UM0459
User manual

ST10F27xZx CAN library

Introduction

The ST10F27xZx product family provides two C-CAN cells with enhanced features compared to the B-CAN cell implemented in the ST10F269 family.

This ST10F27xZx software CAN library user manual gives guidelines for using the C-CAN cell and, describes the different routines and data structures implemented in the CAN library.

The library is compatible with both Tasking and Keil.

Contents

List of tables

1 Using the ST10F27xZx CAN Library

1.1 Library content

The ST10F27xZx CAN library is a software package consisting of a zip file containing the following folders:

- Library containing the C source and header files.
  - st10f27x_can.c C source file containing the C source of the library functions
  - st10f27_can.h: header file containing the function prototypes and the CAN interface definitions
  - st10F27x_types.h: header files containing the definition of data types used by the library.
- Example folder with two examples on how to use the CAN library functions. Polling and interrupt examples are provided.

CAN cell access is made using pointers to the CAN cell addresses. This makes the library compatible with both Keil and Tasking. The provided examples are compiled and tested under Tasking toolchain. Applications can be ported easily to the Keil toolchain.

1.2 How to install the ST10F27xZx CAN Library

The ST10F27xZx CAN library is delivered as an archive file with .zip extension. To install the CAN library, the user needs to unzip the file in the directory where the library has to be copied.

1.3 Getting started with the library

To use the CAN library, the user should include the st10f27x_can.h header file.

The two CAN interfaces are accessed using the structure pointing to their addresses. CAN and IF interface definitions are located in st10f27x_can.h. When calling the CAN library functions, the user should use the CAN1 or CAN2 parameter.

The user should pay attention to the bitrate configuration: if the application environment is different from the library conditions, the user should recalculate the seg1, seg2,brp and sjw CAN bit timing parameters. This is the case when ST10 frequency = 40 MHz and the CAN clock input is derived from the clock generator divided by 2 (see XMISC register).

For more details, please refer to the CAN bit timing configuration in the ST10F27xZx user manual.

The CAN input/output pins must be configured by the user before using the CAN library.

2 Controller area network (CAN) routines

Section 2.1 describes the data structures used in the CAN software library and *Section 2.2* gives a list of the software library functions.

Note: Before using any CAN function, the I/O ports linked to the CAN RX and CAN TX pins must be configured as follows: CAN RX pin in input mode and CAN TX pin in output mode. This is not performed by the library.

2.1 CAN register structure

The structures of the *CAN_TypeDef* and *CAN_MsgObj_TypeDef* registers are defined in the ***st10f27x_can.h*** file as follows:

```
typedef volatile struct
  {
    vu16 CRR;
    vu16 CMR;
    vu16 M1R;
    vu16 M2R;
    vu16 A1R;
    vu16 A2R;
    vu16 MCR;
    vu16 DA1R;
    vu16 DA2R;
    vu16 DB1R;
    vu16 DB2R;
    u16  EMPTY0[13];
} CAN_MsgObj_TypeDef;

typedef volatile struct
{
  vu16 CR;
  vu16 SR;
  vu16 ERR;
  vu16 BTR;
  vu16 IDR;
  vu16 TESTR;
  vu16 BRPR;
  u16  EMPTY0;
  CAN_MsgObj_TypeDef sMsgObj[2];
  u16  EMPTY1[8];
  vu16 TXR1R;
  vu16 TXR2R;
  u16  EMPTY2[6];
  vu16 ND1R;
  vu16 ND2R;
  u16  EMPTY3[6];
  vu16 IP1R;
  vu16 IP2R;
  u16  EMPTY4[6];
  vu16 MV1R;
  vu16 MV2R;
} CAN_TypeDef;
```

CAN registers are listed in the following table:

Table 1. CAN registers

Register	Description
CR	CAN Control Register
SR	CAN Status Register
ERR	CAN Error counter Register
BTR	CAN Bit Timing Register
IDR	CAN Interrupt Identifier Register
TESTR	CAN Test Register
BRPR	CAN BRP Extension Register
CRR	CAN IF1 Command Request Register
CMR	CAN IF1 Command Mask Register
M1R	CAN IF1 Message Mask 1 Register
M2R	CAN IF1 Message Mask 2 Register
A1R	CAN IF1 Message Arbitration 1 Register
A2R	CAN IF1 Message Arbitration 2 Register
MCR	CAN IF1 Message Control Register
DA1R	CAN IF1 DATA A 1 Register
DA2R	CAN IF1 DATA A 2 Register
DB1R	CAN IF1 DATA B 1 Register
DB2R	CAN IF1 DATA B 2 Register
CRR	CAN IF2 Command request Register
CMR	CAN IF2 Command Mask Register
M1R	CAN IF2 Message Mask 1 Register
M2R	CAN IF2 Message Mask 2 Register
A1R	CAN IF2 Message Arbitration 1 Register
A2R	CAN IF2 Message Arbitration 2 Register
MCR	CAN IF2 Message Control Register
DA1R	CAN IF2 DATA A 1 Register
DA2R	CAN IF2 DATA A 2 Register
DB1R	CAN IF2 DATA B 1 Register
DB2R	CAN IF2 DATA B 2 Register
TXR1R	CAN Transmission Request 1 Register
TXR2R	CAN Transmission Request 2 Register
ND1R	CAN New Data 1 Register
ND2R	CAN New Data 2 Register

