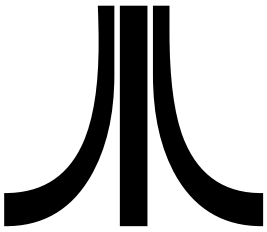

ATARI STE

HARDWARE DOCUMENTATION

Taken from the developer information, and altered to correspond more to observed facts than the original documentation.

Converted to PDF format by LST of LaserForce
On January 25th, 2001
V1.02

CONTENTS

1. INTRODUCTION TO THE STE

The Atari STe is an enhanced version of the ST with the following changes:

1.1. GRAPHICS

The colour palette has been expanded to 4096 colours from 512. Support for Vertical and Horizontal scrolling has been added to the hardware.

1.2. SOUND

Stereo 8-bit DMA sound has been added to the system along with volume and tone control.

1.3. CONTROLLERS

New controllers ports have been added to allow keypad type joysticks, analog paddies and a light pen.

1.4. GENLOCK AND THE STE

The ST (and STe) chipset have the ability to accept external sync. This is controlled by bit 0 at $F820A, 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 synchonized in some other way) to the imput 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 ST).

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 direct). The internal frequency of this clock is 32.215905 MHz (NTSC) and 31.922046 MHz (PAL).

Note: **DO NOT SWITCH CLOCK SOURCE WHILE THE SYSTEM IS ACTIVE.**
(As a result of this GPO is no longer available).

1.5. JOYSTICKS

Four new joystick ports are added. These ports are controlled directly by the Motorola 68000. The current state may be sampled at any time by reading the 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.

1.6. 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 (JOY 0 Left and Right).

1.7. 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 320*200 Mode (ST-Low)
- 8 Pixels in 640*200 Mode (ST-Medium)
- 16 Pixels in 640*400 Mode (ST-High)

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 320*200 only. In order to get correct results in 640*200 mode this number needs to be shifted left one bit and in 640*400 mode, this number needs to be shifted left two bits.

2. HOW TO IMPLEMENT FINE SCROLLING ON THE STE.

2.1. NEW REGISTERS

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

2.1.1. 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 data word(S) of a given line by this register.

2.1.2. LINEWID

This register indicates the number of extra words of data (beyond that required by an ordinary ST at the same resolution) which represents 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).

2.1.3. VBASELO

This register contains the low-order byte of the video display base address. It can be altered at any time and will not affect the display until the next vertical blank interrupt.

2.2. MODIFIED REGISTERS

The following registers have been changed on the STE to allow greater control over the display :

2.2.1. 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.

2.2.2. COLOUR PALETTE

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

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

2.3. VIDEO EFFECTS

2.3.1. 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.

2.3.2. 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 to memory. Naturally, additional data must be available to be displayed. We might imagine this as a tall, skinny screen which we are opening onto for the user. The video display address registers define where this window will start.

2.3.3. 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 incements. 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 is 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 now 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.

2.3.4. 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 other. 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 LINWID, 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 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 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.

3. STE DIGITIZED SOUND OVERVIEW

3.1. INTRODUCTION TO DIGITAL SOUND

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 possition. 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), 25 Khz, 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 program "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.

3.2. 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 some 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.

Note: A zero can be written to the DMA sound control register at anytime to stop playback immediately.

3.3. 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 to 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 a 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 14, 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 resisters, 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 address 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 the 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 and 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 sound 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 and end registers are loaded fron the holding area, and Frame B will begin playing. The end of play 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 and 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 and 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 TimerA and mask its interrupt.

8) 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.

3.4. 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 17 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, monochrme monitors ground the mono-detect signal, so when you read the bit in the MFP you get a zero. Colour 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 as at VBLANK time, to see if you've changed monitors. The ROMs on any machine with DMA sound are approprietely 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 17.

However, with a colour monitor, the edge will be seen as a transition from 0 to 1. Therefore, you have to program the MFP's active-edge regisrer differently depending on which monitor you have. Make sure the DMA sound is idle (write a zero to the control register), then check MFP 17: if it's one, you have a colour 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 one, 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.

3.5. 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.

3.6. THE MICROWIRE INTERFACE

3.6.1. INTRODUCTION

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 addresses by the controller. The lengh 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 the three most significant bits are used. Let's assume: 01 is the devise'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 contigious, 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 16 µsec. 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 thier 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.

3.6.2. 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 (MW) interface. See MICROWIRE INTERFACE 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 setting.

