Getting Started with RTC

Introduction

Author: Victor Berzan, Microchip Technology Inc.

The Real-Time Counter (RTC) counts (prescaled) clock cycles in a Counter register, and compares the content of the Counter register to a Period register and a Compare register. The RTC can generate both interrupts and events on compare match or overflow. It will generate a compare interrupt and/or event at the first count after the counter equals the Compare register value, and an overflow interrupt and/or event at the first count after the counter value equals the Period register value. The overflow will also reset the counter value to zero.

Using the same clock source as the RTC function, the Periodic Interrupt Timer (PIT) can request an interrupt or trigger an output event on every n^{th} clock period ('n' can be selected from {4, 8, 16,.. 32768} for interrupts, and from {64, 128, 256,... 8192} for events).

This technical brief describes how the RTC module works on tinyAVR® 0-series, tinyAVR® 1-series and megaAVR® 0-series microcontrollers. It covers the following use cases:

- **RTC Overflow Interrupt:**
  Initialize the RTC, enable overflow interrupt, toggle an LED on each overflow.
- **RTC Periodic Interrupt:**
  Initialize the RTC PIT, enable periodic interrupt, toggle an LED on each periodic interrupt.
- **RTC PIT Wake from Sleep:**
  Initialize the RTC PIT, enable periodic interrupt, configure device Sleep mode, put CPU in `SLEEP`, the PIT interrupt will wake the CPU.

Note: The code examples were developed on ATmega4809 Xplained Pro (ATMEGA4809-XPRO).

Table of Contents

1. Relevant Devices

This chapter lists the relevant devices for this document.

1.1 tinyAVR® 0-series

The figure below shows the tinyAVR 0-series, laying out pin count variants and memory sizes:

- Vertical migration is possible without code modification, as these devices are fully pin- and feature-compatible.
- Horizontal migration to the left reduces the pin count and, therefore, the available features.
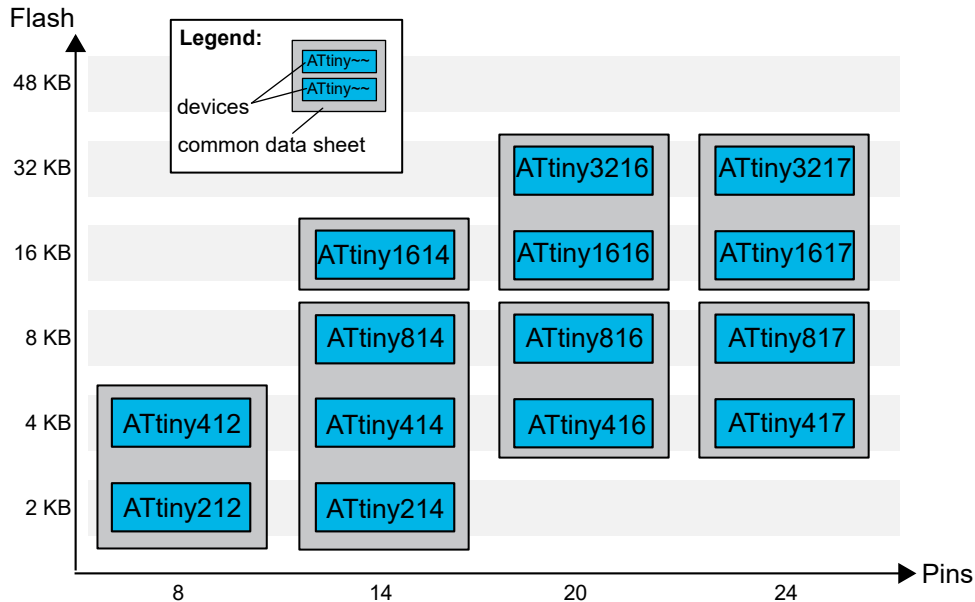
Figure 1-1. tinyAVR® 0-series Overview
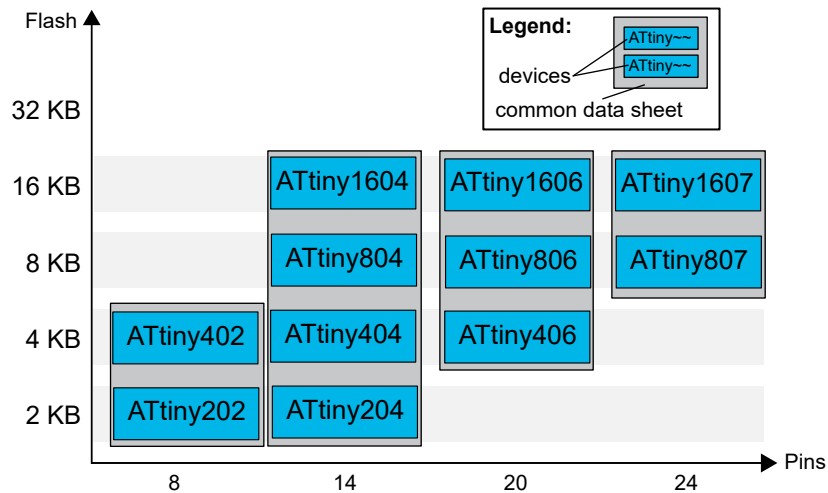
Devices with different Flash memory size typically also have different SRAM and EEPROM.

