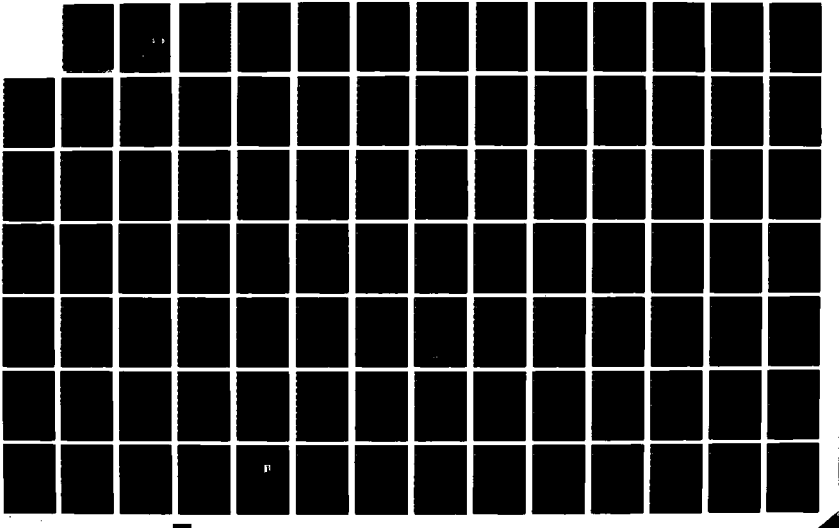
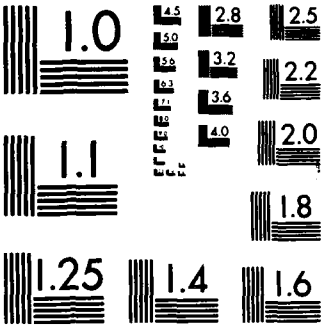
AD-A184 603 SBNR (SIGNED BINARY NUMBER REPRESENTATIONS) DIGITAL SIGNAL PROCESSOR ARCHITECTURE(U) SPACE TECH CORP FORT COLLINS CO M ANDREWS ET AL 31 MAY 87 ARO-20192 7-EL-S 1/2

UNCLASSIFIED DAAG29-83-C-0025 F/G 12/7 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ARO 20192.7-EL

(2)

(ARO) Contract Report No. FTR DAAG29-83-C-0025

FINAL TECHNICAL REPORT

SBNR DIGITAL SIGNAL PROCESSOR ARCHITECTURE

by

Michael Andrews
David James

Period Covered: September 1, 1983 - April 30, 1987

for

U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

May 31, 1987

SPACE TECH CORPORATION
2324 Manchester Court
Fort Collins, CO 80526

SPACE TECH

The views, opinions, and/or findings contained in this report are those of the author(s) and should not be contrued as an official Department of the Army position, policy, or decision, unless so designated by other official documentation.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD-A184 603

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
ARO 20192.7-EL-S	NA	NA

4. TITLE (and Subtitle)

R&D Project No. & Title:
1L161102BH5703 Electronics

SBNR Digital Signal Processor Architecture

5. TYPE OF REPORT & PERIOD COVERED

Final Technical Report
Sept. 1, 1983 - May 30, 1987

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

Dr. Michael Andrews, David James

8. CONTRACT OR GRANT NUMBER(s)

DAAG29-83-C-0025

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Space Tech Corporation
2324 Manchester Court
Fort Collins, CO 80526

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

NA

11. CONTROLLING OFFICE NAME AND ADDRESS

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park, NC 27709-2211

12. REPORT DATE

May 31 , 1987

13. NUMBER OF PAGES

144

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

Defense Contract Audit Agency
Denver Branch Office
721 19th Street, New Custom House, Room 474
Denver, CO 80202

15. SECURITY CLASS. (of this report)

UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

NA

18. SUPPLEMENTARY NOTES

The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Adaptive Signal Processor, Redundant Numbers, Beamformer, Radar, Sonar

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The following research was proposed in a three-year period. This constitutes significantly distinct efforts which complement the existing efforts in current adaptive signal processor architecture research. Briefly, these tasks comprised a study of non-conventional number system implementations focusing on VLSI enhancements attributable to redundant number systems. This increased practical knowledge should add impetus to many potential signal processing tasks (target trackers, beamformers, communication, receivers, spread spectrum). Three diverse, yet tightly

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE 10

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

coupled, research topics were posed, centering on the usage of Signed-Digit (SD) arithmetic to solve mult/acc intensive signal processing tasks (streaming data). Efficient implementations for signed-digit arithmetic were sought for systolic arrays. Connectivity and control were investigated for innerent fault-tolerance. Lastly, multiple-valued logic for the Signed Binary Number Representations (SBNR) was studied for both fault-tolerance and array regularity. The dominant and focused application of this research was efficient solutions of specific signal processing algorithms.

aroftr.doc

Table of Contents

TABLES

FIGURES

SBNR DIGITAL SIGNAL PROCESSOR ARCHITECTURE

1.0 Scope of Work

"There is a great deal of innovation for new complex special-purpose signal processing integrated circuits...often yielding well over a factor of a thousand improvement over even the fastest general-purpose machines," Jonathan Allen, Fellow IEEE, in "Computer Architecture for Digital Signal Processing," Proc. IEEE, Volume 73, Number 5, pp. 852-973, May 1985.

The following research was proposed in a three-year period. This constitutes significantly distinct efforts which complement the existing efforts in current adaptive signal processor architecture research. Briefly, these tasks comprised a study of non-conventional number system implementations focusing on VLSI enhancements attributable to redundant number systems. This increased practical knowledge should add impetus to many potential signal processing tasks (target trackers, beamformers, communication, receivers, spread spectrum). Three diverse, yet tightly coupled, research topics were posed, centering on the usage of Signed-Digit (SD) arithmetic to solve mult/acc intensive signal processing tasks (streaming data). Efficient implementations for signed-digit arithmetic were sought for systolic arrays. Connectivity and control were investigated for inherent fault-tolerance. Lastly, multiple-valued logic for the Signed Binary Number Representations (SBNR) was studied for both fault-tolerance and array regularity. The dominant and focused application of this research was efficient solutions of specific signal processing algorithms.

2.0 Conventional Number Systems Drawbacks

The most serious objection to using the conventional number representations (the sign magnitude, the radix complement and the diminished radix complement representations) for a signal processor cell is that addition in these representations cannot be truly parallel. A computing cell designed for such representations cannot be easily connected to run in parallel with identical cells in such a way that the microsteps involving additions can be carried out in time independent of the number of cells. For signed-digit representations, the number of cells and the precision of the operands will not affect the time of such microsteps. The time needed will only depend on the structure of an adder position.

Another convenience in designing arithmetic modules with signed digits is that no special treatment is required for the most-significant position. For radix complement or diminished radix complement notation, special attention is needed to handle the arithmetic shifts, the sign of multipliers and/or the end-around carries. For the sign magnitude notation, the sign of the result of an addition or subtraction requires dedicated circuitry. All of these little problems do not exist for the signed-digit notation. The shift input for any arithmetic shifts is always zero. The indicator digit (the sign digit) can be treated just like all other digits.

The serial mode of processing has to proceed from the least-significant end to the most-significant end if the conventional number representations are used. The overflow condition or the leading zeros can be detected only after

the last segment of the result has been generated. For the signed-digit notations, serial operations can proceed from the most-significant end. Processing may be stopped by an end symbol in the operands such as the space zero discussed by Avizienis [1]. This can lead to a more efficient serial processing procedure if the allowed precision is an excess of the needed precision. Since the most significant digits of the result can be generated first, the overflow condition or the leading zeros can be detected at the beginning of an operation. Result digits may be stored away in their final positions without subsequent corrective shifts which is not necessarily trivial in a multi-precision environment.

For the signed-digit notations, the basic arithmetic algorithms for each digit position are essentially invariant with the position of the operand digits. Each result digit is dependent only on the operand digits in a fixed number of digit positions. Because of this the detection and the correction of hardware errors can be independently implemented for each digit position as suggested by Avizienis [1]. The "round-off" error resulting from simple truncation is without bias. For a mantissa of m fractional digits, the maximum absolute truncation error in the mantissa is less than one mantissa bit!

Besides the conventional number representations, there are a few other novel number representations which have advantages in special situations but are not suitable for this variable precision module. Examples are the residue number representation and the negative base representation. The residue number system developed from linear congruences does not require carry propagation. The multiplication of two numbers needs as little time as the addition. The main difficulties of the residue number representation pertain to the determination of the relative magnitudes of the two numbers and to the division process. The negative base number system, on the other hand, is not easily implemented in negative bases. The sign of a number in a negative base depends on whether the most significant digit is an even or odd position. This complicates the division process since the signs of the operands and the signs of the intermediate results are essential in any division algorithm.

In short, the signed-digit systems provide two dimensions of freedom: the number of processor modules and the precision of operands. These allow a variable length operation to be practicable in a processor with a variable number of digit positions. The signed-digit systems are natural choices for the present module which is required to process operands with a variable precision either by itself or in parallel with a variable number of identical modules.

2.1 Task Summaries

1. We studied the impact of signed-digit number systems for signal processor implementations. Specifically, we proposed to implement new ALU structures within the context of recursive algorithms (LMS, LS, SVD, Givens Rotations, ...) focusing on fault-tolerant architectures.

2. We analyzed at least four architectures: fully-parallel multiple adder/mult. structures, distributed arithmetic structures, multiple operand adder structures, and ROM/adder structures making maximal use of pipelining and parallel mechanisms.

3. We studied engineering trade-offs among conventional, 2's complement arithmetic and signed-digit arithmetic to reduce pin count and develop more functionally robust devices for signal processors. Multi-valued logic circuits were considered.

2.2 Synopsis of Proposed Method

Classical techniques were applied to these tasks. Namely, VLSI floorplans were produced and area/time figure of merits generated. Analytical comparisons were then established using popular benchmarks as the LMS algorithm and least-squares algorithms applied to signal processing of radar, sonar, and communications tasks.

2.3 Objectives Summary

We sought to demonstrate the effectiveness of each implementation towards design goals such as speed, power, weight, and size. Additionally, we intended to demonstrate the efficiencies of signed-digit implementations which supposedly have minimal interconnects between adjacent digit positions. We demonstrated the superior modular features of signed-digit for ALU's in adaptive signal processors.

3.0 Identification and Significance of Opportunity

This focused architecture study exploited promising memory-oriented structures common to distributed arithmetic organizations because the costly multiply/accumulate cycle (typical in signal processing) reduces to fast shift/add cycles. Secondly, signed binary number representations (SBNR), a subset of redundant number codes, were realized with higher information per wire ratios, thus reducing intercell connections (a relatively high VLSI cost in current conventional number systems). Thirdly, multi-valued-logic (although slow) maps SBNR representation one-to-one. Hence, its effectiveness was studied.

As a result, digital signal processing applications such as FFT's, convolutions, Hartley transforms, beamforming, coding, communications receivers, target trackers, and antenna arrays stand to achieve lower power requirements and higher microminiaturization levels. There is a great need for ultra-fast FFT's in spread spectrum. Because no architecture research operates in a vacuum, we collaborated with NCR, TRW, and RCA foundries to eventually test/develop actual devices. NCR is particularly interested in this study because its local R & D facility designs systolic array devices (notably the NCR 45CG72, the GAPP 6x12 PE chip, for which Space Tech has been writing signal processing algorithms). This is the one of the few available true systolic array chips, and an excellent testbed for our studies.

[A difficulty in terminology now arises. In this research, we studied redundant number systems (sometimes called redundant coding, SDNR, SBNR, and mistakenly called negabinary and/or mirror numbers). We also investigated fault-tolerant properties of this number system partially with redundant circuits (here, "redundancy" refers to more than one circuit). We hope the reader can determine the meaning from its context.]

3.1 Multi-Valued-Logic and Systolic Arrays

In recent papers by Hurst [2] and this Principal Investigator (see Appendix), it is noted that multi-valued logic (MVL) may show great promise in the future for VLSI. At present, binary systems are facing interconnect problems which appear to be insurmountable. Silicon areas devoted to intrachip connections now consume twice the area of active logic elements on the chip [3]. Array implementations whether data-flow, systolic, or otherwise cause a severe escalation of interconnect area thus rendering lower silicon area efficiencies. Likewise, off chip connections are generating new and complex problems for the board designer. These packaging solutions are not without concomitant thermal and mechanical constraints. Such factors cause us to reflect upon denser information content to interconnection ratios. A solution using higher radix arithmetic is proposed and coupled with MVL promises to relieve some of the silicon area inefficiencies when conventional binary arithmetic is used. Even for the regular architectures of systolic arrays, Moraga [4] has shown the effectiveness of such MVL implementations.

3.2 Computational Model

Our VLSI model of computation to derive complexity measures was based on the following generally accepted assumptions [5-7]:

a. Wires have minimal width W=A(const); hence W^2 is the unit of measure for the area.
b. The area required to store one bit of information is $A(W)^2$); the distance between parallel wires is A(W).
c. Double layer metalization is allowed.
d. Wires run only horizontally and vertically.
e. Each transistor needs a minimal transition time, Y=A(k) (k is a constant), to change its state. Thus Y is the unit execution time.
f. A binary signal propagates along a wire in time A(W). Any long wires of length, L, require respective buffer/drivers with area A=A(W) x O(L).

3.3 Signed-Digit Number Representation

In the most general sense, a redundant number system allows both an increase in the number of positive digits and negative digits as follows.

$$w_{red}^{(n,m)} : R \times R \times \ldots \times R \longrightarrow Q \tag{1}$$

$$a_{n-1} \cdots a_0\ a_{-1}\ a_{-2} \cdots a_{-m} \longrightarrow \sum_{i=-m}^{n-1} a_i d_i \tag{2}$$

$$\text{where the digits } d_i \in R := \{-r_1, -r_1+1, \ldots, 0, 1, \ldots, r_2-1, r_2\} \tag{3}$$

$$(r_1, r_2 > 0)$$

The representation described by Eqs. (1-3) is called **redundant notation with base d.** The above mapping of the number representation (or notation), w_{red} > assigns to a sequence $a_{n-1} \ldots a_{-m}$ of digits a value from a range Q where Q may be an integer, the reals, or zero.

The general redundant representation does not lead to efficiencies in algorithm computations or implementations. If subsequent restrictions (Sec 3.4) are placed upon the general redundant notation, very attractive properties support efficient implementations. However, the basic properties of Signed-Digit Number Representations (SDNR) are:

a. The radix, d, is a positive integer.
b. The SDNR of the algebraic quantity, zero, is unique if
$$m=n,\ (d-1) \geq m \quad \text{and} \quad m-1 \geq d-1)/2 \qquad (4)$$
c. Transformations between conventional representations and SDNR exist.
d. Totally parallel addition/subtraction are possible.
e. Addition and subtraction of two numbers are free of serial propagations of carry/borrows.
f. SDNR numbers are positionally weighted.
g. The polarity of an SDNR number is given by the polarity of its most significant non-zero digit.
h. No special treatment is needed for the most-significant position.
i. Addition/subtraction time is independent of operand length.

Avizienis [1], Atkins [8], Tung [9], Ercegovac [10], and Robertson [11] have shown that SDNR can effectively operate in a general purpose digital computer for the following reasons.

1. Redundancy introduced into the adder-subtracter structure reduces (but does not entirely eliminate) carry-borrow propagation leading to rapid multiplication.
2. Full precision comparison of the divisor and partial remainder in division algorithms is not required because quotient digits can be determined from relatively few high order bits.
3. Negation is a simple logical complementing of the sign bits (e.g., unlike two's complement notation which requires an additional step, adding an LSB "one"). As was seen in the ILLIAC III [8], such negation expedites execution of floating-point addition and subtraction.
4. Variable lengtn operand formats and parallel vector arithmetic are facilitated by basic properties of SDNR's. First and foremost, operations can proceed from left-to-right (rather than right-to-left as required in 1's, 2's complement representations). Secondly, if appropriately implemented, the position of the least-significant digit need not be known for adders and subtracters.
5. Because a signed-digit combination adder/subtracter needs no carry/borrow in the LSD, the ALU can be partitioned into identical and cascadable single digit adder/subtracters. VLSI implementations tend to become highly regular.
6. Multiplication with SDNR tends to automatically produce rounded results (of great importance in computationally intensive signal processing applications). In fact, Robertson, based on work by Rohatsch [12], has shown that the probability of obtaining a rounded result is 5/6.
7. SDNR allows unusual algorithms such as wired-in significant-digit arithmetic [13] and dual notation algorithms capable of accepting both SDNR and conventional operands (1's, 2's complement) to produce SDNR results [14].

These observations lead to tne implementation of a universal Aritnmetic Building Element (ABE) capturing not only the preceding algorithms but also efficiently separating functions of logic designs and arithmetic design [15].
8. Overflow detection can occur immediately following the production of most-significant result digits (unlike conventional notation).

3.4 Efficient SDNR Realization

Several implementations based on the SDNR have already been investigated [11,16,17,18]. All of these, however, sought to satisfy general data processing requirements of a mainframe computer. In contrast, signal processing applications are generally multiplication/addition intensive. (Of late, the utility of distributed arithmetic [19,20,21] has shed new light on bit-wise algorithms, also essentially partial product and accumulation intensive.)

An allowed digit set (-1,0,1) which is a subset of tne SDNR is assumed. A redundant Signed Binary Number Representation (SBNR):

$$X_n X_{n-1} \cdots X_1 \text{ --> } X_i \text{ in } (-1,0,1) \tag{5}$$

represents a number whose value is expressed as

$$\sum_{i=1}^{n} X_i \cdot 2^{i-1} \tag{6}$$

The importance of SBNR is as follows:

a. Conversion of unsigned binary numbers to SBNR is unnecessary as they are identical.
b. Since a two's complement binary representation $(X_n X_{n-1} \cdots X_1)_2$ expresses the number

$$-X_n 2^{n-1} + \sum_{i=1}^{n-1} X_i \cdot 2^{i-1} \tag{7}$$

tnis same number can be expressed in SBNR by

$$(X_n X_{n-1} \cdots X_1)_{SD} \tag{8}$$

because the sign bit X_n in 2's complement representation is considered to have weight -2^{n-1}. Hence, conversion from signed binary 2's complement representation to SBNR is simply an inversion of the sign bit alone!

Avizienis [1] further demonstrated tnat the SBNR (radix d=2 with digit values -1,0,1) with a decreased redundancy requirement (invoking a two-step addition by allowing the propagation of the transfer digit over two digital positions to the left) requires only d+1 sum digits. In general, he showed tnat tne lower limit of required redundancy of one digit depends on the number of digital positions tne transfer digits propagate as follows.

If GIVEN:

a. no redundancy utilized
b. s_i, sum digit {d values only}
THEN:

$$s_i = f(z_i, y_i, z_{i+1} \ldots, z_m, y_m) \quad (9)$$

z_i = ith addend digit
y_i = ith augend digit

If, however, s_i in {d+1 values}, then Eq. (9) becomes

$$s_i = f(z_i, y_i, z_{i+1}, y_{i+1}, z_{i+2}, y_{i+2}) \quad (10)$$

and if s_i in {d+2 values or more}, then Eq. (9) becomes

$$s_i = f(z_i, y_i, z_{i+1}, y_{i+1}) \quad (11)$$

Using these observations, a single cell can implement the one digit adder/subtracter if certain choices for a redundant digit are always made. Specifically, let any redundant binary digit be represented by two bits X_s and X_d as follows where $\bar{1} = -1$.

Table 1. Redundant Digit Selection Rule

Redundant Digit X	Representation Sign X_s	Digit X_d
0	0	0
1	0	1
$\bar{1}$	1	1

Invoking this TRIT realization for our SBNR further simplifies the cell implementation without sacrificing the transfer digit propagation advantage. Using this subset allows six types of intermediate results in the first of two addition steps as defined in Table 2.

Table 2. Intermediate Addition Step Classes

Type	Augend (x_i)	Addend (y_i)	Next Lower Position (x_{i-1}, y_{i-1})	Carry (c_i)	Intermed. Sum (s_i)
1	1	1	---	1	0
2	1	0	Both are positive	1	$\bar{1}$
	0	1	At least one is negative	0	1
3	0	0	---	0	0
4	1	$\bar{1}$	---	0	0
	$\bar{1}$	1	---	0	0
5	0	$\bar{1}$	Both are positive	0	$\bar{1}$
	$\bar{1}$	0	At least one is negative	$\bar{1}$	1
6	$\bar{1}$	$\bar{1}$	---	$\bar{1}$	0

The second step in an addition cycle adds s_i and c_{i-1} from the next lower position to obtain a sum digit z_i with no carry/borrow generation required.

If we allow any redundant binary digit to be represented as $X_s X_d$ with the redundant digit selection rule as prescribed in Table 1, the Boolean equations which govern selection and addition per Table 2 produce two critical observations. The ith SBNR carries, C_s and C_d, depend only upon the ith, i-1 digits and i-1 carries. **Hence, carry propagation extends only into the next adjacent digit column. SBNR addition does not require full-word carry propagation as in binary addition. SBNR addition makes systolic array implementations straightforward. Pre-scrambling bits or words is not required.**

A primitive cell suitable for large VLSI arrays and especially for adaptive signal processors must have few interconnections beyond its nearest neighbors and must have very simple controls. VLSI arrays effectively function in a data-flow manner. Fortunately, many signal processing algorithms can be implemented with distributed or bit-serial arithmetic. Mactaggart and Jack [22], and others have shown that bit-serial implementations offer a highly regular design and lower power consumption than conventional arithmetic. One such cell [16] is depicted in Figure 1. This cell implements the basic addition/subtraction steps of Table 2 using the SBNR of (-1,0,1) and the redundant digit selection rule of Table 1.

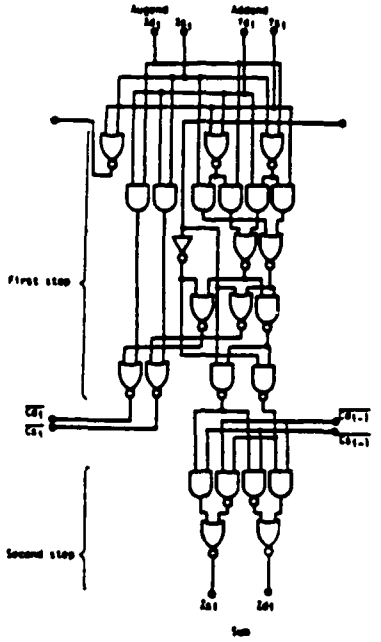


Figure 1. Primitive Bit-Serial Cell [16]

3.5 Matrix X Matrix Multiplication

Most of the computational effort expended by a digital signal processor is devoted to matrix x matrix multiplication. Such matrix operations may be either sums of word level products or sums of bit level products. We now know that a strong relationship exists between word and bit level systolic arrays [23]. If we treat such computational problems from the outset as bit level manipulations, fast area efficient VLSI arrays are possible [24,25]. In our implementations, a systolic-like bit level approach is assumed where each processing cell is a multiplier and gated full adder. However, the multiplier and adder utilize SBNR rather than 2's complement arithmetic for reasons discussed earlier.

Another advantage to SBNR is the absence of special circuitry and algorithms to handle signal operands. In 2's complement arithmetic, the Baugh Wooley algorithm can be used (with an attendant high latency cost). In this procedure, 2's complement words are treated as positive numbers if:

1. A fixed correction term is added to the result for each word level multiplication.
2. All partial products normally with a negative weighting are complemented.

Two's complement implementations on a systolic array require a negative weighting flag or a tag on the partial products which must propagate vertically down through the array. Hence, another latch and control line is required for each columnar path. Furthermore, final addition of correction

terms requires an initialization of the accumulators in the adder trees to a value, which is generally

$$(2^m - 2^{2m-1}) \times n \qquad (12)$$

In general, the number of systolic array cells required to multiply two nxm matrices with elements m bits long in a fully parallel fashion is a total of

$$n \times (2m + \log_2 n) \qquad (13)$$

cells where word growth is taken into account. However, McCanny and McWhirter [26] have identified a procedure to halve the number of cells by removing intermediate zero bits. The procedure is to permit only one set of words within a given row to move at any time slot, keeping the other set of words within the row at fixed sites. Then, in the next time slot, move the fixed set of words and keep the previously moved set fixed. This alternation of left/right moves can be maintained by latching bits using half the system clock speeds. Successive rows in the array must move in anti-phase relative to each other.

3.6 Device Redundancy and Fault-Tolerance

Any practical architectures designed today should be highly fault-tolerant. Circuit redundancy and built-in-self-test are theoretically achievable. Redundancy (of elements, not aritnmetic codes) does offer one additional advantage to the chip builder. The system designer can run models long before production of the new system starts. But, reliably high-yield logic chips for these machines are often difficult to achieve because the system designer always wants the very latest in technology. Redundancy in the basic logic design can enhance the yield by a significant amount and greatly reduce the wafer start requirements. When the yield increases and production starts, this same redundancy is now available to improve reliability.

The model in Section 5.7 demonstrates the dependence of yield on the nature of the defects and, together with gross yield estimates and the appropriate nonredundant yield factor, it will serve as a reasonable starting point to model actual yield data. The existence of complex local correlations and some non-point-like defects will clearly complicate matters, although, in many cases, a perturbative approach will be adequate to model the situation. Understanding yield issues is important to architecture design.

3.7 Redundancy, Fault-Tolerance and Testing

Achieving high reliability in a complex device or system is a difficult but critical task. The investigations for this project have included a careful consideration of reliability and testability considerations. It is now challenging for manufacturers to maintain a compound growth rate in per-circuit reliability of 60%. Past methods are no longer valid. Tne verification of machine reliability due to electronic components poses a significant challenge to tne future. For example, consider two realistic examples. Assume a computer with 1,000 circuits/chip. Suppose that a manufacturer builds 1,000 macnines to acnieve 50K user power-on hours per machine at the usual 1,000 hour MTTF for tne electronics. This corresponds to

a 50 PPM cumulative fraction fail for the chips. To verify this failure rate at a 90 percent confidence level, a manufacturer would have to test 80,000 chips to allow for one failure during testing. For an overall production, this represents eight percent of a production run. Historical trends indicate that the overall reliability rate improvement is approximately 2X each year. By 1994, if we consider the constraints of the first example, the reliability must be theoretically improved by 1,000 times. Verifying reliability to the same confidence level would require 80M chips, or 80 times the number of the actual production run. Clearly, reliability, testing, and redundancy are intimately coupled.

The following aspects of reliability were considered:

1. A major problem with all complex, newly designed devices which use a sufficiently large chip area is poor yield. The yield can be significantly improved by using redundancy. During the final manufacturing stages, devices with only a few "bad" cells can be reconfigured to leave out bad cells.

2. A similar approach for fault-tolerance can be used for hard failures in the field. In this case, reconfiguration has to be dynamic. This means that after a cell had been detected to be faulty, the array configuration has to be altered under program control.

3. To study the effectiveness of fault-tolerance and for optimizing such designs, estimation of hard and soft failure rates is required. Because the handling strategies can be different, hard and soft failures often have to be considered separately. Preliminary estimates are based on empirical techniques. Such estimates are not very accurate, but are still indispensable when evaluating different design options.

Consideration of soft failures is especially important for Multi-Valued Logic (MVL) devices. Because the voltage range is divided into more than two regions, it will take much less energy (from electromagnetic noise or alpha-particles, etc.) to cause an extraneous transition.

4. Testing, both by the manufacturer and in the field, is an integral part of reliability strategy. It is now recognized that testing must be considered during the design phase itself. Two aspects of testing will be considered. Design-For-Testability (DFT) is to be used for easier and faster test-pattern generation and applications. The other is Built-In-Self-Test (BIST), which allows a system to exercise itself and verify correctly operating hardware.

4.0 Technical Objectives

Succinctly, the technical objectives of this effort were:

a. Determine intrinsic properties of SBNR embedded as PE's in a systolic array via distributed arithmetic cells. Capitalize on the inherent modular properties of residue numbers to be implemented in SBNR engines.

b. Establish highly modularized architectures using SBNR arithmetic engines to increase information per wire ratios.

c. Determine engineering trade-offs of power, weight, and size for SBNR array architectures to help a system designer and silicon floorplanner lay out competitive VLSI devices.

4.1 Higher Radix Implementations

We considered at least one implementation of higher radix arithmetic, namely ternary, which when viewed as a redundant or signed-digit number system held promise for signal processing applications in which division-sparse operations occur. We studied signed-digit number representations and basic properties attractive to signal processing applications which manipulate sequential data streams.

4.2 Systolic Array PE

We identified a realization of TRITS (ternary digits) which serves as the primitive VLSI cell. The regular nature of this cell enhances systolic array architectures. Multiple-valued encoding affords us the opportunity to reduce ripple-through carries. Ternary arithmetic may have a balanced as well as an unbalanced coding. Balanced encoding requires less gates when compared to binary and unbalanced encodings. Unfortunately, logic delay increases [27]. However, in the TRIT realizations utilized herein, a balanced encoding coupled with redundancy in the encoding improves both logic delay and gate count.

This Principal Investigator has considerable design experience with systolic array PE's. He has designed control units for the first systolic array (NCR 45CG72) and generated several signal processing algorithms for it in conjunction with NCR (including LMS, LS and SVD for adaptive beamformer applications). From the experiences, a basis for new and faster circuits can be identified. One such candidate, SBNR PE suitable for a systolic array, is shown in Figure 2. This is a derivative of the NCR cell with several critical differences. First, additional latches and data paths exist. Second, RAM is much larger at each cell. Third, internal cell pipelining is used to speed effective instruction execution (not easily shown in a block diagram). Fourth, the cell implements signed-digit arithmetic with fewer intracell connections. Lastly, this single cell can do multiply, add, and subtract in fewer steps. A systolic array module (SAM) of this PE is depicted in the floorplan of Figure 3.

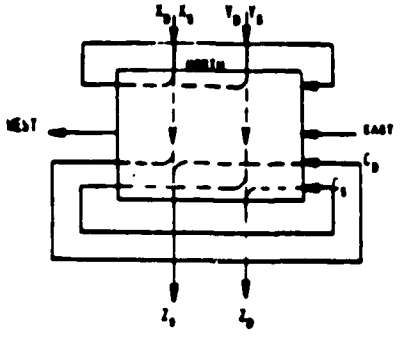


Figure 2. An SBNR Data Flow Cell

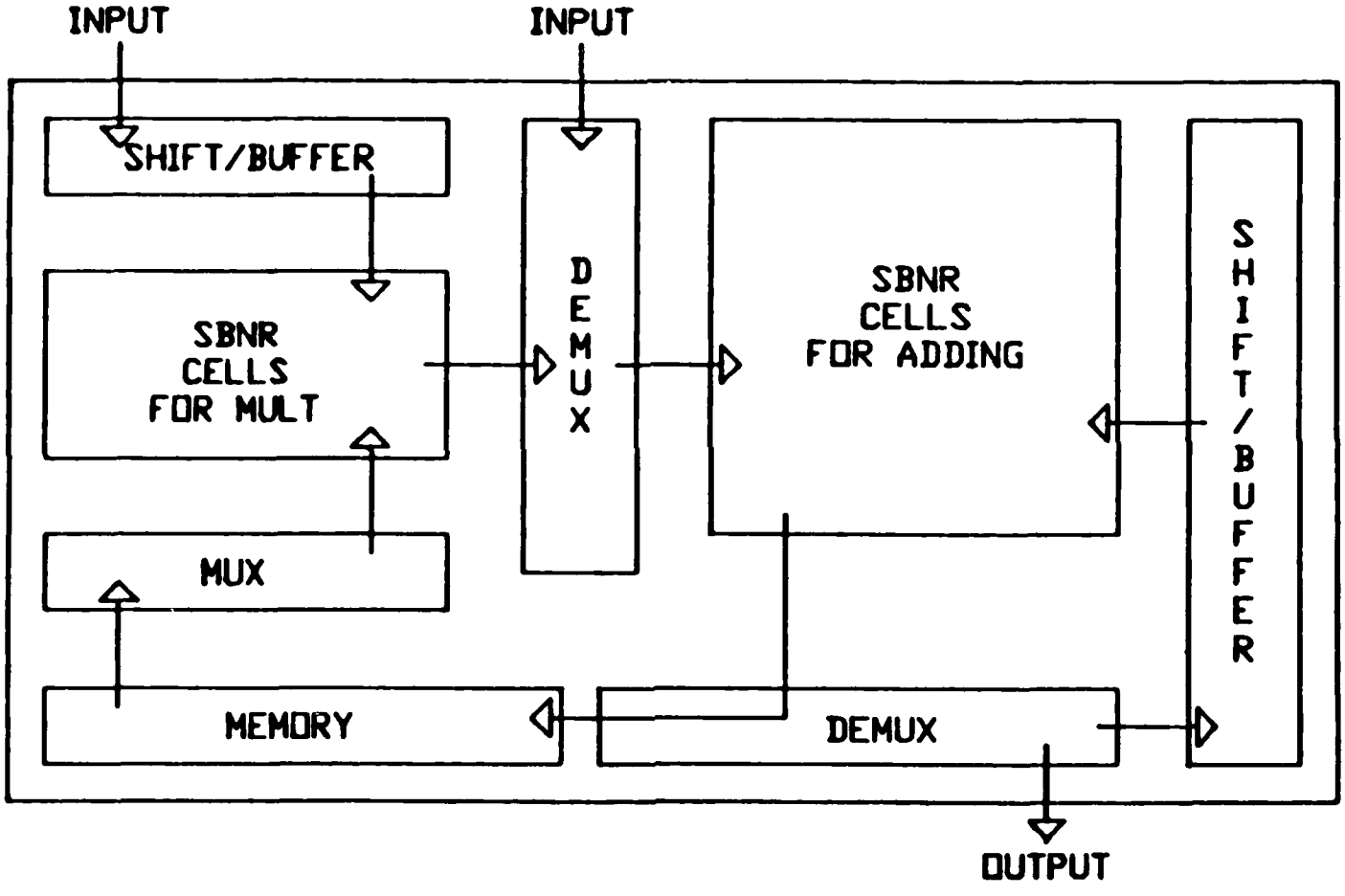


Figure 7 SAM Floorplan.

4.3 Signal Processing Algorithm

There is an intimate coupling between word and bit level matrix x matrix multiplication. A systolic implementation of common algorithms invoking a digit subset of redundant number representations, Signed Binary Number Representations (SBNR) is easily realized with TRIT MVL. A significant property of redundant number systems supports the production of left-to-right (most-significant-digit to least-significant-digit) algorithms. Sips [28] has demonstrated the utility of left-to-right algorithms for a general purpose computer. We found this RTL property extremely beneficial for the ADC and DAC interface.

We analyzed appropriate ADC and DAC SBNR realizations. It is important to note that the realizations directly carry over from a property of redundant numbers. This is vital for real-time signal processing (which is predominantly analog sourced).

4.4 Fault-Tolerant Properties

It is important to identify PE's that are highly regular, have minimal I/O pinout requirements, have minimal gate count, inter- and intra-circuit connectivities and low power requirements that support a high degree of fault-tolerance. VLSI technologists are fast developing wafer scale-integration. A major problem with such assemblies is that some cells are likely to be defective. Hence, a major objective was to determine optimal reconfigurable networks "around" such faults for our SBNR PE systolic arrays. The procedure was to minimize the length of the longest wire in the system, thus minimizing the communication time between cells. Channel width was also a major consideration. The procedure assumed a probabilistic model of cell failure since Leighton and Leiserson [29] have demonstrated many positive results. In many ways this problem is similar to the graph-theoretic models used in the bottleneck traveling salesman problem. Leiserson has already derived bounds on wire length and channel width for two-dimensional arrays. We compared our results with these bounds. Leiserson nicely provides us with results [29] that show there is a simple, linear-time algorithm to connect most of the live cells on an N-cell wafer into a linear array using wires of unit length 1,2, or 3 channels of unit width 2.

5.0 Research Methodology

5.1 Optimized Fault-Tolerant Designs

A four-step procedure is used from the top-level down. At the first and highest step, use of an SBNR allows parallel and modularized operation of MVL arithmetic processors for fast execution of full precision, fixed-point arithmetic.

Second, a memory-intensive arithmetic algorithm is employed which capitalizes on the short internal word lengths of SBNR processors. ROM-based structures have been shown by Peled and Liu [30] to be extremely effective for FIR filters. This PI has made the same discovery for adaptive filters using combinations of ROM's and RAM's. Third, memory accesses within processors can be pipelined. Fourth, transistor-level simulation tools can be employed to design the high-speed memory circuits. The capability of identifying failed

processor bits and maintaining correct DSP output in the presence of errors arises from our use of an extensible algorithm for incorporating redundant processor chips.

5.1.1 Reliability Models

Reliability models to evaluate computer systems must estimate multiple system parameters (e.g., failure rate). The quality of the prediction model rests on the estimates of its input parameters. In practice, extensive testing and burn-in procedures produce a best point estimate with measures of dispersion. In a crude way, we employ only a point estimate; e.g., the mean or an upper or lower bound [31]. Depending on the dispersion in its input parameters, a reliability estimate may or may not be acceptable. For many types of computer components, where reliability is high and failures are low, the uncertainty involved in determining a parameter in question such as the failure rate may be large. A ultrareliability system necessitates investigating the ensuing uncertainty in system reliability. Unfortunately, this problem has not been well studied. Few available reliability evaluation programs offer such sensitivity estimates.

Our model was as follows (see [32]). Assume that system lifetimes are exponentially distributed. To consider the dispersion in parameter estimation, stipulate that the failure rate L is a random variable. It is doubly stochastic [33]. The system reliability at a given time, t, also referred to here as **time point reliability**, is then a random variable $R_L(t)$ [with a particular value $r_L(t)$], with the distribution in L. The variance of $R_L(t)$ is crude but an effective dispersion measure to the random nature of L. We now can exploit the model with variations in failure rate for useful properties of exponential distributions. Use two approaches, an exact model based on the complete distribution of L and an approximation of employs only the first and second parameter distribution moments.

Iyer [32] has shown feasible exact and approximate models. The exact model is based on a gamma distribution and is easily extended to fault-tolerant redundancy configurations, such as TMR, by substituting the appropriate value for system reliability. Iyer develops first and second moments for time point reliabilities.

5.2 Hypergraph Models for Fault-Tolerant Systolic Arrays

We proposed and used a graph theoretic procedure similar to [34] to measure the VLSI effectiveness of our design strategy by the area required to lay out the fault-tolerant processor arrays. We repeat the completeness here in the procedure in [34]. Three design strategies are described briefly.

1. Embed the desired array in a simple graph to model techniques that build a fault-tolerant array. Each PE must contain a robust switching mechanism to configure the good PE's into an array of the desired structure using nearest-neighbor connections.
2. Embed the desired array in a graph with multipoint edges to build a robust array by running buses adjacent to the PE's and interconnecting the fault-free PE's into the bank of buses (say, via laser-welding). Use each array link via a distinct bus.

3. Embed the desired array into a "switched" graph, whose vertices are partitioned into PE vertices and switch vertices. Try to realize inter-PE connections through a switching network external to the PE's thereby allowing one to bypass faulty PE's. In comparison studies by [35,36], no single design strategy as yet appears to be uniformly superior to any other.

Because of the multi-faceted nature, a firm understanding of all three strategies is vital. Methods 1 and 3 have been deeply studied [36-43]. Method 2 was proposed [37] and tangentially studied in [36].

5.2.1 The Design Strategy

As in [34], we followed the same procedures. Assume a target array structure. Construct a fault-tolerant array to simulate this structure. PE's are represented by squares, and both wires and buses are represented by lines as in Figure 4 [34]. Now construct some number of identical PE's that are precisely the PE's that one would design for the ideal array, with the same I/O interfaces. Next, lay the PE's out in a (logical, if not physical) row, with lines coming out of their I/O ports running perpendicular to the row of PE's. Then run some number of buses above the row of PE's. We are told (via some unspecified mechanism) which of the PE's are faulty and which are fault-free. Now use laser welding to connect I/O lines to buses in a way that configures the fault-free PE's into an array of the desired structure.

Use the following area definition of [34]

area(array) = (PE-number) X (PE-width) X (Bus-depth)

Let Bus-depth be the maximum number of buses passing over any point in the layout. (Ignore the contribution of the separate PE's.)

A solution array has two components: specification of the **structure** of the array and of the **configuration procedure.** The procedure is an assignment operation mapping ideal-array PE's onto actual PE's, as well as a mapping of ideal-array edges/communication links to the buses that will simulate them.

5.3 Comparison of Error Detecting Codes

Several techniques to obtain fault tolerance through error detection have been studied. Most of these schemes can be categorized as being hardware redundant or time redundant. The hardware redundant systems (for example, Triple Modular Redundancy [44] and quadded logic [45]) typically require arithmetic to be computed in more than one processor. A checker compares the results to detect errors. These schemes require a factor of at least 2 or 3 in hardware redundancy.

The time redundant scheme requires that each result be calculated twice, with the two answers compared to find errors. Two examples of this approach are alternating logic [46] and recomputing with shifted operands [47]. In the alternating logic technique, the result is recomputed from inverted operands and should be the inverse of the original result. Recomputing with shifted operands verifies that, when the operands are shifted, the result contains a shifted version of the original bit pattern. Both of these systems are effective primarily for stuck-at faults.

A more general approach than hardware or time redundancy is that of algorithm-based fault-tolerance (information redundancy) [48]. The central idea in this technique is that the data are encoded at the system level in the form of some error-correcting or error-detecting code, and the algorithms are designed to operate on encoded input data and produce encoded output data. The result is real-time error detection without a duplication of arithmetic processors or a doubled processing cycle time. The primary entrants in this category are the low-cost residue and inverse residue codes [49,50,51], the checksum code [48,52,53], and the weighted checksum code [54].

5.3.1 Residue Codes

Residue encoding is based on finding the remainder of a sum of operand digits evaluated modulo N, where N is a predetermined base. The binary operand is broken into sections of a bits; each section is considered a digit. The base of the operand is found from the digit size: $N = 2^a-1$. For a k-bit number, all k/a digits are added together, and the sum is evaluated mod N. The remainder of this calculation is the residue code for the operand. The simple residue code will detect a fault in one bit, even after a repetitive calculation like multiplication. It will also detect an error if up to a consecutive bits are faulty [49].

Avizienis devised a scheme in which two or more residue digits are used to detect and then locate an error [49]. Furthermore, the digits also check each other--if only one residue digit indicates an error, then that residue is incorrect--only if both show an error will a fault in the number be corrected.

5.3.1.1 Signed-Digit Residue Code

The residue code has been extended to include numbers expressed in signed-digit number representation [50]. When a single digit of an SDNR number is faulty, any number of (not necessarily consecutive) bits within that digit may be in error, and the fault will still be detected. The only exceptions to this rule are errors which add or subtract the base N from the digit. For example, if a=4, and the signed digit is changed from (9) to (-6), the error will go undetected. However, only a very specific change in the bit pattern will camouflage the fault, so detection is highly probable. Furthermore, if the number is encoded in signed binary number representation, more bits must be changed, and they must be changed to specific values of (1, 0,-1) to hide the error. **The combination of SBNR and residue encoding thus appears to have great fault-tolerance potential.**

5.3.2 Checksum Codes

Unlike the research into residue codes, Abraham's studies of checksums have been directed specifically at matrix encoding [48]. The checksums approach attaches one or more checksums to the end of a row or column of a matrix. These numbers then participate in all calculations as if they were just data. The net effect on a systolic processor is simply an increase in the size of each dimension of one or two rows. No special algorithms are needed to take care of the error codes. Unfortunately, checksum coding was introduced in the context of floating point computers. Fixed-point calculations (like those prevalent in high-speed dedicated signal processors)

require a slightly more difficult coding scheme, because a full-precision checksum would overflow a fixed-point system. In situations where the implementation depends on the number system, this report assumes fixed-point operations.

5.3.2.1 Unweighted Checksums

In the simplest checksum code, an unweighted checksum is formed by adding together all elements in a row or column of a matrix. Overflow bits from this addition are ignored. Depending on the applications, just row or column encoding may be sufficient, or both may be needed. The unweighted checksum will detect a single error in the row or column. It is effective in LU decomposition, matrix inversion, matrix-vector multiplication, matrix-scalar multiplication and singular value decomposition [48] [53].

5.3.2.2 Weighted Checksums

To achieve error correcting capability, the checksum must positionally weight the addends. The result is the weighted checksum [54]. In this system each element of a vector within the matrix is multiplied by a different weighting factor before being added to form the checksum. The simplest weighting scheme consists of powers of 2--the elements e(i) would be multiplied by weight 2^i (left-shifted i bit positions), for example. For a fixed-point system with numbers of length k bits, the sum would quickly overflow, so it is added modulo a specific base. Unlike the residue code, the base for weighted checksums is the largest prime number less than 2^{k+1}. For 16-bit systems, this number is 131 059, and for 32-bit, it is 8 589 934 583.

To allow correction of errors, the weighted checksum vector must be augmented with a vector of unweighted checksums. Thus, as with residue encoding, if one checksum detects an error, the checksum is incorrect; if they both do, the error in the data may be located and corrected. The weighted checksums technique can correct errors in matrix multiplication with a matrix, vector or scalar, matrix inversion, and LU decomposition (by Gaussian elimination).

5.3.3 Comparison of Fault-Tolerant Implementations

To compute any of the error-detecting codes described requires adder trees to sum the digits or elements. In residue encoding, an end-around-carry is generated within the adders. In checksum encoding, the overflow carries are thrown away. In weighted checksum encoding, each level of addition is performed modulo the prime base. Thus, for residue and unweighted checksum, the areas are almost the same for a length-n column of additions--$O(n)$--and the add time is identical--$O(\log_2 n)$. For weighted checksum, each adder must compare its sum to the base, and subtract the base if necessary. The adder can also be used for this subtraction, so the area remains $O(n)$, but the double add cycle means that the relative time is $O(2\log_2 n)$.

In an nxn matrix with row and column encoding, the area-time complexity for the first two cases is $O(n^2 \log_2 n)$. For the weighted checksums, it is $O(2n^2 \log_2 n)$. Keep in mind that for error correcting capability, the A-T product of the residue is doubled, while those of the weighted and unweighted checksums are added together.

Another consideration is the ease with which algorithms may be adapted to allow fault-tolerance. In the case of residue encoding, the residue digits must be handled separately, which increases algorithm complexity. The checksums system, however, merely increases slightly the size of the input to the algorithm, with no special treatment given the checksums themselves.

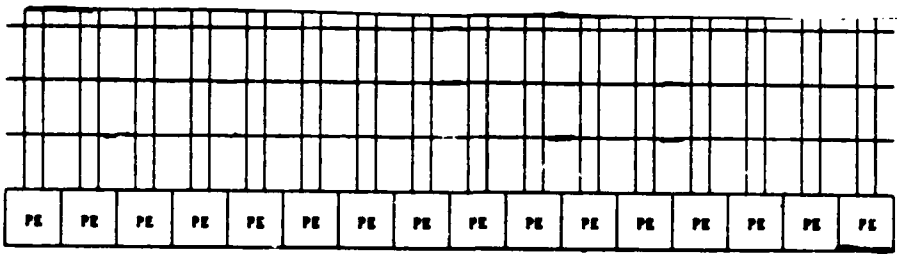


Figure 4. The Fault-Tolerant Line Hypergraph [34]

5.4 Systolic Array PE Study

Systolic arrays may be configured as shown in Figure 4. They include rectangular, hexagonal, or linear systolic arrays. The most likely use for each configuration is indicated. In the application domain of beamformers for towed arrays (for example) it is suggested by many researchers that a triangular array is preferable. Unfortunately, all currently available systolic arrays including the NCR GAPP (Geometric Arithmetic Parallel Processor), incorporating 72 PE's are configurable in rectilinear (6 X 12 units) not triangular fashion. Hence, a triangular array configuration although optimal from an algorithmic standpoint (e.g., recursive LS) does not efficiently utilize commercial arrays.

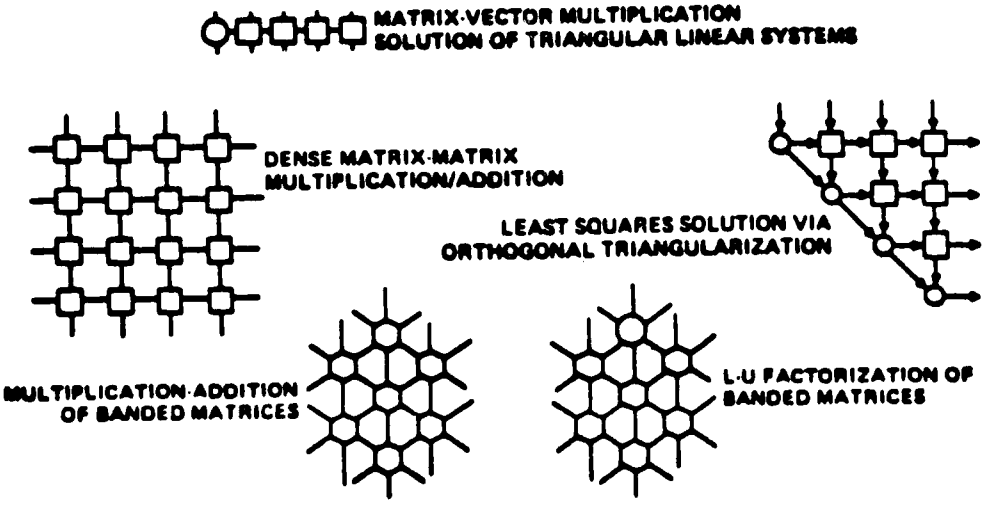


Figure 5. Systolic Array Solutions

Our methodology was to map a class of algorithms onto a non-existing machine. To do so, we must first specify the design constraints such as circuit switching speed, propagation delay throughput, maximum number of gates, etc. Next, we must bound the algorithm classes. Within each class we should first determine the greatest common denominator or building block. In adaptive signal processors, the inner product processor appears to be a suitable common denominator and starting point. Our strategy then is to derive near optimal algorithms invoking this basic signal processing operation, and then map the fast algorithms onto new VLSI circuits. Of course, not every algorithm may be based on this sole operation.

