

Configuring a CAN Node on Hercules™ ARM® Safety MCUs

Pratip Kumar

ABSTRACT

The Hercules family of microcontrollers from Texas Instruments is a family of 32-bit RISC microcontrollers with an advanced ARM architecture and a rich peripheral set.

This application report describes a typical procedure to configure a CAN node and perform CAN communication over the network.

Contents

1	Introduction	1
2	Initializing the CAN Peripheral	3
3	Configuring Message Objects	8
4	Software Flow	12
Appendix A Sample Bit Timings		13

List of Figures

1	CAN Transceiver Interface.....	2
2	A Typical CAN Network	3
3	CAN Module Block Diagram.....	3
4	CAN Baud Rate Configuration	5
5	IF1/2 Register Sets	7
6	IF3 Register Set.....	7
7	CAN Message Object Configuration - Logical Diagram	8
8	Handling Reception of CAN Messages.....	12
9	Sample Bit Timing Calculation for 500 kb.....	13
10	Sample Bit Timing Calculation for 1000 kb	14

1 Introduction

1.1 Controller Area Network (CAN)

The controller area network is a serial multi-master communication protocol that efficiently supports distributed real-time control, with a high level of security, and a communication rate of up to 1Mbps.

The CAN bus is ideal for applications operating in noisy and harsh environments, such as in the automotive and other industrial fields that require reliable communication.

Hercules is a trademark of Texas Instruments.
 ARM is a registered trademark of ARM Limited.
 All other trademarks are the property of their respective owners.

The peripheral supports the following features:

- Protocol:
 - Supports CAN protocol version 2.0 part A, B
- Clock and Speed:
 - Bit rates up to 1 MBit/s
 - Dual clock source
- Message Object / Message Box:
 - 16, 32, 64 or 128 message objects (device specific)
 - Individual identifier mask for each message object
 - Programmable FIFO mode for message objects
- Interrupts:
 - Two interrupt lines to interrupt module
- Low-Power Support:
 - Global power down and wakeup support
 - Local power down and wakeup support
- Burst Transfer Support
 - Data transfer through DMA
- Debug Support:
 - Programmable loop-back modes for self-test operation
 - Direct access to Message RAM during test mode
 - Suspend mode for debug support
- Others:
 - Automatic bus on after Bus-Off state by a programmable 32-bit timer
 - CAN Rx / Tx pins configurable as general purpose IO pins

1.2 CAN Transceiver Interface Diagram

Figure 1 shows a typical CAN transceiver interface.

