Processor Board Specification
for the

DoubleThink

AT&T DSP3210 + MC68040
Processor Module

by Dave Haynie

1. The System

DoubleThink is an Amiga 4000* compatible processor card which combines the MC68040 microprocessor with the AT&T DSP3210 Digital Signal Processor. The combination yields a system with more available integer and floating point performance than any previous Amiga, needed for today's multimedia applications. Various DSP peripherals may be added via a DSP interface header, though a standard audio CODEC is expected to come with the basic card. A diagram of this card is shown below in Figure 1-1.

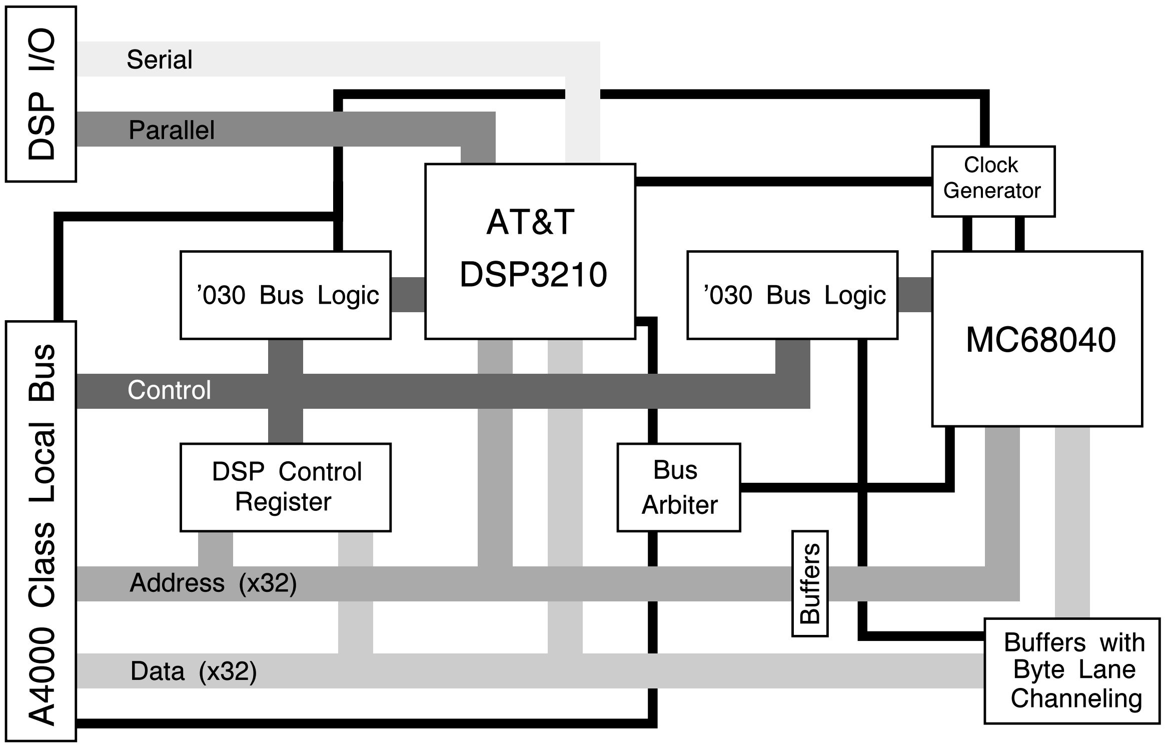

Figure 1-1: The DoubleThink System

1.1 The MC68040 Interface

The MC68040 interface is based directly on the interface used for the MC68040 card designed for the Amiga 4000 by Scott Schaeffer. This consists of a clock generator, any one of the MC68040 processors (68040, 68LC040, or 68EC040), a 68040 to 68030 cycle conversion machine, a bus arbiter, and a byte lane channeling buffer system to support the sizable bus of all Amiga 3000 architecture machines.

The clock generator is an enhanced version of the clock generator on the '040 card. It provides standard 50MHz PCLK and 25MHz BCLK for the '040, the 25MHz CPUCLK and CLK90 pair for the Amiga host, and now the 50MHz clock for the DSP3210. The exact nature of that clock will be elaborated upon in the next section.

* This card is based on the A4000 processor module specifications. It will work in other Amiga 3000 architecture machines with the addition of the two interrupt lines to the coprocessor slot (Amiga 3000 or Amiga 3000T).

The need for the cycle conversion machine is pretty obvious -- the Amiga 3000 architecture is based around the MC68030 bus, this card is based on a MC68040. This machine takes the 68040 bus protocols and converts them to 68030 bus protocols. This process is complicated by the fact that the ’040 doesn’t support the dynamic bus sizing of the ’030, nor does it support asynchronous bus termination. Therefore, this conversion logic also manages the byte lane channeling system. This network of data bus buffers can shift data between byte lanes, as necessary for communications to non-longword ports. As with the original 68040 card, I don’t expect this interface will support the 68030 bus burst modes, as that greatly complicates the bus conversion logic.

