Technical Overview

Version 10

Stephen Moss

20/12/2021

Contents

Overview of the Jaguar Hardware Architecture

If you are new to the Jaguar, we recommend that you look at the first few pages of the **Jaguar Software Reference Manual** document for a basic overview of the Jaguar hardware and systems architecture. After you've taken a look at that, come back to this document for an overview of the developer's kit and some more specific information about certain aspects of the system.

Jaguar Development System Overview

What follows is a brief description of the tools for both the Atari and SkunkBoard Jaguar development systems. Detailed instructions and explanations are found in specific documentation for each item.

These development systems consists of a set of hardware and software components intended to make writing software for the Jaguar the most efficient and rewarding experience it can be. This goal can only be approached, never reached. As a result, all of the components of this systems may be enhanced as time goes by; some will be deleted and others will be added in the future.

Atari Development System

The hardware components of the system are a development Jaguar machine that connects to your existing PC/MS-DOS computer with an 80386 or better CPU[1]. The development system comes with an I/O card for your PC that features an 8-bit by bidirectional parallel port. This is used to interface with the Alpine board that plugs into the Jaguar development console. If your PC already has an 8-bit bidirectional parallel port, you can probably use it instead of the card we supply. However, please note that most inexpensive I/O cards do not have such ports.

The Jaguar development console is a modified version of the standard Jaguar retail machine. It comes with a ROMulator that holds your programs and emulates a ROM cartridge (aka “the Alpine Board”), and other optional components (documentation is included with those components).

The software components are many. In the Jaguar development machine, there is a debugging stub in ROM which communicates with the host computer via the Alpine board interface card. It is designed to take a minimum amount of system Resources. The software under development need not depend on the stub for ANY services, yet the debugging environment is quite complete and powerful.

SkunkBoard Development System

The hardware components of this system are a standard Jaguar retail console that connects to your existing Windows 98b (or higher) computer with a Pentium or better CPU and at least one USB port[2]. The development system comes with a USB A to USB mini B cable which is used to interface the PC with the SkunkBoard that plugs into the Jaguar console.

Depending on which version you have your SkunkBoard will contain either 4Mb or 8Mb of flash ROM that holds your program(s) and emulates a ROM game cartridge.

1 Instead of a PC system, any Atari TOS computer can also be used for development. The choice of TOS computer depends on the uses that the machine will need to perform beyond simply running the development System Software. For best performance and greatest flexibility in a pure debugging environment, an TT030 system with the TTM195 19” monochrome monitor is recommended.

2 The SkunkBoard has also been tested with OS X and Linux kernels capable of running libUSB.

The main software tools are: the Atari debugger DB; software development tools such as the MADMAC (or SMAC) macro assembler, ALN (or SLN linker) and GNU GCC or VBCC compiler. There are also Jaguar specific debugging aids, sample code and library code. Together these provide a set of tools that allow full use of the extensive capabilities of the Jaguar a system (see **A Sample Debugging Session**).

Most of the tools are command line-oriented; you pass them a command line, they do what they're told, and then they quit. In most cases, you don't actually interact with them while they are running.
The exception to this rule is the Atari debugger "Db". Db is a full featured symbolic debugger with aliases and procedures that have been in use in the Atari computer development environment for many years. It has been updated and enhanced with numerous new features and special debugging aliases and procedures for the Jaguar development system. There are two variations; RDBJAG (remote DB for Jaguar) features a simple terminal style interface, while WDB (windowed DB) features a semi-graphic user interface using the mouse, windows, and pull-down menus.

The Object processor in a Jaguar is an unfamiliar mechanism to most programmers and this can be a bit of a hurdle when starting to program the system. To overcome this problem, we have provided a heavily documented routine which is used by several of the sample programs included in the developer's kit. Please see the examples in the JAGUAR\WORKSHOP directory after you have installed your developer's kit discs. The very useful tool for the Object processor programmer is OD, a script procedure for the DB debugger that translates an object list into English and one about common mistakes.

The Jaguar GPU is a high performance custom RISC processor that was optimised to give maximum performance when program in assembly language in graphics applications. The instruction set is general purpose with specific instructions added to do matrix multiplication and simple floating point maths. Db has a GPU disassemble and register dump as well as a GPU single step facility for GPU debugging (see **Debugging the GPU**). The GPU should not be a difficult system facility to master since this instruction set was designed with the programmer and in mind. The DSP is very similar to the GPU in both design and instruction set, the main difference being some extra instructions for sound processing.