1.2 tinyAVR® 1-series

The following figure shows the tinyAVR 1-series devices, laying out pin count variants and memory sizes:

- Vertical migration upwards is possible without code modification, as these devices are pin-compatible and provide the same or more features. Downward migration may require code modification due to fewer available instances of some peripherals.
- Horizontal migration to the left reduces the pin count and, therefore, the available features.

Figure 1-2. tinyAVR® 1-series Overview

Devices with different Flash memory size typically also have different SRAM and EEPROM.

1.3 megaAVR® 0-series

The figure below shows the megaAVR 0-series devices, laying out pin count variants and memory sizes:

- Vertical migration is possible without code modification, as these devices are fully pin and feature compatible.
- Horizontal migration to the left reduces the pin count and, therefore, the available features.
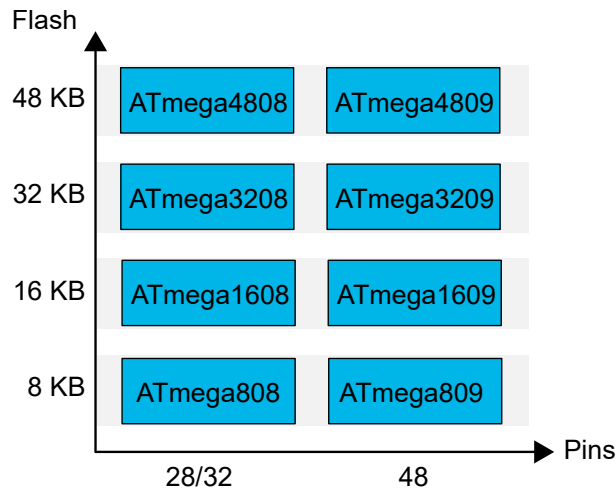
Figure 1-3. megaAVR® 0-series Overview

Devices with different Flash memory size typically also have different SRAM and EEPROM.

2. Overview

The RTC peripheral offers two timing functions: a Real-Time Counter (RTC) and a Periodic Interrupt Timer (PIT). The PIT functionality can be enabled independently of the RTC functionality.

