NUC123xxxAN Series Errata Sheet

Errata Sheet for 32-bit NuMicro® Family

Rev. 1.04 —May 19, 2016

Document Information

Abstract	This errata sheet describes the functional problems known at the release date of this document.
Apply to	NUC123xxxAN Series.

Table of Contents

1 Overview

Functional Problem	Description
ADC External Trigger	When ADC external trigger function is enabled, it may cause an additional ADC start trigger.
ADC VALID Bit	The VALID bit of ADC data register will not be set to 1 if program uninterruptedly reads data register.
BOD Low Power Mode Enable	The Brown-out low power control bit will not be reset by BOD reset.
BOD Reload	When system recovers from BOD reset state, BOD settings in user configuration will be reloaded.
Data Flash	The program code in APROM should read Data Flash by using ISP Read command when Data Flash variable size is disabled.
GPIO Wake-up Trigger Level	The GPIO wake-up function failed if selected pin is already in active state before system enters Power-down mode.
GPIO Wake-up Transient	GPIO may be woken up by I/O transient no matter it is configured as rising edge or falling edge wake-up.
GPIO Debounce Function Fail	GPIO Hardware debounce function cannot filter all noise.
RS-485 Peripheral Clock Setting	When the UART peripheral clock frequency is higher than system clock, UART may receive incorrect data in RS-485 mode.
RS-485 Baud Rate Equation	User should select Mode 2 of baud rate setting if UART works in RS-485 mode.
Timer IP Reset	Timer 1 register cannot be accessed when Timer 0 is doing IP reset. Timer 0 register cannot be accessed when Timer 1 is doing IP reset. Timer 3 register cannot be accessed when Timer 2 is doing IP reset. Timer 2 register cannot be accessed when Timer 3 is doing IP reset.
USB Endpoint 6	Endpoint 6 event interrupt function does not work properly at specific conditions.

2 Functional Problems

2.1 ADC External Trigger

Description:

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

Problem:

The ADC controller will generate an additional conversion with the programming sequence listed below.

1. Force the STADC pin at 0V.
2. Enable ADC clock (APBCLK[28]).
3. Complete the following operations within 4 system clock cycles.
    - Enable STADC pin external trigger with falling-edge condition.
    - Enable ADC analog circuit (ADCR[0]).

Workaround:

After ADC peripheral clock is enabled, it is necessary to delay at least 4 system clocks before the external trigger function or ADC analog circuit is enabled.

2.2 ADC VALID Bit

Description:

When an ADC conversion is completed, the result is saved to the data register of the corresponding channel, and VALID bit of both data and status register are set to 1. Reading data register will clear the VALID bit.

Problem:

The VALID bit of ADC data register is set by hardware when an ADC data conversion is ready and is cleared by CPU reading the ADC data register. However, if hardware sets the VALID bit and software reads the VALID bit at the same time, the VALID bit will not be set, which causes the VALID bit probably always 0. For example, if the user enables ADC channel 0 to convert the input signal and poll channel 0 data register to check whether the data is valid, it may never get data valid because the VALID bit is suppressed by clear read.

Workaround:

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

2.3 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, the BOD response time is faster than that in BOD Low Power mode. The maximum response time of BOD detection is 100ms in Low Power mode. When BOD Low Power mode is disabled, the maximum response time of BOD detection is 1/10 kHz (100us).

Problem:

When BOD works in Low Power mode, it will not be reset to Normal mode when BOD reset.

Workaround:

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

2.4 BOD Reload

Description:

When system reboots from BOD reset state, BOD settings will be reset.

Problem:

When system recovers from BOD reset state, BOD settings in 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 in user configuration.

Workaround:

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

2.5 Data Flash

Description:

When Data Flash variable size is disabled (User Config0[2], DVFSEN = 0), the Data Flash size is 4 KB and the start address is fixed at 0x1f000. The program code in APROM can read Data Flash by using memory read instructions or ISP Read command.

Problem:

When Data Flash variable size is disabled, program code in APROM is unable to read Data Flash by using memory read instructions.

