```
;
; PicoBlaze program to control a PSD chip on new chip board
; George L. Engel
;
; 66 MHz version!!!!!
;
; All constant values are in HEX!
;
; Define the various bit masks
;
            CONSTANT     b0msk, 01
            CONSTANT     b1msk, 02
            CONSTANT     b2msk, 04
            CONSTANT     b3msk, 08
            CONSTANT     b4msk, 10
            CONSTANT     b5msk, 20
            CONSTANT     b6msk, 40
            CONSTANT     b7msk, 80
            CONSTANT     bits_2and3, 0C

            CONSTANT     b0mskN, FE
            CONSTANT     b1mskN, FD
            CONSTANT     b2mskN, FB
            CONSTANT     b3mskN, F7
            CONSTANT     b4mskN, EF
            CONSTANT     b5mskN, DF
            CONSTANT     b6mskN, BF
            CONSTANT     b7mskN, 7F

;
; Picoblaze has 16 general purpose registers
;
; Registers used in data_acq subroutine
;
            NAMEREG      s0, op0         ; Output port #0
            NAMEREG      s1, op1         ; Output port #1
            NAMEREG      s2, op4         ; Output port #4
            NAMEREG      s3, op5         ; Output port #5
            NAMEREG      s4, ip0         ; Input port #0
            NAMEREG      s5, ip1         ; Input port #1

;
; Give more desciptive names to some of our registers
;
            NAMEREG      s6, scr0        ; Scratchpad register
            NAMEREG      s7, scr1
            NAMEREG      s8, scr2
            NAMEREG      s9, cnt         ; Used in for loops
            NAMEREG      sA, isr_reg     ; Used in ISR
            NAMEREG      sB, dreg        ; Used by delays
            NAMEREG      sC, adc_bits    ; Stores sdi_abc and sdi_t

;*****************************************************************
;
; MAIN ROUTINE
;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
            ADDRESS      000

start:      DISABLE      INTERRUPT
            CALL         init       ; Inits output ports (uses scr1)
;
; Now that we have initialized our ports, oK to turn on intterrupts
;
            ENABLE       INTERRUPT
;
; Sit in a tight loop polling the veto_rst line.
; Waiting for veto_rst to be HIGH
; veto_rst is bit 0 of input port 0
;
wait:       INPUT        ip0, 00
            TEST         ip0, b0msk
            JUMP         Z, wait
;
; Call the data acquisition routine
;
            CALL         data_acq

; Wait for veto reset to go LOW (bit 0, input port 0)
;
wait1:      INPUT        ip0, 00
            TEST         ip0, b0msk
            JUMP         NZ, wait1

            OUTPUT       scr0, 07      ; Reset transfer FSM and RD/WR ptrs

            JUMP         wait
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Name: data_acq
;
; This routine will acquire data from the 4 ADCs (A, B, C, T)
; Send 00 followed by 14 0s for (+)
; Send 01 following by 14 0s for (-)
;
; Apply 16 ADC clocks ... first two are special
;
;
data_acq:    OUTPUT     scr0, 07      ; Reset transfer FSM and RD/WR ptrs
             OR         op4, b3msk    ; Set the busy bit
             OUTPUT     op4, 04
;
; Check the OR out of the PSD chip (bit 7 of input port #1)
; If it is LOW then we are done
;
             INPUT      ip1, 01       ; Check the OR from PSD chip
             TEST       ip1, b7msk
             JUMP       Z, acq_done   ; If OR is LOW then we are done!

             AND        op0, b2mskN   ; Bring token_in_PSD LOW.