Table 1. CAN registers (continued)

Register	Description
IP1R	CAN Interrupt Pending 1 Register
IP2R	CAN Interrupt Pending 2 Register
MV1R	CAN Message Valid 1 Register
MV2R	CAN Message Valid 2 Register

The CAN1 and CAN2 are defined in the ***st10f27x_can.h*** file as follows:

```
/* C-CAN modules defintion */
#define CAN1_BASE              0xEF00
#define CAN2_BASE              0xEE00
#define CAN1          ( (CAN_TypeDef *) CAN1_BASE)
#define CAN2          ( (CAN_TypeDef *) CAN2_BASE)
```

Note: To use CAN1 and/or CAN2 interfaces, the CAN1EN and/or CAN2EN bits must be set in the XPERCON register before configuring the SYSCON register.

2.2 Software library functions

The functions of the CAN library are listed in the following table and are described in the corresponding subsections:

Table 2. CAN functions

Function name (corresponding sections)	Description
CAN_Init (*Section 2.2.1*)	Initializes the CAN cell and sets the bitrate.
CAN_EnterInitMode(*Section 2.2.2*)	Switches the CAN to initialization mode.
CAN_LeaveInitMode (*Section 2.2.3*)	Leaves initialization mode (switches to normal mode).
CAN_EnterTestMode (*Section 2.2.4*)	Switches the CAN to test mode.
CAN_LeaveTestMode(*Section 2.2.5*)	Leaves the current test mode (switches to normal mode).
CAN_SetBitrate (*Section 2.2.6*)	Configures a standard CAN bitrate.
CAN_SetTiming (*Section 2.2.7*)	Configures the CAN timing with specific parameters.
CAN_SetUnusedMsgObj (*Section 2.2.8*)	Configures the message object as unused.
CAN_SetTxMsgObj (*Section 2.2.10*)	Configures the message object as TX.
CAN_SetRxMsgObj (*Section 2.2.11*)	Configures the message object as RX.
CAN_SetUnusedAllMsgObj (*Section 2.2.9*)	Configures all message objects as unused.
CAN_ReleaseMessage (*Section 2.2.12*)	Releases the message object.
CAN_UpdateMsgObj (*Section 2.2.14*)	Updates the message object.
CAN_TransmitRequest (*Section 2.2.15*)	Requests the transmission of a message object.
CAN_SendMessage (*Section 2.2.13*)	Updates and starts transmission of a message.

Table 2. CAN functions (continued)

Function name (corresponding sections)	Description
CAN_ReceiveMessage (*Section 2.2.16*)	Gets the message, if received.
CAN_BasicSendMessage (*Section 2.2.17*)	Starts transmission of a message in BASIC mode.
CAN_BasicReceiveMessage (*Section 2.2.18*)	Gets the message in BASIC mode, if received.
CAN_WaitEndOfTx (*Section 2.2.19*)	Waits until the end of transmission of a frame.
CAN_GetMsgReceiveStatus (*Section 2.2.20*)	Tests the waiting status of a received message.
CAN_GetMsgTransmitRequestStatus (*Section 2.2.21*)	Tests the request status of a transmitted message.
CAN_GetMsgInterruptStatus (*Section 2.2.22*)	Tests the interrupt status of a message object.
CAN_GetMsgValidStatus (*Section 2.2.23*)	Tests the validity of a message object (ready to use).
CAN_GetFlagStatus (*Section 2.2.24*)	Returns the status of the TxOK, RxOK, EPASS, EWARN and BOFF flags.
CAN_GetTransmitErrorCounter (*Section 2.2.25*)	Gets the CAN transmit Error counter.
CAN_GetReceiveErrorCounter (*Section 2.2.26*)	Gets the CAN receive Error counter.

2.2.1 CAN_Init function

Table 3. CAN_Init function description

Function name	CAN_Init
Function prototype	`void CAN_Init(CAN_TypeDef* CAN, u16 Mask, u16 Bitrate)`
Behavior description	Initializes the CAN peripheral passed as a parameter.
Input parameter 1	CAN: pointer to the CAN1 or CAN2 CAN cell.
Input parameter 2	*Mask*: any binary value resulting from the following (Refer to the CAN control register definition): – CAN_CR_EIE: CAN Error Interrupt Enable – CAn_CR_SIE: CAN Status Interrupt Enable – CAN_CR_IE: CAN Interrupt Enable – CAN_CR_ DAR: CAN Disable Automatic retransmission – CAN_CR_CCE: CAN Configuration Enable – CAN_CR_TEST: Enable CAN test mode
Input parameter 3	It can be one of the CAN_BITRATE_xxx parameters given in *Table 4: CAN_Bitrate values on page 10*
Output parameter	None
Return parameter	None

Table 3. CAN_Init function description (continued)

