60HZDemo Augmented Using The HI08 Interface

Ellery Blood
September 1999

Index

Introduction:

This demonstration is a modification to the 60HzDemo that comes on the "Assembler/Linker/Demo/Test files for the DSP56303EVM V2.2" disk. The original demo adds a 60Hz hum to an external audio signal, then uses different filters to remove the hum. The demo uses IRQA, IRQB, and IRQD to switch between no filtering, 16 bit filtering, and 24 bit filtering. The no filtering filter simply allows the corrupted signal to pass. The 16-bit filter creates a notch filter at approximately 60Hz to remove the hum. The 24-bit filter creates a better notch filter at exactly 60Hz to remove the hum.

The augmentation to this demo uses the HI08 interface on the DSP56303 to allow the interrupts to be triggered from a different source. In addition to general communication, the HI08 also has a Host Command feature, which allows a host processor connected through the HI08 to issue interrupt requests to any of the 128 interrupt vectors on the DSP. In this version of the demo, the MC68HC11 is used through the interface to trigger the IRQA, B, and D interrupts.

The mechanics of triggering interrupts through the HI08 consist of writing the interrupt vector number to the CVR (Command Vector Register) and setting the Host Command bit (CVR bit 7). The interrupt vectors on the DSP56303 are mapped to the first 256 words of program memory, with each vector occupying 2 words. The vector number is the vector location divided by 2. To trigger a particular interrupt through the HI08, the 7 bit vector number is written to the bottom 7 bits of CVR (bits 0-6), and the Host Command bit is set. When the interrupt is serviced by the interrupt handler, the HI08 hardware resets the Host Command bit (CVR bit 7) to show that the interrupt has been serviced.

Example: activate IRQA through the HI08.

IRQA's vector is located at P:$10 and therefore has a vector number of $08 ($10/2). This added to $80 (bit 7) gives $88. Write $88 to CVR and the interrupt process is started. When the interrupt is serviced, the HI08 hardware will reset the Host Command bit (CVR bit 7), so CVR will read $08.

Setup:

Initially, the HI08 should be connected physically as shown in the schematics in Appendix A. The configuration for the HI08 is included in the demo program (60HzHI08.asm). The following lines were added to the original 60HzDemo to initialize the HI08 and interrupts:

```
; --- Init the HI08 ---
movep #$002e0e,x:M_HPCR
movep #$000060,x:M_HBAR
movep #$000004,x:M_HCR
movep #$002e4e,x:M_HPCR
bset  #$01,x:$fffffe
bset  #$02,x:$fffffe
```

Line by line, the instructions set the following parameters.

```
movep #$002e0e,x:M_HPCR
```

Sets the HPCR (Host Port Control Register) to $2e0e. This begins the initialization of the host port and sets the control bits for enabling address bits 8-10, sets signal polarities, and sets the interface for a multiplexed bus. It does not actually enable the host interface at this time however.

```
movep #$000060,x:M_HBAR
```

Sets the HBAR (Host Base Address Register) to $60. This tells the interface which address space to map itself to on the host bus. $60 corresponds to $6000-$7FFF in the HC11 memory address space. The direct mapping can be seen in Appendix B.

```
movep #$000004,x:M_HCR
```

Sets the HCR (Host Control Register) to $04. This sets the HCIE (Host Command Interrupt Enable) bit which enables the interrupts triggering capabilities of the HI08.

```
movep #$002e4e,x:M_HPCR
```

Sets the HPCR to $2e4e. This is the same as the previous HPCR instruction, with the exception that the HEN (Host ENable) bit is now set.

```
bset  #$01,x:$fffffe
bset  #$02,x:$fffffe
```

These two lines set bits 0 and 1 in the IPR-P (Interrupt Priority Register - Pheriperal). This sets the interrupt priority for the Host Command Interrupts to level 3 (non-maskable). Interrupts can be assigned levels from 0 (lowest) to 3 (highest). Setting the level to 3 assures the Host Command Interrupts will not be masked by lower priority interrupts.

Walkthrough:

Setup:

1. Begin by physically connecting the HC11 and DSP56303 through the HI08 as shown in the schematics in Appendix A.

2. Connect the DSP56303 to COM1 and the HC11 to COM2 using RS-232 serial cables.

3. Connect one stereo audio jack from a sound source to the Line In port on the DSp56303 EVM. Connect headphones to the Headphone jack on the DSP56303 EVM.

4. Power on the HC11 and DSP56303.

5. Open a terminal connection to COM2.

6. Open the debugging software for the DSP (EVM30xw.exe) that comes with the EVM. If the **Command** window is not already open, open it by selecting **CMD** from the View menu. Press the STOP button, then select **Load** from the File menu. Select the file 60HzHI08.cld to load it into the DSP56303's memory.

