```
;*****************************************************************
;   PCITIMER assembly code                                       *
;                                                                *
;*****************************************************************
;                                                                *
;    Filename:      pcitimer.asm                                 *
;    Date:                                                       *
;    File Version:                                               *
;                                                                *
;    Author:       Mitch Randall                                 *
;    Company:      BINET, Inc                                    *
;                                                                *
;                                                                *
;*****************************************************************
;                                                                *
;    Files required:                                             *
;                                                                *
;*****************************************************************
;                                                                *
;    Notes: This version implements all 3 timer modes as of 10/22/01  *
;           The change requires the latest firmware (10/22/01) in the *
;           EPLD                                                 *
;                                                                *
;*****************************************************************


    list      p=16f874            ; list directive to define processor
    #include <p16f874.inc>        ; processor specific variable definitions

;   __CONFIG _CP_OFF & _WDT_ON & _BODEN_ON & _PWRTE_ON & _HS_OSC & _WRT_ENABLE_ON &
_LVP_ON & _CPD_OFF
; this is the latest one
    __CONFIG _CP_OFF & _WDT_ON & _BODEN_ON & _PWRTE_ON & _HS_OSC & _WRT_ENABLE_ON &
_LVP_OFF & _CPD_OFF

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

    #include    "picdpram.h"

;***** CONSTANT DEFINITIONS
; PORTC definitions
RD      EQU 0
WR      EQU 1
ALATCH      EQU 2
SPARE       EQU 4

; PORTB definitions
```

```
DLATCH      EQU 1
PLL_LE      EQU 2
A8      EQU 3
A9      EQU 4
PCIINT      EQU 5
PULSEEN     EQU 6
PPSCLR      EQU 7

; PORTE definitions
DIV0        EQU 0
DIV1        EQU 1
DIV2        EQU 2

;***** VARIABLE DEFINITIONS
w_temp      EQU 0x20    ; variable used for context saving
status_temp EQU 0x21    ; variable used for context saving

timer_lo    EQU 0x22    ; timer count low byte
timer_hi    EQU 0x23    ; timer count high byte
timer_chip  EQU 0x24    ; timer chip number
timer_num   EQU 0x25    ; timer number within chip
timer_mode  EQU 0x26    ; timer mode

rw_data     EQU 0x27
rw_add_lo   EQU 0x28
rw_add_hi   EQU 0x29
rw_chipsel  EQU 0x2A

tempi       EQU 0x2B

int_cnt     EQU 0x2C
int_max_cnt EQU 0x2D

mem_ptr_lo  EQU 0x2E
mem_ptr_hi  EQU 0x2F
mem_data    EQU 0x30

pll_lo      EQU 0x31
pll_md      EQU 0x32
pll_hi      EQU 0x33

wait_cnt_lo EQU 0x34
wait_cnt_hi EQU 0x35

tempj       EQU 0x36
tempk       EQU 0x37
templ       EQU 0x38
timertemp   EQU 0x39
```

```
tempaddlo   EQU 0x3A
tempaddhi   EQU 0x3B
tempw       EQU 0x3C

tempmain    EQU 0x3D


; various chip selects
PERIOD      EQU 0x04
DPRAM       EQU 0x05

; some REQUEST ID's
ID_RESET    EQU 0x00
ID_STOP     EQU 0x01
ID_START    EQU 0x02

;*****************************************************************
; THE BEGINING OF IT ALL
;*****************************************************************
    ORG     0x000             ; processor reset vector
    clrf    PCLATH            ; ensure page bits are cleared
    goto    main              ; go to beginning of program

    ORG     0x004           ; interrupt vector location
    movwf   w_temp          ; save off current W register contents
    movf    STATUS,w        ; move status register into W register
    bcf     STATUS,RP0      ; ensure file register bank set to 0
    movwf   status_temp     ; save off contents of STATUS register
    btfsc   INTCON,INTF ; test for external interrupt
    call    timerisr    ; if so, service
;   btfsc   INTCON,RBIF ; test for PCI interrupt
;   call    pciisr      ; if so, service
    bcf     STATUS,RP0      ; ensure file register bank set to 0
    movf    status_temp,w   ; retrieve copy of STATUS register
