

The Aquarius IIU Node: The Caches, the Address Translation Unit, and the VME bus Interface

Georges E. Smine and Vason P. Srin

Computer Science Division, EECS

University of California, Berkeley, CA 94720

ABSTRACT

This report describes the cache memory system of the Aquarius IIU node along with the address translation unit, and the VME interface. The Aquarius IIU node is designed for the parallel execution of Prolog. It is based on the VLSI-PLM Chip that runs the Warren Abstract Machine Instruction Set [STN87] (an intermediate language for Prolog). We have connected many of these nodes using a shared bus to form a multi, which has its own shared memory and snooping caches and is used as a backend Prolog engine to the host (SUN3/160). On every node, there are two controllers for data and instruction cache that cooperate to support Berkeley's snooping cache-lock state protocol, which minimizes bus traffic associated with locking blocks. The nodes share memory using the VME bus; the page faults and memory management are handled by the host. The components of the Aquarius IIU node have been simulated at the gate level.

Report Documentation Page			Form Approved OMB No. 0704-0188		
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE AUG 1989		2. REPORT TYPE		3. DATES COVERED 00-00-1989 to 00-00-1989	
4. TITLE AND SUBTITLE The Aquarius IIU Node: The Caches, the Address Translation Unit, and the VME bus Interface			5a. CONTRACT NUMBER		
			5b. GRANT NUMBER		
			5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S)			5d. PROJECT NUMBER		
			5e. TASK NUMBER		
			5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of California at Berkeley, Department of Electrical Engineering and Computer Sciences, Berkeley, CA, 94720			8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSOR/MONITOR'S ACRONYM(S)		
			11. SPONSOR/MONITOR'S REPORT NUMBER(S)		
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT This report describes the cache memory system of the Aquarius IIU node along with the address translation unit, and the VME interface. The Aquarius IIU node is designed for the parallel execution of Prolog. It is based on the VLSI-PLM Chip that runs the Warren Abstract Machine Instruction Set (an intermediate language for Prolog). We have connected many of these nodes using a shared bus to form a multi, which has its own shared memory and snooping caches and is used as a backend Prolog engine to the host (SUN3/160). On every node, there are two controllers for data and instruction cache that cooperate to support Berkeley's snooping cache-lock state protocol, which minimizes bus traffic associated with locking blocks. The nodes share memory using the VME bus; the page faults and memory management are handled by the host. The components of the Aquarius IIU node have been simulated at the gate level.					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT Same as Report (SAR)	18. NUMBER OF PAGES 82	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

1. Introduction

The principal motivation behind the Aquarius project is the design of a high-performance system that increases the processing speed of Prolog programs [DeS88]. Several factors in system design were shown to improve the performance of the execution of Prolog programs [Dob87]. Such factors are a tagged architecture for the main processor, a separation of data and instruction memory, and a cache protocol that would guarantee the correct functionality of the system in multi and parallel processing environments. This report describes the main components of the Aquarius IIU node, which is a component of a shared bus multiprocessor system (MULTI) [Bel85]. It is based on the VLSI-PLM processor [STN87]. It has its own instruction and data cache memory along with an interface to a bus on which it is connected.

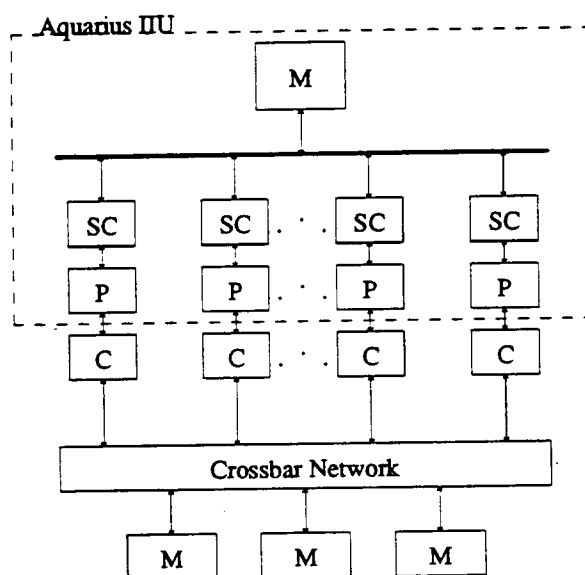


Figure 1: Aquarius IIU Multiprocessing System

For the purpose of simulation, four of these nodes are interfaced to a bus and share main memory. The four node multiprocessing system is a part of the two-tier memory based Aquarius II multiprocessor, shown in Figure 1. The architecture is interfaced to a host workstation, such as Sun 3, to start the program execution and to communicate the results. The components that constitute the Aquarius IIU node are the VLSI-PLM processor [STN87], the prefetcher [Bus89], the cache and snoop controllers [Bus89], the cache and snoop tag memories [Bus89], the two caches (instruction and data), the virtual-to-physical address

translation unit, and the VME bus interface. They are shown in Figure 2. The last three items are described in this report.

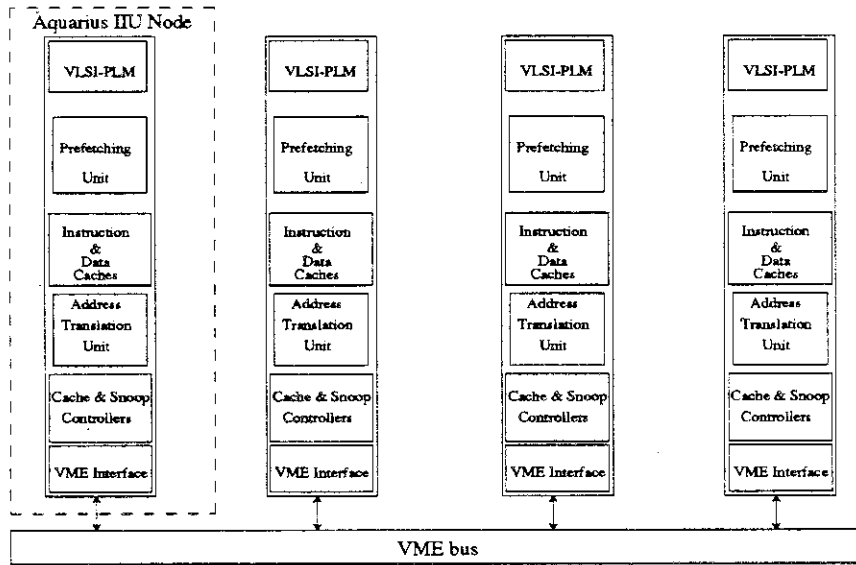


Figure 2: Aquarius IIU Node

The report is divided into 6 sections. Section 2 describes the cache memory. Section 3 presents the address translation unit. Section 4 discusses the VME bus interface. Section 5 describes the gate-level simulation of these various components, and Section 6 presents some conclusions.

2. Cache Description

The caches are designed so that they can be used as a component in Aquarius II or other more advanced systems.

The cache memory system is divided into an instruction and a data cache. Each is direct-mapped with a capacity of 4MB (20 bits for word address). It has been shown that direct-mapped caches were more efficient than other associative schemes for very large size caches [Hil88]. They are also more simple and less expensive than other schemes. Each of the two caches is organized in 256K blocks of four words each (16 bytes). The caches are virtually addressed. The two least significant bits of the processor's address are used to determine which of the four words gets loaded onto the processor's or prefetcher's bus. Thus, only

eighteen bits of the virtual address are used to access a block (excluding the bits for the word offset in a block). While the interface between the caches and the processor/prefetcher has a 32-bit bus, the cache uses a 128-bit bus to send blocks to the VME controller. This will eventually speed up the performance since four words get transferred from/to the VME buffer in one cycle. Since the VME bus uses byte addressing, the four least significant bits of the VME address are neglected in accessing the caches. The different addresses used to access the caches are shown below.

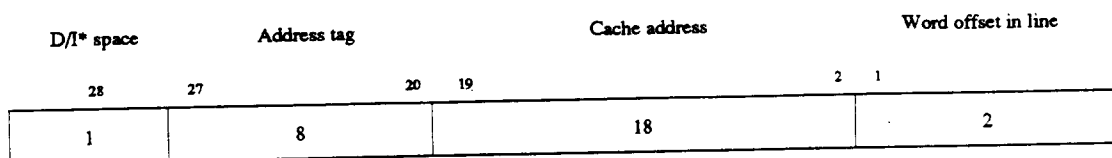


Figure 3: Virtual Address Format on Processor/Prefetcher Side

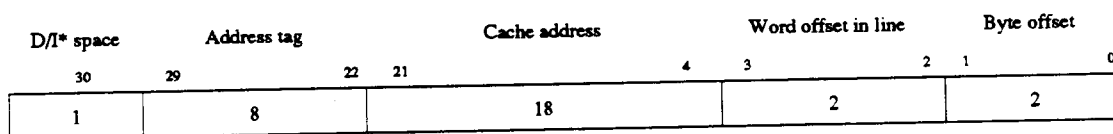


Figure 4: Virtual Address Format on VME Controller side

The tags are stored in a separate hardware unit from the cache content. They are stored in the same memory with the snoop and cache states fields. The 9-bit tag gets compared with corresponding address bits, and generates a hit signal if a match occurs. The comparators are in the same unit as the snoop and cache tags. The caches can be built using Dense-Pac Microsystems' CMOS SRAM DPS256Kx16 parts, which have fast access times of 25 ns. Every cache uses eight DPS256Kx16 chips, where a word is stored in every two chips. All other parts we used in the design are TTL and F parts, especially those that have very low propagation delays.

2.1. Instruction Cache

The instruction cache has the characteristics mentioned in the paragraph above. It can be accessed from the prefetcher unit, the PLM processor, and the VME controller. The cache is divided into four

blocks of RAM where each contains 32 bits. The cache's address lines accept bits <19:0> of the prefetcher or processor's address while it accepts bits <21:4> from the VME side. However bits <1:0> of the prefetcher/processor interface select one of the four 32-bit words at every access. This is done by hard-wiring the two bit lines to decoders, which will select one of the four words in a block that will be read or written to. Thus bits <19:2> of the prefetcher/processor side are used to access a word in a particular RAM. On the other hand, 18 bits from the VME interface address (bits <21:4>) are used since four words at a time are transferred between the VME interface and the caches. However, the main characteristic of the instruction cache is its interface with the prefetcher. The cache always has one of the 4 words enabled on the prefetcher output bus so the controller will not have to worry about enabling and disabling the data on these lines. Figure 5 illustrates the general block diagram of the instruction cache.

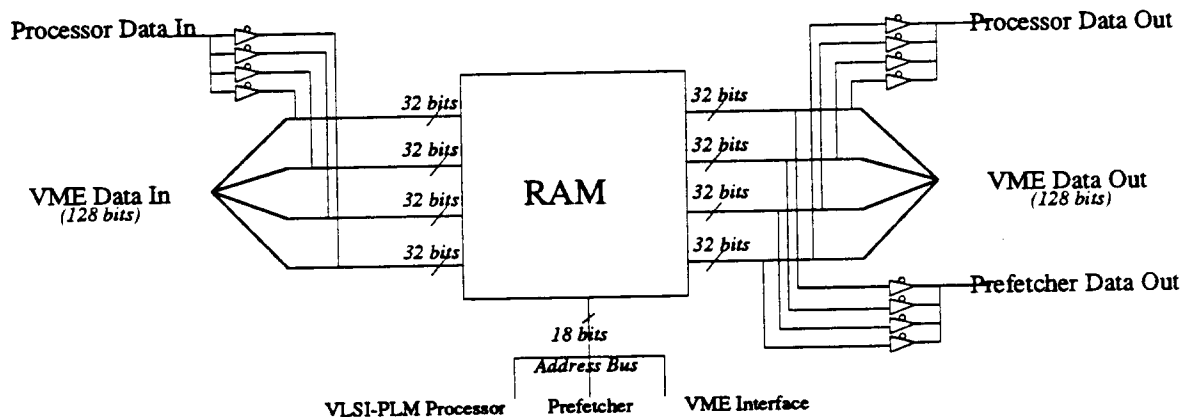


Figure 5: Instruction Cache Block Diagram

2.2. Data Cache

The design of the data cache is very similar to the instruction cache. The only difference is the absence of an interface with the prefetcher unit. However, the general architecture and the access methods are identical to the instruction cache. Figure 6 illustrates the general block diagram of the data cache.

3. Address Translation

The Aquarius IIU system contains 64 Megabytes of physical memory independent from the host,

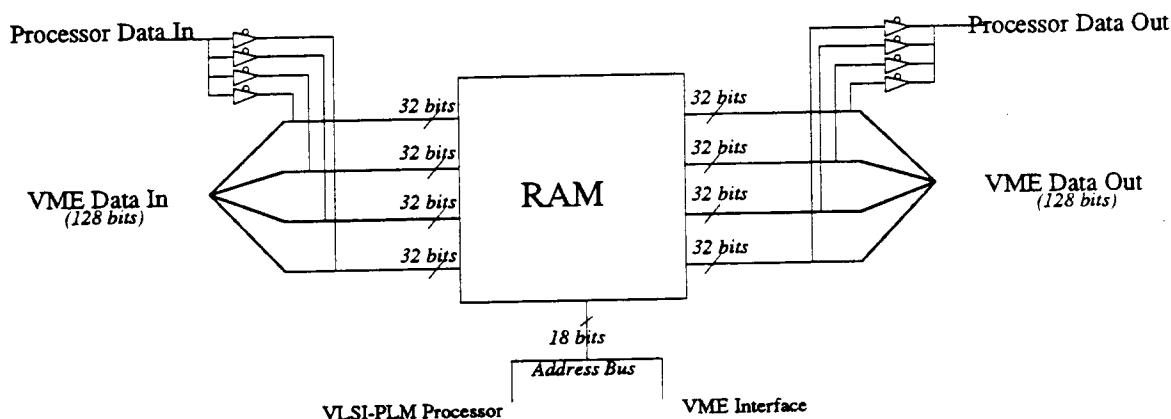


Figure 6: Data Cache Block Diagram

requiring a 26-bit byte addressable physical address. Translation from 29-bit word-addressable virtual addresses to these physical addresses is done in hardware in Aquarius IIU using a virtual-to-physical mapping function and inverted page tables.

Since the cache memory is virtually addressable, a page frame table is selected to map the virtual address space to the physical one. The inverted page table scheme was chosen because of its simplicity and because it matched our needs. A translation look-aside buffer (TLB) is not needed since cache memory is virtually addressable, and segment registers are not a better option since they become necessary when we need large virtual address space. The main components of the translation unit are the page frame table and the hash function, which translates the 29-bit virtual word address to a 26-bit physical byte address. The two least significant bits of the physical address (bits <1:0>) are set to zero since they will be referencing words in main memory (Note that main memory is byte addressable in contrast with the PLM which is word addressable). Although a physical address is needed only when a cache miss occurs, the translation is done in parallel with every cache access, without affecting the performance of the system. In case of a cache miss, the physical address will be ready if that page is in physical memory. If the page is not resident, the host interface handles the page fault, fetching the memory page from disk, and sending signals to update the page tables on every Aquarius IIU node.

3.1. Page Frame Table

The key data structure for address translation is the page frame table. Since the size of main memory is 64MB and the size of every page is 4KB, the page frame table needs to hold 16K page addresses at a time. A size of 4KB per page was chosen because we wanted to maintain large size pages while still not affecting the fill-up time of memory. Bigger pages would result in longer transfer time from disk to main memory, thus paralyzing the VME bus. In our opinion, 4KB was a good threshold value for a page size. The 14 most significant bits of the 26-bit physical address constitute the physical page address, and this address is generated directly from the hash function. The remaining 12 bits will reference a byte in main memory, but they do not have any use in the page frame table. Instead of using space from main memory for the page frame table to do the address translation from virtual to physical, we decided to have a separate table through which physical mapping will be done. The table will be stored in SRAM memory 16K deep and 28-bit wide. The contents of one entry in the table is shown in Figure 7.

Dirty	PID	Virtual Page Address
1 bit	8 bits	19 bits

Figure 7: Page Frame Table's Content Format

The page frame table we are using differs slightly from the regular inverted page table [ChM88]. In regular inverted page tables, there is a field called the chain in every entry that points to the next page entry. This entry is used if the virtual page address did not match the supplied address. The regular method requires three storage accesses, while ours requires only one. The disadvantage of ours is that we do not use chaining and a page fault is caused if the tag does not match the supplied virtual address. Since the size of our memory is big, the absence of chaining will not affect the performance. The description of the collision handling mechanism is in section 3.2. For every entry in the page frame table, the owner process ID is stored with the most significant 19 bits of the virtual address. The 19 bits of the virtual address in the table entry are checked with the supplied virtual address bits and a hit signal is generated if a match occurs. An extra bit is stored in the page table to indicate whether the page is dirty or not. The dirty bit is set after a *write* operation. If a dirty page is being replaced in memory, it will be flushed out to the external storage

before the replacement. Each Aquarius IIU node has a copy of the page frame table. The host modifies the page tables. All page frame tables are identical, and page faults are handled by the host. The external page table and the transfer of data from disk to shared memory is done by the host. The host also updates the external page table and transfers data from disk to shared memory.

The virtual to physical address translation is done in parallel with every cache access. Therefore, the physical address will be ready to be loaded to the VME controller one cycle after the cache access. A page fault can be handled in the cycle following the cache access, however, it will affect the cache lock state coherency protocol. When servicing a page fault, cache-to-cache transfers will not be serviced since the VME bus is used to transfer a page from disk to main memory. Regular snoop operations will still be able to be completed since our system uses extra pins on the VME bus in order to allow snoop communication independently from the data bus.

3.2. Hash Function

The translation of the virtual address to the physical one through the hash function is done in parallel with the access of the caches. After a hit in one of the caches, the cache controller will ignore the address translation done by the unit. The hash function reduces the 29-bit virtual address to a 26-bit physical address. The hash function will look at the 19 most significant bits of the processor's 29-bit virtual address (word address). The 19 most significant bits will be reduced to 14. The reduction circuitry is shown in Figure 8. Since the physical address references bytes, the two least significant bits <0:1> of the physical address are set to zeroes. In addition, bits <2:3>, which reference a word in a block, are set to zeroes because transfers to cache memory later are done block by block. Finally bits <2:9> (the offset of a block in a page) are passed to bits <4:11> of the physical address.

If we look closely at the XOR function on bits 23 & 24 we can draw the following conclusions:

- (a) The smallest range of virtual memory space between two colliding page addresses that map to the same physical location in main memory is 32MB.
- (b) The range between two colliding virtual addresses in main memory is not constant but varies

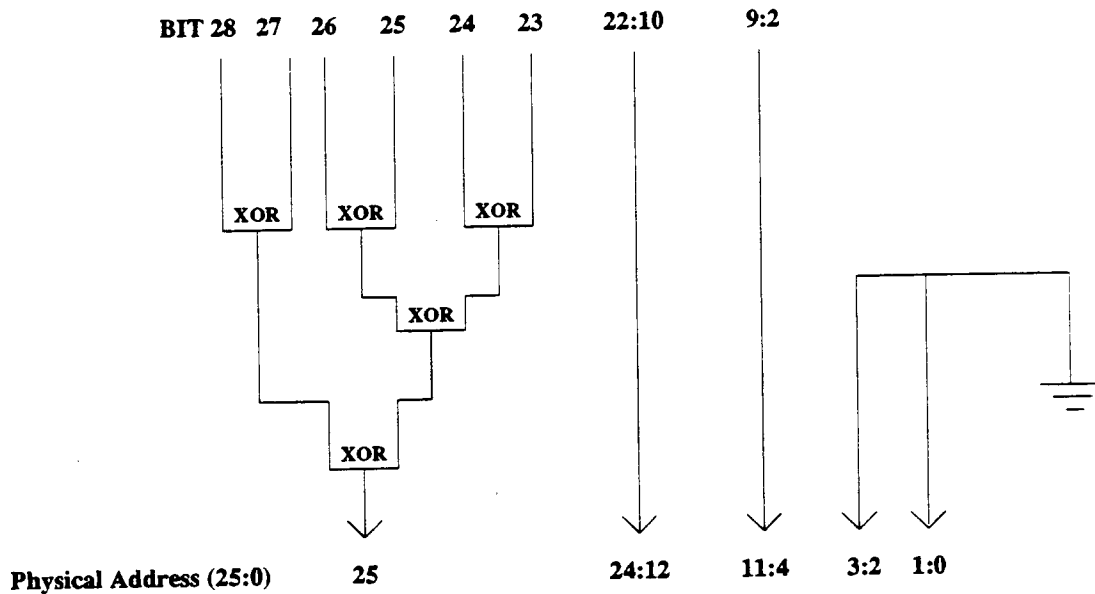


Figure 8: Hash Function Reduction Circuitry
between 32MB and 96MB.

- (c) The XOR scheme might reduce the page miss ratio compared to a regular direct mapped scheme, which gives a constant range of 64MB between consecutive addresses that map to the same physical location. On the average both give the same results. It could be proven that one implementation is better than the other depending on the test cases used.

Since we are using 64MB of main memory, we found that it was simple and efficient to use semi direct-mapped scheme. Direct-mapped memories have demonstrated high performance for large memories [Hil88]. Besides, such a scheme is the most convenient for a backend processor where an operating system is absent and programs are down-loaded to the system in a batch process.

4. VME Interface

The VME interface handles the communication over the VME bus. It can perform block *READ* and *WRITE* operations and allow cache-to-cache transfer in supporting the cache lock state coherency protocol [BiD86]. The VME interface consists of a data register (128-bit wide), an address register (32-bit wide), and a finite state machine. The interface transfers 128 bits of data in one cycle to the cache side, while it

takes four to complete the transfer of four words on the VME bus side. The interface uses extra pins on the J2 connector of the VME to support cache coherency between the nodes of the Aquarius IIU multiprocessor. These pins allow the state bits of every block to be shared between the Aquarius nodes. The interface is divided into three main blocks: the finite state machine, the data buffer, and the address buffer. The block diagram of the interface is shown in Figure 9.

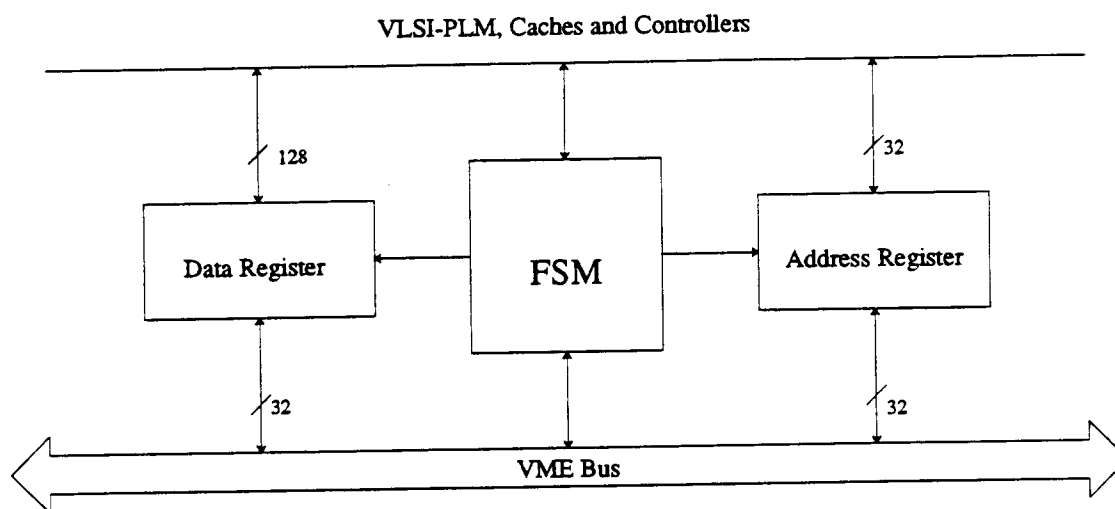


Figure 9: VME Interface

The finite state machine is based on two Programmable Logic Array (PAL20R8 and PAL20L8) where it has 31 states. The state machine is a simple Moore machine model. The state diagram is shown in Appendix A. The signals used in the VME interface and other parts of the Aquarius IIU are described in Appendix B. The output of the sequential machine (the PAL20R8) are fed into the combinational logic section of the FSM (the PAL20L8) where outputs are sent to the different control signals. Since such a fast PAL is not big enough to support all the logic of the controller, some external circuitry was added to the finite state machine. This extra logic includes three counters that will allow driving system resets signals, and detect AC failures and bus errors. The counters are needed for the detections of AC failures and driving system resets because the VME protocol [Mot85] imposed certain timing requirements on the response to such signals. For example, if the board enables SYSRESET* low, it must hold it down for 200 ms. The counter would allow the finite state machine to hold that signal down for 2000 cycles (100ns/cycle). On the other hand, the counter for the AC failure signal would make sure that AC failure is acknowledged to the FSM

only after 200 microseconds as the VME specifications require. Again, the counter for bus errors would allow the interface to try sixteen times for a block transfer before it accepts officially the bus error. In that case, it will send a signal to the cache controller indicating the error. In addition, circuitry that logically *ORs* the reset and fail signals helps in reducing the number of input signals to the PLA. Besides, special synchronizing logic is included in the FSM. This logic allows the transfer or the reception of a word per cycle during a block transfer. It is based on a two-phase non-overlapping clocks that latch a data-acknowledge signal, enable the load signals of the register, and load these registers at the positive edge of the inverted clock. The data-acknowledge (DTACK*) and the bus-error (BERR*) signals are synchronized first by the negative edge of the main clock. All the circuitry required for the correct operation of the VME electrical specifications is done in the block called VME-Signals.

