M253 Series Errata Sheet

Errata Sheet for 32-bit NuMicro® Family

Document Information

Abstract	This errata sheet describes the functional problem known at the release date of this document.
Apply to	M253 Series.

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions. All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.
www.nuvoton.com

Table of Contents

1. OVERVIEW 3

2. FUNCTIONAL PROBLEMS 4
- 2.1 CAN FD register access error 4
- 2.2 CAN FD sends an error message while receiving a new message 5
- 2.3 EADC gets unexpected data in oversampling mode or averaging mode 6
- 2.4 EADC oversampling mode or averaging mode cannot be monitored by compare and window compare functions 7
- 2.5 TIMER One-shot mode can't run normally 8

3. REVISION HISTORY 11

1. Overview

Functional Problem	Description
CAN FD register access error.	CPU reads an incorrect value from CAN FD register while the message RAM is accessed by CAN controller. The incorrect value is 0, but the correct value should not be 0.
CAN FD sends an error message while receiving a new message.	If a sent message and a received message happens at the same time, the sent message data will be overwritten with the received message data during the CAN FD controller reading data from the CAN FD SRAM.
EADC gets unexpected data in oversampling mode or averaging mode.	Reading the EADC_DAT register during an ongoing conversion with the accumulation function enabled returns the intermediate accumulated value instead of the final result.
EADC oversampling mode or averaging mode cannot be monitored by compare and window compare functions.	Incorrect comparison results occur in compare mode when the accumulation function is enabled in EADC.
TIMER One-shot mode can’t run normally	When Timer One-shot mode is enabled, the CNTEN bit is automatically cleared to 0 when the counter reaches the CMPDAT value. However, if a write operation to the TIMERx_CTL register occurs before CNTEN has been cleared to 0, the CNTEN bit may be unintentionally reasserted. As a result, the TIMER may not operate correctly during the next counting cycle.

2. Functional Problems

2.1 CAN FD register access error

Description:

CPU reads an incorrect value from CANFD register while the message RAM under is accessed by CAN controller. The incorrect value is 0, but the correct value should not be 0.

Problem:

When the CAN FD controller receives a message from CAN bus, the CAN FD controller will read or write CAN FD SRAM during this read/write cycle. CPU will always read "0" data from CAN FD register.

Workaround:

The BSP adds a function to wrap reading the CANFD register. If the read value is 0, the function will continue to read 48 counts.

Note: This function supports M253_Series_BSP_CMSIS_V3.00.005 and later versions.

```
#define CANFD_READ_REG_TIMEOUT    48            /* CANFD read register time-out count */

uint32_t CANFD_ReadReg(__I uint32_t *pu32RegAddr)
{
    uint32_t u32ReadReg;
    uint32_t u32TimeOutCnt = CANFD_READ_REG_TIMEOUT;
    u32ReadReg = 0UL;

    do
    {
        u32ReadReg = inpw(pu32RegAddr);

        if (--u32TimeOutCnt == 0UL)
        {
            break;
        }
    } while (u32ReadReg == 0UL);

    return u32ReadReg;
}
```

2.2 CAN FD sends an error message while receiving a new message

Description:

If a sent message and a received message happens at the same time, the sent message data will be overwritten with the received message data during the CAN FD controller reading data from the CAN FD SRAM.

Problem:

The CAN FD controller will access CAN SRAM after Tx message is triggered or Rx message data is received. If the Tx message and Rx message happens at the same time, the Tx message data will be overwritten by Rx message data during CAN FD controller reading data from CAN FD SRAM. As a result, the CAN FD controller will follow incorrect data to transmit.

Workaround:

The BSP driver has been revised to monitor bus communication status while CPU requests to send a new message. The driver will write data to a message buffer while bus is in idle state to avoid the occurrence of CANFD internal message RAM buffer having read-while-write condition.

