CARE AND FEEDING OF THE PIC16C74 AND ITS PERIPHERALS

By: Robert Angelo

The PIC16C74 is one of the latest mid-range microcontrollers from Microchip Technology Inc. In this article we will be addressing a few of the new features and peripherals of this new part. The main focus will be on the A/D (Analog-to-Digital) Converter, the SCI-Serial Communication Interface, and the PWM-Pulse Width Modulator. Our intention is to give you a small program that initializes these peripherals as well as exercise them. A schematic is provided. The PICDEM-2 board from Microchip will run this program. The second trimpot does not exist on the PICDEM-2 board, so the second A/D value may float around. The second trimpot is only used to show a method of changing A/D input pins. If you are using the PICDEM-2 board, then the LED and a current limiting resistor must be connected to the PWM output. When the program is run, the RS-232 terminal will display two A/D values. The brightness of an LED is adjusted using pulse width modulation. The duty cycle is determined by the trimpot setting.

Assumptions

Although dangerous, sometimes we need to make assumptions. For this discussion on the PIC16C74, let us agree that RA0 and RA1 will be connected through a series resistor to the wipers on two potentiometers with the other ends connecting across VDD and ground (see schematic). The oscillator clock will be 4 MHz. First we'll read an A/D input, send its result out the serial port (to be displayed on a PC terminal program), and then switch to the next channel. We will adjust the PWM output pulse width to match the first potentiometer. Each time we are ready to begin a new sequence we will first send a pair of sync bytes to signal the receiving processor. To simplify our discussion, we will forgo using interrupts and we will do this in a polled fashion. The watch dog timer is disabled for this program.

To ensure there are no surprises, it is a good idea to initialize every special function register (SFR) and data register to some known value prior to use.

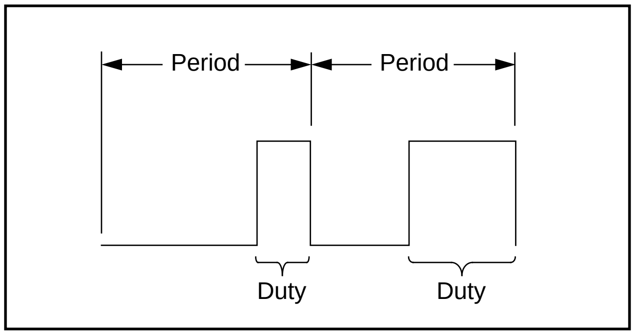
A/D Converter Mysteries

The A/D converter and its eight input channels will be our first topic. Setting up the A/D converter involves two special function registers:

- ADCON0
- ADCON1

In the program included with this article is a code segment `initad` that sets up the A/D. ADCON0 is the work horse register for this peripheral. This register is used to select the conversion clock frequency and channel. This register is also where we signal the start of a conversion and detect the completion of a conversion. ADCON1 has only one purpose in life for this part, and that is A/D port configuration. When ADCON1 is used it does not override the TRISA register controls. The TRISA register must be set up. Once these registers are set up, all the program has to do is select the desired pin and set the GO/DONE bit in ADCON0. The program then waits for the conversion complete bit, GO/DONE, to be cleared by hardware. Then the ADRES (A/D conversion result register) register is read. The value from the first pot's conversion is then used to adjust the PWM pulse width thereby adjusting the LED brightness.

FIGURE 1: PWM PULSE WIDTH



Pulse Width Modulation (PWM)

The PORTC-1 pin is used as the PWM output. The registers that need to be set up for this PWM operation are:

- TRISC
- T2CON
- CCPR2L
- PR2
- CCP2CON.

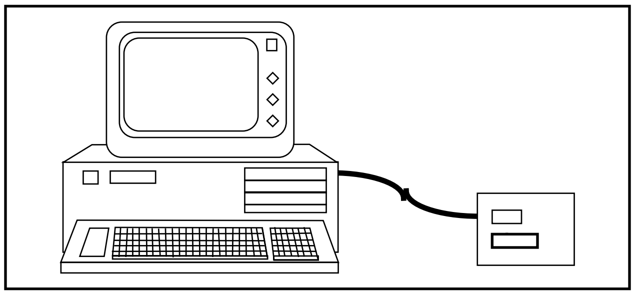
The code `initpwm` is an example of what might be done to initialize the PWM module. TRISC was cleared earlier thus setting PORTC as output. By writing a "4" to the T2CON register, we will set the prescaler equal to 0 and select TIMER2 operation. Writing a 0fh to the CCP2CON register selects PWM mode and standard resolution. The 0fh written to the CCPR2L register sets the high period to a low value initially. Setting the PR2 register to ffh allows the CCPR2L value (from the A/D converter result) to approach a 100% duty cycle. Now we can control the brightness of the LED attached to this pin by adjusting the pot on pin RA0 and writing the A/D result to the CCPR2L register as already described earlier.

