SIEMENS

Microcontrollers

ApNote AP1621

C167CR - CAN Interrupt Structure

The Controller Area Network (CAN) module which has been implemented in the Siemens C167CR microcontrollers allows communication between several stations (CAN nodes) in BASIC-CAN functionality and in FULL-CAN functionality according to CAN specification V2.0B active. This document describes the functionality of the CAN interrupt structure and gives an implementation example of a standard CAN interrupt subroutine.

Author : Dr. Jens Barrenscheen / HL MC PD Microcontroller Product Definition

Contents **Page**

AP1621 ApNote - Revision History		
Actual Revision : 12.96	Previous Revision : none (Original Version)	
Page of actual Rev.	Page of prev.Rel.	Subjects (changes since last release)

1 Interrupt Sources

As many different interrupt sources can generate only one global CAN interrupt request, their internal structure must be taken into account in the interrupt service procedure. The INTID code in the CAN Interrupt Register ($EF02_H$) indicates which source has activated the request. Bit IE in the CAN Control/Status Register ($EF00_H$) globally enables (IE=1) or disables all interrupt sources of the module.

The C167CR CAN controller provides three different types of interrupts:

1.1 Status Interrupts

– They are generated after a status change of the CAN module, which is indicated by the flags in the status part (high byte) of the CAN Control/Status Register. The reason for a status interrupt can be a successful transmission (TXOK is set) or reception (RXOK is set) of any message, or the occurrence of an error (LEC bitfield) during message transfer. These interrupt sources can be enabled by setting bit SIE in the CAN Control/Status Register.

1.2 Error Interrupts

– These interrupts, which can be enabled by bit EIE in the CAN Control/Status Register, are generated after each change of the flags EWRN or BOFF. These flags indicate the level of the two internal error counters (transmit error counter and receive error counter). If one of these counter reaches the value of 96, the error warning flag EWRN is set. If it exceeds the value of 255, the bus off flag BOFF is set. Furthermore, bit INIT is automatically set and the device stops all action on the CAN bus, it goes bus-off. A resynchronisation on the CAN bus can be achieved by resetting bit INIT by software.

1.3 Message Specific Interrupts

– Interrupts of this type are generated by each message object after successful transmission or reception. This function can be individually enabled by setting bits TXIE and/or RXIE in the corresponding CAN Message Control Register (MCR_Mn) located at address $EFn0_H$, with n (1..15) being the number of the corresponding message object.

2 Interrupt Identification by INTID Code

An INTID code of 0 indicates that all requested interrupts have been correctly serviced and no more interrupts are pending. This should be the condition to leave the CAN interrupt service routine.

An INTID value of 1 indicates a status interrupt (if enabled by SIE) or an error interrupt (if enabled by EIE), which cause the interrupt with the highest priority. In the case of a status change due to a successful message transfer, one of the flags TXOK or RXOK in the CAN Status Register is set. An erroneous message transfer is indicated by the LEC bit field. In the case of an error interrupt, at least one of the error flags EWRN and BOFF has changed.

An INTID code of 2 indicates the reception of a message by message object 15. This special case will be discussed later.

An INTID code of 3..16 indicates a message specific transmit (if TXIE set) or receive (if RXIE set) interrupt concerning the message objects 1 to 14 (INTID-2). In addition, the global flags TXOK and RXOK can be checked to decide if it was an interrupt on a received or on a transmitted message. A successful message transfer sets bit INTPND in the corresponding Message Control Register MCR_Mn, which must be cleared by software to reset this interrupt request.

3 General Interrupt Handling

The status part (high byte) of the CAN Control/Status Register must be read in the interrupt service procedure in order to identify the interrupt source and to reset the pending interrupt request. Flags TXOK and RXOK in this part of the register have then to be cleared by software.

The priority of the internal CAN interrupt sources decreases with an increasing INTID code. This structure must also be taken into account for the identification of the interrupt source. For example, a successful transmission of only one message object can cause two independent interrupt requests if bit SIE and the corresponding bit TXIE have been set. While the status interrupt (highest priority) is serviced and bit INTPND of this message object is not cleared, the message specific interrupt remains pending.

