NASA-CR-202424

CMOS VLSI LAYOUT AND VERIFICATION OF A SIMD COMPUTER

GRANT NAG5-2509

By

Jianqing Zheng

A Thesis

Presented to the Faculty of

Bucknell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Electrical Engineering

Approved: Maurice F. Aburdene

Adviser

Maurice F. Aburdene

Department Chairperson

May 12, 1996

Acknowledgments

First I would like to thank my advisor, Professor Maurice F. Aburdene, for his guidance, patience and encouragement which provided me with the necessary help to complete this thesis. I acknowledge my debt to Professors Richard J. Kozick, Paul H. Dietz and Daniel Hyde, who help me much during my two-year graduate study at Bucknell University. The work of John Dorband at NASA Goddard Space Flight Center, Maurice Aburdene, Kamal Khouri, Jason Piatt and Eric Passmore at Bucknell University is deeply appreciated. A word of thanks also goes to Lian Wang of Computer Center at Bucknell University for her help and enthusiasm.

My love and thanks go out to both my wife Jie Li and my daughter Ting Ting for their love and sacrifices.

This material is based upon work partially supported by the National Aeronautics and Space Administration under Grand NO. NAG-5-2509. Any opinions, findings, and conclusions or recommendations in this material are those of the author and do not necessarily reflect the view of the National Aeronautics and Space Administration.

Contents

List of Figures

Abstract

A CMOS VLSI layout and verification of a 3x3 processor parallel computer has been completed. The layout was done using the MAGIC tool and the verification using HSPICE. Suggestions for expanding the computer into a million processor network are presented. Many problems that might be encountered when implementing a massively parallel computer are discussed.

Chapter 1

Introduction

A recent trend in the design of massively parallel computers has shifted to the design of multiple instruction-multiple data (MIMD) machines. However, Single Instruction-Multiple Data (SIMD) machines are still being actively developed since SIMD has several advantages including simple structure, identical control instructions, scalability, high Floating Point Operation Per Second (FLOPS) and low cost [1,2,3,4]. Algorithms and applications based on SIMD machines have been used in image processing, signal processing, numerical analysis and other areas [5,6,7,8].

Our goal is to investigate the design of a SIMD computer with more than one million processors. The architecture and strategy to build such a massively parallel SIMD machine have been presented in [9]. This thesis presents an integrated circuit layout for some key components of the computer using the scalable CMOS technology. The layout and verification of a 3x3 network prototype has been completed. Most of components in the 3x3 layout could be used for a million processor computer expansion.

Whether digital systems are high speed, high density, low power, or low cost, CMOS technology is used in the majority of applications. Another advantage of CMOS technology is the availability of various design tools. MAGIC [10], a popular software

package for CMOS VLSI design, is the layout tool used in this design. SPICE-based tools were used to verify and test the performance of the design.

The layout and verification of the SIMD machine with 9 processing elements (PEs) arranged in a 3x3 matrix has been completed. The layout design is appropriate for fabrication with a 2-μm-SCNA MOSIS process[12].

A version 2.2 scalable CMOS standard cell library [11] has been developed by Advanced Microelectronics Division, the Institute for Technology Development (ITD/AuE). Standard cells from this library were included in the design whenever they were available.

Chapter 2

SIMD Architecture

A SIMD computer structure was first introduced by John. Dorband at Goddard Flight Center and has been studed and refined at Bucknell University since 1993. The SIMD machine is designed with a 2-dimensional (2-D) mesh of interconnect processing elements as shown in Figure 2-1. 2-D links, or 2-D routers, connect four nearest neighbor PEs to form a processor array. Boundary processing elements are wrapped around to form a toroidal network. Thus each PE can communicate with its four neighbors through its two routers. A PE Control Unit (PECU) optimizes the performance of the SIMD and broadcasts the instruction stream to all PEs. A PECU requires a host computer to generate a stream of instructions. However, This thesis focuses on processor array architecture. The operation instructions are included in Appendix A. The design and implementation of PECU will not be discussed in this thesis.

The processor mesh is made up of many one-bit serial PEs interconnected by 2-D routers. The PEs receive the instructions from the PECU and execute them. Each PE can be selectively "masked" by the PECU if it is not expected to participate in the computation or execution. All the unmarked PEs carry out the same operations on their own data. Although the PEs only operate on one bit, one-bit operation does not mean the massively parallel machine can not work on a byte or a word, since eight processors could

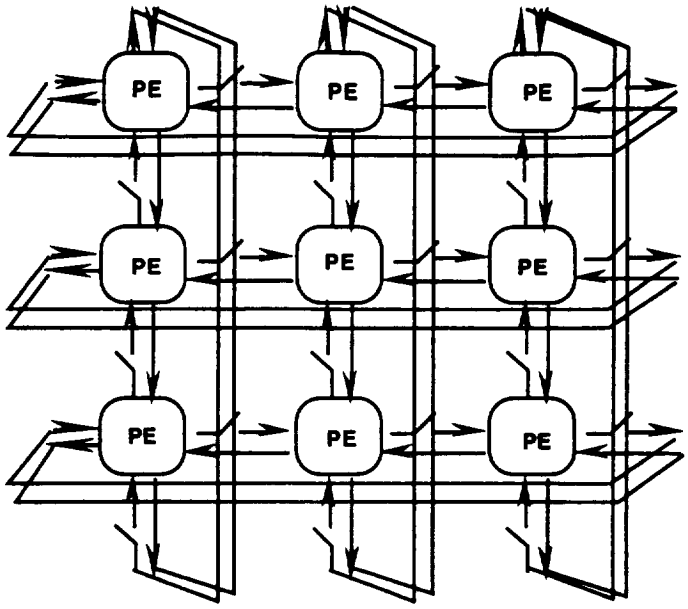


Figure 2-1. SIMD 2-D Mesh Architecture and Toroidal Connection

cooperate and form a byte operation. Because of operating style, all processors are identical and have the structure as shown in Figure 2-2.

The main components in a processing element are a 1k RAM, an arithmetic and logic unit (ALU) and a mask register. An ALU is further divided into a half adder, two D-flip-flops working as an accumulator register and a carry register respectively, and transmission gates for data control and movement. A 2-D router structure which consists of 4 transmission gates in a square formation is shown in Figure 2-3. The horizontal transmission gates (No/We) are controlled by a common signal and the vertical gates (So/Ea) are controlled by another common signal. In the next several sections, a VLSI layout of each part of PEs will be presented.

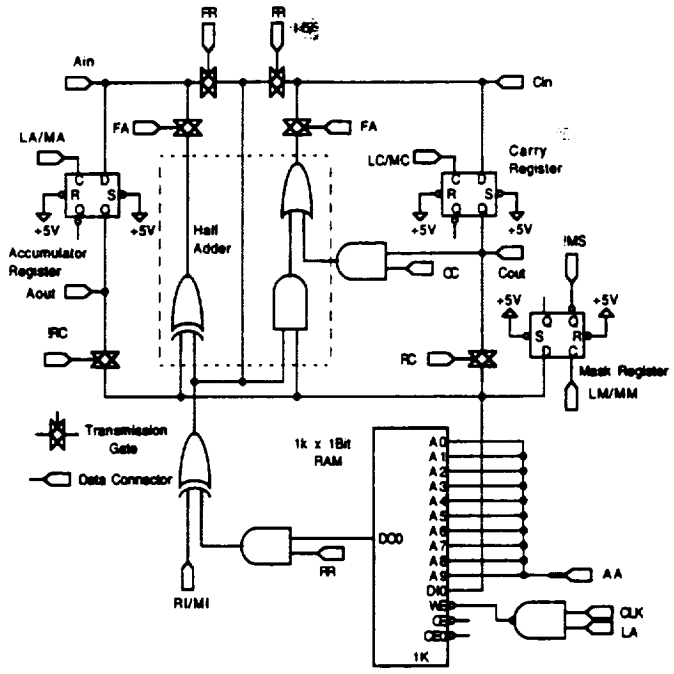


Figure 2-2. One-Bit Processing Element

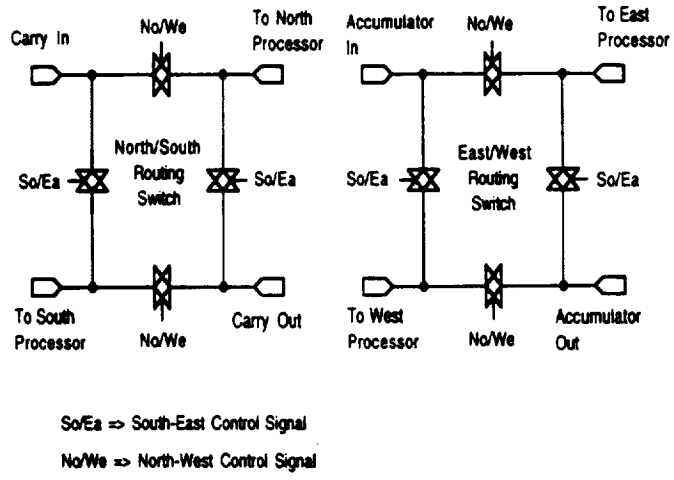


Figure 2-3. Two 2-D Routers

Chapter 3

SRAM

A 16-bit Static Random Access Memory (SRAM) was designed in each PE instead of 1k SRAM since chip area is limited. However, it is not difficult to design a SRAM with higher capacity in the same way. The SRAM is integrated with each processing element providing local storage.

3.1 Introduction

Cells in a static RAM are constructed using latched storage in contrast to dynamic RAM which implements memory with charge stored in a capacitor. Generally, more transistors are needed in static cells than in dynamic cells. Static form is less troublesome than a dynamic form and thus was adopted in this design. The SRAM in each PE contains 16 words with only one bit per word. A typical memory architecture consists of a memory cell array, a row decoder, a column decoder, read/write and bit line conditioning circuits. This design does not have a column decoder because of the small size of the memory. A higher capacity of RAM can be built without much difficulty using a column decoder and duplicating the memory cells to form a bigger memory cell array.

3.2 Storage Cell Structure

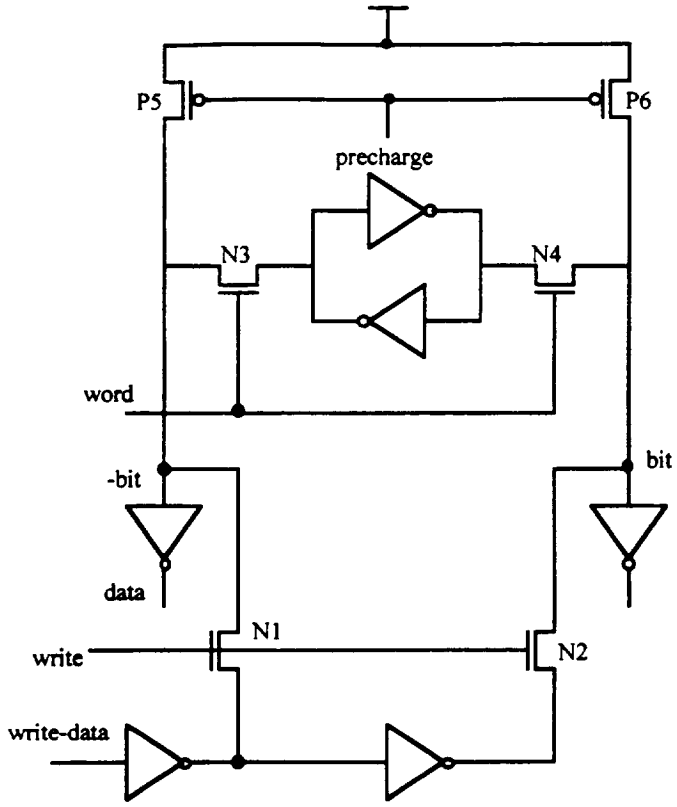


Figure 3-1 A Cell of Static Random Access Memory

A typical storage SRAM cell structure is shown Figure 3-1. The cell consists of two cross-coupled inverters and two n-transistors for a total of 6 transistors. P5, P6 and precharge form a bit line conditioning circuit to refresh bit and -bit lines before reading and writing. The write circuit consists of N1, N2 and two inverters at the bottom of Figure 3-2. N1 and N2 typically require a large area since the currents driving the bit lines and storage cells are supplied by the two n-channel transistors. However, precharge transistors P5 and P6 are small in size. Usually the voltage level of SRAM bit lines

depends on the transistor sizes of precharge, writing circuit and storage cells. Note that typical voltage levels on the bit lines are not 5v for logic 1 (bit(1)) or 0v for logic 0 (bit(0)), but 3.5v or 0.5 instead.

