Application Note

Document No.: AN1162

APM32F402_F403_RTC Timed Wake-up from STOP Mode

Version: V1.0

1 Introduction

This Application Note aims to guide users on how to use the Real-Time Clock (RTC) function to achieve timed wake-up from the STOP low-power mode on the APM32F402/F403 microcontrollers. The document details various low-power modes of the APM32F402 and provides specific code examples, configuration steps, and verification methods for implementing the RTC wake-up function to help users effectively reduce power consumption in practical applications.

This application note applies to the Geehy APM32F402/APM32F403. The APM32F402 is used as an example throughout this document.

Contents

2 Introduction to Low-power Mode

Arranged from highest to lowest power consumption, the APM32F402 has four operating modes: RUN, SLEEP, STOP, and STANDBY. After power-on reset, when the APM32F402 is in the running state, if the kernel does not need to continue running, users can choose to enter the three low-power modes (SLEEP, STOP, STANDBY) to reduce power consumption. In these three modes, the power consumption, wake-up time, and wake-up sources are different. Users need to select the best low-power mode according to the application requirements.

Table 1 Difference among "SLEEP Mode, STOP Mode and STANDBY Mode"

Mode	Description	Entry method	Wake-up mode	Voltage regulator	Effect on 1.2V area clock	Effect on VDD area clock
Sleep	Arm® Cortex®-M4F core stops, and all peripherals including the core peripheral are still working	Call WFI instruction	Any interrupt	On	Only the core clock is disabled, and it has no effect on other clocks and ADC clocks	None
		Call WFE instruction	Wake-up event	On		None
Stop	All clocks have stopped	PDDS_CFG and LPDSCFG bits + SLEEPDEEP bit + WFI or WFE	Any external interrupt	Enable or be in low-power mode	Disable all clocks of 1.2V area	HSICLK and HSECLK oscillators are disabled
Standby	1.2V power is off	PDDS_CFG bit + SLEEPDEEP bit + WFI or WFE	Rising edge of WKUP pin, RTC alarm event, external reset on NRST pin, IWDT reset	OFF	Standby	1.2V power is off

2.1 SLEEP mode

In SLEEP mode, only the kernel clock is turned off, and the kernel stops running, but all on-chip peripherals and kernel peripherals still operate normally. There are two ways to enter the SLEEP mode, and the way of entering also determines the way of waking up from the SLEEP mode, namely WFI (Wait For Interrupt) and WFE (Wait For Event).

Table 2 Characteristics of SLEEP Mode

Characteristics	Description
Enter	Enter the SLEEP mode immediately by executing the WFI or WFE instruction; when SLEEPONEINT is set to 0 and the WFI or WFE instruction is executed, enter the SLEEP mode immediately; when SLEEPONEINT is set to 1, the system exits the interrupt program first and then enters the SLEEP mode immediately.

Wake-up	If WFI instruction is executed to enter the SLEEP mode, wake up by any interrupt; if WFE instruction is executed to enter the SLEEP mode, wake up by an event.
Sleep	The core stops working, all peripherals are still running, and the data in the core registers and memory before sleep are saved.
Wakeup delay	None
Characteristics	**Description**
After wake-up	To wake up by interrupt, first enter the interrupt, exit the interrupt, and then execute the program after WFI instruction; to wake up by event, directly execute the program after WFE instruction.

In the APM32F402, we can use the PMU_EnterSleepMode function to enter the SLEEP mode. Its parameters PMU_SLEEPENTRY_WFI and PMU_SLEEPENTRY_WFE determine how we enter and wake up from the SLEEP mode, representing interrupt and event respectively.

2.2 STOP mode

In STOP mode, on the basis of the SLEEP mode, all other clocks are further turned off, so all peripherals stop working. However, since part of the power supply in the 1.2V area is not turned off and the information in the kernel registers and memory is retained, after waking up from the STOP mode and restarting the clock, the code can continue to be executed from the point where it stopped last time. The STOP mode can be woken up by any external interrupt (EINT). In the STOP mode, the voltage regulator can be selected to operate in either the normal mode or the low-power mode.