The VME interface can handle five operations. The first two it can handle are a *READ* and a *WRITE* block request from the cache controllers. The third operation is a cache-to-cache transfer with concurrent *WRITE* to main memory, where a regular read request would be serviced from another Aquarius node (the source cache) through the VME. While transferring, the source node writes the block to main memory via the VME bus. Although the node that made the original *READ* request is the actual owner of the bus, another node does the writing to the bus. The operation is transparent to the bus (i.e., the VME assumes only one node is doing all operations). The fourth operation is the handling of a cache-to-cache transfer when the request comes from the snoop controller. In this case, the VME interface will transfer the block to the other node (bus owner), and it will write the block to main memory through the bus while the receiving node owns the VMEbus. The final operation is the request from the snoop controller asking for the bus in order to allow a snoop response. Such requests are made before a memory fetch and before a privilege raise.

The finite state machine of the interface transfers blocks of four words each in four cycles. It uses a two-phase clock for synchronizing the acceptance of the data-acknowledgement signals and loading or enabling the data from/on the VME bus. The interface will run at 10 MHz with two inverted non-overlapping clocks as shown in Figure 10.

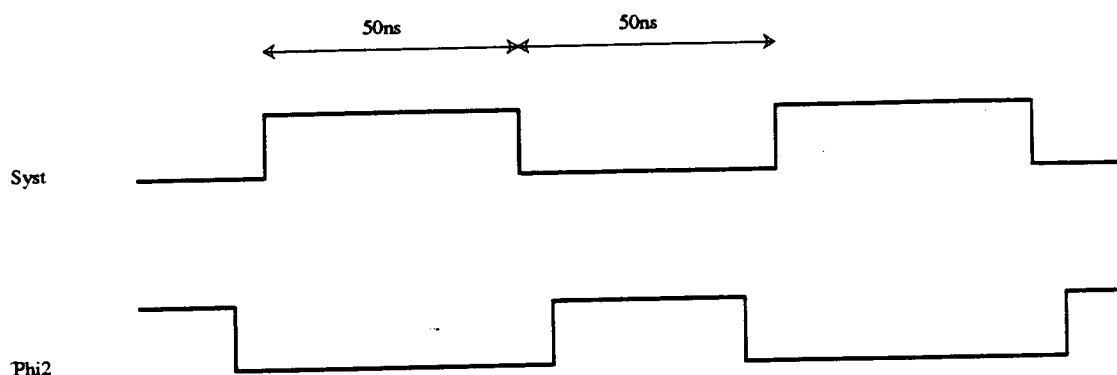


Figure 10: Non-Overlapping Clocking Scheme

As per the VME protocol [Mot85] the FSM would enable its data strobes at the beginning of the state (i.e., after the positive edge of Φ_0). Then, it will clock the data-acknowledged or bus error signals at the negative edge of that same clock. Thus, with the reception of the $DTACK^*$ signal, the data is loaded or enabled into/from the registers at the positive edge of the inverted clock, and the data strobes are then disabled. This scheme allows us to transfer one word in one cycle while still respecting the protocols of the bus.

The logic for the VME controller was created using Bdsyn, which translates a textual description of combinational logic into logic functions that carry out the desired function. Appendix C contains the Bdsyn description of the controller. The blif (Berkeley Logic Interchange Format) output of bdsyn was transferred to a PLA format by misII (a logic minimizer) and was directly used in defining the behavioral model (BLM) of the controller in order to simulate the finite state machine on the Mentor Graphics tools.

5. Simulation

The simulation of these components was done using Mentor Graphics' Quicksim. The design was simulated bottom-up, verifying each of the small components, followed by the integration of all its parts to create the full component. The schematics are shown in Appendix D. We used that methodology since we had to make sure that all low-level parts functioned correctly while simulating the high level ones. A set of "force" files were written to test the behavior of the various components. These "force" files stimulate the circuit by assigning a specific value to each input port at the desired time. These files along with their

corresponding results are shown in Appendix E. Force files were written for complex simulations such as the one for the interface's finite state machine. They end with the suffix ".do". The results of the simulations are shown in the files that end with the suffix "_simu". Debugging the finite state machine was simple since we used the Bdsyn language to define the state machine's behavior. The high level definition was translated into a pla format, which consists 1's and 0's for the AND/OR plane, and then ported on the Mentor Graphics' Tool set. The "pla" files were changed to match Mentor Graphics' BLM format. The caches and the components of the address translation unit were simulated interactively using Quicksim without the use of "force" files. Their simulation was simple and straight forward. The results of the simulation are in the files that end with the suffix "_sim". The result of the simulation was checked manually because of its simplicity. All of the simulation was ran at a clock frequency of 10 MHz, and all the components functioned properly at that frequency. The simulation at the system level was much more complex and required C programs to generate the "force" files along with "expect" files that compared the simulation results with the expected ones [Bus89]. The system level simulation verified the behavior of several units that interact with each others.

Acknowledgements

We would like to thank Darren Busing for his help in integrating all these components with his work to form the Aquarius IIU, and to Mike Carlton, Bruce Holmer and Tam Nguyen for answering all our questions. We are very thankful to Linda Bushnell for her comments, and the other members of the Aquarius project for their help, comments, and suggestions.

This research was partially sponsored by Defense Advanced Research Projects Agency (DoD) monitored by Office of Naval Research under Contract No. N00014-88-K-0579, and NCR Corporation in Dayton, Ohio. Equipment and other support for the project has been provided by DEC, NCR, Apollo, ESL, and Xenologic.

References

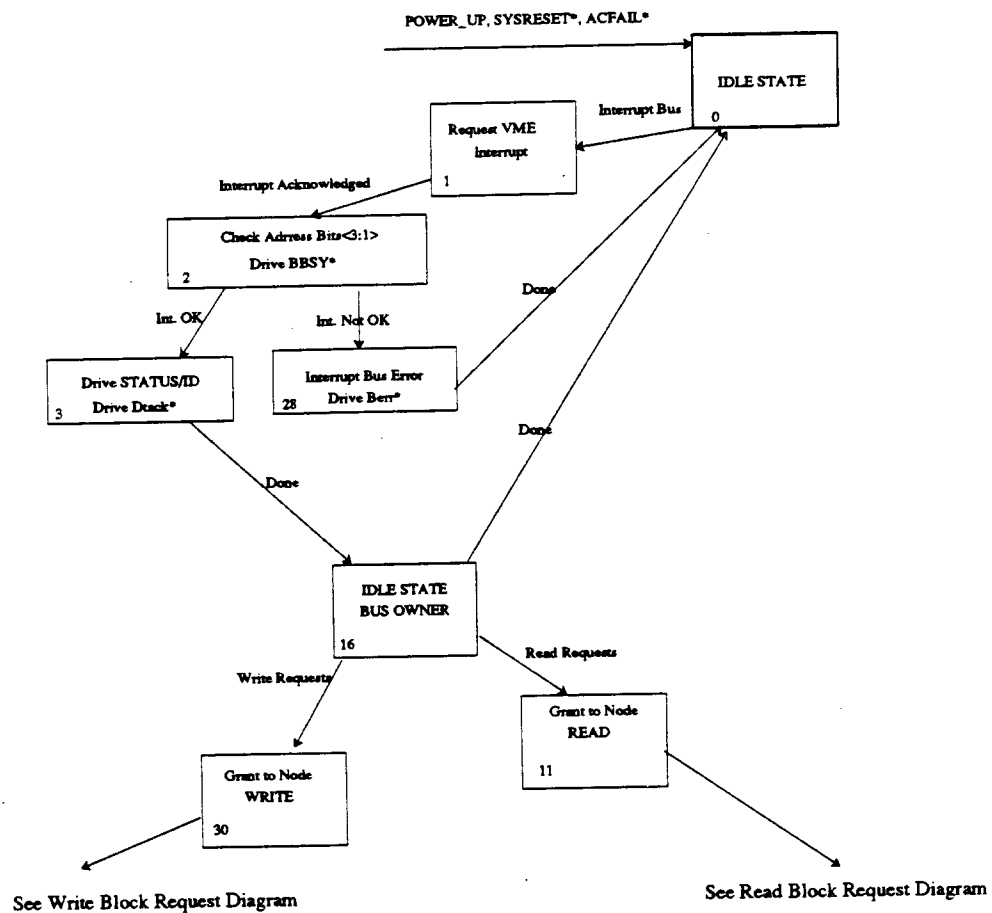
- [Bel85] C. G. Bell, Multis: A New Class of Multiprocessor Computers, *Science* 228 (April 16, 1985), 462-467.
- [BiD86] P. Bitar and A. Despain, Multiprocessor Cache Synchronization Issues, Innovations, Evolution, *Proceedings of the 13th Intl. Symposium on Computer Architecture*, Tokyo, Japan, June 1986, 424-433.
- [Bus89] D. R. Busing, The Aquarius IIU Node: The Prefetcher, and the Cache and Snoop Controllers, Master's Project Report UCB/ CSD 89/, Computer Science Division, University of California at Berkeley, August 1989.
- [ChM88] A. Chang and M. F. Mergen, 801 Storage: Architecture and Programming, *ACM Transactions on Computer Systems* 6, 1 (February 1988), 28-50.
- [DeS88] A. M. Despain and V. P. Srin, Multiprocessor Architecture Research for Prolog, *Proceedings of the State of California MICRO-1986 Report*, March 1988.
- [Dob87] T. Dobry, A High Performance Architecture for Prolog, Ph.D. Thesis UCB/ CSD 87/352, Computer Science Division, University of California at Berkeley, May 1987.
- [Hil88] M. D. Hill, A Case for Direct-Mapped Caches, *Computer*, December 1988, 25-40.
- [Mot85] Motorola Corporation, The VMEbus Specification C.1, October 1985.
- [STN87] V. Srin, J. Tam, T. Nguyen, C. Chen, A. Wei, J. Testa, Y. Patt and A. Despain, VLSI Implementation of a Prolog Processor, *Proceedings of the Stanford VLSI Conference*, March 1987.

Appendix A: State Diagrams

- (1) Every state is indexed by its number in the lower left corner.
- (2) Transitions take place when signals on the arrows are asserted. The state remains unchanged until the transition condition is met.
- (3) The finite state machine is based on the Moore machine model.
- (4) Active low signal: R_W^*
- (5) Active high signal: R_W
- (6) Actual signals are in upper case letters. The description of signals is in lower case.

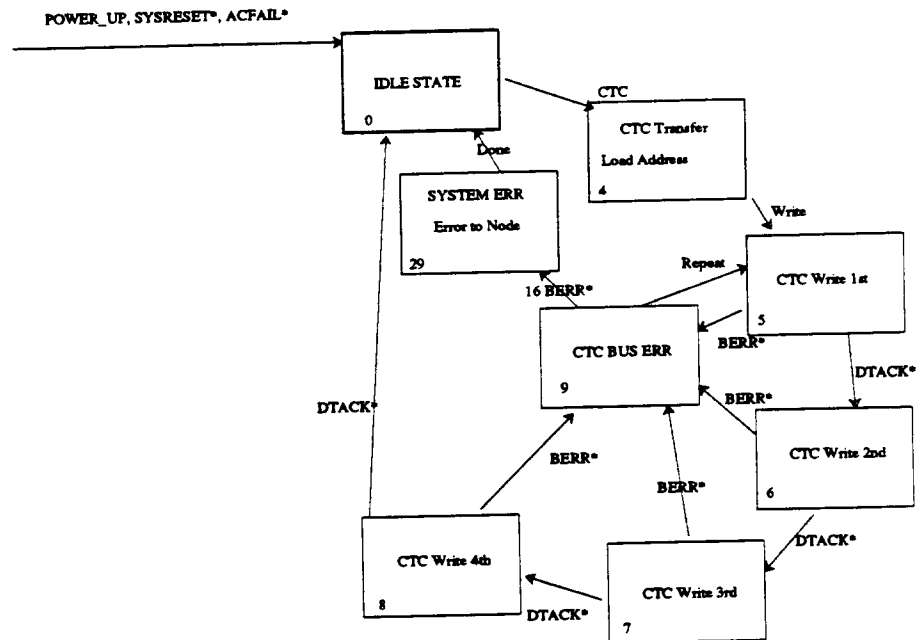
VME INTERFACE FSM STATE DIAGRAM

Bus Interrupt Request



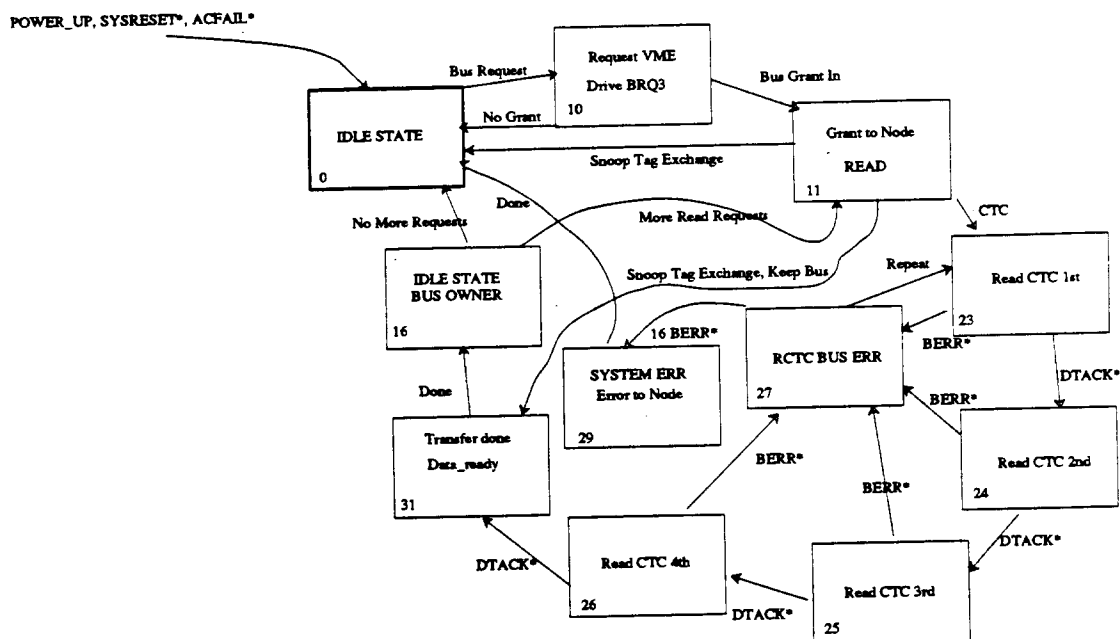
VME INTERFACE FSM STATE DIAGRAM

Cache-To-Cache Transfer WRITE



VME INTERFACE FSM STATE DIAGRAM

Cache-To-Cache Transfer READ



Appendix B: Signal Lists

Table B.1: Connections to the D-I Caches			
Signal	Direction	Connections	Purpose
PF_ADDR(0:19)	Input	Prefetcher	Address for instruction cache requests
P_ADDR(0:19)	Input	VLSI-PLM	Address for cache requests
V_ADDR(4:21)	Input	VME Interface	Address for writing into caches
PFA_OE*	Input	Prefetcher	OE for address coming from Prefetcher for I-cache
PA_OE*	Input	Cache Controller	OE for address coming from PLM
VA_OE*	Input	Cache Controller	OE for address coming from VME Interface
P_X_IN(0:31)	Input	VLSI-PLM	Data from MEMDAT bus of PLM
V_X_IN(0:127)	Input	VME Interface	Data from VME Interface
PF_X_OUT(0:31)	Output	Prefetcher	Data for prefetcher from I-cache
P_X_OUT(0:31)	Output	VLSI-PLM	Data to MEMDAT bus of PLM
V_X_OUT(0:127)	Output	VME Interface	Data to VME Interface
P_I_OE*	Input	Cache Controller	OE for data from PLM
V_I_OE*	Input	Cache Controller	OE for data from VME Interface
P_O_OE*	Input	Cache Controller	OE for data to PLM
V_O_OE*	Input	Cache Controller	OE for data to VME Interface
CE*	Input	Cache Controller	Chip enable signal
WE*	Input	Cache Controller	Write enable signal

Table B.2: Connections to the D-I Cache Tags			
Signal	Direction	Connections	Purpose
P_ADDR(2:28)	Input	VLSI-PLM	Address for cache requests
V_ADDR(4:30)	Input	VME Interface	Address from VME for Snoop response
SNP_TAG_IN(0:4)	Input	Snoop Controller	Coherency tags
CH_TAG_IN(0:4)	Input	Cache Controller	Coherency tags
SNP_TAG_OUT(0:4)	Output	Snoop Controller	Coherency tags
CH_TAG_OUT(0:4)	Output	Cache Controller	Coherency tags
S_WRITE	Input	Snoop Controller	Snoop write enable signal
C_WRITE	Input	Cache Controller	Cache write enable signal
S_HIT	Output	Snoop Controller	Cache hit signal
C_HIT	Output	Cache Controller	Cache hit signal
CE*	Input	Cache	Snoop Controller
WE*	Input	Cache	Snoop Controller

Table B.3: Connections to the Address Translation Unit			
Signal (& Connection)	I/O	To/From where	For what
PID_IN(0:7)	I	Host	Current process ID
PID_OUT(0:7)	O	Host	Process ID of page in memory
P_VIRT_ADDR(0:28)	I	VLSI-PLM	Address to be translated
AD_OE* (Address_output_enable)	I	D-I Caches	Selects address for translation and asserts result to VME
P*_V (VME_address_type)	I	Cache controller	Physical/virtual select
PG_HIT	O	Host	Page present
PG_DIRTY	O	Host	Page needs to be written to disk
R*_W	I	Host	Page table to be updated
VME_ADDR(0:31)	O	VME controller	Translation output

Table B.4: Connections to the VME Interface			
Signal (& Connection)	I/O	To/From where	For what
message_in(0:2)	I	Host	Incoming snoop response
message_out(0:2)	O	Host	Outgoing snoop response
bus_message_in(0:2)	I	BUS	Incoming snoop response
bus_message_out(0:2)	O	BUS	Outgoing snoop response
NODE_ADDR_IN(0:31)	I	Host	Address
NODE_ADDR_OUT(0:31)	O	Host	Address
VMEbus_AD_IN(0:31)	I	BUS	Address
VMEbus_AD_OUT(0:31)	O	BUS	Address
NODE_D_IN(0:127)	I	Host	Data
NODE_D_OUT(0:127)	O	Host	Data
VMEbus_D_IN(0:31)	I	Bus	Data
VMEbus_D_OUT(0:31)	O	Bus	Data
INT_REQ	I	Host	Interrupt Request
BUSREQ	I	Host	Bus Request
DATA_BLOCK	I	Host	Transfer Block
R_W*	I	Host	Read/Write*
CTC	I	Host	Cache-to-cache
BUSGRNT	O	Host	Bus grant signal
DATA_READY	O	Host	Data Ready signal
SYSTEM_ERROR*	O	Host	System Error signal
Phi0	I	Host	System Clock
Phi1	I	Host	2nd Clock
PWR_UP_RESET*	I	Host	Power up signal
BG(3:0)IN*	I	Bus	Bus grant in
BG(3:0)OUT*	O	Bus	Bus grant out
BBSY_IN*	I	Bus	Bus Busy in signal
BBSY*	O	Bus	Bus Busy out signal
BCLR*	I	Bus	Bus clear signal
BR3*	O	Bus	Bus Request signal
DS(1:0)IN*	I	Bus	Data Strobes in
AM_OUT(5:0)	O	Bus	Address Modifier Codes
AS*	O	Bus	Address strobe
DS(1:0)*	O	Bus	Data strobes out
WRITE*	O	Bus	Write signal
BERR*	I	Bus	Bus error in signal
BERR_OUT*	O	Bus	Bus error out signal
DTACK*	I	Bus	Data Acknowledge in signal
DTACK*_OUT*	O	Bus	Data Acknowledge out signal
IRQ2*	O	Bus	Interrupt Request signal
IACK*	I	Bus	Interrupt being serviced signal
IACKIN*	I	Bus	Interrupt Grant in signal
IACKOUT*	O	Bus	Interrupt Grant out signal
ACFAIL*	I	Bus	AC failure
SYSFAIL*	O	Bus	System failure
SYSRESET*	O	Bus	System reset

Appendix C: Bdsyn Files

89/08/14
12:53:53

vmefsm.bds

```
!
! VMEbus Interface
! with the Aquarius II node
! Finite State Machine Description
! State Machine
!
! Designed by Georges Smine
!
! Next State Specifications
!
Model vmefsm
!
! OUTGOING SIGNALS
!
! Outgoing Signals to Next State Register
!
! nextstate<5:0>
!
! INCOMING SIGNALS
!
! Incoming Signals for BOOT UP
!
! PWR_UP_RESET_, ! Pin#24
!
! Incoming Signals from the VME BUS
!
! IACK_, ! pin#23
! IACKIN_, ! pin#22
! BG3IN_, ! pin#21
! BCLR_, ! pin#20
! BBSY_IN_, ! pin#19
! DS0IN_, ! pin#18
! DS1IN_, ! pin#17
! SYSRESET_, ! pin#16
! ACFAIL_IN_, ! pin#15
! D_ACK_, ! pin#14
! BUS_ERR_, ! pin#13 Tells fsm that a bus error occurred, but does
! ! not stop trying until it gets a cache error
!
! Incoming Signals from CACHE & SNOOP CONTROLLERS
!
! INT_REQ_, ! pin#12 Interrupt request
! BUSREQ_, ! pin#11
! DATA_BLOCK_, ! pin#10 Tells fsm if there's a block transfer
! CTC_, ! pin#9 Tells fsm if there's a cache-to-cache transfer
! R_W_, ! pin#8 Tells fsm if it's a read or write request
! !!!RELEASE, Tells fsm if it should release bus
!
! Incoming Signals from BERR (Bus Error) Detection Circuitry
!
! CACHE_ERR_, ! pin#7 Tells fsm that a vme ttransfer has failed for 16 times
! ! Refer to BERR Circuitry
!
! Incoming Signals from Address Register for checking bits <3:1> of the addr.
!
! INT_OK_, ! pin#6 It means that the interrupt handler has given us the
! ! interrupt.
!
! Incoming Signals from Current State Register
! Since BDSYN does not recognize sequential logic, I will represent
! the current state as a regular input
!
```

```
currentstate<5:0>; ! pin# 5:0
!
! CONSTANT
!
! OFF = 0,
! ON = 1,
! OFF_ = 1,
! ON_ = 0,
!
! STATE VALUES
!
! IDLE STATE = 0,
! IDLE_STATE_BUS_OWNER = 16,
! SYS_ERR = 29,
!
! Interrupt Procedures
!
! REQUEST_INTERRUPT = 1,
! IRQ_CHECK_A0_3 = 2,
! IRQ_DRIVE_STATUS_ID = 3,
! INTERRUPT_BERR = 28,
!
! Servicing a CTC
!
! CTC_TRANSFER_WRITE = 4,
! CTC_WR_1 = 5,
! CTC_WR_2 = 6,
! CTC_WR_3 = 7,
! CTC_WR_4 = 8,
! TRANSFER_DONE = 31,
! CTC_WR_BERR = 9,
!
! Servicing a Read Request
! for snoop tags or regular block
!
! REQUEST_VME = 10,
! GRANT_TO_NODE_R = 11,
! GRANT_TO_NODE_W = 30,
! READ_1 = 12,
! READ_2 = 13,
! READ_3 = 14,
! READ_4 = 15,
! READ_BERR = 17,
!
! Servicing a Write Request
! for snoop tags or regular block
!
! WRITE_1 = 18,
! WRITE_2 = 19,
! WRITE_3 = 20,
! WRITE_4 = 21,
! WRITE_BERR = 22,
!
! Receiving a cache to cache transfer
!
! RCTC_1 = 23,
! RCTC_2 = 24,
! RCTC_3 = 25,
! RCTC_4 = 26,
! RCTC_BERR = 27;
!
! ROUTINE main;
```

