A Crossbar System for Multiprocessors

Vason P. Srini and Linda G. Bushnell

Computer Science Division, EECS

University of California

Berkeley, CA 94720

ABSTRACT

Inexpensive multiprocessor systems that obtain notable improvement in performance over sequential processors are currently under development at U.C. Berkeley. We are describing a crossbar system, an interconnection network, as a component of a multiprocessor system that may be used for experimentation with different processor architectures. For instance, one may wish to experiment with (1) interconnecting computing nodes, which contain processors, memory, and caches and (2) connecting processors to memory modules in a "dance hall" configuration. The crossbar system is based on a single bit-slice 16x16 crossbar chip with low latency, i.e., less than 50 ns of delay using a 2 micron static CMOS technology. The chip is designed so that it can be implemented in CMOS or EDFL GaAs. We used three different tools to develop this chip: (1) Lager Tools, (2) NCR and Mentor Graphics tools, and (3) Timberwolfe standard-cell tools. By stacking 33 of these chips, a crossbar system has been designed that interconnects sixteen processing elements (PE) for transferring 32 bits of data and address with one-cycle read/write capability, providing there is no contention between PEs. If a conflict occurs, a tree arbiter impartially selects a PE. A printed circuit board (PCB) version of the crossbar system has also been designed. This multilayer PCB acts as a backplane and contains the crossbar chips on one side and VME connectors to the PEs on the other side.

UCB/CSD Technical Report, October 1989

1. Introduction

A key component of a high-performance multiprocessor system is the interconnection network between the processors and memory modules or between processors. The bandwidth and the latency of the interconnection network are two factors that are significant in the performance of the multiprocessor system. For example, doubling the data path width between the processor and the memory may feasibly double the data available to a processor for a memory read. This may result in fewer memory references and conflicts and in more performance. Reducing the latency in the interconnection network reduces the memory access time, which may in turn improve the effectiveness of the system. Our goal is to design an interconnection network where bandwidth may be enlarged by increasing the width of the data path and latency may be reduced by decreasing the gate delay.

Although crossbars have been in use in computer systems [Bur69, WuB72, WuH78], they do not meet our bandwidth and latency requirements. For instance, the memory references in Prolog programs do not exhibit locality usually found in Pascal or C programs. This means a processor's memory references may change from module to module in adjacent cycles. A similar situation also arises in object oriented programs. To support this type of randomness in memory references, a dynamic crossbar with low latency for changing switch settings is needed. We have designed [Sri88, SrD89] a single bit-slice crossbar chip for connecting 16 processors to 32 memory modules. The switch connection in the crossbar can change every cycle based on the memory requests from the processors. By stacking 33 of these chips, we can have 32-bit words transferred in one cycle. The extra chip is for control. The total delay in the chip is equivalent to the delay through 20 gates. Because of the complexity of 16x32 crossbar system, we have also designed a simpler crossbar chip for interconnecting 16 PEs. This second crossbar system is described in this paper. It is easy to extend the simpler design to connect processors to memory.

Our design is different from that of other designs in the literature. It is oriented towards high performance implementation in submicron CMOS and GaAs technologies. The chip design proposed by Franklin [FWT82, WaF83] employs multiple stages and does not meet our latency requirements. The 16x16 crossbar chip in the MARS [Agr87] system for connecting processors does not meet our specifications. The arbiter used in the chip is based on positional priority, which may cause performance problems in logic programs because of the randomness in memory references.

Section 2 describes current multiprocessor systems for Prolog employing the crossbar. Section 3 details the crossbar chip and section 4 presents the testing of the chip. Section 5 describes the crossbar board. Section 6 gives some conclusions.

2. Multiprocessor System for Prolog

Several proposals for executing Prolog programs on multiple processors have appeared in the literature. Two of the proposals [Bor84, WAD84] extend the stack architecture of Warren [TiW83, War83]. The stacks are allowed to spread across the processors in the shared memory multiprocessor system [Bor84] as the parallelism grows. The AND parallelism and stream parallelism [Con83] are exploited in this multiprocessor. The distributed system of Warren [WAD84] supports the OR parallelism [Con83] in Prolog programs by distributing the clauses in a procedure to many processors connected to a broadcast network. We have been experimenting with a heterogeneous, tightly coupled, multiprocessor system [Dob86, NSD88] with special processors to execute logic programming at the control level and numeric processing to execute arithmetic functions. A block diagram of our proposed multiprocessor is shown in Figure 1. It employs 12 Parallel Prolog Processors (PPP) [SrD89], three floating point processors (FPP), a general purpose processor (GPP),snooping caches (SC), caches (C), memory (M), and the crossbar. The PPP in the figure is based on the Prolog machine (PLM), developed at Berkeley [DPD84, STN87], and will exploit the AND, OR, and stream parallelism in Prolog programs. A static analysis of Prolog programs will be done to detect the AND parallelism. One of the key components of the multiprocessor is the crossbar. There are other multiprocessors (NECTAR, RP3, etc.) that employ crossbars. Some of the early systems such as B6700 [Bur69] and C.mmp [WuB72] have also employed crossbars.

3. Crossbar Chip

The crossbar chip described in this paper is an interconnect-intensive and pad-limited chip. It contains 40,000 transistors. The pin-out of a single bit-slice 16x16 crossbar chip is shown in Figure 2. The inputs to the chip are indicated with a triangle pointing inward; the outputs have the triangles pointing outward. Signal names starting with a P are connected to the source PE, while signal names starting with an M are connected to the destination PE (or memory). These signals are explained in section 5 on the crossbar system design. The timing diagrams for read and write requests are shown in Figure 3.

3.1. Crossbar Chip Design

The physical design of the crossbar chip was done using three sets of tools. The first layout of the chip using the OCT/VEM/Mosaico tools [Spi88] resulted in a chip with a size of 28 mm by 21 mm using the MSU standard cells and a 2 micron CMOS technology. Since the die size exceeded our limit of 20 mm by 20 mm, no fabrication effort was made. The second attempt at layout was done using the Lager IV tools [Bro88] and improved MSU standard cells. The die size is 20 mm by 16 mm for a 2 micron CMOS process.

The third set of tools used in the layout process is the Mentor Graphics' Idea Station and NCR's Tangent router. The gate level design was completed using the Idea Station. The physical design has been completed using a 1.5 micro CMOS technology with NCR supplied standard cells. The resulting chip has a die size of 9 mm by 9 mm. This fabrication of this design with good yield appears to be feasible. The actual fabrication could use a 1.2 micron technology and thus bring the die size to 7.5 mm by 7.5 mm. The standard PGA packages with 210 leads can be used with the above die size, or a tape automated bonding (TAB) packaging technology may be used and the die surface mounted on a PC board. The chip is pad limited and uses metal2 quite heavily for the interconnections.

The bit-slice approach to crossbar design may cause word inconsistency problems [FWT82]. To avoid this problem, reset pins are included in the chip. All the crossbar chips of a word can be initialized to the same state using the reset signal coming from the PEs. Although the number of pins in a chip may be reduced by using bidirectional data lines for the read and write operations, we decided against it to keep the circuits simple and also to facilitate a GaAs implementation.

3.2. Crossbar Chip Components

The crossbar design has five components: decoder, arbiter, crosspoint matrix, input drivers, and output drivers. They are shown in the first sheet of the top-level schematic for the crossbar chip in Figures 4 and 5. The design details of each component are described in this section. To achieve low latency and simplicity in circuit design, chip area has been compromised. Every component contains many blocks, each of which has its delay estimated by running timing simulation. We iterated on the design and added bigger drivers to reduce the time delay in the blocks.

3.2.1. Decoder

There are sixteen 4 input to 16 output decoders per chip. Each decoder receives a 4-bit destination PE address and a PE request signal from a source PE . At most one output of a decoder will be high, denoting the destination PE requested by the source PE connected to it. There are many ways to design the decoder, the simplest and slowest being to use 5 input AND gates. The nominal delay through a 5 input NAND gate for a load of 0.2pf is 5.5 ns and the delay through an inverter is 1.0 ns. Another way is to use two levels of 3 and 2 input NAND gates and NOR gates. The nominal delay through a 3 input NAND gate for a load of 0.2pf is 2.5 ns and the delay through a 2 input NOR gate is 2.0 ns. The two-level design has a delay of 4.5 ns compared to 6.5 ns for a one level design. We chose the two-level design. Figures 6 through 9 show the symbols and logic of the decoder. Figure 7 is only a sample of the decoder's schematic sheet. The Decoder Array in Figure 4 is formed by using sixteen of the decoders.

3.2.2. Arbiter

Contention occurs when two or more PEs request the same destination PE in the same cycle. The arbiter selects one and informs the other PEs to try later by sending a collision signal. To prevent a single PE from holding up a destination PE while others are waiting for the same module, a fairness scheme is introduced in arbitration. Since there are 16 PEs that can simultaneously generate requests to the same destination PE, a one-of-sixteen arbiter is needed for each PE. We designed the arbiter as a tree of one-of-two arbiters shown in Figures 10 through 15. The sixteen requests to the arbiter are received at the leaf level nodes of the tree. Each of the one-of-two arbiters at this level communicates a signal on the request chain (REQC) output. The REQC outputs are the inputs to the arbiters at the next level of the tree. The REQC output will be high if any of the two inputs (REQ0 or REQ1) is high.

The one-of-two arbiter in Figure 13 uses a D flip-flop to provide fair response to requests. For example, if both REQ0 and REQ1 are high during successive cycles then the D flip-flop will be in state 0 (1) followed by state 1 (0), providing that GRANTC is one. This causes GRANT0 (1) followed by GRANT1 (0) to be one if GRANTC is one.

The delay through the arbiter is the sum of the time needed for the REQC signals in the arbiters to propagate to the root of the tree and for the GRANTC signal in the root to propagate to the leaf arbiters. This delay plays an important role in the duty cycle of the clock for the chip. Since the D flip-flops change state during the falling edge of the clock, the duty cycle of the clock must be greater than the propagation delay from the leaves to the root of the arbiter. Table 1 shows the state table of the one_of_two arbiter. The worst case delay (Temperature of 80 degrees Celsius and VDD = 4.5 V) for the arbiter is used in determining the duty cycle. The grant signals coming down from the root of the tree will be stable after clock goes low. This will allow the grant signals at the leaves to be stable when the clock is low.

3.2.3. Crosspoint Matrix

The crosspoint matrix component, shown in Figures 16 through 21, connects the 16 PEs to each other using 256 switches. Each switch is a combinational logic block and is shown in Figure 17. If PE i has been selected to communicate with PE j, then the signal g_i_j will be high. This signal allows the read, write, address, and data to be communicated to the PEs. Sixteen of these switches are grouped into a block where a wired-or connection is used for the read, write, address, and data signals. Since the switches are unbuffered the connections last for one cycle only, making the crossbar chip dynamic.

3.2.4. Input Drivers

The signals entering the chip from the PEs are directed to various components in the chip using drivers. The P_SYNC unit in Figures 22 through 26 has drivers for all signals coming from the input pads. It also supplies conflict information to the PEs.

3.2.5. Output Drivers

The read, write, address, and data signals are sent from the M_SYNC unit in Figures 27 through 30 using drivers. These signals first go to output pads and are then connected to the PEs. The pad drivers supply 8 milliamps of current.

4. Simulating the Crossbar Chip

The complexity of the design of the crossbar chip is augmented by the need to achieve a 50 ns cycle time. We used functional simulation to verify the data transfers from the inputs to outputs. The timing simulation used estimated capacitances to determine the delay through the blocks.

The simulation efforts used a "start small" approach. The functional simulation was performed in two stages. Each of the blocks of the crossbar was simulated separately, then the entire crossbar chip was simulated. The patterns used for simulating the chip are generated by programs and used as an input to the MG QuickSim tool. Since it was not possible to have a simulation run for each combination of read/write requests, patterns were selected from 1, 2, 4, 8, and 16 simultaneous requests to produce consistent test cases.

The timing simulations programs were run for each of the blocks to calculate the delay in each block. This allowed us to redesign some blocks by adding bigger drivers or by changing the circuits to reduce the delay and to meet the timing constraints.

After the crossbar chip design schematics were completed and verified, we used the MG tool Expand to transform the multi-level hierarchical schematics into a flat array of primitive elements consisting only of basic cells and wires. A basic cell is a digital component that has associated with it a computer program that models its functional behavior. The simulations were performed with the QuickSim tool, which interactively allows for the verification of the functionality and timing of the design. All simulation results followed specifications. The stimulants for the one of two arbiter and its output is shown in Table 2. The input and outputs of the one of 16 arbiter is shown in Table 3.

5. Crossbar System

An essential part of the multiprocessor system in Figure 1 is the crossbar interconnection network. This crossbar system employs the crossbar bit-slice chip and was designed for interconnecting 16 PEs. The system is interconnect intensive because each PE has two groups of 80 signals, and is a difficult one to simulate because of its size. There are over 2500 wires, 33 chips with 210 pins each, 288 driver chips with support for driving 2304 signals, and 32 VME DIN connectors.

To manage this wiring complexity, two approaches were explored. One approach used the wafer scale integration (WSI) and innovations in packaging technology. The MIT-Lincoln Laboratory has developed a WSI technique using laser cutting and welding. The technique has been demonstrated by building two signal processors on wafers [Raf85]. Using the WSI technique it is possible to interconnect 9 crossbar chips on a 10 cm (4 inch) diameter wafer using a redundancy factor of four. The redundancy will be used to bypass faulty chips. A new wafer packaging technique that will allow connections to

1200 pads on a wafer is needed for this approach to succeed along with software tools for programming the wafer.

The second approach is to use tape automated bonding (TAB) for the chips and surface mount the dies on a multilayer PC board. The latter approach is pursued in this research project.

In this design, the latency of the crossbar system is the delay through a single chip. Each PE supplies a 4-bit module address to the crossbar chip. The address within the module, data to be communicated, read operation, and write operation are specified by separate pins. Each PE receives address, data, read, and write signals on separate pins from a crossbar chip. If two or more PE address the same destination PE, one will be selected by the chip. The other PE will be sent conflict information. Each PE has a collision pin to receive the above information, 32 address pins, 32 data pins, four module indicator pins, and eight control pins to the crossbar system. The control pins are used to design the access protocol of VME.

5.1. Crossbar Board Design

In this section, we will describe how we connected 33 bit-slice crossbar chips to create the crossbar system that interconnects sixteen PEs for 32 bits of data and control communication. The multilayer crossbar board uses 33 crossbar chips, 32 VME J1P1/J2P2 connector pairs, and 288 driver chips to build a backplane for the multiprocessor system. The crossbar chips are placed on the component side of the board, while the VME connectors are on the connector side.

Figure 31 shows the top sheet of the crossbar board, which contains 33 crossbar chips labeled from zero to 32 to represent the data/address bits zero through 31 and the read/write collision signals control chip, respectively. To provide the required drive for the control signals, each signal is split into two signals via a driver. One signal goes to crossbar chips zero through sixteen; the other goes to chips seventeen through 32. A "Z" is added to the signal name attached to chips seventeen through 32. The control pins are used to design the protocol for communication between PEs using the VME access protocol.

Figure 32 shows the second sheet of the crossbar board, which contains the VME connectors for the sixteen PEs. We used a frame (for I = 0 to 15) around the logic to represent the sixteen PEs to simplify the Neted drawing. For one PE, we used the VME connector pair J1/P1 and J2/P2 along with 144 drivers (74F244) in 18 chips. On the J1/P1 connector, we used pins IRQ3* to IRQ7* for the PE control signals (four for processor module address and one for processor request). These are supposed to be used for interrupt requests, but we decided that only IRQ1* and IRQ2* are needed for interrupts in our design.

We used many 74F244 drivers to guarantee good signal strength at the destination. For example, the signal MDJ(I) for bit J (J = 0 to 31) and PE I (I = 0 to 15) shown in Figure 32 is sent from PE I through the J2/P2 VME connector to a 74F244 driver before it is sent to crossbar chip J. The drivers are controlled by the 32nd bit of the signals, i.e., MD32(I) controls the drivers that send the MDJ(I) (J = 0 to 31) from crossbar J to PE I. The signals PDIN and PD are multiplexed by using two drivers controlled by PDIN32(I) and PD32(I). The signals MDOT and MD are similarly multiplexed. The signals MA and PA did not need drivers, so they come directly out of the crossbar and VME connectors.

Figure 33 shows the inputs and outputs of a PE. The module request signals P0(3:0) to P15(3:0) are sent from PEs zero to fifteen, respectively, to each of the bit-slice crossbar chips. These signals are used to request a destination PE. For instance, PE zero can send the 4-bit address P0(3:0) of a desired destination PE to all 32 crossbar chips. To drive these signals to all 32 crossbar chips, the signals are put through two 74F244 drivers, as displayed in Figure 32 in the upper right corner. One driver's outputs are PJ(3:0) (for J = 0 to 15 for the sixteen PEs), which go to crossbar chips zero through sixteen. The other driver's outputs are PZJ(3:0), which go to crossbar chips seventeen through 32. This signal gave us the most trouble when designing this schematic sheet. To use the frame (for I = 0 to 15 for the sixteen PEs), we had to label these signals ('P'&I)(J) and ('PZ'&I)(J) (for J = 0 to 3) as shown in the upper right corner in Figure 32. When the design is prepared for layout and routing, the MG tool Expand interprets the & symbol to mean concatenate. If this was not done, we would have had to make a separate sheet for each PE, complicating the design and comprehensibility.

The processor request PR(15:0) is sent from the PEs to the crossbar chips. This signal issues a request for contacting a destination PE. For example, PR(0) is sent by PE zero to all 33 crossbar chips. To drive this signal, it is put through a 74F244 driver, as seen in Figure 32 in the upper right corner. The signal PR(I) (for PE I) is connected to two drivers. The outputs of the first driver are PR(J), which is sent to crossbar chips zero to sixteen, and the output of the second driver is PRZ(J), which is sent to crossbar chips seventeen to 32.

The processor address bus PAJ(15:0) (for J = 0 to 32 for the 32-bit address and control bit) is sent from all PEs to the crossbar chips. The signal PAJ(I) holds the Jth bit of the 32-bit address that is sent from PE I to the crossbar chips. For example, PE zero sends every crossbar chip (J = 0 to 32) the bit PAJ(0).

The processor data output bus PDJ(15:0) (for J = 0 to 32 for the 32-bit data and control bit) is sent from all PEs to the crossbar chips. The signal PDJ(I) holds the Jth bit of the 32-bit address that is sent from PE I to the crossbar chip J. For example, PE one sends the crossbar chip zero the bit PD0(1).

The processor data input bus PDINJ(15:0) (for J = 0 to 32 for the 32-bit data and control bit) is sent from the crossbar chips to the PEs. The signal PDINJ(I) holds the Jth bit of the 32-bit address that is sent from crossbar chip J to PE I. For instance, crossbar chip zero sends PE two the bit PDIN0(2).

The data from the destination PE (or memory module) MDOTJ(15:0) (for J = 0 to 32 for the 32-bit data and control bit) is sent from the destination PE to the crossbar chips. Similarly, the data into the destination PE MDJ(15:0) and the destination address MAJ(15:0) are sent from the crossbar chips to the destination PE.

The clock and reset signals also must be put through a driver, as displayed in Figure 32 in the upper right hand corner, to have the necessary drive to reach all 33 crossbar chips.

To illustrate the complexity of the crossbar system, we can look at a sample scenario. Suppose that PE 0 wants to send PE 1 32 bits of data and an address and PE 1 wants to send PE 2 data and an address all in the same cycle. PE 0 first sends PR(0) to all crossbar chips, indicating that it wants to communicate with another PE. It then sends the address of PE 1 in P0(3:0) to all crossbar chips. If there is no contention, PE 0 sends PDJ(0) (for J = 0 to 32) to the crossbar chips, which send this 32-bit data to PE 1 in one cycle. PE 1 receives the data as MDJ(1) (J = 0 to 32). Simultaneously, PE 1 sends PR(1) to all crossbar chips followed by the address of PE 2 in P1(3:0). If there are no conflicts, PE 1 sends PAJ(1) (for J = 0 to 32) to the crossbar chips, which send this 32-bit address to PE 2 as MAJ(2) (J = 0 to 32). All sixteen PEs can be reading and writing data and addresses in the same cycle, providing there is no contention for destination PEs.

