BIGRAM 8/32

A Complete Zorro III PIC
Design Example

Document Revision 1.10

1991 DevCon Release

by Dave Haynie
July 18, 1991

IMPORTANT INFORMATION

"We don't know a millionth of one percent about anything."
-Thomas Alva Edison

This Document Contains Preliminary Information

The information contained here, while a honest attempt to illustrate a good Zorro III card design, is still preliminary in nature and subject to possible errors and omissions. We don't expect any problems with this design, but it's only responsible to supply you with this caveat.

Commodore Technology reserves the right to correct any mistake, error, omission, or viscious lie. Corrections will be published as updates to this document, which will be released as necessary in as developer-friendly a manner as possible. Revisions will be tracked via the revison number that appears on the front cover.

TABLE OF CONTENTS

TABLES AND FIGURES

CHAPTER 1

INTRODUCTION

"The curtain rises on a vast primitive wasteland,
not unlike certain parts of New Jersey."
-Woody Allen

This document fully describes an example Zorro III Plug-In-Card (PIC) design for a simple asynchronous dynamic RAM memory card. Its intent is to describe the procedures and underlying theories behind a basic Zorro III design. However, it is not a Zorro III designer's bible or any such rulebook. It should provide the designer with a better understanding of Zorro III PIC design, and perhaps provide a starting point for the beginning Amiga peripheral designer.

1.1 Intended Audience

This document was written primarily for hardware engineers interested in designing Plug- In-Cards for the Zorro III expansion bus. A reasonable level of microcomputer knowledge is a prerequisite to get much meaning out of these pages. A good understanding of the Zorro III bus theory, as outlined in *The Zorro III Bus Specification* (available from Commodore), is essential. Knowledge of basic TTL digital design with standard MSI and PAL devices is required, as is an understanding of dynamic RAMs. Familiarity with the Motorola 680x0 processors will also be quite useful.

While knowledge of Zorro II PIC design will also be useful, such experience mainly applies to the AUTOCONFIG® sections of a PIC design. The signals and design problems for the Zorro III bus are substantially different than for Zorro II. Zorro III PICs are expected to run considerably faster than those for Zorro II, leading the circuit designer to faster TTL logic families and more use of fast PAL devices. The additional speeds coupled with 32-bit buses will also lead the circuit board designer to multi-layer boards and more critical routing problems. While the Zorro II bus and most Zorro II designs are mainly synchronous, the Zorro III bus is asynchronous. Zorro III designs will typically be either fully asynchronous or self-clocked synchronous with proper attention to stable synchronization with the bus.

1.2 A Few Words About AUTOCONFIG®

If past history is any indication, the first thing to mention about Zorro III PIC design is AUTOCONFIG®, the Amiga mechanism for linking hardware plug-ins with software such that configuration jumpers for addresses are unnecessary, and device driver installation is trivial to even a novice user. And the first thing to say to a hardware designer about AUTOCONFIG® is *Don't Panic*. More than any other issue, the AUTOCONFIG® system seems to have confused Zorro II PIC designers. But there's absolutely nothing to fear about AUTOCONFIG®; it is a very simple concept and very simple to implement as an integral part of any PIC's design.

The concept of configuration hasn't changed for Zorro III, and the implementation is very much the same as for the Zorro II bus. Extensions have been provided for a few Zorro III advanced features, and a few extra things were added to the specification to make the design of a 32 bit PIC as easy as possible. Other than that, if you know Zorro II configuration, you'll pick up Zorro III configuration almost instantly. Chapter 2 walks through the creation of an AUTOCONFIG® circuit for Zorro III and discusses the basic logic likely to be in place on any Zorro III card.

1.3 Design Example Goals

The goal of this example is to design a memory card for the Zorro III bus. While A3000 users won't be running out of motherboard memory (up to 18 Megabytes) quite as fast as A2000 users did, there's already an emerging need for massive memory in Amiga computers. This RAM card meets the following goals:

- Provides a fully asynchronous design example
- Uses the same ZIP memories as the A3000
- Supports up to 8 Megabytes using 256K x 4 DRAMs, up to 32 Megabytes with 1M x 4 DRAMs.
- Hopefully functions as a realtively clear design example

And, of course, this is a fully functional design tested to the best of our ability at the time of this writing.

CHAPTER 2

AUTOCONFIG® LOGIC DESIGN

"Logic is in the eye of the logician."
-Gloria Steinem

Every PIC design has a few things in common, most noticably an AUTOCONFIG® circuit. While such logic can pretty much be created by rote, an optimal design always will incorporate the AUTOCONFIG® and other Zorro III bus logic naturally into the main design. While this chapter concentrates on the AUTOCONFIG® logic, it will cover all of the standard logic elements of any Zorro III design in a sensible order.

Throughout this and the following chapters, references to the schematic pages in Appendix 2 will be. Page one of the schematics is found on page A-13 of this document, and there are six schematic pages. To make things simpler, these will be referred to as S-1 through S-6.

2.1 Bus Buffers

Just like with Zorro II, all Zorro III designs require a number of buffers on the bus logic signals. No PIC may load any bus signal with more than two F-series equivalent gates, and of course outputs from the PIC must be able to drive the bus properly. Any unbuffered signal used by a PIC mut be used close to the bus connector; if a signal trace is longer than a few inches, it must be buffered. In addition, due to the dynamic nature of the high-order Zorro III address lines, some or all of these address lines must be latched for the duration of the bus cycle.

The buffering/latching arrangement is shown on S-1. Since this is a slave-only board, address lines are input-only. Addresses A_{31}-A_8 are transparently latched by 74F373 parts, the latch taking place when /FCS is negated. The transparent latching allows the address comparator to take advantage of the bus's address setup time, important for matching to the board's assigned address as quickly as possible. The circuitry shown here is the most straightforward, but in operation, only A_{24}-A_2 are actually used once the board select is determined. Thus, a fast enough comparator circuit can latch an address match rather than the high-order addresses if it saves on circuit complexity. Since the low order addresses A7-A2 are static, they are simply buffered coming into the RAM board. The extra buffers in that package are used in this design to buffer /FCS and READ, two lines used in several places in this design.

Data buffering is quite simple; D_{31}-D_0 are buffered with bidirectional bus buffers. The data direction and buffer enable signals are quite simple. The buffers point out toward the bus for read cycles when the PIC is selected (/SLAVE asserted), in at all other times; this function is contained in the U200 PAL. The output enable is asserted when the PIC is selected, the DOE signal is asserted, and there's no bus error; this function is contained in the U201 PAL. Because the data bus tristates, I use centering resistors to keep it quiet when it's not being driven. If this design had been supporting Zorro II as well as Zorro III, an additional two data buffers and much more complicated buffering logic, based on the SENSEZ3 line, would be required.

2.2 The AUTOCONFIG® ROM

Reg	Bit	Val	Description
00	7,6	10	This indicates a Zorro III card.
	5	1	The OS will link this as free memory.
	4	0	No autoboot/diagnostic ROM.
	3	0	Only one logical PIC here.
	2-0	001	Using the extended size feature, this is a 32 megabyte board.
04	7-0	01010111	Commodore Product $53.
08	7	1	Hint to the OS that this is memory, not I/O
	6	0	This board can be shut up.
	5	1	Extended sizing being used here.
	4	1	This must be 1, for 1.3 compatibility.
	3-0	0001	Let the OS calculate the logical size of the memory.
0C	7-0	00000000	Reserved.
10	7-0	00000010	Manufacturer's number, high byte.
14	7-0	00000010	Manufacturer's number, low byte. Since this one is a Commodore board, it uses the Commodore number.
18-3C	7-0	00000000	All of these are zeroed. This board does not contain a board serial number or boot/diagnostic ROM.
40	7-0	N/A	Reserved
44	15-0	CFGADDR	This board uses the Zorro III configuration block. It accepts the configuration address as a single write.
48	7-0	N/A	Configuration is completely handled with register 44.
4C	7-0	N/A	A write of any value will cause the board to shut up.
50-7C	7-0	N/A	All remaining registers are reserved.

Table 2-1: Logical AUTOCONFIG® Registers

The complete AUTOCONFIG® ROM is implemented in PAL U200, shown on schematic page S-2. The design of an AUTOCONFIG® ROM is usually very simple, but it does require a complete understanding of how the board is to be used by the system before it can be done. Also, a Zorro III configuration ROM is similar to a Zorro II configuration ROM, with just a few more options available, once the translation for the configuration space chosen is applied.

First of all, the board must be described. Obviously, this is a Zorro III memory board, and since it's my design, it's also from Commodore. On top of that, it can be expanded up to 32 megabytes, and it can also be "shut up" if necessary. That's pretty much the specification, now it has to be translated into Zorro III ROM registers. *The Zorro III Bus Specification* describes these entries starting on page 8-1. The logical register assignments are illustrated in *Table 2-1*. The table actually lists **all of the configuration registers on the board (registers 40-7C** are reserved as write registers, not read registers, but they're mentioned here anyway).

The next step in the design process is to convert these bit assignments to actual logic. As mentioned before, the configuration ROM is implemented as part of the U200 PAL. By design, configuration ROMs fit nicely in a PAL in most cases. The Zorro II and Zorro III specifications call for all read resgisters other than register **00** to be inverted in their physical implementation. Since most bits are logically "0", they'll be physically "1", and "1" is the default output state of a standard PAL. Also taking into account that each logical register is actually made up of two physical registers, both of which assert data only on the D_{31}-D_{28} nybble, the physical register mapping for all read registers is shown in *Table 2-2*. The actual PAL equations for this are on

Address	D_{31}	D_{30}	D_{29}	D_{28}
00	1	0	1	0
02	0	0	0	1
04	0	1	1	0
06	1	1	0	1
08	0	1	0	0
0A	1	1	1	0
12	1	1	0	1
16	1	1	0	1
OTHERS	1	1	1	1

Table 2-2: Physical ROM Registers

page A-3. These are simply a set of equations, one for each data line, that take into account each "0" in the above table, and are active only when the board is selected and not yet configured.

While it makes no difference to the equations for our ROM registers, it is a good idea to point out here the differences in addressing these read registers. Zorro II boards must respond to the configuration space $00E8xxxx, and all registers are mapped on word boundaries. Zorro III boards can respond to the $00E8xxxx address as a 16-bit Zorro II device as well, but many designs, including this one, will choose instead to respond to the Zorro III configuration space at $FF00xxxx. A board responds to this address as a 32-bit device, and it actually need only decode the high-order eight bits of this address; both of these facts can save considerably on the amount of configuration logic necessary for some designs. In both configurations, the first nybble of each register pair is at the offset from base address given by that register number. In the Zorro II space, the second nybble is in the next logical word -- the register number plus two. Zorro III instead maps the second register of the pair at $100 plus the register number. This may sound like the two will be quite different in implementation, but as the example PAL U200 illustrates, if I map A_8 as A_1 in the equations, all ROM equations will be written the same for either configuration space. Using this feature and a multiplex of A_8 and A_1 based on the SENSEZ3 signal can help simplify the design of a card that adjusts to both Zorro II and Zorro III buses.

2.3 The AUTOCONFIG® Registers

This design supports two writable configuration registers, the 16-bit configuration address register **44** and the shutup register **4C**. Recall that configuration address registers are written in a pattern that allows the designer to choose nybble- or byte-wide configuration latches for Zorro II configuration space or byte- or word-wide configuration latches in Zorro III configuration space. Since Zorro II space is only sixteen bits wide and writes must line up consistently, this design would have to latch configuration address bits A_{31}-A_{24} on a write to register **44**, followed by configuration address bits A_{23}-A_{16} on a write to register **48**. Even though a large board such as this never needs to look at A_{23}-A_{16} for its configuration address (Zorro III PICs always live at their natural boundaries), a board configured in Zorro II configuration space isn't configured until a write to register **48**. Since this board instead responds to Zorro III configuration space, the entire sixteen bit configuration address can be written at once with a write to register **44**, and that is also the signal indicating that configuration of the board is complete.

The register logic starts with the same PAL, U200, as used for our ROM logic. This PAL has the important low-order addresses going to it, so it's a natural for this. In this design, there are two signals created for register support in PAL U200. The first of these is a signal called /PRECON, for pre-configuration. The board isn't fully configured until the end of the Zorro III cycle that writes either register **44** or register **4C**; /PRECON is asserted during this last write cycle as soon as data is valid on the bus, and it stays latched until the next reset. The other signal in U200 that's of immediate importance is the CFGLT signal. This line is responsible for latching the configuration address on the bus if this final write is a configuration and not a "shut up" request. This is an active high signal in an inverted-output PAL, so the equation can't be very complicated. This line is asserted when the board is selected, /PRECON is asserted, and A_3 is low, which is true just after /PRECON is asserted for a write to **44**. Like the /PRECON line, CFGLT latches until the next reset. The remainder of the register logic is elsewhere.

The rest of the configuration control logic is in PAL U201, which creates both the /CFGOUT and /SLAVE signals, two signals that must be driven out to the backplane. The /CFGOUT signal is pretty simple. Normally, it is asserted at the end of a cycle in which /PRECON and /CFGIN are asserted, and latched asserted as long as PRECON also stays asserted. It also gets asserted if /CFGIN is asserted along with the SENSEZ3 signal negated. This latter condition indicates that the board has been placed in a Zorro II backplane. This board can't support Zorro II configuration, so it automatically "shuts up", an action required by the Zorro III specification. Note that the SENSEZ3 signal is called /Z2SHUNT in the PAL equations on page A-5.

The next basic piece of the configuration logic is the configuration latch, which in this case is the 74F374 at U202. This edge-triggered latch is triggered by the rising edge of CFGLT, which is asserted when the board's configuration address is written and data is valid on the data bus. At the end of the configuration address cycle, /CFGOUT is asserted, the address as latched is now fed into the /SLAVE generation address comparator, and the board is fully configured in hardware. Since this is an autosized memory board, system software generally will calculate its size and link it into the free memory pool before the next board is configured, though this operation can of course change as the configuration software changes.

