"LUCAS" For the Amiga 1000

A 68020/68881 platform board

by Brad Fowles

Brad Fowles is a design engineer at Anakin Research, and is responsible for the hardware design of Anakin's Easyl drawing tablet. He has been involved with the Amiga since the machine's debut in 1985. Brad tells us that one of his guiding design principles for computer hardware is, 'Don't let out the magic smoke'. Be careful of this if you attempt this project.

Most of you in the Amiga community are well aware of the wonderful software available in the Public Domain. As a hardware type I have often been envious of the ease with which software can be shared among developers and users alike. Ideas and techniques can be distributed through BBS networks to the general benefit of all. In contrast, hardware developers lead a comparatively solitary existence, the exchange of ideas impeded by economic and logistical problems.

Can there be such a thing as Public Domain Hardware? Obviously no one can give away printed circuit boards, but perhaps we can do the next best thing: give away as much information as possible and make bare PCBs available for as close to cost as shipping allows.

The project is a platform board called LUCAS (Little Ugly Cheap Accelerator System) which replaces the 68000 in your Amiga 1000. LUCAS provides greater system performance and allows the use of the 68881 math coprocessor as well as an upgrade path to 32-bit wide memory.

The board has a 68020 and 68881 running at 16 MHz, and interface logic (consisting of four PALs, four discretes, a 16 MHz crystal, two SIP resistor paks, and some capacitors) to transpose 68020 cycles to 68000-like cycles. LUCAS also has a connector which will allow you, at a future date, to add 32 bit wide memory. (I'll try to get the fine people at *Transactor* to publish a memory board for this system in a few months.)

You can order the bare printed circuit board for $40, and the four-PAL chip set for $25 (see ordering information at the end of this article). The rest is readily available from local suppliers. The schematic and PAL equations are published here. Anyone who wants the film plots or Net lists so they can adapt the form factor to the Amiga 500 is welcome to them for whatever it costs me to get and ship them to you. (PCB design was done using P-Cad on er... an AT (...almost said the I word)).

If you own an Amiga 1000 and you would like to experiment with a 68020 and 68881 combination to improve performance, this may be the cheapest way to get there. Unfortunately, the chip set is going to cost you about $370.00 Canadian. Our aim is to make the rest as cheap as possible. You should be able to be up and running for under $475.00, or about three quarters of that if you live in the real world.

I decided to do this project for three reasons. One, I wanted one myself and couldn't afford the commercial versions. Two, some friends of mine who are using *Sculpt 3D* and *Animate* from Byte by Byte (both are available in 68020-68881 versions) needed more horsepower to render their images fast enough to actually make money at it. Three, I figured all of us Amiga 1000 owners out there with true hacker's hearts needed some light in our future since 1 meg of chip ram ain't.

When I started the design of this board, I used as a reference an article from EDN, January 9th 1986, pp 216-219. While looking at this design I became aware of an application note from Motorola AN944/D, *MC68020 and MC68881 Platform Board for Evaluation in a 16-bit System.* I recommend both these documents, especially the latter, if you wish a better understanding of how this board works. Unfortunately it is impossible within a short article such as this one to give more than a brief overview of how the board works. In the technical section of this article, I will try to highlight those aspects that are specific to the Amiga, but a thorough understanding will require some digging on your part. I also recommend the User's Manuals for the MC68020 and the MC68881 which are available from Motorola as "MC68020UM/AD" and "MC68881UM/AD" respectively.

Okay, Here is the disclaimer: If you get one of these bare boards and carefully put it together and then install it into your Amiga, you should have no problem and you'll be up and running in an evening or two. If you have problems, then it's up to your ingenuity to solve them. If you don't have some experience with a soldering iron, please, don't let this be your debut. I will gladly help anyone with problems. There are three ways you can get in touch with me: USENET at anakin@gpu.utcs.toronto.edu (Brad Fowles), BIX in the ANAKIN, AMIGA conference or by regular mail through *Transactor.* If you do manage to get my phone number you'd better be able to sweet talk me within 30 seconds. I hope that if there is sufficient interest out there that local user groups or individuals will add their help to anyone having problems. I have no objections should anyone get the bare boards and put them together and install them for a modest price, but please remember that the purpose of this is to make these available to end users as cheaply as possible.

If I haven't scared you off, please read on. If I have, well... so long, and thanks for all the fish.

Once you get one of the bare boards and procure all the parts, follow the enclosed instructions and carefully solder sockets for all the ICs and the crystal onto the board. Solder the resistor paks and the capacitors into place. Insert the 64 Pin header for the 68000 socket and solder it in.