Figure 2-1. Block Diagram
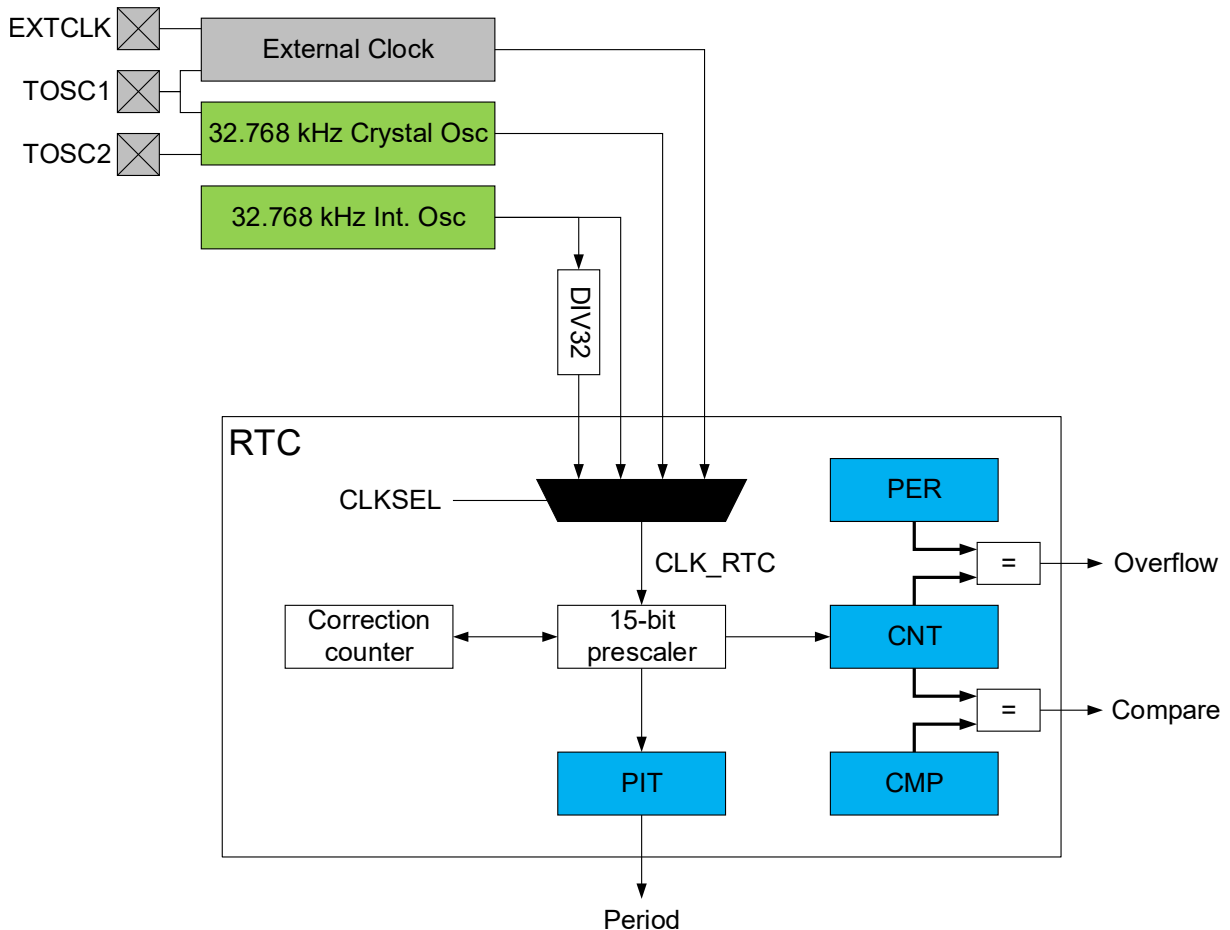
The PIT function and the RTC function are running off the same counter inside the prescaler. The period of the clock signal that increments the CNT is configured by writing the PRESCALER bit field in RTC.CTRLA. The PERIOD bit field in RTC.PITCTRLA selects the bit from the 15-bit prescaler counter to be used as PIT PERIOD output.

3.   RTC Overflow Interrupt

To operate the RTC, the source clock for the RTC counter must be configured before enabling the RTC peripheral, and the desired actions (interrupt requests, output Events). In this example, the 32.768 kHz external oscillator is used as the source clock.

