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.

Table 1. Device summary

Device	Flash memory	Part number
AT32F435	4032 KB	AT32F435ZMT7, AT32F435VMT7, AT32F435RMT7, AT32F435CMT7, AT32F435CMU7
	1024 KB	AT32F435ZGT7, AT32F435VGT7, AT32F435RGT7, AT32F435CGT7, AT32F435CGU7
	256 KB	AT32F435ZCT7, AT32F435VCT7, AT32F435RCT7, AT32F435CCT7, AT32F435CCU7
AT32F437	4032 KB	AT32F437ZMT7, AT32F437VMT7, AT32F437RMT7
	1024 KB	AT32F437ZGT7, AT32F437VGT7, AT32F437RGT7
	256 KB	AT32F437ZCT7, AT32F437VCT7, AT32F437RCT7

Contents

List of tables

1 AT32F435/437 device limitations

Table 2 gives a list of limitations that have been identified so far on the AT32F435/437 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 Failed to filter RTR bit of standard frame in 32-bit identifier mask mode.
	1.1.3 CAN sends unexpected messages in case of narrow pulse disturbance on BS2.
1.2 DMAMUX	1.2.1 Setting EVTGEN bit for DMAMUX synchronization.
1.3 EDMA	1.3.1 Preemption priority between data streams failed in EDMA linked list mode
1.4 I2S	1.4.1 I2S communication failed when SPITI mode and 3-divided frequency are enabled simultaneously.
	1.4.2 First data received is misaligned in I2S PCM standard long frame receive-only mode.
	1.4.3 UDR flag is set in I2S slave transmission mode and in discontinuous communication state.
	1.4.4 Data reception error when I2S 24-bit data is packed into 32-bit format.
1.5 PWC	1.5.1 Unable to wakeup Deepsleep mode after AHB frequency division.
	1.5.2 Unable to select system clock source after waking up Deepsleep mode
	1.5.3 SWEF flag is set when enabling a standby-mode wakeup pin.
	1.5.4 Precautions on LDO use
1.6 SDRAM	1.6.1 SDRAM read error in burst read mode.
	1.6.2 SDRAM low-power mode limitations.
	1.6.3 SDRAM and other XMC static memory usage limitations.
1.7 SPI	1.7.1 SCK and Data synchronization failed in SPI slave hardware CS mode.
	1.7.2 CS pulse flag is set in SPI slave TI mode.
	1.7.3 CS failing edge not synchronized in SPI slave hardware CS mode.
1.8 QSPI	1.8.1 QSPI access error when QSPI is not initialized as an XIP port.
	1.8.2 Data threshold counter failed in QSPI XIP port D mode write configuration.
	1.8.3 QSPI QSPI Cache usage limitation
1.9 USART	1.9.1 USART ROERR flag is set mistakenly
1.10 ADVTM	1.10.1 How to clear TMR-triggered DAM requests.
	1.10. 2 TMR overrun in encoder mode counter.
1.11 ERTC	1.11.1 Writing ERTC occupies APB for 4 ERTC clock cyles

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, it will stop receiving the current data frame as expected and report an error to the bus, but the next data frame will be out of order.

- Workaround:

  Method 1:

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

  This method applies to the scenario where a quick CAN initialization is required in order to ensure a quick resume of CAN communication, avoiding too much CAN data loss.
  Take a CAN1 as an example, its typical code as follows:

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

  Notes:

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

  b) As it takes some time to finish CAN initialization, CAN's inability to resume communication immediately when an error occurs may cause loss of data.

Method 2:

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

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

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 met, follow the “Workaround” to solve this problem:

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

- Workaround:

  Method 1: By software. When filtering a standard frame in 32-bit wide identifier mask mode, the software is used to get the status of the RTR bit (remote frame identifier) and decide if 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.

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;

  …
}
```

1.2 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.

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.

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.

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

- Description:

  When PCLK frequency division factor is greater than 1 and I2S PCM standard long frame receive-only mode is enabled, if I2SCPOL = 0 is set and the SCK line remains high before enabling I2S, the first received data would be out of order.

- Workaround:

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

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

- Description:

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

- Workaround:

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

1.4.4 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.

1.5 PWC

1.5.1 Unable to wakeup Deepsleep mode after AHB frequency division

- Description:

  If AHB frequency division is configured, 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.

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.

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

- Description:

  Before 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, the IO then needs to re-initialized to be pull-down input or analog input before enabling the wakeup pin. For example:

```
gpio_init_type gpio_init_struct;

/* enable the 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);
```

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 five 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 second LDO configuration */
```

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.