Installation is quite simple but should be carefully done. Remove the plastic cover and the EMI shield from the Amiga base unit. On the right hand side of the PCB, just beside the Expansion connector, is the 68000 CPU. Gently pry the 68000 out of its socket, and store it on a piece of styrofoam somewhere safe. Now insert the LUCAS board into the 68000 socket, being careful to ensure that all 64 pins are correctly inserted into the socket. If you want to be really careful, remove the disk drive so you can see better. Watch the ribbon cable for the internal disk drive, as the bends in the cable can make things awkward. As long as you're careful and don't force anything, you should have no problem. You can do initial tests with the cover off, but once you're satisfied it's working put the base unit with its EMI shield back together again. That's all there is to it. Your heart can now resume normal operation.

You don't need to know a great deal about the inner workings of the LUCAS board to enjoy using it, but for those who would like a better understanding of the nature of running a 32-bit 68020 in a 16-bit 68000, read the section at the end of the article to get the key technical points.

Benchmarks

To give some idea of the performance improvement you can expect with the 68020-68881 pair, I have used four programs Al Aburto made available on the DEVCON disks distributed at the Washington Developers' Conference. These benchmarks were run on an Amiga 1000 with a 2 Meg Microbotics Starboard memory board and a Comspec 20 Meg Hard Disk. The operating system was Kickstart 1.21 and Workbench 1.3 Gamma 7. It should be noted that when the 68020-68881 pair is installed, the new IEEE math libraries which support the 68881 are used for floating-point transparently. I ran these benchmarks first with a standard 68000 and then with the LUCAS board.

```
Savage:
        68000 470.0 sec. Error -6.9e-7
        LUCAS   14.5 sec. Error -5.7e-4

Whetstone:
        68000   24 kwhets/sec.
        LUCAS 126 kwhets/sec.

Calcpi:
        68000 4.85 kflops/sec. Error -1.39e-11
        LUCAS 11.9 kflops/sec. Error -2.78e-11

Float:
        68000  10000 iterations  45.74 sec.
              256000 iterations 286.96 sec.
        LUCAS  10000 iterations  12.80 sec.
               256000 iterations 118.56 sec.
```

Of course, speed could be further enhanced by using inline F instructions for the floating point stuff, and even further enhanced by using 32-bit wide no-wait-state static memory.

Please remember that benchmarks are like political speeches; they only seem to make sense.

Software Considerations

Most software runs just fine on the 68020 but there are some programs which will guru on you. One of the major reasons for this is that on the 68020 all the instructions that are on the 68000 are implemented with the inevitable exception of one: the MOVE SR <ea> instruction. On the 68000 this is a user mode instruction; on the 68020 (and 68010 and later parts) it is a supervisor mode only instruction, i.e., if it's executed on a 68020 in an Amiga, you get a privilege violation guru.

If you're writing software, don't use this instruction; use instead the GetCC() library function which translates to a MOVE CC <ea> on the 68020, which is a valid user mode instruction. This function translates to a MOVE SR <ea> if there is a 68000 in the Amiga. This way you're safe both ways. (To run existing software that fails on the 020 for the above reason, you can solve the problem by first running a program called DeciGel that traps the offending instruction and does the right thing. DeciGel can be found on the *TransAmi* disk for this issue, or on Fish disk #18.

If you're one of those people who thought encoding information in the upper eight bits of the address field was a nifty idea ... Oh well, time to learn the error of your ways.

Of course, if you use any instructions from the 68020 superset then this code will never run on a standard Amiga. For further information, see section 21 of the Washington Amiga Developer Conference Notes, *Software Issues in 32-bit Amiga Systems* by Dave Haynie.

The new release of 1.3 has new IEEE Double Precision Math Libraries which take advantage of the 68020-68881 pair if it is present, and can immediately speed up any existing programs which use the math libraries.

If you want blindingly fast floating point, the best way is to recompile your code so that it uses direct inline F instructions. On the disk that comes with the PC board, you'll find programs called Mandslow and Mandfast. They are slight adaptations of RJ's original Mandelbrot program, adapted by Eric Haberfellner. Both programs are the same except that Mandslow was compiled for a standard Amiga, while Mandfast was compiled to use inline F instructions. Using Mandfast, a moderately deep Mandelbrot that runs in 1 hour 20 minutes on a standard Amiga runs in 4 minutes 20 seconds with the LUCAS board installed.

Compatibility

The LUCAS board works with all the expansion boards I have, but I'm sure there will be some out there that will bomb out. I will keep a list of those that do and those that don't and post it regularly on Usenet and BIX. The ones I have are the Comspec 20 Meg hard disk, Comspec 2 Meg Memory board, EASYL, and the Microbotics Starboard 2 Meg Memory board.

As a matter of interest only, the board works fairly well at 20 MHz, but periodically bombs. I have only 16 MHz parts; when I debug the bomb it seems to be the fault of the on-chip instruction cache. If you have 20 MHz parts, try it and let me know. Even if you have 16 MHz parts, it's worth the price of a 20 MHz crystal to see if it will work. Who knows? You might get lucky.

