Notes on the PIC code for the GPS-disciplined VCXO vsn 1.28

The code starts with an initialize section that sets up the interrupt vector, sets the port I/O directions, initializes variables, and finally falls into a infinite loop waiting for the 1 pps interrupts from the gps receiver.

At each 1 sec interrupt, the service routine toggles the "heartbeat" LED on or off to indicate that the code is running, and decrements an interrupt counter. After TCOUNT interrupts (normally every 30 sec), the phase-measuring counter is read and then reset to zero(1). The counter, which contains the total number of gated 24 MHz clock pulses that occurred during the previous 30 seconds, indicates the relative phase of the VCXO and the GPS time pulses.

The code then reads certain porta and portb bits that the user sets by DIP switch to specify operating mode, filter time constant, polarity, etc. The raw binary phase reading is converted to decimal (BCD), translated to ASCII, and transmitted out one of the serial ports at 9600 baud. A test is made to insure that the new phase data is reasonably consistent with previous data. If it isn't, the previous value is used instead (up to a point). The phase difference is converted to an error signal by subtracting from it a desired phase "setpoint", chosen to be 1024.

Depending upon the DIP switch settings, an IIR or a simple type 1 filter is used to compute a new DAC setting. The IIR filter requires high mathematical precision to avoid round-off error for long (many hour) integration times. Calculations are done using 16-bit, 24-bit or 40-bit precision signed numbers, as appropriate(2).

The DAC is set(3) using a serial line protocol and finally, the new DAC setting and the DIP switch setting are transmitted out the ASCII port(4), and the CPU returns to its infinite wait loop.

Footnotes

(1) The code contains support for two separate types of phase counter: the external 74HCT4520, 74HCT404, 74HCT166 arrangement described in the QST article, and a version that eliminates these parts by using the PIC's internal prescaler and counter. I developed the latter version after the article was submitted to QST (however, it uses a sort of kludgy method of reading out the PIC prescaler, which is required to count near its upper frequency limit, and it might not be completely reliable).

(2) The last 3rd of the source listing contains the routines I wrote to perform various mathematical operations on these numbers and to transfer them between the three main (software implemented) 40-bit registers, which I labelled A, B, and C. The 40-bit registers are

created by using 5 consecutive 8-bit PIC storage locations. On some (expensive) CPUs these routines would be replaced by only one or two machine instructions.

Fortunately, there is plenty of time to perform the many instructions required to do precise math on an 8-bit processor. The code executes in just a few msec, and most of this time is spent stalling while the serial port transmits data at 9600 baud. Altogether, less than 0.1% of the available CPU capacity is used!

(3) The code implements a phase-locked loop by sending the filtered and scaled phase error value to the DAC to discipline the VCXO phase. If you are interested in the mathematics of the filtering calculation and the design of stable closed-loop systems I refer you to the references listed in the article and other references on control system design ("Automatic Control Systems," B.C. Kuo, Prentice Hall is a widely-used to textbook). It's an interesting subject and has applications in many different fields.

(4) The ASCII port output is invaluable for debugging the code since intermediate steps in the calculations can be examined easily by attaching a PC running a terminal capture program. You will notice some "commented out" print statements in the code that were used for that purpose I didn't make much use of the Microchip simulator for testing code, preferring to program the code into a UV-erasable 16C73A/JW chip and just try it out. A logic probe and/or a scope is handy when things don't make sense.

PIC programming is described in a new book "Programming and Customizing the PIC Microcontroller," Myke Predko, McGraw-Hill 1998, which I noticed in a bookstore recently. Athough I haven't looked at it carefully it appeared to be quite helpful. Of course, the Microchip PIC16C73X Data Sheet (actually a 300 page booklet) is invaluable. It's available from DigiKey or your Microchip rep.

```
      TITLE "Controller for OCXO Disipline by GPS - Brooks Shera, W5OJM, Nov.
97"

;Copyright (c) 1998, Brooks Shera.  All rights reserved.  This program
;may be used only for non-commercial purposes, and carries no warranty
;of any kind, including any implied warranty of fitness for any particular
;purpose

      LIST n=58, p=PIC16c73
      errorlevel  1
      include P16C73.INC
      __CONFIG _XT_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF

;Can select either PIC internal counter or external counters as phase detector
;by changing the routine called by int_srv.  The former eliminates several
chips.
;IIR filter parameters:Ts  = 30 sec F2 = 64, F1 and Kcpu determined by PORTC lo
bits
;PORTC 3 lo bits=0 selects setup mode, =1 type 1 filter (error signal direct to
dac)
;
;Note:*** check that computed goto's (...addwf PCL,1...) do NOT cross 0xFF prog.
ctr.
;     bounderies since only the lo 8 bits of PCL are changed - no carry occurs
***
;
;I/O pins in use:
;     PORTA 1,2,3 freq low (glitch) LED, GPS heart beat, freq high (unlock) LED
;           4,5   prescaler & counter input, output pin to toggle prescaler
;     PORTB 0     ext interrupt 1pps in
;           1     VCO polarity input (1 means inc V => inc F)
;           2     Open/Closed loop switch [Open (hi) => Open loop]
;           3     74HC166 shift/load control out
;           4     counter reset line out
;           5     DAC data out
;           6     DAC clock out
;           7     DAC latch out
;     PORTC 0,1,2 user inputs
;           3,4   SPI clock & data in
;           6     USART asynch xmt output
;
;DIP switch       1     2     3     4     5     6     7     8
;     port: RC0   RC1   RC2   RB1   RB2   nc    nc    nc
;     use:  setup,type1,iir filters VCO+/-      O/C loop
;
;*****************************************************************
;define storage locations
;*****************************************************************

W_TEMP            equ   0x20  ;;temp storage for w reg in int.