Workaround:

The program code in APROM should read Data Flash by using the ISP Read command when Data Flash variable size is disabled. The following example shows how to read a word at address offset 0x1c:

```
    FMC->ISPCMD = 0x00;                 // ISP Read
    FMC->ISPADR = 0x1f000 + 0x1c;  // Data address
    FMC->ISPTRG = 0x1;                  // Trigger to start ISP process
    while(FMC->ISPTRG);                 // Waiting for ISP command ready
    u32Data = FMC->ISPDAT;              // Get the data in Data Flash
```

2.6 GPIO Wake-up Trigger Level

Description:

All GPIO interrupts can be used to wake up CPU. The types of GPIO interrupt can be triggered include rising-edge, falling-edge, high-level or low-level.

Problem:

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

For example, if a GPIO Pin status is at low before entering Power-down mode, system cannot be woken up by this GPIO when it is set as low edge-trigger interrupt.

Similarly, if the status of a GPIO pin status is at high before entering Power-down mode, system cannot be woken up by this GPIO when it is set as high edge-trigger interrupt.

Workaround:

Software should make sure the input I/O status used to wake up system is stable and does not match with the interrupt active state before entering Power-down mode.

For example, if the interrupt is set as rising-edge triggered, the user should make sure the I/O status of specified pin is at low level before entering Power-down mode; if the configuration is set as falling-edge triggered, the user should make sure the I/O status of selected pin is at high level before entering Power-down mode.

2.7 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 entering Power-down mode. For example, if an I/O state is low before entering Power-down and the I/O is set to be falling edge triggered, this I/O should not be able to wake system up.

Problem:

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

Workaround:

To successfully wake system up, the user needs to set the I/O state before entering Power-down. In other words, if user wants to wake system by rising-edge trigger, I/O state must be set to low when Power-down. If user wants to wake system up by falling-edge trigger, user should make sure the I/O state is high in Power-down mode.

If user does not want system to be woken up by GPIO, the GPIO interrupt should be disabled.

2.8 GPIO Debounce Function Failed

Description:

GPIO debounce function failed.

Problem:

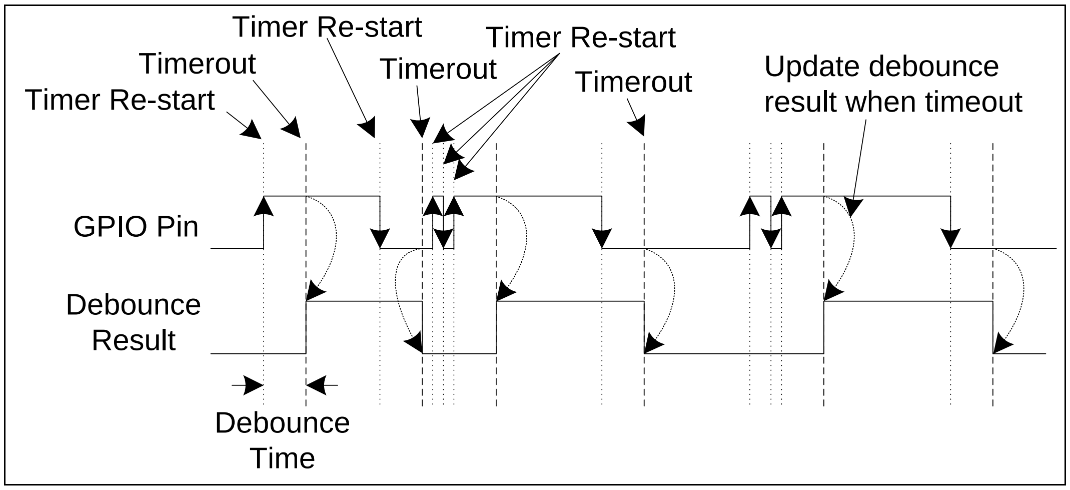
The GPIO debounce function cannot filter noise present at I/O pins completely.

Workaround:

Do not use the GPIO hardware debounce function. Use software debounce to filter unwanted input noise.

