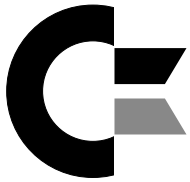


Implementation Notes
for the

AT&T DSP3210

Digital Signal Processor
and the MC68030

by Dave Haynie

1. DSP3210 and the Amiga

The DSP3210 is a 32-bit, bus mastering Digital Signal Processor from AT&T. We have been investigating its use as a local bus coprocessor in an Amiga 3000 class system since late 1991. This paper is a summary of the prototype system, based on a modified AA3000 development machine, and various notes on what was learned building such a system.

1.1 The AA3000 DSP Implementation

The basic DSP system is shown in Section 2. Three PAL devices create the DSP3210 interface to the MC68030 bus. The DMAC chip was originally intended to provide all DSP3210 controls via a programmable I/O added to it for the A3000+ system, but a bug in that port required the addition of another PAL device to act as part of the control port. If parts of the prototype look like kind of a hack, they probably are. The DSP specification was changing over the course of the system development, and I didn't have many shots at a PCB layout.

PAL U701 is used for clock generation, interrupt generation, and bus arbitration. The DSP clock, DSPclk, and its inverse, are created from CPUCLKA run through a tap delay. The idea is to generate a DSP clock that's intentionally skewed from CPUCLK. That lets the 68030 cycle generator PAL, U122, run from dspCLK* and safely clock in the CPUCLK state. This works, but with a better clock generator and more flexible 68030 state logic, I would expect little skew between CPUCLK and the DSP clocks, and the 68030 interface logic would probably run from CPUCLK or CPUCLK*.

The interrupt generator in U701 is very simple. Two port lines from the DSP3210, labelled dspCI2* and dspCI6*, are asserted to cause CPU interrupt levels two and six, respectively. The DSP control port provides interrupt mask lines dspMI2* and dspMI6* which permit the CPU to mask the DSP's interrupt generation. This makes DSP to CPU interrupts easy, since the DSP doesn't have to worry about immediately clearing the dspCIN* lines.

As mentioned, U701 also contains the main bus arbiter for the DSP. This uses a feature of the A3000's Buster chip in which the BR* line to the host CPU is time multiplexed between the Buster chip and another master. PAL U123 takes in the actual dspBR* request and generates a dspREQ* signal to U701. This is used to generate BR* on the right edge, assuming the Buster chip isn't already driving BR*. When U701 receives a BG* in response to BR*, it passes the dspGO* signal back to U123, which finishes the DSP's bus acquisition. This mechanism will more than likely change for the real DSP implementation depending on how the DSP is hooked into the system (eg, on a 68040 card, on a Zorro card, etc.).

As mentioned, U123 manages the rest of the bus arbitration and acquisition for the DSP. Originally, this PAL split the SCSI arbiter channel managed by the Buster chip. The final arbiter scheme, mentioned above, supreceeds this, so in reality, SBR* could go directly to XBR* and XBG* directly to SBG*, there's no need anymore to run this through a PAL. Of course, the actual implementation wouldn't split the SCSI channel anyway, it's more than likely going to be

worked into some kind of MC68040 arbiter. Its important to note that the DSP requires the highest DMA priority and reasonably fast access to the system memory. Because of this, the DSP will probably work better on the MC68030 bus than the MC68030 bus on an A3000 or A4000 CPU card, simply because the bus sizing hardware of the ’040 interface usually costs a clock cycle or two. Also, AT&T recommends delaying four clocks after a DSP cycle is over and dspBR* stays negated before giving back the bus. The logic here does add some delay, though its not quite four cycles.

Anyway, U123 manages the remainder of this arbitration and aquisition. Since the DSP can’t monitor the 68030 bus for takeover in the normal 68030 style, U123 and U122 watch the DSACK* lines, AS*, BGACK*, etc. after receiving a grant before supplying the dspBG* out to the DSP3210. The U123 logic was originally going to help out in support of a DSP3210 to 68030 bus burst translation, but this was removed when I needed a few PAL pins for other unforseen functions.