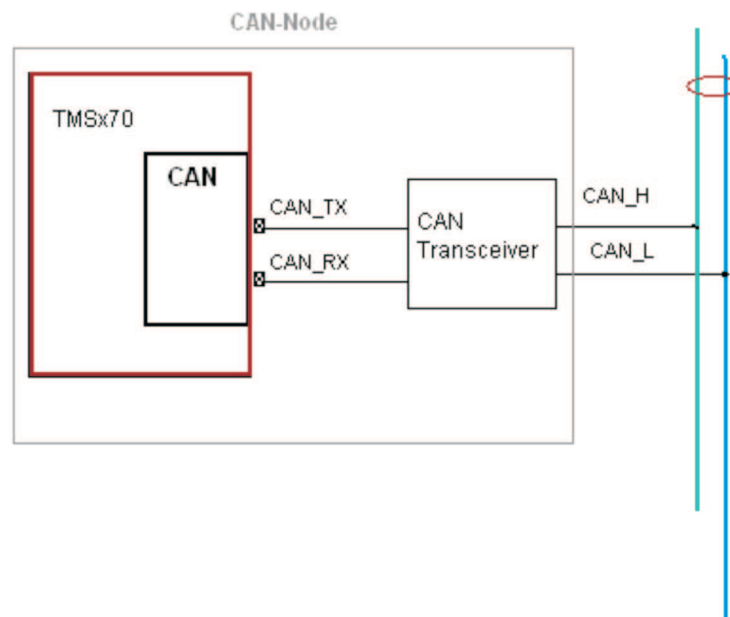


Figure 1. CAN Transceiver Interface

1.3 Typical CAN Network

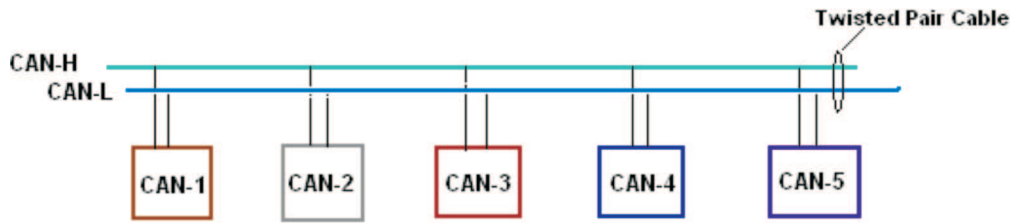


Figure 2. A Typical CAN Network

1.4 CAN Block Diagram

Figure 3 shows the core internal blocks of the CAN controller.

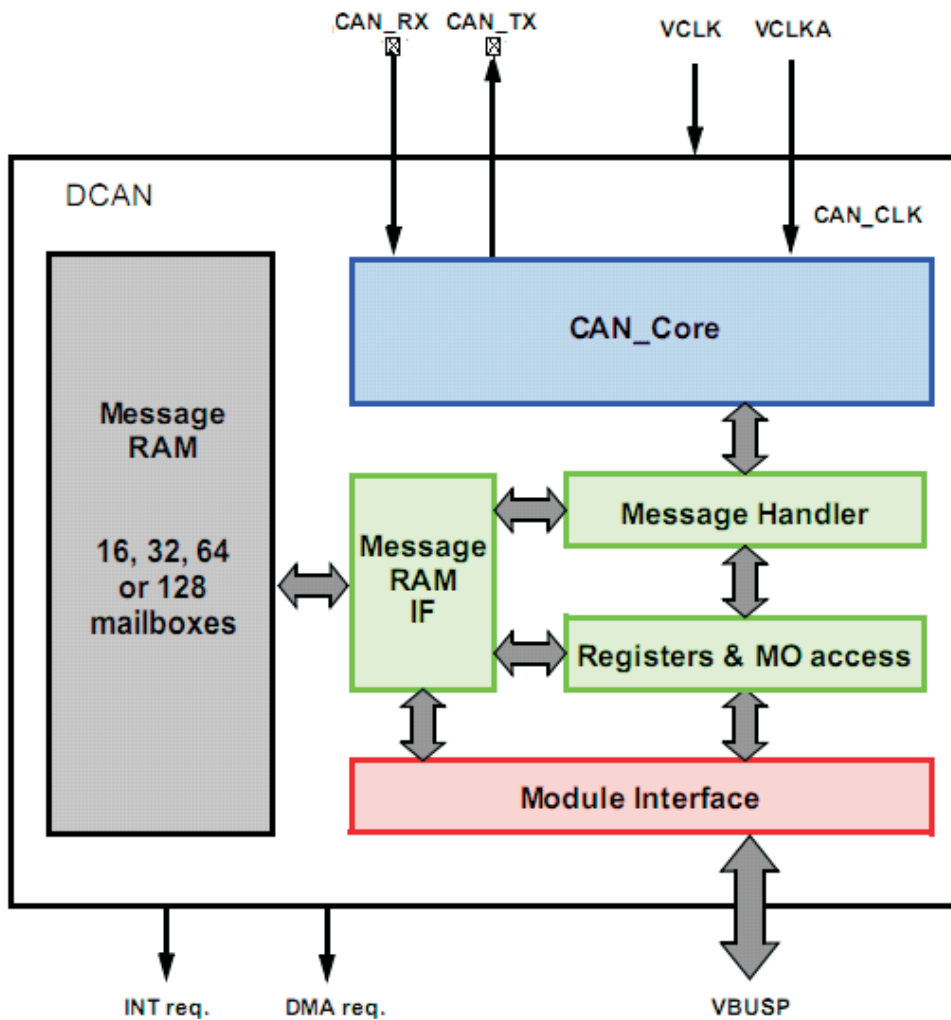


Figure 3. CAN Module Block Diagram

2 Initializing the CAN Peripheral

The following common steps are involved in initializing the CAN.

- CAN RAM initialization (recommended during boot up).
- CAN configuration

- CAN baud rate configuration

After initializing the CAN, the Mailbox needs to be configured as required by the application.

The following sections cover the CAN initialization process in detail.

