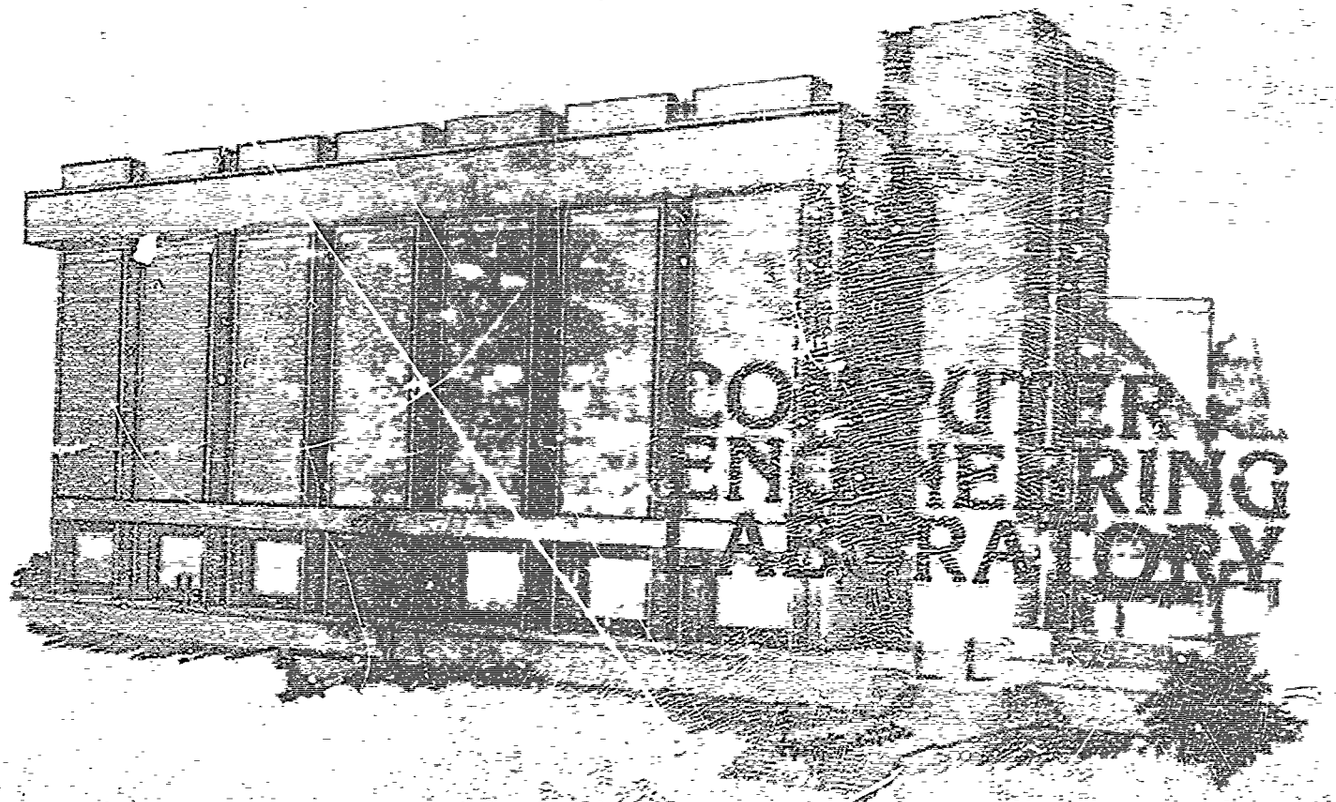
A NEW MULTIPLIER STRUCTURE FOR DIGITAL SIGNAL PROCESSING

Fred J. Taylor

AFOSR-TR- 80 - 1321

A NEW MULTIPLIER STRUCTURE FOR DIGITAL SIGNAL PROCESSING



DR. FRED J. TAYLOR

Department of Electrical & Computer Engineering
University of Cincinnati
Cincinnati, Ohio 45221
U.S.A.

SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED

(12) REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER (14) AFOSR-TR-80-1321	2. GOVT ACCESSION NO. AD-A093 430
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) (6) A NEW MULTIPLIER STRUCTURE FOR DIGITAL SIGNAL PROCESSING	5. TYPE OF REPORT & PERIOD COVERED (9) FINAL rept.
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Dr. Fred J. Taylor	8. CONTRACT OR GRANT NUMBER(s) (15) F49620-79-C-0066
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Cincinnati, Department of Electrical and Computer Engineering, Cincinnati, OH 45221	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F (16) 2304/A6 (17) A6
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research/NM, Bolling AFB, DC 20332 (13) 168	12. REPORT DATE (11) 1980
	13. NUMBER OF PAGES 165
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	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)

18. SUPPLEMENTARY NOTES

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

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

The charge of this AFOSR grant ~~F49620-79-C-0066~~ was the study of a new class of memory intensive digital arithmetic units based on modular algebra. The new class of arithmetic units, developed under this grant, operate at very high speeds, admit VLSI and bit-slice realizations, and can be integrated into digital signal processing systems.

410-114

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

FINAL REPORT

AFOSR Grant F49620-79-C-0066

Title: A New Multiplier Structure For Digital Signal Processing

Principal Investigator: Dr. Fred J. Taylor
Department of Electrical and
Computer Engineering
University of Cincinnati
Cincinnati, Ohio 45221

Grant Monitor: Dr. J. Bram
AFOSR/NM
Bolling Air Force Base, DC
Washington, DC 20332

TABLE OF CONTENTS

INTRODUCTION

The charge of AFOSR Grant F49620-79-C-0066 was the study of a new class of memory intensive digital arithmetic units based on modular algebra. The new class of arithmetic units, developed under this grant, operate at very high speeds, admit VLSI and bit-slice realizations, and can be integrated into digital signal processing systems.

Numerous authors have demonstrated the potential of residue arithmetic for realizing high speed signal processing and computational systems.[1,6] These methods are memory intensive in that the table lookup operations are used to perform modular arithmetic. However, there is a possible flaw in this contemporary residue arithmetic work and it is our dependence on high speed memory. Admittedly, memory is becoming available with higher densities and access speeds. However, they present a non-trivial power demand on the system and are very expensive. For example, Intel's HMOS 1K x 4 memory, having access times of 55, 70, and 80 ns cost on the order of $82, $76, and $62 per copy. INMOS 16K (4k x 4) static RAM is available at 35 ns and Fairchild markets a 1K ECL (high power dissipation) RAM at higher per unit costs[i]. Our research has determined that by using K x 1 HMOS memories, an equivalent 12 x 12 RNS multiplier could be configured which has a pipelined throughput of 35 ns, but it would require 9.9 watts of active power and 1.65 watts standby. Furthermore, the cost per moduli would exceed $1,000. Therefore, high performance residue based signal processing systems may carry a high price tag as well. It may therefore be wise to rethink our dependence on memory intensive arithmetic.

Footnote [i]: This condition will be strongly influenced by the results of DOD's VHSIC Program.

It would seem advantageous to architect future residue arithmetic based systems on those technologies which will provide the highest performance in terms of:

1. speed
2. cost
3. power dissipation
4. packaging
5. availability
6. system compatability

parameters. The last two parameters are unfortunately often neglected in exploratory research efforts. It would reflect poor engineering practice to develop a technology dependent theory which is incompatable with it's elcctronic environment. The technology which seems to yield the greatest promise is the VLSI.[i] High performance arithmetic units are already available in VLSI. For example, the TRW-VLSI carry-save 2's complement multiplier line breaks down as follows:

TABLE 1

UNIT	SIZE	PINS	SPEED(ns)	POWER (watts)
MPY8HJ-1	8 x 8	40	45	1.5
MPY-12HJ	12 x 12	64	80	2.7
MPY-16HJ	16 x 16	64	100	4.0
MPY-24HJ	24 x 24	64	200	5.0

By comparison, the 12 x 12 35 ns RNS multiplier is more than twice as fast as the VLSI unit but consumes more than 3.5 times the power. However, the above VLSI multiplier units are designed to work in 2's complement and

Footnote [i]: Small Scale Integration [SSI] = 50 gates, MSI = 50-100 gates
VLSI = 4000 gates/chip

therefore do not support residue arithmetic directly. Since these basic fixed point 2's complement VLSI multipliers offer outstanding performance for the price, it is desirable to integrate them into a residue number system (RNS) structure.

RESIDUE ARITHMETIC

Before a meaningful dislog on residue arithmetic units and systems can be established, the fundamental properties of this numbering system should be reviewed.

Residue number system (RNS) is mature mathematical study. A serious study of the RNS was offered by Gauss in the 19th century. In 1968 Szabo and Tanaka published the book "Residue Arithmetic and Its Applications to Computer Technology".[7] Due to the technological limitations of the period, the book did not receive wide-spread recognition and was soon out of print. However, due to the recent availability of high density high performance Read Only Memory (ROM) and Random Access Memory (RAM), the RNS is being re-investigated for the application in digital filter design, implementation of fast transforms, convolution, and optical computation.

Let $P=(p_1,p_2,\ldots,p_L)$ be a set of relatively prime integers, and let x be any integer in [0, M-1] (called dynamic range) where $M = \prod_{i=1}^{L} p_i$. Then by the Eucledian algorithm, there exists $k_i, x_i \varepsilon I$(integers), such that

$$x = k_i p_i + x_i \qquad i=1,2,\ldots,L \qquad (1)$$

The quantity x_i is called the ith residue of x, and is usually denoted as $|x|_{p_i}$ or $x \bmod p_i$.

It is easy to show that x and M+x have the same residue representation. Only if $x\varepsilon[0, M-1]$, can x be uniquely determined by the L-tuple $(x_1,x_2,\ldots,x_L)$. In this case denote $\tilde{x}=(x_1,x_2,\ldots,x_L)$.

Another signed encoding scheme can also be used. In this case, the dynamic range is $[-(M-1)/2, (M-1)/2]$ with a negative number $-|x|$ coded as $M-|x|$. There is a trivial the isomorphism which maps $[0, M-1]$ onto $[-(M-1)/2, (M-1)/2]$. This second coding scheme has the advantage that sign detection is not required during arithmetic operations and the sign of the result will take care of itself providing that no overflow (out of dynamic range) had occurred.

The following are some identities which will be used later. The proofs are straight forward and will not be presented.

$$|x\pm y|_p = \left||x|_p \pm |y|_p\right|_p \qquad 2.$$

$$|xy|_p = \left||x|_p|y|_p\right|_p \qquad 3.$$

$$|-x|_p = |p-x|_p \qquad 4.$$

Let Z_p be the set of integers x such that $0\leq x<p$ (ie: residue class). It is well known that Z_p is an abelian ring under addition and multiplication modulo p. For any integer $x\varepsilon Z_p$, the inverse of x is the integer $y\varepsilon Z_p$ such that $|xy|_p = 1$. It is also known that if x is relatively prime to p, then x has an unique inverse, denoted $x^{-1}[p]$. For example in $Z_6, 5^{-1}[6]=5$.

Arithmetic operations in RNS are defined in a straightforward manner. Let $x, y\varepsilon Z$, $x, y\varepsilon[0, M-1]$ and $x\equiv(x_1,x_2,\ldots,x_L)$, $y=(y_1,y_2,\ldots,y_L)$. Then $z=\tilde{x}oy=(z_1,z_2,\ldots,z_L)$ where $z_i=(x_i o y_i) \bmod p_i$, for $i=1,2,\ldots,L$, and "o" denote

the operation x, + or -.

It is clear that the sub-operation within each modulus is independent to each other. That is, no carry information is necessary between moduli. The arithmetic is also exact and therefore free of round-off error. The z is exact if $0 \leq z \leq M-1$, however if $xoy \geq M$ (overflow) then the answer will be incorrect. Hence it is critical to know beforehand that the result will not exceed the finite RNS dynamic range. Division in RNS is known to be difficult. Therefore RNS is considered to be best applied to system where division is not the dominant operations.

Another RNS induced scheme is called the mixed-radix numbering system (MRNS). Given the moduli set $P=(p_1,p_2,\ldots,p_L)$, any integer $x\varepsilon[0, M-1]$ can be expressed uniquely as

$$x=\tilde{x}_1+\tilde{x}_2P_1+\tilde{x}_3P_1P_2+\ldots+\tilde{X}_L(p_1p_2\cdots p_{L-1}) \qquad 5.$$

let $q_1=1$ and $q_i=\prod_{j=1}^{i-1} P_j$, eq. (5) can be written as

$$x=\tilde{x}_1q_1+\tilde{x}_2q_2+\tilde{x}_3q_3+\ldots.+\tilde{x}_Lq_L \qquad 6.$$

or equivalently, x can be represented uniquely by the L-tuple $x=<\tilde{x}_1,\tilde{x}_2,\ldots.,\tilde{x}_L>$. The $\tilde{x}_i$'s are called the mixed radix digits with $0 \leq \tilde{x}_i \leq p_i-1$. The mixed-radix number system is a weighted number system. Therefore carries between digits are necessary in arithmetic operations. A property of a weighted number system is that magnitude comparison is trivial.

It is often necessary to compute the M.R. digits $<\tilde{x}_1,\tilde{x}_2,\ldots.,\tilde{x}_L>$ from given set residue digits $(x_1,x_2,\ldots.,x_L)$. Here

$$x = \tilde{x}_1 + \tilde{x}_2 p_1 + \ldots + \tilde{x}_L \prod_{i=1}^{L-1} p_i \qquad 7.$$

Hence,

$$|x|_{p_1} = x_1 = |\tilde{x}_1 + \tilde{x}_2 p_1 + \ldots + \tilde{x}_L \prod_{i=1}^{L-1} p_i|_{p_i} = |\tilde{x}_1|_{p_1} = \tilde{x}_1 \qquad 8.$$

After subtracting $\tilde{x}_1$ from both sides of eq. (7), one obtains

$$x - \tilde{x}_1 = \tilde{x}_2 p_i + \ldots + \tilde{x}_L \prod_{i=1}^{L-1} p_i \qquad 9.$$

$$|x - \tilde{x}_1|_{p_2} = |\tilde{x}_2 p_1 + \ldots + \tilde{x}_L \prod_{i=1}^{L-1} p_i|_{p_2} = |\tilde{x}_2 p_1|_{p_2}$$

Upon multiply both sides by $p_1^{-1}[p_2]$ which exists by the relatively prime property of p_1 and p_2, one obtains

$$|(x - \tilde{x}_1) p_1^{-1}[p_2]|_{p_2} = |\tilde{x}_2 p_1 p_1^{-1}[p_2]|_{p_2} = |\tilde{x}_2|_{p_2} = \tilde{x}_2 \qquad 10.$$

This process can be carried out successively until all $\tilde{x}_i$'s are obtained. Actually, the iterate process can be realized in parallel form due to the independence of residues. An algorithm found in Szabo and Tanaka[7] can be used.

It was noted that division is difficult in RNS. However, in the case that the divisor is a fixed constant c (where c is relatively prime to $p_i, i=1,\ldots.L$), there is known to exist some simplification of the scaling task. The scaling operation is formly defined as follows: Given $P=(p_1, p_2, \ldots., p_L)$ and $x \equiv (x_1, x_2, \ldots., x_L)$, what is the residue representation

of $|\frac{X}{C}|$? (where $|\ \ |$ denotes the rounding to the closest integer operation.) From the Eucledian Algorithm, namely

$$X = |\frac{X}{C}|\, C + |X|_C \tag{10.}$$

it follows that

$$|\frac{X}{C}| = \frac{X-|X|_C}{C} \tag{11.}$$

which is of integer value. The residue representation of this integer is given in terms of a "scaling kernel" satisfying,

$$|\frac{X}{C}|_i = \|\frac{X}{C}\|_{p_i} = |(X-|X|_C)c^{-1}[p_i]|_{p_i} = |(x_i-|X|_C)c^{-1}[p_i]|_{p_i} \tag{12.}$$

Thus, if $|X|_C$ is known, then the residue representation of $|\frac{X}{C}|$ can be obtained using one subtraction and one multiplication. Since c are relatively prime, $c^{-1}[p_i]$ exists w.r.t. p_i. Usually $|X|_C$ will not be given and have to be found by a base extension algorithm.

The integer value of a residue representation can also be obtained through the use of Chinese Remainder Theorem.

Given $P = (p_1, p_2, \ldots, p_L)$; where p_i, p_j are relatively prime for $i \neq j$, the CRT states;

Theorem (Chinese Remainder Theorem)

$$|X|_M = |\sum_{i=1}^{L} m_i |x_i m_i^{-1}[p_i]|_{p_i}|_M \tag{13.}$$

where

$$m_i = \frac{M}{p_i} = \prod_{\substack{j=1 \\ j \neq i}}^{L} p_j \quad \text{and} \quad |m_i m_i^{-1}[p_i]|_{p_i} = 1 \qquad 13.$$

Proof: Since a residue number represents an integer uniquely in the dynamic range [0, M-1]. It is enough to show that the right-hand side of eq. (13) has residues $(x_1, x_2, \ldots, x_L)$. Since

$$\left| |x|_M \right|_{p_j} = \left| \left| \sum_{i=1}^{L} m_i |x_i m_i^{-1}[p_i]|_{p_i} \right|_M \right|_{p_j} = \left| \sum_{i=1}^{L} m_i |x_i m_i^{-1}[p_i]|_{p_i} \right|_{p_j}$$

$$= |m_j |x_j m_j^{-1}[p_j]|_{p_j}|_{p_j} = |x_j|_{p_j} = x_j \qquad \forall j = 1, \ldots, L$$

The claim follows.

*Notice that the left-hand side of eq. (13) is in the form of $|x|_M$. That is, the resulting integer will be unique if $0 \leq x < M$.

There is yet another method which may be used to decode a residue tuple. This method has been independently reported by Jenkin[8] and Julian.[9] Starting from the residue representation $(x_1, x_2, \ldots, x_L)$, $\langle \tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_L \rangle$ is obtained through a M.R. conversion. Then eq. (5) will be used to reconstruct x. This method is called M.R. reconstruction.

RNS CAPABILITIES

Interest in the RNS is due to its ability to perform high-speed arithmetic. Speed is achieved through the use of a high degree of parallelism and an absence of carry information requirements. Recall that the arithmetic composition of two integers, say $x \mapsto (x_1, \ldots, x_L)$ and $y \mapsto (y_1, \ldots, y_L)$, given by x y (where

$\circ$ denotes addition, subtraction, or multiplication) satisfies $x \circ y \rightarrow (x_1 \circ y_1, \ldots, x_L \circ y_L)$. It can be seen each residue digit, namely $x_i \circ y_i$ can be computed independent of all others (ie: no carry information requirements). In practice, the mapping of x_i and y_i into $x_i \circ y_i$ is accomplished using table lookups where the table residue on randomly accessed read-only memory. Typical high-speed memory modules, which are currently available, are:

TABLE 2

Device	Type	Technology	Configuration	Access-Speed
10149	ROM	ECL	256x4	20 ns
SN54S	ROM	TTL	1024x4	35 ns
2147H-1	RAM	HMOS	4096x1	30 ns
2125H-1	RAM	HMOS	1024x1	20 ns
I2167	RAM	HMOS	16384x1	45 ns
IMS1400	RAM	MOS	16384x4	30 ns

The product of two residues modulo p_i, $p_i \leq 2^n$, can be precomputed and stored in a 2^mxn-bit memory unit where m=2n. Using a large existing high-speed memory (4Kx1 at 30 ns), residues having up to six bit integer values can be used (ex: P = {64,63,...}). Thus, fixed-point multipliers having a dynamic range of [-M/2,M/2) can be architected which have execution rates in the low nanoseconds.

The disadvantages of the residue number systems are manifold. Since the RNS possess no most significant digit, decimal to residue conversion, division, magnitude comparison, and arithmetic shift operations are cumbersome and should be avoided. Register overflow, due to its finite dynamic range, impose a severe constraint on the RNS operations. Unlike weighted numbers (decimal, binary, etc.) where rounding or truncating least significant digits can control overflow, such is not the case in the RNS. Since there is an absence of least significant digits, the more general and inefficient

operation known as scaling must be used. Since scaling is a form of division, its use should be discouraged. To gain insight into this problem, consider the inner product of two 31-dimensional real vectors x and y whose entries are encoded as residue digits with respect to P = {32,31,29,27}. Without scaling, the worst-case value of x and y would be limited to $V \cdot .5$ where V = (M/2)/31 = 25056. Therefore, to insure that no worst case overflow can occur, a 7.3-bit (ie: $V \cdot .5 = 158 \cong 2^{7.3}$) dynamic range limitation must be imposed on x and y. With scaling, larger input ranges can be used at the expense of statistical accuracy in the output space (analogous to roundoff errors).

Due to the dynamic range limitation of RNS systems, one is generally forced to accept one of the following two overflow prevention strategies.

1. Increase the dynamic range to a sufficiently large value by adding more moduli to P, or
2. Make scaling a more efficient operation.

The first option represents a brute force attack to the problem. Such an approach will increase to cost and complexity metrics of a filter. In addition, the moduli set P must be tailored to unique filter. The other approach appears to be the most popular at this time. Szabo and Tanaka, and others, have concentrated on the scaling efficiency through the choice of the three-tuple moduli set $P = \{2^n-1, 2^n, 2^n+1\}$. This moduli set has the ability to efficiently scale a residue number by any one of the chosen moduli. However, there is an intrinsic limitation plaguing this method and it is its dynamic range. Using a large high-speed memory unit, say 4Kx1, the input addressing space is limited to 2^{12}. This means that a moduli p_i

is technically limited to $p_i \leq 2^6$ (ie: $x_i - y_i < 2^{12}$). Therefore, the dynamic range of any modular operation is given by $M=(2^n-1)(2^n)(2^n+1) \sim 2^{3n}=2^{18}$. In many applications, an 18-bit resolution is insufficient resolution.

LARGE MODULI AU

It is desirable to keep the previously discussed three moduli structure for purposes of potential scaling needs. However, in order to overcome the existing disadvantages of this system, that of dynamic range, a new architecture is called for. Since it is unrealistic to assume substantially larger density high-speed memories will continue to become available, it is incumbent that more memory efficient residue arithmetic unit be designed. An efficient algorithm, which is ideally suited for this application, is known as the quarter-square multiplier.[10-12]

$$\langle xy \rangle_p = \langle \phi(s^+) - \phi(s^-) \rangle_p \qquad 14.$$

where $\phi(s) = \langle s^2 \rangle_p$ with $s^+ = (x+y)/2$ and $s^- = (x-y)/2$.

The quarter-squared multiplier has been studied by J.M. Pollard (1976) in a Galois field. Questions of hardware implementation were not considered and, due to the Galois field limitation, only prime moduli could be considered. H. Nussbaumer (1976) studied the quarter-square multiplier over real fields for use in ROM intensive digital filters. Soderstrand and Fields (1977) made brief reference to this multiplier for residue arithmetic but offered no satisfactory hardware realization. Our research has produced a practical residue arithmetic quarter-squared modular multiplier in commercially available hardware.

A problem that would seem to plague the quarter-square multiplier is the need to realize the division by two the sums and differences. In general, the existence of an N^{-1}, such that $\langle N^{-1}N\rangle_p=1$, can only be guaranteed if N is relatively prime to p. Since one of the chosen moduli is $p=2^n$, multiplicative inverse of 2 cannot be guaranteed to exist. Therefore, the quarter cannot be directly interpreted as the equation $\langle\langle 1/4\rangle_{p_i}\langle(x+y)^2-(x-y)^2\rangle_{p_i}\rangle_{p_i}$. The potential problem of dividing the sum of differences, found in equation 4, by 2, will be explicitly and efficiently treated for the first time later in this paper. For a 2^m word memory unit, the direct product architecture (ie: xy) would limit the maximal moduli to be bounded by 2^n, n=m/2. In fact, this claim can be extended to the case where $p = 2^n+1$ through use of the following modification. Observe that if $x_i = 0$, then it automatically follows that $\langle x_i y_i\rangle_{p_i} = 0$. Therefore, if $x_i=0\rightarrow 0_\Delta 0\ldots 0$ (which is detectable condition in that the (n+1)st bit and remaining n-bit block is zero ($0\rightarrow 0_\Delta 00\ldots 0$)) the output register would be automatically cleared. Therefore, the lookup table need not be accessed for this case. Instead, the all zero n-bit portion of the table address, allocated to x_i, can be used to represent $x_i=2^n$ where $x_i = 2^n\rightarrow 1_\Delta 00\ldots 0$. Here, the table would be programmed to map y_i into $\langle 2^n y_i\rangle_{p_i}$ using only a 2^m word memory.

The memory requirements associated with the quarter-square multiplier are substantially less than those of direct mechanizations. First, it should be apparent that the integers s^+ and s^-, found in equation 14, are bounded from above by 2^{n+1}. Therefore, only a (n+1)-bit table addressing space is required to realize (s^+_-) versus the 2n-bit space needed for direct architectures. It would appear however, that there is an exception to this rule. Since one

of the moduli chosen is $p = 2^n+1$. Here the maximal value of s^+(or s^-) is 2^{n+1} which would technically require a (n+2)-bit address. However, by using the protocol found in Figure 2, which is an adaptation of the network found in Figure 1, the table size can be reduced to 2^{n+1} words for all moduli. Here, the overflow bit serves to differentiate $s^{\pm}=0$ from 2^{n+1}.

The quarter-squared architecture is abstracted in Figure 3. It uses a 2^{n+1} word high-speed memory for modular arithmetic lookup operations. Using, for example, the previously referenced 4K-30 ns device, moduli having an 11-bit dynamic range (vs. 6-bit in the direct form) can be mechanized. This would yield a three-moduli dynamic range on the order of $2^{3(11)} \approx 8 \cdot 6 \cdot 10^9$. That is, without an increase in memory size (and therefore access time), the dynamic range of the quarter-squared is $2^{33}/2^{18}=2^{15}$ times larger than that obtainable through direct means! This large increase in dynamic range makes the RNS a viable alternative to traditional filter design methods. Both improved precision and throughput (through the reduction or absence of traditional scaling operations) can be achieved.

Several versions of the multiplier algorithm can be considered. They are summarized in Figure 4. The first, called the sequential form, would have an estimated throughput rate of 240 ns based on a 60 ns lookahead adder and memory having an access time of 30 ns with a cycle time of 60 ns. The second architecture, called the parallel form, would run at a 180 ns rate. The parallel architecture is preferred because its higher speed, simpler control. A 60 ns pipelined execution rate can be purchased at a small hardware cost.

Upon closer investigation of the table lookup data base, a potential nuisance can be found. It can be examplified by observing that if $s^{\pm}$ is,

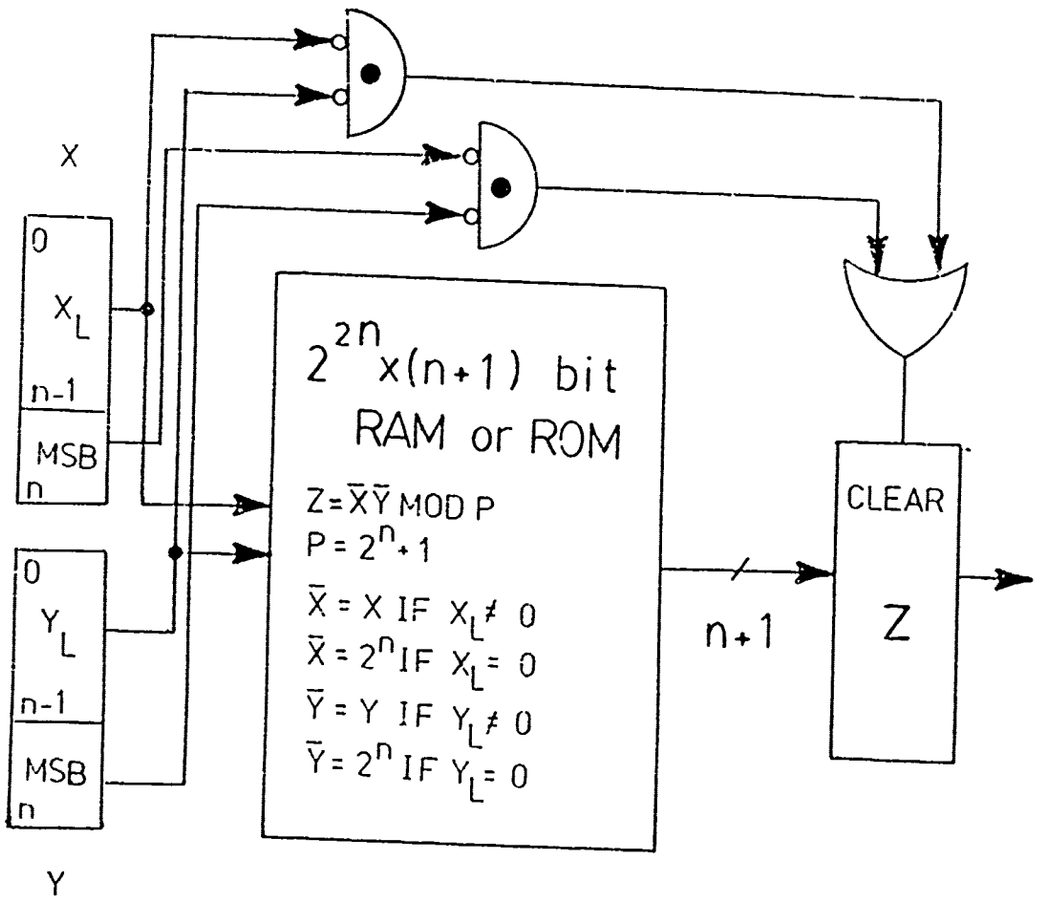


Figure 1. Modulo 2^n+1 ALU

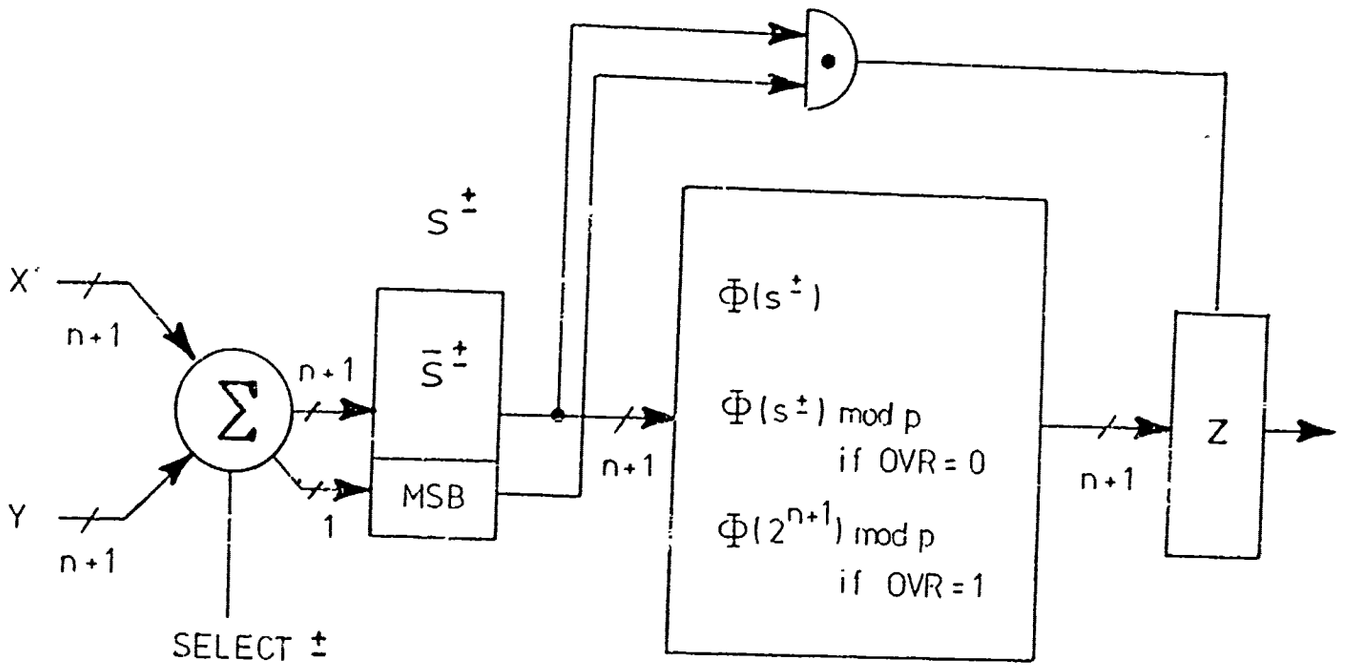


Figure 2. Memory Compression For $s^{\pm}$

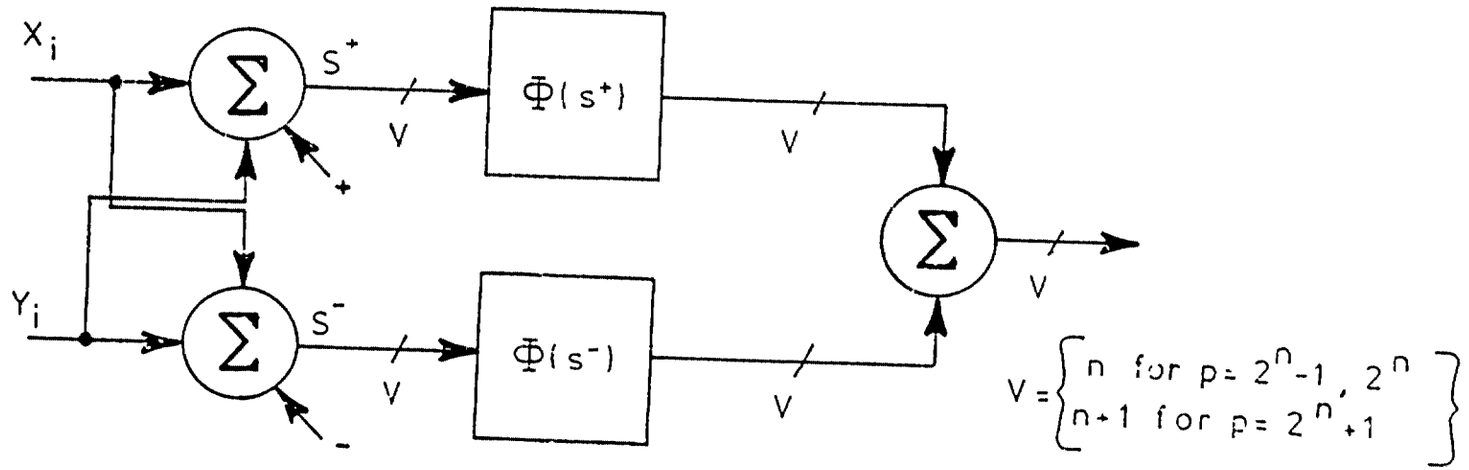


Figure 3. Modulo p_i Multiplier

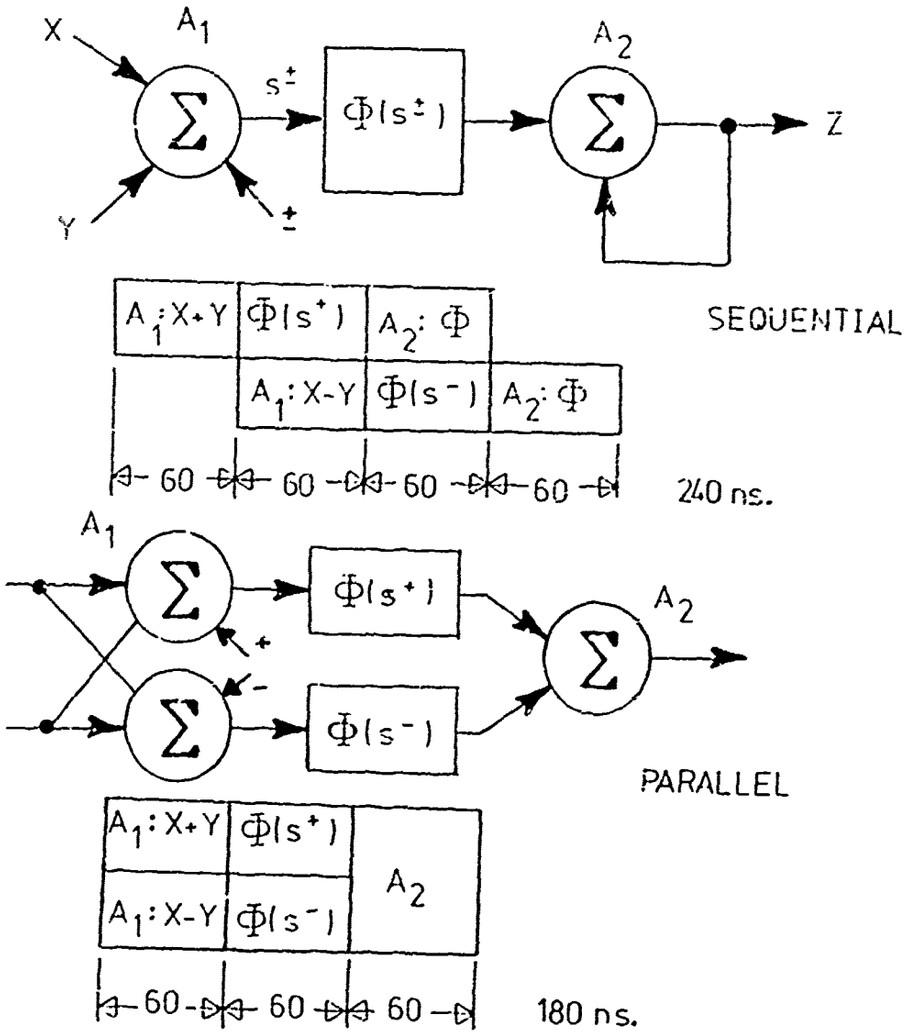


Figure 4. Architectures

odd and $p=2^n$, then $\phi(s^{\pm})=\langle s^2/4\rangle_{32}=x.25$. Therefore, it may be required that two additional fractional bits may need to be added to the table's output wordlength. However, this is not the case as suggested by the following theorem:

Theorem: Let $\| v\|$ denote the integer value of v. Then $z=\langle \| \phi(s^+) \| - \| \phi(s^-) \| \rangle_p$. That is, only the integer value of ϕ need be used and the fractional bits of $\phi(s^{\pm})$ can be ignored.

Proof: For x, y and k integers, one may define two rational numbers, namely $(x+y)/2 \stackrel{\Delta}{=} v+k/2$; $(x-y)/2 \stackrel{\Delta}{=} q+b/2$ where k=0 or 1. Then $z=\langle\langle (x+y)^2/4\rangle_p - \langle (x-y)^2/4\rangle_p\rangle_p = \langle\langle v+kv+b^2/4\rangle_p - \langle q+dv+k^2/4\rangle_p\rangle_p = \langle\langle v+kv\rangle_p + k^2/4 - q+k\rangle_p - b^2/4\rangle_p = \langle \phi(s^+) - \phi(s^-)\rangle_p$.

As a result, the parallel architecture is equivalent to that shown in Figure 5. Furthermore, by deriving the above theorem over a rational field, and showing that the results pertain to the integers, several classical problems are overcome:

1. The quarter-squared multiplier is not restricted to the Galois fields suggested by Pollard.
2. The question of the existence of the multiplicative inverse of 4 is now moot.

MODULO p ADDER

The quarter-square multiplier requires a modulo p adder be used to combine the two component parts of the solution (namely $\phi(s^+)$ and $\phi(s^-)$). Modulo p adders pose an interesting design problem. Unless a fast modulo p adder can be fabricated, the overhead associated with addition will offset any gain in throughput achieved through table lookups. For the moduli chosen, 2^n-1, 2^n, and 2^n+1, only the modulo 2^n adder can be realized directly. (n-bit adder with

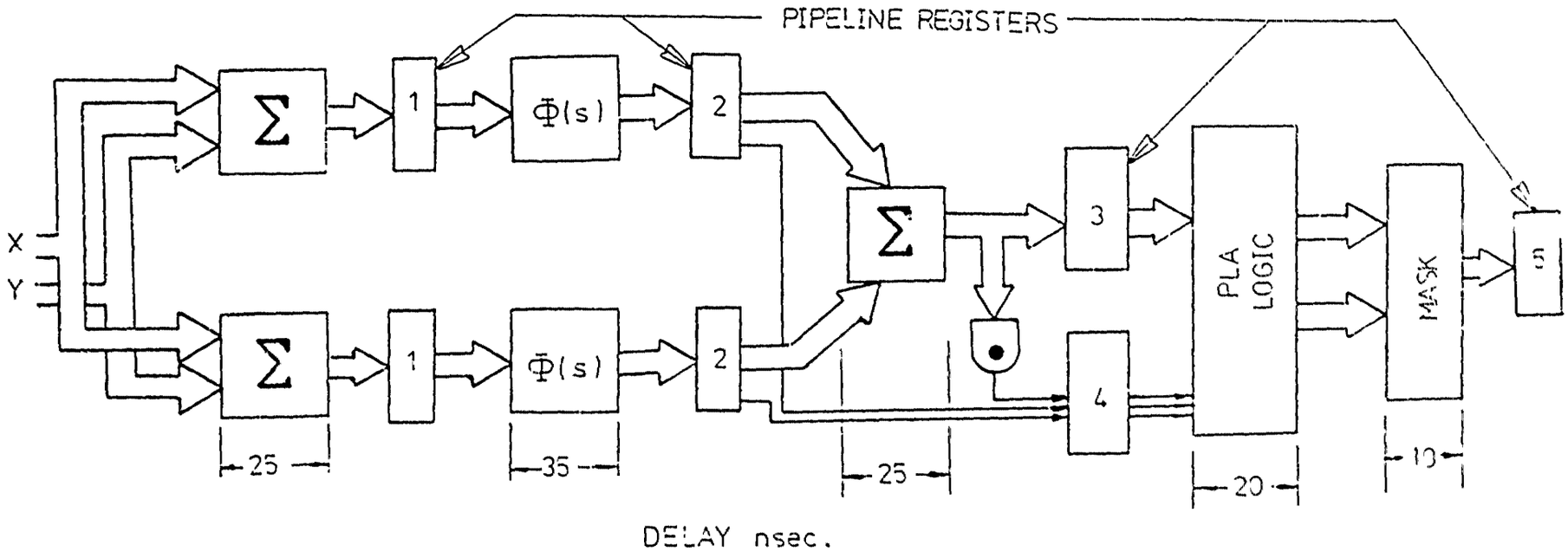


Figure 5. Large Moduli Multipliers

ignored overflow). It would however, be desirable to use a n-bit adder to realize the modulo 2^n-1 and 2^n+1 adder as well. For the purpose of clarity, let s be defined to be the sum of $\phi(s^+)$ and $\phi(s^-)$. The following observation then follows:

TABLE 3

Case	Dynamic Integer Range of S	Modulo $\langle s\rangle_{2^N}$	2^N Adder OVF-BIT	Modulo p_i	p_i Adder $\langle s\rangle_{p_i}$	Example: N=3 s	$\langle s\rangle_{p_i}$
1	$s=0$	0	0	2^N-1	0	0	0
2	$1\leq s\leq 2^N-2$	s	0	2^N-1	s	4	4
3	$s=2^{N-1}$	s	0	2^N-1	0	7	0
4	$s=2^N$	0	1	2^N-1	$s-2^N+1$	8	1
5	$2^N+1\leq s\leq 2^N-4$	$s-2^N$	1	2^N-1	$s-2^N+1$	10	3
6	$s=0$	0	0	2^N	0	0	0
7	$1\leq s\leq 2^N-1$	s	0	2^N	s	4	4
8	$s=2^N$	0	1	2^N	0	8	0
9	$2^N+1\leq s\leq 2^N-2$	$s-2^N$	1	2^N	$s-2^N$	10	2
10	$s=0$	0	0	2^N+1	0	0	0
11	$1\leq s\leq 2^N-1$	s	0	2^N+1	s	4	4
12	$s=2^N$	0	1	2^N+1	s	8	8
13	$2^N+1\leq s\leq 2^{N+1}-1$	$s-2^N$	1	2^N+1	$s-2^N-1$	10	1
14	$s=2^{N+1}$ (special case)	0	0	2^N+1	$s-2^N-1$	16	7

Using n-bit AND gates to sense the zero condition of $\langle s\rangle_{2^N}$, the overflow bit OVF the sign bits of $\phi(s^+)$ and $\phi(s^-)$, combinational logic can be defined which will $\langle s\rangle_{2^N}$ into $\langle s\rangle_{p_i}$. It can be noted from the data found in Table 1

that the mapping requirements are:

1. for $p = 2^N-1$, map s to s or $s-2^N+1=\langle\langle s\rangle_{2^N}+1\rangle_{2^N}$ 15.
2. for $p = 2^N$, map s to $s-2^N=\langle s\rangle_{2^N}$
3. for $p = 2^N+1$, map s to s or $s-2^N-1=\langle\langle s\rangle_{2^N}-1\rangle_{2^N}$

Mapping two is trivially satisfied with an n-bit adder. The other two mappings require that s remains unchanged or it is decremented or incremented by unity. There are several ways to approach this problem. Bioul, Davis, and Quisquater have presented an unorthodox architecture for a modulo (2^n-1) adder using two-input gates.[12] Modulo $(2^n\pm1)$ adders can also be realized through the use of end-around-carries. However, compared to modulo 2^n addition, this approach would almost double the addition delay. This extended delay problem can be overcome through added complexity (ie: time multiplexing two end-around-carry adders). Mapping one and three can be efficiently realized in the manner suggested by the example found in Appendix A. The functional operation of adding one (mapping 1) or subtracting one (mapping 3) from the output of an n-bit adder is performed by a PLA. The PLA will provide an overlay mask which accomplishes the required task. The derivation and utility of the mask can be understood in the context of the following example.

Example: Suppose s is an 11-bit word having a decimal value of $s_{10} = 92$ or s_2=00001011100. If $s_{10}-1 = 91$ or $(s_{10}-1)_2$→ 00001011011 is desired, one notes that only the 3-LSB's of s_2 need be altered. In general, for n=12, only the following 13 distinct binary masks are required to form $(s_{10}-1)_2$.

