ES0002
Errata sheet

AT32F403A/407 device limitations

Chip identification

This errata sheet applies to the version of A & B of ARTERY AT32F403A/407 series. This series features an ARM™ 32-bit Cortex®-M4 core.

Table 2 gives a full list of part numbers. The products are identifiable in Table 1:

- By the revision code marked below the lot number on the device package

Table 1. Device identification

Part number	Revision code marked on the device
AT32F403A/407	"A"
	"B"

1. The bit [78:76] (Mask_Version) in the Flash memory size and device ID register (UID base address 0x1FFF F7E8) shows the revision code of the device. That is, the bit [6:4] in the address 0x1FFFF7F1 can be used to get the revision code, such as

   Revision A: 0b000

   Revision B: 0b001
2. Refer to *2 Revision code on device marking* for details on how to find the revision code on the different device packages.

Table 2. Device summary

Device	Flash memory	Part numbers
AT32F403A/407	1024 KB	AT32F403ACGU7, AT32F403ACGT7, AT32F403ARGT7, AT32F403AVGT7, AT32F407RGT7, AT32F407VGT7, AT32F407AVGT7
	512 KB	AT32F403ACEU7, AT32F403ACET7, AT32F403ARET7, AT32F403AVET7, AT32F407RET7, AT32F407VET7
	256 KB	AT32F403ACCU7, AT32F403ACCT7, AT32F403ARCT7, AT32F403AVCT7, AT32F407RCT7, AT32F407VCT7, AT32F407AVCT7

Contents

List of tables

List of figures

1 AT32F403A/407 limitations

Table 3 summarizes the limitations on AT32F403A/407 device that have been identified so far.

Table 3. Summary of device limitations

Sections	Description	Rev. A	Rev. B
1.1 ADC	1.1.1 Program gets stuck.	Fail	Fixed
	1.1.2 ADC conversion speed reduction in ADC regular master/slave short-shift mode and repetition mode.	Fail	Fixed
	1.1.3 Trigger ADC1 during ADC2 calibration causes converted data error in ADC regular master/slave short-shift mode.	Fail	Fixed
1.2 CAN	1.2.1 Bit stuffing error causes the next data out of order during CAN communication.	Fail	Fail
	1.2.2 Failed to filter RTR bit of standard frame in 32-bit identifier mask mode.	Fail	Fail
	1.2.3 CAN sends unexpected messages in case of narrow pulse disturbance on BS2.	Fail	Fail
	1.2.4 Fail to cancel mailbox transmit command when CAN bus disconnected	Fail	Fail
1.3 GPIO	1.3.1 TMR2 IO multiplexed function failed when using USBFS SOF as TMR2_IS1 input.	Fail	Fixed
1.4 I2C	1.4.1 1st clk sent by master may get shorter when SCL stretching is released by slave in I2C master mode.	Fail	Fixed
	1.4.2 START failed to be sent in I2C master mode.	Fail	Fixed
	1.4.3 I2C slave communication failed when APB equals or less than 4MHz	Fail	Fail
	1.4.4 BUSERR is detected by I2C before start of communication	Fail	Fail
1.5 I2S	1.5.1 Unable to resume communication automatically while I2S CK line is disturbed.	Fail	Fixed
	1.5.2 I2S Philips protocol SOF data error in certain conditions.	Fail	Fixed
	1.5.3 First data received is misaligned in I2S PCM standard long frame receive-only mode.	Fail	Fail
	1.5.4 UDR flag is set in I2S slave transmission mode and discontinuous communication state.	Fail	Fail
	1.5.5 Data reception error when I2S 24-bit data is packed into 32-bit format.	Fail	Fail
1.6 PWC	1.6.1 Unable to wake up Deepsleep mode after AHB frequency division.	Fail	Fail
	1.6.2 Systick interrupt wakes up Deepsleep mode.	Fail	Fail
	1.6.3 Waking up Deepsleep mode while Deepsleep mode is being entered causes instruction operation exception.	Fail	Fixed
	1.6.4 SWEF flag is set when a wakeup pin is enabled.	Fail	Fixed
1.7 RCC	1.7.1 CLKOUT output exception after entering Deepsleep mode.	Fail	Fail
1.8 TMR	1.8.1 TMR1/TMR8 read 0x4C address using DMA may cause DMA request error.	Fail	Fixed
	1.8.2 TMR overflow in encoder mode counter.	Fail	Fail
	1.8.3 Slave timer unable to receive reset signal from master timer	Fail	Fixed
	1.8.4 Break input failed when TMREN=0 (TMR disabled)	Fail	Fail
1.9 USART	1.9.1 Break frame is sent more than once in a specific condition.	Fail	Fixed
	1.9.2 Unable to receive data after enabling smart card mode in a specific condition.	Fail	Fixed
1.10 WWDT	1.10.1 Unable to clear RLDF flag while using WWDT interrupts	Fail	Fixed
1.11 RTC	1.11.1 Actual RTC counter value is the programmed value plus 1.	Fail	Fail
1.12 FLASH	1.12.1 Program error may occur when sLib is placed in NZW area.	Fail	Fail
	1.12.2 Erasing NZW during code execution causes program exception	Fail	Fail
	1.12.3 CPU read Flash causes program exception during SPIM erase.	Fail	Fail
1.13 USB	1.13.1 USB exception when connecting to HUB with broadcast.	Fail	Fixed
	1.13.2 USB IN transfer error at some endpoint numbers.	Fail	Fixed
1.14 SPI	1.14.1 CS failing edge was not synchronized in slave SPI hardware CS mode.	Fail	Fail
1.15 EXINT	1.15.1 Software triggers twice EXINT line interrupt responses	Fixed	Fail

