INTRODUCTION

Many of the embedded control systems designed today require some flavor of an 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 625sp's, 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.:

\[
LSB \text{ size} = \frac{V_{REF}}{2^{12}}
\]

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 (DOUT) and the chip select input signal (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 initiated with the high to low transition of CS/SHDN (active low). The chip select is generated by PORTA:<5> going 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 CS/SHDN is low, the device will output the conversion LSB first. If more clocks are provided to the device while 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 VREF input for conversion scaling.

\[ \text{Digital output code} = \frac{V_{IN} \times F.S.}{V_{REF}} \]

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 5VDC which approximately translates to a corresponding LSB size from 61μV to 1.22mV per bit.

\[ 1.22mV = \frac{5V_{DC}}{2^{12} \text{ bits}} \]

For this simple application, the MCP3201 voltage reference input is tied to 5VDC. 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 0VDC to 5VDC.

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 CS/SHDN is asserted. PortA:<5> is used as the 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 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 CS/SHDN is asserted. PortA:<5> is used as the 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 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
APPENDIX A:

 dear Listeners:nnnnn

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**  - Hi-Tech provided file
  **mcp3201.h**  - 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", 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 mode1
    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 initPorts( void )
{
    PORTA = 0b100000; // set PORTA data latches to initial state
    PORTB = 0x00; // set PORTB data latches to initial state
    PORTC = 0b01100000; // 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 = 0b01100000; // set PORTC pin direction
    TRISD = 0x00; // set PORTD pin direction
    TRISE = 0x00; // set PORTE pin direction
}
// 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
/* **************************************************************************
   * File: config67.h
   * Date: 06/30/99
   * File Version: 1.00
   *
   * **************************************************************************/ 

#endif 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 PWRT_OFF 0x3FFF
#define PWRT_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:

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 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 & _PWRT_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, adctempl ; 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 btfs 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
movf adc_result+1,w ; move adc_result MSB into W
movwf adc_temph ; save W into temp register
movf adc_result+1,f ; write SSPBUF to adc_result

return

; Display ADC data (ASCII and DECIMAL) to USART
Display_Data
banksel offset ; linker to select GPR bank
clf 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
btfas 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 SFR bank
btfas 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
btfas STATUS,C ; done with sending data
goto send_hex1 ; no, so send some more

banksel offset ; linker to select GPR bank
clf 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
btfs    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
           movf    FSR,w ; initialize FSR as pointer
           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
           btfs    STATUS,C ; done with sending data
           goto    send_dec1 ; no, so send some more

send_dec1 movlw  thous ; obtain variable address
           movf    FSR,w ; initialize FSR as pointer
           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
           btfs    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
btfs    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
btfs    TXSTA,TRMT ; test if TSR is empty
goto    $-1 ; no, so stay in loop
return

; 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
           decf    countlo,f ; decrement counter low
           btfss   STATUS,Z ; is result == 0
           goto    inner ; no, stay in loop
inner   decf    countio,f ; decrement counter low
           btfs    STATUS,Z ; is result == 0
           goto    outer ; else, start again
           decf    counthi,f ; decrement count high
           btfss   STATUS,Z ; is result == 0
           goto    outer ; no, so start again
           return

; 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
clrw TRISA ; set PORTA pin direction
clrw TRISB ; set PORTB pin direction
clrw TRISD ; set PORTD pin direction
clrw 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

#include <p16c67.inc>

GLOBAL Hex_Dec, thous
EXTERN adc_result

HEXDEC_VAR UDATA 0x30
thous RES 1
hunds RES 1
tens RES 1
ones RES 1

GLOBAL thous, hunds, tens, ones

; ****************** Subroutine begins here

HEXDEC CODE

Hex_Dec

banksel thous
clr thous
clr hunds
clr tens
clr ones

chk_thous
movlw 0x04
banksel adc_result+1
subwf adc_result+1,w
btfss STATUS,C
goto chk_hunds2
incf thous,f
movlw 0x04
subwf adc_result+1,f
movlw D'24'
btfss STATUS,C
incf adc_result+1,f
go to chk_thous

chk_hunds2
movlw 0x01
subwf adc_result+1,w
btfss STATUS,C
goto chk_hunds1
movlw D'2'
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
incf hunds ; 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, Z ; is adc_result LSB >= 10
incf tens ; else, increment tens
movlw D’10’ ; move literal into W
subwf adc_result, f ; reduce tens count by 10
movlw D’10’ ; move literal into W
subwf adc_result, f ; 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’
#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_templ,w ; move copy of adc_result MSB into W
movwf adc_templ+1 ; make copy ADC result MSB
movlw 0x30 ; move literal into W
movwf adc_templ ; 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
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_templ+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_templ+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'
Note the following details of the code protection feature on PICmicro® MCUs.

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable".
- Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our product.

If you have any further questions about this matter, please contact the local sales office nearest to you.

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, FilterLab, KEELOQ, microID, MPLAB, PIC, PICmicro, PICMASTER, PICSTART, PRO MATE, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

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

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.