Application Note, V1.0, Jan. 2008

AP16131

XC2000/XE166

FIFO Configuration of MultiCAN using DAvE

Microcontrollers

Never stop thinking

Edition 2008-02

Published by
Infineon Technologies AG
81726 Munich, Germany

© 2008 Infineon Technologies AG
All Rights Reserved.

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 be used in life-support devices or systems only 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.

AP16131

Revision History:	2008-01	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 5

2 Message Object FIFO Structure........ 6
2.1 Receive FIFO 7
2.2 Transmit FIFO 8

3 CAN-Node Setup 9

4 FIFO Example setup with XC2287 Starter Kit........ 10

5 FIFO Configuration using DAvE 11

6 Example code for XC2287 19
6.1 Functions CAN_ubWriteFIFO( ) and CAN_ubReadFIFO( ) in Test Applications. 27

7 Abbrevation 29

8 Related Documents and Links........ 30

1 Introduction

Increasing demand for additional features along with the replacement of mechanics by electronic systems drives the number of nodes and bus systems in a vehicle. This increases complexity and challenges for data availability to the different systems within the vehicle.

The CAN network in in-vehicle becomes increasingly complicated, both in terms of the number of controllers and the speed at which they communicate (Data transfer rate up to 1MBaud).

In case of high CPU load it may be difficult to process a series of CAN frames in time. This may happen for the short term reception of multiple messages as well as the transmission of a series with a tight due date.

Therefore a FIFO buffer structure has been implemented in order to avoid loss of incoming messages and to minimize the setup time for outgoing messages. The FIFO structure may also be used to automate the reception or transmission of a series of CAN messages and to generate a single message interrupt when the whole series is done.

This Application Note describes the basic FIFO Functionality and gives an example on how to setup the FIFO using DAvE for XC2287 device.

2 Message Object FIFO Structure

There may be as many FIFOs in parallel as are required by the application. The number of FIFOs and their size are only limited by the number of available message objects. A FIFO may be installed, resized and deinstalled at any time, even during CAN operation.

The basic structure of a FIFO is shown in **Figure 1**. A FIFO consists of a single base object (shown on the left side) and several slave objects (shown on the right side). The slave objects are chained together in one or a list structure. The base object may be allocated to any list. Although Figure 1 shows the base object as a separate item apart from the slave objects, it is also possible to integrate the base object at any place into the chain of slave objects, so that the base object is also a slave object (not possible for gateways). The FIFO structure fully relies on the list structure. The absolute object numbers of the message objects have no impact on the operation of the FIFO.

The base object need not be allocated to the same list as the slave objects. Only the slave object must be allocated to a common list (as they are chained together). The BOT, CUR and TOP pointer link the base object to the slave objects, no matter whether the base object is allocated to the same or to another list than the slave objects.

The absolute minimum FIFO would consist of a single message object which is both FIFO base and FIFO slave (not very useful). The biggest possible FIFO would use all message objects of the MultiCAN module. Any sizes between these extremes are possible.

In the FIFO base object the boundaries of the FIFO are defined. The BOT field in the FIFO/Gateway Pointer Register MOFGPRn of the base object points to the first (bottom) slave element in the FIFO. The TOP field in the FIFO/Gateway Pointer Register of the base object points to the last (top) slave element.

The CUR field in the FIFO/Gateway Pointer Register of the FIFO base object points to the actual slave object selected by the MultiCAN for message transfer. When a message transfer takes place with this object then CUR is moved to the next position. If CUR has already reached the top of the FIFO (CUR = TOP) then it is wrapped around to the bottom of the FIFO (CUR: = BOT). Otherwise CUR is moved to the next message object in the list structure of the slave objects (CUR: = PNEXT of current object). This scheme yields a circular FIFO structure where the fields BOT and TOP just establish the link from the last to the first element, which is missing in the linear structure.

The SEL field in the FIFO/Gateway Pointer register of the base object may be used for monitoring purposes. It allows selection of any slave object and to generate a message interrupt if the CUR pointer reaches the value of SEL. Thus SEL offers a convenient way to determine the end of a predefined series of message transfers, or it may be used to issue a warning to the CPU when the FIFO gets full.

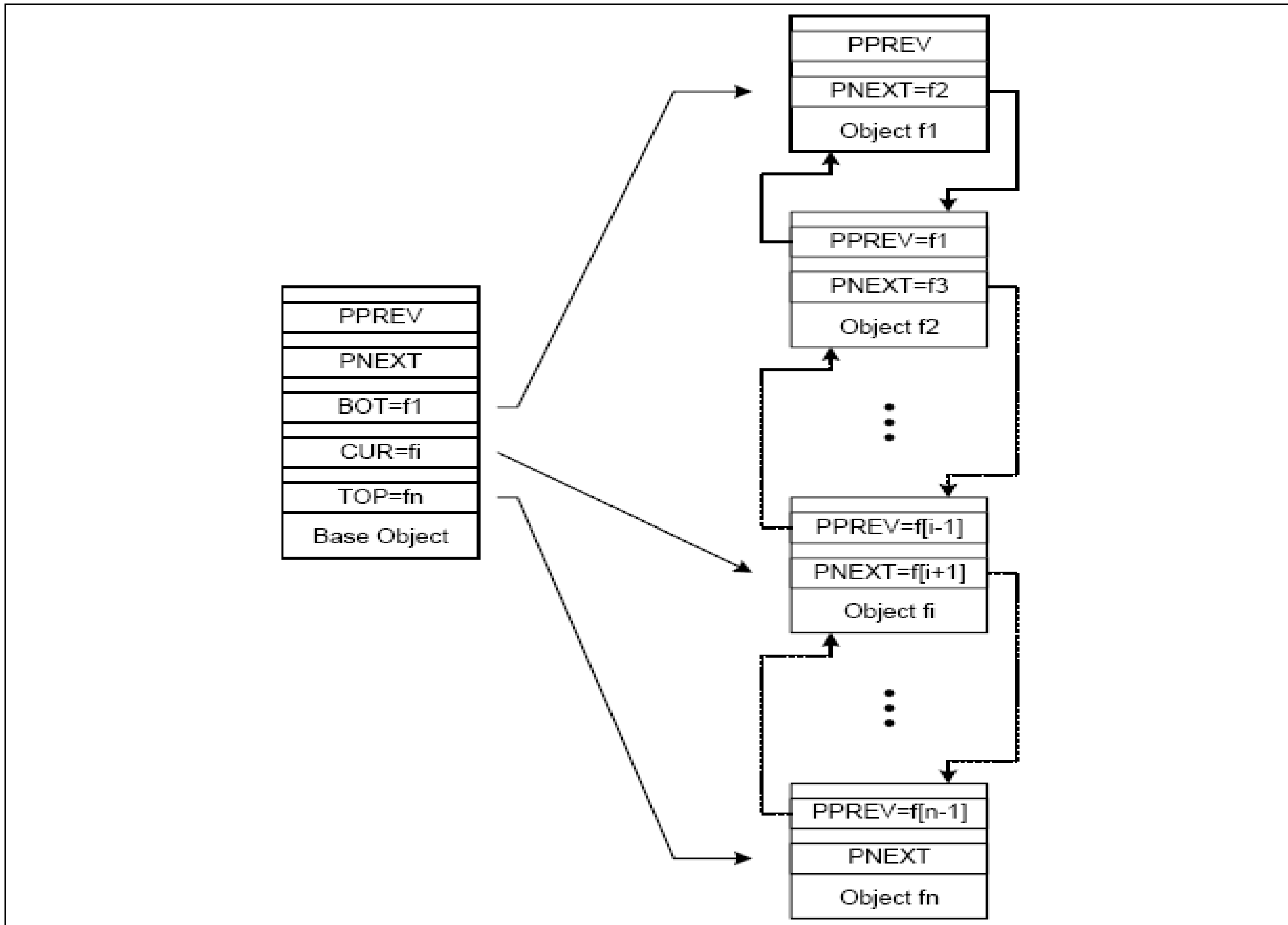

Figure 1 FIFO Structure with FIFO Base and n FIFO Destinations (Slaves)

2.1 Receive FIFO

The Receive FIFO structure is used to buffer incoming (received) remote or data frames.

- A Receive FIFO base object is selected via MMC = 0001_B in the Message Object Function Control Register MOFCRn.
- The message mode of the FIFO slave objects are not relevant (MOFCRn.MMC = 0000_B is implicitly assumed for FIFO slaves) for the operation of the Receive FIFO.
- When the FIFO base object receives a frame from the CAN node it belongs to, then the frame is stored in the message object selected by the pointer of CUR (not stored in the base object).
- There is also no extra acceptance filtering to match the received frame against the identifier, IDE bit and DIR bit of the slave object.
- If bit OVIE is set in the Message Object Function Register MOFCRn of the FIFO base object and the pointer CUR reaches the value stored in SEL then a FIFO overflow interrupt request is generated. (The interrupt request is generated on interrupt output line TXINP (TXINP of the base object) immediately after the storage of the received frame into the slave object. Transmit interrupts are still generated if TXIE is set.)
- A CAN message is stored in a FIFO slave only if MSGVAL = 1 in both FIFO base and slave object.

2.2 Transmit FIFO

The Transmit FIFO structure is used to buffer a series of data or remote frames to be transmitted. A transmit FIFO consists of one base message object and one or more slave message objects.

- A Transmit FIFO base object is selected via MMC = 0010_B in the Message Object Function Control Register of the FIFO base object.
- The Transmit FIFO slave objects is selected via MMC = 0011_B.
- The TXEN1 bits of all message objects except the one which is selected by the CUR pointer of the base object must be cleared by user. TXEN1 of the message object selected by CUR must be set.
- When tagging the message objects of the FIFO valid to start the operation of the FIFO then the base object must be tagged valid (MSGVAL := 1) first.
- When a Transmit FIFO shall be deinstalled during operation, then the slave objects must be tagged invalid (MSGVAL := 0) first.
- Transmit acceptance filtering evaluates TXEN1 for each message object and a message object may win transmit acceptance filtering only if TXEN1 is set.
- When a FIFO element has transmitted a message then in addition to standard transmit postprocessing (clear TXRQ, transmit interrupt etc.) the MultiCAN clears TXEN1 in that message object and moves the CUR pointer in the corresponding FIFO base object to the next message object to be transmitted.
- TXEN1 is set automatically in the next message object. Thus TXEN1 moves along the FIFO structure like a token to select the active element.
- If bit OVIE is set in the Message Object Function Register of the FIFO base object and the pointer CUR reaches the value stored in SEL then a FIFO overflow interrupt request is generated. (The interrupt request is generated on interrupt output line as defined by RXINP (RXINP of the base object) when postprocessing of the received frame is done. Receive interrupts are still generated for the Transmit FIFO base object if bit RXIE is set).

3 CAN-Node Setup

For a CAN bus minimum two (more) CAN nodes are required. Both CAN-nodes should be same bus speed.

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.

