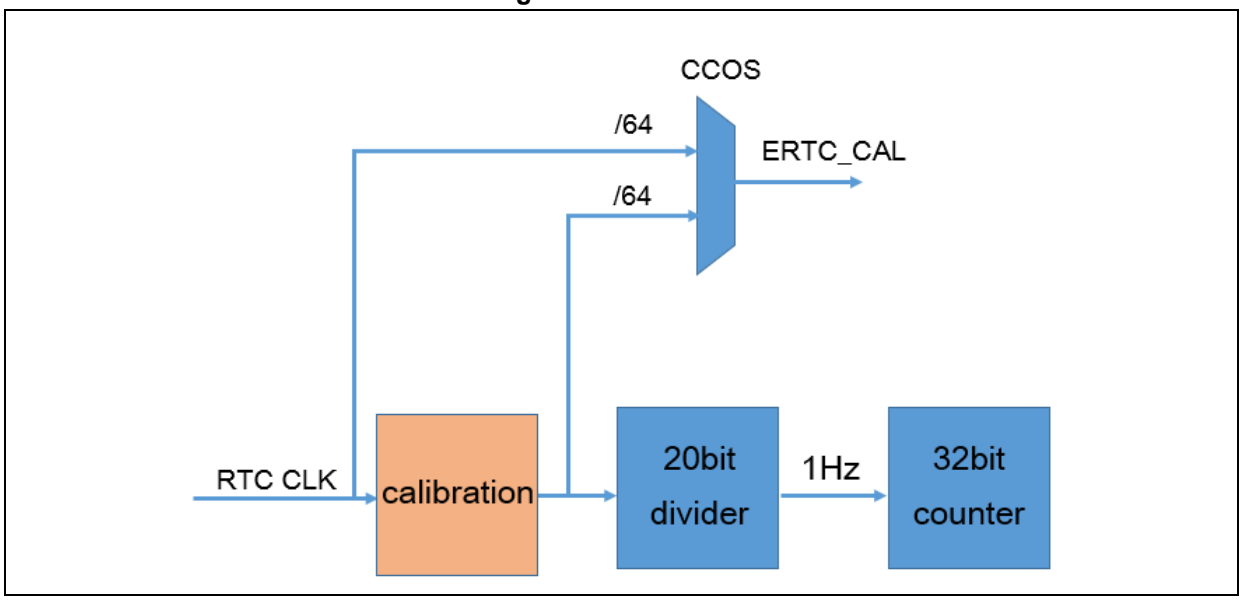
Battery powered domain reset

```
void bpr_reset(void);
```

3.2 RTC calibration

The battery powered domain supports RTC calibration, which is configured through the RTC_CALVAL register.

Figure 6. RTC calibration



When the RTC_CLK is 32.768 kHz, the calibration cycle is 2^{20} x RTC_CLK (about 32 seconds). The CALVAL[7:0] value specifies the number of pulses ignored in 2^{20} x RTC_CLK (maximum 127 pulses can be ignored), which can slow down the clock in the range of 0~121 ppm.

It is allowed to divide the RTC clock (before or after calibration) by 64 and then output to PC13 pin.

Calibration setting correlation functions

Calibration value setting function

```
void bpr_rtc_clock_calibration_value_set(uint8_t calibration_value);
```

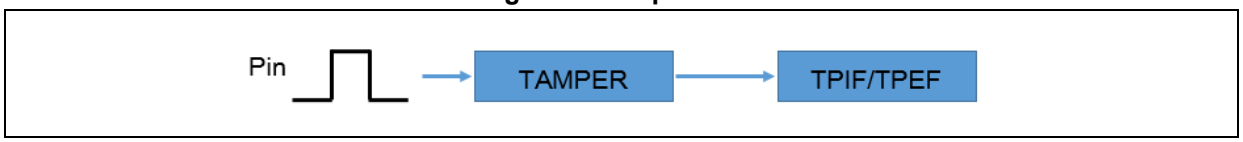
Calibration clock output setting function

```
void bpr_rtc_output_select(bpr_rtc_output_type output_source);
```

3.3 Tamper detection

The battery powered domain provides a set of TAMPER pin. Once a tamper event is detected, the TPEF bit will be set to 1, and the RTC_BPRx register will be cleared accordingly. If the tamper interrupt is enabled, an interrupt will be generated, with the TPIF bit being set to 1. The tamper detection pin is fixed to PC13.

