TB3213

RTC 入门

简介

作者：Victor Berzan，Microchip Technology Inc.

实时计数器（Real-Time Counter，RTC）对计数器寄存器中的（预分频）时钟周期进行计数，并将计数器寄存器的内容与周期寄存器和比较寄存器进行比较。RTC 可以在比较匹配或溢出时产生中断和事件。它将在计数器值等于比较寄存器值后的第一个计数生成比较中断和/或事件，在计数器值等于周期寄存器值后的第一个计数生成溢出中断和/或事件。溢出时还会将计数器值复位为零。

周期性中断定时器（Periodic Interrupt Timer，PIT）使用与 RTC 功能相同的时钟源，可以每隔 *n* 个时钟周期请求一次中断或触发一次输出事件（对于中断，*n* 可从{4, 8, 16,..32768}范围内选择；对于事件，可从{64, 128, 256,...8192}范围内选择）。

本技术简介介绍了 tinyAVR® 0 系列、tinyAVR 1 系列和 megaAVR® 0 系列单片机上的 RTC 模块如何工作。它涵盖以下用例：

- **RTC 溢出中断：**
  初始化 RTC，允许溢出中断，在每次溢出时翻转 LED。
- **RTC 周期性中断：**
  初始化 RTC PIT，允许周期性中断，在每次周期性中断时翻转 LED。
- **RTC PIT 从休眠模式唤醒：**
  初始化 RTC PIT，允许周期性中断，配置器件休眠模式，将 CPU 置于 `SLEEP` 模式，PIT 中断将唤醒 CPU。

注： 示例代码是基于 ATmega4809 Xplained Pro（ATMEGA4809-XPRO）开发的。

目录

1. 相关器件

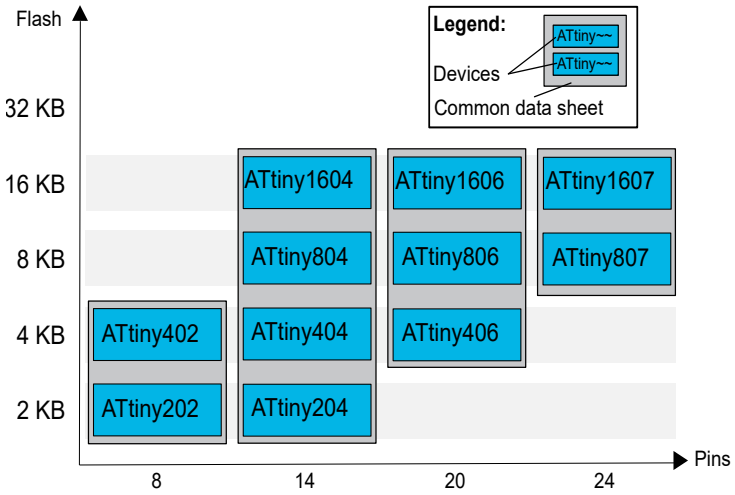
本章列出了文中涉及的相关器件。

1.1 tinyAVR 0 系列

下图所示为 tinyAVR 0 系列器件，注明了不同的引脚数与存储器大小：

- 垂直迁移无需修改代码，因为这些器件的引脚互相兼容，后者可提供相同甚至更多的功能。
- 从右到左迁移会减少引脚数，进而减少可用的功能。

图 1-1. tinyAVR® 0 系列概览



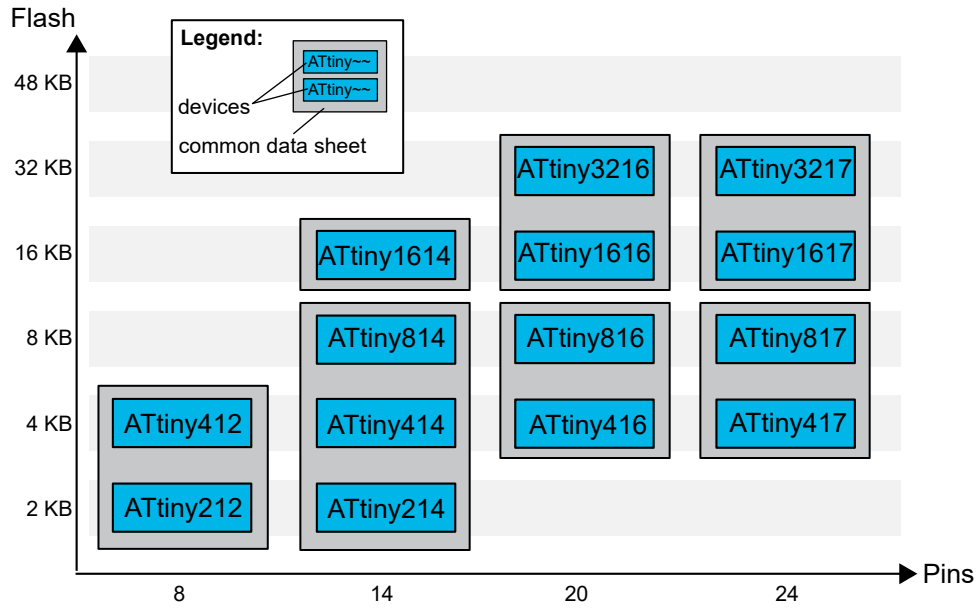
具有不同闪存大小的器件通常也具有不同的 SRAM 和 EEPROM。

1.2 tinyAVR 1 系列

下图所示为 tinyAVR 1 系列器件，注明了不同的引脚数与存储器大小：

- 从下到上迁移无需修改代码，因为这些器件的引脚互相兼容，后者可提供相同甚至更多的功能。而从上到下迁移可能需要修改代码，因为部分外设的可用引脚数较少。
- 从右到左迁移会减少引脚数，进而减少可用的功能。