Function name	CAN_Init
Required preconditions	The standard bitrates are given for a 40 MHz ST10 frequency and the 1/2 prescaler is activated at the CAN input. For other configurations, use the CAN_SetTiming function.
Called functions	CAN_EnterInitMode() CAN_SetBitrate() CAN_EnterTestMode() CAN_LeaveInitMode()

Bitrate

The CAN bitrate values are given in the following table:

Table 4. CAN_Bitrate values

CAN_Bitrate	Meaning
CAN_BITRATE_100K	100 Kbit/s bitrate
CAN_BITRATE_125K	125 Kbit/s bitrate
CAN_BITRATE_250K	250 Kbit/s bitrate
CAN_BITRATE_500K	500 Kbit/s bitrate
CAN_BITRATE_1M	1 Mbit/s bitrate

The given bitrates correspond to an ST10 frequency of 40 MHz with prescaler 2 activated at the CAN input.

Example:

This example illustrates how to initialize the CAN1 at 500 Kbit/s and enable the interrupts.

```
{

   CAN_Init(CAN1, CAN_CR_IE, CAN_BITRATE_500K);
}
}
```

2.2.2 CAN_EnterInitMode function

Table 5. CAN_EnterInitMode function description

Function	CAN_EnterInitMode
Prototype	`void CAN_EnterInitMode(CAN_TypeDef* CAN, u16 Mask)`
Behavior description	Switches the CAN into initialization mode. This function must be used in conjunction with CAN_LeaveInitMode(). This function sets the INIT of the CAN cell and resets the CAN status register.
Input parameter 1	CAN: pointer to the CAN1 or CAN2 CAN cell.
Input parameter 2	*Mask*: any binary value resulting from the following (Refer to the CAN control register definition): – CAN_CR_EIE: CAN Error Interrupt Enable – CAN_CR_SIE: CAN Status Interrupt Enable – CAN_CR_IE: CAN Interrupt Enable – CAN_CR_ DAR: CAN Disable Automatic retransmission – CAN_CR_CCE: CAN Configuration Enable – CAN_CR_TEST: Enable CAN test mode
Output parameter	None
Return value	None
Required preconditions	None
Called functions	None

2.2.3 CAN_LeaveInitMode function

Table 6. CAN_LeaveInitMode function description

Function	CAN_LeaveInitMode
Prototype	`void CAN_LeaveInitMode(CAN_TypeDef* CAN)`
Behavior description	Leaves initialization mode (switch into normal mode) by clearing INIT and CCE bits in the CAN control register. This function must be used in conjunction with CAN_EnterInitMode().
Input parameter	CAN: pointer on the CAN cell CAN1 or CAN2
Output parameter	None
Return value	None
Required preconditions	None
Called functions	None

2.2.4 CAN_EnterTestMode function

Table 7. CAN_EnterTestMode function description

Function	CAN_EnterTestMode
Prototype	`void CAN_EnterTestMode(CAN_TypeDef* CAN, u16 TestMask);`
Behavior description	Switches the CAN into test mode. It sets the TEST bit in the control register and updates the Test register by storing its value with the mask. This function must be used in conjunction with CAN_LeaveTestMode().
Input parameter1	CAN: pointer to the CAN1 or CAN2 CAN cell.
Input parameter2	TestMask: specifies the configuration in test modes. Refer to the *TestMask on page 12* section for more details on the authorized values for this parameter.
Output parameter	None
Return value	None
Required preconditions	None
Called functions	None

TestMask

Specifies the CAN configuration in test modes. The different values are given in the following table:

Table 8. TestMask values

TestMask	Meaning
CAN_TESTR_LBACK	Loopback mode enabled
CAN_TESTR_SILENT	Silent mode enabled
CAN_TESTR_BASIC	Basic mode enabled

Example:

This example illustrates how to switch the CAN1 into loopback mode, that is, RX is disconnected from the bus, and TX is internally linked to RX.

```
{
CAN_EnterTestMode(CAN1, CAN_TESTR_LBACK);
}
```

2.2.5 CAN_LeaveTestMode function

Table 9. CAN_LeaveTestMode function description

Function name	CAN_LeaveTestMode
Prototype	`void CAN_LeaveTestMode(CAN_TypeDef* CAN)`
Behavior description	Leaves the current test mode (switches into normal mode). This function sets the TEST bit in the Control register to enable write access to the Test register, clears the LBACK, SILENT and BASIC bits in the Test register and clears the TEST bit in the Control register to disable write access to the Test register. This function must be used in conjunction with CAN_EnterTestMode().
Input parameter	CAN: pointer to the CAN1 or CAN2 CAN cell.
Output parameter	None
Return value	None
Required preconditions	None
Called functions	None

Example:

```
{
CAN_LeaveTestMode(CAN2);
}
```

2.2.6 CAN_SetBitrate function

Table 10. CAN_SetBitrate function description

Function name	CAN_SetBitrate
Prototype	`void CAN_SetBitrate(CAN_TypeDef* CAN, u16 bitrate);`
Behavior description	Sets up a standard CAN bitrate. This function writes the predefined timing value in the Bit Timing register and clears the BRPR register.
Input parameter1	CAN: pointer to the CAN1 or CAN2 CAN cell.
Input parameter2	BitRate: specifies the bitrate. Refer to the *Bitrate on page 13* section for more details on the authorized values of this parameter.
Output parameter	None
Return value	None
Required preconditions	CAN_EnterInitMode() must have been called before. The ST10 frequency = 40 MHz and the prescaler 1/2 is activated at the CAN clock generator input. See also *Section 2.2.7: CAN_SetTiming function on page 14*
Called Functions	None

Bitrate

The CAN bitrate values are given in *Table 4: CAN_Bitrate values on page 10*.