To configure the oscillator, first it must be disabled by clearing the ENABLE bit in the CLKCTRL.XOSC32KCTRLA register:

Figure 3-1. CLKCTRL.XOSC32KCTRLA – clear ENABLE bit

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 in a safe way: 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 bit
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 unintentional enabling of the oscillator.

```
uint8_t temp;
temp = CLKCTRL.XOSC32KCTRLA;
temp &= ~CLKCTRL_ENABLE_bm;
CPU_CCP = CCP_IOREG_gc;
CLKCTRL.XOSC32KCTRLA = temp;
```

The user must then wait for the corresponding Status bit to become '0':

Figure 3-2. CLKCTRL.MCLKSTATUS – read 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 bit
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)
{
    ;
}
```

The external oscillator must be selected by clearing the SEL bit in the CLKCTRL.XOSC32KCTRLA register:

Figure 3-3. CLKCTRL.XOSC32KCTRLA – clear SEL bit

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 bit
This bit selects the external sources type. 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;
CPU_CCP = CCP_IOREG_gc;
CLKCTRL.XOSC32KCTRLA = temp;
```

Then, the oscillator must be enabled by setting the ENABLE bit in the CLKCTRL.XOSC32KCTRLA register:

```
temp = CLKCTRL.XOSC32KCTRLA;
temp |= CLKCTRL_ENABLE_bm;
CPU_CCP = CCP_IOREG_gc;
CLKCTRL.XOSC32KCTRLA = temp;
```

Afterwards, the user must wait for all registers to be synchronized:

Figure 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)
{
    ;
}
```

The RTC period is set in the RTC.PER register:

Figure 3-5. RTC.PER – set Period

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 chapter from the *"mega AVR® 0-Series Manual (DS40002015)"*.

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. 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 bits
These bits hold the MSB of the 16-bit Period register.

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

The 32.768 kHz External Crystal Oscillator clock is selected in the RTC.CLKSEL register:

Figure 3-6. RTC.CLKSEL – Clock Selection

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 selects 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 bit (SEL) and Run Standby bit (RUNSTDBY) 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 EXTCLK pin

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

To enable the RTC to also run in Debug mode, the DBGRUN bit is set in the RTC.DBGCTRL register:

Figure 3-7. RTC.CLKSEL – set DBGRUN bit

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;
```

The RTC prescaler is set in the RTC.CTRLA register. To enable the RTC to also run in Standby mode, the RUNSTDBY bit is set in the RTC.CTRLA register. To enable the RTC, the RTCEN bit is set in the RTC.CTRLA register.

Figure 3-8. RTC.CTRLA – set Prescaler, RUNSTDBY bit, RTCEN bit

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 bit

Value	DescriptionDescriptionDescription
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 clockc cycles from updating the register until this has an effect. 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

Bit 0 – RTCEN: RCT Enable bit

Value	Description
0	RTC disabled
1	RTC enabled

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

The overflow interrupt is enabled by setting the OVF bit in the RTC.INTCTRL register:

Figure 3-9. RTC.INTCTRL – set OVF bit

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 bit
Enable interrupt-on-counter overflow (i.e., when the Counter value (CNT) matched the Period value (PER) and wraps around to zero).

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

For the interrupt to occur, the global interrupts must be also enabled:

```
sei();
```

The Interrupt Service Routine (ISR) for the RTC overflow will toggle an LED in the example below:

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

Note: The OVF bit from the RTC.INTFLAGS must be cleared by writing '1' to it inside the ISR function.

View Code Example on GitHub
Click to browse repository

Tip: The full code example is also available in the Appendix section.

4. RTC Periodic Interrupt

The source clock configuration for this particular example is the same as for the RTC Overflow Interrupt example. The periodic interrupt is enabled by setting the PI bit in the RTC.PITINTCTRL register.