In order to ensure that all interrupts are correctly serviced, a standard CAN interrupt procedure for the C167CR should respect the following items. Each read operation of the status part (SR) of the CAN Control/Status Register (high byte, located at EF01H) may have an influence on the INTID value. To avoid errors due to this functionality, this register should be read only at the beginning of the interrupt routine and then be stored in a variable (here: status). The following actions, such as flag tests, only refer to this variable. The same structure can be used for the variable intid, which yields the value of the CAN Interrupt Register (IR). Furthermore, bit INTPND in the concerned Message Control Register (MCR_Mn) has to be reset in order to release the interrupt request if it has been activated by one of the message specific bits TXIE or RXIE.

```
//.........................................................
// CAN interrupt
void int_can (void) interrupt CANI
{unsigned char status, intid;
 while (intid = IR)
   {status = SR; SR = 0;                    // read and reset CAN status
    switch (intid)
    { case 1:                               // status and error interrupt
           if (status & 0x0004)             // status interrupts
            {if (status & 0x0800) {...}          // transmit interrupt
             if (status & 0x1000) {...}          // receive  interrupt
             if (status & 0x0700) {...}          // erroneous transfer
           }
           if (status & 0x0008)             // error interrupts
            {if (status & 0x4000) {...}          // EWRN has changed
             if (status & 0x8000) {...}          // BUSOFF has changed
           }
           break;
      case 2:                               // message 15 interrupt
           {...}                            // see special chapter
           break;
      case 3:                               // message 1 interrupt
           MCR_M1 = 0xFFFD;                      // reset INTPND
           if (status & 0x0800) {...}            // transmit interrupt
           if (status & 0x1000) {...}            // receive  interrupt
           break;
```

```
        case 4:                                 // message 2 interrupt
              MCR_M2 = 0xFFFD;                       // reset INTPND
              if (status & 0x0800) {...}             // transmit interrupt
              if (status & 0x1000) {...}             // receive  interrupt
              break;
        case 5:                                 // message 3 interrupt
              MCR_M3 = 0xFFFD;                       // reset INTPND
              if (status & 0x0800) {...}             // transmit interrupt
              if (status & 0x1000) {...}             // receive  interrupt
              break;
        ...
        case 16:                                // message 14 interrupt
              MCR_M14 = 0xFFFD;                      // reset INTPND
              if (status & 0x0800) {...}             // transmit interrupt
              if (status & 0x1000) {...}             // receive  interrupt
              break;
  }  }  }
```

4 Bus-Off and Re-Initialisation

Each transfer error causes the incrementation of one of the two error counters (receive error counter and transmit error counter) of the CAN module. If one these counters reaches the value of 96, bit EWRN is set. If enabled by bits EIE and IE, this action generates an error interrupt. The receive error counter stops up-counting when the device goes into the error passive state, even if more receive errors occur.

The transmit error counter is incremented on each error on transmitted messages. If this counter exceeds the value of 255, bit BOFF is set and an error interrupt can be generated, too. Furthermore, the device then goes into the bus-off state, sets bit INIT and stops all actions on the bus.

The re-initialisation on the CAN module to the CAN bus is started after resetting bit INIT by software. The device then goes into bus-off recovery and checks the logic level of the CAN input line CAN_RxD. A value of 5 is written in the LEC bitfield after monitoring a sequence of 11 recessive bits. After 128 of these sequences, the device is resynchronised on the CAN bus. During this procedure, the LEC code should not be interpreted as an error (same code as for Bit0Error). It indicates that the CAN device is successfully trying to synchronise on the CAN bus and that the bus is not continuously disturbed. If bits SIE and IE are set, a status interrupt is generated after each sequence.

If the user does not want to use this feature, it is possible to disable status interrupts during bus-off recovery by resetting bit SIE in the control part (CR) of the CAN Control/Status Register (low byte, located at EF00H). This can be done by the following instructions in the CAN interrupt routine:

```
 case 1:  ...                  // status and error interrupt
if (status & 0x0008)           // error interrupts
 {if (status & 0x4000)         // EWRN   is set
       {...}                        // other instructions
  if (status & 0x8000)         // BUSOFF is set = bus-off recovery
       {CR = 0x0A;                  // reset SIE, set EIE and IE
        ...}                        // other instructions
  else {CR = 0x0E;                  // set SIE, EIE and IE
        ...}                        // other instructions
 }
break;
```

5 Message 15 Interrupt

The internal structure of message object 15 has been adapted to provide a BASIC-CAN feature. In this mode, all incoming messages with identifiers, which are not matching to other message objects, can be stored in object 15. If no other objects have been activated, all messages can be treated by message object 15. The CPU has to check whether an incoming message addresses the controller (acceptance filtering) or should be ignored. As a consequence, the required CPU time to deal with incoming data increases, but all messages (data or remote frames, standard or extended identifiers) can be handled.

A special double buffer system has been implemented in order to avoid data losses in the case of two messages arriving one just after the other. While the CPU is working on one of these two buffers, the next incoming message is stored in the second one, see figure 1. This double buffer structure concerns data bytes as well as bits in the Message Control Register 15 (MCR_M15)

This structure must be taken into account if a message specific receive interrupt should be generated. Therefore, bit RXIE in Message Control Register 15 has to be set. Note: As this object can only be used as receive object, bit TXIE has no influence on the functionality.

If the CPU needs to work on the buffer containing the latest incoming message, the buffer which has been accessed by the CPU before must be released. This can be done by resetting bits NEWDAT and RMTPND by writing the value of 55DFH into Message Control Register 15. The next CPU access to this message object will now address the buffer containing the latest incoming message and the following CAN message will be stored in the second buffer. Bit INTPND of the buffer currently used by the CPU must then be reset in order to be able to correctly quit the interrupt service routine.