    movwf   STATUS          ; restore pre-isr STATUS register contents
    swapf   w_temp,f
    swapf   w_temp,w        ; restore pre-isr W register contents
    retfie                  ; return from interrupt

; Here is declared sequence and configuration data
; it is located near ORG 0 because the FSR cannot see past address 256
; Note: keep this away from any page break

;*****************************************************************
;           CONFIG DEFINITION
;*****************************************************************
;
; This memory sets up the timers and the pll
; According to a comment below, the timers are located at offest zero in configmemory
```

```
; The first 2 bytes are the low, then high bytes to program the pulsewidth.
; The next 2 bytes are the low, then high bytes for the test pulse delay.
; This repeats for the six timer pairs

configmem
    addwf   PCL,f
; delay_lo, delay_hi, pulsewidth_lo, pulsewidth_hi
    dt  0x10,0x00,0x07,0x00,0x10,0x00,0x07,0x00 ; pretrig 1 and 2
    dt  0x70,0x3B,0x4F,0x00,0x70,0x3B,0x4F,0x00 ; 10us H,V test pulse at 1.9 ms
    dt  0x60,0x00,0x07,0x00,0x60,0x00,0x07,0x00 ; pulse spare
; freqlo, freqmed, freqhi, reflo, refmed, refhi, div, spare
    dt  0x01,0x50,0x10,0x20,0x03,0x10,0x00,0x00
; synclo, synchi, seqdello, seqdelhi, seqlen, seqcnt, timermode
    dt  0x40,0x1F,0x50,0x00,0x01,0x00,0x00    ; timingmode = 2 (sync)
configlen
    dt  dpram_timermode + 1 ; this is the address of the last thing in the list

;*****************************************************************
;           SEQUENCE DEFINITION
;*****************************************************************
;
; Each step of the sequence is defined with 5 bytes
; BYTE0: LOW  BYTE OF PULSE PERIOD 8MHZ COUNTS
; BYTE1: HIGH BYTE OF PULSE PERIOD 8MHZ COUNTS
; BYTE2: INVERTED PULSE ENABLES, The ts2 and ts1 bits are not inverted and have no
delay!
;                          TS2  TS1  PS2  PS1  VTP  HTP XPT2 XPT1
; BYTE3: POLARIZATION SEL (PS) SEQ SPARE (SS) HV CHAN SEL (HV)
;                         HV-V HV-H  SS3  SS2  SS1  PS3  PS2  PS1
; BYTE4: PHASE DATA (PD)  PD8  PD7  PD6  PD5  PD4  PD3  PD2  PD1
; The sequencelen byte is the total number of sequence bytes

sequencemem
    addwf   PCL,f
    dt  0x40,0x1F,0xCF,0x85,0x88,0x3F,0x1F,0xFF,0x85,0x40  ; PLUS 45
    dt  0x3F,0x1F,0xFF,0x03,0x20,0x3F,0x1F,0xFF,0x02,0x14
    dt  0x3F,0x1F,0xFF,0x05,0x00,0x3F,0x1F,0xFF,0x04,0xF0
    dt  0x3F,0x1F,0xFF,0x07,0xA2,0x3F,0x1F,0xFF,0x06,0x50
    dt  0x3F,0x1F,0xFF,0x03,0xC0,0x3F,0x1F,0xFF,0x02,0x31
    dt  0x3F,0x1F,0xFF,0x05,0x70,0x3F,0x1F,0xFF,0x04,0x60
sequencelen
    dt  0x3C    ; this sequence is 60 bytes long (5 * 12)

;*****************************************************************
; Put an address out on the bus
; Assume that page zero is selected on entry
; The address is in the rw_add_lo, rw_add_hi register on entry
address
    ; turn PORTD to an output
```

```
    bsf STATUS,RP0  ;Select bank 1
    clrf    TRISD       ;Set PORT D to outputs

    ; put the data on PORTD
    bcf STATUS,RP0  ;Select bank 0
    movf    rw_add_lo,w
    movwf   PORTD

    ; toggle the latch line
    bsf PORTC,ALATCH    ; go up
    bcf PORTC,ALATCH    ; go down

    ; restore PORTD to inputs
    bsf STATUS,RP0  ;Select bank 1
    movlw   0xFF
    movwf   TRISD       ;Set PORT D to inputs
    bcf STATUS,RP0  ;set back to bank 0

    ; set the high bits
    bcf PORTB,A8    ; clear A8
    bcf PORTB,A9    ; clear A9
    btfsc   rw_add_hi,0 ; look at rw_add_hi, 0
    bsf PORTB,A8    ; set A8
    btfsc   rw_add_hi,1 ; look at rw_add_hi, 1
    bsf PORTB,A9    ; set A9

    return

;*****************************************************************************
; put the w register out the SPI
spiout
    bcf STATUS,RP0  ; go to page 0
    movwf   SSPBUF
    bsf STATUS,RP0  ; go to page 1
spi1    btfss   SSPSTAT,BF  ; wait for it to complete
    goto    spi1
    bcf STATUS,RP0  ; go to page 0
    movf    SSPBUF,w    ; read the buffer
    return

;*****************************************************************************
; Increment the address
; This is often necessary and will come in handy
addr_inc
    incf    rw_add_lo,f
    btfsc   STATUS,Z
    incf    rw_add_hi,f
    return
```

```
;*****************************************************************
chip_select ; return the bit pattern for a particular chip select on PORTA
