

*VLSI Implementations of  
Number Theoretic  
Concepts with  
Applications in Signal  
Processing*

*G. A. Jullien, N.M. Wigley and J. Reilly*

---

U N I V E R S I T Y O F

---

**WINDSOR**

---

*V L S I R e s e a r c h G r o u p*

---

# Table of Contents

1. Introduction.....	1
2. Basic Number Theory and Signal Processing.....	5
2.1 Chinese Remainder Theorem.....	6
3. Inner Products and Finite Rings.....	7
3.1 Direct Product Rings.....	7
3.2 Residue Number System and Isomorphisms.....	8
3.3 Polynomial Rings and Quotient Rings.....	9
3.4 Quadratic Residue Number System.....	10
3.5 Polynomial Rings and Modulus Replication.....	11
4. Multivariate Polynomial Representations and General Theory.....	13
4.1 Main Theorems.....	14
4.2 Representing Real Integers as Polynomials.....	17
4.3 Computation of Inner Products.....	19
4.4 Isomorphisms and Homomorphisms.....	20
4.5 Representations of Complex Sequences.....	24
4.6 Direct Product Representations.....	25
5. Implementation Examples.....	26
5.1 A DSP Example Using a Single Indeterminate (Discrete Cosine Transform).....	26
5.2 Scaling.....	29
5.3 Using Multiple Indeterminates.....	31
5.4 FFT Butterfly Computation.....	33
5.4.1. The Mapping Strategy.....	33

5.4.2. Scaling and Decoding.....	35
5.4.3. Experimental Error Analysis.....	37
6. VLSI Implementation of Pipelined Residue Computations .....	38
6.1. Modular Arithmetic Elements .....	39
6.2 Bit Steered ROM Structures.....	41
6.2.1. The Finite Ring IPSP.....	41
6.3 An Introduction to Switching Trees .....	48
6.4. Definitions and Rules .....	50
6.5. Switching Tree Circuits.....	54
6.6 Modulo 7 Multiplier .....	55
6.7 Results .....	57
7. Conclusions.....	58
8. Acknowledgements .....	59
9. References .....	59

# VLSI Implementations of Number Theoretic Concepts with Applications in Signal Processing

G.A. Jullien, N.M. Wigley  
VLSI Research Group  
University of Windsor

J. Reilly  
Communications Research Lab  
McMaster University

## 1. Introduction

This chapter explores novel techniques involving number theoretic concepts to perform real-time digital signal processing for high bandwidth data stream applications. Often the arithmetic manipulations are simple in form (cascades of additions and multiplications in a well defined structure) but the numbers of operations that have to be computed every second can be enormous. As an example let us consider a basic problem; the filtering of standard television pictures in real-time. We will assume that a two-dimensional filter is required and is given by equation (1.1):

$$y_{m,n} = \sum_{i=-2}^2 \sum_{j=-2}^2 w_{i,j} \cdot x_{m-i,n-j} \quad (1.1)$$

Here  $w_{i,j}$  is the filter kernel, and represents the weights on a two-dimensional averaging procedure around each output sample ( $y_{m,n}$ ). The averaging is based on a neighborhood of input samples  $\{x_{i,j}\}$ , that extends 5 samples in each dimension and centered on the output pixel position. The double summation represents two-dimensional convolution, a basic operation to all linear signal processing problems since it represents the mapping function of an input signal to an output signal through a linear system. The real time constraint is important since, as we will see, an extremely large computational requirement will be necessary in order to construct the filter.

In order to see the speed of operation required by our filter, we will perform some elementary calculations based on typical requirements for the filter. We assume a 525 line scan with 500 picture elements (pixels) in each line and that each primary colour is to be filtered independently with the  $5 \times 5$  filter. With an interlaced system we need to filter 30 images every second. Each output pixel (made up of three primary colour sub-pixels) requires 75 multiply/accumulate operations (MACs) to implement the filter. There are 262,500 pixels in each image and so  $1.96875 \times 10^7$  MACs are required for each image. Since we require to filter 30 images per second, the total number of MACs per second is  $5.9 \times 10^8$ . Assuming that a MAC is considered a basic operation in the processor used to filter the image, then we need a machine that delivers 590 Mega MACs per second (MOPS). In these terms we require a super computer to deliver this performance!

The filtering operation has the advantage of not requiring the full features of a general purpose computer (the filtering operation does not have to be performed within the typical multi-user, complex operating system environment of a main frame machine) and the computations are simply repeated arithmetic operations in a predetermined structure. In fact, it is rather pointless to talk about the computational requirements in normal general computer terms (MIPs, MOPs, MFLOPs, etc.), but rather to talk in direct terms of the data flow requirements; that is, the throughput rate of the filter in samples per second (in this example, a sample is a pixel).

Since we are going to be directly concerned with the implementation of DSP arithmetic, it might be helpful to finish off our example by considering the arithmetic details of each MAC. The representation of each primary colour pixel by 8-bits is more than adequate for very high definition of the image and the quantization of each filter weight by 10-bits is probably sufficient for the types of filter we are interested in building. This naturally leads to an 18-bit multiplier structure. Since there are an accumulation of 25 such multiplications for each output pixel, an extra 5-bits are required for this. With 23-bits we can, in fact, compute each output pixel with no error in the computation process. Since 23-bits is a pessimistic upper bound for the computational dynamic range (only required if both the filter coefficients and the input pixels are at their maximum values -

a very rare occurrence), we can probably specify a 20-bit range for accurate computation of most output samples. A more accurate determination of computational dynamic range can be on the basis of preserving the 'energy' of input and output data streams; i.e.

$$\sum_{j=0}^{N-1} \sum_{i=0}^{N-1} y_{j,i}^2 = \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} x_{j,i}^2 \quad (1.2)$$

over some suitable sample cover (perhaps a frame of typical data). Such an energy bound will reduce the computational dynamic range by several bits compared to the pessimistic bound based on the assumption that both data and filter coefficients are at the maximum value [11].

This calculation leads us to the important observation that we do not need floating point arithmetic. The example represents a fine-grained computation (one in which successive parts of the computation are of a limited dynamic range) and is typical of many DSP algorithms. The limitations we are able to impose on the arithmetic calculations will be a great help in reducing the hardware required in the very high-speed throughput systems to be discussed in this paper.

For our example filter we have to filter pixels at a rate of  $7.875 \times 10^6$  pixels per second. Current  $1\mu$  CMOS technology, with aggressive architectures and design approaches, allows approximately 40 MACs of this dynamic range on a single  $1\text{cm}^2$  silicon die operating at 100M samples/second [5], providing  $4 \times 10^9$  MACs/second. We have therefore reduced the original requirement of a supercomputer to a system contained on only a portion of a state-of-the-art 'chip'.

It is the purpose of this chapter to discuss ways in which number theoretic techniques can be used to perform DSP operations, such as this example filter, by both reducing the amount of hardware involved in the circuitry, and by allowing the construction of very benign architectures down to the individual cells used in the physical implementation.