图 1-2. tinyAVR® 1 系列概览



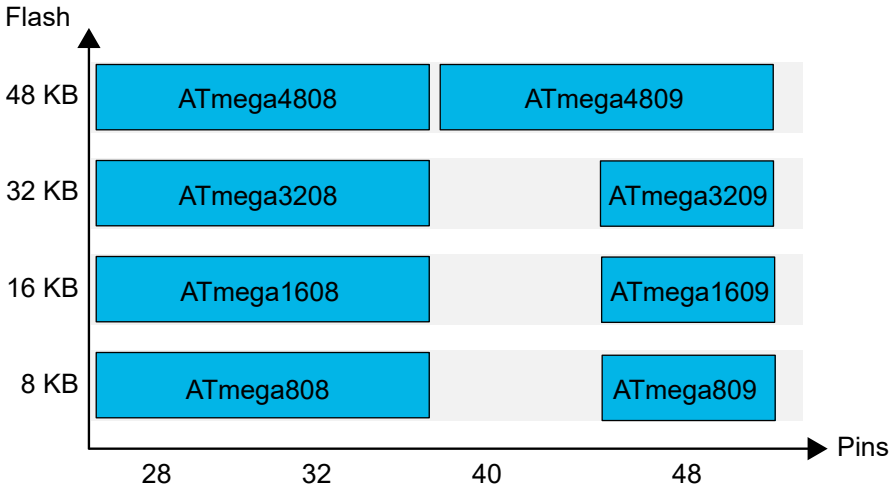
具有不同闪存大小的器件通常也具有不同的 SRAM 和 EEPROM。

1.3 megaAVR 0 系列

下图所示为 megaAVR 0 系列器件，注明了不同的引脚数与存储器大小：

- 垂直迁移无需修改代码，因为这些器件的引脚和功能完全兼容。
- 从右到左迁移会减少引脚数，进而减少可用的功能。

图 1-3. megaAVR® 0 系列概览

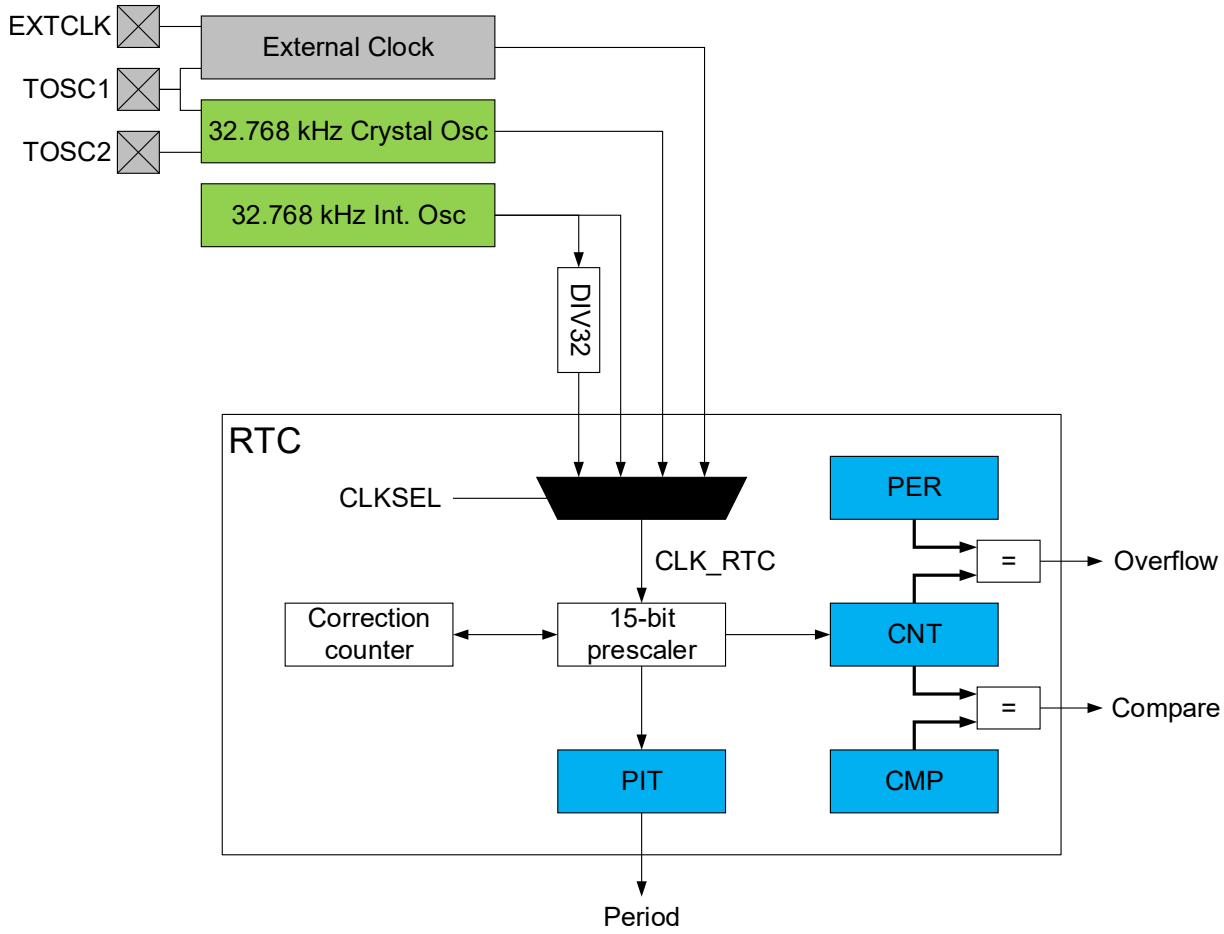


具有不同闪存大小的器件通常也具有不同的 SRAM 和 EEPROM。

2. 概述

RTC 外设提供两种定时功能：实时计数器（RTC）和周期性中断定时器（PIT）。PIT 功能可以独立使能，与 RTC 功能无关。

图 2-1. 框图



PIT 功能和 RTC 功能依靠预分频器内的同一计数器运行。通过写入 RTC.CTRLA 中的 PRESCALER 位域来配置使 CNT 递增的时钟信号的周期。RTC.PITCTRLA 中的 PERIOD 位域从 15 位预分频器计数器中选择要用作 PIT PERIOD 输出的位。

3. RTC 溢出中断

要使 RTC 正常工作，必须先配置 RTC 计数器的源时钟才能使能 RTC 外设和所需操作（中断请求和输出事件）。本示例中将 32.768 kHz 外部振荡器用作源时钟。

要配置振荡器，必须先将 CLKCTRL.XOSC32KCTRLA 寄存器中的 ENABLE 位清零，以禁止振荡器：

图 3-1. CLKCTRL.XOSC32KCTRLA——清零 ENABLE 位

The SEL and CSUT bits cannot be changed as long as the ENABLE bit is set or the XOSC32K Stable bit (XOSC32KS) in CLKCTRL.MCLKSTATUS is high.
To change settings safely: Write a '0' to the ENABLE bit and wait until XOSC32KS is '0' before re-enabling the XOSC32K with new settings.

Bit	7	6	5	4	3	2	1	0
			CSUT[1:0]			SEL	RUNSTDBY	ENABLE
Access			R/W	R/W		R/W	R/W	R/W
Reset			0	0		0	0	0

Bit 0 – ENABLE Enable
When this bit is written to '1', the configuration of the respective input pins is overridden to TOSC1 and TOSC2. Also, the Source Select bit (SEL) and Crystal Start-Up Time (CSUT) become read-only.
This bit is I/O protected to prevent any unintentional enabling of the oscillator.

```
uint8_t temp;
temp = CLKCTRL.XOSC32KCTRLA;
temp &= ~CLKCTRL_ENABLE_bm;
ccp_write_io((void*)&CLKCTRL.XOSC32KCTRLA, temp);
```

随后，用户必须等待相应的状态位变为 0：

图 3-2. CLKCTRL.MCLKSTATUS——读取 XOSC32KS

Bit	7	6	5	4	3	2	1	0
	EXTS	XOSC32KS	OSC32KS	OSC20MS				SOSC
Access	R	R	R	R				R
