THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY[1]

VLSI CROSSBAR DESIGN VERSION TWO

Scott McFarling, Jerry Turney and Trevor Mudge

CRL-TR-8-82

FEBRUARY 1982

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

[1]This work was supported in part by NASA grant No. MCS-8009315. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agencies.

This report is a reissue of the Systems Engineering Laboratory report SEL-TR-165.

ABSTRACT

This report describes the design of a crossbar switch. A crossbar switch can be used to connect multiple processors to multiple memory modules, or simply as a means of constructing a multiport memory. True concurrent memory accesses are then possible for multiprocessing and DMA. However, memory conflicts can still occur if more than one access is made to the same memory module. In crossbar terms this corresponds to more than one input requesting the same output. In any application in which it is known that input requests can conflict in this way (e.g. if the crossbar is used in an MIMD system) logic circuitry must be provided to resolve possible conflicts. This logic can become very complex. In the design presented here this problem is simplified by using a technique which we have termed "free for all": the condition where more than one input requests an output is detected and the inputs have to try again at a later date. This requires minimal logic. The retry policy can be determined by whatever is using the inputs. Fabrication of a prototype is presently being undertaken jointly by the ECE department at the University of Michigan and General Motors Technical Center.

1. INTRODUCTION

In the design of single instruction multiple data (SIMD) and multiple instruction multiple data (MIMD) architectures, a key component is a fast multiport memory. The "front end" of this memory is generally some type of interconnection network.

Various limited interconnection networks have been proposed, however, the most flexible of these contenders is the crossbar switch in which every input port can connect to every output port (provided no more than one input requires connection to each output port).

The major drawback to the use of the crossbar in conventional design has been the amount of discrete logic needed for its construction. The crossbar, however, is a highly regular structure and represents a prime candidate for VLSI implementation.

It would be impossible to build even a modest sized crossbar switch entirely on one chip unless the memories and processors it interconnected were there also because of the pin count. The best approach is to build an $m \times n$ crossbar in which one bit from each of m processor ports is routed to one of n memory ports on a single plane (implemented on one chip) and then these bit planes are "stacked" to form the desired bus width. This approach is illustrated in Fig. 1. The connecting boxes in Fig. 1 are called crosspoints. Bit plane stacking allows the user to determine the widths of his address, data, and control buses appropriate for his application.

Any type of prioritization of the inputs to the crossbar will cost in terms of speed. One simple priority scheme is to daisy chain a request strobe from processor input to processor input (See Fig 2.) giving priority to the the processor physically connected highest on the daisy chain. However, a daisy chain has two disadvantages. First, a daisy chain as in Fig. 2 has a fixed prioritization. There is no flexibility to change priorities. Second, a daisy chain of pass transistors for m inputs takes up a considerable amount of time and the crossbar clock would have to be slowed down to wait for the worst case strobe acknowledge which would be for the last processor line in the chain.

Assuming that one processor can "lock-out" other processors once it has the crosspoint connection, actual requests conflicts are rare, only occurring when two processors request the same bus during the same clock cycle. One can take advantage of the infrequency of request conflicts in the following manner. Allow each processor (unless it is specifically locked out from a line by a processor which already owns the bus) to grab the line of its choice. This is a "free for all" policy. (See Fig. 3 for a sketch of our 4 by 16 implementation of such a crossbar.) This is done in a asynchronous manner. (Of course the action is synchronized to the clock, but there is no specific time slot in which all processors must request a line.) After gaining access to the line, the processor transmits data as if it owns the line. It, however, monitors the line to see if what it transmits is what it sent over the line. If, at anytime, the line fails to agree with its output, the processor aborts its transmission, drops the line and rerequests the line. The crossbar does an internal comparison of the processor line to the connected memory line. The crossbar signals the processor its failure to own the line through a *collision* line. This approach will minimize the amount of hardware needed for prioritization but more importantly the time needed for prioritization. It is based on the approach used in contention networks such as Ethernet.

2. FREE FOR ALL CROSSBAR

The following is a discussion of a trial implementation of the "free for all" crossbar.

2.1. Crossbar pinout

Fig. 4 presents a pinout for the crossbar design. The values $m=4$ and $n=16$ have been chosen for the number of processor ports and memory ports respectively, in order to meet the pinout restriction of 40 pins. Each input port from the processors has 5 pins, 4 of these multiplexed between address and data-control information. There are 4 processors ports, hence a total of 20 pins are used for processor ports. There are 16 bidirectional memory output ports. With power, ground, and clock, *39 pins* are needed for this designed. The extra pin was used as additional input for the clock. A description of the pins follows:

2.1.1. A0-3 (Addresses 0-3). Inputs when A/D high. The address of one of the 16 memory ports desired by the processor. In order to distinguish this address from the address that may be passed to memory through the crossbar, (see Fig. 1) this address will be referred to as the "routing" address. This mode is selected when the A/D line is high.

2.1.2. A/D (Address/Data-control). Input. Multiplexes the routing address into the chip when high. Latches the routing address and requests the memory port specified by the routing address when pulled low. By maintaining A/D high the processor does not request or lock any memory port. This keeps processors which are not using the crossbar from locking out access to a memory port by "sitting" on the memory line.

2.1.3. DO (Data out) and DI (Data in). DI is an input and DO is an output. The data from and to the processor respectively. The DO mode is selected when A/D is low and R/W is high. The DI mode is selected when A/D is low and R/W is low.

2.1.4. CLSN (Collision). Output. If a collision occurs between two processors, i.e., if two or more processors request the same memory port on the same cycle, this line goes low. May be connected to an interrupt line for the processor to allow both processors to pull back until the next cycle.

2.1.5. R/W (Read/Write). Input. Direction of data flow from processor. Line high for a read from memory and low for a write to memory. During a read whereas when A/D is low, PHI is high and R/W is high it turns the selected memory ports into inputs into the chip and turns the DI,DO line in an output when

2.1.6. LCKD (Locked out). Output. Indicates to the processor that the memory line it has requested has been locked by another processor, and hence the requested connection has not been made.

2.1.7. M0-15 (Memory ports 0-15). Bidirectional. Memory ports used as inputs to the chip when PHI high, the port has been selected by one or more processors and the R/W line is high.

2.1.8. PHI. Input. PHI in tandem with A/D play the role of the normal two phase clock PHI1 and PHI2, except that A/D need not be pulled low during the PHI high cycle.