2.4 The SLAVE Logic

Naturally, this brings up the question of how the /SLAVE logic is implemented. Every Zorro II or Zorro III board must assert its private /SLAVE line when it is responding to a bus address. In every case, two addresses must be supported; the configuration space address prior to configuration, and the software-assigned address after configuration. The method used in this example is quite similar to techniques used in many Zorro II designs, and is only slightly more complex.

The core of a /SLAVE circuit is always an address comparator of some kind. In every case, the bus address must be compared with the address to which the board responds. The main comparator in this circuit is the 74F521 at U203. It compares seven bits of possibly-latched bus address, A_{31}-A_{25}, with the corresponding bits on the configuration address latch. This comparison is called /MATCH on the schematics. Prior to configuration, the 74F374 is tri-stated, and the outputs going to the comparator are all pulled high, getting the card well on the way to responding to the $FF00xxxx configuration space.

The twist in this design is that there is a bit more to this comparison than just a simple comparator can handle. First of all, the board needs to look at a full eight bits of the $FF00xxxx address to properly respond during configuration, but only seven bits of address once the board is configured as a 32-megabyte board. This PAL U201 helps out by requiring A_{24} tobe high for a /SLAVE response prior to configuration. Zorro III memory cards must monitor the function codes FC_0-FC_2. PICs must only respond to a valid User or Supervisor mode Code or Data space access; such accesses are given as the exclusive-or of FC_0 with FC_1. The /SLAVE signal is always qualified with the Zorro III full cycle strobe /FCS, and it can occur in only two cases. In the first case, a qualified match occurs, the board is unconfigured, and /CFGIN is asserted. In the latter case, a qualified match occurs, the board is configured, and CFGLT is asserted. As previously mentioned, if the board is configured but CFGLT is negated, the board has been "shut up" rather than configured.

And that is all there is to the basic configuration logic. As demonstrated with U201, it is usually quite reasonable to incorporate this logic in with other board logic, where it'll fit the most efficiently. AUTOCONFIG® logic is intended to make it easy on the designer as well as the user; it's not supposed to scare anyone.

CHAPTER 3

MEMORY SYSTEM DESIGN

"I like them big and stupid."

-Julie Brown

This chapter discusses the actual DRAM logic design for this project. The information here is going to be far less useful for the designer trying to learn about Zorro III designs, since this is the part of the board design which is very specific to the task at hand, a DRAM board. However, there may be some ideas of a more general applicability. If nothing else, it shows that a fully asynchronous DRAM design can be done rather simply with a little planning.

3.1 DRAM Refresh

The most complex part of any hand-made DRAM circuitry is very likely to be the refresh circuitry. Without refresh, DRAM would look pretty much like static memory with a multiplexed address bus (and the folks at TI and National Semiconductor would be selling quite a few less DRAM controllers). While there's nothing wrong with off-the-shelf DRAM controllers, it's really not very difficult to "roll your own".

3.1.1 Refresh Arbitration

If you beilive that DRAM boards are difficult to design, and that refresh is the most difficult part of such a design, then you must believe that refresh arbitration is the most difficult part of the refresh logic. So I'll discuss that part first, and the rest of the circuitry in this chapter will then be simpler.

In this design the refresh arbiter is incorporated with the /SLAVE generator. Refresh arbitration takes place in U201 and is by the nature of the Zorro III bus necessarily asynchronous. A refresh request can come in at any time, and must be serviced as soon as possible without interrupting a cycle. There are three refresh cases: a request outside of a cycle, a request during a cycle to this card, and a request during a cycle to another card. These cases are illustrated in *Figure 3-1*. The problem with any asynchronous refresh arbiter is that it's impossible to determine at a single point if a cycle is starting or not. This can be though of as a potential race condition between the refresh request and the start-of-cycle. So the solution is to create two sampling points, one to give the go-ahead for a refresh cycle, the other to give the go-ahead for a memory cycle.

For the latter, you can use the /SLAVE signal. Virtually everything that happens on a simple Zorro III slave card is gated with /SLAVE. So in order to safely arbitrate refresh, we generate a refresh acknowledge signal, /REFACK, which will always be asserted safely before or safely after /SLAVE. In order to get there before /SLAVE, the /REFACK line will not be asserted outside of a Full Cycle if the /MATCH line is asserted. Since /MATCH and /FCS must both be asserted in order to create /SLAVE, and /FCS always follows /SLAVE, the /REFACK line is guaranteed to get out of U201 prior to /SLAVE, should the refresh request some in just before the PIC is selected. But once a board is selected on the bus, there's no reason to hold off refresh if it's a different board being selected, so /REFACK can be asserted during the data time of some other card's full bus cycle. In either case, the refresh acknowledge is latched as long as refresh request is held.

3.1.2 Refresh Counter

A simple refresh counter is implemented in PAL U306. Although the board supports both 256K x 4 or 1 Meg x 4 DRAM, the actual per-row refresh time is the same; the former part requires a 512 row refresh in 8ms, the latter a 1024 row refresh in 16ms. This amounts to one refresh request every 15,625ns. However, to build in support for burst mode with page-mode or static column DRAM, we use the $T_{RAS,MAX}$ time here, which is 10,000ns. The PAL counter actually counts 140ns clocks, so a count of 71 clocks will get us up to 9,940ns, close to the desired 10,000ns. If burst mode support weren't considered here, a count of 111 clocks could be used in the counter.

The counting is quite simple; the counter goes from zero to its terminal count, then asserts the /REFREQ signal. It then holds onto the /REFREQ signal until a refresh cycle is under way, as indicated by /REFCYC. The /REFCYC line will reset the counter for the duration of the refresh cycle. The process starts over once the refresh cycle is complete.

The clocked counter is used here simply because it's very easy to understand and, being fully digital, always works the same way. It could have been a simple one-shot or 555 timer circuit, as long as component tolerances don't allow the timer to drop below the required refresh frequency. You may recall reading of the evils of such timers in DRAM hint books. While they aren't optimal, due to the aforementioned component tolerance problems, that's not why you were warned off. The main reason for avoiding such timers in most DRAM designs is the problem you're likely to have with an asynchronous refresh request. Since we have already solved the problem of the asynchronous refresh request here, no asynchronous approach is inherently evil to this design.

3.1.3 Refresh Cycle

The actual refresh cycle, illustrated in *Figure 3-2*, is a CAS-before-RAS refresh, and all memory on board is refreshed at the same time. As soon as a refresh cycle is active (/REFCYC asserted), PAL U300 will assert the /REFCAS line. /REFCAS will in turn cause the CAS control PAL, U304, to drive all eight CAS lines. An active /CASDEL or /RASDEL will hold off the assertion of /REFCAS, thus ensuring RAS precharge (T_{RP}) in case the refresh is immediately following a memory cycle. The /REFCAS line is latched by /MUX until /RASEN comes along, so that it's no longer dependent on /REFACK. The /REFACK line will be negated some time before the end of the CAS-RAS cycle; its main use here is to qualify the start of a refresh cycle. Once the /RASEN is asserted, /REFCAS is latched by the negated /RASDEL, as is /RASEN.

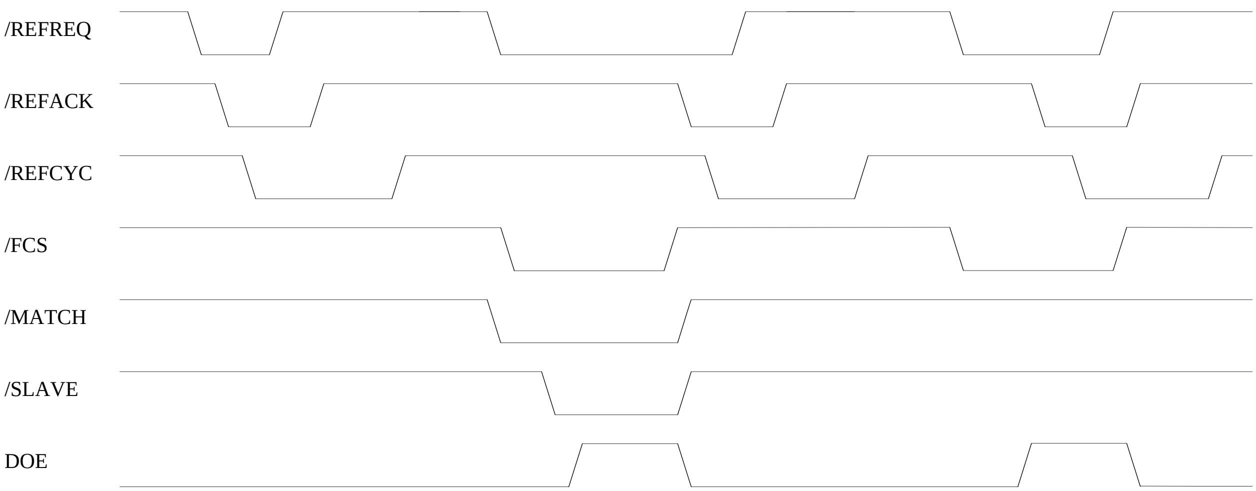


Figure 3-1: Refresh Arbitration

The /CASOUT line of U300 is also driven at the start of the refresh cycle. This of course comes back to U300 as the /CASDEL signal. The refresh /RASEN is driven as soon as /CASDEL is asserted, thereby separating refresh CAS and refresh RAS by roughly the CAS delay time. The /RASEN line drives the buffered /RAS lines to either bank of memory. Once asserted, /RASEN is held until /RASDEL wraps back in. The refresh cycle is held until /RASDEL once again is negated, thus ensuring T_{RP} for the refresh cycle, in the event that this refresh is taking place right before a memory cycle.

3.2 DRAM Access

The DRAM read/write access duing a normal memory cycle uses mainly the same parts for RAS-CAS controlling, along with a few additional bits and pieces to control memory banking. The logic supports several speeds of DRAM, selection being made via jumpers on the tap delays used for RAS and CAS timing. Either the 256K x 4 or 1 Meg x 4 parts can be used, and a jumper is provided to allow the necessary banking modification for this. Finally, Zorro III burst-mode (Multiple Transfer Cycle) support of either page or static column DRAMs is provided for. The design has been tested with a Level 2 Buster and fast-page DRAMs. While static column memories have not been tested, they should present no problem.

3.2.1 Memory Cycle

The basic memory cycle is started by U300 when /SLAVE is asserted and no refresh cycle is acknowledged or in progress. If burst is enabled, /MTCR is driven along with /SLAVE at the cycle's start. The memory cycle will be held off until /RASDEL is negated, to ensure T_{RP} after a refresh or a previous memory cycle. The assertion of /RASEN starts the cycle, and /RASEN is held through the end of the cycle. For fast-page memories, we could drop /RAS before cycle's end lets us gets an early start on RAS precharge. Since /RASEN creates /MUX, however, this optimization couldn't be used for SCRAM parts -- that could result in a column address change before cycle's end (SCRAMs don't latch the column address). Also, /RAS needs to be held through multiple /DTACKs in burst mode, so in this implementation the short /RASEN optimization is not done, though it's something to consider as an improvement. The 100ns tap delay U301 sets the RAS delay, and J300 provides taps for 100ns, 80ns, and 60ns DRAM. The /RASEN line is buffered, as previously mentioned, by two gates from the 74F244 at U303, one creating $/RAS_L$ for the lower bank of 32 memories, the other creating /RAS$_H$ for the upper bank of 32 memories. U303 also buffers the first tap from U301, which becomes /MUX, the line used for multiplexing the DRAM addresses.

The U300 PAL also creates the enable for CAS, the /CASEN line. This is based on /RASEN, DOE, and /MUX asserted, and it’s held through the end of the cycle, until /DTACK is negated. The /CASEN line qualifies CAS, but it doesn’t necessaily start CAS for a full cycle; further consideration of CAS generation is done elsewhere. For fast-page mode operation during a burst cycle, /CASEN follows /MTCR to generate a new CAS for each cycle. For static column operation during a burst cycle, /CASEN is simply held asserted until the cycle’s end.