5.2. Crossbar PCB

The placed and routed PCB for the crossbar board is shown in Figure 34. The PCB measures 20 by 13 inches, has a connector side and a components side, and has eight signal layers and two power layers. On the connector side, we placed the sixteen pairs of VME connectors, which are on the top and bottom of the board. The J1/P1 and J2/P2 connectors for PE zero are on the far left, with J1/P1 above J2/P2. The connectors for PE fifteen are on the far right. On the component side, we placed the 33 crossbar chips and the 288 drivers (74F244). The placement was done manually. We had to redo the placement a few times with a larger board size. Although the board has two sides, there can be no overlap between the VME connectors and the crossbar and driver components because we used through-pin PCB components. The routing was done automatically on the Apollo's DN3000. We are using many passes of the routing algorithms to completely route the PCB. The backplane may be used for other general purpose systems, such as the WARP system at Carnegie-Mellon University and the signal processing systems at UCB.

6. Conclusion

We have described the design and simulation of a bit-sliced crossbar chip and the design of a crossbar system that interconnects processing elements. The crossbar interconnection system is the main component of a multiprocessor system for Prolog that is currently under development at U.C. Berkeley. The crossbar system was designed to interconnect sixteen PEs, but can be easily extended to connect sixteen processors to 32 memory modules or to interconnect 32 computing nodes.

Although this is an on going project, some initial conclusions may be drawn based on design and simulation work. The timing simulation of the crossbar chip shows that using a 1.5 micron CMOS technology a latency time of 50 ns can be achieved for the crossbar chip. This means an extra cycle for read/write access when there is a cache miss and there are no conflicts assuming a 50 ns cache cycle time. This is important for the parallel execution of Prolog programs, which has heavy memory traffic. Since the miss ratios are well under five percent for many of the high performance systems, we expect no noticeable performance degradation if the crossbar system is used.

Since the crossbar chip and the crossbar system designs are huge, the VLSI tools have to be modified to accommodate more than 5000 wires and chips with more than 200 pins. This has motivated tool designers and builders to pay attention to memory management and space allocation per process.

Acknowledgement

We are thankful to Tara Weber for doing parts of the simulation of the crossbar chip. The help provided by Andrea Casotto, Jan Rabey, Bob Brodersen and members of the LagerIV team in understanding the tools while routing the blocks of the crossbar chip is appreciated. We also received help from members of the Aquarius group in the simulation of the chip. The routing of the crossbar chip at NCR Fort Collins was done by a team headed by Harold Crafts, Robin Steele, and Dan Ellsworth.

This work was sponsored by the Defense Advanced Research Projects Agency (DoD) and monitored by the Office of Naval Research under Contract No. N00014-88-0579 through the Electronics Research Laboratory, University of California, Berkeley. This research was also partially funded by NCR Dayton, Ohio and MICRO project of the state of California.

References

[Agr87] P. Agrawal, MARS: A Multiprocessor-based Programmable Accelerator, *IEEE Design & Test of Computers 5*, 4 (October 1987), 28-36.

[Bor84] P. Borgwardt, Parallel Prolog using Stack Segments on Shared-Memory Multiprocessors, *Proceedings of the 1984 International Symposium on Logic Programming*, Atlantic City, New Jersey, February 1984.

[Bro88] R. W. Brodersen, LagerIV Distribution 1.0 Silicon Assembly System Manual, *Electronics Research Laboratory*, University of California, Berkeley, June 1988.

[Bur69] Burroughs Corporation, Burroughs B6700 Reference Manual, 1969.

[Con83] J. S. Conery, The AND/OR Process Model for Parallel Interpretation of Logic Programs, *University of California*, Irvine, CA, June 1983. Ph.D. Thesis.

[DPD84] T. Dobry, Y. Patt and A. M. Despain, Design Decisions Influencing the Microarchitecture For A Prolog Machine, *Proceedings of the MICRO 17*, October 1984.

[Dob86] T. Dobry, Extending a Prolog Machine for Parallel Execution, *Proceedings of Hawaii International Conference on System Sciences*, Honolulu, Hawaii, January 1986.

[FWT82] M. A. Franklin, D. F. Wann and W. J. Thomas, Pin Limitations and partitioning of VLSI Interconnection Networks, *IEEE Transactions on Computers C-31*, 11 (November 1982).

[NSD88] T. M. Nguyen, V. P. Srini and A. M. Despain, A Two-Tier Memory Architecture for High-Performance Multiprocessor Systems, *Proceedings of the International Conference on Super Computing*, Saint-Malo, France, July 1988.

[Raf85] J. I. Raffel, A Wafer-Scale Digital Integrator Using Restructurable VLSI, *IEEE Journal of Solid-State Circuits SC-20*, 1 (February 1985).

[Spi88] R. Spickelmier, Oct Tools Distribution 2.1, *Electronics Research Laboratory*, University of California, Berkeley, March 1988.

[STN87] V. P. Srini, J. T. Tam, T. M. Nguyen, Y. N. Patt and A. M. Despain, A CMOS Chip for Prolog, *Proceedings of the International Conference on Computer Design*, Rye Town, NY, October 1987.

[Sri88] V. P. Srini, An Improved Bit-sliced Cross-connect Chip for use in Multiprocessor Computer Systems, *Patent Application*, University of California, Berkeley, October 1988.

[SrD89] V. P. Srini and A. M. Despain, Multiprocessor Research for Prolog, *Proceedings of the State of California MICRO-1988 Report*, January 1989.

[TiW83] E. Tick and D. H. D. Warren, *Towards a Pipelined Prolog Processor*, SRI International, Menlo Park, CA, August 1983. Technical Report.

[WaF83] D. F. Wann and M. A. Franklin, Asynchronous and Clocked Control Structure for VLSI-Based Interconnection Networks, *IEEE Transactions on Computers C-31*, 3 (March 1983).

[War83] D. H. D. Warren, *An Abstract Prolog Instruction Set*, SRI International, Menlo Park, CA, 1983. Technical Report.

[WAD84] D. S. Warren, M. Ahamad, S. K. Debray and L. V. Kale, Executing Distributed Prolog Programs on a Broadcast Network, *Proceedings of the 1984 International Symposium on Logic Programming*, Atlantic City, NJ, February 1984.

[WuB72] W. A. Wulf and C. G. Bell, C.mmp - A Multi Miniprocessor, *Proceedings of the AFIPS Fall Joint Computer Conference*, Montvale, New Jersey, 1972.

[WuH78] W. A. Wulf and S. P. Harbison, Reflections in a Pool of Processors, *Proceedings of the AFIPS Fall Joint Computer Conference*, Montvale, New Jersey, 1978.

Table I
State Transition Table for one of two Arbiter

Present State	INPUTS			Next State	OUTPUTS		
	RQ0	RQ1	GRC		GR0	GR1	RQC
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1
0	1	1	1	1	0	1	1
0	-	-	0	0	0	0	-
0	0	0	1	0	0	0	0
0	0	1	1	1	0	1	1
1	0	0	0	1	0	0	0
1	0	1	1	1	0	1	1
1	1	1	1	0	1	0	1
1	-	-	0	1	0	0	-
1	0	0	1	1	0	0	0
1	1	0	1	0	1	0	1

Table 2 Simulation Results of One of Two Aribiter

```
 TIME   ^REQ0     ^REQC     ^CLK      ^SETB
           ^REQ1     ^GRANT0   ^CLKB
              ^GRANTC   ^GRANT1   ^RESETB
   0.0 Xr  Xr  Xr  X   X   X   0   1   0   1
  55.0 Xr  Xr  Xr  X   0   0   1   0   0   1
  90.0 Xr  Xr  Xr  X   X   X   0   1   1   1
 100.0 0   0   0   X   X   X   0   1   1   1
 200.0 0   0   0   0   0   0   0   1   0   1
4100.0 1   1   1   1   1   0   0   1   0   1
4155.0 1   1   1   1   0   0   1   0   0   1
4190.0 1   1   1   1   0   0   0   1   0   1
4200.0 1   0   1   1   0   0   0   1   1   1
4204.0 1   0   1   1   1   0   0   1   1   1
4304.0 1   1   1   1   0   1   0   1   1   1
4404.0 1   1   1   1   1   0   0   1   1   1
4504.0 1   1   1   1   0   1   0   1   1   1
4604.0 1   0   1   1   1   0   0   1   1   1
4704.0 0   1   1   1   0   1   0   1   1   1
4804.0 1   1   1   1   1   0   0   1   1   1
4904.0 0   1   1   1   0   1   0   1   1   1
5104.0 1   1   1   1   1   0   0   1   1   1
5204.0 1   1   1   1   0   1   0   1   1   1
5304.0 1   0   1   1   1   0   0   1   1   1
5404.0 1   1   1   1   0   1   0   1   1   1
5504.0 1   1   1   1   1   0   0   1   1   1
5600.0 1   1   1   1   0   1   0   1   1   1
```

Table 3 Input and Outputs of One of 16 Aribter

TIME	^PREQ	^PGRNT	^CLK	^SET	^RESET
200.0	0001	0000	0	0	0
212.0	0001	0001	0	0	0
300.0	0002	0001	0	0	0
302.0	0002	0002	0	0	0
400.0	0004	0002	0	0	0
407.0	0004	0004	0	0	0
500.0	0008	0004	0	0	0
502.0	0008	0008	0	0	0
600.0	0009	0008	0	0	0
607.0	0009	0001	0	0	0
755.0	0009	0008	1	0	0
790.0	0009	0008	0	0	0
855.0	0009	0001	1	0	0
955.0	0009	0008	1	0	0
1000.0	000A	0001	0	0	0
1004.0	000A	0002	0	0	0
1055.0	000A	0002	1	0	0
1155.0	000A	0008	1	0	0
1255.0	000A	0002	1	0	0
1355.0	000A	0008	1	0	0
1400.0	000C	0002	0	0	0
7100.0	FFFF	0000	0	0	0
7113.0	FFFF	0100	0	0	0
7201.0	FFFF	0001	0	0	0
7301.0	FFFF	1000	0	0	0
7401.0	FFFF	0010	0	0	0
7501.0	FFFF	0400	0	0	0
7601.0	FFFF	0004	0	0	0
7701.0	FFFF	4000	0	0	0
7801.0	FFFF	0040	0	0	0
7901.0	FFFF	0200	0	0	0
8001.0	FFFF	0002	0	0	0
8101.0	FFFF	2000	0	0	0
8201.0	FFFF	0020	0	0	0
8301.0	FFFF	0800	0	0	0
8401.0	FFFF	0008	0	0	0
8501.0	FFFF	8000	0	0	0
8601.0	FFFF	0080	0	0	0
8701.0	FFFF	0100	0	0	0
8902.0	0000	0000	0	0	0
TIME	^PREQ	^PGRNT	^CLK	^SET	^RESET

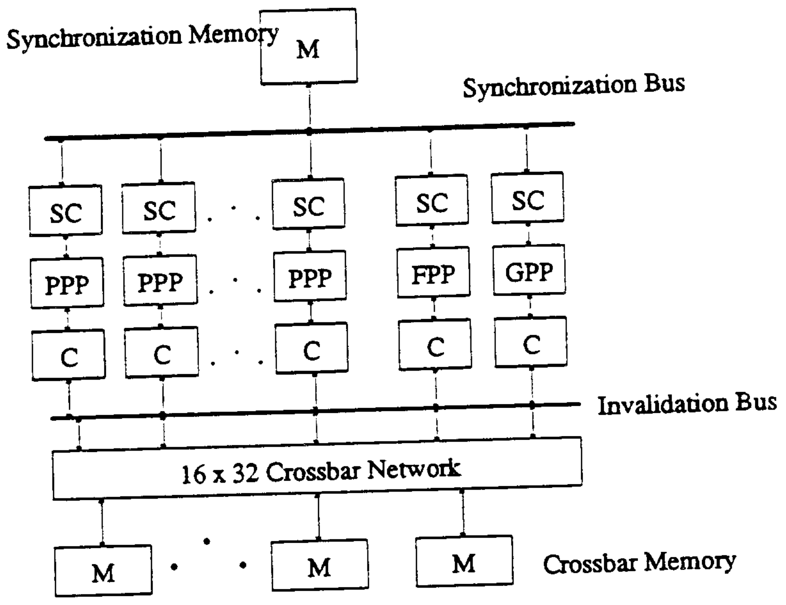


Figure 1 Multiprocessor System for Prolog

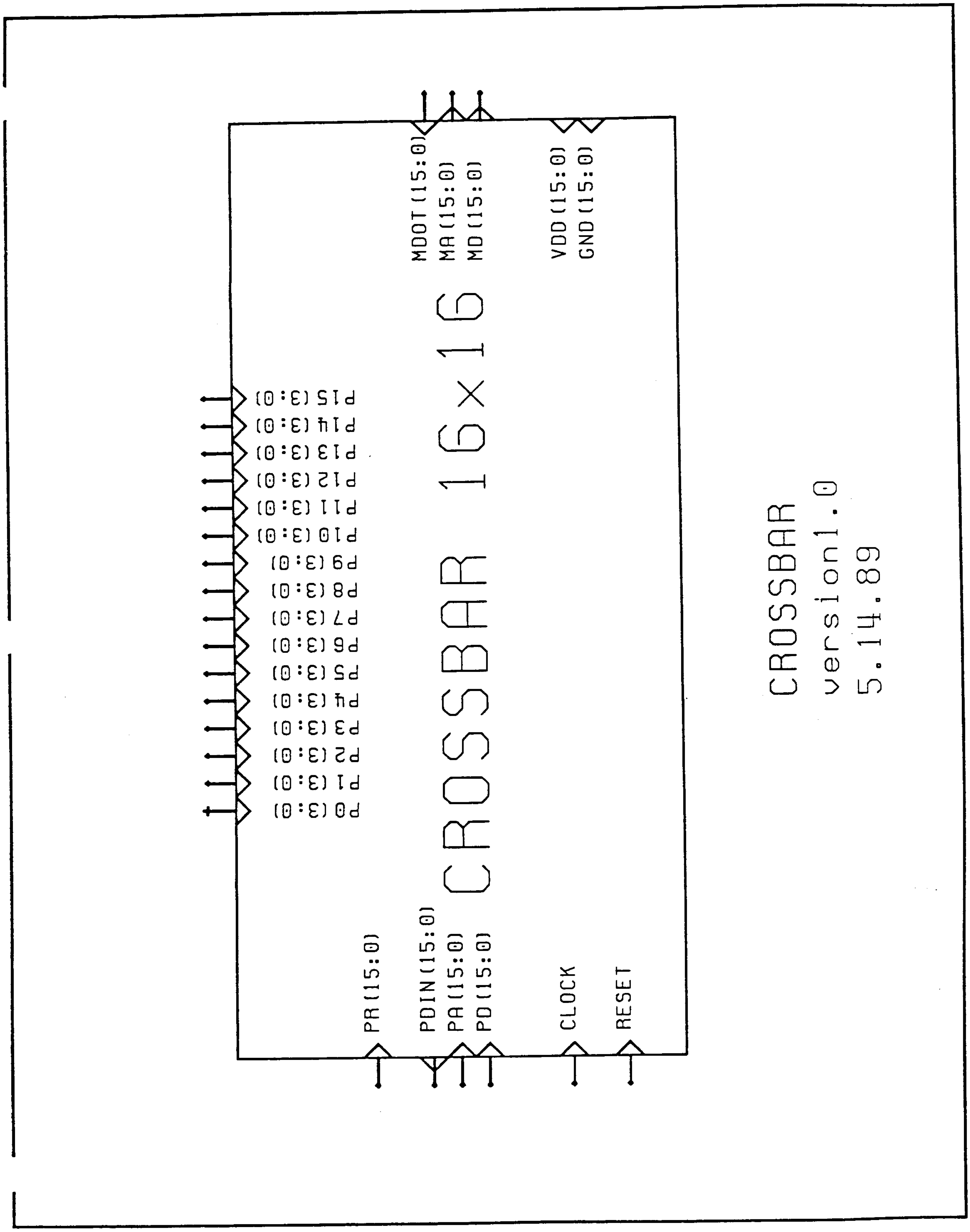


Figure 2 Crossbar Chip Symbol

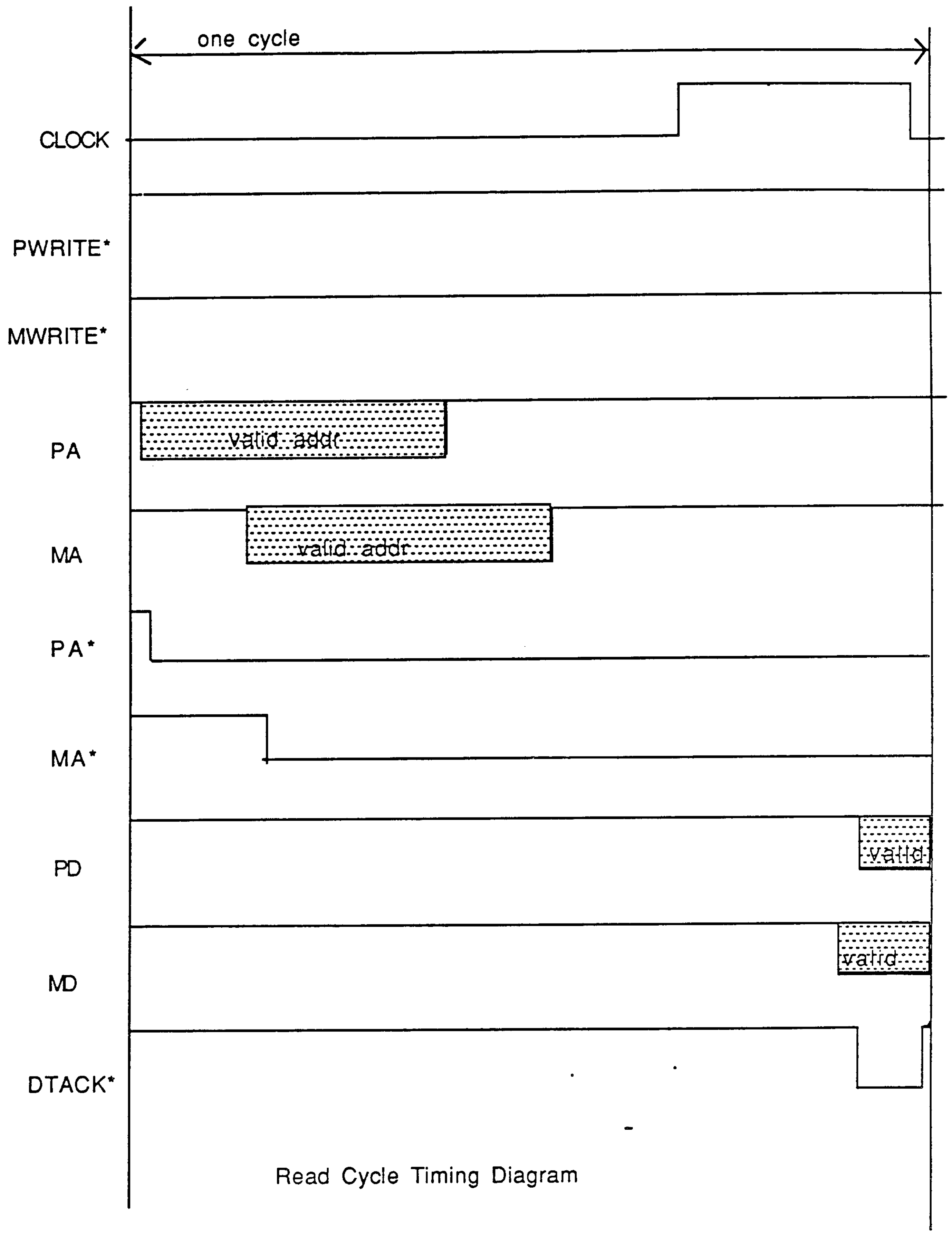
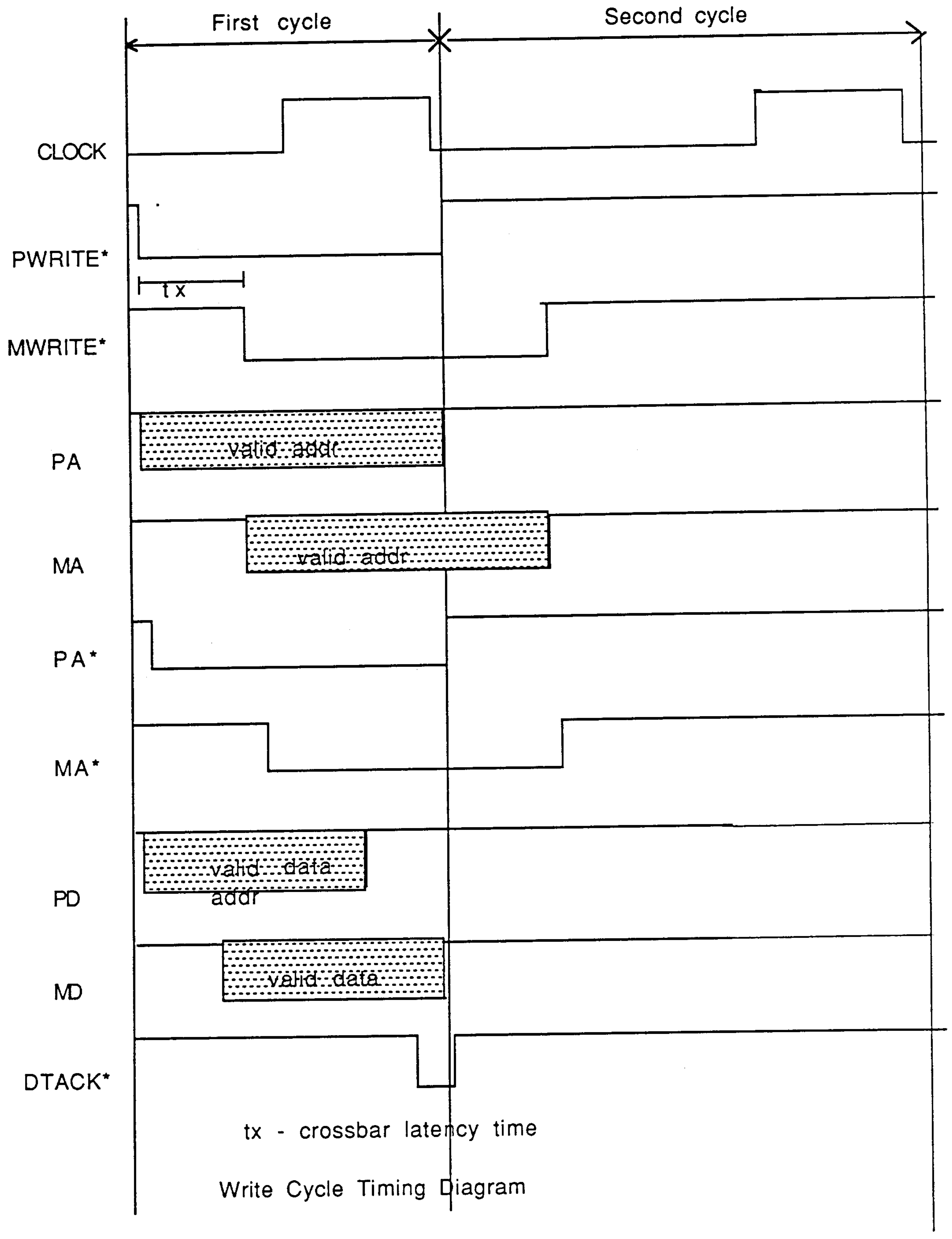


