STE Developer Addendum

The Atari **ST**^E

Compatible with ST, 1000s of software titles available.

New

- Extended color palette of 4096 colors, from 512
- Hardware support for horizontal and vertical scrolling
- Ready for external GENLOCK
- Stereo 8 bit PCM sound
- Light gun, paddle and new joystick ports.
- 256K ROM from 192K includes
  - Move as well as copy files
  - Rename folders
  - Autoboot GEM applications
  - New file selector
  - Faster desktop
  - Large palette support
  - Fast hard-disk support
  - Folder limitations lifted
  - Memory management improved
  - Keyboard reset

STE Developer Addendum

This addendum is a set of documents that allows the ST developer to use the new features of the STE. These new features are in the areas of graphics, sound and interface ports.

The STE has a palette of 4096 colors compared to the ST palette of 512 colors. Also the STE has hardware support for vertical and horizontal scrolling. Support has also been added for external GENLOCK.

Sound on the STE has the ST sound as well as 8 bit stereo DMA sound with variable playback frequencies.

The STE also has two new controller ports that allow for new joysticks as well as a light gun and paddle controllers.

Genlock and the STE

The ST (and STE) chip set have the ability to accept external sync. This is controlled by bit 0 at FF820A, as documented in the ST Hardware Specification. This was done to allow the synchronization of the ST video with an external source (a process usually known as GENLOCK). However, in order to do this reliably the system clock must also be phase-locked (or synchronized in some other way) to the input sync signals. No way to do this was provided in the ST, as a result the only GENLOCKs available are internal modifications (usually for the MEGA).

The STE allows this to be done without opening the case. To inject a system clock ground pin three (GPO) on the monitor connector and then inject the clock into pin 4 (mono detect). The internal frequency of this clock is 32.215905 MHz (NTSC) and 32.084988 MHz (PAL). Note: DO NOT SWITCH CLOCK SOURCE WHILE THE SYSTEM IS ACTIVE.

As a result of this GPO is no longer available.

Controllers

FF9200

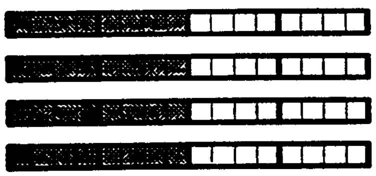
Fire Buttons

FF9202

Joy 3 Joy 1 Joy 2 Joy 0

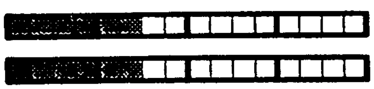
Joy sticks.
Four new joystick ports are added. These ports are controlled directly by the 68000. The current state may be sampled at any time by reading the above locations. Joystick 0 and Joystick 2 direction bits are read/write. If written to they will be driven until a read is performed. Similarly, they will not be driven after a read until a write is performed.

FF9210 (X Paddle 0)
FF9212 (Y Paddle 0)
FF9214 (X Paddle 1)
FF9216 (Y Paddle 1)



Paddles.
One pair of paddles can be plugged into Joystick 0 (Paddle 0). A second set can be plugged into Joystick 1 (Paddle 1). The current position of each of the four paddles is reported at these locations. The fire buttons are the same as for the respective joystick. The triggers for the paddles are read as bits one and two of FF9202 (JOY0 Left and Right)

FF9220 (X Position)
FF9222 (Y Position)



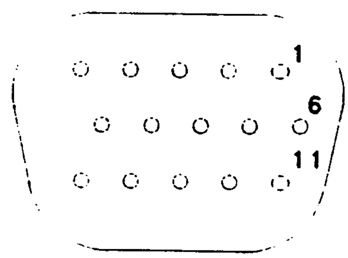
Light Gun / Pen.
A light gun or pen can be plugged into Joystick 0. The current position that the gun or pen is pointing to is reported by these registers. The position is accurate to within (X direction only):

4 Pixels in 320x200 Mode
8 Pixels in 640x200 Mode
16 Pixels in 640x400 Mode

Accurate to 1 pixel in the Y direction in all modes. Accuracies do not account for the quality of the light gun or pen. Note that the X position is given in pixels for 320x200 only. In order to get correct results in 640x200 mode this number needs to be shifted left one bit and in 640x400 mode this number needs to be shifted left two bits.

New Controller Pinout

This pinout is for ports 0 and 1.
Ports 2/3 are on the other DB15 connector.



1 UP 0
2 DN 0
3 LT 0
4 RT 0
5 PAD 0Y
6 FIRE 0
7 VCC
8 NC
9 GND
10 FIRE 1
11 UP 1
12 DN 1
13 LT 1
14 RT 1
15 PAD 0X

Video Modifications

FF8204  (High)

FF8206 

FF8208  (Low)

Video Address Counter.
Now read/write. Allows update of the video refresh address during the frame. The effect is immediate, therefore it should be reloaded carefully (or during blanking) to provide reliable results.

FF820C 

Low byte of the video base address. This register completes the set on ST
Allows positioning screen on word boundaries and thus vertical scrolling.

FF820E 

Offset to next line.
Number of words from end of line to beginning of next line minus one. Allows virtual screen to be wider than physical screen. Acts like an ST when cleared. Cleared at reset.

FF8240
through
FF825E

Color Pallete.
A fourth bit of resolution is added to each color. Note that the least significant bit is added above the old most significant bit to remain compatible with the ST.

FF8264 

Horizontal Bit-wise Scroll.
Delays the start of screen by the specified number of bits.

How to Implement Fine Scrolling on the STE.

The purpose of this document is to describe how to use the capabilities of the STE to achieve bit-wise fine-scrolling and vertical split screens. Horizontal and vertical scrolling are discussed and an example program is provided. Split screen effects are discussed and an example program with multiple independent scrolling regions is provided.

Three new registers are provided to implement fine-scrolling and split screen displays:

1) HSCROLL - This register contains the pixel scroll offset. If it is zero, this is the same as an ordinary ST. If it is non-zero, it indicates which data bits constitute the first pixel from the first word of data. That is, the leftmost displayed pixel is selected from the first data word(s) of a given line by this register.

2) LINEWID - This register indicates the number of extra words of data (beyond that required by an ordinary ST at the same resolution) which represent a single display line. If it is zero, this is the same as an ordinary ST. If it is non-zero, that many additional words of data will constitute a single video line (thus allowing virtual screens wider than the displayed screen). *CAUTION*- In fact, this register contains the word offset which the display processor will add to the video display address to point to the next line. If you are actively scrolling (HSCROLL <> 0), this register should contain the additional width of a display line *minus one data fetch* (in low resolution one data fetch would be four words, one word for monochrome, etc.).

3) VBASELO - This register contains the low-order byte of the video display base address. It can be altered at any time and will affect the next display processor data fetch. It is recommended that the video display address be altered only during vertical and horizontal blanking or display garbage may result.

These registers, when used in combination, can provide several video effects. In this document we will discuss only fine-scrolling and split-screen displays.

Fine Scrolling:
Many games use horizontal and vertical scrolling techniques to provide virtual playfields which are larger than a single screen. We will first discuss vertical scrolling (line-wise), then horizontal scrolling (pixel-wise) and finally the example program "neowall.s" which combines both.

Vertical Scrolling:
To scroll line-wise, we simply alter the video display address by one line each time we wish to scroll one line. This is done at vertical blank interrupt time by writing to the three eight-bit video display address registers to define a twenty-four-bit pointer into memory. Naturally, additional data must be available to be displayed. We might imagine this as a tall, skinny screen which we are opening a window onto for the user. The video display address registers define where this window will start.

Horizontal Scrolling:
To scroll horizontally we might also adjust the video display address. If that was all we did, we would find that the screen would jump sideways in sixteen pixel increments. To achieve smooth pixel-wise scrolling we must use the HSCROLL register to select where within each sixteen pixel block we wish to start displaying data to the screen. Finally, we must adjust the LINEWID register to reflect both the fact that each line of video data is wider than a single display line and any display processor fetch incurred by a non-zero value of HSCROLL. All this is done at vertical blank interrupt time. Naturally, additional data must be available to be

displayed. We might imagine this as an extremely wide screen which we are opening a window onto for the user. These registers define where this window will start.

For Example:
The program "neowall.s" reads in nine NEOchrome™ picture files, organizes them into a three by three grid and allows the user to scroll both horizontally and vertically over the images. The heart of this program (the only interesting thing about it actually) is the vertical blank interrupt server. This routine first determines the pixel offset and loads it into HSCROLL. The LINEWID register is now set to indicate that each virtual line is three times longer than the actual display width. If we are actively scrolling, this amount is reduced to reflect the additional four-plane data fetch which will be caused by the scrolling. Finally the video display address is computed to designate a window onto the grid of pictures. This twenty-four-bit address determines where the upper-left corner of the displayed region begins in memory. Thus, every frame an arbitrary portion of the total image is selected for display. The speed and resolution of this scrolling technique is limited only by the dexterity of the user.

Split Screen:
In many applications it is desirable to subdivide the screen into several independent regions. On the STE you may reload some video registers on a line-by-line basis (using horizontal blanking interrupts) to split the screen vertically into multiple independent regions. A single screen no longer need be a contiguous block of storage, but could be composed of dozens of strips which might reside in memory in any order. The same data could be repeated on one or more display lines. Individual regions might each have their own individual data and scrolling directions.

