INTRODUCTION

This application note describes the design, development and implementation of a smart, low cost, stand-alone Controller Area Network (CAN) node. It combines the Microchip 8-pin PIC12C672 microcontroller with the Microchip 18-pin MCP2510 stand-alone CAN controller. This creates a fully autonomous CAN node, which supports both “time-based” and “event driven” message transmission.

The node is interrupt driven, capable of monitoring five external inputs (two analog and three digital) and automatically generating messages based upon their value, controlling two digital outputs, responding to message requests via the CAN network and generating a repeating, time-based message.

The system supports a maximum CAN bus speed of 125 kbits per second, with both standard or extended frames. The system is presented using standard frames. Some code changes would be required to implement extended frames.

This application note focuses on the design and development of the node from the system level. No discussion of the nature of the analog signals is presented. The digital inputs are, simply, switch contacts whose purpose is left for the reader to define. This application note concentrates on the unique requirements of implementing the CAN node functions using an I/O limited microcontroller and a stand-alone CAN protocol controller.

SYSTEM DESCRIPTION

Overview

Figure 1 shows the block diagram of the overall system. There are two functional blocks. The first is the Control Logic block. This function is performed by the PIC12C672 microcontroller. The PIC12C672 was chosen because of the low pin count and powerful feature set, which includes an internal oscillator, on-board, multi-channel, 8-bit analog-to-digital converter (ADC), multiple interrupt sources and low power sleep mode. The second is the CAN interface block. This block is comprised of the MCP2510 CAN controller and the MCP2551 transceiver. The MCP2510 provides a full CAN 2.0 implementation with message filtering, which relieves the host microcontroller from having to perform any CAN bus related overhead. This is a key feature given the limited available code space of the PIC12C672.

FIGURE 1: CAN NODE BLOCK DIAGRAM
Communication between the Control Logic block and the CAN interface block implements the MCP2510’s built-in support for the SPI™ protocol. The PIC12C672 does not have a hardware SPI interface, so the necessary functions are implemented in firmware.

Two external analog signals are tied directly to the analog input pins of the PIC12C672. An A/D conversion is performed automatically for analog channel 0 (AN0), based upon the internal timer setup. The value is automatically transmitted over the CAN bus once the conversion is completed.

The node also utilizes the MCP2510’s multiple filters to respond to four additional CAN Message Identifiers received via the CAN bus. The masks and filters are set to accept messages into receive buffer 1 only. The identifiers are interpreted as one of the following, depending upon which filter is matched:

- **Read Analog Channel 1**
  - Perform A/D conversion for analog channel 1 (AN1) and initiate transmission of the value back to the requesting node
- **Read Digital Inputs**
  - Read the value of the MCP2510 input pins and transmit the value back to the requesting node
- **Update Digital Output 1**
  - Write the received value to the MCP2510 digital output 1
- **Update Digital Output 2**
  - Write the received value to the MCP2510 digital output 2

Since only receive buffer 1 is used, the mask registers for receive buffer 0 must all be set to a ‘1’. This action should be followed by setting the filter bits to match an unused message identifier (typically all ‘0’ or all ‘1’) in order to take advantage of the greater number of filters associated with that receive buffer.

**Message Identifier Format**

As presented, the system requires that messages intended for the node have a standard identifier which has a value of 0x3F0 to 0x3F3, with each of the four filters configured to accept one of these messages. For messages the node returns, it uses the same identifier as the received message, with the exception that the ID3 bit is set to a ‘1’. Therefore, when the ‘Read Analog Channel 1’ message is received (ID=0x3F0), the node returns the data using a message ID of 0x3F8. The time-based message for the value of analog channel 0 is transmitted with an identifier of 0x3FE. If a system error is detected, the system error message uses the identifier 0x3FF. Table 1 summarizes the various transmit and receive message identifiers used. All transmitted messages use data byte 1 of the CAN message to hold the data to be sent.

<table>
<thead>
<tr>
<th>TABLE 1: MESSAGE IDENTIFIERS</th>
<th>ID</th>
<th>Tx/Rx</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3F0</td>
<td>Rx</td>
<td></td>
<td>Read Analog Channel 1</td>
</tr>
<tr>
<td>3F1</td>
<td>Rx</td>
<td></td>
<td>Read Digital Inputs</td>
</tr>
<tr>
<td>3F2</td>
<td>Rx</td>
<td></td>
<td>Change Digital Output 1</td>
</tr>
<tr>
<td>3F3</td>
<td>Rx</td>
<td></td>
<td>Change Digital Output 2</td>
</tr>
<tr>
<td>3F8</td>
<td>Tx</td>
<td></td>
<td>Analog Channel 1 value</td>
</tr>
<tr>
<td>3F9</td>
<td>Tx</td>
<td></td>
<td>Current values of digital inputs</td>
</tr>
<tr>
<td>3FA</td>
<td>Tx</td>
<td></td>
<td>3F2 Command Acknowledgement</td>
</tr>
<tr>
<td>3FB</td>
<td>Tx</td>
<td></td>
<td>3F3 Command Acknowledgement</td>
</tr>
<tr>
<td>3FE</td>
<td>Tx</td>
<td></td>
<td>Analog Channel 0 value</td>
</tr>
<tr>
<td>3FF</td>
<td>Tx</td>
<td></td>
<td>System Error</td>
</tr>
</tbody>
</table>

**HARDWARE DESCRIPTION**

**Design/Performance Considerations**

When designing a system, there are a number of design considerations/tradeoffslimitations that must be taken into account. Proper selection allows the system to achieve optimal performance from available resources and to determine if the desired performance can be achieved. The overall performance of the system is a function of several things:

- The system clock rate
- The throughput of the SPI bus
- Interrupt latency
- External interrupt request frequency

**System Clock**

The PIC12C672 has only six available I/O pins and all of these are used. Two for analog inputs and four (three SPI and one INT) to interface to the MCP2510. This requires the system to take advantage of the internal RC oscillator of the PIC12C672. The internal RC oscillator provides a 4 MHz system clock to the microcontroller, which translates to a 1 µs instruction cycle.

