# CAN Library for the C166 family Reference Manual

$Id$

Generated by Doxygen 1.3

Fri Jun 6 10:11:17 2003

# Contents

# Chapter 1

# CAN Driver library for the C166 family

## 1.1 Introduction

This library provides a more convienient API to the CAN controller of the C166 family of microcontrollers than the *Special Function Registers* (SFRs) themselves. The library is based on the Siemens/Infineon Application Note AP2922. It was extended and modified in a few ways to further enhance convienience and fully support the publisher/subscriber library developed at the University of Ulm.

## 1.2 API

The API consists of a total of 13 functions, and one typedef'd struct:

1. init_can_16x()
2. def_mo_16x()
3. undef_mo_16x()
4. ld_modata_16x()
5. send_mo_16x()
6. rd_modata_16x()
7. rd_mo15_16x()
8. check_mo_16x()
9. check_mo15_16x()
10. check_busoff_16x()

11. set_global_mask()
12. can_write()
13. can_read()
14. canmsg

## 1.3 Usage

### 1.3.1 the CAN module

To use the CAN controller on the C166, together with this library, you must make sure that access to the on-chip CAN and XRAM is enabled with your compiler. Also check any startup-code that is involved, so that it enables the CAN module and don't forget to set the XPEN bit in the SYSCON register, as this also controls the CAN controller for newer versions of the C167CR (steppings ES-GA, GA, GA-T, GA-T 6, ES-JA and later; see also Infineon Errata Sheet for the C167CR, Release 1.2, June 18, 2001) and may not be documented in your version of the C167 manual.

For a quick-start with the KEIL Compiler suite, make sure that in the "Options for Target XXX" dialog, you have the following settings:

- Tab Target: set the Memory Model to HLarge, and check the Use On-Chip CAN+XRAM checkbox. If using a Phytec MM167 module, enter for External Memory locations #1 type ROM, start address 0x00000, size 0x40000, and for #2 type RAM, start address 0x40000, size 0x40000.
- Tab Output: select Create Library and set the output filename to canlib.lib if you want to (re)create the library.

If you want to use the (pre-compiled) CAN library with the KEIL suite, select "Add files to Group", and look for the library file (default is canlib.lib). Then look for CANR_16X.H and add that one too; it contains all the prototypes of the CAN library.

In your application project, make sure, that you set the Target Options as described above. They must match for all libraries used, and the application project itself. For the application, you also must edit the assembly startup file, for me, it's called START167.A66, and was just magically there on the PC. If you need it, contact me. In that file, you need to change the following definitions:

- _XPEN EQU 1 to enable the CAN module
- $SET (BUSCON1 = 1)
- %DEFINE (ADDRESS1) (040000H)
- %DEFINE (RANGE1) (256K)
- $SET (BUSCON4 = 1)
- %DEFINE (ADDRESS4) (100000H)
- %DEFINE (RANGE4) (4K)

### 1.3.2 the CAN module

To initialize the CAN module, call the init_can_16x() function. It will reset the CAN module, and configure it to your needs.

Before sending or receiving any message through one of the 15 so-called *message objects* of the C166 CAN module, these message objects must be ”defined” (configured). Use def_mo_16x() for that.

Before sending a CAN message with payload, you need to load the payload into the corresponding message object. This can be done using ld_modata_16x().

Sending the message is finally triggered by calling send_mo_16x().

For reception of CAN messages, you can check a message object by calling check_mo_16x() (for message objects 1 through 14) or check_mo15_16x() (for message object 15). If the return value of these messages is ”true”, then a new message arrived and can be read using read_modata_16x() and rd_mo15_16x(), respectively.

The function check_busoff_16x() can be used for some basic error checking and recovery in case of a *busoff* state.

After initialization, the *global mask* for message objects 1 through 14 is set in a way that for a successful reception of a message, all identifier bits of the message must match the repective identifier bits of the message object. Message object 15, on the other hand, is configured to be able receive all messages, regardless of their identifier, unless they already could be matched to a properly configured message object 1..14. To change this behaviour for the message objects 1..14, use set_global_mask().

If you are done using a configured message object and don’t want to be bothered by it anymore, use undef_mo_16x() to disable it.

Finally, can_write() and can_read() give you some nice wrappers for some of the above functions. As one of their parameters, they take a pointer to a canmsg structure, holding a complete can message. can_write() always uses message object 1 to send CAN messages. can_read() can be used on any of the message objects 2..14. These functions are mainly meant to be used by the publisher/subscriber library, but can save some typing in other cases, too ;).

### 1.3.3 Hints

Do not access the CAN controller by any other means than this library. Better yet, do not use this library directly, but use the publisher/subscriber library. It’s even more convienient to use, and you don’t need to bother with a lot of low-level things.

This library doesn’t support the interrupt generation feature of the CAN controller for now. If you enable interrupts, you need to write your own interrupt service routines.

# Chapter 2

# CAN Library for the C166 family Module Index

## 2.1 CAN Library for the C166 family Modules

Here is a list of all modules:

# Chapter 3

# CAN Library for the C166 family Data Structure Index

## 3.1 CAN Library for the C166 family Data Structures

Here are the data structures with brief descriptions:

# Chapter 4

# CAN Library for the C166 family File Index

## 4.1 CAN Library for the C166 family File List

Here is a list of all documented files with brief descriptions:

# Chapter 5

# CAN Library for the C166 family Module Documentation

## 5.1 CAN registers

definitions of CAN module control registers

### Defines

- #define CR ∗(unsigned char∗) 0xef00

  *The Command register.*

- #define SR ∗(unsigned char∗) 0xef01

  *The Status register.*

- #define IR ∗(unsigned char∗) 0xef02

  *The Interrupt register.*

- #define BTR ∗(unsigned int ∗) 0xef04

  *The baudrate register.*

- #define GMS ∗(unsigned int ∗) 0xef06

  *The global short mask (11 Bit IDs).*

- #define UGML ∗(unsigned int ∗) 0xef08

  *The upper half of the global long mask (29 Bit IDs).*

- #define LGML ∗(unsigned int ∗) 0xef0a

  *The lower half of the global long mask (29 Bit IDs).*

- #define UMLM ∗(unsigned int ∗) 0xef0c

*The upper half of the long mask for the last message.*

- #define LMLM ∗(unsigned int ∗) 0xef0e

  *The lower half of the long mask for the last message.*

### 5.1.1 Detailed Description

definitions of CAN module control registers

The C166 CAN module has 9 SFRs dedicated to it. They are used to control it, set the baudrate, set message masks, and other things.