7. Start the demo by typing **`GO start`** in the command menu. The demo is now running. Start the audio source, and you should hear that through the headphones with a 60 Hz hum added to it.

Filtering:

1. You can activate the 24 bit filter by pressing the IRQA switch 16 bit filter by pressing the IRQD switch. You should notice a distinct difference between the two filtered effects.

2. To activate the filters through the HI08 go to the Terminal window to COM2.

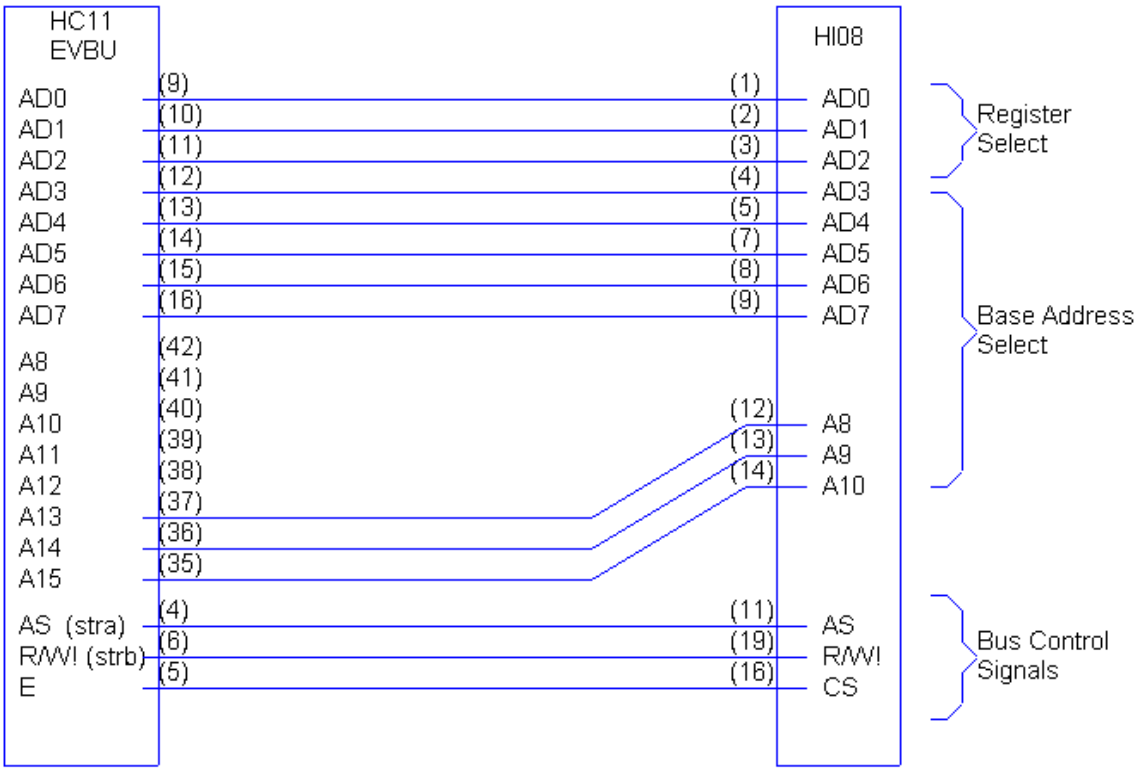
3. Type in **MM 6001 [enter]** then type **88 [enter]** to activate the 24 bit filter.

