ES0012
勘误手册

AT32A403A使用限制

前言

该勘误表适用于雅特力科技的 AT32A403A 系列芯片。该芯片系列集成了 ARM™ 32 位 Cortex®-M4 内核。

表 1. 芯片概览

涉及到的芯片	闪存存储器	芯片型号
AT32A403A	1024 K字节	AT32A403AACGU7, AT32A403AACGT7, AT32A403AARGT7, AT32A403AAVGT7,
	512 K字节	AT32A403AACEU7, AT32A403AACET7, AT32A403AARET7, AT32A403AAVET7,
	256 K字节	AT32A403AACCU7, AT32A403AACCT7, AT32A403AARCT7, AT32A403AAVCT7,

目录

表目录

1 AT32A403A 芯片的使用限制

下表是所有已经发现的局限性概览：

表 2. 芯片局限性列表

章节	内容
1.1 CAN	1.1.1 CAN 通讯数据域期间出现位填充错误会导致下一帧数据错位问题
	1.1.2 32 位宽标识符掩码模式下无法有效过滤标准帧的 RTR 位
	1.1.3 CAN 在有窄脉冲干扰 BS2 段的条件下有概率会发送非预期的报文
	1.1.4 CAN 总线在人为或异常断开后执行邮箱的取消发送命令无效
1.2 I2C	1.2.1 在 APB 时钟小于等于 4MHz 时，I2C 做从机无法在 400kHz 的速度下通讯
	1.2.2 I2C 在通信开始前，当总线上出现 BUSERR 时序，会误检测到 BUSERR
1.3 I2S	1.3.1 I2S 从发模式非连续通讯状态下误置位 UDR 标志问题
	1.3.2 I2S 24 位数据封装成 32 位帧格式接收异常问题
1.4 PWC	1.4.1 AHB 分频后 DEEPSLEEP 模式无法被唤醒
	1.4.2 Systick 中断误唤醒 DEEPSLEEP
1.5 CRM	1.5.1 进入 DEEPSLEEP 模式后 CLKOUT 可能有时钟输出问题
1.6 TMR	1.6.1 TMR 在编码器模式下的溢出事件
	1.6.2 未使能定时器时（TMREN = 0），刹车输入无效
1.7 RTC	1.7.1 设置 RTC 计数值时，实际值可能是设置值+1
1.8 FLASH	1.8.1 sLib 安全库区配置在非零等待区时，特殊情况下可能导致程序出错
	1.8.2 擦除零等待区域时，擦除期间执行代码可能导致程序出错
	1.8.3 擦除 SPIM 时，擦除期间出现 cpu 读 flash 操作可能导致程序出错
1.9 SPI	1.9.1 SPI 从机硬件 CS 模式下 CS 下降沿不会做重同步
1.10 EXINT	1.10.1 EXINT line 一次软件触发会产生两次中断响应

1.1 CAN

1.1.1 CAN 通讯数据域期间出现位填充错误会导致下一帧数据错位问题

- 问题描述：

  如果CAN因为外部干扰导致通讯的数据域期间出现位填充错误，此时会按照期望停止接收当前帧数据并回馈错误到总线上，但是下一帧通讯报文会出现数据错序，且随后的报文又自动恢复正常的现象。

- 解决方法：

  方法1：开启CAN的错误类型记录中断号对应的错误中断（中断优先级需设定为最高），在CAN错误类型记录中断的中断函数内检测到出现位填充错误时，复位CAN（可只复位CAN寄存器，其相关的GPIO等、NVIC不需复位），并在CAN错误中断函数内完成CAN的重新初始化。

  此方法适用于期望快速完成CAN的初始化，以保障CAN及时参与通讯，避免过多CAN数据丢失的场景。

  以CAN1为例，其典型示例代码如下：

```
/* 开启 CAN 的上次错误中断号对应的错误中断并设定中断最高优先级 */
  nvic_irq_enable(CAN1_SE_IRQn, 0x00, 0x00);

  can_interrupt_enable(CAN1, CAN_ETRIEN_INT, TRUE);
```

```
  can_interrupt_enable(CAN1, CAN_EOIEN_INT, TRUE);
/* 中断服务函数 */
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);
      /* 调用CAN初始化函数 */
    }
  }
}
```

注意事项

a) CAN错误类型记录中断的优先级需设定为最高；

b) 由于CAN初始化存在耗时，出现问题后CAN不能及时恢复参与通讯，因此存在丢数据的现象。

方法2：开启CAN的错误类型记录中断号对应的错误中断（中断优先级需设定为最高），在CAN错误类型记录中断的中断函数内检测到出现位填充错误时，复位CAN（可只复位CAN寄存器，其相关的GPIO等、NVIC不需复位），及记录复位事件，并通过在其他低优先级中断或main函数内重新进行CAN初始化。

