ES0001

Errata Sheet

AT32F415 device limitations

Device identification

This errata sheet applies to Artery's AT32F415 series 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
AT32F415	"B"
	"C"

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 B: 0b001
   Revision c: 0b010

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

Table 2. Device summary

Devices	Flash memory	Part numbers
AT32F415	256 KB	AT32F415RCT7, AT32F415RCT7-7, AT32F415CCT7, AT32F415CCU7, AT32F415KCU7-4
	128 KB	AT32F415RBT7, AT32F415RBT7-7, AT32F415CBT7, AT32F415CBU7, AT32F415KBU7-4
	64 KB	AT32F415R8T7, AT32F415R8T7-7, AT32F415C8T7, AT32F415K8U7-4

Contents

List of tables

List of figures

1 AT32F415 device limitations

Table 3 gives a list of limitations that have been identified so far on the AT32F415 devices.

Table 3. Summary of the device limitations

Section	Description	Revision B	Revision C
1.1 ADC	1.1.1 ADC regular group conversion error due to preempted group configuration change.	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 ERTC	1.3.1 How to enable wakeup event output on TAMPER PIN.	Fail	Fixed
	1.3.2 How to update TIME and DATE register value.	Fail	Fail
	1.3.3 3 to 6 LEXT clock cycles delay after each system reset when LEXT as ERTC clock source.	Fail	Fixed
1.4 GPIO	1.4.1 PC0~5 pull-down resistors are turned on abnormally.	Fail	Fixed
	1.4.2 FT (5V tolerant pin) maintains at intermediate level in floating input mode	Fail	Fail
1.5 I2S	1.5.1 Failed to resume communication when I2S CK line is interfered.	Fail	Fixed
	1.5.2 I2S Philips protocol Start Frame data error under certain conditions.	Fail	Fixed
	1.5.3 The first received data is misaligned in I2S PCM standard long frame receive-only mode.	Fail	Fail
	1.5.4 UDR flag is set mistakenly 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 OTG	1.6.1 VBUS (PA9) cannot be released to other peripherals in OTG_FS Device mode.	Fail	Fixed
1.7 PWC	1.7.1 PVM event generation after PVM enable when VDD is above PVM threshold.	Fail	Fail
	1.7.2 Unable to wakeup Deepsleep mode after AHB frequency division.	Fail	Fail
	1.7.3 Systick interrupt wakes up Deepsleep mode mistakenly.	Fail	Fail
	1.7.4 Waking up Deepsleep mode while Deepsleep mode is being entered causes instruction operation exception.	Fail	Fixed
	1.7.5 SWEF flag is set when enabling a standby-mode wakeup pin.	Fail	Fixed
	1.7.6 Unable to configure system clock after waking up Deepsleep mode.	Fail	Fail
	1.7.7 How to save more power during Run and Sleep mode.	Fail	Fixed
	1.7.8 VBAT powered domain register power-on reset failure	Fail	Fail
1.8 SPI	1.8.1 Data reception transfer DMA request fails to be cleared by reading DT register.	Fail	Fail
	1.8.2 CS failing edge was not synchronized in slave SPI hardware CS mode.	Fail	Fail
1.9 TMR	1.9.1 Suspend mode failed in external clock mode B.	Fail	Fixed
	1.9.2 How to clear TMR-triggered DAM requests	Fail	Fixed
	1.9.3 TMR overrun in encoder mode counter.	Fail	Fixed
	1.9.4 TMR accessing 0x4C address with DMA causes DMA request error.	Fail	Fixed
	1.9.5 Slave timer unable to receive reset signal from master timer	Fail	Fail
	1.9.6 Break input failed when TMREN=0 (TMR disabled)	Fail	Fail
	1.9.7 Fail to generate CxORAW clear event when dead-time is disabled	Fail	Fail
1.10 USART	1.10.1 Enabling USART3 and TMR1/TMR3 causes PA7 error.	Fail	Fixed
	1.10.2 USART failed to receive data in IrDA mode.	Fail	Fixed
	1.10.3 Clearing TDC flag immediately after USART initialization causes data transfer error.	Fail	Fixed
	1.10.4 Clearing RDBF bit only by reading data register.	Fail	Fixed

