ES0004
Errata Sheet

AT32F425 device limitations

Device identification

This errata sheet applies to ARTERY AT32F425 microcontrollers that feature an ARM™ 32-bit Cortex®-M4 core.

Table 1. Device summary

Device	Flash memory	Part number
AT32F425	64 KB	AT32F425R8T7, AT32F425R8T7 -7, AT32F425C8T7, AT32F425C8U7, AT32F425K8T7, AT32F425K8U7-4, AT32F425F8P7
	32 KB	AT32F425R6T7, AT32F425R6T7 -7, AT32F425C6T7, AT32F425C6U7, AT32F425K6T7, AT32F425K6U7-4, AT32F425F6P7

Contents

List of tables

1 AT32F425 device limitations

Table 2 gives a list of limitations that have been identified so far on the AT32F425 devices.

Table 2. Summary of device limitations

Sections	Description
1.1 CAN	1.1.1 Bit stuffing error causes the next data out-of-order during CAN communication 1.1.2 Unable to filter RTR of standard frame in 32-bit identifier mask mode 1.1.3 CAN sends unexpected messages in case of narrow pulse disturbance on BS2 1.1.4 Fail to cancel mailbox transmit command when CAN bus disconnected
1.2 GPIO	1.2.1 PD2 is occupied after LEXT clock enable
1.3 I2S	1.3.1 I2S communication failed when SPITI mode and 3-divided frequency are enabled 1.3.2 First data error in I2S PCM standard long frame receive-only mode 1.3.3 UDR flag is set in I2S slave transmission mode and discontinuous communication state 1.3.4 Data reception error when I2S 24-bit data is packed into 32-bit format
1.4 PWC	1.4.1 Unable to wakeup Deepsleep mode after AHB frequency division 1.4.2 Unable to select system clock source after waking up Deepsleep mode 1.4.3 SWEF flag is set when enabling a standby-mode wakeup pin
1.5 SPI	1.5.1 CS pulse flag is set before enabling SPI in slave SPI TI mode 1.5.2 CS failing edge not synchronized in slave SPI hardware CS mode
1.6 USART	1.6.1 USART ROERR flag is set
1.7 TMR	1.7.1 How to clear TMR-triggered DAM requests 1.7.2 Break input failed when TMREN=0 (TMR disabled)
1.8 CRM	1.8.1 PLL 2x or 3x multiplication factor
1.9 ERTC	1.9.1 Writing ERTC occupies APB for 4 ERTC clock

1.1 CAN

1.1.1 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) 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 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). 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 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 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.

1.1.2 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 present, follow the “Workaround” to solve this problem:

  1. Enable 32-bit wide identifier mask mode
  2. Filter standard frames but not expect to receive remote frames that meet filtering conditions

- 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 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.

1.1.3 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;

  …
}
```

1.1.4 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);
```

1.2 GPIO

1.2.1 PD2 is occupied after LEXT clock enabled

- Description:

  The PD2 pin is occupied after LEXT clock is enabled, so that it cannot be released for other possible use. Normally, after LEXT clock is turned on, it is PC14 and PC15 rather than PD2 which should be owned by LEXT.

- Workaround:

  None.

1.3 I2S

1.3.1 I2S communication failed when SPITI mode and 3-divided frequency are enabled

- Description:

  If three-divided frequency feature or SPI TI mode is enabled, I2S communication error would occur.

- Workaround:

  This is an abnormal operation. Neither SPI TI mode nor three-divided frequency feature are applicable to I2S. They are forbidden in I2S applications.

1.3.2 First data error 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 asserted and the SCK line remains high before enabling I2S, the first data received would be incorrect.

- Workaround:

  Pull up or pull down the SCK pin externally or internally, depending on the I2SCLKPOL configuration.

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

- Description:

  The UDR flag is set in I2S slave transmission mode alongside 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.

1.3.4 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.

1.4 PWC

1.4.1 Unable to wakeup Deepsleep mode after AHB frequency division

- Description:

  If AHB frequency division is configured, no wake 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.