此方法适用于可接受CAN不能及时参与通讯，需严格保障CAN的重新初始化不影响其他应用逻辑的实现场景。

以CAN1为例，其典型示例代码如下：

```
/* 开启 CAN 的上次错误中断号对应的错误中断并设定中断最高优先级 */
  nvic_irq_enable(CAN1_SE_IRQn, 0x00, 0x00);
  can_interrupt_enable(CAN1, CAN_ETRIEN_INT, TRUE);
  can_interrupt_enable(CAN1, CAN_EOIEN_INT, TRUE);
/* 中断服务函数 */
__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;
      }
    }
}
```

应用再在期望的地方（例如main函数内）查询can_reset_index是否置位，置位后调用CAN初始化函数。

注意事项

a) CAN错误类型记录中断的优先级需设定为最高;

b) 由于CAN初始化及其他应用中断存在耗时，出现问题后CAN不能及时恢复参与通讯，因此存在丢数据的现象。

方法3：开启CAN的错误类型记录中断号对应的错误中断（中断优先级需设定为最高），在CAN错误类型记录中断的中断函数内检测到出现位填充错误时，强制发送一帧标识符优先级最高的无效报文。

此方法适用于不期望消耗时间去复位CAN，CAN总线上的所有报文标识符均为已知，且CAN各个节点有严格按照标识符过滤条件来接收报文的实现场景。

以CAN1为例，其典型示例代码如下：

```
/* 强制发送一帧标识符优先级最高的无效报文 */
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);
```

```
}
/* 开启 CAN 的上次错误中断号对应的错误中断并设定中断最高优先级 */
  nvic_irq_enable(CAN1_SE_IRQn, 0x00, 0x00);
  can_interrupt_enable(CAN1, CAN_ETRIEN_INT, TRUE);
  can_interrupt_enable(CAN1, CAN_EOIEN_INT, TRUE);
/* 中断服务函数 */
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();
    }
  }
}
```

注意事项

a) CAN错误类型记录中断的优先级需设定为最高;

b) 此方法仅适用于发送FIFO优先级由报文标识符决定的场景；

c) 此方法无效报文的标识符可修改，但一定要确保其标识符的优先级是CAN总线上最高的，且不会被其他节点当做正常报文接收。

1.1.2 32 位宽标识符掩码模式下无法有效过滤标准帧的 RTR 位

- 问题描述：

  当CAN的过滤器模式设置为32位宽标识符掩码模式时，在标准帧的过滤过程中，RTR位（即远程帧标识符）无法被有效过滤。

  即同时满足如下使用条件时，需要采用解决方法描述进行处理：

  1. 过滤器模式选用32位宽标识符掩码模式
  2. 进行标准帧过滤且不期望接收符合过滤条件的远程帧

- 解决方法：

  方法1：使用软件补充RTR位的过滤。即32位宽标识符掩码模式下过滤标准帧时，使用软件判断RTR位（远程帧标识符）的状态，来决定该帧报文是否被应用需要。参考示例：

```
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 */
    }
  }
}
```

方法2：更换使用其他过滤模式。结合实际应用需求，使用其他过滤模式来替代，比如32位宽标识符列表模式、16位宽标识符掩码模式或16位宽标识符列表模式。

1.1.3 CAN 在有窄脉冲干扰 BS2 段的条件下有概率会发送非预期的报文

- 问题描述：

  当CAN总线上有大量的窄脉冲干扰（脉冲宽度小于1tq），CAN节点发送时有一定概率发出非预期报文的情况，例如将数据帧发送成远程帧，将标准帧发送成扩展帧，或数据段出现错误。

- 解决方法：

  设置同步跳跃宽度RSAW = BTS2段宽度，以避免出现发出非预期错误的现象。

  需要注意的是，在设置RSAW = BTS2后，在CAN总线有大量干扰的情况下，CAN总线的通信效率会降低。

```
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;

  …
}
```

1.1.4 CAN 总线在人为或异常断开后执行邮箱的取消发送命令无效

- 问题描述：

  CAN作为报文发送节点，若同时满足如下条件，则在CAN错误被动中断内执行邮箱的取消发送命令将会无效，使得断开CAN总线时刻邮箱内的待发报文的发送并未被实际取消，其会在等待

