Application Note, V1.0, Sep. 2007

AP29005

MultiCAN

CAN-Gateway Functionality Witout CPU Interaction

Microcontrollers

Never stop thinking

Edition 2007-09-28

Published by
Infineon Technologies AG
81726 München, Germany

LEGAL DISCLAIMER

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

AP29005
Revision History: 2007-09 V1.0
Previous Version: none

Page	Subjects (major changes since last revision)

We Listen to Your Comments

Any information within this document that you feel is wrong, unclear or missing at all?
Your feedback will help us to continuously improve the quality of this document.
Please send your proposal (including a reference to this document) to:

mcdocu.comments@infineon.com ✉

Table of Contents **Page**

1 Introduction

Today the CAN bus is widespread in many applications and not seldom you find more than one CAN bus in a system. These busses could run on different speeds and exchange different classes of information. For example in a vehicle you can find separated CAN busses for engine management, safety functions, body and chassis functions and infotainment.

For some reasons data from e.g. a sensor connected on one CAN bus is as well used by an ECU (electronic control unit) connected to a different CAN bus. To transfer the data it is required to pass over from one CAN bus to another. Usually a microcontroller handles the CAN messages and needs to control the reception and transmission.

The MultiCAN Module, implemented in different microcontroller products, offer a so called Gateway Mode to transfer CAN messages from on one CAN bus to another without CPU involvement. This offloads the CPU and displaces the workload to hardware. An intelligent and autarchic peripheral like the MultiCAN takes over this functionality.

This Application Note describes the basic Gateway Functionality and gives an example on how to setup the peripheral registers.

2 Gateway Functionality

The Gateway Mode allows establishing an automatic information transfer between two independent CAN bus systems without CPU interaction. For each CAN bus a different CAN-node is used. Each CAN-node can be setup and act independently in terms of bus-speed, used I/O pins and assigned message objects.

The Gateway Mode operates on message object level. In Gateway mode, information is transferred between two message objects, resulting in an information transfer between the two CAN nodes to which the message objects are allocated. A gateway may be established between any pair of CAN nodes and there may be as many gateways as there are message objects available to build the gateway structure. Usually you need to define a message object for a specific message direction (receive or transmit).

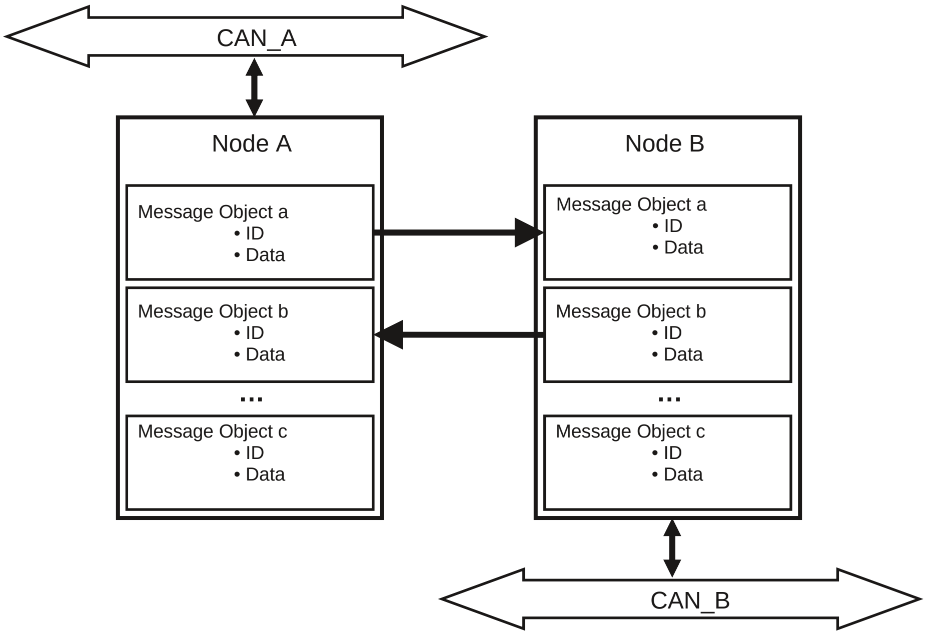

Figure 1 **Gateway between two CAN-busses**

3 CAN-Node Setup

For each CAN bus a separate CAN node is required. A CAN-node can be setup independently from each other. So each node can run its own bus speed. A very flexible port pin routing allows the adaptation to your PCB layout.