Figure 3 Read and Write Cycle Timing Diagram



tx - crossbar latency time

Write Cycle Timing Diagram

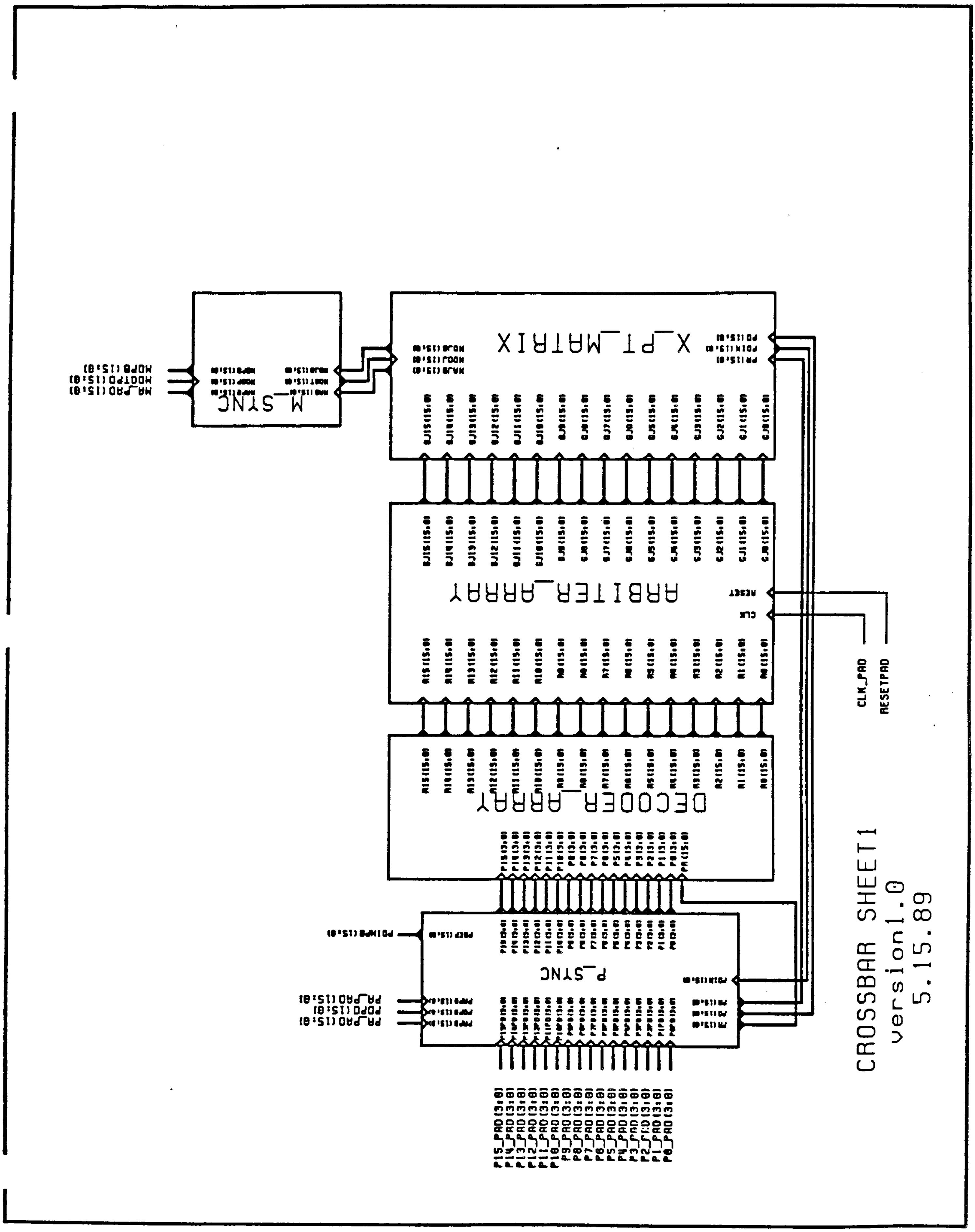


Figure 4 Top-level of Crossbar Chip Sheet 1

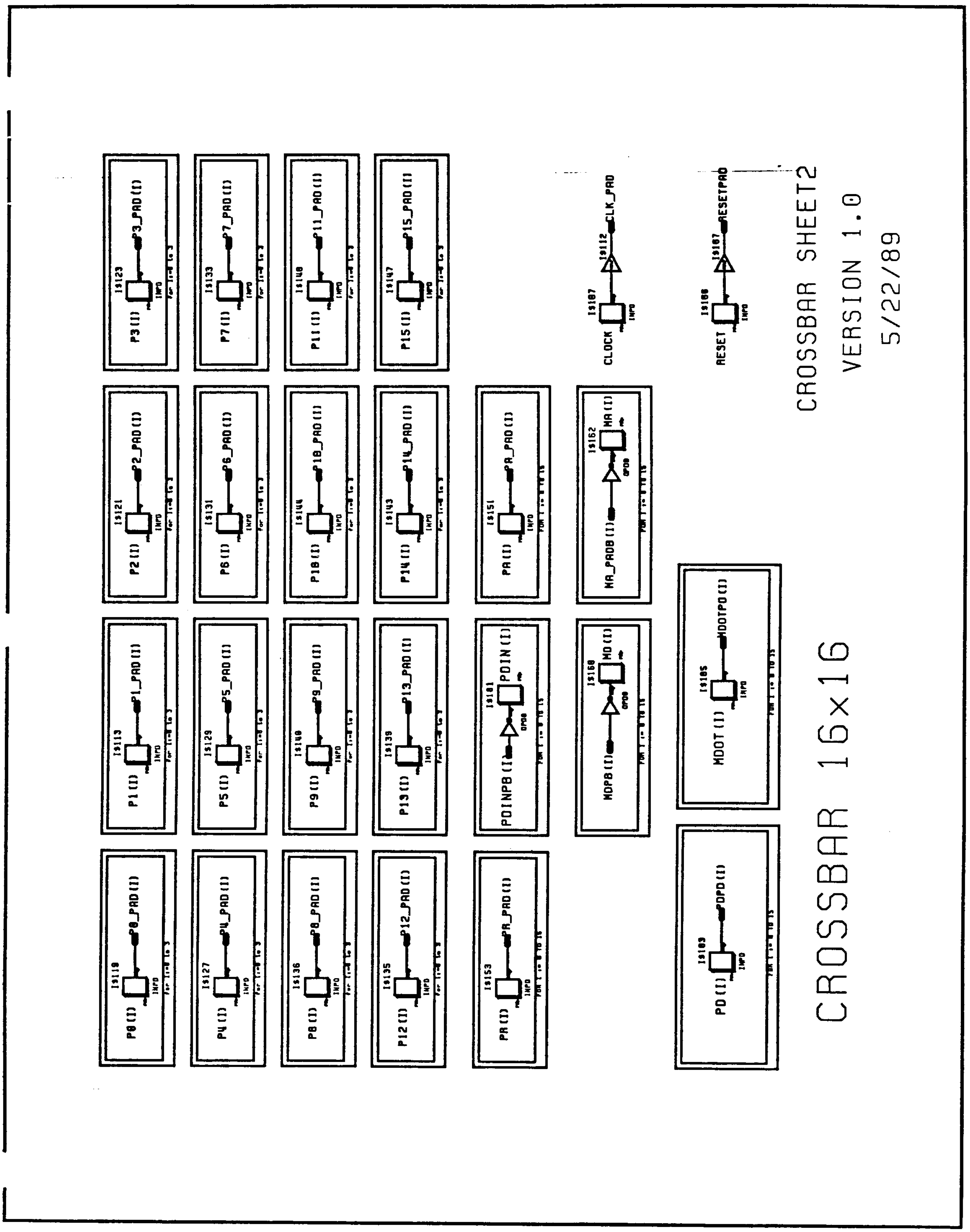


Figure 5 Top-level of Crossbar Chip Sheet 2

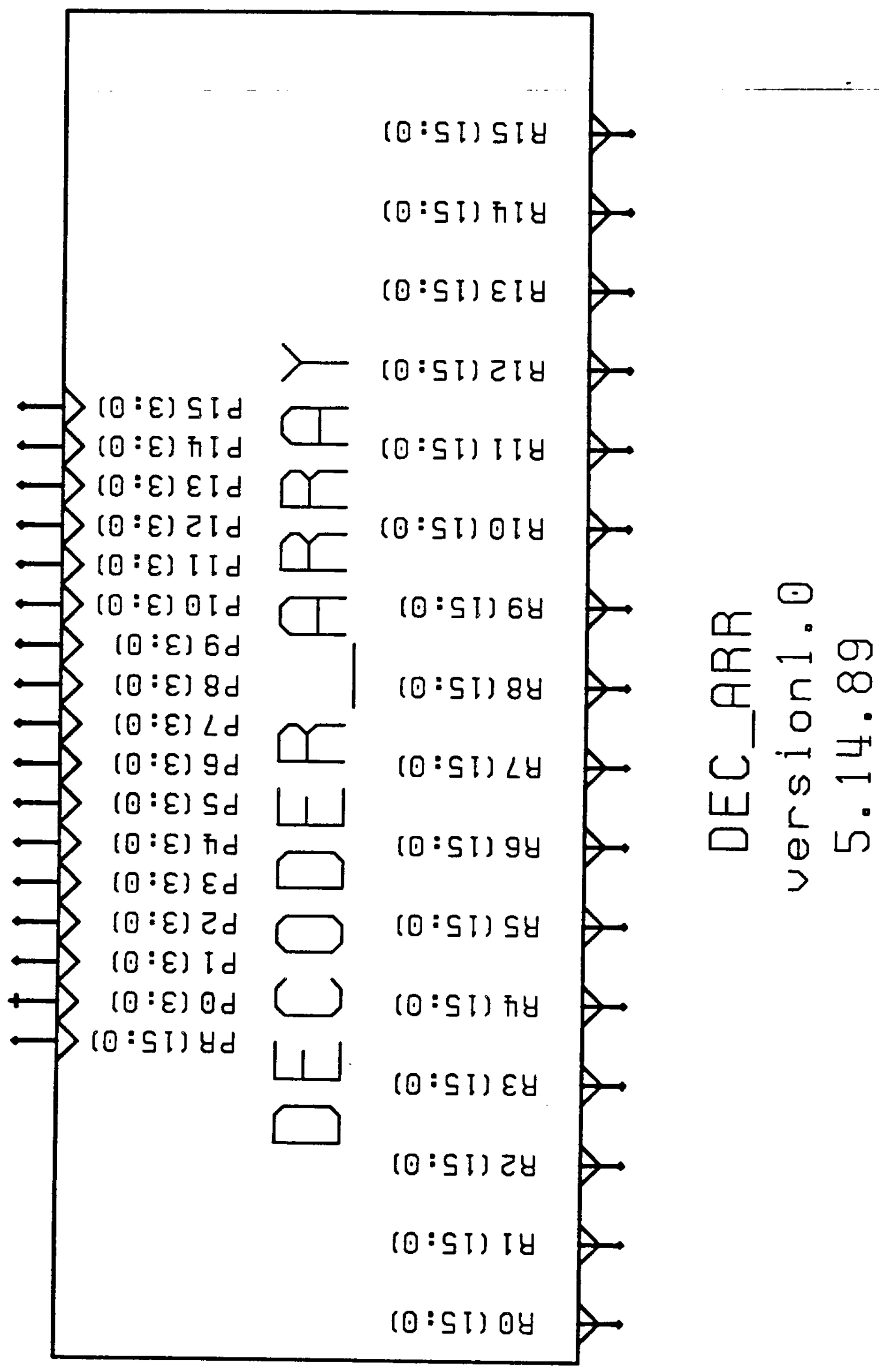


Figure 6 Decoder Array Symbol

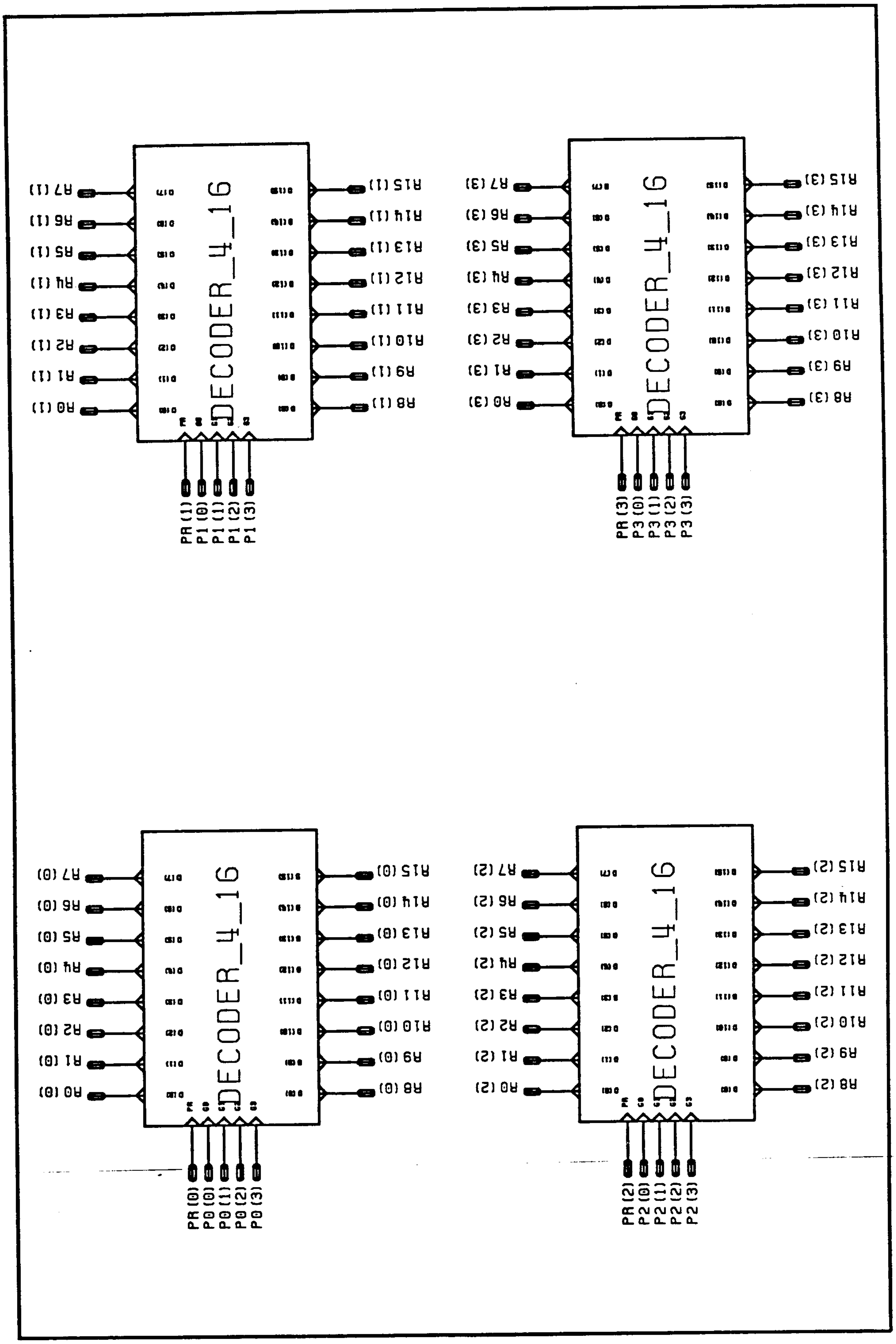


Figure 7 Decoder Array Schematic

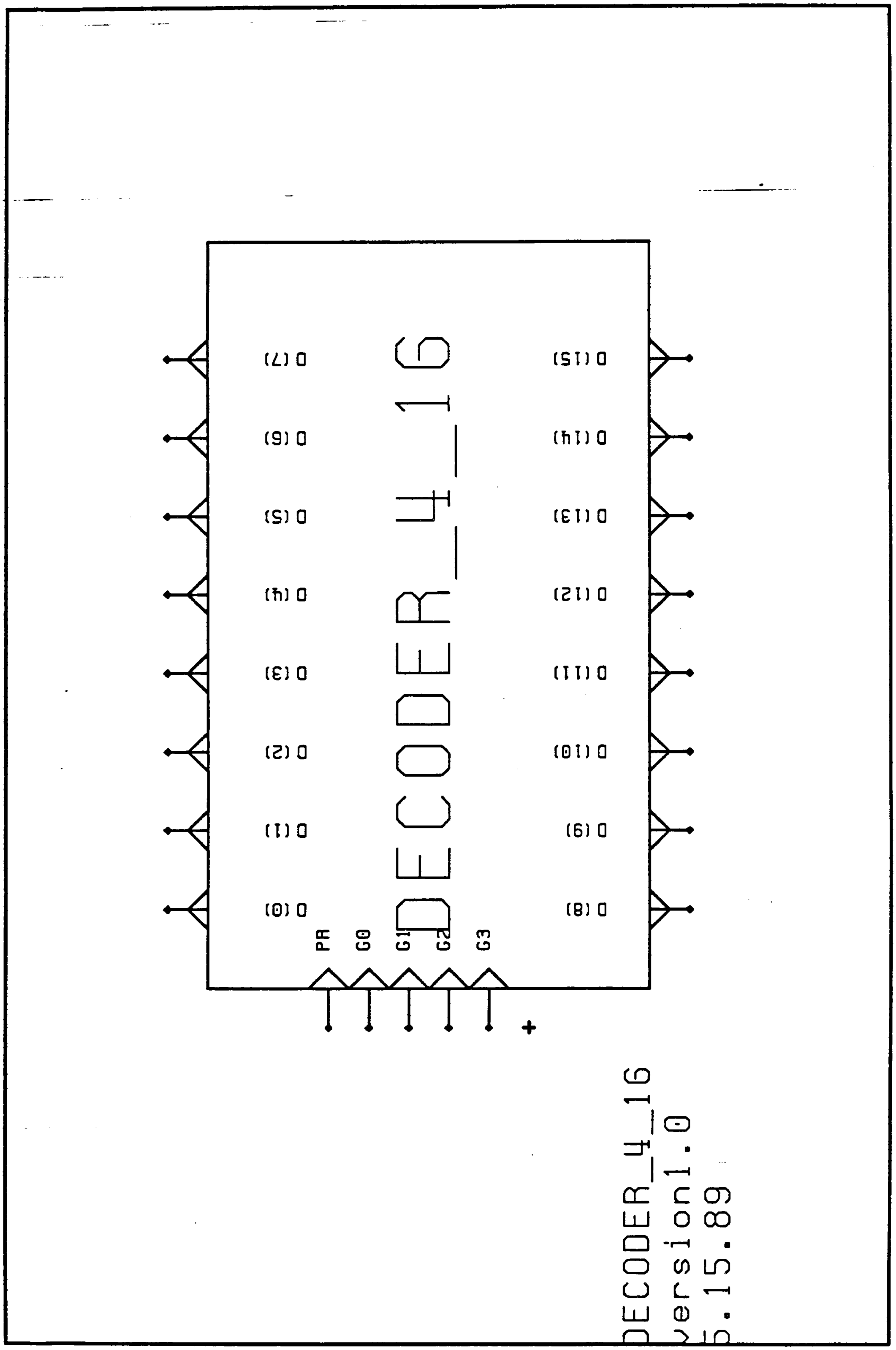


Figure 8 Decoder_4_16 Symbol

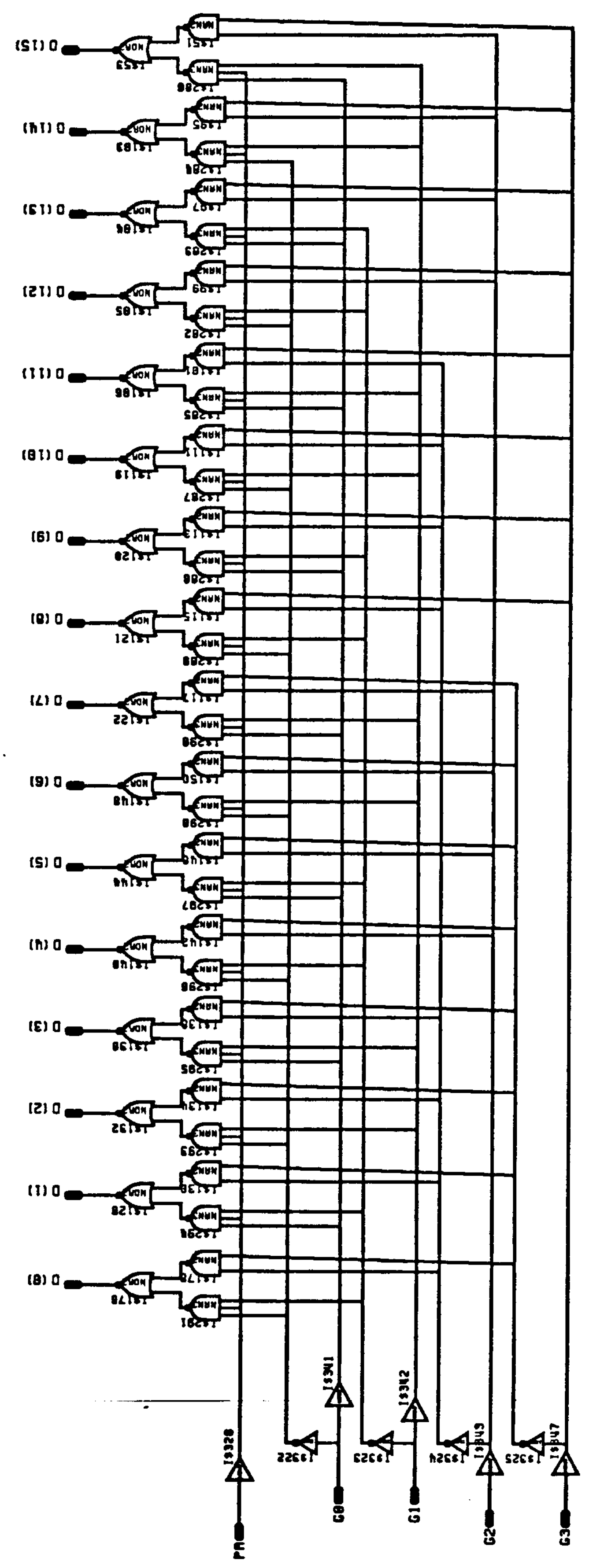


Figure 9 Decoder_4_16 Schematic