The MADMAC (or SMAC) macro assembler provided in the developer's kit is capable of generating code for the GPU and DSP as well as the 68000. Older versions of the developer's kit also provide the GASM macro assembler four GBU/DSP, but this has been made obsolete by SMAC and newer versions of MADMAC.

The ALN (or SLN) linker is used to link your object modules and libraries compiled or assembled from different source code files and create an executable file ready to be run on your Jaguar.

There is also a set of programming utilities included in the system. These include a MAKE utility, a file hex DUMP utility, a version of GREP (the UNIX search utility), and a variety of object modules & executable file information utilities. These are documented individually in the **Tools** section.

A text editor is not provided with the system because we expect that you will probably already have an editor that you are familiar with and would be unlikely to want to switch.

Note: The Atari Debugger, DUMP, GREP, RGBJAG and WBD are not designed to use a USB port and therefore cannot be used with the SkunkBoard.

A Sample Debugging Session (Alpine Board only)

To help you become acquainted with the Atari debugging environment, we will load in a program that uses both the 68000 and the GPU and take a look around. The programs that we will use is JAGMAND, a very simple Mandelbrot set generator. This is the same program that we used in the **Getting Started** section to verify that the system was working correctly, so we already have built the executable. Change to the\JAGUAR\SOURCE\JAGMAND directory and start the debugger from the shell by typing `"rdbjag"` (pressing return is implicit here, this instruction will not be repeated).

We won't go into details about how the sample program itself works, as this is explained elsewhere.

First we load the program into memory in the Alpine board. The debugger uses the first part of the system memory for variables, stack, and added GPU specific code. Therefore, all RAM below $4000 is reserved. All cartridge-based Jaguar programs must start at $802000.

To load in the programme we type `"aread jagmand.cof"`. This loads the sample program into memory at the locations specified by the executable (as specified by the commands given to linker). A map of the memory space used is also displayed. An alternative to the AREAD command is the LOAD command, which loads and executes a script file which can in turn load binary data into the Jaguar's memory by using READ or FREAD commands.

At this time we can look at our program by typing `"l 80200"`. This will disassemble (or list) the 68000 code starting at address $802000. (Note that the debugger uses hexadecimal notation by default.) If you first set the program counter using the command `"xpc 802000"`, you can trace one instruction at a time using the `"t"` command, or execute a subroutine with the `"tw"` command. Try this for the first few instructions and subroutines.

At this point, let's set a break point at the label "*_start*". This is done by typing `"b ._start"`. Before the break point is reached, the program's startup code has been executed. This startup code initialise is the Jaguar hardware correctly, sets up an object list and displays a simple startup screen.

Type `"g 80200"` to begin execution at the start of the program (or, if you traced some of the programme already you can just type `"g"`) and run until the break point is reached. When the break point is reached the interval state of the 68000 is displayed and the debugger waits for another command.

At this point the memory starting at the *listbuf* lable contains the object list created by the startup code for the startup picture. Type `"od .listbuf"` to see a display of the object of list that is being used. It should be noted that the object list should be viewed before video processing is started because the object processor changes values in the objects during processing. These are restored each frame by interrupt software, but looking at an active object list with `"od"` will not give correct data for the data pointer or the object height fields.

Type `"g ,.Mandle"` to skip the 68000 code the copies the GPU code to GPU RAM. This will take a few seconds, because the program has a short delay so that the startup screen may be seen. Note that the debugger will print the message `"Press Control-C to stop waiting"` on the screen. This is because the Jaguar system did not respond quickly to the `"g"` command and return control to the debugger.

Now let's look at some code in the GPU. To do this type `"lg f03000"`. The address used here is the location of the start of GPU RAM. To see the values in the GPU registers type `"xg"`. At this point the GPU may be single-stepped by setting the GPU program counter by typing `"setgpc f03000"` and then typing `"tg"` a number of times. Although nothing terribly interesting is likely to be learned, let's give it a try.

Next we won our program by typing `"g"`. There are a few interesting things to note of this stage. First, the Mandelbrot computation is REALLY quick (despite this, there is AT LEAST the factor of two times more performance that can be squeezed out of the system). Second, the debugger again printed the message `"Press Control-C to stop waiting"`. However, once the program completed one pass over the Mandelbrot set it is stopped in a rather brute force, but effective, way. It executed an illegal instruction. This got the debuggers attention and control is returned to the debugger. Despite this, there is an interrupt happening once a frame still running to fix up the object list.

