

DDS V1.asm

\*\*\*\*\*

Source code for Direct Digital Synthesis VFO controller

\*\*\*\* NOW PORTED TO PIC16C84 \*\*\*\* (see revision note for 10/01/98)
\*\*\*\* NOW USING MICROCHIP MPASM ASSEMBLER (see revision note for 10/14/98)
\*\*\*\* WORKS WITH PIC16F84A without further change (03/22/05)

This code is available as is.

Description:

The program main loop, (Check\_Encoder) polls a rotary shaft encoder.
If the input from the shaft encoder has changed, then direction (up/down)
is determined. The program then computes the corresponding
frequency digits to be displayed on the Liquid Crystal Display.
This data is transferred to the LCD using a 4 bit interface.
A new DDS control word is then computed and transferred
serially to the AD9850 DDS chip.
The program then returns to the shaft encoder polling loop.

Target Controller - PIC1654A (originally)

Program Size - 474 words, 38 words free

Assembler - Parallax SPASM v4.2

Author - Curtis W. Preuss - WB2V

Initial creation 11/08/96

Modification History

11/24/96 Finished debugging

12/14/96 Edited comments

10/01/98 Ported source to PIC16C84 -- G. Heron, N2APB

- 1) Needed to shift the variables down by 5 bytes due to changes in memory architecture
2) Needed to modify some of the indirect addressing "reads" during compares to list ends (16C84 has more FSR bits)
3) Changed assembler directive device code
4) Added 2 more function sets during LCD init, per the data sheets for my wacko (bargain basement) Seiko M1641 displays
5) Changed delays during LCD init (added wait60ms rtn), (again, due to wacko Seiko M1641 LCDs?)
6) Removed assembler directive for reset vector and just added "jmp start" at loc 0 and a new "org 5" to bypass some of the interrupt vector locs at start of code space
7) Added explicit and separate "setb LCD\_rw" in Busy\_check routine to more reliably read Busy bit. (Again, perhaps only because of my wacko model Seiko LCD.)
8) Slightly modified source formatting to make it easier (for me) to read
9) Comments & questions welcome! Email to: n2apb@amsat.org

10/14/98 Ported source to Microchip MPASM -- Craig B. Johnson, AA0ZZ

- 1) Generates identical HEX code as previous PIC16C84 version
2) Added a few comments (for me!)
3) Comments & questions -- Email to: johns516@tc.umn.edu

10/23/98

- 1) Corrected translation bug in routine SendMax1

03/22/05

- 1) Digit weights recalculated for a 66.00051MHz reference oscillator

\*\*\*\*\*

list p=16f84
radix hex

Indirect equ 0
Status equ 3

DDS V1. asm

```

FSR          equ 4
PortA       equ 5
PortB       equ 6

; Assign names to I/O pins
LCD_VDD     equ 3           ; (PortA) Power to LCD module
AD9850_fqu  equ 0           ; (PortB) Update pin on AD9850
LCD_rs      equ 1           ; (PortB) 0=instruction, 1=data
LCD_rw      equ 2           ; (PortB) 0=write, 1=read
LCD_e       equ 3           ; (PortB) 0=disable, 1=enable
AD9850_wc   equ 5           ; (PortB) AD9850 write clock
AD9850_dat  equ 7           ; (PortB) AD9850 serial data input

; Allocate variables in general purpose register space
LCD_Dat     equ 0x0C        ; current Display data
AD9850      equ 0x13        ; control word for DDS chip
OP1         equ 0x18        ; Work Space for
OP2         equ 0x1C        ; the multiply routine
temp1       equ 0x1D        ; Temp storage
temp2       equ 0x1E        ; Temp storage
temp3       equ 0x1F        ; Temp storage
old         equ 0x20        ; old encoder input data
new         equ 0x21        ; new encoder input data
count1      equ 0x22        ; used for timing the shaft encoder
count2      equ 0x23        ;

; Status Bits
C           equ 0           ; Carry
DC          equ 1           ; Di git (Hal f) Carry
Z           equ 2           ; Zero

        org 0
        goto Start

        org 5
; *****
; BusyCheck Subroutine
;
; LCD read/write operations are sloooooow, this subroutine
; polls the LCD busy flag to determine if previous operations are completed.
; Note side effect that the LCD_rw and LCD_rs are left in write data mode.

BusyCheck
    movl w 0xF0             ; Tristate the 4 data pins
    tris PortB
    bsf PortB, LCD_rw      ; Raise the r/w bit to read the LCD
Busy
    bsf PortB, LCD_e       ;
    movf PortB, w          ; Get first nibble of input data
    movwf temp1           ; and save it
    bcf PortB, LCD_e       ;
    nop                   ;
    bsf PortB, LCD_e       ; Get next nibble to keep LCD happy
    bcf PortB, LCD_e       ;
    btfsc temp1, 7        ; Check for busy flag
    goto Busy             ; It is busy
    bcf PortB, LCD_rw     ; Leave LCD in Write Data mode
    bsf PortB, LCD_rs     ;
    return

; *****
; Send Character to LCD Subroutine
;

