PIC Microcontroller Data Acquisition System

EE3780 Final Project

Mark Panzer
Gang Feng
5-09-2005

Abstract

A data acquisition system was developed as a final project for the microcontroller design course at UW-Platteville. This system was developed to monitor power supplies and aid as a development tool for the design of a 200 W switch-mode power supply. A PIC microcontroller from Microchip was used as the heart of the data collection system. Internal analog to digital converters acquired data from an analog interface. The analog subsystem gathered data from temperature, voltage, and current sensors. Data was recorded through HyperTerminal in Windows. Once the data was gathered MS Excel was used to convert, scale, and plot the data.

Introduction

Measurement of the three basic quantities: voltage, temperature, and current can provide enough information to allow for debugging of almost any electrical circuit. During the development of a switch-mode power supply it was determined that some sort of data logging was necessary to protect the supply and determine if the supply was operating properly. Out of this need a data acquisition system was developed. The data acquisition system measures one channel of voltage from 0 to 20 volts, one channel of current from –50 to +50 amps, and two channels of temperature (one ambient and one load). This amount of data is enough to determine supply efficiency and temperature rise. With additional channels or faster sampling rates it would be possible to measure and calculate inrush current, supply stability, and transient response.

Implementation options

Two options were initially considered to implement the data logger. First, adding an outboard analog to digital converter (ADC) to the 8052 board used for development in class was considered. This approach had the advantage of allowing me to use a known good development system, software toolset, and the ability to get support from teachers and other students. Unfortunately, I had a difficult time finding multi-channel ADC converters featuring parallel interfaces to 8-bit busses that could resolve more than 8 bits. Most modern ADCs found used a serial interface. This was a problem on the 8052 board as the serial port was tied up for communication to the PC. Also fully developing a software-based serial protocol on a relatively slow processor like the 8052 could be difficult. Figure 1 shows the proposed block diagram for the 8052 based data acquisition system.

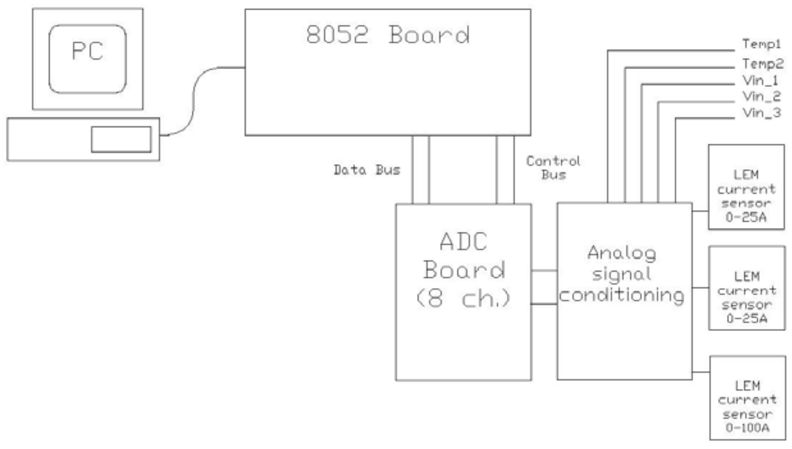


Figure 1, data acquisition system using 8052 board

The second option considered the use of an altogether different processor, the Microchip PIC controller. PICs are self-contained microcontrollers often including clock, I/O, and a host of peripherals on-chip. The great advantage seen by adopting the PIC was a chip with onboard analog to digital converters was available in a small 14-pin DIP package. In addition a serial port, and multiple timer/counters were available. A low-cost ($35) development kit is available from Microchip to try out any of the 14-pin series of micrcontrollers. Additionally, for the intended application a low-power small form factor device was a plus. Essentially all the PIC needs to create the system is the analog interface and a voltage regulator. Just the 8052 board requires 2 to 3 times the space of the board designed for the PIC controller and analog board. The downside of this approach hinged around learning a new assembly language for the PIC microcontroller and learning a new development environment and device programmer. Figure 2 shows the block diagram for the PIC data acquisition system.