Reset	0	0	0	0				0

Bit 6 – XOSC32KS XOSC32K Status
The Status bit will only be available if the source is requested as the main clock or by another module. If the oscillator RUNSTDBY bit is set, but the oscillator is unused/not requested, this bit will be 0.

Value	Description
0	XOSC32K is not stable
1	XOSC32K is stable

```
while(CLKCTRL.MCLKSTATUS & CLKCTRL_XOSC32KS_bm)
{
    ;
}
```

必须通过将 CLKCTRL.XOSC32KCTRLA 寄存器中的 SEL 位清零来选择外部振荡器：

图 3-3. CLKCTRL.XOSC32KCTRLA——将 SEL 位清零

Bit	7	6	5	4	3	2	1	0
			CSUT[1:0]			SEL	RUNSTDBY	ENABLE
Access			R/W	R/W		R/W	R/W	R/W
Reset			0	0		0	0	0

Bit 2 – SEL Source Select

This bit selects the type of external source. It is write protected when the oscillator is enabled (ENABLE=1).

Value	Description
0	External crystal
1	External clock on TOSC1 pin

```
temp = CLKCTRL.XOSC32KCTRLA;
temp &= ~CLKCTRL_SEL_bm;
ccp_write_io((void*)&CLKCTRL.XOSC32KCTRLA, temp);
```

必须通过将 CLKCTRL.XOSC32KCTRLA 寄存器中的 ENABLE 位置 1 来使能振荡器。

```
temp = CLKCTRL.XOSC32KCTRLA;
temp |= CLKCTRL_ENABLE_bm;
ccp_write_io((void*)&CLKCTRL.XOSC32KCTRLA, temp);
```

之后，用户必须等待所有寄存器同步：

图 3-4. RTC.STATUS

Bit	7	6	5	4	3	2	1	0
					CMPBUSY	PERBUSY	CNTBUSY	CTRLABUSY
Access					R	R	R	R
Reset					0	0	0	0

Bit 3 – CMPBUSY: Compare Synchronization Busy bit
This bit is indicating whether the RTC is busy synchronizing the Compare register (RTC.CMP) in the RTC clock domain.

Bit 2 – PERBUSY: Period Synchronization Busy bit
This bit is indicating whether the RTC is busy synchronizing the Period register (RTC.PER) in the RTC clock domain.

Bit 1 – CNTBUSY: Counter Synchronization Busy bit
This bit is indicating whether the RTC is busy synchronizing the Count register (RTC.CNT) in the RTC clock domain.