1.1 ADC

1.1.1 Program gets stuck in dual ADC mode

- Description:

  In ADC master/slave mode, an ADC1 calibration starts only when the ADC2 is powered on. Enabling ADC1 and ADC1 calibration alone (without enabling ADC2) would cause program to get stuck at the command "while(adc_calibration_status_get(ADC1))".

- Workaround:

  Modify the execution sequence of ADC enable and calibration functions, as shown below:

  - Before change: original execution order for ADC enable and calibration functions

```
adc_enable(ADC1, TRUE);
adc_calibration_init(ADC1);
while(adc_calibration_init_status_get(ADC1));
adc_calibration_start(ADC1);
while(adc_calibration_status_get(ADC1));

adc_enable(ADC2, TRUE);
adc_calibration_init(ADC2);
while(adc_calibration_init_status_get(ADC2));
adc_calibration_start(ADC2);
while(adc_calibration_status_get(ADC2));…
```

  - After change: new execution order for ADC enable and calibration

```
adc_enable(ADC1, TRUE);
adc_enable(ADC2, TRUE);
adc_calibration_init(ADC1);
while(adc_calibration_init_status_get(ADC1));
adc_calibration_start(ADC1);
while(adc_calibration_status_get(ADC1));
adc_calibration_init(ADC2);
while(adc_calibration_init_status_get(ADC2));
adc_calibration_start(ADC2);
while(adc_calibration_status_get(ADC2));…
```

- Revision plan

  This issue has been fixed in revision B.

1.1.2 ADC conversion speed reduction in ADC regular master/slave short-shift mode and repetition mode

- Description:

  In ADC regular master/slave short-shift mode and repetition mode, the ADC2 sampling and conversion starts at the end of the ADC1 conversion, rather than following the rule of conversion start time + 7 ADCCLK cycles, which leads to lower conversion speed.

- Workaround:

  Turn to regular master/slave long-shift mode (14 ADCCLK cycles between two adjacent conversions)

- Revision plan:

  This issue has been fixed in revision B.