89/08/14
12:53:53

vmefsm.bds

2

```
SELECT currentstate FROM
[IDLE_STATE]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (int_req) AND (lack_EQL OFF_)
    then nextstate = REQUEST_INTERRUPT
  else if (ctc) then nextstate = CTC_TRANSFER_WRITE
  else if (busreq) AND (bbsy_in_EQL OFF_) then
    nextstate = REQUEST_VME
  else nextstate = IDLE_STATE;
end;

[REQUEST_INTERRUPT]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (lackin_EQL ON_) then
    nextstate = IRQ_CHECK_A0_3
  else nextstate = REQUEST_INTERRUPT;
end;

[IRQ_CHECK_A0_3]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (ds0in_EQL ON_) AND (ds1in_EQL ON_) then
    if (int_ok) then
      nextstate = IRQ_DRIVE_STATUS_ID
    else if (bus_err_) then
      nextstate = INTERRUPT_BERR
    else nextstate = IRQ_CHECK_A0_3;
end;

[IRQ_DRIVE_STATUS_ID]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else ! drive DTACK OUT* low
    ! and STATUS_ID low
    if (ds0in_EQL OFF_) AND (ds1in_EQL OFF_) then
      nextstate = IDLE_STATE_BUS_OWNER
    else nextstate = IRQ_DRIVE_STATUS_ID;
end;

[INTERRUPT_BERR]: begin
  !drive BERR_OUT* low
  nextstate = IDLE_STATE;
end;

[CTC_TRANSFER_WRITE]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (r_w_EQL OFF_) then nextstate = CTC_WR_1
  else nextstate = IDLE_STATE;
end;

[CTC_WR_1]: begin
  !drive ds* low...
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (d_ack_EQL ON_) then nextstate = CTC_WR_2
  else if (bus_err_EQL ON_) then
    nextstate = CTC_WR_BERR
  else nextstate = CTC_WR_1;
end;
```

```
[CTC_WR_2]: begin
  !drive ds* low...
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (d_ack_EQL ON_) then nextstate = CTC_WR_3
  else if (bus_err_EQL ON_) then
    nextstate = CTC_WR_BERR
  else nextstate = CTC_WR_2;
end;

[CTC_WR_3]: begin
  !drive ds* low...
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (d_ack_EQL ON_) then nextstate = CTC_WR_4
  else if (bus_err_EQL ON_) then
    nextstate = CTC_WR_BERR
  else nextstate = CTC_WR_3;
end;

[CTC_WR_4]: begin
  !drive ds* low...
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (d_ack_EQL ON_) then nextstate = IDLE_STATE
  else if (bus_err_EQL ON_) then
    nextstate = CTC_WR_BERR
  else nextstate = CTC_WR_4;
end;

[CTC_WR_BERR]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (cache_err) then nextstate = SYS_ERR
  else nextstate = CTC_WR_1;
end;

[IDLE_STATE_BUS_OWNER]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (bclr_EQL ON_) then nextstate = IDLE_STATE
  else if (busreq) then
    if (r_w_) then
      nextstate = GRANT_TO_NODE_R
    else nextstate = GRANT_TO_NODE_W
  else nextstate = IDLE_STATE;
end;

[SYS_ERR]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else ! drive SYSTEM_ERROR_low in this state
    nextstate = IDLE_STATE;
end;

[REQUEST_VME]: begin
  ! drive br3* low and wait
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (BG3IN_EQL ON_) then
    !drive bbsy_low & bus granted
    if (r_w_) then nextstate = GRANT_TO_NODE_R
    else nextstate = GRANT_TO_NODE_W
  else nextstate = IDLE_STATE;
```

89/08/14
12:53:53

vmefsm.bds

3

```
end;

[GRANT_TO_NODE_R]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else ! drive bbsy_low and bus granted
    if (busreq_EQL OFF) then nextstate = IDLE_STATE
    else if (ctc) then nextstate = RCTC_1
      else if (data_block_EQL OFF) then
        nextstate = TRANSFER_DONE
      else nextstate = READ_1;
    end;
  end;

[GRANT_TO_NODE_W]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else ! drive bbsy_low and bus granted
    if (busreq_EQL OFF) then nextstate = IDLE_STATE
    else if (data_block_EQL OFF) then
      nextstate = TRANSFER_DONE
    else nextstate = WRITE_1;
  end;

[READ_1]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (d_ack_EQL ON_) then nextstate = READ_2
  else if (bus_err_EQL ON_) then
    nextstate = READ_BERR
  else nextstate = READ_1;
end;

[READ_2]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (d_ack_EQL ON_) then nextstate = READ_3
  else if (bus_err_EQL ON_) then
    nextstate = READ_BERR
  else nextstate = READ_2;
end;

[READ_3]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (d_ack_EQL ON_) then nextstate = READ_4
  else if (bus_err_EQL ON_) then
    nextstate = READ_BERR
  else nextstate = READ_3;
end;

[READ_4]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (d_ack_EQL ON_) then
    nextstate = TRANSFER_DONE
  else if (bus_err_EQL ON_) then
    nextstate = READ_BERR
  else nextstate = READ_4;
end;

[TRANSFER_DONE]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else nextstate = IDLE_STATE_BUS_OWNER;
```

```
end;

[READ_BERR]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (cache_err) then nextstate = SYS_ERR
  else nextstate = READ_1;
end;

[WRITE_1]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (d_ack_EQL ON_) then nextstate = WRITE_2
  else if (bus_err_EQL ON_) then
    nextstate = WRITE_BERR
  else nextstate = WRITE_1;
end;

[WRITE_2]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (d_ack_EQL ON_) then nextstate = WRITE_3
  else if (bus_err_EQL ON_) then
    nextstate = WRITE_BERR
  else nextstate = WRITE_2;
end;

[WRITE_3]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (d_ack_EQL ON_) then nextstate = WRITE_4
  else if (bus_err_EQL ON_) then
    nextstate = WRITE_BERR
  else nextstate = WRITE_3;
end;

[WRITE_4]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (d_ack_EQL ON_) then
    nextstate = TRANSFER_DONE
  else if (bus_err_EQL ON_) then
    nextstate = WRITE_BERR
  else nextstate = WRITE_4;
end;

[WRITE_BERR]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (cache_err) then nextstate = SYS_ERR
  else nextstate = WRITE_1;
end;

[RCTC_1]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
  OR (ACFAIL_IN_EQL ON_) then nextstate = RCTC_2
  else if (bus_err_EQL ON_) then
    nextstate = RCTC_BERR
  else nextstate = RCTC_1;
end;

[RCTC_2]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
```

89/08/14
12:53:53

vmefsm.bds

4

```
OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
else if (d_ack_EQL ON_) then nextstate = RCTC_3
  else if (bus_err_EQL ON_) then
    nextstate = RCTC_BERR
  else nextstate = RCTC_2;
end;

[RCTC_3]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (d_ack_EQL ON_) then nextstate = RCTC_4
    else if (bus_err_EQL ON_) then
      nextstate = RCTC_BERR
    else nextstate = RCTC_3;
end;

[RCTC_4]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (d_ack_EQL ON_) then
    nextstate = TRANSFER_DONE
  else if (bus_err_EQL ON_) then
    nextstate = RCTC_BERR
  else nextstate = RCTC_4;
end;

[RCTC_BERR]: begin
  if (pwr_up_reset_EQL ON_) OR (sysreset_EQL ON_)
OR (ACFAIL_IN_EQL ON_) then nextstate = IDLE_STATE
  else if (cache_err) then nextstate = SYS_ERR
  else nextstate = RCTC_1;
end;

ENDSELECT;
ENDROUTINE;
ENDMODEL;
```

89/08/14
12:54:18

vmeoutputs.bds

```
!
! VMEbus Interface
! with the Aquarius II node
! Outputs of Finite State Machine
!
! Designed by Georges Smine
!
```

Model vmeoutput

! OUTGOING SIGNALS

! Outgoing Signals to the VME bus

```
!
! BRQ3, !Pin#29 It requests the bus on line 3. Refer to VME specs.
! IRQ, !Pin#28 It request an interrupt on line 2. IRQ2*, Refer to VME specs.
! BUSY, !Pin#27 This signal is BBSY* on the VME, it goes through a buffer
! like many other signals in order to respect the VMEbus
! electrical specifications. It is set whenever the bus is
! is used and owned by the node.
! WRITE_, !Pin#26 It signals a read or a write operation.
! AS_, !Pin#25 Address Strobe signalling that address is valid on the bus
! DS0_, !Pin#24 Data Strobe ....
! DS1_, !Pin#23 .....
! DTACK_OUT, !Pin#22 Send data ack out after an interrupt. Signal inverted
! BERR_OUT, !Pin#21 Send bus error signal out after an interrupt. Outside PAL
```

! Outgoing Signals to VME DATA Buffer (internal to the VME interface)

```
! LD_<3:0>, !Pin#20:17 Load signals that load any or all of the 32bit data regs
! of the 128-bit data register.
! VEN_<3:0>, !Pin#16:13 Output enable signals from the VME bus to each of the
! 4 32-bit regs.
! EN_<3:0>, !Pin#12:9 OE signals from each of the 4 regs to the VME bus.
! INT_OE_, !Pin#8 OE signal, that enable the STATUS/ID on the bus after an
! interrupt.
! VME_AD_LD_, !Pin#7 OE signal from the VME address bus to the address reg.
! VME_AD_EN_, !Pin#6 OE signal from the address reg to the VME address bus.
! LDA_, !Pin#5 Load Address Signal.
```

! Outgoing Signals to the Cache & Snoop Controllers

```
! BUSGRNT, !Pin#4 Bus granted
! DATA_READY, !Pin#3 Data is ready in regs for read op, or write op is done
! SYSTEM_ERROR_, !Pin#2 16 consecutive BERR* signals. Something is wrong.
```

! Outgoing signals to Internal Cktry of FSM
! Used for selecting which AM_OUT code

```
! AM_24bit_MODE_, !Pin#1
! AM_32bit_MODE_, !Pin#0
```

! Incoming Signals

```
! CURRENTSTATE<5:0>;
```

! CONSTANT

```
! OFF = 0,
! ON = 1,
! OFF_ = 1,
```

```
ON_ = 0,
```

! STATE VALUES

```
! IDLE_STATE = 0,
! IDLE_BUS_OWNER = 16,
! SYS_ERR = 29,
```

! Interrupt Procedures

```
! REQUEST_INTERRUPT = 1,
! IRQ_CHECK_A0_3 = 2,
! IRQ_DRIVE_STATUS_ID = 3,
! INTERRUPT_BERR = 28,
```

! Servicing a CTC

```
! CTC_TRANSFER_WRITE = 4,
! CTC_WR_1 = 5,
! CTC_WR_2 = 6,
! CTC_WR_3 = 7,
! CTC_WR_4 = 8,
! CTC_WR_BERR = 9,
```

! Servicing a Read Request
! for snoop tags or regular block

```
! REQUEST_VME = 10,
! GRANT_TO_NODE_R = 11,
! GRANT_TO_NODE_W = 30,
! READ_1 = 12,
! READ_2 = 13,
! READ_3 = 14,
! READ_4 = 15,
! TRANSFER_DONE = 31,
! READ_BERR = 17,
```

! Servicing a Write Request
! for snoop tags or regular block

```
! WRITE_1 = 18,
! WRITE_2 = 19,
! WRITE_3 = 20,
! WRITE_4 = 21,
! WRITE_BERR = 22,
```

! Receiving a cache to cache transfer

```
! RCTC_1 = 23,
! RCTC_2 = 24,
! RCTC_3 = 25,
! RCTC_4 = 26,
! RCTC_BERR = 27;
```

! ROUTINE main;

```
SELECT currentstate FROM
```

```
[IDLE_STATE]: begin
```

```
BRQ3 = 0; !!!
IRQ = 0;
BUSY = 0;
WRITE_ = 1;
AS_ = 1;
```

89/08/14
12:54:18

vmeoutputs.bds

2

```
DS0_ = 1;
DS1_ = 1;
DTACK_OUT = 0;
BERR_OUT = 0;
LD <3:0> = 1111#2;
VEN <3:0> = 1111#2;
EN <3:0> = 1111#2;
INT_OE = 1;
VME_AD_LD = 1;
VME_AD_EN = 1;
LDA_ = 1;
BUSGRNT = 0;
DATA_READY = 0;
SYSTEM_ERROR = 1;
AM_24bit_MODE = 1;
AM_32bit_MODE = 1;

end;

[REQUEST_INTERRUPT]: begin
BRQ3 = 0; !!!
IRQ = 1;
BUSY = 0;
WRITE_ = 1;
AS_ = 1;
DS0_ = 1;
DS1_ = 1;
DTACK_OUT = 0;
BERR_OUT = 0;
LD <3:0> = 1111#2;
VEN <3:0> = 1111#2;
EN <3:0> = 1111#2;
INT_OE = 1;
VME_AD_LD = 1;
VME_AD_EN = 1;
LDA_ = 1;
BUSGRNT = 0;
DATA_READY = 0;
SYSTEM_ERROR = 1;
AM_24bit_MODE = 1;
AM_32bit_MODE = 1;

end;

[IRQ_CHECK_A0_3]: begin
BRQ3 = 0; !!!
IRQ = 1;
BUSY = 1;
WRITE_ = 1;
AS_ = 1;
DS0_ = 1;
DS1_ = 1;
DTACK_OUT = 0;
BERR_OUT = 0;
LD <3:0> = 1111#2;
VEN <3:0> = 1111#2;
EN <3:0> = 1111#2;
INT_OE = 1;
VME_AD_LD = 0;
VME_AD_EN = 1;
LDA_ = 1;
BUSGRNT = 1;
DATA_READY = 0;
SYSTEM_ERROR = 1;
AM_24bit_MODE = 1;
AM_32bit_MODE = 1;
```

```
end;

[IRQ_DRIVE_STATUS_ID]: begin
BRQ3 = 0; !!!
IRQ = 1;
BUSY = 1;
WRITE_ = 1;
AS_ = 1;
DS0_ = 1;
DS1_ = 1;
DTACK_OUT = 1;
BERR_OUT = 0;
LD <3:0> = 1111#2;
VEN <3:0> = 1111#2;
EN <3:0> = 1111#2;
INT_OE = 0;
VME_AD_LD = 1;
VME_AD_EN = 1;
LDA_ = 1;
BUSGRNT = 1;
DATA_READY = 0;
SYSTEM_ERROR = 1;
AM_24bit_MODE = 1;
AM_32bit_MODE = 1;

end;

[INTERRUPT_BERR]: begin
BRQ3 = 0; !!!
IRQ = 1;
BUSY = 1;
WRITE_ = 1;
AS_ = 1;
DS0_ = 1;
DS1_ = 1;
DTACK_OUT = 0;
BERR_OUT = 1;
LD <3:0> = 1111#2;
VEN <3:0> = 1111#2;
EN <3:0> = 1111#2;
INT_OE = 1;
VME_AD_LD = 1;
VME_AD_EN = 1;
LDA_ = 1;
BUSGRNT = 0;
DATA_READY = 0;
SYSTEM_ERROR = 0;
AM_24bit_MODE = 1;
AM_32bit_MODE = 1;

end;

[CTC_TRANSFER_WRITE]: begin
BRQ3 = 0; !!!
IRQ = 0;
BUSY = 0;
WRITE_ = 1;
AS_ = 1;
DS0_ = 1;
DS1_ = 1;
DTACK_OUT = 0;
BERR_OUT = 0;
LD <3:0> = 0000#2;
VEN <3:0> = 1111#2;
EN <3:0> = 1111#2;
INT_OE = 1;
```

89/08/14
12:54:18

vmeoutputs.bds

3

```
VME_AD_LD = 0;
VME_AD_EN = 1;
LDA = 0;
BUSGRNT = 0;
DATA_READY = 0;
SYSTEM_ERROR = 1;
AM_24bit_MODE = 1;
AM_32bit_MODE = 0;

end;

[CTC_WR_1]: begin
    BRQ3 = 0; !!!
    IRQ = 0;
    BUSY = 0;
    WRITE = 1;
    AS = 1;
    DS0 = 0;
    DS1 = 0;
    DTACK_OUT = 0;
    BERR_OUT = 0;
    LD <3:0> = 1111#2;
    VEN <3:0> = 1111#2;
    EN <3:0> = 0111#2;
    INT_OE = 1;
    VME_AD_LD = 1;
    VME_AD_EN = 1;
    LDA = 1;
    BUSGRNT = 0;
    DATA_READY = 0;
    SYSTEM_ERROR = 1;
    AM_24bit_MODE = 1;
    AM_32bit_MODE = 0;

end;

[CTC_WR_2]: begin
    BRQ3 = 0; !!!
    IRQ = 0;
    BUSY = 0;
    WRITE = 1;
    AS = 1;
    DS0 = 0;
    DS1 = 0;
    DTACK_OUT = 0;
    BERR_OUT = 0;
    LD <3:0> = 1111#2;
    VEN <3:0> = 1111#2;
    EN <3:0> = 1011#2;
    INT_OE = 1;
    VME_AD_LD = 1;
    VME_AD_EN = 1;
    LDA = 1;
    BUSGRNT = 0;
    DATA_READY = 0;
    SYSTEM_ERROR = 1;
    AM_24bit_MODE = 1;
    AM_32bit_MODE = 0;

end;

[CTC_WR_3]: begin
    BRQ3 = 0; !!!
    IRQ = 0;
    BUSY = 0;
    WRITE = 1;
    AS = 1;
```

```
DS0 = 0;
DS1 = 0;
DTACK_OUT = 0;
BERR_OUT = 0;
LD <3:0> = 1111#2;
VEN <3:0> = 1111#2;
EN <3:0> = 1101#2;
INT_OE = 1;
VME_AD_LD = 1;
VME_AD_EN = 1;
LDA = 1;
BUSGRNT = 0;
DATA_READY = 0;
SYSTEM_ERROR = 1;
AM_24bit_MODE = 1;
AM_32bit_MODE = 0;

end;

[CTC_WR_4]: begin
    BRQ3 = 0; !!!
    IRQ = 0;
    BUSY = 0;
    WRITE = 1;
    AS = 1;
    DS0 = 0;
    DS1 = 0;
    DTACK_OUT = 0;
    BERR_OUT = 0;
    LD <3:0> = 1111#2;
    VEN <3:0> = 1111#2;
    EN <3:0> = 1110#2;
    INT_OE = 1;
    VME_AD_LD = 1;
    VME_AD_EN = 1;
    LDA = 1;
    BUSGRNT = 0;
    DATA_READY = 0;
    SYSTEM_ERROR = 1;
    AM_24bit_MODE = 1;
    AM_32bit_MODE = 0;

end;

[CTC_WR_BERR]: begin
    BRQ3 = 0;
    IRQ = 0;
    BUSY = 0;
    WRITE = 1;
    AS = 1;
    DS0 = 1;
    DS1 = 1;
    DTACK_OUT = 0;
    BERR_OUT = 0;
    LD <3:0> = 1111#2;
    VEN <3:0> = 1111#2;
    EN <3:0> = 1111#2;
    INT_OE = 1;
    VME_AD_LD = 1;
    VME_AD_EN = 1;
    LDA = 1;
    BUSGRNT = 0;
    DATA_READY = 0;
    SYSTEM_ERROR = 1;
    AM_24bit_MODE = 1;
    AM_32bit_MODE = 0;
```

89/08/14
12:54:18

vmeoutputs.bds

4

end;

```
[IDLE_BUS_OWNER]: begin
    BRQ3 = 0; !!!
    IRQ = 0;
    BUSY = 1;
    WRITE_ = 1;
    AS_ = 1;
    DS0_ = 1;
    DS1_ = 1;
    DTACK_OUT = 0;
    BERR_OUT = 0;
    LD_<3:0> = 1111#2;
    VEN_<3:0> = 1111#2;
    EN_<3:0> = 1111#2;
    INT_OE_ = 1;
    VME_AD_LD_ = 1;
    VME_AD_EN_ = 1;
    LDA_ = 1;
    BUSGRNT = 1;
    DATA_READY = 0;
    SYSTEM_ERROR = 1;
    AM_24bit_MODE_ = 1;
    AM_32bit_MODE_ = 1;
end;
```

end;

```
[SYS_ERR]: begin
    BRQ3 = 0; !!!
    IRQ = 0;
    BUSY = 0;
    WRITE_ = 1;
    AS_ = 1;
    DS0_ = 1;
    DS1_ = 1;
    DTACK_OUT = 0;
    BERR_OUT = 0;
    LD_<3:0> = 1111#2;
    VEN_<3:0> = 1111#2;
    EN_<3:0> = 1111#2;
    INT_OE_ = 1;
    VME_AD_LD_ = 1;
    VME_AD_EN_ = 1;
    LDA_ = 1;
    BUSGRNT = 1;
    DATA_READY = 0;
    SYSTEM_ERROR = 0;
    AM_24bit_MODE_ = 1;
    AM_32bit_MODE_ = 1;
end;
```

end;

```
[REQUEST_VME]: begin
    BRQ3 = 1; !!!
    IRQ = 0;
    BUSY = 0;
    WRITE_ = 1;
    AS_ = 1;
    DS0_ = 1;
    DS1_ = 1;
    DTACK_OUT = 0;
    BERR_OUT = 0;
    LD_<3:0> = 1111#2;
    VEN_<3:0> = 1111#2;
    EN_<3:0> = 1111#2;
    INT_OE_ = 1;
end;
```

```
VME_AD_LD_ = 1;
VME_AD_EN_ = 1;
LDA_ = 0;
BUSGRNT = 0;
DATA_READY = 0;
SYSTEM_ERROR = 1;
AM_24bit_MODE_ = 1;
AM_32bit_MODE_ = 0;
```

end;

```
[GRANT_TO_NODE_R]: begin
    BRQ3 = 0;
    IRQ = 0;
    BUSY = 1;
    WRITE_ = 1;
    AS_ = 0;
    DS0_ = 1;
    DS1_ = 1;
    DTACK_OUT = 0;
    BERR_OUT = 0;
    LD_<3:0> = 1111#2;
    VEN_<3:0> = 1111#2;
    EN_<3:0> = 1111#2;
    INT_OE_ = 1;
    VME_AD_LD_ = 1;
    VME_AD_EN_ = 0;
    LDA_ = 0;
    BUSGRNT = 1;
    DATA_READY = 0;
    SYSTEM_ERROR = 1;
    AM_24bit_MODE_ = 1;
    AM_32bit_MODE_ = 0;
end;
```

end;

```
[GRANT_TO_NODE_W]: begin
    BRQ3 = 0; !!!
    IRQ = 0;
    BUSY = 1;
    WRITE_ = 0;
    AS_ = 0;
    DS0_ = 1;
    DS1_ = 1;
    DTACK_OUT = 0;
    BERR_OUT = 0;
    LD_<3:0> = 0000#2;
    VEN_<3:0> = 1111#2;
    EN_<3:0> = 1111#2;
    INT_OE_ = 1;
    VME_AD_LD_ = 1;
    VME_AD_EN_ = 0;
    LDA_ = 0;
    BUSGRNT = 1;
    DATA_READY = 0;
    SYSTEM_ERROR = 1;
    AM_24bit_MODE_ = 1;
    AM_32bit_MODE_ = 0;
end;
```

end;

```
[READ_1]: begin
    BRQ3 = 0;
    IRQ = 0;
    BUSY = 1;
    WRITE_ = 1;
    AS_ = 0;
end;
```

89/08/14
12:54:18

vmeoutputs.bds

5

```
DS0_ = 0;
DS1_ = 0;
DTACK_OUT = 0;
BERR_OUT = 0;
LD_<3:0> = 0111#2;
VEN_<3:0> = 0111#2;
EN_<3:0> = 1111#2;
INT_OE_ = 1;
VME_AD_LD_ = 1;
VME_AD_EN_ = 0;
LDA_ = 1;
BUSGRNT = 1;
DATA_READY = 0;
SYSTEM_ERROR_ = 1;
AM_24bit_MODE_ = 1;
AM_32bit_MODE_ = 0;

end;

[READ_2]: begin
BRQ3 = 0; !!!
IRQ = 0;
BUSY = 1;
WRITE_ = 1;
AS_ = 0;
DS0_ = 0;
DS1_ = 0;
DTACK_OUT = 0;
BERR_OUT = 0;
LD_<3:0> = 1011#2;
VEN_<3:0> = 1011#2;
EN_<3:0> = 1111#2;
INT_OE_ = 1;
VME_AD_LD_ = 1;
VME_AD_EN_ = 0;
LDA_ = 1;
BUSGRNT = 1;
DATA_READY = 0;
SYSTEM_ERROR_ = 1;
AM_24bit_MODE_ = 1;
AM_32bit_MODE_ = 0;

end;

[READ_3]: begin
BRQ3 = 0; !!!
IRQ = 0;
BUSY = 1;
WRITE_ = 1;
AS_ = 0;
DS0_ = 0;
DS1_ = 0;
DTACK_OUT = 0;
BERR_OUT = 0;
LD_<3:0> = 1101#2;
VEN_<3:0> = 1101#2;
EN_<3:0> = 1111#2;
INT_OE_ = 1;
VME_AD_LD_ = 1;
VME_AD_EN_ = 0;
LDA_ = 1;
BUSGRNT = 1;
DATA_READY = 0;
SYSTEM_ERROR_ = 1;
AM_24bit_MODE_ = 1;
AM_32bit_MODE_ = 0;
```

```
end;

[READ_4]: begin
BRQ3 = 0; !!!
IRQ = 0;
BUSY = 1;
WRITE_ = 1;
AS_ = 0;
DS0_ = 0;
DS1_ = 0;
DTACK_OUT = 0;
BERR_OUT = 0;
LD_<3:0> = 1110#2;
VEN_<3:0> = 1110#2;
EN_<3:0> = 1111#2;
INT_OE_ = 1;
VME_AD_LD_ = 1;
VME_AD_EN_ = 0;
LDA_ = 1;
BUSGRNT = 1;
DATA_READY = 0;
SYSTEM_ERROR_ = 1;
AM_24bit_MODE_ = 1;
AM_32bit_MODE_ = 0;

end;

[TRANSFER_DONE]: begin
BRQ3 = 0; !!!
IRQ = 0;
BUSY = 1;
WRITE_ = 1;
AS_ = 1;
DS0_ = 1;
DS1_ = 1;
DTACK_OUT = 0;
BERR_OUT = 0;
LD_<3:0> = 1111#2;
VEN_<3:0> = 1111#2;
EN_<3:0> = 1111#2;
INT_OE_ = 1;
VME_AD_LD_ = 1;
VME_AD_EN_ = 0;
LDA_ = 1;
BUSGRNT = 1;
DATA_READY = 1;
SYSTEM_ERROR_ = 1;
AM_24bit_MODE_ = 1;
AM_32bit_MODE_ = 1;

end;

[READ_BERR]: begin
BRQ3 = 0; !!!
IRQ = 0;
BUSY = 1;
WRITE_ = 1;
AS_ = 1;
DS0_ = 1;
DS1_ = 1;
DTACK_OUT = 0;
BERR_OUT = 0;
LD_<3:0> = 1111#2;
VEN_<3:0> = 1111#2;
EN_<3:0> = 1111#2;
INT_OE_ = 1;
```