STAT_TEMP   equ   0x21  ;ditto for status reg
CHAR        equ   0x22  ;the next char to be xmitted

      CBLOCK      0x23        ;storage used by binary-to-BCD routine
            Lbyte,Hbyte
            R2, R1, R0  ;must stay in sequence R2, R1, R0
```

```
            count, wtmp
            Ldata,Hdata ;storage for data from counter
            tick        ;count 1 pps interrupts
            tsecs       ;seconds to count - set according to usr
            toc         ;storage for minute counter for IIR entry
            minutes           ;minutes to count - set according to usr
            Ha, Ma, La, Sa, SSa, Hb, Mb, Lb, Sb, SSb, Hc, Mc, Lc, Sc, SSc ;math
regs
            Hsm, Msm, Lsm     ;storage for smoothed counter data
            Hfi, Lfi    ;storage for filter input (phase error)
            Hfil, Lfil  ;storage for previous filter input
            Hfol, Mfol, Lfol, Sfol, SSfol;storage for previous filter output
            Hpdat,Lpdat ;storage for previous phase data input
            Hsp, Lsp    ;storage for phase set point
            Hdac, Ldac  ;storage for DAC current setting
            FrstPss           ;first pass flag to init interrupt code
            negFlag           ;used by multiply routine to indicate neg. value
            cLoBits           ;used by iir to set Kcpu - 3 lowest bits of PORTC
            pLoBits           ;used by adjust filter parameters - previous 3
lowest bits
            glcctr            ;used by dGlitch to limit multiple entries
            ;portAr           ;image of PORTA in a register, to permit bcf, bsf
to porta
            temp        ;temporary storage
            temp2       ;more temp storage
      ENDC

;Constants

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

;set interrupt vector and start of code
      org   0           ;initialize code
      goto  start
      org   4           ;interrupt routine
      goto  int_srv

;initialize bank 0 ports and control registers


start clrf  PORTA       ;clear port output latches
      clrf  PORTB
      clrf  PORTC
      clrwdt                  ;clear watch dog & prescaler before switching
prescaler
      clrf  TMR0
      clrf  INTCON            ;disable all interrupts for now
      movlw 0x30        ;set sync serial port f_osc/4, rcv on rising edge
      movwf SSPCON

;initialize bank 1 control regs

      bsf   STATUS,RP0  ;select bank 1
      movlw 0x30        ;set PORTA pins 0,1,2,3 as outputs, 4,5 as inputs
      movwf TRISA       ;(during operation pin 5 is switched between input and
output)
      movlw 0x07        ;set all PORTB pins as output
```

```
      movwf TRISB       ;except interrupt RB0, VCO polarity RB1, O/C loop RB2
      movlw 0x07        ;set PORTA pin as digital (not ADC inputs)
      movwf ADCON1
      movlw 0x67        ;int. on rising edge of RB0, prescaler 256 -> timer0 ->
ext. ctr.
      movwf OPTION_REG
      movlw 0xd7        ;SDI pin input (SSP mode), serial port pins, 3 user
inputs
      movwf TRISC
      clrf  PIE1        ;no int. on async xmt (tst TRMT instead)
      bsf   TXSTA,TXEN  ;enable USART xmt (async mode)
      bsf   TXSTA,BRGH  ;set USART hi speed mode
      movlw D'38'       ;set async rate at 9600 baud (25. for 4 MHz xtal,
BRGH=1)
      movwf SPBRG       ;                            (38. for 6 MHz xtal,
BRGH=1)

;back to bank 0

      bcf   STATUS,RP0  ;bank 0 for RCSTA reg
      bsf   RCSTA,SPEN  ;enable serial port
      bsf   INTCON,INTE ;enable interrupt on RB0
      bsf   INTCON,GIE  ;enable interrupts
      clrf  Ldata
      clrf  Hdata
      clrf  glcctr
      bsf   PORTB,7           ;set DAC latch hi
      bcf   PORTB,6           ;set DAC clock line low
      bcf   PORTB,5           ;set DAC data out line lo
      bsf   FrstPss,0   ;set first pass flag
      movlw 0x04        ;initialize phase set point to 1024 (0x0400)
      movwf Hsp
      clrf  Lsp
      movlw D'1'        ;set IIR sample time to just use tsecs value
      movwf minutes           ;(this is historical - allows reporting time
      movwf toc         ;to be < sample time if desired)

;read user input pins and initialize accordingly

      call  setFilt           ;set IIR filter constants based on PORTC user pins
      movf  tsecs,W           ;initialize tick counter accordingly
      movwf tick
      bsf   PORTB,4           ;clear phase detector counter
      bcf   PORTB,4
      clrf  TMR0        ;clear internal counter

wait  goto  wait        ;wait for interrupt

;interrupt service routine

int_srv     movwf W_TEMP            ;"push" instructions (see AN611)
      swapf STATUS,W
      bcf   STATUS,RP0
      movwf STAT_TEMP
      bcf   INTCON,INTF ;clear interrupt, else int again immediately!
      bcf   PORTA,2           ;toggle heartbeat on/off on alternate gps pulses
      btfss tick,0            ;test low bit of tick counter
```

```
      bsf   PORTA,2           ;turn LED on if tick is even
      decfsz      tick        ;read phase counter, filter, set DAC every TCOUNT
sec
      goto  pop           ;otherwise just clean up & return from interrupt

;begin useful code

      movf  tsecs,W           ;reinitialize tick down counter
      movwf tick
      call  setFilt           ;check setting of user inputs on each interrupt
      call  read166           ;get phase counter reading from external shift
register
;*****       call  readctr           ;or read data from internal counter and
prescaler
      call  Bin2BCD           ;convert data to BCD
      call  sendnum           ;convert to ascii & transmit - BEFORE dGlitch or
limiter
      call  dGlitch           ;ignore large phase changes (only in iir modes 4
to 7)
      call  limiter           ;limit filter input to =<2047 (for Ts = 30 sec)


;compute and output OCXO control voltage with user-selected filter

      btfss PORTB,1           ;usr input for VCO polarity - 1 means V^ => F^
      goto  revpol            ;pin 1 clear means V^ => F decreases
      call  Data2A            ;get current phase reading to A
      call  SP2B          ;get desired phase set point to B
      goto  getdiff
revpol      call  Data2B            ;current phase reading (rev. pol.for
HP5328A)
      call  SP2A          ;get desired phase set point (see above)
getdiff     call  Sub24             ;B - A -> B reg
      call  B2FIBCD           ;copy B to filter input store & ascii conv
      ;call print         ;print phase difference from setpoint
      movf  cLoBits,W     ;test lo bits of PORTC - 0 for setup mode
      btfsc STATUS,Z      ;if set at zero...
