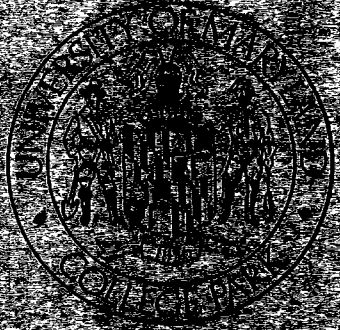
THESIS REPORT
Master's Degree



SYSTEMS RESEARCH CENTER

Supported by the National Science Foundation Engineering Research Centers Program (NSFD CDR-8803012), Industry and the University

The Design and the Testing of a 64-Processor Array

by : H. Wang
Advisor: J. JáJá

M.S. 89-4
Formerly TR 88-46

The Design and the Testing of a 64-Processor Array[1]

Hsinshih Wang
Department of Electrical Engineering
Systems Research Center
University of Maryland
College Park, MD. 20742

Joseph JáJá
Department of Electrical Engineering
Institute for Advanced Computer Studies
Systems Research Center
University of Maryland
College Park, MD. 20742

[1]Supported by NASA contract number NAG5-776.

Abstract

Array architectures based on the VLSI technology allow the processing speed to increase by several orders of magnitude. While VLSI holds the promise of high parallelism by offering almost unlimited hardware at very low cost, there are several inherent constraints with respect to communication, design complexity, testability, etc..

In this paper, we are concerned with designing and testing of such an architecture. An array-processor chip consisting of 8×8 processing elements (PEs) each with 512 bits of memory was fully designed and fabricated using 2μ CMOS technology. One of the novel features of this design is the capability to load data fast into all the PEs simultaneously. Extensive simulations were carried out on this design. This general purpose parallel processor chip was tested using the test workstation IMS–VS2000. An application board was also built by using the Macintoch II as the host controller.

CHAPTER
ONE
Introduction

The Massively Parallel Processor (MPP) was developed by Goodyear Aerospace Corporation for the NASA Goddard Space Flight center [Ba1,Ba2,Po]. The MPP is a unique Single Instruction Control of Multiple Data Streams (SIMD) processor configured as a 128 by 128 array of processors (Composed of 2,048 LSI chips, which has 8 PEs per chip). The high speed digital processor using thousands of processor elements (PE's) operating simultaneously (massive parallelism) is designed to solve two-dimensional data processing problems such as those encountered in satellite imagery. A block diagram of the hardware elements of the MPP is shown in figure 1.1.

An integrated circuit chip which has 64 PEs configured as a 8 by 8 array and 512 bits of on-chip memory per PE was designed. The integration of complete processors onto a single area of silicon offers significant potential advantages in terms of speed, reliability and package economy. Speed is improved by removing the need to drive highly capacitive off-chip loads. The speed advantage gained may be as high as by a factor of 10 for CMOS [Neuman, 1985]. There are a wide range of applications, for example, signal/image processing, matrix based operations, dynamic programming (Kung 1982), etc..

1.1 System Overview

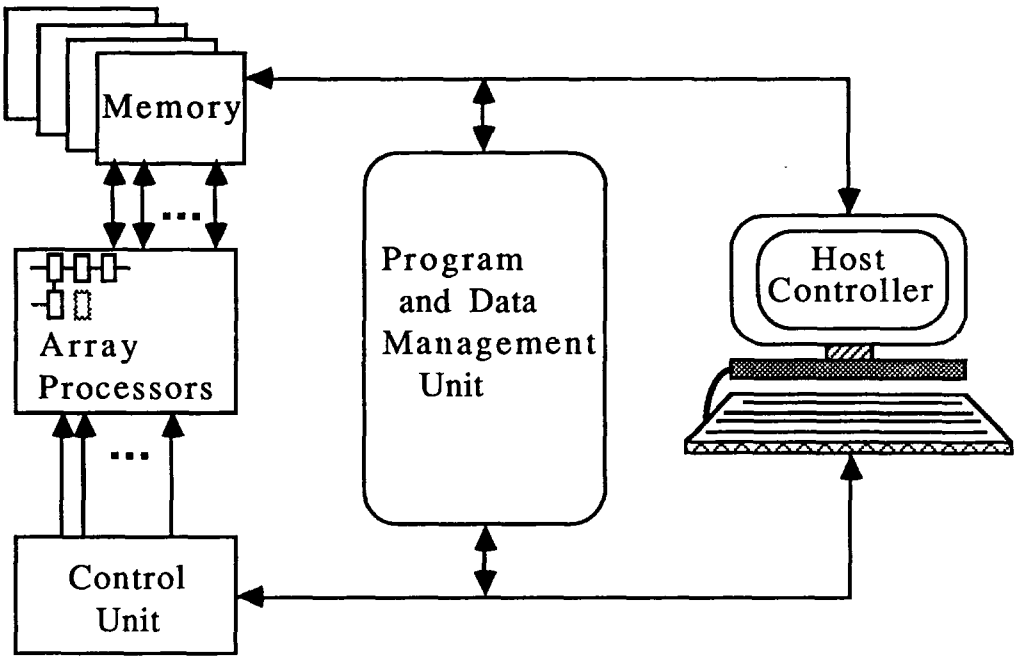


Figure 1.1 MPP Block Diagram

In this section we would like to illustrate the chip design, the architecture of the 64–processor array chip, and the design and building of application board.

1.1.1 Hierarchical Design Methodology

Establishing an efficient design methodology is one of the keys to successful VLSI chip design. Figure 1.2 shows the flowchart of the chip design process.

Some of the problems associated with VLSI design are:

1. *How to design* large complex systems in a reasonable time and with reasonable effort. This is a problem shared with other approaches to system design.

2. *The type of architectures* best suited to take full advantage of VLSI.

3. *The testability* of large/complex systems once implemented in

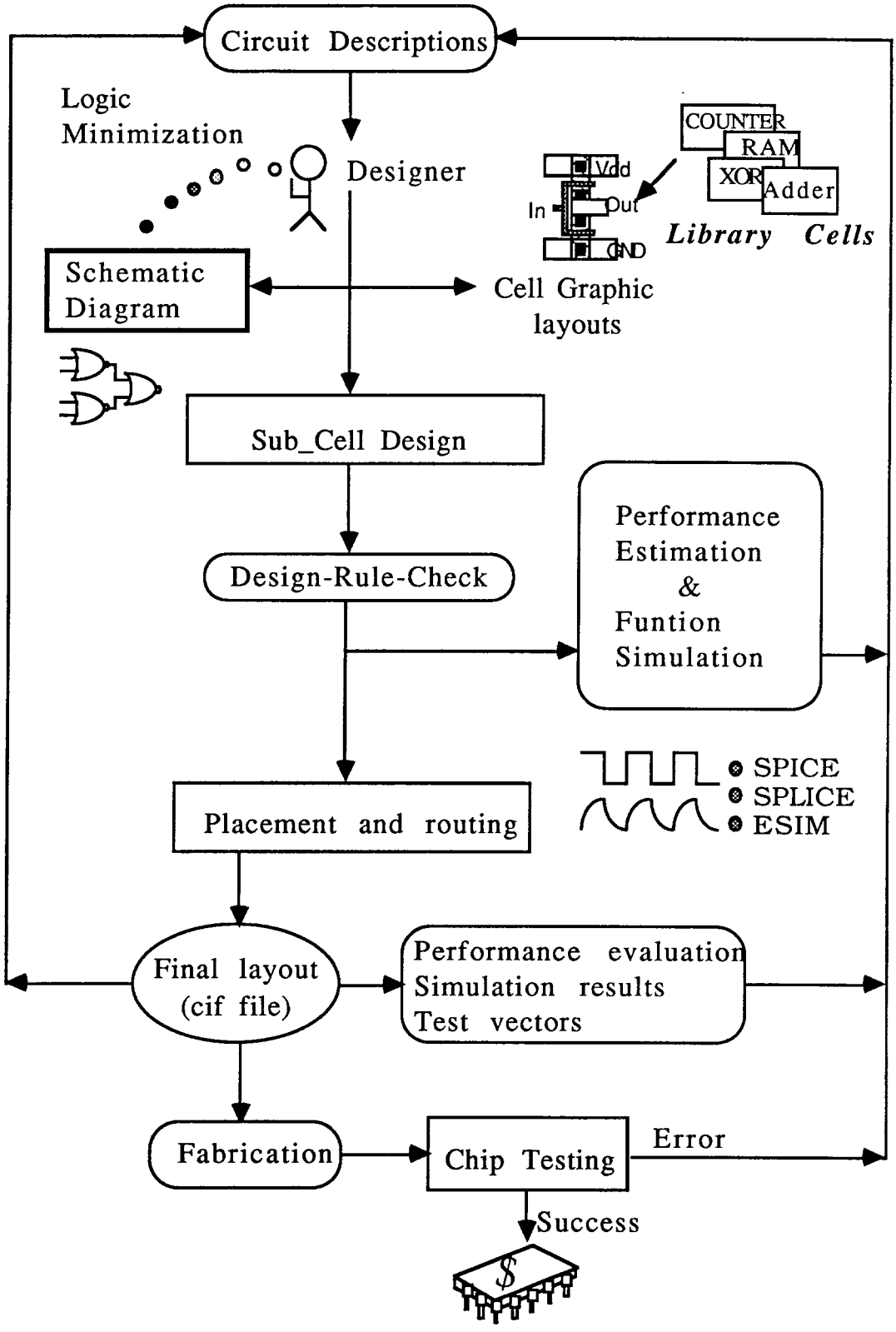


Figure 1.2 Chip Design Flowchart

silicon.

The design approach in a top-down manner and with adequate computer-aided tools can greatly reduced problem 1. Design testability into the system from the outset can solve problem 3, but be prepared to devote a significant proportion (up to 30%) [Pucknell, 1985], of the total chip area for testing and diagnostic facilities.

In tackling the design of a system, we must bear in mind that topological properties are generally far more significant than the logical operations being performed. It may be said that *It is better to duplicate than to communicate*, and this is indeed the case. Communications must therefore be given the highest priority early in the design process and a communication strategy should be involved and adhered to throughout the design process.

1.1.2 System Architecture

Figure 1.3 shows the architecture of the MPP application board. By using Macintosh as a host controller, we load program and data into the program memory and data memory respectively through the Hurdler card (will be described in section 1.1.3). A single instruction from the Mac II will stimulate the control unit and cause the MPP chip to fetch both data and commands automatically from the data/program memory, after the task is accomplished, a signal (interrupt) will be sent back to the host controller and cause the host controller to load the desired data into the host controller from the data memory.

Note that the MPP chip is synchronized with the host controller, i.e. the MPP chip's clock is obtained from the host controller, so that during the *hand shake* they won't lose any data.

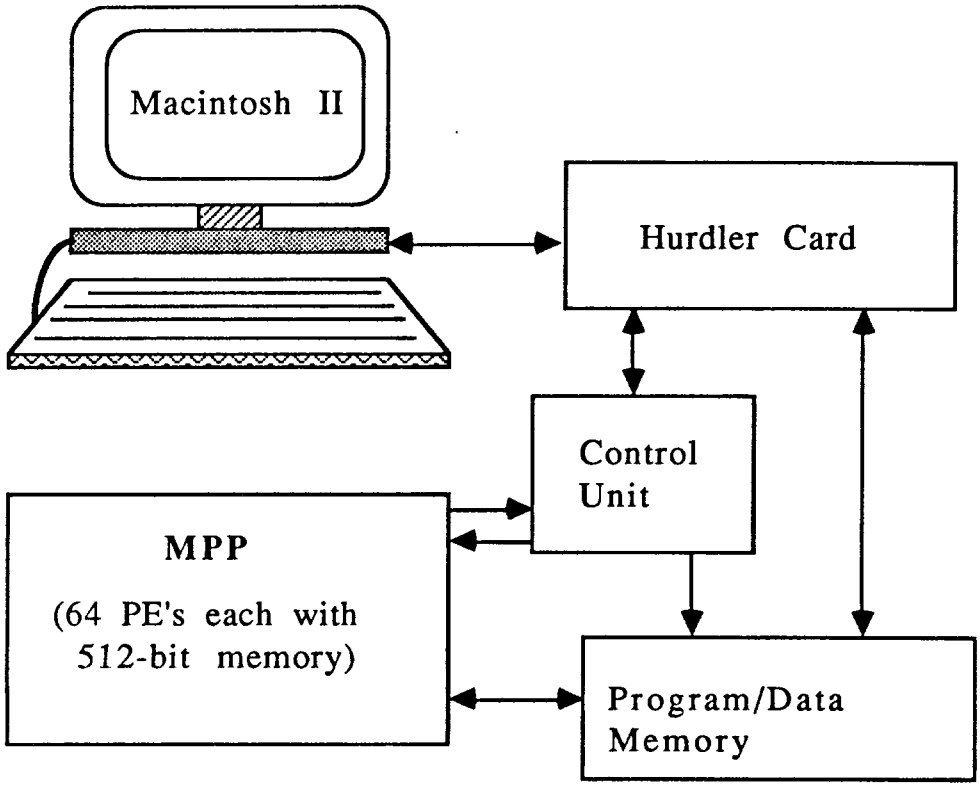


Figure 1.3 Architecture of the MPP board

1.1.3 The Hurdler II Interface card

In order to connect Macintosh II with the MPP application board, we need an interface between the Macintosh II's NuBus and the data/address/control bus of the board.

A simplified block diagram of the Hurdler card is shown in figure 1.4. The interface circuitry is mainly responsible for latching addresses and buffering data lines. The address decoder detects that the NuBus has addressed the card, and supplies an address latch signal to the interface circuitry, and a slot selected signal to the state sequencer. The state sequencer provides basic timing and control signals. The state decoder synthesizes, based upon the address presented to the interface circuitry. Throughout the board, important signals are brought out to user accessible pads and headers. By using this interface card, all control and data signals can be loaded into the data/program RAM, thus control the MPP application board.

1.2 The Massively Parallel Processor (MPP) Chip

The MPP chip is essentially an array of processors with the nearest neighbour interconnections and some on/off chip memory used as data storage. The block diagram of the MPP chip is shown in figure 1.5.

1.2.1 Goodyear 8–Processor Array Chip

The basic building block for the Goodyear Aerospace Corporation's MPP machine is a array unit (ARU), the PE's in the ARU are implemented on a custom designed LSI chip which is partitioned to include eight PE's and is configured in a 2 by 4 array. Each processor is attached to an off–chip 1 K

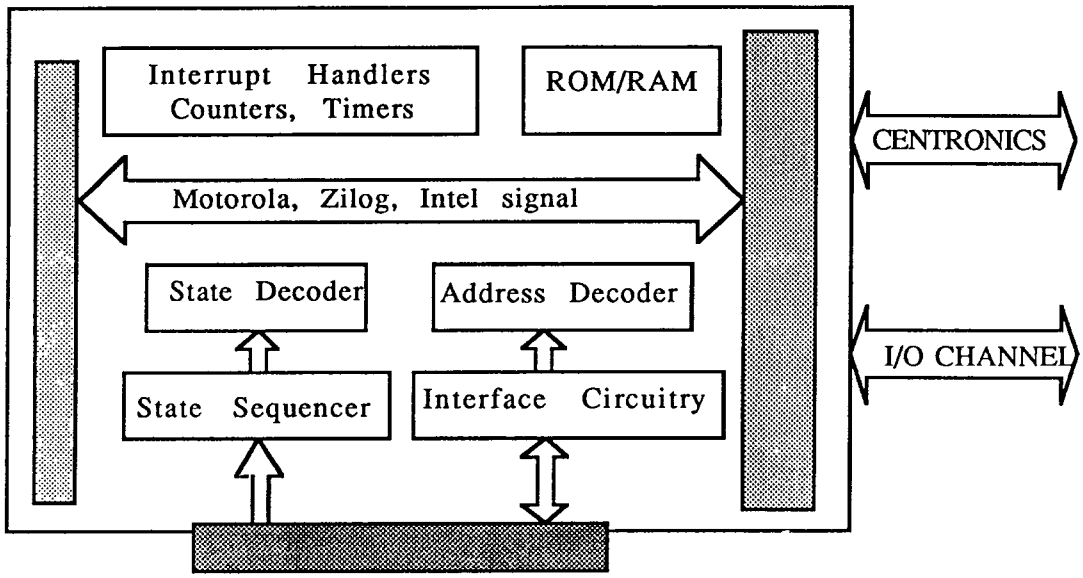


Figure 1.4 Block Diagram of the Hurdler II Interface Card

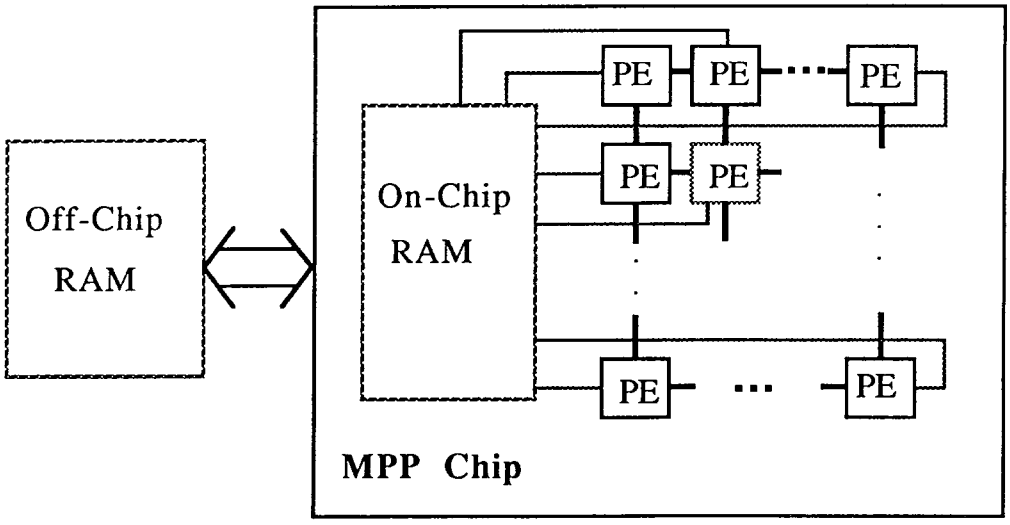


Figure 1.5 Block Diagram of the Massively Parallel Processor (MPP) Chip

bytes of memory on a standard RAM chip.