The methodology is a two-step process. In the first step, we want to obtain fast adaptive signal processing algorithms. Here we will bound the problem to study recursive and non-recursive adaptive algorithms. During this step, we will be sensitive to the computational processes which are expensive, such as matrix manipulations. Realizing that recursive algorithms contain basic computational tasks, identical to non-recursive algorithms, we will begin with non-recursive algorithms. Here, we want to identify inherent parallelism possible with adaptive signal processing algorithms.

The inherent parallel nature of an algorithm is then displayed by mapping the initial adaptive algorithm into a sequential set of tasks (commonly called "straight-line" algorithms) to be represented by a directed acyclic flow graph (DAG), each node being a task (multiply, divide, etc.) and each edge or vertex representing a data dependency relation. That is, briefly, predecessor nodes compute data needed by their successor nodes. From this graphical setting, we can reduce the longest or critical path by hand (if obvious) or by computer (using well-known graph reduction algorithms, c.f. Chapter 6 of [55]). Any concurrency so identified will provide us with speed-up via parallelism. One method to obtain concurrency is to use the adjacency matrix of the flowgraph to compute the earliest and latest precedence relationships. Map these onto a resource matrix (machine environment such as number of adders, subtracters, multipliers, convolvers, etc.) to identify concurrency. Another method is the divide-and-conquer scheme proved successfully in polynomial multiplication.

To date, the complexity of most signal processing algorithms has been estimated from their number of multiplications and sometimes from their number of additions. This is not always prudent. In fact, we should say an algorithm is deemed to be efficient if its final implemented form takes minimal time. The execution time consists of data snuffling operations as well as arithmetic operations. Hence, algorithms with fewer mathematical operations alone may not always be the best in its final implemented form. Fast algorithms identified in this step will most likely be modified later to insure optimal implementation. However, these initial results will serve as a good starting point. The best approach seems to be to first design an algorithm which is efficient in terms of the number of mathematical operations, and then modify it to take full advantage of VLSI characteristics.

As Lamagna [56] has pointed out, "The straight-line algorithm paradigm neglects the cost of the overhead associated with loop control and testing operations, as well as the time required to fetch and store information inside a computer's memory. These costs can vary greatly from computer to computer and will not even be the same for two programming language compilers

implemented on the same machine. Fortunately, the overall times of the algorithms studied are driven primarily by the underlying structure of the arithmetic operations performed, rather than such overhead considerations, so the results obtained are generally accurate to within a small constant factor for actual implementations."

A second step is to organize the VLSI for the fast adaptive algorithms. One basic building block is the inner product processor. Rectangular and hexagonal geometries can be incorporated. We intend to organize the inner product processor as efficiently as possible in array structures in order to capture the inherent pipelining, parallelism, and recursive nature of the adaptive signal algorithms. We anticipate that cyclic convolution and a cyclic convolver may be quite beneficial in casting the algorithms into VLSI.

The basic multiplication step itself was examined. The employment of distributed arithmetic implementation [30,57,58] successful for fixed-point digital filters was evaluated on an area x time basis for adaptive algorithms. Because adaptive algorithms, like filter algorithms, are essentially finite state machines, the multiplication and addition steps can be replaced by a partial table look-up of precalculated products. The analysis is, then, a trade-off between multipliers and memory space on the chip. This comparative analysis is not trivial since the recursive nature of IIR adaptive algorithms forces us to compute an entire result before reloading data registers (temporary scratchpad space) to generate the next table look-up entry. However, some pipelining is possible and can be exploited as much as possible.

A tentative method to wire up the algorithms is to use the "evaluation-interpolation" method successfully employed in [59] to obtain (area X time) optimal convolvers by observing the necessary algebraic steps and polynomial evaluations that can be cast directly into a parallel computational process. These algebraic steps, as organized, nicely prescribe optimal VLSI structures. Computational tasks can be divided into those circuits which are amenable to regular and simple interconnections and those which are not. Matrix multiplication tasks obviously can be regularized. ADC, DAC, and other analog computational tasks are not amenable to regular structures as we presently know them. The control circuits (such as found in firmware-oriented architectures) are amenable to regular implementations.

5.5 Error Tolerant Design with Multi-Valued Logic (MVL) Circuits

In this research, the effectiveness of MVL circuits realizing signed binary number arithmetic must be considered with respect to the inherent fault-tolerance of MVL circuits. Polylogic logic circuits, of which MVL is one case, have been studied by Porter [60] for intrinsic error tolerance. Our work plan is to use his technique to prove out low rejection rate and/or reduced component stringency requirements. Here, logic circuit failures (such as "stuck-at faults") and the effects of resistor, capacitor, and inductor error values (necessary for hybrid signal processors which incorporate ADC's and DAC's) should be studied. Note that possibly small fluctuations in component values are not appropriate for binary circuits; however, they are quite relevant and natural in MVL. Polylogic families include binary, multi-value, and threshold logic.

The procedure is to define a finite alphabet (R) and identify the set of all possible values of switching tuplets (R^n). Then, multilinear mappings onto the finite R are sought which eventually produce polynomic realizations of the desired switching function. The major question is, "Does any mapping exist, if component errors are modeled and included in our polylogic MVL subset?" Porter has already shown that such mappings exist and, in fact, several do. Hence, a circuit designer can choose among the more optimal circuits, performing engineering trade-offs as needed. This is the flexibility possible in this research to obtain fault-tolerant MVL circuits that are optimal in the sense of circuit complexity, power, and speed.

5.6 Design for Testability

Efficient test generation for logic circuits is a matter of prime importance [61-63]. Yet most major fault detection problems are to be NP-complete, generally. MVL is no exception. Hence, design for testability is necessary. Built-In-Test (BIT) circuits are highly desired. The study by Fujiwara and Toida [64] can be used to compare our fault-tolerant "testable" designs with their benchmark complexities. They also provided clever procedures to insert a few additional test-points into an arbitrary circuit to make it easily testable. Heavy use of PLA's is made. Their studies show that some circuits (linear circuits, decoders, parallel adders, ...) can be "tested" in polynomial time.

5.7 Redundancy for Increased Yield

Typically, for a new and complex device, most of the chips from a manufacturing batch contain defects. The yield is quite low. This can affect both cost and reliability.

During the fabrication process, defects that can result in faults can occur at any time during processing. For a chip at N circuits, there will be l_i fault-causing defects introduced during fabrication process step i. In all we can expect to find $L = \Sigma l_i$; fault-causing defects (14). If all these defects are Poisson distributed, then the yield will be given by

$$Y = e^{-L} \tag{14}$$

assuming random point defects are our only yield detractors. In general, fault-causing defects will not be randomly distributed but will be clustered. Furtnermore, the clustering nature will vary from step to step. Nonetheless, the assumption of gamma distributed defects, where the same clustering parameter, a, characterizes all the defects, leads to the following yield formula (15) that has been successfully used to model a large body of data.

$$Y = (1 + L_0/a)^{-a} \tag{15}$$

where L_0 is the average number of fault-causing defects per chip. In the limit that $a \rightarrow \infty$, (15) reduces to (14). In actual situations, a is typically in the range 1/2 - 4, and the yield can be appreciably better than predicted by (14). In the case of redundant designs as may be required for Wafer Scale Integration (WSI), the calculation of yield becomes more complex, and the role of clustering and correlation of defects becomes even more important.

Yield projections are a primary consideration in wafer scale integration. Ketchen [65] develops a point defect yield model for a two-way redundancy scheme appropriate for random logic. The model assumes that the fault-causing defects are randomly distributed locally but that the defect density can vary across a wafer as well as from one wafer to another. The importance of the distinction between on-wafer and wafer-to-wafer variations in defect density is demonstrated. This model demonstrates the dependence of yield on the nature of the defects, and, together with gross yield estimates and the appropriate nonredundant yield factor, it will serve as a good starting point to model actual yield data. The existence of complex local correlations and some non-point-like defects will clearly complicate matters; although, in many cases, a perturbative approach is adequate to model the situation.

Redundancy can be used to improve the yield significantly. Such methods are commonly used for memory chips. Faulty components are left out of final interconnection. The strategies used include eliminating affected row and column, or eliminating the affected half. A processor array can be reconfigured in more complex arrays [66-68]. To obtain an array of specified dimensions, one would then start with a larger and thus redundant array. Redundancy does not always increase the yield, because the larger chip area required tends to decrease the yield. Using Koren and Breuer's approach [66], expressions for yields for both simple and fault-tolerant arrays can be obtained, and optimal designs which maximize yield can be obtained. Faults affecting both PE's and interconnections have to be considered.

5.8 Fault-Tolerance for Higher Reliability

Dynamic reconfiguration can be used to overcome hard faults occurring in the field. Before any PE's are removed from active configuration, it would be necessary to detect such failures. To control error propagation, such detection should have low latency (time between error occurrence and external fault manifestation). For concurrent testing, duplication is the most effective technique; however, it can significantly reduce yield. It has been shown that some self-checking with limited redundancy can significantly improve yield [66].

Dynamic reconfiguration in processor arrays can be done in different ways [67]. Because reconfiguration results in fewer PE's, there is some degradation in performance. The effectiveness of different schemes depends on efficiency of partitioning of algorithms for execution on reduced size arrays. Trade-offs between reconfiguration strategies must consider optimizing, reliability, coverage, performability [69] and computational availability.

Processor arrays can support a special form of fault-tolerance. In real-time applications, successive data points can exhibit considerable correlation. A sudden and significant change in a point array may suggest the onset of a soft (temporary) or hard fault. Correctness of the value can be confirmed by recomputation (which is a form of time redundancy); however, the same faulty hardware, because of a hard or a long soft fault, is likely to generate the same incorrect result. However, in a processed array, recomputation can be done by mapping the process to a shifted set of PE's. In this case, a faulty PE will almost always generate a different result.

A considerable degree of fault-tolerance can be achieved by encoding information. Kuang and Abraham have described a scheme for matrix multiplication with processor arrays which requires only limited hardware and time redundancy [52]. Suitable SDNR arrays are available.

5.9 Hard/Soft Errors

Reliability with respect to hard and soft faults will be considered separately. While methods exist which use the same measure to include effects of both types of faults, such a measure can be hard to interpret.

For binary devices, the failure rate is generally estimated by using techniques in MIL-HDBK 217 and its updates. The fact that the learning and the quality factors alone can change the result drastically suggests that exact results can not be expected. By characterizing L (see Section 5.1.1) itself by a statistical distribution, these limitations can be taken into account.

It can be expected that the failure rate data for binary devices is not directly applicable to ternary devices. The physical degradation, that will not cause a logical failure in a binary device, may cause a failure in a ternary device. On the other hand, a ternary device uses fewer logical nodes, interconnections and specialty pins, which can significantly enhance reliability. How the available data on failure rates for binary devices can be adapted for ternary devices will be a problem to be examined.

The alpha-particles have been a major cause of soft failures. However, now they can be very effectively combated by proper choice of encapsulating material and by coating. Also, the new CHMOS technology is remarkably robust against alpha-particles. Various types of noise [70] remain a problem. Here reduced noise immunity makes noise an important consideration. Soft failure rates due to such causes can be estimated satisfactorily, but assumptions remain to be examined.

5.10 Design-For-Testability and Built-In-Self-Testing

Efficient test generation for logic circuits is now recognized to be a matter of prime importance [61-63]. Yet most major fault detection problems are generally NP-complete. The proposed MVL PE array is no exception.

In a regular array, there are two major testing considerations. One is how to test a single PE element, assuming its inputs and outputs can be directly accessed. Next, part of the problem is how to exercise each PE element when they form a regular array. Some arrays possess a special feature called C-testability [71]. A C-testable array can be tested by wiring a fixed number of tests, regardless of the dimensions of the array. It has shown that often arrays which are not C-testable, can be made so by using only minor modifications [72].

Several scan-path techniques like LSSD have been suggested. These reduce the problems of testing sequential circuits to that of testing purely combinational circuits. This enormously simplifies test-pattern generation. The scan-path techniques are also applicable for PE arrays. An implementation

has been described for a CMOS, two's complement serial convolver chip [73]. Applicability of Ternary-Scan-Design, as proposed in [74], for our proposed scheme is relevant.

The study of Fujiwara and Tioda [64] compares fault-tolerant "testable" design with benchmark complexities. They also provide clever procedures to insert a few additional test-points into an arbitrary circuit to make it easily testable. Heavy use of PLA's is made. Their studies show that some circuits, decoders, parallel adders, ... can be "tested" in polynomial time.

Built-In-Self-Test circuitry allows a device to test itself without using expensive test equipment. It is also valuable for assuring device integrity in the field. For a PE array, BIST must be incorporated within each element. It is also necessary to have the necessary circuits to support BIST globally so that the interconnections are tested, and also the go/no go information is routed to some external output or outputs.

5.11 Information Redundancy

Low cost residue and inverse residue codes for error detection in signed-digit arithmetic were proposed for this project. These codes capitalize on the fact that they can be used to check storage, transmission, and computing functions using the same checking algorithms. These algorithms compute the module: a residue of messages, operands, and results in a serial or parallel fashion. The residue digits are then tested to indicate whether or not an error exists [50].

As noted by Avizienis [50], the effectiveness of Signed Digit Residue Codes (SDRC) can be assessed by observing that undetectable errors are caused only by faults that change the value of the signed-digit number by a multiple of 2^b-1 (where 2^b is the radix). Such changes are highly unlikely. A detailed study of effectiveness requires the full knowledge of the internal representation of digit values and an analysis of the effects of repeated-use faults when they may affect the operands or the result.

The algorithms proposed in [50] only employ one residue digit for an entire K digit SD operand. While this minimizes the cost of encoding, it may be inconvenient in variable-precision operations that generate the most-significant-digits of the results first and that are "chained", executing further operations on high-significance-digits of an intermediate result X even before the lower-significance-digits become available. The Serial Checking Algorithm is completed only after all digits of X have been obtained, the residue digit X is then computed and compared to test for the presence of an error.

An error indication requires the cancellation of all results that have used at least one digit of X. The cancellation must reach k+3 digit levels downstream in the chain and identify all potentially erroneous results. Two solutions may be applied to shorten the "span" of the cancellation that must follow an error indication: (a) the **segmentation** of operands into check segments, and (b) **single-digit encoding** that employs a checking element within each arithmetic unit that performs single-digit operations.

Segmentation divides the k-digit operand into check segments of p digits length each and attaches one residue digit to the right end of each check segment, rather than using one residue digit at the end of the entire operand. The cancellation span is reduced to p+3 digit levels downstream in the chain. Furthermore, error detection effectiveness in the case of repeated-use faults may be increased because of the shorter length of the segment being checked. The cost of segmentation consists of the extra time and storage required by the proliferation of residue digits.

Single-Digit Encoding appears most suitable for VLSI-implemented arithmetic units that can execute the algorithms for digits of S-D representations with relatively large radices, such as $r=2^8$, $r=2^{12}$, $r=10^3$, or even greater values. Here each individual S-D representation digit carries its residue digit modulo (r^*-1), where

$$r^* = 2^q \text{ when } r = 2^b, \text{ and } b = kq \ (q \geq 2) \tag{16}$$

$$r^* = 10^q \text{ when } r = 10^c, \text{ and } c = kq \ (q \geq 1) \tag{17}$$

The single-digit encoding approach is an extension of the segmentation concept. Each digit of the radix $r = 2^b$ is treated similarly to a k digits long segment of the radix 2^q representation that is checked by one modulo 2^q-1 residue digit. The evident advantage of this approach is the pinpointing of the error to the single arithmetic unit.

5.12 Hardware Redundancy

Some drawbacks of arithmetic codes are their inability to detect errors in logical operations, and single errors in group carry-lookahead structures [47]. The latter is not a problem if SBNR is used. Thus, hardware redundancy has been recognized as the most effective technique to identify faults in logical operations. In [47], Patel and Fung describe a technique in which coding and decoding functions (in the form of shift left and shift right) are employed. Here, the arithmetic/logic operation is performed twice. The first time it is performed without shifting, and the results are stored in a general register. The second time, the inverse shift operation is executed and then compared with the contents of the storage register. A mismatch indicates an error in computation.

The hardware redundancy technique described has been implemented in binary number systems. Nevertheless, it is prudent to assume that it, or any other binary technique, can be adapted to SBNR architectures. Of course, a trade-off study of cost versus circuit complexity should also be completed.

The binary fault-tolerant ALU implemented by Patel and Fung can be constructed using a CMOS family of ternary logic circuits. These circuits, proposed by Mouftan and Heung [75], use two power supplies, each below the transistors threshold voltage, and do not include resistors. All transistors are 5 microm x 5 microm. The threshold voltages for the p-channel and n-channel enhancement-type transistors are -1v and +1v. They have opposite polarity for the depletion-type devices. With the use of voltage power supplies below the transistors turn-on voltage and the exclusion of resistors, it is possible to implement this circuitry in VLSI. Added features include

low power consumption, high speed, and comparable performance to their binary counterparts.

For the ALU proposed in [47] to be fault-tolerant, the encoder, decoder, and comparator circuitry must be Totally Self-Checking (TSC). They can be implemented with PLA's. One advantage of using PLA's is that their regular structure simplifies analysis of the effects of faults on their output and therefore facilitates test vector generation and determination of fault coverage.

The most elementary fault model used for PLA's includes three types of faults:

1. Stuck-at faults on an input line, product term line, or output line.
2. A short between two adjacent or crossing lines that forces both of them to the same logic value.
3. A missing or extra crosspoint device in the AND array or in the OR array.

Since breaks in lines (that are not equivalent to stuck-at faults) are one of the main causes of failures in VLSI circuits [76-77], it is clear that the above simple fault model does not accurately reflect the possible physical defects in an MOS PLA. A more complete fault model is given in [78].

6.0 Research Results of Current Period

6.1 SBNR Architectures

This PI has been investigating the Least-Mean-Square (LMS) adaptive filter algorithm for signal processors [79-81]. Recently, these studies have focused on redundant arithmetic implementations in distributed and systolic array architectures [20,82,83]. It has been discovered that some of the inherent borrow/carry propagation properties tend to make implementations very compact and modular. This tends to suggest that fault-tolerant properties abound for SBNR realizations. As early as the ILLIAC III, Atkins [8] showed that higher radix implementations (of which SBNR is a reduced yet very powerful subset) produced superior fault-tolerant arithmetic engines when using redundant or signed number codes.

The papers in the Appendix by corporate personnel have demonstrated some of the advantages to SBNR. Note particularly that others are identifying similar advantages. Sicuranza and Ramponi [84] also exploit memory-oriented structures properly matching the characteristics of distributed arithmetic for adaptive nonlinear filters described by truncated discrete Volterra series. Their use of offset binary code (a form of SBNR) and address splitting (available to SBNR) established efficient, although dedicated, architectures. They, as well as us, show that the memory dimension is <u>not</u> $(2^n)^K$ words because of the dramatic reductions possible with SBNR and symmetry.

Another promising approach to efficient implementations of redundant number realizations is described by Owens and Irwin [85]. Here, a primitive cell, including its operation suite, are used in a DFT application demonstrating the highly regular array structures achieving good AT^2 bounds. They partitioned functions into "interface, storage, or arithmetic" to implement digit-on-line algorithms. We can exploit the same digit-on-line

properties for ultra-fast processing of analog signals. For example, digital signal processing can begin as soon as the MSB (which is the first bit) is conveted by the ADC! For bit serial distributed arithmetic schemes, Linderman [86] has shown how clock rates of 70 MHz are possible here. Denyer [87], Denyer and Renshaw [88], Jaggernauth, et. al. [72], and others [89] make similar promising discoveries about bit-serial implementations. Particularly encouraging is the CUSP (digital signal processor, VLSI) of Linderman, et. al. [90], since this device is a sixteen 20-bit serial multiplier by 24 serial adder/subtracter, driven by a 5[illegible] MHz two-phase non-overlapping clock. This device again exemplifies the power of bit-serial approaches.

6.1.1 Design of SBNR Array Multiplier

The need for high-speed computation has spurred much research into various forms of parallel processing. The two most common of these architectures in signal processing applications are the pipelined processor and the array processor. Developments in parallelism have become quite popular with the revival of interest in the Signed-Digit Number Representation (SDNR) characterized by Avizienis [1]. Implementation of the faster architectures in VLSI is a concern for devices needing powerful processing in limited space (e.g., mobile, self-contained and space-based vehicles.

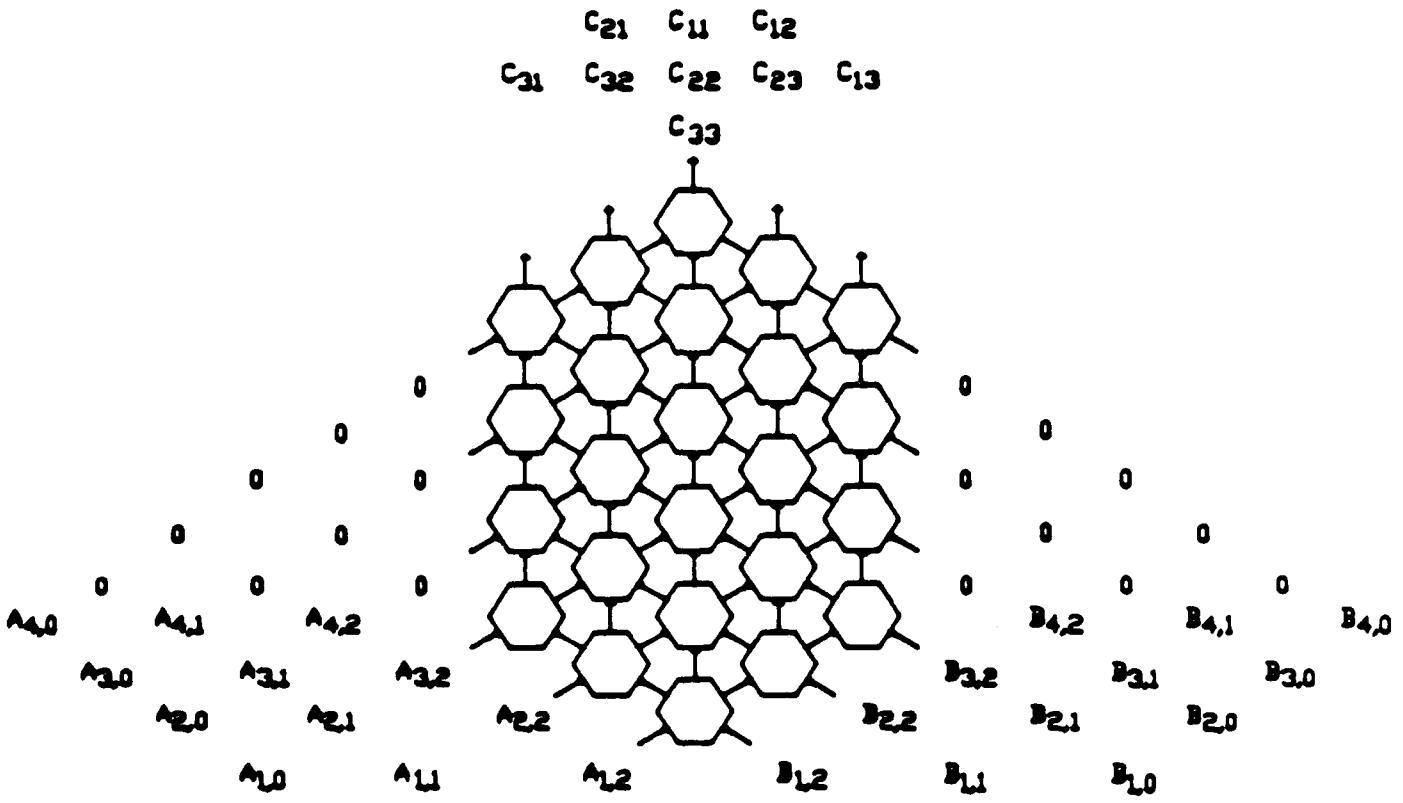
The design described below, and shown in Figure 6, is a systolic array for matrix multiplication which is compatible with digit online architectures [91]. This array is similar to one described by Irwin [92] in which two vectors to be multiplied enter the array most-significant-bit-first--the distinguishing characteristic of online networks. The array uses the Processing Element (PE), diagrammed in Figure 7, and shown schematically in Figures 8 and 9, to perform bitwise vector multiplication. For matrix x matrix multiplication a parallel multiply/accumulate element may be substituted for the bit-level PE. In such a large scale system, asynchronous operation may be faster than the clocked method shown.

If the array is used for vector multiplication, it performs the operation

$$\begin{bmatrix} A_1 & A_2 & \cdots & A_m \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_m \end{bmatrix} = C, \text{ where } C = A_1B_1 + A_2B_2 \cdots + A_mB_m.$$

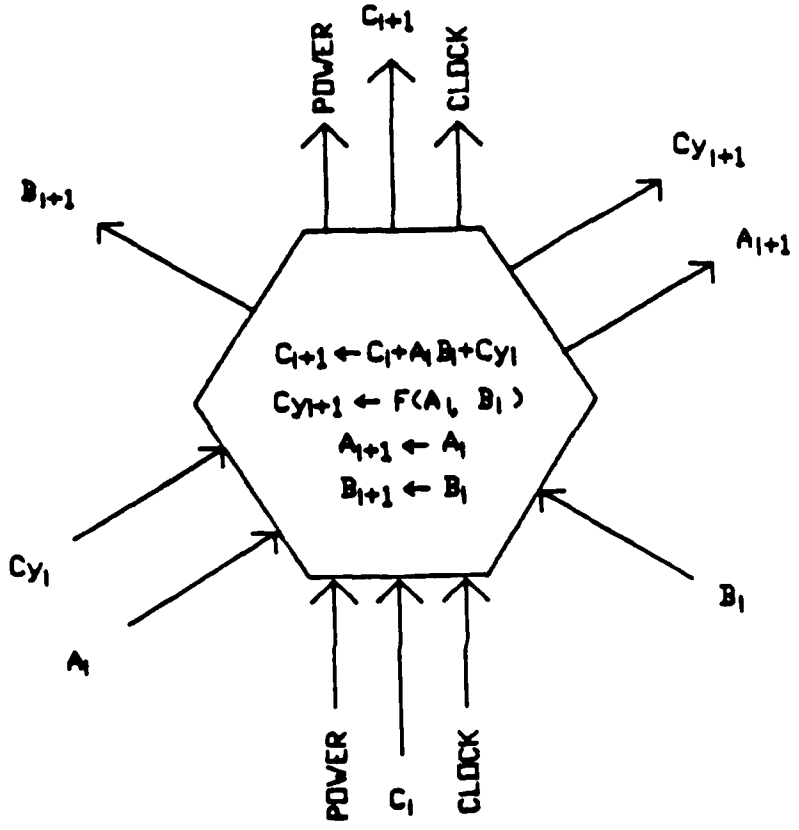
Each vector element is a word consisting of n signed-binary bits, where each bit may be a '1', '0' or '-1'. When the vector elements

Figure 6. Systolic Array Multiplier

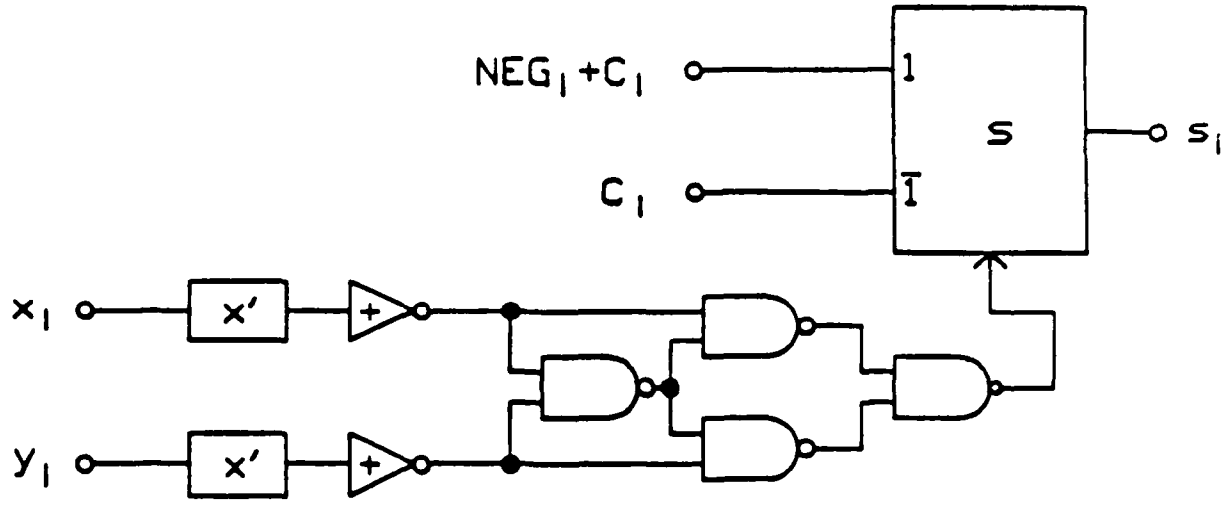


MATRIX MULTIPLIER

Figure 7. Functional Diagram of Array Cell



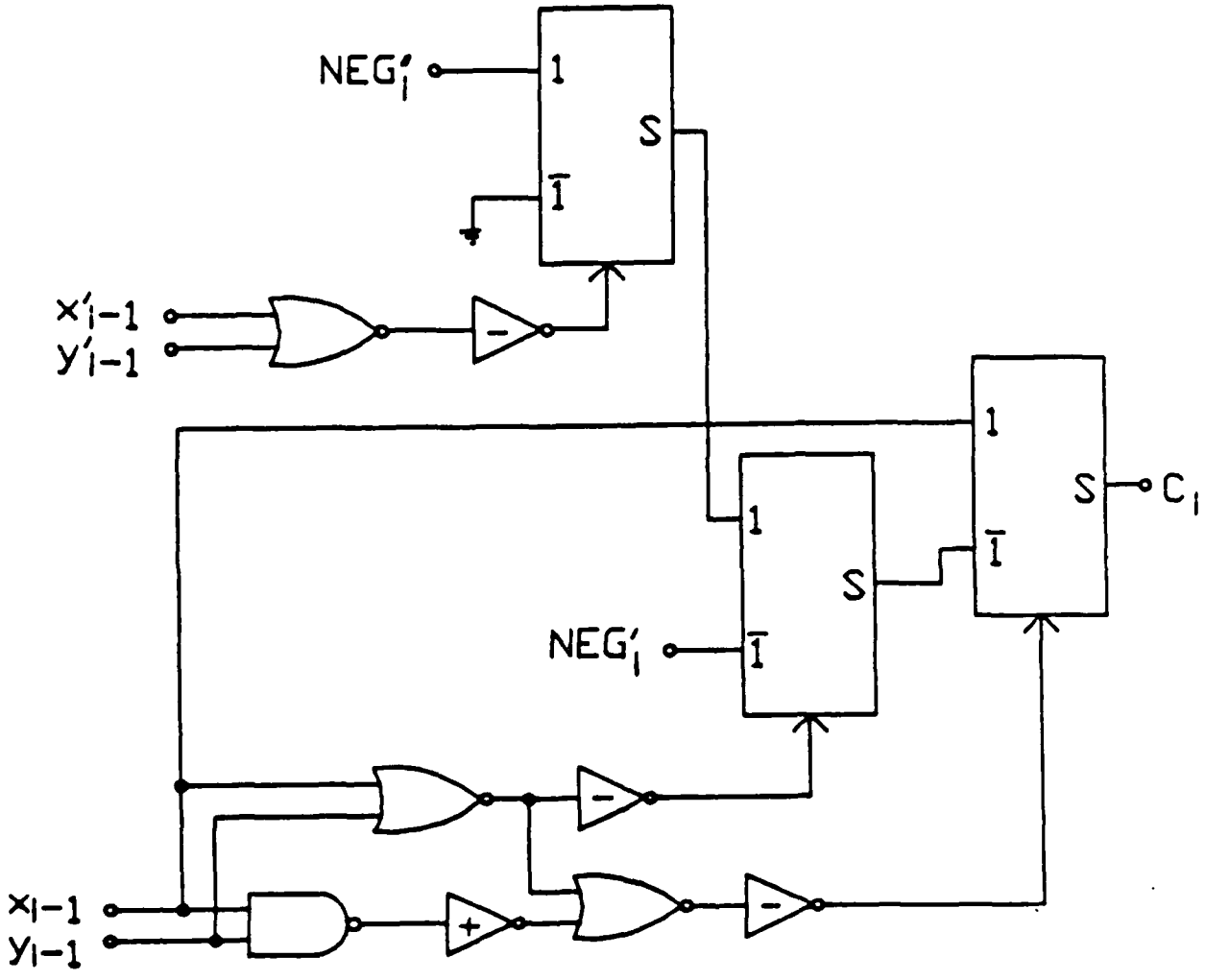
ARRAY CELL



x_I	y_I	S_I
$\bar{I}$	$\bar{I}$	C_I
$\bar{I}$	0	C_I+NEG$_I$
$\bar{I}$	1	C_I
0	$\bar{I}$	C_I+NEG$_I$
0	0	C_I
0	1	C_I+NEG$_I$
1	$\bar{I}$	C_I
1	0	C_I+NEG$_I$
1	1	C_I

Figure 8. Sum Generation Circuit and Truth Table

Figure 9. Carry Generation Circuit



are expanded into bit representation, one can see that the systolic array is actually performing matrix multiplication on the bits of the vectors:

$$\begin{bmatrix} A_{1,n-1} & A_{2,n-1} & \cdots & A_{m,n-1} \\ \vdots & \vdots & & \vdots \\ A_{1,1} & A_{2,1} & \cdots & A_{m,1} \\ A_{1,0} & A_{2,0} & \cdots & A_{m,0} \end{bmatrix} \begin{bmatrix} B_{1,n-1} & \cdots & B_{1,1} & B_{1,0} \\ B_{2,n-1} & \cdots & B_{2,1} & B_{2,0} \\ \vdots & & \vdots & \vdots \\ B_{m,n-1} & \cdots & B_{m,1} & B_{m,0} \end{bmatrix}$$

$$= \begin{bmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,n} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,n} \\ \vdots & \vdots & & \vdots \\ C_{n,1} & C_{n,2} & \cdots & C_{n,n} \end{bmatrix} \qquad (19)$$

The bits of the answer are found by adding C along the lower-left-to-upper-right diagonals:

$$
\begin{aligned}
C = {} & C_{1,1} \; C_{1,2} \; C_{1,3} \; \cdots \; C_{1,n} \; C_{2,n} \; \cdots \; C_{n-1,n} \; C_{n,n} \\
& + \; C_{2,1} \; C_{2,2} \; \cdots \; \cdot \; \cdot \; \cdots \; C_{n,n-1} \\
& + \; C_{3,1} \; \cdots \; \cdot \; C_{n,2} \\
& + \; C_{n,1}
\end{aligned}
\tag{20}
$$

To see how this happens, refer to Figure 6. On the first clock cycle, $A_{1,2}$ and $B_{1,2}$ (the MSBs of A_1 and B_1, respectively) are shifted into the bottom cell, multiplied, and added to $C_{1,1}$ (initially zero). All the other MSV's are shifted into their first cells. On the next cycle, $C_{1,1}$ shifts up, and adds to the product of $A_{2,2}$ and $B_{2,2}$. The products $C_{1,2}$ and $C_{2,1}$ are formed to the right and left, respectively, of $C_{1,1}$. The entry $C_{2,2}$ is started just below $C_{1,1}$. Subsequent clock cycles shift the multiplicands in, and the answer out in the order shown in Figure 6.

The total time for this calculation is given by

$$T = (2n + m - 1)t, \tag{21}$$

where t is the time for each clock cycle. Since this is a digit online network, calculation is started on the MSB's before the LSB's are needed. Another significant measure of performance is the time between the entry into the array of the MSB's, and the exit from the array of the answer MSB. This time may be calculated by

$$T_{MSB} = (n + m - 1)t. \tag{22}$$

In VLSI applications, the number of elements and the number of interconnections are both significant. These values are given by

$$N_E = n^2 - 2n + 2nm - m + 1 \tag{23}$$

and

$$N_I = 5n^2 - 18n + 10nm - 9m + 12. \tag{24}$$

Note that all of the equations are also true for the matrix x matrix multiplication, in the fully digit online case. If the data are shifted in parallel, more interconnections will be needed.

6.1.1.1 Observations

Reports by Mouftah [93] and Aytac [94], on which much of the PE is based, indicate that the addition logic may be too slow for many applications. The speed is not known absolutely, however, because the logic gates presented in the reports are based on use of 5 um line widths. Implementation of the array multiplier would be in 2 or 1 micron technology, which could result in a significant speed increase.

Another possible problem is the transmission of two control signals, carry and neg, through the array. Though these signals should be local, they are not synchronized with the clock, and may not be nearest-neighbor transmittable. Further research into this problem will probably yield a satisfactory local-communication solution.

Future work in this area could include a comparison between this network and alternative matrix multipliers, including pipelined parallel or digit online multipliers, binary and higher-radix implementations, and different array configurations. In addition, the PE cell should be simulated using a MOS simulation program, and characterized in terms of speed and area. Using this data, or making logical assumptions about it, the speed and area of the entire network will be calculated from the formulas presented earlier. In addition, the speed and area of alternative PE's should be investigated and compared.

6.1.2 Digit Online Vector Multiplier Using SBNR Adder Tree

In pipelined signal processing systems, the maximum rate of data flow through the pipe is determined by the slowest element. Traditional pipelined systems consist of a few slow elements, connected by parallel data paths. In digit online systems [91], a redundant number system is used to allow data to flow as a stream of bits, with the most-significant-bit leading the stream.

Irwin and Owens have identified many advantages to this mode of operation. The first is that the bit-stream approach allows the system to perform bit-level operations on the data. Since the slowest of these operations is much faster than the slowest word parallel computation, a much faster clock rate may be used, possibly increasing data throughput. The second advantage of digit online architectures is that result bits can begin streaming out of a processor after only a small online delay from the start of the input data. The result can then be used in the next processor. This effect allows several links in the processing chain to operate simultaneously on results generated from a single data word. Thus, the effective throughput of an element is determined more by its online delay (latency) than the total time of computation. The third advantage of digit online systems is that of chip pinout. Since the data are transmitted in bit-serial mode, the number of pins on a chip does not depend on the length of the data words.

Space Tech has investigated a digit online multiplier that computes the fixed-point inner product of two vectors. The vector elements arrive simultaneously on separate data paths in bit-serial format. The multiplier can accept either the most- or least-significant-bit-first with no change in calculation time. The answer bits appear in the same order as the input. The multiplier uses Signed Binary Number Representation (SBNR) to allow fully

parallel addition internally and to make it compatible with digit online systems.

Inner Products

In many digital signal processing applications, the primary function of the processor is finding the inner product of two vectors: $\underline{a}^T \underline{b} = c$. For two vectors $\underline{a}$ and $\underline{b}$, each composed of m elements, a_i and b_i, the inner product is defined as:

$$c = \sum_{i=1}^{m} a_i \, b_i \tag{25}$$

When each element is defined by an n-bit signed binary word, the vector multiplication can be decomposed into a matrix inner product on the bits of the vectors. Thus, if each element a_i is represented as

$$a_i = \sum_{k=0}^{n-1} 2^k \, a_{i,k} \tag{26}$$

then the inner product of the vectors can be rewritten as

$$c = \sum_{i=1}^{m} \sum_{k=0}^{n-1} \sum_{j=0}^{n-1} 2^{jk} \, a_{i,k} \, b_{i,j} \tag{27}$$

It is this function that the vector multiplier implements.

6.1.2.1 Vector Multiplier Structure

Figure 10 shows the architecture of the vector multiplier. The bits of vector $\underline{A}$ appear on the m-wide bus at the top. When the first bit appears, the MSB line is brought high, thus latching the MSB's of $\underline{A}$ into the first partial product cell. At the same time, the first bits of $\underline{B}$ appear at the cell (AB_1), and are multiplied by the corresponding bits of $\underline{A}$. The results are added, and the sum appears at the bottom of AB_1. The architecture for this operation is similar to the Takagi multiplier [95].

In the next clock cycle, the MSB signal is latched to the right, thus latching the bits on the $\underline{A}$ bus into the second cell. The MSB's of vector $\underline{B}$ also latch into the second cell, and the next bits of $\underline{B}$ appear at AB_1. Thus, AB_2 contains $\underline{A}_{n-2}$ and $\underline{B}_{n-1}$, while AB_1 contains $\underline{A}_{n-1}$ and $\underline{B}_{n-2}$. These partial products are then calculated. The product from the first cycle is latched into the top of the adder tree.

On the third cycle, the two second-level partial products from AB_1 and AB_2 are added together in the m-wide parallel adder. The $\underline{B}$ bits are right-shifted, and the third bits of $\underline{A}$ are latched into AB_3. The MSB partial product moves down to the next-level adder.

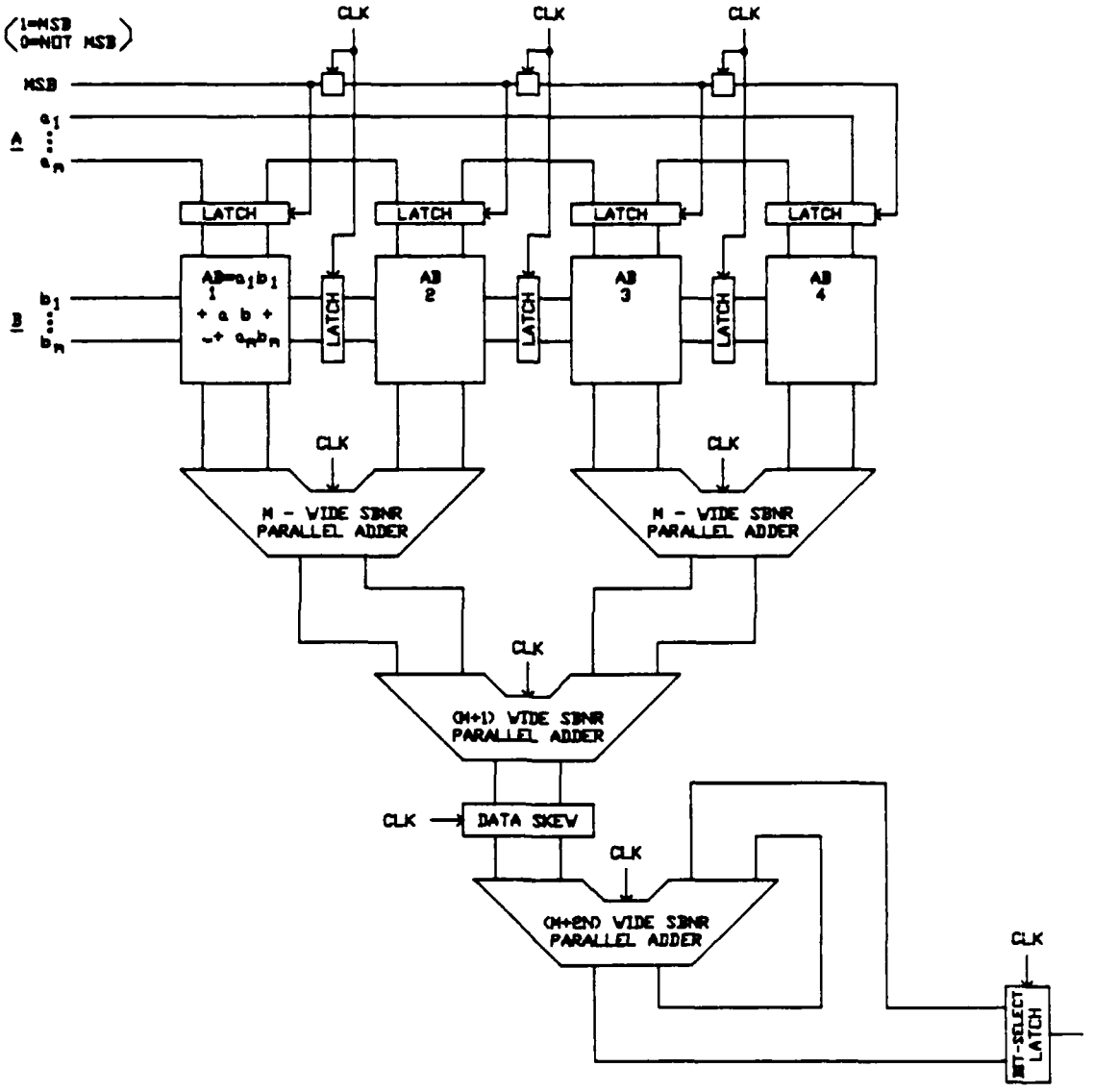


Figure 10. Digit Online Vector Multiplier

Similarly, each subsequent clock cycle generates the next-lower bit position of the product. Each of these cascades through the adder tree until it hits the data skew block.

The function of this element is to shift the incoming words to their proper bit position in the result. The skewed word is then added to the cumulative sum to produce the final sum. The result is shifted out of the bit-select latch.

6.1.2.2 Area and Time Complexities

The area and time factors of this multiplier are excellent for single-chip implementation. The area complexity is $O(mn+m^2)$, which is better than that of the bit-level systolic array [96]. The time for calculation is 2n clock cycles, but the latency is just $n+\log_2 n$, and is not dependent on m. For the case of m=16 and n=16, this multiplier takes 76,600 transistors, which is half the number needed for 16 paralleled Takagi multipliers. If the latency were reduced to something less than n, the next stage of the pipeline could operate on the inner product as it was being calculated. This reduction may or may not be possible, however.

In addition, the architecture seems too hardware intensive for a pipelined system. Further research should be done to try and use recursive properties to reduce or eliminate the full-parallel elements and/or the adder tree. The next step is investigation into alternatives, specifically, a structure like that proposed by Rhyne and Strader [97]. Any alternatives found should be characterized with respect to this and other architectures. If none are found to be better, more comparisons should be made between this multiplier and the alternatives.

6.1.3 **Systolic LMS Architecture**

Recursive least-mean-square algorithms have wide application in many types of estimation problems. One such application is adaptive beamforming. Beamforming is commonly used in radar and sonar applications, both in transmitting directed wavefronts and receiving from selected directions.

A trade-off exists between the speed of adaptation and the stability of the formed beam [98]. In general, the more recursions needed for adaptation, the more stable is the steady-state performance. However, an increase in system throughput will speed up the adaptation without affecting the steady-state stability. The architecture proposed by Space Tech provides the increased throughput needed for high bandwidth communications.

6.1.3.1 Recursive LMS Algorithm

Widrow's LMS algorithm consists of an adaptively weighted input stage and a weight update stage. We modified the algorithm to allow pipelining [83], but the architecture included two systolic array processors. An alternative design uses a pipelinable algorithm, but only a single systolic array. To see how the algorithm works, define the following variables:

K = system word length in bits
N = number of receiving antennas
S = k x n bit matrix of input samples
W = k x n bit matrix of weighting coefficients
$\underline{y}$ = bit vector of filter output
$\underline{d}$ = bit vector of input reference signal
$\underline{e}$ = d - y = bit vector of filter error
$\underline{u}$ = convergence rate factor

(The value of a bit vector is just the vector multiplied by powers of two:

$$y = [2^{k-1} \quad 2^{k-2} \ldots 2^1 \quad 1] \underline{y})$$

The output of the filter is given by

$$\underline{y} = S^T W \tag{28}$$

The filter weights, W, are determined by iteratively comparing the filter output to the training signal, d. This difference, when multiplied by the input matrix, gives the line of steepest descent toward convergence [98]. The equation to calculate the new filter weights is

$$W_j = W_{j-1} + 2ue_{j-1}S_{j-1}. \tag{29}$$

Left multiplying this equation by the current input gives

$$S_j^T W_j = S_j^T W_{j-1} + 2ue_{j-1}S_j^T S_{j-1}.$$

If we assume that the inputs are uncorrelated (i.e. $E[S_j \quad S^T_{j+1}] = 0$), then we can make the following approximation:

$$S_j^T W_j = S^T_{j-1} W_{j-1} + 2ue_{j-1}S_j^T S_j. \tag{30}$$

Andrews implemented a similar function using two systolic arrays, one to calculate $\underline{y}_j$ for S_j, and one to update W_{j-1}. The architecture below uses a single systolic array, combined with a small number of latches, serial adders, and serial multipliers. Also, since the weights have been removed in Equation 3, no weight-update phase is required.

6.1.3.2 Architecture Details

The arrangement shown in Figure 11 implements the LMS algorithm of (30). The systolic array multiplies the input by itself, S^TS. The output from the array is on K parallel lines, all of equal significance. Thus, at any clock cycle, each line carries a value in the same bit position as all the other lines.

