
ECAN™ Operation with DMA on dsPIC33F and PIC24H Devices

<i>Author: Jatinder Gharoo Microchip Technology Inc.</i>
--

INTRODUCTION

This application note is focused on helping customers understand the role of Direct Memory Access (DMA) in implementing the functionality of the Enhanced Controller Area Network (ECAN™) module.

This material will be of interest to engineers who use the CAN protocol for communication.

The information presented assumes you have a working knowledge of the CAN protocol. For those who are new to CAN, refer to the following resources available from Microchip:

- CAN resources such as application notes and Web seminars can be accessed at: www.microchip.com/CAN
- Sample code for various dsPIC® DSC devices can be accessed at: www.microchip.com/codeexamples
- Our Regional Training Centers (RTC) can help you get started with ECAN and offer a range of classes. For more information, visit: www.microchip.com/rtc
- Additional material at the end of this application note includes references to literature and vocabulary

OVERVIEW

The ECAN module works in conjunction with the DMA controller in dsPIC33F and PIC24H devices. The DMA controller is a very important subsystem in Microchip's high-performance 16-bit dsPIC33F and PIC24H devices. The DMA controller allows data transfer from RAM to a peripheral and vice versa without any CPU assistance, and operates across its own data bus and address bus with no impact on CPU operation.

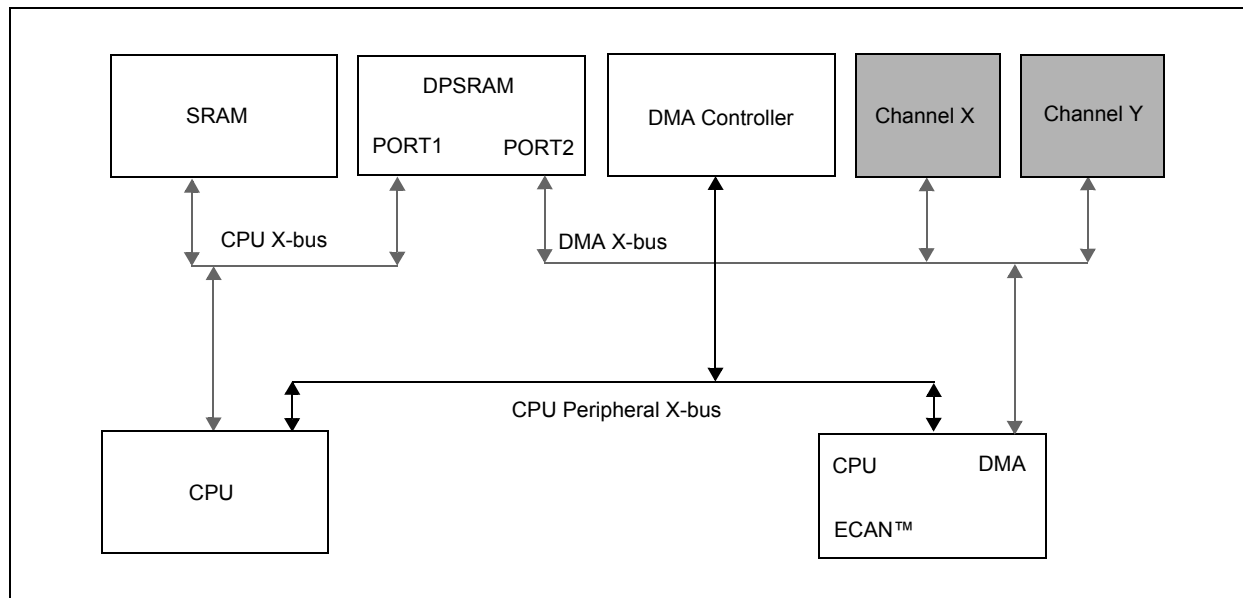
The DMA subsystem supports eight independent channels. Because each channel is unidirectional, two channels must be allocated to read and write to the ECAN peripheral using DMA. One channel is allocated for reading messages from the ECAN peripheral and the other channel is allocated for writing messages to the ECAN peripheral.

When more than one DMA channel receives a request to transfer data, a simple fixed-priority scheme that is based on the channel number dictates the specific channel that completes the transfer and the channels that are left pending. Each channel has a fixed priority. The channels with a lower number have higher priority, with channel 0 having the highest priority, and channel 7 having the lowest priority.

Each dsPIC33F or PIC24H device contains up to 2 Kbytes of Dual Port SRAM (DPSRAM), which is adequate to concurrently support multiple buffers for several peripherals. Figure 1 highlights the DMA integration with the architecture of dsPIC33F and PIC24H devices. The CPU communicates with conventional SRAM across the data space X-bus known as the CPU X-bus, as shown in Figure 1. It also communicates to port 1 of the new dual port SRAM block across the same X-bus.

The CPU communicates to the ECAN peripheral across a separate peripheral data space bus known as the CPU Peripheral X-bus, shown in Figure 1, which also resides in the X data space. The DMA controller communicates with port 2 of the dual port SRAM and the DMA port of ECAN module across a dedicated DMA transfer bus known as the DMA X-bus.

FIGURE 1: ECAN™ DMA BLOCK DIAGRAM



Note: Microchip's 16-bit CPU architecture is capable of read and write access within each CPU bus cycle. The DMA read and write timing is the same as the CPU timing, and can complete the transfer of a byte or a word in every bus cycle across its dedicated bus. This also guarantees that all DMA transfers are atomic. This ensures that once the data transfer has started, it is completed within the same cycle, regardless of the activity of other channels.

Microchip's ECAN module on the dsPIC33F or PIC24H device can be used with or without DMA to send and receive messages. The biggest advantage of using DMA with ECAN is that the data can be moved without involving the CPU or stealing CPU cycles. This implementation is optimized for performance of a real-time embedded application where system latency is a priority and CPU timing must be predictable.

CAN Data Frames

A CAN network can be configured to communicate with both of the following formats:

- Standard format - intended for standard messages that use 11 identifier bits
- Extended format - intended for extended messages that use 29 identifier bits

The ECAN module on the dsPIC33F and PIC24H devices supports both the standard and extended formats.

The ECAN module distinguishes between the CAN standard frame and CAN extended frame using the IDE bit, which is part of the ECAN message that is transmitted as dominant (logical '0') for an 11-bit frame (standard), and recessive (logical '1') for a 29-bit frame (extended).

The CAN bus frame consists of two main fields:

- User-controlled field
- Module-controlled field

The user specifies the ID and message data to which the ECAN module adds the applicable fields to ensure that the message frame meets the CAN specification.

Standard Data Frames

The standard data frame messages start with a Start-of-Frame (SOF) bit followed by the message. The user application provides the following fields to the ECAN module:

- Arbitration field
- Control field
- Data field

The Cyclic Redundancy Check (CRC), Acknowledge (ACK) and End-of-Frame (EOF) fields are automatically appended by the ECAN module to the user-provided fields and are sent over the CAN bus as a single message.

FIGURE 2: STANDARD DATA FRAME

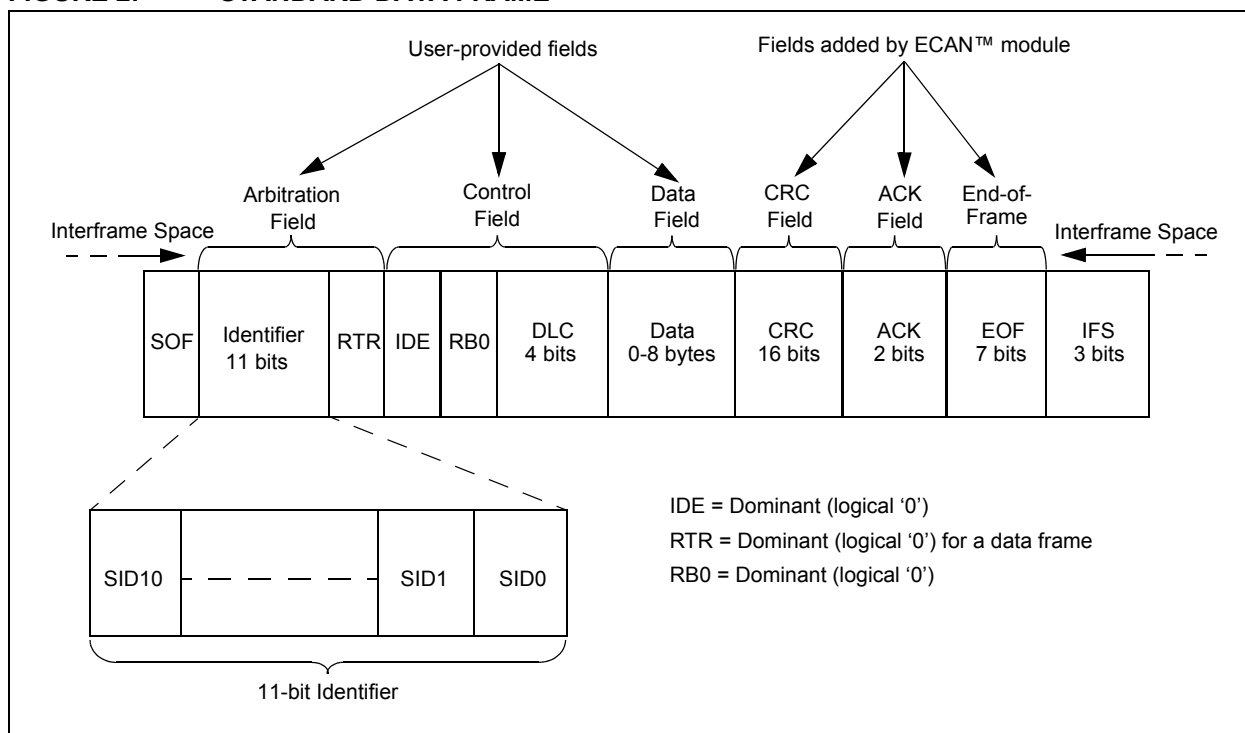


TABLE 1: ECAN™ STANDARD FRAME MESSAGE FIELDS

Field	Length	Application Usage
Start-of-Frame (SOF)	1 bit	Indicates the start of frame transmission.
Identifier A	11 bits	A (unique) identifier for the data.
Remote Transmission Request (RTR)	1 bit	Can be dominant in Data frame (logical '0') or recessive (logical '1').
Identifier Extension bit (IDE)	1 bit	Must be dominant (logical '0').
Reserved bit (RB0)	1 bit	Must be set to dominant (logical '0').
Data Length Code (DLC)	4 bits	Number of bytes of data (0-8 bytes).
Data field	0-8 bytes	Data to be transmitted (length dictated by DLC field).
CRC	15 bits	Cyclic redundancy check.
CRC delimiter	1 bit	Must be recessive (logical '1').
ACK slot	1 bit	Transmitter sends recessive (logical '1') and a receiver will assert a dominant (logical '0'), if message is received with no errors.
ACK delimiter	1 bit	Must be recessive (logical '1').
End-of-Frame (EOF)	7 bits	Must be recessive (logical '1').