All CAN nodes share a common set of message objects, where each message object may be individually allocated to one of the CAN nodes. The message objects are organized in double chained lists, where each CAN node has its own list of message objects. A CAN node stores frames only into message objects that are allocated to the list of the CAN node. It only transmits messages from objects of this list.

A powerful, command driven list controller performs all list operations.

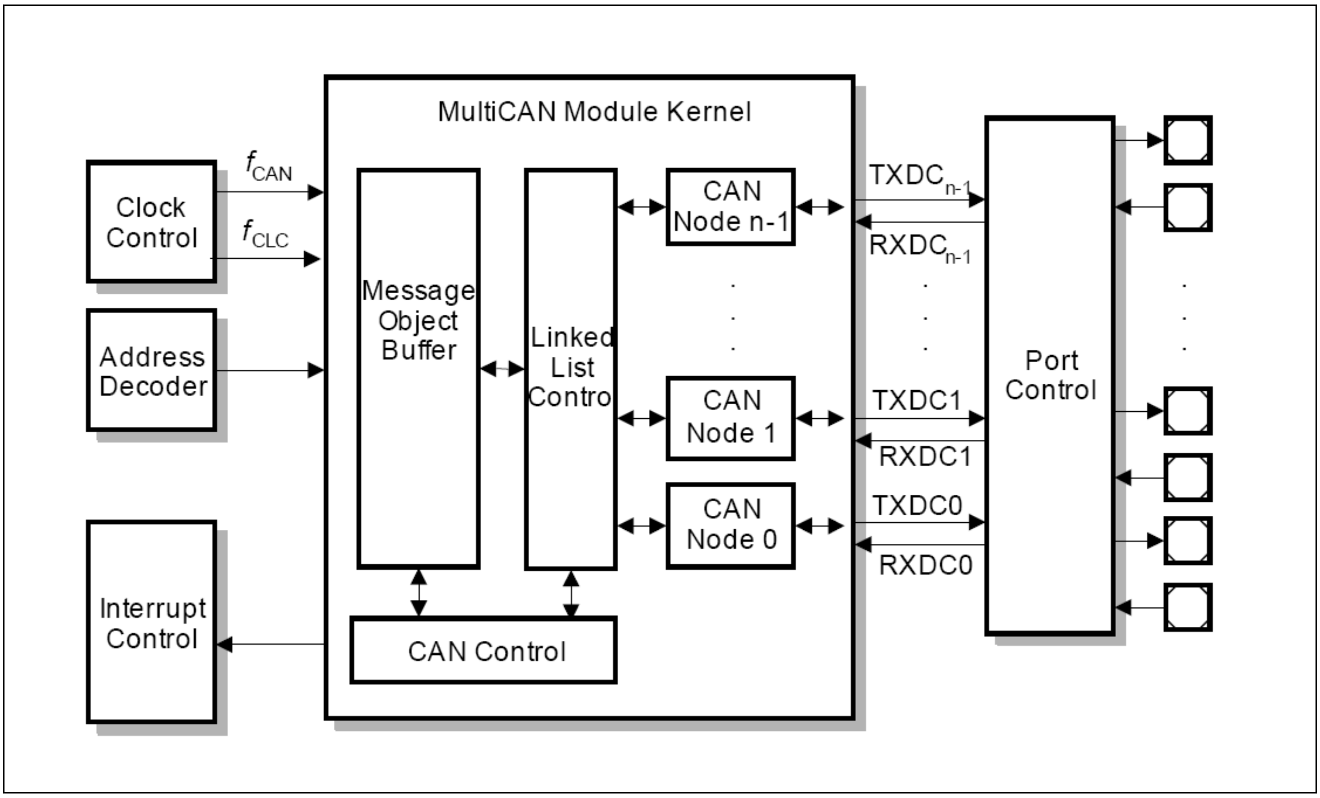

Figure 2 Overview of the MultiCAN Module

4 Message Object Setup

The Gateway Mode is selected in the Message Object Function Control register by the bit fields MMC. Another bit field (CUR pointer) defines the gateway destination message object. To complete the autonomous function you need to set the Gateway Data Frame Send bit (GDFS) in the Message Object Function Control Register. This forces a transmit request of the destination object after data transfer from the source message object.