2.2. Details of operation

The basic path of the crossbar consists of the select, the crosspoint cell and the memory buffer. These major blocks are detailed below.

2.2.1. Select Logic. Fig. 5 indicates in more detail the assignment of memory pads. The first, second, and fourth pad from the top are bidirectional pads (with lightening arrestors incorporated). The top two pads function as outputs only when PHI is high and when A/D is low. This prevents the pads from becoming outputs when the processor is not using the crossbar and has maintained its A/D line high. It also allows the address drivers a chance to float before the outputs from the crossbar are active, since PHI and A/D are non-overlapping.

The fourth pad is turned into an output only if the write bar line is high. The write bar is automatically held low until A/D goes low and PHI goes high. Then when these conditions are met, it reflects the state of the R/W line. This prevents the state of the R/W pad which doubles as the A1 pad from changing the direction of the DO,DI pad unless an address has been selected and the PHI clock is high.

Fig. 6 displays the timing diagrams of the write cycle. A/D is brought low. Addresses A0-3 are latched. Output addresses from the selector SA0-3 are forced high when A/D is high in order to keep the processor from controlling a memory line when the processor is not using the crossbar as mentioned above. When A/D goes low these select addresses are sent out to the cell in double rail logic (SA0-3, SA0-3 bar) to select the crosspoint cell of interest. Data input lines are held high and the write bar lines are held low until A/D is low and PHI is high to serve two purposes. One purpose is to speed up the total write operation, since its timing is critical. Every read from memory through one set of bit planes must be preceded by a write operation on a separate set of bit planes (See Fig. 1). Hence the timing of the write is critical. Here we start with the crossbar already in the write state. The memory ports are also prevented from turning inward until PHI goes high. The second purpose is to disable the tristate to the DO pad while either A/D is high or PHI is low. Fig. 8 illustrates the select logic. Note that PHI high also maintains the addresses once A/D goes low.

Also included in Fig. 8 is the logic for collision detection. Since the DI lines are wired OR-ed as seen in Fig. 9 later, the only collision that can be detected is when DI is 1, DO is 0, and R/W is low (and, of course, when A/D is low since CLSN is only sampled when A/D is in the data-control mode). If DI is 0, DO is guaranteed to be 0 whether there is a collision or not because of the wire ORing, and in addition the only time one can detect collisions is during a write since it makes no sense to compare what is arbitrarily on the DI line with what is being read on the DO line.

ID bit planes are included in Fig. 1 to prevent the following trouble. If two processor write to the same memory port, the odds are they will disagree. To prevent a memory location from being trashed, the ID planes can be check for valid ID's. The ID planes also guarantee a collision. For example if 0011 is the ID of processor 0 and 0110 is the ID of processor 1, a conflict is guaranteed and the memory can determine a collision has occurred and prevent a write since the ORed ID, 0010, corresponds to no legitimate ID.

2.2.2. The Crosspoint Cell. Examine Fig. 9. Assume the lock bar line is high. Then if a processor selects this crosspoint, the select line goes high and the S bar input to the latch is 0, while the R bar input is 1. This sets the latch and a connection is made. As long as the select stays high, the processor has the connection (or correspondingly as long as the processor keeps its A/D line low, it has a connection). With a connection establish the lock line is pulled low.

Assume the lock line is low before the processor selects this crosspoint. Then if a processor selects this crosspoint, the latch is not set and the LCKD bar line is pulled low signaling that the processor has been locked out of the crosspoint it attempted to select.

Once a connection has been established, the processor monitors the MEM line of the crosspoint he has connected, it order to check for collision during a write mode. During a read, the processor simply reads from the MEM line.

2.2.3. The Memory Buffer. Examine Fig. 10. The memory buffer provides input from the memory port when WMEM bar is high and provides output to the port when WMEM is low. WMEM is pulled low during PHI low to provide faster writes and to keep MEMI off the MEM line during PHI low again providing faster writes.

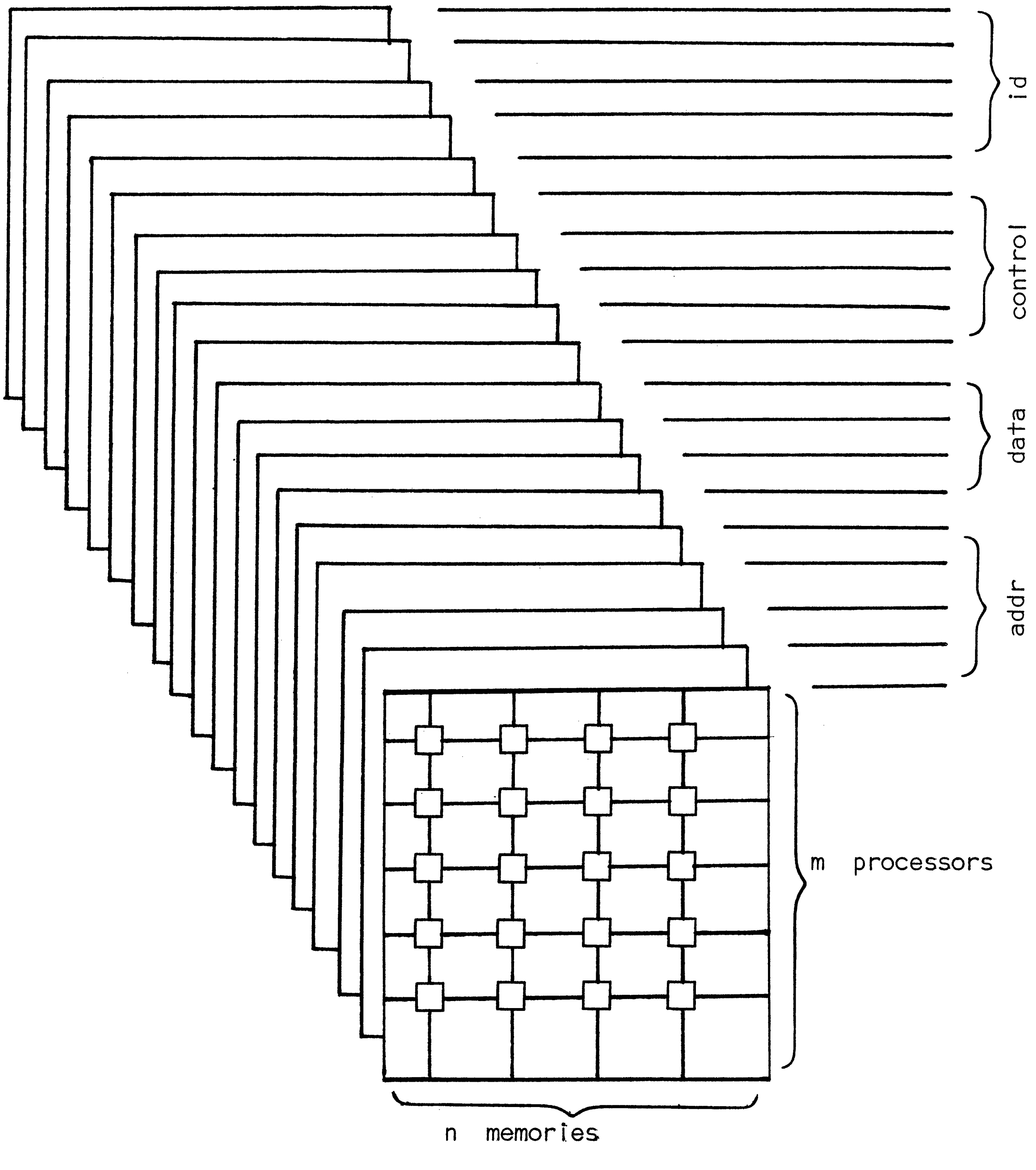
A pullup for the lock line is also provided in the block of logic.