```

```
              OUTPUT     op0, 00

acq0:         OR         op0, b6msk    ; Bring acq_clk high
              OUTPUT     op0, 00
              CALL       sample        ; Sample A, B, C, T voltages
              CALL       convert       ; Apply a 3.2 usec convert signal
              INPUT      ip1, 01       ; Read PSD addresses
              AND        op0, b6mskN   ; Bring acq_clk_PSD back low
              OUTPUT     op0, 00
              INPUT      ip0, 00       ; Check token_out_PSD
              INPUT      ip0, 00       ; Check token_out_PSD
              TEST       ip0, b6msk
              JUMP       Z, acq2       ; If token_out is LOW then done

acq1:         OR         op0, b6msk    ; Bring acq_clk high
              OUTPUT     op0, 00
              CALL       sample        ; Sample A, B, C, T voltages
              OUTPUT     scr0, 06      ; Tell FSM to xfer data
              CALL       convert       ; Apply a 3.2 usec convert signal
              INPUT      ip1, 01       ; Read PSD addresses
              AND        op0, b6mskN   ; Bring acq_clk_PSD back low
              OUTPUT     op0, 00
              INPUT      ip0, 00       ; Check token_out_PSD
              INPUT      ip0, 00       ; Check token_out_PSD
              TEST       ip0, b6msk
              JUMP       NZ, acq1      ; If token_out is HIGH then continue

acq2:         CALL       sample        ; Read out last set of ADC values
              OUTPUT     scr0, 06      ; Tell FSM to transfer data

acq_done:     OR         op0,b2msk    ; Bring token_in_PSD high
              OUTPUT     op0, 00

              AND        op4, b3mskN  ; Unset the busy flag
              OUTPUT     op4, 04

              RETURN

; ***********************************************************
; Routine to sample the A, B, C, T signals
; Send 16 ADC serial clocks
; ***********************************************************

sample:       LOAD       op5, b3msk     ; Bring ADC clk HIGH
              OUTPUT     op5, 05        ; Write to output port 5
              OR         op5, adc_bits  ; Config sdi_abc and sdi_t bits
              AND        op5, b3mskN    ; Bring ADC clk LOW
              OUTPUT     op5, 05        ; Write to ouput port 5
; clk pulse 2
              OR         op5, b3msk   ; Bring ADC clk HIGH
              OUTPUT     op5, 05      ; Write to output port 5
              AND        op5, b3mskN  ; Bring ADC clk LOW
              OUTPUT     op5, 05      ; Write to ouput port 5
; clk pulse 3
              OR         op5, b3msk   ; Bring ADC clk HIGH
              OUTPUT     op5, 05      ; Write to output port 5
```

```
             AND        op5, b3mskN  ; Bring ADC clk LOW
             OUTPUT     op5, 05      ; Write to ouput port 5
; clk pulse 4
             OR         op5, b3msk   ; Bring ADC clk HIGH
             OUTPUT     op5, 05      ; Write to output port 5
             AND        op5, b3mskN  ; Bring ADC clk LOW
             OUTPUT     op5, 05      ; Write to ouput port 5
; clk pulse 5
             OR         op5, b3msk   ; Bring ADC clk HIGH
             OUTPUT     op5, 05      ; Write to output port 5
             AND        op5, b3mskN  ; Bring ADC clk LOW
             OUTPUT     op5, 05      ; Write to ouput port 5
; clk pulse 6
             OR         op5, b3msk   ; Bring ADC clk HIGH
             OUTPUT     op5, 05      ; Write to output port 5
             AND        op5, b3mskN  ; Bring ADC clk LOW
             OUTPUT     op5, 05      ; Write to ouput port 5
; clk pulse 7
             OR         op5, b3msk   ; Bring ADC clk HIGH
             OUTPUT     op5, 05      ; Write to output port 5
             AND        op5, b3mskN  ; Bring ADC clk LOW
             OUTPUT     op5, 05      ; Write to ouput port 5
; clk pulse 8
             OR         op5, b3msk   ; Bring ADC clk HIGH
             OUTPUT     op5, 05      ; Write to output port 5
             AND        op5, b3mskN  ; Bring ADC clk LOW
             OUTPUT     op5, 05      ; Write to ouput port 5
; clk pulse 9
             OR         op5, b3msk   ; Bring ADC clk HIGH
             OUTPUT     op5, 05      ; Write to output port 5
             AND        op5, b3mskN  ; Bring ADC clk LOW
             OUTPUT     op5, 05      ; Write to ouput port 5
; clk pulse 10
             OR         op5, b3msk   ; Bring ADC clk HIGH
             OUTPUT     op5, 05      ; Write to output port 5
             AND        op5, b3mskN  ; Bring ADC clk LOW