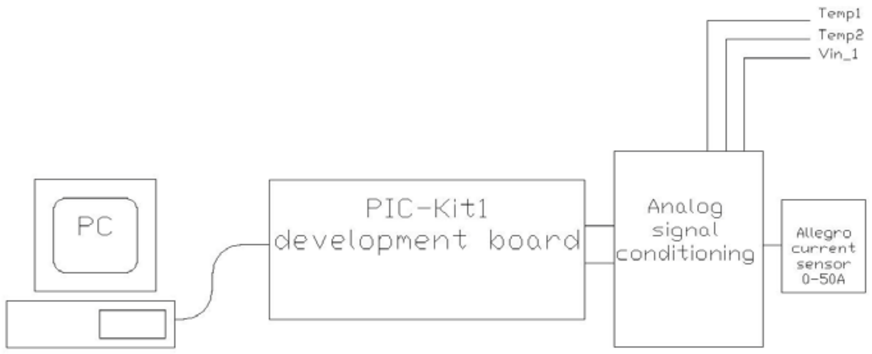

Temp1
Temp2
Vin_1
PC
PIC-Kit1 development board
Analog signal conditioning
Allegro current sensor 0-50A

Figure 2, PIC processor based data acquisition system

PIC selection and setup

There are literally hundreds of PIC microcontrollers to pick from. The programmer I had already purchased narrowed this selection down to 8 or 14 pin devices. At minimum 4 ADC channels, a UART, and one counter timer were needed. The first device found to meet these specifications was the PIC16F688. The ‘688 contains 8 channels of 10-bit AD converters, an enhanced UART, two timer counters, analog input comparison modules, an internal 32kHz to 8MHz clock, and flash program memory.

In order to use the PIC, settings for all the internal registers needed to be determined or calculated via the datasheet. The internal oscillator was used and set to 4MHz. Next, the serial port was configured. The enhanced UART (EUSART) has an internal baud rate generator (no external timer is needed). The EUSART was setup to communicate at 9600 bps, 1 stop bit, 8 data bits, and no parity. This seemed to be a common serial data rate that was easily achieved with little error in bit-rate on the PIC (about 0.16%). The commonly used baud rates were all available in tables in the ‘688 datasheet. After the serial port was configured the analog to digital converter was set-up. A conversion time of 4.0us as dictated by the datasheet was selected. Then, registers were set up to use the positive supply as the ADC reference voltage along with selecting the location of the most significant bit of the result. Finally, a timer was set up to control the rate of data sampling. The timer values were set to allow for maximum delay that turned out to be around ¾ of a second. The timer overflow bit was checked via polling. This approach was used because exact time intervals were not needed and quick response was not necessary.

Microchip provides an integrated development environment (IDE) called MPLAB for coding, compiling, setting up, and controlling programmers for the PIC series of microcontrollers. Included in the IDE is a debugger and compiler. The debugger worked well until additional peripherals were initialized and used which then caused the debugger to crash the IDE. Thus, all further testing needed to be conducted on actual hardware. A simple USB powered programmer interfaced to the IDE and reprogrammed the flash program memory in PIC controllers.

Microcontroller Firmware

After configuration of peripherals the microcontroller firmware consisted of a simple loop that acquired samples, converted them to a format acceptable to HyperTerminal, and echoed them to the serial port. Code is attached to the end of this document in listing 1. Peripherals were first initialized as described in the previous section. Following this a timer set to approximately one second would overflow triggering a capture event. The capture event consists of setting ADC registers, then waiting for the conversion to be completed. These events are repeated four times to cover all the input channels. The results are then converted to a three digit octal number via shifts and bit masks. Only 8 bits of the 10-bit result are converted to octal, as it appeared the lower value bits only added noise to the acquired signal. Finally, the converted values were output on the serial port to HyperTerminal in ASCII format delimited with commas. Figure 3 shows the block diagram for the microcontroller firmware.

