# NUC130/140CN Errata Sheet

Errata Sheet for 32-bit NuMicro® Family

**Rev. 1.00 — May 6, 2013**

## Document Information

| | |
|---|---|
| **Abstract** | This errata sheet describes the functional problems known at the release date of this document. |
| **Apply to** | NUC130/140CN. |

## Table of Contents

# 1 Overview

| Functional Problem | Description |
|---|---|
| ADC External Trigger | When enabling ADC external trigger function, it may cause an additional ADC start trigger. |
| ADC VALID Flag | The VALID flag of ADC data register may be missed when polling it by CPU. |
| ADST Re-trigger | After ADC finishes the conversion and clears the ADST bit, software must wait one ADC clock cycle before setting the ADST bit to 1 again. |
| ADC Continuous/Single-Cycle Scan Mode | If user clears the ADST bit to 0 in Continuous or Single-Cycle Scan mode during ADC conversion, the conversion result of the smallest enabled ADC channel may be incorrect. |
| ADC_N Setting | In ADC single conversion mode, if the ADC clock source is HCLK, the ADC_N cannot be set to 0. |
| ADC PDMA Transfer | If user transfers the ADC conversion result with PDMA, only the 12 bits conversion result will be transferred. The VALID bit, OVERRUN bit and the sign extension bits will be ignored. |
| BOD Low Power Mode Enable | Brown-out low power control bit will not be reset by BOD reset. |
| BOD Reload | When system recover from BOD reset state, BOD settings of user configuration will be reloaded. |
| BOD Reset | If WDT generates WDT time-out reset in BOD reset status. System will exit BOD reset status and start booting for 960us. |
| GPIO Wake-up Trigger Level | GPIO wake-up function failed if trigger pin is in active state before system enters power down. |
| GPIO Wake-up Transient | GPIO may wake-up by I/O transient no matter it is configured as rising edge or falling edge wake-up. |
| GPIO Double Interrupts | GPIO wake-up function will cause double I/O interrupt events if GPIO de-bounce function is enabled. |
| PWM Capture | When PWM capture input channel is enabled and PWM capture input channel has a transition before PWM timer start to count, PWM timer will be halted. |
| Power-down Enable Bit Cleared | The power-down enable bit (PWR_DOWN_EN) is cleared by hardware if any GPIO interrupt occurred before WFI is executed. |

| | |
|---|---|
| RS-485 peripheral Clock Setting | When peripheral clock frequency is higher than system clock, RS-485 mode may receive wrong data. |
| RTC Wake-up Flag | If system has been woken up by RTC tick or alarm interrupt, the internal RTC wake-up signal will always keep one RTC peripheral clock period even if the tick or alarm interrupt flag has been cleared to 0. |
| Timer Module Reset | Timer 1 register cannot be accessed when Timer 0 is doing module reset.<br>Timer 0 register cannot be accessed when Timer 1 is doing module reset.<br>Timer 3 register cannot be accessed when Timer 2 is doing module reset.<br>Timer 2 register cannot be accessed when Timer 3 is doing module reset. |
| UART Error Flags | If one of the Error flags (BIF/FEF/PEF) is set, the flag cannot write '1' to be cleared. |
| UART Buffer Error Flag | When one of Error Flags (BIF/FEF/PEF) is set, the BUF_ERR_IF flag will be set at the same time. |
| WDT Wake-up Flag | When writing 1 to clear the WTWKF flag, the internal WDT wake-up signal will always keep one WDT peripheral clock period then transit to 0 even if WTWKF flag is read as 1 by CPU. |

# 2 Functional Problems

## 2.1 ADC External Trigger

**Description:**

If user enables the external trigger function, user can trigger the A/D conversion by STADC pin. The trigger condition is determined by TRGCOND[1:0] in ADCR register. It can be low-level trigger, high-level trigger, falling-edge trigger and rising-edge trigger.

**Problem:**

If software enables ADC with the following flow, the ADC controller will generate an additional ADC start trigger.

1. Force the STADC pin at 0V.

2. Enable ADC clock (APBCLK[28]).

3. After enabling ADC clock, configure the ADC as the following conditions within 4 system clock cycles.

- Enable STADC pin external trigger function.
- External trigger condition is falling-edge trigger.
- Enable ADC analog circuit (ADCR[0]).