Section	Description	Revision B	Revision C
	1.10.5 USART can still receive data using DMA in silent mode.	Fail	Fail
1.11 WWDT	1.11.1 Unable to clear RLDF flag while using WWDT interrupts.	Fail	Fixed
1.12 WDT	1.12.1 Entering Standby mode immediate after enabling WDT will trigger a reset.	Fail	Fixed
	1.12.2 Entering Deepsleep mode immediately after enabling WDT causes WDT enable failure.	Fail	Fixed
1.13 CRM	1.13.1 CLKOUT clock output exception after entering Deepsleep mode.	Fail	Fail
	1.13.2 PLL 2x or 3x multiplication factor failure.	Fail	Fail
1.14 I2C	1.14.1 I2C slave communication error when APB equals or less than 4MHz	Fail	Fail
	1.14.2 I2C communication error when BUSERR is detected on bus	Fail	Fail
1.15	1.15.1 SLib and boot memory AP mode settings	Fail	Fail

1.1 ADC

1.1.1 ADC regular group conversion error due to preempted group configuration change

- Description:

  In ADC sequential and repetition conversion mode of regular group, attempting to change channel configuration of preempted group during a regular group conversion will cause regular group conversion data out of order.

  For example, when the regular group is converting the channel 1, 2, 3 and 4, attempting to change preempted group configuration during channel-2 conversion will have the channel 2 converted twice, which results in the convertion order of regular group to be 1, 2, 2, 3, and 4.

- Workaround:

  When using multi-channels for regular and preempted conversion simultaneously, do not try to change its channel configuration after preempted group is configured.

- Revision plan:

  Revision C has fixed this issue.

1.1.2 Unable to clear and set ADC preempted channel conversion end flag

- Description:

  When “PCCE” (preempted channel conversion end flag) and “CCE” (ordinary channel conversion end flag) events occur simultaneously, it is likely that PCCE flag cannot be cleared immediately, causing the next PCCE flag unable to be set.

- Workaround:

  Perform the same command one more time under the original PCCE flag clear command to clear this flag. See below:

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

- Revision plan:

  None.