One software debounce method is to work with GPIO interrupt and a timer. The GPIO interrupt could be used to detect any I/O transient. The timer could be used to check if the I/O has transient within a period of time. By detect I/O transient with a period of time, user can skip all I/O transients shorter than specified period of time to achieve I/O debounce function.

The following figure shows the behavior of the software debounce flow. The timer is re-started whenever I/O transient. If there is no I/O transient within time-out period, the GPIO pin state at time-out will be assigned to a debounce result. If there is an I/O transient within time-out period, the timer will just be re-started without updating any debounce result.



A recommended software debounce steps are as follows:

1. Enable timer clock
2. Select timer clock source as LIRC (10 kHz)

3. Reset timer counter
4. Enable timer IRQ
5. Set GPIO pin as input
6. Enable GPIO pin interrupt with rising and falling interrupt type.
7. Enable GPIO IRQ
8. Prepare the GPIO IRQ handler:
    a. Reset timer counter
    b. Set timer time-out value (Debounce time)
    c. Start timer with one shot mode
    d. Clear GPIO interrupt flag
9. Prepare the timer IRQ handler:
    a. Get the debounce result
    b. Clear timer interrupt flag

The following code is an example to debounce pin PC.8. Timer 0 is used as the debounce timer. The debounce result is assigned to a global variable *g_u32Debounce*. PC.9 is used to monitor the result of *g_u32Debounce*.

```
#define DEBOUNCE_TIME           30     /* Debounce time in 0.1ms unit */

volatile uint32_t g_u32Debounce = 0;   /* Global variable for debounce result */

void DBNCE_Init(void)
{
    /* Enable Timer Clock Source */
    CLK->APBCLK |= CLK_APBCLK_TMR0_EN_Msk;

    /* Select Clock as LIRC 10kHz */
    CLK->CLKSEL1 = (CLK->CLKSEL1 & (~CLK_CLKSEL1_TMR0_S_Msk)) | CLK_CLKSEL1_TMR0_S_LIRC;

    /* Reset Timer */
    TIMER0->TCSR = TIMER_TCSR_CRST_Msk;
    while(TIMER0->TCSR);

    /* Enable Timer IRQ */
    NVIC_EnableIRQ(TMR0_IRQn);
    /*****************************************************************
     Modify Here:
        All Debounce GPIO should be configured here.
          1. Set GPIO to be input mode
          2. Enable GPIO interrupt with rising + falling edge trigger.
          3. Enable GPIO IRQ
     *****************************************************************/

    /* Set GPIO Input */
    PC->PMD = (PC->PMD & (~(0x3<<8*2))) | (GPIO_PMD_INPUT << 8*2);

    /* Interrupt Type: Both Edge */
    PC->IMD |= (GPIO_IMD_EDGE << 8);
    PC->IEN |= ((1 << 8) << GPIO_IEN_IR_EN_Pos) | ((1 << 8) << GPIO_IEN_IF_EN_Pos);

    /* Enable GPIO IRQ */
    NVIC_EnableIRQ(GPCDF_IRQn);

}

void TMR0_IRQHandler(void)
{
    /*****************************************************************
     Modify Here:
       Debounce Ok when timer timeout.
       All GPIO debounce result should be return by global variable here.
     *****************************************************************/
     g_u32Debounce = PC8;

    /* Clear Timer Interrupt Flag */
    TIMER0->TISR = TIMER_TISR_TIF_Msk;
}

void GPCDF_IRQHandler(void)
{
    /* Reset Timer Counter */
    TIMER0->TCSR = TIMER_TCSR_CRST_Msk;
    while(TIMER0->TCSR);
    /* Debounce Time */
    TIMER0->TCMPR = DEBOUNCE_TIME - 1;

    /* Start Timer */
    TIMER0->TCSR = TIMER_TCSR_IE_Msk | TIMER_TCSR_CEN_Msk |
                   TIMER_TCSR_TDR_EN_Msk | TIMER_TCSR_CACT_Msk;
    while(TIMER0->TCSR & TIMER_TCSR_CACT_Msk);

    /*******************************************************************
     Modify Here:
       Clear relative GPIO interrupt flag here
     *******************************************************************/
    GPIO_CLR_INT_FLAG(PC, (1 << 8));
}

int main(void)
{
    /* Unlock Protected Registers */
    SYS_UnlockReg();

    /* Configure PC.9 as Output mode */
    PC->PMD = (PC->PMD & (~GPIO_PMD_PMD9_Msk)) | (GPIO_PMD_OUTPUT << GPIO_PMD_PMD9_Pos);

    DBNCE_Init();

    while(1)
    {
        PC9 = g_u32Debounce;
    }
}
```

