

## AT32F413 device limitations

## Device identification

This errata sheet applies to ARTERY AT32F413 microcontrollers that feature an ARM™ 32-bit Cortex®-M4 core.

Table 1. Device summary

Device	Flash memory	Part number
AT32F413	256 KB	AT32F413RCT7, AT32F413CCT7, AT32F413CCU7, AT32F413KCU7
	128 KB	AT32F413RBT7, AT32F413CBT7, AT32F413CBU7, AT32F413KBU7
	64 KB	AT32F413C8T7

## Contents

<b>1</b>	<b>AT32F413 device limitations .....</b>	<b>5</b>
1.1	GPIO .....	6
1.1.1	FT (5V tolerant pin) maintains at intermediate level in floating input mode.....	6
1.2	ADC.....	6
1.2.1	ADC regular group conversion error due to preempted group configuration change .....	6
1.2.2	Unable to clear and set ADC preempted group end of conversion flag .....	6
1.3	CAN.....	7
1.3.1	Bit stuffing error causes the next data out of order during CAN communication ....	7
1.3.2	Unable to filter RTR of standard frame in 32-bit identifier mask mode .....	10
1.3.3	CAN sends unexpected messages in case of narrow pulse disturbance on BS2.....	11
1.4	CRM .....	11
1.4.1	CLKOUT clock output exception after entering DeepSleep mode .....	11
1.5	I2S.....	12
1.5.1	Uable to resume communication when I2S CK line is interfered.....	12
1.5.2	I2S Philips protocol Start Frame data error under certain conditions .....	12
1.5.3	First data received is misaligned in I2S PCM standard long frame receive-only mode.....	12
1.5.4	UDR flag is set in I2S slave transmission mode and discontinuous communication state .....	12
1.5.5	Data reception error when I2S 24-bit data is packed into 32-bit format .....	13
1.6	PWC.....	13
1.6.1	PVM event generation after PVM enable when VDD is above PVM threshold.....	13
1.6.2	Unable to wakeup DeepSleep mode after AHB frequency division .....	13
1.6.3	Systick interrupt wakes up DeepSleep mode .....	13
1.6.4	Waking up DeepSleep mode while DeepSleep mode is being entered causes instruction operation exception.....	14
1.6.5	SWEF flag is set when enabling a standby-mode wakeup pin.....	14
1.7	SPI .....	15
1.7.1	Uable to clear data reception DMA transfer request by reading DT register.....	15
1.7.2	CS falling edge not synchronized in slave SPI hardware CS mode .....	15
1.8	TMR .....	15
1.8.1	Suspend mode failed in external clock mode B .....	15

1.8.2	How to clear TMR-triggered DAM requests .....	15
1.8.3	TMR overrun in encoder mode counter .....	16
1.8.4	TMR1/TMR8 accessing 0x4C address using DMA causes DMA request error .....	16
1.9	USART .....	17
1.9.1	USART failed to receive data in IrDA mode .....	17
1.9.2	Clearing TDC flag immediately after USART initialization causes data transfer error .....	17
1.9.3	Clearing RDBF bit only by reading data register.....	17
1.9.4	USART can still receive data using DMA in silent mode.....	17
1.10	WWDT.....	18
1.10.1	Unable to clear RLDF flag while using WWDT interrupts .....	18
1.11	WDT .....	18
1.11.1	Entering Standby mode immediately after enabling WDT will trigger a reset .....	18
1.11.2	Entering Deepsleep mode immediately after enabling WDT would cause WDT enable failure .....	18
1.12	RTC.....	19
1.12.1	Actual RTC counter value becomes programmed value plus 1.....	19
1.13	Flash .....	19
1.13.1	Erase NZW during code execution causes program exception.....	19
1.13.2	CPU read Flash cause program exception during SPIM erase .....	19
1.14	USB.....	20
1.14.1	USB exception when connecting to HUB with broadcast feature .....	20
1.14.2	USB IN transfer error at some endpoint numbers.....	20
1.15	I2C.....	20
1.15.1	I2C slave communication failed when APB equals or less than 4MHz.....	20
1.15.2	BUSERR is detected by I2C before start of communication.....	20
<b>2</b>	<b>Document revision history .....</b>	<b>21</b>

## List of tables

Table 1. Device summary .....	1
Table 2. Summary of device limitations .....	5
Table 3. Document revision history.....	21

## 1 AT32F413 device limitations

Table 2 gives a list of limitations that have been identified so far on the AT32F413 devices.