For Example:
The program "hscroll.s" reads in a NEOchrome™ picture file and duplicates each line of the image. This, combined with the proper use of LINEWID, effectively places two copies of the same picture side-by-side. Next, both vertical and horizontal blanking interrupt vectors are captured and the horizontal blanking interrupt is enabled in counter mode. To prevent flicker caused by keyboard input, the IKBD/MIDI interrupt priority is lowered below that of the HBL interrupt. Note that the program 'main loop' doesn't even call the BIOS to check the keyboard, since the BIOS sets the IPL up and causes flicker by locking out horizontal interrupts - this may cause trouble for programs in the real world. The screen is effectively divided into ten regions which scroll independently of one another. There are two ten-element arrays which contain the base address of each region and its current scroll offset. At vertical blank interrupt time we compute the final display values for each region in advance and store them into a third array. We then initialize the display processor for the first region and request an interrupt every twenty lines (actually every twenty horizontal blankings). During each horizontal interrupt service, we quickly reload the video display address registers and the HSCROLL register. This must be done immediately - before the display processor has time to start the current line or garbage may result. Note that horizontal blank interrupts are triggered by the display processor having finished reading the previous data line. You have approximately 144 machine cycles to reload the HSCROLL and video display registers before they will be used again by the display processor. Finally, the LINEWID register is set, this need only be done before the processor finishes reading the data for the current display line. We then pre-compute the data we will need for the next horizontal interrupt to shave few more cycles off the critical path and exit.

```
                                ;
                                ;       HSCROLL.S   Horizontal Scrolling Demo
                                ;       THE ONE LINE VERSION
                                ;
                                ;       Copyright 1988 ATARI CORP.
                                ;       Started 9/12/88 .. Rob Zdybel
                                ;

                                        .text
                                        .include atari
                                        .list

                                ;
                                ;       HARDWARE CONSTANTS
                                ;
         =FFFF820D              vbaselo =       $ffff820d               ; Video Base Address (lo)
         =FFFF820F              linewid =       $ffff820f               ; Width of a scan-line (Words, minus 1)
         =FFFF8265              hscroll =       $ffff8265               ; Horizontal scroll count (0 .. 15)

                                ;
                                ;       SYSTEM CONSTANTS
                                ;
         =00000070              vblvec  =       $70             ; System VBlank Vector
         =00000118              ikbdvec =       $118            ; IKBD/MIDI (6850) Vector
         =00000120              hblvec  =       $120            ; Horizontal Blank Counter (68901) Vector

                                ;
                                ;       LOCAL CONSTANTS
                                ;


                                ;
                                ;       System Initialization
                                ;
                                start:
00000000  2A4F                          move.l  a7,a5
00000002  2E7Cxxxxxxxx                  move.l  #mystack,a7     ; Get Our Own Local Stack
00000008  2A6D0004                      move.l  4(a5),a5        ; a5 = basepage address
0000000C  202D000C                      move.l  TEXTSZ(a5),d0
00000010  D0AD0014                      add.l   DATASZ(a5),d0
00000014  D0AD001C                      add.l   BSSSZ(a5),d0
00000018  D0BC00000100                  add.l   #$100,d0        ; RAM req'd = text+bss+data+BasePageLength
0000001E  2800                          move.l  d0,d4           ; d4 = RAM req'd
                                        Mshrink a5,d0           ; Return Excess Storage
00000020  2F00                  •       move.l  d0,-(sp)
00000022  2F0D                  •       move.l  a5,-(sp)
00000024  4267                  •       clr.w   -(sp)
                                •       Gemdos $4a,12
00000026  3F3C004A              •       move.w  #$4a,-(sp)
0000002A  4E41                  •       trap    #1
                                •       .if $c <= 8
                                -       addq    #$c,sp
                                -       .else
0000002C  DEFC000C              •       add.w   #$c,sp
                                •       .endif
                                ;
                                ;       Other Initialization
                                ;
                                        Super                   ; enter supervisor mode
00000030  42A7                  •       clr.l   -(sp)
00000032  3F3C0020              •       move.w  #$20,-(sp)
00000036  4E41                  •       trap    #1
00000038  5C4F                  •       addq    #6,sp
0000003A  2F00                  •       move.l  d0,-(sp)        ; WARNING - Old SSP saved on stack.

                                        Fgetdta
                                •       Gemdos $2f,2
0000003C  3F3C002F              •       move.w  #$2f,-(sp)
00000040  4E41                  •       trap    #1
                                •       .if $2 <= 8
00000042  544F                  •       addq    #$2,sp
                                •       .else
                                -       add.w   #$2,sp
                                -       .endif
00000044  2840                          move.l  d0,a4
00000046  D8FC001E                      adda    #30,a4          ; a4 = Filename ptr
                                        Fsfirst #neofile,#0
0000004A  3F3C0000              •       move.w  #$0,-(sp)
0000004E  2F3Cxxxxxxxx          •       move.l  #neofile,-(sp)
                                •       Gemdos $4e,8
00000054  3F3C004E              •       move.w  #$4e,-(sp)
00000058  4E41                  •       trap    #1
                                •       .if $8 <= 8
0000005A  504F                  •       addq    #$8,sp
                                •       .else
                                -       add.w   #$8,sp
                                -       .endif
0000005C  4A40                          tst     d0
0000005E  6B00xxxx                      bmi     abort           ; IF (No NEO files) ABORT
                                        Fopen   a4,#0
00000062  3F3C0000              •       move.w  #$0,-(sp)
00000066  2F0C                  •       move.l  a4,-(sp)
```

```
                             •         Gemdos $3d,8
00000068  3F3C003D           •         move.w  #$3d,-(sp)
0000006C  4E41               •         trap    #1
                             •         .if $8 <= 8
0000006E  504F               •         addq    #$8,sp
                             •         .else
                             -         add.w   #$8,sp
                             -         .endif
00000070  4A40                         tst     d0
00000072  6B00xxxx                     bmi     abort           ; IF (Error opening file) ABORT

00000076  33C0xxxxxxxx                 move    d0,handle
                                       Fread   d0,#32128,#neobuff
0000007C  2F3Cxxxxxxxx       •         move.l  #neobuff,-(sp)
00000082  2F3C00007D80       •         move.l  #$7d80,-(sp)
00000088  3F00               •         move.w  d0,-(sp)
                             •         Gemdos $3f,12
0000008A  3F3C003F           •         move.w  #$3f,-(sp)
0000008E  4E41               •         trap    #1
                             •         .if $c <= 8
                             -         addq    #$c,sp
                             -         .else
00000090  DEFC000C           •         add.w   #$c,sp
                             •         .endif
00000094  4A80                         tst.l   d0
00000096  6B00xxxx                     bmi     abort           ; IF (File Read Error) ABORT
                                       Fclose  handle
0000009A  3F39xxxxxxxx       •         move.w  handle,-(sp)
                             •         Gemdos $3e,4
000000A0  3F3C003E           •         move.w  #$3e,-(sp)
000000A4  4E41               •         trap    #1
                             •         .if $4 <= 8
000000A6  504F               •         addq    #$4,sp
                             •         .else
                             -         add.w   #$4,sp
                             -         .endif
000000A8  4A40                         tst     d0
000000AA  6B00xxxx                     bmi     abort           ; IF (Error Closing a file) ABORT

000000AE  45F9xxxxxxxx                 lea     neobuff+4,a2
000000B4  41F88240                     lea     palette,a0
000000B8  43F9xxxxxxxx                 lea     oldpal,a1
000000BE  303C000F                     move    #15,d0
000000C2  32D0                .ploop:  move.w  (a0),(a1)+      ; save old color palette
000000C4  30DA                         move.w  (a2)+,(a0)+     ; create new color palette
000000C6  51C8FFFA                     dbra    d0,.ploop

000000CA  303C00A0                     move    #160,d0         ; Double each display line
000000CE  41F9xxxxxxxx                 lea     bigbuff,a0
000000D4  43F9xxxxxxxx                 lea     neobuff+128,a1
000000DA  343C00C7                     move    #199,d2
000000DE  323C0027            .linlp:  move    #39,d1          ; FOR (200 Lines) DO
000000E2  21910000            .dublp:  move.l  (a1),(a0,d0)    ; duplicate line
000000E6  20D9                         move.l  (a1)+,(a0)+
000000E8  51C9FFF8                     dbra    d1,.dublp
000000EC  D0C0                         adda    d0,a0
000000EE  51CAFFEE                     dbra    d2,.linlp

000000F2  41F9xxxxxxxx                 lea     baseaddr,a0
000000F8  43F9xxxxxxxx                 lea     xoffset,a1
000000FE  45F9xxxxxxxx                 lea     bigbuff,a2
00000104  303C0009                     move    #9,d0
00000108  32FC0000            .strip:  move    #0,(a1)+        ; FOR (10 Strips) DO Init base and offset
0000010C  20CA                         move.l  a2,(a0)+
0000010E  D4FC1900                     adda    #320*20,a2
00000112  51C8FFF4                     dbra    d0,.strip

00000116  23F80118xxxxxxxx             move.l  ikbdvec,oldikbd
0000011E  21FCxxxxxxxx0000             move.l  #ikbd,ikbdvec   ; IPL 5 hack for IKBD/MIDI

00000126  23F80070xxxxxxxx             move.l  vblvec,oldvbl
0000012E  21FCxxxxxxxx0000             move.l  #vbl,vblvec     ; Capture System VBlank Interrupt

00000136  21FCxxxxxxxx0000             move.l  #hbl,hblvec     ; Capture HBlank Interrupt
0000013E  08F80000FA13                 bset.b  #0,imra
00000144  08F80000FA07                 bset.b  #0,iera         ; Enable Hblank


                              ;
                              ;        Scrolling Demo loop
                              ;
                              wavelp:
                                       Bconstat CON            ; Keyboard Polling
0000014A  3F3C0002           •         move.w  #CON,-(sp)
                             •         Bios 1,4
0000014E  3F3C0001           •         move.w  #$1,-(sp)
00000152  4E4D               •         trap    #13
                             •         .if $4 <= 8
00000154  504F               •         addq    #$4,sp
                             •         .else
                             -         add.w   #$4,sp