AN1249

Extended Data Frames

Extended data frame messages start with a Start-of-Frame (SOF) bit followed by the message. The user application provides the following fields to the ECAN module:

- Arbitration field
- Control field
- Data field

The Cyclic Redundancy Check (CRC), Acknowledge (ACK), and End-of-Frame (EOF) fields are automatically appended by the ECAN module to the user-provided fields and are sent over the CAN bus as a single message.

FIGURE 3: EXTENDED DATA FRAME

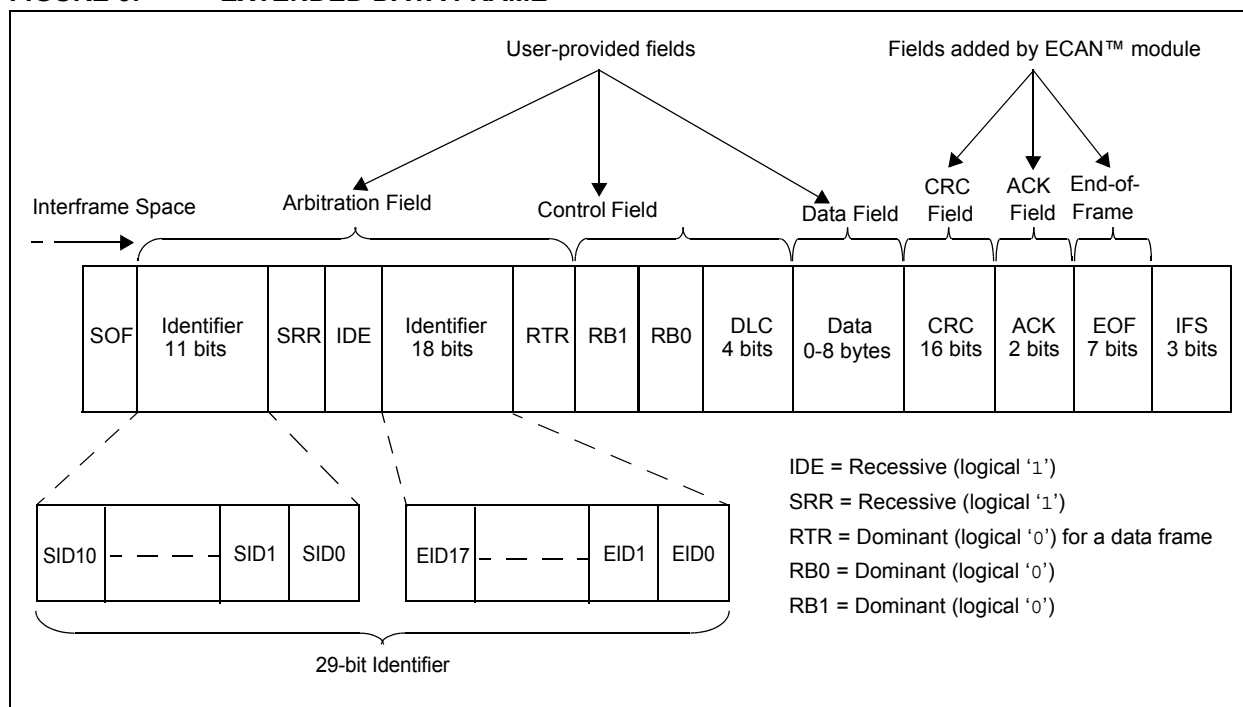


TABLE 2: ECAN™ EXTENDED FRAME MESSAGE FIELDS

Field	Length	Application Usage
Start-of-Frame (SOF)	1 bit	Indicates the start of frame transmission.
Identifier A	11 bits	First part of the (unique) identifier for the data.
Substitute Remote Request (SRR)	1 bit	Must be recessive (logical '1').
Identifier Extension bit (IDE)	1 bit	Must be recessive (logical '1').
Identifier B	18 bits	Second part of the (unique) identifier for the data.
Remote Transmission Request (RTR)	1 bit	Can be dominant in Data frame (logical '0') or recessive (logical '1').
Reserved bit (RB0, RB1)	2 bits	Must be set to dominant (logical '0').
Data Length Code (DLC)	4 bits	Number of bytes of data (0-8 bytes).
Data field	0-8 bytes	Data to be transmitted (length dictated by DLC field).
CRC	15 bits	Cyclic redundancy check.
CRC delimiter	1 bit	Must be recessive (logical '1').
ACK slot	1 bit	Transmitter sends recessive (logical '1') and a receiver will assert a dominant (logical '0'), if message is received with no errors.
ACK delimiter	1 bit	Must be recessive (logical '1').
End-of-Frame (EOF)	7 bits	Must be recessive (logical '1').

ECAN MODULE OVERVIEW

The dsPIC33F and PIC24H ECAN module implements the CAN Protocol 2.0B, which is used in a variety of applications. The differential serial data communication has been designed to be a robust means of communication in an electrically noisy environment. The ECAN module consists of a CAN protocol engine and message filters, along with masks and a transmit and receive interface with the DMA module.

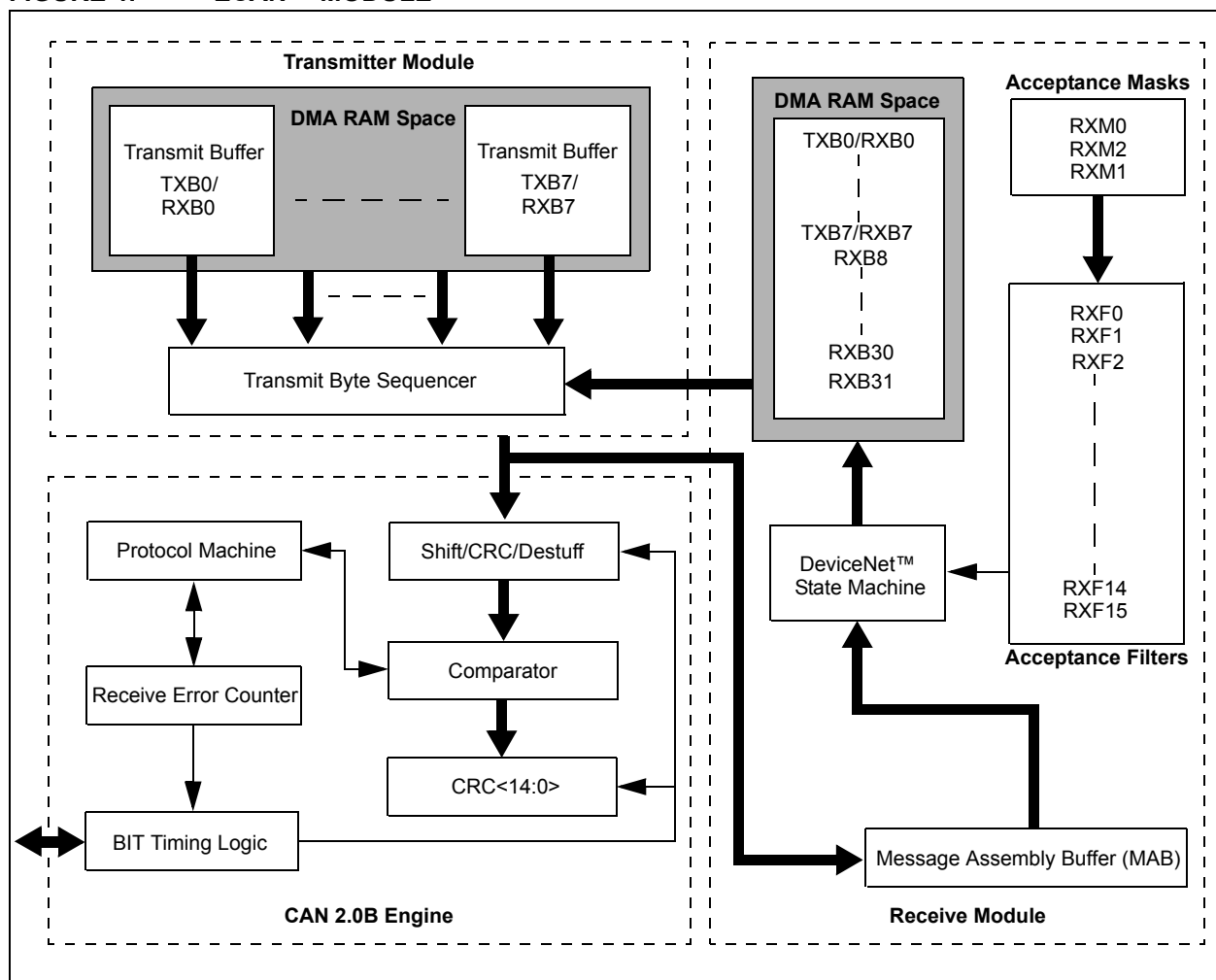
The ECAN module can operate in one of the following user-selectable modes:

- Configuration mode
- Normal mode
- Listen only mode
- Listen All Messages mode
- Loopback mode
- Disable mode

For a detailed description of these operating modes, refer to **Section 21. “Enhanced Controller Area Network (ECAN™)”** (see “References”). Normal mode of operation is the most widely used mode for the ECAN peripheral. The peripheral is designed to be used with the DMA module for dsPIC33F and PIC24H devices.