## 5.2 Constants for defining Message Objects

these defines can be used to make the definition of message objects more readable

### Defines

- #define USE_XTID 1

  *Use the extended frame format with 29 bit identifiers.*

- #define USE_STID 0

  *Use the standard frame format with 11 bit identifiers.*

- #define DIR_XMIT 1

  *Make the message object a transmit object.*

- #define DIR_RECV 0

  *Make the message object a receive object.*

- #define NO_TX_INT 0

  *Disable transmit interrupts.*

- #define NO_RX_INT 0

  *Disable receive interrupts.*

- #define TX_INT 1

  *Enable transmit interrupts.*

- #define RX_INT 1

  *Enable receive interrupts.*

### 5.2.1 Detailed Description

these defines can be used to make the definition of message objects more readable

## 5.3 CAN controller API functions

### Data Structures

- struct canmsg

  *Structure for a can message.*

### Functions

- void init_can_16x (unsigned int baud_rate, unsigned char eie, unsigned char sie, unsigned char ie)

  *Initialize the CAN module.*

- void send_mo_16x (unsigned char nr)

  *Send a message object.*

- void rd_modata_16x (unsigned char nr, unsigned char ∗downl_data_ptr, unsigned long ∗mo_id_ptr, unsigned char ∗mo_dlc_ptr)

  *Read data from a message object.*

- void rd_mo15_16x (unsigned char ∗mo15_db_ptr, unsigned long ∗mo15_id_ptr, unsigned char ∗mo15_dlc_ptr)

  *Read data from the last message object.*

- void ld_modata_16x (unsigned char nr, unsigned char ∗upl_data_ptr)

  *Load data into a message object.*

- void def_mo_16x (unsigned char nr, unsigned char xtd, unsigned long id, unsigned char dir, unsigned char dlc, unsigned char txie, unsigned char rxie)

  *Define a message object.*

- void undef_mo_16x (unsigned char nr)

  *Clear a message object.*

- unsigned char check_mo_16x (unsigned char nr)

  *Check a message object for freshly received data.*

- unsigned char check_mo15_16x (void)

  *Check the last message object for new data.*

- unsigned char check_busoff_16x (void)

  *Checks for a busoff state of the controller.*

- void set_global_mask (unsigned long mask)

  *Sets the global reception mask.*

- void can_write (canmsg ∗cm)

  *Send a can message.*

- int can_read (int nr, canmsg ∗cm)

  *Try reading a can message.*

### 5.3.1 Function Documentation

#### 5.3.1.1 int can_read (int *nr*, canmsg ∗ *cm*)

Try reading a can message.

This function checks the specified message object for new data.

**Parameters:**

***nr*** number of the message object to read from

∗***cm*** pointer to a canmsg struct that will hold the received data.

**Returns:**

new data indication

**Return values:**

***0*** There is no new data, ∗cm is not touched.

>***0*** There is new data. The message had at least 1 byte of payload. The data (and status info like the message id) is copied to ∗cm, and the length of the message is returned.

***-1*** There is new data. The message had no payload. The data (and status info) is copied to ∗cm, but

Definition at line 66 of file mask.c.

References check_mo15_16x(), check_mo_16x(), canmsg::d, canmsg::id, canmsg::len, rd_mo15_16x(), and rd_modata_16x().

```
{
        unsigned char len;
        unsigned long id;

        // check for the last object
        if( nr < 15 ) {
                // new data?
                if( !check_mo_16x( nr ) )
                        return 0;
                // yes, get it.
                rd_modata_16x( nr, cm->d, &id, &len );
                cm->id = id;
        } else {
                // new data?
                if( !check_mo15_16x() )
```

```
                        return 0;
                // yes, get it.
                rd_mo15_16x( cm->d, &id, &len );
                cm->id = id;
        }
        // generate a cool return value ;)
        cm->len = (len)?((int)len):-1;
        return cm->len;
}
```

#### 5.3.1.2 void can_write (canmsg * *cm*)

Send a can message.

This function handles everything from (re-)defining the first message object to loading the data and transmitting it over the CAN bus. It is used by the publisher/subscriber library. Can also be very useful if you ”just want to send a message” ;)

**Parameters:**

*∗cm* pointer to a canmsg struct holding the message to be sent.

**Warning:**

Always uses the first message object. Do not use that one otherwise or do not use this function.

Definition at line 39 of file mask.c.

References canmsg::d, def_mo_16x(), DIR_XMIT, canmsg::id, ld_modata_16x(), canmsg::len, NO_RX_INT, NO_TX_INT, send_mo_16x(), and USE_XTID.

```
{
        // double check the size
        if(cm->len > 8)
                cm->len = 8;
        // (re)define the (first) message object.
        def_mo_16x( 1, USE_XTID, cm->id, DIR_XMIT, cm->len, NO_TX_INT, NO_RX_INT );
        // load the payload into the message object
        ld_modata_16x( 1, cm->d );
        // transmit the message
        send_mo_16x( 1 );
}
```

#### 5.3.1.3 unsigned char check_busoff_16x (void)

Checks for a busoff state of the controller.

Checks the status register of the CAN controller to see, if it has gone busoff, i.e. doesn’t work anymore. Tries to recover from busoff by reinitializing the CAN controller.

**Returns:**

busoff state

**Return values:**

*0* Controller is working and online.

*1* Controller is busoff. Recovery started.