      goto  setupMd           ;branch for setup mode
      call  locktst           ;if phase near setpoint everything under control
      movf  cLoBits,W     ;get PORTC bit again
      andlw 0x06          ;test for bit 1 only set - type 1 filter
      btfss STATUS,Z      ;if higher bits zero, not IIR mode
      goto  iir           ;branch out to use IIR filter
      bsf   FrstPss,0     ;reset IIR init flag in case we ever renter IIR
      call  FI2B          ;restore B (locktest takes |phase diff|)
      call  B2C           ;error signal direct to dac mode
      call  mulCStr           ;multiply the error value
      goto  mulC5         ;x32 - set so max output +/- 3v for Ts=30 sec (x8, mulC3
??)
c2dac movf  Hc,W          ;save for later BCD conversion & in Hdac,Ldac
      movwf Hbyte
      movwf Hdac
      movf  Mc,W
      movwf Lbyte
      movwf Ldac
      call  compC18           ;Dac needs 2s compliment data
      btfss PORTB,2           ;user input = 1 means Open loop (don't change DAC)
```

```
      call  setDAC (10)          ;send value to the DAC
sendpop     call  print          ;print filter output (DAC value to set VCO input
voltage)
stordat     call  Data2pd              ;save current phase data as previous data
      call  prntUSt              ;print mode input pins and status LEDS

;end useful code

pop   bcf   STATUS,RP0  ;"pop" instructions (see AN611)
      swapf STAT_TEMP,W
      movwf STATUS
      swapf W_TEMP,F
      swapf W_TEMP,W

      retfie                     ;return from interrupt

;read external counter (shift register) to get phase data

read166     bcf   PORTB,3              ;set '166 shift/load pin to load
      call  read_spi    ;just to toggle the '166 clock (8 times!)
      bsf   PORTB,3              ;set '166 pin to shift to get data
      call  read_spi
      movwf Hdata       ;save the hi 8 bits
      movwf Hbyte       ;save again for B_2_BCD
      call  read_spi
      movwf Ldata       ;ditto for lo 8 bits
      movwf Lbyte
      bsf   PORTB,4              ;toggle i/o pin to clear counter
      bcf   PORTB,4
      return

;read internal prescaler and counter to get phase data (1.5 msec max @ 6 MHz)
;[RA4 (counter input) tied to RA5, and to input source via 470 ohm resistor]


readctr     movf  TMR0,W               ;get main counter value (8 MSBs)
      movwf Hdata       ;save
      movwf Hbyte       ;save again for B_2_BCD routine
      bsf   STATUS,RP0  ;set bank 1 to access tris
      bcf   TRISA,5              ;make RA5 an output
      bcf   STATUS,RP0  ;back to bank 0
      clrf  temp        ;clear loop counter
toggle      incf  temp,1               ;increment prescaler until overflow (AN592)
      bcf   PORTA,5              ;toggle pin connected to count (prescaler) input
      bsf   PORTA,5
      movf  TMR0,W               ;check for overflow to main counter
      subwf Hdata,W              ;subtract earlier reading
      btfsc STATUS,Z    ;if tmr0 incremented get out of loop
      goto  toggle               ;otherwise toggle again
      comf  temp,1               ;complement number of toggles needed
      incf  temp,W               ;+1
      movwf Ldata       ;save
      movwf Lbyte       ;and again
      bsf   STATUS,RP0  ;set bank 1 to access tris
      bsf   TRISA,5              ;make RA5 an input again to release counter
      bcf   STATUS,RP0  ;back to bank 0
      clrf  TMR0        ;clear the counter & prescaler
```

```
        return

;IIR filter (digital vsn of amplified lead/lag) - expects current input in B
;parameters are Ts=30, F2=64, F1 and Kcpu selected by user pins (PORTC 0,1,2)
;fo=fol + (1/F1 + 1/F2)*fi + (1/F1 - 1/F2)*fil
;The input data is placed in the top two bytes of the 40-bit math reg
;to allow division w/o losing accuracy. Output is taken from the
;top 18 bits. The filter gain is controlled
;by left-shifting the output value before sending to the DAC.