1.1.3 Trigger ADC1 during ADC2 calibration causes converted data error in ADC regular master/slave short-shift mode

- Description:

  In ADC regular master/slave short-shift mode, if the ADC1 were triggered during ADC2 calibration, the ADC1 continues converting until the end of ADC2 calibration. The subsequent trigger would cause ADC converted data disorder.

- Workaround:

  Do not perform trigger operation during calibration, which can be done by the following:

  For software trigger — enable software trigger at the end of calibration

  For external trigger — enable trigger source at the end of calibration

- Revision plan:

  This issue has been fixed in revision B.

1.2 CAN

1.2.1 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 due to external disturbance, CAN will stop receiving the current data frame and send an error to the bus. In such circumstance, a disorder issue will happen to the next data frame, but the subsequent messages are able to return to normal automatically.

- Workaround:

  Method 1:

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

  This method applies to the scenario where a quick CAN initialization is required to ensure a quick resume of CAN communication in order to avoid excess 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) The interrupt in CAN Error Type Record (ETR) 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 error interrupt (its priority must be set as very high) corresponding to the CAN error interrupt number in the Error Type Record (ETR bit). Once a bit stuffing error is detected, reset CAN (only reset CAN registers and relevant GPIOs, without the need of 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 must be re-initialized in order not to affect operations of other applications.

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). If set, call the CAN initialization function.

Notes:

a) The interrupt in CAN Error Type Record should be given 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 CAN error interrupt (its priority must be set as very high) corresponding to the CAN error interrupt number in the Error Type Record (ETR bit). Once a bit stuffing error is detected, send an invalid message with a very-high-priority identifier.

This method applies to the scenario in which one doesn’t want to spend time on resetting CAN , 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 the error interrupt corresponding to the last CAN error interrupt number and set very high interrupt
      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 function*/
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) The interrupt in CAN Error Type Record should be configured as very high priority
This method is only applicable to the scenario where the transmit FIFO priority is determined by the message identifier.

b) The identifier of the invalid message of this method is changeable. But its priority must be the highest among the CAN bus, and it cannot be received as a normal message by other nodes.

- Revision plan: None.

1.2.2 Failed to filter RTR bit 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) is unable to 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 decide if whether this frame is of interest. 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.

- Revision plan: None.

1.2.3 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:

  Set 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;

   …
}
```

- Revision plan:

  None.

1.2.4 Fail to cancel mailbox transmit command when CAN bus disconnected

- Description:

  As a node for data transmission, if the following two conditions are both present for CAN, it is not possible to clear or cancel a transmit command in a mailbox within CAN error passive interrupt, causing that the to-be-sent message command has not been canceled during the period of CAN bus disconnection, and that such message would be retransmitted after CAN bus communication resumes.

  1. CAN bus (CANH/L) is disconnected deliberately or accidentally
  2. Automatic retransmission feature is enabled

- Workaround:

  Enable CAN error passive interrupt and disable its automatic retransmission before re-enabling automatic retransmission in the message transmit function, as shown below:

  1) Enable error passive interrupt during CAN initialization

```
nvic_irq_enable(CAN1_SE_IRQn, 0x00, 0x00);
can_interrupt_enable(CAN1, CAN_EPIEN_INT, TRUE);
can_interrupt_enable(CAN1, CAN_EOIEN_INT, TRUE);
```

  2) Disable automatic transmission feature in CAN error passive interrupt function

```
void CAN1_SE_IRQHandler(void)
{
  if(can_flag_get(CAN1,CAN_EPF_FLAG) != RESET)
  {
    CAN1->mctrl |= (uint32_t)(1<<4);
    can_flag_clear(CAN1, CAN_EPF_FLAG);
  }
}
```

  3) Re-enable automatic transmission feature in CAN message transmit function