The instruction cycle time directly affects the achievable speed of the SPI bus. This, in turn, determines the interrupt latency time as SPI communication makes up the majority of the time required for the Interrupt Service Routing (ISR).

**SPI Bus**

The standard SPI interface has been modified in this application to use a common signal line for both Serial In (SI) and Serial Out (SO) lines, isolated from each other via a resistor. This method requires only three I/O pins to implement the SPI interface, instead of the usual four. Using this configuration does not support the full duplex mode of SPI communications, which is not an issue in this application.
The system achieves an overall SPI bus rate of slightly more than 80 kbps, with the raw SPI clock rate averaging 95 kbps. The clock low time is a fixed 5 µs, and the clock high time is either 5 µs or 6 µs, depending upon whether a '0' or a '1' is being sent/received, which gives a worst case (sending the value 0xFF) of 90.9 kbps raw clock rate. The overall effective speed achieved includes the additional software overhead of 'bit-banging' the SPI protocol.

Interrupts
There are two interrupt sources in the system. The first is the PIC12C672 Timer0 interrupt, which occurs every 10.16 ms. The second interrupt source is the INT pin of the PIC12C672 and is connected to the INT output of the MCP2510. This interrupt occurs anytime a valid message is received, or if the MCP2510 detects a CAN bus related error. This external interrupt is completely asynchronous with respect to the rest of the system.

Interrupt Latency
It is necessary to carefully consider the interrupt latency requirements during the system design/development phase. This system has two interrupt sources: the internal timer interrupt, which occurs approximately every 10 ms, and the external INT pin interrupt, which is generated by the MCP2510 CAN controller, and may occur anytime. The latency time for the Timer ISR is essentially fixed. This parameter is a function of the time it takes for the ADC to perform a conversion on both channels, write the values to the transmit buffer and issue a Request-to-Send (RTS) command to the MCP2510 via the SPI interface. This takes approximately 428 µs to complete.

Digital Inputs and outputs
The MCP2510 has three pins that can be configured as general purpose inputs, and two pins that can be configured as digital outputs. Both are implemented in this design. These are discussed only at the simplest level within the scope of this application note. The inputs shown are connected to switch contacts, with the outputs being connected to LED indicators. The MCP2510 inputs have internal pull-up resistors. They will read high when the attached switch is open, and low when it is closed.

The MCP2510 has two I/O pins (RX0BF and RX1BF) that can be configured as general purpose outputs. These pins are configured as outputs and are connected to LED’s to function as some type of indicator lights, controlled via the CAN bus.

CAN Bus
The CAN bus is configured to run at 125 kbps. The clock source for the MCP2510 is a standard 8 MHz crystal connected to the OSC1 and OSC2 inputs. The CAN physical layer has been implemented using an industry standard CAN transceiver chip (e.g., Microchip MCP2551). This device supports CAN bus rates of up to 1 Mbps and is more than adequate for the application presented here.

FIRMWARE DESCRIPTION
The firmware is written in PICmicro® microcontroller (MCU) assembly code. The relative simplicity and small size of this application makes assembly language a more than suitable choice.

Figure 2 shows the top level flowchart for the overall system operation. The PICmicro MCU, after going through self initialization and initializing the MCP2510, goes to sleep and waits for an interrupt to occur. The following sections provide more detailed discussion of the operation of each of the major blocks in the firmware.
FIGURE 2: NODE OPERATION

System POR

Initialize PICmicro® MCU and MCP2510

Sleep

Interrupt Occurred?

Yes

Timer Interrupt?

Yes

Perform A/D conversion on AN0

Write A/D value to MCP2510

Transmit Buffer

Send Request to Send Command to MCP2510

Clear Interrupt flags in PIC12C672

No

Read MCP2510 Interrupt Flags

Error Interrupt?

Yes

Error Handler Routine

No

Read MCP2510 Rx Filters

Filter Match?

Yes

Process Request

No

SysErr(InvMsg)

Clear Interrupt flags in PIC12C672 and MCP2510
PICmicro MCU Initialization

Initialization of the PIC12C672 is straightforward. There are three major functions that need to be properly configured within the PIC12C672:

- General Purpose I/O pins (GPIO)
- Timer0 module
- A/D converter module

Additionally, the configuration word must also be programmed to enable/disable code protection and select the oscillator type.

GENERAL PURPOSE I/O PINS

The GPIO pins are the six I/O pins that are used to interface the PIC12C672 to the MCP2510 and sample the analog signals. The PICmicro MCU OPTION, TRIS and INTCON registers are used to control the setup of the GPIO pins. In this case, the OPTION register is programmed to disable the internal pull-up resistors on GP0/GP1/GP3, and configure GP2 to generate an interrupt on the negative going edge (to match the MCP2510’s active low INT output). The TRIS register, which controls whether each I/O pin is configured as an input or an output, is configured to set GP0/GP1/GP3 as inputs, and GP2 and GP5 as outputs. With the exception of GP4, all of the GPIO pins will remain in their initially configured state. GP4 will be changed between input and output mode, depending on whether an SPI read or write operation is being performed by the PIC12C672. The final step of configuring the port pins is to program the INTCON register to enable the interrupt-on-change feature for GP2. This allows the MCP2510 to generate an interrupt for the PIC12C672.