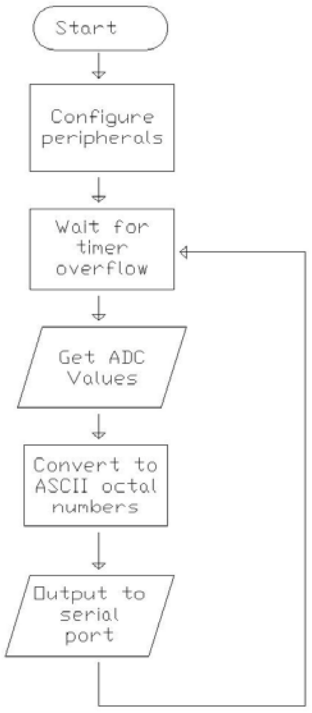


Figure 3, A block diagram of the PIC's firmware

Hardware

The high integration of the PIC controller leads to a very simple hardware solution. On the digital side the PIC controller is connected to a MAX232, RS-232 to logic-level converter. A 5 V power supply and some supply decoupling capacitors round out the digital section of the hardware. Figure 4 shows the implementation of the digital board that was constructed on the PICKit-1 development board.

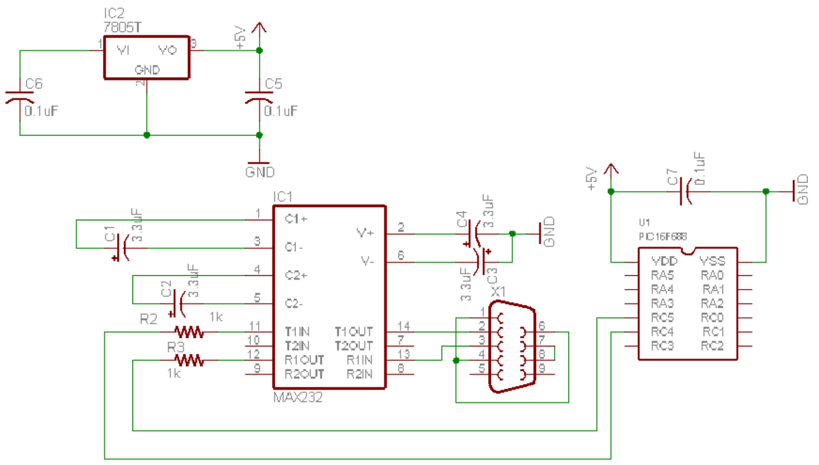


Figure 4, the digital section of the data acquisition system

Highly integrated sensors reduced the difficulty of implementing an analog interface board. The LM35 temperature sensor features a conditioned output with a 10mV/C output slope. It was decided to use the LM35's output directly, without amplification, with slightly reduced resolution. A hall-effect current sensor the ASC750SCA-050 made by Allegro was chosen as an easy integrated solution for current sensing. The current sensor is capable of resolving –50 to +50 A and outputs a 0 to 5 V signal corresponding to the current through the device. In the original application it was expected to see load currents up to 30 A. However, when demonstrating the device it was not possible to find power supplies capable of supplying more than 2.5A, thus the captured waveforms appear very noisy due to the small currents measured. Figure 5 shows the schematic of the analog board.

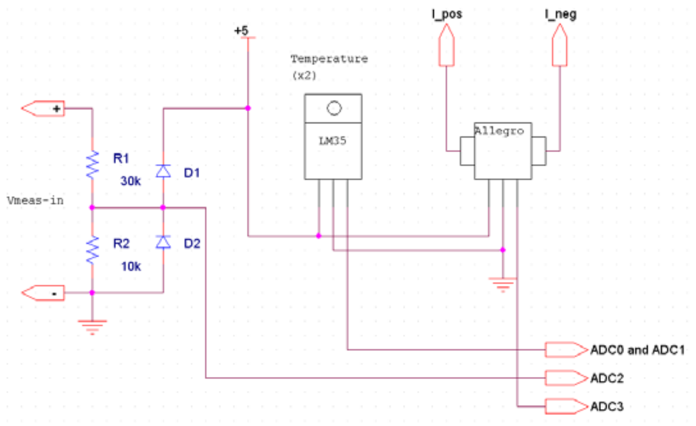


Figure 5, analog interface section of the data acquisition system

The physical hardware is shown in photo 1. Two LM35 temperature sensors are attached via twisted cable, enabling them to be clamped onto heatsinks to measure power supply temperatures. A set of binding posts is provided for voltage measurement and current measurement input. The 9 pin serial port hangs off the left side of the development board (black PCB on top). Unregulated (9 to 37 VDC) DC input is supplied via two wires exiting the back of the board.