To leave the debugger type `"q"`. This will sever the communications at the computer side but leave the development system ready for more commands. Type `"rdbjag"` and the stub should "check out ok".

If for some reason the stub and debugger fail to communicate, type the `"wait"` command in Db and press the reset button on the Alpine Board. This will get the attention of the debugger whenever it is "Waiting...".

A Simple Sample Program

We have looked at the JAGMAND sample program twice now. Aside from drawing the Mandelbrot set fractal, this program also point out many of the features characteristics of both the Jaguar and the development system. While it is in many ways very simple, note that the JAGMAND program does use the Blitter to clear the screen.

There are a number of very mundane things that must be considered when writing a Jaguar program. In no particular order these include:

1) **Where in memory will the various segments be?**

   The debugger in the development system takes up the lower 16K of memory. Programs should therefore use no RAM lower than $4000. The rest of RAM is yours to do with as you please. The ROMulator should be used to hold the programmes text and data segments. The first part of ROMulator memory is also reserved, this time for the security code. Cartridge-based programs must always start at $802000.

2) **Where is the 68000 stack?**

   Keeping in mind the restrictions mentioned above, you can put the stack anywhere in RAM above $4000 you want. Probably the best place is at address $1FFFFC. This is 1 long word away from the end of RAM.

3) **How do you set up video, clear interrupts, and initialise memory at startup time?**

   We supply a standard startup routine that initialises the entire system and then jumps to your program code. This is contained in the JAGUAR\STARTUP directory. The JAGMAND programme includes the STARTUP.S file, containing this startup code.

4) **Setting up an object list.**

   The choice of object list structure is quite complex and depends greatly on what your goals are. Since there is no good general solution we give you a VERY simple one here. A single full screen object. This uses an unscaled bit mapped object. The object is the height of the screen.

5) **Putting stuff in the object to be displayed.**

   The JAGMAND program draws a Mandelbrot fractal into the bitmap displayed by the object. Of course, your program is going to draw whatever is appropriate for it.

Jaguar and Memory

This document describes the memory map of the Jaguar (Tom and Jerry) development system.

Main system RAM in Jaguar is 64 bits wide. It contains a single two-megabyte bank starting at memory location $00000000. The rest of the system memory consists of hardware registers. These registers include the internal high speed SRAM for holding the GPU and DSP programs and data. This starts at $00F00000.

The GPU, DSP and Blitter internal registers of 32 bits wide and MUST be read and written as such. When accessing these memory locations with the 68000 CPU they must be read and written as 32 bit entities. This is especially important with regard to the GPU and DSP internal SRAM. Transfers to (and from) this memory, to pass parameters between CPU and GPU for example, must be made at long word boundaries. Please note, to clear a long in internal GPU/DSP RAM space use the **move** instruction, because the **clr.l** instruction will not be reliable. (Please see the **Hardware Bugs & Warnings** section for further information about this subject.)

The last kind of memory in the development system is the ROMulator, described later in this section.

The Jaguar Blitter

The Jaguar Blitter is a very powerful piece of the Jaguar graphics system. This document will introduce the major functional parts and shows some of the many ways in which they can work together.

The programming model of the Blitter consists of:

1) Two address generators.
2) A Logical Function unit.
3) A Pattern Data register.
4) A Gouraud Shading unit.
5) A Z-buffer unit.
6) A collision detection system.

The two address generators are easy to use because they work in pixel units, not addressed units. This greatly simplifies the coding tasks for Blitter use.

The basic concept used in both address the generators is the “window”. In a window is a rectangle of memory whose width is taken from the list of allowed widths (see BLIT.INC for the Allowed Widths). The maximum allowed height of a window is 4096. If no outer loop is used, the window width is not relevant and the maximum sized blit allowed is 32767 pixels.

There are two address generators A1 and A2.

A1 has the ability to traverse its window in fractional steps with complete independence in X and Y. The inner and outer loops are controlled independently and the outer loop increment may also contain independent, factional X and Y values. These features combine to allow arbitrary rotation, skewing and scaling of rectangular areas.

A2’s special ability allows it to repeat a source pattern over a large destination by masking the pixel offsets. The masks can be any power of two size up to 2^15.