2.2.7 CAN_SetTiming function

Table 11. CAN_SetTiming function description

Function name	CAN_SetTiming
Prototype	`void CAN_SetTiming(CAN_TypeDef* CAN, u16 tseg1, u16 tseg2, u16 sjw, u16 brp);`
Behavior description	Sets up the CAN timing with specific parameters. It writes the timing value in the Bit Timing Register, from tseg1, tsg2, sjw and bits 5..0 of brp parameter and writes the BRPR register with bits 9..0 of brp parameter. All written values are the real values decremented by one unit.
Input parameter 1	CAN: pointer to the CAN1 or CAN2 CAN structure.
Input parameter 2	tseg1: Time segment before the sample point position. It can take values from 1 to 16.
Input parameter 3	tseg2: Time segment after the sample point position. It can take values from 1 to 8.
Input parameter 4	sjw: Synchronization jump width. It can take values from 1 to 4.
Input parameter 5	brp: Baud rate prescaler. It can take values from 1 to 1024.
Output parameter	None
Return value	None
Required preconditions	CAN_EnterInitMode() must have been called before.
Called functions	None

Example:

This example illustrates how to enable the configuration change bit, to be able to set the specific timing parameters: TSEG1=11, TSEG2=4, SJW=4, BRP=5

```
{
CAN_EnterInitMode(CAN1, CAN_CR_CCE);
CAN_SetTiming(CAN1, 11, 4, 4, 5);
CAN_LeaveInitMode( CAN1 );
}
```

2.2.8 CAN_SetUnusedMsgObj function

Table 12. CAN_SetUnusedMsgObj function description

Function name	CAN_SetUnusedMsgObj
Prototype	`ErrorStatus CAN_SetUnusedMsgObj(CAN_TypeDef* CAN, u16 msgobj);`
Behavior description	Configures the message object as unused / invalid. This function searches for a free message interface from IF0 and IF1, sets the WR/RD, Mask, Arb, Control, DataA and DataB bits in the Command Mask register, clears the Mask1 and Mask2 registers, clears the Arb1 and Arb2 register, clears the Message Control register, clears the DataA1, DataA2, DataB1, DataB2 registers and writes the value 1+msgobj in the Command Request register.
Input parameter 1	CAN: pointer to a CAN1 or CAN2 CAN structure.
Input parameter 2	msgobj: message object number, from 0 to 31.
Output parameter	None
Return value	An ErrorStatus enumeration value: – SUCCESS: Found interface to treat the message – ERROR: No interface found to treat the message
Required preconditions	None
Called functions	CAN_GetFreeIF()

Example:

This example illustrates how to invalidate the message objects from 16 to 31 of CAN1: these objects will not be used by the hardware.

```
{
for (i=16; i<=31; i++) CAN_SetUnusedMsgObj(CAN1, i);
}
```

2.2.9 CAN_SetUnusedAllMsgObj function

Table 13. CAN_SetUnusedAllMsgObj function description

Function name	CAN_SetUnusedAllMsgObj
Input parameter	CAN: pointer to a CAN1 or CAN2 CAN_TypeDef structure.
Prototype	ErrorStatus CAN_SetUnusedAllMsgObj(CAN_TypeDef* CAN);
Behavior description	Configures all the message objects as unused.
Output parameter	None
Return value	An ErrorStatus enumeration value: – SUCCESS: found interface to treat all messages – ERROR: no interface found to treat a message
Required preconditions	None
Called functions	CAN_SetUnusedMsgObj()

2.2.10 CAN_SetTxMsgObj function

Table 14. CAN_SetTxMsgObj function description

Function name	CAN_SetTxMsgObj
Prototype	`ErrorStatus CAN_SetTxMsgObj(CAN_TypeDef* CAN, u16 msgobj, u16 idType, FunctionalSate RemotEn)`
Behavior description	Configures the message object to transmit a message object with idType passed as a parameter. This function performs the following operations: – searches for a free message interface from IF0 and IF1. – Sets the WR/RD, Mask, Arb, Control, DataA and DataB bits in the Command Mask register. – Sets the masks to 7FF for standard ID and 0x1FFFFFFF for extended ID. – Sets the Mdir and MXtd – If remote functionality is enabled, then the US_MASK is not set in the message control register in order not to handle the remote frames received – Enables the message object interrupt. – Clears the data registers – Writes 1+msgobj into the command request register to start the transfer to the message RAM.
Input parameter 1	CAN: pointer to a CAN1 or CAN2 CAN_TypeDef structure.
Input parameter 2	msgobj: message object number, from 0 to 31.
Input parameter 3	idType: the identifier type of the frames that will be transmitted using this message object. The value is one of the following: CAN_STD_ID (standard ID, 11-bit) CAN_EXT_ID (extended ID, 29-bit)
Input parameter 4	RemoteEN: if enabled, the message object answers remote frames with matching ID. The value is one of the following: ENABLE: To enable the remote functionality DISBALE: To disable the remote functionality
Output parameter	None
Return value	An ErrorStatus enumeration value: – SUCCESS: Interface to handle the message – ERROR: No interface to handle the message
Required preconditions	None
Called functions	CAN_GetFreeIF()