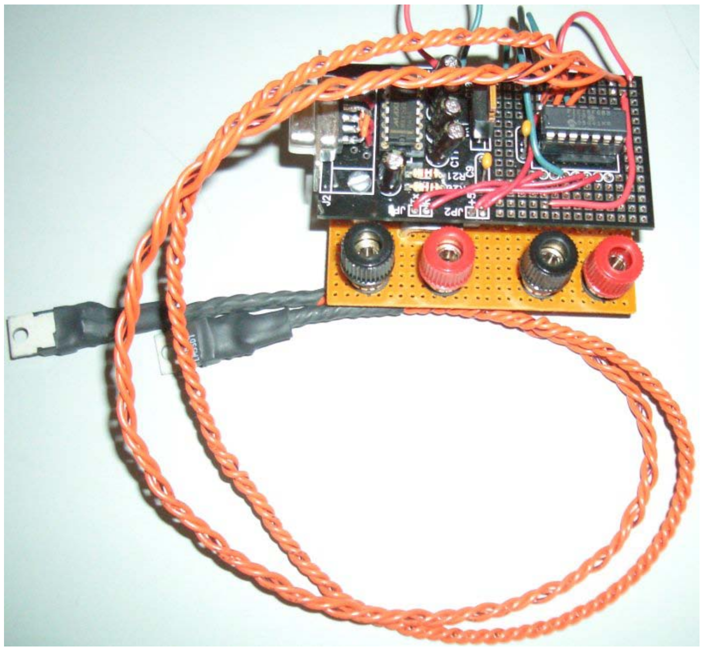


Photo 1, physical realization of the data acquisition system

Results

To test the data acquisition system an unregulated power supply was used. Real power supplies have internal resistance that can be demonstrated if the load of the supply is varied. The circuit in figure 6 was built to show the effect of loading the supply under various conditions.

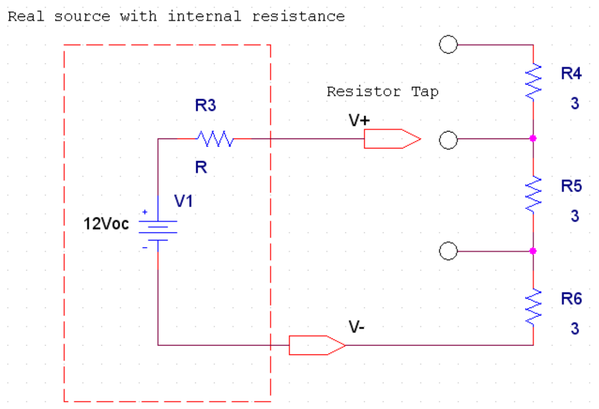
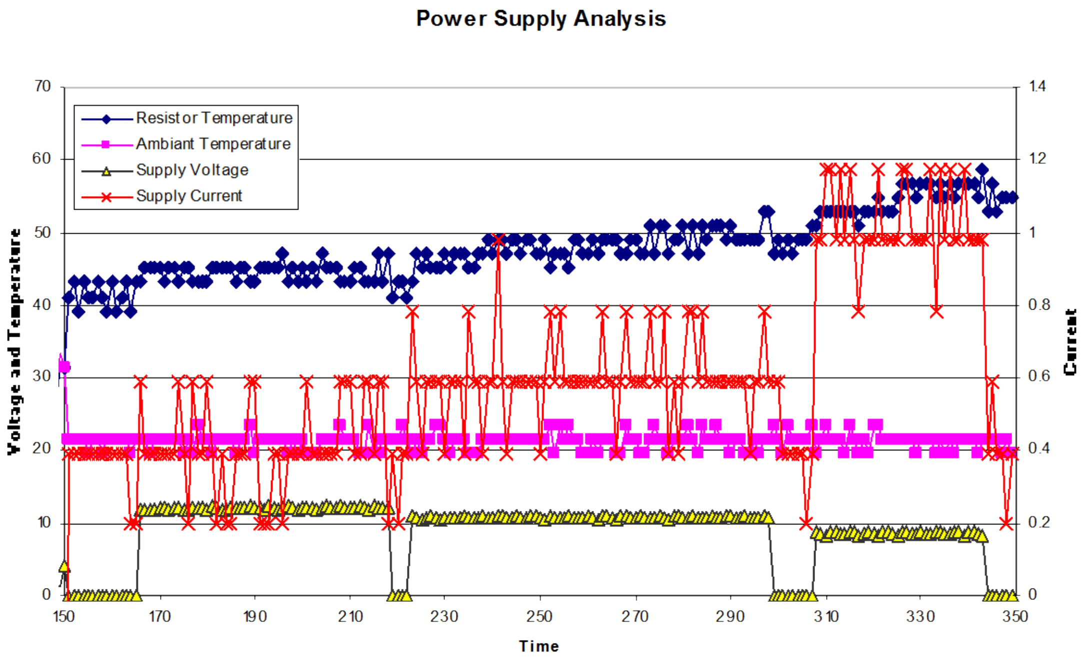