It can also be used without DMA to send and receive messages from the CAN bus. However, this method is not recommended as it defeats the advanced architecture of the dsPIC33F and PIC24H devices.

FIGURE 4: ECAN™ MODULE



AN1249

DMA MODULE OVERVIEW

Direct Memory Access (DMA) is a subsystem that allows the user to move data from one module to another without CPU intervention. This feature allows data transfer to and from peripherals with much less CPU overhead than those without a DMA module. This is highly efficient, if the system is operating on a high traffic CAN bus. The CPU can be interrupted only when the receive buffers must be serviced. The DMA module allows the flexibility to select when the CPU should be interrupted for message processing.

The DMA consists of a DMA controller and eight channels that allow the module to interface with different peripherals. The DMA subsystem uses DPSRAM and a register structure that allows the DMA module to operate across its own, independent data bus and address bus with no impact on CPU operation.

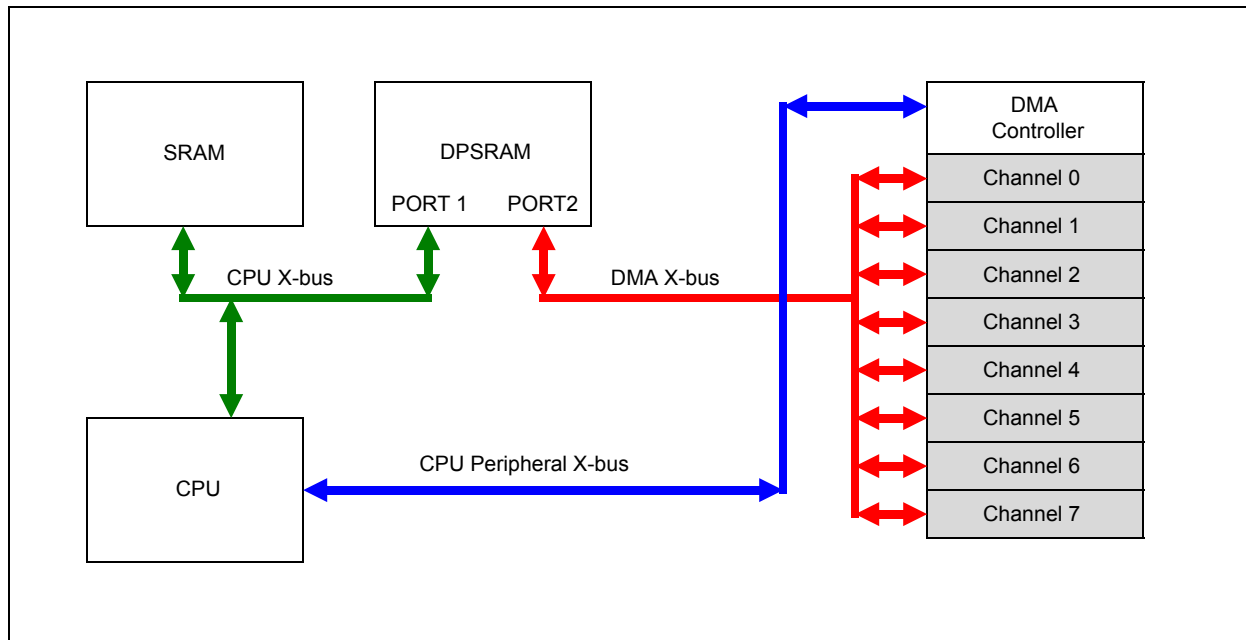
Every DMA channel offers the flexibility of byte/word transfer. The built-in priority scheme in the DMA module allows it to arbitrate when more than one request is received at the same time. Each DMA channel has the capability of moving a block of up to

1024 data words (or 2048 data bytes) before interrupting the CPU to indicate that the block is available for processing. The DMA request for each channel can be configured individually from any supported interrupt source. DMA supports the following modes of operation:

- Post-increment or Static DPSRAM Addressing
- Peripheral Indirect Addressing
- One-Shot or Continuous Block Transfer
- Ping-Pong mode
- Manual mode of operation

The ECAN peripheral in dsPIC33F and PIC24H devices is supported by the DMA controller. Each DMA channel is unidirectional, which requires at least two channels to be allocated for transmission and reception of messages from the CAN bus. The involvement of both ECAN and DMA modules along with code examples and some useful macros, will be discussed in subsequent sections.

FIGURE 5: DMA BLOCK DIAGRAM



ECAN MODULE CONFIGURATION

The ECAN module must be configured for sending and receiving messages on the CAN bus.

The configuration steps are application dependent. For the selection details, such as operating modes and Baud Rate, refer to **Section 21. “Enhanced Controller Area Network (ECAN™)”** (see “References”) and the specific device data sheet. The minimum configuration can be done as specified in the following steps:

- Step 1: Request Configuration Mode from the ECAN Module
- Step 2: Select ECAN Clock and Bit Timing
- Step 3: Assign Number of Buffers Used by ECAN Module in DMA Memory Space
- Step 4: Set Up Filters and Masks
- Step 5: Put the ECAN Module in Normal Mode
- Step 6: Set Up the Transmit/Receive Buffers

The sequence is not important in configuring the ECAN module as long as the module is in Configuration mode. However, the bit time control registers (CiCFG1 and CiCFG2), and the filter and mask registers can only be modified in Configuration mode. Refer to **Appendix A: “Flow Charts”**.

Step 1: Request Configuration Mode from the ECAN Module

The ECAN module must be in Configuration mode to access some of the configuration registers. The code in Example 1 requests Configuration mode and waits for confirmation.

EXAMPLE 1: REQUESTING CONFIGURATION MODE

```
C1CTRL1bits.REQOP=4;
while (C1CTRL1bits.OPMODE!=4);
```

Note: The ECAN module starts in Configuration mode at hardware reset. While in this mode, the ECAN registers have the Reset values and all error counters are cleared.

Step 2: Select ECAN Clock and Bit Timing

For a detailed description of each operating mode and system clock, refer to **Section 21. “Enhanced Controller Area Network (ECAN™)”** (see “References”) and the specific device data sheet.

The following system parameters are used as code defines to get the CAN bus timing:

- CAN clock = 40 MHz
- Bit rate = 1 Mbps
- Using 20 TQ in a bit
- The Baud Rate Prescaler (BRP) value (calculated using Equation 1):

EQUATION 1: BRP FORMULA

$$BRP = \frac{FCAN}{[2*(N)TQ*Bit\ Rate]-1}$$

EXAMPLE 2: CODE FOR BRP CONFIGURATION

```
/* CAN Baud Rate Configuration */
#define FCAN 40000000
#define BITRATE 1000000
#define NTQ 20 //20 time quanta in a bit time
#define BRP_VAL ((FCAN/(2*NTQ*BitRate))-1)
```

The system parameters for the clock and timing initialization code are defined, as shown in Example 3.

Note: FCAN cannot exceed 40 MHz.

EXAMPLE 3: CLOCK AND TIMING INITIALIZATION CODE

```
/* FCAN is selected to be FCY */
/* FCAN = FCY = 40 MHz */

C1CTRL1bits.CANCKS = 0x1;

/* Bit Time = (Sync Segment + Propagation Delay + Phase Segment 1 + Phase Segment 2) = 20 * TQ */
/* Phase Segment 1 = 8 TQ */
/* Phase Segment 2 = 6 TQ */
/* Propagation Delay = 5 TQ */
/* Sync Segment = 1 TQ */
/* CiCFG1<BRP> = (FCAN/(2 * N * FBAUD)) - 1 */
/* BIT RATE OF 1 Mbps */

C1CFG1bits.BRP = BRP_VAL;

/* Synchronization Jump Width set to 4 TQ */
C1CFG1bits.SJW = 0x3;

/* Phase Segment 1 time is 8 TQ */
C1CFG2bits.SEG1PH = 0x7;

/* Phase Segment 2 time is set to be programmable */
C1CFG2bits.SEG2PHTS = 0x1;

/* Phase Segment 2 time is 6 TQ */
C1CFG2bits.SEG2PH = 0x5;

/* Propagation Segment time is 5 TQ */
C1CFG2bits.PRSEG = 0x4;

/* Bus line is sampled three times at the sample point */
C1CFG2bits.SAM = 0x1;
```

Following are the requirements for selecting the ECAN clock and timing parameters:

- The total number of time quanta in a Nominal Bit Time (NBT) must be programmed between 8 TQ to 25 TQ
- NBT = Synchronization Segment (always 1 TQ) + Propagation Segment (1 TQ - 8 TQ) + Phase Segment 1 (1 TQ - 8 TQ) + Phase Segment 2 (1 TQ - 8 TQ) are user selectable
- Propagation Segment + Phase Segment 1 \geq Phase Segment 2
- Phase Segment 2 > Synchronization Jump Width (SJW)
- Sampling of the bit happens at the end of Phase Segment 1 and must take place at about 60-70% of the bit time. Therefore, it is recommended that Phase Segment 2 be selected at about 30%.
- Synchronization Jump Width (SJW) is used to compensate for the phase shifts between the oscillator frequencies of the different bus nodes. Each CAN controller must be able to synchronize in the hardware signal edge of the incoming signal regardless of the clock that is used. This is handled in the hardware.

- The number of time quanta must divide evenly into the FCAN clock. For example, using an FCAN of 40 MHz, 20 TQ and 10 TQ are good choices, whereas 16 TQ is not ($40 \div 20 = 2.0$, $40 \div 10 = 4.0$, etc., yields an integer result, whereas $40 \div 16 = 2.5$ yields a decimal result and cannot be used). As a general rule of thumb, always use the highest number of time quanta to provide the best bit timing.

Step 3: Assign Number of Buffers Used by ECAN Module in DMA Memory Space

The code in Example 4 assigns four buffers in DMA RAM.