**Table 2. Summary of device limitations**

Sections	Description
1.1 GPIO	1.1.1 FT (5V tolerant pin) maintains at intermediate level in floating input mode
1.2 ADC	1.2.1 ADC regular group conversion error due to preempted group configuration change
	1.2.2 Unable to clear and set ADC preempted group end of conversion flag
1.3 CAN	1.3.1 Bit stuffing error causes the next data out of order during CAN communication
	1.3.2 Unable to filter RTR of standard frame in 32-bit identifier mask mode
	1.3.3 CAN sends unexpected messages in case of narrow pulse disturbance on BS2
1.4 CRM	1.4.1 CLKOUT clock output exception after entering Deepsleep mode
1.5 I2S	1.5.1 Uable to resume communication when I2S CK line is interfered
	1.5.2 I2S Philips protocol Start Frame data error under certain conditions
	1.5.3 First data received is misaligned in I2S PCM standard long frame receive-only mode
	1.5.4 UDR flag is set in I2S slave transmission mode and discontinuous communication state
	15.5 Data reception error when I2S 24-bit data is packed into 32-bit format
1.6 PWC	1.6.1 PVM event generation after PVM enable when VDD is above PVM threshold
	1.6.2 Unable to wakeup Deepsleep mode after AHB frequency division
	1.6.3 Systick interrupt wakes up Deepsleep mode
	1.6.4 Waking up Deepsleep mode while Deepsleep mode is being entered causes instruction operation exception
	1.6.5 SWEF flag is set when enabling a standby-mode wakeup pin
1.7 SPI	1.7.1 Uable to clear data reception DMA transfer request by reading DT register
	1.7.2 CS falling edge not synchronized in slave SPI hardware CS mode
1.8 TMR	1.8.1 Suspend mode failed in external clock mode B
	1.8.2 How to clear TMR-triggered DAM requests
	1.8.3 TMR overrun in encoder mode counter
	1.8.4 TMR1/TMR8 accessing 0x4C address using DMA causes DMA request error
1.9 USART	1.9.1 USART failed to receive data in IrDA mode
	1.9.2 Clearing TDC flag immediately after USART initialization causes data transfer error
	1.9.3 Clearing RDBF bit only by reading data register
	1.9.4 USART can still receive data using DMA in silent mode
1.10 WWDT	1.10.1 Unable to clear RLDF flag while using WWDT interrupts
1.11 WDT	1.11.1 Entering Standby mode immediately after enabling WDT will trigger a reset
	1.11.2 Entering Deepsleep mode immediately after enabling WDT would cause WDT enable failure
1.12 RTC	1.12.1 Actual RTC counter value becomes programmed value plus 1
1.13 Flash	1.13.1 Erase NZW during code execution causes program exception
	1.13.2 CPU read Flash cause program exception during SPIM erase
1.14 USB	1.14.1 USB exception when connecting to HUB with broadcast feature
	1.14.2 USB IN transfer error at some endpoint numbers
1.15 I2C	1.15.1 I2C slave communication failed when APB equals or less than 4MHz
	1.15.2 BUSERR is detected by I2C before start of communication

## 1.1 GPIO

### 1.1.1 FT (5V tolerant pin) maintains at intermediate level in floating input mode

- Description:  
The 5V tolerant pin still has a pull-up capability of less than 10  $\mu$ A in floating input mode, causing it to maintain about 2.0 V.
- Workaround:  
Add an external pull-down resistor (200 k $\Omega$  or below)

## 1.2 ADC

### 1.2.1 ADC regular group conversion error due to preempted group configuration change

- Description:  
In ADC sequential and repetition conversion mode of regular group, attempting to change channel configuration of preempted group during a regular group conversion will cause regular group conversion data out of order.  
  
For example, when the regular group is converting the channels 1, 2, 3 and 4, attempting to change preempted group configuration during channel-2 conversion will cause the channel 2 to be converted twice, and thus the final conversion sequence of regular group becomes 1, 2, 2, 3, and 4.
- Workaround:  
When using multi-channel for regular and preempted conversion simultaneously, do not try to change its channel configuration after preempted group is already configured.

### 1.2.2 Unable to clear and set ADC preempted group end of conversion flag

- Description:  
When “PCCE” (preempted channel end of conversion flag) and “CCE” (regular channel end of conversion event) take place simultaneously, there is a problem of being unable to clear PCCE flag promptly and thus to set the next PCCE flag.
- Workaround:  
Execute PCCE flag clear command again at the place where the original PCCE flag clear command is. In other words, add one more PCCE flag clear command at its original location.