TIMER0 MODULE

The Timer0 module operation is controlled by the OPTION register and the TMR0 register. The OPTION register contains the control bits for the Timer0 prescaler, which is set to divide by 256. The TMR0 register is the counting register for Timer0 and generates an interrupt when it rolls over from 0xFF to 0x00. This register must be reloaded as part of the ISR in order to correctly control the time period between Timer0 interrupts. The target time period between Timer0 messages is 10 ms. In order to approach that target, it is necessary to determine the amount of time required to complete the Timer0 ISR, since the time between messages will be the sum of the Timer0 value and the ISR execution time. The A/D conversion takes approximately 19 µs. The SPI communication to write the A/D result to the MCP2510 transmit buffer and then send the RTS command requires approximately 409 µs to complete. This implies a total of approximately 428 µs for the ISR execution gives a total time between messages of 10.156 ms, which is within 2% of the target.

ANALOG-TO-DIGITAL CONVERTER MODULE

The Timer0 module is configured to use the Fosc/8 option for the conversion clock, which gives a TAD value of 2 µs, and an overall conversion time of 19 µs (TAD * 9.5). This is more than adequate when compared to the amount of time spent on SPI communications during the ISR.

MCP2510 Initialization

Before the system can communicate on the CAN bus, the MCP2510 must be properly configured. Configuration of the MCP2510 is accomplished by loading the various control registers with the desired values. The firmware is written to take advantage of the table read functionality of the PICmicro MCU. The values for each register are stored at the top of the PICmicro ROM memory. During the MCP2510 initialization, the values are sequentially read by the table read function and then written to the MCP2510 via the SPI interface.

CAN BUS TIMING

The CAN bit rate configuration is controlled by the CNF1, CNF2 and CNF3 registers. The details behind determining what the “best” configuration of these registers, for a given CAN bus system, is beyond the scope of this application note. The MCP2510 is configured to operate at a CAN bus rate of 125 kbps, using the following parameters:

- 8 MHz oscillator
- Baud rate prescaler equivalent to divide by four
- 8 TQ per bit time
- Sync segment: 1 TQ
- Prop segment: 1 TQ
- Phase Segment 1: 3 TQ
- Phase Segment 2: 3 TQ
- Sync Jump Width: 1 TQ

Refer to the MCP2510 data sheet (DS21291) for more detailed information regarding the setting of these parameters.

In order to make use of the MCP2510’s general purpose input and output pins, it is necessary to configure the TXRTSCTRL and BFPCTRL registers, respectively.

TXRTSCTRL

To enable the use of the TXnRTS pins as general purpose inputs, the mode control bit <BnRTSM> is cleared. This register also holds the current state of each of the inputs pins in bits 3:5, which can be read by the microcontroller at any time via the SPI interface.
BFPCTRL

To use the RXnBF pins of the MCP2510 as output pins it is necessary to functionally enable the pin by setting the BnBFE bits and then selecting the general purpose output mode of operation by clearing the BnBFM bits. Once the register has been configured, it is used to control the state of the output pins by toggling the BnBFS bits, which is accomplished via the MCP2510’s built-in Bit Modify Command, which only allows the desired bit to be modified.

CANINTE

The MCP2510’s CANINTE register controls the individual interrupt source enables. For this application only, the error interrupt (ERRIE) and the receive buffer 1 interrupts (RX1IE) are enabled. In this configuration, the MCP2510 will generate an interrupt when a valid message is accepted into the receive buffer, or when any of the various error conditions in the EFLG register occur.

Interrupt Service Routine

Once an interrupt occurs, the PICmicro MCU begins executing the ISR routine. Figure 3 shows the flow chart for the ISR. The ISR first determines the source of the interrupt (Timer0 or external INT pin) then branches to the appropriate code to process the interrupt. Figure 4 and Figure 5 show the program flow for the Timer0 and CAN message received interrupts, respectively.

TIMER0 INTERRUPT

When the Timer0 interrupt occurs (see Figure 4), the PICmicro MCU initiates an A/D conversion on AN0, constantly polling the ADDONE bit until the conversion is complete. Once the conversion is completed, the ADRES value is loaded into the MCP2510 transmit buffer 0, data byte 0 and an RTS command is issued for buffer 0. The TMR0 register is then reloaded and the interrupt flag is cleared. The interrupts are re-enabled by the execution of the RETIE command at the end of the ISR.

FIGURE 3: INTERRUPT SERVICE ROUTINE (ISR) FLOWCHART
MESSAGE RECEIVED INTERRUPT

When an interrupt is generated by the MCP2510, the PIC12C672 reads the CANINTF register of the MCP2510 to determine the source of the interrupt. If a valid message has been received, the MsgRcvd subroutine is executed (see Figure 5). If an error has occurred, the error handling subroutine is executed (see Figure 6).

When a valid message is received, the FILHIT<2:0> bits of the RXB1CTRL register are read to determine which message has been received.

If the match occurred on Filter 2, the PICmicro MCU initiates an A/D conversion on AN1, waits for the conversion to complete, loads the ADRES register value into the MCP2510 transmit buffer 0, data byte 0 and sends the RTS command.

If the match occurred on Filter 3, the PICmicro MCU reads the TXRTSCTRL register for the value of the three input pins, loads this value into the MCP2510 transmit buffer and sends the RTS command.

A match on Filter 4 or Filter 5 causes the PICmicro MCU to read the first byte of the received message and write it to the appropriate output pin via the MCP2510 BFPCTRL register.
FIGURE 5: CAN MSG RECEIVED FLOW

INT pin (CAN Msg Rx)

Read MCP2510 Rx Filter

Filter Match = 2?

Yes

Perform A/D conversion on AN1

No

Filter Match = 3?

Yes

Read value of three MCP2510 digital inputs

No

Filter Match = 4?

Yes

Sent Request to Send command to MCP2510

No

Update MCP2510 Digital Output Control Register

Read MCP2510 Rx Buffer for digital output 1 data

Filter Match = 5?

Yes

Sent Request to Send command to MCP2510

No

SysErr(InvMsg)

Clear Interrupt flags in PIC12C672 and MCP2510

Exit ISR

Read MCP2510 Rx Buffer for digital output 1 data

Read MCP2510 Rx Buffer for digital output 2 data

Write A/D value to MCP2510 Transmit Buffer
FIGURE 6: ERROR HANDLER FLOW

Error Handler

Read ERRIF Register

Rx B1 Overflow?