In the original 68040 card design, the bus arbiter needed to consider motherboard requests, as channeled through Buster, and the ’040 requests. Additionally, this logic now needs to consider the DSP3210 bus request. Some portion of the logic needs to "snoop" the bus during arbitration for the DSP3210, as it doesn’t monitor the bus for acquisition like ’030 bus masters are required to do.

1.2 The DSP3210 Interface

The DSP3210 interface will more than likely be loosely based on the AA3000 based DSP3210 prototype systems. This system is described fully in the document Implementation Notes for the AT&T DSP3210 Digital Signal Processorand the MC68030. As shown in Figure 1-1, the DSP3210 is actually located on the 68030-based local bus of the host system rather than on the 68040 bus. While the DSP3210 is actually just a bit more at home on a 68040 bus, the 68040 bus interface will add wait states to all memory accesses. The DSP3210 realtime response and host bus utilization is dependent on fast memory, so its living on the 68030 bus will let it access all longword ports at full bus speed. While the DSP3210 doesn’t support bus sizing, there is relatively little use in giving it access to eight or sixteen bit port resources, since all these must be managed by the OS on the host processor.

The DSP3210 runs from a 50MHz basis clock. The exact phasing of this with respect to the motherboard clocks will be dependent on the exact design of the DSP3210 to 68030 bus interface logic. The AA3000-based prototype used a 50MHz clock intentionally skewed from the 68030 clock, but that was more out of need than desire. I expect the DSP clock to look something like the 68040’s PCLK. Some of the 68030 bus interface, and the DSP’s data bus latch, will run directly from the host system’s CPUCLK (or a buffered copy of it).

There are two main functions of the DSP3210 to MC68030 bus interface. The first part concerns the start of the cycle. The DSP3210’s address strobe falls based on the DSP clock, so it can fall in either phase of the 68030 cycle. Logic must generate the 68030 AS* and DS* based on the DSP’s AS* and the READ strobe. Once the cycle has started, the interface logic must presample the DSACK$_1$*/DSACK$_0$* combination and combine that with STERM* to create the DSP’s SRDY* termination strobe. SRDY* must be driven synchronously to be sampled while 68030 bus data is valid. The 3210 is actually made to latch 68030 bus data whenever CPUCLK is low, so the timing on this strobe is not overly critical. Its also necessary to sample the 68030 bus error signal, BERR*, and present it to the DSP3210’s error input synchronously.

The original design for the prototype DSP3210 interface logic called for DSP3210 burst cycles to be converted to 68030 burst cycles. This is still something that can be added, though it is a bit more complex. The 3210 doesn't handshake bursts, it simply tells external logic when a burst is being driven. This could limit the kind of memory the 3210 can talk to. The DSP3210 burst cycle is more like the 68040 cycle than the 68030 cycle, especially in the way the sizing bits work. Therefore, any burst circuitry will have to provide '030 compatible SIZ1 and SIZ0 during bursts. At all other times, the sizing bits work the same as the 68030 sizing bits.

For bus arbitration, the DSP3210 generates a BR* strobe. When it gets a BG* synchronously returned to it, it generates its BGACK* to indicate it has acquired the bus. It doesn't monitor its BGACK* or any of the 68030 signals, of course, so external logic must monitor things like AS*, STERM*, the DSACK*s, etc. once the arbiter determines that the DSP can be granted the bus. This logic will also need to drive the 68030 BGACK* signal based on this monitoring. The DSP3210 must be the highest priority master in the system, due to its deterministic requirements. Additionally, AT&T recommends that arbiter logic wait four clock cycles after a DSP cycle terminates and no new BR* is generated before relinquishing the bus to a new master. This seems to be a good idea, and it would be reasonable just to park the bus at the DSP3210 (or 68040) if no requests come in once a relinquish condition has been determined.

The final piece of DSP3210 interface logic is the DSP control register (DCR). This is actually two registers, one for input, one for output. The processor slot's memory select strobe is used to generate the chip selects for the registers, which should be located away from the base of this region to avoid being written by the OS's memory polling routine (used to determine is memory is present in this slot). More information on the mapping of this register is in the software chapter.

The main point of the DCR is to provide CPU control over various actions of the DSP, and a few DSP signals back to the CPU. The CPU can reset the DSP or cause either level 0 or level 1 interrupts on the DSP. These are normally self-cleared by the DSP's interrupt acknowledge lines, though one bit in the DCR allows the level 1 interrupt to be made "sticky", following only the CPU given setting, ignoring the acknowledge line. This is used to support a DSP debugger. The DSP, in tun, can cause CPU interrupts INT2* or INT6*, generally driven by two of the DSP's BIO port lines. To allow the CPU to shut these down when necessary, the DCR provides two mask bits. The DCR outputs power up with the DSP in reset, both DSP interrupts negated, sticky mode negated, and the CPU interrupt masks asserted.

1.3 The Audio Interface

The most immediately useful DSP peripheral is a hi-fi audio I/O interface. I expect that such a card will be the standard DSP I/O module for this system. The audio interface is on a separate card, connected to the DoubleThink PCB via ribbon cable to theDSP port. It provides for a stereo microphone input, stereo line-level output, both via external connecter and shielded cable for connection to internal audio mixer inputs available on systems like the Amiga 3000T and Amiga 4000.