Note: When defining which message object number to use for TX or RX, you must take into account the priority levels when processing the objects. The lowest number (0) has the highest priority and the highest number (31) has the lowest priority, whatever the type of the object. Also, for optimum performance, it is recommended not to have “holes” in the object list.

Example:

This example illustrates how to define transmit message object 0 of CAN1 with standard identifiers and, how to disable the answer to remote frames with matching ID.

```
{
CAN_SetTxMsgObj(CAN1, 0, CAN_STD_ID, DISABLE);
}
```

2.2.11 CAN_SetRxMsgObj function

Table 15. CAN_SetRxMsgObj function description

Function name	CAN_SetRxMsgObj
Input parameter 1	CAN: pointer to a CAN_TypeDef structure
Prototype	`ErrorStatus CAN_SetRxMsgObj(CAN_TypeDef* CAN, u16 msgobj, u16 idType, u32 idLow, u32 idHigh, bool singleOrFifoLast);`
Behavior description	Configures the message object as RX. This message receives all message objects with id between idLow and idHigh values and whose id type corresponds to the idType passed as a parameter. This function performs the following operations: – Searches for a free message interface from IF0 and IF1. – Sets the WR/RD, Mask, Arb, Control, DataA and DataB bits in the Command Mask register. – Writes the ID mask value generated by idLow and idHigh in the Mask1 and Mask2 registers, – Sets M_XTD (to filter according to IDE) and MDIR to filter remote frames. – Writes the ID arbitration value generated by idLow and idHigh in the Arb1 and Arb2 registers, set the MsgVal bit, and also Xtd bit in the Arb2 register if extended ID is used. – Sets the RxIE and UMask bits in the Message Control register, and also the EoB bit if the parameter singleOrFifoLast is TRUE. Clears the DataA1, DataA2, DataB1, DataB2 registers. – Writes the value 1+msgobj to the Command Request register to copy the selected registers into the message RAM.
Input parameter 2	msgobj: message object number, from 0 to 31.
Input parameter 3	idType: the identifier type of the frames that is transmitted using this message object. The value is one of the following: CAN_STD_ID (standard ID, 11-bit) CAN_EXT_ID (extended ID, 29-bit)
Input parameter 4	idLow: the lower part of the identifier range used for acceptance filtering. It can have values from 0 to 0x7FF for standard IDs, and values from 0 to 0x1FFFFFFF for extended IDs.
Input parameter 4	idHigh: the higher part of the identifier range used for acceptance filtering. It can have values from 0 to 0x7FF for standard IDs, and values from 0 to 0x1FFFFFFF for extended IDs. idHigh must be above idLow. For convenience, use one of the following values to set the maximum ID: CAN_LAST_STD_ID or CAN_LAST_EXT_ID
Input parameter 5	singleOrFifoLast: End-of-buffer indicator, it can have the following values: – TRUE for a single receive object or a FIFO receive object that is the last one in the FIFO – FALSE for a FIFO receive object that is not the last one
Output parameter	None
Return value	An ErrorStatus enumeration value: – SUCCESS: Interface to handle the message – ERROR: No interface to handle the message

Table 15. CAN_SetRxMsgObj function description (continued)

Function name	CAN_SetRxMsgObj
Required preconditions	None
Called functions	CAN_GetFreeIF()

Note: Care must be taken when defining an ID range: all combinations of idLow and idHigh will not always produce the expected result, because of the way identifiers are filtered by hardware. The criteria applied to keep a received frame is as follows: (received ID) AND (ID mask) = (ID arbitration), where AND is a bitwise operator.
Consequently, for idLow, it is generally better to choose a value with some LSBs cleared, and for idHigh, a value that "logically contains" idLow and with the same LSBs set. Example: the range 0x100-0x3FF will work, but the range 0x100-0x2FF will not work because 0x100 is not logically contained in 0x2FF (that is: 0x100 & 0x2FF = 0).