The corresponding part in the interrupt service routine can be written in the following way:

```
case 2:                                    // message 15 interrupt
       {MCR_M15 = 0x55DFH                     // release buffer (old)
        MCR_M15 = 0xFFFDH                     // reset INTPND   (new)
        ...}                                  // other instructions
       break;
```

6 Summary

The Siemens C167CR microcontroller with on-chip CAN module provides all features of a FULL-CAN controller combined with the BASIC-CAN option on message object 15. Thanks to the multitude of different CAN interrupt sources, all information which are necessary for normal data transfer and error handling are directly available. For these reasons, the C167CR can be easily used for data transfer and treatment in CAN bus systems. All program sequences given in this paper are examples, which are showing an efficient way how to use the powerful interrupt structure of this CAN module.

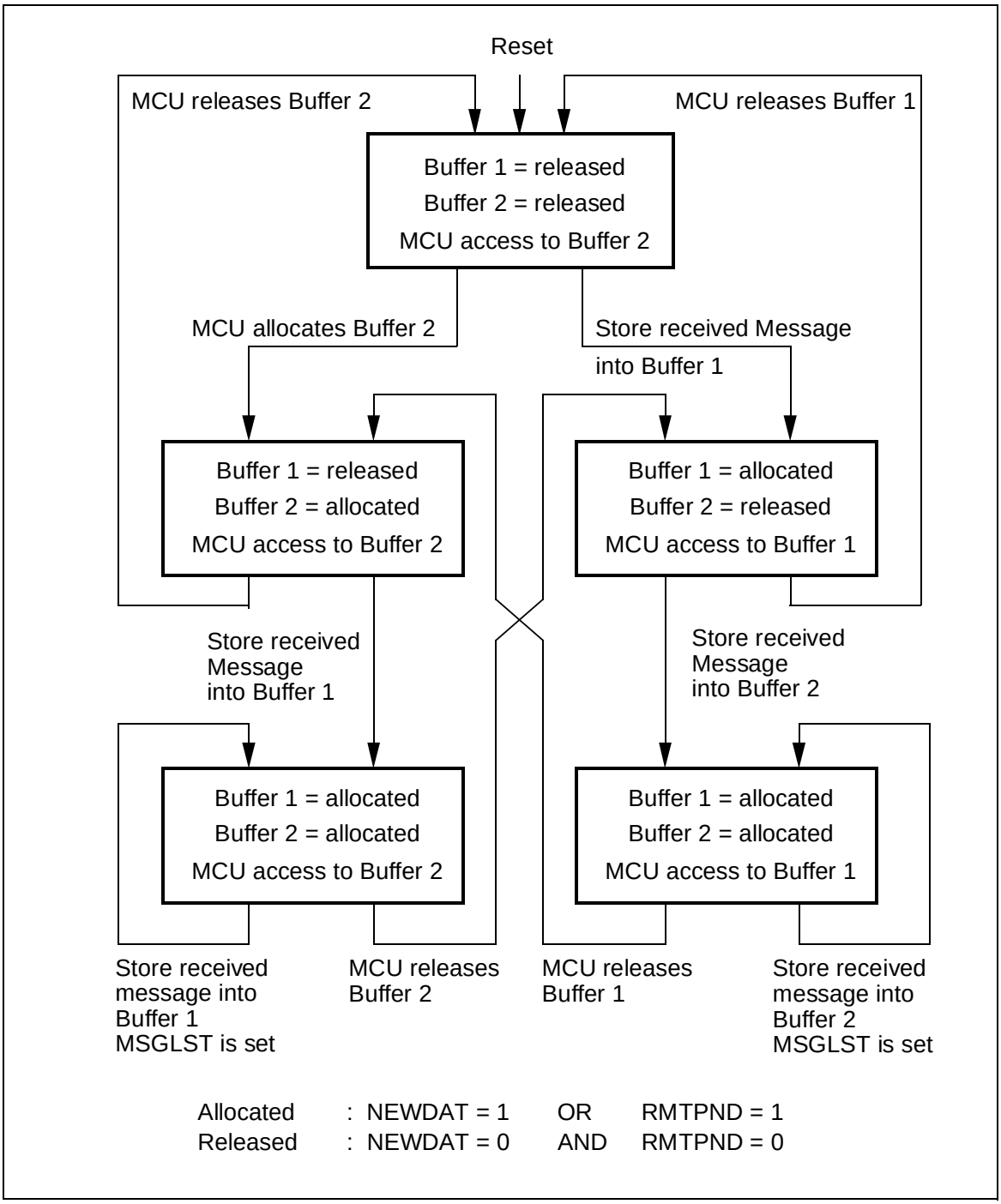

Figure 1 :
CPU Handling of the Last Message Object's Alternating Buffer