1.2 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 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_reset(CAN1);
      /* Call CAN initialization function */
    }
  }
}
```

Notes:

a) CAN error interrupts should be given a very high priority

b) It takes some time to finish CAN initialization, so 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 not necessary 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 a very high priority

b) It takes some time to finish CAN initialization, so 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 the user 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 a 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 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) cannot be filtered effectively during a standard frame filtering.

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

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

- Workaround:

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

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

Method 2: Use other filtering mode according to 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:

  Configure synchronization jump 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 = 10;
  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 ERTC

1.3.1 How to enable wakeup event output on TAMPER PIN

- Description:

  No wakeup event output on the TAMPER PIN at a wakeup event.

- Workaround:

  Enable wakeup timer interrupt (WATIEN = 1), an interrupt is generated as soon as a wakeup event occurs, read then the wakeup timer flag (WATF) and clear it in the interrupt routine functions.

- Revision plan:

  Revision C has fixed this issue.

1.3.2 How to update TIME and DATE register value

- Description:

  If no operation is performed on the ERTC register, the TIME and DATE registers are not updated, holding the value updated when the ERTC register was accessed last time.

- Workaround:

  Read status register first before reading TIME and DATE registers.

- Revision plan:

  None.

1.3.3 3 to 6 LEXT clock cycles delay after each system reset when LEXT as ERTC clock source

- Description:

  When LEXT is used as the clock source of ERTC, 3~6 LEXT clock cycles are delayed after each system reset (for instance, wake up Standby mode, power-down reset and watchdog reset)

- Workaround:

  None.

- Revision plan:

  Revision C has fixed this issue.

1.4 GPIO

1.4.1 PC0~5 pull-down resistors are turned on abnormally

- Description:

  The GPIOs should remain floating during reset, but the pull-down resistors from PC0 to PC5 were turned on abnormally.

- Workaround:

  None. It is recommended to design circuitry by enabling LED or peripherals through high level.

- Revision plan:

  Revision C has fixed this issue.

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

- Description:

  The 5V tolerant pin still has a pull-up capability of less than 10 μA in floating input mode, which causes it to maintain about 2.0 V.

- Workaround:

  Add an external pull-down resistor (150 kΩ or below)

- Revision plan:

  None.

1.5 I2S

1.5.1 Failed to resume communication when I2S CK line is interfered

- Description:

  The I2S CK and WS signals are not synchronized, so that when the clock line is interfered during communication, this noise/interference would be treated as a CK signal by I2S, causing communication not to resume automatically.

- Workaround:

  Pull up or pull down the WS and CK pins internally or externally, depending on the desired audio protocols and I2SCLKPOL configuration. When communication error is detected, it is possible to disable and enable I2S to resume communication.

- Revision plan:

  Revision C has fixed this issue.

1.5.2 I2S Philips protocol Start Frame data error under certain conditions

- Description:

  In case of I2S Philips protocol, master receive and slave transmission and I2SCLKPOL high level, the WS signal falling edge corresponding to the left channel of the first data frame would not be output effectively, causing some devices to fail to receive data from the left channel.

- Workaround:

  Pull up or pull down the WS and SCK pins internally or externally, depending on the desired audio protocols and I2SCLKPOL configuration.

- Revision plan:

  Revision C has fixed this issue.

1.5.3 The first received data 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 mistakenly in I2S slave transmission mode and discontinuous communication state

- Description:

  The UDR flag is set incorrectly in I2S slave transmission mode in discontinuous communication state, even if data have been written before communication.

- Workaround:

  For continuous communication, it is recommended to use DMA or interrupts for data transfer in I2S slave transmission mode according to the protocols.

- 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 this frame format using software.

- Revision plan:

  None.

1.6 OTG

1.6.1 VBUS (PA9) cannot be released to other peripherals in OTG_FS Device mode

- Description:

  In OTG_FS Device mode, the PA9 must be used as an OTG VBUS input pin, and cannot be allocated to GPIOs or other peripherals. It is recommended that the PA9 is connected with a pull-up resistor to V_{BUS} or V_{DD}

- Workaround:

  None.

- Revision plan:

  Revision C has fixed this issue. For revision C, PA9 can be released to other peripherals by setting the bit 21 VBUSIG of the OTGFS_GCCFG register.

1.7 PWC

1.7.1 PVM event generation after PVM enable when VDD is above PVM threshold

- Description:

  When the VDD is greater than PVM threshold, an unwanted PVM event is generated as soon as PWC voltage monitoring is enabled.

- Workaround:

  Clear the unwanted PVM event during PVM initialization.

- Revision plan:

  None.

1.7.2 Unable to wakeup Deepsleep mode after AHB frequency division

- Description:

  If AHB frequency division is configured, it is impossible to wake up Deepsleep mode by any one wakeup source.

- 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.

- Revision plan:

  None.

1.7.3 Systick interrupt wakes up Deepsleep mode mistakenly

- Description:

  If Systick or Systick interrupt is not disabled before the Deepsleep mode is entered, the Systick then would keep running after the Deepsleep mode is entered, and the resulting Systick interrupt would wake up Deepsleep mode.

- Workaround:

  Disable Systick or Systick interrupts before entering Deepsleep mode.

- Revision plan:

  None.

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

- Description:

  When a Deepsleep wakeup source arrives at the very moment Deepsleep mode is being entered (it takes around 3 LICK clock cycles), some instructions may be missed (fail to be performed) after waking up Deepsleep mode.

- Workaround:

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

- Revision plan:

  Revision C has fixed this issue.

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

- Description:

  Before being enabled, if a wakeup pin (waking up Standby mode) 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 the wakeup pin (waking up Standby mode) was used as a GPIO before, then the IO needs to be re-initialized to be 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);
```

- Revision plan:

  None.

1.7.6 Unable to configure system clock after waking up Deepsleep mode

- Description:

  The following two 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: 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 2: 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.

1.7.7 How to save more power during Run and Sleep mode

- Description:

  When the 0x4000_7050 [2] is kept at its default value 0, the battery powered domain clock is still present, but many peripherals in the clock domain would consume a large amount of power during Run and Sleep mode in the V_{DD} domain. This consumption still remains even if no read/write access is ongoing in the battery powered domain.

- Workaround:

  It is recommended to set 0x4000_7050 [2]=1 by software, and let the hardware to manage the battery powered domain automatically so as to save power during Run and Sleep mode.