    addwf   PCL,f
    dt  0xFE,0xFD,0xFB,0xF3,0xEF,0xDF

;*****************************************************************
; write a data byte to a particular address and chip select
; on entry the data is in the data register,
; the address is in the address register, and the chip number is
; in the chip select register
; it is assumed that bank 0 is already set
write_address
    ; latch the address to the address register
    call    address

    ; put the data on the data bus
    bsf STATUS,RP0  ;Select bank 1
    clrf    TRISD       ;Set PORT D to outputs
    bcf STATUS,RP0  ;Select bank 0
    movf    rw_data,w
    movwf   PORTD

    ; toggle the write line low
    bcf PORTC,WR

    ; assert the chip select
    movf    rw_chipsel,w    ; get the chip number
    call    chip_select ; figure out the chip select bit pattern
    movwf   PORTA

    ; take away the chip select
    movlw   0xFF
    movwf   PORTA

    ; toggle the write line high
    bsf PORTC,WR

    ; now return data direction to defaults
    bsf STATUS,RP0  ;Select bank 1
    movlw   0XFF        ;Set data direction bits (all inputs)
    movwf   TRISD       ;Move to port D control register
    bcf STATUS,RP0  ;Select bank 0

    return

;*****************************************************************
; read a data byte from a particular address and chip select
; on return the data is in the data register,
; the address is in the address register, and the chip number is
```

```
; in the chip select register
; it is assumed that portD is already set as an input
; it is assumed that rw is already high
; it is assumed that bank 0 is already set
read_address
    ; latch the address to the address register
    call    address

    ; assert the chip select
    movf    rw_chipsel,w    ; get the chip number
    call    chip_select ; figure out the chip select bit pattern
    movwf   PORTA

    movf    PORTD,w     ;get the data
    movwf   rw_data     ;put it in the data register

    ; take away the chip select
    movlw   0xFF
    movwf   PORTA

    movf    rw_data,w   ;put it in the data register

    return


;*****************************************************************************
; skip every fourth DPRAM address
; save address in temp, multiply address by four
dpramfix
    movf    rw_add_hi,w ; save address to temparary locatons
    movwf   tempaddhi
    movf    rw_add_lo,w
    movwf   tempaddlo

        ; shift the address over to fix the PCI data addressing problem
    ; we'll end up throwing away all the hi address bits (they get shifted away)
    ; This shift effectively reduces the memory size to 256 bytes (not 1024)
    bcf STATUS,C    ; clear the carry
    rlf rw_add_lo,f ; multiply by 2 place back in register
    rlf rw_add_hi,f ; shift into hi byte
    bcf STATUS,C    ; clear the carry
    rlf rw_add_lo,f ; multiply by 2 place back in register
    rlf rw_add_hi,f ; shift into hi byte

    return

;*****************************************************************************
; restore the address
; remembers the w register, too
```

```
undpramfix
    movwf   tempw       ; save the contents of the w register
    movf    tempaddhi,w ; restore address from temparary locatons
    movwf   rw_add_hi
    movf    tempaddlo,w
    movwf   rw_add_lo
    movf    tempw,w     ; restore contents of w register

    return

;*****************************************************************
; read a value from config mem
; the offset is in the w register
readsequence
    movwf   rw_add_lo
    clrf    rw_add_hi
   bcf rw_add_lo,6
   bcf rw_add_lo,7
    movlw   DPRAM
    movwf   rw_chipsel
   call dpramfix
    call    read_address
   call undpramfix

    return

;*****************************************************************
; read a value from config mem
; the offset is in the w register
readconfig
    movwf   rw_add_lo
    movlw   0x03
    movwf   rw_add_hi
   bsf rw_add_lo,6      ; config is 39 bytes. Put it at 192 of 256
   bsf rw_add_lo,7      ; (space = 64)
   clrf rw_add_hi
    movlw   DPRAM
    movwf   rw_chipsel
   call dpramfix
    call    read_address
   call undpramfix
    return

;*****************************************************************
control_byte    ;return the control word to select a timer num
    addwf   PCL,f
    dt  0x30,0x70,0xB0

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