```
CAN1->mctrl &= (uint32_t)~(1<<4);
```

- Revision plan: None

1.3 GPIO

1.3.1 TMR2 IO multiplexed function failed when using USBFS SOF as TMR2_IS1 input

- Description:

  When the SOF bit of the USBFS is used as TMR2_IS1 input (internal connection), the input and output channels of the TMR2 will be affected by this, causing that its input channel only can use the default IO with No Remap, and that its output channel cannot be used.
- Workaround:

  None.
- Revision plan:

  This issue has been fixed in revision B.

1.4 I2C

1.4.1 1st clk sent by master may get shorter when SCL stretching is released by slave in I2C master mode

- Description:

  In I2C master mode, when the slave releases the SCL bus at the place where the slave enables SCL stretching feature, the first clk sent by the master may got shorter so that the slave cannot receive data normally.
- Workaround:

  Add a delay to the location where SCL stretching feature is enabled to avoid sending data immediately after SCL stretching is released.
- Revision plan:

  This issue has been fixed in revision B.

1.4.2 START failed to be sent in I2C master mode

- Description:

  In I2C master mode, if the slave holds down the bus while sending RESTART signal, the START may fail to be sent.
- Workaround:

  Add a delay to the place where the slave enables SCL stretching in order to avoid sending data immediately after SCL stretching is released.
- Revision plan:

  This issue has been fixed in revision B.

1.4.3 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.
- Revision plan:

  None.

1.4.4 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.
- Revision plan:

  None.

1.5 I2S

1.5.1 Unable to resume communication automatically while I2S CK line is disturbed

- Description:

  If the CK and WS signals of the I2S are not synchronized inside the device, when glitch on the clock line occurs during actual communication, such glitch will be treated as a CK signal by the I2S, causing communication unable to resume automatically.
- Workaround:

  Pull-up/pull-down the WS and CK pins internally or externally (depending on the desired audio protocol and I2SCLKPOL configuration). When communication error is detected, disabling and then enabling I2S can help resume communication.
- Revision plan:

  This issue has been fixed in revision B.

1.5.2 I2S Philips protocol SOF data error in certain conditions

- Description:

  In the case of I2S Philips protocol, master receive and slave transmit, and I2SCLKPOL set high, the falling edge of the WS signal on the left channel of the first data frame would not be output, which causes some devices unable to receive data from the left channel.
- Workaround:

  Pull-up/pull-down the WS and CK pins internally or externally (depending on the desired audio protocol and I2SCLKPOL configuration)
- Revision plan:

  This issue has been fixed in revision B.

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.
- Revision plan:

  None.

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

- Description:

  The UDR flag is set incorrectly when I2S is in slave transmission mode and in discontinuous communication state, even if data have been written before the start of communication.
- Workaround:

  According to the protocol, it is recommended to use DMA or interrupts for fast data transfer in I2S slave transmission mode to ensure smooth communication.
- Revision plan:

  None.

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 the frame format using software.

- Revision plan:

  None.

1.6 PWC

1.6.1 Unable to wake up Deepsleep mode after AHB frequency division

- Description:

  After the AHB frequency is divided, no wakeup sources can wake up the device from Deepsleep mode.

- Workaround:

  Do not perform AHB frequency division in Deepsleep mode.
  Remove AHB frequency division before entering Deepsleep mode, and then configure the desired AHB frequency division after waking up from Deepsleep mode.

- Revision plan:

  None.

1.6.2 Systick interrupt wakes up Deepsleep mode

- Description:

  If the Systick or Systick interrupt is not disabled before entering Deepsleep mode, the Systick would keep running after Deepsleep mode entry, and the subsequent Systick interrupt would wake up Deepsleep mode.

- Workaround:

  Disable Systick or Systick interrupt before entering Deepsleep mode.

- Revision plan:

  None.

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

- Description:

  When a wakeup source arrives at the moment the Deepsleep mode is being entered (it takes 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).

- Revision plan:

  This issue has been fixed in revision B.

1.6.4 SWEF flag is set when a wakeup pin is enabled

- Description:
  Before being enabled, if a Standby wakeup pin (waking up Standby mode) were used as a general GPIO push-pull output (high) or pull-up input, a SWEF flag would be set once the pin is enabled.
- Workaround:
  If the Standby mode wakeup pin were used as a GPIO before, the IO then needs to be re-initialized to be pull-down input or analog input before enabling the wakeup 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);
```

- Revision plan: None.