Conclusion

The performance of the Amiga 1000 with the LUCAS board installed will be improved, but it won't perform miracles. For general purpose computing, I find that compiles are only about 1.4 times faster, hardly worth the trouble. However, any program which uses floating point will be improved considerably, and those which have embedded F instructions will indeed appear miraculous.

On the other hand, the board does allow for 32-bit wide expansion memory, and if installed you can expect considerable general purpose performance improvements as well. I plan to design two boards: one with standard 100 or 120 ns DRAMs and a second with some high speed static RAM for no-wait-state operation at 16 MHz. You get most of the performance increase by having the memory 32 bits wide, but I can't resist seeing how fast it will go with no wait states at 16 MHz.

Stay tuned to *TransAmi* and the Nets for updates. Enjoy!

* * * * *

Brad is making the bare board and the PAL chips for this project available at a nominal cost: $40 for the board (which comes with a disk), and $25 for the four PALs. You can order directly from Brad at the following address:
RR #5 Caledon East
Ontario, Canada
L0N 1E0

Technical Discussion

Once the LUCAS board has been installed, we essentially have divided the CPU time into two discrete blocks: One, seemingly operating at 7.16 MHz and synchronous to the special purpose chips responsible for the video, sound, etc. and two, a 16 megahertz asynchronous system between the 68020 and 68881 and any possible 32-bit wide memory connected to the LUCAS bus.

The essential design criteria I used for the board were that it should be able to run asynchronously to the Amiga clock (so speeds of 16 MHz or greater could be achieved) and that there be no connection other than through the 68000 socket (to simplify installation.)

In order to achieve this, the board must look like a 68000 (4 clock standard bus cycle) running at 7.16 MHz when it is running its bus cycles, but when it is doing internal processing or talking to the MC68881 or future 32 bit wide expansion ram, it should run at the full 16 MHz (3 clock bus cycle).

90% of the problem in making this board work comes down to the problem of making the 68020 appear exactly like the original 68000 it replaces as it has been used architecturally in the Amiga, but able to go like stink when it gets the chance.

The address and data lines are easily implemented, as they are connected directly from the 68020 to the 68000 socket. Note that the 16 data bits are connected to data bits D16 through D31. The upper eight address bits on the 68020 are simply left unconnected.

I have used the * convention to indicate low true signals for ease in typesetting the article, i.e., *AS means AS is a low true signal. The PAL equations are written in CUPL format so I apologize to all you PALASM users.

68020 to 68000 Interface

The 68000 has an asynchonous bus structure. It asserts Address Strobe (*AS) to begin a bus cycle then waits for the assertion of *DTACK to end the cycle. This is usually 4 or 6 cycles, but may be held off by some peripheral device. The 68020 works much the same way except there are two *DTACK-like signals, *DSACK0 and *DSACK1. Because the 68020 can address in bytes (8 bits), words (16 bits) and longwords (32 bits) it must be able to differentiate between them. It does this by use of its dynamic bus sizing capability. A peripheral responds to a bus cycle by asserting one or both of the *DSACKx signals which tells the 020 the size of the transfer.

DSACK0	DSACK1	TRANSFER SIZE
0	0	32 bit transfer
1	0	16 bit transfer
0	1	8 bit transfer
1	1	Insert Wait States

Bus cycles on for the Amiga are always 16 bits wide so we will assert only *DSACK1 when responding to Amiga cycles. When we are running cycles for the 68881 (FPU) or 32-bit wide RAM on the LUCAS board expansion connector we must assert the appropriate *DSACKx combination.

In general terms with no wait states the 68000 will run a bus cycle in 4 clock cycles; the 020, however, will run the same bus cycle in 3 clock cycles. To correct this we must delay *AS and *DS (Data Strobe) from reaching the Amiga until after the rising edge of the S2 phase of the 7.16 MHz CPU clock. This is accomplished by the flip-flops U8a and U8b: inverting *AS from the 020 and using the complementary output with the flip-flop's reset tied to the inverted *AS will delay *AS the desired amount and terminate *AS20DLY when the *AS from the 020 terminates. This same technique is used for *DS. This creates the two timing signals *AS20DLY and *DS20DLY.