**Workaround:**

To avoid the additional ADC start trigger, user must enable the external trigger function or enable ADC analog circuit at least 4 system clock after enabling ADC clock.

## 2.2 ADC VALID Flag

**Description:**

When ADC finishes a conversion, ADC will write the conversion result to the corresponding ADC data register and set the VALID bit to 1. User can find the VALID flag in data register and status register.

**Problem:**

The VALID bit of ADC data register is set by hardware when ADC data conversion is ready

and it is cleared by CPU reading ADC data register. However, if hardware set VALID bit and software read VALID bit at the same time, the VALID bit won't be set. It causes the VALID bit may be always 0. For example, if user enables ADC channel 0 to convert the input signal and polling channel 0 data register to check whether data valid, it may never get data valid because VALID bit is suppressed by read clear.

**Workaround:**

User should poll the ADF bit or the VALID bit of ADC status register instead of polling the VALID bit of ADC data register to check if the conversion is finished or not.

## 2.3 ADST Re-trigger

**Description:**

In Single Mode and Single Cycle Scan mode, the ADST bit will be cleared to 0 automatically after ADC finishes the conversion. In Continuous Scan mode, the conversion will be stopped if the ADST bit is cleared to 0 by software. When every ADST is 0, user can set ADST to 1 to re-trigger ADC conversion.

**Problem:**

After the ADST bit is cleared to 0, if the ADST bit is set to 1 immediately, the ADC may not work.

**Workaround:**

After the ADST bit is cleared to 0, software must wait at least one ADC clock cycle before setting the ADST bit to 1 again. For example, if ADC clock rate is 10 MHz, the delay loop of the following example must be greater than 0.1us (1/10MHz).

```
ADC->ADCHER = 0xFF;              // Enable all ADC channels
ADC->ADCR = 9;                   // Single Cycle Scan mode
ADC->ADCR_BITS.ADST = 1;         // Trigger ADC
while(ADC->ADSR_BITS.ADF==0);    // Wait ADC finishes the conversion
ADC->ADSR = 0x1;                 // Clear the ADF bit
for(u32DelayCount=0; u32DelayCount<0x1000; u32DelayCount++);    // Delay loop
ADC->ADCR_BITS.ADST = 1;         // Re-trigger ADC
```

## 2.4 ADC Continuous/Single-Cycle Scan Mode

**Description:**

If the ADST bit is cleared to 0 by software to stop Continuous or Single-Cycle Scan mode, ADC controller will finish the current conversion and write the conversion result to the data register of the smallest enabled ADC channel.

**Problem:**

After clearing the ADST bit in Continuous or Single-Cycle Scan mode, the conversion result of the smallest enabled ADC channel may be incorrect. The following example shows a case of enabling all ADC channels in Continuous Scan Mode. It set ADST=0 to stop ADC arbitrarily when ADC conversion and causes the result of ADC channel 0 may be the result of any enabled channel.

```
ADC->ADCHER = 0xFF;              // Enable all ADC channels
ADC->ADCR = 0xD;                 // Continuous scan mode
ADC->ADCR_BITS.ADST = 1;         // Trigger ADC
…
ADC->ADCR BITS.ADST = 0;         // Stop ADC when ADC converting
…
// The ADDR[0] may be the result of any ADC channel.
u32AdcData = ADC->ADDR[0];
```

**Workaround:**

If enabling more than one ADC channel in Continuous or Single-Cycle Scan mode, the user should read the valid conversion result before clearing the ADST bit. As shown in the following example, user needs to read available channel data before stopping ADC.

```
ADC->ADCHER = 0xFF;              // Enable all ADC channels
ADC->ADCR = 0xD;                 // Continuous scan mode
ADC->ADCR_BITS.ADST = 1;         // Trigger ADC
While(ADC->ADSR_BITS.ADF==0); // Wait ADC finishes the conversion
for(u32DataCount=0; u32DataCount<8; u32DataCount++)
{
    u32AdcData = ADC->ADDR[u32DataCount]; // Read the valid data
    …
}
```

```
    ADC->ADCR BITS.ADST = 0;        // Stop ADC
```

## 2.5 ADC_N Setting

**Description:**

When A/D conversion starts, only the enabled ADC channel will be converted.

