FAQ0116

Frequently Asked Questions

CAN receive data stuffing error

Questions1: CAN receive data stuffing error

While using CAN, if an exception/error (this is mainly due to sampling point drift or external interference) is detected on CAN bus during the period of receiving data field, the receive data bit stuffing error may happen and this causes a whole of frame out of order, even though the frame can automatically resume order later on.

This problem can be fixed through software intervention. Here there are three solutions based on the actual application scenarios.

Answer1：

Enable CAN error interrupt (it must be given top priority) corresponding to the previously-reported error type. Once a bit stuffing error is detected inside CAN error interrupt function, reset CAN (just need reset CAN registers and related GPIOs, excluding NVIC) and initialize it.

This method is used in a scenario where users want to initialize CAN as soon as possible so as to ensure CAN's quick communication and prevent loss of too much CAN data.

See the following typical code based on CAN1：

```
/*Enable CAN error interrupt corresponding to the last-reported error type and give it top priority*/
can_interrupt_enable(CAN1, CAN_ETRIEN_INT, TRUE);
can_interrupt_enable(CAN1, CAN_EOIEN_INT, TRUE);
nvic_irq_enable(CAN1_SE_IRQn, 0x00, 0x00);
```

Interrupt handling function:

```
/* can1 interrupt function se */
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) //<judge if this is a bit stuffing error
    {
      can_reset(CAN1);
     /*Call CAN initialization function can_configuration(); */
    }
  }
}
```

Cautions:

A. CAN error interrupt must be given a top priority

B. As it takes some time for CAN to initialization, the inability of CAN to resume consumption when an error occurs would cause loss of CAN data.

Answer2：

Enable CAN error interrupt (it must be given top priority) corresponding to the previously-reported error type. Once a bit stuffing error is detected inside CAN error interrupt function, reset CAN (just need reset CAN registers and related GPIOs, excluding NVIC) and initialize it within other low-priority interrupt or main function.

This method is used in a scenario where users tolerant CAN's inability to take part in communication in time, but requires it to be re-initialized so as to serve other applications.

See the following typical code based on CAN1：

```
/* Enable CAN error interrupt corresponding to the last-reported error type and give it top priority*/
can_interrupt_enable(CAN1, CAN_ETRIEN_INT, TRUE);
can_interrupt_enable(CAN1, CAN_EOIEN_INT, TRUE);
nvic_irq_enable(CAN1_SE_IRQn, 0x00, 0x00);
```

Interrupt handling function:

```
/* can1 interrupt function se */
__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) //<judge if this is a bit stuffing error
    {
      can_reset(CAN1);
      can_reset_index = 1;
    }
  }
}
```

The application checks whether the *can_reset_index* is set or not at the desired place (such as, main function), and call CAN initialization function (after *can_reset_index* is set)

Cautions:

A. CAN error interrupt must be given a top priority

B. As it takes some time for CAN to initialization, the inability of CAN to resume consumption when an error occurs would cause loss of CAN data.

Answer3：

Enable CAN error interrupt (it must be given top priority) corresponding to the previously-reported error type. Once a bit stuffing error is detected inside CAN error interrupt function, forcefully send an invalid message which has the very-high-priority identifier.

This method is used in a scenario where there is no need of spending time on resetting CAN, and all message identifiers on CAN bus have been known, as well as CAN nodes strictly follow identifier filtering rules in receiving messages.

See the following typical code based on CAN1：

```
/*Forcefully send an invalid message with very-high-priority identifiers*/
static void can_transmit_data(void)
{
  can_tx_message_type tx_message_struct;
```

```
    tx_message_struct.standard_id = 0x000;
    tx_message_struct.extended_id = 0;
    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-reported error type and give it top priority */
can_interrupt_enable(CAN1, CAN_ETRIEN_INT, TRUE);
can_interrupt_enable(CAN1, CAN_EOIEN_INT, TRUE);
nvic_irq_enable(CAN1_SE_IRQn, 0x00, 0x00);
```

Interrupt handling function:

```
/* can1 interrupt function se */
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) //< judge if this is a bit stuffing error
    {
      can_transmit_data();
    }
  }
}
```

Cautions:

A. This method can be used only in a scenario where transmit FIFO priority is determined by message identifiers;

B. This method is not applied to AT32F403;

C. In this method, the identifier of the invalid message is changeable, but it is necessary to ensure that it has the highest priority on CAN bus so that it is not received by other nodes as a normal message.

Questions2：CAN messages are transmitted repeatedly

When using CAN, if Hardware Automatic Retransmission feature is disabled, but users obtain auto retransmission through software polling transmit status, this may cause certain messages to be retransmitted more than one at a certain moment.

Answer：

The root cause of this issue is that CAN's "transmit complete flag" is not set with "transmit success or failure flag" simultaneously, meaning a PCLK interval in between them.

1. Enable Hardware Automatic Retransmission during CAN initialization
2. Use CAN driver after BSP 1.3.0
3. For older BSP, change driver files according to the following code:

```
uint8_t CAN_TransmitStatus(CAN_Type* CANx, uint8_t TransmitMailbox)
{
  uint32_t state = 0;
  switch (TransmitMailbox)
  {
case (CAN_TXMAILBOX_0):
      if((CANx->TSTS &  CAN_TSTS_RQC0) == CAN_TSTS_RQC0)
      {
        if((CANx->TSTS &  CAN_TSTS_TOK0) == CAN_TSTS_TOK0)
        {
          state = CAN_TxStatus_Ok;
        }
        else
        {
          state = CAN_TxStatus_Failed;
        }
      }
      else
      {
        state = CAN_TxStatus_Pending;
      }
      break;
case (CAN_TXMAILBOX_1):
      if((CANx->TSTS &  CAN_TSTS_RQC1) == CAN_TSTS_RQC1)
      {
        if((CANx->TSTS &  CAN_TSTS_TOK1) == CAN_TSTS_TOK1)
        {
          state = CAN_TxStatus_Ok;
        }
        else
        {
          state = CAN_TxStatus_Failed;
        }
      }
      else
      {
        state = CAN_TxStatus_Pending;
      }
      break;
```

```
    case (CAN_TXMAILBOX_2):
          if((CANx->TSTS &  CAN_TSTS_RQC2) == CAN_TSTS_RQC2)
          {
            if((CANx->TSTS &  CAN_TSTS_TOK2) == CAN_TSTS_TOK2)
            {
              state = CAN_TxStatus_Ok;
            }
            else
            {
              state = CAN_TxStatus_Failed;
            }
          }
          else
          {
            state = CAN_TxStatus_Pending;
          }
          break;
    default:
          state = CAN_TxStatus_Failed;
          break;
      }
      return (uint8_t) state;
    }
```

Type: MCU application
Applicable products: AT32F403, AT32F403A, AT32F407, AT32F413, AT32F415
Main function: CAN
Other function: None

Document revision history

Date	Revision	Changes
2022.3.2	2.0.0	Initial release