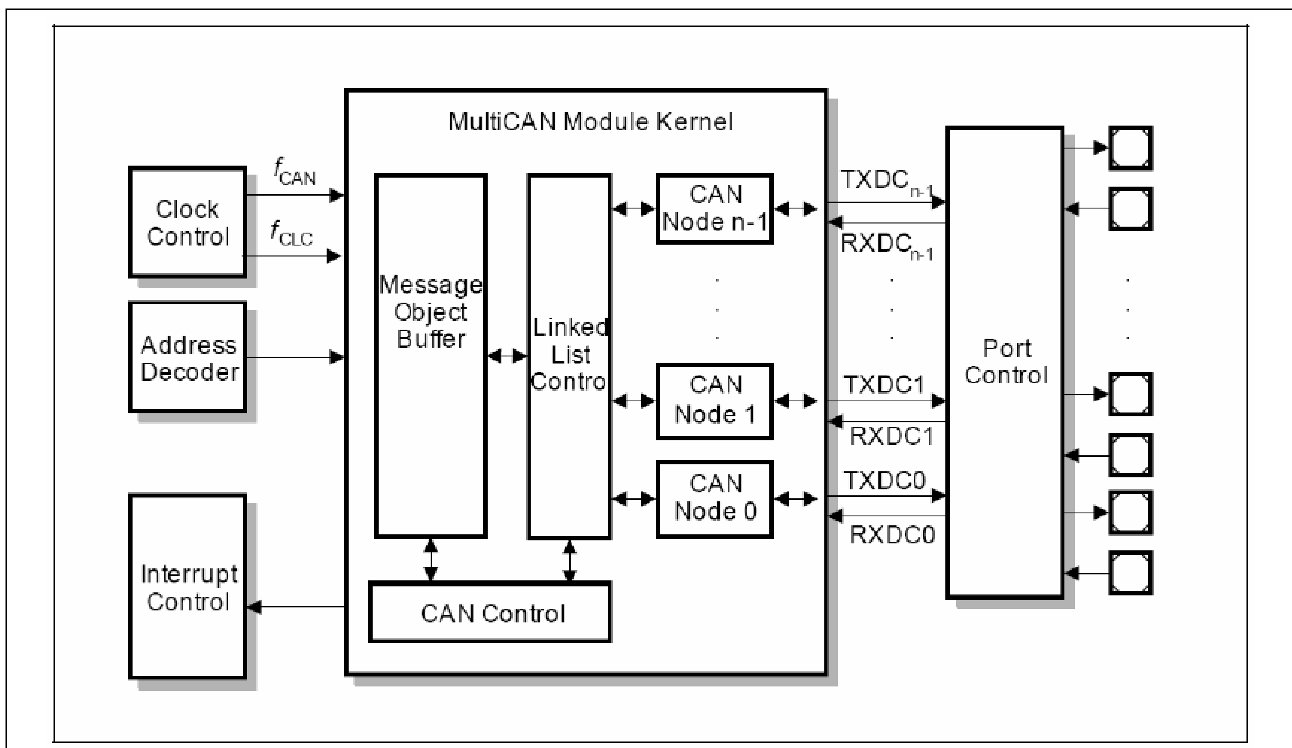

Figure 2 MultiCAN Module

4 FIFO Example setup with XC2287 Starter Kit

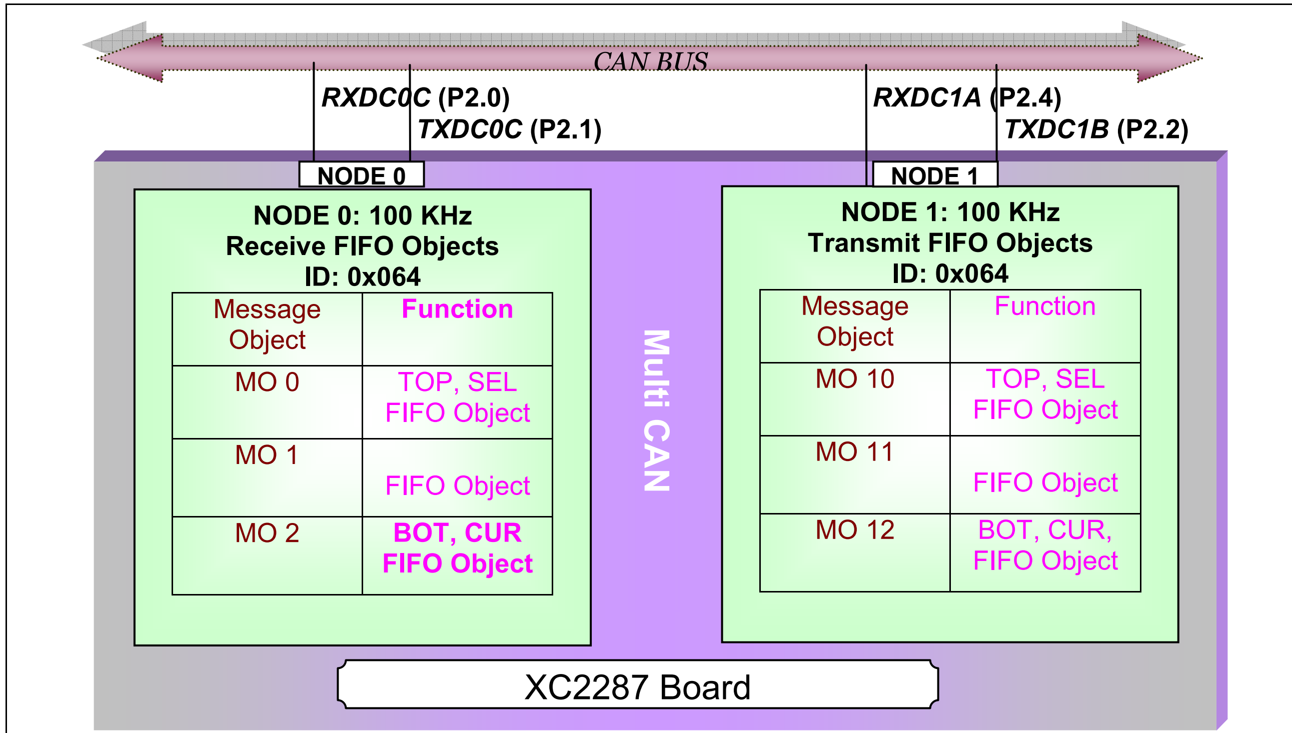

Figure 3 FIFO Example setup with XC2287 Starter Kit

Figure 4 XC2287 Starter Kit

5 FIFO Configuration using DAvE

1. Select MultiCAN Bubble from XC22xx GUI

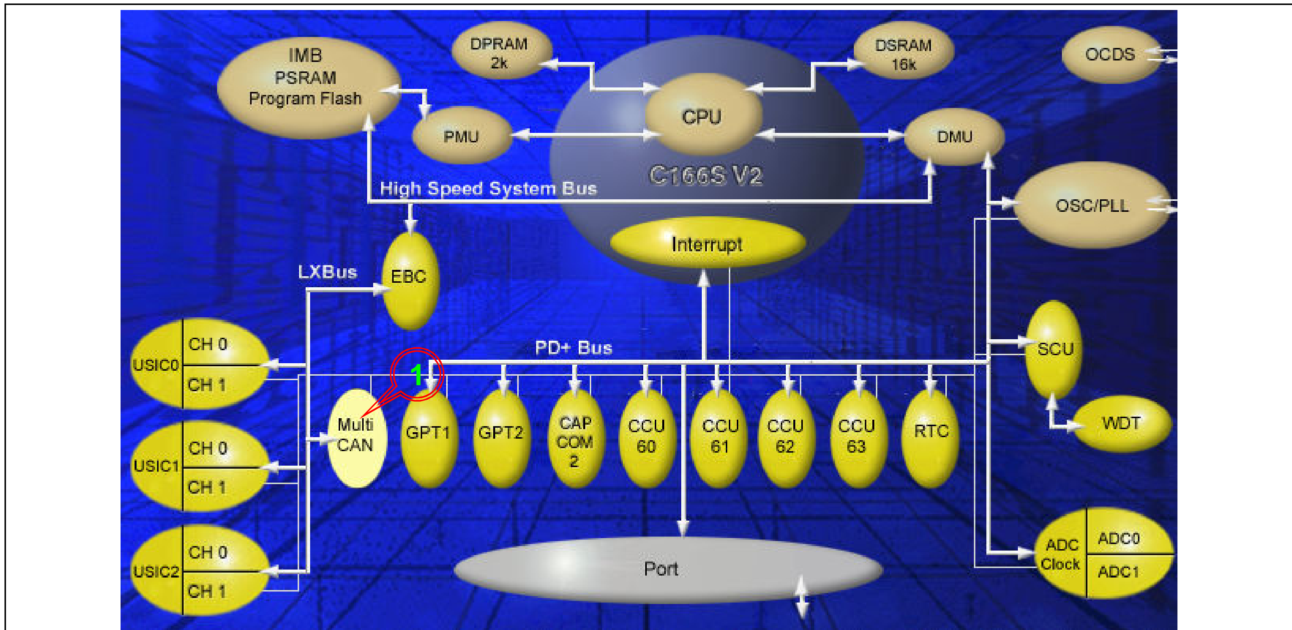

Figure 5 DAvE Main bit map showing MultiCAN module.

2. Configuration of General Page
   1. Select - Enable module; the peripheral is supplied with the clock signal.
   2. Select – Node 0 button follow step 3.
   3. After Node 0 configuration Select Node 1 button (step 5).

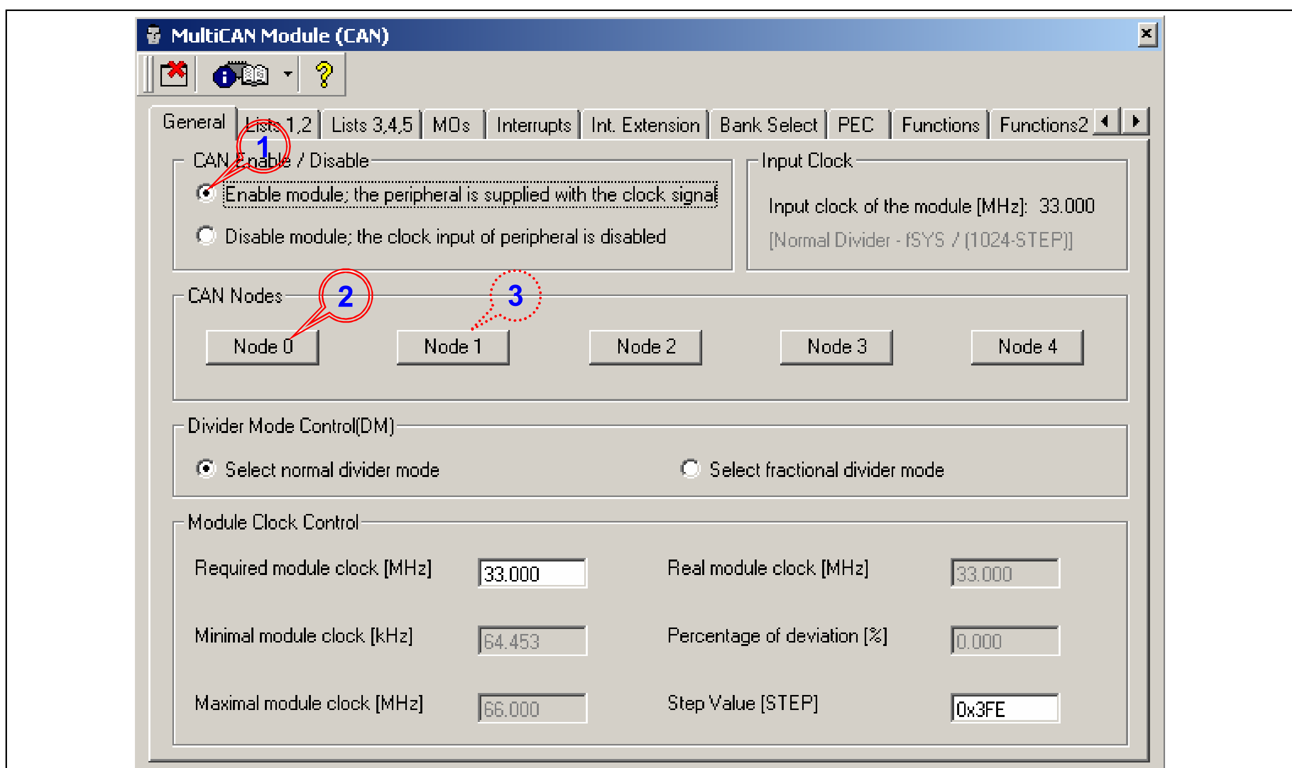

Figure 6 MultiCAN General Page

3. Configuration for Node 0 Control Page
   1. Select – *Use Pin RXDC0C (P2.0) as Input* from Receive Input Selection.
   2. Select – *Use Pin TXDC0C (P2.1) as Output* from Transmit Output Selection.
   3. Enable – Initialize the CAN Node0 (INIT)