Actually the size of N3 and N4 determines the bit(1) and bit(0) voltages. A bigger size results in a low bit(1) voltage and a high bit(0) voltage and helps improve memory speed. However, as the transistors are limited in size by the desire to keep the SRAM cell small, a design trade-off has to be made between speed and the differential bit voltage. The data and control lines are illustrated in Figure 3-1.

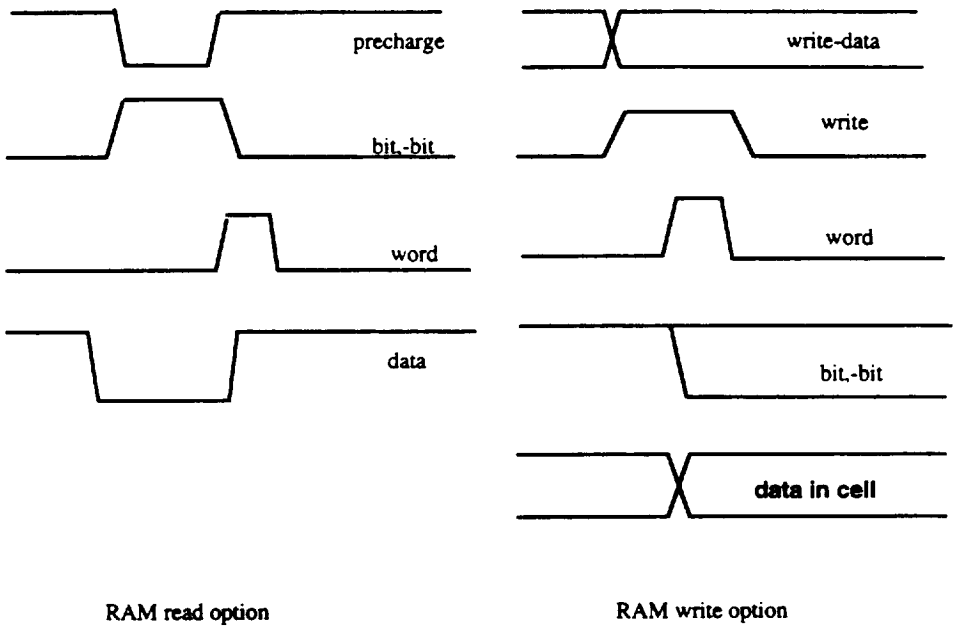


Figure 3-2. Timing Diagram of SRAM

a. Precharge---make the bit line and -bit line be high (5v) because one SRAM cell will attempt to pull down either the bit or -bit line depending on the stored data. Thus precharge works as a reset function for reading from a SRAM.

b. Data -- data output from the cell.

c. Word -- this signal comes from the decoder output. Whenever an address is accessed the corresponding word line will be high or be selected.

d. Write -- should be high while writing a data into a cell.

e. Write-data -- the data needed to be stored in the RAM cell.

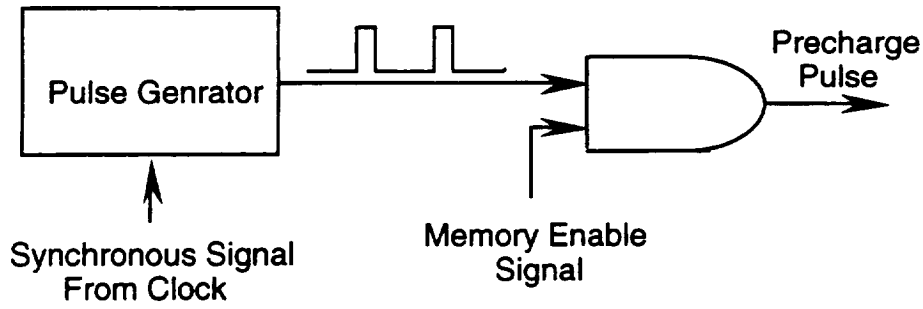


Figure 3-3. Precharge Signal Generation

The timing shown in Figure 3-2 is critical for the SRAM to work properly. When writing, it is required that the write-data and write come up before the word. When reading, precharge should come earlier than all other signals. The precharge signal is produced as depicted in Figure 3-3. The signal generator (or PECU) output a series of pulses which must be synchronized with the clock sequence. The generation of precharge signals is controlled by the memory enable signal through the AND gate. When reading from and storing to a memory, the memory enable signal is high. Otherwise, it is low. In this design, the precharge signal set is needed when either a read or a write operation is performed.

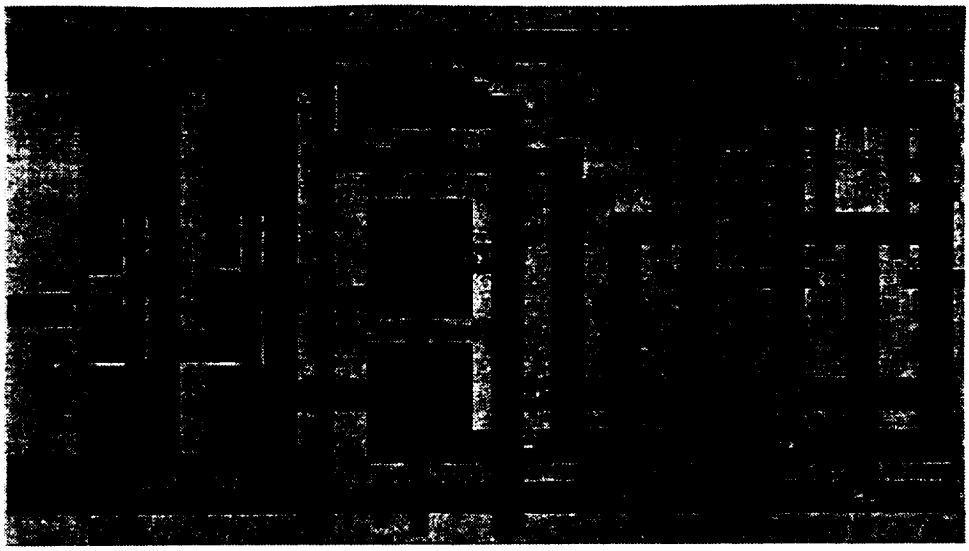
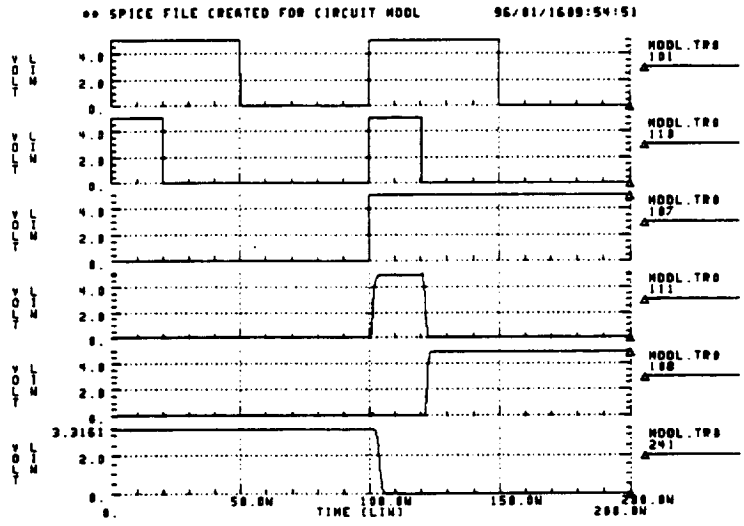
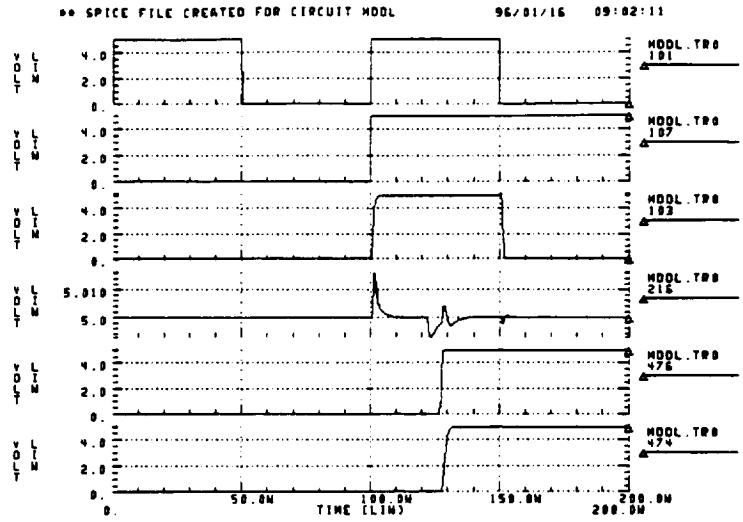


Figure 3-4. Layout Structure of SRAM Cell



101— Clock, 110 — Pulse, 107 — Enable, 111 — Precharge, 108 — OE, 241 — Data in cell

Figure 3-5. Timing Diagram for Read Operation



101 --- Clock, 107 --- Enable, 103 --- Word, 216 --- Write-data, 474 --- OE, 474 --- Data in cell

Figure 3-6. Timing Diagram for Write Operation

How much time to precharge the bit lines depends on the fabrication processing technology. A typical minimum of precharge time in the MOSIS SCNA 2-μm CMOS technology is 10 ns for this design.

The MAGIC layout for the precharge circuit, the writing circuit and the storage cell are shown in Figure 3-4. Performing extraction yielded a netlist which was simulated in HSPICE [12]. The resulting timing diagrams are shown in Figures 3-5 and 3-6 respectively.

3.3 Decoder

An example, 4-word SRAM decoder structure, is shown in Figure 3-7. Four nand gates are needed for decoding and another four nand gates are used for word enable control in the circuit. The enabling of word is necessary to ensure correct timing of the word signals. Because of the SRAM cell structure, a slow rise time and fast fall time on the word-decode gate is advantageous. A 16-word decoder is constructed in the same way except there are four address lines with each line connecting two inverters in series, 16 nand gates for decoding and 16 nand gates for the enable control signals. A part of the decoder layout is shown in Figure 3-8.