The Logical Function Unit takes the source and destination and produces an output based on the logical OR’ing of the four possible minterms. Four of these combinations are of particular use:

Destination	<=	Source		
Destination	<=	(Source)	|	(Destination)
Destination	<=	(Source)	&	(Destination)
Destination	<=	(Source)	^	(Destination)

A complete listing of these is given in the system include file BLIT.INC

The Pattern Data register is where the Blitter gets its data without the need for reading source data. This is used, for example, in drawing lines.

The Gouruad Shading Unit is one of the most powerful features of the Blitter. It allows the automatic shading of CRY pixels. (See the description of the CRY colour model in the **Jaguar Software Reference Manual** for more information). The Gouraud shader uses the Pattern Data register as the source with the added capability of adding a constant (fractional) intensity to each pixel. This allows the generation of a smoothly shaded line with no explicit computations done at the pixel level.

In the same way that shading is handling hardware, the line produced by the Blitter can also have a Z value automatically provided for each pixel and the Blitter can be instructed to suppress writing of pixels with Z values that correspond to 3D points that should not be visible.

Note: Gouraud shading and Z mode are only available with 16 bit pixels.

Another important concept to understand in the Jaguar Blitter is phrase mode. The inner loop increment used by the Blitter is controlled by the first few bits of the FLAGS register for each address generator. These modes are fairly self explanatory, except for phrase mode.

In phrase mode the Blitter reads and writes 64 bits of data at a time. The Blitter handles all fringe cases and data alignment automatically in8 and 16 bits per pixel. For smaller numbers of bits per pixel, pixel mode should be used. Note: BOTH address generators must be in phrase mode. It cannot be half set.

There are two extra complexities when dealing with phrase mode. It is possible that the first data right requires an extra phrase read. This happens whenever the data for the first write is not contained in the first data read. Consider for example a 16 bit per pixel blit:

(The vertical bars are 64 bit phrase boundaries)

```
Source:
|    |    |
   abcd

Destination:
ABCD
```

The Blitter needs two source reads to get all of the data for the first data write. This extra read is caused by setting the SourCe ENable eXtra (SCRENX) bit in the B_CMD register. Other situations also require this bit to be set. For example:

```
Source:
|    |    |
    abcd

Destination:
  ABCD
```

The other extra complication involves the STEP value used in the outer loop. Since the Blitter always advances to the end of a phrase the STEP size is not always the width of the blit. An example should make the general principle clear:

```
Source:
|    |    |    |
   abcdefgh

Destination:
   ABCDEFGH
```

In both cases the STEP goes from the end of the third phrase to the beginning of the data. In this case this gives a STEP of -10 for the source and -9 for the destination.

Also remember that if SCRENX is set an extra phrase worth must be subtracted from the source STEP value.

Phrase mode also has an effect on Gouraud shading. Since the Blitter writes four pixels at once all four pixels must be placed in the Pattern Data register and the value of the intensity increment must be multiplied by four. This means that the maximum intensity increment that will work in phrase mode is 31.

Since the intensity addition saturates and the increment is signed there are a few cases that will fail. These all share the following characteristics: The first pixel to plot is not on a phrase boundary and the extrapolated value for the first pixel falls outside of the allowed values. Software authors need to be aware of this condition. It should either be rigidly excluded or a switch to pixel mode is needed.

Stubulator & ROMulator

The *Stubluator* is what we call the version of the Jaguar console that is used as part of the Atari Jaguar Development System. Also know as a *Jaguar Test Station*, it is essentially a standard Jaguar console which has been modified to use a special debugging version of the boot ROM, and which has an extra cable attached which connects to the ROMulator board to handle the stop button interrupt.