The Processors themselves consist of an *Arithmetic Unit, a Logic Unit, a Mask Unit and an Input-Output Unit.* Collectively the processors make the array unit, the main component of the MPP machine. In the original design the PE memory was not included within the chip since it was felt that inclusion of the local memory would reduce the number of PE's per chip and complicate the chip design and development.

The main constraint of this MPP is that all data must pass through the VAX memory to get to or from each processor, this significantly reduce the system speed because of driving highly capacitive off-chip loads.

1.2.2 160–Processor Array Chip

An attemp has been made to overcome these problems with the design of a chip that has 16 rows by 10 columns of PE's and 64 bits of on-chip memory per PE. as shown in figure 1.6. However, this chip also has certain disadvantages. First, the *arithmetic unit* of each PE does not contain shift registers, which implies that each PE only has its own 64 bits of memory which it can use for storage of data and temporary results. This is clearly insufficient even for single precision arithmetic. Second, the only means of inputting data into this chip and outputting data from this chip is through the memory. Thus, input and output of data can be done only column by column which reduces the throughput of the system.

1.2.3 64–Processor Array Chip

Keeping all above considerations into mind, an integrated circuit chip which has 64 PEs and 512 bits of on–chip memory per PE is built, the block

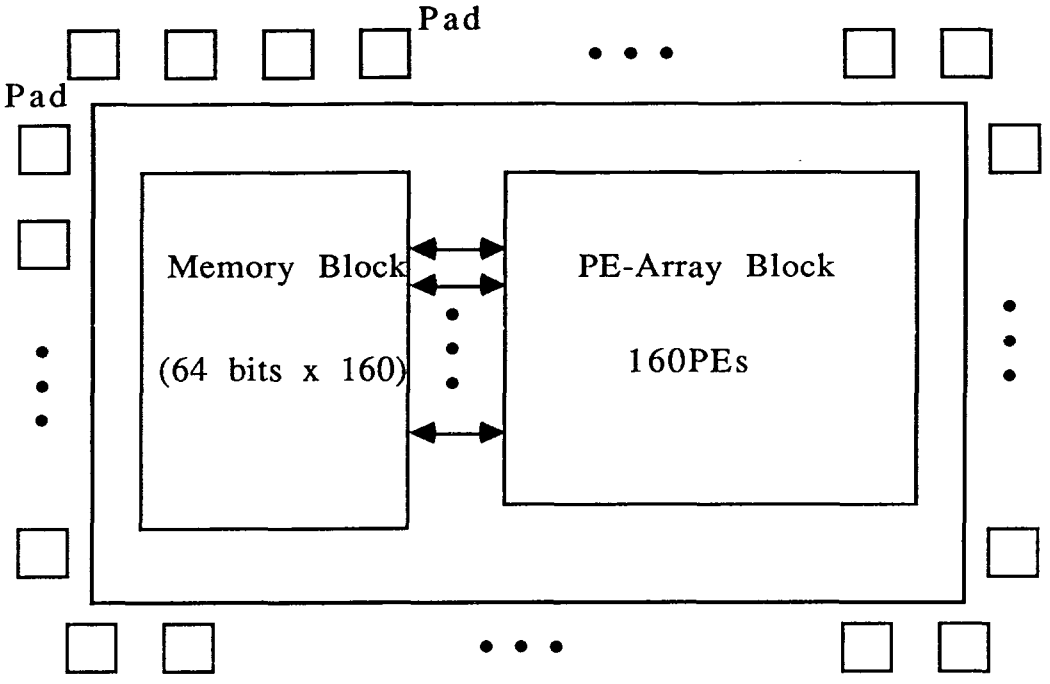


Figure 1.6 160–Processor Array Chip

diagram of this chip is shown in figure 1.7.

Data can be input into and output from all PEs of this chip via its own local memory in one machine cycle. Furthermore, a variable length shift register (up to 32 bits) cell is built into each PE. Combining both 512 bit local memory and the variable length shift register (can be set length to 2, 4, 8, 16, and 32) for each PE, there are sufficient temporary storage even for double precision arithmetic. The design of this chip is described in chapter 2.

1.3 Testing Enviornment

After a chip is sent back from a fabrication factory, one has to do the following:

1. Functional verification.
2. Timing measurement.

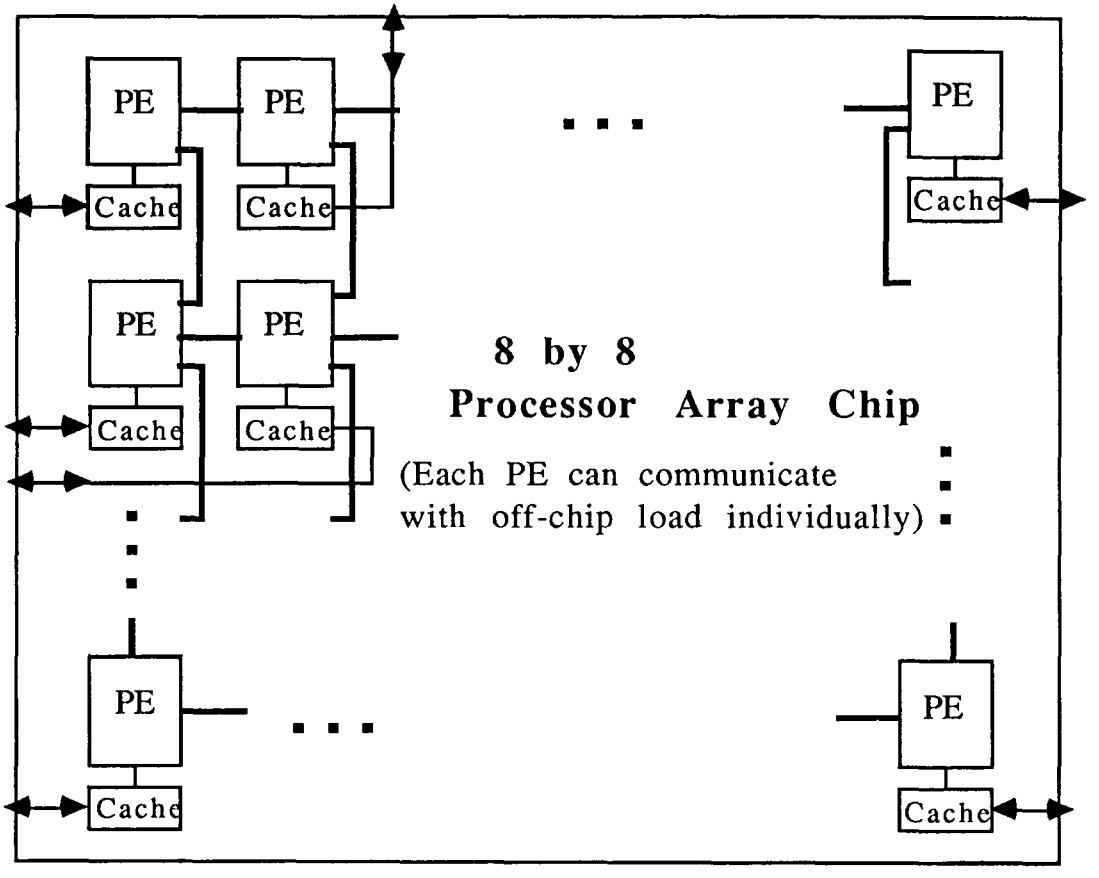


Figure 1.7 64–Processor Array Chip

3. Maxium frequency measurement.

4. DC parameter measurement.

The IMS workstation is an interactive test and verification system for digital design. It provides a structured environment for prototype analysis and allow real-time comparison of the design's performance against expected data output.

The basic system is made up of a host computer, one or more IMS HS1000 mainframes and a verification station. By using this workstation, the 64–processor array is tested, This will be discussed in more details in chapter 3.

CHAPTER
TWO

The 64–Processor Array Chip Design

2.1 Chip Design

The floor plan of the 64-processor array chip, designed using scalable CMOS technology is shown in figure 2.1. The chip was fabricated using 2μ ($\lambda = 1\mu$) digital process by MOSIS. This 64-processor chip contains 64 bit-serial processing elements (PE), and 512 bits of local memory per PE. The layout of the chip is shown in plate 1.

2.1.1 Instruction Set of The Chip

The operations performed by the PE's during a machine cycle are governed by a 16-bit instruction code. The instruction code is broadcasted to all PE's in synchronism, i.e. all PE execute the same instruction in the same time, but may fetch or execute different data, depending on what kind of data were stored in the data memory, or just remain idle by setting the mask register to logic 0.

The 16-bit operation codes are labeled $I_0, I_1, I_2, ...I_{14}, I_{15}$. While bits I_0, I_1, and I_2 control the source of data on the data bus. Bits I_3 through I_7 specify the logic/routing/set-lengths (of shift registers) operation via register P, Q, and data bus. Bits I_8, I_9, and I_{10} control the operation of the arithmetic unit. Bits

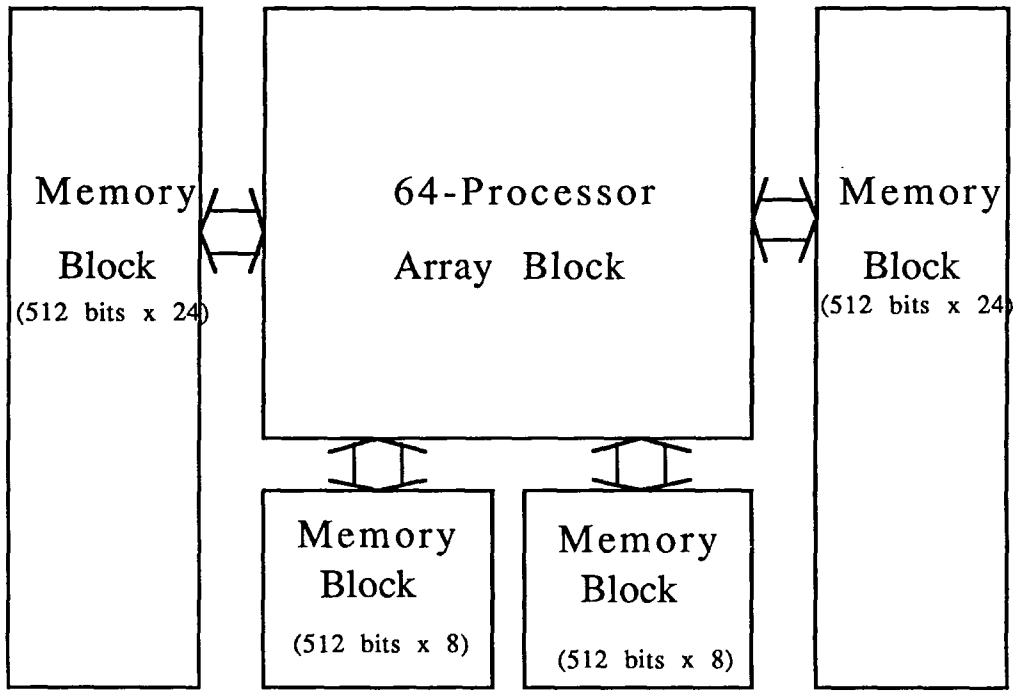


Figure 2.1 64-Processor Array Chip Floor Plan

I_{11} and I_{12} specify register A, G, and Q; while bits I_{13}, I_{14} and I_{15} control the memory's read or write. The instruction set is shown in table 2.1.

Five PE operations can occur simultaneously during the same machine cycle. The operations are synchronized so that when a register changes its state, it won't affect another operation's output, i.e. by using a two phase non-overlapping clocks ϕ_1 and ϕ_2, as shown in figure 2.2, the PE's can do 5 parallel operations in the same machine cycle.

2.1.2 The PE Array Block

This block contains 64 PE's arranged as an array of 8 rows by 8 columns.

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	I_{11}	I_{12}	I_{13}	I_{14}	I_{15}
$D \leftarrow B$	0	0	0													
$D \leftarrow C$	0	0	1													
$D \leftarrow P$	0	1	0													
$D \leftarrow Cache$	0	1	1											0	1	1
$P \leftarrow 0$				1	0	0	0	0								
$P \leftarrow D\bar{Q}$				1	0	0	0	1								
$P \leftarrow \bar{D}\bar{Q}$				1	0	0	1	0								
$P \leftarrow \bar{Q}$				1	0	0	1	1								
$P \leftarrow \bar{D}Q$				1	0	1	0	0								
$P \leftarrow D \oplus Q$				1	0	1	0	1								
$P \leftarrow \bar{D}$				1	0	1	1	0								
$P \leftarrow \bar{D} + \bar{Q}$				1	0	1	1	1								
$P \leftarrow DQ$				1	1	0	0	0								
$P \leftarrow D$				1	1	0	0	1								
$P \leftarrow \overline{D \oplus Q}$				1	1	0	1	0								
$P \leftarrow D + \bar{Q}$				1	1	0	1	1								
$P \leftarrow Q$				1	1	1	0	0								
$P \leftarrow D + Q$				1	1	1	0	1								
$P \leftarrow \bar{D} + Q$				1	1	1	1	0								
$P \leftarrow 1$				1	1	1	1	1								
$P \leftarrow P_n$				0	X	0	0	0								
$P \leftarrow P_s$				0	1	0	0	1								
$P \leftarrow P_w$				0	X	0	1	0								
$P \leftarrow P_e$				0	X	0	1	1								
Set length = 2				0	0	1	0	0								
Set length = 4				0	0	1	0	1								
Set length = 8				0	0	1	1	0								
Set lenght = 16				0	0	1	1	1								
Set lenght = 32				0	0	0	0	1								
Full add									0	0	0					
Half add									0	0	1					
Set C									0	1	0					
Clear C									0	1	1					
Load A												0	0			
Load G												0	1			
Load Q												1	0			
$Cache \leftarrow D$														0	0	0
$Cache \leftarrow RAM$														0	0	1
$RAM \leftarrow Cache$														0	1	0
NOP	1	1	X	0	1	1	X	X	1	X	X	1	1	1	X	X
STOP	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Table 2.1 Instruction Set for the 64-Processor Array Chip

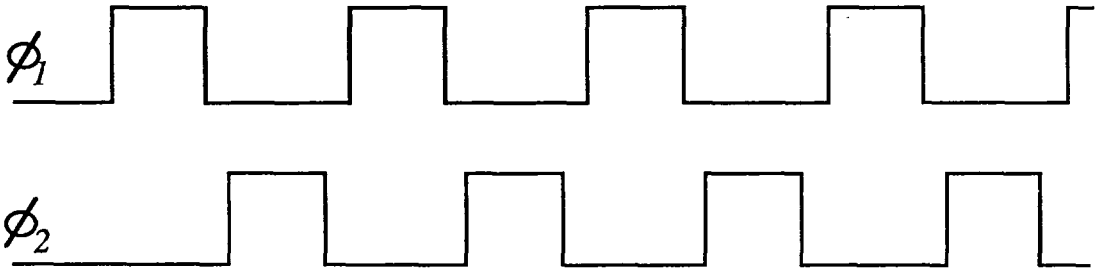


Figure 2.2 The Clocks of the 64-Processor Array Chip

The PE's are identical bit serial processors capable of handling either masked or unmasked arithmetic and logic operations. Each processor is connected to its nearest neighbours; and can communicate with the neighbors via the P-register.

The key to have a compact PE array is to design the PE as compact as possible, once the PE has been designed as compact as possible and adequate care has been taken into consideration the direction of data flow and the control signals, the job of gluing up a compact PE array block is simple. In section 2.2 we shall describe the design of a compact PE.

2.1.3 Memory Block

The memory block consists of several parts, namely the dynamic memory itself, the corresponding precharge circuitry, the column decoder/driver, row selector/driver and the refresh circuitry.