2.3. Tests performed on the design

The entire design has been successfully design rule checked and simulated in tsim, esim and static sim.

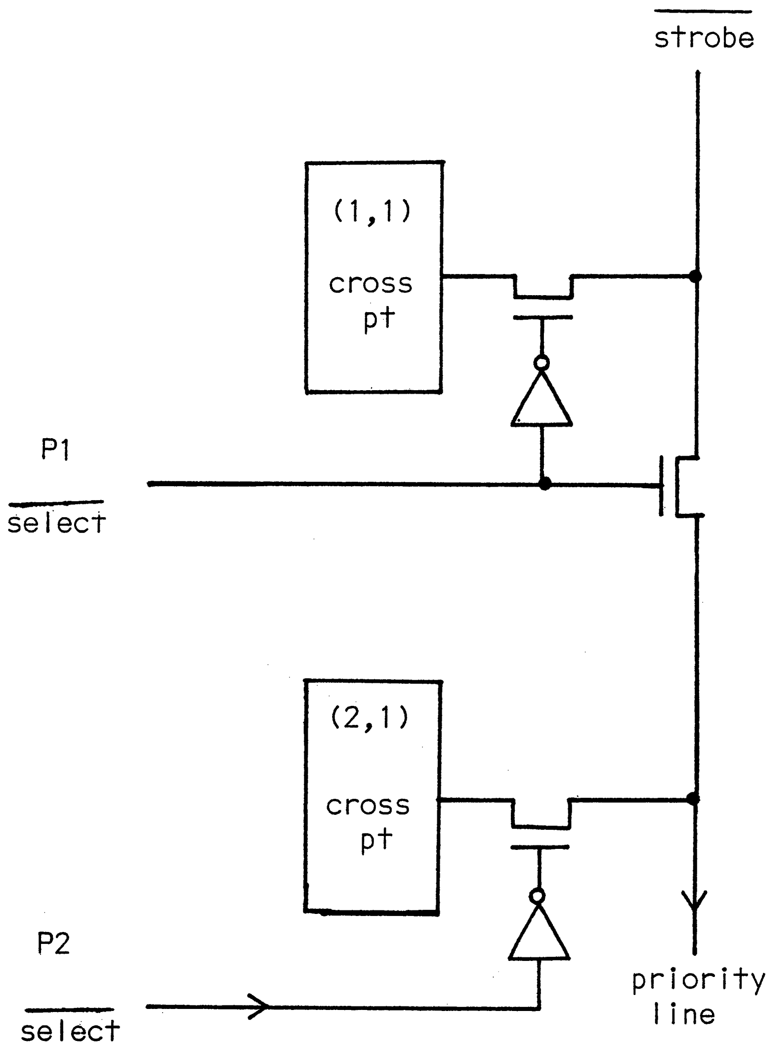
3. CONCLUSION

The principal innovation in our crossbar design is the "free for all" policy in which every processor is allowed to grab the memory line of its choice providing that it then monitor its writes in order to determine "collisions" with other processors. This has allowed us to design a crossbar switch with minimal logic which still retains flexibility of operation. In addition, the design allows multiple reads from the same address so values in memory can be broadcast. The design is clean, simple, and easily implemented. We have also, wherever possible, left the design open for double metalization. Most of our poly lines have been given spacing of 3 λ in order to easily convert them to a second layer of metal. This eventuality should allow the speeds necessary to make this design practical as a multiport memory front end. Fig. 11 shows a stiple plot of the crossbar's layout.



CROSSBAR BIT PLANES

F i g u r e 1


$\overline{\text{strobe}}$
(1,1)
cross pt
(2,1)
cross pt
P1
$\overline{\text{select}}$
P2
$\overline{\text{select}}$
priority line