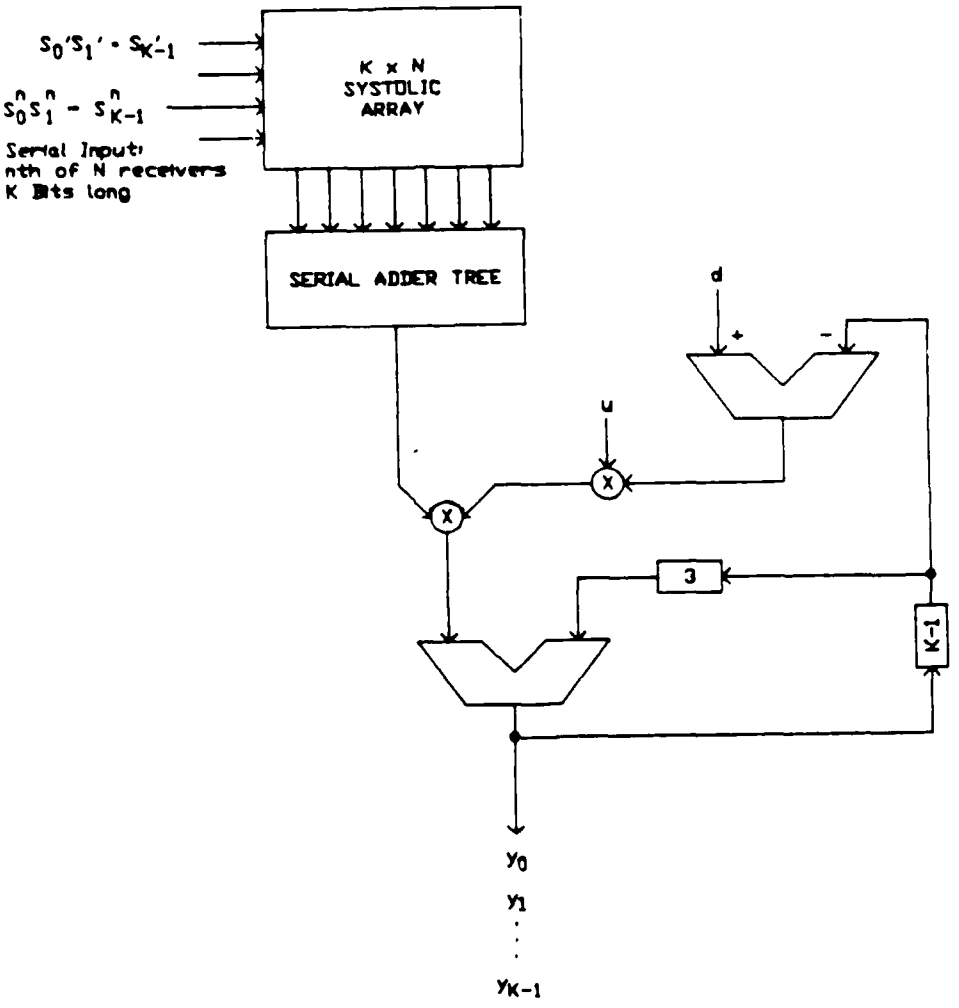


Figure 11. Recursive LMS Architecture

These wires are fed into a synchronous, bit-serial adder tree that accumulates all of the partial products. The result of this accumulation is the inner product S^*S, which is fed into a bit-serial multiplier at the same time as the product ue. The x2 factor results from bringing S^*S into the multiplier one clock cycle ahead of ue, thus performing a left shift on the product. The resulting value is the steepest-descent gradient of the error surface. The new output is formed by adding the error value to the old output.

If all arithmetic is performed with length-K operands and results, the latency of the architecture is

$$\text{latency} = N + \log_2 K + 1 \qquad (31)$$

and a new sample may be entered every K clock cycles. The clock period is determined by the speed of the serial multiplier. If Signed Binary Number Representation is used, all arithmetic inside the multiplier may be done in parallel, with a total delay of 35t, regardless of word length (t = delay of one transistor). Total device count for the circuit is:

KxN	Multiply - accumulators
K+1	Serial adders
2	Serial multipliers
Kx(N+2)	Latches

Thus, the device-latency product is $O(N^2K + NK\log_2 K)$.

6.1.3.3 An Adaptive Beamformer Application

Digital adaptive beamforming is commonly applied in communications. Applications range from voice communication over VHF/HF bands in the tens of KHz up to secure spread-spectrum data links with RF bandwidths in the 10 MHz area [99]. The latter case places strict requirements on the throughput of the processor. To allow sampling at the Nyquist rate each link in the pipe must have a bit delay no greater than 50/K nsec. For a 16-bit word length, each Processing Element must accept a new bit every 3 nsec, corresponding to a clock rate of 333 MHz.

This rate would be extremely difficult to sustain if the circuit were spread over a large board. Fortunately, the proposed architecture is primarily a systolic array with an adder tree, so the interconnections are regular and nearest-neighbor. Thus, most, if not all, of the circuit may be implemented on a single VLSI chip.

6.1.4 Time and Area Calculations for SBNR Array Multiplier

This section describes Space Tech's studies of a systolic array architecture which uses MVL and a SBNR to multiply two vectors. The architecture is compatible with digit online pipelined networks [9], in which data are transmitted serially and most-significant-bit-first. The array structure makes chip layout easy, and the local communication paths reduce interconnection area. Therefore, this systolic array should be easy to implement in VLSI.

6.1.4.1 Array Structure

The fundamental operation performed by the processor is the vector multiplication

$$\begin{bmatrix} A_1 & A_2 & \cdots & A_m \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \cdot \\ \cdot \\ \cdot \\ B_m \end{bmatrix} = C = A_1B_1 + A_2B_2 + \cdots + A_mB_m \qquad (32)$$

To maximize speed, one can use a systolic array of PE's in which many bit-level calculations occur simultaneously. Such a structure is shown in Figure 12. In this case, the vectors are four words long, and the words are three bits long.

Vectors a and b are shifted in from the bottom, most-significant-bit-first. The result c comes out the top, as shown. All carries from additions are transmitted asynchronously to the upper right of each cell. The PE to the upper right always holds one portion of the next-higher bit. The "horn" that extends up and to the right from the central multiplying core is present to add up the carries from lower-order bits. (Since the array uses SDNR arithmetic, a cell's carry depends only on its operands and the **neg** control signal from the lower cell, and not on the carry coming in to the cell. This limits carry propagation to a single cell, and gives signed binary a significant speed advantage over conventional binary.) As the answer is shifted out of the top, an adder tree adds up the bits that occupy each position of significance. The final result is then shifted out of the adder tree.

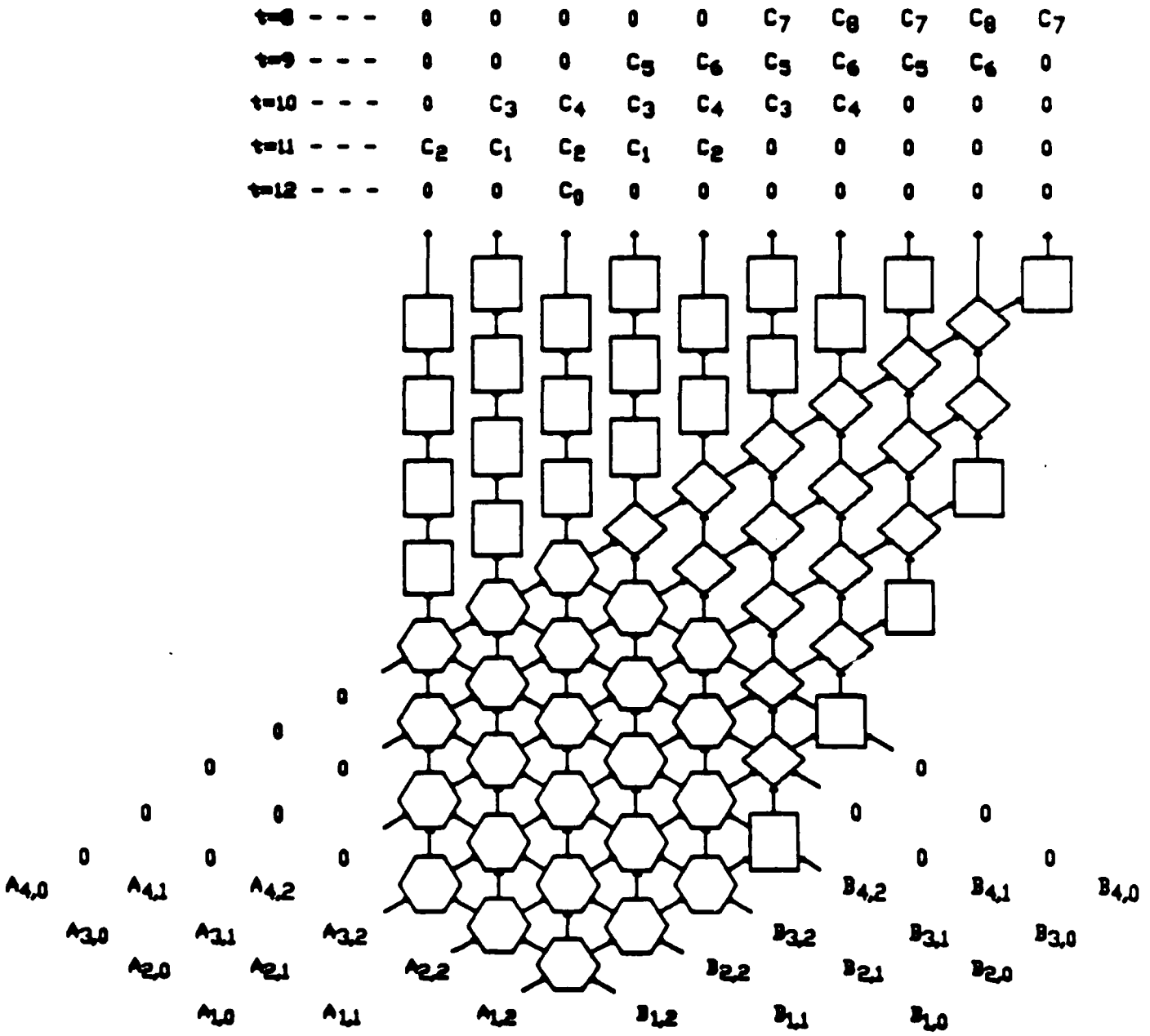


Figure 12. Systolic Array Multiplier

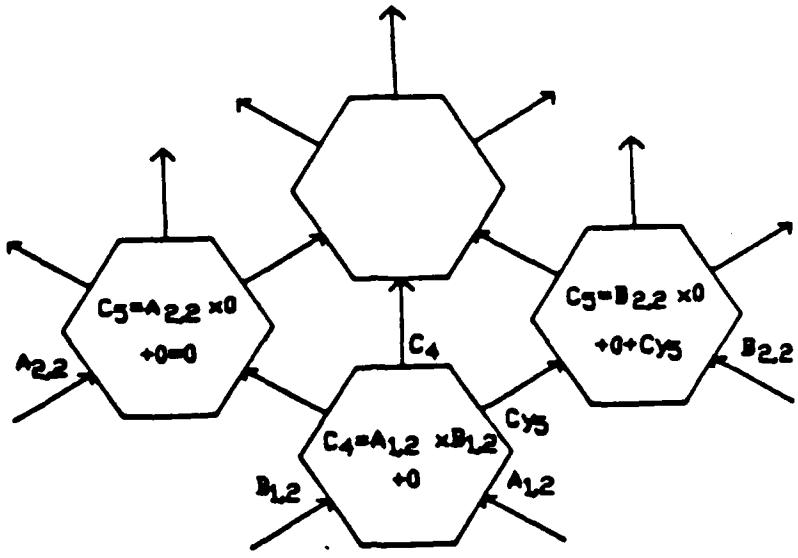


Figure 13a. First Clock Cycle

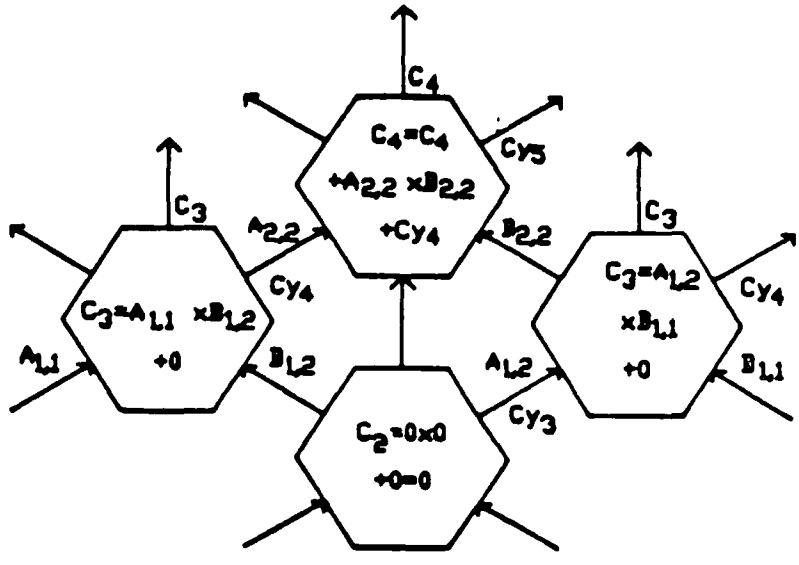


Figure 13b. Second Clock Cycle

The method of operation of the array is shown in Figure 13. On the first clock cycle (Figure 13a), the MSB's of A_1 and B_1 are shifted into the bottom cell and multiplied together. The result is added to C_4 (initially zero), and the carry travels up and to the right where it is added to C_5 (also zero). A control signal **neg** is also propagated to C_5 to be used in the addition. At the same time, all the other bits in the first row ($A_{2,2}$, $A_{3,2}$, ... and $B_{2,2}$, $B_{3,2}$, ...) are shifted into their first cells and multiplied by zero.

On the second clock cycle (shown in Figure 13b), the new C_4 and C_5 are shifted up, C_4 is added to $A_{2,2} \times B_{2,2}$, and that carry is added to C_5 as before. Any carry generated from the term $(C_5 + 0)$ propagates up and to the right into C_6. In addition, the terms

$$C_3 = A_{1,1} \times B_{1,2} \tag{33}$$

and

$$C_3 = A_{1,2} \times B_{1,1} \tag{34}$$

are generated just below, and on either side of, C_4. The shift registers at the top of the array ensure that all bits of a given significance arrive simultaneously.

The equations below give the number of different cells required to construct a systolic array that will multiply two m-word vectors, where each word is n bits long and is represented in SBNR. The number of multiply-accumulate PE's (hexagons in Figure 12) is given by

$$PE = n^2 + (2n-1)(m-1); \tag{35}$$

the number of adders (diamonds) is

$$Add = 1/2 \left[n(n-1) + (n+m-1)(n+m-2) - 1 \right]; \tag{36}$$

and the number of shift registers (squares) is

$$Reg = (3n+m-4)^2 + (n+m-2). \tag{37}$$

The total multiplication time for the array is

$$T = (2n+m)t, \text{ where } t = \text{clock period}. \tag{38}$$

In pipelined systems, another important measure is the latency--the time from the start of the incoming data to the start of the outgoing. For the array this number is

$$L = (n+m)t. \tag{39}$$

Thus, the MSB's of the vectors are clocked in, and (n+m) cycles later, the answer MSB is clocked out. Every cycle after that, two more bits can be made available, or they can be buffered and streamed out. The latter method gives a total multiply time of (3n+m) cycles.

6.1.4.2 Processing Element Circuit

The three types of cells needed, their functions, and their I/O paths are shown in Figure 14. Figure 14a is the Ternary Multiply-Accumulate Cell (TMAC), which forms the core of the array. Part b is a PE which does not multiply any numbers, it simply adds the previous sum to zero and generates a new sum and carry. The shift register in Figure 3c is used to align carries with new rows, or to align the answer bits at the output.

The TMAC in Figure 14a multiplies the incoming A and B bits, and adds them to the incoming sum C. The output signals, carry and neg, are generated from AB and C for use in the PE to the upper right. The next C is a function of AB, C and the carry and neg signals from the lower-left PE. On the next clock cycle, the sum and two multiplicands are shifted to the next higher cells.

The ternary adder cell in Figure 14b adds the incoming C to zero, since there is no AB term. In SDNR, unlike binary, this addition can generate a carry. So the adder cell operation is the same as that of the TMAC, except that no multiplication is necessary. As a secondary function, if the adder PE is in the path of the B coefficients, it acts as a shift register for them.

The ternary latch cells (Figure 14c) act as one-cycle delays for whatever they are latching. Sometimes a latch cell will transmit a C value vertically. Sometimes it will take a carry from one column, and turn it into a sum, C, in the next column. The latches that are in the path of the B coefficients will also shift them.

Figure 15 summarizes the ternary CMOS logic gates developed by Mouftah [93], Huertas [100], and Balla [101] that were used in the design of the PE's. Also shown is an S-gate (switch gate) which is a pair of transmission gates that pass one of two inputs based on the control signal.

The circuit used to implement the TMAC, and its truth tables, are shown in Figure 16. The inputs are latched into the flip-flops at the bottom. From there, A and B are multiplied and the neg signal is generated. Using A, B and C, the TMAC generates the carry, and with the carry coming up from the next lower cell, it generates the sum. The total time, from the rising edge of the clock to the stable sum, is no longer than 17 transistor delays. This amounts to about 17 ns in 5 micron technology. During the low phase of the clock cycle, A and B are latched into the outputs. The sum is not latched because it will stay stable for much longer than the latches of the next cell require to shift it. The TMAC uses 212 transistors.

The circuit for the adder (not shown) is similar to the TMAC, except that the circuitry which multiplies A and B is gone, and all inputs with A, B, or AB are grounded. Some of the combinational logic is also simplified. The adder uses 98 transistors.

The latch circuit consists only of two flip-flops. One latches a value in on the positive clock level, and the other latches it out when the clock is negative. The latch circuit requires 32 transistors.

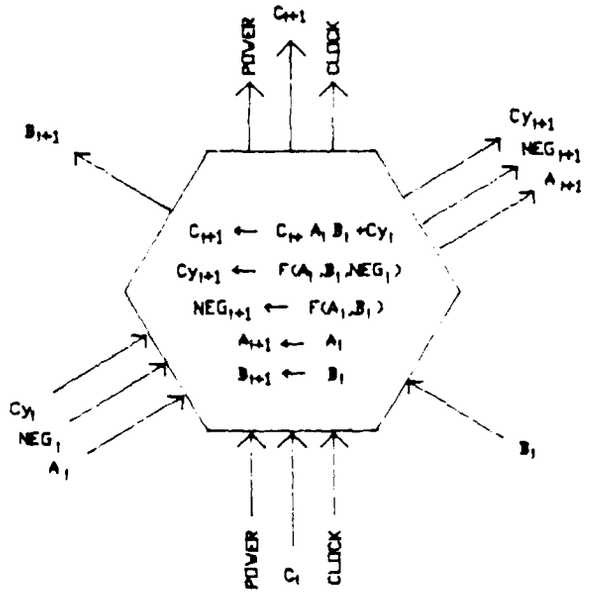


Figure 14a. Ternary Multiply-Accumulate Cell.

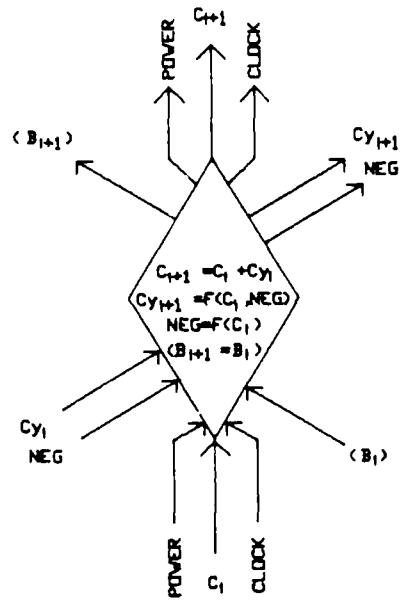


Figure 14b. Ternary Adder Cell.

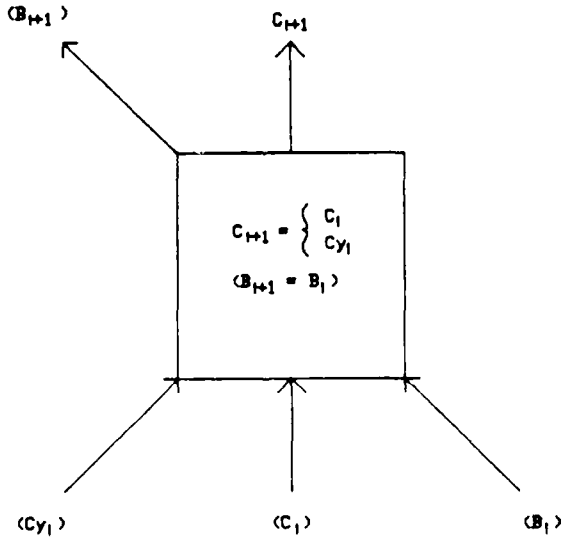
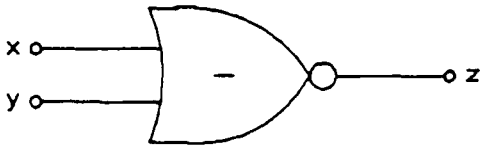
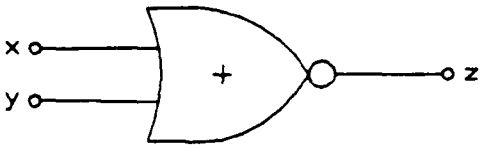


Figure 14c. Ternary Shift Cell.



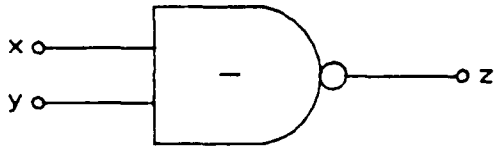
x	y	z
T	T	1
T	0	T
T	1	T
0	T	T
0	0	T
0	1	T
1	T	T
1	0	T
1	1	T

Figure 15a. Negative Ternary NOR.



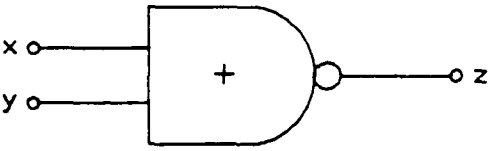
x	y	z
T	T	1
T	0	1
T	1	T
0	T	1
0	0	1
0	1	T
1	T	T
1	0	T
1	1	T

Figure 15b. Positive Ternary NOR.



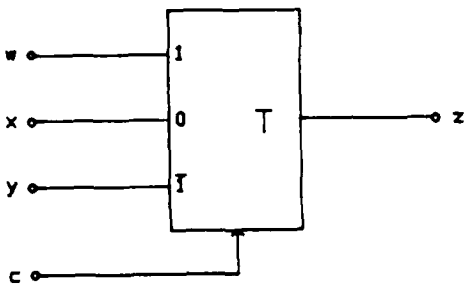
x	y	z
T	T	1
T	0	1
T	1	1
0	T	1
0	0	T
0	1	T
1	T	1
1	0	T
1	1	T

Figure 15c. Negative Ternary NAND.



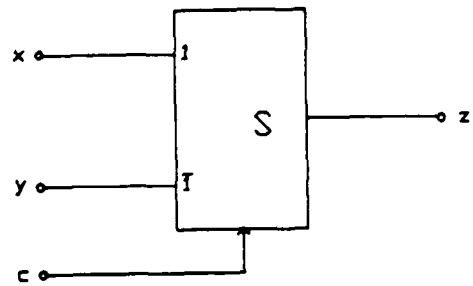
x	y	z
T	T	1
T	0	1
T	1	1
0	T	1
0	0	1
0	1	1
1	T	1
1	0	1
1	1	T

Figure 15d. Positive Ternary NAND.



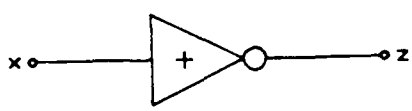
c	z
1	w
0	x
T	y

Figure 15e. T-Gate



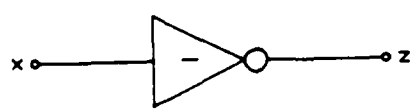
c	z
1	x
T	y

Figure 15f. S-Gate



x	z
T	1
0	1
1	T

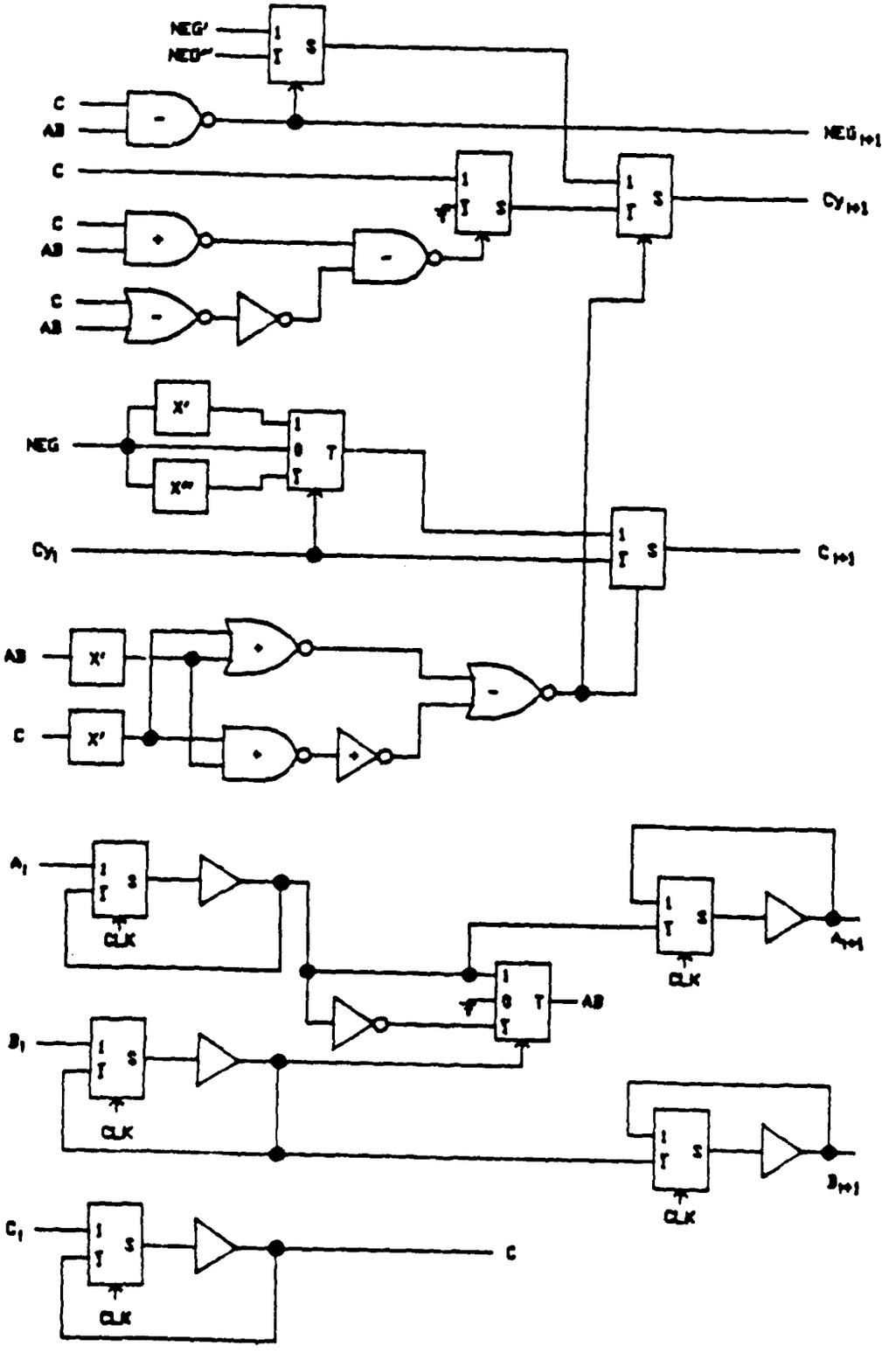
Figure 15g. Positive Ternary Inverter



x	z
T	1
0	T
1	T

Figure 15h. Negative Ternary Inverter

Figure 16. Ternary Multiply-Accumulate Cell



6.1.4.3 Observations

One can calculate the number of transistors needed to implement a specific array using the equations of Section 6.1.4.1. The total number is given by

$$\text{transistors} = (PE)(\text{trans. per PE}) + (Add)(\text{trans. per Add}) + (Reg)(\text{trans. per Reg}) \quad (40)$$

Table 3 gives the number of CMOS transistors in the array for different vector sizes and word lengths. Those readers who are familiar with current VLSI capability will recognize that, despite a highly regular structure and low interconnection area, most of the numbers given are larger than can be implemented on a single chip. Even among those that are small enough, though, the chip yield will be rather low.

Table 3. Area and Time Data for Systolic Array

Word Length n	Vector Length m	Sys. Array Number of Transistors	Time to MSB*	Total Time*	Sys. Array Area-Time Product	Multiplier Area-Time Product
8 bits	16	130,933	24	32	4.20×10^6	1.45×10^6
	32	275,205	40	48	13.2×10^6	2.90×10^6
	64	688,165	72	80	55.1×10^6	5.80×10^6
	128	2,011,749	136	144	$290. \times 10^6$	11.6×10^6
	256	6,649,573	264	272	1.81×10^9	23.2×10^6
	512	23,887,845	520	528	12.6×10^9	46.4×10^6
10 bits	16	172,597	26	36	6.21×10^6	
	32	339,717	42	52	17.7×10^6	
	64	798,373	74	84	67.1×10^6	
	128	2,213,349	138	148	$328. \times 10^6$	
	256	7,033,957	266	276	1.94×10^9	
	512	24,637,797	532	542	13.4×10^9	
12 bits	16	219,045	28	40	8.76×10^6	
	32	409,013	44	56	22.9×10^6	
	64	913,365	76	88	80.4×10^6	
	128	2,419,733	140	152	$368. \times 10^6$	
	256	7,423,125	268	280	2.08×10^9	
	512	25,392,533	534	546	13.9×10^9	
14 bits	16	270,277	30	44	11.9×10^6	
	32	483,093	46	60	29.0×10^6	
	64	1,033,141	78	92	95.1×10^6	
	128	2,630,901	142	156	$410. \times 10^6$	
	256	7,817,077	270	284	2.22×10^9	
	512	26,152,053	536	550	14.4×10^9	
16 bits	16	326,293	32	48	15.7×10^6	10.0×10^6
	32	561,957	48	64	36.0×10^6	20.0×10^6
	64	1,157,701	80	96	$111. \times 10^6$	40.0×10^6
	128	2,846,853	144	160	$455. \times 10^6$	80.0×10^6
	256	8,215,813	272	288	2.37×10^9	$160. \times 10^6$
	512	26,916,357	538	554	14.9×10^9	$320. \times 10^6$

*in clock cycles.

Unfortunately, this extra space does not buy a faster processor. The time for a 16-bit multiply of length-16 vectors is 48 clock cycles. That amounts to about 816 ns. The time for the same multiply implemented with 16 parallelled 16-bit multipliers [95] is 120 ns. That multiplier setup requires only 350,000 transistors, a size increase of 10 percent over the array. Ten percent is a small price in area for a seven-fold speedup.

To compare area and time trade-offs, one uses the area-time product. The second column from the right in Table 3 contains the area-time products for the systolic array. The last column represents the comparative numbers from the Takagi multiplier. Note that at the closest, these numbers are separated by a factor of 1.5, and that that distance increases with vector length.

This relationship is easily seen from the complexities of the two architectures. The area of the systolic array is $O(n^2+m^2)$, while the area of the parallel multiplier is $O(mn^2\log_2 n)$. From this, one can see that the area of the multiplier increases faster with n than the area of the systolic array. Unfortunately, the array starts out so much bigger, and the multiplier cannot easily catch up. The time complexity of the array is $O(n+m)$. Since the PE's in the parallel multiplier operate simultaneously, the multiply time is independent of m, and furthner, the multiply time is not strongly dependent on n, being only $O(\log_2 n)$.

The multiplier actually has a higher-order area-time complexity, $O(n^2m\log^2 n)$, than the systolic array, $O(n^3+m^3)$. However, the basic cell in the systolic array is more complex than its counterpart in the multiplier. Since the array is clocked, each cell must contain a latch for each data bit going in. In addition, many of the ternary logic gates in the array use more transistors than the binary gates in the multiplier, which adds even more area. The result of these factors is that the area-time product remains lower for the multiplier than for the array as long as the word length is less than 97 bits.

Figure 17 compares the area-time products for the systolic array and the parallel multiplier for $n \leq 128$ and $m \leq n$. The objective of a lower area-time product for the systolic array is achieved, but only over a limited range. The curves show that when $m=0.5n$, and $n>97$, the area-time product of the systolic array is lower than that of the parallel multiplier. One can also see how the stronger dependence on n is causing a sharper upward slope in the multiplier curves, relative to those of the array. In fact, for $n \geq 128$, the area-time product of the array is lower than that of the multiplier for values of m as high as 0.75n. The other two curves will also catch up if n is made larger.

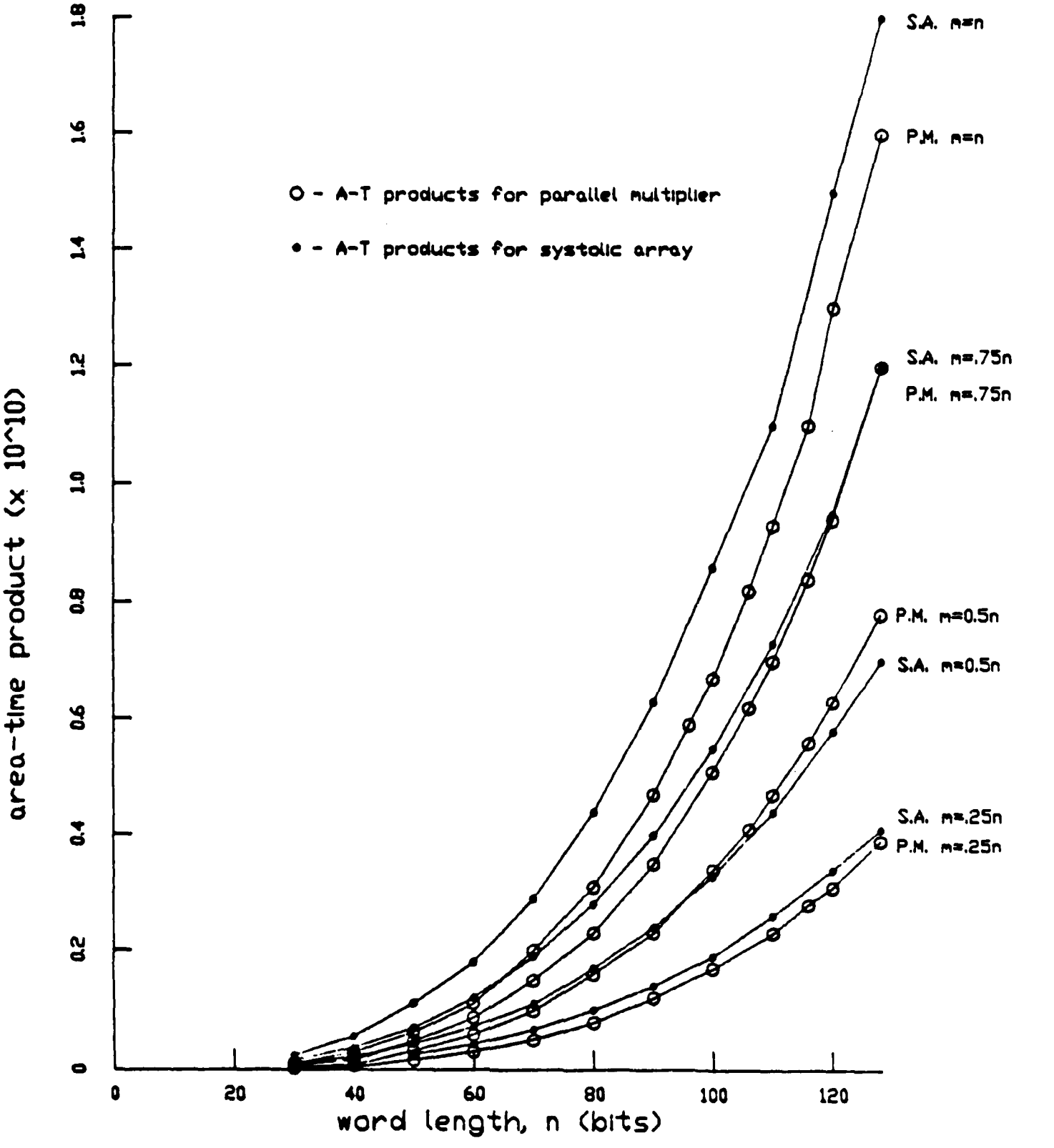
Unfortunately, word lengths of 100 bits or more are very rare. This fact limits the applicability of the array architecture described to extremely high precision operations. To make the systolic array competitive for shorter words, its hardware must be simplified. A two-fold reduction in the number of transistors in each cell would halve the area-time product at every point, placing the systolic array significantly lower than the multiplier on the area-time graph. This reduction would allow the systolic array to outperform the parallel multiplier on word lengths as low as 26 bits. For maximum benefit, the vector length should be kept to around half the word length.

6.1.4.4 Recommendations For Future Research

Optimization of the PE circuit is essential for implementation as a vector multiplier. As was shown in Section 6.1.4.3, a two-fold reduction in PE hardware would make the systolic array useful for much smaller word lengths (such as 32 bits). For that reason, future research should concentrate on characterizing various cell configurations such as signed binary logic, ECL, I^2L and CCD gates. The design in another technology might yield significant savings in hardware.

The results suggest that this architecture might be better suited to word-level multiplication of matrices, using constant precision operands and results. Such a setup would eliminate the extensive hardware otherwise needed

Figure 17. Area-Time Products for Systolic Array and Multiplier



to handle carries out of the multiplication core. Data transmission between the cells could be pipelined or parallel. If it is deemed necessary to obtain a higher-precision product, either more time could be allowed for the serial transmission of the extra bits, or additional lines could be added between cells for parallel communication.

To augment the streamlining of the systolic array, research should continue in alternative architectures, both pipeline and parallel. The objective is a vector multiplier that can operate on long vectors (>32 words) composed of short words (8 - 16 bits). The best multiplier would be able to handle variable-length operands without any serious slowing. Included in this investigation should be other forms of systolic arrays, and combinations of parallel and pipeline structures.

6.2 Fault-Tolerant Architectures

6.2.1 Residue Number Systems (RNS)

A major competitive number system offering high reliability modularity and thus capable of fault-tolerance is the Residue Number System (RNS). As summarized by Jenkins [102], an RNS is defined by a set of moduli, $M = \{m_1,\ldots,m^L\}$ which are pairwise relatively prime integers (i.e., no pair from the set contains a common non-unity factor). Natural numbers in the range $R = [0,M-1]$, with $M = m_1 m_2 \ldots m_L$, are encoded by L residue digits $x_1 x_2 \ldots x_L$, where $x_i = (X) \bmod m_i$, $i = 1,\ldots,L$, where X in R. Residue arithmetic is defined by

$$(x_1 x_2 \ldots x_L) * (y_1 y_2 \ldots y_L) = (z_1 z_2 \ldots z_L) \qquad (4')$$

the $z_i = (x_i * y_i) \bmod m_i$, where * is one of addition, subtraction, or multiplication. Note that RNS arithmetic has a natural modular structure that leads to modularity and parallelism in the hardware.

The lack of communication among digits in residue arithmetic suggests that if an error occurs in one digit it cannot be propagated into other digit positions during subsequent operations involving addition, subtraction, or multiplication. This property provides a basis for fault-tolerance that is inherent in the basic algebraic structure, and which can be used to obtain fault-tolerant hardware architectures. During some of the more difficult RNS operations such as scaling, division, or magnitude comparison, there is interaction between residue digits and this error isolation property is not preserved. Therefore, the fault-tolerant properties of RNS arithmetic are particularly useful for certain types of signal processing applications where most of the computation consists of addition, subtraction, and multiplication. Two-dimensional digital filtering used in image enhancement and feature extraction is an example of a computation intensive operation that is ideally suited for RNS techniques.

The nonweighted structure of the RNS code is another basic property that makes residue arithmetic useful in the design of fault-tolerant hardware structures. If a particular residue digit is consistently erroneous, the corresponding faulty module can be identified by RNS error checking techniques and disconnected without affecting the other modules. If the original RNS contains enough dynamic range, the reduced processor can continue functioning with a reduced dynamic range. This concept is called soft failure because the processor does not catastrophically fail when a hardware failure occurs, but rather the faulty module is disabled, and the remaining modules continue functioning in a useful although restricted manner. If desirable, error correction can be used to replace the function of the faulty module provided enough redundancy is designed into the code.

6.2.1.1 Residue Number Implementations

Applications of RNS theory to general purpose computers, as well as the use of redundant residue digits to provide error detection/correction in RNS structures, has been researched for a number of years. More recently, advances in VLSI circuit technology have renewed interest in RNS applications to Digital Signal Processing (DSP). Although some problems still exist with

magnitude comparison, division, scaling, and related operations, technological improvements have provided economical ways to work around these shortcomings. Paul [103] recently examined such implementations and has shown remarkable fault-tolerant capabilities for the realization of high-performance DSP systems. He showed how system reliability can be enhanced by a parallel processor structure partitioning word length among processors (just as we propose to do for SBNR). His implementations of Redundant Residue Number Systems (RRNS) enhanced identification of faulty processors and modular system degradation. As we also propose, he showed the efficacy of short word length of the residues. A clue to his gracefully degradable design is the incorporation of pipelined memory accesses optimized for speed. SBNR realization handily captures the same pipelined enhancements. Even more so, since SBNR has elements in (-1,0,1) instead of several primed modules as found in RNS. We believe that the use of redundancy residue codes for fault-tolerance capability already examined by [104-107] carry over directly to SBNR realizations.

6.2.2 Graceful Degradation [108]

Mapping algorithms onto processor arrays has been widely investigated todate [67,109-113]. Some general observations can be made form these efforts. Even though the dynamic reallocation of data/instructions is complicated and relatively slow, dynamic array configuration is just as injurious in the graceful degradation issue. Even an array with complete reconfigurability is difficult. In our architectural studies proposed herein, the large array configurations exacerbate these issues. Hence, greater concern for graceful degradation issues are necessary.

There exist alternative solutions, device redundancy being one of them. Another is to attempt to map smaller algorithms onto the same configuration size, assuming that spare processors are freed up for fault-tolerant purposes. [114] provides a critical assessment. As Fortes notes, when we examine classic architectures such as the MPP, ILLIAC [115], CHiP, Diogenes arrays, NCR 45CG72, PC Systolic machines, turbo boards, and hardware accelerators, a consensus draws the conclusion that, unless a functioning replica of the original array is up, graceful degradation is impossible. Other solutions to graceful degradation encompass algorithm rescheduling methods [116] and classic error detection correction schemes [52]. Also, [117] reports on a clever connectivity preservation scheme for VLSI multiprocessor systems.

6.3 Wafer Scale Integration

Packaged integrated device reliability is improved with less pin out [118]. Input/output pads are susceptible to electrostatic discharge, especially on MOS circuits. Also, relative I/O pad area in small-scale IC's is high. Driver area is high and they are power hogs. Finally, I/O pins are mechanical failure prone.

Pin estimates for each package is provided by Rent's Rule. Rent's Rule applies especially to small sub-modules embedded in larger systems. When the package contains a major subsystem or the entire system, Rent's Rule is overbiased. Consequently, WSI is particularly attractive for a complete processor integration.

Integrating these chips on one wafer is space efficient because of inter-package connections. Shorter lines have less self-capacitance. This reduces the size and power consumption of line drivers. Since connections dominate most circuit layouts, WSI can substantially speed systems, reduce total internal power requirements and improve density. Nevertheless, caution is advised because the interconnections have not disappeared altogether.

Board-level system designers address signal quality, noise, and power-distribution problems. In the dense layout found in VLSI or WSI they become just as acute at even lower frequencies. The increase in interconnection density also produces interline coupling problems caused by mutual capacitance. Interline coupling can arise between adjacent lines on a given layer as well as between lines on overlapping layers (most dangerous on long runs of parallel lines). The decreased separation between lines in WSI, compared with that in other packaging arrangements, plagues designers. Hence, long parallel runs are to be avoided in layout. To fabricate a WSI system the size of a full wafer demands either a repair capability, a tolerance for failure, or a combination of both. (Consider the HP RISC chip set.) Tolerance for failure implies redundancy in some form, while repair capability implies intervention in the fabrication process. Yet, the wafer-yield enhancement resulting from this redundancy or intervention is still the most attractive benefit of WSI.

6.4 Reliable MVL Systolic Arrays

Some very complex algorithms have already been implemented on systolic arrays by Kung, Leiserson, and Andrews. Only recently have MVL systolic array implementations been studied. Andrews has established the efficacy of MVL arrays for the least-mean-square algorithm which is much simpler than the proposed applications to be studied herein. However, Moraga [119] has demonstrated MVL array effectiveness for Christenson transforms (a Walsh transform is a subset) computations. He shows how a Christenson spectrum of n-ary, n-place, p-valued functions are configured in a MVL systolic system. (Note, these are complex valued functions.)

His VLSI PE's behave as an MIMD machine, unlike all other array studies which are SIMD. This useful study provides us with important clues to develop our algorithms and some very important preliminary results as discussed next. Moraga conveniently provides us with an algorithm to generate a complete test set for detecting stuck-type MVL faults.

In a binary solution, 32 bit ALU's generate 32 bit additions and complex multiplications, producing 32 bit truncated results. 32 bit data and intermediate results would have to be stored/transferred between cells (hence, at least 32 wires are needed for cell interconnections). In an MVL design, Moraga [119] shows that we can use ' digit arguments and the most complex operation is a ' digit subtraction mod p. Exact results for the spectral elements are obtained as p . n digit words. Hence, even for a 6-place, 5-valued (n-ary) function, we require less than 32 digits. One-out-of-p coding and parallel updating of counters is then realized in a shorter time than that required for the multiplication of two 16 x 16 bit (complex) numbers. Two important advantages are now evident with MVL in systolic arrays. First, far less interconnections between cells occur. Second, simpler and faster circuits can be used in the PE. Note, the '-out-of-p coding and parallel

update counter scheme is another realization of distributed arithmetic.) We can incorporate these same promising contributions within our SBNR PE's. Reliable MVL circuits can then be analyzed using the M-difference calculus proposed by Lu and Lee [120] for fault-detection of single and multiple stuck-at faults in MVL, based on earlier work [121,122].

6.5 Ternary Logic

Three-valued or ternary logic may have an edge on binary logic [123]. The information per wire ratio is higher; the complexity of interconnections can be reduced; chip area reductions appear likely; and efficient error-detection and error-correction codes can be employed. Serial arithmetic operations are faster. As such, these advantages encouraged study. [123-129] offer several realizations. Given a dynamic range, the ternary circuit complexity [101] is comparable to that of corresponding binary circuits. Nevertheless, the associated reduction in the word length tends to ameliorate the pin-limitation problems.

A new family of ternary logic circuits based on both depletion and enhancement types of complementary MOS transistors (DECMOS) has been shown to be useful in the design of ternary digital systems. With the use of voltage power supplies below the transistors threshold voltage and the exclusion of resistors, it is possible to implement this circuitry in VLSI. Also this offers a low power consumption, high speed and comparable performance to the binary counterpart circuitry. New ternary logic circuits based on the use of Depletion/Enhancement Complementary Metal-Oxide-Semiconductor (DECMOS) Integrated Circuits has been demonstrated. The circuits use two power supplies each below the transistors threshold voltage and do not include any resistor. The circuit design of basic ternary operators (inverters, NAND, NOR) and an example on the use of these basic ternary operators as building blocks in the design of a ternary full adder is now available [75]. In [75] the Simple Ternary Inverter (STI), the Positive Ternary Inverter (PTI) and the Negative Ternary Inverter (NTI) are three possible ternary operators.

6.6 Taxonomy of Fault-Tolerant Schemes [78]

6.6.1 Fault-Tolerant Nodes

In this scheme, spare PE's are placed at each array node. The spare PE's may be arranged in a number of different ways. Figure 18 illustrates the case where three PE's and a checker are placed at each node.

6.6.2 Temporal Redundancy

In many systolic arrays, the PE's are idle for a large percentage of the time. The basis of the temporal redundancy scheme is to replace a faulty PE by an idle neighbor, on a cyclic basis [130].

Alternatively, this idle time can be created by setting up each PE with two separate inputs and forcing the PE to devote half of its cycles to each of these inputs [131]. Appropriate steering circuits must also be provided as illustrated in Figure 19.