Definition at line 41 of file CHKBO16X.C.

References CR, and SR.

```
{
        unsigned char busoff_var=0;

        if (SR & 0x80) {        /* if BOFF = 1 */
           busoff_var=1;
           CR=CR & 0xfe;        /* recover from Bus Off (clear INIT) */
        }
        return busoff_var;
}
```

#### 5.3.1.4 unsigned char check_mo15_16x (void)

Check the last message object for new data.

Checks if the last message object (15) has new data. If you want to check the other message objects, please use check_mo_16x().

**Returns:**

indication of new data

**Return values:**

*0* No new data received.

*1* New data received.

Definition at line 41 of file CHM1516X.C.

References msgctrl_ptr_16x.

Referenced by can_read().

```
{
        unsigned char new_event_var=0;

        /* if NEWDAT or RMTPND is set, return 1 */
        if (*msgctrl_ptr_16x[15] & 0x8200)
                new_event_var=1;

        return new_event_var;
}
```

#### 5.3.1.5 unsigned char check_mo_16x (unsigned char *nr*)

Check a message object for freshly received data.

Checks if the specified message object (1..14) has new data. If you want to check the last message object, please use check_mo15_16x().

**Parameters:**

*nr* number of the message object to check (1..14)

**Returns:**

indication of new data

**Return values:**

*0* No new data received.

*1* New data received.

Definition at line 45 of file CHKMO16X.C.

References msgctrl_ptr_16x.

Referenced by can_read().

```
46 {
47         unsigned char new_data_var=0;
48
49         if ((nr<15) && (nr)) {
50                 /* if NEWDAT is set, return 1 */
51                 if (*msgctrl_ptr_16x[nr] & 0x0200)
52                         new_data_var=1;
53         }
54         return new_data_var;
55 }
```

#### 5.3.1.6 void def_mo_16x (unsigned char *nr*, unsigned char *xtd*, unsigned long *id*, unsigned char *dir*, unsigned char *dlc*, unsigned char *txie*, unsigned char *rxie*)

Define a message object.

Use this function to define a message object, i.e. to configure it as extended frame or standard, assign a transmission id, set the data length, enable or disable interrupts on transmit/receive, and of course, whether it's going to be sent or received.

**Parameters:**

*nr* Number of the message object to configure.

*xtd* Flag whether to use extended frames (USE_XTID) or to use standard frames (USE_STID).

*id* Identifier of the message.

*dir* Flag for the direction of the message. Use DIR_XMIT for transmission objects and DIR_RECV for receiving objects

*dlc* Length of the message. Only important for transmission objects.

*txie* Flag for enabling/disabling transmit interrupts (TX_INT/NO_TX_INT)

*rxie* Flag for enabling/disabling receive interrupts (RX_INT/NO_RX_INT)

Definition at line 62 of file DEFMO16X.C.

References dir_bit_16x, dlc_16x, id_ptr_16x, msgconf_ptr_16x, msgctrl_ptr_16x, and xtd_bit_16x.

Referenced by can_write().

```
{
        unsigned int dummy_int;
        unsigned int *dummy_idptr;

        if ((nr<16) && (nr)) {
                dummy_idptr=id_ptr_16x[nr] + 1; /* set dummy ptr to LAR */

                if (xtd) { /* load Arbitration Registers with XTD ID: */
                        /* load Upper Arb. Reg.: */
                        id=id<<3;
                        dummy_int=(unsigned int) (id>>16);
                        *id_ptr_16x[nr]=(dummy_int<<8)+(dummy_int>>8);

                        /* load Lower Arb. Reg. */
                        dummy_int=(unsigned int) id;
                        *dummy_idptr=(dummy_int<<8)+(dummy_int>>8);
                } else {        /* load Arbitration Registers with STD ID: */
                        /* load Upper Arb. Reg.: */
                        dummy_int=(unsigned int) id;
                        dummy_int=dummy_int<<5;
                        *id_ptr_16x[nr]=(dummy_int<<8)+(dummy_int>>8);

                        /* load Lower Arb. Reg.: */
                        *dummy_idptr=0x0800;
                }

                /* prepare Message Control Register: */
                if (txie==1)
                        (txie=0x20);
                else
                        (txie=0x10);
                if (rxie==1)
                        (rxie=0x08);
                else
                        (rxie=0x04);
                if (dir==1)
                        dummy_int = (0x5981 | txie | rxie); /* CPUUPD set */
                else
                        dummy_int = (0x5581 | txie | rxie);     /* MSGLST reset */
                *msgctrl_ptr_16x[nr]=dummy_int; /* load Mess. Contr. Reg. */

                /* prepare Message Configuration Register: */
                if (dlc>8)
                        dlc=8;
                dlc_16x[nr]=dlc;
                dir_bit_16x[nr]=dir;
                xtd_bit_16x[nr]=xtd;
                *msgconf_ptr_16x[nr] = (dlc<<4) + (dir<<3) + (xtd<<2);
        }
}
```

#### 5.3.1.7 void init_can_16x (unsigned int *baud_rate*, unsigned char *eie*, unsigned char *sie*, unsigned char *ie*)

Initialize the CAN module.

This function must be called before any CAN communication can take place. It resets the CAN controller, sets the baud rate and enables interrupts (if specified) for errors, status changes, and globally for the CAN controller.

**Parameters:**

***baud_rate*** Baud rate in kBit/s. You can specify 50, 125, 250, 500 or 1000. If you specify anything else, the CAN bus will be set to 500kBit/s as a default.

***eie*** Flag whether to enable Error Interrupts (see C166 manuals)

***sie*** Flag whether to enable Status Change Interrupts (see C166 manuals)

***ie*** Flag whether to enable Interrupt propagation from the CAN module to the C166 core

Definition at line 87 of file INCAN16X.C.

