MICROCHIP

AN719

Interfacing Microchip's MCP3201 Analog-to-Digital Converter to the PICmicro® Microcontroller

Author: Richard L. Fischer
Microchip Technology Inc.

INTRODUCTION

Many of the embedded control systems designed today require some flavor of a Analog-to-Digital (A/D) Converter. Embedded system applications such as data acquisition, sensor monitoring and instrumentation and Control all have varying A/D Converter requirements.

For the most part, these A/D Converter requirements are a combination of performance, cost, package size, and availability. Microchip offers a variety of solutions to meet these design requirements. The first possible solution is to implement the PICmicro® microcontroller (MCU). The PICmicro MCU offers many options for smart solutions. One of these features is the A/D Converter module. These A/D Converter modules are primarily successive approximation register (SAR) type and range in functionality from 8- to 12-bit with channel size ranges of 4 to 16. For example, the PIC16C77 has 8-channels of 8-bit A/D Converter, while the PIC17C766 has 16-channels of 10-bit A/D Converter.

These on-board A/D Converter modules fit well into embedded applications, which requires a 10-35ksps A/D Converter.

For those applications which require a higher performance or remote sense capability, the Microchip MCP3201, 12-bit A/D Converter fits very nicely.

The MCP3201 employs a classic SAR architecture. The device uses an internal sample and hold capacitor to store the analog input while the conversion is taking place. Conversion rates of 100ksps are possible on the MCP3201. Minimum clock speed (10kHz or 625sps, assuming 16 clocks) is a function of the capacitors used for the sample and hold.

The MCP3201 has a single pseudo-differential input. The (IN–) input is limited to ±100mV. This can be used to cancel small noise signals present on both the (IN+) and (IN–) inputs. This provides a means of rejecting noise when the (IN–) input is used to sense a remote signal ground. The (IN+) input can range from the (IN–) input to $V_{REF.}$

The reference voltage for the MCP3201 is applied to V_{REF} pin. V_{REF} determines the analog input voltage range and the LSB size, i.e.:

$$LSB\ size = \frac{V_{REF}}{2^{12}}$$

As the reference input is reduced, the LSB size is reduced accordingly.

Communication with the MCP3201 is accomplished using a standard SPI™ compatible serial interface. This interface allows direct connection to the serial ports of MCUs and digital signal processors.

In order to simplify the design process for implementing the MCP3201, Microchip has written C and assembly code routines for a PIC16C67 to communicate with the MCP3201 A/D Converter.

Figure 1 shows the hardware schematic implemented in this application. Appendix A contains a listing of the C source code. Appendix B contains a listing of the assembly source code.

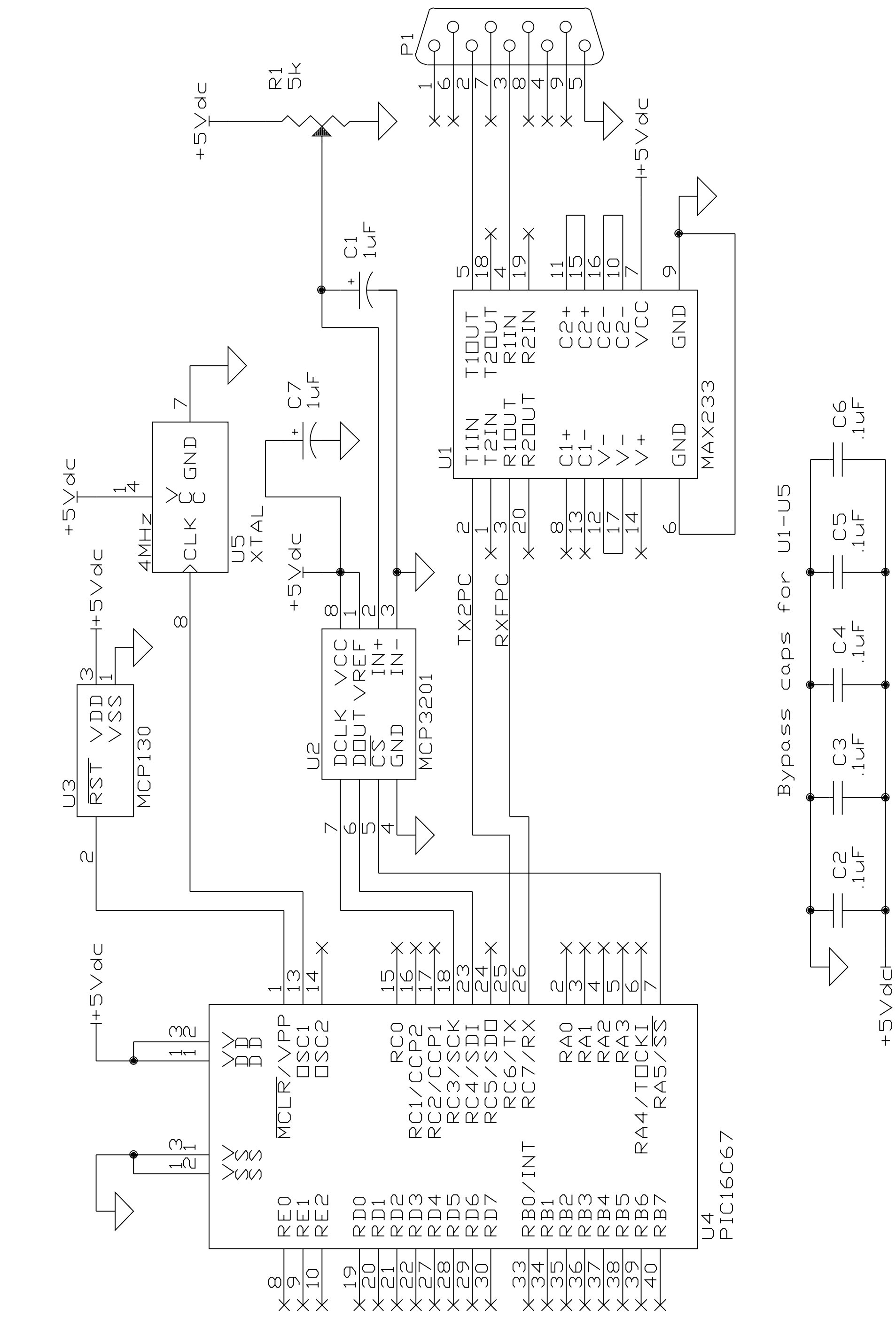

FIGURE 1: *MCP3201 A/D Converter to PICmicro MCU Interface.*

CIRCUIT DESCRIPTION

The serial interface of the Microchip MCP3201 A/D Converter has three wires, a serial clock input (DCLK), the serial data output (D_{OUT}) and the chip select input signal ($\overline{CS}$/SHDN). For this simple circuit interface, the PICmicro PIC16C67 SPI port is used. PortC:<3> is configured for the serial clock and PortC:<4> is the data input to the PICmicro. The SPI clock rate for this application is set at 1MHz.