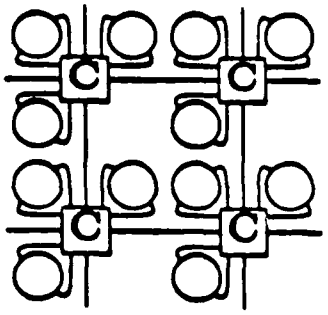


Figure 18. Node Redundancy Scheme [78]

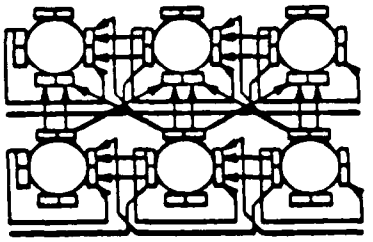


Figure 19. Temporal Redundancy Scheme [131]

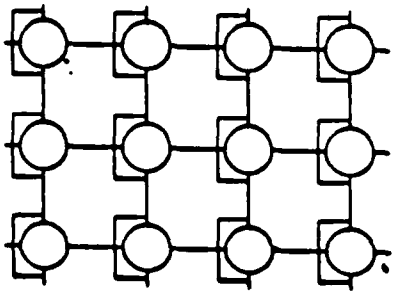


Figure 20. Row Bypass Scheme [133]

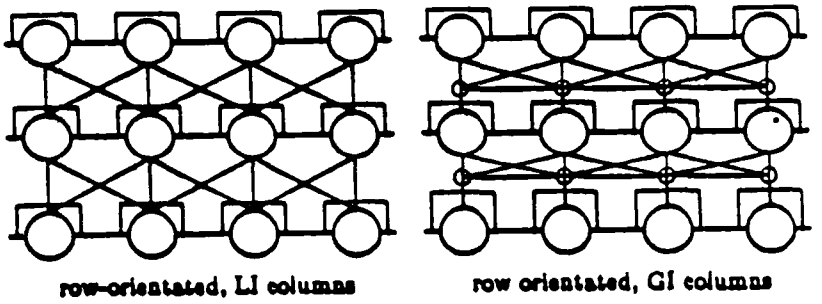


Figure 21. Row-Oriented Schemes [137]

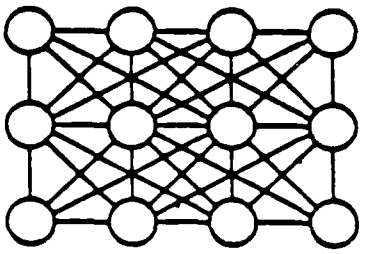


Figure 22. Two-D Perturbation Scheme with CE and LI switches [138]

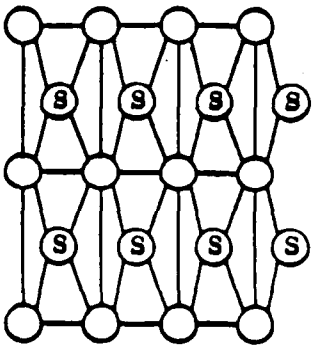


Figure 23. Interstitial Redundancy Scheme [139]

6.6.3 Interconnection Reconfiguration

With interconnection reconfiguration, fault-tolerance is provided by rerouting inter-PE connections, thereby bypassing faulty cells. The programmable interconnect required can be provided by a wire joining and fusing process or by using active switches.

The fuse and join technologies require extra processing steps for their implementation. However, area overhead is reduced. Furthermore, since no switches are involved, propagation delays may be smaller. On the other hand, using active switches to provide the reconfigurable routing does not involve any special processing steps. It also offers the potential for reliability enhancement which is absent in other technologies.

6.6.4 Switch Types

Three types of switches can be used in these schemes [132]:

1. Using the PE as a Connecting Element (CE) only, thereby bypassing its normal function.

2. Multiplexing PE I/O ports so that PE's can communicate with a fixed choice of near-neighbor PE's (LI = Local Interconnections).

3. Using external switches that can be as flexible as desired (GI = General Interconnection).

With these three switch types the area overhead, programming difficulty and reconfiguration flexibility increase from the CE to the GI switch.

6.6.5 Row Bypass

In the row bypass scheme [133], a complete row of PE's is bypassed, using GI switches. An example of such a scheme is illustrated in Figure 20. This scheme is suitable to situations with high PE yield, such as bit-serial arrays.

6.6.6 Row Oriented

In the row oriented schemes, columns are organized through the rows, taking one and only one column element from each and every row. The unused elements in each row are then bypassed using GI switches. Schemes where columns are organized through LI switches are described in [134,135,136]. Each of these papers describes simple circuits which enable the reconfiguration around faulty PE's to be carried out completely internally and automatically.

Schemes where the columns are organized through GI switches can be found in [137], the simplest of which is illustrated in Figure 21. Because GI switches are used, no simple internal reconfiguration scheme has been found. The switches must be programmed externally.

6.6.7 2D Perturbation

In this scheme, the PE's are "perturbed", in both directions, but by only a small number of PE sites. A scheme described in [138] using CE and LI switches only is illustrated in Figure 22.

6.6.8 Interstitial Redundancy

In an interstitial scheme, as illustrated in Figure 23, spare PE's are provided between node sites [139]. The spare PE's are switched into the array as required, using LI type switches. With similar complexity to the row LI column schemes, this scheme performs slightly better than those when the PE yields are between 65% and 80%.

6.6.9 Hierarchical Scheme

In [140], a GI switched scheme is described. Here PE's are organized into blocks of 12 of which only 4 are required. If 4 good PE's could not be organized from the block then the whole column of blocks would be bypassed. For PE yields between 40% and 60% this scheme performs best out of all the schemes presented. In fact, Hedlund intended the scheme to be used for an array where 33% of the PE's were faulty on the average.

On comparison of the schemes, temporal redundancy appears to be efficient only in situations where PE idle time already exists. Interstitial redundancy has application for certain yield classes. For the reconfiguration schemes, the more complex the switch type, the greater the flexibility afforded but also the greater the area overhead. In fact, the more complex and flexible schemes only become useful for lower yields.

REFERENCES

[1] Avizienis, A., "Signed-Digit Number(s) Representations for Fast Parallel Arithmetic," IRE Transactions on Electronic Computers, Vol. EC-10, pp. 389-400, Sept. 1961.
[2] Hurst, S.L., "Multiple-Valued Logic, Its Status and Its Future," IEEE Trans. on Computers, Vol. C-33, pp. 1160-1179, 1984.
[3] Trimburger, S., "Automating Chip Layout," IEEE Spectrum, pp. 28-35, 1982.
[4] Moraga, C., "Systolic Arrays and MVL," Abteilung Informatik, University of Dortmund, Germany, Technical Report, 1984.
[5] Thompson, C.D., "A Complexity Theory for VLSI," Ph.D. Dissertation, Carnegie-Mellon University, C.S. Dept., Pittsburg, PA, 1980.
[6] Brent, R.P. and H.T. Kung, "The Area-Time Complexity of Binary Multiplication," Tech. Report CMU-CS-79-136, 1979.
[7] Mead, C. and L. Conway, "Introduction to VLSI Systems", Reading, Mass., Addison Wesley, 1980.
[8] Atkins, D.E., "Design of the Arithmetic Units of ILLIAC III: Use of Redundancy and Higher Radix Methods," IEEE Trans. on Computers, Vol. C-19, pp. 720-733, 1970.
[9] Tung, C., "Signed-Digit Division Using Combinational Arithmetic Nets," IEEE Trans. on Computers, Vol C-19, pp. 746-749, 1970.
[10] Ercegovac, M.D., "A General Hardware-Oriented Method for Evaluation of Functions and Computations in a Digital Computer," IEEE Trans. on Computers, Vol. C-26, pp. 667-680, 1977.
[11] Robertson, J.E., "A New Class of Digital Division Methods," IRE Trans. Electron. Computers, Vol EC-7, pp. 218-222, 1958.
[12] Ronatsch, F.A., "A Study of Transformations Applicable to the Development of Limited Carry-Borrow Propagation Adders," Dept. of Computer Science, Univ. of Illinois, Urbana, Rept. 226, 1967.
[13] Avizienis, A., "On a Flexible Implementation of Digital Computer Arithmetic," Information Processing, C.M. Popplewell Editor, Amsterdam, North Holland, 1963.
[14] _____, "Binary Compatible Signed-Digit Arithmetic," 1964 Fall Joint Computer Conference, AFIPS Proceedings, Vol 26, 1964, pp. 663-672, 1964.
[15] _____, and Tung, C., "A Universal Arithmetic Building Element (ABE) and Design Methods for Arithmetic Processors." IEEE Trans. on Computers, Vol. C-19, pp. 733-745, 1970.
[16] Takagi, N., H. Yasuura, and S. Yakima, "A VLSI-Oriented High-Speed Multiplier Using a Redundant Binary Addition Tree," Systems-Computers-Controls, Vol. 14, pp. 19-28. 1983.
[17] Chow, C.Y.F., "A Variable Precision Processor Module." Ph.D. Dissertation, University of Illinois, Computer Science Dept., Urbana, 1980.
[18] Robertson, J.E., "Design of the Combinational Logic for Radix 16 Digit Slice for a Variable Precision Processor Module," Proc. 1983 IEEE International Conference on Computer Design, Port Chester, N.Y., 1983.
[19] Caraiscos, C., and B. Liu, "A Round-Off Error Analysis of the LMS Adaptive Algorithm," ICASSP 83, pp. 29-32, 1983.
[20] Andrews, M., "Comparative Implementations of the LMS Algorithm," Int. Journal of Computers and Electrical Engineering, May 1986.
[21] Morgul, A., P.M. Grant, F.F.N. Cowan, "Wide-Band Hybrid Analog/Digital Frequency Domain Adaptive Filter," IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-32, 1984, pp. 762-769.

[22] Mactaggart, I.R., and N.A. Jack, "A Single Chip Radix-2 FFT Butterfly Architecture Using Parallel Data Distributed Arithmetic," IEEE Journal of Solid-State Circuits, Vol. SC-19, 1984, pp. 368-373.
[23] McCanny, J.V., L.W. Wood, J.G. McWhirter, and C.J. Oliver, "The Relationship Between Word and Bit Level Systolic Arrays as Applied to Matrix X Matrix Multiplication," Real Time Signal Processing VI, Proceedings of SPIE 495, pp. 114-130, 1983.
[24] Evans, R.A., D. Wood, K. Wood, J.V. McCanny, J.G. McWhirter, and A.P.H. McCabe, "A CMOS Implementation of a Systolic Multibit Convolver Chip," VLSI 83, Norway, 1983.
[25] Corry, A. and K. Patel, "Architecture of a CMOS Correlator," IEEE Int. Symp. on Circuits and Systems, 1983.
[26] McCanny, J.V. and J.G. McWhirter, "A Bit Level Systolic Array for Matrix x Vector Multiplication," IEEE Part G, Electronic Circuits and Systems.
[27] Hamacher, V.C. and Z.G. Vranesic, Multivalued Logic: Theory and Applications, D.C. Rine, Editor, Amsterdam, The Netherlands, North-Holland, 1977.
[28] Sips, H.J., "Bit-Sequential Arithmetic for Parallel Processors," IEEE Trans. on Computers, Vol. C-33, pp. 7-20, 1984.
[29] Leighton, T. and E.E. Leiserson, "Wafer-Scale Integration of Systolic Arrays," IEEE Trans. on Computers, Vol. C-34, pp. 448-461, 1985.
[30] Peled, A. and B. Liu, "A New Hardware Realization Of Digital Filters," IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-2, No. 6, pp. 456-462, December, 1974.
[31] Intel Corporation, Reliability Reps., RR-6, 9, 11, 12, 14, 17, 18, 1975-1979.
[32] Iyer, R.K., "Reliability Evaluation of Fault-Tolerant Systems-Effect of Variability In Failure Rates," IEEE Trans. on Computers, Vol. C-33, No. 2, pp. 197-199, Feb. 1984.
[33] Cox, D.R. and P.A.W. Lewis, The Statistical Analysis of a Series of Events, London: Methuen, 1966.
[34] Rosenberg, A.L., "A Hypergraph Model for Fault-Tolerant VLSI Processor Arrays," IEEE Trans. on Computers, Vol. C-34, pp. 578-584, June 1985.
[35] Rogers, C.E., "A Critical Survey of Four Techniques for the Construction of Fault Tolerant Multiprocessor Arrays," Proc. 21st Southeast Region ACM Conf., pp. 186-206, 1982.
[36] Rosenberg, A.L., "On Designing Fault-Tolerant VLSI Processor Arrays," in Advances in Computing Research 2, CT: Jai Press, pp. 181-204.
[37] Bhatt, S.N. and C.E. Leiserson, "How to Assemble Tree Machines," in Advances in Computing Research 2, CT: Jai Press, pp. 95-104.
[38] Chung, F.R.K., F.T. Leighton, and A.L. Rosenberg, "DIOGENES, A Methodology for Designing Fault-Tolerant Processor Arrays," Proc. 13th Int. Conf. on Fault-Tolerant Comput., pp. 26-32, 1983.
[39] _____, "Embedded Graphs in Books: A Layout Problem with Applications to VLSI Design," to be published.
[40] Greene, J.W. and A. El Gamal, "Area and Delay Penalties in Restructurable Wafer-Scale Arrays," Proc. 3rd Caltech Conf. VLSI, 1983.
[41] Hayes, J.P., "A Graph Model for Fault-Tolerant Computing Systems," IEEE Trans. Computers, Vol. C-25, pp. 875-883, 1976.
[42] Rosenberg, A.L., "Routing with Permuters: Toward Reconfigurable and Fault-Tolerant Networks," Dept. Computer Science, Duke Univ., NC, Tech. Rep. CS-1981-13, 1981.
[43] Wong, W.W. and C.K. Wong, "Minimum k-Hamiltonian Graphs," J. Graph Theory, Vol. 8, pp. 155-165, 1984.

[44] Von Neuman, J., "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," Automata Studies, No. 34, Princeton University Press, 1956, pp. 43-99.
[45] Tryon, J.G., Quadded Logic in Redundancy Techniques for Computing Systems, 1962, pp. 205-228.
[46] Reynolds, D.A. and G. Metze, "Fault Detection Capabilities of Alternating Logic," IEEE Transactions on Computers, Vol. C-27, 1978, pp. 1093-1098.
[47] Patel, J.H. and L.Y. Fung, "Concurrent Error Detection in ALUs by Recomputing with Shifted Operands," IEEE Transactions on Computers, Vol. C-31, 1982, pp. 589-595.
[48] Kuang, K.H. and J.A. Abraham, "Algorithm-Based Fault-Tolerance for Matrix Operations," IEEE Transactions on Computers, Vol. C-33, 1984, pp. 518-528.
[49] Avizienis, A., "Arithmetic Error Codes: Cost and Effectiveness Studies for Application in Digital System Design," IEEE Transactions on Computers, Vol. C-20, 1971, pp. 1322-1331.
[50] _____, "Low-Cost Residue and Inverse Residue Error-Detecting Codes for Signed-Digit Arithmetic," Proceedings--5th IEEE Symposium on Computer Architecture, 1981, pp. 165-168.
[51] _____, "Arithmetic Algorithms for Operands Encoded in Two-Dimensional Low-Cost Error Codes," Proceedings--7th IEEE Symposium on Computer Arithmetic, 1985, pp. 285-292.
[52] Kuang, K.H. and J.A. Abraham, "Low Cost Schemes for Fault-Tolerance in Matrix Operations with Processor Arrays," Proceedings--12th Int. Symp. on Fault Tolerant Computing, 1982.
[53] Chen, C.Y. and J.A. Abraham, "Fault-Tolerant Systems for the Computation of Eigenvalues and Singular Values," Proceedings--SPIE 30th Annual International Technical Symposium on Optical and Optoelectronic Applied Sciences and Engineering, 1986.
[54] Jou, J.Y. and J.A. Abraham, "Fault-Tolerant Matrix Operations on Multiple Processor Systems Using Weighted Checksums," Real Time Signal Processing VII, Proceedings SPIE 495, 1984, pp. 94-101.
[55] Andrews, M., Principles of Firmware Engineering in Microprogram Control, Computer Science Press, Potomac, Maryland, 1980.
[56] Lamagna, E.A., "Fast Computer Algebra," Computer, pp. 43-56, September, 1982.
[57] Roberts, R., "On Computing Hardware Implementation of Fixed Point Digital Filters," IEEE Circuits and Systems, Vol. 3, June, 1981.
[58] Burrus, C.S., "Digital Filter Structures Described by Distributed Arithmetic," IEEE Trans. on Circuits and Systems, Vol. CAS-24, No. 12, pp. 674-680, 1977.
[59] Baudet, G.M., et. al., "Area-Time Optimal VLSI Circuits for Convolution," INRIA, France, No. 30, pp. 1-16, 1980.
[60] Porter, W.A., "Error Tolerant Design of Multilevel Logic Functions," IEEE Trans. on Computers, Vol. C-31, pp. 551-554, 1982.
[61] Roth, J.P., "Diagnosis of Automata Failures: A Calculus and a Method," IBM J. Res. Develop., Vol. 10, pp. 278-291, 1966.
[62] Goel, P., "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," IEEE Trans. Computers, Vol. C-30, pp. 215-222, 1981.
[63] Breuer, M.A. and A.D. Friedman, "Diagnosis and Reliable Design of Digital Systems", Woodland Hills, CA: Computer Science Press, 1976.

[64] Fujiwara, H. and S. Toida, "The Complexity of Fault Detection Problems for Combinational Logic Circuits," IEEE Trans. Computers, Vol. C.31, pp. 555-559, 1982.
[65] Ketchen, Mark B., "Point Defect Yield Model for Wafer Scale Integration," IEEE, p. 24, 1985.
[66] Koren, I. and M.A. Breuer, "On Area and Yield Considerations for Fault Tolerant VLSI Processor Arrays," Dept. Elec. Eng. - Syst., Univ. Southern California, Los Angeles, Tech. Rep. DISC/82-5, 1982.
[67] Fortes, J.A.B. and C.S. Raghavenda, "Dynamically Reconfigurable Fault-Tolerant Array Processors," Proc. Parallel Processing Conference, pp. 958-960, 1983.
[68] Greene, J.W. and A. El Gamel, "Configurations of VLSI Arrays in the Presence of Defects," J. ACM, Vol. 31, No. 4, pp. 694-717, Oct. 1984.
[69] Meyer, J.F., "On Evaluation the Performability of Degradable Computer Systems," IEEE Trans. on Computers, Vol. C-29, No. 8, pp. 720-731, August 1980.
[70] Li, R.Y., "A Methodology for Evaluating Computer Noise Failure Rate," Proc. Asilomar Conference on Circuits, Systems and Computers, pp. 463-465, October 1984.
[71] Cerny, C. and E.M. Aboulnamid, "Built-In-Testing of pi Testable Iterative Arrays," Proc. Int. Symp. on Fault-Tolerant Computing, pp. 33-36, June 1983.
[72] Jaggernauth, H., A.C.P. Loui, and A.N. Venetsanopolous, "Real-Time Image Processing by Distributed Arithmetic Implementation of Two-Dimensional Digital Filters," IEEE Acoustics, Speech, and Signal Processing, No. 6, December 1985.
[73] Wood, D., R.A. Evans, and K.W. Wood, "An 8-Bit Serial Convolver Chip Based on Bit Level Systolic Array," Proc. Custom Integrated Circuit Conference, pp. 256-261, May 1983.
[74] Hu, M. and K.C. Smith, "Ternary Scan Design for VLSI Testability," IEEE Trans. on Computing, Vol. C-35, No. 2, pp. 167-170, February 1986.
[75] Heung, A. and H.T. Mouftah, "DECMOS A Low Power Family of Three Valued Logic Circuits for VLSI Implementation," Proc. 14th Int. Symp. on Multiple-Valued Logic, Vol. 14, 1984, pp. 120-124.
[76] Courtois, B., "Failure Mechanism, Fault Hypotheses and Analytical Testing of LSI-NMOS (HMOS) Circuits," VLSI 81, J.P. Gray, Ed., New York: Academic Press, 1981, pp. 341-350.
[77] Galiay, J., Y. Crouzet, and M. Vergniault, "Physical Versus Logical Fault Models MOS LSI Circuits: Impact on Their Testability," IEEE Trans. on Computers, Vol. C-29, pp. 527-531, June 1980.
[78] Franzon, P., "Interconnect Strategies for Fault-Tolerant 2D VLSI Arrays," ICCD, October 1986.
[79] Andrews, M., "Real-Time Adaptive Filters, A Least-Mean-Square Approach," Computers, Electronics, and Control Symposium, Canada, 1974.
[80] _____, "Real-Time Adaptive Filters II," 6th Nonlinear and Adaptive Estimation Symposium, Arizona, 1974.
[81] _____, and R. Fiten, "FWLA Noise Effects on the LMS Adaptive Weights," IEEE Conf. on Acoustics, Speech, and Signal Processing, Connecticut, 1977.
[82] _____, "The ADF, An Experimental Self-Adaptive Digital Filter," Computers and Elect. Eng., Vol. 8, pp. 237-244, 1981.
[83] _____, "A Systolic SBNR Adaptive Signal Processor," IEEE Transactions on Circuits and Systems, Vol. CAS-33, No. 2, Feb. 1986.
[84] Ramponi, G. and G.L. Sicuranza, "Adaptive Nonlinear Digital Filters Using Distributed Arithmetic," IEEE ASSP, No. 3, June 1986.

[85] Irwin, M. and R.M. Owens, "Towards Designing, Testing, and Validating High Performance VLSI Signal Processors," Department of Computer Science, The Pennsylvania State University, May 1985.
[86] Linderman, R.W., "Signal Processing Application of 70 MHz Bit-Serial Hardware," SPIE Processing, Vol. 698, August, 1986.
[87] Denyer, P., "An Introduction to Bit-Serial Arithmetic for VLSI Signal Processing," Proc. 1984 Conference on Res. in VLSI, MIT, pp. 179-183, 1984.
[88] Denyer, P. and D. Renshaw, VLSI Signal Processing, A Bit-Serial Approach, Addison-Wesley, New York, 1986.
[89] Blanut, R., Fast Algorithms for Digital Signal Processing, Addison-Wesley, New York, 1986.
[90] Linderman, R., P. Chou, W. Ku, and P. Reusens, CUSP, "A 2 Micron CMOS Digital Signal Processor," 1984 Int. Conference on ASSP, pp. 1611-1614, March 1984.
[91] Irwin, M.J. and R.M. Owens, "Digit Online Architectures," U.S. Army Research Office, 1982.
[92] _____, and _____, "Fully Digit On-Line Networks," IEEE Transactions on Computers, Vol. C-32, April 1983, pp. 402-411.
[93] Mouftah, H.T. and A.I. Garba, "VLSI Implementation of a 5-Trit Full Adder," IEEE Proceedings, Vol. 131, Pt. G, Oct. 1984, pp. 214-220.
[94] Aytac, H.M., "Ternary Logic Based on a Novel MOS Building Block Circuit," Proc. 1986 Symp. on Multiple-Valued Logic, 1986, pp. 20-25.
[95] Takagi, N., H. Yasuura, and S. Yajima, "High-Speed VLSI Multiplication Algorithm with a Redundant Binary Adder Tree," IEEE Transactions on Computers, Vol. C-34, Sept. 1985, pp. 789-796.
[96] James, D.M., "Time and Area Calculations for SBNR Array Multiplier," Space Tech Corporation, Internal Report, Oct. 1986.
[97] Rhyne, T. and N. Strader II, "A Signed Bit-Sequential Multiplier," IEEE Proceedings on Computers, Vol. C-35, Oct. 1986, pp. 896-901.
[98] Widrow, B., J. McCool, M. Larimere, C. Johnson, "Stationary and Nonstationary Learning Characteristics of the LMS Adaptive Filter," Proceedings of the IEEE, Vol. 64, Aug. 1976, pp. 1151-1162.
[99] Ward, C., A. Robson, P. Hargrave, and J. McWhirter, "Application of a Systolic Array to Adaptive Beamforming," IEEE Proceedings, Vol. 131, Pt. F, Oct. 1973, pp. 638-645.
[100] Huertas, J.L. and J.M. Carmona, "Low-Power Ternary C-MOS Circuits," Proceedings 9th Int. Symp. on Multiple-Valued Logic, Vol. 9, 1979, pp. 170-174.
[101] Balla, P.C. and A. Antoniou, "Low-Power-Dissipation MOS Ternary Logic Family," Proceedings 14th Int. Symp. on Multiple-Valued Logic, Vol. 14, 1984, pp. 133-139.
[102] Jenkins, W.J., "VLSI Digital Signal Processors with Fault-Tolerance," Research Summary.
[103] Paul, D.F., "A Fault-Tolerant Integrated Digital Filter Using High-Speed Residue Arithmetic," Report CSG-21, Thesis, Univ. of Illinois at Urbana-Champaign, 1983.
[104] Watson, R.W. and C.W. Hastings, "Self-Checked Computation Using Residue Arithmetic," Proc. IEEE, Vol. 54, pp. 1920-1931, 1966.
[105] Mandelbaum, D., "Error Correction in Residue Arithmetic," IEEE Trans. Computers, Vol. C-21, pp. 538-545, 1972.
[106] Barsi, F. and P. Maestrini, "Error Correcting Properties of Redundant Residue Number Systems," IEEE Trans. Computers, Vol. C-22, pp. 307-315, 1973.

[107] Etzel, M.H. and W.K. Jenkins, "Redundant Residue Number Systems for Error Detection and Correction in Digital Filters," IEEE Trans. Acoustics, Speech, Signal Processing, Vol. ASSP-28, pp. 191-201, 1980.
[108] Fortes, J.A. and C.S. Raghavendra, "Gracefully Degradable Processor Arrays," IEEE Trans. on Computers, Vol. C-34, No. 11, pp. 1033-1044, Nov. 1985.
[109] Hwang, K. and F.A. Briggs, Parallel Computer Architecture, New York: McGraw-Hill, 1984.
[110] Uhr, L., Computer Arrays and Networks: Algorithm-Structured Parallel Architectures, New York: Academic, 1982.
[111] Batcher, K.E., "Design of a Massively Parallel Processor," IEEE Trans. Computers, Vol. C-29, pp. 826-840, 1980.
[112] Kung, H.T. and C. Leiserson, "Algorithms for VLSI Processor Arrays," Introduction to VLSI Systems, Mead and Conway, ed., Addison-Wesley Publishing Company, 1980, pp. 271-292.
[113] Kung, S.Y., K.S. Arun, R.J. Gal-Ezer, and D.V.B. Rao, "Wavefront Array Processor: Language, Architecture and Applications," IEEE Trans. Computers, Vol. C-31, pp. 1054-1066, 1982.
[114] Rosenberg, A.L., "The Diogenes Approach to Testable Fault Tolerant Arrays of Processors," IEEE Trans. Computers, Vol. C-32, pp. 902-910, 1983.
[115] Baqai, A.I. and T. Lang, "Reliability Aspects of the Illiac IV Computer," Proc. Conf. Parallel Processing, pp. 123-131, 1976.
[116] Kunn, R.H., "Yield Ennancement by Fault-Tolerant Systolic Array (Summary)," USC Workshop VLSI Modern Signal Processing, pp. 145-152, 1982.
[117] Pradhan, D.K., "Fault-Tolerant Architectures for Multiprocessors and VLSI Systems," Proc. 13th Fault-Tolerant Computer Symp., pp. 436-441, 1983.
[118] McDonald, J.F., E.H. Rogers, K. Rose, and A.J. Steckl, "The Trials of Wafer-Scale Integration," IEEE Spectrum, pp. 32-37, Oct. 1984.
[119] Moraga, C., "Systolic Systems and Multi-Valued Logic," 14th Int. Symp. Multi-Valued Logic, pp. 98-108, 1984.
[120] Lo, H. and S.C. Lee, "Fault Detection in M-Logic Circuits Using the M-Difference," 14th Int. Symp. Multi-Valued Logic, pp. 62-70, 1984.
[121] _____, and _____, "A Map-Partition Method for the Fault Detection of Multi-Valued and Multi-Level Combinational Logic Circuits," Proc. Int. Symp. Multi-Valued Logic, pp. 283-289, 1984.
[122] _____, "Generalized Fault Detection for Multi-Valued Logic System," Proc. Int. Symp. Multi-Valued Logic, pp. 178-186, 1984.
[123] Mouftan, H.T. and I.B. Jordan, "Integrated Circuits for Ternary Logic," Proc. Int. Symp. Multi-Valued Logic, pp. 285-302, 1974.
[124] _____, and _____, "Design of Ternary COS/MOS Memory and Sequential Circuits," IEEE Trans. on Computers, Vol. C-26, pp. 281-288, 1977.
[125] _____, "A Study on the Implementation of Three-Valued Logic," Proc. Int. Symp. Multi-Valued Logic, pp. 123-126, 1976.
[126] Huertas, J.L., J.I. Acha, and J.M. Carmona, "Design and Implementation of Tristables Using CMOS Integrated Circuits," IEE J. Electron. Circuits Syst., Vol. 1, pp. 88-94, 1977.
[127] Mouftan, H.T., "Design and Implementation of Tristables Using CMOS Integrated Circuits," IEE J. Electron. Circuits Syst., Vol. 2, pp. 61-62, 1978.
[128] Mouftan, H.T. and K.C. Smith, "Three-Valued CMOS Cycling Gates," Electron. Lett., Vol. 14, pp. 36-37, 1978.
[129] Carmona, J.M., J.L. Huertas, and J.I. Acha, "Realization of Three-Valued CMOS Cycling Gates, Electron. Lett., Vol. 14, pp. 288-290, 1978.

[130] Kuhn, R.H., "Interstitial Fault Tolerance - A Technique for Making Systolic Arrays Fault Tolerant," Proc. 16th Hawaii Int. Conf. on System Sciences, 1983, pp. 215-224.
[131] Sami, M.G., and R. Stefanelli, "Fault Tolerance of VLSI Processor Arrays: the Time Redundant Approach," Proc. 1984 Real Time Systems Conf., IEEE, 1984, pp. 200-207.
[132] Koren, I., and D.K. Pradhan, "Yield and Performance Enhancement Through Redundancy in VLSI and WSI Multiprocessor Systems," Proc. IEEE, Vol. 74, 1986, pp. 699-711.
[133] Moore, W.R., "Switching Circuits for Yield Enhancement of an Array Chip," Electronics Letters, 1984, pp. 667-669.
[134] Marwood, W., and A.P. Clarke, "Fault-Tolerant Systolic Architectures," 3rd National Workshop on Fault Tolerant Computing, Melbourne, Australia, July 1985.
[135] Sami, M, and R. Stefanelli, "Reconfigurable Architectures for VLSI Processing Arrays," Proc. Nat. Comp. Conf., AFIPS, 1983, pp. 565-577.
[136] Evans, R.A., "A Self-Organizing Fault Tolerant, 2D Array," VLSI-85, Tokyo, Japan, 1985, pp. 233-242.
[137] Moore, W.R., and R. Manat, "Fault-Tolerant Communications for a Wafer-Scale Integration of a Processor Array," Microelectron., Vol. 25, No.2, 1985, pp. 291-295.
[138] Negrini, R., M. Sami, and R. Stefanelli, "Fault Tolerance Approaches for VLSI/WSI Arrays," 4th Phoenix Computer Communications Conference (IEEE), 1985, pp. 505-509.
[139] Singh, A.D., "An Area Efficient Redundancy Scheme for Wafer Scale Processor Arrays," ICCD, 1985, pp. 505-509.
[140] Hedlund, K., "Wafer Scale Integration of Configurable Highly Parallel Processors," Ph.D. Thesis, Purdue University, 1982.

PAPERS PUBLISHED

UNDER

THIS EFFORT

A Systolic SBNR Adaptive Signal Processor

MICHAEL ANDREWS

Abstract — A new realization for adaptive signal processing units is proposed which uses a special subset of signed digit number representations (SDNR's). This signed binary number representation (SBNR) captures all of the efficiencies of SDNR arithmetic but also makes circuit realizations less complex. Furthermore, a natural interface between analog and digital numbers is provided. The serial on-line processing nature of SBNR utilizes the MSB first. An area/time complexity for VLSI implementations in comparable systolic array architectures contrasts the effectiveness of five different primitive VLSI cells and organizations.

I. Introduction

HURST [1] has noted that multi-valued logic (MVL) may show promise in the future for VLSI. At present, binary systems are facing interconnect problems which appear to be insurmountable. Silicon areas devoted to intrachip connections now consume twice the area of active logic elements on the chip [2]. Array implementations cause a severe escalation of interconnect area. Likewise, off chip connections are generating new and complex thermal and mechanical problems for the board designer. Such factors seek denser information content to interconnection ratios. In this paper, a redundant arithmetic solution is proposed and coupled with MVL relieves some of the silicon area inefficiencies with conventional binary arithmetic.

We examine one implementation of ternary arithmetic which when viewed as redundant numbers holds promise for division-sparse signal processing applications. Section II briefly describes basic digit number properties attractive to signal processing which manipulate sequential data streams. Section III discloses an efficient TRIT ternary digits realization which serves as the primitive VLSI cell. The realization utilizes a balanced encoding coupled with encoded redundancy to improve both logic delay and gate count.

Section IV identifies the intimate coupling between word and bit level matrix × matrix multiplication. Section V describes a systolic implementation of the least mean square (LMS) algorithm invoking signed binary number representations (SBNR's), which is easily realized with MVL. The LMS algorithm is a difficult adaptive signal processing benchmark because in-place coefficient methods do not apply. Section VI identifies appropriate ADC and DAC SBNR realizations. Section VII contrasts comparable realizations.

Manuscript received March 1, 1985; revised September 30, 1985. This work was sponsored by the U.S. Army Research Office under Grant DAAG29-83-C-0025.

The author is with Space Tech Corporation, Fort Collins, CO 80526.

IEEE Log Number 8406485

II. Signed Digit Number Representations (SDNR)

In the most general sense, a redundant number system allows both an increase in the number of positive and negative digits as follows.

Definition 1.

$$w_{\text{red}}^{(n,m)}: R \times R \times \cdots \times R \to Q \tag{1}$$

$$a_{n-1} \cdots a_0 a_{-1} a_{-2} \cdots a_{-m} \to \sum_{i=-m}^{n-1} a_i d^i \tag{2}$$

where the digits

$$d_i \in R = \{-r_1, -r_1+1, \cdots, 0, 1, \cdots, r_2-1, r_2\}, \quad r_1, r_2 > 0 \tag{3}$$

The representation described by (1), (2), and (3) is called **redundant notation with base d**. The basic properties of general SDNR are identified in Table I. Avizienis [3], Atkins [4], Tung [5], Ercegovac [6], and Robertson [7] have shown that SDNR can effectively operate in a general purpose digital computer for the reasons noted. However, the general redundant representation does not lead to efficient implementations unless restrictions are placed upon the number set.

III. Efficient SDNR Realization

Several implementations based on the SDNR have already been investigated [8]–[11], all of which sought to satisfy general data processing requirements of a mainframe computer. In contrast, signal processing applications are multiplication/addition intensive. An efficient SDNR realization is possible if we select the following redundant signed binary number representation (SBNR).

$$\sum_{i=1}^{n} X_i 2^{i-1}, \qquad X_i \in (-1, 0, +1) \tag{4}$$

with the notation $\bar{X}_i$ for $-X_i$ ($\bar{1}$ for -1). The redundant representation for 1 is 01 or $1\bar{1}$ while for -1 it is $0\bar{1}$ or $1\bar{1}$.

If we assume a two digit SBNR, nine states are possible covering the range -3 to $+3$ with all representations unique except for 1 and -1 which are (01 or $1\bar{1}$) and ($0\bar{1}$ or $\bar{1}1$), respectively. Of the 27-state inputs for a full adder truth table (i.e., three states each for the two digits to be added x_i and y_i and the carry out from the previous column (c_{i-1}), only six distinct cases described in Table II are necessary if we always represent 1 as 01 and $\bar{1}$ as $1\bar{1}$. Furthermore, in four entries (1, 3, 4, and 6) the carry out is completely determined by x_i and y_i. Hence, an adder need

0098-4094/86/0200-0230$01.00

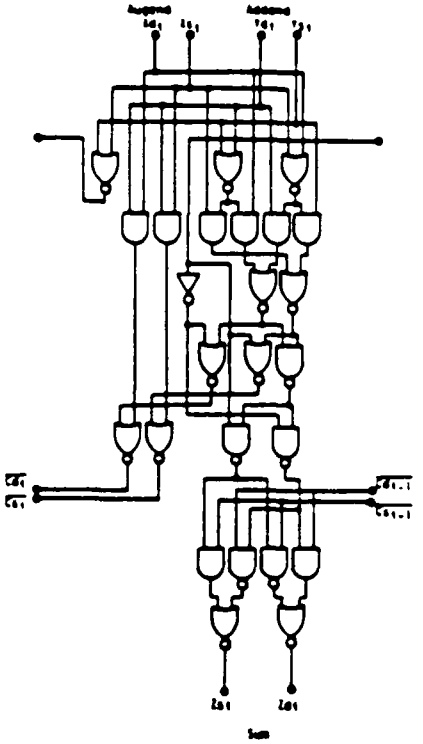


Fig. 1. Primitive bit-serial cell.

TABLE I
SDNR PROPERTIES

Parameter	Property
Zero in SDNR	Unique if $m = n$, $(d-1) \geq m$ and $m-1 \geq (d-1)/2$
Addition/subtraction	Totally parallel with minimal carry/borrow propagation and operation time independent of word length
Most significant digit	No special treatment
Digits	Positionally weighted with sign
Negation	Simple logical complementation of sign bits
Variable length operands	Handled easily by right-to-left methods
Multiplication	Tends to produce rounded results
Overflow detection	Immediately follows production of most significant digit
End-around carry	None hence single digit ALU slices are identical making VLSI highly regular

only consider carry in for cases 2 and 5 in order to generate carry out. For these two cases, carry out depends on whether the previous x_{i-1} or y_{i-1} is negative. From these considerations, the VLSI circuit proposed by Takagi *et al.* [9] in Fig. 1 suffices.

A primitive cell suitable for large VLSI arrays and especially for adaptive signal processors must have few interconnections beyond its nearest neighbors and very simple controls. Fortunately, many signal processing algorithms can be implemented with bit-serial arithmetic.

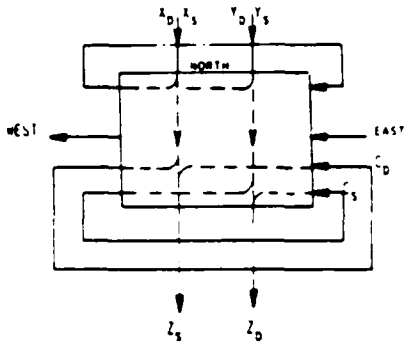


Fig. 2. Primitive cell (internal data flow).

TABLE II
INTERMEDIATE ADDITION STEP CLASSES

Type	Augend (x_i)	Addend (y_i)	Next Lower Position (x_{i-1}, y_{i-1})	Carry (c_i)	Intermed Sum (s_i)
1	1	1	—	1	0
2	1	0	Both are positive	1	$\bar{1}$
	0	1	At least one is negative	0	1
3	0	0	—	0	0
4	1	$\bar{1}$	—	0	0
	$\bar{1}$	1	—	0	0
5	0	$\bar{1}$	Both are positive	0	$\bar{1}$
	$\bar{1}$	0	At least one is negative	$\bar{1}$	1
6	$\bar{1}$	$\bar{1}$	—	$\bar{1}$	0

Fig. 2 represents the cell conceptually with dashed lines indicating the "data flow" internal to the cell where North, South, East, and West (N, S, E, W) data paths are available. Intercell connections shall only be to the nearest neighbors. Furthermore, latches on the north and east are incorporated in the cell to aid systolic latching of operand bits. The cell is now utilized in a systolic array where the dominant operation of $n \times n$ matrix multiplication is invoked.

IV. Matrix × Matrix Multiplication

Matrix operations may be either sums of word level products or sums of bit level products. Furthermore, a strong relationship exists between word and bit level systolic arrays [12]. Treated as bit level manipulations, fast area efficient VLSI arrays are possible [13], [14]. In our SBNR implementations, a systolic-like bit level approach is assumed where each processing cell is a multiplier and gated full adder.

To understand this word/bit dualism, we consider the implementations at the word level and show how bit level similarities apply. Multiplication of two $n \times n$ matrices, $S = (s_{i,k})$ and $H = (h_{k,j})$ to form the matrix product $Y = (y_{i,j})$ becomes

$$y_{i,j} = \sum_{k=1}^{n} s_{i,j} h_{k,j}, \qquad i, j = 1, 2, \cdots, n \tag{5}$$

Without any loss of generality, Y may be considered as independent vectors y_i. The aggregate of n matrix × vector

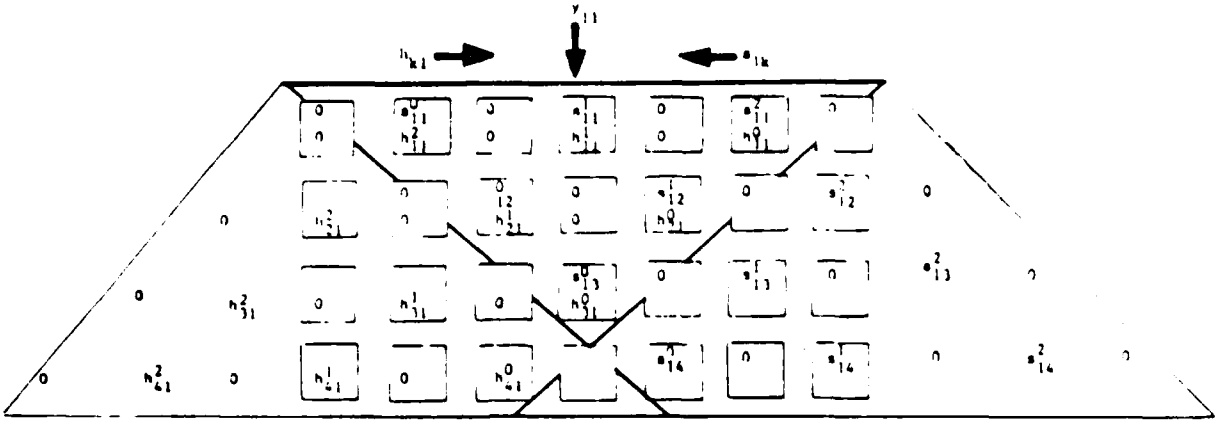


Fig. 3 Interaction of the bits in words s_{1k} and h_{k1} to form y_{11}

product evaluations, each of the type in (6) comprises the matrix Y.

$$y_i = \sum_{k=1}^{n} s_{ik} h_{kj}. \tag{6}$$

However, each of (6) is also an aggregate of n inner product evaluations of the type

$$y_i = \sum_{k=1}^{n} s_k h_k. \tag{7}$$

Multiplication of two matrices now becomes a series of unit multiplications of (7) and an accumulation of relevant product terms. For this reason, systolic arrays use a multiplier/accumulator PE. Equation (7) can be partitioned further into a series of bit level sum of products. The coefficient of each power of two in the result now becomes a convolution of the coefficients in the two operands. This important discovery allows us to organize the input signal streams so that operations at the bit level are pipelined onto our array, as in Fig. 3. The tantamount constraint is that the physical significance position in the array must be static so that partial products are accumulated correctly. We do not require the complicated carry/borrow strategies found in two's complement systems because our SBNR has a minimal carry/borrow distance.

Consider the multiplication of two 4×4 matrices as in Fig. 3. This diagram portrays the interaction of the bits of two sets of words, s_{1k} and h_{k1}, which compute y_{11}. Each square is a gated full adder unit of cells in Fig. 2. The data words in Fig. 3 have been expanded into their respective individual bits and the kth row is associated with the kth set of words. Words s_{1k} enter from the right while words h_{k1} enter from the left in a bit serial manner. Although we show the least significant bits (s_{1k}^0, h_{k1}^0) entering ahead of the next significant bits (s_{1k}^1, h_{k1}^1), the MSB's can also enter first in SBNR. Upon forming partial products, the intermediate results, y_{ij}, are passed vertically downward.

On a larger scale, the partial products y_{ij} are generated as in Fig. 4 in the shaded areas. Dashed lines adjacent to the parallelogram edges are guardbands to allow for growth generated by carry bits. For m-bit operands, $m + \lfloor 2 \log_2 n \rfloor$ bits are necessary. These guard bands are

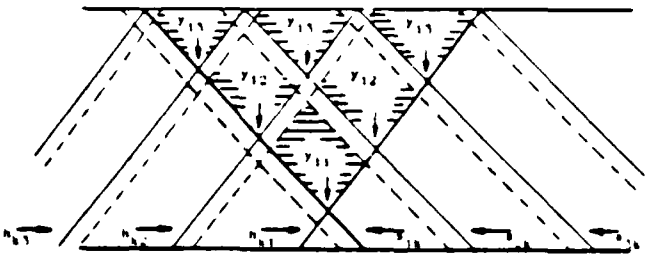


Fig. 4 Partial product generation of matrix × matrix multiplication.

equivalent to spacing input parallelograms with guardbands filled in with zero bits.

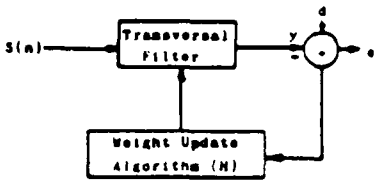
The shaded areas which move down vertically generate partial products such that successive cells at a given location in the shaded diamond area accumulate all terms in

$$y_{ij}^{pq} = \sum_{k=1}^{n} s_{ik}^{p} h_{kj}^{q}. \tag{8}$$

The full sum of products is formed by the accumulation of diamonds emerging from the bottom. A pipelined tree of adder cells connected to the bottom edge generates the full sum which can be clocked out least significant bit first or most significant bit first (SBNR only). The full sum is then computed every $2m + 1$ clock cycles.

It is important to note that the symmetry of diamonds in Fig. 4 carries over directly into regular VLSI cells with few intercell connections, resulting in an extremely efficient VLSI computational array. If SBNR numbers are not used, carry/borrow logic and intercell data paths would be complicated by the same level of complexity necessary to fabricate full carry lookahead adders (where carry propagate logic grows as a function of wordlength). In an SBNR implementation, only nearest neighbor cell paths and same cell replication are required.

Another advantage to SBNR is the absence of special circuitry and algorithms to handle signed operands. In two's complement arithmetic, the Baugh Wooley algorithm can be used [15]. In this procedure, two's complement words are treated as positive numbers if 1) a fixed correction term is added to the result for each word level multiplication, and 2) all partial products normally with a



y = Filter Output (Scalar)
e = Error Signal (Scalar)
d = Training/Conditioning Signal (Scalar)

Fig. 5. Adaptive signal processor.

negative weighting are complemented. Two's complement implementations on a systolic array require a negative weighting flag or a tag on the partial products which must propagate vertically down through the array. Hence, another latch and control line is required for each columnar path. Furthermore, final addition of correction terms requires an initialization of the accumulators in the adder trees.

V. An Implementation of the LMS Algorithm

An N-sampled LMS adaptive filter as depicted in Fig. 5 captures a signal, S, into a transversal filter whose scalar output, y, is obtained by convolving S with adapting coefficients H. An error signal, e, derived from the filter output and a training signal, d, drives the LMS weight update algorithm. The transversal filter has a set of N registers, each of length K bits which provides storage for the $N \times K$ array of bit values, B, for the signal S. Boldfaced characters are vectors or matrices. The independent time variable, t, is omitted, but is implicit to discussions. Necessary filter scalars and matrices are defined in Tables III and IV.

The signal vector, S, can be partitioned into the $BN \times K$ array as in (9).

$$B = \begin{bmatrix} s_0^1 & s_0^2 & \cdots & & s_0^N \\ s_1^1 & s_1^2 & \cdots & & s_1^N \\ s_2^1 & \cdot & & & \cdot \\ s_3^1 & \cdot & & & \cdot \\ s_4^1 & \cdot & & & \cdot \\ \vdots & & & & \\ s_k^1 & s_k^2 & \cdots & s_k^{N-1} & s_k^N \end{bmatrix},$$

$$\text{such that } s_i^j;\ 1 \leq j \leq N,\ 0 \leq i \leq k \quad (9)$$

where a superscript denotes a sampling moment and a subscript denotes a bit position in a K-bit word. The signal vector can be expressed as

$$S = BX. \quad (10)$$

The output of the filter is given by the convolution

$$y = S^T H \quad (11)$$

where the column vector H represents the set of N filter coefficients.

TABLE III
Adaptive Filter Scalers

$h(n)$ = nth coefficient of an N-point digital adaptive filter.
$f(k)$ = kth partial product used in the output accumulation.
$s(n)$ = K-bit input signal sample present at point n of an N-point digital filter.
y = digital filter output.
d = input training signal to digital adaptive filter.
$e = d - y$ = error sample generated by digital adaptive filter.

TABLE IV
Adaptive Filter Matrices

$S^T = (s(1), s(2), \cdots, s(n), \cdots, s(N))$
$H^T = (h(1), h(2), \cdots, h(n), \cdots, h(N))$
$F^T = (f(1), f(2), \cdots, f(k), \cdots, f(K))$
$X^T = (2^{-1}, 2^{-2}, \cdots, 2^{-k})$, i.e., the set of the first K negative integer powers of 2.
B = the $N \times K$ array of bit values which results when a K-bit input signal vector is stored in an N-point digital filter.