Figure 7. Tamper detection



The tamper detection mode is divided into high-level and low-level detections.

Tamper detection correlation functions

Set tamper detection active level

```
void bpr_tamper_pin_active_level_set(bpr_tamper_pin_active_level_type active_level);
```

Tamper detection enable

```
void bpr_tamper_pin_enable(confirm_state new_state);
```

Tamper detection flag get

```
flag_status bpr_flag_get(uint32_t flag);
```

Tamper detection flag clear

```
void bpr_flag_clear(uint32_t flag);
```

Tamper interrupt enable

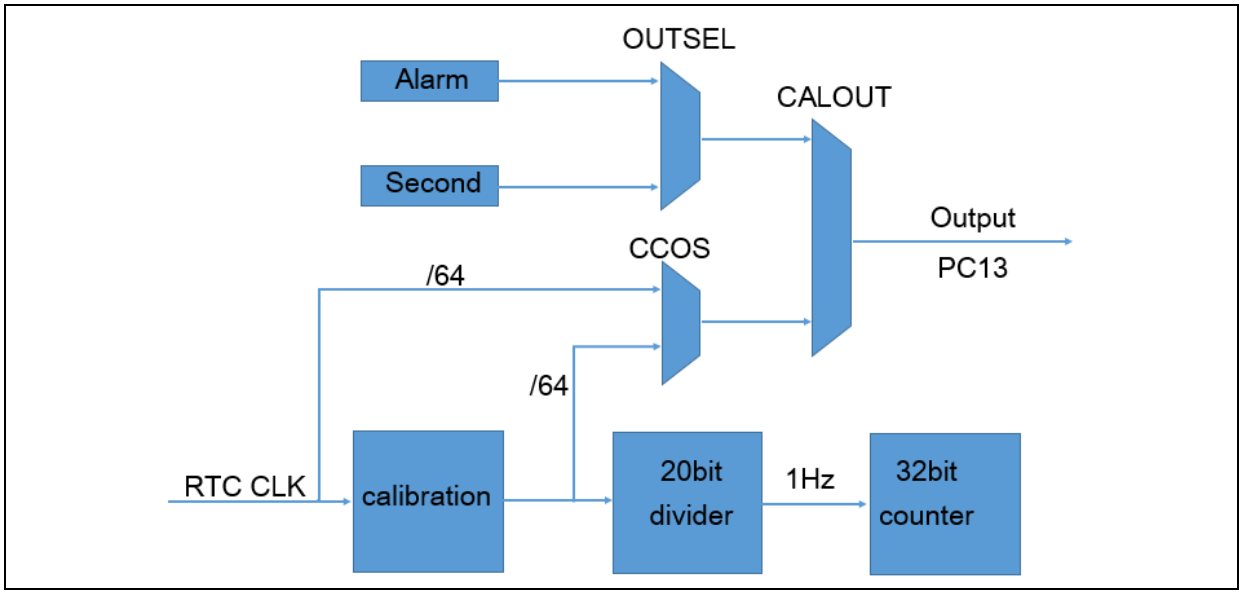
```
void bpr_interrupt_enable(confirm_state new_state);
```

3.4 Event output

The battery powered domain provides a set of MUX outputs. The following events can be output through PC13 pin:

- Calibration output: output of frequency divided by 64 before/after calibration
- Event output: alarm event, second event

Figure 8. Event output



If the output mode is event output (alarm event, second event), a pulse is output when the corresponding event occurs, and the pulse width is one RTC clock cycle.