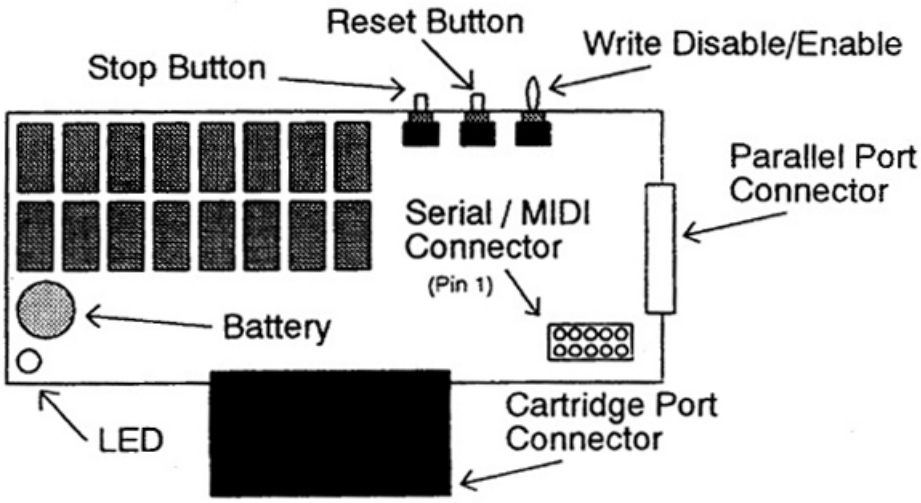

Figure 1, The ROMulator Board (Front)

The ROMulator, also known as the Alpine Board, serves two purposes. First, it allows the Jaguar console to communicate with your computer via a parallel port or serial connections. Second, it contains 2 or 4 megabytes of battery backed-up static RAM[3] which is used to emulate a ROM cartridge. Hereafter we will refer to the RAM memory on the ROMualtor as ROM in order to distinguish between it and the RAM inside the Jaguar console.

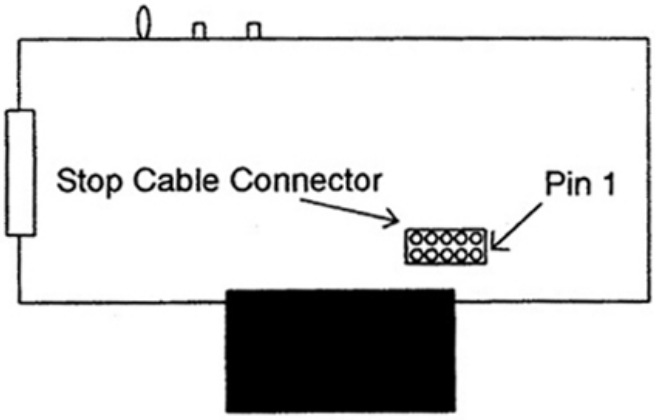

Figure 2, The ROMualtor Board (Back)

3 The standard Alpine board shipped with the Atari Jaguar Development system contains two megabytes of static RAM. However, some four megabyte (32 megabits) Alpine boards are also available.

The Alpine board has a variety of components that you should become familiar with, as highlighted in figure #1 and #2. The table below briefly describes each one.

Component	Description
Stop Button	When pressed, this button generates a non maskable 68000 interrupt in the Stubulator. The debugger stub handles this interrupt, stops the current process and then passes control to the debugger. If a program has severely crashed, the 68000 had been disabled, or a program has altered the interrupt vector, then the stop button may have no effect. This is rare, but it does happen occasionally.
Reset Button	This button generates a hardware reset of both the Jaguar console and the Alpine board. The current program is halted, and one or more of the following actions are taken: 1) The debugger stub initialises itself to use memory in the $800000 to $801FFF range of the Alpine board. If the Alpine board is write protected, or if a ROM cartridge is plugged in, then it proceeds to action #2. 2) The debugger stub initialises itself to used memory in low DRAM (below $4000) in the console. Then it proceeds to action #3, #4, or #5. 3) The cartridge port is checked for a 32-bit cartridge. If found, then the 68000 starts executing code at $802000. If not, it proceeds to action #5. 4) The cartridge port is checked for an 8-bit cartridge. If found, then the 68000 starts executing code at $802000. If not, it proceeds to action #5. 5) The debugger stub displays the “Jaguar Development System” screen and attempts to communicate via the Alpine board with the debugger interface running on your computer. The exact combination of actions depends on which buttons of Joypad in controller port #1 are pressed when the reset button is pushed. If no buttons are pressed, then actions #1, #2, and #5 are taken. If the “B” button is held down, then actions #3 and #5 are taken. If the “C” button is held down then actions #4 and #5 are taken. Neither console RAM or Alpine board SRAM is cleared by this reset. However, interrupts are cleared.
Write Enable / Disable Switch	This switch allows you to control if the RAM on the Alpine board may be written to. If this is set to “Write Disable”, then the write lines of the memory chips are physically disconnected so that the memory contents cannot be altered.
Battery	This is used to maintain the contents of the static RAM when the console power is turned off or when the Alpine board is not plugged in.
LED	This is lit when the Write-Disable switch is set, the Alpine board is plugged in and the console is turned on.
Serial / MIDI Connector	This is the connection used either for a serial link to your host computer or the Alpine MIDI adaptor.
Parallel Port Connector	This is the connection used to communicate with your host computer’s bi-directional parallel port.
Cartridge Port Connector	This is where the Alpine board plugs into the Jaguar console.
Stop Cable Connector	This is where the stop cable coming out of a developer Jaguar console connects to the Alpine board, in order for the Stop button function to work.