Yes

Send Error Msg

0x3FF, error code

0x11

No

Yes

Rx Warning Flag Set?

Send Error Msg

0x3FF, error code

0x01

No

Rx Error Passive Flag Set?

Yes

Send Error Msg

0x3FF, error code

0x02

No

Tx Error Passive Flag Set?

Yes

Send Error Msg

0x3FF, error code

0x03

No

Bus Off Flag Set?

Yes

1st Bus Off Occurrence?

No

Send Error Msg

0x3FF, error code

0x04

Yes

Set Bus Off Flag and re initialize MCP2510

Send Error Msg

0x3FF, error code

0x12

Idle PICmicro®

MCU must RESET system node

Message transmitted successfully

Yes

Done

No
Error Handling
The system also provides for error detection for a number of different types of errors that may occur. This includes CAN bus errors detected by the MCP2510 and invalid system state errors (see Figure 6). When any of these errors are detected, the system transmits a message with the ID of 0x3FF. This message contains one data byte, which is a code used to represent the type of error that occurred. Refer to Appendix B for a listing of various errors and the associated code. The one exception to this is the Bus-Off condition that the MCP2510 may enter if a large number of transmit errors are detected. If the Bus-Off condition is detected, the PICmicro MCU performs a re-initialization of the MCP2510 and then attempts to transmit the error message (ID=0x3FF) with an error code of 0x12. After initiating a request to send for the error message, the PICmicro MCU checks to ensure that the message was transmitted successfully. If it was successfully transmitted, the PICmicro MCU sets an internal flag to indicate that a Bus-Off condition has occurred, then resumes normal operation. If the error message fails to transmit correctly, or if the Bus-Off condition is detected a second time, the PICmicro MCU automatically enters an idle loop and remains there until a system reset occurs via power on.

SUMMARY
This application note demonstrates that a smart CAN node can be implemented with low cost, low pin count devices, such as the PIC12C672 microcontroller and MCP2510 Stand-Alone CAN controller, providing a very flexible and effective solution for a variety of applications.

REFERENCE DOCUMENTS
For additional information, the reader is directed to the following documents:
- PIC12C67X Data Sheet, DS30561; Microchip Technology, Inc.
- MCP2510 Stand-Alone CAN Controller Data Sheet, DS21291; Microchip Technology, Inc.
- Controller Area Network (CAN) Basics, AN713; DS00713; Microchip Technology, Inc.
- MCP2551 High-Speed CAN Transceiver Data Sheet, DS21667; Microchip Technology, Inc.
Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the “Company”) for its PICmicro® Microcontroller is intended and supplied to you, the Company’s customer, for use solely and exclusively on Microchip PICmicro Microcontroller products. 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.

APPENDIX B: SOURCE CODE

; ***********************************************
; * Spincan.asm                                 *
; * Revision 1.0   September 2000               *
; * Developed by Rick Stoneking                 *
; * Developed using MPLAB V4.12 and MPASM V2.3  *
; *                                              *
; * This code demonstrates how a very low cost  *
; * CAN node can be implemented using a         *
; * Microchip PIC12C672 8-pin microcontroller   *
; * and a Microchip MCP2510 Stand-Alone CAN    *
; * controller.                                *
; *                                              *
; ***********************************************

; ***********************************************
; * Setup the MPASM assembler options           *
; ***********************************************

LIST    p=12C672

; ***********************************************
; * Include the standard PIC12C672 include file *
; * and the custom MCP2510 support files        *
; ***********************************************

#include <p12c672.inc>
#include "mcp2510.inc"

; ***********************************************
; * Setup the PIC12C672 configuration Word      *
; ***********************************************

__CONFIG   _CP_OFF & _WDT_OFF & _MCLRE_OFF & _INTRC_OSC

; ***********************************************
; * Constants definitions                       *
; ***********************************************

TMR_COUNT   EQU     0xD9              ; Timer Reload value: 0xD9 = 38 * 256 * 1us = 9.728ms

; ***********************************************
; * Variable definitions                        *
; ***********************************************
temp        EQU     0x20
temp1       EQU     0x21yte_cnt    EQU     0x22
addr       EQU     0x23
tmp_data   EQU     0x24

; ****************************************************
; * PIC Initialization                            *
; *****************************************************
org     0x00
goto    start ; Jump around ISR vector

; ****************************************************
; * Interrupt service vector initialization      *
; *****************************************************
org     0x04
goto    isr     ; Point ISR vector to the ISR handler

; *****************************************************
; * Start of Main Code                             *
; *****************************************************
start
bsf     STATUS,RP0 ; select bank1
movlw   0x87      ; Disable internal pullups
movw    0x87      ; Interrupt on negative going edge on GP2
movwf   OPTION_REG ; Prescaler = 1:256
movlw   0x0B      ; --001011
movwf   TRISIO    ; set all ports output except GP3/1/0
bsf     INTCON,GPIE ; enable GPIO Interrupt on change
movlw   0x04      ; GP4&2= DIO, GP0&1= ADC, Vref=VDD
movwf   ADCON1
movlw   0x04      ; GPIE set - interrupt on pin change
movlw   0x04      ; GIE cleared - global interrupts disabled
bcf     STATUS,RP0 ; select bank0

; Initialize the A/D converter
movlw   0x40      ; AN0 conversion clock = Fosc/8 (TAD=2us)
movwf   ADCON0    ; Turn off A/D module

; Initialize Timer0
movlw   TMR_COUNT ; Initialize Timer0
movwf   TMR0      ; Timer0 interrupt every 9.728mS