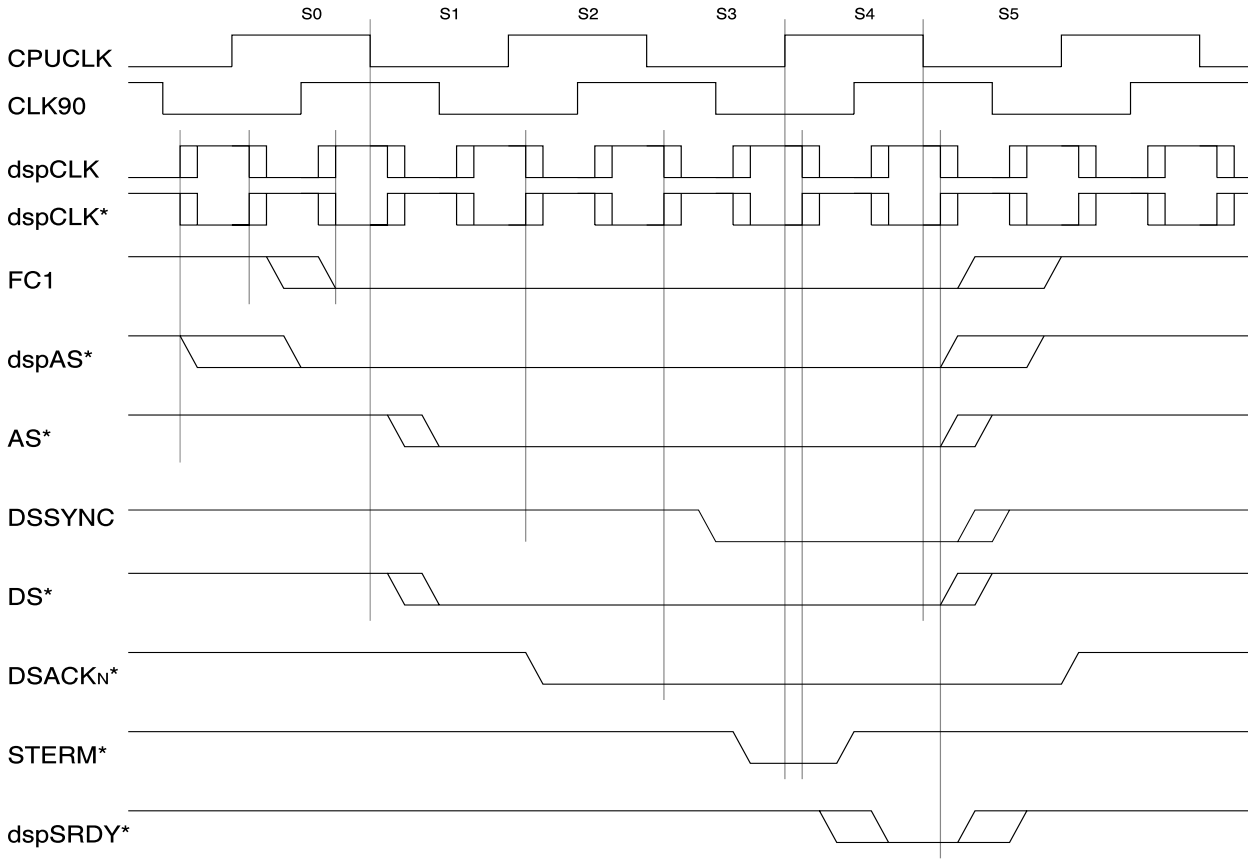


Figure 1-1: DSP to 68030 Cycle Translation

The U122 PAL manages the basic DSP to 68030 cycle translation. In the 3210’s big-endian Motorola bus mode, much of this interface is a direct connection. It is necessary, however, to make sure that the 68030 AS* comes out properly phased to CPUCLK, and a DS* signal needs to be created, based on AS* and R/W. The dspMS3 and dspMS2 lines directly correspond to SIZ1 and SIZ0, respectively, for normal cycles. However, the DSP lines follow

MC68040 conventions for burst cycles, so some translation would be necessary here to properly support a 68030 burst cycle. As mentioned, it was originally my intention to support burst, but this was sacrificed for other uses.

The PAL U122 also manages the end of cycle translations. The DSP3210 only communicates to 32-bit wide memory ports, which is fine. No attempt is made here to handle word or byte ports, but logic must take into account the fact that a longword port may be terminated with either the synchronous STERM* signal or asynchronous $DSACK_1$* and $DSACK_0$*. The DSSYNC* signal is the result of both DSACKs sampled around the falling edge of CPUCLK. This or STERM* sampled around the rising edge of CPUCLK gives rise to the DSP termination signal, dspSRDY. Originally, this was timed very tightly, such that the DSP would sample read date around the next falling edge of CPUCLK, just like the 68030 does. As it turns out, that was pretty hard to manage. The final circuitry uses the DSP's data latch feature, running CPUCLK to the data latch input directly, to make the dspSRDY* timing much less critical.