Table 3 Characteristics of STOP Mode

Characteristics	**Description**
Enter	Set the SLEEPDEEP bit in the core register to 1 and the PDDS_CFG bit in the register PMU_CTRL to 0, and then execute the WFI or WFE instruction to immediately enter the STOP mode; when the LPDSCFG bit in the register PMU_CTRL is set to 0, the voltage regulator operates in the normal mode; when the LPDSCFG bit in the register PMU_CTRL is set to 1, the voltage regulator operates in the low-power mode.
Wake-up	If WFI instruction is executed to enter the SLEEP mode, wake up by any interrupt; if WFE instruction is executed to enter the SLEEP mode, wake up by an event.
Stop	The core and the peripheral will stop working, and the data in the core register and memory before stop will be saved.
Wakeup delay	Wake-up time of HSICLK oscillator + wake-up time of voltage regulator from low-power mode.
After wake-up	To wake up by interrupt, first enter the interrupt, exit the interrupt, and then execute the program after WFI instruction; to wake up by event, directly execute the program after WFE instruction.

In the APM32F402, we can enter the STOP mode through the PMU_EnterSTOPMode function. Its parameter 1 determines whether the voltage regulator is in the normal mode (PMU_REGULATOR_ON) or the low-power mode (PMU_REGULATOR_LOWPOWER). Its parameter 2 determines whether to enter the STOP mode through an interrupt or an event,

which are PMU_STOP_ENTRY_WFI and PMU_STOP_ENTRY_WFE respectively.

2.3 STANDBY mode

In addition to turning off all clocks, the STANDBY mode also completely turns off the power supply in the 1.2V area. That is to say, after waking up from the STANDBY mode, there is no running record of the previous code. We can only reset the chip, re-detect the boot conditions, and execute the program from the beginning. There are four ways to wake it up, namely the rising edge of the WKUP (PA0) pin, the RTC alarm event, the reset of the NRST pin, and the IWDG (Independent Watchdog) reset. The various characteristics of the STANDBY mode are shown in the following table:

Table 4 Characteristics of STANDBY Mode

Characteristics	Description
Enter	SLEEPDEEP bit of the core register is set to 1, PDDS_CFG bit of the register PMU_CTRL is set to 1, WUEFLG bit is set to 0 and when WFI or WFE instruction is executed, it will enter the STANDBY mode immediately.
Wake-up	Wake up by rising edge of WKUP pin, RTC alarm, wake-up, tamper event or NRST pin external reset and IWDT reset.
Standby	The core and the peripheral will stop working, and the data in the core register and memory will be lost.
Wakeup delay	Chip reset time.
After wake-up	The program starts executing from the beginning.

3 Implementation of RTC Timed Wake-up for STOP Mode

According to the previous description, the STOP mode can be woken up by any interrupt. In the EINT section, we can see that the RTC Alarm event is mapped to the EINT 17 line. Therefore, the STOP mode can be woken up periodically by configuring the RTC alarm.

3.1 RTC alarm configuration

In the RTC alarm configuration, we use the internal LSI as the RTC clock source, configure a 5-second alarm, and associate the RTC Alarm event with the EINT 17 line.

Note: Before implementing the code, the alarm time macro ALARM_TIME_INTERVAL needs to be defined first.

```
#define ALARM_TIME_INTERVAL (5U)
```