```

DDS V1.asm

; The character to be sent must have been placed in "temp2" prior to calling  
; the subroutine. LCD\_rw and LCD\_rs must be set up prior to entry.

SendChar

```

movl w    0x00           ;
tris     PortB          ; Enable port B
movf     temp2, w       ; Save temp2
movwf    temp1          ;   in temp1
movl w    0x0F           ;
andwf    PortB, f       ;
movl w    0xF0           ;
andwf    temp2, f       ;
movf     temp2, w       ;
iorwf    PortB, f       ; Load first nibble
bsf     PortB, LCD_e    ; Transfer first nibble
bcf     PortB, LCD_e    ;
swapf    temp1, f       ;
movl w    0x0F           ;
andwf    PortB, f       ;
movl w    0xF0           ;
andwf    temp1, f       ; Clear data nibble
movf     temp1, w       ;
iorwf    PortB, f       ;
bsf     PortB, LCD_e    ; Transfer second nibble
bcf     PortB, LCD_e    ;
return

```

\*\*\*\*\*

; BCD increment

; This routine does a binary coded decimal increment to the values in LCD\_Dat.  
; On entry the FSR register must point to the LCD\_Dat digit to be incremented.  
; If the result is over 20MHz the value is reset back to 20MHz.

BCDI nc

```

movf     Indi rect, w   ;
movwf    temp2         ;
movl w    0x01         ;
addwf    temp2, f      ;
movf     temp2, w      ;
movwf    temp1         ;
movl w    0x06         ;
addwf    temp1, f      ;
btfsc    Status, DC    ; See if Digit Carry is clear
goto     BCDI ncCarry  ; Digit Carry is set, so jump
movl w    0x0F         ;
andwf    temp2, f      ;
movf     temp2, w      ;
movwf    Indi rect    ;
movl w    0x02         ;
subwf    LCD_Dat+0, w  ; Test for max frequency
btfsc    Status, C     ; See if Carry is clear
goto     SetMax        ; Carry is set, so jump
return

```

BCDI ncCarry

```

clrf     temp2         ;
movf     temp2, w      ;
movwf    Indi rect    ;
decf    FSR, f         ;
goto     BCDI nc      ;

```

SetMax

```

movl w    0x12         ; Set the display to 20,000.00
movwf    FSR           ;