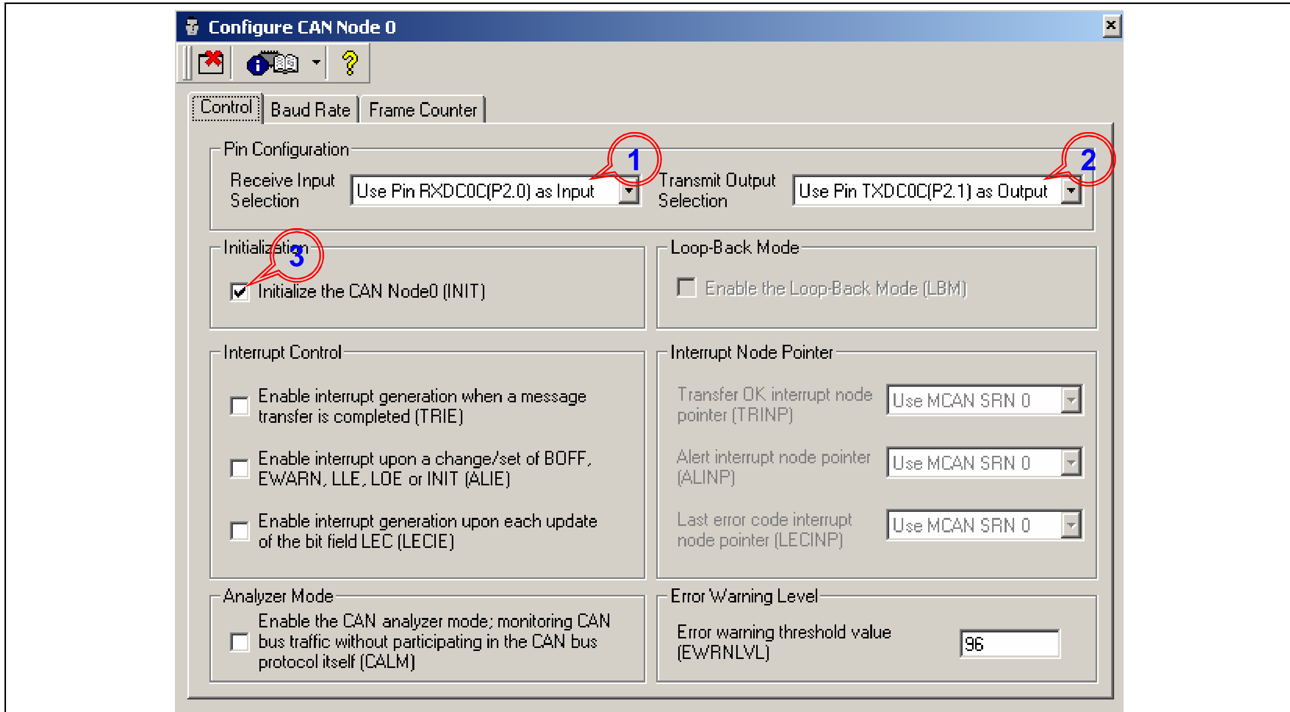

Figure 7 MultiCAN Node 0 Control Page

4. Observe – Required & Real baud rate = 100 kbaud

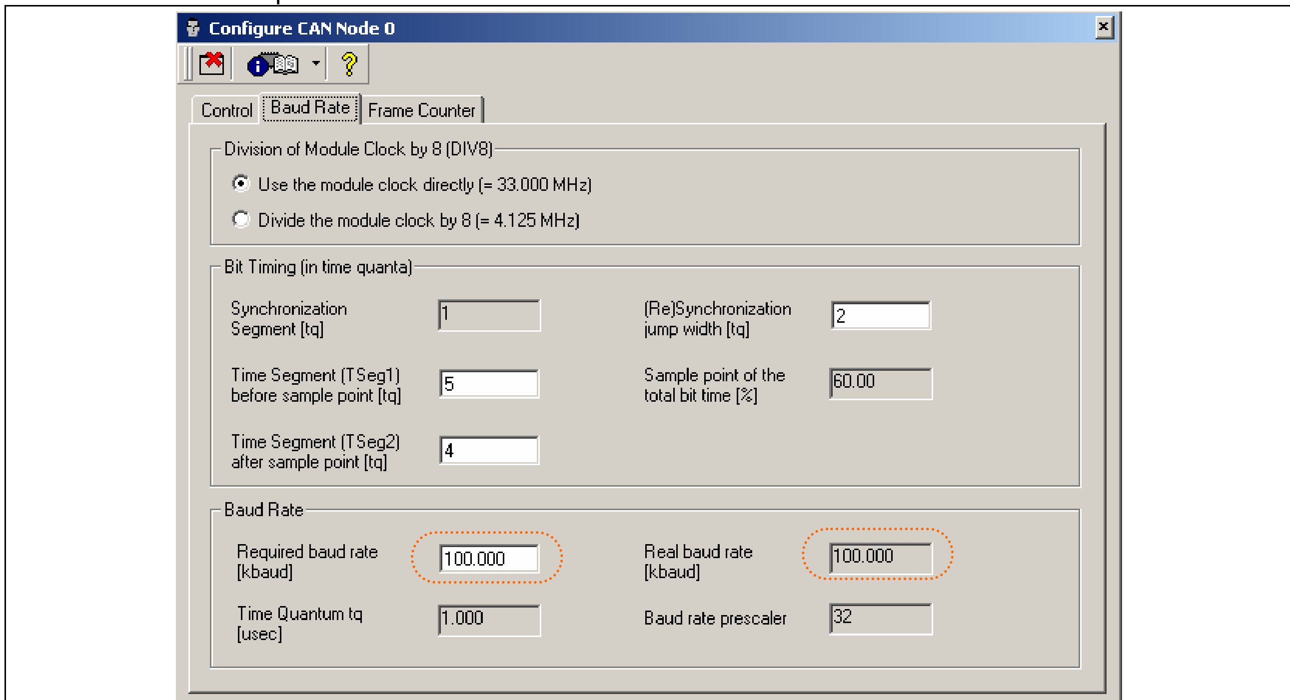

Figure 8 MultiCAN Node 0 Baud Rate Page

5. Configuration for Node 1 Control Page
   1. Select – *Use Pin RXDC1A (P2.4) as Input* from Receive Input Selection.
   2. Select – *Use Pin TXDC1B (P2.2) as Output* from Transmit Output Selection.
   3. Enable – Initialize the CAN Node0 (INIT)