The following is the RTC initialization configuration code:

```
void RTC_Config_Init(void)
{
    EINT_Config_T EINT_Configure;

    /* Enable BKP and PWR clocks */
    RCM_EnableAPB1PeriphClock(RCM_APB1_PERIPH_PMU | RCM_APB1_PERIPH_BAKR);
    /* Allow access to the backup domain */
    PMU_EnableBackupAccess();
    /* Reset backup domain */
    BAKPR_Reset();

    /* Enable LSI */
    RCM_EnableLSI();
    /* Wait until LSI is ready */
    while(RCM_ReadStatusFlag(RCM_FLAG_LSIRDY) == RESET);
    /* Select LSI as the RTC clock source */
    RCM_ConfigRTCCLK(RCM_RTCCLK_LSI);
    /* Enable RTC clock */
    RCM_EnableRTCCLK();

    /* Wait for RTC register synchronization */
    RTC_WaitForSynchro();
    /* Wait for the last write operation on the RTC register to complete */
    RTC_WaitForLastTask();
    /* Enable RTC alarm interrupt */
    RTC_EnableInterrupt(RTC_INT_ALR);
    /* Wait for the last write operation on the RTC register to complete */
    RTC_WaitForLastTask();
```

```
    /* Set the RTC prescaler value: Set the RTC period to 1s */
    RTC_ConfigPrescaler(40000);
    /* Wait for the last write operation on the RTC register to complete */
    RTC_WaitForLastTask();
    /* Set the RTC counter value to 0 */
    RTC_ConfigCounter(0U);
    /* Wait for the last write operation on the RTC register to complete */
    RTC_WaitForLastTask();
    /* Set the RTC alarm value to 5s */
    RTC_ConfigAlarm(ALARM_TIME_INTERVAL);
    /* Wait for the last write operation on the RTC register to complete */
    RTC_WaitForLastTask();

    /* EXTI configuration */
    EINT_Reset();
    EINT_Configure.line = EINT_LINE_17;
    EINT_Configure.lineCmd = ENABLE;
    EINT_Configure.mode = EINT_MODE_INTERRUPT;
    EINT_Configure.trigger = EINT_TRIGGER_RISING;
    EINT_Config(&EINT_Configure);

    /* Flag clearing */
    RTC_ClearStatusFlag(RTC_FLAG_ALR);
    EINT_ClearIntFlag(EINT_LINE_17);

    /* NVIC configuration */
    NVIC_EnableIRQRequest(RTC_Alarm_IRQn, 1, 1);
}
```

3.2 RTC alarm interrupt configuration

In the RTC alarm interrupt service function, after detecting the RTC alarm interrupt, we first clear the alarm flag and the flag of EINT 17, and then reset the RTC counter value and the alarm value to implement a periodic 5-second alarm. The following is the code for the RTC alarm interrupt service function:

```
void RTC_Alarm_IRQHandler(void)
{
    if(RTC_ReadIntFlag(RTC_INT_ALR) == SET)
    {
        /* Clear the RTC alarm and EXTI_Line17 interrupt flags */
        RTC_ClearIntFlag(RTC_INT_ALR);
        EINT_ClearIntFlag(EINT_LINE_17);
```

```
        /* Wait for RTC register synchronization */
        RTC_WaitForSynchro();
        /* Wait for the last write operation on the RTC register to complete
*/
        RTC_WaitForLastTask();
        /* Set the RTC counter value to 0 */
        RTC_ConfigCounter(0U);

        /* Wait for the last write operation on the RTC register to complete
*/
        RTC_WaitForLastTask();
        /* Set the RTC alarm value to 5s */
        RTC_ConfigAlarm(ALARM_TIME_INTERVAL);
        /* Wait for the last write operation on the RTC register to complete
*/
        RTC_WaitForLastTask();

    }
}
```