The PIC16C67 is configured in the master mode with its CKP bit set to logic 1 and CKE bit set to logic 0. This configuration is the SPI bus mode 1,1.

A conversion is intiated with the high to low transition of $\overline{CS}$/SHDN (active low). The chip select is generated by PORTA:<5> of the PICmicro. The device will sample the analog input from the rising edge on the first clock after $\overline{CS}$ goes low for 1.5 clock cycles. On the falling edge of the second clock, the device will output a low null bit. The next 12 clocks will output the result of the conversion with the MSB first (See Figure 2 and Figure 3). Data is always output from the device on the falling edge of the clock. If the device continues to receive clocks while $\overline{CS}$/SHDN is low, the device will output the conversion LSB first. If more clocks are provided to the device while $\overline{CS}$/SHDN is still low (after the LSB first data has been transmitted), the device will clock out zeros indefinitely.

As the analog input signal is applied to the IN+ and IN- inputs, it is ratioed to the V_{REF} input for conversion scaling.

$$\textit{Digital output code} = \frac{V_{IN} \times F.S.}{V_{REF}}$$

Where:

V_{IN} = analog input voltage V(IN+) - V(IN–)

V_{REF} = reference voltage

F.S. = full scale = 4096

V_{REF} can be sourced directly from V_{DD} or can be supplied by an external reference. In either configuration, the V_{REF} source must be evaluated for noise contributions during the conversion. The voltage reference input, V_{REF} of the MCP3201 ranges from 250mV to $5V_{DC}$ which approximately translates to a corresponding LSB size from 61µV to 1.22mV per bit.

$$1.22mV = \frac{5V_{DC}}{2^{12}\ bits}$$

For this simple application, the MCP3201 voltage reference input is tied to $5V_{DC}$. This translates to a 1.22mV / bit resolution for the A/D Converter module. The voltage input to the MCP3201 is implemented with a multi-turn potentiometer. The output voltage range of this passive driver is approximately $0V_{DC}$ to $5V_{DC}$.

Finally, a simple RS-232 interface is implemented using the USART peripheral of the PICmicro and a MAX233 transceiver IC. The USART transmits the captured A/D Converter binary value, both in ASCII and corresponding voltage to the PC terminal at 9600 baud.

With a few discrete components, a MCP3201 A/D Converter IC., and a PICDEM-2 demonstration board, this simple application can be implemented.

As with all applications which require moderate to high performance A/D Converter operation, proper grounding and layout techniques are essential in achieving optimal performance. Proper power supply decoupling and input signal and V_{REF} parameters must be considered for noise contributions.

SOURCE CODE DESCRIPTION

The code written for this application performs six functions:

1. PICmicro Initialization
2. A/D Conversion
3. Conversion to ASCII
4. Conversion to Decimal
5. Conversion to Voltage (*C code only)
6. Transmit ASCII, Decimal and Voltage to PC for display.

C CODE:

Upon power up, three initialization routines are called and executed. These routines initialize the PICmicro Port pins, USART peripheral and SSP module for SPI functionality. The default PICmicro SPI bus mode is 1,1. To place the PICmicro in SPI bus mode 0,0, comment out the “#define mode11” definition statement and rebuild the project.

Upon completion of the initialization routines, the main code loop is entered and executed every ~150ms. This continuous loop consists of performing an analog conversion, transmitting the results to the PC for display, delaying for ~150ms and then repeating the loop.

The A/D conversion sequence is initiated every time $\overline{CS}$/SHDN is asserted. PortA:<5> is used as the $\overline{CS}$/SHDN to the MCP3201. After asserting PortA:<5>, the SSPBUF register is written to, for initiating a SPI bus cycle. When the SPI cycle is complete, (BF flag is set to logic 1), the received data is read from the SSPBUF register and written to the RAM array variable "adc_databyte[1]". The SSPBUF register is again written to, which initiates a SPI bus cycle, and the second 8-bits are received and written to the RAM array variable "adc_databyte[0]". The $\overline{CS}$/SHDN is then negated and the MCP3201 enters into the shutdown mode.

Next, the “Display_Adc_Result” routine is called and executed. Here the composite result, located in array variable “adc_databyte” is right adjusted one bit location. Then a printf statement is executed which formats and sends the data to the USART for transmission to the PC for display. The data output is in three formats: ASCII, Decimal and Voltage.

ASSEMBLY CODE:

Upon power up, three initialization routines are called and executed. These routines initialize the PICmicro Port pins, USART peripheral and SSP module for SPI functionality. The default PICmicro SPI bus mode is 1,1. To place the PICmicro in SPI bus mode 0,0, comment out the "#define mode11" statement and rebuild the project.