In this chapter we will restrict ourselves to the computation of linear filter and transform algorithms which have, for the one-dimensional case, the inner product form:

$$y_n = \sum_{m=0}^{N-1} x_m \cdot w_{f(n,m)} \quad (1.3)$$

In the case of linear filtering (convolution)  $f(n,m) = n-m$ ; for the case of a linear transform with the convolution property,  $w_{f(n,m)}$  is a primitive  $N$ th root of unity. For the DFT,  $w_{f(n,m)} = e^{-i\frac{2\pi}{N}nm}$ . Although this limitation appears overly restrictive, the general form of equation (1.3) probably accounts for the vast majority of digital signal processing functions implemented commercially.

Computations using finite rings can offer distinct advantages over integer computations using binary arithmetic. The most visible use of such computations is in the coding of integers as elements of a set of rings, with relatively prime moduli, allowing large dynamic range closed operations (addition, multiplication) to be carried out by a set of parallel small ring calculations. This is known as the Residue Number System (RNS) [15]. If the calculations are carried out simultaneously, the independence of the calculations (there are no dynamic range carries) allows a relaxation in synchronization requirements that can have considerable advantages in VLSI implementations [7]. We will start by reviewing the elements of number theory that we will use to introduce the various architectural techniques. We will then show the use of RNS systems in building signal processing systems, finally introducing a recently discovered technique that removes mapping and implementation restrictions inherent in current RNS techniques.

## 2. Basic Number Theory and Signal Processing

In RNS systems we deal with rings, or fields, that are used for the actual implementation and rings that are isomorphic to direct products of implementation rings or extensions of them. A given digital signal processing algorithm is mapped from real or complex integer arithmetic to the implementation rings, the computation is carried out there, and the result is then mapped back to obtain the final answer.

Let  $m$  be a positive integer. We denote by  $\mathfrak{R}(m)$  the ring of integers modulo  $m$ , i.e.

$$\mathfrak{R}(m) = \{S: \oplus_m, \otimes_m\}; S = \{0, 1, \dots, m-1\} \quad (2.1)$$

Hence  $\mathfrak{R}(m)$  consists of the elements of  $S$  together with the notions of addition and multiplication. If  $a, b \in S$  then  $a \oplus_m b$  denotes that (unique) element  $c$  of  $S$  for which  $a + b - c$  is an integer multiple of  $m$ , i.e.  $m$  divides  $a + b - c$ . Similarly  $a \otimes_m b$  denotes that (unique) element  $d$  of  $S$  for which  $ab - d$  is divisible by  $m$ . Using the notation  $a \equiv c \pmod{m}$  to mean that  $m$  divides  $a - c$ , we observe that  $a + b \equiv a \oplus_m b$ , and  $ab \equiv a \otimes_m b$ .

We can extend the notion of addition and multiplication from the elements of  $S$  to all of the integers. If  $a$  is any integer whatsoever then we denote by  $\langle a \rangle_m$  the unique integer in  $S$  for which  $m$  divides the difference  $a - \langle a \rangle_m$ , i.e.  $a \equiv \langle a \rangle_m \pmod{m}$ . It is easily established that

$$\langle \langle a \rangle_m + \langle b \rangle_m \rangle_m = \langle a + b \rangle_m \text{ and } \langle \langle a \rangle_m \cdot \langle b \rangle_m \rangle_m = \langle a \cdot b \rangle_m \quad (2.2)$$

Our notation  $a \oplus_m b$  and  $a \otimes_m b$  will imply the residue reduction of  $a$  and  $b$ .

The integer  $\langle a \rangle_m$  is called the least residue of  $a \pmod{m}$ . The following theorem is critical to what follows.

**Theorem 1** If  $a$  and  $b$  are integers satisfying  $-m < a - b < m$ , and if  $a \equiv b \pmod{m}$ , then  $a = b$ .

**Proof:** The congruence  $a \equiv b \pmod{m}$  says that  $a - b$  is a multiple of  $m$ , say  $a - b = km$ . The inequalities show that  $k$  must equal zero.

If  $\mathfrak{R}_1$  and  $\mathfrak{R}_2$  are any two rings then we can define the cross-product ring  $\mathfrak{R}_1 \times \mathfrak{R}_2$  as the set of pairs  $(s_1, s_2) \in S_1 \times S_2$ , with addition and multiplication defined componentwise, i.e. by

$$\begin{aligned} (a_1, a_2) \oplus_{\mathfrak{R}_1 \times \mathfrak{R}_2} (b_1, b_2) &= (a_1 \oplus_{\mathfrak{R}_1} b_1, a_2 \oplus_{\mathfrak{R}_2} b_2) \\ (a_1, a_2) \otimes_{\mathfrak{R}_1 \times \mathfrak{R}_2} (b_1, b_2) &= (a_1 \otimes_{\mathfrak{R}_1} b_1, a_2 \otimes_{\mathfrak{R}_2} b_2) \end{aligned} \quad (2.3)$$

As an example, let  $\mathfrak{R}_1 = \mathfrak{R}(5)$  and  $\mathfrak{R}_2 = \mathfrak{R}(7)$ . Then  $(2,3) \oplus_{\mathfrak{R}_1 \times \mathfrak{R}_2} (4,6) = (2 \oplus_5 4, 3 \oplus_7 6) = (1,2)$  and  $(2,3) \otimes_{\mathfrak{R}_1 \times \mathfrak{R}_2} (4,6) = (2 \otimes_5 4, 3 \otimes_7 6) = (3,4)$ .

## 2.1 Chinese Remainder Theorem

When  $m_1$  and  $m_2$  are moduli which are relatively prime there is a special relationship between the rings  $\mathfrak{R}(m_1 m_2)$  and the cross-product ring  $\mathfrak{R}(m_1) \times \mathfrak{R}(m_2)$ , namely, that they are isomorphic. In particular one such isomorphism (there may be several isomorphisms) is given by mapping the element  $x \in \mathfrak{R}(m_1 m_2)$  to the element  $(x_1, x_2)$  where  $x_1 = \langle x \rangle_{m_1}$  and  $x_2 = \langle x \rangle_{m_2}$ . It is not difficult to check that this map is indeed a homomorphism, and that it is 1-1. That it is an onto map is given by the *Chinese Remainder Theorem*, which states the following [19]:

*Chinese Remainder Theorem.* Let  $m_1, m_2, \dots, m_r$  denote  $r$  positive integers that are relatively prime in pairs, and let  $a_1, a_2, \dots, a_r$  denote any  $r$  integers. Then the congruences  $x \equiv a_i \pmod{m_i}$ ,  $i = 1, 2, \dots, r$ , have common solutions. Any two solutions are congruent modulo  $m_1 m_2 \dots m_r$ .

## 3. Inner Products and Finite Rings

In order to perform the computation within finite rings, using the RNS and CRT mapping isomorphism, we select several relatively prime moduli,  $m_1, m_2, \dots, m_r$ , with  $M$  denoting the product,  $M = m_k$ . The integers  $a_k$  and  $b_k$  (assumed real for the moment) are then considered as elements of the ring  $\mathfrak{R}(M)$ , and the computation of the inner product is presumed to be carried out in this ring. Suppose the input integers are all subject to a common bound, say  $0 \leq |a_k| \leq L$ ,  $0 \leq |b_k| \leq L$ . The inner product  $\sum_{k=0}^{N-1} a_k b_k$  then has the bound  $NL^2$ . Thus under the assumption that  $M > NL^2$ , the computation may be carried out in the ring  $\mathfrak{R}(M)$ , because the answer then must lie in the  $S = \{0, 1, \dots, M-1\}$ , and the answer obtained by modular arithmetic will be the same as the true answer; this follows from theorem 1.