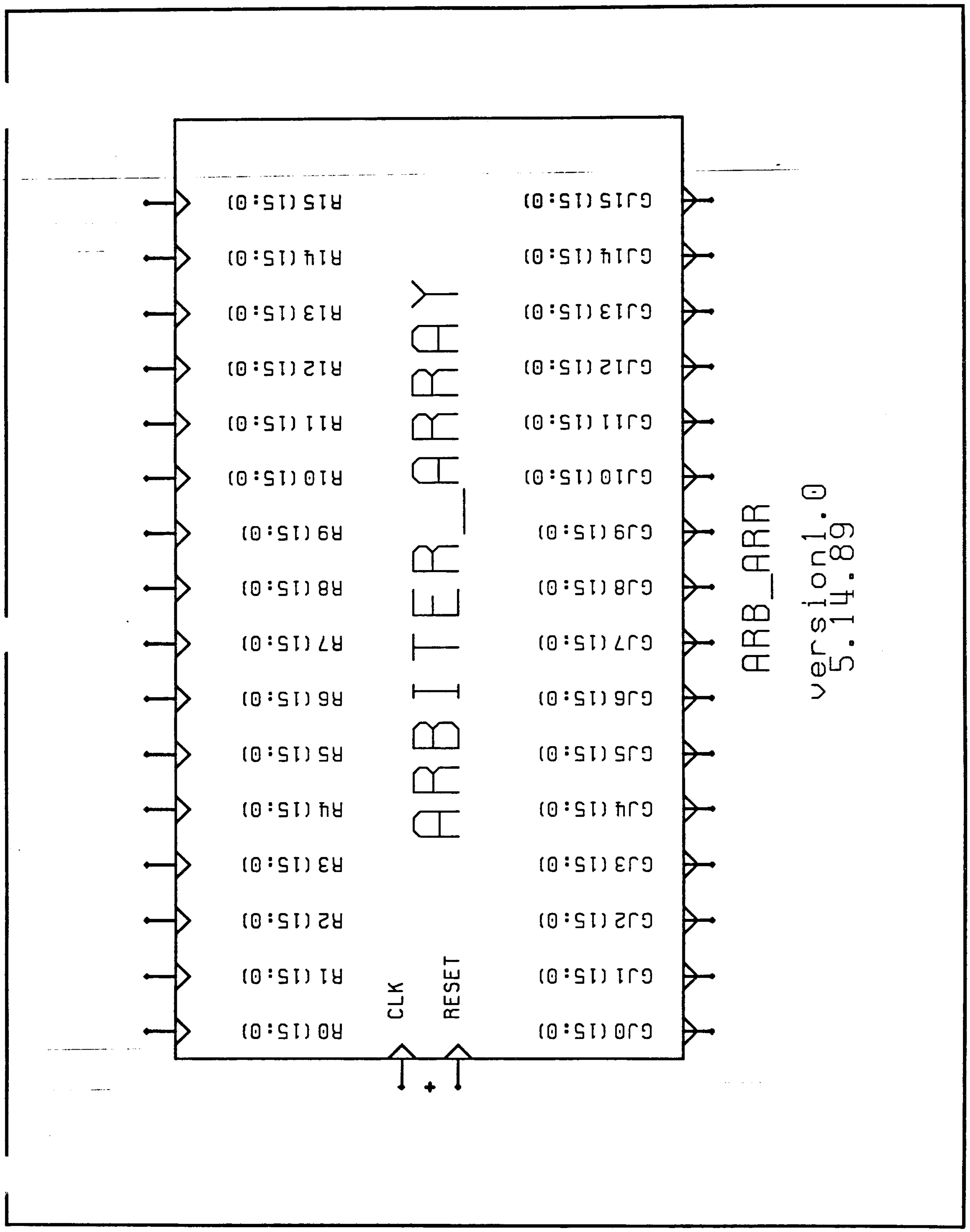


Figure 10 Arbiter Array Symbol

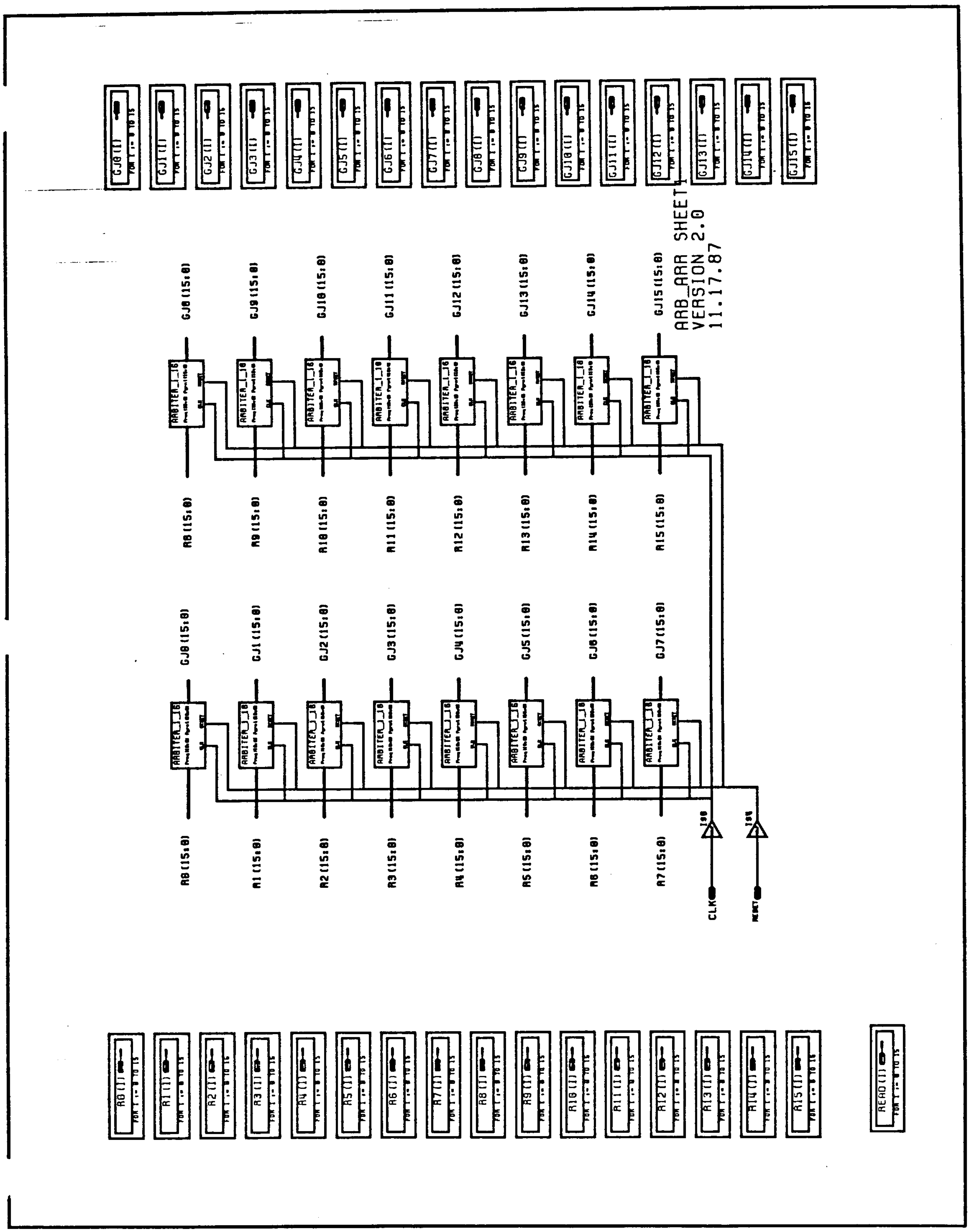


Figure 11 Arbiter Array Schematic

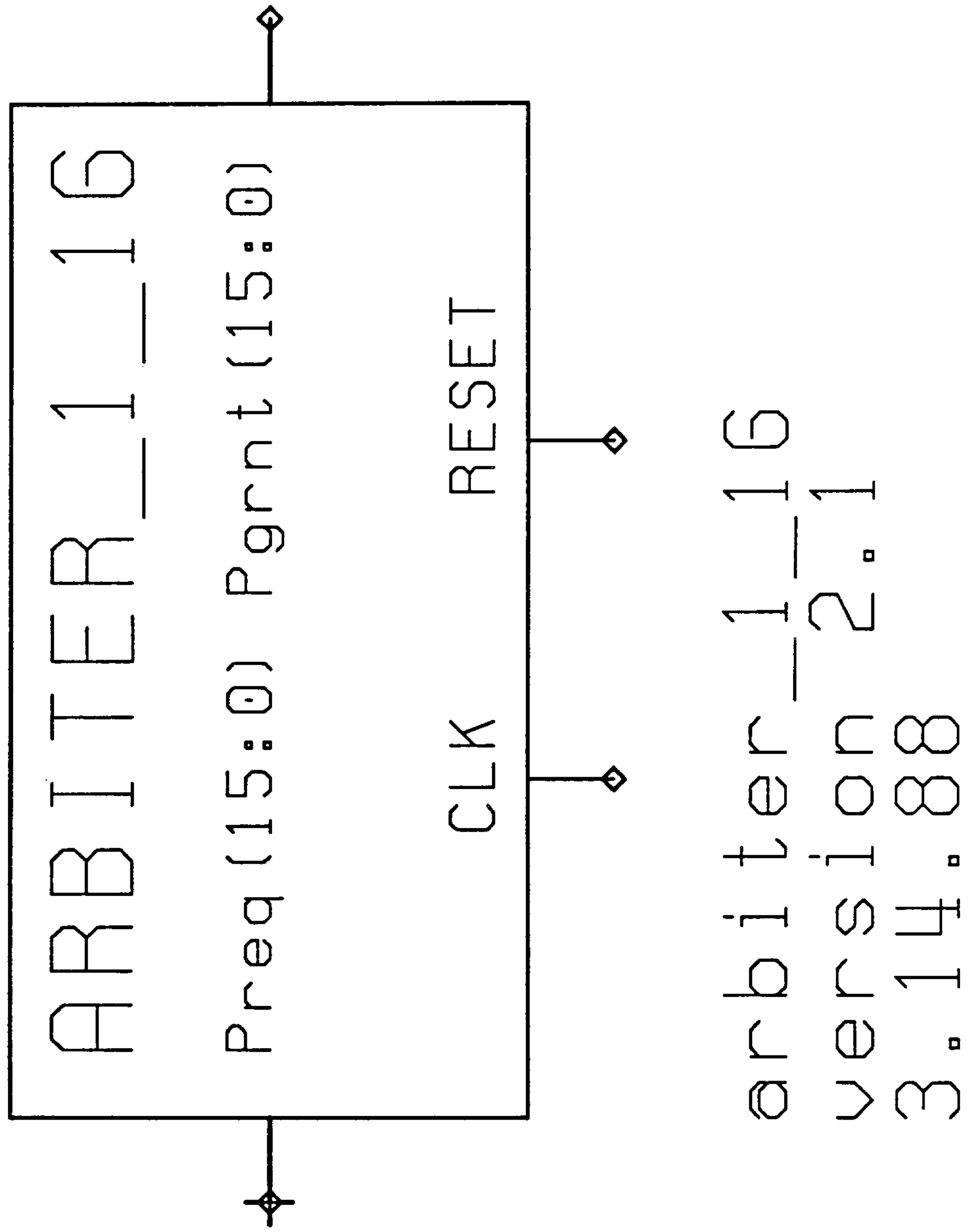


Figure 12 Arbiter_1_16 Symbol

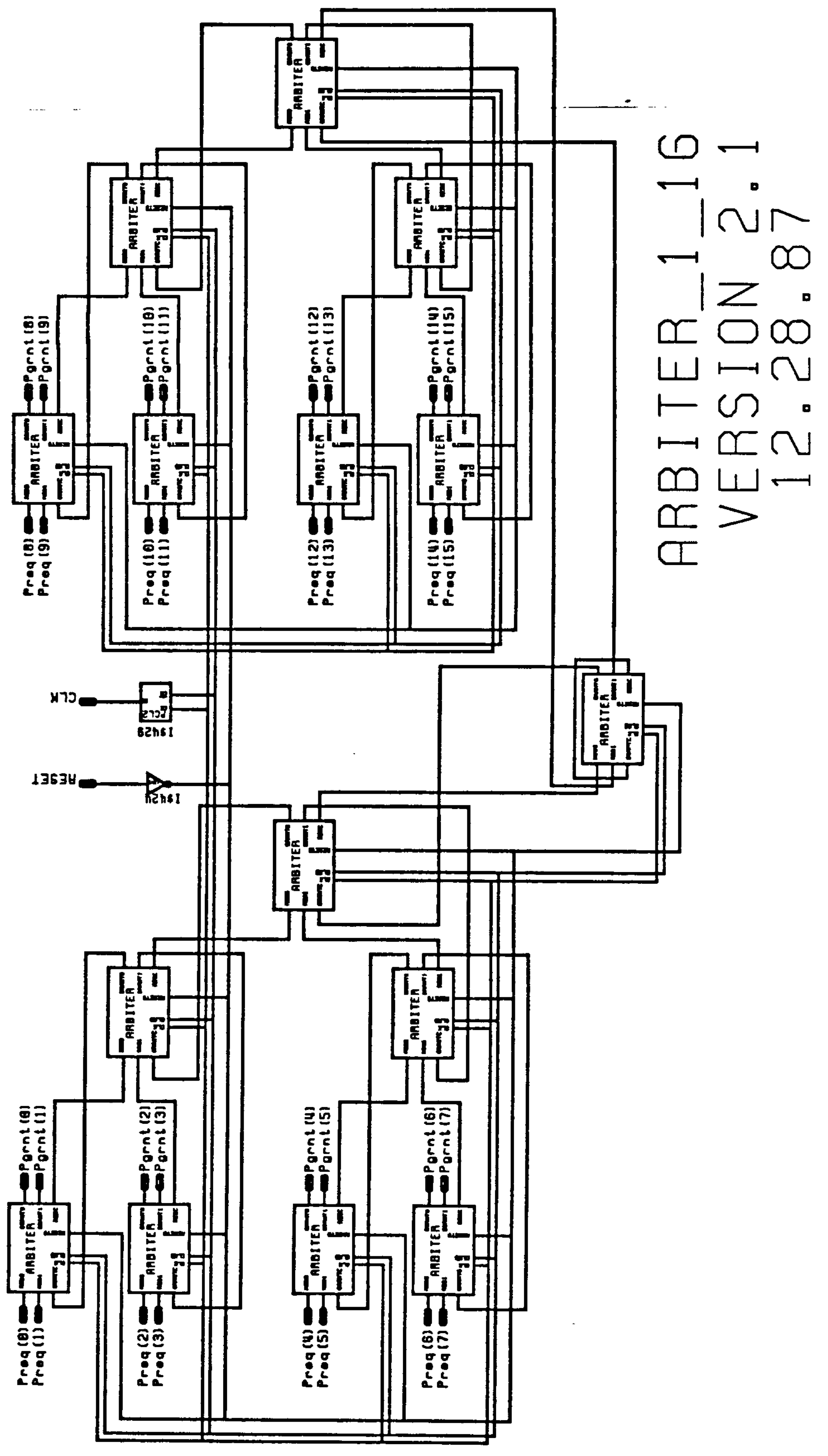


Figure 13 Arbiter_1_16 Schematic

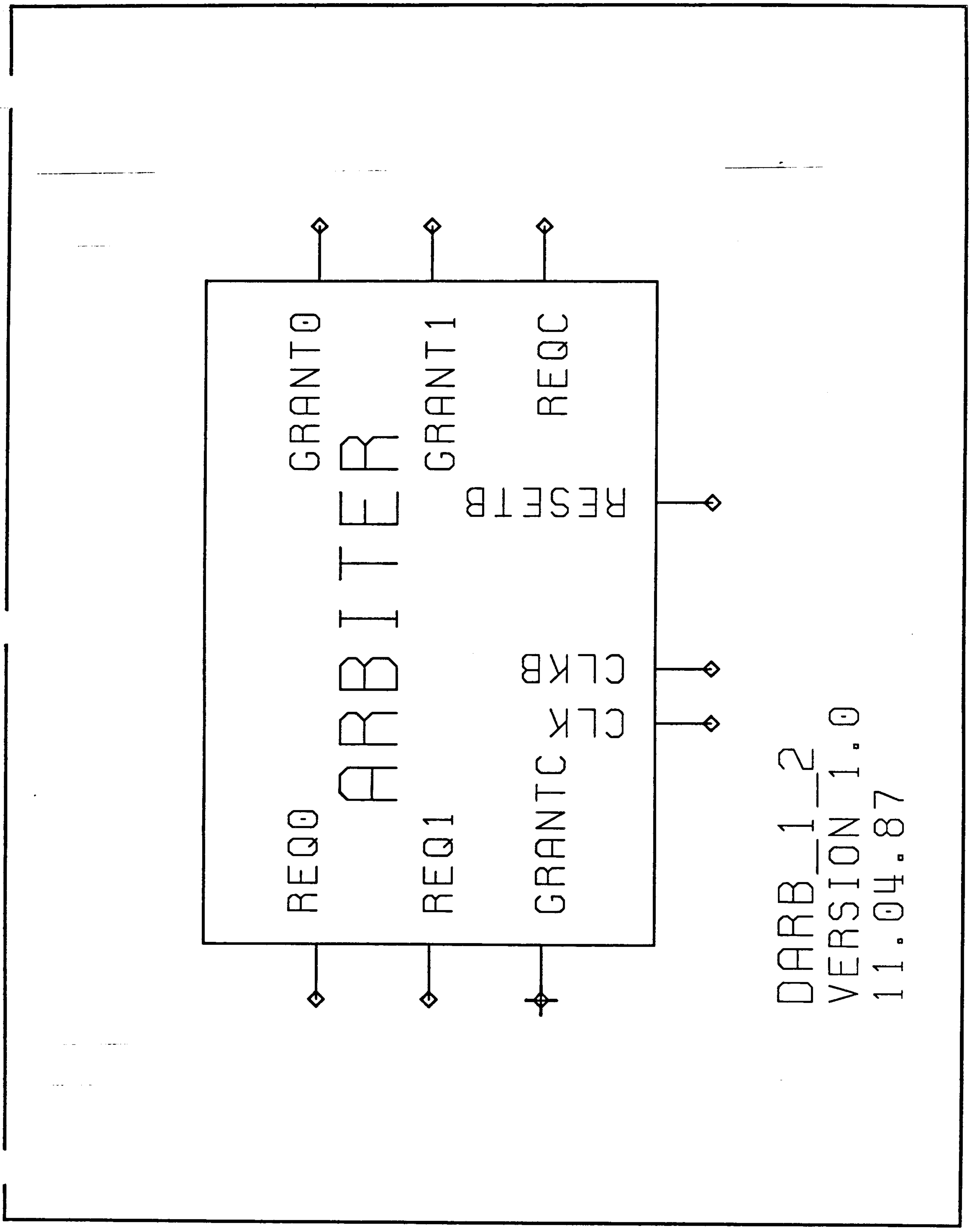


Figure 14 Arbiter Symbol

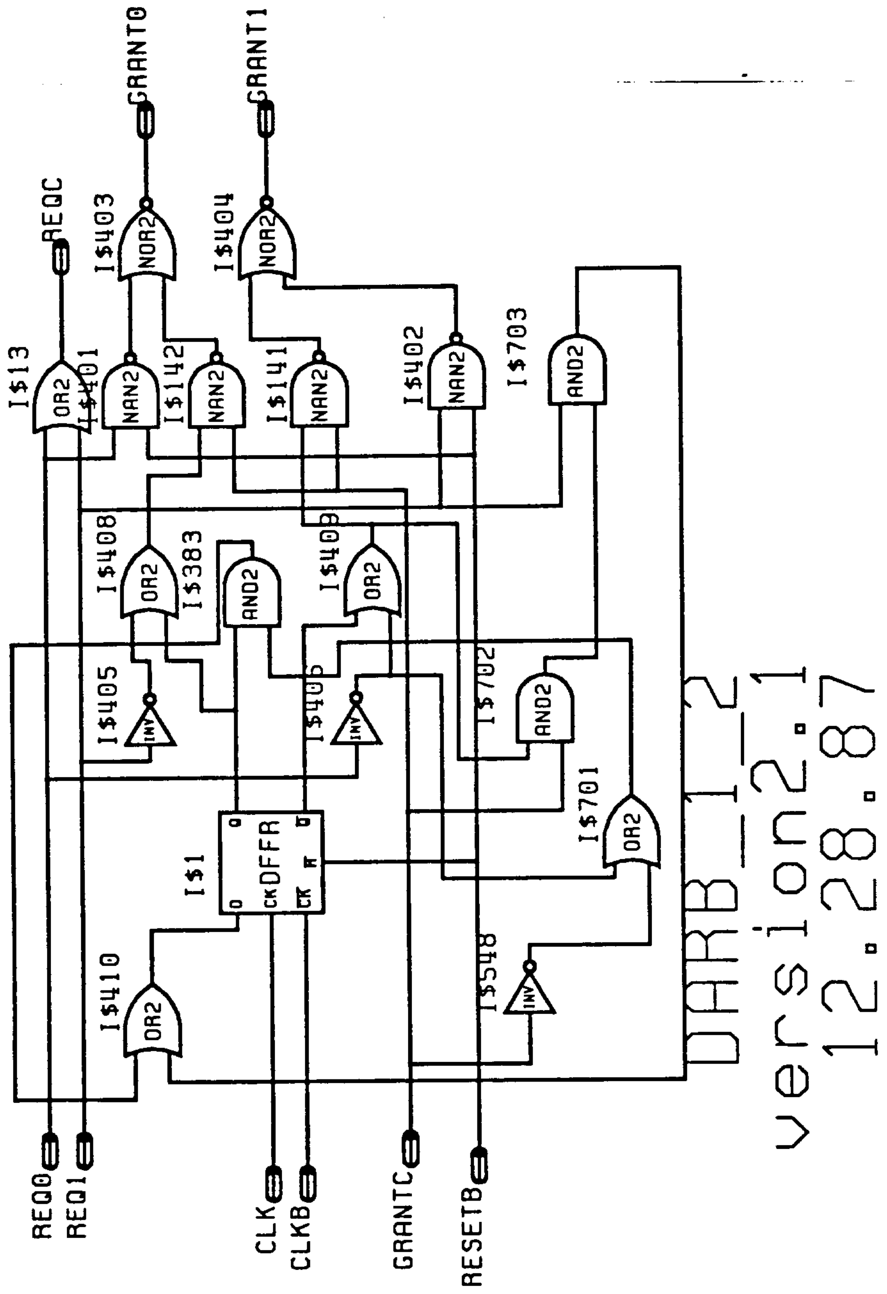


Figure 15 Arbiter Schematic

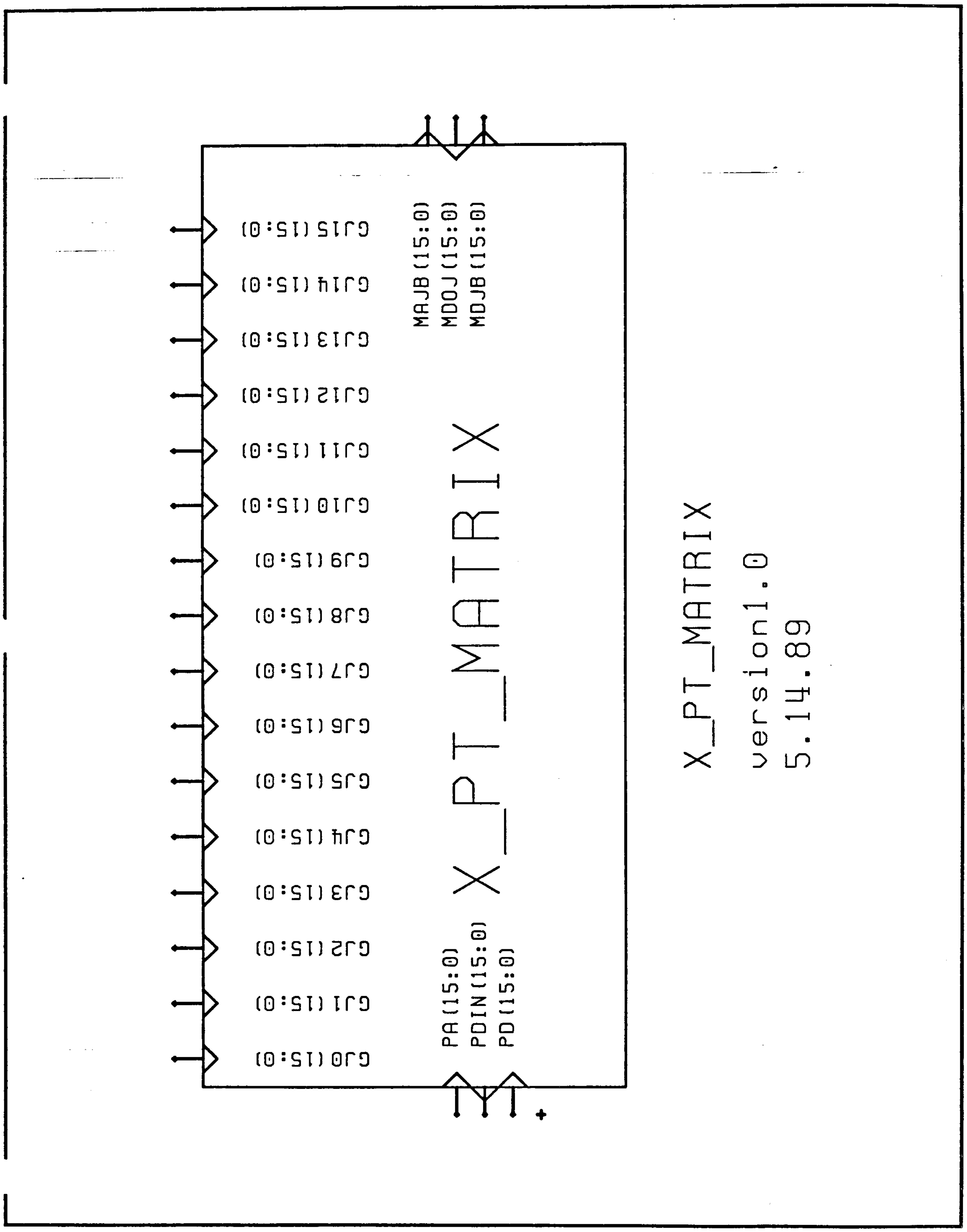


Figure 16 X Pt Matrix Symbol

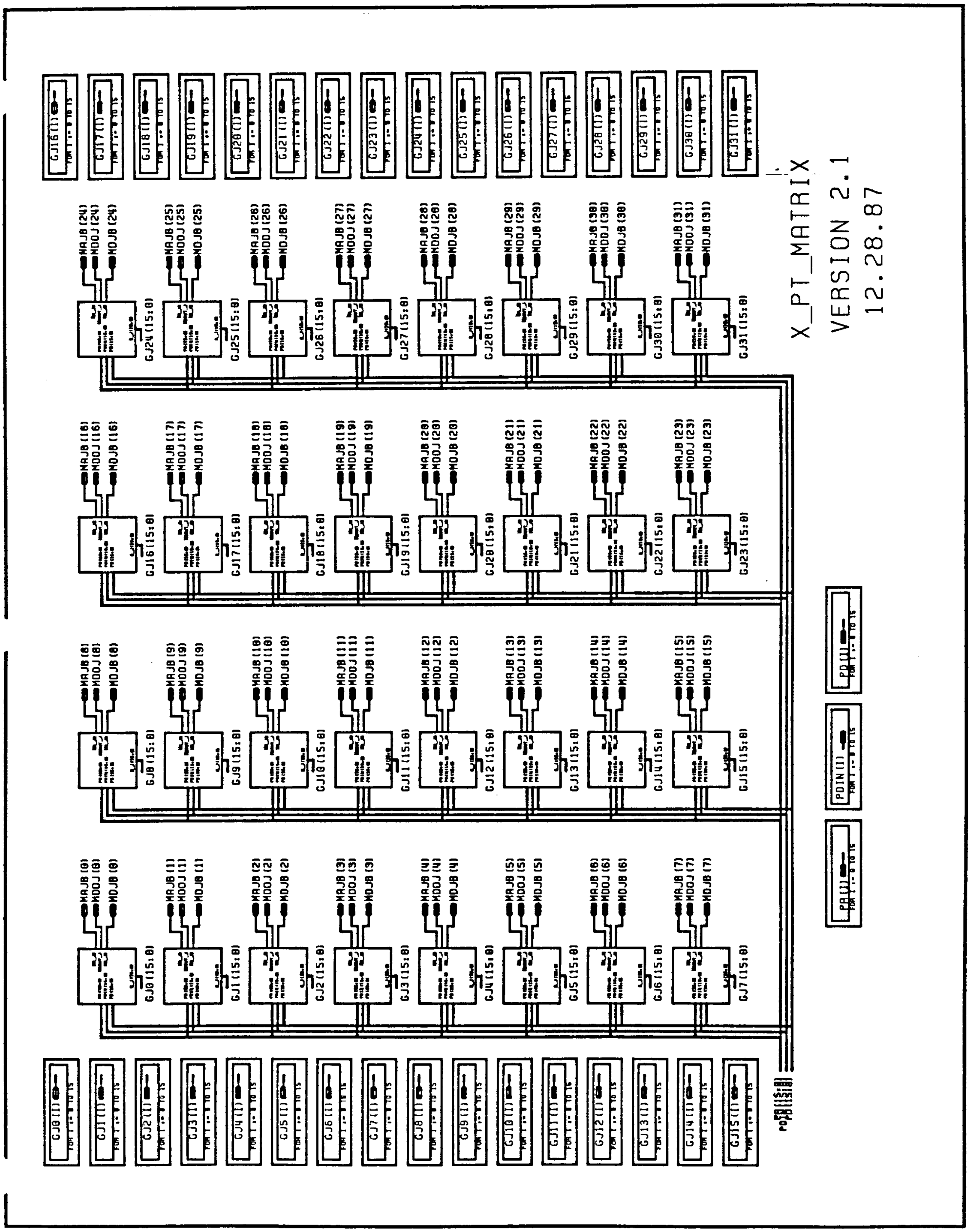


Figure 17 X Pt Matrix Schematic