MSB Pattern LSB	Notation
X X X X X X X X X X X	X = leave corresponding bit
X X X X X X X X X X 0	location of s_2 unchanged 1(or 0)
X X X X X X X X X 0 1	= change corresponding bit
⋮ ⋮	location of s_2 to 1 (or 0)
X 0 1 1 1 1 1 1 1 1 1 1	
0 1 1 1 1 1 1 1 1 1 1 1	Table II. MASK

Suppose the moduli $p = 2^n+1$, n = 12, is to be implemented. By using two commercially available 16x9 PLA's in parallel, the 12-bit output of an n-bit adder (shown as $\langle s \rangle_{2^N}$ in Table I) and the four previously specified control bits, can be converted to 13-bit mask. The mask would transform the output of a high-speed n-bit adder to s or $s-2^n-1$, depending on the state of the 4 control bits. Based on a 25-ns 12-bit Schottky lookahead adder, a 20-ns 16x9 PLA, and 10-ns FET mask switches (in notation comments of Table II) a 65-ns modulo p adder, for $p=2^n-1$, 2^n, and 2^n+1 can be realized. The presence of a 65-ns modulo p adder will now allow a 140-ns large moduli residue multiplier based on 35-ns 4Kx1 HMOS memory units. (See Figure 5) For a moduli set $\{2^{12}-1, 2^{12}, 2^{12}+1\}$, a fixed point multiplier, having an output dynamic range of $2^{36}-2^{12}$, can thus be fabricated having a word rate of 7.143 M multiplications per second. This compares favorably with new 16x16 VLSI multipliers. Using a pipelined architecture, which requires the insertion of the storage registers found in Figure 5, a very impressive throughput figure of 28.5 M multiplications per second. It is important, and fortunate to realize that the Intel HMOS memory unit, used in this analysis, has a cycle time equal to the access

time. If, as is often found in practice, a memory unit has a cycle time approximately twice the access time, then pipeline delay would increase from 35-ns to 70-ns.

VLSI RNS MULTIPLIERS

As previously noted, 16-bit three-moduli 35 ns pipelined multiplier is more than three times as fast as the VLSI unit but consumes more than four times the power and is significantly more complex. However, the above VLSI multiplier units are designed to work in 2's complement are therefore do not support residue arithmetic directly. In this paper, the algebraic elegance and speed of the RNS is combined with the technological advantages of VLSI to achieve high-performance modular multiplier.

Since the RNS is an exact numbering system, the nesting of modular arithmetic operations can result in register overflow. Register overflow occurs when the result of an arithmetic operation exceeds the admissible dynamic range M. For a set of relatively prime moduli set $P=\{p_1,\ldots,p_L\}$, $M=\Pi p_i, i=1,2,\ldots,L$. Overflow prevention in the RNS is accomplished through the use of a relatively inefficient operation referred to as scaling. This can be mechanized using the mixed-radix conversion algorithm or the Chinese Remainder Theorem.[5] To insure that there will be some degree of efficiency in the scaling operation, the moduli set must be carefully chosen.[6] A particularly useful moduli set is $P=\{2^n-1, 2^n, 2^n+1\}$. Based on this choice of moduli, a VLSI based residue multiplier can be realized in commercially available hardware.

VLSI-RNS MULTIPLIER STRUCTURE

This structure will be presented as three special cases.

Moduli $p=2^n$

For the purpose of discussion, consider $p=2^n$ to be a moduli and x, $x\epsilon Z_p$, to be the composite number

$$\langle x\rangle_p = x = 2^m x_{HI} + x_{LO};\ x_m = x_{HI} \text{ or } x_{LO},\ 0 \leq x_m \leq 2^m - 1 \qquad 16.$$

where $m=n/2$. Here x_{HI} and x_{LO} are m-bit positive integers and Z_p is the residue class of integers modulo p. For a y having the same format, it follows that $z=\langle xy\rangle_p$ is given by

$$z = \langle xy\rangle_p = \langle 2^n a + b + 2^m(c+d)\rangle_p \qquad 17.$$

where:

$$a = x_{HI}y_{HI};\ 0 \leq a \leq (2^m-1)^2 < 2^n - 1 \qquad 18.$$

$$b = x_{LO}y_{LO};\ 0 \leq b \leq (2^m-1)^2 < 2^n - 1$$

$$c = x_{HI}y_{LO};\ 0 \leq c \leq (2^m-1)^2 < 2^n - 1$$

$$d = x_{LO}y_{HI};\ 0 \leq d \leq (2^m-1)^2 < 2^n - 1$$

$$v = c + d;\ 0 \leq V \leq 2(2^m-1)^2$$

Under the hypothesis that $p=2^n$, and noting $2^m = 2^n/2^m$, z computes to be

$$z = \langle\langle 2^n \overset{0}{\nearrow} a\rangle_{2^n} + \langle b\rangle_{2^n} + 2^n(c+d)/2^m\rangle_{2^n}\rangle_{2^n} \qquad 19.$$

The last term in equation 19 may seem to pose a potential hardware realization difficulty. However, this need not be the case in light of the following interpretation. Suppose that the (n+1)-bit binary representation of the

positive integer v=(c+d) has the form xx...x (x=0 or 1). Then $V/2^m$ can be formed by simply defining the binary point to precede the mth LSB. That is, $V/2^m$ = IV+.XV where IV is the integer part of $V/2^m$ and XV the fractional part. Thus $\langle 2^n V/2^m \rangle_{2^n} = \langle 2^n(IV+.XV) \rangle_{2^n} = \langle 2^n \overrightarrow{0} IV \rangle_{2^n} + \langle 2^n(XV) \rangle_{2^n}$. Computing $\langle 2^n(XV) \rangle_{2^n}$ could promise to be an inefficient operation if conventional digital methods are used. However, this need not be the case since XV is known to be a m bit word where m is n/2. For example, if a 24-bit moduli is desired (which represents a substantial improvement over the 5-bit moduli typically found in the literature), then m=12, and a 4K x 1 high speed (35ns) memory can be used to implement the mapping $\langle 2^n(XV) \rangle_{2^n}$ as a table lookup operation. The partial product terms could then be combined by a moduli 2^n adder to form $z=\langle xy \rangle_p$.

Moduli $p=2^n-1$

Equation 16 can be rewritten in terms of the following set of relationships

$$\text{i:}\quad 2^n=(2^n-1)+1$$

$$\text{ii.}\quad 2^m=2^n/2^m=(2^n-1)/2^m+1/2^m \qquad 20.$$

with

$$z=\langle((2^n-1)a+a)+b+((2^n-1)(V/2^m)+V/2^m)\rangle_p \qquad 21.$$

From the previous analysis, one notes that $\langle a \rangle_p=a$, $\langle b \rangle_p=b$ and $V/2^m=IV+.XV$, with

$$\langle(2^n-1)V/2^m\rangle_p=\langle(2^n-1)(IV+.XV)\rangle_p=\langle(2^n-1).XV\rangle_p. \qquad 22.$$

Using lookup operations and a 2^m word memory unit the modular mapping can again be implemented directly. The term $V/2^m$ is, as previously stated, is

simply reassignment of the binary point of V. Again the partial product terms would be recombined using a moduli p adder.

Moduli $p=2^n+1$

This case requires special attention since it is not completely analogous to the previous case considered. In particular, not all the residues in the residue class Z_p can be encoded into an n-bit word and represented as $x=2^m x_{HI}+x_{LO}$. In other words, the admissible residue $x=2^n$ does not conform to the accepted data format. However, $x=2^n$ is an easily detected case since it is represented by x→1000...0 (ie: test MSB for 1 and AND with n-LSB's of 0's). If x is detected to have a value of 2^n, then only the following events are admissible

TABLE 4

x	y	$z=\langle xy\rangle_p$; $p=2^n+1$	example, n=6
2^n	$<2^n$	$\langle(2^n+1)y-y\ 2^n\rangle_{2^n+1}=\langle -y\rangle_{2^n+1}$	$\langle 64(y=5)\rangle_{65}$ =65-6=60
2^n	2^n	$\langle 2^{2n}\rangle_{2^n+1}$	
		$=\langle(2^n+1)^2-2(2^n+1)+1\rangle_{2^n+1}=1$	$\langle 4096\rangle_{65}=1$

These two possible events can be separately programmed without reducing

throughput. That is, upon receipt of x (or y) = 2^n, the output will be immediately set to $<-y>_p$ (or 1).

An architecture capable of realizing the proposed large moduli multiplier in VLSI is suggested in Figure 5. This system is composed of four commercially available VLSI multipliers, one custom VLSI Quad moduli p adder, and memory units for table lookup use. More will be said on the structure of the modulo p adder in the next section. For values of n=24 or 16-bits, and based on commercial multiplier specifications, a three moduli multiplier system can be built having a dynamic range on the order of 72 to 48-bits. Furthermore, based on these parameters, a multiplier can be partitioned into four 100 ns operations. This translates into a real-time throughput of 2.5 M multipliers per second for a serial realization or 10 M multiplier per second when pipelined (a most impressive 72 to 48-bit multiplication rate). The multiplier, suggested in Figure 5 performs the following operations.

TABLE 5

Operation	$p=2^n$	$p=2^n-1$	$p=2^n+1$	Level	Remarks
$S1: \langle 2^n x_{HI} y_{HI} \rangle_p \rightarrow$	0	a	-a	1	VLSI multiplier
$S2: \langle x_{LO} y_{LO} \rangle_p \rightarrow$	b	b	b	1	VLSI multiplier
$T_1: x_{HI} y_{LO} \rightarrow$	c	c	c	1	VLSI multiplier
$T_2: x_{LO} y_{HI} \rightarrow$	d	d	d	1	VLSI multiplier
$U: \langle a+b \rangle_p \rightarrow$	b	$\langle a+b \rangle_p$	b-a	2	mod p adder
$V: c+d \rightarrow$	IV	IV	IV	2	adder-shift register
$V':$	0	$+IV/2^m$	$-IV/2^m$	2	adder-shift register
$XV:$	.XV	.XV	.XV	2	adder-shift register
$W1: \langle U+V \rangle_p$	w_1	w_1	w_1	3	mod p adder
$W2: \langle p.XV \rangle_p$	w_2	w_2	w_2	3	table lookup
$Z: \langle w_1+w_2 \rangle_p$	2	2	2	4	mod p adder

Example: n=6, m=n/2=3, $p=2^6=64$ $\langle xy \rangle_{64} = \langle 558 \rangle_{64} = 46$

Let x = 18 = 2(8) +2 2:2 (HI:LO) $\langle xy \rangle_{63} = \langle 558 \rangle_{63} = 54$

y = 31 = 3(8)+7 3:7 (HI:LO) $\langle xy \rangle_{65} = \langle 558 \rangle_{65} = 38$

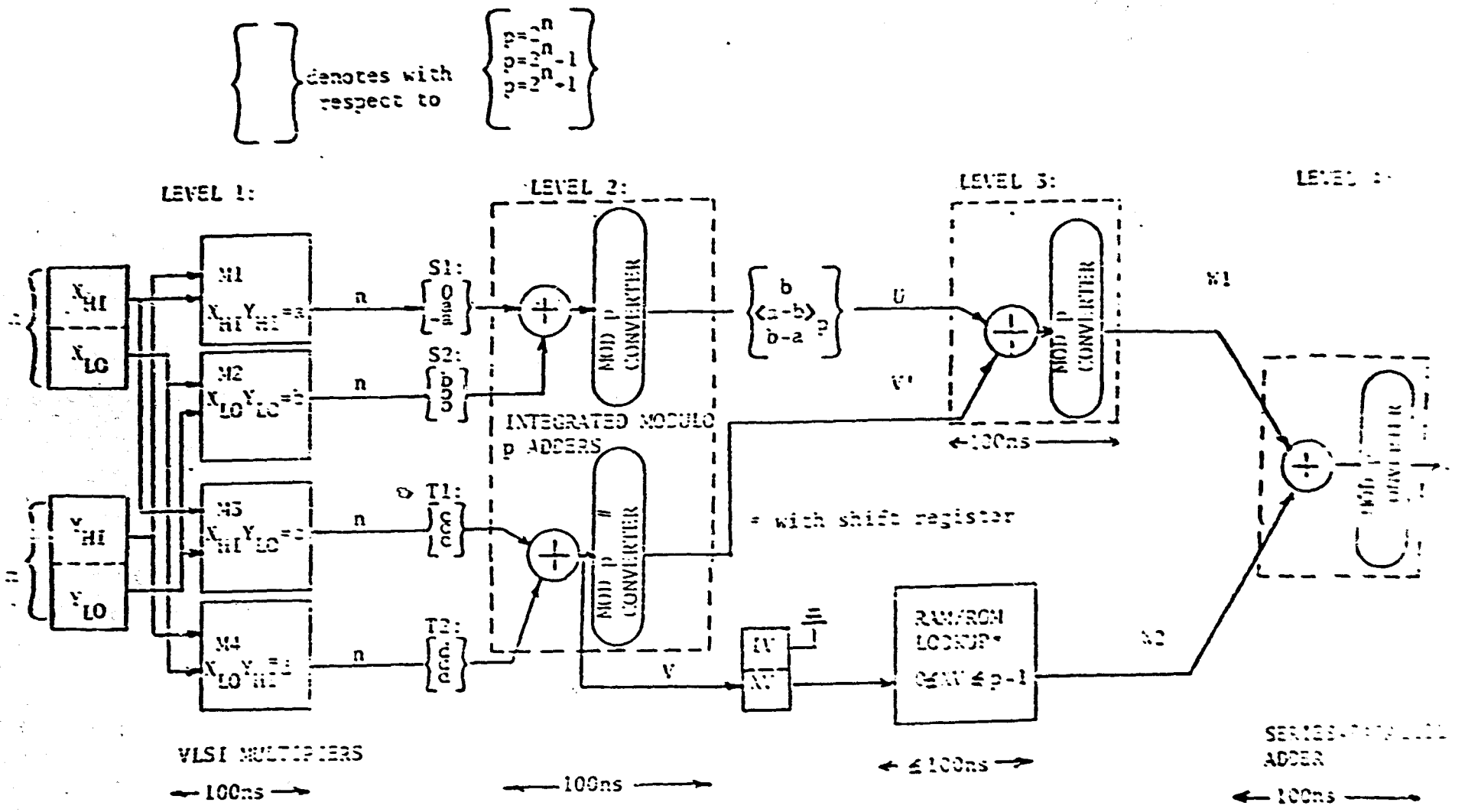
a=6, b=14, c=14, d=6

TABLE 6

Operation	p=64	p=63	p=65
S1:a	$\langle 64(6)\rangle_p=0$	$\langle 63(6)+6\rangle_p=6$	$\langle 65(6)-6\rangle_p=-6$
S2:b	$\langle 14\rangle_p=14$	$\langle 14\rangle_p=14$	$\langle 14\rangle_p=14$
T1:c	14	14	14
T2:d	6	6	6
U:u	14	20	8
V:v	20→0010100	20→0010100	20→0010100
V':	SET=0	20/8→+0010.100	-20/8→-0010.100
.XV	.100	.100	.100
W1:w_1	14→0001110	22.5→0010110.100	5.5→000011.100
W2:w_2(lookup)	32→0100000	31.5→0011111.100	32.5→0100000.100
Z:z	14+32=46	22.5+31.5=54	5.5+32.5=38

RNS TO DECIMAL CONVERSION

One of the major disadvantages of the residue numbering system (RNS) is its inability to efficiently perform magnitude comparison. Magnitude comparisons are critical to general purpose RNS operations since they are generic to the management of dynamic range (register) overflow and conditional branching. Unlike weighted numbering systems, where overflow can be efficiently handled by comparing data fields starting at the most significant digit, RNS



10 M multiplications/second

Dynamic Range { for n=16, DR=48-bits
for n=24, DR=72-bits }

VLSI/RNS MULTIPLIER

FIGURE 5

procedures are complex and time consuming.[4] Various versions of these RNS-to-decimal routines have been published which make use of modular table look-up operations and distributed (bit-slice) arithmetic.[13,14] However, the methods reported in the literature require a significant hardware investment and consume a disaproportionate amount of run-time compared to other RNS computational operations (viz: addition, subtraction, and multiplication). In the AFOSR program, a three new RNS-to-decimal has been developed which is significantly more efficient than existing techniques.

With respect to the moduli set $P = \{p_1,p_2,p_3\}$ there exists, for $0 \leq x < M$, three unique mixed radix conversion (MRC) digits $x \overset{MRC}{\longleftrightarrow} (\bar{x}_1,\bar{x}_2,\bar{x}_3)$ such that

$$x = \bar{x}_1 + \bar{x}_2 p_2 + \bar{x}_3 p_2 p_3 \qquad 23.$$

where $x \overset{RNS}{\longleftrightarrow} (x_1,x_2,x_3)$ with

$$\bar{x}_1 = x_2$$

$$\bar{x}_2 = \langle p_2^{-1}[p_3]*\langle x_3-x_1\rangle_{p_3}\rangle_{p_3} \qquad 24.$$

$$\bar{x}_3 = \langle p_3^{-1}[p_1]*p_2^{-1}[p_1]*[\langle x_1-x_2\rangle_{p_1}-\langle p_2^{-1}[p_3]*\langle x_3-x_2\rangle_{p_3}\rangle_{p_3}p_2]\rangle_{p_1}$$

More specifically, for the choice of moduli $P=\{2^n-1,2^n,2^n+1\}$, it follows that

$$p_2^{-1}[p_1]=1;\ p_2^{-1}[p_3]=2^n;\ p_3^{-1}[p_1]=2^{n-1} \qquad 25.$$

Upon substituting these multiplicative inverses into equation 2, one obtains

$$\bar{x}_1 = x_2$$

$$\bar{x}_2 = \langle 2^n*\langle x_3-x_2\rangle_{2^n+1}\rangle_{2^n+1} = \langle -(x_3-x_2)\rangle_{2^n+1} \qquad 26.$$

$$\bar{x}_3 = \langle 2^{n-1} * [\langle x_1 - x_2 \rangle_{2^n-1} - \langle x_3 - x_2 \rangle_{2^n+1}] \rangle_{2^n-1}$$

Functionally, it can be seer. that

$$\bar{x}_1 = f_1(x_2);\ \bar{x}_2 = f_2(x_1, x_3);\ \bar{x}_3(x_1, x_2, x_3) \qquad 27.$$

The MRC algorithm can of course be realized by using sequential methods. Here, nexted modulo p_i adders, and $p_i^{-1}[j]$ multipliers would be used to compute $\bar{x}_1, \bar{x}_2, \bar{x}_3$. The three-tuple of mixed radix digits would be used to compute $\bar{x}$ (equation 23) using these multiplications and additions. The disadvantage of the direct approach is execution speed due to the sequential complexity of the algorithm. Throughput improvements and a reduction in complexity can be achieved by using memory based table lookup operations to replace some arithmetic. If high speed is to be achieved, high-speed memory units must be employed. Such memories have a fairly restrictive input addressing space (5 to 12-bits). If mapping f_3 is to be realized, by presenting all three residues to a 4K-35ns RAM or ROM, then n=4, and $M<2^{12}$.

Consider again the three moduli case where $P=\{2^n+1, 2^n, s^2-1\}$ which specifies an RNS dynamic range $M=p_1p_2p_3$. Based on a 4K-word high-speed memory model, the previous medium moduli RNS-to-decimal converter was practically limited to a size of six-bits per moduli (ie: $M\backsim 2^{18}$). The method presented in this research targets a 12-bit moduli for practical use (ie: $M\backsim 2^{36}$). The developed large moduli scheme can be easily motivated by the data found in Figures 6a and 6b plus Table 7. The data found in these figures and tables are based on the moduli set P={5,4,3} and M=60. The first three entries found in Table 7, namely x_3, x_2 and x_1, are the residues digits of x for x monotonically increasing over [0,59]. The fourth and fifth entries namely J1 and J3, are hybrid parameters. Since $|p_2-p_1|=|p_3-p_2|=1$, the values of J1 and J3 will increase by unity (in a moduli p_i sense) for monotonically

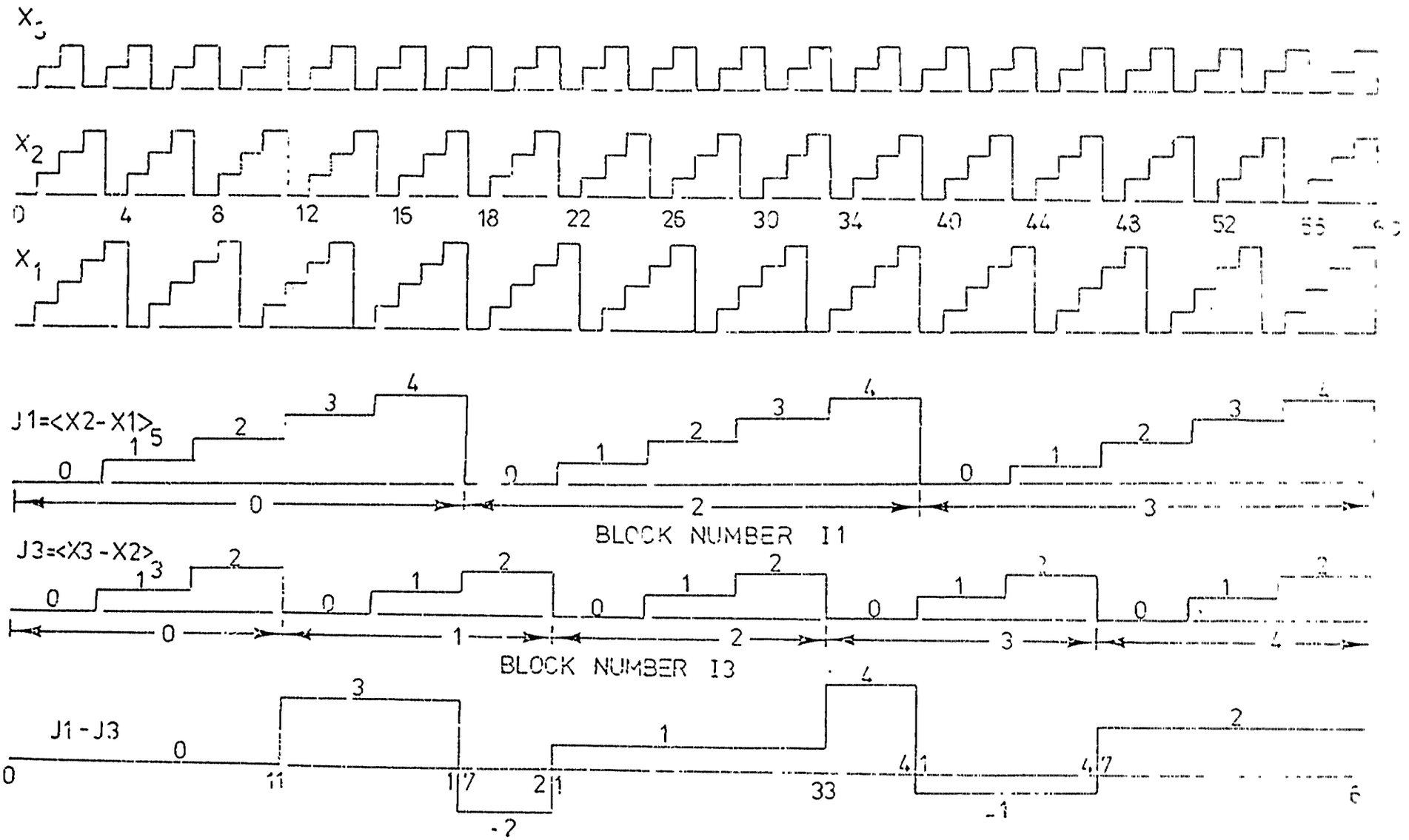
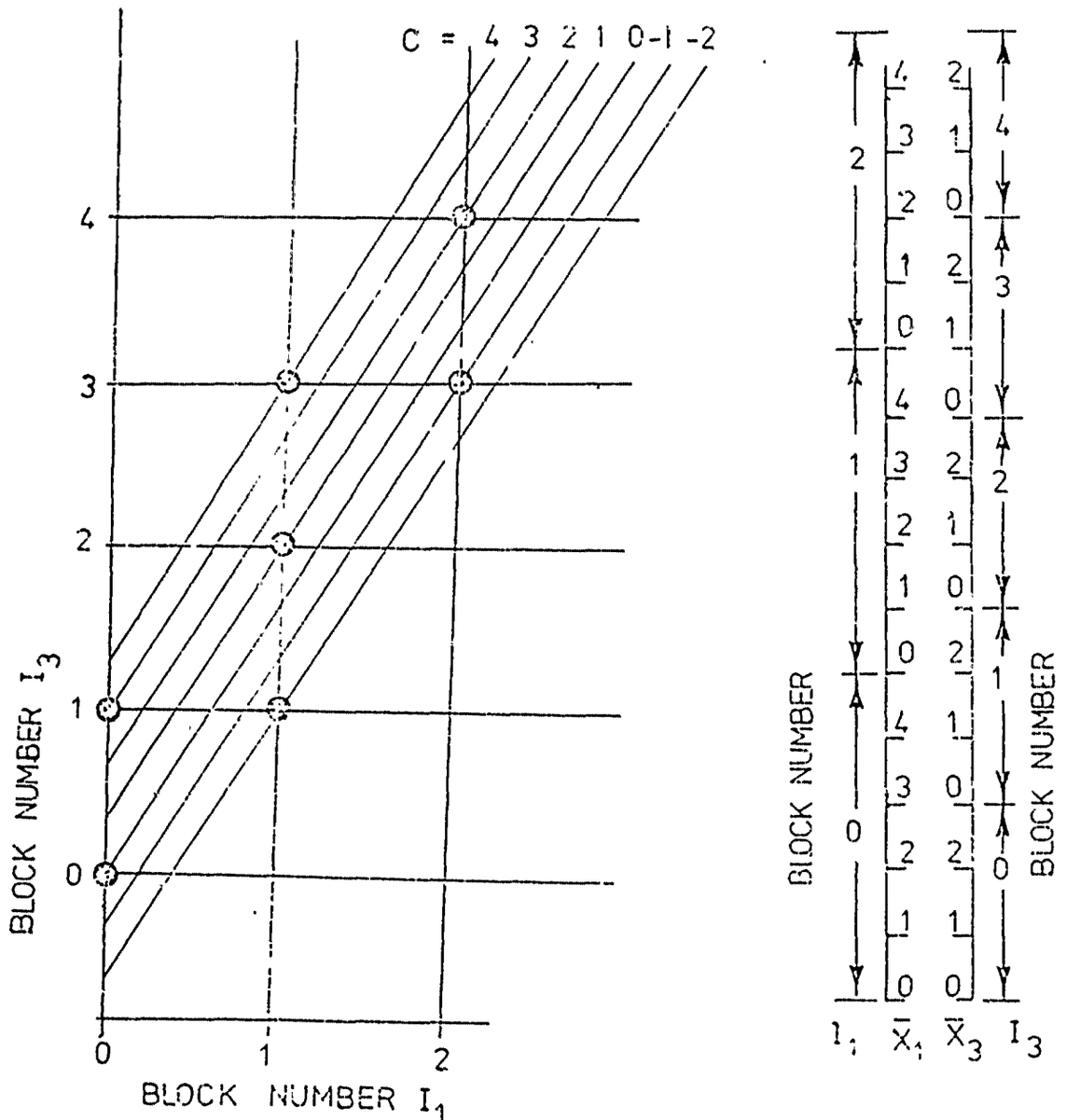
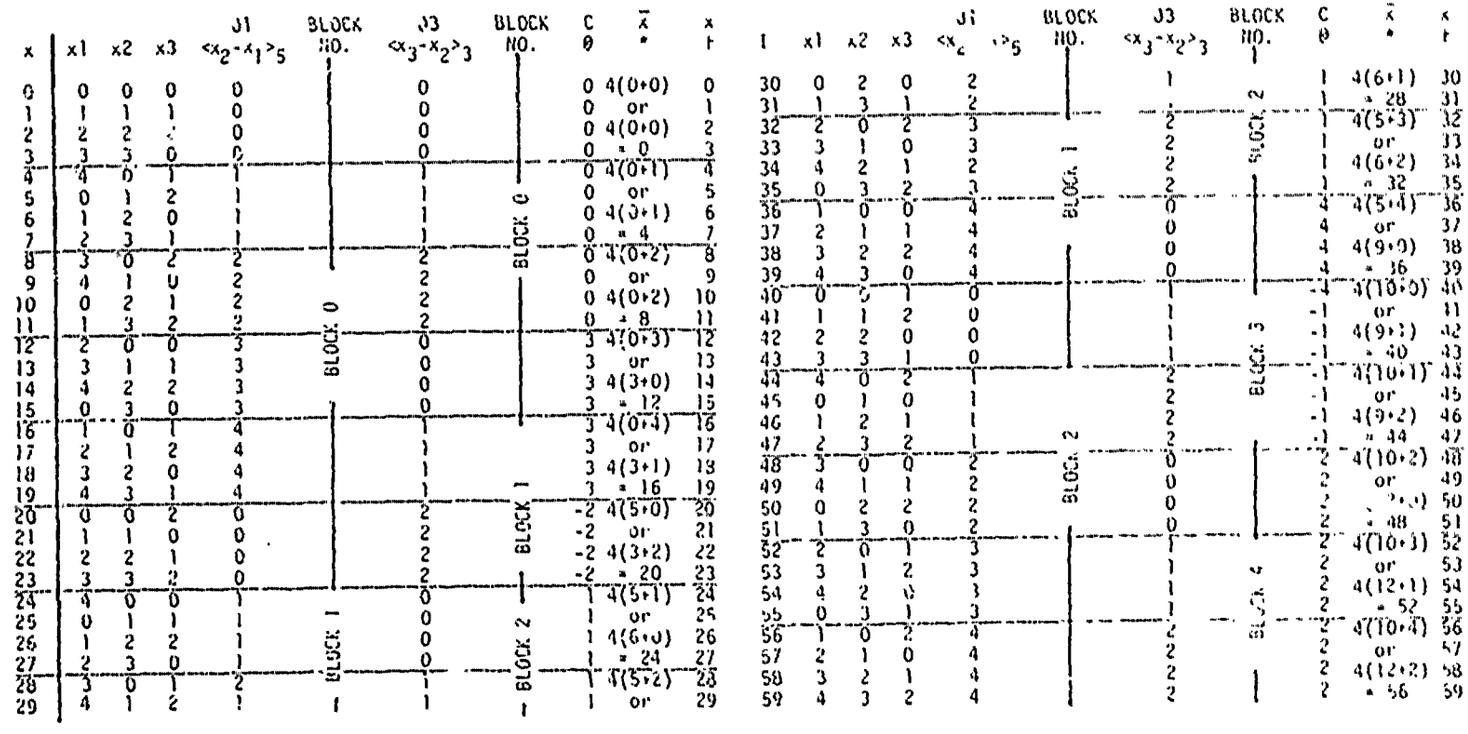

X3
X2
0 4 8 12 15 18 22 26 30 34 40 44 48 52 55
X1
J1=<X2-X1>5
BLOCK NUMBER I1
J3=<X3-X2>
BLOCK NUMBER I3
J1-J3
DIOPHANTINE EQUATION PARAMETERS

FIGURE 6a



ADMISSABLE SOLUTIONS

FIGURE 6b



EXAMPLE PROBLEMS

Table 7

increasing values of x. The important observation is that J1 and J2 naturally decompose into a system of cyclic patterns which shall be denoted S_1^2 and S_3^2 over a subcover of M, say S_2. More specifically,

S_1^2 = three sets of five subsets of four elements each and $O(S^1{}_1{}^2)=60$

S_3^2 = five sets of three subsets of four elements each and $O(S_3^2)=60$

$S_2 = \{kp_2 \mid\mid 0 \leq k < p_1 p_3\}$

In general, for $P=\{2^n+1, 2^n, 2^n-1\}=\{p_1, p_2, p_3\}$:

S_1^2 = p_3 sets of p_1 subsets of p_2 elements each and $O(S_1^2)=M=\pi p_i$

S_3^2 = p_1 sets of p_3 subsets of p_2 elements each and $O(S_3^2)=M=\pi p_i$

Using more traditional algebraic terminology S_1^2, S_3^2 and S_2 are ideals in the ring of integers modulo M (ie: Z_M). It is well known that in general the mapping

$$x \overset{RNS}{\longrightarrow} (x_1, x_2, \ldots, x_L),\ x_i = \langle x \rangle_{I_i} \qquad 28.$$

is an onto homomorphism with kernel $_j \cdot I_j$. For $I_i=\{kp_i\}$ (as is this case here), $_jI_j=0$ and Maher has shown the mapping to be isomorphic.[16] It should also be apparent that due to the cyclic nature of J1 and J2, that any $\overline{x}$ belonging to the subcover S_2 has the block representation

$$\overline{x} = \{(p_1 I1 + J1) * p_2;\ 0 \leq I1 < p_3;\ 0 \leq J1 < p_1\} \epsilon S_1^2 \qquad 29.$$

or

$$\overline{x} = \{(p_3 I3 + J3) * p_2;\ 0 \leq I3 < p_1;\ 0 \leq J3 < p_3\} \epsilon S_3^2 \qquad 30.$$

where I1,I3,J1, and J3 are integers. Equating equations 12 and 13, one obtains

$$p_3I3-p_1I1 = (J1-J3) \qquad 31.$$

which is of the form ax+by=c. Equation 14 possesses a very important property which will now be derived.

Lemma 1:[17] If a, b, c are integers and at least one a, b is nonzero, set d=gcd(a,b), then a solution to the Diophantine equation

$$ax + by = c \qquad 32.$$

exists for integer values of x and y if and only if d/c.

Lemma 2: If b is relatively prime to a, then the congruence by = c mod a has an integral solution x. Any solutions x_1 and x_2 are congruent modulo a.

From these two lemmas, the following theorem can be stated:

Theorem: Given the Diophantine equation 14

$$p_1I1-p_3I3 = (J1-J3) = c \quad \text{(see Figure 3b)} \qquad 33.$$

the solution two-tuple (I1,I3) is unique.

The proof is straightforwarded and is based on the fact that p_1 and p_3 are relatively prime, $I1\varepsilon[0,p_3-1]$, and $I3\varepsilon[0,p_1-1]$. Therefore, by specifying c, the block indices (I1,I3) can be uniquely determined. Observe that $\bar{x}\varepsilon S_2$ can be derived from knowledge of the two-tuple (I1,I3). However, (I1,I3) is uniquely determined by c=J1-J3. Therefore, upon presenting a (n+1)-bit word c to a (n+1)-bit high-speed RAM or ROM, the precomputed value of $S1=P_2P_1I1$ or $S3=p_2p_3I3$ can be outputted. The corresponding value of $\bar{x}\varepsilon S_2$ can be realized by adding to s, the integer p_2J1 to S1 or p_2J3 to S3. Lastly, if

$x\varepsilon[0,M)$, one only needs to add x_2 to $\bar{x}$. The decimal value x can therefore be computed in the composite form

$$x = x_2+p_2J1+I1p_1p_2 \qquad 34.$$

where, due to uniqueness, the mixed radix digits are $(x_2,J1,I1)$.

In general, for $P=\{2^n+1,2^n,2^n-1\}$, the routine would proceed as follows:

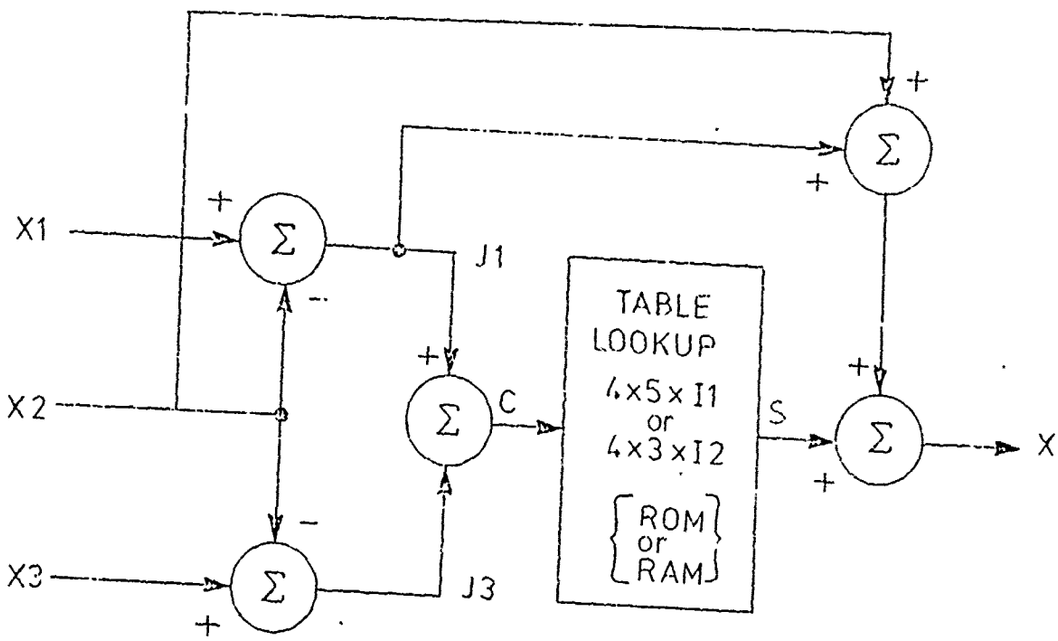
1. Accept $x \overset{RNS}{\quad} (x_1,x_2,x_3)$
2. Form $J1=\langle x_2-x_1\rangle_{p_1}$ and $J2=\langle x_3-x_2\rangle_{p_2}$
3. Form J1-J3 = c
4. Map $\phi(c)=p_1p_2I1=S1$ or $p_3p_2I3=S3$
5. FORM $\bar{x}=p_2J1+S1$ or $\bar{x}=p_2J3+S3$; $\bar{x}\varepsilon S_2$
6. FORM $x=\bar{x}+x_2$;

These steps are numerically examplified in Table 7 and diagrammed in Figure 7 for the {5,4,3} system.

Compared to conventional RNS-to-decimal conversion algorithms, the derived algorithm possesses the following attributes:

1. no modulo M addition required as in the case of CRT or MRC methods
2. practical realization of very large moduli RNS systems
3. simple architecture and reduced complexity.

Additional refinements in the proposed method can also be obtained. First, observe that c, the hybrid parameter which defines the argument of the mapping ϕ (item 4) is a signed integer such that $c\varepsilon[-(2^n-2),2^n]$. Technically, to do the mapping $\phi(c)$, a (n+1)-bit high-speed RAM or ROM would be needed. This



GENERAL ARCHITECTURE

FIGURE 7

suggests that the largest admissible moduli is 11-bits (using a 4K-memory model). Furthermore, since $c^{*}_{max}=2^{n}$, it would appear as though the output register for the signed-adder found at sten 3 would have (n+2)-bits if a standard binary weighted code is to be used (eg: 2's complement). This problem can be overcome through the following modifications.

1. Using an (n+1)-bit (at least) sign-magnitude adder, the sum c=J1-J3 can be represented as a (n+1)-bit word having the format MSB:xx$\cdots$x.LSB. The sum c can be partitioned into two sets V and Z given by:

   $x\varepsilon V$ if MSB of x = "0"

   $x\varepsilon Z$ if MSB of x = "1"

   More specifically, V is a set of 2^{n}-1 elements determined by $V = \{y \mid y = x \text{ for } x\varepsilon[0,2^{n}-1]\}$. Also, z is a set of 2^{n}-1 elements determined by $Z=\{z|z=x \text{ if } x\varepsilon[-(2^{n}-2),-1], z=0 \text{ if } x=2^{n}\}$. It can be seen that the sets Y and Z are defined by the magnitude digits of the signed magnitude value of c with the membership to Y and Z determined by the MSB (sign-bit location) of c. The importance of this partition is that two 2^{n} word tables can be used to map c into $\phi(c)$. The device select line would be tied to the MSB of c as suggested in Figure 8.

2. Another efficiency can be realized by using data packing. More specifically, the term p_2J1+x_2, for the considered choice of moduli, can be rewritten to read $2^{n}J1+x_2$. Since $0\leq x_2<2^{n}$, and $0\leq J1\leq 2^{n}$, the the term $2^{n}J1+x_2$ can be directly, and uniquely encoded into a (2n+1)-bit register. This is suggested in Figure 8.

3. The proposed architecture, as in the direct realization of the mixed

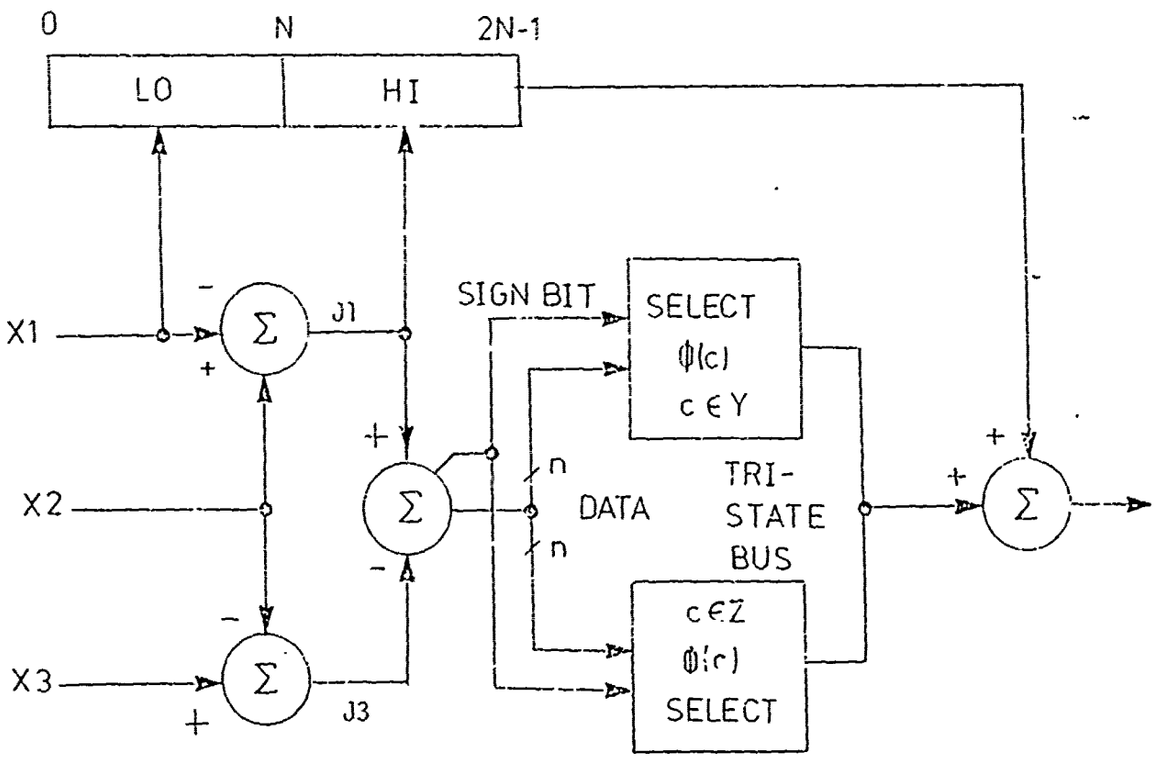
mixed radix conversion, requires moduli p_i for $p_i=2^n-1$ or 2^n+1. Several such modulo adders have been reported in the open literature. A very efficient 40nsec modulo 2^n+1 and 2^n-1 adder, for $n\leq 12$, has been reported in reference 18.