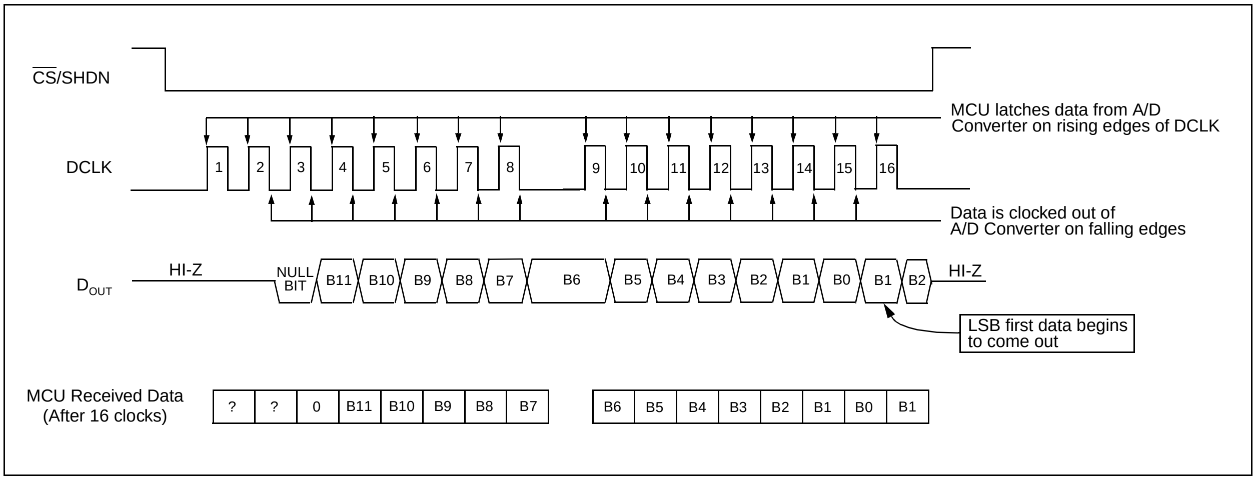

FIGURE 2: *SPI Communication using 8-bit segments (Mode 0,0: DCLK idles low).*

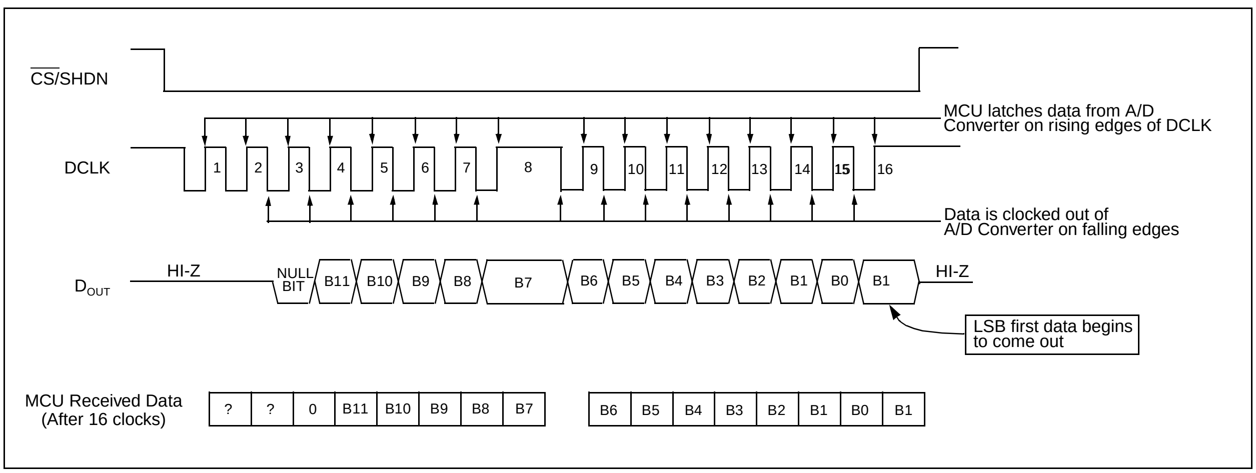

FIGURE 3: *SPI Communication using 8-bit segments (Mode 1,1: DCLK idles high).*

Upon completion of the initialization routines, the main code loop is entered and executed every ~150ms. This continuous loop consists of performing an analog conversion, converting the A/D Converter binary data into Decimal and ASCII and then transmitting the results to the PC for display, delaying for ~150ms and then repeating the loop.

The A/D conversion sequence is initiated every time $\overline{CS}$/SHDN is asserted. PortA:<5> is used as the $\overline{CS}$/SHDN to the MCP3201. After asserting PortA:<5>, the SSPBUF register is written to, for initiating a SPI bus cycle. When the SPI cycle is complete, (BF flag is set to logic 1), the received data is read from the SSPBUF register and written to the RAM variable "adc_result+1". The SSPBUF register is again written to, which initiates a SPI bus cycle, and the second 8-bits are received and written to the RAM variable "adc_result". Here the composite result, located in variable adc_result is right adjusted one bit location. The $\overline{CS}$/SHDN is negated and the MCP3201 enters into the shutdown mode.

Next, the “Hex_Dec” and “Hex_Ascii” routines are executed which convert the raw A/D Converter binary data into Decimal and ASCII values. Then, the ”Display_Data” routine is executed which sends the data to the USART for transmission to the PC for display.

REFERENCES

Williams, Jim, “Analog Circuit Design”, *Butterworth-Heinemann.*

Baker, Bonnie, “Layout Tips for 12-bit A/D Converter Applications”, *AN688, Microchip Technology Inc.*

MCP3201 12-bit A/D Converter with SPI Serial Interface, *Microchip Technology, Document # DS21290B, 1999.*