Most of the CAS generation is handled in U304, the CAS generation PAL. The CAS strobes are used to select between two banks of DRAM, and to select the appropriate bytes to access during write cycles; this is covered in detail in the next section. Other than qualifiying by

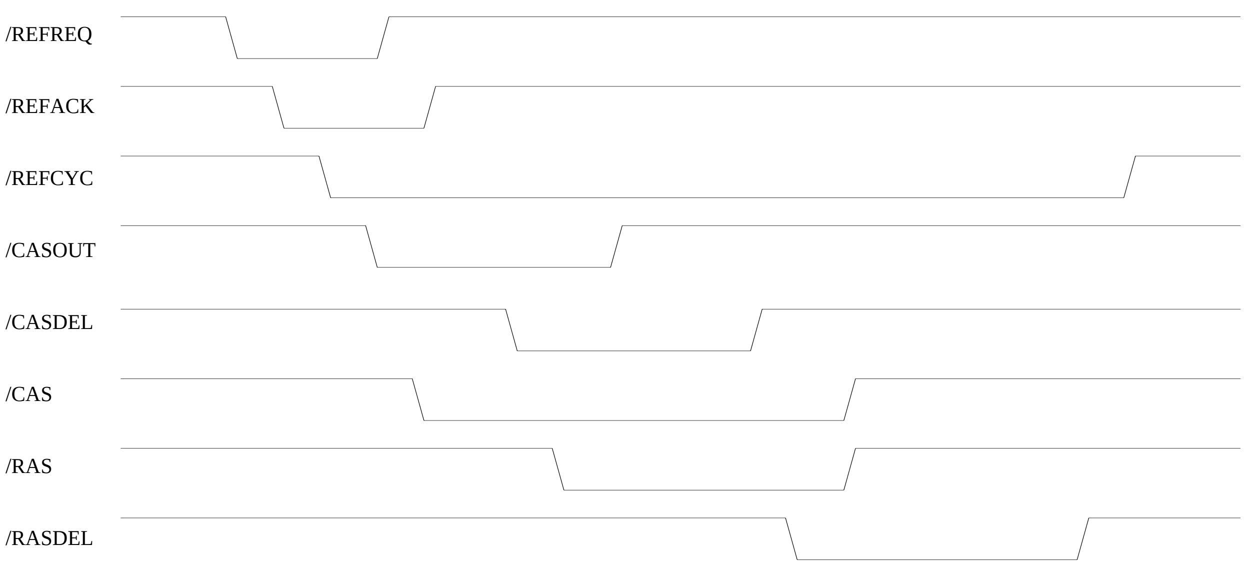


Figure 3-2: Refresh Cycle

bank and byte, the CAS generation PAL qualifies all CAS with READ. During read cycles, all four bytes in the accessed memory bank are activated, in order to support caching of this memory. Write cycles, on the other hand, are qualified with the appropriate data strobe, to assure that data is valid before a write-cycle CAS latches write data. All CAS strobes are of course qualified by /CASEN. They’re also all qualified with /CADDR, which is a strobe that assures column address setup time to CAS. This is just the 60ns tap from the RAS timing tap delay. The 40ns tap would just about make it, but leaves absolutely no margin. Since column access is rarely the limiting factor, the 60ns tap is used, for a 30ns worst case /MUX to /CADDR delay, assuming a 5% per-tap tolerance on the tap delay.

3.2.2 Bank Selection

The refresh cycle’s CAS-before-RAS logic, along with the fact that the whole board is refreshed at once, keeps things pretty simple when refresh is taking place. A normal memory cycle, however, must take into account the memory devices that actually need to be addressed. This discussion is concentrating mainly on the 256K x 4 devices, but the same principles apply to the 1M x 4 devices as well.

The basic memory unit is a 4-bit DRAM, and thus two devices are necessary to form a byte, the basic unit of interest to the Zorro III bus. This makes the smallest chunk of 32 bit memory a one megabyte chunk. So for the total of eight megabytes, we’ll have eight 1- megabyte memory banks. We want to keep RAS common among all DRAM, so it can’t be used to control banking at all. The best thing to do is divide and conquer, and that’s just what we do; find something to select between these various natural divisions.

As mentioned previously, the /CAS strobes are used to select individual bytes within a one megabyte bank of memory. This is a very natural use of /CAS, since it’s not needed until late in the memory cycle, and the data strobe lines and write data aren’t valid until later in the cycle either. The CAS PAL could easily generate a /CAS$_0$-/CAS$_3$, based directly on corresponding data strobes /DS$_0$-/DS$_3$. However, there are twice the number of output lines on this PAL device as needed for four /CAS lines, and we're still looking for a banking mechanism. With the addition of the MEG4 signal for memory sizing and the address lines A_{22} and A_{24}, the PAL comes to drive eight total /CAS lines, controlling not only byte enables but the most significant RAM bank. For 256K x 4 parts, A_{22} chooses between two 4-megabyte banks. For 1M x 4 parts A_{24} chooses between two 16-megabyte banks.

Within the 4-megabyte banks, another banking control is used. In this case, most of the work is done by the 74F138 decoder at U305. This device creates a read enable for one of four device during a read, or a write enable for one of four devices during a write. The selection of device is controlled by the BK_0 and BK_1 lines from U300. BK_0 and BK_1 are simply A_{20} and A_{21} for 256K x 4 support, or A_{22} and A_{23} for 1M x 4 support. That's all there is to bank selection. Zorro III autosizing requires board memory to be added from the lowest to the highest address on-board, but there are no hardware requirements for this.

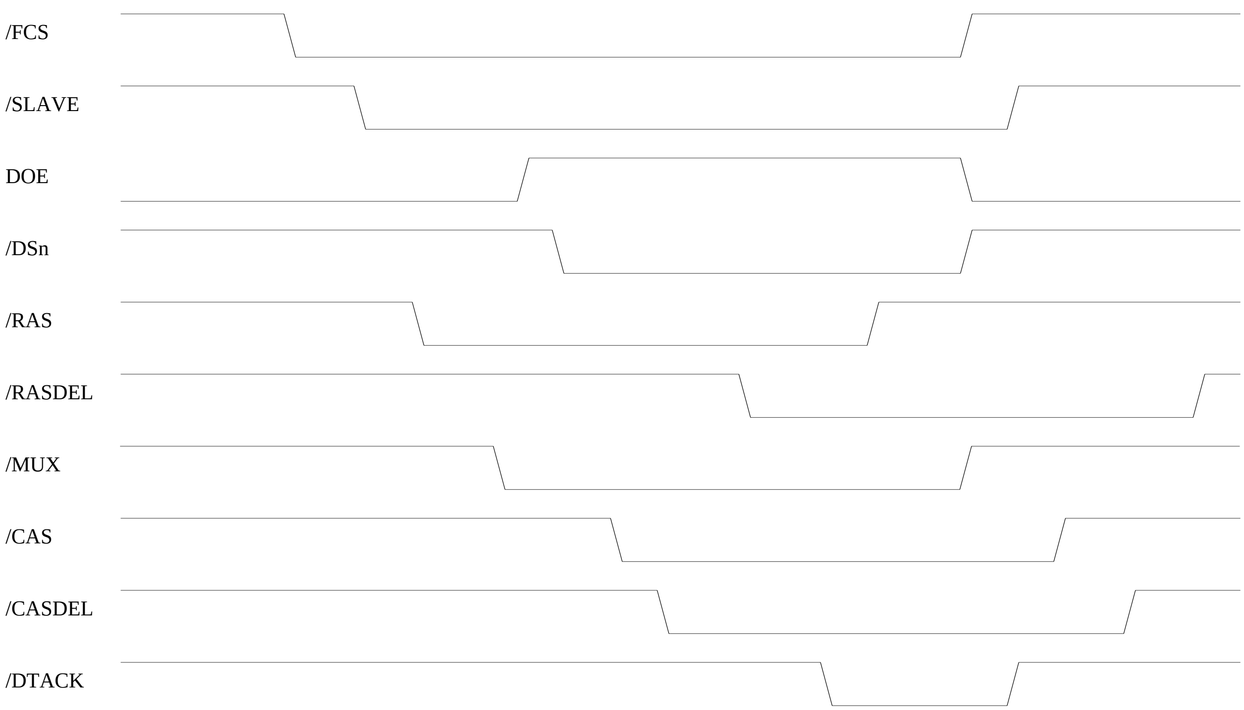


Figure 3-3: Memory Access

3.2.3 Address Multiplexing

There's nothing really complicated about the address multiplexing on this card, but it should be explained. All of the multiplexing is done with 74F258 multiplexers, and all of them are multiplexed by the /MUX signal. The first four or sixteen megabytes of memory is driven by $/MAL_0$-$/MAL_9$, the second by $/MAH_0$-MAH_9, but the multiplexing scheme is identical for both banks. When /MUX is high, the row addresses $/MA_0$-$/MA_9$ are set to the inverted A_{10}-A_{17}, A_{19}, and A_{21}, respectively. For /MUX low, the column addresses $/MA_0$-$/MA_9$ are set to the inverted A_2-A_9, A_{18}, and A_{20}, respectively. This organization may seem strange, but it makes A_2-A_7 (the Multiple Transfer static addresses), the low-order column addresses, so that Multiple Transfer Cycles can be supported via fast page or static column DRAM. This banking scheme also makes $/MA_9$, which is used only by 1M x 4 DRAM, a no-op for 256K x 4 DRAM, since BK_0-BK_1 look at A_{20} and A_{21}.

CHAPTER 4

GOING FURTHER

"There is more to life than increasing its speed."
-Mahatma Gandhi

In the general case, you can always do better. When specifics get involved, though, you may not always want to. In the specific case of this design example, you can certainly do a bit better. And if you want to make this example into a real product at some point, you should do better (thanks to this article, anyone can do at least this well just by copying).

Currently, with the Revision G Buster chip in a 25MHz A3000, this design with 80ns DRAM is running at just over half the speed of A3000 local bus memory. But part of that is the current Zorro III implementation -- this same configuration is running only about 15% slower than our prototype 50ns SRAM board! You can fully expect the Level 2 Buster chip to improve cycle times considerably, as well as supporting the faster MultipleTransfer Cycles. So, as I said, you can always do better.

4.1 Designed-In Enhancements

While not quite in the "quick and dirty" category, this example went from start to final working version in about five working days. Most of the careful design work was spent on getting the AUTOCONFIG® logic correct and understandable, since that's the most likely part of the design to be replicated in other Zorro III PICs. The actual DRAM part of it was designed, above all else, to work right the first time, since there really wasn't any time to revise the board. Because I felt that presenting a design example at a Developer's Conference without a working sample in hand would certainly be a cause for developers worry about the design's quality. So this card was designed to work, above all other concerns.

As it turns out, the original concept for the DRAM memory cycle worked fine, but the refresh logic has a rather serious flaw that hadn't been considered originally. When the design was created, the /REFACK signal was seen as the refresh control that stays valid for the entire refresh cycle, while the /REFCYC signal, then called /REFHOLD, was an end-of-cycle signal used to control the RAS precharge delay. That didn't work, and fortunately, the current mechanism could be created by changing the PAL equations, so the board was working a day after it was built up without a single cut or jumper.

However, the original memory cycle left a bit to be desired. Initially, the CAS enable didn't go out until the full RAS time had been met (eg, /RASDEL is asserted). This worked, but made CAS quite a bit later than it could have been. With a single extra wire, the CAS PAL was modified to hold off CAS until column addresses became valid. This allowed the memory timing PAL to enable CAS as soon as possible, and resulted in a 15% speedup.

The point here is that the design, as presented, isn't completely fixed. There are a considerable number of things one could do to change the memory cycle by playing around with PALs. It's conceivable that even without any additional PCB modifications, the memory cycle efficiency could be enhanced.

4.2 Modification Ideas

Opening up the design to a few PCB modifications can make things much more interesting. Of course, the ultimate modification might be to throw out the complete DRAM logic here and simply go to an off-the-shelf DRAM controller. While there's nothing wrong with that approach, and modern DRAM controllers even have an asynchronous operating mode that would work very nicely with Zorro III, there is still some performance that can be squeezed from this basic design. Most of these might have been incorporated with an extra day or so worth of design time.

4.2.1 Tighter RAM Cycles

The entire memory cycle run here is a bit less than optimal. Part of the problem is that the memory timing and CAS control PALs don't always have the same idea of when CAS should start. If the controller has a very good idea of when data is going to become valid, whether driven by T_{RAC} or T_{CAS}, the /DTACK line can be driven optimally. And, of course, the cycle can be fully T_{RAC} driven, which is usually going to be the fastest possible cycle.

Another less than optimal feature of the design is the T_{RP} assurance logic. In order to manage T_{RP} between a cycle immediately following refresh or refresh immediately following a cycle, all new cycles are held off until /RASDEL is negated. This works just fine, but the time between /RAS negated and /RASDEL negated is very close to the T_{RAS} time. For all standard DRAMs, the T_{RP} time is less, sometimes much less, than the required time for T_{RAS}. The CAS precharge time is never a problem for full cycle to full cycle operation, and unlikely to be a problem for Multiple Transfer Cycles.

The built-in support for Multiple Transfer Cycles can also be improved. The main problem for such burst cycles that doesn't crop up elsewhere is the $T_{RAS,MAX}$ time of most DRAMs in burst or static column modes. This board makes sure that a burst transfer can't exceed this limit by setting the refresh time to something just under $T_{RAS,MAX}$. When refresh comes along, it causes /MTACK to be negated at the appropriate subcycle boundary, thus making the full cycle terminate so that refresh can take place. This has two shortcomings. First of all, it makes refresh related slowdowns over 50% more likely than necessary. Additionally, the start of the burst cycle isn't synchronized with the refresh counter, so a burst can be interrupted by refresh long before necessary. Ideally, separate counters could be added for burst and refresh timeouts. Alternately, the refresh counter could be modified to change its count based on whether or not a burst cycle is under way.

Additionally, this can cause bursts to last for strange counts. When a 68030 or 68040 driving the Zorro III bus it will ask for four count burst cycles. The 68030 can handle a shorter burst, but a 68040 can't. For that reason, this design will probably require that burst be disabled when used in an '040 based Amiga systems. While the Zorro III bus doesn't require it, it's a good idea to make sure that, if possible, designs that support burst will run at least four cycles. If the refresh counter, U306, were to hold off refresh requests during burst until at least four cycles had run, that would solve the problem.

4.2.2 Read/Write Optimizations

[illegible]

[illegible] DS_N lines can latch data to the board and effect an immediate /DTACK, thereby possibly saving some of the T_{RAS} and T_{CAS} time. In fact, this could also help reads, since a latched data bus would allow the DRAMs to shut off as soon as data's latched, rather than at the end of the Zorro III cycle.

4.2.3 Standard DRAM Tricks

As with any DRAM design, the standard DRAM tricks apply here. With a bit of logic duplication, doubling up on the RAS-CAS and refresh logic, memory bank interleaving can be used to hide the RAS precharge time in most cases. Multiple Transfer Cycles can be though of as an automatic page detect, so conventional page mode or static column optimizations may not be all that useful. Then again, the Zorro III page is only 256 bytes, so perhaps a larger page could be of some help. Nybble mode memories won't really be of much use; although any burst cycle resulting from 68030 burst mode will be nybble compatible, there's no guarantee of linear addressing within a Zorro III burst cycle.

