Design and VLSI Implementation of an Address Generation Coprocessor *

Paul T.Hulina, Lee D.Coraor
Lizyamma Kurian and Eugene B.John+

Department of Electrical and Computer Engineering
The Pennsylvania State University
University Park, PA. 16802
(814)-865-1265

ldc@ecl.psu.edu

Abstract

Most applications of general purpose VLSI processors are developed using high level languages. In these languages, information is generally handled in a structured form. Compilers generate a considerable amount of code to navigate through the data structures and considerable processing time is spent performing address calculations required to access the data structures. In this paper, an alternative to software address generation, a hardware memory reconfiguring unit or an address generation coprocessor is presented.

To demonstrate the VLSI feasibility of the designed device, it is implemented in VLSI using the *Octtool* suite of tools. The tools used and the implementation procedure are described. VLSI design aspects such as regularity, modularity, scalability etc are discussed. It is observed that this chip is suitable for fault–tolerant design incorporating some redundancy and for wafer–scale integration.

The performance of the device is evaluated using assembly language programs that implement popular signal processing algorithms such as convolution, correlation, FFT and matrix multiplication. A system with the address generation unit exhibits a speed up of between approximately 1.5 and 2.5.

Index Terms: VLSI Implementation, Efficient Memory Access, Memory reconfiguring unit, Address Generation, Access Overhead, Primitive Address Sequences.

** This work was supported in part by the National Science Foundation under grant number MIP–8912455.*

1. INTRODUCTION

Modern general purpose VLSI processors are designed with the objective of a high average performance for a wide range of applications. However, the efficient solution of some tasks demands the enhancement of the processing capabilities of the processor (e.g. floating point computations). In spite of the advances in microelectronic technology, the addition of functional units to the VLSI processor is restricted by the limited amount of real estate available. A viable alternative for increasing computer performance is based on the use of special purpose units which take some of the computation burden from the main processor. Floating-point processors, CRT controllers, disk controllers, and DMA controllers, are but a few examples of currently available devices. In some cases, these units are designed to perform certain restricted data processing tasks in a very efficient manner. In other cases, they help in the management of the computer system resources. For example, memory management units (MMUs) assist the main processor in controlling the elements of the memory hierarchy of a computer system, which consists of a cache, primary storage, and secondary storage.

Most present-day computer applications are developed using structured high level languages (HLLs) such as Pascal, Ada, etc. In an HLL, information is generally handled in a structured form, e.g. as procedures and data structures. In general, HLL compilers will generate a considerable amount of code just to navigate through the data structures. This fact aggravates the problem of the addressing overhead. Efficient address generation for data structures is therefore a primary goal in order to obtain high performance in a computer system. This is particularly true in a Reduced Instruction Set Computer (RISC) environment where address calculations are almost exclusively performed in software since available addressing modes in a RISC machine are limited. Another area where address computation is important is that of digital signal processing (DSP). Significant effort has been focused on the design of special purpose processors [1,3]. More recently, Nwachukwu [12] has proposed an array indexing unit to be used for address generation in an array processor to produce a system which is more versatile than conventional FFT array processors. Although these machines each have their own advantages, each is designed for a particular set of tasks. The success of the current special purpose machines is strongly determined by the intended task domains. Hence the machines tend to be well suited for one particular application, but are usually inflexible.

Instead of building special purpose architectures, some designers have concentrated their efforts on using general purpose processors to achieve versatility at the expense of having to generate addresses in software. One example is the fast Fourier transform algorithm and its many variations [2,4]. Although these algorithms improve the efficiency of the various DSP operations, a significant amount of time is wasted in address calculations since a general purpose processor is used.

1.1 Objective

In this paper we first describe a hardware unit that is tailored to function more efficiently with data structures associated with DSP applications. This device can in fact be viewed as a memory reconfiguring unit (MRU) or a coprocessor for accomplishing address transformations.

The objective is to design an MRU which is easy to interface between the host processor and memory and which would not require any modifications either to the host processor or the memory. Special opcodes have not been introduced into the instruction repertoire nor does the design presume any modifications to the existing operating system. The function of a unit such as the MRU is to provide the CPU with a set of specialized addressing modes as defined by algorithms which are frequently used in several major applications.

Then the details of the VLSI implementation of the MRU are presented. The *Octtools* tool suite [13] is used for the design. The feasibility of a VLSI implementation is thus demonstrated.