OVERFLOW TOLERANT RNS MULTIPLIERS

In order to extend the dynamic range of the autoscale multiplier to a more useful size (say 12 to 16 bits), based on a 4K memory model, data compression will be required. A suitable compression algorithm, based on the quarter-square algorithm has been reported in an earlier section. Furthermore, the theoretical foundation of a compression scheme has been motivated in the previous section. Here, data compression will be studied in the context of the popular three-moduli system $P=\{2^n+1,2^n,2^n-1\}$ such that $M=p_1p_2p_3=2^{3n}-2^n$. Any integer over $[0,M)$ has the unique RNS representation $x \overset{RNS}{\sim} (x_1,x_2,x_3)$. Consider now a subcover of the range $[0,M)$ generated by all numbers $\bar{x}$ having an RNS representation $\bar{x} \overset{RNS}{\sim} (\bar{x}_1,0,\bar{x}_3)$. Obviously $\bar{x}$ is defined over a subcover of $[0,M)$, say S_2 where $S_2 = \{kp_2 | 0\leq k<p_1p_3\}$. The utility of this operation is that of data compression. More specifically, only 2n-bits of data are needed to uniquely quantify $\bar{x}\underline{\sim}(\text{ie: } \bar{x}\underline{\sim}(\bar{x}_1,\bar{x}_3))$ versus 3n-bits for x (ie: $x\underline{\sim}(x_1,x_2,x_3)$). The digits of $\bar{x}$ can conveniently be defined to be:

$$\begin{aligned} \bar{x}_1 &= x_2-x_2 \\ \bar{x}_2 &= x_2-x_2=0 \\ \bar{x}_3 &= x_2-x_3 \end{aligned} \qquad 35.$$

As a point of interest, this is also an operation found in the residue to



REFINED ARCHITECTURE

Figure 8

mixed radix conversion algorithm used to determine the mixed radix coefficients of the weighted representation

$$x = \tilde{x}_1 + \tilde{x}_2 p_2 + \tilde{x}_3 p_2 p_3 \qquad 36.$$

ERROR ANALYSIS

The mean and error variance for the extended range autoscale multiplier is a function of the chosen moduli set. Even the simplest analysis becomes burdened with nested sums and binomial coefficients. Instead, the error statistics of the multiplier was studied using numerical simulation. A general purpose FORTRAN program, written on a PDP 11/60 under RSX-11, is reported in Figure 9. In Figure 10, the product of x=16 and $y\varepsilon[0,29]$, for P={3,4,5} is reported. The parameter Z_1 is the autoscaled product over S_2, Z_2 is the theoretical autoscaled product, with the last column exhibiting the error. The test software operations in either a deterministic or statistical mode. In either mode, the user specifies the choice of moduli (ie: $P=\{p_1,p_2,p_3\}$) and the number of fractional bits used to define the table lookup data. That is, the output wordlength is given by $[\log_2 M]$+ number of fractional bits. In the deterministic mode, all possible values of x and y over [0,N) are tested. However, if N is large, long execution delays can result. To overcome this problem, a statistical approach may be used where the integer value of x and y is randomly chosen from a uniformly distributed process over [0,N). The test is repeated M times and the statistics analyzed. The software presents to the user both error mean and error standard deviation. For example, for P={7,8,9} and zero fractional bits of accuracy, the deterministics error mean and standard deviation was determined to be e=-.00011476 and

ANALYSIS SOURCE CODE

```
C       [illegible]
        [illegible]
 10     FORMAT(' ANALYSIS TYPE : DETERMINISTIC=0, STATISTICAL=1')
        READ(2,*) ITYPE
        IF(ITYPE.EQ.1) GO TO 33
        WRITE(3,11)
 11     FORMAT(' DETERMINISTIC ANALYSIS')
        GO TO 77
 33     WRITE(3,44)
 44     FORMAT(' STATISTICAL ANALYSIS')
 77     CONTINUE
        WRITE(1,99)
 99     FORMAT(' LISTING DEVICE UNIT')
        READ(2,*) IL
        WRITE(1,100)
100     FORMAT(' INPUT P1, P2, P3')
        READ(2,*)P1,P2,P3
        WRITE(IL,120) P1,P2,P3
120     FORMAT(' MODULI CHOICE P1,P2,P3=',3F6.3)
111     CONTINUE
        WRITE(1,160)
160     FORMAT(' INPUT NUMBER OF FRACTIONAL BITS')
        READ(2,*) MM
        MM=INT(P2/2.)
        SCALE=2**MM
        SCALE=1/SCALE
        XM=P1*P2*P3
        S0=0.
        C0=0.
        S1=0.
        M0=INT(XM/2.)
        XXM=SQRT(XM)
        M1=INT(XXM)
        IF(ITYPE.EQ.1) M0=M1
        ISEED=35547
        DO 200 I=0,M0-1
        DO 200 J=0,M0-1
        IF(ITYPE.EQ.0) GO TO 280
        CALL RANDU(ISEED,IY,Z)
        II=INT(XM*Z/2)
        ISEED=IY
        CALL RANDU(ISEED,IY,Z)
        ISEED=IY
        JJ=INT(XM*Z/2)
        GO TO 300
280     II=I
        JJ=J
300     XI=II
        II=P2*INT(XI/P2)+RND
        XJ=JJ
        JJ=P2*INT(XJ/P2)+RND
        T0=2*II*JJ/XM
        YM=SQRT(2*XM)
        T11=((II+JJ)/YM)**2
        T12=((II-JJ)/YM)**2
        T1=(T11-T12)/SCALE
        XTEST=XM/SCALE
        IF(T1.LT.XTEST) GO TO 190
        WRITE(1,220) T11,T12,T1,II,JJ
220     FORMAT(' ERROR',3F16.4,2I9)
190     IT1=INT(T1)
        T1=SCALE*IT1
        C0=C0+1
        E=T1-T1
        S1=S1+E
        S0=S0+E*E
200     CONTINUE
        WRITE(1,360) C0
360     FORMAT(' TEST BASED ON',F10.1,' EXPERIMENTS')
        S1=S1/C0
        S0=S0/C0-(S1)**2
        S1=S1/XM
        S0=(SQRT(S0))/XM
        WRITE(1,400) S1,S0,XM
        WRITE(IL,400)S1,S0,XM
400     FORMAT(' MEAN AND STD. DEVIATION',2G14.8,' FOR M=',I3)
        WRITE(1,410)
410     FORMAT(' ENTER ITEST=1 FOR CONTINUANCE')
        READ(2,*) ITEST
        IF (ITEST.NE.1) GO TO 500
        GO TO 100
500     CONTINUE
        STOP
        END
```

FIGURE 9

X	Y	Z_1	Z_2	ERROR
16	0	1.000000	0.000000	-1.000000
16	1	1.000000	0.533333	-0.466667
16	2	1.000000	1.066667	0.066667
16	3	1.000000	1.600000	0.600000
16	4	3.000000	2.133333	-0.866667
16	5	3.000000	2.666667	-0.333333
16	6	3.000000	3.200000	0.200000
16	7	3.000000	3.733333	0.733333
16	8	6.000000	4.266667	-1.733333
16	9	6.000000	4.800000	-1.200000
16	10	6.000000	5.333333	-0.666667
16	11	6.000000	5.866667	-0.133333
16	12	8.000000	6.400000	-1.600000
16	13	8.000000	6.933333	-1.066667
16	14	8.000000	7.466667	-0.533333
16	15	8.000000	8.000000	0.000000
16	16	10.000000	8.533334	-1.466666
16	17	10.000000	9.066667	-0.933333
16	18	10.000000	9.600000	-0.400000
16	19	10.000000	10.133333	0.133333
16	20	13.000000	10.666667	-2.333333
16	21	13.000000	11.200000	-1.800000
16	22	13.000000	11.733334	-1.266666
16	23	13.000000	12.266666	-0.733334
16	24	15.000000	12.800000	-2.200000
16	25	15.000000	13.333333	-1.666667
16	26	15.000000	13.866667	-1.133333
16	27	15.000000	14.400000	-0.600000
16	28	17.000000	14.933333	-2.066667
16	29	17.000000	15.466666	-1.533334

MULTIPLIER OPERATION

FIGURE 10

σ_e=.00379230. In the statistical mode, the results were e=-.00023643 and σ_e=.00396498, which can be seen to be in close agreement. Table 8 and Figure 11 summarizes the results of several experiments. They are:

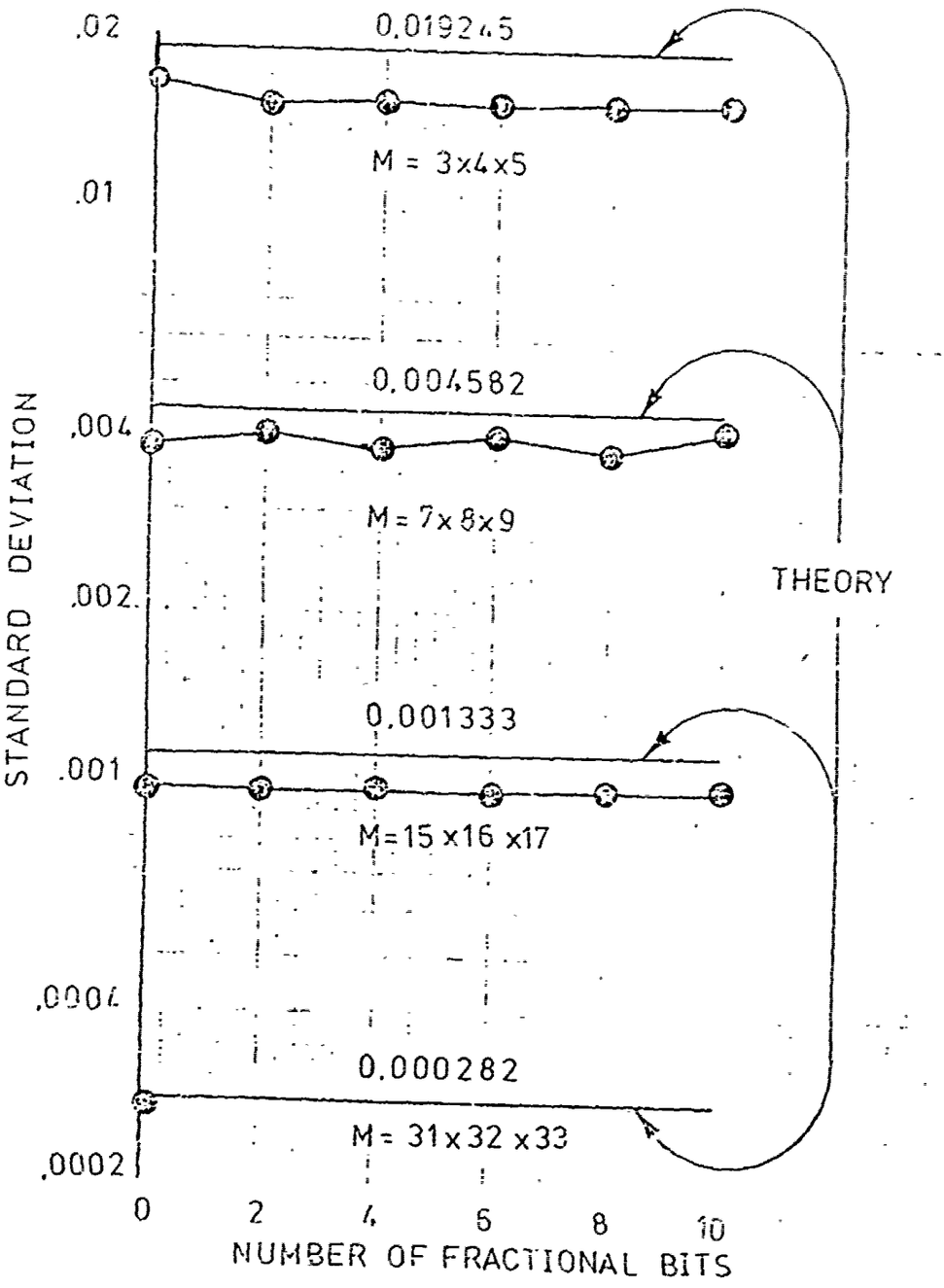
1. Deterministic for P={3,4,5}
2. Statistical for P={7,8,9}
3. Statistical for P={15,15,17}
4. Statistical for P={31,32,33}

for various choices of fractional-bit accuracy (denoted NN). The error standard deviation data has been interpreted in graphical form in Figure 6 and compared to usual theoretical model given by $\sigma_e^2=Q^2/12$ or $\sigma_e=Q/\sqrt{12}$. Here Q is the quantization step size which, over S_2, is given by $Q=1/p_1p_3$. The data is shown to be in close agreement with the theoretical model. Lastly, it can be observed that the multipliers performance is more-or-less invariant to the number of fractional bits used to generate the tables.

DETERMINISTIC ANALYSIS				
MODULI CHOICE P1,P2,P3=	3.	4.	5.	
MEAN AND STD. DEVIATION	-0.00178704	0.01487452	FOR NN=	0
MEAN AND STD. DEVIATION	-0.00710185	0.01502258	FOR NN=	2
MEAN AND STD. DEVIATION	-0.00583796	0.01524284	FOR NN=	4
MEAN AND STD. DEVIATION	-0.00217014	0.01544447	FOR NN=	6
MEAN AND STD. DEVIATION	-0.00227643	0.01543744	FOR NN=	8
MEAN AND STD. DEVIATION	-0.00229941	0.01543648	FOR NN=	10
STATISTICAL ANALYSIS				
MODULI CHOICE P1,P2,P3=	7.	8.	9.	
MEAN AND STD. DEVIATION	-0.00023643	0.00294498	FOR NN=	0
MEAN AND STD. DEVIATION	-0.00067036	0.00414181	FOR NN=	2
MEAN AND STD. DEVIATION	-0.00110580	0.00383812	FOR NN=	4
MEAN AND STD. DEVIATION	-0.00089492	0.00406694	FOR NN=	6
MEAN AND STD. DEVIATION	-0.00116032	0.00389042	FOR NN=	8
MEAN AND STD. DEVIATION	-0.00087850	0.00415991	FOR NN=	10
STATISTICAL ANALYSIS				
MODULI CHOICE P1,P2,P3=	15.	16.	17.	
MEAN AND STD. DEVIATION	-0.00000465	0.00098343	FOR NN=	0
MEAN AND STD. DEVIATION	-0.00009251	0.00098051	FOR NN=	2
MEAN AND STD. DEVIATION	-0.00011501	0.00098657	FOR NN=	4
MEAN AND STD. DEVIATION	-0.00011647	0.00098206	FOR NN=	6
MEAN AND STD. DEVIATION	-0.00012014	0.00098343	FOR NN=	8
MEAN AND STD. DEVIATION	-0.00012051	0.00098035	FOR NN=	10
STATISTICAL ANALYSIS				
MODULI CHOICE P1,P2,P3=	31.	32.	33.	
MEAN AND STD. DEVIATION	0.00001294	0.00027552	FOR NN=	0

EXPERIMENTATION SUMMARY

TABLE 8



ERROR ANALYSIS

FIGURE 11

PART II SYSTEM DESIGN IN THE RNS

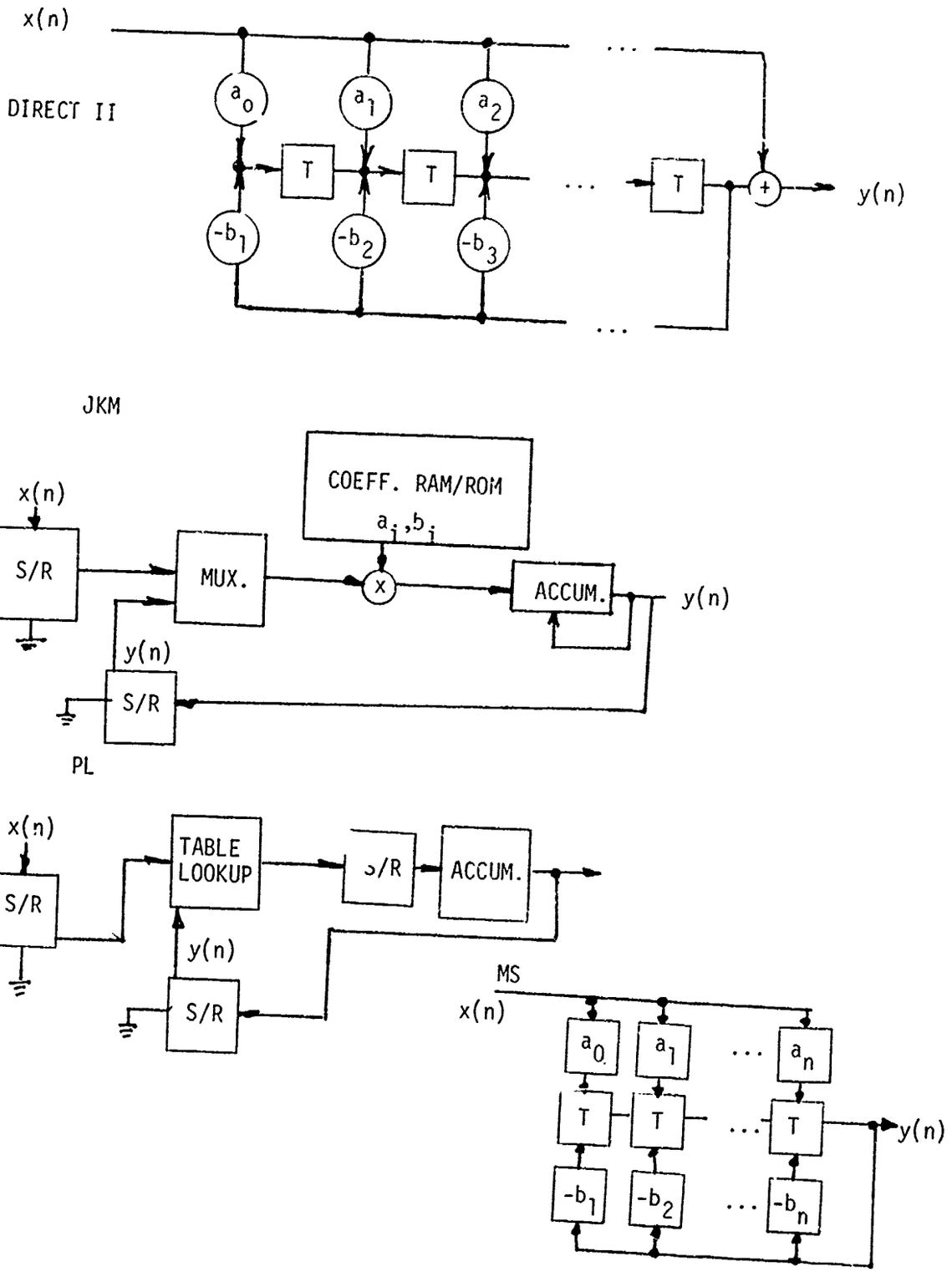
Under the AFOSR grant, new residue arithmetic units were conceptualized, researched, and tested. Key breakthroughs were an efficient RNS to decimal converter and an autoscale multiplier. In this section, these building blocks will be put to use in designing high speed digital systems.

The utility of the RNS in digital filtering was forwarded by Jenkins and Leon [1] through their work in non-recursive filtering (FIR: finite impulse response). In this case the problem of register overflow, in the RNS, was overcome through the use of a ℓ_1 norm argument. Given a FIR, satisfying $y(n)=\Sigma a_i x(n-i)$, $i=1,\ldots,N$, with $|x(j)|\leq|1$, it follows that $|x(n)|_1\leq\Sigma|a_i|=V$ over all i. In order to insure that dynamic range overflow will not occur, the RNS dynamic range $M=\pi p_i$ would be chosen so that $M>V$. However, the design of recursive filters (FIR: infinite impulse response) is significantly more complex. Soderstrand [4] approached the problem through base-extension methods. Other authors have used the Chinese Remainder Theorem (CRT) or mixed radix conversion algorithm to control dynamic range. This has been strongly criticized because of the overhead normally associated with these operations. The RNS concepts, developed in Part I, overcome many of these objections.

The classic digital filter architecture, often referred to as the Jackson-Kaiser-McDonald (JKM) filter, realizes a filter in terms of general purpose multipliers, adders, and shift-registers. In the mid-1970's, several new memory-intensive linear shift-invariant digital filter architectures were introduced. First, the distributed filter [Peled-Liu], or PL filter, was introduced in 1974.[10] Next, the Monkewich-Stunaart, or M-S, filter was reported in 1975.[20] All three architectures are summarized in Figure 12. Compared to conventional architectures, this class of memory intensive filters offer the potential for high throughput. Execution speed is achieved by replacing the relatively slow process of general digital multiplication with table lookup scaling operations. Jenkins and Leon, in 1977, studied a memory intensive filter architecture based on residue [modular] arithmetic.[1] By exploiting the parallel nature of the residue numbering system, and using table lookup operations to mechanize modular arithmetic, ultra-high speed digital JKM filters were realized (see Figure 13). In most reported cases, a residue arithmetic filter is organized into a decimal to residue encoder stage, arithmetic-filter section, and residue to decimal converter stage.[4,6] In this work, the fundamental structure of the residue arithmetic-filter section will be developed and new results presented.

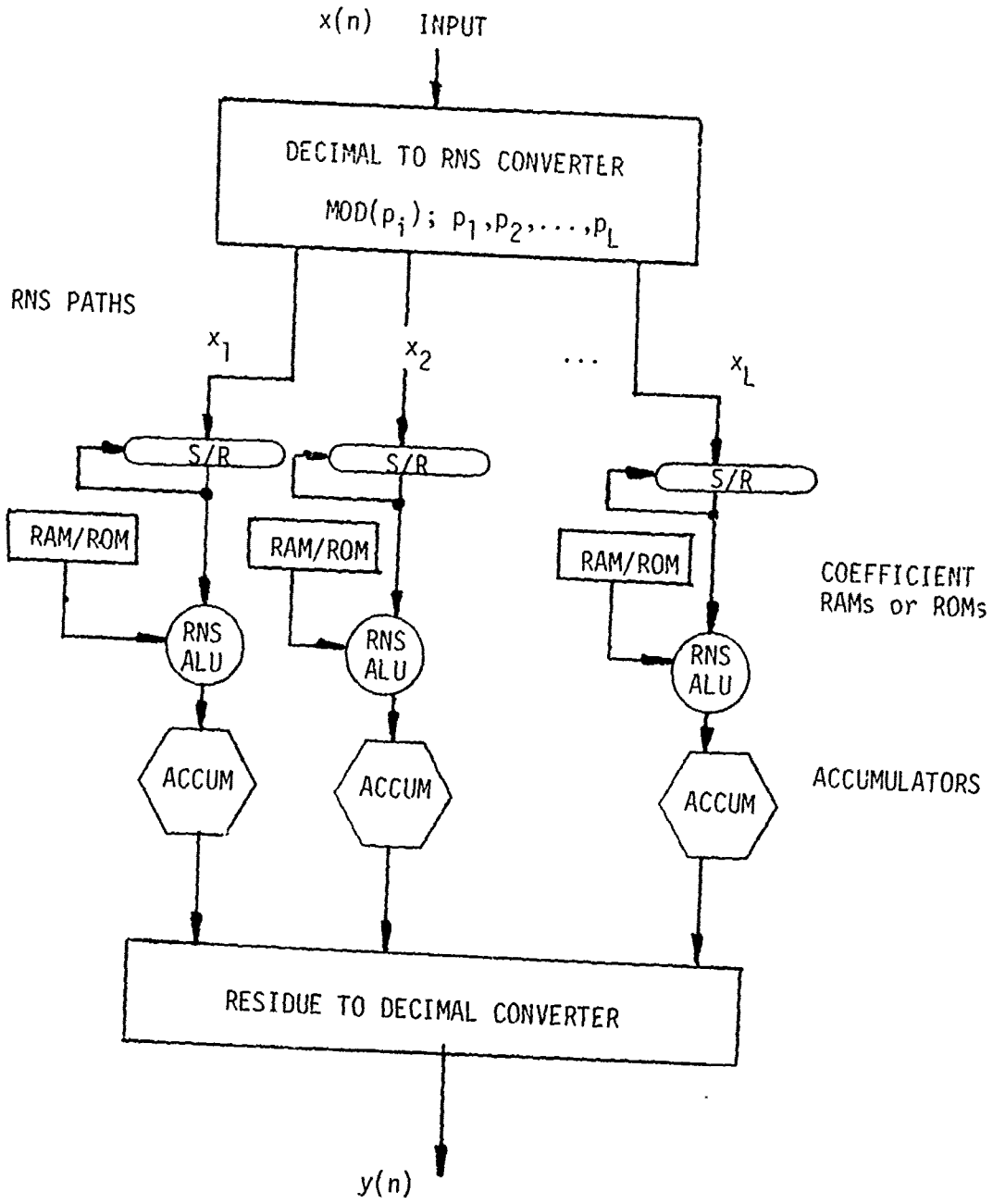
One of the principal limitations to the residue concept is its intolerance of register overflow. This is a consequence of finite ring theory. Specifically, for a set of relatively prime moduli, say $P = \{p_1, p_2, \ldots, p_L\}$, the residue representation for a signal integer i, is given by $i \rightarrow (i_1, i_2, \ldots, i_L)$, where

$$i_j = \left\{ \begin{array}{l} i \bmod p_j \text{ if } i \geq 0 \\ (M-|i|) \bmod p_j \text{ if } i<0 \end{array} \right\}$$



ARCHITECTURES

FIGURE 12



FIR RNS FILTER ARCHITECURE

FIGURE 13

and $0 \leq i_j < p_j$, $M = \prod_{k} p_k$, $k=1,2,\ldots,i$.. The integer i will have a unique representation if and only if $-M/2 < i < M/2$.

In order to insure the satisfactory performance of a residue arithmetic filter, dynamic range overflow cannot be tolerated. If for example, a shift-invariant filter of the form $y(k+1)=\Sigma a_i y(k-i)+\Sigma b_i x(k-1)$ is considered, the ℓ_{00} bound on y(j) (ie: max (y(j)) for all j) must be less than M/2 otherwise uniqueness can be guaranteed. As a result, the JKM residue arithmetic filter suffered from a severe dynamic range restriction. For example, in order for z=ax+by to be correctly represented in a residue system z must be bounded by M. For $0 \leq a < A$, $0 \leq b < B$, $0 \leq x < X$, $0 \leq y < Y$, then $AX+BY<M$. If A,B,x,y are on the order of r-bits of precision, then M must be on the order of 2r+1-bits in range. However, this is not the only constraint. If highspeed RAM or ROM is to be used to perform the algebra (in a lookup sense), then for n=14, a table addressing space of 29-bits would be required. Suppose, for the sake of discussion, M≈33-bits and r≈16 bits. Using a 16K highspeed memory unit (30-50 nsec) as a model,[1] the maximum value of a moduli p_i is $2^7=128$. In order to achieve the 33-bit system dynamic range (ie: M), at least 5 (ie: [33/7]) moduli, on the order of 7 bits each, must be used. This means that five parallel paths, complete in memory and logic, must be configured and integrated into a complete system.

Footnote 1: 16Kx1 units: INTEL 2167: access line=40ns, enable time=40ns, cycle time=40ns, active power 500mW, standby power=75mW: INMOS 1400, access time=30ns, enable time=35ns, cycle time 30ns, active power=375ns, standby power=35ns.

M-S RESIDUE ARCHITECTURE (MSR)

The algebraic operations found in a linear shift invariant filter are data delays, additions, and scaling (in lieu of general multiplication). Replacing general multipliers with residue scalars has been proposed by several authors.[1,2,14,18,19] A residue multiplier would present a 2r-bit two tuple (a_i,x_i) to a 2r word memory unit and respond with the precomputed value of $(a_i x_i)$ mod p_i. Using the same 2r-word memory unit, a scalar would accept a 2r-bit representation of x_i. The table would respond with the precomputed value of $(a_i x_i)$ mod p_i where a_i is known apriori. For example, using three 16K NMOS 30 nsec static RAMs and three moduli of the form $P=\{2^n-1, 2^n, 2^n+1\}$, a scalar having a dynamic range on the order of 42-bits can be realized. Using such scalars, the M-S filter architecture found in Figure 12 can be realized.

FINITE WORDLENGTH EFFECTS

Generally, a digital filter is a finite precision approximation to some user defined discrete filter defined over a real coefficient field. The errors, due to finite wordlength effects, are well documented. It is generally assumed that the expected truncation error variance, per multiplier, is given by $Q/2$ and $Q^2/12$ respectively (Q represents quantization level). However, in a residue arithmetic filter must be defined over a ring of integers. Real numbers cannot be tolerated. For example, suppose $a=3.251$ and $x=10$, then $ax=32.51$. Rounding this results would yield an estimated product 33. In a residue sense, with respect to a moduli set $P=(3,4,5)$; $M=60$, $x \xrightarrow{RNS} (x_1,x_2,x_3)=(1,2,0)$, one may make two sets of calculations, namely (i) and (ii).

i)	ii)
$a = 3.251$	$[a] = 3$
$ax = 32.51$	$ax = 30$
$\langle ax_1 \rangle_3 = \langle 3.251 \rangle_3 = .251;\ [251] = 0$	$\langle ax_1 \rangle_3 = \langle 3 \rangle_3 = 0$
$\langle ax_2 \rangle_4 = \langle 6.502 \rangle_4 = 2.502;\ [2.502] = 3$	$\langle ax_2 \rangle_4 = \langle 6 \rangle_4 = 2$
$\langle ax_3 \rangle_5 = \langle 0 \rangle_5 = 0;\ [0] = 0$	$\langle ax_3 \rangle_5 = \langle 0 \rangle_5 = 0$

The calculations found in column i use the decimal value of a in forming product (ax) modulo p_i. The resulting products are then rounded. The final residue digits are (0,3,0) which is equivalent to a decimal value of 15. In column ii, the integer value of a is used to form the product ax in the usual residue arithmetic sense. The result is seen to produce the correct truncated value of product, namely $(0,2,0) \xrightarrow{RNS} 30$. Therefore, since all filter parameters are to be integer value over [0,M], traditional finite wordlength error modeling and analysis techniques apply.

If a large dynamic range is required, in limited RNS hardware, magnitude scaling is required. A similar strategy is used in designing filters using weighted fixed point arithmetic where rounding or trunication is used to control the growth of dynamic range. In a RNS system, the problem is compounded by the fact that the magnitude of a number must be known, if it is to be scaled, and magnitude determination in the RNS is difficult. That is, in order to support scaling in the residue number system, some sort of residue-to-decimal conversion is required. Most existing residue scaling routines makes use of base extension or mixed radix conversion schemes.[3] It has been shown in reference [15] that in a realistic RNS system, a ten to twenty-fold increase in computational overhead can be expected if scaling is present. As a result, the overall throughput of an IIR-RNS filter would be compromised.

In order to achieve high data rates, over realistic dynamic ranges, in limited hardware, a new low-overhead RNS arithmetic unit must be developed. In the next section, such a unit will be presented.

M-K RNS FILTER ARCHITECTURE

In order to insure the uniqueness of a modular product of two numbers of dynamic range V, the modular dynamic range must be bounded by V^2 (ie: $V^2 \geq M$). This can be achieved through the use of a newly developed auto-scale policy. The auto-scale arithmetic units will be shown to support memory intensive filter architectures. Assumed that there is a practical memory wordsize constraints. For high-speed ($\sim$30 ns) applications, memory size is presently limited to 4 to 16K words (ie: 12 to 14-bits).

For reasons that will become self-evident later in this section, the three moduli system, given by $P=\{p_1=2^n-1,\ p_2=2^n,\ p_3=2^n+i\}$, will be considered.

Using such a moduli choice, signed integers $X\varepsilon[[L-M/2], [M/2]]$ are uniquely represented by the three-tuple (x_1,x_2,x_3) where $x_j=x_i \bmod p_j$. In order to scale x, using memory table lookup operations, the magnitude of x must be known. That is, the RNS three-tuple (x_1,x_2,x_3) must be simultaneously presented to a memory module which is programmed to output $(cx) \bmod p_j$ where $cx\varepsilon[[L-M/2], LM/2]]$. For high-speed application, the limiting 16K = 2^{14} memory units require $\Pi_1^3 p_i \underline{\sim} 2^{3n} \underline{\sim} 2^{14}$. That is, the design would be constrained to consider moduli on the order of 4-bits each. Also, the dynamic range is limited to 14-bits. Referring to Figure 7, it can be observed that an integer $\bar{x}\varepsilon[L-M/2], LM/2]]$, satisfying $\bar{x}=kp_2$, $0\underline{\leq}k\underline{\leq}p_1p_3-1$, has the unique RNS description

$$\begin{aligned} \bar{x} \xrightarrow{RNS} & ((x_1-x_2)\bmod p_1,\ (x_2-x_2)\bmod p_2,\ (x_3-x_2)\bmod p_3) \\ &= ((x_1-x_2)\bmod p_1,\ 0,\ (x_2-x_2)\bmod p_3) \\ &\triangleq (\bar{x}_1,0,\bar{x}_3) \end{aligned} \qquad 37.$$

Observe that $\bar{x}$ is defined over a subcover of [L-M/2], LM/2]], and it can be uniquely represented as the <u>two-tuple</u> $(\bar{x}_1, \bar{x}_3)$. The two tuple approximation of x, namely $\bar{x}$, establishes a memory size constraint given by $2^{2n}<2^{14}$ or $n\underline{\leq}7$. Now 7-bit (vs. 4-bit) moduli are admissible) with the dynamic range extended to 3n or 21-bits. The memory units can be programmed to output $[cx] \bmod p_i$ where c is a user specified constant. Overflow prevention can be achieved by introducing a scale factor K so that [cx/K]=[c'x] will not exceed the permitted RNS dynamic range. For the application under study (viz: IIR-RNS filtering), it shall be assumed that all system variables and

constants belong to the integer range [-M/2, M/2) where $M=p_1p_2p_3=2^{3n}-2$.[i] Therefore, the scale factor K needed to insure the absence of arithmetic overflow, is K=M/2.

The error statistics associated with each auto-scaled multiplication can be shown to be bounded in mean by $p_2c/2M$ and in variance by $\sigma^2=p_2^2/36M^2$. This has been experimentally verified: For example, for P = (15,16,17) and (255,256,257), the error variance for the integer valued product y=cx, for $c\in[0, M]$ given and $x\in[0,M]$ randomly choosen, is plotted in Figures 14 and 15. The error is defined to be $e=(cx/M-[c\bar{x}/M])$ where $\bar{x}=kp_2$, $k\in[0,p_1p_3-1]$.

A M-S recursive RNS filter can be architected using the auto-scale arithmetic unit. A dedicated auto-scale unit must be configured for each unique filter coefficient. Each unit, in the three-moduli case, would require three memory devices each. For example, a 9 coefficient 21-bit resolution filter, based on 4Kx1 RAM/ROM, would require

N = 9 (coef.) × 3 (moduli) × 6 (wordlength per moduli) = 162 (16Kx1) memory units

A detailed description of an autoscaled arithmetic unit is found in Figure 16. The modulo 2^n-1 and 2^n+1 adders can be realized in the manner suggested by Taylor.[19] This architecture uses PLA's to augment a conventional n-bit integer adder. Other realizations have been reported in the open literature.[13] For example, a modulo 2^n-1 adder can be realized in a simple straightforward

---

Footnote i: Unlike a 2's compliment system, where partial sum overflow can be tolerated if the complete sum of products is bounded, each partial sum must be bounded to [-M/2, M/2) in the RNS.

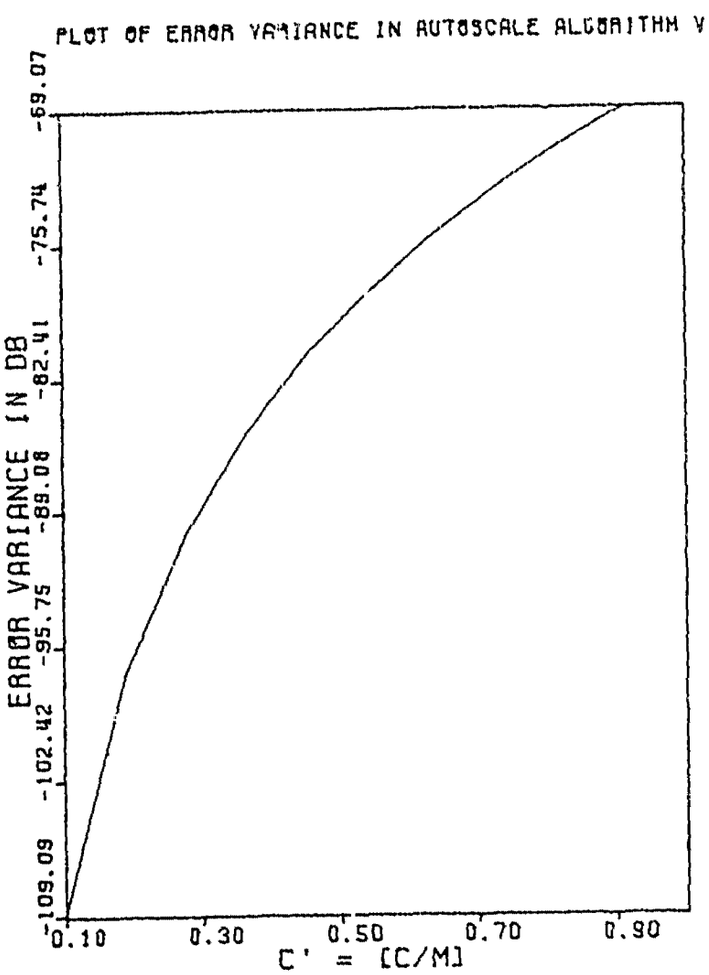


FIGURE 14

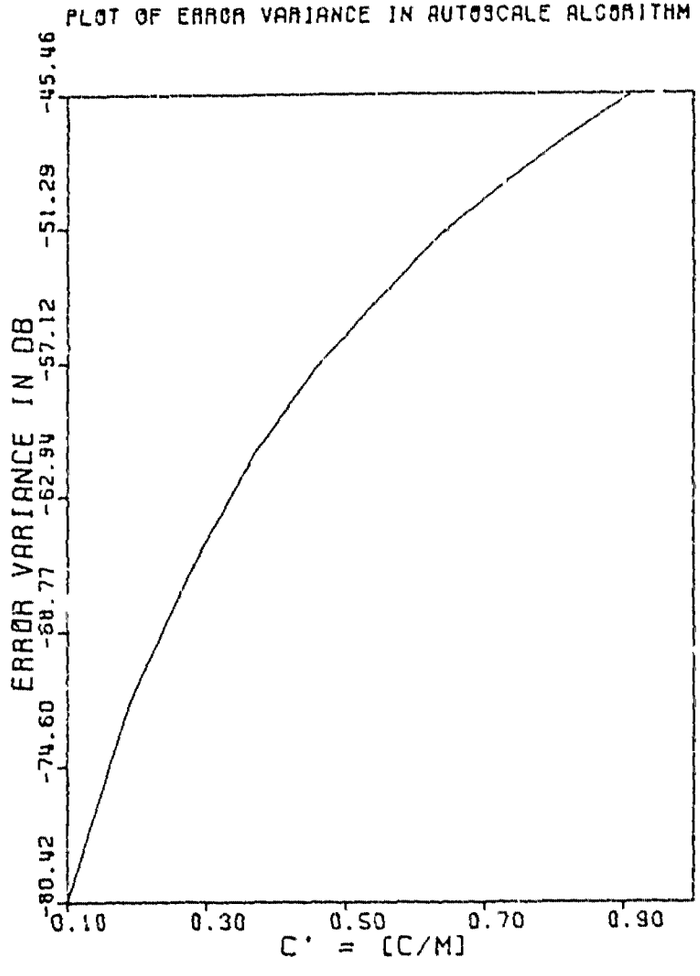
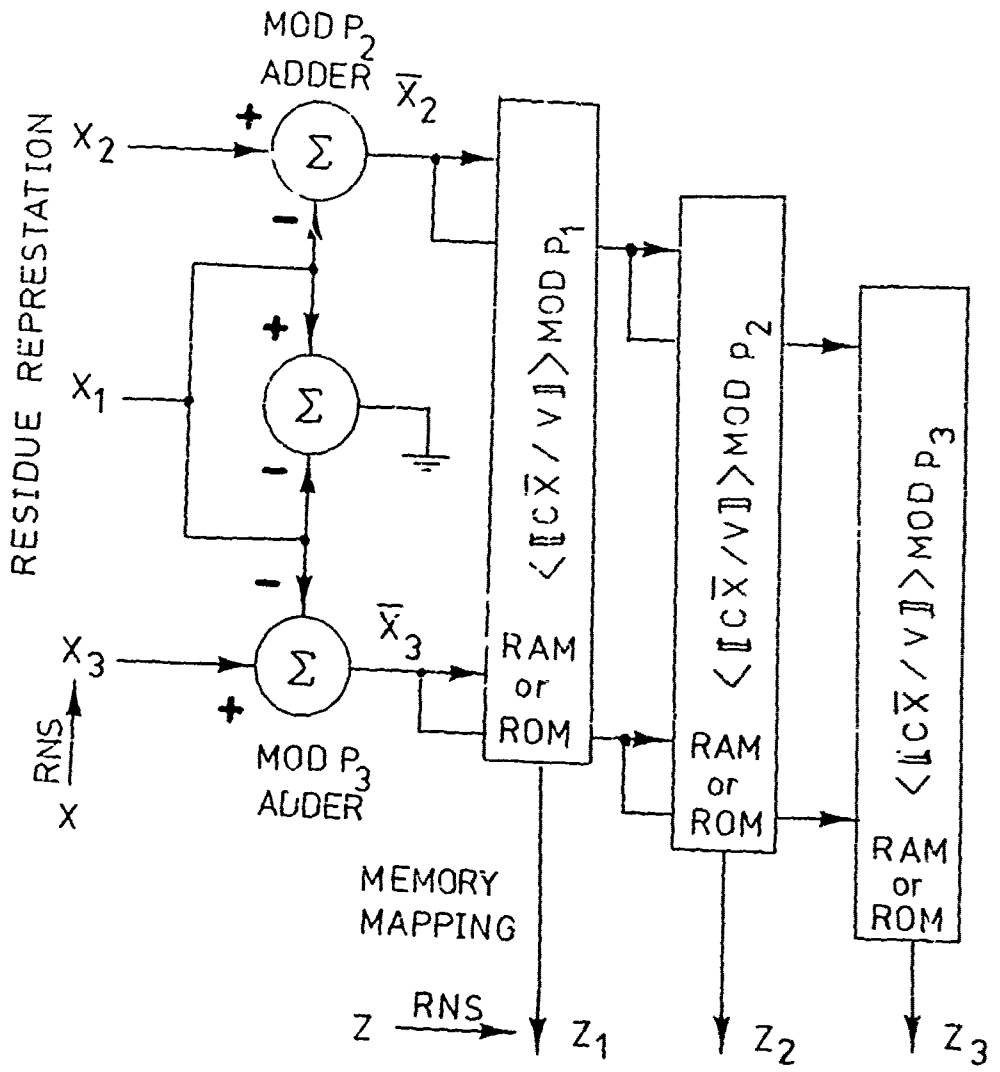


FIGURE 15



THREE MODULI EXTENDED RANGE
AUTOSCALE ARCHITECTURE

FIGURE 16

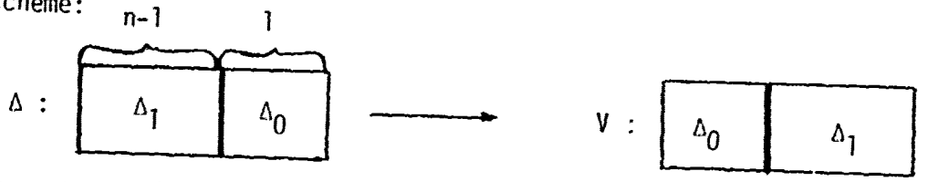
manner. Consider the term (x_1-x_2) mod 2^n-1 with $x_1 \varepsilon Z_{2^n-1}$ and $x_2 \varepsilon Z_{2^n}$. For (x_1-x_2) positive, (x_1-x_2) mod $2^n-1=(x_1-x_2)$ but for (x_1-x_2) negative, (x_1-x_2) mod $2^n-1=|x_1-x_2|^c_{[2]}$ where $|x_1-x_2|^c_{[2]}$ denotes the complement of the binary representation of $|x_1-x_2|$. For example, -5 mod 7 = $\overline{101}_{[2]}=010_{[2]}=2$.

If a real-time, or pipelined architecture is required, then it's desired to design the modular adder which have identical propogation delay. Using the PLA-supported architecture, modulo $2^n\mp1$ adders can be realized in commercially available hardware, for $n\leq12$, having a 30 nsec delay.

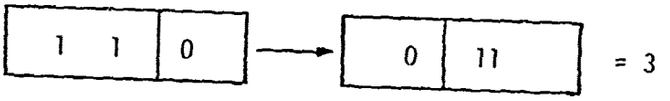
A second arithmetic operation found in the three moduli namely the computation of $v=(2^{n-1}\Delta)$ mod 2^n-1, where $\Delta=\{(x_1-x_2)\text{mod } 2^n-1\} - \{(x_2-x_3)\text{mod } 2^n+1\}$, can be simply computed. It is directly verifiable that $\Delta\varepsilon[-2^n,2^n-2]$. Consider

$$\Delta = \begin{cases} 2\Delta_1+\Delta_0 & \text{if } \Delta\geq 0 \\ -2\Delta_1-\Delta_0 & \text{if } \Delta<0 \end{cases} \qquad 38.$$

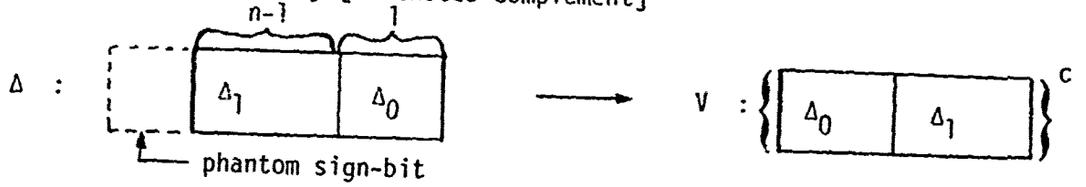
where $\Delta_0 \varepsilon Z_2$ and $\Delta_1 \varepsilon Z_{2^{n-1}}$. For $\Delta\geq0$, v can be computed using the following scheme:



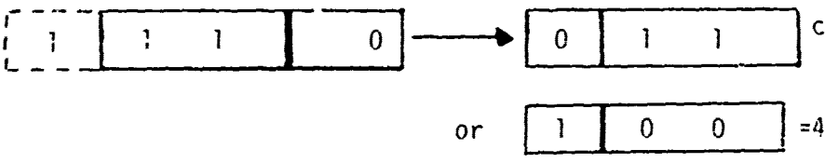
Example: $\Delta=6$, n=3, (4·6) mod 7=3.



For $\Delta<0$, v is given by [c denotes complement]



Example: =-6, n=3, (-6·4) mod 7=4



As a result, v can be computed with negligable overhead and hardware.

then

Δ	$(2^{n-1}\Delta) \bmod 2^n-1$	Δ(binary)	Δ'(binary)	Δ'(decimal)
6	$<24>_7=3$	110	011	3
5	$<20>_7=6$	101	110	6
4	$<16>_7=2$	100	010	2
3	$<12>_7=5$	011	101	5
2	$<8>_7=1$	010	001	1
1	$<4>_7=4$	001	100	4
0	$<0>_7=0$	000	000	0

The discussed parmutation, from a modulo 2^n-1 adder to a buffer register, can be realized by a hardware mapping. Here the LSB of the adder is connected to the MSB of the buffer. The other (n-1)-bits are mapped to the buffer with shift of one bit location.