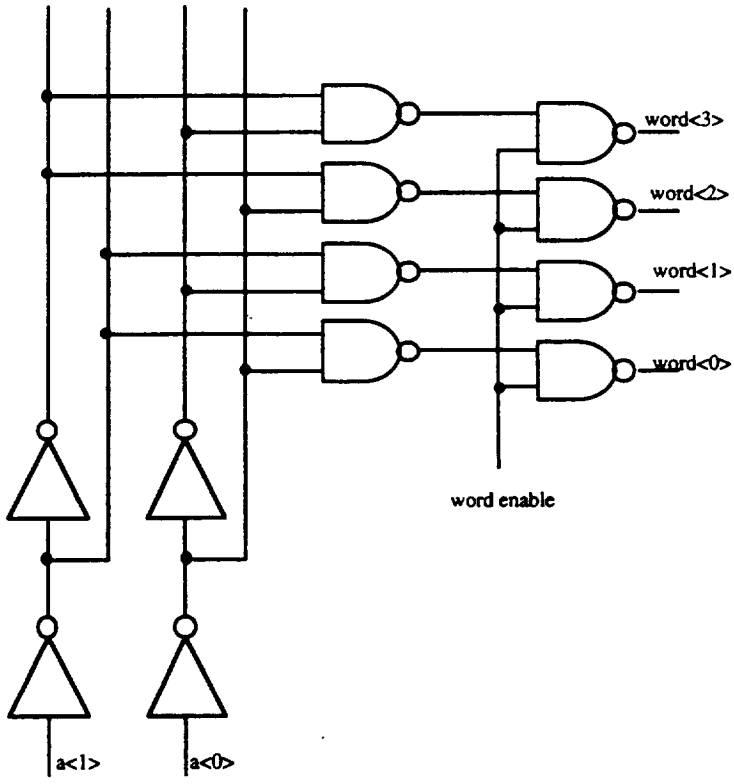


Figure 3-7. Decoder Circuit of Memory

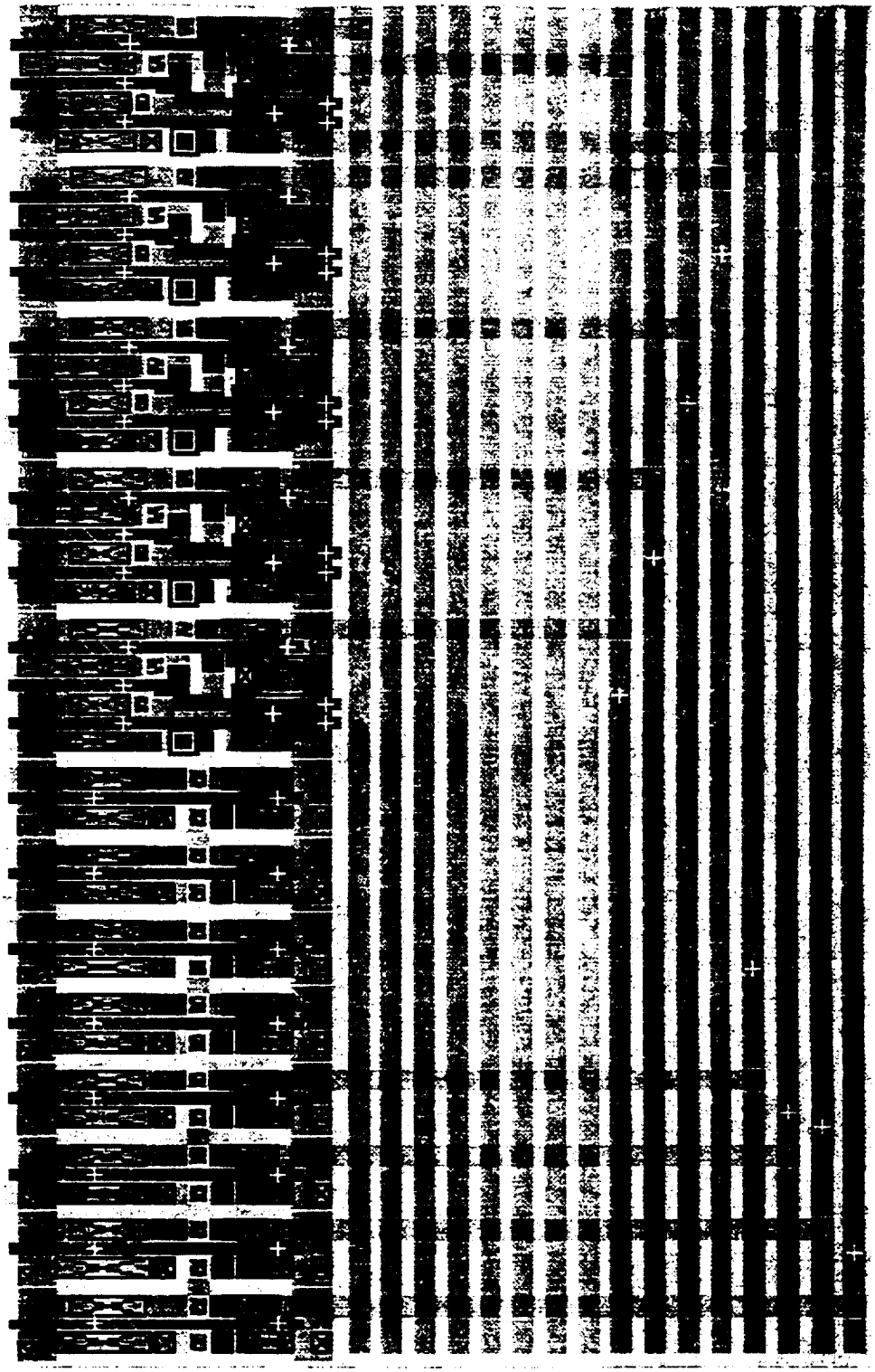


Figure 3-8. A Part of Layout of Row-only Decoder

3.4 Output Enable Control Signal

As mentioned earlier enable control is an essential signal when reading from memory and writing to memory. The memory enable is created along with the precharge pulse. The logic circuit for producing a precharge pulse and Output Enable (OE) is illustrated in Figure 3-9. In the logic circuit, the memory enable signal is not available until the precharge pulse ends as shown in Figure 3-5 and 3-6. The word-line assertion is made when the memory enable signal is set (refer to Figure 3-7). One must be careful that if the word-line assertion precedes the end of the precharge cycle, the SRAM cells may flip states. For a read operation no read signals are required, while for a write operation, write signal and write-data should be set up when writing. In this design, the precharge signal set is needed when either a read or a write operation is performed.

The layout structure is shown in Figure 3-10. The verification of this circuit could be checked through read and write operations, as shown in Figures 3-5 and 3-6.

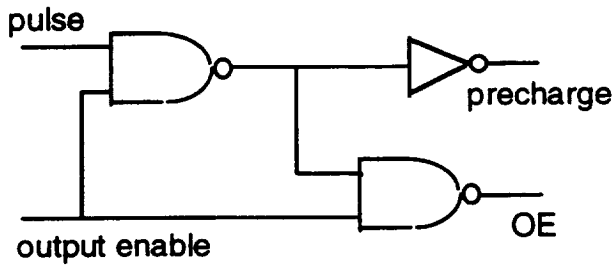


Figure 3-9. Output Enable Control Signal

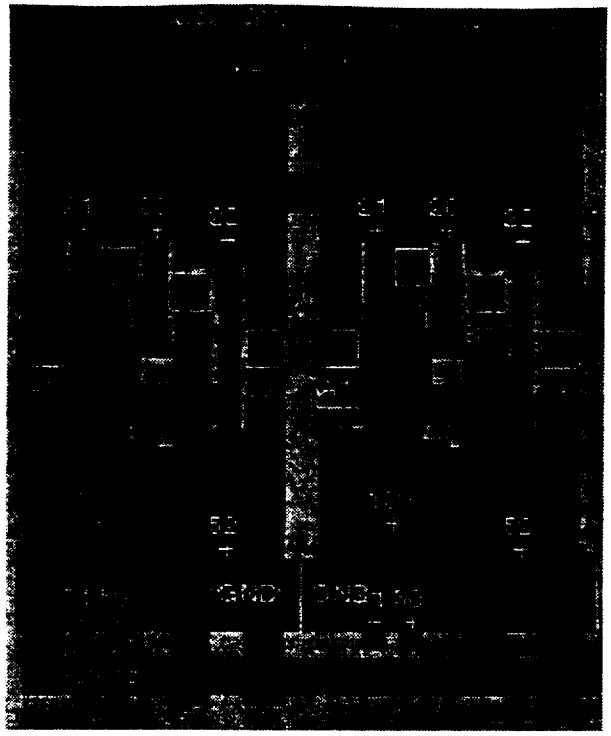


Figure 3-10 Memory Enable Logic Layout

3.5 Operation Instructions

Some operation instructions presented in the appendix of [9] should be modified since the extra control signal called output enable is introduced. These modifications are included in Appendix A of this thesis. There are two purposes for using the output enable (OE) control signal. One use is to make RAM available for read and write. The other use is as a switching signal to turn on or turn off the precharge pulse (Figgie 3-4 and Figure 3-9). Thus, whenever an operation involves read from RAM (RR) and load to RAM (LR) the instruction should be altered so that all RRs and LRs are replaced by RR, OE and LR, OE. For example, the following is a change for the addition operation instruction,

Addition:

LC// clear c

repeat

RR OE AC0 RC LC LA AA(op1++) in--// 1st half add

RR OE AC0 OC LC LA AA(op2++) // 2nd half add

LR OE AA(op3++)

while (in)

where OE is short for Output Enable signal.

3.6 Verification

The memory layout file was named RAM16.mag. A netlist was extracted and named RAM16.spice. To create a HSPICE file, open the SRAM layout file with

%magic RAM16 //UNIX command

and perform the following,

:extract //Magic Command

:quit //Quit from Magic

% ext2spice RAM16 //UNIX command

Once the HSPICE file RAM16.spice has been created, it should be edited to substitute all of the NFETs and PFETs in the file with CMOSNs and CMOSPs

respectively, and add the necessary power supply, input signals, display statement and the CMOS model file. To simulate the Hspice file, use the following command.

% hspice(filename) *//UNIX command*

The simulation will take a while to complete depending on the chip size. A verification result file is created and named RAM16.tro. The GSI graphics package included with HSPICE, can be used to display waveforms of any nodes. To do so, input the command as follows:

% gsi RAM16.tro //UNIX command,

This procedure was used to verify all parts of the proposed SIMD machine.

The verification of the RAM includes read and write operations using 5v and 0v as stored data. A logic 1 (5v) was stored into the memory at address (0001) in a time range of 50ns. Then, a read operation from address (0001) was followed in the second 50ns. From 100ns to 200ns write and read operations repeat with 0v data stored in address (0010). The results of RAM verifications are shown in Figures 3-5 and 3-6.

Chapter 4

1-Bit Processing Element

As shown in Figure 2-2, the processing element consists of a bit serial arithmetic and logic unit (ALU) and a distributed bit serial RAM. We have presented in chapter 3 the RAM structure and how it works. In this chapter we discuss how to lay out an ALU, control logic and their corresponding components.

4.1 Half-adder

A half-adder is the fundamental component of the ALU. The two operands may come from local RAM, accumulator output, carry output or neighboring PEs. The outputs of the half adder, sum and carry, are stored in the accumulator register and carry register, respectively. 12 transistors should be used if standard OR and AND gates are used for the implementation of the logic circuit, which consists of the OR and AND gates in the half adder, and the AND with OC being one of its inputs (Figure 4-1). However, in this design a standard model aof2201 logic was used instead of two AND and one OR gates. Model aof2201 is shown in Figure 4-2 and consists of 10 transistors. The model performs the logic functions as shown in Figure 4-1.

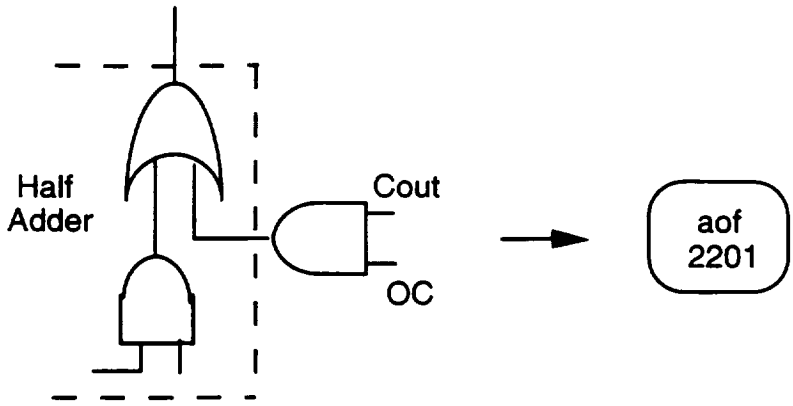


Figure 4-1. Part of half adder and the AND gate form Model aof2201

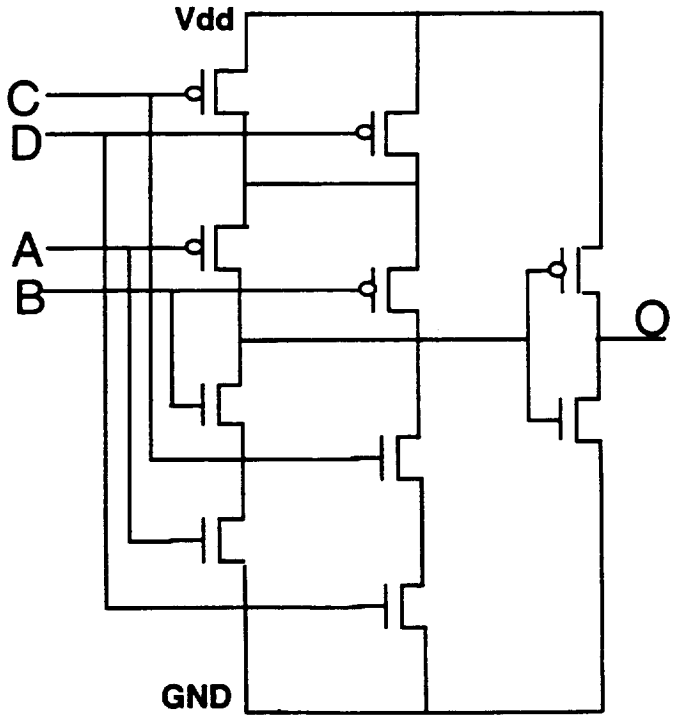


Figure 4-2. Circuit of Model aof2201

The layouts of aof2201 (left) and a standard XOR gate (Model xorf201, right) are depicted in Figure 4-3.