                             -         .endif
```

```
00000156  4A40                        tst     d0
00000158  6700xxxx                    beq     noexit        ; IF (Keyboard Input Available) THEN
                                      Bconin  CON
0000015C  3F3C0002            •       move.w  #CON,-(sp)
                              •       Bios 2,4
00000160  3F3C0002            •       move.w  #$2,-(sp)
00000164  4E4D                •       trap    #13
                              •       .if $4 <= 8
00000166  584F                •       addq    #$4,sp
                              •       .else
                              -       add.w   #$4,sp
                              -       .endif
00000168  0C3C0003                    cmp.b   #'C'-64,d0
0000016C  6700xxxx                    beq     exit          ; CTRL-C ==> EXIT
                              noexit:
00000170  6808                        bra     waveip
                              exit:
                              ;
                              ;       System Tear-Down
                              ;
00000172  08B80000FA07                bclr.b  #0,iera
00000178  08B80000FA13                bclr.b  #0,imra       ; Disable Hblank
0000017E  21F9xxxxxxxx0000            move.l  oldikbd,ikbdvec ; Restore System IKBD/MIDI Interrupt
00000186  21F9xxxxxxxx0000            move.l  oldvbl,vblvec ; Restore System VBlank Interrupt

                                      Gettime
                              •       Xbios   $17,2
0000018E  3F3C0017            •       move.w  #$17,-(sp)
00000192  4E4E                •       trap    #14
                              •       .if $2 <= 8
00000194  544F                •       addq    #$2,sp
                              •       .else
                              -       add.w   #$2,sp
                              -       .endif
00000196  23C0xxxxxxxx                move.l  d0,vbltemp    ; Get IKBD Date/Time
                                      Tsettime d0
0000019C  3F00                •       move    d0,-(sp)
                              •       Gemdos  $2d,4
0000019E  3F3C002D            •       move.w  #$2d,-(sp)
000001A2  4E41                •       trap    #1
                              •       .if $4 <= 8
000001A4  584F                •       addq    #$4,sp
                              •       .else
                              -       add.w   #$4,sp
                              -       .endif
                                      Tsetdate vbltemp      ; Set GEMDOS Time and Date
000001A6  3F39xxxxxxxx        •       move    vbltemp,-(sp)
                              •       Gemdos  $2b,4
000001AC  3F3C002B            •       move.w  #$2b,-(sp)
000001B0  4E41                •       trap    #1
                              •       .if $4 <= 8
000001B2  584F                •       addq    #$4,sp
                              •       .else
                              -       add.w   #$4,sp
                              -       .endif

000001B4  41F9xxxxxxxx                lea     oldpal,a0
000001BA  43F88240                    lea     palette,a1
000001BE  303C000F                    move    #15,d0
000001C2  32D8                .unplp: move.w  (a0)+,(a1)+
000001C4  51C8FFFC                    dbra    d0,.unplp     ; restore old color palette

                              abort:  User                  ; return to user mode
                              •       Gemdos  $20,6
000001C8  3F3C0020            •       move.w  #$20,-(sp)
000001CC  4E41                •       trap    #1
                              •       .if $6 <= 8
000001CE  5C4F                •       addq    #$6,sp
                              •       .else
                              -       add.w   #$6,sp
                              -       .endif
                                      Pterm0                ; return to GEMDOS
000001D0  4267                •       clr.w   -(sp)
000001D2  4E41                •       trap    #1
000001D4  4AFC                •       illegal


                              ;
                              ;       VBL     Vertical-Blank Interrupt Server
                              ;
                              vbl:
000001D6  48E7C0E0                    movem.l d0-d1/a0-a2,-(sp)

000001DA  41F9xxxxxxxx                lea     video,a0      ; a0 = Display list (scroll.base)
000001E0  43F9xxxxxxxx                lea     xoffset,a1    ; a1 = Xoffset list
000001E6  45F9xxxxxxxx                lea     baseaddr,a2   ; a2 = Base address list
000001EC  323C0009                    move    #9,d1
                              .reglp:                       ; FOR (10 scrolling regions) DO
000001F0  3011                        move    (a1),d0       ; d0 = current Xoffset
000001F2  08010000                    btst.l  #0,d1
000001F6  6600xxxx                    bne     .odd
000001FA  5240                        addq    #1,d0         ; EVEN --> Increment
```

```
000001FC  B07C00A0              cmp     #160,d0
00000200  6D00xxxx              blt     .join
00000204  7000                  moveq   #0,d0          ; Wrap-up
00000206  6000xxxx              bra     .join
0000020A  5340          .odd:   subq    #1,d0          ; ODD --> Decrement
0000020C  6C00xxxx              bge     .join
00000210  303C009F              move    #159,d0        ; Wrap-down
00000214  3280          .join:  move    d0,(a1)        ; New Xoffset
00000216  E240                  asr     #1,d0
00000218  C0BC0000FFF8          and.l   #$0fff8,d0     ; d0 = byte offset within line
0000021E  D09A                  add.l   (a2)+,d0       ; d0 = Regions video base
00000220  2080                  move.l  d0,(a0)
00000222  3019                  move    (a1)+,d0
00000224  C07C000F              and     #$0f,d0        ; d0 = Regions horizontal scroll count
00000228  1080                  move.b  d0,(a0)
0000022A  5880                  addq.l  #4,a0
0000022C  51C9FFC2              dbra    d1,.reglp

00000230  41F9xxxxxxxx          lea     video,a0
00000236  1018                  move.b  (a0)+,d0
00000238  11C08265              move.b  d0,hscroll
0000023C  11D88205              move.b  (a0)+,vcounthi
00000240  11D88207              move.b  (a0)+,vcountmid
00000244  11D88209              move.b  (a0)+,vcountlo  ; Initialize first region

00000248  323C0050              move    #80,d1         ; Double normal ST line width
0000024C  4A00                  tst.b   d0
0000024E  6700xxxx              beq     .zero          ; IF (non-zero scroll count) Reduce line width
00000252  5941                  subq    #4,d1
00000254  11C1820F      .zero:  move.b  d1,linewid

00000258  2018                  move.l  (a0)+,d0
0000025A  E198                  rol.l   #8,d0
0000025C  23C0xxxxxxxx          move.l  d0,videodata   ; Init next lines data
00000262  23C8xxxxxxxx          move.l  a0,videoptr    ; Init display list ptr

00000268  11FC0000FA1B          move.b  #0,tbcr
0000026E  11FC0014FA21          move.b  #20,tbdr       ; Interrupt every twenty HBlanks
00000274  11FC0008FA1B          move.b  #8,tbcr

0000027A  4CDF0703              movem.l (sp)+,d0-d1/a0-a2
0000027E  4EF9                  .dc.w   $4ef9
00000280  00000000      oldvbl: .dc.l   0              ; JMP (Old-Vblank)
00000284  4AFC                  illegal


                        ;
                        ;       IKBD    IKBD/MIDI Interrupt Server
                        ;
                        ikbd:
00000286  3F00                  move    d0,-(sp)

00000288  40C0                  move    sr,d0
0000028A  C07CF8FF              and     #$f8ff,d0
0000028E  807C0500              or      #$500,d0
00000292  46C0                  move    d0,sr          ; Set IPL down to 5

00000294  301F                  move    (sp)+,d0
00000296  4EF9                  .dc.w   $4ef9
                        oldikbd:
00000298  00000000              .dc.l   0              ; JMP (Old-IKBD)
0000029C  4AFC                  illegal


                        ;
                        ;       HBL     *ONE LINE* Horizontal-Blank Interrupt Server
                        ;
                        hbl:
0000029E  48E78080              movem.l d0/a0,-(sp)    ;                       (44+28=72)

000002A2  2039xxxxxxxx          move.l  videodata,d0   ; d0 = vcount/scroll    (20)
000002A8  41F88205              lea     vcounthi,a0    ; a0 = movep base       (8)
000002AC  11C08265              move.b  d0,hscroll     ; set HScroll           (12)
000002B0  01C80000              movep.l d0,(a0)        ; set VideoBase         (24)
                                                       ;                       (total = 136+ cycles)
000002B4  4A00                  tst.b   d0
000002B6  6700xxxx              beq     .zero          ; IF (non-zero scroll count) Reduce line width
000002BA  11FC004C820F          move.b  #76,linewid
000002C0  6000xxxx              bra     .join
000002C4  11FC0050820F  .zero:  move.b  #80,linewid
                        .join:
000002CA  2079xxxxxxxx          move.l  videoptr,a0
000002D0  2018                  move.l  (a0)+,d0
000002D2  E198                  rol.l   #8,d0
000002D4  23C0xxxxxxxx          move.l  d0,videodata   ; Init next regions data
000002DA  23C8xxxxxxxx          move.l  a0,videoptr

000002E0  4CDF0101              movem.l (sp)+,d0/a0
000002E4  08B80000FA0F          bclr.b  #0,isra        ; Clear In-Service bit
000002EA  4E73                  rte


                        ;
                        ;       DATA STORAGE
```

```
                                         ;
000002EC                                         .data
                                         neofile:                           ; NEO filename search string
00000000  2A2E6E656F00                           .dc.b   "*.neo",0

                                                 .even


                                         ;
                                         ;       RANDOM DATA STORAGE
                                         ;
00000006                                          bss

                                         oldpal:
00000000 =00000010                               .ds.l   16                 ; Original color palette
                                         handle:                            ; Active Handle
00000040 =00000001                               .ds.w   1

                                         baseaddr:                          ; Image Base address for each strip
00000042 =0000000A                               .ds.l   10
                                         xoffset:                           ; Pixel-offset for each strip
0000006A =0000000A                               .ds.w   10
                                         video:                             ; HScroll and Video Base address for each strip
0000007E =0000000A                               .ds.l   10
                                         videoptr:                          ; Display list ptr
000000A6 =00000001                               .ds.l   1
                                         videodata:                         ; Next regions display info
000000AA =00000001                               .ds.l   1

                                         neobuff:                           ; NEO-Image Buffer
000000AE =00007D80                               .ds.b   32128
                                         bigbuff:                           ; Mega-Image Buffer
00007E2E =0000FA00                               .ds.b   2*32000

                                         vbltemp:                           ; Vblank Temporary Storage
0001782E =00000001                               .ds.l   1

00017832 =00000100                               .ds.l   256                ; (stack body)
                                         mystack:
00017C32 =00000001                               .ds.l   1                  ; Local Stack Storage

                                                 .end
```