Bit 0 – CTRLABUSY: Control A Synchronization Busy bit
This bit is indicating whether the RTC is busy synchronizing the Control A register (RTC.CTRLA) in the RTC clock domain.

```
while (RTC.STATUS > 0)
{
    ;
}
```

在 RTC.PER 寄存器中设置 RTC 周期：

图 3-5. RTC.PER——设置周期

The RTC.PERL and RTC.PERH register pair represents the 16-bit value, PER. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to **Accessing 16-bit Registers** in the CPU section.

Due to synchronization between the RTC clock and system clock domains, there is a latency of two RTC clock cycles from updating the register until this has an effect. The application software needs to check that the PERBUSY flag in RTC.STATUS is cleared before writing to this register.

Bit	15	14	13	12	11	10	9	8
	PER[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1

Bit	7	6	5	4	3	2	1	0
	PER[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1

Bits 15:8 – PER[15:8] Period High Byte
These bits hold the MSB of the 16-bit Period register.

Bits 7:0 – PER[7:0] Period Low Byte
These bits hold the LSB of the 16-bit Period register.

在 RTC.CLKSEL 寄存器中选择 32.768 kHz 外部晶振时钟：

图 3-6. RTC.CLKSEL——时钟选择

Bit	7	6	5	4	3	2	1	0
							CLKSEL[1:0]	
Access							R/W	R/W
Reset							0	0

Bits 1:0 – CLKSEL[1:0] Clock Select bits
Writing these bits select the source for the RTC clock (CLK_RTC).
When configuring the RTC to use either XOSC32K or the external clock on TOSC1, XOSC32K needs to be enabled, and the Source Select (SEL) bit and Run Standby (RUNSTDBY) bit in the XOSC32K Control A register of the Clock Controller (CLKCTRL.XOSC32KCTRLA) must be configured accordingly.

Value	Name	Description
0x0	INT32K	32.768 kHz from OSCULP32K
0x1	INT1K	1.024 kHz from OSCULP32K
0x2	TOSC32K	32.768 kHz from XOSC32K or external clock from TOSC1
0x3	EXTCLK	External clock from the EXTCLK pin

```
RTC.CLKSEL = RTC_CLKSEL_TOSC32K_gc;
```

要使 RTC 也能在调试模式下运行，应将 RTC.DBGCTRL 寄存器中的 DBGRUN 位置 1：

图 3-7. RTC.CLKSEL——将 DBGRUN 位置 1

Bit	7	6	5	4	3	2	1	0
								DBGRUN
Access								R/W
Reset								0

Bit 0 – DBGRUN: Debug Run bit

Value	Description
0	The peripheral is halted in Break Debug mode and ignores events
1	The peripheral will continue to run in Break Debug mode when the CPU is halted

```
RTC.DBGCTRL |= RTC_DBGRUN_bm;
```

在 RTC.CTRLA 寄存器中设置 RTC 预分频比。要使 RTC 也能在待机模式下运行，应将 RTC.CTRLA 寄存器中的 RUNSTDBY 位置 1。要使能 RTC，应将 RTC.CTRLA 寄存器中的 RTCEN 位置 1。

图 3-8. RTC.CTRLA——设置预分频比、RUNSTDBY 位和 RTCEN 位

Bit	7	6	5	4	3	2	1	0
	RUNSTDBY	PRESCALER[3:0]				CORREN		RTCEN
Access	R/W	R/W	R/W	R/W	R/W	R/W		R/W
Reset	0	0	0	0	0	0		0

Bit 7 – RUNSTDBY Run in Standby

Value	Description
0	RTC disabled in Standby sleep mode
1	RTC enabled in Standby sleep mode

Bits 6:3 – PRESCALER[3:0] Prescaler bits
These bits define the prescaling of the CLK_RTC clock signal. Due to synchronization between the RTC clock and system clock domains, there is a latency of two RTC clock cycles from updating the register until this has an effect. The application software needs to check that the CTRLABUSY flag in RTC.STATUS is cleared before writing to this register.

Value	Name	Description
0x0	DIV1	RTC clock/1 (no prescaling)
0x1	DIV2	RTC clock/2
0x2	DIV4	RTC clock/4
0x3	DIV8	RTC clock/8
0x4	DIV16	RTC clock/16
0x5	DIV32	RTC clock/32
0x6	DIV64	RTC clock/64
0x7	DIV128	RTC clock/128
0x8	DIV256	RTC clock/256
0x9	DIV512	RTC clock/512
0xA	DIV1024	RTC clock/1024
0xB	DIV2048	RTC clock/2048
0xC	DIV4096	RTC clock/4096
0xD	DIV8192	RTC clock/8192
0xE	DIV16384	RTC clock/16384
0xF	DIV32768	RTC clock/32768

Bit 0 – RTCEN RTC Peripheral Enable

Value	Description
0	RTC peripheral disabled
1	RTC peripheral enabled

```
RTC.CTRLA = RTC_PRESCALER_DIV32_gc | RTC_RTCEN_bm | RTC_RUNSTDBY_bm;
```

可以通过将 RTC.INTCTRL 寄存器中的 OVF 位置 1 来允许溢出中断：

图 3-9. RTC.INTCTRL——将 OVF 位置 1

Bit	7	6	5	4	3	2	1	0
							CMP	OVF
Access							R/W	R/W
Reset							0	0

Bit 0 – OVF Overflow Interrupt Enable
Enable interrupt-on-counter overflow (i.e., when the Counter value (CNT) matches the Period value (PER) and wraps around to zero).

```
RTC.INTCTRL |= RTC_OVF_bm;
```

