1988

VLSI fault-tolerant systolic architectures.

Majid. Taheri
University of Windsor

VLSI FAULT-TOLERANT SYSTOLIC ARCHITECTURES

By

Majid Taheri

A Dissertation
Submitted to the
Faculty of Graduate Studies and Research
through the Department of
Electrical Engineering in partial fulfillment
of the requirements for the Degree of
Doctor of Philosophy
at the University of Windsor

Windsor, Ontario, Canada

1988

ISBN 0-315-43775-8

ABSTRACT

This work presents systolic architectures for implementing finite rings and fields operations in VLSI. By decomposing these operations, at the bit level, a generic cell, consisting of a small ROM and ancillary circuitry, can be used to compute most of the commonly used digital signal processing algorithms.

Various implementations for modular addition and multiplication, based on this cell, are given and compared to conventional single ROM techniques. The advantages of the new method, from the point of view of saving memory locations, silicon area, and increasing throughput are discussed.

The application of the technique in the area of Digital Signal Processing has been investigated. Structures for implementing Finite Impulse Response (FIR) filters, both bit serial and bit parallel, are presented. The design of fixed coefficient FIR filters implemented using Residue Number Systems (RNS) and binary are compared, and it is shown that the RNS approach requires less silicon area and operates at very high speeds.

A novel distributed fault detection technique has been introduced. A small amount of extra hardware enables the cell to detect run time faults. It is shown that only one redundant modulus is required to correct errors when this distributed fault detection is used, as opposed to two moduli in conventional Redundant Residue Number Systems (RRNS). The issue of the testability of the fabricated designs has also been addressed, and a simple procedure for testing arrays of these generic cells is proposed which takes

advantage of the contents of the ROMs and the capability of by-passing sub-sections of the array.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Dr. G.A. Jullien for his invaluable advice, guidance and constant encouragement throughout the progress of this thesis. The advice of Dr. W.C. Miller, Dr. S. Bandyopadhyay is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

gcd (a,b)	greatest common divisors of a, and b
$\lvert a^{-1} \rvert_m$	multiplicative inverse of a modulo m
L	number of moduli
$\lvert x \rvert_{m_i}$	the residue of x modulo m_i
B	binary operation on groups
e	is an identity element
$<G_a, B>$	group with operation B
$< R, B,] >$	ring with operations B,]
R(m)	ring with modulo m addition and multiplication
$\oplus_m$	addition modulo m
$\otimes_m$	multiplication modulo m
$\sum_m$	summation modulo m
$\lfloor . \rfloor$	round down operation
$\lceil . \rceil$	round up operation
$\overline{X}$	The additive inverse of X
M	dynamic range
M_T	total dynamic range
B	number of bits in m
r	number of redundant moduli
$y^{[i]}$	ith bit in y
$a \vee b$	logical OR of a and b; a, and b $\in \{0, 1\}$

CHAPTER 1

Introduction

Over the past four decades the computer industry has experienced four generations of development, physically marked by the rapid evolution of building blocks from relays and vacuum tubes, to discrete diodes and transistors, to small and medium-scale integrated circuits, and, finally, to large and very-large-scale integrated devices. An increase of many orders of magnitude of device speeds, together with dramatic reductions in hardware cost and physical size, have greatly enhanced computer performances. Constant demand for higher performance architectures, however, has accelerated the need for processing systems that are radical departures from the classical von Neumann architecture, or control flow computer, which has dominated computer design over the last four decades .

There are two main limitations to the von Neumann architecture which must be overcome:

A. The sequential nature of such machines is a major bottleneck to high speed processing and places a basic limit on their operating capabilities. The purely sequential nature of most conventional computing systems also imposes certain constraints on the associated algorithms and programs used.

B. In many typical computers, processors are separated from memory by long communication paths. Ultimately, the computation speed of such a machine will be dominated by the time taken to communicate information between the processor and the memory rather than the intrinsic switching speeds of logic devices. The full potential of future high speed switching technologies will therefore not be realized unless computational circuits are developed in which communication paths are short.

The limitations described above have been recognized for some twenty years or more and have stimulated research on novel forms of computer architecture. As Kuhn points out [Kuh'81]:

> *" Whenever a computer designer has reached for a level of performance beyond that provided by his contemporary technology, parallel processing has been his apprentice."*

Parallelism in computing systems is simply simultaneous processing of more than one task at a time, and clearly there is no limit in principle to the number of concurrent actions. Thus, parallelism has the potential of offering an arbitrary degree of improvement in computing speed. The fundamental argument is that, if a job can be broken down into N independent sub-tasks, then up to an N fold increase in computation speed may be obtained by allowing these sub-tasks to be processed on N elements running concurrently. More information on the background and the subject of parallel computers, is available in the text by Hwang and Briggs [Hwa'84], Kuhn [Kuh'81], and Zakharov [Zak'84] and their references.

Parallel computation is not a new idea and it seems to have occurred, in one form or another, to nearly all the pioneers of computing from the earliest times. The idea of parallelism certainly appears to have been considered by John von Neumann who was also the major pioneer in parallel computing via his research on array computers or cellular automata. It is important to note that there have been a large number of good ideas in parallel processing that could not be implemented in the past, or only implemented with poor effectiveness, because of the difficulties associated with hardware limitations.

It is only with the advent of VLSI and the promise of cheap processing power that the subject of Parallel Processing has really come of age. VLSI provides the medium through which many, previously conceptual, ideas can be implemented economically. With this advance, it has been realized that methods of exploiting the potentials are not well understood and that we are only beginning to appreciate the constraints imposed by the technology.

The digital microelectronic technologies with the highest complexity also have definite performance limitation[illegible]he fundamental limitation is the high cost of communication, relative to logic and storage. Communication is expensive in chip area, most of the area of a chip is covered with wires on several levels with transistor switches rarely taking more than five percent of the area on the lowest level [Sei'84]. When it comes to performance, communication is expensive in delay where the non zero resistance of the wires, together with the parasitic distributed capacitance, imposes a delay in the wire itself that is becoming increasingly significant at smaller geometries. Communication is also expensive in sending signals between chips, when package pin limitation, the area used for bonding pads and pad drivers are considered.

Finally, the dynamic power supplied to the chip, and dissipated in the circuit that switch capacitive signal nodes, is typically dominated by the parasitic capacitance of the internal wires, bonding pads, and inter chip wires, rather than by the capacitance of the transistor gates.

Thus both the cost and performance matrices of VLSI favour architectures in which communication is localized. This principle of locality is seen at every level of VLSI design.

In VLSI, there is an emphasis on keeping the overall architecture as regular and modular as possible, thus reducing the overall complexity. If a structure can truly be decomposed into a few types of simple substructures or building blocks great saving can be achieved. This is specially true for VLSI design where a single chip comprises hundreds of thousands of components.

Homogeneous machines are certainly simpler to design; one merely patterns the layout for one of these processing elements and replicates this pattern appropriately. Such iterative layout patterns have been applied in the past for silicon memories, registers, and array multipliers. Similar techniques also have been used in constructing large software systems.

A "good" VLSI architecture in this context should possess one or more of the following properties [Fos'80]:

A. The architecture is implementable by a few different types of simple cells.

B. The architecture's data and control path are simple and regular, so that the cells may be connected by a network with local and regular interconnections.

C. The architecture uses extensive pipelining and multiprocessing. In this way a large number of cells are active at one time so that the overall computational rate of the simple cell is high.

Under certain circumstances, mostly in real-time application areas, the supervisory overhead incurred in general-purpose computers often makes them too slow and expensive for real time signal and image processing. To achieve a throughput rate adequate for these applications, the only feasible alternative appears to be massively concurrent processing by special-purpose hardware.

During the late 70's, H.T. Kung and his colleagues at Carnegie-Mellon University, observed that certain problems can be solved by means of a regular, locally interconnected array of identical Processing Elements (PEs) that repeatedly execute a set of identical instructions in a lock-step fashion. They used the term "systolic array" to describe such hardware architectures. These architectures capitalize on regular and modular structures that match the computational requirements of many algorithms. Much research has been done and much has been written about the design of algorithms and architectures suitable for such structures. The applications for which systolic designs have been proposed can be broadly categorized into, signal and image processing, matrix arithmetic, and non-numeric applications.

Digital signal and image processing encompasses a variety of mathematical and algorithmic techniques. Most signal and image processing algorithms are dominated by convolution and correlation filtering, and certain key linear algebraic methods. These algorithms possess properties such as regularity, recursiveness, and locality, and these properties can be exploited using systolic design concepts.

To synthesize a systolic array from the description of an algorithm, a designer needs an understanding and familiarity with the principles behind systolic design concepts, the application, the algorithm, and the technology. The design process is slow and error prone and normally the resulting designs are not guaranteed to be optimal. Progress has been made in the development of systematic design techniques to automate this process [For'85]. These techniques are not complete, however, and until more is learned about automating the design process there will continue to be need for 'hand-crafting' designs by skilled people. The basic operation performed in each cycle by each processing element in the various systolic arrays can range from a simple bit-slice operation [Mac'82], to word level [Cap'81], and even execution of a complete program [Fou'87].

During the last few years special attention have been paid to some aspects of the systolic design concept. These include array synchronization, testing of the fabricated chips, run time reliability, partitioning of large problem, and universal building blocks.

Synchronization:

In large synchronous systolic arrays, clock lines, the only global signals in true systolic designs, can introduce clock skew. Possible approaches that avoid the problem of clock skew include using efficient layout of the the clock distribution network, and unidirectional data flow. An alternative to the design of a globally synchronous array is to achieve a self timed system through the use of an asynchrounous handshaking mechanism established between the adjacent cells. This category of array designs are referred to as wavefront arrays [Kun'83].

Testing:

Even the most advanced manufacturing techniques may produce faulty chips. The yield figures drop rapidly as the number of devices increase. The increase in the ratio of logic to pins which drastically reduces the controllability and observability of the logic on the chip which makes fault diagnoses difficult. Testing of the existing complex chips are already causing difficulties, and it is expected to be more difficult with the higher complexity chips that are being proposed. Techniques for reducing the test time and simpler methods for generating test vectors applicable to special cases have been proposed in the past [Kor'86] but more is left to be done in this area.

Reliability:

Even satisfactorily tested chips may not be flawless, or they may become damaged during their operation. Since any functional error in a high performance system may seriously jeopardize the operation of the system, some level of fault tolerance must be incorporated. The problem of run time fault tolerance also requires more attention with the goal of maximizing the reliability while minimizing the corresponding overhead [Abr'86].

Partitioning of large problems:

State of the art technology allows only a limited number of processing elements, with respect to the size of large problems, to be fabricated on the same chip or wafer. The problem must be partitioned so that the same algorithm can be used to solve the smaller problems and so that an array of small, fixed size can be used [Mol'86]. A partitioned approach may also be needed to circumvent the I/O problem of the systolic arrays [Haw'84].

Universal building blocks:

The modular design of systolic arrays allows designers to rapidly prototype their designs using off-the-shelf devices, such as microprocessors. This has led to the development of "universal building blocks" that can be used for many systolic arrays. Commercially available chips worthy of consideration as basic modules include INMOS Transputer, TMS32010, and NEC dataflow chip μPD 7281 [For'87]. The problems involved with this approach include the development of programming tools and support for flexible interconnections.

It appears that designing a chip containing hundreds of thousands of devices to perform highly complex functions is not the sole goal any more. If testing of the chip becomes too expensive or the reliability of the system based on this chip is lower than an acceptable level, it would not have commercial success.

Let us first examine the problem of clock skew more closely. Consider a simple 1-D convolution which, in practice, maps onto a 2-D grid array with the carry and data information flowing in different directions. It is known that clocking a large 2-D array is a non trivial matter, while large linear arrays can be effectively synchronized, even in the presence of large signal propagation delays. As Fisher and H.T. Kung [Fis'85], pointed out:

> *" ... one-dimensional arrays can be clocked at a rate independent of their size under fairly robust assumption, while two-dimensional arrays and other graphs with similar properties cannot."*

A possible solution is to employ Residue Number Systems (RNS) to alleviate the carry flow across the second dimension. RNS naturally restricts the propagation of the carries within smaller computational rings rather than the entire dynamic range. Also, the fact that Redundant Residue Number Systems (RRNS) have properties useful for error detection, error correction, and fault tolerance in digital signal processing necessitates its thorough study for the implementation of suitable algorithms in VLSI. The following provides a brief background on RNS.

1.1. RNS and Signal Processing

The first published report on a residue based electronic computer, was that of Svoboda and Valach in Czechoslovakia [Svo'55]. This report described a hard-wired small-moduli, RNS computer used to study error codes. At approximately the same time, Garner [Gar'59], worked independently on the basic concepts of RNS arithmetic.

During 50's researchers became interested in fault tolerant properties of RNS which could improve the reliability of vacuum tube digital computers. But with the availability of transistors, more reliable digital computers, were designed. This resulted in a decline in the interest in fault tolerant residue arithmetic. Fortunately the interest did not disappear entirely, researchers developed the theory of RNS arithmetic and error detection and correction further in late 60's and early 70's [Bar,73], [Man'72]. The availability of inexpensive semiconductor chips brought more interested researchers into this field, a number of digital signal processors based on RNS were implemented [Jen'77], [Hua'81], [Nag'83], [Mil'84].

The VLSI era arrived in the 70's, but it took sometime before researchers, who were interested in RNS, started exploring the new media [Jen'82b], [Ban'86].

1.2. The Objectives and Review of the Research Work

There is a strong belief that RNS can alleviate some of the problems discussed earlier in this chapter. To prove this, we will start anew by taking a different approach, one that satisfies, in the first place, the requirements of "good" VLSI design. This is because mapping designs suitable for discrete components does not take into account the limitations of the technology and may create more problems than they can solve. Once this basic goal is achieved, the objective is to implement all aspects of digital signal processing algorithms, from binary to RNS encoding , the signal processing algorithm itself, and decoding back into binary.

Finally the capabilities of the new technique, in mitigating some of the problems related to VLSI systems, such as fault detection and correction, and testing will be studied.

1.3. Organization of Thesis

Chapter 2 covers the pre-requisite material for the research. In particular a concise review of some of the fundamentals on finite ring and field structure is presented and the mathematical concepts of Residue Number Systems

(RNS) are introduced. This chapter provides a background for the definitions and notations used through the thesis.

Chapter 3 describes our proposed architecture, a generic cell which can be used to perform finite ring addition, and multiplication operations. Comparison of two techniques for performing modular addition, the conventional approach and the proposed technique, is given. Examples of subtraction, multiplication, and binary to residue encoder operations based on this architecture are given.

Chapter 4 discusses the use of the new structure for Digital Signal Processing Applications. Different structures for implementation of Finite Impulse Response (FIR) filters-- both bit serial and bit parallel approaches are provided. A fixed coefficient FIR filter implemented using both, a very efficient bit sliced binary and the proposed approach in this chapter are compared. Finally the operation of residue to binary decoding and scaling are discussed and suitable structures are given.

Chapter 5 deals with the important issues of fault detection and correction using the cell discussed in previous chapters. It is shown that a small hardware overhead enables the cell to perform a concurrent fault detection operation in the structure. A simplified Redundant Residue Number System (RRNS) which uses one redundant modulus instead of two is discussed and used in a fault correcting system.

Enhancements in testability of the structures based on the new cell is also discussed in this chapter. It is shown that the ability to bypass the entire memory and a major part of the logic used in the cell simplifies testing the chip and shortens the time period required.

Chapter 6 summarizes the result of this research.

CHAPTER 2

DISCRETE MATHEMATICS AND THE RESIDUE NUMBER SYSTEMS

Abstract

This chapter provides the theoretical background required for the understanding of the work developed later in this dissertation. Some of the fundamental concepts of discrete mathematics are reviewed to the extent needed for appreciation of Residue Number Systems (RNS). A brief introduction to the RNS then follows.

2.1. Introduction

There is at present a growing body of opinion that in the decades ahead discrete mathematics will be of increasing importance. Certainly, one reason for this opinion is the rapid development of computer technology, and the use of discrete mathematics as one of its major tools. The increasing interest in the investigation of the theory of formal languages, computer cryptography, and theory of coding are some of the uses of discrete mathematical structures.

In this chapter we intend to present some of the preliminary concepts of discrete mathematics, the emphasis is on the basics which form the foundations of the work under taken in this thesis. A review of discrete mathematical structures of groups, rings, and fields, their properties and various ways of forming them are given. Residue number systems and certain RNS operations, the so called difficult operations, used in this work are explained.

The notation used throughout this dissertation will also be introduced. A notation already exists for residue number systems; however, for cascaded, mixed ring (field) operations, the notation can become confusing, with unwieldy nesting of modulo parentheses. We will use special symbols for operating over the different systems, so that such operations can be readily understood in an expression. Numerous references are provided for the interested reader to further explore in these topics.

2.2. Discrete Mathematical Structures

As a starting point, it would seem appropriate to formally define the meaning of a binary operation:

Definition

A binary operation, $+$, on a set is a rule (a method, a function, or a formula) that assigns to each ordered pair of elements of the set some element of the set.

Note that the word "ordered" is very important in this definition, because it allows the possibility that the element assigned to the pair (a,b) be different from the element assigned to the pair (b,a). Also this definition permits the result of the binary operation to be any one of the operands or a third element of the set (property of closure). Addition and multiplication on an infinite set of integers is an example of a binary operation.

Definition

A binary operation $+$ is said to be associative if and only if

$$(a + b) + c = a + (b + c)$$

and is commutative if and only if

$$a + b = b + a$$

for all a,b,c belonging to the set.

With the above definitions we can now review some of the algebraic structures which are used frequently in discrete mathematics. We start with the structure of a group, a term coined by Galois about 150 years ago to describe sets of one-to-one functions on finite sets that could be grouped together to form a closed set [Gal'86,34].

Definition

A group $<G, + >$ is a set G, together with a binary operation $+$ on G, such that the following axioms are satisfied:

g1. The binary operation is associative.

g2. There is an element e in G such that
$e + x = x + e = x$,
this element, e is an identity element for the binary operation on G.

g3. For each a in G, there is an element a' in G with the property that

$$a' + a = a + a' = e.$$

The element a' is the inverse of a with respect to binary operation $+$.

Note that the property of closure is implied as a consequence of the binary operation and therefore is not added to the above list of axioms.

If the binary operation is also commutative the group is named abelian. The set of all the integers with the operation of addition generates a group, while the same set and the operation of multiplication is not a group because not all the elements have inverses. The simple algebraic structure of the group enables us to solve all the linear equations of the form $a + x = b$.

The order of an element a in a group G is the smallest positive integer n such that $a^n = e$. The number of elements of a group (finite or infinite) is called its order, the symbol $|G|$ is normally used to denote the order of the group, G.

Definition

Let $<G_a, +^a>$ and $<G_b, +^b>$ be groups. A one to one onto (bijective) function $f: G_a \Rightarrow G_b$: with the property that for any two elements g_a^i and g_a^k in G_a

$$f(g_a^i +^a g_a^k) = f(g_a^i) +^b f(g_a^k)$$

is called an isomorphism from G_a to G_b [Pin'82,89]. For example, consider the groups defined by multiplication modulo 3, shown as $\otimes_3$, on $Z'_3 = Z_3 - \{0\}$ and addition modulo 2, shown as $\oplus_2$, on Z_2 as given in Fig (2.1).

	0	1
0	0	1
1	1	0

Table for addition mod 2

	0	1	2
0	0	1	2
1	0	$\boxed{1}$	$\boxed{2}$
2	0	$\boxed{2}$	$\boxed{1}$

Z'_3

Table for multiplication mod 3

Fig 2.1 Operation tables for $<Z_2, \oplus_2>$ and $<Z'_3, \otimes_3>$

It is simple to find the $f: Z'_3 \Rightarrow Z_2$ for which $f(2)=1$ and $f(1)=0$ maps $<Z'_3, \otimes_3>$ isomorphicaly into $<Z_2, \oplus_2>$.

When the group contains an element g which generates all the group's elements as given here

$$G = \{g^n ; n \in Z\}$$

then g is said to be a generator and the group G is a cyclic group.

It can be proven that the set {0,1,2,3,...,n-1} is a cyclic group Z_n of elements under addition modulo n.

Next we will discuss the algebraic structure of a ring. The term "ring" was coined by the German mathematician David Hilbert, in the late 19 century, to define a set of elements with two binary operations.

Definition

A ring $< R, +, * >$ is a set R together with two binary operations $+$ and $*$, (normally called addition and multiplication), defined on R such that the following axioms are satisfied:

r1. $<R, +>$ is an abelian group.

r2 $*$ or multiplication is associative.

r3. For all $a,b,c \in R$ the distributive law holds.

$$a * (b + c) = (a * b) + (a * c)$$

$$(a + b) * c = (a * c) + (b * c)$$

For example the set of complex numbers, C, together with the operations of addition and multiplication is a ring. Another appropriate example of the rings is the set of Z_n={0,1,2,...,n-1} and the operation of addition and multiplication modulo n.

Note that the operation of $*$, is considerably weaker than the operation of $+$ in the rings since the operation does not have to be commutative and multiplicative identity need not exist.

Groups and rings can be built in many different ways, as an example assume the group structures

$$G_a = < (g_a^0, g_a^1, .g_a^{(a-1)}; +^a> \qquad \text{and} \quad G_b = < (g_b^0, g_b^1, .g_b^{(b-1)}; +^b>$$

it is proven that the structure

$$\tilde{G} = < (g_a^0, g_b^0), (g_a^0, g_b^1), (g_a^0, g_b^1), ...(g_a^{(a-1)}, g_b^{(b-1)}); \tilde{+} >$$

is a group with order $|\tilde{G}| = ab$, under the operation $\tilde{+}$ defined as follows

$$(g_a^j, g_b^k) \tilde{+} (g_a^p, g_b^q)) = (g_a^j +^a g_a^p, g_b^k +^b g_b^q))$$

The group, $\tilde{G}$, formed as above, is said to be the direct product of G_a and G_b shown as $G_a \times G_b$.

The same technique can be utilized to create new rings. Suppose we have the two rings $<R_a; +^a, *^a>$ and $<R_b; +^b, *^b>$. Using a similar method the resultant ring is $<\tilde{R}; \tilde{+}, \tilde{*}>$, the elements of which are (r_a^i, r_b^k), $0 \leq i \leq a-1$ and $0 \leq k \leq b-1$ and the operations are defined as:

$$(r_a^i, r_b^k) \tilde{+} (r_a^p, r_b^q) = (r_a^i +^a r_a^p, r_b^k +^b r_b^q)$$

$$(r_a^i, r_b^k) \tilde{*} (r_a^p, r_b^q) = (r_a^i *^a r_a^p, r_b^k *^b r_b^q)$$

It seems logical to name the ring $\tilde{R}$ as constructed above as the direct product of rings R_a and R_b, shown as $R_a \times R_b$; however, this is not the case and the term direct sum is most often used†. This construction can be iterated any finite number of times to obtain the direct product (sum) of a finite number (n) of groups (rings).

Another interesting example of forming rings is to use the set of all polynomials in the indeterminate x with integer coefficients in ring R, this structure is a ring under polynomial addition and multiplication as described below.

Polynomial Rings:

Let R be a commutative ring with unity. Each expression of the form

$$a_n x^n + \ldots + a_2 x^2 + a_1 x^1 + a_0$$

is called a polynomial in x with coefficients, a_k, in ring R. Obviously there exists infinitely many such polynomials (even when ring R is finite).
The set of all these polynomials and the operations of addition and multiplication, defined for these polynomials, give algebraic structures known as polynomial rings, shown as R[x].

Polynomial ring addition is defined by combining the like terms binary operations of addition addition operation of the ring R. Multiplication is defined by combining the polynomial coefficients as performed in conventional polynomial multiplication, using the addition and multiplication over the ring R [Blo'87]. Ring isomorphisms are similar to

† In fact the term direct sum and direct product has been used interchangeably in different texts for the procedure explained above [Gal'86], [Pin'82], [Jac'51], [Sto'73], [Gil,76], [Gol'73].

group isomorphisms as defined below.

Definition

Let $<R_a, +^a, *^a>$ and $<R_b, +^b, *^b>$ be two rings. If there exists a one to one onto (bijective) function such as f for which

$$f(r_a^i +^a r_a^k) = f(r_a^i) +^b f(r_a^k)$$

$$f(r_a^i *^a r_a^k) = f(r_a^i) *^b f(r_a^k)$$

and the identities are the same $f(1^{R_a}) = 1^{R_b}$, then two rings are isomorphic rings [Gil'76,174], shown as $R_a \cong R_b$.

It is proven that the direct product (sum) of $R_a \times R_b$ is identical to R_{ab} to within isomorphism if and only if gcd (a,b) = 1 [Gil'76,177]. It is this theorem that enables us to slice a large ring into several smaller rings, the bases for building residue number systems.

A nonzero element of a commutative ring need not have a multiplicative inverse. If all the elements of a ring, except the null element, have multiplicative inverses, the structure is said to be a field. For example $<Q,+,x>$ is a field, where Q is the set of all rational numbers. It is proven that the ring Z_p and addition and multiplication modulo p is a field, when p is a prime, and is denoted as a Galios Field, GF(p), after the French mathematician Evariste Galois. GF(p) has exactly p elements {0,1,2,...,p-1}, and since there are infinitely many primes, this construction yields infinitely many finite fields.

Every finite field has at least one generator $g \in GF(p)$; such that the first (p-1) powers of g , together with 0, constitutes the complete set GF(p) of the field elements. For example with p=11, g=2 is a generator.

$$0,\ g^1=2,\ g^2=4,\ g^3=8,\ g^4=5,\ g^5=10$$
$$g^6=9,\ g^7=7,\ g^8=3,\ g^9=6,\ g^{10}=1$$

The exponents are referred to as indices, and can be viewed as a set of finite field logarithms. Suppose a and b are elements of GF(p) and $a=g^{\alpha}$, $b=g^{\beta}$,then

$$a \otimes_p b = g^{(\alpha\ \oplus_{p-1}\ \beta)}$$

This is a good example of the use of isomorphism between a multiplicative group g having elements $\{g_n\}=\{1, 2, 3,..., p-1\}$ with multiplication modulo p, and the additive group k having elements $\{k_n\}=\{1, 2, 3,..., p-2\}$. In chapter 3, we will present a structure which uses this property to multiply two numbers.

Instead of forming a field from Z_p, with p prime, one can take a ring of polynomials together with an irreducible polynomial where coefficients $\in Z_p$, and build an extension field. As an example let us take GF(2) and form a ring of polynomials with the coefficients in GF(2) and the irreducible polynomial as $p(x)=x^3+x+1$. Since p(x) is cubic, we can take as representatives of (mod p) equivalence classes the 8 polynomials of the form $a^2x^2+a^1x+a^0$, $a^i \in GF(2)$. The number of elements in these types of field is always p^n, where [McE'87,25]

$$n = \deg(p(x))$$

Addition in the extension field simply follows the addition rule discussed previously, while multiplication of two elements such as $a^2x^2+a^1x+a^0$ and

$a'^2x^2+a'^1x+a'^0$ generates x^4 and x^3 terms which in our example must be replaced by $x^3 = x+1$, and $x^4 = x^3+x$ in order to have the field properties preserved.

Rings and fields with modular addition and multiplication show special properties. Some of these properties were discussed above, in the following a brief discussion of a general class of modular systems formed by selecting several simple modular structures (either rings or fields) called residue number systems will be given.

The notations for ring with modular addition and modular multiplication are similar to that used so far in this chapter, only the modular ring will be given as a subscript of the operations. Thus $\oplus_m, \otimes_m, \sum_m$ indicates the addition, multiplication, and summation over the ring with modulus m.

2.3. Residue Number Systems

The roots of this number system go back almost 2000 years. The residue number system is a non weighted integer system which differs radically in many aspects from the more conventional weighted magnitude number systems, such as the binary and decimal, fixed radix systems.

Like many other number systems it has advantages and disadvantages. It is easy and efficient to add and multiply; however, not so easy to divide and compare. It is also easy to perform forward transformation (code) but more difficult to perform reverse transformation (decode). It has been shown that it is inefficient to build general purpose computers based on RNS [Sza'67], however some very efficient special purpose hardware has been reported using this system [Fou'82], [Nag'83], [Mil'84].

It is more appropriate and helpful to review RNS in terms of 2 classes: standard residue number systems (RNS) and redundant residue number systems (RRNS).

2.3.1. Standard Residue Number system (RNS)

As discussed in section 2.2, the direct sum of L rings, $(z_{m_1}, z_{m_2}, z_{m_3}, \ldots, z_{m_L})$ and Z_M are identical to within isomorphism if and only if

$$\gcd(m_i, m_k) = 1 ; \qquad \text{for } i,k \in \{1, 2, \ldots, L\} \text{ and } i \neq k$$

where

$$M = \prod_{i=1}^{L} m_i$$

This unique mapping between a large ring and several smaller rings is used to code and decode numbers represented.

The RNS is defined in terms of a set of L relatively prime moduli.

$$(m_1, m_2, m_3, \ldots, m_L)$$

where L is the number of moduli in the system and $\gcd(m_i, m_k) = 1$ for $i \neq k$, $i,k \in \{1,2,\ldots,L\}$. Any integer $X<M$ can be represented as an L-tuple

$$X = (x_1, x_2, x_3, \ldots, x_L)$$

where $x_i = X \bmod m_i = |X|_{m_i} \quad i \in \{1,2,\ldots,L\}$

The dynamic range is defined as the set of integers that can be encoded in the RNS system and is given by

$$M = \prod_{i=1}^{L} m_i$$

~~Let~~ $*$ represent the binary operation of addition, subtraction, or multiplication. If $Z = X * Y$, then we can easily show that

$$|Z|_{m_i} = z_i = x_i *_{m_i} y_i \tag{2.1}$$

This equation shows the independence of binary operations on different moduli. It is this property which allows very fast parallel implementation of addition and multiplication, in contrast with conventional fixed-radix representation where a carry or a borrow is passed between adjacent digits.

2.3.2. Signed Residue Number System

The dynamic range is partitioned into two portions the lower portion which represents the positive numbers and the upper portion representing negative numbers. Signed numbers can be introduced by defining a number X to be positive if

$$X < \lfloor M/2 \rfloor$$

and negative if

$$X \geq \lceil M/2 \rceil$$

where $\lfloor \alpha \rfloor$ and $\lceil \alpha \rceil$ are round down and round up functions respectively.

The additive inverse of X , $\overline{X}$ is defined by the relationship

$$X \oplus_M \overline{X} = 0$$

where $\overline{X} = (\overline{x}_1, \overline{x}_2, \ldots \overline{x}_L)$ with $\overline{x}_i = m_i - x_i$; for $i \in \{1,2,\ldots,L\}$

If we use the above definitions to represent a signed number system, then the operations defined by eqn (2.1), preserve the rule of signed arithmetic [Jul'78]. Thus, binary operations can be carried out between signed numbers without an explicit knowledge of the sign or magnitude of the two numbers. Since sign determination is difficult, it is essential to use such a system.

2.3.3. Redundant Residue Number system (RRNS)

A redundant Residue Number System is defined as a standard residue number system to which r additional moduli are appended. All L+r moduli must be relatively prime to ensure a unique representation for each number. The r redundant moduli are not considered to increase the conventional dynamic range

$$M = \prod_{i=1}^{L} m_i \qquad (2.2)$$

The total range is defined as:

$$M_T=\prod_{i=1}^{L+r} m_i \tag{2.3}$$

Any integer in the legitimate range $0 \le X < M$ is represented by L+r residue digits; the interval, $[M,M_T)$, is called the illegitimate range.

Obviously, the RRNS has all the properties of the standard RNS, discussed above. The redundant moduli enable the system to detect errors if the reverse mapping of the number falls into illegitimate range. It is proven that an RRNS with r redundant moduli is capable of detecting r errors, or correcting $\frac{r}{2}$ errors [Man'72].

2.4. Operations in RNS

As was described above, performing closed operations in RNS (addition, subtraction, and multiplication) in RNS is easy and straight forward with no interaction between the residue digits. However, there are some other operations such as division and RNS to binary conversion in which the different residue digits have to be combined together . In the following we will briefly discuss direct sum forward and reverse mappings.

2.4.1. Forward Mapping

Encoding an S bit binary number, x, into RNS is a simple and straightforward operation considering that

$$x = \sum_{i=0}^{S-1} x^{[i]} 2^i \tag{2.4}$$

where $x^{[i]}$ is the ith bit in x. The residue of x mod m, $|x|_m$ can be computed using the following [Sza'67]:

$$|x|_m = \sum_{i=0}^{S-1}{}_m x^{[i]} \, |2^i|_m \tag{2.5}$$

2.4.2.Reverse Mapping

Decoding an RNS number into a weighted magnitude number system, is a more difficult operation and can be performed using two different techniques, as explained in the following.

2.4.2.1. Chinese Remainder Theorem

The Chinese Remainder Theorem (CRT) is the classical method of reconstructing an integer number from a set of residue digits.
It is given by

$$|X|_M = \sum_{i=0}^{L-1}{}_M \hat{m}_i \otimes_M (x_i \otimes_{m_i} \hat{m}_i^{-1}) \tag{2.6}$$

where

$$\hat{m}_i = \frac{M}{m_i} \qquad (2.7)$$

and $\hat{m}_i^{-1} \otimes_m m_i = 1$. The fact that the moduli are relatively prime ensures that $\hat{m}_i^{-1}$ exists. One of the major difficulties in using this method is reducing the result modulo, M, usually a large number and not close to a power of 2.

2.4.2.2. Mixed Radix Conversion

Mixed radix systems are weighted number system and can be defined in general using a set of radices, $\{\ldots b_2, b_1, b_0; b_{-1}, b_{-2}, b_{-3}\ldots\}$. A real number X is represented as:

$$X \Leftrightarrow \{\ldots a_2, a_1, a_0;\ a_{-1}, a_{-2}, a_{-3}\ldots\}$$

where a_i are mixed radix digits. X can be computed using [Knu'81'192]

$$X = \ldots a_3 b_2 b_1 b_0 + a_2 b_1 b_0 + a_1 b_0 + a_0 + a_{-1}/b_{-1} + a_{-2}/b_{-1} b_{-2} + \ldots \qquad (2.8)$$

In the simplest form the $\{b_i\}$ are positive integers and $0 \le a_i < b_i$. A residue number system is said to be associated with a mixed radix system if $b_i = m_i$ for $i \in \{1, 2, \ldots, L\}$. As an example the integer X, can be represented using residue arithmetic and mixed radix notation as:

$$X = (x_1, x_2, x_3, \ldots, x_L) \qquad (2.9)$$

$$X = (a_1, a_2, a_3, \ldots, a_L) \qquad (2.10)$$

It is a simple task to map the integer X into a more conventional fixed radix number system such as decimal using the mixed radix digits, a_i, as given below:

$$X= a_L (m_{L-1}. m_{L-2}. m_{L-3} .... m_1) +...+a_3 (m_2. m_1) + a_2 m_1 +a_1 \qquad (2.11)$$

or in its compact form [Sod'86]:

$$X=\prod_{k=1}^{L} a_k \sum_{i=1}^{k-1} m_i \qquad (2.12)$$

It is preferable to decode an RNS number to binary in two levels, first from RNS to its associated mixed-radix form; once the mixed radix digits are computed it is a straightforward matter to convert it to the binary form. The large modulo M adder is removed in this approach; however, a large binary adder is still needed. Another widely used operation in RNS is extension of the base, which will be reviewed next.

2.4.2.3. Extension of Base

In residue arithmetic, it is frequently necessary to find the residue digit x_L when a new modulus m_L is appended to the existing set. The problem, therefore, is to find the residue digits for a new set of moduli, given the residue digits relative to another set of moduli [Sza'67]. This is best performed by a mixed-radix conversion followed by an extra step as follows.