Symbol Table

.dublp	000000E2	t
.join	00000214	t
.join	000002CA	t
linlp	000000DE	t
.odd	0000020A	t
.ploop	000000C2	t
.reglp	000001F0	t
.strip	00000188	t
unplp	000001C2	t
.zero	00000254	t
.zero	000002C4	t
AUX	00000001	ea
BBASE	00000018	a
BLEN	0000001C	a
BPSZ	00000100	ea
BSIZE	0000000A	a
BSSSZ	0000001C	ea
CMDLINE	00000080	a
CON	00000002	ea
CR	0000000D	ea
CURS_BLINK	00000002	ea
CURS_GETRATE	00000005	ea
CURS_HIDE	00000000	ea
CURS_NOBLINK	00000003	ea
CURS_SETRATE	00000004	ea
CURS_SHOW	00000001	ea
DATASZ	00000014	ea
DBASE	00000010	a
DLEN	00000014	a
DSIZE	00000006	a
DTA	00000020	a
ENVIR	0000002C	a
FILE_ID	00000000	a
HEADSIZE	0000001C	ea
HITPA	00000004	a
IKBD	00000004	ea
LF	0000000A	ea
LOWTPA	00000000	a
MIDI	00000003	ea
MYDTA	00000020	ea
PARENT	00000024	a
PRT	00000000	ea
RAWCON	00000005	ea
SSIZE	0000000E	a
TAB	00000009	ea
TBASE	00000008	a
TEXTSZ	0000000C	ea
TLEN	0000000C	a
TSIZE	00000002	a
XXX1	00000012	a
XXX2	00000016	a
XXX3	0000001A	a
XXXX	00000028	a
______md	0000049E	ea
_autopath	000004CA	ea
_bootdev	00000446	ea
_bufl	000004B2	ea
_cmdload	00000482	ea
_drvbits	000004C2	ea
_dskbufp	000004C6	ea
_frclock	00000466	ea
_fverify	00000444	ea
_hz_200	000004BA	ea
_membot	00000432	ea
_memtop	00000436	ea
_nflops	000004A6	ea
_prt_cnt	000004EE	ea
_prtabt	000004F0	ea
_shell_p	000004F6	ea
_sysbase	000004F2	ea
_timr_ms	00000442	ea
_v_bas_ad	0000044E	ea
_vbclock	00000462	ea
_vbl_list	000004CE	ea
_vblqueue	00000456	ea
abort	000001C8	t
aer	FFFFFA03	ea
baseaddr	00000042	b
bigbuff	00007E2E	b
cmdreg	00000080	ea
colorptr	0000045A	ea
constate	000004A8	ea
conterm	00000484	ea
criticret	0000048A	ea
datareg	00000086	ea
ddr	FFFFFA05	ea
defshiftmd	0000044A	ea
diskctl	FFFF8604	ea
dmahi	FFFF8609	ea
dmalo	FFFF860D	ea
dmamid	FFFF860B	ea
dtr	00000010	ea
end_os	000004FA	ea
etv_critic	00000404	ea
etv_term	00000408	ea
etv_timer	00000400	ea
etv_xtra	0000040C	ea
exec_os	000004FE	ea
exit	00000172	t
fifo	FFFF8606	ea
flock	0000043E	ea
giaamp	00000008	ea
gibamp	00000009	ea
gicamp	0000000A	ea
gicrnvlp	0000000C	ea
gifienvlp	0000000B	ea
gimixer	00000007	ea
ginoise	00000006	ea
giporta	0000000E	ea
giportb	0000000F	ea
giread	FFFF8800	ea
giselect	FFFF8800	ea
gitoneac	00000001	ea
gitoneaf	00000000	ea
gitonebc	00000003	ea
gitonebf	00000002	ea
gitonecc	00000005	ea
gitonecf	00000004	ea
giwrite	FFFF8802	ea
gpip	FFFFFA01	ea
gpo	00000040	ea
handle	00000040	b
hbl	0000029E	t
hblvec	00000120	ea
hdv_boot	0000047A	ea
hdv_bpb	00000472	ea
hdv_init	0000046A	ea
hdv_mediach	0000047E	ea
hdv_rw	00000476	ea
hscroll	FFFF8265	ea
iera	FFFFFA07	ea
ierb	FFFFFA09	ea
ikbd	00000286	t
ikbdvec	00000118	ea
imra	FFFFFA13	ea
imrb	FFFFFA15	ea
ipra	FFFFFA0B	ea
iprb	FFFFFA0D	ea
isra	FFFFFA0F	ea
isrb	FFFFFA11	ea
keybd	FFFFFC02	ea
keyctl	FFFFFC00	ea
linewid	FFFF820F	ea
memcntlr	00000424	ea
memconf	FFFF8001	ea
memval2	0000043A	ea
memvalid	00000420	ea
mfp	FFFFFA00	ea
midi	FFFFFC06	ea
midictl	FFFFFC04	ea
mystack	00017C32	b
neobuff	000000AE	b
neofile	00000000	d
noexit	00000170	t
nvbls	00000454	ea
oldikbd	00000290	t
oldpal	00000008	b
oldvbl	00000280	t
palette	FFFF8240	ea
palmode	00000448	ea
phystop	0000042E	ea
prv_aux	00000512	ea
prv_auxo	0000050E	ea
prv_lst	0000050A	ea
prv_lsto	00000506	ea
resvalid	00000426	ea
resvector	0000042A	ea
rezmode	FFFF8260	ea
rsr	FFFFFA2B	ea
sav_context	000004AE	ea
save_row	000004AC	ea
savptr	000004A2	ea
scr	FFFFFA27	ea
scr_dump	00000502	ea
screenpt	0000045E	ea
secreg	00000084	ea
seekrate	00000440	ea
sshiftmd	0000044C	ea
start	00000000	t
strobe	00000020	ea
swv_vec	0000046E	ea
syncmode	FFFF820A	ea
tacr	FFFFFA19	ea
tadr	FFFFFA1F	ea
tbcr	FFFFFA1B	ea
tbdr	FFFFFA21	ea
tcdcr	FFFFFA1D	ea
tcdr	FFFFFA23	ea
tddr	FFFFFA25	ea
themd	0000048E	ea
trkreg	00000082	ea
trp14ret	00000486	ea
tsr	FFFFFA2D	ea
ucr	FFFFFA29	ea
udr	FFFFFA2F	ea
vbasehi	FFFF8201	ea
vbaselo	FFFF820D	ea
vbasemid	FFFF8203	ea
vbl	000001D6	t
vblsem	00000452	ea
vbltemp	0001782E	b
vblvec	00000070	ea
vcounthi	FFFF8205	ea
vcountlo	FFFF8209	ea
vcountmid	FFFF8207	ea
video	0000007E	b
videodata	000000AA	b
videoptr	000000A6	b
vr	FFFFFA17	ea
wavelp	0000014A	t
xoffset	0000006A	b
xrts	00000008	ea

```
                                ;
                                ;       NEOWALL.S  Horizontal and Vertical Scrolling Demo
                                ;
                                ;       Copyright 1988 ATARI CORP.
                                ;       Started 10/10/88 .. Rob Zdybel
                                ;

                                        .text
                                        .include atari
                                        .list

                                ;
                                ;       HARDWARE CONSTANTS
                                ;
          =FFFF820D             vbaselo =       $ffff820d       ; Video Base Address (lo)
          =FFFF820F             linewid =       $ffff820f       ; Width of a scan-line (Words, minus 1)
          =FFFF8265             hscroll =       $ffff8265       ; Horizontal scroll count (0 .. 15)

                                ;
                                ;       SYSTEM CONSTANTS
                                ;
          =00000070             vblvect =       $70             ; System VBlank Vector

          =FFFFFFCE             movec   =       -50             ; LineA Mouse-Motion Vector offset
          =FFFFFDA6             cur_x   =       -602            ; LineA Current mouse Xpos
          =FFFFFDA8             cur_y   =       -600            ; LineA Current mouse Ypos

                                ;
                                ;       LOCAL CONSTANTS
                                ;


                                ;
                                ;       System Initialization
                                ;
                                start:
00000000  2A4F                          move.l  a7,a5
00000002  2E7Cxxxxxxxx                  move.l  #mystack,a7     ; Get Our Own Local Stack
00000008  2A6D0004                      move.l  4(a5),a5        ; a5 = basepage address
0000000C  202D000C                      move.l  TEXTSZ(a5),d0
00000010  D0AD0014                      add.l   DATASZ(a5),d0
00000014  D0AD001C                      add.l   BSSSZ(a5),d0
00000018  D0BC00000100                  add.l   #$100,d0        ; RAM req'd = text+bss+data+BasePageLength
0000001E  2800                          move.l  d0,d4           ; d4 = RAM req'd
                                        Mshrink a5,d0           ; Return Excess Storage
00000020  2F00                  •       move.l  d0,-(sp)
00000022  2F0D                  •       move.l  a5,-(sp)
00000024  4267                  •       clr.w   -(sp)
                                •       Gemdos $4a,12
00000026  3F3C004A              •       move.w  #$4a,-(sp)
0000002A  4E41                  •       trap    #1
                                •       .if $c <= 8
                                -       addq    #$c,sp
                                -       .else
0000002C  DEFC000C              •       add.w   #$c,sp
                                •       .endif
                                 ;
                                 ;      Other Initialization
                                 ;
                                        Super                   ; enter supervisor mode
00000030  42A7                  •       clr.l   -(sp)
00000032  3F3C0020              •       move.w  #$20,-(sp)
00000036  4E41                  •       trap    #1
00000038  5C4F                  •       addq    #6,sp
0000003A  2F00                  •       move.l  d0,-(sp)        ; WARNING - Old SSP saved on stack.

                                        Fgetdta
                                •       Gemdos $2f,2
0000003C  3F3C002F              •       move.w  #$2f,-(sp)
00000040  4E41                  •       trap    #1
                                •       .if $2 <= 8
00000042  544F                  •       addq    #$2,sp
                                •       .else
                                -       add.w   #$2,sp
                                -       .endif
00000044  2840                          move.l  d0,a4
00000046  D8FC001E                      adda    #30,a4          ; a4 = Filename ptr
0000004A  7800                          moveq   #0,d4           ; d4 = Loop Count
                                        Fsfirst #neofiles,#0
0000004C  3F3C0000              •       move.w  #$0,-(sp)
00000050  2F3Cxxxxxxxx          •       move.l  #neofiles,-(sp)
                                •       Gemdos $4e,8
00000056  3F3C004E              •       move.w  #$4e,-(sp)
0000005A  4E41                  •       trap    #1
                                •       .if $8 <= 8
0000005C  504F                  •       addq    #$8,sp
                                •       .else
                                -       add.w   #$8,sp
                                -       .endif
                                 .neoloop:                      ; FOR (Nine NEO Files) DO
0000005E  4A40                          tst     d0
00000060  6B00xxxx                      bmi     abort           ; IF (No more NEO files) ABORT
```

```
                                     Fopen   a4,#0
00000064  3F3C0000          •        move.w  #$0,-(sp)
00000068  2F0C              •        move.l  a4,-(sp)
                            •        Gemdos $3d,8
0000006A  3F3C003D          •        move.w  #$3d,-(sp)
0000006E  4E41              •        trap    #1
                            •        .if $8 <= 8
00000070  504F              •        addq    #$8,sp
                            •        .else
                            -        add.w   #$8,sp
                            -        .endif
00000072  4A40                       tst     d0
00000074  6B00xxxx                   bmi     abort           ; IF (Error opening a file) ABORT
00000078  41F9xxxxxxxx               lea     handlist,a0
0000007E  31804000                   move    d0,(a0,d4)      ; Save the Handle
00000082  5444                       addq    #2,d4
00000084  B87C0010                   cmp     #16,d4
00000088  6E00xxxx                   bgt     .gotnine
                                     Fseek   #128,d0,#0      ; Skip NEO Header
0000008C  3F3C0000          •        move.w  #$0,-(sp)
00000090  3F00              •        move.w  d0,-(sp)
00000092  2F3C00000080      •        move.l  #$80,-(sp)
                            •        Gemdos $42,10
00000098  3F3C0042          •        move.w  #$42,-(sp)
0000009C  4E41              •        trap    #1
                            •        .if $a <= 8
                            -        addq    #$a,sp
                            -        .else
0000009E  DEFC000A          •        add.w   #$a,sp
                            •        .endif
000000A2  4A80                       tst.l   d0
000000A4  6B00xxxx                   bmi     abort           ; IF (File Seek Error) ABORT
                                     Fsnext
                            •        Gemdos $4f,2
000000A8  3F3C004F          •        move.w  #$4f,-(sp)
000000AC  4E41              •        trap    #1
                            •        .if $2 <= 8
000000AE  544F              •        addq    #$2,sp
                            •        .else
                            -        add.w   #$2,sp
                            -        .endif
000000B0  60AC                       bra     .neoloop
                               .gotnine:
                                     Fread   d0,#128,#bigbuff
000000B2  2F3Cxxxxxxxx      •        move.l  #bigbuff,-(sp)
000000B8  2F3C00000080      •        move.l  #$80,-(sp)
000000BE  3F00              •        move.w  d0,-(sp)
                            •        Gemdos $3f,12
000000C0  3F3C003F          •        move.w  #$3f,-(sp)
000000C4  4E41              •        trap    #1
                            •        .if $c <= 8
                            -        addq    #$c,sp
                            -        .else
000000C6  DEFC000C          •        add.w   #$c,sp
                            •        .endif
000000CA  4A80                       tst.l   d0
000000CC  6B00xxxx                   bmi     abort           ; IF (File Read Error) ABORT
000000D0  45F9xxxxxxxx               lea     bigbuff+4,a2
000000D6  41F88240                   lea     palette,a0
000000DA  43F9xxxxxxxx               lea     oldpal,a1
000000E0  303C000F                   move    #15,d0
000000E4  32D0                 .ploop: move.w  (a0),(a1)+      ; save old color palette
000000E6  30DA                       move.w  (a2)+,(a0)+     ; create new color palette
000000E8  51C8FFFA                   dbra    d0,.ploop

000000EC  23FCxxxxxxxxxxxxxxxx       move.l  #bigbuff,buffptr
000000F6  7E00                       moveq   #0,d7           ; d7 = Row Count
000000F8  49F9xxxxxxxx         .rowlp: lea     threebuf,a4     ; FOR (Three rows) DO
000000FE  4BF9xxxxxxxx               lea     handlist,a5
00000104  DAC7                       adda    d7,a5
00000106  3C3C0002                   move    #2,d6           ; d5 = Column Count