In this chip, each PE has 512 bits of local memory, ie. the chip has altogether 32 K 1-bits dynamic RAM. The 512 bits of local memory is organized

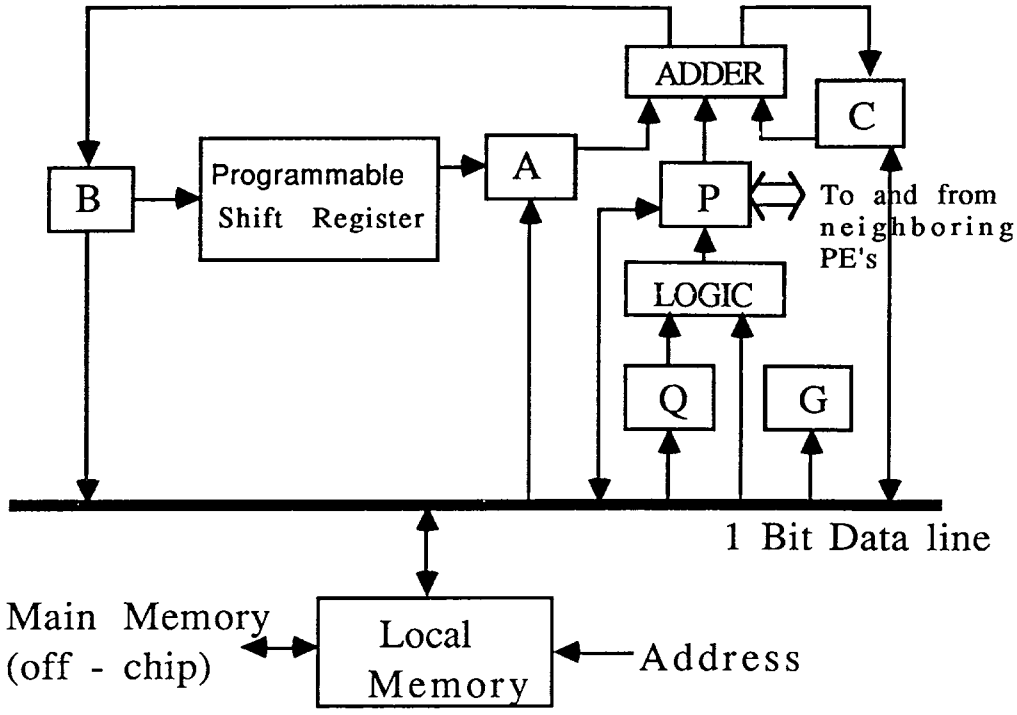


Figure 2.3 Block Diagram of the Processing Element (PE)

as an array of 8 rows by 64 columns, this was done by taking into consideration the maximum chip size available. The memory designed will be discussed in section 2.3

2.2 The PE Design

There are altogehter 6 single bit registers – A, B, C, G, P and Q; a variable-length shift register (up to 32 bits); an adder; a router and some combinational logic in this single bit processing element (PE), A block diagram of the PE including the local memory is shown in figure 2.3. The PE can be further divided into three subunits – Arithmetic subunit, Logic and Routing subunit and the Masking subunit. These subunits have independent control codes but

share two common clocks ϕ_1 and ϕ_2. The data bus is interconnected to the local memory by a bidirectional bus, all the operations of the PE and the data flow are governed by the 16 bit operation codes shown in table 2.1, the layout of the PE is shown in plate 2.

2.2.1 Arithmetic Subunit

The arithmetic subunit consisted of three registers (A, B, and C) and a variable-length shift register block. The operations of the arithmetic subunit is governed by instructions I_3 through I_{10}. This subunit can perform either half-addition or full-addition, the temporary result can be stored in the variable-length shift register.

A full-addition operation takes the bits stored in the registers A, C and P at the beginning of the machine cycle, perform the full-addition operation and stores the carry-bit in the C register and the sum-bit in the B register at the end of the machine cycle. A half-addition operation is similar to the full-addition except that the datum stored in the P-register is ignored and only bits from register A and C are added. The truth table of the full-addition and half-addition operation are shown in table 2.2. The programmer can also set the C-register to either logic 1 or 0 by simply specifying the instruction codes I_8, I_9, and I_{10} to 0, 1, 0 or 0, 1, 1 respectively.

The programmer can also use the variable-length shift register for local storage of partial products. This feature significantly improves multiplication and division. The shift-register has N stages, where N can be set to 0, 2, 6, 14 or 30 controlled by the instruction '*set length*' with instruction codes I_3 through I_7. The variable-length shift register is connected from the B-register to the A-register so that operands can circulate around the path through the shift-register. The A-register and B-register add two stages of delay in the

Inputs			Outputs	
A	P (t)	C	B	C (t+1)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a)

Inputs		Outputs	
A	C (t)	B	C (t+1)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(b)

Table 2.2 (a)Truth Table for Full-Addition (b) Truth Table for Half-Addition

recirculation path so that the total number or stages in the path is $N+2$, or 2, 4, 8, 16 and 32.

The shift-register can be shifted one place each machine cycle. Each shift causes the data in the register to move one stage to the right (from register B to A). The first stage of the register is loaded with the state of the B-register at the begining of the machine cycle. Data in the last stage is shifted out to the A-register at the end of the machine cycle.

After some logic simplifications we can get the expressions for register A, B and C as follows:

$$A \leftarrow \bar{I}_{11}\bar{I}_{12}D + \bar{I}_3\bar{I}_4[I_5\bar{I}_6\bar{I}_7B + (I_5I_6 + \bar{I}_6I_7)SR_{30}]$$

$$B \leftarrow \bar{I}_8[\bar{I}_9\bar{I}_{10}(\bar{A}P\bar{C} + \bar{A}\bar{P}C + A\bar{P}\bar{C} + APC) + \bar{I}_9I_{10}(A\bar{C} + \bar{A}C)]$$

$$C \leftarrow \bar{I}_8[\bar{I}_9\bar{I}_{10}(AP + PC + AC) + \bar{I}_9I_{10}(A + C)]$$

, where SR_{30} is the logic value stored in the last stage of the shift register block. the corresponding layout of the variable-length shift register block, register A, register B, and register C are shown in plate 3,4,5 and 6 respectively. Note that all the operations of the Arithmetic subunit are masked if register G is set to logic 0.

2.2.2 Logic and Routing Subunit

The Logic and Routing subunit is formed by P and Q registers, the Data bus and some combinational logic circuitry, which can perform 16 different kinds of logic operations (all the Boolean functions of two variables) including OR, AND, NAND, NOR, XOR, XNOR, ... etc., and the nearest neighbor interconnections. These operation is governed by instruction codes I_3 through I_7.

A logic operation combines the data bit D on the data bus with the status of the Q-register at the begining of the machine cycle, the result is stored in the P-register at the end of the machine cycle. A routing operation reads the status of the P-register in a neighboring PE (from the direction of either north, south, east or west) at the begining of the machine cycle and stores the result in the P-register at the end of the machine cycle.

The operations of registers P and Q are specified in the operation codes shown in table 2.1, this gives:

$$\begin{aligned} P &\leftarrow I_3(I_4DQ + I_5\bar{D}Q + I_6\bar{D}\bar{Q} + I_7D\bar{Q}) + \\ &\quad \bar{I}_3\bar{I}_5(\bar{I}_6\bar{I}_7P_n + I_4\bar{I}_6I_7P_s + I_6\bar{I}_7P_w + I_6I_7P_e) \\ Q &\leftarrow I_{11}\bar{I}_{12}D + (\bar{I}_{11} + I_{12})Q \end{aligned}$$

,where P_n, P_s, P_w, and P_e represent the status of the P-register from the north,

south, west and east PE's. The layouts of the P-register and Q-register can be found in plate 7 and 8 respectively. Note that all the logic and routing operations are masked if register G is set to logic 0.

2.2.3 Masking Subunit

The Masking subunit is formed by the G-register. The Mask bit is loaded from the data bus when instructions code I_{11} is low and I_{12} is high.

All circuit design has been done using precharged logic with two non-overlapping clocks ϕ_1 and ϕ_2. The operations are performed during clock phase ϕ_1. The result of the operation is stored in the corresponding registers of all the PEs during clock phase ϕ_2, if the G-register of that PE is unmasked. . The registers of the masked PEs retain the state they were before the operation. This is achieved by modifying clock ϕ_2 to form clock ϕ_{22}, ϕ_{22} is asserted only if ϕ_2 is asserted and the G-register of that particular PE is not masked. The Boolean equations for the operations are

$$\begin{aligned} G &\leftarrow \bar{I_{11}} I_{12} D + (I_{11} + \bar{I_{12}}) G \\ \phi_{22} &\leftarrow G\phi_2 \end{aligned}$$

the layout of the G-register is shown in plate 9. Note that the result of each operation in the corresponding registers is stored during clock ϕ_{22} instead of clock ϕ_2.

2.3 Memory Design

In the 64-Processor Array chip, each PE has its own 512 bits memory, so

altogether this chip contains 32K 1-bit memory. Because of the chip area, it is impossible to build up a static memory in this chip, only dynamic memory is possible. By using the 3-transistor dynamic RAM as the basic building block, together with some precharge/refresh, and column/row decoder/encoder, the 32K 1-bit memory is built.

In order to increase the speed of the dynamic RAM, a mixed-mode memory architecture is used [Al]. The decoding circuits are static CMOS circuits with low power consumption, but the memory array consists of dynamic cells. This type of architecture would not be possible with NMOS because the power consumption of the decoding circuit would be too great. The layout of the 512 bit memory together with the refresh/precharge circuits and the row/column decoder/encoder is shown in plate 10.

2.3.1 Dynamic Memory Cell

The basic principle of operation of the cell can be explained with reference to figure 2.4, which shows one bit slice of memory with associated control circuits. The basic bit cell is enclosed in the dashed box. The $\overline{R\ data}$ line is a precharged bus which is charged high during ϕ_1. Reading and writing are done during ϕ_2. If W data=1 and W control line is high during ϕ_2, the transistor T_1 will be on, and transistor T_3 will be off (can not read and write in the same time), the charge will be stored in the gate capacitance C of transistor T_2; for W data=0 in the same case the charge stored in the gate capacitance C of T_2 will be discharged. For the read operation: when the R control line is high during ϕ_2, transistor T_3 will be on and transistor T_1 will be off, if the charge stored in the gate capacitance of transistor T_2 is 1, transistor T_2 will be on and the $\overline{R\ data}$ bus will be pull low; if the charge stored in the gate capacitance of

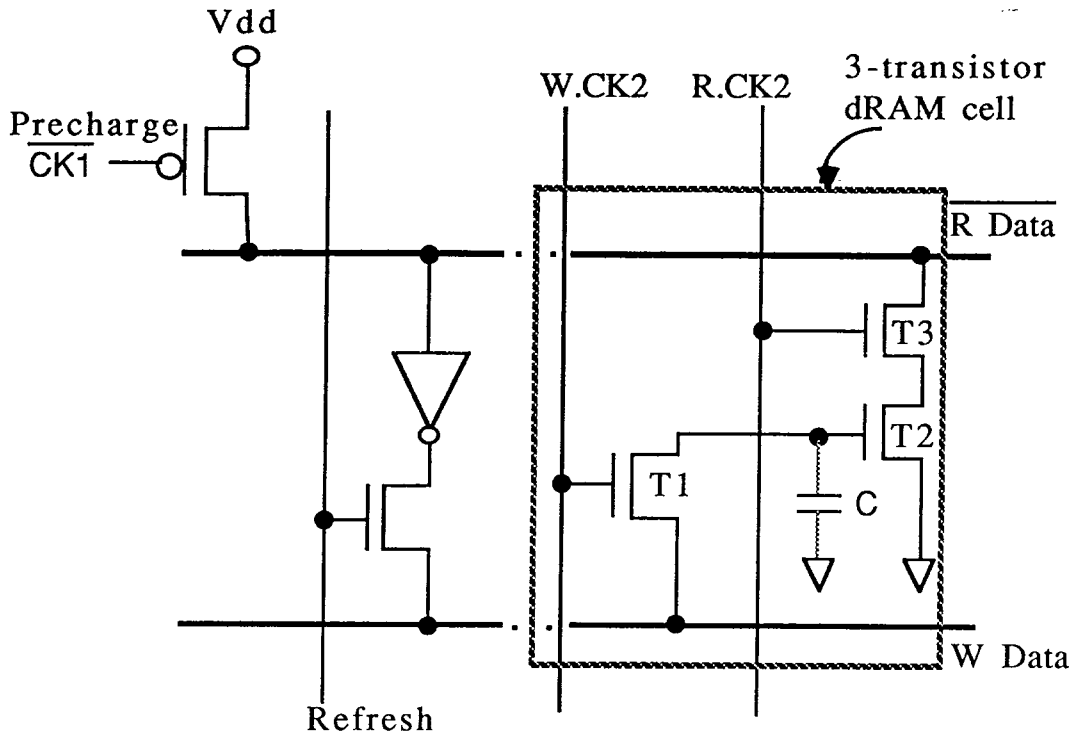


Figure 2.4 Three-transistor dynamic memory cell

transistor T_2 is 0, transistor T_2 will be off so that the $\overline{R\ data}$ bus will remain charged. Note that the reading is non-destructive, and that the $\overline{R\ data}$ line get the complement of the stored bit. By placing a large invert-buffer at the right end of the bus, we can read out the true value of that particular cell. The layout of the three-transistor dynamic memory cell is shown in plate 11.

2.3.2 Precharge/Refresh Circuitry

The dynamic memory cell we used in this chip is sometimes classified as a 3-2-2 configuration (three transistors, two addresses, and two bit lines), although the reading is nondestructive, we need to refresh the cell in order to restore the loss of charge due to leakage, this can be done by driving both R and W line high simultaneously during ϕ_2 on a given column and turning on all the refresh amplifiers for all the rows at the same time. Thus the overhead

support circuitry for this type of memory is moderate to high, but the dynamic memory array is about four times as dense as static RAM [Mu].

The use of precharge techniques in the design can greatly enhance the speed of the operation. Sense amplifiers are used for each row in order to boost the signal levels at the bit lines. The corresponding layout of the precharge/refresh circuit is shown in plate 12.

2.3.3 Row/Column Decoder/Selector

One cell out of the 512 cells in the local memory of the PE is addressed by decoding a 9 bit address line A_0 through A_8. Decoding is done in two steps whose results are combined. The column-decoder is a 6 : 64 decoder whose inputs are the address lines A_3, A_4, A_5, A_6, A_7 and A_8, The 64 outputs of the column-decoder are given by the following Boolean equation:

$$X_i \leftarrow f(A_3, A_4, A_5, A_6, A_7, A_8) = \sum(i) \qquad 0 \leq i \leq 63$$

During any machine cycle, only one out of the 64 X_i lines will be high. These lines are used to select one out of 64 columns of the local memory of a PE.

The row-selector is a 8:1 Encoder controlled by three address lines A_0, A_1 and A_2. The 8 control output lines of the row encoder are given by the following Boolean equation:

$$Y_i \leftarrow f(A_0, A_1, A_2) = \sum(i) \qquad 0 \leq i \leq 7$$

During any machne cycle, only one out of the 8 Y_i lines will be selected. These lines are used to select one out of 8 rows of the local memory of the PE. The decoder was implemented in NOR logic so as to reduce the delay within the decoder.

2.4 Pads

Four different constant-height pads – V_{DD}, Ground, Input and I/O are built, the power and ground bus widths are calculated from the worst-case estimate of the power dissipation of a die and from a consideration of providing good supply voltages. The V_{DD} bus is placed as the outermost track, so that the frame of the chip is automatically generated.

2.4.1 V_{DD} and Ground Pads

These pads are easily designed and consist of a metal pad connected to the appropriate bus. The broken path is completed in Metal 1, and a large number of vias are used in the connections between Metal 1 and Metal 2. The corresponding layout of these two pads are shown in plate 13 and 14.

2.4.2 Input Pad

When designing the input pad, one has to take precaution that the gate connection of a MOS transistor has a very high input resistance (10^{12} to $10^{13}\Omega$). The voltage at which the oxide punctures and breaks down is about 40 to 100 volts. The voltage that can build up on a gate may be as high as 300 volts. A combination of resistance and diode clamps (electrostatic protection) are used to limit this potentially destructive voltage, the circuit is shown in figure 2.5. Clamp diodes D_1 and D_2 will turn on if the voltage at node X rises above V_{DD} or below 0 volt. 100Ω resistance is used by taking into consideration the speed of the chip, the corresponding layout is shown in plate 15.

2.4.3 I/O Pad

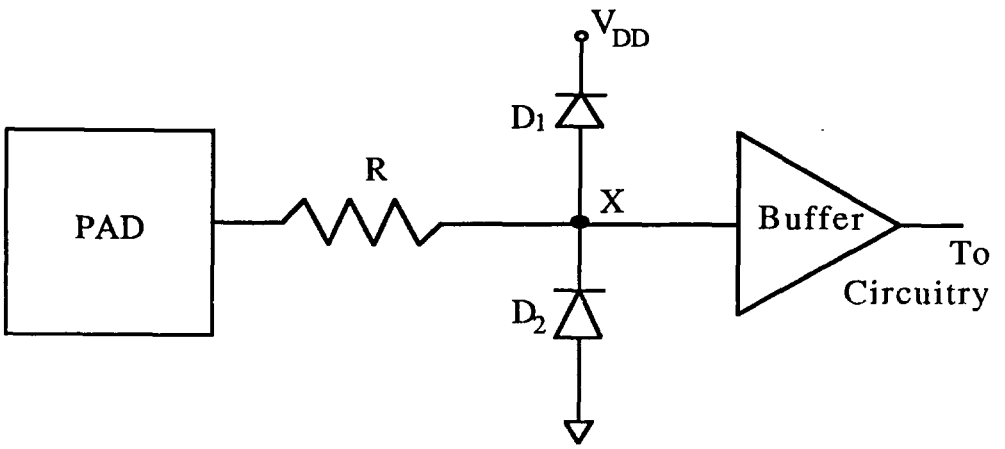


Figure 2.5 Input Protection Circuit

First and foremost, an output pad must have sufficient drive capability to achieve adequate rise and fall times into a given load capacitance. A output buffer which can drive 10 pf capacitance and meets the speed requirement ($10MHz$) is built. Combining a output buffer and input pad that previous designed, an I/O pad is created. The corresponding layout is shown in plate 16.