Assume an associated mixed-radix form for the extended RNS as

$$X=a_{L+1}\prod_{i=0}^{L}m_i+a_L\prod_{i=0}^{L-1}m_i+\ldots+a_3m_1m_2+a_2m_1+a_1 \tag{2.13}$$

at the final step of computing mixed-radix digits, the fact that $a_{L+1}=0$ is used to form an equation. The unknown residue digit is the solution of this equation. For example consider a residue system with 3 moduli, m_1, m_2, and m_3 with the dynamic range $M = m_1\, m_2\, m_3$. Assume an expansion of the base for m_4 is required. An integer X in this associated mixed radix system can be represented as :

$$X= a_4\,(m_1\, m_2\, m_3) +a_3\,(m_1\, m_2) +a_2\, m_1 +a_1 \tag{2.14}$$

where $a_i<m_i$ and a larger dynamic range of $M' = m_1\, m_2\, m_3\, m_4$. Hence we must have $a_4 = 0$ in order to satisfy $X<M$.

2.5. Summary

This chapter has presented a review of some of the fundamental concepts of discrete mathematics. Algebraic structures of groups, rings, and fields have been discussed along with some of their properties. Alternative approaches for constructing these structures were specified and isomorphisms of the groups and rings were also discussed. The fundamental issue in decomposing a large algebraic structure into several smaller structures was examined.

The basics of both standard and redundant residue number systems, were outlined. The notation used throughout this dissertation for the ring and field operations was introduced.

CHAPTER 3

Finite Ring and Field Architectures, A Unified Approach

Abstract

This chapter presents a class of completely pipelined VLSI architectures. It is well suited to digital signal processing applications based on finite rings and fields. These architectures are based on using regular interconnections of a number of simple basic "cells" to perform complex functions at a very high throughput rates.

The technique evolves from the decomposition of each closed calculation using the ring/field associativity property. The cells are perfectly matched to the VLSI medium. Linear systolic arrays, composed of a number of cells, each of a single generic form, are used to carry out the required computations. Examples of different operations are given to illustrate the technique.

3.1. Introduction

The concept of systolic array design, introduced by H.T. Kung and C.E.

Leiserson [Kun'78], is the result of advances in semiconductor technology and of applications that require high throughput. The term "array" originates in the systolic array's resemblance to a grid in which each point corresponds to a processor and a line corresponds to a link between the processors.

Systolic arrays derive their efficiency from:

Pipelining	execution of the operations belonging to different instances of the algorithm simultaneously.
Regularity	use of regular interconnection topology between the cells.
Homogeneity	implementing the algorithm with the use of a few types of identical building blocks.

Algorithms suitable for implementing in systolic arrays have been found in many applications, such as digital signal and image processing, linear algebra and non-numeric applications. These applications are computationaly intensive and require high throughput for their implementations when used in real-time environments.

Many applications and suitable algorithms have been discussed and implemented in the past few years. Often the algorithms can been described by recurrence equations where the required computation is defined recursively using a few simple operations.

Many specialized systolic arrays can be regarded as hardware implementations of a given algorithm with the basic operation performed in each cycle by the processing elements, ranging from a simple bit level operation [Mca'82], to word level multiplication and addition, and even to execution of a complete program.

With the advent of VLSI, the case of architectures based on computations over finite rings and fields is an unexplored situation. The major issue is the question of what the offerings of VLSI are in this case. To answer this question, two main approaches can be taken; the first is to take the existing designs and simply map them into the new medium, [Tay'82]. Simple mappings obviously may not lead to an efficient usage of the new technology. The second approach is to look for styles and techniques which make best use of the advantages and avoid the limitations of the technology, as much as possible.

It is the purpose of this chapter to present an alternative architecture for implementation of finite ring operations in VLSI. It will be shown that our proposed architectures require less silicon area and at the same time provide higher speeds of operation than previously published works[Sod'86]. The structure also satisfies the requirements of: modularity, homogeneity and local communication, sought after in 'good' VLSI designs. By repeatedly using a generic cell structure, we will show that all closed computations can be performed with parallel linear systolic structures. In this chapter the fundamentals of Inner Product Step Processors, (IPSP), [Lei'81] both at word and bit level will be discussed. Different techniques in implementing RNS based operations will be reviewed and a novel structure will be introduced, which implements a finite ring IPSP at the bit level.

3.2. Inner Product Step Processor

An Inner Product Step Processor (IPSP) is formed from combining an adder and a multiplier. This processing cell has 3 inputs: X_{in}, A_{in} and Y_{in} and three outputs: X_{out}, A_{out} and Y_{out}. The IPSP is described below:

$$Y_{out} = Y_{in} + (A_{in} \cdot X_{in}) \tag{3.1a}$$

$$X_{out} = X_{in} \tag{3.1b}$$

$$A_{out} = A_{in} \tag{3.1c}$$

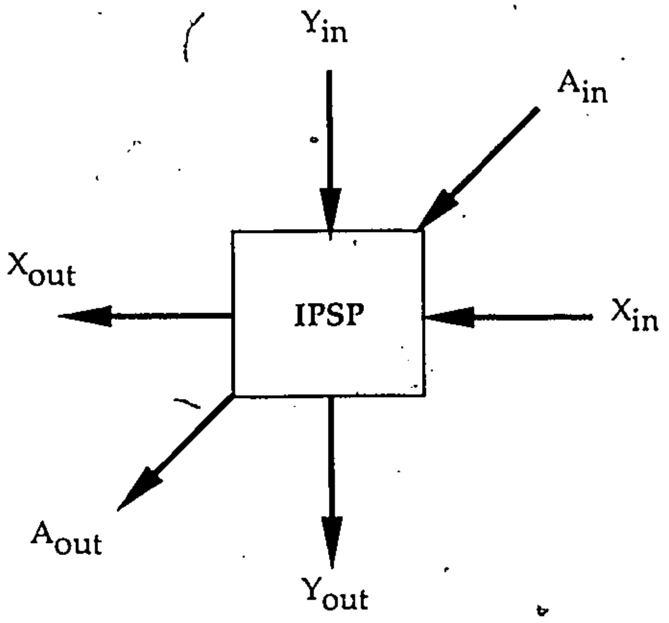


Fig 3.1 The IPSP

and is shown, diagrammatically, in Fig (3.1). The inputs and outputs are word level signals and will contain the number of bits required for the dynamic range of the computation.

The internal multiplier adder has to process several bits per clock cycle resulting in large area and clock cycle time. The interconnection of several of these processors, in a regular form, is used to process the required task. The IPSP can be sliced into bits [Cap'81], [McC'82], each a gated full adder with four inputs: a_{in}, x_{in}, y_{in} and c_{in} (carry in) and four output bits: a_{out}, x_{out}, y_{out} and c_{out}. The operation of the bit-sliced IPSP (BIPSP) is described by:

$$y_{out} = y_{in} \oplus_2 c_{in} \oplus_2 (a_{in} \otimes_2 x_{in}) \tag{3.2a}$$

$$c_{out} = \mathrm{Maj}(y_{in}, a_{in}, c_{in}) \tag{3.2b}$$

$$a_{out} = a_{in} \tag{3.2c}$$

$$x_{out} = x_{in} \tag{3.2d}$$

where $\oplus_2$ and $\otimes_2$ are addition and multiplication modulo 2 and functionally equal to logical exclusive OR and AND, $\mathrm{Maj}(\{x_i\})$ selects the boolean element in the majority in the set $\{x_i\}$, $a_i \in \{0,1\}$. The BIPSP is shown in Fig (3.2).

Several of these slices can be interconnected to form a word level IPSP [Cap'81], [McC'82], [Hat'86]. Pipelining these simpler processors normally results in a much higher throughput compared to the word level IPSP.

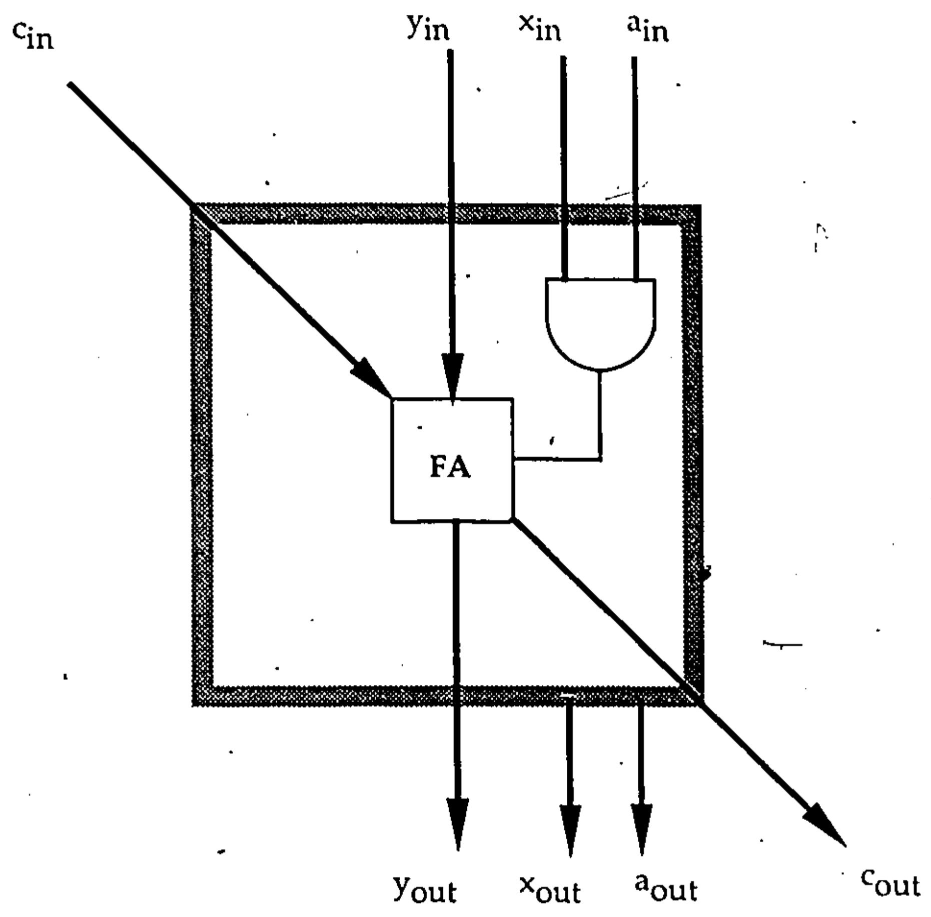


Fig 3.2 The BIPSP.

3.3. RNS Based Inner Product Step Processor

Implementation of RNS based operations has also followed the trend of the technology. Commercially available memory chips have played an important role in simplifying RNS hardware implementations.

The most general approach to computing eqn (3.1), using finite ring arithmetic, is to use a ROM as a direct truth table implementer as shown in Fig (3.3) [Jul'78].

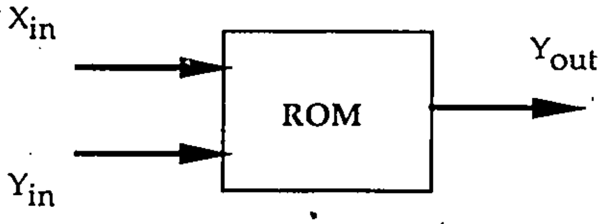


Fig 3.3 IPSP for Finite Ring Calculations

The ROM stores all the possible outcomes of the operation between all possible ring elements. A simple addressing operation, using the concatenated variables as the address, produces the result in the access time of the ROM. For an IPSP with a fixed multiplier we can write the relationship between input (address) variables and output (contents) variable as:

$$Y_{out} = Y_{in} \oplus [A \otimes X_{in}] \qquad (3.3)$$

All the inputs and outputs are B bit ring elements $Y_{out}, Y_{in}, A, X_{in} \in R(m)$. By cascading 2 ROMs it is possible to implement a general, ~~rather~~ than fixed, multiplier.

One of the very few attempts in investigation of bit level RNS has been presented by Jenkins [Jen'77] in which he suggested bit level designs for coding and decoding binary to and from RNS. In the decoding scheme the Chinese Remainder Theorem was converted to an array multiplication form, and the Peled and Liu technique [Pel'74] was used to compute the result. A unique coding procedure was discussed in which the reduced weight of each binary bit is stored in memory, and a counter is used to address the proper location of this table and the outputs are added in a mod m adder. S clock cycles are required to reduce an S bit binary number modulo m.

Here we will consider a structure equivalent to the BIPSP for finite ring calculations. We will use the symbol BIPSP_m to indicate that the processor is operating over the finite ring, modulo m. It will be seen that the BIPSP_m has similar advantages over the IPSP_m that the BIPSP has over the IPSP. In addition the (Area x Period) product of VLSI implementations can be reduced considerably, and dramatically reduced when the pipelining action is taken into account.

3.4. Bit Level Inner Product Step Processor for Finite Rings

The operation of the fixed multiplier BIPSP_m can be defined by rewriting eqn (3.3) as:

$$Y_{out} = Y_{in} \oplus \sum_{i=0}^{B-1}{}_{m} (A \otimes x^{[i]} \otimes 2^i) \qquad (3.4)$$

where Σ_m is the summation operator over the ring m, R(m), and $x^{[i]}$ is the i^{th} bit of $X_{in} \in R(m)$ and $B = \lceil \log_2 m \rceil$.

Using the associativity of the ring operations, eqn (3.4) can be computed by the following recursion:

$$y^{(0)} = 0 \qquad (3.5a)$$

$$y^{(i+1)} = y^{(i)} \oplus [A \otimes x^{[i]} \otimes 2^i] \qquad (3.5b)$$

$Y_{out}=y^{(B)}$ (3.5c)

where i is the spatial array index, $y^{(i+1)}$, $y^{(i)} \in R(m)$.

A possible implementation of the $BIPSP_m$ cell is shown in Fig (3.4).

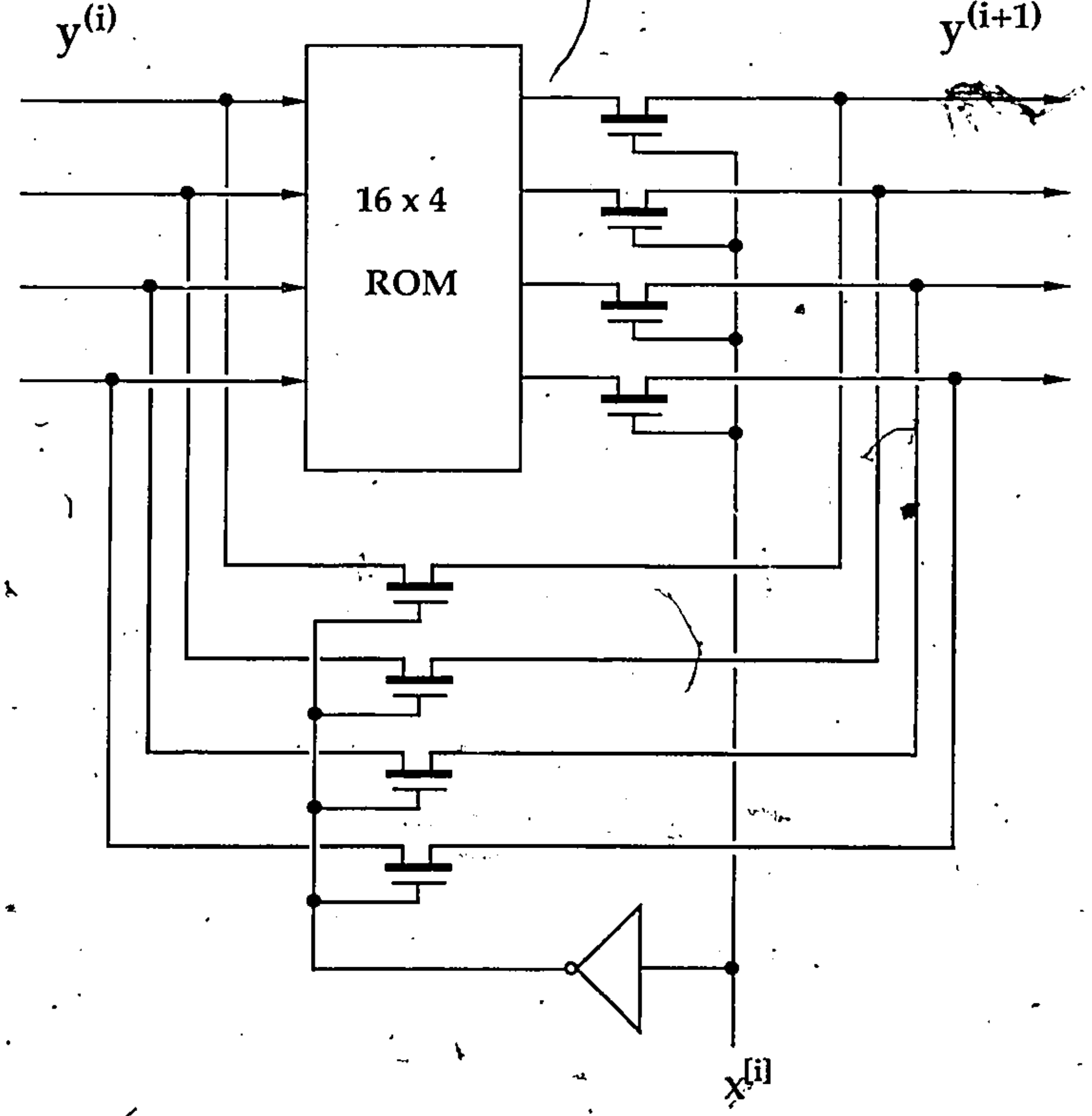


Fig 3.4 Implementation of the $BIPSP_m$ cell

Inputs to the cell are $y^{(i)}$ and $x^{[i]}$; the output is $y^{(i+1)}$.

The cell contains a ROM of size m.B bits and a set of steering switches. The ROM stores the operation of $y^{(i)} \oplus_m [A \otimes_m 2^i]$. The cell computes the following:

For $x^{[i]}=1$: $\quad y^{(i+1)} = y^{(i)} \oplus [A \otimes 2^i]$ (3.6a)

For $x^{[i]}=0$: $\quad y^{(i+1)} = y^{(i)}$ (3.6b)

It can be seen that the operator $IPSP_m$ is equivalent to a linear array containing B stages of $BIPSP_m$, as shown in Fig (3.5).

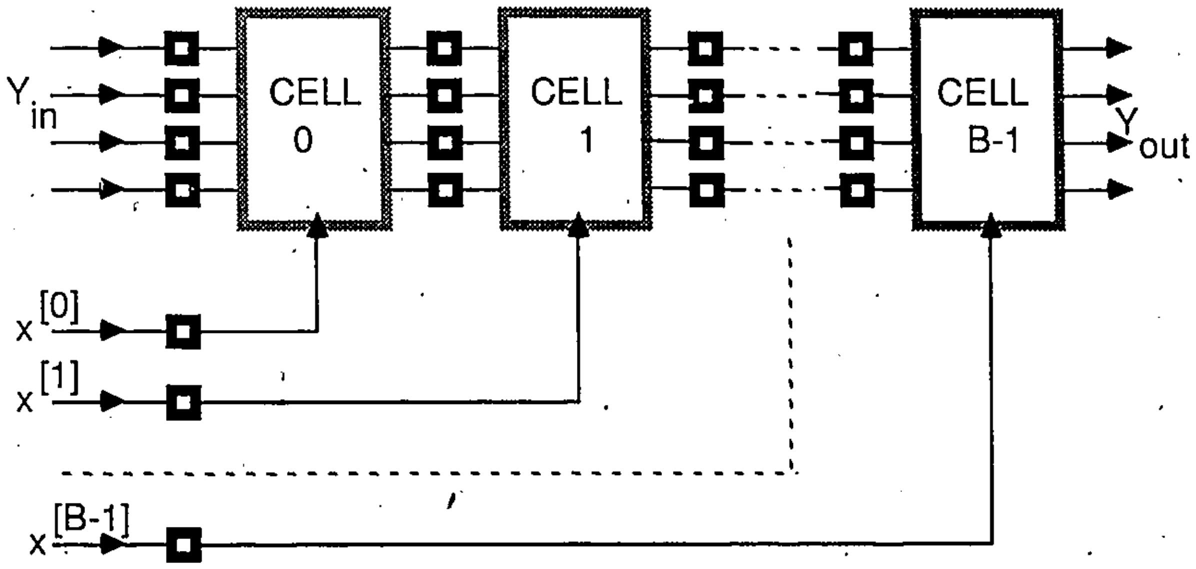


Fig 3.5 The $IPSP_m$ cell implemented using an array of $BIPSP_m$ cells

This technique only requires ROMs with m locations B bits each, a total of mB^2 bit compared to m^2B locations for the single ROM IPSP structure. This yields significant savings for large m.

Without pipelining the array, a delay of $B\tau_m$ is experienced compared to a delay of τ for the single ROM implementation. τ_m is the delay through the $BIPSP_m$ ROM and τ the delay through the $IPSP_m$ ROM. When we consider the use of pipelining (as shown by the latches, ◘, on the diagram), the throughput rate of the $IPSP_m$ cell is inversely proportional to $(\tau + t_L)$ compared to $(\tau_m + t_L)$ for the $BIPSP_m$ array. Where t_L is the latch delay. The ratio $\frac{(\tau + t_L)}{(\tau_m + t_L)}$ can be large for large m and so the $BIPSP_m$ array can offer much larger throughput rates compared to the single $IPSP_m$ cell.

Although the processing section of the cells are uniform throughout the array, the geometry used for feeding the X bits, and their pipeline latches, cause a certain non-uniformity to the VLSI layout. Fig (3.6) shows the $BIPSP_m$ cell with all the X bits fed in parallel through each cell. The bits are circularly shifted by one position for each cell, automatically presenting the correct bit to the steering switches in each cell. This results in a regular, common, cell structure; the trade off is the requirement for extra latches. The multiplying factor is $\frac{2B}{(B+1)}$ which asymptotically approaches 2. For typical values of $B \leq 5$ we have a maximum increase of 66%.

Although this appears considerable, the regularity of the modified structure, mitigates the apparent increase in area and the availability of the entire X sequence at the output of the B^{th} cell is very useful. We have now created a universal systolic cell that only connects to its immediate neighbors. The cell is small, fast, and result in more elegant designs. Fig (3.6) shows the cell with the pipeline latches connected to the input lines. An alternative scheme

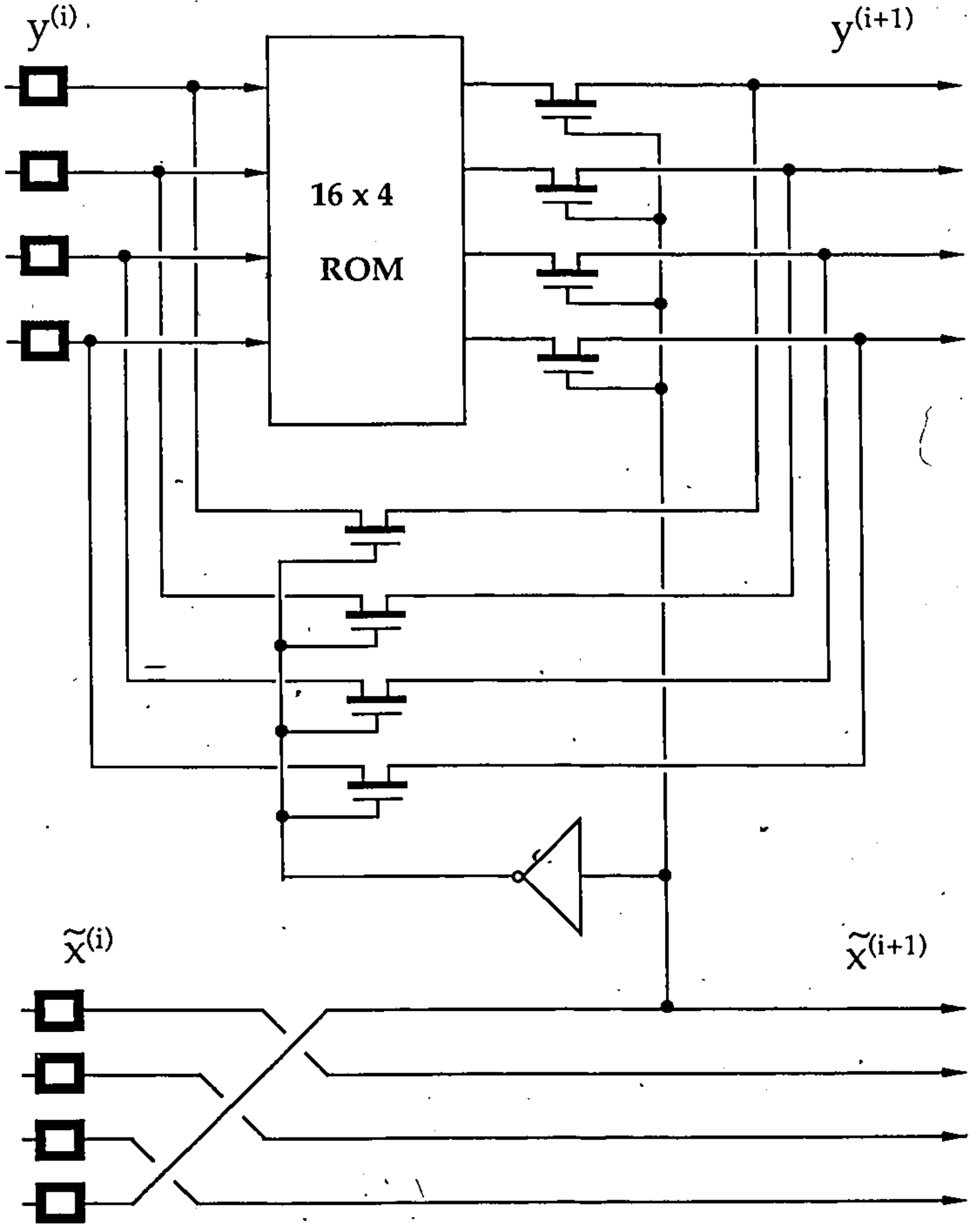


Fig 3.6 Systolic $BIPSP_m$ Cell

allows the pipeline latches at the output.

A more radical change to this design involves removing the steering switches and doubling the ROM size. This is an inferior design for 2 reasons:

A) Increase in Area

B) Decrease in ease of testability

The issue of testability will be covered in chapter 5.

3.5. Complexity Analysis

The complexity of an algorithm is determined by the amount of work (resources) required to solve an instance of a problem. This normally depends on the size of the problem and represents the asymptotic behavior of the algorithm [Big'85]. For example, the number of additions and multiplications is the accepted measure of complexity for the DFT and FFT algorithms, when these operations have to be performed on a general purpose computer or be implemented with random logic on a board. If the above algorithms are to be executed on a VLSI chip, however, measures of requirements other than the addition and multiplication elements are essential.

Generally, in VLSI, the silicon area occupied by the design is one of the most important circuit complexity measure, while the time needed to solve the problem is another widely used measure.

In the special purpose computer design e.g., real time DSP applications where the chip accepts one set of input per clock cycle and produces one set of output in the same period, throughput rate is normally a more important measure.

It is a simple task to derive the throughput and area complexities for the $BIPSP_m$ as given in (3. 7):

$$A=O\ (m\lceil\log_2 m\rceil^2) \quad (3.7a)$$

$$T=O\ (1) \quad (3.7b)$$

and compare it with area and time complexities of the conventional single ROM adders given in (3.8) [Pri'86].

$$A=O\ (m^2\lceil\log_2 m\rceil) \quad (3.8a)$$

$$T=O\ (1) \quad (3.8b)$$

From this analysis one can conclude that the rate of growth of the area is proportional $\lceil\log_2 m\rceil^2$ in $BIPSP_m$ and to m^2 in $IPSP_m$. The throughput rate is fixed and does not vary as the size of the problem grows, in both cases, when pipelining is in effect.

As previously discussed the $BIPSP_m$ is intended to perform operations over finite rings and fields. This imposes a practical limitation of $3<\lceil\log_2 m\rceil<8$. Hence asymptotic behavior of the architecture for large m is not of much interest.

For the purposes of comparison we will investigate the area and access time of a ROM using different approaches.

Case 1

Mead and Conway [Mead'80] have provided a relatively simple model to approximate the area and access time of a ROM. This model suggests an area proportional to M and an access time of $\sqrt{M}$ for a ROM of size M.

	Single ROM adder	$BIPSP_m$ adder	Ratio of the two
Area	$2^{10} \times 5$	$2^5 \times 5^2$	6.4
Throughput with pipelining	72	12.6	5.6
Throughput without pipelining	72	63	1.1

Table (3.1) Comparison of $IPSP_m$ and $BIPSP_m$

Using this simple model the two approaches are compared for a general 5 bit residue adder in table (3.1)

Case 2

A more detailed study by Glasser [Gla'85] partitions the ROM into sub-blocks such as decoder, memory plane, and selector given in Fig (3.7).

A model is developed based on the number of wires in each block. Using this model the area of the ROM is given by:

$$A \propto N2^{N+1} + P2^{R} + P(R-N)2^{R-N+1} \tag{3.9}$$

where 2^R is the number of addressable locations, P is the number of bits in an output word and N is the number of rows in the ROM. The minimum area occurs at

$$1+N\ln 2 = P(1+(R-N)\ln 2)2^{R-2N} \tag{3.10}$$

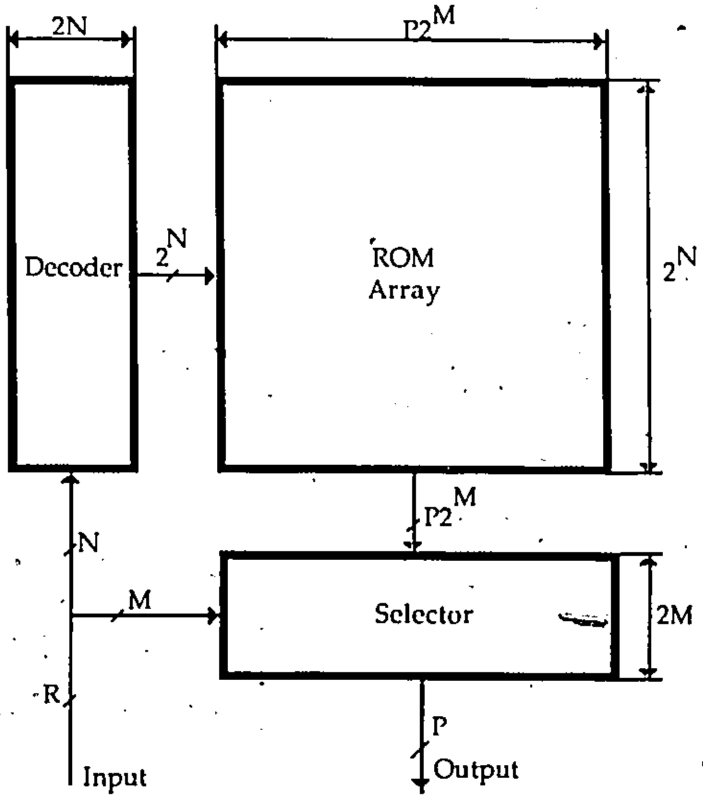


Fig 3.7 ROM structure [Gla'85]

For the example under consideration, the minimum occurs at N=6 for the single ROM implementation , resulting in $A \propto 6528$. The $BIPSP_m$ cell has a minimum area of A $\propto$ 288 for N=3. The area ratio is $\frac{6528}{5 \times 288} = 4.53$ and the access time ratio is 4.76.

Case 3

Mead [Mea'79], [Mea'80] also has presented a memory model which takes the row and column decoders into consideration. The perimeters of the memory module can be computed by:

$$L=1+2\log\alpha+\alpha\, b_0 \tag{3.11}$$

where L is length or width of the layout, α is the number of row or column, and b_0^2 is the area occupied by a transistor. The area for the two type of ROM can be calculated as following:

$$A_1=(\log 72+\log 72+1+72\, b_0)^2 \tag{3.12a}$$

$$A_2=(\log 13+\log 13+1+13\, b_0)^2 \tag{3.12b}$$

if we assume b_0 to be unity the ratio of the areas will be $\frac{85^2}{5 \times 21^2}=3.2$ and the access time would be $\frac{85}{21} = 4$.

Case 4

One also can relate the number of transistors to the area of the design and its access time. Table (3.2) gives the number of transistors for a single ROM and a single cell $BIPSP_m$.

This technique results an area ratio of $\frac{6416}{5 \times 324}=3.9$ and access time ratio is 4.4.

	Single ROM	$BIPSP_m$
Memory	5120	160
Row decoder	64 (2 + 4)=1024	8 x 6=48
Column decoder	16 x 3 x 4=192	4 x 4=16
Latches	8 x 10	8 x 10
Steering switches	------	2 x 10
Total	6416	324*

Table (3.2) Number of transistors in the cells

The results of the above analysis are summarized in table (3.3) indicating a single ROM adder requires at least 3 times more area and runs 4 times slower than the $BIPSP_m$ based array. Another point worth noting is the throughput rate of the $BIPSP_m$ based adders, without pipelining, is normally close, and even better in case 1, to the throughput rate of the single ROM adders. Also similar figures for the area and throughput rate of the ROMs, confirming the previous results, are obtained based on the models provided by Bayoumi [Bay,85].

* The number of transistors in one version of the layout designed by Peter Bird is 293.

	Area Ratio	Access time Ratio
case #1 [Mead,80]	6.4	5.6
case #2 [Glasser,85]	4.53	4.76
case #3 [Mead'79]	3.2	4
case #4 [#Transistors]	3.9	4.4

Table(3.3) Area and time ratio using different methods

3.5. RNS Subtraction

RNS Subtraction can be represented as:

$$X_{out} = X_{in} \oplus_m |-[Y_{in}]|_m \tag{3.13a}$$

$$X_{out} = X_{in} \oplus_m [(m-1) \otimes_m Y_{in}] \tag{3.13b}$$

$$X_{out} = X_{in} \oplus_m \left(\sum_{i=0}^{B-1}{}_m \left((m-1) \otimes_m x^{[i]} \otimes_m 2^i\right) \right) \tag{3.13c}$$

which can be computed on the same array as Fig(3.8).

The contents of the ROMs are shown inside the cell, the cross hatched box represents the circular shift on Y bits so that each cell is controlled by different bits of Y. As an example, let us assume X=7, and Y=9 (' 1001' in binary). The 4 stages of pipeline the data go through before the final result appears at the output can be described as following:

1. The input to the first ROM is X=7 and the ROM in this cell is accessed, since the first bit in Y is 1. The output of this cell will be 6.

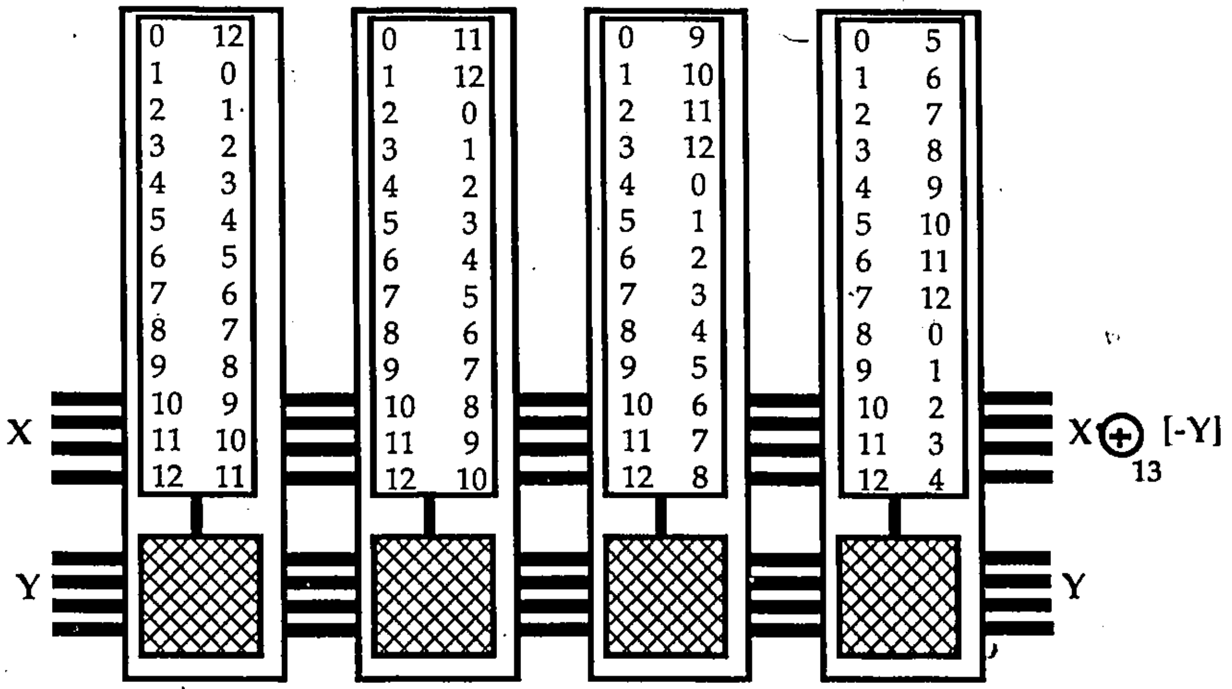


Fig 3.8 Subtraction using BIPSP$_m$

2,3. The second and third ROMs will be by-passed due to the 0's in Y, the input to these cells will appear at the output.