; Set up initial conditions for the SPI
movlw   0x24      ; CS high, INT high, data/clk low
movwf   GPIO      ; write to port
bsf     GPIO,cs_pin ; set CS pin high
bcf     GPIO,sck_pin ; clear the sck pin
bcf     GPIO,so_pin ; clear the sdo pin
call    mcp2510_init ; initialize the mcp2510
; ****************************************************
; * Main wait loop                                    *
; *****************************************************

wait; wait for interrupt to occur
sleep; sleep while not processing a message
nop; NOP executed when waking up from sleep
nop; NOP executed after ISR completes
goto wait; go back to sleep and wait

; *****************************************************
; * MCP2510 Initialization                            *
; *****************************************************

mcp2510_init
movlw CAN_WRITE; write command
bcf GPIO,cs_pin; lower CS to enable MCP2510
call spi_send; send command
movlw CANCTRL; select CANCTRL register address
call spi_send; and send it
movlw REQOP_CONFIG; Request Config Mode
call spi_send; send data
bsf GPIO,cs_pin; raise CS to terminate operation
cbf GPIO,sck_pin; set clock and
cbf GPIO,sdo_pin; data pins low

movlw 0x71; number of addresses to be written
movwf byte_cnt; load into byte counter
movlw CAN_WRITE; write command
bcf GPIO,cs_pin; enable MCP2510
call spi_send; send command
movlw 0x00; start writing at address 0x00
call spi_send; send address
movlw 0x01
movwf addr

seq_wr; sequential write loop
movlw HIGH reg_init_tbl; get high byte of reg_int_tbl address
movwf PCLATH; load into high byte of PC counter
movfw addr; write into jump table pointer (addr)
decf addr, 1;
movf addr, W;
call reg_init_tbl; fetch byte to be written
call spi_send; send it to MCP2510
incf addr,1; increment the jump table pointer
incf addr,1; twice to point to the next byte
decfsz byte_cnt,1; decrement the byte counter and test for zero
goto seq_wr; not done so repeat
bsf GPIO,cs_pin; raise CS to terminate operation

movlw CAN_WRITE; write command
bcf GPIO,cs_pin; enable MCP2510
call spi_send
movlw CANCTRL; write to CANCTRL register
call spi_send
movlw REQOP_NORMAL; Normal Mode
call spi_send
bsf GPIO,cs_pin; terminate operation

movlw 0x00; clear byte_cnt variable
movwf byte_cnt
bsf INTCON,GIE; Enable Interrupts

return
**Interrupt Service Routine**

The ISR determines whether a TMR0 interrupt or an external INT pin interrupt occurs and then proceeds accordingly.

```assembly
isr
    bcf STATUS,RP1 ; select bank 0/1
    btfss INTCON,T0IE ; Timer0 interrupt?
goto intpin ; No, so jump to external interrupt pin ISR

    movlw TMR_COUNT ; reload
    movwf TMR0 ; Timer0

    bcf ADCON0,CHS0 ; select ADC channel 0
    call adc_cnv ; go do the conversion

    bcf GPIO,cs_pin ; enable MCP2510
    movlw CAN_WRITE ; send write command to MCP2510
    call spi_send ;

    movlw TXB0D0 ; set write address to TXB0D0
    call spi_send ;

    movfw ADRES ; write ADC conversion result
    call spi_send ;
    bsf GPIO,cs_pin ; terminate SPI operation

    bcf GPIO,cs_pin ; enable MCP2510
    movlw CAN_RTS_TXB0 ; Send RTS command for TXB0
    call spi_send ;
    bsf GPIO,cs_pin ; terminate operation

    bcf INTCON, T0IF ; clear TMR0 interrupt flag
    return ; exit isr

intpin
    movlw CAN_READ ; Message received interrupt
    bcf GPIO,cs_pin ; lower CS line
    call spi_send ; send read command to MCP2510

    movlw CANINTF ; Check for RXB1IF flag by reading
    call spi_send ; the interrupt flag register (CANINTF)
    call spi_rx ; read the data from the MCP2510
    bsf GPIO,cs_pin ; terminate SPI read

    movwf tmp_data ; save CANINTF value

    btfsb tmp_data,1 ; test CANINTF for RXIF
    call msg_rcvd ; if RXIF set go process message

    btfsb tmp_data,5 ; test CANINTF for ERRIF
    call can_err ; if ERRIF set go process CAN error

    movlw B'11011101' ; mask off RXB1IF and ERRIF bits
    andwf tmp_data,1 ; of CANINTF

    btfsb STATUS,2 ; if any bit set process invalid interrupt
    call sys_err ; Not an error interrupt so initiate an invalid interrupt

    bcf INTCON,GPIF ; reset interrupt flag
    retfie ; return to main routine
```
; *************************************************************************
; * CAN Error routine                                               *
; * This routine reads the value of the MCP2510 Error flag (EFLG)   *
; * register, writes it to byte 0 of TXB1, and then transmits the   *
; * TXB1 message                                                    *
; *************************************************************************
can_err

    movlw   CAN_READ              ; SPI Read operation
    bcf     GPIO, cs_pin           ; enable MCP2510
    call    spi_send              ;
    movlw   EFLG                  ; EFLG register to be read
    call    spi_send              ;
    call    spi_rx                ; read the data
    bsf     GPIO, cs_pin           ; terminate SPI operation
    movwf   tmp_data              ; save the value of EFLG register

    movlw   CAN_WRITE             ; now write to MCP2510
    bcf     GPIO, cs_pin           ;
    movlw   TXB1D0                ; write to data byte 0 of TXB1
    call    spi_send              ;
    movwf   tmp_data              ; write EFLG register contents
    call    spi_send              ;
    bsf     GPIO, cs_pin           ; terminate SPI operation

    movlw   CAN_RTS_TXB1          ; send request to send
    bcf     GPIO, cs_pin           ; for transmit buffer 1
    call    spi_send              ;
    bsf     GPIO, cs_pin           ; exit isr and re-enable interrupts
    retfie