### 3.1 Direct Product Rings

In general we require a modulus  $M$  which is i) large enough to allow the full dynamic range of the computation, and ii) factors into pairwise relatively prime factors:  $\{m_k\}$ . We then use the isomorphism guaranteed by the Chinese Remainder Theorem to map the computation from the (large) ring  $\mathfrak{R}(M)$  to the smaller rings  $\mathfrak{R}(m_k)$ .

We will define the ring (field) of computation by:

Base Ring (Field<sup>1</sup>) :  $\mathfrak{R}(m_k)$  or  $\text{GF}(m_k) = \{S: \oplus_{m_k}, \otimes_{m_k}\}; S = \{0, 1, \dots, m_k-1\}$

We will define the ring, to which the computation is finally mapped, as:

Direct Product Ring:  $\mathfrak{R}(m_k) = \{ \overline{S} : \overline{\oplus}, \overline{\otimes} \}; \overline{S} = \{0, 1, \dots, M-1\}; M = \prod_{k=1}^L m_k.$

Note that direct product rings are not rings over which actual implementation of the digital signal processing algorithm takes place; they are the direct product of the corresponding implementation rings. We have therefore used the symbols  $\overline{\oplus}$  and  $\overline{\otimes}$  rather than the expected symbols  $\oplus_M$  and  $\otimes_M$ . Results become available over these rings following the appropriate isomorphic mapping (e.g. the Chinese Remainder Theorem; for others, see below).

### 3.2 Residue Number System and Isomorphisms

This is a brief introduction using the above notation. A digit in the residue number system is represented by an L-tuple of residues [15]:

$$X = (x_0, x_1, \dots, x_{L-1}) \tag{3.1}$$

---

<sup>1</sup> When  $m_k$  is prime the ring is a field, and is frequently denoted by  $\text{GF}(m_k)$  (for Galois).

where  $x_i = \langle X \rangle_{m_i}$  is the  $i$ th residue and  $m_i$  is the  $i$ th modulus. Closed computations (addition, multiplication) over the implementation rings map to the direct product ring via the Chinese Remainder Theorem (CRT). We will write this succinctly as:

$$A \oplus B \Leftrightarrow \{a_0 \oplus b_0, a_1 \oplus b_1, \dots, a_L \oplus b_L\}; \quad A \otimes B \Leftrightarrow \{a_0 \otimes b_0, a_1 \otimes b_1, \dots, a_L \otimes b_L\}$$

with  $A, B \in \mathfrak{R}(M)$ ;  $a_k, b_k \in \mathfrak{R}(m_k)$  and  $\mathfrak{R}(M) \cong_{\otimes} \mathfrak{R}(m_k)$  (direct product).

The isomorphism ( $\cong$ ) between  $\mathfrak{R}(M)$  and the direct product of  $\{\mathfrak{R}(m_k)\}$  means that calculations over  $\mathfrak{R}(M)$  can be effectively carried out, over each  $\mathfrak{R}(m_k)$ , independently and in parallel. A final mapping (e.g. CRT) to  $\mathfrak{R}(M)$  is performed at the end of a chain of calculations. We have therefore broken down a calculation set in a large dynamic range,  $M$ , to a set of  $L$  calculations set in small dynamic ranges given by the  $\{m_k\}$ . This is the main advantage of using the RNS over a conventional weighted value numbering system (e.g. binary).

The final mapping is found from the CRT:

$$X = \sum_{k=1}^L \underset{M}{\overset{\wedge}{m}_k} \left\{ \overset{\wedge}{m}_k \otimes_M \left[ x_k \otimes_{m_k} (\overset{\wedge}{m}_k)^{-1} \right] \right\} \quad (3.2)$$

with  $\overset{\wedge}{m}_k = M / m_k$ ,  $X \in \mathfrak{R}(M)$ ,  $x_k \in \mathfrak{R}(m_k)$  and  $(\bullet)^{-1}$  the multiplicative inverse operator. We have also used the notation  $\underset{M}{\overset{\wedge}{m}_k}$  to indicate summation over the ring  $\mathfrak{R}(M)$ .

### 3.3 Polynomial Rings and Quotient Rings

If  $\mathfrak{R}$  is a ring then we let  $\mathfrak{R}[X]$  denote the ring of polynomials in the indeterminate  $X$ :

$$\mathfrak{R}[X] = \left\{ \sum_{k=0}^n a_k X^k : a_k \in \mathfrak{R}, n \geq 0 \right\}$$

If  $X_1, X_2, \dots, X_S$  are indeterminates then we define the ring  $\mathfrak{R}[X_1, X_2, \dots, X_S]$  to be the ring of multivariate polynomials in the indeterminates  $X_1, X_2, \dots, X_S$ . Below we shall use such

polynomial rings where the base ring  $\mathfrak{R}$  is a modular ring  $\mathfrak{R}(M)$ ; in such a case we shall write  $\mathfrak{R}_M[X_1, X_2, \dots, X_S]$  in place of  $\mathfrak{R}(M)[X_1, X_2, \dots, X_S]$ .

For a given polynomial  $g(X) \in \mathfrak{R}[X]$  we consider the set  $(g(X))$  of all (polynomial) multiples of  $g(X)$ . This set  $(g(X))$  is called the ‘ideal’ generated by the polynomial  $g(X)$  in the ring  $\mathfrak{R}[X]$ . The quotient ring  $\mathfrak{R}[X]/(g(X))$  is then defined to consist of all elements of the form  $f(X) + (g(X))$ , with  $f(X) \in \mathfrak{R}[X]$ . The more usual way of considering the quotient ring is to consider sums and products of polynomials reduced according to the equation  $g(X) = 0$ , that is, to consider the remainder after division by  $g(X)$ . Let us, for example, take  $g(X) = X^3 - 3X^2 + X - 2$ . Then polynomials of degree  $\geq 3$  are reduced according to the rule  $X^3 = 3X^2 - X + 2$ . For example, if  $f(X) = 2X^2 + 4X - 1$  and  $h(X) = -3X^2 - X + 7$ , then the product becomes, in the ring  $\mathfrak{R}[X]/(g(X))$ ,

$$\begin{aligned} (2X^2 + 4X - 1) \cdot (-3X^2 - X + 7) &= -6X^4 - 14X^3 + 13X^2 + 29X - 7 \\ &= -6X(3X^2 - X + 2) - 14(3X^2 - X + 2) = -77X^2 + 49X - 71 \end{aligned}$$

where the equalities are considered as equal in the ring  $\mathfrak{R}[X]$ . This identity can be verified by checking that  $f(X)h(X) = g(X) \cdot q(X) + r(X) = g(X) \cdot (-6X - 32) - 77X^2 + 49X - 71$ , where  $q(X)$  is the quotient and  $r(X)$  is the remainder when the product  $f(X)h(X)$  is divided by  $g(X)$ .

### 3.4 Quadratic Residue Number System

In many applications of digital signal processing the data are given as complex integers. In this case the arithmetic can be carried out in two separate channels, but with cross-channel communication being required for any complex multiplications. This has the undesired effect of unduly complicating the hardware design .