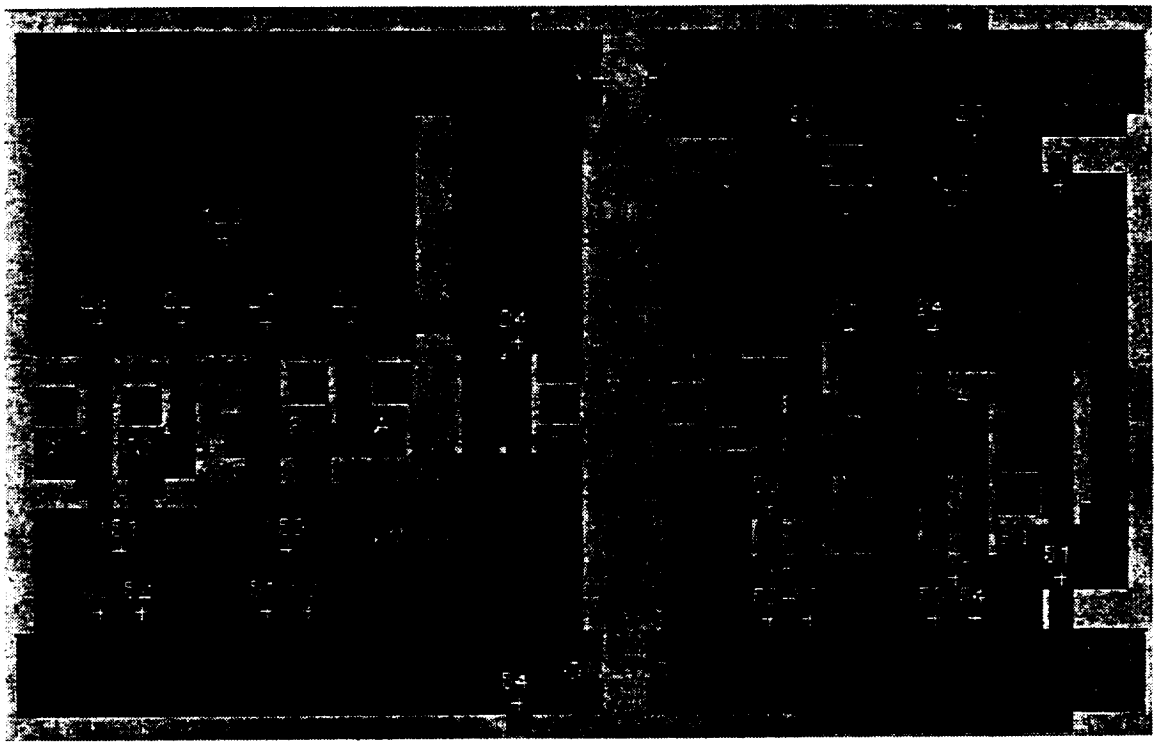


Figure 4-3. Layout of aof2201 and XOR Gate

4.2 Register

Three of D flip-flops, triggered by the falling-edge, form the accumulator register, carry register and mask register, respectively. Accumulator register A and carry register C not only store the answers of an add operation, but also receive data from, and transfer data to, other PEs. The data transferred in North/South direction will pass through the

carry register, while the data transferred in East/West direction will pass through the accumulator. The mask register M enables certain processors not to participate in a given operation. The output Q of register M is used as a signal to control the corresponding PE. Whether the M register is loaded depends on the trigger signal LM/MM. The Model dfbf311 in the standard library is used as layout structures for three registers as shown in Figure 4-4.

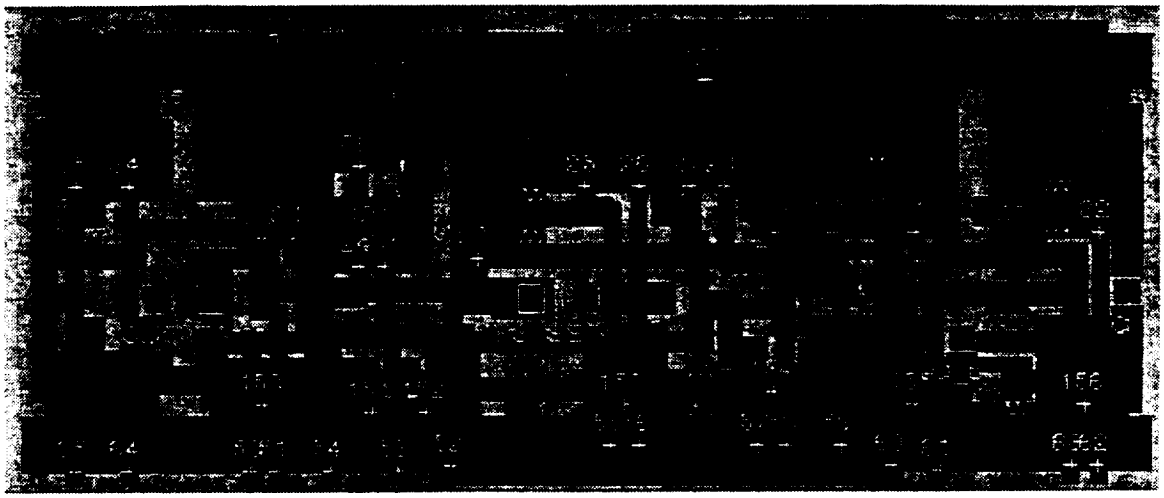


Figure 4-4. 1-bit Register (D flip-flop) layout

4.3 Transmission Gates and Router

Transmission gates are widely used in the SIMD computer to make sure there are no conflicting signals. There are six transmission gates in each processing element. Basically each transmission gate consists of four transistors. The layout of a transmission gate is shown in Figure 4-5.

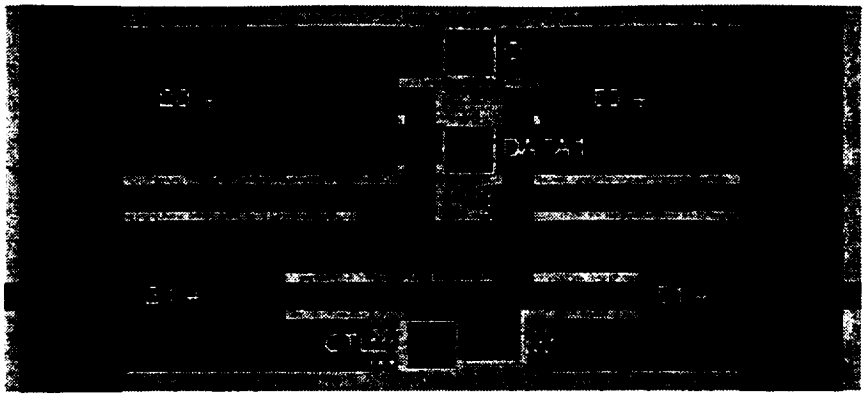


Figure 4-5 Layout of Transmission Gate

A router consists of only transmission gates. Verification of a router causes problems. When a transmission gate is off, the voltages of its input and output depend on its characteristics such as capacitance and resistance. A transmission gate tends to keep the previous voltage levels at its input and output when it is turned off. When a circuit contains several transmission gates which are connected together, like the routers, the load resistance and load capacitance vary corresponding to number of transmission gates connected together. Verifying a router with its load together is the way to avoid the problem. As shown in Figure 4-6 it is appropriate that D flip-flops and inverters were used as the loads for a router.

4.4 Control Logic

There are three levels at which the processing element may receive and transfer data. A model of this configuration is shown in Figure 4-7. These levels are:

A. Local Operation

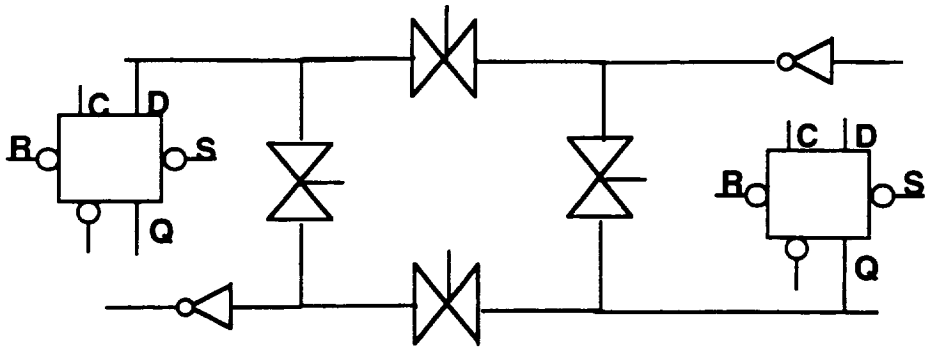


Figure 4-6. Router Is Loaded with D Flip-Flops and Inverters for Verification

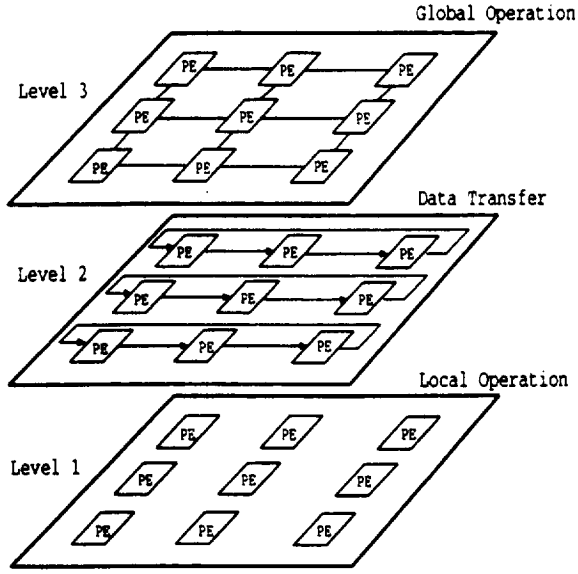


Figure 4-7. Three Operation Levels

B. Data Transfer

C. Global Operation

Each processing element may only operate at one of these three levels at a time. Thus, the three types of control logic equip the SIMD computer to work at the three different levels.

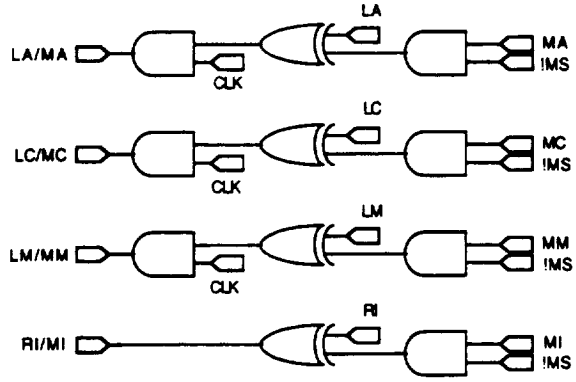


Figure 4-8. Control Logic For PE Local Operation

At the local operation level, the basic control signals are FA, FR, RI/MI, LA/MA, LC/MC, LM/MM. These control signals are created by the logic circuits shown in Figure 4-8 and Figure 4-9. Signals RT and GB produced in Figure 4-9 are used as control signals for data transfer and global operations. Figure 4-10 shows the logic for PE router control signals.

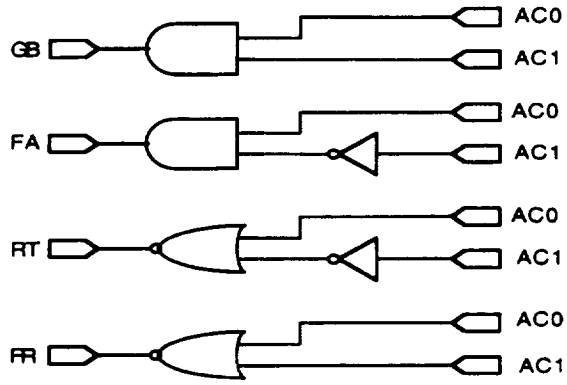


Figure 4-9. Control Logic for Network and Global Operations

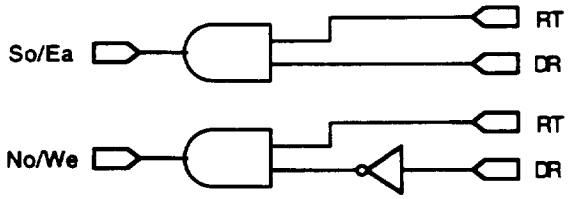


Figure 4-10. Control Logic for Routers

The layouts of control logic for three operation levels are shown in Figures 4-11 and 4-12.