; *************************************************************************
; * System Error Handler Routine                                    *
; * This routines transmits the TXB2 message to indicate that a       *
; * unidentifiable system error has occurred.                        *
; *************************************************************************
sys_err

    movlw   CAN_RTS_TXB2          ; send request to send
    bcf     GPIO, cs_pin           ; for transmit buffer 2
    call    spi_send              ; when a system error occurs
    bsf     GPIO, cs_pin           ;
    retfie

; *************************************************************************
; * CAN Msg Received Routine                                        *
; * This routine is called when a message has been received into     *
; * TXB0 of the MCP2510. This routine reads the filter bits to       *
; * determine the type of message received and then initiates the    *
; * appropriate response.                                           *
; *************************************************************************
msg_rcvd

    movlw   CAN_READ              ; SPI read command
    bcf     GPIO, cs_pin           ; enable MCP2510
    call    spi_send              ;
    movlw   RXB0CTRL              ; Read buffer 0 control register
    call    spi_send              ;
    call    spi_rx                ;
    bsf     GPIO, cs_pin           ; terminate function
andlw B'00000111'  ; mask off all but the FILHIT bits
movwf temp          ; store value in temp

movlw 0x01
subwf temp,1
btfsc STATUS,Z       ; filter 1 match?
goto filter1

movlw 0x02
subwf temp,1
btfsc STATUS,Z       ; filter 2 match
goto filter2

movlw 0x03
subwf temp,1
btfsc STATUS,Z       ; filter 3 match
goto filter3

movlw 0x04
subwf temp,1
btfsc STATUS,Z       ; filter 4 match
goto filter4

filter1
 call wrt_txb0sidh    ; load the transmit buffer SIDH register

bsf ADCON0,CHS0       ; select ADC channel 1
call adc_cnv           ; go do the conversion

bcf GPIO,cs_pin        ; enable MCP2510
movlw CAN_WRITE        ; send write command to MCP2510
call spi_send          ;
movlw TXB0D0           ; set write address to TXB0D0
call spi_send          ;
movfw ADRES            ; write ADC conversion result
call spi_send          ;
bsf GPIO,cs_pin        ; terminate SPI operation

goto filter_done

filter2
 call wrt_txb0sidh    ; load the transmit buffer SIDH register

bcf GPIO,cs_pin        ; enable MCP2510
movlw CAN_READ         ; send read command to MCP2510
call spi_send          ;
movlw TXRTSCCTRL       ; set read address to TXRTSCCTRL
call spi_send          ;
call spi_rx            ; read data
bsf GPIO,cs_pin        ;

bcf GPIO,cs_pin        ;
movlw CAN_WRITE        ; write TXRTSCCTRL value
call spi_send          ; to data byte zero of
movlw TXB0D0           ; transmit buffer zero
call spi_send          ;
bsf GPIO,cs_pin        ; terminate SPI operation

goto filter_done

filter3
 call wrt_txb0sidh    ; load the transmit buffer SIDH register

movlw CAN_READ         ; Read contents of receive buffer zero
bcf GPIO,cs_pin        ; byte zero to get value to write to
call spi_send ; GP output pin of MCP2510
movlw RXB1D0

movlw CAN_BIT_MODIFY
bcf GPIO,cs_pin ; set/reset the B0BFS bit of BFPCTRL register

call spi_send
movlw BFPCTRL
call spi_send
movlw B0BFS
call spi_send
movlw 0xFF ; assume that B0BFS is to be set
btfss tmp_data,0 ; test the value received in message and if it is 0
movlw 0x00 ; load w register to reset bit in BFPCTRL register

call spi_send
bsf GPIO,cs_pin
goto filter_done

filter4

call wrt_txb0sidh ; load the transmit buffer SIDH register

movlw CAN_READ ; Read contents of receive buffer zero
bcf GPIO,cs_pin ; byte zero to get value to write to

call spi_send ; GP output pin of MCP2510
movlw RXB1D0

call spi_send
call spi_rx
bsf GPIO,cs_pin
movwf tmp_data ; store value in tmp_data

movlw CAN_BIT_MODIFY ; use bit modify command to
bcf GPIO,cs_pin ; set/reset the B0BFS bit of BFPCTRL register

call spi_send
movlw BFPCTRL
call spi_send
movlw B1BFS
call spi_send
movlw 0xFF ; assume that B1BFS is to be set
btfss tmp_data,0 ; test the value received in message and if it is 0
movlw 0x00 ; load w register to reset bit in BFPCTRL register

call spi_send
bsf GPIO,cs_pin
goto filter_done

filter_done

movlw CAN_RTS_TXB0 ; last step is to send the
bcf GPIO,cs_pin ; request to send command for

call spi_send ; transmit buffer zero
bsf GPIO,cs_pin

return
; **************************************************************************
; * write TXB0SIDH                                                        *
; * This routine reads the SIDH register from the received message       *
; * and then sets the SID3 bit and writes the new value to the TX buffer. *
; **************************************************************************

wrt_tcb0sidh
    movlw    CAN_READ        ; SPI read command
    bcf      GPIO,cs_pin     ; enable MCP2510
    call     spi_send
    movlw    RXB0SIDH        ; Read received SIDH register
    call     spi_send
    call     spi_rx
    bsf      GPIO,cs_pin     ; terminate function
    movwf    tmp_data        ; store SIDH value in data
    bcf      GPIO,cs_pin
    movlw    CAN_WRITE
    call     spi_send
    movlw    TXB0SIDH        ; write to the SIDH register
    call     spi_send
    movf     tmp_data,0      ; retrieve SIDH value of received message
    bsf      W,0             ; set bit SID3 high
    call     spi_send
    bsf      GPIO,cs_pin
    return