Define the column vector F as

$$F = B^T H \quad (12)$$

and substituting (10) in (11), using the property of matrix transposition, we have

$$y = X^T F \quad (13)$$

where the filter coefficients, F, is a set of partial products. The LMS algorithm updates

$$F' = F + 2ueB^T BX \quad (14)$$

where u is a convergence rate factor. Equations (13) and (14) form the iterative computational tasks of the filter.

Cowan *et al.* [16] have observed that the output filter formulation of (13) when compared with (11) reveals the essential elements of the distributed arithmetic architecture of the LMS algorithm depicted in Fig. 6. The input (analog-to-digital converter) signals are presented serially to a set of N cascaded K-bit shift registers. As this serial bit stream enters the shift registers, the shift register parallel outputs generate K N-bit address words on the RAM address bus. Each RAM datum is then right-shifted K bits and accumulated. The accumulation is complete after K memory accesses. Finally, an output sample is converted to an output analog signal. As in our implementation, the distributed arithmetic architecture uses no hardware multipliers. Using (14) in a matrix by matrix multiplication scheme naturally captures the bit-serial word-parallel power of systolic arrays behaving as SIMD data-flow engines.

An additional circuit reduction is possible when we utilize the latches in the primitive cells of Fig. 2 to store the input signal, S. Now, external RAM is no longer required. As a result, the VLSI implementation is more compact. Furthermore, vector and matrix transposition operations are easily accomplished by routing signals in the orthogonal direction since the primitive cells have NS and EW bidirectional ports saving considerable time. A circuit to implement the LMS algorithm is shown in Fig. 7.

This architecture utilizes two $n \times m$ cell systolic arrays and an adder tree. The upper array computes the filter

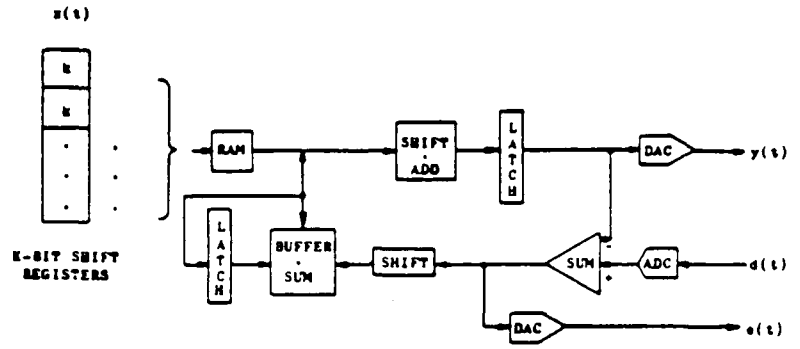


Fig. 6. Distributed arithmetic architecture.

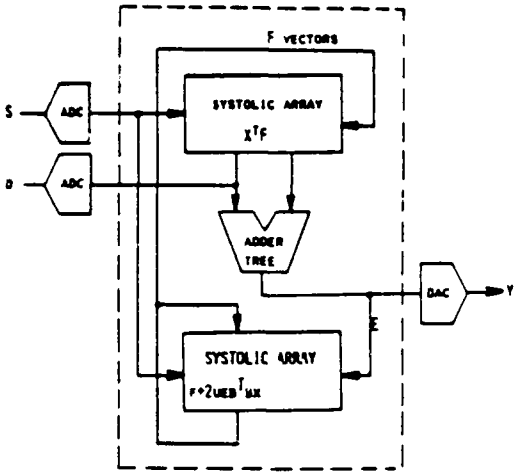


Fig. 7. Systolic LMS architecture.

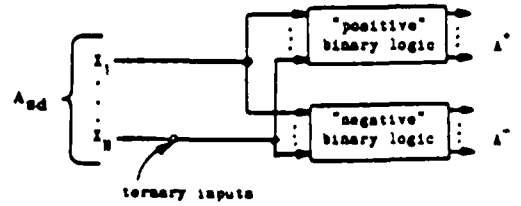


Fig. 8. 3-valued circuit.

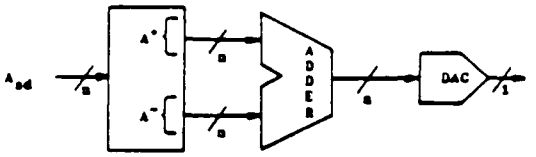


Fig. 9. Signed digit digital-to-analog converter.

output $y = X^T F$ while the lower array updates the filter coefficients. With two systolic arrays, as configured in Fig. 7, filter output and weight update can be pipelined so that the total computational delay from signal input sample, S, to output signal sample, y, is no greater than one-bit conversion of the ADC. An expensive ADC flash conversion is not necessary.

VI. ADC and DAC Methods

It is easily seen that a balanced redundant encoded number, A_{sd} can be represented by a "positive" part and a "negative" part, A^+ and A^-, respectively, as in (15).

$$A_{sd} = A^+ + A^- \tag{15}$$

where the operator "+" is the normal arithmetic operation. For example, the signed digit number, $\bar{1}10\bar{1}$, is the sum of

$$\begin{aligned}\bar{1}10\bar{1} &= (2^2) + (-2^3 - 2^0) \\ &= 4 - 9 = -5.\end{aligned} \tag{16}$$

This property makes digital-to-analog conversion trivial. The circuit of Fig. 8 displays the essential components [17].

Separating A_{sd} as above then permits us to simply add the parts together in a conventional adder whose result is represented in the number system compatible with the interconnected DAC as in Fig. 9.

The ADC realization is greatly simplified by noting that our TRIT representations require no positive number recoding. Negative numbers need only change the representation of the leading "one" to "$\bar{1}$" [9]. Two's complement binary numbers carry straight across to SBNR except for the leading digit and only if the number is negative. As a result, any ADC can be directly used which generates binary numbers (biased, offset, one's or two's complement, sign-magnitude). It is noteworthy to observe that these ADC/DAC efficiencies do not carry over for SDNR numbers.

VII. Comparative Performance

In this section, the LMS systolic SBNR architecture is compared to four other architectures. These cases are: 1) conventional 2's complement binary full-parallel adder/multipliers, 2) distributed arithmetic variation of (1) using bit-wise adders across the filter taps, 3) redundant arithmetic cells replacing the adders/multipliers of (1), and 4) bit-sequential arithmetic cells replacing the adders/multipliers of (1).

The LMS algorithm can be implemented in any of these architectures with either sparse or fully parallel/pipelined hardware. When implemented with $2N$ multipliers and $2N$ adders, as in Case 1 above, no faster implementation is possible. However, for most applications, $2N$ multipliers are overwhelmingly expensive in VLSI real estate.

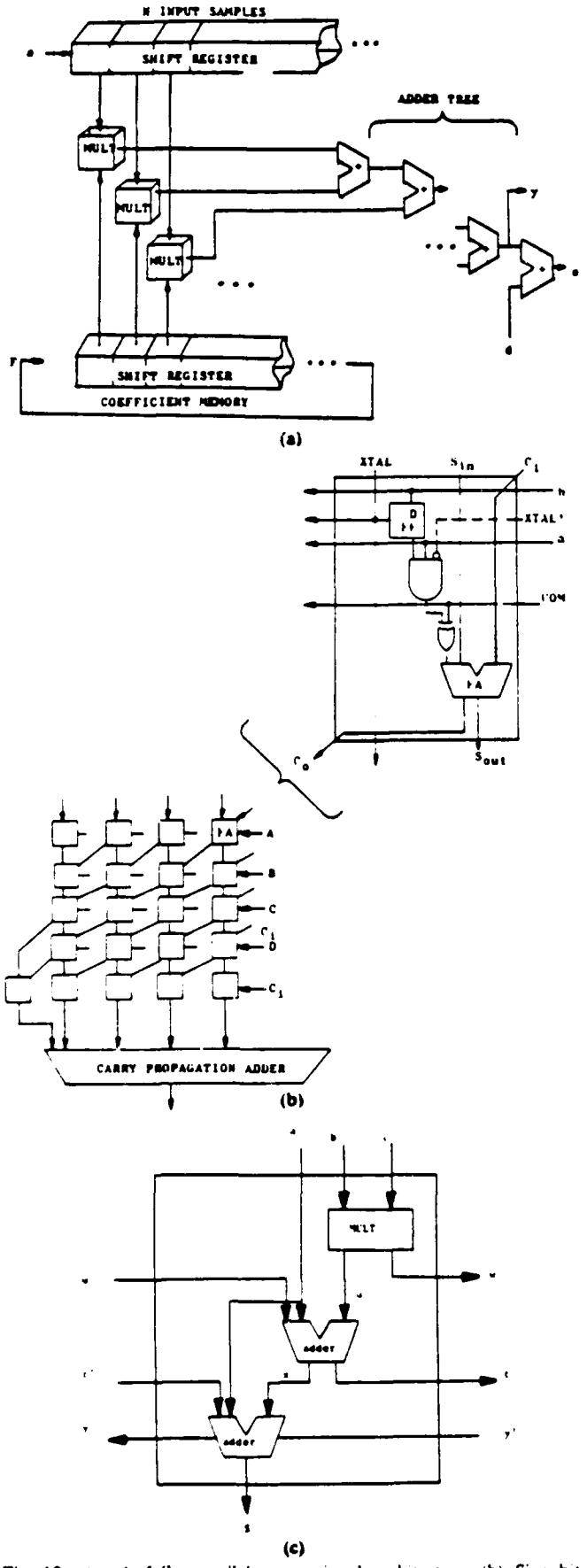


Fig. 10 (a) A fully parallel conventional architecture. (b) Sips bit-sequential architecture. (c) Redundant arithmetic cell architecture.

Comparable architectures are depicted in Fig. 10. Case 1 (Fig. 10(a)) utilizes the most hardware ($2N$ multipliers and $2N$ adders) in the conventional fully parallel sense. Case 2 is essentially the Cowan architecture of Fig. 6. Case 3 (Fig. 10(c)) is a redundant arithmetic cell proposed by Chow [10] where an SDNR implementation is assumed. Here, each cell incorporates two signed-digit adders and one signed-digit multiplier where signed-digits obey the properties of (1)–(4). Case 4 (Fig. 10(b)) is a bit-sequential cell approach also replacing the adders/multipliers of Case 1. This arrangement proposed by Sips [18] makes use of redundant arithmetic but not as efficiently as SBNR implementations because higher radices require more wire interconnect space.

The Sips bit-sequential cell can be configured in a linear two-dimensional array for $+$, $-$, $\times$, and $\div$ operations. Fig. 10(b) depicts the individual full adder (FA) cell with a D flip-flop for latching operands, a 3-input AND gate, and a 2-input XOR gate. If east/west as well as north/south paths are necessary, an additional flip-flop is required. The XOR gate obtains the complemented operand (for negative values). Control lines XTL and XTL' load successively new operand bits into the adjacent column for systolic addition as shown in the lower portion of the figure.

The SDNR cell depicted in Fig. 10(c) has been proposed by Chow for radix 16 members of the set $(-10, -9, \cdots, 0, 1, 2, \cdots, +9)$. The cell operations are described in Appendix A. Similar to the Sips cell, it uses redundant numbers, but in a two-level adder scheme. The second adder converts signed-digit numbers to conventional binary. Irwin and Owens [19] show that a systolic array of such cells can perform digit addition/subtraction in four gate delays, multiplication in six gate delays and shifting in zero gate delays. This systolic array has one severe drawback. Owens [20] shows that the redundant number set must be symmetric (i.e., $|r_1| = |r_2|$ in (3)) and multiplication operand digits must be fractions. As yet, no rapid integer, nonsymmetric multiplication algorithms exist for SDNR.

Gate costs listed in Table V for MOS realizations for each primitive logic element are used to derive the relative area/time complexities of typical CAD library cells needed for each architecture contrasted. We assume that the full adder (FA) circuits require 18 MOSFET's, 4 cells, 3 levels, and 11 intraconnections. Latches are fabricated from D-type flip-flops each requiring 16 MOSFET's, 8 cells, 7 levels, and 9 intraconnections [21, p. 207]. Dynamic shift registers require 8 MOSFET's, 4 cells, 2 levels, and 9 intraconnections per bit [21, p. 222]. Static MOS RAM cells each use 6 MOSFET's, 4 cells, 1 level, and 10 intraconnections [21, p. 249]. An N-input NAND ($N < 4$) gate requires $N+1$ MOSFET's, $N+1$ cells, 1 level, and $N+2$ intraconnections [21, p. 144].

Any VLSI chip is composed of interconnection area, effective chip area occupied by library cells, and an overhead area. Assuming then that a silicon compiler is used, the area and time complexities for common library cells of Kronlof [22] are relevant here. Table VI in conjunction with Table V relates the wordlength K to the L successive

TABLE V
MOS REALIZATIONS OF BOOLEAN FUNCTIONS

Function	#MOSFET's	Cells	Levels	Intraconnections
Inverter	2	1	1	2
NAND	N+1	N+1	1	N+2
Buffered NAND	N+5	N+5	3	N+5
NOR	N+1	N+1	1	N+2
XOR	3N+3	3N+3	3	3N+6
2-Bit Half Adder	15	4	3	9
2-Bit Full Adder	18	4	3	11
1-Bit RAM (Static)	6	4	1	10
1-Bit ROM	2	1	1	5
1-Bit Shift Register	8	4	2	9
D-Flip Flop	16	8	7	9
D-Flip-Flop (Master Slave)	32	16	14	20
S-R Latch	6	4	2	6

TABLE VI
LIBRARY CELL RELATIVE AREA/TIME COMPLEXITIES

Component Type	Area Compl.	Time Compl.
Parallel Multiplier ($B \times B$)	B^2	B
Accumulator ($K = LB + 2$)		
—adder (Brent–Kung)	$K \log K + 1$	$\log K + 1$
—shifter, e.g.	K	1
Adders (Brent–Kung)	$B \log B + 1$	$\log B + 1$
Coefficient Memory	BL	1
Pipeline Register	BL	1
Register, Ports, e.g.	B	1
LB out of 2B switch (MUX's)	$LB(L + 2B)$	1
Iteration Control (Counter)	$L \log L$	1
Queue Elements	BL	1
Systolic Cells		
Chow (SDNR)	4	2
Sips	1	1
SBNR	1	1

TABLE VII
COMPARISON OF ARCHITECTURAL COMPLEXITY SYSTOLIC ARRAY

	Conventional Binary ($2N$ multipliers) ($2N$ adders)	Distributed Arithmetic (Cowan)	Bit-Sequential Cells (Sips)	Redundant Arith. Cells (Chow)	Redundant Arith. Cells (SBNR)
Gate Complexity	$O(2mN)$	$O(kN)$	$O(km)$	$O(km)$	$O(kN)$
Latency	$N+1$ memory writes	kN bit shifts	δ + one ADC bit conversion	one ADC digit conversion	one ADC bit conversion
VLSI Amenable	structure irregular	moderate	yes	yes	yes
Estimated Pin Count/Cell	not appropriate	not appropriate	40*	10 m	10
Area Complexity	$> 2N(B^2 + K \log K + 1 + K + BL)$	$2K \log K + 1 + 2K + 2BL + 2B$	$K \log K + 1 + B \log B + 1 + 2B$	$2(B/m)^2 + 4(B \log B + 1)$	$K \log K + 1 + B \log B + 1$
Time Complexity	$> \max(B \text{ or } \log K + 1)$	$\log B + 1$	$\log B + 1 + 2 \log K + 1$	mB	B

*Assumes each cell is a 4-bit slice.
m = number of digits in a word.
k = number of bits in each shift register ($k < m$).
N = number of filter coefficients.
δ = small positive constant (3 or 4) less than the time to complete a full bit-parallel operation.

bit fields of B where

$$B = \lfloor K/L \rfloor \text{ bits.} \tag{17}$$

For two's complement numbers in binary fixed-point representation, the real value, X, can be represented by an unsigned binary integer value, X_b, and the MSB bit, X_0 as in

$$X = X_b/2^{LB-1} - 2X_0. \tag{18}$$

Hence, a wordlength K has LB bits. The area penalty of wires is proportional to B and to the square root of the effective chip area. Power distribution lines and bonding pads constitute the overhead area. An SBNR cell is assumed to have unity area and time complexity because each cell is basically a "one-bit" device. The Chow cell essentially has an area complexity four times the SBNR because its SDNR realization basically assumes 4 bits per digit on a radix 16 representation. The time complexity is double because another level of logic depth is required.

Using Table VI, the area/time complexities of each of the five architectures can be compared. Table VII also lists the gate complexity latency, VLSI-suitability, and pin count/cell.

VIII. Conclusions

The conventional binary architecture is hardware intensive yet is ultimately the fastest. The distributed arithmetic is a compromise between speed and silicon space. However, a regular design for VLSI is not easily achieved since no repetitive cell is utilized as in the three systolic implementations. Of these, the SBNR systolic implementation is highly regular, possessing very short signalling wires. Furthermore, local control in this self-timed synchronous system eliminates the need for global control lines which degrade performance of synchronous systems as in the conventional binary architecture.

A number system entirely composed of signed-bits $(-1, 0, 1)$ amenable to ternary valued circuits has been proposed for signal processing units where add/multiply cycles dominate. Such SBNR implementations can be configured as a systolic array to perform $n \times n$ matrix operations. Because the carry/borrow distance is minimal for SBNR, intercell communication is reduced. As a result, extensive carry-propagation, lookahead hardware is no longer required and mathematical operations are no longer dependent on wordlength as in conventional two's complement binary systems. Thus synchrony so vital to systolic arrays is more easily achieved and true data-flow SIMD machines result.

Although the area and time complexities of the three systolic arrays are comparable, the latency (time interval from input signal to output signal) is smallest for the SBNR array. Furthermore, the SBNR offers a successful fault tolerant implementation [4], [10]. The estimated pin count/cell is smallest for the SBNR array. This is because the left-to-right (MSB-to-LSB) computing property of SBNR numbers allows us to begin computations upon receipt of the MSB from the ADC. In view of these properties, we conclude that the SBNR systolic array is a competitive if not superior alternative to other implementations. We anticipate future signal processing architectures will take advantage of SBNR.

Appendix A
SDNR Chow Cell Operations

The SDNR cell of Fig. 10(c) is capable of addition, subtraction, multiplication, and assimilation (which assists data conversion). All operands are assumed to be normalized floating-point numbers of the form

$$X = r^{(E_x)} \cdot \sum_{i=0}^{m} x_i r^{-i} \tag{A-1}$$

where E_x is an integer with an e-bit two's complement representation. In the following operations, t' and w' are transfer digits which perform the same intermediate carry/borrow functions as our SBNR Z_{d_i}, Z_{s_i}, and C_{d_i}, C_{s_i} bits except that t' and w' digits each require multiple lines (e.g., a radix 16 SDNR digit can be represented with 5 bits, one for sign and four for magnitude).

Definitions

$w_{max} = \lfloor (r-1)/2 \rfloor$.
$t_{max} = \lfloor (r-1)/2 \rfloor$ if $\rho - \lfloor (r-1)/2 \rfloor$ is even,
$= \lfloor (r-1)/2 \rfloor + 1$, otherwise.
$u_{max} = \lfloor (\rho^2 - w_{max})/r \rfloor$.
$x_{max} = \lfloor (\rho + u_{max} + w_{max} - t_{max})/r \rfloor$.
D_h contains digits from 0 to $r-1$.
$D_{(r-1)}+$ contains digits from 0 to $r-1$.
$D_\rho \sqcup D_{(r-1)}+$ contains digits from ρ to $r-1$.

Input Digits

The input digits a, b, and c belong to the digit sets $D_\rho \sqcup D_{(r-1)}+$, D_ρ, and D_ρ, respectively.

The transfer digits t' and w' belong to the digit sets $D_{(\rho - x_{max})}$ and $D_{(w_{max})}$, respectively.

The borrow β' is either 0 or 1.

Functions of the Three Levels

[M] If M-MULT = 1
then $ru + w \leftarrow bc$ with u in $D_{(u_{max})}$ and w in $D'_{(w_{max})}$,
else $u \leftarrow b$ and $w \leftarrow 0$.

[S1] If S1-ADD = 1
then $rx + t \leftarrow a + w' + u$,
else $rx + t \leftarrow a - w' - u$.
In either case, x is in $D_{(x_{max})}$ and t is in $D_{(t_{max})}$

[S2] If ASSIM = 1
then $-r\beta + s \leftarrow a - \beta'$ with s in $D_{(r-1)}+$ and β is 0 or 1,
else $s \leftarrow t' + x$.

References

[1] S. L. Hurst, "Multiple-valued logic, its status and its future," *IEEE Trans. Computers*, vol. C-33, pp. 1160–1179, Dec. 1984.

[2] S. Trimberger, "Automating chip layout," *IEEE Spectrum*, pp. 28–35, June 1982.

[3] A. Avizienis, "Signed-digit number[s] representations for fast parallel arithmetic," *IRE Trans. Electron. Computers*, vol. EC-10, pp. 389–400, Sept. 1961.

[4] D. E. Atkins, "Design of the Arithmetic Units of ILLIAC III: Use of Redundancy and Higher Radix Methods," *IEEE Trans. Com-*

puters, vol. C-19, pp. 720–733, Aug. 1970.
[5] C. Tung, "Signed-digit division using combinational arithmetic nets," *IEEE Trans. Computers*, vol. C-19, pp. 746–749, Aug. 1970.
[6] M. D. Ercegovac, "A general hardware-oriented method for evaluation of functions and computations in a digital computer," *IEEE Trans. Computers*, vol. C-26, pp. 667–680, July, 1977.
[7] J. E. Robertson, "A new class of digital division methods," *IRE Trans. Electron. Computers*, vol. EC-7, pp. 218–222, Sept. 1958.
[8] J. E. Robertson, "Parallel digit arithmetic unit utilizing a signed-digit format," U.S. Patent 3 462 589, 1969.
[9] N. Takagi, H. Yasuura, and S. Yakima, "A VLSI-oriented high-speed multiplier using a redundant binary addition tree," *Systems-Computers-Controls*, vol. 14, pp. 19–28, 1983.
[10] C. Y. F. Chow, "A Variable Precision Processor Module," Ph.D. Dissertation, University of Illinois, Computer Science Dept., Urbana, 1980.
[11] J. E. Robertson, "Design of the combinational logic for radix 16 digit slice for a variable precision processor module," in *Proc. 1983 IEEE Int. Conf. on Computer Design*, Port Chester, NY, Oct. 1983.
[12] J. V. McCanny, K. W. Wood, J. G. McWhirter, and C. J. Oliver, "The relationship between word and bit level systolic arrays as applied to matrix × matrix multiplication," SPIE, Real Time Signal Processing VI, vol. 495, pp. 114–120, 1983.
[13] R. A. Evans, D. Wood, K. Wood, J. V. McCanny, J. G. McWhirter, and A. P. H. McCabe, "A CMOS implementation of a systolic multibit convolver chip," presented at *VLSI 83*, Norway, Aug. 1983.
[14] A. Corry and K. Patel, "Architecture of a CMOS correlator," presented at the *IEEE Int. Symp. on Circuits and Systems*, May 1983.
[15] K. Hwang and Y. H. Cheng, "VLSI computing structures for large scale linear systems of equations," in *Int. Conf. on Parallel Processing*, pp. 217–227, Aug. 1980.
[16] C. F. Cowan, S. G. Smith, and J. H. Elliott, "A digital adaptive filter using a memory-accumulator architecture: Theory and realization," *IEEE Acoustics, Speech, Signal Processing*, vol. ASSP-31, pp. 541–549, June 1983.
[17] A. Pugh, "Application of binary devices and Boolean algebra to the realization of 3-valued circuits," *Proc. Inst. Elect. Eng.*, vol. 114, pp. 335–338, 1967.
[18] H. J. Sips, "Bit-sequential arithmetic for parallel processors," *IEEE Trans. Computers*, vol. C-33, pp. 7–20, Jan., 1984.
[19] M. J. Irwin and R. M. Owens, "Fully digit on-line networks," *IEEE Trans. Computers*, vol. C-32, pp. 402–406, 1983.
[20] R. M. Owens, "Techniques to reduce the inherent limitations of fully digit on-line arithmetic," *IEEE Trans. Computers*, vol. C-32, pp. 406–411, 1983.
[21] S. Muraga, *VLSI System Design*. New York, Wiley-Interscience, 1982, pp. 144, 163, 207, 222, 249.
[22] K. Kronlof, I. Hartimo, and O. Simula, "Cost Evaluation of Digital Signal Processing Systems on VLSI," in *IEEE Int. Symp. on Circuits and Systems*, pp. 276–279, May 1984.

Michael Andrews (M'74–SM'83) received the B.A. and B.S. degrees from Rutgers University, NJ, in 1963 and the M.S. and Ph.D. degrees from the University of Arizona in 1972.

From 1974 to 1985, he has been with the electrical engineering and computer science departments at Colorado State University and with Space Tech Corporation as senior scientist. From 1982 to 1983, he was with the Army Research Office, Research Triangle Park, N.C., managing university research contracts in computer science. He has published over 4 workbooks, 60 papers, and two texts, *Principle of Firmware Engineering*, CS Press, and *Programming Microprocessor Interfaces for Control and Instrumentation*, (Prentice Hall, Englewood Cliffs, NJ). His current interests include systolic array, signal processing, redundant arithmetic, and microprogrammable architectures.

EFFICIENT NUMBER SYSTEMS FOR HIGH SPEED VLSI SIGNAL PROCESSORS

Michael Andrews
Space Tech Corporation
Fort Collins, Colorado

Published at the
Vail Workshop
Vail, Colorado
February, 1987

ABSTRACT

A comparative study of various number systems i[n] the relative merits for real-time signal processing[,] signed digit, redundant number, and a new repr[esen]binary number representation (SBNR) are contrasted[for] complexity. It is shown that the SBNR system h[as] complexity and regularity attributes amenable to V[LSI] proposed for minimal intercell connectivities, a prere[quisi]te for systolic array implementations. A test case realizing the least-squares algorithm in a systolic array for adaptive beamforming applications indicates comparative differences. Executing a square-root free Givens rotation matrix operation iteratively in a two systolic array configuration demonstrates real time signal processing for beamforming.

INTRODUCTION

A comparative architecture study was performed in order to implement the scaled Givens rotation solution to the least-squares minimization problem. Three architectures are examined: a) Conventional Systolic Array, b) Distributed Arithmetic Array, and c) SBNR Systolic Array. Important considerations in adaptive beamforming algorithm to architecture mapping include gate count estimates for some of the architectures and tables for performing these estimates. A systolic architecture for an adaptive beamformer tracking system is developed for performing recursive least-squares minimization.

The purpose of this project is to identify engineering trade-offs and interconnection strategies capable of achieving real-time implementation of signal processing algorithms via limited user-programmable mechanisms (e.g., firmware). Flexible firmware-oriented architectures dedicated to signal processing can then be identified. The specific test algorithm performs an orthogonal triangularization of the data matrix using a pipelined sequence of Givens rotations and generates the required residual without having to solve the associated triangular linear system by back-substitution.

Array Architectures

Systolic array architectures remain diverse. At the extreme ends are the WARP array and the GAPP array. WARP utilizes 68000 microprocessors in each processing element (PE). GAPP uses a 1 bit ALU with 128 bit RAM as each PE. Although more primitive (68000 is a 16 bit parallel engine), GAPP is a single chip of 72 PE's. Because of its high speed and availability, GAPP is viable. Between these outlying architectures lie conventional and distributed arithmetic processing cell compositions.

This work was sponsored in part by Army Research Office Contract #DAAG29-83-C-0025. The views, opinions, and findings contained in this report are those of the author and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

A study of basic PE's was made. A new PE utilizing a primitive cell is proposed for a systolic array PE. Anotner alternative PE based on a distributed arithmetic cell was studied. This cell increases computation speed by reducing multiplication to table-lookup of partial products and a series of shift/add operations.

Beamforming Architectures

An antenna beam is a collection of point sources or receptors where geometry governs the characteristic equations of the system. For a uniformly spaced line array as depicted in Figure 1, the following form applies:

$$G(a) = \sum_{n=0}^{N-1} g_n e^{-j2\pi(n d\cos(a)/\lambda)} \qquad (1)$$

Equation 1 has the basic form of a DFT. When we consider that $\cos a_k = (k/N)(\lambda/d)$, the beamformer output at angles a_k is computable by the DFT as follows.

$$G_k = \sum_{n=0}^{N-1} g_n e^{-j2\pi nk/N} \qquad (2)$$

From this we can easily see that a 2-D temporal-spatial Fourier transform can form beams in the nonuniformly spaced look directions. Adaptive beamforming must then cause the beam pattern to favor certain spatial or spectral parameters.

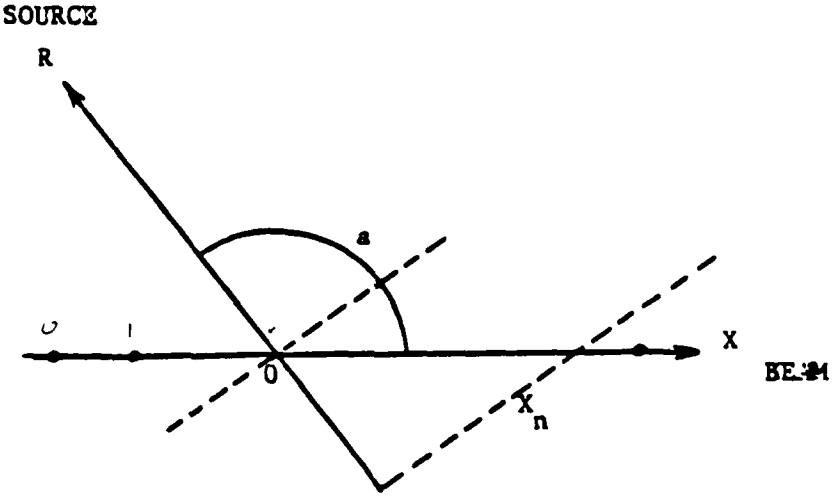


Figure 1 Line Array Beamforming

Adaptive Beamformer and Tracker System

Feintuch, et. al.[1], investigated an adaptive tracking system which employed the LMS algorithm to minimize the error between two beams of a split array. The weights generated are analyzed to determine the max

weight. It roughly corresponds to the delay between the phase centers of the two beams. The phase or time-delay is then used to provide a bearing estimate (for adaptive nulling, etc.).

We consider a least-square implementation of the adaptive tracker to construct a completely systolic adaptive beamformer/tracker from the systolic Givens rotation, DFT, and backsubstitution architectures. Feintuch provides a suitable starting point for incorporating adaptability. Simply stated, we use the peaks between weights to electronically steer the beam to force nulls at jammer angles. A time-domain least-squares adaptive tracking system can be configured as shown in Figure 2. Two inputs are required for the adaptive tracker. Multiple time "snapshots" of these samples are collected to form the $\underline{y}(n)$ and $\mathbf{X}(n)$ and computes the weight vector $\underline{W}(n)$ which minimizes the least-squares norm:

$$E(n) = ||\underline{e}(n)|| = ||\mathbf{X}(n)\underline{W}(n)+\underline{y}(n)|| \qquad (3)$$

The largest tap of the weight vector is then found and provides the phase bearing estimate.

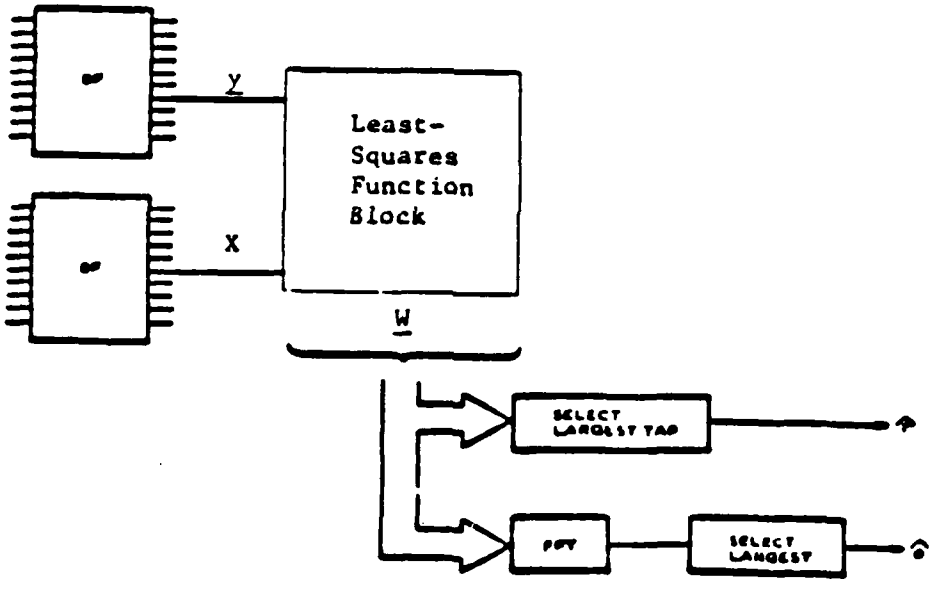


Figure 2 Least-Squares Time Domain Adaptive Tracker

In the frequency domain solution, shown in Figure 2, domain inputs undergo a Fourier transform. Multiple time "snapshots" of one half the array are taken to produce a matrix of frequency components for the LS algorithm. The largest tap over the frequency weights is selected and the phase provides a bearing estimate which is used to steer the beam.

Systolic Adaptive Beamformer and Tracking System

Figure 3 shows a complete frequency-domain adaptive beamformer and tracking system. The computational intensive components of the system are systolic array modules.

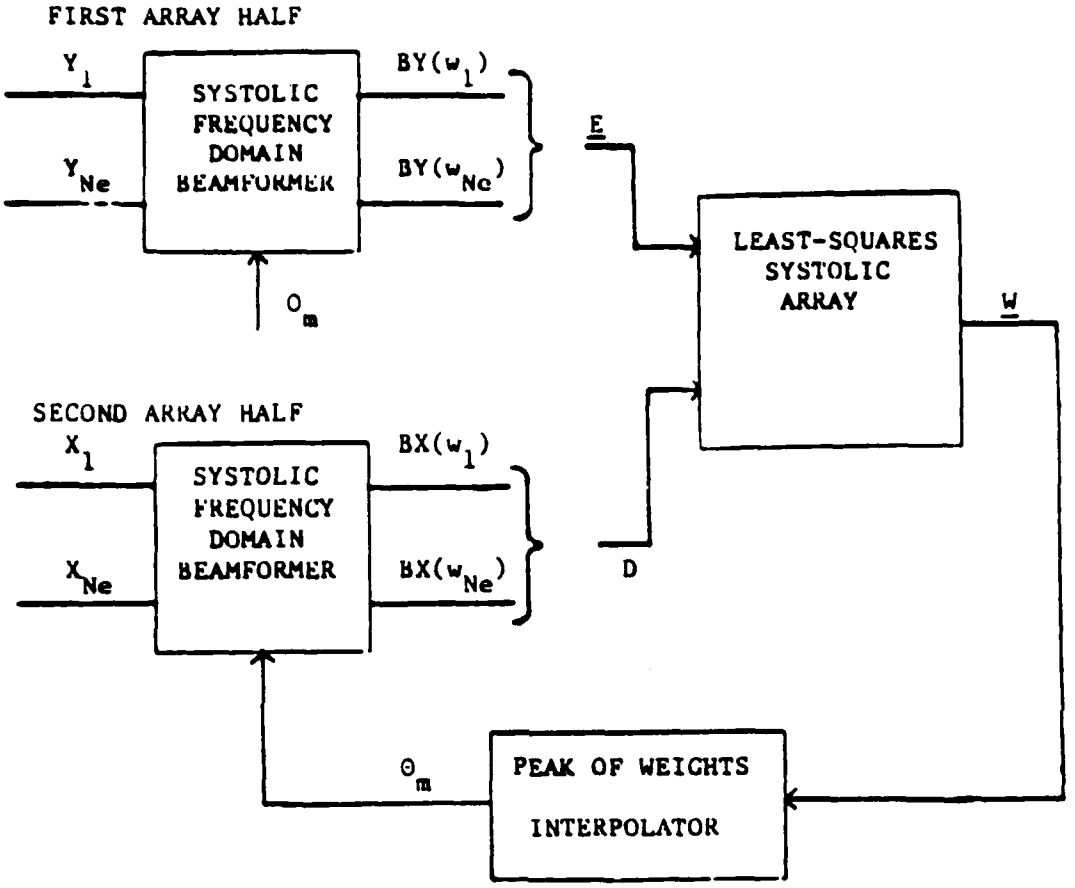


Figure 3 Frequency Domain Adaptive Beamformer and Tracker

The K-point DFT modules of the system perform a Fourier transform of the time domain input data. A system consideration at this point is the Fourier transform throughput. The phase shift multiply is driven by the phase estimate from the adaptive tracker. Each frequency component from the Fourier transform is multiplied by the term

$$e^{-iw_m T_m}$$

where T_M is a function of the steering angle. A multiplier array operates at this function block. A conventional distributed arithmetic engine or SBNR distributed arithmetic engine may be ideal for this array since W_m is only position dependent and the steer angle is the only variable and non-position dependent quantity in the computation of T_m for a fixed sensor array. Hence, an ultra fast table look-up of $e^{-iw_m T_m}$ can occur based solely on the steering angle.

An adder array is used to form the frequency bins of the beam. Since the beam is already in the frequency domain after the summing operation, the bins can be fed directly to the LS algorithm. The LS block consists of a Givens rotation systolic array and a backsubstitution array to compute the weights. The peak of the weight vector can be found using a quadratic interpolator. The interpolator performs a quadratic fit to the largest weight element and the two adjacent weights in the frequency domain.

Systolic Array Least Squares Solution

McWhirter[2] proposes a set of 3 primitive cells arranged in a triangular systolic array which performs recursive least-squares minimization. Orthogonal triangularization of the data matrix is performed using a pipelined sequence of square-root free Givens rotations. The square-root free Givens rotation triangular systolic array is shown in Figure 4. The associated primitive cells are given in Figure 5. To provide a common performance testbed, the conventional binary, SBNR[3], and distributed arithmetic architectures were studied based on an implementation of this systolic array.

Barlow and Ispen[4] developed a scaled Givens rotation systolic algorithm. The scaled Givens rotation algorithm operates on banded matrices of width $w = p + q + 1$ where p is the number of superdiagonals and q is the number of subdiagonals. Assuming m rows in the banded matrix and s right hand side vectors, the number of computation steps are given by:

$$2m + 3(q+1) + z + 1 \qquad (4)$$

The individual cell complexity (the number of equations solved at each cell) is approximately the same as those for the square-root free Givens rotation. Only one division operation is required in the scaled Givens rotation and many of the multiplies are reduced to shift operations. Both scaled Givens rotations and square-root free Givens rotations have processor utilizations of approximately 50%.

Conventional Binary Implementation

The GAPP™ is a commercially available systolic array device providing 72 conventional binary processing elements, dimensioned as a 6 X 12 rectilinear array. The square-root free or scaled Givens rotation is easily implemented on this device. In order to obtain realistic speed estimates for a conventional binary implementation of the square-root free Givens rotation, code for the GAPP device has been written in a C-like language known as GAL™.

The GAL square-root free Givens rotation code requires approximately

$$(2r + c + 1)(83n^2 + 224n + 156) \qquad (5)$$

instruction cycles where n is the bit length of the input operands, r is the number of matrix rows, and c is the number of matrix columns. The latency from first input to first residual output is

$$(c + r + 1)(83n^2 + 224n + 156) \qquad (6)$$

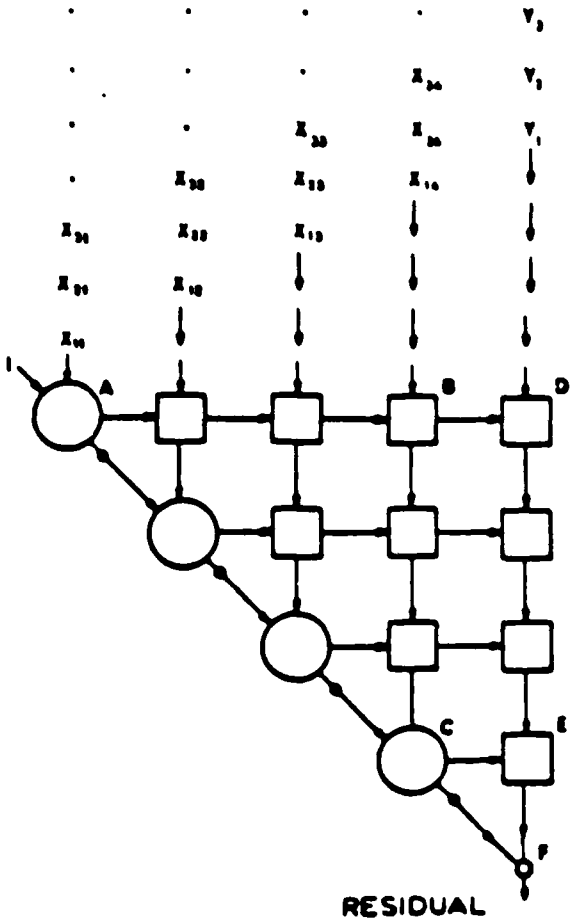


Figure 4 Systolic Array for Recursive Least-Squares Minimization

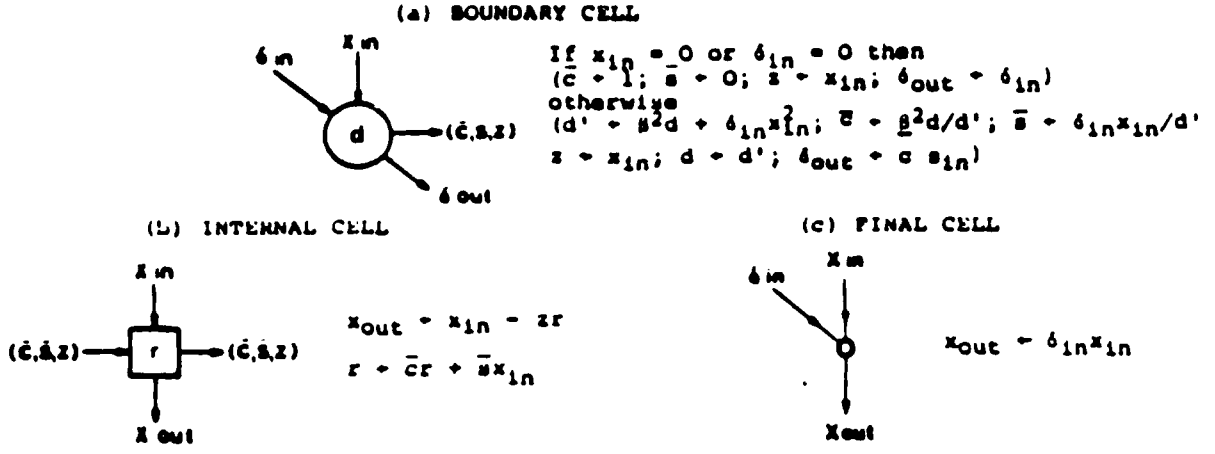


Figure 5 Cells Required for the Square-Root Free Givens Rotation

The time to complete the entire matrix reduction increases linearly with the size of the array. The number of elements processed, however, increases as the square of the array size. To estimate the processing power of the GAPP solution, compute the number of array elements processed per instruction cycle. Assuming a fixed word length, the word length dependent term is a constant $K = 85n^2 + 224n + 156$. The number of array elements processed per cycle is:

$$rc/(2r + c + 1)k \qquad \text{elements/cycle} \tag{7}$$

It can be seen that for square matrices ($r = c$) the number of elements processed per cycle increases quadratically as the array size increases. Improvements in speed may be obtained if concurrency can be achieved in the operation of the three cell types of McWhirter's algorithm. A promising solution is to use three separate arrays (one for each cell type), and carefully synchronize data flow between the arrays.

SBNR Implementation

A mesh connected systolic array of SBNR cells is used to implement McWhirter's algorithm. A single SBNR cell is shown in Figure 6. It consists of an appropriate set of registers which act as input to an intermediate SBNR ALU and a final SBNR ALU.

It is possible to derive the minimum execution speed and latency for this particular SBNR implementation of McWhirter's systolic array by considering the data dependencies in the equations. The maximum data dependency path length for the boundary, internal, and final cells are 5, 2, and 10, respectively. Using the 2N+1 formula for latency, we can compute latencies and speed estimates for each cell.

The maximum boundary cell latency is 11 cycles. At the internal cell, the maximum latency is 5 cycles. At the final cell, the maximum latency is 3 cycles. If r is the number of rows and c is the number of columns, then the maximum latency to the first residual is:

$$L(c,r) = 11(c + r + 1) \tag{8}$$

The execution time for the entire Givens rotation is:

$$S(c,r,n) = (2r + c + 1)(11 + n - 1) \tag{9}$$

where n is the bit length of the operands. Notice that the execution time is linearly dependent ($O(n)$) on the bit length of the operands where the GAPP array execution time is quadratically dependent ($O(n^2)$). This is a result of the reduction of multiplication complexity in SBNR arithmetic.

Distributed Arithmetic Implementation

The goal in distributed arithmetic architectures is to reduce computation time by performing table look-up to produce partial computation results. For example, multiplication can be reduced to a table look-up of partial products followed by a series of shift and adds to obtain the final result. In an N bit by N bit multiply, it is possible to divide each operand into k segments. By combining each segment of one operand with every other segment of the other operand, an address in RAM

of each partial product is formed. The partial products are looked up and, through a series of shifts and adds, accumulated to form the final product. Table 1 shows a comparison of a typical N bit multiply using conventional binary versus distributed arithmetic.

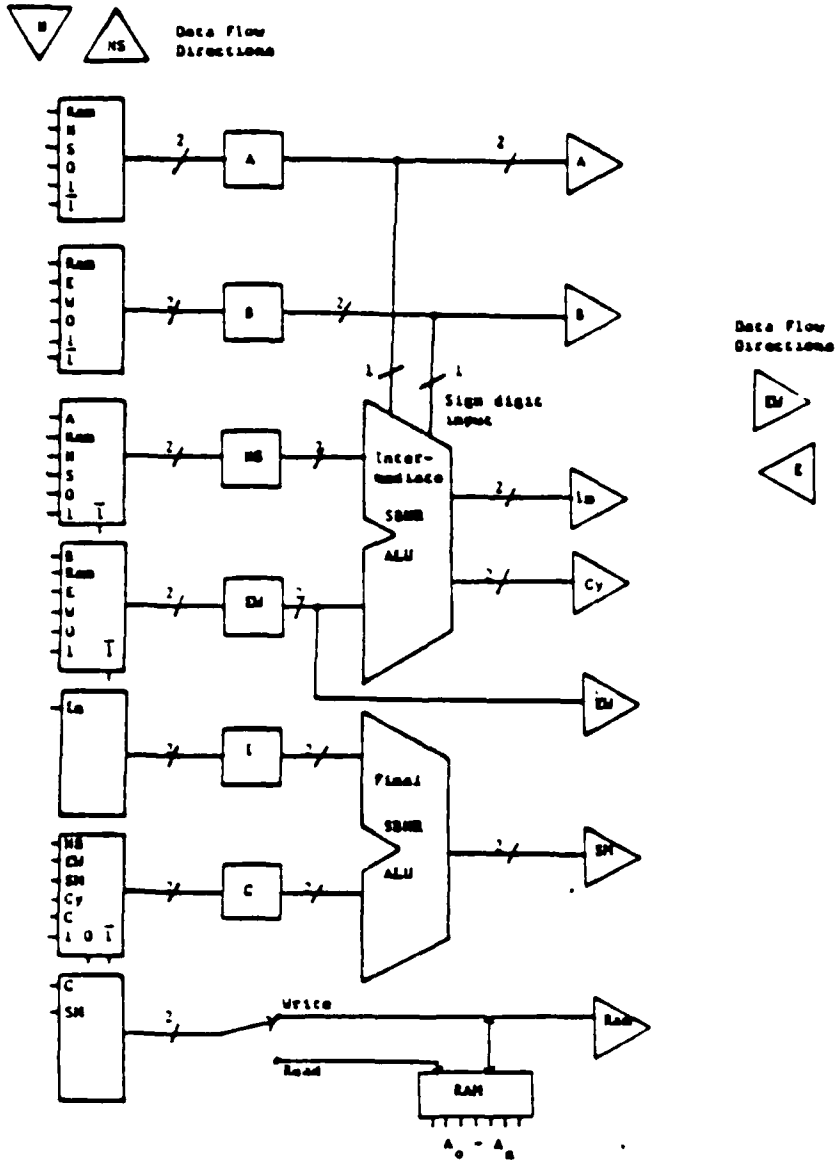


Figure 6 An SBNR Primitive Cell for a 2-D Mesh Connected Array

Table 1 N Bit By N Bit Multiply Comparison

Conventional Binary Operations	Distributed Arithmetic Operations
n shifts	k^2-1 shifts
n adds	k^2-1 adds
n^2 ands (bit-wise)	k^2 table look-ups

If RAM access speed is greater than the computation time for n^2 AND operations, then distributed arithmetic provides better performance than conventional arithmetic for $k < n$. There is a trade-off between table size and computation speed. The table size for any distributed arithmetic multiplier is $2^{2n/k}$ words. While computation time is directly proportional to k, the table is indirectly proportional to k (i.e., large k implies larger computation time but smaller table size).

An n bit distributed arithmetic computational element is shown in Figure 7. This computational element is a single bit ALU with the special feature that a table address register can be loaded and a partial product retrieved for further computation.