89/08/14
12:54:18

vmeoutputs.bds

6

```
VME_AD_LD_ = 1;
VME_AD_EN_ = 1;
LDA_ = 1;
BUSGRNT = 1;
DATA_READY = 0;
SYSTEM_ERROR_ = 1;
AM_24bit_MODE_ = 1;
AM_32bit_MODE_ = 0;

end;

[WRITE_1]: begin
    BRQ3 = 0; !!!
    IRQ = 0;
    BUSY = 1;
    WRITE_ = 0;
    AS_ = 0;
    DS0_ = 0;
    DS1_ = 0;
    DTACK_OUT = 0;
    BERR_OUT = 0;
    LD_<3:0> = 1111#2;
    VEN_<3:0> = 1111#2;
    EN_<3:0> = 0111#2;
    INT_OE_ = 1;
    VME_AD_LD_ = 1;
    VME_AD_EN_ = 0;
    LDA_ = 1;
    BUSGRNT = 1;
    DATA_READY = 0;
    SYSTEM_ERROR_ = 1;
    AM_24bit_MODE_ = 1;
    AM_32bit_MODE_ = 0;

end;

[WRITE_2]: begin
    BRQ3 = 0; !!!
    IRQ = 0;
    BUSY = 1;
    WRITE_ = 0;
    AS_ = 0;
    DS0_ = 0;
    DS1_ = 0;
    DTACK_OUT = 0;
    BERR_OUT = 0;
    LD_<3:0> = 1111#2;
    VEN_<3:0> = 1111#2;
    EN_<3:0> = 1011#2;
    INT_OE_ = 1;
    VME_AD_LD_ = 1;
    VME_AD_EN_ = 0;
    LDA_ = 1;
    BUSGRNT = 1;
    DATA_READY = 0;
    SYSTEM_ERROR_ = 1;
    AM_24bit_MODE_ = 1;
    AM_32bit_MODE_ = 0;

end;

[WRITE_3]: begin
    BRQ3 = 0; !!!
    IRQ = 0;
    BUSY = 1;
    WRITE_ = 0;
    AS_ = 0;
```

```
DS0_ = 0;
DS1_ = 0;
DTACK_OUT = 0;
BERR_OUT = 0;
LD_<3:0> = 1111#2;
VEN_<3:0> = 1111#2;
EN_<3:0> = 1101#2;
INT_OE_ = 1;
VME_AD_LD_ = 1;
VME_AD_EN_ = 0;
LDA_ = 1;
BUSGRNT = 1;
DATA_READY = 0;
SYSTEM_ERROR_ = 1;
AM_24bit_MODE_ = 1;
AM_32bit_MODE_ = 0;

end;

[WRITE_4]: begin
    BRQ3 = 0; !!!
    IRQ = 0;
    BUSY = 1;
    WRITE_ = 0;
    AS_ = 0;
    DS0_ = 0;
    DS1_ = 0;
    DTACK_OUT = 0;
    BERR_OUT = 0;
    LD_<3:0> = 1111#2;
    VEN_<3:0> = 1111#2;
    EN_<3:0> = 1110#2;
    INT_OE_ = 1;
    VME_AD_LD_ = 1;
    VME_AD_EN_ = 0;
    LDA_ = 1;
    BUSGRNT = 1;
    DATA_READY = 0;
    SYSTEM_ERROR_ = 1;
    AM_24bit_MODE_ = 1;
    AM_32bit_MODE_ = 0;

end;

[WRITE_BERR]: begin
    BRQ3 = 0; !!!
    IRQ = 0;
    BUSY = 1;
    WRITE_ = 0;
    AS_ = 1;
    DS0_ = 1;
    DS1_ = 1;
    DTACK_OUT = 0;
    BERR_OUT = 0;
    LD_<3:0> = 1111#2;
    VEN_<3:0> = 1111#2;
    EN_<3:0> = 1111#2;
    INT_OE_ = 1;
    VME_AD_LD_ = 1;
    VME_AD_EN_ = 1;
    LDA_ = 1;
    BUSGRNT = 1;
    DATA_READY = 0;
    SYSTEM_ERROR_ = 1;
    AM_24bit_MODE_ = 1;
    AM_32bit_MODE_ = 1;
```

89/08/14
12:54:18

vmeoutputs.bds

end;

[RCTC_1]: begin

```
BRQ3 = 0; !!!  
IRQ = 0;  
BUSY = 1;  
WRITE = 0;  
AS = 0;  
DS0 = 1;  
DS1 = 1;  
DTACK_OUT = 0;  
BERR_OUT = 0;  
LD <3:0> = 0111#2;  
VEN <3:0> = 0111#2;  
EN <3:0> = 1111#2;  
INT_OE = 1;  
VME_AD_LD = 1;  
VME_AD_EN = 0;  
LDA = 1;  
BUSGRNT = 1;  
DATA_READY = 0;  
SYSTEM_ERROR = 1;  
AM_24bit_MODE = 1;  
AM_32bit_MODE = 1;
```

end;

[RCTC_2]: begin

```
BRQ3 = 0; !!!  
IRQ = 0;  
BUSY = 1;  
WRITE = 0;  
AS = 0;  
DS0 = 1;  
DS1 = 1;  
DTACK_OUT = 0;  
BERR_OUT = 0;  
LD <3:0> = 1011#2;  
VEN <3:0> = 1011#2;  
EN <3:0> = 1111#2;  
INT_OE = 1;  
VME_AD_LD = 1;  
VME_AD_EN = 0;  
LDA = 1;  
BUSGRNT = 1;  
DATA_READY = 0;  
SYSTEM_ERROR = 1;  
AM_24bit_MODE = 1;  
AM_32bit_MODE = 1;
```

end;

[RCTC_3]: begin

```
BRQ3 = 0; !!!  
IRQ = 0;  
BUSY = 1;  
WRITE = 0;  
AS = 0;  
DS0 = 1;  
DS1 = 1;  
DTACK_OUT = 0;  
BERR_OUT = 0;  
LD <3:0> = 1101#2;  
VEN <3:0> = 1101#2;  
EN <3:0> = 1111#2;  
INT_OE = 1;
```

```
VME_AD_LD = 1;  
VME_AD_EN = 0;  
LDA = 1;  
BUSGRNT = 1;  
DATA_READY = 0;  
SYSTEM_ERROR = 1;  
AM_24bit_MODE = 1;  
AM_32bit_MODE = 1;
```

end;

[RCTC_4]: begin

```
BRQ3 = 0; !!!  
IRQ = 0;  
BUSY = 1;  
WRITE = 0;  
AS = 0;  
DS0 = 1;  
DS1 = 1;  
DTACK_OUT = 0;  
BERR_OUT = 0;  
LD <3:0> = 1110#2;  
VEN <3:0> = 1110#2;  
EN <3:0> = 1111#2;  
INT_OE = 1;  
VME_AD_LD = 1;  
VME_AD_EN = 0;  
LDA = 1;  
BUSGRNT = 1;  
DATA_READY = 0;  
SYSTEM_ERROR = 1;  
AM_24bit_MODE = 1;  
AM_32bit_MODE = 1;
```

end;

[RCTC_BERR]: begin

```
BRQ3 = 0; !!!  
IRQ = 0;  
BUSY = 1;  
WRITE = 0;  
AS = 1;  
DS0 = 1;  
DS1 = 1;  
DTACK_OUT = 0;  
BERR_OUT = 0;  
LD <3:0> = 1111#2;  
VEN <3:0> = 1111#2;  
EN <3:0> = 1111#2;  
INT_OE = 1;  
VME_AD_LD = 1;  
VME_AD_EN = 1;  
LDA = 1;  
BUSGRNT = 1;  
DATA_READY = 0;  
SYSTEM_ERROR = 1;  
AM_24bit_MODE = 1;  
AM_32bit_MODE = 1;
```

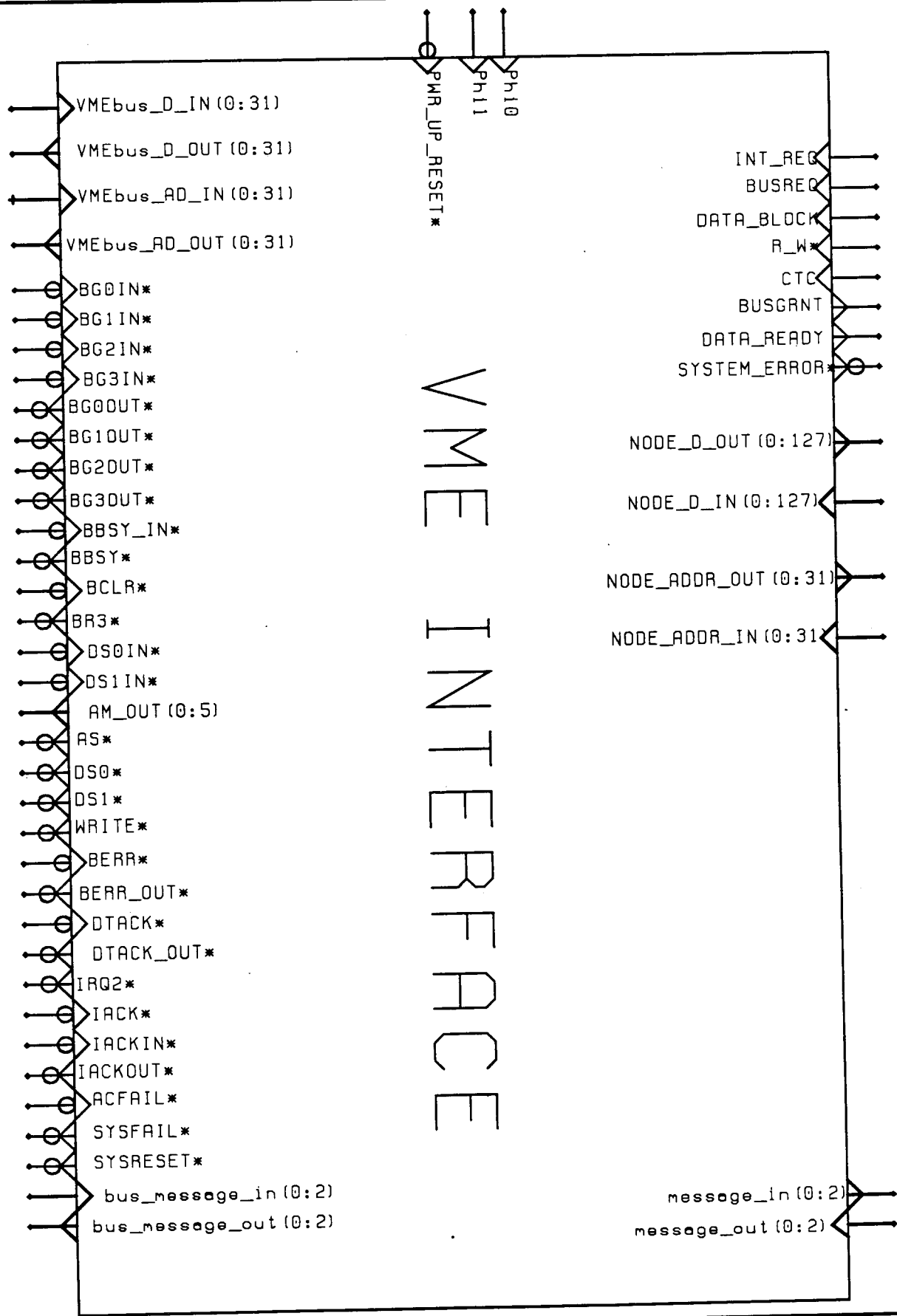
end;

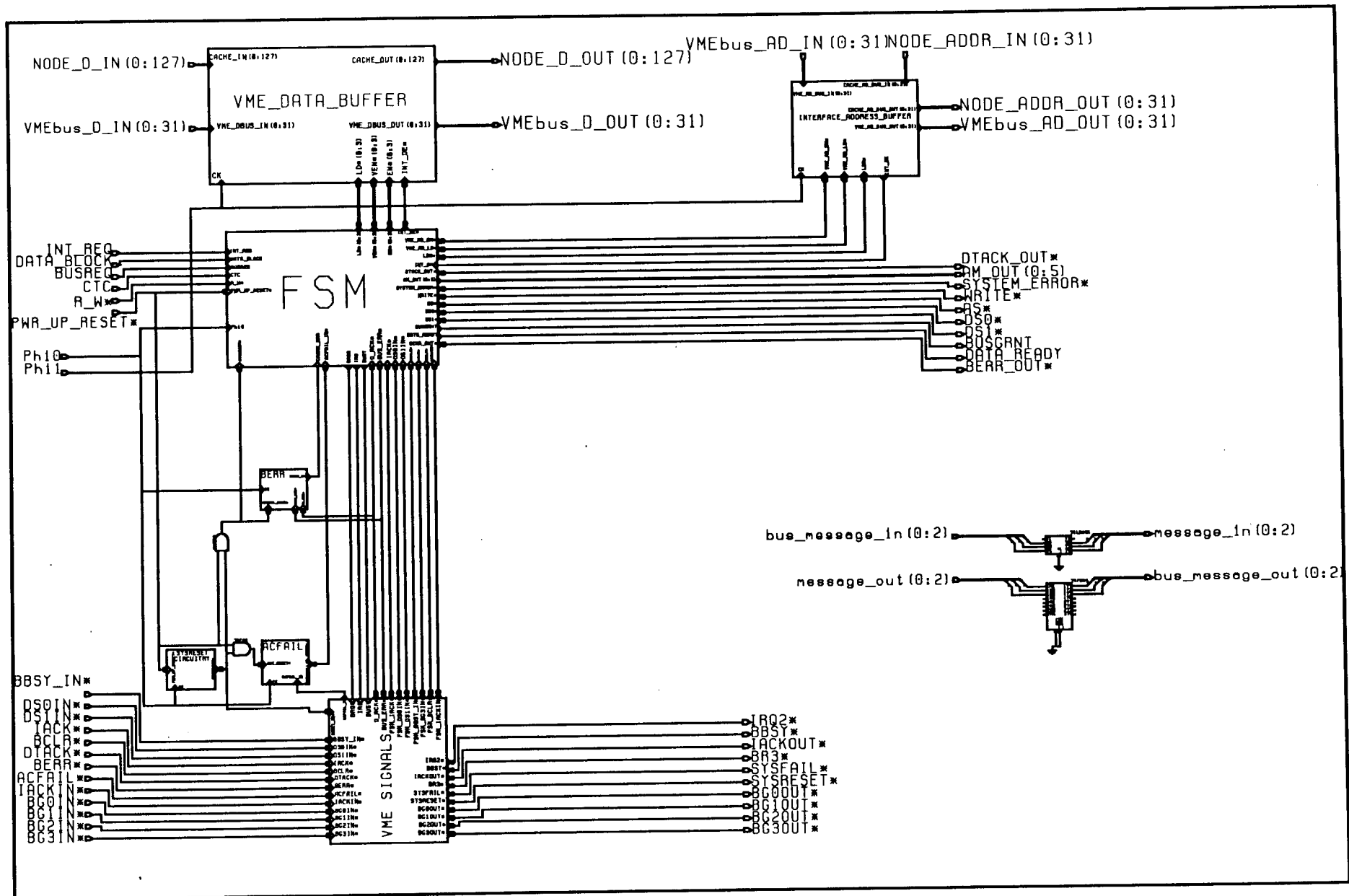
ENDSELECT;

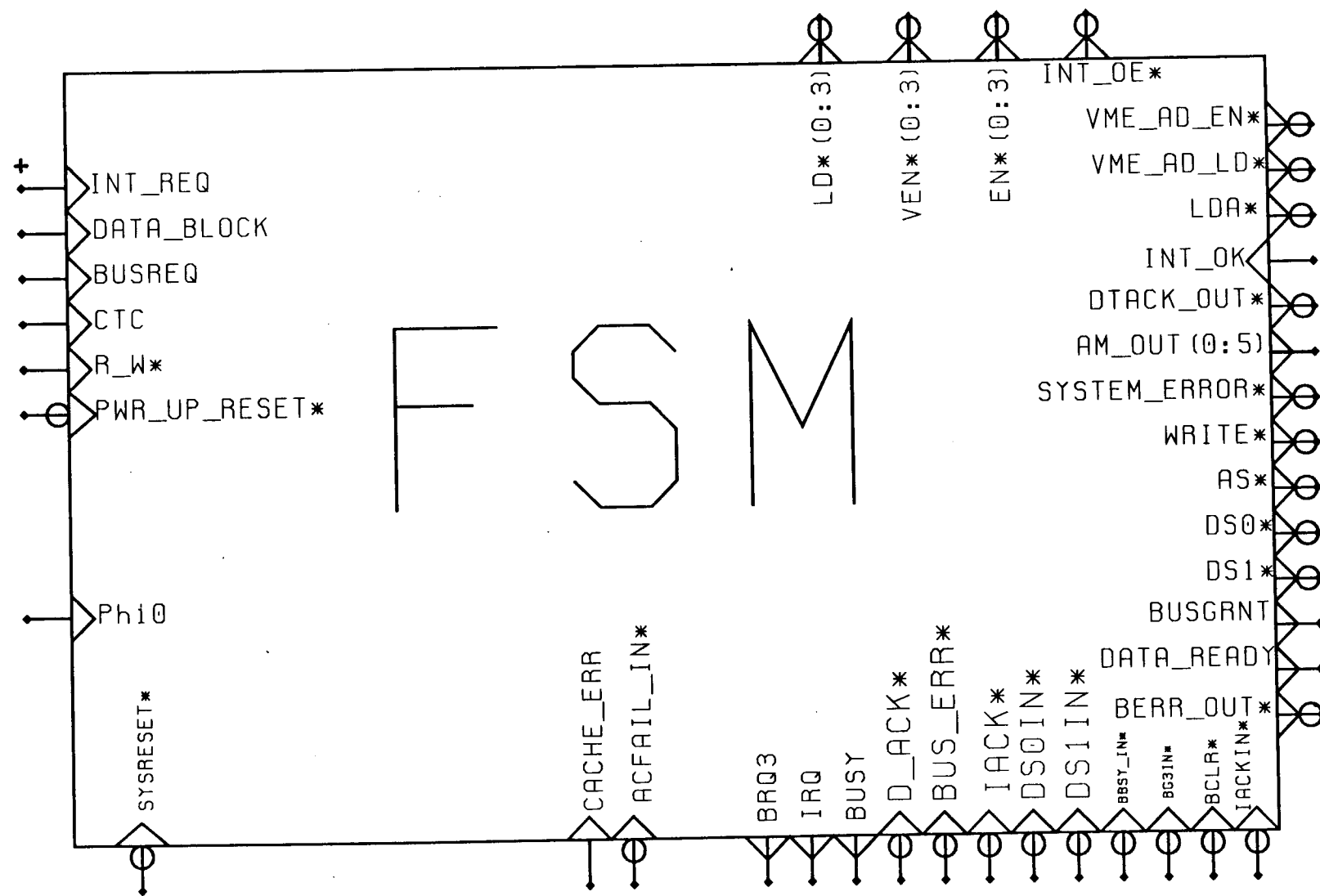
ENDROUTINE;

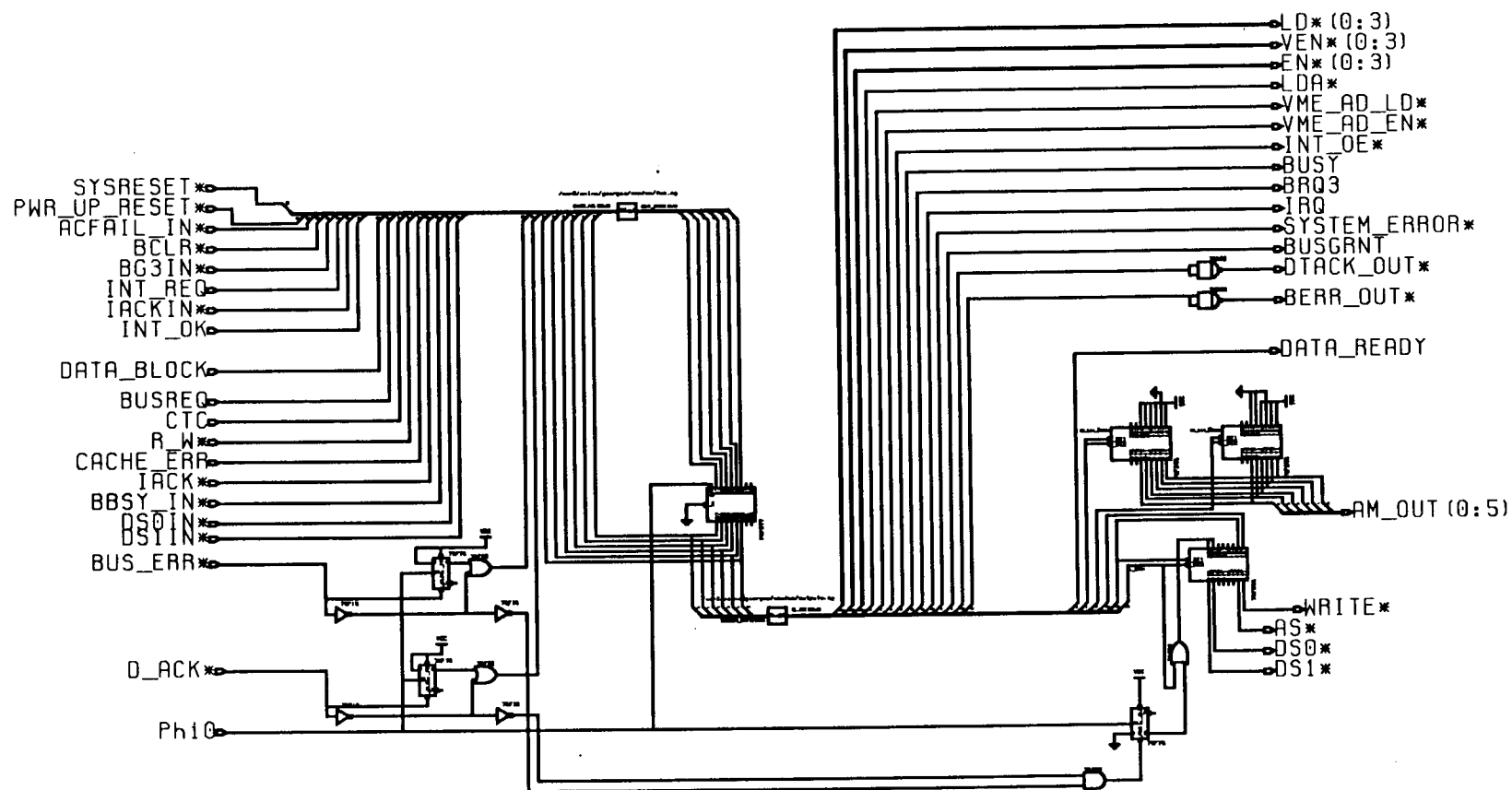
ENDMODEL;