RNS FILTER DESIGN

MK Architecture

In a M-K architecture, eacn filter coefficient c_i is realized with a dedicated RNS table lookup unit. Based on the three moduli MRC algorithm and a given c_i, a MRC multiplier unit similar to the one detailed in Figure would have to be configured. Here, for convenience, the multiplier 2^{n-1}

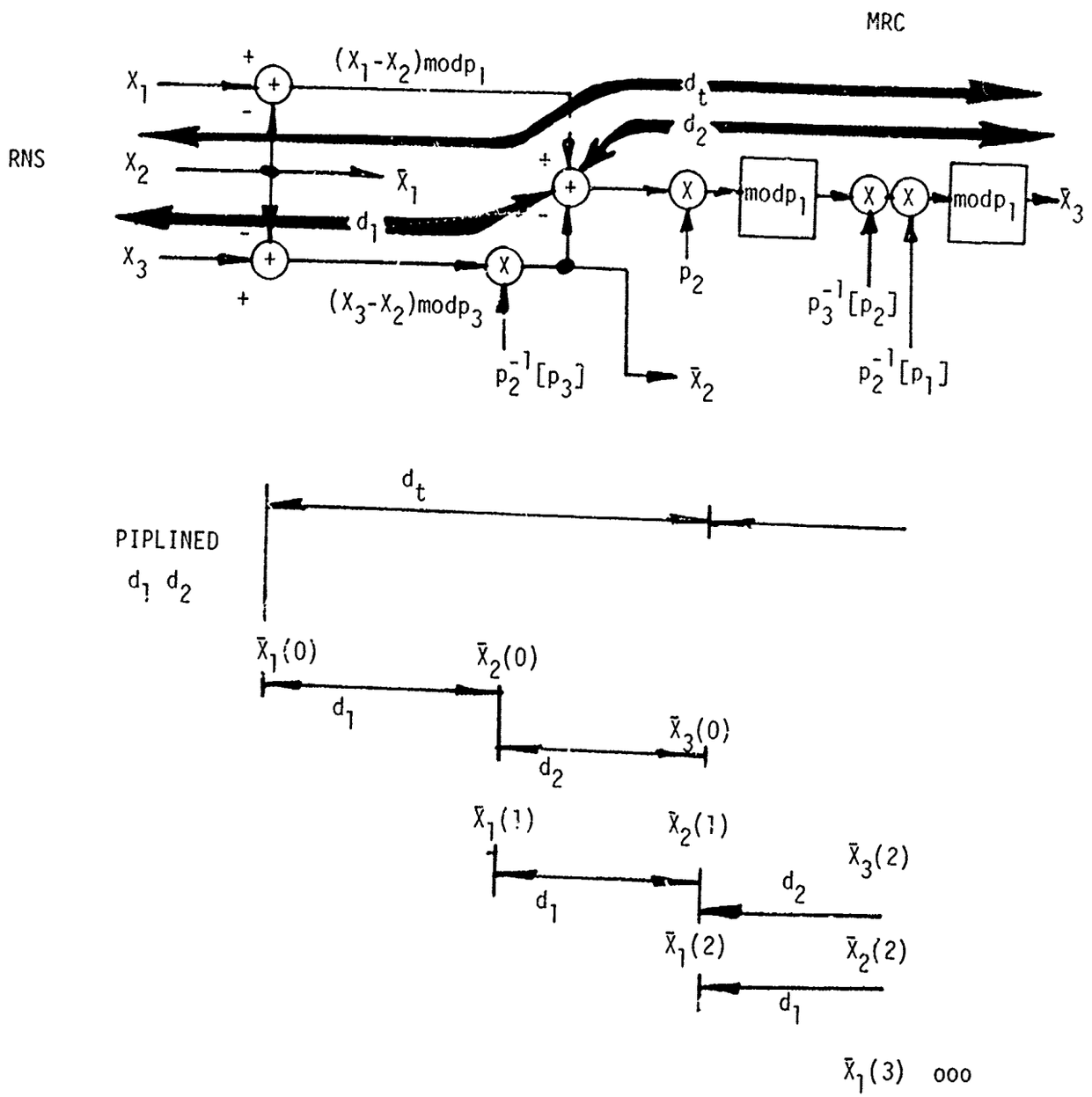
is imbedded into a lookup table. Each unit would consist of nine 2^nxn-bit memory units. It must be stated that in order to use a 2^nxn memory in a modulu (2^n+1) operation, some form of data compression is required. The simplist compression routine would differentiate between the two external number in Z_{2^n+1}, namely 0 and 2^n. Those two numbers have a (n+1)-bit (ie: common n-bit data bus plus 1-bit control line) representation of 00...00 and 10...00 respectively. By ANDing the n LSB's and sensing the MSB, the two conditions can be easily identified. For x=0, it follows that [cx/M]=0 and the output registers would be zeroed without any memory (table lookup) action. This means that one of the 2^n memory addresses, namely x=0, is superfluous and can be assigned to $x=2^n+1$.

It follows directly from the MRC representation that

$$[\frac{xc_i}{M}]\text{mod } p_j = ([\frac{\bar{x}_i c_i}{M}]\text{mod } p_i + [\frac{\bar{x}_2 c_i p_1}{M}]\text{mod } p_i + [\frac{\bar{x}_3 c_i p_1 p_3}{M}]\text{mod } p_j)\text{mod } p_j \qquad 39.$$

That is, the outputs of modular tables (viz: $[\bar{x}_1 c_i/M]\text{mod } p_j,\ldots,[\bar{x}_3 c_i p_1 p_3/M]$ mod p_j) must

From a design standpoint, it is desired to configure a system which has minimum complexity. The 3!=6 possible permutations of a three moduli set are summarized in Table 9 in general and for the specific example x=100 for P=(7,8,9). A key feature of the general architecture are the propogation delay paths d_t (total delay), d_1 and d_2 (see Figure 17). For sequential operation, the MRC digits will be available for use after t_d seconds. The lengh od delay is due primarily to the nesting of four multipliers. In addition, t_d is a function of the multiplication philosophy used (bit-serial,



MRC DIAGRAM AND TIMMING

FIGURE 17

lookup, general purpose, etc.). If high throughputs and low complexity is desired, t_d should be minimized. One could also consider a pipelined architecture of depth two having an effective throughput rate of t_2 second per MRC. The design of an efficient pipelined MRC processor is promised on the condition that t_2 is small and t_1 and t_2 are comparable. Referring to the data summarized in Table 9, it would appear as though the first ordering admits the best design. Therefore, this ordering will be used as the model throughout this section. Based on this model

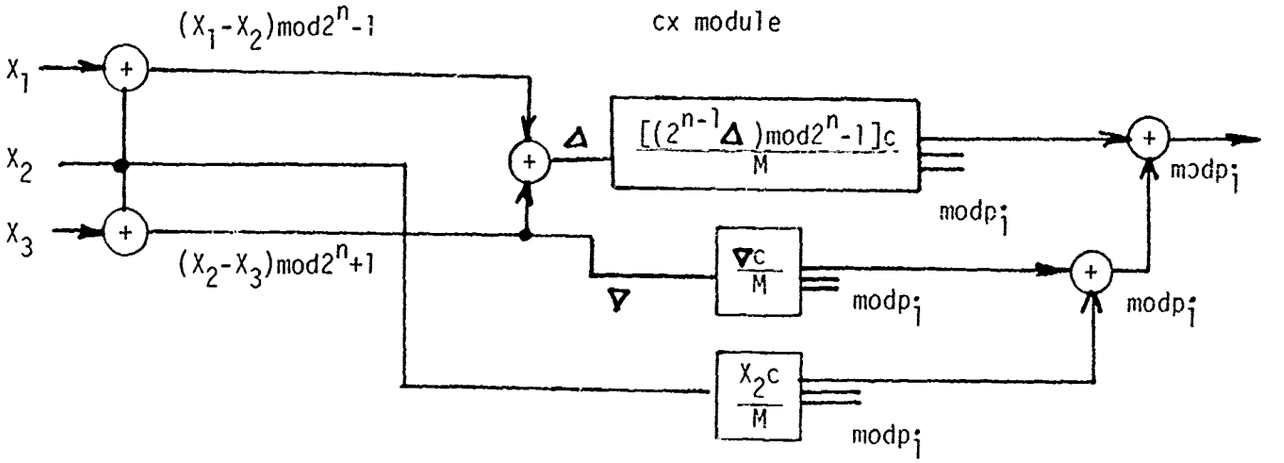
$$\bar{x}_1 = x_2 \qquad 40.$$

$$\bar{x}_2 = (x_2 - x_3) \bmod (2^n+1)$$

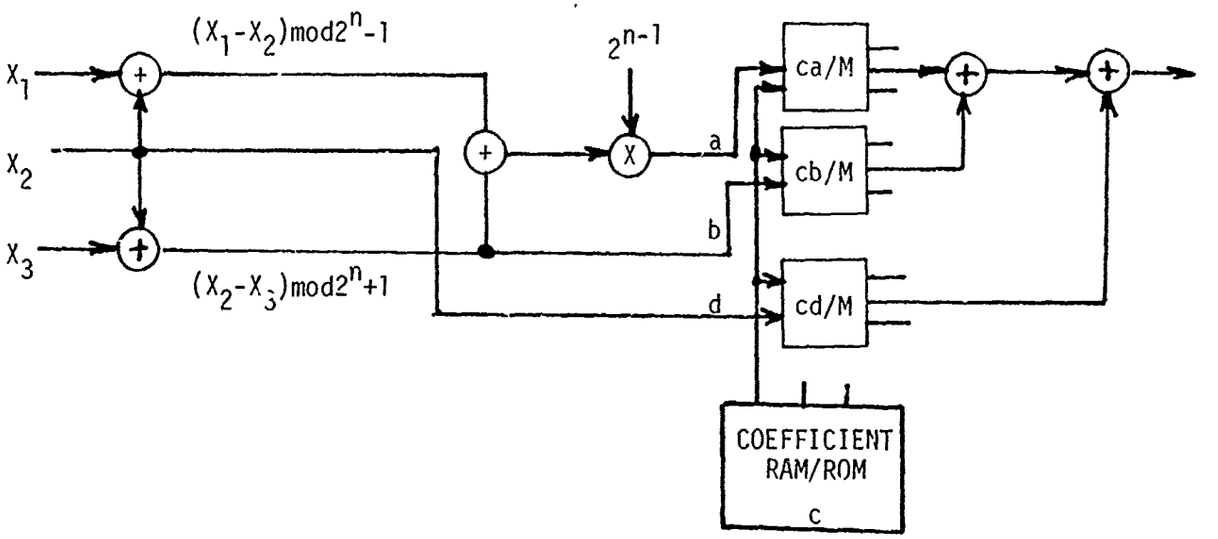
$$\bar{x}_3 = (2^{n-1} \{(x_1 - x_2) \bmod (2^n-1) - (x_2 - x_3) \bmod (2^n+1)\}) \bmod (2^n-1).$$

The 2^n+1 adders found in this architecture have been previously discussed.

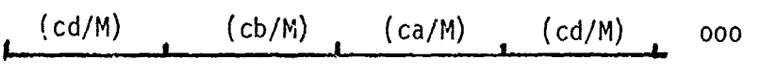
In a M-K architecture, each filter coefficient c_i is realized with a dedicated RNS table lookup unit. Based on the three moduli MRC algorithm and a given c_i, a MRC multiplier-scalar can be configured as suggested in Figure 18. Here, for convenience, the multiplier 2^{n-1} is imbedded onto a lookup table. Each unit would consist of nine 2^nxn-bit memory unit. It must be stated that in order to use a 2^nxn memory in a modulo (2^n+1) operation, some form of data compression is required. The simplist compression routine would differentiate between the two external number in Z_{2^n+1}, namely 0 and 2^n. Those two numbers have a (n+1)-bit (ie: common n-bit data bus plus 1-bit control line) representation of 00...00 and 10...00 respectively. By ANDing and n LSB's and sensing the MSB, the two conditions can be easily identified.



MS FILTER



KAISER FILTER



AUTOSCALE FILTER ARCHITECTURES

$P=(2^n-1, 2^n, 2^n+1)$

FIGURE 18

For x=o, it follows that [cx/M]=0 and the output registers would be zeroed without any memory (table lookup) action. This means that one of the 2^n memory addresses, namely x=o, is superfluous and can be assigned to $x=2^n+1$.

It follows directly from the MRC representation that

$$\left|\frac{XC_i}{M}\right| \bmod p_j = \left(\left|\frac{\bar{x}_1 c_i}{M}\right| \bmod p_i + \left|\frac{\bar{x}_2 c_i p_1}{M}\right| \bmod p_i \left|\frac{\bar{x}_3 c_i p_1 p_3}{M}\right| \bmod p_j\right) \bmod p_j$$

That is, the outputs of modular tables (viz: $|\bar{x}_1 c_i/M| \bmod p_j, \ldots, \bar{x}_3 c_i p_1 p_3/M| \bmod p_j$) must be recombined, in an additive modulo p_j sense. Again, a sequential or pipelined architecture can be realized.

Example

Using an MS architecture, a 4th order Chebyshev filter was realized. The response is reported in Figure 19. It has been experimentally determined that it requires a 16-bit moduli three-tuple to achieve satisfactory performance.

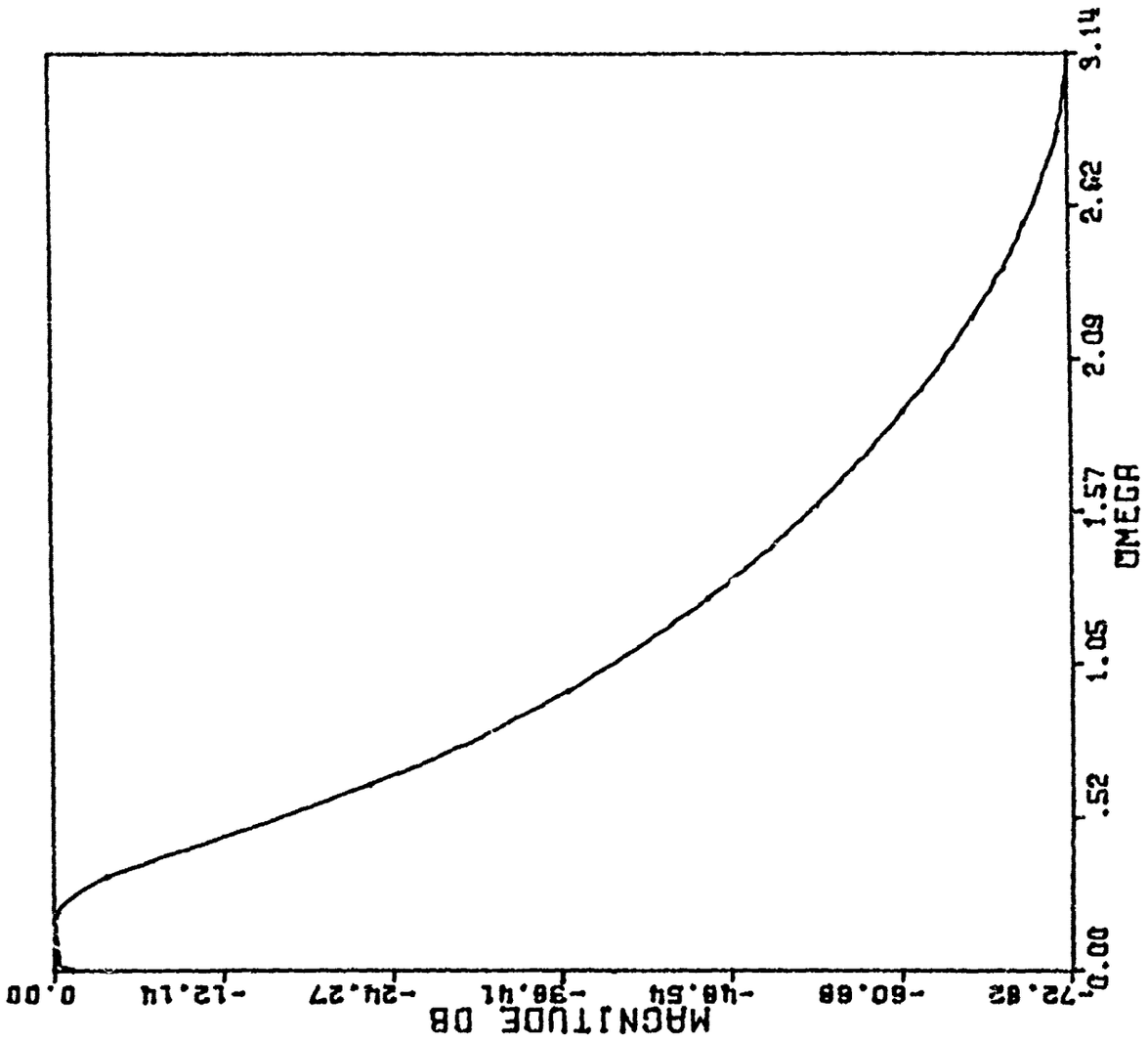

MAGNITUDE DB
0.00
-12.14
-24.27
-36.41
-48.54
-60.68
-72.82
0.00
.52
1.05
1.57
2.09
2.62
3.14
OMEGA

FIGURE 19

JKL-RNS Architecture

The disadvantage of the MRC-RNS-MK filter is the rapid growth of hardware as a function of filter order. In order to reduce hardware complexity associated with RNS-FIR filtering, reusable (undedicated) overflow-free RNS arithmetic units may be considered. Again, if overflow scaling is imbedded the table lookup operations, the magnitude of RNS coded numbers must be known. As previously noted, this has been the historical obstical to IIR filtering in the RNS. The architecture which can achieve this goal is detailed in Figure 18. The multiplier 2^{n-1}, as previously noted, is a zero-overhead operation. A timing diagram is offered in the referenced figure. It is assumed that the modular addition delays are less than the lookup table access times (say t_M). The difficulty with this proposed architecture are the delays associated with reprogramming the tables, for each c_i, from high-density low-speed (comparatively) main memory or mass storage. As a result, the overall throughput of this architecture will be unattractive.

Distributed RNS Filter

A powerfull linear constant coefficient filter policy is distributed arithmetic (alias: bit-serial, bit-slice, or Peled and Liu filter). In a B-bit radix-2 binary weighted, the output of a discrete filter, satisfying

$$y(n)=\sum_{i=1}^{n} a_i y(n-i)+\sum_{i=0}^{n} b_i x(n-i) \qquad 41.$$

is given

$$y(n)=\sum_{j=1}^{B-1}\left(\sum_{i=1}^{n} a_i y(n-i;j)+\sum_{i=0}^{n} b_i x(n-i;j)\right)2^{i}-\left(\sum_{j=0}^{n} a_i y(n-i_j B)-\sum_{j=0}^{n} b_i x(n-i_j B)\right)2^{B}$$

with j denoting bit location. An equivalent statement for RNS systems can be made in the RNS using the MRC. Here, a system variable would be given in MRC form as

$$z=\bar{z}+\bar{z}_2 p_1+\bar{z}_3 p_1 p_2$$

There is a minor structural between a distributed filter using a MRC and radix-2 format. For a three-moduli system, distributed partial products must be recombined modulo p_1,p_2, and p_3. Table requirements, for this architecture, are correlated directly to n, the order of the filter.

Bit Slice

Example: A 2nd order discrete Chelyshev filter was designed in the usual way. The infinite precision response used double precision floating point arithmetic. It can be noted, from the data displayed in Figure 20 that three -8 bit moduli filter performs better than a 12-bit fixed point filter and closely approximates 16-bit precision using a 4th order model, 8-bit moduli can again be seen to offer acceptable performance (see Figure 21).

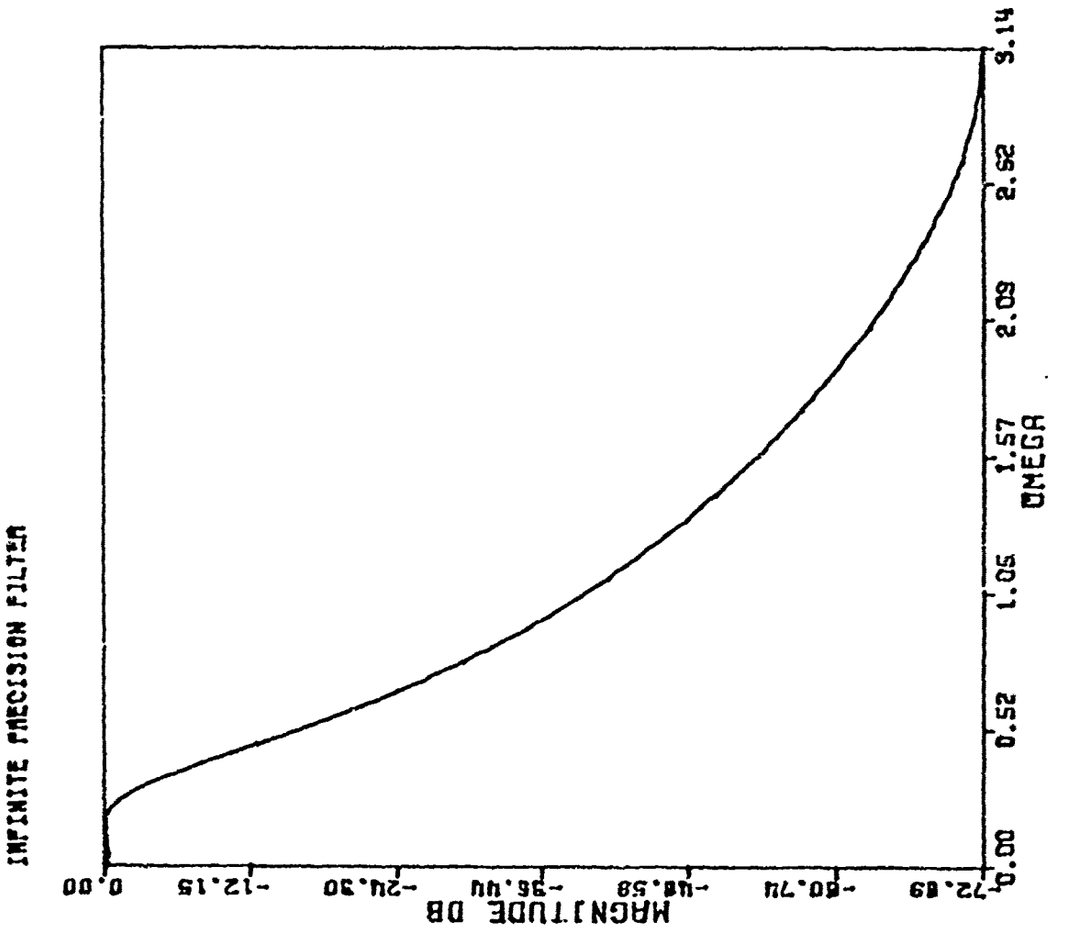

INFINITE PRECISION FILTER
MAGNITUDE DB
0.00
-12.15
-24.30
-36.44
-48.59
-60.74
-72.89
0.00
0.52
1.05
1.57
2.09
2.62
3.14
OMEGA

FIGURE 20 a

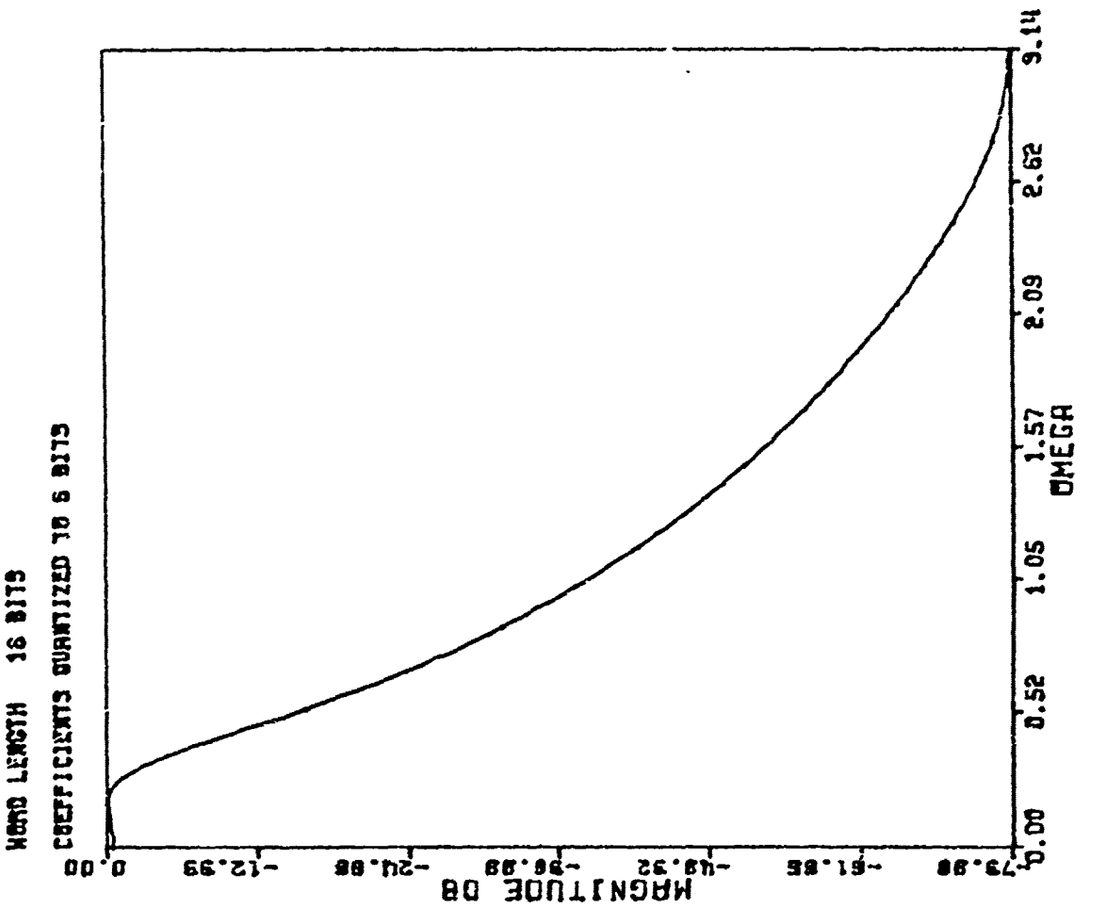

WORD LENGTH 16 BITS
COEFFICIENTS QUANTIZED TO 6 BITS
MAGNITUDE DB
0.00
-12.99
-24.98
-36.98
-48.97
-61.96
-73.96
0.00
0.52
1.05
1.57
2.09
2.62
3.14
OMEGA

FIGURE 20b

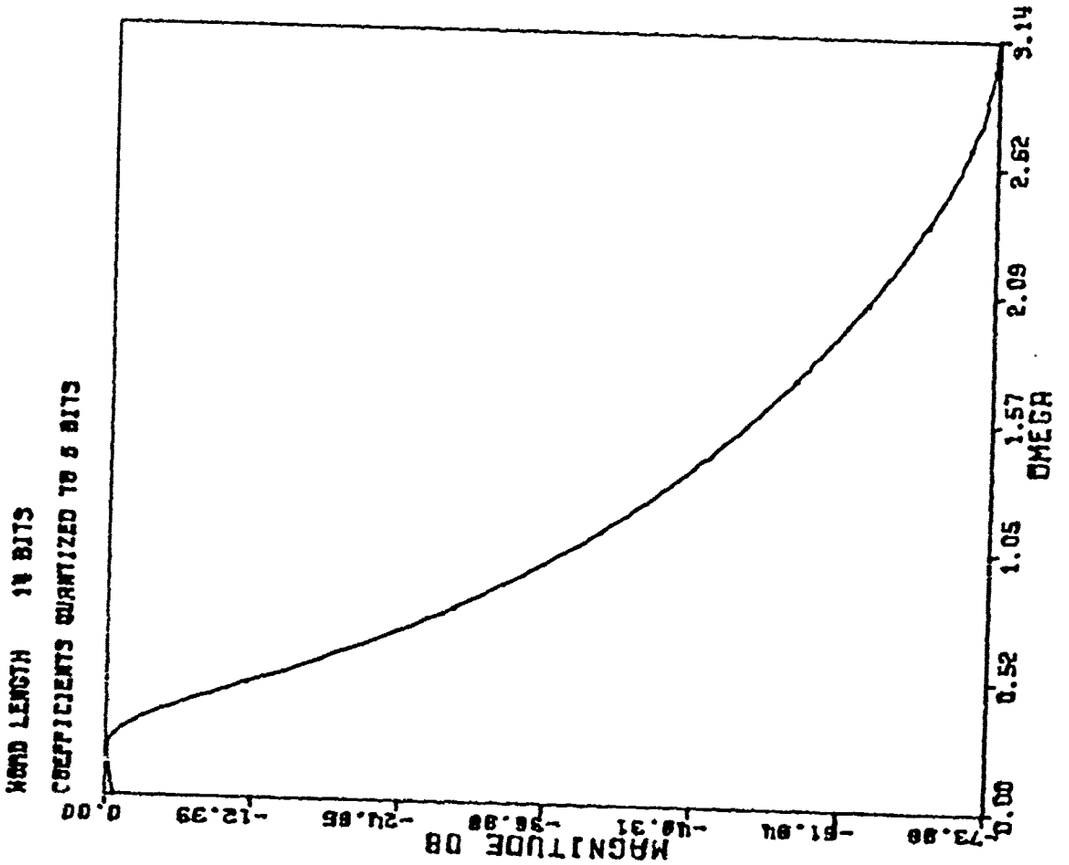

WORD LENGTH 16 BITS
COEFFICIENTS QUANTIZED TO 8 BITS
MAGNITUDE DB
0.00
-12.39
-24.78
-36.98
-48.37
-61.96
-73.35
0.00
0.52
1.05
1.57
2.09
2.62
3.14
OMEGA

FIGURE 20c

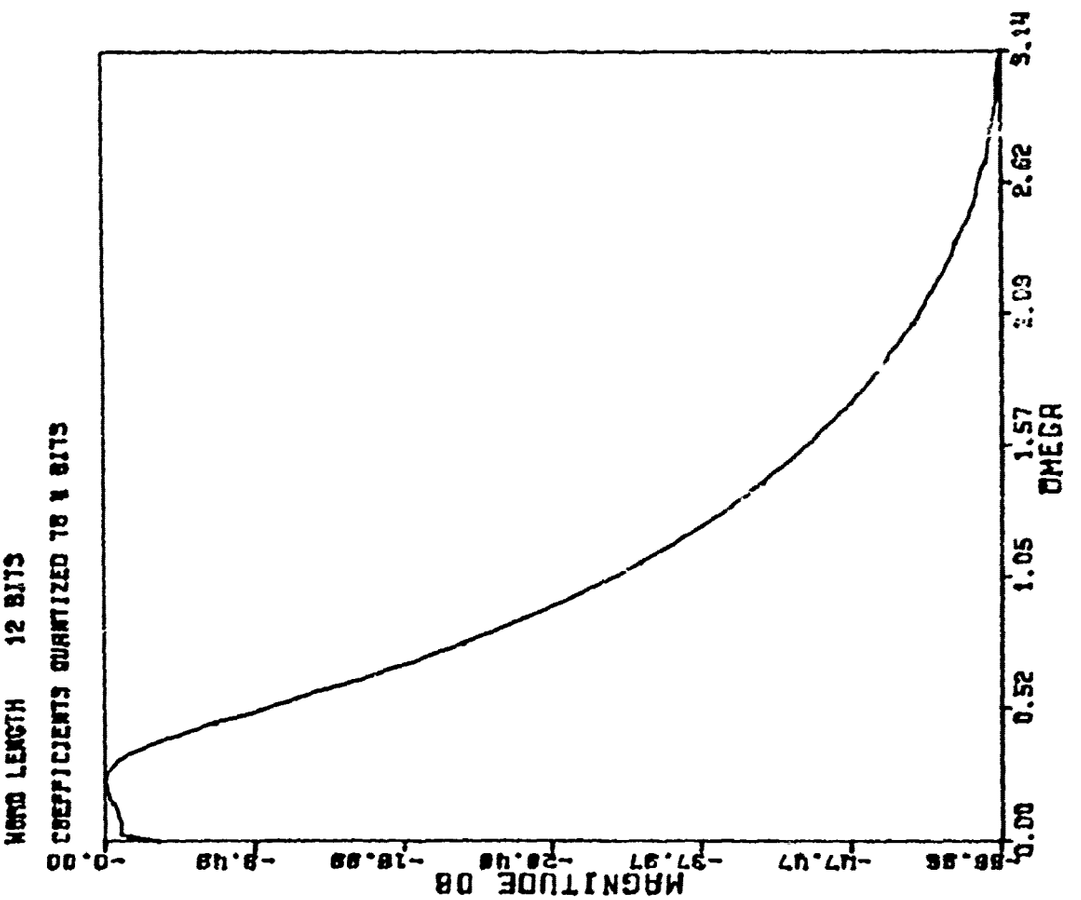

WORD LENGTH 12 BITS
COEFFICIENTS QUANTIZED TO 8 BITS
MAGNITUDE DB
-0.00
-8.43
-18.86
-28.29
-37.72
-47.15
-56.58
0.00
0.52
1.05
1.57
2.09
2.62
3.14
OMEGA

FIGURE 20d

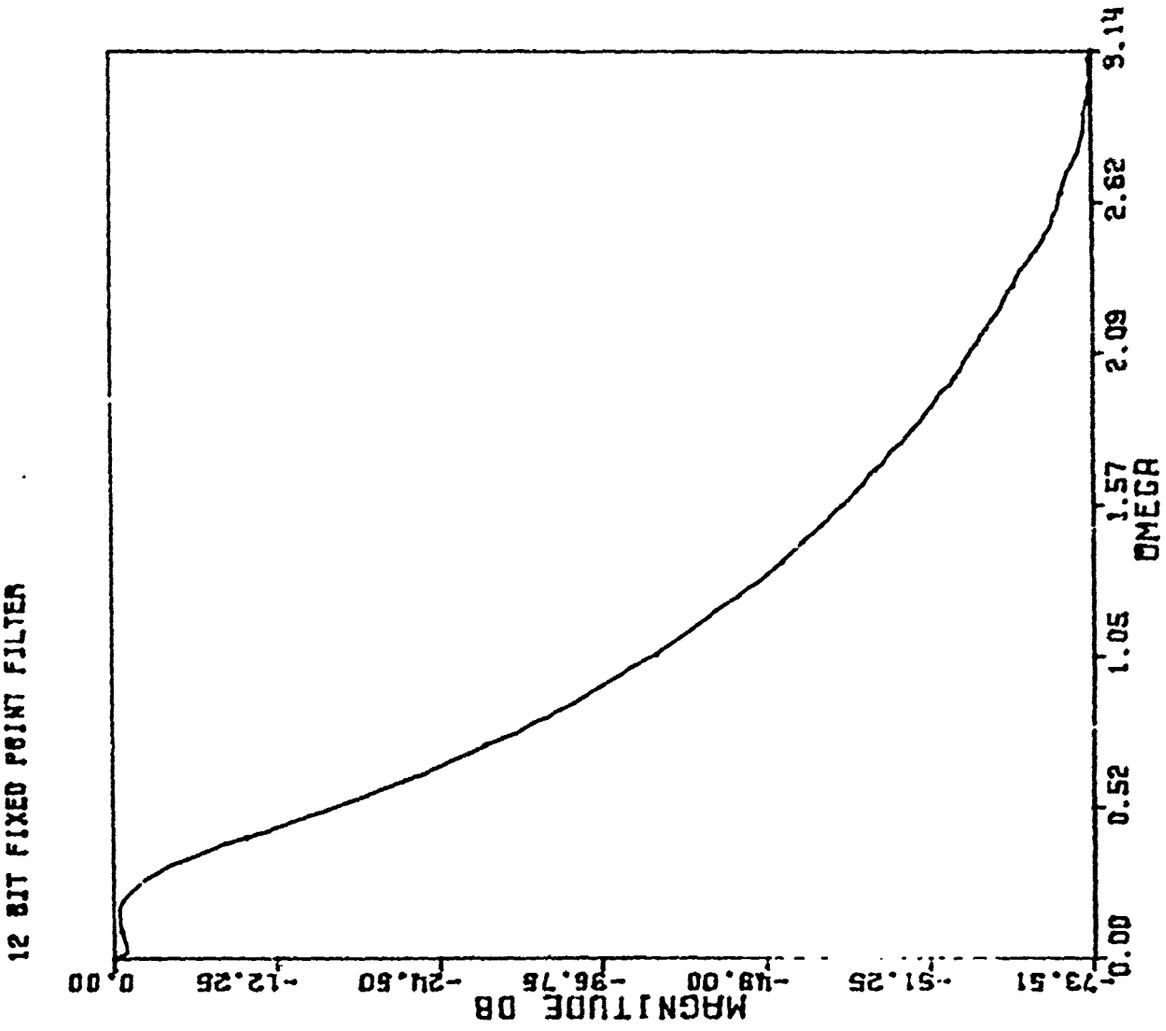

12 BIT FIXED POINT FILTER
MAGNITUDE DB
0.00
-12.25
-24.50
-36.75
-49.00
-61.25
-73.51
0.00
0.52
1.05
1.57
2.09
2.62
3.14
OMEGA

FIGURE 20e

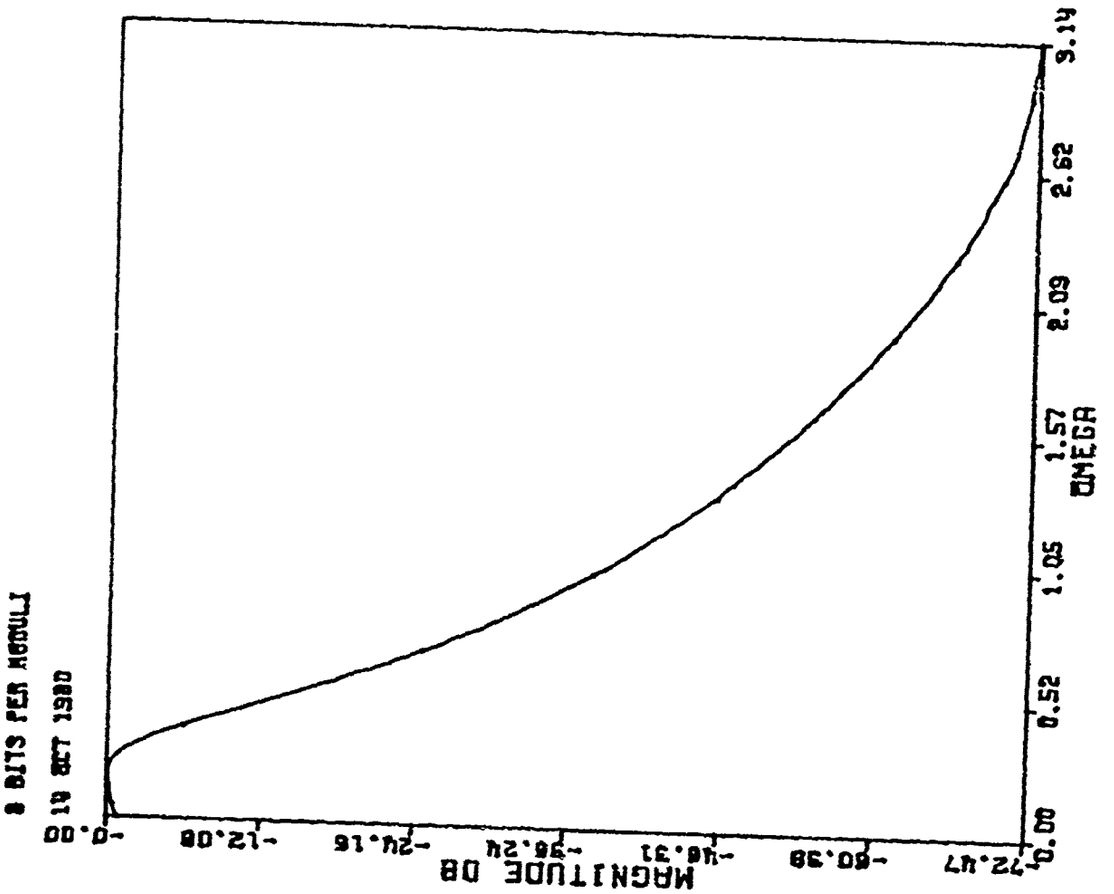

8 BITS PER MODULI
10 OCT 1980
MAGNITUDE DB
-0.00
-12.08
-24.16
-36.24
-48.31
-60.39
-72.47
OMEGA
0.00
0.52
1.05
1.57
2.09
2.62
3.14

FIGURE 20f

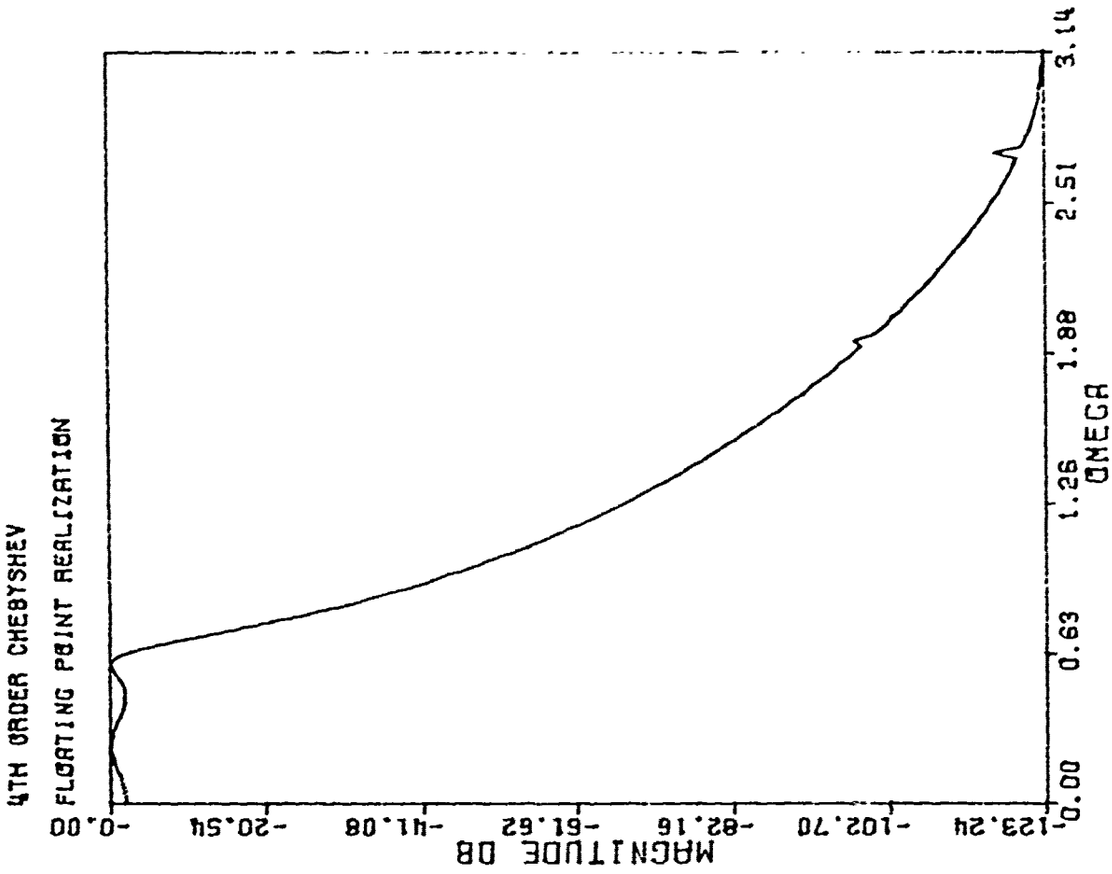

4TH ORDER CHEBYSHEV
FLOATING POINT REALIZATION
MAGNITUDE DB
-0.00
-20.54
-41.08
-61.62
-82.16
-102.70
-123.24
0.00
0.63
1.26
1.88
2.51
3.14
OMEGA

FIGURE 21a

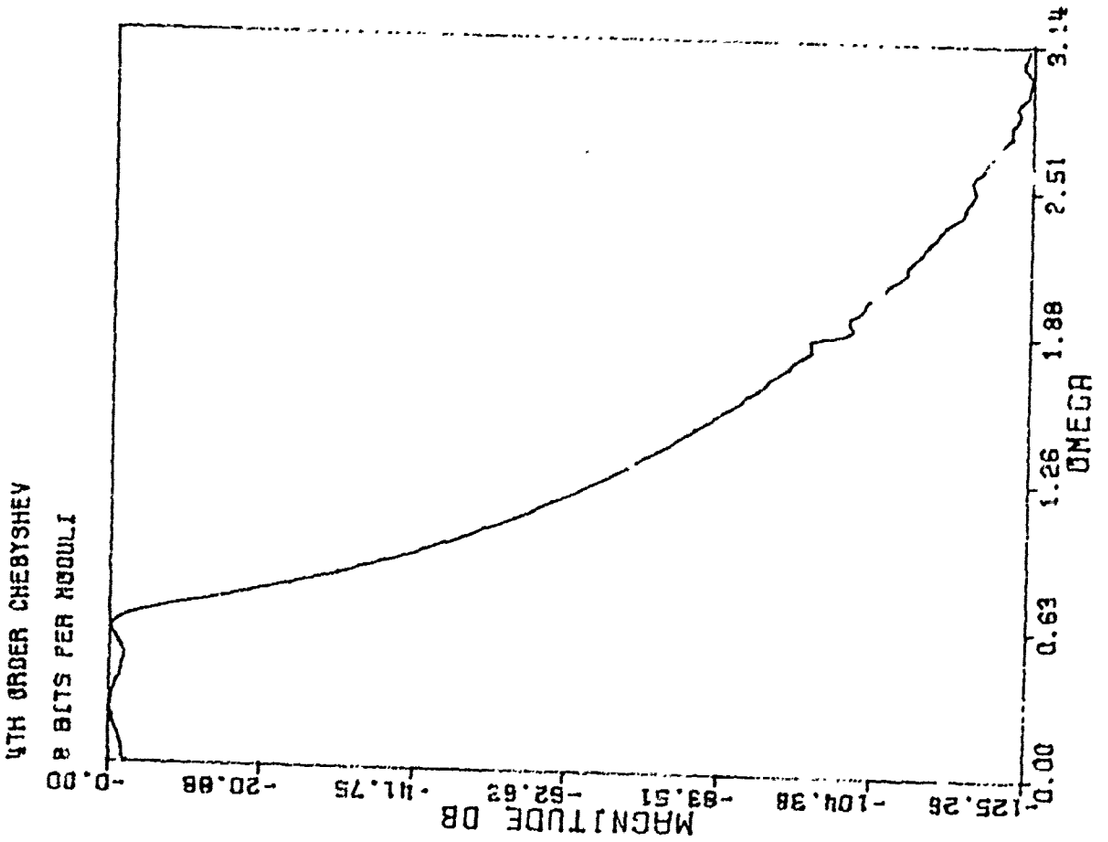

4TH ORDER CHEBYSHEV
8 BITS PER MODULI
MAGNITUDE DB
-0.00
-20.88
-41.75
-62.62
-83.51
-104.38
-125.26
0.00
0.63
1.26
1.88
2.51
3.14
OMEGA

FIGURE 21b

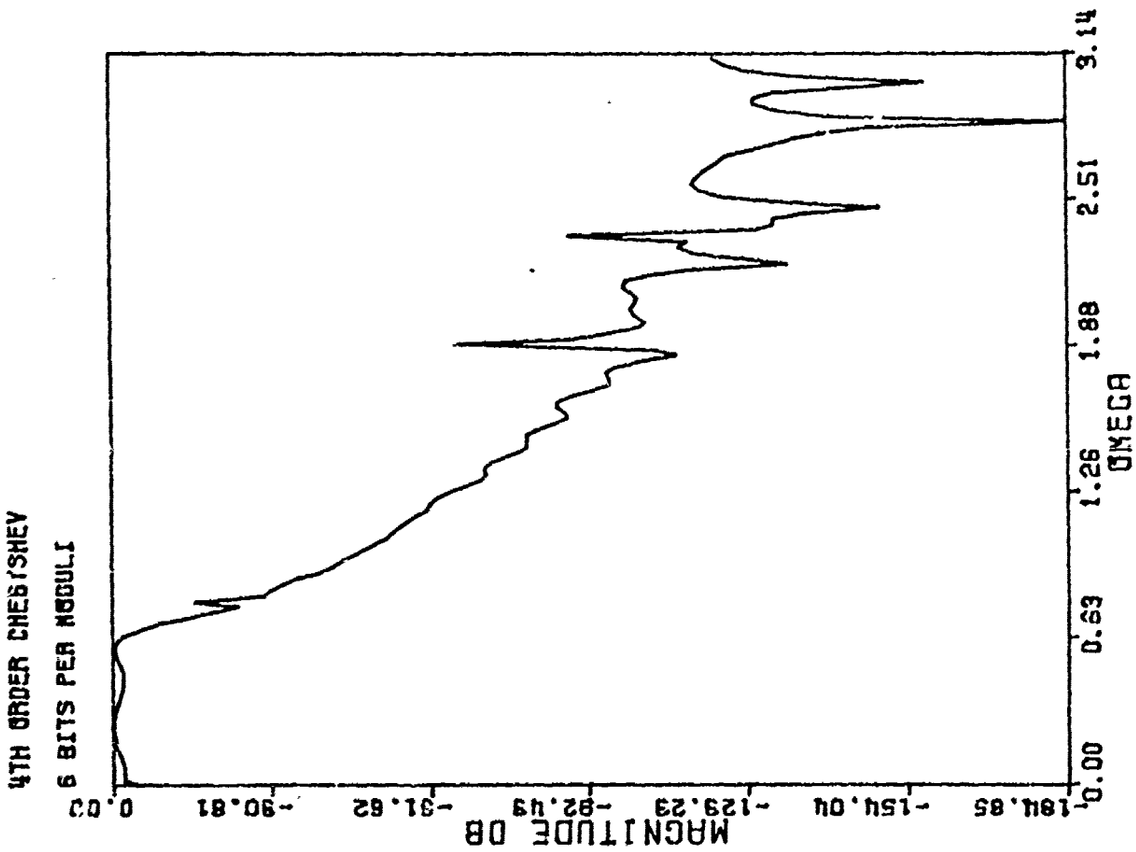

4TH ORDER CHEBYSHEV
6 BITS PER MODULI
MAGNITUDE DB
0.00
-90.81
-31.62
-92.49
-129.23
-154.04
-184.85
0.63
1.26
1.88
2.51
3.14
OMEGA

FIGURE 21c

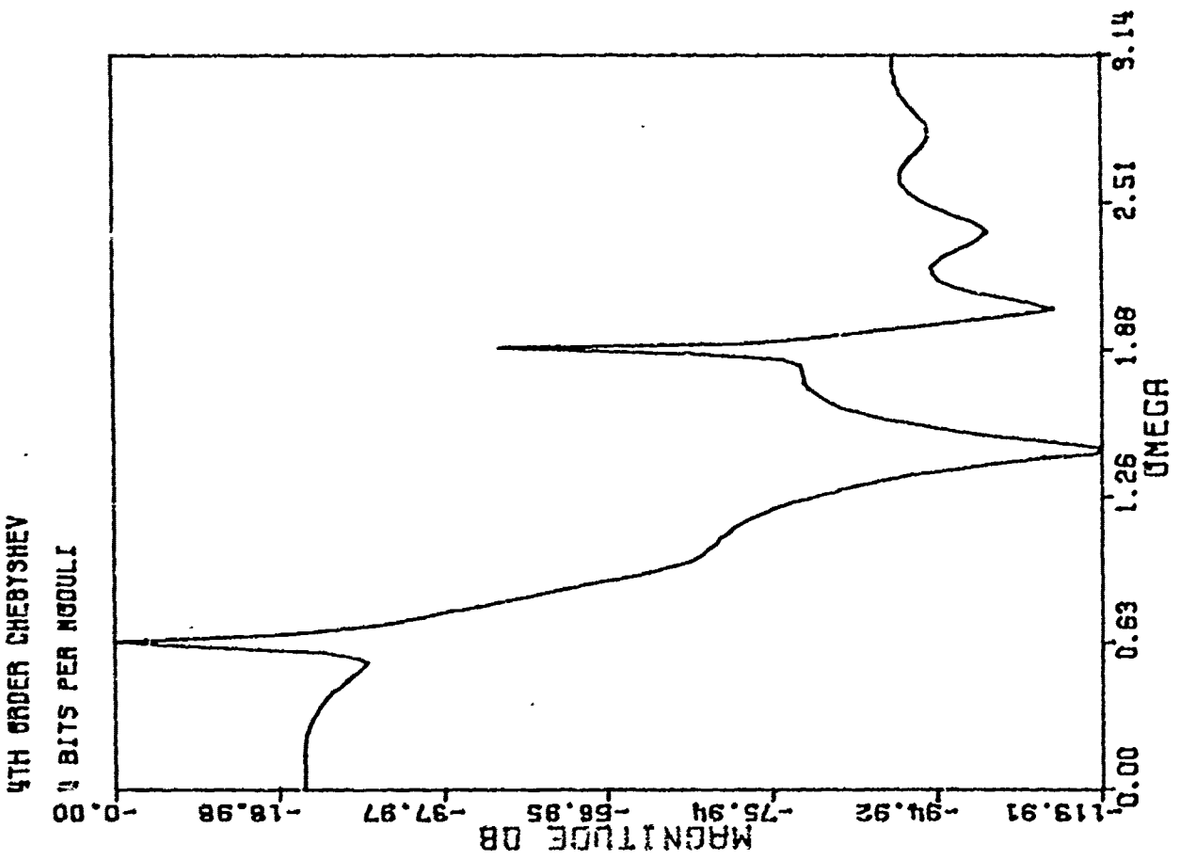

4TH ORDER CHEBYSHEV
4 BITS PER MODULI
MAGNITUDE DB
-0.00
-18.98
-37.97
-56.85
-75.94
-94.92
-113.91
0.00
0.63
1.26
1.88
2.51
3.14
OMEGA

FIGURE 21d

PART III APPLICATIONS

The RNS arithmetic, developed under this AFOSR grant, has been tested in a MS, JKM, and distributed architecture. Several applications were considered. The uniqueness of these applications, are on to themselves, worthy of special treatment. Therefore, have been included in this report in appendices. Appendix A treats the problem of realizing a real-time Kalman filter. The development filter (submitted for publications) represents an original approach to this class of problem. In Appendix B, a linear adaptive noise canceller is presented. Appendix C contains other papers published, or under review, which contain an AFOSR credit line.