EXAMPLE 4: ASSIGNING FOUR BUFFERS

```
C1FCTRLbits.DMABS=0b000;
```

At least four buffers have to be assigned to the ECAN module. The maximum number of buffers that can be accessed directly in DMA RAM is 16. All 32 buffers are only available in First-In-First-Out (FIFO) mode.

Step 4: Set Up Filters and Masks

The ECAN module can receive both the Standard and Extended messages from the CAN bus. For details on how the filters and masks operate in the Microchip ECAN module, refer to 21.7.1 “Message Reception and Acceptance Filtering” in Section 21. “Enhanced Controller Area Network (ECAN™)” (see “References”).

Apply the following parameters during the set up of filters and masks:

- There are 16 filters available on the ECAN module to implement message filtering
- Three mask registers that are available on the ECAN module to be used along with the filters
- ECAN SFR space is implemented using a memory window scheme. Certain Register access depends on the access window bit (CiCTRL1<WIN>). Some registers are visible regardless of the window select bit. For example, Address 0x0420 holds both C1BUPNT1 and C1RXFUL1, and access depends on the status of WIN bit.
- Ensure that the SFR Map Window Select (CiCTRL1<WIN>) bit is changed before modifying the filter and mask registers.

The code in Example 5 provides macros to facilitate message filtering.

EXAMPLE 5: MESSAGE FILTERING MACROS

```

/* Filter and mask defines */
/* Macros used to write filter/mask ID to Register CiRXMxSID and CiRFXxSID */
/* For example, to set up the filter to accept a value of 0x123, the macro when called as */
/* CAN_FILTERMASK2REG_SID(0x123) will write the register space to accept message with ID 0x123 */
/* Use for Standard Messages Only */
#define CAN_FILTERMASK2REG_SID(x) ((x & 0x07FF)<<5)

/* The Macro will set the MIDE bit in CiRXMxSID */
#define CAN_SETMIDE(sid) (sid|0x0008)

/* The Macro will set the EXIDE bit in CiRFXxSID to only accept extended messages */
#define CAN_FILTERXTD(sid) (sid|0x0008)

/* The macro will clear the EXIDE bit in CiRFXxSID to only accept standard messages */
#define CAN_FILTERSTD(sid) (sid & 0xFFFF7)

/* Macro used to write filter/mask ID to Register CiRXMxSID, CiRXMxEID, CiRFXxSID and CiRFXxEID */
/* For example, to set up the filter to accept a value of 0x123, the macro when called as */
/* CAN_FILTERMASK2REG_SID(0x123) will write the register space to accept message with ID 0x123 */
/* Use for Extended Messages only*/
#define CAN_FILTERMASK2REG_EID0(x) (x & 0xFFFF)
#define CAN_FILTERMASK2REG_EID1(x) (((x & 0x1FFC)<<3) | (x & 0x3))

```

AN1249

The code in Example 6 shows how to configure filter 0 to use mask 0 to only accept the standard message ID, 0x123. All of the mask bits are set to a logical '1' to enable a check on every bit of the standard message ID. The messages that are accepted by filter 0 are configured to be received in buffer 1.

EXAMPLE 6: CONFIGURING FILTER 0 TO USE MASK 0 TO ONLY ACCEPT STANDARD MESSAGE ID 0x123

```
/* Select acceptance mask 0 filter 0 buffer 1 */
C1FMSKSEL1bits.F0MSK = 0;

/* Configure acceptance mask - match the ID in filter 0 */
/* setup the mask to check every bit of the standard message, the macro when called as */
/* CAN_FILTERMASK2REG_SID(0x7FF) will write the register C1RXM0SID to include every bit in */
/* filter comparison */
C1RXM0SID=CAN_FILTERMASK2REG_SID(0x7FF);

/* Configure acceptance filter 0 */
/* Set up the filter to accept a standard ID of 0x123, the macro when called as */
/* CAN_FILTERMASK2REG_SID(0x123) will write the register C1RXF0SID to only accept standard */
/* ID of 0x123 */
C1RXF0SID=CAN_FILTERMASK2REG_SID(0x123);

/* Set filter to check for standard ID and accept standard ID only */
CAN_SETMIDE(C1RXM0SID);
CAN_FILTERSTD(C1RXF0SID);

/* Acceptance filter to use buffer 1 for incoming messages */
C1BUFENT1bits.F0BP = 0b0001;

/* Enable filter 0 */
C1FEN1bits.FLTENO = 1;
```

The code in Example 7 shows how to configure filter 1 to use mask 1 to accept the extended message ID, 0x1234568. All the mask bits are set to a logical '1' to enable a check on every bit of the extended message ID. The messages that are accepted by filter 1 are configured to be received in buffer 2.

EXAMPLE 7: CONFIGURING FILTER 1 TO USE MASK 1 TO ACCEPT EXTENDED MESSAGE ID 0x1234568

```

/* Select acceptance mask 1 filter 1 and buffer 2 */
C1FMSKSEL1bits.F1MSK = 0b01;

/* Configure acceptance mask1 */
/* Match ID in filter 1. Setup the mask to check every bit of the extended message, the macro */
/* when called as CAN_FILTERMASK2REG_EID0(0xFFFF) will write the register C1RXM1EID to include */
/* extended message ID bits EID0 to EID15 in filter comparison. */
/* The macro when called as CAN_FILTERMASK2REG_EID1(0x1FFF) will write the register C1RXM1SID */
/* to include extended message ID bits EID16 to EID28 in filter comparison.*/
C1RXM1EID=CAN_FILTERMASK2REG_EID0(0xFFFF);
C1RXM1SID=CAN_FILTERMASK2REG_EID1(0x1FFF);

/* Configure acceptance filter 1 */
/* Configure acceptance filter 1 - accept only XTD ID 0x12345678. Setup the filter to accept */
/* only extended message 0x12345678, the macro when called as CAN_FILTERMASK2REG_EID0(0x5678) */
/* will write the register C1RXF1EID to include extended message ID bits EID0 to EID15 when */
/* doing filter comparison. The macro when called as CAN_FILTERMASK2REG_EID1(0x1234) will write */
/*
/* The register C1RXF1SID to include extended message ID bits EID16 to EID28 when doing */
/* filter comparison. */
C1RXF1EID=CAN_FILTERMASK2REG_EID0(0x5678);
C1RXF1SID=CAN_FILTERMASK2REG_EID1(0x1234);

/* Filter to check for extended ID only */
CAN_SETMIDE(C1RXM1SID);
CAN_FILTERXTD(C1RXF1SID);

/* Acceptance filter to use buffer 2 for incoming messages */
C1BUFNT1bits.F1BP=0b0010;

/* Enable filter 1 */
C1FEN1bits.FLTEN1=1;

```

Note: A high-frequency message on the CAN bus can generate overflow errors on the module if there is only one filter enabled to receive that message ID.

This can be resolved by enabling multiple filters to accept the same message ID. If a new message arrives before the previous buffer is read, the next available filter accepts the message and uses an empty buffer.

Setting the mask bit to “don’t care” (logical ‘0’), will disable filtering on that specific bit of the message. This technique can be used to accept a range of messages in a filter.

Step 5: Put the ECAN Module in Normal Mode

The code in Example 8 puts the module in Normal mode of operation and waits until the mode is confirmed by the peripheral.

EXAMPLE 8: CODE REQUESTING NORMAL MODE OF OPERATION

```
C1CTRL1bits.REQOP = 0;
while(C1CTRL1bits.OPMODE! = 0);
```

Step 6: Set Up the Transmit/Receive Buffers

By default, the first seven buffers are configured as receive buffers. To configure a buffer as a Transmit buffer, set the TXENx bit in the control register of the buffer to high.

EXAMPLE 9: ENABLING BUFFERS AND SETTING TX BUFFER PRIORITY

```
C1TR01CONbits.TXEN0 = 1;           /* ECAN1, Buffer 0 is a Transmit Buffer */
C1TR01CONbits.TXEN1 = 0;           /* ECAN1, Buffer 1 is a Receive Buffer */
C1TR01CONbits.TXOPR1 = 0b11;      /* Message Buffer 0 Priority Level */
```

Note 1: CiTRmnCON, 'where 'i' = 1 or 2 depending on the device and refers to ECAN1 or ECAN2 module.

2: The control register is 16-bits wide and each physical register controls two buffers. Following are the possible combinations with this register:

- CiTR01CON - To access buffer 0 and buffer 1 control registers
- CiTR23CON - To access buffer 2 and buffer 3 control registers
- CiTR45CON - To access buffer 4 and buffer 5 control registers
- CiTR67CON - To access buffer 6 and buffer 7 control registers

The ECAN module provides a mechanism of prioritizing the messages within each node for pending transmittable messages. Prior to sending the Start-of-Frame (SOF), the priority of each buffer that is ready for transmission is compared, and the buffer with the highest priority is sent first. If two buffers have the same priority, the buffer with the highest address will be sent first.

For example, if transmit buffer 0 and 1 are both ready for transmission and transmit buffer 0 has highest priority, transmit buffer 0 will be transferred first.

Inversely, if buffer 0 and buffer 1 have the same priority, buffer 1 is sent first.

There are four levels of transmit priority. For more details, refer to **Section 21. "Enhanced Controller Area Network (ECAN™)"** (see "References").

Example 9 shows that buffer 0 and buffer 1 have the same level of priority: highest priority. This completes the necessary configuration of the ECAN module.

DMA MODULE CONFIGURATION

The DMA module must be configured to operate with the ECAN peripheral. Each DMA channel can be configured individually to interface between the peripheral and the DMA controller. Before studying the information related to DMA, the role of the compiler/assembler/linker in using the dsPIC33F and PIC24H devices must be known.

Role of Compiler/Assembler/Linker

The DMA controller must know the target address of every message that is received. The compiler makes this easy by providing built-in attributes. The compiler must know where to reserve space for the message buffers and how to access the reserved space when needed. If MPLAB® C Compiler for PIC24 MCUs and dsPIC DSCs is used, the code snippet shown in Example 10 will reserve and align space for the message buffers in DMA RAM. Refer to the DMA initialization flow chart in Figure A-1.

The declaration is a standard C declaration except for the inclusion of “space” and “align” attributes of MPLAB C30.