Example:

```
/* Before change */
adc_flag_clear(ADC1, ADC_PCCE_FLAG);
/* After change */
adc_flag_clear(ADC1, ADC_PCCE_FLAG);
adc_flag_clear(ADC1, ADC_PCCE_FLAG);
```

- Revision plan:  
None.

## 1.3 CAN

### 1.3.1 Bit stuffing error causes the next data out of order during CAN communication

- Description:

If a bit stuffing error occurs in the data filed during CAN communication, it will stop receiving the current data frame as expected and report an error to the bus, but the next data frame will be out of order.

- Workaround:

**Method 1:**

Enable the CAN error interrupt (its priority must be set very high) corresponding to the interrupt number in the CAN error type record. Once a bit stuffing error is detected, reset CAN (only need reset CAN registers and relevant GPIOs, without resetting NVIC), and re-initialize CAN in the CAN error interrupt functions.

This method applies to the scenario where a quick CAN initialization is required in order to ensure a quick resume of CAN communication, avoiding too much CAN data loss.

Take a CAN1 as an example, its typical code as follows:

```
/* Enable CAN error interrupt corresponding to the last CAN error interrupt number and give very high
priority */
nvic_irq_enable(CAN1_SE_IRQn, 0x00, 0x00);
can_interrupt_enable(CAN1, CAN_ETRIEN_INT, TRUE);
can_interrupt_enable(CAN1, CAN_EOIEN_INT, TRUE);
/* Interrupt service functions */
void CAN1_SE_IRQHandler(void)
{
    __IO uint32_t err_index = 0;
    if(can_flag_get(CAN1,CAN_ETR_FLAG) != RESET)
    {
        err_index = CAN1->ests & 0x70;
        can_flag_clear(CAN1, CAN_ETR_FLAG);
        if(err_index == 0x00000010)
        {
            can_reset(CAN1);
            /* Call CAN initialization function */
        }
    }
}
```

**Notes:**

- a) CAN error interrupts should be given as very high priority;
- b) As it takes some time to finish CAN initialization, CAN's inability to resume communication immediately when an error occurs may cause loss of data.

**Method 2:**

Enable the CAN error interrupt (its priority must be set very high) corresponding to the interrupt number in the CAN error type record. Once a bit stuffing error is detected, reset CAN (only need reset CAN registers and relevant GPIOs, without resetting NVIC), record the reset event, and re-initialize CAN in other low-priority interrupts or main functions.

This method applies to the scenario where the CAN communication is unable to resume in time, but the CAN re-initialization must be performed in order not to affect the operations of other application logic.

Take a CAN1 as an example, its typical code as follows:

```
/*Enable CAN error interrupt corresponding to the last CAN error interrupt number and give very high
priority*/
nvic_irq_enable(CAN1_SE_IRQn, 0x00, 0x00);
can_interrupt_enable(CAN1, CAN_ETRIEN_INT, TRUE);
can_interrupt_enable(CAN1, CAN_EOIEN_INT, TRUE);
/* Interrupt service functions*/
__IO uint32_t can_reset_index = 0;
void CAN1_SE_IRQHandler(void)
{
    __IO uint32_t err_index = 0;
    if(can_flag_get(CAN1,CAN_ETR_FLAG) != RESET)
    {
        err_index = CAN1->ests & 0x70;
        can_flag_clear(CAN1, CAN_ETR_FLAG);
        if(err_index == 0x00000010)
        {
            can_reset(CAN1);
            can_reset_index = 1;
        }
    }
}
```

Then the application polls whether “can\_reset\_index” is set or not at the desired place (in main functions, say). Call the CAN initialization function, if available.

**Notes:**

- a) CAN error interrupts should be given as very high priority;
- b) As it takes some time to finish CAN initialization, CAN’s inability to resume communication immediately when an error occurs may cause loss of data.

**Method 3:**

Enable the CAN error interrupt (its priority must be set very high) corresponding to the interrupt number in the CAN error type record. Once a bit stuffing error is detected, forcibly send an invalid message with a very-high-priority identifier.



This method applies to the scenario where one doesn't want to spend time on CAN reset, all message identifiers on CAN bus are known, and each CAN node receives messages in accordance with the identifier filtering conditions.