The gateway destination object just needs to be valid (MSGVAL = 1), all other settings are not relevant for the information transfer from the source object to the destination object.

A gateway source object behaves like a standard message objects, but when a CAN frame has been received and stored in the source object, some additional actions are performed by the MultiCAN (Figure 3):

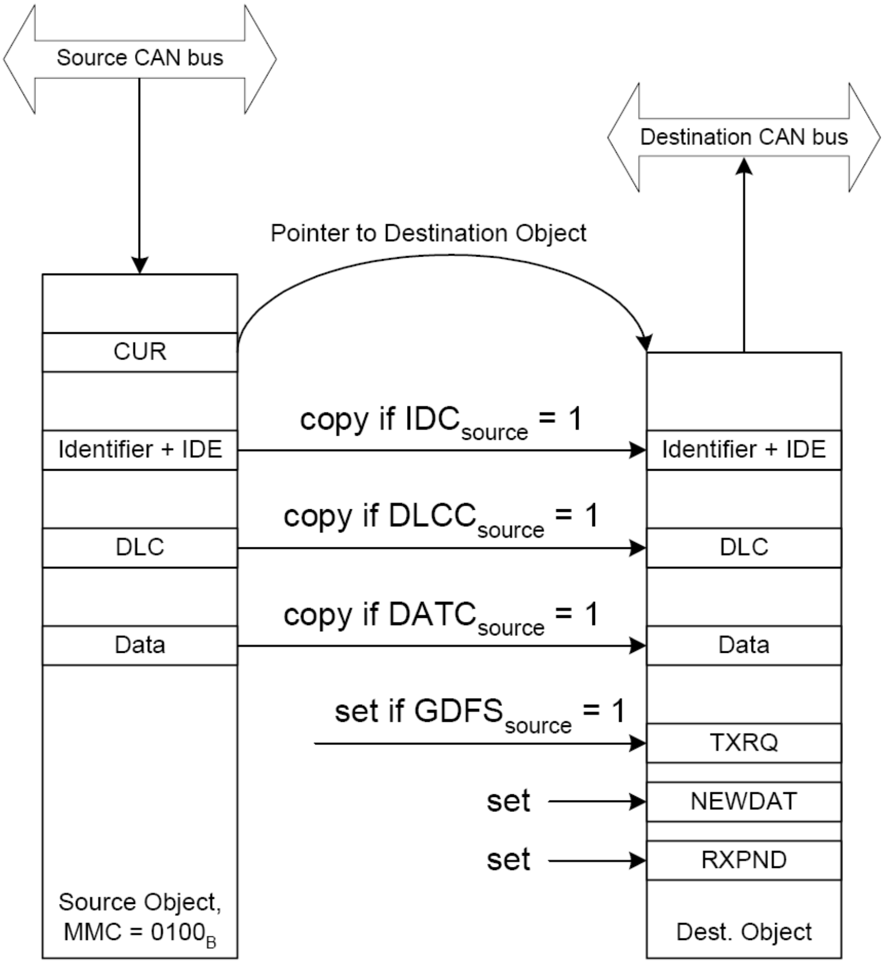

Figure 3 Gateway Transfer from Source to Destination

1. If bit DLCC is set in the Message Object Function Register of the source object, then the DLC code is copied from the source object to the destination object.
2. If bit IDC is set in the Message Object Function Register of the source object, then the identifier and the IDE bit are copied from the source object to the destination object.
3. If bit DATC is set in the Message Object Function Register of the source object, then the data field is copied from the source object to the destination object.
4. If bit GDFS is set in the Message Object Function Register of the source object, then TXRQ is set in the Message Object Control Register of the destination object.
5. RXPND and NEWDAT are set in the Message Object Control Register of the destination object.
6. A message interrupt request is generated for the destination object if RXIE is set in the Message Object Control Register of the destination object.
7. The current pointer CUR in the FIFO/Gateway Pointer Register of the source object is moved to the next destination object according to the FIFO rules. A gateway with a single (static) destination object is obtained by means of setting TOP = BOT = CUR = destination object.

The Gateway functionality is in addition to all other settings for a Message Object like identifier, identifier selection, acceptance mask, message direction, data length, data field, priority class and single transmittion features.