Always keep in mind the future. The Zorro III bus implementation that's currently on the A3000, as mentioned before, is already slated to improve. In the future memory will go faster on the bus than it does now, even if motherboard clocks don't go beyond 25MHz. And we expect future Zorro III machines will be running a faster Zorro III bus, going beyond what's possible in an A3000 even tomorrow.

CHAPTER 5

ADDITIONAL ZORRO III ADVICE

"Cute rots the intellect."

-Garfield

Going beyond this specific example just a bit, there are a few good things to think about when working on any Zorro III design. A large portion of this is just plain good design sense. Those without much design sense or experience should read this chapter twice, and probably learn more about the first two points from some outside materials.

5.1 Watch Those Synchronizations!

The foremost thing to be concerned about when designing for Zorro III is the fact the the bus is running asynchronously. Fully asynchronous designs, of course, will not find this to be any problem. Obviously a simply I/O chip with a 100ns access time can be timed with a delay line, keeping things very simple. Self-timed devices work real well, and are easy to hook into a Zorro III design. At the other end of the spectrum of complexity, clever clocked VLSI chips often internally synchronize things, much the way the 680x0 processors handle their "asynchronous" inputs, appearing to the world as if they're capable of running asynchronously.

If, however, you're doing your own TTL level design, such as this one, be very careful. Fully asynchronous circuits can be very tricky to design correctly; missing a strobe by a nanosecond or so can be fatal, and it may only happen every so often. The best bet is to use overlapping signals and feedback to create new signals, and *never* count on delays through PALs or TTL to provide repeatable delays. Tap delays, while not perfect, are reasonably accurate, and can be used to design reliable circuits. Inherently synchronous devices, such as many modern microprocessors, must be synchronized with external logic.

Synchronous design is usually easier, and therefore more reliable for the average designer to create. The problem here is coupling the synchronous design to the Zorro III bus. Such a design will have its own clock, but that clock can't reliably sample any Zorro III signal on a single edge. Double clocking any important Zorro III inputs with high quality flip-flops that go to clocked logic is a necessity. The problem you'll have with single clocking, metastability, won't always be immediately noticed, but it's going to be there. Better to avoid it from the start.

5.2 Design for Speed

Zorro III cards currently run around four times faster than Zorro II cards, and the limit, at least in theory, is over ten times faster. That should be a good indication that Zorro III designs are more sensitive to problems than Zorro II cards. To further aggravate the situation, you may not see any problems until faster Zorro III bus masters come along. So proper design practices are your best option. There are three main design problems that typically come up.

The speed of the design is one problem area, though it's not that much of a problem if you're up-to-date on the logic of the 1990s. While FCT and F series TTL are good for buffers and small logic functions, most fast designs these days rely heavily on programmable logic, mainly PALs. A single level of PAL logic can replace several levels of TTL, and they're aways pushing PAL speeds just a little bit more. Larger PLDs and gate arrays (programmable or custom) are always handy for complex circuits, providing they're fast enough.

5.3 Watch Out For Noise!

Noise problems are partially a result of the higher speeds involved. Eliminating such problems is achieved via a combination of circuit design and PCB layout. For noise reducing design, you need bypass capacitors of various sizes in the appropriate places. Every TTL part should have a small capacitor; we generally use something in the 0.1µF-0.22µF range. For DRAM or other surging parts, we use 0.33µF or greater. It's also a good idea to have a high frequency bypass, maybe 0.01µF or so, and a couple of larger capacitors, something in the 10µF-100µF range, randomly distributed around the design. More noise reduction can be achieved with good signal termination. Small value series termination resistors, something in the 22Ω-68Ω range works well; the values must often be tuned to the design. Tri-statable buses often benefit from some kind of parallel termination; pullups, pulldowns, or centering resistors depending on the design.

The other half of the noise problem is solved in PCB layout. Zorro III boards are expected to be multi-layer boards. Trace lengths are to be kept as short as possible, especially those on the bus side of a card; it's extremely important to minimize the noise that a card introduces to the bus. Fast and noisy signals, such as clock lines or fast control signals, should generally be given priority when routed. Component placement is also a very important job; the lengths of interconnects is directly affected by this planning. If the circuit designer isn't doing the board layout personally, he/she should develop a good working relationship with the PCB designer. If your CAD system allows the designer to specify signal importance for layout, use that feature. Any work done on keeping the design quiet will very likely be time well spent; it's going to help out in reliability, operation with other boards in the system, and government noise certifications such as FCC or FTZ.

5.4 Follow the Specifications

Let's say it once again! Any current Zorro III bus implementation is likely to be far more relaxed than the bus specification. That's going to eventually change. A proper design built today should work in tomorrow's 50MHz superAmiga, a substandard design could fail on an A3000 with the Buster Level 2 chip (Level 2 Buster drives the Zorro III bus approximately 10% faster then Level 1 does). Build in your long term viability at the design stage and save a great deal of potential future grief. You may not get fully tested on your design for some time to come.

APPENDICES

"It ain't the meat, it's the motion"
-Southside Johnny

A.1 PAL Equations

The following section contains the complete PAL equations for the five PAL devices in the BIGRAM design. All the equations are in the CUPL™ format, but should be easily translated to any other format if required. This format uses the & character to represent AND, the # symbol to represent OR, the $ symbol for XOR, and the ! symbol for negation. Standard outputs are indicated simply by name, registered outputs are indicated with the .D extension, and output enables are indicated with the .OE extension. The CUPL™ compiler minimizes equations where possible; should any equations here appear to be too large, rest assured that they will actually fit in the specified PAL.

A.1.1 Autoconfiguration Control PAL

This device is responsible for providing the AUTOCONFIG® ROM, registers, and data buffer direction control. This is to be programmed into a 15ns 16L8 or equivalent device.

```
        PARTNO      U200 ;
        NAME        U200 ;
        DATE        May 30, 1990 ;
        REV         2;
        DESIGNER    Dave Haynie ;
        COMPANY     Commodore-Amiga ;
        ASSEMBLY    BIGRAM ;
        LOCATION    U200 ;

/************************************************************************/
/*                                                                      */
/* Zorro III BIGRAM Configuration Control                               */
/*                                                                      */
/*      This device acts as configuration ROM and configuration         */
/*      register controller.                                            */
/*                                                                      */
/************************************************************************/
/*                                                                      */
/* DEVICE DATA:                                                         */
/*                                                                      */
/*      Device:     16L8-15                                             */
/*      Clock:      NONE                                                */
/*      Unused:     NONE                                                */
/*                                                                      */
/************************************************************************/

/* INPUTS: */

PIN 1  =          !SLAVE     ;       /* Board selected? */
PIN 2  =          !RST       ;       /* Board reset */
PIN 3  =          !DS3       ;       /* High order data strobe. */
PIN 4  =           READ      ;       /* Read cycle strobe */
PIN 5  =           A2        ;       /* Bus Addresses. */
PIN 6  =           A3        ;
PIN 7  =           A4        ;
PIN 8  =           A5        ;
PIN 9  =           A6        ;
PIN 11 =           A1        ;       /* This is really A8. */
PIN 16 =          !CFGOUT    ;       /* Board configured? */

/* OUTPUTS: */

PIN 19 =           D28       ;       /* Configuration data ROM nybble. */
PIN 12 =           D31       ;
PIN 13 =           D30       ;
PIN 14 =           D29       ;
PIN 15 =           DBDIR     ;       /* Data buffer direction. */

/* BIDIRECTIONALS: */

PIN 17 =          !PRECON    ;       /* Preconfiguation strobe. */
PIN 18 =           CFGLT     ;       /* Configuration address latch. */


/** INTERNAL TERMS: **/

/* Mapping A8 as A1 here makes the register pairs line up just
as they would under Zorro II configuration. */

field addr        = [A6..1];


/** OUTPUT TERMS: **/

/* The configuration ROM is created here.  The logical ordering
of it is as follows:

        REG         76543210

        00          10100001   Zorro III, autolink, 32 megabytes
        04          10010010   Product $53
        08          10110001   Extended Memory board, supports
                                Shutup, autosized in software.
        0C          00000000   Reserved
        10          00000010   Manufacturer's code (C-A)
        14          00000010
        18-3C       00000000   Zeroed options/reserved.

The autoconfiguration specs call for every readable register
except for 0 to be inverted in the physical implementation.
So the resulting map is:
```

```
        ADDR        D31 D30 D29 D28
         00          1   0   1   0
         02          0   0   0   1
         04          0   1   1   0
         06          1   1   0   1
         08          0   1   0   0
         0A          1   1   1   0
         0C          1   1   1   1
         0E          1   1   1   1
         10          1   1   1   1
         12          1   1   0   1
         14          1   1   1   1
         16          1   1   0   1
        OTHERS       1   1   1   1

Only the Zero terms are explicitly entered here; anything not specifically
driven low will be driven high.
*/

!D31                = addr:02
                    # addr:04
                    # addr:08;

!D30                = addr:00
                    # addr:02;

!D29                = addr:02
                    # addr:06
                    # addr:08
                    # addr:12
                    # addr:16;

!D28                = addr:00
                    # addr:04
                    # addr:08
                    # addr:0A;

[D31..28].OE        = SLAVE & !CFGOUT & READ;

/* This signal is driven to indicate an address latch request.
Note that the board uses 16 bit configuration write feature
to configure all at once; this isn't available in the Zorro II
configuration space. */

CFGLT               = SLAVE  & PRECON & !A3
                    # CFGLT & !RST;

/* If the board is told to shut up or configure, this line is
asserted and held through reset.  The logical SHUTUP line
is PRECON & !CFGLT, once FCS is negated. */

PRECON              = SLAVE & DS3 & !READ  &  addr:4C
                    # SLAVE & DS3 & !READ & addr:44
                    # PRECON & !RST;

/* This controls the data buffer direction between the PIC's
local bus and the expansion bus. */

DBDIR               = SLAVE & READ;
```

A.1.2 Board Control PAL

This device controls an assortment of board functions. It creates the /SLAVE, /CFGOUT, and /MTACK signals for Zorro III. It creates the data buffer enable for the bus buffers, and the burst-enable line used by the memory system. And it arbitrates DRAM refresh. This is programmed into a 10ns 20L8 or equivalent PAL.

```
        PARTNO      U201 ;
        NAME        U201 ;
        DATE        May 30, 1990 ;
        REV         3 ;
        DESIGNER    Dave Haynie ;
        COMPANY     Commodore-Amiga ;
        ASSEMBLY    BIGRAM ;
        LOCATION    U201 ;

/************************************************************************/
/*                                                                      */
/* Zorro III BIGRAM Board Control                                       */
/*                                                                      */
/*      This device controls the main features of the BIGRAM board.     */
/*                                                                      */
/************************************************************************/
/*                                                                      */
/* DEVICE DATA:                                                         */
/*                                                                      */
/*      Device:     20L8-10                                             */
/*      Clock:      NONE                                                */
/*      Unused:     22(O)                                               */
/*                                                                      */
/************************************************************************/

/* INPUTS: */

PIN 1  =           !MATCH     ;       /* Address match from comparator. */
PIN 2  =            CFGLT     ;       /* Configuration latch. */
PIN 3  =           !PRECON    ;       /* Board was configed or shutup. */
PIN 4  =           !FCS       ;       /* Full Cycle Strobe. */
PIN 5  =           !CFGIN     ;       /* Configuration chain in. */
PIN 6  =            FC0       ;       /* Function codes, don't ignore these! */
PIN 7  =            FC1       ;
PIN 8  =           !REFREQ    ;       /* Refresh request from refresh counter */
PIN 9  =           !Z2SHUNT   ;       /* Zorro II backplane bypass. */
PIN 10 =            DOE       ;       /* Data enable. */
PIN 11 =           !BERR      ;       /* Bus error, all off. */
PIN 13 =           !REFCYC    ;       /* We're in a refresh cycle. */
PIN 14 =           !BRENB     ;       /* Burst/Multiple transfer enable. */
PIN 20 =           !MTCR      ;       /* We're in a multiple cycle. */
PIN 23 =            A24       ;       /* Latched bus address 24. */

/* OUTPUTS: */

PIN 15 =           !DBOE      ;       /* Data buffer output enable. */

/* BIDIRECTIONALS: */

PIN 16 =           !CFGOUT    ;       /* Board is configured. */
PIN 17 =           !REFACK    ;       /* Refresh acknowledge. */
PIN 18 =           !MTACK     ;       /* Multiple transfer acknowledge. */
PIN 19 =           !SLAVE     ;       /* Board select. */
PIN 21 =           !BURST     ;       /* This is a burst cycle. */

/** INTERNAL TERMS: **/

/* The valid board address consists of a comparator match and a valid
memory space.  The valid spaces are as follows:

        SPACE                 FC2 FC1 FC0
        Reserved               0   0   0
        User Data              0   0   1
        User Program           0   1   0
        Reserved               0   1   1
        Reserved               1   0   0
        Supervisor Data        1   0   1
        Supervisor Program     1   1   0
        CPU                    1   1   1

This reduces to the equation used: FC0 XOR FC1. The external comparator
only looks at A31..A25, which is OK for normal operation (we're a 32
meg board), but bad for configuration.  So if we're not yet configured,
A24 must be high for a select match.
*/

select             = MATCH & (FC0 $ FC1) & (CFGOUT # A24);
```

```
/* This indicates a normal board select; SLAVE starts the cycle, FCS
cuts it off quickly at the end. */

hit                = SLAVE & FCS;

/* OUTPUT TERMS: */

/* This output controls the data buffer enable pins.  Data buffers
turn on when DOE is asserted and the board is selected, they
turn off as quickly after a cycle ends as possible. */

DBOE               = hit & DOE & !BERR;

/* This signal indicates that the board is configured. The board is
considered configured if actually configured, shut up, or placed
in a Zorro II backplane.  It only responds if actually configured,
of course. This signal must only change at the end of a cycle, if
actually operating. */

CFGOUT             =  PRECON & CFGIN & !FCS & !DOE
                   #  PRECON & CFGOUT
                   # Z2SHUNT & CFGIN;

/* This is the refresh acknowledge cycle.  When the a refresh request
comes in, and the coast is clear, this line is asserted to start
the refresh machine.  Determining when the coast is clear, eg,
arbitrating refresh, is the trick to all hand-made DRAM controllers.
This one works pretty simply.  The coast is clear when there's no
bus cycle happening, or when a bus cycle is happening but another
slave is responding.  The trick is avoid races; FCS could be
changing just as REFREQ comes in.  Therefore, the second half of
this arbiter is in the RAS cycle generation, which doesn't start
until REFACK is negated and SLAVE is asserted.  */

REFACK             = REFREQ & !FCS & !MATCH
                   # REFREQ &  FCS & !SLAVE & DOE
                   # REFACK &  REFREQ;

/* The multiple cycle transfer acknowledge.  If the jumper enables
them, and a refresh isn't already requested, we'll acknowledge
them.  If a refresh request comes in, we'll negate MTACK after
the current cycle finishes, which will result in one more
burst cycle before the full cycle terminates and the refresh
can be acknowledged.  I do it this way because I use the
refresh timer to handle the TRASMAX limitation of the DRAM as
wall as handling refresh. */

MTACK              = hit & BRENB & !REFREQ
                   # hit & MTACK & !DOE
                   # hit & MTACK &  MTCR;

MTACK.OE           = hit;

/* This is SLAVE, the board select line.  Most board activity centers
around this line.  If the board is selected and unconfigured,
always respond.  Once configured, only respond if it's not shutup
or shunted.  This line is held through the cycle's end. */

SLAVE              = select & FCS & CFGIN & !CFGOUT
                   # select & FCS & CFGLT &  CFGOUT;

/* This indicates if the cycle is a burst cycle.  The first cycle is
always a non-burst cycle.  If, at the end of the first cycle,
MTCR and MTACK are asserted, all subsequent cycles are burst
until FCS is negated. */

BURST              = SLAVE & DOE & MTCR & MTACK
                   # BURST & FCS;
```