Byte addressability on the 68000 is accomplished by the Upper Data Strobe (*UDS) and the Lower Data Strobe (*LDS). The 020 has only a single Data Strobe (*DS). It uses a combination of the two SIZE pins and A0 and A1 to define the transfer pattern from the 020's internal multiplexer to the external data bus. (Note: bytes appear on data bits 24-31, words appear on data bits 16-31). It is therefore necessary for us to create *UDS and *LDS. This is accomplished by the following PAL equations. Note: The data strobes are not asserted during a Coprocessor cycle. (CPCS)

```
!UDS = (!DS20DLY) & (!A0)   & (CPCS)
!LDS = (!DS20DLY) & ( SIZ1) & (CPCS)
(!DS20DLY) & (!SIZ0) & (CPCS)
(!DS20DLY) & ( A0 ) & (CPCS)
```

The 68000 contains logic to support the 6800 family of products, and the Amiga uses this to interface to the 8250s. We must also emulate this interface as it is not present on the 020. A secondary clock called the E clock must be generated. It has a frequency of 1/10th the CPU clock and has a duty cycle of 60% low and 40% high. This is done by a decade counter in PAL U4. When running a 6800 family cycle the Amiga or peripheral generates a Valid Peripheral Address signal (*VPA). The 68000 then syncs itself with the E clock and issues a Valid Memory Address (*VMA) and ends the cycle on the falling edge of the E clock. The equation,

```
!Z3 = !QD #
QC #
QB #
QA ;
```

on PAL U4 in combination with the equation

```
!Z1.D = (DS20DLY) & (!Z1) #
(DS20DLY) & ( Z3) & (!VMA);
```

asserts *DSACK1 in the 9th state of E clock by the generation of the Z1 signal so that the long VPA, VMA cycle can be terminated correctly.

68020 to 68881 Interface

The MC68881 chip select (*CS) must be decoded from the 020. The 020 generates a 111 on the Function Code pins (CPU Space), a 0010 on the address lines A16-A19 which means this is a FPU coprocessor cycle, and a coprocessor ID on Address lines A13-A15. Since there is only one coprocessor in this design, A13-A15 are undecoded. The rest is decoded by PAL U4 in the following equation:

```
CPCS = (FC2) & (FC1) & (FC0) & (!A19) & (!A18)
 & (A17) & (!A16)
```

This generates the *CS (Chip Select) to the 68881.

Zen and the Art of Cycle Termination (An Asynchronous Synchronous Asynchronicity)

The generation of the *DSACK1 signal from the Amiga *DTACK caused me at times to doubt not only my own sanity but that of the universe itself. The *DTACK signal from the Amiga should appear and be sampled during the S4 phase of the clock cycle. Unfortunately it doesn't quite know that. It responds more or less correctly when it is talking to internal RAM but when external (fast) RAM is accessed, *DTACK comes back almost right away. Remember that *DTACK is the only way we have of determining the length of a cycle. We will cope with this anomaly in a moment.

Since the 020 is operating at 16 MHz - i.e., quite asynchronous to the Amiga clock - you have to sync up somewhere along the line with the Amiga 7.16 MHz clock. The ideal place to do this is when the two Amiga clocks C1 and C3 are in the condition C1 high and C3 low. These signals are not available at the processor and for a long time I had these two lines coming up off the motherboard. However the 7.16 MHz clock that is available at the processor can produce a reasonable facsimile. I divide the 7.16 MHz clock by two using U9a then logically OR it with the original 7.16 MHz clock and this turns out to have the same timing as C1 high and C3 low (my faith in the universe began to rekindle at this point.)

In the PAL equations this is DTPRELIM (DTack PRELIMinary). Now we have a reference point to sync back up with the Amiga.

In a saner world the combination of *DTACK and the Z1 signal (for termination of VPA, VMA cycles) would be sufficient to create the term SYSDSPRE1 (SYStem DSack1 PREliminary 1), but we have to delay till *DTPRELIM is true to sync up with the Amiga, plus cope with the quick response of *DTACK anomaly when talking to fast RAM And sync back up with the 16 MHz 68020 when we do finally issue a *DSACK1.

Confused? Wait! It gets better

Most dynamic memory boards, when connected to the Amiga expansion bus, will assert XRDY to hold off the assertion of

*DTACK while they do a refresh cycle. This puts in enough wait states so that the memory board can complete a refresh cycle. The problem is, soon as XRDY is asserted, a 20-30 ns glitch occurs in *DTACK, prompting the 020 to terminate the cycle before the data is even thinking about arriving on the bus. The solution is to avoid decoding it till the S4 phase of the Amiga 7.16 MHz clock. I delay *AS20DLY again for *DTQUAL and again for *DTQUAL1. DTQUAL becomes part of the *DTACK term and *DTQUAL1 is wired to QUAL (I needed the 7 ns across the PAL) then QUAL is added to the *DTACK term, giving:

```
!SYSDSPRE1 = !Z1#
(!DTACK)&(!AS20DLY)&(!DTQUAL)&(!QUAL)
```