Take a CAN1 as an example, its typical code as follows:

```
/*Forcibly send a frame of invalid message with a very-high-priority identifier*/
static void can_transmit_data(void)
{
    uint8_t transmit_mailbox;
    can_tx_message_type tx_message_struct;
    tx_message_struct.standard_id = 0x0;
    tx_message_struct.extended_id = 0x0;
    tx_message_struct.id_type = CAN_ID_STANDARD;
    tx_message_struct.frame_type = CAN_TFT_DATA;
    tx_message_struct.dlc = 8;
    tx_message_struct.data[0] = 0x00;
    tx_message_struct.data[1] = 0x00;
    tx_message_struct.data[2] = 0x00;
    tx_message_struct.data[3] = 0x00;
    tx_message_struct.data[4] = 0x00;
    tx_message_struct.data[5] = 0x00;
    tx_message_struct.data[6] = 0x00;
    tx_message_struct.data[7] = 0x00;
    can_message_transmit(CAN1, &tx_message_struct);
}
/* Enable CAN error interrupt corresponding to the last CAN error interrupt number and give very high
priority */
nvic_irq_enable(CAN1_SE_IRQn, 0x00, 0x00);
can_interrupt_enable(CAN1, CAN_ETRIEN_INT, TRUE);
can_interrupt_enable(CAN1, CAN_EOIEN_INT, TRUE);
/* Interrupt service functions*/
void CAN1_SE_IRQHandler(void)
{
    __IO uint32_t err_index = 0;
    if(can_flag_get(CAN1, CAN_ETR_FLAG) != RESET)
    {
        err_index = CAN1->ests & 0x70;
        can_flag_clear(CAN1, CAN_ETR_FLAG);
        if(err_index == 0x00000010)
        {
```

```

        can_transmit_data;
    }
}
}

```

**Notes:**

- a) CAN error interrupts should be given as very high priority;
- b) This method is only applicable to the scenario where the transmit FIFO priority is determined by message identifiers;
- c) The identifier of the invalid message in this method is changeable. But its priority must be given the highest among the CAN bus, and it cannot be received as a normal message by other nodes.

### 1.3.2 Unable to filter RTR of standard frame in 32-bit identifier mask mode

- Description:

When the CAN filter mode is configured in 32-bit identifier mask mode, the RTR bit (remote frame identifier) cannot be filtered effectively during a standard frame filtering.

When the following conditions are met, follow the “Workaround” to solve this problem:

1. 32-bit wide identifier mask mode is used
2. A standard frame is being filtered but the remote frame passing through filter is unwanted

- Workaround:

Method 1: By software. When filtering a standard frame in 32-bit wide identifier mask mode, the software is used to get the status of the RTR bit (remote frame identifier) and determine whether this frame of message is needed or not by the application. For example:

```

void CAN1_RX0_IRQHandler(void)
{
    can_rx_message_type rx_message_struct;
    if(can_flag_get(CAN1,CAN_RF0MN_FLAG) != RESET)
    {
        can_message_receive(CAN1, CAN_RX_FIFO0, &rx_message_struct);
        /* only store the data frame,discard the remote frame */
        if((rx_message_struct.id_type == CAN_ID_STANDARD) && (rx_message_struct.frame_type ==
CAN_TFT_DATA))
        {
            /* user store the receive data */
        }
    }
}

```

Method 2: Use other filtering mode according to the needs, such as, 32-bit wide identifier list mode, 16-bit wide identifier mask mode or 16-bit wide identifier list mode.

### 1.3.3 CAN sends unexpected messages in case of narrow pulse disturbance on BS2

- Description:

In case of a large amount of narrow pulses (pulse width less than 1tp) on CAN bus, the CAN nodes are likely to send unexpected messages, for instance, a data frame is sent as a remote frame, a standard frame as an extended one, or data phase error occurs.

- Workaround:

Configure synchronization width RSAW = BTS2 segment width in order to avoid unexpected errors.

It should be noted that after RSAW =BTS2 is asserted, the CAN bus communication speed is reduced when there is a lot of disturbance on CAN bus.

```
static void can_configuration(void)
{
    ...

    /* can baudrate, set baudrate = pclk/(baudrate_div *(3 + bts1_size + bts2_size)) */
    can_baudrate_struct.baudrate_div = 12;
    can_baudrate_struct.rsaw_size = CAN_RSAW_3TQ;
    can_baudrate_struct.bts1_size = CAN_BTS1_8TQ;
    can_baudrate_struct.bts2_size = CAN_BTS2_3TQ;

    ...
}
```

## 1.4 CRM

### 1.4.1 CLKOUT clock output exception after entering Deepsleep mode

- Description:

In case of DEEPSLEEP\_DEBUG=0 and CLKOUT configured as system clock output, there would still be clock output (with LICK clock frequency) on the CLKOUT pin after entering Deepsleep mode.