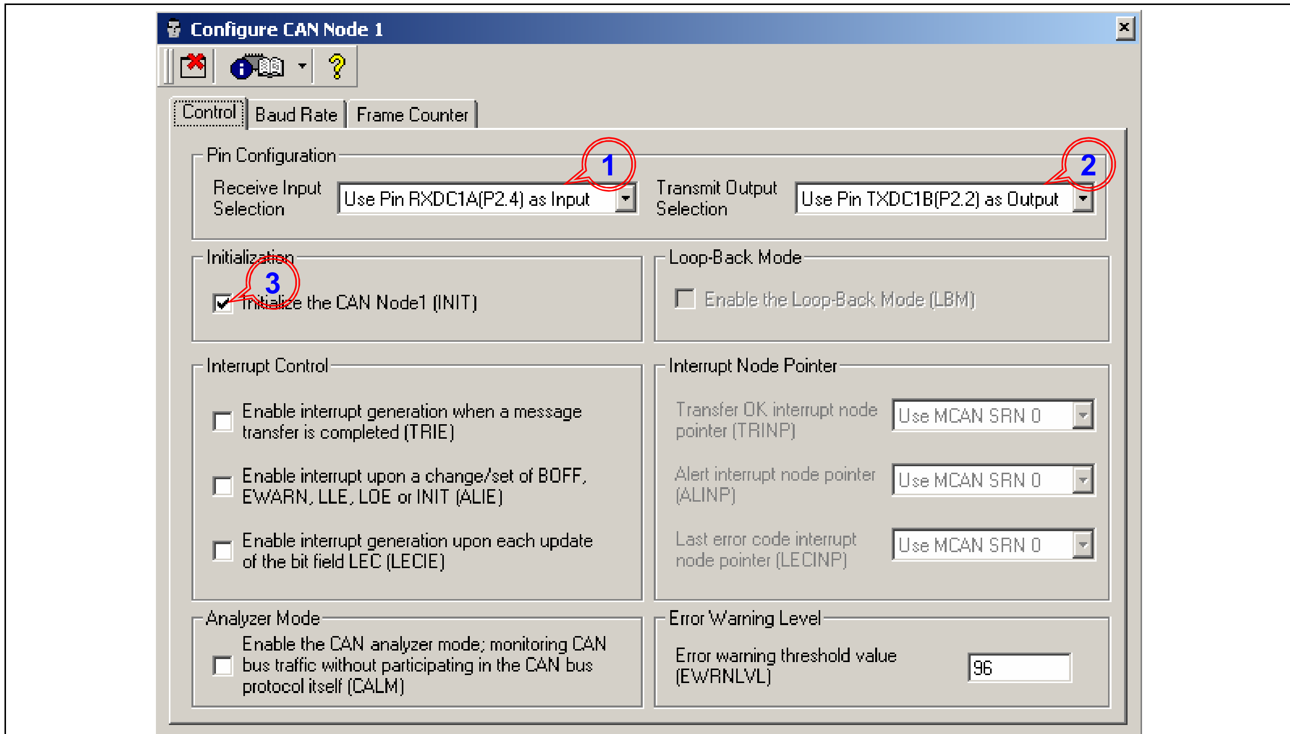

Figure 9 MultiCAN Node 1 Control Page

6. Observe – Required & Real baud rate = 100 kbaud

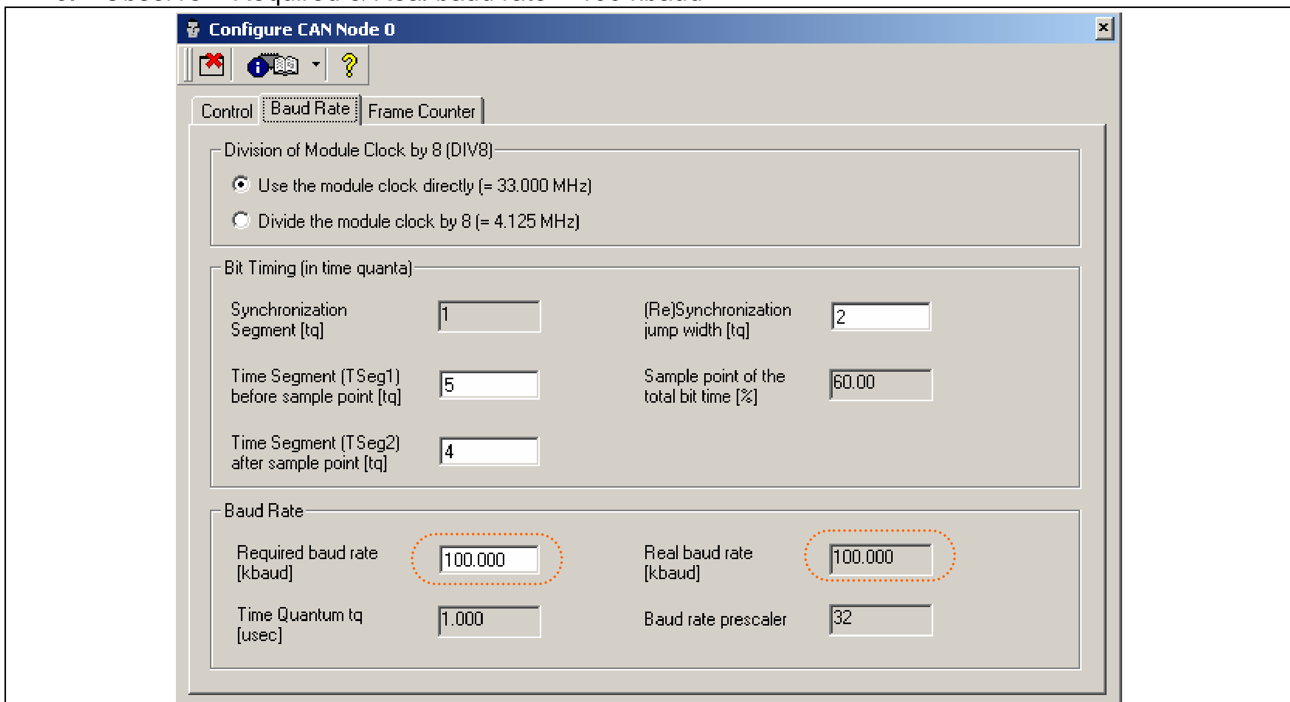

Figure 10 MultiCAN Node 1 Baud Rate Page

7. Configurations of MOs to Lists – Drag & Drop required MOs to List 1 & List 2 as shown

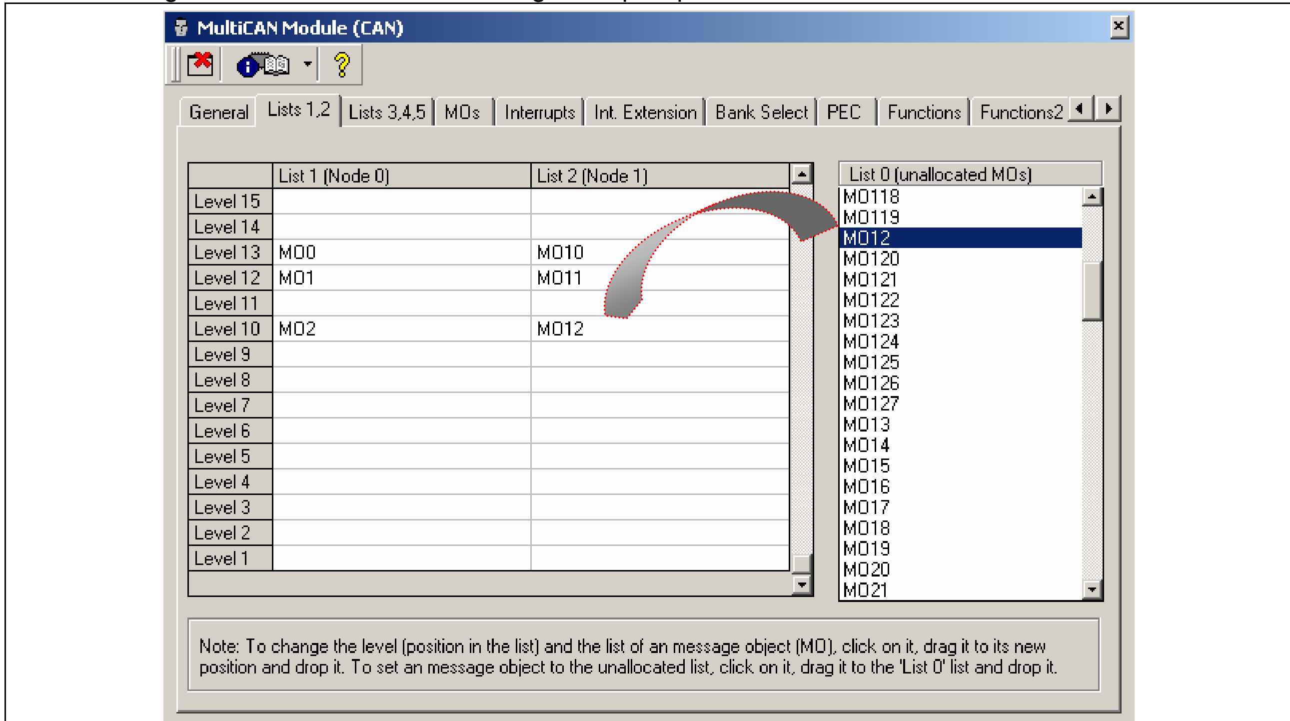

Figure 11 MultiCAN List1, 2 Page

8. After Configuration of MOs to list - MOs Page looks like this. Configure respect MOs as shown in next slides.

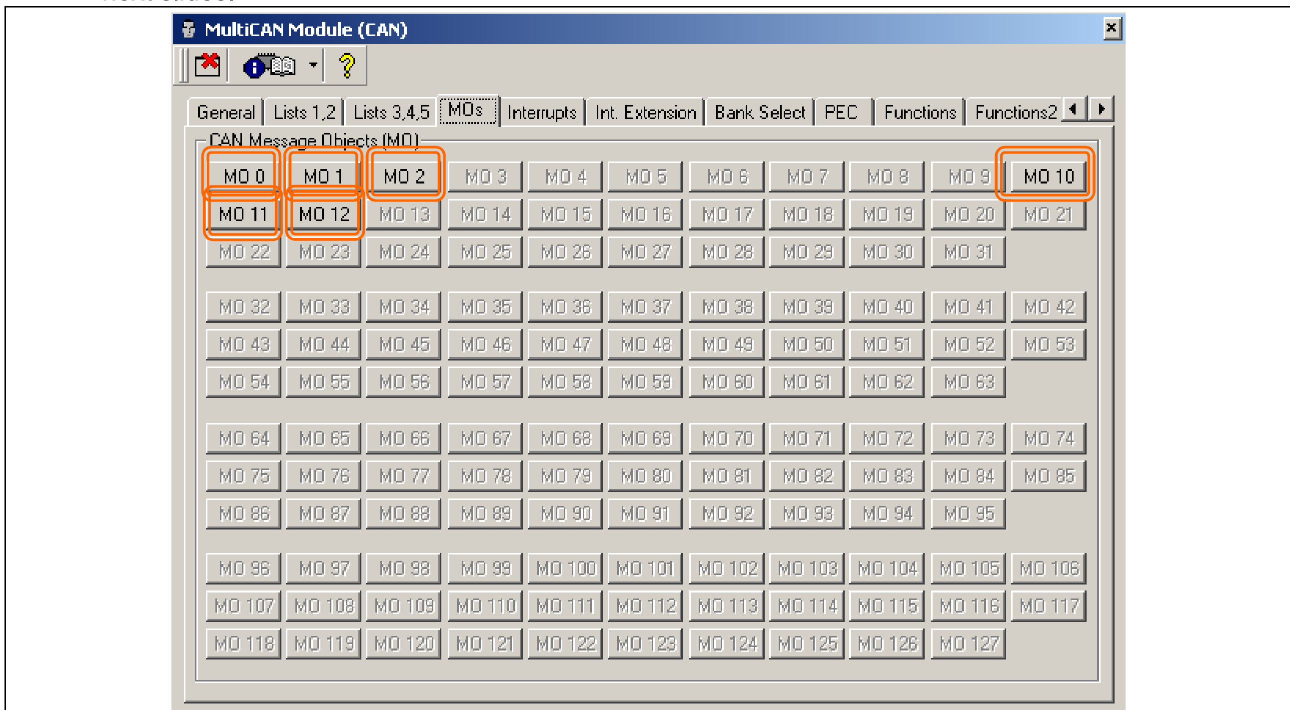

Figure 12 MultiCAN MOs Page

9. Receive FIFO Object (FIFO base) Object Configurations
   1. Enable Message Object
   2. Receive data frames, transmit remote frames
   3. Data Length [8 data bytes]
   4. Standard 11-bit identifier
   5. Identifier (11-bit)

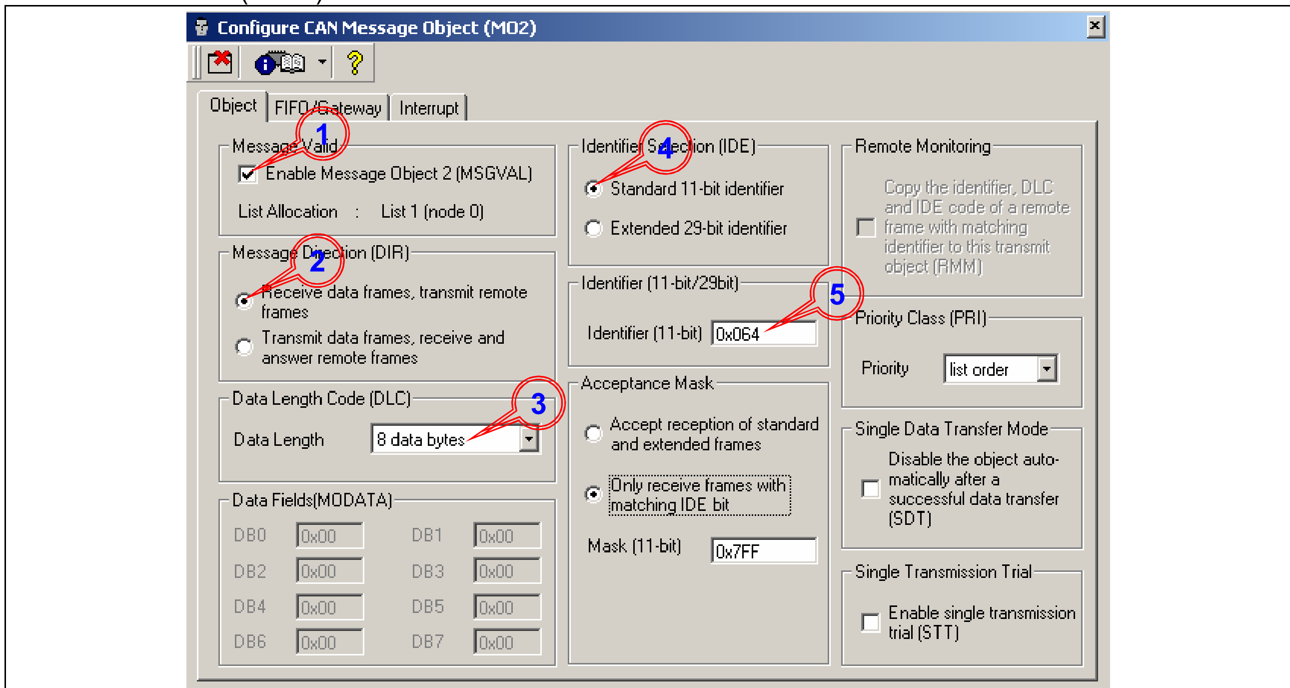

Figure 13 MultiCAN Message Object Page (Receive FIFO base)

10. Configuration - Receive FIFO Object (FIFO base) FIFO:
    1. Select - Receive FIFO base object
    2. 3. 4. Select - BOT, TOP & SEL pointers