A.1.3 Memory Timing PAL

This device controls RAS and CAS timing, /DTACK generation, and high order RAM banking. This must be programmed into a 10ns 20L8 or equivalent device.

```
        PARTNO     U300 ;
        NAME       U300 ;
        DATE       January 10, 1991 ;
        REV        6 ;
        DESIGNER   Dave Haynie ;
        COMPANY    Commodore-Amiga ;
        ASSEMBLY   BIGRAM ;
        LOCATION   U300 ;

/************************************************************************/
/*                                                                      */
/* Zorro III BIGRAM DRAM Timing                                         */
/*                                                                      */
/*      This device controls the standard and refresh timing of the    */
/*      dynamic RAM. Big-Time asynchronicity ahead!  This also controls */
/*      banking within a CAS controlled memory bank.                    */
/*                                                                      */
/************************************************************************/
/*                                                                      */
/* DEVICE DATA:                                                         */
/*                                                                      */
/*      Device:               20L8-10                                   */
/*      Clock:                NONE                                      */
/*      Unused:               NONE                                      */
/*                                                                      */
/************************************************************************/
/*                                                                      */
/* REVISION HISTORY:                                                    */
/*                                                                      */
/*      DBH 05-02: Original version.                                    */
/*      DBH 05-24: Hacked refresh logic to remove CAS bounce.           */
/*      DBH 05-24: More refresh logic adjustments.                      */
/*      DBH 05-24: Fixed refresh end-of-cycle.                          */
/*      DBH 05-30: Fixed slow CAS problems.                             */
/*                                                                      */
/*      DBH 01-10: Fixed burst RAS problem.                             */
/*      DBH 01-10: Fixed page-mode RAS/CAS generation.                  */
/*                                                                      */
/************************************************************************/

/* INPUTS: */

PIN 1  =           !RASDEL    ;       /* RAS strobe delay */
PIN 2  =           !MUX       ;       /* DRAM Address multiplexer */
PIN 3  =           !MTCR      ;       /* Multiple cycle request. */
PIN 4  =           !BURST     ;       /* We're in burst mode. */
PIN 5  =            DOE       ;       /* Data time */
PIN 6  =           !SLAVE     ;       /* The board is responding */
PIN 7  =           !REFACK    ;       /* We're servicing a refresh request */
PIN 8  =            SCRAM     ;       /* We're using static column RAM. */
PIN 9  =            A23       ;       /* System addresses */
PIN 10 =            A22       ;
PIN 11 =            A21       ;
PIN 13 =            A20       ;
PIN 14 =            MEG4      ;       /* 4 Meg parts? */
PIN 23 =           !CASDEL    ;       /* CAS strobe delay */

/* BIDIRECTIONALS: */

PIN 16 =           !CASEN     ;       /* CAS strobe enable */
PIN 17 =           !CASOUT    ;       /* CAS delay input */
PIN 18 =           !REFCAS    ;       /* CAS for refresh */
PIN 19 =           !REFCYC    ;       /* We're in a refresh cycle. */
PIN 20 =           !DTACK     ;       /* Data is valid on bus */
PIN 21 =           !RASEN     ;       /* RAS strobe enable */

/* OUTPUTS: */

PIN 22 =           BK0        ;       /* Small Bank bit 0 */
PIN 15 =           BK1        ;       /* Small Bank bit 1 */

/** OUTPUT TERMS: **/

/* The data valid signal.  Data is valid on the bus if we're not in a refresh cycle,
the board is selected, and something's happened.  The non-burst cycle is driven by RAS
delay only, the burst cycle by CAS delay only. */

DTACK              = SLAVE & !BURST & !REFACK & !REFCYC & DOE & RASDEL & CASDEL
                   # SLAVE & !BURST &  DTACK
                   # SLAVE &  BURST & !REFACK & !REFCYC & DOE & CASOUT & CASDEL & MTCR
                   # SLAVE &  BURST &  DTACK & MTCR;
```

```
DTACK.OE           = SLAVE;

/* The RAS enable strobe.  If we're not in refresh, it goes as soon
as we're sure the board is selected.  If refresh is called for,
start a RAS cycle after the CAS delay. */

RASEN              = !REFACK & !REFCYC & !RASDEL & !CASEN & SLAVE
                   # !REFACK & !REFCYC &   RASEN &  SLAVE
                   #  REFCYC &  CASDEL & !RASDEL
                   #  REFCYC &   RASEN & !RASDEL;

/* The CAS enable works differently for burst vs. non-burst.  For non-burst,
it follows RASEN after DOE and MUX are asserted.  In a burst cycle, it
follows MTCR.  For refresh, CAS can't be enabled until we're sure that
RASDEL is negated, thus ensuring RAS precharge when a refresh cycle
immediately follows a standard memory cycle.  */

CASOUT             = !REFACK & !REFCYC & !BURST &   RASEN &   MUX &   DOE & !RASDEL
                   # !REFACK & !REFCYC & !BURST &  CASOUT & SLAVE
                   # !REFACK & !REFCYC &  BURST & !CASDEL & MTCR
                   # !REFACK & !REFCYC &  BURST &  CASOUT & MTCR
                   #  REFACK &  REFCYC & !RASEN & !RASDEL
                   #  CASOUT &  REFCYC & !RASEN;

/* The actual CAS that goes out is modified by our use of SCRAMs.  If
SCRAMs are in use, CASEN goes low and stays low, while CASOUT works
the DTACK line.  Otherwise, CASEN and CASOUT are the same. */

CASEN              = !REFACK & !REFCYC & !SCRAM & CASOUT
                   # !REFACK & !REFCYC &  SCRAM & CASOUT
                   # !REFACK & !REFCYC &  SCRAM &  CASEN & SLAVE;

/* This is the rest of the refresh machine.  A refresh cycle starts with a
valid refresh acknowledge and the assertion of the standard and refresh
CAS.  RAS for refresh is asserted one CASDEL later, and standard CAS is
negated at the same point.  The refresh counter will clear REFREQ when
REFCYC is asserted, and clear REFACK when REFREQ is negated. */

REFCAS             = REFACK & REFCYC & !CASDEL & !RASDEL
                   # REFCAS & REFCYC & !MUX
                   # REFCAS & REFCYC &  RASEN & !RASDEL;

REFCYC             = REFACK & !CASDEL & !RASDEL
                   # REFCYC &  CASOUT & !RASDEL
                   # REFCYC &   RASEN
                   # REFCYC &  RASDEL;

/* Bank control. The bank is controlled by A23 and A22 for 4 Meg memory,
A21 and A20 for 1 Meg memory. */

BK0                = A22 &  MEG4
                   # A20 & !MEG4;

BK1                = A23 &  MEG4
                   # A21 & !MEG4;
```

A.1.4 CAS Control PAL

This device controls the CAS generation and banking. This must be programmed into a 15ns 20L8 PAL device or equivalent.

```
        PARTNO     U304 ;
        NAME       U304 ;
        DATE       May 30, 1990 ;
        REV        3 ;
        DESIGNER   Dave Haynie ;
        COMPANY    Commodore-Amiga ;
        ASSEMBLY   BIGRAM ;
        LOCATION   U304 ;

/*************************************************************************/
/*                                                                       */
/* Zorro III BIGRAM DRAM CAS Select                                      */
/*                                                                       */
/*      This device controls the CAS strobes, which control DRAM byte    */
/*      enables and most significant bank.                               */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/* DEVICE DATA:                                                          */
/*                                                                       */
/*      Device:                 20L8-15                                  */
/*      Clock:                  NONE                                     */
/*      Unused:                 NONE                                     */
/*                                                                       */
/*************************************************************************/

/* INPUTS: */

PIN 1  =           !CASEN     ;       /* Normal CAS enable */
PIN 2  =           !DTACK     ;       /* Zorro III cycle termination */
PIN 3  =           !REFCAS    ;       /* CAS for refresh cycle */
PIN 4  =           !REFACK    ;       /* We're in refresh */
PIN 5  =           !DS3       ;       /* Zorro III data strobes */
PIN 6  =           !DS2       ;
PIN 7  =           !DS1       ;
PIN 8  =           !DS0       ;
PIN 9  =            SCRAM     ;       /* Using static column memories */
PIN 10 =            READ      ;       /* Zorro III Read enable */
PIN 11 =            MEG4      ;       /* Are we using 4 Meg parts? */
PIN 13 =           !CADDR     ;       /* Column Address Valid */
PIN 14 =            A24       ;       /* Address lines */
PIN 23 =            A22       ;

/* OUTPUTS: */

PIN 15 =           !CASL0     ;       /* Lower bank CAS */
PIN 16 =           !CASL1     ;
PIN 17 =           !CASL2     ;
PIN 18 =           !CASL3     ;
PIN 19 =           !CASH0     ;       /* Upper bank CAS */
PIN 20 =           !CASH1     ;
PIN 21 =           !CASH2     ;
PIN 22 =           !CASH3     ;

/** INTERNAL TERMS: **/

/* The CAS lines are the highest order banking control.  If we're using 1 Meg
parts, lower is $0000000-$03fffff, upper is $0400000-$07fffff, so A22 controls
the banking.  If we're using 4 Meg parts, lower is $0000000-$0ffffff, upper is
$1000000-$1ffffff, so A24 controls the banking.  */

lower              = !A24 &  MEG4 & CASEN & CADDR
                   # !A22 & !MEG4 & CASEN & CADDR;
upper              =  A24 &  MEG4 & CASEN & CADDR
                   #  A22 & !MEG4 & CASEN & CADDR;

/** OUTPUT TERMS: **/

/* The CAS terms are simple.  There are two banks of memory, and the banking
is controlled as above.  On writes, the data strobes control the particular
CAS line, and we wait for WRDEL so that data is guaranteed valid on the
DRAM bus.  On reads, all CAS lines in a bank are asserted ASAP.  On
refresh, all CAS lines are asserted. */

CASL0              = lower & !READ & DS0
                   # lower &  READ
                   # REFCAS;

CASL1              = lower & !READ & DS1
                   # lower &  READ
                   # REFCAS;
```

```
CASL2               = lower & !READ & DS2
                    # lower &  READ
                    # REFCAS;

CASL3               = lower & !READ & DS3
                    # lower &  READ
                    # REFCAS;

CASH0               = upper & !READ & DS0
                    # upper &  READ
                    # REFCAS;

CASH1               = upper & !READ & DS1
                    # upper &  READ
                    # REFCAS;

CASH2               = upper & !READ & DS2
                    # upper &  READ
                    # REFCAS;

CASH3               = upper & !READ & DS3
                    # upper &  READ
                    # REFCAS;
```