3.3 IO configuration before entering the STOP mode

To achieve the lowest power consumption in the STOP mode, it is recommended to configure all unused I/O ports as analog input mode before entering this mode. The following is the code for GPIO initialization configuration:

```
void GPIO_ALL_Init(void)
{
    GPIO_Config_T GPIO_Configure;

    /* Enable all GPIO clocks */
    RCM_EnableAPB2PeriphClock(RCM_APB2_PERIPH_GPIOA | RCM_APB2_PERIPH_GPIOB |
RCM_APB2_PERIPH_GPIOC | RCM_APB2_PERIPH_GPIOD);

    /* Configure IO for analog input. To achieve the lowest power consumption in
STOP mode, keep the PA0 interrupt wake-up function */
    GPIO_Configure.mode = GPIO_MODE_ANALOG;
    GPIO_Configure.pin = GPIO_PIN_ALL&(~GPIO_PIN_0);
    GPIO_Config(GPIOA, &GPIO_Configure);

    GPIO_Configure.pin = GPIO_PIN_ALL;
    GPIO_Config(GPIOB, &GPIO_Configure);
    GPIO_Config(GPIOC, &GPIO_Configure);
```

```
    GPIO_Config(GPIOD, &GPIO_Configure);

    /* Disable all GPIO clocks */
    RCM_DisableAPB2PeriphClock(RCM_APB2_PERIPH_GPIOA | RCM_APB2_PERIPH_GPIOB |
RCM_APB2_PERIPH_GPIOC | RCM_APB2_PERIPH_GPIOD);
}
```

Note: In this example, PA0 is retained as an (optional) interrupt wake-up pin, so it is not configured as an analog input.

(1) After the configuration is completed, the peripheral clock of the GPIO can be disabled to further reduce power consumption.

3.4 Enter STOP mode

The STOP mode can be entered by calling the library function. The following is the code of the function to enter the STOP mode:

```
void System_Enter_StopMode(void)
{
    /* Enable PWR clock */
    RCM_EnableAPB1PeriphClock(RCM_APB1_PERIPH_PMU);
    /* Clear wake-up flag */
    PMU_ClearStatusFlag(PMU_FLAG_WUE);
    /* Enter STOP mode */
    PMU_EnterSTOPMode(PMU_REGULATOR_LOWPOWER, PMU_STOP_ENTRY_WFI);
}
```