Finally the performance of MRU is evaluated using popular signal processing algorithms such as convolution, correlation, FFT etc. These algorithms are selected because they utilize different address sequences. The performance of a computer system with the MRU and one without the MRU are evaluated to illustrate the speed factor contributed by the MRU.

2. PRIMITIVE ADDRESS SEQUENCES

The approach which has been used here is to classify a desired address sequence into a composition of a series of "primitive" address sequences. A different address sequence may then be generated by specifying a different composition of these primitive address sequences. The selection of the address sequences is based on a survey of frequently used algorithms for the solution of real-time DSP problems. They are summarized in Table 1. Based on an addressing sequence of length N, if A_i is address i contained in the range $0 \leq i \leq N-1$, the primitive addressing sequences are defined as follows:

(1) Sequential addressing

$A_0, A_1, A_2, ..., A_{N-1}$

(2) Sequential with offset (k) addressing

$A_{0+k}, A_{1+k}, A_{2+k}, ..., A_{N-1+k}$

(3) Shuffled addressing (base r, N/r=p)

$A_0, A_p, A_{2p}, ...A_1, A_{p+1}, A_{2p+1}, ...A_2, A_{p+2}, A_{2p+2}, ..., A_{N-1}$

(4) Bit-reversed addressing (eg. N=8)

$A_0, A_4, A_2, A_6, A_1, A_5, A_3, A_7$

(5) Reflected addressing

$A_0, A_{N-1}, A_1, A_{N-2} .....A_i, A_{N-i} ...., A_{N/2-1}, A_{N/2}$

The MRU or address generation unit that is designed and implemented in this paper is capable of generating these addressing sequences in hardware.

3. DESCRIPTION OF THE MRU SYSTEM

Figure 1 illustrates the manner in which the proposed MRU is interfaced to a processor. After the introduction of the MRU, the signal flow between the processor and memory is through the MRU and dictates that the MRU be transparent to normal signal flow. No appreciable delay should be introduced when instructions or unmapped data are to be fetched from memory. The system functions in one of three modes, depending on the address sent out by the CPU. (The term mapped data refers to data sequences which need to be accessed in specified addressing patterns while unmapped data refers to data which need not.)

The MRU is a memory mapped device in which each user accessible register is assigned a unique memory address. When the CPU writes to any of these MRU registers the MRU is placed in the "INIT" mode for one instruction cycle. The CPU thus initializes the MRU registers to set up the desired primitive address sequences.

If the host processor accesses data which is not to be mapped or if it fetches an instruction, the MRU is in the "PASS" mode and lets the signals, data, and address, to pass through to the memory/processor unaltered. Only a decode delay is introduced by the MRU in order to determine the memory space being addressed, whether mapped or unmapped. If a computer system with a Harvard architecture** is being employed, the instruction memory could be directly connected to the processor with the MRU being introduced only in the data path. In such a case, even the decode delay will not affect the fetching of instructions.

When the host wants to access mapped data it sends out a base address which is recognized by the MRU as one of the base addresses specified during initialization. The MRU enters the "MAP"mode for one instruction cycle. Each time a particular primitive address sequence is referenced the CPU will send out the same base address. The base address thus indicates the address sequence to be generated by the MRU and corresponds to the location at which the first piece of mapped data is stored for this pattern.

The name memory reconfiguring unit or MRU is used because by specifying an addressing pattern in a short initialization routine (with the MRU in the INIT mode), and by specifying a base address whenever the address pattern is to be used (with the MRU in the MAP mode), the MRU makes the memory appear reconfigurable. The MRU provides a set of specialized hardware addressing modes so that different addressing sequences may be generated.

4. HARDWARE DESCRIPTION

A functional representation of the address generation unit is shown in Figure 2. Primitive Generation Units (PGUs) are the basic components in the MRU and are used to generate the specific address pattern required. The Primitive Generation Unit is combined with additional circuitry that controls the PGUs to yield the Offset Generation Unit. If several data structures are being simultaneously referenced in the same application, each of them will need an OGU. Several OGUs can be integrated to yield the Offset Generation Module (OGM). Sometimes several arrays or matrices may be stored in contiguous memory and may be accessed using just one OGU, depending on the programming technique. But such techniques may not be applicable in all cases, and many applications require several OGUs. A Decode and Control Logic Unit (DCLU) determines the system mode and is also used to trap/regenerate signals to/from memory.