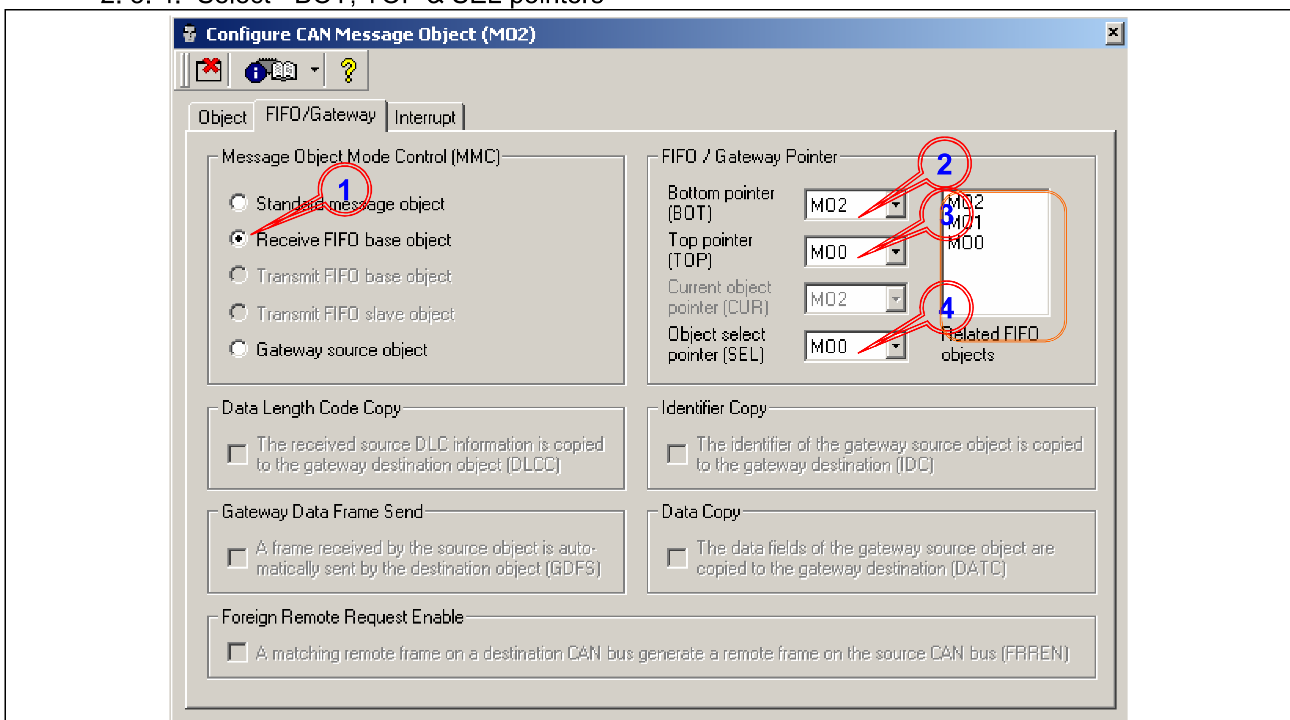

Figure 14 MultiCAN Message Object FIFO/Gateway Page

11. Configure Enable FIFO overflow interrupt (OVIE) and select *Use CAN SRN1* from Overflow Interrupt Node Pointer

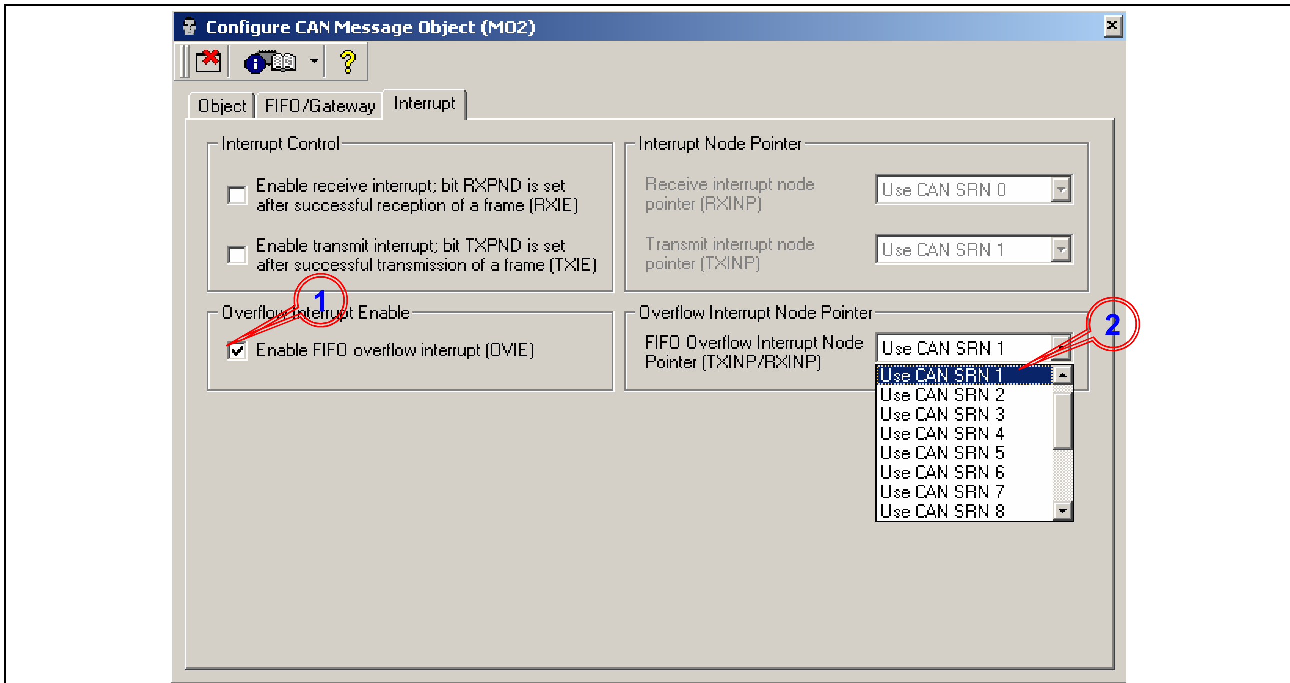

Figure 15 MultiCAN Message Object Interrupt Page

12. Observe – All options are disabled for Receive FIFO Object (FIFO slave) Object & FIFO/Gateway Page

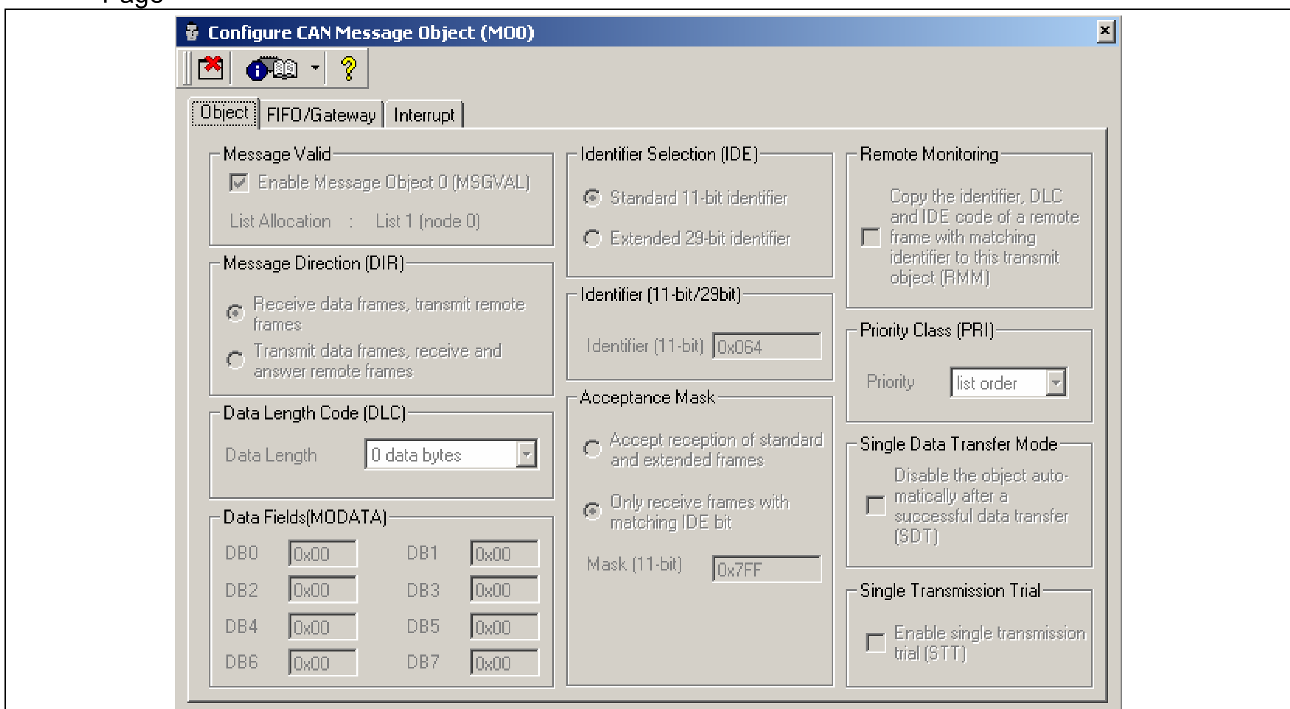

Figure 16 MultiCAN Message Object (Receive Object Slave)

13. Transmit FIFO Object (FIFO base) Object Configuration:
    1. Select – Enable Message Object
    2. Select – Transmit data frames, receive and answer remote frames
    3. Select – Data Length [8 data bytes]
    4. Configure – Data Fields (This can be configured by using Software Message Object Structure)
    5. Select – Standard 11-bit identifier
    6. Configure – Identifier (11-bit)

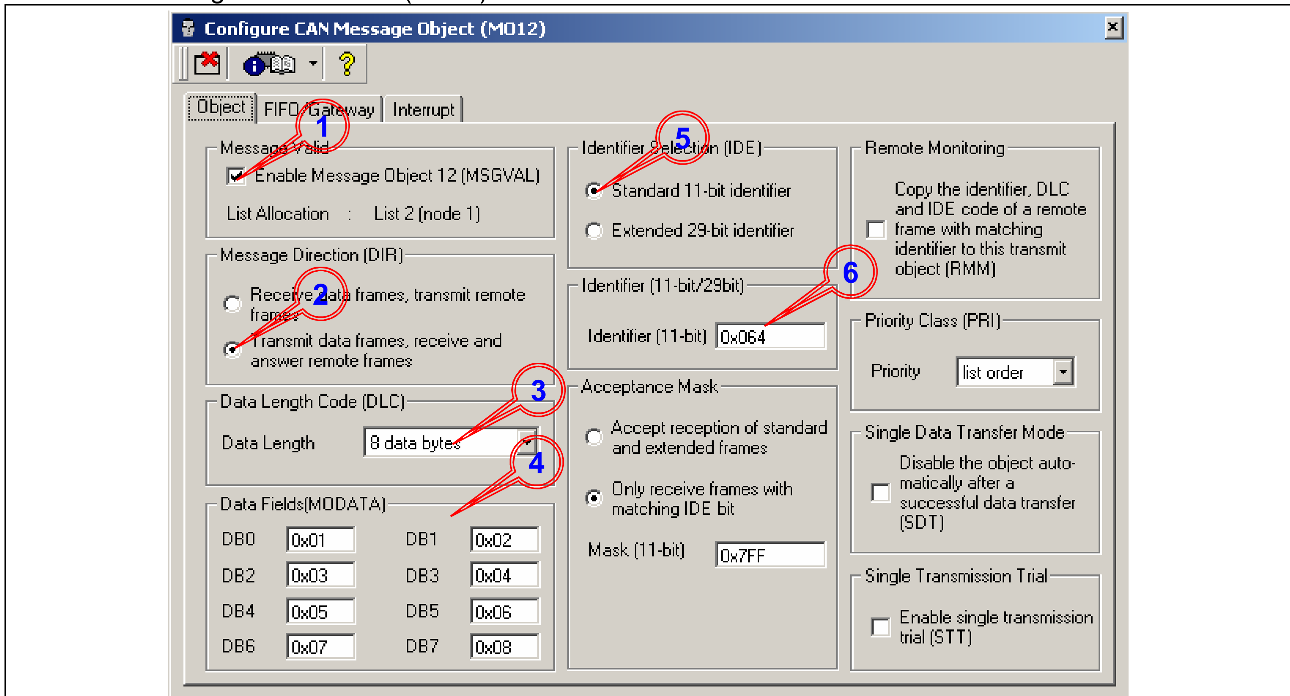

Figure 17 MultiCAN Message Object (Transmit FIFO base)

14. Configuration of Transmit FIFO Object (FIFO base) FIFO:
    1. Select – Transmit FIFO base object
    2. 3. 4. Select – BOT, TOP, SEL pointers