2.5 Chip Specifications

Number of PEs : 64

Feature size : 2μ

Technology : P–well CMOS process with two metal layers

Clock rate : 10 MHz

Power supply : 5 V

Chip size : 392 $\times$ 465 mil^2

Number of pins : 93

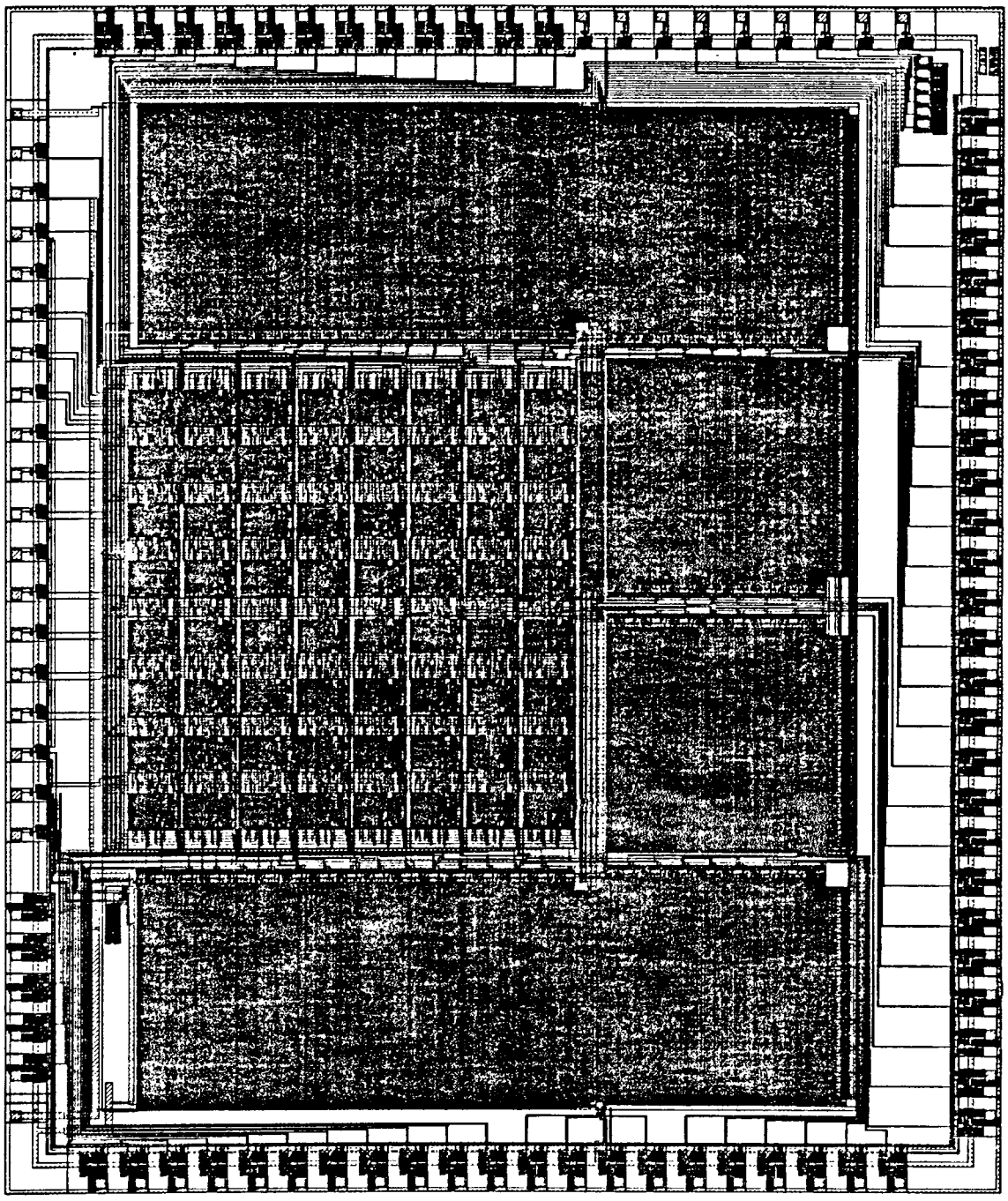


Plate 1. Layout of the 64–Processor Array Chip

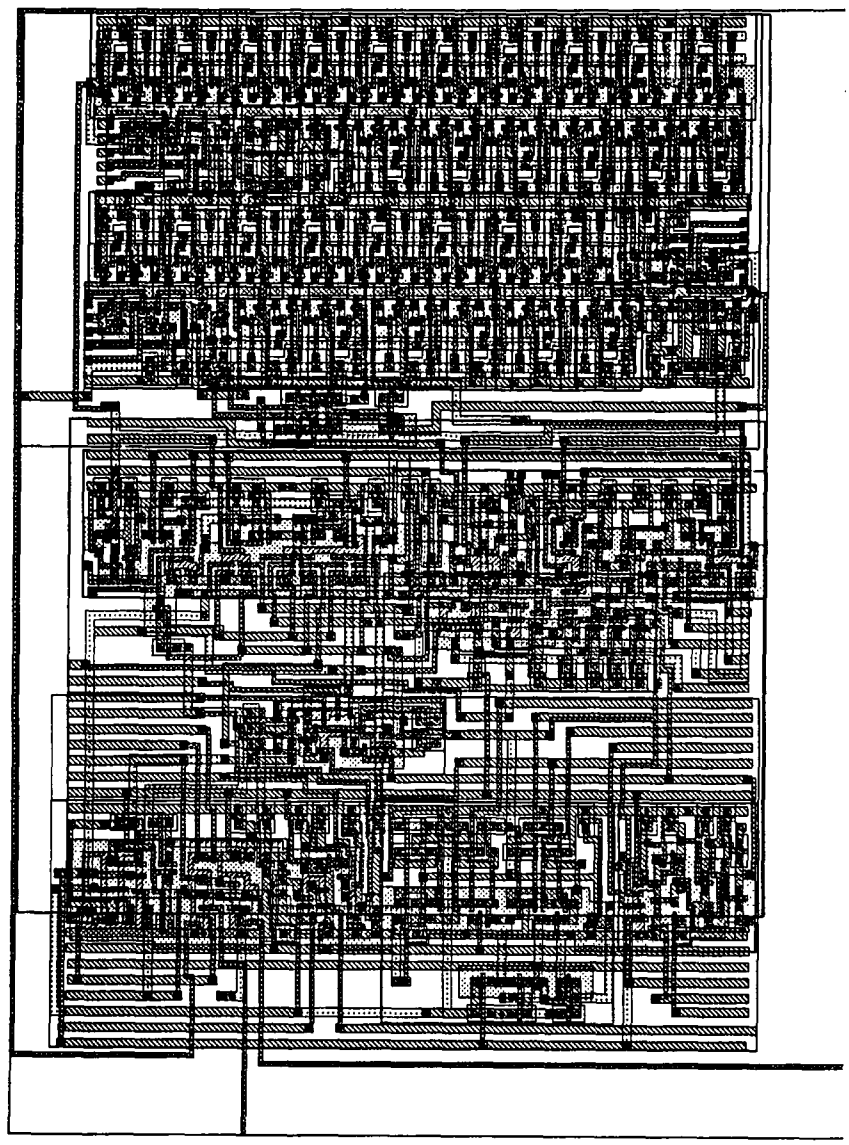


Plate 2. Layout of the Processing Element

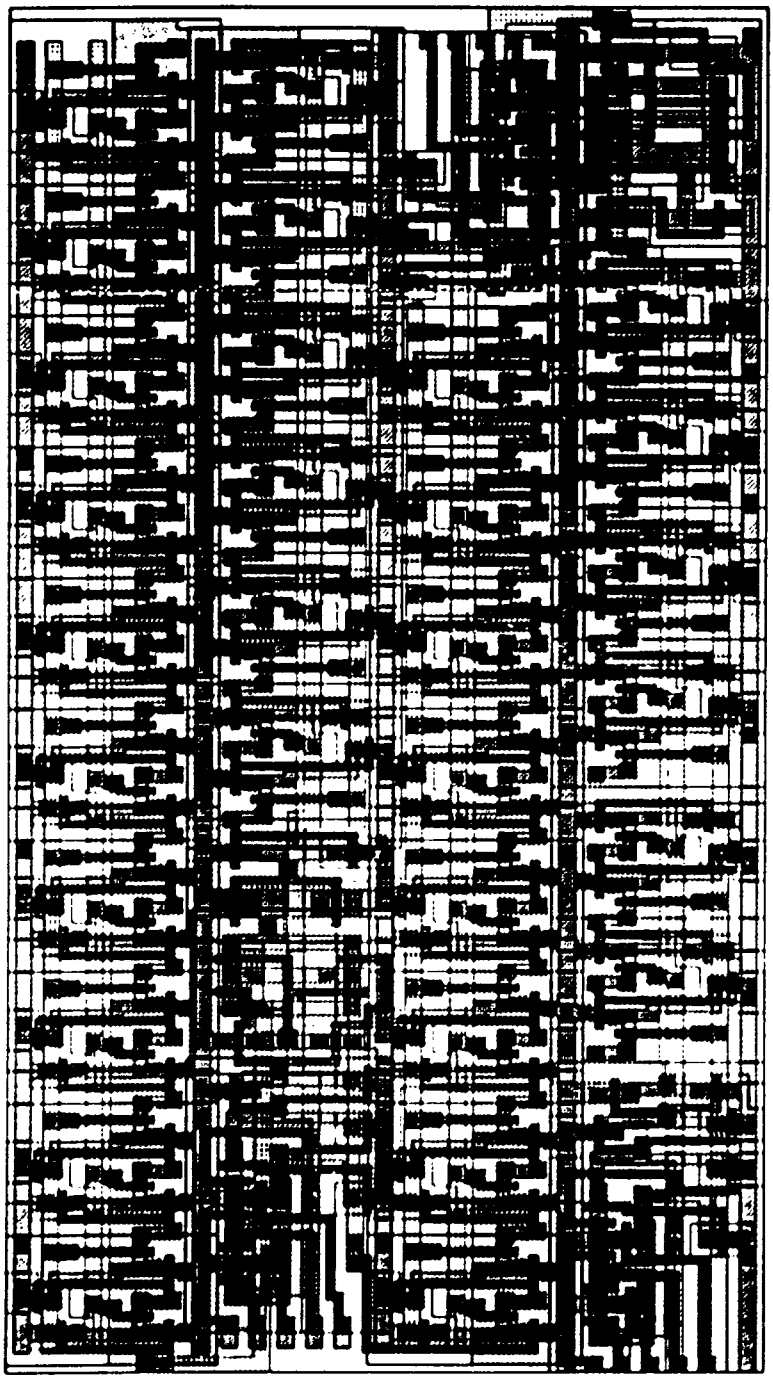


Plate 3. Layout of the Variable-length Shift Register

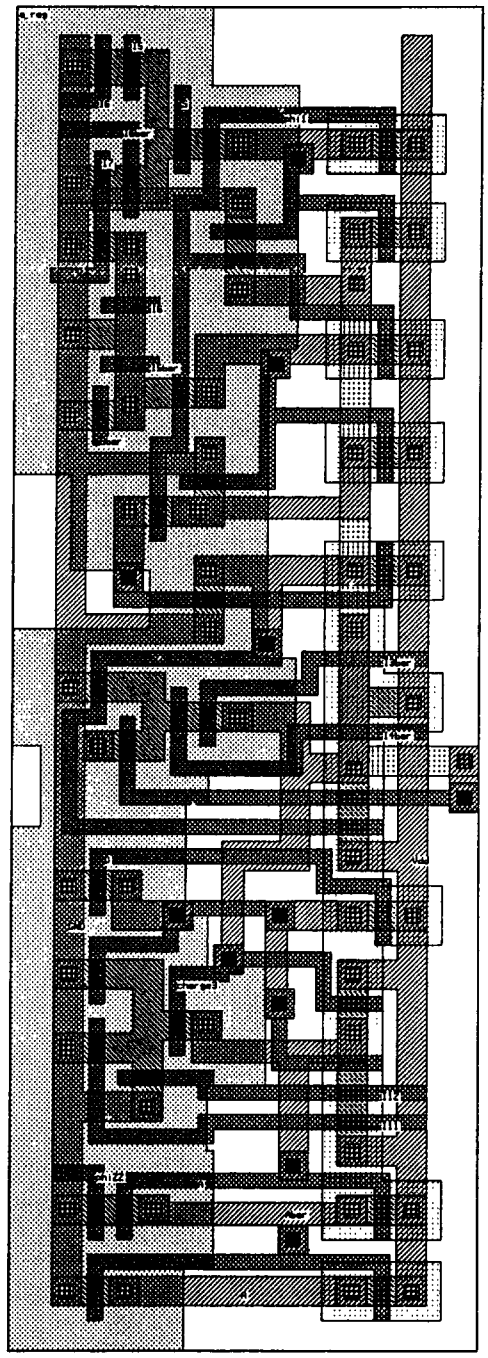


Plate 4. Layout of Register A