5 Extensions to the Gateway Functionality

5.1 FIFO Structure

In case of a series of CAN frames in time a FIFO can be added on the destination site. The link from the source to the destination object works in the same way as the link from a FIFO source to a FIFO slave. This means that a gateway with an integrated destination FIFO may be created.

5.2 Foreign Remote Requests

When a remote frame received on a CAN node is stored in a message object, then a transmit request is set in order to trigger the answer (data frame transmission) to the request or to automatically issue a secondary request. If bit FRREN is cleared (FRREN = 0) in the Function Control register of the message object where the remote request is stored, then TXRQ is set in the Control Register of the same message object. If bit FRREN is set (FRREN = 1: foreign remote request enabled) then TXRQ is set in the message object that is referenced by pointer CUR in the FIFO/Gateway Pointer Register. The value of CUR is, however, not changed by this feature. Although the foreign remote request feature works independently from the selected message mode, it is especially useful for gateways to issue a remote request on the source of a gateway upon the reception of a remote request on the gateway destination.

According to the setting of FRREN in the gateway destination object there are two ways to handle remote requests that appear on the destination side (assuming that the source object is a receive object and the destination is a transmit object, i.e. DIRsource = 0 and DIRdestination = 1):

FRREN = 0 in the Gateway Destination Object

1. A remote frame is received by gateway destination.
2. TXRQ is set automatically in the gateway destination object.
3. A data frame with the current data stored in the destination object is transmitted on the destination bus.

FRREN = 1 in the Gateway Destination Object

1. A remote frame is received by gateway destination.
2. TXRQ is set automatically in the gateway source object (must be referenced by CUR pointer of the destination object).
3. A remote request is transmitted by the source object (which is a receive object) on the source CAN bus.
4. The receiver of the remote request responds with a data frame on the source bus.
5. The data frame is stored in the source object.
6. The data frame is copied to the destination object (gateway action).
7. TXRQ is set in the destination object (assuming GDFSsource = 1).
8. The new data stored in the destination object is transmitted on the destination bus, as response to the initial remote request on the destination bus.

5.3 Interrupt and Error Handling

The Gateway Functionality is by intention without any CPU involvement. This offloads the CPU from low level data handling. In some cases a notification of the CPU by interrupt is required e.g. in cases of special received or transmitted frames as well in case of an error handling.

Using the Gateway Functionality allows the same receive, transmit and error interrupt mechanisms as using without Gateway Functionality