The Alpine board plugs into the Jaguar console in the same manner as a standard Jaguar cartridge. The front of the Alpine board, as shown above, faces the front of the console (where the power switch and controller connectors are located). A Jaguar Test Station should also have a 10-pin ribbon cable coming out of the back. This is the Stop cable which connects to the back of the Alpine board. Make sure that the red-striped wire of the ribbon cable always goes to pin 1 of the connector on the Alpine.

Newer releases of the Alpine board come with a 32MHz crystal and a header fitted in space J4 (J4 is marked as the Serial / MIDI connector in figure 1). Only those Alpines with those components can be used with the MIDI add-on board. If your Alpine is an older model you will not be able use the Jaguar MIDI board.

ROMulator Memory

The ROMulator memory starts at $800 000, the same address space used by a cartridge, and is treated by the system is 32-bits wide. In order to emulate a ROM cartridge, the ROMulator memory may be write protected. This is accomplished using the WRITE DISABLE/ENABLE switch at the top of the board. The ROMulator is write protected when the LED in the bottom left corner is ON.

Just as with a real cartridge, all static code and data must start in ROM and get copied to the consoles RAM by the program as needed. No write to ROM should be done by the game code. This may be tested by the following steps:

1) Load the program into the ROMulator using the debugger.

2) Turn the switch to WRITE DISABLE

3) Turn the machine off for a few seconds, then on again.

4) Run the program and make sure it functions normally.

Debugger Stub

The debugger stub also uses a section of the ROMulator space. To leave room for the security code that will be in each cartridge, the first $2000 of the ROMulator (from $800000 to $801FFF) is NOT to be used by your programs. The restriction on the use of the first 16K of RAM ($0000 to $3FFF) is also still in effect.

The debugging stub normally tries to use memory in the ROMulator, but it can optionally use DRAM if necessary. The sign-on message shown by the debugger indicates how the stub is using memory. There are two possible reasons for the stub not to use the ROMulator:

1) The ROMulator is not present or damaged in some way.
2) The ROMulator is write-protected AND the stub is NOT ALREADY loaded.

This allows the system to be reset with a write protected ROMulator and still work. If the stub reports that it is running from DRAM, the ROMulator data has probably been disturbed.

To force the stub to use DRAM, you can hold down the ‘A’ button on the Jagpad in controller port #1 while turning on the Jaguar’s power supply or pressing the ROMulator reset button. Normally, however, this should not be necessary.

MIDI Add-On Board

The MIDI Add-ON board is a special add-on board that connects to the serial port of an Alpine board and allows you to feed MIDI data to a special version of the Jaguar Synthesizer. This effectively turns the Jaguar into a stand-alone synthesizer which can be controlled by an external keyboard, sequencer, or by a computer equipped with a MIDI port and MIDI software. This allows you to preview your music on the Jaguar itself.

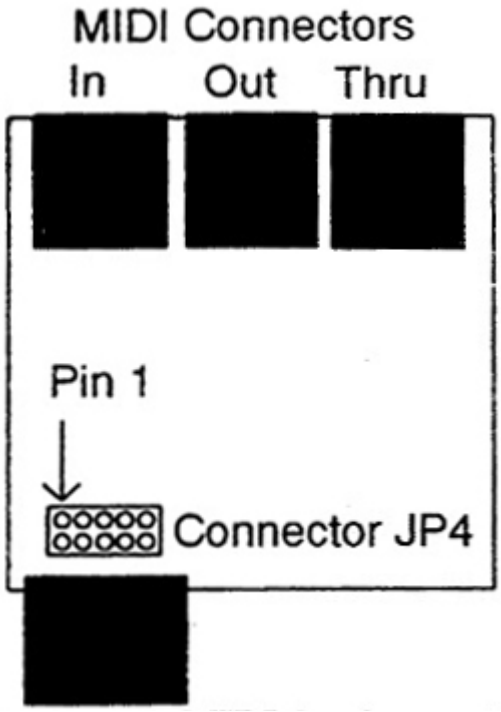

Figure 3, Jaguar MIDI development board

To connect the Jaguar MIDI board, simply connect one end of the supplied 10-pin ribbon cable to connector JP4 on the MIDI board and connect the other end to the Serial port / MIDI connector of the Alpine board. Make sure that the red-striped wire of the ribbon cable goes to pin 1 at both ends.