A.1.5 Refresh Counter PAL

This device is responsible for timing the CAS-before-RAS refresh used by the DRAM system. This must be programmed into a 25ns 16R8 or equivalent device.

```
        PARTNO     U306 ;
        NAME       U306 ;
        DATE       May 30, 1990 ;
        REV        1 ;
        DESIGNER   Dave Haynie ;
        COMPANY    Commodore-Amiga ;
        ASSEMBLY   BIGRAM ;
        LOCATION   U306 ;

/***********************************************************************/
/*                                                                     */
/* Zorro III BIGRAM DRAM Refresh Counter.                              */
/*                                                                     */
/*      This device is responsible for generating refresh request.     */
/*                                                                     */
/***********************************************************************/
/*                                                                     */
/* DEVICE DATA:                                                        */
/*                                                                     */
/*      Device:    16R8-25                                             */
/*      Clock:     C7M                                                 */
/*      Unused:    NONE                                                */
/*                                                                     */
/***********************************************************************/

/* INPUTS: */

PIN 2   =          !REFACK    ;       /* We're servicing a refresh request */
PIN 3   =          !REFCYC    ;       /* We're in a refresh cycle. */

/* BIDIRECTIONALS: */

PIN 19 =           !REFREQ    ;       /* Refresh request */

/* USED INTERNALLY: */

PIN 18 =           !R0        ;       /* Counter bits */
PIN 17 =           !R1        ;
PIN 16 =           !R2        ;
PIN 15 =           !R3        ;
PIN 14 =           !R4        ;
PIN 13 =           !R5        ;
PIN 12 =           !R6        ;

/** INTERNAL TERMS: **/

field   count      = [R6..0];

/** OUTPUT TERMS: **/

/* The refresh request is asserted when the terminal count has been reached.
It's held until REFHOLD is asserted. */


REFREQ.D           = count:70
                   # REFREQ & !REFCYC;

/* The refresh counter is pretty simple.  We're assuming one refresh cycle
every 15,625ns, which works out fine for the 8ms, 512 row 1 Meg parts
or the 16ms, 1024 row 4 Meg parts.  However, the maximum TRAS period
is only 10,000ns, which must be taken into account to support burst
mode.  Counting 71 140ns C7M clocks gets me to 9,940ns, close enough.
The counter resets when REFCYC comes along. */

R0.D               = !REFCYC & !R0;

R1.D               = !REFCYC &  R0 & !R1
                   # !REFCYC & !R0 &  R1;

R2.D               = !REFCYC &  R0 &  R1 & !R2
                   # !REFCYC &       !R1 &  R2
                   # !REFCYC & !R0       &  R2;

R3.D               = !REFCYC &  R0 &  R1 &  R2 & !R3
                   # !REFCYC &             !R2 &  R3
                   # !REFCYC &       !R1       &  R3
                   # !REFCYC & !R0             &  R3;

R4.D               = !REFCYC &  R0 &  R1 &  R2 &  R3 & !R4
                   # !REFCYC &                   !R3 &  R4
                   # !REFCYC &             !R2       &  R4
```

```
                   # !REFCYC &       !R1             &  R4
                   # !REFCYC & !R0                   &  R4;

R5.D               = !REFCYC &  R0 &  R1 &  R2 &  R3 &  R4 & !R5
                   # !REFCYC &                          !R4 &  R5
                   # !REFCYC &                   !R3       &  R5
                   # !REFCYC &             !R2             &  R5
                   # !REFCYC &       !R1                   &  R5
                   # !REFCYC & !R0                         &  R5;

R6.D               = !REFCYC &  R0 &  R1 &  R2 &  R3 &  R4 &  R5 & !R6
                   # !REFCYC &                                !R5 &  R6
                   # !REFCYC &                         !R4       &  R6
                   # !REFCYC &                   !R3             &  R6
                   # !REFCYC &             !R2                   &  R6
                   # !REFCYC &       !R1                         &  R6
                   # !REFCYC & !R0                               &  R6;
```

A.2 Schematics

The following pages contain the schematics for the example memory board. The list of parts is as follows:

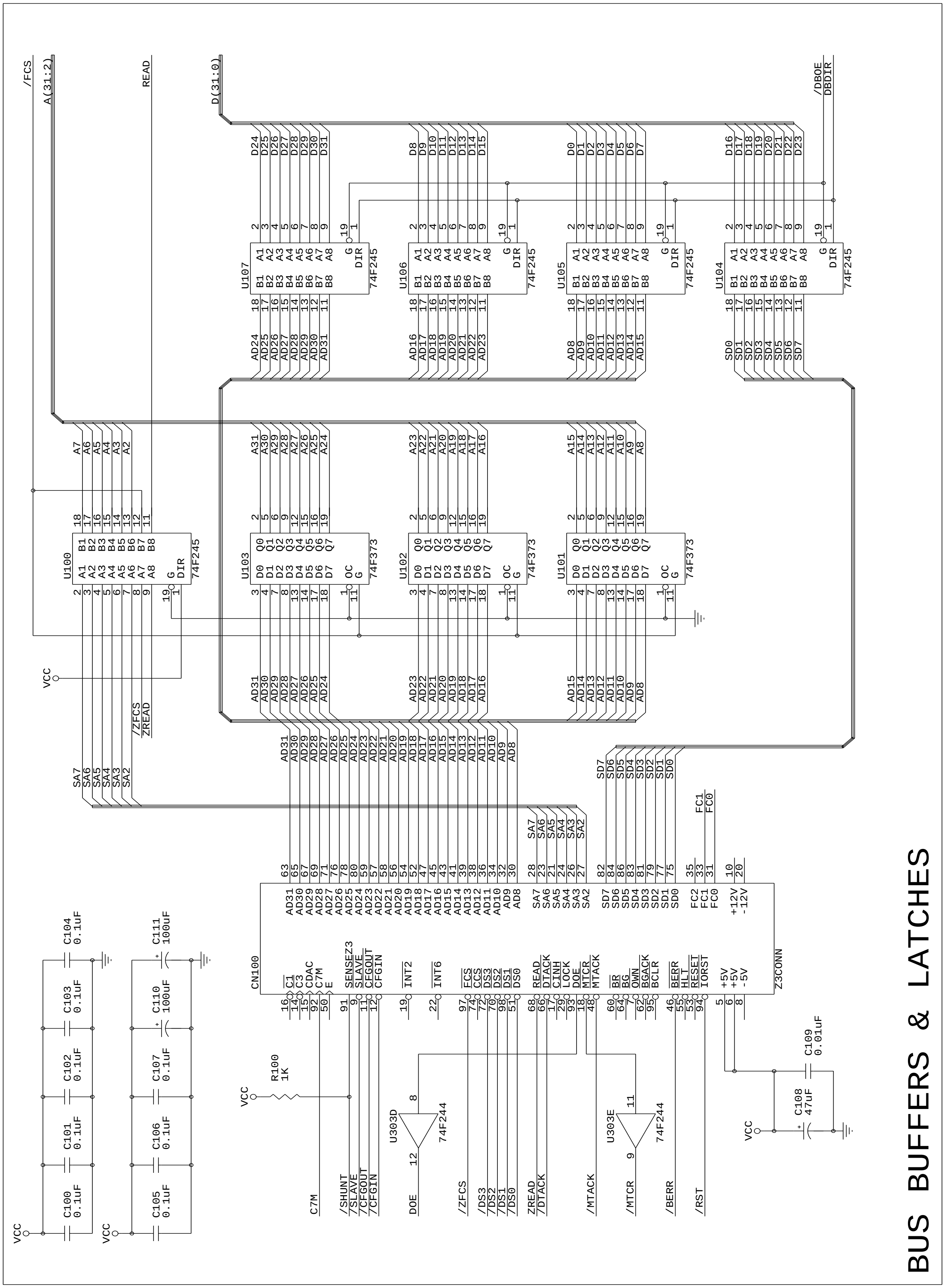
Capacitors

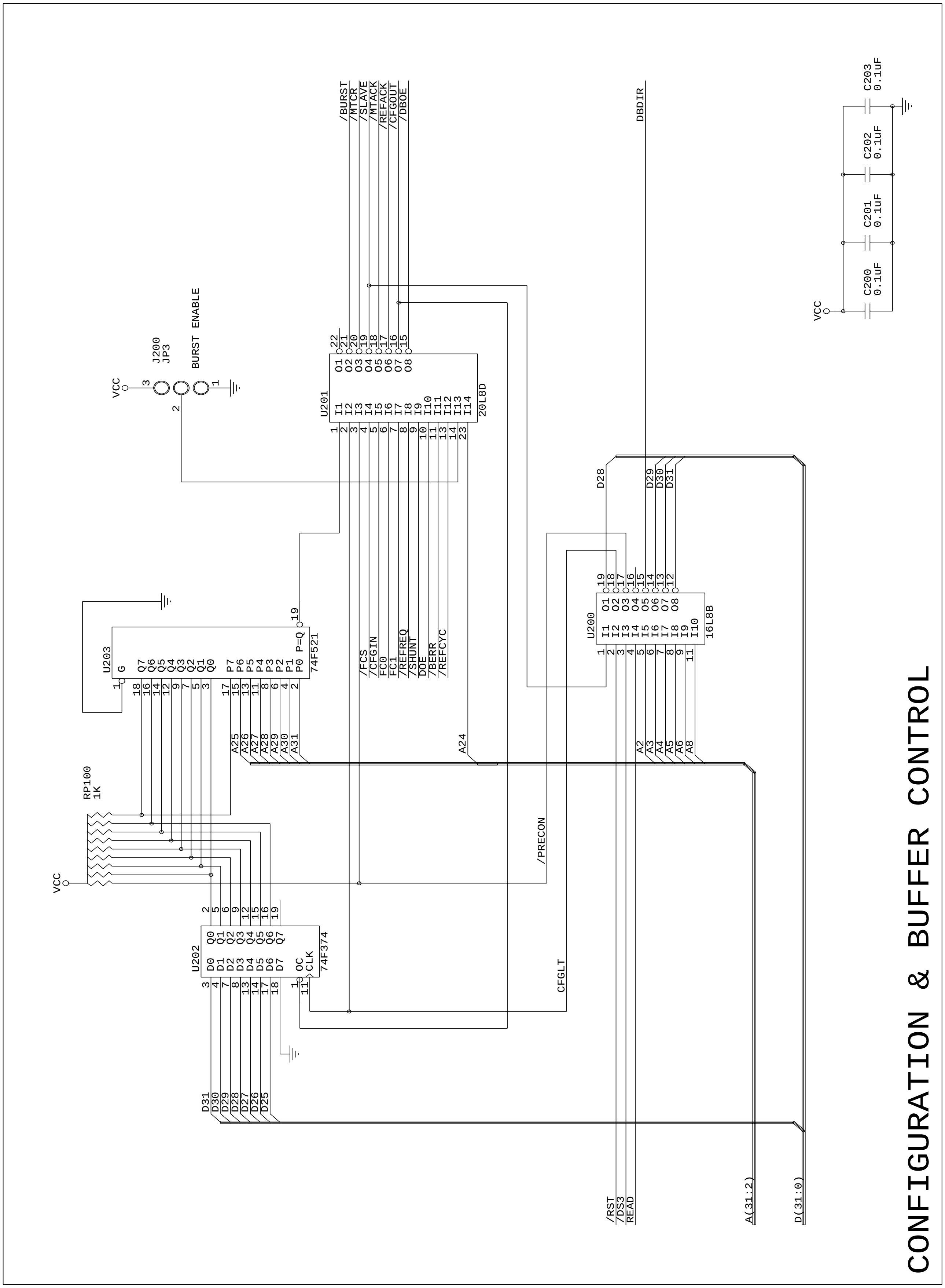
0.01 μF MLC C109

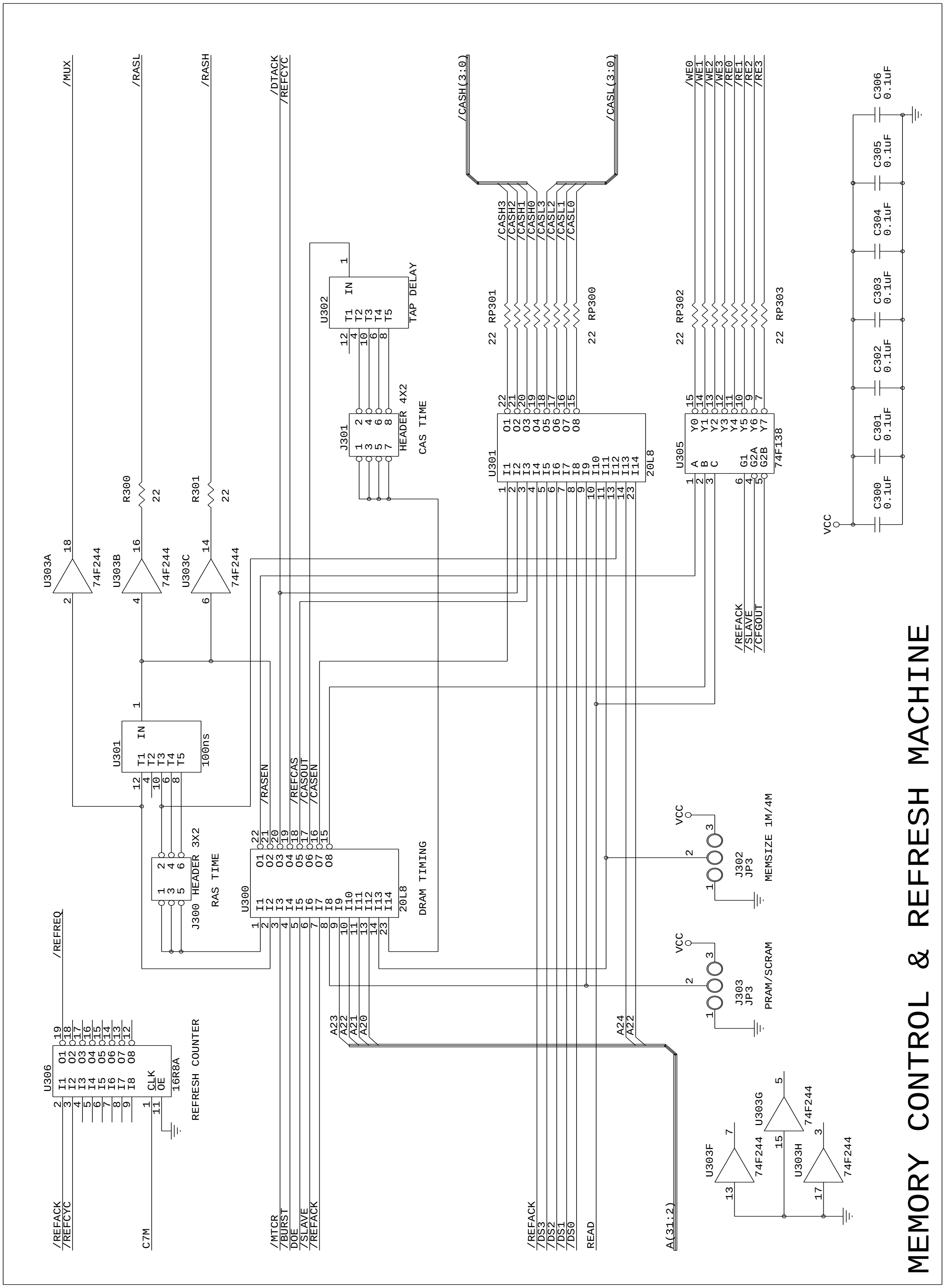
0.10 μF MLC

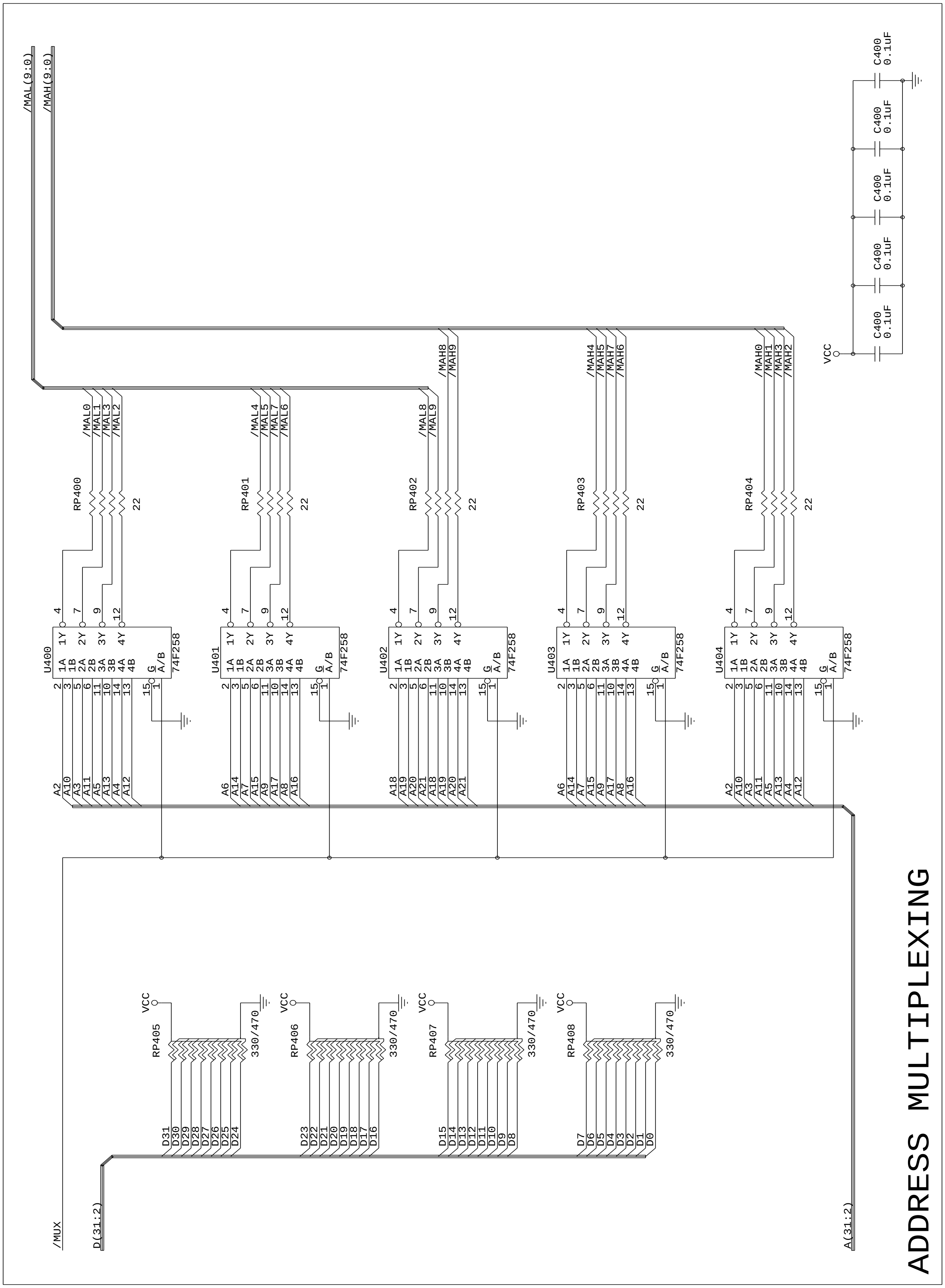
C100-C107,C200-C203,C300-C306,C400-C404

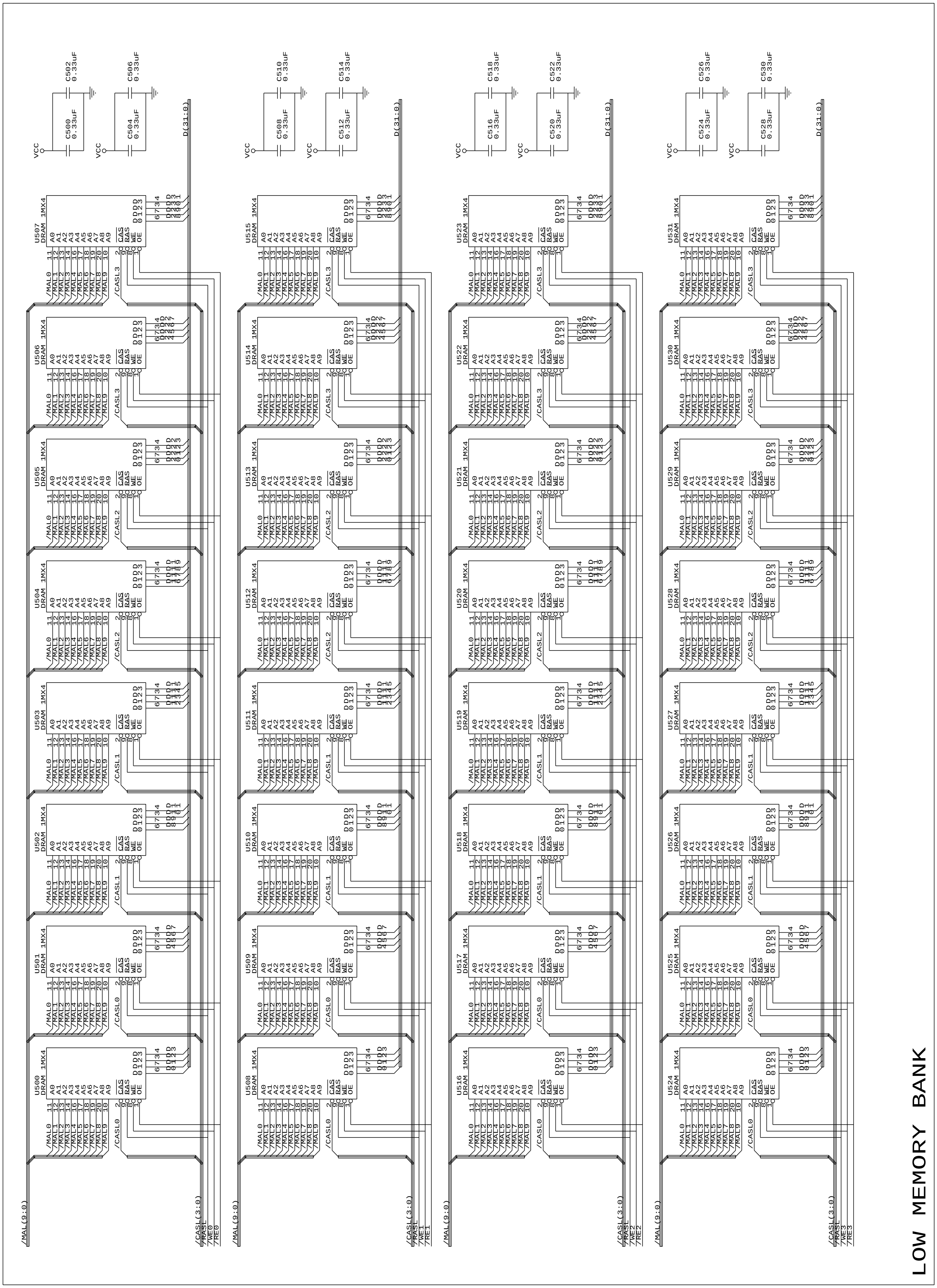
0.33 μF MLC

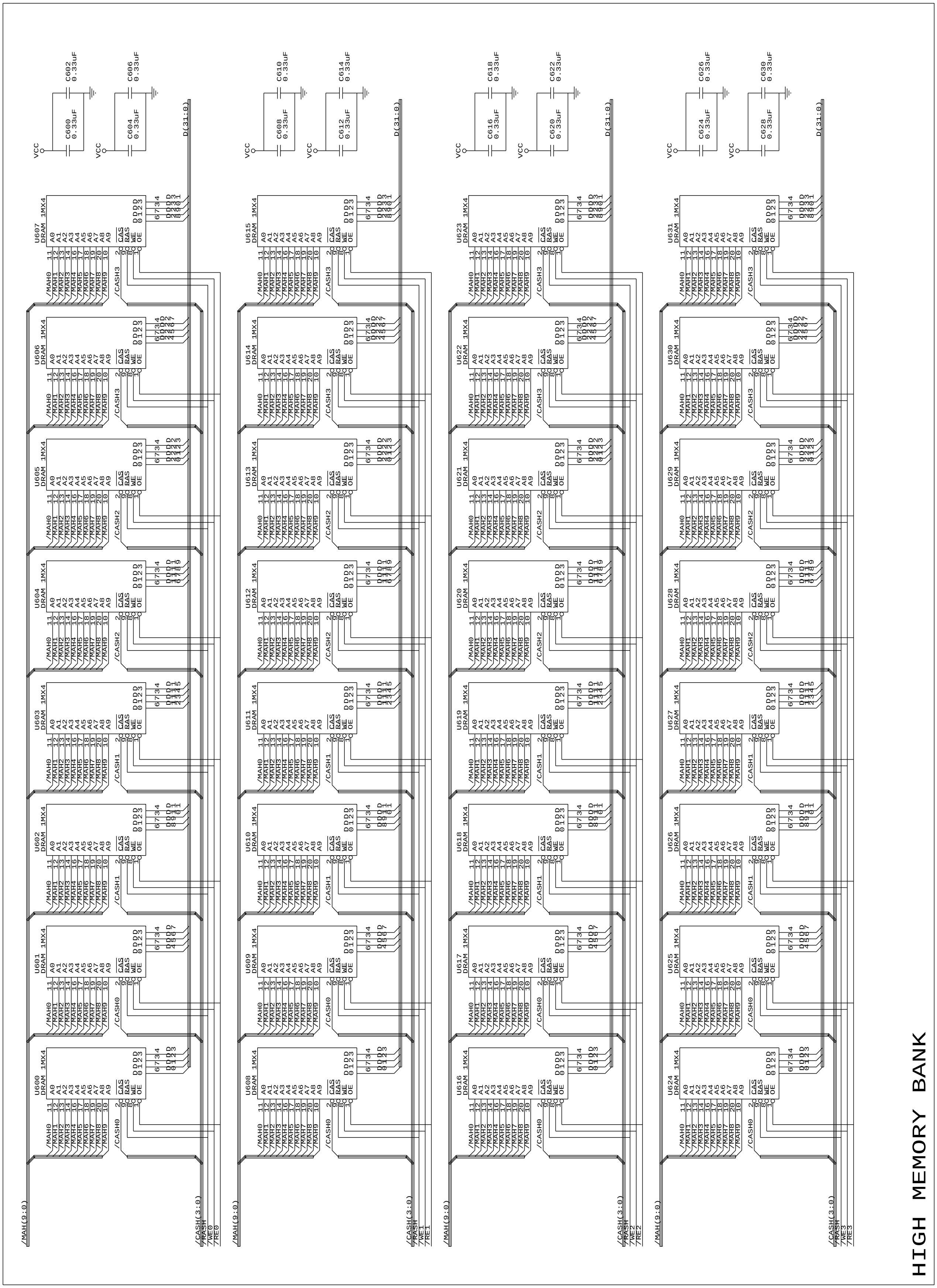

BUS BUFFERS & LATCHES
U100 74F245
U101 74F373
U102 74F373
U103 74F373
U104 74F245
U105 74F245
U106 74F245
U107 74F245
CN100 Z3CONN
U303D 74F244
U303E 74F244
R100 1K
C100 0.1uF
C101 0.1uF
C102 0.1uF
C103 0.1uF
C104 0.1uF
C105 0.1uF
C106 0.1uF
C107 0.1uF
C108 47uF
C109 0.01uF
C110 100uF
C111 100uF


CONFIGURATION & BUFFER CONTROL


MEMORY CONTROL & REFRESH MACHINE
U306 16R8A REFRESH COUNTER
U300 20L8 DRAM TIMING
U301 20L8
U305 74F138
U301 100ns
U302 TAP DELAY
J300 HEADER 3X2 RAS TIME
J301 HEADER 4X2 CAS TIME
J302 JP3 MEMSIZE 1M/4M
J303 JP3 PRAM/SCRAM
U303A 74F244
U303B 74F244
U303C 74F244
U303F 74F244
U303G 74F244
U303H 74F244
22 RP300
22 RP301
22 RP302
22 RP303
R300 22
R301 22
C300 0.1uF
C301 0.1uF
C302 0.1uF
C303 0.1uF
C304 0.1uF
C305 0.1uF
C306 0.1uF