The final PAL is U124, which as mentioned was added to help make up for a bug in the DMAC's DSP control register. All DSP control outputs are handled by this PAL. To get a chip select for it, the SCSI* chip select out of Gary is split based on A_7, and the register defined by this PAL is mapped as an 8-bit port (via DSACK0*) when SCSI* is adderted and A_7 is high. There are various DSP controls managed here. The CPU has global control over the DSP via the dspRST* line. It can mask CPU interrupt s caused by the DSP via the $dspMI_2$* and $dspMI_6$* lines. Finally, it can cause DSP interrupts at level 1 or level 0 via the $dspINT_1$* and $dspINT_0$* lines. Interrupts caused to the DSP by the CPU are cleared by the assertion of the corresponding IACK signal from the DSP. One restriction on this is mentioned in the next section.

1.2 DSP Control Requirements

Obviously a CPU or expansion based DSP card isn't going to have the DMAC around for part of a DSP control register, so I will call out the actual requirements of a proper DSP control register here. Additionally, AT&T has one new requirement that's not called out in their current documentation that should be mentioned here, even though it was not implemented on the prototype. Normally, when the CPU causes an interrupt to the DSP, that interrupt is cleared in hardware by the DSP's corresponding IACK singal. This hardware interrupt acknowledge is far more efficient than any software based scheme could be. However, there's a debugger used for DSP code that uses the high priority interrupt as some kind of tracing aid. For this to work there has to be some way to make the CPU setting of the $dspINT_1$* line sticky. To support this, I have added one new bit definition to the DSP control register. Normally, $dspINT_1$* works as it did in the prototype system, with clearing performed by $IACK_1$. But when the INT1STICK bit is asserted, the status of $IACK_1$ is ignored -- $dspINT_1$* will simply follow the setting of its register bit.

The bit assignments for the suggested DSP3210 control register are as follows:

Bit	Direction	Function
7	Read/Write	DSPRESET*. This bit comes up set low. When low, the DSP isin reset, when high, out of reset.
6	Read/Write	INT1STICK. This bit comes up set low. When low, IACK1 causes dspINT1* to automatically negate. When high, dspINT1* always follows the setting of the DSPINT1 register, ignoring IACK1.
5	Read/Write	MASK INT6*. This bit comes up set low. When low, INT6* fromthe DSP to the CPU is masked out, when high, the DSP may causea CPU level 6 interrupt.
4	Read/Write	MASK INT2*. This bit comes up set low. When low, INT2* fromthe DSP to the CPU is masked out, when high, the DSP may causea CPU level 2 interrupt.
3	Read Only	CPU INT6*. This bit reads low when the DSP is trying to cause alevel 6 CPU interrupt, high otherwise.
2	Read Only	CPU INT2*. This bit reads low when the DSP is trying to cause alevel 2 CPU interrupt, high otherwise.
1	Read/Write	DSP INT1*. This bit is written low when the CPU wants to cause alevel 1 DSP interrupt. It stays low until acknowledged by the DSP, for INT1STICK 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 alevel 0 DSP interrupt. It stays low until acknowledged by the DSP.

2. The AA3000 Development Implementation

2.1 PAL Equations

The latest PAL equations for the AA3000 prototype DSP system follow.

2.2 Schematics

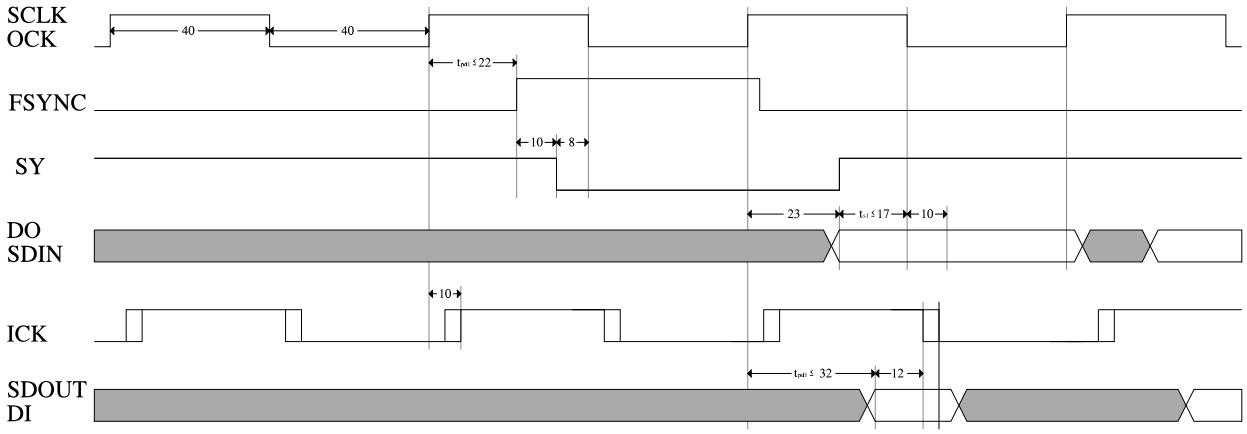
The final AA3000 Development System schematics follow.