References BTR, BTR_VALUE_125KBAUD, BTR_VALUE_1MBAUD, BTR_-VALUE_250KBAUD, BTR_VALUE_500KBAUD, BTR_VALUE_50KBAUD, CR, db0_ptr_16x, dir_bit_16x, dlc_16x, GMS, id_ptr_16x, LGML, LMLM, msgconf_ptr_-16x, msgctrl_ptr_16x, SR, UGML, UMLM, and xtd_bit_16x.

```
{
        unsigned char i, n;
        unsigned char *dummy_dbptr;

        /* Initialization PORT4 (CAN) (P4.6 to output; P4.5 to input): */
        _bfld_ (P4, 0x0060, 0x0060);
        _bfld_ (DP4, 0x0060, 0x0040);

        /* Load C167 pointers: */
        for (i=1;i<16;i++) {
                /* set pointers to data bytes 0 of MO 1..15 */
                 db0_ptr_16x[i] = (unsigned char *)(0xef07+i*16);
                 /* set pointers to id’s of MO 1..15 (UARs) */
                 id_ptr_16x[i] = (unsigned int *)(0xef02+i*16);
                 /* set pointers to Message Conf. Registers of MO 1..15 */
                 msgconf_ptr_16x[i] = (unsigned char *)(0xef06+i*16);
                 /* set pointers to Message Control Registers of MO 1..15 */
                 msgctrl_ptr_16x[i] = (unsigned int *)(0xef00+i*16);

                 dir_bit_16x[i] = 0;             /* clear DIR bit array */
                 xtd_bit_16x[i] = 0;             /* clear XTD bit array */
                 dlc_16x[i] = 0;                 /* clear data length code array */
         }

         /* Load General CAN-Registers: */
         CR=0x41;        /* set CCE and INIT in Control Register (EF00h) */
         SR=0x00;        /* Clear Status Partition (EF01h) */

         switch (baud_rate) {
                 case 50:        BTR=BTR_VALUE_50KBAUD;
                                         break;
                 case 125:       BTR=BTR_VALUE_125KBAUD;
```

```
                                break;
                case 250:       BTR=BTR_VALUE_250KBAUD;
                                break;
                case 500:       BTR=BTR_VALUE_500KBAUD;
                                break;
                case 1000:      BTR=BTR_VALUE_1MBAUD;
                                break;
                default:        BTR=BTR_VALUE_500KBAUD;
                                break;
        }
        /* 0 1 1 1  1 0 1 0  1 1 0 0  0 0 0 0 = 500 kBit/s @ 20MHz  */
        /* - TSEG2   TSEG1   SJW|<--  BRP -->|                      */

        /* each bit of standard ID must match to store mess. */
        GMS=0xe0ff;     /* Global Mask Short (EF06h) */

        /* each bit of extended ID must match to store mess. */
        UGML=0xffff;    /* Upper Global Mask Long (EF08h) */
        LGML=0xf8ff;    /* Lower Global Mask Long (EF0Ah) */

        /* every message into MO 15 (Basic CAN Feature)*/
        UMLM=0x0000;    /* Upper Mask of Last Message (EF0Ch) */
        LMLM=0x0000;    /* Lower Mask of Last Message (EF0Eh) */

        /* reset all elements incl MSGVAL in all Message Object Ctrl. Reg.: */
        for (i=1;i<16;i++) *msgctrl_ptr_16x[i] = 0x5555;

        /* reset all data bytes in all Message Objects: */
        for (i=1;i<16;i++) {
                dummy_dbptr=db0_ptr_16x[i];
                for (n=0;n<8;n++)
                        *dummy_dbptr++ = 0x00;
        }

        /* end initialization (CCE=0, INIT=0); Interrupts EIE, SIE, IE=user: */
        CR = (0x00 | (eie<<3) | (sie<<2) | (ie<<1));
}
```

### 5.3.1.8 void ld_modata_16x (unsigned char *nr*, unsigned char ∗ *upl_data_ptr*)

Load data into a message object.

Use this function to load the data just before sending the message object. The message object must of course be defined before using def_mo_16x().

**Parameters:**

***nr*** Number of the message object to load (1..14)

∗***upl_data_ptr*** Pointer to the data to be copied into the message object.

**Warning:**

∗upl_data_ptr should point to exactly the amount of data specified when defining the message object. Otherwise, the message might contain useless ”random” data.

Definition at line 48 of file LDMOD16X.C.

References db0_ptr_16x, dlc_16x, and msgctrl_ptr_16x.

Referenced by can_write().

```
{

        unsigned char i;
        unsigned char *dummy_dbptr;

        if ((nr<15) && (nr)) {
                *msgctrl_ptr_16x[nr]=0xfaff;            /* set CPUUPD, NEWDAT */
                dummy_dbptr=db0_ptr_16x[nr];            /* load dummy ptr (db 0) */
                /* move data bytes from upload buffer to MO data bytes */
                for (i=0;i<dlc_16x[nr];i++)
                        *dummy_dbptr++ = *upl_data_ptr++;
                /* reset CPUUPD (NEWDAT is reset by CAN Module on Transm.) */
                *msgctrl_ptr_16x[nr]=0xf7ff;
        }
}
```

#### 5.3.1.9 void rd_mo15_16x (unsigned char ∗ *mo15_db_ptr*, unsigned long ∗ *mo15_id_ptr*, unsigned char ∗ *mo15_dlc_ptr*)

Read data from the last message object.

Use this function to read received data (and the message id) from the last message object (#15).

**Parameters:**

∗***mo15_db_ptr*** Pointer to a buffer where the received data will be put.

∗***mo15_id_ptr*** Pointer to a long where the message id of the newly received message will be put.

∗***mo15_dlc_ptr*** Pointer to a character where the length of the received message will be put.

**Warning:**

∗mo15_db_ptr must point to an array large enough to hold all received data. Always make this large enough for the largest possible amount of payload, being 8 bytes.

Definition at line 51 of file RDM1516X.C.

References db0_ptr_16x, dlc_16x, id_ptr_16x, msgconf_ptr_16x, msgctrl_ptr_16x, and xtd_bit_16x.