后续CAN总线恢复后重新发送出来。

1. 人为或异常断开CAN总线（CANH/L）
2. 自动重传功能有开启

- 解决方法：

使能CAN的错误被动中断，在其中断函数内关闭自动重传，并在报文发送函数内重新开启自动重传。代码实现如下

1） 在CAN的初始化时使能错误被动中断

```
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） 在CAN的错误被动中断内关闭自动重传功能

```
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） 在CAN的报文发送函数内重新开启自动重传功能

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

1.2 I2C

1.2.1 在 APB 时钟小于等于 4MHz 时，I2C 做从机无法在 400kHz 的速度下通讯

- 描述：

在APB时钟小于等于4MHz时，I2C做从机无法在400kHz的速度下通讯。

- 解决方法：

APB时钟增加到8MHz或者I2C速度降到100kHz运行。

1.2.2 I2C 在通信开始前，当总线上出现 BUSERR 时序，会误检测到 BUSERR

- 描述：

I2C同时满足下列条件时，会检测到BUSERR条件，导致不能正常通信

条件1　I2C使能

条件2　通信开始前

条件3　总线上出现BUSERR时序

- 解决方法：

通讯开始前检查BUSERR标志是否置起，如果置起，清除标志后便可正常通讯。

也可以开启错误中断，当BUSERR标志置起后在中断里清除。

1.3 I2S

1.3.1 I2S 从发模式非连续通讯状态下误置位 UDR 标志问题

- 问题描述：

  I2S从发模式，不连续通讯时，虽在通讯起始前有写入待发数据，但还是会异常置位UDR标志。

- 解决方法：

  结合协议特点，I2S 从发模式建议使用 DMA 或中断等高效的数据传输方式，保障通讯连续。

1.3.2 I2S 24 位数据封装成 32 位帧格式接收异常问题

- 问题描述：

  I2S在24位数据封装成32位帧格式时，8个无效CLK对应的数据会被接收方当做正常数据接收。

- 解决方法：

  解法一：收发双方采用相同的 24 位数据封装成 32 位帧格式的方式；

  解法二：采用软件处理，在此帧格式条件下，丢弃 8 个无效 CLK 对应的数据。

1.4 PWC

1.4.1 AHB 分频后 DEEPSLEEP 模式无法被唤醒

- 问题描述：

  如果将AHB做分频配置后，任何唤醒源唤醒DEEPSLEEP模式都会存在无法唤醒的情况。

- 解决方法：

  使用DEEPSLEEP模式时，不能对AHB进行分频。

  即进DEEPSLEEP模式前，将AHB分频修改为不分频，唤醒后再按照期望设定AHB的分频。

1.4.2 Systick 中断误唤醒 DEEPSLEEP

- 问题描述：

  若进DEEPSLEEP前未关闭Systick或Systick中断时，进DEEPSLEEP后Systick将保持运行，且随后产生的Systick中断会唤醒DEEPSLEEP。

- 解决方法：

  进DEEPSLEEP前关闭Systick或Systick中断。

1.5 CRM

1.5.1 进入 DEEPSLEEP 模式后 CLKOUT 可能有时钟输出问题

- 问题描述：

  当DEEPSLEEP_DEBUG位设置为0,CLKOUT配置输出系统时钟，在进入DEEPSLEEP模式后,CLKOUT脚上还会有时钟输出,频率为LICK时钟的频率。

- 解决方法：

  在进入DEEPSLEEP模式前，将CLKOUT的时钟输出配置为NOCLK，等退出DEEPSLEEP模式后再配置为输出系统时钟。

1.6 TMR

1.6.1 TMR 在编码器模式下的溢出事件

- 问题描述：

  在编码器模式计数时，如Counter是在0和PR之间来回计数，此上溢或下溢时TMR的OVFIF事件不会置位。

- 解决方法：

  方法1：需占用当前使用编码器TMR的C3IF、C4IF通道为输出模式，设C3DT = AR、C4DT = 0，并使能C3IF、C4IF中断。

  在中断中判断“C3IF事件 & 向下计数”，则此时发生了“下溢”；

  在中断中判断“C4IF事件 & 向上计数”，则此时发生了“上溢”；

  注此解法有限制：如编码器计数的输入信号频率太快，需反复进中断并由软件处理，可能会来不及处理。可应用于编码器的外部输入信号频率不太快的情况。

  方法2：换用有增强模式的定时器(Counter能由16bit扩展为32bit的位宽来计数)，以增加编码器检测正反转的计数范围，将Counter的初值设为PR/2，不让定时器产生溢出。