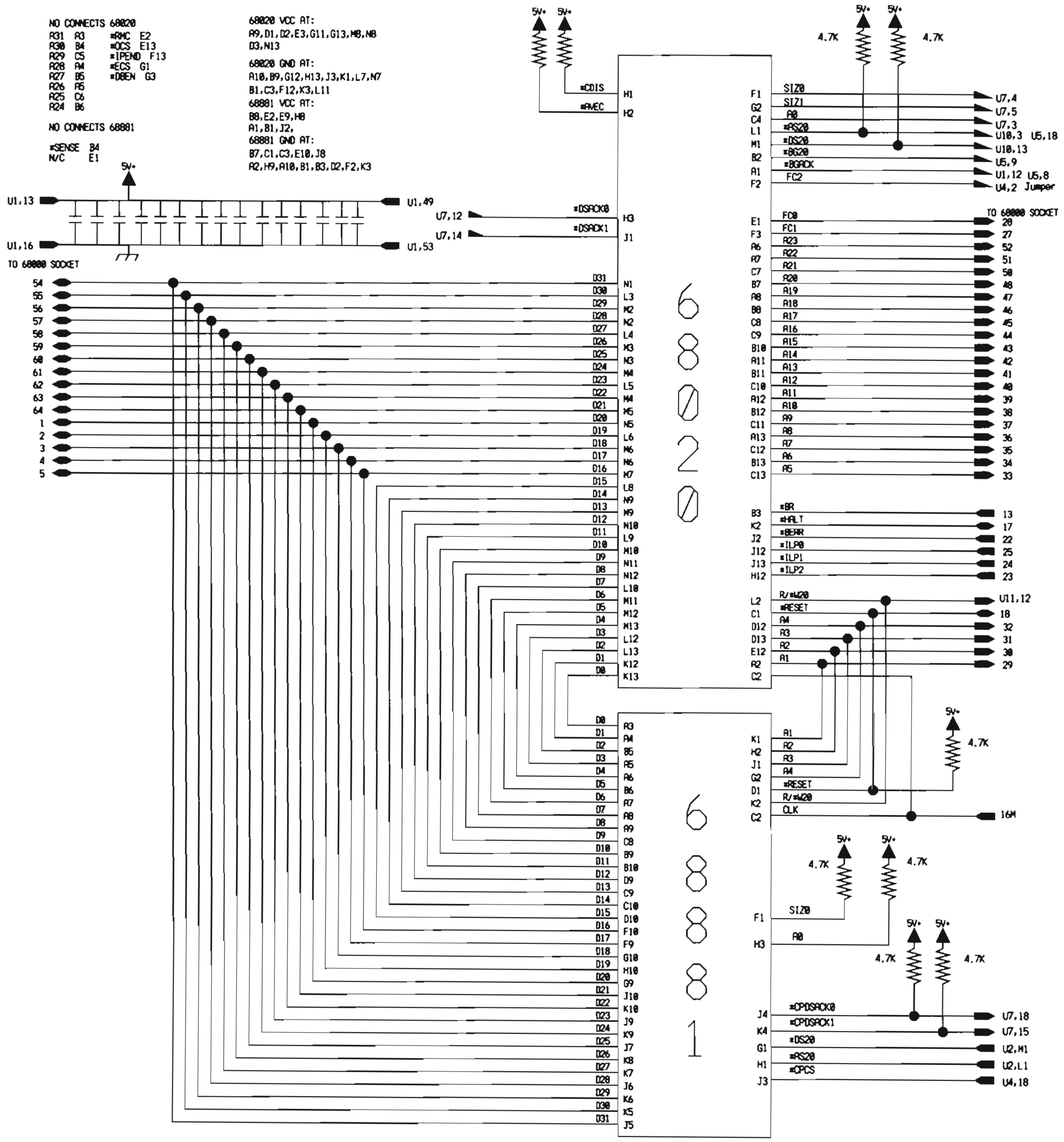
This solves the quick *DTACK problem. We buffer this (another 7 ns) by:

```
!BUFOUT = !SYSDSPRE1
```

Add in the Amiga syncronizing term

```
!DTTRIG = (!BUFOUT) & (!DTPRELIM)
```

Now we have an edge which is syncronous to the 7.16 MHz Amiga system. We then use this to trigger a Flip-Flop which has patiently been waiting for all this tomfoolery to end, and will ship !SYSDACK1 to yet another Flip-Flop to sync it back up to the 16 MHz 020 clock, and then to the awaiting PAL U7 for



additional decoding. I feed the Flip-Flop U9b with *ASDLY so that the !SYSDSACK1 signal will terminate when *AS does.

We're almost done.

PAL U7 then combines *SYSDSACK1 with the 68881 *DSACK1, CPDSACK1, and *SRDSACK1 (which comes from the expansion connector for future Static RAM), and finally and enthusiastically begets *DSACK1.