4. Type in **MM 6001 [enter]** then type **8B [enter]** to activate the 16 bit filter.

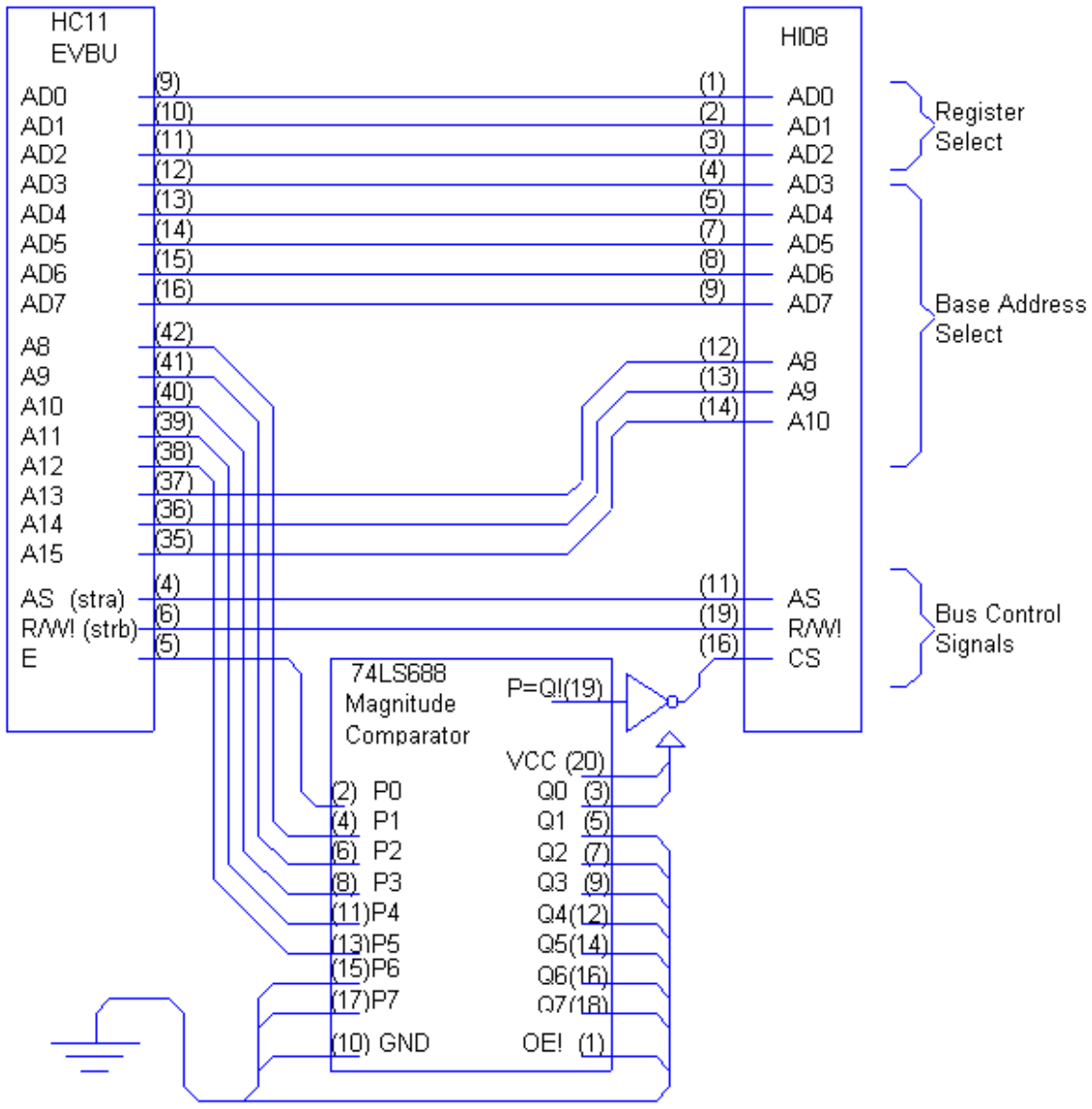
5. Type in **MM 6001 [enter]** then type **89 [enter]** to return to the unfiltered state.