PART IV SUMMARY

Under an AFOSR grant, RNS arithmetic, hardware, and architectures have made major strides. Using a three moduli system, practical ultra-high speed RNS units have been developed. The major problem of RNS-to-decimal conversion plus magnitude scaling has been successfully treated. In addition, new filter architectures were derived and analyzed.

The future of the RNS is considerably brighter as a result of this study. In particular, the RNS techniques developed during this grant period, will be further inhanced with the advent of VLSI.

References

1. W.K. Jenkins and B.J. Leon, "The Use Of Residue Number System in the Design of Finite Impulse Response Filters," IEEE C&S, CA5-24, April 1977.

2. G.A. Jullien, "Residue Number Scaling and Other Operations Using ROM Arrays," IEEE Trans. Computers, Vol C-27, No. 4, pp 325-337, April 1978.

3. C.H. Huang and F.J. Taylor, "Memory Compression Scheme For Modular Arithmetic, to be published, IEEE ASSP.

4. M.A. Soderstrand, "A High Speed Low-Cost Recursive Filter Using Residue Arithmetic," Proc. IEEE, Vol. 65, No. 7, July 1977.

5. N.S. Szabo and R.I. Tanaka, Residue Arithmetic and Its Applications To Computer Technology, McGraw-Hill, 1967.

6. W.J. Jenkins, "A Highly Efficient Residue-Combinatorial Architecture for Digital Filters," Proc. of the IEEE (Letter), Vol. 66, No. 6, June 1978. pp 700-702.

7. N.S. Szabo and R.I. Tanaka, Residue Arithmetic and Its Applications to Computer Technology, New York, McGraw-Hill, 1967.

8. W.K. Jenkins, "Techniques for Residue-to-Analog Conversion For Residue-Encoded Digital Filters," IEEE Trans. on Circuits and Systems, Vol. CAS-25, No. 7, July 1978.

9. A. Baraniecka and G.A. Jullien, "On Decoding Techniques for Residue Number System Realizations of Digital Signal Processing Hardware," IEEE Trans. on Circuits and Systems, Vol. CAS-25, No. 11, November 1978.

10. J.M. Pollard, "Implementation of Number-Theoretic Transforms," Electronics Letters, Vol. 12, No. 22, July 1976, pp 378-379.

11. H. Nussbaumer, "Digital Filters Using Read-Only Memories," Electronics Letters, Vol. 11, 1976, pp 294-295.

12. M.A. Soderstrand and E.L. Fields, "Multiplier for Residue-Number-Arithmetic Digital Filters," Electronic Letters, Vol. 13, No. 6, March 17, 1977.

13. G. Bioul, M. Davis, and J.J. Quisquarter," A Computation Scheme for an Adder Modulo (2^n-1)," Digital Process, Georgi Publisher, Switzerland, No. 1, (1975), pp 309-318.

14. W.K. Jenkins, "A New Algorithm for Scaling in Residue Numbers Systems with Applications to Recursive Digital Filtering," Proc. 1977, IEEE Int. Symp. on Circuits and Systems, Phoenix, AZ, pp. 56-59.

15. H. Huang and F.J. Taylor, "High Speed DFT's Using Residue Numbers," IEEE Int. Conf. on Acoustics, Speech & Signal Processing, Denver, Colorado, April, 1980, pp. 238-242.

16. D.P. Maher, "Mathematical Background For Generalized, Partial, and Incomplete Discrete Fourier Transforms, Proceedings of the 1980 ASSP Conference, Denver, Colorado, April 1980.

17. J.A. McClellan and C.M. Rader, Number Theory in Digital Signal Processing, Prentice Hall, 1979, Ch. III.

18. F.J. Taylor, "Large VLSI Module Multipliers," IEEE Int. Symp. on Circuits and Systems, Houston, TX, April 1980.

19. F.J. Taylor, "Large Moduli Multipliers," Proceedings of the ICASSP 80, Denver, Colorado, April 1980.

20. A. Antonou, Digital Filter: Analysis and Design, McGraw-Hill, 1979.

APPENDIX A

THE REALIZATION OF ADAPTIVE KALMAN FILTERS

George Papadourakis

Fred J. Taylor

Department of Electrical and Computer Engineering
University of Cincinnati
Cincinnati, Ohio 45221

Portions of this work was supported under AFOSR Grant
F49620-79-C-0066

Keywords: Adaptive filters, real-time computation, white stochastic process, Pfeifer-Blankenship algorithm.

ABSTRACT:

An adaptive Kalman filter is considered for which the input noise covariance and the output covariance are unknown. A new procedure is presented for the identification of the unknown covariances. The proposed identification algorithm uses the autocorrelation information contained in the innovation error sequence to determine the ratio of the unknown noise covariances and to obtain the optimal Kalman filter gain. The proposed method can be easily implemented through the use of high speed digital autocorrelation algorithms which operate at real time data rates.

1.0 INTRODUCTION

The Kalman-Bucy formulation of the minimal variance filtering problem assumes complete a priori knowledge of the input and output noise covariances, say Q and R. In most practical applications Q and R are either assumed to be unknown or approximated. Several authors have presented schemes for the identification of the unknown covariances with some success (2), (4), (7) and (11) with the use of the innovation sequence in the identification of the unknown covariances, introduced by Mehra.

The identification scheme presented in this paper makes

use of the first delayed value of the autocorrelation function of the innovation sequence to determine the ratio of the unknown covariances. The method presented is applicable for those systems which exhibit a shift-invariant (constant coefficient) property.

A high speed real-time algorithm for the computation of the autocorrelation function has been presented by Pfeifer and Blankenship (12), (13). The speed of the algorithm can be further improved through the use of special codes (ie. in-line, threaded, knotted) which make possible the computation of the autocorrelation function in real time (14).

2.0 PFEIFER/BLANKENSHIP (BP) AUTOCORRELATION ALGORITHM

A discrete autocorrelation function r(k), is given by:

$$r(k) = \sum_{n=1}^{N} f(n)\, f(n+k) \qquad 2.1$$

If r(k) is desired for k on the order of N/2, then using FFTs to compute $DFT^{-1}(x(f)x^*(f)) = r(k)$ is computationally optimal. A total of $N \log_2(N+N)$ complex multiplies are required (a complex multiply equals 4 real multiplies). Direct computation of the above requires N^2 real multiplies.

The, BP version of the autocorrelation algorithm satisfies:

$$r(k)=\sum_{j=0}^{p-1}\sum_{i=1}^{K} f(2jk+i+k)(f(2jk+i)+f(2jk+i+2k)) \quad 2.2$$
$$k=1,2,\ldots,p$$

For N>>P the BP algorithm essentially halves the number of multiplications normally associated with direct computation. The multiplication count for N>>P can also be considerably less than the associated with FFT mechanizations.

The speed of the algorithm can be further improved by eliminating the time consumed to compute data invariant indices (ie: (2jk+k), (2jk+i), (2jk+i+2k)). This can be achieved through the use of so called in-line code. In an in-line code, the code is arranged in a top-down fashion. Here entry is made at the top and without looping, run to completion. The desired in-line code having all data invariant parameters, can be properly computed and properly sequenced through the use of AUTOGEN methodology (14).

Another alternative is possible through the use of threaded code. It replaces a standard program with a series of modules which are threaded together. A thread is a precomputed data array in which all prerequist information is found. The array is serially scanned at run time and there-

fore removes the overhead associated with the original computation of the parameters. Compared to the in-line code the threaded code is twice as fast and occupies less memory (14).

Another option is available and is known as a knotted code. Knots will be tied in the thread indicating that the program will move down the thread, "run-around" inside a knot for a while then continue down the thread to the next knot. The knot represents a subprogram in which there may exist elementary looping. Using the knotted code the memory requirements reduced considerably (14). It has been shown in reference 14 that a 128-point time series can be autocorrelated, (up to a delay of 11 samples) using a PDP 11 minicomputer the following time intervals:

	PDP 11/55	PDP 11/70*	PDP 11/60*	Program Size #
Conventional	9.69	9.72	17.35	80
In-line	5.25	9.42	13.03	7552
Knotted	6.70	7.95	11.35	768

* Cache Machines
\# Words

Realizing that the computation of the autocorrelation function can be performed at real time speeds, we can proceed to the analysis of the problem.

3.0 STATEMENT OF THE PROBLEM

A discrete stochastic dynamic system can be represented as:

$$\underline{x}_{i+1} = F\,\underline{x}_i + \underline{u}_i \qquad 3.1.a$$

$$\underline{z}_i = H\,\underline{x}_i + \underline{v}_i \qquad 3.1.b$$

where

$i=0,1,2....$

$\underline{x}_i$ = nx1 state vector

F = nxn state transition matrix (constant)

$\underline{u}_i$ = nx1 vector of Gaussian input white noise

$\underline{z}_i$ = rx1 output vector

H = rxn output matrix (constant)

$\underline{v}_i$ = rx1 vector of Gaussian measurement errors (white)

It is assumed that:

$$E(\underline{u}_i) = 0 \qquad 3.2.a \qquad E(\underline{u}_i \underline{u}_j^T) = Q\delta_{ij} \qquad 3.2.b$$

$$E(\underline{v}_i) = 0 \qquad 3.3.a \qquad E(\underline{v}_i \underline{v}_j^T) = R\delta_{ij} \qquad 3.3.b$$

We consider the system to be a lowpass or bandpass filter and completly controllable and observable.

Given an apriori estimate of the inital state $\hat{x}_0$ and the state covariance $P_0(-)$ and given the apriori statisti-

cal information of the input noise covariance QI and output noise covariance RI an estimate of the state of the system defined by (3.1) is obtained sequentially for k=1,2,3.. with the standard Kalman filter:

1. State Estimation Extrapolation:

$$\underline{\hat{x}}_K(-) = F\,\underline{\hat{x}}_{K-1}(+) \qquad 3.4$$

2. Error Covariance Extrapolation:

$$PE_K(-) = FPE_{K-1}(+)F^T + QI_{K-1} \qquad 3.5$$

3. State Estimation Update:

$$\underline{\hat{x}}_K(+) = \underline{\hat{x}}_K(-) + K_K(\underline{z}_K - H\underline{\hat{x}}_K(-)) \qquad 3.6$$

4. Error Covariance Update:

$$PE_K(+) = (I - K_K H)\,PE_K(-) \qquad 3.7$$

5. Kalman Gain Matrix:

$$K_K = PE_K(-)H^T(HPE_K(-)H^T + RI_K)^{-1} \qquad 3.8$$

By definition:

$$PT_K(-) = E(\underline{\tilde{x}}_K(-)\underline{\tilde{x}}_K(-)^T) \qquad 3.9$$

where

$$\underline{\tilde{x}}_K(-) = \underline{\hat{x}}_K(-) - \underline{x}_K \qquad 3.10$$

4.3 SCALAR-INPUT SCALAR-OUTPUT SYSTEMS

Consider the single-input, single-output model (ie: r=1 and u(k) a scalar time series) where Q1 is a diagonal matrix with only one non-zero element and R1 is a scalar matrix.

The innovation error of a Kalman filter is defined as

$$\underline{\nu}_K = \underline{Z}_K - H\underline{\hat{x}}_K(-) \qquad 4.1$$

By taking the autocorrelation function of the innovation error, the first two terms can be derived to be:

$$E(\underline{\nu}_K\underline{\nu}_K^T) = HP(-)H^T + R \qquad 4.2$$

$$E(\underline{\nu}_K\underline{\nu}_{K-1}^T) = HF\,(P(-)H^T - K(HP(-)H^T + R)) \qquad 4.3$$

Note that when optimally configured, the Kalman gain K is given by:

$$K = P(-)H^T\,(HP(-)H^T + R)^{-1} \qquad 4.4$$

and (4.3) vanishes. Since we are considering scalar-input scalar-output system the autocorrelation of the innovation error is also a scalar.

The performance of the Kalman filter depends on the choice of R1 and Q1 which define the optimal steady state Kalman gain. It can be shown through direct substitution

that the optimal steady state gain can be defined in terms of the ratio of the true statistics $(R_I/Q_I)=(R/Q)$. In this case $P_E(-)$ is directly proportional to $P_T(-)$ by a factor λ where:

$$\lambda = (R/R_I) = (Q/Q_I) \qquad 4.5$$

By expanding the definition of $P_T(-)$ in (3.9) we get:

$$P_T(-) = FP_T(-)F^T + Q \qquad 4.6$$

$$P_T(+) = (I-KH)P_T(-)(I-KH)^T + KRK^T \qquad 4.7$$

The following observations can be made regarding equations (4.6), (4.7) and (3.5).

1. $P_T(-)$ depends explicitly upon the output apriori statistics of the system model (viz: Kalman gain), and the true noise covariances R and Q.

2. $P_E(-)$ depends upon apriori statistics only.

Consider the case in which $R_I=R$ and $Q_I>Q$. It can be seen clearly from equations (3.9), (4.6) and (4.7) that $P_E(-)>P_T(-)$. Consider also the alternative case in which $Q_I=Q$ and $Q>Q_I$, it is clear that $P_T(-)>P_E(-)$. These cases are illustrated in Figures 1(a), 1(b) and 1(c). Here each iteration represents the Kalman filtering of a 25° point

time series with R1=0 and Q1 changing. The steady state covariances are plotted for each iteration with the ratio of R1/Q1 distributed logarithmicly from .01 to 10.

Observe that:

1. The crossover point of the curves is the optimal case in which we obtain the optimal Kalman gain.

2. The theoretical covariance is increasing after the crossover point. It can be explained by the Kalman filter gain which is decreasing in each iteration. Therefore, the innovation error is magnified.

3. It should be mentioned that in a real system PT(-) can not be explicitly determined.

At each iteration in the above simulation the autocorrelation function of the innovation process was calculated directly. Observe that the second term of the autocorrelation function (4.3) starts with negative values and increases. When (R1/Q1)=(R/Q), we have the optimal case in which $\Gamma(\underline{\varepsilon}_K \underline{\varepsilon}_{K-1}^T) = 0$ as previously stated. This is illustrated by Figures 2(a), 2(b), 2(c). If (R/Q)>(R1/Q1) the Kalman gain is high, PT(-) is also high and

$$\Sigma(\Sigma\Phi T(-)\Sigma^T+R) > \Phi T(-)\Sigma^T \qquad (4.4)$$

which makes $\Sigma(\underline{y}_K\underline{y}_{K-1}^T)$ negative. In the other case where $(R1/Q1)>(R/Q)$, the results are opposite.

5.3 ALGORITHM AND PROCEDURE

The algorithm is based upon the autocorrelation function of the innovation process.

In the first iteration, the first term of the autocorrelation $R(0)$ gives a good estimate of R (4.2) especially when $(R/Q)>1$. Each iteration represents the Kalman filtering of a 256 point time series with a constant R1 and variable Q1. Using as R1 the R(0) of the first iteration, forty values of Q1 are chosen so $10>(R1/Q1)>.01$. After each iteration the first delayed valued of the autocorrelation function is calculated as a function of the ratio R1/Q1. When the process is over, a curve fitting of these points and the order of the approximation is determined. Using the bisection algorithm, the value of λ at which $R(1)\equiv 0$ can be determined. The iteration number at which $R(1)\equiv 0$ can be determined and then R can be calculated from (4.3) since R(0) and $\Phi P\Sigma(-)\Psi^T$ have been stored as fuctions the iterations index. Knowing the ratio and R, Q can be easily calculated.

Knowledge of the operating curve can be used to define

an adaptive policy. An adaptive algorithm can be derived which, depending upon the current location on the operating curve, can converge to the optimal case and give the optimal steady-state Kalman gain.

The above algorithm can be easily implemented in a microprocessor due to it's simplicity. Real time throughput can be achived using the DP algorithm to calculate the first two terms of the autocorrelation function.

6.0 NUMERICAL EXAMPLE

A fourth order elliptic lowpass filter was simulated on a PDP 11/60 digital microcomputer. The actual values of F and H are:

$$F = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -0.5 & 0.6 & -1.5 & 1.2 \end{bmatrix}$$

$$H = (0.05 \quad 0.222 \quad 0.711 \quad 0.254)$$

The program was developed such that the true noise covariances can be inputed and by using Gaussian subroutine the noises were generated. The program yields the ratio of the unknown covariances L and 1. Figures 2(a), 2(b) and

2(c) illustrate the results. The dots represent the actual values of R(1) at each iteration. The coordinates of them are also presented. The curve fitting of these points is represented by the continous line. The unknown noise statistics of the system, IN corresponding for Q and OUT corresponding for R are the values in the parenthesis and their estimation is shown below them. Results were good with minimal errors.

7.0 SUMMARY AND CONCLUSION

In summary, an algorithm has been developed in identifing unknown noise covariances in a lowpass filter with the use of a Kalman filter. The algorithm makes use of the first and second terms of the autocorrelation function of the innovation process. By determining first, the ratio of the unknown covariances their actual values can be calculated. Implementation of the algorithm is possible and real time throughput can be achived with the use of fast autocorrelation algorithms. A numerical example illustrates the algorithm.

There are two minor problems with the algorithm.

1. When the output to input noise ratio is very small (less than .2), relative precise calculation of the actual noise statistics is difficult, although

their ratio can be determined precisly. In this case we have complete knowledge of the optimal Kalmán gain.

2. When the output to input noise ratio is very high (greater than 1%.), relative calculation of the ratio is not very accurate because of the slope of the curve. However the estimation of the output noise covariance is very accurate.

References

1. R. E. Kalman, "New methods and results in linear prediction and filtering theory," in Proc. Sym. on Eng. Appl. of Random Function Theory and Probability. New York: Wiley, 1961, pp. 27[illegible].

2. R. K. Mehra, "On the identification of variances and adaptive Kalman filtering," IEEE Trans. Automat. Contr., vol. AC-15, April 1970, pp. 175-184.

3. A. Gelb, Applied Optimal Estimation, Massachusetts: M.I.T Press, 1974.

4. G. C. Carew and P. E. Belanger, "Identification of optimal filter steady-state gain for systems with unknown noise covariance," IEEE Trans. Automat. Contr., vol. AC-18, December 1973, pp. 582-587.

5. A. E. Bryson and Y.C. Ho, Applied Optimal Control, Washington D.C.: Hemisphere Publishing Corporation, 1975.

dynamic systems with applications to Kalman filtering," IEEE Trans. Automat. Contr., vol. AC-15, February 1971, pp. 12-21.

7. D. K. Mehra, "Approaches to adaptive filtering," IEEE Trans. Automat. Contr., vol. AC-17, October 1972, pp. 693-698.

8. D. L. Alspach "A parallel filtering algorithm for linear systems with unknown time varying noise statistics," IEEE Trans. Automat. Contr., vol. AC-19, October 1974, pp. 552-556.

9. K. A. Myers and B. D. Tapley, "Adaptive sequential-estimation with unknown noise statistics," IEEE Trans. Automat. Contr., vol. AC-21, August 1976, pp. 520-523.

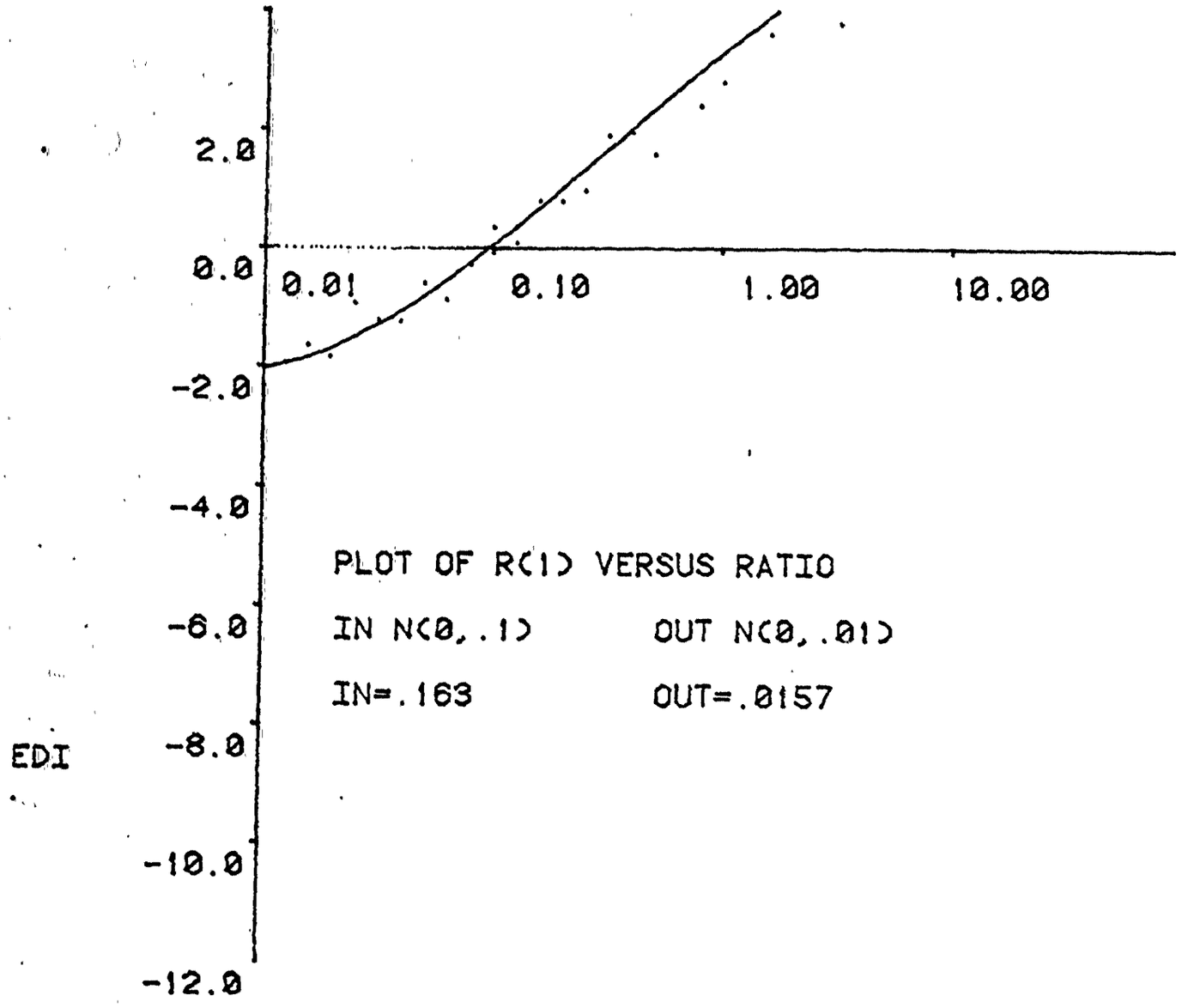
10. B. Carnahan, H. A. Luther and J. O. Wilkes, Applied Numerical Methods, New York: Wiley, 1969.

11. G. L. Smith, "Sequence estimation of observation error variances in a trajectory estimation problem," AIAA J., vol. 5, pp. 1964-1970, November 1967.

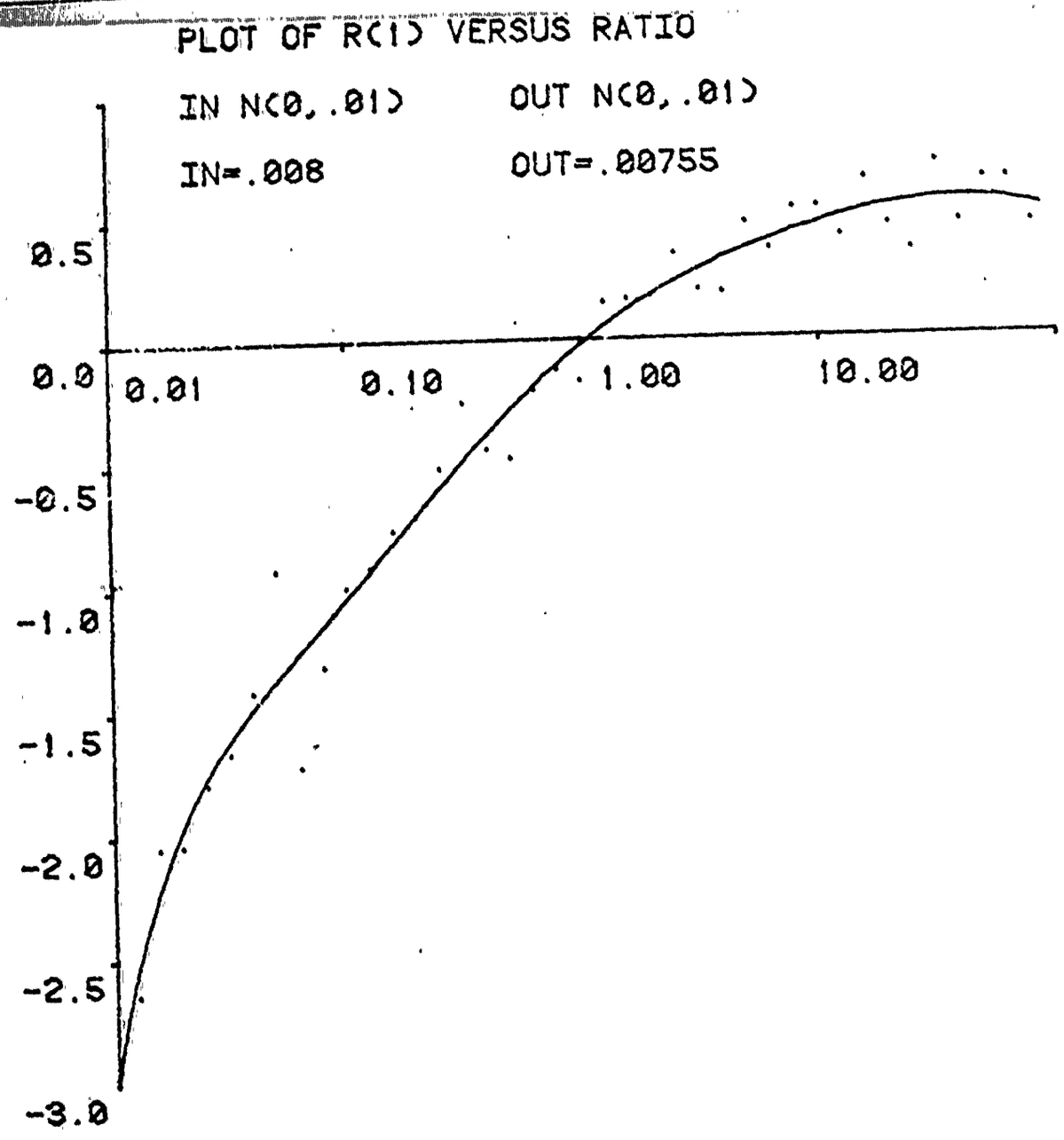
[6]. L. L. Pfeifer, "Multiplication reduction in short term autocorrelation," IEEE Trans. Audio and Electroacoustics, vol. AU-21, pp. 351-[illegible], October 1973.

[7]. W. A. Blankinship, "Note on computing autocorrelations," IEEE Trans. Acoustics, Speech, Signal Processing, vol. Assp-23, pp.76-77, February 1975.

[8]. A. M. Bell[illegible] and L. T. Morris, Minicomputer Systems, New Jersey: Prentice-Hall 1[illegible].

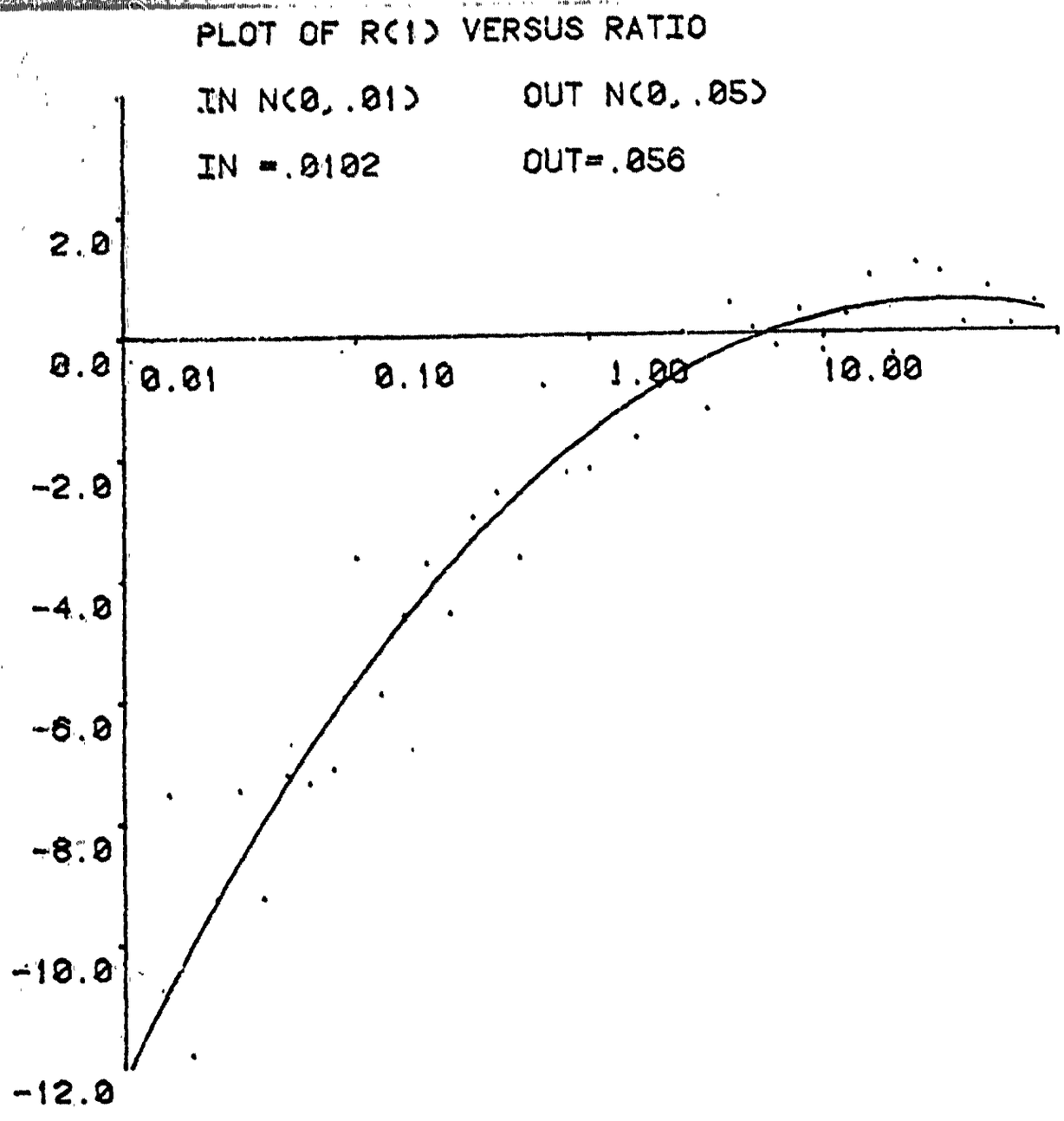



```
RATIO= 0.096
N=   0.01      R2=-2.316
N=   0.01      R2=-1.940
N=   0.02      R2=-1.637
N=   0.02      R2=-1.349
N=   0.03      R2=-.9102
N=   0.03      R2=-1.202
N=   0.04      R2=-1.249
N=   0.05      R2=-.5873
N=   0.06      R2=-.8654
N=   0.08      R2=-.2732
N=   0.10      R2=0.3724
N=   0.13      R2=0.9573E-01
N=   0.16      R2=0.8117
N=   0.20      R2=0.7772
N=   0.25      R2=0.9887
N=   0.32      R2= 1.922
N=   0.40      R2= 2.000
N=   0.50      R2= 1.982
N=   0.63      R2= 4.001
N=   0.79      R2= 2.407
N=   1.00      R2= 2.803
N=   1.26      R2= 4.057
N=   1.58      R2= 3.606
N=   2.00      R2= 4.564
N=   2.51      R2= 4.808
N=   3.16      R2= 3.779
N=   3.98      R2= 6.823
N=   5.01      R2= 4.521
N=   6.31      R2= 5.076
N=   7.94      R2= 5.845
N=  10.00      R2= 5.231
N=  12.59      R2= 6.201
N=  15.85      R2= 5.981
N=  19.95      R2= 4.757
N=  25.12      R2= 8.145
N=  31.62      R2= 5.420
N=  39.81      R2= 5.562
N=  50.12      R2= 6.399
N=  63.10      R2= 5.742
N=  79.43      R2= 6.493
```



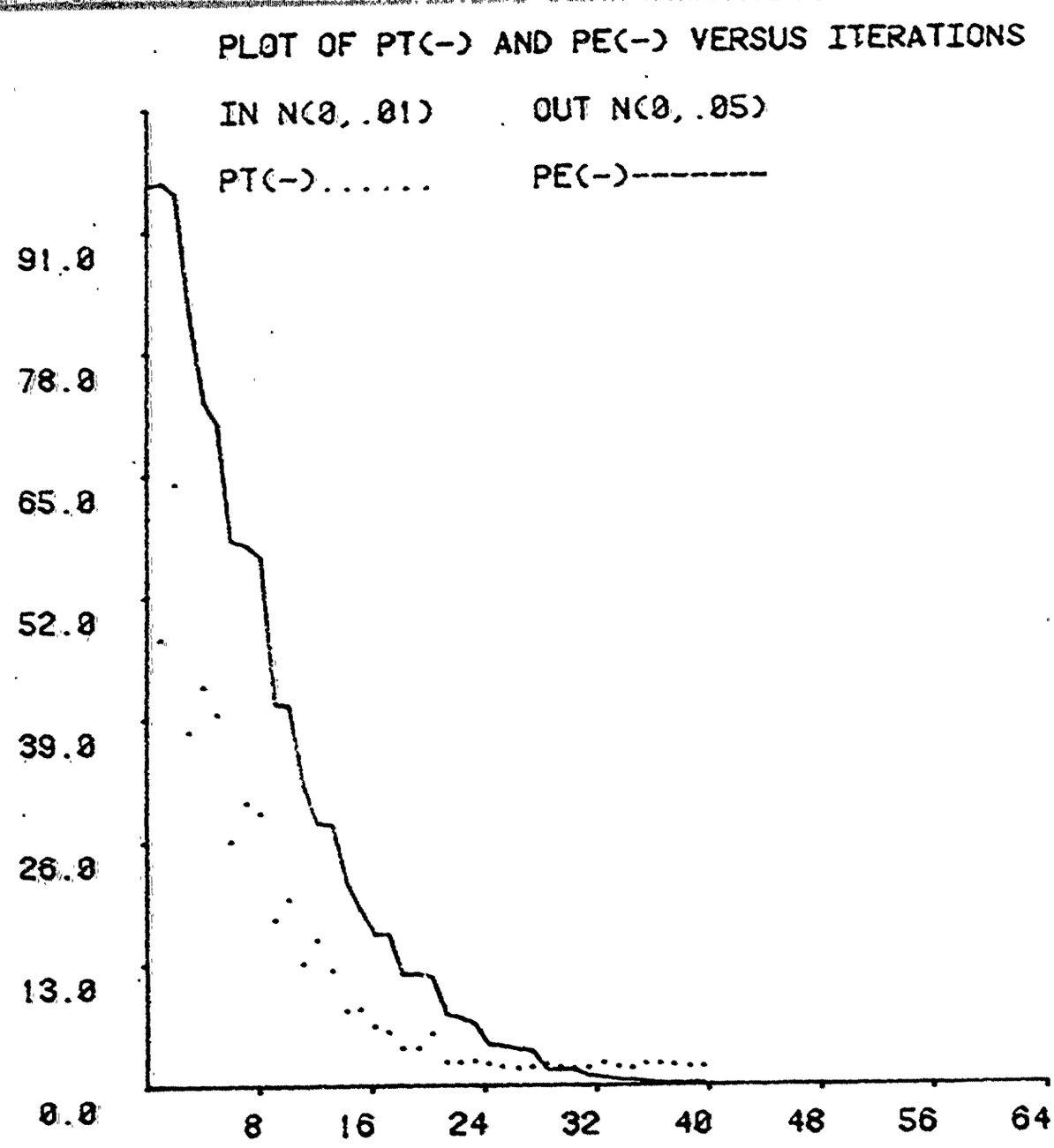
RATIO= 1.040

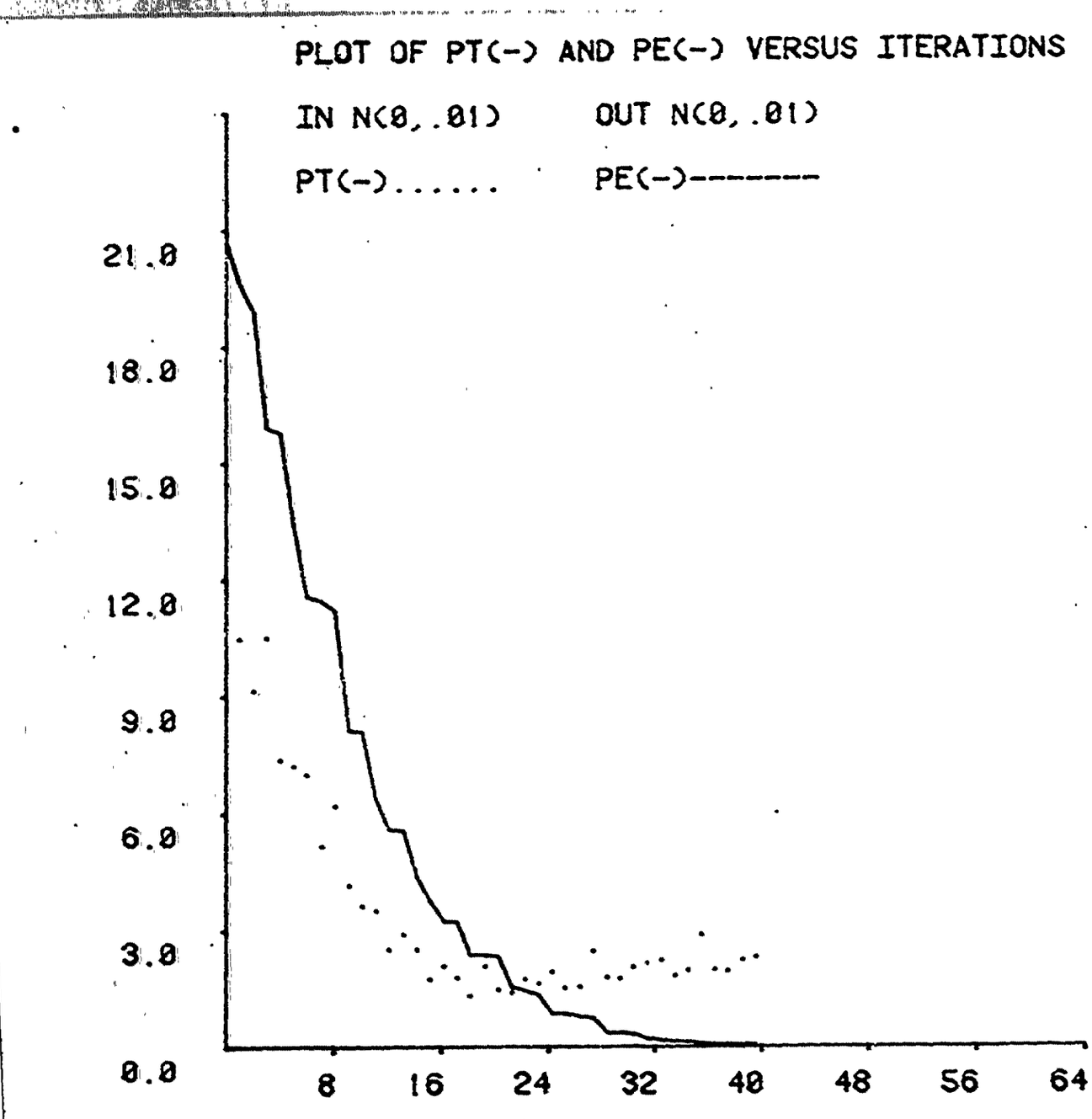
N= 0.01	R2=-2.972
N= 0.01	R2=-2.642
N= 0.02	R2=-3.047
N= 0.02	R2=-2.043
N= 0.03	R2=-1.792
N= 0.03	R2=-1.664
N= 0.04	R2=-1.413
N= 0.05	R2=-.9244
N= 0.06	R2=-1.726
N= 0.08	R2=-1.317
N= 0.10	R2=-.9938
N= 0.13	R2=-.9171
N= 0.16	R2=-.7640
N= 0.20	R2=-.7084
N= 0.25	R2=-.5130
N= 0.32	R2=-.2469
N= 0.40	R2=-.4422
N= 0.50	R2=-.4792
N= 0.63	R2=-.1993
N= 0.79	R2=-.1179
N= 1.00	R2=-.1637
N= 1.26	R2=0.1523
N= 1.58	R2=0.1655
N= 2.00	R2=0.1773
N= 2.51	R2=0.3498
N= 3.16	R2=0.1998
N= 3.98	R2=0.1849
N= 5.01	R2=0.4742
N= 6.31	R2=0.3621
N= 7.94	R2=0.5282
N= 10.00	R2=0.5322
N= 12.59	R2=0.4182
N= 15.85	R2=0.6461
N= 19.95	R2=0.4617
N= 25.12	R2=0.3562
N= 31.62	R2=0.7106
N= 39.81	R2=0.4627
N= 50.12	R2=0.6404
N= 63.10	R2=0.6421
N= 79.43	R2=0.4599