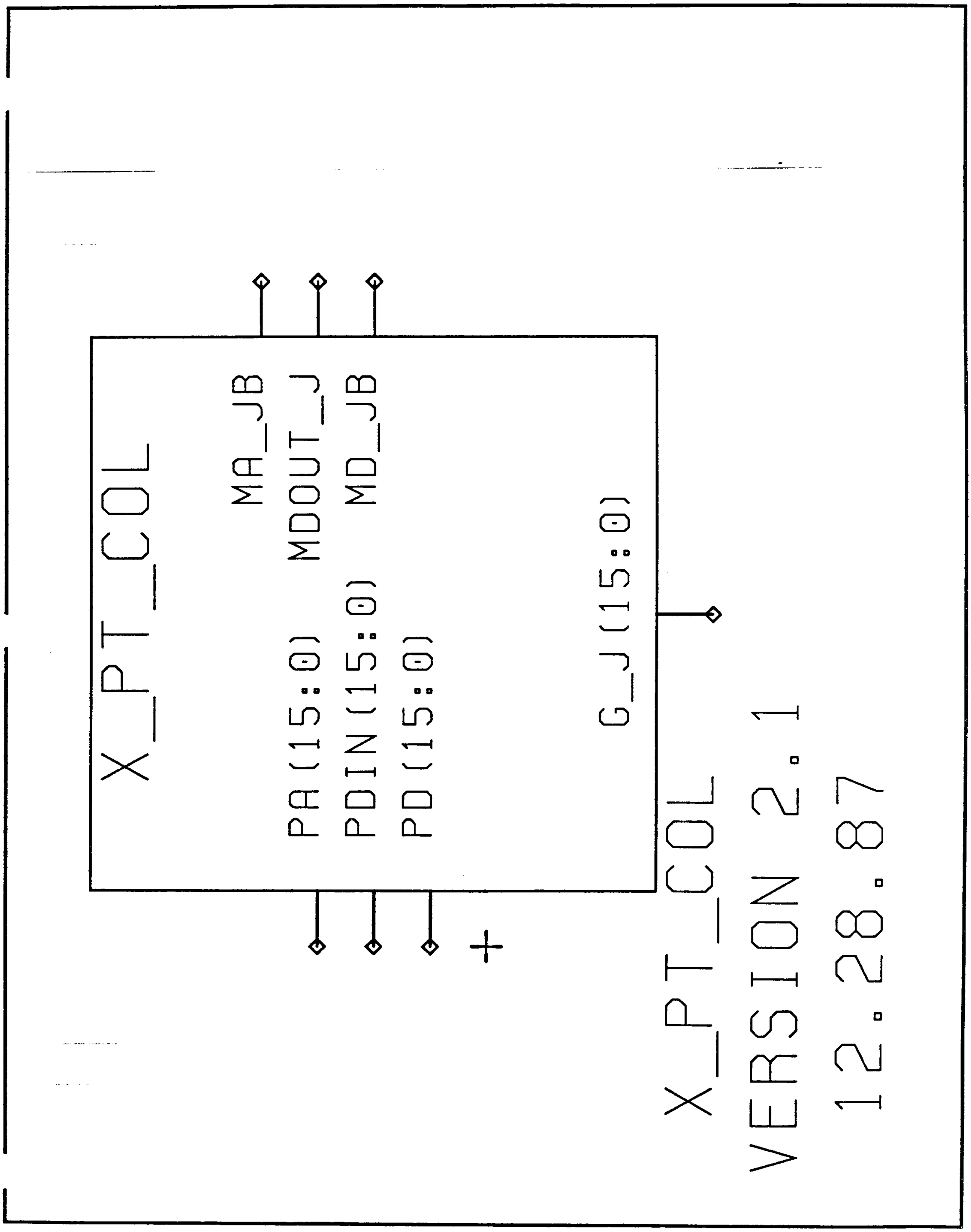


Figure 18 X_Pt_Col Symbol

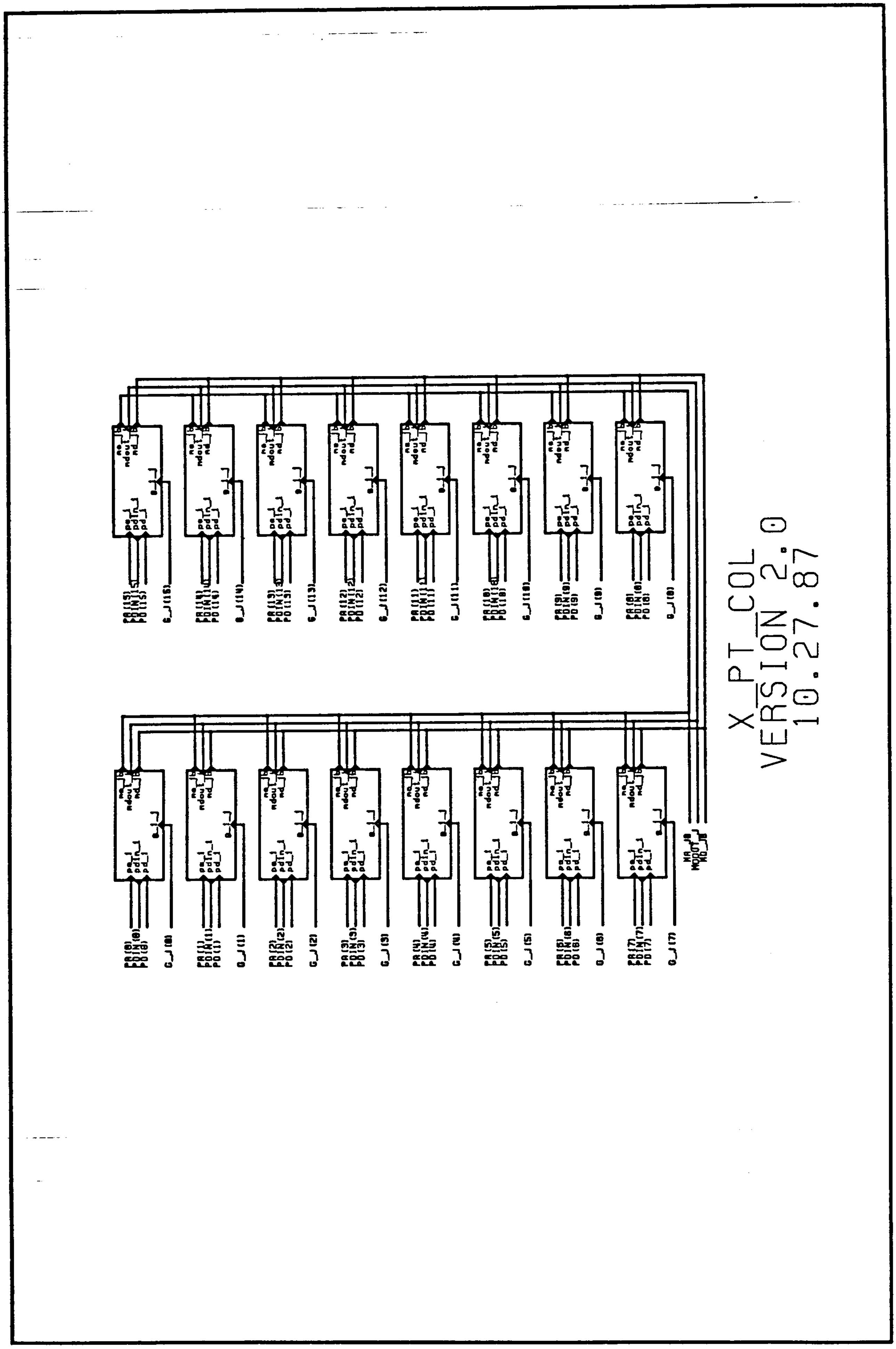


Figure 19 X_Pt_Col Schematic

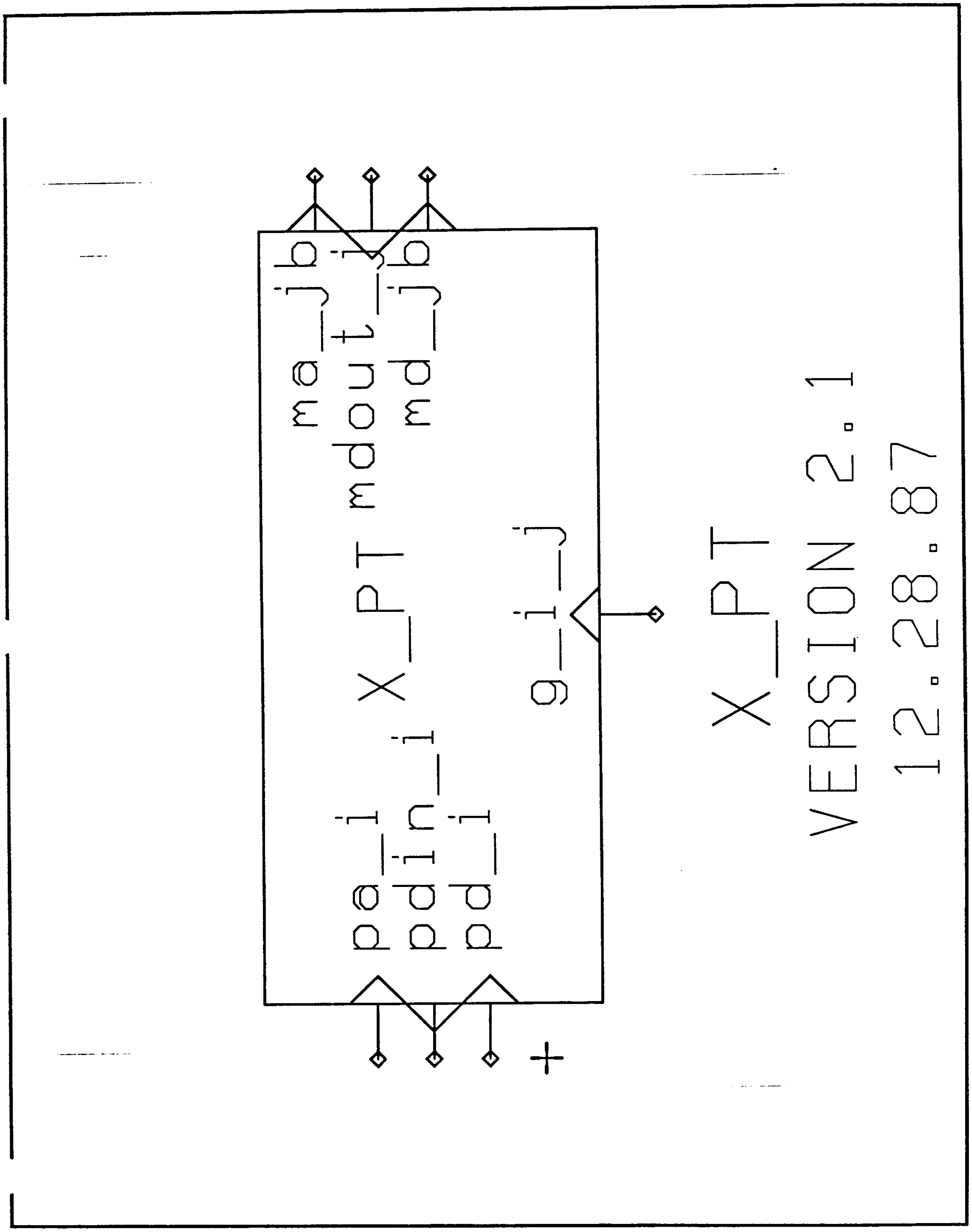


Figure 20 X_Pt Symbol

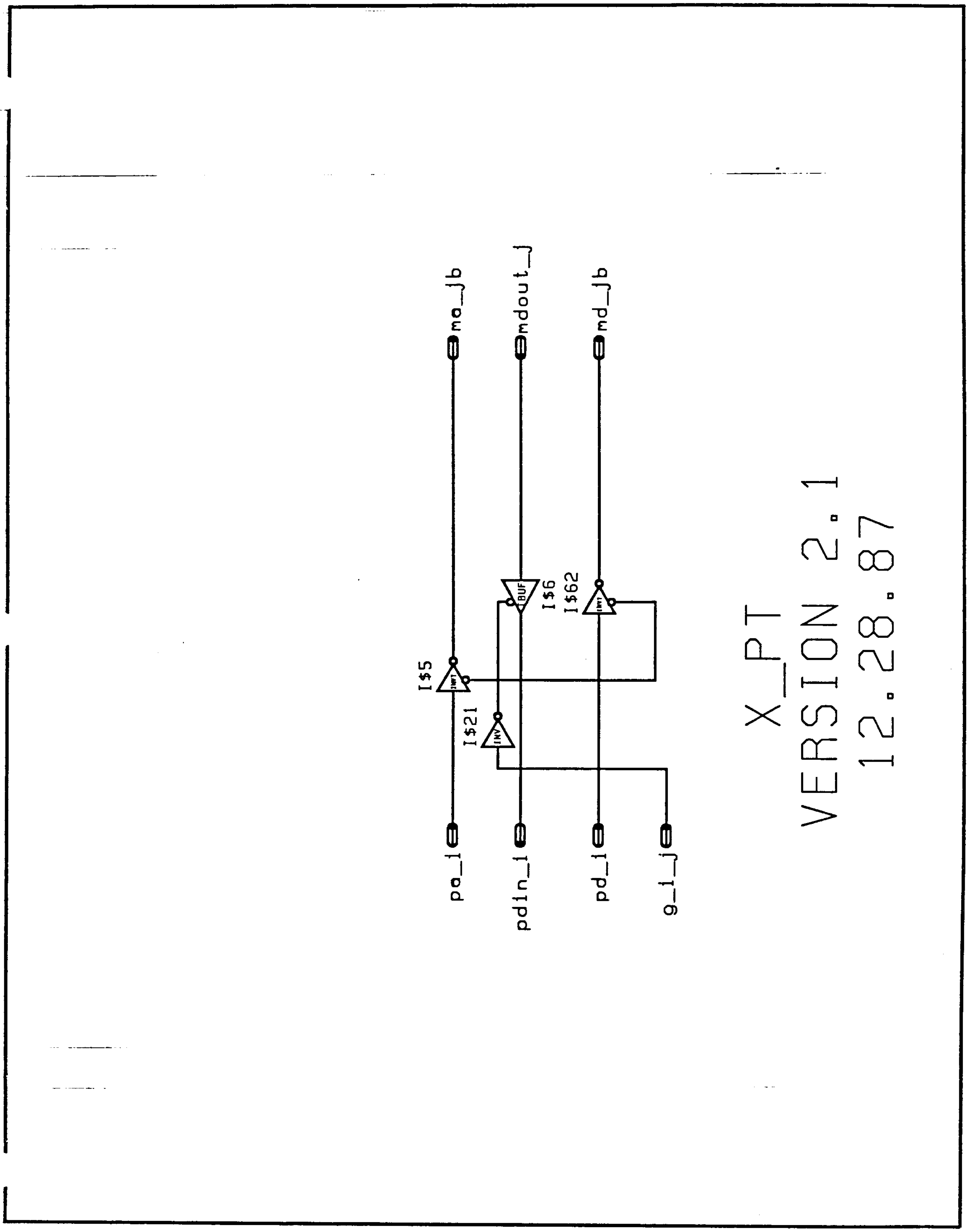


Figure 21 X_Pt Schematic

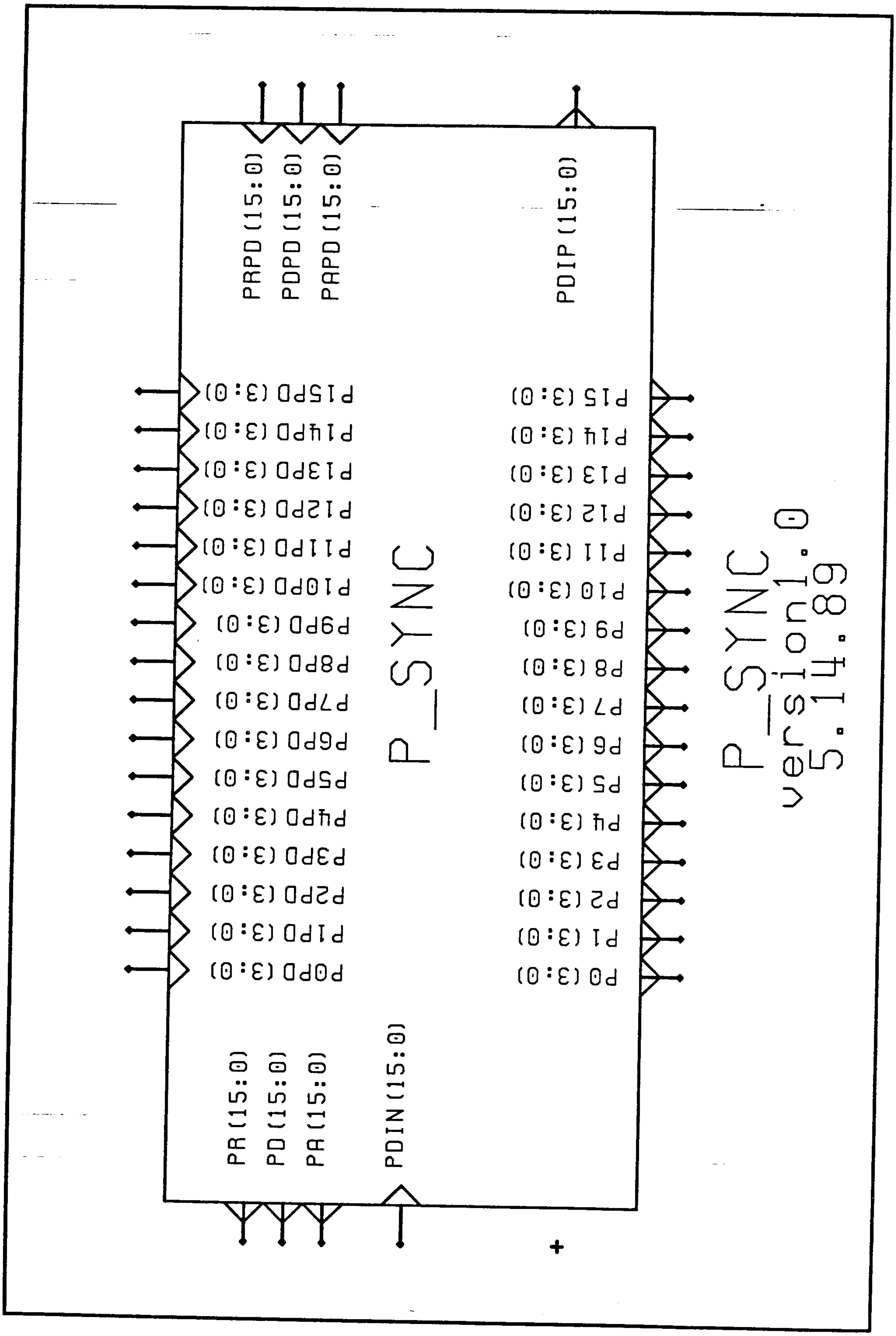


Figure 22 P Sync Symbol

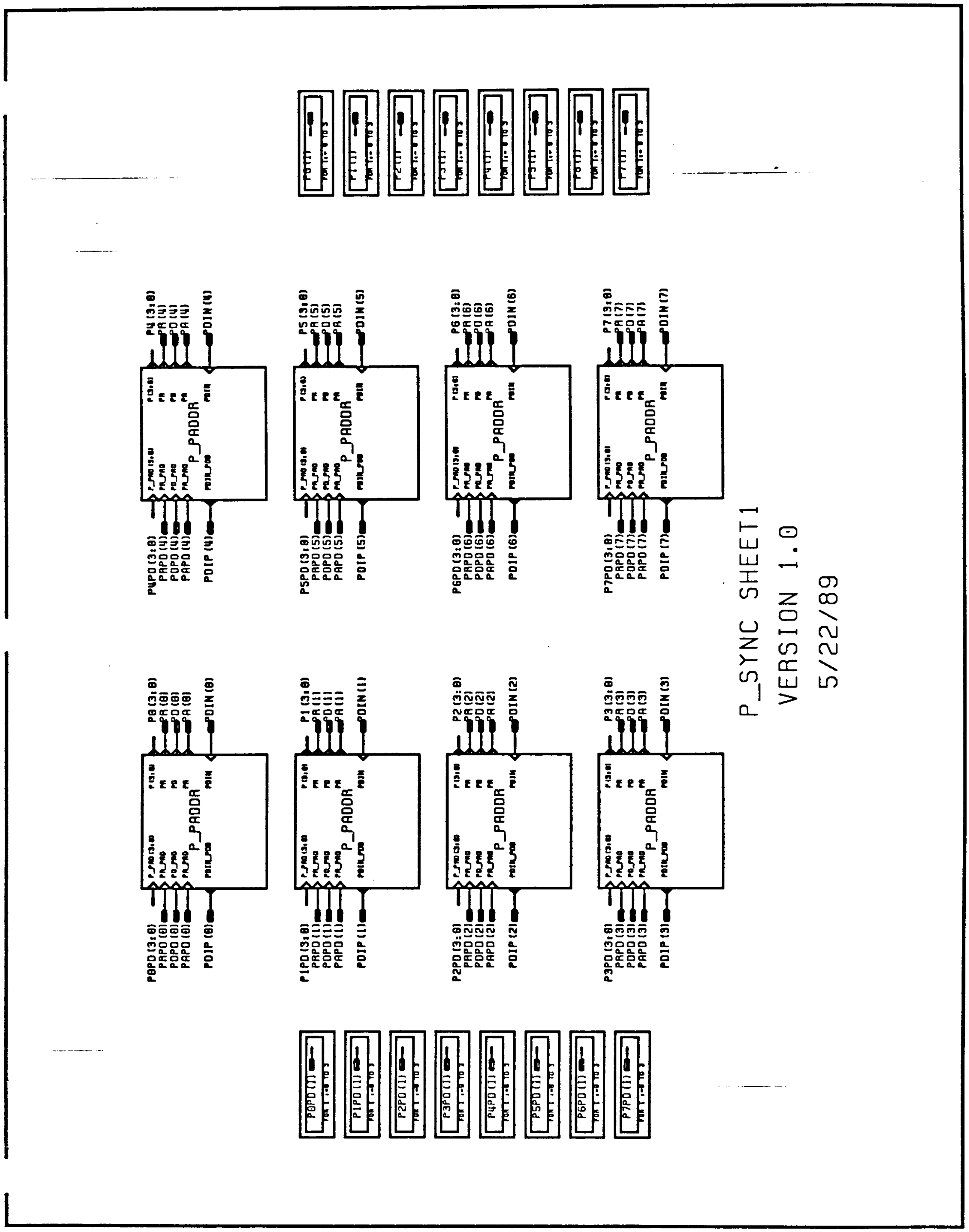


Figure 23 P Sync Schematic Sheet 1

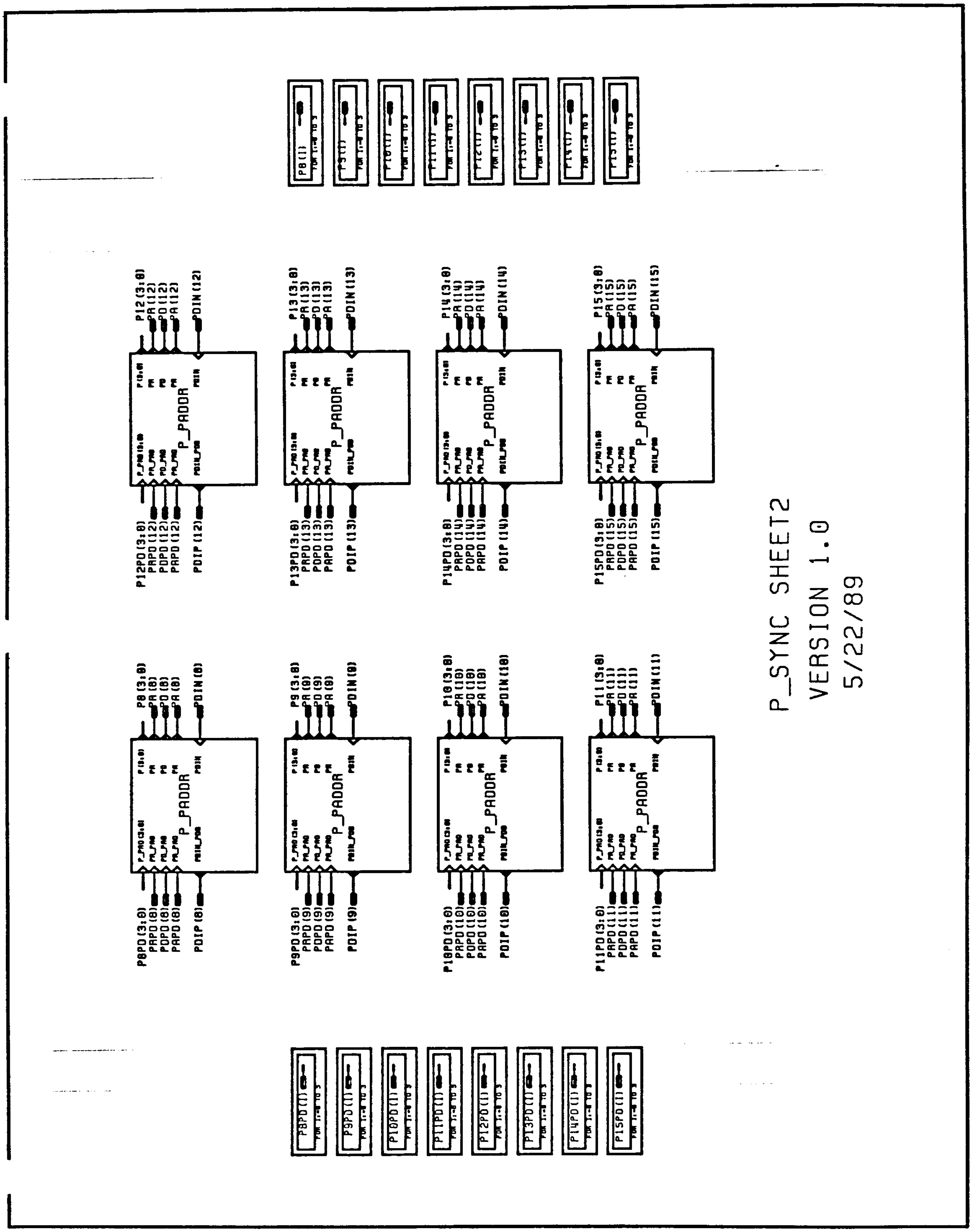


Figure 24 P Sync Schematic Sheet 2

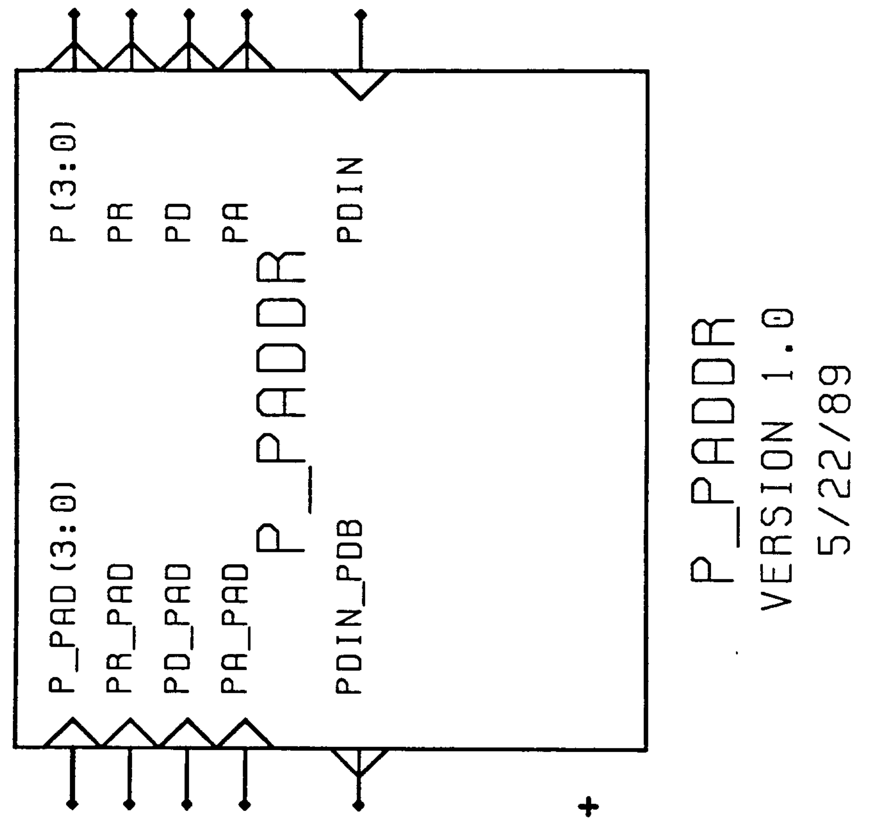


Figure 25 P_Paddr Symbol