Referenced by can_read().

```
{

        unsigned char i;
        unsigned int dummy_int1, dummy_int2;
        unsigned char *dummy_dbptr;
        unsigned int *dummy_idptr;

        /* read actual data length code from momentarily accessed buffer: */
        *mo15_dlc_ptr=dlc_16x[15]=(*msgconf_ptr_16x[15]>>4);

        /* read actual identifier from momentarily accessed buffer */
        /* and convert it into a usable long: */
        dummy_idptr=id_ptr_16x[15];
        if (xtd_bit_16x[15]) {          /* calculate XTD ID: */
                dummy_int1=*dummy_idptr++;
                dummy_int2=*dummy_idptr;
                dummy_int1=(dummy_int1<<8)+(dummy_int1>>8);
                dummy_int2=(dummy_int2<<8)+(dummy_int2>>8);
                *mo15_id_ptr=((((unsigned long)(dummy_int1))<<16) + dummy_int2)>>3;
        } else {                                /* calculate STD ID: */
                dummy_int1=*dummy_idptr;
                *mo15_id_ptr=(unsigned long) (((dummy_int1<<8)+(dummy_int1>>8))>>5);
        }

        /* get actual data bytes from momentarily accessed buffer: */
        dummy_dbptr=db0_ptr_16x[15];
        for (i=0;i<dlc_16x[15];i++)
                *mo15_db_ptr++ = *dummy_dbptr++;

        /* clr NEWDAT,INTPND & RMTPND to release mom. acc. buffer: */
        *msgctrl_ptr_16x[15]=0xedfd;
}
```

#### 5.3.1.10 void rd_modata_16x (unsigned char *nr*, unsigned char ∗ *downl_data_ptr*, unsigned long ∗ *mo_id_ptr*, unsigned char ∗ *mo_dlc_ptr*)

Read data from a message object.

Use this function to read received data (and the message id) from the specified message object (#1..#14).

**Parameters:**

***nr*** Number of the message object to read from (1..14)

∗***downl_data_ptr*** Pointer to a buffer where the received data will be put.

∗***mo_id_ptr*** Pointer to a long where the message id of the newly received message will be put.

∗***mo_dlc_ptr*** Pointer to a character where the length of the received message will be put.

**Warning:**

∗downl_data_ptr must point to an array large enough to hold all received data. Always make this large enough for the largest possible amount of payload, being 8 bytes.

**Note:**

The reading of the data bytes is repeated if a new message comes in for message ’nr’ during the data bytes are read, because the CAN module sets NEWDAT in this case.

Definition at line 53 of file RDMOD16X.C.

References db0_ptr_16x, dlc_16x, id_ptr_16x, msgconf_ptr_16x, msgctrl_ptr_16x, and xtd_bit_16x.

Referenced by can_read().

```
{

        unsigned char i, dummy_char;
        unsigned char *dummy_dbptr;
        unsigned int *dummy_idptr;
        unsigned int dummy_int1, dummy_int2;

        if ((nr<15) && (nr)) {
                do {
                        dummy_char=*msgconf_ptr_16x[nr];
                        // added hp 2003-02-24:
                        *mo_dlc_ptr = (dummy_char>>4);

                        dummy_idptr=id_ptr_16x[nr];
                        if (xtd_bit_16x[nr]) {          /* calculate XTD ID: */
                                dummy_int1=*dummy_idptr++;
                                dummy_int2=*dummy_idptr;
                                dummy_int1=(dummy_int1<<8)+(dummy_int1>>8);
                                dummy_int2=(dummy_int2<<8)+(dummy_int2>>8);
                                *mo_id_ptr=((((unsigned long)(dummy_int1))<<16) + dummy_in
                        } else {                                /* calculate STD ID: */
                                dummy_int1=*dummy_idptr;
                                *mo_id_ptr=(unsigned long) (((dummy_int1<<8)+(dummy_int1>>
                        }
                        // end of addition

                        /* store actual data length code */
                        dlc_16x[nr]=(dummy_char>>4);
                        /* clr NEWDAT and INTPND */
                        *msgctrl_ptr_16x[nr]=0xfdfd;
                        /* load dummy ptr (db 0) */
                        dummy_dbptr=db0_ptr_16x[nr];
                        /* move data bytes from MO's data bytes to download buffer */
                        for (i=0;i<dlc_16x[nr];i++)
                                *downl_data_ptr++ = *dummy_dbptr++;
                } while (*msgctrl_ptr_16x[nr] & 0x0200);        /* while NEWDAT=1 */
        }
}
```

### 5.3.1.11 void send_mo_16x (unsigned char *nr*)

Send a message object.

This function transmits the specified message object. Before that, it must be configured as a transmit object and it should also contain valid and meaningful data.

**Parameters:**

***nr*** Number of the message object to transmit (1..14).

Definition at line 41 of file SNDMO16X.C.

References msgctrl_ptr_16x.

Referenced by can_write().

```
{
        if ((nr<15) && (nr))
                *msgctrl_ptr_16x[nr]=0xefff;  /* set TXRQ */
}
```

#### 5.3.1.12 void set_global_mask (unsigned long *mask*)

Sets the global reception mask.

Each incoming message is filtered through the global mask. Each bit that is set to 1 in the global mask must match the corresponding id bits of the receiving message objects. Each bit that is set to 0 means ”don’t care” when comparing the id of the incoming message with the receive objects.

**Parameters:**

***mask*** Only the lower 29 bits are used. As the layout of the CAN registers within the C166 is kind of strange, we need to do lots of shifting. The exact layout is of no interest to the user of this driver, because it takes care of that. If anyone is interested in the details behind this, (s)he may consult the C166 user’s manuals.

Definition at line 24 of file mask.c.

References LGML, and UGML.

```
{
        UGML = ((mask >> 21) & 0xff) | (((mask >> 13) & 0xff) << 8);
        LGML = ((mask >>  5) & 0xff) | ((mask         & 0x1f) << 11);
}
```

#### 5.3.1.13 void undef_mo_16x (unsigned char *nr*)

Clear a message object.