Appendix D: Schematics



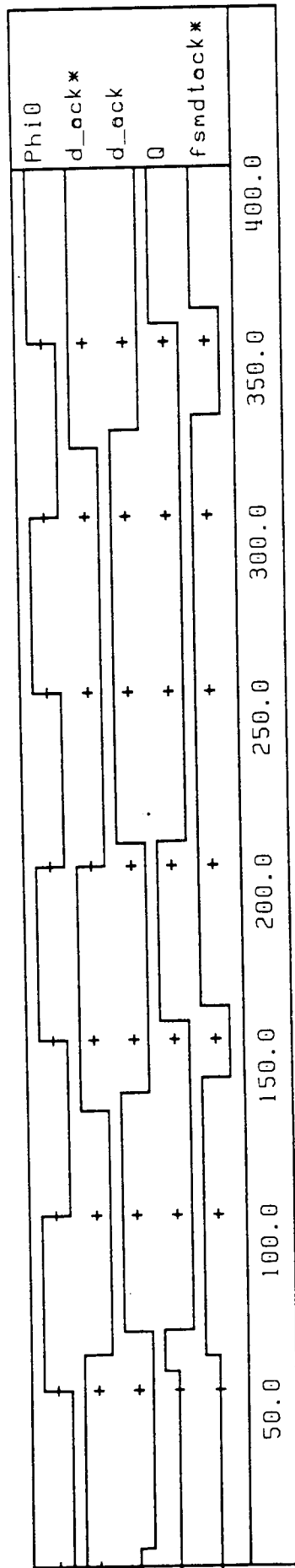


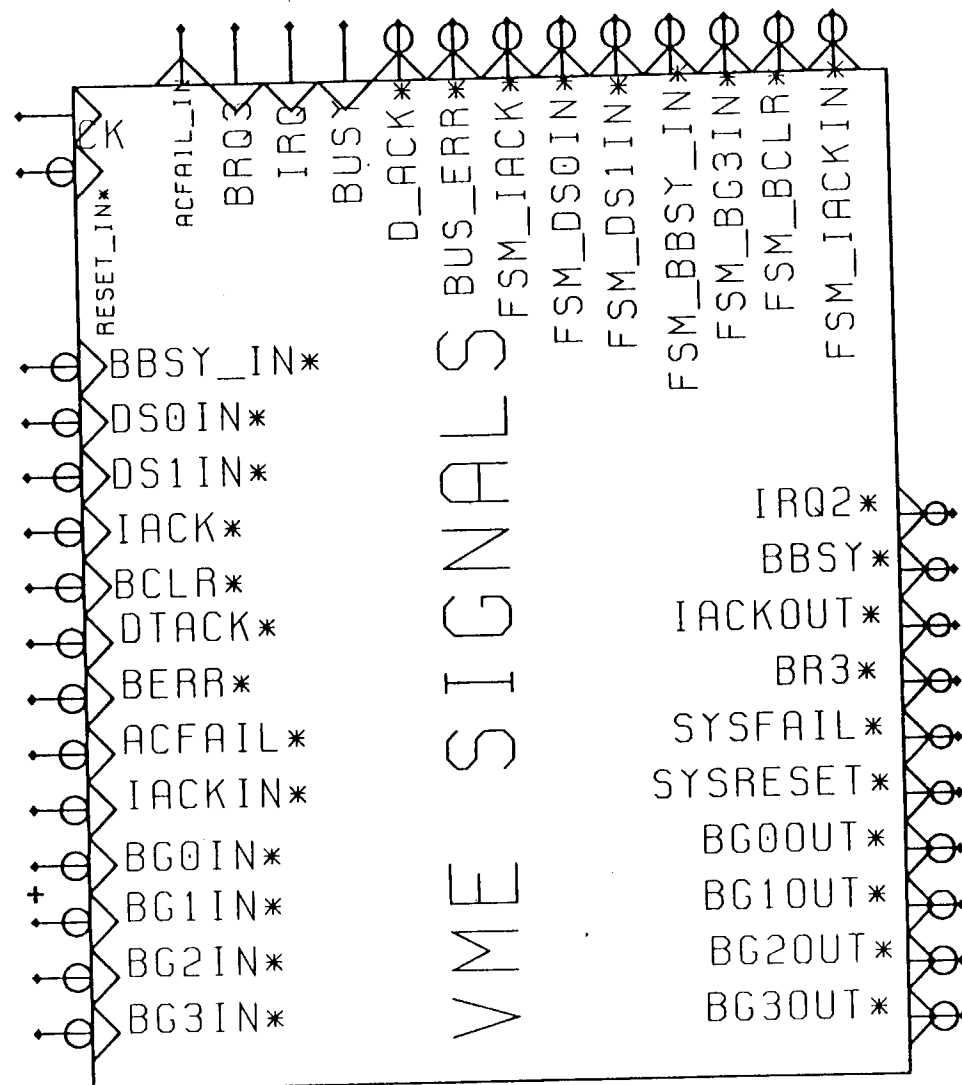




VME INTERFACE FINITE STATE MACHINE
 VERSION 3.1
 5/29/89

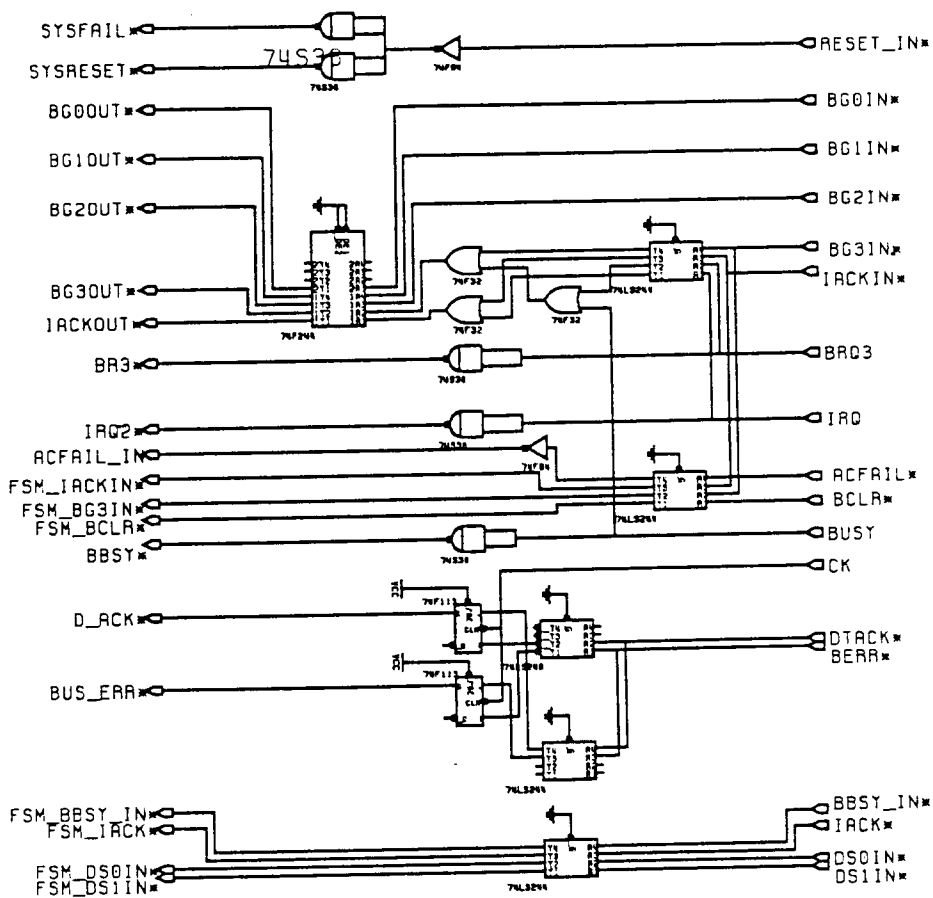
[illegible]

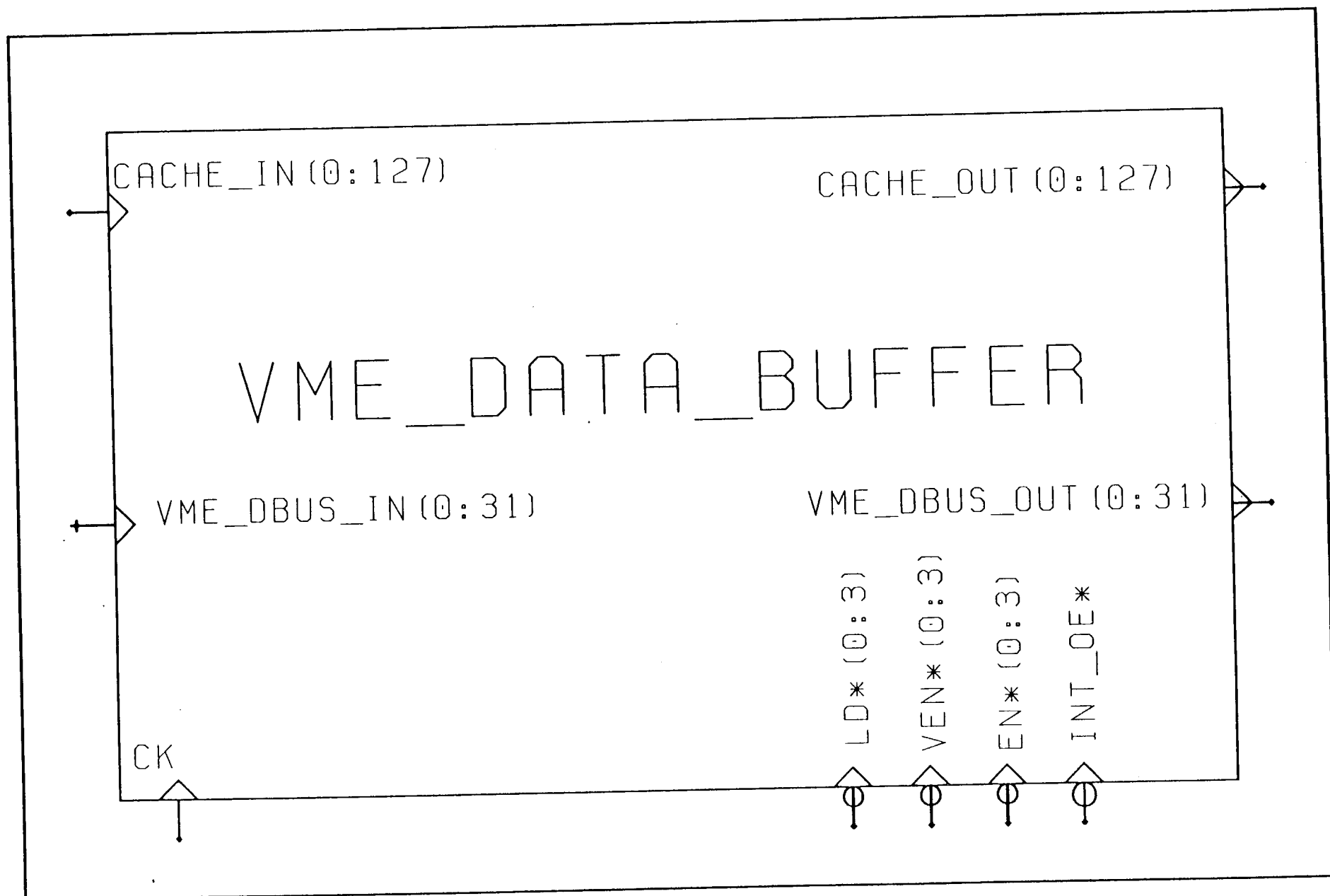


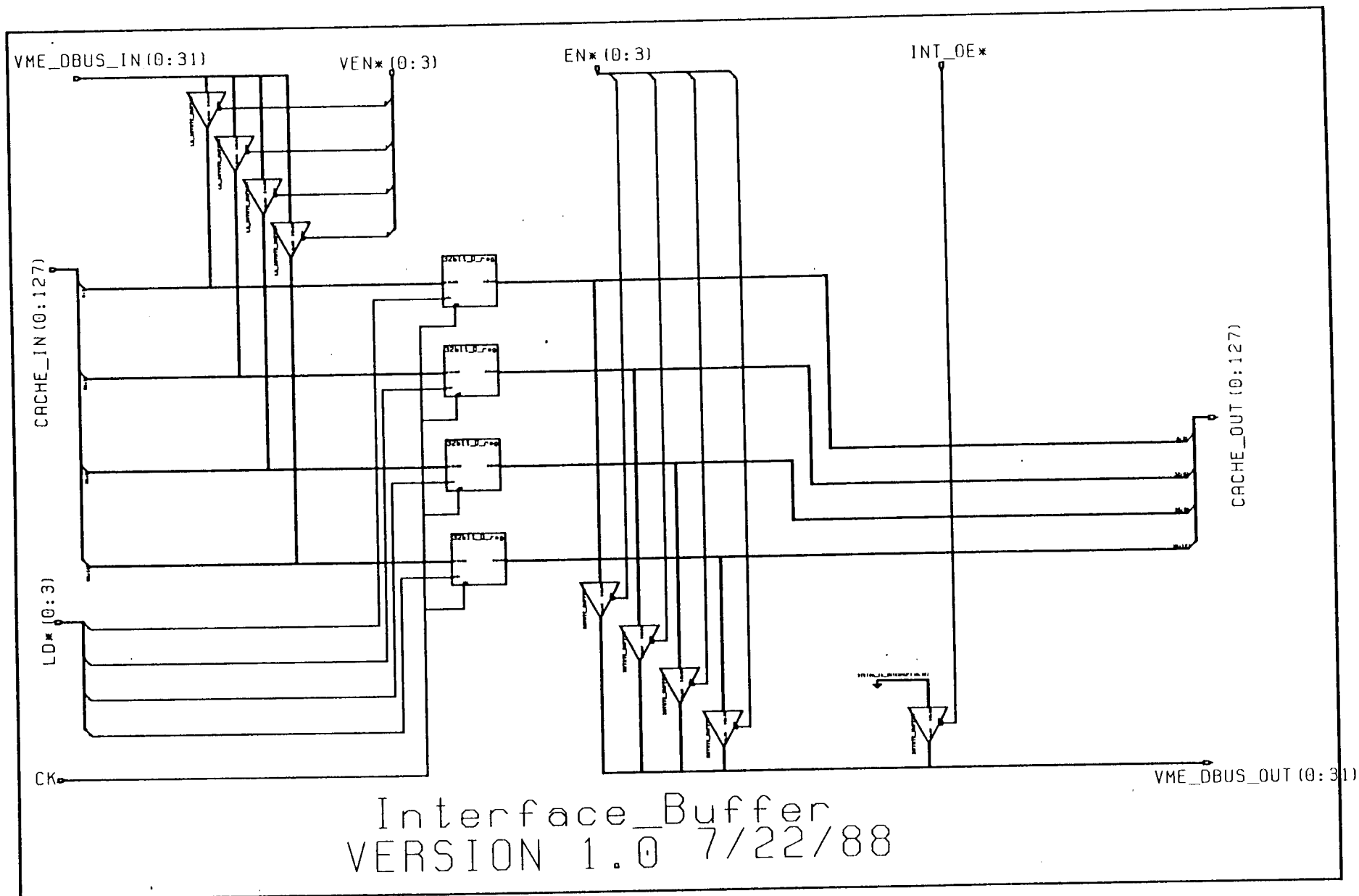


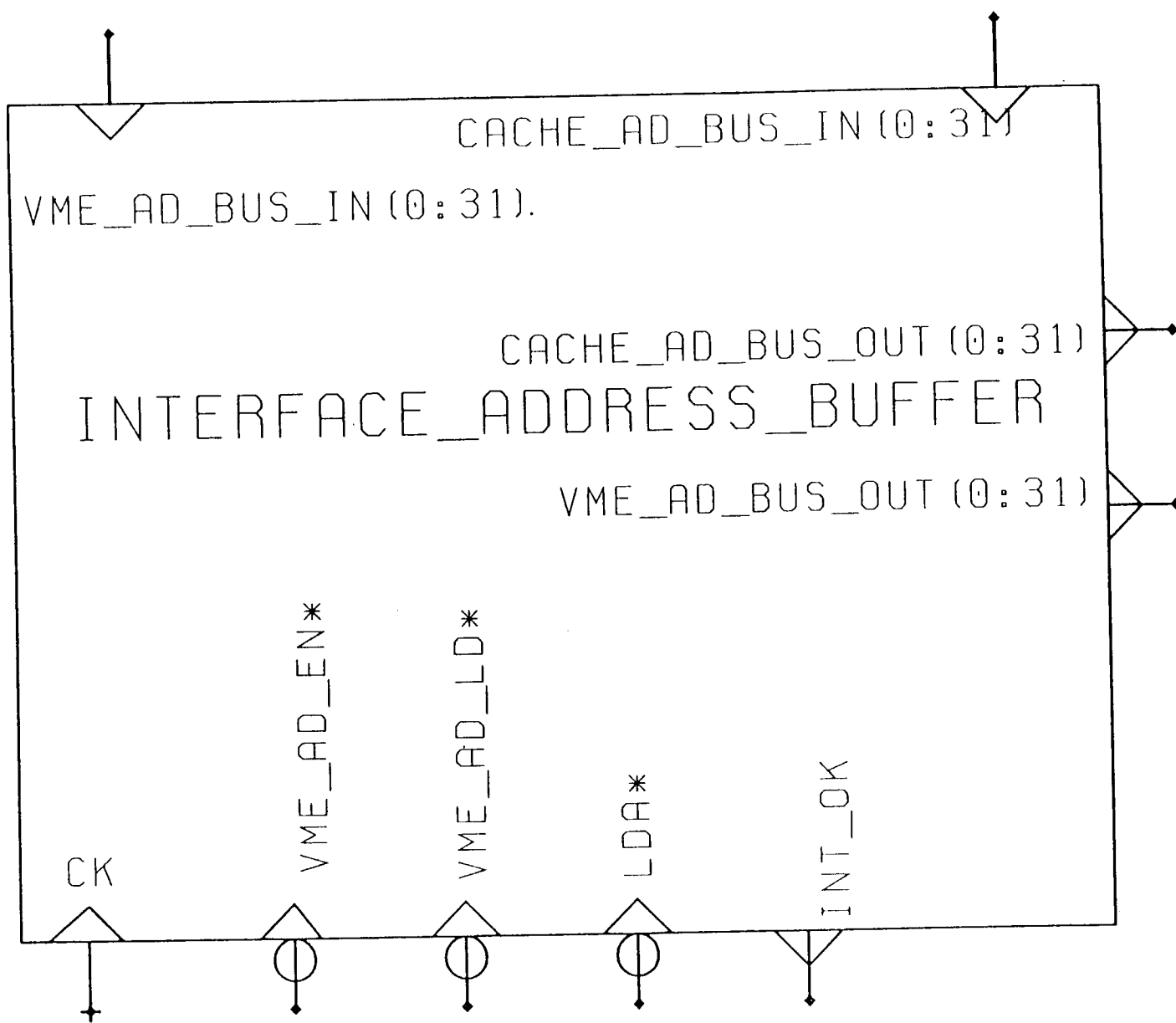
5/31/89

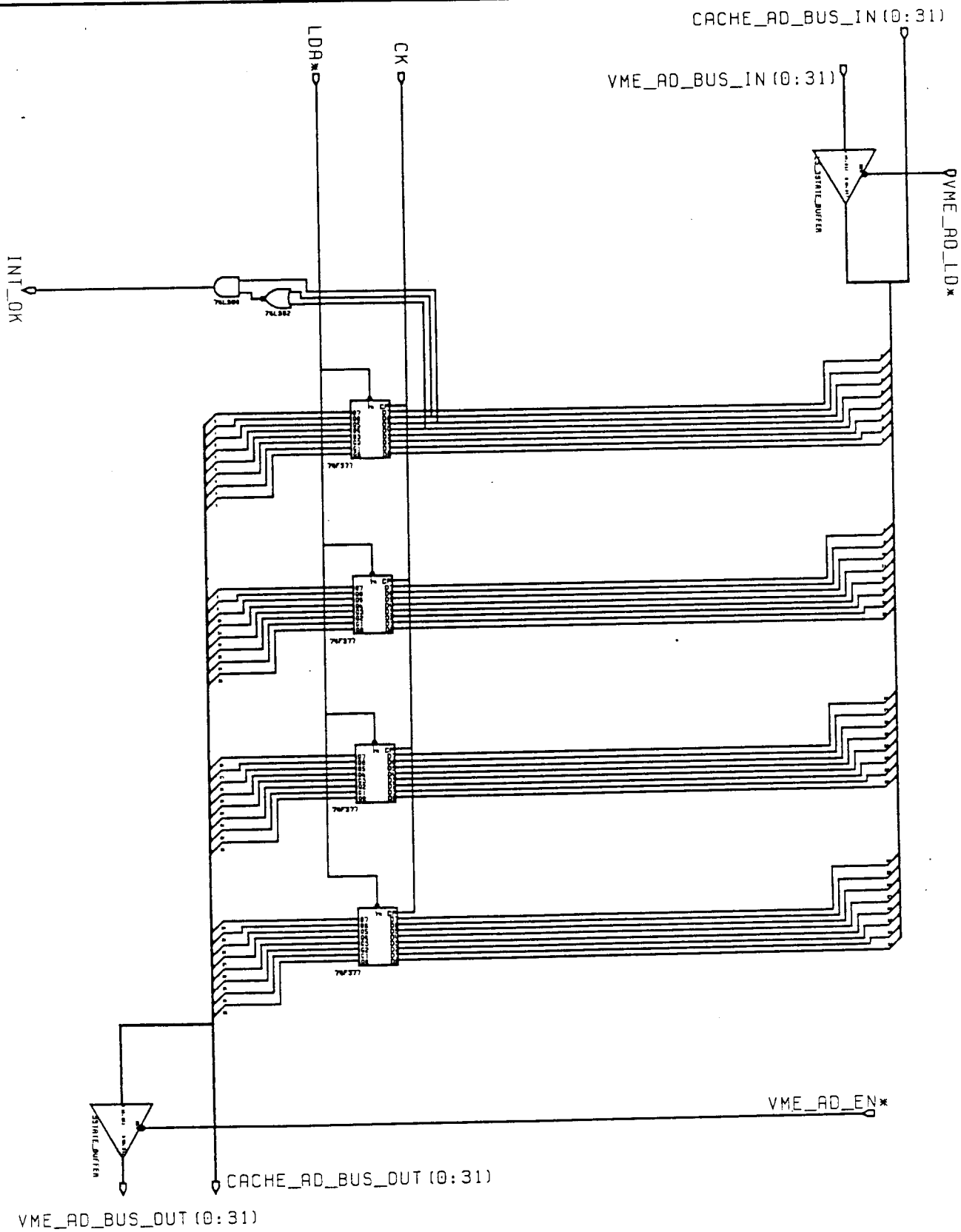
VME SIGNALS









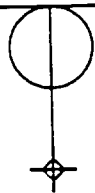
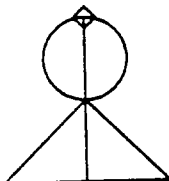