Event output correlation functions

Set and enable event output

```
void bpr_rtc_output_select(bpr_rtc_output_type output_source);
```

4 Notes on RTC usage

4.1 Read/write operation to battery powered registers

Before performing read and write operations to battery powered registers (BPR_DT, RTC registers, and the BPDC register of CRM), enable PWC and BPR clocks, unlock write protection of battery powered domain (BPWEN=1) and execute the following codes:

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

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

In the below example, enable the PWC and BPR clocks and unlock write protection of battery powered domain (BPWEN=1), and then read the BPR_DT register, write the BPDC register of CRM and write the RTC register.

```
/* Enable PWC and BPR clocks */
crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_BPR_PERIPH_CLOCK, TRUE);

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

/* Check the RTC for initialization */
if(bpr_data_read(BPR_DATA1) != 0x1234)
{
  /* Reset battery powered registers */
  bpr_reset();

  /* Enable LEXT clock */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_LEXT, TRUE);
  /* Wait until the LEXT clock becomes stable */
  while(crm_flag_get(CRM_LEXT_STABLE_FLAG) == RESET);
  /* Select RTC clock source */
  crm_rtc_clock_select(CRM_RTC_CLOCK_LEXT);

  /* Enable RTC clock */
  crm_rtc_clock_enable(TRUE);
```

```
    /* Waite for RTC register synchronization */
    rtc_wait_update_finish();

    /* Wait for the completion of the previous register configuration synchronization */
    rtc_wait_config_finish();

    /* Configure RTC divider */
    rtc_divider_set(32767);

     /* Wait for the completion of the previous register configuration synchronization */
    rtc_wait_config_finish();

    /* Set time */
    rtc_time_set(calendar);

    /* Write the flag to battery powered registers */
    bpr_data_write(BPR_DATA1, 0x1234);
}
```

4.2 Notes on sBPWEN bit operation

Do not enable or disable the BPWEN bit in the program. If you want to operate BPR data, enable the PWC clock, BPR clock and BPWEN bit during code initialization.

```
/* Enable PWC and BPR clocks */
crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_BPR_PERIPH_CLOCK, TRUE);

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

Then, BPR DT can be modified directly (do not enable or disable the BPWEN bit)

```
bpr_data_write(BPR_DATA1, 0x1234);
```

5 Case 1: Use of calendar and alarm features

5.1 Function overview

Demonstrate the calendar and alarm features.

5.2 Resources preparation

1) Hardware environment

   AT-START BOARD of the corresponding part model

2) Software environment

   project\at_start_f4xx\examples\rtc\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.

5.3 Software design

1) Configuration process:
   - Enable PWC and BPR clocks;
   - Unlock battery powered domain write protection;
   - Check if the calendar is initialized (if the conversion between clock and calendar is correct, the calendar does not need to be initialized; otherwise, initialize the calendar and alarm);
   - Print the calendar information every second in the main function;
   - Alarm event occurs at 21-05-01 12:00:05.
2) Code
   - main function code

```
int main(void)
{
  calendar_type time_struct;

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

  /* Initialize system clock */
  system_clock_config();

  /* Initialize ATSTART board */
  at32_board_init();

  /* Initialize the serial port */
  uart_print_init(115200);

  /* RTC initialization */
  time_struct.year  = 2021;
  time_struct.month = 5;
  time_struct.date  = 1;
```

```
  time_struct.hour    = 12;
  time_struct.min     = 0;
  time_struct.sec     = 0;
  rtc_init(&time_struct);

  /* Alarm initialization */
  alarm_init();

  printf("initial ok\r\n");

  while(1)
  {
    /* Second updated */
    if(rtc_flag_get(RTC_TS_FLAG) != RESET)
    {
      at32_led_toggle(LED3);

      /* Get the current calendar */
      rtc_time_get();

      /* Print date: Y-M-D */
      printf("%d/%d/%d ", calendar.year, calendar.month, calendar.date);

      /* Print time: H:M:S */
      printf("%02d:%02d:%02d %s\r\n", calendar.hour, calendar.min, calendar.sec,
weekday_table[calendar.week]);

      /* Wait for the completion of the previous register configuration synchronization */
      rtc_wait_config_finish();

      /* Clear second clock flag */
      rtc_flag_clear(RTC_TS_FLAG);

