ES0003
Errata Sheet

AT32F435/437 device limitations

Device identification

This errata sheet applies to ARTERY AT32F435/437 microcontrollers based on an ARM™ 32-bit Cortex®-M4 core.

The full list of part numbers is shown in Table 2. The products are identifiable as shown in table 1:

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

Table 1. Device identification

Part number	Revision code printed on device
AT32F435/437	“A”
	“B”

1. The Bit [78:76] Mask_Version in the device capacity and unique ID (UID base address 0x1FFF F7E8) shows the revision code of the device. That is, the bit [6:4] at the address 0x1FFFF7F1 can be used to get the revision code, for example
   Revision A: 0b000
   Revision B: 0b001

2. Refer to *Chapter 2* for details on how to identify the revision code on the different packages.

Table 2. Device summary

Device	Part number
AT32F435	AT32F435 series
AT32F437	AT32F437 series

Contents

List of tables

1 AT32F435/437 device limitations

Table 3 gives a list of limitations that have been identified so far on the AT32F435/437 devices.

Table 3. Summary of device limitations

Section	Description	Revision	Revision
1.1 CAN	1.1.1 Bit stuffing error causes the next data out-of-order during CAN communication.	Fail	Fixed
	1.1.2 Failed to filter RTR bit of standard frame in 32-bit identifier mask mode.	Fail	Fixed
	1.1.3 CAN sends unexpected messages in case of narrow pulse disturbance on BS2.	Fail	Fixed
	1.1.4 Fail to cancel mailbox transmit command when CAN bus disconnected	Fail	Fail
1.2 DMAMUX	1.2.1 Setting EVTGEN bit for DMAMUX synchronization.	Fail	Fail
1.3 EDMA	1.3.1 Preemption priority between data streams failed in EDMA linked list mode	Fail	Fail
1.4 I2S	1.4.1 I2S communication failed when SPITI mode and 3-divided frequency are enabled simultaneously.	Fail	Fail
	1.4.2 First data error in I2S PCM standard long frame receive-only mode.	Fail	Fail
	1.4.3 UDR flag is set in I2S slave transmission mode and discontinuous communication state.	Fail	Fail
	1.4.4 Data reception error when I2S 24-bit data is packed into 32-bit format.	Fail	Fail
1.5 PWC	1.5.1 Unable to wakeup Deepsleep mode after AHB frequency division.	Fail	Fail
	1.5.2 Unable to select system clock source after waking up Deepsleep mode	Fail	Fixed
	1.5.3 SWEF flag is set when enabling a standby-mode wakeup pin.	Fail	Fail
	1.5.4 Precautions on LDO use	Fail	Fail
	1.5.5 Entering Deepsleep mode during DMA/EDMA transfer causes data transfer error	Fail	Fail
	1.5.6 Unable to configure system clock after waking up Deepsleep mode	Fail	Fail
	1.5.7 VBAT powered domain register power-on reset failure	Fail	Fail
	1.5.8 LEXT fails to vibrate when both VBAT and VDD power-on at 3ms/V simultaneously	Fail	Fail
1.6 SDRAM	1.6.1 SDRAM read error in burst read mode.	Fail	Fixed
	1.6.2 SDRAM low-power mode limitations.	Fail	Fail
	1.6.3 SDRAM and other XMC static memory usage limitations.	Fail	Fixed(1)
1.7 SPI	1.7.1 CS pulse flag is set in SPI slave TI mode	Fail	Fail
	1.7.2 CS failing edge not synchronized in SPI slave hardware CS mode	Fail	Fail
	1.7.3 Unable to clear data reception DMA transfer request by reading DT register	Fail	Fail
1.8 QSPI	1.8.1 QSPI access error when QSPI is not initialized as an XIP port	Fail	Fixed
	1.8.2 Counter error in QSPI XIP port D mode write configuration	Fail	Fixed
	1.8.3 QSPI Cache usage limitations	Fail	Fixed
	1.8.4 QSPI clock polary selection limitation	Fail	Fixed
	1.8.5 DMA P2M mode usage condition in QSPI command port mode	Fail	Fail
	1.8.6 Excess dummy clock sent after read operation in QSPI command port	Fail	Fail