1.7 CRM

1.7.1 CLKOUT output exception after entering Deepsleep mode

- Description:
  If the DEEPSLEEP_DEBUG bit is set 0, and the CLKOUT is used as a system clock output, there would still be clock output (at LICK clock frequency) on the CLKOUT pin after the Deepsleep mode is entered.
- Workaround:
  Configure CLKOUT as NOCLK before entering Deepsleep mode, and then configure it as system clock output after leaving Deepsleep mode.
- Revision plan:
  None.

1.8 TMR

1.8.1 TMR1/TMR8 read 0x4C address using DMA may cause DMA request error

- Description:
  When a TMR (TMR1/TMR8) sends a DMA request, the lower 8-bit address of the current address bus is 0x4C, and the DMA transfer operation does not change the APB bus address where the TMR is. In this case, a single TMR DMA request would be set again after being cleared, causing possible DMA transfer error.
- Workaround:
  Use other timers than TMR1/8.
- Revision plan:
  This issue has been fixed in revision B.

1.8.2 TMR overflow in encoder mode counter

- Description:
  In encoder mode counter, if the counter counts back and forth between 0 and PR, the OVFIF bit of the TMR would not be set either at an overflow or underflow event.
- Workaround 1:
  The C3IF and C4IF channels of the TMR need to be configured as output mode. C3DT = AR, and C4DT = 0, and enable C3DT 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.
- Workaround 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 set the initial value of the counter to PR/2 so as to avoid timer overflow

Note: 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. Thus This method applies to the scenario where the rotation of the encoder is controlled at certain range.

- Revision plan:

  None.

1.8.3 Slave timer unable to receive reset signal from master timer

- Description:

  When the two following conditions are both present, the slave TMR is unable to receive a reset signal, causing it unable to be triggered for reset.

  The two conditions are as follows:

  1. The slave mode of master TMR is configured in reset mode, and the trigger source of slave mode is from an external signal input
  2. The reset signal from master TMR is being sent to slave TMR while the slave mode of slave TMR is configured in reset mode

- Workaround:

  Change the output signal of master TMR from reset signal to overflow signal. In this way, when the master TMR is reset, so is the slave timer.

- Revision plan:

  Revision B has fixed this issue.

1.8.4 Break input failed when TMREN=0 (TMR disabled)

- Description:

  When TMREN=0 (Timer is not enabled), break input failed to work, causing it unable to trigger break event or interrupt.

  Example: in single-pulse mode, TMREN is cleared (0) automatically at the end of one-cycle counting. But due to above-mentioned reason relating to break input, output enable bit (OEN) cannot be cleared, nor can a break flag be set.

- Workaround:

  None.

- Revision plan:

  None.

1.9 USART

1.9.1 Break frame is sent more than once in a specific condition

- Description:

  While a break frame is being transmitted in receiver mute mode, if a wakeup condition takes places at the end of transmission of break frame, then the control bit of the transmitted break frame (bit 0 in the USART_CTRL1 register) cannot be cleared by hardware, causing that another break frame is sent later forcibly.
- Workaround:

  None.
- Revision plan:

  This issue has been fixed in revision B.

1.9.2 Unable to receive data after enabling smart card mode in a specific condition

- Description:

  If a serial port is enabled before enabling the smart card mode, an idle frame is forced to be sent by hardware when the smart card mode is enabled, causing the serial port unable to receive data during the transmission of the idle frame (reception resumes at the end of the idle frame transmission)
- Workaround:

  Change the execution order of initialization: enable the smart card mode before the serial port.
- Revision plan:

  This issue has been fixed in revision B.