prioritization

is fixed

and

for large number

of processors there is

excessive delay

$\downarrow$

Figure 2

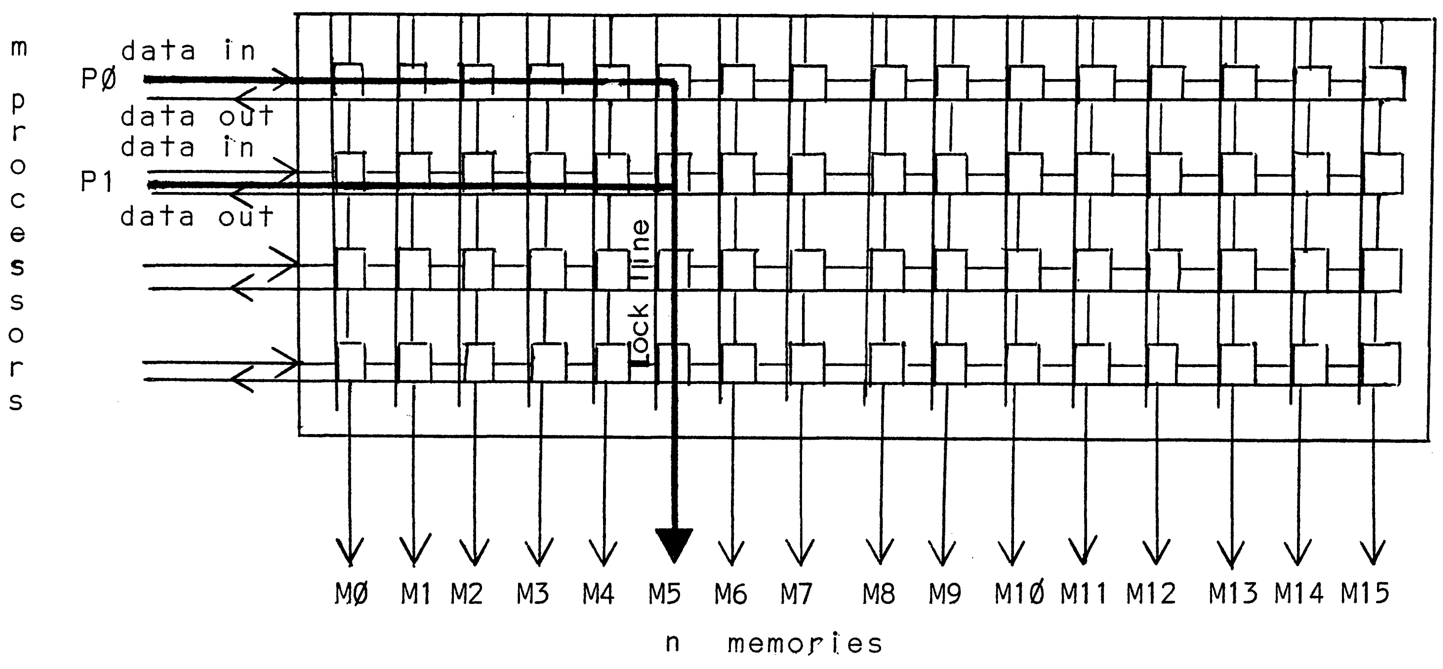
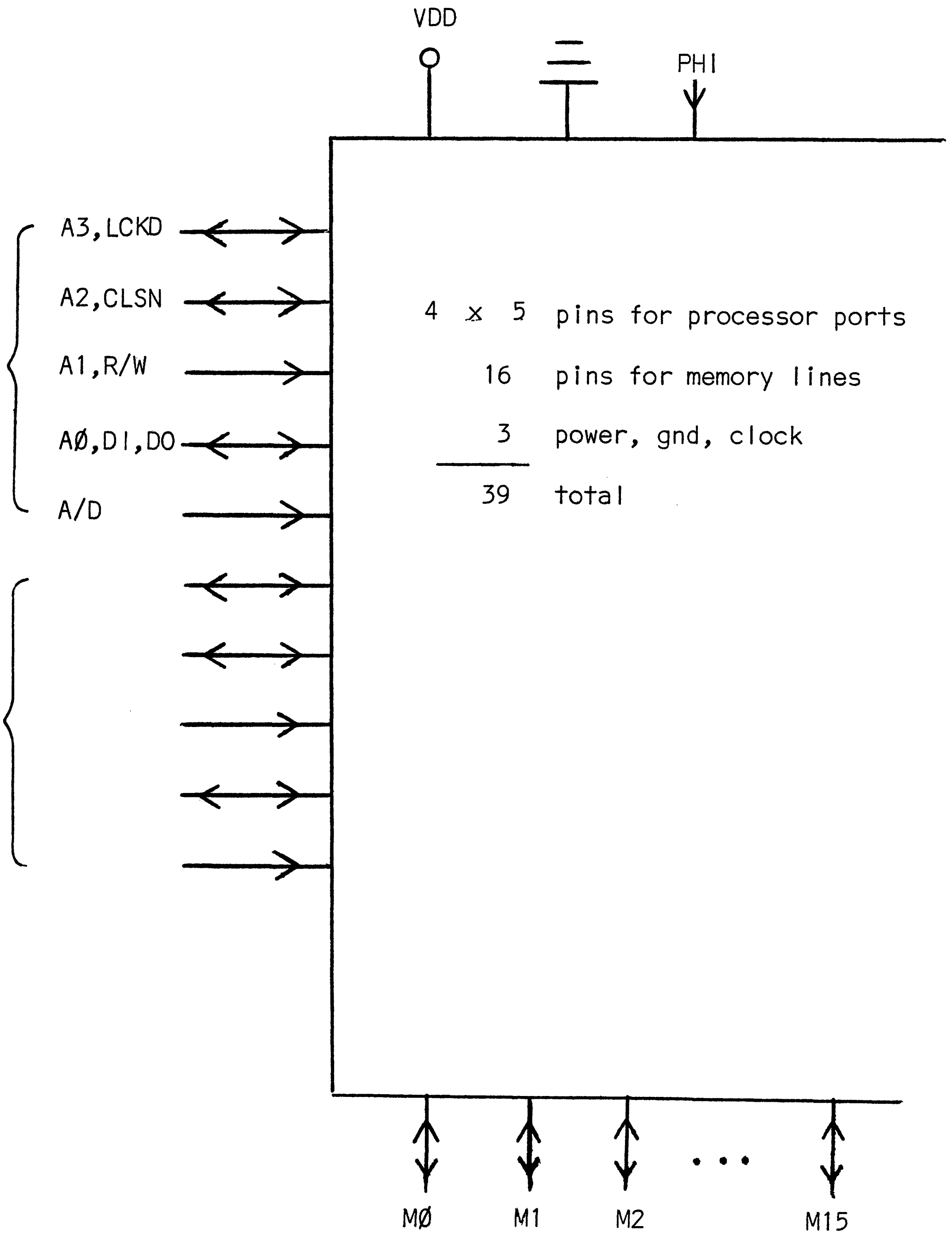
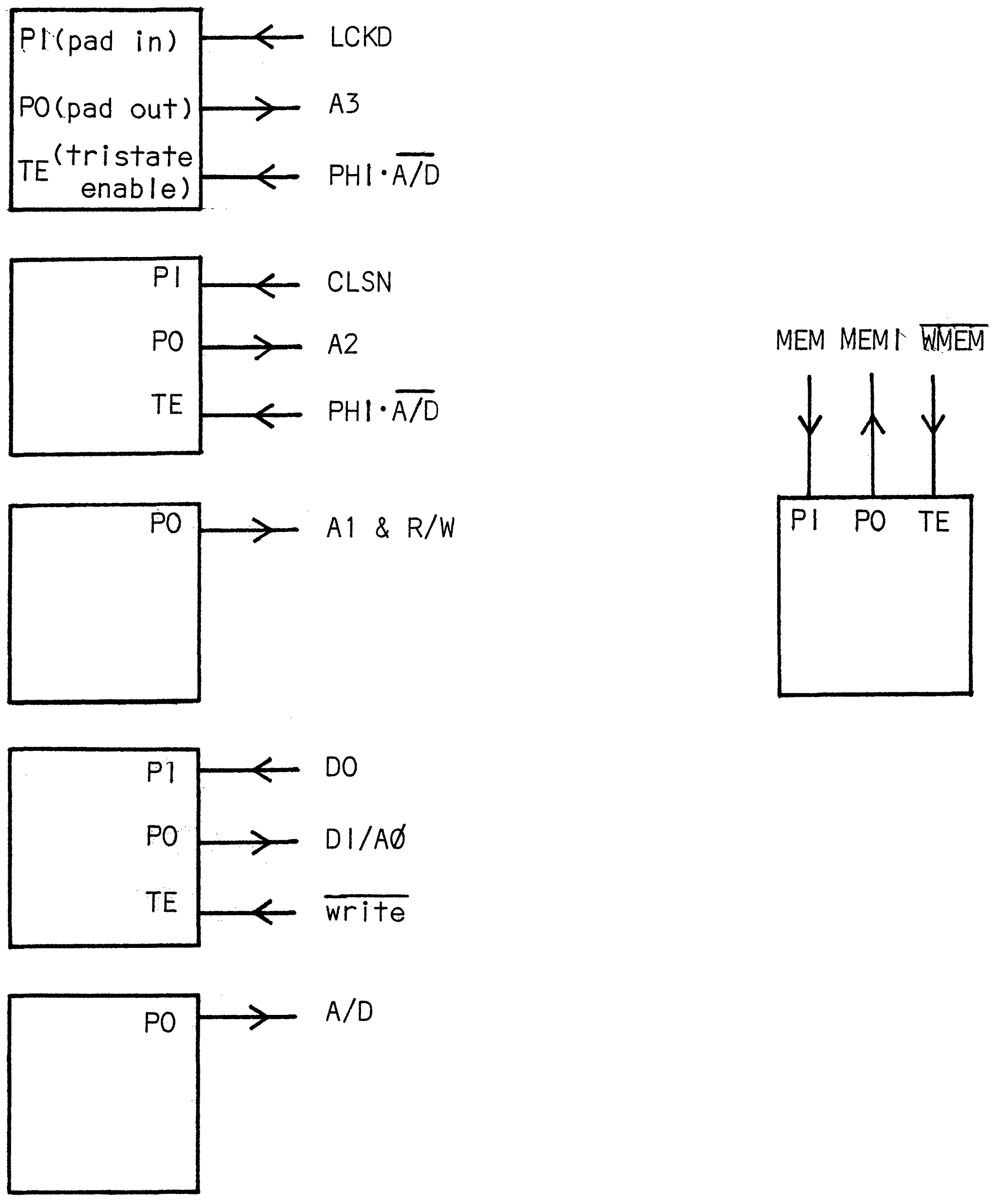