Normally, the compiler allocates variables in general data space. The “space” attribute is used to direct the compiler to allocate a variable in DMA memory space. Variables in DMA memory can be accessed using ordinary C statements.

The “alignment” attribute specifies a minimum alignment for the variable, measured in bytes and must be a power of two.

Using the two attributes “space” and “alignment”, the compiler is directed to reserve and align a continuous block of 64-bytes in DMA memory.

EXAMPLE 10: RESERVE AND ALIGN MESSAGE BUFFER SPACE IN DMA RAM

```
/* ECAN message buffer length */
#define ECAN1_MSG_BUF_LENGTH 4

typedef unsigned int ECAN1MSGBUF[ECAN1_MSG_BUF_LENGTH][8];
ECAN1MSGBUF ecan1msgBuf__attribute__((space(dma),aligned(ECAN1_MSG_BUF_LENGTH*16)));
```

Configuring DMA Module

This section configures DMA channel 0 for transmission and channel 2 for reception with the following settings that are relevant for the code provided with this application note:

- DMAxCON Register
- DMAxPAD Register
- DMAxCNT Register
- DMAxREQ Register
- DMAxSTA Register

Example 11 provides code for configuring each of these registers.

DMAxCON REGISTER

- Transfer size is configured to be two bytes (word)
- Read from DMA RAM and write to peripheral (transmission)
- Write to DMA RAM and read from peripheral (reception)
- Select DMA operating mode as Continuous, Ping-Pong mode disabled
- Select DMA channel addressing mode as Peripheral Indirect Addressing mode

Note: Byte mode is not supported when DMA is used with the ECAN peripheral.
--

DMAxPAD REGISTER

For a DMA transfer to operate correctly, the DMA channels must be associated with the ECAN peripheral. This information is supplied to the channel through the DMAxPAD register.

AN1249

DMAxCNT REGISTER

The value in the DMAxCNT register is independent of the transfer size selected in the DMAxCON register. The value used in this register determines when the buffer transfer is considered complete by the DMA controller. Each DMA channel must be configured to service requests before the data transfer is considered complete. Since the buffer size of the CAN message is eight words, the value 7 is used in this register that transfers or receives eight words of data to/from the peripheral.

Note: During receptions, the ECAN module always sends eight words to the DMA (regardless of the DLC value), which implies that the DMAxCNT register must be a multiple of eight for the RX channel.

DMAxREQ REGISTER

This register configures when the DMA transfer requests are serviced. The DMA channel requests can be triggered manually by setting the FORCE bit in the DMAxREQ register. For the ECAN module, the transfer request is automatically serviced by the DMA controller and the interrupt request is sent to the CPU, if enabled.

DMAxSTA REGISTER

The DMAxSTA register stores the offset from the beginning of the DMA memory, since the ECAN peripheral can be used only in Peripheral Indirect mode.

The built-in C30 attribute `__builtin_dmaoffset()` function can be used to find the correct offset that is used for calculating the addresses of message buffers.

EXAMPLE 11: CODE FOR CONFIGURING THE DMA REGISTERS

```
/* Initialize the DMA channel 0 for ECAN TX and clear the collision flags */
DMAC0 = 0;

/* Set up Channel 0 for peripheral indirect addressing mode normal operation, word operation */
/* and select TX to peripheral */
DMA0CON = 0x2020;

/* Set up the address of the peripheral ECAN1 (C1TXD) */
DMA0PAD = 0x0442;

/* Set the data block transfer size of 8 */
DMA0CNT = 7;

/* Automatic DMA TX initiation by DMA request */
DMA0REQ = 0x0046;

/* DPSRAM atart address offset value */
DMA0STA = __builtin_dmaoffset(&ecan1msgBuf);

/* Enable the channel */
DMA0CONbits.CHEN = 1;

/* Initialize DMA Channel 2 for ECAN RX and clear the collision flags */
DMAC2 = 0;

/* Set up Channel 2 for Peripheral Indirect addressing mode (normal operation, word operation */
/* and select as RX to peripheral */
DMA2CON = 0x0020;

/* Set up the address of the peripheral ECAN1 (C1RXD) */
DMA2PAD = 0x0440;

/* Set the data block transfer size of 8 */
DMA2CNT = 7;

/* Automatic DMA Rx initiation by DMA request */
DMA2REQ = 0x0022;

/* DPSRAM atart address offset value */
DMA2STA = __builtin_dmaoffset(&ecan1msgBuf);

/* Enable the channel */
DMA2CONbits.CHEN = 1;
```

ENABLING THE INTERRUPTS FOR DATA EXCHANGE

There are several alternatives to send and receive messages with/without using interrupts. For the complete list of available interrupts, refer to the specific device data sheet. Only interrupts that are relevant to operating the ECAN module in Normal mode during the exchange of data will be discussed.

Transmit Buffer Interrupt

The message buffers 0 to 7 that are configured for message transmission will set the transmit buffer flag (CiINTF<TBIF>) and the ECAN event flag (IFS2<CiIF>), and generate an ECAN transmit event interrupt (CiINTE<TBIE> and IEC2<CiIE>) if enabled, once the CAN message is transmitted successfully. To get the source of interrupt, the CiVEC<ICODE> flag can be checked. The TBIF and CiIF flags must be cleared in the Interrupt Service Routine (ISR). Figure 6 depicts the interrupts that are generated during a message transmission.

The code in Example 12 enables the transmit interrupts to generate an interrupt when a message transmission is complete. Once the TXREQ bit is set and the bus is available, the ECAN module will transmit the message without any CPU interference.

EXAMPLE 12: ENABLE TX INTERRUPT

```
/* Enable ECAN1 interrupt */
IEC2bits.CiIE = 1;

/* Enable transmit interrupt */
CiINTEbits.TBIE = 1;
```

Receive Buffer Interrupt

When a message is successfully received and loaded into one of the enabled receive buffers, the receive buffer flag (CiINTE<RBIF>) and ECAN event flag (IFS2<C1IF> or IFS3<C2IF>), are set and an interrupt (CiINTE<RBIE>) is generated. If enabled, the ICODE bits indicate the source of interrupt along with the RXFUL flag in CiRXFUL1 or CiRXFUL2. Both CiIF and RBIF flags must be cleared in the ISR. The RXFUL flag must be cleared after the message has been read from the buffer. Figure 7 depicts the level of interrupts that are generated by the ECAN module during message reception.

FIGURE 6: TX INTERRUPT SOURCE

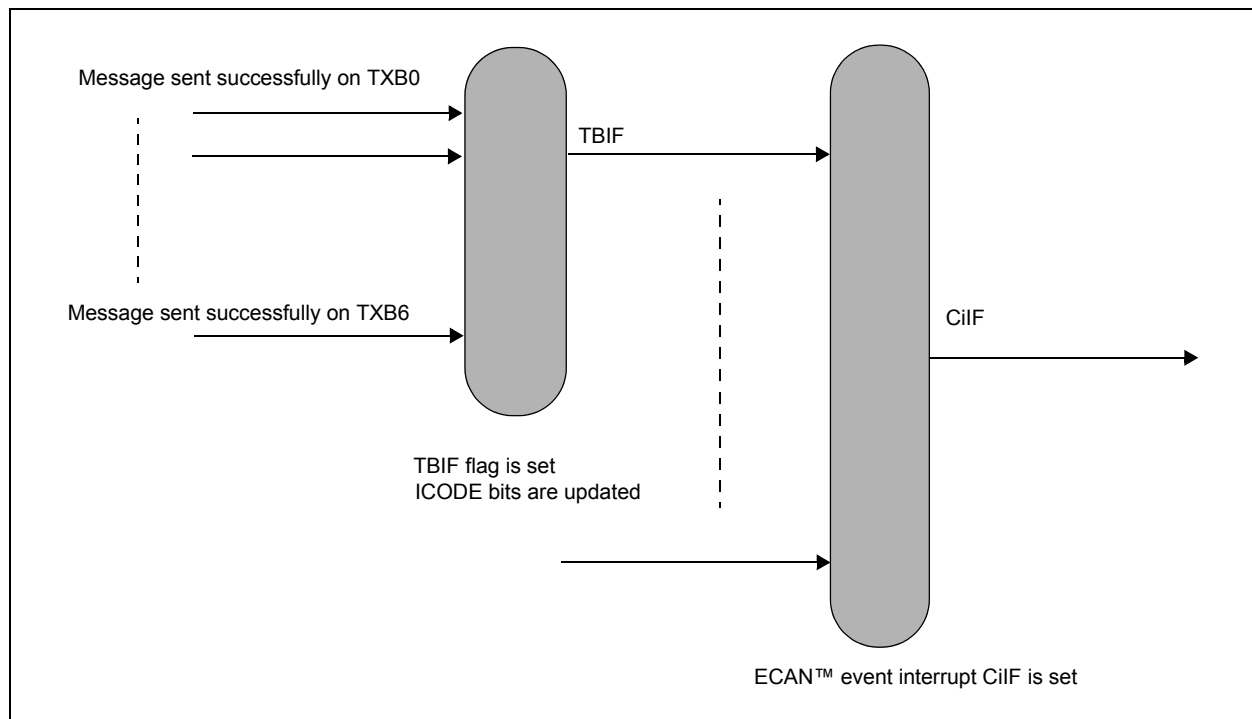
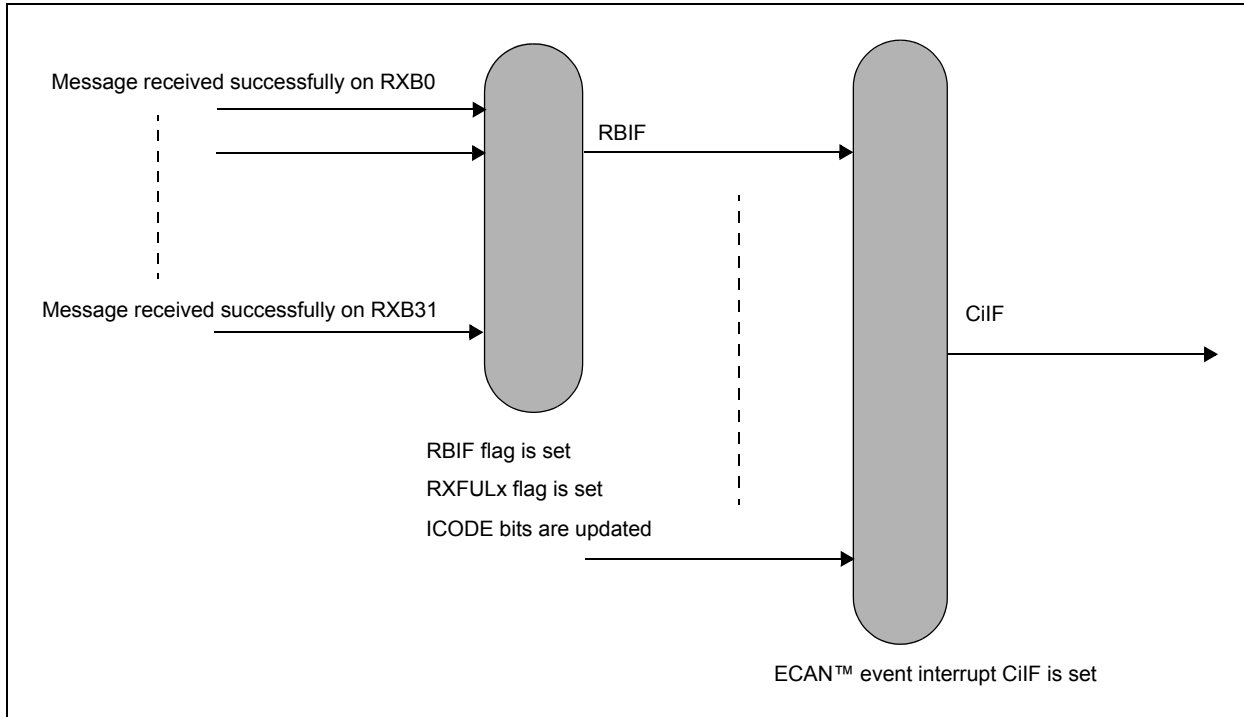