Figure 6, an unregulated power supply loaded with various power resistors

The output voltage, current, ambient temperature, and temperature of the load resistor were all monitored via the data acquisition system. After logging the octal numbers to a text file via HyperTerminal, the data was scaled and plotted in MS Excel.



Plot 1, result of power supply testing, showing voltage, current, and two temperatures

Plot 1 shows the results of loading the supply with three different values of load resistors. As expected the supply voltage sags with decreasing resistance. The output current changes from 400mA to about 1A over the three different resistors. Finally, the temperature of the load resistor peaks at about 55 degrees Celsius. This simple application example shows the utility and some of the capabilities of the data acquisition system.

Conclusions and Future Work

The data acquisition system proved to be a successful, yet challenging final project. Choosing a different microcontroller platform on which to base the project made hardware design much simpler. However, using the PIC controller greatly increased the difficulty of software design. A new assembly language was learned to take advantage of the PIC. Additionally, it was necessary to learn the development environment, compiler, and device programmer. Hardware design went relatively smoothly. Highly integrated sensors made the analog interface a snap. For some sensors such as the current sensor and to a lesser degree the temperature sensors the data acquired was of low resolution. Ideally, a higher resolution ADC converter would be needed to take better data, or a variable gain amplifier could be used. The primary problem with the current sensor was because it was intended for a 0 to 50 A range. In the original application this high current sensor was fine. Under testing, low-current supplies were used that showed the low resolution of the ADC. This combined with using a very small region of the current sensors range resulted in noisy captured data.

This data acquisition system, while functional, could be improved. Primarily, the improvements are centered on the lack of resolution of the ADC in the PIC controller and the need for better PC interface software. Other microcontrollers such as the MSC series from Texas Instruments have on-board ADC converters with up to 24 bits of resolution. These microcontrollers also feature programmable gain amplifiers, which effectively increase the dynamic range of the ADC and thus further increase resolution. An improved voltage measurement input is also desired. Currently, the system can only measure positive ground referenced voltages. With a true differential amplifier input, positive and negative voltages could be read without the fear of shorting a measurement channel to ground. On the software side two major improvements are warranted. First, the capability to display real-time values of the captured data would greatly increase the utility of the data acquisition system. Next, an integrated capture and display program would greatly simplify data capture. The current system requires capturing data via HyperTerminal to a text file. This text file then is updated in an excel spreadsheet. Excel's update feature works most of the time, however, trouble was occasionally experienced resulting in Excel crashing. The current solution is far from a one-click data capture. With dedicated software it is possible that a single click would enable a user to view live data and concurrently save the trend data to a file.