APPENDIX A:

```
/*********************************************************************
*                                                                    *
*   Interfacing Microchip’s MCP3201 ADC to the PICmicro MCU          *
*                                                                    *
**********************************************************************
*                                                                    *
*   Filename:       mcp3201.c                                        *
*   Date:           06/30/99                                         *
*   File Version:   1.00                                             *
*                                                                    *
*   Compiler:       Hi-Tech PIC C Compiler V7.84 PL1                 *
*                   MPLAB V4.12.00                                   *
*                                                                    *
*   Author:         Richard L. Fischer                               *
*                   Microchip Technology Incorporated                *
*                                                                    *
**********************************************************************
*                                                                    *
*   Files required:                                                  *
*                                                                    *
*                   pic.h      - Hi-Tech provided file               *
*                   stdio.h    - Hi-Tech provided file               *
*                   cnfig67.h                                        *
*                   mcp3201.h                                        *
*                                                                    *
*                   mcp3201.c                                        *
*                   mprnt.c    - Hi-Tech provided file               *
*                                                                    *
**********************************************************************
*                                                                    *
*                                                                    *
*  This code demonstrates how the Microchip MCP3201 Analog-to-Digital*
*  Converter (ADC) is interfaced to the Synchronous Serial Peripheral*
*  (SSP) of the PICmicro MCU. For this application note the PICmicro *
*  PIC16C67 is selected. The interface uses two Serial Peripheral    *
*  Interface (SPI) lines (SCK, SDI) on the PICmicro for the clock    *
*  (SCK) and data in (SDI). A chip select (CS) to the MCP3201 is     *
*  generated with a general purpose port line PORTA:<5>. The simple  *
*  application uses Mode 1,1 to define bus clock polarity and        *
*  phase.                                                            *
*                                                                    *
*  For this application, the SPI data rate is set to one fourth      *
*  (FOSC/4) of the microcontroller clock frequency. The PIC16C67     *
*  device clock frequency used for this application is 4MHz. This    *
*  translates to an ADC throughput of approximately 62.5kHz. In      *
*  order to obtain the maximum throughput (100kHz) from the          *
*  MCP3201 ADC the PIC16C67 should be clocked at 6.4Mhz.             *
*                                                                    *
*                                                                    *
*                                                                    *
*********************************************************************/

    #include     <pic.h>                          // processor if/def file
    #include     <stdio.h>
    #include     “cnfig67.h”                      // configuration word definitions
    #include     “mcp3201.h”

_CONFIG   ( CONBLANK & BODEN_ON & PWRTE_ON & CP_OFF & WDT_OFF & XT_OSC );

/* SPI Bus mode selection */
#define mode11                                    // comment out and rebuild for mode 00
```

```
/*******************************************************************

  MAIN PROGRAM BEGINS HERE

*******************************************************************/

void main( void )
{
    Init_Ports();                              // initialize ports
    Init_SSP();                                // initialize SSP module
    Init_Usart();                              // initialize USART module

    while ( TRUE )                             // loop forever
    {
        Read_Adc( );                           // initiate MCP3201 conversion and read result
        Display_Adc_Result();                  // display results via USART to PC
        Delay_10mS( 15 );                      // 150mS delay
    }
}



void Delay_10mS( char loop_count )             // approximate 10mS base delay
{
    unsigned int inner;                        // declare integer auto variable
    char outer;                                // declare char auto variable

    while ( loop_count )                       // stay in loop until done
    {
      for ( outer = 9; outer > 0; outer-- )
       for ( inner = 249; inner > 0; inner-- );
      loop_count--;
    }
}



void putch( char data )
{
    while ( !TRMT );                           // wait until TSR is empty
    TXREG = data;                              // write data to USART
}



void Read_Adc( void )
{
    CS = 0;                                    // assert MCP3201 chip select
    SSPBUF = 0x01;                             // initiate a SPI bus cycle
    while ( !STAT_BF );                        // wait until cycle completes
    adc.databyte[1] = SSPBUF;                  // transfer ADC MSbyte into buffer

    SSPBUF = 0x81;                             // initiate a SPI bus cycle
    while ( !STAT_BF );                        // wait until cycle completes
    CS = 1;                                    // negate MCP3201 chip select
    adc.databyte[0] = SSPBUF;                  // transfer ADC LSbyte into buffer
}



void Display_Adc_Result( void )
{
    double temp;                               // define auto type variable
    adc.result >>= 1;                          // adjust composite integer for 12 valid bits
    adc.result &= 0x0FFF;                      // mask out upper nibble of integer
    temp = ( adc.result * 0.001225585 );       // compute floating point result
```

```
    printf( “Hex->0x%X  :  Decimal->%u  :  %4.3f Vdc\n\r”, adc.result, adc.result, temp );
}



void Init_Usart( void )
{
    SPBRG = 25;                                 // set baud rate for 9600 @ 4MHz
    TXSTA = 0x24;                               // BRGH = 1, enable transmitter
    RCSTA = 0x90;                               // enable serial port
}



void Init_SSP( void )
{
#ifdef mode11
    SSPSTAT = 0b00000000;                       // Master sample data in middle, data xmt on
                                                // rising edge
    SSPCON = 0b00110000;                        // enable Master SPI, bus mode 1,1, FOSC/4

#else if
    SSPSTAT = 0b01000000;                       // Master sample data in middle, data xmt on
                                                // rising edge
    SSPCON = 0b00100000;                        // enable Master SPI, bus mode 0,0, FOSC/4

#endif
}



void Init_Ports( void )
{
    PORTA = 0b100000;                           // set PORTA data latches to initial state
    PORTB = 0x00;                               // set PORTB data latches to initial state
    PORTC = 0b11010000;                         // set PORTC data latches to initial state
    PORTD = 0x00;                               // set PORTD data latches to initial state
    PORTE = 0x00;                               // set PORTE data latches to initial state

    TRISA = 0b000000;                           // set PORTA pin direction
    TRISB = 0x00                                // set PORTB pin direction
    TRISC = 0b11010000;                         // set PORTC pin direction
    TRISD = 0x00;                               // set PORTD pin direction
    TRISE = 0x00;                               // set PORTE pin direction
}
```

```
/*******************************************************************
*                                                                  *
*   Filename:       mcp3201.h                                      *
*   Date:           06/30/99                                       *
*   File Version:   1.00                                           *
*                                                                  *
*                                                                  *
*******************************************************************/



// FUNCTION PROTOTYPES DECLARED HERE

void Read_Adc( void );
void Display_Adc_Result( void );
void Delay_10mS( char loop_count );
void Init_Usart( void );
void Init_SSP( void );
void Init_Ports( void );


union {
           char databyte[2];                       // declare temp array for adc data
           unsigned int result;                    // declare integer for adc result
} adc;                                             // define union variable


#define TRUE    1

#define PortBit(port,bit)    ((unsigned)&(port)*8+(bit))

static bit CS  @  PortBit(PORTA,5);                // MCP3201 Chip Select
```

```
/*******************************************************************
*                                                                  *
*   Filename:       cnfig67.h                                      *
*   Date:           06/30/99                                       *
*   File Version:   1.00                                           *
*                                                                  *
*                                                                  *
*******************************************************************/

/*****  CONFIGURATION BIT DEFINITIONS FOR PIC16C67 PICmicro  *****/

#define       CONBLANK          0x3FFF

#define       CP_ALL            0x00CF
#define       CP_75             0x15DF
#define       CP_50             0x2AEF
#define       CP_OFF            0x3FFF
#define       BODEN_ON          0x3FFF
#define       BODEN_OFF         0x3FBF
#define       PWRTE_OFF         0x3FFF
#define       PWRTE_ON          0x3FF7
#define       WDT_ON            0x3FFF
#define       WDT_OFF           0x3FFB
#define       LP_OSC            0x3FFC
#define       XT_OSC            0x3FFD
#define       HS_OSC            0x3FFE
#define       RC_OSC            0x3FFF
```