; **************************************************************************
; * Analog to Digital Conversion Routine                                 *
; * This routine initiates the A/D conversion. The ADC channel select     *
; * select bits (CHS1:0) have to be set prior to this routine being called.*
; * The routine waits for the conversion to complete before returning to  *
; * the calling function.                                                 *
; **************************************************************************

adc_cnv
    bsf      ADCON0,GO
adc_busy
    btfsc    ADCON0,GO_DONE  ; wait for ADC to complete
    goto     adc_busy
    movlw    CAN_WRITE      ; SPI write command
    bcf      GPIO,cs_pin     ; lower CS line
    call     spi_send
    movlw    TXB0DO         ; data being written to data byte zero of buff 0
    call     spi_send
    movf     ADRES,0        ; Move ADC value to W register
    call     spi_send
    bsf      GPIO,cs_pin    ; terminate SPI command
    return

; **************************************************************************
; * Include the custom three wire SPI support file                       *
; **************************************************************************

#include "spi.inc"            ; SPI routines
; *******************************************************************
; * MCP2510 register initialization table                           *
; * Store at the end of ROM memory                                  *
; * Note that all addresses are initialized to simplify the        *
; * initialization code.                                            *
; *******************************************************************

org 0x0700
reg_init_tbl
addwf PCL, 1 ; Register Addr
; --------- ----
retlw 0xff    ; RXF0SIDH 0x00
retlw 0xff    ; RXF0SIDL 0x01
retlw 0xff    ; RXF0EID8 0x02
retlw 0xff    ; RXF0EID0 0x03
retlw 0xff    ; RXF1SIDH 0x04
retlw 0xff    ; RXF1SIDL 0x05
retlw 0xff    ; RXF1EID8 0x06
retlw 0xff    ; RXF1EID0 0x07
retlw 0x7e    ; RXF2SIDH 0x08    Filter 2 matches 0x3f0
retlw 0x00    ; RXF2SIDL 0x09
retlw 0xff    ; RXF2EID8 0x0A
retlw 0xff    ; RXF2EID0 0x0B
retlw 0x3c    ; BFPCTRL  0x0C    BFP pins as digital outputs, initial
; state hi
retlw 0x00    ; TXRTSCTRL 0x0D    TXRTS pins as digital inputs
retlw 0x80    ; CANSTAT  0x0E
retlw 0x80    ; CANCTRL  0x0F
retlw 0x7e    ; RXF3SIDH 0x10    Filter 3 matches 0x3f1
retlw 0x20    ; RXF3SIDL 0x11
retlw 0xff    ; RXF3EID8 0x12
retlw 0xff    ; RXF3EID0 0x13
retlw 0x7e    ; RXF4SIDH 0x14    Filter 4 matches 0x3f2
retlw 0x40    ; RXF4SIDL 0x15
retlw 0xff    ; RXF4EID8 0x16
retlw 0xff    ; RXF4EID0 0x17
retlw 0x7e    ; RXF5SIDH 0x18    Filter 5 matches 0x3f3
retlw 0x50    ; RXF5SIDL 0x19
retlw 0xff    ; RXF5EID8 0x1A
retlw 0xff    ; RXF5EID0 0x1B
retlw 0x00    ; TEC    0x1C
retlw 0x00    ; REC    0x1D
retlw 0x80    ; CANSTAT  0x1E
retlw 0x80    ; CANCTRL  0x1F
retlw 0xff    ; RXM0SIDH 0x20    Enable all mask bits so that no msg’s
retlw 0xff    ; RXM0SIDL 0x21    are received into RXB0
retlw 0xff    ; RXM0EID8 0x22
retlw 0xff    ; RXM0EID0 0x23
retlw 0x7e    ; RXM1SIDH 0x24    Set RXM1 to match msg ID’s of 0x3f0
; to 0x3ff
retlw 0x00    ; RXM1SIDL 0x25
retlw 0x00    ; RXM1EID8 0x26
retlw 0x00    ; RXM1EID0 0x27
retlw 0x02    ; CNF3    0x28    PHSEG2 = 3TQ
retlw 0x90    ; CNF2    0x29    PHSEG1 = 3TQ, PRSEG = 1TQ
retlw 0x03    ; CNF1    0x2A    SJW = 1TQ, BRP set to 4
retlw 0x22    ; CANINTE  0x2B    MERRIE and RX1IE enabled
retlw 0x00    ; CANINTF  0x2C
retlw 0x00    ; EFLG    0x2D
retlw 0x80    ; CANSTAT  0x2E
retlw 0x80    ; CANCTRL  0x2F
retlw 0x03    ; TXB0CTRL 0x30    Highest priority
retlw 0x7e ; TXB0SIDH 0x31
retlw 0x00 ; TXB0SIDL 0x32
retlw 0x00 ; TXB0EID8 0x33
retlw 0x00 ; TXB0EID0 0x34
retlw 0x01 ; TXB0DLC 0x35
retlw 0x00 ; TXB0DB0 0x36
retlw 0x00 ; TXB0DB1 0x37
retlw 0x00 ; TXB0DB2 0x38
retlw 0x00 ; TXB0DB3 0x39
retlw 0x00 ; TXB0DB4 0x3A
retlw 0x00 ; TXB0DB5 0x3B
retlw 0x00 ; TXB0DB6 0x3C
retlw 0x00 ; TXB0DB7 0x3D
retlw 0x80 ; CANSTAT 0x3E
retlw 0x80 ; CANCTRL 0x3F

retlw 0x03 ; TXB1CTRL 0x40 Highest priority
retlw 0x7e ; TXB1SIDH 0x41
retlw 0xe0 ; TXB1SIDL 0x42
retlw 0x00 ; TXB1EID8 0x43
retlw 0x00 ; TXB1EID0 0x44
retlw 0x01 ; TXB1DLC 0x45
retlw 0x00 ; TXB1DB0 0x46
retlw 0x00 ; TXB1DB1 0x47
retlw 0x00 ; TXB1DB2 0x48
retlw 0x00 ; TXB1DB3 0x49
retlw 0x00 ; TXB1DB4 0x4A
retlw 0x00 ; TXB1DB5 0x4B
retlw 0x00 ; TXB1DB6 0x4C
retlw 0x00 ; TXB1DB7 0x4D
retlw 0x80 ; CANSTAT 0x4E
retlw 0x80 ; CANCTRL 0x4F