要产生中断，必须允许全局中断：

```
sei();
```

在以下示例中，RTC 溢出的中断服务程序（Interrupt Service Routine，ISR）将翻转 LED：

```
ISR(RTC_CNT_vect)
{
    RTC.INTFLAGS = RTC_OVF_bm;
    LED0_toggle();
}
```

注： 要将 RTC.INTFLAGS 中的 OVF 位清零，必须在 ISR 函数内对该位写 1。

View Code Example on GitHub
Click to browse repository

提示： 有关完整的示例，请参见附录部分。

4. RTC 周期性中断

此特定示例的源时钟配置与 RTC 溢出中断示例相同。可以通过将 RTC.PITINTCTRL 寄存器中的 PI 位置 1 来允许周期性中断。

图 4-1. RTC.PITINTCTRL——将 PI 位置 1

Bit	7	6	5	4	3	2	1	0
								PI
Access								R/W
Reset								0

Bit 0 – PI: Periodic Interrupt bit

Value	Description
0	The periodic interrupt is disabled
1	The periodic interrupt is enabled

```
RTC.PITINTCTRL = RTC_PI_bm;
```

在 RTC.PITCTRLA 寄存器中设置 PIT 周期。可以通过将 RTC.PITCTRLA 寄存器中的 PITEN 位置 1 来使能 PIT。

图 4-2. RTC.PITCTRLA——将 PITEN 位置 1

Bit	7	6	5	4	3	2	1	0
		PERIOD[3:0]						PITEN
Access		R/W	R/W	R/W	R/W			R/W
Reset		0	0	0	0			0

Bits 6:3 – PERIOD[3:0]: Period bits
Writing this bit field selects the number of RTC clock cycles between each interrupt.

Value	Name	Description
0x0	OFF	No interrupt
0x1	CYC4	4 cycles
0x2	CYC8	8 cycles
0x3	CYC16	16 cycles
0x4	CYC32	32 cycles
0x5	CYC64	64 cycles
0x6	CYC128	128 cycles
0x7	CYC256	256 cycles
0x8	CYC512	512 cycles
0x9	CYC1024	1024 cycles
0xA	CYC2048	2048 cycles
0xB	CYC4096	4096 cycles
0xC	CYC8192	8192 cycles
0xD	CYC16384	16384 cycles
0xE	CYC32768	32768 cycles
0xF	-	Reserved

Bit 0 – PITEN: Periodic Interrupt Timer Enable bit
Writing a '1' to this bit enables the Periodic Interrupt Timer.

```
RTC.PITCTRLA = RTC_PERIOD_CYC32768_gc | RTC_PITEN_bm;
```

要产生中断，必须允许全局中断：

```
sei();
```

在以下示例中，RTC PIT 的中断服务程序（ISR）将翻转 LED：

```
ISR(RTC_PIT_vect)
{
    RTC.PITINTFLAGS = RTC_PI_bm;
    LED0_toggle();
}
```

注： 要将 RTC.INTFLAGS 中的 OVF 位清零，必须在 ISR 函数内对该位写 1。

提示： 有关完整的示例，请参见附录部分。

5. RTC PIT 从休眠模式唤醒

PIT 中断可将 CPU 从休眠模式唤醒。

在 SLPCTRL.CTRLA 寄存器中配置休眠模式。可以通过将 SLPCTRL.CTRLA 寄存器中的 SEN 位置 1 来使能休眠功能。

图 5-1. SLPCTRL.CTRLA——设置休眠模式和 SEN 位

Bit	7	6	5	4	3	2	1	0
						SMODE[1:0]		SEN
Access	R	R	R	R	R	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 2:1 – SMODE[1:0] Sleep Mode
Writing these bits selects the sleep mode entered when the Sleep Enable (SEN) bit is written to '1', and the `SLEEP` instruction is executed.

Value	Name	Description
0x0	IDLE	Idle sleep mode enabled
0x1	STANDBY	Standby sleep mode enabled
0x2	PDOWN	Power-Down sleep mode enabled
other	-	Reserved

Bit 0 – SEN Sleep Enable
This bit must be written to '1' before the `SLEEP` instruction is executed to make the MCU enter the selected sleep mode.

```
SLPCTRL.CTRLA |= SLPCTRL_SMODE_PDOWN_gc;
SLPCTRL.CTRLA |= SLPCTRL_SEN_bm;
```

可以通过调用以下函数使 CPU 进入休眠模式：

```
sleep_cpu();
```

PIT 中断会将 CPU 从休眠模式唤醒。要产生中断，必须允许全局中断：

```
sei();
```

在以下示例中，RTC PIT 的中断服务程序（ISR）将翻转 LED：

```
ISR(RTC_PIT_vect)
{
    RTC.PITINTFLAGS = RTC_PI_bm;
    LED0_toggle();
}
```

注： 要将 RTC.INTFLAGS 中的 OVF 位清零，必须在 ISR 函数内对该位写 `1`。

View Code Example on GitHub
Click to browse repository

提示： 有关完整的示例，请参见附录部分。

6. 参考资料

有关 RTC 和 PIT 工作模式的更多信息，请访问以下链接：

1. ATmega4809 产品页面： www.microchip.com/wwwproducts/en/ATMEGA4809
2. megaAVR 0-Family Data Sheet
3. ATmega809/1609/3209/4809——48 引脚 megaAVR 0 系列
4. ATmega4809 Xplained Pro 网页：https://www.microchip.com/developmenttools/ProductDetails/atmega4809-xpro