                               .redlp: Fread   (a5)+,#32000,a4 ; FOR (3 Files) DO Read into temp buff
0000010A  2F0C              •        move.l  a4,-(sp)
0000010C  2F3C00007D00      •        move.l  #$7d00,-(sp)
00000112  3F1D              •        move.w  (a5)+,-(sp)
                            •        Gemdos $3f,12
00000114  3F3C003F          •        move.w  #$3f,-(sp)
00000118  4E41              •        trap    #1
                            •        .if $c <= 8
                            -        addq    #$c,sp
                            -        .else
0000011A  DEFC000C          •        add.w   #$c,sp
                            •        .endif
0000011E  4A80                       tst.l   d0
00000120  6B00xxxx                   bmi     abort           ; IF (File Read Error) ABORT
00000124  D8FC7D00                   adda    #32000,a4
00000128  51CEFFE0                   dbra    d6,.redlp

0000012C  43F9xxxxxxxx               lea     threebuf,a1
00000132  45F9xxxxxxxx               lea     threebuf+32000,a2
00000138  47F9xxxxxxxx               lea     threebuf+64000,a3
```

```
00000013E  2079xxxxxxxx              move.l  buffptr,a0
000000144  3C3C00C7                  move    #199,d6         ; d6 = Scan Line Count
000000148  3A3C0027         .linlp:  move    #39,d5          ; FOR (200 Lines) DO
00000014C  20D9             .t1:     move.l  (a1)+,(a0)+     ; Copy a line from screen0
00000014E  51CDFFFC                  dbra    d5,.t1
000000152  3A3C0027                  move    #39,d5
000000156  20DA             .t2:     move.l  (a2)+,(a0)+     ; Copy a line from screen1
000000158  51CDFFFC                  dbra    d5,.t2
00000015C  3A3C0027                  move    #39,d5
000000160  20DB             .t3:     move.l  (a3)+,(a0)+     ; Copy a line from screen2
000000162  51CDFFFC                  dbra    d5,.t3
000000166  51CEFFE0                  dbra    d6,.linlp
00000016A  23C8xxxxxxxx              move.l  a0,buffptr
000000170  5C47                      addq    #6,d7
000000172  BE7C000C                  cmp     #12,d7
000000176  6F88                      ble     .rowlp

000000178  7810                      moveq   #16,d4
00000017A  49F9xxxxxxxx              lea     handlist,a4
000000180  3F344000         .close:  move    (a4,d4),-(sp)   ; FOR (Nine files) DO Close all
                                     Gemdos  $3e,4           ; Fclose
000000184  3F3C003E       •          move.w  #$3e,-(sp)
000000188  4E41           •          trap    #1
                          •          .if $4 <= 8
00000018A  584F           •          addq    #$4,sp
                          •          .else
                          -          add.w   #$4,sp
                          -          .endif
00000018C  4A40                      tst     d0
00000018E  6B00xxxx                  bmi     abort           ; IF (Error Closing a file) ABORT
000000192  5544                      subq    #2,d4
000000194  6AEA                      bpl     .close

000000196  4EB9xxxxxxxx              jsr     initmaus        ; Install our own mouse handler

00000019C  23F80070xxxxxxxx          move.l  vblvect,oldvbl
0000001A4  21FCxxxxxxxx0000          move.l  #vbl,vblvect    ; Capture System VBlank Interrupt

                            ;
                            ;        Scrolling Demo loop
                            ;
                            wavelp:
                                     Bconstat CON            ; Keyboard Polling
0000001AC  3F3C0002       •          move.w  #CON,-(sp)
                          •          Bios 1,4
0000001B0  3F3C0001       •          move.w  #$1,-(sp)
0000001B4  4E4D           •          trap    #13
                          •          .if $4 <= 8
0000001B6  584F           •          addq    #$4,sp
                          •          .else
                          -          add.w   #$4,sp
                          -          .endif
0000001B8  4A40                      tst     d0
0000001BA  6700xxxx                  beq     noexit          ; IF (Keyboard Input Available) THEN
                                     Bconin  CON
0000001BE  3F3C0002       •          move.w  #CON,-(sp)
                          •          Bios 2,4
0000001C2  3F3C0002       •          move.w  #$2,-(sp)
0000001C6  4E4D           •          trap    #13
                          •          .if $4 <= 8
0000001C8  584F           •          addq    #$4,sp
                          •          .else
                          -          add.w   #$4,sp
                          -          .endif
0000001CA  0C3C0003                  cmp.b   #'C'-64,d0
0000001CE  6700xxxx                  beq     exit            ; CTRL-C ==> EXIT
                            noexit:
0000001D2  60D8                      bra     wavelp
                            exit:
                            ;
                            ;        System Tear-Down
                            ;
0000001D4  21F9xxxxxxxx0000          move.l  oldvbl,vblvect  ; Restore System VBlank Interrupt

0000001DC  4EB9xxxxxxxx              jsr     unmaus          ; Restore System mouse handler

0000001E2  41F9xxxxxxxx              lea     oldpal,a0
0000001E8  43F88240                  lea     palette,a1
0000001EC  303C000F                  move    #15,d0
0000001F0  32D8             .unplp:  move.w  (a0)+,(a1)+
0000001F2  51C8FFFC                  dbra    d0,.unplp       ; restore old color palette

                            abort:   User                    ; return to user mode
                          •          Gemdos  $20,6
0000001F6  3F3C0020       •          move.w  #$20,-(sp)
0000001FA  4E41           •          trap    #1
                          •          .if $6 <= 8
0000001FC  5C4F           •          addq    #$6,sp
                          •          .else
                          -          add.w   #$6,sp
                          -          .endif
```

```
                                          Pterm0                  ; return to GEMDOS
    000001FE  4267               *        clr.w   -(sp)
    00000200  4E41               *        trap    #1
    00000202  4AFC               *        illegal

                                 ;
                                 ;        VBL     Vertical-Blank Interrupt Server
                                 ;
                                 vbl:
    00000204  48E78080                    movem.l d0/a0,-(sp)

    00000208  3039xxxxxxxx                move    xmouse,d0
    0000020E  C07C000F                    and     #$0f,d0
    00000212  11C08265                    move.b  d0,hscroll      ; Xpos MOD 16 = Scroll count
    00000216  4A00                        tst.b   d0
    00000218  6600xxxx                    bne     .non0           ; IF (Scrolling) THEN 4 word offset
    0000021C  11FC00A0820F                move.b  #160,linewid
    00000222  6000xxxx                    bra     .join
    00000226  11FC009C820F       .non0:   move.b  #156,linewid
                                 .join:
    0000022C  41F9xxxxxxxx                lea     bigbuff,a0
    00000232  3039xxxxxxxx                move    ymouse,d0
    00000238  C0FC01E0                    mulu    #3*160,d0       ; Ypos * Linewid = Vertical offset
    0000023C  D1C0                        adda.l  d0,a0
    0000023E  3039xxxxxxxx                move    xmouse,d0
    00000244  E240                        asr     #1,d0
    00000246  C07CFFF8                    and     #$fff8,d0       ; 8*(Xpos DIV 16) = Line offset
    0000024A  D0C0                        adda    d0,a0           ; a0 = Video Base Address
    0000024C  23C8xxxxxxxx                move.l  a0,vbltemp
    00000252  11F9xxxxxxxx0000            move.b  vbltemp+1,vcounthi
    0000025A  11F9xxxxxxxx0000            move.b  vbltemp+2,vcountmid
    00000262  11F9xxxxxxxx0000            move.b  vbltemp+3,vcountlo

    0000026A  4CDF0101                    movem.l (sp)+,d0/a0
    0000026E  4EF9                        .dc.w   $4ef9
    00000270  00000000           oldvbl:  .dc.l   0               ; JMP (Old-Vblank)
    00000274  4AFC                        illegal


                                 ;
                                 ;        MOUSE HANDLING
                                 ;


                                 ;
                                 ;        INITMAUS   Capture system mouse
                                 ;
                                 ;        Given:
                                 ;                Control
                                 ;
                                 ;        Returns:
                                 ;                With motion and button vectors captured
                                 ;
                                 ;        Register Usage:
                                 ;                destroys d0-d3 and a0-a3
                                 ;
                                 ;        Externals:
                                 ;                none
                                 ;
                                 initmaus:
    00000276  A000                        .dc.w   $a000           ; Line-A Trap
    00000278  33E8FDA6xxxxxxxx            move    cur_x(a0),xmouse
    00000280  33E8FDA8xxxxxxxx            move    cur_y(a0),ymouse
    00000288  23E8FFCExxxxxxxx            move.l  movec(a0),moldvec
    00000290  217Cxxxxxxxx0000            move.l  #ourmaus,movec(a0)      ; Take over mouse motion
    00000298  4E75                        rts
                                 ;
                                 ;        Mouse Motion Interrupt
                                 ;
                                 ourmaus:
    0000029A  33C0xxxxxxxx                move    d0,xmouse
    000002A0  33C1xxxxxxxx                move    d1,ymouse       ; Save new mouse position
    000002A6  4EF9                        .dc.w   $4ef9
                                 moldvec:
    000002A8  00000000                    .dc.l   0               ; JMP (Old motion vector)
    000002AC  4AFC                        illegal
                                 ;
                                 ;        UNMAUS   Restore mouse to system
                                 ;
                                 ;        Given:
                                 ;                Control
                                 ;
                                 ;        Returns:
                                 ;                Mouse and button vectors restored to system
                                 ;
                                 ;        Register Usage:
                                 ;                destroys d0-d3 and a0-a3
                                 ;
                                 ;        Externals:
                                 ;                none
                                 ;
                                 unmaus:
```

```
000002AE  A000                            .dc.w   $a000           ; Line-A Trap
000002B0  2179000002A8FFCE                move.l  moldvec,movec(a0)       ; Restore mouse motion
000002B8  4E75                            rts


                                  ;
                                  ;       DATA STORAGE
                                  ;
000002BA                                  .data
                                  neofiles:                       ; NEO filename search string
00000000  2A2E6E656F00                    .dc.b   "*.neo",0

                                          .even

                                  ;
                                  ;       RANDOM DATA STORAGE
                                  ;
00000006                                  .bss

                                  oldpal:
00000000 =00000010                        .ds.l   16              ; Original color palette

                                  handlist:                       ; Array of Active Handles (9)
00000040 =00000009                        .ds.w   9
                                  buffptr:                        ; Load ptr for bigbuff
00000052 =00000001                        .ds.l   1
                                  bigbuff:                        ; Mega-Image Buffer
00000056 =00046500                        .ds.b   9*32000
                                  threebuf:                       ; Temporary Triple-Image Buffer
00046556 =00017700                        .ds.b   3*32000

                                  vbltemp:                        ; Vblank Temporary Storage
0005DC56 =00000001                        .ds.l   1

                                  xmouse:                         ; Latest mouse Xposn
0005DC5A =00000001                        .ds.w   1
                                  ymouse:                         ; Latest mouse Yposn
0005DC5C =00000001                        .ds.w   1

0005DC5E =00000100                        .ds.l   256             ; (stack body)
                                  mystack:
0005E05E =00000001                        .ds.l   1               ; Local Stack Storage

                                          .end
```