ADDRESS MULTIPLEXING
/MAL(9:0)
/MAH(9:0)
U400
U401
U402
U403
U404
74F258
RP400
RP401
RP402
RP403
RP404
22
RP405
RP406
RP407
RP408
330/470
VCC
C400
0.1uF
/MUX
D(31:2)
A(31:2)


LOW MEMORY BANK


HIGH MEMORY BANK

A.3 Zorro III Configuration

While AmigaOS 2.0 understands Zorro III AUTOCONFIG® conventions, the following routine is useful for configuring simple Zorro III boards in an AmigaOS 1.3 system. Note that many popular MMU configurations don't map in the Zorro III configuration space at $FF000000, so this program is not likely to work with an MMU mapping in place.

```
/* ====================================================================== */

/* A very simple configuration utility for Zorro III boards.  This code will
    configure Zorro III cards that are placed after any Zorro II cards in
    the A3000.  All configuration is done based on 16 meg slots and no magic
    for autoboot, etc.  */

#include <exec/types.h>
#include <exec/memory.h>
#include <libraries/configregs.h>
#include <libraries/configvars.h>
#include <libraries/expansionbase.h>
#include <stdio.h>
#include <ctype.h>
#include <functions.h>

/* ====================================================================== */

/* Modified configuration information. */

/* Extensions to the TYPE field. */

#define E_Z3EXPBASE        0xff000000L
#define E_Z3EXPSTART       0x10000000L
#define E_Z3EXPFINISH      0x7fffffffL
#define E_Z3SLOTSIZE       0x01000000L
#define E_Z3ASIZEINC       0x00010000L

#define ERT_ZORROII        ERT_NEWBOARD
#define ERT_ZORROIII       0x80

/* Extensions to the FLAGS field. */

#define ERFB_EXTENDED      5L
#define ERFF_EXTENDED      (1L<<5)

static BoardSize[2][8] = {
    { 0x00800000,0x00010000,0x00020000,0x00040000,
      0x00080000,0x00100000,0x00200000,0x00400000 },
    { 0x01000000,0x02000000,0x04000000,0x08000000,
      0x10000000,0x20000000,0x40000000,0x00000000 }
};

#define ERFB_QUICKVALID    4L
#define ERFF_QUICKVALID    (1L<<4)

#define ERF_SUBMASK        0x0fL
#define ERF_SUBSAME        0x00L
#define ERF_SUBAUTO        0x01L
#define ERF_SUBFIXED       0x02L
#define ERF_SUBRESERVE     0x0eL

static SubSize[16] = {
    0x00000000,0x00000000,0x00010000,0x00020000,
    0x00040000,0x00080000,0x00100000,0x00200000,
    0x00400000,0x00600000,0x00800000,0x00a00000,
    0x00c00000,0x00e00000,0x00000000,0x00000000
};

#define PRVB(x)if (verbose) { printf(x); }

static BOOL                verbose = TRUE;
static BOOL                anyone = FALSE;
struct ExpansionBase       *ExpansionBase;
static ULONG               Z3Space = 0x10000000L;

/* ====================================================================== */

/* These functions are involved in finding a Zorro III board. */

/* This function reads the logical value stored at the given Zorro III
    ROM location.  This corrects for complements and the differing offsets
    depending on location. */
```

```
UBYTE ReadZ3Reg(base,reg)
WORD *base;
WORD reg;
{
    ULONG *Z3base;
    UWORD result;

    if (base == (WORD *)E_EXPANSIONBASE) {
        base += (reg>>1);
        result = ((*base++)&0xf000)>>8;
        result = ((*base)&0xf000)>>12;
    } else {
        Z3base = (ULONG *)(base+(reg>>1));
        result = ((*Z3base)&0xf0000000)>>24;
        result |= ((*(Z3base+0x40))&0xf0000000)>>28;
    }
    if (reg) result = ~result;

    return (UBYTE)result;
}

/* This function types the board in the system, returning the type code.
    There are four possibilities -- no board, a Zorro II board, a Zorro III
    board at the Zorro II configuration slot, and a Zorro III board at the
    Zorro III configuration slot. */

#define BT_NONE        0
#define BT_Z2          1
#define BT_Z3_AT_Z2    2
#define BT_Z3_AT_Z3    3

BYTE TypeOfPIC() {
    UBYTE type;
    UWORD manf;

    type = ReadZ3Reg(E_EXPANSIONBASE,0x00);
    manf = ReadZ3Reg(E_EXPANSIONBASE,0x10)<<8 | ReadZ3Reg(E_EXPANSIONBASE,0x14);

    if (manf != 0x0000 && manf != 0xffff) {
        if ((type & ERT_TYPEMASK) == ERT_ZORROII) return BT_Z2;
        if ((type & ERT_TYPEMASK) == ERT_ZORROIII) return BT_Z3_AT_Z2;
    }
    type = ReadZ3Reg(E_Z3EXPBASE,0x00);
    manf = ReadZ3Reg(E_Z3EXPBASE,0x10)<<8 | ReadZ3Reg(E_Z3EXPBASE,0x14);

    if (manf != 0x0000 && manf != 0xffff)
        if ((type & ERT_TYPEMASK) == ERT_ZORROIII) return BT_Z3_AT_Z3;

    return BT_NONE;
}

/* This function fills the configuration ROM field of the given
    ConfigDev, form the given address, based on the appropriate mapping
    rules. */

void InitZ3ROM(base,cd)
WORD *base;
struct ConfigDev *cd;
{
    struct ExpansionRom *rom;

    rom = &cd->cd_Rom;

    rom->er_Type = ReadZ3Reg(base,0x00);
    rom->er_Product = ReadZ3Reg(base,0x04);
    rom->er_Flags = ReadZ3Reg(base,0x08);
    rom->er_Reserved03 = ReadZ3Reg(base,0x0c);
    rom->er_Manufacturer = ReadZ3Reg(base,0x10)<< 8 | ReadZ3Reg(base,0x14);
    rom->er_SerialNumber = ReadZ3Reg(base,0x18)<<24 | ReadZ3Reg(base,0x1c)<<16 |
    ReadZ3Reg(base,0x20)<< 8 | ReadZ3Reg(base,0x24);
    rom->er_InitDiagVec = ReadZ3Reg(base,0x28)<< 8 | ReadZ3Reg(base,0x2c);
    rom->er_Reserved0c = ReadZ3Reg(base,0x30);
    rom->er_Reserved0d = ReadZ3Reg(base,0x34);
    rom->er_Reserved0e = ReadZ3Reg(base,0x38);
    rom->er_Reserved0f = ReadZ3Reg(base,0x3c);
}

/* This function locates a Zorro III board.  If it finds one in the
    unconfigured state, it allocates a ConfigDev for it, fills in the
    configuration data, and returns that ConfigDev.  Otherwise it returns
    NULL.  It knows the basics of what to do should it encounter a
    Zorro II board sitting in the way. */

struct ConfigDev *FindZ3Board() {
    struct ConfigDev *cd;

    while (TRUE) {
        if (!(cd = AllocConfigDev())) return NULL;
```

```
        switch (TypeOfPIC()) {
            case BT_NONE        :
                FreeConfigDev(cd);
                return NULL;
            case BT_Z2      :
                PRVB("FOUND: Z2 Board, Configuring");
                if (!ReadExpansionRom(E_EXPANSIONBASE,cd))
                    if (!ConfigBoard(E_EXPANSIONBASE,cd))
                        AddConfigDev(cd);
                anyone = TRUE;
                break;
            case BT_Z3_AT_Z2    :
                PRVB("FOUND: Z3 Board (Z2 Space), Configuring");
                InitZ3ROM(E_EXPANSIONBASE,cd);
                cd->cd_BoardAddr = (APTR)E_EXPANSIONBASE;
                anyone = TRUE;
                return cd;
            case BT_Z3_AT_Z3    :
                PRVB("FOUND: Z3 Board (Z3 Space), Configuring");
                InitZ3ROM(E_Z3EXPBASE,cd);
                cd->cd_BoardAddr = (APTR)E_Z3EXPBASE;
                anyone = TRUE;
                return cd;
        }
    }
    return NULL;
}

/* ====================================================================== */

/* These functions are involved in configuring a Zorro III board. */

/* This function writes the configuration address stored in the given
    ConfigDev to the board in the proper way. */

void WriteCfgAddr(base,cd)
UWORD *base;
struct ConfigDev *cd;
{
    UBYTE nybreg[4],bytereg[2],*bytebase;
    UWORD wordreg,i,*wordbase;

    wordreg = (((ULONG)cd->cd_BoardAddr)>>16);
    bytereg[0] = (UBYTE)(wordreg & 0x00ff);
    bytereg[1] = (UBYTE)(wordreg >> 8);
    nybreg[0] = ((bytereg[0] & 0x0f)<<4);
    nybreg[1] = ((bytereg[0] & 0xf0));
    nybreg[2] = ((bytereg[1] & 0x0f)<<4);
    nybreg[3] = ((bytereg[1] & 0xf0));

    bytebase = (UBYTE *)(base + 22);
    wordbase = (UWORD *)(base + 22);

    if (base == (UWORD *)E_EXPANSIONBASE) {
        (*(bytebase+0x002)) = nybreg[2];
        (*(bytebase+0x000)) = bytereg[1];
        (*(bytebase+0x006)) = nybreg[1];
        (*(bytebase+0x004)) = bytereg[0];
    } else {
        (*(bytebase+0x104)) = nybreg[0];
        (*(bytebase+0x004)) = bytereg[0];
        (*(bytebase+0x100)) = nybreg[2];
        (*(wordbase+0x000)) = wordreg;
    }
}


/* This function automatically sizes the configured board described by the
    given ConfigDev.  It doesn't attempt to preserve the contents. */

void AutoSizeBoard(cd)
struct ConfigDev *cd;
{
    ULONG i,realmax,logicalsize = 0;

    realmax = ((ULONG)cd->cd_SlotSize) * E_Z3SLOTSIZE + (ULONG)cd->cd_BoardAddr;

    for (i = (ULONG)cd->cd_BoardAddr; i < realmax; i += E_Z3ASIZEINC)
        *((ULONG *)i) = 0;

    for (i = (ULONG)cd->cd_BoardAddr; i < realmax; i += E_Z3ASIZEINC) {
        if (*((ULONG *)i) != 0) break;
        *((ULONG *)i) = 0xaa5500ff;
        if (*((ULONG *)i) != 0xaa5500ff) break;
        logicalsize += E_Z3ASIZEINC;
    }
    cd->cd_BoardSize = (APTR)logicalsize;
}
```

```
/* This function configures a Zorro III board, based on the initialization
   data in its ConfigDev structure. */

void ConfigZ3Board(cd)
struct ConfigDev *cd;
{
    APTR base = cd->cd_BoardAddr;
    UWORD sizecode,extended,subsize;
    ULONG physsize,logsize;
    char *memname;

    /* First examine the physical sizing of the board. */

    sizecode = cd->cd_Rom.er_Type & ERT_MEMSIZE;
    extended = ((cd->cd_Rom.er_Flags & ERFF_EXTENDED) != 0);

    physsize = BoardSize[extended][sizecode];

    cd->cd_BoardAddr = (APTR)Z3Space;
    cd->cd_BoardSize = (APTR)physsize;
    cd->cd_SlotAddr = (Z3Space-E_Z3EXPSTART)/E_Z3SLOTSIZE;
    cd->cd_SlotSize = ((physsize/E_Z3SLOTSIZE)>0)?(physsize/E_Z3SLOTSIZE):1;
    Z3Space += cd->cd_SlotSize * E_Z3SLOTSIZE;

    /* Next, process the sub-size, if any. */

    if (subsize = (cd->cd_Flags & ERF_SUBMASK))
        cd->cd_BoardSize = (APTR)SubSize[subsize];

    if (verbose) {
        printf("  BOARD STATS:");
        printf("                ADDRESS: $%lx",cd->cd_BoardAddr);
        if (cd->cd_BoardSize)
            printf("                   SIZE: $%lx",cd->cd_BoardSize);
        else
            printf("                   SIZE: AUTOMATIC => ");
    }

    /* Now, configure the board. */

    WriteCfgAddr(base,cd);
    if (!cd->cd_BoardSize) {
        AutoSizeBoard(cd);
        printf("$%lx",cd->cd_BoardSize);
    }

    if (cd->cd_BoardSize && (cd->cd_Rom.er_Type & ERTF_MEMLIST)) {
        strcpy(memname = (char *)AllocMem(20L,MEMF_CLEAR),"Zorro III Memory");
        AddMemList(cd->cd_BoardSize,MEMF_FAST|MEMF_PUBLIC,10,cd->cd_BoardAddr,memname);
    }

    AddConfigDev(cd);
}

/* ====================================================================== */

/* This is the main program. */

void main(argc,argv)
int argc;
char *argv[];
{
    int i;
    struct ConfigDev *cd;

    if (!(ExpansionBase = (struct ExpansionBase *)OpenLibrary("expansion.library",0L))) {
        printf("Error: Can't open "expansion.library"");
        exit(10);
    }
    if (argc > 1)
        for (i = 1; i < argc; ++i) switch (toupper(argv[i][0])) {
            case 'Q':  verbose = FALSE;   break;
            case 'V':  verbose = TRUE;    break;
    }

    while (cd = FindZ3Board()) ConfigZ3Board(cd);

    if (!anyone) PRVB("No PICs left to configure");
    CloseLibrary((struct ExpansionBase *)ExpansionBase);
}
```