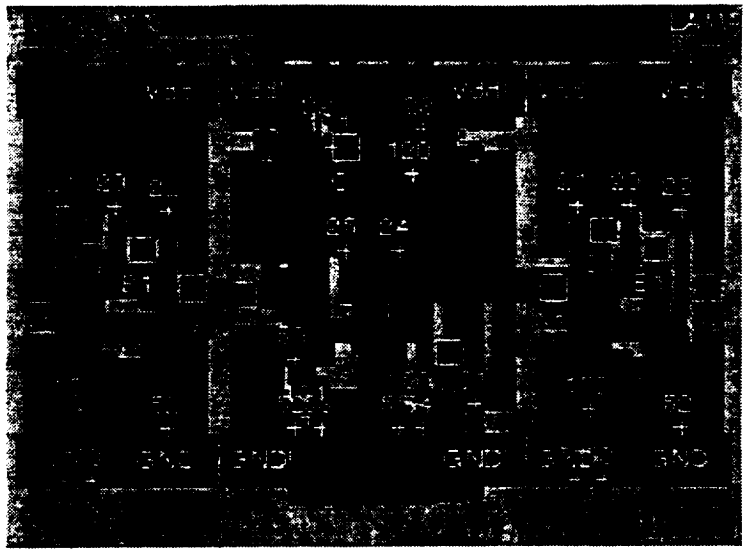


Figure 4-11. Layout of Control for Local Operations

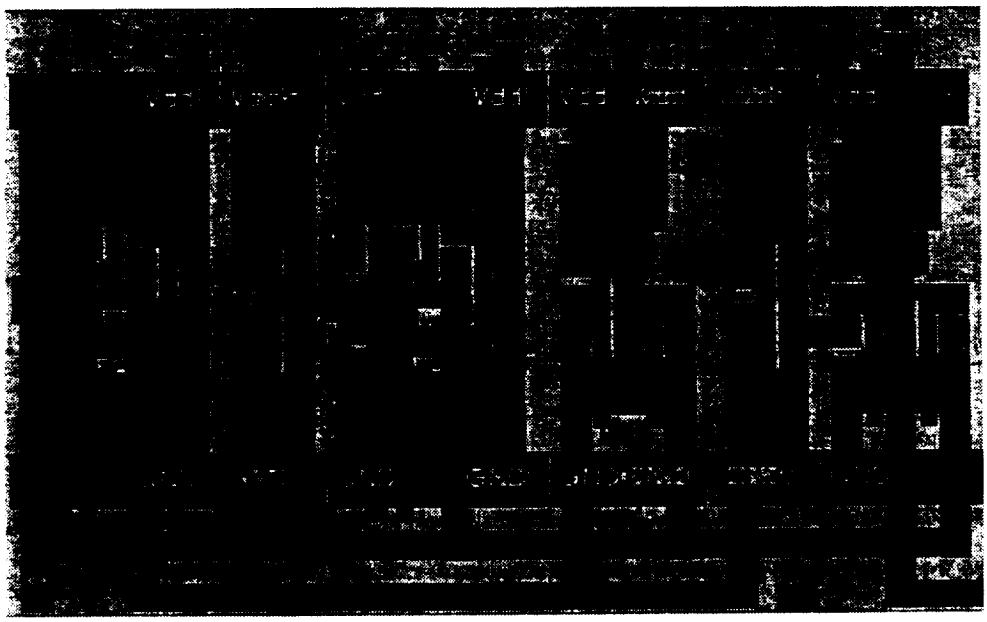


Figure 4-12. Layout of Control for Network and Global Operations

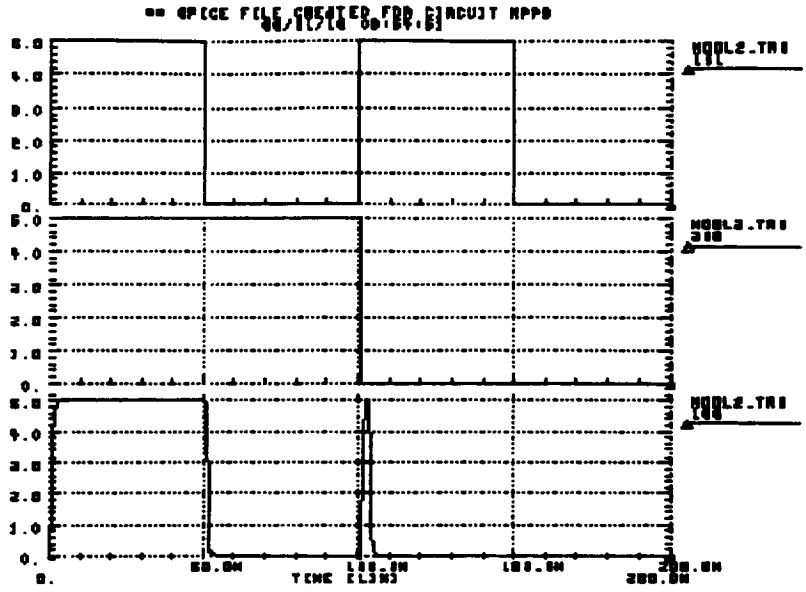


Figure 4-13. Little Burst Makes An Undesirable Register Flip

Problems were encountered during the verification. For example, sometimes the registers A, C and M were falsely triggered. The reason is that there is a delay for each gate. For example, LA has a high voltage (5v) at 0ns and goes down to 0v at 50ns, while clock operates as usual so that it has a high voltage (5v) from 0ns to 50ns and falls to 0v at 50ns (refer to Figure 4-13). An undesirable burst appears in LA/MA waveform. The little burst is big enough to trigger accumulator register and to make it flip. The solution is to arrange the signals clock, LA and MA properly.

The original control signals such as LA, LC, LM, MA and clock, come from PECU. These should be arranged to occur at the right time for a correct operation.

4.5 Global OR Gate

Each PE has two global OR gates associated with it, one for ORing columns together and the other for ORing rows together. Figure 4-14 shows global connection in one row of the OR gates. A similar connection is set up for columns of processors. Two cycles are needed for each global OR operation (except the read cycles). During the first cycle, the accumulator register content in each row of PEs is ORed and the answer is stored in the carry register in each PE. During the second cycle, the carry register content in each column is ORed. The result is the global OR result and may be stored in the accumulator register or in the carry register.

A problem arises because of the global operation style and serial connection of the OR gates. If there is 10ns delay time for each OR gate and there are N row OR gates, then the total delay of the operation for ORing a row of PEs will be 10N ns. Therefore, the clock cycle will be limited and should be less than 10N ns. The Hspice simulation results from 3 OR gates connected in series for global OR operation are depicted in Figure 4-15. For small scale network or the network working at low speeds there are no problems. However, for large scale network the problem has to be considered properly.

The verification of global OR is included in 3x3 network verification result shown in Appendix E.

4.6 Verification

The verification results for the processing element are shown in Appendix D. The inputs of global OR gates were set at logic 0 (0v) and the outputs of the global OR gates were connected with a 10kohm resistor as a load. Read from RAM, Add and Load to

RAM were used for the functionality test of one PE. The two routers included in the PE layout can not be verified since a move data operation can not be used as a test function in this case. However, routers were tested without connecting any PEs as mentioned in router section of this chapter.

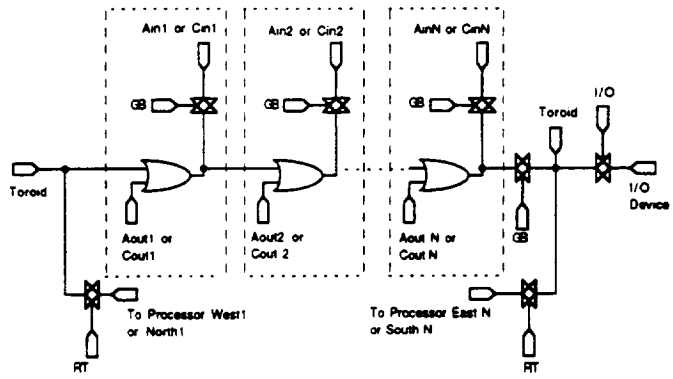


Figure 4-14. Global OR Gates Connected in Series

Chapter 5

3x3 Network

The 3x3 network topology is shown in Figure 2-1. When working on a 3x3 network all PEs must work properly. Any little mistake in the PE layout could cause much trouble at network level.

5.1 Layout Structure

In the original design[9] there are six interfaces linked to PEs on the left column and bottom row. In this layout three I/O interfaces are implemented in PEs on the left column instead of six I/O interfaces. The I/O interface was re-designed in this layout so that input and output are separated as shown in Figure 5-1. The advantage to isolate the input and output is that they could be connected with data load device and output device independently.

Nine PEs are respectively named NW (North-west), North, NE (north-east), West, Mddl (middle), East, SW (south-west), South and SE (south-east) according to their locations. Due to the toriodal connections and I/O interfaces, the structures of the processors on the edges are slightly different from the middle processing elements (refer to Appendix F). Transmission gates are needed for connecting the wrap around links in

North/South direction and East/West direction. Four transmission gates were included in processor NW (northwestern corner). Two transmission gates in processor NW are used for North/South toriod connection and the other for East/West toriod connection. Each processor on the north edge (or the top row), , such as processor North, contains two transmission gates which are used as north/south connection. For processor NE there are both two transmission gates for global link and one I/O interface.

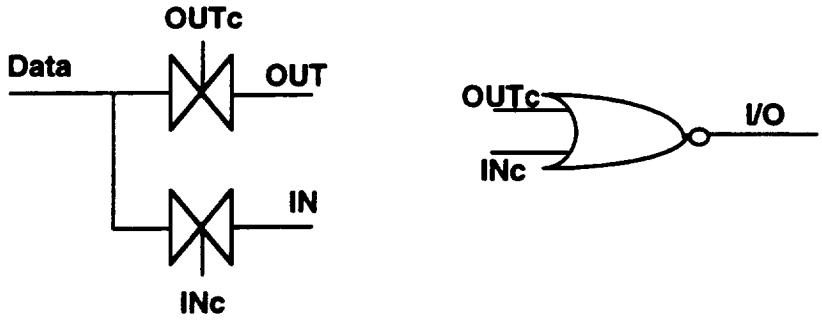


Figure 5-1. I/O Interface with Input and Output Isolation

Therefore in total there are six of PEs. They are:

A. NW type---with two global links for both north/south and east/west directions.

B. North type--- with one global link for north-south direction.

C. NE type ---with one global link for north-south connection and one I/O interface.

D. West type ---with one global link for west-east connection.

E. East type ---with one I/O interface.

F. Mddl type --- with no global link and I/O interface.

5.2 Frame

An n-well frame in 4600x6800 size was designed for the 3x3 network, if it is desired to fabricate a 3x3 network for tests at a chip level before a large scale machine is built. The frame has 40 pads as shown in Appendix F. The pin arrangement is shown in the table of Appendix F.

I/O structures require a designer to have the most amount of circuit-design experiences and detailed process knowledge. Thus it is probably inappropriate for a system designer to contemplate I/O pad design. There is a pad library in *http://www.isi.edu/mosis/cell-libraries/scn20-pads/* for p-well and n-well processing chips. Power and ground pads called Vdd and Vss are easily designed and consists of a sandwich of the metal pad layers connected to the appropriate bus. A two-level metal process affords good crossovers[12]. A standard logic circuit for I/O pad is shown in figure 5-4. From the tri-state I/O pad the INPUT and OUTPUT pads are derived. Connecting ENABLE to GND will turn this I/O pad into an input pad. Otherwise, connecting ENABLE to Vdd will result in an OUTPUT pad.

There are several lines in the HSPICE file converted from pad layout left open. Therefore for verification those lines should connected to a certain load. The load capacity and delay parameters about the n-well pads can be found in [12].

5.3 Verification

In order to verify the 3x3 network connection environment, it is recommended that all outgoing interfaces be connected to a resistance with an appropriate value. Be sure not to leave the interfaces open or shorted.

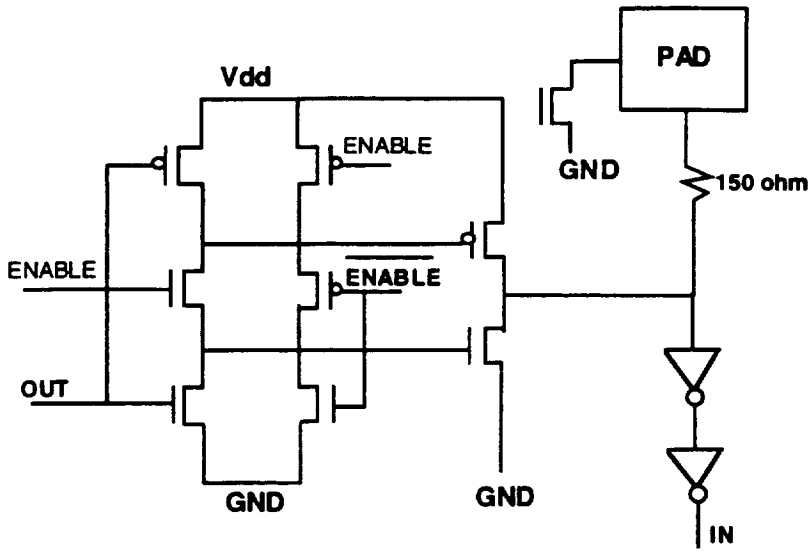


Figure 5-3. Logic Circuit for I/O pads

Several operation were performed to verify the 3x3 network layout. The operations are data input from east, data movement from east to west, store (LR), read from RAM (RR) and global OR. The cycles are described as follows,

1st cycle: Input (0 0) into processors on the east edge.

2nd cycle: move the vector (1 0 0) to middle column processors and inputting vector (0 1 0) into the processors on the east edge.

3th cycle: move the vector (1 0 0) to the left column processors, move the vector (0 1 0) to the middle column and input the vector (0 0 1) into the processors on the east edge.