Once the Jaguar MIDI board is connected, it can be used with the Jaguar Sound Tool (the patch editor for the Jaguar Synthesizer). See the documentation for the Sound Tool for further information.

SkunkBoard

The SkunkBoard is essentially a Flash ROM cartridge that serves two purposes. First it allows the Jaguar console to communicate with your PC via a USB connection. Second it contains either 4Mb or 8Mb of flash ROM which is used to emulate a ROM game cartridge.

If you have an 8Mb SkunkBoard you can use it as either two banks of 4Mb for storing two separately executable sets of 4Mb game code when developing games for a standard 4Mb cartridge, alternatively you can use it as one 6Mb bank if developing games for a 6Mb game cartridge.

Three versions of the SkunkBoard have been produced...

V1 has 4Mb of Flash ROM
V2 has 8Mb of Flash ROM
V3 is identical to V2 except the issue in note 1 (below) has been corrected

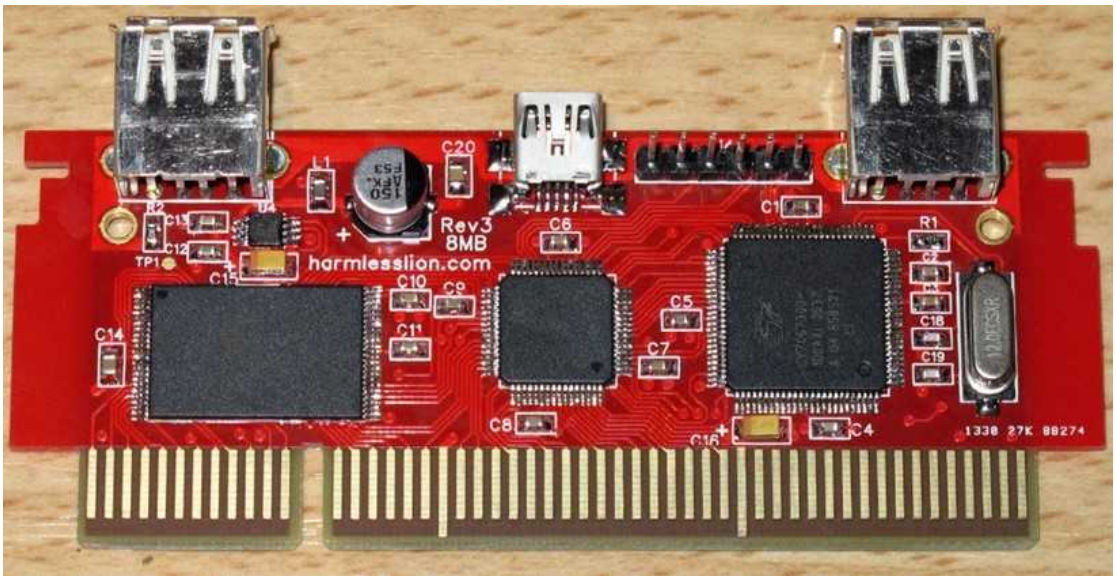

Figure 4, SkunkBoard (Component/Rear Side)

The SkunkBoard plugs into the Jaguar console in the same manner as a standard Jaguar cartridge. The component side of the SkunkBoard, as shown above, faces the rear of the console (where the power connector, DSP and Video ports are located).

Note:

1. Some SkunkBoard V2s have repotedly burnt out their Flash memory, the producers believe that this is caused by a power surge when the power unit is connected/switched on while the Jaguars power switch is in the on position.
   It is therefore recommended that you ensure the power switch is in the off position or remove you SkunkBoard before applying power to your Jaguar.

2. Although supplied as a basic assembled PCB the SkunkBoard can be mounted inside a game cartridge case, although some modification to the cartridge case will be required. Mounting the SkunkBoard inside a cartridge can both provide extra protection and make it generally easier to handle.

Jaguar Controller Support

The Jaguar supports a variety of different controller types beyond the Joypad that comes with every console. In order to insure that controllers are correctly supported, we urge developers to pay close attention to the **Jaguar Controller & Controller Port Specification** section of the **Technical Reference document.**