Appendix A : Wiring Schematics

Glueless interface



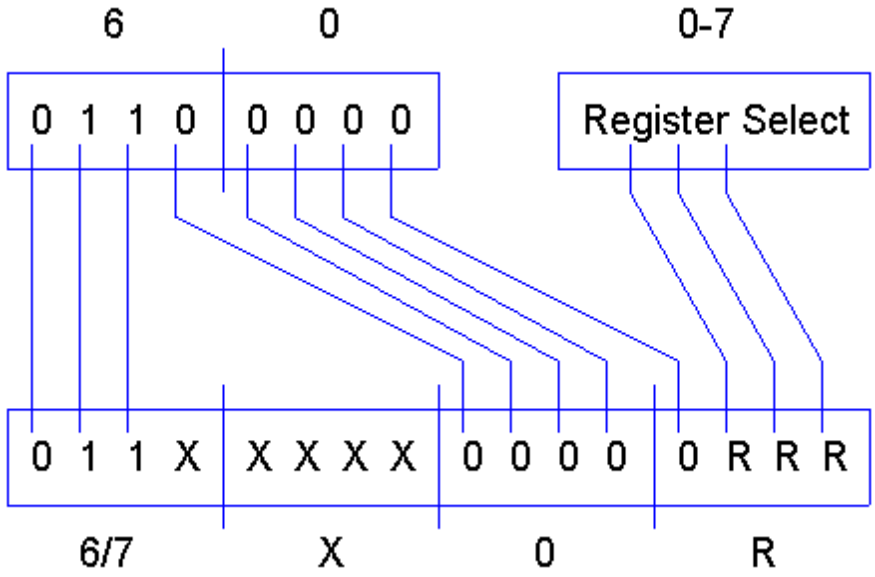
Full Address Decoding interface



Appendix B : Memory Mapping

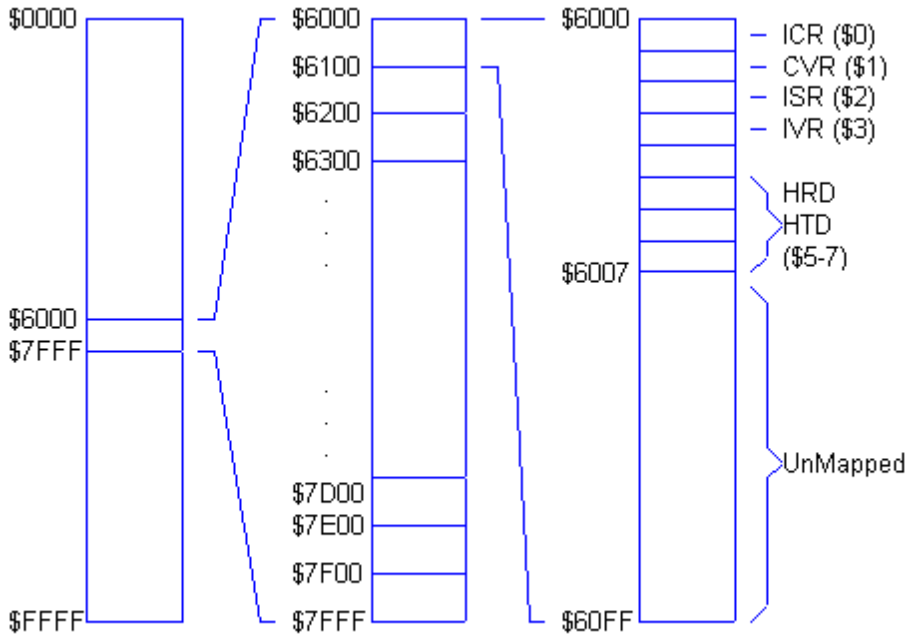
The HI08 maps itself onto the memory bus of other processors by defining a set address that it interfaces to. In Multiplexed mode, the 8 lower data/address lines are multiplexed to carry both the lower 8 bits of the address, and 8 bits of data. The HI08 also uses 3 additional lines for address decoding. This makes 11 lines in total. Using the Address Strobe, the HI08 captures the address bits. It compares the top 8 address bits to the value stored in the HBAR (Host Base Address Register) to determine if the bus is addressing the HI08 or not. The 3 lowest bits are used to address one of the eight registers mapped to the Host address space. (See figure 1 for a visual representation)

In the glueless configuration with the HC11, five of the address lines of the HC11 are left unconnected (denoted as Xs in figure 1). The ambiguity that the unconnected address lines create causes the eight registers to take up 2KB of memory. In the configuration shown below, the HBAR is set to $60, fixing the HI08 to the memory range $6000-$7FFF.



(figure 1)

More specifically, the whole 2KB of memory is not completely taken up with the HI08 because of the eight lower address lines as shown in figure 2. These lines map the 8 registers to the first eight bytes of a 256-byte block (the far right block on figure 2). This block is repeated 32 times throughout the 2KB block.



(figure 2)

The tradeoff for the simplicity of the glueless interface is the large amount of memory it occupies. When external logic is added, the memory map can be reduced to only the eight memory locations minimally required for the eight registers.

Appendix C : Resources

DSP56303 Users Manual (DSP56303UM/AD), Motorola inc. 1996

DSP56300 24-bit Digital Signal Processor Family Manual, Motorola inc.

H68HC11EVBU Universal Evaluation Board User's Manual (M68HC11EVBU/AD2)