m processors
data in
PØ
data out
data in
P1
data out
Lock line
MØ
M1
M2
M3
M4
M5
M6
M7
M8
M9
M1Ø
M11
M12
M13
M14
M15
n memories

Figure 3



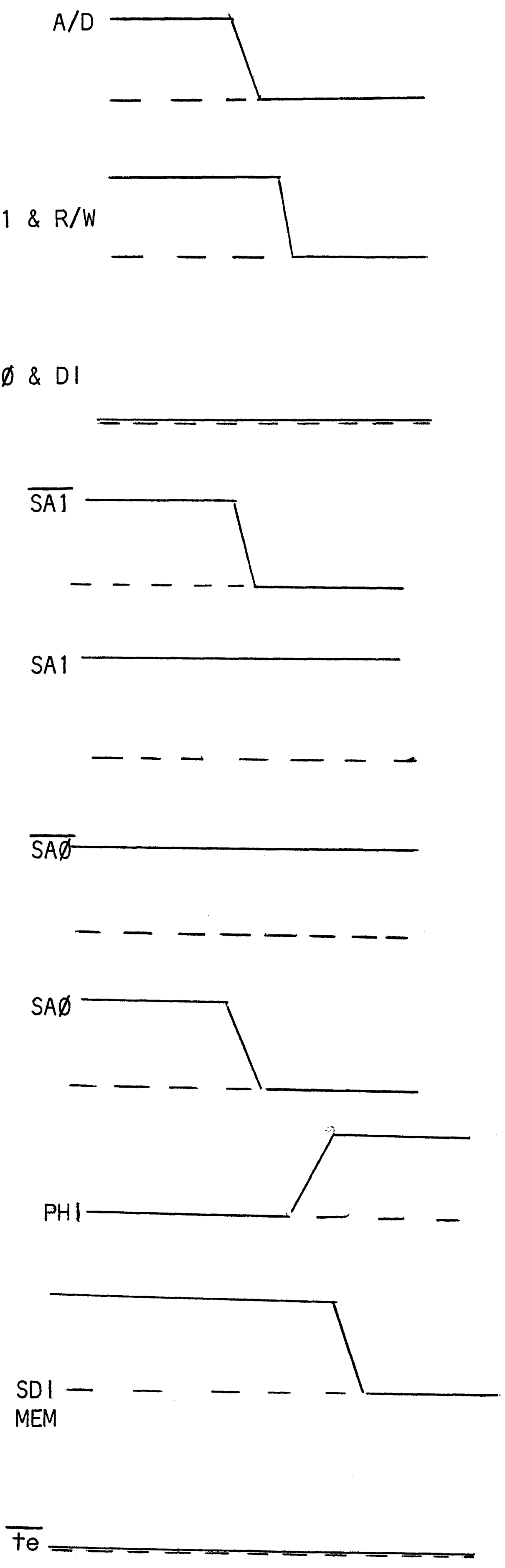
PINOUT

Figure 4

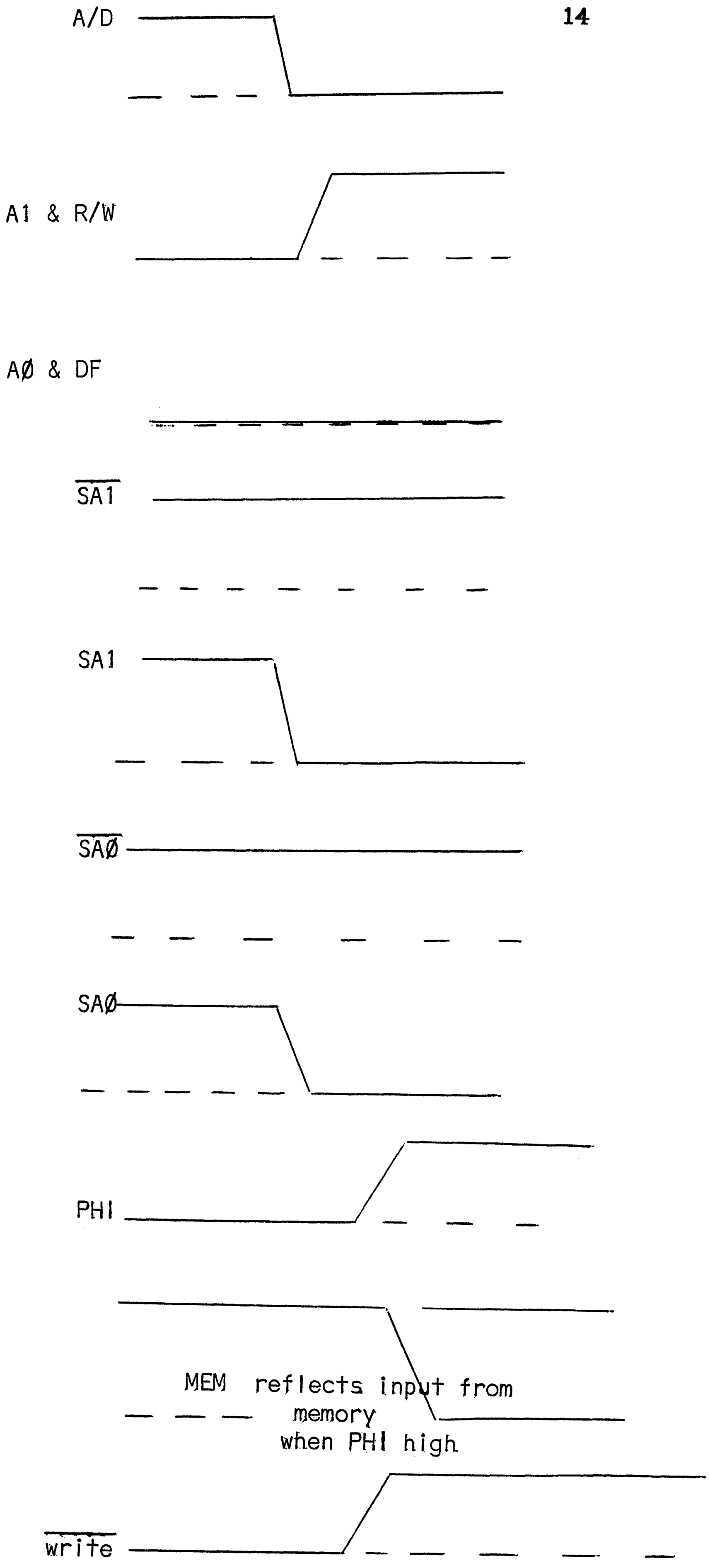