FIGURE 7: RX INTERRUPT SEQUENCE



The code in Example 13 enables the ECAN module to generate an interrupt when the message is received in the buffer. An interrupt is generated only when a message passes the filtering stage and an ECAN event interrupt is enabled.

EXAMPLE 13: ENABLE RX INTERRUPT

```
/* Enable ECAN1 receive Interrupt */
CIINTEbits.RBIE = 1;
```

In addition to the ECAN module interrupts, each DMA channel provides an interrupt when block transfer is complete. These interrupts can be used to monitor ECAN transmit and receive, if needed. However, the ECAN peripheral interrupts are selected for this document. Refer to **Section 21. “Enhanced Controller Area Network (ECAN™)”** (see “References”) and the specific device data sheet for more details on how to use the DMA channel interrupts.

Sending CAN Messages

Message buffers 0-7 can be used to transmit messages over the CAN bus. Each transmit buffer can be assigned a user-defined priority level using the CiTRmnCON<TXnPRI> bits. The data is loaded into Dual Port SRAM (DPSRAM) and the <TXREQm> bit in the CiTRmnCON register is set to request the transmission of the message. At this point, the message is “pending” for transmission and the module starts transmitting as soon as the CAN bus is available. Refer to **Section 21. “Enhanced Controller Area Network (ECAN™)”** (see “References”) and the specific device data sheet for more details on the registers used in this document.

WRITING TO THE DMA RAM FOR TRANSMISSION

In Example 14, the message is written in buffer 0 for transmission.

Note: Refer to **Appendix B: “CAN Message Structure and Define Statements”** for message structure and other definitions.

EXAMPLE 14: CODE FOR WRITING MESSAGE INTO DMA RAM

```

/* Message Format: */
/* Word0 : 0bUUUx xxxx xxxx xxxx */
/*          |_____|_|_| */
/*          SID10:0   SRR IDE (bit 0) */
/* Word1 : 0bUUUU xxxx xxxx xxxx */
/*          |_____| */
/*          EID17:6 */
/* Word2 : 0bxxxx xxx0 UUU0 xxxx */
/*          |_____| |_____| */
/*          EID5:0 RTR   DLC */

/* Remote Transmission Request Bit for standard frames */
/* SRR-> "0" Normal Message */
/*      "1" Message will request remote transmission */
/* Substitute Remote Request Bit for extended frames */
/* SRR-> should always be set to "1" as per CAN specification */

/* Extended Identifier Bit */
/* IDE-> "0" Message will transmit standard identifier */
/*      "1" Message will transmit extended identifier */

/* Remote Transmission Request Bit for extended frames */
/* RTR-> "0" Message transmitted is a normal message */
/*      "1" Message transmitted is a remote message */
/* Do not care for standard frames */

/* check to see if the message has an extended ID */
if(message->frame_type==CAN_FRAME_EXT)
{
    /* get the extended message id EID28..18*/
    word0=(message->id & 0x1FFC0000) >> 16;
    /* set the SRR and IDE bit */
    word0=word0+0x0003;
    /* the the value of EID17..6 */
    word1=(message->id & 0x0003FFC0) >> 6;
    /* get the value of EID5..0 for word 2 */
    word2=(message->id & 0x0000003F) << 10;
}
else
{
    /* get the SID */
    word0=((message->id & 0x000007FF) << 2);
}
/* check to see if the message is an RTR message */
if(message->message_type==CAN_MSG_RTR)
{
    if(message->frame_type==CAN_FRAME_EXT)
        word2=word2 | 0x0200;
    else
        word0=word0 | 0x0002;

    ecanlmsgBuf [message->buffer] [0]=word0;
    ecanlmsgBuf [message->buffer] [1]=word1;
    ecanlmsgBuf [message->buffer] [2]=word2;
}
else
{
    word2=word2+(message->data_length & 0x0F);
    ecanlmsgBuf [message->buffer] [0]=word0;
    ecanlmsgBuf [message->buffer] [1]=word1;
    ecanlmsgBuf [message->buffer] [2]=word2;
    /* fill the data */
    ecanlmsgBuf [message->buffer] [3]=((message->data[1] << 8) + message->data[0]);
    ecanlmsgBuf [message->buffer] [4]=((message->data[3] << 8) + message->data[2]);
    ecanlmsgBuf [message->buffer] [5]=((message->data[5] << 8) + message->data[4]);
    ecanlmsgBuf [message->buffer] [6]=((message->data[7] << 8) + message->data[6]);
}

```

AN1249

SETTING THE TXREQ BIT

Setting the TXREQ bit in the control register initiates the transmission. Setting the TXREQ bit does not guarantee that the transmission will be successful. In Example 15, buffer 0 is set for transmission.

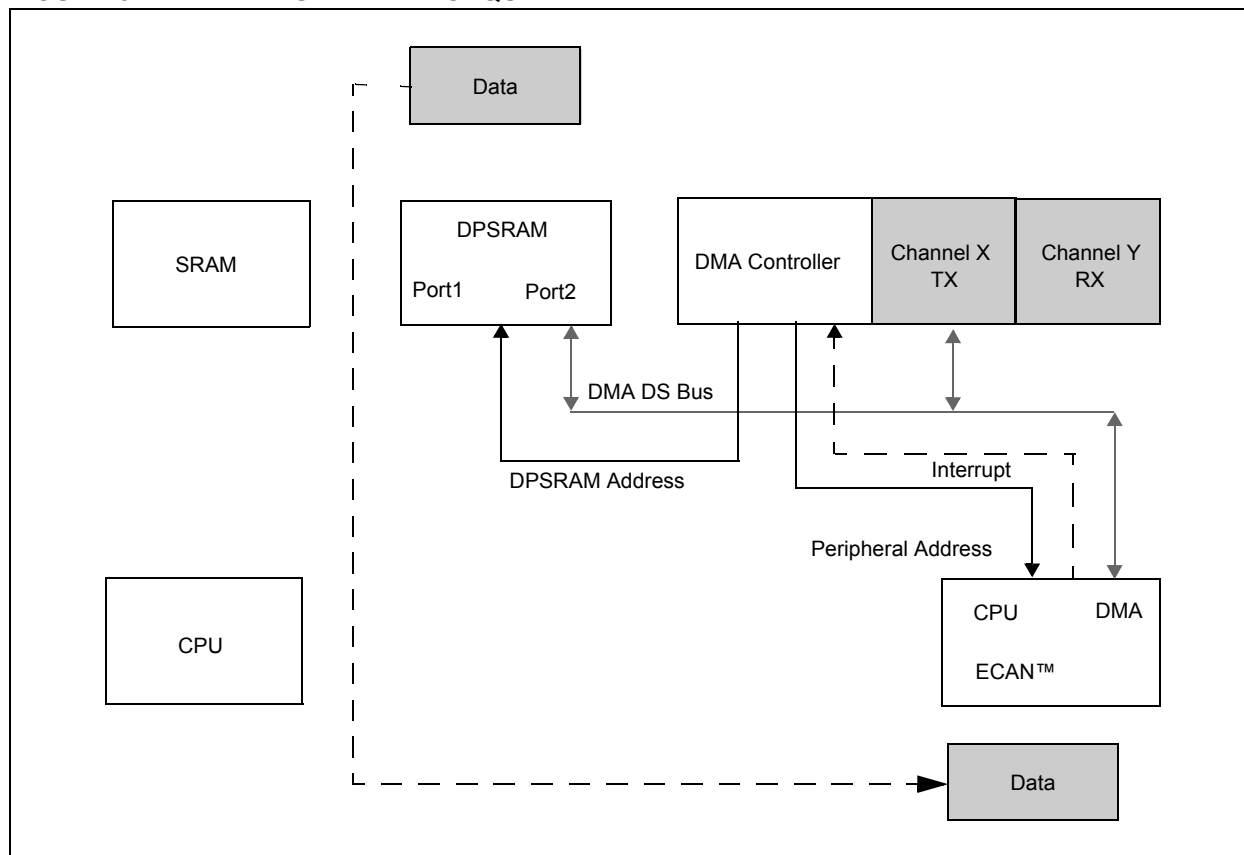
Note: Before writing to the buffer, it is a good idea to check the TXREQ bit to make sure there are no pending messages that have yet to be transmitted.