iir     decfsz      toc         ;filter sample time is "minutes" minutes
        goto  nottime           ;not time yet
        movf  minutes,W   ;reset toc counter
        movwf toc
        btfsc FrstPss,0   ;first time thru the filter?
        call  initIIR           ;yes  - init previous value storage
        call  scale       ;check if Tc was changed since last entry
        call  FI2B        ;get current filter input: i(n)
        call  RshBL6              ;i(n)/64
        call  B2A         ;tranfer to A for add
        call  divByF1             ;i(n)/F1 - note: already divided by 64
        call  Add40       ;B has i(n)*(1/F1 + 1/64)
         ;call      printB
        call  FOL2A       ;get o(n-1) - last filter output
        call  Add40       ;B has o(n-1)+i(n)*(1/F1 + 1/64)
         ;call      printB
        call  B2C         ;save in C for a bit
        call  FIL2B       ;get i(n-1) - last filter input
         ;call      printB
        call  RshBL6              ;i(n-1)/64
        call  B2A         ;transfer to A for subtraction
        call  divByF1             ;i(n-1)/F1 - note: already divided by 64
        call  Sub40       ;B has i(n-1)*(1/F1 - 1/64), a neg value
         ;call      printB
        call  C2A         ;get previous partial sum
        call  Add40       ;o(n-1)+i(n)*(1/F1+1/64)+i(n-1)*(1/F1-1/64)
         ;call      printB
        call  B2C         ;save for output to dac, etc
        call  B2FOL       ;also save as o(n-1)
         ;call      prntB36          ;debug
        call  FI2FIL              ;move i(n) to i(n-1)
        call  mulCStr             ;initialize multiply routine that sets loop gain
        movf  cLoBits,W   ;setting of users bits selects gain (Kcpu)
        andlw 0x07        ;mask to discard hi bits  debug *****
        addwf PCL,1       ;index into table - mulC_ goes to c2dac
        goto  mulC11              ;should not get here with cLoBits=0
        goto  mulC11              ;multiply by 2^11
        goto  mulC10              ;or 2^10, etc
        goto  mulC9
        goto  mulC8
        goto  mulC7
        goto  mulC6
        goto  mulC5
```

```
initIIR     call  B2ILOL            ;copy current error sig to previous i, clear
o
      movfw cLoBits           ;initialize previous filter mode storage
      movwf pLoBits
      clrf  FrstPss
      return

nottime     movf  Hdac,W            ;not IIR sample time yet
      movwf Hbyte       ;just copy old dac value for ascii ouput
      movf  Ldac,W
      movwf Lbyte
      goto  sendpop           ;jump back to output ascii and retfie


;user-selected initialize routines (selected by 3 lo bits (value 0-7) in PORTC

setFilt     movf  PORTC,W           ;read user inputs
      andlw 0x07        ;mask to discard hi bits
      movwf cLoBits           ;save for mulC_ jump table indexing, etc
      addwf PCL,1       ;index into table to initialize sample time
      goto  usr0 (7)    ;NOTE: all the sample time stuff is historical
      goto  usr1
      goto  usr2
      goto  usr3
      goto  usr4
      goto  usr5
      goto  usr6
      goto  usr7

usr0  movlw D'30'       ;special case if needed for setup mode
      goto  settsec
usr1                    ;or type 1 filter
usr2                    ; IIR filt
usr3                          1
usr4                          2
usr5                          3
usr6                          4
usr7  movlw D'30'       ;sample time 30 sec
settsec     movwf tsecs       ;store selected value into tsecs
      return

divByF1     movf  cLoBits,W   ;setting of user bits selects F1
      addwf PCL,1       ;index into table - RshBl_ returns to caller
  0   goto  RshBL4            ;should not get here - not using iir filter
  1   goto  RshBL4            ;F1=2^10 (already shifted 2^6 (64)
  2   goto  RshBL5            ;2^11   2 right shift 5 times
  3   goto  RshBL6            ;2^12     "     6 times
  4   goto  RshBL7            ;2^13     "     7 times
  5   goto  RshBL8            ;2^14     "     8 times
  6   goto  RshBL9            ;2^15     "     9 times
  7   goto  RshBL10           ;2^16     "     10 times

;change iir filter data (last filter output) when user changes Tc to avoid
;a transient.  Note we only scale a step at a time, so change Tc progressively.

scale movf  pLoBits,W   ;has user changed Tc since last entry?
```

```
        subwf cLoBits,0    ;current - previous
        btfsc STATUS,Z     ;are they the same?
        return                   ;yes, nothing to do
        btfsc STATUS,C     ;test sign of difference - if negative C is zero
        goto  up           ;increasing Tc
        call  FOL2A        ;decreasing Tc - need to divide last output by two
        call  A2C
        call  C2B
        call  RshftBL            ;divide - works for both + and - values
stofol  call  B2FOL        ;store it
        movf  cLoBits,W    ;update previous setting
        movwf pLoBits
        return
up      call  FOL2A        ;increasing Tc - need to multiply last output by two
        call  A2C
        call  mulCStr            ;multiply set up - works for both + and - values
;this bit of code is the same as multC1 except for the returns - combine later
        call  LshfC40
        bcf   Hc,7         ;make sure it's positive
        btfss negFlag,0    ;was the original value negative?
        goto  sc1          ;no - nothing to do, just go back to main stream
        call  compC40            ;yes - restore sign
        bcf   negFlag,0    ;clear flag

sc1     call  C2B
        goto  stofol

;ignore current phase data, d[i], if d[i] - d[i-1] is too large.  Use d[i-1]
;instead. To avoid hangups in case of valid phase offset use only 3 times in a
row.
;set glitch LED.

dGlitch     movf  cLoBits,W    ;use only in slower iir filter modes: 4,5,6,7
        andlw 0x04         ;fast modes make large steps that conflict
        btfsc STATUS,Z         ;test mode bits, 4 bit must be set
        goto  clrret             ;just return, we're not in iir mode
        call  Data2B             ;begin computation of d[i]-d[i-1]
        call  pd2A
        call  Sub24        ;current-previous in B
        btfsc Hb,7         ;want |diff|
        call  compB        ;so if neg make it positive
        ;call  printB
        movlw D'30'        ;glitch limit constant
        movwf Ma           ;into A low bits of 16 bit word
        clrf  Ha
        clrf  La
        call  Sub24        ;|diff|-limit in B
        btfsc Hb,7         ;negative?
        goto  clrret             ;yes, within limit, nothing to do
        incf  glcctr             ;increment successive glitch counter
        btfsc glcctr,2     ;test counter for >3 in-a-row
        goto  clrret             ;too many in-a-row, don't overwrite current value
        movf  Hpdat,W            ;otherwise get d[i-1]
        movwf Hdata        ;and overwrite d[i]
        movf  Lpdat,W            ;note: this will cause d[i-1] to remain
        movwf Ldata        ;unchanged on successive entries
        bsf   PORTA,1            ;turn glitch LED on
```

```
        return                  ;hence the need for the counter
clrret  clrf  glcctr                ;clear counter and return
        bcf   PORTA,1           ;turn glitch LED off (if on)
        return


;limits the max input from the phase detector so low freq VCOs do not
;cause a large data input values to the filter. Called before setpoint
;is subtracted.