Figure 4-1. RTC.PITINTCTRL – set PI bit

Bit	7	6	5	4	3	2	1	0
								PI
Access								R/W
Reset								0
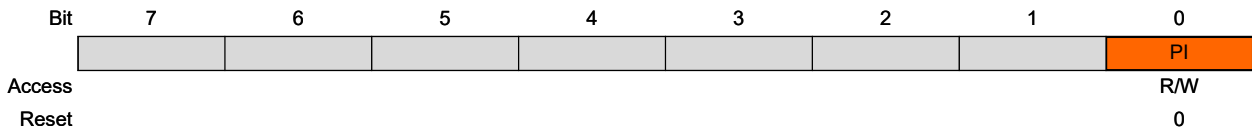
Bit 0 – PI: Periodic Interrupt bit

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

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

The PIT period is set in the RTC.PITCTRLA register. The PIT is enabled by setting the PITEN bit in the RTC.PITCTRLA register.

Figure 4-2. RTC.PITCTRLA – set PITEN bit

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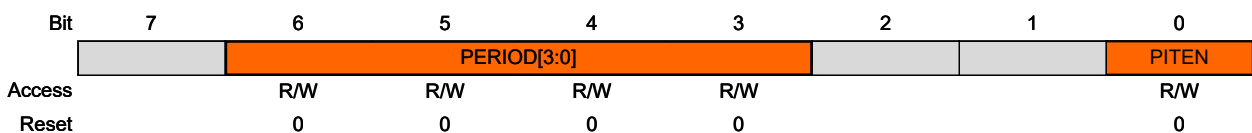
Bits 6:3 – PERIOD[3:0]: Period bits
Writing this bit field selects the number of RTC clock cycles between each interrupt.

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;
```

For the interrupt to occur, the global interrupts must also be enabled:

```
sei();
```

The Interrupt Service Routine (ISR) for the RTC PIT will toggle an LED in the example below:

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

Note: The PI bit from the RTC.PITINTFLAGS must be cleared by writing '1' to it inside the ISR function.

View Code Example on GitHub
Click to browse repository

Tip: The full code example is also available in the Appendix section.

5.   RTC PIT Wake from Sleep

The PIT interrupt can wake the CPU from sleep.

The Sleep mode is configured in the SLPCTRL.CTRLA register. The sleep feature is enabled by setting the SEN bit in the SLPCTRL.CTRLA register.

Figure 5-1. SLPCTRL.CTRLA – set Sleep mode, SEN bit

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 bits
Writing these bits selects the Sleep mode entered when the Sleep Enable bit (SEN) 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 bit
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;
```

The CPU can be put in sleep by calling the following function:

```
sleep_cpu();
```

The PIT interrupt will wake the CPU from sleep. For the interrupt to occur, the global interrupts must be also enabled:

```
sei();
```

The Interrupt Service Routine (ISR) for the RTC PIT will toggle an LED in the following example:

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

Note: The PI bit from the RTC.PITINTFLAGS must be cleared by writing '1' to it inside the ISR function.

View Code Example on GitHub
Click to browse repository

Tip: The full code example is also available in the Appendix section.

6. References

More information about the RTC and PIT operation modes can be found at the following links:

1. ATmega4809 product page: https://www.microchip.com/wwwproducts/en/ATMEGA4809
2. 'megaAVR® 0-Series Manual' (DS40002015)
3. 'ATmega3209/4809 - 48-pin Data Sheet megaAVR® 0-Series' (DS40002016)
4. ATmega4809 Xplained Pro web page: https://www.microchip.com/developmenttools/ProductDetails/atmega4809-xpro

7.   Appendix

Example 7-1. RTC Overflow Interrupt Code Example

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

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

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

void RTC_init(void)
{
    uint8_t temp;

    /* Initialize 32.768kHz Oscillator: */
    /* Disable oscillator: */
    temp = CLKCTRL.XOSC32KCTRLA;
    temp &= ~CLKCTRL_ENABLE_bm;
    /* Enable writing to protected register */
    CPU_CCP = CCP_IOREG_gc;
    CLKCTRL.XOSC32KCTRLA = temp;

    while(CLKCTRL.MCLKSTATUS & CLKCTRL_XOSC32KS_bm)
    {
        ; /* Wait until XOSC32KS becomes 0 */
    }