If the message object is not to be used anymore, you can disable it using this function.

**Parameters:**

***nr*** Number of the message object to disable.

Definition at line 44 of file DEFMO16X.C.

References msgctrl_ptr_16x.

```
{
           *msgctrl_ptr_16x[nr]=0x5555;
}
```

## 5.4 Bit Timing values

### Defines

- #define BTR_VALUE_50KBAUD 0x7aC9

  *Bit timing for 50kBit/s.*

- #define BTR_VALUE_125KBAUD 0x7aC3

  *Bit timing for 125kBit/s.*

- #define BTR_VALUE_250KBAUD 0x7aC1

  *Bit timing for 250kBit/s.*

- #define BTR_VALUE_500KBAUD 0x7aC0

  *Bit timing for 500kBit/s.*

- #define BTR_VALUE_1MBAUD 0x25c0

  *Bit timing for 1MBit/s.*

## 5.5 SFRs for the message objects

### Variables

- unsigned int ∗ id_ptr_16x [16]

  *pointer to message id's (UAReg)*

- unsigned char ∗ db0_ptr_16x [16]

  *pointer to 'databyte 0's*

- unsigned int ∗ msgctrl_ptr_16x [16]

  *pointer to message control registers*

- unsigned char ∗ msgconf_ptr_16x [16]

  *pointer to message configuration registers*

## 5.6 Global variables for the CAN driver.

### Variables

- unsigned char dir_bit_16x [16]

  *DIR bits MO 1...15.*

- unsigned char xtd_bit_16x [16]

  *XTD bits MO 1...15.*

- unsigned char dlc_16x [16]

  *data byte lengths MO 1...15*

# Chapter 6

# CAN Library for the C166 family Data Structure Documentation

## 6.1 canmsg Struct Reference

Structure for a can message.

`#include <CANR_16X.H>`

### Data Fields

- unsigned long id

  *message identifier*

- int rtr

  *remote transmission request flag*

- int len

  *data length (0..8)*

- unsigned char d [8]

  *the data*

- time_t timestamp

  *time when it was sent or received*

### 6.1.1 Detailed Description

Structure for a can message.

This is mainly used by the publisher/subscriber library to transmit and receive messages.

Definition at line 279 of file CANR_16X.H.

The documentation for this struct was generated from the following file:

- CANR_16X.H

# Chapter 7

# CAN Library for the C166 family File Documentation

## 7.1 CANR_16X.H File Reference

```
#include "../sys/time.h"
```

### Data Structures

- struct canmsg

  *Structure for a can message.*

### Defines

- #define CR ∗(unsigned char∗) 0xef00

  *The Command register.*

- #define SR ∗(unsigned char∗) 0xef01

  *The Status register.*

- #define IR ∗(unsigned char∗) 0xef02

  *The Interrupt register.*

- #define BTR ∗(unsigned int ∗) 0xef04

  *The baudrate register.*

- #define GMS ∗(unsigned int ∗) 0xef06

  *The global short mask (11 Bit IDs).*

- #define UGML ∗(unsigned int ∗) 0xef08

  *The upper half of the global long mask (29 Bit IDs).*

- #define LGML ∗(unsigned int ∗) 0xef0a

  *The lower half of the global long mask (29 Bit IDs).*

- #define UMLM ∗(unsigned int ∗) 0xef0c

  *The upper half of the long mask for the last message.*

- #define LMLM ∗(unsigned int ∗) 0xef0e

  *The lower half of the long mask for the last message.*

- #define CAN_INTERRUPT 0x40

  *The interrupt vector of the CAN module.*

- #define USE_XTID 1

  *Use the extended frame format with 29 bit identifiers.*

- #define USE_STID 0

  *Use the standard frame format with 11 bit identifiers.*

- #define DIR_XMIT 1

  *Make the message object a transmit object.*

- #define DIR_RECV 0

  *Make the message object a receive object.*

- #define NO_TX_INT 0

  *Disable transmit interrupts.*

- #define NO_RX_INT 0

  *Disable receive interrupts.*

- #define TX_INT 1

  *Enable transmit interrupts.*

- #define RX_INT 1

  *Enable receive interrupts.*

## Functions

- void init_can_16x (unsigned int baud_rate, unsigned char eie, unsigned char sie, unsigned char ie)

  *Initialize the CAN module.*

- void send_mo_16x (unsigned char nr)

  *Send a message object.*

- void rd_modata_16x (unsigned char nr, unsigned char ∗downl_data_ptr, unsigned long ∗mo_id_ptr, unsigned char ∗mo_dlc_ptr)

  *Read data from a message object.*

- void rd_mo15_16x (unsigned char ∗mo15_db_ptr, unsigned long ∗mo15_id_ptr, unsigned char ∗mo15_dlc_ptr)

  *Read data from the last message object.*

- void ld_modata_16x (unsigned char nr, unsigned char ∗upl_data_ptr)

  *Load data into a message object.*

- void def_mo_16x (unsigned char nr, unsigned char xtd, unsigned long id, unsigned char dir, unsigned char dlc, unsigned char txie, unsigned char rxie)

  *Define a message object.*

- void undef_mo_16x (unsigned char nr)

  *Clear a message object.*

- unsigned char check_mo_16x (unsigned char nr)

  *Check a message object for freshly received data.*

- unsigned char check_mo15_16x (void)

  *Check the last message object for new data.*

- unsigned char check_busoff_16x (void)

  *Checks for a busoff state of the controller.*

- void set_global_mask (unsigned long mask)

  *Sets the global reception mask.*

- void can_write (canmsg ∗cm)

  *Send a can message.*

- int can_read (int nr, canmsg ∗cm)

  *Try reading a can message.*

### 7.1.1 Detailed Description

**Author:**

Axel Wolf, SCI Cupertino
Dr. Jens Barrenscheen, HL MC PD, Munich
Hubert Piontek, University of Ulm

CAN Driver include file for the C166 family. $Id$ This file contains the definitions for the CAN driver for the C166 family. Most of this is based von the Infineon/Siemens Application Note AP2922. Some additions made at University of Ulm, February 2003.