APPENDIX B:

```
;*********************************************************************
;                                                                    *
;   Interfacing Microchip’s MCP3201 ADC to the PICmicro MCU          *
;                                                                    *
;*********************************************************************
;                                                                    *
;   Filename:       mcp3201.asm                                      *
;   Date:           06/30/99                                         *
;   File Version:   1.00                                             *
;                                                                    *
;   Assembler:      MPASM  V2.30.00                                  *
;   Linker:         MPLINK V1.30.01                                  *
;                   MPLAB  V4.12.00                                  *
;                                                                    *
;   Author:         Richard L. Fischer                               *
;   Company:        Microchip Technology Incorporated                *
;                                                                    *
;*********************************************************************
;                                                                    *
;   Files required:                                                  *
;                                                                    *
;                   mcp3201.asm                                      *
;                   hexdec.asm                                       *
;                   hexascii.asm                                     *
;                                                                    *
;                   p16c67.inc                                       *
;                   16c67.lkr                                        *
;                                                                    *
;                                                                    *
;*********************************************************************
;                                                                    *
;  This code demonstrates how the Microchip MCP3201 Analog-to-Digital*
;  Converter (ADC) is interfaced to the Synchronous Serial Peripheral*
;  (SSP) of the PICmicro MCU. For this application note the PICmicro *
;  PIC16C67 is selected. The interface uses two Serial Peripheral    *
;  Interface (SPI) lines (SCK, SDI) on the PICmicro for the clock    *
;  (SCK) and data in (SDI). A chip select (CS) to the MCP3201 is     *
;  generated with a general purpose port line PORTA:<5>. The simple  *
;  application uses Mode 1,1 to define bus clock polarity and        *
;  phase.                                                            *
;                                                                    *
;  For this application, the SPI data rate is set to one fourth      *
;  (FOSC/4) of the microcontroller clock frequency. The PIC16C67     *
;  device clock frequency used for this application is 4MHz. This    *
;  translates to an ADC throughput of approximately 62.5kHz. In      *
;  order to obtain the maximum throughput (100kHz) from the          *
;  MCP3201 ADC the PIC16C67 should be clocked at 6.4Mhz.             *
;                                                                    *
;                                                                    *
;                                                                    *
;********************************************************************/


    list         p=16c67                        ; list directive to define processor
    #include     <p16c67.inc>                   ; processor specific variable definitions

    __CONFIG   _BODEN_ON & _PWRTE_ON & _CP_OFF & _WDT_OFF & _XT_OSC


#define mode11                                  ; if SPI bus mode 1,1 is desired
                                                ; else comment out and rebuild for mode 0,0
```

```
;***** VARIABLE DEFINITIONS

TEMP_VAR         UDATA      0x20                 ;
adc_result       RES        2                    ; variable used for context saving
offset           RES        1
temp             RES        1


TEMP_VAR1        UDATA_OVR                       ; create udata overlay section
counthi          RES        1
countlo          RES        1


           GLOBAL   adc_result                   ; make variables available to other modules
           EXTERN   Hex_Dec                      ; reference linkage
           EXTERN   Hex_Ascii                    ; reference linkage
           EXTERN   adc_temph, adc_templ         ; reference linkage
           EXTERN   thous                        ; reference linkage


#define       CS      PORTA,5                    ; MCP3201 Chip Select
#define       CR      0x0D                       ; macro for carriage return
#define       LF      0x0A                       ; macro for line feed


;**********************************************************************

RESET_VECTOR     CODE       0x000                ; processor reset vector
           movlw    high start                   ; move literal into W
           movwf    PCLATH                       ; initialize PCLATH
           goto     start                        ; go to beginning of program


INT_VECTOR       CODE       0x004                ; interrupt vector location
; no interrupt code needed for this application



MAIN             CODE       0x040                ; set code section to start at 0x040
start
           call       Init_Ports                 ; initialize ports
           call       Init_SSP                   ; initialize SSP module
           call       Init_Usart                 ; initialize USART module

forever    call       Read_Adc                   ; read MCP3201 ADC
           call       Hex_Dec                    ; convert adc_result to decimal
           call       Hex_Ascii                  ; convert adc_result to ASCII
           call       Display_Data               ; display data to PC
           call       Delay_150mS                ; 150mS delay
           goto       forever                    ; continuos loop



; Read MCP3201 ADC for 2 bytes
Read_Adc
           banksel    PORTA                      ; linker to select SFR bank
           bcf        CS                         ; assert MCP3201 chip select
           movlw      0x01                       ; move literal into W
           banksel    SSPBUF                     ; linker to select SFR bank
           movwf      SSPBUF                     ; initiate SPI bus cycle
           banksel    SSPSTAT                    ; linker to select SFR bank
spi_busy1  btfss      SSPSTAT,BF                 ; test, is bus cycle complete?
           goto       spi_busy1                  ; wait, bus cycle not complete
           banksel    SSPBUF                     ; linker to select SFR bank
           movf       SSPBUF,w                   ; read SSPBUF and place into W
           banksel    adc_result                 ; linker to select GPR bank
```

```
          movwf      adc_result+1                 ; write SSPBUF to adc_result

          movlw      0x81                         ; move literal into W
          banksel    SSPBUF                       ; linker to select SFR bank
          movwf      SSPBUF                       ; initiate SPI bus cycle
          banksel    SSPSTAT                      ; linker to select SFR bank
spi_busy2 btfss      SSPSTAT,BF                   ; test, is bus cycle complete?
          goto       spi_busy2                    ; wait, bus cycle not complete
          banksel    PORTA                        ; linker to select SFR bank
          bsf        CS                           ; negate MCP3201 chip select
          movf       SSPBUF,w                     ; read SSPBUF and place into W
          banksel    adc_result                   ; linker to select GPR bank
          movwf      adc_result                   ; write SSPBUF to adc_result

          rrf        adc_result+1,f               ; adjust MSB 1 position right
          rrf        adc_result,f                 ; adjust LSB 1 position right and include carry
          movlw      0x0F                         ; move literal into W
          andwf      adc_result+1,f               ; mask out upper nibble of ADC result

          movf       adc_result,w                 ; move adc_result LSB into W
          movwf      adc_templ                    ; save W into temp register
          movf       adc_result+1,w               ; move adc_result MSB into W
          movwf      adc_temph                    ; save W into temp register
          return                                  ; return from subroutine



; Display ADC data ( ASCII and DECIMAL ) to USART
Display_Data
          banksel    offset                       ; linker to select GPR bank
          clrf       offset                       ; initialize table index value
          movlw      high  msg1                   ; move high byte of table address -> W
          movwf      PCLATH                       ; initialize PCLATH

txlp1     movf       offset,w                     ; move offset value into W
          call       msg1                         ; retrieve table element
          movwf      temp                         ; move element into temp
          btfsc      temp,7                       ; test for end of string
          goto       send_hex                     ; end of message so send the data
          banksel    TXREG                        ; linker to select SFR bank
          movwf      TXREG                        ; initiate USART transmission
          banksel    TXSTA                        ; linker to select SFR bank
          btfss      TXSTA,TRMT                   ; test if TSR is empty
          goto       $-1                          ; stay in testing loop
          banksel    offset                       ; linker to select GPR bank
          incf       offset,f                     ; increment table index
          goto       txlp1                        ; stay in transmit loop

send_hex  movlw      adc_temph                    ; obtain variable address