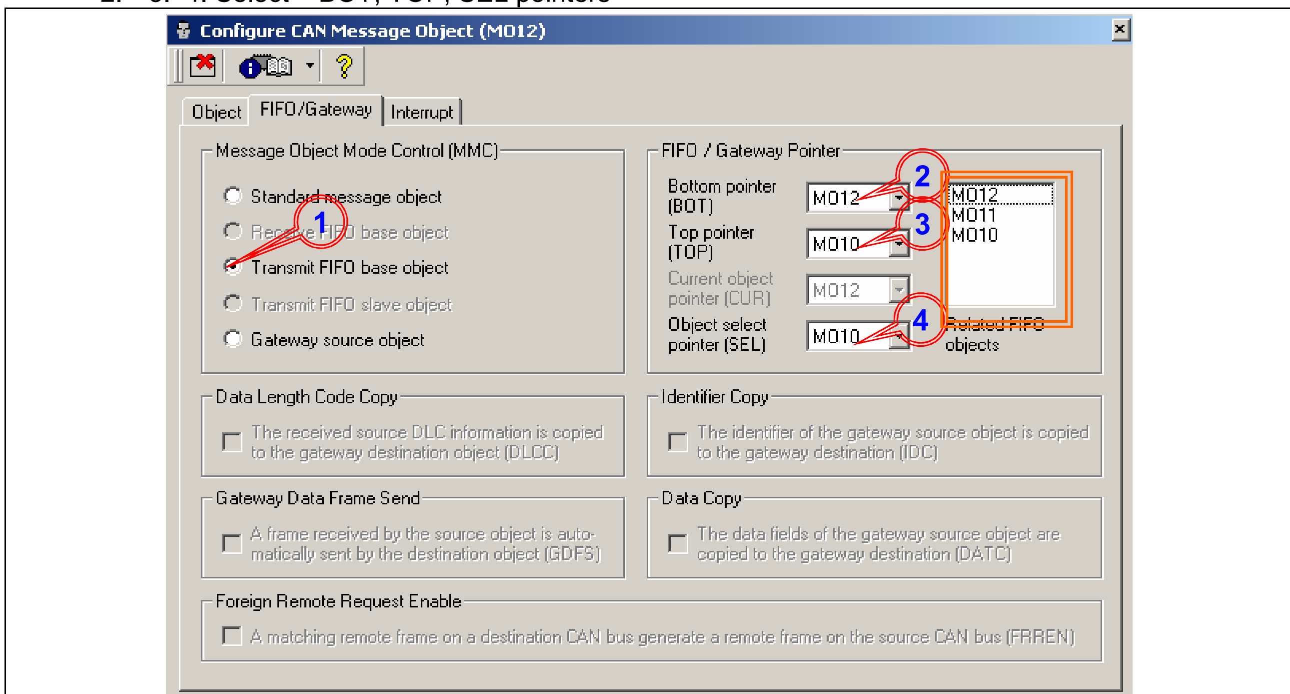

Figure 18 MultiCAN Message Object FIFO/Gateway Page

15. Observe - Transmit FIFO Object (FIFO slave) FIFO Page (configured as FIFO slave object)

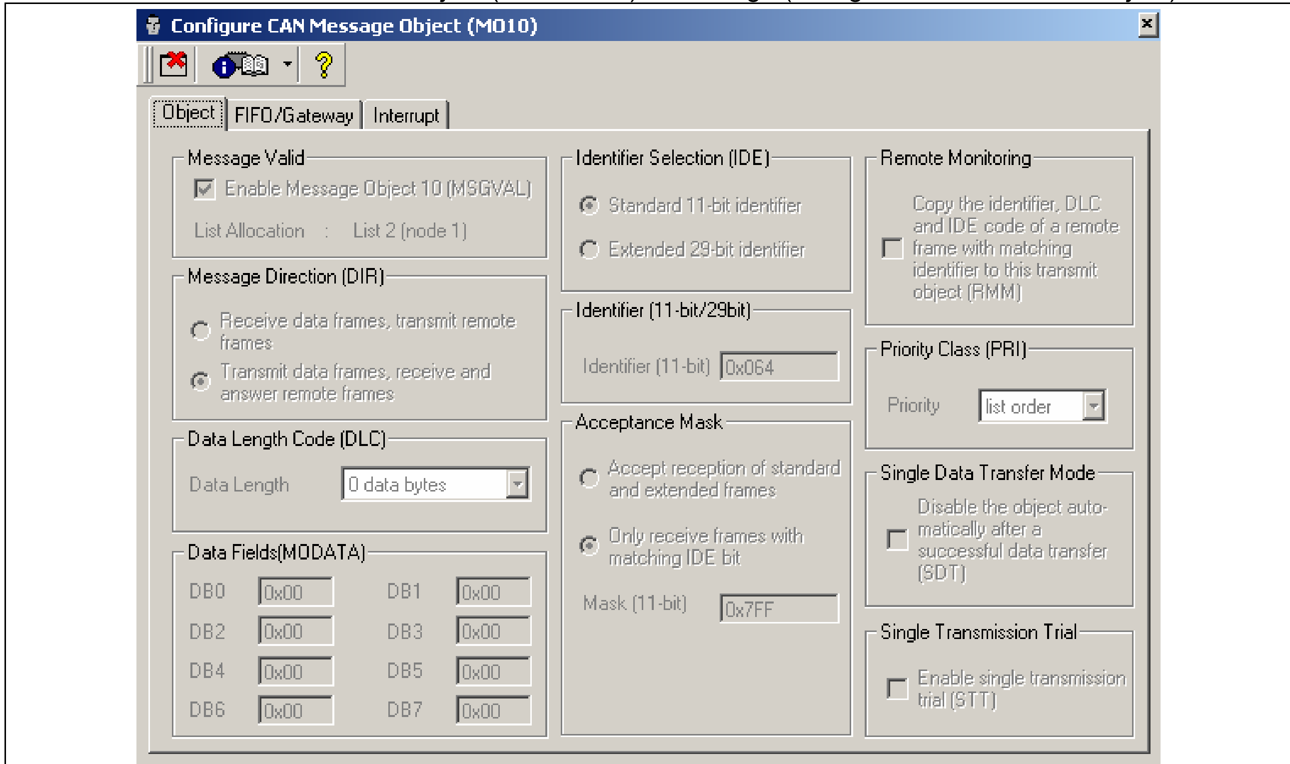

Figure 19 MultiCAN Message Object FIFO/Gateway Page (Transmit Object Slave)

16. Configure CAN INT 1 (Interrupt) by drag and drop the Interrupt to desired group and level.

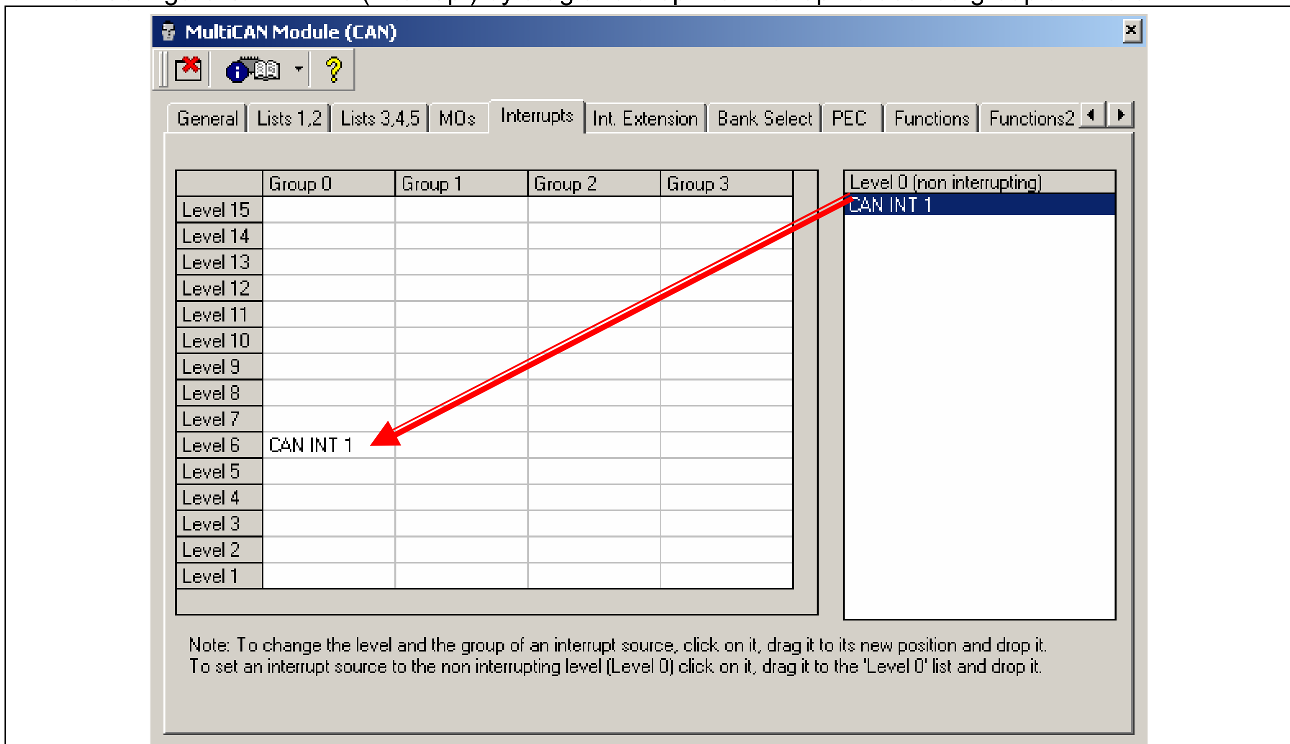

Figure 20 MultiCAN Interrupts Page

17. Select - Functions
    1. CAN_vInit
    2. CAN_ubWriteFIFO
    3. CAN_ubReadFIFO

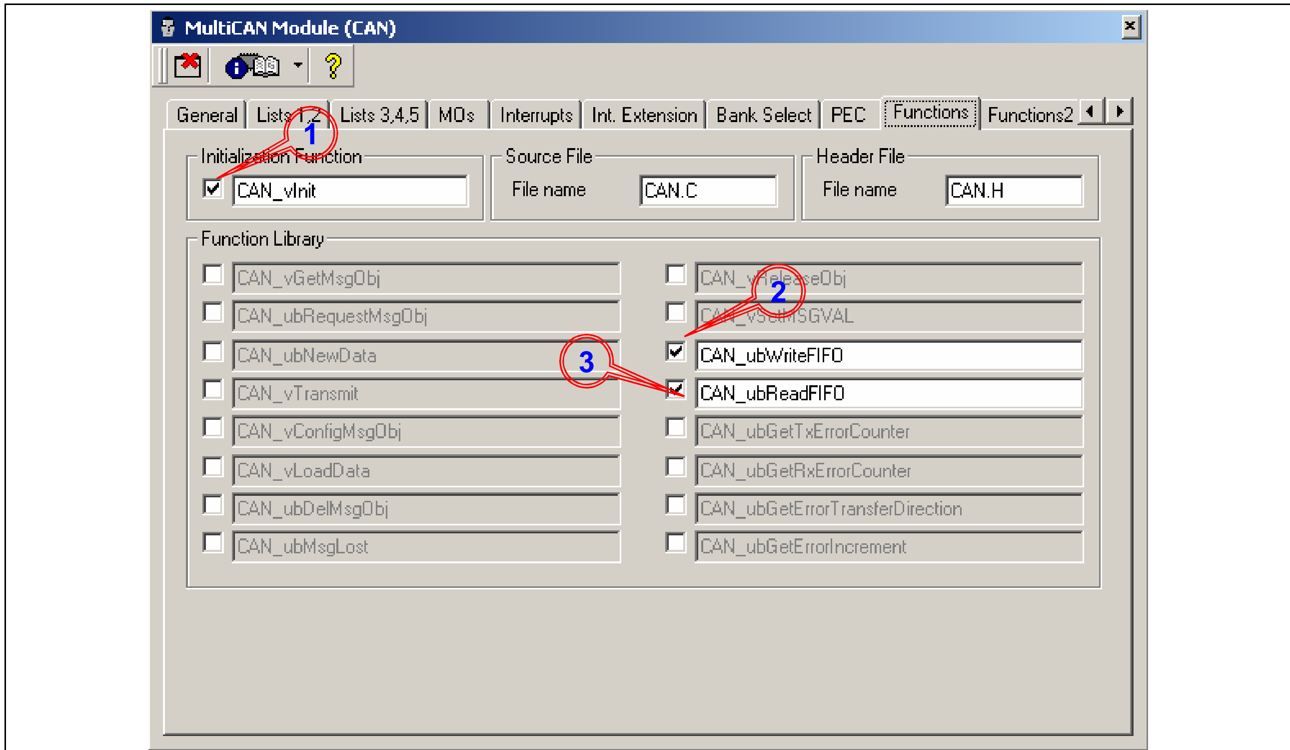

Figure 21 MultiCAN Functions Page

6 Example code for XC2287

The following C-code shows how to setup the MultiCAN for a CAN FIFO Functionality. It's an initialization code written for a XC2287 device for Tasking classic compiler.

Tools used:
DAvE v2.1r22
XC22xx_Series_v1.0.dip
EASYKIT XC2287.100A
TASKING Tools for C166/ST10 v8.6 r3
Universal Debug Engine, Release : 2.00.11 (Universal access device 2)