4. Finally the last ROM will use the 6 as its input and produces the final result of 11.

3.6. General RNS Multiplication

Although we will be concentrating on fixed coefficient BIPSP$_m$ cells, it is possible to use similar cell structures for general multiplication. In the following, several different methods for designing modular multipliers based on BIPSP$_m$ cells are presented .

Method 1   General Multiplication

Multiplication of two B bits residue numbers can be decomposed as given in eqn (3.14).

$$X \otimes_m Y = \sum_{l=0}^{B-1} \sum_{k=0}^{B-1} y^{[l]} \otimes_2 x^{[k]} \otimes_m |2^{k+l}|_m \tag{3.14}$$

For the purposes of generating a systolic array realization, we use the following recursive formulation:

$$\mathrm{SUM2}\,(0,-1) = 0 \tag{3.15a}$$

$$\mathrm{SUM2}\,(l,k) = \mathrm{SUM2}(l,k-1) \oplus_m (y^{[l]} \otimes_2 x^{[k]} \otimes_m |2^{k+l}|_m) \tag{3.15b}$$

$$\mathrm{SUM\_P}(l) = \mathrm{SUM2}\,(l,B-1) \tag{3.15c}$$

$$\mathrm{SUM1}(-1) = 0 \tag{3.16a}$$

$$\mathrm{SUM1}(i) = \mathrm{SUM1}\,(i-1) \oplus_m \mathrm{SUM\_P}(i) \tag{3.16b}$$

$$X \otimes_m Y = \mathrm{SUM1}\ (B-1) \qquad (3.16c)$$

Eqns (3.15) represents the inner summation, while eqns (3.16) the outer summation.. Note that the ring operator, $\otimes_2$, implies the logical AND function, since $y^{[i-1]}, x^{[j]} \in \{0,1\}$.

Clearly there are several alternate realizations and both one and two dimensional systolic arrays (with two types of cells) can be constructed. For ROM based cells it is much better to use a linear array which results in a reduced input word width to the ROM element.

Using the same generic cell structure (with the addition of a second word input) we can implement the structure implied by eqns (3.15b) and (3.16b). The two basic cells required are shown in Fig (3.9) and Fig (3.10) which are labeled Cell M and Cell M'.

An array composed of B-1 cell M and one cell M' called cell N implements eqn(3.15) as shown in Fig (3.11). Eqn (3.15) can be implemented using B cell N cascaded in a linear array given in Fig(3.12), B=4 for is used for the figures. Note that a total of B^2 cells are used to implement a general modular multiplier.

Method 2 Prime moduli Multiplication

As was discussed in chapter 2, the nonzero elements of the multiplication table for prime modulus p and addition table modulo p-1 are identical to within isomorphism. This property can be used to perform multiplication

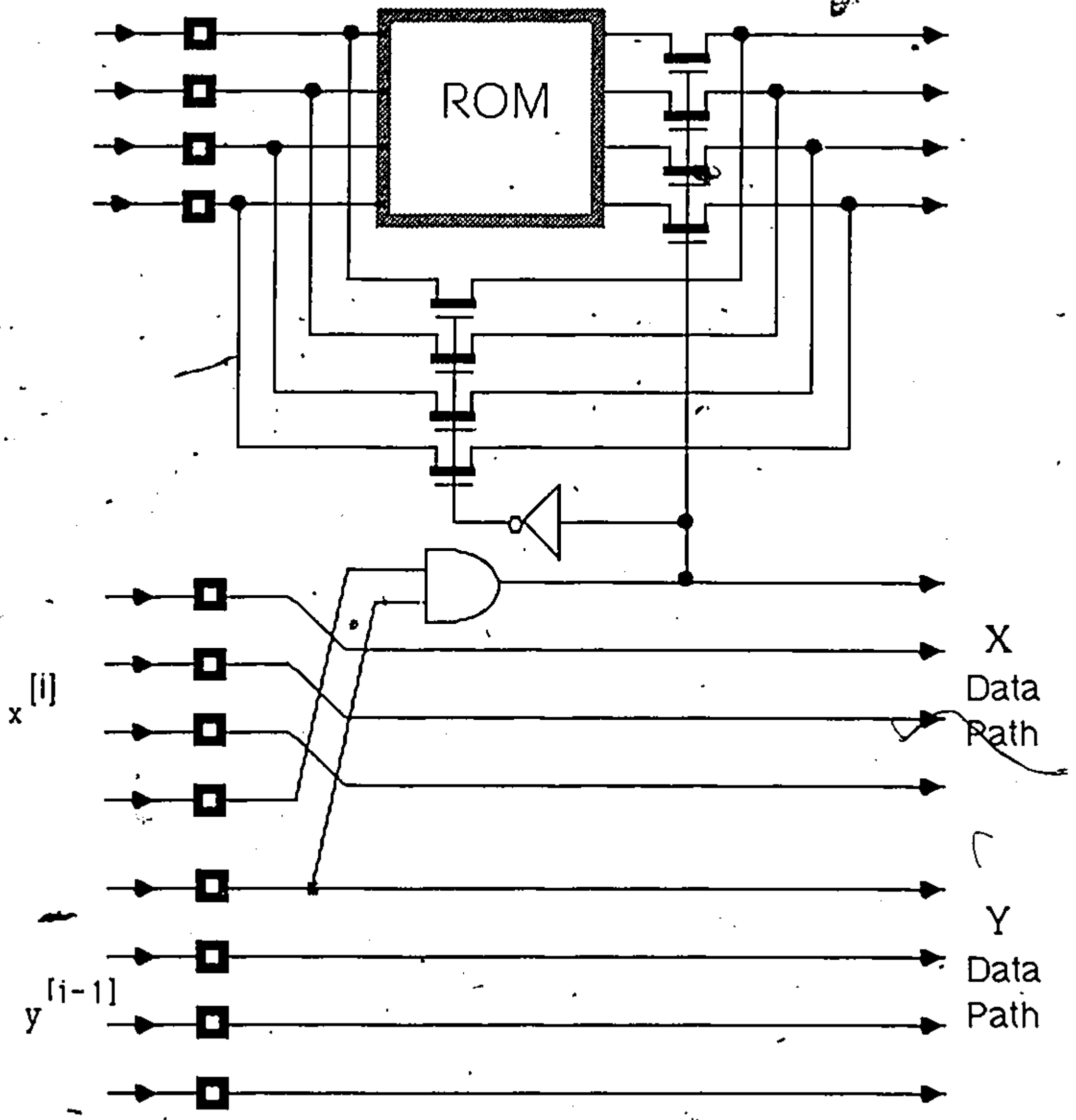
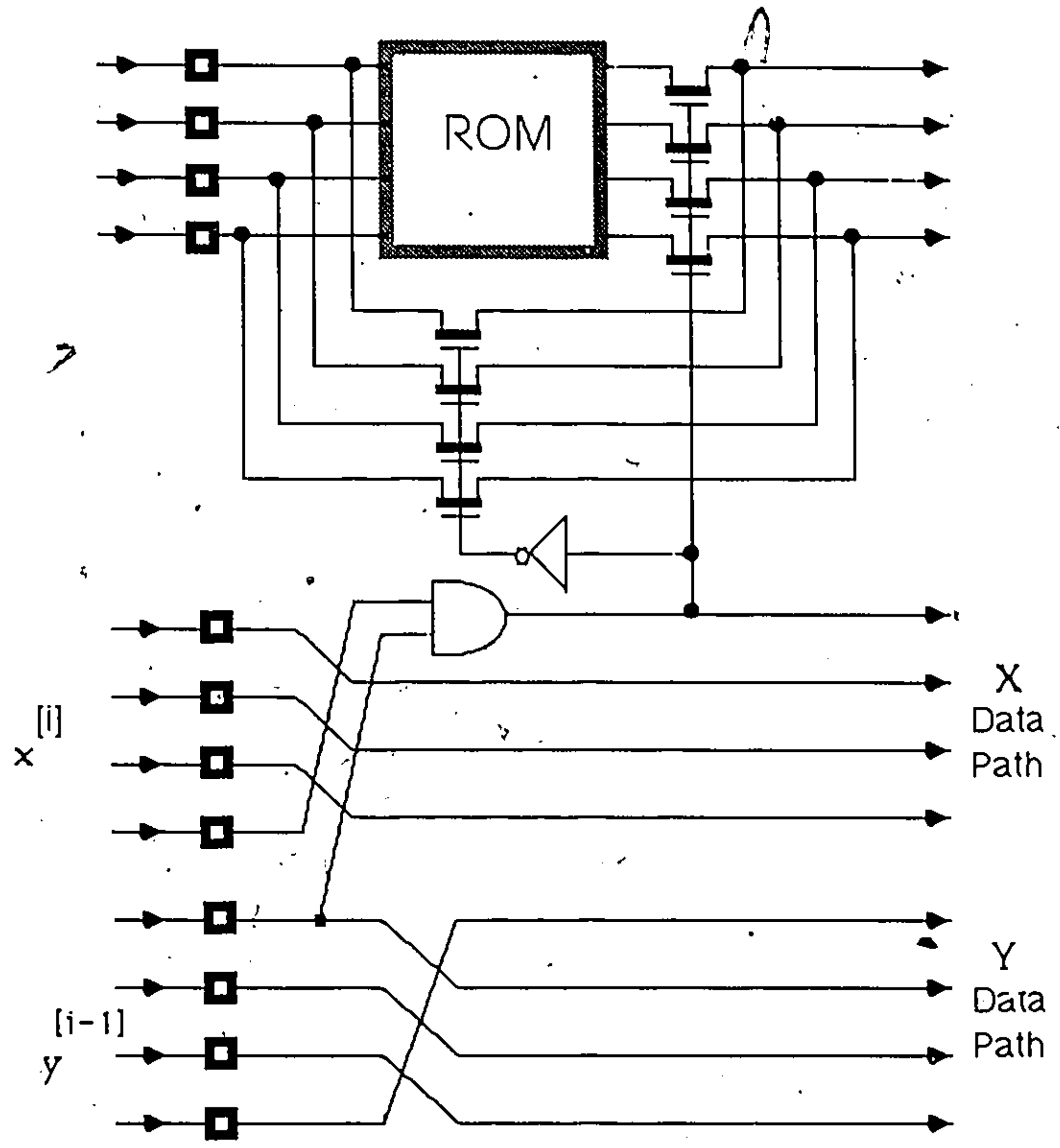


Fig 3. 9 Cell M for the multiplier array



$x^{[i]}$

$y^{[i-1]}$

Fig 3. 10 Cell M' for the multiplier array

modulo a prime number. The indices are looked up for multiplicand and multiplier first, then these indices are added modulo p-1, and finally an inverse index lookup operation gives the result of the operation. In this technique the two operands must be checked to detect zero(s), in which case the result is set to zero.

The proposed cell architecture, with minor modifications, can be used to implement multiplication modulo a prime number, as described below:

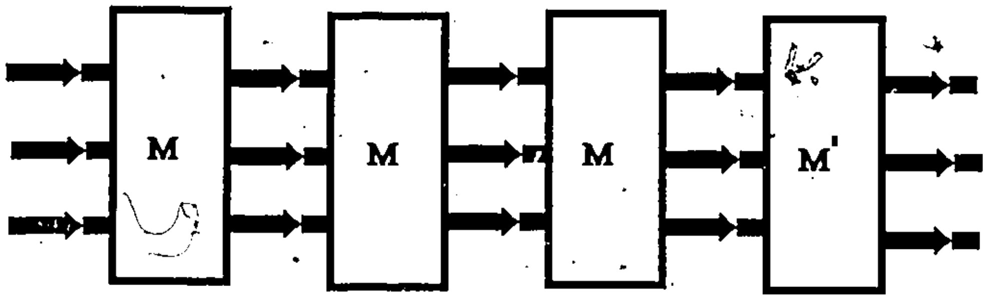


Fig 3.11 Cell N, a substructure for modular multiplication

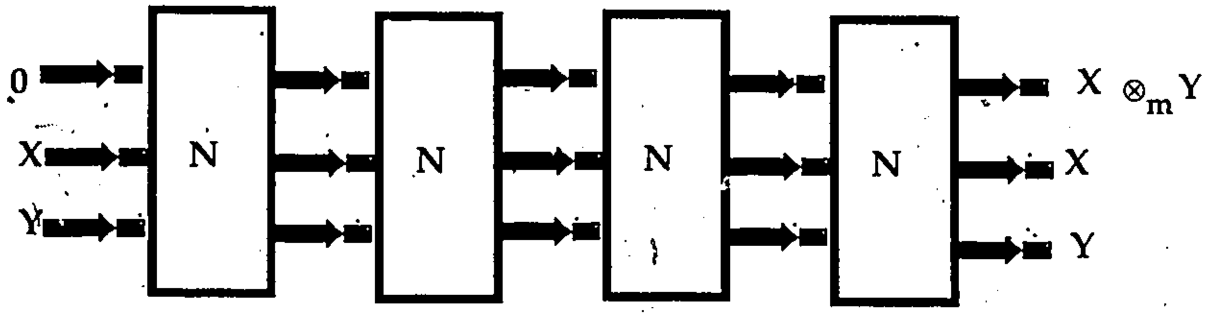


Fig 3.12 Modular multiplication using BIPSP$_m$

Let us assume that the multiplicand $Y \neq 0$, while the multiplier "X" can take any value in the range {0,1,2,...,p-1}.

Implementation of a multiplier based on the technique described above requires B+3 cells,as shown in Fig (3.13). The first two cells contain index tables for multiplier and multiplicand. The following B cells perform the addition modulo p-1, and finally the last cell performs the inverse index

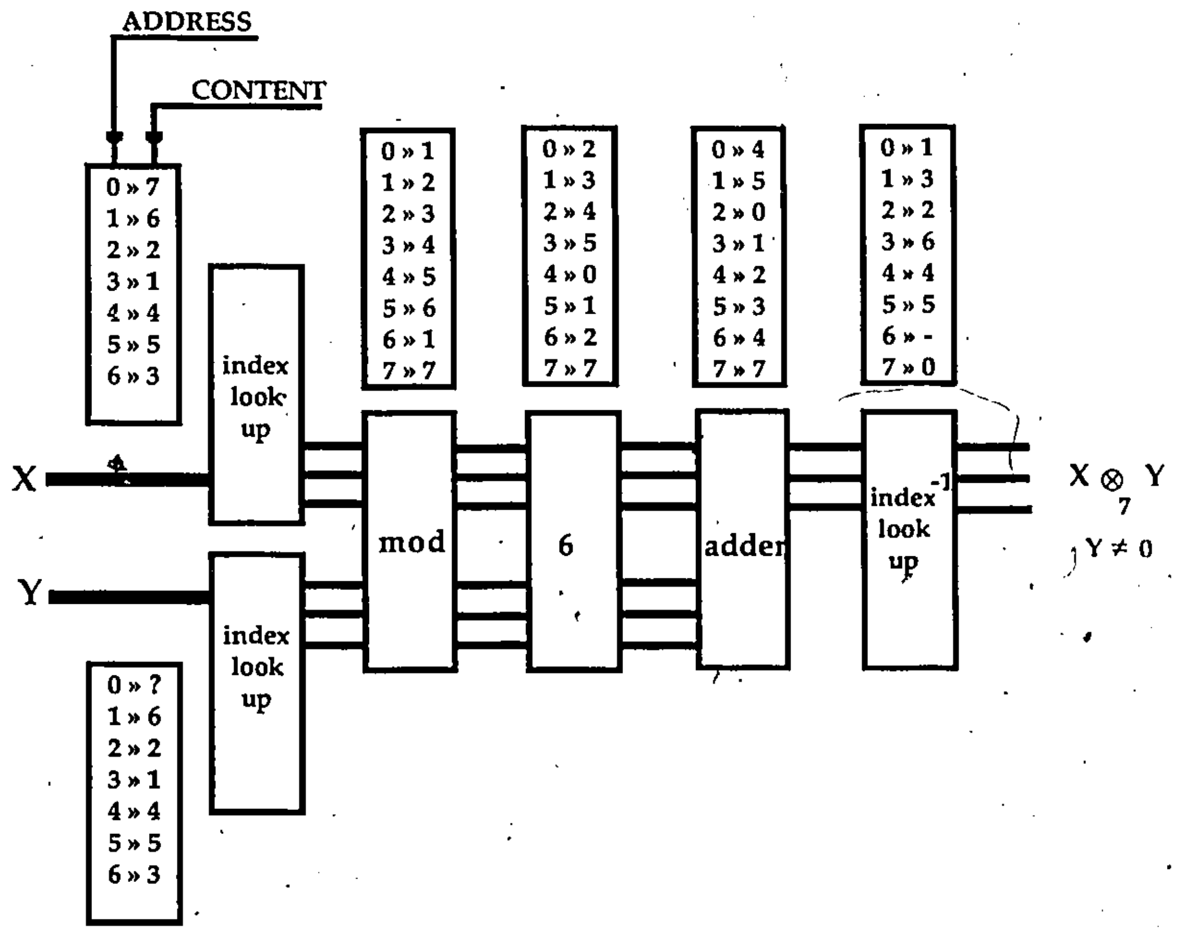


Fig 3.13 Prime base multiplication when Y≠0,
contents of the ROMs are shown next to each cell.

lookup. The special case of X = 0 is handled by programming a forbidden state in the index table and the adder ROMs [Jul'78]. The content for this address in the inverse index table is zero.

Removing the above limitation, and letting the multiplicand Y=0 the array given in Fig (3.13) can not be used.

This case can be handled if the structure and the cells are modified as explained below:

1) The entry in the index table for the multiplicand, Y is set to "0" as shown in Fig (3.14)

2) All the bits in the multiplicand are checked to detect an all "0" case.

3) An extra cell is added to the mod p-1 adder. The contents of its ROM is used only when X=0 is detected. The ROM in this cell is programmed to store the forbidden state (i.e., 7 in the example used in Fig (3.14)). This method of multiplication requires (B+4) cells.

In order to detect if X=0 or not the BIPSP$_m$ is modified and given in Fig (3.15). An extra gate is used to perform a logical OR function between two bits in every cell. Cascading B of these cells provide the following function:

$$y^{[0]} \vee y^{[1]} \vee y^{[2]} \ldots y^{[B-1]}$$

Clearly this results a 0 if X=0 and can be used to set the output to 0 as was described above.

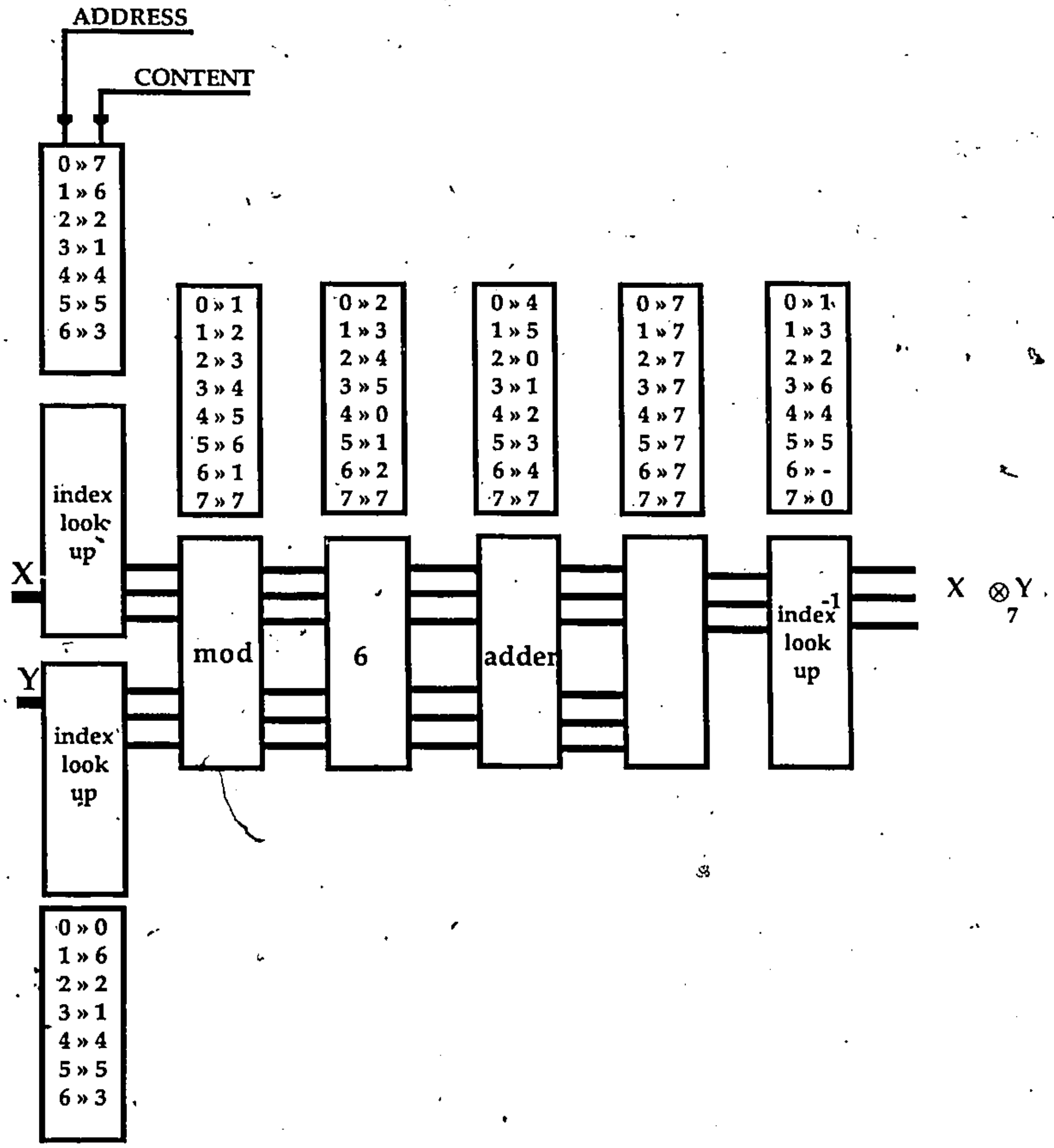


Fig 3. 14 A structure for prime moduli multiplication

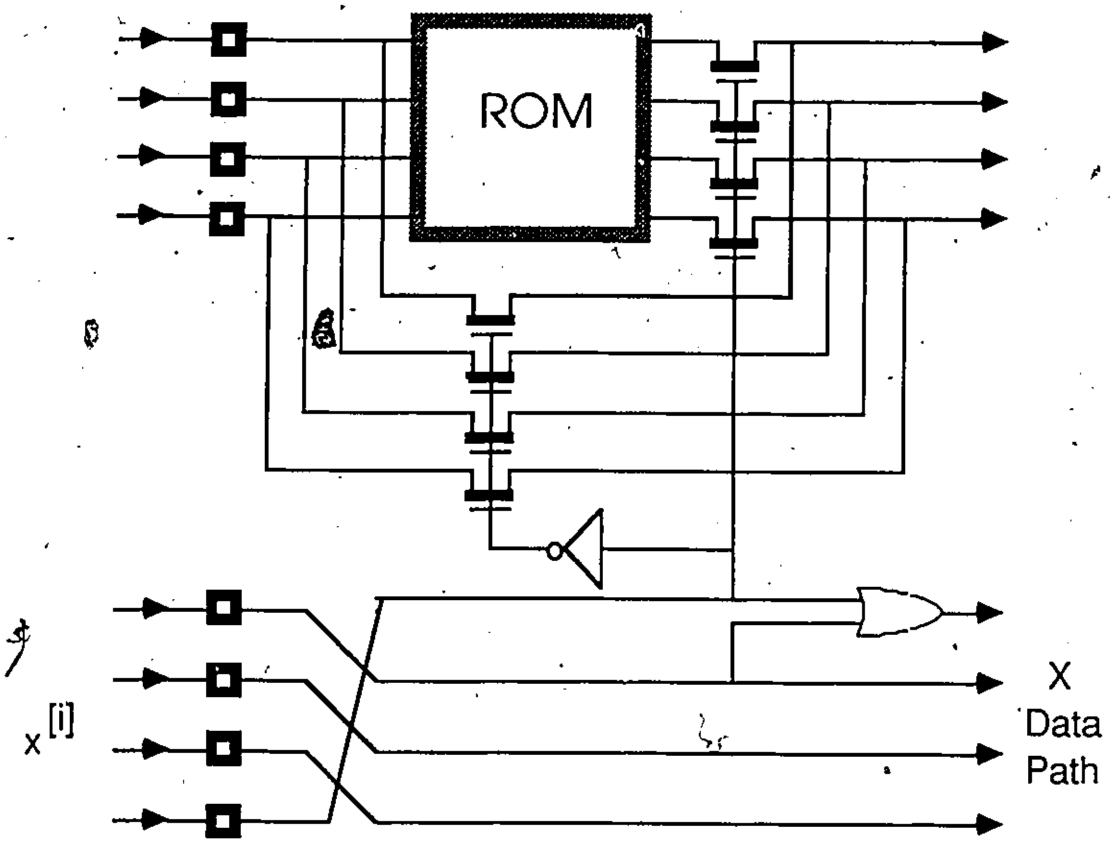


Fig 3.15 Cell for prime moduli multiplication

Method 3 Large Prime Moduli Multiplication

This technique was proposed by Jullien [Jul'80] for multiplying two B bit numbers when B is large. It is based on the rule of indices described above, with the addition modulo p-1 performed in auxiliary sub residue systems. This technique allows the adder to be built using ROMs with 2^B locations instead of ROMs with 2^{2B} locations. Note that the proposed technique we always uses ROMs with 2^B locations throughout the multiplier, for the index tables and the adder. The use of the technique described by Jullien to simplify the adder section, would require ROMs with 2^B locations for the index tables and smaller ROMs for the adder. This non-homogeneity in the design may offset the advantages of the technique.

Method 4   Quarter Square Multipliers

The quarter square method of multiplication is one of the most popular techniques in analog circuitry. It is based on eqn(3.17).

$$X \times Y = \left(\frac{X+Y}{2}\right)^2 - \left(\frac{X-Y}{2}\right)^2 \tag{3.17}$$

This approach was studied by Nussbaumer [Nus'76] to implement digital filters using ROMs and Pollard [Pol'76] studied it for use in multiplication over a Galois field GF(p), p prime. Soderstrand generalized the technique to rings where the multiplicative inverse for 4 exists. In this case eqn (3.16) can be rewritten as:

$$X \oplus_m Y = \left| \left(\frac{X+Y}{2}\right)^2 - \left(\frac{X-Y}{2}\right)^2 \right|_m$$

$$= \left|[(X \oplus Y)^2 \otimes 4^{-1}] \oplus \{-[(X \oplus [-Y])^2 \otimes 4^{-1}]\}\right|_m \tag{3.18}$$

Note that all the operations can be performed in ring m. This technique can be used to implement a general finite ring multiplier based on the $BIPSP_m$ cell, as shown in Fig (3.16). The first B cells in the top row implements the addition, $(X \oplus Y)$, and the last cell performs the operation defined as $|[(X \oplus Y)^2 \otimes 4^{-1}]|_m$. The first B cells in the lower row are programmed to implement $(X \oplus [-Y])$, while the final cell in this row performs the same function as the last cell in the top row is similar to the last cell in the top row, $|[(X \oplus [-Y])^2 \otimes$

$4^{-1}]|_m$. Finally these two results are subtracted in the middle row. A total of 3B+2 cells is used for this scheme.

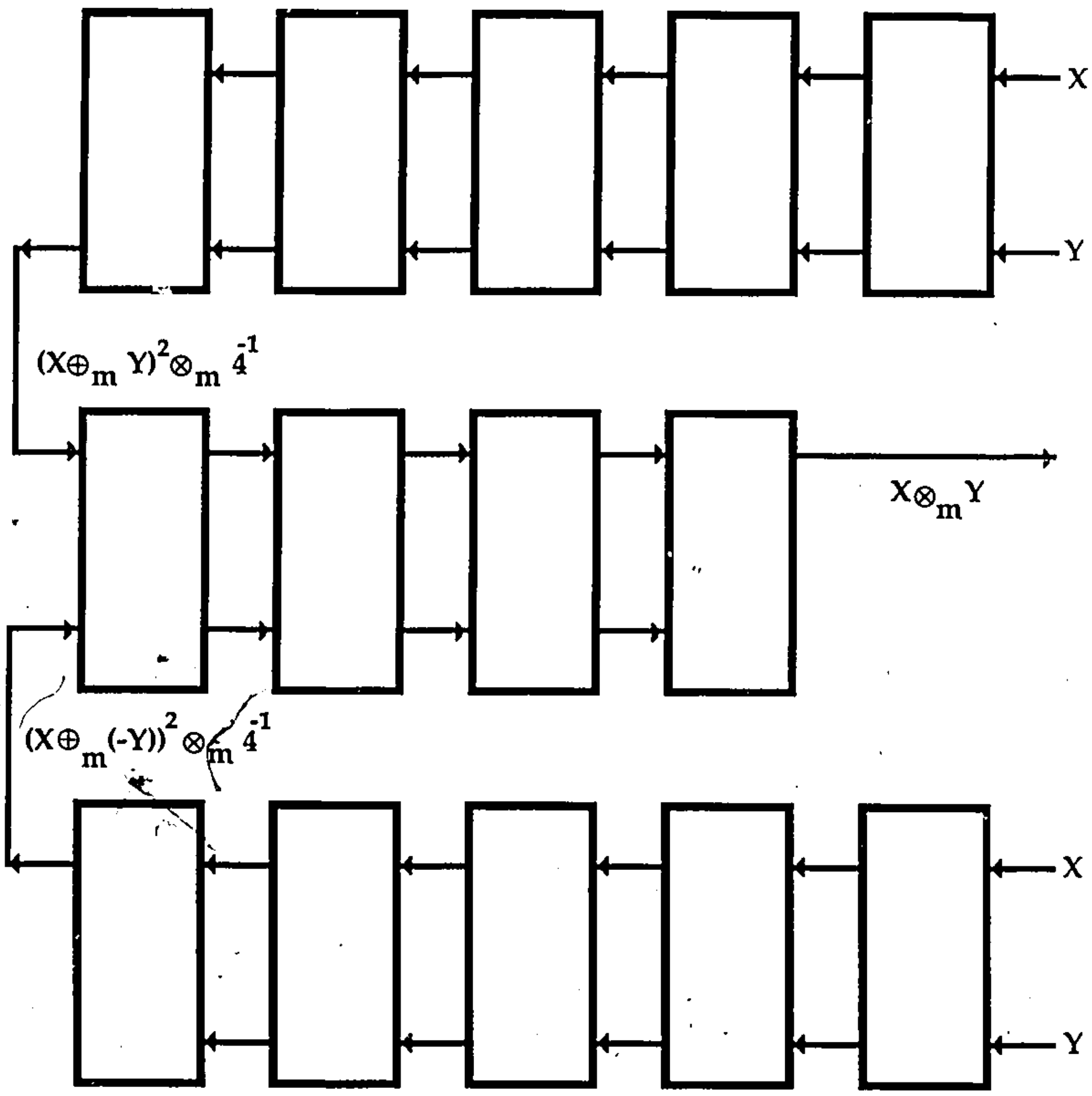

X
Y
$(X \oplus_m Y)^2 \otimes_m 4^{-1}$
$X \otimes_m Y$
$(X \oplus_m (-Y))^2 \otimes_m 4^{-1}$
X
Y

Fig 3.16 A mod m multiplier based on quarter square technique

Taylor [Tay'81] proved that the square law multiplier technique is general and can be used on rings with an even number of members or in other words when no multiplicative inverse for 4 exists. This is only applicable when x+y and x-y are computed with ordinary arithmetic. Using quarter square method as a general multiplication technique then requires the size of the ROMs performing the addition and subtraction to be increased to 2^{B+1}, double the size of the case where $|4^{-1}|_m$ exists. Next we will consider the problem of conversion of a binary number into a residue number.

3.6 Forward Mapping

The mapping $R(M) \Rightarrow \sum R(m_i)$ is straightforward. The mapping operator is $\{|.|_{m_i}\}$ and maps $X \Rightarrow \{x_0, x_1, \ldots, x_{L-1}\}$. An S bit binary number can be reduced, modulo m, as follows:

$$|X|_m = \sum_{b=0}^{S-1}{}_m\, 2^b \otimes_m x^{[b]} \tag{3.19}$$

Eqn (20) can be computed via the following recursion:

$$Y^{(0)} = 0 \tag{3.20a}$$

$$Y^{(i+1)} = Y^{(i)} \oplus_m \{2^i \otimes_m x^{[i]}\} \tag{3.20b}$$

$$|X|_m = Y^{(S)} \tag{3.20c}$$

Eqn (3.20b) can be calculated using the $BIPSP_m$, modified to contain L bits in its data path. A linear array containing L stages of $BIPSP_m$ cells is capable of modulo m reduction. This is shown in Fig (3.17).

If it is required to multiply the input by a fixed coefficient, A, this can be performed without any extra hardware. The stage can also correctly encode 2's complement form for the input binary sequence, by generating the additive inverse of an element in the ring, modulo m, if required. This is performed

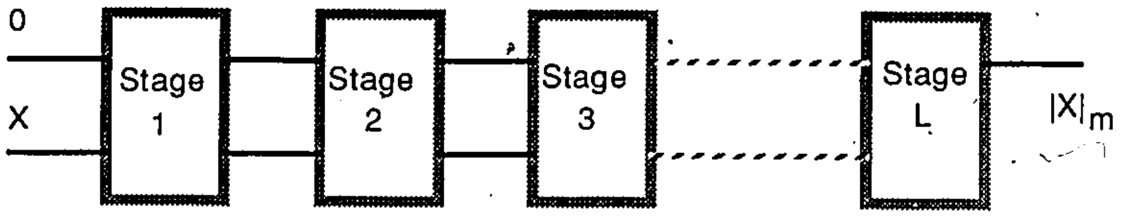


Fig 3.17 Modulo m reduction using an array of $BIPSP_m$ cells

by mapping address to content within the final stage (MSB stage) ROM as follows:

$$\text{Address (i)} \oplus_m \{ |-2^{L-1}|_m \otimes_m A \} \Rightarrow \text{Content (i)} \qquad (3.21)$$

The procedure is illustrated in Fig (3.18). The columns represent ROM contents for a 16 bit binary to modulo 11 ring mapping. The mapping also includes a pre-stored multiplication of 13. An example of mapping -29x13 is shown, with active ROM contents outlined. The ROM addresses are the column on the left, and the serial binary input is shown, starting at the LSB on the left, as the top row. Where the binary input is 0, no ROM contents are active (addresses steered past the ROMs). This example also illustrates the cyclic nature of the ROM contents. The only information required to generate the entire ROM contents is the first location. The other locations follow in a cyclic fashion; this is direct evidence of the ROM contents forming an additive group under the ring addition operation.