DSP56303EVM - M68HC11EVBU HI08 Interface, Ellery Blood, September 1999

Appendix D : Code Listing

```
         page    132,60
;****************************************************************************
;       60HZHI08.ASM
;       Example program to demonstrate the difference between 24-bit data
;       capabilities and 16-bit data capabilities
;
;       Modified to work with the HI08 to trigger the different filtering modes
;       Ellery Blood September 1999
;
;       Copyright (c) MOTOROLA 1994
;                     Semiconductor Products Sector
;                     Digital Signal Processing Division
;
;       ver. 1.1 3/16/95 Placed coefficients on modulo boundaries.  Moved the
;                        "state" variable locations. Set sine amplitude to 0.3.
;                6/20/96 PFS Modified for 56303
;                9/16/99 Modified for HI08 interface
;
;****************************************************************************
; This code demonstrates the benefits of a 24-bit architecture over that of
; a 16-bit architecture.  The demonstration runs on the DSP56303EVM evaluation
; module.  An external audio signal is input through the microphone connector
; on the EVM.  A 60 Hz tone is generated by the DSP (via a digital oscillator)
; and then added to the digitized audio signal.       The resulting data is then
; sent through a filter with one of three sets of coefficients.      The first
; set of coefficients is located at location y:no_filter and performs no
; filtering at all, simply allowing the corrupted signal to pass.  This first
; set of coefficients are used by the filter when the demo first begins or
; following the IRQB (IRQB).  The second set of coefficients
; (located at y:coef_24) are 24-bit coefficients that make up a 60 Hz notch
; filter which removes the 60 Hz portion of the corrupted signal.  This second
; set of coefficients are used following the external interrupt A (IRQA).  The
; final set of coefficients (located at y:coeff_16) are the same coefficients
; as those for the 24-bit filter, only rounded to 16 bits.  These 16-bit
; coefficients are used by the filter following the external interrupt D
; (IRQD).
;****************************************************************************
         nolist
         include 'ioequ.asm'
         include 'intequ.asm'
         include 'ada_equ.asm'
         list
         include 'vectors.asm'

Fs      set     48000.0                         ;Specify sampling frequency.
PI      set     2.0*@asn(1.0)                   ;Compute PI as 2.0*arcsin(1.0)
factor  set     PI/180.0                        ;Multiplier for degrees to radians
eighteen set    18
hex_twenty  set         $20

;****************************************************************************
; Specification for tone a.
freq_a  set     60.0                        ;Specify frequency in Hertz.
phi_a   set     360.0*(freq_a/Fs)               ;Compute phi
phase_a set     0.0                             ;Specify the phase angle in
                                                ; degrees (-180 -- +180).
amp_a   set     0.3                             ;Specify amplitude (0-1).
theta2_a        set     (phase_a-(2.0*phi_a))   ;Compute theta2
theta1_a        set     (phase_a-phi_a)                 ;Compute theta1
s2_a    set     amp_a*@sin(factor*theta2_a)     ;Compute s2
s1_a    set     amp_a*@sin(factor*theta1_a)     ;Compute s1
coeff_a set     @cos(factor*phi_a)              ;Compute rcoef in 2:14 format


;****************************************************************************
; These three interrupts (IRQA, IRQD, and IRQB) load the program memory location
; of the appropriate filter routine (24-bit, 16-bit, or NMI).
```

```
        org     p:$0
        jmp     START
         org     p:$10                  ; IRQA--Filter in 24 bit mode.
         move    #coef_24+1,r4          ; Load code pointer. (command 08)
         org     p:$12                  ; IRQB--no filtering
         move    #no_filter+1,r4        ; Load code pointer. (command 09)
         org     p:$14                  ; IRQC--Filter in 16 bit mode.
         move    #coef_16+1,r4          ; Load code pointer. (command 0A)
         org     p:$16                  ; IRQD--Filter in 16 bit mode.
         move    #coef_16+1,r4          ; Load code pointer. (command 0B)


;*****************************************
; Equates for the HI08 interface
;
;M_HTX    EQU     $FFFFC7        ; Host Transmit Register
;M_HRX    EQU     $FFFFC6        ; Host Receive Register
;M_HBAR   EQU     $FFFFC5        ; Host Base Address Register
;M_HPCR   EQU     $FFFFC4        ; Host Port Control Register
;M_HSR    EQU     $FFFFC3        ; Host Status Register
;M_HCR    EQU     $FFFFC2        ; Host Control Register


;       org p:$30
;        jsr       ssi_rx_isr                ;- ESSI0 Receive Data
;        jsr       ssi_rxe_isr               ;- ESSI0 Receive Data w/ Exception Status
;        jsr       ssi_rxls_isr                      ;- ESSI0 Receive last slot