**Problem:**

In ADC single conversion mode, if the ADC clock source is HCLK and the ADC_N is set to 0, ADC controller always converts the ADC channel 0 no matter the channel 0 is enabled or not.

**Workaround:**

In ADC single conversion mode, if the ADC clock source is HCLK, the ADC_N cannot be set to 0.

## 2.6 ADC PDMA Transfer

**Description:**

If user sets the PTEN bit to 1 to generate a request to PDMA, the valid ADC conversion result as well as the relative flags should be transferred to a user-specified RAM space by PDMA controller.

**Problem:**

If user transfers the ADC conversion result with PDMA, only the 12 bits conversion result will be transferred. The VALID bit, OVERRUN bit and the sign extension bits will be ignored.

**Workaround:**

User can set PDMA to do 16-bit transfer of each data and keep the conversion data in unsigned format when work with PDMA. By doing this, the data in SRAM after PDMA transfer would be ADC conversion result data in unsigned format.

## 2.7 BOD Low Power Mode Enable

**Description:**

BOD (Brown-Out Detection) in low power mode will be reset to normal mode when reset occurred. In BOD normal mode, BOD response time is faster than BOD low power mode. The maximum response time of BOD detection is 100ms in low power mode. When BOD low power mode disabled, the maximum response time of BOD detection is 1/10KHz (100us).

**Problem:**

When BOD works in low power mode, it won't be reset to normal mode when BOD reset.

**Workaround:**

If BOD reset flag, RSTS_BOD(RSTSRC[4]), be set after BOD reset, software can disable low power mode function to speed up the response time of BOD detection.

## 2.8 BOD Reload

**Description:**

When system reboot from BOD reset state, BOD settings will be kept.

**Problem:**

When system reboot from BOD reset state, BOD settings of user configuration (Brown-out enable, Brown-out reset enable and Brown-out detector threshold voltage) will be reloaded. It causes the settings of BOD control registers to be reloaded as BOD settings of user configuration.

**Workaround:**

Do not modify BOD_OUT(BODCR[6]), BOD_VL(BODCR[2:1]) and BOD_EN(BODCR[0]) by software to keep consistent with BOD settings of user configuration. If it is necessary to change BOD settings, user can modify BOD settings of user configuration and reset system.

## 2.9 BOD Reset

**Description:**

When BOD detected and BOD reset enabled, the system will be held in BOD reset state until the supply voltage comes back to BOD detection level.

**Problem:**

When BOD detected and BOD reset enabled, the system will be held in BOD reset state. However, in BOD reset state, WDT still works if it is enabled. If WDT generates WDT time-out reset in BOD reset state, the system will exit BOD reset status and start booting for 960us. After 960us from booting, the system will back to be held in BOD reset state.

**Workaround:**

Before system enters BOD reset state, user can disable watchdog timer to prevent WDT generated time-out. As shown in the following example, user enables BOD interrupt mode and enables IRQ0 for BOD interrupt in main function. User can disable watchdog timer in BOD interrupt handler and then set BOD reset mode. After setting to BOD reset mode, system will enter BOD reset state immediately.

```
Void main(void)
{
    SYS->BODCR &= ~SYS_BODCR_BOD_RSTEN_Msk; // Set BOD interrupt mode - BODCR[3]
    NVIC_EnableIRQ(BOD_IRQn);   //Enable IRQ0 for BOD interrupt
}

void BOD_IRQHandler(void)
{
    WDT->WTCR = WDT->WTCR & ~WDT_WTCR_WTE_Msk;  //Disable watchdog timer - WTCR[7]
    SYS->BODCR |= SYS_BODCR_BOD_RSTEN_Msk;  //Set BOD reset mode - BODCR[3]
    /* System will enter BOD reset state */
    ...
}
```

## 2.10 GPIO Wake-up Trigger Level

**Description:**

All GPIO interrupts can be used to wake up CPU. The GPIO interrupt sources could be triggered by rising-edge, falling-edge, high-level or low-level.

**Problem:**

System cannot be woken up by GPIO edge-trigger interrupt while specify I/O pin status is at active edge-trigger level before entering Power-down mode.

For example,

If an I/O pin status is at low before system enters Power-down mode, system cannot be woken up by this GPIO when it is set as low edge-trigger interrupt.