An algebraic device to avoid this unwelcome complication is afforded by the Quadratic Residue Number System. In this system a preliminary mapping is performed to map the two streams of

data, the real and imaginary parts, to two new streams of data where the computation will be performed. In these two new streams there is no cross-channel communication, i.e. the computations are performed simultaneously and independently in each channel. The method is based on a peculiarity of number theory, namely, that for some moduli there exists the square root of -1.

The Quadratic Residue Ring is defined by using the solution of the polynomial  $X^2+1=0$ ;  $X = j = \sqrt{-1}$ , such that  $j \in \mathfrak{R}(m_k)$ . If the modulus  $m_k$  is prime then there will be either two such solutions in  $\mathfrak{R}(m_k)$  or zero solutions. Unlike the generation of extension fields by adjoining the solution of an irreducible polynomial, the *QR* ring is defined with a reducible polynomial; but from a well-known result of number theory, the reduction only takes effect if the prime  $m_k$  is of the form  $4s + 1$ , i.e.  $m_k \equiv 1 \pmod{4}$ . Effectively we represent  $j$  with an indeterminate  $X$  and then take the algebraic quotient modulo the polynomial  $X^2 + 1$ . We choose a modulus  $M$  such that  $X^2 + 1$  factors over the ring  $Z_M$ . The condition on  $M$  for the existence of such a factorization is that *each* of its prime factors be of the form  $4s + 1$ .

We formally define the *QR* and direct product *QR* rings as:

*Quadratic Residue Ring:*

$$QR(m_k) = \{S: \oplus, \otimes\}; S = \{A^\circ, A^*\} \text{ with } A^\circ, A^* \in \mathfrak{R}(m_k) \text{ and } A^\circ = a^r + ja^i, A^* = a^r - ja^i; \\ j = \sqrt{-1}; a^r, a^i, j \in \mathfrak{R}(m_k), m_k = \prod_i p_i^{e_i}, p_i = 4k+1, \text{ a prime. } A^\circ \text{ will be referred to as the } \textit{normal}$$

component of element  $\mathbf{A} = (A^\circ, A^*)$  and  $A^*$  as the *conjugate* component of element  $\mathbf{A}$ . The multiplication and addition operators both compute component-wise.

*Direct Product Quadratic Ring:*

$$QR(M) = \{ S : \overline{\oplus}, \overline{\otimes} \}; S = \{(A_1^\circ, A_1^*), \dots, (A_L^\circ, A_L^*)\}; M = \prod_{k=1}^L m_k.$$

We effectively form a residue number system as a direct product of a set of rings that support the  $QR$  requirements.

Following the  $QR$  mapping, we are able to carry out all closed complex calculations with only component-wise addition and multiplication. This both reduces the number of base field operations required for multiplication and effectively separates the two channels of computation.

### 3.5 Polynomial Rings and Modulus Replication

In the above formulations of the RNS and QRNS approaches to DSP it will be seen that an adequate dynamic range requires the use of large moduli. An increase in the size of the dynamic range is then effected by the incorporation of additional moduli. Each new modulus must, of course, be relatively prime to each of the existing moduli; if QRNS is desired then the new moduli must also satisfy the QRNS condition of having prime factors which are all congruent to 1 (mod 4).

Behind the use of RNS and QRNS are the desire for speed and efficiency of implementation. Since speed and efficiency of implementation come from parallelism, and parallelism comes from the direct-product representation of numbers, it seems natural to use algebraic formulations of the problem which lead to direct-product rings, namely quotient rings with respect to an ideal whose generators split into smaller factors. In the RNS method the modulus  $M$  splits into its factors  $m_i$ , and the ring  $Z_M$  splits into corresponding factors  $Z_{m_i}$ . An increase in the dynamic range, without a concomitant increase in the modulus  $M$ , may be gained through the judicious use of polynomial rings.

The Modulus Replication Residue Number System (MRRNS) is a technique that allows both real and complex inner product processing with replications of a very small number of moderately sized rings ( $m_k$  32).

The method has the following advantages:

- 1) There are no quantization problems. The data, either real or complex, are assumed to be of a given fixed bitlength. No approximations or scaling are used in encoding the data.
- 2) The polynomials used are of a general nature, so that no restrictions are placed on the prime divisors of the moduli, except in the case of a QRNS representation of complex data, in which case the condition is the usual one of  $p \equiv 1 \pmod{4}$  for prime divisors  $p$  of the modulus  $M$ .
- 3) The same small moduli can be used many times, which allows VLSI implementations of systems which can process data of a large bitlength, using direct products of many copies of modular rings with small moduli.
- 4) Encoding is a simple matter of diverting the bits of the input data to the proper channels. Decoding is only complicated insofar as the Chinese Remainder Theorem is used, and even then only for a limited number of small moduli. Scaling, if used in decoding, is simplified by the ring structures used; certain monomials can be ignored as they represent insignificant digits.

The indeterminates represent various powers of 2, thus allowing the data to be expressed as polynomials with small coefficients. These coefficients are then mapped to a direct product ring consisting of many copies of  $Z_M$  as factors. Since the above-mentioned coefficients are small, the modulus  $M$  does not have to be very large. The direct product repeats the factor  $Z_M$  many times, so that the same prime divisors of  $M$  are used repeatedly, thus obviating the need for additional, larger primes.

The following section presents a formal theoretical approach to the multivariate finite polynomial ring mapping technique.

## 4. Multivariate Polynomial Representations and General Theory

At this point we shall give a formal presentation of the theory. The following theorems can be used to represent real or complex integers as elements of direct-product rings. The integers are expanded in a certain binary form and the coefficients of the expansions are treated as coefficients of polynomials in several indeterminates, each indeterminate representing a different power of 2. The polynomials are then transformed by the isomorphism of theorem 2 below to elements of a direct product of RNS rings.

An excellent discussion on direct products of RNS and polynomial rings is given in [12]. A thorough treatise on the basic theory of polynomial rings is given in [4].

### 4.1 Main Theorems

In the following  $G(Y)$  will denote the ideal generated by the two polynomials  $F(X)$  and  $G(Y)$ .

**Lemma 1.** Let  $\mathfrak{R}$  be a commutative ring and let  $r_1, r_2, \dots, r_d$  be distinct elements of  $\mathfrak{R}$ . Let  $X$  be an indeterminate over  $\mathfrak{R}$ , let  $f(X) \in \mathfrak{R}[X]$ , and suppose for  $i = 1, 2, \dots, d$  that  $f(r_i) = 0$ . Then  $f(X) \in (g(X))$  where  $g(X) = \prod_{i=1}^d (X - r_i)$ .

**Proof.** This follows by induction from [4].

The next lemma is used to characterize the kernel of the evaluation homomorphisms used in Theorem 1 below.  $\square$

**Lemma 2.** Let  $\mathfrak{R}$  be a commutative ring with identity. For  $i = 1, 2, \dots, n$  let  $X_1, X_2, \dots, X_n$  be indeterminates over  $\mathfrak{R}$ , let  $d_i \geq 1$ , let  $r_{i1}, r_{i2}, \dots, r_{id_i} \in \mathfrak{R}$ , and for each such  $i$  suppose that the set  $\{r_{ij} - r_{ik} : 1 \leq j < k \leq d_i\}$  consists of invertible elements of  $\mathfrak{R}$ . Let  $f(X_1, X_2, \dots, X_n) \in \mathfrak{R}[X_1, X_2, \dots, X_n]$  and suppose that  $f(r_{1j_1}, r_{2j_2}, \dots, r_{nj_n}) = 0$  for each  $i = 1, 2, \dots, n$  and each  $j_i =$

$1, 2, \dots, d_i$ . Then  $f$  belongs to the ideal  $G$  generated by the polynomials  $g_i(X_i) = \prod_{j=1}^{d_i} (X_i - r_{ij})$ ,  $i = 1, 2, \dots, n$ .