Section	Description	Revision	Revision
	mode		
	1.8.7 CS line kepts low when QSPI in XIP port D mode	Fail	Fail
1.9 USART	1.9.1 USART ROERR flag is set mistakenly	Fail	Fixed
1.10 ADVTM	1.10.1 How to clear TMR-triggered DAM requests.	Fail	Fixed
	1.10. 2 TMR overrun in encoder mode counter.	Fail	Fail
	1.10.3 Break input failed when TMREN=0	Fail	Fail
1.11 ERTC	1.11.1 Writing ERTC occupies APB for 4 ERTC clock cyles	Fail	Fail
1.12 FLASH	1.12.1 UID or F_SIZE read error during ZW erase session	Fail	Fail

(1) For Revision B, SDRAM, XMC PSRAM, NOR FLASH and SRAM can be used at the same time but it should be noted that SDRAM must be initialized before use and SDRAM can not be set in Low-power mode. Other XMC static memories such as NAND and PC card cannot be used simultaneously.

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, but the next data frame will be out of order while the subsequent messages are able to return to normal automatically.

- 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 and avoid 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 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 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.

d) The invalid message may be a remote frame with no payload.

e) If there is a need for data transmission via nodes, it is necessary to first cancel the message to send in the mailbox before being able to send the invalid message

f) If the bus is in poor conditions which are likely to lead to node passive error status, this method is not applicable.

- Revision: This issue is fixed in Revision B.

1.1.2 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) 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 of message 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: This issue is fixed in Revision B.

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 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: This issue is fixed in Revision B.

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 being disconnected, and such message would be retransmitted after CAN bus communication resumes.

  1. CAN bus (CANH/L) is disconnected intentially or accidentally
  2. Auotmaitc 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:

  None.

1.2 DMAMUX

1.2.1 Setting EVTGEN bit for DMAMUX synchronization

- Description:

  To use DMAMUX synchronization, the EVTGEN must be set to 1 in addition to SYCEN=1, otherwise the synchronization signal does not take effect.

- Workaround:

  Set the EVTGEN bit while configuring synchronization by software.

- Revision: None.

1.3 EDMA

1.3.1 Preemption priority between data streams failed in EDMA linked list mode

- Description:

  When more than one data streams are configured in linked list mode, the preemption priority between data streams becomes invalid.

- Workaround:

  None.

- Revision: None.

1.4 I2S

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

- Description:

  If three-divided frequency feature and SPI TI mode are enabled simultaneously, I2S communication error would occur.

- Workaround:

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

- Revision: None.

1.4.2 First data error in I2S PCM standard long frame receive-only mode

- Description:

  When the following three conditions are present for I2S, it is likely that the first data you receive is incorrect but the subsequent data can return to normal

  The three conditions are as follows:

  1. Set PCM long frame standard receive-only mode
  2. I2SCPOL = 0
  3. SCK remains high, which is abnormal, before I2S enable

- Workaround:

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

- Revision: None.

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

- Description:

  The UDR flag is set in I2S slave transmit mode combined with 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.

- Revision: None.

1.4.4 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.

- Revision: None.

1.5 PWC

1.5.1 Unable to wakeup Deepsleep mode after AHB frequency division

- Description:

  If AHB frequency is divided, no wakeup sources 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 waking up Deepsleep mode.

- Revision: None.

1.5.2 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 being entered, either HEXT or PLL could no longer be selected as the clock source of system clock.
- Workaround:
  After waking up Deepsleep mode, wait around 3 LICK clock cycles before starting system clock configuration.
- Revision: This issue is fixed in Revision B.

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

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

```
gpio_init_type gpio_init_struct;

/* enable the wakeup pin 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 = GPIO_PINS_0;
gpio_init_struct.gpio_pull = GPIO_PULL_DOWN;
gpio_init(GPIOA, &gpio_init_struct);

/* enable wakeup pin1-pa0 */
```

```
pwc_wakeup_pin_enable(PWC_WAKEUP_PIN_1, TRUE);
```

- Revision: None.

1.5.4 Precautions on LDO use

- Description:

  The LDO output voltage can be adjusted to reduce overall power consumption, but the following two points worth noting:

  1） Disable PWC clock five LICK clock cycles after LDO configuration by software

  2） The interval time between two LDO configurations must be greater than 5 LICK clock cycles

- Workaround: Follow the instructions below after adjusting LDO output voltage

```
pwc_ldo_output_voltage_set(PWC_LDO_OUTPUT_1V0);
crm_sysclk_switch_status_get();///<access SBUS
for ( delay_index = 0; delay_index < 80; delay_index++) ///< 5 LICK clock delay at 8MHz
{
   __ISB();
 }
/* Disable PWC clock, enter sleep mode, enter deepsleep mode, start another LDO configuration */
```

- Revision: None.