- Workaround:

Configure CLKOUT as NOCLK before entering Deepsleep mode, and then configure it as system clock output after leaving Deepsleep mode.

## 1.5 I2S

### 1.5.1 Unable to resume communication when I2S CK line is interfered

- Description:  
The I2S CK and WS signals are not synchronized, so that when the clock line is interfered during communication, this noise/interference would be treated as a CK signal by I2S, causing communication not to resume automatically.
- Workaround:  
Pull up or pull down the WS and CK pins internally or externally, depending on the desired audio protocols and I2SCLKPOL configuration. When communication error is detected, it is possible to disable and enable I2S to resume communication.

### 1.5.2 I2S Philips protocol Start Frame data error under certain conditions

- Description:  
In case of I2S Philips protocol, master receive and slave transmission and I2SCLKPOL high level, the WS signal falling edge corresponding to the left channel of the first data frame would not be output effectively, causing some devices unable to receive data from the left channel.
- Workaround:  
Pull up or pull down the WS and SCK pins internally or externally, depending on the desired audio protocols and I2SCLKPOL configuration.

### 1.5.3 First data received is misaligned in I2S PCM standard long frame receive-only mode

- Description:  
When PCLK frequency division factor is greater than 1, and I2S PCM standard long frame receive-only mode is enabled, if I2SCPOL = 0 is set and the SCK line remains high before enabling I2S, the first received data would be out of order.
- Workaround:  
Pull up or pull down the SCK pin externally or internally, depending on the I2SCLKPOL configuration.

### 1.5.4 UDR flag is set in I2S slave transmission mode and discontinuous communication state

- Description:  
The UDR flag is set incorrectly in I2S slave transmission mode and in discontinuous communication state, even if data have been written before the start of communication.
- Workaround:  
For continuous communication, it is recommended to use DMA or interrupts for fast data transfer in I2S slave transmission mode according to the protocols.

### 1.5.5 Data reception error when I2S 24-bit data is packed into 32-bit format

- Description:  
When I2S 24-bit data is packed into 32-bit frame format, the remaining 8 invalid CLK data would be received by the receiver as normal data.
- Workaround:  
Method 1: Both the receiver and transmitter use the same way of packing 24-bit data into 32-bit format.  
Method 2: Discard these 8 invalid CLK data in this frame format using software.

## 1.6 PWC

### 1.6.1 PVM event generation after PVM enable when VDD is above PVM threshold

- Description:  
When the VDD is greater than PVM threshold, an unwanted PVM event is generated once as soon as PWC voltage monitoring is enabled.
- Workaround:  
Clear the unwanted PVM event during PVM initialization.

### 1.6.2 Unable to wakeup Deepsleep mode after AHB frequency division

- Description:  
If AHB frequency division is configured, no wakeup source can wake up Deepsleep mode.
- Workaround:  
Do not divide AHB frequency in Deepsleep mode.  
Remove AHB frequency division before entering Deepsleep mode. Configure then the desired AHB frequency after wakeup.

### 1.6.3 Systick interrupt wakes up Deepsleep mode

- Description:  
If Systick or Systick interrupt is not disabled before the Deepsleep mode is entered, the Systick then would keep running after Deepsleep mode entry, and the subsequent Systick interrupt would wake up Deepsleep mode.
- Workaround:  
Disable Systick or Systick interrupts before entering Deepsleep mode.

### 1.6.4 Waking up Deepsleep mode while Deepsleep mode is being entered causes instruction operation exception

- Description:

When a Deepsleep wakeup source arrives at the moment while the Deepsleep mode is being entered (around 3 LICK clock cycles), this behavior may cause some instructions to be missed or not performed after waking up Deepsleep mode.

- Workaround:

After waking up Deepsleep mode, wait around 3 LICK clock cycles before performing instructions (Refer to FAQ0114 for details).