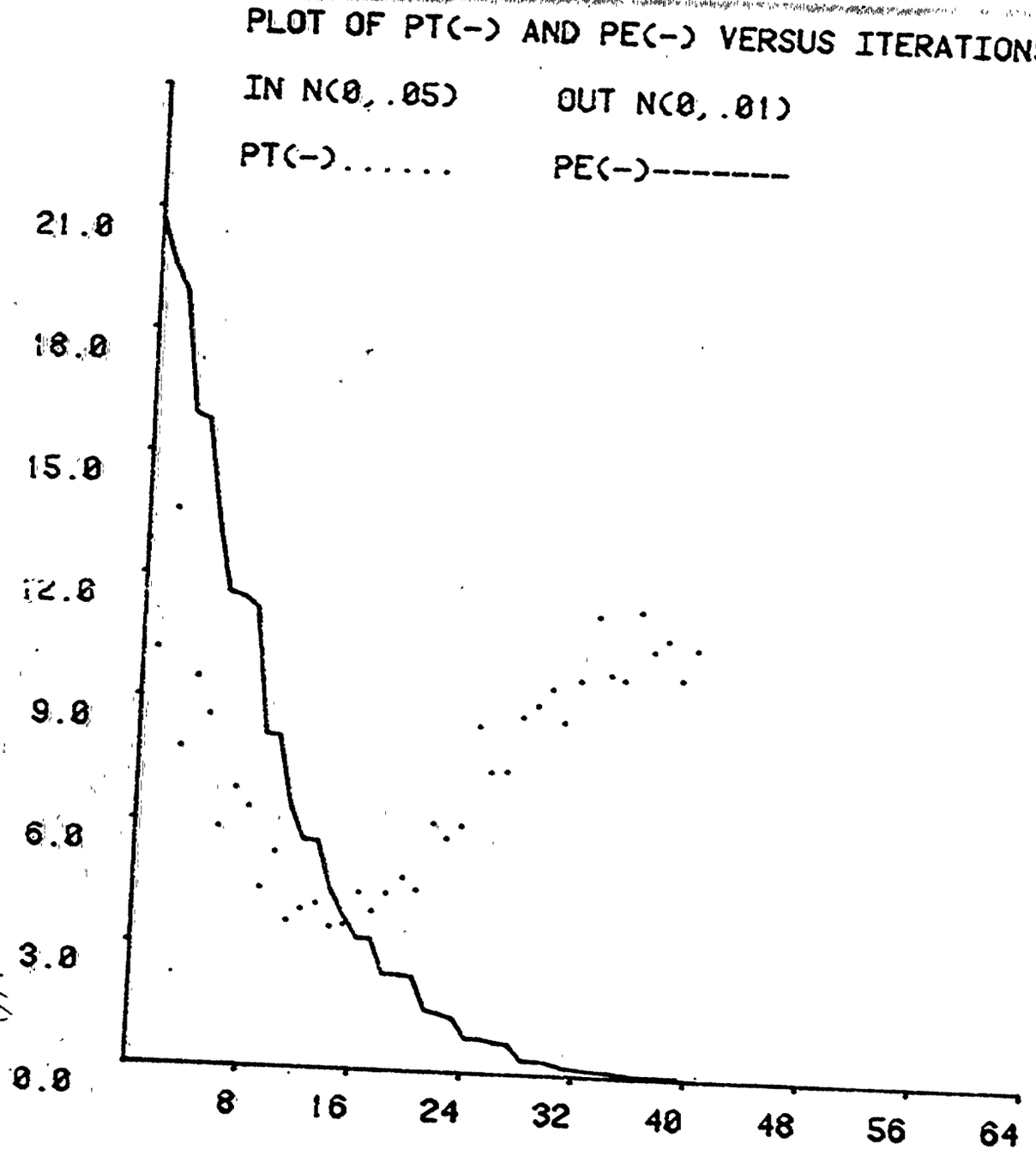


RATIO= 5.536

N= 0.01	R2=-11.93
N= 0.01	R2=-13.32
N= 0.02	R2=-7.513
N= 0.02	R2=-11.82
N= 0.03	R2=-9.247
N= 0.03	R2=-7.468
N= 0.04	R2=-9.228
N= 0.05	R2=-7.205
N= 0.06	R2=-7.370
N= 0.08	R2=-7.113
N= 0.10	R2=-3.608
N= 0.13	R2=-5.879
N= 0.16	R2=-4.596
N= 0.20	R2=-3.712
N= 0.25	R2=-4.572
N= 0.32	R2=-2.954
N= 0.40	R2=-2.552
N= 0.50	R2=-3.637
N= 0.63	R2=-.8104
N= 0.79	R2=-2.269
N= 1.00	R2=-2.181
N= 1.26	R2=-.7928
N= 1.58	R2=-1.685
N= 2.00	R2=-.8317
N= 2.51	R2=0.2573E-01
N= 3.16	R2=-1.228
N= 3.98	R2=0.5218
N= 5.01	R2=0.8414E-01
N= 6.31	R2=-.2237
N= 7.94	R2=0.3927
N= 10.00	R2=-.2926
N= 12.59	R2=0.3129
N= 15.85	R2=0.9490
N= 19.95	R2=-.3697
N= 25.12	R2= 1.145
N= 31.62	R2=0.9952
N= 39.81	R2=0.1378
N= 50.12	R2=0.7343
N= 63.10	R2=0.1110
N= 79.43	R2=0.5000


PLOT OF PT(-) AND PE(-) VERSUS ITERATIONS
IN N(0,.01)
OUT N(0,.05)
PT(-).....
PE(-)-------
91.0
78.0
65.0
52.0
39.0
26.0
13.0
0.0
8
16
24
32
40
48
56
64


PLOT OF PT(-) AND PE(-) VERSUS ITERATIONS
IN N(0,.01)
OUT N(0,.01)
PT(-).....
PE(-)-------
21.0
18.0
15.0
12.0
9.0
6.0
3.0
0.0
8
16
24
32
40
48
56
64


PLOT OF PT(-) AND PE(-) VERSUS ITERATIONS
IN N(0,.05)
OUT N(0,.01)
PT(-).....
PE(-)-------
21.0
18.0
15.0
12.0
9.0
6.0
3.0
0.0
8
16
24
32
40
48
56
64

APPENDIX B

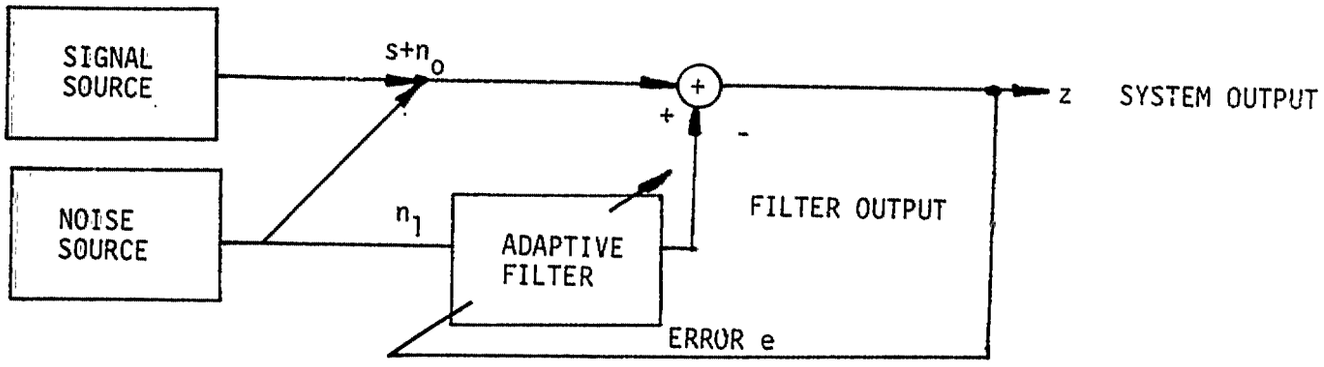
ADAPTIVE NOISE CANCELLING

Adaptive Noise cancelling is a method of estimating signals corrupted by additive noise or interference. This method uses a 'Primary' input containing the corrupted signal and a 'Reference' input containing Noise which is similar to the primary noise. . The reference input is adaptively filtered and subtracted from the primary input to obtain the signal estimate. .

Adaptive filtering before subtraction allows the treatment of inputs that are deterministic or stochastic, stationary or time variable. When the reference input is free of signal and when certain other conditions are met, the noise in the primary input can be eliminated without distortion. . It is further shown that in treating periodic interference, the adaptive noise canceller acts as a notch filter with narrow bandwidth and the capability of tracking the exact frequency of interference. .

Noise cancelling is a variation of optimal filtering that is highly advantageous in many applications. It makes use of a reference input derived from one or more sensors located at points in the noise field where the signal is very weak or undetectable. . This input is filtered and subtracted from the primary input containing both the signal andthe noise. . As a result, the primary noise attenuated or elliminated by cancellation. .

If done improperly, subtracting noise from a received signal, would result in an increase in the output noise power. . However when the filtering and subtraction are controlled by an appropriate adaptive process, noise reduction if not complete noise elimination, can be acomplished. . Adaptive filtering may not be applicble in all of the filtering situations. . This adaptive filtering would not be possible when, for example, the reference noise input is unavailable. . In circumstances where the adaptive noise cancelling is applicable, the levels of noise reduction are often attainable, that would be difficult to achieve in direct filtering. .



ADAPTIVE NOISE CANCELLER

FIGURE 1: ADAPTIVE NOISE CANCELLING CONCEPT

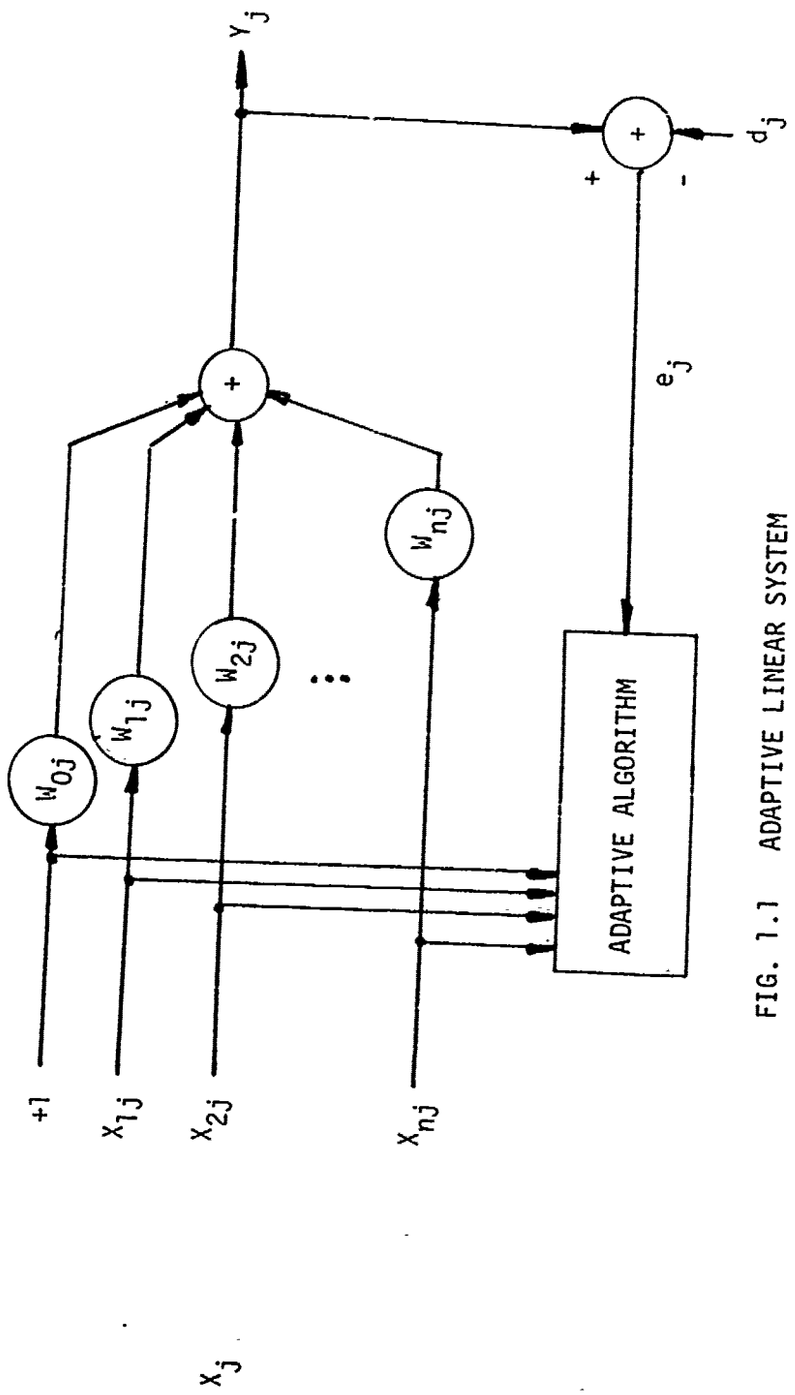


FIG. 1.1 ADAPTIVE LINEAR SYSTEM

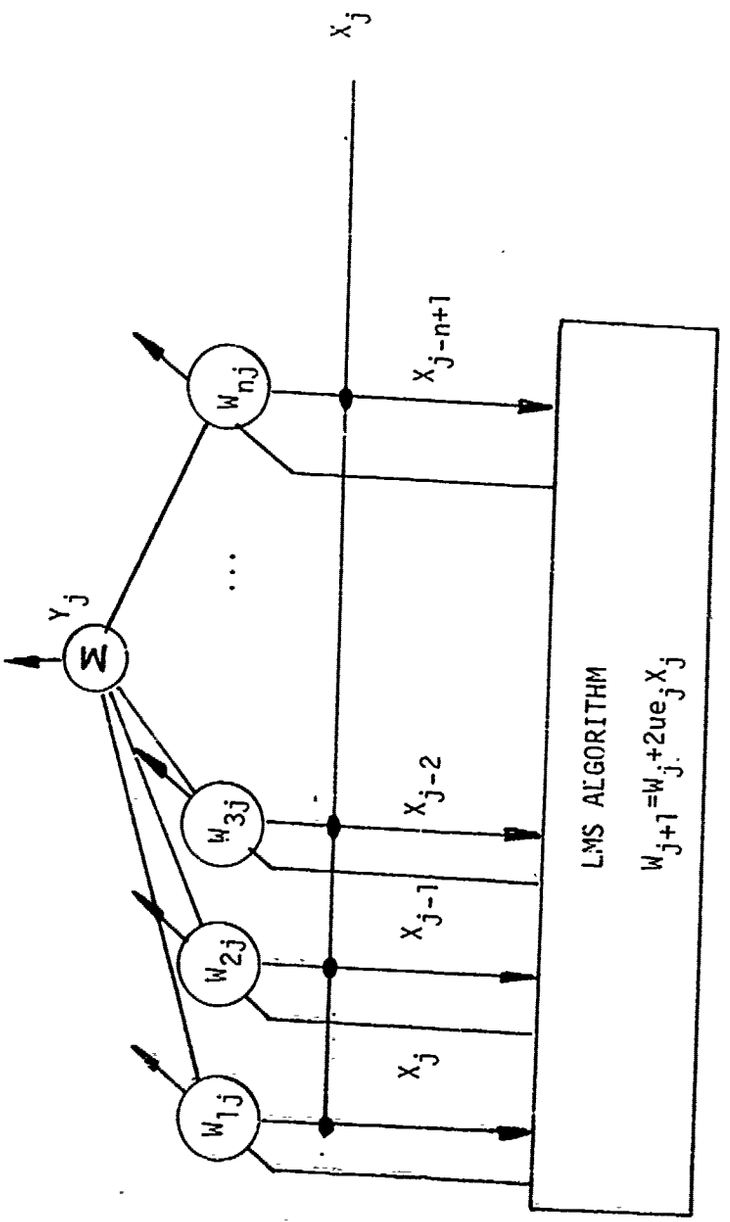

X_j
Y_j
W_{nj}
W_{3j}
W_{2j}
W_{1j}
X_{j-n+1}
X_{j-2}
X_{j-1}
X_j
LMS ALGORITHM
$W_{j+1}=W_j+2ue_jX_j$

FIGURE 1.3

In the noise cancelling systems, the system output Z=S+n0-Y should be a best fit in the least squares sense to the signal S. This is acomplished by feeding the system output back to the adaptive filter and adjusting the filter through an LMS adaptive algorithm to minimise the total system output power. Thus the system output serrves as the error signal for the adaptive process.

THE LMS ADAPTIVE FILTER:

The LMS adaptive filter is the basic element of the adaptive noise cancelling systems. The principal componant of most adaptive systems is the adaptive linear combiner shown in fig1.1. The combiner weighs and sums a set of input signals to form the output signal. The input signal vector X_j is defined as:

$$X_j = \begin{pmatrix} X_{0j} \\ X_{1j} \\ \vdots \\ X_{nj} \end{pmatrix}$$

The input signal componants are assumed to appear simultaneously on all input lines a discrete times indexed by the subscript 'j'. The componant X_0 is a constant normally set to 1 unless biasing is desired. The weighting coefficients W0,W1,W2...Wn are adjustible as shown in fig 1.1. The weight vector is:

$$W = \begin{pmatrix} W_0 \\ W_1 \\ \vdots \\ W_n \end{pmatrix}$$

Where W0 is the bias weight. The output Y is the innerproduct of W and X_j.

That is: $Y(j)=X_j^T W=W^T X_j$

Error e(j) is defined as the difference between the desired response d(j) and the actual response Y(j). In the noise cancelling systems, d(j) is the primary input itself.

$$e(j)=d(j)-X_j^T W=d(j)-W^T X_j$$

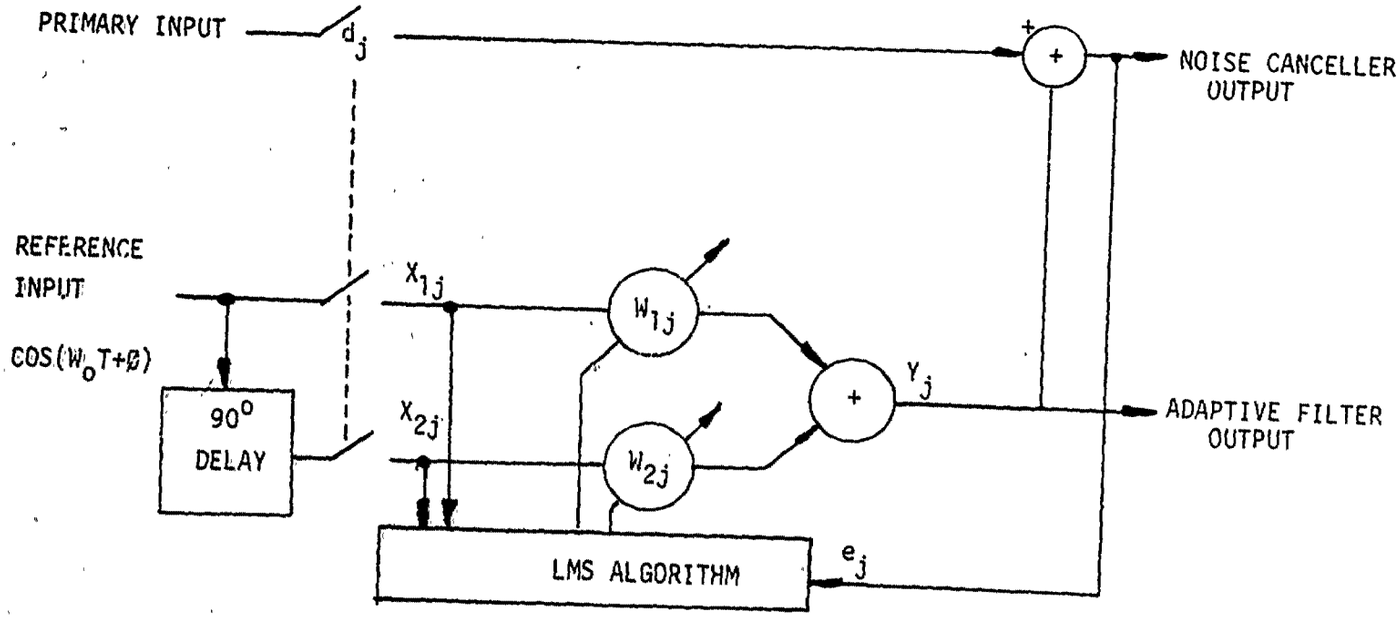
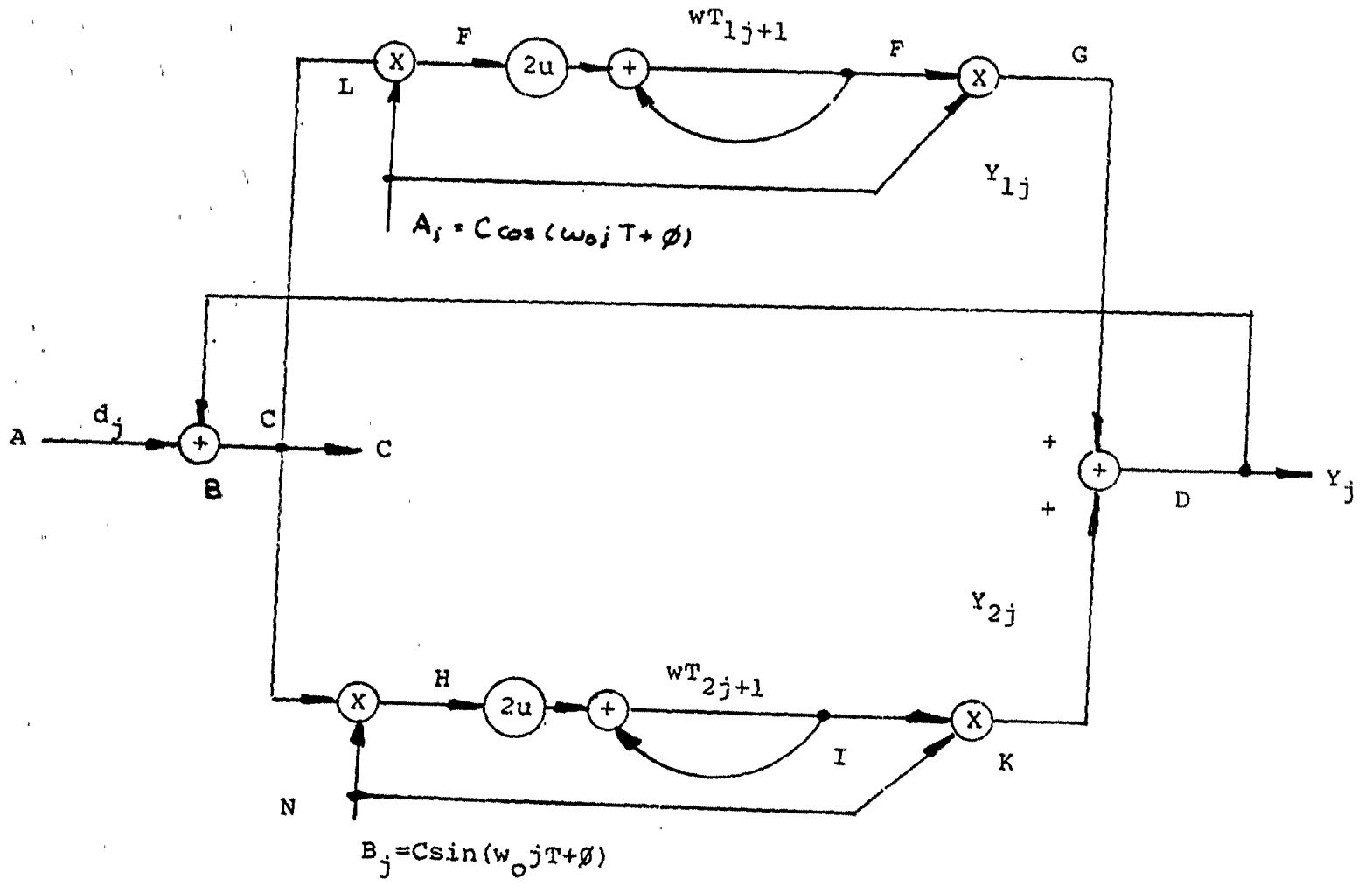


FIG. 2: SINGLE FREQUENCY ADAPTIVE NOISE CANCELLATION

FIG. 3: FLOW DIAGRAM SHOWING SIGNAL PROPOGATION IN SINGLE FREQUENCY ADAPTIVE NOISE CANCELLER



The adaptive algorithm has to adjust the weights of the adaptive linear combiner to minimise the least mean square error. The adaptive linear combiner aong with the tapped delay line forms the adaptive filter shown in fig 1.2. As before, the input signal vector is:

$$X_j = \begin{bmatrix} x_j \\ x_{j-1} \\ \vdots \\ x_{j-n+1} \end{bmatrix}$$

The componants of this vector are delayed versions of the input signal X . This filter permits the adjustment of gain and phase at many frequencies simultaneously. The total length of the delay line is determined by the reciprocal of the desired filter frequency resolution.

ADAPTIVE NOISE CANCELLER AS A NOTCH FILTER:

The notch filter is required in the situation where the primary input is corrupted by an additive undesired sinusoidal interference. A notch filter can easily realised by an adaptive noise canceller. The advantage is that it offers easy control of bandwidth and the capability of tracking the exact frequency of interference.

Fig 2 shows the single frequency noise canceller with two adaptive weights. The primary input is assumed to be any kind of signal-- periodic or transient or stochastic or the combination of these. The reference input assumed to be a pure sine wave C cos(w0+ ϕ). The primary and the reference inputs are digitised at 2*pi/T rad/sec sampling. The reference input is also phase shifted by 90 deg and again digitised.

Fig 3 is the flow diagram. It shows the operation of the LMS algorithm. The weights are updated as shown in the diagram by,

wT1(j+1)=wT1(j)+2ue(j)A(j)

wT2(j+1)=wT2(j)+2ue(j)B(j)

where e(j) is the error

The reference inputs are:

$$A(j)=C*\cos(w0\ j\ T+\phi)$$

$$B(j)=C*\sin(w0\ j\ T+\phi)$$

The isolated impulse response from the error e(j) to the filter output is obtained with the feedback loop from poit 'D' to point 'B' being assumed to be broken.

Let an impulse of amplitude 'a' be applied at the point of error signal that is at point 'C' at a discrete time j=k.

That is: at i=0, e(j)=a;

i.e. $e(j) = a * d(i)$;

and $d(i) = 1$ for $i = 0$;

$d(i)=0$ for $i = 0$;

Therefore $e(j) = a * d(j - k)$ --(3)

The response at point 'E' is then:

$$e(j)*A(j) = a * C \cos(w0\ kT+\phi) \text{ for } j=k$$

$$= 0 \quad \text{for } j = k$$

This is the input impulse scaled in amplitude by the instantaneous value of A(j) at j = k.

The signal flow path from the point 'E' to point

'F' is that of a digital integrator with transfer function of 2u/(z-1) and the impulse response $2*\mu*$ U(j-1) where U(J) is the discrete unit step function.

The response at point 'F' is obtained by convolving 2u * U(J-1) with e(j)A(j).

i.e. $A(j) = 2\mu\, a\, C \cos(w0\ kT+\phi)$ where $j \geq k+1$ -(5)

This step function which was scaled at 'E' and delayed at 'F' is now multiplied by A(j)to yield the response at 'G' as

$$y1(j)=2\mu aC\cos(w0jT+\phi)\cos(w0kT+\phi) \quad -(6)$$

where $j \geq k+1$

the response at point 'K' is also obtained. The signal flow path from point 'H' to point 'I' would show an impulse response of 2μ U(j-1) with U(j) being the step function. The response at point 'I' is then

$$Bj=2\mu\, a\, C \sin(w0\ kT+\phi) \quad --(7)$$

where $j \geq k+1$

This again is multiplied by Bj to obtain the response at 'K' as

$$y2(j) = 2\mu\, a\, C^2 \sin(w0\ jT+\phi)\sin(w0\ kt+\phi) \quad --(8)$$

where $j \geq k+1$

The combination of Y1(j) and Y2(j) yields the response at the filter output point 'D' as

$$Y(j) = 2\mu\, a\, C^2 \cos\ w0T(j-k) \quad --(9)$$

$$=2\mu aC^2 U(j-k-1)\cos w0T(j-k) \quad --(10)$$

This is a time invariant impulse response which is proportional to the input impulse. The Linear transfer

function for the noise canceller can now be derived as follows. If the time k is set to zero, the unit impulse response from the point 'C' to 'D' is

$$Y(j) = 2\mu C^2 U(j-1) \cos(w0\ jT) \qquad --(11)$$

The transfer function of this path is

$$G(z)=2uC^2\left\{\frac{z(z-\cos(w_0T)}{(z^2-2z\cos(w_oT)+1} - 1\right\}$$

$$=2uC^2\left\{\frac{z\cos(w_oT) - 1}{z^2-2z\cos(w_oT) + 1}\right\}$$

This function can be expressed in terms of radian sampling frequency $\Omega = 2\ pi/T$ as

$$G(z) = 2uC^2\left\{\frac{z\cos(2\pi w_o\Omega^{-1}) - 1}{z^2 - 2z\cos(2\pi w_o\Omega^{-1}) + 1}\right\}$$

If the feed back point from 'D' to 'B' is now closed, the transfer function H(Z) is obtained from the formulae $H = G/(1+G)$ as

$$H(z) = \left\{\frac{z^2 + 2z\cos(2\pi w_o\Omega^{-1}) - 1}{z^2 - 2z\cos(2\pi w_o\Omega^{-1}) + 1}\right\}$$

Equation 15 shows that the single frequency noise canceller has the properties of a notch filter at the reference frequency w0 .This is also shown experimentally. .

APPLICATIONS:

There are a variety of practical applications such like the cancellation of Noise in speech signals, cancellation of antenna sidelobe interference cancellation of several kinds of interference in Electrocardiography etc.. . The simulated experiments and their results will now be shown. . These would indicate the use of the adaptive noise canceller in various environments. .

Filter 1 is the hypothetical case where various fixed frequencies are present and the undesired frequency is to be cancelled. Consider an input of fixed signals at 60c/s, 350c/s, 400c/s and 450c/s. . If the 60 c/s signal is to be elliminated, the program for Filter 1 is shown along with the output plots for the filter.

Filter 2 is another form of notch filter. . This removes the 60 hz signal from a primary input of varying frequencies. . Signal varying frequencies in the range of 300 hz to 800 hz and 40 hz to 70 hz are generated as the primary input to the filter. As before the 60 hz will be removed adaptively. . The Filter 2 and it's output plots ae shown.

Adaptive filter is equally applicable to filter any type of varying signals and varying frequencies. . This is shown by Filter 3. . The primary input has the signal in varying freqs as before and the lower frequencies in the rage of 40 hz to 70 hz are completely eliminaed. The response for Filter 3 is also shown .

```
C     THIS SUBROUTINE PERFORMS THE FFT
      SUBROUTINE FFT(A,M)
      COMPLEX A(512),U,W,T
      DIMENSION AA(512),PHASE(512)
      N=2**M
      NFOUNT=N
      U=1
      WRITE(3,*) N,M
      NV2=N/2
      NM1=N-1
      J=1
      DO 7 I=1,NM1
      IF(I.GE.J) GO TO 5
      T=A(J)
      A(J)=A(I)
      A(I)=T
5     K=NV2
6     IF(K.GE.J) GO TO 7
      J=J-K
      K=K/2
      GO TO 6
7     J=J+K
      PI=3.14159265
      DO 20 L=1,M
      LE=2**L
      LE1=LE/2
      U=(1.0,0.0)
      W=CMPLX(COS(PI/LE1),(-1)**TK*SIN(PI/LE1))
      DO 20 J=1,LE1
      DO 10 I=J,N,LE
      IP=I+LE1
      T=A(IP)*U
      A(IP)=A(I)-T
      A(I)=A(I)+T
      PHASE(I)=ATAN(AIMAG(A(I))/REAL(A(I)))
C     AA(I)=CABS(A(I))
C10   WRITE(3,*) AA(I),A(I)
10    CONTINUE
20    U=U*W
      RETURN
      END
```

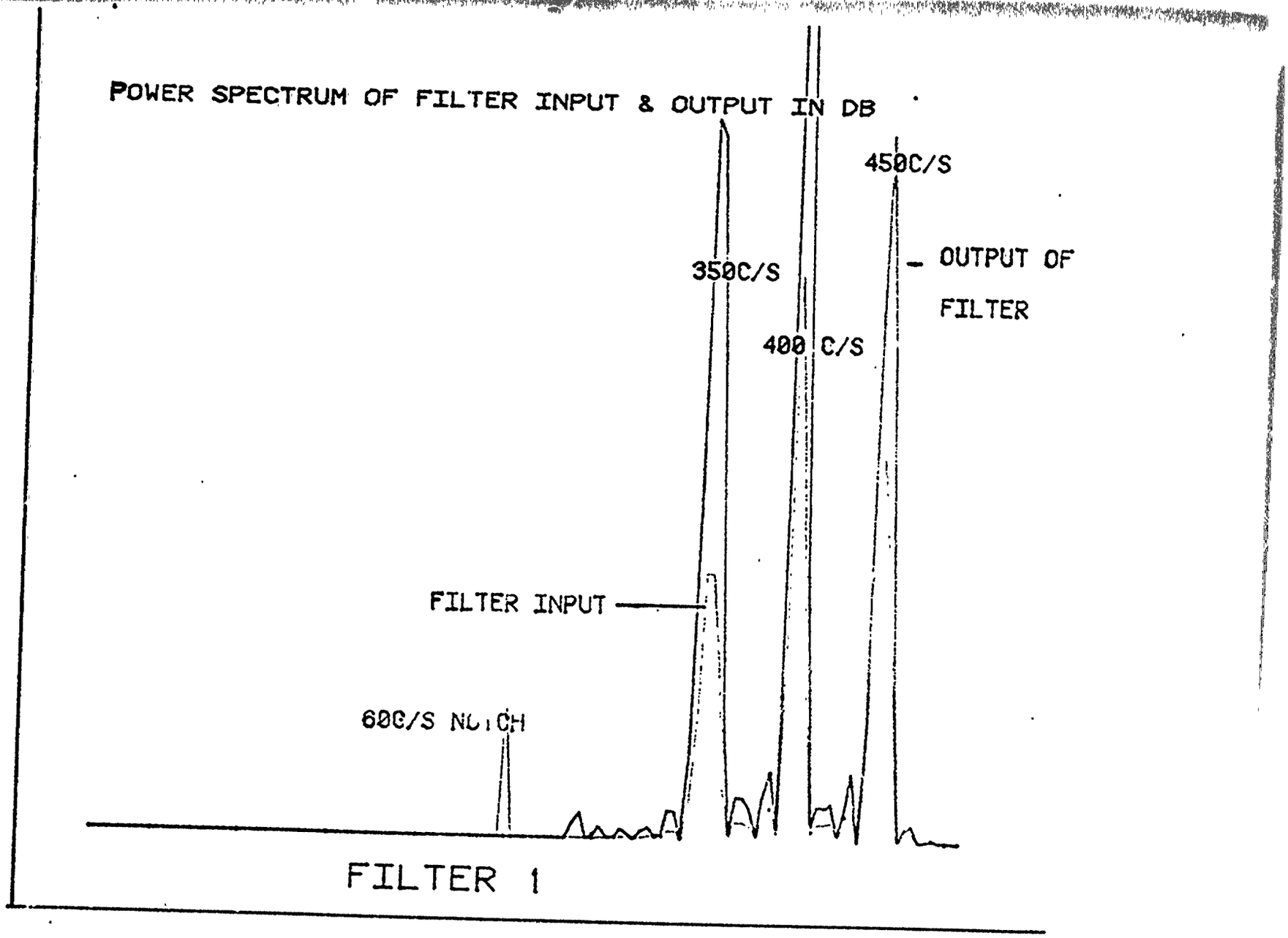
```
C       ************************************************************
C       FILTER 1:THIS FILTER HAS FIXED PRIMARY INPUT FREQUENCIES
C       AT 60HZ,350HZ,400HZAND 450HZ WHICH ARE ALL
C       SINUSOIDAL. THE 60HZ IS BEING CONSIDERED AS THE NOISE
C       FREQUENCY TO ABE REMOVED ADAPTIVELY.
C       THE OUTPUT IS A PLOT OF POWER SPECTRUM IN DB.
C       THE FILTER I/P AND O/P PLOTS ARE EXACTLY IN SAME SCALE
C       ************************************************************
        COMPLEX P1,P2,X2,X3,X4,X6,X7,X9
        COMPLEX Y1(257),S(257),RF1(257),RF2(257),X1(257)
        T=0.
        P1=(0.,0.)
        T2=0.
        P2=(0.,0.)
        X2=(0.,0.)
        X3=(0.,0.)
        X4=(0.,0.)
        X6=(0.,0.)
        X7=(0.0.)
        X9=(0.,0.)
        DO 9 I=1,257
        Y1(I)=(0.,0.)
        RF1(I)=(0.,0.)
        RF2(I)=(0.,0.)
        X1(I)=(0.,0.)
9       CONTINUE
        A=0.
        F=350.0
        DO 1 I=1,256
C       ------PRIMARY INPUT --------
        IF(I.GE.75) F=400.0
        IF(I.GE.175) F=450.0
        A=15.0*SIN(T)
        B=3.0*SIN(T2)
        P1=CMPLX(A,0.)
        P2=CMPLX(B,0.)
        S(I)=P1+P2
        T=T+2.*3.14*F/1000.0
        T2=T2+2.0*3.14*60.0/256.0
1       CONTINUE
        A=0.
C       --------REFERENCE INPUT-----
        T=0.
        DO 2 I=1,256
        A=2.0*SIN(T)
        RF1(I)=CMPLX(A,0.)
        T=T+2.0*3.14*60.0/256.0
2       CONTINUE
        T=0.
C       -------90 DEG PHASE SHIFTED REFERENCE INPUT-------
        A=0.
        DO 3 I=1,256
        A=2.0*COS(T)
        RF2(I)=CMPLX(A0.)
```

Filter 1:contd

```
      T=T+2.0*3.14*60.0/256.0
    3 CONTINUE
    C ---THE FILTER-----
      Y1(1)=(0.,0.)
      DO 5 I=1,256
      X1(I)=S(I)-Y1(I)
      X2=X1(I)*RF1(I)
      X2=X2*0.125
      X6=X7
      X7=X2+X6
      X2=X7*RF1(I)
      X3=X1(I)*RF2(I)
      X3=X3*0.125
      X9=X4
      X4=X3+X9
      X3=X4*RF2(I)
      Y1(I+1)=X2+X3
    5 CONTINUE
      IM=8
      CALL FFT(X1,IM)
      CALL FFT(Y1,IM)
      CALL FFT(S,IM)
    C S(I) PRESENTLY HAS THE FFT OF I/P S(I)
      DO 50 I=1,256
      GG=REAL(X1(I))**2+AIMAG(X1(I))**2
      GG=GG/1000.0
      X1(I)=CMPLX(GG,0.0)
      GK=REAL(Y1(I))**2+AIMAG(Y1(I))**2
      GK=GK/1000.0
      Y1(I)=CMPLX(GK,0.)
      PP=REAL(S(I))**2+AIMAG(S(I))**2
      PP=PP/1000.0
      S(I)=CMPLX(PP,0.)
   50 CONTINUE
      CALL INITT(120)
      CALL PLOTS(IBUF1,5)
    C CALL AXIS(0.,0.,'X AXIS',-6,10.,0.,0.,1.)
    C CALL AXIS(0.,0.,'Y AXIS',6,8.,90.,0.,1.)
  558 FORMAT(I4)
      WRITE (1,660)
  660 FORMAT(' ','TO START THE PLOT, HIT RETURN ')
      READ (1,558) IED
      X=0.
      CALL PLOT (0.0,0.0,-3)
      DO 303 I=1,128
      Y=REAL(S(I))
      X=X+6
      IC=2
      CALL FACTOR(0.01)
      CALL PLOT (X,Y,IC)
  303 CONTINUE
    C P.S. OF O/P (G(I)) IS PLOTTED
```

Filter 1:contd

```
      X=0.
      READ (1,558) IZK
C     CALL AXIS(0.,0.,'POW SPEC',8,9.,0.,0.,1.)
C     CALL AXIS(0.,0.,' ',1,8.,90.,0.,1.)
      CALL PLOT (0.0,0.0,-3)
      DO 90 I=1,128
      Y=REAL(X1(I))
      IC=2
      X=X+6
      CALL FACTOR (0.01)
      CALL PLOT (X,Y,IC)
90    CONTINUE
      CALL ANMODE
      CALL FINITT(0,0)
      STOP
      END
>
```

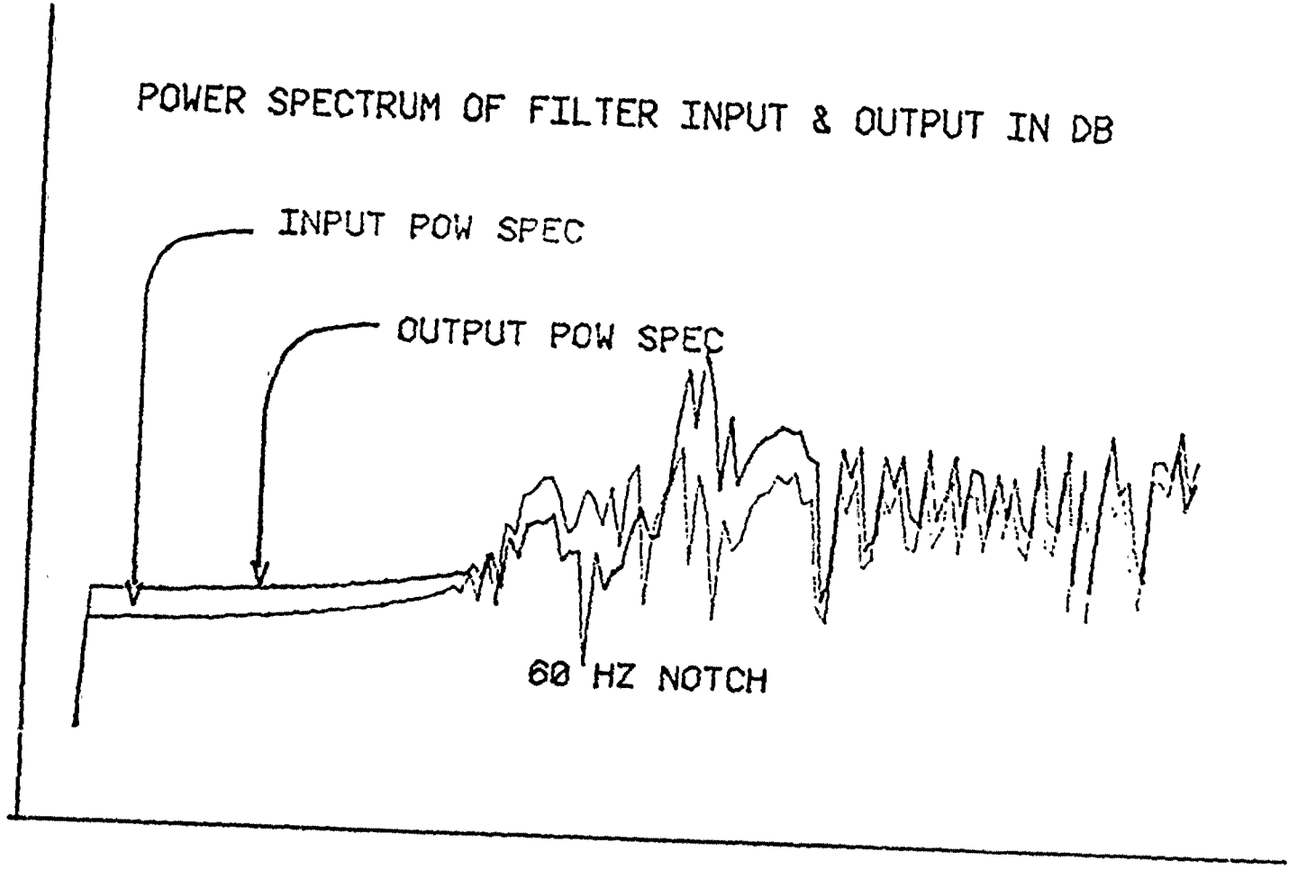

POWER SPECTRUM OF FILTER INPUT & OUTPUT IN DB
450C/S
350C/S
OUTPUT OF FILTER
400 C/S
FILTER INPUT
60C/S NOTCH
FILTER 1

```
C       ************************************************************
C
C
C       FILTER 2: THIS FILTER HAS A PRIMARY INPUT OF VARYING
C       FREQUENCIES IN THE RANGE OF 40C/S TO 70C/S AND
C       300 C/S TO 800C/S. THE FREQUENCY OF THE GENERATED SIGNAL
C       IS CONTINUOUSLY VARYING SINUSOIDALLY.
C       THE AMPLITUDE VARIATION OF THE SIGNAL IS ALSO SINUSOIDAL
C       60C/S IS ASSUMED TO BE THE NOISE FREQ TO BE ELLIMINATED
C       THE OUT IS A PLOT OF POWER SPECTRUM IN DB.
C       THE I/P AND O/P PLOTS ARE IN THE SAME SCALE
C
C
C       ************************************************************
        COMPLEX P1,P2,X2,X3,X4,X6,X7,X9
        COMPLEX Y1(257),S(257),RF1(257),RF2(257),X1(257)
        T=0.
        P1=(0.,0.)
        T2=0.
        P2=(0.,0.)
        X2=(0.,0.)
        X3=(0.,0.)
        X4=(0.,0.)
        X6=(0.,0.)
        X7=(0.,0.)
        X9=(0.,0.)
        DO 9 I=1,257
        Y1(I)=(0.,0.)
        RF1(I)=(0.,0.)
        RF2(I)=(0.,0.)
        X1(I)=(0.,0.)
9       CONTINUE
        A=0.
        U=0.
        GA=0.0
        U2=0.0
        GA2=0.0
C       ------PRIMARY INPUT (NOISE CURRUPTED)-----
        DO 1 I=1,256
        U=U+4.0
        GA=U/100
        F=400.0+(SIN(GA)*100.0)
        U2=U2+4.0
        GA2=U2/100.0
        F2=60.0+(SIN(GA)*10.0)
        A=5.0*SIN(T)
        B=3.0*SIN(T2)
        P1=CMPLX(A,0.)
        P2=CMPLX(B,0.)
        S(I)=P1+P2
        T=T+2.*3.14*F/1000.0
        T2=T2+2.0*3.14*F2/256.0
1       CONTINUE
C       -----REFERENCE INPUT-------
        A=0.
        T=0.
```

filter 2:contd

```
      DO 2 I=1,256
      A=2.0*SIN(T)
      RF1(I)=CMPLX(A,0.)
      T=T+2.0*3.14*60.0/256.0
2     CONTINUE
C     ------PHASE SHIFTED REFERENCE INPUT------
      T=0.
      A=0.
      DO 3 I=1,256
      A=2.0*COS(T)
      RF2(I)=CMPLX(A,0.)
      T=T+2.0*3.14*60.0/256.0
3     CONTINUE
C     -----THE FILTER----
      Y1(1)=(0.,0.)
      DO 5 I=1,256
      X1(I)=S(I)-Y1(I)
      X2=X1(I)*RF1(I)
      X2=X2*0.125
      X6=X7
      X7=X2+X6
      X2=X7*RF1(I)
      X3=X1(I)*RF2(I)
      X3=X3*0.125
      X9=X4
      X4=X3+X9
      X3=X4*RF2(I)
      Y1(I+1)=X2+X3
5     CONTINUE
      IM=8
      CALL FFT(X1,IM)
      CALL FFT(S,IM)
C     S(I) PRESENTLY HAS THE FFT OF I/P S(I)
      DO 50 I=1,128
      GG=REAL(X1(I))**2+AIMAG(X1(I))**2
      PP=REAL(S(I))**2+AIMAG(S(I))**2
520   FORMAT(' ','INPUT=',4X,E14.5,10X,'OUTPUT=',4X,E14.5)
      S(I)=CMPLX(PP,0.)
      X1(I)=CMPLX(GG,0.0)
50    CONTINUE
      CALL INITT(120)
      CALL PLOTS(IBUF,1,5)
      CALL PLOT (1.0,1.0,-3)
      WRITE(1,660)
      READ(1,558) IED
C     CALL AXIS(0.,0.,'INPUT POW SPEC IN DB',-20,9.,0.,0.,100.)
C     CALL AXIS(0.,0.,' ',1,7.,90.,0.,1.)
558   FORMAT(I4)
660   FORMAT(' ','TO START THE PLOT, HIT RETURN ')
      X=0.
C     CALL PLOT (1.0,1.0,-3)
      IC=2
      DO 303 I=1,128
```

```
      PP=REAL(S(I))
      PP=20.0*LOG10(PP)
      Y=PP
      X=X+3
      CALL FACTOR(0.02)
      CALL PLOT (X,Y,IC)
303   CONTINUE
C     P.S. OF O/P (G(I)) IS PLOTTED
      X=0.
      READ (1,458) IZK
458   FORMAT(I4)
C     CALL PLOT (1.0,1.0,-3)
      IC=2
C     CALL AXIS(0..0.,'POW SPEC IN DB',-14,9.,0.,0.,1.)
C     CALL AXIS(0.,0.' ',1,7.,90.,0.,1.)
      CALL PLOT (1.0,1.0,-3)
      DO 90 I=1,128
      GG=REAL(X1(I))
      GG=20.0*LOG10(GG)
      Y=GG
      X=X+3
      CALL FACTOR (0.02)
      CALL PLOT (XY,IC)
90    CONTINUE
      CALL ANMODE
      CALL FINITT(0,0)
      STOP
      END
>
```