2.9 RS485 Peripheral Clock Setting

Description:

The peripheral clock frequency of UART could be higher or lower than system clock frequency even UART works 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 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 system clock frequency is 50 MHz, the peripheral clock frequency of UART must lower or equal to 25 MHz.

2.10 RS485 Baud Rate Equation

Description:

The UART is equipped with a programmable baud rate generator to produce the serial clock for transmitter and receiver. The baud rate generator offers three programmable modes for both UART and RS-485 modes. The mode selection setting and baud rate calculation formula are listed below.

Mode	DIV_X_EN	DIV_X_ONE	Divider X	BRD	Baud Rate Equation
0	0	0	Don’t care	A	UART_CLK / [16 * (A+2)]
1	1	0	B	A	UART_CLK / [(B+1) * (A+2)] , B must >= 8
2	1	1	Don’t care	A	UART_CLK / (A+2), A must >=9

Problem:

Suppose that UART works in RS-485 mode and baud rate setting selects Mode 0 or 1. The receiver will stop if arrival data’s value is larger than or equal to 0x80.

Workaround:

User should select Mode 2 of baud rate setting if UART works in RS-485 mode.

2.11 Timer Module Reset

Description:

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

Problem:

Timer 0, 1 are running on APB1 bus and Timer 2, 3 are running on APB2 bus. If the specified Timer is at reset state, the other 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 returns 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:

User should avoid setting the IPRSTC2 register to reset Timer while Timer is running. It is recommended to set the CRST bit in the TCSR register to reset timer function if necessary.

2.12 USB Endpoint 6

Description:

All endpoints of USB device controller can be configured as IN or OUT Endpoint. If a USB event occurs on a specific endpoint, the corresponding USB event flag in the USB_INTSTS register will be set to 1. User can read the USB_EPSTS register to identify which kind of USB event occurred.

Problem:

The USB event flag of Endpoint 6 may not work properly. Its behavior correlates with the configuration of Endpoint 5.

Endpoint 5 STATE (USB_CFG5[6:5])	Endpoint 6 STATE (USB_CFG6[6:5])	Description
OUT	OUT	Work properly
OUT	IN	1. Work properly if Endpoint 6 is configured as ISO IN (USB_CFG6[6:5]=2; USB_CFG6[4]=1). Otherwise, 2. Endpoint 6 cannot recognize IN ACK event.
IN	OUT	When Endpoint 5 recognizes an IN ACK event, both Endpoint 5's and Endpoint 6's USB event interrupt flags will be set to 1.
IN	IN	1. Endpoint 6 cannot recognize IN ACK event. 2. When Endpoint 5 recognizes an IN ACK event, both Endpoint 5's and Endpoint 6's USB event interrupt flags will be set to 1.

Workaround:

Endpoint 6 can work properly if both Endpoint 5 and Endpoint 6 are configured as OUT endpoint. Otherwise, user should follow the constraints below for Endpoint 6 configuration.

1. Do not use Endpoint 6 if Endpoint 5 is configured as IN endpoint.
2. If Endpoint 5 is configured as OUT endpoint and Endpoint 6 is configured as IN endpoint, Endpoint 6 only can be used as an ISO IN endpoint.

Revision History

Revision	Date	Description
1.00	Feb. 20, 2013	Initially issued.
1.01	Apr. 17, 2013	Modified some terminologies.
1.02	Sep. 9, 2013	Added USB Endpoint 6 problem.
1.03	Dec. 1, 2014	Modified USB problem and workaround description.
1.04	May 19, 2016	Added GPIO debounce issue and software debounce method.

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.