      /* Wait for the completion of register configuration synchronization */
      rtc_wait_config_finish();
    }
  }
}
```

■ RTC initialization rtc_init function code

```
uint8_t rtc_init(calendar_type *calendar)
{
  /* Enable PWC and BPR clocks */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_BPR_PERIPH_CLOCK, TRUE);
```

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

/* Check if RTC is initialized */
if(bpr_data_read(BPR_DATA1) != 0x1234)
{
  /* Reset battery powered registers */
  bpr_reset();

  /* Enable LEXT clock */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_LEXT, TRUE);
  /* Wait until the LEXT clock becomes stable */
  while(crm_flag_get(CRM_LEXT_STABLE_FLAG) == RESET);
  /* Select RTC clock source */
  crm_rtc_clock_select(CRM_RTC_CLOCK_LEXT);

  /* Enable RTC clock */
  crm_rtc_clock_enable(TRUE);

  /* Wait for RTC register synchronization */
  rtc_wait_update_finish();

  /* Wait for the completion of the previous register configuration synchronization */
  rtc_wait_config_finish();

  /* Configure RTC divider*/
  rtc_divider_set(32767);

   /* Wait for the completion of the previous register configuration synchronization */
  rtc_wait_config_finish();

  /* Set time */
  rtc_time_set(calendar);

  /* Write flag to battery powered register */
  bpr_data_write(BPR_DATA1, 0x1234);

  return 1;
}
else
{
  /* Wait for RTC register synchronization */
  rtc_wait_update_finish();

  /* Wait for the completion of the previous register configuration synchronization */
  rtc_wait_config_finish();
```

```
    return 0;
  }
}
```

- Alarm interrupt function code

```
void RTC_IRQHandler(void)
{
  /*Set alarm interrupt flag */
  if(rtc_flag_get(RTC_TA_FLAG) != RESET)
  {
    at32_led_toggle(LED4);

    /* Clear alarm interrupt flag */
    rtc_flag_clear(RTC_TA_FLAG);
  }
}
```

5.4 Test result

- Information is printed through serial port, and users can get the information from serial assistant interface on PC.
- The calendar information is printed every second in the main function.
- An alarm event occurs at 21-05-01 12:00:05; at this time, LED4 is ON.

6 Case 2: Use LICK clock and calibrate

6.1 Function overview

Select LICK clock as the RTC clock, and use a timer to measure the LICK clock frequency, and then adjust RTC divider according to the measured frequency, so that to realize time calibration within a certain range.

6.2 Resources preparation

1) Hardware environment

   AT-START BOARD of the corresponding part model

2) Software environment

   project\at_start_f4xx\examples\rtc\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.

6.3 Software design

1) Configuration process

   - Initialize RTC;
   - Configure the timer used to measure LICK frequency;

- Re-configure RTC divider according to the measured frequency.