SYSRESET CIRCUITRY

PWR_UP_RESET*

CK

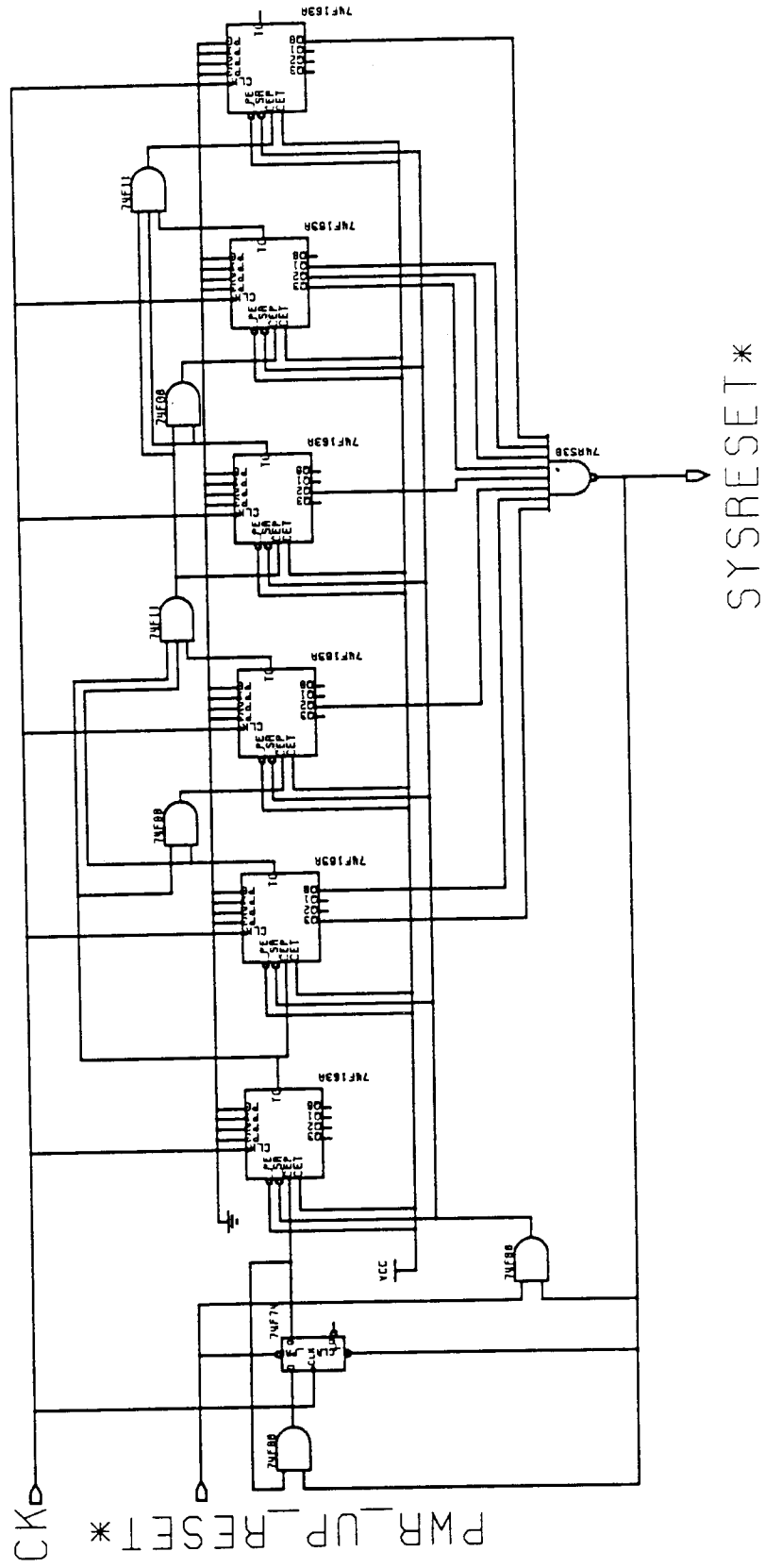
SYSRESET*

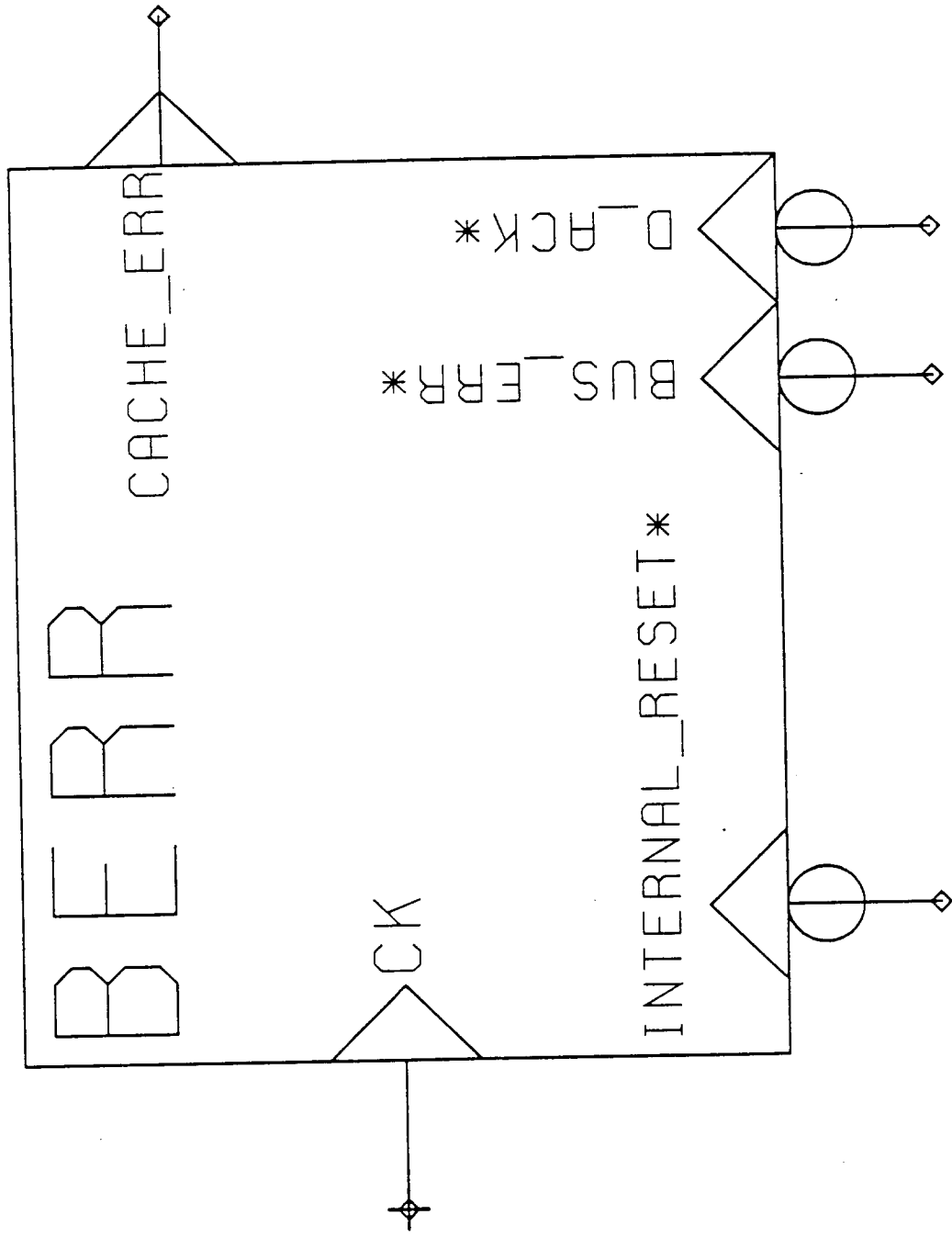


SYSRESET*_CIRCUITRY

VERSION 1.1

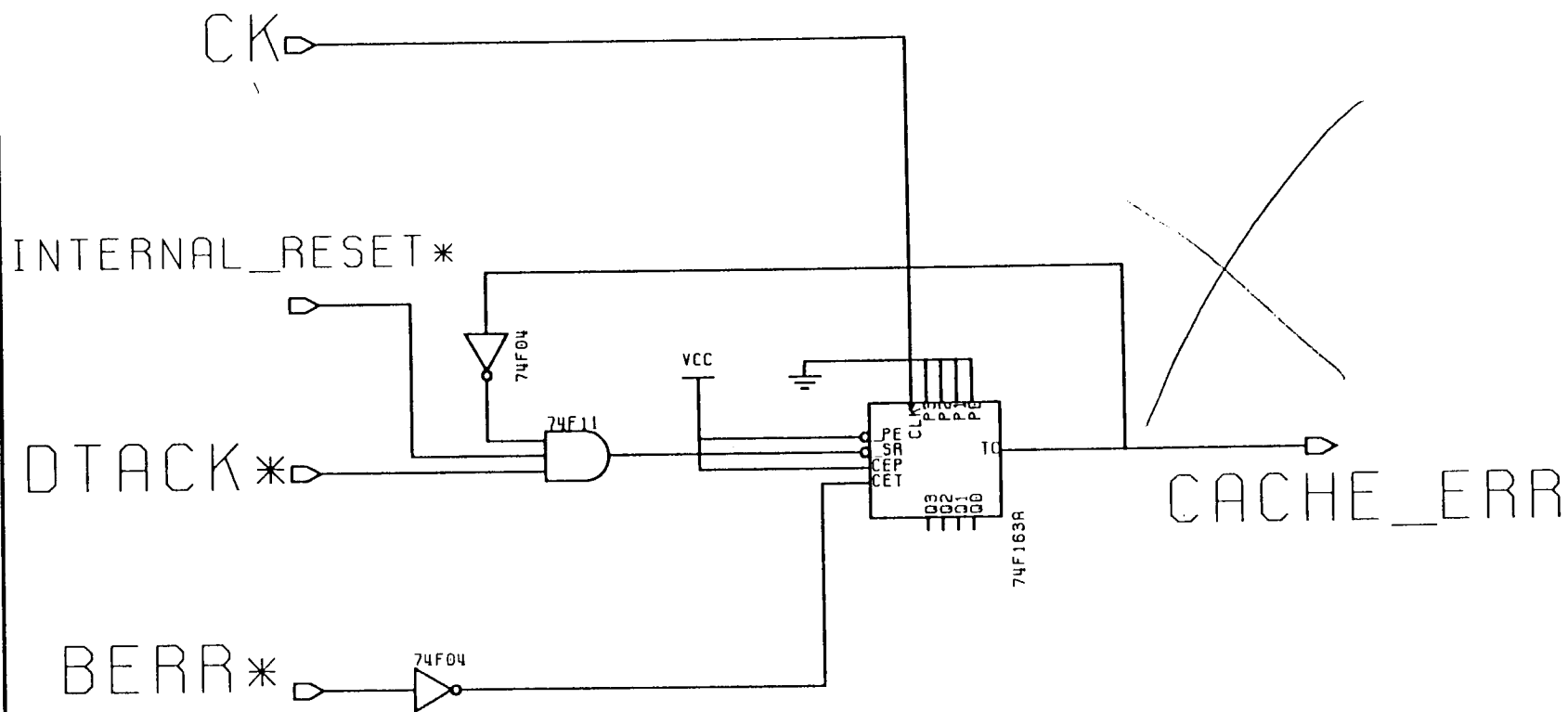
8/01/88

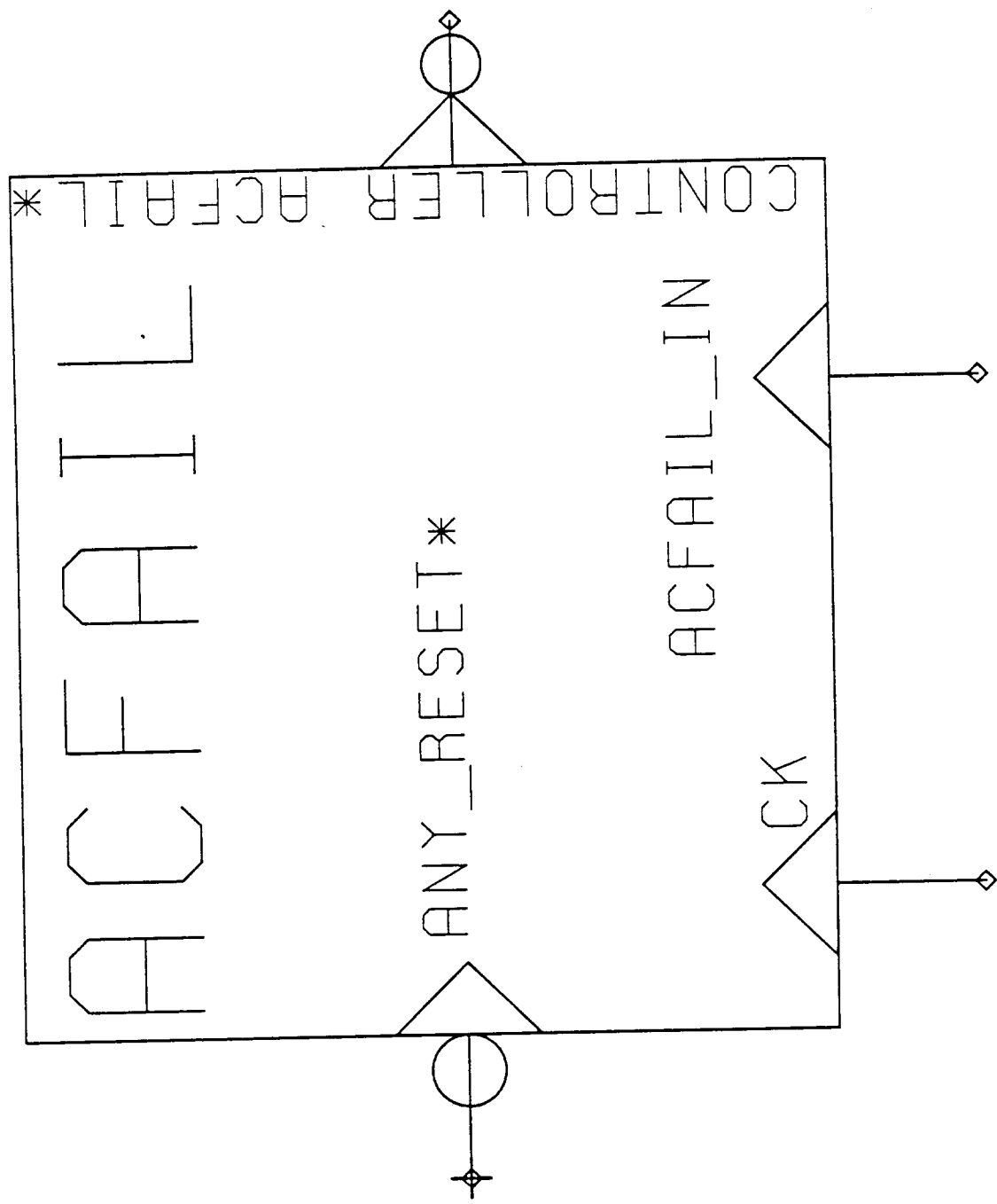




BERR*_DETECTION_CIRCUITRY

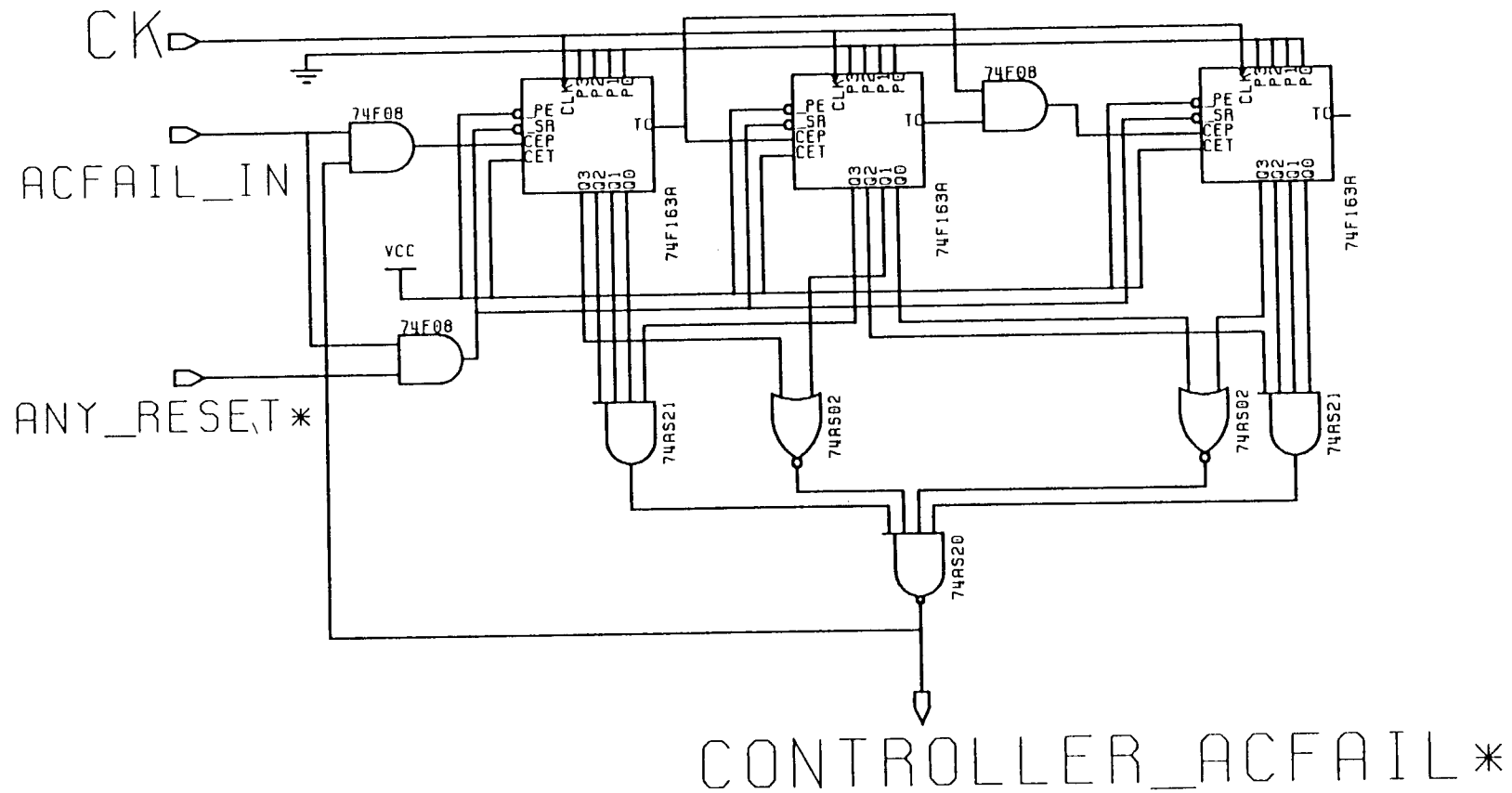
8/01/88

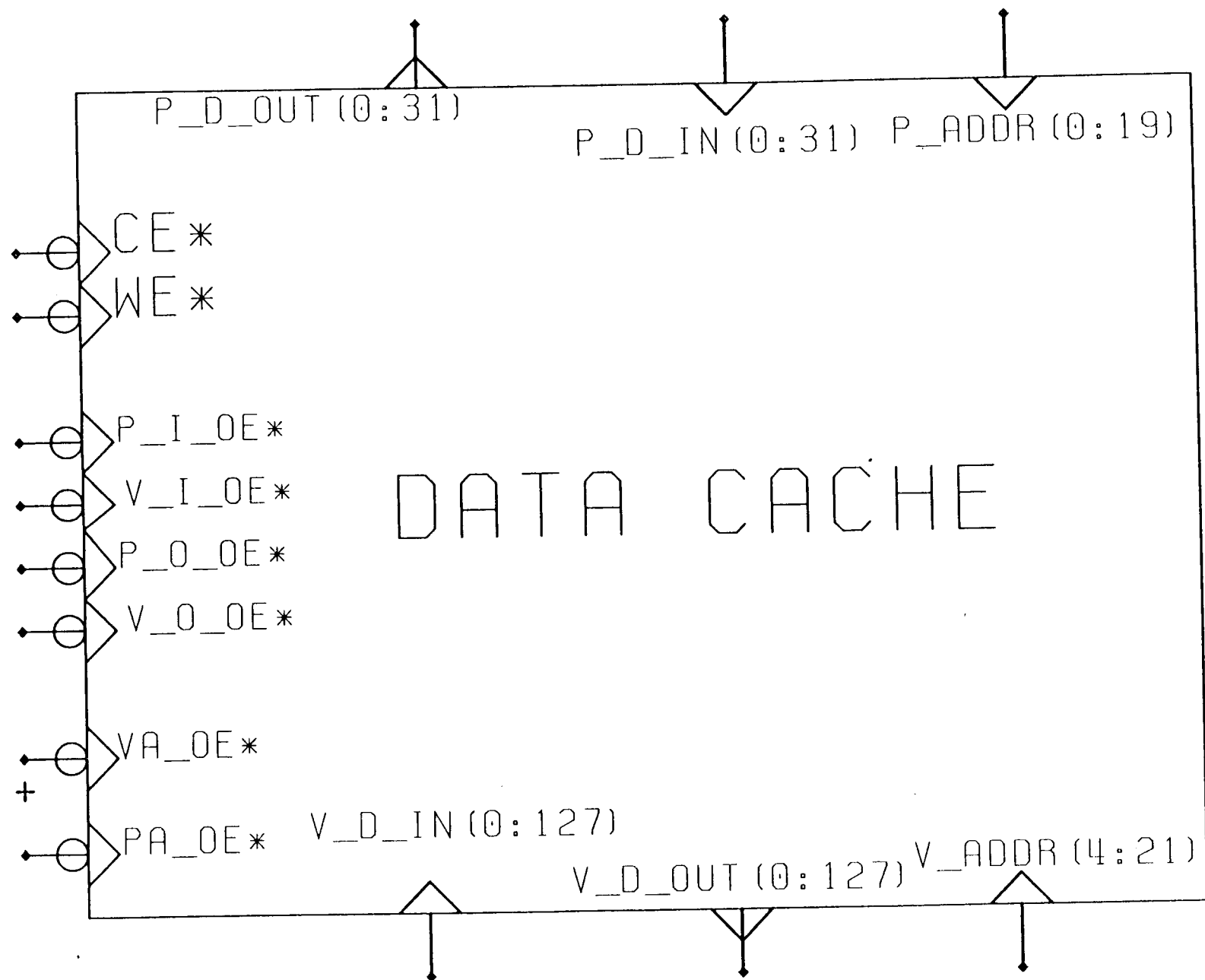




ACFAIL*_DETECTION_CIRCUITRY

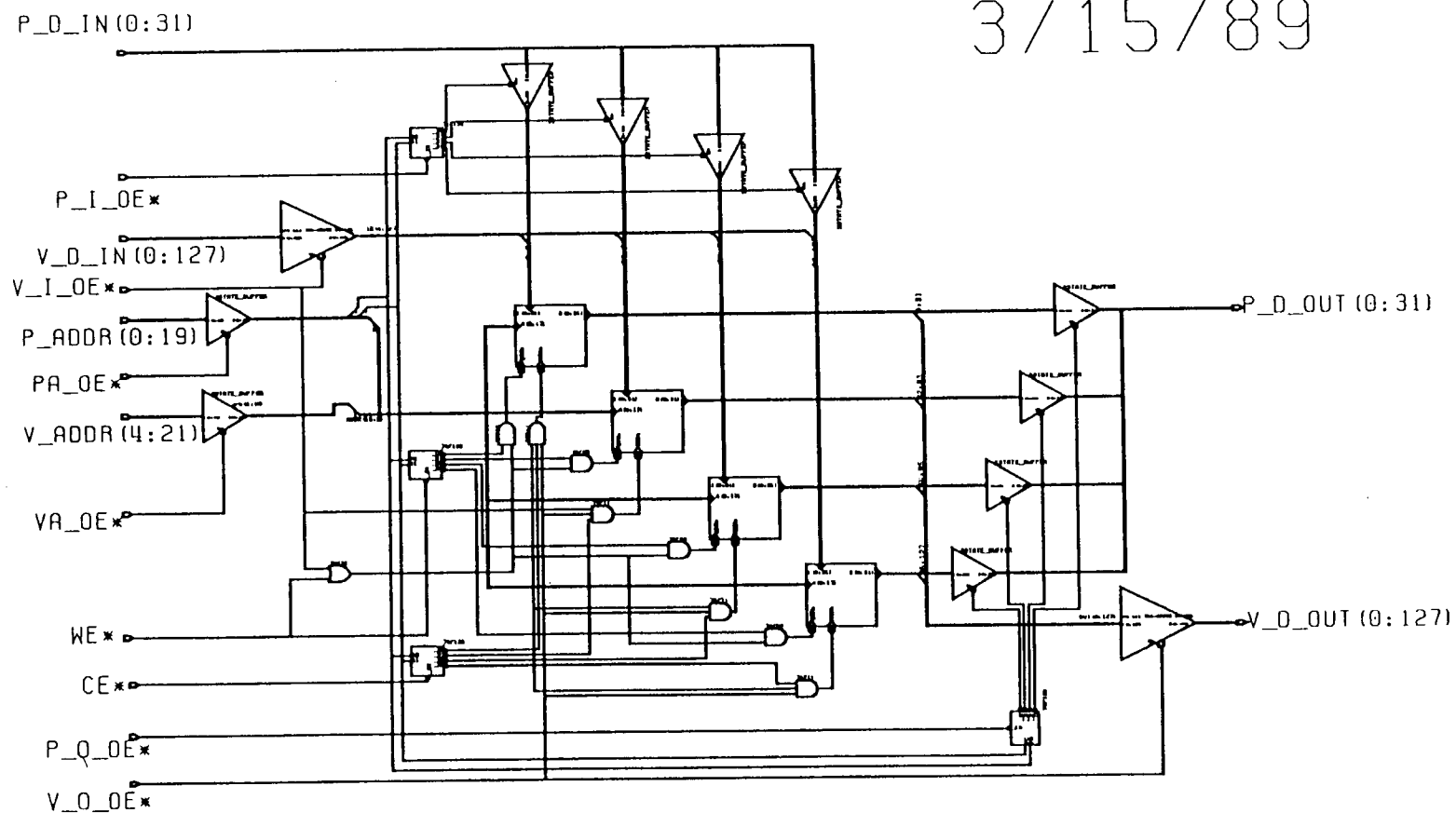
5/31/89

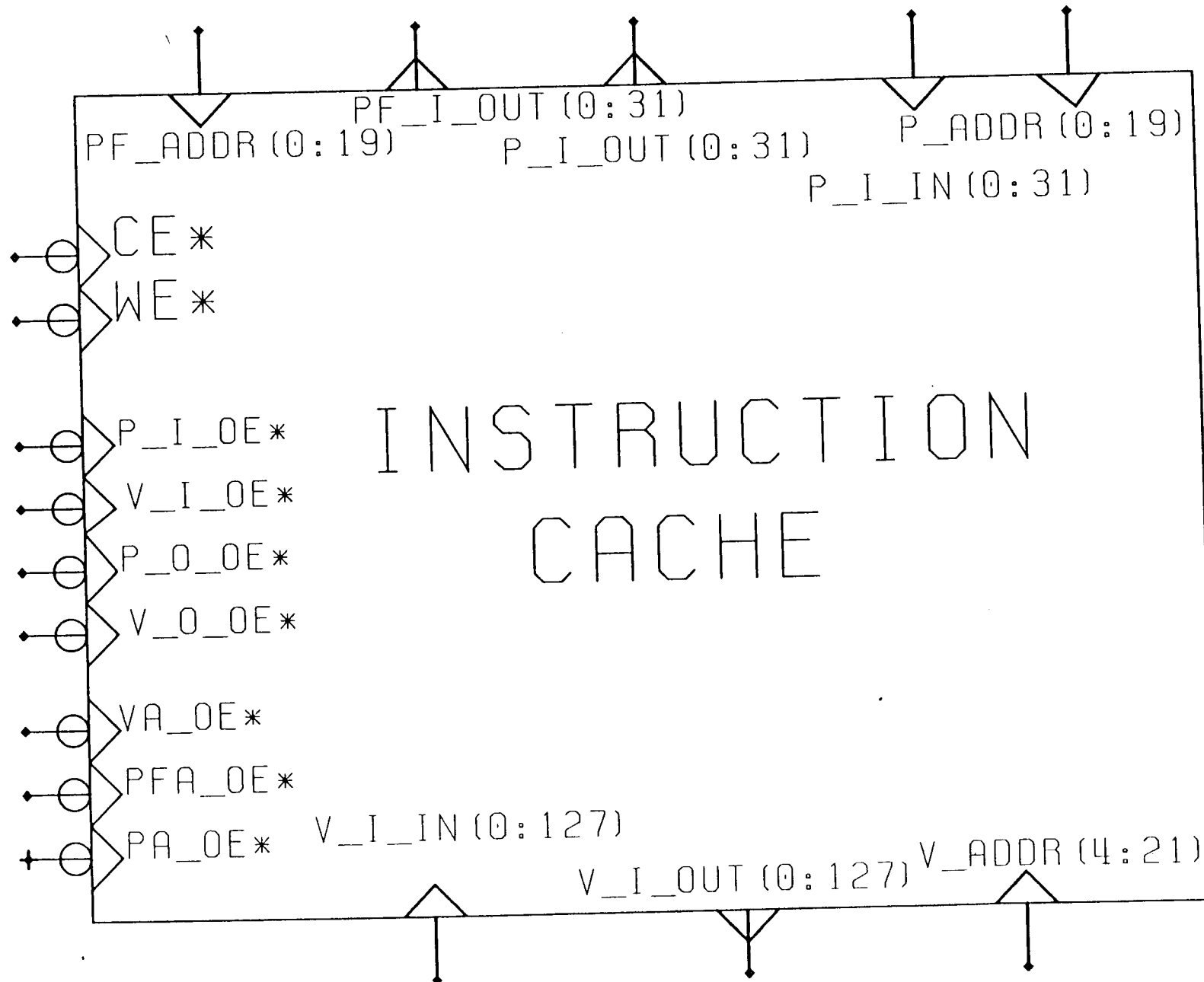




4MB CACHE

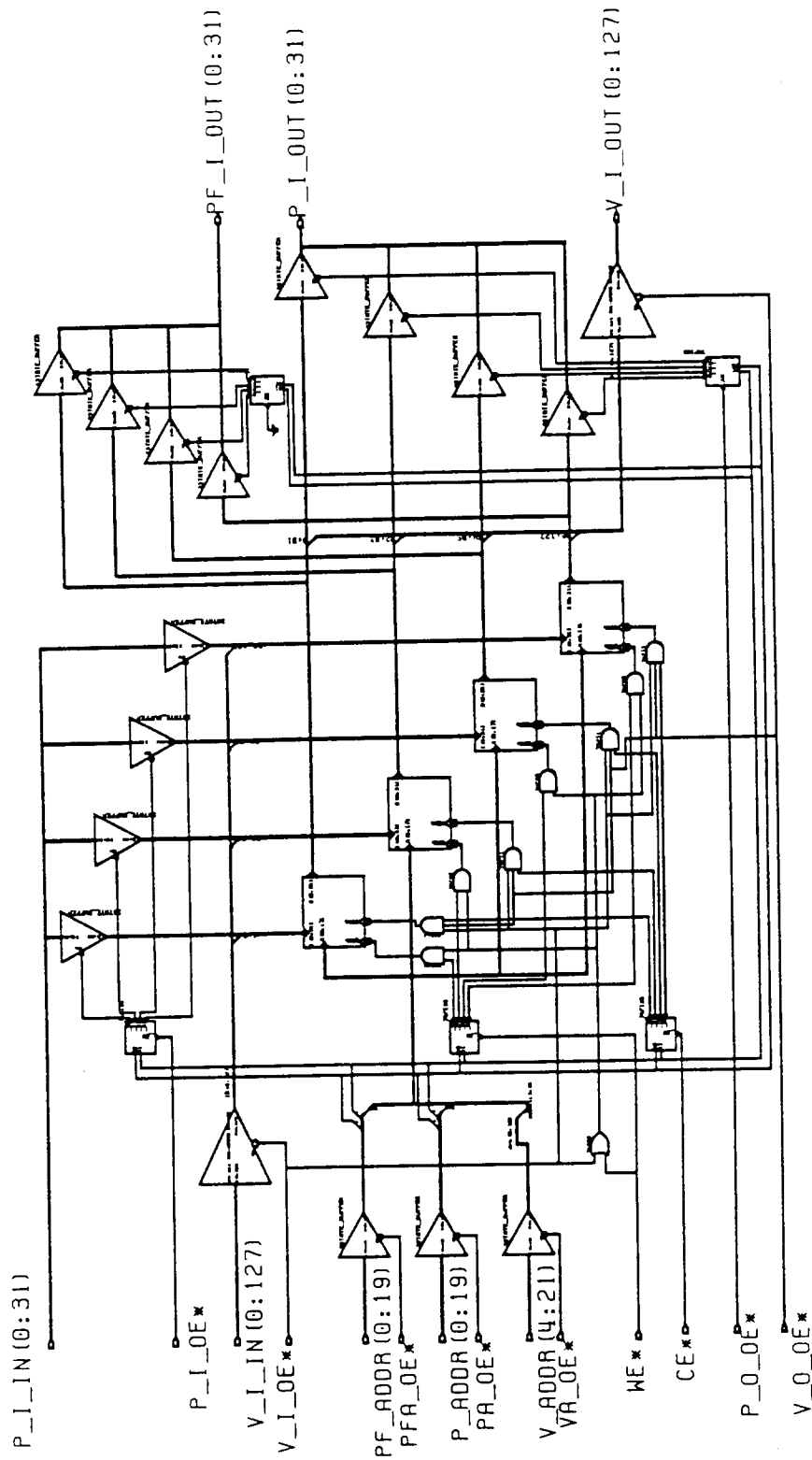
3/15/89

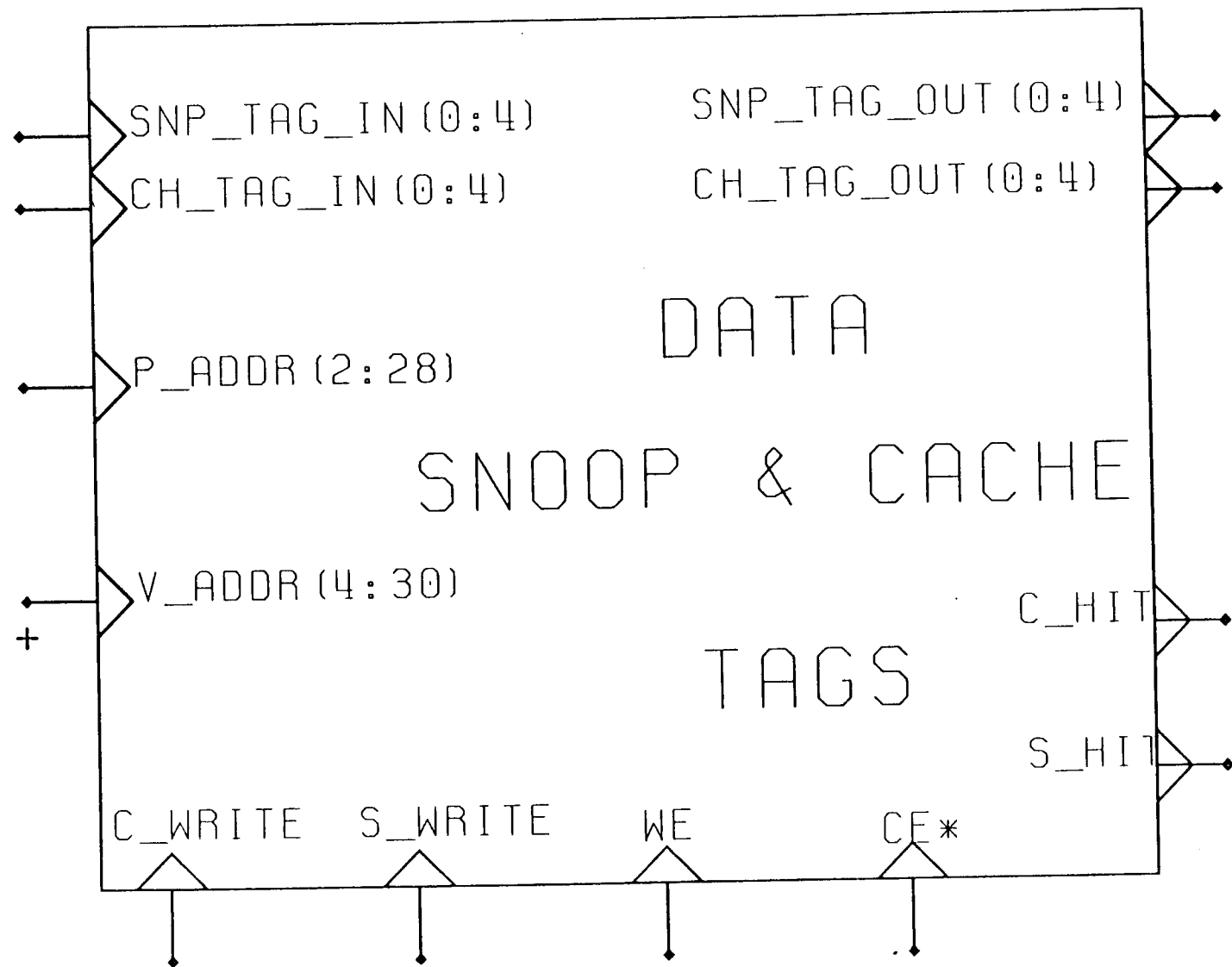