If an I/O pin status is at high before system enters Power-down mode, system cannot be woken up by this GPIO when it is set as high edge-trigger interrupt.

**Workaround:**

Software must make sure the input I/O status is stable and do not match with the interrupt active state before entering power-down mode.

For example, if configuring the wake-up event occurred by I/O rising-edge trigger, user must make sure the I/O status of specified pin is at low level before entering Power-down mode; and if configure I/O falling-edge trigger to trigger a wake-up event, user must make sure the I/O status of the specified pin is at high level before entering Power-down mode.

## 2.11 GPIO Wake-up Transient

**Description:**

System cannot be woken up by GPIO edge-trigger interrupt while I/O pin status is matched with the interrupt active state before system enters Power-down mode. For example, if an I/O state is low before entering power-down and set the I/O to be falling edge trigger, this I/O should not be able to wake up system.

**Problem:**

The transient of some I/O pins will induce internal glitch, which causes system be woken up by edge-trigger interrupt no matter the I/O pin status matches with interrupt active state or not before system enters Power-down mode.

**Workaround:**

In general, user needs to make sure the I/O state before entering power down to make sure wake-up success. In other words, to wake up by rising-edge trigger, the user must make I/O state low when power down. To wake up by falling-edge trigger, the user must make I/O state high when power down.

If user doesn't want to wake up system by GPIO, the user can disable GPIO interrupt to forbid the wake-up function of GPIO.

## 2.12 GPIO Double Interrupts

**Description:**

System can be woken up by specify GPIO interrupt. Only once interrupt event occurred on this GPIO pin when system woken up successfully.

**Problem:**

If the specified GPIO wake-up pin with enabled de-bounce function, while system has been woken up by this GPIO pin, system will encounter two GPIO interrupt events. One is caused by wake-up function, the other one is caused by I/O de-bounce function after system wake-up.

**Workaround:**

User should be disable the I/O de-bounce function before entering Power-down mode to avoid the second interrupt event occurred after system wake-up which caused by I/O de-bounce function. Then, software can enable the I/O de-bounce function again while system runs in normal mode.

## 2.13 Power-down Enable Bit

**Description:**

The power-down enable bit – PWR_DOWN_EN (PWRCON[7]) is used to enter Power-down mode. User needs to execute WFI instruction with PWR_DOWN_EN = 1 and SLEEPDEEP = 1 (SCR[2]) to enter power down. PWR_DOWN_EN bit will be self-clear when system wakes up.

**Problem:**

The power-down enable bit - PWR_DOWN_EN (PWRCON[7]) may be cleared by hardware if any GPIO interrupt or specific wake-up source interrupt occurred before WFI executed. It causes failed to enter power down.

**Workaround:**

The power-down enable bit - PWR_DOWN_EN(PWRCON[7]) needs to be enabled by software again in GPIO interrupt handler and specific wake-up interrupt handler to make sure the power-down enable bit is enabled. For example, if system has WDT time-out interrupt and GPIO PA.0 interrupt wake-up sources, software should enable the PWR_DOWN_EN bit in WDT_IRQn and GPAB_IRQn handler.

```
void GPAB_IRQHandler(void)
{
    GPIOA->ISRC |= GPIOA->ISRC; // Clear interrupt source flag
    SYSCLK->PWRCON |= SYSCLK_PWRCON_PWR_DOWN_EN_Msk; // Enable PWR_DOWN_EN bit
}

void WDT_IRQHandler(void)
{
    // Clear WDT time-out interrupt and wake-up flag
    WDT->WTCR |= (WDT_WTCR_WTIF_Msk | WDT_WTCR_WTWKF_Msk);
    SYSCLK->PWRCON |= SYSCLK_PWRCON_PWR_DOWN_EN_Msk; // Enable PWR_DOWN_EN bit
}
```

## 2.14 PWM Capture

**Description:**

When PWM timer is enabled in capture mode and PWM capture input channel has a transition (rising/falling), the PWM counter value will be latched into CRLRn/CFLRn registers.

**Problem:**

When PWM timer is enabled in capture mode and start PWM counter, PWM timer will be halted if input signal transient before PWM counter really starts. (PWM counter may delay one PWM counter clock source when counter is enabled.)