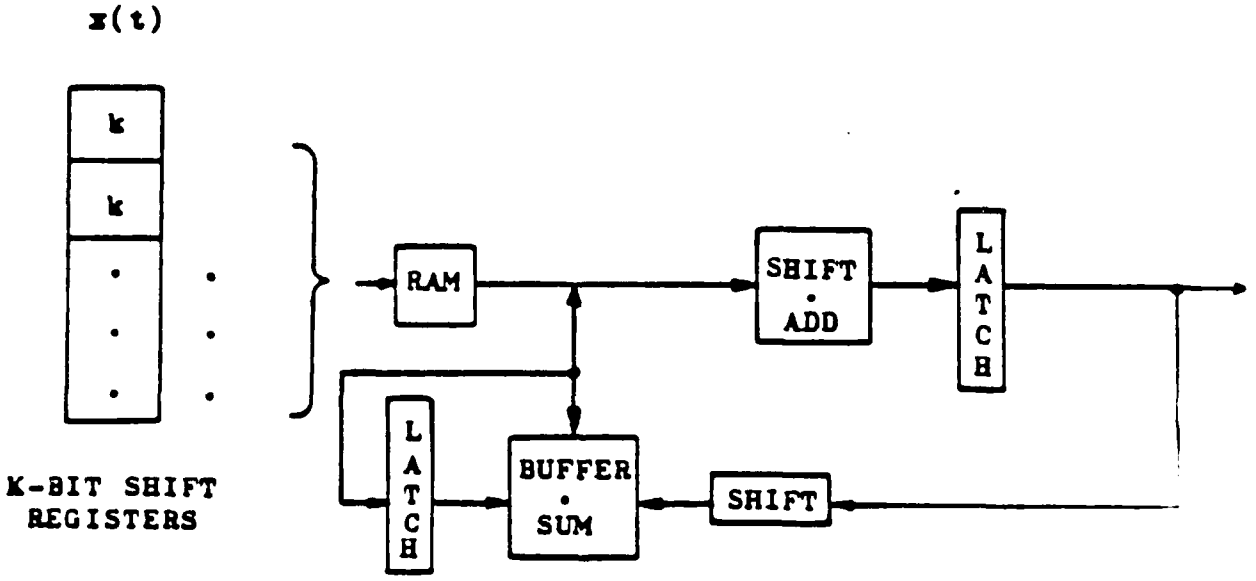


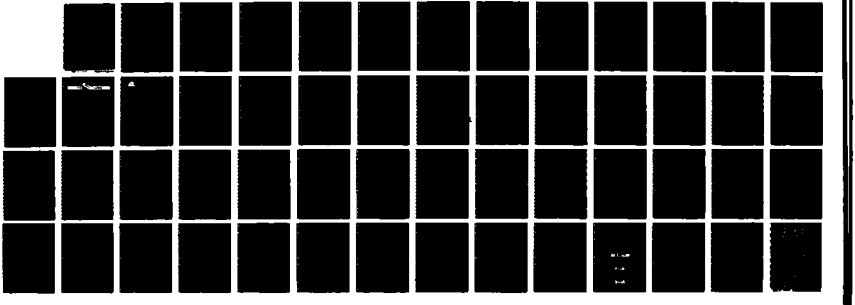
Figure 7 Distributed Arithmetic Primitive Element Architecture

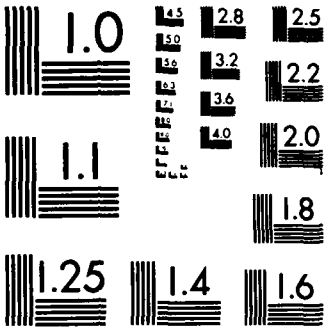
This cell can be incorporated in a mesh-connected system to perform the Givens rotation by McWhirter's algorithm. The cell is assumed to operate like the bit-sequential cell of the GAPP except that the multiplication is no longer $O(n^2)$ but $O(n)$. The latency and execution time can be made by removing the overhead estimates made for GAPP. Thus the latency for the distributed arithmetic implementation of McWhirter's algorithm is

$$(c + r + 1)(224n + 156)$$

AD-A184 603 SBNR (SIGNED BINARY NUMBER REPRESENTATIONS) DIGITAL SIGNAL PROCESSOR ARCHITECTURE(U) SPACE TECH CORP FORT COLLINS CO M ANDREWS ET AL 31 MAY 87 ARO-20192 7-EL-S 2/2

UNCLASSIFIED DAAG29-83-C-0025 F/G 12/7 NL





and the execution time is approximately

$$(2r + c + 1)(224n + 156) \qquad (11)$$

COMPARATIVE ANALYSIS

A comparison of systolic primitive elements is presented in Table 2. Five architectures are examined.

a. Conventional binary bit sequential cell (GAPP)
b. Conventional binary (complex cell)
c. Distributed arithmetic
d. Signed binary number representation (complex cell)
e. Signed binary number representation (mesh-connected PE's)

The architectures were contrasted assuming a square-root free Givens rotation implementation to obtain speed and latency estimates. The conventional binary (complex cell) and SBNR (complex cell) are both algorithm dedicated architectures. As a result, they have irregular structures and are not VLSI amenable. The conventional binary (complex cell) architecture has O(1) speed and latency. The SBNR (complex cell) and the SBNR (mesh-connected PE), which, by the way, is VLSI amenable, have O(n) and O(1) speed and latency, respectively. The distributed arithmetic architecture exhibits O(n) speed and latency.

The conventional binary (complex cell) is superior in terms of speed and latency only. The distributed arithmetic and SBNR (mesh-connected PE) architectures have excellent speed and latency and both are VLSI amenable. Distributed arithmetic bandwidth is smaller than SBNR (mesh-connected PE) bandwidth, however, SBNR (mesh-connected PE) has a superior latency. For adaptive beamformer implementation, both architectures are viable.

Table 2 Comparison of Systolic Array Architectures

	Complexity and Performance Cell Type		
	Conventional Binary Bit Seq. Cell (GAPP)	Conventional Binary (1 Adders)[1] (3 Multipliers)	Distributed Arithmetic
Speed	$(2r+c+1)(83n^2+224n+156)$	$3(2r+c+1)$	$(2r+c+1)(224n+156)$
Latency	$(r+c+1)(83n^2+224n+1)$	$3(r+c+1)$	$(r+c+1)(224n+1)$
Cell Complexity	Simple	Complex	Simple
I/O Bandwidth	c	cn	c
VLSI Amenable	Yes	structure irregular	Yes
Algorithm Dedicated	No	Yes	No
Gate Counts	--	--	--

1. For Boundary Cell. Internal cell requires 2 Adders, 2 Multipliers. Final cell requires 1 Multiplier.

	SBNR (Complex cell)	SBNR (mesh-connected PE's)
Speed	$(2r+c+1)(20+n)$	$0(n)$
Latency	$20(r+c+1)$	$0(1)$
Cell Complexity	Complex (multiple SBNR PE's)	Simple
I/O Bandwidth	2c	2c
VLSI Amenable	structure irregular	Yes
Algorithm Dedicated	Yes	No
Gate Count	$rc(2768 + 71\log_2 w + 42w + 64n)$	$rc(187 + \log_2 w + 6w)$

Table 2 Notation:

r - rows of rectilinear matrix
c - columns of rectilinear matrix
n - word length

REFERENCES

1. P.L. Feintuch, F.A. Reed, N.J. Bershad, and C.M. Flynn, "Final Report on Phase 3 of the Adaptive Tracking System Study," DTIC Distribution FR81-11-70, 1980.
2. J.G. McWhirter, "Recursive Least-Squares Minimization Using A Systolic Array," SPIE, pp. August, 1983, pp. 106-112.
3. M. Andrews, "A Systolic SBNR Adaptive Signal Processor," IEEETC&S, Vol. CAS-33, No. 2, February 1986, pp. 230-238.
4. J.L. Barlow and I.C.F. Ipsen, "Givens Rotation for the Solution of Linear Least Squares Problems on Systolic Arrays," Penn. State Univ., Univ. Park, PA, 1985.

VLSI-MVL IMPLEMENTATION OF A FAST ARITHMETIC CELL WITH SBNR

David M. James
Space Tech Corporation
215 East Oak Street, Suite 2
Fort Collins, CO 80526

Published at the
Vail Workshop
Vail, Colorado
February, 1987

ABSTRACT

To meet the demands of high-speed si within the size constraints of VLSI implementati rithmetic processing element is described. This elemen gned Binary Number Representation (SBNR) to achieve fully el addition. The result is an adder, suitable for use in V applications, whose throughput is independent of word length. Use of SBNR also reduces intrachip connection area, thus allowing a higher device density to give each chip correspondingly greater processing power.

INTRODUCTION

While the demand for faster, more powerful signal processors has increased, the space allotted them has decreased. A need exists, therefore, for very fast arithmetic circuits that can be very densely integrated. This requires that the circuits designed consume little power and require little area for devices and interconnections. In addition, a system should be found that allows fast arithmetic processing with little added chip area.

Conventional adders add two numbers from lowest digit-place to highest, propagating carry down the length of the number. Thus, the length of time for the addition depends on the word length. This paper describes an adder that meets the requirements for VLSI implementation. The adder uses signed binary number representation (SBNR) to achieve totally parallel, carry-free addition. The logic circuits used were developed to allow VLSI implementation of multiple valued logic. The adder is compared to conventional designs to show a speed increase for various operand lengths.

ADDITION IN SIGNED BINARY NUMBER REPRESENTATION

A redundant number representation is one in which several different digit combinations can represent the same number. The

This work was sponsored in part by Army Research Office Contract #DAAG29-83-C-0025. The views, opinions, and findings contained in this report are those of the author and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

signed digit number representation (SDNR)[1] uses positive and negative digits to represent a number. This special coding allows faster arithmetic, because the carry propagation distance is limited to one digit position. Standard number systems require that the carry into a digit be known before the carry out can be generated. SDNR does not require such foreknowledge, so arithmetic is performed in parallel. A signed digit number set of radix r would consist of the digits

$$x_i \in \{k-1, k-2, \cdots, 1, 0, -1, \cdots, -(k-2), -(k-1)\}, \text{ where } k \leq r.$$

This digit set represents a redundant number

$$X = x_n r^{n-1} + x_{n-1} r^{n-2} + \cdots + x_2 r + x_1.$$

When this type of number system is used in an arithmetic unit, propagation of carry is limited to two digit places. Since the carry does not propagate down the entire length of the number, an addition is done in constant time, independent of the length of the operands, n.

The radix-two subset of SDNR is called signed binary number representation--SBNR.[2] (It is also referred to as redundant binary representation.) The digits of SBNR belong to the set

$$x_i \in \{1, 0, \bar{1}\}$$

where $\bar{1}$ represents -1. For example, the binary number 1011 = $1(2^3)+0(2^2)+1(2^1)+1(2^0) = 11_{10}$ can be represented in SBNR as

$$1011 = 11\bar{1}1 = 110\bar{1}$$

Though circuit implementations of SBNR are not, in general, as simple as those of binary, they are much simpler than implementations of the higher radices. Therefore, SBNR provides a reasonable compromise between the parallel arithmetic advantages of SDNR, and the simpler structure of binary circuits.

In addition, conversions between binary and SBNR are simple. For instance, two's complement numbers are converted to SBNR merely by changing the sign of the most significant bit. Numbers in SBNR are recoded into binary by subtracting the negative bits from the positive in a conventional binary adder. The system should be designed such that any slowdown caused by the binary adder is offset by the speedup offered by using SBNR arithmetic.

Table 1 shows the addition rules for [illegible] derived from Avizienis.[3] Six types of input bit patte[illegible] identified. Notice that in all cases, the carry out is independent of the carry in. In most cases, the carry is independent of the previous bits. In the rest, the carry depends only upon the signs of the next-lower operand bits. If both bits are negative

(Neg_i=1), the carry out is the lesser of the two operand bits; otherwise, it is the greater.

For example, to add two SBNR numbers -- $1010_2 = 10_{10}$ and $\overline{1}\overline{1}11_2 = -9_{10}$ -- first the adder generates the carries, then uses the carry to find the final sum:

operand1	1010	=	10_{10}
operand2	$\overline{1}\overline{1}11$	=	-9_{10}
carry	0010		
sum	0001$\overline{1}$	=	1_{10}

Type	A_i	B_i	Neg_i (Signs of next-lower digits)	Carry Out Cy_{i+1}	Final Sum C_i
<1>	1	1	---------	1	Cy_i
<2>	0	1	$\overline{1}$ (both previous bits nonnegative)	1	$\overline{1}+Cy_i$
	1	0	1 (one previous bit negative)	0	$1+Cy_i$
<3>	0	0	---------	0	Cy_i
<4>	1	$\overline{1}$	---------		
	$\overline{1}$	1	---------		
<5>	0	$\overline{1}$	$\overline{1}$ (both previous bits nonnegative)	0	$\overline{1}+Cy_i$
	$\overline{1}$	0	1 (one previous bit negative)	$\overline{1}$	$1+Cy_i$
<6>	$\overline{1}$	$\overline{1}$	---------	$\overline{1}$	Cy_i

Table 1. Truth table for SBNR addition.

When the input is (0,1) or (0,$\bar{1}$), however, the signs of the next-lower bits must be known to generate sum and carry. Since the sum of the next-higher bit depends on the carry, and the carry depends on the next-lower bits (and not on the next-lower carry), one can see that signal propagation is limited to two bit places. This eliminates the delays of carry-ripple adders and the interconnection complexities of carry look-ahead adders.

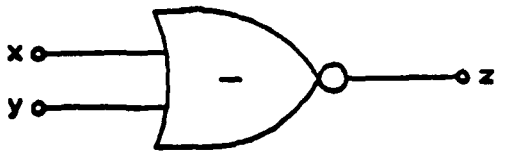
MULTIPLE VALUED LOGIC

The last decade has seen a growing interest in the computational applications of multivalued logic (MVL).[4-10] Much of the motivation for this research stems from the reduction in area and pinout offered by MVL. An increase in the number of logic levels on a single wire causes a decrease in the number of wires. This decrease reduces both the interconnection area on a chip and the number of pins required to transfer data to the rest of the MVL system.

Since there are three logic levels in SBNR (1,0,$\bar{1}$), the best MVL system to use should have three states. Those technologies in which logic levels are represented by current (I^2L, ECL) or charge (CCD) hold the greatest promise for radices higher than four. Voltage-mode logics (CMOS, NMOS, TTL) seem better suited to the lower radices.[4]

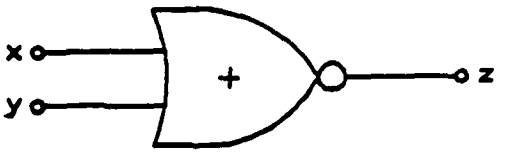
VLSI IMPLEMENTATION OF MVL

Only the low-power families (CMOS, NMOS, CCD) will allow a great enough packing density for VLSI. Of these families, CMOS circuits have received the most attention. Unfortunately, it is difficult to transfer the advantages of binary CMOS--high speed and low static power dissipation--to MVL implementations. Early designs required resistors to generate the center voltage, which slowed down the gates and increased the power dissipated. However, some of the more recent research[11-13] has succeeded in creating low-power all-transistor designs, at the cost of higher gate complexity. The circuits operate on voltages of (+5, 0, -5) volts to achieve logic levels (1, 0, $\bar{1}$).



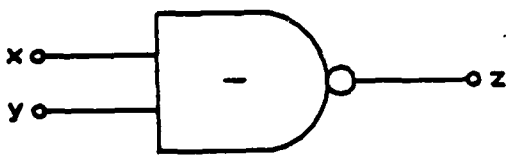
x	y	z
T	T	1
T	0	T
T	1	T
0	T	T
0	0	T
0	1	T
1	T	T
1	0	T
1	1	T

Figure 1a. Negative Ternary NOR.



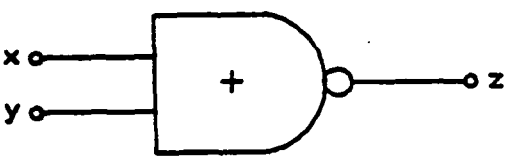
x	y	z
T	T	1
T	0	1
T	1	T
0	T	1
0	0	1
0	1	T
1	T	T
1	0	T
1	1	T

Figure 1b. Positive Ternary NOR.



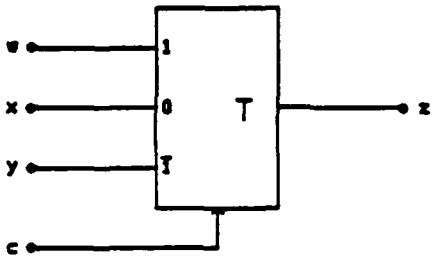
x	y	z
T	T	1
T	0	1
T	1	1
0	T	1
0	0	T
0	1	T
1	T	1
1	0	T
1	1	T

Figure 1c. Negative Ternary NAND.



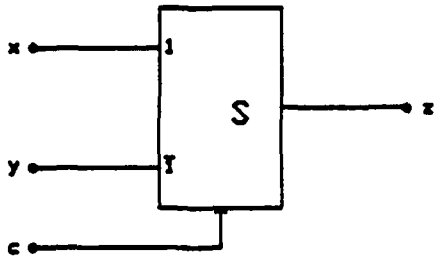
x	y	z
T	T	1
T	0	1
T	1	1
0	T	1
0	0	1
0	1	1
1	T	1
1	0	1
1	1	T

Figure 1d. Positive Ternary NAND.



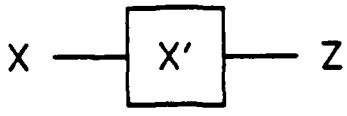
c	z
1	w
0	x
T	y

Figure 1e. T-gate.



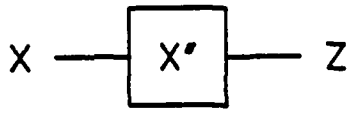
c	z
1	x
T	y

Figure 1f. S-gate.



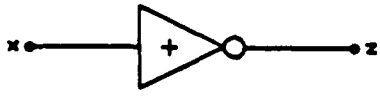
X	Z
$\bar{1}$	0
0	1
1	$\bar{1}$

Figure 1g. Positive Ternary Cycling Gate.



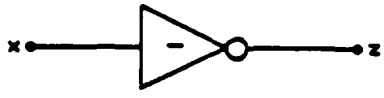
X	Z
$\bar{1}$	1
0	$\bar{1}$
1	0

Figure 1h. Negative Ternary Cycling Gate.



x	z
T	1
0	1
1	T

Figure 1i. Positive Ternary Inverter.



x	z
T	1
0	T
1	T

Figure 1j. Negative Ternary Inverter.

These gates, and their truth tables, are summarized in Figure 1. The circuit configurations of the ternary NAND, NOR and inverter are all identical to their binary counterparts. Their functional difference is a result of changing the parameters of the transistors. The T-gate (transmission gate) selects one of three inputs based on a control signal. The S-gate (switch gate) consists of two pass-transistor pairs and an inverter: when the control signal goes low, one input is passed through its shorted pass transistors to the output; when the control is high, the second set of pass transistors shorts to allow the other input through. The X' and X" gates add one and subtract one, respectively, mod 3.

These gates are combined to form a ternary adder cell which uses SBNR to perform the function

$$C_i = A_i + B_i .$$

The state signal NEG_i tells the cell when one of the next-lower digits is negative. It is the negative ternary NAND of A_i and B_i. Thus, when an operand is negative, $NEG_{i+1}=1$, otherwise, $NEG_{i+1}=\overline{1}$. This signal controls the sum and carry, as shown in Table 1. The logic diagram of the arithmetic cell is pictured in Figure 2.

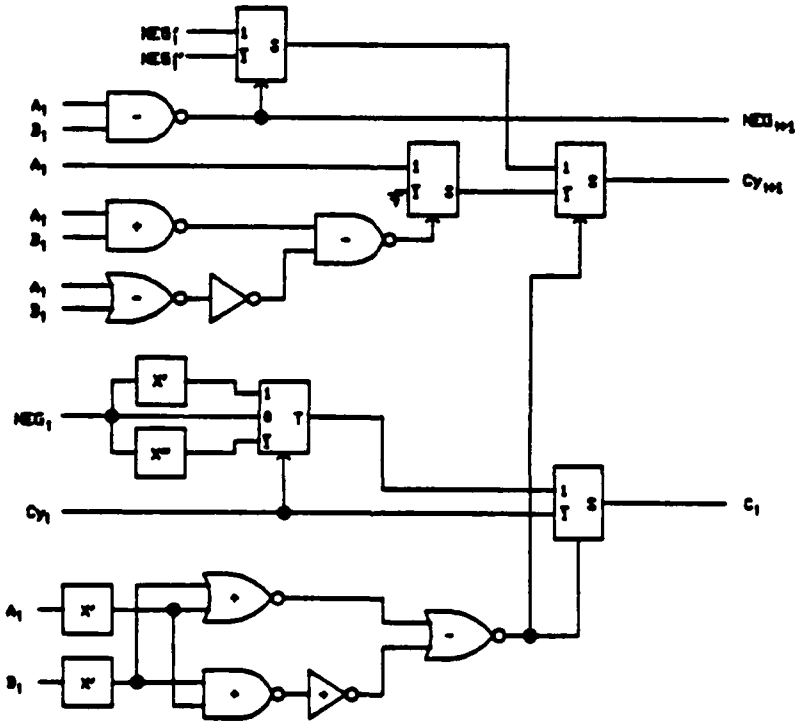


Figure 2. Ternary adder cell

The implementation of the cell requires 112 transistors with ± 5 V power supplies, and the longest signal path is 11 transistor delays, t, long. The ternary gates used have been tested using four micron design rules. In that case, each transistor delay was about 1.5 nsec. If the gates could be implemented using a gate width of one micron, each transistor delay would be about 0.1 nsec[13]. Thus the total time from the appearance of the operands to the appearance of the answer could be as low as 1.1 nsec. Since the addition is totally parallel, a row of these cells operating simultaneously could add two n-width words in the same 1.1 nsec as a single bit.

Conventional ripple-carry adders (length n) consist of n full-adder cells connected in parallel. Carry from the addition of the first bit moves to the second, where it is used to find the carry to the third. The minimum cell for this operation has a gate delay of two transistor delays.[15] Thus, the total add time is 2nt. The SBNR adder is a factor 0.18n faster than the ripple-carry adder, for n>5.

Brent and Kung[16] considered a scheme for implementing carry lookahead adders in VLSI. In this paper, they demonstrated a network that computed a length-n sum in $(4t)\log_2 n$ seconds. This setup is faster than the carry-ripple circuit, but slower than the SBNR adder by $(0.36)\log_2 n$ for n>6.

The area complexities of both the SBNR adder and the ripple carry adder are O(n) (though the SBNR adder requires about three times the area of the ripple carry adder). The area of the carry-lookahead adder is $O(n \log_2 n)$. Thus, the area-time complexities for the adders are:

SBNR - - - - - - - $O(n)$,
Ripple-carry - - $O(n^2)$,
Carry-lookahead- $O(n \log_2 n)$.

The lowest of these is the SBNR, making it the logical choice for implementation in VLSI. Furtner study is needed to determine whether the three-level system is desirable. A binary-encoded SBNR system might provide the same area-time advantages as ternary while eliminating the negative power supply.

SUMMARY AND CONCLUSIONS

Use of SBNR to eliminate carry propagation has resulted in a totally parallel adder. The adder has been shown to be faster than conventional binary implementations. A comparison of the speed increase provided is given in Table 2. In high-precision systems, use of SBNR can provide a significant increase in processor speed and system throughput.

Word Length n	SBNR Adder	Ripple Carry Adder	Carry Lookahead Adder
8	11	16	12
16	11	32	16
32	11	64	20
64	11	128	24

Table 2. Number of gate delays for various word lengths.

One disadvantage of SBNR is the added complexity of the bit-level adder circuits. Another problem is conversion back to binary. The usual method is to separate the poitive digits from the negative, and then add the two numbers in a conventional adder. This brings back the old problem of carry propagation. However, a new method proposed by Chen[17] converts an SBNR number to binary in a time proportional to the longest string of consecutive zeros in the number. This method could make the SBNR adder viable for a larger class of problems.

In a system that uses SBNR exclusively, of course, no conversion need take place. In most systems, however, custom-designing all components to operate under SBNR is too expensive. Thus, SBNR should be used only in applications where the conversion overhead is negligible compared to the computation time saved.

Such an application is an adder tree for multioperand addition.[18] In this case, a large number of operands can be added together in SBNR, converting only the final result to binary. A similar application is parallel multiplication, with parallel accumulation of partial products.[19] In this case, bit-level multiplications are performed on the operands, the partial products are added in an SBNR adder tree, and the result is converted using one carry lookahead adder.

REFERENCES

[1] Avizienis, A., "Signed-Digit Number Representations for Fast Parallel Arithmetic," IRE Transactions on Electronic Computers, Vol. EC-10, Sept. 1961, pp. 389-400.

[2] Andrews, M., "A Systolic SBNR Adaptive Signal Processor," IEEE Transactions on Circuits and Systems, Vol. CAS-33, Feb. 1986, pp. 230-238.

[3] Harata, Y., Y. Nakamura, H. Nagase, M. Takigawa, N. Takagi, "High Speed Multiplier Using a Redundant Binary Adder Tree," Proceedings--IEEE ICCD '84, Oct. 1984.

[4] Smith, K.C., "The Prospects for Multivalued Logic: a Technology and Applications View," IEEE Transactions on Computers, Vol. C-30, Sept. 1981, pp.619-634.

[5] Current, K.W., D.A. Freitas, F.A. Edwards, "CMOS Quaternary Threshold Logic Full Adder Circuits," Proceedings--15th International Symposium on Multiple-Valued Logic, 1985, pp. 318-322.

[6] Mouftah, H.T., A.N.C. Heung, L.M.C. Wong, "QTC-1: a CMOS Ternary Computer," Proceedings--14th International Symposium on Multiple-Valued Logic, 1984, pp.125-132.

[7] Current, K.W., L.B. Wheaton, T.M. Luich, D.A. Mow, "Characteristics of Integrated Quaternary Threshold Logic Full Adders," Proceedings--10th International Symposium on Multiple-Valued Logic, 1980, pp.24-30.

[8] Soderstrand, M.A., R.A. Escott, VLSI Implementation in Multiple-Valued Logic of an FIR Digital Filter Using Residue Number System Arithmetic," IEEE Transactions on Circuits and Systems, Vol. CAS-33, Jan. 1986, pp. 5-25.

[9] Kameyama, M., T. Higuchi, "Design of Radix 4 Signed-Digit Arithmetic Circuits for Digital Filtering," Proceedings--10th International Symposium on Multiple Valued Logic, 1980, pp. 272-277.

[10] Current, K.W., "High Density Integrated Circuit Computing Circuitry with Multiple Valued Logic," IEEE Transactions on Computers, Vol. C-29, Feb. 1980,pp. 191-195.

[11] Balla, P.C., A. Antoniou, "Low-Power-Dissipation MOS Ternary Logic Family," Proceedings--14th International Symposium on Multiple Valued Logic, 1984, pp.133-139.

[12] Huertas, J.L., J.M. Carmona, "Low-Power Ternary C-MOS Circuits," Proceedings--9th International Symposium on Multiple Valued Logic, 1979, pp.170-174.

[13] Mouftah, H.T., A.I. Garba, "VLSI Implementation of a 5-Trit Full Adder," IEEE Proceedings, Vol. 131, Pt. G, Oct. 1984, pp.214-220.

[14] Mead, C., L. Conway, ed., Introduction to VLSI Systems, 1980.

[15] Lai, H.C., S. Muroga, "Minimum Parallel Binary Adders with NOR (NAND) Gates," IEEE Transactions on Computers, Vol. C-28, Sept. 1979, pp.648-659.

[16] Brent, R.P., H.T. Kung, "A Regular Layout for Parallel Adders," IEEE Transactions on Computers, Vol. C-31, Mar. 1982, pp.260-264.

[17] Chen, C., "Maximal Redundancy Signed-Digit Systems," Proceedings--7th Symposium on Computer Arithmetic, 1985, pp. 296-300.

[18] Atkins, D.E., S.C. Ong, "Time-Component Complexity of Two Approaches to Multiperand Binary Addition," IEEE Transactions on Computers, Vol. C-28, Dec. 1979, pp.918-926.

[19] Takagi, N., H. Yasuura, S. Yajima, "High-Speed VLSI Multiplication Algorithm With a Redundant Binary Addition Tree," IEEE Transactions on Computers, Vol. C-34, Sept. 1985, pp. 789-796.

THE OPEN CHANNEL

Processor capability for hardware implementation of Kalman filters*

A Kalman filter is a set of *nxm* matrices and *n* vectors that compute equations similar to those in the first six equations below.

$$\hat{x}(t+1) = A\hat{x}(t) + K(t)[y(t) - C\hat{x}(t)] + Bu(t)\ \hat{x}(t_0) = \hat{x}_0$$

where $K(t)$ is the filter gain given by

$$K(t) = [A\Sigma(t)C^T + S][C\Sigma(t)C^T + R]^{-1}$$

$\Sigma(t)$ is the state error covariance, that is,

$$\Sigma(t) \triangleq E[[\hat{x}(t) - x(t)][\hat{x}(t) - x(t)]^T | y(t-1), y(t-2), \ldots, y(t_0)]$$

$\Sigma(t)$ satisfies the following Riccati difference equation:

$$\Sigma(t+1) = A\Sigma(t)A^T + Q - K(t)[C\Sigma(t)C^T + R]K(t)^T\ \Sigma(t_0) = \Sigma_0$$

Several published papers imply that these equations can be computed in *n*log*n* time using array processors.[1,2,3,4] Such works present only a partial picture. The larger question is, "How many processors are needed?" The most serious concern is, "What is the complexity of each processor?"

A reasonable figure-of-merit (FOM) should be

$\Omega =$ [number of computational steps] *[number of processors] *[processor complexity]

This treatise only addresses the processor complexity issue, since other papers have sufficiently studied the remaining two factors. Claims made for an O[*n*log*n*] bound to the number of computational steps assume a large array of processing elements, or PEs. In fact, *nxm* PEs are required as a minimum. This reputed bound does not take into account the attendant communications and control circuitry of the array processor.

The complexity of these O[*n*log*n*] type PEs is equivalent to that of a 68000 microprocessor, which has 70K transistors, multiplies (16 × 16) in 6.75 microseconds and adds in 1 microsecond. No conventional processor has several 68000 chips on a single substrate; such devices are optimistically forecasted for 1995. At present, only one commercial array processor chip is available (NCR's GAPP, #NCR45CG72). This state-of-the-art array is a 6 × 12 set of PEs with one-bit (not eight or 16) ALUs. As an algorithm designer for this chip, this writer has found no software tools available, no test circuits, and no emulators. A study of support tools (software and hardware) for any array processor and a thorough comparison of SIMD (single instruction multiple data), MIMD, array processors and data flow machines (which are SIMD-like) are desperately needed.

The brief comparison herein shows that the current multiprocessor architectures (e. g., the WSMR-DCM[5]) are still superior. This is primarily because hardware mult/div/add cycles are 400 times faster than any available or conjectured array processor PE. The DCM performs a 16 × 16 multiplication in 220 nsec (compared to 6.75 microseconds required by the equivalent PEs necessary to do Kalman filtering). Futhermore, a support environment with hardware/software already exists. An *nxm* Kalman filter would be executable in approximately $n^2 \log n$ steps in an architecture such as the DCM. Assuming, then, that the processor complexity is equivalent, the FOM for each approach (systolic array versus DCM) is

$$FOM_{DCM} = [n^2 \log n][1][1]$$

while the systolic approach reported by Kailath, Kung, and others is

$$FOM_{array} = [n \log n][n^2][1]$$

Hence,

$$FOM_{array} = n \text{ times } FOM_{DCM}$$

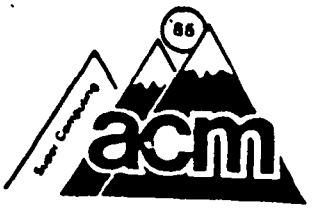
Since no software environment exists for any array processor, the DCM-like architectures remain clearly superior.

1. J. M. Cioffi and T. Kailath, "A Classification of Fast Fixed-Order RLS Algorithms," 1984 Digital Signal Processing Workshop, Chatham, Mass. Oct. 1984, pp. 1.1.1-1.1.2.
2. W. Hsu, Y. J. Leung, M. A. Shanblatt, "Simulating VLSI Systolic Array Structures for Fast Matrix Triangulation," 13th Annual Pittsburgh Conf. on Modeling and Simulation, Pittsburgh, Pa., Apr. 1982.
3. A. L. Fisher, H. T. Kung, and K. Sarocky, "Experience with the CMU Programmable Systolic Chip," *Proc. SPIE Symp. Real-Time Signal Processing VII*, Vol. 495, Aug., 1984.
4. H. T. Kung and O. Menzilcioglu, "Warp: A Programmable Systolic Array Processor," *Proc. SPIE Symp. Real-Time Signal Processing VII*, Vol. 495, Aug. 1984.
5. M. Andrews and R. E. Boring, "Firm ware Engineering for Cascadable Microcomputer Module," Space Tech Corporation Final Report for Battelle Columbus Laboratories, TR#BAT-1171-84-WSMR, Dec. 1984.

Michael Andrews
Space Tech Corporation
2324 Manchester Court
Fort Collins, CO 80526

*This study was sponsored by the U.S. Department of the Army under Contract #DAAD07-84-C-0139. The views, opinions, and findings contained in this paper are those of the author.

The Open Channel is a forum for the exchange of technical ideas. We welcome all contributions (short of libel or obscenity), but please double space them and keep them to a maximum of 1000 words. Send everything to Jim Haynes, Computer Center, UC Santa Cruz, CA 95064.



Concurrency and Parallelism - Future of Computing

M. Andrews and J. S. Walicki *

Abstract: This paper discusses parallelism and concurrency in the light of current computing practices. A special case of SIMD machines, also known as systolic arrays, is analysed. A new architectural engine, the GAPP systolic array, is studied in the application domains of signal and image processing. Also included are database and associative processing cases. Some interesting conclusions can be drawn from a PE which can also be viewed as an intelligent memory.

1 Introduction

The computer age is very short but rather turbulent. No other field of technology has been changing so dramatically. Increases in speed and efficiency of computing are unprecedented. Computers of the first two generations of computer evolution (1940's to early 1960's) were primarily used for data processing. Computing strategies applied to that type of processing were simply extensions of traditional human processing of data. That is, the processing was sequential in nature, but relatively fast due to increases in speed of execution. This human oriented model of computing was clearly reflected in Von Neumann's definition of a digital computer.

However, data intensive applications, like weather analysis/prediction and various aspects of signal processing, pushed the classical computer architecture to speed limits imposed by an existing technology.

*M. Andrews is with Space Tech Corp., Fort Collins, CO 80526. J. Walicki is with Computer Science Dept., Colorado State University, Fort Collins, CO 80523.

This study was sponsored by the U.S. Army Research Office under grant DAAG29-83-C-0025. The U.S. Government assumes no responsibility for the information presented.

Therefore, it is not surprising that the future of computing lies in increased exploitation of concurrency in computing tasks. Concurrent computation implies these, not necessarily disjoint, classes of activities:

parallel ocurring in different resources in the same time interval,

simultaneous taking place at the same moment,

pipelined activated in overlapped time frames.

The highest level of parallel activities takes place among multiple tasks/programs. This level requires the development of parallel processable algorithms which depends on the efficient allocation of limited resources to individual tasks. The next level of parallelism may occur among procedures or program segments within the same program which requires decomposition of a program into multiple tasks. The next lower level exploits concurrency among multiple instructions. Such a concurrency is revealed by data dependency analysis. Additionally, vectorization of sufficiently large scalar operations can be performed. These three levels of parallelism are most often dealt with in software. The last and the lowest level of parallelism is concerned with concurrency of operations within each instruction, and it is usually implemented in hardware We can identify several kinds of parallelism even in single processor systems. Parallelism can be exploited, or induced, by using multiple functional units. For example, CDC-6600 has 10 functional units performing arithmetic and logic operations. The units are independent and can operate in parallel (Baer80, Hwan84). Parallelism within the CPU is achieved, in both large processors and modern microprocessors, by overlapping (pipelining) the fetch, decode and execute phases. Finally, Overlapping of CPU and I/O operations can be performed by using separate I/O controllers.

In this paper we want to examine an SIMD architecture which holds promise of a new threshold of computer architectures which will impact the marketplace for some time. This architecture is configured about a VLSI primitive cell of 72 processing elements regularly organized. The term 'primitive cell' is a misnomer since the PE contains 72 individual ALU's. We conjecture that this VLSI cell is the forerunner of numerous offspring with even greater singularized computational power Before we can place this novel device in the spectrum of modern processors it is necessary to present existing taxonomies of digital processors.

.1 Parallel architectures

'he basic architectural classes of parallel machines are

- pipeline computers
- array processors
- multiprocessor systems

\ pipeline computer performs overlapped computations to exploit **temporal parallelism**. An *array processor* achieves **patial parallelism** by using *synchronized arithmetic logic* units. A *multiprocessor system* achieves asynchronous parallelism in a set of interacting processors with shared resources Hwan84). The above classification is by no means perfect. New developments in the area of systolic systems blur distinction among all three types of parallel systems.

Different taxonomies arise depending on the primary feature chosen to distinguish among different architectures. Three major classifications are those of Flynn (Flyn66), Feng (Feng77) and Händler (Händ77).

1.1.1 Flynn's classification

This historically first classification is based on the interaction of instruction and data streams. The stream is a sequence of items (data or instructions) operated upon by a single processor. This leads to classes of structures determined by the multiplicity of the functional units devoted to servicing the instruction and data streams. Flynn identifies the following four types of machines:

SISD *single instruction stream-single data stream*

SIMD single instruction stream-multiple data stream

MISD *multiple instruction stream-single data stream*

MIMD multiple instruction stream-multiple data stream

SISD architecture is the simplest, composed of a single processing unit (PE), a control unit (CU) and a memory module (MM). This is also the structure of the large percentage of the existing machines. Many of the uniprocessors (**SISD** machines) are pipelined, that is, instructions are overlapped in their sequential execution. Exemplary machines with one functional unit are: IBM 7090, PDP VAX11/780. Examples of existing **SISD** machines with multiple functional units are: IBM 360/91, IBM370/168UP, CDC 6600, CDC Star-100, FPS AP-120B, Cray-1, CDC Cyber-205.

SIMD machines have multiple processing elements governed by the same control unit. The processors respond to the same instruction stream but usually operate on different data subsets. The memory is shared and usually contains several memory modules. This class of machines is exemplified by array processors and systolic arrays. The following machines belong to this class: Burroughs' Illiac-IV and BSP, Staran, MPP (massively parallel processor by Goodyear Aerospace).

The **MISD** organisation has not been implemented in practice and it is included in this classification for the sake of completeness. In the **MISD** concept n processors, controlled by distinct instruction streams, operate on the same data stream. The output of one processor becomes the input to the next processor, and therefore, this scheme can be efficiently realised in the **SISD** structure.

Finally, the **MIMD** organisation is the most challenging and the most promising in terms of speed and efficiency. The proper **MIMD** architecture consists of closely interacting processors which operate on the data streams derived from the same data space shared by all processors. It is possible to envision the system in which processors operate on independent data sets. This type of organization is called *multiple* **SISD** since it is a set of independent **SISD** machines. Examples of closely coupled machines of this kind are: C.mmp, Cray-2 and Cray XMP, Denelcor HEP, Burroughs D-825.

1.1.2 Feng's classification

The classification proposed by Feng is based on the degree of parallelism. If P_i is the number of bits processed within the *ith* cycle of total T processor cycles, then the *average parallelism degree* is

$$P_a = \frac{\sum_{i=1}^{T} P_i}{T}$$

Since, in general, an average parallelism degree is less then the hypothetical maximum parallelism degree P, the *utilization rate* σ can be defined as

$$\sigma = \frac{P_a}{P} = \frac{\sum_{i=1}^{T} P_i}{T * P}$$

The maximum degree of parallelism P is equal to the product of the *wordlength* w and the bit-slice length m. A size of the bit-slice is determined by the number of bits that can be processed by a system in the same instance. For example, a processing unit has two four-stage pipelines, which yields 8 bit-slice. In Feng's taxonomy the relationship between the wordlength and size of the bit-slice determines a machine class. Machines like Staran and MPP have a short wordlength and very long bit-slices. On the other side of the spectrum are machines with a relatively long wordlength and short bit-slice. IBM 370/168, PDP-11 and Cray-1 belong to this class. According to Feng's classification particular mixtures of the bit-slice size and the wordlength give rise to four classes of processing methods, which are listed below.

WSBS word-serial, bit-serial

WPBS word-parallel, bit-serial

WSBP word-serial, bit-parallel

WPBP word-parallel, bit-parallel

The first category (**WSBS**) includes the first generation computers with bit-serial arithmetic. Most contemporary machines are of that kind, also called word-slice computers. **WPBP** signifies the fastest, fully parallel processing in which whole blocks of bits are processed at a time.

1.1.3 Händler's classification

This classification is based on the degree of parallelism and pipelining present in the hardware of a computer. Three hardware subsystems are considered:

- control unit (CU)
- arithmetic logic unit (ALU)
- bit-level circuit (BLC - serial combinational circuitry in ALU)

A computer C can be assigned the figure of merit $T(C)$:

$$T(C) = (K \times K', D \times D', W \times W')$$

where

K = the number of CPU's

D = the number of ALUs (or PEs) under control of a CU

W = the word length of an ALU

W' = the number of pipeline stages in all ALU's

D' = the number of ALUs that can be pipelined (chaining)

K' = the number of CUs that can be pipelined (macropipelining)

The Händler's taxonomy can be explained on the example of the C.mmp multiprocessor system developed at Carnegie-Mellon University. The C.mmp consists of 16 PDP-11 minicomputers, shared memory modules and 16 by 16 crossbar interconnection network. The system is unique because it can operate in various configurations. The normal mode of operation is the MIMD. However, under control of synchronizing unit it can operate in the SIMD mode, and if all processors are cascaded so that they operate on the same data stream the MISD mode results. All three operating modes can be characterised in the following way

$$T(C.mmp) = (16, 1, 16) + (1 \times 16, 1, 16) + (1, 16, 16)$$

2 Geometric Regularity and Systolic Systems

Regularity of computer structures is appealing not only for aesthetic reasons, but it also has distinct functional advantages. This fact was recognised early in the computer era when the concept of cellular interconnection arrays was developed for the purpose of data switching. Processing elements of these arrays were relatively primitive and contained on the order of ten elementary gates. Such limited processing capabilities were not overly restrictive as long as the main goal was switching of data. Developments in the field of VLSI and increased interest in networking resulted in the new category of systems which combine concepts of parallelism, pipelining and interconnection. These are so called systolic systems in which processors compute and transmit data in the synchronised fashion. Systolic systems are mature cellular systems in which elementary processing units ('E's) are relatively complex. The PE's are usually placed in the nodes of the simple one- or two-dimensional grid. Tight coupling and pipelining ability of the systolic systems result in constant-time processing. A more rigorous description of a systolic system follows (Leis83).

A systolic system is a synchronous network of processors. Each processor is composed of a constant number of Moore machines (state-output FSMs) which are defined by the quintuple (Q, I, O, θ, ρ) where

Q - set of internal states

I - set of input symbols

O - set of output symbols

θ - state transition function

ρ - output function: $o(t+1) = \rho(q(t+1))$

In pure systolic systems only Moore machines are allowed. Inclusion of Mealy machines produces *semi*-systolic systems. If Mealy machines were included and connected together, then logic signals could ripple through several machines in one clock period. The exclusion of Mealy machines is important because it guarantees that the clock period does not grow with system size. Thus a clock period becomes a measure of time which is independent of system size. The structure of a systolic system S is given by a machine graph G = (V,E), where V is a set of Moore machines and E is a set of edges linking the machines. The neighborhood of a machine $v \in V$ is the set of machines with which it communicates:

$$Neigh(v) = \{w | (v, w) \in E or (w, v) \in E\}$$

For S to be systolic, it it required that the Moore machines be small in the sense that Q_v, I_v, O_v and Neigh(v) are bounded, which implies that the system S may grow, but individual processors may not! It is important to preserve inequality between a propagation delay between processors and a processor delay. For that reason the systolic systems with only nearest neighbor connections are especially attractive since a propagation delay is insignificant. Therefore, the global communication in systolic systems should be avoided as it imposes difficult timing restrictions and contributes to additional circuit complexity and area.

On the other hand, the global communication is convenient because it provides an efficient way for initializing PEs (broadcasting) and gathering status information. Fortunately, *the systolic conversion lemma* due to Leiserson allows one to design semi-systolic systems (with broadcasting) and implement equivalent systolic structures.

2.1 Application of Systolic Systems

Computational tasks are usually are compute-bound or I/O-bound. In general, if the total number of operations is larger than the total number of input and output elements, then the

computation is compute-bound. The systolic systems solve efficiently compute-bound problems. A large spectrum of problems has been attacked using the systolic approach. The following is a brief list of major types of applications:

Matrix arithmetic:

- matrix-vector multiplication
- matrix-matrix multiplication
- matrix triangulization (solution of linear systems)
- QR decomposition (eigenvalues, least-square computations)
- solution of triangular linear systems.

Non-numeric applications:

- data structures (stacks, queues, priority queues, searching and sorting)
- graph algorithms
- language processing (string matching, regular expressions)
- dynamic programming
- encoders (polynomial division)
- relational data-base operations.

Finally, enormously important class of real-time applications contains primarily digital signal processing tasks. It is worth noting that the systolic systems allow real-time (or near that) implementations of powerful signal processing algorithms (LMS and Kalman adaptive filtering). Some of the signal processing applications are (Urqu84, Cann84, Ward84, Spei83):

- FIR, IIR filtering
- 1D and 2D FFT
- 1D and 2D convolution and correlation
- median filtering
- adaptive filtering

The next section presents the existing 72-processor systolic array and discusses practical aspects of its utilization.

3 GAPP – bit-level systolic array

The GAPP - Geometric Arithmetic Parallel Processor (NCR45C may be considered to be the forerunner of many new systolic processing elements. However, it is the first device to recognise the value of bit-level implementations. Any systolic array, in order to be effective, should have

- complete and very regular PE's
- as many PE's on a chip as possible
- cell arithmetic functions which can be performed in one cycle
- nearest neighbor communications only

The GAPP is just such a chip. Its cell features are illustrated in Fig. 1. Here , we see 4 PE's with nearest neighbor coupling and a single global broadcast line. These lines reduce the VLSI interconnect space to a realistic amount.

Of particular note is the ability of the GAPP device to be cascaded. That alone makes the GAPP significantly powerful. Several GAPP applications have been reported which cascade multiple GAPP devices in image,signal,and data processing tasks. Furthermore, each PE has an autonomous 128-bit RAM. If cascaded properly, a GAPP array can not only process, but also perform frame buffering possibly at video rates. This is important to many image processing applications where frame buffering is required. In that case, the cascaded GAPPs store as well as process,both simultaneously. In practice, a designer should view a GAPP array as intelligent memory. This then opens the design space to even larger opportunities.

Each processing element in the GAPP array consists of a bit-serial ALU, 128 x 1 individually addressable RAM and 4 single bit latches. The I/O latch allows communication through the PE without interrupting the ALU, and the remaining latches hold inputs to the ALU. The GAPP operates as a SIMD machine, that is instructions are broadcast to each cell from an external control store, loaded in turn from the host computer. Proper address sequencing can be provided by any general address sequencer. The instructions directed to the processing elements consist of a 12-bit control field which specifies the array connectivity and arithmetic/logic operations, and a 7-bit RAM address. These instructions can be sent to the GAPP array at the rate of 10 MIPS. The whole array has a global broadcast of input data and a global output. One GAPP array device can function as a modular component in a larger array, thus enabling word or bit-length growth of an array as needed. Computations are performed in bit-serial arithmetic. All primitive arithmetic operations execute in a single processor cycle. Each processor accepts data from RAM, from each of its four nearest neighbors, or from constant data provided by the instruction. A carry latch allows the extension of bit-serial computation to user defined fixed length words (Gapp84a).

From the point of view of the interconnection network taxonomy the GAPP device can be described as having the synchronous operation mode, centralized control strategy, circuit switching methodology, and regular and static network topology. In Flynn's classification it is a SIMD device. It can be classified as a word-parallel, bit-serial (WPBS) machine according to Feng's taxonomy, and the Händler's figure of merit is

$$T(GAPP) = (12 \times 1, 12 \times 12, 1 \times 1)$$

3.1 Applications of GAPP Processor

3.1.1 Adaptive Filtering

Classical techniques aimed at increasing a signal-to-noise ratio (SNR) usually employ information derived from a single signal sensor. No additional information is provided, other than

the signal and perhaps statistical properties of both signal and noise. However, chances for the restoration of the original signal can be increased if multiple measurements of the signal wave are available. An array of sensors provides such an opportunity. The array generates parallel streams of data and the information in this form can be processed using the GAPP arrays. Processing of signals arriving from the sensor array can be accomplished most efficiently if processing method is adaptive in nature. The least mean square (LMS) algorithm is such a method. It is possible to partition the LMS adaptation rule into the basic operations suitable for the processor array implementation (Andr85c).