Note: This function supports M253_Series_BSP_CMSIS_V3.00.005 and later versions.

```
/* CAN FD communication state.*/
typedef enum
{
    eCANFD_SYNC         = 0,
    eCANFD_IDLE         = 1,
    eCANFD_RECEIVER     = 2,
    eCANFD_TRANSMITTER  = 3
} E_CANFD_COMMUNICATION_STATE;

/* Get Monitors the Module’s CAN Communication State Flag */
#define CANFD_GET_COMMUNICATION_STATE(canfd)    (((canfd)->PSR  & CANFD_PSR_ACT_Msk) >>
CANFD_PSR_ACT_Pos)
```

The Tx trigger timing should be controlled when the CAN bus is in idle state.

```
while(1)
{
     __disable_irq();
     if(CANFD_GET_COMMUNICATION_STATE (CANFD0) == eCANFD_IDLE)
     {
          CANFD_TransmitTxMsg(CANFD0, 0, psTxMsg);
     }
     __enable_irq();
}
```

Note1 : To avoid long interrupt delay, the CANFD interrupt priority should be set as the highest and CANFD_TransmitTxMsg() is executed during disable interrupt (__disable_irq()).

Note2 : This function supports M253_Series_BSP_CMSIS_V3.00.005 and later versions.

2.3 EADC gets unexpected data in oversampling mode or averaging mode

Description:

Reading the EADC_DAT register during an ongoing conversion with the accumulation function enabled returns the intermediate accumulated value instead of the final result.

Problem:

With the accumulation function enabled, the EADC accumulates multiple samples before producing a final conversion result. If EADC_DAT is read during this accumulation process, the value obtained reflects partial data rather than the data for a complete conversion. This may cause incorrect data interpretation if the result is used prematurely.

Workaround:

Use PDMA to retrieve conversion results in applications that require frequent or real-time data access while operating in oversampling or averaging mode.

1. Configure PDMA.
2. Configure EADC in accumulation function.
3. Enable the EADC's PDMA function.
4. Trigger EADC.

```
/* Configure PDMA peripheral mode form EADC to memory */
PDMA_Init();

/* Set the EADC and enable the A/D converter */
EADC_Open(EADC, 0);

/* Configure the sample module and trigger source */
EADC_ConfigSampleModule(EADC, u32ModuleNum, EADC_SOFTWARE_TRIGGER, u32ChannelNum);

/* Enable Accumulate feature */
EADC_ENABLE_ACU(EADC, u32ModuleNum, EADC_MCTL1_ACU_8);

/* Enable Average feature */
EADC_ENABLE_AVG(EADC, u32ModuleNum);

/* Enable EADC's PDMA */
EADC_ENABLE_PDMA(EADC, u32ModuleNum);

/* Trigger sample module to start A/D conversion */
EADC_START_CONV(EADC, u32ModuleMask);
```

2.4 EADC oversampling mode or averaging mode cannot be monitored by compare and window compare functions

Description:

Incorrect comparison results occur in compare mode when the accumulation function is enabled in EADC.

Problem:

While the comparison of accumulated data by compare mode is completed before conversion, the data is still in a transient state, resulting in incorrect comparison results.

Workaround:

To solve this issue, a software-based compare function can be implemented in place of the hardware compare mode.

1. Get completed EADC conversions. (ADIF set and result valid, or transferred to SRAM via PDMA)
2. Compare obtained data against a threshold.

```
/* Transferred to SRAM via PDMA */
i32ConversionData = g_ai16ConversionData[0];

/* Compare target */
i32Target = 0x600;

if (i32ConversionData >= i32Target)
{
    printf("The conversion result is >= 0x%03X\n", i32Target);
}
else
{
    printf("The conversion result is < 0x%03X\n", i32Target);
}
```

2.5 TIMER One-shot mode can’t run normally

Description:

When Timer One-shot mode is enabled, the CNTEN bit is automatically cleared to 0 when the counter reaches the CMPDAT value. However, if a write operation to the TIMERx_CTL register occurs before CNTEN has been cleared to 0, the CNTEN bit may be unintentionally reasserted. As a result, the TIMER may not operate correctly during the next counting cycle.