EXAMPLE 15: SET MESSAGE FOR TRANSMISSION

```
/* set the message for transmission */  
C1TR01CONbits.TXREQ=1;
```

When data is ready to be transmitted on the CAN bus, an interrupt is issued by the ECAN module as shown in Figure 8. The architecture of the dsPIC33F and PIC24H devices allow the user to choose between the CPU and DMA controller to service the interrupt. However, it is assumed that the user will exclusively configure either the CPU/DMA controller to service the request. The interrupt in Example 14 is serviced by the DMA controller. The DMA channel will read data from DMA RAM and then transfer the data to the peripheral. The DMA channel concurrently moves the data element and checks the block transfer counter. When the transfer counter reaches the user-defined limit, the block transfer is considered complete and a CPU interrupt and DMA interrupt is asserted to alert the modules.

FIGURE 8: TRANSMIT DATA SEQUENCE



Receiving CAN Message

The code in Example 16 receives the message in an ISR and clears all the flags once the message is read from the ECAN module.

EXAMPLE 16: SERVICING C1 ISR AND CLEARING INTERRUPT FLAGS

```

/* Interrupt Service Routine 1
/* No fast context save, and no variables stacked */
void __attribute__((interrupt, no_auto_psv)) _C1Interrupt(void)
{
    /* check to see if the interrupt is caused by receive */
    if(C1INTFbits.RBIF)
    {
        /*check to see if buffer 1 is full */
        if(C1RXFUL1bits.RXFUL1)
        {
            /* set the buffer full flag and the buffer received flag */
            canRxMessage.buffer_status=CAN_BUF_FULL;
            canRxMessage.buffer=1;
        }
        /* check to see if buffer 2 is full */
        else if(C1RXFUL1bits.RXFUL2)
        {
            /* set the buffer full flag and the buffer received flag */
            canRxMessage.buffer_status+CAN_BUF_FULL;
            canRxMessage.buffer=2;
        }
        /* check to see if buffer 3 is full */
        else if(C1RXFUL1bits.RXFUL3)
        {
            /* set the buffer full flag and the buffer received flag */
            canRxMessage.buffer_status=CAN_BUF_FULL;
            canRxMessage.buffer=3;
        }
        else;
        /* clear flag */
        C1INTFbits.RBIF = 0;
    }
    else if(C1INTFbits.TBIF)
    {
        puts_ecanTc(&canTxMessage);
        /* clear flag */
        C1INTFbits.TBIF = 0;
    }
    else;
    /* clear interrupt flag */
    IFS2bits.C1IF = 0;
}

```

The code for RxECAN (&canRxMessage); that reads the buffer into a temporary buffer in RAM, is shown in Example 17.

EXAMPLE 17: COPYING MESSAGE FROM DMA INTO SRAM

```
void rxECAN(mID * message)
{
    unsigned int ide=0;
    unsigned int rtr=0;
    unsigned long id=0;

    /* Standard Message Format: */
    /* Word0 : 0bUUUx xxxx xxxx xxxx */
    /*          |_____|_|_| */
    /*          SID10:0 SRR IDE (bit 0) */
    /* Word1 : 0bUUUU xxxx xxxx xxxx */
    /*          |_____| */
    /*          EID17:6 */
    /* Word2 : 0bxxxx xxx0 UUU0 xxxx */
    /*          |_____| |__| */
    /*          EID5:0 RTR  DLC */
    /* Word3-Word6: Data bytes */
    /* Word7: Filter hit code bits */

    /* Remote Transmission Request Bit for standard frames */
    /* SRR-> "0 " Normal Message */
    /*        "1" Message will request remote transmission */
    /* Substitute Remote Request Bit for extended frames */
    /* SRR-> Should always be set to '1' as per CAN specification */

    /* Extended Identifier Bit */
    /* IDE-> "0" Message will transmit standard identifier */
    /*        "1" Message will transmit extended identifier */

    /* Remote Transmission Request Bit for extended frames */
    /* RTR-> "0" Message transmitted is a normal message */
    /*        "1" Message transmitted is a remote message */
    /* Don't care for standard frames */
    /* read word 0 to see the message type */
    ide=ecanlmsgBuf[message->buffer][0] & 0x0001;

    /* check to see what type of message it is */
    /* message is standard identifier */
    if(ide==0)
    {
        message->id=(ecanlmsgBuf[message->buffer][0] & 0x1FFC) >> 2;
        message->frame_type=CAN_FRAME_STD;
        rtr=ecanlmsgBuf[message->buffer][0] & 0x0002;
    }
    /* message is extended identifier */
    else
    {
        id=ecanlmsgBuf[message->buffer][0] & 0x1FFC;
        message->id=id << 16;
        id=ecanlmsgBuf[message->buffer][1] & 0x0FFF;
        message->id=message->id+(id << 6);
        id=(ecanlmsgBuf[message->buffer][2] & 0xFC00) >> 10;
        message->id=message->id+id;
        message->frame_type=CAN_FRAME_EXT;
        rtr=ecanlmsgBuf[message->buffer][2] & 0x0200;
    }
    /* check to see what type of message it is */
    /* RTR message */
    if(rtr==1)
    {
        message->message_type=CAN_MSG_RTR;
    }
    ...
}
```


CONCLUSION

This application note provides an overview of the DMA and compiler role in implementing the ECAN module in your application. To get a thorough understanding of both modules, refer to the appropriate section of the dsPIC33F or PIC24H Family Reference Manual (see “References”).

This application note has been written to provide a working example for sending and receiving messages on the CAN bus, and applied to dsPIC33F and PIC24H devices that have an ECAN module.

REFERENCES

- **Section 21. “Enhanced Controller Area Network (ECAN™) Module”** (DS70226) of the *“PIC24H Family Reference Manual”*
- **Section 21. “Enhanced Controller Area Network (ECAN™) Module”** (DS70185) of the *“dsPIC33F Family Reference Manual”*
- **Section 22. “Direct Memory Access (DMA)”** (DS70223) of the *“PIC24H Family Reference Manual”*
- **Section 22. “Direct Memory Access (DMA)”** (DS70182) of the *“dsPIC33F Family Reference Manual”*
- Microchip Web Seminar: *“dsPIC33F and PIC24H Direct Memory Access (DMA) Module”*
- TB3008: *“PLL Jitter and its Effects on ECAN™ Technology Communications”* (DS93008)

These documents and the Web seminar can be downloaded from the Microchip Web site at: (www.microchip.com).