**Workaround:**

Before the capture input channel - CAPENR is enabled, user should enable PWM timer – CHxEN(PCR) first and make sure PWM timer starts counting by checking PWM data register - PDRx is not equal to zero. As shown in the following example, user enables PWM Timer2 first and then software waits until PDR2 is not equal to zero to make sure PWM Timer2 start to count. After Timer2 has begun to count, user enables capture2 input path – CAPNER[2].

```
void main(void)
{
    ...
    /* Set the PWMA channel 2 for capture function */
    _PWM_SET_TIMER_PRESCALE(PWMA,PWM_CH2, 1); //Set Timer2 prescaler
    _PWM_SET_TIMER_CLOCK_DIV(PWMA,PWM_CH2,PWM_CSR_DIV1);//Set Timer2 clock divider
    _PWM_SET_TIMER_AUTO_RELOAD_MODE(PWMA,PWM_CH2); //Set Timer2 auto-reload mode
    _PWM_ENABLE_CAP_FUNC(PWMA, PWM_CH2); //Enable PWM2 capture function
    PWMA->CNR2 = 0xFFFF; //Set Timer2 loaded value
    _PWM_ENABLE_CAP_FALLING_INT(PWMA, PWM_CH2); //Enable falling edge interrupt
    NVIC_EnableIRQ((IRQn_Type)(PWMA_IRQn)); //Enable PWMA NVIC interrupt

    _PWM_ENABLE_TIMER(PWMA, PWM_CH2); //Enable PWM Timer2 – CH2EN(PCR[16])
    while(PWMA->PDR2==0); //Wait until PWM Timer2 start to count – PDR2
    _PWM_ENABLE_CAP_IN(PWMA, PWM_CH2); //Enable capture2 input path – CAPENR[2]
    ...
}
```

## 2.15 RS485 Peripheral Clock Setting

**Description:**

The UART peripheral clock frequency could be higher or lower than system clock frequency, even UART is working in RS-485 mode.

**Problem:**

When UART peripheral clock frequency is higher than system clock, the received data will be incorrect in RS-485 mode.

**Workaround:**

Set the UART source clock frequency divider (UART_N) and make sure UART peripheral clock frequency (RS-485 operation clock frequency) is less than or equal to half of system clock. For example, if the system clock frequency is 50 MHz, the peripheral clock frequency of UART must be lower or equal to 25 MHz.