;        jsr       ssi_tx_isr                ;- ESSI0 Transmit Data
;        jsr       ssi_txe_isr               ;- ESSI0 Transmit Data w/ Exception Status
;        jsr       ssi_txls_isr                      ;- ESSI0 Transmit last slot


nsec   equ     3             ;number of second order sections
scount equ     1             ;final shift count


;*************************************************************************
;---Buffer for talking to the CS4215

       org    x:0
RX_BUFF_BASE   equ     *
RX_data_1_2    ds      1       ;data time slot 1/2 for RX ISR
RX_data_3_4    ds      1       ;data time slot 3/4 for RX ISR
RX_data_5_6    ds      1       ;data time slot 5/6 for RX ISR
RX_data_7_8    ds      1       ;data time slot 7/8 for RX ISR


TX_BUFF_BASE   equ     *
TX_data_1_2    ds      1       ;data time slot 1/2 for TX ISR
TX_data_3_4    ds      1       ;data time slot 3/4 for TX ISR
TX_data_5_6    ds      1       ;data time slot 5/6 for TX ISR
TX_data_7_8    ds      1       ;data time slot 7/8 for TX ISR

RX_PTR         ds      1       ; Pointer for rx buffer
TX_PTR         ds      1       ; Pointer for tx buffer



       org     x:
DOSC_BUFF_BASE EQU     *
coeff          ds      1       ; data location for osc. a's coeff.
s1             ds      1       ; data location for osc. a's sr1.
s2             ds      1       ; data location for osc. a's sr2.
LEFT_HUM       ds      1       ;storage for Left Signal + Hum
RIGHT_HUM      ds      1       ;storage for Left Signal + Hum
       org     x:$10
state1         dsm     nsec
       org     x:$1c
state1_r       dsm     nsec
       org     x:$20
state2         dsm     nsec
```

```
        org     x:$2c
state2_r        dsm     nsec

;************************************************************************
        org     y:
coef_24
        dc      $3FFD61         ;b(*,0)/2   =0.49992001     section number  1
        dc      $800641         ;b(*,1)/2   =-.99980915     section number  1
        dc      $7FF9C0         ;a(*,1)/2   =0.99980927     section number  1
        dc      $3FFD61         ;b(*,2)/2   =0.49992001     section number  1
        dc      $C0053E         ;a(*,2)/2   =-.49984002     section number  1
        dc      $3E3C48         ;b(*,0)/2   =0.48621464     section number  2
        dc      $83886C         ;b(*,1)/2   =-.97239923     section number  2
        dc      $7FFDB8         ;a(*,1)/2   =0.99993038     section number  2
        dc      $3E3C48         ;b(*,2)/2   =0.48621464     section number  2
        dc      $C0013F         ;a(*,2)/2   =-.49996197     section number  2
        dc      $20E7A1         ;b(*,0)/ 4  =0.25706875     section number  3
        dc      $BE3142         ;b(*,1)/ 4  =-.51412177     section number  3
        dc      $7FFDCE         ;a(*,1)/ 2  =0.99993300     section number  3
        dc      $20E7A1         ;b(*,2)/ 4  =0.25706875     section number  3
        dc      $C00136         ;a(*,2)/ 2  =-.49996305     section number  3
        dc      0

coef_16
        dc      $3FFD00         ;b(*,0)/2   =0.49991        section number  1
        dc      $800600         ;b(*,1)/2   =-.99982        section number  1
        dc      $7FFA00         ;a(*,1)/2   =0.99982        section number  1
        dc      $3FFD00         ;b(*,2)/2   =0.49991        section number  1
        dc      $C00500         ;a(*,2)/2   =-.49985        section number  1
        dc      $3E3C00         ;b(*,0)/2   =0.48621        section number  2
        dc      $838800         ;b(*,1)/2   =-.97241        section number  2
        dc      $7FFE00         ;a(*,1)/2   =0.99994        section number  2
        dc      $3E3C00         ;b(*,2)/2   =0.48621        section number  2
        dc      $C00100         ;a(*,2)/2   =-.49997        section number  2
        dc      $20E800         ;b(*,0)/ 4  =0.25708        section number  3
        dc      $BE3100         ;b(*,1)/ 4  =-.51413        section number  3
        dc      $7FFE00         ;a(*,1)/ 2  =0.99994        section number  3
        dc      $20E800         ;b(*,2)/ 4  =0.25708        section number  3
        dc      $C00100         ;a(*,2)/ 2  =-.49997        section number  3
        dc      0

no_filter
        dc      $400000         ;b(*,0)/2   =0.50000000     section number  1
        dc      $000000         ;b(*,1)/2   =0.00000000     section number  1
        dc      $000000         ;a(*,1)/2   =0.00000000     section number  1
        dc      $000000         ;b(*,2)/2   =0.00000000     section number  1
        dc      $000000         ;a(*,2)/2   =0.00000000     section number  1
        dc      $400000         ;b(*,0)/2   =0.50000000     section number  2
        dc      $000000         ;b(*,1)/2   =0.00000000     section number  2
        dc      $000000         ;a(*,1)/2   =0.00000000     section number  2
        dc      $000000         ;b(*,2)/2   =0.00000000     section number  2
        dc      $000000         ;a(*,2)/2   =0.00000000     section number  2