    /* SEL = 0 (Use External Crystal): */
    temp = CLKCTRL.XOSC32KCTRLA;
    temp &= ~CLKCTRL_SEL_bm;
    /* Enable writing to protected register */
    CPU_CCP = CCP_IOREG_gc;
    CLKCTRL.XOSC32KCTRLA = temp;

    /* Enable oscillator: */
    temp = CLKCTRL.XOSC32KCTRLA;
    temp |= CLKCTRL_ENABLE_bm;
    /* Enable writing to protected register */
    CPU_CCP = CCP_IOREG_gc;
    CLKCTRL.XOSC32KCTRLA = temp;

    /* Initialize RTC: */
    while (RTC.STATUS > 0)
    {
        ; /* Wait for all register to be synchronized */
    }

    /* Set period */
    RTC.PER = RTC_EXAMPLE_PERIOD;

    /* 32.768kHz External Crystal Oscillator (XOSC32K) */
    RTC.CLKSEL = RTC_CLKSEL_TOSC32K_gc;

    /* Run in debug: enabled */
    RTC.DBGCTRL |= RTC_DBGRUN_bm;

    RTC.CTRLA = RTC_PRESCALER_DIV32_gc  /* 32 */
    | RTC_RTCEN_bm              /* Enable: enabled */
    | RTC_RUNSTDBY_bm;          /* Run In Standby: enabled */

    /* Enable Overflow Interrupt */
    RTC.INTCTRL |= RTC_OVF_bm;
}

void LED0_init(void)
{
    /* Make High (OFF) */
    PORTB.OUT |= PIN5_bm;
```

```
        /* Make output */
        PORTB.DIR |= PIN5_bm;
}

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

ISR(RTC_CNT_vect)
{
        /* Clear flag by writing '1': */
        RTC.INTFLAGS = RTC_OVF_bm;

        LED0_toggle();
}

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

        /* Enable Global Interrupts */
        sei();

        while (1)
        {
        }
}
```

Example 7-2. RTC Periodic Interrupt Code Example

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

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

void RTC_init(void)
{
        uint8_t temp;

        /* Initialize 32.768kHz Oscillator: */
        /* Disable oscillator: */
        temp = CLKCTRL.XOSC32KCTRLA;
        temp &= ~CLKCTRL_ENABLE_bm;
        /* Enable writing to protected register */
        CPU_CCP = CCP_IOREG_gc;
        CLKCTRL.XOSC32KCTRLA = temp;

        while(CLKCTRL.MCLKSTATUS & CLKCTRL_XOSC32KS_bm)
        {
             ; /* Wait until XOSC32KS becomes 0 */
        }

        /* SEL = 0 (Use External Crystal): */
        temp = CLKCTRL.XOSC32KCTRLA;
        temp &= ~CLKCTRL_SEL_bm;
        /* Enable writing to protected register */
        CPU_CCP = CCP_IOREG_gc;
        CLKCTRL.XOSC32KCTRLA = temp;

        /* Enable oscillator: */
        temp = CLKCTRL.XOSC32KCTRLA;
        temp |= CLKCTRL_ENABLE_bm;
        /* Enable writing to protected register */
        CPU_CCP = CCP_IOREG_gc;
        CLKCTRL.XOSC32KCTRLA = temp;

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

```
            ; /* Wait for all register to be synchronized */
    }

    /* 32.768kHz External Crystal Oscillator (XOSC32K) */
    RTC.CLKSEL = RTC_CLKSEL_TOSC32K_gc;

    /* Run in debug: enabled */
    RTC.DBGCTRL = RTC_DBGRUN_bm;