limiter movf  Hdata,W           ;if data >= 2048, set to 2047 (for setpoint
= 1024)
        andlw 0xf8          ;get bits 2048 and above
        btfsc STATUS,Z      ;if zero value is 2047 or less
        return                  ;OK as it is
        movlw 0x7           ;otherwise make it 2047
        movwf Hdata
        movlw 0xff
        movwf Ldata
        return

;test for phase reading near setpoint: an indicator (over time) that PLL is
locked

locktst call    Data2B              ;get current phase reading
        call  SP2A          ;get setpoint
        call  Sub24         ;phase-setpoint in B
        btfsc Hb,7          ;we want |data-sp|, so change sign if negative
        call  compB
        bcf   PORTA,3           ;turn LED off first
        movf  Hb,W          ;get high byte of |diff|
        btfss STATUS,Z      ;any bits set means |diff| > 255
        bsf   PORTA,3           ;so turn on LED
        return


;setup mode - assist user in coarse adjustment of VCXO frequency

setupMd clrf  temp2         ;setup mode - turn on LEDS when phase is
                        ;increasing (freq lo) or decreasing (freq hi)
        clrf  Hc            ;set DAC to zero to center VCO
        clrf  Mc            ;(also useful to measure DAC offset)
        clrf  Lc
        call  setDAC
        bcf   PORTA,1           ;initially set both off
        bcf   PORTA,3
        call  Data2B            ;get current phase data
        ;call printB            ;debug
        call  pd2A          ;get previous phase
        call  Sub24         ;(current - previous) phase in B
        ;call printB            ;debug
        btfss Hb,7          ;is difference negative?
        goto  dpl           ;no - branch around negative case stuff
        incf  temp2         ;yes - set sign flag for later use
        call  compB         ;and make it positive, i.e. we have |B| now
dpl     ;call printB            ;debug
        movlw D'60'         ;limit value, |diff| > limit means turn an LED on
```

```
        movwf Ma            ;limit to A reg
        clrf  Ha
        clrf  La
        call  Sub24         ;B - A -> B
        ;call       printB              ;debug
        btfsc Hb,7          ;if negative |diff| is within limits
        goto  stordat             ;so just go back to interrupt loop
        btfsc temp2,0             ;outside limits, check direction
        goto  hiLED         ;freq too hi, jump
        ;movlw      D'99'          ;next 4 lines are debug
        ;movwf      Lbyte
        ;clrf Hbyte
        ;call print
        bsf   PORTA,1             ;freq too lo - set lo LED on
        goto  stordat             ;back to interrupt loop
hiLED   bsf   PORTA,3             ;freq too hi - set hi LED on
        ;movlw      D'101'               ;next 4 lines are debug
        ;movwf      Lbyte
        ;clrf Hbyte
        ;call print
        goto  stordat


;output the 18-bit word in C reg to a Burr-Brown PCM-16P DAC

setDAC      movlw D'17'         ;setup loop counter
        movwf temp
        ;reset latch only after next MSB - funny business for the PCM-16
        call  ssc           ;shift out next bit, set data line, clock
        bsf   PORTB,7             ;set latch hi AFTER the first bit of the new word
dloop   ;now do the lower 17 bits
        call  ssc           ;shift out next bit, set data line, clock
        decfsz      temp,1              ;dec loop count
        goto  dloop         ;loop 18 times
        bcf   PORTB,7             ;to set new value into DAC set latch lo...
        return                    ;note - this routine destroys data in C
ssc     call  LshiftC             ;shift input reg: hi bits out first
        btfss STATUS,C      ;to equalize timing - remove later
        nop                 ;(only makes a better scope display)
        btfsc STATUS,C      ;was top bit a one?