**Proof.** For the case  $n = 1$  the lemma follows from Lemma 1. Assume the lemma is true for any  $n$  indeterminates and that we have the hypotheses of the lemma for  $n+1$  variables  $X_1, X_2, \dots, X_n, X_{n+1}$ . The set  $\{d_i; 0 \leq i \leq n+1\}$  has a least element; without loss of generality assume it is  $d_{n+1}$ .

If  $d_{n+1} = 1$  then we can write, using the division algorithm,

$$f(X_1, X_2, \dots, X_{n+1}) = f_0(X_1, X_2, \dots, X_{n+1})(X_{n+1} - r_{n+1,1}) + f_1(X_1, X_2, \dots, X_n).$$

Set  $X_{n+1} = r_{n+1,1}$ . Since by hypothesis the polynomial  $f(X_1, X_2, \dots, X_n, r_{n+1,1})$  vanishes for  $X_i = r_{i,j_i}$  for  $i = 1, 2, \dots, n$  and  $1 \leq j_i \leq d_i$ , it follows that  $f_1(X_1, X_2, \dots, X_n) \in (g_1(X_1), \dots, g_n(X_n))$ .

Since  $d_{n+1} = 1$ , the generator of  $(g_{n+1}(X_{n+1}))$  is  $X_{n+1} - r_{n+1,1}$ . It follows that  $f(X_1, \dots, X_{n+1}) \in (g_1(X_1), \dots, g_{n+1}(X_{n+1})) = G$ .

Assume the lemma true for any values of  $d_1, d_2, \dots, d_{n+1}$  when the least of these numbers is equal to  $\hat{d} - 1$ . Assume now that the lemma is to be proved when the minimum of the  $d_i = \hat{d} + 1$ ; again we assume that the minimum is attained at  $d_{n+1}$ , so that  $d_{n+1} = \hat{d} + 1$ .

As before we write, again using the division algorithm,

$$f(X_1, X_2, \dots, X_{n+1}) = f_0(X_1, X_2, \dots, X_{n+1})(X_{n+1} - r_{n+1,d_{n+1}}) + f_1(X_1, X_2, \dots, X_n).$$

First we set  $X_{n+1} = r_{n+1,d_{n+1}}$ . Again we see that  $f_1(X_1, \dots, X_n) = f(X_1, \dots, X_n, r_{n+1,d_{n+1}})$  vanishes for  $X_i = r_{i,j_i}$  when  $i = 1, 2, \dots, n$  and  $1 \leq j_i \leq d_i$ , and thus  $f_1(X_1, \dots, X_n) \in (g_1(X_1), \dots, g_n(X_n))$ .

Now setting  $X_{n+1}$  equal to  $r_{n+1,j}$  for any  $1 \leq j < d_{n+1}$  and retaining the values  $X_i = r_{i,j_i}$ , we obtain

$$0 = f_0(r_{1j_1}, r_{2j_2}, \dots, r_{nj_n}, r_{n+1,j})(r_{n+1,j} - r_{n+1,d_{n+1}}).$$

Since the second factor is, by hypothesis, invertible, it follows that the polynomial  $f_0$  vanishes at the indicated roots. By the inductive hypotheses this implies that  $f_0$  belongs to the ideal generated by the polynomials  $g_1(X_1), g_2(X_2), \dots, g_n(X_n)$ , and  $\hat{g}(X_{n+1})$ , where  $\hat{g}(X_{n+1}) = \prod_{j=1}^{d_{n+1}} (X_{n+1} - r_{n+1,j})$ , the product taken over  $1 \leq j \leq d_{n+1}$ . Finally, we have  $g_{n+1}(X_{n+1}) = \hat{g}(X_{n+1})(X_{n+1} - r_{n+1,d_{n+1}})$ , and thus the inductive step is complete.  $\square$

**Theorem 1.** Under the same hypotheses as in Lemma 2, there exists an isomorphism between  $\mathfrak{R}[X_1, X_2, \dots, X_n]/G$  and  $\mathfrak{R} \times \mathfrak{R} \times \dots \times \mathfrak{R}$ , where the direct product is taken  $\prod_{i=1}^n d_i$  times.

**Proof:** For  $k = 1, 2, \dots, n$  and  $1 \leq j_k \leq d_k$  we define the components  $\Phi_{j_1 j_2 \dots j_n} : \mathfrak{R}[X_1, X_2, \dots, X_n] \rightarrow \mathfrak{R}$  of a ring homomorphism  $\Phi$  from  $\mathfrak{R}[X_1, X_2, \dots, X_n]$  to  $\mathfrak{R} \times \mathfrak{R} \times \dots \times \mathfrak{R}$  by setting

$$\Phi_{j_1 j_2 \dots j_n} \left( \sum_{i_1, i_2, \dots, i_n} a_{i_1 i_2 \dots i_n} X_1^{i_1} X_2^{i_2} \dots X_n^{i_n} \right) = \sum_{i_1, i_2, \dots, i_n} a_{i_1 i_2 \dots i_n} r_{1j_1}^{i_1} r_{2j_2}^{i_2} \dots r_{nj_n}^{i_n}$$

Here the sum is taken over  $0 \leq i_k \leq d_k$ . Thus the component homomorphisms, applied to a polynomial, are defined by evaluating the polynomial at the corresponding roots of the  $g_i$ . The map is onto because the component homomorphisms act as the identity on constant polynomials. The kernel of the map, by Lemma 2, is clearly the ideal  $G$ .  $\square$

We now address the question of existence of such isomorphisms.

**Lemma 3.** Let  $M$  and  $d$  be positive integers. Then there exist distinct elements  $r_1, r_2, \dots, r_d \in Z_M$  such that each element of  $\{r_i - r_j : 1 \leq i < j \leq d\}$  is invertible in  $Z_M$  if and only if each prime divisor  $p$  of  $M$  satisfies  $p \nmid d$ .

Proof. If each such  $p \leq d$  then take, e.g.,  $r_i = i$  for  $1 \leq i \leq d$ . For such values the congruence  $i \equiv j \pmod{p}$  is impossible, and hence  $r_i - r_j$  is relatively prime to each  $p$  dividing  $M$ , and thus invertible in  $Z_M$ .

Conversely if some prime  $p$  divides  $M$  and  $p < d$  then any set  $\{r_1, r_2, \dots, r_d\}$  of residues modulo  $M$  will be reduced modulo  $p$  to a set in which two residues are congruent  $\pmod{p}$ . Then the corresponding difference  $r_i - r_j$  is not relatively prime to  $M$  and hence cannot be invertible. This completes the proof.  $\square$

The next theorem gives the criteria for the modulus  $M$  and its prime divisors, as well as some sufficient conditions for the polynomials which generate the ideal  $G$ . Both  $M$  and  $G$  determine the direct-product representation used for the actual computations.