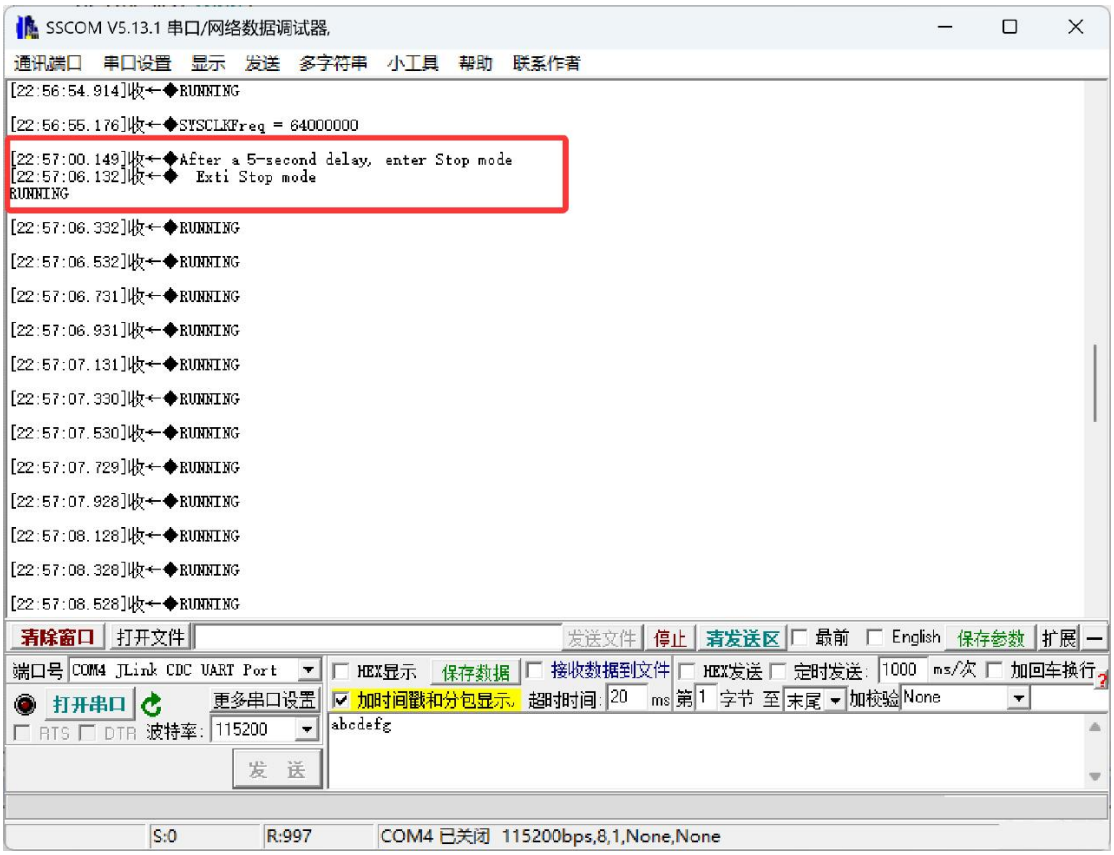
4 Function Verification

In the main function, we first initialize the system and peripherals. After a 5-second delay, we call the relevant function to enter the STOP mode. When the RTC alarm interrupt occurs after 5 seconds, the MCU will be woken up. After being woken up, the program will continue to execute from the next instruction after entering the STOP mode. The following is the verification code for the main function:

```
int main(void)
{
    NVIC_ConfigPriorityGroup(NVIC_PRIORITY_GROUP_2);

    BSP_HSI64_SysclkConfig();
    BSP_Systick_Init(64);
    BSP_LED_Init(GPIOB, GPIO_PIN_8);
    BSP_LED_Init(GPIOB, GPIO_PIN_9);
    BSP_KEY2_Init(BUTTON_MODE_EXTI);
    BSP_USART1_Init(115200);

    printf("SYSCLKFreq = %d\r\n", RCM_ReadSYSCLKFreq());

    // Delay for 5 seconds before entering the STANDBY mode to facilitate erasing
and re-downloading of the program during reset
    delay_ms(1000);
    delay_ms(1000);
    delay_ms(1000);
    delay_ms(1000);
    delay_ms(1000);
    printf("After a 5-second delay, enter Stop mode\r\n");
    GPIO_ALL_Init();
    RTC_Config_Init();
    System_Enter_StopMode();

    // The clock needs to be reconfigured after waking up
    BSP_HSI64_SysclkConfig();
    BSP_Systick_Init(64);
    BSP_LED_Init(GPIOB, GPIO_PIN_8);
    BSP_LED_Init(GPIOB, GPIO_PIN_9);
    BSP_KEY2_Init(BUTTON_MODE_EXTI);
    BSP_USART1_Init(115200);
    printf("Exti Stop mode\r\n");
    while (1)
```

```
    {
        printf("RUNNING\r\n");
        delay_ms(200);
    }
}
```

Through the serial port debugging tool, the following printed information can be observed, which proves that the MCU is successfully woken up after entering the STOP mode for 5 seconds and continues to execute the program. The following are the results printed by the serial port:



Note:

(1) After waking up from the STOP mode, the system clock will be restored to the internal high-speed clock (HSI). The system clock and the peripherals that depend on the clock need to be reconfigured.

(2) In the example code, after waking up, the system clock, Systick, LED, buttons, and serial port are reconfigured, and information is printed in the loop to verify the successful wake-up.

5 Revision History

Table 5 Document Revision History

Date	Version	Revision History
August, 2025	1.0	New

Statement

This manual is formulated and published by Zhuhai Geehy Semiconductor Co., Ltd. (hereinafter referred to as "Geehy"). The contents in this manual are protected by laws and regulations of trademark, copyright and software copyright. Geehy reserves the right to correct and modify this manual at any time. Please read this manual carefully before using the product. Once you use the product, it means that you (hereinafter referred to as the "users") have known and accepted all the contents of this manual. Users shall use the product in accordance with relevant laws and regulations and the requirements of this manual.