6 Example with XC2287

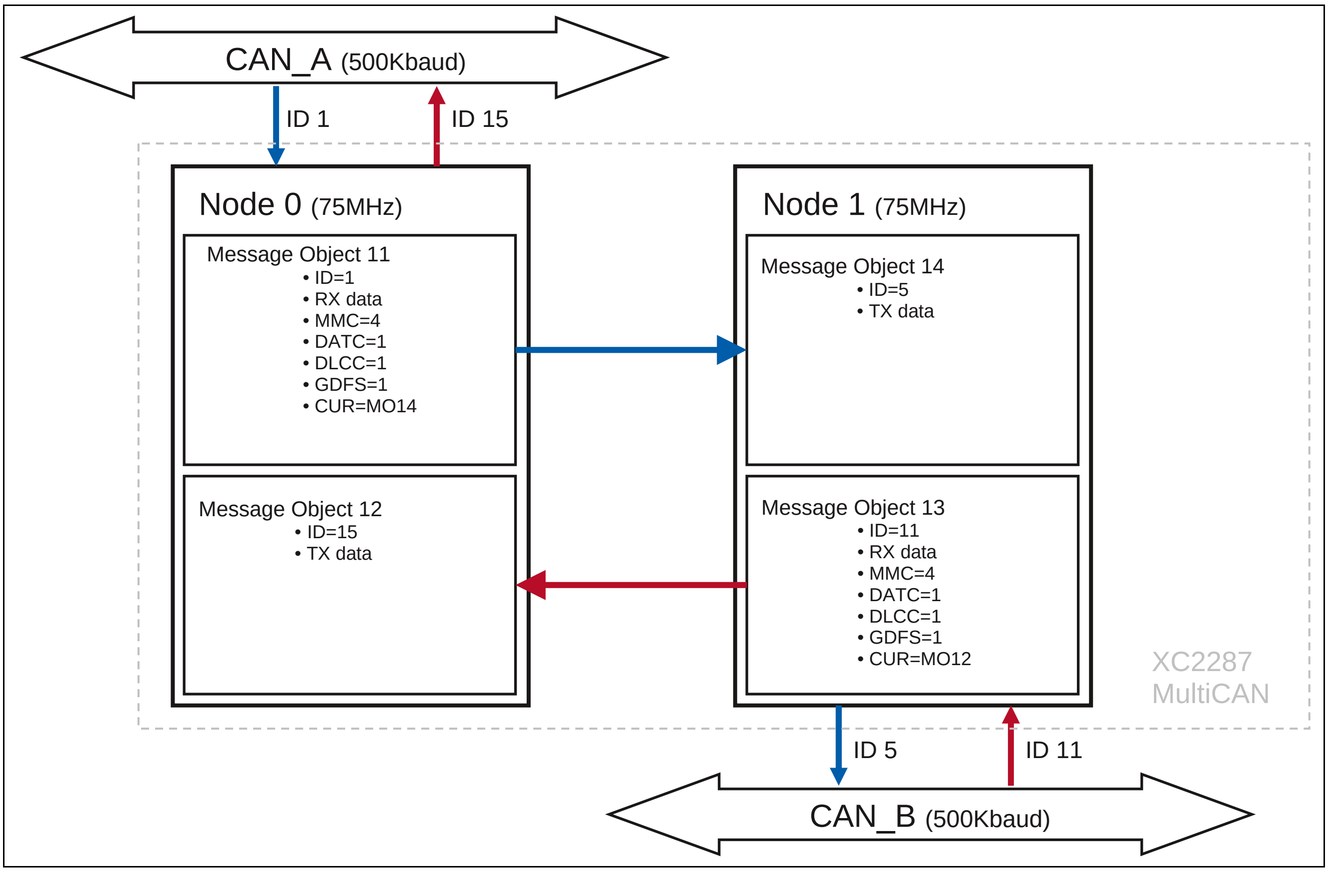

Figure 4 Example of a Gateway with ID Modification

The following C-code shows how to setup the MultiCAN for a CAN Gateway Functionality. It’s an initialization code written for a XC2287 device.