SCI

The Serial Communications Interface Module is our RS-232 communications channel. We will configure the SCI as an asynchronous full duplex serial port. This is done with the routine at `initsci` in the program provided. There are a few fine points to remember relative to this peripheral. The baud rate is determined by a dedicated eight bit baud rate generator and can be used to derive standard baud rate frequencies from the oscillator. Since we are not using interrupts, there are only five registers to deal with:

- RCSTA - receiver status
- TXSTA - transmitter status
- TXREG - transmit buffer
- RCREG - receive buffer
- SPBRG - to set the baud rate generator

FIGURE 2: SERIAL COMMUNICATIONS INTERFACE MODULE



First global interrupts are disabled. The `initsci` code does the serial port setup and the `sendat` code handles the actual sending of the data.

The SCI is setup for 2400 baud, 8 data bits and 1 stop bit with no parity. A terminal program, such as TERMINAL in Windows“, set to the same settings can be used to see our output. If you use the Windows terminal program, then set the communications parameters to 2400 baud, 8 data bits, 1 stop bit, no parity and hardware handshake.

Tying The Pieces Together

The main loop for getting the process running and restarting it again is `mloop`. The `adcnvrt` routine handles port pin selection and actual conversion control. The `dopwm` routine handles updating the PWM duty cycle register CCPR2L. The routine `sendat` checks transmit ready status and loads the transmit buffer when the status reports ready. You will notice there is no error recovery routine. It is up to the user to determine.

Here is what the program will do:

Once all peripherals have been initialized two sync bytes "< >" are sent to the terminal. The A/D conversion results are then sent and the LED brightness is adjusted to match the RA0 trimpot setting. To simplify displaying A/D values, only the highest nibble is used, and thirty is added to put it into an ASCII range.

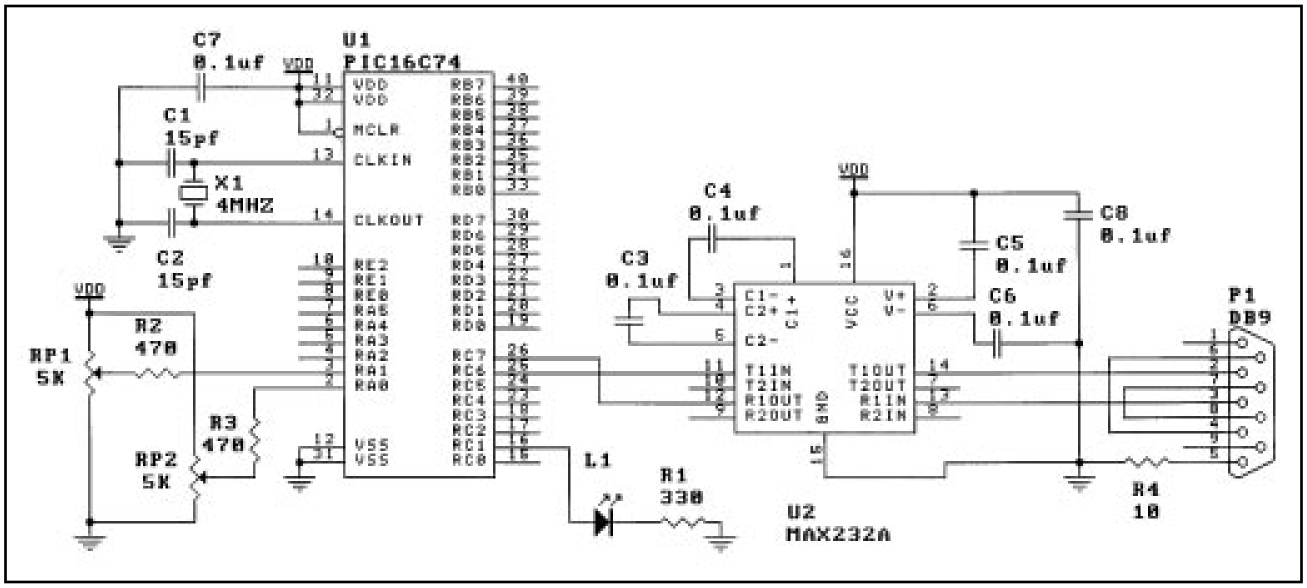
Care and Feeding of PIC16C74 Source Code