```
; timer_lo, timer_hi, timer_chip, timer_num, timer_mode
timer   ; program the timer

    ; set the chip number once
    movf    timer_chip,w    ; get timer chip number
    movwf   rw_chipsel  ; put it in rw chipsel reg

    ; get the timer mode and double it (for control register use)
    movf    timer_mode,w    ; get the timer mode
    movwf   timertemp
    bcf STATUS,C    ; clear the carry
    rlf timertemp,f

    ; write the control register
    movf    timer_num,w ; get the timer number
    call    control_byte    ; figure out the control byte
    addwf   timertemp,w ; add twice the mode to the control word
    movwf   rw_data     ; put it in data register
    movlw   0x03        ; write the control register
    movwf   rw_add_lo   ; put it in address
    call    write_address   ; note: hi add is ignored

    ; write the low byte (to the right spot)
    movf    timer_lo,w  ; get the low byte
    movwf   rw_data     ; put it in data register
    movf    timer_num,w ; get the timer number
    movwf   rw_add_lo   ; put it in address
    call    write_address   ; note: hi add is ignored

    ; write the high byte (to the right spot)
    movf    timer_hi,w  ; get the high byte
    movwf   rw_data     ; put it in data register
    movf    timer_num,w ; get the timer number
    movwf   rw_add_lo   ; put it in address
    call    write_address   ; note: hi add is ignored

    return


;*****************************************************************
; Do the required tasks on each interrupt
; uses: rw_add_lo, rw_add_hi, rw_chipsel, int_cnt, mem_ptr_lo, mem_ptr_hi
timerisr
    movlw   dpram_timermode ; timermode 0
    call    readconfig
    sublw   0x02        ; if timermode = 2 (sync)
    btfsc   STATUS,Z
    bcf PORTC,SPARE ; set to internal triggers
```

```
    ; compute the array pointer
    movf    int_cnt,w   ; multiply by 5
    movwf   rw_add_lo
    bcf STATUS,C
    rlf rw_add_lo,f ; rotate left (through carry)
    bcf STATUS,C
    rlf rw_add_lo,w ; rotate left (put in W)
    addwf   int_cnt,w   ; add int_cnt
    movwf   mem_ptr_lo  ; save result in address register
    clrf    mem_ptr_hi  ; set upper addr to zero (page 0)

    movf    mem_ptr_lo,w    ; get the first of two bytes from dual port memory for the
timer
    call    readsequence    ; data from array[ptr] in rw_data
    movlw   0x01        ; write the low byte (to the right spot)
    movwf   rw_add_lo   ; period timer is address 1
    movlw   PERIOD      ; period timer chip number
    movwf   rw_chipsel  ; put it in rw chipsel reg
    call    write_address   ; note: hi add is ignored

    incf    mem_ptr_lo,f    ; increment the array pointer
    btfsc   STATUS,Z
    incf    mem_ptr_hi,f
    movf    mem_ptr_lo,w    ; get second of two bytes from dual port memory for the
timer
    call    readsequence    ; data from array[ptr] in rw_data
    movlw   0x01        ; write the high byte (to the right spot)
    movwf   rw_add_lo   ; period timer is address 1
    movlw   PERIOD      ; period timer chip number
    movwf   rw_chipsel  ; put it in rw chipsel reg
    call    write_address   ; note: hi add is ignored

    ; get three bytes and put into sequence latch
    incf    mem_ptr_lo,f    ; increment the array pointer
    btfsc   STATUS,Z
    incf    mem_ptr_hi,f
    movf    mem_ptr_lo,w    ; get next byte from dual port memory
    call    readsequence    ; data from array[ptr] in rw_data
    movf    rw_data,w   ; put it out on the SPI
    call    spiout

    incf    mem_ptr_lo,f    ; increment the array pointer
    btfsc   STATUS,Z
    incf    mem_ptr_hi,f
    movf    mem_ptr_lo,w    ; get next byte from dual port memory
    call    readsequence    ; data from array[ptr] in rw_data
    movf    rw_data,w   ; put it out on the SPI
    call    spiout
```

```
    incf    mem_ptr_lo,f    ; increment the array pointer
    btfsc   STATUS,C
    incf    mem_ptr_hi,f
    movf    mem_ptr_lo,w    ; get next byte from dual port memory
    call    readsequence    ; data from array[ptr] in rw_data
    movf    rw_data,w   ; put it out on the SPI
    call    spiout
    bsf PORTB,DLATCH    ; latch the data into the sequence register output
    bcf PORTB,DLATCH

    ; increment interrupt counter
    incf    int_cnt,f   ; increment int_cnt in place
    movf    int_max_cnt,w   ; get into W register
    subwf   int_cnt,w   ; subtract w from f (max_cnt from int_cnt)
    btfsc   STATUS,C    ; test if cnt >= max_cnt
    clrf    int_cnt     ; if so, clear it

    goto isr0

        ; this code is what used to be executed.
    ; Now we are going to always do the polling in the main routine.