2) Code

- main function code

```
int main(void)
{
  tmr_input_config_type tmr_ic_init_structure;

  /* Initialize system clock */
  system_clock_config();

  /* Initialize ATSTART board */
  at32_board_init();

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

  /* RTC initialization */
  rtc_configuration();

  printf("\r\n\nstart calib\r\n\r\n");

  /* Get system clock frequency */
  crm_clocks_freq_get(&crm_clocks);

  /* Enable the timer */
  crm_periph_clock_enable(CRM_TMR5_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_IOMUX_PERIPH_CLOCK, TRUE);

  /* Connect LICK to the timer */
  gpio_pin_remap_config(TMR5CH4_MUX, TRUE);

  /* Timer initialization */
  tmr_base_init(TMR5, 0xFFFF, 0);
  tmr_cnt_dir_set(TMR5, TMR_COUNT_UP);
  tmr_clock_source_div_set(TMR5, TMR_CLOCK_DIV1);

  /* Timer initialization */
  tmr_input_default_para_init(&tmr_ic_init_structure);
  tmr_ic_init_structure.input_channel_select = TMR_SELECT_CHANNEL_4;
  tmr_ic_init_structure.input_polarity_select = TMR_INPUT_RISING_EDGE;
  tmr_ic_init_structure.input_mapped_select = TMR_CC_CHANNEL_MAPPED_DIRECT;
  tmr_ic_init_structure.input_filter_value = 0;
  tmr_input_channel_init(TMR5, &tmr_ic_init_structure, TMR_CHANNEL_INPUT_DIV_1);

  /* Initialize variables */
```

```
  operationcomplete = 0;

  /* Enable timer input capture */
  tmr_counter_enable(TMR5, TRUE);

  /* Clear timer flag */
  tmr_flag_get(TMR5, TMR_C4_FLAG);

  /* Enable timers */
  tmr_interrupt_enable(TMR5, TMR_C4_INT, TRUE);

  /* Interrupt initialization */
  nvic_configuration();

  /* Wait for the completion of measurement*/
  while(operationcomplete != 2);

  /* Calculate LICK frequency */
  if(periodvalue != 0)
  {
    lickfreq = (uint32_t)((uint32_t)(crm_clocks.apb1_freq * 2) / (uint32_t)periodvalue);
  }

  printf("apb1_freq      = %d\r\n", crm_clocks.apb1_freq);
  printf("period_value = %d\r\n", periodvalue);
  printf("lick_freq      = %d\r\n", lickfreq);

  /* Adjust RTC divider */
  rtc_divider_set((lickfreq - 1));

  /* Wait for the completion of register write operation */
  rtc_wait_config_finish();

  /* turn on led2 */
  at32_led_on(LED2);

  while(1)
  {
  }
}
```

6.4 Test result

- Information is printed through serial port, and users can get the information from serial assistant interface on PC.
- The measured LICK frequency and division factor are printed on the serial port.

- The calendar information is printed every second.

7 Case 3: Read/write operation to battery powered registers

7.1 Function overview

Perform read and write operations to battery powered registers (RTC_BPRx).

7.2 Resources preparation

1) Hardware environment

   AT-START BOARD of the corresponding part model

2) Software environment

   project\at_start_f4xx\examples\bpr\bpr_data

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
   - Enable PWC and BPR clocks;
   - Unlock the battery powered domain write protection;
   - Check if the battery powered domain data is correct;
   - Disable PWC and BPR clocks to reduce power consumption.
2) Code
   - Main function code

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

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

  /* Enable PWC and BPR clocks */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_BPR_PERIPH_CLOCK, TRUE);

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

  /* Clear tamper detection flag */
  bpr_flag_clear(BPR_TAMPER_EVENT_FLAG);

  /* Check battery powered domain data */
```

```
  if(bpr_reg_check() == TRUE)
  {
    printf("bpr reg => none reset\r\n");
  }
  else
  {
    printf("bpr reg => reset\r\n");
  }

  /* Reset battery powered registers */
  bpr_reset();

  /* Write operation to battery powered registers */
  bpr_reg_write();

  /* Check battery powered domain data */
  if(bpr_reg_check() == TRUE)
  {
    printf("write bpr reg ok\r\n");
  }
  else
  {
    printf("write bpr reg fail\r\n");
  }

  while(1)
  {
  }
}
```

7.4 Test result

- Information is printed through serial port, and users can get the information from serial assistant interface on PC.
- If data in the register is correct, “bpr reg => none reset” will be printed.
- If data in the register is correct, “bpr reg => reset” will be printed.
- The calendar information is printed every second in the main function.

8 Case 4: Tamper detection

8.1 Function overview

Demonstrate the tamper detection feature. When a rising edge is detected on PC13 pin, the tamper detection is triggered. If a tamper event occurs, battery powered registers will be cleared.

8.2 Resources preparation

1) Hardware environment

   AT-START BOARD of the corresponding part model

2) Software environment

   project\at_start_f4xx\examples\bpr\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.

8.3 Software design

1) Configuration process
   - Initialize RTC;
   - Initialize tamper detection;
   - Initialize battery powered registers.

2) Code
   - main function code

```
int main(void)
{
  /* Configure NVIC priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* Initialize system clock */
  system_clock_config();

  /* Initialize ATSTART board */
  at32_board_init();