```

DDS V1. asm

```

SetMax1
    movl w    0x00
    movwf    Indi rect
    decf     FSR, f
    movl w    0xF3
    addwf    FSR, w
    btfsc    Status, C           ; See if Carry is clear
    goto     SetMax1           ; Carry is set, so jump
    movl w    0x02
    movwf    Indi rect
    return
; *****
; BCD decrement
;
; This routine does a binary coded decimal decrement to the values in LCD_Dat.
; On entry the FSR register must point to the LCD_Dat digit to be decremented,
; (one of 10's, 100's, 1K, 10K or 100K digit)
; If the result is under 100KHz the value is reset back to 100KHz.

BCDDec
    movf     Indi rect, w
    movwf    temp2
    movl w    0x01
    subwf    temp2, f
    btfss    Status, DC         ; See if Digit Carry is set
    goto     BCDDecCarry       ; Digit Carry is clear, so jump
    movf     temp2, w
    movwf    Indi rect
    movl w    0xFF
    addwf    LCD_Dat+0, w
    btfsc    Status, C         ; See if Carry is clear
    goto     BCDDecExit        ; Carry is set, so jump to exit
    movl w    0xFF
    addwf    LCD_Dat+1, w
    btfsc    Status, C         ; See if Carry is clear
    goto     BCDDecExit        ; Carry is set, so jump to exit
    movl w    0xFF
    addwf    LCD_Dat+2, w
    btfsc    Status, C         ; See if Carry is clear
    goto     BCDDecExit        ; Carry is set, so jump to exit
    goto     SetMi n
BCDDecExit
    return
BCDDecCarry
    movl w    0x09
    movwf    temp2
    movf     temp2, w
    movwf    Indi rect
    decf     FSR, f
    goto     BCDDec
SetMi n
    movl w    0x12
    movwf    FSR
SetMi n1
    movl w    0x00
    movwf    Indi rect
    decf     FSR, f
    movl w    0xF1
    addwf    FSR, w
    btfsc    Status, C         ; See if Carry is clear
    goto     SetMi n1         ; Carry is set, so jump
    movl w    0x01
    movwf    Indi rect

```

DDS V1. asm

```

decf    FSR, f      ;
movl w  0x00        ;
movwf   Indi rect  ;
decf    FSR, f      ;
movl w  0x00        ;
movwf   Indi rect  ;
return  ;

```

```

; *****
; Del ay 24ms subrou ti ne
;

```

```

Wai t24ms
  movl w  0x28      ; use 40 for 24ms wait if clock is 4Mhz
  movwf   temp1     ;
Wai t240uter
  movl w  0xC8      ; use 200 for 24 ms wait
  movwf   temp2     ;
Wai t24I nner
  decfsz  temp2, f  ;
  goto    Wai t24I nner ;
  decfsz  temp1, f  ;
  goto    Wai t240uter ;
return

```

```

; *****
; Del ay 60ms subrou ti ne
;

```

```

Wai t60ms
  movl w  0x64      ; use 100 for 60ms wait if clock is 4Mhz
  movwf   temp1     ;
Wai t600uter
  movl w  0xC8      ; use 200 for 60ms wait
  movwf   temp2     ;
Wai t60I nner
  decfsz  temp2, f  ;
  goto    Wai t60I nner ;
  decfsz  temp1, f  ;
  goto    Wai t600uter ;
return

```

```

; *****
; 32 bi t add
;

```

; The value in registers OP1 is added to the current 32 bit value in AD9850

```

Add32
  movf    OP1+0, w  ;
  addwf   AD9850+0, f ;
  btfss   Status, C ; See if Carry is set
  goto    Add1      ; Carry is clear, so jump
  incfsz  AD9850+1, f ;
  goto    Add1      ;
  incfsz  AD9850+2, f ;
  goto    Add1      ;
  incf    AD9850+3, f ;
Add1
  movf    OP1+1, w  ;
  addwf   AD9850+1, f ;
  btfss   Status, C ; See if Carry is set
  goto    Add2      ; Carry is clear, so jump
  incfsz  AD9850+2, f ;
  goto    Add2      ;

```

DDS V1. asm

```

Add2  incf    AD9850+3, f      ;
      movf   OP1+2, w        ;
      addwf  AD9850+2, f      ;
      btfss  Status, C       ; See if Carry is set
      goto  Add3             ; Carry is clear, so jump
      incf   AD9850+3, f      ;
Add3  movf   OP1+3, w        ;
      addwf  AD9850+3, f      ;
      return ;

```

\*\*\*\*\*

```

; Write Data
;
; This subroutine does two things:
; (1) it sends the contents LCD_dat to the LCD,
;     (LCD_Dat must contain BCD digits)
; (2) the contents of AD9850 (40 bits) are serial shifted
;     into the AD9850 module
;