The MultiCAN module is found on several 8-,16- and 32-bit devices. The MultiCAN register length is based on 32-bit. That means for 16- and 8-bit devices the MultiCAN registers need be divided into several portions like high/low-word for 16-bit.

```
//****************************************************************************
// @Module        MultiCAN Controller
// @Filename      CAN.c
// @Project
//----------------------------------------------------------------------------
// @Controller    Infineon XC2287
// @Compiler
// @Description   This file contains functions that use the CAN module.
//
//                Node0:    500kBaud, use CAN1 output connector
//                                 MO11: RX, ID01 -> Node1, MO14
//                                 MO12: TX, ID15 <- Node1, MO13
//
//                Node1:    500kBaud, use CAN2 output connector
//                                 MO13: RX, ID11 -> Node0, MO12
//                                 MO14: TX, ID05 <- Node0, MO11
//
// Version:  01   2007-06-13
//
//----------------------------------------------------------------------------
// @Date          xxx
//****************************************************************************

#include "CAN.h"

#define CAN_PANCTR_BUSY         0x0100
#define CAN_INIT_LIST           0x02


//****************************************************************************
// @Function      void CAN_vInit_MCAN(void)
//----------------------------------------------------------------------------
// @Description   This is the initialization function of the CAN function
//                library. It is assumed that the SFRs used by this library
//                are in their reset state.
//----------------------------------------------------------------------------
// @Returnvalue   None
//----------------------------------------------------------------------------
// @Parameters    None
//----------------------------------------------------------------------------
// @Date
//****************************************************************************
void CAN_vInit_MCAN(void)
{
uword uwDummy;

  ///  -----------------------------------------------------------------------
  ///  Configuration of the Module Clock:
  ///  -----------------------------------------------------------------------
  Sys_Protection(0);
  MCAN_KSCCFG     = 0x0003;   // Module enable
  uwDummy = MCAN_KSCCFG;     // Dummy read
  CAN_FDRL        =  0x83C0;  // Fraction divider mode --> fcan = fsys*step/1024
= 75MHz
  Sys_Protection(1);

}       //End of CAN_vInit_MCAN
```

```
//****************************************************************************
// @Function      void CAN_vInit_Node0(void)
//----------------------------------------------------------------------------
// @Description   This is the initialization function of the CAN function
//                library. It is assumed that the SFRs used by this library
//                are in their reset state.
//----------------------------------------------------------------------------
// @Returnvalue   None
//----------------------------------------------------------------------------
// @Parameters    None
//----------------------------------------------------------------------------
// @Date
//****************************************************************************
void CAN_vInit_Node0(void)
{
  ///  -----------------------------------------------------------------------
  ///  Configuration of CAN Node 0:
  ///  -----------------------------------------------------------------------
  ///  General Configuration of the Node 0:
  ///  - set INIT and CCE
  CAN_NCR0       =  0x0041;  // load node 0 control register    SUSEN=1

  ///  Configuration of the used CAN Port Pins: I/O PIN Configuration for CAN 1
  CAN_NPCR0 = 0x0002;  // P2.0 Rx = receive input C NO LOOP BACK

  ///  Configuration of the used CAN Port Pins: I/O PIN Configuration for CAN 1
  P2_IOCR00  = 0x0020;   // P2.0 as input (pull up)
  P2_IOCR01  = 0x0090;   // P2.1 as output (ALT1, push pull)

    ///  Configuration of the Node 0 Baud Rate:
  ///  - required baud rate = 500,000 kbaud
  CAN_NBTR0L  =  BAUD_0500_000_WITH_MHZ_75;  //  Set  Baud  Rate  of  Node  0  at  500
kbaud at 75 MHz

  ///  -----------------------------------------------------------------------
  ///  Configuration of the CAN Message Object List Structure:
  ///  -----------------------------------------------------------------------
  ///  Allocate MOs for list 1/Node0:
  SetListCommand(1, 11, CAN_INIT_LIST);
  SetListCommand(1, 12, CAN_INIT_LIST);

}      //End of CAN_vInit_Node0



//****************************************************************************
// @Function      void CAN_vInit_Node1(void)
//----------------------------------------------------------------------------
// @Description   This is the initialization function of the CAN function
//                library. It is assumed that the SFRs used by this library
//                are in their reset state.
//----------------------------------------------------------------------------
// @Returnvalue   None
//----------------------------------------------------------------------------
// @Parameters    None
//----------------------------------------------------------------------------
// @Date
                                                                 n
//****************************************************************************
void CAN_vInit_Node1(void)
{