PROCESSOR PORT PADS
& MEMORY PORT PAD

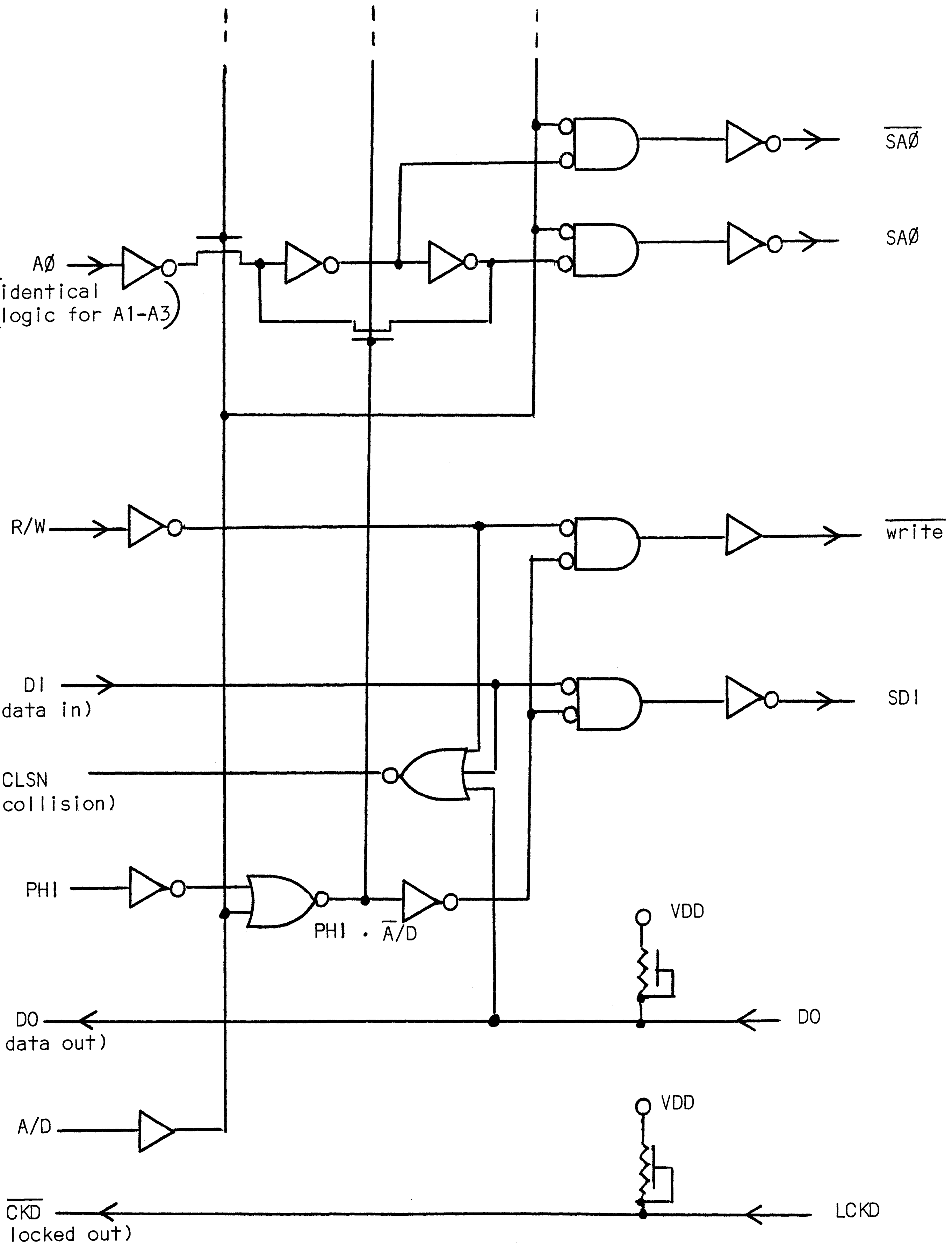
Figure 5



write cycle
F i g u r e   6

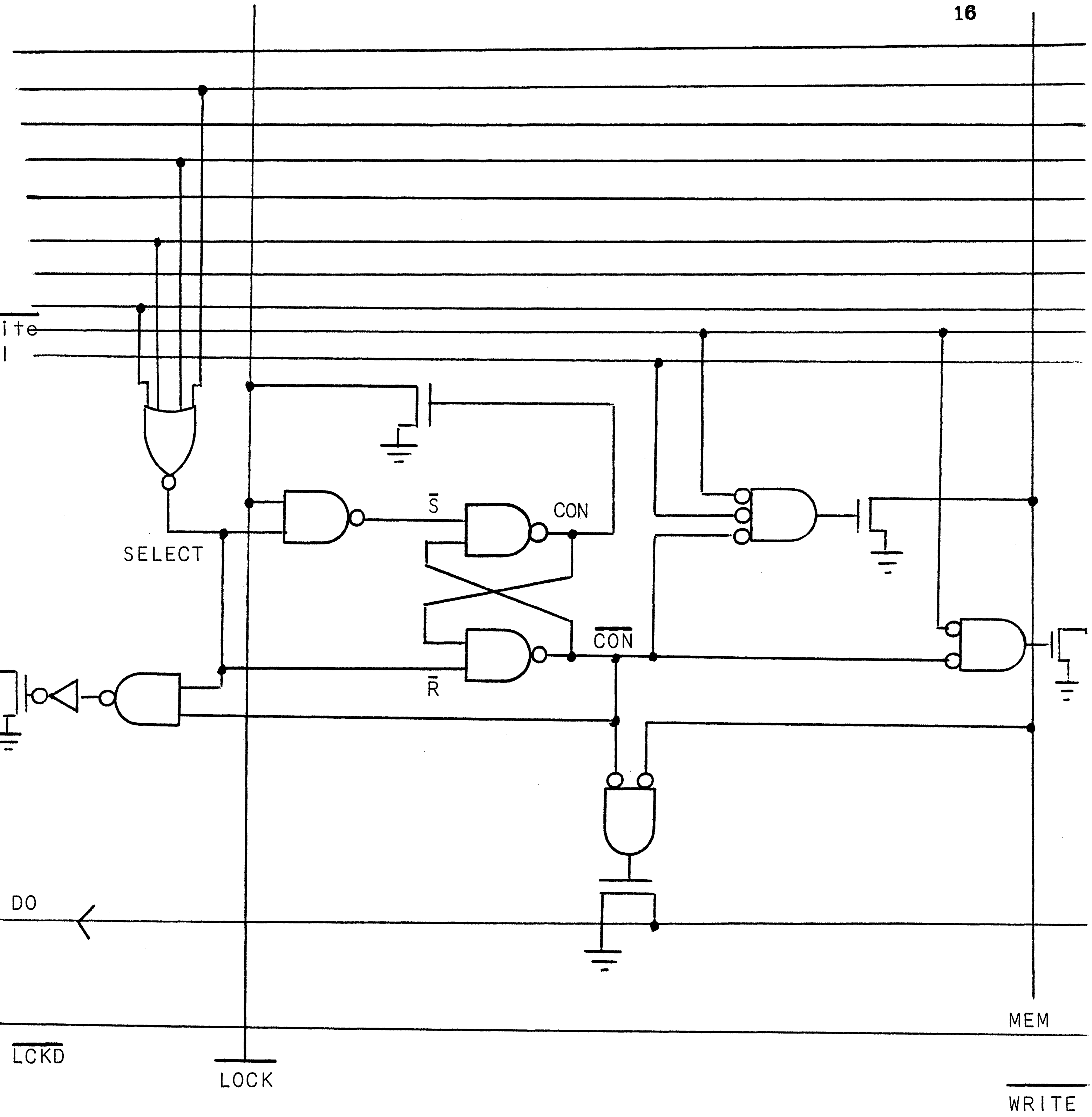


read cycle
F i g u r e   7



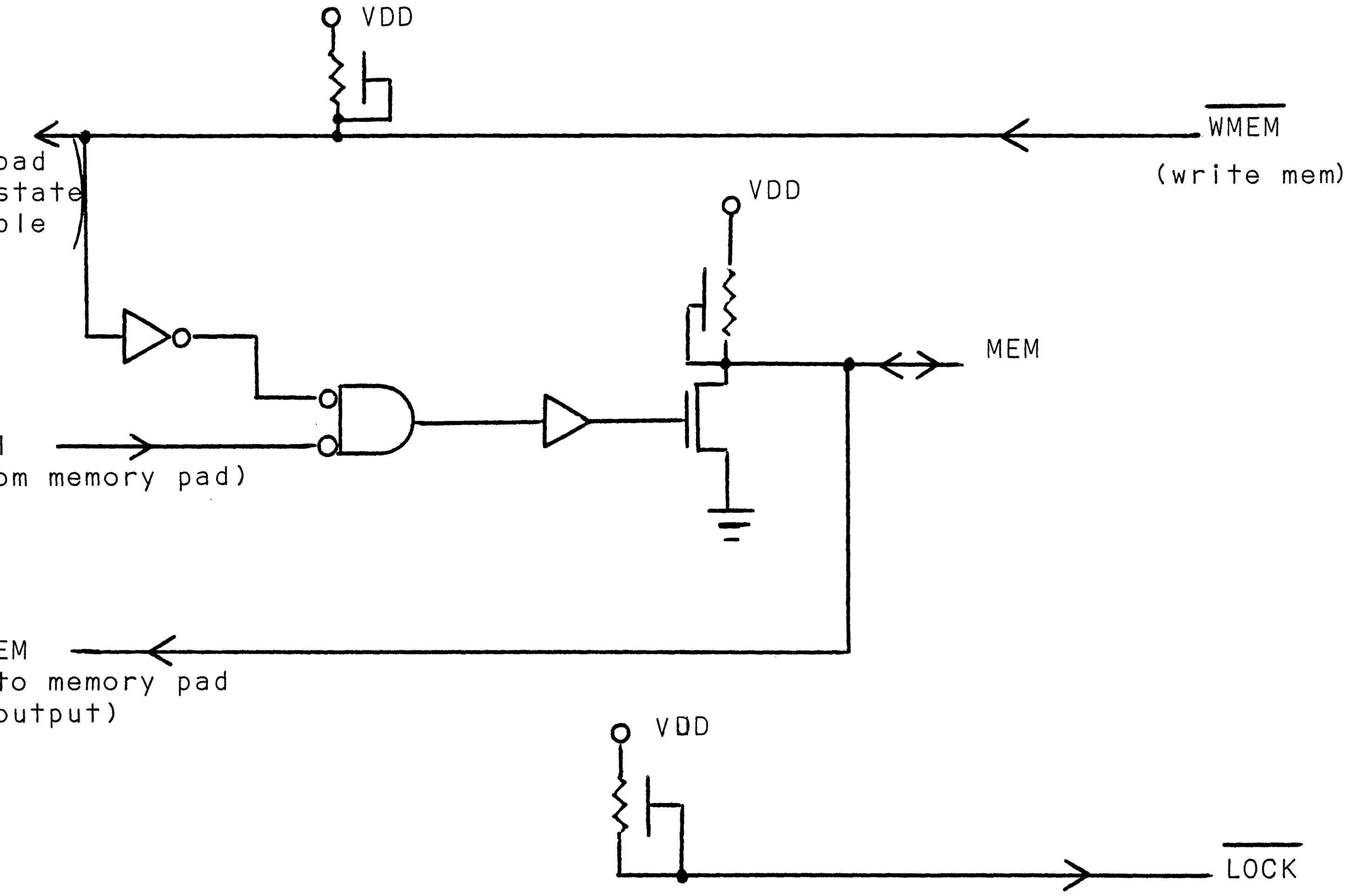
SELECT & COLLISION