Operation 1: weight input vector $\mathbf{w}^T\mathbf{x}$
Operation 2: sum partial products to get $y(k)$
Operation 3: compute error $e(k)$
Operation 4: scale error $2\Delta_e e(k) = C$
Operation 5: update weights $\mathbf{w} \leftarrow \mathbf{w} + C\mathbf{w}$

It can be seen from the above list of operations that they can be grouped into two distinct phases – compute and update.

The basic processing steps listed above require a variety of elementary matrix operations. Operation 1 requires multiplication of two vectors. Operation 5 calls for multiplication of a vector by scalar and for addition of two vectors. Operations 2, 3 and 4 require operations on scalars. The values to be operated on are represented by k-bit words. Existing processor (systolic) arrays posses a serial architecture, mainly because it is still prohibitively expensive to build fully parallel single-wafer multi-processor array. Therefore, the operations mentioned above have to be performed at the bit level.

Efficient use of the processor array is important in order to make up for losses caused by the use of serial arithmetic. As Urquhart and Wood (Urqu84) show the array utilization depends on properly feeding the elements to be processed. Particularly, if one matrix of arguments is kept static on the processor array and the other matrix is entered properly skewed, then the array is 100% efficient. In our case it is quite natural to keep the coefficients w fixed in the array, while bit-streams of input samples x march in from the array sensors. The arrangement described above is a basis for the implementation of the first operation, namely formation of the inner product $\mathbf{w}^T\mathbf{x}$. In the next step partial products are summed yielding an output sample y. In the third step the error sample is generated by subtracting (in a single adder) the y stream from the d stream. The d stream is a serialized sample of a reference signal. The obtained value of the error is scaled in the next operation (4). This is accomplished easily by properly shifting e(k). The last step involves multiplying a vector by a scalar and addition of vectors (bit matrices). The first operation in this step is the multiplication of the weight vector w by the scaled error e(k). Since all weights are scaled by the same scalar value it amounts to shifting original weights by a number of places determined by the value of $2\Delta_e e(k)$. The last step requires adding the old weights to the scaled values obtained in the previous operation. These steps complete the update phase and a new input sample is processed as in the first step.

If the least significant bits of x and w interact first then partial products of equal significance leave the edge of the array at the same time. These partial products can be accumulated using the adder tree constructed from MSI adders. If, however, the least significant bit x_j^0 interacts with the most significant bit of w then the partial products of equal significance appear skewed and then the linear chain of full adders can be used to accumulate the final output. If the second option is chosen then a single row of PE's of a second processing array can be used to accumulate partial products.

The second array implements operations 2, 3 and 4, that is the computations of the output sample y(k) and the scaled error $2\Delta_e e(k)$. These partial products are accumulated in the elements of the first row. Because of the additional skewing of the partial products leaving the first array, the delay is needed between the accumulating PE's. In the GAPP device this can be easily accomplished by storing the elementary sums in the local RAM locations. An error value e(k) is computed by subtracting the output sample from the reference sample d(k) which is shifted into the fourth row of the array simultaneously with the loading operation of the third row. The result of this operation remains in the fourth row and is shifted according to the value of $2\Delta_e$. The quantity obtained in this step is the scaled error value and it is sent to the host controller. The controller uses this value to scale (shift) the original coefficients residing in the main array. However, before this operation is performed the original weights must be saved in the RAM locations. The scaled weights are also uploaded into the local RAM, and then both quantities are summed ($\mathbf{w} \leftarrow \mathbf{w} + 2\Delta_e \mathbf{w}$) yielding the new updated coefficients, which are used to compute a new output sample and the process is repeated.

3.1.2 Hardware Database Machine

A database machine should have the ability to:

- support simple and complex queries
- provide JOIN and SEMI-JOIN macros
- order data rapidly
- invoke fixed and variable length record format

If the GAPP is cascaded as a set of basic building blocks as shown in Fig. 2. The aggregate system forms an efficient and regular database machine. Much of the normal software operations are handled directly in hardware. The comparator block accepts an input comparand or Record B as 6-bit wide words on the CMS lines. Input record A is entered on the S (south) lines. The comparand is stored in EW registers. Any time the data stream matches the comparand the GO (global output) line goes high. At 5 million characters per second, this exact match operation searches text files for specific words at blinding speeds.

3.1.3 GAPP as Associative Device

An associative memory operates on the basis of matching the contents not the address of the information. The associative search can be accomplished efficiently if all memory cells

are checked for a desired information in *parallel*. The associative (or content addressable) memory can be built from GAPP devices (Wall84).

The associative memory unit consists of an associative array and an associative array controller unit. The associative array is composed of cells, each with a tag bit. If the tag of a cell is set then the cell is a responder. Each cell holds one entry. The cells perform specific functions - compare, read and write. The control unit has two important registers - comparand and mask registers.

The GAPP devices combined with an appropriate control structure form a complete associative array. The associative array consists of a cascaded array of GAPP devices. Each element of GAPP is utilised as a bit-serial associative cell. Since each element is serial in nature, multibit words are emulated by using the local 128-bit RAM which also serves as storage for an intermediate operations. The NS (north-south) register performs various data functions and stores the tag bit. The tags are combined and form the Global Output signal providing a responder signal to the control unit. I/O functions are performed through the CM bus. Input data is usually loaded as word-serial, bit-parallel but operation of the array is word-parallel, bit-serial, therefore, the *corner turning* operation on data must be performed which could be done using GAPP devices or specialised circuitry. Global input to every cell in the array is easily accommplished by using the op code lines to command the C register to load 1/0.

Control routines for basic functions (compare, write, read) are written in the register-transfer like language. From these basic function more complex operations can be build, like 'exact match', 'limit search' and 'maximum value search'. In this specific implementation, an 8 bit 'exact match' search of entire array is performed in 4.4 microseconds, and 8 bit read of a responder can be accomplished in 3.2 microseconds.

3.1.4 Image Manipulation

The GAPP device was originally conceived for the purpose of image processing so it is not surprising that the most successful applications of the GAPP are in that domain (Gapp84b). The GAPP architecture is radically different from convential image processing structures. These systems required a frame buffer to store an image, a high speed image processor, and another frame buffer for storing the processed image. The memory-processor transmission bandwidth limits throughput of conventional systems. GAPP deals with this problem by providing one processor per pixel. Thus, all pixels can be processed in parallel. The local processor RAM offers the possibility of eliminating frame buffers entirely; instead, sufficiently long serial-to-parallel shift registers on input and output sides of the processing array can be used. The SIPO registers may be also built from a row of GAPPs. The shift registers are long enough to contain one full video line which is shifted into the GAPP array during the horizontal retrace. Bits of each pixel are stored during each cycle in the local RAM cells. Each processor location is read into the CM register prior to each CM=CMS shift so that the first video line is shifted up and written into the next higher row of PEs as each new video line if fed into the bottom row of processing elements. The GAPP based imaging systems have an ability to process information in real time.

4 Future of SIMD Processing – Conclusions

We have presented both theoretical and practical aspects of parallel computing, specifically computing based on SIMD structures. As technology reaches its physical reduction limits, the structural changes in computing become more and more crucial. It is obvious by now that real-time processing of images and other complex signals cannot be accomplished without structural concurrency and parallelism.

SIMD machines of the future will have many bit-intensive like features. However these features will remain transparent to the user. We are just now discovering the rich coupling among matrix-manipulation for multiplication especially when we examine the word- and bit-level operations. McCanny et al (Cann83) have reported that most of the current research effort has been expended at the word and system level for systolic arrays. However, they show that the systolic array approach at the bit level is nearly identical to that at the word level. Most importantly, they show that many important signal processing and data processing applications can be implemented using the regular structure of one or two primitive cells. Andrews (Andr85a,b) has further shown that such similarities between the bit- and word-levels make carry/borrow distances shortened if special number representations are invoked.

The thrust of these studies show that we can treat such mathematical intensive applications at the bit-level, capturing the power of VLSI along the way. Thus maintaining data flows at the bit-level have no significant impediments if non-conventional number systems are realized. This observation may even pave the way for TRIT's or ternary valued logic. In a recent paper by Hurst (Hurs84), such multi-valued logic (MVL) shows great promise for VLSI. His arguments correctly identify the present limitations of conventional binary logic. First, we are backing into the packaging thermal limits of VLSI. Second, a severe escalation of chip interconnection space is occurring. Some recent estimates indicate that silicon real estate just for on-chip wiring consumes more than half of the available die. As a result, we are now judiciously examining the denser information content to interconnection ratios afforded by MVL.

Although at first we are inclined to dismiss bit-level research as a backwards step, current studies now show that many matrix manipulations make word-level operations nearly identical to bit-level operations. From this basic observation, many researchers have concluded that systolic arrays may have practical implications sooner than expected. It was surmised that the primitive cell in the array must be very powerful. Hence, one immediately anticipates a VLSI structure with several hundred 68000's (or equivalent). Now, we can see that effective pipelining at the bit level permits us to bit-level primitive cells in a computationally powerful machine. The NCR GAPP (Geometric Arithmetic Parallel Processor) is one such realization. We contend that it holds promise of many new commercially available systolic devices to come.

Therefore, multi-valued logic may support new and denser VLSI structures. If we continue to explore the other number systems, even greater information densities may be possible. We can only conclude that the Illiac III may have been way ahead of its time (it used redundant numbers in the ALU!).

Finally, it can be noted that the existing systolic systems (like the GAPP) are really the first generation of many new generations of parallel devices. Additionally to obvious increases in density (number of PEs per wafer) and in computational power, we can also expect distribution of control functions which are now handled by one master controller. This distribution of control will result in systolic devices crossing the line between SIMD and MIMD concepts, which is also to be expected as the creation of 'intelligent' MIMD structure is probably one of the major goals in the design of computing systems.

5 References

(Baer80) Baer J-L., Computer Systems Architecture, Computer Science Press 1980.

(Hwan84) Hwang K. and Briggs F. A., Computer Architecture and Parallel Processing, McGraw-Hill,Inc.,1984.

(Flyn66) Flynn M. J., 'Very High-Speed Computing Systems,' Proc. IEEE, vol. 54, 1966.

(Feng77) Feng T. Y., 'Parallel Processors and Processing' ACM Computing Surveys, vol. 9, no. 1, Mar. 1977.

(Hand77) Händler W., 'The Impact of Classification Schemes on Computer Architecture,' Proc. 1977 Int. Conf. on Parallel Processing.

(Urqu84) Urquhart R. B. and Wood D., 'Systolic Matrix and Vector Multiplication Methods for Signal Processing,' IEE Proc.-F, Vol. 131, Oct. 1984.

(Cann84) McCanny J. V., McWhirter J. G. and Wood K., 'Optimised Bit Level Systolic Array for Convolution,' ibid.

(Ward84) Ward C. R. et al.,'Application of a Systolic Array to Adaptive Beamforming,' ibid.

(Spei83) Speiser J. M. and Whitehouse H. J.,'A Review of Signal Processing with Systolic Arrays,' Proc. SPIE, Real-Time Signal Processing,' Vol. 431, Aug. 1983.

(Cann83) McCanny J. V.,Wood K. M.,McWhirter J. G.and Oliver C. J.,'The Relationship Between Word and Bit Level Systolic Arrays as applied to matrix X matrix multiplication,' ibid.

(Andr85a) Andrews M., 'An SBNR Adaptive Signal Processor', Space Tech Corp. Technical Report ARO-143, Mar .1985, Fort Collins, CO.

(Andr85b) Andrews M.,'Comparative Implementations of the LMS Algorithm,' Int. Journal of Computers and Electrical Engineering, in review, 1985.

(Hurs84) Hurst S. L., 'Multi-valued Logic, Its Status and Its Future,' IEEE Trans. on Computers, Vol. C-3, no. 12, Dec. 1984.

(Gapp84a) NCR Technical Note NCR45CG72, Geometric Arithmetic Parallel Processor,Fort Collins, CO 80523, 1984.

(Andr85c) Andrews M. and Walicki J. S., 'Parallel Implementation of the LMS Algorithm' to be presented at the Int. Conf. on Parallel Processing, 1985

(Wall84) Wallis L.,'Utilising the GAPP in Associative Processor Applications,' NCR Internal Report, Fort Collins, CO, 1984.

(Gapp84b) GAPP Application Note No.1, NCR, Fort Collins, CO., 1984.

Comput. & Elect. Engng Vol. 12, No. 3/4, pp. 119–135, 1986
Printed in Great Britain

0045-7906/86 $3.00 + 0.00
Pergamon Journals Ltd

COMPARATIVE IMPLEMENTATIONS OF THE LMS ALGORITHM

MICHAEL ANDREWS†
Space Tech Corporation, 2324 Manchester Court, Fort Collins, Colorado 80526, U.S.A.

Abstract—A study of available hardware algorithms was made in order to design adaptive signal processors with VLSI. A suitable model invoking synchrony, topology, and granularity has been chosen to investigate design figures-of-merit for each implementation. At present, redundant arithmetic is being contrasted, basically because carry-free operations are possible resulting in a speed up. This paper focuses on models and primitive computational elements for the least-mean-square (LMS) algorithm embedded in conventional twos complement, bit-serial or distributed arithmetic, and redundant arithmetic processors.

1. INTRODUCTION

A technology-independent architectural study for adaptive signal processors is being pursued. The basic approach is to analyze the hardware/software trade-offs of conventional (von Neumann) and nonconventional (parallel, pipeline, vector, array, and custom processors) in an attempt to identify optimal structures (of order area × time) which are computationally fast, yet flexible. Regular and simple interconnections and geometries among high-performance parallel structures are sought. A two-step process is assumed; first the sequential algorithms are to be speeded up (seeking inherent parallelism) and second, fast algorithms are to be mapped onto new VLSI architectures (via recursion and pipelining). The purpose of this research is to provide theoretical design tools and interconnection strategies capable of achieving real-time implementation of adaptive algorithms via limited user programmable mechanisms (e.g. firmware).

In this effort, design rules for establishing implementation techniques which execute nonrecursive and recursive adaptive algorithms must be stated. The design rules should identify structures suitable for VLSI. Trade-offs between various hardware techniques are then possible. This study has particular relevance to the problem of realizing a minimized system architecture for monolithic adaptive signal processors. VLSI devices capable of being organized by the proposed rules could possess high bandwidth, low power attributes in a microminiature configuration to enhance performance of adaptive antennae arrays, spectrum analyzers, acoustics, cryptography (adaptive keys), image processing, and communications.

1.1 *Opportunity and the challenges*

VLSI technology opens unprecedented opportunities for synthesizing complex computational algorithms from various fields of engineering. It is now possible to integrate a huge amount of hardware on a small silicon area. However, many traditional computer design concepts are no longer justified technologically, and this leads to the formulation of new and challenging problems. A distinct characteristic of VLSI devices is that the data communication and its VLSI interconnect area dominate the cost and performance. In contrast, traditional parallel processing finds memories and the processors dominating other design factors.

2. DESIGNING WITH VLSI

VLSI has now circumscribed classical methodologies of digital system design. The traditional criteria of component count, whether applied to processors or to simpler

† Staff Scientist

devices, are no longer adequate to establish a scale of comparison among various solutions to a given problem. Indeed, number-of-elements criteria are substantially based on the fact that processing elements and their interconnections are realized by different media. This difference disappears in VLSI, which "integrates" both processing elements and their interconnection in a two-dimensional geometry, the surface of the silicon chip. A meaningful figure-of-merit is represented by the area occupied by the total system, thus capturing the complexity of both computation and data communication[1]. As a result, the VLSI solution to a given computational problem involves the conception of an *interconnection architecture, its layout, and the design of an algorithm for that architecture*. For any given problem, it is of great interest to explore the trade-offs between the production cost (area) and the incremental cost (time) of a dedicated circuit developed to solve that problem. The area of a chip can be partitioned into interconnect or wire area $A(w)$, gate area $A(g)$, and wire pad area $A(p)$. And it appears, so far, that wire area dominates gate area. At least for the class of transitive functions (cyclic shifts, matrix product, integer product, and linear transforms), Vuillemin[2] has shown such VLSI circuits must satisfy

$$A > A(g)N + A(p)B + A(w)B,$$

where A is the chip area, N is the number of inputs, and B is the average bandwidth (bits per second passing through the chip pins). Interestingly, Rent's rule[3–5] is partially substantiated here. (A Rent's rule relation states that bandwidth is an increasing function of area.) Vuillemin's result substantially supports claims that interconnect area is VLSI expensive. Furthermore, circuits based on a recursive construction are particularly well suited to automated design.

2.1 *The set of* VLSI *design goals*

This research is concerned with the study of algorithms for VLSI arrays, and focuses on the transformation of sequential/quasirecursive programs into VLSI algorithms. To do so, it is necessary to define a set of objectives. Although only a partial list, the following objectives are considered to be important:

1. Correctness and accuracy of the algorithm.
2. Small computation time; computation time includes processing time and communication time, but not necessarily control time (which should be transparent).
3. Small number and size of interprocessor communication links attempting to minimize the excessive interconnect area on current VLSI.
4. Modularity and simplicity of cells, hopefully very similar.
5. Small number of processing cells which may achieve regularity.

Of course, for specific applications the relative weights of these objectives do vary depending on many factors including technology, yields, manufacturing limits, die size, power, and speed. Obviously, the accuracy of the result is a prime concern of the design. The processing time results from the requirements of the algorithm. Here, architects often seek trade-offs. In VLSI systems, the communication time is at least as important as the processing time, because physical distance is relatively long. Designers, therefore, search for algorithms which are neither computationally nor communication saturated. Many researchers contend that "balanced" algorithms can be mapped more easily onto high-performance VLSI architectures. For instance, Kung[6] has indicated that interprocessor communication links consume a great deal of silicon area, time, and energy. In that event, it is desirable to have as few links as possible, and moreover, to restrict the data communication only to adjacent cells which may be achieved by adequate transformations of algorithms. This goal, however, places a heavy constraint on the architecture. A higher modularity as well as the simplicity of the cells lead to a smaller design cost.

A hardware model (G, F, T). Transformation of algorithms can proceed when a hardware model is specified. A model proposed by Moldovan[7] is useful to transform the abstract features of the algorithm into the hardware. We assume the following features for VLSI networks.

1. The network consists of a planar mesh connected network of processing cells. Nonplanar meshes are still academic matters.
2. The cells can be of different types and perform different functions, but a minimal set is desirable.
3. The interconnections between cells are buses which transfer parallel words and represent a topology.
4. The timing operation of the network is synchronous.

Moldovan proposes an organization and operation for VLSI arrays which can be formally described by a set of 3-tuples (G, F, T). This set appears to be a promising starting point because it takes into consideration the three dominant design factors (topology, granularity, and synchrony).

The network geometry G refers to the topology. The position of each processing cell in a plane is to be described by its Cartesian coordinates. Arbitrarily choosing a small enough grid makes it feasible to represent these coordinates by integers. Interconnections between cells is now a matter reasonably described by the position of terminal cells. These interconnections support the flow of data through a network of links. As with all current VLSI structures, a simple and regular geometry is desirable.

The functions F associated with each processing cell represent the granularity of arithmetic/logic expressions of a cell. Most VLSI implementations assume that each cell consists of a small number of registers, ALU, and control logic. Several different types of processing cells might be implemented in the same topology; however, one reasonable design goal is to reduce the number of cell types and, hence, their granularity.

The network timing T specifies the processing instant of a cell. As a matter of synchrony, obviously, a correct timing means that the appropriate data arrives at destination cells at the correct time. The data stream speeds through a network defined as the ratio between the distance of a communication link over the communication time. Often, networks with constant data speeds are preferable solely because they require less control logic.

2.2 VLSI *Figure-of-merit*

To provide a meaningful gauge for the evaluation of a given design, a computational model of VLSI has been developed through the initial efforts of Mead and Conway[8] and Thompson[9]; later refinements have been proposed by Brent and Kung[10] and Vuillemin[2]. We briefly recall the model with amendments.

A circuit is a graph whose nodes are I/O ports or gates connected by wires. A wire has minimal width q and, at most, L wires overlap at any point (planarity); a gate has minimal area and computes a Boolean function of two inputs; an I/O port has some minimal area and each input bit is available just once. As regards computation time, the combined gate computation and result transmission (on a wire between two nodes) takes some time R dependent upon the technology. One relevant global time parameter is the output period P of the circuit, defined as the maximum time between two successive data passages at any output port when the circuit is used in a pipelined fashion at the highest data rate. Another measure is the time T which elapsed between the beginning and the end of one computation by the chip for one instance (rather than repeated instances) of a given problem. On the basis of arguments on the information transfer inside the VLSI chip of area A realizing the circuit, natural measures of complexity in the given module are the area–time products AP^2 and AP.

If we define the problem size as the larger of the total number of bits used to specify either the input or the output, a simple argument by Vuillemin[2] shows that the circuit area A, period P, and problem size N satisfy the relationships

$$AP^2 > N$$

and

$$AP > N$$

for such fundamental problems as integer multiplication, merging, cyclic shift, cyclic and open convolution, and linear transform. Computing time T and period P are obviously related to $T > P$ so the above inequalities imply $AT^2 > N$ and $AT > N$. Several of the above mentioned problems have been considered elsewhere[10, 12, 13], and circuits have been proposed which are optimal with respect to the AP^2 or AP measure. This study is devoted to seeking design rules implementing adaptive signal processors which are optimal with respect to area × time.

3. THE ALGORITHM SPACE

3.1 *Adaptive algorithms*

The theory of adaptive algorithms is relatively well developed and many algorithms have been proposed[14–35]. However, many of these algorithms are computationally complex and are really only suitable for non-real-time implementation on digital computers. In our study, the algorithms to be used should be as simple as possible (to reduce user-programmability requirements) and also be tolerant of device noises (to help increase computational speed). This background material is given as a guide to the eventual selection of a class of adaptive algorithms suitable for real-time processor implementation.

In general, as a filter, an elemental adaptive signal processor may be viewed as a system supplied with two inputs, a signal input and a desired output. The signal is applied to the input of a FIR (finite impulse response) filter with a programmable (time variable) impulse response. The impulse response of this filter is adjusted in such a way that the filter output approximates the desired output as closely as possible. IIR (infinite impulse response) realizations are also possible where internal feedforward as well as feedback signal paths exist. For discussion purposes, we loosely classify the former as nonrecursive algorithms and the latter as recursive algorithms.

3.1.1 *Nonrecursive algorithms.* In the elementary case described now, a popular updating algorithm used is the Widrow least-mean-square (LMS) algorithm which updates the weight vector W to minimize the mean-square error between some desired signal $d(t)$ and the filter output $y(t)$. A derivation of the algorithm may be found in Ref. [16]. Briefly stated, the updated weight vector H' is given by $H' = H - 2\mu e(t)S$ where H is the previous value of the weight vector, S is the signal vector, μ is a convergence factor, $e(t) = d(t) - y(t)$ is the error output, and $d(t)$ is the desired or training signal. Proofs of the convergence of this algorithm assuming perfect device parameters may also be found in Ref. [36]. A nonrecursive adaptive filter structure is displayed in Fig. 1.

Currently, the "sliding window" exact least-squares algorithms (also known as a sliding window covariance) has more superior tracking properties than the LMS (gradient) algorithm[37]. Hence, our studies on nonrecursive algorithms to be cast into VLSI schemes shall include those from LMS to the sliding window formulations.

3.1.2 *Recursive algorithms.* The previously described algorithm is one of many

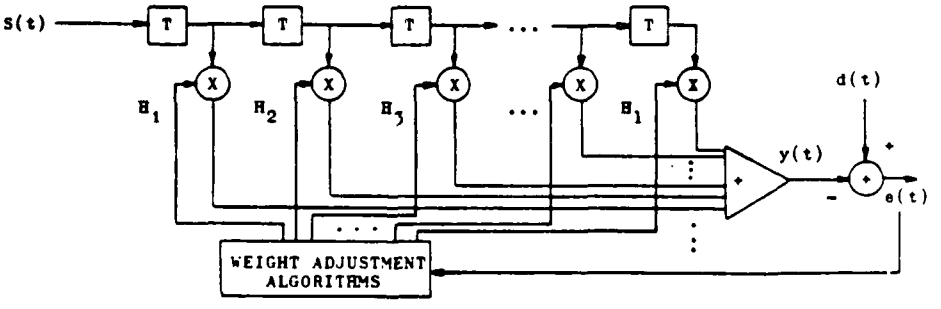


Fig. 1. A Non-recursive adaptive processor.

nonrecursive algorithms. Adaptive IIR or recursive adaptive formulations also enjoy a wide selection of algorithms. Popular among these are the:

1. Stearns–White recursive gradient algorithm.
2. Feintuch gradient approximation algorithm.
3. Soderstrand–Gitlin recursivelike echo canceler algorithm.

As with many recursive structures, due to the presence of poles, maintaining stability during adaptation becomes crucial[27]. In addition, the LMS error surface is nicely quadratic with respect to the weights. The IIR algorithms' corresponding performance surface is nonquadratic and may contain multiple minima[34].

Let us formulate a system identification problem as in Fig. 2. The transfer functions are defined as

$$\mathbf{H}(z) = \frac{A(z)}{B(z)} = \frac{a_0 + a_1 z^{-1} + \cdots + a_n z^{-n}}{1 - b_1 z^{-1} - \cdots - b_m z^{-m}}, \tag{1}$$

$$\hat{\mathbf{H}}(z) = \frac{\hat{A}(z)}{\hat{B}(z)} = \frac{\hat{a}_0 + \hat{a}_1 z^{-1} + \cdots + \hat{a}_n z^{-n}}{1 - \hat{b}_1 z^{-1} - \cdots - \hat{b}_m z^{-m}}. \tag{2}$$

This adaptive processor attempts to adjust the coefficients of $\mathbf{H}(z)$ so that the minimum mean-square output error $E(e^2)$ is obtained.

Update algorithms for the weights take on the form

$$H_{k+1} = \mathbf{H}_k + \mathbf{M}\mathbf{R}^{-1}(-\nabla_k), \tag{3}$$

where

$\mathbf{H}$ is an adaptive weight vector,
∇_k is the performance surface gradient vector,
$\mathbf{M}$ is a diagonal matrix of convergence factors,
$\mathbf{R}$ is a correlation matrix (elements determined by selected adaptive algorithm).

The weight vector and gradient vector are, respectively,

$$\mathbf{H}_k = | \hat{a}_0 \cdots \hat{a}_n, \hat{b}_1 \cdots \hat{b}_m |^T, \tag{4}$$

$$\nabla_k = \frac{\partial E(e^2)}{\partial H_k} = \left| \frac{\partial E(e^2)}{\partial \hat{a}_0} \cdots \frac{\partial E(e^2)}{\partial \hat{b}_m} \right|^T. \tag{5}$$

The Stearns–White algorithm first proposed in 1975[32] is a gradient algorithm similar to the LMS technique. However, the parameter update method uses a recursive calculation of the gradient. This approach is computationally expensive because of the complicated nature of the error function $e(t)$ in recursive filters. Here, the instantaneous output error is used as a local estimate of its own mean value. The adaptive updates

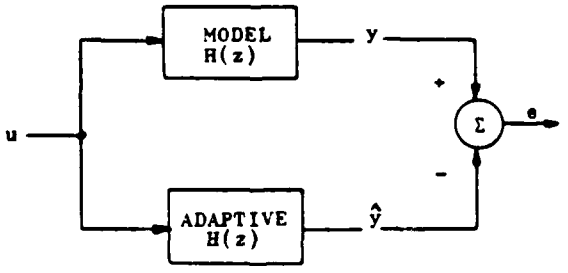


Fig. 2. A system identification problem.

are expressed as

$$a_n(k) = -\frac{\partial e(k)}{\partial \hat{a}_n} = u(k-n) + \sum_{i=1}^{n} \hat{b}_i(k) a_n(k-i), \tag{6}$$

$$b_m(k) = -\frac{\partial e(k)}{\partial \hat{b}_m} = \hat{y}(k-m) + \sum_{i=1}^{m} \hat{b}_i(k) b_m(k-i), \tag{7}$$

where the instantaneous output error is

$$e(k) = y(k) - \hat{y}(k) \tag{8}$$

and the matrices of convergence factors, correlation matrix, and gradient vector become

$$\mathbf{M} = \text{diag}| \, u_0 \cdots u_n P_1 \cdots P_m \, |, \tag{9}$$

$$\mathbf{R} = \mathbf{I} = \text{Identity matrix}, \tag{10}$$

$$\hat{\nabla}_k = e(k) \mid a_0(k) \cdots a_n(k) b_1(k) \cdots b_m(k) \mid^T. \tag{11}$$

Feintuch[35] proposes a greater simplification to reduce computations per iteration with a gross approximation of the gradient. This algorithm is based on the assumption that the covariance terms which arise in taking the expected value of the squared output error are constants when differentiated with respect to the coefficient vector **H**. As a result, the gradient vector can be estimated as

$$\hat{\nabla}_k = e(t) X^T(t)_k, \tag{12}$$

where

$$X^T(t)_k = \mid u_k \cdots u_{k-n} \hat{y}_{k-1} \cdots \hat{y}_{k-n} \mid. \tag{13}$$

The third algorithm is a modification of the Gitlin algorithm[33]. A new error function is defined to which the LMS algorithm is applied. In all IIR cases, we must recognize that we are incurring much larger computational costs than with the simple LMS structure alone. But there are strong results that suggest that fewer filter coefficients are needed for IIRs. Therefore, fewer multipliers may be necessary, thus reducing the computational costs somewhat.

Soderstrand[17] makes a revealing comparison among these choices based on equal cost implementation. Here, he assumes that the multiplier is the overwhelmingly expensive element in any physical realization, so his comparisons rest on each implementation with identical numbers of multipliers. Although the results are qualitative, we can make the following observations on the relative merits of the LMS nonrecursive algorithm and the three identified above. For 63 multiplier realizations, the LMS algorithm performs as well as the best recursive choice (Soderstrand–Gitlin). Unfortunately, we cannot unreservedly choose an LMS over recursive methods based on Soderstrand's work. Even though he measures performance of the algorithm by how close their respective filter parameters converge to the least-mean-square approximations to five test cases, he has offered no quantitative measures.

3.2 *Effectiveness of redundant arithmetic*

Ercegovac[38] has proposed fast computational methods amenable for efficient hardware level implementation as viable alternatives to parallel algorithms implemented and to implementation-dependent algorithms, primarily operating in a fixed-point number representation system. We can generally classify fast methods as: (1) those implemented with multiple general purpose processors and with the corresponding algo-

rithms, and (2) those implemented with special purpose processors with algorithms embedded in the hardware (operation) level.

When considering a computational method for possible implementation, we are concerned with: (1) the application domain, (2) the required set of algorithms, and (3) the required set of primitive operators. With these, we can then associate a set of desired or required properties; for instance, the speed, the complexity, and the cost of implementation, the fault tolerance capability, numerical characteristics of the algorithms, etc. The objective of this study is to define a method that would have a sufficient generality to adaptive signal processor applications and such functional properties in order to justify a hardware level implementation.

Ercegovac's method is compatible in many respects with the method invoking multiple processors because the computational algorithm is simple and problem invariant. There is no shift operator, which in the previously proposed methods must have a variable shift capability (e.g. distributed arithmetic schema of Peled and Liu[39], Roberts[40], Cowan *et al.*[41], and others[42–44]). When a redundant representation is introduced in order to increase the speed of computation, a variable shift operator can considerably affect the complexity of implementation.

It can be demonstrated that certain arithmetic expressions, multiple products and sums, inner products, integral powers, and solving of systems of linear equations under certain conditions are among the possible applications. Basic arithmetics, in particular, multiplication and division, can be performed by this method. Furthermore, it has a useful functional property in that the results are generated in a digit-by-digit fashion with the most significant digits appearing first so that an overlap of computations can be utilized. Equally noteworthy is the fact that AD conversion gives the most significant digit first so overlapped signal processing is possible.

3.3 *Basic division/multiplication in redundant arithmetic*

Let us consider problems of division and multiplication in a computational environment whereby basic arithmetic algorithms satisfy an "on-line" property. In other words, to generate the jth digit of the result, it will be necessary and sufficient to have the operands available up to the $(j + \delta)$th digit, where the index difference δ is a small positive constant. At first, we will accumulate δ initial digits of the operands before we can produce the first digit of the result. Subsequently, one digit of the result is produced upon receiving one digit of each of the operands. Remarkably, δ is the on-line delay which can be arbitrarily small. Such algorithms are attractive because of the inherent speed up due to their potential to perform an overlapped sequence of operations. Fast variable precision arithmetic is also possible. The on-line property will implement a left-to-right digit-by-digit type of algorithm using a redundant representation for the results.

Consider an m-digit radix r number $N = \sum_{i=1}^{m} n_i r^{-i}$. In the conventional representation, each digit n_i can take any value from the digit set $\{0, 1, \ldots, r-1\}$. Such representations, which allow only values in the digit set, are nonredundant since there is a unique representation for each (representable) number. By contrast, number systems that allow more than r values in the digit set are redundant, and often speed up arithmetic operations[45, 46]. Note that a redundant number representation may be required for on-line algorithms. In a nonredundant number system, addition and subtraction incur a carry propagation penalty. Redundancy limits the carry propagation to one digital position [cf. Ref. [45], an on-line algorithm for addition (and subtraction) with $\delta = 1$, and an on-line algorithm for multiplication with $\delta = 1$].

3.4 *The computational algorithm using redundant arithmetic*

Suppose a linear system of L equations is given. An algorithm for solving a system L is desired whereby an iterative, digit-by-digit method occurs. That is, the algorithm generates one digit of each of the elements of the solution vector in one step. Some redundant number representation definitions are now appropriate.

DEFINITION 1

An m-digit radix r representation of a number x, $|x| < 1$, is a polynomial expansion

$$x = \operatorname{sgn} x \sum_{i=1}^{m} x_i r^{-i}, \tag{14}$$

where $x_i \in D \; \forall \; i$, and D is a digit set.

DEFINITION 2

Given the radix r, a set of consecutive integers D including zero is

(1) nonredundant if its cardinality satisfies $|D| = r$.
(2) redundant if $|D| > r$.

DEFINITION 3

A symmetric redundant digit set is defined as

$$D_p = \{-p, -(p-1), \ldots, -1, 0, 1, \ldots, p-1, p\}, \tag{15}$$

where

$$\tfrac{1}{2}r \leq p \leq r - 1. \tag{16}$$

DEFINITION 4

A symmetric redundant digit set D_p is said to be

(1) minimally redundant if

$$|D_p| = r + 1, \tag{17}$$

i.e. $p = \frac{1}{2}r$ (assuming an even radix r);

(2) maximally redundant if

$$|D_p| = 2r - 1, \tag{18}$$

i.e. $p = r - 1$. Let D and D_p denote nonredundant and redundant digit sets, respectively. Then the representation of a number x is nonredundant or redundant depending on whether $x_i \in D$ or $x_i \in D_p$.

4. ON-LINE MULTIPLICATION

The following describes an on-line algorithm for multiplication which can be made compatible with on-line division. It is a type of incremental multiplication (using digital differential analyzers[47, 48], combined with the redundant numbers).

Let

$$X = \sum_{i=1}^{m} x_i r^{-i}, \tag{19}$$

$$Y = \sum_{i=1}^{m} y_i r^{-i} \tag{20}$$

be the radix r representations of the positive multiplicand and the multiplier, respectively. Define

$$X_j = \sum_{i=1}^{j} x_i r^{-i} = X_{j-1} + x_j r^{-j}, \tag{21}$$

$$Y_j = \sum_{i=1}^{j} y_i r^{-i} = Y_{j-1} + y_j r^{-j} \tag{22}$$

to be the j-digit representations of the operands X and Y, available at the jth step by definition of an on-line algorithm. The corresponding partial product is then

$$X_j \cdot Y_j = X_{j-1} \cdot Y_{j-1} + (X_j \cdot y_j + Y_{j-1} \cdot x_j) r^{-j}. \tag{23}$$

Let P_j be the scaled partial product, i.e.

$$P_j = X_j \cdot Y_j \cdot r^j, \tag{24}$$

so that the recursion of the multiplication algorithm can be expressed as

$$P_j = rP_{j-1} + X_j \cdot y_j + Y_{j-1} \cdot x_j. \tag{25}$$

Let $P_0 = 0$. Then the scaled product $P_m = XYr^m$ can be generated in m steps of eqn (25). Note that if a nonredundant number system is used in representing the partial products, the digits of the desired product appear in a right-to-left fashion, as determined by the conventional carry propagation requirements. However, for a redundant number representation, left-to-right generation of the product digits is possible and desirable. Furthermore, the execution time to perform a recursive step will be independent of the operand precision because carry-free addition is possible. We now essentially describe the applicability of Ref. [49] to our current work.

We will use a symmetric redundant digit set which is

$$D_p = \{-p, -(p-1), \ldots, -1, 0, 1, \ldots, p-1, p\}, \tag{26}$$

where

$$\tfrac{1}{2}r \leq p \leq r - 1. \tag{27}$$

According to the general computational method of Ref. [50], the basic recursion (25) for the multiplication is

$$w_j = r(w_{j-1} - d_{j-1}) + X_j \cdot y_j + Y_{j-1} \cdot x_j, \tag{28}$$

where the digits $d_j \in D_p$ are determined by the selection function

$$d_j = S(w_j) = \operatorname{sgn} w_j [\, | w_j | + \tfrac{1}{2}]. \tag{29}$$

Then, from (25) and (28), the following relation can be obtained by induction:

$$w_j = P_j - \sum_{i=1}^{j-1} d_i \cdot r^{(j-i)}. \tag{30}$$

Substituting $j = m$ in (30) and rearranging, we obtain

$$P_m = X \cdot Y \cdot r^m = r^m \sum_{i=1}^{m-1} d_i r^{-i} + w_m \tag{31}$$

or

$$X \cdot Y = \sum_{i=1}^{m} d_i r^{-i} + (w_m - d_m) r^{-m}. \tag{32}$$

According to the selection function $S(w_j)$ where $| w_m - d_m | \leq \frac{1}{2}$, $\sum_{i=1}^{m} d_i r^{-i}$ is now the redundant representation of the most significant half of the product $X \cdot Y$. Convergence of the algorithm is now guaranteed.

Let us assume that the selection function (29) can produce the digit d_j such that $|d_j| < p$. This condition is satisfied provided that

$$|w_j| < p + \tfrac{1}{2}, \tag{33}$$

where $j = 1, 2, \ldots, m$. We next bound M on the values of the operands X and Y to ensure that (32) holds. Let $|X| < M$. Noting that $|w_{j-1} - d_{j-1}| < \frac{1}{2}$ and $|y_j| \leq p$, from (33) we obtain

$$|w_j| \leq \tfrac{1}{2}r + 2Mp. \tag{34}$$

Rearranging (32) we obtain

$$M < \frac{1}{2}\frac{r-1}{4p}. \tag{35}$$

Notice that in a minimally redundant system ($p = r/2$), the required operand bound must be

$$|X|, |Y| < \frac{1}{2r}, \tag{36}$$

whereas in a maximally redundant system ($p = r - 1$), the required operand bound must be

$$|X|, |Y| < \tfrac{1}{4}. \tag{37}$$

These bounds are not tight. In binary radix, $|X|, |Y| < \frac{1}{4}$ suffices.

The time required for the computation of w_j is made independent of the precision of the corresponding operands by allowing

$$\tfrac{1}{2} < |w_j - d_j| < 1, \tag{38}$$

which implements a carry-propagation-free addition per (29). This last observation is most critical to the redundant arithmetic LMS implementation.

4.1 LMS *signal conventions*

In subsequent discussions, all one-dimensional matrices are represented by column vectors. Boldfaced characters are vectors or matrices. As usual, the superscript t refers to the transpose of a column vector or matrix, and printed vectors represent updated vectors. (The variable t, denoting time, is omitted from subsequent expressions, but is implicit to discussions.) The necessary scalars are defined as:

$h(n)$ = nth coefficient of an N-point digital transversal filter
$f(k)$ = kth partial product used in the output accumulation of a distributed arithmetic filter
$s(n)$ = A-bit input signal sample present at point n of an N-point digital filter
y = digital filter output
d = input training signal to digital adaptive filter
e = $d - y$ = error sample generated by digital adaptive filter

These scalars form the following matrices:

$\mathbf{S}^t$ = $(s(1), s(2), \ldots, s(n), \ldots, s(N))$
$\mathbf{H}^t$ = $(h(1), h(2), \ldots, h(n), \ldots, h(N))$
$\mathbf{F}^t$ = $(f(1), f(2), \ldots, f(k), \ldots, f(A))$
$\mathbf{X}^t$ = $(2^{-1}, 2^{-2}, \ldots, 2^{-A})$, i.e. the set of the first A negative integer powers of 2
$\mathbf{B}$ = the $N \times A$ array of bit values which results when a A-bit input signal vector is stored in an N-point digital filter. $\mathbf{B}$ is merely S decomposed into component bits.

Consider a set of N registers, each of length K bits which provides storage for the $N \times K$ array of bit values. This array is represented by the $N \times K$ matrix of binary values **B**. We then view a classical N-point transversal filter as capturing signal vector **S** containing N K-bit components. The signal vector can be expressed as

$$\mathbf{S} = \mathbf{BX}, \tag{39}$$

where it is assumed that the signal is coded in offset binary, wherein logical 0 takes the value -1. The output of the filter is given by the convolution

$$y = \mathbf{S}^T\mathbf{H}, \tag{40}$$

where the column vector **H** represents the set of N filter coefficients. Define the column vector **F** as

$$\mathbf{F} = \mathbf{B}^T\mathbf{H}, \tag{41}$$

and substituting (39) in (40), using the property of matrix transposition we have

$$y = \mathbf{X}^T\mathbf{F}, \tag{42}$$

where **F** is a set of partial products.

Cowan *et al.*[41] has observed that the output filter formulation of (42) when compared with (40) reveals the essential elements of the distributed arithmetic architecture of the LMS algorithm depicted in Fig. 3. Simply stated, Fig. 3 is a summation over K-bit planes rather than over N filter points. Only the basic hardware configuration for a fixed-response distributed arithmetic filter is depicted in Fig. 3. The input ADC (analog-to-digital converter) signals are presented serially to a set of N cascaded K-bit shift registers. As this serial bit stream enters the shift registers, the shift register parallel outputs generate K N-bit address words on the RAM address bus. Each RAM datum is then right shifted k bits and accumulated. The accumulation is complete after K memory accesses. Finally, an output sample is converted to an output analog signal. Since the filter coefficients F are adaptable, the buffer/sum block will generate the coefficient updates to the RAM.

4.2 *Conventional binary* ALU

The LMS algorithm can be implemented in conventional twos complement arithmetic architectures using hardware intensively or sparsely. Obviously, if the algorithm is implemented with $2N$ multipliers and $2N$ adders (very intensive), no faster computation speeds can be achieved. The on-line delay is approximately equal to that of

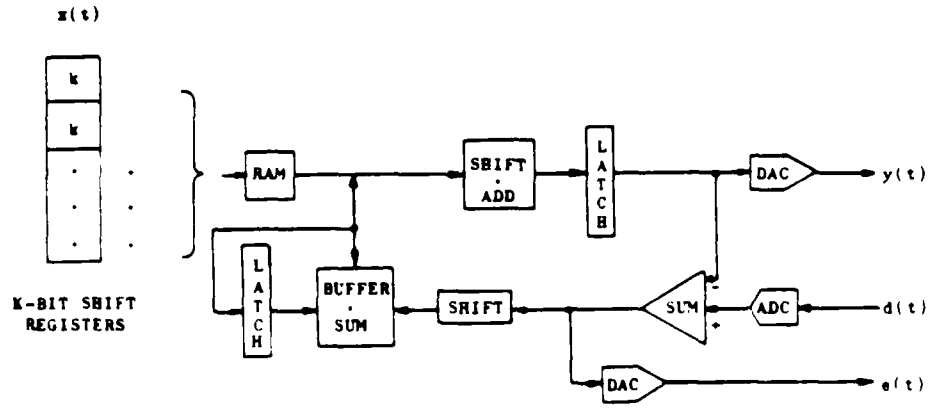


Fig. 3. Distributed arithmetic architecture

the slowest computational element (adder, multiplier, ADC). In practice, we realistically expect a trade-off between computation speed and the amount of hardware.

Two cases are considered: case 1 with a one adder and one multiplier element and case 2 with $2N$ adders and $2N$ multipliers (where N is the number of filter coefficients). In both cases, multipliers and adders process twos complement numbers in parallel m-bit fashion. The architecture of Fig. 4 is assumed in either case, except that the adders and multipliers are replicated $2N$ times for case 2.

Of great interest is the minimum sampling time T of each implementation. This figure-of-merit is a function of the following processing element periods:

T_a = m-bit, twos complement addition cycle,

T_w = memory write cycle time,

T_r = memory read cycle time,

T_s = one-bit shift,

T_m = m-bit multiplication of two operands.

Assume that the LMS algorithm invokes the equations

$$y = \mathbf{S}^T\mathbf{H} \qquad \text{(filter output)}, \tag{43}$$

$$e = d - y \qquad \text{(filter error)}, \tag{44}$$

$$\mathbf{H}' = \mathbf{H} + ue\mathbf{S} \qquad \text{(weight update)}. \tag{45}$$

where upper case and lower case denote vectors and scalars, respectively. The filter convergence rate is determined by the scalar u. In practice, u is simplified to 2^K, $K \in \{0, -1, -2, -3, \ldots\}$. We make the assumptions that analog-to-digital conversion time T_{ADC} and digital-to-analog conversion time T_{DAC} are relatively short:

$$T_{DAC},\ T_{ADC} \ll \text{processing time of slowest computational element.} \tag{46}$$

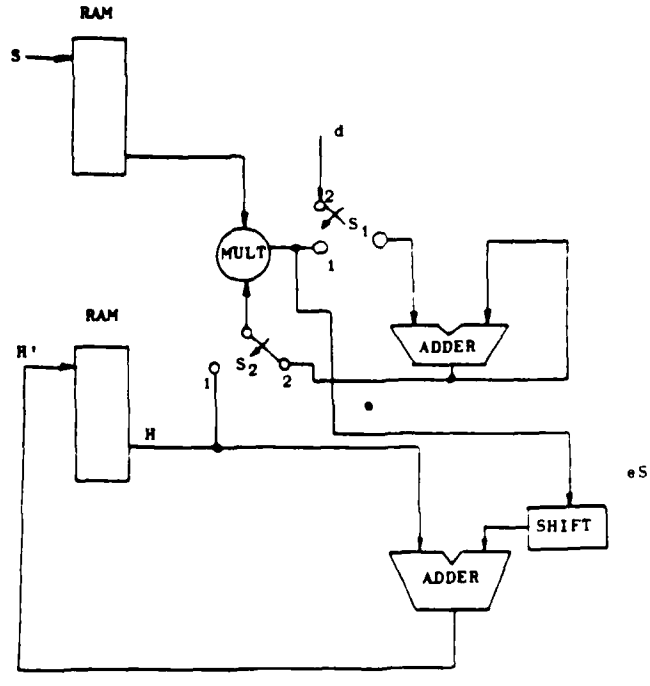


Fig. 4. Conventional binary architecture.

Let A_a, A_m, and A_M represent the chip area for the m-bit parallel adder, multiplier, and memory.

Case 1: One twos complement m-bit parallel adder and one m-bit multiplier.

$$T = 2N(T_w + T_r + T_m + T_a) + uNT_s + T_a. \tag{47}$$

The area × time figure of merit, FOM, can be no better than

$$\text{FOM} \geq (A_m + A_a + A_M)[2N(T_w + T_r + T_m + T_a) + uNT_s + T_a]. \tag{48}$$

Case 2: $2N$ multipliers and m-bit parallel adders. After an initial delay (N + 1 samplings to fill the memory), it is possible to produce a filter output at every sampling instant. Here, eqn (46) may no longer apply. Even so, signal conversion can be pipelined with data processing causing only a slight penalty:

$$T = T_r + T_m + 3T_a + T_w, \tag{49}$$

$$\text{FOM} \geq 2N(A_m + A_a + A_M)(T_r + T_m + 3T_a + T_w). \tag{50}$$

The FOMs in both cases are lower limits because no interconnect area is considered. However, the bound in eqn (48) is much tighter than the bound in eqn (50), simply because case 2 utilizes $2(N - 1)$ more computational elements than case 1.