        movlw   0x3F        ; read the PCI request flag from DPRAM and store it in local
memory
    call    readconfig
;   movwf   request
;   btfss   request,0   ; a request causes timer and interrupts to halt
    goto    isr0
;   movlw   0x01
;   movwf   stopped     ; report that the system has stopped
    call    wait
    bcf INTCON,INTE ; disable external interrupt
    bcf PORTB,PULSEEN   ; stop the timers
    bsf PORTB,PPSCLR    ; This clears the sync flip-flop
    bcf PORTB,PPSCLR    ; to shut off "GATE0" (And stop the PRT timer)
    bcf INTCON,INTF ; clear interrupt (not sure if this is needed)
    bcf INTCON,GIE  ; now turn off global ints (not a good reason, perhaps)
    return

isr0


; I hope this eventually goes away
   ; keep reading the sequence length
;   movlw   dpram_seqlength ;get the sequence length pointer for DPRAM
;   call    readconfig  ;get config data from that offset
;   movwf   int_max_cnt ;save in local copy of max_cnt

    bcf INTCON,INTF ; clear interrupt
```

```
    return


;*****************************************************************
pciisr
    ; check to see what command was requested
    ; (and clear the DPRAM interrupt)
;   movlw   0xFF
;   call    readconfig

    ; case statements

    bcf INTCON,RBIF ; clear interrupt
    return



;*****************************************************************
; wait for 50 ms (= 50000 loops of 4 instructions)
wait    movlw   0x50        ; low wait count
    movwf   wait_cnt_lo
    movlw   0xC3        ; high wait count
    movwf   wait_cnt_hi
wait1   nop
    decfsz  wait_cnt_lo,f
    goto    wait1
    decfsz  wait_cnt_hi,f
    goto    wait1
    clrwdt          ; clear the watchdog timer
    return


;*****************************************************************
; write to the phase locked loop chip
; data comes from the dpram_synth area
; This assumes LE is low already
pll
    movf    pll_hi,w    ; put out high byte
    call    spiout
    movf    pll_md,w    ; put out medium byte
    call    spiout
    movf    pll_lo,w    ; put out low byte
    call    spiout

    bsf PORTB,PLL_LE    ; toggle PLL latch enable line
    bcf PORTB,PLL_LE

    return
```

```
;*****************************************************************
; initialize the dpram config memory with source code declared values
initdpram

    ; first initialize all dpram to zero :)
    movlw   DPRAM
    movwf   rw_chipsel
    clrf    rw_add_lo
    clrf    rw_add_hi
    clrf    rw_data
dpram0  call    write_address
    incfsz  rw_add_lo,f
    goto    dpram0
    incf    rw_add_hi,f
    movf    rw_add_hi,w
    sublw   2       ; stay away from the last address (do 3/4)
    btfsc   STATUS,C
    goto    dpram0

dploop  call    write_address   ; do almost up to the last one
    incf    rw_add_lo,f
    movf    rw_add_lo,w
    sublw   0xFC        ; don't do the last 3
    btfsc   STATUS,C
    goto    dploop

    ; initialize the write address
    clrf    rw_add_lo
    movlw   0x03
    movwf   rw_add_hi
;  movlw    0xC0
;  movwf    rw_add_lo
;  clrf rw_add_hi
    movlw   DPRAM
    movwf   rw_chipsel

    ; do a loop here
    call    configlen
    movwf   tempi
    clrf    int_cnt     ; just use this as a temp pointer
dpram1  movf    int_cnt,w   ; put pointer in W
    call    configmem   ; do lookup table
    movwf   rw_data     ; stick it in data reg
    call    write_address   ; put in DPRAM
    call    addr_inc    ; increment DPRAM pointer
   call addr_inc    ; increment DPRAM pointer
   call addr_inc    ; increment DPRAM pointer
   call addr_inc    ; increment DPRAM pointer
    incf    int_cnt,f   ; increment declared data pointer
```

```
    decfsz  tempi,f
    goto    dpram1

    ; initialize the write address
    clrf    rw_add_lo
    clrf    rw_add_hi

    ; do a loop here
    call    sequencelen
    movwf   tempi
    clrf    int_cnt     ; just use this as a temp pointer
dpram2  movf    int_cnt,w   ; put pointer in W
    call    sequencemem ; do lookup table
    movwf   rw_data     ; stick it in data reg
    call    write_address   ; put in DPRAM
    call    addr_inc    ; increment DPRAM pointer
   call addr_inc    ; increment DPRAM pointer
   call addr_inc    ; increment DPRAM pointer
   call addr_inc    ; increment DPRAM pointer
    incf    int_cnt,f   ; increment declared data pointer
    decfsz  tempi,f
    goto    dpram2

    return


;*****************************************************************************
; initialize the six timers with the configuration data
; timer init data starts at zero in config mem
timerinit
    movlw   6
    movwf   tempi       ;use temp as an index (6 timers)
    clrf    tempj       ;the chip select index
    clrf    tempk       ;the timer number of chip index
    clrf    templ       ;the memory point