```

```

WriteData
  movl w 0x83           ; LCD position for first digit
  movwf temp2
  call BusyCheck
  bcf PortB, LCD_rs    ; Set LCD for instruction write
  call SendChar
  movf LCD_Dat+0, w
  movwf temp2
  movl w 0x20           ; Send a space if first digit is zero
  iorwf temp2, f
  movl w 0x20           ; Set up for space check
  subwf temp2, w       ; Subtract to compare
  btfsc Status, Z      ; See if Zero is clear (not a space)
  goto Wd2             ; Zero is set, so go send the space
  movl w 0x10           ; Zero is clear, so convert BCD
  iorwf temp2, f       ; to ASCII
Wd2  call BusyCheck
  call SendChar        ; Send 10Mhz digit
  movf LCD_Dat+1, w
  movwf temp2
  movl w 0x30           ;
  iorwf temp2, f
  call BusyCheck
  call SendChar        ; Send 1Mhz digit
  movl w 0x2C           ;
  movwf temp2
  call BusyCheck
  call SendChar        ; Send a comma
  movf LCD_Dat+2, w
  movwf temp2
  movl w 0x30           ;
  iorwf temp2, f
  call BusyCheck
  call SendChar        ; Send a 100KHz digit
  movf LCD_Dat+3, w
  movwf temp2
  movl w 0x30           ;
  iorwf temp2, f
  call BusyCheck      ; Send 10 KHz digit
  call SendChar
  movl w 0xC0           ; Point LCD at digit#9 (Gap in LCD address)

```

DDS V1. asm

```

movwf temp2
call BusyCheck ;
bcf PortB, LCD_rs ; Set LCD for instruction write
call SendChar ;
movf LCD_Dat+4, w ;
movwf temp2
movlw 0x30
iorwf temp2, f
call BusyCheck
call SendChar ; Send 1Khz digit
movlw 0x2E
movwf temp2
call BusyCheck
call SendChar ; Send a period
movf LCD_Dat+5, w ;
movwf temp2
movlw 0x30
iorwf temp2, f
call BusyCheck
call SendChar ; Send 100Hz digit
movf LCD_Dat+6, w ;
movwf temp2
movlw 0x30
iorwf temp2, f
call BusyCheck
call SendChar ; Send 10Hz digit
WriteAD9850
clrf PortB ; Clear PortB
movlw 0x05 ; Send 5 bytes
movwf temp3 ; to AD9850
movlw 0x00 ;
tris PortB ; Enable PortB
movlw 0x13 ; Point at AD9850+0
movwf FSR
NextByte
movf Indirect, w ;
movwf temp1
movlw 0x08 ; set bit counter to 8
movwf temp2
NextBit
rrf temp1, f
btfss Status, C ; See if carry is set
goto Send0 ; Carry is clear, so jump
bsf PortB, AD9850_dat
bsf PortB, AD9850_wc ; toggle write clock
bcf PortB, AD9850_wc ; (repeated 40 times)
goto Break
Send0
bcf PortB, AD9850_dat
bsf PortB, AD9850_wc
bcf PortB, AD9850_wc
Break
decfsz temp2, f
goto NextBit
incf FSR, f
decfsz temp3, f
goto NextByte
bsf PortB, AD9850_fqu ; send load signal to AD9850
bcf PortB, AD9850_fqu ; Clear after last data dbit is written
return

```

\*\*\*\*\*  
; 7 Digit BCD by 32 bit binary multiply

DDS V1.asm

; The 7 BCD digits stored in LCD\_dat are each multiplied by a precalculated digit weight, (Digit weights assume a 66.666MHz input clock). The sum of all 7 digits times their weight is stored in the AD9850 registers.

Mul t

```

cl rf    temp1
cl rf    AD9850+0
cl rf    AD9850+1
cl rf    AD9850+2
cl rf    AD9850+3
cl rf    AD9850+4

```

Mul t10

```

movl w   0x84           ;
movwf   OP1+0
movl w   0x02           ;
movwf   OP1+1
cl rf   OP1+2
cl rf   OP1+3
movf    LCD_Dat+6, w   ;
movwf   OP2            ;
goto    Go

```

Mul t100

```

movl w   0x2A           ;
movwf   OP1+0           ;
movl w   0x19           ;
movwf   OP1+1           ;
cl rf   OP1+2
cl rf   OP1+3
movf    LCD_Dat+5, w   ;
movwf   OP2
goto    Go

```

Mul t1K

```

movl w   0xA9           ;
movwf   OP1+0           ;
movl w   0xFB           ;
movwf   OP1+1           ;

cl rf   OP1+2
cl rf   OP1+3
movf    LCD_Dat+4, w   ;
movwf   OP2
goto    Go