The MultiCAN module is found on several 8-, 16- and 32-bit devices. The MultiCAN register length is based on 32-bit, for 16- bit devices (XC2000/XE166) the MultiCAN registers are divided into high/low-word. For example `CAN_MOCTRn` is divided into `CAN_MOCTRnH & CAN_MOCTRnL`.

```
//****************************************************************************
///  ----------------------------------------------------------------------------
///  @Definition of a structure for the CAN data
///  ----------------------------------------------------------------------------

// The following data type serves as a software message object. Each access to
// a hardware message object has to be made by forward a pointer to a software
// message object (MCAN_SWObj). The data type has the following fields:
//
// ubMOCfg:
```

```
// this byte contains the 'Data Length Code(DLC)',
// 'Identifier Extension bit(IDE) - if 0 standard frame, else 1 extended frame',
// 'Message Direction(DIR) - if 0 receive Object, else 1 transmit Object'
// 'Acceptance Mask bit(MIDE) - if 0 receives both, else 1 receives matching IDE'
//
//
//          7     6     5     4     3     2     1     0
//       |------------------------------------------------|
//       |     | MIDE |IDE | DIR |          DLC           |
//       |------------------------------------------------|
//
// ulID:
// this field is four bytes long and contains either the 11-bit identifier
// or the 29-bit identifier
//
// ulMask:
// this field is four bytes long and contains either the 11-bit mask
// or the 29-bit mask
//
// ubData[8]:
// 8 bytes containing the data of a frame
//
// uwCounter:
// this field is two bytes long and contains the counter value

typedef struct
{
    ubyte ubMOCfg;    // message object configuration
    ulong ulID;       // standard (11-bit)/extended (29-bit) identifier
    ulong ulMask;     // standard (11-bit)/extended (29-bit) mask
    ubyte ubData[8];  // 8-bit data bytes
    uword uwCounter;  // frame counter(MOIPRnH[15-0])
}stCAN_SWObj;
```

```
//****************************************************************************
// @Module        MultiCAN Controller
// @Filename      CAN.C
// @Project       MCAN_FIFO
//----------------------------------------------------------------------------
// @Controller    Infineon XC2287
// @Description   This file contains functions that use the CAN module.
//
//     NODE 0 : 100kBaud, use CAN1 output connector
//              Receive FIFO Objects: MO 0, MO 1, MO 2
//
//     NODE 1 : 100kBaud, use CAN2 output connector
//              Transmit FIFO Objects: MO 10, MO 11, MO 12
//
// Version: 01 2008-01-08
//****************************************************************************

//****************************************************************************
// @Project Includes
//****************************************************************************

#include "MAIN.H"

//****************************************************************************
// @Global Variables
//****************************************************************************

static ubyte ubFIFOWritePtr[13];
static ubyte ubFIFOReadPtr[13];

//****************************************************************************
// @Function      void CAN_vInit(void)
```

```
//
//----------------------------------------------------------------------------
// @Description   This is the initialization function of the CAN function
//                library. It is assumed that the SFRs used by this library
//                are in reset state.
//----------------------------------------------------------------------------
// @Returnvalue   None
//----------------------------------------------------------------------------
// @Parameters    None
//****************************************************************************

void CAN_vInit(void)
{
ubyte i;

    MCAN_KSCCFG = 0x0003;      // load Kernel State Configuration Register
    _nop();  // one cycle delay
    _nop();  // one cycle delay

  ///  -----------------------------------------------------------------------
  ///  Configuration of the Module Clock:
  ///  -----------------------------------------------------------------------
  ///  - the CAN module clock = 33.00 MHz
    CAN_FDRL = 0x43FE;      // load Fractional Divider Register

  ///  - wait until Panel has finished the initialisation
    while(CAN_PANCTRL & CAN_PANCTR_BUSY){ // wait until Panel has finished
                                          // the initialisation
      }
  ///  -----------------------------------------------------------------------
  ///  Configuration of CAN Node 0:
  ///  -----------------------------------------------------------------------
    CAN_NCR0 = 0x0041;      // load NODE 0 control register[15-0]

  ///  - P2.0 is used for CAN0 Receive input(RXDC0C)
  ///  - P2.1 is used for CAN0 Transmit output(TXDC0C)
  P2_IOCR01 = 0x0090; //set direction register
  CAN_NPCR0 = 0x0002; // load node0 port control register - Loop-back mode is disabled

  ///  Configuration of the Node 0 Baud Rate:
  ///  - required baud rate = 100.000 kbaud

    CAN_NBTR0L = 0x3460;      // load NBTR0_DIV8, TSEG2, TSEG1, SJW and BRP

  ///  -----------------------------------------------------------------------
  ///  Configuration of CAN Node 1:
  ///  -----------------------------------------------------------------------
    CAN_NCR1 = 0x0041;      // load NODE 1 control register[15-0]

  ///  Configuration of the used CAN Port Pins:
  ///  - P2.4 is used for CAN1 Receive input(RXDC1A)
  ///  - P2.2 is used for CAN1 Transmit output (TXDC1B)
  P2_IOCR02 = 0x0090;     // set direction register
  CAN_NPCR1 = 0x0000;     // load node1 port control register - Loop-back mode is disabled

  ///  Configuration of the Node 1 Baud Rate:
  ///  - required baud rate = 100.000 kbaud
    CAN_NBTR1L = 0x3460;      // load NBTR1_DIV8, TSEG2, TSEG1, SJW and BRP

  ///  -----------------------------------------------------------------------
  ///  Configuration of the CAN Message Object List Structure:
  ///  -----------------------------------------------------------------------

  ///  Allocate MOs for list 1:

    SetListCommand(0x0102,0x0002);  // MO2 for list 1
```

```
  SetListCommand(0x0101,0x0002);  // MO1 for list 1
  SetListCommand(0x0100,0x0002);  // MO0 for list 1

///  Allocate MOs for list 2:

  SetListCommand(0x020C,0x0002);  // MO12 for list 2
  SetListCommand(0x020B,0x0002);  // MO11 for list 2
  SetListCommand(0x020A,0x0002);  // MO10 for list 2

///  -------------------------------------------------------------------
///  Configuration of Message Object 0:
///  -------------------------------------------------------------------
///  - message object 0 is valid
///  - message object is used as receive object
///  - this message object is assigned to list 1 (node 0)
  CAN_MOCTR0H = 0x00A0;        // load MO0 control register high
  CAN_MOCTR0L = 0x0000;        // load MO0 control register low

///  - identifier 11-bit:      0x064
  CAN_MOAR0H = 0x4190;         // load MO0 arbitration register high
  CAN_MOAR0L = 0x0000;         // load MO0 arbitration register low

///  Configuration of Message Object 0 FIFO/Gateway pointer:
  CAN_MOFGPR0H = 0x0000;       // load MO0 FIFO/gateway pointer register high
  CAN_MOFGPR0L = 0x0000;       // load MO0 FIFO/gateway pointer register low

///  Configuration of Message Object 0 Function control:
///  - this object is a RECEIVE FIFO SLAVE OBJECT connected to the base object 2
///  - 0 valid data bytes
  CAN_MOFCR0H = 0x0000;        // load MO0 function control register high
  CAN_MOFCR0L = 0x0000;        // load MO0 function control register low

///  -------------------------------------------------------------------
///  Configuration of Message Object 1:
///  -------------------------------------------------------------------
///  - message object 1 is valid
///  - message object is used as receive object
///  - this message object is assigned to list 1 (node 0)
  CAN_MOCTR1H = 0x00A0;        // load MO1 control register high
  CAN_MOCTR1L = 0x0000;        // load MO1 control register low

///  - identifier 11-bit:      0x064
  CAN_MOAR1H = 0x4190;         // load MO1 arbitration register high
  CAN_MOAR1L = 0x0000;         // load MO1 arbitration register low

///  Configuration of Message Object 1 FIFO/Gateway pointer:
  CAN_MOFGPR1H = 0x0000;       // load MO1 FIFO/gateway pointer register high
  CAN_MOFGPR1L = 0x0000;       // load MO1 FIFO/gateway pointer register low

///  Configuration of Message Object 1 Function control:
///  - this object is a RECEIVE FIFO SLAVE OBJECT connected to the base object 2
///  - 0 valid data bytes
  CAN_MOFCR1H = 0x0000;        // load MO1 function control register high
  CAN_MOFCR1L = 0x0000;        // load MO1 function control register low

///  -------------------------------------------------------------------
///  Configuration of Message Object 2:
///  -------------------------------------------------------------------
///  - message object 2 is valid
///  - message object is used as receive object
///  - this message object is assigned to list 1 (node 0)
  CAN_MOCTR2H = 0x00A0;        // load MO2 control register high
  CAN_MOCTR2L = 0x0000;        // load MO2 control register low

///  - identifier 11-bit:      0x064
  CAN_MOAR2H = 0x4190;         // load MO2 arbitration register high
```

```
  CAN_MOAR2L = 0x0000;          // load MO2 arbitration register low

///  Configuration of Message Object 2 interrupt pointer:
///  - use message pending register 0 bit position 2
  CAN_MOIPR2H = 0x0000;         // load MO2 interrupt pointer register high
  CAN_MOIPR2L = 0x0210;         // load MO2 interrupt pointer register low

///  Configuration of Message Object 2 FIFO/Gateway pointer:
///  - current select pointer : MO2
///  - object select pointer : MO0
  CAN_MOFGPR2H = 0x0002;        // load MO2 FIFO/gateway pointer register high

///  - bottom pointer : MO2
///  - top pointer : MO0
  CAN_MOFGPR2L = 0x0002;        // load MO2 FIFO/gateway pointer register low

///  Configuration of Message Object 2 Function control:
///  - this object is a RECEIVE FIFO BASE OBJECT
///  - 8 valid data bytes
///  - enable FIFO overflow interrupt
  CAN_MOFCR2H = 0x0804;         // load MO2 function control register high
  CAN_MOFCR2L = 0x0001;         // load MO2 function control register low

///  -----------------------------------------------------------------------
///  Configuration of Message Object 10:
///  -----------------------------------------------------------------------
///  - message object 10 is valid
///  - message object is used as transmit object
///  - this message object is assigned to list 2 (node 1)
  CAN_MOCTR10H = 0x0AA8;        // load MO10 control register high
  CAN_MOCTR10L = 0x0000;        // load MO10 control register low

///  - identifier 11-bit:       0x064
  CAN_MOAR10H = 0x4190;         // load MO10 arbitration register high
  CAN_MOAR10L = 0x0000;         // load MO10 arbitration register low

///  Configuration of Message Object 10 FIFO/Gateway pointer:
///  - current select pointer : MO12
  CAN_MOFGPR10H = 0x000C;       // load MO10 FIFO/gateway pointer register high
  CAN_MOFGPR10L = 0x0000;       // load MO10 FIFO/gateway pointer register low

///  Configuration of Message Object 10 Function control:
///  - this object is a TRANSMIT FIFO SLAVE OBJECT connected to the base object 12
///  - 0 valid data bytes
  CAN_MOFCR10H = 0x0000;        // load MO10 function control register high
  CAN_MOFCR10L = 0x0003;        // load MO10 function control register low

///  -----------------------------------------------------------------------
///  Configuration of Message Object 11:
///  -----------------------------------------------------------------------
///  - message object 11 is valid
///  - message object is used as transmit object
///  - this message object is assigned to list 2 (node 1)
  CAN_MOCTR11H = 0x0AA8;        // load MO11 control register high
  CAN_MOCTR11L = 0x0000;        // load MO11 control register low

///  - identifier 11-bit:       0x064
  CAN_MOAR11H = 0x4190;         // load MO11 arbitration register high
  CAN_MOAR11L = 0x0000;         // load MO11 arbitration register low

/// Configuration of Message Object 11 FIFO/Gateway pointer:
///  - current select pointer : MO12
  CAN_MOFGPR11H = 0x000C;       // load MO11 FIFO/gateway pointer register high
  CAN_MOFGPR11L = 0x0000;       // load MO11 FIFO/gateway pointer register low

///  Configuration of Message Object 11 Function control:
```

```
  ///  - this object is a TRANSMIT FIFO SLAVE OBJECT connected to the base object 12
  ///  - 0 valid data bytes
    CAN_MOFCR11H = 0x0000;        // load MO11 function control register high
    CAN_MOFCR11L = 0x0003;        // load MO11 function control register low

  ///  -----------------------------------------------------------------------
  ///  Configuration of Message Object 12:
  ///  -----------------------------------------------------------------------
  ///  - message object 12 is valid
  ///  - message object is used as transmit object
  ///  - this message object is assigned to list 2 (node 1)
    CAN_MOCTR12H = 0x0EA8;        // load MO12 control register high
    CAN_MOCTR12L = 0x0000;        // load MO12 control register low

  ///  - identifier 11-bit:       0x064
    CAN_MOAR12H = 0x4190;         // load MO12 arbitration register high
    CAN_MOAR12L = 0x0000;         // load MO12 arbitration register low

  ///  Configuration of Message Object 12 Data:
    CAN_MODATA12HH = 0x0807;      // load MO12 Data Bytes(DB7 & DB6)
    CAN_MODATA12HL = 0x0605;      // load MO12 Data Bytes(DB5 & DB4)
    CAN_MODATA12LH = 0x0403;      // load MO12 Data Bytes(DB3 & DB2)
    CAN_MODATA12LL = 0x0201;      // load MO12 Data Bytes(DB1 & DB0)

  ///  Configuration of Message Object 12 FIFO/Gateway pointer:
  ///  - current select pointer : MO12
  ///  - object select pointer : MO10
    CAN_MOFGPR12H = 0x0A0C;       // load MO12 FIFO/gateway pointer register high

  ///  - bottom pointer : MO12
  ///  - top pointer : MO10
    CAN_MOFGPR12L = 0x0A0C;       // load MO12 FIFO/gateway pointer register low

  ///  Configuration of Message Object 12 Function control:
  ///  - this object is a TRANSMIT FIFO BASE OBJECT
  ///  - 8 valid data bytes
    CAN_MOFCR12H = 0x0800;        // load MO12 function control register high
    CAN_MOFCR12L = 0x0002;        // load MO12 function control register low

  ///  -----------------------------------------------------------------------
  ///  Configuration of Service Request Nodes 0 - 15:
  ///  -----------------------------------------------------------------------
  ///  SRN1 service request node configuration:
  ///  - SRN1 interrupt priority level (ILVL) = 6
  ///  - SRN1 interrupt group level (GLVL) = 0
  ///  - SRN1 group priority extension (GPX) = 0
  CAN_1IC        = 0x0058;

  ///  -----------------------------------------------------------------------
  ///  Initialization of the FIFO Pointer:
  ///  -----------------------------------------------------------------------
  for (i = 0; i < 13; i++)
  {
    ubFIFOWritePtr[i]  = (ubyte)(CAN_HWOBJ[i].uwMOFGPRL & 0x00FF);
    ubFIFOReadPtr[i]   = (ubyte)(CAN_HWOBJ[i].uwMOFGPRL & 0x00FF);
  }
  //   -----------------------------------------------------------------------
  //   Start the CAN Nodes:
  ///  - ------------- CAN_NCR0 ----------------------------------------------
    CAN_NCR0 &= ~(uword)0x0041; // reset INIT and CCE

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

} //  End of function CAN_vInit
```

```
//****************************************************************************
// @Function      ubyte CAN_ubWriteFIFO(ubyte ubObjNr, stCAN_SWObj *pstObj)
//----------------------------------------------------------------------------
// @Description   This function sets up the next free TRANSMIT message object
//                which is part of a FIFO. This includes the 8 data bytes,
//                the identifier (11- or 29-bit) and the data number (0-8
//                bytes). The direction bit (DIR) and the EDE-bit can not be
//                changed. The acceptance mask register and the Frame Counter
//                remains unchanged. This function checks whether the choosen
//                message object is still executing a transmit request, or if
//                the object can be accessed exclusively.
//                The structure of the SW message object is defined in the
//                header file CAN.H (see CAN_SWObj).
//                Note:
//                This function can only used for TRANSMIT objects which are
//                configured for FIFO base functionality.
//----------------------------------------------------------------------------
// @Returnvalue   0: message object is busy (a transfer is active); 1: the
//                message object was configured and the transmite is
//                requested; 2: this is not a FIFO base object
//----------------------------------------------------------------------------
// @Parameters    ubObjNr:
//                Number of the FIFO base object
// @Parameters    *pstObj:
//                Pointer on a message object
//****************************************************************************

ubyte CAN_ubWriteFIFO(ubyte ubObjNr, stCAN_SWObj *pstObj)
{
  ubyte i, j;
  ubyte ubReturn;
  ubReturn = 2;

  if((CAN_HWOBJ[ubObjNr].uwMOFCRL & 0x000F) == 0x0002)    // if transmit FIFO base object
  {
    j = ubFIFOWritePtr[ubObjNr];
    ubReturn = 0;
    if((CAN_HWOBJ[j].uwMOCTRL & 0x0100) == 0x0000)        // if reset TXRQ
    {
      if(j == (CAN_HWOBJ[j].uwMOCTRH & 0xFF00) >> 8)      // last MO in a list
      {
        // WritePtr = BOT of the base object
        ubFIFOWritePtr[ubObjNr] = (ubyte)(CAN_HWOBJ[ubObjNr].uwMOFGPRL & 0x00FF);
      }
      else
      {
        // WritePtr = PNEXT of the current selected slave
        ubFIFOWritePtr[ubObjNr] = (ubyte)((CAN_HWOBJ[j].uwMOCTRH & 0xFF00) >> 8);
      }
      CAN_HWOBJ[j].uwMOCTRL = 0x0008;                     // reset NEWDAT
      CAN_HWOBJ[j].uwMOARH  &= ~(uword)0x1FFF;

      if(CAN_HWOBJ[j].uwMOARH & 0x2000)                   // extended identifier
      {
        CAN_HWOBJ[j].uwMOARH |= (uword)((pstObj->ulID >> 16) & 0x1FFF);
        CAN_HWOBJ[j].uwMOARL = (uword)(pstObj->ulID & 0xFFFF);
      }
      else                                                // if standard identifier
      {
        CAN_HWOBJ[j].uwMOARH |= (uword)((pstObj->ulID & 0x07FF) << 2);
        CAN_HWOBJ[j].uwMOARL = 0x0000;
      }
      CAN_HWOBJ[j].uwMOFCRH &= ~(uword)0x0F00;
      CAN_HWOBJ[j].uwMOFCRH |= (((uword)(pstObj->ubMOCfg & 0x0F)) << 8);

      for(i = 0; i < (pstObj->ubMOCfg & 0x0F); i++)
```

```
      {
        CAN_HWOBJ[j].ubData[i] = pstObj->ubData[i];
      }
      CAN_HWOBJ[j].uwMOCTRH  = 0x0128;              // set TXRQ, NEWDAT, MSGVAL
      ubReturn = 1;
    }
  }
  return(ubReturn);
} //  End of function CAN_ubWriteFIFO

//****************************************************************************
// @Function      ubyte CAN_ubReadFIFO(ubyte ubObjNr, stCAN_SWObj *pstObj)
//----------------------------------------------------------------------------
// @Description   This function reads the next RECEIVE message object which
//                is part of a FIFO. It checks whether the selected RECEIVE
//                OBJECT has received a new message. If so the forwarded SW
//                message object is filled with the content of the HW message
//                object and the functions returns the value "1". The
//                structure of the SW message object is defined in the header
//                file CAN.H (see CAN_SWObj).
//                Note:
//                This function can only used for RECEIVE objects which are
//                configured for FIFO base functionality.
//                Be sure that no interrupt is enabled for the FIFO objects.
//----------------------------------------------------------------------------
// @Returnvalue   0: the message object has not received a new message; 1:
//                the message object has received a new message; 2: this is
//                not a FIFO base object; 3: a previous message was lost; 4:
//                the received message is corrupted
//----------------------------------------------------------------------------
// @Parameters    ubObjNr:
//                Number of the FIFO base object
// @Parameters    *pstObj:
//                Pointer on a message object to be filled by this function
//****************************************************************************

ubyte CAN_ubReadFIFO(ubyte ubObjNr, stCAN_SWObj *pstObj)
{
  ubyte i,j;
  ubyte ubReturn;
  ubReturn = 2;

  if((CAN_HWOBJ[ubObjNr].uwMOFCRL & 0x000F) == 0x0001)    // if receive FIFO base object
  {
    j = ubFIFOReadPtr[ubObjNr];
    ubReturn = 0;
    if(CAN_HWOBJ[j].uwMOCTRL & 0x0008)                    // if NEWDAT
    {
      CAN_HWOBJ[j].uwMOCTRL = 0x0008;                     // clear NEWDAT

      if(j == (CAN_HWOBJ[j].uwMOCTRH & 0xFF00) >> 8)      // last MO in a list
      {
        // ReadPtr = BOT of the base object
        ubFIFOReadPtr[ubObjNr] = (ubyte)(CAN_HWOBJ[ubObjNr].uwMOFGPRL & 0x00FF);
      }
      else
      {
        // ReadPtr = PNEXT of the current selected slave
        ubFIFOReadPtr[ubObjNr] = (ubyte)((CAN_HWOBJ[j].uwMOCTRH & 0xFF00) >> 8);
      }

      // check if the previous message was lost
      if(CAN_HWOBJ[j].uwMOCTRL & 0x0010)                  // if set MSGLST
      {
        CAN_HWOBJ[j].uwMOCTRL = 0x0010;                   // reset MSGLST
        return(3);
```

```
        }
        pstObj->ubMOCfg = (ubyte)((CAN_HWOBJ[j].uwMOFCRH & 0x0F00)>>8); //MOFCRnH[11-8] DLC

        for(i = 0; i < pstObj->ubMOCfg; i++)
        {
          pstObj->ubData[i] = CAN_HWOBJ[j].ubData[i];
        }
        pstObj->ubMOCfg|=(ubyte)((CAN_HWOBJ[j].uwMOCTRL & 0x0800)>>7); // set DIR if tx obj

        if(CAN_HWOBJ[j].uwMOARH & 0x2000)              // if extended identifier
        {
          pstObj->ulID=(((ulong)(CAN_HWOBJ[j].uwMOARH & 0x1FFF))<<16)+CAN_HWOBJ[j].uwMOARL;

          pstObj->ubMOCfg = pstObj->ubMOCfg | 0x20;          // set IDE
        }
        else                                                 // standard identifier
        {
          pstObj->ulID = (CAN_HWOBJ[j].uwMOARH & 0x1FFF) >> 2;
        }
        pstObj->uwCounter = CAN_HWOBJ[j].uwMOIPRH;

        // check if the message was corrupted
        if(CAN_HWOBJ[j].uwMOCTRL & 0x0008)                   // if NEWDAT
        {
          CAN_HWOBJ[j].uwMOCTRL = 0x0008;                    // clear NEWDAT
          return(4);
        }
        ubReturn = 1;
      }
    }
  return(ubReturn);
} //  End of function CAN_ubReadFIFO
```