- Revision plan:

  Revision C has fixed this issue.

1.7.8 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.

1.8 SPI

1.8.1 Data reception transfer DMA request fails to be cleared by reading DT register

- Description:

  For example, in SPI full-duplex Tx/Rx mode, invalid data reception transfer DMA request that is set during SPI transmission cannot be cleared by reading DT register.
- Workaround:

  When SPI reception DMA channel is turned off, disable SPI instead of reading DT register, and then enable SPI at a place where you want to start communication.
- Revision plan:

  None.

1.8.2 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.

1.9 TMR

1.9.1 Suspend mode failed in external clock mode B

- **Description:**

  The gated mode does not work (the timer still keeps counting no matter whether the gated level is in high or low) in external clock mode 2.
- Description:

  When Suspend mode is configured in external clock mode B, the Suspend mode doesn’t work (No matter whether it is high or low, the Timer is always running)
- Workaround:

  None.
- Revision plan:

  Revision C has fixed this issue.

1.9.2 How to clear TMR-triggered DAM requests

- Description:

  TMR cannot clear DMR request by reset/set corresponding DMA request enable bit.
- Workaround:

  Before enabling DMA channel transfer, reset TMR (reset CRM clock of TMR) and initialize TMR to clear pending DMA requests.
- Revision plan: Revision C has fixed this issue.

1.9.3 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 interrupts for C3IF and C4IF channels.
  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, it would be interrupted repeatedly and processed by software, so that there is no time to deal with interrupts. Thus this method applies to the scenario where the external input frequency of the encoder is not so fast.

- Method 2:

  Turn to a TMR with enhanced mode (the counter can be extended from 16-bit to 32-bit width) in order to expand the encoder's counting range that detects forward and reverse rotation, and configure the initial value of the counter to PR/2 so as to prevent the timer from overflowing.

  This method has its limitation: The forward and reverse rotation of the encoder can only 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 controllable.

- Revision plan: Revision C has fixed this issue.

1.9.4 TMR accessing 0x4C address with DMA causes DMA request error

- Description:

  If the TMR access to 0x4C offset address using DMA, it is likely to encounter DMA request error when one of the following two conditions is met.

  Scenario 1: When TMR is issuing a DMA request (TMR DMA BURST disabled), the lower 8 bits of the address bus where the TMR is located is 0x4C

  Scenario 2: TMR_DMA BURST feature is enabled

- Workaround:

  For scenario 1: user other timers than TMR1

  For scenario 2: do not use TMR_DMA BURST feature (disable TMR_DMA BURST)

- Revision plan:

  Revision C has fixed this issue.

1.9.5 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.

1.9.6 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.

1.9.7 Fail to generate CxORAW clear event when dead-time is disabled

- Description:

  When the following four conditions are met, a pulse jitter with a periodic dead-time width will happen to the complementary channel output. And the CxORAW signal clear event will delay and not generate until the occurrence of overflow event, even if all conditions to generate CxORAW signal clear event are met.

  Condition 1: PWM complementary output mode is enabled

  Condition 2: CxORAW signal is configured to be cleared by external events

  Condition 3: Dead-time is set

  Condition 4: External input is set as active level

- Workaround:

  None.

- Revision plan:

  None.

1.10 USART

1.10.1 Enabling USART3 and TMR1/TMR3 causes PA7 error

- Description:

  When USART3 is enabled and without remapping, the TMR1/3 cannot use PA7 channel.
- Workaround:

  None.
- Revision plan: Revision C has fixed this issue.

1.10.2 USART failed to receive data in IrDA mode

- Description:

  When USART baud rate is configured to be less than or equal to 38400 in USART IrDA mode, the USART is unable to receive data.
- Workaround:

  The USART baud rate must not be lower than or equal to 38400.
- Revision plan:

  Revision C has fixed this issue.

1.10.3 Clearing TDC flag immediately after USART initialization causes data transfer error

- Description:

  If the TDC flag is cleared immediately after USART initialization, wait to send data after TDC is set. In this case, data transmission error occurs.
- Workaround:

  Do not clear TDC flag by software. When TDC flag is set, it indicates that data transmission is complete. If the TDC is cleared, it cannot be set again before next data transmission, otherwise, the TDC flag would always remain 0.
- Revision plan: Revision C has fixed this issue.