Listing 1, PIC16F688 assembly code for the data acquisition system

```
        list            p=16f688                ; list directive to define processor
        #include        <P16F688.inc>           ; processor specific variable definitions

        __CONFIG    _CP_OFF & _CPD_OFF & _BOD_OFF & _PWRTE_ON & _WDT_OFF &
_INTRC_OSC_NOCLKOUT & _MCLRE_OFF & _FCMEN_OFF & _IESO_OFF


; '__CONFIG' directive is used to embed configuration data within .asm file.
; The labels following the directive are located in the respective .inc file.
; See respective data sheet for additional information on configuration word.






;***** VARIABLE DEFINITIONS
w_temp          EQU     0x71            ; variable used for context saving
status_temp     EQU     0x72            ; variable used for context saving
pclath_temp     EQU     0x73            ; variable used for context saving

ADCH0           EQU B'00000001'         ;setup ch 0, Vdd for Vref, and left justified result
ADCH1           EQU B'00000101'         ;set to Ch1 same settings as above
ADCH2           EQU B'00001001'         ;set to Ch2 same settings as above
ADCH3           EQU B'00001101'         ;set to Ch3 same settings as above
COUNTER         EQU 0x60                ;counter variable
TXPREBUF        EQU 0x61                ;A buffer for the serial transmit buffer (a prebuf)
OCTTOVERT       EQU 0x20                ;stores the a/d value when it's being converted to
octal

;**********************************************************************
        ORG             0x0             ; processor reset vector
        goto            main            ; go to beginning of program

main

;***********************************************************
;Setup For 9600 bps tx serial interface
;***********************************************************
        bsf             STATUS, RP0     ;bank 1
        movlw B'00111111'               ;setup RC0-5 as inputs for serial i/o and A/D
        movwf TRISC
        bcf             STATUS, RP0     ;bank 0
        bsf             RCSTA, SPEN     ;serial port enable
        bsf             BAUDCTL, BRG16  ;use 16 bit baudrate generator
        bcf             TXSTA, BRGH     ;use low baudrate speed
        movlw d'25'
        movwf SPBRG                     ;setup baudrate generator
        bcf             TXSTA, SYNC     ;asynchronous serial i/o
        bsf             TXSTA, TXEN     ;enable transmission

;***********************************************************
;Setup A/D converter
;***********************************************************
        bsf STATUS, RP0                 ;bank 1
        movlw B'00011111'               ;Setup all of port A as inputs for A/D
        movwf TRISA
        movlw 0xFF
        movwf ANSEL                     ;Enable all available Analog channels
        movlw B'01010000'
        movwf ADCON1                    ;Set 4uS conversion time w/ 4MHz internal clk
        bcf     STATUS, RP0             ;bank 0
        movlw ADCH0
        movwf ADCON0                    ;AD: right justified result, ch 0, VDD ref. voltage
        call WaitForAD

;***********************************************************
```

```
;Setup Timer1 for 1/10 second wait period
;************************************************************
       movlw B'00110001'
       movwf T1CON                   ;load timer1 config register

GoAgain:
       movlw 0x01                    ;set to max for approx 1 sec delay
       movwf TMR1L
       movlw 0x00
       movwf TMR1H
       ;bsf T1CON, TMR1ON            ;start timer
       bcf PIR1, TMR1IF              ;clear overflow flag

;This is the delay between sample periods from the analog inputs
Wait:
       btfss PIR1, TMR1IF            ;check if timer has overflowed
       goto Wait

       call GetADValues

; loop through 4 values here and print them out
       movlw 0x4                     ;loop 4 times
       movwf COUNTER
       movlw 0x21                    ;point to first a/d value
       movwf FSR                     ;data pointer

       OutLoop:
              movf INDF, W           ;getting indirect data
              movwf OCTTOVERT        ;store a/d value we need to convert

              call TO_OCTAL          ;convert currently selected A/D Channel to octal

              movf 0x30, W           ;output all bytes
              movwf TXPREBUF
              call SENDBYTE

              movf 0x31, W
              movwf TXPREBUF
              call SENDBYTE

              movf 0x32, W
              movwf TXPREBUF
              call SENDBYTE

              movlw ','      ;do checking on this one to see if we're at the last byte
              movwf TXPREBUF
              call SENDBYTE

              incf FSR               ;goto next a/d value