### 1.6.5 SWEF flag is set when enabling a standby-mode wakeup pin

- Description:

If a wakeup pin (waking up Standby mode) were used as a GPIO push-pull output (high) or pull-up input before being enabled, a SWEF flag would be set exceptionally once the pin is enabled.

- Workaround:

If the wakeup pin (waking up Standby mode) was used as a GPIO before, then the IO has to be re-initialized to pull-down input or analog input mode before enabling the pin. For example:

```
gpio_init_type gpio_init_struct;

/* enable the button clock */
crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);

/* set default parameter */
gpio_default_para_init(&gpio_init_struct);

/* configure wakeup pin as input with pull-down */
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_mode = GPIO_MODE_INPUT;
gpio_init_struct.gpio_pins = USER_BUTTON_PIN;
gpio_init_struct.gpio_pull = GPIO_PULL_DOWN;
gpio_init(GPIOA, &gpio_init_struct);

/* enable wakeup pin - pa0 */
pwc_wakeup_pin_enable(PWC_WAKEUP_PIN_1, TRUE);
```

## 1.7 SPI

### 1.7.1 Unable to clear data reception DMA transfer request by reading DT register

- Description:  
For example, for those applications using SPI full-duplex function for time-sharing receive and transmit, the invalid data reception DMA transfer request, which is set during SPI transmission, cannot be cleared by reading DT register.
- Workaround:  
When SPI reception DMA channel is turned off, disable SPI instead of reading DT register, and then enable SPI at a place where you want to start communication.

### 1.7.2 CS falling edge not synchronized in slave SPI hardware CS mode

- Description:  
In SPI slave hardware CS mode, the initial CLK synchronization for data transfer is not performed at each CS falling edge.
- Workaround:  
Solution A: Strictly control the slave CS line, pull high the CS line as soon as the communication is complete.  
Solution B: Enable CRC check. Once a CRC error is detected, reset SPI and restart handshake communication.

## 1.8 TMR

### 1.8.1 Suspend mode failed in external clock mode B

- Description:  
Suspend mode does not work in external clock mode B, that is, the timer still keeps running, regardless of if the Suspend mode is set high or low.
- Workaround:  
None.

### 1.8.2 How to clear TMR-triggered DAM requests

- Description:  
TMR-induced DMA request cannot be cleared by resetting/setting the corresponding DMA request enable bit in the TMRx\_IDEN register.
- Workaround:  
Before enabling DMA channel transfer, reset TMR (reset CRM clock of TMR) and initialize TMR to clear pending DMA requests.

### 1.8.3 TMR overrun in encoder mode counter

- Description:  
In encoder counting mode, if the counter counts back and forth between 0 and PR, the OVIF is not set at an overrun or underrun event.
- Method 1:  
Configure the C3IF and C4IF channels of the TMR (where an encoder is being used) as output mode, C3DT = AR, C4DT = 0, and enable C3IF and C4IF interrupts.  
C3IF event & downcounting indicates an underrun;  
C4IF event & upcounting indicates an overrun;  
This method has its limitation: If the input frequency of the encoder mode counter were too fast, interrupts would occur frequently and need to be handled by software, causing not enough time to deal with interrupts. Thus this method applies to the scenario where the external input frequency of the encoder is not so fast
- Method 2:  
Turn to a TMR with enhanced mode (the counter can be extended from 16-bit to 32-bit width) in order to expand the encoder's counting range that detects forward and reverse rotation, and configure the initial value of the counter to PR/2 so as to prevent the timer from overflowing.  
This method has its limitation: The forward and reverse rotation of the encoder must be limited to a certain range. An overflow still occurs if the encoder were always rotated in one direction.  
This method applies to the scenario where the rotation of the encoder is controlled at a certain range.

### 1.8.4 TMR1/TMR8 accessing 0x4C address using DMA causes DMA request error

- Description:  
When TMR (TMR1/TMR8) issues a DMA request, the lower 8 bits of the current address bus is 0x4C, and DMA transfer doesn't change the APB bus address where the TMR is located. In this case, a single TMR DMA request will be set again after being cancelled, causing potential DMA transfer error.
- Workaround:  
User other timers than TMR1/8.



## 1.9 USART

### 1.9.1 USART failed to receive data in IrDA mode

- Description:  
When USART baud rate is configured to be less than or equal to 38400 in USART IrDA mode, the USART is unable to receive data.
- Workaround:  
The USART baud rate must not be lower than or equal to 38400.