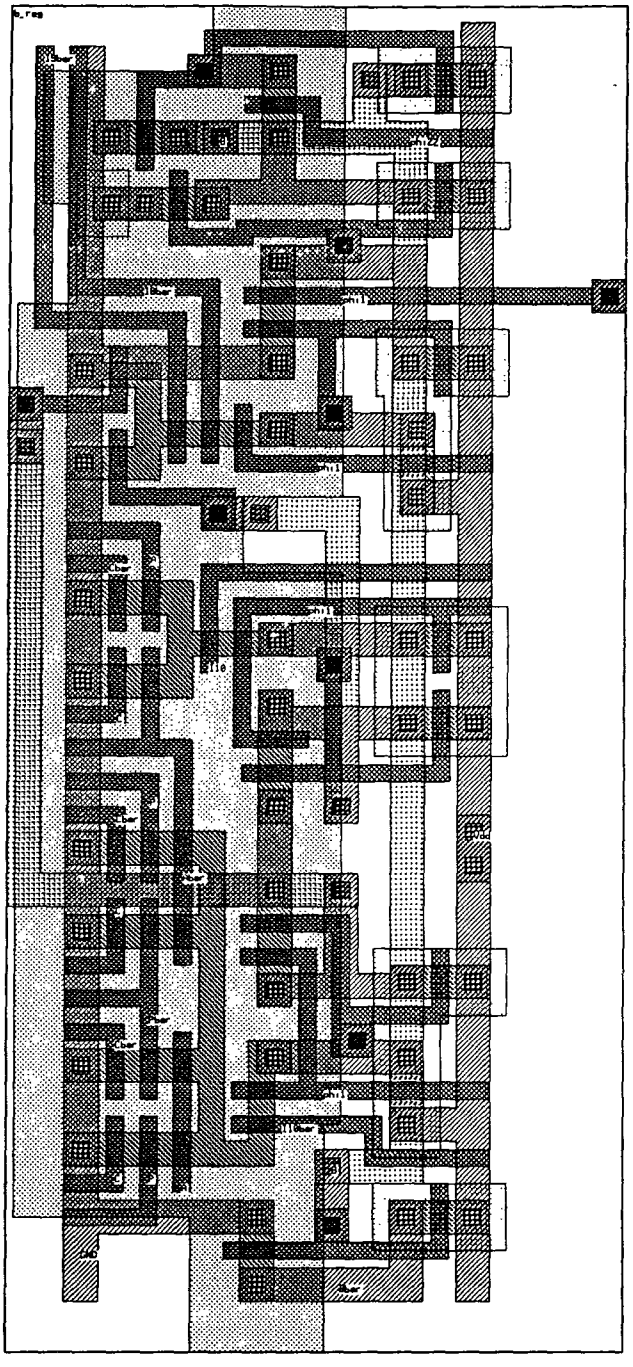


Plate 5. Layout of Register B

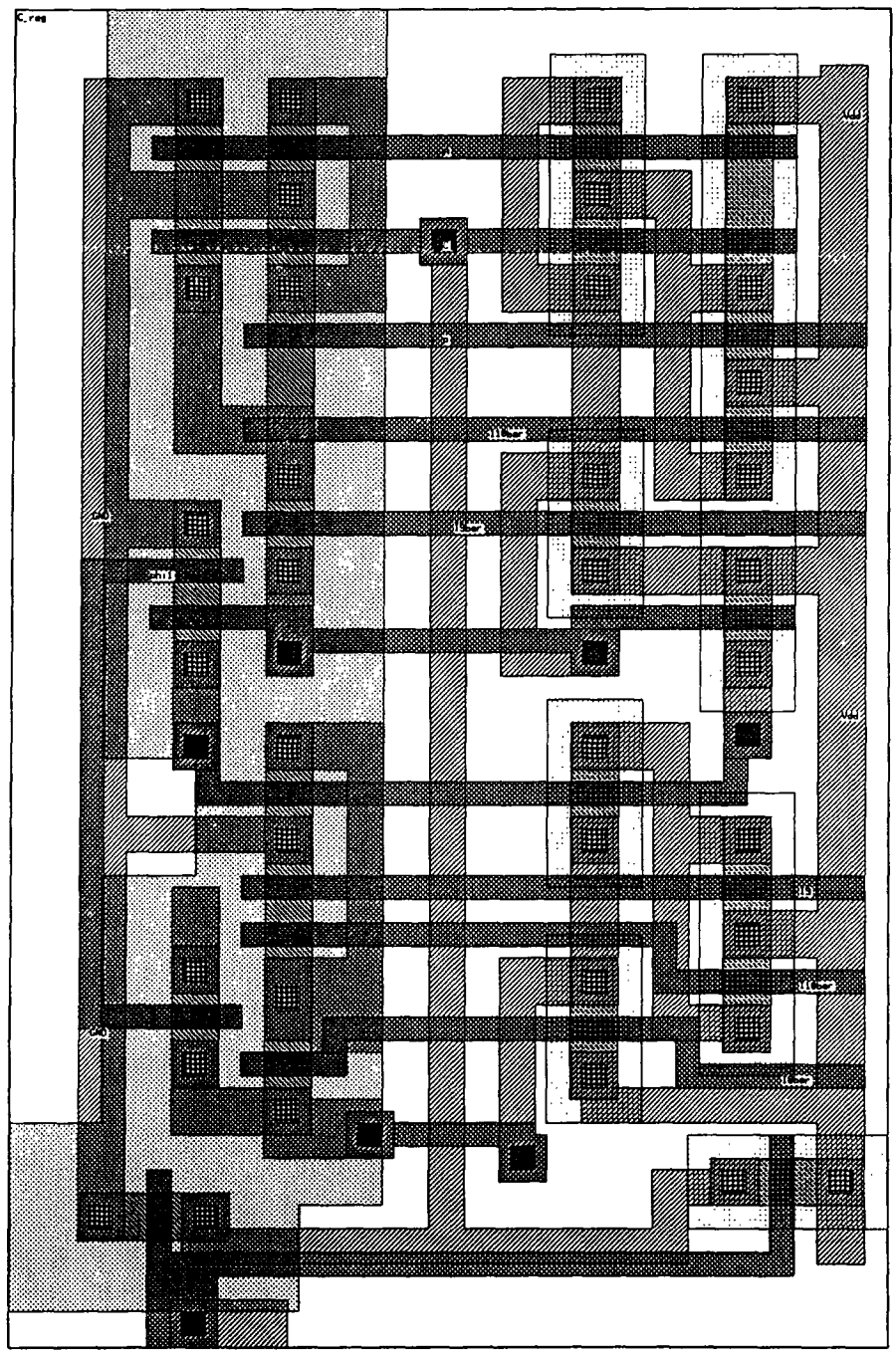


Plate 6. Layout of Register C

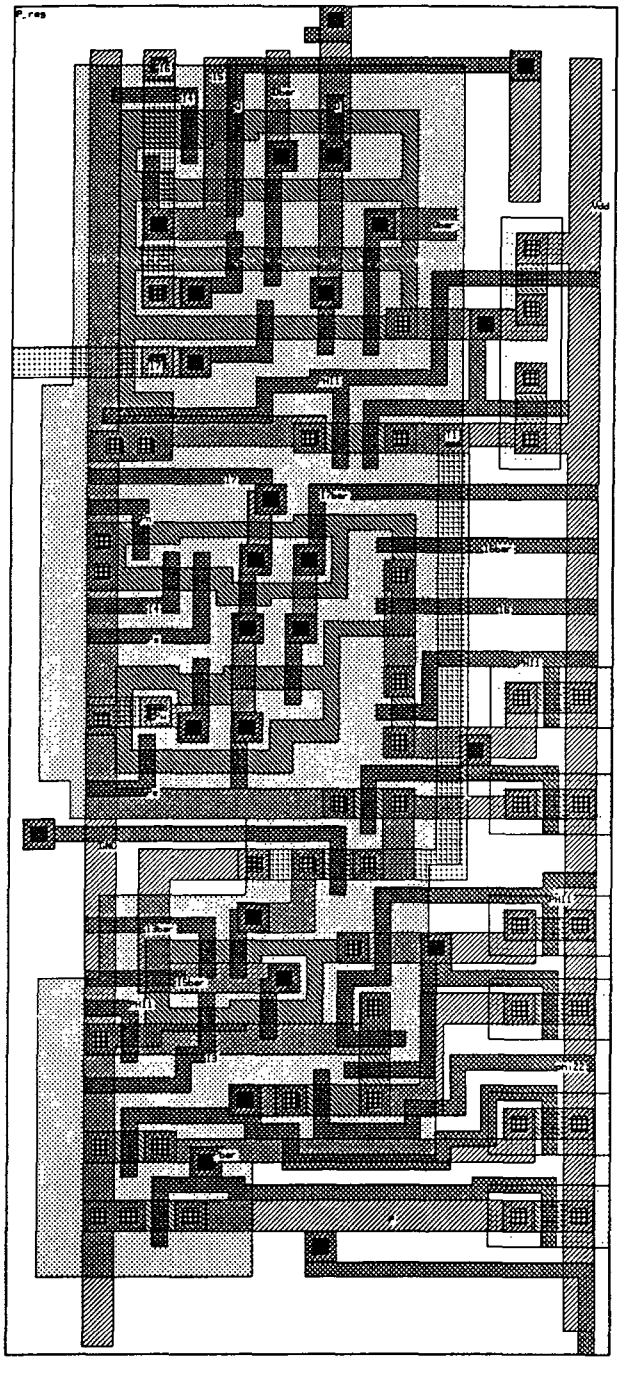


Plate 7. Layout of Register P

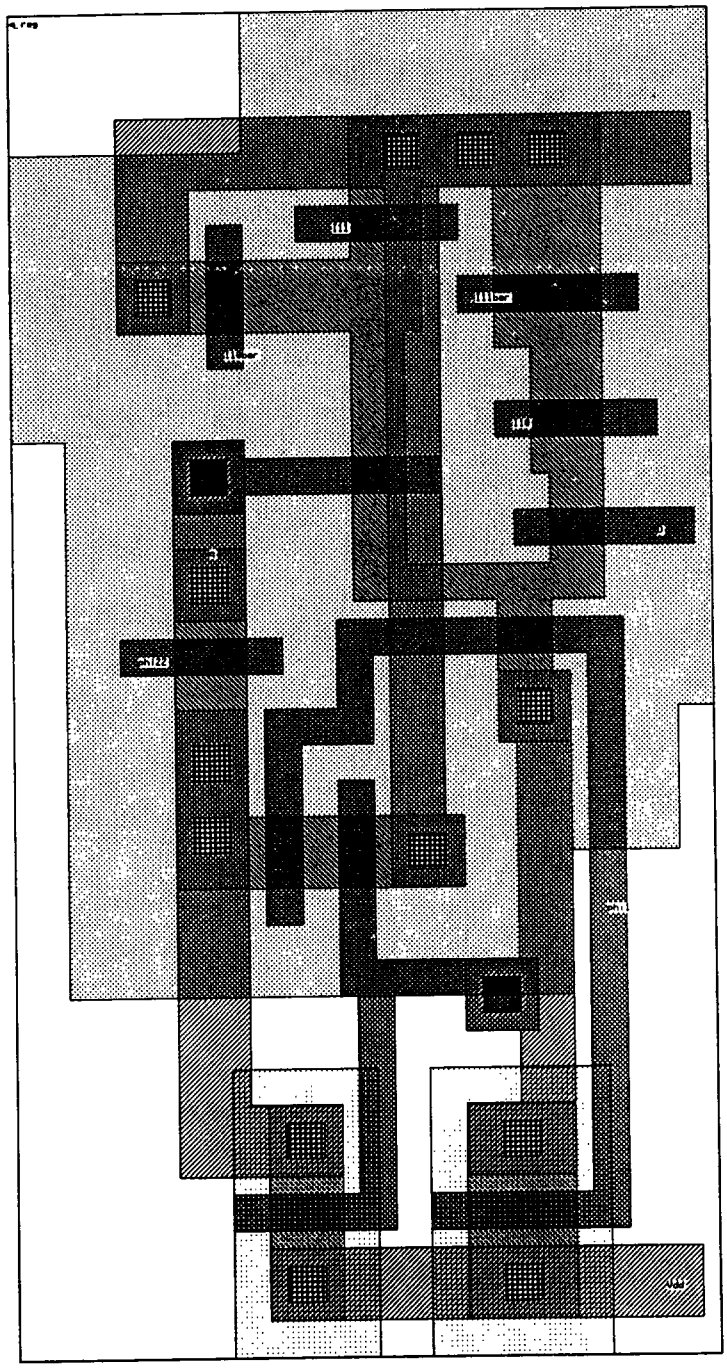


Plate 8. Layout of Register Q

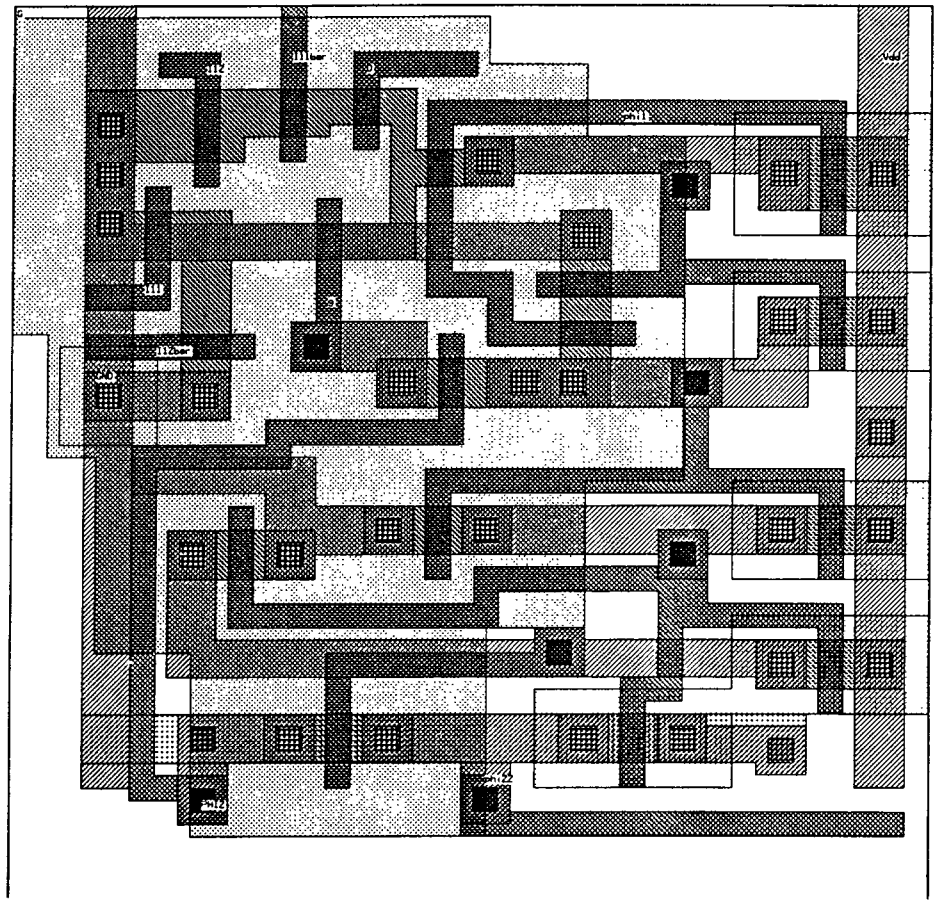


Plate 9. Layout of Register G

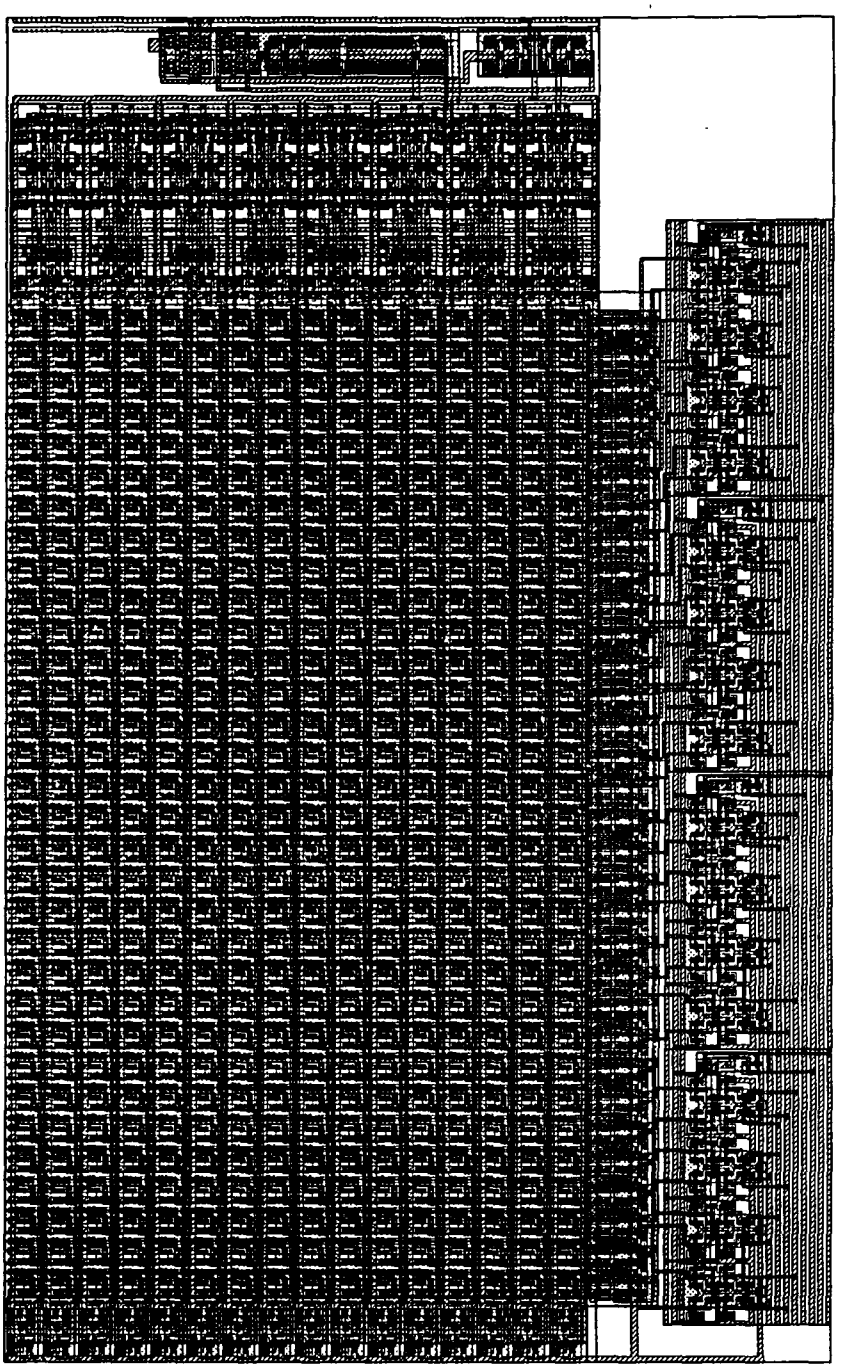


Plate 10. Layout of the 512-bit Memory

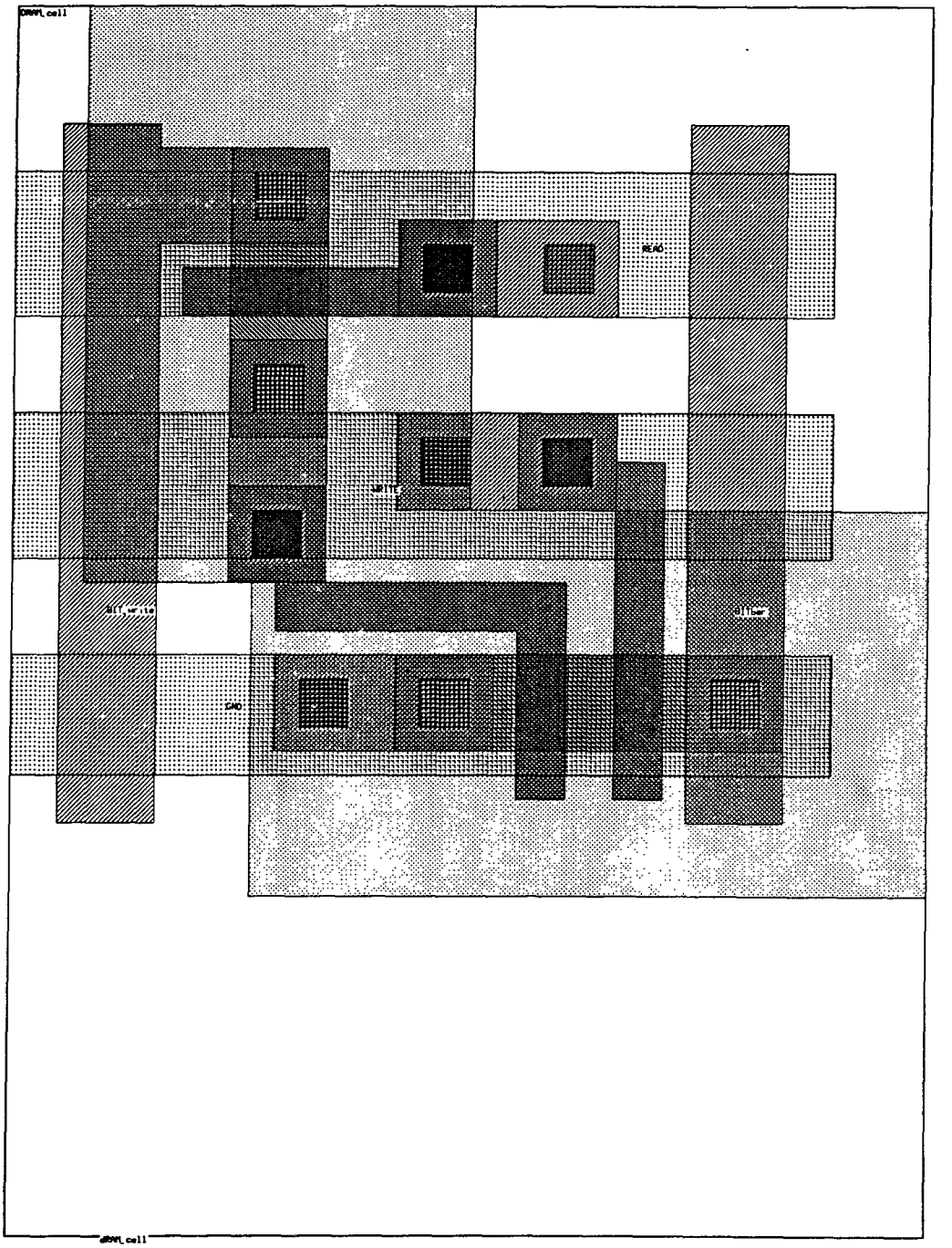


Plate 11. Layout of the Three-Transistor Dynamic Memory Cell