  注此解法有限制：编码器计数的正反转，只能在一定范围内去转动，如总往一个方向转也会产生溢出。可应用于编码器检测的正反转是在一定范围内去转动的情况。

1.6.2 未使能定时器时（TMREN = 0），刹车输入无效

- 问题描述：

  未使能定时器时（TMREN = 0），刹车输入无效，从而导致刹车输入无法触发刹车事件或中断。

  例如：当使用单周期模式时，一个周期的计数完成后硬件会自动将TMREN清0，此时刹车输入由于上述原因将被屏蔽，从而导致输出使能位（OEN）无法被清零、刹车标志无法置位。

- 解决方法：

  无。

1.7 RTC

1.7.1 设置 RTC 计数值时，实际值可能是设置值+1

- 描述：

  设置RTC计数值时，实际值可能是设置值+1。

- 解决方法：

  设置RTC计数值时，先设置分频值。

```
rtc_wait_config_finish();
rtc_divider_set(32767);

rtc_wait_config_finish();
rtc_counter_set(100);
```

1.8 FLASH

1.8.1 sLib 安全库区配置在非零等待区时，特殊情况下可能导致程序出错

- 问题描述：

  当sLib设置在非零等待区，程序运行中出现cpu访问i-code(or d-code)，并快速切bus id的行为，由于内部访问flash区域需要时间，所以可能导致使用切换后的bus id访问sLib，判断bus id不匹配导致回复错误的指令(or 数据)给cpu，最终导致程序出错。

- 解决方法：

  将sLib区域配置在零等待区（ZW）。

1.8.2 擦除零等待区域时，擦除期间执行代码可能导致程序出错

- 问题描述：

  当擦除的是零等待区，在擦除期间程序可以继续执行，如果程序有零等待区取指，然后又有非零等待区的取指，则会读出错误的非零等待区数据，最终导致程序出错。

  举例：擦除零等待区期间中断可以响应，如果中断处理函数调用过程中包含非零等待区和零等待区的取指操作，那么可能导致程序出错

- 解决方法：

  原理是在擦除零等待区域时，需保证擦除期间的所有执行代码全都位于同一区域（全在零等待区或者全在非零等待区）。

  为实现上述原理：可在擦除前关闭中断使能，擦除完成再打开中断使能，并且保证擦除函数相关代码编译到同一区域。

1.8.3 擦除 SPIM 时，擦除期间出现 cpu 读 flash 操作可能导致程序出错

- 问题描述：

  当擦除SPIM期间有cpu read flash情况，会导致该笔read指令被误判断为读SPIM，数据读错，最终导致程序出错。

  举例：擦除SPIM函数本身编译在非零等待区，擦除时该函数本身执行就会从非零等待区取指，即有read flash操作，那么可能导致程序出错

- 解决方法：

  原理是在擦除SPIM时，需保证擦除期间的所有执行代码不能有read flash的行为。

  为实现上述原理：可在擦除前关闭中断使能，擦除完成再打开中断使能，并且保证擦除函数相关代码编译到零等待区或者RAM。

1.9 SPI

1.9.1 SPI 从机硬件 CS 模式下 CS 下降沿不会做重同步

- 问题描述：

  SPI 从机硬件 CS 模式下，不会在每个 CS 下降沿进行数据传输的起始 CLK 同步。

- 解决方法：

  解法一：严格约束从机CS线，在通讯完毕后及时拉高CS；

  解法二：开启CRC校验，当检测到CRC校验错误时，复位SPI并重新进行握手通讯。

1.10 EXINT

1.10.1 EXINT line 一次软件触发会产生两次中断响应

- 问题描述：

  当使用 EXINT line 的软件触发功能时，会在每个软件触发命令执行后立即产生两次该 EXINT 中断线的中断响应。

- 解决方法：

  软件触发对应的中断函数内，将EXINT标志清除命令连续执行两次。

  可使用最新版BSP库的标志清除函数（如下），或参考此方法自行修改代码。

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

2 版本历史

表 3. 文档版本历史

日期	版本	变更
2023.08.03	2.0.0	最初版本

重要通知 – 请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和 / 或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途（及其依据任何司法管辖区的法律的对应情况），或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力产品并非设计或专门用于下列用途的产品：（A）对安全性有特别要求的应用，例如：生命支持、主动植入设备或对产品功能安全有要求的系统；（B）航空应用；（C）航天应用或航天环境； （D）武器，且/或（E）其他可能导致人身伤害、死亡及财产损害的应用。如果采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，风险及法律责任仍将由采购商单独承担，且采购商应独力负责在前述应用中满足所有法律和法规要求。

经销的雅特力产品如有不同于本文档中提出的声明和 / 或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。