4MB INSTRUCTION CACHE

5/19/89

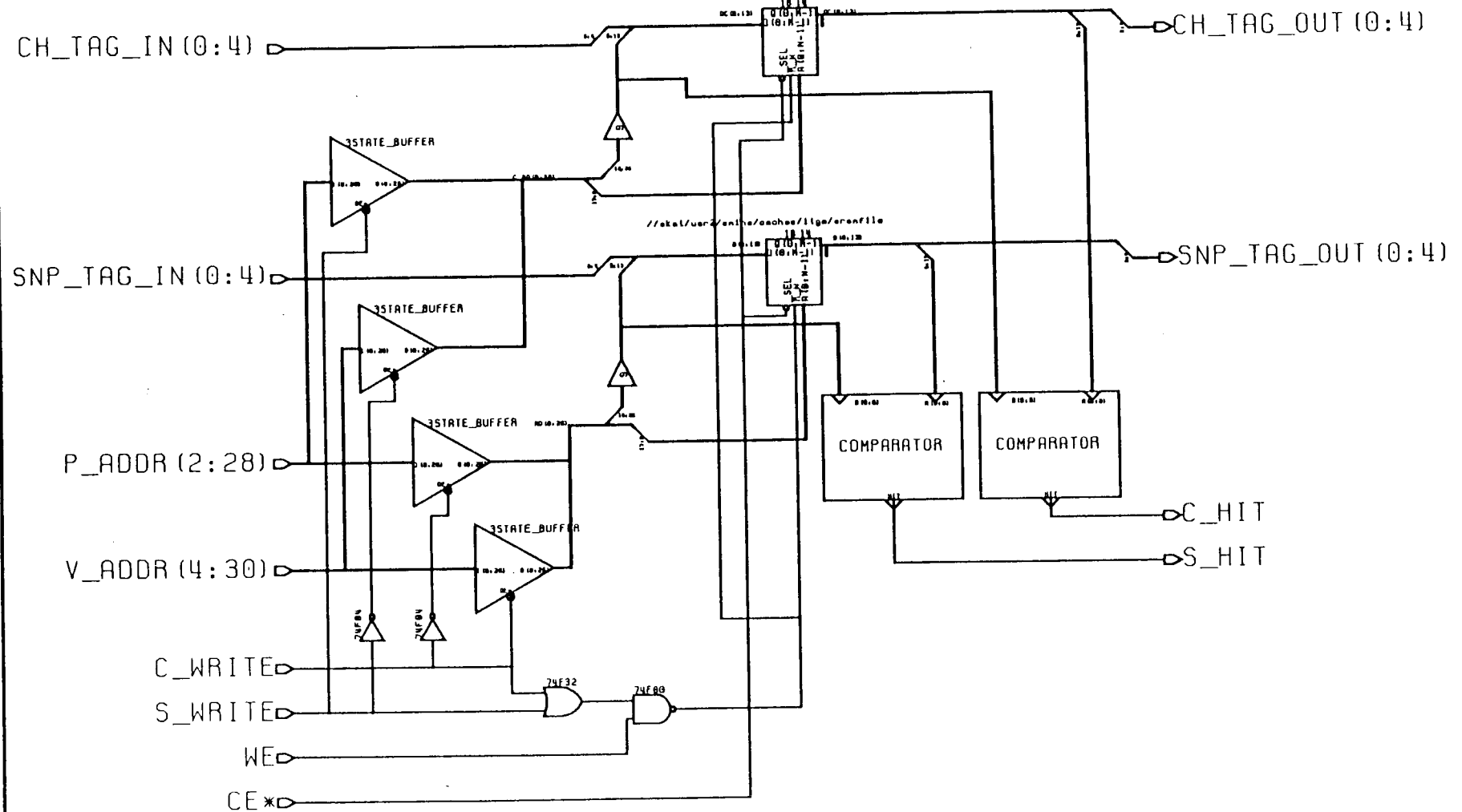


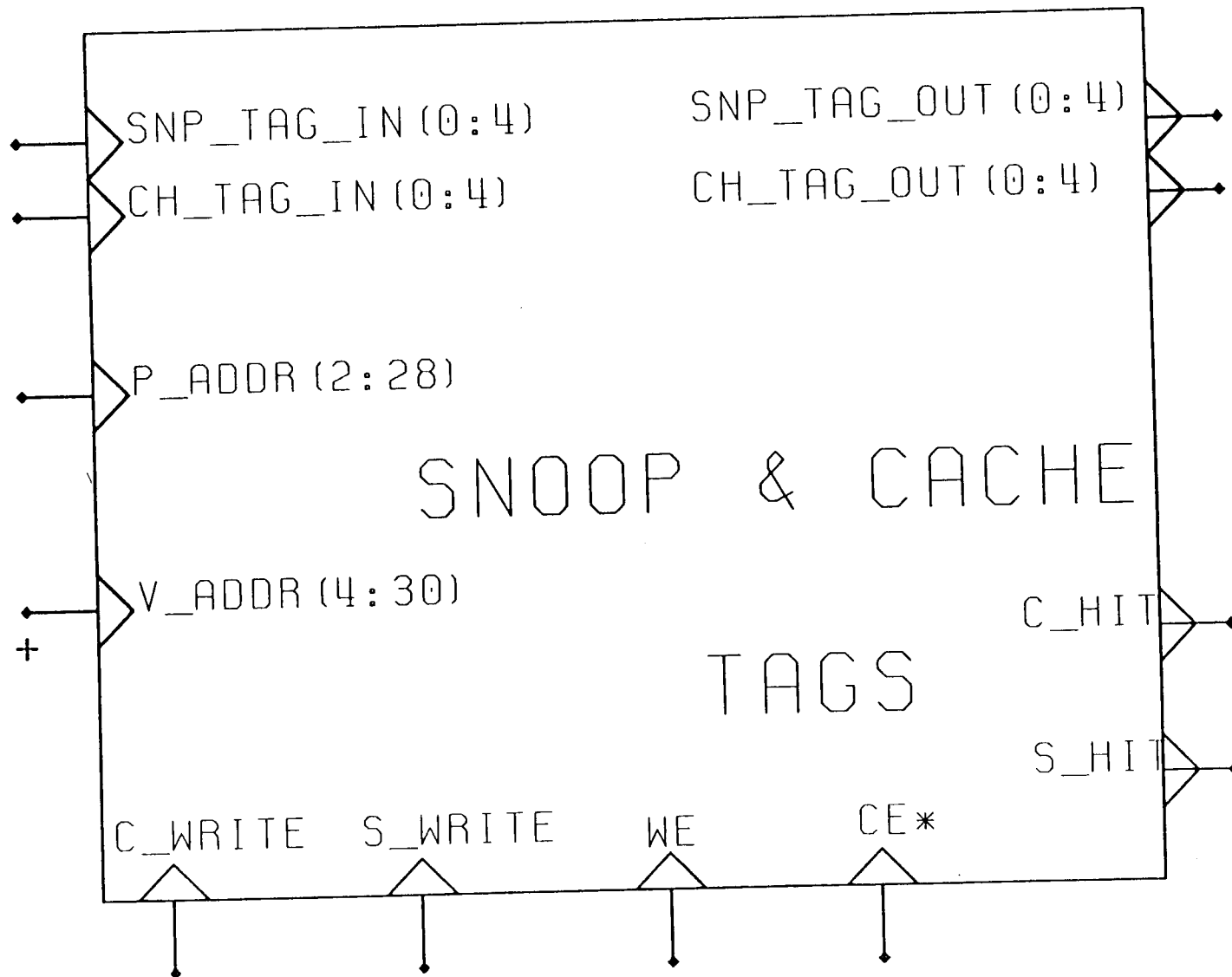


SNOOP AND CACHE TAGS

3/20/89

//akal/usr2/online/caches/lige/cranfile

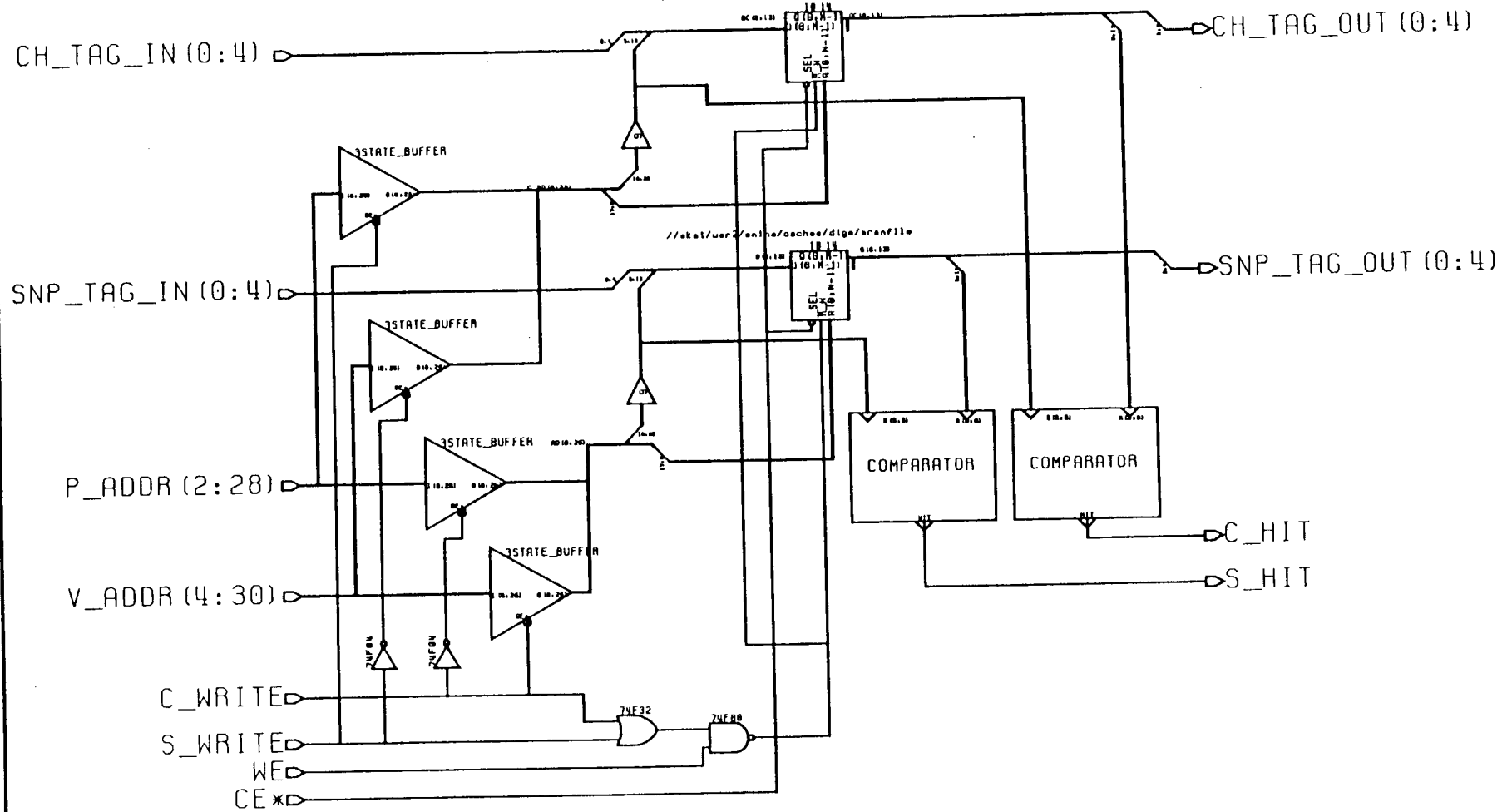




SNOOP AND CACHE TAGS

3/20/89

//skel/user2/enine/caches/dlga/oranFile



P_VIRT_ADDR (0:28)

ADDRESS
TRANSLATION
UNIT

VME_ADDR (0:31)

DRTY_IN

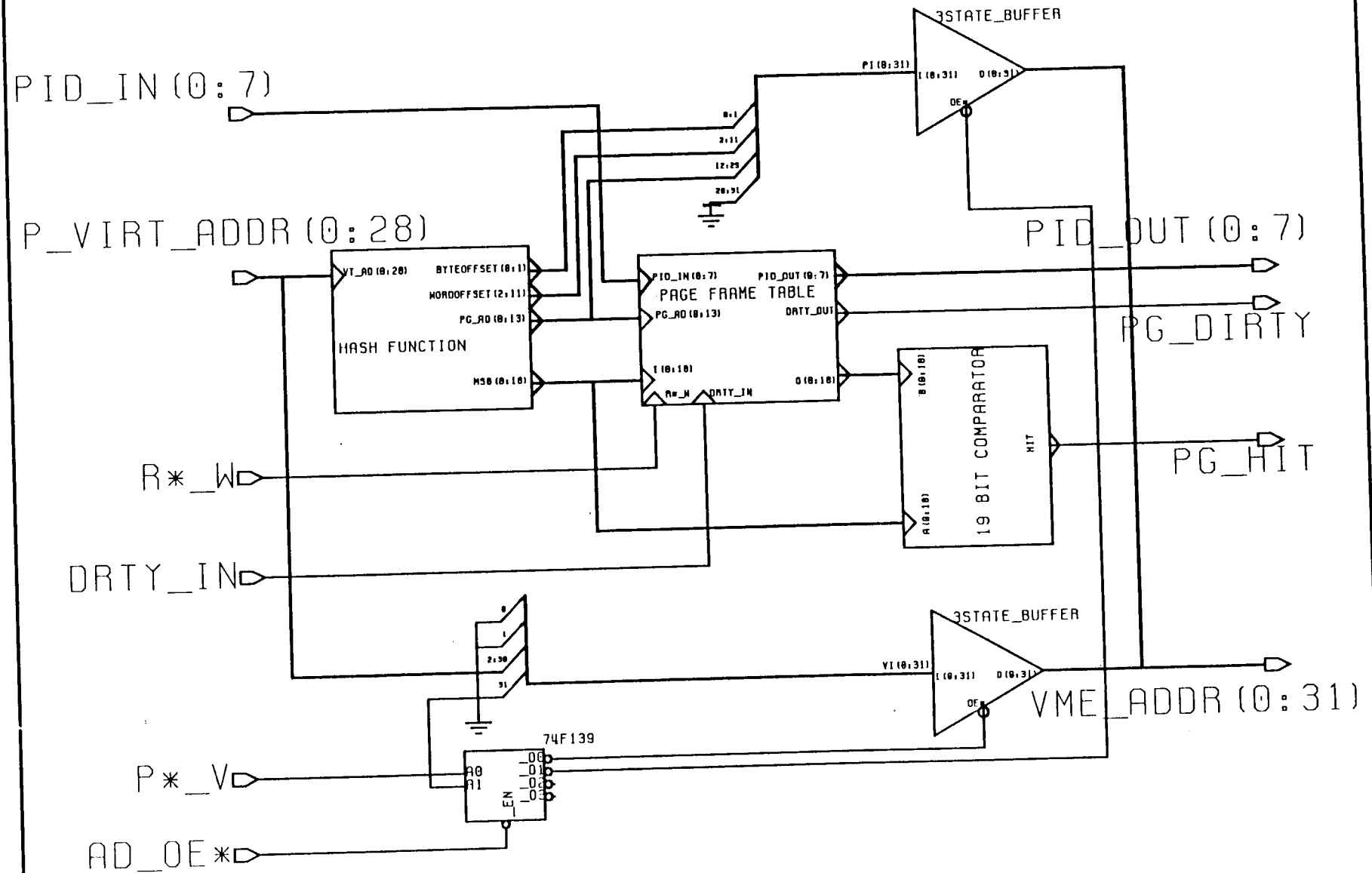
R*_W

P*_V

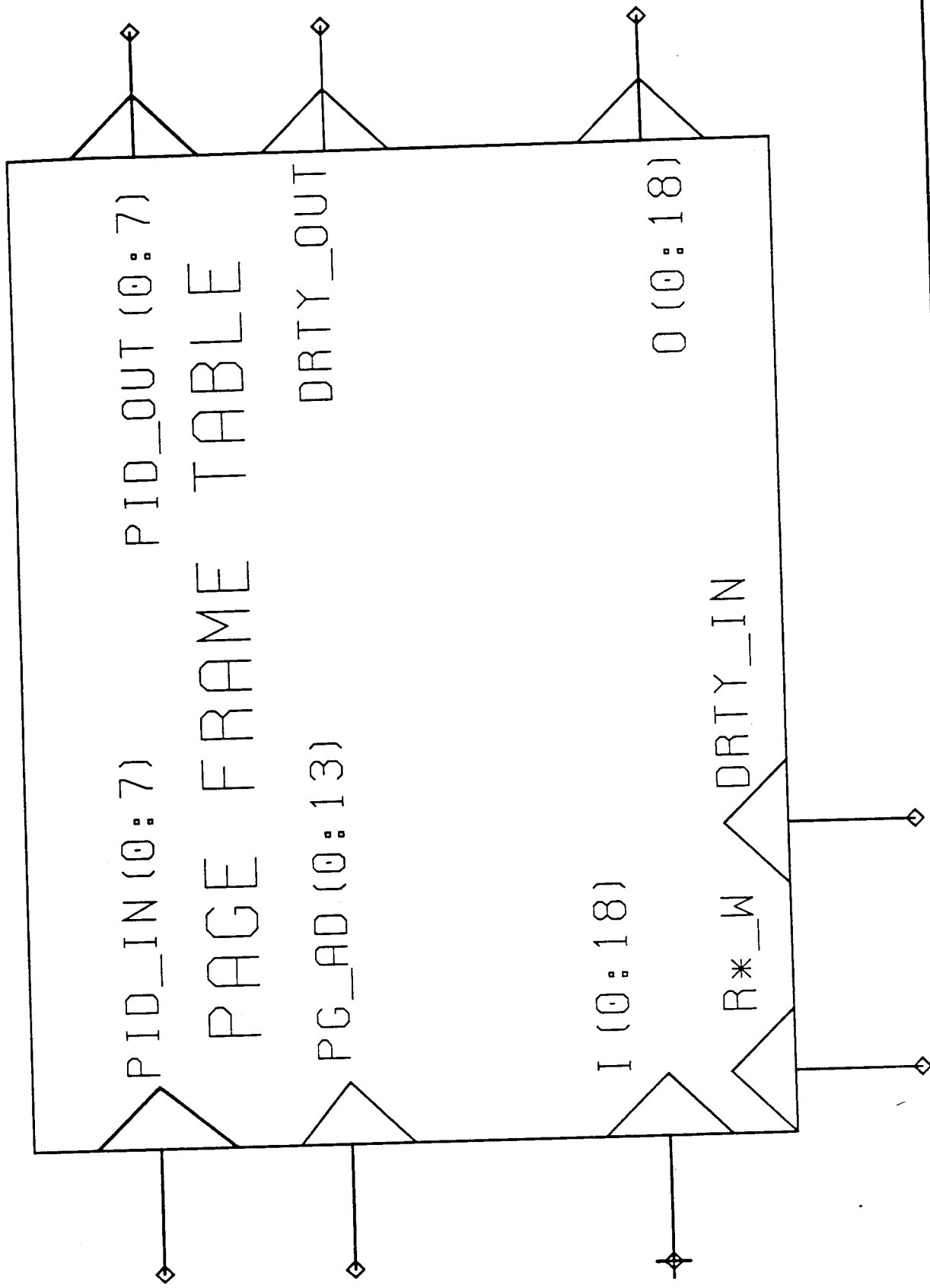
AD_OE*

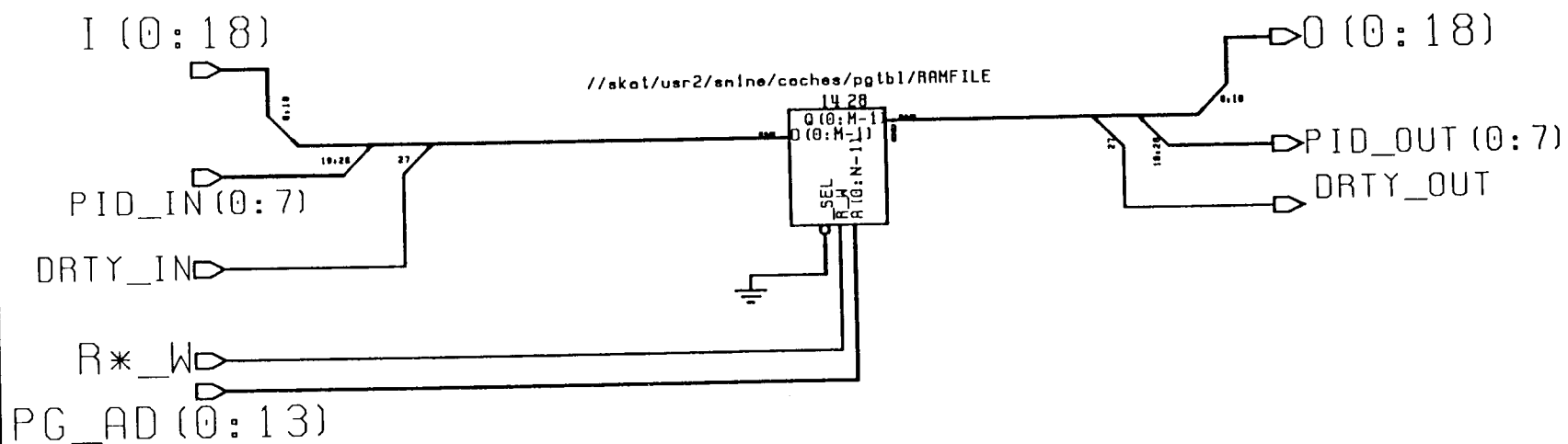
PG_DIRTY

PG_HIT



ADDRESS TRANSLATION





PAGE FRAME TABLE
 VERSION 2.0
 3/01/89

VT_AD (0:28)