The performance of the three steps results in an array of processors containing

$$A \Leftarrow \begin{pmatrix} 100 \\ 010 \\ 001 \end{pmatrix}$$

4th cycle: store data into RAM cell with address 0001.

5th cycle: test the global OR (using the data in RAM cell of address 0001).

6th cycle: store the global OR result into RAM cell with address 0000.

The simulation timing results for these operations are shown in Figure 6-1.

5.4 Power and Size Estimation

The highest transient current in 3x3 network simulation is about 144.9mA. The average current is quite low --- about 20mA. Therefore, the power consumption power is about 0.1 watt or 3x3 network. For a 1000x1000 network the power might raise to 10KW.

SRAM takes about 50% of area in each PE. If a higher capacity of memory is required, then the chip area will increase significantly. The exact size of 3x3 network in the layout is 3837(lambda) x 3534(lambda). The area is divided as follows,

40% for components

40% for connection wire

20% for die area

If the layout is fabricated using 2.0um processing technique it takes about 7674u x 7078u for a 3x3 network. For 0.6u processing technique the size could be about 2302u x 2120u. A 1000x1000 network takes a size of 2 m x1.8 m and a size of 0.6m x 0.5m respectively for 2um and .6um processing technique

Chapter 6

Discussion and Summary

6.1 Expansion

There are six types of PEs used for configuration of a 1000x1000 mesh network. They are NW, NE, North, West, East and Mddl types. The NW PE, NE PE, East PE and West PE should be placed on north-west, north-east, south-east and south-west corners respectively. On the north edge of the network North PEs should be used, while on the east and west edges the East and West type PEs would be used. For all other PEs the Mddl type PE should be selected to configure the network.

The best way to implement a machine with more than one million processors is to set the chips on several boards where one chip should have a limited number of processing elements such as 64, 128 or 256. Some essential wires to make connection between chips should be set on the boards.

A wrap-around path delay might arise in implementation of a million PE network. Since the length of a wrap-around connection path is at least 1000 times the length of the neighboring link paths when moving data from one PE to another, the operation might have to take a longer time. For example, if the length of the wrap-around path is 30 cm, the time for data to pass through the path is at least 10ns. It is one of the factors that limit the operation speed of the SIMD machine.

As mentioned in Chapter 4, a global operation such as global OR or global AND operations can not be completed in 2 clock cycles (refer to instruction collection in Appendix B). This problem has to be solved in software and hardware before implementing a massively parallel computer.

6.2 Speed Limit

Two factors determine the speed of operation of the SIMD. The first is the local operation speed which mainly depends on gate delays. Since the PE was designed to be simple in structure and small in size, then, a routing path delay could be neglected for local operations.

The second factor is the operation speed at the network and global levels. It is obvious that gate delay will affect the operation speed as it does in local operation speed case. However, the layout topology of the networks and the routing strategy to connect them together are critical to the data transfer and global operation. For example, a random register flip caused by delay of gates connected in series will limit the clock rate. The routing way in which instructions are distributed to each PE also affect the speed of data transfer and global operations. An ideal situation is each PE receives the instruction exactly at the same time, which only could be realized by making the control path length equal to each other.

The goal to raise speed might conflict with other requirements. For example, the problem about wrap-around path delay might be solved by distributing PEs uniformly on a ring. However, the structure probably takes more area and even a mesh network is impossible to be implemented.

6.3 Summary

A CMOS VLSI layout and verification of a 3x3 processor parallel computer has been completed. The layout was done using the MAGIC tool and verification using HSPICE. Suggestions for expanding the computer into a million processor network are presented. Many problems that might be encountered for implementing a massively parallel computer are discussed.

Bibliography

[1] Y. Li, A. W. Lohmann, Z. G. Pan, S. B. Rao, I. Redmond and T. Wang, "Optical Multiple-Access Mesh-Connected Bus Interconnects", *Proceeding of the IEEE, Vol. 82, No.11*, pp. 1690-1699, Nov. 1994.

[2] D. A. Reed, K. A. Shields, W. H. Scullin, L. F. Tavera and C. L. Elford, "Virtual Reality and Parallel Systems Performance Analysis", *Computer, Vol. 28, No.11*, pp. 57-67, Nov. 1995.

[3] M. T. Heath, A. D. Malony and D. T. Rover, " The Visual Display of Parallel Performance Data", *Computer, Vol. 28, No.11*, pp. 21-28, Nov. 1995.

[4] H. Li, and Q. F. Stout, "Reconfigurable SIMD Massively Parallel Computers", *Proceedings of The IEEE, Vol. 79, No.4*, pp. 429-443, Apr. 1991.

[5] K.-S. Huang, C. B. Kuznia, B. K. Jenkins and A. A. Sawchuk, "Parallel Architecture for Digital Optical Cellular Image Processing", *Proceedings of The IEEE, Vol.82, No.11*, pp. 1711-1723, Nov. 1994.

[6] N. Jayant, B. D. Ackland, V. B. Lawrence and L. R. Rabiner, "Multimedia: Technology Dimensions and Challenges", *AT&T Technical Journal*, Sep./Oct. 1995.

[7] J. P. Strong, "Computations on the Massively Parallel processor at the Goddard Space Flight Center," *Proceedings of The IEEE, Vol.79, No.4*, pp. 548-558, Apr. 1991.

Bibliography

[8] Michael J. Quinn, *Parallel Computing Theory and Practice*, McGraw-Hill, Inc. New York, 1991.

[9] J. Dorband, M. F. Aburdene, K. Khouri, J. Piatt and J. Zheng, "Design of Massively Parallel Computer Using Bit Serial Processing Eleements", *Proceeding of The 29th Annual Conference On Information Sciences and Systems,* The John Hopkins University, Baltimore, MD 21218, pp. 828-833, 1995.

[10] *MAGIC Manual,* Computer Science Division, Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720, Sept. 1990.

[11] *HSPICE Manual,* Customer Service Department, Meta-Software, INC., Campbell, CA 95008, 1992.

[12] http://www.isi.edu/mosis/

[13] D. Elliot, "Computational RAM: A Memory-SIMD Hybrid and Its Applications to DSP", *Proceedings. Custom Integrated Circuits Conference,* Boston, MA, pp. 30.6.1-30.6.4, May 1992.

[14] L. Snyder, "Introduction to the Configurable Highly Parallel Computer", *IEEE Computer, Vol.15*, pp. 47-56, 1982.

[15] S. L. Tanimoto, "Memory Systems for Highly Parallel Computers", *Proceedings of The IEEE, Vol.79, No.4*, pp. 403-415, Apr. 1991.

Appendix A

Operation Instructions

The following section contains operation and the lines of code needed to describe their execution. Each line of code consists of a list of signals, a memory address location (AA), and a comment line (comment lines start with "//") or the loop statements "repeat" and "while (i)." It is assumed that "repeat" does not take any time, and "while (i)" is an end of loop comparison combined with the last instruction of the loop. All other lines of code take one clock cycle to execute. If a signal is represented in the line of code, then it takes on a value of 1, otherwise it is 0 for that instruction. Names written in lower case, such as op1, op2 and sz, are counter and register values generated by the array control unit, when an instruction is being executed by the array. The notation AA(op1++) means address "op1" will be used in the current instruction to access array memory address AA, while it is being incremented in the control unit.

```
Addition:
        LC      // clear C
        repeat
                RR OE AC0 RC LC LA      AA(op1++) in -- // 1st half add
```

```
        RR OE AC0 OC LC LA     AA(op2++)       // 2nd half add
        LR OE                  AA(op3++)
      while (in)
```

Subtraction:

```
      RI LC   // set C
      repeat
        RR OE AC0 RC LC LA       AA(op1++) in-- // 1st half add
        RR OE AC0 OC LC LA RI    AA(op2++)      // 2nd half sub
        LR OE                    AA(op3++)
      while (in)
```

Multiplication:

```
      // op1 -- LSB of multiplier
      // op2 -- LSB of multiplicand
      // op3 -- LSB of product
      // sz1 -- size of multiplier
      // sz2 -- size of multiplicand

      // load multiplicand into product

      // load A with 1st bit of multiplier
        RR OE LA      AA(op1++) (sz->in) (sz1->inn) (op3->tp3) (op2->tp2)
      // Load M from A, clear A and C
```

```
LM LR OE LC                    (op3++) (inn--)
repeat      // load product masked
   RR OE LA MA        AA(tp2++) (in--)
  LR OE               AA(tp3++)
while (in)
LR OE RC LA LC         AA(tp3)

  // perform multiply

repeat
        RR OE LR AA(op1++) (sz->in) (op3->tp3) (op2->tp2)
     LM                    (op3++) (inn--)
     repeat        // add in multiplicand masked
        RR OE AC0 RC LC LA          AA(tp3++) in--
       RR OE AC0 OC LC MC LA MA  AA(tp2++)
        LR OE                          AA(tp3++)
     while (in)
   LR OE RC LA LC
 while (inn)
```

Divide:

```
    // op1 -- LSB of divisor
    // op2 -- MSB of dividend
```

```
// sz1 -- size of divisor
// sz2 -- size of dividend

// clear (sz1) bits more significant than op2
   LA ((op2+1)->tp2) (sz1->in1)//clear A
repeat
        LR OE   AA(op2++) (in--)
 while (in1)

  LA RR OE        AA(op2) (op2->tp2) (sz2->in2) (op1->tp1)
repeat
          LC LM LR OE AA(tp2) (sz1->in1) (op2->tp2) in2--
      repeat
          RR OE AC0 RC LC LA          AA(tp2++) in1--
          RR OE AC0 RC LC LA RI MI  AA(tp1++)
          LR OE                          AA(tp2++)
      while (in1)
      RI AC0 LA              (op2--) (op1->tp1)
 while (in2)
LR OE          AA(tp2)

// op2        -- LSB of remainder (size=sz1)
// tp2        -- LSB of quotient (size=sz2)
```

Logic Function:

And:

```
repeat
        RR OE LC            AA(op1++0 in--
      RR OE RC AC0 LC     AA(op2++)
        RC LR OE            AA(op1++)
  while (in)
```

Or:

```
RI LC
repeat
        RR OE LC            AA(op1++) in--
      RR OE OC AC0 LC     AA(op2++)
        RC LR OE            AA(op1++)
  while (in)
```

Xor:

```
repeat
        RR OE LA            AA(op1++) in--
      RR OE AC0 LA        AA(op2++)
      LR OE               AA(op1++)
  while (in)
```

Not:

```
repeat
        RR OE RI LA          AA(op1++) in--
       LR OE                 AA(op2++)
  while (in)
```

Comparison

```
// op1 -- MSB of 1st operand
// op2 -- MSB of 2nd operand
// C    -- greater than flag
// M    -- equal flag

LA RI
repeat
        LC MC LA MA RR OE LM      AA(op1--) sz--
       RR OE RI LA MA             AA(op2--)
 while (sz)
```

Equal:

```
  // A-- result flag
 perform Comparison
RI MI LA
```

Greater Than Or Equal:

// A -- result flag

perform Comparison

perform Equal

RC AC0 MA

lecithin Or Equal:

// A -- result flag

perform Comparison

perform Equal

RI RC AC0 MA

Not Equal:

// A -- result flag

perform Comparison

MI LA

Greater Than:

// A-- result flag

perform Comparison

perform Not Equal

RC AC0 LA MA

lecithin:

```
        // A -- result flag
        perform Comparison
        perform Not Equal
      RI RC AC0 LA MA
```

Memory to Memory Moves:

Move:

```
        // op1 -- LSB of source
        // op2 -- LSB of destination
        repeat
              LA RR          AA(op1++) sz--
              LR             AA(op2++)
        while (sz)
```

Move (masked):

```
        // op1 -- LSB of source
        // op2 -- LSB of destination
        // ms -- exchange mask
      LA RR OE          AA(ms)
      LM
        repeat
              LA RR OE          AA(op2) sz--
              LA MA RR OE       AA(op1++)
              LR OE             AA(op2++)
```

```
        while (sz)
```

Memory to Memory Exchange:

Exchange:

```
        // op1 -- location 1
        // op2 -- location 2
       repeat
            RR OE LA              AA(op1) sz--
           RR OE LC               AA(op2)
            LC RC                AA(op1++)
            LR OE                AA(op2++)
        while (sz)
```

Exchange (masked):

```
        // op1 -- location 1
        // op2 -- location 2
       // ms -- exchange mask
      LA RR OE                AA(ms)
      LM
       repeat
            RR OE LA MA MC          AA(op1) sz--
           RR OE MA LC MC           AA(op2)
            LR OE RC                AA(op1++)
```

```
        LR OE                   AA(op2++)
    while (sz)
```

Processor to Processor Moves (masked):