Symbol Table

```
       .close 00000188  t
     .gotnine 00000062  t
        .join 0000022C  t
       .linlp 00000148  t
     .neoloop 0000005E  t
        .non0 00000226  t
       .ploop 000000E4  t
       .redlp 0000010A  t
       .rowlp 000000F8  t
          .t1 0000014C  t
          .t2 00000156  t
          .t3 00000160  t
       .unplp 000001F0  t
          AUX 00000001 ea
        BBASE 00000018  a
         BLEN 0000001C  a
         BPSZ 00000100 ea
        BSIZE 0000000A  a
        BSSSZ 0000001C ea
      CMDLINE 00000080  a
          CON 00000002 ea
           CR 0000000D ea
   CURS_BLINK 00000002 ea
 CURS_GETRATE 00000005 ea
    CURS_HIDE 00000000 ea
 CURS_NOBLINK 00000003 ea
 CURS_SETRATE 00000004 ea
    CURS_SHOW 00000001 ea
       DATASZ 00000014 ea
        DBASE 00000010  a
         DLEN 00000014  a
        DSIZE 00000006  a
          DTA 00000020  a
        ENVIR 0000002C  a
      FILE_ID 00000000  a
     HEADSIZE 0000001C ea
        HITPA 00000004  a
         IKBD 00000004 ea
           LF 0000000A ea
       LOWTPA 00000000  a
         MIDI 00000003 ea
        MYDTA 00000020 ea
       PARENT 00000024  a
          PRT 00000000 ea
       RAWCON 00000005 ea
        SSIZE 0000000E  a
          TAB 00000009 ea
        TBASE 00000008  a
       TEXTSZ 0000000C ea
         TLEN 0000000C  a
        TSIZE 00000002  a
         XXX1 00000012  a
         XXX2 00000016  a
         XXX3 0000001A  a
         XXXX 00000028  a
     ______md 0000049E ea
    _autopath 000004CA ea
     _bootdev 00000446 ea
        _bufl 000004B2 ea
     _cmdload 00000482 ea
     _drvbits 000004C2 ea
     _dskbufp 000004C6 ea
     _frclock 00000466 ea
     _fverify 00000444 ea
      _hz_200 000004BA ea
      _membot 00000432 ea
      _memtop 00000436 ea
      _nflops 000004A6 ea
     _prt_cnt 000004EE ea
      _prtabt 000004F0 ea
     _shell_p 000004F6 ea
     _sysbase 000004F2 ea
     _timr_ms 00000442 ea
    _v_bas_ad 0000044E ea
     _vbclock 00000462 ea
    _vbl_list 000004CE ea
    _vblqueue 00000456 ea
        abort 000001F6  t
          aer FFFFFA03 ea
      bigbuff 00000056  b
      buffptr 00000052  b
       cmdreg 00000080 ea
     colorptr 0000045A ea
     constate 000004A8 ea
      conterm 00000484 ea
    criticret 0000048A ea
        cur_x FFFFFDA6 ea
        cur_y FFFFFDA8 ea
      datareg 00000086 ea
          ddr FFFFFA05 ea
   defshiftmd 0000044A ea

      diskctl FFFF8604 ea
        dmahi FFFF8609 ea
        dmalo FFFF860D ea
       dmamid FFFF860B ea
          dtr 00000010 ea
       end_os 000004FA ea
   etv_critic 00000404 ea
     etv_term 00000408 ea
    etv_timer 00000400 ea
     etv_xtra 0000040C ea
      exec_os 000004FE ea
         exit 000001D4  t
         fifo FFFF8606 ea
        flock 0000043E ea
       giaamp 00000008 ea
       gibamp 00000009 ea
       gicamp 0000000A ea
     gicrnvlp 0000000C ea
     gifienvlp 0000000B ea
      gimixer 00000007 ea
      ginoise 00000006 ea
      giporta 0000000E ea
      giportb 0000000F ea
       giread FFFF8800 ea
     giselect FFFF8800 ea
     gitoneac 00000001 ea
     gitoneaf 00000000 ea
     gitonebc 00000003 ea
     gitonebf 00000002 ea
     gitonecc 00000005 ea
     gitonecf 00000004 ea
      giwrite FFFF8802 ea
         gpip FFFFFA01 ea
          gpo 00000040 ea
     handlist 00000040  b
     hdv_boot 0000047A ea
      hdv_bpb 00000472 ea
     hdv_init 0000046A ea
  hdv_mediach 0000047E ea
       hdv_rw 00000476 ea
      hscroll FFFF8265 ea
         iera FFFFFA07 ea
         ierb FFFFFA09 ea
         imra FFFFFA13 ea
         imrb FFFFFA15 ea
     initmaus 00000276  t
         ipra FFFFFA0B ea
         iprb FFFFFA0D ea
         isra FFFFFA0F ea
         isrb FFFFFA11 ea
        keybd FFFFFC02 ea
       keyctl FFFFFC00 ea
      linewid FFFF820F ea
     memcntlr 00000424 ea
      memconf FFFF8001 ea
      memval2 0000043A ea
     memvalid 00000420 ea
          mfp FFFFFA00 ea
         midi FFFFFC06 ea
      midictl FFFFFC04 ea
      moldvec 000002A8  t
        movec FFFFFFCE ea
      mystack 0005E05E  b
     neofiles 00000000  d
       noexit 000001D2  t
        nvbls 00000454 ea
       oldpal 00000000  b
       oldvbl 00000270  t
     ourmaus 0000029A  t
      palette FFFF8240 ea
      palmode 00000448 ea
      phystop 0000042E ea
      prv_aux 00000512 ea
     prv_auxo 0000050E ea
      prv_lst 0000050A ea
     prv_lsto 00000506 ea
     resvalid 00000426 ea
    resvector 0000042A ea
      rezmode FFFF8260 ea
          rsr FFFFFA2B ea
  sav_context 000004AE ea
     save_row 000004AC ea
       savptr 000004A2 ea
          scr FFFFFA27 ea
     scr_dump 00000502 ea
     screenpt 0000045E ea
       secreg 00000084 ea
    seekrate 00000440 ea
     sshiftmd 0000044C ea
        start 00000000  t
       strobe 00000020 ea

      swv_vec 0000046E ea
     syncmode FFFF820A ea
         tacr FFFFFA19 ea
         tadr FFFFFA1F ea
         tbcr FFFFFA1B ea
         tbdr FFFFFA21 ea
        tcdcr FFFFFA1D ea
         tcdr FFFFFA23 ea
         tddr FFFFFA25 ea
        themd 0000048E ea
    threebuf 00046556  b
       trkreg 00000082 ea
     trp14ret 00000486 ea
          tsr FFFFFA2D ea
          ucr FFFFFA29 ea
          udr FFFFFA2F ea
       unmaus 000002AE  t
      vbasehi FFFF8201 ea
      vbaselo FFFF820D ea
     vbasemid FFFF8203 ea
          vbl 00000204  t
       vblsem 00000452 ea
      vbltemp 0005DC56  b
      vblvect 00000070 ea
     vcounthi FFFF8205 ea
     vcountlo FFFF8209 ea
    vcountmid FFFF8207 ea
           vr FFFFFA17 ea
       wavelp 000001AC  t
       xmouse 0005DC5A  b
         xrts 00000008 ea
       ymouse 0005DC5C  b
```