          movwf      FSR                          ; initialize FSR as pointer
send_hex1 movf       INDF,w                       ; retrieve data byte
          banksel    TXREG                        ; linker to select SFR bank
          movwf      TXREG                        ; initiate USART transmission
          banksel    TXSTA                        ; linker to select bank
          btfss      TXSTA,TRMT                   ; test if TSR is empty
          goto       $-1                          ; stay in testing loop
          incf       FSR,f                        ; update pointer
          movlw      adc_temph+4                  ; compose end of string address value
          subwf      FSR,w                        ; do compare
          btfss      STATUS,C                     ; done with sending data
          goto       send_hex1                    ; no, so send some more


          banksel    offset                       ; linker to select GPR bank
          clrf       offset                       ; initialize table index value
txlp2     movf       offset,w                     ; move offset value into W
```

```
            call        msg2                        ; retrieve table element
            movwf       temp                        ; move element into temp
            btfsc       temp,7                      ; test for end of string
            goto        send_dec                    ; end of message so send the data
            banksel     TXREG                       ; linker to select SFR bank
            movwf       TXREG                       ; initiate USART transmission
            banksel     TXSTA                       ; linker to select SFR bank
            btfss       TXSTA,TRMT                  ; test if TSR is empty
            goto        $-1                         ; stay in testing loop
            banksel     offset                      ; linker to select GPR bank
            incf        offset,f                    ; increment table index
            goto        txlp2                       ; stay in transmit loop


send_dec    movlw       thous                       ; obtain variable address
            movwf       FSR                         ; initialize FSR as pointer
send_dec1   movf        INDF,w                      ; retrieve data byte
            banksel     TXREG                       ; linker to select SFR bank
            movwf       TXREG                       ; initiate USART transmission
            banksel     TXSTA                       ; linker to select SFR bank
            btfss       TXSTA,TRMT                  ; test if TSR is empty
            goto        $-1                         ; stay in loop
            incf        FSR,f                       ; update pointer
            movlw       thous+4                     ; compose end of string address value
            subwf       FSR,w                       ; do compare
            btfss       STATUS,C                    ; done with sending data
            goto        send_dec1                   ; no, so send some more

            movlw       CR                          ; move literal into W
            banksel     TXREG                       ; linker to select SFR bank
            movwf       TXREG                       ; initiate USART transmission
            banksel     TXSTA                       ; linker to select SFR bank
            btfss       TXSTA,TRMT                  ; test if TSR is empty
            goto        $-1                         ; no, so stay in loop

            movlw       LF                          ; move literal into W
            banksel     TXREG                       ; linker to select SFR bank
            movwf       TXREG                       ; initiate USART transmission
            banksel     TXSTA                       ; linker to select SFR bank
            btfss       TXSTA,TRMT                  ; test if TSR is empty
            goto        $-1                         ; no, so stay in loop
            return                                  ; return from subroutine



; Delay for ~ 150mS
Delay_150mS
            movlw       D'150'                      ; move literal into W
            banksel     counthi                     ; linker to select GPR bank
            movwf       counthi                     ; initialize upper counter
outer       movlw       D'250'                      ; move literal into W
            movwf       countlo                     ; initialize lower counter
inner       decf        countlo,f                   ; decrement counter low
            btfss       STATUS,Z                    ; is result == 0
            goto        inner                       ; no, stay in loop
            decf        counthi,f                   ; else, decrement count high
            btfss       STATUS,Z                    ; is result == 0
            goto        outer                       ; no, so start again
            return                                  ; return from subroutine


; Initialize USART Module
Init_Usart  movlw       D'25'                       ; move literal into W
            banksel     SPBRG                       ; linker to select SFR bank
            movwf       SPBRG                       ; set baud rate for 9600 @ 4MHz
            movlw       B'00100100'                 ; move literal into W
```

```
          movwf     TXSTA                         ; BRGH = 1, enable transmitter
          movlw     B'10010000'                   ; move literal into W
          banksel   RCSTA                         ; linker to select SFR bank
          movwf     RCSTA                         ; enable serial port
          return                                  ; return from subroutine


; Initialize SSP Module
Init_SSP
#ifdef mode11
          movlw     B'00110000'                   ; move literal into W
          banksel   SSPCON                        ; linker to select SFR bank
          movwf     SSPCON                        ; enable Master SPI, bus mode 1,1, FOSC/4
          banksel   SSPSTAT                       ; linker to select SFR bank
          clrf      SSPSTAT                       ; Master sample data in middle, data xmt on
                                                  ; rising edge

#else
          movlw     B'00100000'                   ; move literal into W
          banksel   SSPCON                        ; linker to select SFR bank
          movwf     SSPCON                        ; enable Master SPI, bus mode 0,0, FOSC/4
          movlw     B'01000000'                   ; move literal into W
          banksel   SSPSTAT                       ; linker to select SFR bank
          movwf     SSPSTAT                       ; Master sample data in middle, data xmt on
                                                  ; rising edge
#endif
          return                                  ; return from subroutine


; Initialize PORTS
Init_Ports movlw    0x00                          ; move literal into W
          banksel   PORTA                         ; linker to select SFR bank
          movwf     PORTB                         ; set PORTB data latches to initial state
          movwf     PORTD                         ; set PORTD data latches to initial state
          movwf     PORTE                         ; set PORTE data latches to initial state
          movlw     B'100000'                     ; move literal into W
          movwf     PORTA                         ; set PORTA data latches to initial state
          movlw     B'11010000                    ; move literal into W
          movwf     PORTC                         ; set PORTC data latches to initial state

          banksel   TRISA                         ; linker to select SFR bank
          clrf      TRISA                         ; set PORTA pin direction
          clrf      TRISB                         ; set PORTB pin direction
          clrf      TRISD                         ; set PORTD pin direction
          clrf      TRISE                         ; set PORTE pin direction
          movlw     B'11010000'                   ; move literal into W
          movwf     TRISC                         ; set PORTC pin direction
          return                                  ; return from subroutine


TABLE_DATA CODE     0x200                         ; table starts here
msg1      addwf     PCL,f                         ; generate computed goto
          DT        "HEX-> 0x",80

msg2      addwf     PCL,f                         ; generate computed goto
          DT        "  :  DECIMAL-> ",80


          END                                     ; directive 'end of program'
```

```
;*********************************************************************
;                                                                   *
;   Hex to Decimal conversion of ADC result for display             *
;                                                                   *
;*********************************************************************
;                                                                   *
;   Filename:       hexdec.asm                                      *
;   Date:           06/30/99                                        *
;   File Version:   1.00                                            *
;                                                                   *
;   Assembler:      MPASM  V2.30.00                                 *
;   Linker:         MPLINK V1.30.01                                 *
;                   MPLAB  V4.12.00                                 *
;                                                                   *
;   Author:         Richard L. Fischer                              *
;   Company:        Microchip Technology Incorporated               *
;                                                                   *
;*********************************************************************

   #include <p16c67.inc>                         ; processor specific variable definitions


           GLOBAL  Hex_Dec, thous                ; make subroutine 'Hex_Dec' available to other
                                                   modules
           EXTERN     adc_result                 ; reference linkage

HEXDEC_VAR UDATA      0x30                       ; create udata variable section