MoveN (masked):

```
    // op1 -- LSB of source
    // op2 -- LSB of destination
    // sz -- size of operands
    // ds -- distance
    // ms -- exchange mask
  LA RR OE      AA(ms)
  LM
  repeat
      LC RR OE          AA(op1++) sz--
      repeat
          LC AC1            ds--        //if moving north
          // LC AC1 DR                    //if moving south
    while (sz)
```

MoveW (masked):

```
    // op1 -- LSB of source
    // op2 -- LSB of destination
    // sz -- size of operands
```

```
// ds -- distance
// ms -- exchange mask
LA RR OE      AA(ms)
LM
repeat
      LA RR OE                AA(op1++) sz--
      repeat
            LA AC1          ds--        // if moving west
            // LA AC1 DR        // if moving east
       while (ds)
      MC RR OE          AA(op2)
      LR OE RC          AA(op2++)
while(sz)
```

Processor to Processor Exchange (masked):

ExchangeNS (masked):

```
// op1 -- LSB of source
// sz -- size of operands
// ds -- distance
// nm -- northern mask
// em -- exchange mask
LC RR OE          AA(ms)
LM
```

```
repeat
    LC RR OE        AA(op1) sz-- (ds->dss) // load C
   repeat
        LC AC1          ds--          // north
    while (dss)
   RI AC0 LC                          // move C to A
    LC RR OE    AA(op1)    (ds->dss)     // load C again
    repeat
        LC AC1 DR          ds--  // south
   while (ds)
  LC RR OE          AA(nm)    // load western mask
 LM
  LC MC AC0        // load C from A in northen PEs
 LC RR OE          AA(em)   // load exchange mask
 LM
MC RR OE RC                 // load C masked if not exchange
  LR OE             AA(op1++)          // store A
while (sz)
```

Processor to Processor Crossover Exchange:

CrossOverNS:

```
// op1 -- LSB of source
// op2 -- LSB of source
// sz  -- size of operands
// ds  -- distance
// ms  -- northern mask
LC,RR OE            AA(ms)
LM
repeat
      LC RR OE      AA(op1) sz-- (ds->dss) // load A op1
     repeat
          LC AC1          dss--       // north
      while (dss)
     RI AC0 LA                        // move C to A
     LC RR OE     AA(op2)    (ds->dss)   // load C op2
     repeat
          LC AC1 DR       ds--        // south
      while (ds)
     LC MC RC      AA(op1)            // load C from AA(op1)
     LR OE         AA(op1++)          // store C
     LA MA       AA(op2)              // load A from AA(op2)
      LR OE        AA(op2++)              // store A
   while (sz)
```

Global Operations:

Global OR:

```
    // op1 -- parallel operand
    // op2 -- pointer into scalar operand register (SS)
  repeat
      LC RR OE          AA(op1++) sz--
      AC0 AC1 LC        SS(op2++)
  while (sz)
 repeat
      RC AC0 LA         AA(op1++) sz--
      AC0 AC1  LA       SS(op2++)
  while(sz)
```

Appendix B

Pin Arrangement for 3x3 Network Frame

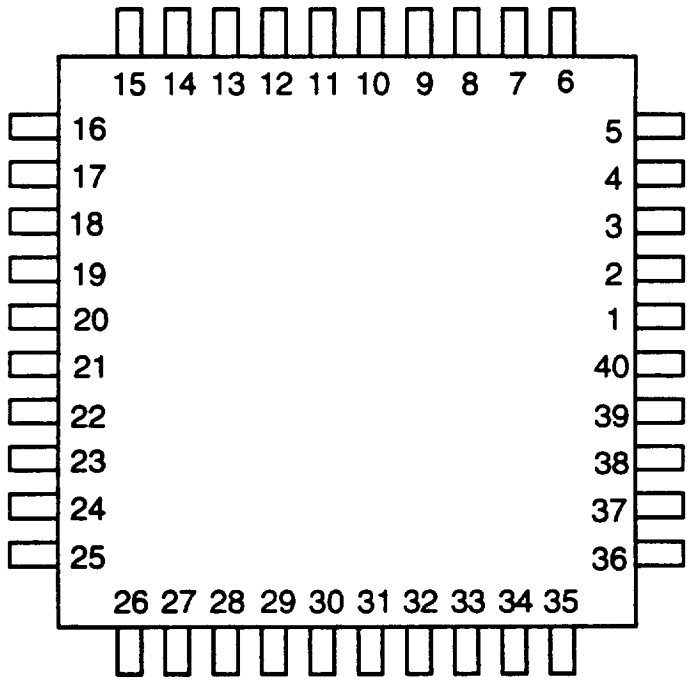


Figure B-1 Pin Arrangement on Frame

Appendix B. Pin Arrangement for 3x3 Network Frame

Pad #	Type	Signal
1	input pad	MA
2	input pad	MC
3	input pad	MM
4	input pad	MI
5	Ground pad	GND
6	output pad	Output 3
7	output pad	Output 2
8	output pad	Output 1
9	output pad	Ain6
10	output pad	Ain5
11	output pad	Ain4
12	output pad	Ain3
13	output Pad	Ain2
14	output pad	Ain1
15	power pad	Power (5v)
16	input pad	clock
17	input pad	pulse
18	input pad	memory enable
19	input pad	INc
20	input pad	LR
21	input pad	LM
22	input pad	LA
23	input pad	LC

Appendix B. Pin Arrangement for 3x3 Network Frame

24	input pad	AC0
25	input pad	AC1
26	output pad	Ain7
27	output pad	Ain8
28	input pad	Outc
29	input pad	Address 0
30	input pad	Address 1
31	input pad	Address 2
32	input pad	Address 3
33	input pad	Data input 1
34	input pad	Data input 2
35	input pad	Data input 3
36	input pad	RC
37	input pad	OC
38	input pad	DR
39	input pad	RR
40	input pad	RI

Appendix C

Model Paremeter of Transistor (Level 2)

```
* N55O SPICE LEVEL2 PARAMETERS
*  Revision  History:
* 9/18/95 - Added HDIF=2.5U and ACM=3 parameters...

.MODEL CMOSN NMOS LEVEL=2 PHI=0.700000 TOX=4.1000E-08
XJ=0.200000U TPG=1
+ VTO=0.8078 DELTA=2.7770E+00 LD=1.6250E-07 KP=4.5573E-05
+ UO=541.1 UEXP=1.4140E-01 UCRIT=1.1220E+05 RSH=2.51E+01
+ GAMMA=0.5901 NSUB=7.4410E+15 NFS=1.0360E+11
VMAX=5.5230E+04
+ LAMBDA=3.0850E-02 LAMBDA=3.0850E-02 CGDO=5.0529E-10
+ CGSO=5.0529E-10 CGBO=3.4581E-10 CJ=1.0113E-04 MJ=0.6944
+ CJSW=5.0290E-10 MJSW=0.3071 PB=0.800000
+ HDIF=2.5U ACM=0
* Weff = Wdrawn - Delta_W
* The suggested Delta_W is 2.0000E-09
.MODEL CMOSP PMOS LEVEL=2 PHI=0.700000 TOX=4.1000E-08
XJ=0.200000U TPG=-1
+ VTO=-0.9343 DELTA=3.6930E+00 LD=1.6800E-07 KP=1.6634E-05
+ UO=197.5 UEXP=2.4470E-01 UCRIT=9.1200E+04 RSH=5.75E+01
+ GAMMA=0.6572 NSUB=9.2300E+15 NFS=3.50E+11 VMAX=9.9990E+05
+ LAMBDA=4.0630E-02 LAMBDA=4.0630E-02 CGDO=5.1224E-10
+ CGSO=5.1224E-10 CGBO=4.0070E-10 CJ=3.2249E-04 MJ=0.5946
+ CJSW=2.9531E-10 MJSW=0.3594 PB=0.800000
+ HDIF=2.5U ACM=0
* Weff = Wdrawn - Delta_W
* The suggested Delta_W is -3.4120E-07
```

Appendix D

PE Verification Result

D.1 Input HSPICE file 1 for verification

```
* Power Supply
VSUPPLY 1 0 5V
* Functionality tests here are Store, Add (0+0), so that the schedule
* of cycles as folows,
*                0 --- 100ns Clear C
*                   100 --- 200ns LR 0 in (0001)
*                   200 --- 300ns Read 0 from (0001)
*                   300 --- 400ns Read 0 From (0001) and Add
*                   400 --- 500ns LR the sum (0) into (0001)

*       0 + 0 = 0 (Aout), 0 (Cout)

* 101 --- clock, 128--- Ain, 124 --- Aout, 119 --- Cin, 115 --- Cout,
* 474 --- cell1 (address 0001)
* 102 -- LR, 107 --- Output Enable, 216 --- Di,
```

```
Vclk 101 0 PU 0v 5v 0ns .1ns .1ns 50ns 100ns
Vpulse 110 0 PU 0v 5v 0ns .1ns .1ns 20ns 100ns

Vri 248 0 PWL 0ns 0v
Voc 255 0 PWL 0ns 0v 300ns 0v 300.1ns 5v 400ns 5v 400.1ns 0v
Vlc 190 0 PWL 0ns 5v 400ns 5v 400.1ns 0v
Vla 205 0 PWL 0ns 0v 200ns 0v 200.1ns 5v 400ns 5v 400.1ns 0v
Vlm 175 0 PWL 0ns 0v

Vrr 242 0 PWL 0ns 0v 200ns 0v 200.1ns 5v 400ns 5v 400.1ns 0v
Vac0 143 0 PWL 0ns 0v 200ns 0v 200.1ns 5v 400ns 5v 400.1ns 0v
Vac1 144 0 PWL 0ns 0v
Vrc 212 0 PWL 0ns 0v 200ns 0v 200.1ns 5v 300ns 5v 300.1ns 0v
Vdr 156 0 PWL 0ns 0v

Vma 209 0 PWL 0ns 0v
Vmc 194 0 PWL 0ns 0v
Vmm 179 0 PWL 0ns 0v
Vmi 162 0 PWL 0ns 0v

VOuten 107 0 PWL 0ns 0v 100ns 0v 100.1ns 5v
```

```
Vlr 102 0 PWL 0ns 0v 100ns 0v 100.1ns 5v 200ns 5v 200.1ns 0v 400ns 0v
400.1ns 5v
Va0 399 0 PWL 0ns 5v 400ns 5v
Va1 398 0 PWL 0ns 0v
Va2 397 0 PWl 0ns 0v
Va3 396 0 PWL 0ns 0v

VginNS 136 0 PWL 0ns 0v
VginEW 141 0 PWL 0ns 0v

* Analysis
.trans 25ns 500ns
.print tran Vclk
*Include the model file for the transistor arrays
.include "scna20-model.l2"
*Make the output available to gsi
.options POST
.end
```