2.1 CAN RAM Initialization

The CAN RAM holds the CAN message objects (also called mail box). To begin, the RAM space should be initialized to zeros through the hardware by configuring the system registers.

To initialize CAN RAM:

1. Switch to memory-initialization mode: `MINITGCR.MINITGNA = 0xA`
2. Set the MSINENA. Memory Initialization Enable Register (MSIENAx) bit corresponding to the DCAN RAM.
3. Wait for the memory initialization complete by checking the Memory Hardware Initialization Status Register (MINISTAT).
4. Come out of memory-initialization mode.

Sample code to initialize RAM:

```
SYS_Ptr->MINITGCR_UN.MINITGCR_UL = 0xA;
SYS_Ptr->MSINENA_UL                = MEM_CH_DCANRAM;
while (SYS_Ptr->MINISTAT_UL == 0);
SYS_Ptr->MINITGCR_UN.MINITGCR_UL = 0x5;
```

NOTE: For MEM_CH_DCANRAM, see the *Memory Initialization* table in the device-specific data sheet.

If you plan to use the parity or ECC, you need to enable it prior to the RAM initialization procedure. The following is the sample code to enable parity/ECC:

```
DCAN1_Ptr->CCR_UN.CCR_ST.PMD_B4 = 0xA
```

2.2 CAN Configuration

The following features of CAN are configurable through software:

- Disable/enable automatic wakeup on bus activity. [Bit: CCR.WUBA-Wake up on Bus Activity]
- Enable/disable auto bus on timer. [Bit: CCR.ABO-Auto Bus On]
- Enable/disable parity/ECC. [Bit: CCR.PMD-Parity Mode Disable]
- Enable/disable Global Interrupt Line 0 and 1. [Bit: CCR.IEx- Interrupt Enable]
- Disable/enable error interrupts. [Bit: CCR.EIE-Error Interrupt Enable]
- Disable/enable status interrupts. [Bit: CCR.SIE-Status Interrupt Enable]
- Setup automatic retransmission of messages. [Bit: DAR-Disable Automatic Retransmission]
- Request write access to the configuration registers. [Bit: CCR.CCE-Configuration Change Enable]
- Setup message completions before entering debug state. [Bit: CCR.IDS]
- Disable/enable local power-down mode. [Bit: CCR.PDR]
- Disable/enable DMA request lines. [Bit: DEX]
- Enable/disable test mode. [Bit: CCR.Test]

Sample CAN Configuration Code:

```

Void CanInit (void
{
    /**@b Initializ @b DCAN1: */

    canREG1->CTL = 0x00021443U; /* Configure CAN */

    /** - Clear all pending error flags and reset current status */
    canREG1->ES = 0x0000031Fu;

    /** - Assign interrupt level for messages */
    canREG1->INTMUXx[0U] = MessageBoxNo;
    canREG1->INTMUXx[1U] = 0x00000000U;

    /** - Setup auto bus on timer period */
    canREG1->ABOTR = 0U;
}
    
```

2.3 Configuring CAN Baud Rate

The CAN data transfer baud rate is set by calculating the bit timing value and programming this value in the Bit Timing Register (BTR).

To initialize the CAN baud rate:

1. Set CANCONTROL.init bit. This will put the CAN in initialization mode.
2. Set the Configuration Change Enable (CANCONTROL.CCE) bit. This will enable write access to the BTR register.
3. Write the calculated bit timing component values into the BTR register.
4. Clear the CANCONTROL.CCE bit followed by the CANCONTROL.init bit.