The audio CODEC used here is up to the designer, of course. Two main contenders are the ASCO 2300 from ITT and the AD1849 from Analog Devices (or pin-compatible CS4215 from Crystal Semiconductor). These latter parts have on-chip clock synthesizers and should cost $10-$12 in quantity, the former requires an external clock generator and should cost around $6 in quantity. Either system should be capable of sampling at CD rates (44.1kHz), and it would also be reasoble to look into support of AAA system sample and playback rates.

A PAL of some kind interfaces the serial bus of the CODEC to the flexible DSP3210 serial port. There are no hi-fi quality audio CODECs that hook directly to the DSP3210 available for a reasonable price, but neither interface should be too complex. On the audio side, proper amplifiers for condenser microphone input and line-level output is provided.

2. The Software

The major implementation-dependent impact on software for any DSP hookup is the physical format of the DSP Control Register (impacts the dsp3210.resource for the system) and the choice of DSP peripheral hardware (VCOS device drivers must be written to support such a device). More information on this can be found in the document Implementation Notes for the AT&T DSP3210 Digital Signal Processor and the MC68030.

The DSP Control Register is located for reads at $09000000 and for writes at $09000004. The suggested bit assignments, based on the AA3000 prototype system, are as follows:

Bit	Direction	Function
7	Read/Write	DSPRESET*. This bit comes up set low. When low, the DSP is in reset,when high, out of reset.
6	Read/Write	INT1STICK. This bit comes up set low. When low, IACK1 causesdspINT1* to automatically negate. When high, dspINT1* always followsthe setting of the DSPINT1 register, ignoring IACK1.
5	Read/Write	MASK INT6*. This bit comes up set low. When low, INT6* from theDSP to the CPU is masked out, when high, the DSP may cause a CPUlevel 6 interrupt. INT6* is caused by DSP BIO bit 6.
4	Read/Write	MASK INT2*. This bit comes up set low. When low, INT2* from theDSP to the CPU is masked out, when high, the DSP may cause a CPUlevel 2 interrupt. INT2* is caused by DSP BIO bit 7.
3	Read Only	CPU INT6*. This bit reads low when the DSP is trying to cause a level 6CPU interrupt, high otherwise.
2	Read Only	CPU INT2*. This bit reads low when the DSP is trying to cause a level 2CPU interrupt, high otherwise.
1	Read/Write	DSP INT1*. This bit is written low when the CPU wants to cause a level 1DSP interrupt. It stays low until acknowledged by the DSP, forINT1STICK negated, or low until changed here, for INT1STICK asserted.
0	Read/Write	DSP INT0*. This bit is written low when the CPU wants to cause a level 0DSP interrupt. It stays low until acknowledged by the DSP.

The details of the DSP peripheral hardware will depend on the chosen hardware and interface. The specification of either of the suggested audio CODEC chips contains all the register information needed for dealing with the CODEC itself. Additional information will be supplied in the system specification to document the serial port interface, so that software will know which serial signals to drive in which direction, and also if any BIO port lines are being used in the interface.

3. Other Notes

A few other bits and pieces of documentation are included here for completeness.

3.1 The AA3000 DSP Serial Bus Interconnect

The AA3000 prototype systems support a DSP serial bus interconnect much like what is needed in the Multimedia Engine implementation. This needs to be something that can run over short lengths of ribbon cable and provides most of the DSP serial and BIO port signals. The pinout of the AA3000 connector is as follows:

Pin	Signal	Pin	Signal	Description
1	+5Vdc	2	DI	Serial data input
3	-5Vdc	4	ICK	Serial input clock
5	GND	6	ILD	Serial input load strobe
7	GND	8	DO	Serial data output
9	GND	10	OCK	Serial output clock
11	GND	12	OLD	Serial output load strobe
13	GND	14	SY	Serial frame sync
15	+12Vdc	16	BIO5	BIO port bit 5
17	-12Vdc	18	N/C	Reserved
19	GND	20	BIO4	BIO port bit 4
21	GND	22	BIO3	BIO port bit 3
23	GND	24	BIO2	BIO port bit 2
25	GND	26	BIO1	BIO port bit 1
27	GND	28	BIO0	BIO port bit 0
29	GND	30	N/C	Reserved
31	GND	32	N/C	Reserved
33	GND	34	dspRST*	DSP full reset

3.2 Other DSP Peripherals

The hi-fi audio CODEC is only one possible DSP I/O device. The previous DSP systems investigates the use of a phone-line interface with phase correcting audio CODEC, to support various modem and FAX protocols. The document *Implementation Notes for the AT&T DSP3210 Digital Signal Processor and the MC68030* shows an example phone line interface using the Analog Devices AS28msp01 CODEC device. This would be a good starting point for a phone-line interface for the DoubleThink board.

In terms of more general expandability, the NeXT computer provides a D-Shell connector with DSP interface signals. This has become something of a de-factor standard for external DSP perhipherals. Some investigation toward providing this port would also be reasonable.