             OUTPUT     op5, 05      ; Write to ouput port 5
; clk pulse 11
             OR         op5, b3msk   ; Bring ADC clk HIGH
             OUTPUT     op5, 05      ; Write to output port 5
             AND        op5, b3mskN  ; Bring ADC clk LOW
             OUTPUT     op5, 05      ; Write to ouput port 5
; clk pulse 12
             OR         op5, b3msk   ; Bring ADC clk HIGH
             OUTPUT     op5, 05      ; Write to output port 5
             AND        op5, b3mskN  ; Bring ADC clk LOW
             OUTPUT     op5, 05      ; Write to ouput port 5
; clk pulse 13
             OR         op5, b3msk   ; Bring ADC clk HIGH
             OUTPUT     op5, 05      ; Write to output port 5
             AND        op5, b3mskN  ; Bring ADC clk LOW
             OUTPUT     op5, 05      ; Write to ouput port 5
; clk pulse 14
             OR         op5, b3msk   ; Bring ADC clk HIGH
             OUTPUT     op5, 05      ; Write to output port 5
             AND        op5, b3mskN  ; Bring ADC clk LOW
```

```
             OUTPUT     op5, 05      ; Write to ouput port 5
; clk pulse 15
             OR         op5, b3msk   ; Bring ADC clk HIGH
             OUTPUT     op5, 05      ; Write to output port 5
             AND        op5, b3mskN  ; Bring ADC clk LOW
             OUTPUT     op5, 05      ; Write to ouput port 5
; clk pulse 16
             LOAD       op5, b3msk   ; Bring ADC clk HIGH
             OUTPUT     op5, 05      ; Write to output port 5
             LOAD       op5, 00      ; Bring ADC clk LOW
             OUTPUT     op5, 05      ; Write to ouput port 5

             RETURN

; ************************************************************
; Routine to send 3.2 us conversion signal to ADCs
; ************************************************************

convert:     OR         op5, b2msk    ; Bring ADC conv high
             OUTPUT     op5, 05
             CALL       dly_3200ns    ; Uses dreg
             AND        op5, b2mskN   ; Bring ADC conv low
             OUTPUT     op5, 05
             RETURN

;
; ************************************************************
; Routine to define values on the various bits in the 6 output ports
; Make copies of what we write out in locations $20 thru $25
; Uses scr1 register and does not restore it
; ************************************************************************

init:       LOAD        op0, 00
;
; Output port #0 bits all LOW except ...
;
            OR          op0, b2msk     ; HIGH on token_in_PSD
            OUTPUT      op0, 00
;
; Make all bits in output port #1 LOW
;
            LOAD        op1, 00
            OUTPUT      op1, 01
;
; Make all bits in output port #2 LOW
; Port #2 is lower byte of DB bus
;
            LOAD        scr0, 00
            OUTPUT      scr0, 02
            STORE       scr0, 22
;
; Make all bits in output port #3 LOW
; Port #3 is upper byte of DB bus
;
            OUTPUT      scr0, 03
            STORE       scr0, 23
```

```
;
; Make all bits in output port #4 LOW except bit 7 which is the selFIFO line
; The FIFO output should drive Jon's DB bus.
; Shadow register stuff and the BIDIR control signals
;
            LOAD        op4, 00
            OR          op4, b7msk    ; HIGH on the selFIFO line
            OUTPUT      op4, 04
;
; Port #5 contains the ADC signals
; All lines should be LOW
;
            LOAD        op5, 00
            OUTPUT      op5, 05
;
; A write to port #7 of ANY value will cause RD and WR pointers
; along with FSMs in FIFO to be reset
;
            OUTPUT      scr0, 07
;
; Load the adc_bits register with appropriate values
;
            LOAD        adc_bits, 00
            INPUT       scr0, 09           ; Get CR32 - CR39
            AND         scr0, b7msk        ; Inspect polarity bit

;            JUMP        Z, init0           ; Original

            JUMP        NZ, init0           ; 9 sep
            OR          adc_bits, b0msk    ; sdi_abc
init0:      OR          adc_bits, b1msk    ; sdi_t

            RETURN
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Name:  config_PSD
; Uses: scr1, scr2, cnt, dreg
;
; Routine used to configure the PSD chip.
; Need to shift in 48 bits of configuration information.
; Bit 47 goes in first and bit 0 goes in last
;
; We will use a sclk of 500 kHz (1 us high, 1 us low)
; Load a byte at a time and send it ...
;
; Assign all chips an ID of 00
; So first byte we send should always by $00
;
; Routine uses the scr1 register and does not restore it.