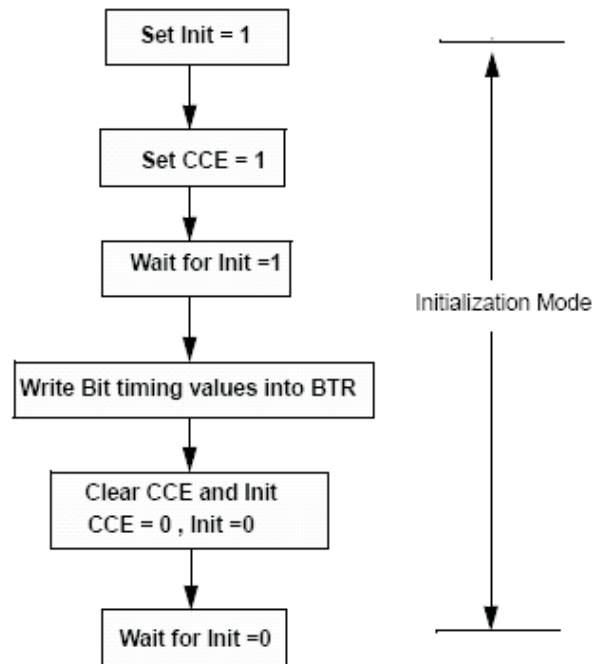


Figure 4. CAN Baud Rate Configuration

```

/* Baudrate configuration */
DCAN1_Ptr->CCR_UN.CCR_ST.INIT_B1 = 1 ;          /* Enter Initialization Mode */
DCAN1_Ptr->CCR_UN.CCR_ST.CCE.B1 = 1 ;

while(DCAN1_Ptr->CCR_UN.CCR_ST.INIT_B1 == 0) ;

/* With CAN clock (CAN_CLK) of 8 MHz and BRPE = 0x00,
   the BRT value of 0x2301 configures the CAN
   for a bit rate of 500kBits/s. */

DCAN1_Ptr->CCR_UN.BTR_UL = 0x2301 ;

DCAN1_Ptr->CCR_UN.CCR_ST.CCE.B1 = 0 ;
DCAN1_Ptr->CCR_UN.CCR_ST.INIT_B1 = 0 ;
while(DCAN1_Ptr->CCR_UN.CCR_ST.INIT_B1 == 1) ; /* Enter Normal Mode */

```

2.4 CAN Message Objects

The structure of the CAN message object (also known as CAN Mailbox) is shown in [Table 1](#).

Table 1. CAN Message Object

Message Object												
UMask	Msk[28:0]	MXtd	MDir	EoB	Unused	NewDat	MsgLst	RxIE	TxE	IntPnd	RmtEn	TxRqst
MsgVal	ID[28:0]	Xtd	Dir	DLC[3:0]	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7

[Table 1](#) has the following three key components:

- ID: Message ID (with Extended Message ID, Xtd)
- DLC: Message Length
- Datax : Message Data (up to 8 Bytes)

Also, [Table 1](#) has the following control bits:

- Message Valid (**MsgVal**): Indicates that the message is valid.
- Direction (**Dir**): Indicates whether the mailbox is to transmit or receive
- Identifier Mask (**Msk**): Indicates the bits that are to be masked in ID
- Mask Extended Identifier (**MXtd**): Indicates the Mask bits for extended ID Xtd
- Mask Message Direction (**MDir**): Indicates whether or not the Dir is supposed to be masked
- Use Acceptance Mask (**UMask**): Indicates whether or not Mask bits are to be used
- End of Block (**EoB**): Indicates the last message of the FIFO buffer
- Transmit Interrupt Enable (**TxE**): Provides an interrupt after transmission of the data.
- Receive Interrupt Enable (**RxIE**): Provides an interrupt after reception of the data.
- Interrupt Pending (**IntPnd**) : Indicates that interrupt is pending for this message object.
- New Data (**NewDat**): Indicates that new data is available in the message object.
- Transmit Request (**TxRqst**): Requests the transmission of the data
- Remote Enable (**RmtEn**): Enables the message object to accept remote frame.

2.5 CAN Interface Registers

The message objects can be accessed only through the Interface Registers (IFx). This is to avoid arbitration that would occur if both CPU and DMA try to access the message objects. During each IFx access, selected items in the message objects can be updated as required. Only one message object can be accessed (for read/write) at a time. There are a total of three interface registers. During the operation of the interface registers (IFx), the corresponding busy bit stays high.

NOTE: For details on the number of supported CAN Mail boxes, see the device-specific data sheet.

Message object 1 has the highest priority, while the last implemented message object has the lowest priority. If more than one transmission request is pending, they are serviced due to the priority.

The interface register IF1 and IF2 are identical.

Message Interface Register Sets 1 and 2

Address [CAN Base +]	31	IF1 Register Set 16 15	0	Address [CAN Base +]	31	IF2 Register Set 16 15	0
0x100		IF1 Command		0x120		IF2 Command	
0x104		IF1 Mask		0x124		IF2 Mask	
0x108		IF1 Arbitration		0x128		IF2 Arbitration	
0x10C		IF1 Message Control		0x12C		IF2 Message Control	
0x110		IF1 Data A		0x130		IF2 Data A	
0x114		IF1 Data B		0x134		IF2 Data B	

Figure 5. IF1/2 Register Sets

The command register specifies the direction of the data transfer (from/to message object) and which portion of a message object that is to be transferred and selects a message object in the message RAM as target or source for the transfer.