STE Digitized Sound
Developer information

The Atari STE™ family of computers is equipped to reproduce digitized sound using DMA (direct memory access; that is, without using the 68000). This document provides the information required to understand and use this feature.

OVERVIEW

Sound is stored in memory as digitized samples. Each sample is a number, from -128 to +127, which represents displacement of the speaker from the "neutral" or middle position. During horizontal blanking (transparent to the processor) the DMA sound chip fetches samples from memory and provides them to a digital-to-analog converter (DAC) at one of several constant rates, programmable as (approximately) 50KHz (kilohertz), 25KHz, 12.5KHz, and 6.25KHz. This rate is called the sample frequency.

The output of the DAC is then filtered to a frequency equal to 40% of the sample frequency by a four-pole switched low-pass filter. This performs "anti-aliasing" of the sound data in a sample-frequency-sensitive way. The signal is further filtered by a two-pole fixed frequency (16kHz) low-pass filter and provided to a National LMC1992 Volume/Tone Controller. Finally, the output is available at an RCA-style output jack on the back of the computer. This can be fed into an amplifier, and then to speakers, headphones, or tape recorders.

There are two channels which behave as described above; they are intended to be used as the left and right channels of a stereo system when using the audio outputs of the machine. A monophonic mode is provided which will send the same sample data to each channel.

The stereo sound output is also mixed onto the standard ST audio output sent to the monitor's speaker. The ST's GI sound chip output can be mixed to the monitor and to both stereo output jacks as well.

DATA FORMAT

Each sample is stored as a signed eight-bit quantity, where -128 (80 hex) means full negative displacement of the speaker, and 127 (7F hex) means full positive displacement. In stereo mode, each word represents two samples: the upper byte is the sample for the left channel, and the lower byte is the sample for the right channel. In mono mode each byte is one sample. However, the samples are always fetched a word at a time, so only an even number of mono samples can be played.

A group of samples is called a "frame." A frame may be played once or can automatically be repeated forever (until stopped). A frame is described by its start and end addresses. The end address of a frame is actually the address of the first byte in memory *beyond* the frame; a frame starting at address 21100 which is 10 bytes long has an end address of 21110.

Before continuing, please familiarize yourself with the DMA sound chip register set:

REGISTER DESCRIPTIONS

```
FF8900 ---- ---- ---- --cc RW Sound DMA Control
    cc:
    00  Sound DMA disabled (reset state).
    01  Sound DMA enabled, disable at end of frame.
    11  Sound DMA enabled, repeat frame forever.

FF8902 ---- ---- 00xx xxxx RW Frame Base Address (high)
FF8904 ---- ---- xxxx xxxx RW Frame Base Address (middle)
FF8906 ---- ---- xxxx xxx0 RW Frame Base Address (low)

FF8908 ---- ---- 00xx xxxx RO Frame Address Counter (high)
FF890A ---- ---- xxxx xxxx RO Frame Address Counter (middle)
FF890C ---- ---- xxxx xxx0 RO Frame Address Counter (low)

FF890E ---- ---- 00xx xxxx RW Frame End Address (high)
FF8910 ---- ---- xxxx xxxx RW Frame End Address (middle)
FF8912 ---- ---- xxxx xxx0 RW Frame End Address (low)

FF8920 0000 0000 m000 00rr RW Sound Mode Control
    rr:
    00   6258 Hz sample rate (reset state)
    01  12517 Hz sample rate
    10  25033 Hz sample rate
    11  50066 Hz sample rate
    m:
    0    Stereo Mode (reset state)
    1    Mono Mode

FF8922 xxxx xxxx xxxx xxxx RW MICROWIRE™ Data register
FF8924 xxxx xxxx xxxx xxxx RW MICROWIRE™ Mask register
```

Note: a zero can be written to the DMA sound control register at any time to stop playback immediately.

The frame address registers occupy the low bytes of three consecutive words each. The high bytes of these words do not contain anything useful, and it is harmless to read or write them. The frame address counter register is read-only, and holds the address of the next sample word to be fetched.

PROGRAMMING CONSIDERATIONS

The simplest way to produce a sound is to assemble a frame in memory, write the start address of the frame into the Frame Start Address register, and the end address of the frame into the Frame End Address register, set the Mode register appropriately (set stereo or mono, and the sample frequency), and write a one into the Sound DMA Control register. The frame will play once, then stop.

To produce continuous sound, and link frames together, more elaborate techniques are required.

The DMA sound chip produces a signal called "DMA sound active" which is one when the chip is playing sounds, and zero when it's not. When a frame ends in the repeat mode (mode 3), there is a transition from "active" to "idle" and back again on this signal. The signal is presented as the external input to MFP Timer A. You can put Timer A into Event Count mode and use it to generate an interrupt, for example when a frame has played a given number of times. Because of the design of the MFP, the active edge for this signal must be the same as the input on GPIP I4, which is the interrupt line from the keyboard and MIDI interfaces. It is, and the Active Edge Register is already programmed for that, so you need not worry about that if you use Timer A to count frames.

The DMA Sound chip's mode 3 (repeat mode) ensures seamless linkage of frames, because the start and end registers are actually double-buffered. When you write to these registers, what you write really goes into a "holding area". The contents of the holding area go into the true registers at the end of the current frame. (Actually, they go in when the chip is idle, which means right away if the chip was idle to begin with.)

If you have two frames which you want played in succession, you can write the start and end addresses of the first frame into the chip, then set its control register to 3. The first frame will begin playing. You can then immediately write the start and end addresses of the second frame into the chip: they will be held in the holding area until the first frame finishes, then they'll be copied into the true registers and the second frame will play. The interrupt between frames will still happen, so you can tell when the first frame has finished. Then, for instance, you can write the start and end registers for the start of a *third* frame, knowing that it will begin as soon as the second frame has finished. You could even write new data into the first frame and write its start and end address into the chip; this kind of ping-pong effect is rather like double-buffering of a graphics display.