4.1 Decode and Control Logic Unit (DCLU)

The DCLU determines the MRU operating mode and provides control of signal flow between the host processor and the memory. The mode selection and data switching portion of the Decode and Control Logic Unit (DCLU), as shown in Figure 3, determines the system mode by decoding the address lines sent by the host processor. Figure 4 is the functional diagram of the Signal Flow Control (SFC) unit which is responsible for coordinating all the signal flow. This unit basically performs a multiplexing function. For example, the addresses to memory come from either the CPU or the OGU depending on whether or not address mapping is to be done. The operation of the DCLU can be described as follows [8] [14].

PASS MODE: When an address associated with either an instruction or unmapped data is sent by the host processor, the DCLU decodes the address in order to determine that the MRU is in the PASS mode. Consequently the SFC allows the address, data and control signals to pass through with only a decode delay.

INIT MODE: If the address sent by the host processor indicates the INIT mode, the DCLU will send the subsequent data to the OGU in order to initialize the appropriate registers and counters. Typically the data includes the length of the sequence, the type of sequence, constant offset and the other control words. During the INIT mode, base address registers are loaded with base addresses by the SFC. The address indicating the INIT mode is sent out repeatedly until all data required for initializing the MRU have been written.

MAP MODE: A mappable base address is decoded by the DCLU which compares the coming address with previously stored base addresses. If a match is found, the SFC intercepts the R/W signal and holds the data bus until the mapped address is generated. After the mapped address generated by the OGU is sent to the SFC, it enables the R/W signal and the data flow. Every time, the base address is received, the DCL activates the OGU and the appropriate segment of the PGU to generate the requested addressing sequence.

4.2 Offset Generation Module (OGM)

The Offset Generation Module (OGM) consists of several Offset Generation Units (OGUs) (three in Fig. 2). Each Offset Generation Unit consists of a Primitive

Generation Unit (PGU), a control word register, an adder and a register. The control word register contains the control information necessary to activate each PGU.

The control word registers are loaded by the host processor while the MRU is in the INIT mode. In the adder, the base address from the base address register will be added with the element of the requested addressing sequence from the PGU to provide the final mapped address. The final mapped address is then routed back to the SFC in the DCL, which then generates the R/W signal and puts the address on the address bus and passes the contents of the data bus to/from the memory.

4.2.1 Primitive Generation Unit (PGU)

The PGU is the most essential part of the OGU. It consists of four different hardware units which are responsible for generating the different primitive addressing sequences outlined in section 2.

(1) *Sequential addressing sequences* can be generated by a simple parallel-input parallel–output up/down counter.

(2) *Sequential with offset (k) addressing sequences* can be generated by an up/down counter and an adder/subtractor unit. One of the two operands is obtained from the sequential counter and the other is obtained from the constant offset k stored in a register which has been previously loaded while the MRU was in the INIT mode.

(3) *Shuffled and bit–reversed addressing sequences* can be generated by a counter-multiplexer combination unit as shown in Figure 5. To generate shuffled addressing for a sequence of $N = 2^n$, the output of an n-bit counter is input to n, n-to-1 multiplexers. Each multiplexer is controlled by $log_2 n$ control bits corresponding to the different base of the required shuffled addressing sequence and thus selects one of n inputs for generating an n-bit word. For the bit-reversed addressing sequence, the control bits in the unit are simply set to select the counter output bits in reversed order, i.e. counter outputs $C_{n-1}, C_{n-2}, ...C_1, C_0$ are selected in the order $C_0, C_1, ...C_{n-2}, C_{n-1}$.

(4) *Reflected addressing sequences* may be generated by a hardware unit as shown in Figure 6. In the reflected address generation unit, the counter is incremented every other clock cycle. For a counter value "i", the i^{th} element A_i is fetched and on the next cycle, "i" is subtracted from the constant $(N-1)$ to fetch the element A_{N-1-i}.

5. VLSI IMPLEMENTATION

We implemented a scaled down version of the address generation unit described above. This version has 8-bit address and data lines. Our purpose is to implement the 8-bit version and then analyze the scalability of the design to find out how much more complex a 32-bit version would be. We can thus study the VLSI feasibility of our design.

5.1 VLSI Tools

The chip was developed using VEM and the OCT tool suite [13]. VEM is an interactive graphics editor for designs represented using the OCT Data manager. VEM supports physical and symbolic editing of IC designs, as well as schematic capture. The OCT tools consist of a complete set of tools to facilitate the design process.