thous      RES        1                          ; reserve one location
hunds      RES        1                          ; reserve one location
tens       RES        1                          ; reserve one location
ones       RES        1                          ; reserve one location


           GLOBAL  thous, hunds, tens, ones


; *****    Subroutine begins here

HEXDEC     CODE                                  ; create code section "HEXDEC"
Hex_Dec
           banksel    thous                      ; linker to select GPR bank
           clrf       thous                      ; initialize 'thousands' variable
           clrf       hunds                      ; initialize 'hundreds' variable
           clrf       tens                       ; initialize 'tens' variable
           clrf       ones                       ; initialize 'ones' variable

chk_thous  movlw      0x04                       ; move literal into W ... 1024 (0x0400)
           banksel    adc_result+1               ; linker to select GPR bank
           subwf      adc_result+1,w             ; subtract 1024 from adc_result MSB
           btfss      STATUS,C                   ; is adc_result MSB > 1024
           goto       chk_hunds2                 ; no, so check hundreds
           incf       thous,f                    ; else, increment thousands
           movlw      0x04                       ; move literal into W
           subwf      adc_result+1,f             ; subtract 1000 from adc_result MSB
           movlw      D'24'                      ; move literal into W
           addwf      adc_result,f               ; add remainder 24 into adc_result LSB
           btfsc      STATUS,C                   ; was there a carry into adc_result MSB?
           incf       adc_result+1,f             ; yes, so increment
           goto       chk_thous                  ; go check thousands again


chk_hunds2 movlw      0x01                       ; 256 (0x0100)
           subwf      adc_result+1,w             ; subtract 200 from adc_result MSB
           btfss      STATUS,C                   ; is adc_result MSB >= 256
           goto       chk_hunds1                 ; no, so check multiples of 100
           movlw      D'2'                       ; else,
```



```
          addwf     hunds,f                     ; add 2 into hundreds
          movlw     0x01                        ; move literal into W
          subwf     adc_result+1,f              ; subtract 200 from adc_result MSB
          movlw     D'56'                       ; move remainder into W
          addwf     adc_result,f                ; add remainder 56 into adc_result LSB
          btfsc     STATUS,C                    ; was there a carry into adc_result MSB
          incf      adc_result+1,f              ; yes, so increment adc_result MSB

          movlw     D'10'                       ; move literal into W
          subwf     hunds,w                     ; check to see if hunds = 1000
          btfss     STATUS,Z                    ; is result == 0?
          goto      chk_hunds2                  ; no, so check hundreds (200) again
          clrf      hunds                       ; clear hundreds
          incf      thous,f                     ; increment thousands
          goto      chk_hunds2                  ; go check hundreds (200) some more


chk_hunds1 movlw    D'100'                      ; move literal into W
          subwf     adc_result,w                ; subtract 100 from adc_result LSB
          btfss     STATUS,C                    ; is adc_result >= 100
          goto      chk_tens                    ; no so check tens
          incf      hunds,f                     ; else, increment hundreds
          movlw     D'100'                      ; move literal into W
          subwf     adc_result,f                ; reduce hundreds count by 100

          movlw     D'10'                       ; move literal into W
          subwf     hunds,w                     ; check to see if hunds may = 1000
          btfss     STATUS,Z                    ; is result == 0?
          goto      chk_hunds1                  ; no, so check hundreds (100) again
          clrf      hunds                       ; clear hundreds
          incf      thous,f                     ; increment thousands
          goto      chk_hunds1                  ; go check hundreds (100) some more


chk_tens  movlw     D'10'                       ; move literal into W
          subwf     adc_result,w                ; subtract 10 from adc_result LSB
          btfss     STATUS,C                    ; is adc_result LSB >= 10
          goto      chk_ones                    ; no, so check ones
          incf      tens,f                      ; else, increment tens
          movlw     D'10'                       ; move literal into W
          subwf     adc_result,f                ; reduce tens count by 10
          goto      chk_tens                    ; go check tens again


chk_ones  movf      adc_result,w                ; read adc_result LSB and store into W
          movwf     ones                        ; save off as ones
          movlw     0x30                        ; move literal into W
          iorwf     thous,f                     ; compose ASCII byte (thousands)
          iorwf     hunds,f                     ; compose ASCII byte (hundreds)
          iorwf     tens,f                      ; compose ASCII byte (tenths)
          iorwf     ones,f                      ; compose ASCII byte (ones)
          return                                ; return from subroutine

          END                                   ; directive 'end of program'
```

```
;*********************************************************************
;                                                                    *
;   Hex to ASCII conversion of ADC result for display                *
;                                                                    *
;*********************************************************************
;                                                                    *
;   Filename:       hexascii.asm                                     *
;   Date:           06/30/99                                         *
;   File Version:   1.00                                             *
;                                                                    *
;   Assembler:      MPASM  V2.30.00                                  *
;   Linker:         MPLINK V1.30.01                                  *
;                   MPLAB  V4.12.00                                  *
;                                                                    *
;   Author:         Richard L. Fischer                               *
;   Company:        Microchip Technology Incorporated                *
;                                                                    *
;*********************************************************************

  #include <p16c67.inc>                      ; processor specific variable definitions

           GLOBAL    Hex_Ascii               ; make subroutine 'Hex_Ascii' available to
                                               other modules
           GLOBAL    adc_temph, adc_templ    ; reference linkage


TEMP_VAR1  UDATA_OVR                         ; create udata overlay section
adc_temph  RES       2
adc_templ  RES       2


HEXASCII   CODE                              ; create code section "HEXASCII"
Hex_Ascii
           banksel   adc_templ               ; linker to select GPR bank
           movf      adc_templ,w             ; move copy of adc_result LSB into W
           movwf     adc_templ+1             ; make copy ADC result LSB
           movf      adc_temph,w             ; move copy of adc_result MSB into W
           movwf     adc_temph+1             ; make copy ADC result MSB
           movlw     0x30                    ; move literal into W
           movwf     adc_temph               ; place a ASCII zero in MS digit location

           swapf     adc_templ,f             ; swap nibbles
           movlw     0x0F                    ; move literal into W
           andwf     adc_templ,f             ; mask out upper nibble
           andwf     adc_templ+1,f           ; mask out upper nibble

           movlw     D'10'                   ; move literal into W
           subwf     adc_templ,w             ; test byte
           btfsc     STATUS,C                ; was a borrow generated
           goto      add_37L                 ; no, so must be A - F
           movlw     0x30                    ; else it is 0 - 9
           addwf     adc_templ,f             ; compose ASCII byte
chk_lsd    movlw     D'10'                   ; move literal into W
           subwf     adc_templ+1,w           ; test value
           btfsc     STATUS,C                ; was a borrow generated
           goto      add_37L1                ; no, so must be A - F
           movlw     0x30                    ; else it is 0 - 9
           addwf     adc_templ+1,f           ; compose ASCII byte

chk_msd    movlw     D'10'                   ; move literal into W
           subwf     adc_temph+1,w           ; test byte
           btfsc     STATUS,C                ; was a borrow generated
           goto      add_37H                 ; no, so must be A - F
           movlw     0x30                    ; else it is 0 - 9
           addwf     adc_temph+1,f           ; compose ASCII byte
           goto      exit                    ; exit routine
```

```
add_37L     movlw     0x37                        ; move literal into W
            addwf     adc_templ,f                 ; compose ASCII character
            goto      chk_lsd                     ; check least significant digit
add_37L1    movlw     0x37                        ; move literal into W
            addwf     adc_templ+1,f               ; compose ASCII character
            goto      chk_msd                     ; check most significant digit

add_37H     movlw     0x37                        ; move literal into W
            addwf     adc_temph+1,f               ; compose ASCII character

exit        return                                ; return from subroutine

            END                                   ; directive ‘end of program’
```