Note: This source code can be downloaded from our BBS from the "PICTIPS" library as filenameC74feed.ZIP.

```
LIST P=16C74
INCLUDE P16CXX.INC              ;new include file that comes with MPASM (on BBS)
Adcnt   equ     20h             ;a/d converter pin count register
Adcntw  equ     21h             ;a/d converter pin work register
Temp    equ     22h             ;temporary data holding register seems we always need one
        org     0
        goto    init            ;go to where our code really begins
        org     5h              ;begin program above interrupt service vector address
init    bcf     INTCON,7        ;make sure we don't get interrupted
        clrf    PORTA           ;don't rely on anything, set port latches where you want them
        clrf    PORTB
        clrf    PORTC
        clrf    PORTD
        clrf    PORTE
        clrf    Adcnt           ;clear RAM registers we will be using
        clrf    Adcntw
        clrf    Temp
        bsf     STATUS,RP0      ;switch to page 1 to access trisX registers
        clrf    TRISB           ;set all ports outputs
        clrf    TRISC           ;just for this program to minimize current
        clrf    TRISD           ; and prevent pins from floating
        clrf    TRISE
        movlw   0Bh
        movwf   TRISA           ;set analog inputs as inputs, the rest as outputs
        bcf     STATUS,RP0      ;
initad  movlw   0C1h            ;Internal RC A/D clock, input channel 0 , A/D on
        movwf   ADCON0          ;(user must wait for specified period before sampling)
        bsf     STATUS,RP0      ;select page 1 of the SFRs
        movlw   4
        movwf   ADCON1          ;setup a/d inputs on RA0, RA1 and RA3 with Vref = Vdd
                                ;we are still in page 1 of the SFRs
initsci movlw    19h            ;setup 2400 baud
        movwf   SPBRG
        movlw   20h             ;setup for async operations
        movwf   TXSTA
        bcf     STATUS,RP0      ;back to page 0 for a moment
        movlw   80h             ;enable serial port operations and the associated pins
        movwf   RCSTA
        clrf    TXREG           ;clear our serial port buffers for start up
        clrf    RCREG
initpwm movlw   4h              ;setup T2CON with prescaler = 0 and timer2 on
        movwf   T2CON
        movlw   0fh             ;setup capture/compare to PWM mode standard resolution
        movwf   CCP2CON
        movlw   0fh             ;set compare register to half for now
        movwf   CCPR2L
        bsf     STATUS,RP0      ;select page 1 for the PR2 register
        movlw   0ffh
        movwf   PR2
        bcf     STATUS,RP0
mloop   movlw   0dh             ;send a carriage return character
        call    sendat
        movlw   3ch             ;begin main loop for data gathering and serial transmission
        call    sendat          ;these are our sync bytes to tell receiving micro a new
        movlw   3eh             ;sequence is beginning
        call    sendat
        clrf    Adcnt           ;our first time through select AN0 pin
adloop  call    adcnvrt         ;go do a conversion and  send the result
        movf    Adcnt,0         ;get Adcnt into the W register
        xorlw   2               ;(# determines number of AD inputs to scan)
        btfss   STATUS,2        ;have we sampled all of the pins yet?
        goto    dopwm           ;go adjust the PWM output
        goto    mloop           ;all done go do it again
```

```
adcnvrt movf    Adcnt,0         ;get a/d count value
        movwf   Adcntw          ;put in work register
        bcf     STATUS,0        ;clear the carry flag for the upcoming rotate operations
        rlf     Adcntw,1        ;rotate left and leave the number in adcntw
        rlf     Adcntw,1        ;need to do it three times to put the count in the right
        rlf     Adcntw,1        ;position to select the next A/D pin
        movlw   0C1h            ;load the initial ADCON0 value excepting channel select
        iorwf   Adcntw,0        ;set the pin select bits we want
        movwf   ADCON0          ;set the new ADCON0 with new channels selected
        call    wait            ;wait about twenty micro seconds
        bsf     ADCON0,2        ;start conversion
        incf    Adcnt           ;increment pin counter register
adwait  btfsc   ADCON0,2        ;wait for conversion done
        goto    adwait          ;not done yet
        swapf   ADRES,0         ;conversion done, swap result nibbles into W register
        andlw   0Fh             ;mask off the upper nibble to limit number to an ascii range
        addlw   30h             ;convert to ascii character to make it visible on terminal
sendat  bsf     STATUS,RP0      ;select page one
        btfss   TXSTA,1         ;check transmit status ready to send
        goto    sendat          ;if not ready go try again
        bcf     STATUS,RP0      ;back to page 0
        movwf   TXREG           ;transmit buffer empty send new data
        return
dopwm   movf    ADRES,0         ;get the a/d conversion value
        movwf   CCPR2L          ;put the value into the PWM duty cycle register
        goto    adloop
wait    movlw   08h             ;do a wait loop of before using a/d converter
        movwf   Temp
w1      decfsz  Temp
        goto    w1
        return
        end                     ;end of program
```

FIGURE 3: PIC16C74 DEMO SCHEMATIC



© 1996 Microchip Technology Inc.