1.5.5 Entering Deepsleep mode during DMA/EDMA transfer causes data transfer error

- Description:

  Executing Deepsleep command during DMA/EDMA transfer likely causes DMA/EDMA to transfer wrong data after waking up from Deepsleeep mode.

- Workaround:

  Disable DMA/EDMA prior to Deepsleep mode entry, and then enable it after waking up from Deepsleeep mode. See below:

```
/* disable dma channel */
dma_channel_enable(DMAx_CHANNELy, FALSE);

/* enter deep sleep mode */
pwc_deep_sleep_mode_enter(PWC_DEEP_SLEEP_ENTER_WFI);

/* enable dma channel */
dma_channel_enable(DMAx_CHANNELy, TRUE);
```

- Revision:
  None.

1.5.6 Unable to configure system clock after waking up Deepsleep mode

- Description:

  The following three conditions, if both present, are likely to result in failure in system clock configurations (that is, HEXT or PLL enable command cannot be activated), after waking up from Deeepsleep mode.

  Condition 1: DBG_DEEPSLEEP = 1

  Condition 2: attempt to wake up from Deepsleep mode at the transit period during which the system is in the process of entering Deepsleep mode. In other words, attempt to wake up Deepsleep mode within three LICK cycles after Deepsleep mode entry comman is performed

  Condition 3: Configure system clock (enable HEXT, PLL) as sonn as wake up from Deepsleep mode

- Workaround:

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

- Revision plan:

  None.

1.5.7 V_{BAT} powered domain register power-on reset failure

- Description:

  The two following conditions, if both present, are likely to result in failure in V_{BAT} powered domain register power-on reset and then lead to uncertainity in its content

  Condition 1: V_{DD} is tied to V_{BAT}

  Condition 2: V_{DD} does not drop below 0.1V before power on

- Workaround:

  Depends on the PORRSTF bit in the CRM_CTRLSTS register. If the PORRSTF bit is set, the user needs to reset VBAB (by writing “1” and then “0” to the BPDRST bit in the CRM_BPDC register, and re-initialize VBAT register before clearing the PORRSTF bit.

- Revision plan:

  None.

1.5.8 LEXT fails to vibrate when both V_{BAT} and V_{DD} power-on at 3ms/V simultaneously

- Description:

  When the following conditions are both present, LEXT may not vibrate after enabled.

  Condition 1: V_{DD} is tied to $V_{BAT,}$ or both are powered on simultaneously

  Condition 2: V_{DD} and V_{BAT} are being powered on at a rate of 3ms/V

- Workaround:

  Enabled PVM feature and use it to judge if power voltage is higher than 2.6V. When 2.6V is reached, check the PORRSTF (power-on reset flag) in the CRM_CTRLSTS register, and then perform a system reset if PORRSTF is set. After that, clear the PORRSTF flag bit.

  It is recommended to peform the reference code below before clock initialization.

```
int main(void)
{
  crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);
  pwc_pvm_level_select(PWC_PVM_VOLTAGE_2V6);
  pwc_power_voltage_monitor_enable(TRUE);
  while(pwc_flag_get(PWC_PVM_OUTPUT_FLAG)==SET);
  if((CRM->ctrlsts_bit.porrstf == SET) && (CRM->ctrlsts_bit.swrstf == RESET))
  {
    NVIC_SystemReset();
  }
  crm_flag_clear(CRM_ALL_RESET_FLAG);
  ……
  system_clock_config();
  ……
  ……
}
```

- Revision plan:

  None.

1.6 SDRAM

1.6.1 SDRAM read error in burst read mode

- Description:

  When BSTR (burst read) bit of the SDRAM controller is enabled, SDRAM read error may occur.

- Workaround:

  Do not use BSTP (Burst Read) feature in SDRAM.

- Revision: This issue is fixed in Revision B.

1.6.2 SDRAM low-power mode limitations

- Description:

  When SDRAM is configured in self-refresh or power-down mode, read/write access to SDRAM device in the process of SDRAM entering low-power mode may not be executed.

- Workaround:

  Do not read/write from/to SDRAM when the self-refresh or power-down mode is being entered.