Although the contents always follow in this cyclic fashion, our efforts, so far, show that a general ROM structure is the most efficient implementation mechanism. The forward mapping technique can be simplified further more by the following observations:

1. If A is an S bit binary number to be reduced modulo m

	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1
0	2	4	8	5	10	9	7	[illegible]	6	1	2	4	8	5	10	2
1	3	5	9	6	0	10	8	4	7	2	3	[illegible]	9	6	0	3
2	4	[illegible]	10	7	1	0	9	5	8	3	4	6	10	7	1	4
3	5	7	0	8	2	1	10	6	[illegible]	4	5	7	0	8	2	5
4	6	8	1	9	3	2	[illegible]	7	10	5	6	8	1	9	3	6
5	7	9	2	10	4	3	1	8	0	6	7	9	2	10	4	7
6	8	10	3	0	5	[illegible]	2	9	1	7	8	10	3	0	5	[illegible]
7	9	0	4	1	6	5	3	10	2	8	9	0	4	1	[illegible]	9
8	10	1	5	2	7	6	4	0	3	9	10	1	5	2	7	10
9	0	2	6	3	8	7	5	1	4	[illegible]	0	2	6	3	8	0
10	1	3	7	4	9	8	6	2	5	0	[illegible]	3	7	4	9	1

Fig 3.18 ROM Contents for -29 $\otimes_{11}$ 13

$$|A|_m = \left|\sum_{i=0}^{S-1} 2^i a^{[i]}\right|_m = \left|\sum_{k1=B-1}^{S-1} 2^{k1} a^{[k1]}\right|_m \oplus_m \left|\sum_{k2=0}^{B-2} 2^{k2} a^{[k2]}\right|_m \qquad (3.22)$$

$$\left|\sum_{p=0}^{B-2} 2^p\, a^{[p]}\right|_m = \sum_{p=0}^{B-2} 2^p\, a^{[p]} \tag{3.23}$$

where $B=\lceil \log_2 m \rceil$, hence the first B-1 bits of A is already reduced mod m.

2. Input A can be split into smaller partitions as suggested in the past [Jul'78], this allows us to use the same cell to be used in the operation of conversion. For example consider reducing a 14 bit binary number, A, modulo a 4 bit number (e.g., m=13).

$$|A|_m = \left|\sum_{i=0}^{6} 2^i\, a^{[i]}\right|_m \oplus_m \left|\sum_{k=7}^{13} 2^k\, a^{[k]}\right|_m$$

$$= \left|\sum_{i1=3}^{6} 2^{i1}\, a^{[i1]}\right|_m \oplus_m \sum_{i2=0}^{2} 2^{i2}\, a^{[i2]}$$

$$\oplus_m \left|\sum_{k1=10}^{13} 2^{k1}\, a^{[k1]}\right|_m \oplus_m \sum_{k2=7}^{9} 2^{k2}\, a^{[k2]} \tag{3.24}$$

Hence no modification of the data path is required. Fig (3.19) shows the encoder with the simplifications discussed above. The latency is reduced in this case.

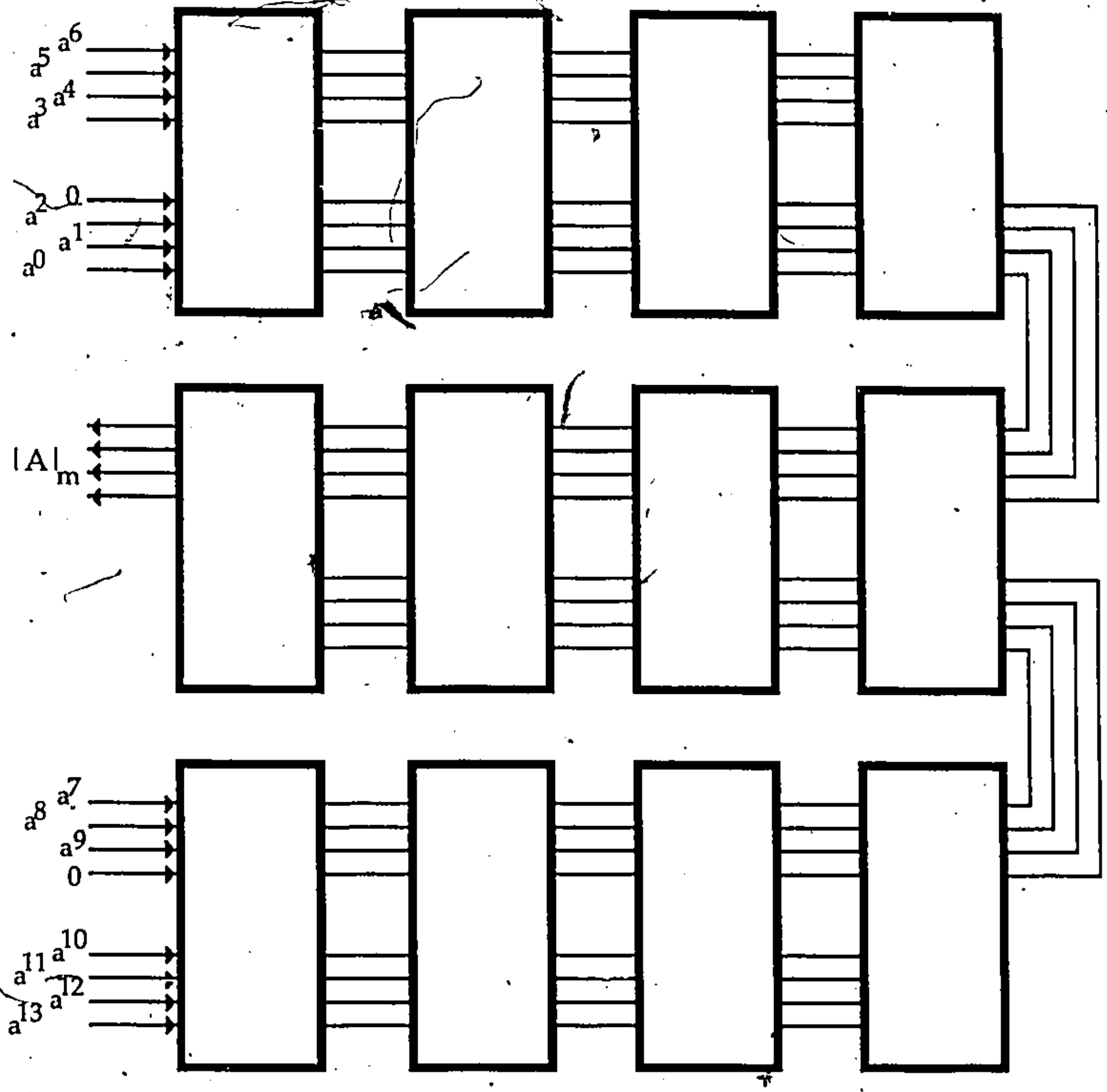


Fig 3.19 Reducing a 14 bit binary number using the original cell

3.7. Summary

In this chapter we have introduced a general architecture for implementing basic RNS operations. The implementation is based on bit-slicing the basic inner product sum processor (IPSP) to yield a BIPSP. The finite ring counterparts are $IPSP_m$ and $BIPSP_m$. The $BIPSP_m$ yields to a linear systolic implementation of finite ring operations. The area and time cost of the proposed technique have been compared to the costs of a conventional, single ROM, modulo adder, using different models. We have proved that the area of $BIPSP_m$ based adder is at most 30% of the single ROM adders, while the access time is 25%.

CHAPTER 4

Structures for High Speed DSP Algorithms Using RNS Based VLSI Module

Abstract

This chapter presents simple structures based on the BIPSP_m cell, introduced in chapter 3. These structures are capable of performing convolution, variable coefficient correlation and RNS to binary decoding. The structures evolve from the decomposition of each closed calculation using the ring/field associativity property. Linear systolic arrays, formed with multiple elements, each of a single generic form, are used for all calculations. The pipeline cycle is determined from the generic cell ~~and~~ is predicted to be very fast based on a critical path analysis. Designs for implementing FIR filters, both in bit serial and bit parallel are given and compared with alternate design techniques in the literature. A binary correlator and its BIPSP_m based counter part are briefly discussed and compared. An interesting comparison with efficient and high speed binary based FIR filters reveals the advantages of the bit level RNS over binary array implementations in area and power consumption without any sacrifice in speed. Finally an RNS to binary decoder, is built around the BIPSP_m cell, is discussed.

4.1. Introduction

A simple architecture which performs computations over finite rings/fields, was introduced and discussed in chapter 3. Examples of simple operations such as modular addition, subtraction, multiplication, and binary encoding were given. This chapter, which is a continuation of chapter 3 deals with algorithms which have greater computational complexity including higher level DSP applications of digital filtering, digital correlation and RNS to binary decoding to prove the capabilities of the $BIPSP_m$.

This chapter begins with the investigation of Finite Impulse Response filters because finite duration sequences have certain desirable properties from the point of view of filter design. For example, the question of stability and realizability never arise since FIR filters, with finite coefficients, are always stable and, with an appropriate finite delay, can always be made realizable [Rab'75]. Further more, the filter impulse response can be designed so that their frequency response have an exactly linear phase characteristic. Finally the lack of feedback loops make them ideal for systolic array implementation [Dan'84]. One of the disadvantages of FIR filters are the large values of N, the order of the filter, for most useful filter characteristics which makes them computationaly intensive and costly to implement.

4.2. RNS based FIR filters

Using residue arithmetic, an N^{th} order fixed coefficient FIR filter can be expressed as:

$$|Y(n)|_m = \sum_{i=0}^{N-1}{}_m (A(i) \otimes_m X(n-i)) \tag{4.1}$$

where X, A, Y are input, filter coeffecient, and output and A,X and $Y \in R(m)$, and $\sum_m$ implying that the summation is performed over the ring m, R(m). Slicing (4.1) at the bit level yields:

$$|Y(n)|_m = \sum_{b=0}^{B-1}{}_m 2^b \otimes \sum_{i=0}^{N-1}{}_m (A(i) \otimes x^{[b]}(n-i)) \tag{4.2}$$

where $x^{[b]}$ is the b^th bit of X and $B = \lceil \log_2 m \rceil$. Eqn(4.2) can be rewritten as:

$$|Y(n)|_m = \sum_{b=0}^{B-1}{}_m (2^b \otimes |y_b(n)|_m) \tag{4.3}$$

where:

$$|y_b(n)|_m = \sum_{i=0}^{N-1}{}_m (A(i) \otimes x^{[b]}(n-i)) \tag{4.4}$$

In the following sections, different architectures, capable of computing eqn (4.4), which are based on the $BIPSP_m$ will be discussed. The structures depend on the system input data flow. In general, the input data flow in a digital signal processing system can be categorized into 4 different groups. These groups are defined by different levels of parallelism at the implementation level. The first and simplest form is *word serial/ bit serial* input in which the input bits are supplied in a serial fashion, and the input words enter the system sequentially. In the second case, *word serial/ bit parallel* input, the input words enter sequentially, with all the bits in each word presented in one clock period. The other two cases namely *word parallel/ bit serial* and *word parallel/ bit parallel* differ from the previous cases by inputting several words from the input sequence in parallel during each clock period. A graphical representation of the above is provided in Fig(4.1) [Kes'87]. The dots, in Fig (4.1) represent bits, a straight line interconnecting several dots represent a word.

In the following section structures for the implementation of *word serial* input data flow using the $BIPSP_m$ cell will be discussed.

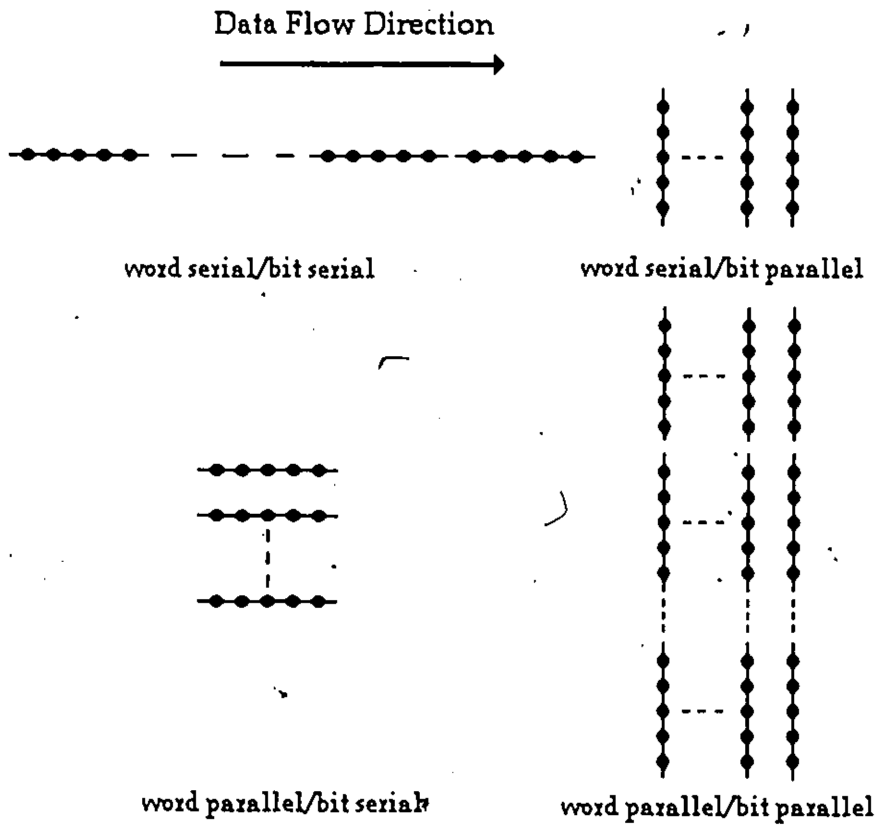


Fig 4.1 Input data flow

4.2.1. Word serial/bit serial FIR filter

In a typical word serial/bit serial FIR filter, the input bits

$$\{\dots x^0(n), x^1(n)\dots x^{B-1}(n), x^0(n+1)\dots\}$$

enter the filter, one per clock cycle. A single array can be used to compute the partial results, $\{|y_b(n)|_m;\ b\in [0, B)\}$. One partial result per clock cycle is computed as given in eqn (4.4). The outer summation in eqn (4.2) is performed in a mod m adder [Tah'87b]. The block diagram of the filter is given in Fig (4.2) and its implementation will be discussed in the following sub-section.

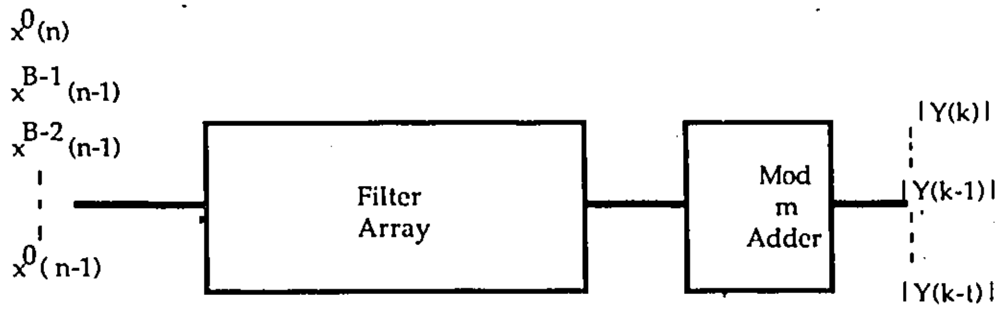


Fig 4.2 Block diagram for *word serial/bit serial*

4.2.1.1 Word serial/bit serial FIR filter implementation

The partial results, $|y_b(n)|_m$, given in eqn(4.4), can be computed by the following recursive relationship:

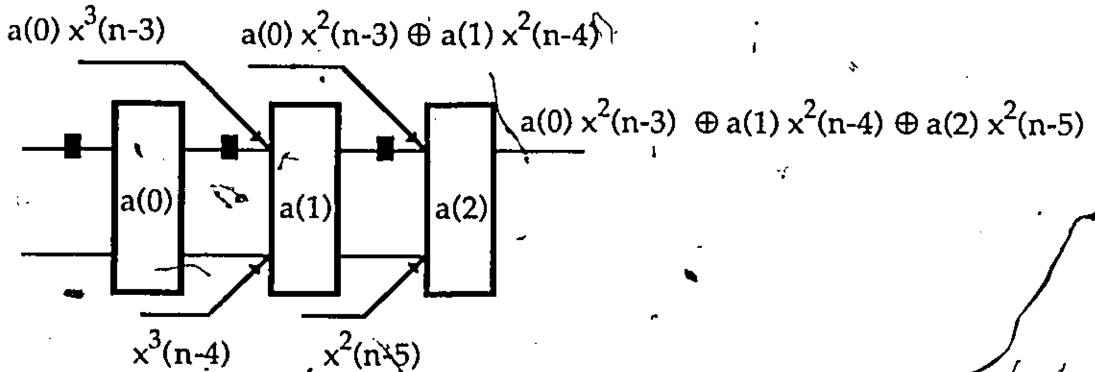
$$|y_b^{(0)}(n)|_m = 0 \quad (4.5a)$$

$$|y_b^{(i+1)}(n)|_m = |y_b^{(i)}(n)|_m \oplus (A(i) \otimes x^{[b]}(n-i)) \quad (4.5b)$$

$$|y_b(n)|_m = |y_b^{(N)}(n)|_m \quad (4.5c)$$

Computing (4.5b) for b={0, 1, 2, 3, ..., B-1} using one array poses a problem when the input string contains bits with different significance interleaved.

Our approach in solving this problem can be best explained using an example. Let us assume B=4, N=3, the filter coefficients a(0), a(1), a(2) are stored in a three cell array as shown in Fig (4.3). The required output of the array for 4 consecutive clock periods are:



a(0) x^3(n-3)
a(0) x^2(n-3) ⊕ a(1) x^2(n-4)
a(0) x^2(n-3) ⊕ a(1) x^2(n-4) ⊕ a(2) x^2(n-5)
a(0)
a(1)
a(2)
x^3(n-4)
x^2(n-5)

Fig 4.3 Timing for a serial bit FIR filter

t=0 $\quad a(0)\,x^1(n-2) \oplus a(1)\,x^1(n-3) \oplus a(2)\,x^1(n-4)$

t=-1 $\quad a(0)\,x^0(n-2) \oplus a(1)\,x^0(n-3) \oplus a(2)\,x^0(n-4)$

t=-2 $\quad a(0)\, x^3(n-3) \oplus a(1)\, x^3(n-4) \oplus a(2)\, x^3(n-5)$

t=-3 $\quad a(0)\, x^2(n-3) \oplus a(1)\, x^2(n-4) \oplus a(2)\, x^2(n-5)$

Now suppose that the output belonging to t=-3 appears at the output of the array, it is obvious that the inputs to the final cell, computing eqn (4.5b), are $x^2(n-5)$ on the lower line and $a(0)\, x^2(n-3) \oplus a(1)\, x^2(n-4)$ as partial result input on the top line. Since the array is pipelined, the output belonging to the instance t=-2 is being processed in the second cell during this same clock period, meaning that the input to the second stage must be $x^3(n-4)$ on the lower line and $a(0)\, x^3(n-3)$ as the partial result input. From the above it becomes clear that the lower input lines in Fig (4.3) contain data skew of 5 ($x^2(n-5)$ to $x^3(n-4)$), or B+1 in general.

With this in mind eqn (4.5b) can be computed by using a slightly modified version of the BIPSP$_m$, introduced in chapter 3, shown in Fig (4.4).

The structure is similar to the BIPSP$_m$ cell discussed in chapter 3, but modified in its X data path and the number of the latches. The ROM stores the function:

$$|y_b^{(i+1)}(n)| = |y_b^{(i)}(n)|_m \oplus A(i) \qquad (4.6)$$

Multiplication by $x^{[b]}$ is performed by looking up the contents of the ROM $(x^{[b]}=1)$ or steering the address to the output $(x^{[b]}=0)$.
An array of N of these cells computes [illegible]rtial result, $\{|y_b(n)|_m\}$, one at a time.

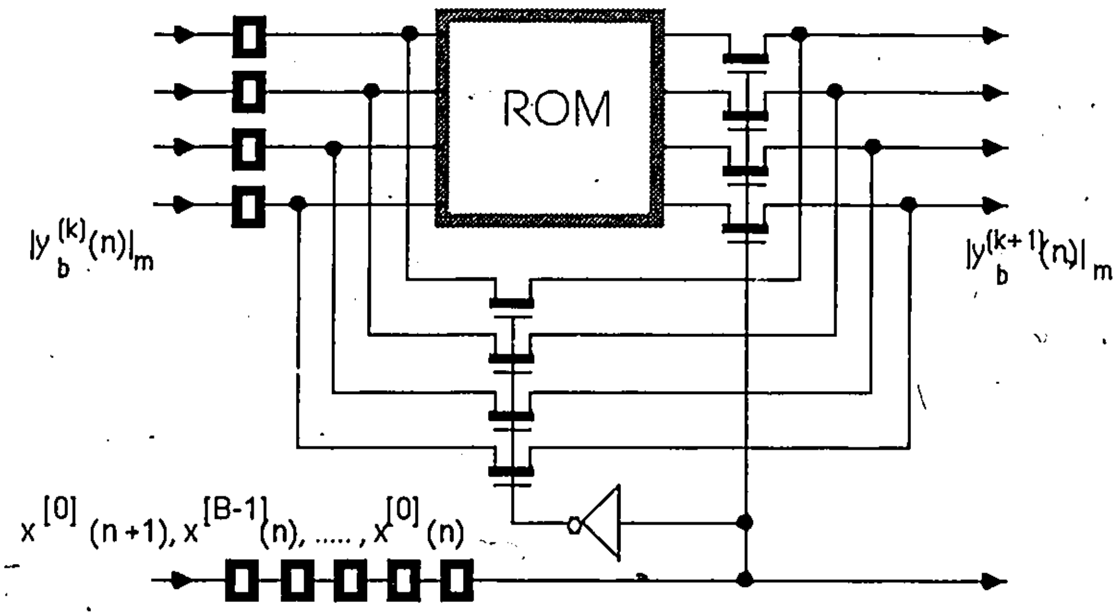


Fig 4.4 $BIPSP_m$ cell for the bit serial FIR filter

Following the timing schedule in Fig (4.5) gives a better insight into the behavior of the array in the time domain. N=3, and B=4 are used for this example. The last 10 consecutive signals appearing on each bold line are headed by the t=0 sample (boxed). Note that the lower bold lines contain 5 delay elements while the upper ones contain only one delay and a(i) $x^{[j]}$ (n-k) in Fig (4.4) implies a logical AND with a(i), $x^{[j]}$ (n-k) $\in [1,0]$. The output is a sequence of $\{|y_b(n)|_m ; b=0, 1, 2, 3\}$ as expected.

The B partial results occur in the same order of significance as the input bits are fed into the system. Hence B clock periods are needed to calculate all the partial results which exist in any output sample. The array is fully pipelined and 100% efficient in that all cells are used at every clock period.

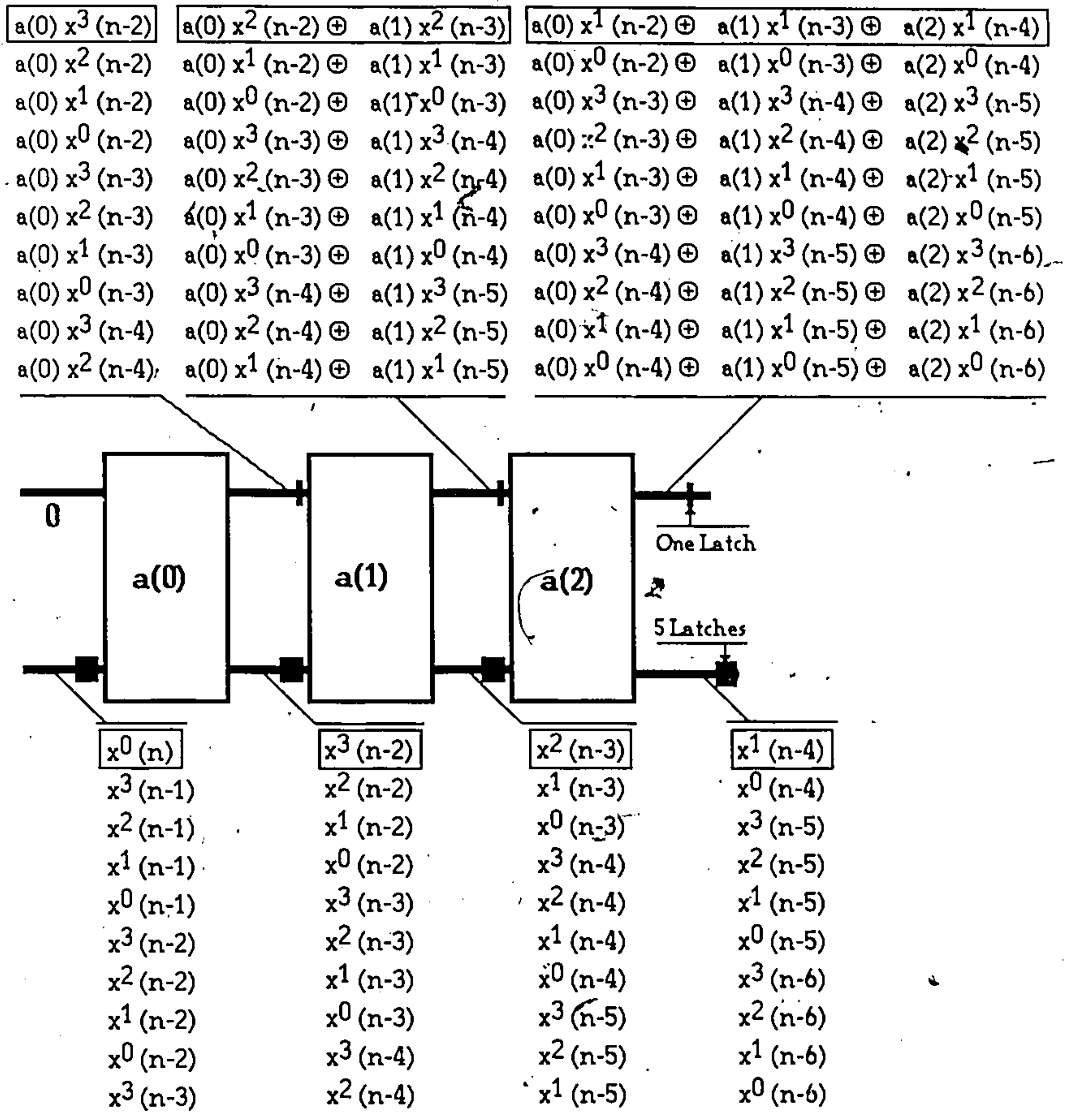

$a(0)x^3(n-2)$
$a(0)x^2(n-2)$
$a(0)x^1(n-2)$
$a(0)x^0(n-2)$
$a(0)x^3(n-3)$
$a(0)x^2(n-3)$
$a(0)x^1(n-3)$
$a(0)x^0(n-3)$
$a(0)x^3(n-4)$
$a(0)x^2(n-4)$
$a(0)x^2(n-2) \oplus a(1)x^2(n-3)$
$a(0)x^1(n-2) \oplus a(1)x^1(n-3)$
$a(0)x^0(n-2) \oplus a(1)x^0(n-3)$
$a(0)x^3(n-3) \oplus a(1)x^3(n-4)$
$a(0)x^2(n-3) \oplus a(1)x^2(n-4)$
$a(0)x^1(n-3) \oplus a(1)x^1(n-4)$
$a(0)x^0(n-3) \oplus a(1)x^0(n-4)$
$a(0)x^3(n-4) \oplus a(1)x^3(n-5)$
$a(0)x^2(n-4) \oplus a(1)x^2(n-5)$
$a(0)x^1(n-4) \oplus a(1)x^1(n-5)$
$a(0)x^1(n-2) \oplus a(1)x^1(n-3) \oplus a(2)x^1(n-4)$
$a(0)x^0(n-2) \oplus a(1)x^0(n-3) \oplus a(2)x^0(n-4)$
$a(0)x^3(n-3) \oplus a(1)x^3(n-4) \oplus a(2)x^3(n-5)$
$a(0)x^2(n-3) \oplus a(1)x^2(n-4) \oplus a(2)x^2(n-5)$
$a(0)x^1(n-3) \oplus a(1)x^1(n-4) \oplus a(2)x^1(n-5)$
$a(0)x^0(n-3) \oplus a(1)x^0(n-4) \oplus a(2)x^0(n-5)$
$a(0)x^3(n-4) \oplus a(1)x^3(n-5) \oplus a(2)x^3(n-6)$
$a(0)x^2(n-4) \oplus a(1)x^2(n-5) \oplus a(2)x^2(n-6)$
$a(0)x^1(n-4) \oplus a(1)x^1(n-5) \oplus a(2)x^1(n-6)$
$a(0)x^0(n-4) \oplus a(1)x^0(n-5) \oplus a(2)x^0(n-6)$
0
a(0)
a(1)
a(2)
One Latch
5 Latches
$x^0(n)$
$x^3(n-1)$
$x^2(n-1)$
$x^1(n-1)$
$x^0(n-1)$
$x^3(n-2)$
$x^2(n-2)$
$x^1(n-2)$
$x^0(n-2)$
$x^3(n-3)$
$x^3(n-2)$
$x^2(n-2)$
$x^1(n-2)$
$x^0(n-2)$
$x^3(n-3)$
$x^2(n-3)$
$x^1(n-3)$
$x^0(n-3)$
$x^3(n-4)$
$x^2(n-4)$
$x^2(n-3)$
$x^1(n-3)$
$x^0(n-3)$
$x^3(n-4)$
$x^2(n-4)$
$x^1(n-4)$
$x^0(n-4)$
$x^3(n-5)$
$x^2(n-5)$
$x^1(n-5)$
$x^1(n-4)$
$x^0(n-4)$
$x^3(n-5)$
$x^2(n-5)$
$x^1(n-5)$
$x^0(n-5)$
$x^3(n-6)$
$x^2(n-6)$
$x^1(n-6)$
$x^0(n-6)$

Fig 4.5 Bit serial FIR filter for N=3, B=4

As the partial results are produced, they must be weighted appropriately by the 2^b factor in eqn (4.2), and added to each other, modulo m. This can be achieved using B-1 modulo m cascaded, adders as shown in Fig (4.6) for B=4.

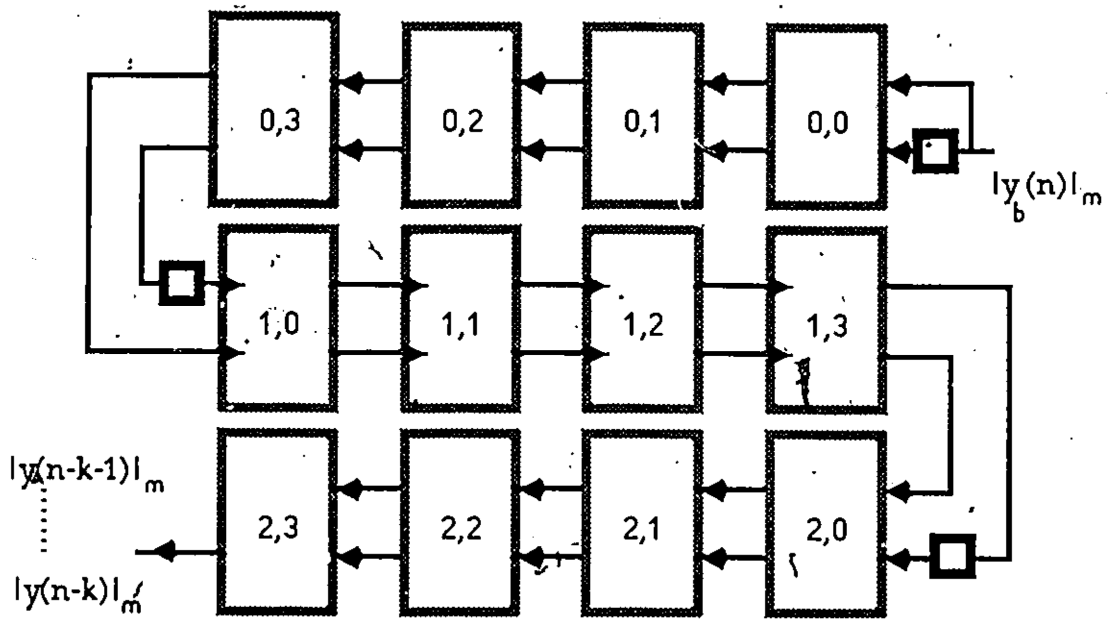


Fig 4.6 Adder block for bit serial FIR filter

The sequence $\{|y_b(n)|_m ; b\in (0,...B-1)\}$, enters the array at the top row. An additional set of latches is used to provide the required alignment for the input data and the partial results at the input of each section. The top row of the adder computes $|y_0(n)|_m \oplus 2 \otimes |y_1(n)|_m$ which is delayed at the input to the second row in order to be concurrent with $|y_2(n)|_m$. The output of the second row is $|y_0(n)|_m \oplus 2 \otimes |y_1(n)|_m \oplus 4 \otimes |y_2(n)|_m$. The above operation is repeated till all the partial results are exhausted and a final output sample is ready. The contents of the ROMs in the adder block can be determined by the following equation:

$$\text{Content}\ (i, k, \textit{Address}\ ) = \textit{Address} \oplus |2^{i+k+1}|_m \qquad (4.7)$$

where i, and k give the location of the appropriate $BIPSP_m$ cell in the adder block while *Address* determines a specific location in the $BIPSP_m$ cell memory.

Clearly, complete output samples are available B clock periods apart from each other, as dictated by the serial input method. This yields a $\frac{100}{B}$ % efficiency for the adder block. The separation of adjacent valid output samples by dotted lines is used to indicate this in Fig (4.6).

Table (4.1) gives the hardware requirements for the implementation of the *word serial/bit serial* FIR filter.

Filter cells, Fig (4.3)	Adder cells Fig (3.6)	Input latches	Throughput
N	B(B-1)	B(B-1)	$\frac{1}{B}$ sample/clock period

Table 4.1 Hardware requirements and the throughput for an Nth order FIR filter

For the cases where high speed filtering is required the above approach may be considered too slow, hence not be acceptable. To remedy this, an entire word or even multi words (block filtering) has to be presented at the input at every clock period. In light of this we will next discuss a *word serial/bit* *parallel* structure in which all the input bits are read at one time and an output sample is computed per clock cycle.

4.2.2. Word serial/ bit parallel FIR filter

If all the bits in an input word are supplied to the filter at one time, each bit can be processed in a separate array to produce a partial result [Tah'87c]. The block diagram of a *word serial/bit parallel* filter structure is given in Fig (4.7).

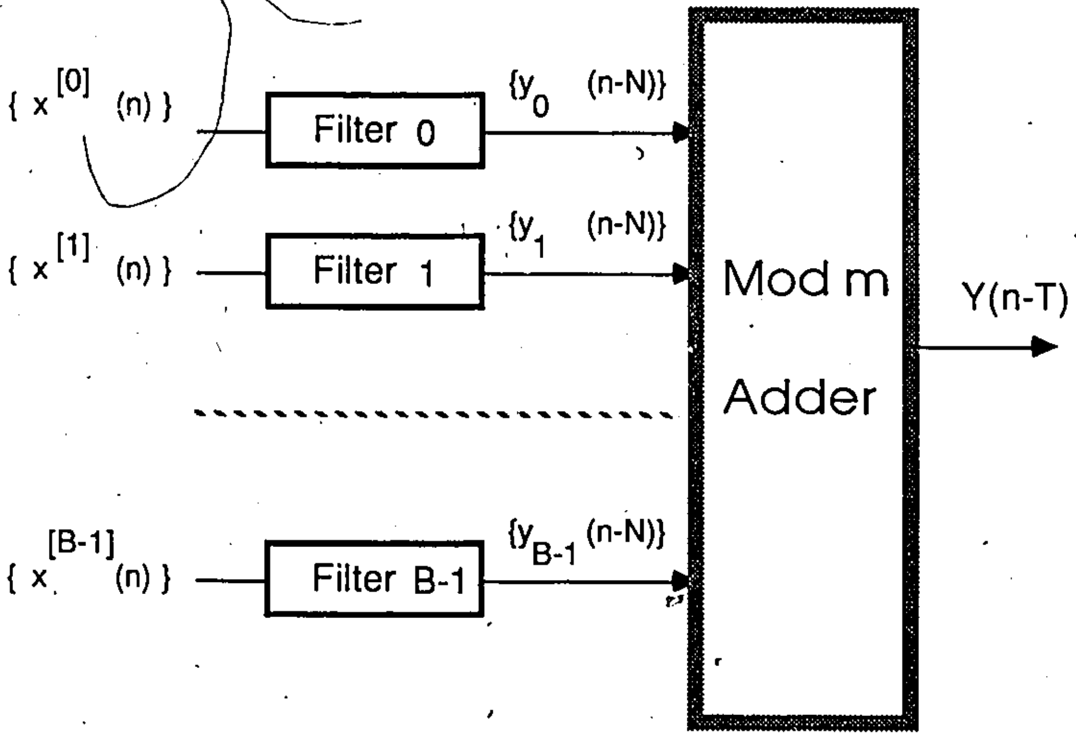


Fig 4.7 Bit parallel FIR filter structure

Each array processes one input bit producing one partial result per clock period $\{|y_b(n)|_m ; b=0, 1, 2, 3\}$. The $\{|y_b(n)|_m ; b=0, ..., B-1\}$ sequence is added together in the final adder block in order to compute an output sample according to eqn(4.2).