1.6.2 SDRAM low-power mode limitations

- Description:

  If read/write access to SDRAM happened while the SDRAM is entering self-refresh or power-down mode, such read/write operation may not be executed.

- Workaround:

  Do not read/write from/to the 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.

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:

  Do not use SDRAM and other XMC static memories simultaneously.

  If it is needed, PSRAM or SRAM is recommended.

1.7 SPI

1.7.1 SCK and Data synchronization failed in SPI slave hardware CS mode

- Description:

  In SPI slave hardware CS mode, the synchronization between SCK and Data does not take place at each CS falling edge, affecting SPI's interference immunity.

- Workaround:

  Solution A: Strictly control the slave CS line, that is, the host must pull high the CS line as soon as the communication is complete.

  Solution B: Enable CRC check. Once a CRC error is detected, re-enable the slave SPI, and restart handshake communication.

1.7.2 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.

1.7.3 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.

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 or debug mode will get the program stuck.

- Workaround:

  Do not read QSPI addresses when the QSPI is not configured as an XIP port.

1.8.2 Data threshold counter failed in QSPI XIP port D mode write configuration

- Description 1:
  When the QSPI is initialized as an XIP port, and D mode write configuration is selected, if the value of the XIPW_DCNT [21:16] is greater than 0x1F, then during the period of wrting data to the consecutive addresses, if the number of data written reaches the maximum threshold defined in a single write operation, the command neither stops nor the CS signal becomes high, resulting in QSPI error.
- Workaround:
  The maximum value of the XIPW_DCNT[21:16] can only be configured as 0x1F (lower 5 bits are valid), and bit[21] must be 0.
- Description 2:
  When the QSPI is initialized as an XIP port, and D mode write configuration is selected, if the value of the XIPW_DCNT [21:16] is greater than 1, in this case, when the number of data written data is not equal to the programmed value (for instance, write to discontinuous addresses through several operations, the amount of data written is less than the programmed value), the XIPW_DCNT bit may fail, in the same way as XIPW_DCNT=1. The subsequent operations become less efficient, mismatching the programmed value.
- Workaround:
  Mode T is recommended. If mode D is needed, the resulting performance reduction should be taken into account.

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 this feature)

  This feature is suitable for the following scenarios only:

  1 XIP Read only (extend external Flash)

  2 XIP T mode (XIPR_SEL bit is set to 1)

  3 CSDLY value is equal to or above 6

- Workaround:

  XIP read only mode, for instance, connected to external Flash.
  Select T mode by setting the bit[0] XIPR_SEL to 1 in the XIP CMD_W2 register.
  Set the bit[3:0] CSDLY value to be 6 and above in the ACTR register

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 performing consecutive data transfer.
- 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.

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.

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 to deal with interrupts. Thus this method applies to the scenario where the external input frequency of the encoder is not so fast.
- Method 2:
  Turn to a TMR with enhanced mode (the counter can be extended from 16-bit to 32-bit width) in order to expand the encoder's counting range that detects forward and reverse rotation, and configure the initial value of the counter to PR/2 so as to prevent the timer from overflowing.
  This method has its limitation: The forward and reverse rotation of the encoder must be limited to a certain range. An overflow still occurs if the encoder were always rotated in one direction.
  This method applies to the scenario where the rotation of the encoder is controlled at a certain range.

1.11 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 APB to be occupied 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
2021.9.30	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 *Data threshold counter failed 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 received is misaligned in I2S PCM standard long frame receive-only mode.* 4. Added *UDR flag is set in I2S slave transmission mode and in 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 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*
2022.09.27	2.0.7	Added *1.11.1 Writing ERTC occupies APB for 4 ERTC clock cyles* Added *1.5.4 Precautions on LDO use*

IMPORTANT NOTICE – PLEASE READ CAREFULLY

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

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

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

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

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

© 2022 Artery Technology -All rights reserved