1.10 WWDT

1.10.1 Unable to clear RLDF flag while using WWDT interrupts

- 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 RLDF flag.
- Workaround:

  During WWDT interrupt function handling, feed the WWDT before clearing the RLDF flag.

  ```
  void WWDT_IRQHandler(void)
  {
    wwdt_counter_set(127);
    wwdt_flag_clear();
  }
  ```
- Revision plan:

  This issue has been fixed in revision B

1.11 RTC

1.11.1 Actual RTC counter value is the programmed value plus 1

- Description:

  The actual RTC counter value turns out to be the programmed value plus 1.

- Workaround:

  Set 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);
```

- Revision plan:

  None.

1.12 Flash

1.12.1 Program error may occur when sLib is placed in NZW area

- Description:

  When the sLib (security library) is placed in the non-zero-wait area, if the CPU accesses I-code and D-code back and forth quickly during program run, it is likely that the mismatched bus is used to access sLib during bus ID switching because accessing internal Flash memory takes some time. This mismatching operation sends an erroneous instruction (or data) to CPU, causing program error.

- Workaround:

  Place sLib in the zero-wait area.

- Revision plan:

  None.

1.12.2 Erasing 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 ZW area erase operation, but if the interrupt handler functions involve both ZW and ZW areas, the program exception may occur.

- Workaround:

  The reason behind this issue is that reading Flash during erase operation is not allowed.

  To slove this problem, you need disable interrupt enable bits before starting erase, and then enable them after the completion of erase. Meanwhile, the code related to erase functions must be placed into ZW area or RAM.

- Revision plan:

  None.

1.12.3 CPU read Flash causes program exception during SPIM erase

- Description:

  If CPU reads Flash memory during SPIM erase operation, the read Flash command would be considered as reading SPIM mistakenly, causing data error and then 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 operation.

  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.
- Revision plan:

  None.

1.13 USB

1.13.1 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. Under special conditions, such data would affect the USB and cause an enumeration error.
- Workaround:

  Reduce APB1 clock to 48M or below.
- Revision plan:

  This issue has been fixed in version B.

1.13.2 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 NAK response.
- Workaround:

  Use endpoint numbers 2/6/12/14 for IN data transfer.
- Revision plan:

  This issue has been fixed in version B.

1.14 SPI

1.14.1 CS failing edge was 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.

- Revision plan:

  None.

1.15 EXINT

1.15.1 Software triggers twice EXINT line interrupt responses

- Description:

  When the software trigger function is used for EXINT line, twice interrupt responses are generated on the same EXINT line at each software trigger command.

- Workaround:

  Enter the interrupt function associated with software trigger, and perform the EXINT flag clear command twice.

  Alternatively, use the flag clear function defined in the latest version of BSP (described below), or modify code based on the following method.

- Revision plan:

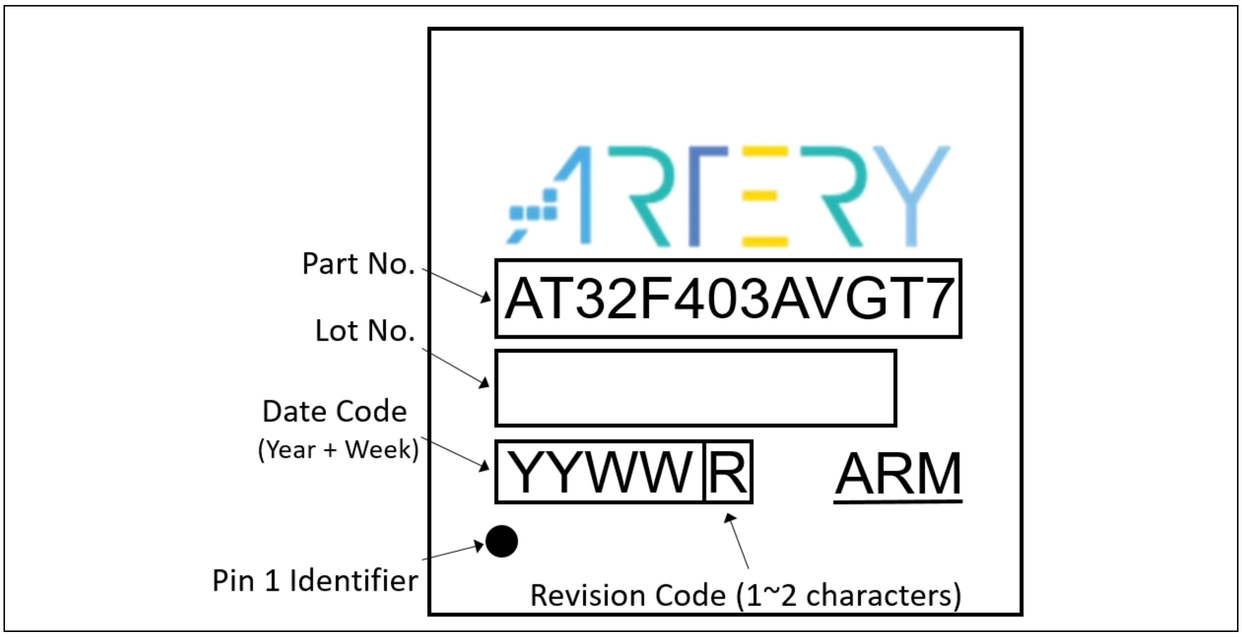
  None.