retlw 0x03 ; TXB2CTRL 0x50
retlw 0x7e ; TXB2SIDH 0x51
retlw 0xe0 ; TXB2SIDL 0x52
retlw 0x00 ; TXB2EID8 0x53
retlw 0x00 ; TXB2EID0 0x54
retlw 0x00 ; TXB2DLC 0x55
retlw 0x00 ; TXB2DB0 0x56
retlw 0x00 ; TXB2DB1 0x57
retlw 0x00 ; TXB2DB2 0x58
retlw 0x00 ; TXB2DB3 0x59
retlw 0x00 ; TXB2DB4 0x5A
retlw 0x00 ; TXB2DB5 0x5B
retlw 0x00 ; TXB2DB6 0x5C
retlw 0x00 ; TXB2DB7 0x5D
retlw 0x80 ; CANSTAT 0x5E
retlw 0x80 ; CANCTRL 0x5F

retlw 0x20 ; RXB0CTRL 0x60 Receive only Standard Id’s that match
; Masks/Filters
retlw 0x00 ; RXB0SIDH 0x61
retlw 0x00 ; RXB0SIDL 0x62
retlw 0x00 ; RXB0EID8 0x63
retlw 0x00 ; RXB0EID0 0x64
retlw 0x00 ; RXB0DLC 0x65
retlw 0x00 ; RXB0DB0 0x66
retlw 0x00 ; RXB0DB1 0x67
retlw 0x00 ; RXB0DB2 0x68
retlw 0x00 ; RXB0DB3 0x69
retlw 0x00 ; RXB0DB4 0x6A
retlw 0x00 ; RXB0DB5 0x6B
retlw 0x00 ; RXB0DB6 0x6C
retlw 0x00 ; RXB0DB7 0x6D
retlw 0x80 ; CANSTAT 0x6E
retlw 0x80 ; CANCTRL 0x6F
retlw 0x20 ; RXB1CTRL 0x70 Receive only Standard Id’s that match

Masks/Filters
END
Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip’s products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, KEELoo, MPLAB, PIC, PICmicro, PICSTART and PRO MATE are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

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

dsPIC, dsPICDEM.net, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, PICC, PICDEM, PICDEM.net, rPIC, Select Mode and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

Serialized Quick Turn Programming (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.

© 2002, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

Printed on recycled paper.

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999 and Mountain View, California in March 2002. The Company’s quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, non-volatile memory and analog products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001 certified.
AMERICAS
Corporate Office
2335 West Chandler Blvd.
Chandler, AZ  85224-6199
Tel: 480-792-7200  Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

Rocky Mountain
2335 West Chandler Blvd.
Chandler, AZ  85224-6199
Tel: 480-792-7966  Fax: 480-792-4338

Atlanta
500 Sugar Mill Road, Suite 200B
Atlanta, GA  30350
Tel: 770-640-0034  Fax: 770-640-0307

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848  Fax: 978-692-3821

Chicago
333 Pierce Road, Suite 180
Itasca, IL  60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423  Fax: 972-818-2924

Detroit
Tri-Atria Office Building
2200 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Kokomo
2767 S. Albright Road
Kokomo, Indiana 46902
Tel: 765-864-8360 Fax: 765-864-8387

Los Angeles
18201 Von Karman, Suite 1009
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7965

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699  Fax: 905-673-6509

ASIA/PACIFIC
Australia
Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

China - Beijing
Microchip Technology Consulting (Shanghai) Co., Ltd., Beijing Liaison Office
Unit 915
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu
Microchip Technology Consulting (Shanghai) Co., Ltd., Chengdu Liaison Office
Rm. 2401, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-86766200 Fax: 86-28-86766599

China - Fuzhou
Microchip Technology Consulting (Shanghai) Co., Ltd., Fuzhou Liaison Office
Unit 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-753506 Fax: 86-591-753521

China - Shanghai
Microchip Technology Consulting (Shanghai) Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

China - Shenzhen
Microchip Technology Consulting (Shanghai) Co., Ltd., Shenzhen Liaison Office
Rm. 1315, 13/F, Shenzhen Kerry Centre,
Renninan Lu
Shenzhen 518001, China
Tel: 86-755-2366086

China - Hong Kong SAR
Microchip Technology Hong Kong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

India
Microchip Technology India
India Liaison Office
Divyacree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O’Shaughnessy Road
Bangalore, 560 025, India
Tel: 91-80-2290061  Fax: 91-80-2290062

Japan
Microchip Technology Japan K.K.
Benex S-1 6F
3-18-20, Shin yokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471-6168 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-862
Tel: 82-2-554-7200  Fax: 82-2-558-5934

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road #07-02 Prime Centre
Singapore, 188960
Tel: 65-6334-8870 Fax: 65-6334-8850

Taiwan
Microchip Technology (Barbados) Inc.,
Taiwan Branch
11F-3, No. 207
Tung Hua North Road
Taipei, 106, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE
Austria
Microchip Technology Austria GmbH
Durisolstrasse 2
A-4600 Wels

Tel: 43-7242-2244-399 Fax: 43-7242-2244-393

Denmark
Microchip Technology Nordic ApS
Regus Business Centre
Lautrup høj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Microchip Technology SARL
Parc d’Activite du Moulin de Massy
43 Rue du Saule Trappe
Batiment A - 1er Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Microchip Technology GmbH
Steinheilstrasse 10
D-85737 Ismaning, Germany
Tel: 49-89-627-144-0 Fax: 49-89-627-144-44

Italy
Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza

Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom
Microchip Ltd.
505 Eskdale Road
Winnersh Triangle
Berkshire, England RG41 5TU

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