;

config_PSD:    LOAD         scr1, 00       ; Chip is always $00
               CALL         send_byte

               INPUT        scr1, 09       ; Input port #9
```

```
                CALL        send_byte

                INPUT       scr1, 08       ; Input port #8
                CALL        send_byte

                INPUT       scr1, 07       ; Input port #7
                CALL        send_byte

                INPUT       scr1, 06       ; Input port #6
                CALL        send_byte

                INPUT       scr1, 05       ; Input port #5
                CALL        send_byte

                RETURN
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Name: send_byte (passed into routine in scr1)
;
; Routine to send (bitwise serially) 1 byte of information.
; Byte to transmitted is passed into routine in scr1 register
; Use scr2 as our bit mask
; Use cnt to keep track of how many bits we have sent
; scr0 gets used by the delay_1us routine
;
; Bit 3 of output port 0 is sin_PSD
; Bit 4 of output port 0 is sclk_PSD
;
send_byte:      LOAD        cnt, 08            ; Number of bits to be shifted
                LOAD        scr2, b7msk        ; scr2 is our bit mask

send_byte0:     OR          op0, b3msk         ; Make sin high
                TEST        scr1, scr2         ; Should be high?
                JUMP        NZ, send_byte1     ; If suppose to be high skip
next instr
                XOR         op0, b3msk         ; else clear sin bit
send_byte1:     OUTPUT      op0, 00            ; Output sin to port
                XOR         op0, b4msk         ; Bring sclk low
                OUTPUT      op0, 00            ; Write to port #0
                CALL        delay_1us          ; Wait 1 us
                XOR         op0, b4msk         ; Bring sclk back high
                OUTPUT      op0, 00            ; Write to port #0
                CALL        delay_1us          ; Wait  1 us
                SR0         scr2               ; Shift mask to point to next
bit
                SUB         cnt, 01            ; Update bit counter
                JUMP        NZ, send_byte0

                RETURN
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Name: delay_1us
;
```

```
; Delay of 1 usec
; Delay is [(4 * dreg) + 6] / Fclk
; Routine uses dreg and does not restore it
;
; 14 assumes 84 MHz clk
; Modified Jan 12, 2011
;
delay_1us:  LOAD        dreg, 14
wait_1us:   SUB         dreg, 01
            JUMP        NZ, wait_1us
            RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Name: dly_3200ns
;
; Delay is [(4 * dreg) + 6] / Fclk
;
; 42 assumes a 84 MHz clock
; Modified Jan 12, 2011
;
dly_3200ns:  LOAD        dreg, 42
dly0:        SUB         dreg, 01
             JUMP        NZ, dly0
             RETURN

;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; ISR
;
; An interrupt occurs when "sclk" transitions high
; Configuration data is about to be sent.
; Job of ISR is just to wait for all of the config data
; to be read.
;
; We will know that all of the configuration data has
; been sent when there is a pulse on the dac_sgn line.
;
; All we need to do is sit and wait for dac_sgn pulse
; dac_sgn is bit 4 of port #4
;
; Uses scr1 and does not restore register
;
            ADDRESS     300

isr:        INPUT       isr_reg, 04
            AND         isr_reg, bits_2and3     ; sc1 and sc0
            SUB         isr_reg, bits_2and3
            JUMP        NZ, isr                 ; wait for rising edge
;
; Now wait for trailing  edge of dac_sgn pulse
;

isr1:       INPUT       isr_reg, 04
            AND         isr_reg, bits_2and3
```

```
            SUB         isr_reg, bits_2and3
            JUMP        Z, isr1
;
; Configure the PSD chip
; Lets do it twice so we can watch the data on the PSD sout lines
;
           CALL        config_PSD
;          CALL        config_PSD     ; commented out on 9 sep 2010
;
; Assert the clr_int control to purge the pending interrupt
; else we will load the PSD config register a second time
;
           OR          op0, b0msk
           OUTPUT      op0, 00
           AND         op0, b0mskN
           OUTPUT      op0, 00
;
; Reinit everything
;
           CALL        init
;
           RETURNI     ENABLE
;
; Interrupt vector is stored at location $3FF
; We will jump to our ISR routine
;
            ADDRESS    3FF
            JUMP       isr
```