The arbitration register enables/disables the message object with the message ID and configures it as transmit or receive.

The message control register defines the data size and other configurations that is to be done during initialization of the message object.

The data register holds the data that is to be transferred or that is received.

Each message object can be configured as required with the interface registers.

The IF3 register set can be automatically updated with the received message object without the need to initiate the transfer from message RAM. Automatic update functionality can be individually programmed for each message object.

Message Interface Register Set 3

Address [CAN Base +]	31	IF3 Register Set 16 15	0
0x100		IF3 Observation	
0x104		IF3 Mask (Read Only)	
0x108		IF3 Arbitration (Read Only)	
0x10C		IF3 Message Control (Read Only)	
0x110		IF3 Data A (Read Only)	
0x114		IF3 Data B (Read Only)	

Figure 6. IF3 Register Set

3 Configuring Message Objects

Basically, a message object can be configured as:

- Transmit message object
- Receive message object

Figure 7 is a logical diagram for configuring a CAN message object.

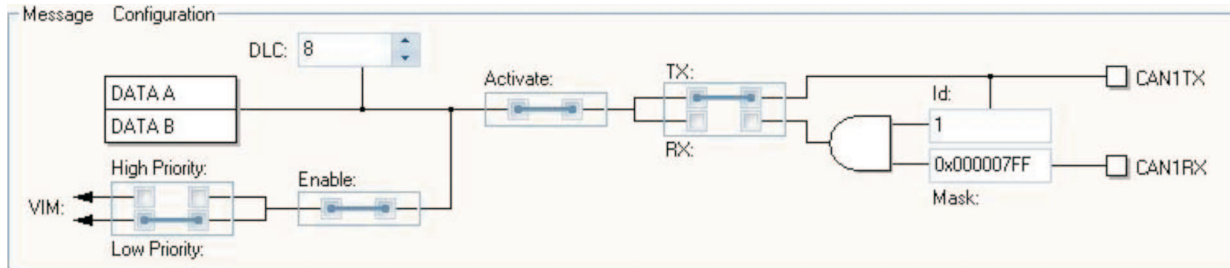


Figure 7. CAN Message Object Configuration - Logical Diagram

The configuration of the message objects has to be done through the interface registers (IFx).

The minimal steps involved to configure a message object would be:

1. Wait until IFx is ready for use.
2. Set message mask.
3. Set message arbitration.
4. Set message control word.
5. Set IFx control byte.
6. Set IFx message number.

CAN communication can be carried out by any of the two methods:

- Interrupt method
- Polling method

3.1 Configuring CAN Message Object to Transmit

The following is a sample code to configure a transmit message object:

```
while (canREG1->IF1STAT & 0x80) ;

canREG1->IF1MSK = 0xC0000000U | ((0x000007FFU & 0x1FFFFFFFU0 << 0U) ;
canREG1->IF1ARB = 0x80000000U | 0x40000000U | 0x20000000U | ((1U & 0x1FFFFFFFU) << 0U) ;
canREG1->IF1MCTL = 0x00001080U | 0x00000C00U | 8U ;
canREG1->IF1CMD = 0x20 ;
canREG1->IF1NO = 1 ;
```

Sample Code Explanation:

1. Wait if IF1 is busy.
2. Configure mask register.

MXtd	= 1	Use Extended ID Mask
MDir	= 1	Use message direction mask
Msk	= 0x7FF	Use bits 10:0 for filtering

3. Configure arbitration register.

MsgVal	=	1	Enable the message object
Xtd	=	1	Extended 25 bit identifier
Dir	=	1	Transmit mail box
ID		1	Message ID 0x1

4. Configure message control register.

UMask	=	1	Use mask for filtering
EoB	=	1	Single message object
TxIE	=	1	Enable transmit interrupt
RxIE	=	1	Enable receive interrupt
DLC	=	8	Set data length as 8

5. Configure command register.

Message Number	=	0x20	Message number is 0x20
-----------------------	---	------	------------------------