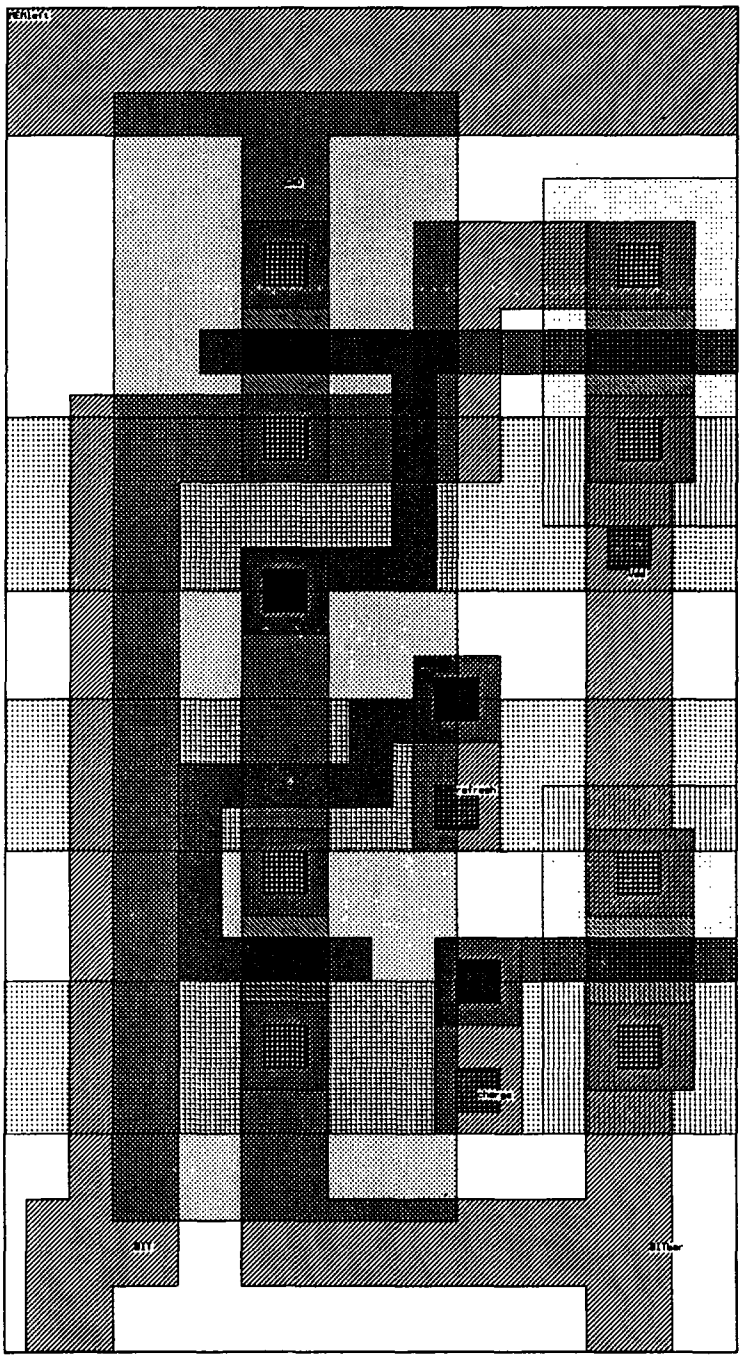


Plate 12. Layout of the Precharge/Refresh Circuit

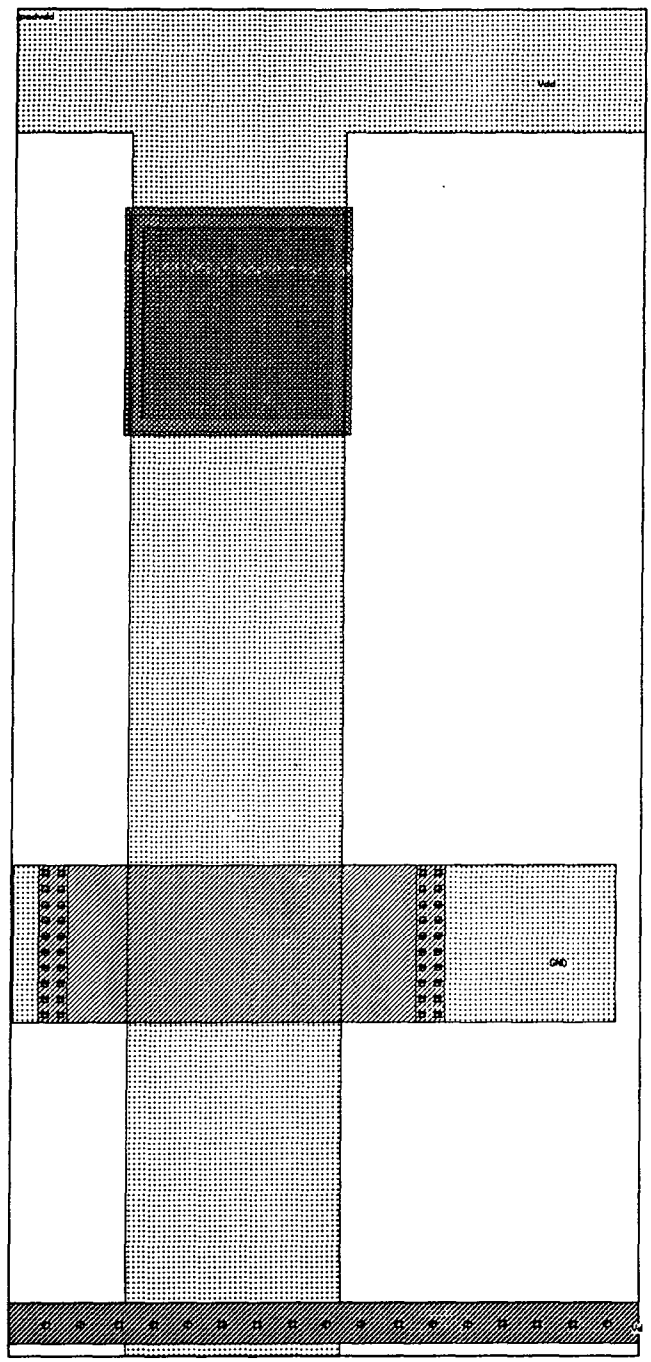


Plate 13. Layout of the V_{DD} Pad

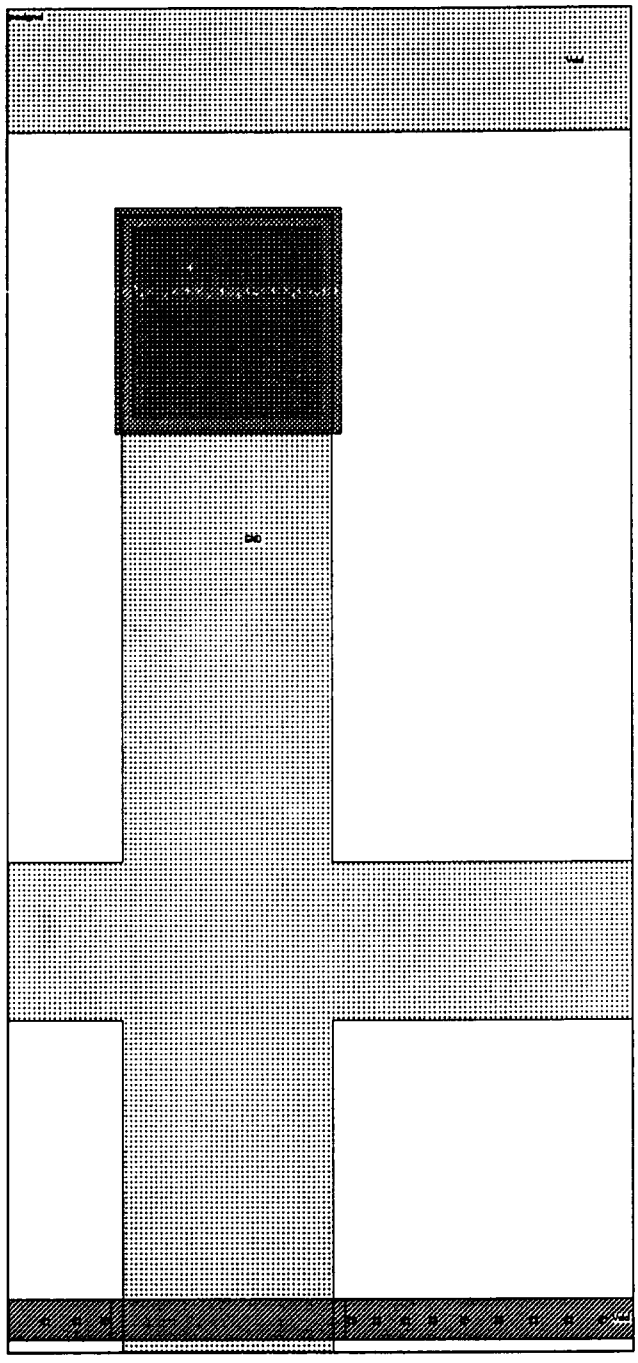


Plate 14. Layout of the Ground Pad

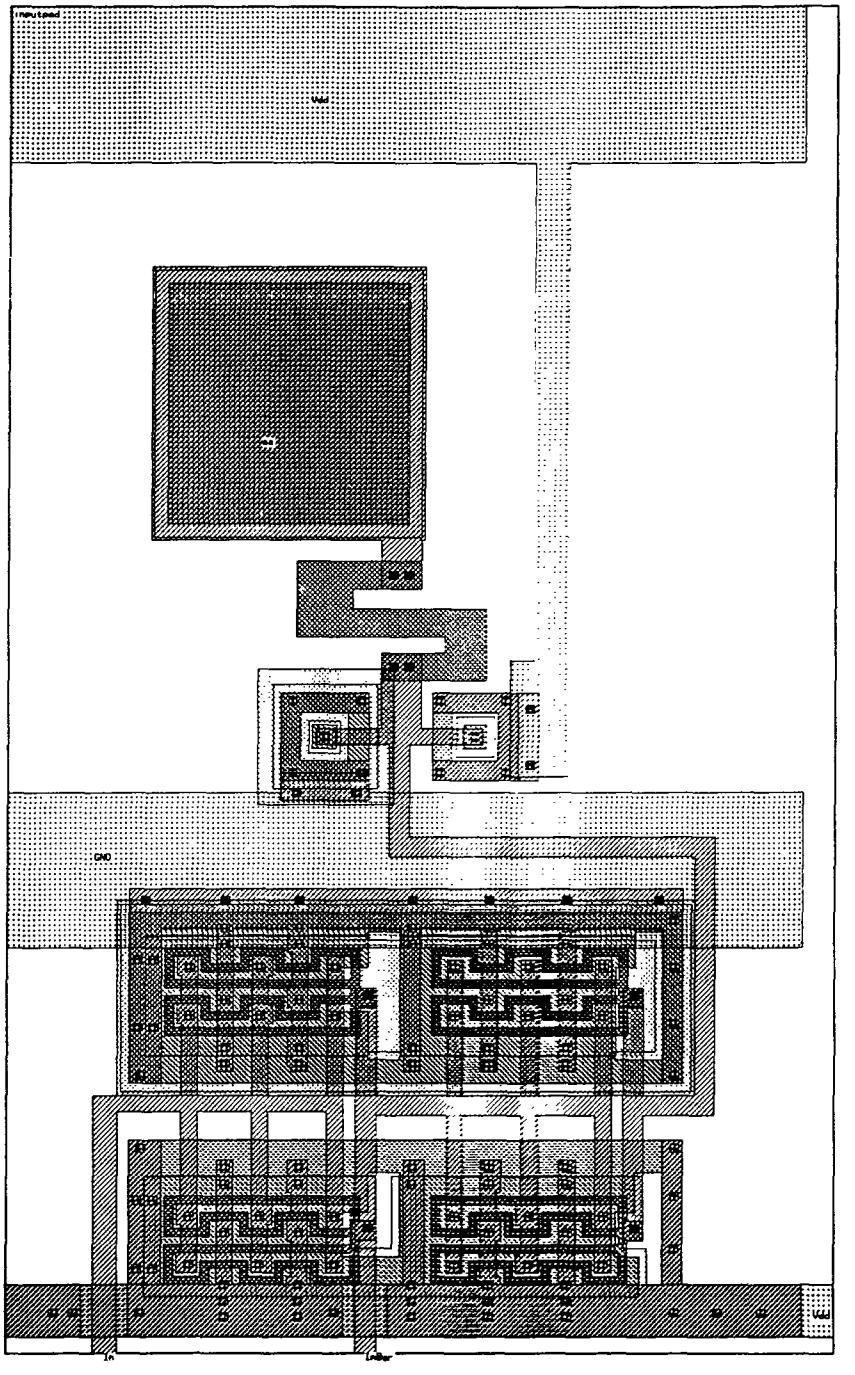


Plate 15. Layout of the Input Pad

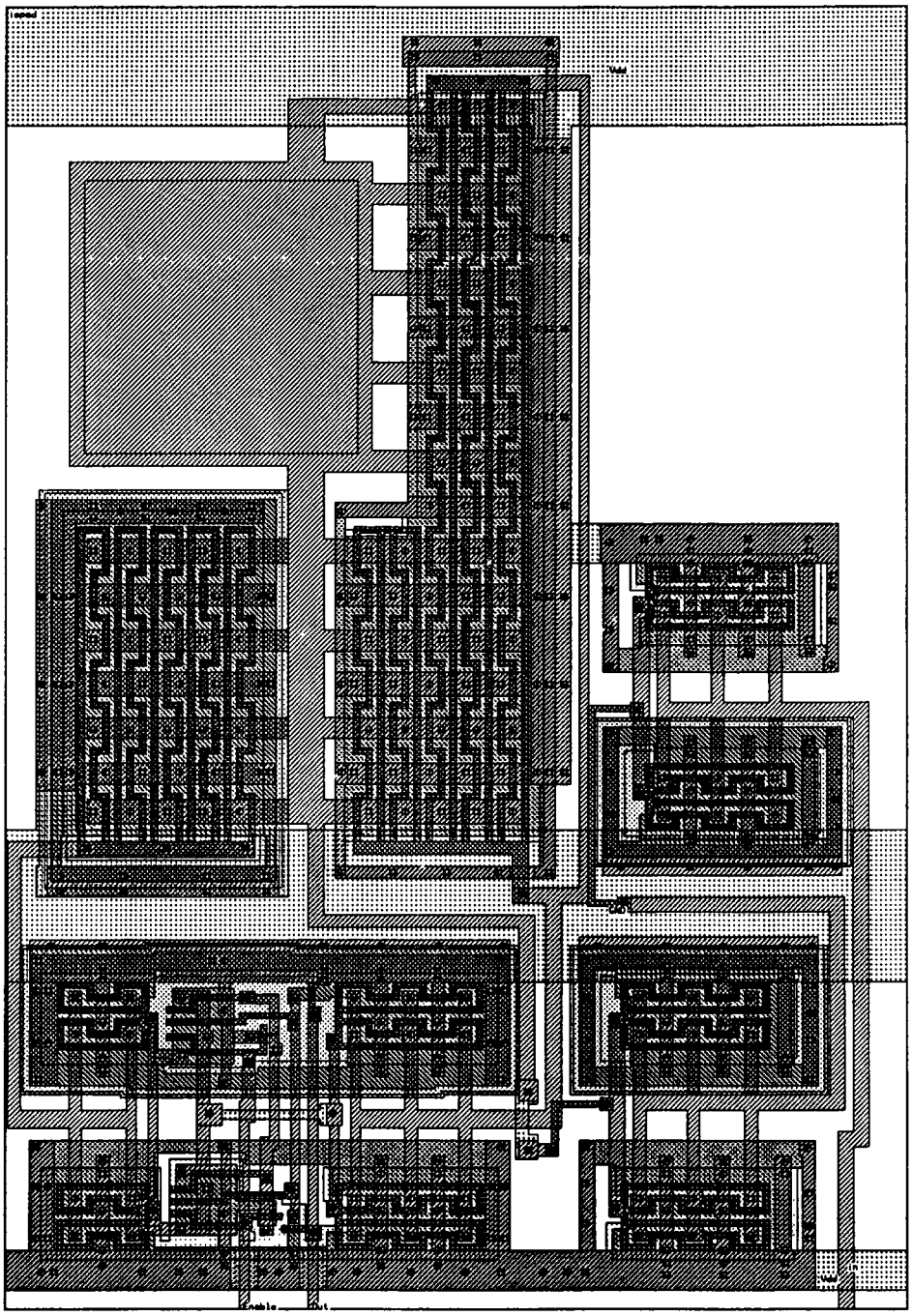


Plate 16. Layout of the Input/Output Pad

CHAPTER
THREE

The Testing of the 64–Processor Array

Errors, which may occur in circuit VLSI design, can be classified into three categories [Yo].