APPENDIX A: FLOW CHARTS

FIGURE A-1: ECAN™ INITIALIZATION FLOW CHART

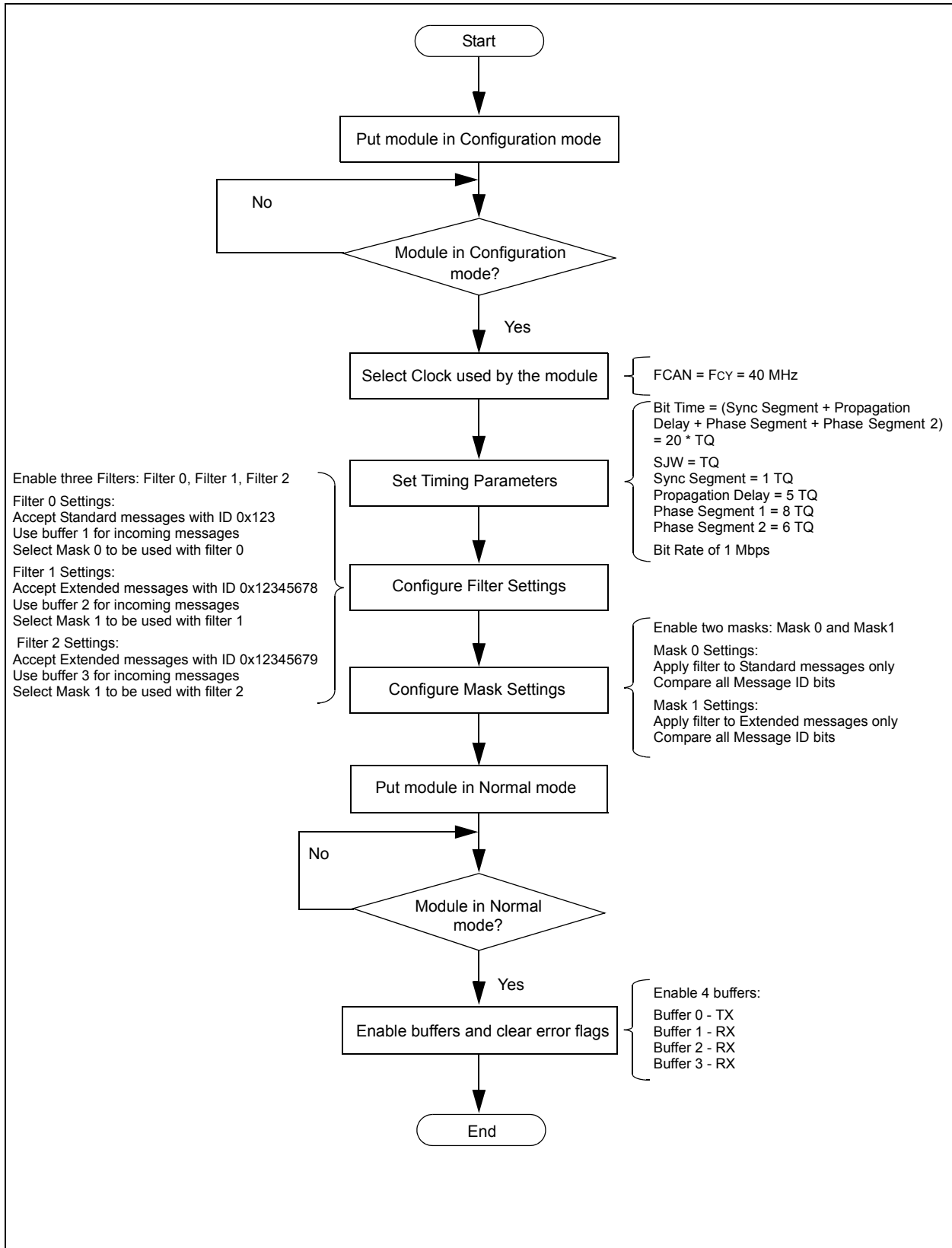


FIGURE A-2: DMA INITIALIZATION FLOW CHART

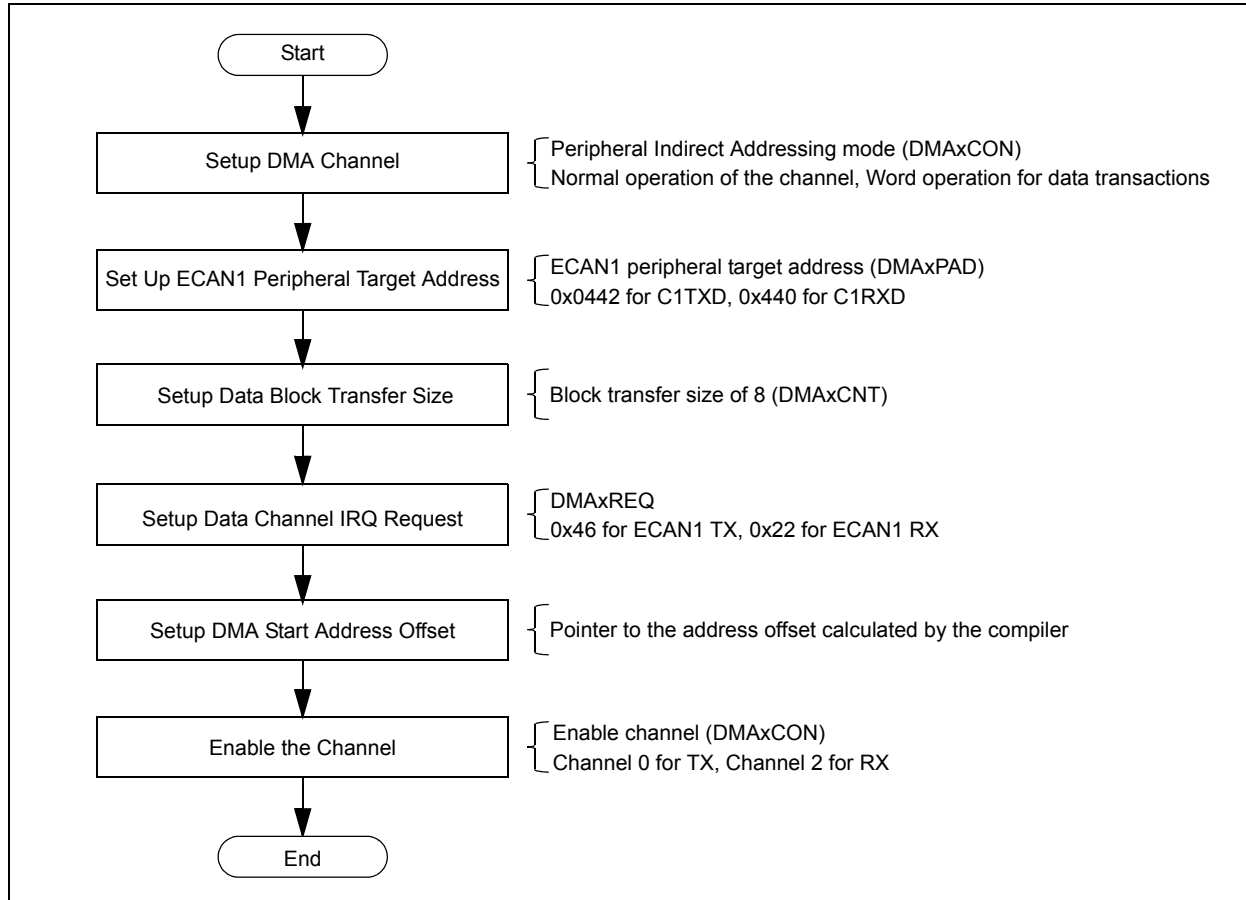
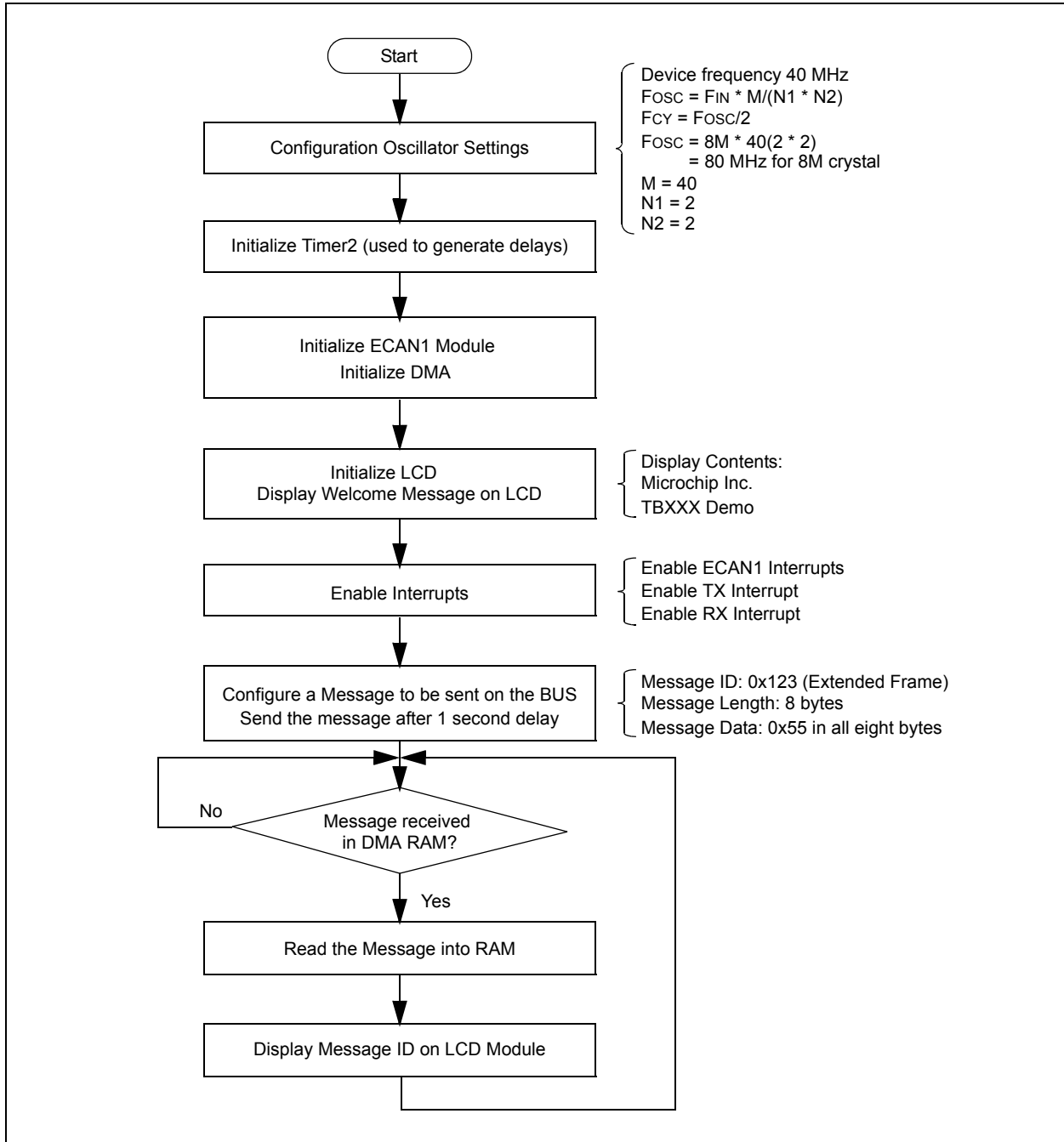


FIGURE A-3: MAIN APPLICATION FLOW CHART



APPENDIX B: CAN MESSAGE STRUCTURE AND DEFINE STATEMENTS

EXAMPLE B-1: CAN MESSAGE STRUCTURE

```
/* message structure in RAM */
typedef struct{

/* Keep track of the buffer status */
unsigned char buffer_status;

/* RTR message or data message */
unsigned char message_type;

/* Frame type extended or standard */
unsigned char frame_type;

/* Buffer being used to send and receive messages */
unsigned char buffer;

/* 29 bit ID max of 0x1FFF FFFF */
/* 11 bit ID max of 0x7FF */
unsigned long id;
unsigned char data[8];
unsigned char data_length;
}mID;
```

EXAMPLE B-2: CAN DEFINES USED IN THE APPLICATION CODE

```
/* ECAN message buffer length */
#define ECAN1_MSG_BUF_LENGTH 4

/* ECAN message type identifiers */
#define CAN_MSG_DATA 0x01
#define CAN_MSG_RTR 0x02
#define CAN_FRAME_EXT 0x03
#define CAN_FRAME_STD 0x04
#define CAN_BUF_FULL 0x05
#define CAN_BUF_EMPTY 0x06
```

APPENDIX C: SOURCE CODE

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the "Company") is intended and supplied to you, the Company's customer, for use solely and exclusively with products manufactured by the Company.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

All of the software covered in this application note is available as a single WinZip archive file. This archive can be downloaded from the Microchip corporate Web site at:

www.microchip.com

AN1249

NOTES:

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, rfPIC, SmartShunt and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


FilterLab, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, PICkit, PICDEM, PICDEM.net, PICtail, PIC³² logo, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rfLAB, Select Mode, Total Endurance, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2009, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

**QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
== ISO/TS 16949:2002 ==**

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC[®] MCUs and dsPIC[®] DSCs, KEELOQ[®] code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.



WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
<http://support.microchip.com>
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4080

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820