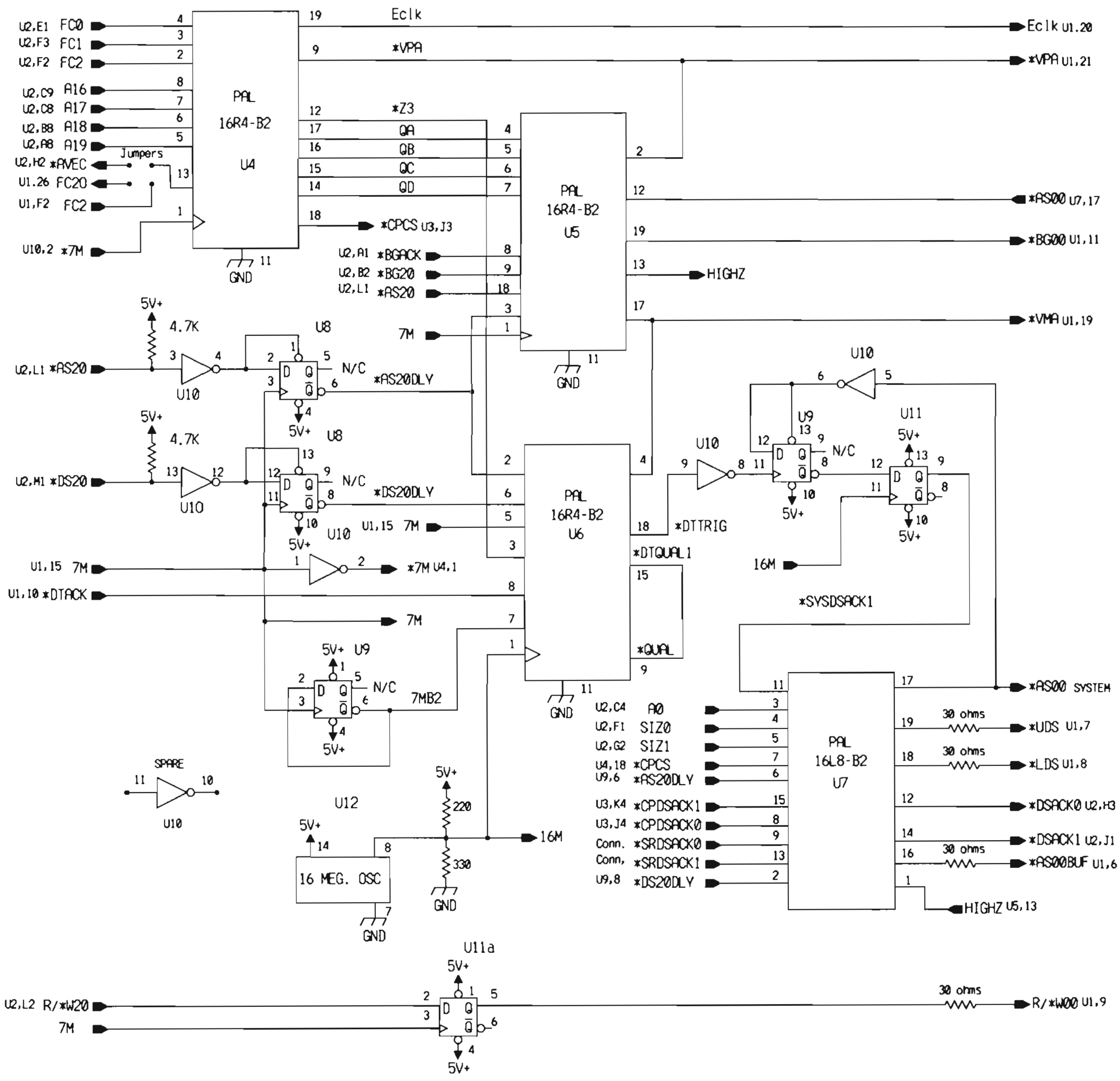
What could be simpler?

*DSACK0 is generated from the 68881 and from the future static RAM only.

Bus Arbitration

The Bus arbitration technique is quite similar to the 68000 with one exception. During coprocessor cycles the *AS is blocked from the 68000 bus. This gives rise to a possible problem. If the 68020 begins a coprocessor cycle with *AS blocked and responds to an alternate bus master's *BR (Bus Request) with a *BG (Bus Grant), the 68020 will assume the alternate bus master will wait for the negation of *AS. Unfortunately, *AS is blocked and therefore already negated. The result is bus contention. Therefore we must prevent the assertion of *BG until the interface negates *AS. This is done with the equation,

!BG00 = (!BG20) & (!Z2) & (AS20)