1. **Geometrical design rule error** Each technology has its own geometrical tolerances, such as minimum spacing between shapes, minimum internal width of a shape, and so on. A set of such rules is called the *geometrical design rule.* A violation of the geometrical design rule usually decreases the chip manufacturing yield, or results in non-functional VLSIs.

   Because of using the automatic layout tool – MAGIC, there will be no such errors, since the MAGIC will automatically check the error(s) which is (are) violating the design rules and feed it back to the designer immediately after he adds that (those) particular rectangular(s) (layer(s)) to his design.

2. **Logical error** *Logical errors* include mistakes in electrical connections between circuit elements and improper structure of circuit elements. These errors usually result in circuit malfunctions. In most cases these errors are fatal.

3. **Electrical performance error** The circuit should be designed to meet electrical characteristics requirements, such as power dissipation, timing

specifications and others. This kind of error is caused by inadequate device dimensions and disregarded parasitic effects, and results in VLSIs with logically correct function but non-satisfactory performance.

There is no error detection circuits built in this chip, the only way to test the 64-processor array chip is by using a test workstation named IMS-VS2000, the process of the testing will be described in section 3.2. A application board which uses the Macintosh II as the host controller is also built which will be discussed in section 3.3.

3.1 Test-Vector Generation

Test-vector generation can involve testing a design, or testing a device [Si]. In the former case we are not sure if the different parts of a design are mutually consistent, and in the latter case we are not sure if a device is functioning correctly. For both cases, test-vector generation involves generating a collection of tests each test specifies the values for some inputs and the values to observe at some outputs for these inputs. In this thesis we are focusing on testing the functionality of a device, and are not addressing testing other properties.

In this thesis we generate the test-vectors for the usage of the testing workstation IMS-VS2000. By dividing the task into four parts, and keeping all other three parts fixed when generating a particular groups of test-vectors; four group of test-vectors are created, namely the Arithmetic unit, the Logic unit, the Router unit and the Mask unit. The test-vectors of these four groups are shown in appendix A.

3.2 IMS-VS2000 Test Workstation

The IMS-VS2000 workstation is an interactive test and verification system for digital designs. It provides a structured environment for prototype analysis, and allows real-time comparison of the design's performance against expected data output.

The Logic Master is now controlled from a IBM PC over GPIB card by using either the Screen Interface or the Command Language Interface.

There are five basic steps to our test.

1. **Verify System Configuration:** Verify that the system contains the I/O card and pods to match the current test requirements.

2. **Name Channels and Create Groups:** Assign device signals to pod channels and organize channels into logical groups. Organizing channels into groups provides a short-hand method for setting test parameters for multiple channels at once.

3. **Wire the Verification Station for the Device:** Once channels are assigned to groups, it is easy to see which device pins should be wired to which pods.

4. **Specify Data Formats, Timing, and Voltage:** Set thresholds, display radix, format types, delay, and comparison times.

5. **Enter and Run the Pattern:** Enter the pattern sequences, press START SYSTEM (Shift F9), and compare the test results with your expected data.

By using the test-vectors described in section 3.1, the 64-processor array chip is tested, the corresponding input specifications and the test results are shown below:

Input specifications:

Clock pulse width: 30 ns.

Time between pulses of the two phase clock: 12 ns.

Clock period: 100 ns.

Clock waveform format: Return to zero.

Input data waveform format: Not return to zero.

Logic high drive: 3.5 $volts$.

Logic low drive: 1.0 $volts$.

Power supply: 5 $volts$ (respect to ground).

Maximum current supply: $250mA$.

System enable at start.

Test results

Average current needed: $150mA$.

Power Consumption: $750mW$.

3.3 Application Board

In order to program the 64-processor array chip from a host controller (in this thesis, the host controller is a Macintosh II), we use a Hurdler II interface card and some interface control circuits to connect the Macintosh II with the 64-processor array chip.

The Hurdler II interface card as described in section 1.1.3, contains:

- Complete interrupt driven Nubus slave interface to peripheral devices.
- Three 16-bit counter/timers.
- Separate 12 bit, bi-directional, parallel interface brought to a Centronics compatible connector.

- Software driver that provides high level interface to card, accessible from any development language (in this thesis, we use C language).

We can get address lines from A_0 to A_{11} and data line from D_0 to D_7 from the Nubus of the Macintosh II vias the Hurdler card. By using A_8, A_9, A_{10} and A_{11} as the control pins and some interface circuitry, the application board is built, which is shown in figure 3.1.

There are five steps to program the board:

1. **Load instructions and data into the RAM:** The programmer can load both instructions and data into the instruction RAM and data RAM respectively. This is done by holding the control lines A_8, A_9, A_{10} and A_{11} to 1, 1, 1, 0 while the programmer keying in the data. When A_8, A_9, A_{10} and A_{11} are 1, 1, 1, 0, the chip select ($\overline{CS}$) of the data RAM and instruction RAM are activated, and the data on the data lines D_0 through D_7 will be loaded into the data RAM and instruction RAM.

2. **Clear the address counter:** After loading the data and instruction for the operation, we should clear the address counter, this is done by setting A_8, A_9, A_{10} and A_{11} to 1, 1, 1, 1.

3. **Start the operations:** We can start the operation of the 64-processor array chip after we cleared the address counter so that we can make sure that the operation is from the beginning. This is done by setting A_8, A_9, A_{10} and A_{11} to 1, 1, 0, 1.

4. **Stop the operation:** Whenever the instruction codes I_0 and I_1 are 1 and 0 during clock ϕ_1, the operation of the 64-processor array chip will stop, the instruction set of this chip is shown in table 2.1. The stop operation is done by stopping the address counter.

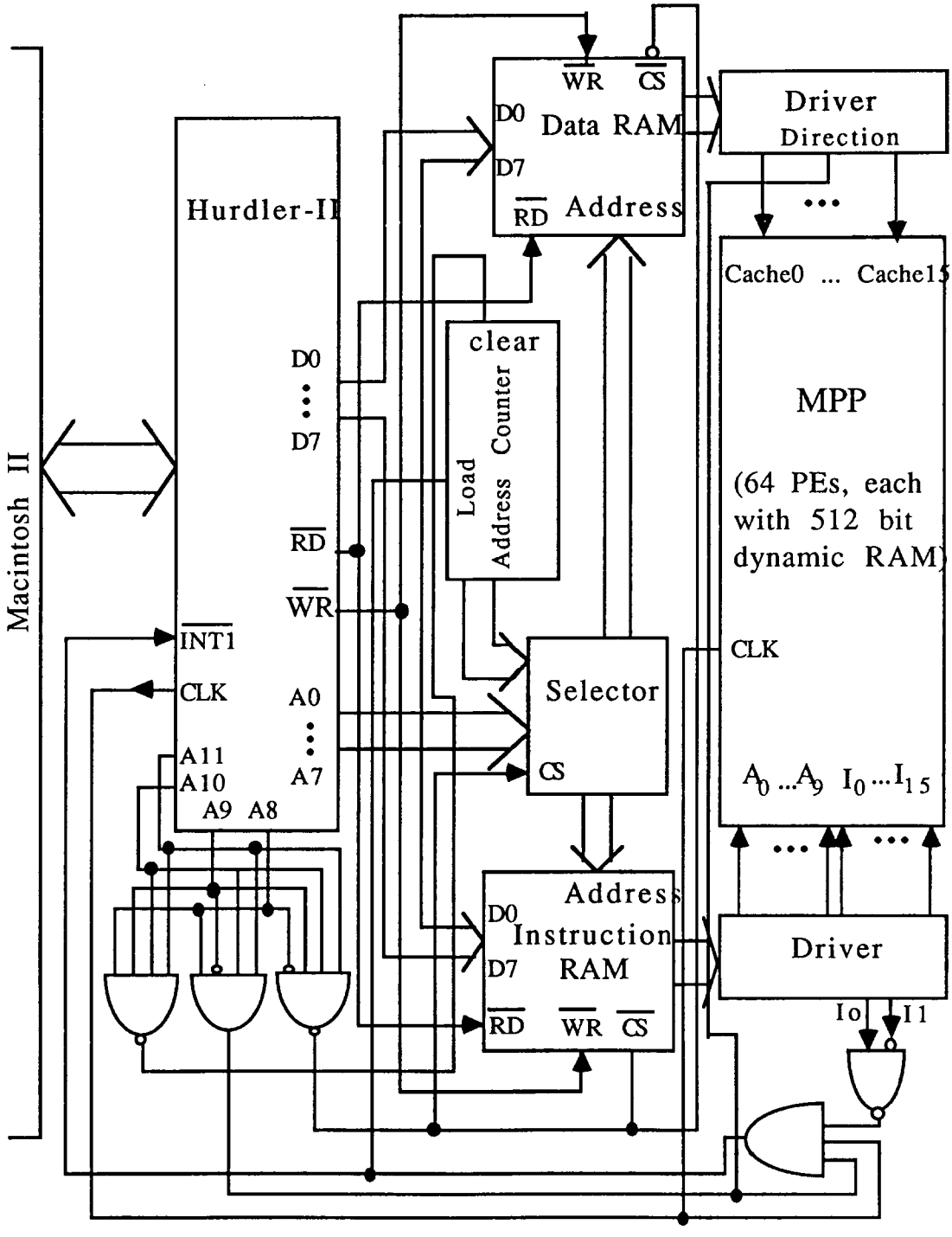


Figure 3.1 The Application Board of the 64-Processor Array Chip

5. **Load data from the data memory into the Macintosh II:** when the 64-processor array chip is stopped, it will simultaneously interrupt the Macintosh II by activating the $\overline{INT_1}$ of the Macintosh II. After the Macintosh receive the interrupt signal, it will load data from the data memory by programming the control lines A_8, A_9, A_{10} and A_{11} to 1, 1, 1, 0 and get data from D_0 through D_7.

By getting the clock from the Macintosh II, the application board is operating in synchronizely with the Macintosh II, this is very important else maybe some data will lost during the '*hand shaking*' between the Macintosh II and the 64-processor array chip.

CHAPTER
FOUR
Conclusion

The Massively Parallel Processor (MPP) chip was developed by Goodyear Aerospace Corporation for the NASA Goddard Space Flight Center, which is used to solve two-dimensional satellite image processing problems at very high speed. However, its large speed, large memory and I/O reformatting capability makes it also very useful for a variety of other problems where a large volume of data needs to be processed in parallel.

The 64-processor array chip has been developed for use in the MPP. This chip meets all the critical functional and timing specifications of the original chip on the MPP. In addition, new features have been incorporated into the chip to make it both easy to use and program in the MPP machine. The layout of the entire chip was done using the 2μ scalable CMOS design rules provided by MOSIS. It is easy to see that using the current available 1.2μ scalable CMOS technology, we can put more PEs into the chip although the design rule is a little different from that of the 2μ scalable CMOS technology. The PE itself can operate at much higher clock frequency (about 40 MHz), however, the speed of the whole chip will be limited to 10 MHz because of the access time of its Random Access Memory. The use of the precharge logic allows the replacement of the PMOS circuitry in the static CMOS logic by a single PMOS transistor, which leads to a less area layout.

There are about 160 thousand transistors in the 64-processor array chip, which makes the chip very difficult to test because of the large number of test-vectors needed to toggle every node from the external pins. A number of test-vectors has been developed for the testing, which are sufficient for the logical performance test, and have been used on the IMS-VS2000 test workstation as the testing data inputs. Furthermore, an prototyping application board is also developed for the 64-processor array chip which is different from the original MPP machine which used a VAX computer as the host controller.

One of the main constraints of the MPP is the amount of memory in the array, in the current MPP machine it is limited to 1024 bits. This is augmented by 512-bit memory and a 32-bit shift register per PE, however even this may not sufficient for some classes of problems where large number of variables are used. Another limitation of the system is that all the data must pass through the host controller (either VAX or Macintosh). This I/O limitation proves to be a big bottle-neck in the system and needs to be overcome. The memory per PE could be increased by having a off-chip shared memory block for all PE's, in addition to the on-chip local memories.

Despite these limitations, the MPP system is still an extremely powerful tool for satellite image processing as well as other applications.

APPENDIX

Test-Vectors of the 64-Processor Array Chip

$I_0 - I_2$	$I_3 - I_7$	$I_8 - I_{10}$	$I_{11} - I_{12}$	$I_{13} - I_{15}$	$A_0 - A_8$
NOP	$P \leftarrow 0$	*NOP*	*NOP*	*NOP*	–
$D \leftarrow P$	*NOP*	*NOP*	*NOP*	*NOP*	–
NOP	*NOP*	*Clear C*	*Load A*	*NOP*	–
NOP	*NOP*	*Half Add*	*NOP*	*NOP*	–
$D \leftarrow B$	*NOP*	*NOP*	*NOP*	*NOP*	–
$D \leftarrow C$	*NOP*	*Clear C*	*NOP*	$Cache \leftarrow D$	60
NOP	*NOP*	*NOP*	*NOP*	$Cache \leftarrow D$	61
$D \leftarrow P$	*NOP*	*NOP*	*NOP*	$RAM \leftarrow Cache$	60
NOP	*NOP*	*NOP*	*Load A*	$RAM \leftarrow Cache$	61
–	–	–	–	–	–
NOP	*NOP*	*Set C*	*NOP*	*NOP*	–
NOP	*NOP*	*Half Add*	*NOP*	*NOP*	–
$D \leftarrow B$	*NOP*	*NOP*	*NOP*	*NOP*	–
$D \leftarrow C$	*NOP*	*NOP*	*NOP*	$Cache \leftarrow D$	62
NOP	$P \leftarrow 0$	*NOP*	*NOP*	$Cache \leftarrow D$	63
$D \leftarrow P$	*NOP*	*Clear C*	*NOP*	$RAM \leftarrow Cache$	62
NOP	*NOP*	*NOP*	*Load A*	$RAM \leftarrow C$	63
–	–	–	–	–	–
NOP	$P \leftarrow 1$	*Set C*	*NOP*	*NOP*	–
$D \leftarrow P$	$P \leftarrow 0$	*NOP*	*NOP*	*NOP*	–
NOP	*NOP*	*NOP*	*Load A*	*NOP*	–
NOP	*NOP*	*Half Add*	*NOP*	*NOP*	–
$D \leftarrow B$	*NOP*	*NOP*	*NOP*	*NOP*	–
$D \leftarrow C$	*NOP*	*NOP*	*NOP*	$Cache \leftarrow D$	64
NOP	$P \leftarrow 0$	*NOP*	*NOP*	$Cache \leftarrow D$	65
$D \leftarrow P$	*NOP*	*Clear C*	*NOP*	$RAM \leftarrow Cache$	64
NOP	*NOP*	*NOP*	*Load A*	$RAM \leftarrow Cache$	65
–	–	–	–	–	–

$I_0 - I_2$	$I_3 - I_7$	$I_8 - I_{10}$	$I_{11} - I_{12}$	$I_{13} - I_{15}$	$A_0 - A_8$
NOP	$P \leftarrow 1$	NOP	NOP	NOP	–
$D \leftarrow P$	NOP	NOP	NOP	NOP	–
NOP	NOP	NOP	$Load\ A$	NOP	–
NOP	NOP	$Half\ Add$	NOP	NOP	–
$D \leftarrow B$	NOP	NOP	NOP	NOP	–
$D \leftarrow C$	NOP	NOP	NOP	$Cache \leftarrow D$	2
NOP	$P \leftarrow 0$	NOP	NOP	$Cache \leftarrow D$	3
$D \leftarrow P$	NOP	$Clear\ C$	NOP	$RAM \leftarrow Cache$	2
NOP	NOP	NOP	$Load\ A$	$RAM \leftarrow Cache$	3
–	–	–	–	–	–
NOP	$P \leftarrow 1$	NOP	NOP	NOP	–
$D \leftarrow P$	NOP	NOP	NOP	NOP	–
NOP	NOP	NOP	$Load\ A$	NOP	–
NOP	NOP	$Half\ Add$	NOP	NOP	–
$D \leftarrow B$	NOP	NOP	NOP	NOP	–
$D \leftarrow C$	NOP	NOP	NOP	$Cache \leftarrow D$	0
NOP	$P \leftarrow 0$	NOP	NOP	$Cache \leftarrow D$	1
$D \leftarrow P$	NOP	$Clear\ C$	NOP	$RAM \leftarrow Cache$	0
NOP	NOP	NOP	$Load\ A$	$RAM \leftarrow Cache$	1
–	–	–	–	–	–
NOP	$P \leftarrow 0$	NOP	NOP	NOP	–
$D \leftarrow P$	NOP	$Clear\ C$	NOP	NOP	–
NOP	NOP	NOP	$Load\ A$	NOP	–
NOP	NOP	$Full\ Add$	NOP	NOP	–
$D \leftarrow B$	NOP	NOP	NOP	NOP	–
$D \leftarrow C$	NOP	NOP	NOP	$Cache \leftarrow D$	4
NOP	$P \leftarrow 0$	NOP	NOP	$Cache \leftarrow D$	5
$D \leftarrow P$	NOP	$Clear\ C$	NOP	$RAM \leftarrow Cache$	4
NOP	NOP	NOP	$Load\ A$	$RAM \leftarrow Cache$	5
–	–	–	–	–	–
NOP	$P \leftarrow 1$	NOP	NOP	NOP	–
NOP	NOP	$Full\ Add$	NOP	NOP	–
$D \leftarrow B$	NOP	NOP	NOP	NOP	–
$D \leftarrow C$	NOP	NOP	NOP	$Cache \leftarrow D$	11
NOP	$P \leftarrow 0$	NOP	NOP	$Cache \leftarrow D$	12
$D \leftarrow P$	NOP	$Clear\ C$	NOP	$RAM \leftarrow Cache$	11
NOP	NOP	NOP	$Load\ A$	$RAM \leftarrow Cache$	12
–	–	–	–	–	–
NOP	$P \leftarrow 0$	$Set\ C$	NOP	NOP	–
NOP	NOP	$Full\ Add$	NOP	NOP	–
$D \leftarrow B$	NOP	NOP	NOP	NOP	–
$D \leftarrow C$	NOP	NOP	NOP	$Cache \leftarrow D$	21
NOP	$P \leftarrow 0$	NOP	NOP	$Cache \leftarrow D$	22
$D \leftarrow P$	NOP	$Clear\ C$	NOP	$RAM \leftarrow Cache$	21
NOP	NOP	NOP	$Load\ A$	$RAM \leftarrow Cache$	22