```

Mul t10K

```

movl w   0x9B           ;
movwf   OP1+0           ;
movl w   0xD4           ;
movwf   OP1+1           ;
movl w   0x09           ;
movwf   OP1+2           ;
cl rf   OP1+3
movf    LCD_Dat+3, w   ;
movwf   OP2
goto    Go

```

Mul t100K

```

movl w   0x19           ;
movwf   OP1+0           ;
movl w   0x4E           ;
movwf   OP1+1           ;
movl w   0x62           ;
movwf   OP1+2           ;
cl rf   OP1+3
movf    LCD_Dat+2, w   ;

```

```

movwf OP2 ;
goto Go
Mul t1M
movl w 0xC1 ;
movwf OP1+0 ;
movl w 0x0C ;
movwf OP1+1 ;
movl w 0xD7 ;
movwf OP1+2 ;
movl w 0x03 ;
movwf OP1+3 ;
movf LCD_Dat+1, w ;
movwf OP2 ;
goto Go
Mul t10M
movl w 0x90 ;
movwf OP1+0 ;
movl w 0x7F ;
movwf OP1+1 ;
movl w 0x66 ;
movwf OP1+2 ;
movl w 0x26 ;
movwf OP1+3 ;
movf LCD_Dat+0, w ;
movwf OP2 ;
goto Go
Go
movl w 0x04
movwf temp2
Di gi tLoop
bcf Status, C ; Clear Carry bit
rrf OP2, f ; Rotate right (through carry)
btfss Status, C ; See if Carry is set
goto NoCarry ; Carry is clear, so jump
call Add32
NoCarry
bcf Status, C ; Clear Carry bit
rlf OP1+0, f ; Rotate left through carry
rlf OP1+1, f ;
rlf OP1+2, f ;
rlf OP1+3, f ;
decfsz temp2, f
goto Di gi tLoop
incf temp1, f
movl w 0x01
subwf temp1, w
btfsc Status, Z ; See if Zero is clear
goto Mul t100 ; Zero is set, so jump
movl w 0x02
subwf temp1, w
btfsc Status, Z ; See if Zero is clear
goto Mul t1K ; Zero is set, so jump
movl w 0x03
subwf temp1, w
btfsc Status, Z ; See if Zero is clear
goto Mul t10K ; Zero is set, so jump
movl w 0x04
subwf temp1, w
btfsc Status, Z ; See if Zero is clear
goto Mul t100K ; Zero is set, so jump
movl w 0x05
subwf temp1, w
btfsc Status, Z ; See if Zero is clear

```

```

                                DDS V1.asm
goto    Mul t1M                ; Zero is set, so jump
movl w  0x06
subwf   temp1, w
btfsc   Status, Z              ; See if Zero is clear
goto    Mul t10M               ; Zero is set, so jump
return

```

```

; *****
; Initialize PIC controller
;
; Start here after reset or power on
; Set DDS initially to 10,000.00

```

Start

```

clrf    PortA                  ; Clear data Port A
clrf    PortB                  ; Clear data Port B
movl w  0x00
tris    PortB                  ; Enable port B drivers
movl w  0xFF
tris    PortA                  ; Tristate Port A drivers
movl w  0x01
movwf   LCD_Dat+0
movl w  0x00
movwf   LCD_Dat+1
movl w  0x00
movwf   LCD_Dat+2
movl w  0x00
movwf   LCD_Dat+3
movl w  0x00
movwf   LCD_Dat+4
movl w  0x00
movwf   LCD_Dat+5
movl w  0x00
movwf   LCD_Dat+6
clrf    count1
clrf    count2

```

```

; *****
; Power on initialization of Liquid Crystal Display.
; The LCD controller chip must be equivalent to an Hitachi 44780.
; The LCD is assumed to be a 16 X 1 display.