Theorem 2. Let  $M, n$  and  $d_1, d_2, \dots, d_n$  be positive integers. Let  $X_1, X_2, \dots, X_n$  be indeterminates over the ring  $Z_M$ . Suppose that for each prime divisor  $p$  of  $M$  we have  $p \nmid d_i$  for  $i = 1, 2, \dots, n$ . Then for each such  $i$  there exist elements  $r_{i1}, r_{i2}, \dots, r_{id_i} \in Z_M$  such that each difference  $r_{ij} - r_{ik}$  ( $1 \leq j < k \leq d_i$ ) is invertible in  $Z_M$ . Let  $g_i(X_i) \in Z_M[X_i]$  be the monic polynomial of degree  $d_i$  whose roots are  $\{r_{ij}: 1 \leq j \leq d_i\}$ . Let  $G$  be the ideal in  $Z_M[X_1, X_2, \dots, X_n]$  generated by the polynomials  $g_1(X_1), g_2(X_2), \dots, g_n(X_n)$ . Then there exists a ring isomorphism

$$Z_M[X_1, X_2, \dots, X_n]/G \cong Z_M \times Z_M \times \dots \times Z_M$$

where the direct product on the right is taken  $\prod_{i=1}^n d_i$  times.

Proof. The existence of the elements  $r_{ij}$  follows from lemma 3 above. The isomorphism is given in Theorem 1.  $\square$

## 4.2 Representing Real Integers as Polynomials

Let there be given a sequence of integers  $B = \beta_0 > \beta_1 > \beta_2 > \dots > \beta_n = \beta$ . We shall give a method of representing all  $B$ -bit nonnegative integers  $s \in Z$  as elements of the ring  $Z[X_1, X_2, \dots, X_n]$ . Such representations are in general not unique, and we shall not be concerned with matters of uniqueness.

Assume  $0 \leq s < 2^{\beta_0} - 1$  and expand  $s$  in powers of  $2^{\beta_1}$  :

$$s = \sum_{i_1=0}^{d_1} s_{i_1} 2^{i_1 \beta_1} \quad (4.1)$$

where  $0 \leq s_{i_1} < 2^{\beta_1} - 1$ . Any integer  $s$  in the interval  $[0, 2^{\beta_0} - 1]$  can be represented in the form (4.1) provided we make the assumption  $d_1 + 1 \leq \beta_0 / \beta_1$ .

We then expand the coefficients  $s_{i_1}$  in powers of  $2^{\beta_2}$ :

$$s_{i_1} = \sum_{i_2=0}^{d_2} s_{i_1 i_2} 2^{i_2 \beta_2} \quad (4.2)$$

where  $0 \leq s_{i_1 i_2} < 2^{\beta_2} - 1$ . We now make the requirement  $d_2 + 1 \leq \beta_1 / \beta_2$  in order to assure the existence of the representation (4.2) for  $0 \leq s_{i_1} < 2^{\beta_1} - 1$ .

Continuing, we arrive at

$$s = \sum_{i_1=0}^{d_1} \sum_{i_2=0}^{d_2} \dots \sum_{i_n=0}^{d_n} s_{i_1 i_2 \dots i_n} 2^{i_1 \beta_1} 2^{i_2 \beta_2} \dots 2^{i_n \beta_n} \quad (4.3)$$

where the coefficients satisfy (recall  $\beta_n = \beta$ )

$$0 \leq s_{i_1 i_2 \dots i_n} < 2^{\beta} - 1. \quad (4.4)$$

Again we require  $d_n + 1 \leq \beta_{n-1} / \beta_n$ .

We can also include negative integers  $s < 0$ , by expanding the mantissa of  $s$ , i.e. the positive integer  $-s$ . Thus it follows that we can represent any integer  $s$ , with one sign bit and  $B$  mantissa bits, by writing its mantissa in the form (4.1) with the restriction (4.4).

Next, for  $i = 1, 2, \dots, n$  we replace the radix  $2^{\beta_i}$  with the indeterminate  $X_i$ . Thus the integer  $s$  becomes a polynomial in the indeterminates  $X_1, X_2, \dots, X_n$ :

$$s = \sum_{i_1=0}^{d_1} \sum_{i_2=0}^{d_2} \dots \sum_{i_n=0}^{d_n} s_{i_1 i_2 \dots i_n} X_1^{i_1} X_2^{i_2} \dots X_n^{i_n} \quad (4.5)$$

evaluated at  $X_i = 2^{\beta_i}, i = 1, 2, \dots, n$ . This yields the following theorem.

**Theorem 3.** Let  $B = \beta \prod_{i=1}^n (1 + d_i)$ . Then any integer  $s$  (or its negative) lying in the range  $[2^{B+1}, 2^B - 1]$  has a representation (4.3) where  $0 \leq s_{i_1 i_2 \dots i_n} < 2^{\beta_i}$  and the  $X_i (i=1, 2, \dots, n)$  are to be evaluated by  $X_i = 2^{\beta_i}$ .

### 4.3 Computation of Inner Products

Let  $N > 0$  and let  $\mathbf{a}$  and  $\mathbf{b}$  be sequences of integers in  $Z^N$ . Suppose that each component  $a^l$  and  $b^l$  of  $\mathbf{a}$  and  $\mathbf{b}$  can be represented by  $B_a$  and  $B_b$  bits, respectively, in addition to a sign bit (as in Theorem 3). We assume the inequalities  $B_a \leq \beta \prod_{i=1}^n (1 + d_i^a)$  and  $B_b \leq \beta \prod_{i=1}^n (1 + d_i^b)$ .

We shall give a method for computing the inner product

$$\mathbf{a} \circ \mathbf{b} = \sum_{l=1}^N a^l b^l \quad (4.6)$$

We use Theorem 3 to represent the components  $a^l$  and  $b^l$  of  $\mathbf{a}$  and  $\mathbf{b}$  as polynomials in the ring  $Z[X_1, X_2, \dots, X_n]$ . Let  $a^l$  have the representation

$$A^l(X_1, X_2, \dots, X_n) = \sum_{i_1=0}^{d_1^l} \sum_{i_2=0}^{d_2^l} \dots \sum_{i_n=0}^{d_n^l} a_{i_1 i_2 \dots i_n}^l X_1^{i_1} X_2^{i_2} \dots X_n^{i_n} \quad (4.7)$$

with  $-2\beta + 1 \leq a_{i_1 i_2 \dots i_n}^l \leq 2\beta - 1$ , and assume a similar representation for  $b^l$ :

$$B^l(X_1, X_2, \dots, X_n) = \sum_{i_1=0}^{d_1^b} \sum_{i_2=0}^{d_2^b} \dots \sum_{i_n=0}^{d_n^b} b_{i_1 i_2 \dots i_n}^l X_1^{i_1} X_2^{i_2} \dots X_n^{i_n} \quad (4.8)$$

Define the ring homomorphism  $\Psi: Z[X_1, X_2, \dots, X_n] \rightarrow Z$  by evaluation of the  $X_i$ :

$$\Psi(f(X_1, X_2, \dots, X_n)) = f(2^{\beta_1}, 2^{\beta_2}, \dots, 2^{\beta_n}).$$