Example:

This example illustrates how to define a 2 message FIFO and acceptance filtering:

```
{
/*Define a receive FIFO of depth 2 (objects 0 and 1) in CAN1 for standard
identifiers, in which IDs are filtered in the range 0x400-0x5FF*/
CAN_SetRxMsgObj(CAN1, 0, CAN_STD_ID, 0x400, 0x5FF, FALSE);
CAN_SetRxMsgObj(CAN1, 1, CAN_STD_ID, 0x400, 0x5FF, TRUE);
/*Define a single receive object for extended identifiers, in which all IDs are
filtered in*/
CAN_SetRxMsgObj(CAN1, 2, CAN_EXT_ID, 0, CAN_LAST_EXT_ID, TRUE);
}
```

2.2.12 CAN_ReleaseMessage function

Table 16. CAN_ReleaseMessage function description

Function name	CAN_ReleaseMessage
Prototype	`ErrorStatus CAN_ReleaseMessage(CAN_TypeDef* CAN, u16 msgobj);`
Behavior description	Releases the message object. This function searches for a free message interface from IF0 and IF, sets the ClrIntPnd and TxRqst/NewDat bits in the Command Mask register and writes the 1+msgobj value to the Command Request register to copy the selected registers into the message RAM.
Input parameter 1	CAN: pointer to a CAN1 or CAN2 CAN_TypeDef structure.
Input parameter 2	msgobj: message object number, from 0 to 31.
Output parameter	None
Return value	An ErrorStatus enumeration value: – SUCCESS: Interface to treat the message – ERROR: No interface to treat the message
Required preconditions	None
Called functions	CAN_GetFreeIF()

Example:

This example illustrates how to release the message object 0 of CAN2.

```
{
CAN_ReleaseMessage(CAN2, 0);
}
```

2.2.13 CAN_SendMessage function

Table 17. CAN_SendMessage function description

Function name	CAN_SendMessage
Prototype	`ErrorStatus CAN_SendMessage(CAN_TypeDef* CAN, u16 msgobj, canmsg* pCanMsg);`
Behavior description	Updates the CAN message object with the pCanMsg fields and starts the transmission of the message.
Input parameter 1	CAN: Pointer to a CAN1 or CAN2 CAN_TypeDef structure.
Input parameter 2	msgobj: message object number, from 0 to 31.
Input parameter 3	pCanMsg: Pointer to the canmsg structure that contains the data to transmit: ID type, ID value, data length, data values.
Output parameter	None
Return value	An ErrorStatus enumeration value: – SUCCESS: Transmission started OK – ERROR: No transmission started
Required preconditions	The message object must have been set up properly.
Called functions	CAN_UpdateMsgObj() CAN_TransmitRequest()

Example:

This example illustrates how to send a standard ID data frame containing 4 data bytes.

```
{
canmsg CanMsg = { CAN_STD_ID, 0x111, 4, {0x10, 0x20, 0x40, 0x80} };
/*CAN1 Sends a standard ID data frame containing 4 data values*/
CAN_SendMessage(CAN1, 0, &CanMsg);
}
```

2.2.14 CAN_UpdateMsgObj function

Table 18. CAN_UpdateMsgObj function description

Function name	CAN_UpdateMsgObj
Prototype	`ErrorStatus CAN_SendMessage(CAN_TypeDef* CAN, u16 msgobj, canmsg* pCanMsg);`
Behavior description	Updates the CAN message object with pCanMsg fields.
Input parameter 1	CAN: Pointer to a CAN1 or CAN2 CAN_TypeDef structure.
Input parameter 2	msgobj: message object number, from 0 to 31.
Input parameter 3	pCanMsg: Pointer to the canmsg structure that contains the data to transmit: ID type, ID value, data length, data values.
Output parameter	None
Return value	An ErrorStatus enumeration value: – SUCCESS: IF found to treat the message and the pCanMsg id corresponds to the message idType configuration – ERROR: IF not found to treat the message or if the pCanMsg id does not correspond to the message idType configuration
Required preconditions	The message object must have been set up properly.
Called functions	CAN_GetFreeIF

2.2.15 CAN_TransmitRequest function

Table 19. CAN_TransmitRequest function description

Function name	CAN_TransmitRequest
Prototype	`ErrorStatus CAN_TransmitRequest(CAN_TypeDef* CAN, u16 msgobj);`
Behavior description	Starts the transmission of a message object. A data frame is transmitted if the message object is configured in transmission mode. A remote frame is transmitted if the message object is configured in reception mode.
Input parameter 1	CAN: Pointer to a CAN1 or CAN2 CAN_TypeDef structure.
Input parameter 2	msgobj: message object number, from 0 to 31.
Output parameter	None
Return value	An ErrorStatus enumeration value: – SUCCESS: IF found to treat the message – ERROR: IF not found to treat the message
Required preconditions	The message object must have been set up properly.
Called functions	CAN_GetFreeIF

2.2.16 CAN_ReceiveMessage function

Table 20. CAN_ReceiveMessage function description

Function name	CAN_ReceiveMessage
Prototype	`ErrorStatus CAN_ReceiveMessage(CAN_TypeDef* CAN, u16 msgobj, bool release, canmsg* pCanMsg);`
Behavior description	Gets the message, if received. This function performs the following operations: – Tests the bit corresponding to the message object number in the NewData registers. – Clears the RxOk bit in the Status register. – Copies the message contents from the message RAM to the registers and to the structure, and releases the message object if asked.
Input parameter 1	CAN: pointer to a CAN1 or CAN2 CAN_TypeDef structure.
Input parameter 2	msgobj: message object number, from 0 to 31.
Input parameter 3	Release: message release indicator, it can have the following values: – TRUE: the message object is released at the same time as it is copied from message RAM, then it is free for next reception – FALSE: the message object is not released, it is up to the caller to release it
Input parameter 4	pCanMsg: pointer to the canmsg structure where the received message is copied.
Output parameter	None
Return value	An ErrorStatus enumeration value: – SUCCESS: Reception OK – ERROR: No reception
Required preconditions	The message object must have been set up properly.
Called functions	CAN_GetFreeIF() CAN_GetMsgReceiveStatus()