  After self-refresh or power-down command is sent, it is necessary to ensure that the SDRAM status has switched successfully to self-refresh/power-down mode (get SDRAM status by reading SDRAM_STS register), and wait until the BUSY bit becomes 0 before performing read/write access to SDRAM.

- Revision:
  None.

1.6.3 SDRAM and other XMC static memory usage limitations

- Description:

  It is not allowed to access SRAM and other XMC static memories simultaneously.

- Workaround:

  If there is a need to use SDRAM and other XMC static memories simultaneously, PSRAM or SRAM is recommended.

- Revision:
  For Revision B, SDRAM, XMC PSRAM, NOR FLASH and SRAM can be used at the same time but it should be noted that SDRAM must be initialized before use and SDRAM can not be set in Low-power mode. But except this, other XMC static memories such as NAND and PC card cannot be used simultaneously.

1.7 SPI

1.7.1 CS pulse flag is set in SPI slave TI mode

- Description:

  In SPI slave TI mode, if CS and SCK pin are disturbed when SPI is not enabled, a frame format error would occur and an error interrupt is generated.

- Workaround:

  Enable or disable TI mode and SPI simultaneously.

- Revision:
  None.

1.7.2 CS failing edge not synchronized in SPI slave hardware CS mode

- Description:

  In SPI slave hardware CS mode (non TI 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:
  None.

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

- Description:

  For example, for those applications which use 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, you can clear DMA request by disabling SPI (not reading DT register), and then enabling SPI at a place where you want to start communication.

- Revision:
  None.

1.8 QSPI

1.8.1 QSPI access error when QSPI is not initialized as an XIP port

- Description:

  If the QSPI is not initialized as an XIP port, reading QSPI address through memory read or debug mode will get program error.

- Workaround:

  Do not read QSPI addresses when the QSPI is not yet be initialized as an XIP port.

- Revision: This issue is fixed in Revision B.

1.8.2 Counter error in QSPI XIP port D mode write configuration

- Description 1:
  When the following two conditions are present during the use of QSPI, it is likely that the XIPW_DCNT becomes invalid, acting like XIPW_DCNT=1, so that the efficiency of QSPI write operation will be reduced compared with its expected values.

  The two conditions are as follows:

  1. QSPI is initialized as an XIP endpoint
  2. Select mode D when it comes to write mode configuration

- Workaround:
  Try to use mode T as much as possible. If there is a need to use mode D, it is necessary to evaluate its impact on write operation.

- Revision: This issue is fixed in Revision B.

1.8.3 QSPI Cache usage limitations

- Description:

  QSPI Cache is an enhanced edition of XIP port. This feature is enabled or disabled through the BYPASSC bit of the XIP CMD_W3 register. (It is enabled by default. BYPASSC=1 disables it)

  Such feature, however, has its prerequisites for use: it can be used only when the following scenarios are all present, otherwise, an error may occur.

- Workaround:

  There are two limitations for this workaround:

  Limitation condition 1: For XIP Read only (extend external Flash)

  Limitation condition 2: For XIP T mode only (Set XIPR_SEL bit to 1)

- Revision:

  Limitation condition 2 is fixed in Revision B.

1.8.4 QSPI clock polary selection limitation

- Description:

  When the division value of CLKDIV is 2/4/6/8, it is possible to select mode 0 or mode 3 using the SCKMODE bit.

  When the division value of CLKDIV is 3/5/10/12, SCKMODE bit configuration is invalid and the actual SCK output is mode 0.

- Workaround:

  If there is a need to use mode 3, the CLKDIV division value should be 2/4/6/8.

- Revision: This issue is fixed in Revision B.

1.8.5 DMA P2M mode usage condition in QSPI command port mode

- Description:

  When QSPI is configured in command port mode, a specific condition must be met for data transfer using DMA P2M mode, detailed as follows.

- Workaround:

  When QSPI is configured in command port mode, to use DMA P2M mode to transfer data, the MSIZE must select word format, and data size must be a multiple of 4.

- Revision:

  None.

1.8.6 Excess dummy clock sent after read operation in QSPI command port mode

- Description:

  When QSPI is configured in command port mode, after the completion of read access, an additional dummy clock will be sent, which has no impact on applications in most cases.

- Workaround:

  None.

- Revision: None.

1.8.7 CS line kepts low when QSPI in XIP port D mode

- Description:

  In XIP port D mode, the CS line may remain low in idle state following read operation, which may result in failure in QSPI SRAM's internal refreshing mechanism and data error.