Define the polynomial  $C(X_1, X_2, \dots, X_n)$  as the inner product

$$C(X_1, X_2, \dots, X_n) = \sum_{l=1}^N A^l(X_1, X_2, \dots, X_n) B^l(X_1, X_2, \dots, X_n). \quad (4.9)$$

Let  $c = \Psi(C(X_1, X_2, \dots, X_n))$ . Then, since  $\Psi$  is a ring homomorphism, the desired original inner product is given by

$$\begin{aligned} c &= \Psi(C(X_1, X_2, \dots, X_n)) = \Psi \left[ \sum_{l=1}^N A^l(X_1, X_2, \dots, X_n) B^l(X_1, X_2, \dots, X_n) \right] \\ &= \sum_{l=1}^N \Psi(A^l(X_1, X_2, \dots, X_n)) \Psi(B^l(X_1, X_2, \dots, X_n)) = \sum_{l=1}^N a^l b^l \end{aligned} \quad (4.10)$$

It thus suffices to compute  $C(X_1, X_2, \dots, X_n)$ . The remainder of this section will treat this computation.

#### 4.4 Isomorphisms and Homomorphisms; Representations of Integers

We shall now map the polynomials in  $Z[X_1, X_2, \dots, X_n]$  to a finite quotient ring. This ring will later be shown isomorphic to a direct-product ring. It is in the direct-product ring that the computations will ultimately be performed.

Let  $M$  be a positive integer. Assume each of its prime divisors  $p$  to satisfy the inequality  $p > d_i^a + d_i^b$ . Let  $\Phi_M: Z \rightarrow Z_M = Z/(M)$  be the ring homomorphism which maps an integer  $s$  to its residue class  $\langle s \rangle_M = s + (M)$ . We use the same symbol  $\Phi_M$  to denote the obvious extension of  $\Phi_M$  to the polynomial ring:

$$\Phi_M: Z[X_1, X_2, \dots, X_n] \rightarrow Z_M[X_1, X_2, \dots, X_n]. \quad (4.11)$$

Since the polynomials  $A^l$  and  $B^l$  are of degrees  $d_i^a$  and  $d_i^b$ , respectively, in the variable  $X_i$ , we choose polynomials  $g_i(X_i)$  of degree  $d_i = 1 + d_i^a + d_i^b$ , according to the hypotheses of Theorem 2:

$$g_i(X_i) = \prod_{j=1}^{d_i} (X_i - r_{ij}).$$

Here, for convenience, we choose consecutive integers for the  $r_{ij}$  by setting, when  $d_i$  is odd,  $r_{i1} = (1-d_i)/2$  and  $r_{i,j+1} = r_{ij} + 1$  for  $1 \leq j < d_i - 1$ . If  $d_i$  is even, we use the starting point  $r_{i1} = 1 - d_i/2$ .

Let  $G$  be the ideal in  $Z_M[X_1, X_2, \dots, X_n]$  generated by the  $g_i(X_i)$ :

$$G = (g_1(X_1), g_2(X_2), \dots, g_n(X_n)),$$

and let  $\Phi_G: Z_M[X_1, X_2, \dots, X_n] \rightarrow Z_M[X_1, X_2, \dots, X_n]/G$  be the quotient map:

$$\Phi_G(f(X_1, X_2, \dots, X_n)) = f(X_1, X_2, \dots, X_n) + G.$$

From (Eq 17) we obtain, since  $d_i = 1 + d_i^a + d_i^b$ ,

$$C = \sum_{l=1}^N A^l B^l = \sum_{k_1 < d_1} \sum_{k_2 < d_2} \dots \sum_{k_n < d_n} c_{k_1 k_2 \dots k_n} X_1^{k_1} X_2^{k_2} \dots X_n^{k_n}$$

where

$$c_{k_1 k_2 \dots k_n} = \sum_{l=1}^N \sum_{i_1+j_1=k_1} \sum_{i_2+j_2=k_2} \dots \sum_{i_n+j_n=k_n} a_{i_1 \dots i_n}^l b_{j_1 \dots j_n}^l \quad (4.12)$$

We consider the product

$$(\Phi_G \circ \Phi_M) \left( \sum_{l=1}^N A^l B^l \right) = \sum_{k_1 < d_1} \sum_{k_2 < d_2} \dots \sum_{k_n < d_n} \langle \gamma_{k_1 k_2 \dots k_n} \rangle_M X_1^{k_1} X_2^{k_2} \dots X_n^{k_n} + G$$

where  $\gamma_{k_1 k_2 \dots k_n}$  is chosen to lie in  $(-M/2, M/2]$  and to satisfy

$$\langle \gamma_{k_1 k_2 \dots k_n} \rangle_M = \sum_{l=1}^N \sum_{i_1+j_1=k_1} \sum_{i_2+j_2=k_2} \dots \sum_{i_n+j_n=k_n} \langle a_{i_1 \dots i_n}^l \rangle_M \langle b_{j_1 \dots j_n}^l \rangle_M$$

If the coefficients  $\gamma_{k_1 k_2 \dots k_n}$  and  $c_{k_1 k_2 \dots k_n}$  are equal, then computation of the inner product  $C = A^l B^l$  can be effected by doing the computation with coefficients lying in the finite ring  $Z_M$ . Moreover, by taking the quotient under the homomorphism  $\Phi_G$  the computation is further reduced to a ring of finitely many polynomials. It is this reduction that enables us to use theorem 2, so that the actual computation is carried out in a finite direct-product ring.

The following theorem gives a criterion for the size of the modulus  $M$  in order to ensure the above-mentioned equality of coefficients.

**Theorem 4.** Suppose that  $M$  satisfies the inequality

$$M > 2N(2^\beta - 1)^2 \prod_{i=1}^n [1 + \min(d_i^a, d_i^b)] \quad (4.13)$$

Then for all  $i = 1, 2, \dots, n$  and  $0 < k_i < d_i$  we have  $\gamma_{k_1 k_2 \dots k_n} = c_{k_1 k_2 \dots k_n}$ .

**Proof.** We have

$$\Phi_G(\Phi_M(C)) = \sum_{k_1 < d_1} \sum_{k_2 < d_2} \dots \sum_{k_n < d_n} \langle c_{k_1 k_2 \dots k_n} \rangle_M X_1^{k_1} X_2^{k_2} \dots X_n^{k_n} + G$$

On the other hand, since  $\Phi_M$  and  $\Phi_G$  are homomorphisms, we have

$$\Phi_G(\Phi_M(C)) = (\Phi_G \circ \Phi_M) \left( \sum_{l=1}^N A^l B^l \right) = \sum_{k_1 < d_1} \sum_{k_2 < d_2} \dots \sum_{k_n < d_n} \langle \gamma_{k_1 k_2 \dots k_n} \rangle_M X_1^{k_1} X_2^{k_2} \dots X_n^{k_n}.$$

Since the right hand sides of the two representations of  $\Phi_G \Phi_M(C)$  are equal, they lie in the same equivalence class of  $Z_M[X_1, X_2, \dots, X_n]/G$ . But the only polynomial in  $G$  which has degree  $< d_i$  in each  $X_i$  is the zero polynomial in  $Z_M[X_1, X_2, \dots, X_n]$ . Thus we conclude that the coefficients are equal:

$$\langle c_{k_1 k_2 \dots k_n} \rangle_M = \langle \gamma_{k_1 k_2 \dots k_n} \rangle_M \tag{4.14}$$