        bsf   PORTB,5             ;yes - set data line hi
        bsf   PORTB,6             ;data now set toggle clock line
        bcf   PORTB,6
        bcf   PORTB,5             ;set data out line lo again
        return

;output the BCD nibbles in R2,R1,R0 as ascii to async port

sendnum     movf  R0,W          ;send most sig digit first
        movwf CHAR
        call  sendchar
        movf  R1,W
        movwf CHAR
        swapf CHAR,1              ;first hi nibble
        call  sendchar
        swapf CHAR,1              ;then low nibble
        call  sendchar
```

```
        movf  R2,W          ;ditto for R2 nibbles
        movwf CHAR
        swapf CHAR,1
        call  sendchar
        swapf CHAR,1
        call  sendchar
        movlw "\r"          ;send carrage return by direct call to xmt
        call  xmt
        movlw "\n"          ;then send newline
        call  xmt
        return

;convert to ascii and send bits <0:3> in CHAR

sendchar
        movf  CHAR,W              ;get the char
        andlw 0x0f          ;mask off high nibble
        addlw "0"           ;make into ascii
        call  xmt           ;nibble in w, send it
        return

;send whatever is in W out the async serial port

xmt
        bsf   STATUS,RP0    ;set bank1 to access TXSTA reg
xmt1    btfss TXSTA,TRMT    ;test for xmt reg empty
        goto  xmt1          ;wait if necessary
        bcf   STATUS,RP0    ;back to bank 0
        movwf TXREG         ;send it

        return

;Binary to BCD (16 Bit) Conv Routine - adapted for 16Cxx from AN544
;       input in Lbyte, Hbyte and output in R2, R1, R0

Bin2BCD
        bcf   STATUS,C
        clrf  count
        bsf   count,4             ;set count=16
        clrf  R0
        clrf  R1
        clrf  R2
loop16
        rlf   Lbyte
        rlf   Hbyte
        rlf   R2
        rlf   R1
        rlf   R0
;
        decfsz      count         ;no decfsnz inst on the 16Cxx
        goto  n1
        return                    ;done
n1      movlw R2            ;load addr of R2 as indirect addr
        movwf FSR
;adjust BCD
        movf  INDF,W              ;get R2 (later R1, R0) via indirect addr
        addlw 0x03
```

```
      movwf wtmp        ;move sum to file reg for bit test
      btfsc wtmp,3            ;test if result >7
      movwf INDF        ;yes - store sum
      movf  INDF,W            ;get original or sum
      addlw 0x30        ;test hi byte
      movwf wtmp        ;move sum to file reg for bit test
      btfsc wtmp,7            ;test if result >0x70
      movwf INDF        ;save as BCD
;
      incf  FSR
;
      movf  INDF,W            ;repeat for R1
      addlw 0x03
      movwf wtmp
      btfsc wtmp,3
      movwf INDF
      movf  INDF,W
      addlw 0x30
      movwf wtmp
      btfsc wtmp,7
      movwf INDF
;
      incf  FSR
;
      movf  INDF,W            ;repeat for R0
      addlw 0x03
      movwf wtmp
      btfsc wtmp,3
      movwf INDF
      movf  INDF,W
      addlw 0x30
      movwf wtmp
      btfsc wtmp,7
      movwf INDF
;
      goto  loop16

;subroutine to read data from the SPI port, return in W

read_spi
      movwf SSPBUF            ;initiate a read cycle by writing SSPBUF
      bsf   STATUS,RP0  ;set bank1 to access SSPSTAT
loopspi     btfss SSPSTAT,BF  ;test i/o complete
      goto  loopspi           ;wait - not ready yet
      bcf   STATUS,RP0  ;back to bank 0
      movf  SSPBUF,W    ;get data byte into W
      return

;24-bit math routines (see AN611) and xfer routines used to compute SM,IIR, etc

SM2A  movf  Hsm,W       ;transfer smoothed value to A reg
      movwf Ha
      movf  Msm,W
      movwf Ma
      movf  Lsm,W
      movwf La
      return
```

```
SP2A  movf  Hsp,W       ;transfer phase setpoint value (16 bits) to A reg
      movwf Ha
      movf  Lsp,W
      movwf Ma
      clrf  La
      clrf  Sa
      clrf  SSa
      return

SP2B  movf  Hsp,W       ;transfer phase setpoint value (16 bits) to B reg
      movwf Hb
      movf  Lsp,W
      movwf Mb
      clrf  Lb
      clrf  Sb
      clrf  SSb
      return

Data2A      movf  Hdata,W           ;transfer Data (2 bytes) to A reg
      movwf Ha
      movf  Ldata,W
      movwf Ma
      clrf  La
      clrf  Sa
      clrf  SSa
      return

Data2B      movf  Hdata,W           ;transfer Data (2 bytes) to B reg
      movwf Hb
      movf  Ldata,W
      movwf Mb
      clrf  Lb
      clrf  Sb
      clrf  SSb
      return

Data2pd     movf  Hdata,W           ;transfer Data (2 bytes) to previous data
store
      movwf Hpdat
      ;movwf      Hbyte       ;debug print
      movf  Ldata,W
      movwf Lpdat
      ;movwf      Lbyte       ;debug print
      ;call print       ;debug print
      return

pd2A  movf  Hpdat,W           ;transfer previous data (2 bytes) to A reg
      movwf Ha
      movf  Lpdat,W
      movwf Ma
      clrf  La
      clrf  Sa
      clrf  SSa
      return

FI2A  movf  Hfi,W       ;filter input to A (2 bytes)
```

```
        movwf Ha
        movf  Lfi,W
        movwf Ma
        clrf  La
        clrf  Sa
        clrf  SSa
        return

FI2B    movf  Hfi,W       ;filter input to B (2 bytes)
        movwf Hb
        movf  Lfi,W
        movwf Mb
        clrf  Lb
        clrf  Sb
        clrf  SSb
        return

FIL2B movf  Hfil,W            ;previous filter input to B (2 bytes)
        movwf Hb
        movf  Lfil,W
        movwf Mb
        clrf  Lb
        clrf  Sb
        clrf  SSb
        return