- Workaround:

  Method 1: Use XIP port T mode

  Method 2: If XIP port D mode, set the XIPR_DCNT value in the XIP CMD_W2 register to no more than 0x20

- Revision:

  None.

1.9 USART

1.9.1 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 tramsmitter and ROERR flag to be set when sending consecutive data.
- Workaround:
  Do not use ROERR flag to determine whether data reception overruns or not. The USART must not enable error interrupt ERRIEN during DMA reception.
- Revision: This issue is fixed in Revision B.

1.10 ADVTM

1.10.1 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, reset TMR (reset CRM clock of TMR) and initialize TMR to clear pending DMA requests.
- Revision: This issue is fixed in Revision B.

1.10.2 TMR overrun in encoder mode counter

- Description:
  In encoder counting mode, if the counter counts back and forth between 0 and PR, the OVFIF 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 for software 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.

- Revision: None.

1.10.3 Break input failed when TMREN=0

- Description:

  When TMREN=0 (timer is disabled), break input is inactive, causint it unable to trigger break event or interrupt.
  As an example, in one-pulse mode, the TMREN is automatically cleared at the end of one-cycle counting. In such case, due to the break input being disabled, the output enable bit (OEN) cannot be cleared, nor is the break flag bit set.
- Workaround: None.
- Revision: None.

1.11 ERTC

1.11.1 Writing ERTC occupies APB for 4 ERTC clock cyles

- 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 during this period 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.
- Revision: None.

1.12 FLASH

1.12.1 UID or F_SIZE read error during ZW erase session

- Description:

  Reading UID or F_SIZE registers listed in the table below during Flash zero-waite-state area (ZW) erase period will return erroneous data.

Table 1-5 List of abbreviations for registers

Register abbr.	Base address	Reset value
F_SIZE	0x1FFF F7E0	0xXXXX
UID[31: 0]	0x1FFF F7E8	0xXXXX XXXX
UID[63: 32]	0x1FFF F7EC	0xXXXX XXXX
UID[95: 64]	0x1FFF F7F0	0xXXXX XXXX

- Workaround:

  Method 1: obtain the values of UID or F_SIZE registers during program initialization period and store them as variables, and then call them if needed, see code below:

```
uint32_t uid_buf[3] ={0};
uint32_t fsize = 0;


uid_buf[0] = *(uint32_t*)0x1FFFF7E8;
```

```
uid_buf[1] = *(uint32_t*)0x1FFFF7EC;
uid_buf[2] = *(uint32_t*)0x1FFFF7F0;
fsize = *(uint32_t*)0x1FFFF7E0;
```

Method 2: disable interrupts prior to Flash erase operation to avoid UID or F_SIZE registers being read within interrupts, and then re-enable interrupts after the completion of erase session.

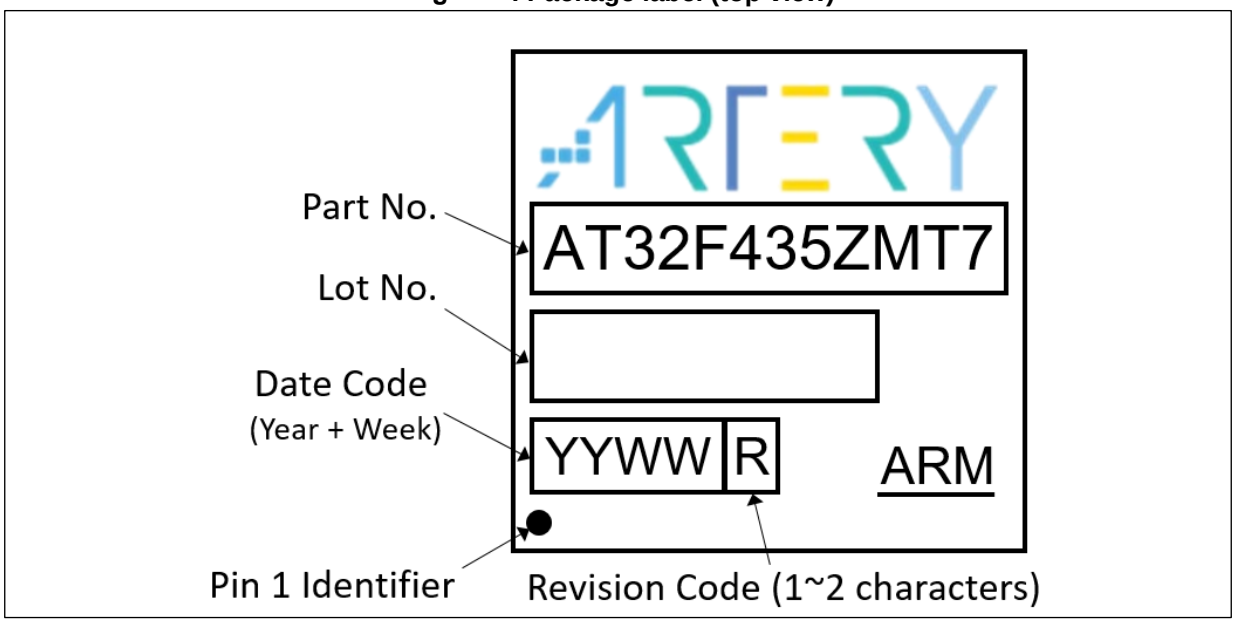
```
__disable_irq();
/* call erase function to start erasing*/
__enable_irq();
```