### 1.9.2 Clearing TDC flag immediately after USART initialization causes data transfer error

- Description:  
If the TDC flag is cleared immediately after USART initialization, and wait until the TDC is set before sending data, this would cause data unable to be transferred.
- Workaround:  
Do not clear TDC flag by software. When TDC flag is set, it indicates that data transmission is complete. If the TDC is cleared, it cannot be set again before next data transmission, the TDC flag would always remain 0.

### 1.9.3 Clearing RDBF bit only by reading data register

- Description:  
In regular asynchronous communication mode, there are two ways to clear the RDBF bit (non-empty flag) of the read data register. Either clear this bit by reading USART\_DT register, or by writing 0 to RDBF bit of the status register. However, for revision B, the RDBF bit can only be cleared by reading USART\_DT register.
- Workaround:  
None.

### 1.9.4 USART can still receive data using DMA in silent mode

- Description:  
When the USART sends data to RX in silent mode (address matching wakeup mode), it still can generate a DMA reception request and the data can also be received by DT data register even though the RDBF is not set. In this case, silent mode does not work.
- Workaround:  
None.

## 1.10 WWDT

### 1.10.1 Unable to clear RLDF flag while using WWDT interrupts

- Description:

While using WWDT interrupts, it is impossible to clear RLDF flag when CNT=0x40 is reached in the interrupt service routine. Thus after entering an interrupt, it is necessary to feed the watchdog first before clearing the RLDF flag.

- Workaround:

For WWDT interrupt handler, first feed the watchdog before clearing RLDF flag.

```
void WWDT_IRQHandler(void)
{
    wwdt_counter_set(127);
    wwdt_flag_clear();
}
```

## 1.11 WDT

### 1.11.1 Entering Standby mode immediately after enabling WDT will trigger a reset

- Description:

Entering Standby mode immediately after enabling WDT (WDT\_CMD = 0xCCCC) will trigger an immediate reset.

- Workaround:

Insert a delay of a few  $\mu$ s after entering WDT, and then enter Standby mode.

### 1.11.2 Entering Deepsleep mode immediately after enabling WDT would cause WDT enable failure

- Description:

Entering Deepsleep mode immediately after enabling WDT (WDT\_CMD = 0xCCCC) would cause WDT not to be enabled successfully.

- Workaround:

Insert around 30  $\mu$ s delay after enabling WDT, and then enter Deepsleep mode.

## 1.12 RTC

### 1.12.1 Actual RTC counter value becomes programmed value plus 1

- Description:  
After configuring RTC counter value, the actual value becomes the programmed value plus 1.
- Workaround:  
First program a frequency division value before programming RTC counter value.

```
rtc_wait_config_finish();  
rtc_divider_set(32767);  
  
rtc_wait_config_finish();  
rtc_counter_set(100);
```

## 1.13 Flash

### 1.13.1 Erase NZW during code execution causes program exception

- Description:  
The erase operation in the zero-wait (ZW) area does not affect program running. However, if the program contains instructions from both zero-wait (ZW) and non-zero-wait (NZW) area, the program exception may occur because of reading data in NZW area.  
For instance, an interrupt can be handled during zero-wait area erase period, but if the interrupt handler functions involve both ZW and NZW areas, the program exception may occur.
- Workaround:  
The principle here is that all codes must be located in the same area (either all in zero-wait area or in non-zero-wait area) during zero-wait area erase phase.  
To make this happen, disable interrupt enable bits first before starting erase, and then enable interrupt enable bits after the completion of erase. And the codes related to erase functions must be compiled into the same area.

### 1.13.2 CPU read Flash cause program exception during SPIM erase

- Description:  
If CPU reads Flash memory during SPIM erase, the read Flash command would be considered as read SPIM mistakenly, causing data error and program exception.  
For instance, SPIM erase functions are compiled in NZW area, so they fetch instructions from NZW area during erase operation. In this case, reading Flash will cause program error.
- Workaround:  
The principle here is that read Flash is forbidden during SPIM erase.  
To make this happen, disable interrupt enable bits before starting erase, and then enable interrupt enable bits after the completion of erase. And the codes related to erase functions must be compiled in ZW area or RAM.

## 1.14 USB

### 1.14.1 USB exception when connecting to HUB with broadcast feature

- Description:

When the USB is connected to the HUB (broadcast data) port that is linked to other devices at the same time, there is a possibility that the USB receives data the host sends to other devices. In special conditions, such data would affect the USB and cause an enumeration error.