  /* Tamper interrupt configuration */
  nvic_irq_enable(TAMPER_IRQn, 0, 0);

  /* Enable PWC and BPR clocks */
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_BPR_PERIPH_CLOCK, TRUE);

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

  /* Disable tamper detection function */
  bpr_tamper_pin_enable(FALSE);
```

```
  /* Disable tamper interrupt */
  bpr_interrupt_enable(FALSE);

  /* Set tamper detection pin active low */
  bpr_tamper_pin_active_level_set(BPR_TAMPER_PIN_ACTIVE_LOW);

  /* Clear tamper detection flag */
  bpr_flag_clear(BPR_TAMPER_EVENT_FLAG);

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

  /* Enable tamper interrupt */
  bpr_interrupt_enable(TRUE);

  /* Enable tamper detection */
  bpr_tamper_pin_enable(TRUE);

  /* Check the value of battery powered registers */
  if(bpr_reg_check() == TRUE)
  {
    at32_led_on(LED2);
  }
  else
  {
    at32_led_off(LED2);
  }

  while(1)
  {
  }
}
```

■ Tamper interrupt handler code

```
void TAMPER_IRQHandler(void)
{
  if(bpr_flag_get(BPR_TAMPER_INTERRUPT_FLAG) != RESET)
  {
    /* Check if battery powered registers are cleared */
    if(bpr_reg_judge() == 0)
    {
      /* ok, bpr registers are reset as expected */
      at32_led_on(LED3);
    }
    else
```

```
    {
      /* bpr registers are not reset */
      at32_led_off(LED3);
    }

    /* Clear tamper interrupt flag */
    bpr_flag_clear(BPR_TAMPER_INTERRUPT_FLAG);

    /* Clear tamper detection event flag */
    bpr_flag_clear(BPR_TAMPER_EVENT_FLAG);

    /* Disable tamper detection */
    bpr_tamper_pin_enable(FALSE);

    /* Enable tamper detection */
    bpr_tamper_pin_enable(TRUE);
  }
}
```

8.4 Test result

- Information is printed through serial port, and users can get the information from serial assistant interface on PC.
- If a tamper event occurs (low level on PC13), LED3 will be ON, indicating that battery powered registers are cleared.

9 Revision history

Table 8. Document revision history

Date	Version	Revision note
2022.02.11	2.0.0	Initial release

IMPORTANT NOTICE – PLEASE READ CAREFULLY

Purchasers are solely responsible for the selection and use of ARTERY's products and services; ARTERY assumes no liability for purchasers' selection or use of the products and the relevant services.

No license, express or implied, to any intellectual property right is granted by ARTERY herein regardless of the existence of any previous representation in any forms. If any part of this document involves third party's products or services, it does NOT imply that ARTERY authorizes the use of the third party's products or services, or permits any of the intellectual property, or guarantees any uses of the third party's products or services or intellectual property in any way.

Except as provided in ARTERY's terms and conditions of sale for such products, ARTERY disclaims any express or implied warranty, relating to use and/or sale of the products, including but not restricted to liability or warranties relating to merchantability, fitness for a particular purpose (based on the corresponding legal situation in any unjudicial districts), or infringement of any patent, copyright, or other intellectual property right.

ARTERY's products are not designed for the following purposes, and thus not intended for the following uses: (A) Applications that have specific requirements on safety, for example: life-support applications, active implant devices, or systems that have specific requirements on product function safety; (B) Aviation applications; (C) Aerospace applications or environment; (D) Weapons, and/or (E) Other applications that may cause injuries, deaths or property damages. Since ARTERY products are not intended for the above-mentioned purposes, if purchasers apply ARTERY products to these purposes, purchasers are solely responsible for any consequences or risks caused, even if any written notice is sent to ARTERY by purchasers; in addition, purchasers are solely responsible for the compliance with all statutory and regulatory requirements regarding these uses.

Any inconsistency of the sold ARTERY products with the statement and/or technical features specification described in this document will immediately cause the invalidity of any warranty granted by ARTERY products or services stated in this document by ARTERY, and ARTERY disclaims any responsibility in any form.