    ;program delay
timer1  movf    templ,w
    call    readconfig  ;get timer config delay low data
    movwf   timer_lo    ;put it to timer variable
    incf    templ,f

    movf    templ,w
    call    readconfig  ;get timer config delay high data
    movwf   timer_hi    ;put it to timer variable
    incf    templ,f

    movf    tempj,w     ;the timer chipselect number
    movwf   timer_chip  ;set the timer chipselect number
```

```
    movf    tempk,w     ;the timer number within chip
    movwf   timer_num   ;set the timer number within chip
    movlw   0x05        ;set the timer mode (one shot)
    movwf   timer_mode  ;put it to timer variable
    call    timer

    ;program pulsewidth
    movf    templ,w
    call    readconfig  ;get timer config pulsewidth low data
    movwf   timer_lo    ;put it to timer variable
    incf    templ,f

    movf    templ,w
    call    readconfig  ;get timer config pulsewidth high data
    movwf   timer_hi    ;put it to timer variable
    incf    templ,f

    movf    tempj,w     ;the timer chipselect number
    addlw   0x01        ;the pulsewidth chipselect is one higher than the delay
chipselect
    movwf   timer_chip  ;set the timer chipselect number
    movf    tempk,w     ;the timer number within chip
    movwf   timer_num   ;set the timer number within chip
    movlw   0x01        ;set the timer mode (pulsewidth)
    movwf   timer_mode  ;put it to timer variable
    call    timer

    ;loop housekeeping
    incf    tempk,f     ;now work on the next highest timer within chip
    movlw   0x03        ;if chipnumber bigger than 2, then reset and double inc chipsel
    subwf   tempk,w
    btfss   STATUS,C    ;test if tempk < 0x03
    goto    tskip       ;if so, then skip to tskip
    clrf    tempk       ;reset to timer number 0
    incf    tempj,f     ;chipsel += 2
    incf    tempj,f
tskip   decfsz  tempi,f     ;check to see if we're done
    goto    timer1

    return          ;done!!!


; The service request flag is in config 0x3F
; The request ID is in config 0x3E
; The request ID describes which function has been requested from the timer card
; This routine is a case statment switching on ID

; Service requests are polled by the interrupt routine
```

```
service
    ; STOP COMMAND
    movlw   0x3E        ;get the request ID
    call    readconfig  ;get config data from that offset
    sublw   ID_STOP
    btfss   STATUS,Z    ;skip if ID_STOP
    goto    serv0

    call    stop
    call    clear_request
    return

serv0
    ; RESET COMMAND
    movlw   0x3E        ;get the request ID
    call    readconfig  ;get config data from that offset
    sublw   ID_RESET
    btfss   STATUS,Z    ;don't skip if ID_RESET
    goto serv1

        call    initsystem  ; Initialize the system with the contents of the DPRAM
    call    clear_request
    return

serv1
    ;START COMMAND
    movlw   0x3E        ;get the request ID
    call    readconfig  ;get config data from that offset
    sublw   ID_START
    btfss   STATUS,Z    ;don't skip if ID_START
    goto serv2

    call    clear_request
    call    start
    return

serv2
    ; UNIMPLEMENTED COMMAND or just plain done
    call    clear_request
    return

clear_request
    movlw   DPRAM       ; set the done bit in DPRAM!
    movwf   rw_chipsel
    movlw   0xFC
    movwf   rw_add_lo
    movlw   0x03
    movwf   rw_add_hi
    movlw   0x02
```

```
    movwf   rw_data
    call    write_address
    return

stop
    ; now stop the timers
    bcf INTCON,GIE  ; disable global interrupts
    bcf PORTB,PULSEEN   ; stop the timers
    bsf PORTB,PPSCLR    ; This clears the sync flip-flop
    call    wait        ; wait for timers to put out any last pulses
    bcf PORTB,PPSCLR    ; to shut off "GATE0" (And stop the PRT timer)
    return

start   ; this routine starts the timers based on the mode
    ; TIMINGMODE 0 = CONTINUOUS
    ; TIMINGMODE 1 = TRIGGERED
    ; TIMINGMODE 2 = SYNCED

    ; "switch(timermode)"
    movlw   dpram_timermode ; timermode 0
    call    readconfig
    sublw   0x00
    btfss   STATUS,Z
    goto    start1

    ; case timermode 0
    bcf PORTC,SPARE ; set to internal triggers
    bsf PORTB,PULSEEN
    bsf PORTB,PPSCLR    ; generate a start pulse manually on TIMER0
    bcf PORTB,PPSCLR
    bcf INTCON,INTF ; clear external interrupt flag
    bsf INTCON,INTE ; enable external interrupt
    bsf INTCON,GIE  ; now turn it on and let it rip (the global int, that is)
    return

start1  movlw   dpram_timermode ; timermode 1
    call    readconfig
    sublw   0x01
    btfss   STATUS,Z
    goto    start2