1.4.2 Unable to select system clock source after waking up Deepsleep mode

- Description:

  When a wakeup source arrives at the moment while the Deepsleep mode is entered, either HEXT or PLL could no longer be selected as the clock source of system clock after waking up Deepsleep mode.

- Workaround:

  After waking up Deepsleep mode, wait around 3 LICK clock cycles before starting system clock configuration.

1.4.3 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 immediately 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 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);
```

1.5 SPI

1.5.1 CS pulse flag is set before enabling SPI in slave SPI TI mode

- Description:

  In SPI slave TI mode, if CS and SCK pin are disturbed before enabling SPI, a frame format error would occur and an error interrupt is generated.

- Workaround:

  Enable/disable TI mode and SPI simultaneously.

1.5.2 CS failing 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.

1.6 USART

1.6.1 USART ROERR flag is set exceptionally

- Description:

  As a receiver, if the RX line low level is detected and a Start bit is detected accordingly during STOP bit, the ROERR flag will be set exceptionally. This causes a higher baud rate of the transmitter and the ROERR flag to be set when performing consecutive data transfer.

- Workaround:

  Do not use ROERR flag to determine whether data reception overruns or not. The USART must not enable the error interrupt ERRIEN during DMA reception.

1.7 TMR

1.7.1 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.

1.7.2 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.

1.8 CRM

1.8.1 PLL 2x or 3x multiplication factor error

- Description:

  PLL output clock should be greater than or equal to 16 MHz due to PLL output range limitations. A lower PLL input clock frequency may cause an error when 2x or 3x multiplication factor is used.

- Workaround:

  Try not to use 2x or 3x multiplication factor of the PLL.

1.9 ERTC

1.9.1 Writing ERTC occupies APB for 4 ERTC clock cycles

- Description:

  Writing ERTC register takes approximately four ERTC CLK clock cycles to be synchronized with the battery powered domain, causing APB1 to be occupied and DMA transfer on APB1 to be halted until the completion of the operation process.

- Workaround:

  After ERTC initialization, if ERTC features can satisfy users' needs, try to reduce the times of writing ERTC registers so as to reduce its impact on system.

2 Document revision history

Table 3. Document revision history

Date	Revision	Changes
2021.12.21	2.0.0	Initial release
2022.03.01	2.0.1	1. Added *Unable to filter RTR of standard frame in 32-bit identifier mask mode.* 2. Added *PLL 2x or 3x multiplication factor .*
2022.04.15	2.0.2	1. Added *SWEF flag is set when enabling a standby-mode wakeup pin*. 2. Added *CAN sends unexpected messages in case of narrow pulse disturbance on BS2.* 3. Added *First data error in I2S PCM standard long frame receive-only mode.* 4. Added *UDR flag is set in I2S slave transmission mode and discontinuous communication state.* 5. Added Data reception error when I2S 24-bit data is packed into 32-bit format. 6. Added *CS failing edge not synchronized in slave SPI hardware CS mode.*
2022.04.28	2.0.3	1. Added an example case in the *1.1.2 Unable to filter RTR of standard frame in 32-bit identifier mask mode* 2. Added an example case in the *1.4.3 SWEF flag is set when enabling a standby-mode wakeup pin*
2022.09.21	2.0.4	Added *1.9.1 Writing ERTC occupies APB for 4 ERTC clock*
2023.02.08	2.0.5	Updated the description of *1.9.1 Writing ERTC occupies APB for 4 ERTC clock*
2023.08.03	2.0.6	Added the section *1.7.2 Break input failed when TMREN=0 (TMR disabled)* Updated the descriptions of section *1.1.1 Bit stuffing error causes the next data out-of-order during CAN communication* Added the section *1.1.4 Fail to cancel mailbox transmit command when CAN bus disconnected*

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 granted 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 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 on any patent, copyright or other intellectual property right.

Purchasers hereby agree 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 aircraft application; (C) any aerospace application or environment; (D) any weapon application, and/or (E) or other uses where the failure of the device or product could result in personal injury, death, property damage. Purchasers' unauthorized use of them in the aforementioned applications, even if with a written notice, is solely at purchasers' risk, and Purchasers are solely responsible for meeting all legal and regulatory requirements in such use.

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