  ///  -----------------------------------------------------------------------
  ///  Configuration of CAN Node 1:
  ///  -----------------------------------------------------------------------
  ///  General Configuration of the Node 1:
  ///  - set INIT and CCE

  CAN_NCR1       =  0x0041;  // load node 1 control register   SUSEN=1
```

```
  CAN_NIPR1      =  0x0000;  // load node 1 interrupt pointer register

  ///  Configuration of the used CAN Port Pins: I/O PIN Configuration for CAN 1
  CAN_NPCR1 = 0x0000;   // P2.4 Rx = receive input A + NO LOOP BACK

  P2_IOCR04  = 0x0020;   // P2.4 as input (pull up)
  P2_IOCR02  = 0x0090;   // P2.2 as output (ALT1, push pull)

  ///  Configuration of the Node 1 Baud Rate:
  CAN_NBTR1L  = BAUD_0500_000_WITH_MHZ_75;  // Set Baud Rate of Node 1 at 500
Kbaud at 75 MHz

  ///  ------------------------------------------------------------------------
  ///  Configuration of Service Request Nodes 0 - 15:
  ///  ------------------------------------------------------------------------
                  //IE<<6 | ILEV<<2 | GLEV
  CAN_0IC        =  (1<<6) | (5<<2) | 0;

  ///  ------------------------------------------------------------------------
  ///  Configuration of the CAN Message Object List Structure:
  ///  ------------------------------------------------------------------------
  ///  Allocate MOs for list 2/Node1:
  SetListCommand(2, 13, CAN_INIT_LIST);
  SetListCommand(2, 14, CAN_INIT_LIST);

} //  End of function CAN_vInit_Node1



//*****************************************************************************
// @Function      void CAN_vInit_MessageObjects (void)
//-----------------------------------------------------------------------------
// @Description   This is the initialization function of the CAN function
//                library. It is assumed that the SFRs used by this library
//                are in their reset state.
//-----------------------------------------------------------------------------
// @Returnvalue   None
//-----------------------------------------------------------------------------
// @Parameters    None
//-----------------------------------------------------------------------------
// @Date

//*****************************************************************************
void CAN_vInit_MessageObjects(void)
{

  ///  ------------------------------------------------------------------------
  ///  Configuration of Message Object 11:
  ///  ------------------------------------------------------------------------
  ///  - message object 11 is valid
  ///  - message object is used as receive object
  ///  - this message object is assigned to list 1 (node 0)
  CAN_MOCTR11L     =  0x0000;  // load MO11 control register
  CAN_MOCTR11H     =  0x0080;  // load MO11 control register

  CAN_MOFCR11L     =  0x0D04;  // load MO11 function control register
  CAN_MOFCR11H     =  0x0100;  // MO11 function control register

  CAN_MOFGPR11L    =  0x0E0E;  // load MO11 function control register
  CAN_MOFGPR11H    =  0x0E0E;  // MO11 function control register

  CAN_MOAR11L      =  0x0000;  // load MO11 arbitration register
  CAN_MOAR11H      =  0x8004;  // MO11 arbitration register

  CAN_MODATA11LL   =  0x1111;  // load MO11 data register low
  CAN_MODATA11LH   =  0x1111;  // load MO11 data register low
  CAN_MODATA11HL   =  0x1111;  // load MO11 data register high
  CAN_MODATA11HH   =  0x1111;  // load MO11 data register high
```

```
CAN_MOCTR11L     =  0x0000;
CAN_MOCTR11H     =  0x0020;  // set MSGVAL - enable mssg obj

///  -----------------------------------------------------------------------
///  Configuration of Message Object 12:
///  -----------------------------------------------------------------------
///  - message object 12 is valid
///  - message object is used as transmit object
///  - this message object is assigned to list 1 (node 0)
CAN_MOCTR12L     =  0x0000;  // load MO12 control register
CAN_MOCTR12H     =  0x0E88;  // load MO12 control register

CAN_MOFCR12L     =  0x0000;  // load MO12 function control register
CAN_MOFCR12H     =  0x0100;  // MO12 function control register

CAN_MOFGPR12L    =  0x0000;  // load MO12 function control register
CAN_MOFGPR12H    =  0x000D;  // MO12 function control register

CAN_MOAR12L      =  0x0000;  // load MO12 arbitration register
CAN_MOAR12H      =  0x803C;  // MO12 arbitration register

CAN_MODATA12LL   =  0x1212;  // load MO12 data register low
CAN_MODATA12LH   =  0x1212;  // load MO12 data register low
CAN_MODATA12HL   =  0x1212;  // load MO12 data register high
CAN_MODATA12HH   =  0x1212;  // load MO12 data register high

CAN_MOCTR12L     =  0x0000;
CAN_MOCTR12H     =  0x0020;  // set MSGVAL - enable mssg obj

///  -----------------------------------------------------------------------
///  Configuration of Message Object 13:
///  -----------------------------------------------------------------------
///  - message object 13 is valid
///  - message object is used as receive object
///  - this message object is assigned to list 2 (node 1)
CAN_MOCTR13L     =  0x0000;  // load MO13 control register
CAN_MOCTR13H     =  0x0080;  // load MO13 control register

CAN_MOFCR13L     =  0x0D04;  // load MO13 function control register
CAN_MOFCR13H     =  0x0100;  // MO13 function control register

CAN_MOFGPR13L    =  0x0C0C;  // load MO13 function control register
CAN_MOFGPR13H    =  0x0C0C;  // MO13 function control register

CAN_MOAR13L      =  0x0000;  // load MO13 arbitration register