Example:

This example illustrates how to receive a message.

```
{
  canmsg CanMsg;
/*Receive a message in the object 0 and ask for release*/
  if (CAN_ReceiveMessage(CAN1, 0, TRUE, &CanMsg))
  {
    /*Check or copy the message contents*/
  }
  else
  {
    /* Error handling*/
  }
}
```

2.2.17 CAN_BasicSendMessage function

Table 21. CAN_BasicSendMessage function description

Function name	CAN_BasicSendMessage
Prototype	`ErrorStatus CAN_BasicSendMessage(CAN_TypeDef* CAN, canmsg* pCanMsg);`
Behavior description	Starts transmission of a message in BASIC mode. This function performs the following operations: – Tests if the IF0 is free. – Writes the Arbitration, Message Control, DataA and DataB registers of message interface 0, with the message contents. – Writes the 1+msgobj value to the Command Request register to start the transmission This mode does not use the message RAM.
Input parameter 1	CAN: pointer to a CAN1 or CAN2 CAN_TypeDef structure.
Input parameter 2	pCanMsg: pointer to the canmsg structure that contains the data to transmit: ID type, ID value, data length, data values.
Output parameter	None
Return value	An ErrorStatus enumeration value: – SUCCESS: Transmission started OK – ERROR: No transmission
Required preconditions	The CAN must have been switched into BASIC mode.
Called functions	None

Example:

This example illustrates how to send frames.

```
{
/*Send consecutive data frames using message object 0*/
for (i = 0; i < 10; i++)
{
  CAN_SendMessage(CAN1, CanMsgTable[i]);
  CAN_WaitEndOfTx();
}
}
```

2.2.18 CAN_BasicReceiveMessage function

Table 22. CAN_BasicReceiveMessage function description

Function name	CAN_BasicReceiveMessage
Prototype	`ErrorStatus CAN_BasicReceiveMessage(CAN_TypeDef*CAN, canmsg* pCanMsg);`
Behavior description	Gets the message in BASIC mode, if received. This function performs the following operations: – Tests the NewDat bit in the Message Control register of message interface 1. – Clears the RxOk bit in the Status register. – Copies the message contents from the message interface 1 registers to the structure. This mode does not use the message RAM.
Input parameter 1	CAN: pointer to a CAN_TypeDef structure
Input parameter 2	pCanMsg: pointer to the canmsg structure where the received message is copied.
Output parameter	None
Return value	An ErrorStatus enumeration value: – SUCCESS: Reception OK – ERROR: No message pending
Required preconditions	The CAN must have been switched into BASIC mode.
Called functions	None

Example:

This example illustrates how to receive frame in basic mode.

```
{
 canmsg CanMsg;
/*Receive a message in BASIC mode*/
if (CAN_BasicReceiveMessage(CAN1, &CanMsg))
{
  /* Check or copy the message contents*/
}
else
{
  /* Error handling*/
}
}
```

2.2.19 CAN_WaitEndOfTx function

Table 23. CAN_WaitEndOfTx function description

Function name	CAN_WaitEndOfTx
Function prototype	`void CAN_WaitEndOfTx(CAN_TypeDef* CAN);`
Behavior description	Waits until current transmission is finished. This function should be called between two consecutive transmissions to ensure the latest frame has been completely transmitted on the bus, and therefore cannot be aborted anymore. A timeout should be added to avoid the infinite state if an error occurs.
Input parameter	Pointer to a CAN1 or CAN2 CAN structure.
Output parameter	None
Return value	None
Required preconditions	A message must have been sent before.
Called functions	None

2.2.20 CAN_GetMsgReceiveStatus

Table 24. CAN_GetMsgReceiveStatus function description

Function name	CAN_GetMsgReceiveStatus
Prototype	`FlagStatus CAN_GetMsgReceiveStatus(CAN_TypeDef* CAN, u16 msgobj);`
Behavior description	Tests the waiting status of a received message by reading the corresponding bit in the NewData 1 or 2 registers.
Input parameter 1	CAN: pointer to a CAN1 or CAN2 CAN_TypeDef structure.
Input parameter 2	msgobj: message object number, from 0 to 31.
Output parameter	None
Return value	1 if the corresponding message object has received a message awaiting to be copied, else 0.
Required preconditions	The corresponding message object must have been set as RX.
Called functions	None

Example:

This example illustrates how to test the new data registers for the message object 0.

```
{
/*Test if a message is pending in the receive message object 0*/
if (CAN_GetMsgReceiveStatus(CAN1, 0))
{
  /* Receive the message from this message object (i.e. get its data from message 
RAM)*/
}
}
```