D.2 Timing Diagram Corresponding to Input File 1

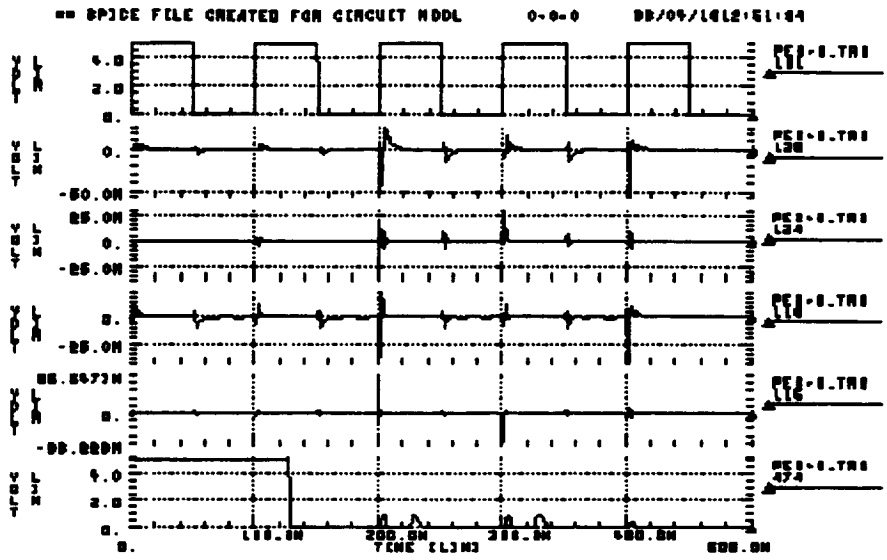


Figure D-1 Verification Result (file 1) of PE

D.3 Input HSPICE file 2 for Verification

```
* Power Supply
VSUPPLY 1 0 5V

* Functionality tests here are Store, Add (1+1), so that the schedule
* of cycles as folows,
*                  0 --- 100ns Store Aout (1) in cell1 (0001)
*                   100 --- 200ns Clear C
*                   200 --- 300ns Read 1 from (0001)
```

```
*                    300 --- 400ns Read 1 From (0001) and Add
*                    400 --- 500ns LR the sum (0) into (0001)

*        1 + 1 = 0 (Aout), 1 (Cout)

* 101 --- clock, 128--- Ain, 124 --- Aout, 119 --- Cin, 115 --- Cout,
* 474 --- cell1 (address 0001)
* 102 -- LR, 107 --- Output Enable, 216 --- Di,

Vclk 101 0 PU 0v 5v 0ns .05ns .05ns 50ns 100ns
Vpulse 110 0 PU 0v 5v 0ns .05ns .05ns 20ns 100ns

Vri 248 0 PWL 0ns 5v 100ns 5v 100.1ns 0v
Voc 255 0 PWL 0ns 0v 299.9ns 0v 300ns 5v 399.9ns 5v 400ns 0v

* Note that signals LA and LC come up 2ns ealier intentionally
* to cancel the burst caused by XOR gates in local control logic
Vlc 190 0 PWL 0ns 0v 97.9ns 0v 98ns 5v 397.9ns 5v 398ns 0v
Vla 205 0 PWL 0ns 0v 197.9ns 0v 198ns 5v 397.9ns 5v 398ns 0v

Vlm 175 0 PWL 0ns 0v
Vrr 242 0 PWL 0ns 0v 199.9ns 0v 200ns 5v 399.9ns 5v 400ns 0v
Vac0 143 0 PWL 0ns 0v 199.9ns 0v 200ns 5v 399.9ns 5v 400ns 0v
Vac1 144 0 PWL 0ns 0v
```

```
Vrc 212 0 PWL 0ns 0v 199.9ns 0v 200ns 5v 299.9ns 5v 300ns 0v
Vdr 156 0 PWL 0ns 0v

Vma 209 0 PWL 0ns 0v
Vmc 194 0 PWL 0ns 0v
Vmm 179 0 PWL 0ns 0v
Vmi 162 0 PWL 0ns 0v

VOuten 107 0 PWL 0ns 5v 99.9ns 5v 100ns 0v 199.9ns 0v 200ns 5v
Vlr 102 0 PWL 0ns 5v 99.9ns 5v 100ns 0v 399.9ns 0v 400ns 5v
Va0 399 0 PWL 0ns 5v 199.9ns 5v
Va1 398 0 PWL 0ns 0v
Va2 397 0 PWl 0ns 0v
Va3 396 0 PWL 0ns 0v

VginNS 136 0 PWL 0ns 0v
VginEW 141 0 PWL 0ns 0v

* Analysis
.trans 25ns 500ns
.print tran Vclk
*Include the model file for the transistor arrays
.include "scna20-model.l2"
*Make the output available to gsi
```

.options POST

.end

D.4 Timing Diagram Corresponding to Input File 2

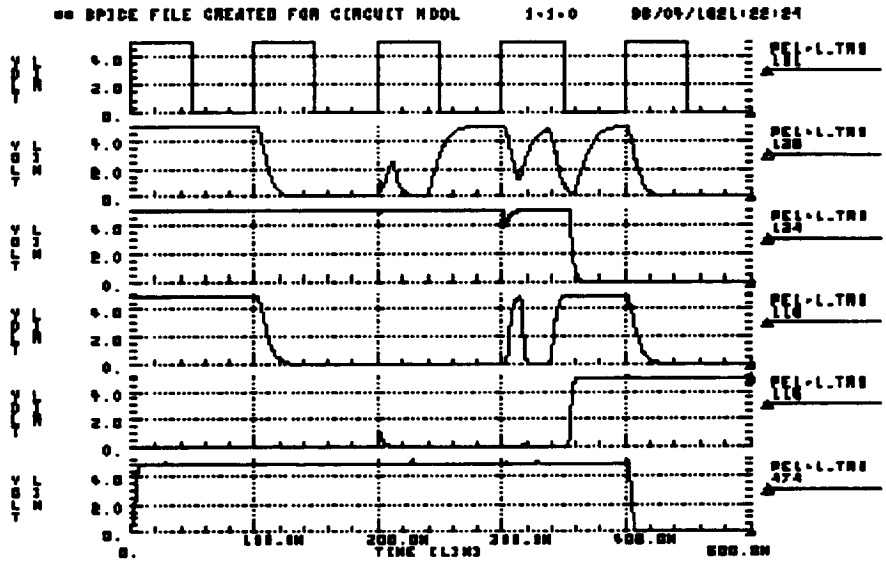


Figure D-2 Verification Result (file 2) of PE

Appendix E

3x3 Network Verification Result

E.1 Input HSPICE file

```
* Power Supply
VSUPPLY 1 0 5V

* Input signals ...
*      0---300ns  input  1  0  0
*                        0 1 0    into PE array
*                        0 0 1
*         300 --- 400ns store the array
*        400 --- 500ns LC RR      (global OR begins)
*         500 --- 600ns AC0 AC1 LC (global OR continues)
*       600 --- 700ns RC AC0 LA (global OR continues)
*        700 --- 800ns AC0 AC1 LA (global OR ends)
*       800 --- 900ns store global OR into cell1 (address 0001)

Vclk 296 0 PU 0v 5v 0ns .1ns .1ns 50ns 100ns
Vpulse 289 0 PU 0v 5v 0ns .1ns .1ns 20ns 100ns

Vri 226 0 PWL 0ns 0v
Voc 184 0 PWL 0ns 0v *500ns 0v 500.1ns 5v 600ns 5v 600.1ns 0v

*Note that LC and LA come up 2 ns ealier for cancelling the Xor gate
delay
Vlc 205 0 PWL 0ns 0v 397.9ns 0v 398ns 5v 597.9ns 5v 598ns 0v
```

```
Vla 219 0 PWL 0ns 5v 297.9ns 5v 298ns 0v 597.9ns 0v 598ns 5v
797.9ns 5v 798ns 0v
Vlm 233 0 PWL 0ns 0v

Vrr 212 0 PWL 0ns 0v 400ns 0v 400.1ns 5v 500ns 5v 500.1ns 0v
Vac0 191 0 PWL 0ns 0v 500ns 0v 500.1ns 5v 800ns 5v 800.1ns 0v
Vac1 177 0 PWL 0ns 5v 300ns 5v 300.1ns 0v 500ns 0v 500.1ns 5v
600ns 5v 600.1ns 0v
+ 700ns 0v 700.1ns 5v 800ns 5v 800.1ns 0v
Vrc 170 0 PWL 0ns 0v 600ns 0v 600.1ns 5v 700ns 5v 700.1ns 0v
Vdr 198 0 PWL 0ns 5v 300ns 5v 300.1ns 0v

Vma 240 0 PWL 0ns 0v
Vmc 254 0 PWL 0ns 0v
Vmm 268 0 PWL 0ns 0v
Vmi 282 0 PWL 0ns 0v

Venabl 275 0 PWL 0ns 0v 300ns 0v 300.1ns 5v 500ns 5v 500.1ns 0v
800ns 0v 800.1ns 5v
Vlr 247 0 PWL 0ns 0v 300ns 0v 300.1ns 5v 400ns 5v 400.1ns 0v
800ns 0v 800.1ns 5v
Va0 142 0 PWL 0ns 5v
Va1 135 0 PWL 0ns 0v
Va2 128 0 PWl 0ns 0v
Va3 121 0 PWL 0ns 0v

Vin0 114 0 PWL 0ns 5v 100ns 5v 100.1ns 0v
Vin1 107 0 PWL 0ns 0v 100ns 0v 100.1ns 5v 200ns 5v 200.1ns 0v
Vin2 100 0 PWL 0ns 0vv 200ns 0v 200.1ns 5v
Vinc 261 0 PWL 0ns 5v 300ns 5v 300.1ns 0v
Voutc 149 0 PWL 0ns 0v

* A big resistor needed to be connected on 80#
r1 106 0 100k
r2 113 0 100k
r3 120 0 100k
r4 127 0 100k
r5 134 0 100k
```

```
r6  141  0  100k
r7  148  0  100k
r8  155  0  100k
r9  176  0  100k
r10  183  0  100k
r11  190  0  100k
r12  197  0  100k
r13  204  0  100k
r14  211  0  100k
r15  218  0  100k
r16  225  0  100k
r17  232  0  100k
r18  239  0  100k
r19  246  0  100k
r20  253  0  100k
r21  260  0  100k
r22  267  0  100k
r23  274  0  100k
r24  281  0  100k
r25  288  0  100k
r26  295  0  100k
r27  302  0  100k


*  Analysis
.trans  25ns  900ns
.print  tran  Vclk

*Include  the  model  file  for  the  transistor  arrays
.include  "scna20-model.l2"

*Make the output available to gsi
.option POST

.end
```

E.2 Timing Diagrams

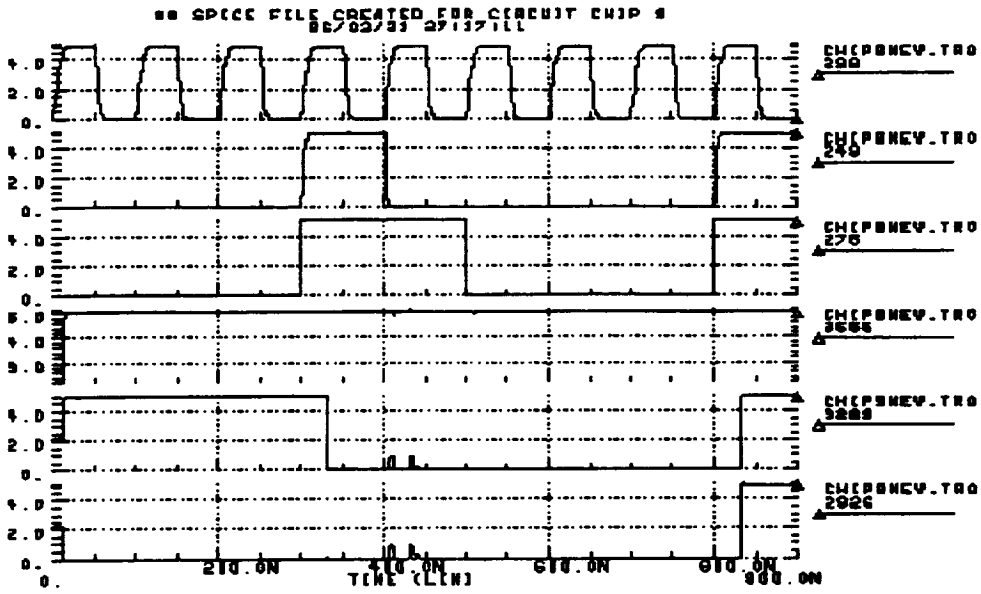


Figure E-1. Verification Result of 3x3 Network

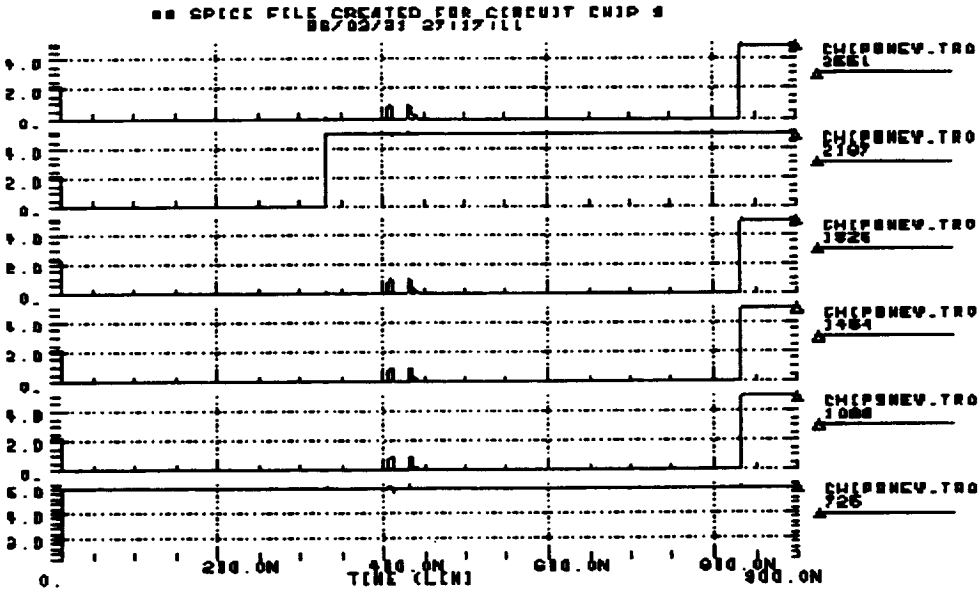


Figure E-2 Verification Result of 3x3 Network (continued)