CAN_MOAR13H      =  0x802C;  // MO13 arbitration register

CAN_MODATA13LL   =  0x0202;  // load MO13 data register low
CAN_MODATA13LH   =  0x0202;  // load MO13 data register low
CAN_MODATA13HL   =  0x0202;  // load MO13 data register high
CAN_MODATA13HH   =  0x0202;  // load MO13 data register high

CAN_MOCTR13L     =  0x0000;
CAN_MOCTR13H     =  0x0020;  // set MSGVAL - enable mssg obj

///  -----------------------------------------------------------------------
///  Configuration of Message Object 14:
///  -----------------------------------------------------------------------
///  - message object 14 is valid
///  - message object is used as transmit object
///  - this message object is assigned to list 2 (node 1)
CAN_MOCTR14L     =  0x0000;  // load MO14 control register
CAN_MOCTR14H     =  0x0E88;  // load MO14 control register

CAN_MOFCR14L     =  0x0000;  // load MO14 function control register
CAN_MOFCR14H     =  0x0100;  // MO14 function control register
```

```
  CAN_MOFGPR14L    =  0x0000;  // load MO14 function control register
  CAN_MOFGPR14H    =  0x000B;  // MO14 function control register

  CAN_MOAR14L      =  0x0000;  // load MO14 arbitration register
  CAN_MOAR14H      =  0x8014;  // MO14 arbitration register

  CAN_MODATA14LL   =  0x1414;  // load MO14 data register low
  CAN_MODATA14LH   =  0x1414;  // load MO14 data register low
  CAN_MODATA14HL   =  0x1414;  // load MO14 data register high
  CAN_MODATA14HH   =  0x1414;  // load MO14 data register high

  CAN_MOCTR14L     =  0x0000;
  CAN_MOCTR14H     =  0x0020;  // set MSGVAL - enable mssg obj

} //  End of function CAN_vInit_MessageObjects


//****************************************************************************
// @Function      void CAN_vStartNodes (void)
//----------------------------------------------------------------------------
// @Description   This is the initialization function of the CAN function
//                library. It is assumed that the SFRs used by this library
//                are in their reset state.
//----------------------------------------------------------------------------
// @Returnvalue   None
//----------------------------------------------------------------------------
// @Parameters    None
//----------------------------------------------------------------------------
// @Date
                                                                n
//****************************************************************************
void CAN_vStartNodes(void)
{

  //   -----------------------------------------------------------------------
  //   Start the CAN Nodes:
  //   -----------------------------------------------------------------------

  CAN_NCR0      &=  ~(uword)0x0041;   // reset INIT and CCE
  CAN_NCR1      &=  ~(uword)0x0041;   // reset INIT and CCE

} //  End of function CAN_vStartNodes


//****************************************************************************
// @Function      void CAN_vTransmit(ubyte ubObjNr)
//----------------------------------------------------------------------------
// @Description   This function triggers the CAN controller to send the
//                selected message.
//                If the selected message object is a TRANSMIT OBJECT then
//                this function triggers the sending of a data frame. If
//                however the selected message object is a RECEIVE OBJECT
//                this function triggers the sending of a remote frame.
//----------------------------------------------------------------------------
// @Parameters    ubObjNr:
//                Number of the message object (0-127)
//****************************************************************************
void CAN_vTransmit(ubyte ubObjNr)
{
  CAN_HWOBJ[ubObjNr].MO_CTRL.MOCTRLn.uwRegister = 0x0000;
  CAN_HWOBJ[ubObjNr].MO_CTRH.MOCTRHn.uwRegister = 0x0100;   // set TXRQ
} //  End of function CAN_vTransmit
```

```
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
// XC2287_Multican_SetListCommand()
//
// set list command (append object into the list)
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
void SetListCommand(ubyte PANAR2, ubyte PANAR1, ubyte PANCMD)
{
  while ( (CAN_PANCTRL & CAN_PANCTR_BUSY) ); // wait as long as the CAN is busy
  CAN_PANCTRH = (PANAR2 << 8) | PANAR1;
  CAN_PANCTRL = PANCMD;
}
```

7 Abbreviations

CAN	Controller Area Network, a message object oriented serial communication protocol
CUR	Current Object Pointer; links to the actual target object within a FIFO/Gateway structure.
DATC	Data Copy; the data field of the gateway source object (after storing the received frame in the source) is copied to the gateway destination.
DLCC	Data Length Code Copy; data length code of the gateway source object (after storing the received frame in the source) is copied to the gateway destination.
ECU	Electronic Control Unit
GDFS	Gateway Data Frame Send; a transmit request is set in the gateway destination object after the transfer of a data frame from the gateway source to the gateway destination.
IDC	Identifier Copy; the identifier of the gateway source object (after storing the received frame in the source) is copied to the gateway destination.
MMC	Message Mode Control; controls the functionality of the message object.
MO	CAN Message object

8 Related Documents

User’s Manual, V1.0, June 2007, Vol 1, System Unit

User’s Manual, V1.0, June 2007, Vol 2, Peripheral Unit

http://www.infineon.com

Published by Infineon Technologies AG