7. 附录

例 7-1. RTC 溢出中断代码示例

```
/* RTC 周期 */
#define RTC_EXAMPLE_PERIOD            (511)

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/cpufunc.h>

void RTC_init(void);
void LED0_init(void);
inline void LED0_toggle(void);

void RTC_init(void)
{
    uint8_t temp;

    /* 初始化 32.768 kHz 振荡器： */
    /* 禁止振荡器：*/
    temp = CLKCTRL.XOSC32KCTRLA;
    temp &= ~CLKCTRL_ENABLE_bm;
    /* 写入受保护的寄存器 */
    ccp_write_io((void*)&CLKCTRL.XOSC32KCTRLA, temp);

    while(CLKCTRL.MCLKSTATUS & CLKCTRL_XOSC32KS_bm)
    {
        ; /* 等待 XOSC32KS 变为 0 */
    }

    /* SEL = 0（使用外部晶振）： */
    temp = CLKCTRL.XOSC32KCTRLA;
    temp &= ~CLKCTRL_SEL_bm;
    /* 写入受保护的寄存器 */
    ccp_write_io((void*)&CLKCTRL.XOSC32KCTRLA, temp);

    /* 使能振荡器：*/
    temp = CLKCTRL.XOSC32KCTRLA;
    temp |= CLKCTRL_ENABLE_bm;
    /* 写入受保护的寄存器 */
    ccp_write_io((void*)&CLKCTRL.XOSC32KCTRLA, temp);

    /* 初始化 RTC：*/
    while (RTC.STATUS > 0)
    {
        ; /* 等待所有寄存器同步 */
    }

    /* 设置周期 */
    RTC.PER = RTC_EXAMPLE_PERIOD;

    /* 32.768 kHz 外部晶振（XOSC32K）*/
    RTC.CLKSEL = RTC_CLKSEL_TOSC32K_gc;

    /* 在调试模式下运行：已使能 */
    RTC.DBGCTRL |= RTC_DBGRUN_bm;

    RTC.CTRLA = RTC_PRESCALER_DIV32_gc  /* 32 */
    | RTC_RTCEN_bm             /* 使能：已使能 */
    | RTC_RUNSTDBY_bm;         /* 在待机模式下运行：已使能 */

    /* 允许溢出中断 */
    RTC.INTCTRL |= RTC_OVF_bm;
}

void LED0_init(void)
{
    /* 设为高电平（关闭）*/
    PORTB.OUT |= PIN5_bm;
```

```
    /* 设置输出 */
    PORTB.DIR |= PIN5_bm;
}

inline void LED0_toggle(void)
{
    PORTB.OUTTGL |= PIN5_bm;
}

ISR(RTC_CNT_vect)
{
    /* 通过写入 1 将标志清零: */
    RTC.INTFLAGS = RTC_OVF_bm;

    LED0_toggle();
}

int main(void)
{
    LED0_init();
    RTC_init();

    /* 允许全局中断 */
    sei();

    while (1)
    {
    }
}
```

例 7-2. RTC 周期性中断代码示例

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/cpufunc.h>

void RTC_init(void);
void LED0_init(void);
inline void LED0_toggle(void);

void RTC_init(void)
{
    uint8_t temp;

    /* 初始化 32.768 kHz 振荡器: */
    /* 禁止振荡器: */
    temp = CLKCTRL.XOSC32KCTRLA;
    temp &= ~CLKCTRL_ENABLE_bm;
    /* 写入受保护的寄存器 */
    ccp_write_io((void*)&CLKCTRL.XOSC32KCTRLA, temp);

    while(CLKCTRL.MCLKSTATUS & CLKCTRL_XOSC32KS_bm)
    {
        ; /* 等待 XOSC32KS 变为 0 */
    }

    /* SEL = 0（使用外部晶振）: */
    temp = CLKCTRL.XOSC32KCTRLA;
    temp &= ~CLKCTRL_SEL_bm;
    /* 写入受保护的寄存器 */
    ccp_write_io((void*)&CLKCTRL.XOSC32KCTRLA, temp);

    /* 使能振荡器: */
    temp = CLKCTRL.XOSC32KCTRLA;
    temp |= CLKCTRL_ENABLE_bm;
    /* 写入受保护的寄存器 */
    ccp_write_io((void*)&CLKCTRL.XOSC32KCTRLA, temp);

    /* 初始化 RTC: */
    while (RTC.STATUS > 0)
    {
```

```
        ; /* 等待所有寄存器同步 */
    }

    /* 32.768 kHz 外部晶振（XOSC32K）*/
    RTC.CLKSEL = RTC_CLKSEL_TOSC32K_gc;

    /* 在调试模式下运行：已使能 */
    RTC.DBGCTRL = RTC_DBGRUN_bm;

    RTC.PITINTCTRL = RTC_PI_bm; /* 周期性中断：已使能 */

    RTC.PITCTRLA = RTC_PERIOD_CYC32768_gc /* RTC 时钟周期 32768 */
                 | RTC_PITEN_bm; /* 使能：已使能 */
}

void LED0_init(void)
{
    /* 设为高电平（关闭）*/
    PORTB.OUT |= PIN5_bm;
    /* 设置输出*/
    PORTB.DIR |= PIN5_bm;
}

inline  void LED0_toggle(void)
{
    PORTB.OUTTGL |= PIN5_bm;
}

ISR(RTC_PIT_vect)
{
    /* 通过写入 1 将标志清零：*/
    RTC.PITINTFLAGS = RTC_PI_bm;

    LED0_toggle();
}

int main(void)
{
    LED0_init();
    RTC_init();