4.2.2.1. Word serial/bit parallel FIR filter implementation

The cells in the linear arrays resemble the one used in the bit serial FIR filter, except that the number of latches has been modified. The input to the arrays are arranged as:

$$\{\ldots x^{b}(n), x^{b}(n-1), \ldots x^{b}(n-N+1)\ldots\}$$

Only 2 latches are needed here, one to compensate for the delay in the $|y_b^{(i)}(n)|_m$ path, this provides the sliding effect required between the input data and the partial results. The bit parallel cell is shown in Fig (4.8) . The content of the ROMs can be calculated from eqn(4.6).

One array is needed to process each input bit hence B arrays are needed. At the final stage of each array, one partial output, $|y_b(n)|_m$, is produced. Fig (4.9) gives a linear array containing N cells each programmed with one of the filter coefficients.

Here all the partial results required to compute the final output are produced at the same time. The multiplication by 2^b as given in eqn (4.2), can be performed either in the arrays or inside the adder. Let us assume this multiplication takes place within the adder.

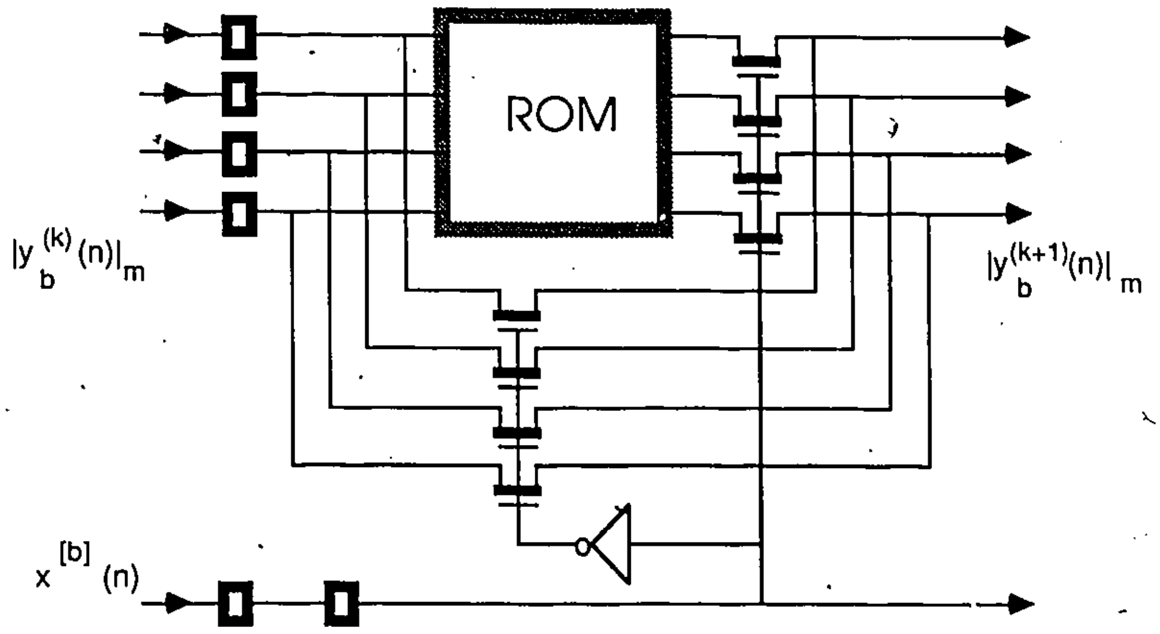


Fig 4.8 $BIPSP_m$ with a modified X data path

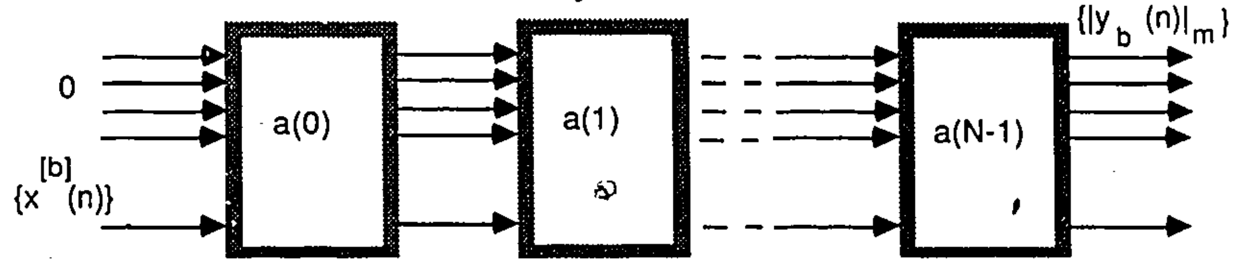


Fig 4.9 Linear array for an Nth order FIR filter

The final adder consists of B-1 mod m adders cascaded as shown in Fig (4.10). This differs from the adder discussed in section 4.2.1.1 in its input ports and timing. First $|y_0(n)|_m$ and $|y_1(n)|_m$ are added together to form

$(|y_0(n)|_m \oplus 2 \otimes |y_1(n)|_m)$. In the second stage, $4 \otimes |y_2(n)|_m$ is added to this result, this process continues until all the components are added. The adder is fully pipelined and reads one set of inputs per clock cycle. It takes B clock

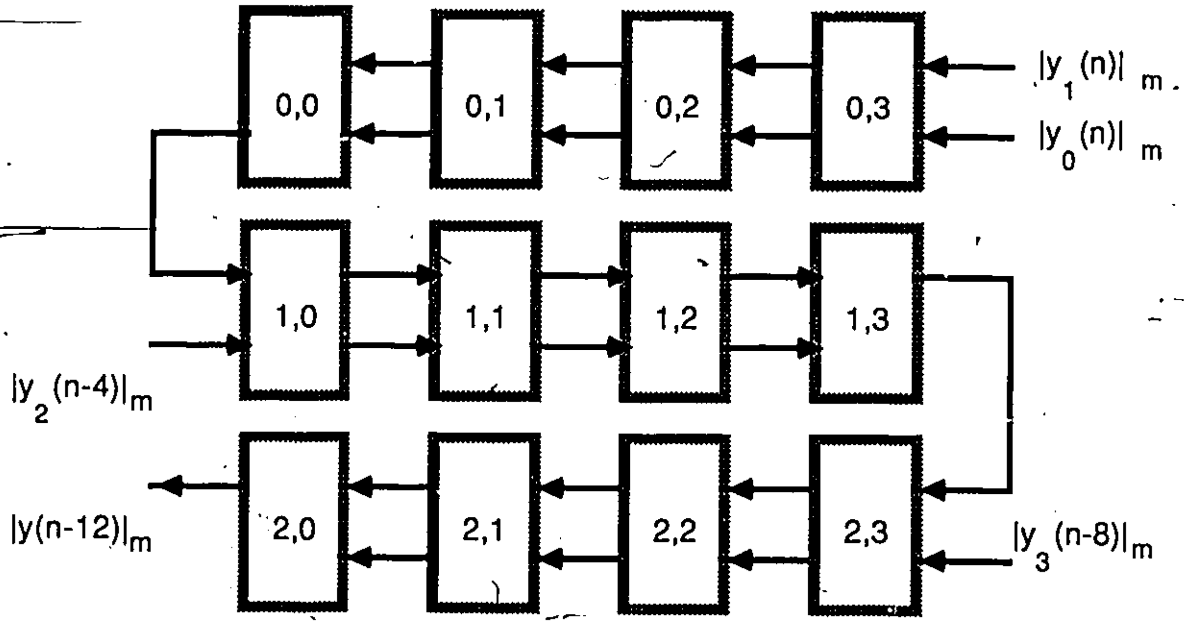


Fig 4.10 Modulo m adder for FIR filter

periods for the result of an input to appear at the output, hence $|y_2(n)|_m$ must be delayed B clock periods with respect to $|y_0(n)|_m$ and $|y_1(n)|_m$. The same delay must exist between the other partial results. The obvious method is to provide the required delays at the adder input ports, which means that B(B-1)(B-2) delays are required. A better scheme is to delay the filter inputs accordingly, this reduces the delay count to (B-1)(B-2).

The final adder can be viewed as a fixed overhead since its size and structure is not a function of the filter order, N, and only depends on B. A total of

B(B-1) adder cells, are required using this approach.

4.2.2.2. Word serial/bit parallel FIR filter without adder

Finally a simplified version of the parallel bit FIR filter with the final adder removed will be presented.

In this scheme eqn(4. 2) is rewritten in the following form:

$$|Y(n)|_m = \sum_{i=0}^{N-1}{}_m \sum_{b=0}^{B-1}{}_m 2^b \otimes (A(i) \otimes x^{[b]}(n-i)) \qquad (4.8)$$

Only a single array is used to implement eqn (4.8) , the addition being performed in a distributed manner as the partial results move along the array. The cell is a modified version of the original cell of Fig (3.6) and is shown in Fig(4.11).

An extra delay is added to the structure of Fig (3.6) in order to delay the entire input sample one clock period every B cells.

NB of the above cells can be cascaded together to form an Nth order FIR filter. The weighting by 2^b, and the inner summation in eqn (4.8), are performed in consecutive groups of B cells. The free addition property of the cell is used to perform the inner summation in eqn (4.8) as the output components are produced and move along the array. This method is more advantageous in many ways to the ones discussed previously and to the reported works in the literature. Some of the advantages of this final design over the previous filters are:

A) The entire filter is designed using only one cell and not multiple cells. It is known that improvement in solid state technology and maturity of the fabrication process reduces the failure rate of a single component; however, an increase in component

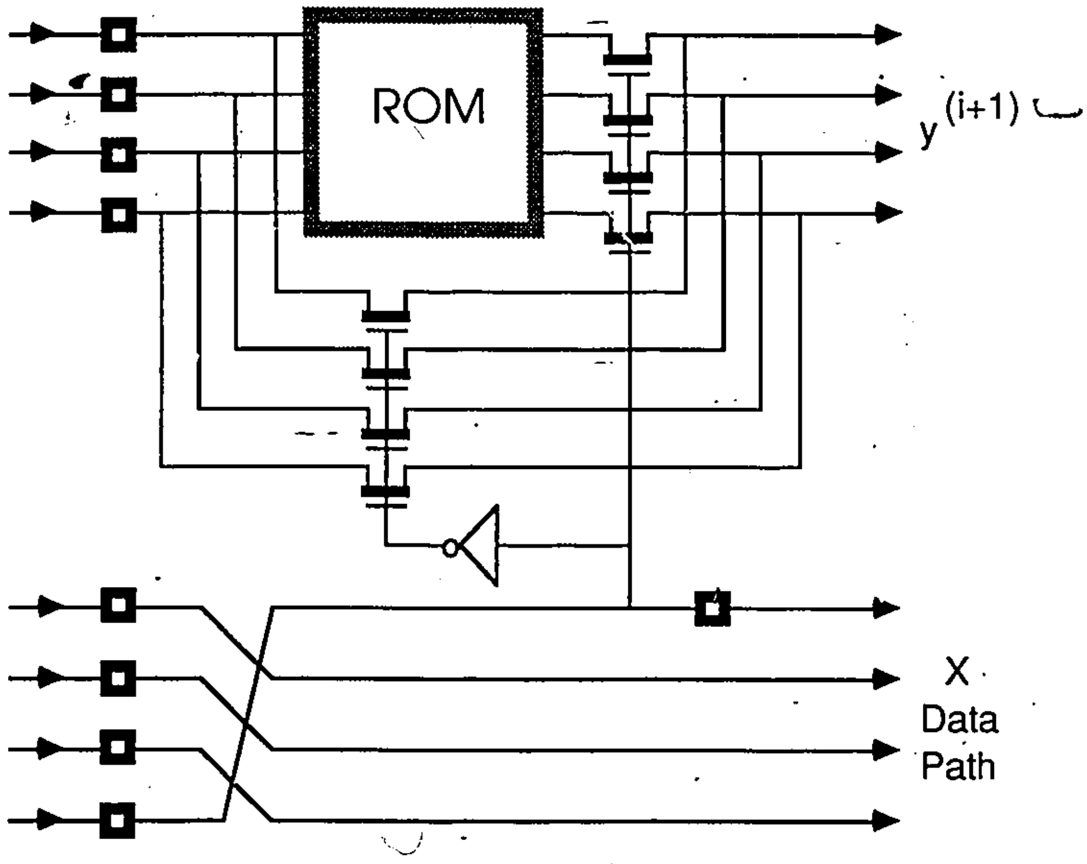


Fig 4.11 The modified cell for *word serial/bit parallel* FIR filter

count deceases yield figures. The yield can be improved by employing one time restructuring technique with the use of redundant components [Kor'86]. Using a single cell in the array, reduces the number of redundant components to be supplied with each die, and allows more efficient configurations to be employed if the ROMs are replaced with field programmable memories.

B) The modification does not increase the critical path delay over the original delay. In other words the new design is not a simple hardware/ time trade off.

C) Since the modification has not affected the critical path delay of the cells, the same throughput from the two structures is expected. The latency, however, is increased considerably because the parallel arrays are replaced with cascaded arrays. This creates a latency of NB compared to N+B(B-1) for the first case. NB>N+B(B-1) for all practical cases.

D) This method of implementation requires less hardware .

Hardware comparisons for an N^{th} order *word serial/bit parallel* FIR filter implemented using both methods are given in table (4.2).

	No. of ROMs	No. of Latches
Type 1 filter	NB	NB(2B+1)
Type 2 filter	NB+B(B-1)	NB(B+2)+B(B-1)(2B+1)

Table 4.2 Hardware comparison of the two bit parallel approaches

4.3. Comparisons

In this section our proposed technique will be compared with the approaches reported in the literature for both binary and RNS implementations.

4.3.1. RADIUS, an RNS based image processor

Nudd and his group at Hughes Research Laboratories [Fou'82],[Nud'85] reported a 2-D image processing system based on RNS arithmetic. The system, RADIUS, uses a 5x5 programmable kernel which allows real time processing on 512 x 512 pixels at 30 frame/sec. A brief description of this machine will be given first, followed by a description of a RADIUS like scheme based on the $BIPSP_m$ cell.

RADIUS is built around a four, (5) bit modulus RNS system. The input for each of the kernel rows are supplied to a 5 bit, 5 stage shift register. These shift registers convert the serial signals to parallel which are then and applied to 25 pre-programmed RAMs holding the coefficients. The 100 multiplications required by the 4 moduli are performed in one clock cycle. A simplified block diagram of the system is given in Fig (4.12). The RAM outputs of the 4 different moduli are added using 24 mod m adders. The mod m adders are implemented using 1024 x 5 bits ROMs. The reported system uses 20 copies of an LSI chip. Each containing the circuitry needed to calculate one row of the kernel. A special interface has been developed which enables PDP 11/34 and VAX machines to communicate with RADIUS through DEC UNIBUS. The host initializes the system with the proper coefficients before the operation starts. The host also feeds the input sequence and stores the output of the system during its operation.

Edge enhancement, low pass/ high pass filtering, and texture analysis are a few of the many capabilities of RADIUS.

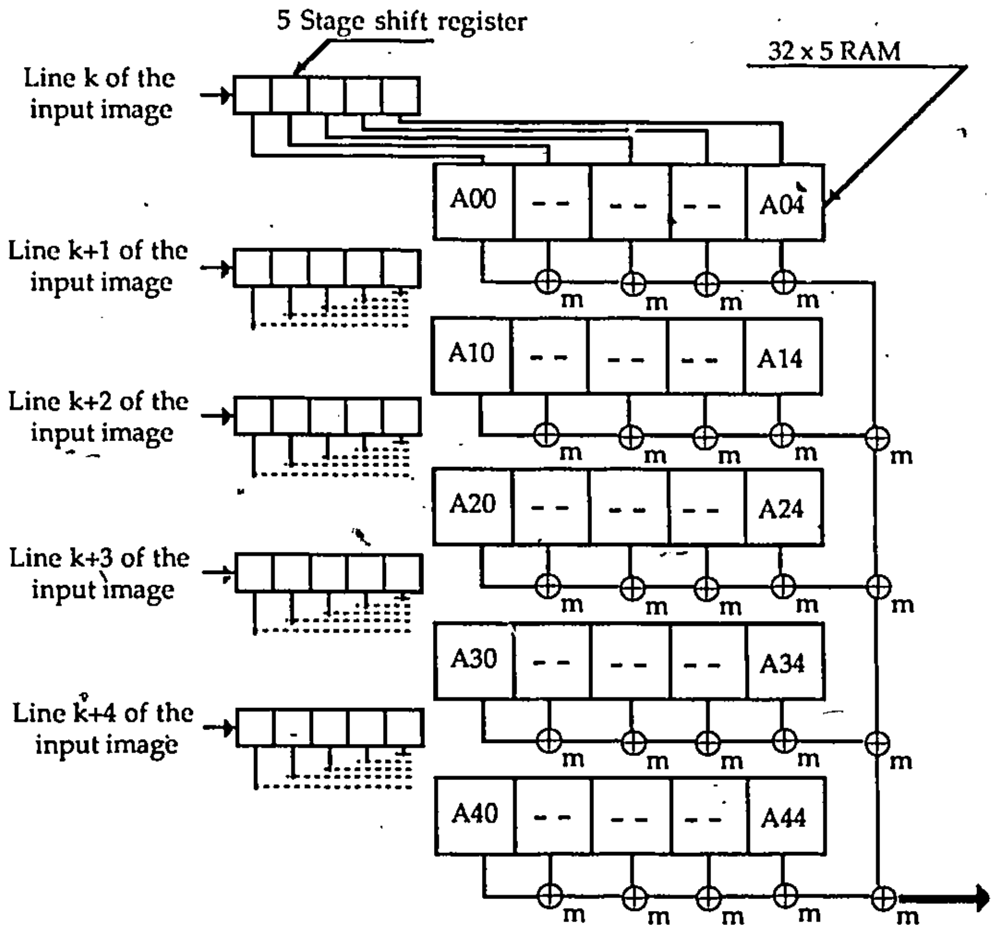

5 Stage shift register
32 x 5 RAM
Line k of the input image
Line k+1 of the input image
Line k+2 of the input image
Line k+3 of the input image
Line k+4 of the input image
A00
A04
A10
A14
A20
A24
A30
A34
A40
A44
m

Fig 4.12 A simplified block diagram of RADIUS

From the above it becomes clear that the adder trees used in this scheme are the major part of the circuit and as the kernel size increases, the number of adders increases proportionally. The number of mod m adders for an n x n kernel is $4(n^2-1)$. As an example, for a 10 x 10 kernel a total of 396 mod m adders, each requiring a 1024 x 5 bit ROM. The throughput of the system is clearly limited by the speed of the adders.

4.3.2. Implementation of RADIUS based on $BIPSP_m$

Implementation of RADIUS using the $BIPSP_m$ cell is more advantageous since the adder tree can be removed and only one type of cell is used throughout the system. To perform the operation of multiplication/addition 5 $BIPSP_m$ cells are needed for each kernel coefficient. A total of 125 $BIPSP_m$, cells with the internal ROMs replaced with the same size RAMs, interconnected as Fig (4.13) suggests. Table (4.3) compares the hardware requirements for one modulus of the two

	32 x 5 RAMs	1024 x 5 ROMs	Shift Register bits
Hughes	25	24	125
BIPSPm	125	-	-

Table 4.3 Hardware requirement for RADIUS based on different implementation

approaches.

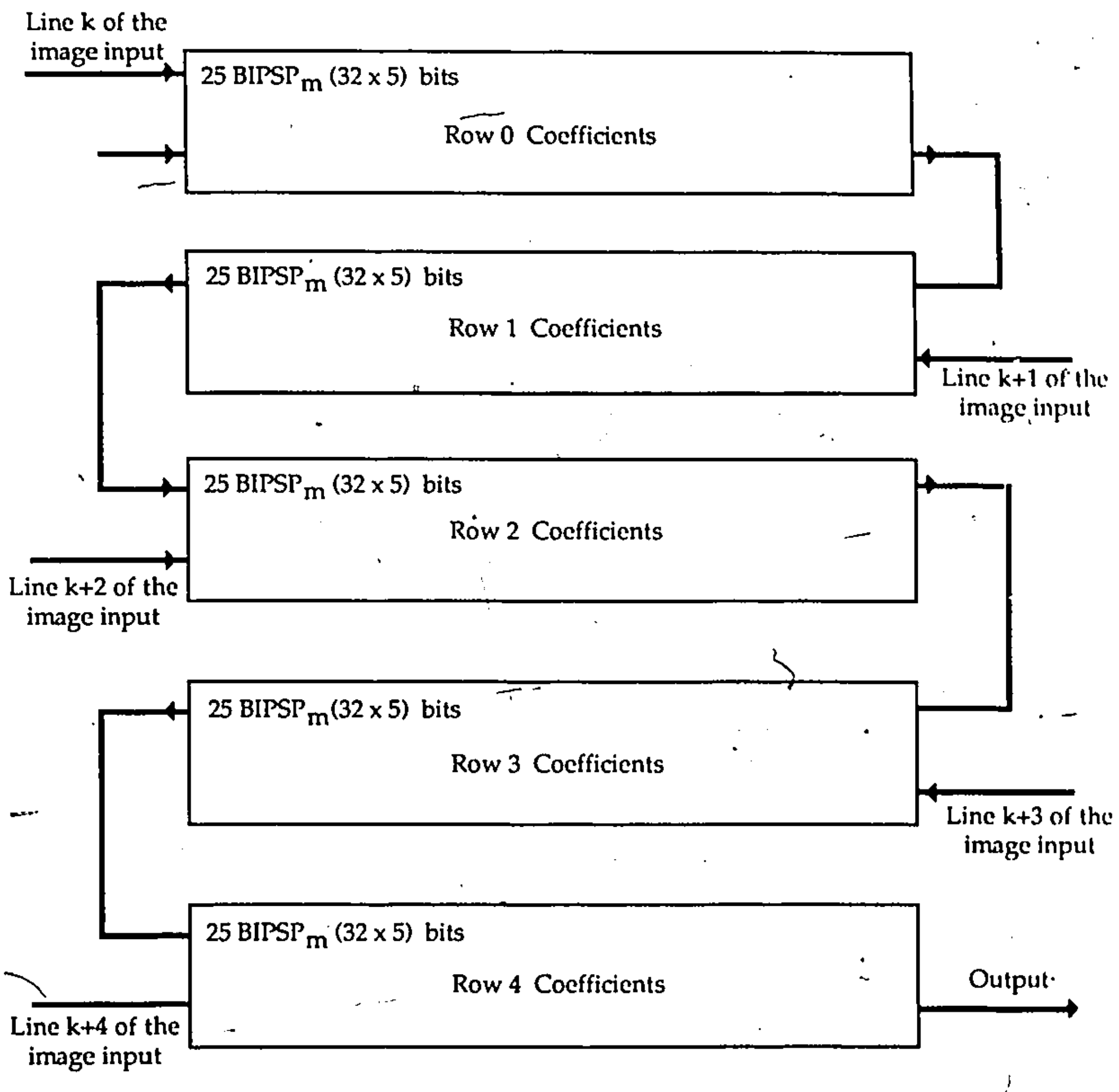


Fig 4.13 Block Diagram of RADIUS using $BIPSP_m$

From table (4.3) it is obvious that 100 RAMs each 32 x 5 bits, replace the 24 ROMs plus 125 latches. Lack of precise layout and simulation data on 32 x 5 RAMs do not allow us to provide an exact comparison between the two techniques. We will use the number of transistors in each approach as a rough measure of area estimation. Different cell structures exist for implementation of RAMs. The number of transistors needed can vary from 6 to 8 transistors per bit for static RAMs to one transistor per bit for dynamic RAMs [Gla'85]. A typical 32 x 5 RAM would contain 160 x 6+ 64 or 1024 transistors (including the decoders), using 6 transistors per bit. Table (3.2) gives a total of 6416 transistors for a 1024 x 5 ROM, and therefore the transistor count ratio of the two techniques is:

$$\frac{25 \times 1024 + 24 \times 6416 + 125 \times 8}{125 \times 1024} = 1.4 \qquad (4.9)$$

This simply means even though we have used 125 RAM based BIPSP$_m$ cells instead of 25 small RAMs and 24 single ROM adders, the total hardware cost of our approach is much less.

In the design of VLSI systems, modularity and localized data transfer are two factors affecting both the architecture and the total hardware cost of the system. It should be noted that the component cost measure given in table (4.3) does not reflect the total hardware cost, since communication and routing costs in VLSI systems are very expensive in terms of area, time, and power. This tends to increase the above ratio by a much higher number. The conclusion of this comparison is that a RADIUS like machine, based on BIPSP$_m$ cells, would require 30% less hardware and would run faster than the the existing machine.

The next section describes an investigation into the design and operation of variable coefficient systolic array correlators with binary and RNS implementations; a comparison of the two techniques is also presented.

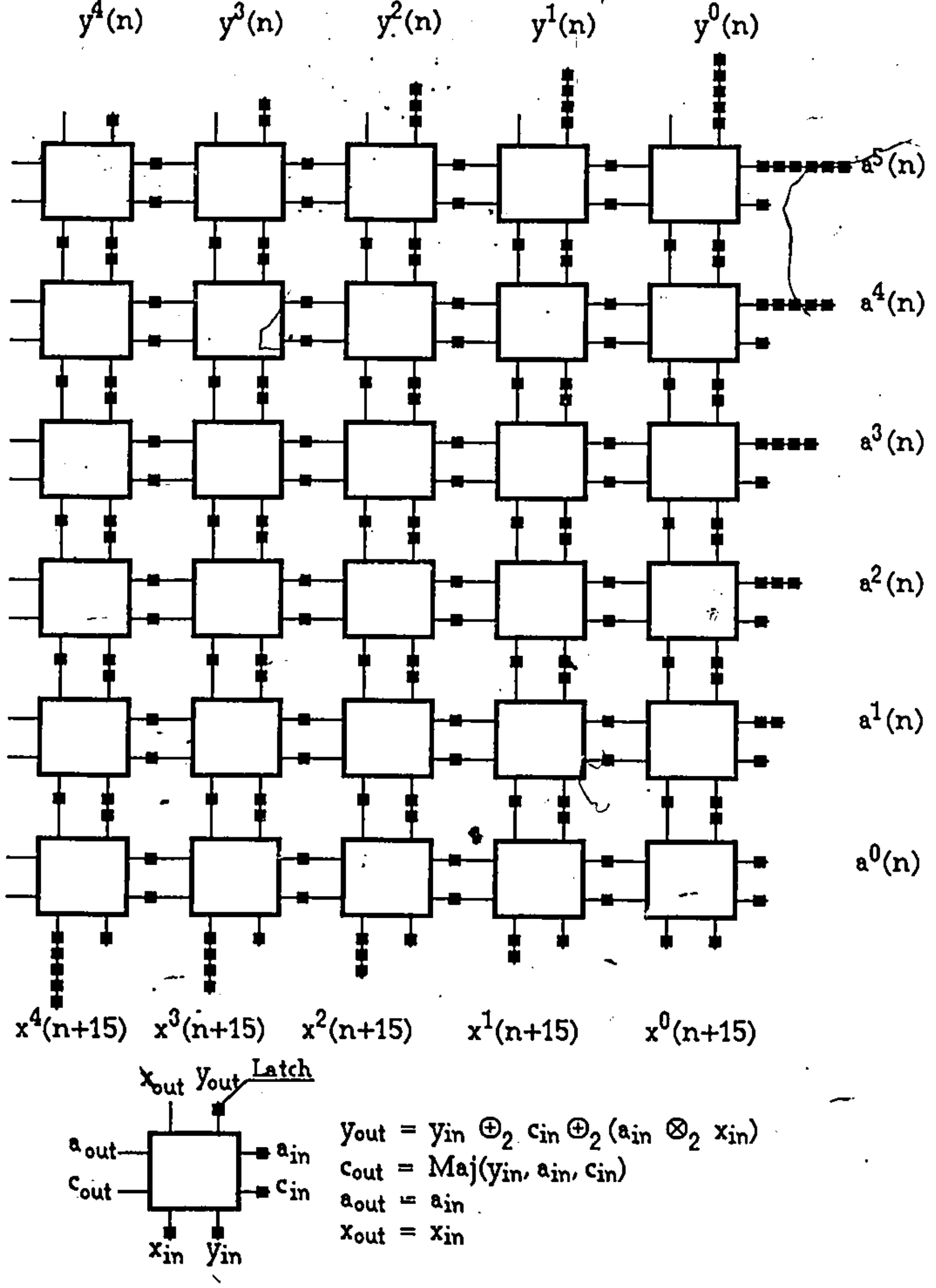
4.4. High Speed Correlator

Basically correlation can be viewed as convolution with either time reversed signal or coefficient data. In this study, the coefficients are variable unlike the fixed coefficient convolution schemes discussed earlier in this chapter. The 1-D correlation sum is given by eqn (4.10).

$$Y(n) = \sum_{i=0}^{N-1} a(i)\, x(n+i) \qquad ; a(i) \in \{0, 1\} \qquad (4.10)$$

The problem of very high speed implementation of this summation has attracted many researchers over the last decade and several different binary based designs exist which implement eqn (4.11). The most efficient approach, using systolic arrays, can be found in the following references [McC'82b], [Cor'83], [McC'83], [McC'84], [McC'86]. A commercial implementation of this algorithm has been discussed in [Chr'86]. A brief explanation of McCanny's approach will now be given to allow a comparison between the RNS and binary based correlators.

The basic cell in the binary correlator is a gated full adder, its structure and operation was explained in chapter 3. The correlator is formed by using an M x N array of gated full adders, interconnected in a rectangular fashion,



$$
\begin{aligned}
y_{out} &= y_{in} \oplus_2 c_{in} \oplus_2 (a_{in} \otimes_2 x_{in}) \\
c_{out} &= \mathrm{Maj}(y_{in}, a_{in}, c_{in}) \\
a_{out} &= a_{in} \\
x_{out} &= x_{in}
\end{aligned}
$$

Fig 4.14 Structure for binary correlator

where N is the order of the correlator and M is the output word width. Fig (4.14) shows the basic system for a 6 x 5 systolic array correlator.

An input sample x(n) enters the array from the bottom and moves up through the array interacting with the signature sequence, { a(i) ; i∈ (0,...N-1)}, one bit per word, to form the partial bits. These are also clocked up through the array. Carries generated in the process are clocked to the right towards the columns of higher significance. The partial results $\{y^{(i)}\}$ are updated, if a(i) is 1, x(n-i) is added to $y^{(i)}$ else the partial results are left unchanged. For the array to operate properly, 3 sets of skewing latches must be used on the {x(n)} input, the {a(i)} signature, and the {y(n)} output. The array is fully pipelined, which means that it is capable of reading one set of inputs and producing one set of outputs per clock cycle.

Performing variable coefficient filtering and using prestored data is naturally unmatched. However this special case deals with one bit coefficients and, in this case, general multiplication can be replaced by gated addition; which can be performed using a slightly modified BIPSP_m, as shown in Fig (4.15).

Here the signature bit is ANDed with one of the input bits in order to create the control signal for performing the following operations:

For $x^{[i]} \otimes_2 a^{[i]}=1$: $y^{(i+1)} = y^{(i)} \oplus 2^i$

For $x^{[i]} \otimes_2 a^{[i]}=0$: $y^{(i+1)} = y^{(i)}$

An array consisting of NB cells produces the result. Skewing and deskewing latches are not needed in this case.

To compare the two designs, binary and RNS, an example of M=19 is used which requires 19N gated full adders for the binary design. A 4 moduli RNS system using 5 bit per modulus provides the same dynamic range, which in turn translates to 20N $BIPSP_m$ cells shown in Fig (4.15).

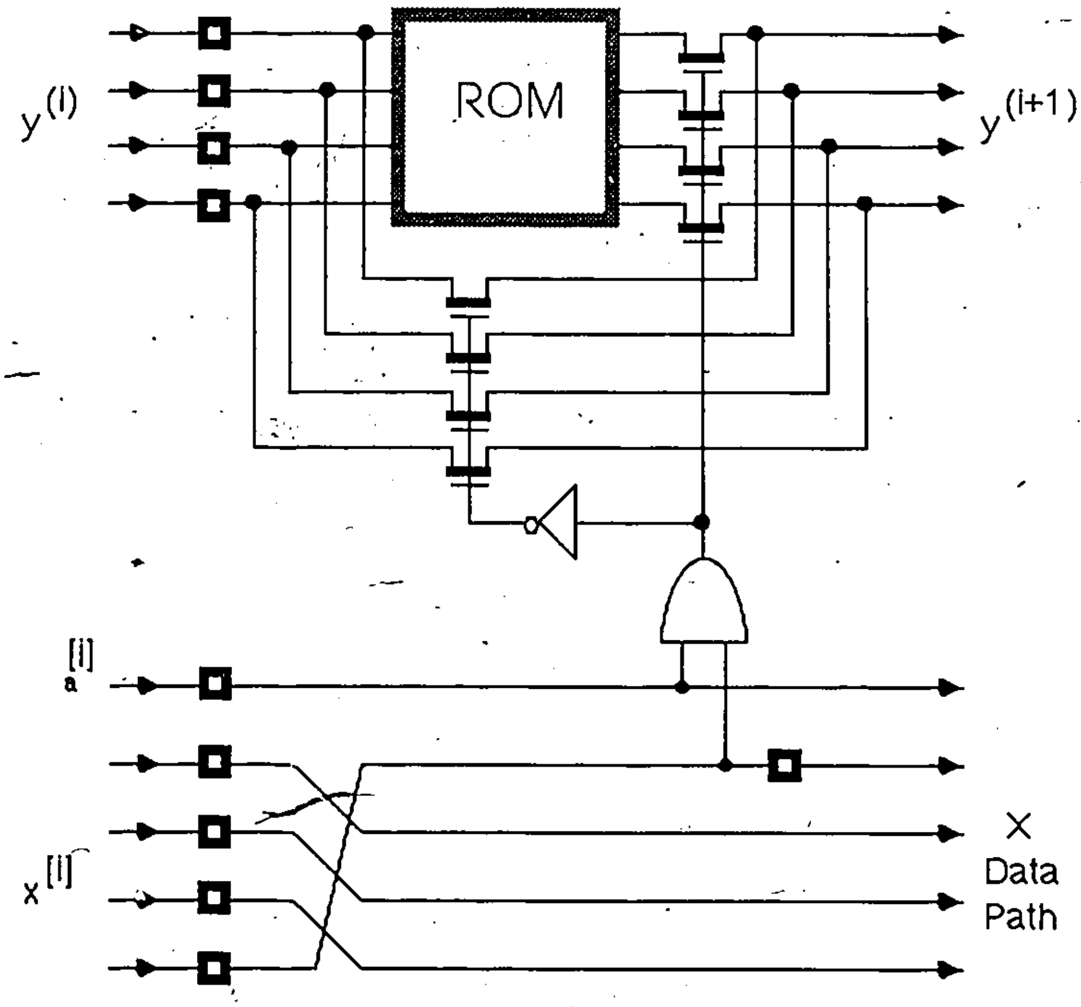


Fig 4.15 The correlator cell based on $BIPSP_m$

The fixed overhead of coding and decoding from and to binary and skewing latches must also be taken into account for the purpose of comparisons. The fact that the layout of the $BIPSP_m$ occupies more area than the binary cells makes the binary approach more attractive.