Here is an example of using Timer A in Event Count mode to play a controlled series of frames. Suppose you have three frames, A, B, and C, and you want to play frame A three times, then frame B five times, and finally frame C twice. The sequence of steps below will accomplish this. Numbered steps are carried out by your program; the bracketed descriptions are of things which are happening as a result.

1. Set Timer A to event count mode, and its counter to 2 (not 3).

2. Write Frame A's start & end addresses into the registers.

3. Write a 3 to the sound DMA control register. [Play begins.] Go do something else until interrupted.

   [At the end of the second repetition of Frame A, the timer's interrupt fires. At the same time, frame A begins its third repetition.]

4. Write Frame B's start and end addresses into the DMA sound chip. These values will be held until the third repetition of Frame A finishes.

5. Set Timer A's count register to 5, then go away until interrupted

   [When the current repetition finishes, the start & end registers are loaded from the holding area, and Frame B will begin playing. The end-of-frame signal will cause Timer A to count from 5 to 4. At the end of Frame B's fourth repetition, its fifth will start, the timer will count down from 1 to 0, and the interrupt will occur.]

6. Write frame C's start & end addresses into the registers, and program Timer A to count to 2. Go away until interrupted.

   (When the current repetition (B's fifth) finishes, the start & end registers are loaded from the holding area, and Frame C will begin playing. The end-of-frame signal causes Timer A to count down from 2 to 1. When Frame C finishes its first repetition, Timer A counts down from 1 to 0 and interrupts.)

7. Write a 1 to the DMA Sound Control Register to play the current frame, then stop. Disable Timer A and mask its interrupt. You're done.

As you can see, you program the timer to interrupt after one repetition *less* than the number of times you want a frame to play. That is so you can set up the next frame while the DMA sound chip is playing the last repetition of the current frame. This ensures seamless linkage of frames.

INTERRUPTS WITHOUT TIMER A

Besides going to the external input signal of Timer A, the DMA-sound-active signal, true high, is exclusive-ORed with the monochrome-detect signal, and together they form the GPIP I7 input to the M68901 MFP. The intent of this is to provide for interrupt-driven sound drivers without using up the last general-purpose timer in the MFP. It is a little trickier to use, however. For one thing, it causes the interrupt at the end of every frame, not after a specified number of frames. For another, the "interesting" edge on this signal depends on what kind of monitor you have.

On an ST, monochrome monitors ground the mono-detect signal, so when you read the bit in the MFP you get a zero. Color monitors do not ground it, so it reads as a one. When the DMA sound is idle (0), this is still the case. However, when the sound is active (1), the mono-detect signal is inverted by the XOR, so the bit in the MFP reads the opposite way. (The one place where the OS reads this bit is at VBLANK time, to see if you've changed monitors. The ROMs on any machine with DMA sound are appropriately modified, so you need not worry about this.)

If you want to use the mono-detect / DMA interrupt signal, you have to set up the active-edge register in the MFP to cause the interrupt at the right time. The interesting edge on the DMA signal is the falling edge, that is, from active to idle; this happens when a frame finishes. If you have a monochrome monitor, this edge is seen as a transition from 1 to 0 on MFP bit I7. However, with a color monitor, the edge will be seen as a transition from 0 to 1. Therefore, you have to program the MFP's active-edge register differently depending on which monitor you have. Make sure the DMA sound is idle (write a zero to the control register), then check MFP I7: if it's one, you have a color monitor, and you need to see the rising edge. If it's zero, you have a monochrome monitor and you need to see the falling edge.

The DMA sound active signal goes from "active" to "idle" when a frame finishes. If it was playing in mode 1, it stays "idle" and the control register reverts to zero. If it was playing in mode 3, the signal goes back to "active" as the next frame begins. In this case, the signal is actually in the "idle" state for a very short time, but the MFP catches it and causes the interrupt, so don't worry.

Additional Considerations

Regardless of how you manage your interrupts, there is more you should know: the signal goes from "active" to "idle" when the DMA sound chip has *fetched* the last sample in the frame. There is a four-word FIFO in the chip, however, so it will be eight sample-times (four in stereo mode) before the sound actually finishes. If you are using mode 1, you can use this time to set up the chip with the start and end addresses of the next frame, so it will start as soon as the current one ends. However, if the interrupt should be postponed for four or eight sample-times, you could miss your chance to start the sound seamlessly. Therefore, for seamless linkage, use the pre-loading technique described above.

MICROWIRE™ Interface

The MICROWIRE™ interface provided to talk to the National LMC1992 Computer Controlled Volume / Tone Control is a general purpose MICROWIRE™ interface to allow the future addition of other MICROWIRE™ devices. For this reason, the following description of its use will make no assumptions about the device being addressed.

The MICROWIRE™ bus is a three wire serial connection and protocol designed to allow multiple devices to be individually addressed by the controller. The length of the serial data stream depends on the destination device. In general, the stream consists of N bits of address, followed by zero or more don't care bits, followed by M bits of data. The hardware interface provided consists of two 16 bit read/write registers: one data register which contains the actual bit stream to be shifted out, and one mask register which indicates which bits are valid.

Let's consider a mythical device which requires two address bits and one data bit. For this device the total bit stream is three bits (minimum). Any three bits of the register pair may be used. However, since the most significant bit is shifted first, the command will be received by the device soonest if the three most significant bits are used. Let's assume: 01 is the device's address, D is the data to be written, and X's are don't cares. Then all of the following register combinations will provide the same information to the device.

```
1110 0000 0000 0000 Mask
01DX XXXX XXXX XXXX Data

0000 0000 0000 0111 Mask
XXXX XXXX XXXX X01D Data

0000 0001 1100 0000 Mask
XXXX XXX0 1DXX XXXX Data

0000 1100 0001 0000 Mask
XXXX 01XX XXXD XXXX Data

1100 0000 0000 0001 Mask
01XX XXXX XXXX XXXD Data
```

As you can see, the address bits must be contiguous, and so must the data bits, but they don't have to be contiguous with each other.

The mask register must be written before the data register. Sending commences when the data register is written and takes approximately 16μsec. Subsequent writes to the data and mask registers are blocked until sending is complete. Reading the registers while sending is in progress will return a snapshot of the shift register shifting the data and mask out. This means that you know it is safe to send the next command when these registers (or either one) return to their original state. Note that the mask register does not need to be rewritten if it is already correct. That is, when sending a series of commands the mask register only needs to be written once.

Volume and Tone Control

The LMC1992 is used to provide volume and tone control. Before you go and find a data sheet for this part, be warned that we do not use all of its features. Commands for the features we do use are listed below.

Communication with this device is achieved using the MICROWIRE™ interface. See MICROWIRE INTERFACE the section for details. The device has a two bit address field, address = 10, and a nine bit data field. There is no way to reading the current settings.

Volume / Tone Controller Commands

Device address = 10

Data Field

```
011 DDD DDD Set Master Volume
    000 000 -80 dB
    010 100 -40 dB
    101 XXX  0 dB

101 XDD DDD Set Left Channel Volume
     00 000 -40 dB
     01 010 -20 dB
     10 1XX  0 dB

100 XDD DDD Set Right Channel Volume
     00 000 -40 dB
     01 010 -20 dB
     10 1XX  0 dB

010 XXD DDD Set Treble
      0 000 -12 dB
      0 110  0 dB (Flat)
      1 100 +12 dB

001 XXD DDD Set Bass
      0 000 -12 dB
      0 110  0 dB (Flat)
      1 100 +12 dB

000 XXX XDD Set Mix
         00 -12 dB
         01 Mix GI sound chip output
         10 Do not mix GI sound chip output
         11 reserved
```

Note: The volume controls attenuate in 2 dB steps. The tone controls attenuate in 2 dB steps at 50 Hz and 15 kHz (Note: These frequencies may change).

Using the MICROWIRE™ Interface and the Volume/Tone Control Chip

The MICROWIRE™ interface is not hard to use: once you get it right, you'll never have to figure it out again.

The easiest way to use it is to ignore the flexibility, and just use one form for all commands. Since the Volume/Tone chip is the only device, and it has a total of 11 bits of address and data, your mask should be $07ff. If you're picky, you can use $ffe0, because the high-order bits are shifted out first, but it adds conceptual complexity. With a mask of $07ff, the lower 9 bits of the data regsiter are used for the data, and the next higher two bits are for the address:

```
Mask:     %0000 0111 1111 1111
Data:     %xxxx x10d dddd dddd
```

Replace the d's with the command code and its data. For example, this combination sets the master volume to $14:

```
Mask:     %0000 0111 1111 1111
Data:     %xxxx x100 1101 0100
```

The other important concept you must understand is that the bits shift out of these registers as soon as you write the data, and it takes an appreciable time (16 μsec) to finish. You can't attempt another write until the first one is finished. If you read either register while it's being shifted out, you will see a "snapshot" of the data being shifted. You know the shifting is complete when the mask returns to its original value. (This theory is wrong if you use a mask which equals its original value sometime during the shifting, but $07ff never does.)

Assuming you write $07ff into the mask register ahead of time, the following routine can be used to write new data from the D0 register to the volume/tone control chip:

```
MWMASK    equ  $ffff8924
MWDATA    equ  $ffff8922

mwwrite:
     cmp.w     #$07ff,MWMASK  ; wait for prev to finish
     bne.s     mwwrite        ; loop until equal
     move.w    d0,MWDATA      ; write the data
     rts                      ; and return
```

The purpose of the loop at the beginning is to wait until a previous write completes. This loop is at the beginning of the routine, not the end, because waiting at the end would always force at 16 μsec delay, even if it's been longer than that since the last write.