        dc      $200000         ;b(*,0)/ 4  =0.25000000     section number  3
        dc      $000000         ;b(*,1)/ 4  =0.00000000     section number  3
        dc      $000000         ;a(*,1)/ 2  =0.00000000     section number  3
        dc      $000000         ;b(*,2)/ 4  =0.00000000     section number  3
        dc      $000000         ;a(*,2)/ 2  =0.00000000     section number  3

left_a          ds      1
left_y0         ds      1
right_a         ds      1
right_y0        ds      1


;************************************************************************
CTRL_WD_12      equ     MIN_LEFT_ATTN+MIN_RIGHT_ATTN+LIN2+RIN2
CTRL_WD_34      equ     MIN_LEFT_GAIN+MIN_RIGHT_GAIN


        org     p:$100
```

```
START
main
; Initialize SC0, SC1 as GPIO inputs
    movep   #$040003,x:M_PCTL        ; PLL = 4 x 16.9344Mhz = 67.7Mhz
    movep   #$012421,x:M_BCR         ; one wait state for external spaces
    movep   #$000E07,X:M_IPRC        ; IRQA/IRQD/SSI level 3 interrupts. edge sensitive
    move    #0,omr
    movec   #0,sp


    move    #$40,r6              ; initialize stack pointer
    move    #-1,m6               ; linear addressing
    move    #RX_BUFF_BASE,x0
    move    x0,x:RX_PTR                 ; Initialize the rx pointer
    move    #TX_BUFF_BASE,x0
    move    x0,x:TX_PTR                 ; Initialize the tx pointer

; --- INIT THE CODEC ---

    jsr     ada_init             ; Jump to initialize the codec

; --- Init the HI08 ---
       movep   #$002e0e,x:M_HPCR     ; Set Port Control signals
       movep   #$000060,x:M_HBAR     ; Set Base Address for HC11 to $6000
       movep   #$000004,x:M_HCR      ; Set Host Command Interrupt Enable bit
       movep   #$002e4e,x:M_HPCR     ; Enable HI08
       bset    #$01,x:$fffffe        ; $fffffe = IPR-P (Interrupt Priority Register P)
       bset    #$02,x:$fffffe        ; Set HI08 interrupt priority to 3 (non-maskable)


; Initialize Filter Parameters
    move    #state1,r3           ;point to filter state1
    move    #state2,r1           ;point to filter state2
    move    #no_filter,r4        ;Initialize for unity filter.
    move    #5*nsec-1,m4         ;addressing modulo 5*nsec
    clr     a                    ;initialize internal state storage
       nop ; avoid pipeline stall
    move    a,y:left_a
    move    a,y:right_a
    rep     #4                   ;*    zero state1       (BAK)
    move    a,x:(r3)+            ;*
    rep     #4                   ;*    zero state2       (BAK)
    move    a,x:(r1)+            ;*
    move    #state1_r,r3         ;point to filter state1  (BAK)
    move    #state2_r,r1         ;point to filter state2  (BAK)
    rep     #4                   ;*    zero state1_r                  (BAK)
    move    a,x:(r3)+            ;*                        (BAK)
    rep     #4                   ;*    zero state2_r                  (BAK)
    move    a,x:(r1)+            ;*                        (BAK)
    move              y:(r4)+,y0  ;a must be initially zero ,y0=b10/2
    move    y0,y:left_y0
    move    y0,y:right_y0
    move    #filter,r7        ; Load filter location.

; Initialize Digital Oscillator Parameters
       movep   #$FF310C,x:M_CRB0   ; Enable the SSI interrupts.
       move    #coeff_a,x0
       move    x0,x:DOSC_BUFF_BASE    ; Load coeff. for osc. a.
       move    #s1_a,x0
       move    x0,x:DOSC_BUFF_BASE+1  ; Load s1 for osc. a.
       move    #s2_a,x0
       move    x0,x:DOSC_BUFF_BASE+2  ; Load s2 for osc. a.

; Main Loop
loop_1
       jset    #2,x:M_SSISR0,*              ; Wait for frame sync to pass.
       jclr    #2,x:M_SSISR0,*              ; Wait for frame sync.

       move    #CTRL_WD_12,y0       ; headphones, line out, mute spkr, no attn.
       move    y0,x:TX_BUFF_BASE+2
       move    #CTRL_WD_34,y0         ; no input gain, monitor mute
```

```
        move    y0,x:TX_BUFF_BASE+3
        move    #DOSC_BUFF_BASE,r5      ; Load pointer to osc's coeff, sr1 and sr2
        jsr     dosc_sin                ; Call oscillator routine.
        move    a,x0                    ; Copy new tone into x0.
        move    x:RX_BUFF_BASE,a        ; Load the left channel input.
        add     x0,a                    ; Add the tone.
          nop ; avoid pipeline stall
        move    a,x:LEFT_HUM
        move    x:RX_BUFF_BASE+1,b      ; Load the right channel input.
        add     x0,b                    ; Add the tone.
          nop ; avoid pipeline stall
        move    b,x:RIGHT_HUM

        move    #state1,r3
        move    #state2,r1
        jsr     bst60ev6_l              ;bandstop filter at 60 Hz (Elliptical)
        move    a,x:TX_BUFF_BASE        ; Put value in left channel tx.
        move    #state1_r,r3
        move    #state2_r,r1
        jsr     bst60ev6_r              ;bandstop filter at 60 Hz (Elliptical)
        move    b,x:TX_BUFF_BASE+1      ; Put value in right channel tx.

        jmp     loop_1                  ; Loop back.


;*****************************************************************************