    ; case timermode 1
    bsf PORTC,SPARE ; set to external triggers
    bsf PORTB,PULSEEN
    bcf INTCON,INTF ; clear external interrupt flag
    bsf INTCON,INTE ; enable external interrupt
    bsf INTCON,GIE  ; now turn it on and let it rip (the global int, that is)
    return
```

```
start2  movlw   dpram_timermode ; timermode 2
    call    readconfig
    sublw   0x02
    btfss   STATUS,Z
    return          ; command not implemented (this is a little different than the
others)

    ; case timermode 2
    bsf PORTC,SPARE ; set to external triggers (for first one)
    bcf PORTB,PULSEEN   ; (should already be clear)
    bsf PORTB,PPSCLR    ; clear the PPS latch that generates GATE0
    bcf PORTB,PPSCLR
    bsf PORTB,PULSEEN   ; allow pulses to get out
    bcf INTCON,INTF ; clear external interrupt flag
    bsf INTCON,INTE ; enable external interrupt
    bsf INTCON,GIE  ; now turn it on and let it rip (the global int, that is)
    return

initsystem

    ; initialize all the pulse timers
    call    timerinit

    ; initialize int_cnt and int_max_cnt from DPRAM to on-board RAM
    movlw   dpram_seqcount  ;get pointer to sequence count in DRPAM
    call    readconfig  ;get config data from that offset
    movwf   int_cnt     ;save in local copy of int_cnt
    movlw   dpram_seqlength ;get the sequence length pointer for DPRAM
    call    readconfig  ;get config data from that offset
    movwf   int_max_cnt ;save in local copy of max_cnt

    ; initialize the pll reference frequency
    movlw   dpram_pllrefhi
    call    readconfig
    movwf   pll_hi
    movlw   dpram_pllrefmd
    call    readconfig
    movwf   pll_md
    movlw   dpram_pllreflo
    call    readconfig
    movwf   pll_lo
    call    pll

    ; initialize the pll output frequency
    movlw   dpram_pllfreqhi
    call    readconfig
    movwf   pll_hi
    movlw   dpram_pllfreqmd
    call    readconfig
```

```
    movwf   pll_md
    movlw   dpram_pllfreqlo
    call    readconfig
    movwf   pll_lo
    call    pll

    ; program the timers accounting for the correct trigger mode
    ; All modes:    set delay in cnt0,mode5 (one shot)
    ;       set prt in cnt1, mode2 (continuous)
    ;       set seqdelay in cnt2, mode5 (one shot)
    ; note: in timer mode 2, set cnt0 for software triggered one-shot
    ; In trigger mode 1 (triggers), set PORTC,SPARE hi

    ; set the sync timer
    movlw   dpram_synclo    ;get the timer low byte pointer
    call    readconfig  ;get config data from that offset
    movwf   timer_lo    ;put it to timer variable
    movlw   dpram_synchi    ;get the timer high byte pointer
    call    readconfig  ;get config data from that offset
    movwf   timer_hi    ;put it to timer variable
    movlw   0x04        ;the timer chip number
    movwf   timer_chip  ;set the timer chip number
    movlw   0x00        ;the timer chip number
    movwf   timer_num   ;set the timer chip number
    movlw   0x05        ;set the timer mode (one shot)
    movwf   timer_mode  ;put it to timer variable
    call    timer

    ; set the period timer for an arbitrary period so mode gets set
    movlw   0x00        ;get the pointer to the first seq value
    call    readsequence    ;get config data from that offset
    movwf   timer_lo    ;put it to timer variable
    movlw   0x01        ;get the timer high byte pointer
    call    readsequence    ;get config data from that offset
    movwf   timer_hi    ;put it to timer variable
    movlw   0x04        ;the timer chip number
    movwf   timer_chip  ;set the timer chip number
    movlw   0x01        ;the timer chip number
    movwf   timer_num   ;set the timer chip number
    movlw   0x02        ;set the timer mode (continuous)
    movwf   timer_mode  ;put it to timer variable
    call    timer

    ; now set all of the registers and timers for the first PRT
    call    timerisr

    ; set the sequence delay timer
    movlw   dpram_seqdello  ;get the timer low byte pointer
    call    readconfig  ;get config data from that offset
```