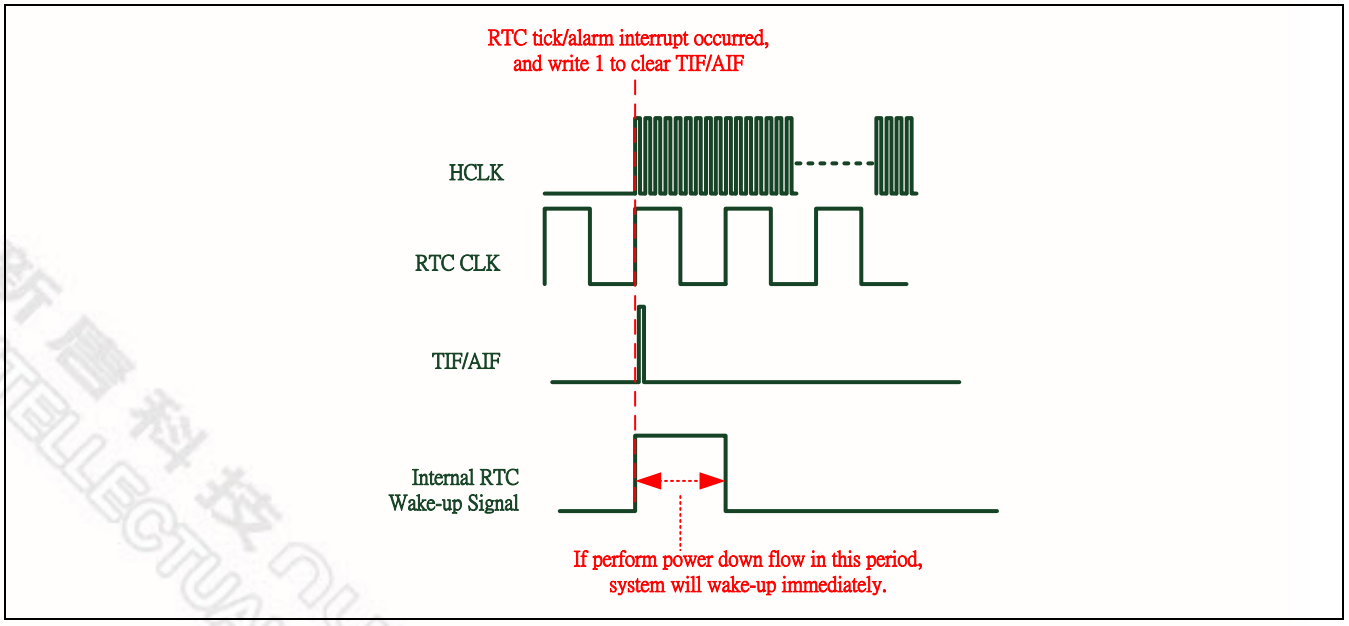
## 2.16 RTC Wake-up Flag

**Description:**

When system has been woken up by RTC tick or alarm interrupt event, software may write 1 to clear tick or alarm interrupt flag and then perform power-down flow to enter Power-down mode again.

**Problem:**

In general case, system can enter Power-down mode normally while RTC tick interrupt flag (TIF) and alarm interrupt flag (AIF) are all cleared. While system has been woken up by RTC interrupt source, the internal RTC wake-up signal will also keep to 1. When software wants writing 1 to clear TIF or AIF bit, TIF and AIF will clear to 0 immediately but the internal RTC wake-up signal will keep one RTC peripheral clock period then transit to 0 automatically. If user perform power-down flow while internal RTC wake-up signal keeps 1, system will be woken up immediately by internal RTC wake-up signal. The detailed timing is as follows.



**Workaround:**

If user wants to enter Power-down mode after previous RTC wake-up event occurred, user must delay at least one RTC peripheral clock (about 30.52 us) to avoid entering power mode failed.

## 2.17 Timer Module Reset

**Description:**

Software can reset the individual Timer channel to default settings (chip power-on settings) by setting IPRSTC2 bit[5:2].

**Problem:**

Timer 0, 1 are running on APB1 bus and Timer 2, 3 are running on APB2 bus. If specified Timer is at reset state, the others Timer register of the same bus cannot be accessed.

For example, If Timer 1 is at reset state, Timer 0 register cannot be accessed until Timer 1 return to normal state. If Timer 0 is at reset state, Timer 1 register cannot be accessed until Timer 0 return to normal state. If Timer 3 is at reset state, Timer 2 register cannot be accessed until Timer 3 return to normal state. If Timer 2 is at reset state, Timer 3 register cannot be accessed until Timer 2 return to normal state.

**Workaround:**

Software must avoid resetting Timer by setting IPRSTC2 register while Timer is running. It is recommended to set CRST bit in TCSR register to reset timer function if necessary.

## 2.18 UART Error Flags

**Description:**

When Break Interrupt Flag or Parity Error Flag or Frame Error Flag (BIF/PEF/FEF) is set, user needs to write 1 to clear relative error flags.

**Problem:**

When Break Interrupt Flag, Parity Error Flag or Frame Error Flag (BIF/PEF/FEF) is set, user cannot write “1” to clear relative error flags.

**Workaround:**

When one of the error flags (BIF/PEF/FEF) is set, user needs to flush FIFO (TFR/RFR) to clear relative error flags.

## 2.19 UART Buffer Error Flag

**Description:**

When TX or RX FIFO is overflow (TX_OVER_IF=1 or RX_OVER_IF=1) and the Buffer Error interrupt function is enabled, the Buffer Error interrupt will be generated (BUF_ERR_IF=1).

**Problem:**

When Break Interrupt Flag or Parity Error Flag or Frame Error Flag (BIF/PEF/FEF) is set, the BUF_ERR_IF flag is set to 1 at the same time.

**Workaround:**

If Buffer Error flag (BUF_ERR_IF) is set, check TX or RX FIFO is overflown or not. If TX or RX FIFO is not overflown, user needs to flush FIFO to clear the Error flag (BIF/PEF/FEF).