    /* 允许全局中断 */
    sei();

    while (1)
    {
    }
}
```

例 7-3. RTC PIT 从休眠模式唤醒的代码示例

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <avr/cpufunc.h>

void RTC_init(void);
void LED0_init(void);
inline void LED0_toggle(void);
void SLPCTRL_init(void);

void RTC_init(void)
{
    uint8_t temp;

    /* 初始化 32.768 kHz 振荡器：*/
    /* 禁止振荡器：*/
    temp = CLKCTRL.XOSC32KCTRLA;
    temp &= ~CLKCTRL_ENABLE_bm;
    /* 写入受保护的寄存器 */
    ccp_write_io((void*)&CLKCTRL.XOSC32KCTRLA, temp);
```

```
    while(CLKCTRL.MCLKSTATUS & CLKCTRL_XOSC32KS_bm)
    {
        ; /* 等待 XOSC32KS 变为 0 */
    }

    /* SEL = 0（使用外部晶振）: */
    temp = CLKCTRL.XOSC32KCTRLA;
    temp &= ~CLKCTRL_SEL_bm;
    /* 写入受保护的寄存器 */
    ccp_write_io((void*)&CLKCTRL.XOSC32KCTRLA, temp);

    /* 使能振荡器: */
    temp = CLKCTRL.XOSC32KCTRLA;
    temp |= CLKCTRL_ENABLE_bm;
    /* 写入受保护的寄存器 */
    ccp_write_io((void*)&CLKCTRL.XOSC32KCTRLA, temp);

    /* 初始化 RTC: */
    while (RTC.STATUS > 0)
    {
        ; /* 等待所有寄存器同步 */
    }

    /* 32.768 kHz 外部晶振（XOSC32K）*/
    RTC.CLKSEL = RTC_CLKSEL_TOSC32K_gc;

    /* 在调试模式下运行：已使能 */
    RTC.DBGCTRL = RTC_DBGRUN_bm;

    RTC.PITINTCTRL = RTC_PI_bm; /* 周期性中断：已使能 */

    RTC.PITCTRLA = RTC_PERIOD_CYC32768_gc /* RTC 时钟周期 32768 */
                 | RTC_PITEN_bm; /* 使能：已使能 */
}

void LED0_init(void)
{
    /* 设为高电平（关闭）*/
    PORTB.OUT |= PIN5_bm;
    /* 设置输出*/
    PORTB.DIR |= PIN5_bm;
}

inline void LED0_toggle(void)
{
    PORTB.OUTTGL |= PIN5_bm;
}

ISR(RTC_PIT_vect)
{
    /* 通过写入 1 将标志清零: */
    RTC.PITINTFLAGS = RTC_PI_bm;

    LED0_toggle();
}

void SLPCTRL_init(void)
{
    SLPCTRL.CTRLA |= SLPCTRL_SMODE_PDOWN_gc;
    SLPCTRL.CTRLA |= SLPCTRL_SEN_bm;
}

int main(void)
{
    LED0_init();
    RTC_init();
    SLPCTRL_init();

    /* 允许全局中断 */
    sei();

    while (1)
    {
```

```
        /* 使 CPU 进入休眠模式 */
        sleep_cpu();

        /* PIT 中断将唤醒 CPU */
    }
}
```

8. 版本历史

文档版本	日期	备注
B	2019 年 12 月	在代码中将 CCP 寄存器写操作替换为 `ccp_write_io()`函数。
A	2019 年 05 月	文档初始版本。

Microchip 网站

Microchip 网站（www.microchip.com/）为客户提供在线支持。客户可通过该网站方便地获取文件和信息。我们的网站提供以下内容：

- **产品支持**——数据手册和勘误表、应用笔记和示例程序、设计资源、用户指南以及硬件支持文档、最新的软件版本以及归档软件
- **一般技术支持**——常见问题解答（FAQ）、技术支持请求、在线讨论组以及 Microchip 设计伙伴计划成员名单
- **Microchip 业务**——产品选型和订购指南、最新 Microchip 新闻稿、研讨会和活动安排表、Microchip 销售办事处、代理商以及工厂代表列表

产品变更通知服务

Microchip 的产品变更通知服务有助于客户了解 Microchip 产品的最新信息。注册客户可在他们感兴趣的某个产品系列或开发工具发生变更、更新、发布新版本或勘误表时，收到电子邮件通知。

欲注册，请访问 www.microchip.com/pcn，然后按照注册说明进行操作。

客户支持

Microchip 产品的用户可通过以下渠道获得帮助：

- 代理商或代表
- 当地销售办事处
- 应用工程师（ESE）
- 技术支持

客户应联系其代理商、代表或 ESE 寻求支持。当地销售办事处也可为客户提供帮助。本文档后附有销售办事处的联系方式。

也可通过 www.microchip.com/support 获得网上技术支持。

Microchip 器件代码保护功能

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术规范。
- Microchip 确信：在正常使用的情况下，Microchip 系列产品非常安全。
- 目前，仍存在着用恶意、甚至是非法的方法来试图破坏代码保护功能的行为。我们确信，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这种试图破坏代码保护功能的行为极可能侵犯 Microchip 的知识产权。
- Microchip 愿与那些注重代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了《数字器件千年版权法案（Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

法律声明

提供本文档的中文版本仅为了便于理解。请勿忽视文档中包含的英文部分，因为其中提供了有关 Microchip 产品性能和使用情况的有用信息。Microchip Technology Inc.及其分公司和相关公司、各级主管与员工及事务代理机构对译文中可能存在的任何差错不承担任何责任。建议参考 Microchip Technology Inc.的英文原版文档。