In OCT, the basic design unit is the cell. Each cell is an arbitrarily complex portion of the design and may include other cells in a hierarchical fashion. Cells can be initially described in a Pascal-like high level language *bds*. Then the *bds* functional description is translated to logic equations using *bdsyn*. The logic is optimized and mapped into standard cells by a multiple-level-logic synthesizer *misII*. Then *wolfe* will randomly assign locations for all terminals and pass the information to *TimberwolfSC* which is a standard cell place and route program. Minimizing the number of vias in a design will improve the yield and *mizer* performs this task.

BDSYN is a tool for quickly describing and implementing combinational logic. But it can be used for describing combinational logic blocks only (blocks which do not use signal latching of any kind). Describing a finite state machine requires the logic designer to add external latches to the described combinational logic blocks. If the designer wants to specify the logic from the gate level, logic gates and flip-flops, these can be interconnected with *bdnet*. Thus *bdnet* yields the exact hardware that is in the designer's mind. Again, during hierarchical design, *bdnet* is the tool used to integrate the sub-modules to yield larger modules and finally the chip. *Mosaico* is a complete set of tools for floorplanning, placement, and routing of macro-cells. Macro cells are functional blocks that are created by module generators, manual layout or any number of other means. Mosaico consists of five basic steps executed in sequence as a pipeline: floorplanning and placement, channel definition, global routing, detailed routing, and compacting. All these tools can be used to yield area efficient IC designs easily.

5.2 Implementation Details

The elements of the DCLU such as the Decode unit and the SFC section were realized using gates in the standard cell library. The standard cell based designs are easy to modify. The SFC unit is constructed in a bit-slice fashion to take advantage of the inherent regularity. After the SFC elements for one bit are constructed, the instances are just replicated to yield the SFC for the whole bus.

To construct the Offset Generation Module, first the four circuits to generate the different address sequences were independently realized. They were integrated together with a multiplexer to yield the Primitive Generation Unit (PGU). The various registers in the OGU including control word registers were wired using flip-flops from the cmos library and they were integrated with the PGU and the adder to yield an Offset Generation Unit. The OGUs can then be replicated resulting in the Offset Generation Module and the OGM can be integrated with the DCLU to obtain the final chip. During the integration, the OGUs should be placed in such a way that the wires from DCLU to each OGU can be routed with minimum complexity.

The implementation of the MRU with one OGU is shown in Figure 8. One OGU consumes less than $2mm$ X $2mm$ of chip area if fabricated using 1 micron technology. More OGUs may be integrated to the unit in Figure 8 to yield a full capability MRU. A layout with three OGUs, according to the floor plan in Fig. 9 is shown in Figure 10. This version with 3 OGUs can be fabricated on a 4 mm X $4mm$ die, (assuming 1 micron technology).

5.3 VLSI Aspects

There are several important aspects to bear in mind when designing an architecture for VLSI implementation; the major ones are regularity, modularity, scalability, chip area etc.

VLSI designs should be regular structures since the time and effort required to produce a design depends on the number of different elementary cells required more than on the total number of transistors used. For instance, it is quicker to produce a large register bank by replicating a properly designed single register bit than generating a random logic function which uses far fewer transistors. Random logic can also be implemented in regular structures such as the PLA or ROM, but an implementation based on gates has higher speed. The intermediate approach is to use a cell library that consists of standard gates, latches, flip-flops etc. The designer picks up appropriate cells and wires them together. This might not result in the smallest possible layout, but the speed can be good. Modular designs help in easy debugging and facilitate modifications and future expansions.

Efforts should be made to reduce the area of a chip. The cost of manufacture of an integrated circuit depends on chip area (A). More important is the fact that die defects make costs increase more rapidly than O(A) and chips above a certain area cannot be practically manufactured. Pin requirements of a chip is another important determinant of chip cost. Every wire that connects a pin on the package to a bond pad on the chip is a potential source of a defect and the yield of packages with large numbers of pins is usually small. So chips with large numbers of pins are generally expensive.

The various elements of the MRU are designed keeping these aspects in mind. We exploited all regularity in building up the chip. The SFC Unit was constructed in a bit slice fashion to take advantage of the regularity. The SFC elements for a single bit were first designed and then it was replicated to yield the whole SFC Unit. In the Offset Generation Module (OGM) also, there are several copies of the OGU and associated hardware. Careful custom design of one OGU was done. This OGU may be replicated to create the OGM.