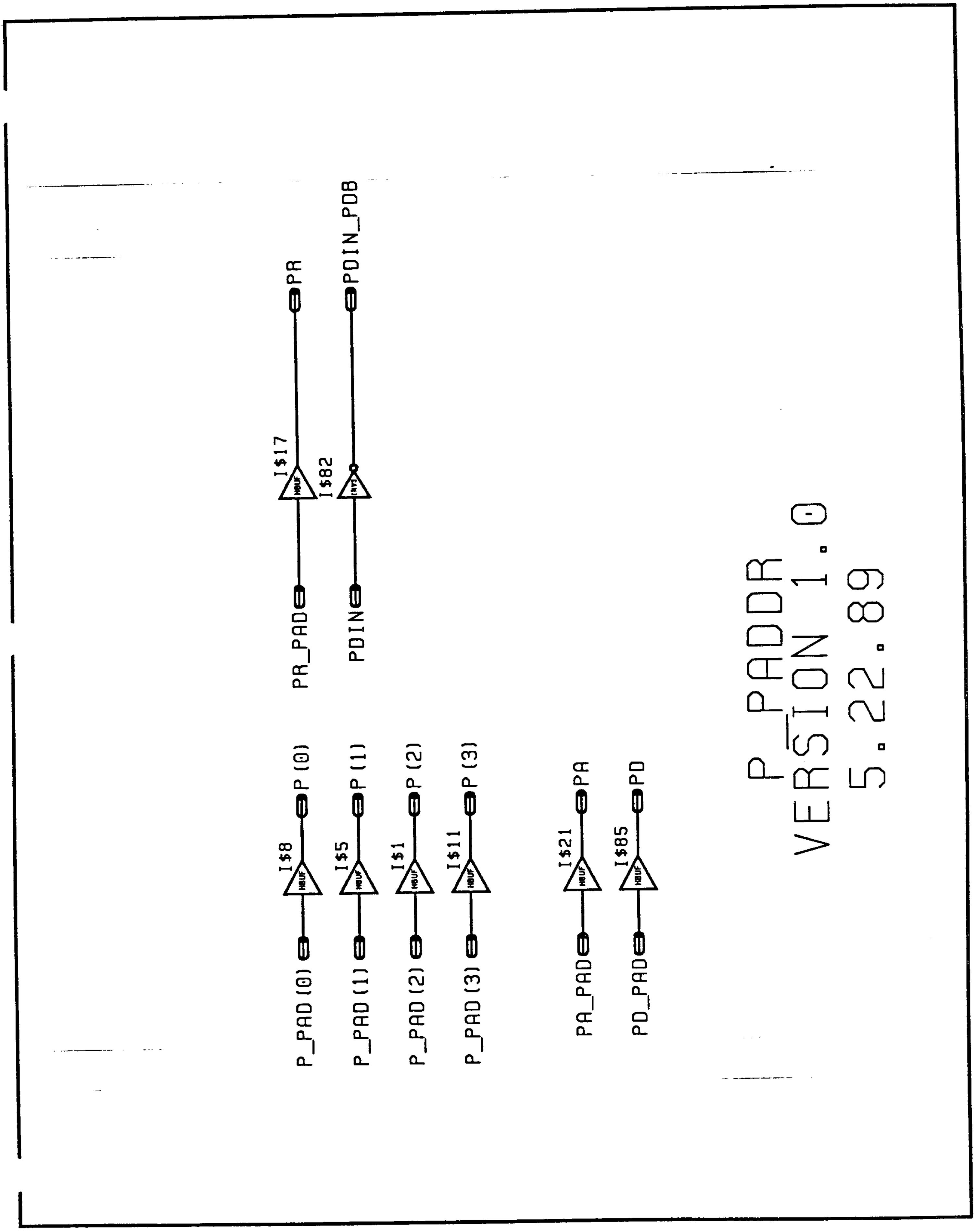


Figure 26 P_Paddr Schematic

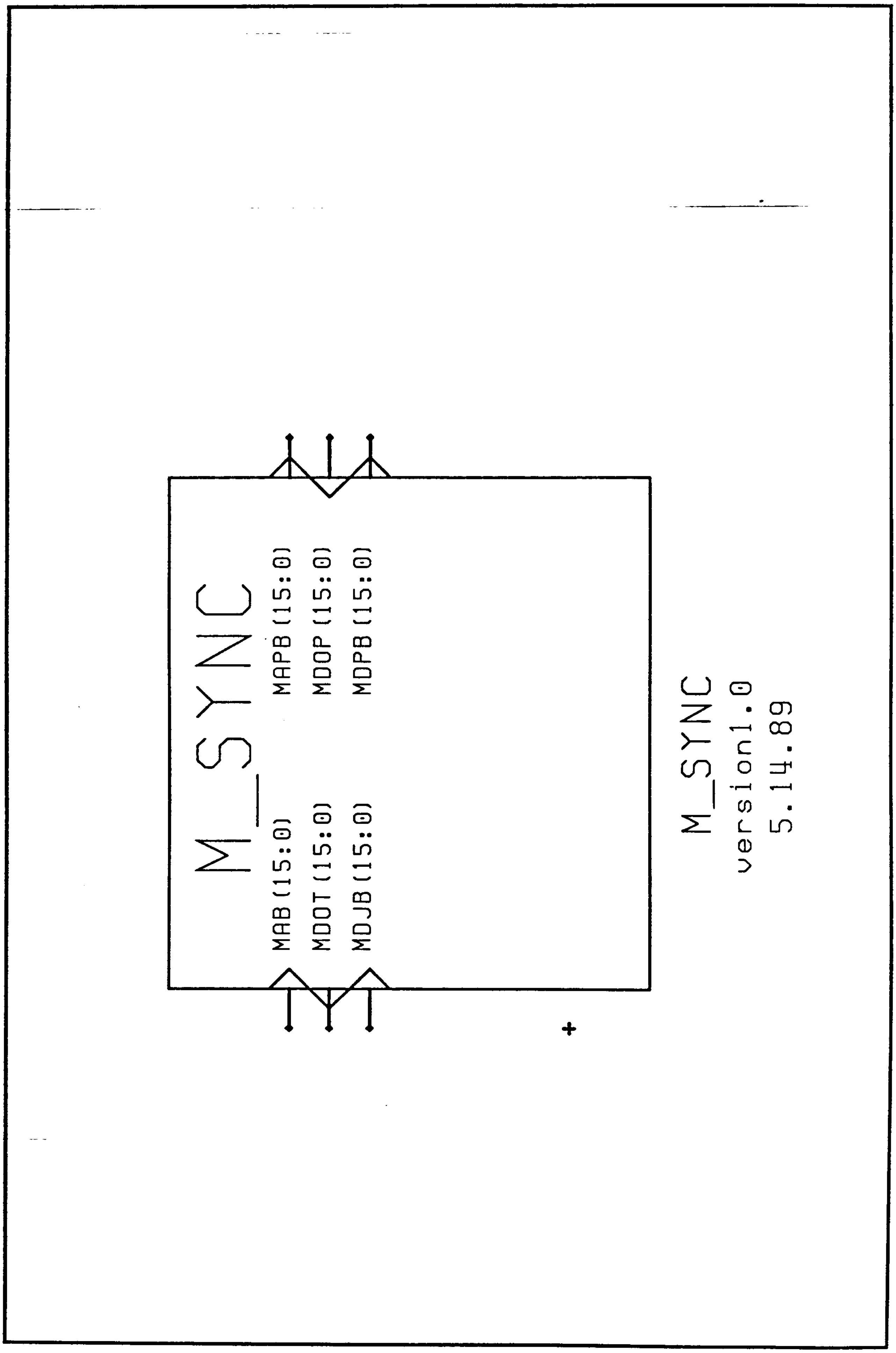


Figure 27 M_Sync Symbol

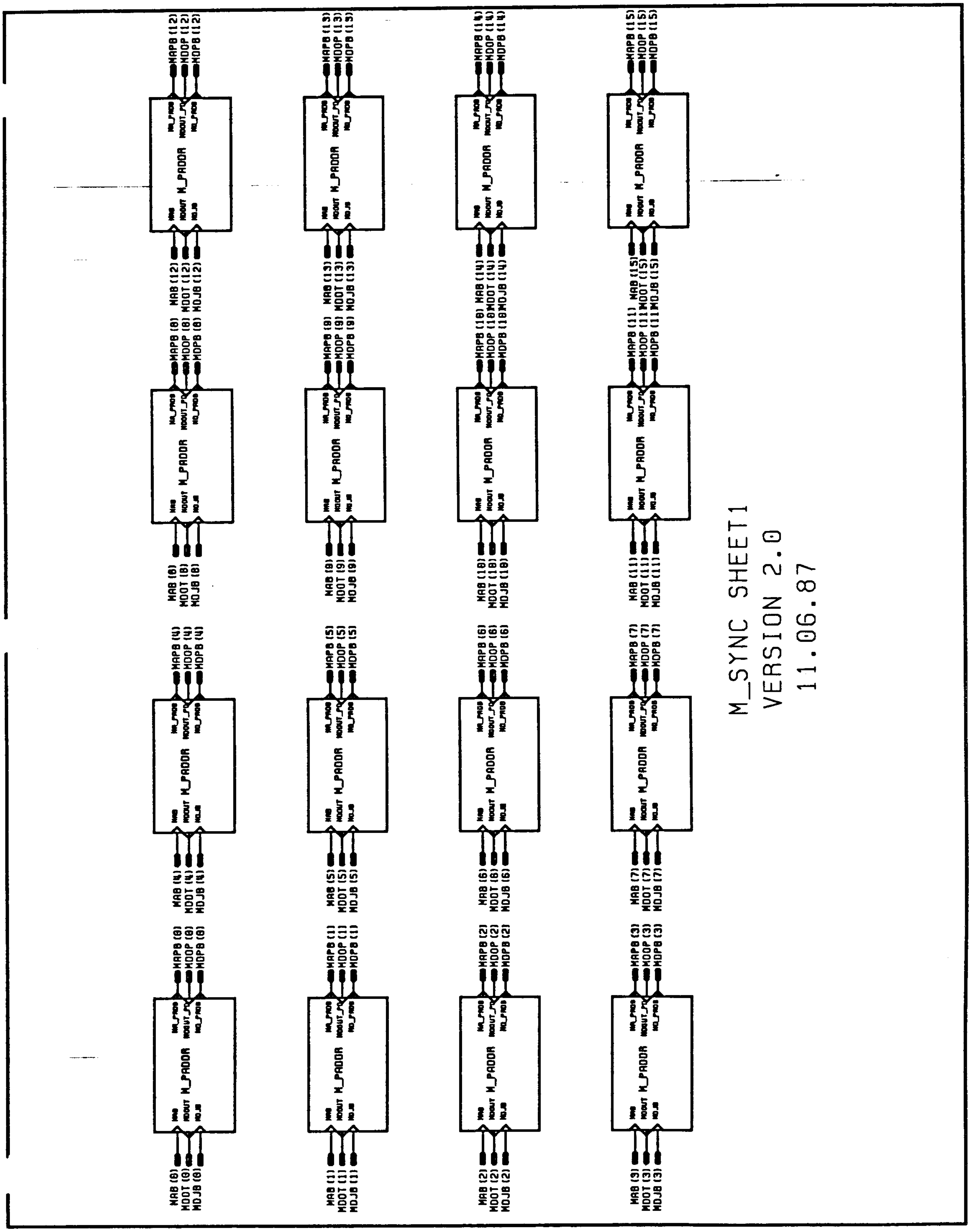
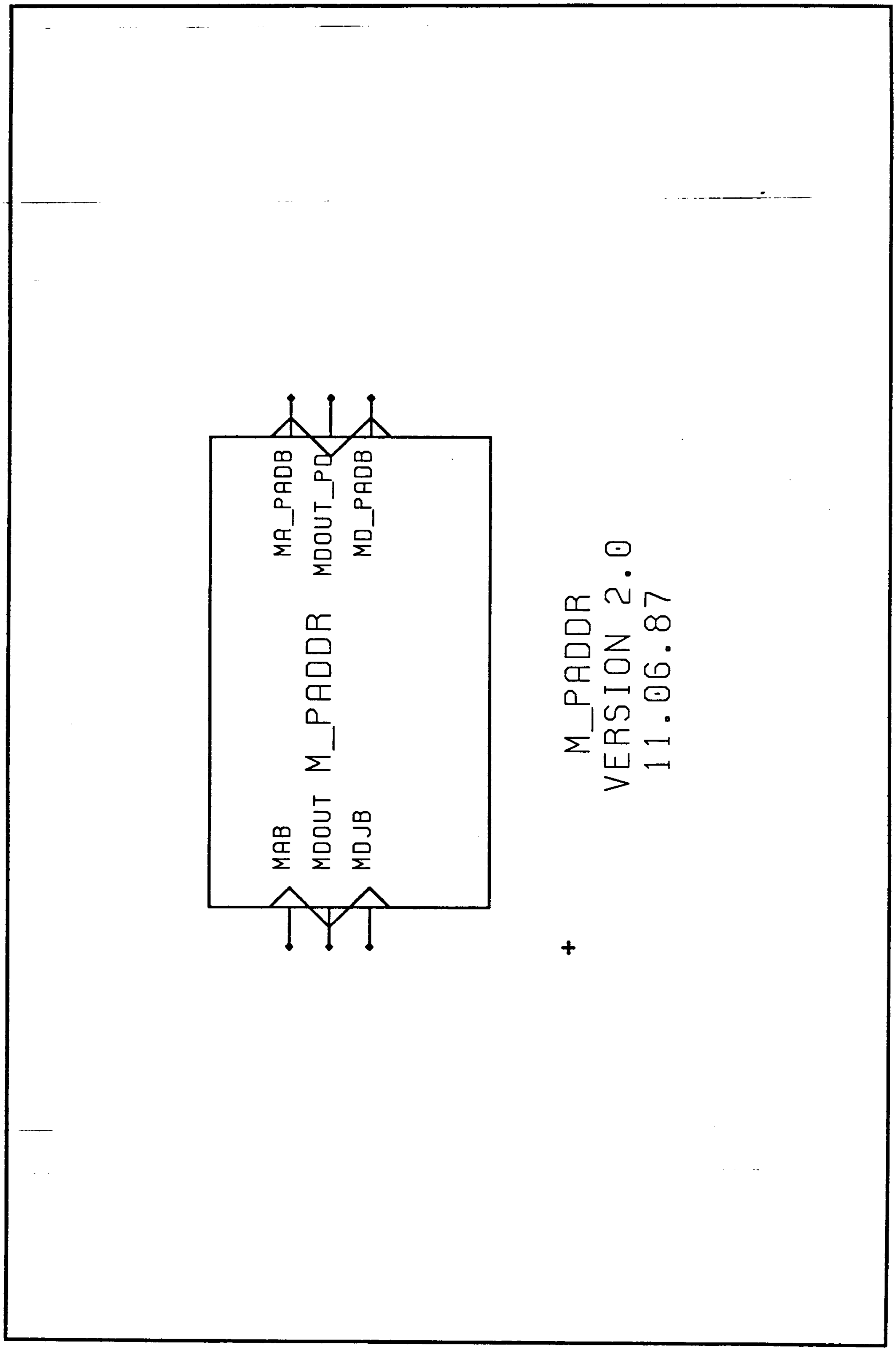

M_SYNC SHEET1
VERSION 2.0
11.06.87
M_PADDR
MAB
MDOUT
MDJB
MA_PADB
MDOUT_P
MD_PADB
MAB (0) MDOT (0) MDJB (0)
MAPB (0) MDOP (0) MDPB (0)
MAB (1) MDOT (1) MDJB (1)
MAPB (1) MDOP (1) MDPB (1)
MAB (2) MDOT (2) MDJB (2)
MAPB (2) MDOP (2) MDPB (2)
MAB (3) MDOT (3) MDJB (3)
MAPB (3) MDOP (3) MDPB (3)
MAB (4) MDOT (4) MDJB (4)
MAPB (4) MDOP (4) MDPB (4)
MAB (5) MDOT (5) MDJB (5)
MAPB (5) MDOP (5) MDPB (5)
MAB (6) MDOT (6) MDJB (6)
MAPB (6) MDOP (6) MDPB (6)
MAB (7) MDOT (7) MDJB (7)
MAPB (7) MDOP (7) MDPB (7)
MAB (8) MDOT (8) MDJB (8)
MAPB (8) MDOP (8) MDPB (8)
MAB (9) MDOT (9) MDJB (9)
MAPB (9) MDOP (9) MDPB (9)
MAB (10) MDOT (10) MDJB (10)
MAPB (10) MDOP (10) MDPB (10)
MAB (11) MDOT (11) MDJB (11)
MAPB (11) MDOP (11) MDPB (11)
MAB (12) MDOT (12) MDJB (12)
MAPB (12) MDOP (12) MDPB (12)
MAB (13) MDOT (13) MDJB (13)
MAPB (13) MDOP (13) MDPB (13)
MAB (14) MDOT (14) MDJB (14)
MAPB (14) MDOP (14) MDPB (14)
MAB (15) MDOT (15) MDJB (15)
MAPB (15) MDOP (15) MDPB (15)