3.2 Configuring CAN Message Object to Receive

The following is a sample code to configure a receive message object:

```
while (canREG1->IF2STAT & 0x80) ;

canREG1->IF2MSK = 0xC0000000U | ((0x000007FFU & 0x1FFFFFFFU0 << 0U) ;
canREG1->IF2ARB = 0x80000000U | 0x40000000U | 0x00000000U | ((2U & 0x1FFFFFFFU) << 0U) ;
canREG1->IF2MCTL = 0x00001080U | 0x00000C00U | 8U ;
canREG1->IF2CMD = 0x22 ;
canREG1->IF2NO = 2 ;
```

Sample Code Explanation:

1. Wait if IF2 is busy.
2. Configure mask register.

MXtd	=	1	Use Extended ID Mask
MDir	=	1	Use message direction mask
Msk	=	0x7FF	Use bits 10:0 for filtering

3. Configure arbitration register.

MsgVal	=	1	Enable the message object
Xtd	=	1	Extended 25 bit identifier
Dir	=	0	Receive mail box
ID		2	Message ID 0x2

4. Configure message control register.

UMask	=	1	Use Mask for filtering
EoB	=	1	Single message object
TxIE	=	1	Enable transmit interrupt
RxIE	=	1	Enable receive interrupt
DLC	=	8	Set data length as 8

5. Configure command register.

Message Number = 0x22 Message number is 0x22

3.3 CAN Transmit/Receive Operation

```

unsigned canTransmit (canBASE_t *node, unsigned messageBox, const unsigned char *data)
{
    unsigned I;
    unsigned success = 0U;
    unsigned regIndex = (messageBox - 1U) >> 5U;
    unsigned bitIndex = 1U << ((messageBox - 1U) & 0x1FU);

    /** - Check for pending message:
    *     - pending message, return 0
    *     - no pending message, start new transmission
    */
    if (node->TXRQx[regIndex] & bitIndex)
    {
        return success;
    }

    /** - Wait until IF1 is ready for use */
    while (node->IF1STAT & 0x80);

    /** - Copy TX data into IF1 */
    for (I = 0U < 8U; I++)
    {
        node->IF1DATx[s_canByteOrder[i]] = *data++;
    }

    /** - Copy TX data into message box */
    node->IF1NO = messageBox;

    success = 1U;

    /** @note The function canINIT has to be called before this function can be used. \n
    *       The user is responsible to initialize the message box.
    */
    return success;
}
    
```

The following is a sample code to receive CAN data:

```

unsigned canGetData (canBASE_t *node, unsigned messageBox, unsigned char * const data)
{
    unsigned          I;
    unsigned          size;
    unsigned char     *pData   = (unsigned char *) data;
    unsigned          success  = 0U;
    unsigned          regIndex = (messageBox - 1U) >> 5U;
    unsigned          bitIndex = 1U << ((messageBox - 1U) & 0x1FU);

    /** - Check if new data has arrived:
    *     - no new data, return 0
    *     - new data, get received message
    */
    if (!(node->NWDATx[regIndex] & bitIndex))
    {
        return success;
    }

    /** - Wait until IF2 is ready for use */
    while (node->IF2STAT & 0x80);

    /** - Copy data into IF2 */
    node->IF2NO = messageBox

    /** - Wait until data are copied into IF2 */
    while (node->IF2STAT & 0x80);

    /** - Get number of received bytes */
    size = node->IF2MCTL & 0xFU;

    /** - Copy RX data into destination buffer */
    for (I = 0U; I < size; I++)
    {
        *pData++ = node->IF2DATx[s_canByteOrder[i]];
    }

    success = 1U;

    /** - Check if data has been lost:
    *     - no data lost, return 1
    *     - data lost, return 3
    */
    if (node->IF2MCTL & 0x4000U)
    {
        success = 3U;
    }

    return success;
}

```

Figure 8 describes the flow to handle receive interrupts.