The design was done in a highly modular fashion. This helped in developing the chip, module by module, doing sufficient testing at the module level before integrating and removing bugs at the sub–module level itself. Another factor that had to be taken care of was the I/O requirement for each sub-module. We designed modules in such a way that the area for interconnection could be minimized.

Regarding area/time trade-offs, reducing time was the main emphasis. Despite the fact that area had to be minimum, we traded area for speed wherever the choice had to be made. Since the device is to be introduced between the CPU and the memory, speed is of primary concern.

Another aspect in a VLSI implementation is scalability. It is particularly important here, since the original design was for a 32-bit processor, but only an 8-bit version was implemented. From the 8-bit implementation, we can speculate about the complexity of the 32-bit chip. The 8 bit DCLU can be expanded to a 32-bit version with linear increase in complexity. The 32-bit OGU can also be realized with O(n) complexity. Hence we may conclude that the chip can be constructed with linear increase in complexity. Another factor in scalability is the complexity with an increasing number of Offset Generation Units (OGUs). Figure 10 incorporates three OGUs. More OGUs can be incorporated with a linear increase in complexity.

Designing circuits with testability aspects in mind is also important since testing costs can be a significant portion of the total device costs. The modular and regular structure of the chip makes fault tolerant design with redundant OGUs very easy. This chip is suitable for wafer–scale integration also. The number of OGUs to be incorporated in the OGM can be decided at the wafer–scale integration stage.

6. PERFORMANCE EVALUATION

The usefulness of the MRU device may be illustrated with several frequently used digital signal processing algorithms such as convolution, correlation, fast Fourier transform, and matrix multiplication. To illustrate how the MRU can be used for these algorithms, a detailed analysis of one of the algorithms (convolution), is provided.

Algorithm Analysis

If $A(x)$ and $B(x)$ are functions defined on $(-\infty, \infty)$, the convolution, $C(x)$ of A and B, is the function defined by [5]:

$$C(x) = \lim_{T \to \infty} \int_{-T}^{T} A(t)B(x-t)\,dt$$

For convenient numerical calculation, the discrete convolution is typically used. The discrete convolution of two sequences A and B with equal length N is the set of numbers $C_i, i = 0, 1, ..., N-1$, defined by [9] as

$$C_i = \sum_{j=0}^{i} A_j B_{i-j}$$

The algorithm can be written as:

for (i=0 to N-1) do
$C_i = 0;$
for (j=0 to i) do
$C_i = C_i + A_j B_{i-j};$

Actually there are only two simple addressing sequences, sequential addressing and sequential with offset addressing, used for sequences A, B, and C in the above algorithm. But the addressing sequence for A and B can be treated as reflected addressing when the sequences of A and B are stored in consecutive memory locations. The address of the first item of A is loaded as the base address and for every iteration, the value $(N+i+1)$, is used to reset the reflected address generation unit as the length of that sequence. For example, when $N = 3$,

(1) In the first iteration (i=0), the reflected addressing sequence is 0,3. The actual data fetched are A_0, B_0.
(2) In the second iteration (i=1), the reflected addressing sequence is 0, 4, 1, 3. The actual data fetched are A_0, B_1, A_1, B_0.
(3) In the third iteration (i=3), the reflected addressing sequence is 0, 5, 1, 4, 2, 3. The actual data fetched are $A_0, B_2, A_1, B_1, A_2, B_0$.

This implementation requires the use of only two base address registers and two OGUs, one for sequence A and B together and one for sequence C.

Performance Analysis

The four signal processing algorithms were implemented on an HP/UNIX system using assembly language. The time required for execution of the programs was calculated based on the assembly language program. The convolution and correlation programs deal with fixed–point arithmetic operations. The FFT and matrix multiplication programs operate on floating–point data. The MC68020 microprocessor is used as the host processor and the MC68881 floating point coprocessor is employed when needed in both the cases with the MRU and without the MRU. Calculation of the execution time is based on the MC68020 and the MC68881 instruction execution times in terms of external clock cycles. There are three values given for each instruction [10] [11].

(1) Best Case – The best case reflects the time when the instruction is in the cache and benefits from the maximum overlap due to other instructions.
(2) Cache Case – When the instruction is in the cache but has no overlap, the value of the cache-only-case is employed.
(3) Worst Case – The worst case reflects the time when the instruction is not in cache or the cache is disabled and there is no instruction overlap.