FI2FIL      movf  Hfi,W       ;filter input to previous filter input (2 bytes)
        movwf Hfil
        movf  Lfi,W
        movwf Lfil
        return

B2FOL movf  Hb,W        ;transfer 40 bits B reg to previous output
        movwf Hfol
        movf  Mb,W
        movwf Mfol
        movf  Lb,W
        movwf Lfol
        movf  Sb,W
        movwf Sfol
        movf  SSb,W
        movwf SSfol
        return

FOL2A movf  Hfol,W            ;transfer 40 bits previous output to A reg
        movwf Ha
        movf  Mfol,W
        movwf Ma
        movf  Lfol,W
        movwf La
        movf  Sfol,W
        movwf Sa
        movf  SSfol,W
        movwf SSa
        return

;FOL2B      movf  Hfol,W           ;not used
```

```
        movwf Hb
        movf  Mfol,W
        movwf Mb
        movf  Lfol,W
        movwf Lb
        return

A2C     movf  Ha,W          ;transfer 40 bits A reg to C reg
        movwf Hc
        movf  Ma,W
        movwf Mc
        movf  La,W
        movwf Lc
        movf  Sa,W
        movwf Sc
        movf  SSa,W
        movwf SSc
        return

B2A     movf  Hb,W          ;transfer 40 bits B reg to A reg
        movwf Ha
        movf  Mb,W
        movwf Ma
        movf  Lb,W
        movwf La
        movf  Sb,W
        movwf Sa
        movf  SSb,W
        movwf SSa
        return

B2C     movf  Hb,W          ;transfer 40 bits B reg to C reg
        movwf Hc
        movf  Mb,W
        movwf Mc
        movf  Lb,W
        movwf Lc
        movf  Sb,W
        movwf Sc
        movf  SSb,W
        movwf SSc
        return

C2A     movf  Hc,W          ;transfer 40 bits C reg to A reg
        movwf Ha
        movf  Mc,W
        movwf Ma
        movf  Lc,W
        movwf La
        movf  Sc,W
        movwf Sa
        movf  SSc,W
        movwf SSa
        return

C2B     movf  Hc,W          ;transfer 40 bits C reg to B reg
        movwf Hb
```

```
        movf  Mc,W
        movwf Mb
        movf  Lc,W
        movwf Lb
        movf  Sc,W
        movwf Sb
        movf  SSc,W
        movwf SSb
        return

B2SMBCD movf  Hb,W          ;transfer B reg to SM and
        movwf Hsm          ;two hi B bytes to Bin2BCD work regs
        movwf Hbyte
        movf  Mb,W
        movwf Msm
        movwf Lbyte
        movf  Lb,W
        movwf Lsm
        return

B2FIBCD movf  Hb,W          ;transfer B reg to filter input and
        movwf Hfi          ;to Bin2BCD work regs (2 bytes)
        movwf Hbyte
        movf  Mb,W
        movwf Lfi
        movwf Lbyte
        return

B2ILOL  movf  Hb,W          ;xfer B reg to previous input, clr prev. output
        movwf Hfil
        movf  Mb,W
        movwf Lfil
        clrf  Hfol
        clrf  Mfol
        clrf  Lfol
        clrf  Sfol
        clrf  SSfol
        return

RshiftB btfsc Hb,7          ;right-shift 24 bit B reg by 1
        goto  shfBneg       ;highest bit set - consider a neg no.
        bcf   STATUS,C     ;clear carry flag before shifting
        rrf   Hb           ;shift
        rrf   Mb
        rrf   Lb
        return
shfBneg call  compB         ;make a neg no positive
        bcf   STATUS,C     ;clear carry flag before shifting
        rrf   Hb           ;shift
        rrf   Mb
        rrf   Lb
        call  compB        ;make it neg again
        return

RshftBL btfsc Hb,7          ;right-shift 40 bit B reg by 1
        goto  shfBLng       ;highest bit set - consider a neg no.
        bcf   STATUS,C     ;clear carry flag before shifting
```

```
        rrf   Hb          ;shift
        rrf   Mb
        rrf   Lb
        rrf   Sb
        rrf   SSb
        return
shfBLng call  compB40           ;make a neg no positive
        bcf   STATUS,C    ;clear carry flag before shifting
        rrf   Hb          ;shift
        rrf   Mb
        rrf   Lb
        rrf   Sb
        rrf   SSb
        call  compB40           ;make it neg again
        return

RshftB4 call  RshiftB           ;right-shift B reg four times
        call  RshiftB
        call  RshiftB
        call  RshiftB
        return

RshBL12 call  RshftBL           ;right-shift long (40 bit) B reg 12 times
        call  RshftBL
RshBL10 call  RshftBL
RshBL9  call  RshftBL
RshBL8  call  RshftBL
RshBL7  call  RshftBL           ;right-shift long (40 bit) B reg 7 times
RshBL6  call  RshftBL           ;right-shift long (40 bit) B reg 6 times
RshBL5  call  RshftBL           ;right-shift long (40 bit) B reg 5 times
RshBL4  call  RshftBL
        call  RshftBL
        call  RshftBL
        call  RshftBL
        return


LshiftC bcf   STATUS,C    ;left-shift 24 bit C reg by 1
        rlf   Lc,1
        rlf   Mc,1
        rlf   Hc,1
        return

LshfC40 bcf   STATUS,C    ;left-shift 40 bit C reg by 1
        rlf   SSc,1
        rlf   Sc,1
        rlf   Lc,1
        rlf   Mc,1
        rlf   Hc,1
        return

mulCStr btfss Hc,7              ;set up for signed multiply of value in C reg
        return                  ;value is positive, just return
        bsf   negFlag,0   ;negative - set flag
        call  compC40           ;make positive
        return
mulC11  call  LshfC40           ;multiply by shifting left
```

```
mulC10      call  LshfC40           ;enter at n, get (2^n)
mulC9 call  LshfC40           ;
mulC8 call  LshfC40
mulC7 call  LshfC40
mulC6 call  LshfC40
mulC5 call  LshfC40
mulC4 call  LshfC40
mulC3 call  LshfC40
mulC2 call  LshfC40
mulC1 call  LshfC40
      bcf   Hc,7          ;make sure it's positive
      btfss negFlag,0     ;was the original value negative?