This CAN driver is mainly meant to be used by the publisher/subscriber library developed at the University of Ulm, 1999-2003.

Definition in file CANR_16X.H.

## 7.2 CHKBO16X.C File Reference

```
#include <CANR_16X.H>
```

### Functions

- unsigned char check_busoff_16x (void)

  *Checks for a busoff state of the controller.*

### 7.2.1 Detailed Description

**Author:**

Axel Wolf, SCI Cupertino
Dr. Jens Barrenscheen, HL MC PD, Munich
Hubert Piontek, University of Ulm

Routine for checking for busoff situations on the C166 CAN module.

Most of this is based von the Infineon/Siemens Application Note AP2922. Some additions made at University of Ulm, February 2003.

This CAN driver is mainly meant to be used by the publisher/subscriber library developed at the University of Ulm, 1999-2003.

Definition in file CHKBO16X.C.

## 7.3 CHKMO16X.C File Reference

### Functions

- unsigned char check_mo_16x (unsigned char nr)
  *Check a message object for freshly received data.*

### Variables

- unsigned int ∗ msgctrl_ptr_16x [16]
  *pointer to message control registers.*

### 7.3.1 Detailed Description

**Author:**

Axel Wolf, SCI Cupertino
Dr. Jens Barrenscheen, HL MC PD, Munich
Hubert Piontek, University of Ulm

Routine for checking for new data in one of the message objects 1..14

Most of this is based von the Infineon/Siemens Application Note AP2922. Some additions made at University of Ulm, February 2003.

This CAN driver is mainly meant to be used by the publisher/subscriber library developed at the University of Ulm, 1999-2003.

Definition in file CHKMO16X.C.

## 7.4 CHM1516X.C File Reference

### Functions

- unsigned char check_mo15_16x (void)

  *Check the last message object for new data.*

### Variables

- unsigned int * msgctrl_ptr_16x [16]

  *pointer to message control registers*

### 7.4.1 Detailed Description

**Author:**

Axel Wolf, SCI Cupertino
Dr. Jens Barrenscheen, HL MC PD, Munich
Hubert Piontek, University of Ulm

Routine for checking for new data in the last message object.

Most of this is based von the Infineon/Siemens Application Note AP2922. Some additions made at University of Ulm, February 2003.

This CAN driver is mainly meant to be used by the publisher/subscriber library developed at the University of Ulm, 1999-2003.

Definition in file CHM1516X.C.

## 7.5 DEFMO16X.C File Reference

### Functions

- void undef_mo_16x (unsigned char nr)

  *Clear a message object.*

- void def_mo_16x (unsigned char nr, unsigned char xtd, unsigned long id, unsigned char dir, unsigned char dlc, unsigned char txie, unsigned char rxie)

  *Define a message object.*

### Variables

- unsigned int ∗ id_ptr_16x [16]

  *pointer to message id's (UAReg)*

- unsigned char ∗ db0_ptr_16x [16]

  *pointer to 'databyte 0's*

- unsigned int ∗ msgctrl_ptr_16x [16]

  *pointer to message control registers*

- unsigned char ∗ msgconf_ptr_16x [16]

  *pointer to message configuration registers*

- unsigned char dir_bit_16x [16]

  *DIR bits MO 1...15.*

- unsigned char xtd_bit_16x [16]

  *XTD bits MO 1...15.*

- unsigned char dlc_16x [16]

  *data byte lengths MO 1...15*

### 7.5.1 Detailed Description

**Author:**

Axel Wolf, SCI Cupertino
Dr. Jens Barrenscheen, HL MC PD, Munich
Hubert Piontek, University of Ulm

Routine for defining a message object.

Most of this is based von the Infineon/Siemens Application Note AP2922. Some additions made at University of Ulm, February 2003.

This CAN driver is mainly meant to be used by the publisher/subscriber library developed at the University of Ulm, 1999-2003.

Definition in file DEFMO16X.C.

## 7.6 INCAN16X.C File Reference

```
#include <REG167.H>
#include <CANR_16X.H>
#include <intrins.h>
```

### Defines

- #define BTR_VALUE_50KBAUD 0x7aC9

  *Bit timing for 50kBit/s.*

- #define BTR_VALUE_125KBAUD 0x7aC3

  *Bit timing for 125kBit/s.*

- #define BTR_VALUE_250KBAUD 0x7aC1

  *Bit timing for 250kBit/s.*

- #define BTR_VALUE_500KBAUD 0x7aC0

  *Bit timing for 500kBit/s.*

- #define BTR_VALUE_1MBAUD 0x25c0

  *Bit timing for 1MBit/s.*

### Functions

- void init_can_16x (unsigned int baud_rate, unsigned char eie, unsigned char sie, unsigned char ie)

  *Initialize the CAN module.*

### Variables

- unsigned int ∗ id_ptr_16x [16]

  *pointer to message id's (UAReg)*

- unsigned char ∗ db0_ptr_16x [16]

  *pointer to 'databyte 0's*

- unsigned int ∗ msgctrl_ptr_16x [16]

  *pointer to message control registers*

- unsigned char ∗ msgconf_ptr_16x [16]

  *pointer to message configuration registers*

- unsigned char dir_bit_16x [16]

  *DIR bits MO 1...15.*

- unsigned char xtd_bit_16x [16]

  *XTD bits MO 1...15.*

- unsigned char dlc_16x [16]

  *data byte lengths MO 1...15*

### 7.6.1 Detailed Description

**Author:**

Axel Wolf, SCI Cupertino
Dr. Jens Barrenscheen, HL MC PD, Munich
Hubert Piontek, University of Ulm

Routine for initializing the CAN module of the C166

Most of this is based von the Infineon/Siemens Application Note AP2922. Some additions made at University of Ulm, February 2003.