    RTC.PITINTCTRL = RTC_PI_bm; /* Periodic Interrupt: enabled */

    RTC.PITCTRLA = RTC_PERIOD_CYC32768_gc /* RTC Clock Cycles 32768 */
                 | RTC_PITEN_bm; /* Enable: enabled */
}

void LED0_init(void)
{
    /* Make High (OFF) */
    PORTB.OUT |= PIN5_bm;
    /* Make output */
    PORTB.DIR |= PIN5_bm;
}

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

ISR(RTC_PIT_vect)
{
    /* Clear flag by writing '1': */
    RTC.PITINTFLAGS = RTC_PI_bm;

    LED0_toggle();
}

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

    /* Enable Global Interrupts */
    sei();

    while (1)
    {
    }
}
```

Example 7-3. RTC PIT Wake from Sleep Code Example

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

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

void RTC_init(void)
{
    uint8_t temp;

    /* Initialize 32.768kHz Oscillator: */
    /* Disable oscillator: */
    temp = CLKCTRL.XOSC32KCTRLA;
    temp &= ~CLKCTRL_ENABLE_bm;
    /* Enable writing to protected register */
    CPU_CCP = CCP_IOREG_gc;
    CLKCTRL.XOSC32KCTRLA = temp;

    while(CLKCTRL.MCLKSTATUS & CLKCTRL_XOSC32KS_bm)
```

```
        {
            ; /* Wait until XOSC32KS becomes 0 */
        }

        /* SEL = 0 (Use External Crystal): */
        temp = CLKCTRL.XOSC32KCTRLA;
        temp &= ~CLKCTRL_SEL_bm;
        /* Enable writing to protected register */
        CPU_CCP = CCP_IOREG_gc;
        CLKCTRL.XOSC32KCTRLA = temp;

        /* Enable oscillator: */
        temp = CLKCTRL.XOSC32KCTRLA;
        temp |= CLKCTRL_ENABLE_bm;
        /* Enable writing to protected register */
        CPU_CCP = CCP_IOREG_gc;
        CLKCTRL.XOSC32KCTRLA = temp;

        /* Initialize RTC: */
        while (RTC.STATUS > 0)
        {
            ; /* Wait for all register to be synchronized */
        }

        /* 32.768kHz External Crystal Oscillator (XOSC32K) */
        RTC.CLKSEL = RTC_CLKSEL_TOSC32K_gc;

        /* Run in debug: enabled */
        RTC.DBGCTRL = RTC_DBGRUN_bm;

        RTC.PITINTCTRL = RTC_PI_bm; /* Periodic Interrupt: enabled */

        RTC.PITCTRLA = RTC_PERIOD_CYC32768_gc /* RTC Clock Cycles 32768 */
                     | RTC_PITEN_bm; /* Enable: enabled */
}

void LED0_init(void)
{
    /* Make High (OFF) */
    PORTB.OUT |= PIN5_bm;
    /* Make output */
    PORTB.DIR |= PIN5_bm;
}

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

ISR(RTC_PIT_vect)
{
    /* Clear flag by writing '1': */
    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();

    /* Enable Global Interrupts */
    sei();

    while (1)
    {
        /* Put the CPU in sleep */
        sleep_cpu();
```

```
        /* The PIT interrupt will wake the CPU */
    }
}
```

The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Customer Change Notification Service

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at http://www.microchip.com/. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.

- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, AnyRate, AVR, AVR logo, AVR Freaks, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, Heldo, JukeBlox, KeeLoq, Kleer, LANCheck, LINK MD, maXStylus, maXTouch, MediaLB, megaAVR, MOST, MOST logo, MPLAB, OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, Prochip Designer, QTouch, SAM-BA, SpyNIC, SST, SST Logo, SuperFlash, tinyAVR, UNI/O, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

ClockWorks, The Embedded Control Solutions Company, EtherSynch, Hyper Speed Control, HyperLight Load, IntelliMOS, mTouch, Precision Edge, and Quiet-Wire are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, motorBench, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

Silicon Storage Technology is a registered trademark of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

Quality Management System Certified by DNV

ISO/TS 16949

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.