$I_0 - I_2$	$I_3 - I_7$	$I_8 - I_{10}$	$I_{11} - I_{12}$	$I_{13} - I_{15}$	$A_0 - A_8$
NOP	$P \leftarrow 1$	$Set\ C$	NOP	NOP	–
NOP	NOP	$Full\ Add$	NOP	NOP	–
$D \leftarrow B$	NOP	NOP	NOP	NOP	–
$D \leftarrow C$	NOP	NOP	NOP	$Cache \leftarrow D$	101
NOP	$P \leftarrow 0$	NOP	NOP	$Cache \leftarrow D$	102
$D \leftarrow P$	NOP	$Clear\ C$	NOP	$RAM \leftarrow Cache$	101
NOP	NOP	NOP	$Load\ A$	$RAM \leftarrow Cache$	102
–	–	–	–	–	–
NOP	$P \leftarrow 1$	NOP	NOP	NOP	–
$D \leftarrow P$	NOP	NOP	NOP	NOP	–
NOP	$P \leftarrow 0$	$ClearC$	$Load\ A$	NOP	–
NOP	NOP	$Full\ Add$	NOP	NOP	–
$D \leftarrow B$	NOP	NOP	NOP	NOP	–
$D \leftarrow C$	NOP	NOP	NOP	$Cache \leftarrow D$	0
NOP	NOP	NOP	NOP	$Cache \leftarrow D$	1
$D \leftarrow P$	NOP	$Clear\ C$	NOP	$RAM \leftarrow Cache$	0
NOP	NOP	NOP	$Load\ A$	$RAM \leftarrow Cache$	1
–	–	–	–	–	–
NOP	$P \leftarrow 1$	NOP	NOP	NOP	–
$D \leftarrow P$	NOP	NOP	NOP	NOP	–
NOP	NOP	NOP	$Load\ A$	NOP	–
NOP	NOP	$Full\ Add$	NOP	NOP	–
$D \leftarrow B$	NOP	NOP	NOP	NOP	–
$D \leftarrow C$	NOP	NOP	NOP	$Cache \leftarrow D$	4
NOP	NOP	NOP	NOP	$Cache \leftarrow D$	5
$D \leftarrow P$	NOP	$Clear\ C$	NOP	$RAM \leftarrow Cache$	4
NOP	NOP	NOP	$Load\ A$	$RAM \leftarrow Cache$	5
–	–	–	–	–	–
NOP	$P \leftarrow 1$	NOP	NOP	NOP	–
$D \leftarrow P$	NOP	NOP	NOP	NOP	–
NOP	NOP	$Set\ C$	$Load\ A$	NOP	–
	NOP	$Full\ Add$	NOP	NOP	–
$D \leftarrow B$	NOP	NOP	NOP	NOP	–
$D \leftarrow C$	NOP	NOP	NOP	$Cache \leftarrow D$	11
NOP	$P \leftarrow 0$	NOP	NOP	$Cache \leftarrow D$	12
$D \leftarrow P$	NOP	$Clear\ C$	NOP	$RAM \leftarrow Cache$	11
NOP	NOP	NOP	$Load\ A$	$RAM \leftarrow Cache$	12

$I_0 - I_2$	$I_3 - I_7$	$I_8 - I_{10}$	$I_{11} - I_{12}$	$I_{13} - I_{15}$	$A_0 - A_8$
NOP	$P \leftarrow 0$	NOP	NOP	NOP	–
$D \leftarrow P$	$P \leftarrow 1$	NOP	NOP	NOP	–
$D \leftarrow P$	NOP	NOP	NOP	$Cache \leftarrow D$	1
NOP	$P \leftarrow 0$	NOP	$Load\ Q$	$Cache \leftarrow D$	2
$D \leftarrow P$	NOP	NOP	NOP	$RAM \leftarrow Cache$	1
NOP	NOP	NOP	NOP	$RAM \leftarrow Cache$	2
–	–	–	–	–	–
NOP	$P \leftarrow D\bar{Q}$	NOP	NOP	NOP	–
$D \leftarrow P$	$P \leftarrow \bar{D}\bar{Q}$	NOP	NOP	NOP	–
$D \leftarrow P$	$P \leftarrow Q$	NOP	NOP	$Cache \leftarrow D$	3
$D \leftarrow P$	$P \leftarrow \bar{Q}$	$Full\ Add$	NOP	$Cache \leftarrow D$	4
$D \leftarrow P$	$P \leftarrow \bar{D}Q$	NOP	NOP	$Cache \leftarrow D$	5
$D \leftarrow P$	NOP	NOP	NOP	$RAM \leftarrow Cache$	3
NOP	$P \leftarrow Q$	NOP	NOP	$RAM \leftarrow Cache$	4
$D \leftarrow P$	NOP	NOP	NOP	$RAM \leftarrow Cache$	5
–	–	–	–	–	–
NOP	$P \leftarrow D \oplus Q$	NOP	NOP	NOP	–
$D \leftarrow P$	$P \leftarrow \bar{Q}$	NOP	NOP	NOP	–
$D \leftarrow P$	$P \leftarrow \bar{D} + \bar{Q}$	NOP	NOP	$Cache \leftarrow D$	5
$D \leftarrow P$	$P \leftarrow DQ$	NOP	NOP	$Cache \leftarrow D$	6
$D \leftarrow P$	NOP	NOP	NOP	$Cache \leftarrow D$	7
NOP	NOP	NOP	NOP	$Cache \leftarrow D$	8
NOP	$P \leftarrow 1$	NOP	NOP	$RAM \leftarrow Cache$	5
$D \leftarrow P$	NOP	NOP	NOP	$RAM \leftarrow Cache$	6
NOP	$P \leftarrow 0$	NOP	$Load\ Q$	$RAM \leftarrow Cache$	7
$D \leftarrow P$	NOP	NOP	NOP	$RAM \leftarrow Cache$	8
–	–	–	–	–	–
NOP	$P \leftarrow \overline{D \oplus Q}$	NOP	NOP	NOP	–
$D \leftarrow P$	$P \leftarrow D + \bar{Q}$	NOP	NOP	NOP	–
$D \leftarrow P$	$P \leftarrow Q$	NOP	NOP	$Cache \leftarrow D$	9
$D \leftarrow P$	$P \leftarrow D + Q$	NOP	NOP	$Cache \leftarrow D$	10
$D \leftarrow P$	$P \leftarrow \bar{D} + Q$	NOP	NOP	$Cache \leftarrow D$	11
$D \leftarrow P$	NOP	NOP	NOP	$Cache \leftarrow D$	12
NOP	NOP	NOP	NOP	$RAM \leftarrow Cache$	9
NOP	NOP	NOP	NOP	$RAM \leftarrow Cache$	10
NOP	NOP	NOP	NOP	$RAM \leftarrow Cache$	11
NOP	NOP	NOP	NOP	$RAM \leftarrow Cache$	12

$I_0 - I_2$	$I_3 - I_7$	$I_8 - I_{10}$	$I_{11} - I_{12}$	$I_{13} - I_{15}$	$A_0 - A_8$
NOP	$P \leftarrow D$	*Clear C*	*NOP*	*NOP*	–
$D \leftarrow P$	*NOP*	*NOP*	*NOP*	*NOP*	–
NOP	$P \leftarrow 1$	*Clear C*	*Load A*	*NOP*	–
NOP	*NOP*	*Full Add*	*NOP*	*NOP*	–
NOP	*Set Length* = 2[1]	*Set C*	*NOP*	*NOP*	–
NOP	*Set Length* = 2	*Set C*	*NOP*	*NOP*	–
NOP	*Set Length* = 2	*Half Add*	*NOP*	*NOP*	–
$D \leftarrow C$	*NOP*	*NOP*	*NOP*	*NOP*	–
NOP	*NOP*	*NOP*	*NOP*	*Cache* $\leftarrow D$	1
NOP	*NOP*	*NOP*	*NOP*	*RAM* $\leftarrow$ *Cache*	1
–	–	–	–	–	–
NOP	*NOP*	*NOP*	*NOP*	*Cache* $\leftarrow$ *RAM*	50
NOP	*NOP*	*NOP*	*NOP*	$D \leftarrow$ *Cache*	50
NOP	$P \leftarrow 0$	*NOP*	*NOP*	*NOP*	–
NOP	$P \leftarrow P_n$	*NOP*	*NOP*	*NOP*	–
$D \leftarrow P$	$P \leftarrow P_s$	*NOP*	*NOP*	*NOP*	–
$D \leftarrow P$	$P \leftarrow P_w$	*NOP*	*NOP*	*Cache* $\leftarrow D$	0
$D \leftarrow P$	$P \leftarrow P_e$	*NOP*	*NOP*	*Cache* $\leftarrow D$	1
$D \leftarrow P$	*NOP*	*NOP*	*NOP*	*Cache* $\leftarrow D$	2
NOP	*NOP*	*NOP*	*NOP*	*Cache* $\leftarrow D$	3
NOP	*NOP*	*NOP*	*NOP*	*RAM* $\leftarrow$ *Cache*	0
NOP	$P \leftarrow 0$	*NOP*	*NOP*	*RAM* $\leftarrow$ *Cache*	1
$D \leftarrow P$	*NOP*	*NOP*	*NOP*	*RAM* $\leftarrow$ *Cache*	2
NOP	*NOP*	*NOP*	*NOP*	*RAM* $\leftarrow$ *Cache*	3
–	–	–	–	–	–
NOP	$P \leftarrow 1$	*NOP*	*NOP*	*NOP*	–
NOP	*NOP*	*NOP*	*NOP*	*Cache* $\leftarrow$ *RAM*	40
NOP	*NOP*	*NOP*	*NOP*	$D \leftarrow$ *Cache*	40
NOP	*NOP*	*NOP*	*Load G*	*NOP*	–
$D \leftarrow P$	*NOP*	*NOP*	*NOP*	*NOP*	–
NOP	*NOP*	*NOP*	*NOP*	*Cache* $\leftarrow D$	40
NOP	*NOP*	*NOP*	*NOP*	*RAM* $\leftarrow$ *Cache*	40
NOP	*NOP*	*NOP*	*Load A*	*RAM* $\leftarrow$ *Cache*	5
–	–	–	–	–	–
NOP	*NOP*	*NOP*	*NOP*	*Cache* $\leftarrow$ *RAM*	41
NOP	*NOP*	*NOP*	*NOP*	$D \leftarrow$ *Cache*	41
NOP	*NOP*	*NOP*	*Load G*	*NOP*	–
$D \leftarrow P$	*NOP*	*NOP*	*NOP*	*NOP*	–
NOP	*NOP*	*NOP*	*NOP*	*Cache* $\leftarrow D$	41
NOP	*NOP*	*NOP*	*NOP*	*RAM* $\leftarrow$ *Cache*	41

[1] Change *Set Length* = 2 to 4, 8, 16 and 32 to check the variable-length shift register.

$I_0 - I_2$	$I_3 - I_7$	$I_8 - I_{10}$	$I_{11} - I_{12}$	$I_{13} - I_{15}$	$A_0 - A_8$
NOP	$P \leftarrow D$	$Clear\ C$	NOP	NOP	–
$D \leftarrow P$	NOP	NOP	NOP	NOP	–
NOP	$P \leftarrow 1$	$Clear\ C$	$Load\ A$	NOP	–
NOP	NOP	$Full\ Add$	NOP	NOP	–
NOP	$Set\ Length = 2^1$	$Set\ C$	NOP	NOP	–
NOP	$Set\ Length = 2$	$Set\ C$	NOP	NOP	–
NOP	$Set\ Length = 2$	$Half\ Add$	NOP	NOP	–
$D \leftarrow C$	NOP	NOP	NOP	NOP	–
NOP	NOP	NOP	NOP	$Cache \leftarrow D$	1
NOP	NOP	NOP	NOP	$RAM \leftarrow Cache$	1
–	–	–	–	–	–
NOP	NOP	NOP	NOP	$Cache \leftarrow RAM$	50
NOP	NOP	NOP	NOP	$D \leftarrow Cache$	50
NOP	$P \leftarrow 0$	NOP	NOP	NOP	–
NOP	$P \leftarrow P_n$	NOP	NOP	NOP	–
$D \leftarrow P$	$P \leftarrow P_s$	NOP	NOP	NOP	–
$D \leftarrow P$	$P \leftarrow P_w$	NOP	NOP	$Cache \leftarrow D$	0
$D \leftarrow P$	$P \leftarrow P_e$	NOP	NOP	$Cache \leftarrow D$	1
$D \leftarrow P$	NOP	NOP	NOP	$Cache \leftarrow D$	2
NOP	NOP	NOP	NOP	$Cache \leftarrow D$	3
NOP	NOP	NOP	NOP	$RAM \leftarrow Cache$	0
NOP	$P \leftarrow 0$	NOP	NOP	$RAM \leftarrow Cache$	1
$D \leftarrow P$	NOP	NOP	NOP	$RAM \leftarrow Cache$	2
NOP	NOP	NOP	NOP	$RAM \leftarrow Cache$	3
–	–	–	–	–	–
NOP	$P \leftarrow 1$	NOP	NOP	NOP	–
NOP	NOP	NOP	NOP	$Cache \leftarrow RAM$	40
NOP	NOP	NOP	NOP	$D \leftarrow Cache$	40
NOP	NOP	NOP	$Load\ G$	NOP	–
$D \leftarrow P$	NOP	NOP	NOP	NOP	–
NOP	NOP	NOP	NOP	$Cache \leftarrow D$	40
NOP	NOP	NOP	NOP	$RAM \leftarrow Cache$	40
NOP	NOP	NOP	$Load\ A$	$RAM \leftarrow Cache$	5
–	–	–	–	–	–
NOP	NOP	NOP	NOP	$Cache \leftarrow RAM$	41
NOP	NOP	NOP	NOP	$D \leftarrow Cache$	41
NOP	NOP	NOP	$Load\ G$	NOP	–
$D \leftarrow P$	NOP	NOP	NOP	NOP	–
NOP	NOP	NOP	NOP	$Cache \leftarrow D$	41
NOP	NOP	NOP	NOP	$RAM \leftarrow Cache$	41

Reference

[Al] Altnether, J., "*A Look at CMOS Dynamic Memory* ",BYTE, Sept. 1983.

[Ba1] Batcher,K.E., "*MPP – A Massively Parallel Processor* ", International Conference on Parallel Processing, August 1979.

[Ba2] Batcher,K.E., "*Design of a Massively Parallel Processor* ", IEEE Transactions on Computers, vol. C–29, No. 9, Sept. 1980, pp. 836–840.

[Bu] Burkley,J.T., "*MPP VLSI Multi–Processor Intergrated Circuit Design* ", Proceedings of the 1982 International Conference on Parallel Processing, August 1983.

[Cl] Cluley, J. C., "*Minicomputer and Microproessor Interface* ", Crane Russak Company, Inc., 1982.

[De] Denyer,P. and Renshaw,D. "*VLSI Signal Processing : A Bit–Serial Approach* ,"Addison–Wesley Publishing Co., 1985.

[Gl] Glasser,L.W. and Dobberpuhl,D.W. "*The Design and Analysis of VLSI Circuits* ," Addison Wesley Systems Series, 1985.

[Hw] Hwang,K. "*Computer Arithmetic, Principles, Architecture and Design* ," John Wiley and Sons, 1979.

[Kung] Kung,H.T.,"*Why Systolic Architectures?* ", IEEE Comp., Jan 1982, pp 37–46.

[Me] Mead,C. and Conway,L. "*Introduction to VLSI Systems* ," Addison–Wesley Series in Computer Science, 1980.

[Mo] Moore, W., McCabe, A. and Urquhart, R. "*Systolic Arrays*", Adam Hilger, 1986.

[Mu] Mukherjee, A. "*Introduction to nMOS and CMOS VLSI System Design*", Prentice-Hall, 1986.

[Neuman] Neuman,M.J. "*Is WSI too big?* ", VLSI–85, Tokyo Japan, Aug.,1985. pp97-106.

[Newkirk] Newkirk, J. A. and Mathews, R. "*The VLSI Designer's Library* ", Addison-Wesley, 1983.

[Puncknell] Puncknell,D.A. "*Basic VLSI Design, Principles & Applications* ," Prentice-Hall of Australia Pty Ltd, 1985.

[Si] Singh,N. "*An Arificial Intelligence Approach to Test Generation*," Kluwer Academic Publishers, 1987.

[Ul] Ullman,J.D. "*Computational Aspects of VLSI* ," Computer Science Press, 1984.

[We] Weste,N. and Eshraghian,K. "*Principles of CMOS VLSI Design – A Systems Perspective* ," Addison–Wesley VLSI Systems Series, 1985.

[Y0] Yoshida,K "*Layout Verification*," Elsevier Science Publications. T. Ohtsuki ed.,1986.