6.1 Functions CAN_ubWriteFIFO( ) and CAN_ubReadFIFO( ) in Test Applications.

```
//****************************************************************************
// @Function      void Test_vApplication(void)
//----------------------------------------------------------------------------
// @Description   This function Checks FIFO features.
//  NOTE: This function should be called in MAIN.C
//        Please make sure that Node 0 & Node 1 are connected by loop back wire.
//----------------------------------------------------------------------------
// @Returnvalue   None
//----------------------------------------------------------------------------
// @Parameters:     None
//****************************************************************************

void Test_vApplication(void)
{
ubyte i, k;
ubyte MaxRxFIFOobj = 3;

stCAN_SWObj SW_obj0, SW_obj1, SW_obj2;
// Message Objects of Different DATAs for TransmitFIFO MOs
SW_obj0.ubMOCfg = 0x58;
SW_obj0.ulID = 0x064;
SW_obj0.ubData[0] = 0x00; SW_obj0.ubData[1] = 0x01;
SW_obj0.ubData[2] = 0x02; SW_obj0.ubData[3] = 0x03;
SW_obj0.ubData[4] = 0x04; SW_obj0.ubData[5] = 0x05;
SW_obj0.ubData[6] = 0x06; SW_obj0.ubData[7] = 0x07;

SW_obj1.ubMOCfg = 0x58;
SW_obj1.ulID = 0x064;
SW_obj1.ubData[0] = 0x10; SW_obj1.ubData[1] = 0x11;
SW_obj1.ubData[2] = 0x12; SW_obj1.ubData[3] = 0x13;
```

```
SW_obj1.ubData[4] = 0x14; SW_obj1.ubData[5] = 0x15;
SW_obj1.ubData[6] = 0x16; SW_obj1.ubData[7] = 0x17;

SW_obj2.ubMOCfg = 0x58;
SW_obj2.ulID = 0x064;
SW_obj2.ubData[0] = 0x20; SW_obj2.ubData[1] = 0x21;
SW_obj2.ubData[2] = 0x22; SW_obj2.ubData[3] = 0x23;
SW_obj2.ubData[4] = 0x24; SW_obj2.ubData[5] = 0x25;
SW_obj2.ubData[6] = 0x26; SW_obj2.ubData[7] = 0x27;

Doprintf("\r\n***********************************");
Doprintf("\r\n TestCase ID : CAN_FIFO");
Doprintf("\r\n Used pin for UART: P7.3 -> TxD, P7.4 -> RxD");
Doprintf("\r\n Please Connect Loop Back Wire between NODE 0 & NODE 1");
Doprintf("\r\n***********************************\r\n");

Doprintf("\r\n Transmitting MsgObj[12] (CAN BASE OBJ).");
if(CAN_ubWriteFIFO(12, &SW_obj0) == 1)
  Doprintf("\r\nFIFO Tx Success...!");
else
  Doprintf("\r\nError: FIFO Writing...!");

if(CAN_ubWriteFIFO(12, &SW_obj1) == 1)
  Doprintf("\r\nFIFO Tx Success...!");
else
  Doprintf("\r\nError: FIFO Writing...!");

if(CAN_ubWriteFIFO(12, &SW_obj2) == 1)
  Doprintf("\r\nFIFO Tx Success...!");
else
  Doprintf("\r\nError: FIFO Writing...!\r\n");

if(ubIntrTest1 == 1)
  Doprintf("\r\n  OVIE INTR Occured !\r\n");
else
  Doprintf("\r\n  NO OVIE INTR Occured !\r\n");

Doprintf("\r\n\r\n Reading FIFO Objects...\r\n");
for (i = 0; i < MaxRxFIFOobj; i++)
{
  if(CAN_ubReadFIFO(2, &SWrd_obj) == 1)
  {
       Doprintf("\r\nFIFO Rx Success...!\r\n");
       sprintf(s, "MOCfg: %x ID: %x\r\n", SWrd_obj.ubMOCfg, SWrd_obj.ulID); Doprintf(s);
       Doprintf("          [ 0] [ 1] [ 2] [ 3] [ 4] [ 5] [ 6] [ 7]\r\n-DATAs -");
       for ( k = 0; k < 8; k++)
       {
         sprintf(s, " 0x%02x", SWrd_obj.ubData[k]); Doprintf(s);
       }
  }
  else
  {
       Doprintf("\r\nError: FIFO Reading...!\r\n");
  }
}
Doprintf("\r\n\r\n");
}//end of function Test_vApplication()
```

7 Abbrevation

CAN	Controller Area Network, a message object oriented serial communication protocol
MCAN	MultiCAN
BOT	Bottom Object Pointer
CUR	Current Object Pointer; links to the actual target object within a FIFO/Gateway structure.
TOP	Top Object Pointer
SEL	Object Select Pointer
MO	CAN Message Object
SRN	Service Request Node

8 Related Documents and Links

Referred Document:

User's Manual, V1.0, June 2007, Vol 1, System Unit
User's Manual, V1.0, June 2007, Vol 2, Peripheral Unit

XC2000 Product Information:

http://www.infineon.com/cms/en/product/channel.html?channel=ff80808112ab681d0112ab6b687b0811

Starter Kit (Board Manual) SK-EB XC2287 Starter Kit:

http://www.infineon.com/cms/en/product/channel.html?channel=db3a304312dc768d0112e740471d0bcb

XC2000 Development Tools and Software:

http://www.infineon.com/cms/en/product/channel.html?channel=ff80808112ab681d0112ab6b4b7407b7

DAvE for the Infineon XC2000 microcontroller Family:

http://www.infineon.com/cms/en/product/channel.html?channel=db3a304316f66ee80117630459b369df

http://www.infineon.com

Published by Infineon Technologies AG