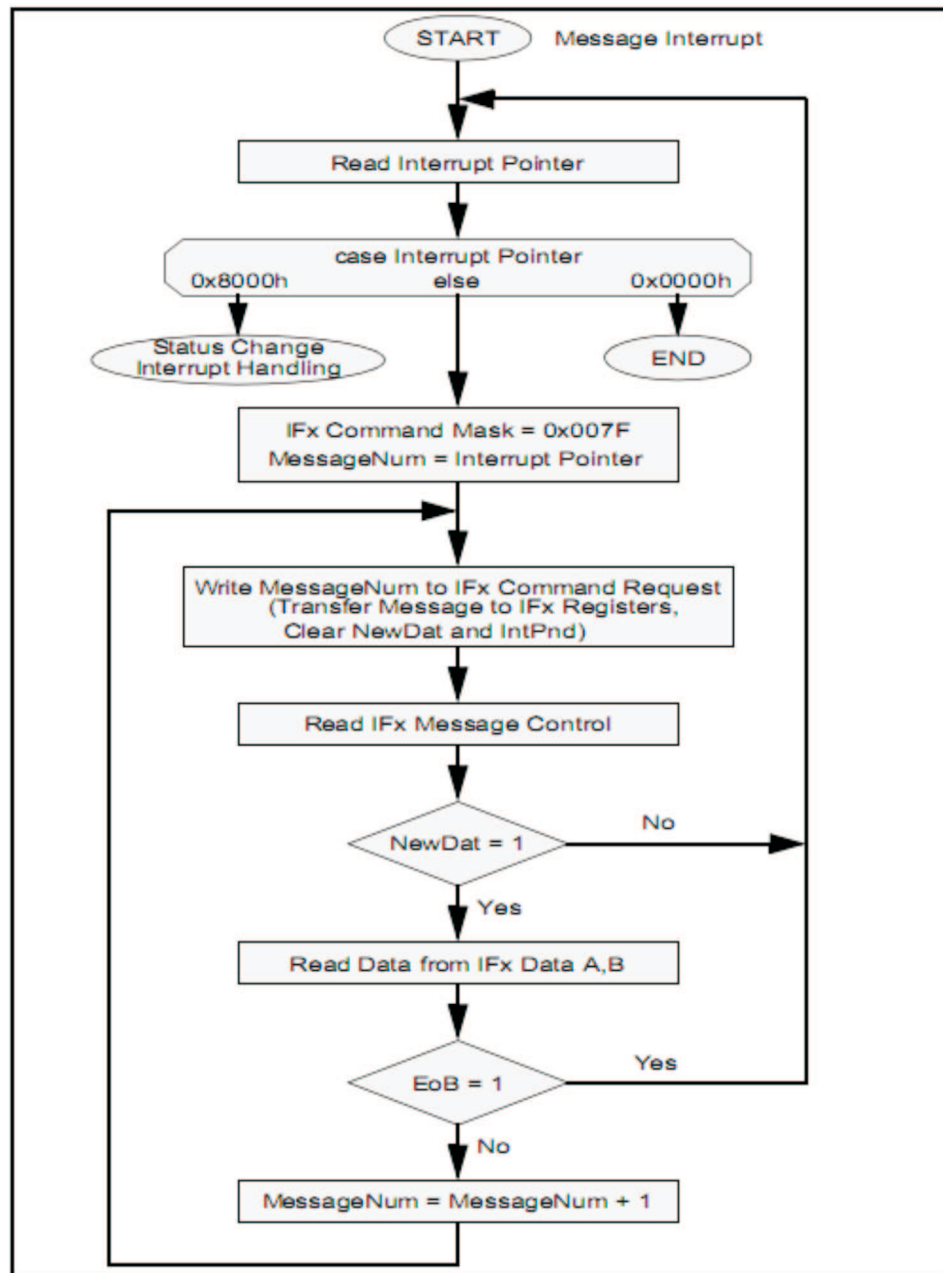


Figure 8. Handling Reception of CAN Messages

4 Software Flow

This section shows the software flow summary for configuring a CAN node.

1. Initialize CAN RAM space for message objects.
2. Configure CAN general parameters (DMA, interrupts, automatic wakeup, etc).
3. Configure the required CAN baud rate.
4. Configure the required Message objects with required IDs and masks.
5. Provide the transmit/receive routines depending on interrupt mode or polling mode.

Appendix A Sample Bit Timings

Figure 9 and Figure 10 illustrate sample bit timings calculated for a bit rate of 500 and 1000 that are to be programmed into the bit timing register.