Figure 28 M_Sync Schematic



Fi re 29 M_Paddr Symbol

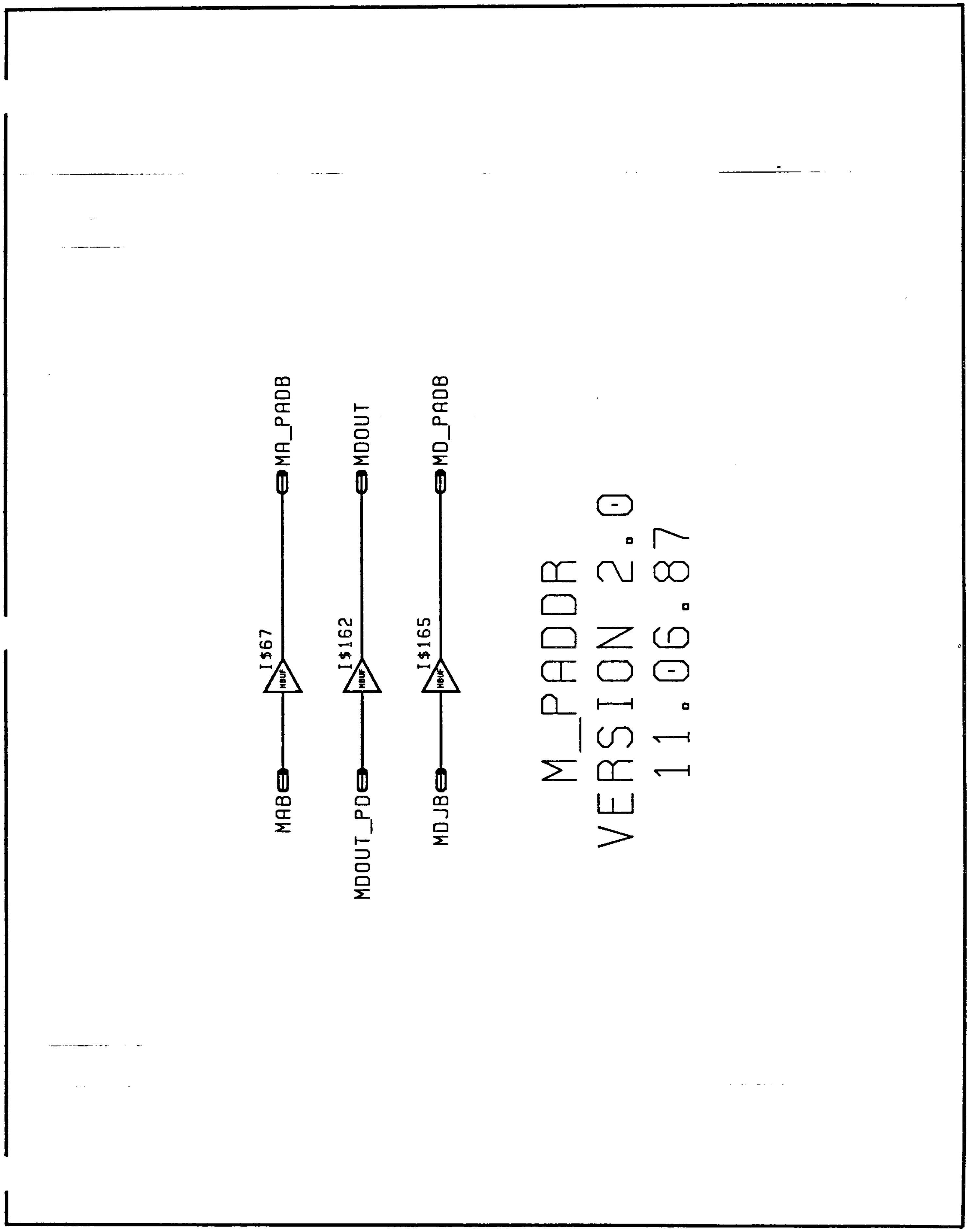


Figure 30 M_Paddr Schematic

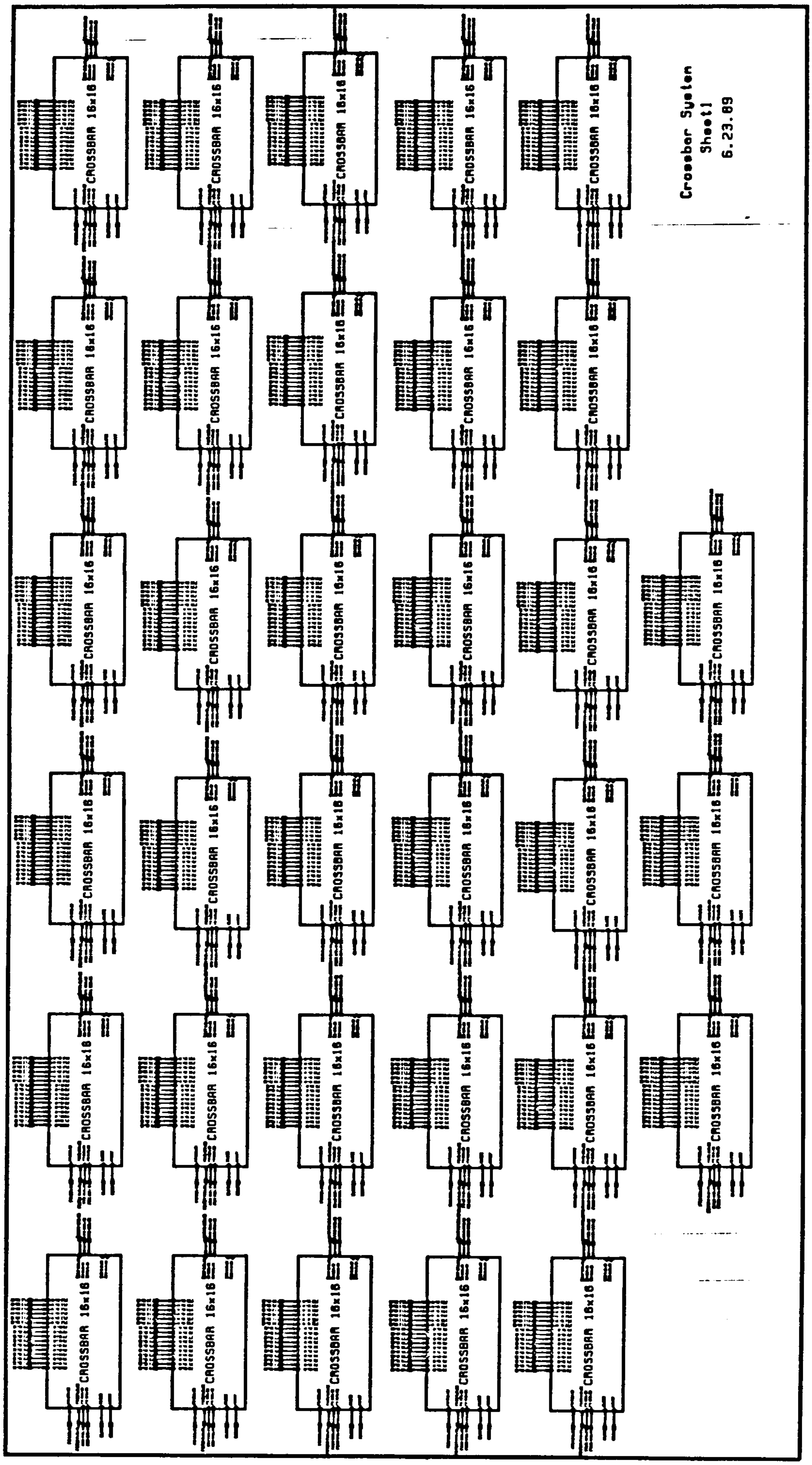
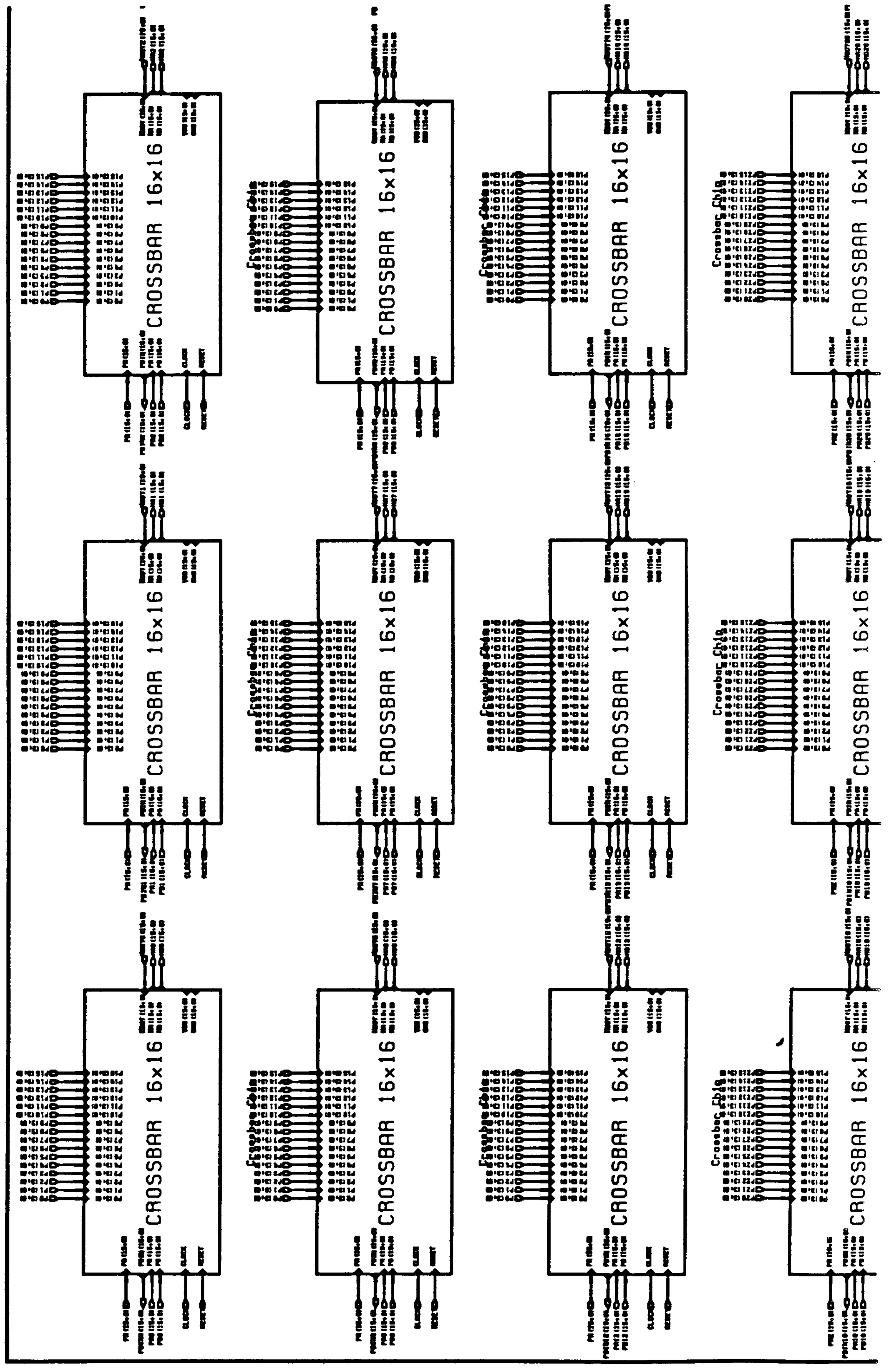


Figure 31 Crossbar System Sheet 1


CROSSBAR 16x16
CROSSBAR 16x16
CROSSBAR 16x16
CROSSBAR 16x16
CROSSBAR 16x16
CROSSBAR 16x16
CROSSBAR 16x16
CROSSBAR 16x16
CROSSBAR 16x16
CROSSBAR 16x16
CROSSBAR 16x16
CROSSBAR 16x16
Crossbar Chip
CLOCK
RESET

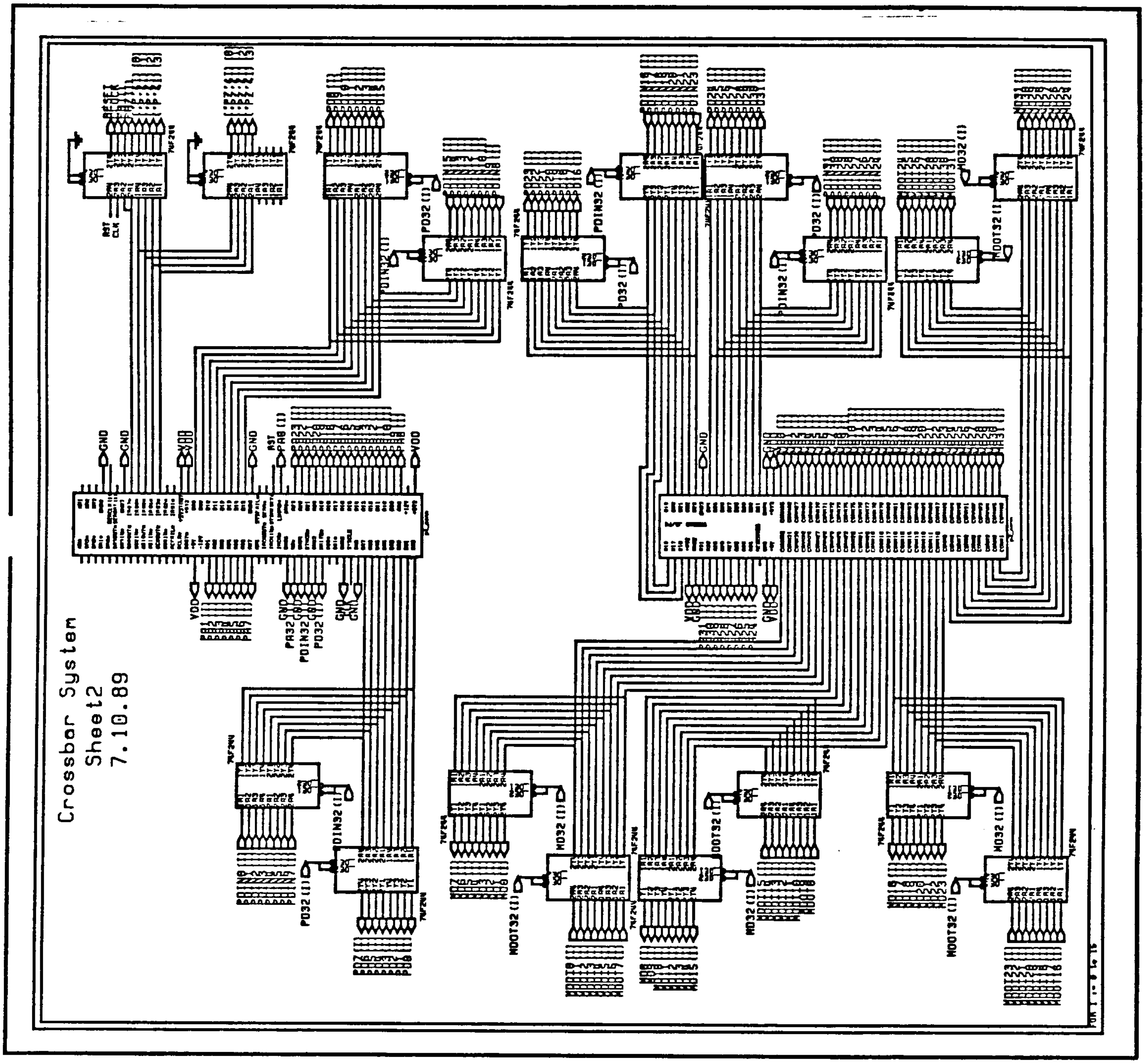
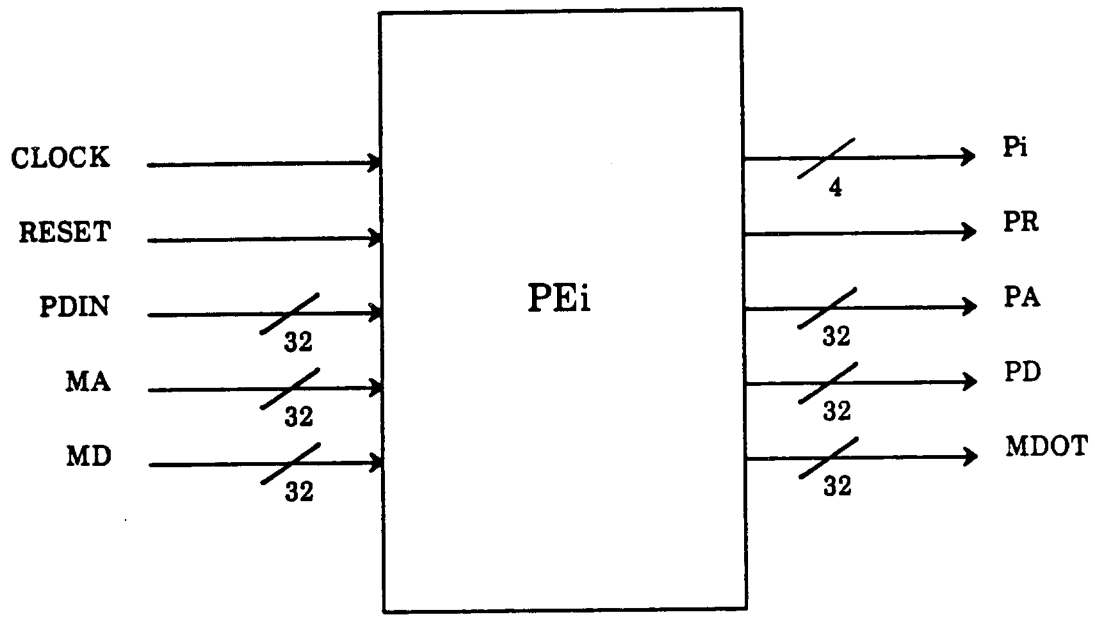


Figure 32 Crossbar System Sheet 2



Processing Element i

Figure 33 PE Symbol

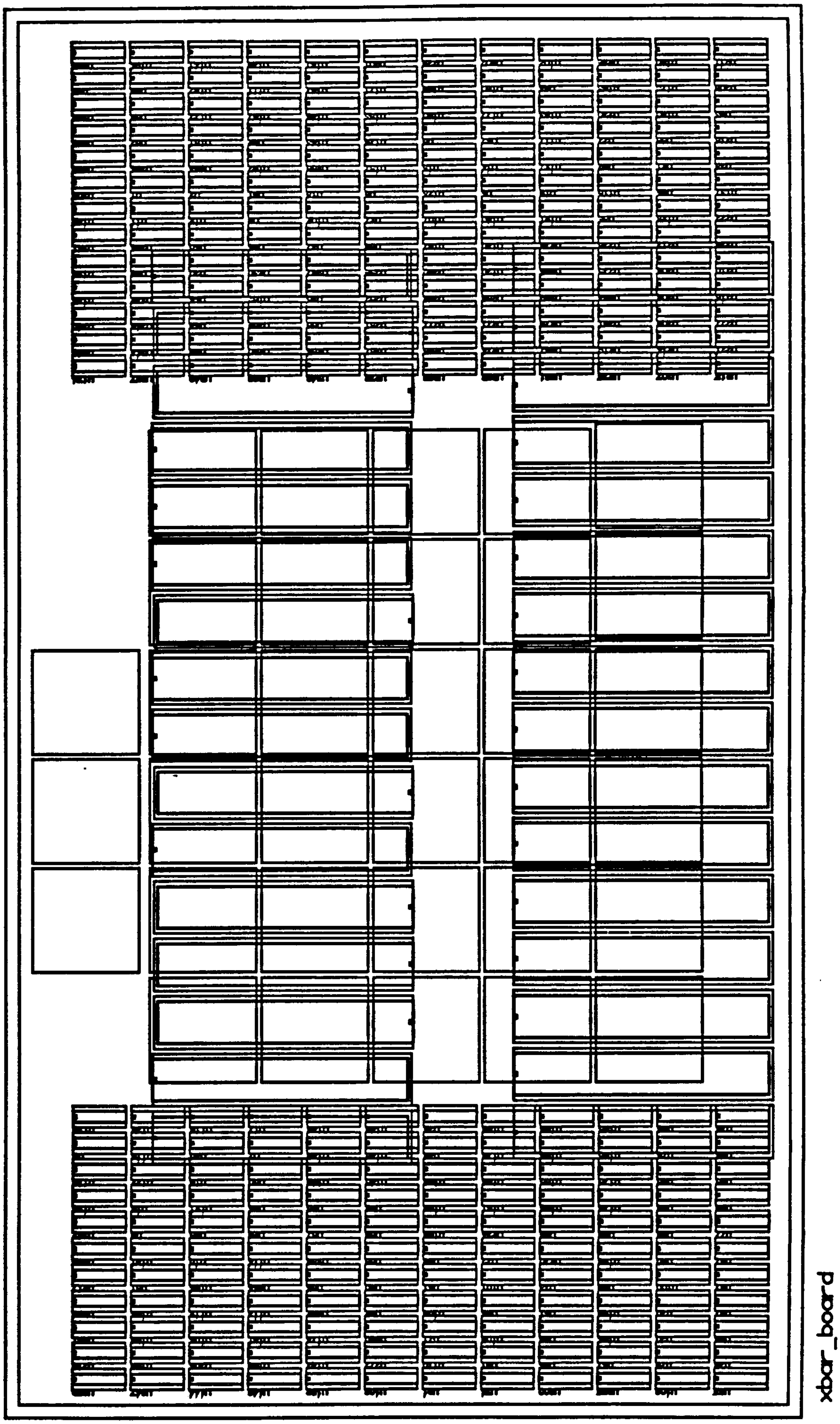


Figure 34 Crossbar System PCB