It is known that the correlation process demonstrates special properties if the signature sequence is replaced with an *m-sequence* [McE'87]. An m-sequence is a binary stream, not identically zero, that satisfies a linear recurrence whose characteristic polynomial is primitive. For example x^2+x+1, and x^3+x+1 are primitive polynomials with m=2, and m=3 respectively in GF(2^m). The corresponding recursions are :

$$x(n) = x(n-2) \oplus_2 x(n-1) \qquad (4.11a)$$

$$x(n) = x(n-2) \oplus_2 x(n-3) \qquad (4.11b)$$

These recursions generate maximal sequences of length $n=2^m-1$ which are referred to as *m-sequences* or *PN* (*pseudo-noise*). Practical *m-sequence* generators are easily implemented through the use of shift registers with feedback.

In general, correlation which is a measure of similarity between two sequences $A=(a_1, a_2... a_n)$ and $X=(x_1, x_2... ,x_n)$, is defined as:

$$C(X,A) = D-D' \qquad (4.12)$$

where D is the number of agreements and D' is the number of disagreements between A and X, hence C(X,A)=n if A and X are identical and C(A,X)=-n if A and X disagree in every component. For computational purposes, it is best to assume that the components of *m-sequence* are +1, -1, rather than 1, 0. It can be proved that *m-sequence* autocorrelation yields only 2 possible values as given in eqn (4.13) [McE'87,158].

$$C(\tau) = -1 \qquad \text{if } |\tau|_n \neq 0 \tag{4.13a}$$

$$C(\tau) = n \qquad \text{if } |\tau|_n = 0 \tag{4.13b}$$

Eqn (4.10) can be rewritten as:

$$Y(n) = \sum_{i=0}^{N-1} -(-1)^{a(i)}\, x(n+i) \tag{4.14}$$

$$= \sum_{i=0}^{N-1} a'(i)\ x(n+i)) \tag{4.15}$$

with a(i) being an *m-sequence* , $a'(i) = -(-1)^{a(i)}$. In order to benefit from this larger discrimination, both binary and $BIPSP_m$ cells must be modified.

The basic cell in the binary array, the gated full adder, can be modified by adding an extra gate so that the array performs the operation suggested by the eqn (4.15), this will not be discussed any further.

Implementation of eqn (4.15) using $BIPSP_m$ is somewhat different. The simplest form is to carry a negated version of the input, $|[-x(n)]|_m$, along the array and use the recursion given by (4.16).

$$|y_b^{(0)}(n)|_m = 0 \tag{4.16a}$$

$$|y_b^{(i+1)}(n)|_m = |y_b^{(i)}(n)|_m \oplus [-x\,(n+i)]\,[a(i) \oplus_2 1]$$

$$\oplus\, x(n+i)\ a(i) \tag{4.16b}$$

$$Y(n) = |y_b^{(N-1)}(n)|_m \tag{4.16c}$$

The extra set of bits are used whenever a subtraction is required. Therefore a(i) is used to select between the two sets of data, x(n) or $|[-x(n)]|_m$. The modifications needed here include the addition of a set of B+1 extra latches, which carry $|[-x(n)]|_m$, and the circuitry to multiplex the correct data bits. Eqn (4.15) can be rewritten as:

$$Y(n) = \sum_{i=0}^{N-1} -(-1)^{a(i)} \sum_{b=0}^{B-1} x^{[b]}(n+i) \tag{4.17}$$

which shows any a(i) is used to control all the bits in x(n+i), therefore instead of carrying the extra information, $|[-x(n)]|_m$, along the array of $BIPSP_m$ cell can be used to form the required input, x(n) or $|[-x(n)]|_m$, which is used in the following B cells. Consecutive 0's in the signature tend to create an undesirable string of positive and negative x(n) which must be avoided. This can be achieved through modifying the signature using the relation given in eqn (4.18).

$$c(i) = a(i) \oplus_2 a(i-1) \quad i \in \{0, 1, \ldots, N-1\} \tag{4.18}$$

with a(-1) = 1. This way a 0 is created for every change in the signature. It is possible to use the same x(n) throughout the array and compliment the partial results for the blocks where a subtraction has to be performed.

It is not necessary to go into much detail in order to compare the binary approach and the $BIPSP_m$ arrays for this application, the result is similar to what was discussed for the previous correlator and favours the binary structure. The reason for this seems to be the limitation of the ROM based structure in handling variable coefficient filtering. To discuss this further let us consider a widely used general fixed coefficient FIR filter implemented both using McCanny's approach and $BIPSP_m$ base arrays. Let N be the order FIR filter, S the number of coefficient bits, and a dynamic range M.

The basic building block for a fixed coefficient binary based FIR filter is a full adder rather than a gated full adder, given in Fig (3.2). Removing the gate from the cell reduces its transistor count by 6. However the structure of the array remains the same. To accommodate the S bit wide coefficients S similar arrays must be cascaded[Chr'86]. For the above example this translates to S x N x 19 full adder cells. The same filter can be implemented in a four moduli residue system, each 5 bits. Therefore 4 arrays each containing 5N cells, require a total of 20N $BIPSP_m$ cells. The result of this comparison becomes surprising when the area ratio of a full adder and $BIPSP_m$ are considered.

The layouts for the gated full adder and the $BIPSP_m$ are given in appendix A. The gated full adder contains 86 transistors and occupies an area equal to 310 x 292 μ^2 using a 3 micron double metal technology. As discussed above for this special application requires a full adder cell rather than a gated full adder which has 6 or 7% less transistors. The $BIPSP_m$ contains 293 transistors, 3.5 times as many as the full adder, however its layout using the same technology occupies an area equal to 421 x 377 μ^2. Then the area ratio of the two cells is:

$$\xi = \frac{421 \times 377}{(310 \times 292) \times 93\%} = 1.9 \qquad (4.19)$$

Note that the transistor ratio is larger than the actual layout ratio in this case, This is what was expected because more than 50% of the $BIPSP_m$ transistors form the ROMs memory plane which is very regular consumes a very small area.

Using ξ, the overall area ratio of the filters can be calculated as following:

$$\Gamma = \frac{20N \times 1.9}{19SN} = \frac{2}{S} \tag{4.20}$$

This means that FIR filters implemented using full adder cell and $BIPSP_m$ occupies the same area for S=2 while the $BIPSP_m$ based filter only requires 25% of the area if S=8.

Based on SPICE simulation the $BIPSP_m$ cell can be clocked at 41 MHz [Bir'87] and consumes 3.2 mWatts @20 MHz.

4.5.Reverse Mapping

The mapping from RNS to binary is always difficult, and has been the subject of many works on residue number systems[Sod'86]. The mathematical mapping is the Chinese Remainder Theorem (CRT) as shown in chapter 2. The CRT requires a modulo M adder, normally a drawback unless special moduli (close to a power of 2) are used[Sod'86]. For our purposes a scheme which uses the same small ring structures that we have been computing over would be ideal. The Mixed Radix Conversion procedure[Sza'67] is suitable, but produces a weighted magnitude output in an awkward form. A scheme recently disclosed [Sen'-] allows computation over the individual rings and also produces the output in bit-sliced binary. The basic concept is to iterate around a loop that contains base extension (generating a new ring) to a power of 2 modulus, followed by scaling with that same ring modulus. The output is therefore produced in slices of bits, with the number of bits equal to the log base 2 of the extended ring modulus. If we use a ring modulus of, for example, 32, then the output will be in 5-bit slices. The operations required are all individual ring operations, and are associative; therefore we can use our linear systolic structure (with small modifications in data paths) to perform the mapping.

Often the mapped data are not required over the precision of the direct sum ring. Scaling strategies [Jul'78] can be used to great effect in this situation. The following example illustrates the point:

With four 5-bit moduli (say 32, 31, 29, 27) we have greater than 19 bits of dynamic range. If we only require the mapped output to have about 10 bits of dynamic range, then we can adopt an exact division scaling strategy[Jul'78]. By carefully arranging the order of the moduli, we will be able to produce the output in bit slices with only one iteration through the scaling array and no base extension. In the example used here, we arrange the moduli in the order $\{m_1=29, m_2=27, m_3=31, m_4=32\}$ and we plan to scale by 27x31. This will reduce the dynamic range to $32 \times 29 \approx 2^{10}$.

A possible construction is shown in Fig (4.16) with the half tone blocks equivalent to 5 linearly connected cells and the solid block representing

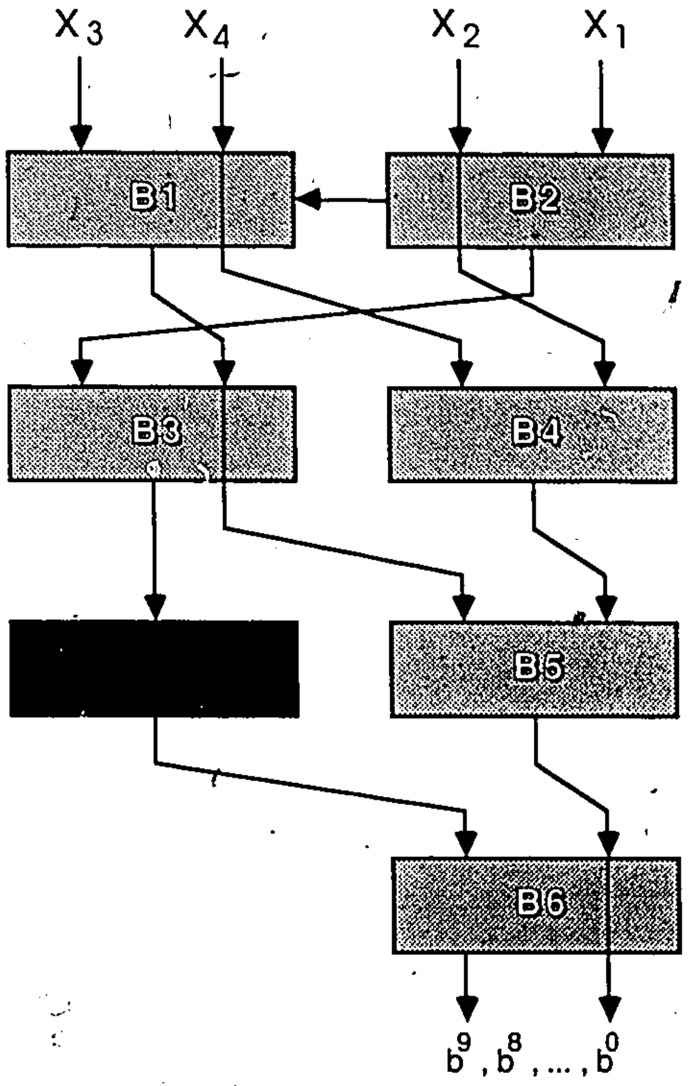


Fig 4. 16 Scaling and reverse mapping array for four 5-bit moduli

5 cascades of 5 latches each. The data line shown superimposed on the half tone blocks represents the cyclically rotated parallel data path with access to the top serial bit. The arrow between blocks 1 and 2 indicates that the serial bit for block 1 is obtained from the serial bit used for block 2, rather than its own serial bit. This allows existing data paths within each cell to be used more effectively. The output is obtained at the bottom of the array, with the

ordering of bits as shown.

The contents of each block have the form $R_i = \{(A \oplus_{m_k} B) \otimes_{m_k} K\}$, where A and B are variables and K is a constant multiplier. This is amenable to bit slicing and so each block in Fig (4.16) is capable of being sliced. The contents of the 6 blocks are given in eqns (4.21).

$$B_1 = \{(X_3 \oplus_{m3} \gamma) \oplus_{m3} [-(X_2 \oplus_{m2} \gamma)]\} \otimes_{m3} \{|m_2^{-1}|_{m3}\} \qquad (4.21a)$$

$$B_2 = \{(X_1 \oplus_{m1} \gamma) \oplus_{m1} [-(X_2 \oplus_{m2} \gamma)]\} \otimes_{m1} \{|m_2^{-1}|_{m1}\} \qquad (4.21b)$$

$$B_3 = \{(B_2 \oplus_{m1} [-B_1]\} \otimes_{m1} \{|m_3^{-1}|_{m1}\} \qquad (4.21c)$$

$$B_4 = \{(X_4 \oplus_{m4} \gamma) \oplus_{m4} [-(X_2 \oplus_{m2} \gamma)]\} \otimes_{m4} \{|m_2^{-1}|_{m4}\} \qquad (4.21d)$$

$$B_5 = \{(B_4 \oplus_{m4} [-B_1]\} \otimes_{m4} \{|m_3^{-1}|_{m4}\} \qquad (4.21e)$$

$$B_6 = \{(B_3 \oplus_{m1} [-B_5]\} \otimes_{m1} \{|m_4^{-1}|_{m1}\} \qquad (4.21f)$$

In eqns (4.21) the addition constant $\gamma = \frac{m_1 . m_2}{2}$ is used to allow rounding of the estimate to the nearest integer rather than truncation, as normally happens with exact division scaling techniques.

The set of eqns (4.21) can be simplified if ϑ_1, ϑ_3, and ϑ_4 are defined as given below:

$$\vartheta_1 = |m_4^{-1}|_{m1} \otimes_{m1} |m_3^{-1}|_{m1} \otimes_{m1} |m_2^{-1}|_{m1} \qquad (4.22a)$$

$$\vartheta_3 = |m_3^{-1}|_{m1} \otimes_{m1} |m_4^{-1}|_{m1} \qquad (4.22b)$$

$$\vartheta_4 = |m_3^{-1}|_{m_4} \otimes_{m_1} |m_2^{-1}|_{m_4} \tag{4.22c}$$

It can be seen that ϑ_1, ϑ_3, and ϑ_4 can be calculated in advance. Using a set of constants, as shown in eqns (4.23) the input to the decoder, X_1, X_2, X_3, and X_4 can be mapped to $\bar{X}_1$, $\bar{X}_2$, $\bar{X}_3$, and $\bar{X}_4$ employing only a single BIPSP$_m$ cell.

$$\bar{X}_1 = (X_1 \oplus_{m_1} \gamma) \otimes_{m_1} \vartheta_1 \tag{4.23a}$$

$$\bar{X}_2 = (X_2 \oplus_{m_2} \gamma) \tag{4.23b}$$

$$\bar{X}_3 = (X_3 \oplus_{m_3} \gamma) \otimes_{m_3} |m_2^{-1}|_{m_3} \tag{4.23c}$$

$$\bar{X}_4 = (X_4 \oplus_{m_4} \gamma) \otimes_{m_4} \vartheta_4 \tag{4.23d}$$

Now the decoding and scaling operation can be performed using mod m adder blocks if the free constant multiplication property of the cells are used. Hence the operation of $R_k = \{(A \oplus_{mk} B) \otimes_{mk} K\}$ used in Fig (4.16) can be replaced by $\bar{R}_k = A \oplus_{mk} B \otimes_{mk} K$ requiring 25% hardware for this example. The details of the operation is given in eqns (4.24) and appendix B provides the software simulations of the scheme.

$$\bar{B}_1 = \bar{X}_3 \oplus_{m_3} \{[-\bar{X}_2] \otimes_{m_3} |m_2^{-1}|_{m_3}\} \tag{4.24a}$$

$$\bar{B}_2 = \bar{X}_1 \oplus_{m_1} \{[-\bar{X}_2] \otimes_{m_1} \vartheta_1\} \tag{4.24b}$$

$$\bar{B}_3 = \bar{B}_2 \oplus_{m_1} \{[-\bar{B}_1] \otimes_{m_1} \vartheta_3\} \tag{4.24c}$$

$$\bar{B}_4 = \bar{X}_4 \oplus_{m_4} \{[-\bar{X}_2] \otimes_{m_4} \vartheta_4\} \tag{4.24d}$$

$$B_5 = \overline{B}_4 \oplus_{m_4} \{[-\overline{B}_1]\} \otimes_{m_4} |m_3^{-1}|_{m_4}\} \qquad (4.24e)$$

$$B_6 = \overline{B}_3 \oplus_{m_1} \{[-B_5] \otimes_{m_1} |m_4^{-1}|_{m_1}\} \qquad (4.24f)$$

4.6. Summary

This chapter introduced a general technique for implementing DSP operations, based on intensive multiply/add requirements, over finite rings. The implementation is based on bit-slicing the basic inner product sum processor. The BIPSP$_m$ yields to a linear systolic implementation of any DSP algorithm that requires intensive multiply/add operations. Examples of the use of the technique to implement a fixed coefficient FIR filters, and variable coefficient correlators suitable for many communication requirements, has also been presented. Finally, it has been demonstrated that the basic cell can also be used in reverse mapping from the finite ring structure to more conventional representations, such as binary. An example is given illustrating the use of pre-scaling to limit the computational overhead in reverse mapping.

CHAPTER 5

Distributed Fault Detection and Correction in RNS

Abstract

This chapter presents a distributed fault detection technique suitable for arrays consisting of $BIPSP_m$ cells, discussed in previous chapters. The technique is based on parity bit checking and requires a small extra hardware. It is also shown that faults can be corrected using the information provided by the error detecting circuitry.

Finally an efficient procedure for testing these arrays are given and the advantages of the approach is discussed.

5.1. Introduction

Achieving reliable operation becomes increasingly important and difficult with growing number of devices per chip. Imperfect manufacturing process, subsequent wearout in the field and intermittent environmental effects are the sources of unreliable operation of the VLSI chips. Although improvement of the manufacturing and testing process, to minimize manufacturing defects, helps in raising the reliability levels, but it has been very costly to achive and difficult to implement and does not guarantee reliable operation in the field. Also improvement in the reliablity levels can be achieved if the design incorporates some fault tolerant techniques i.e., can detect and correct errors during its normal operation. A more practical approach combines manufacturing testing and fault tolerance in order to relax the test requirments or design for multiple error detection and correction.

This chapter presents techniques which improves the reliability of the array, consisting of $BIPSP_m$, during run time operation. A simple procedure for comprehensive testing of the arrays will be also discussed.

The use of residue arithmetic for fault detection/correction is well known [Bar'73], [Man'72], [Sza'67]. The interest in the RNS in the early days of computer technology was for its properties as a softly degradable numbering system. This was an important issue for the undependable components (vacuum tubes, relays) that were used in the early machines. The commercial use of transistors, and the associated increase in reliability, removed the necessity for such measures and so interest faded somewhat in the use of the RNS in error detection/correction systems. Over the past decade there has been a small but steady resurgence of interest in the use of RNS fault-tolerant systems [Etz'83], [Tay'84], but the overhead involved is quite large [Jen'83]. The approach usually taken is to define a redundant residue system in which the independent operations over the various rings or fields of the system can be in error in any one of the channels (individual ring or field). Through sub-projection techniques, the faulty channel can be located and removed from the isomorphic mapping process that converts the RNS number back to the normal weighted magnitude number representation. The projections require extensive hardware, though the system of error detection also provides most of the correction mechanism [Jen83]. The fact that the RNS operates over entirely independent channels provides this natural ability for error isolation.

In this chapter we re-examine fault-tolerant computations over finite rings and fields. We will show that, using a new approach, in which fault detection is a separate, distributed, process within each computational element, that the fault-tolerant mechanism can be produced in a different and more efficient way than previous procedures. The natural error isolation of the RNS is preserved using this new technique.

Section 5.2 gives the basic definition and terminology related to this field. Approaches used to improve the reliability of the system have also been discussed. Section 5.3 discusses the modification required to the cell which enables the simple fault detection mechanism to operate. We also show, in this section, that fault avoidance and fault-tolerant techniques can be used for a higher efficiency of the structure. A fault model for the single cell used as a building block in an entire processing structure, is based on the properties of the single cell and no block in the system need be more reliable than any other. Section 5.4 discusses a fault-tolerant system based on this structure.

5.2. Errors, Faults, and Reliability

In this section the various types of errors that are encountered and appropriate approaches for designing reliable systems will be discussed. We will also present a simple modification to the BIPSP$_m$ cell which enhances the operation in terms of error detection.

5.2.1. Errors

An erroneous output may be caused by fault(s) on the chip which may be the result of improper design, imperfect manufacturing process, etc. Detecting a high percentage of the faults in this category requires rigorous testing and is expensive and time consuming. Even satisfactorily tested chips may not be flawless or they may become damaged during operation. Transient effects, such as alpha particle penetration; cosmic radiation; transients ~~on the~~ power lines; general logic switching noise, are sources of soft errors and in some cases they may damage the chip permanently [Abr'86], [McE'85], [Sar'84]. A more reliable system can be designed by incorporating techniques which identify and correct erroneous outputs, such as the use of redundant coding techniques.

5.2.1.1. Codes

The output of a system can be encoded so that the output takes only a subset

{S} of all possible values {U}, during its normal operation (operation without errors). The appearance of any output Y∉ {S} indicates a failure in the system. However the Y ∈ {S} does not guarantee an error free output [Wak'78]. An output "C" is said to be code word if C∈ {S}, otherwise it is referred to as a noncode word.

In general, an error detecting code is chosen so that likely failures produce a noncode word output. When a fault generates Y∉ {S}, the error is said to be a detectable error. However, if Y ∈ {S} the error is a non-detectable error.

The output information can be simply interpreted in a decoder which distinguishes between code words and non-code words. If the decoder only notifies the reception of a non-code word, the system is said to have error detecting capability. It is also possible to design a decoder that attempts to associate a non-code word with a code word that was likely to have been generated. In this case the system has error correction properties. Codes are, in general, divided into two groups: systematic where the code digits constitute a separate block as shown in Fig (5.1) [Lin'70], and non-systematic codes where the code digits and data are not blocked independently.

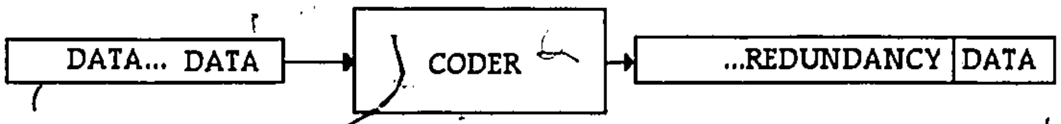


Fig. 5.1 Systematic codes

An example is the set of parity check codes which are used to detect/correct errors where no arithmetic operations take place on the data, i.e data communication and data transfer to and from memory. Although the parity-check codes are useful for checking data transmission and storage, they are not preserved by arithmetic operations [Pra'74]. For example, when data in a parity-check code are used as the input to an adder, the output, in general, is not a code word.

Definition [Wak'78]

A code {S} is said to be preserved or closed under a binary operation ✱, if for X,Y ∈ {S} the following is true :

$$X \ast Y \in \{S\}$$

Arithmetic codes are closed under arithmetic operations and are generally classified into separate, and nonseparate codes. Fig (5.2) shows the operation of addition on a separate arithmetic code in which the information and the check are processed separately. Assume the check corresponding to the information symbol I be denoted by $C(I)$, so a code word is of the form $\{I, C(I)\}$. There are two operations involved in the addition of the code words. The operation of "+" is the ordinary addition performed on the information part and the operation of "✣", explained later in this section, is performed on the check part. The advantage of separate code is that the arithmetic and checking operations can be performed in parallel. Therefore, the speed of the system does not deteriorate by the addition of redundant bits to check digits.

Peterson [Pet'58] has shown that all separate codes are equivalent to residue codes in which $C(i)$ is $|I|_m$ and the operation "✣" is addition modulo m, $\oplus_m$,where m is called the check base and divides the dynamic range M. From the point of view of error detection and correction, Redundant RNS (RRNS) is an arithmetic code which is closed under the binary operations of addition, subtraction, and multiplication. Specific details of RRNS will be discussed in section 5.4.1.

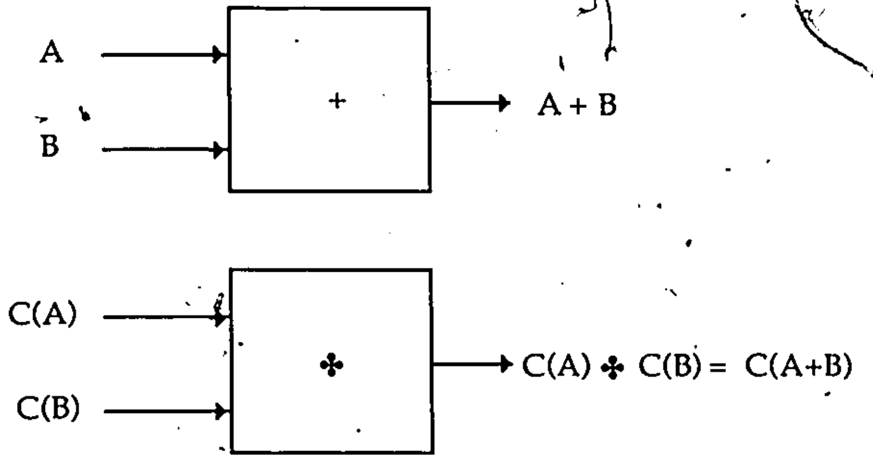


Fig 5.2 Separate adder and checker

5.2.2. RELIABLE SYSTEMS

A reliable system provides correct results in spite of faults having occurred. Such systems can be designed based on one of the following strategies[Sza'67]:

1) fault avoidance (fault intolerance) .

2) fault-tolerance

1. *Fault avoidance* : Fault avoidance seeks to reduce the possibility

of failures occurring in the system. This is achieved through the use of sophisticated design rules. Using a set of special 'high yield' design rules on some parts of the chip enables one to assume that those parts are fault free; this sometimes allows more straightforward procedures to be used on the remainder of the chip in terms of detecting/correcting faults. Mangir and Avizienis [Man'82] discuss the yield improvements when this technique is used on the interconnections between fault-tolerant cells. Designing a highly reliable system based only on this philosophy, however, will be difficult and costly.

2. *Fault-tolerance* : Fault-tolerance is achieved by the use of redundancy, and is usually split into two categories: temporal redundancy, and physical redundancy. In temporal redundancy, once an error is encountered certain procedures are activated which re-compute the result. This may be achieved by reusing the faulty system if the probability of transient failures are high. Physical redundancy, on the other hand, is achieved through the use of extra hardware[Sie'84]. Fault masking is one of the approaches in this latter category where all of the redundant components are active at all times. When a fault occurs, the effect of the faulty component is instantaneously and concurrently masked by redundant circuits, There are no separate steps for fault detection followed by correction. An example of the fault masking technique is to triplicate the original circuit (TMR), a technique ~~first proposed~~ by von Neumann [Neu'56]; a reliable majority voter

on the output decides between the end results [Tam'84]. This leads to a three fold increase in hardware. TMR can be generalized to N-Modular Redundancy (NMR), in which an odd number of modules, t, are used to mask $\frac{t-1}{2}$ failures [Mat'70].

A combination of masking and standby redundancy called hybrid redundancy has been described in [Mat'70] in which TMR or NMR plus one extra spare are used. This way the faulty module can be replaced by a standby spare.

In this chapter the main interest is in correcting isolated 'soft' errors, though our technique will also handle hard errors. We will combine fault avoidance techniques with physical redundancy to form a reliable signal processing system. Our approach is to use fault avoidance by constructing fault detection circuitry around the generic cell; the 'fault free' circuitry comprising a small fraction of the cell hardware.

A major component of the structure, introduced in chapter 3, is the ROM table. Fault detection and correction techniques for semiconductor memories such as various coding strategies, have been reported in the literature [Sar'84], [Tan'84], [McE'85]. An efficient scheme has been described[Che'85] which uses three parity bits to detect faults in ROMs. In this section we will show how the use of two parity bits allows fault detection in the ROM and the steering switches when the ROM is used in a linear array [Tah,87a]. In this chapter we use an established fault model given by Chen and Abraham [Che'85] for NMOS and CMOS ROMs with multiple-dimension decoders and multiplexers. This fault model covers a single permanent or non-permanent fault given below:

(1) A single line stuck-at-0.

(2) A single line stuck-at-1. (except word select lines)

(3) A short between two adjacent lines.

(4) A short between two orthogonal lines.

(5) An extra/missing device in the decoder or word array.

Also it is assumed that a short between two lines always results in the logical ANDing or ORing of the logic values of the two lines.

5.3.Fault Detection in BIPSP$_m$

Fig (5.3) shows the modified version of the cell. The ROM wordlength and input/output lines are increased by two bits; a checking circuit, consisting of two gates, has been added to the original circuit.

The inputs to the cell are a set of data lines $\{y^{(i-1)}, X^{(i-1)}\}$, a parity bit $P_{con}^{(i-1)}$ and a fault bit $F^{(i-1)}$. Input data lines to the ith stage of the systolic array, $y^{(i-1)}$, are used to look-up the output data $y^{(i)}$. At the same time, two parity bits are looked-up at the same location. The first is the output parity bit, $P_{add}^{(i)}$, which we refer to as the 'content parity check'. This parity check is calculated based on the following:

$$P_{con}^{(i)} = y^{[0](i)} \oplus_2 y^{[1](i)} \oplus_2 y^{[2](i)} \oplus_2 y^{[3](i)} \qquad (5.1)$$

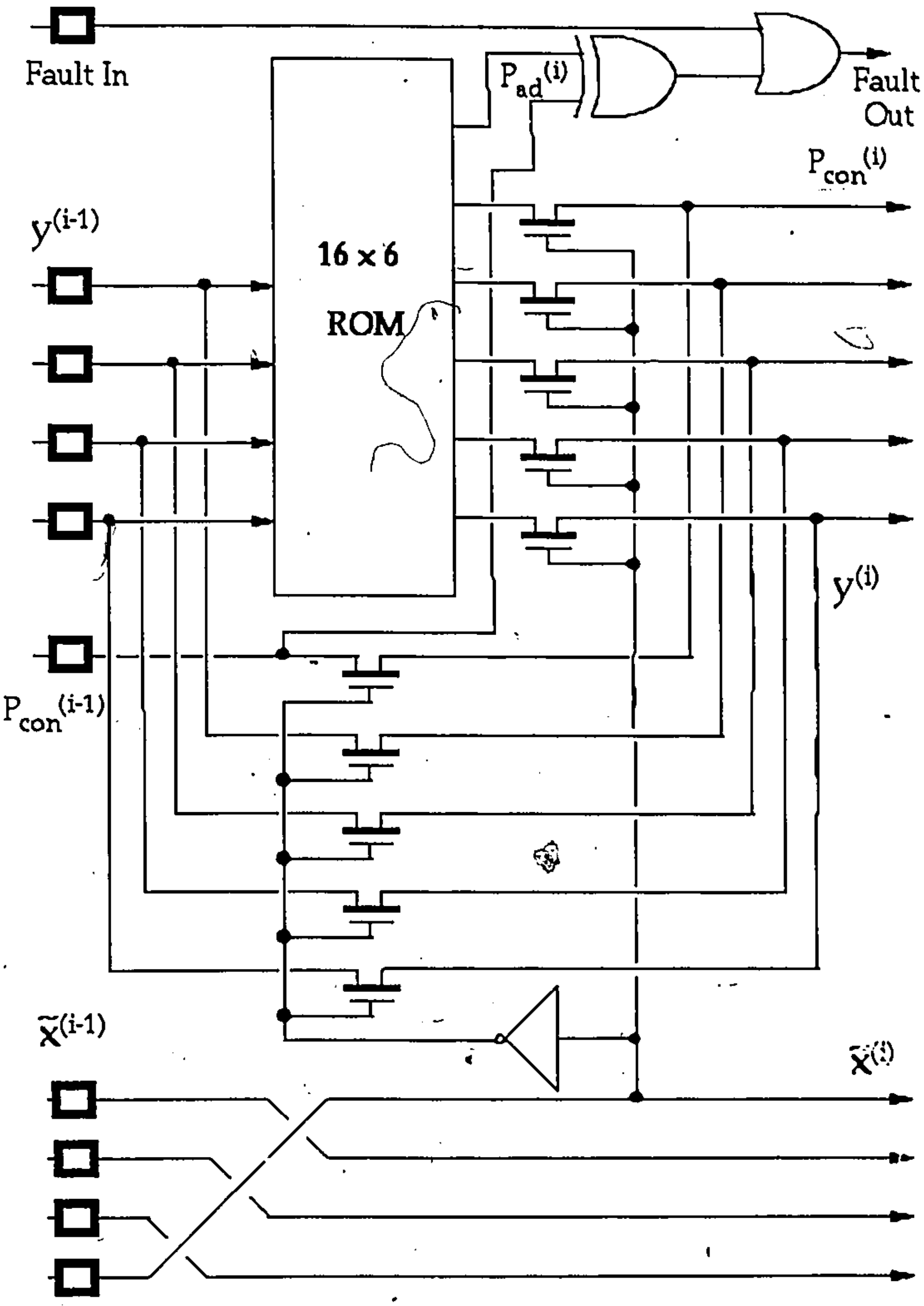


Fig 5.3 Systolic Cell with Fault Detection Circuit

where $y^{[j](i)}$ is the jth bit of $y^{(i)}$. Note that $\oplus_2$is the exclusive-OR function. The second parity bit, $P_{add}^{(i)}$, which we refer to as the 'address parity check', is generated as follows:

$$P_{add}^{(i)} = y^{[0](i-1)} \oplus_2 y^{[1](i-1)} \oplus_2 y^{[2](i-1)} \oplus_2 y^{[3](i-1)} \qquad (5.2)$$

In the zero error case, the fact that the address parity check in any cell is equal to the content parity check in the previous cell is the key to the operation of this fault detection structure. The fault detection circuitry generates the fault signal:

$$\textit{Fault Out} = \textit{Fault In} \vee \{ P_{add}^{(i)} \oplus_2 P_{con}^{(i-1)} \} \qquad (5.3)$$

where *Fault Out* is true if there has been a fault detected in any of the previous cells. This signal is pipelined through the array with the fault detection logic acting in a 'daisy chain' fashion. The error signal travels along with the processed data, so that, at the end of the chain, the erroneous output samples are synchronously flagged. Fig (5. 4) shows how the consecutive cells in the array perform the checking operation. As was discussed before only the content information and P_{con} appear at the output of each cell while P_{add} is compared with P_{con}, a single bit received from previous cell. The following section discusses the error detecting capabilities of the $BIPSP_m$ modified as explained above.

In order to determine the detection properties for singly occurring faults, we can partition the cell into four different categories and investigate each separately as below:

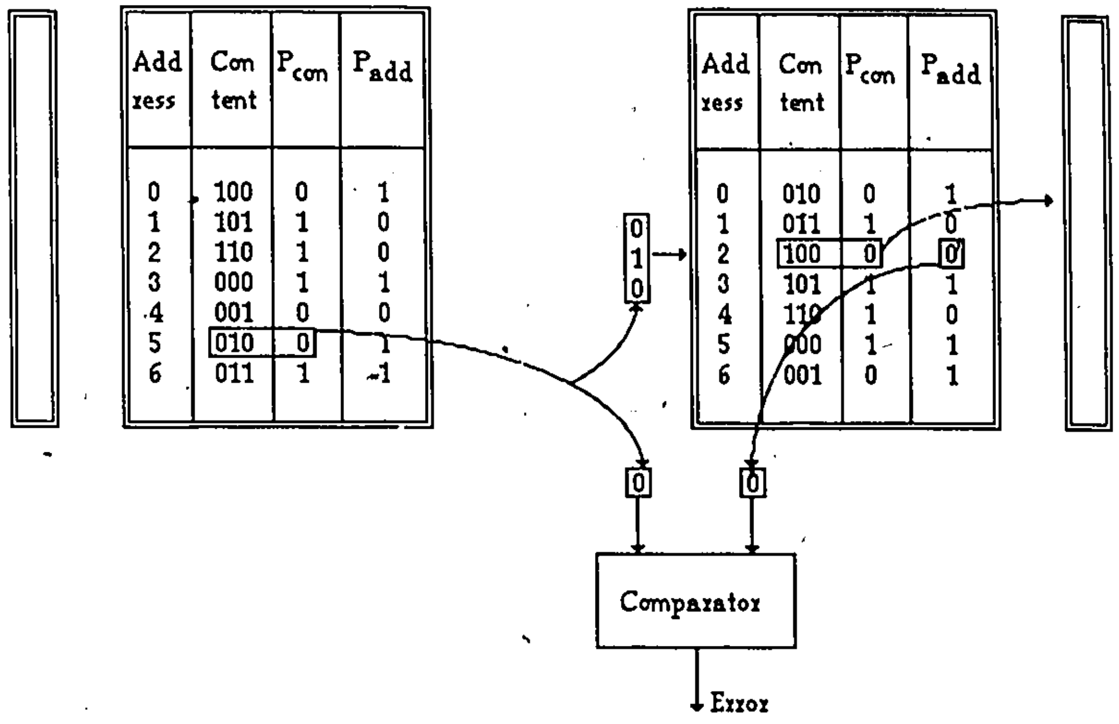


Fig 5.4 Operation of the cell in an array

A) Consider the elements which only contribute to the generation of data bits $y^{(i)}$. These elements are the memory planes which store this information, the lines to and from steering switches and the latches on these lines. A fault in any one of these elements (by assumption) will change only one output bit, therefore the check circuit will flag these and the fault will be detected.