4.3 *Bit-sequential cell*

Sips[44] has proposed a primitive bit-sequential cell and a linear two-dimensional processing element for addition, subtraction, multiplication, and division as shown in Fig. 5. Multiplication is based on the well-known technique of incremental multipli-

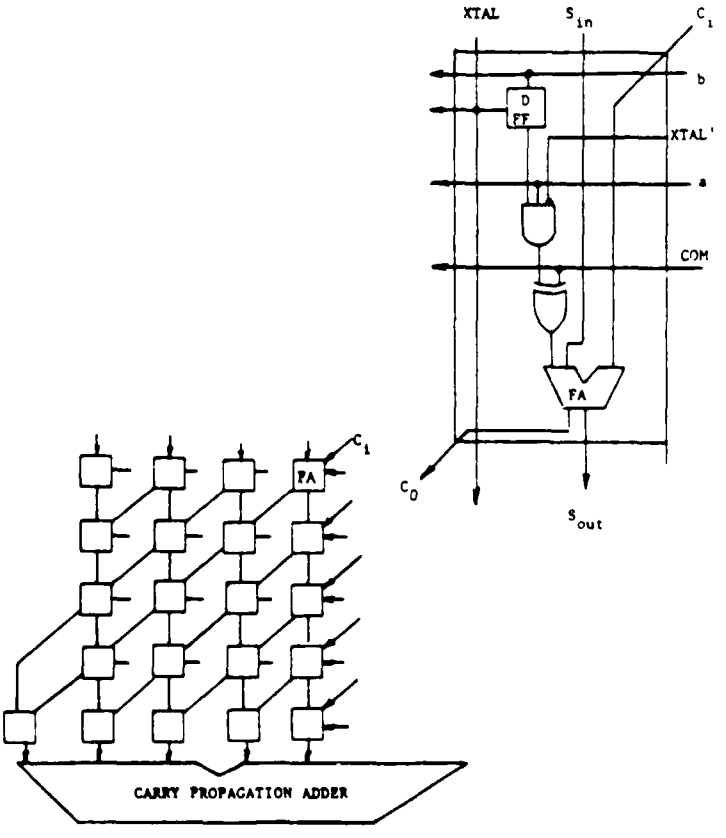


Fig. 5. Bit-sequential cell architecture

cation[49] in which the MSB is produced first. Figure 5 depicts an individual full adder (FA) cell and a bitwise floor plan to execute the basic algorithms of the product pairs *AB* and *CD*, essentially using a systolic array with one-bit full adders. Processing begins in the upper right corner and proceeds downward through the array.

Each cell is comprised of a D flip-flop, 3-input AND gate, 2-input XOR gate, and a full adder. The XOR gate is provided for obtaining the complemented operand (if the operand has a negative weight). Two operands are provided to each cell via the "*a*" and "*b*" lines. Control lines XTL and XTL' load successively each new bit of the operand in the next column (to perform the systoliclike addition).

The basic feature of this architecture configured with the bit-sequential cell is that all operations have the same operation time and that the operation time is linearly dependent on the number of bits of the operands. The hardware complexity described in Ref. [51] has been shown to be of $O(m)$ complexity.

4.4 *Redundant arithmetic cell*

A digit slice proposed in Ref. [51] can be used as the basic computational redundant arithmetic cell. The cell is a three-level digit slice capable of implementing addition, subtraction, multiplication [using eqn (32)], and division. A restriction on the digit ranges per (36) applies here. It is important to note that the basic LMS algorithm implemented with these cells can operate on the most significant digit first. Hence, this computational feature makes best use of an analog-to-digital converter which first produces the most significant digit. In this case, processing and conversion can be overlapped.

The cell depicted in Fig. 6 assumes that the input digits a, b, and $c \in D$ belong and that the data outputs consist of a result digit s and three transfers w, t, and y. Primed digits are intermediate transfers between cells. The cell performs the following functions.

1. Product of b and c:

$$ru + w \leftarrow bc \qquad \text{where } r \text{ is the radix.} \tag{51}$$

2. Multidigit addition of a, w', u:

$$rx + t \leftarrow a + w' + u. \tag{52}$$

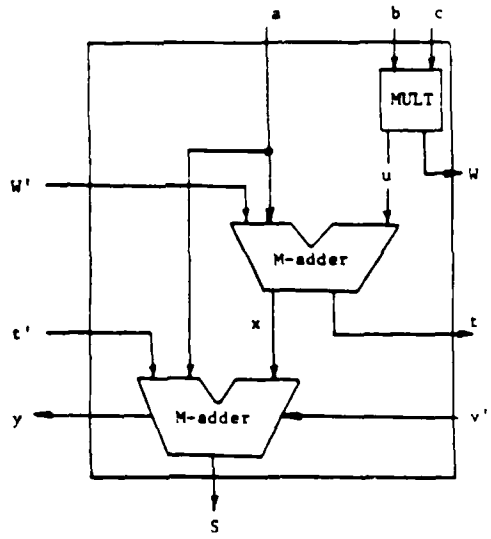


Fig. 6. Redundant arithmetic cell architecture

Table 1. Comparison of architectural complexity

	Conventional binary (2N multipliers) (2N adders)	Distributed arithmetic	Redundant arithmetic cell	Bit-sequential cell
Gate complexity	$O(2\ mN)$	$O(kN)$	$O(m)$	$O(m)$
Latency	N + 1 memory writes	kN bit shifts	one ADC bit conversion	δ + 1 ADC bit conversion
VLSI amenable	structure irregular	yes	yes	yes
Estimated pin count/cell	technology sensitive	technology sensitive	$10m$	40†

† Assumes each cell is a 4-bit slice.
m = number of digits in a word.
k = number of bits in each shift register ($k < m$)
N = number of filter coefficients
δ = small positive constant (3 or 4) less than T_a, the time to complete a full bit-parallel operation

3. Multidigit subtraction:

$$rx + t \leftarrow a - w' + u. \tag{53}$$

4. Assimilation (maintenance of signed digit code):

$$-ry + s \leftarrow a - y' \qquad \text{where } y \in (0, 1). \tag{54}$$

The digit slice is configured in a linear one-dimensional array. In left-to-right fashion, any intercell transfers propagate via the w', t', and y' lines. Results appear at the bottom of each cell. With the floor plan for the basic adder/multiplier unit, few bridges or vias are anticipated. Thus, "stray" transistors caused by metallic bridges can be minimized.

5. SUMMARY

Four architectures are compared for their suitability for VLSI implementation of the LMS algorithm:

1. Conventional twos complement binary full-parallel adder/multipliers.
2. Distributed arithmetic variation of (1) using bitwise adders across the filter taps.
3. Redundant arithmetic cells replacing the adders/multipliers of (1).
4. Bit-sequential arithmetic cells replacing the adders/multipliers of (1).

Table 1 lists the relative complexity of each architecture. The conventional architecture has the overwhelmingly highest gate complexity, but also a very short latency. Latency is the waiting interval before actual computed results appear at the output. The conventional architecture is also very general purpose and can support other algorithms much more conveniently. As expected, increasing hardware tends to expand the application base of an architecture. However, no analysis of control unit requirements has been made. A future paper will compare control, communications, and data transport requirements.

It is important to note that the area time figure-of-merit for all architectures applied to the LMS algorithm comply with either eqns (48) or (50). Only the individual values of A_s and T_s will change dependent on the particular technology. Each FOM is very optimistic since the interconnection area between the PEs has not been considered. Such important factors are being studied and will be reported in a future paper.

Acknowledgement—This work has been supported in part by ARO Research Grant #DAA29-83-C-0025.

REFERENCES

1. G. M. Baudet, F. P. Preparata and J. E. Vuillemin, *Area-Time Optimal VLSI Circuits for Convolution*, National Institute of Research and Information, France (1980).

2. J. Vuillemin, A Combinational Limit to the Computing Power of VLSI Circuits, Proc. of the 21st Annual Symposium on Foundations of Computer Science, pp. 294–300 (1980).
3. M. Feuer, Connectivity of Random Logic, *IEEE Trans. Comput.* **C-31**, 29–33 (1982).
4. W. E. Donath, Placement and Average Interconnection Lengths of Computer Logic, *IEEE Trans. Circuits Systems*, **CAS-26**, 272–277 (1979).
5. B. S. Landman and R. L. Russo, On a Pin Versus Block Relationship for Partitions of Logic Graphs, *IEEE Trans. Comput.* **C-20**, 1469–1479 (1971).
6. H. T. Kung, Let's Design Algorithms for VLSI Systems, Proc. of Caltech Conference on VLSI, pp. 65–90 (1979).
7. D. I. Moldovan, On the Analysis and Synthesis of VLSI Algorithms, *IEEE Trans. Comput.* **C-31**, 1121–1126 (1982).
8. C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, Mass. (1980).
9. C. D. Thompson, Area-Time Complexity for VLSI, Proc. of the 11th Annual ACM Conference on Theory of Computing, pp. 81–88 (1979).
10. R. P. Brent and H. T. Kung, The Chip Complexity of Binary Arithmetic, Proc. of 12th Annual ACM Symposium on Theory of Computing, pp. 190–200 (1980).
11. M. Buttner and H. W. Schussler, On Structures for the Implementation of the Distributed Arithmetic, *Nachrichtentech Z.* **29** (1976).
12. F. P. Preparata and J. E. Vuillemin, Area-Time Optimal VLSI Networks Based on the Cube-Connected-Cycles, Rapport de Recherche INRIA, p. 13 (1980).
13. F. P. Preparata and J. E. Vuillemin, Area-Time Optimal VLSI Networks For Multiplying Matrices, *Inf. Processing Lett.* **11**, 77–80 (1980).
14. B. Widrow, J. R. Glover, J. M. McCool, J. Kaunitz, C. S. Williams, R. H. Hearn, J. R. Zeidler, E. Dong, Jr., and R. Goodlin, Adaptive Noise Cancelling: Principles and Applications, *Proc. IEEE* **63**, 1692–1716 (1975).
15. V. A. Vitols, Construction of Adaptive Filters by Use of Subfilters, 13th Asilomar Conf. on Circuits, Systems and Computers, pp. 421–424 (1980).
16. B. Widrow, *Adaptive Filters in Aspects of Network and System Theory* (Edited by R. Kalman and N. DeClaris), pp. 563–587, Holt, Rinehart and Winston, New York (1971).
17. M. A. Soderstrand, Cost and Performance Comparisons of Several Implementations of Adaptive Recursive Filters, 13th Asilomar Conference on Circuits, Systems and Computers, pp. 416–420 (1979).
18. R. L. Reigler and R. T. Compton, Jr., An Adaptive Array for Interface Rejection, *Proc. IEEE* **61**, 748–758 (1975).
19. M. Andrews, The ADF, An Experimental Self-Adaptive Digital Filter, *Comput. Elect. Engng* **8**, 237–244 (1981).
20. M. Andrews, Real Time Adaptive Filters: An LMS Approach, Computers, Electronics and Controls Symposium (1974).
21. Intel Corporation, 2920-16 Signal Processor, AFN 01748A, Santa Clara, California (1980).
22. M. Andrews, Real Time Adaptive Filters II, Sixth Nonlinear and Adaptive Estimation Symposium (1974).
23. M. Andrews, Preprocessors for Real-Time Spectrum Analysis, 92nd Meeting of Acoustical Society of America (1975).
24. M. Andrews, FWLA Noise Effects on the LMS Adaptive Weights, ASSP IEEE Conference (1977).
25. D. Parikh, N. Ahmed, and S. D. Stearns, A Comparison of Two Algorithms for Adaptive IIR Filtering, 13th Asilomar Conference on Circuits, Systems and Computers, pp. 408–411 (1979).
26. M. G. Larimore, Hyperstability and Adaptive Filtering, 13th Asilomar Conference on Circuits, Systems and Computers, pp. 412–415 (1979).
27. C. R. Johnson, M. G. Larimore, J. R. Treichler, and B. D. O. Anderson, SHARF Convergence Properties, *IEEE Trans. Acoust., Speech, and Signal Processing* **ASSP-29**, 659–669 (1981).
28. P. A. Thompson, An Adaptive Spectral Analysis Technique for Unbiased Frequency Estimation in the Presence of White Noise, 13th Asilomar Conference on Circuits, Systems and Computers, pp. 529–533 (1979).
29. D. M. Etter and S. D. Stearns, Convergence Properties of the Adaptive Delay Element in Delay-Lock Loops and Frequency Tracking, 13th Asilomar Conference on Circuits, Systems, and Computers, pp. 534–537 (1979).
30. *ADAP, Automatic Digital Audio Processor*, Rockwell International, Anaheim, California (1978).
31. M. E. Hoff, Jr., IC Technology: Trends and Impact on Digital Signal Processing, IC ASSP 80 Proc., pp. 1–6 (1980).
32. S. White, An Adaptive Recursive Digital Filter, 9th Annual Asilomar Conf. on Circuits, Systems and Computers (1975).
33. R. D. Gitlin and J. S. Thompson, A New Structure for Adaptive Digital Echo Cancellation, Proc. of National Telecommunications Conference (1976).
34. S. D. Stearns, Error Surfaces of Recursive Adaptive Filters, *IEEE Trans. Circuits Systems* **CAS-28** (1981).
35. P. L. Feintuch, An Adaptive Recursive LMS Filter, *Proc. IEEE* **64**, 1622–1624 (1976).
36. B. Widrow, Adaptive Switching Circuits, IRE Wescon Convention Record, Part 4, pp. 96–104 (1960).
37. T. Soderstrom, L. Ljung, and I. Gustavsson, A Theoretical Analysis of Recursive Identification Methods, *Automatica* **14**, 231–244 (1978).
38. M. D. Ercegovac, A General Hardware-Oriented Method for Evaluation of Functions and Computations in a Digital Computer, *IEEE Trans. Comput.* **C-26**, 667–680 (1977).
39. A. Peled and B. Liu, A New Hardware Realization of Digital Filters, *IEEE Trans. Acoust., Speech, Signal Processing* **ASSP-22**, 456–462 (1974).
40. R. Roberts, Digital Signal Processing Structures for VLSI, Univ. Southern California Workshop on VLSI and Modern Signal Processing, pp. 83–88 (1983).
41. C. F. Cowan, S. G. Smith, and J. H. Elliott, A Digital Adaptive Filter Using a Memory-Accumulator Architecture: Theory and Realization, *IEEE Trans. Acoust., Speech, Signal Processing* **ASSP-31**, 541–549 (1983).

42. C. S. Burrus, Digital Filter Structures Described by Distributed Arithmetic. *IEEE Trans. Circuits Systems* **CAS-24**, 674–680 (1981).
43. T. A. C. M. Claasen, W. F. G. Mecklenbrauker, and J. B. H. Peek, Some Considerations on the Implementation of Digital Systems for Signal Processing. Philips Res. Rep. #30, pp. 73–84 (1975).
44. H. J. Sips, Bit-Sequential Arithmetic for Parallel Processors. *IEEE Trans. Comput.* **C-33**, 7–20 (1984).
45. A. Avizienis, Signed Digit Number Representation for Fast Parallel Arithmetic. *IRE Trans. Electron. Comput.* **EC-10**, 389–400 (1961).
46. J. E. Robertson, A New Class of Digital Division Methods. *IRE Trans. Electron. Comput.* **EC-7**, 218–222 (1958).
47. J. O. Campeau, Communication and Sequential Problems in the Parallel Processor, in *Parallel Processor System, Technologies and Applications* (Edited by L. C. Hobbs) Spartan Press, New York (1970).
48. E. L. Braum, *Digital Computer Design–Logic, Circuitry, and Synthesis*. Academic, New York (1963).
49. K. Trivedi and M. D. Ercegovac, On-Line Algorithms for Division and Multiplication, *IEEE Trans. Comput.* **C-26**, 681–687 (1977).
50. M. D. Ercegovac, A General Method for Evaluation of Functions and Computations in a Digital Computer, Ph.D. dissertation #750, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana (1975).
51. C. Y. Chow, A Variable Precision Processor Module, Proc. IEEE Int. Conf. on Computer Design (1983).

PARALLEL IMPLEMENTATION OF THE LMS ALGORITHM

by
M. Andrews
J.S. Walicki

Submitted to
1985 International Conference on Parallel Processing
St. Charles, Illinois
August 20-23, 1985

Co-sponsored By
The Pennsylvania University
IEEE Computer Society
Association for Computing Machinery

Space Tech Corporation
2324 Manchester Court
Fort Collins, CO 80526
303 484-9903

Parallel Implementation of the LMS Algorithm

M. Andrews and J. S. Walicki[1]

Abstract: Fast and accurate adaptive filtering implemented in the digital hardware is necessary in many areas of communication and general signal processing. This paper discusses the realization of the least mean square algorithm for adaptive filtering on a rectangular systolic array. Two such arrays are needed for six coefficient, twelve-bit adaptive filter. Extension to a larger filter, or higher resolution can be easily accomplished by cascading processing arrays. The implementation proposed here is oriented towards the 72-processor array manufactured by the NCR Corporation. Efficient utilization of the processing array is accomplished by proper feeding of input data arriving from a sensor array. Processing is performed at the bit level and the vector operations of the original LMS algorithm become matrix operations. The processing is divided into two phases - compute and update. In the first phase a new output of the filter is computed, and filter cofficients are updated according to the LMS rule in the next phase. Effects of the finite-word length are discussed but with 12-bit representation of signals these effects are not severe.

1 Introduction

1.1 Adaptive Filtering and the LMS Algorithm

It is well known that any transmission of information is vulnerable to degradation caused by various types of interferences (noises). These noises may be accidental like the RF interference or intentional like electronic countermeasures. In any case, the reduction of interference is important.

Classical techniques aimed at increasing a signal-to-noise ratio (SNR) usually employ information derived from a single signal sensor. No additional information is provided, other than the signal and perhaps statistical properties of both signal and noise. However, chances for the restoration of original signal can be increased if multiple measurements of the signal wave are available. An array of sensors provides such an opportunity. Such a sensor array attracted our interests because it generates a vector of signal components which can be processed in parallel.

[1]J. Walicki is with Computer Science Department, Colorado State University, Fort Collins, CO 80523; M. Andrews is with Space Tech Corp., Fort Collins, CO 80526

This study was sponsored by the U.S. Army Research Office under grant DAAG29-83-C-0025. The U.S. Government assumes no responsibility for the information presented

The goal of such processing is well defined – produce a signal as close the original signal as possible (allowing for the least mean square error, for example) without prior knowledge of the signal/noise properties. This goal can be achieved if the processing is self-optimizing, that is, if processing parameters are continuously modified so that the error criterion mentioned above is satisfied. An adaptive array is therefore an array of sensor elements plus an adaptive control algorithm.

The implementation of the adaptive control algorithm can be viewed as a problem of realizing a normal digital filter with changing filter weights. Several performance measures exist and the nature of the adaptive algorithm depends on the selection of particular measure. Adoption of the mean square error for the criterion of optimality leads to the elegant Wiener solution for optimal weights.

If an array produces output y(t) which is obtained by weighing input vector $\mathbf{x}(t)$ then an error signal is a difference between a reference signal d(t) and the output y(t):

$$\epsilon(t) = d(t) - \mathbf{w}^T\mathbf{x}(t) \tag{1}$$

The expected value of the squared error is then:

$$E(\epsilon^2(t)) = \overline{d^2(t)} - 2\mathbf{w}^T\mathbf{r}_{xd} + \mathbf{w}^T\mathbf{R}_{xx}\mathbf{w} \tag{2}$$

where $\mathbf{R}_{xx}$ is the autocorrelation matrix of input signal, and $\mathbf{r}_{xd}$ is the correlation vector of input and reference signals. If $d(t) = s(t)$ then $\overline{d^2(t)} = S$ which is a signal power. It can be shown that the value of $\mathbf{w}$ which minimizes $E(\epsilon^2(t))$ must satisfy the Wiener-Hopf equation:

$$\mathbf{w}_{opt} = \mathbf{R}_{xx}^{-1}\mathbf{r}_{xd} \tag{3}$$

For the adaptive array described above the performance measure (MSE) is a quadratic function of the weights. Therefore the performance measure is a bowl-shaped surface and the goal of the adaptive processor is to find a bottom of that bowl. It can be accomplished by any 'hill climbing' method. The idealized version of the 'hill climbing' is the method of steepest descent which assumes that the statistics of the signal environment are perfectly known. In many practical situations, however, the signal statistics are stationary but unknown. For such problems the Widrow's Least Mean Square (LMS) algorithm described in the next section is particularly useful (Monz80).

1.2 Implementations of Digital Filters and the LMS algorithm

Filtering by means of digital hardware offers many advantages such as repeatability and controlability, but also presents a problem of fast and efficient execution, especially if implemented on a classical SISD digital processor. Attempts at speeding-up the processing had been directed at improving the execution efficiency via nontraditional arithmetic such as the residue number system or the distributed arithmetic (Pele74,Cowa81,Cowa83).

Efficient realization of digital filters aimed at increasing speed and reducing power consumption has been investigated by Peled (Pele74). He proposed storing a finite number of results of intermediate arithmetic operations of a filter, and using them to obtain output samples. In that scheme

only additions and shifts are necessary. Elimination of multipliers speeds up and simplifies the execution. The intermediate results are stored in ROM which is addressed by proper combination of bits of the digitized input signal. Cowan and others (Cowa81,83) proposed implementation of an adaptive filter using the distributed arithmetic. Again the major increase in execution speed is obtained by prestoring all possible intermediate results, and the ROM is substituted by RAM since updated weights of the adaptive filter affect the prestored intermediate results. Both Peled and Cowan deal with single input stream.

Ercegovac (Erce77)proposed a fast computational method which exploits the redundant number representation system. The method utilizes only fixed-point additions without overflow, and therefore, no roundoff errors occur. However, errors due to the finite precision representation have to be handled by extending the working precision. The E-method, as it is called, generates one digit of each of the elements of the solution vector in one step, starting with the most significant digits. The main feature of this method is fast evaluation of polynomials, rational functions and arithmetic expressions in a fixed-point number representation system. The redundant arithmetic has been used in the digit slice proposed in (Chow83). The cell is a three-level digit slice capable of implementing basic arithmetic operations. The digit slice is configured as a linear one-dimensional array. Results appear at the bottom of each cell. It is worth noting that the basic LMS algorithm implemented with these cells can operate on the most significant digit first. Hence , this computational feature makes best use of an analog-to-digital converter which produces the most significant digit first.

In what follows we propose a fully parallel (SIMD) architecture for realization of an adaptive filter array which processes multiple input streams. This approach became feasible with advent of processor arrays (also called systolic arrays). This presentation describes a proposed architecture in general terms. However, the 72 processor array manufactured by NCR directly realizes the structure presented here.

Section 2 describes the discrete LMS algorithm from the point of view of processor array implementation, and proposes basic phases of processing. In the section 3 implementation of the basic operation phases is discussed. Section 4 deals with effects of the finite wordlength. Section 5 offers a summary and conclusions.

2 Parallel LMS algorithm

An array of sensors generates vectors of samples in response to a signal wavefront entering the array. This input vector is then fed into the pattern forming array where every input sample is scaled by the appropriate weight factor. Contributions of individual weighted samples are summed yielding the single output sample:

$$y(k) = \sum_{i=1}^{n} w_i x_i(k) \tag{4}$$

The error sample e(k) is then obtained:

$$e(k) = d(k) - y(k) \tag{5}$$

The error value is scaled by the convergence factor and the resulting value is used in the LMS rule of coefficients update:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + 2\Delta_s e(k)\mathbf{x}(k) \tag{6}$$

where Δ_s is a scalar constant controlling rate of convergence and stability. The equation above offers a simple prescription for the weight adjustment – the new weight vector is determined by adding the input signal vector scaled by the error value to the old weight vector. It can be shown that after a sufficient number of iterations the LMS solution converges to the Wiener solution.

It is possible to partition the LMS adaptation rule into the basic operations suitable for the processor array implementation.

Operation 1: weight input vector $\mathbf{w}^T\mathbf{x}$

Operation 2: sum partial products to get y(k)

Operation 3: compute error e(k)

Operation 4: scale error $2\Delta_s e(k) = C$

Operation 5: update weights $\mathbf{w} \leftarrow \mathbf{w} + C\mathbf{w}$

It can be seen from the above list of operations that they can be grouped into two distinct phases – compute and update. It is worth noting that this grouping is not only a matter of convenience but also that there exists an underlying reason for such an arrangement. The operation of weighting the input vector is a convolution like operation. This fact places specific restrictions upon the system under consideration. Namely, the system has to be linear and shift(time)-invariant. In practice it means that the weight coefficients must not change once filter convergence is reached. Therefore, the computations are performed with constant coefficients which change only during the update phase.

The basic processing steps listed above require a variety of elementary matrix operations. Operation 1 requires multiplication of two vectors. Operation 5 calls for multiplication of a vector by scalar and for addition of two vectors. Operations 2, 3 and 4 require operations on scalars. The values to be operated on are represented by k-bit words. Existing processor (systolic) arrays are serial architectures, mainly because it is still prohibitively expensive to build a fully parallel single-wafer multi-processor arrays. Therefore, the operations mentioned above have to be performed at the bit level. As the result the vector operations become operations on matrices of binary representations of signal samples!

Efficient use of a processor array is thus important in order to make up for losses caused by the use of serial arithmetic. As Urquhart and Wood (Urqu84) show, array utilization depends on properly feeding in samples to be processed. Particularly, if one matrix of arguments is kept static on the processor array and the other matrix is entered properly skewed, then the array is 100% efficient. In our case it is quite natural to keep the coefficients $\mathbf{w}$ fixed in the array, while bit-streams of input samples $\mathbf{x}$ march in from array sensors. The arrangement described above is a basis for the implementation of the first operation, namely formation of the inner product $\mathbf{w}^T\mathbf{x}$.

The second operation, that of summing partial products in order to obtain a value of the output sample y(t), is accomplished by using an adder tree. If weight coefficients and input samples interact

in such a way that the least significant bits are processed first, then the output from the adder tree is a serial stream of bits with the least significant bit leading.

In the third step, the error sample is generated by subtracting the y stream from the d stream. The d stream is a serialized sample of a reference signal.

The computed value of the error is scaled in the next operation (4). This is accomplished easily by properly shifting the e(k).

The remaining fifth step is the most complex, since it involves multiplying a vector by a scalar and addition of vectors (bit matrices). The first operation in this step multiplies the weight vector w by the scaled error e(k). Since all weights are scaled by the same scalar value it merely requires a simple shifting of old weights by a number of places prescribed by the value of $2\triangle_{,}e(k)$. This operation can be performed in a separate processor array with original weights doubled just for the purpose of that processing. Or, given a sufficiently powerful processing element (as we will see in the next section), it is possible to scale weights *in situ*. The last step requires us to add the old weights to the scaled values obtained in the previous operation. Again, given an appropriate processing element (like the processor array mentioned in the introduction), this operation can be done in the original array. These steps complete the update phase and a new input sample is processed as in the first step.

The next section contains a detailed description of the architecture of the LMS adaptor. This implementation is general but can be easily implemented in the NCR's 72 processor array.

3 Architecture of the LMS Processor

This section describes the overall architecture of the LMS processor. The heart of the system is a processor array whose main functions are: the storage of adaptive coefficients (weights) and generation of an output sample y(k).

Without loss of generality let us assume that the input vector is generated by a six-element sensor array, and that both input samples and weight coefficients are represented as 12-bit quantities. It can also be assumed that these quantities are positive and less then 1.0. Extension to the two's complement representation is rather straightforward (Cowa81,McCa84).

These assumptions allow direct implementation of the 72-processor (6 x 12) array manufactured by the NCR Corp. The Geometric Arithmetic Parallel Processor (GAPP-II) is a rectangular systolic array processor chip which can be cascaded in both north/south and east/west orientations. Each 1-bit processor element (PE) can communicate with its four neighbors. Each PE consists of a bit-serial ALU, 128 x 1 individually addressable RAM and 4 single bit latches. The I/O latch allows communication through the PE without interrupting the ALU, the remaining latches hold inputs to the ALU. The GAPP operates as a SIMD machine. That is, instructions are broadcast to each cell from an external control store, loaded in turn from the host computer. Proper address sequencing can be provided by any general address sequencer like the microprogrammable AMD 2910 sequencer. The instructions directed to the processing elements consist of a 13-bit control field which specifies the array connectivity and arithmetic/logic operations, and 7-bit RAM address. These instructions can be sent to the GAPP array at the rate of 10 MIPS. The array has a global

broadcast of input data and a global output. One GAPP array device can function as a modular component in a larger array, thus enabling word or bit-length growth of an array as needed. In LMS applications it may be desirable to increase the number of coefficients, which can be achieved by simply cascading another array expanding a filter to 12 coefficients. Likewise, cascading one array in the direction of input stream, increases the wordlength from 12 to 24 bits. (Davi84,Gapp84).

The following diagram schematically shows operation of the first processor array which contains coefficients in RAM locations and computes $\mathbf{w}^T\mathbf{x}$.

$$
\begin{array}{cccccccccccccccccccc}
w_0^0 & w_0^1 & w_0^2 & .. & w_0^9 & w_0^{10} & w_0^{11} & x_0^{11} & x_{10}^4 & x_0^9 & .. & x_0^2 & x_0^1 & x_0^0 & & & & & & \\
w_1^0 & w_1^1 & w_1^2 & .. & w_1^9 & w_1^{10} & w_1^{11} & \leftarrow & x_1^{11} & x_1^{10} & x_1^9 & .. & x_1^2 & x_1^1 & x_1^0 & & & & & \\
w_2^0 & w_2^1 & w_2^2 & .. & w_2^9 & w_2^{10} & w_2^{11} & & \leftarrow & x_2^{11} & x_2^{10} & x_2^9 & .. & x_2^2 & x_2^1 & x_2^0 & & & & \\
w_3^0 & w_3^1 & w_3^2 & .. & w_3^9 & w_3^{10} & w_3^{11} & & & \leftarrow & x_3^{11} & x_3^{10} & x_3^9 & .. & x_3^2 & x_3^1 & x_3^0 & & & \\
w_4^0 & w_4^1 & w_4^2 & .. & w_4^9 & w_4^{10} & w_4^{11} & & & & \leftarrow & x_4^{11} & x_4^{10} & x_4^9 & .. & x_4^2 & x_4^1 & x_4^0 & & \\
w_5^0 & w_5^1 & w_5^2 & .. & w_5^9 & w_5^{10} & w_5^{11} & & & & & \leftarrow & x_5^{11} & x_5^{10} & x_5^9 & .. & x_5^2 & x_5^1 & x_5^0 &
\end{array}
$$

The w_j^i's are bits of a digital representation of the weight coefficient w_j which are stored in the RAM locations of the individual processing elements. Partial products appear on the bottom edge of the array, represented here by w_5^i bits. Every processing element performs the following basic operations:

$$x_{west} \leftarrow x_{east}$$

$$y_{south} \leftarrow y_{north} + x_{east} z_{accumulated}$$

If the least significant bits of $\mathbf{x}$ and $\mathbf{w}$ interact first, then partial products of equal significance leave the edge of the array at the same time. These partial products can be accumulated using the adder tree constructed from MSI adders. If, however, the least significant bit x_j^0 interacts with the most significant bit of $\mathbf{w}$ then, the partial products of equal significance appear skewed and the linear chain of full adders can be used to accumulate the final output. If the second option is chosen then a single row of PE's of a second processing array can be used to accumulate partial products. In order to handle properly the operations in the main array guard bands of $log_2 n$ bits are appended to the input parallelogram.

The second array implements operations 2, 3 and 4. That is, the computations of the output sample y(k) and the scaled error $2\Delta_e e(k)$. As it was mentioned above, the first step is to sum partial products arriving from the main processing array. These partial products are accumulated in the elements of the first row. Because of the additional skewing of the partial products leaving the first array, the delay is needed between the accumulating PE's. In the GAPP device this can be easily accomplished by storing the elementary sums in the local RAM locations. Properly accumulated result y(k) leaves the east processing element and is fed into the third row of processing elements of the second array (see Fig.1). This is the end of the compute phase. The remaining processing is devoted to updating the coefficient weights.

An error value e(k) is computed by subtracting the output sample from the reference sample d(k) which is shifted into the fourth row of the array simultaneously with the loading operation of the third row. The result of this operation remains in the fourth row and is shifted according to

the value of $2\Delta_e$. The quantity obtained in this step is the scaled error value and it is sent to the host controller.

The controller uses this value to scale (shift) the original coefficients residing in the main array. However, before this operation is performed the original weights must be saved in the RAM locations. The scaled weights are also uploaded into the local RAM, and then both quantities are summed ($\mathbf{w} \leftarrow \mathbf{w} + 2\Delta_e\mathbf{w}$) yielding the new updated coefficients, which are used to compute a new output sample, and the computation processes is repeated.

4 Effects of the Finite Wordlength

The derivation of the LMS algorithm presented in the previous sections assumes the infinite precision arithmetic. Under this assumption the only source of inaccuracy is the fundamental principle of the algorithm. That is, the fact that the gradient which steers towards the optimal solution cannot be computed accurately, but it has to be estimated. However, practical implementations have to deal with effects of the limited wordlength, especially those that use relatively few bits (8-16).

Effects of the finite wordlength manifest themselves in various stages of processing, or even before the actual processing begins as is the case with the quantization noise. Assuming the 12-bit representation, it can be shown that the input signal-to-noise ratio is around 70 dB. If the noise is white and the processing is realized without an error, then the mean and the variance of output noise are:

$$m_y = m_x \sum_{n=-\infty}^{\infty} h(n) \tag{7}$$

$$\sigma_y^2 = \sigma_x^2 \sum_{n=\infty}^{\infty} |\, h(n)\,|^2 \tag{8}$$

where h(n) is the impulse response of the linear filter (Oppe75).

In the real implementation of the LMS algorithm there exist two types of noise. The gradient noise caused by estimating the gradient with a finite amount of input data, and noise caused by imprecise results of intermediate processing operations due to the finite wordlength. The operations of addition and multiplication at each weight adaptation cycle are affected by random errors e_a and e_m. Additionally, the iterative nature of the algorithm causes noise errors to accumulate during every compute/update cycle:

$$\mathbf{w}_{j+1} = \mathbf{w}_j + 2\Delta_e e(j)\mathbf{x}_j + \sum_{i=1}^{j} e_{m_i} + \sum_{i=1}^{j} e_{a_i} \tag{9}$$

Let's define

$$\mathbf{w}'_j = \mathbf{w}_j + \sum_{i=1}^{j} (e_{m_i} + e_{a_i}) \tag{10}$$

Then

$$\mathbf{w}'_{j+1} = \mathbf{w}'_j + e_{m_{j+1}} + e_{a_{j+1}} + 2\Delta_e e(j)\mathbf{x}_j \tag{11}$$

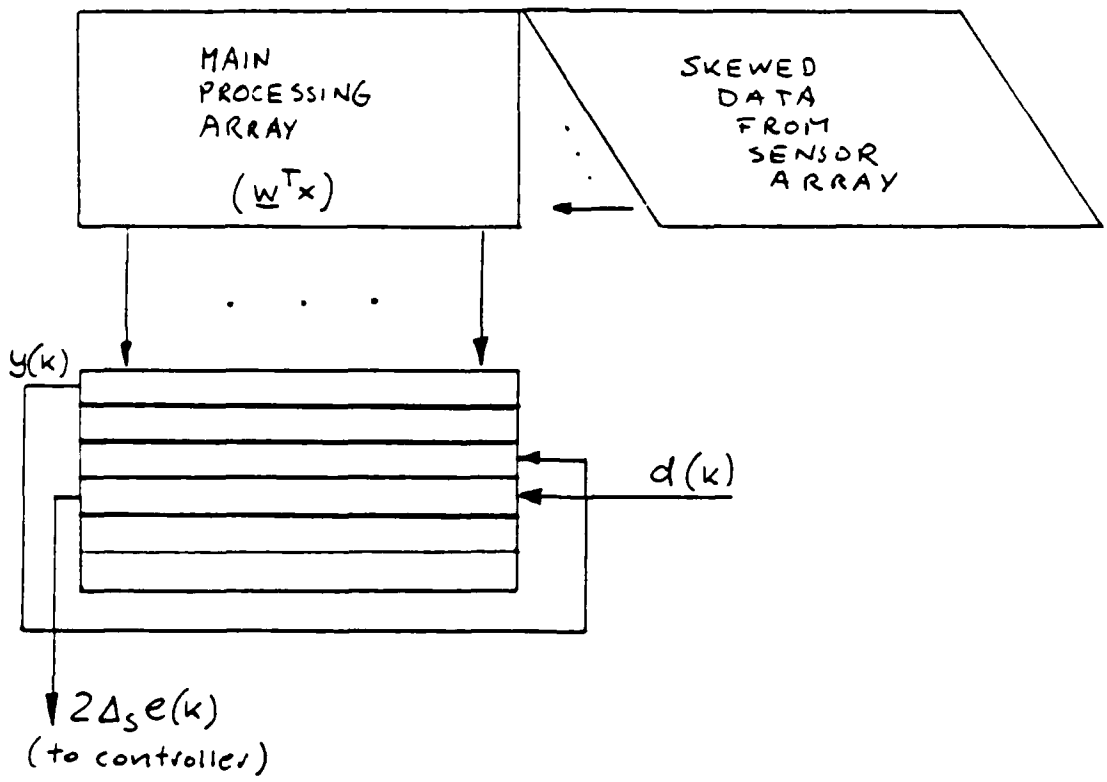


Fig.1 General Flow of Processing

It can be safely assumed that the errors e_a and e_m are uncorrelated (Andr77,Andr83). The computation noise causes deviation of updated weights from the optimal values $\mathbf{w}_{opt}$ resulting in the difference

$$\mathbf{v}'_j = \mathbf{w}'_j - \mathbf{w}_{opt} \tag{12}$$

The $j+1$ difference is then

$$\mathbf{v}'_{j+1} = \mathbf{w}_j + 2\Delta_s e(j)\mathbf{x}_j + e_{m_{j+1}} + e_{a_{j+1}} - \mathbf{w}_{opt} \tag{13}$$

And finally from 12:

$$\mathbf{v}'_{j+1} = \mathbf{v}'_j + 2\Delta_s e(j)\mathbf{x}_j + e_{m_{j+1}} + e_{a_{j+1}} \tag{14}$$

Equation 14 accounts for both the gradient estimation noise and the computational noise. Computer simulations based on the presented model lead to the following conclusions (Andr78). The LMS algorithm can be successfully implemented in a short digital word (8-12 bits) but large values of the convergence factor $2\Delta_s$ are needed to ensure convergence and unbiased behavior of the adaptive filter. The testbed realization of the 8-bit adaptive filter by Cowan and others (Cowa83) also confirms these conclusions. Therefore the 12-bit realization of the LMS on the systolic array should not suffer from the relatively short wordlength.

5 Summary and Conclusions

We have presented the systolic implementation of the LMS algorithm. The proposed architecture is fully parallel and operates on the bit level which efficiently exploits the serial representation of data. Even in the basic structure of two elementary arrays (GAPP devices) the relatively high resolution of 12 bits is achieved. Cascading more devices allows for increase in the resolution and a filter size. Fast and accurate adaptive filtering by means of the digital hardware becomes more important as the communication and intelligence needs increase. For example the array processor like the one described here may digitally control individual radiating elements of large phased array radars and the same time digitally enhance desired signals by eliminating clutter and jamming signals.

The GAPP architecture is ideally suited to distributed arithmetic implementations of many signal processing algorithms. Our LMS filter is more demanding than fixed-coefficient filters because weights change with each iteration. The local memory of the GAPP array allows storing and updating the filter coefficients. In conclusion, GAPP devices add a new dimension to parallel implementations.

6 References

(Monz80) Monzingo R. and Miller T., Introduction to Adaptive Arrays, J. Wiley and Sons, 1980.

(Pele74) Peled A. and Liu B., 'A New Hardware Realization of Digital Filters', IEEE Trans. on ASSP, vol. ASSP-22, No.6, Dec. 1974

(Cowa81) Cowan C.F.N and Mavor J.,'A New Digital Adaptive Implementation Using Distributed-Arithmetic Techniques', IEE Proc.,vol.128,Pt.F,No.4,Aug.1981.

(Cowa83) Cowan C.F.N, Stewart G.S., and Elliot J.H.,'A Digital Adaptive Filter Using a Memory-Accumulator Architecture: Theory and Realization', IEEE Trans. on ASSP, vol.ASSP-31,No.3,June 1983.

(Urqu84) Urquhart R.B. and Wood D.,'Systolic Matrix and Vector Multiplication Methods for Signal Processing,' IEE Proc.,Vol.131,Pt.F,No.6,Oct.1984

(McCa84) McCanny J.V.,McWhirter J.G. and Wood K.,'Optimised Bit Level Systolic Array for Convolution,' IEE Proc.,Vol.131,Pt.F,No.6,Oct.1984.

(Davi84) Davis R.,Thomas D.,'Systolic Array Chip Meets the Demands of Heavy-Duty Processing,' Electronic Design, Oct.1984.

(Gapp84) NCR Technical Note NCR45CG72, Geometric Arithmetic Parallel Processor,NCR Microelectronics Div.,Ft.Collins,CO 80525, 1984.

(Oppe75) Oppenheim A. V. and Schafer R. W., Digital Signal Processing, Prentice-Hall 1975.

(Andr77) Andrews M. and Fitch R.,'Finite Wordlength Arithmetic - Computational Error Effects on the LMS Adaptive Weights,' ICASSP 1977.

(Andr83) Andrews M.,'Computational Noise Effects on Adaptive Filter Algorithms,' Comput. & Elect. Engng. vol 10, no.1,pp.29-40,1983.

(Erce77) Ercegovac M.D.,'A General Hardware-Oriented Method for Evaluation of Functions and Computations in a Digital Computer,' IEEE Trans. on Comp.,Vol. C-26,No.7,July 1977

(Chow83) Chow C. Y., 'A Variable Precision Processor Module,' Proc. 1983 IEEE Int. Conf. on Computer Design.

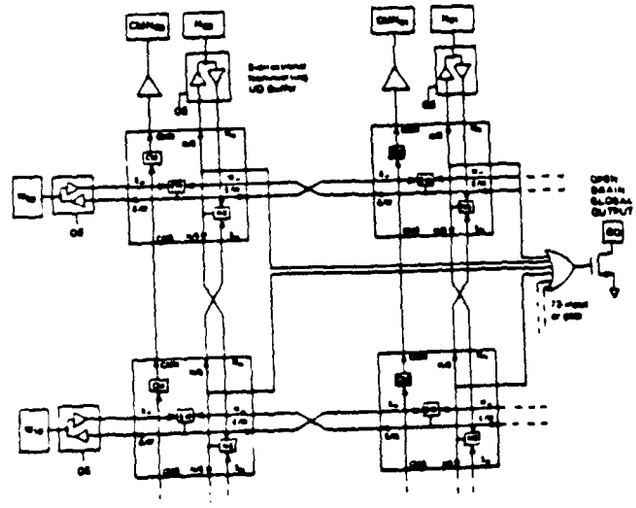


Fig.1 Organization of GAPP array

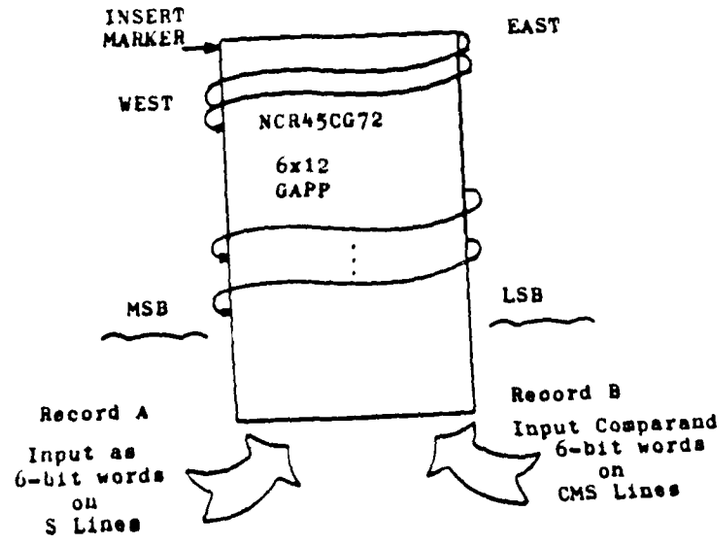


Fig.2 Data Base Machine

FINAL PROGRAM
AND
PAPER SUMMARIES FOR
THE 1984 DIGITAL SIGNAL PROCESSING WORKSHOP

OCTOBER 8-10, 1984
CHATHAM BARS INN
CHATHAM, MASSACHUSETTS

Sponsored by
IEEE Acoustics, Speech and Signal Processing Society

original

modified

reconstructed

COMPARATIVE VLSI IMPLEMENTATIONS OF THE LMS ALGORITHM

Dr. Michael Andrews
Space Tech Corporation
2324 Manchester Court
Fort Collins, Colorado 80526
303-484-9903

Alternative arithmetic engines are being analyzed to determine applicability to adaptive signal processors. Synchrony, topology, and granularity are important design figures of merit for any VLSI implementation. In this paper, redundant arithmetic is considered with the desirable attributes, basically because carry-free operations are possible resulting in a speed-up. A primitive computational element is discussed for the least-mean-square (LMS) algorithm embedded in redundant arithmetic processors.

The basic approach has been to analyze the hardware/software trade-offs of conventional (von Neumann) and non-conventional (parallel, pipeline, vector, array, and custom processors) in an attempt to identify optimal structures (of order area x time) which are computationally fast yet flexible. Regular and simple interconnections and geometries among high-performance parallel structures were sought. A two step process can be assumed; first the sequential algorithms are speeded up (seeking inherent parallelism) and second, fast algorithms are to be mapped onto new VLSI architectures (via recursion and pipelining). The current research is to provide theoretical design tools and interconnection strategies capable of achieving real-time implementation of adaptive algorithms via limited user-programmable mechanisms (e.g., firmware).

The traditional criteria of component count, whether applied to processors or to simpler devices, are no longer adequate to establish a scale of comparison among various solutions to a given VLSI problem. Indeed, number-of-elements criteria are substantially based on the fact that processing elements and their interconnection are realized by different media. This difference disappears in VLSI, which "integrates" both processing elements and their interconnections in a two-dimensional geometry. As a result, the VLSI solution to a given computational problem involves the conception of an interconnection architecture, its layout, and the design of an algorithm for that architecture. It is useful to examine the trade-offs between the production cost (area) and the incremental cost (time) of a dedicated circuit developed to solve that problem. The area of a chip can be partitioned into interconnect or wire area, A(w), gate area, A(g), and wire pad area, A(p). At least for the class of transitive functions (cyclic shifts, matrix product, integer product, and linear transforms), Vuillemin has shown such VLSI circuits satisfy

$$A > A(g)N + A(p)B + A(w)B \qquad \text{where in practice } A(w) \gg A(g), A(p)$$

where A is the chip area, N is the number of inputs, and B is the average bandwidth (bits per second passing through the chip pins).

A good arithmetic unit for digital signal processing, in addition to the above, should have the following:

1. As modules, the ALU number of modules and the precision of the operands should not affect the time to perform one arithmetic step such as addition, subtraction, multiplication, and division.
2. Each digit produced should not be dependent on very many adjacent digits to eliminate excessive carry propagation and hardware error checking. "Column" operations help detect and correct hardware errors independently.
3. Round-off error from truncation should have no bias.

4. The leading digit or sign bit should be processed just like any other digit.
5. Overflow should be detected early. At best, overflow could be detectable whe the first few digits or leading digits are generated.

Points two and five should not be taken lightly. Signal processing application frequently require robust implementations which can operate in harsh environments Faulty units need to be self-checking and correcting. Furthermore, the wide dynami range of signal inputs as found in radar often causes overflow in conventional two' complement engines, thereby invoking elaborate dynamic scaling operations Detecting the overflow at the onset of the most significant digit productio minimizes the scaling overhead and subsequent pipeline flushes.

The redundant number digit slice proposed in [1] and cascaded as 4-digit slice into a module incorporates all of the attributes listed above. Depicted in Fig. is a 15-digit module with three levels. The function of each level is described i Table 1.

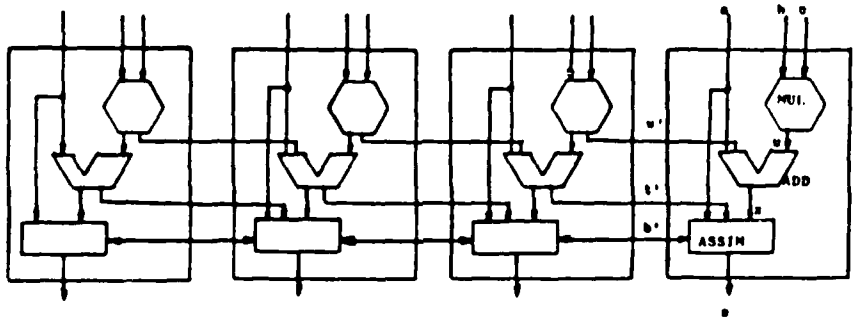


Fig. 1 A 15-Digit Redundant Number Module

Table 1 Digit Slice Functions

Multiplication		ru + w <— bc
Addition		rx + t <— a + w' + u
	or	rx + t <— a - w' - u
Assimilate		-rb" + s <— a - b'
	or	s <— t' + x

The digit slices can be configured easily into VLSI. The inputs are a, b, and c. The output is s. Interslice transfers include w, t, and b' for eastward direction and w', t', and b" for westward direction. All of these are intermediate results used in adjacent digit positions. The assimilate function assists in the conversion from redundant to non-redundant number representation. It has already been shown that a digit slice with 400 gates constructed with 16 cascaded sum-of-products circuits can implement redundant arithmetic in radix-16 with 10 as the maximum digit [1]. Current research is exploring the processing delay, latency, and execution time of the redundant number system processors [2].

[1] C. Chow, "A Variable Precision Processor Module," Proceedings of the 1983 IEEE International Conference on Computer Design.
[2] M. Andrews, "Comparative Implementations of the LMS Algorithm," to be published in International Journal of Computers and Electrical Engineering, 1984.

This work was sponsored by Space Tech Corporation under a grant from the Army Research Office (#DAA29-83-C-0025).

END

10-87

DTIC