1.10.4 Clearing RDBF bit only by reading data register

- Description:

  In regular asynchronous communication mode, there are two ways to clear the RDBF bit (non-empty flag) of the read data register. Either clear this bit by reading USART_DT register, or by writing 0 to RDBF bit of the status register. However, for revision B, the RDBF bit can only be cleared by reading USART_DT register.
- Workaround:

  None.
- Revision plan:

  Revision C has fixed this issue.

1.10.5 USART can still receive data using DMA in silent mode

- Description:

  When the USART sends data to RX in silent mode (address matching wakeup mode), it still can generate a DMA reception request and the data can also be received by DT data register even though the RDBF is not set. This causes silent mode to fail to work.

- Workaround:

  None.

- Revision plan:

  None.

1.11 WWDT

1.11.1 Unable to clear RLDF flag while using WWDT interrupts

- Description:

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

- Workaround:

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

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

- Revision plan:

  Revision C has fixed this issue.

1.12 WDT

1.12.1 Entering Standby mode immediate after enabling WDT will trigger a reset

- Description:

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

- Workaround:

  Insert a delay of a few us after entering WDT, and then enter Standby mode.

- Revision plan:

  Revision C has fixed this issue.

1.12.2 Entering Deepsleep mode immediately after enabling WDT causes WDT enable failure

- Description:

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

- Workaround:

  Insert around 30 us delay after enabling WDT, and then enter Deepsleep mode.

- Revision plan: Revision C has fixed this issue.

1.13 CRM

1.13.1 CLKOUT clock output exception after entering Deepsleep mode

- Description:

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

- Workaround:

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

- Revision plan: None.

1.13.2 PLL 2x or 3x multiplication factor failure

- 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.

- Revision plan: None.

1.14 I2C

1.14.1 I2C slave communication error 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 to 100kHz.

- Revision plan: None.

1.14.2 I2C communication error when BUSERR is detected on bus

- Description:

  When the following three conditions are present for I2C, a BUSERR condition will be detected, causing communication error.

  The three conditions are as follows:

  1. I2C is enabled
  2. Before the start of communication
  3. BUSERR timing is detected on bus

- Workaround:

  Prior to communication, first check if BUSERR flag is set or not. If set, clear it before starting communication.

  Besides, you can enable error interrupt so that it can be cleared after the BUSERR flag is set.

- Revision plan: None.

1.15 FLASH

1.15.1 SLib and boot memory AP mode settings

- Description:

  In order to ensure robustness and security of configurations, it is not recommended to use code to configure SLib and boot memory AP mode.

- Workaround:

  Use ICP or ISP to set SLib and boot memory AP mode.

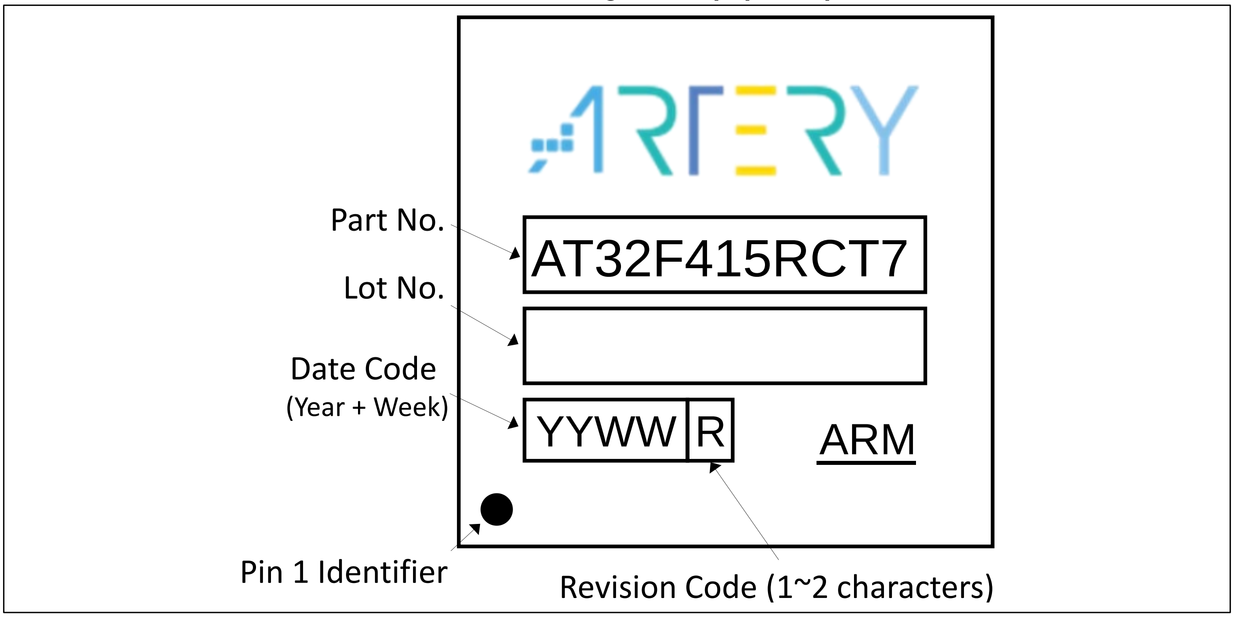
- Revision plan:

  None.

2 Revision code on device marking

Figure 1 shows the location of revision code on AT32F415 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 Revision history

Table 4. Document revision history

Date	Revision	Changes
2021.11.22	2.0.0	Initial release.
2022.01.20	2.0.1	1. Added *CLKOUT clock output exception after entering Deepsleep mode.* 2. Change BPR as battery powered battery domain.
2022.03.01	2.0.2	1. Added *Failed to filter RTR bit of standard frame in 32-bit identifier mask mode.* 2. Added *PLL 2x or 3x multiplication factor failure.*
2022.04.07	2.0.3	1. Added *CAN sends unexpected messages in case of narrow pulse disturbance on BS2.* 2. Added *3 to 6 LEXT clock cycles delay after each system reset when LEXT as ERTC clock source.*
2022.04.15	2.0.4	1. Added *The first received data is misaligned in I2S PCM standard long frame receive-only mode.* 2. Added *UDR flag is set mistakenly in I2S slave transmission mode and discontinuous communication state.* 3. Added *Data reception error when I2S 24-bit data is packed into 32-bit format.* 4. Added *CS failing edge was not synchronized in slave SPI hardware CS mode.*
2022.04.27	2.0.5	1. Added an example case in the *1.2.2 Failed to filter RTR bit of standard frame in 32-bit identifier mask mode* 2. Added an example case in the *1.7.5 SWEF flag is set when enabling a standby-mode wakeup pin* 3. Added *1.4.2 FT (5V tolerant pin) maintains at intermediate level in floating input mode* 4. Added *1.14.1 I2C slave communication error when APB equals or less than 4MHz*
2022.06.02	2.0.6	Updated the description of how to get the revision code of MCU.
2022.09.06	2.0.7	Added *section I2C communication error when BUSERR is detected on bus*
2022.09.27	2.0.8	Added *section Unable to clear and set ADC preempted channel conversion end flag*
2023.03.08	2.0.9	Added *Slave timer unable to receive reset signal from master timer*
2023.08.03	2.0.10	Added *1.9.6 Break input failed when TMREN=0 (TMR disabled)* Updated descriptions of *1.2.1 Bit stuffing error causes the next data out of order during CAN communication* Added *1.2.4 Fail to cancel mailbox transmit command when CAN bus disconnected*
2024.01.31	2.0.11	1.Added Section *1.9.7 Fail to generate CxORAW clear event when dead-time is disabled* 2.Added Section *1.15.1 SLib and boot memory AP mode settings*
2024.05.15	2.0.12	Updated descriptions in *Section 1.9.4 TMR accessing 0x4C address with*

		DMA causes DMA request error
2025.1.09	2.0.13	1. Updated "Method 3" descriptions in *Section 1.2.1 Bit stuffing error causes the next data out of order during CAN communication* 2. Updated descriptions in *Section 1.7.6 Unable to configure system clock after waking up Deepsleep mode* 3. Added *Section 1.7.8 VBAT powered domain register power-on reset failure*

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.

© 2025 Artery Technology -All rights reserved