```
void exint_flag_clear(uint32_t exint_line)
{
        if((EXINT->swtrg & exint_line) == exint_line)
        {
            EXINT->intsts = exint_line;
            EXINT->intsts = exint_line;
        }
        else
        {
            EXINT->intsts = exint_line;
        }
}
```

2 Revision code on device marking

Figure 1 shows the revision code on AT32F403A device marking. The first code is R (revision code). For example, if “A” were shown at “R” location, it means that the hardware revision of this device is silicon A.

Figure 1. Device marking (Top view)



3 Document revision history

Table 4. Document revision history

Date	Version	Changes
2021.5.27	2.0.0	Initial release
2022.03.01	2.0.1	Added *1.2.2 Failed to filter RTR bit of standard frame in 32-bit identifier mask mode.*
2022.03.30	2.0.2	Added *1.11.1 Actual RTC counter value is the programmed value plus 1.*
2022.04.07	2.0.3	Added *1.2.3 CAN sends unexpected messages in case of narrow pulse disturbance on BS2.* Added Flash-related erratas. Added USB related erratas.
2022.04.15	2.0.4	Added *1.5.3 First data received is misaligned in I2S PCM standard long frame receive-only mode.* Added *1.5.4 UDR flag is set in I2S slave transmission mode and discontinuous communication state.* Added *1.5.5 Data reception error when I2S 24-bit data is packed into 32-bit format* Added *1.4.2 START failed to be sent in I2C master mode.* Added *1.14.1 CS failing edge was not synchronized in slave SPI hardware CS mode.*
2022.04.28	2.0.5	1. Added an example case in the *1.6.4 SWEF flag is set when a wakeup pin is enabled* 2. Added an example case in the *1.2.2 Failed to filter RTR bit of standard frame in 32-bit identifier mask mode* 3. Added *1.4.3 I2C slave communication failed when APB equals or less than 4MHz*
2022.07.20	2.0.6	Added *1.15.1 Software triggers twice EXINT line interrupt responses*
2022.09.06	2.0.7	Added *1.4.4 BUSERR is detected by I2C before start of communication*
2022.10.24	2.0.8	Updated description of *section 1.12.2 Erasing NZW during code execution causes program exception*
2023.03.08	2.0.9	Added *section 1.8.3 Slave timer unable to receive reset signal from master timer*
2023.08.03	2.0.10	Added *section 1.8.4 Break input failed when TMREN=0 (TMR disabled)* Updated the descriptions of *section 1.2.1 Bit stuffing error causes the next data out of order during CAN communication* Added *section 1.2.4 Fail to cancel mailbox transmit command when CAN bus disconnected*