      goto  c2dac         ;no - nothing to do, just go back to main stream
      call  compC40             ;yes - restore sign
      bcf   negFlag,0     ;clear flag
      goto  c2dac         ;go back to main stream


initSM      call  Data2B            ;on the first pass set SM = 1st data value
      call  B2SMBCD           ;(use of existing code requires 2 calls)
      clrf  FrstPss
      return

Add24 movf  La,W          ;A + B -> B  (24 bits)
      addwf Lb            ;add low bytes
      btfsc STATUS,C      ;add in carry if necessary
      goto  add2
add1  movf  Ma,W
      addwf Mb            ;add mid bytes
      btfsc STATUS,C      ;add in carry if necessary
      incf  Hb
      movf  Ha,W
      addwf Hb
      return
add2  incf  Mb
      btfsc STATUS,Z
      incf  Hb
      goto  add1

Add40 movf  SSa,W         ;A + B -> B  (40 bits)
      addwf SSb           ;add low bytes
      btfsc STATUS,C      ;add in carry if necessary
      call  addl3
      movf  Sa,W
      addwf Sb
      btfsc STATUS,C
      call  addl2
      movf  La,W
      addwf Lb
      btfsc STATUS,C
      call  addl1
      movf  Ma,W
      addwf Mb
      btfsc STATUS,C
      incf  Hb
      movf  Ha,W
      addwf Hb
```

```
        return

addl3   incf    Sb              ;when Z is clear skip the rest of the chain
        btfsc   STATUS,Z
addl2   incf    Lb
        btfsc   STATUS,Z
addl1   incf    Mb
        btfsc   STATUS,Z
        incf    Hb
        return

Sub24   call    compA           ;B - A -> B (24 bits)
        call    Add24
        return

Sub40   call    compA40         ;B - A -> B (40 bits)
        call    Add40
        return


compA   comf    La              ;2s complement of A -> A
        comf    Ma              ;invert all the bits in A
        comf    Ha
        incf    La              ;add one to A
        btfsc   STATUS,Z
        incf    Ma
        btfsc   STATUS,Z
        incf    Ha
        return

compA40     comf    SSa             ;2s complement of A -> A
        comf    Sa              ;invert all the bits in A
        comf    La
        comf    Ma
        comf    Ha
        incf    SSa             ;add one to A
        btfsc   STATUS,Z
        incf    Sa
        btfsc   STATUS,Z
        incf    La
        btfsc   STATUS,Z
        incf    Ma
        btfsc   STATUS,Z
        incf    Ha
        return

compB   comf    Lb              ;2s complement of B -> B
        comf    Mb              ;invert all the bits in B
        comf    Hb
        incf    Lb              ;add one to B
        btfsc   STATUS,Z
        incf    Mb
        btfsc   STATUS,Z
        incf    Hb
        return

compB40     comf    SSb             ;2s complement of B -> B
```

```
        comf   Sb           ;invert all the bits in B
        comf   Lb
        comf   Mb
        comf   Hb
        incf   SSb          ;add one to B
        btfsc  STATUS,Z     ;a skip chain
        incf   Sb
        btfsc  STATUS,Z
        incf   Lb
        btfsc  STATUS,Z
        incf   Mb
        btfsc  STATUS,Z
        incf   Hb
        return

compC   comf   Lc           ;2s complement of C -> C
        comf   Mc           ;invert all the bits in C
        comf   Hc
        incf   Lc           ;add one to C
        btfsc  STATUS,Z
        incf   Mc
        btfsc  STATUS,Z
        incf   Hc
        return

compC18     comf   Lc           ;2s compliment of 18 bit value in C -> C
        comf   Mc           ;invert all the bits in A
        comf   Hc
        movlw  0x40         ;add 1 (18th bit, not 24th!)
        addwf  Lc,1
        btfsc  STATUS,Z
        incf   Mc
        btfsc  STATUS,Z
        incf   Hc
        return

compC40     comf   SSc          ;2s complement of C -> C
        comf   Sc           ;invert all the bits in C
        comf   Lc
        comf   Mc
        comf   Hc
        incf   SSc          ;add one to C
        btfsc  STATUS,Z     ;a skip chain
        incf   Sc
        btfsc  STATUS,Z
        incf   Lc
        btfsc  STATUS,Z
        incf   Mc
        btfsc  STATUS,Z
        incf   Hc
        return


;print code for debugging - uses only R0, R1, R2 regs

B2BCD   movf   Hb,W         ;transfer B reg
        movwf  Hbyte        ;two hi B bytes to Bin2BCD work regs
```

```
      movf  Mb,W
      movwf Lbyte
      return

printB      call  B2BCD       ;get B (2 bytes) to print
print call  Bin2BCD           ;value already in work regs
      call  sendnum
      return

prntB36     call  printB            ;print two 16 bit words H,M,L,S reg of B
      movf  Lb,W
      movwf Hbyte       ;put lower bytes into print work regs
      movf  Sb,W
      movwf Lbyte
      goto  print       ;"print" will return to caller

prntUSt     movf  PORTC,W           ;print usr input bits and status
      andlw 0x07        ;mask user bits
      btfsc PORTB,2           ;check open loop bit (1 if open loop)
      addlw D'50'       ;50 indicates open loop operation
      btfsc PORTA,3           ;"unlock" LED on?
      addlw D'100'            ;if on add 100.
      btfsc PORTA,1           ;dglitch LED on?
      addlw D'10'       ;if on add 10
      movwf Lbyte       ;print the total
      clrf  Hbyte
      goto  print


;test the DAC output code - just sequence thru all the values

dactst      clrf  Ha          ;set A reg to 1 in 18th bit down
      clrf  Ma
      clrf  La
      bsf   La,6
      clrf  Hb          ;clear B reg
      clrf  Mb
      clrf  Lb
dtloop      call  Add24       ;A + B -> B
      call  B2C         ;B -> C (to preserve B)
      call  compC18           ;the DAC needs 2s complement values
      call  setDAC            ;send the data
      goto  dtloop            ;loop forever incrementing the output by 1

;test Bin2BCD and output code
;just increment and output 16 bit number

inctst      incf  Ldata,1           ;increment 16 bit value
      btfsc STATUS,C    ;tst carry
      incf  Hdata,1           ;increment if necessary
      movf  Ldata,W           ;mov to work registers
      movwf Lbyte
      movf  Hdata,W
      movwf Hbyte
      return

      de    "vsn 1.28, copyright Brooks Shera 1998"
```

```
de      " "

END
```