Listing: Equations for the four PAL chips in CUPL format

```
PARTNO      U4 ;
NAME        Finalu4 ;
REV         03 ;
DATE        May 23rd, 1988 ;
DESIGNER    Brad Fowles ;
COMPANY     Anakin ;
ASSEMBLY    Lucas ;
LOCATION    U4 ;

/* PAL16R4B2 */
/* PAL DESIGN SPECIFICATION  */
/* 68020-68881/68000 AMIGA INTERFACE */

PIN 1 = 7MN ;
PIN 2 = FC2I ;
PIN 3 = FC1 ;
PIN 4 = FC0 ;
PIN 5 = A19 ;
PIN 6 = A18 ;
PIN 7 = A17 ;
PIN 8 = A16 ;
PIN 9 = VPA ;
PIN 12 = Z3 ;
PIN 13 = FC20 ;
PIN 14 = QD ;
PIN 15 = QC ;
PIN 16 = QB ;
PIN 17 = QA ;
PIN 18 = CPCS ;
PIN 19 = E ;

!E  = (!QD )& (!QC )#
      (!QD )& (!QB );

!CPCS  = (FC2I )& (FC1 )& (FC0 )& (!A19 )& (!A18 )& (A17 )& (!A16 );

!FC20   = !FC2I ;

!Z3  = !QD #
        QC #
        QB #
        QA ;

!QA.D = QA ;

!QB.D = (!QB ) & (!QA )#
        ( QB ) & ( QA )#
        ( QD );

!QC.D = (!QC ) & (!QA )#
        (!QC ) & (!QB )#
        ( QC ) & ( QB ) & (QA );

!QD.D = (!QC ) & (!QD )#
        (!QB ) & ( QA )#
        (!QD ) & (!QA );

/*
DESCRIPTION: DECADE COUNTER,6800 CLOCK, AND COPROCESSOR SELECT LOGIC.
*/PARTNO      U5 ;
NAME        Finalu5 ;
REV         04 ;
DATE        May 23rd, 1988 ;
DESIGNER    Brad Fowles ;
COMPANY     Anakin ;
ASSEMBLY    Lucas ;
LOCATION    U5 ;

/* PAL16R4B2 */
/* PAL DESIGN SPECIFICATION  */
/* 68020-68881 /68000 AMIGA INTERFACE */
PIN 1 = 7M ;
PIN 2 = VPA ;
PIN 3 = AS20DLY ;
PIN 4 = QA ;
PIN 5 = QB ;
PIN 6 = QC ;
PIN 7 = QD ;
PIN 8 = BGACK ;
PIN 9 = BG20 ;
PIN 12 = AS00 ;
PIN 13 = HIGHZ ;
PIN 14 = Z2;
PIN 15 = BGACK2 ;
PIN 16 = BGACK1 ;
PIN 17 = VMA ;
PIN 18 = AS20 ;
PIN 19 = BG00 ;



HIGHZ   = (BGACK2 ) & (!AS20DLY )#
          (BGACK2 ) & ( BG20  );


!BG00   = (!BG20 )& (!Z2 )& (AS20 );


!BGACK1.D = !BGACK ;

!BGACK2.D = !BGACK1 ;


!Z2.D     = (!AS00 )#
            ( AS20 );

!VMA.D = (!QD )  & (!QC ) & (QB ) & (QA ) & (!VPA )#
         (!VMA ) & ( QD )#
         (!VMA ) & ( QC );


/*
DESCRIPTION: BUS ARBITRATION LOGIC AND VMA GENERATION
*/PARTNO      U6 ;
NAME        Finalu6 ;
REV         03 ;
DATE        May 23rd, 1988 ;
DESIGNER    Brad Fowles ;
COMPANY     Anakin ;
ASSEMBLY    Lucas ;
LOCATION    U6 ;

/* PAL16R4B2 */
/* PAL DESIGN SPECIFICATION */
/* 68020-68881 /68000 AMIGA INTERFACE */

PIN 1 = 16M ;
PIN 2 = AS20DLY ;
PIN 3 = Z3 ;
PIN 4 = VMA ;
PIN 5 = 7M ;
PIN 6 = DS20DLY ;
PIN 7 = 7MB2 ;
PIN 8 = DTACK ;
PIN 9 = QUAL ;
PIN 12 = DTPRELIM ;
PIN 13 = BUFOUT ;
PIN 14 = DTQUAL ;
PIN 15 = DTQUAL1 ;
PIN 16 = Z1 ;
PIN 18 = DTTRIG ;
PIN 19 = SYSDSPRE1 ;


!DTPRELIM = (!7M ) & ( !7MB2 ) ;

!DTTRIG = (!BUFOUT ) & (!DTPRELIM) ;
```

```
!SYSDSPRE1  = !Z1 #
       (!DTACK ) & (!AS20DLY ) & (!DTQUAL) & (!QUAL) ;


!Z1.D = (!DS20DLY ) & (!Z1 ) #
        (!DS20DLY ) & ( Z3 ) & (!VMA );

!DTQUAL.D = !AS20DLY ;

!DTQUAL1.D = !DTQUAL ;

!BUFOUT = !SYSDSPRE1 ;

/*
DESCRIPTION: SYSTEM DSACK1 GENERATION
*/PARTNO      U7 ;
NAME         Finalu7 ;
REV          05 ;
DATE         May 23rd, 1988 ;
DESIGNER     Brad Fowles ;
COMPANY      Anakin  ;
ASSEMBLY     Lucas ;
LOCATION     U7 ;

/* PAL1618B2 */
/* PAL DESIGN SPECIFICATION */
/* 68020-68881 /68000 AMIGA INTERFACE */

PIN 1 = HIGHZ ;
PIN 2 = DS20DLY ;
PIN 3 = A0 ;
PIN 4 = SIZ0 ;
PIN 5 = SIZ1 ;
PIN 6 = AS20DLY ;
PIN 7 = CPCS ;
PIN 8 = CPDSACK0 ;
PIN 9 = SRDSACK0 ;
PIN 11 = SYSDSACK1 ;
PIN 12 = DSACK0 ;
PIN 13 = SRDSACK1 ;
PIN 14 = DSACK1 ;
PIN 15 = CPDSACK1 ;
PIN 16 = AS00BUF ;
PIN 17 = AS00 ;
PIN 18 = LDS ;
PIN 19 = UDS ;


AS00.OE = HIGHZ ;
!AS00   = (CPCS )& (!AS20DLY );

AS00BUF.OE = HIGHZ ;
!AS00BUF   = (CPCS )& (!AS20DLY );

UDS.OE = HIGHZ ;
!UDS  = (!DS20DLY )& (!A0 ) & (CPCS) ;

LDS.OE = HIGHZ ;
!LDS  = ( !DS20DLY ) & ( SIZ1 ) & (CPCS) #
        ( !DS20DLY ) & (!SIZ0 ) & (CPCS) #
        ( !DS20DLY ) & ( A0 ) & (CPCS) ;

!DSACK1 = (!SRDSACK1 )& (!AS20DLY )#
          (!CPDSACK1 )& (!AS20DLY )#
          (!SYSDSACK1)& (!AS20DLY ) ;

!DSACK0 = (!SRDSACK0 )& (!AS20DLY )#
          (!CPDSACK0 )& (!AS20DLY ) ;

/*
DESCRIPTION: ADDRESS STROBE, UPPER AND LOWER DATA STROBE AND
             FINAL DSACKX GENERATION
*/
```