POWER SPECTRUM OF FILTER INPUT & OUTPUT IN DB
INPUT POW SPEC
OUTPUT POW SPEC
60 HZ NOTCH

FILTER 2

Filter 2:contd

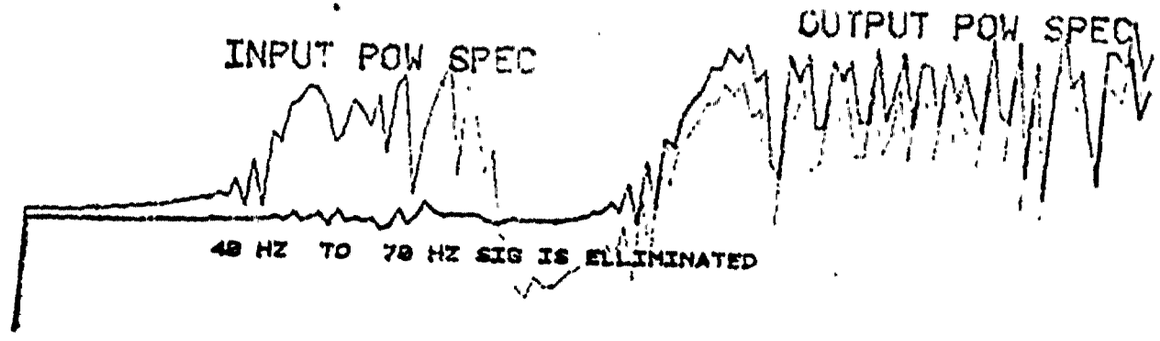
```
      PP=REAL(S(I))
      PP=20.0*LOG10(PP)
      Y=PP
      X=X+3
      CALL FACTOR(0.02)
      CALL PLOT (X,Y,IC)
303   CONTINUE
C     P.S. OF O/P (G(I)) IS PLOTTED
      X=3.
      READ (1,458) IZK
458   FORMAT(I4)
C     CALL PLOT (1.0,1.0,-3)
      IC=2
C     CALL AXIS(0..0.,'POW SPEC IN DB',-14,9.,0.,0.,1.)
C     CALL AXIS(0.,0.' ',1,7.,90.,0.,1.)
      CALL PLOT (1.0,1.0,-3)
      DO 90 I=1,128
      GG=REAL(X1(I))
      GG=20.0*LOG10(GG)
      Y=GG
      X=X+3
      CALL FACTOR (0.02)
      CALL PLOT (XY,IC)
90    CONTINUE
      CALL ANMODE
      CALL FINITT(0,0)
      STOP
      END
>
```

filter 2:contd

```
      DO 2 I=1,256
      A=2.0*SIN(T)
      RF1(I)=CMPLX(A,0.)
      T=T+2.0*3.14*60.0/256.0
2     CONTINUE
C     ------PHASE SHIFTED REFERENCE INPUT------
      T=0.
      A=0.
      DO 3 I=1,256
      A=2.0*COS(T)
      RF2(I)=CMPLX(A,0.)
      T=T+2.0*3.14*60.0/256.0
3     CONTINUE
C     -----THE FILTER----
      Y1(1)=(0.,0.)
      DO 5 I=1,256
      X1(I)=S(I)-Y1(I)
      X2=X1(I)*RF1(I)
      X2=X2*0.125
      X6=X7
      X7=X2+X6
      X2=X7*RF1(I)
      X3=X1(I)*RF2(I)
      X3=X3*0.125
      X9=X4
      X4=X3+X9
      X3=X4*RF2(I)
      Y1(I+1)=X2+X3
5     CONTINUE
      IM=8
      CALL FFT(X1,IM)
      CALL FFT(S,IM)
C     S(I) PRESENTLY HAS THE FFT OF I/P S(I)
      DO 50 I=1,128
      GG=REAL(X1(I))**2+AIMAG(X1(I))**2
      PP=REAL(S(I))**2+AIMAG(S(I))**2
520   FORMAT(' ','INPUT=',4X,E14.5,10X,'OUTPUT=',4X,E14.5)
      S(I)=CMPLX(PP,0.)
      X1(I)=CMPLX(GG,0.0)
50    CONTINUE
      CALL INITT(120)
      CALL PLOTS(IBUF,1,5)
      CALL PLOT (1.0,1.0,-3)
      WRITE(1,660)
      READ(1,558) IED
C     CALL AXIS(0.,0.,'INPUT POW SPEC IN DB',-20,9.,0.,0.,100.)
C     CALL AXIS(0.,0.,' ',1,7.,90.,0.,1.)
558   FORMAT(I4)
660   FORMAT(' ','TO START THE PLOT, HIT RETURN ')
      X=0.
C     CALL PLOT (1.0,1.0,-3)
      IC=2
      DO 303 I=1,128
```

```
C       ************************************************************
C
C
C       FILTER 2: THIS FILTER HAS A PRIMARY INPUT OF VARYING
C       FREQUENCIES IN THE RANGE OF 40C/S TO 70C/S AND
C       300 C/S TO 800C/S. THE FREQUENCY OF THE GENERATED SIGNAL
C       IS CONTINUOUSLY VARYING SINUSOIDALLY.
C       THE AMPLITUDE VARIATION OF THE SIGNAL IS ALSO SINUSOIDAL
C       60C/S IS ASSUMED TO BE THE NOISE FREQ TO BE ELLIMINATED
C       THE OUT IS A PLOT OF POWER SPECTRUM IN DB.
C       THE I/P AND O/P PLOTS ARE IN THE SAME SCALE
C
C
C       ************************************************************
        COMPLEX P1,P2,X2,X3,X4,X6,X7,X9
        COMPLEX Y1(257),S(257),RF1(257),RF2(257),X1(257)
        T=0.
        P1=(0.,0.)
        T2=0.
        P2=(0.,0.)
        X2=(0.,0.)
        X3=(0.,0.)
        X4=(0.,0.)
        X6=(0.,0.)
        X7=(0.,0.)
        X9=(0.,0.)
        DO 9 I=1,257
        Y1(I)=(0.,0.)
        RF1(I)=(0.,0.)
        RF2(I)=(0.,0.)
        X1(I)=(0.,0.)
9       CONTINUE
        A=0.
        U=0.
        GA=0.0
        U2=0.0
        GA2=0.0
C       ------PRIMARY INPUT (NOISE CURRUPTED)-----
        DO 1 I=1,256
        U=U+4.0
        GA=U/100
        F=400.0+(SIN(GA)*100.0)
        U2=U2+4.0
        GA2=U2/100.0
        F2=60.0+(SIN(GA)*10.0)
        A=5.0*SIN(T)
        B=3.0*SIN(T2)
        P1=CMPLX(A,0.)
        P2=CMPLX(B,0.)
        S(I)=P1+P2
        T=T+2.*3.14*F/1000.0
        T2=T2+2.0*3.14*F2/256.0
1       CONTINUE
C       -----REFERENCE INPUT-------
        A=0.
        T=0.
```

POWER SPECTRUM OF FILTER INPUT &OUTPUT IN DB



FILTER 3

Appendix C

1. A VLSI Residue Arithmetic Multiplier, IEEE Transactions of Circuits and Systems, W.K. Jenkins Editor, revised paper accepted for publication, approx. 10 pages.

2. Large Multiplier Multipliers, ICASSP 80 Proceedings Denver, Colorado, April 8-11, 4 pages.

3. Large Moduli Multipliers, 1980 International Symposium on Circuits and Systems, Houston, Texas, April 28-30, 3 pages.

4. A New Technique For WFTA Input/Output Reordering, International Journal of Computer and Information Sciences, J. Tou editor, accepted for Vol. 10, Number 1, approx. 15 pages.

5. The Realization of Adaptive Kalman Filter, pending ACTA, M. Hamza Editor.

PUBLISED IN THE PROCEEDINGS
OF THE ICASSP 80
DENVER, COLORADO
APRIL 9,10,11

LARGE MODULI MULTIPLIERS

Fred J. Taylor

Department of Electrical and Computer Engineering
University of Cincinnati, Cincinnati, Ohio 45221

ABSTRACT:

In this work we present a new table look-up storage scheme and a class of table look-up multipliers capable of working with exact (modular) numbering systems.

Memory savings associated with the new look-up multiplier, when compared to contemporary methods, are shown to be on the order of 2/N where $N=2^n$, n=input word length. Throughput is shown to be equal to that obtained using VLSI and classic architectures.

Introduction:

Digital signal processing is a study undergoing accelerated growth, acceptance, and application. With the possible exception of number theoretic transforms, digital signal processing has been principally advanced through technological achievements. These include the microprocessor, low cost high performance memory, and the read only memory (ROM). The availability of the ROM has challenged our traditional attitude towards performing digital arithmetic. In particular, the art of fixed point multiplication has undergone a partial metamorphosis through the use of ROM based table look-ups. Since multiplication has been a principal speed-cost-and complexity/limitation to digital filtering, advancements in this area have been warmly received.

EXISTING LOOK-UP ARITHMETIC TECHNIQUES:

Much of the reported work on ROM based fixed point multiplication has been in support of linear shift invariant digital filtering. Authors such as Jenkins and Leon, Soderstrand, etc., have studied the cost-speed metrics of digital filters using the residue numbering system. The principal advantage of the residue numbering system is that supports fixed point multiplication and addition without need of preserving "carry information". Thus, parallel operations are admissible. In addition, modular multipliers were shown to be realizable using table look-up methods and ROM's.

Jenkins recently questioned whether the performance of the residue numbering system was due to the intrinsic properties of the system or the use of look-up multipliers. It was concluded that "it appears that when no rounding (scaling) is required, the residue structure always provides better performance with regard to multiplication. When all multiplications must be rounded, the 2's complement structure provides better performance. Since most non-trivial filter and transform applications require a high plurality of multiply and add operations (almost always insuring the overflow of the limited integer dynamic ranges currently being implemented ($\leq 2^{16}$ typ.)) the future of residue based digital system may appear limited at first glance. We shall, in this work, present some new results which overcome this contemporary deficiency and in fact make the potential of modular arithmetic systems even more exciting.

Residue ALU's

The disadvantages of the residue number systems are manifold. Since the RNS possess no significant digit, decimal to residue conversion, division, magnitude comparison, and arithmetic shift operations are cumbersome and should be avoided. Register overflow, due to its finite dynamic range, impose a severe constraint on the RNS operations. Unlike weighted numbers (decimal, binary, etc.) where rounding or truncating least significant digits can control overflow, such is not the case in the RNS. Since there is an absence of least significant digits, the more general and inefficient operation known as scaling must be used. Since scaling is a form of division, its use should be discouraged. To gain insight into this problem, consider the inner product of two 31-dimensional real vectors x and y whose entries are encoded as residue digits with respect to P={32,31,29,27}. Without scaling, the dynamic range of x and y would be limited to $V^{.5}$ where V=(M/2)/31=25056. Therefore, to insure that no worst case overflow can occur, a 7.3-bit (ie: $V^{.5}=158\approx 2^{7.3}$) dynamic range limitation must be imposed on $\bar{x}$ and $\bar{y}$. With scaling, larger input ranges can be used at the expense of statistical accuracy in the output space (analogous to roundoff errors).

Due to the dynamic range of RNS systems, one is generally forced to accept one of the following two overflow prevention strategies.

1. Increase the dynamic range to a sufficiently large value by adding more moduli to P, or
2. Make scaling a more efficient operation.

The first option represents a brute force attack to the problem. Such an approach will increase to

cost and complexity metrics of a filter. In addition, the moduli set P must be tailored to a unique filter. The other approach appears to be the most popular at this time. Szabo and Tanaka, and others, have concentrated on the scaling efficiency through the choice of the three-tuple moduli set $P=\{2^n-1, 2^n, 2^n+1\}$. This moduli set has the ability to efficiently scale a residue number by any one of the chosen moduli. However, there is an intrinsic limitation plaguing this method and it is its dynamic range. Using a large high-speed memory unit, say 4Kx1, the input addressing space is limited to 2^{12}. This means that a moduli p_i is technically limited to $p_i \leq 2^6$ (ie: $x_i \& y_i < 2^{12}$). Therefore, the dynamic range of any modular operation is given by $M=(2^n-1)(2^n)(2^n+1) \leq 2^{3n}=2^{18}$. In many applications, an 18-bit resolution is insufficient resolution.

New Results

Two new memory efficient algorithms have been derived and is based on a novel factorization of a bilinear form. Over a real field it is obvious that

$$xy=((x+y)/2)^2-((x-y)/2)^2 \qquad 1.$$

which in modular form, becomes

$$\langle xy \rangle_p = \langle \phi(s^+)-\phi(s^-) \rangle_p \qquad 2.$$

where $\phi(s)=\langle s^2 \rangle_p$ with $s^+=(x+y)/2$ and $s^-=(x-y)/2$.

At first glance this algorithm, which shall be referred to as a minimal memory modular multiplier (M^4), would appear to be counterproductive with respect to a memory conservation metric. The memory requirements associated with the M^4 will be shown to be substantially less than those of direct mechanizations. First, it should be apparent that the integer s^+ and s^- found in equation 2 is bounded from above by 2^{n+1}. Therefore, only a (n+1)-bit table addressing space is required to realize $\phi(s^\pm)$ versus the 2n-bit space needed for direct architectures. It would appear however, that there is an exception to this rule. Since one of the moduli chosen is $p=2^n+1$. Here the maximal value of s^+(or s^-) is 2^{n+1} which would technically require a (n+2)-bit address. However, by using the protocol found in figure 1, the table size can be reduced to 2^{n+1} words for all moduli. Here, the overflow bit serves to differentiate $s^\pm=0$ from 2^{n+1}.

The M^4 system architecture is abstracted in Figure 2. This uses 2^{n+1} word high-speed memory for modular arithmetic look-up operations. Using, for example, the previously referenced 4K-30ns device, moduli having an 11-bit dynamic range (vs. 6-bit in the direct form) can be mechanized. This would yield a three-moduli dynamic range on the order of $2^{3(11)} \approx 8.6 \cdot 10^9$. That is, without an increase in memory size (and therefore access time), the dynamic range of the M^4 is $2^{33}/2^{18}=2^{15}$ times larger than that obtainable through direct means! This large increase in dynamic range makes the RNS a viable alternative to traditional filter design methods. Both improved precision and throughput (through the reduction or absence of traditional scaling operations) can be achieved. Finally, several versions of the M^4 algorithm can be considered. They are summarized in figure 3.

Upon closer investigation of the table look-up data base, a potential nuisance can be found. It can be examplified by observing that if $s^-=9$, $p=32$, then $\phi(s^-)=\langle 9^2/4 \rangle_{32}=20.25$. Therefore, it may be required that two additional fractional bits may need to be added to the table's output word length. However, this is not the case as suggested by the following theorem:

Theorem: Let $\|v\|$ denote the integer value of v.

$$\text{Then } z=\langle \|\phi(s^+)\|-\|\phi(s^-)\| \rangle_p.$$

That is, only the integer value of ϕ need be used and the fractional bits of $\phi(s^\pm)$ can be ignored.

Proof: Let $(x+y)/2=v+k/2$; $(x-y)/2=q+b/2$ where $k=0$ or 1. Then

$$z=\langle\langle (x+y)^2/4 \rangle_p - \langle (x-y)^2/4 \rangle_p \rangle = \langle\langle v+kv+b^2/4 \rangle_p - \langle q+kv+k^2/4 \rangle_p \rangle_p$$
$$= \langle\langle v+kv \rangle_p + k^2/4 - \langle q+kq \rangle_p - b^2/4 \rangle_p = \langle \phi(s^+)-\phi(s^-) \rangle_p$$

As a result, the parallel architecture is equivalent to that shown in figure 4.

Modulo p Adder

The M^4 multiplier requires a modulo p adder be used to combine the two component parts of the solution (namely $\phi(s^+)$ and $\phi(s^-)$). Modulo p adders pose an interesting design problem. Unless a fast modulo p adder can be fabricated, the overhead associated with addition will offset any gain in throughput achieved through table look-ups. For the moduli chosen, 2^n-1, 2^n, and 2^n+1, only the modulo 2^n adder can be realized directly (n-bit adder with ignored overflow). It would however, be desirable to use a n-bit adder to realize the modulo 2^n-1 and 2^n+1 adder as well.

Using n-bit AND gates to sense the zero condition of $\langle s \rangle_{2^N}$, the overflow bit OVF, and the sign bits of $\phi(s^+)$ and $\phi(s^-)$, a combinational logic routine can be defined which will convert $\langle s \rangle_{2^N}$ into $\langle s \rangle_p$. It can be noted that the mapping requirements are:

1. for $p=2^N-1$, map s to s or $s-2^N+1=\langle\langle s \rangle_{2^N}+1\rangle_{2^N}$
2. for $p=2^N$, map s to $s-2^N=\langle s \rangle_{2^N}$
3. for $p=2^N+1$, map s to s or $s-2^N-1=\langle\langle s \rangle_{2^N}-1\rangle_{2^N}$

Suppose the moduli $p=2^n+1$, n=12, is to be implemented. By using two commercially available 16x9 PLA's in parallel, the 12-bit outcome of an n-bit adder and the four control bits, can be converted to 13-bit mask. The mask would transform the output of a high-speed n-bit adder to s or $s-2^n-1$, depending on the state of the 4 control bits. Based on a 25-ns 12-bit Schottky look-ahead adder, a 20-ns 16x9 PLA, and 10-ns FET mask switches a 65-ns modulo p adder, for $p=2^n-1$, 2^n, and 2^n+1 can be realized. The

presence of a 65-ns modulo p adder will now allow a 140-ns large moduli residue multiplier to be based on 35-ns 4Kx1 NMOS memory units. For a moduli set $\{2^{12}-1, 2^{12}, 2^{12}+1\}$, a fixed point multiplier, having an output dynamic range of $2^{36}-2^{12}$, can thus be fabricated having a word rate of 7.143M multiplications per second or 28.5M multiplications per second if a pipelined architecture is used.

Summary:

The residue number system offers the potential for high speed parallel arithmetic. This class of arithmetic has been demonstrated to be useful in designing recursive algorithms, transforms, and digital filters. One of the principal limitations to its use is its limited practical dynamic range. To overcome this problem, a large moduli multiplier, for the moduli set $\{2^n-1, 2^n, 2^n+1\}$, was designed. This high-speed large moduli system was the product of the new M^4 algorithm and new technologies (RAM and PLA's). The practical residue multiplier is capable of supporting a pipelined execution rate of 28.5M multipliers per second.

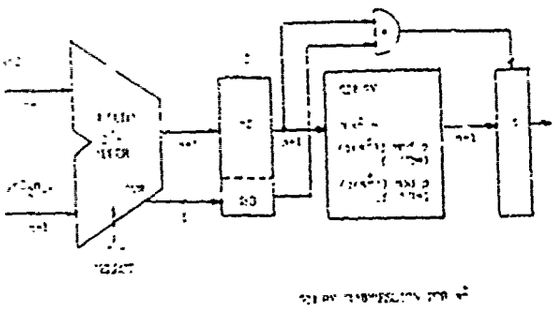


Figure 1

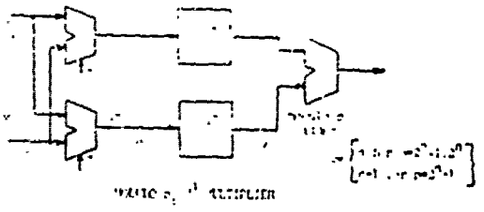


Figure 2

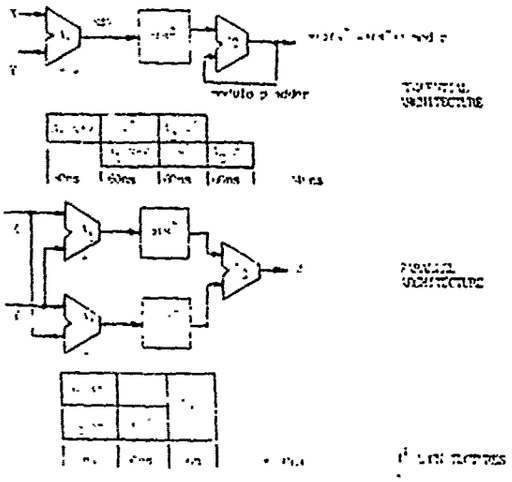


Figure 3

References

1. G.A. Jullien, "Residue Number Scaling and Other Operations Using ROM Arrays," IEEE Trans. Computers, Vol C-27, No. 4, pp 325-337, April 1978.

2. C.H. Huang and F.J. Taylor, "Memory Compression Scheme For Modular Arithmetic, to be published, IEEE ASSP.

3. M.A. Soderstrand, "A High Speed Low-Cost Recursive Filter Using Residue Arithmetic," Proc. IEEE, Vol. 65, No. 7, July 1977.

4. N.S. Szabo and R.I. Tanaka, Residue Arithmetic and Its Applications To Computer Technology, McGraw-Hill, 1967.

5. W.K. Jenkins and B.J. Leon, "The Use Of Residue Number System in the Design of Finite Impulse Response Filters," IEEE C&S, CAS-24, April 1977.

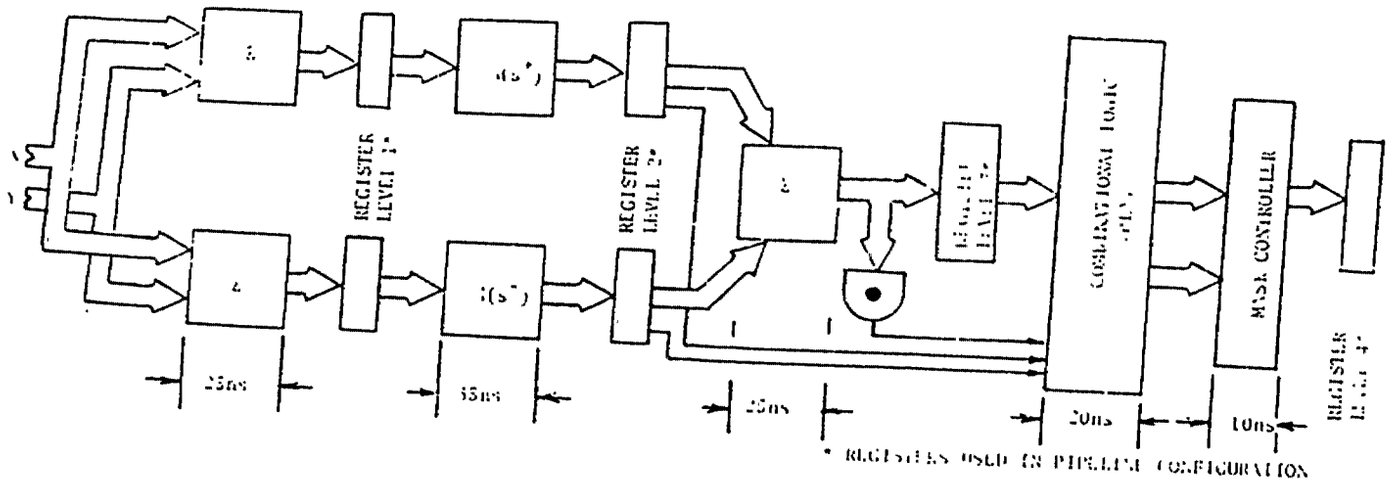
6. W.K. Jenkins, "Techniques for Residue-to-Analog Conversion for Residue-Encoded Digital Filters," IEEE C&S, CAS-25, July 1978.

7. A. Peled and B. Liu, "A New Hardware Realization of Digital Filters," IEEE ASSP, ASSP-22, December 1974.

8. W.J. Jenkins, "A Highly Efficient Residue-Combinatorial Architecture for Digital Filters," Proc. of the IEEE (Letter), Vol. 66, No. 6, June 1978, pp 700-702.

This work was partially supported under AFOSP Grant No. AF49620-79-C-0066.

140



* REGISTERS USED IN PIPELINE CONFIGURATION

LARGE MODULE MULTIPLIER

LARGE MODULI MULTIPLIERS

Fred J. Taylor

Department of Electrical and Computer Engineering
University of Cincinnati, Cincinnati, Ohio 45221

ABSTRACT:

In this work we present a new table look-up storage scheme and a class of table look-up multipliers capable of working with exact (modular) numbering systems.

Memory savings associated with the new look-up multiplier, when compared to contemporary methods, are shown to be on the order of 2/N where $N=2^n$, n=input wordlength. Throughput is shown to be equal to that obtained using VLSI and classic architectures.

Introduction:

Digital signal processing is a study undergoing accelerated growth, acceptance, and application. With the possible exception of number theoretic transforms, digital signal processing has been principally advanced through technological achievements. These include the microprocessor, low cost high performance memory, and the read only memory (ROM). The availability of the ROM has challenged our traditional attitude towards performing digital arithmetic. In particular, the art of fixed point multiplication has undergone a partial metamorphosis through the use of ROM based table look-ups. Since multiplication has been a principal speed-cost-and complexity limitation to digital filtering, advancements in this area have been warmly received.

EXISTING LOOK-UP ARITHMETIC TECHNIQUES:

Much of the reported work on ROM based fixed point multiplication has been in support of linear shift invariant digital filtering. Authors such as Jenkins and Leon, Soderstrand, etc., have studied the cost-speed metrics of digital filters using the residue numbering system. The principal advantage of the residue numbering system is that supports fixed point multiplication and addition without need of preserving "carry information". Thus, parallel operations are admissible. In addition, modular multipliers were shown to be realizable using table look-up methods and ROM's.

Jenkins recently questioned whether the performance of the residue numbering system was due to the intrinsic properties of the system or the use of look-up multipliers. It was concluded that "it appears that when no rounding (scaling) is required, the residue structure always provides better performance with regard to multiplication. When all multiplications must be rounded, the 2's complement structure provides better performance. Since most non-trivial filter and transform applications require a high plurality of multiply and add operations (almost always insuring the overflow of the limited integer dynamic ranges currently being implemented ($<2^{16}$ typ.)) the future of residue based digital system may appear limited at first glance. We shall, in this work, present some new results which overcome this contemporary deficiency and in fact make the potential of modular arithmetic systems even more exciting.

Residue ALU's

The disadvantages of the residue number systems are manifold. Since the RNS possess no significant digit, decimal to residue conversion, division, magnitude comparison, and arithmetic shift operations are cumbersome and should be avoided. Register overflow, due to its finite dynamic range, impose a severe constraint on the RNS operations. Unlike weighted numbers (decimal, binary, etc.) where rounding or truncating least significant digits can control overflow, such is not the case in the RNS. Since there is an absence of least significant digits, the more general and inefficient operation known as scaling must be used. Since scaling is a form of division, its use should be discouraged. To gain insight into this problem, consider the inner product of two 31-dimensional real vectors x and y whose entries are encoded as residue digits with respect to P={32,31,29,27}. Without scaling, the dynamic range of x and y would be limited to $V^{\cdot 5}$ where V=(M/2)/31=25056. Therefore, to insure that no worst case overflow can occur, a 7.3-bit (ie: $V^{\cdot 5} \approx 2^{7.3}$) dynamic range limitation must be imposed on x and y. With scaling, larger input ranges can be used at the expense of statistical accuracy in the output space (analogous to roundoff errors).

Due to the dynamic range of RNS systems, one is generally forced to accept one of the following two overflow prevention strategies.

1. Increase the dynamic range to a sufficiently large value by adding more moduli to P, or
2. Make scaling a more efficient operation.

The first option represents a brute force attack to the problem. Such an approach will increase to

CH1559-4/80/0000-0792$00.75 © 1980 IEEE

cost and complexity metrics of a filter. In addition, the moduli set P must be tailored to a unique filter. The other approach appears to be the most popular at this time. Szabo and Tanaka, and others, have concentrated on the scaling efficiency through the choice of the three-tuple moduli set $P=\{2^n-1, 2^n, 2^n+1\}$. This moduli set has the ability to efficiently scale a residue number by any one of the chosen moduli. However, there is an intrinsic limitation plaguing this method and it is its dynamic range. Using a large high-speed memory unit, say 4Kx1, the input addressing space is limited to 2^{12}. This means that a moduli p_i is technically limited to $p_i<2^6$ (ie: $x_i \& y_i<2^{12}$). Therefore, the dynamic range of any modular operation is given by $M=(2^n-1)(2^n)(2^n+1)\approx 2^{3n}=2^{18}$. In many applications, an 18-bit resolution is insufficient resolution.

New Results

Two new memory efficient algorithms have been derived and is based on a novel factorization of a bilinear form. Over a real field it is obvious that

$$xy=((x+y)/2)^2-((x-y)/2)^2 \qquad 1.$$

which in modular form, becomes

$$\langle xy\rangle_p=\langle\phi(s^+)-\phi(s^-)\rangle_p \qquad 2.$$

where $\phi(s)=\langle s^2\rangle_p$ with $s^+=(x+y)/2$ and $s^-=(x-y)/2$.

At first glance this algorithm, which shall be referred to as a minimal memory modular multiplier (M^4), would appear to be counterproductive with respect to a memory conservation metric. The memory requirements associated with the M^4 will be shown to be substantially less than those of direct mechanizations. First, it should be apparent that the integer s^+ and s^- found in equation 2 is bounded from above by 2^{n+1}. Therefore, only a (n+1)-bit table addressing space is required to realize $\phi(s^{\pm})$ versus the 2n-bit space needed for direct architectures. It would appear however, that there is an exception to this rule. Since one of the moduli chosen is $p=2^n+1$. Here the maximal value of s^+(or s^-) is 2^{n+1} which would technically require a (n+2)-bit address. However, by using the protocol found in figure 1, the table size can be reduced to 2^{n+1} words for all moduli. Here, the overflow bit serves to differentiate $s^{\pm}=0$ from 2^{n+1}.

The M^4 system architecture is abstracted in Figure 2. This uses 2^{n+1} word high-speed memory for modular arithmetic look-up operations. Using, for example, the previously referenced 4K-30ns device, moduli having an 11-bit dynamic range (vs. 6-bit in the direct form) can be mechanized. This would yield a three-moduli dynamic range on the order of $2^{3(11)}\approx 8.6\cdot 10^9$. That is, without an increase in memory size (and therefore access time), the dynamic range of the M^4 is $2^{33}/2^{18}=2^{15}$ times larger than that obtainable through direct means! This large increase in dynamic range makes the RNS a viable alternative to traditional filter design methods. Both improved precision and throughput (through the reduction or absence of traditional scaling operations) can be achieved. Finally, several versions of the M^4 algorithm can be considered. They are summarized in figure 3.

Upon closer investigation of the table look-up data base, a potential nuisance can be found. It can be examplified by observing that if $s^{\pm}=9$, $p=32$, then $\phi(s^{\pm})=\langle 9^2/4\rangle_{32}=20.25$. Therefore, it may be required that two additional fractional bits may need to be added to the table's output word length. However, this is not the case as suggested by the following theorem:

Theorem: Let $||v||$ denote the integer value of v.

Then $z=\langle||\phi(s^+)||-||\phi(s^-)||\rangle_p$.

That is, only the integer value of ϕ need be used and the fractional bits of $\phi(s^{\pm})$ can be ignored.

Proof: Let $(x+y)/2\triangleq v+k/2$; $(x-y)/2\triangleq q+b/2$ where k=0 or 1. Then $z=\langle\langle(x+y)^2/4\rangle_p-\langle(x-y)^2/4\rangle_p\rangle_p=\langle\langle v+kv+b^2/4\rangle_p-\langle q+kv+k^2/4\rangle_p\rangle_p$
$=\langle\langle v+kv\rangle_p+k^2/4-\langle q+kq\rangle_p-b^2/4\rangle_p=\langle\phi(s^+)-\phi(s^-)\rangle_p$

As a result, the parallel architecture is equivalent to that shown in figure 4.

Modulo p Adder

The M^4 multiplier requires a modulo p adder be used to combine the two component parts of the solution (namely $\phi(s^+)$ and $\phi(s^-)$). Modulo p adders pose an interesting design problem. Unless a fast modulo p adder can be fabricated, the overhead associated with addition will offset any gain in throughput achieved through table look-ups. For the moduli chosen, 2^n-1, 2^n, and 2^n+1, only the modulo 2^n adder can be realized directly (n-bit adder with ignored overflow). It would however, be desirable to use a n-bit adder to realize the modulo 2^n-1 and 2^n+1 adder as well.

Using n-bit AND gates to sense the zero condition of $\langle s\rangle_{2^N}$, the overflow bit OVF, and the sign bits of $\phi(s^+)$ and $\phi(s^-)$, a combinational logic routine can be defined which will convert $\langle s\rangle_{2^N}$ into $\langle s\rangle_p$. It can be noted that the mapping requirements are:

1. for $p=2^N-1$, map s to s or $s-2^N+1=\langle\langle s\rangle_{2^N}+1\rangle_{2^N}$
2. for $p=2^N$, map s to $s-2^N=\langle s\rangle_{2^N}$
3. for $p=2^N+1$, map s to s or $s-2^N-1=\langle\langle s\rangle_{2^N}-1\rangle_{2^N}$

Suppose the moduli $p=2^n+1$, n=12, is to be implemented. By using two commercially available 16x9 PLA's in parallel, the 12-bit outcome of an n-bit adder and the four control bits, can be converted to 13-bit mask. The mask would transform the output of a high-speed n-bit adder to s or $s-2^n-1$, depending on the state of the 4 control bits. Based on a 25-ns 12-bit Schottky look-ahead adder, a 20-ns 16x9 PLA, and 10-ns FET mask switches, a 65-ns modulo p adder, for $p=2^n-1$, 2^n, and 2^n+1 can be realized. The

presence of a 65-ns modulo p adder will now allow a 140-ns large moduli residue multiplier to be based on 35-ns 4Kx1 HMOS memory units. For a moduli set $\{2^{12}-1, 2^{12}, 2^{12}+1\}$, a fixed point multiplier, having an output dynamic range of $2^{36}-2^{12}$, can thus be fabricated having a word rate of 7.143M multiplications per second or 28.5M multiplications per second if a pipelined architecture is used.

Summary:

The residue number system offers the potential for high speed parallel arithmetic. This class of arithmetic has been demonstrated to be useful in designing recursive algorithms, transforms, and digital filters. One of the principal limitations to its use is its limited practical dynamic range. To overcome this problem, a large moduli multiplier, for the moduli set $\{2^n-1, 2^n, 2^n+1\}$, was designed. This high-speed large moduli system was the product of the new M^4 algorithm and new technologies (RAM and PLA's). The practical residue multiplier is capable of supporting a pipelined execution rate of 28.5M multipliers per second.

References

1. G.A. Jullien, "Residue Number Scaling and Other Operations Using ROM Arrays," IEEE Trans. Computers, Vol C-27, No. 4, pp 325-337, April 1978.

2. C.H. Huang and F.J. Taylor, "Memory Compression Scheme For Modular Arithmetic, to be published, IEEE ASSP.

3. M.A. Soderstrand, "A High Speed Low-Cost Recursive Filter Using Residue Arithmetic," Proc. IEEE, Vol. 65, No. 7, July 1977.

4. N.S. Szabo and R.I. Tanaka, Residue Arithmetic and Its Applications To Computer Technology, McGraw-Hill, 1967.

5. W.K. Jenkins and B.J. Leon, "The Use Of Residue Number System in the Design of Finite Impulse Response Filters," IEEE C&S, CA5-24, April 1977.

6. W.K. Jenkins, "Techniques for Residue-to-Analog Conversion for Residue-Encoded Digital Filters," IEEE C&S, CA5-25, July 1978.

7. A. Peled and B. Liu, "A New Hardware Realization of Digital Filters," IEEE ASSP, ASSP-22, December 1974.

8. W.J. Jenkins, "A Highly Efficient Residue-Combinatorial Architecture for Digital Filters," Proc. of the IEEE (Letter), Vol. 66, No. 6, June 1978, pp 700-702.

This work was partially supported under AFOSR Grant No. AF49620-79-C-0066.

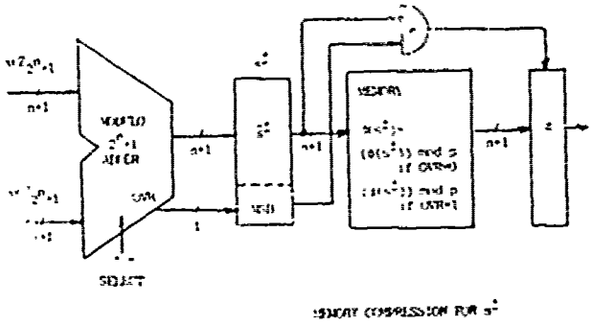


Figure 1

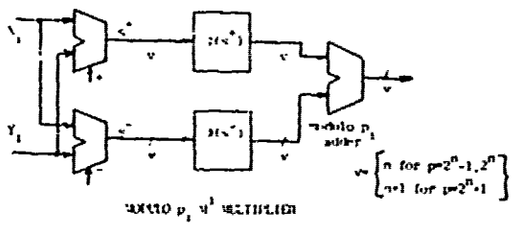


Figure 2

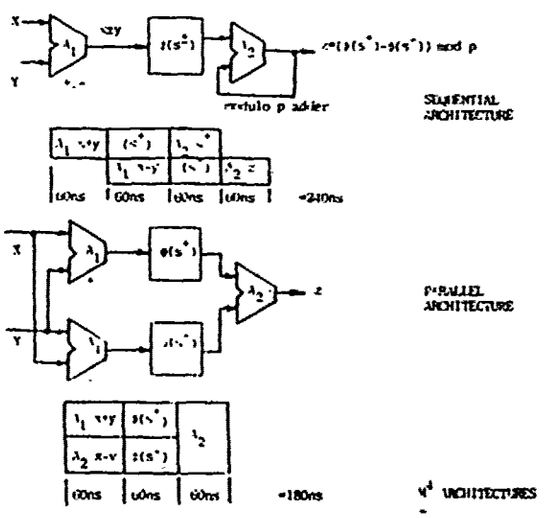


Figure 3

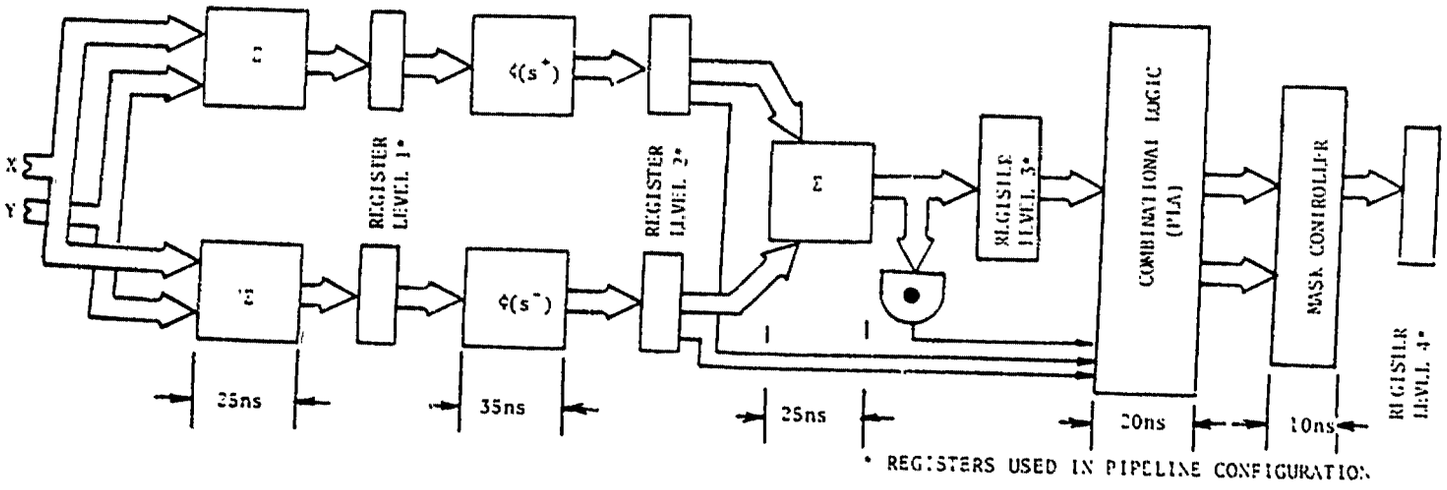


Figure 4

LARGE MODULI MULTIPLIER

LARGE MODULI MULTIPLIERS FOR SIGNAL PROCESSING

by

F.J. Taylor
University of Cincinnati

Abstract

The residue number system has recently been shown to be a viable signal processing media. However, it does possess limitations. One of the most serious is overflow prevention through magnitude scaling. One method of overcoming this defect is to increase the dynamic range of the numbering system. To this end a new high-speed large moduli multiplier has been developed. The multiplier, which is the result of combining the quarter squared algorithm with recent breakthroughs in device technology. As a result, equivalent 18-bit full precision products can be obtained at a pipelined rate of 28.5M multiples per second.

This work was partially supported under AFOSR grant F49620-79-C-0066.

I. INTRODUCTION

Recently, the residue number system (RNS) has received renewed attention in the literature [1-3]. This mathematically mature study was, until the present, in the background of digital system design because of its historic inability of digital hardware to support RNS arithmetic [4]. However, recent breakthroughs in the area of read-only memory technology has significantly altered this case. Using high-speed bipolar ROM's, the ability of the RNS to support ultra high-speed digital filtering has been experimentally demonstrated [5]. The question of decimal-to-residue I/O operations has also been addressed [6]. However, a major obstical to the cause of RNS filtering has been register overflow protection. In order to guarantee that system registers do not overflow during run-time, an inefficient operation referred to as scaling has to be performed. If scaling were not required, RNS filters was shown to possess higher throughput rates than those obtainable using distributed arithmetic (ie: bit-slice; ref [7]) [8]. However, when scaling is required, the RNS architecture was shown to be at a disadvantage. It should be remembered however that the distributed arithmetic filter is a constant coefficient device (ie: shift-invariant) whereas the linear RNS filter is general (ie: variable coefficient). Therefore the RNS provides the user with the versatility needed to perform adaptive, optimal (ex: minimal variance in a non-stationary stochastic environment), frequency tuneable filtering which cannot be supported in a bit-slice configuration.