Table 2 gives equations which specify the execution time for the four algorithms studied. T_{MRU} and T_{NMRU} represent the execution time in clock cycles for the system with the MRU and without the MRU respectively. The equations in Table 2 consist of $N^3, MN^2, MN, Nlog_2N, N, log_2N$, and constant terms. The

N^3, MN^2, N^2, MN, and $Nlog_2N$ terms are due to nested loops with different boundary values. The N and log_2N terms result from single loops. The length of the data sequences in the convolution, correlation, and FFT programs is N. The dimension of the matrix in the matrix program is MXN, with M not necessarily equal to N. The programs included in this study are not the only way to implement the algorithms. Therefore the values of execution time obtained are not absolutely exact. The execution times in systems with and without the MRU may be used to calculate the speed up contributed by the MRU. We define a speed up factor, T_{NMRU} divided by T_{MRU}, to indicate the improvement contributed by the MRU. Figure 11 illustrates the variation in speed up factor as the length of the data vector N varies. As shown in Table 3, for large values of N, the speed up varies between approximately 1.5 and 2.4 for the four algorithms.

7. CONCLUSION

We have presented the design, VLSI implementation, and performance evaluation of an MRU, a hardware unit that performs the address generation task required during access of data structures, especially in signal processing applications. The operation of the device, the hardware modules involved, the design of the various functional units etc are described. Various addressing sequences involved in signal processing algorithms are analyzed and hardware necessary to generate them are designed. The whole design is targeted for a VLSI implementation and the various trade-offs involved in the design process are explained.

Details of a VLSI implementation of the MRU are then presented. A scaled down version of the design is implemented using the *Octtools* tool suite. Performance of the device is then evaluated with four frequently used algorithms from digital signal processing. The algorithms used are convolution, correlation, FFT and matrix manipulation. A computer system employing the MRU can speed up these algorithms by factors of as much as 2.4 compared with a system without the MRU. The high performance results from reduction in the overhead associated with repetitive calculation of addresses in software.

The MRU system can be modified in hardware to adapt to unique environments found in digital signal processing applications and also to more general demands required in general purpose computing.

Acknowledgements

The authors wish to express their gratitude to S.W.Sun for the help in performance evaluation of the MRU, and to Dr.Martin Brady for his help in using the VLSI tools.

References

1. J. Allen, "Computer Architecture for Signal Processing", Proc. of the IEEE, Vol.63, No.4, April 1978.
2. R.E.Blahut, Fast Algorithm for Digital Signal Processing, John Wiley, UK, 1985.
3. B.A.Bowen and W.R.Brown, VLSI Systems Design for Digital Signal Processing, Prentice Hall, NJ. 1985.
4. C.S.Burrus and T.W.Parks, Topics in Digital Signal Processing, John Wiley, UK, 1985.
5. C-T Chen, System and Signal Analysis, Holt, Rinehart and Winston, Inc., New York, N.Y. 1989.
6. S.B.Furber, "VLSI RISC Architecture and Organization", Marcel Dekker Inc., New York and Basel, 1989.
7. P.T.Hulina and L.D.Coraor, "Coprocessor Architecture for Efficient Address Computation and Memory Accessing", Computer Systems Science and Engineering, Vol.5, No.3, July 1990.
8. P.T.Hulina, L.D.Coraor and S.W.Sun, "Performance Analysis of an Address Generation Coprocessor", Proc.Intl. Conf. on Parallel Processing, Vol. I, pp.136-143, 1991.
9. D.Kahaner, C.Moler, and S.Nash, Numerical Methods and Software, Prentice Hall, Englewood Cliffs, N.J. 1989.
10. Motorola Inc., MC68020 32–bit Microprocessor User's Manual, Second Edition, Prentice Hall, Englewood Cliffs, N.J., 1985.
11. Motorola Inc., MC68881 Floating Point Coprocessor User's Manual, First Edition, Prentice Hall, Englewood Cliffs, N.J. 1985.
12. E.O.Nwachukwu,"Address Generation in an Array Processor", IEEE Trans. on Computers, Vol C-34, No.2, February 1985.
13. OCT tools Users' Guide, University of California, Berkeley, 1990.
14. V. Singhal, P.T.Hulina and L.D.Coraor, "A Memory Reconfiguring Unit for versatile Work-Stations", 1986 IEEE Workstation Technology and Systems Conference.
15. R.D. Strum and D.E.Kirk, First Principle of Discrete Systems and Digital Signal Processing, Addison-Wesley, Mass, 1988.