B) Consider the elements which only contribute to the generation of parity bits. These elements are the parity memory planes and the fault flag, *Fault Out* . Since we have assumed a maximum of one failure in the cell the only fault situations that can occur with these elements are erroneous detection of a fault when the data bits are in fact valid or no fault detection when the data is in error. In the first case the false signal causes the cell result to be flagged as erroneous; this does not jeopardize the operation since, by assumption, all the other channels, including any redundant channel, are operating correctly. The other case requires both an output fault and check circuit fault at the same time. By the single fault assumption, this can not occur.

C) Consider the elements which contribute to the generation of both the fault flag *Fault Out* and the output data $y^{(i)}$. These elements are the ROM row or column decoder. Chen et al [Che'85] have classified the functional failures of the decoder into 3 groups:

group (1): Instead of the correct word select line L_i, one or more word select lines L_j's are selected.

group (2): In addition to line L_i, one or more L_j's are selected.

group (3): None of the lines are selected.

Chen also has provided a theorem showing that the address of each incorrectly selected location will differ by exactly one bit from the address of the correct location, as a result of group (1) or group (2) failures. In the following theorem we show that a faulty decoder, with a single fault, will be detected.

Theorem 1: Let the input to the decoder be the set

$$Y^{(i)} = \{ y^{[0](i)}, y^{[1](i)}, ..., y^{[k](i)}, ..., y^{[B-1](i)} \} \tag{5.4}$$

and assume that the faulty decoder misinterprets one bit of the input data, this fault will be detected.

Proof: Suppose that the input set selects location R with an address parity of P_y. Assume, as a result of the fault, that the location selected is

$$\bar{Y}^{(i)} = \{ y^{[0](i)}, y^{[1](i)}, ..., \bar{y}^{[k](i)}, ..., y^{[B-1](i)} \} \tag{5.5}$$

with a parity bit, $P_{\bar{Y}}$ Since the set Y and $\bar{Y}$ differ in one bit, then their parities, P_Y and $P_{\bar{Y}}$ will not be equal and the fault will be detected.

Failures classified under group (3) are expected to produce similar logic values at the output, i.e all 1 or all 0. Hence faults in this group can be detected by use of the appropriate parity type i.e odd parity. For example consider a ROM storing operations based on a 5 bit modulus, let us assume as the result of no selection the ROM output to be all 1's or 0's. It is obvious that an odd parity scheme detects group (3) faults.

D) The $X^{[i]}$ bits and their corresponding latches are not checked in this scheme. Next two suitable approaches for checking these latches will be described.

1) The fact that these latches comprise a small part of the entire cell makes them a suitable candidate for fault avoidance techniques such as sophisticated design rules, discussed earlier in this chapter.

2) Introduction of a second parity bit x_p for the X input such that

$$x_p^e = x^{[0]} \oplus_2 x^{[1]} \dots \oplus_2 x^{[B-1]} \qquad \text{for even parity} \qquad (5.6)$$

$$x_p^o = x^{[0]} \oplus_2 x^{[1]} \dots \oplus_2 x^{[B-1]} \oplus_2 1 \qquad \text{for odd parity} \qquad (5.7)$$

if the data path is modified by adding a piplining latch, carrying x_p, and an exclusive-OR gate to compute the following recursion:

$$P^{(0)} = x_p \qquad (5.8)$$

$$P^{(i+1)} = P^{(i)} \oplus_2 x^{[i]} \qquad (5.9)$$

It can be seen that after B iterations $P^{(B-1)}$ can be computed as one of the following:

$$P^{(B-1)} = x_p^e \oplus_2 x^{[0]} \oplus_2 x^{[1]} \dots \oplus_2 x^{[B-1]} \qquad (5.10)$$

$$P^{(B-1)} = x_p^o \oplus_2 x^{[0]} \oplus_2 x^{[1]} \dots \oplus_2 x^{[B-1]} \qquad (5.11)$$

replacing (5.6) and (5.7) into (5.10) and (5.11) results in

$$P^{(B-1)} = 0 \qquad \text{for even parity} \qquad (5.12)$$

$$P^{(B-1)} = 1 \qquad \text{for odd parity} \qquad (5.13)$$

in either case an error has occurred in this section of the cells if the expected value is not received at the output.

We note that the cell is capable of detecting errors even if the input, $X^{(i)}$, is zero. The reason for this is that the ROM still generates the parity bit outputs even if the steering switches by-pass the ROM for the $y^{(i)}$ data.

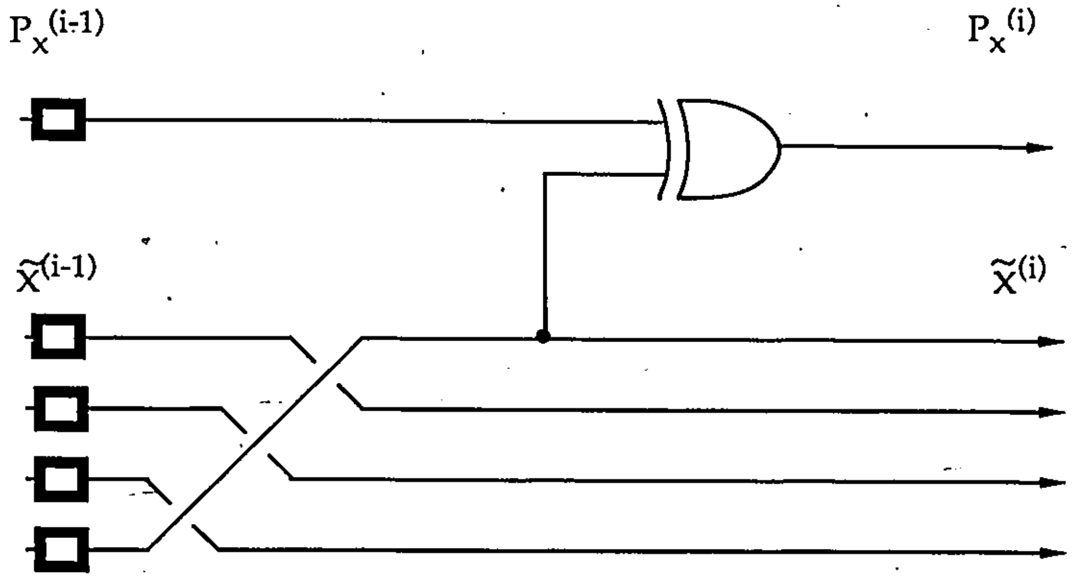


Fig 5.5 Parity fault detection circuitry for $X^{(i)}$ data

5.4. Application of $BIPSP_m$ in RRNS

In this section we present an example of a fault-tolerant system based on the fault detection architecture described above. A review of the 'classical' fault detection and correction techniques, using RNS representations, will be given first.

5.4.1. REVIEW OF RNS FAULT-TOLERANT TECHNIQUES

A standard residue number system is represented with a set of relatively prime moduli $\{m_1, m_2, 0..., m_L\}$. The dynamic range of the system is $M=\prod_{i=1}^{L} m_i$.

A redundant RNS (RRNS) is defined with a moduli set augmented by r redundant moduli

$$\{m_1, m_2, \ldots, m_L, m_{L+1}, \ldots, m_{L+r}\}$$

The legitimate range is $M=\prod_{i=1}^{L} m_i$ and the total range is $M_T= \prod_{i=1}^{L+r} m_i$.

Any number belonging to the interval $[M, M_T)$ is illegitimate[Etz'80]. Mandelbaum [Man'72] proved that a redundant residue number with r redundant moduli will be able to detect r errors and correct $\frac{r}{2}$ errors. Barasi and Masetrini[Bar'73] showed that an error in any of the residue digits in a RRNS maps the number into the illegitimate range, they also showed that, in an RRNS with r=2, if a single residue digit is in error only one of the possible combinations of L+1 out of L+2 moduli maps into the legitimate range and the remainder map into the illegitimate range. Etzel and Jenkins[Etz'80] used the digits from a mixed radix conversion of the number to check the viability of the projections. Later, Jenkins[Jen'83] designed an error checker (the device which implements both error detection and error location) with self-checking capability.

The main purpose of the error checker is not so much for error correction, but rather isolation of the faulty modulus[Jen'83]. Basically, the error checker projects the represented number into various subspaces in the RNS algebra using a mixed radix conversion (MRC) algorithm, and checks the (MRC) digits generated in order to detect the faulty channel. The main problem is that the hardware required for the checking and detection is cumbersome and as prone to errors as the system it is protecting! If we implement an RNS system based on the simple cell described in Section II, then error detection (over a subset of the error space) can also be implemented in a straightforward fashion.

It is important to show that this error detection scheme can be used in a correction scheme. The following theorem shows that a single error can be corrected with the information restricted to knowing the residue in error.

Theorem 2: A redundant residue system with one redundant modulus (r=1) allows correction of one error if the erroneous modulus is discarded.

Proof: Let a legitimate number, X, be represented as:

$$X=(x_1, x_2, \ldots, x_L, x_{L+1})$$

in an RRNS with a moduli set of $\{m_1, m_2 \ldots, m_L, m_{L+1}\}$*, where* m_{L+1} *is the redundant modulus,* $m_{n+1}>m_i$ *for* $i=1,2,\ldots L$ *and the dynamic range of the system is* $M=\prod_{i=1}^{L} m_i$ *. Assume there is an* $X'<M$ *which can be represented by the same set of residue digits, with the erroneous digit,*

x_j, removed. Then $X' = (x_1, x_2, \ldots x_{j-1}, x_{j+1}, \ldots, x_L, x_{L+1})$.

Since $M < \prod_{i=1\, i \neq j}^{L+1} m_i$, we also know that $X = (x_1, x_2, \ldots x_j, x_{j+1}, \ldots, x_L, x_{L+1})$.

Since the mapping is one-to-one and onto , X and X' must be identical.

This theorem basically allows us to use the information provided by the fault flag in each modulus and reduce the redundancy from r=2 to r=1.

5.4.2. Application Example

As a simple example, consider a system with 3 non-redundant moduli and one redundant modulus. The systolic cells, discussed in Section 5., are used to generate a processing array; e.g. a FIR filter with encoding from binary [Tah'87b], [Tah'87c]. The fault flag at the end of the linear systolic structure reflects the validity of the output; this flag is fed into a redundant RNS to binary decoder where each data output is examined in order to find three error free channels out of the four. This is accomplished by examining the fault flags accompanying each data line. By logically ORing the fault flags from each combination of 3 out of the 4 flags, the first OR output that is logical zero (i.e. all flags register correct data) represents the correct decoder output. This output is again the logical OR of the outputs from two parallel channels in the decoder (see the next discussion). Fig (5.6) shows the redundant decoder block diagram for this example.

It is important that the decoder be at least self checking, otherwise the fidelity of the entire system is in question. Ideally the generic fault detecting cell, discussed in this chapter, will be used to perform decoding, leading to a unified implementation procedure over the entire processing system. With this in mind we will examine the existing decoding methods.

Principally there are two methods for converting from RNS to WMS; the Chinese remainder theorem (CRT) and the mixed radix (MRC) technique. Both techniques have been examined extensively in the literature [Sod'83], [Sza'67], [Tay'84]. The CRT usually requires either a modulo M adder (where M is the dynamic range of the RNS being converted) or, in the case of a recent fractional conversion scheme [Sod'83], an L-input binary adder that is $\lceil \log_2 M \rceil + L$ bits wide. The extra L bits are required to correct for round-off errors in each of the fractional elements being summed. Both approaches require a summer with at least the full dynamic range of the number system, and are clearly not suitable candidates for our small dynamic range generic cell. The use of a large dynamic range connected element (in this case an adder), also introduces problems in error detection and correction which our approach, so far, has overcome. The MRC technique naturally retains computations over the individual rings of the RNS being converted. In its traditional form it requires a triangular array of interacting residue calculations (the height of the triangle is L), and the result is obtained in a mixed radix WMS. The radices are the same as the RNS moduli; this choice is crucial for the MRC to work successfully. Jenkins [Jen'83] presented a fault-tolerant procedure based on the MRC in which both physical and temporal redundancies are used. The technique projects all the possible L+1 out of L+2 residues in sequence and checks for a projection which falls in the legitimate interval. The encoder has a self checking design; however, finding the correct output may require up to L iterations.

The most natural vehicle for implementation, using our systolic cell, is found in a recently disclosed scheme [Sen'-]. This technique allows computation over the individual rings and also produces the output in bit-sliced binary. The basic concept is to repeatedly perform base extension to a modulus of the form 2t; in this way the binary output is computed in slices of t bits. This technique lends itself to implementation, as shown in Fig (5.6), for our example system of L=3, r=1.

Each decoder in Fig (5.6) is represented in detail in Fig (5.7); note that the structure of each decoder is identical. Each decoder consists of a linear systolic array of B cells. There is some redundancy between the blocks of the different decoders and it is possible to reduce the total number of blocks implemented.

The contents of each block have the form:

$$R = \{A \oplus_m B\} \otimes_m K \tag{5.14}$$

where A and B are variables and K is a constant multiplier. This is amenable to bit slicing and so each block in Fig (5.7) is capable of being sliced. For 5-bit moduli and a base extension of 32, the contents of the blocks for decoder 1 are given in the following:

$$B_1 = \{X_2 \oplus_{m2} [-X_3]\} \otimes_{m2} \{m_3^{-1}\} \tag{5.15a}$$

$$B_2 = \{X_1 \oplus_{m1} [-X_3]\} \otimes_{m1} \{m_3^{-1}\} \tag{5.15b}$$

$$B_3 = \{B_1 \otimes_{32} m_3\} \oplus_{32} X_3 \tag{5.15c}$$

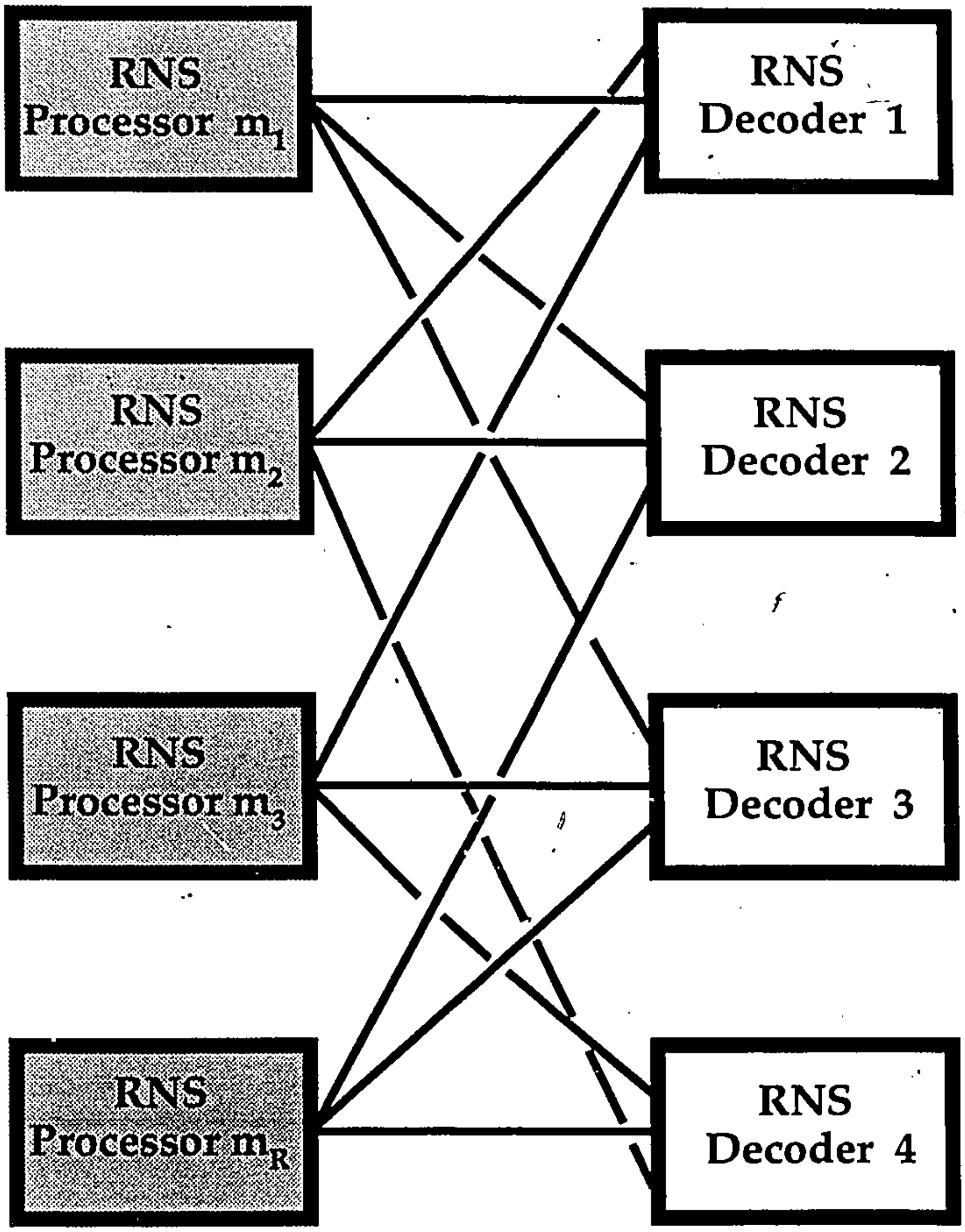


Fig 5.6 Redundant Decoder for L=3 r=1

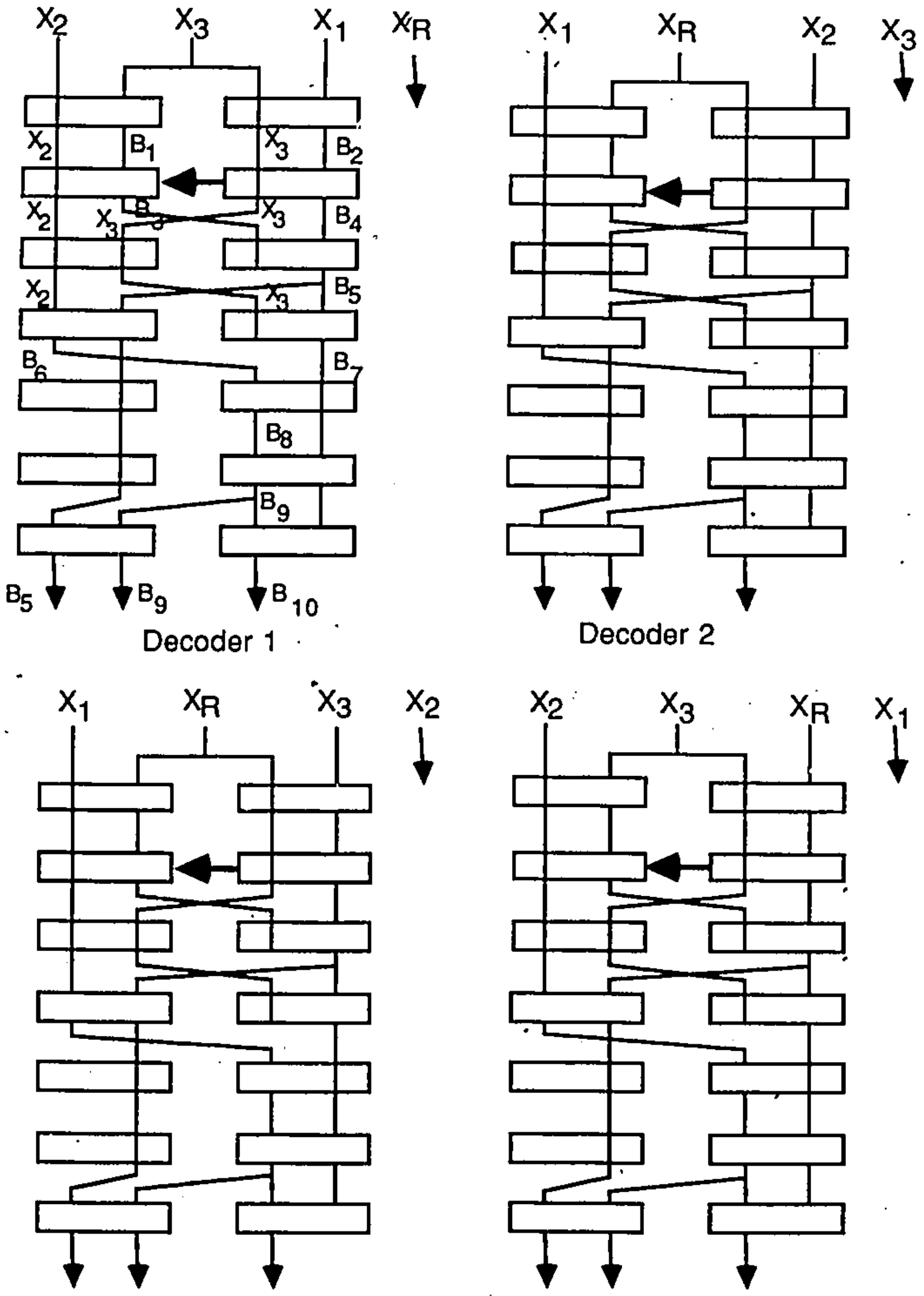


Fig 5.7 Example of Fault-tolerant decoder for L=3, r=1

$$B_4 = \{B_2 \oplus_{m1} [-B_1]\} \otimes_{m1} \{m_2^{-1}\} \quad (5.15d)$$

$$B_5 = \{B_3 \oplus_{32} \{B_4 \otimes_{32} m_2 \otimes_{32} m_3\} \quad (5.15e)$$

$$B_6 = \{X_2 \oplus_{m2} [-B_5]\} \otimes_{m2} \{32^{-1}\} \quad (5.15f)$$

$$B_7 = \{X_3 \oplus_{m3} [-B_5]\} \otimes_{m3} \{32^{-1}\} \quad (5.15g)$$

$$B_8 = \{B_6 \oplus_{m2} [-B_7]\} \otimes_{m2} \{m_3^{-1}\} \quad (5.15h)$$

$$B_9 = \{B_8 \otimes_{32} m_3\} \oplus_{32} B_7 \quad (5.15i)$$

$$B_{10} = \{B_7 \oplus_{m3} [-B_9]\} \otimes_{m3} \{32^{-1}\} \quad (5.15j)$$

The output is obtained as the concatenation of B_{10}, B_9, B_5 (15 bits) in descending order of significance. For this example L+r=4 of these decoders, each operating independently, are required. A single fault occurring in any one of the moduli channels, or in the decoder, is flagged as an error at the end of the decoding operation. The correct output (in the case of no error, all of the outputs are the same and correct) is indicated as such by its flag.

The redundancy required by the system is:

- ➢ 2 extra bits per word in each ROM of the cell
- ➢ 2 extra steering switches per cell
- ➢ 2 two-input gates per cell
- ➢ 1 extra processing channel
- ➢ L extra decoders

In some case it is required to monitor the different sections of the system for later improvement. The scheme described above only supplies this information at the end of the operation; however, by including simple binary counters, clocked by the Fault Out signals at different locations, the entire pipeline array can be monitored. A permanent or soft error which affects one or several stages of one module can be corrected since each stage performs operations on different samples at the same time. This fact allows several stages of the array to become faulty for a period of time and still enables correct results to be recovered. The only restriction is that the computation on any sample can fail, at most, in one modulus for the same set of clock periods.

5.5.Design For Testability (DFT)

As discussed in 5.2.1, imperfect manufacturing processes produce faulty chips. The manufacturer normally performs different tests on the products in order to remove the faulty product from production lines.

It is known that the production cost of a chip can be broken down into:

A) One time expenses such as layout and test pattern generation costs.

B) Many time expenses such as test application or socket time costs.

In some cases a major percentage of the total cost is the result of part B above.

The main issues in testability are *Controllability* and *Observability* as defined below [McC'86], [Abr'86]:

> *Controllability* refers to the ease of producing specific internal signal values by applying signals to the circuit input leads.
>
> *Observability* refers to the ease with which the state of an internal signal can be determined at the circuit output leads.

In the following we briefly review some of the techniques which improves the above two factors.

Ad hoc techniques:

In this approach a set of guidelines are specified before hand. The set normally consists of a list of design features that create testing problems together with suggestions of preferred implementations. For example it is advisable to provide circuitry to break the feedback loops during the test. A major difficulty with this approach is the requirement of adding extra control inputs or observation outputs.

Scan path techniques [Wil'73]:

In this technique the chip is placed into test mode first in which all circuit flip-flops are interconnected as a shift register. An arbitrary test pattern is clocked into this shift register and then the chip is placed back to its normal mode from the test mode. At this time the combinatorial circuitry acts on the flip-flop contents and primary input signals, the results are then stored in the in the flip flops. Now, by returning to the test mode again the contents of the shift register can be clocked out and compared to the expected sequence.

Testing circuits which include ROM, using scan path techniques, is not straightforward. McCluskey [McC'86, pp.121] specifies three problems with this approach:

1) *Tests for some of the faults in the combinational logic may require that the memory outputs be set to specified values.*

2) *Tests for some of the faults in the combinational logic may require that the effect of the fault be propagated through the memory in order for the effect appear at a latch or primary output.*

3) *To test a ROM fully it is necessary to read each cell.*

In the following the testing techniques for the arrays based on BIPSP_m will be explained.

5.5.1. Testing the Arrays Based on BIPSP_m

A suitable testing procedure for the arrays consists of a few sequential steps which are performed on different sub-sections of all the cells in the array. Let us first partition the BIPSP_m cell into sub-sections;

i) The ROM, its steering switches and the latches.

ii) The by-passing route around the ROM and the related steering switches.

iii) The data path with the associated latches.

As was discussed earlier in this chapter, a comprehensive test on ROMs requires reading of all the locations. In the following we are going to show that there exists a scheme that allows all the locations in the ROMs to be tested without the need for any extra hardware or input/output pins. Let us introduce a new symbol for a ROM as shown in Fig (5.8). The numbers on the left are addresses and the straight line represents the mapping function of the ROM. The ROMs used in the designs, presented in previous chapters, store the generic function given in eqn (3. 6a) and repeated here

$$\text{output} = \text{input} \oplus_m [A \otimes_m 2^i] \qquad (3.6a)$$

Since A, and i are constant the above equation can be rewritten as:

$$\text{output} = \text{input} \oplus_m \alpha \qquad (5.16)$$

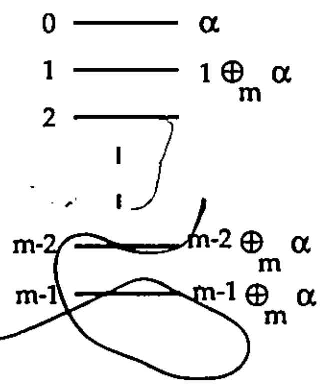


Fig 5.8 A representation of a ROM with its function

Now let us assume N of these ROMs are cascaded in a linear array with the output of each ROM used as an input to the following cell, shown in Fig (5.9). The path initiated with an input is also shown.

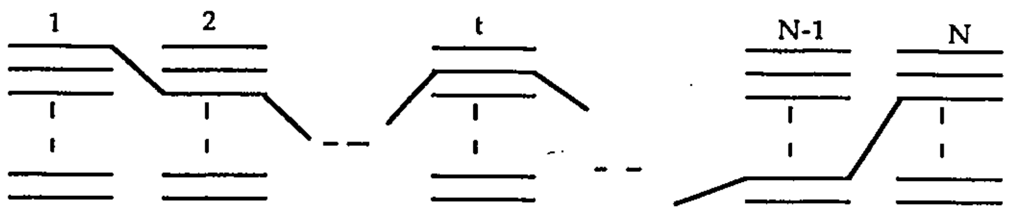


Fig 5.9 Test vector path in the array

Theorem 5.3:

Assume the set {0, 1, 2, ..., m-1} is used, sequentially, as the input to the array shown in Fig (5.9). The path for two different inputs will have no common segment.

Proof:

First let us assume that the two different input sequences x and y refer to location k of cell t in the array. In other words, there is a common segment or path for these input sequences in cell k.

It is clear that x is mapped to $x \oplus_m \alpha_1$ by the first cell, the second cell produces $x \oplus_m \alpha_1 \oplus_m \alpha_2$, this continues in the first t-1 cells producing an output equivalent to k. This can be written as:

$$x \oplus_m \alpha_1 \oplus_m \alpha_2 \oplus_m \alpha_3 \dots \oplus_m \alpha_t = k \qquad (5.18)$$

For a different input $y \neq x$ to refer to the same location in cell "t" we must have:

$$y \oplus_m \alpha_1 \oplus_m \alpha_2 \oplus_m \alpha_3 \dots \oplus_m \alpha_t = k \qquad (5.19)$$

Clearly both (5.18) and (5.19) can not be true, hence x, and y can not refer to the same location in the cell t. Thus there is no common segment in the two paths.

Since the number of elements in the input set is equivalent to the number of ROM locations and from the above theorem no location can be referred to more than once, it follows that every location is read exactly once.

This analysis can be used to produce a simple test for the arrays consisting of $BIPSP_m$ cells.

1) Activate the ROMs in the array by applying a binary equivalent of 2^B-1 or "11...11" to the data path.

2) Use all the elements of $\{0, 1, 2, ..., m-1\}$ as the input to the array sequentially. It is obvious that only m clocks are required to read in all the inputs.

During the ROM test, the by-pass circuits are not active, hence they have to be tested separately. This can be performed by applying "00 ... 00" to the data path in order to by-pass the ROMs. Each line and its associated steering switch can be tested by using these two sequences $\{0, 0, ..., 0\}$ and $\{1, 1, ..., 1\}$ or an exhaustive test by using all the possible ring elements, requiring m clock cycles. The data path itself can be tested exhaustively, using all the possible inputs..

The procedure discussed above can be used on array of any length and the total number of clock cycles required is given by:

$$T_{tot}=t_{ROM} + t_{by\text{-}pass} + t_{data\ path} + N \qquad (5.20)$$

where T_{tot} is the total number of clock cycles needed to test the entire array, t_{ROM}, $t_{by\text{-}pass}$, and $t_{data\ path}$ are the number of clock cycles needed to test the ROMs, the by-pass circuits, and the data path. An extra N clock cycles is added this is to allow for the last test input to shift out of the array.

As an example let us compare testing of a mod m adder implemented both using a single ROM and $BIPSP_m$ cells. A complete test of a ROM with m^2 locations requires m^2 clock cycles, while testing a $BIPSP_m$ based mod m adder only requires 3m + B clocks.

5.6. SUMMARY

This chapter has shown how a simple parity check fault detection circuit can be constructed around the cell, and has demonstrated that it is capable of detecting all single faults within the cell. In a linear systolic array, the cells communicate the fault information through a pipelined 'daisy chain' structure, which allows the fault flag to accompany the data through the array. The chapter has also described techniques for producing fault-tolerant systems by the use of redundant decoding techniques. The redundancy required by the system is detailed and a technique for monitoring of long term system faults has also been briefly described.

Finally the by-pass circuit in the cell enabled us to devise a test procedure to test the cells in the entire array very efficiently.

CHAPTER 6

CONCLUSIONS

The primary aim and objective of this dissertation has been the investigation and development of suitable structures for implementing finite rings and fields operations in VLSI, exploring the advantages of using such structures and comparing them with existing techniques.

The contributions of this work lie in the following areas:

1. Development of structures of generic cells for performing finite ring/field operations suitable for implementation in VLSI.

2. Use of these fundamental structures in Digital Signal Processing applications.

3. Presenting a novel fault detection technique using the generic cell.

4. Modification of the conventional RRNS to incorporate the new fault detection technique.

5. Simplification of the test procedure for arrays consisting of these fundamental structures.

Taking the requirements of "good" VLSI design as a guide line, a new structure consisting of a small ROM, steering switches, and pipeline latches has been presented. It has been demonstrated that utilizing a bit slicing approach, finite ring/field operations can be performed, using these structures, in linear arrays. Among the advantages of this approach are:

A. Area saving:

In general performing a mod m addition requires $m^2 \lceil \log_2 m \rceil$ bits of memory, using the single ROM approach, compared to $m \lceil \log_2 m \rceil^2$ bits when the technique described in chapter 3 is utilized. A 5 bit modulus was taken as an example to show the saving in practical applications. Table (3.3) provides the comparison results using different models.

B. Higher throughput:

Use of several small ROMs, which are naturally faster than a large single ROM, also provide more piplining opportunity in the circuit. The throughput rate of the $BIPSP_m$ based mod m adders is shown to be at least 4 times better than a single ROM design.

Chapter 4 has presented modified versions of the $BIPSP_m$ cell tailored for digital signal processing algorithms. These algorithms normally require intensive multiply/add operations. The ability of $BIPSP_m$ to perform fixed coefficient multiplication was exploited to develop efficient FIR filter structures. Our proposed structure was compared to the existing binary based filters and found to have area advantage.

Structures for encoding binary to RNS and decoding from RNS to binary based on the $BIPSP_m$ cell have been presented. This allows the entire system from encoding to processing to decoding to be implemented using the same generic cell with the same clock rate.

It was shown that the operations of scaling and decoding can be combined together for the cases that the output can be represented in a smaller dynamic range.

It was also shown how a simple parity check fault detection circuit can be used around the cell to detect for single errors in any part of the cell. An output fault flag is 'daisy chained' throughout the systolic array, and is pipelined along with the data. A fault is detected by the flag having a 'true' value, and this signal moves in synchronism with the faulty data. By using a redundant modulus, a reduced RNS fault tolerant procedure can be invoked to correct the output data. The important fact is that the fault tolerant decoder also uses the same cell as the processor, and this allows easy checking of the validity of the output from this decoder. Also a simple scheme has been briefly discussed for the logging of continuous errors (indicating hard faults or long term soft errors) in any of the channels of the processor.

Finally the advantages of the approach in the area of fabrication testing have been discussed. The fact that the ROM and the steering switches in the cells can be independently controlled was used to devise a simplified test procedure. It was shown that performing an exhaustive test for the first cell will generate suitable test inputs for the subsequent cells in the array. The steering switches and the data paths can be tested very efficiently using exhaustive test techniques.

APPENDIX A

Implementation of BIPSP_m and a 5 bit mod m adder

The lay out and a brief description of the BIPSP_m cell and a 5 bit mod m adder based on this cell is provided in this appendix.

Fig (A.1) shows the lay out for the BIPSP_m cell. The cell is designed using a 3 micron double metal CMOS technology and occupies 159000 μ^2 (421.2 μ by 377.4 μ). A total of 293 transistor is used in the cell, 160 of them form the ROM memory plane and are shown in the lower left corner of the plot. The remaining 54% of the transistors in the cell form the ROM decoders, latches, and transmission gates and occupy almost 3/4 of the total area. The maximum clock rate based on SPICE simulation for the cell is 40 MHz, the cell consumes 3.2 mWatts.

Fig (A.2) gives the layout of a 5 bit mod m adder based on the above cell. The layout also contain an separate cell which is included for test purposes.