A) VOLUME / TONE CONTROLLER COMMANDS

Device address = 10 (Address precedes command)

Data field

Set Master Volume	
001 DDD DDD	Set Master Volume
001 000 000	-80 dB
001 010 100	-40 dB
001 101 XXX	0 dB

Set Left Channel Volume	
101 XDD DDD	Set Left Channel Volume
101 X00 000	-40 dB
101 X01 010	-20 dB
101 X10 1XX	0 dB

Set Right Channel Volume	
100 XDD DDD	Set Right Channel Volume
100 X00 000	-40 dB
100 X01 010	-20 dB
100 X10 1XX	0 dB

Set Treble	
010 XXD DDD	Set Treble
010 XX0 000	-12 dB

010 XX0 100	0 dB (Flat)
010 XX1 100	+12 dB

Set Base	
001 XXD DDD	Set Base
001 XX0 000	-12 dB
001 XX0 110	0 dB (Flat)
001 XX1 100	+12 dB

Set Mix	
000 XXX XDD	Set Mix
000 XXX X00	-12 dB
000 XXX X01	Mix GI sound chip output
000 XXX X10	Do not mix GI sound chip output
000 XXX X11	Reserved

Note: The volume controls attenuate in 2 dB streps. The tone controls attenuate in 2 dB steps at 50 kHz (Note that these frequencies may change).

B) USING THE MW 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 register 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 usec) 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 previous write completes.

This loop is at the beginning of the routine, not at the end because waiting at the end would always force a sixteen usec delay. Even if it has been longer than that since the last write.

4. NEW CONTROLLER PINOUT

4.1. ARRANGEMENT

Looking towards STE :

5	4	3	2	1
10	9	8	7	6
15	14	13	12	11

4.2. CONNECTIONS (PORT A)

1: UP 0	5: PAD 0Y	9: GND	13: LT 2
2: DN 0	6: FIRE 0	10: FIRE 2	14: RT 2
3: LT 0	7: +5v	11: UP 2	15: PAD 0X
4: RT 0	8: n/c	12: DN 2	

4.3. CONNECTIONS (PORT B)

1: UP 1	5: PAD 1Y	9: GND	13: LT 3
2: DN 1	6: FIRE 1	10: FIRE 3	14: RT 3
3: LT 1	7: +5v	11: UP 3	15: PAD 1X
4: RT 1	8: n/c	12: DN 3	

4.4. CONTROLLER MEMORY LOCATIONS

FF9201	---- 3120	RO	Fire Buttons
FF9202	UDLR UDLR UDLR UDLR	RW	Joystick 3/2/1/0 Directions
FF9211	xxxx xxxx	RO	X-Paddle 0
FF9213	xxxx xxxx	RO	Y-Paddle 0
FF9215	xxxx xxxx	RO	X-Paddle 1
FF9217	xxxx xxxx	RO	Y-Paddle 1
FF9220	---- --xx xxxx xxxx	RO	Light Gun X-Pos

FF9222	---- --xx xxxx xxxx	RO	Light Gun Y-Pos

4.5. VIDEO REGISTER MODIFICATIONS

FF8205	--xx xxxx	RW	Video Address High
FF8207	xxxx xxxx	RW	Video Address Mid
FF8209	xxxx xxx-	RW	Video Address Low
FF820D	xxxx xxx-	RW	Video Base Address Low
FF820F	xxxx xxxx	RW	Over-Length Line Width
FF8264	---- xxxx	RW	Undocumented HSCROLL register: no extra fetch
FF8265	---- xxxx	RW	Documented HSCROLL register
FF8240+	---- 0321 0321 0321	RW	Red/Green/Blue Colour settings

4.6. DMA SOUND REGISTERS

FF8901	---- --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.			
FF8903	--xx xxxx	RW	Frame Base Address (high)
FF8905	xxxx xxxx	RW	Frame Base Address (middle)
FF8907	xxxx xxx-	RW	Frame Base Address (low)
FF8909	--xx xxxx	RO	Frame Address Counter (high)
FF890B	xxxx xxxx	RO	Frame Address Counter (middle)
FF890D	xxxx xxx-	RO	Frame Address Counter (low)
FF890F	--xx xxxx	RW	Frame End Address (high)
FF8911	xxxx xxxx	RW	Frame End Address (middle)
FF8913	xxxx xxx-	RW	Frame End Address (low)
FF8921	m--- --rr	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