This CAN driver is mainly meant to be used by the publisher/subscriber library developed at the University of Ulm, 1999-2003.

Definition in file INCAN16X.C.

## 7.7 LDMOD16X.C File Reference

### Functions

- void ld_modata_16x (unsigned char nr, unsigned char ∗upl_data_ptr)

  *Load data into a message object.*

### Variables

- unsigned int ∗ id_ptr_16x [16]

  *pointer to message id's (UAReg)*

- unsigned char ∗ db0_ptr_16x [16]

  *pointer to 'databyte 0's*

- unsigned int ∗ msgctrl_ptr_16x [16]

  *pointer to message control registers*

- unsigned char ∗ msgconf_ptr_16x [16]

  *pointer to message configuration registers*

- unsigned char dir_bit_16x [16]

  *DIR bits MO 1...15.*

- unsigned char xtd_bit_16x [16]

  *XTD bits MO 1...15.*

- unsigned char dlc_16x [16]

  *data byte lengths MO 1...15*

### 7.7.1 Detailed Description

**Author:**

Axel Wolf, SCI Cupertino
Dr. Jens Barrenscheen, HL MC PD, Munich
Hubert Piontek, University of Ulm

Routine for loading data into message objects.

Most of this is based von the Infineon/Siemens Application Note AP2922. Some additions made at University of Ulm, February 2003.

This CAN driver is mainly meant to be used by the publisher/subscriber library developed at the University of Ulm, 1999-2003.

Definition in file LDMOD16X.C.

## 7.8 mask.c File Reference

```
#include "canr_16x.h"
#include "../ser/ser.h"
```

### Functions

- void set_global_mask (unsigned long mask)

  *Sets the global reception mask.*

- void can_write (canmsg ∗cm)

  *Send a can message.*

- int can_read (int nr, canmsg ∗cm)

  *Try reading a can message.*

### 7.8.1 Detailed Description

**Author:**

Axel Wolf, SCI Cupertino
Dr. Jens Barrenscheen, HL MC PD, Munich
Hubert Piontek, University of Ulm

Added functions to set the global reception mask and low-level interface for the publisher/subscriber library.

Definition in file mask.c.

## 7.9 RDM1516X.C File Reference

`#include <REG167.H>`

`#include <CANR_16X.H>`

### Functions

- void rd_mo15_16x (unsigned char ∗mo15_db_ptr, unsigned long ∗mo15_id_ptr, unsigned char ∗mo15_dlc_ptr)

  *Read data from the last message object.*

### Variables

- unsigned int ∗ id_ptr_16x [16]

  *pointer to message id's (UAReg)*

- unsigned char ∗ db0_ptr_16x [16]

  *pointer to 'databyte 0's*

- unsigned int ∗ msgctrl_ptr_16x [16]

  *pointer to message control registers*

- unsigned char ∗ msgconf_ptr_16x [16]

  *pointer to message configuration registers*

- unsigned char dir_bit_16x [16]

  *DIR bits MO 1...15.*

- unsigned char xtd_bit_16x [16]

  *XTD bits MO 1...15.*

- unsigned char dlc_16x [16]

  *data byte lengths MO 1...15*

### 7.9.1 Detailed Description

**Author:**

Axel Wolf, SCI Cupertino
Dr. Jens Barrenscheen, HL MC PD, Munich
Hubert Piontek, University of Ulm

Routine for reading data from the last message object

Most of this is based von the Infineon/Siemens Application Note AP2922. Some additions made at University of Ulm, February 2003.

This CAN driver is mainly meant to be used by the publisher/subscriber library developed at the University of Ulm, 1999-2003.

Definition in file RDM1516X.C.

## 7.10 RDMOD16X.C File Reference

### Functions

- void rd_modata_16x (unsigned char nr, unsigned char ∗downl_data_ptr, unsigned long ∗mo_id_ptr, unsigned char ∗mo_dlc_ptr)

  *Read data from a message object.*

### Variables

- unsigned int ∗ id_ptr_16x [16]

  *pointer to message id's (UAReg)*

- unsigned char ∗ db0_ptr_16x [16]

  *pointer to 'databyte 0's*

- unsigned int ∗ msgctrl_ptr_16x [16]

  *pointer to message control registers*

- unsigned char ∗ msgconf_ptr_16x [16]

  *pointer to message configuration registers*

- unsigned char dir_bit_16x [16]

  *DIR bits MO 1...15.*

- unsigned char xtd_bit_16x [16]

  *XTD bits MO 1...15.*

- unsigned char dlc_16x [16]

  *data byte lengths MO 1...15*

### 7.10.1 Detailed Description

**Author:**

Axel Wolf, SCI Cupertino
Dr. Jens Barrenscheen, HL MC PD, Munich
Hubert Piontek, University of Ulm

Routine for reading data from the last message object

Most of this is based von the Infineon/Siemens Application Note AP2922. Some additions made at University of Ulm, February 2003.

This CAN driver is mainly meant to be used by the publisher/subscriber library developed at the University of Ulm, 1999-2003.

Definition in file RDMOD16X.C.

## 7.11 SNDMO16X.C File Reference

```
#include <REG167.H>
#include <CANR_16X.H>
```

### Functions

- void send_mo_16x (unsigned char nr)
  *Send a message object.*

### Variables

- unsigned int ∗ msgctrl_ptr_16x [16]
  *pointer to message control registers*

### 7.11.1 Detailed Description

**Author:**

Axel Wolf, SCI Cupertino
Dr. Jens Barrenscheen, HL MC PD, Munich
Hubert Piontek, University of Ulm

Routine for sending a message object

Most of this is based von the Infineon/Siemens Application Note AP2922. Some additions made at University of Ulm, February 2003.

This CAN driver is mainly meant to be used by the publisher/subscriber library developed at the University of Ulm, 1999-2003.

Definition in file SNDMO16X.C.

# Index