- Workaround:

Reduce APB1 clock to 48M or below.

### 1.14.2 USB IN transfer error at some endpoint numbers

- Description:

USB IN transfer error may happen in some addresses and endpoint numbers, including wrong IN data transfer or issuing NAK response mistakenly.

- Workaround:

Use endpoint numbers 2/6/12/14 for IN data transfer.

## 1.15 I2C

### 1.15.1 I2C slave communication failed when APB equals or less than 4MHz

- Description:

I2C is unable to communicate at 400kHz in slave mode when the APB clock is equal to or less than 4MHz.

- Workaround:

Increase the APB clock to 8 MHz, or reduce the I2C speed to 100kHz.

### 1.15.2 BUSERR is detected by I2C before start of communication

- Description:

When all the following conditions are met, BUSERR conditions would be detected by I2C, causing communication error.

Condition 1: I2C is enabled

Condition 2: Before the start of communication

Condition 3: BUSERR timing takes place on the bus

- Workaround:

Check if the BUSERR flag is set or not before the start of communication. If it is set, just need clear this flag to enable communication. Optionally, enable error interrupt, and clear it in the interrupt after the BUSERR flag is set.

## 2 Document revision history

**Table 3. Document revision history**

Date	Revision	Changes
2021.11.22	2.0.0	Initial release
2022.03.01	2.0.1	Added <i>Unable to filter RTR of standard frame in 32-bit identifier mask mode.</i>
2022.3.30	2.0.2	Added <i>Actual RTC counter value becomes programmed value plus 1</i>
2022.04.15	2.0.3	<ol style="list-style-type: none"> <li>1. Added <i>CAN sends unexpected messages in case of narrow pulse disturbance on BS2</i></li> <li>2. Added <i>Flash-related errata description.</i></li> <li>3. Added <i>USB-related errata description.</i></li> <li>4. Added <i>First data received is misaligned in I2S PCM standard long frame receive-only mode</i></li> <li>5. Added <i>Data reception error when I2S 24-bit data is packed into 32-bit format.</i></li> <li>6. Added <i>CS failing edge not synchronized in slave SPI hardware CS mode.</i></li> </ol>
2022.04.28	2.0.4	<ol style="list-style-type: none"> <li>1. Added an example case in the <i>1.3.2 Unable to filter RTR of standard frame in 32-bit identifier mask mode</i></li> <li>2. Added an example case in the <i>1.6.5 SWEF flag is set when enabling a standby-mode wakeup pin</i></li> <li>3. Added the section <i>1.1.1 FT (5V tolerant pin) maintains at intermediate level in floating input mode</i></li> <li>4. Added the section <i>1.15.1 I2C slave communication failed when APB equals or less than 4MHz</i></li> </ol>
2022.09.06	2.0.5	Added <i>1.15.2 BUSERR is detected by I2C before start of communication</i>
2022.09.27	2.0.6	Added <i>1.2.2 Unable to clear and set ADC preempted group end of conversion flag</i>

**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

Purchasers are solely responsible for the selection and use of ARTERY's products and services, and ARTERY assumes no liability whatsoever relating to the choice, selection or use of the ARTERY products and services described herein.

No license, express or implied, to any intellectual property rights is granted under this document. If any part of this document deals with any third party products or services, it shall not be deemed a license grant by ARTERY for the use of such third party products or services, or any intellectual property contained therein, or considered as a warranty regarding the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

Unless otherwise specified in ARTERY's terms and conditions of sale, ARTERY provides no warranties, express or implied, regarding the use and/or sale of ARTERY products, including but not limited to any implied warranties of merchantability, fitness for a particular purpose (and their equivalents under the laws of any jurisdiction), or infringement of any patent, copyright or other intellectual property right.

Purchasers hereby agrees that ARTERY's products are not designed or authorized for use in: (A) any application with special requirements of safety such as life support and active implantable device, or system with functional safety requirements; (B) any air craft application; (C) any automotive application or environment; (D) any space application or environment, and/or (E) any weapon application. Purchasers' unauthorized use of them in the aforementioned applications, even if with a written notice, is solely at purchasers' risk, and is solely responsible for meeting all legal and regulatory requirement in such use.

Resale of ARTERY products with provisions different from the statements and/or technical features stated in this document shall immediately void any warranty grant by ARTERY for ARTERY products or services described herein and shall not create or expand in any manner whatsoever, any liability of ARTERY.

© 2022 Artery Technology -All rights reserved