In this work, a new multiplier architecture is developed which significantly enhances the case for RNS filters by significantly reducing scaling overhead. The high-speed residue multiplier will be shown to increase the dynamic range of the RNS to a value which either reduces the number of scaling operations to a small fraction of their original number

or make scaling unnecessary. All this is accomplished without increasing the memory budget over above that found in contemporary RNS designs.

II. RNS OVERVIEW

Interest in the RNS is due to its ability to perform high-speed arithmetic. Speed is achieved through the use of a high degree of parallelism and an absence of carry information requirements. These two attributes are a byproduct of the fact that there does not exist a most (least) significant residue digit. That is, all residue digits are of equal importance. More specifically, if P is a moduli set such that $P = \{p_1,\ldots,p_L\}$, and the p_i's are relatively prime, then if $x \in [-M/2, M/2)$, x is uniquely represented by the L-tuple

$$x \mapsto (x_1,\ldots,x_L) \qquad 1.$$

with

$$x_i = \begin{cases} \langle x \rangle_{p_i} & \text{if } p_i \geq 0 \\ p_i - \langle |x| \rangle_{p_i} & \text{otherwise} \end{cases} \qquad 2.$$

where $\langle x \rangle_{p_i}$ denotes x modulo p_i and $M = \prod_{i=1}^{L} p_i$. The bilinear composition of two integers, say $x \mapsto (x_1,\ldots,x_L)$ and $y \mapsto (y_1,\ldots,y_L)$, is given by $x \circ y$ (where $\circ$ denotes addition, subtraction, or multiplication) is given by

$$x \circ y \mapsto (x_1 \circ y_1,\ldots,x_L \circ y_L). \qquad 3.$$

It can be seen each residue digit, namely $x_i \circ y_i$ can be computed independent of all others (ie: no carry information requirements). In practice, the mapping of x_i and y_i into $x_i \circ y_i$ is accomplished using table lookups where the table residue on randomly accessed read-only memory. Typical high-speed memory modules, which are currently available, are:

Device	Type	Technology	Configuration	Access-Speed
10149	ROM	ECL	256x4	20 ns
SN54S	ROM	TTL	1024x4	35 ns
2147H-1	RAM	HMOS	4096x1	30 ns
2125H-1	RAM	HMOS	1024x1	20 ns
I2167	RAM	HMOS	16384x1	45 ns

The product of two residues modulo p_i, $p_i \cdot 2^n_i$ can be precomputed and stored in a 2^mxn-bit memory unit where m=2n. Using a large existing high-speed memory (4Kx1 at 30 ns), residues having up to six bit integer values can be used (ex: P = {64,63,...}). Thus, fixed-point multipliers having a dynamic range of [-M/2,M/2) can be architected which have execution rates in the low nanoseconds.

The disadvantages of the residue number systems are manifold. Since the RNS possess no most significant digit, decimal to residue conversion, division, magnitude comparison, and arithmetic shift operations are clumbersome and should be avoided. Register overflow, due to its finite dynamic range, impose a severe constraint on the RNS operations. Unlike weighted numbers (decimal, binary, etc.) where rounding or truncating least significant digits can control overflow, such is not the case in the RNS. Since there is an absence of least significant digits, the more general and inefficient operation known as scaling must be used. Since scaling is a form of division, its use should be discouraged. To gain insight into this problem, consider the inner product of two 31-dimensional real vectors x and y whose entries are encoded as residue digits with respect to P = {32,31,29,27}. Without scaling, the worst-case value of x and y would be limited to $V^{.5}$ where V = (M/2)/31 = 25056. Therefore, to insure that no worst case overflow can occur, a 7.3-bit (ie: $V^{.5} \approx 158 \approx 2^{7.3}$) dynamic range limitation must be imposed on x and y. With scaling, larger input ranges can be used at

the expense of statistical accuracy in the output space (analogous to roundoff errors).

Due to the dynamic range limitation of RNS systems, one is generally forced to accept one of the following two overflow prevention strategies.

1. Increase the dynamic range to a sufficiently large value by adding more moduli to P. or
2. Make scaling a more efficient operation.

The first option represents a brute force attack to the problem. Such an approach will increase to cost and complexity metrics of a filter. In addition, the moduli set P must be tailored to unique filter. The other approach appears to be the most popular at this time. Sazbo and Tanaka, and others, have concentrated on the scaling efficiency through the choice of the three-tuple moduli set $P = \{2^n-1, 2^n, 2^n+1\}$. This moduli set has the ability to efficiently scale a residue number by any one of the chosen moduli. However, there is an intrinsic limitation plauging this method and it is its dynamic range. Using a large high-speed memory unit, say 4Kx1, the input addressing space is limited to 2^{12}. This means that a moduli p_i is technically limited to $p_i \leq 2^6$ (ie: $x_i - y_i < 2^{12}$). Therefore, the dynamic range of any modular operation is given by $M = (2^n-1)(2^n)(2^n+1) \sim 2^{3n} = 2^{18}$. In many applications, an 18-bit resolution is insufficient resolution.

III. Principal Result

It is desirable to keep the previously discussed three moduli structure for purposes of potential scaling needs. However, in order to overcome the existing disadvantages of this system, that of dynamic range, a new approach is called for. Since it is unrealistic to assume substantially larger density high-speed memories will continue to become available, it is incumbent that more memory efficient residue arithmetic unit be designed.

An efficient algorithm, which is ideally suited for this application, is known as the quarter-square multiplier [9-11].

Over a real field it is obvious that

$$xy = ((x+y)/2)^2 - ((x-y)/2)^2 \qquad 4.$$

which in modular form, it becomes

$$\langle xy \rangle_p = \langle \phi(s^+) - \phi(s^-) \rangle_p \qquad 5.$$

where $\phi(s) = \langle s^2 \rangle_p$ with $s^+ = (x+y)/2$ and $s^- = (x-y)/2$.

The quarter-squared multiplier has been studied by J.M. Pollard (1976) in a Galois field. Questions of hardware implementation were not considered and, due to the Galois field limitation, only prime moduli could be considered. H. Nussbaumer (1976) studied the quarter-square multiplier over real fields for use in ROM intensive digital filters. Soderstrand and Fields (1977) made brief reference to this multiplier for residue arithmetic but offered no satisfactory hardware realization. In this paper, a practical residue arithmetic quarter-squared multiplier will be architected using commercially available hardware.

A problem that would seem to plauge the quarter-square multiplier is the need to realize the division by two the sums and differences found in Eq. 4. In general, the existance of an N^{-1}, such that $\langle N^{-1} N \rangle_p = 1$, can only be guaranteed if N is realitively prime to p. Since one of the chosen moduli is $p=2^n$, multiplicative inverse of 2 cannot be guaranteed to exist. Therefore, equation 4 cannot be interpreted as the equation $\langle\langle 1/4 \rangle_{pi} \langle (x+y)^2 - (x-y)^2 \rangle_{pi} \rangle_{pi}$. The potential problem of dividing the sum of differences, found in equation 4, by 2, will be explicitly and efficiently treated for the first time later in this paper. For a 2^m word memory unit, the direct product architecture (ie.: xy) would limit the maximal moduli to be bounded by 2^n, $n=m/2$. In fact, this

claim can be extended to the case where $p = 2^n+1$ through use of the following modification. Observe that if $x_i = 0$, then it automatically follows that $\langle x_i y_i \rangle_{p_i} = 0$. Therefore, if $x_i = 0 \rightarrow 0_\Lambda 0 \ldots 0$ (which is detectable condition in that the (n+1)st bit and remaining n-bit block is zero ($0 \rightarrow 0_\Lambda 00 \ldots 0$)) the output register would be automatically cleared. Therefore, the lookup table need not be accessed for this case. Instead, the all zero n-bit portion of the table address, allocated to x_i, can be used to represent $x_i = 2^n$ where $x_i = 2^n \rightarrow 1_\Lambda 00 \ldots 0$ (see Figure 1). Here, the table would be programed to map y_i into $\langle 2^n y_i \rangle_{p_i}$ using only a 2^m word memory.

The memory requirements associated with the quarter-square multiplier are substantially less than those of direct mechanizations. First, it should be apparent that the integers s^+ and s^-, found in equation 5, are bounded from above by 2^{n+1}. Therefore, only a (n+1)-bit table addressing space is required to realize $(s^{\pm})$ versus the 2n-bit space needed for direct architectures. It would appear however, that there is an exception to this rule. Since one of the moduli chosen is $p = 2^n+1$. Here the maximal value of s^+(or s^-) is 2^{n+1} which would technically require a (n+2)-bit address. However, by using the protocol found in Figure 2, which is an adaptation of the network found in Figure 1, the table size can be reduced to 2^{n+1} words for all moduli. Here, the overflow bit serves to differentiate $s^{\pm}=0$ from 2^{n+1}.

The quarter-squared architecture is abstracted in Figure 3. It uses a 2^{n+1} word high-speed memory for modular arithmetic lookup operations. Using, for example, the previously referenced 4K-30ns device, moduli having an 11-bit dynamic range (vs. 6-bit in the direct form) can be mechanized. This would yield a three-moduli dynamic range on the order of $2^{3(11)} \simeq 8.6 \cdot 10^9$. That is, without an increase in memory size (and therefore access time), the

dynamic range of the quarter-squared is $2^{33}/2^{18} = 2^{15}$ times larger than that obtainable through direct means! This large increase in dynamic range makes the RNS a viable alternative to traditional filter design methods. Both improved precision and throughput (through the reduction or absence of traditional scaling operations) can be achieved.

Several versions of the multiplier algorithm can be considered. They are summarized in Figure 4. The first, called the sequential form, would have an estimated throughput rate of 240 ns based on a 60 ns lookahead adder and memory having an access time of 30 ns with a cycle time of 60 ns. The second architecture, called the parallel form, would run at a 180 ns rate. The parallel architecture is preferred because its higher speed, simpler control. A 60 ns pipelined execution rate can be purchased at a small hardware cost.

Example: $p = 2^{11} = 2048$, $x = 1040$, $y = 352$, then

$z = \langle xy \rangle_p = 1536$

$A1{:}s^+ = 1376$; $\phi(s^+) = \langle 484416 \rangle_p = 1088$;

$A1{:}s^- = 688$; $\phi(s^-) = \langle 118336 \rangle_p = 1600$;

$A2 = \langle \phi(s^+) - \phi(s^-) \rangle_p = \langle -512 \rangle_p = 1536$

//

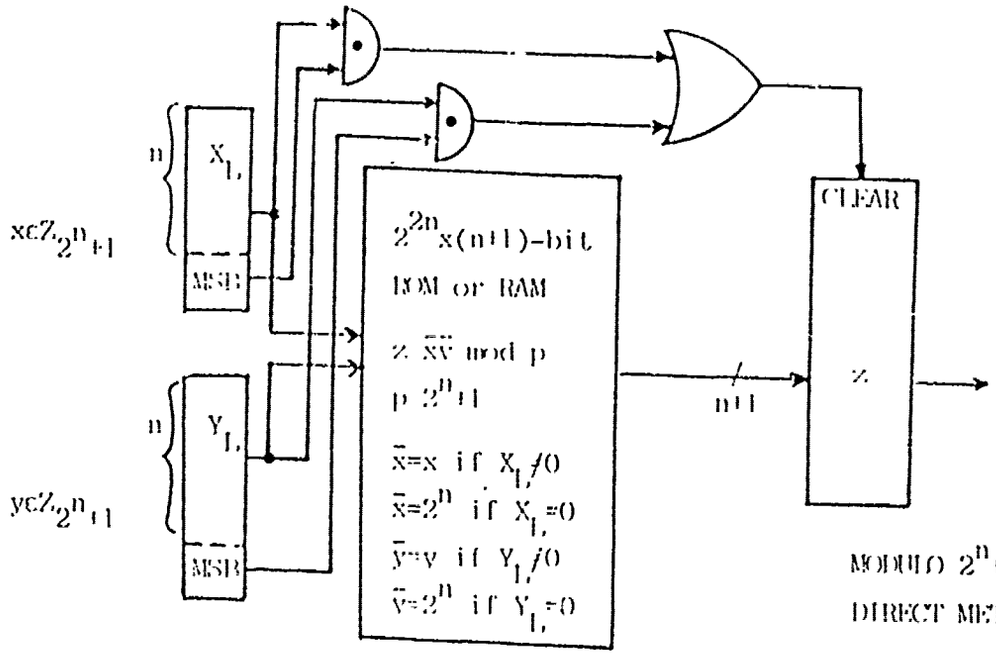
Upon closer investigation of the table lookup data base, a potential nuisance can be found. It can be examplified by observing that if $s^{\pm} = 9$, $p = 32$, then $\phi(s^{\pm}) = \langle 9^2/4 \rangle_{32} = 20.25$. Therefore, it may be required that two additional fractional bits may need to be added to the table's output wordlength. However, this is not the case as suggested by the following theorem:

Theorem: Let $[\![v]\!]$ denote the integer value of v. Then $z = \langle [\![\phi(s^+)]\!] - [\![\phi(s^-)]\!] \rangle_p$. That is, only the integer value of ϕ need be used and the fractional bits of $\phi(s^{\pm})$ can be ignored.

Proof: For x, y and k integers, one may define two rational numbers, namely $(x+y)/2 \triangleq v+k/2$; $(x-y)/2 \triangleq q+b/2$ where k = 0 or 1. Then $z = \langle\langle(x+y)^2/4\rangle_p - \langle(x-y)^2/4\rangle_p\rangle_p = \langle\langle v+kv+b^2/4\rangle_p - \langle q+dv+k^2/4\rangle_p\rangle_p = \langle\langle v+kv\rangle_p + k^2/4-q+k\rangle_p - b^2/4\rangle_p = \langle\phi(s^+) - \phi(s^-)\rangle_p$.

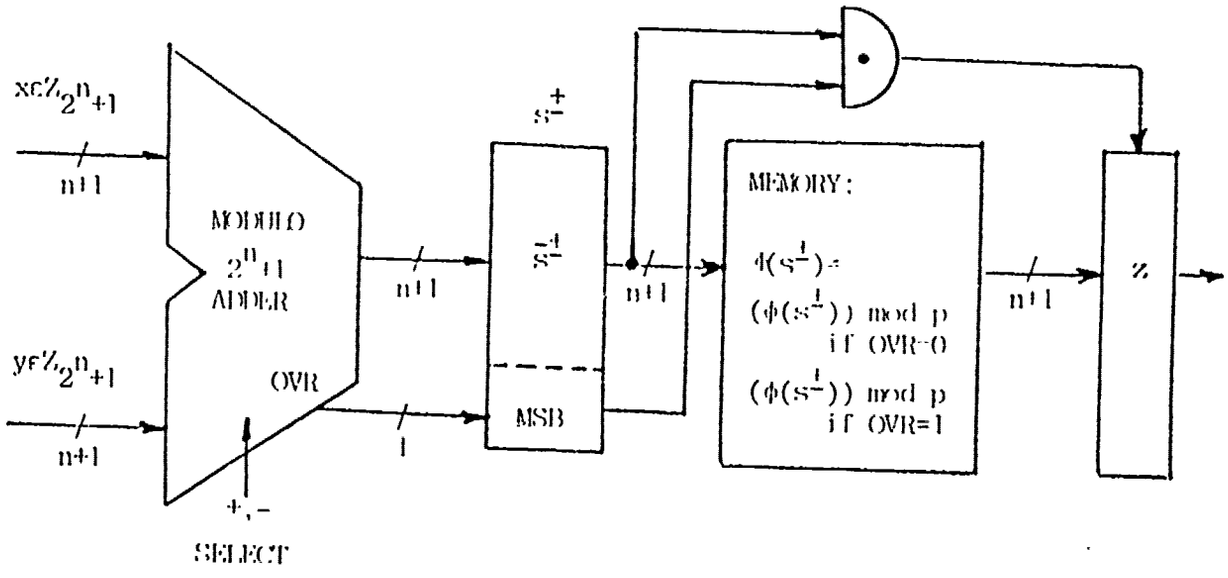
As a result, the parallel architecture is equivalent to that shown in Figure 5. Furthermore, by deriving the above theorem over a rational field, and showing that the results pertain to the integers, several classical problems are overcome:

1. The quarter-squared multiplier is not restricted to the Galois fields suggested by Pollard.
2. The question of the existence of the multiplicative inverse of 4 is now moot.



MODULO 2^n+1 ALU
DIRECT METHOD

Figure 1.



MEMORY COMPRESSION FOR $s^{\pm}$

Figure 2

Modulo p Adder

The quarter-square multiplier requires a modulo p adder be used to combine the two component parts of the solution (namely $\phi(s^+)$ and $\phi(s^-)$). Modulo p adders pose an interesting design problem. Unless a fast modulo p adder can be fabricated, the overhead associated with addition will offset any gain in throughput achieved through table lookups. For the moduli chosen, 2^n-1, 2^n, and 2^n+1, only the modulo 2^n adder can be realized directly (n-bit adder with ignored overflow). It would however, be desirable to use a n-bit adder to realize the modulo 2^n-1 and 2^n+1 adder as well. For the purpose of clarity, let s be defined to be the sum of $\phi(s^+)$ and $\phi(s^-)$. The following observation then follows:

TABLE 1

Case	Dynamic Integer Range of S	Modulo $\langle s \rangle_{2^N}$	2^N Adder OVF-BIT	Modulo p_i	p_i Adder $\langle s \rangle_{p_i}$	Example: N=3 s	$\langle s \rangle_{p_i}$
1	$s=0$	0	0	2^N-1	0	0	0
2	$1 \leq s \leq 2^N-2$	s	0	2^N-1	s	4	4
3	$s=2^{N-1}$	s	0	2^N-1	0	7	0
4	$s=2^N$	0	1	2^N-1	$s-2^N+1$	8	1
5	$2^N+1 \leq s \leq 2^N-4$	$s-2^N$	1	2^N-1	$s-2^N+1$	10	3
6	$s=0$	0	0	2^N	0	0	0
7	$1 \leq s \leq 2^N-1$	s	0	2^N	s	4	4
8	$s=2^N$	0	1	2^N	0	8	0
9	$2^N+1 \leq s \leq 2^N-2$	$s-2^N$	1	2^N	$s-2^N$	10	2
10	$s=0$	0	0	2^N+1	0	0	0
11	$1 \leq s \leq 2^N-1$	s	0	2^N+1	s	4	4
12	$s=2^N$	0	1	2^N+1	s	8	8
13	$2^N+1 \leq s \leq 2^{N+1}-1$	$s-2^N$	1	2^N+1	$s-2^N-1$	10	1
14	$s=2^{N+1}$ (special case)	0	0	2^N+1	$s-2^N-1$	16	7

Using n-bit AND gates to sense the zero condition of $\langle s \rangle_{2^N}$, the overflow bit OVF the sign bits of $\phi(s^+)$ and $\phi(s^-)$, combinational logic can be defined which will $\langle s \rangle_{2^N}$ into $\langle s \rangle_{p_i}$. It can be noted from the data found in Table 1 that the mapping requirements are:

1. for $p = 2^N-1$, map s to s or $s-2^N+1 = \langle\langle s \rangle_{2^N+1}\rangle_{2^N}$
2. for $p = 2^N$, map s to $s-2^N = \langle s \rangle_{2^N}$
3. for $p = 2^N+1$, map s to s or $s-2^N-1 = \langle\langle s \rangle_{2^N-1}\rangle_{2^N}$

Mapping two is trivially satisfied with an n-bit adder. The other two mappings require that s remains unchanged or it is decremented or incremented by unity. There are several ways to approach this problem. Bioul , Davis, and Quisquater have presented an unorthodox architecture for a modulo (2^n-1) adder using two-input gates[12]. Modulo $(2^n \pm 1)$ adders can also be realized through the use of end-around-carries. However, compared to modulo 2^n addition, this approach would almost double the addition delay. This extended delay problem can be overcome through added complexity (ie: time multiplexing two end-around-carry adders). Mapping one and three can be efficiently realized in the manner suggested by the example found in Appendix A. The functional operation of adding one (mapping 1) or subtracting one (mapping 3) from the output of an n-bit adder is performed by a PLA. The PLA will provide an overlay mask which accomplishes the required task. The derivation and utility of the mask can be understood in the context of the following example. Example: Suppose s is an 11-bit word having a decimal value of $s_{10} = 92$ or $s_2 = 00001011100$. If $s_{10}-1 = 91$ or $(s_{10}-1)_2 = 00001011011$ is desired, one notes that only the 3-LSB's of s_2 need be altered. In general, for n=12, only the following 13 distinct binary masks are required to form $(s_{10}-1)_2$.

MSB Pattern LSB	Notation
X X X X X X X X X X X	X = leave corresponding bit
X X X X X X X X X X 0	location of s_2 unchanged 1(or 0)
X X X X X X X X X 0 1	= change corresponding bit
⋮ ⋮	location of s_2 to 1 (or 0)
X 0 1 1 1 1 1 1 1 1 1	
0 1 1 1 1 1 1 1 1 1 1	Table II. MASK

Suppose the moduli $p = 2^n+1$, n = 12, is to be implemented. By using two commercially available 16x9 PLA's in parallel, the 12-bit output of an n-bit adder (shown as $\langle s \rangle_{2^N}$ in Table I) and the four priviously specified control bits, can be converted to 13-bit mask. The mask would transform the output of a high-speed n-bit adder to s or $s-2^n-1$, depending on the state of the 4 control bits. Based on a 25-ns 12-bit Schottky lookahead adder, a 20-ns 16x9 PLA, and 10-ns FET mask switches (in notation comments of Table II) a 65-ns modulo p adder, for $p = 2^n-1$, 2^n, and 2^n+1 can be realized. The presence of a 65-ns modulo p adder will now allow a 140-ns large moduli residue multiplier based on 35-ns 4Kx1 HMOS memory units. (See Figure 5) For a moduli set $\{2^{12}-1, 2^{12}, 2^{12}+1\}$, a fixed point multiplier, having an output dynamic range of $2^{36} - 2^{12}$, can thus be fabricated having a word rate of 7.143 M multiplications per second. This compares favorably with new 16x16 VLSI multipliers. Using a pipelined architecture, which requires the insertion of the storage registers found in Figure 5, a very impressive throughput figure of 28.5M multiplications per second. It is important, and fortunate to realize that the Itel HMOS memory unit, used in this analysis, has a cycle time equal to the access time. If, as is often found in practice, a memory unit has a cycle time approximately twice the access time, then pipeline delay would increase from 35-ns to 70-ns.

Summary:

The residue number system offers the potential for high speed parallel arithmetic. This class of arithmetic has been demonstrated to be useful in designing recursive algorithms, transforms, and digital filters. One of the principal limitations to its use is its limited practical dynamic range. To overcome this problem, a large moduli multiplier, for the moduli set $(2^n-1, 2^n, 2^n+1)$, was designed. This high-speed large moduli system was the product of the novel algorithm and new technologies (RAM and PLA's). The practical residue multiplier is capable of supporting a pipelined execution rate of 28.5 M multipliers per second.

Lastly, the performance of the residue multiplier is noted to be technology dependent. As memory densities increase and speed improve, the multiplier performance will directly benefit. As a result, the higher speeds associated with the next generation of submicron technology devices can provide a speed-up of two to five. In the more distant future, when and if the Josephsen technology becomes a viable design tool, residue multiplication rates, using the proposed methodology, may approach 500M multiplication per second.

References

1. G.A. Jullien, "Residue Number Scaling and Other Operations Using ROM Arrays," IEEE Trans. Computers, Vol C-27, No. 4, pp 325-337, April 1978.

2. C.H. Huang and F.J. Taylor, "Memory Compression Scheme For Modular Arithmetic, to be published, IEEE ASSP.

3. M.A. Soderstrand, "A High Speed Low-Cost Recursive Filter Using Residue Arithmetic," Proc. IEEE, Vol. 65, No. 7, July 1977.

4. N.S. Szabo and R.I. Tanaka, Residue Arithmetic and Its Applications To Computer Technology, McGraw-Hill, 1967.

5. W.K. Jenkins and B.J. Leon, "The Use Of Residue Number System in the Design of Finite Impulse Response Filters," IEEE C&S, CA5-24, April 1977.

6. W.K. Jenkins, "Techniques for Residue-to-Analog Conversion for Residue-Encoded Digital Filters," IEEE C&S, CA5-25, July 1978.

7. A. Peled and B. Liu, "A New Hardware Realization of Digital Filters," IEEE ASSP, ASSP-22, December 1974.

8. W.J. Jenkins, "A Highly Efficient Residue-Combinatorial Architecture for Digital Filters," Proc. of the IEEE (Letter), Vol. 66, No. 6, June 1978, pp 700-702.

9. J.M. Pollard, "Implementation of Number-Theoretic Transforms," Electronics Letters, Vol. 12, No 22, July 1976, pp 378-379.

10. H. Nussbaumer, "Digital Filters Using Read-Only Memories," Electronics Letters, Vol. 11, 1976, pp 294-295.

11. M.A. Soderstrand and E.L. Fields, "Multiplier for Residue-Number-Arithmetic Digital Filters," Electronic Letters, Vol. 13, No 6, March 17, 1977.

12. G. Bioul, M. Davis, and J.J. Quisquarter," A Computation Scheme for an Adder Modulo (2^n-1)," Digital Process, Georgi Publisher, Switzerland, Nol 1, (1975), pp 309-318.

APPENDIX A:

An example of a PLA controlled 2^n+1 adder, for n=3, is diagrammed in Figure A.1. In this figure, the sum A=5 and B=5 modulo (2^n+1) (ie: (5+6) modulo 9=2) is outlined. Also, the addition delay for n=12, based on commercially available hardware, is computed to be 40+20+5=65nsec. The general architecture of the adder is diagrammed in Figure A.2.

FIGURE CAPTIONS:

Figure 1: Modulo 2^n+1 ALU

Figure 2: Memory Compression for S_{ij}^{+}

Figure 3: Modulo p_i Multiplier

Figure 4: Architectures

Figure 5: Large Moduli Multiplier

Figure A.1: Example Problem

Figure A.2: General Architecture

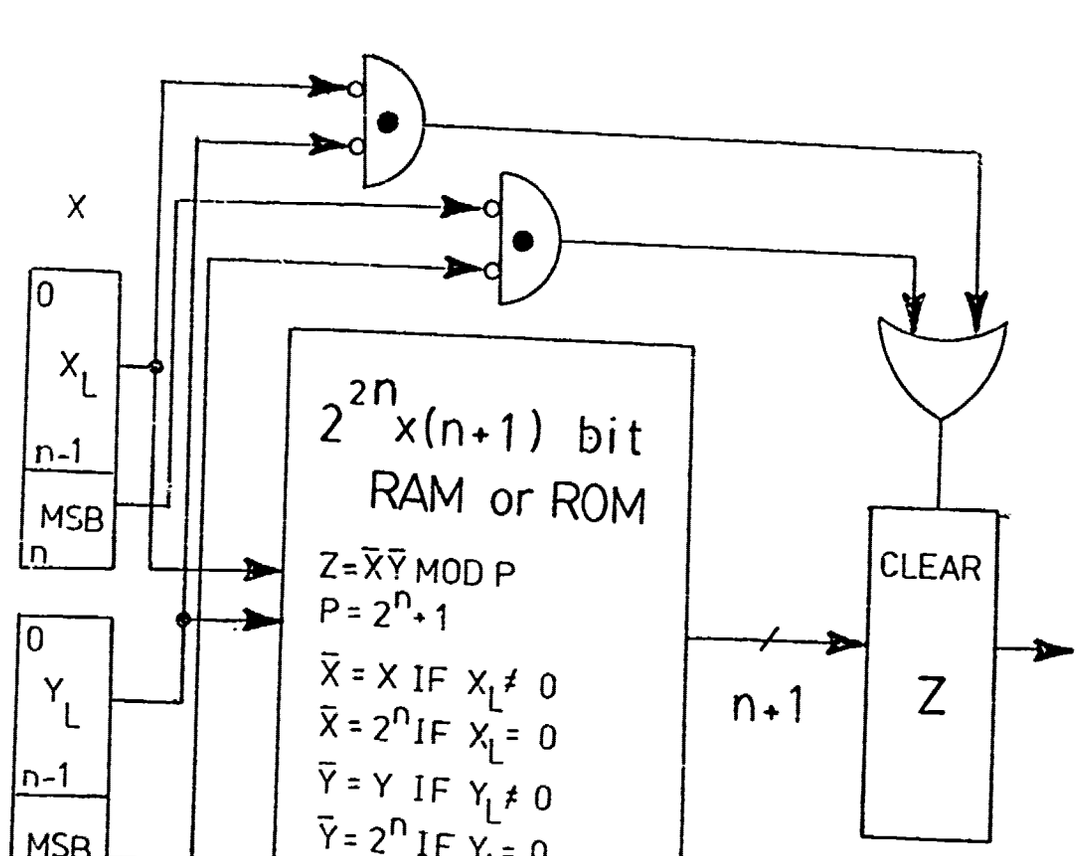


Figure 1. Modulo 2^n+1 ALU

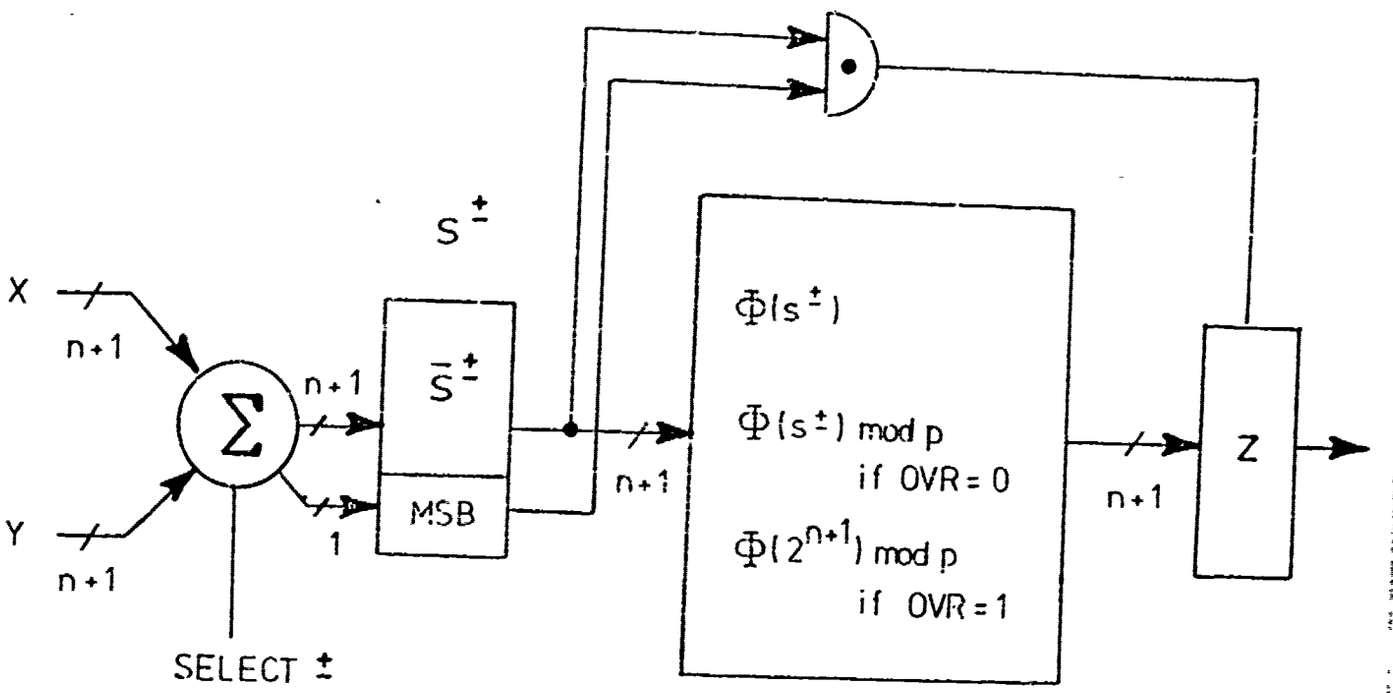


Figure 2. Memory Compression For $s^{\pm}$

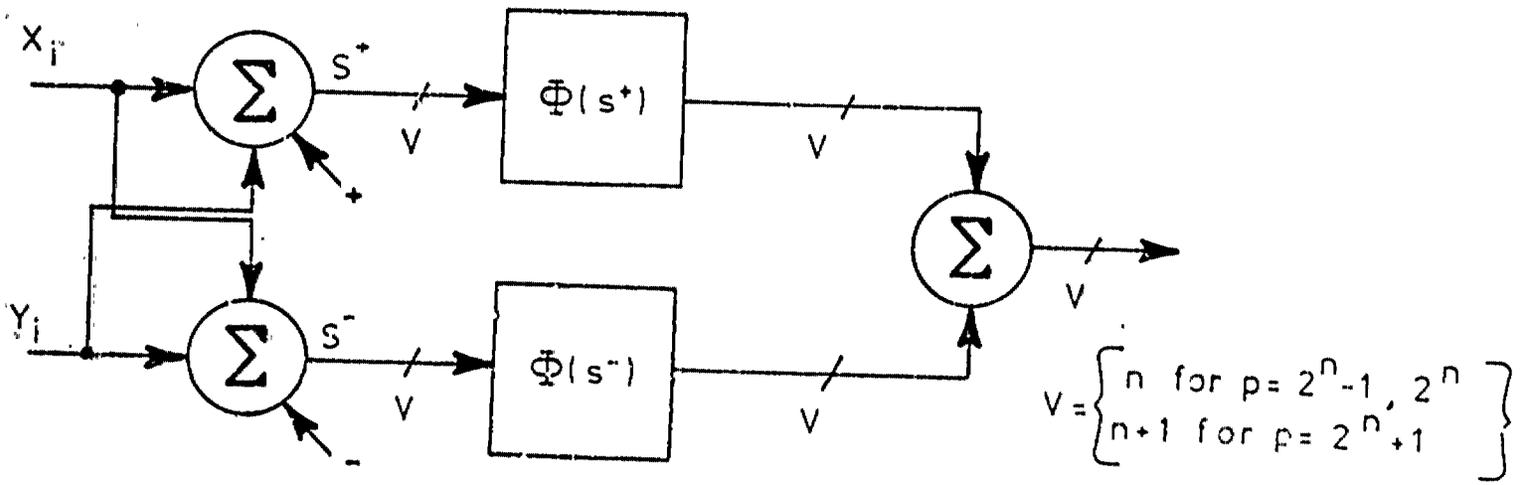


Figure 3. Modulo p_i Multiplier

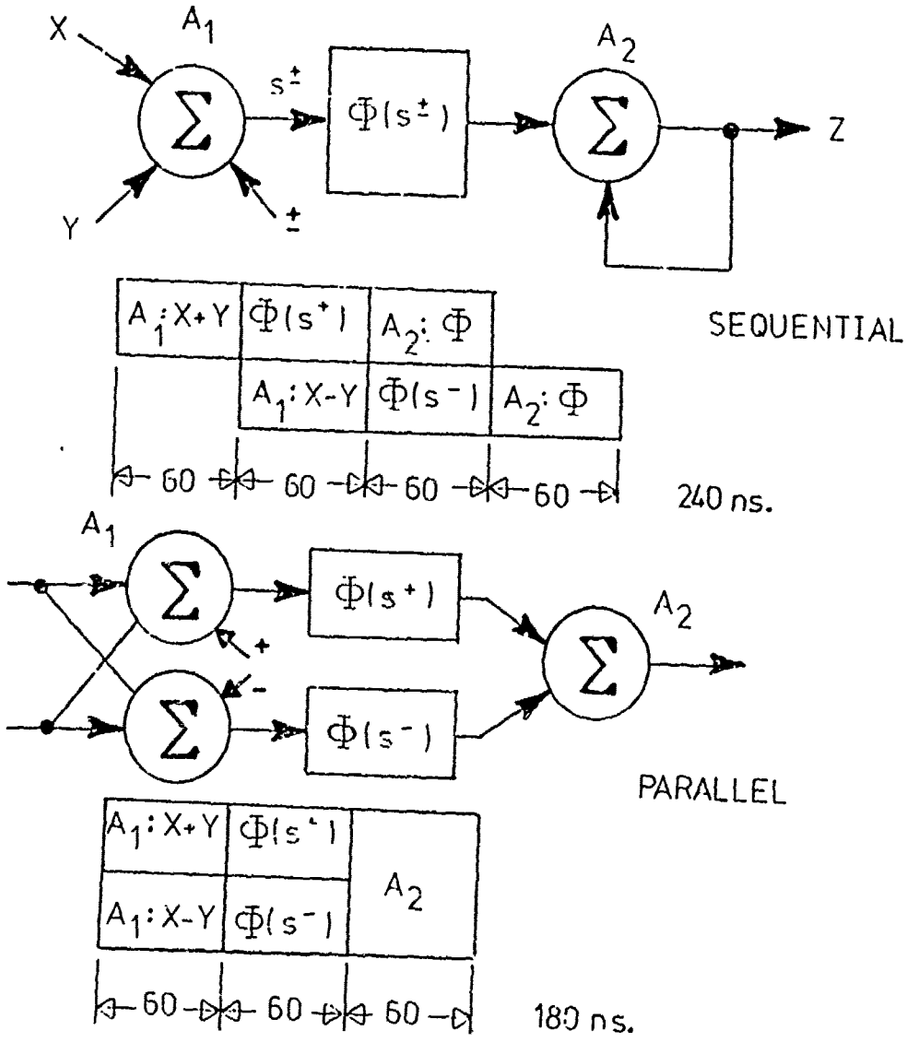


Figure 1. Architectures

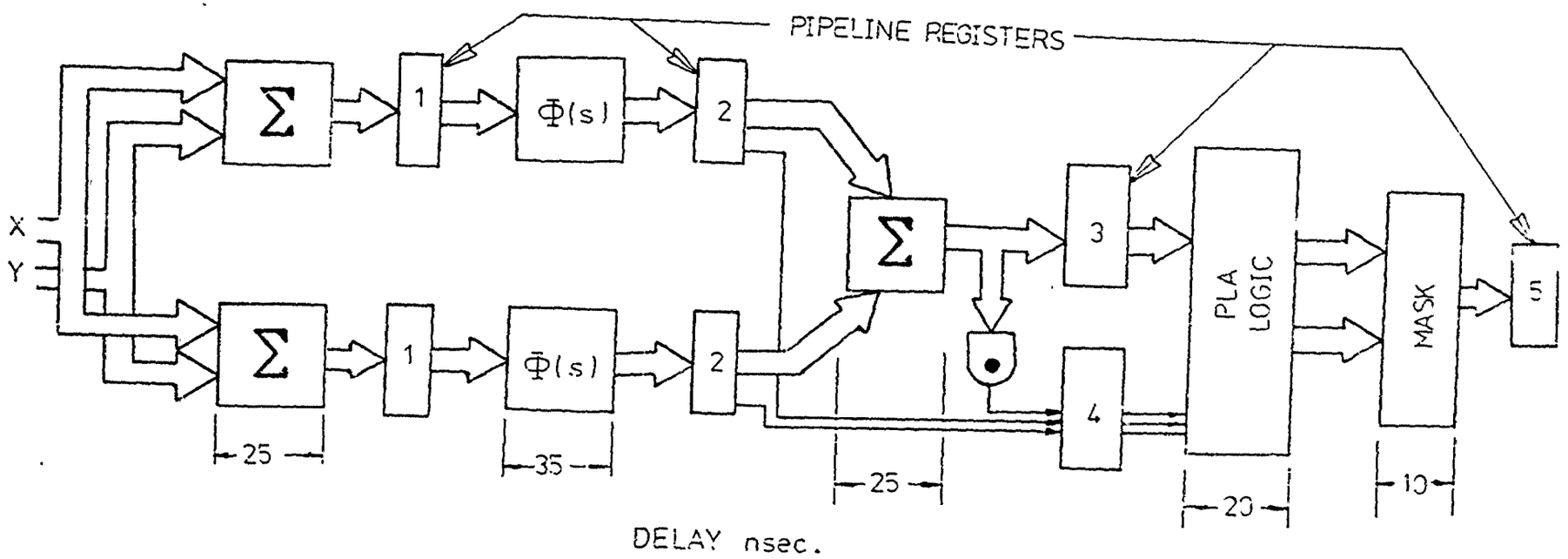


Figure 5. Large Moduli Multipliers

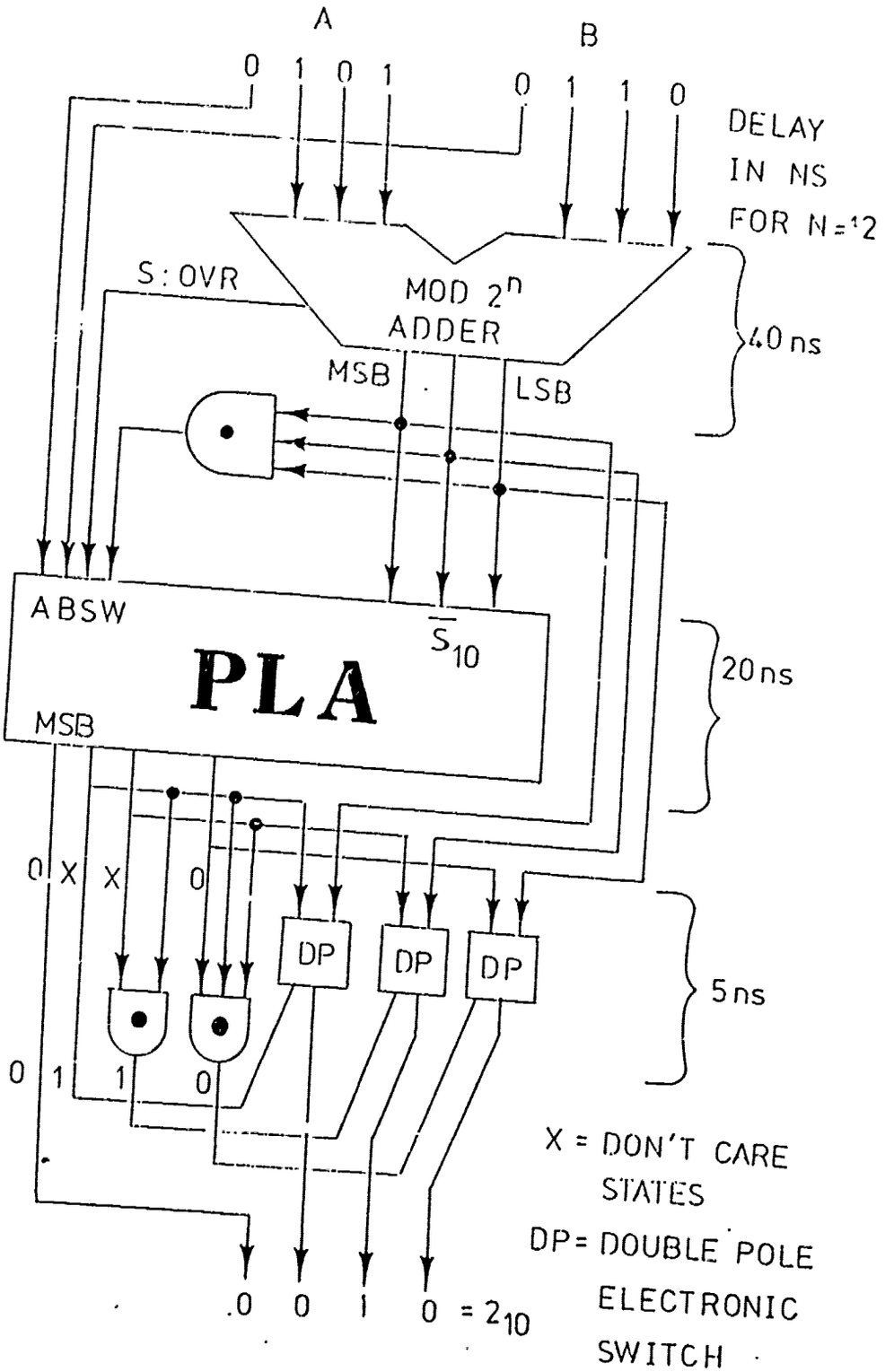


Figure A.1: Example Problem

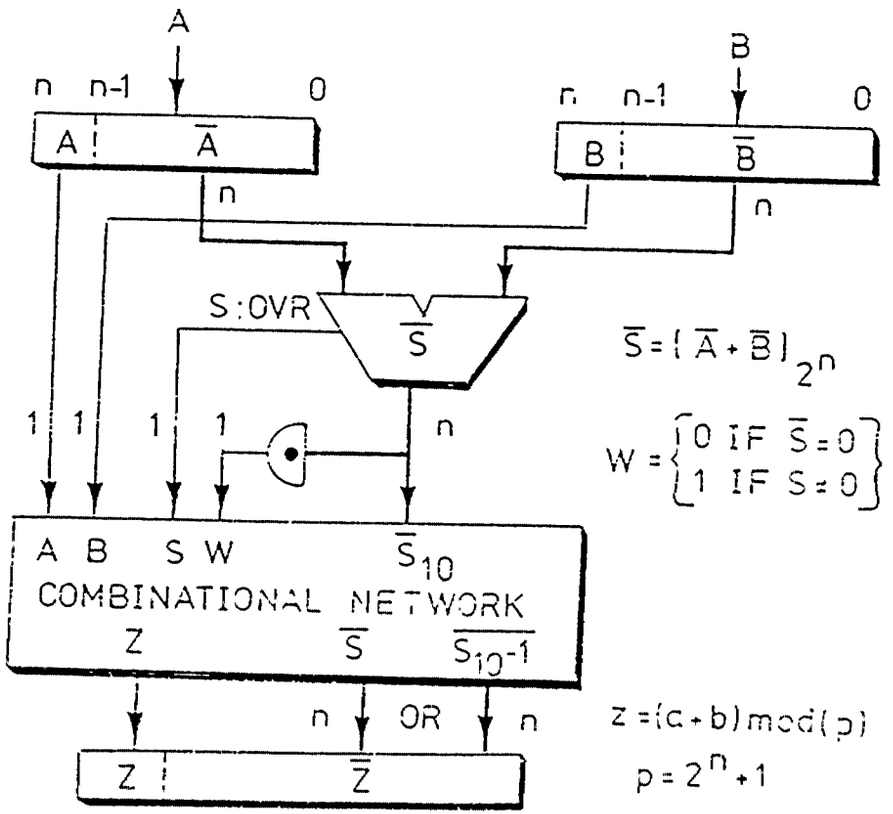


Figure A.2: General Architecture