Sample Bit Timing Calculation for Bit Rate = 500

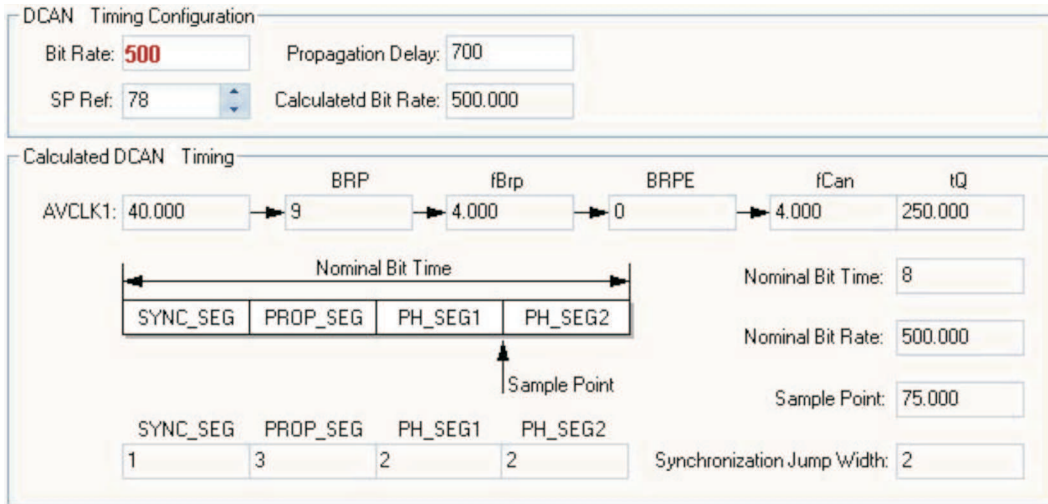


Figure 9. Sample Bit Timing Calculation for 500 kb

Bit timing register values:

$$BRPE = (0U \ll 16U)$$

$$TSEG2: PHASE SEG2 = ((2U - 1U) \ll 12U)$$

$$TSEG1: PROP SEG + PHASE SEG1 = (((3U + 2U) - 1U) \ll 8U)$$

$$SJW: ((2U - 1U) \ll 6U)$$

$$BRP = 9U;$$

Sample Bit Timing Calculation for Bit Rate = 1000

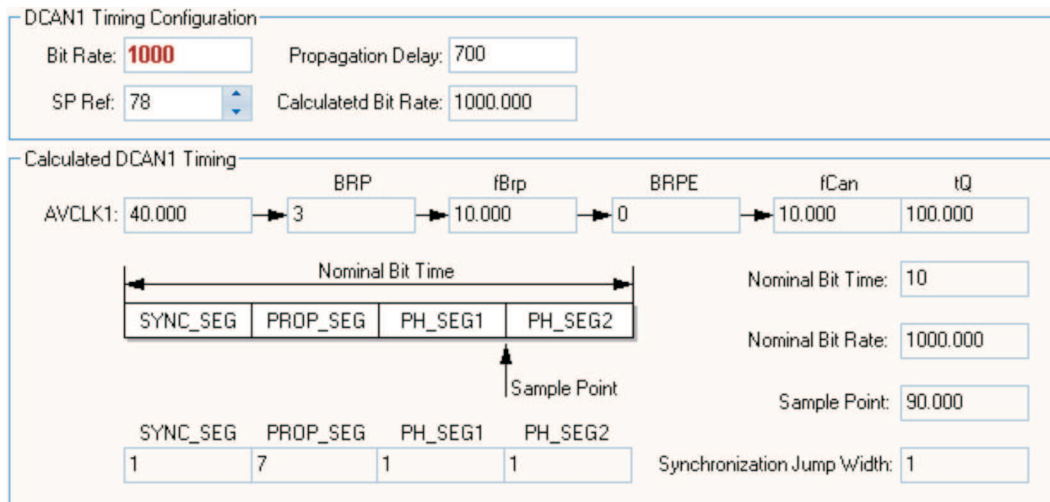


Figure 10. Sample Bit Timing Calculation for 1000 kb

Bit timing register values:

BRPE = (0U << 16U)

TSEG2: PHASE SEG2 = ((1U - 1U) << 12U)

TSEG1: PROP SEG + PHASE SEG1 = (((7U + 1U) - 1U) << 8U)

SJW: ((1U - 1U) << 6U)

BRP = 3U;

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2022, Texas Instruments Incorporated