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-786-7200 Fax: 480-786-7277
Technical Support: 480-786-7627
Web Address: http://www.microchip.com

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

Boston
Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508-480-9990 Fax: 508-480-8575

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

Dallas
Microchip Technology Inc.
4570 Westgrove Drive, Suite 160
Addison, TX 75248
Tel: 972-818-7423 Fax: 972-818-2924

Dayton
Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Detroit
Microchip Technology Inc.
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York
Microchip Technology Inc.
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: 408-436-7955

AMERICAS (continued)

Toronto
Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

Hong Kong
Microchip Asia Pacific
Unit 2101, Tower 2
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

Beijing
Microchip Technology, Beijing
Unit 915, 6 Chaoyangmen Bei Dajie
Dong Erhuan Road, Dongcheng District
New China Hong Kong Manhattan Building
Beijing 100027 PRC
Tel: 86-10-85282100 Fax: 86-10-85282104

India
Microchip Technology Inc.
India Liaison Office
No. 6, Legacy, Convent Road
Bangalore 560 025, India
Tel: 91-80-229-0061 Fax: 91-80-229-0062

Japan
Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa 222-0033 Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

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

Shanghai
Microchip Technology
RM 406 Shanghai Golden Bridge Bldg.
2077 Yan'an Road West, Hong Qiao District
Shanghai, PRC 200335
Tel: 86-21-6275-5700 Fax: 86 21-6275-5060

ASIA/PACIFIC (continued)

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan, R.O.C
Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5858 Fax: 44-118 921-5835

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

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

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 München, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

11/15/99

DNV Certification, Inc. USA
DNV MSC The Netherlands Accredited by the RvA
ISO 9001 / QS-9000 REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.