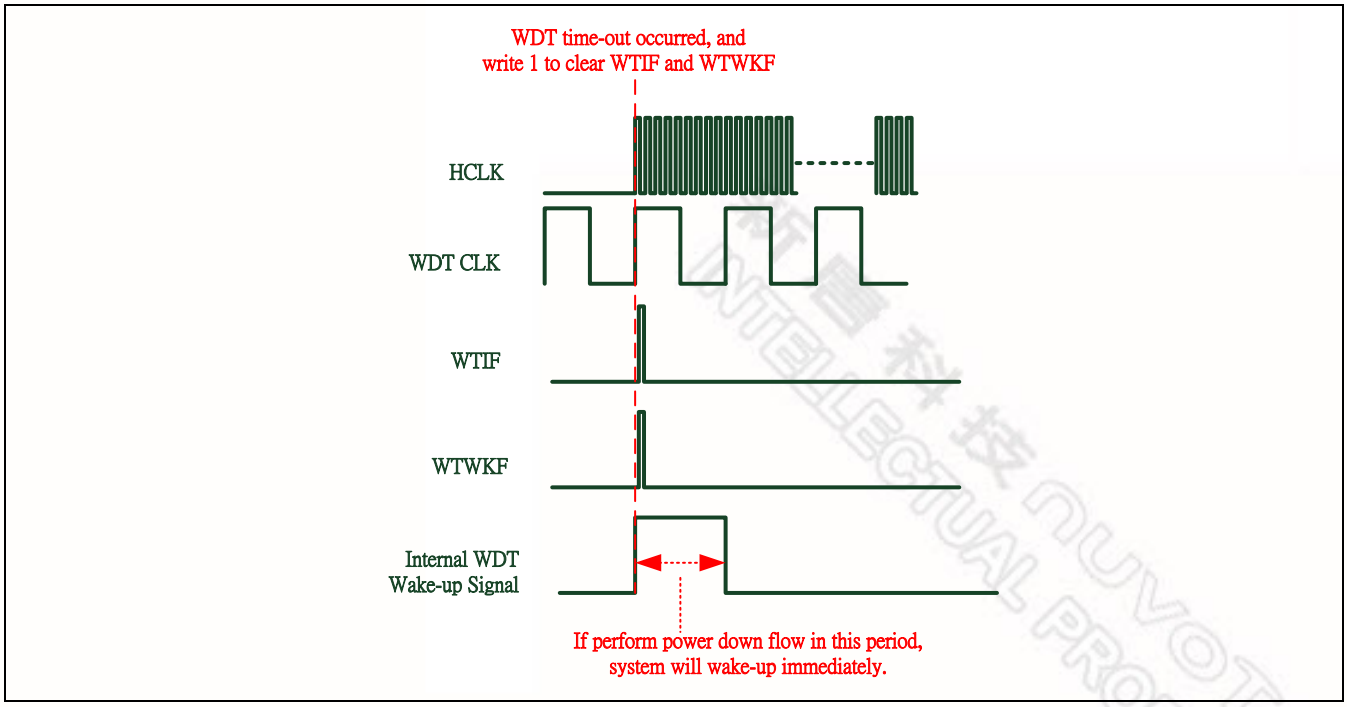
## 2.20 WDT Wake-up Flag

**Description:**

When system is woken up by WDT time-out interrupt, the WTWKF (WDT wake-up flag) bit will set to 1 and software can write 1 to clear this bit. Then user can enter power down again by performing power-down flow.

**Problem:**

In general case, system can enter Power-down mode normally while WTWKF is 0. While system wake-up by WDT time-out interrupt the WTWKF will be set to 1 and the internal WDT wake-up signal is also kept as 1. When software writes 1 to the WTWKF bit, the WTWKF bit is cleared to 0 immediately but the internal WDT wake-up signal will keep one WDT peripheral clock period then transit to 0. If user performs power-down flow while internal WDT wake-up signal keeps 1, system will be woken up immediately by the internal WDT wake-up signal. The detailed timing is as follows.




**Workaround:**

If user wants to enter Power-down mode after previous WDT time-out wake-up occurred, the user must delay one WDT peripheral clock as least to avoid entering power mode failed. For example, if the current WDT source clock is 10 kHz, the software delay time should be larger than 100 us for entering the next Power-down mode normally.

## Revision History

| Revision | Date | Description |
|---|---|---|
| 1.00 | May 6, 2013 | Initially issued. |

## Important Notice

**Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".**

**Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.**

**All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.**