              decfsz COUNTER         ;is counter zero yet? if so skip goto
              goto OutLoop
; end of above loop

       movlw 0x0A                    ;Line Feed (newline, hopefully)
       movwf TXPREBUF
       call SENDBYTE

       ;movlw 0x0D                   ;carridge return
       ;movwf TXPREBUF
       ;call SENDBYTE

       goto GoAgain                  ;Reset the timer for another go around


;************************************************************
;Subroutines
;************************************************************

WaitForAD:                           ;wait 44us to init, 176 instruction cycles
       movlw D'176'          ;really only need 1/2 this many cycles due to test and set
       movwf COUNTER
```

```
FourFourWait:
        decf COUNTER
        bnz FourFourWait              ;keep decrementing until we reach 0

        return

;get a/d values for 4 ch
GetADValues:
        movlw 0x21                    ;start of free ram
        movwf FSR                     ;setup indirect addressing register

        movlw ADCH0                   ;goto channel 0
        movwf ADCON0
        bsf ADCON0, GO                ;start conversion
        btfsc ADCON0, GO              ;conversion done?
        goto $-1                      ;keep looping
        movf ADRESH, W                ;get high byte into acc W
        movwf INDF                    ;store adresh to 21h
        incf FSR                      ;move ptr to 22h

        call WaitForAD

        movlw ADCH1                   ;goto channel 1
        movwf ADCON0
        bsf ADCON0, GO                ;start conversion
        btfsc ADCON0, GO              ;conversion done?
        goto $-1                      ;keep looping
        movf ADRESH, W                ;get high byte into acc W
        movwf INDF                    ;store adresh to 22h
        incf FSR                      ;move ptr to 23h

        call WaitForAD

        movlw ADCH2                   ;goto channel 2
        movwf ADCON0
        bsf ADCON0, GO                ;start conversion
        btfsc ADCON0, GO              ;conversion done?
        goto $-1                      ;keep looping
        movf ADRESH, W                ;get high byte into acc W
        movwf INDF                    ;store adresh to 23h
        incf FSR                      ;move ptr to 24h

        call WaitForAD

        movlw ADCH3                   ;goto channel 3
        movwf ADCON0
        bsf ADCON0, GO                ;start conversion
        btfsc ADCON0, GO              ;conversion done?
        goto $-1                      ;keep looping
        movf ADRESH, W                ;get high byte into acc W
        movwf INDF                    ;store adresh to 24h


        return

;convert to 3 octal bytes in ASCII
;stores bytes in 0x30 to 0x32, w/ lsb in 0x32 msb in 0x30
;input value stored in 0x20, ie OCTTOVERT
TO_OCTAL:
        clrc    ;clear carry
        movlw B'00000111'             ;bit mask field
        andwf OCTTOVERT, W            ;and with ADRESH
        addlw 0x30                    ;make this into an ASCII number
        movwf 0x32                    ;store result away
;octal byte 2
        movlw B'00111000'             ;bit mask field
        andwf OCTTOVERT, W            ;get 2nd set of 3 bits
        movwf 0x31                    ;place back into file
        rrf 0x31                      ;rotate to LSB, need 3x
        rrf 0x31
        rrf 0x31
        movf 0x31, W                    ;now get it back
        addlw 0x30                      ;and add 30h to make a ascii number
        movwf 0x31                      ;store again
;octal byte 3
        clrc    ;clear carry
        movlw B'11000000'               ;bit mask field
        andwf OCTTOVERT, W              ;get last 2 bits
        movwf 0x30
        rrf 0x30
        rrf 0x30
        rrf 0x30
        rrf 0x30
        rrf 0x30
        rrf 0x30
        ;btfsc 0x21, 0                  ;check the LSB value from the last byte
        ;bsf 0x31, 2
        movf 0x30, W                    ;get value back
        addlw 0x30                      ;convert to ASCII
        movwf 0x30                      ;store it

        return

;transmit data on serial port
SENDBYTE:
        btfss TXSTA, TRMT               ;check tx buffer ready bit
        goto SENDBYTE                   ;try again, keep polling

        movf TXPREBUF, W                ;get data from pre-buffer
        movwf TXREG                     ;move data to tx register and start transmission

        return

        ORG     0x2100                  ; data EEPROM location
        DE      1,2,3,4         ; define first four EEPROM locations as 1, 2, 3, and 4


        END                              ; directive 'end of program'
```