3. Other Notes

3.1 CODECS

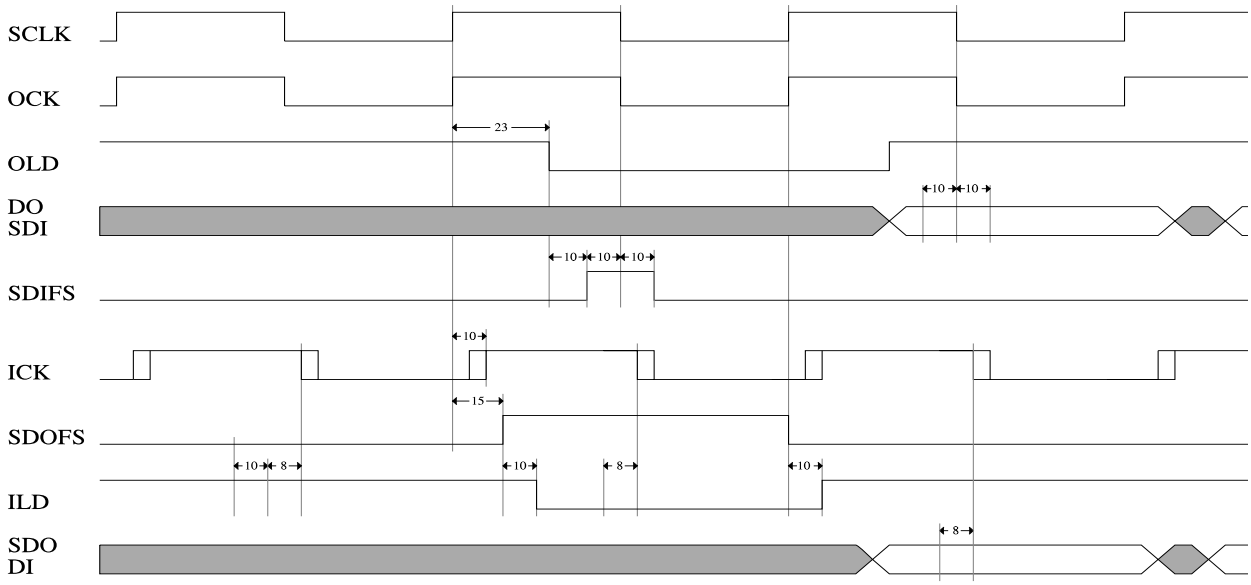
There are some suggested CODEC chips for use in the analog section of the DSP system, those selected for the A3000+ project. For audio sampling, the AD1849/CS4215 HiFi Stereo Audio CODECs are suggested. There's also a HiFi Stereo Audio CODEC from ITT, the ASCO 2300, worth consideration. Audio rate sampling should certainly support the CD frequency, 44.1kHz, directly, and should also consider AAA system 16-bit HiFi sampling rates. A preliminary serial port conversion diagram is shown below.

CS4215 to 3210 Serial Conversion Timing



For modem applications, the AD28msp01 Echo-Cancclling Modem Analog Front End is suggested. Telephone interfaces should consider the V22 and V32 standards for modem and the V27 and V29 standards for Fax. A preliminary serial port conversion diagram is shown below.

AS28msp01 to 3210 Serial Conversion Timing



3.2 DSP Audio Subsystem

There was no audio test system built for the working AA3000 DSP prototype. I did a design for the original A3000+ which is untested, but reproduced here anyway. We do have a demo board for both the ITT ASCO2300 and some set of Burr Brown chips which may eventually be useful for audio testing. I expect that any Processor slot implementation of a DSP card will use a DSP connector similar to that of the AA3000 and support its audio subsystem on a separate card. Also, some investigation of the NeXT DSP port would be reasonable, since its unlikely every possible DSP peripheral is going to fit on the DSP audio board we're likely to build.