; The following subroutine calculates the next sinusoidal output value as a
; function by the digital oscillator given that r5 points to the memory
; location that contains the "coeff" value followed by the memory location
; that contains the "s1" value, followed by the memory location that contains
; the "s2" value.  The formula and block diagram of the oscillator are:
;
;               s1[n] = coeff*s1[n-1] - s2[n-2] = coeff*s1[n-1] - s1[n-2]
;
;
;                 _______     _______
;               |       |  s1   |       |  s2
;           +--->|  z^-1 |--+--->|  z^-1 |----+
;           |    |_______|  |   |_______|    |
;           |                |                |
;           |             ___V___          ___V___
;           |           |       |        |       |
;           |           | coeff |        |  -1   |
;           |           |_______|        |_______|
;           |               |                |
;           |               |                |
;           |               |                |
;           |               +----->( + )<-----+
;           |                      |
;           |                      |
;           +----------------------+---------> sine output
dosc_sin

        move    x:(r5)+,x0              ; Load coeff
        move    x:(r5)+,y0              ; Load s1 into a.
        move    x:(r5)-,b               ; Load s2 into b, r5 points to s1.
        mpy     x0,y0,a                 ;Get coef*s1 in a in 2:14 format
        subl    b,a          ;y0,b      ;Get (coef*s1 -s2)=sin_val in a
                                        ;  in fractional format
          nop ; avoid pipeline stall
        move    a,x:(r5)+               ; Save new s1.
        move    y0,x:(r5)               ; Save new s2.
        rts


;BST60EL6

;
; This code segment implements cascaded biquad sections in transpose form
; The "bit_24" section of the code implements the filter with 24 bit data
; and 24 bit coefficients.  The "bit_16" section of the code implements the
; filter with coefficients that are truncated to 16 bits.  The "bit_16" code
```

```
; also truncates the data to 16 bits before writing any accumulator value
; to memory.  This "bit_16" code is to simulate the performance of a 16 bit
; processor implementing the same filter.  In actuality, the filtering
; performed by "bit_16" will yield a better performance than that which
; a 16 bit processor would yield.
;

;
;  multiple shift left macro
;
mshl      macro    scount,acc
          if       scount
          rep      #scount
          asl      acc
          endif
          endm

bst60ev6_l                          ;Bandstop filter at 60Hz (Elliptical
       ori     #$08,mr              ;set scaling mode
       move    x:LEFT_HUM,y1        ;load left signal + Hum

       move    y:left_a,a
       move    y:left_y0,y0
       jsr     (r7)                 ; Call cascade biquad routine.
       move    y0,y:left_y0
       move    a,y:left_a
       move    y1,a
       mshl    scount,a              ;bring gain back to 0 dB
       andi    #$f7,mr              ;disable scaling mode?
       rts

bst60ev6_r                          ;Bandstop filter at 60Hz (Elliptical
       ori     #$08,mr              ;set scaling mode
       move    x:RIGHT_HUM,y1       ;load right signal + Hum

       move   y:right_a,a
       move   y:right_y0,y0
       jsr    (r7)                 ; Call cascade biquad routine.
       move   y0,y:right_y0
       move   a,y:right_a
       move   y1,b
       mshl   scount,b                     ;bring gain back to 0 dB
       andi    #$f7,mr              ;disable scaling mode?
       rts


;     assumes each section's coefficients are divided by 2
;
filter  EQU     *
       do      #nsec,_end_filter                    ;do each section
       macr    y0,y1,a  x:(r1),b   y:(r4)+,y0       ;a=x(n)*bi0/2+wi1/2,b=wi2,y0=bi1/2
         nop ; avoid pipeline stall
       asr     b        a,x0                        ;b=wi2/2,x0=y(n)
       mac     y0,y1,b            y:(r4)+,y0        ;b=x(n)*bi1/2+wi2/2,y0=ai1/2
       macr    x0,y0,b            y:(r4)+,y0        ;b=b+y(n)*ai1/2,y0=bi2/2
         nop ; avoid pipeline stall
       mpy     y0,y1,b  b,x:(r3)+  y:(r4)+,y0       ;b=x(n)*bi2/2,save wi1,y0=ai2
       macr    x0,y0,b  x:(r3),a   a,y1             ;b=b+y(n)*ai2/2,a=next iter wi1,
                                                    ;y1=output of section i
         nop ; avoid pipeline stall
       asr     a        b,x:(r1)+  y:(r4)+,y0       ;a=next iter wi1/2,save wi2,
                                                    ;y0=next iter bi0
_end_filter
       rts                                          ; Return from filter routine.

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

       include 'ada_init.asm'

;*************************************************************************
echo           ;added to provide complince with the same
```

```
            ; vectors.asm that echo.asm and fltr_tst.asm use
end
```