BYTEOFFSET (0:1)

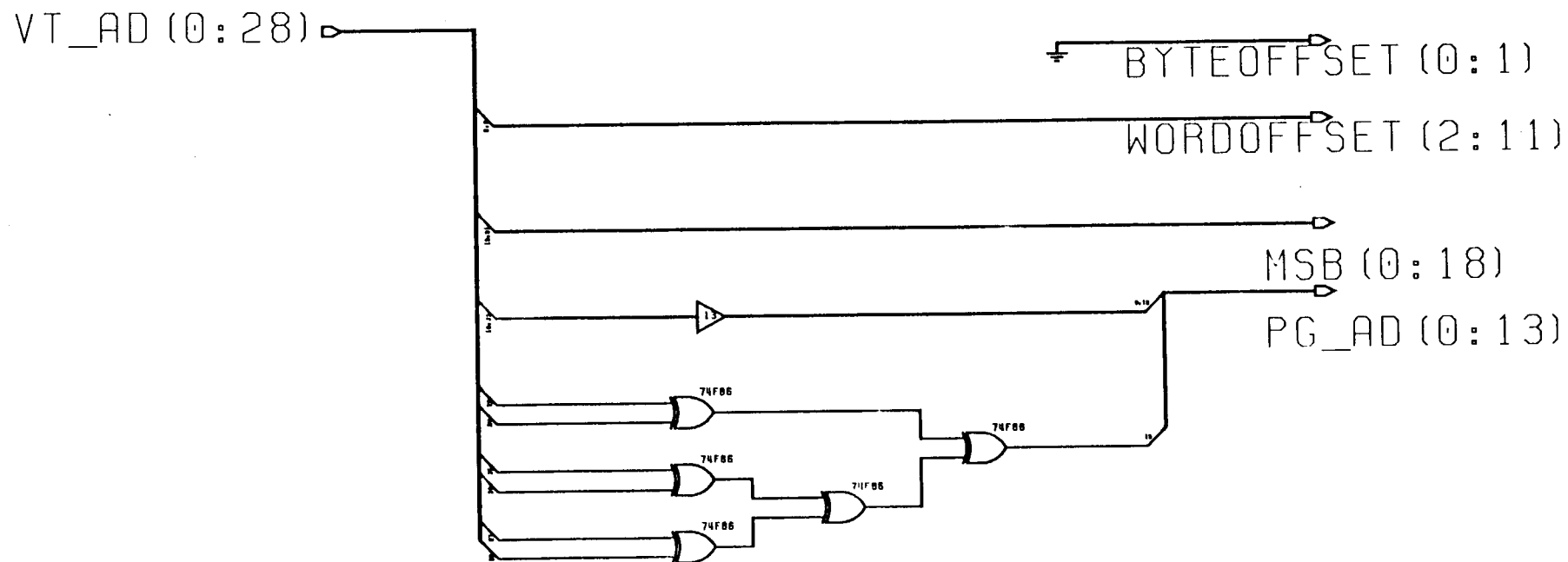
WORDOFFSET (2:11)

PG_AD (0:13)

HASH FUNCTION

MSB (0:18)

+



HASH FUNCTION
VERSION 3.0
3/15/89

Appendix E: Simulation Results

ADDRESS TRANSLATION

at_sim

TIME	p_virt_addr	base_addr	inst	ad_opcode	p_v	pg_hit
1760.0	00000000	0779F7C0	1	0	0	1
1760.0	00000000	0779F7C0	1	0	0	1
1800.0	02334455	0779F7C0	1	0	0	1
1820.0	02334455	046688C0	1	0	0	1
1840.0	02334455	046688C0	1	0	0	1
1860.0	02334455	046688C0	1	0	0	1
1880.0	02334455	046688C0	1	0	0	1
1900.0	02334455	046688C0	1	0	0	1
1920.0	02334455	046688C0	1	0	0	1
1940.0	02334455	046688C0	1	0	0	1
1960.0	02334455	046688C0	1	0	0	1
1980.0	02334455	046688C0	1	0	0	1
2000.0	02334455	046688C0	1	0	0	1
2020.0	02334455	046688C0	1	0	0	1
2040.0	02334455	046688C0	1	0	0	1
2060.0	02334455	046688C0	1	0	0	1
2080.0	02334455	046688C0	1	0	0	1
2100.0	02334455	046688C0	1	0	0	1
2120.0	02334455	046688C0	1	0	0	1
2140.0	02334455	046688C0	1	0	0	1
2160.0	02334455	046688C0	1	0	0	1
2180.0	02334455	046688C0	1	0	0	1
2200.0	02334455	046688C0	1	0	0	1
2220.0	02334455	046688C0	1	0	0	1
2240.0	02334455	046688C0	1	0	0	1

89/06/28
12:26:52

0.0	00000000	XXXXXXXX	1	1	1	X
20.0	00000000	00000000	1	1	1	X
40.0	00000000	00000000	1	1	1	X
60.0	00000000	00000000	1	1	1	X
80.0	00000000	00000000	1	1	1	X
100.0	00000000	00000000	1	1	1	X
120.0	00000000	00000000	1	1	1	X
140.0	00000000	00000000	1	1	1	X
160.0	00000000	00000000	1	1	1	X
180.0	00000000	00000000	1	1	1	X
200.0	02334455	046688C0	1	0	0	1
220.0	02334455	046688C0	1	0	0	1
240.0	02334455	046688C0	1	0	0	1
260.0	02334455	046688C0	1	0	0	1
280.0	02334455	046688C0	1	0	0	1
300.0	02334455	046688C0	1	0	0	1
320.0	02334455	046688C0	1	0	0	1
340.0	02334455	046688C0	1	0	0	1
360.0	02334455	046688C0	1	0	0	1
380.0	02334455	046688C0	1	0	0	1
400.0	02334455	046688C0	1	0	0	1
420.0	02334455	046688C0	1	0	0	1
440.0	02334455	046688C0	1	0	0	1
460.0	02334455	046688C0	1	0	0	1
480.0	02334455	046688C0	1	0	0	1
500.0	02334455	046688C0	1	0	0	1
520.0	02334455	046688C0	1	0	0	1
540.0	02334455	046688C0	1	0	0	1
560.0	02334455	046688C0	1	0	0	1
580.0	02334455	046688C0	1	0	0	1
600.0	02334455	046688C0	1	0	0	1
620.0	02334455	046688C0	1	0	0	1
640.0	02334455	046688C0	1	0	0	1
660.0	02334455	046688C0	1	0	0	1
680.0	02334455	046688C0	1	0	0	1
700.0	02334455	046688C0	1	0	0	1
720.0	02334455	046688C0	1	0	0	1
740.0	02334455	046688C0	1	0	0	1
760.0	02334455	046688C0	1	0	0	1
780.0	02334455	046688C0	1	0	0	1
800.0	02334455	046688C0	1	0	0	1
820.0	02334455	046688C0	1	0	0	1
840.0	02334455	046688C0	1	0	0	1
860.0	02334455	046688C0	1	0	0	1
880.0	02334455	046688C0	1	0	0	1
900.0	02334455	046688C0	1	0	0	1
920.0	02334455	046688C0	1	0	0	1
940.0	02334455	046688C0	1	0	0	1
960.0	02334455	046688C0	1	0	0	1
980.0	02334455	046688C0	1	0	0	1
1000.0	02334455	046688C0	1	0	0	1
1020.0	02334455	046688C0	1	0	0	1
1040.0	02334455	046688C0	1	0	0	1
1060.0	02334455	046688C0	1	0	0	1
1080.0	02334455	046688C0	1	0	0	1
1100.0	02334455	046688C0	1	0	0	1
1120.0	02334455	046688C0	1	0	0	1
1140.0	02334455	046688C0	1	0	0	1
1160.0	02334455	046688C0	1	0	0	1
1180.0	02334455	046688C0	1	0	0	1
1200.0	02334455	046688C0	1	0	0	1
1220.0	02334455	046688C0	1	0	0	1
1240.0	02334455	046688C0	1	0	0	1
1260.0	02334455	046688C0	1	0	0	1
1280.0	02334455	046688C0	1	0	0	1
1300.0	02334455	046688C0	1	0	0	1
1320.0	02334455	046688C0	1	0	0	1
1340.0	02334455	046688C0	1	0	0	1
1360.0	02334455	046688C0	1	0	0	1
1380.0	02334455	046688C0	1	0	0	1
1400.0	02334455	046688C0	1	0	0	1
1420.0	02334455	046688C0	1	0	0	1
1440.0	02334455	046688C0	1	0	0	1
1460.0	02334455	046688C0	1	0	0	1
1480.0	02334455	046688C0	1	0	0	1
1500.0	02334455	046688C0	1	0	0	1
1520.0	02334455	046688C0	1	0	0	1
1540.0	02334455	046688C0	1	0	0	1
1560.0	02334455	046688C0	1	0	0	1
1580.0	02334455	046688C0	1	0	0	1
1600.0	02334455	046688C0	1	0	0	1
1620.0	02334455	046688C0	1	0	0	1
1640.0	02334455	046688C0	1	0	0	1
1660.0	02334455	046688C0	1	0	0	1
1680.0	02334455	046688C0	1	0	0	1
1700.0	02334455	046688C0	1	0	0	1
1720.0	02334455	046688C0	1	0	0	1
1740.0	02334455	046688C0	1	0	0	1

d_sim

[illegible][illegible]

2

[illegible]

5240,0	09999999	45454545	0000	1	0	0	1	0000	1	1
5260,0	09999999	22222222	0000	1	0	0	0	1	0000	1

d_sim



89/06/28
12:26:11

```
0.0 00000000 000 00000 000X Xr
50.0 00000000 000 00000 0000
100.0 00000000 000 00000 0000
150.0 00000000 000 00000 0000
200.0 10000000 000 00000 0000
250.0 10000000 000 00000 0000
300.0 10000000 000 00000 0000
350.0 00000000 000 00000 0000
400.0 00000000 000 00000 0000
450.0 00000000 000 00000 0000
500.0 12345600 046 45600 2280
550.0 12345600 046 45600 2280
600.0 12345600 000 00010 0000
650.0 00000010 000 00010 0001
700.0 00000010 000 00010 0001
750.0 00000020 000 00020 0001
800.0 00000020 000 00020 0001
850.0 00000020 000 00020 0001
900.0 00000030 000 00030 0001
950.0 00000030 000 00030 0000
1000.0 00000030 000 00030 0000
1050.0 00000030 000 00030 0000
1100.0 00000040 000 00040 0002
1150.0 00000040 000 00040 0002
1200.0 00000040 000 00040 0002
1250.0 00000050 000 00050 0002
1300.0 00000050 000 00050 0003
1350.0 00000050 000 00050 0003
1400.0 00000060 000 00060 0003
1450.0 00000060 000 00060 0003
1500.0 00000060 000 00060 0003
1550.0 00000070 000 00070 0003
1600.0 00000070 000 00070 0002
1650.0 00000070 000 00070 0002
1700.0 00000070 000 00070 0002
1750.0 00000070 000 00070 0002
1800.0 00000070 000 00070 0002
1850.0 00000070 000 00070 0002
1900.0 00000060 000 00060 0002
1950.0 00000060 000 00060 0003
2000.0 00000060 000 00060 0003
2050.0 00000050 000 00050 0003
2100.0 00000050 000 00050 0003
2150.0 00000050 000 00050 0003
2200.0 00000040 000 00040 0002
2250.0 00000040 000 00040 0002
2300.0 00000040 000 00040 0002
2350.0 00000030 000 00030 0000
2400.0 00000030 000 00030 0000
2450.0 00000030 000 00030 0000
2500.0 00000020 000 00020 0000
2550.0 00000020 000 00020 0001
2600.0 00000010 000 00010 0001
2650.0 00000010 000 00010 0001
2700.0 00000010 000 00010 0001
2750.0 00000000 000 00000 0000
2800.0 00000000 000 00000 0000
2850.0 00000000 000 00000 0000
2900.0 00000000 000 00000 0000
```

hash_sim

HASH FUNCTION SIMULATION

1

89/06/28
12:26:31

PAGE TABLE

1

pgtbl_sim

0.0	00000	XXXXX	0000	1
50.0	00000	XXXXX	0000	1
100.0	00000	XXXXX	0000	1
150.0	00000	XXXXX	0000	1
200.0	00000	XXXXX	0000	1
250.0	00000	XXXXX	0000	1
300.0	00000	XXXXX	0000	1
350.0	00000	XXXXX	0000	0
400.0	00000	XXXXX	0000	0
450.0	00000	XXXXX	0000	0
500.0	00000	XXXXX	0000	0
550.0	00000	00000	0000	1
600.0	00000	00000	0000	1
650.0	00000	00000	0000	1
700.0	00000	XXXXX	3FF5	1
750.0	00000	XXXXX	3FF5	1
800.0	00000	XXXXX	3FF5	1
850.0	00000	XXXXX	3FF5	1
900.0	02345	XXXXX	3FF5	0
950.0	02345	XXXXX	3FF5	0
1000.0	02345	XXXXX	3FF5	0
1050.0	02345	XXXXX	3FF5	0
1100.0	02345	02345	3FF5	1
1150.0	02345	02345	3FF5	1
1200.0	02345	02345	3FF5	1
1250.0	02345	02345	3FF5	1

TIME ^i ^o ^pg_ad ^r_w

89/06/20
12:47:14

VMEFSM SIMULATION

1

.ctcr.forces.do

```
do reset.do
forc node_d_in 00000000000000000000000000000000z
forc node_addr_in aabbccdd
forc vmebus_d_in 00ff11ee
forc vmebus_ad_in eeeeeeee
# power up everything
forc pwr_up_reset* 0
forc pwr_up_reset* 1 300
run 500
# try a ctc read request
forc busreq 1
force bbsy_in* 1
forc r_w* 1
forc hg3in* 0 400
forc ctc 1 800
forc dtack* 0 1400
forc dtack* 1 1600
forc dtack* 0 1800
forc dtack* 1 2000
forc dtack* 0 2200
forc dtack* 1 2400
forc dtack* 0 2600
forc dtack* 1 2800
run 4000
write list ctcr_simu -R
```

89/06/20
12:50:06

ctcw.forces.do

```
do reset.do
forc node_d in 00000334444555555666667777778888
forc node_addr in 00000000z
forc vmebus_d in 01234567
forc vmebus_ad in eeeeeeee
run 500
# Reset everything
forc pwr_up_reset* 0
forc pwr_up_reset* 1 300
run 500
# Try a cache to cache transfer : response (write)
forc ctc 1
forc ctc 0 400
forc r_w* 0
forc r_w* 1 400
forc dtack* 0 400
forc dtack* 1 600
forc dtack* 0 800
forc dtack* 1 1000
forc dtack* 0 1200
forc dtack* 1 1400
forc dtack* 0 1600
forc dtack* 1 1800
run 2000
# try a ctc with a berr*
forc ctc 1
forc ctc 0 400
forc r_w* 0
forc r_w* 1 400
forc berr* 0 400
run 5000
forc berr* 1
forc ctc 1
forc ctc 0 400
forc r_w* 1
run 800
# try it with a pwr_up_reset
forc ctc 1
forc ctc 0 400
forc r_w* 0
forc r_w* 1 400
forc dtack* 0 400
forc dtack* 1 600
forc pwr_up_reset* 0 800
run 2000
# try it with few berr
forc ctc 1
forc ctc 0 400
forc r_w* 0
forc r_w* 1 400
forc dtack* 0 400
forc dtack* 1 600
forc dtack* 0 800
forc dtack* 1 1000
forc berr* 0 1200
forc berr* 1 2000
forc dtack* 0 2000
forc dtack* 1 2200
forc dtack* 0 2400
forc dtack* 1 2600
forc dtack* 0 2800
forc dtack* 1 3000
forc dtack* 0 3200
forc dtack* 1 3400
```

```
run 4000
write list ctcw_simu -R
```

89/06/13
18:29:19

int.forces.do

1

```
do reset.do
forc node_d_in 00000000000000000000000000000000z
forc node_addr_in 00000000z
forc vmebus_d_in 01234567
forc vmebus_ad_in eeeeeeee
run 500
# try power up
forc pwr_up_reset* 0
forc pwr_up_reset* 1 200
run 500
# try interrupt request
# with correct response
# but first when iack* is low
forc iack* 0
forc int_req 1 400
forc iackin* 0 750
forc iack* 1 1000
forc iackin* 1 1000
forc iack* 0 1400
forc iackin* 0 1460
forc vmebus_ad_in 20000000 1700
forc ds0in* 0 1700
forc ds1in* 0 1700
forc ds0in* 1 2100
forc ds1in* 1 2100
forc int_req 0 2100
forc iack* 1 2200
forc iackin* 1 2200
run 2800
# try interrupt request, with false address response
forc iack* 1
forc iackin* 1
forc int_req 1 200
forc iack* 0 400
forc iackin* 0 460
forc vmebus_ad_in f0000000 700
forc ds0in* 0 700
forc ds1in* 0 700
forc ds0in* 1 2100
forc ds1in* 1 2100
forc iack* 1 2100
forc iackin* 1 2100
run 2500
# try interrupt request with acfail in middle
forc iack* 1
forc iackin* 1
forc int_req 1 200
forc iack* 0 460
forc iackin* 0 460
forc vmebus_ad_in f0000000 700
forc acfail* 0 700
forc ds0in* 0 2700
forc ds1in* 0 2700
period list 10000
run 201000
write list int_simu -R
```

89/06/13
18:29:43

```
do reset.do
forc node_d_in 0123456789abcdef0123456789abcdefz
forc node_addr_in baddac12
forc vmebus_d_in 00000001
forc vmebus_ad_in eeeeeeee
run 500
# Power up reset
forc pwr_up_reset* 0
forc pwr_up_reset* 1 300
run 500
# request bus while busy
forc ctc 0
forc busreq 1
forc bbsy_in* 0
run 400
# request bus now while not busy anymore just for a snoop tag exchange
forc bbsy_in* 1
forc r_w* 1
forc busreq 0 300
run 600
# request bus now while not busy for a data block
forc busreq 1
forc bbsy_in* 1
forc r_w* 1
forc data_block 1
forc dtack* 0 400
forc dtack* 1 600
forc dtack* 0 800
forc dtack* 1 1000
forc dtack* 0 1200
forc dtack* 1 1400
forc dtack* 0 1600
forc dtack* 1 1800
run 3000
# read with berr*
forc berr* 0 400
run 5000
forc berr* 1
forc ctc 1
forc ctc 0 400
forc r_w* 1
run 800
# try read with pwr in middle
forc ctc 1
forc ctc 0 400
forc r_w* 0
forc r_w* 1 400
forc dtack* 0 400
forc dtack* 1 600
forc pwr_up_reset* 0 800
run 2000
# try read with berr in middle
forc r_w* 1 400
forc dtack* 0 400
forc dtack* 1 600
forc dtack* 0 800
forc dtack* 1 1000
forc berr* 0 1200
forc berr* 1 2000
forc dtack* 0 2000
forc dtack* 1 2200
forc dtack* 0 2400
forc dtack* 1 2200
forc dtack* 0 2600
```

read.forces.do

```
forc dtack* 1 2800
forc dtack* 0 3000
forc dtack* 1 3200
run 4000
write list read_simu -R
```

89/06/20
12:55:23

reset.do

1

```
reset sim time
period list 100
clock period 100
forc ph10 1 15 -R
forc ph10 0 65 -R
clock period 100
forc ph11 1 70 -R
forc ph11 0 10 -R
forc bg3in* 1
forc bbsy_in* 1
forc bclr* 1
forc ds0in* 1
forc dslin* 1
forc iack* 1
forc iackin* 1
forc berr* 1
forc dtack* 1
forc acfail* 1
forc pwr_up_reset* 1
forc int_req 0
forc busreq 0
forc data_block 0
forc etc 0
forc r_w* 1
run 500
```

89/06/20
12:43:27

setup.do

1

```
check -nospike
define bus vmesigin bg3in* bbsy_in* bclr* ds0in* dslin* iack* iackin* berr* dtack* acfail
define bus vmesigout bg3out* bbsy* br3* as* ds0* dsl* write* irq2* iackout* sysfail* sysr
define bus nodesigin pwr_up_reset* int_req busreq data_block ctc r_w*
define bus nodesigout busgrnt data_ready system_error*
list b vmesigin vmesigout nodesigin nodesigout -c
list hex node_d_in node_d_out node_addr_in node_addr_out -c
list hex vmebus_d_in vmebus_d_out vmebus_ad_in vmebus_ad_out -c
```

89/06/20
12:50:50

write.forces.do

1

```
do reset.do
forc node_d_in 0123456789abcdef0123456789abcdef
forc node_addr_in baddac12
forc vmebus_d_in 00000001
forc vmebus_ad_in eeeeeeee
run 500
#try pwr_up_reset
forc pwr_up_reset* 0
forc pwr_up_reset* 1 300
run 500
# try bus request while bus busy...
forc busreq 1
forc bbsy_in* 0
forc ctc 0
run 400
# try write while bus not busy any more... for a snoop tag exchange
forc bbsy_in* 1
forc r_w* 0
forc busreq 0 400
run 600
# try write while bus not busy any more... for a data block
forc busreq 1
forc bbsy_in* 1
forc r_w* 0
forc data_block 1
forc dtack* 0 400
forc dtack* 1 600
forc dtack* 0 800
forc dtack* 1 1000
forc dtack* 0 1200
forc dtack* 1 1400
forc dtack* 0 1600
forc dtack* 1 1800
run 2000
# try write with berr*
forc berr* 0
run 5000
# try write with one berr*
forc berr* 1
forc dtack* 0 400
forc dtack* 1 600
forc berr* 0 800
forc berr* 1 1000
forc dtack* 0 1200
forc dtack* 1 1400
forc dtack* 0 1600
forc dtack* 1 1800
forc dtack* 0 2000
forc dtack* 1 2200
forc dtack* 0 2400
forc dtack* 1 2600
run 3000
# try it with pwr up
forc dtack* 0 400
forc dtack* 1 600
forc pwr_up_reset* 0 800
run 2000
write list write_simu -R
```