Workaround:

To avoid this issue, software must confirm that the current counting cycle has fully completed, including the transition of CNTEN from 1 to 0, before starting the next cycle. In other words, when the TIMER is configured in One-shot mode, software must verify that the CNTEN bit is 0 before setting it to 1 again.

In addition, as shown in Table 1, CNTEN is located at TIMERx_CTL[30]. Therefore, any write operation to the TIMERx_CTL register may also affect the CNTEN setting. For example, when updating the INTEN bit at TIMERx_CTL[29], it also sets the CNTEN bit value in the same write operation.

Timer Control Register (TIMERx_CTL)

Register	Offset	R/W	Description	Reset Value
TIMER0_CTL	TMR01_BA+0x00	R/W	Timer0 Control Register	0x0000_0005
TIMER1_CTL	TMR01_BA+0x100	R/W	Timer1 Control Register	0x0000_0005
TIMER2_CTL	TMR23_BA+0x00	R/W	Timer2 Control Register	0x0000_0005
TIMER3_CTL	TMR23_BA+0x100	R/W	Timer3 Control Register	0x0000_0005

31	30	29	28	27	26	25	24
ICEDEBUG	CNTEN	INTEN	OPMODE		Reserved	ACTSTS	EXTCNTEN
23	22	21	20	19	18	17	16
WKEN	CAPSRC	TGLPINSEL	PERIOSEL	INTRGEN	Reserved		
15	14	13	12	11	10	9	8
FUNCSEL	Reserved						
7	6	5	4	3	2	1	0
PSC							

Table 1 TIMERx_CTL Register

The sample code below demonstrates how to ensure that the CNTEN bit is successfully re-enabled in One-shot mode.

```
volatile uint32_t g_u32TMRINTCount;

void TMRx_IRQHandler(void)
{
```

```
    if (TIMER_GetIntFlag(TIMERx))
    {
        /* Clear TIMERx time-out interrupt flag */
        TIMER_ClearIntFlag(TIMERx);

        /* Disable TIMERx interrupt */
        TIMER_DisableInt(TIMERx);
        g_u32TMRINTCount++;
    }
}

void main(void)
{
    uint32_t u32PrevTMRINTCount;

    g_u32TMRINTCount = u32PrevTMRINTCount = 0;

    /* Open TIMERx in the One-shot Mode with 1000 Hz */
    TIMER_Open(TIMERx, TIMER_ONESHOT_MODE, 1000);

    /* Enable TIMERx interrupt */
    TIMER_EnableInt(TIMERx);
    NVIC_EnableIRQ(TMRx_IRQn);

    /* Start TIMERx */
    TIMER_Start(TIMERx);
    while (1)
    {
        if (g_u32TMRINTCount != u32PrevTMRINTCount)
        {
            u32PrevTMRINTCount = g_u32TMRINTCount;
            if ((TIMERx->CTL & TIMER_CTL_CNTEN_Msk) == 0)
            {
                TIMER_EnableInt(TIMERx);
                TIMER_Start(TIMERx);
            }
        }
    }
}
```

In the sample code above, the CNTEN bit is enabled in the TIMER_Start() function, while the INTEN bit is enabled and disabled in the TIMER_EnableInt() and TIMER_DisableInt() functions, respectively.

When the TIMER is configured in One-shot mode, software must verify that the CNTEN bit has been cleared to 0 before performing any write operation to the TIMERx_CTL register. By following this procedure, the current counting cycle has fully completed and allows the TIMER

to operate normally in the next counting cycle.

3. Revision History

Date	Revision	Description
2022.08.27	1.00	• Initial version.
2025.06.27	1.01	• Added issue of EADC accumulation function returns incomplete data during conversion. • Added issue of EADC compare mode behavior issue when using accumulation function.
2026.07.22	1.02	• Added Timer One-shot mode issue.

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.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.