```
    movwf   timer_lo    ;put it to timer variable
    movlw   dpram_seqdelhi  ;get the timer high byte pointer
    call    readconfig  ;get config data from that offset
    movwf   timer_hi    ;put it to timer variable
    movlw   0x04        ;the timer chip number
    movwf   timer_chip  ;set the timer chip number
    movlw   0x02        ;the timer number within chip
    movwf   timer_num   ;set the timer number within chip
    movlw   0x05        ;set the timer mode (one shot)
    movwf   timer_mode  ;put it to timer variable
    call    timer

; this may not be necessary here
    ; set the SPARE signal according to the timing mode
    ; TIMINGMODE 0 = CONTINUOUS
    ; TIMINGMODE 1 = TRIGGERED
    ; TIMINGMODE 2 = SYNCED
;   bcf PORTC,SPARE ; set to internal triggers
;   movlw   dpram_timermode
;   call    readconfig
;   btfsc   rw_data,0
;   bsf PORTC,SPARE ; set to external triggers

    return


;*****************************************************************************
main    ; start exectution here

    bcf STATUS,RP1  ;Set to bank 0 or 1 (this should never change)

    ; initialize PORTA
    bcf STATUS,RP0  ;Set to bank 0
    movlw   0XFF        ;Set all bits to one
    movwf   PORTA       ;Put pattern to the port
    bsf STATUS,RP0  ;Select bank 1
    movlw   0X00        ;Set data direction bits (all outputs)
    movwf   TRISA       ;Move to port A control register

    ; initialize PORTB
    bcf STATUS,RP0  ;Set to bank 0
    movlw   0X00        ;Set all bits to zero
    movwf   PORTB       ;Put pattern to the port
    bsf STATUS,RP0  ;Select bank 1
    movlw   0X21        ;Set data direction bits
    movwf   TRISB       ;Move to port B control register

    ; initialize PORTC
    bcf STATUS,RP0  ;Set to bank 0
```

```
    movlw   0X03        ;Set two bits to one
    movwf   PORTC       ;Put pattern to the port
    bsf STATUS,RP0  ;Select bank 1
    movlw   0X80        ;Set data direction bits
    movwf   TRISC       ;Move to port C control register
    ; note PORTC ALATCH is nominally low

    ; initialize PORTD
    bcf STATUS,RP0  ;Set to bank 0
    movlw   0X00        ;Set all bits to zero
    movwf   PORTD       ;Put pattern to the port
    bsf STATUS,RP0  ;Select bank 1
    movlw   0XFF        ;Set data direction bits (all inputs)
    movwf   TRISD       ;Move to port D control register

    ; initialize PORTE
    bcf STATUS,RP0  ;Set to bank 0
    movlw   0X00        ;Set all bits to zero
    movwf   PORTE       ;Put pattern to the port
    bsf STATUS,RP0  ;Select bank 1
    movlw   0X00        ;Set data direction bits (all outputs)
    movwf   TRISE       ;Move to port E control register
    movlw   0x07
    movwf   ADCON1      ;This must be set to allow port E to be digital I/O

    bcf STATUS,RP0  ;Select bank 0 (this is assumed in many routines)

        ; initialize the SPI
    movlw   0x22        ;this is the slowest rate (0x20 is highest)
    movwf   SSPCON
    bsf STATUS,RP0  ;Select bank 1
    movlw   0x40
    movwf   SSPSTAT
    bcf STATUS,RP0  ;Select bank 0 (this is assumed in many routines)

    ; initialize and reset the pll
    bcf PORTB,PLL_LE    ; do this just to be sure
    movlw   0x10
    movwf   pll_hi
    movlw   0xFA
    movwf   pll_md
    movlw   0xD3    ; Note: was 93 for digital lock detector
    movwf   pll_lo
    call    pll

    ; fill up the configuration and sequence ram with some intial values
    call initdpram

    call initsystem     ; read in the DPRAM and set lots of things up
```

```
    ; now that timers are programmed, wait for them to stop
    call stop

    ; now start puting out pulses for testing purposes
    ; this can only be called before interrupts are enabled!
    call    clear_request   ; clear the request flag (local and DPRAM)
        call    start

    ; keep reading in the request flag located at 0x3F
    ; When it is non-zero, then call "service"
    ; Clearing of the request flag is done elsewhere

        ; blink the LED's

blink   movlw   0x04
    xorwf   PORTE,f

;  goto mainskip1       ; skip the host communication stuff

        ; this used to be in the interrupt loop
        movlw   0x3F        ; read the PCI request flag from DPRAM and store it in local
memory
    bcf INTCON,INTE ; disable external interrupt
    call    readconfig
    bsf INTCON,INTE ; enable external interrupt

    andlw   0x01
    btfsc   STATUS,Z    ; a request causes timer and interrupts to halt
    goto    mainskip1

    call    service

mainskip1

    clrwdt          ; clear the watchdog timer

    call    wait

    goto    blink

    END                     ; directive 'end of program'
```