商标

ISBN： 978-1-5224-7334-3

质量管理体系

有关 Microchip 的质量管理体系的信息，请访问 www.microchip.com/quality。

全球销售及服务网点

美洲	亚太地区	亚太地区	欧洲
公司总部 2355 West Chandler Blvd. Chandler, AZ 85224-6199 电话：480-792-7200 传真：480-792-7277 技术支持： www.microchip.com/support 网址： www.microchip.com **亚特兰大** 德卢斯，佐治亚州 电话：678-957-9614 传真：678-957-1455 **奥斯汀，德克萨斯州** 电话：512-257-3370 **波士顿** 韦斯特伯鲁，马萨诸塞州 电话：774-760-0087 传真：774-760-0088 **芝加哥** 艾塔斯卡，伊利诺伊州 电话：630-285-0071 传真：630-285-0075 **达拉斯** 阿迪森，德克萨斯州 电话：972-818-7423 传真：972-818-2924 **底特律** 诺维，密歇根州 电话：248-848-4000 **休斯顿，德克萨斯州** 电话：281-894-5983 **印第安纳波利斯** 诺布尔斯维尔，印第安纳州 电话：317-773-8323 传真：317-773-5453 电话：317-536-2380 **洛杉矶** 米慎维荷，加利福尼亚州 电话：949-462-9523 传真：949-462-9608 电话：951-273-7800 **罗利，北卡罗来纳州** 电话：919-844-7510 **纽约，纽约州** 电话：631-435-6000 **圣何塞，加利福尼亚州** 电话：408-735-9110 电话：408-436-4270 **加拿大 - 多伦多** 电话：905-695-1980 传真：905-695-2078	**澳大利亚 - 悉尼** 电话：61-2-9868-6733 **中国 - 北京** 电话：86-10-8569-7000 **中国 - 成都** 电话：86-28-8665-5511 **中国 - 重庆** 电话：86-23-8980-9588 **中国 - 东莞** 电话：86-769-8702-9880 **中国 - 广州** 电话：86-20-8755-8029 **中国 - 杭州** 电话：86-571-8792-8115 **中国 - 香港特别行政区** 电话：852-2943-5100 **中国 - 南京** 电话：86-25-8473-2460 **中国 - 青岛** 电话：86-532-8502-7355 **中国 - 上海** 电话：86-21-3326-8000 **中国 - 沈阳** 电话：86-24-2334-2829 **中国 - 深圳** 电话：86-755-8864-2200 **中国 - 苏州** 电话：86-186-6233-1526 **中国 - 武汉** 电话：86-27-5980-5300 **中国 - 西安** 电话：86-29-8833-7252 **中国 - 厦门** 电话：86-592-2388138 **中国 - 珠海** 电话：86-756-3210040	**印度 - 班加罗尔** 电话：91-80-3090-4444 **印度 - 新德里** 电话：91-11-4160-8631 **印度 - 浦那** 电话：91-20-4121-0141 **日本 - 大阪** 电话：81-6-6152-7160 **日本 - 东京** 电话：81-3-6880-3770 **韩国 - 大邱** 电话：82-53-744-4301 **韩国 - 首尔** 电话：82-2-554-7200 **马来西亚 - 吉隆坡** 电话：60-3-7651-7906 **马来西亚 - 槟榔屿** 电话：60-4-227-8870 **菲律宾 - 马尼拉** 电话：63-2-634-9065 **新加坡** 电话：65-6334-8870 **台湾地区 - 新竹** 电话：886-3-577-8366 **台湾地区 - 高雄** 电话：886-7-213-7830 **台湾地区 - 台北** 电话：886-2-2508-8600 **泰国 - 曼谷** 电话：66-2-694-1351 **越南 - 胡志明市** 电话：84-28-5448-2100	**奥地利 - 韦尔斯** 电话：43-7242-2244-39 传真：43-7242-2244-393 **丹麦 - 哥本哈根** 电话：45-4485-5910 传真：45-4485-2829 **芬兰 - 埃斯波** 电话：358-9-4520-820 **法国 - 巴黎** 电话：33-1-69-53-63-20 传真：33-1-69-30-90-79 **德国 - 加兴** 电话：49-8931-9700 **德国 - 哈恩** 电话：49-2129-3766400 **德国 - 海尔布隆** 电话：49-7131-72400 **德国 - 卡尔斯鲁厄** 电话：49-721-625370 **德国 - 慕尼黑** 电话：49-89-627-144-0 传真：49-89-627-144-44 **德国 - 罗森海姆** 电话：49-8031-354-560 **以色列 - 若那那市** 电话：972-9-744-7705 **意大利 - 米兰** 电话：39-0331-742611 传真：39-0331-466781 **意大利 - 帕多瓦** 电话：39-049-7625286 **荷兰 - 德卢内市** 电话：31-416-690399 传真：31-416-690340 **挪威 - 特隆赫姆** 电话：47-72884388 **波兰 - 华沙** 电话：48-22-3325737 **罗马尼亚 - 布加勒斯特** 电话：40-21-407-87-50 **西班牙 - 马德里** 电话：34-91-708-08-90 传真：34-91-708-08-91 **瑞典 - 哥德堡** 电话：46-31-704-60-40 **瑞典 - 斯德哥尔摩** 电话：46-8-5090-4654 **英国 - 沃金厄姆** 电话：44-118-921-5800 传真：44-118-921-5820