ZING!®

ZING! - Mouse Driven CLI - $79.95
ZING!Keys - Macro program! - $49.95
The Demonstrator - Create Tutorials $39.95

NEW :
ZING!Spell - Check & Correct your Spelling as you Type; Batch Checking too; ARexx interface; Word expansion; - $79.95
Cordless Mouse for the AMIGA - $179.95
Touchscreen for the AMIGA - CALL.

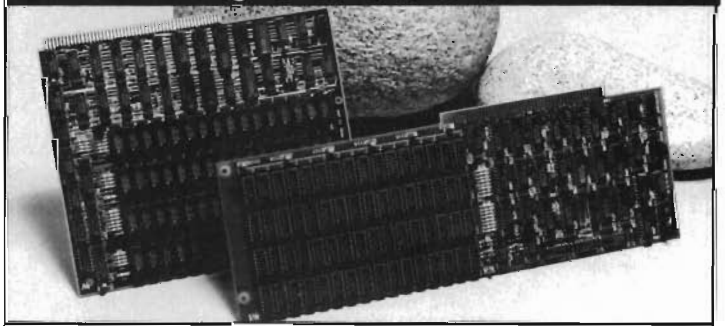
Find out about our **Trade-UP** policy which will save you up to **50%**. Write or call today and receive more info on all our products.

MERIDIAN™ SOFTWARE INC.
9361 W. Brittany Avenue
Littleton, CO 80123

(303) 979-4140
Credit Cards and Dealer Inquiries Welcome!

AMIGA is a registered trademark of Commodore-AMIGA, Inc.
ZING! is a registered trademark of Meridian Software, Inc.

Q U A L I T Y



8 megabytes for the Amiga!

ASDG designs and builds in quality from the start. Our 8 meg board comes with 0, 2, 4, 6, or 8 meg installed. Of course it's no-wait-state memory, of course it's fully auto-configuring, and comes in your choice of the Amiga® 1000 or Amiga® 2000 form factor. You wouldn't expect less from ASDG.

We back up our quality with a full 18 month warranty. With memory boards from ½ meg to 8 meg ASDG gives you more choices for memory expansion. For more information contact ASDG at
(608) 273-6585
925 Stewart Street
Madison, Wisconsin 53713

ASDG Inc.
(608) 273-6585

P E R F O R M A N C E