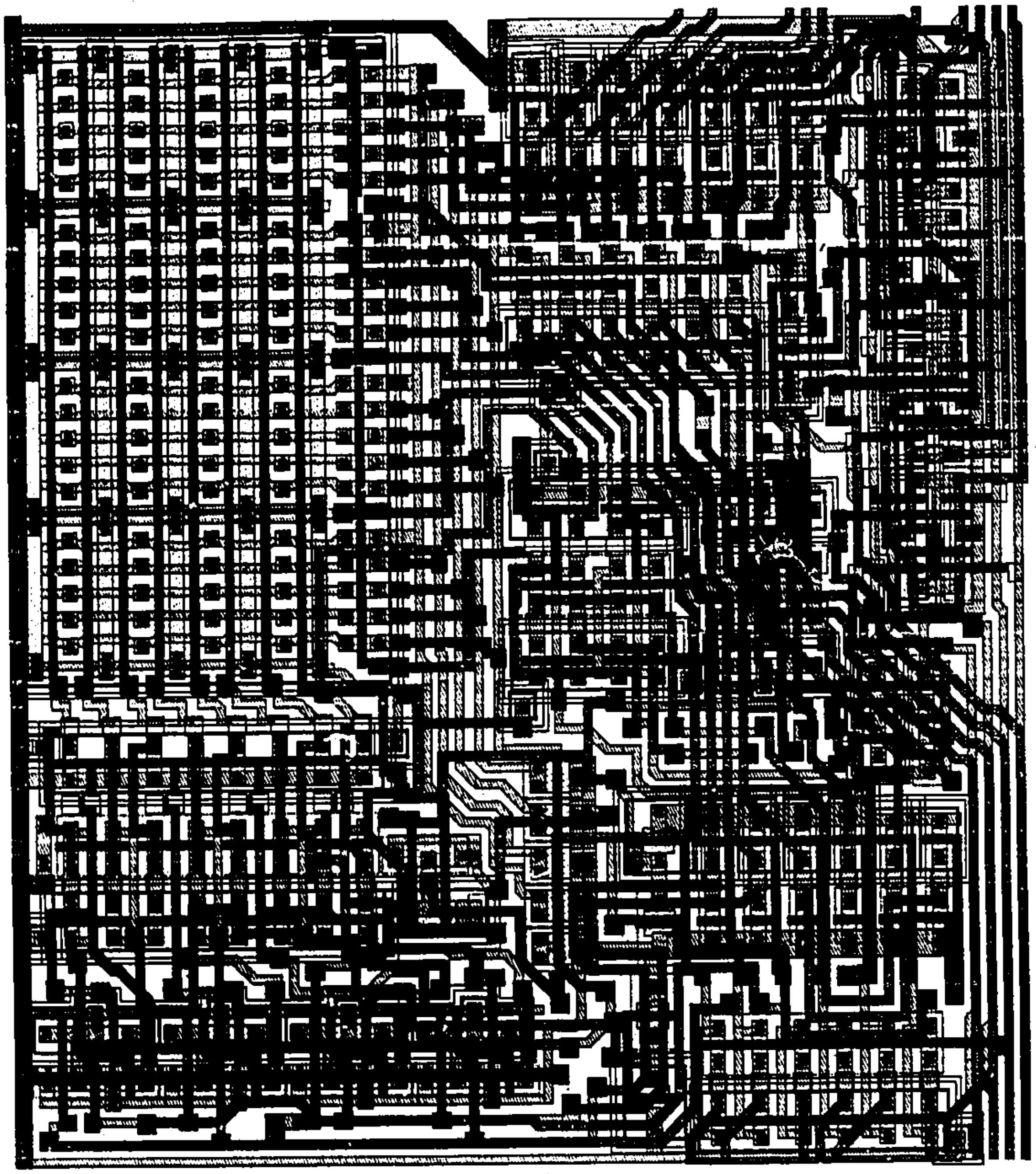


Fig (A.1) Layout for the BIPSP$_m$ cell

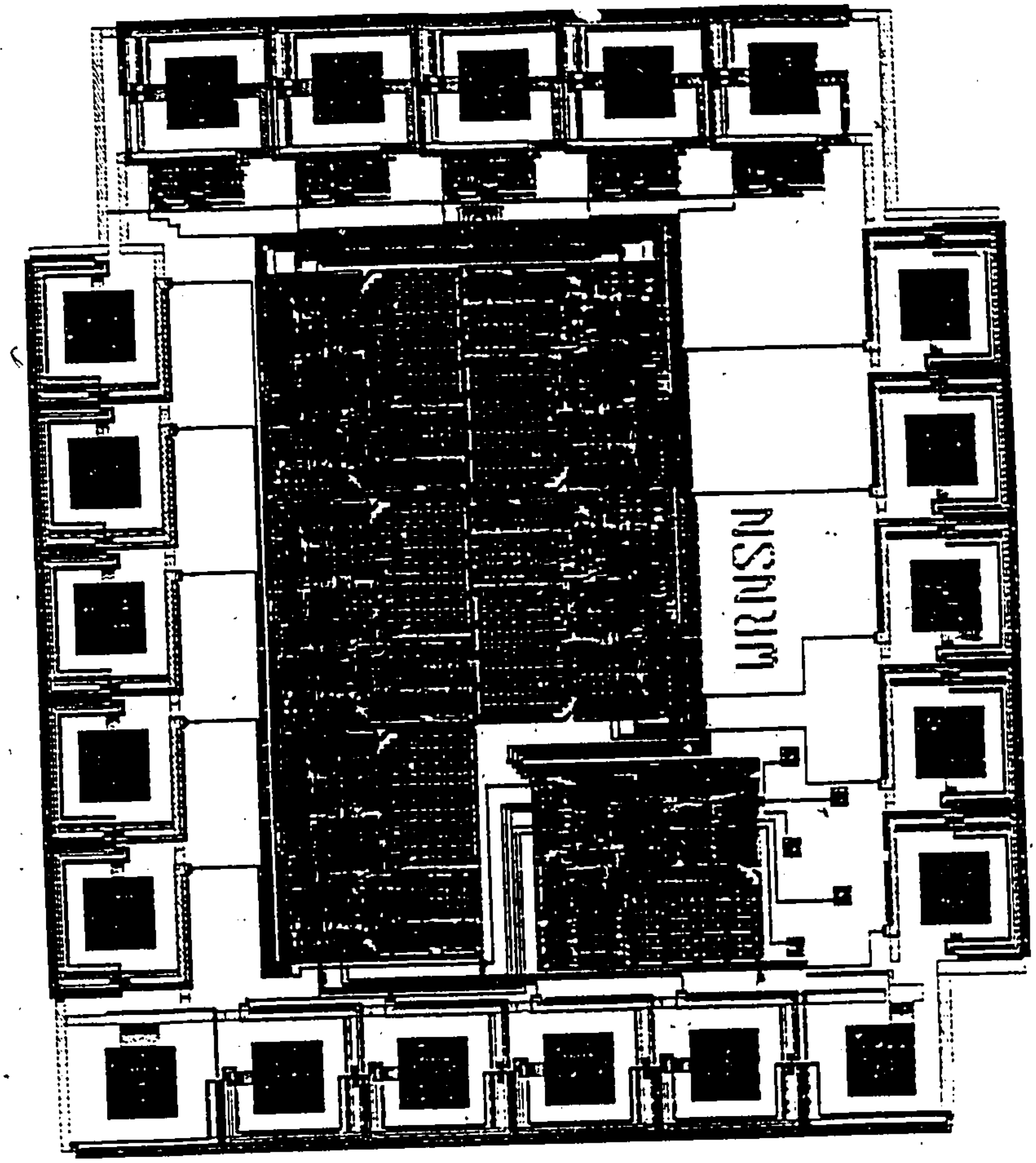


Fig (A.2) Layout for a mod m adder

APPENDIX B

This program simulates the hardware given in section (4.5) for reverse mapping of RNS into binary combined with scaling. The signal names in Fig (4.16) correspond to the variables in the program.

```
INTEGER X,X1,X2,X3,X4,XP,XP1,XP2

m1=29
m2=27
m3=31
m4=32

GAMMA=(27*31-1)/2

DO X=0,M1*M2*M3*M4-1
        X1=MOD(X+GAMMA,M1)
        X2=MOD(X+GAMMA,M2)
        X3=MOD(X+GAMMA,M3)
        X4=MOD(X+GAMMA,M4)
        XP=INT((X+GAMMA)/(27*31))
        XP4=MOD(XP,M4)
        XP1=MOD(XP,M1)
        XP3=MOD(XP,M3)
        B1=MOD(X3-X2+10*M3,M3)
        B1=MOD(B1*INV(M2,M3),M3)

        B2=MOD(X1-X2+10*M1,M1)
        B2= MOD(B2*INV(M2,M1),M1)

        B3=MOD(B2-B1+10*M1,M1)
        B3=MOD(B3*INV(M3,M1),M1)

        B4=MOD(X4-X2+10*M4,M4)
        B4=MOD(B4*INV(M2,M4),M4)

        B5=MOD(B4-B1+10*M4,M4)
        B5=MOD(B5*INV(M3,M4),M4)



        B6=mod(B3-B5+10*M1,M1)
        B6=MOD(B6*INV(M4,M1),M1)

        B11=B5+32*B6
            IF(ABS(B11-XP).EQ.0) THEN
    ELSE
            PRINT*,'ERROR AT X=',X,X/GAMMA,B5,B6,B11
    GO TO 13
    END IF
    END DO
    PRINT*, 'X=',X,'    NO ERROR WAS FOUND'
13  STOP
    END

    INTEGER FUNCTION INV(K1,K2)
    I=1
    do while (mod(K1*I,K2).ne.1)
    I=I+1
    end do
    INV=i
    END
```

This program simulates the hardware given in section (4.5) for improved reverse mapping of RNS into binary combined with scaling. The signal names in Fig (4.16) correspond to the variables in the program.

```
        INTEGER X,X1,X2,X3,X4,XP,XP1,XP2


        m1=29
        m2=27
        m3=31
        m4=32

*************COMPUTING  THE  INVERSES******************

        inv_m2_m1=inv(m2,m1)
        inv_m3_m1=inv(m3,m1)
        inv_m4_m1=inv(m4,m1)
        inv_m2_m3=inv(m2,m3)
        inv_m3_m4=inv(m3,m4)
        inv_m2_m4=inv(m2,m4)

*************COMPUTING  THE  CONSTANT  COEFF******************

        X1_COEF=INV_M2_M1*INV_M3_M1*INV_M4_M1
        X4_COEF=INV_M2_M4*INV_M3_M4
        B3_COEF=INV_M3_M1*INV_M4_M1

****************scaling   factor********************

        GAMMA=(27*31-1)/2
```

```
**********computing the inputs pre multiplied with the coeff*********

          DO X=0,M1*M2*M3*M4-GAMMA-1
          X1=MOD(MOD(mod(X,M1)+MOD(GAMMA,M1),M1)*X1_COEF,m1)
          X2=MOD(MOD(X,M2)+MOD(GAMMA,M2),M2)
          X3=MOD(mod(MOD(X,M3)+MOD(GAMMA,M3),M3)*inv_m2_m3,m3)
          X4=MOD(mod(MOD(X,M4)+MOD(GAMMA,M4),M4)*X4_COEF,m4)

**********computing the results  for comparison*******************

          XP=INT((X+GAMMa)/(27*31))
          B1=MOD(X3+mod(-X2+2*M3,M3)*inv_m2_m3,m3)
          B2=MOD(X1+mod(-X2+2*M1,M1)*X1_COEF,m1)
          B3=MOD(B2+mod(-B1+2*M1,M1)*B3_COEF,m1)
          B4=MOD(X4+mod(-X2+2*M4,M4)*X4_COEF,m4)
          B5=MOD(B4+mod(-B1+2*M4,M4)*inv_m3_m4,m4)
          B6=mod(B3+mod(-B5+2*M1,M1)*inv_m4_m1,m1)
          B11=B5+32*B6


          IF(ABS(B11-XP).EQ.0) THEN
          IF(MOD(X,10000).EQ.0) PRINT*, X
          ELSE
          PRINT*,'ERROR AT X=',X,X/GAMMA,B5,B6,B11
          GO TO 13
          END IF
          END DO

          PRINT*, 'X=',X,'    NO ERROR WAS FOUND'
13        STOP
          END
```

****** FUNCTION TO CALCULATE THE INVERSES******

```
      INTEGER FUNCTION INV(K1,K2)
      I=1
      do while (mod(K1*I,K2).ne.1)
      I=I+1
      end do
      INV=i
      END
```

This program simulates the hardware given in section (5.4.2) for reverse mapping of RNS into binary . The signal names in Fig (5.7) correspond to the variables in the program.

```
INTEGER X,X1,X2,X3,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10
write(*,*) 'enter M1 and M2 AND M3'
read(*,*)M1,M2,M3

DO X=0,M1*M2*M3-1
X1=MOD(X,M1)
X2=MOD(X,M2)
X3=MOD(X,M3)


B1=MOD(X2-X3+M2,M2)
B1=MOD(B1*INV(M3,M2),M2)

B2=MOD(X1-X3+M1,M1)
B2= MOD(B2*INV(M3,M1),M1)

B3=MOD(B1*M3,32)
B3=MOD(B3+X3,32)

B4=MOD(B2-B1+M1,M1)
B4=MOD(B4*INV(M2,M1),M1)

B5=MOD(B4*M2*M3,32)
B5=MOD(B3+B5,32)

B6=mod(X2-B5+10*M2,M2)
B6=MOD(B6*INV(32,M2),M2)

B7=MOD(X3-B5+10*M3,M3)
B7=MOD(B7*INV(32,M3),M3)
```

```
      B8=MOD(B6-B7+10*M2,M2)
      B8=MOD(B8*INV(M3,M2),M2)

      B9=MOD(B8*M3,32)
      B9=MOD(B9+B7,32)

      B10=MOD(B7-B9+10*M3,M3)
      B10=MOD(B10*INV(32,M3),M3)

      B11=B5+32*B9+1024*B10
      IF(B11.EQ.X) THEN
      PRINT*, B11,X
      ELSE
      PRINT*,'ERROR AT X=',X,B5,B6,B7,B8,B9,B10,B11
      GO TO 13
      END IF
      END DO
13    STOP
      END

      INTEGER FUNCTION INV(K1,K2)
      I=1
      do while (mod(K1*I,K2).ne.1)
      I=I+1
      end do
      INV=i

      END
```

REFERENCES

[Abr'86] J.A. Abraham and W.K. Fuchs, "Fault and Error Models for VLSI," (invited paper), Proceedings of IEEE, Vol. 74, No. 5, May 1986, pp. 639-654.

[Aga'75] R.C. Agarwal and C.S. Burrus,"Number Theoretic Transforms to Implement Fast Digital Convolution," Proc. IEEE, Vol-63, April 1975, pp: 550-560.

[Avi'84] Algirdas Avizienis and John P.J. Kelly, "Fault Tolerance by Design Diversity: Concepts and Experiments," IEEE Computer Magazine, Vol.17, No. 8, August 1984, pp. 67-80.

[Bal'83] R.E. Blahut, "Theory and Practice of Error Control Codes," Addison-Wesly, Reading MA., 1983.

[Ban'86] Subir Bandyopadhyay, G.A. Jullien, and M. Bayoumi, "Systolic Arrays Over Finite Rings with Applications to Digital Signal Processing," in Systolic Arrays, a collection of papers presented at the First International Workshop on Systolic Arrays, Oxford, edited by W. Moore, A.McCabe, and R. Urquhart, Adam Hilger, Bristol, 1986.

[Bar'73] Ferruccio Barsi and Piero Maestrini, "Error Correcting Properties of Redundant Residue Number Systems," IEEE Trans. on Computers, Vol. C-22, No. 3, March 1973, pp. 307-315.

[Bar'74] Ferruccio Barsi and Piero Maestrini, "Error Detection and Correction by Product Codes in Residue Number Systems," IEEE Trans. on Computers, Vol. C-23, No. 9, September 1974, pp. 915-924.

[Bay'85] M.A. Bayoumi," VLSI Implementation of Residue Number Systems Architectures," Ph.D. Dissertation, Department of Electrical Engineering, University of Windsor, 1985.

[Bay'87] M.A. Bayoumi, G.A. Jullien, W.C. Miller, "A Look-up Table VLSI Design Methodology for RNS Structures Used in DSP Applications," IEEE Trans. on Circuits and Systems, July 1987.

[Big'85] Norman L. Biggs " Discrete Mathematics," Clarendon Press. Oxford. 1985.

[Bir'77] Garrett Birkhoff and Saunders MacLane," A Survey of Modern Algebra," 4th edition, MacMillan Publishing Co. New York.

[Bir'87] Peter D. Bird, "The Application of Multi-Valued Logic to the Implementation of Residue Number System Hardware," M.A. Sc. Thesis, Department of Electrical Engineering, University of Windsor, 1987.

[Blo'87] Norman J. Bloch," Abstract Algebra with applications," Prentice-Hall, Inc. 1987.

[Cap'81] P.R. Capello and K. Steiglitz, "Digital Signal Processing Applications of Systolic Algorithms," in VLSI Systems and Computations, H.T. Kung, R. Sproull and G. Steele eds., Computer Science Press, 1981, pp. 245-254.

[Cap'82] P.R. Capello, "VLSI Architectures for Digital Signal Processing," Ph.D. Dissertation, Electrical Engineering and Computer Science, Princeton University, October 1982.

[Che'85] C. Y. Chen, W. K. Fuchs, and J.A. Abraham, "Error Detection in PLAs and ROMs," Proc. of IEEE Inter. Conf. on Computer Design, ICCD'85, October 1985, pp. 525-529.

[Chr'86] Bob Christie, " The MA7170 Systolic Correlator, Architecture and Applications," in Systolic Arrays, a collection of papers presented at the First International Workshop on Systolic Arrays, Oxford, edited by W. Moore, A.McCabe, and R. Urquhart, Adam Hilger, Bristol, 1986.

[Cor'83] A.G. Corry, and K. Patel,"Architecture of a CMOS Correlator," Proc. IEEE Symp. on Circuits and Systems, 1983,pp. 522-525.

[Dan'84] Pre E. Danielsson, "Serial/Parallel Convolvers," IEEE Trans. on Computers, Vol. C-33, No. 7, Jully 1984, pp. 652-667.

[Den'83] D.E.R. Denning, "Cryptography and Data Security," Addison-Wesly, Reading MA., 1983.

[Etz'80] Mark H. Etzel and W.K. Jenkins, "Redundant Residue Number Systems for Error Detection and Correction in Digital Filters," IEEE Trans. on Acoustic, Speech, and Signal Processing, Vol. ASSP-28, No. 5, October 1980, pp. 538-545.

[Etz'82] Mark H. Etzel and W.K. Jenkins, "The Design of specialized residue classes for efficient recursive digital filter realization," IEEE Trans. on Acoustic, Speech, and Signal Processing, Vol. ASSP-30, No. 3, June 1982, pp. 370-380.

[Far'82] John B Fraleigh, "A First Course in Abstract Algebra" Third edition, Addison Wesley, Reading, MA. 1982.

[Fis'85] A.L. Fisher, and H.T. Kung, " Synchronizing Large VLSI Processor Arrays," IEEE Trans. on Computers, Vol. C-34, No. 8, August 1985, pp. 734-740.

[For'85] J.A.B. Fortes, K.S. Fu, and B.W. Wah , " Systematic Approaches to the Design of Algorithmically Specified Systolic Arrays," Proc. of IEEE Intr. Conf. on Acoustics, Speech, and Signal Processing, ICASSP'85, April 1987,pp. 8.9.1-8.9.5.

[For'87] J.A.B. Fortes, and B.J. Wah, "Systolic Arrays From Concept to Imlementation," Computer, Vol. 20, No. 7, July 1987, pp. 12-17.

[Fos'80] Foster M.J. , Kung H.T., "The Design of special-Purpose VLSI Chips". IEEE Computer Magazine, January 1980, pp.26-40.

[Fou'82] S.D. Fouse, G.R. Nudd and A. D. Cumming, " A VLSI Architecture For Pattern Recognition Using Residue Arithmetic," Proceedings of 6th International Conference on Pattern Recognition, Munich, Germany, Oct. 1982.

[Fou'87] D.E. Foulser and R.Schrriber, "The Saxpy Matrix-1: A General-Purpose Systolic Computer," Computer, Vol. 20, No. 7, July 1987, pp. 35-43.

[Fuc'84] W.K. Fuchs and J.A. Abraham, " A Unified Approach to Concurrent Error Detection in Highly Structured Logic Arrays," Proc. 14th Int. Symp. on Fault-Tolerant Computing, June 1984, pp. 4-9

[Gal'86] Joseph A. Gallian, "Contemporary Abstract Algebra," D.C. Heath and Company, Lexington MA. 1986.

[Gol'73] Larry J. Goldstein, " Abstract Algebra," Prentice-Hall, N.J. 1973.

[Gil'76] William J. Gilbert " Modern Algebra with Applications," John Wiley & Sons. N.Y. 1976.

[Hat'86] M. Hatamian and G.L. Cash, " High Speed Signal Processing, Pipelining, and VLSI," Proc. ICASSP, April 1986, pp. 1173-1176.

[Hua'81] C.H. Huang, D.G. Peterson, H.E. Rauch, J.W. Teague, and D. F. Fraser, " Implementation of Fast Digital Processor Using the Residue Number System," in Residue Number System Arithmetic: Modern Applications in Digital Signal Processing, Edited by M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor. IEEE Press, New York, 1986.

[Hua'84] Kuang-Hua Huang and Jacob A. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations," IEEE Trans. on Computers, Vol. C-33, No. 6, June 1984, pp. 518-528.

[Haw'84] Kai Hwang and Fay A. Briggs, " ComputerArchitecture and Parallel Processing," McGraw-Hill, New York, 1984.

[Jac'51] Nathan Jacobson " Lectures in Abstract Algebra, Basic Concepts" Springer-Verlag, New York, 1951.

[Jen'77] W.K. Jenkins and B.J. Leon, " The Use of Residue Number Systems in the Design of Finite Impulse Response Digital Filters," IEEE Trans. Circuits and Systems, Vol..CAS-24, April 1977, pp. 191-201.

[Jen'80] W.K. Jenkins, C.F. Lee,"Complex Residue Number System for Digital Signal Processing," Proc. 14th Asilomar Conf. on

Circuit, Systems and Computers, Pacific Grove, C.A. Nov. 1980.

[Jen'82a] W.K. Jenkins , "Residue number system error checking using expanded projection," Electron. Lett., Vol. 18, No. 21, pp. 927-928, Oct.1982.

[Jen'82b] W.K. Jenkins,"A standard computational element for the VLSI realization of digital processors using modular arithmetic," Proceedings of the 1982 Asilomar Conference on Circuits Systems and Computers, Pasific Grove, CA. November 1982, pp. 202-206.

[Jen'83] W. Kenneth Jenkins, " The Design of Error Checkers for Self-Checking Residue Number Arithmetic," IEEE Trans. on Computers, Vol. C-32, No. 4, April 1983, pp. 388-396.

[Jen'87] W.K. Jenkins and J.V. Krogmeier, "The Design of Dual-Mode Complex Signal Processors Based on Quadratic Modular Number Codes," IEEE. Trans. Circuits and Systems, (invited paper), Vol. CAS-34, No. 4, April 1987, pp. 354-364

[Jha'84] Niraj K. Jha and Jacob A. Abraham, " Totally Self-Checking MOS Circuits Under Realistic Physical Failures," Proc. of IEEE Inter. Conf. on Computer Design, ICCD'84, October 1984, pp. 665-670.

[Jul'78] G.A. Jullien, "Residue Number Scaling and Other Operations Using ROM Arrays," IEEE Trans on Computers; Vol. C-27, No.4, April 1978, pp. 325-336.

[Jul'87a] G.A. Jullien, R.Krishnan and W.C. Miller,"Complex Digital Signal Processing Over Finite Rings," IEEE. Trans. Circuits and Systems, (invited paper), Vol. CAS-34, No. 4, April 1987, pp. 365-377.

[Jul'87b] G.A. Jullien, M. Taheri and W.C. Miller, "Fault-Tolerant Techniques for a Class of Residue Arithmetic Processors," Proceedings of the 1987 Asilomar Conference on Circuits Systems and Computers, Pasific Grove, CA. November 1987.

[Jul'88] G.A. Jullien, M. Taheri J. Carr, G Thomsen, and W.C. Miller, "A VLSI Systolic Quadratic Residue DFT with Fault-

Tolerance," Proceedings of the 1988 IEEE International Symposium on Circuit and Systems, Finland, Helsinki, June 1988.

[Kes'87] Keshab K. Parhi, and David G. Messerschmitt," Concurrent Cellular VLSI Adaptive Filter Architectures, "IEEE Transaction on Circuits and Systems, Vol. CAS-34, No. 10, October 1987, pp. 1141-1151.

[Kor'86] Israel Koren, and Dhiraj K. Pradhan, "Yield and Performance Enhancement Through Redundancy in VLSI and WSI Multiprocessor ," Proceedings of IEEE, Vol. 74, No. 5, May 1986, pp. 699-711.

[Kro'83] J.V. Krogmeier and W.K. Jenkins,"Error Detection and Correction in Quadratic Residue Number Systems," Proceedings of 1983 Mid-West Symposium on Circuits and Systems, Puebla, Mexico, August 1983.

[Kuh'81] R. H. Kuhn and D. A Padua, Eds., IEEE Computer Society Tutorial on Parallel Processing, 1981.

[Kun'78] H.T. Kung, C.E. Leiserson,"Systolic Arrays (for VLSI)," Sparse Matrix Proc. 1978,1979, Academic Press, Orlando, Fla., pp.256-282.

[Kun'82] H.T. Kung, " Why Systolic Architectures," IEEE Computer Magazine, Vol. 15, No.1, Jan. 1982, pp. 37-46

[Kun'83] S.Y. Kung and J. Annevelink, "VLSI Design for Massively Parallel Signal Processors," Microprocessors and Microsystems, Vol.7, No.10, December 1983, pp. 461-467.

[Lei'81] C.E. Leiserson,"Area Efficient VLSI Computation," Ph.D. Dissertation, Dept. Computer Science, Carnegie-Mellon University, Oct. 1981.

[Lin'70] Shin Lin] "An Introduction to Error-Correcting Codes,"Prentice-Hall, Inc. New Jersey

[Mac'77] F.J. MacWilliams and N.J.A. Sloane, "The Theory of Error-Correcting Codes," New York: North Holland, 1977.

[Man'72] David Mandelbaum, "Error Correction in Residue Arithmetic," IEEE Trans. on Computers, Vol. C-21, No. 6, June 1972, pp. 538-545.

[Man'82] T.E. Mangir and A. Avizienis, "Fault-Tolerant Design for VLSI: Effect of Interconnect Requirements on Yield Improvements of VLSI Design," IEEE Trans. on Computers, Vol. C-31, No. 7, July 1982, pp.609-616.

[Mat'70] F. Mathur, and A. Avizienis, "Reliability analysis and architecture of hybrid-redundant digital system: Generalized triple modular redundancy with self repair," AFIPS Conf. Proc., 1970 SJCC, Vol. 36, pp. 375-383. Baltimore: Spartan Books.

[McC'82a] J.V. McCanny and J.G. McWhirter, "Implementation of Signal Processing Functions using 1-bit Systolic Arrays," Electronic Letters, Vol. 18, No. 6, March 1982, pp. 241-243.

[McC'82b] J.V. McCanny, and J.G. McWhirter,"Completely Iterative, Pipelined Multiplier Array Suitable for VLSI," IEE Proceedings, Vol 129, Pt. G, No. 2, April 1982, pp. 40-46.

[McC'82c] J.V. McCanny and J.G. McWhirter, "Implementation of Signal Processing Functions using 1-bit Systolic Arrays," Electronic Letters, Vol. 18, No. 6, March 1982, pp. 241-243.

[McC'83] J.V. McCanny, and J.G. McWhirter," Bit-level Systolic Array Circuit for Matrix Vector Multiplication," IEE Proceedings, Vol 130, Pt. G, No. 4, Agust 1983, pp. 125-130.

[McC'84] J.V. McCanny, and J.G. McWhirter,"Optimised Bit Level Systolic Array for Convolution," IEE Proceedings, Vol 131, Pt. G, No. 6, October 1984, pp. 632-637.

[McC'86] J.V. McCanny, R.A. Evans, and J.G. McWhirter, "Use of Unidirectional Data Flow in Bit Level Systolic Array Chips" Electronics Letters, Vol. 22, No. 10, May 1986, pp. 540-541.

[McC'87] J.V. McCanny and J.G. McWhirter, " Some Systolic Array Developments in the United Kingdom," (invited paper), Computer Magazine, Vol 20, N0. 7, July 1987, pp.51- 62.

[McE'85] Robrt J. McEliece, " The Reliability of Computer Memories," American Scientific, Vol. 252, No. 1, January 1985, pp. 88-95.

[McE'87] Robrt J. McEliece, " Finite Fields for Computer Scientists and Engineers," Kluwer Academic Publishers, Norwell, MA. 1987.

[McW'83] J.G. McWhirter, "Systolic Arrays for Recursive Least-Square Minimisation," Electronic Letters, Vol. 19, No. 18, September 1983, pp. 729-730.

[Mil'84] Dale D. Miller and John N. Polky, " An Implementation of the LMS Algorithm in the Residue Number System" , in Residue Number System Arithmetic: Modern Applications in Digital Signal Processing, Edited by M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor. IEEE Press, New York, 1986.

[Mol'83] D.I. Moldovan, "On the Design of Algorithms for VLSI Systolic Arrays," Proc. of IEEE, Vol. 71, No. 1, Jan. 1983, pp. 113-120.

[Mol'86] D.I. Moldovan, and J.A.B. Fortes, "Partitioning and Mapping Algorithms Into Fixed-Sized Systolic Arrays," IEEE Trans. on Computers, Vol. C-35, No. 1, Jan. 1986, pp. 1-12.

[Nag'83] H. K. Nagpal, G.A. Jullien, and W. C. Miller, " Processor Architectures for Two-Dimensional Convolvers Using a Single Multiplexed Element with Finite Field Arithmetic" , in Residue Number System Arithmetic: Modern Applications in Digital Signal Processing, Edited by M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor. IEEE Press, New York, 1986.

[Nud'85] G.R. Nudd, and G.R. Nash, "Application of Concurrent VLSI Systems to Two- Dimensional Signal Processing," in VLSI and Modern Signal Processing, edited by S.Y. Kung, H.J. Whitehouse, and T. Kailath, Prentice-Hall, INC., New Jersy, 1985.

[Neu'56] Von Numann, "Probablistic logics and synthesis of reliable organism from unreliable components," in Automata Studies, Annals of Mathematics Studies, No. 34, pp. 43-99. Princeton U. Press.

[Ope'75] A.V. Oppenheim and R.W. Schafer, "Digital Signal Processing," Prentice-Hall, Englewood Cliffs, N.J. 1975.

[Pet'58] W. W. Peterson, " On cchecking an adder," IBM J. Res. Devlop., Vol. 2, No. 2, pp. 166-168, April 1958.

[Pel'74] A. Peled, and B. Liu, " A New Hardware Relization of Digital Filters," IEEE Trans. on Acoustic, Speech, and Signal Processing, Vol. ASSP-22, No. 6, December 1974, pp. 456-462.

[Pin'82] Chares C. Pinter " A Book of Abstract Algebra" McGraw-Hill, New York. 1982.

[Pra'74] Pradhan D. , "Fault-Tolerant Carry-Save Adders," IEEE Trans. on Computers, Vol. C-23, No. 12, December 1974, pp. 1320-1322.

[Pri'86] W. Pries, A. Thanailakis, and H. Card, ' Group Properties of Cellular Automata and VLSI Applications," IEEE Trans. on Computers, Vol. C-35, No. 12, December 1986, pp. 1013-1024.

[Rab'75] Lawrence R. Rabiner, and Bernard Gold, " Theory and Application of Digital Signal Processing," Prentice-Hall, Englewood Cliffs, N.J. 1975.

[Ree'75] I.S. Reed and T.K. Trong," The Use of Finite Fields to Compute Convolutions," IEEE Trans. on Inform. Theory, Vol. IT-21, March 1975, pp. 208-213.

[Ren'84] David A. Rennels, "Fault-Tolerant Computing Concepts and Examples," IEEE Trans. on Computers, Vol. C-33, No. 12, December 1984, pp. 1116-1129.

[Sar'84] David B. Sarrazin and Miroslaw Malek, "Fault-Tolerant Semiconductor Memories," IEEE Computer Magazine, Vol. 17, No. 8, August 1984, pp. 49-56.

[Sei'84] Charles L. Seitz, "Concurrent VLSI Architectures," IEEE Transactions on Computers, Vol C-33, No. 12, December 1984, pp. 1247-1265.

[Sen'-] A.P. Senoy, R. Kumaresan, " Residue to Binary Conversion for RNS Arithmetic Using only Modular Look-Up Tables," Submitted for publication to IEEE trans. Circuits and Systems.

[Sie'84] Daniel P. Siewiorek, " Architecture of Fault-Tolerant Computers," IEEE Computer Magazine, Vol.17, No.8, August 1984, pp. 9-18.

[Sie'86] D. Siewiorek, "Architecture of Fault-Tolerant Computers," in Fault-Tolerant Computing: Theory and Techniques, Dhiraj K. Pradhan editor, Prentice-Hall 1986, pp. 417-466.

[Sod'86] M.A. Soderstrand, W.K. Jenkins, G.A. Jullien, F.J. Taylor, "Residue Number System Arithmetic: Modern Applications in Digital Signal Processing," IEEE Press, 1986.

[Sod'83] M.A. Soderstrand, C. Vernia, and Jui-Hua Chang, " An Improved Residue Number System Digital-to-Analog Converter," IEEE Trans. on Circuits and Systems, Vol. CAS-30, No.12, December 1983, pp.903-907.

[Sto'73] Harold S. Stone " Discrete Mathematical Structures and their Applications" Science Research Associates, INC. Chicago, 1973.

[Sny'82] L. Sny der,"Introduction to the configurable, Highly Parallel Computer,"Computer, Vol. 15, No. 1, Jan. 1982, pp. 47-64.

[Sza'67] N.S. Szabo and R.I Tanaka, "Residue Arithmetic and its Applications to Computer Technology," McGraw-Hill, New York, 1967.

[Svo'55] A. Svoboda and M. Valach, "Operational circuits," Storje Na Zpracovani Informací, Vol. 3 Nakl. CSAV, Praha, 1955.

[Tah'87a] M. Taheri, G.A. Jullien, and W.C. Miller, " Fault Detection in RNS Systolic Arrays," IEE Electronics Letters, Vol. 23, No. 4, Feb. 1987, pp. 165-166

[Tah'87b] M. Taheri, G.A. Jullien, and W.C. Miller, " Systolic ROM Arrays For Implementing RNS FIR Filters," IEEE Intr. Conf. on Acoustics, Speech, and Signal Processing, ICASSP'87, April 1987,pp. 771-774.

[Tah'87d] M. Taheri, G.A. Jullien, and W.C. Miller, " Fault Detection in Distributed RNS Processing," IEEE Intr. Conf. on Computer Design: VLSI in Computers & Processors, October 1987, pp 302-305.

[Tah'87e] M. Taheri, G.A. Jullien, and W.C. Miller, "Fault-Tolerant Techniques for Finite Ring Arithmetic Processors," Submitted for publication in IEEE Transaction on Computers, May 1987.

[Tah'88] M. Taheri, G.A. Jullien, and W.C. Miller, "High Speed Signal Processing Using Systolic Arrays Over Finite Rings," IEEE Journal of Selected Areas in Communications April 88.

[Tam'84] Yuval Tamir and Carlo H. Sequin, "Design and Application of Self-Testing Comparators Implemented with MOS PLA's " IEEE Trans. on Computers, Vol. C-33. No. 6, June 1984, pp. 493-506.

[Tan'84] R. Michael Tanner, " Fault-Tolerant 256K Memory Designs," IEEE Trans. on Computers, Vol. C-33, No. 4, April 1984, pp. 314-322.

[Tay'84] Fred J. Taylor, "Residue Arithmetic: A Tutorial with Examples," IEEE Computer Magazine, Vol. 17, No.5, May 1984, pp. 50-62.

[Tsi'87] Tsividis P. Yannis, "Operation and Modeling of the MOS Transistor" MacGraw-Hill, 1987.

[Urq'84] R.B. Urquhart, and D. Wood,"Systolic Matrix Vector Multiplication Methods for Signal Processing," IEE Proceedings, Vol 131, Pt. F, No. 6, October 1984, pp. 623-631.

[Wak'78] John Wakerly, "Error Detecting Codes, Self-Checking Circuits and Applications," North-Holland, New york, 1978.

[Zak'84] Vasilii Zakharov, " Parallelism and Array Processing," IEEE Trans. on Computers, Vol. C-33, No. 1, January 1984, pp. 45-76.