2.2.21 CAN_GetMsgTransmiRequestStatus

Table 25. CAN_GetMsgTransmitRequestStatus function description

Function name	CAN_GetMsgTransmitRequestStatus
Prototype	`FlagStatus CAN_GetMsgTransmitRequestStatus(CAN_TypeDef* CAN, u16 msgobj);`
Behavior description	Tests the request status of a transmitted message. It reads the corresponding bit in the Transmission Request 1 or 2 registers.
Input parameter1	CAN: pointer to a CAN_TypeDef structure.
Input parameter2	msgobj: message object number, from 0 to 31.
Output parameter	None
Return value	1 if transmission of the corresponding message was requested, else 0.
Required preconditions	A message must have been sent before.
Called functions	None

Example:

This example illustrates how to test the transmit request.

```
{
/*Send a message using object 0*/
CAN_SendMessage(CAN1, 0, &CanMsg);
/*Wait for the end of transmit request*/
while (CAN_GetMsgTransmitRequestStatus(CAN1, 0));
/*Now, the message is being processed by the priority handler of the CAN cell, and
ready to be emitted on the bus*/
}
```

2.2.22 CAN_GetMsgInterruptStatus function

Table 26. CAN_GetMsgInterruptStatus function description

Function name	CAN_GetMsgInterruptStatus
Prototype	`FlagStatus CAN_GetMsgInterruptStatus(CAN_TypeDef* CAN, u16 msgobj);`
Behavior description	Tests the interrupt status of a message object. It reads the corresponding bit in the Interrupt Pending 1 or 2 registers.
Input parameter1	CAN: pointer to a CAN1 or CAN2 CAN_TypeDef structure.
Input parameter2	msgobj: message object number, from 0 to 31.
Output parameter	None
Return value	1 if the corresponding message has an interrupt pending, else 0.
Required preconditions	The interrupts must have been enabled.
Called functions	None

Example:

This example illustrates how to test interrupt pending.

```
{
/*Send a message using object 0*/
CAN_SendMessage(CAN1, 0, &CanMsg)
/* Wait for the TX interrupt*/
```

```
while (!CAN_GetMsgInterruptStatus(CAN1, 0));
}
```

2.2.23 CAN_GetMsgValidStatus function

Table 27. CAN_GetMsgValidStatus function description

Function name	CAN_GetMsgValidStatus
Prototype	`FlagStatus CAN_GetMsgValidStatus(CAN_TypeDef* CAN, u16 msgobj);`
Behavior description	Tests the validity of a message object (ready to use). It reads the corresponding bit in the Message Valid 1 or 2 registers. A valid object means that it has been set up either as TX or RX, and so is used by the hardware.
Input parameter1	CAN: pointer to a CAN1 or CAN2 CAN_TypeDef structure.
Input parameter2	msgobj: message object number, from 0 to 31.
Output parameter	None
Return value	1 if the corresponding message object is valid, else 0.
Required preconditions	None
Called functions	None

Example:

This example illustrates how to test the validity of message object 10 of CAN2 cell.

```
{
if (CAN_GetMsgValidStatus(CAN2, 10))
{
  /* Do something with message object 10*/
}
}
```

2.2.24 CAN_GetFlagStatus function

Table 28. CAN_GetFlagStatus function description

Function name	CAN_GetFlagStatus
Prototype	`FlagStatus CAN_GetFlagStatus(CAN_TypeDef* CAN, u16 CAN_Flag);`
Behavior description	Returns the status of the TxOK, RxOK, EPASS, EWARN and BOFF CAN flags.
Input parameter1	CAN: pointer to a CAN1 or CAN2 CAN_TypeDef structure.
Input parameter 2	CAN_Flag is one of the following parameters: – CAN_SR_TXOK – CAN_SR_RXOK – CAN_SR_EPASS – CAN_SR_EWRN – CAN_SR_BOFF
Output parameter	None
Return value	1 if the flag passed as a parameter is set; else 0
Required preconditions	None
Called functions	None

2.2.25 CAN_GetTransmitErrorCounter function

Table 29. CAN_GetTransmitErrorCounter function description

Function name	CAN_GetTransmitErrorCounter
Prototype	`u8 CAN_GetTransmitErrorCounter(CAN_TypeDef* CAN);`
Behavior description	Returns the transmit error counter contents.
Input parameter 1	CAN: pointer to a CAN1 or CAN2 CAN_TypeDef structure.
Output parameter	None
Return value	Transmit Error Counter value
Required preconditions	None
Called functions	None

2.2.26 CAN_GetReceiveErrorCounter function

Table 30. CAN_GetReceiveErrorCounter function description

Function name	CAN_GetReceiveErrorCounter
Prototype	`u8 CAN_GetReceiveErrorCounter(CAN_TypeDef* CAN);`
Behavior description	Returns the receive error counter contents.
Input parameter 1	CAN: pointer to a CAN1 or CAN2 CAN_TypeDef structure.
Output parameter	None
Return value	Receive Error Counter value
Required preconditions	None
Called functions	None

3 Revision history

Table 31. Document revision history

Date	Revision	Changes
04-Mar-2008	1	Initial release