```

```

movl w  0x07                  ; Enable LCD_VDD pin
tris    PortA                  ; power up LCD
bsf     PortA, LCD_VDD
call    Wait60ms              ; Wait for LCD to power up
bsf     PortB, LCD_e
movl w  0x38
movwf   PortB
bcf     PortB, LCD_e
call    Wait60ms
bsf     PortB, 3
movl w  0x38
movwf   PortB
bcf     PortB, LCD_e
call    Wait60ms
bsf     PortB, LCD_e
movl w  0x38
movwf   PortB
bcf     PortB, LCD_e
call    Wait60ms
bsf     PortB, LCD_e
movl w  0x38
movwf   PortB
bcf     PortB, LCD_e
call    Wait60ms
bsf     PortB, LCD_e
movl w  0x28                  ; 4bit_mode instruction + enable

```

DDS V1.asm

```

movwf  PortB
bcf    PortB, LCD_e
call   Wait60ms
movlw  0x01           ; Clear and reset cursor
movwf  temp2
call   BusyCheck
bcf    PortB, LCD_rs
call   SendChar
movlw  0x06           ; increment no_shift mode
movwf  temp2
call   BusyCheck
bcf    PortB, LCD_rs
call   SendChar
movlw  0x0C           ; Display on
movwf  temp2
call   BusyCheck
bcf    PortB, LCD_rs
call   SendChar

```

; Send initial 10,000.00 display to LCD

```

call   Mult
call   Wait24ms       ; allow AD9850 clock time to start
call   Wait24ms       ;
call   WriteData      ; side effect reset power down bits of AD9850
call   WriteData      ; load the initial display value

```

; Get the power on encoder value

```

movf   PortA, w       ; get encoder value
movwf  temp2
movlw  0x03
andwf  temp2, w       ; isolate direction bits
movwf  old            ; save initial values

```

; \*\*\*\*\*

```

; Check encoder
; PortA.0 shaft encoder output A
; PortA.1 shaft encoder output B
; PortA.2 shaft encoder push button switch
; Timing checks assume a 4MHz cpu clock and that the encoder
; has 32 detents

```

ChkEncoder

```

incf   count1, f
movlw  0x75           ; Check encoder loop takes 17 us
subwf  count1, w
btfss  Status, C     ; See if Carry is set
goto   Ce            ; Carry is clear, so jump
incf   count2, f
clrf   count1        ; 0x75 * 17us = 2ms

```

Ce

```

movf   PortA, w       ; Get encoder value
movwf  temp1
movf   temp1, w
movwf  temp2         ; Double latch the input
movlw  0x03
andwf  temp2, w
movwf  new
xorwf  old, w
btfsc  Status, Z     ; See if Zero is clear
goto   ChkEncoder    ; Zero is set, so jump

```

```

                                DDS V1.asm
movl w    0x0E                    ; Pointer set to 100KHz digit
movwf    FSR
btfss    temp2, 2                ; If pb switch is closed (active low)
goto     CeGo                    ;
movl w    0x12                    ; Pointer set to 10's
movwf    FSR
movl w    0x0F
subwf    count2, w              ; If speed < 1.0 RPM
btfsc    Status, C              ; See if Carry is clear
goto     CeGo                    ; Carry is set, so jump
movl w    0x11                    ; Pointer set to 100's
movwf    FSR
movl w    0x05
subwf    count2, w              ; If (1.0RPM < Speed < 3.0RPM)
btfsc    Status, C              ; See if Carry is clear
goto     CeGo                    ; Carry is set, so jump
movl w    0x10                    ; Pointer set to 1K
movwf    FSR
CeGo
clrf     count2                  ; Add time to write LCD etc to count1
movl w    0x46                    ; (about 1.2 ms)
movwf    count1
bcf      Status, C              ; Clear Carry bit
rlf      old, f                  ; Rotate left
movf     new, w
xorwf    old, f                  ; Determine direction
btfsc    old, DC                ; See if Digit Carry is clear
goto     CeUp                    ; Digit Carry is set, so jump
call     BCDDec                  ; Calculate new LCD display
call     Mult                    ; Calculate new AD9850 control word
goto     CeWrite
CeUp
call     BCDInc                  ; calculate new LCD display
call     Mult                    ; calculate new AD9850 control word
CeWrite
call     WriteData
movf     new, w
movwf    old
goto     ChkEncoder              ; continue polling the encoder

; ID
; ID
; ID
; ID
; Fuses (CP=0ff, PWRTE=Enabl ed, WDTE=Di sabl ed, OSC=XT)
; 0001
;

END

```