LOGIC

FIGURE 8



CROSSPOINT CELL LOGIC

Figure 9



MEMORY & LOCK LINE
LOGIC

Figure 10

ırn: Thu Jun 10 12:36:31 1982
ifplot* Window: 23417 576582 -576582 -23417 --- Scale: 1 micron is 0.001330 i

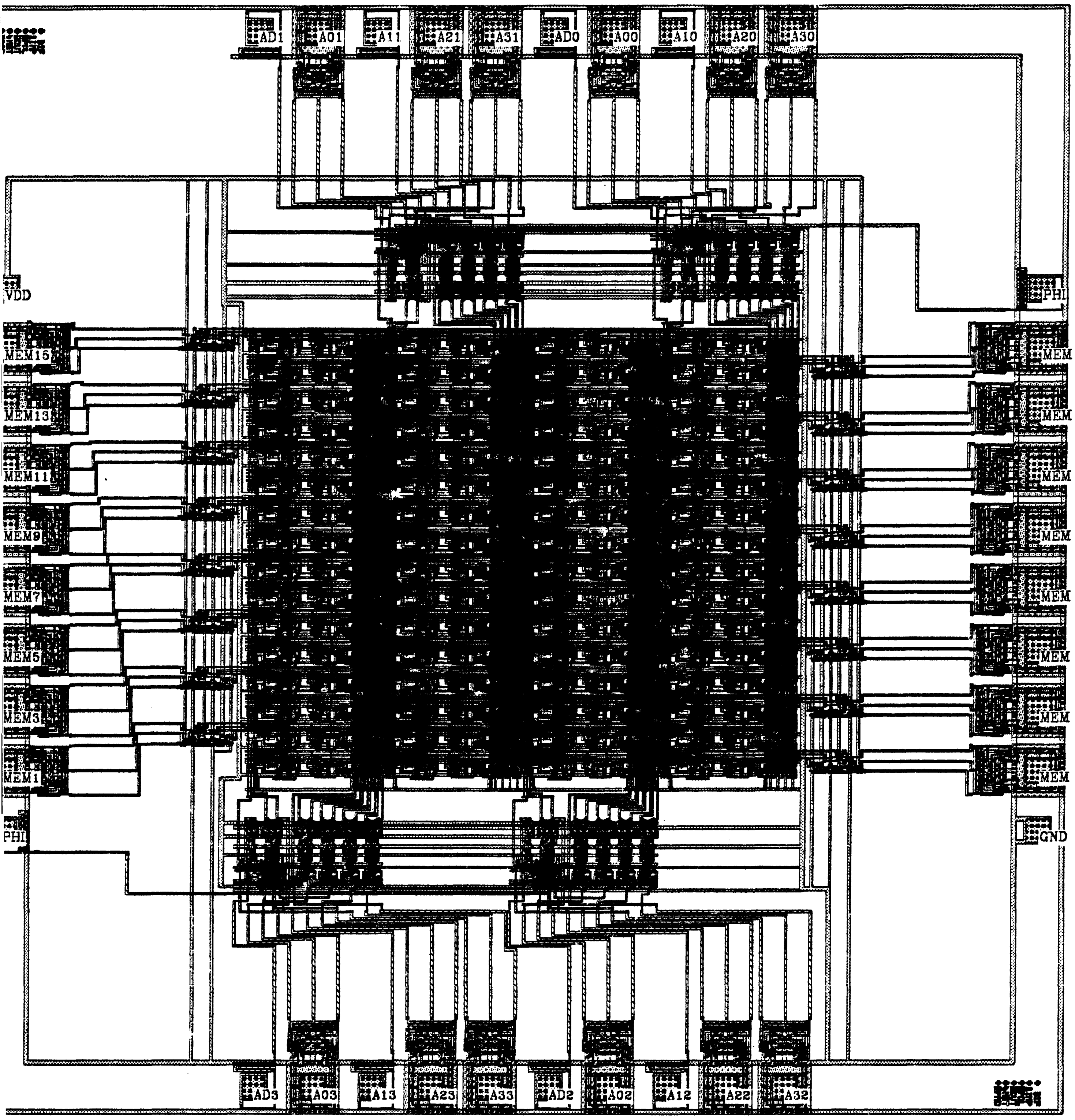


Figure 11