- Revision plan: None.

2 Revision code on device marking

Figure 1 shows the location of revision code on AT32F435/437 device marking. The first code is R (revision code). For example, if B is shown in the R location, it means that the hardware revision of this device is silicon B.

Figure 1. Package label (top view)



3 Document revision history

Table 4. Document revision history

Date	Revision	Changes
2021.9.30	2.0.0	Initial release
2022.3.1	2.0.1	1. Added *SDRAM low-power mode limitations.* 2. Added *SDRAM and other XMC static memory usage limitations.* 3. Added *Counter error in QSPI XIP port D mode write configuration* 4. Added *Failed to filter RTR bit of standard frame in 32-bit identifier mask mode.*
2022.3.30	2.0.2	1. Added *SWEF flag is set when enabling a standby-mode wakeup pin.* 2. Added *QSPI Cache .*
2022.04.15	2.0.3	1. Added *CAN sends unexpected messages in case of narrow pulse disturbance on BS2.* 2. Added *QSPI Cache .* 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 *1.7.2 CS failing edge not synchronized in SPI slave hardware CS mode*
2022.04.27	2.0.4	1. Modified the description of the section *1.8.3 QSPI Cache usage limitation* 2. Added an example case in the *1.1.2 Failed to filter RTR bit of standard frame in 32-bit identifier mask mode* 3. Added an example case in the *1.5.3 SWEF flag is set when enabling a standby-mode wakeup pin*
2022.08.15	2.0.5	Updated *1.8.3 QSPI Cache usage limitation*
2022.08.23	2.0.6	Added *1.6.3 SDRAM and other XMC static memory usage limitations*
2022.10.19	2.0.7	Added *1.11.1 Writing ERTC occupies APB for 4 ERTC clock cyles* Added *1.5.4 Precautions on LDO use* Added *1.8.5 QSPI clock polary selection limitation*
2023.03.09	2.0.8	Added *Table 1 Device identification*
2023.08.03	2.0.9	1. Added *1.10.3 Break input failed when TMREN=0* 2. Updated the descriptions of *1.1.1 Bit stuffing error causes the next data out-of-order during CAN communication* 3. Added *1.1.4 Fail to cancel mailbox transmit command when CAN bus disconnected* 4. Added *1.8.5 DMA P2M mode usage condition in QSPI command port mode* 5. Added *1.8.6 Excess dummy clock sent after read operation in QSPI command port mode*
2023.08.17	2.0.10	6. Updated the descriptions of *1.8.3 QSPI Cache usage limitations*
2024.06.24	2.0.11	Added Section *1.12.1 UID or F_SIZE read error during ZW erase session*
2024.07.12	2.0.12	Addded Section *1.8.7 CS line kepts low when QSPI in XIP port D mode*
2024.12.05	2.0.13	Updated “Workaround” descriptions in *Section 1.8.3 QSPI Cache usage limitations*
2025.01.09	2.0.14	1. Updated “Method 3” descriptions in *Section 1.1.1 Bit stuffing error causes the next data out-of-order during CAN communication*

		2. Updated descriptions in *Section 1.5.6 Unable to configure system clock after waking up Deepsleep mode* 3. Added *Section 1.5.7 VBAT powered domain register power-on reset failure*
2025.04.23	2.0.15	Added *Section 1.5.8 LEXT fails to vibrate when both VBAT and VDD power-on at 3ms/V simultaneously*

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.