1. Ownership of rights

This manual can only be used in combination with chip products and software products of corresponding models provided by Geehy. Without the prior permission of Geehy, no unit or individual may copy, transcribe, modify, edit or disseminate all or part of the contents of this manual for any reason or in any form.

The "Geehy" or "Geehy" words or graphics with "®" or "TM" in this manual are trademarks of Geehy. Other product or service names displayed on Geehy products are the property of their respective owners.

2. No intellectual property license

Geehy owns all rights, ownership and intellectual property rights involved in this manual.

Geehy shall not be deemed to grant the license or right of any intellectual property to users explicitly or implicitly due to the sale and distribution of Geehy products and this manual.

If any third party’s products, services or intellectual property are involved in this manual, Geehy shall not be deemed to authorize users to use the aforesaid third party’s products, services or intellectual property, nor shall it be deemed to provide any form of guarantee for third-party products, services, or intellectual property, including but not limited to any

non-infringement guarantee for third-party intellectual property, unless otherwise agreed in sales order or sales contract of Geehy.

3. Version update

Users can obtain the latest manual of the corresponding products when ordering Geehy products.

If the contents in this manual are inconsistent with Geehy products, the agreement in Geehy sales order or sales contract shall prevail.

4. Information reliability

The relevant data in this manual are obtained from batch test by Geehy Laboratory or cooperative third-party testing organization. However, clerical errors in correction or errors caused by differences in testing environment are unavoidable. Therefore, users should understand that Geehy does not bear any responsibility for such errors that may occur in this manual. The relevant data in this manual are only used to guide users as performance parameter reference and do not constitute Geehy's guarantee for any product performance.

Users shall select appropriate Geehy products according to their own needs, and effectively verify and test the applicability of Geehy products to confirm that Geehy products meet their own needs, corresponding standards, safety or other reliability requirements. If losses are caused to users due to the user's failure to fully verify and test Geehy products, Geehy will not bear any responsibility.

5. Compliance requirements

Users shall abide by all applicable local laws and regulations when using this manual and the matching Geehy products. Users shall understand that the products may be restricted by the export, re-export or other laws of the countries of the product suppliers, Geehy, Geehy distributors and users. Users (on behalf of itself, subsidiaries and affiliated enterprises) shall agree and undertake to abide by all applicable laws and regulations on the export and re-export

of Geehy products and/or technologies and direct products.

6. Disclaimer

This manual is provided by Geehy on an "as is" basis. To the extent permitted by applicable laws, Geehy does not provide any form of express or implied warranty, including without limitation the warranty of product merchantability and applicability of specific purposes.

Geehy products are not designed, authorized, or guaranteed to be suitable for use as critical components in military, life support, pollution control, or hazardous substance management systems, nor are they designed, authorized, or guaranteed to be suitable for applications that may cause injury, death, property, or environmental damage in case of product failure or malfunction.

If the product is not labeled as "Automotive grade", it means it is not suitable for automotive applications. If the user's application of the product is beyond the specifications, application fields, and standards provided by Geehy, Geehy will assume no responsibility.

Users shall ensure that their application of the product complies with relevant standards, and the requirements of functional safety, information security, and environmental standards. Users are fully responsible for their selection and use of Geehy products. Geehy will bear no responsibility for any disputes arising from the subsequent design and use of Geehy products by users.

7. Limitation of liability

In any case, unless required by applicable laws or agreed in writing, Geehy and/or any third party providing this manual and the products on an "as is" basis shall not be liable for damages, including any general or special direct, indirect or collateral damages arising from the use or no use of this manual and the products (including without limitation data loss or inaccuracy, or losses suffered by users or third parties), which cover damage to personal safety, property, or environment, for which Geehy will not be responsible.

8. Scope of application

The information in this manual replaces the information provided in all previous versions of the manual.