From (4.14) and Theorem 3 it follows that

$$|c_{k_1 k_2 \dots k_n}| \leq N(2^\beta - 1)^2 (1 + \min(d_1^a, d_1^b)) \dots (1 + \min(d_n^a, d_n^b))$$

Thus  $-M < c_{k_1 k_2 \dots k_n} - \gamma_{k_1 k_2 \dots k_n} < M$ , and hence by (4.13) we must have  $\gamma_{k_1 k_2 \dots k_n} = c_{k_1 k_2 \dots k_n}$ . This proves the theorem.  $\square$

Hence the inner product  $c = \sum_{l=1}^N a^l b^l$  can be computed by mapping the data  $a^l$  and  $b^l$  to the polynomial ring  $Z[X_1, X_2, \dots, X_n]$  and then continuing via  $\Phi_M$  and  $\Phi_G$ . In the ring  $Z_M[X_1, X_2, \dots, X_n]/G$  the inner product

$$\sum_{l=1}^N \sum_{i_1=0}^{d_1^a} \sum_{j_1=0}^{d_1^b} \dots \sum_{i_n=0}^{d_n^a} \sum_{j_n=0}^{d_n^b} \langle a_{i_1 \dots i_n}^l \rangle_M \langle b_{j_1 \dots j_n}^l \rangle_M X_1^{i_1+j_1} \dots X_n^{i_n+j_n}$$

is formed, and it is the representation of the solution in the ring  $Z_M[X_1, X_2, \dots, X_n]/G$ . To map the computation back to  $Z[X_1, \dots, X_n]$  we then find integers  $c_{k_1 k_2 \dots k_n}$  lying in the range  $-M/2 < c_{k_1 k_2 \dots k_n} < M/2$  which satisfy

$$c_{k_1 k_2 \dots k_n} = \sum_{l=1}^N \sum_{i_1+j_1=k_1} \sum_{i_2+j_2=k_2} \dots \sum_{i_n+j_n=k_n} \langle a_{i_1 \dots i_n}^l \rangle_M \langle b_{j_1 \dots j_n}^l \rangle_M$$

The final answer to the inner product is given by

$$c = \sum_{k_1 < d_1} \sum_{k_2 < d_2} \dots \sum_{k_n < d_n} c_{k_1 k_2 \dots k_n} 2^{\beta_1 k_1} 2^{\beta_2 k_2} \dots 2^{\beta_n k_n}$$

We will thus ensure a correct representation of  $c$  if the following conditions are met:

$$1) \quad M > 2N(2^\beta - 1)^2 \prod_{i=1}^n [1 + \min(d_i^a, d_i^b)] \quad (4.15)$$

$$2) \quad p \mid d_i > d_i^a + d_i^b \text{ for each } i = 1, 2, \dots, n \text{ and each prime divisor } p \text{ of } M. \quad (4.16)$$

#### 4.5 Representations of Complex Sequences

Suppose now that  $\mathbf{a}$  and  $\mathbf{b}$  are sequences of complex integers. We first represent the real and imaginary parts as polynomials in the ring  $Z_M[X_1, X_2, \dots, X_n]$ , using the same mapping as given in section 3. Thus we assume the same hypotheses on the real and imaginary parts as we did earlier on the components of the real sequences. Hence a typical sequence component, say  $a^l$ , will be represented as follows:

$$A^l = A_{Re}^l(X_1, X_2, \dots, X_n) + jA_{Im}^l(X_1, X_2, \dots, X_n).$$

We replace the complex unit  $j$  with the indeterminate  $Y$ . Then we have

$$A^l = A_{Re}^l(X_1, X_2, \dots, X_n) + YA_{Im}^l(X_1, X_2, \dots, X_n).$$

evaluated at  $Y = j$ . Choose  $M$  in such a way that all of its prime factors are of the form  $4k+1$ . We can then represent the complex numbers by mapping to the quotient ideal

$$Z_M[X_1, X_2, \dots, X_n, Y] \rightarrow Z_M[X_1, X_2, \dots, X_n, Y]/(Y^2+1).$$

Due to the choice of the prime factors of  $M$ , the polynomial  $Y^2+1$  factors in  $Z_M[Y]$ , say  $Y^2+1 = (Y-j_1)(Y-j_2)$ . It therefore factors in  $Z_M[X_1, X_2, \dots, X_n, Y]$ , and hence the quotient ring is (e.g. by Theorem 1) therefore isomorphic to the ring  $Z_M[X_1, X_2, \dots, X_n] \times Z_M[X_1, X_2, \dots, X_n]$  under the mapping  $Y \rightarrow (j_1, j_2)$ . This is the usual QRNS map, but in this case it includes extra indeterminates  $X_1, X_2, \dots, X_n$  (on which the QRNS isomorphism does not act).

By considering real and imaginary parts, it is easy to see that conditions (4.15) and (4.16) should be modified to

$$1') \quad M > 4N(2^\beta - 1)^2 \prod_{i=1}^n [1 + \min(d_i^a, d_i^b)] \quad (4.15')$$

$$2') \quad \text{For each prime divisor } p \text{ of } M, p \equiv 1 \pmod{4} \text{ and } p \nmid d_i > d_i^a + d_i^b \text{ for each } i = 1, 2, \dots, n. \quad (4.16')$$

#### 4.6 Direct Product Representations

We now use Theorem 2 to map the inner product computation to one in a direct-product ring. The isomorphism is given in Theorem 1. It would normally be assumed that the modulus  $M$  factors into smaller, relatively prime factors,  $M = m_k$ , so that the ring  $Z_M$  factors as well:

$Z_M \cong Z_{m_1} \times Z_{m_2} \times \dots \times Z_{m_K}$  Thus the inner product computation will take place in a direct-product ring whose only factors are the rings  $Z_{m_k}$  for  $1 \leq k \leq K$ .

The inverse isomorphism is computed by considering, in addition to the Chinese Remainder Theorem for the moduli  $m_k$ , the following polynomials ( $1 \leq i \leq n, 1 \leq j \leq d_i$ ):

$$g_{ij}(X_i) = \prod_{\substack{k=1 \\ k \neq j}}^{d_i} (X_i - r_{ik}).$$

Since the evaluated polynomial  $g_{ij}(r_{ij})$  is, by hypothesis, invertible, we can construct polynomials  $h_{ij}(X_i)$  which = 0 at  $X_i = r_{ik}$  for  $k \neq j$ , and = 1 if  $k = j$ . These polynomials will then yield the inverse isomorphism in the usual manner (see [12] for a discussion of such inverse isomorphisms).

## 5. Implementation Examples

### 5.1 A DSP Example Using a Single Indeterminate (Discrete Cosine Transform)

Let us give as an example the computation of the discrete cosine transform (DCT). The data is real and so a complex operator indeterminate is not required. We will also use only a single indeterminate in the mapping of the binary numbers representing the samples. The DCT is given in equation (5.1).

$$Y(l) = \rho(l) \sum_{m=0}^{N-1} y(m) \cos \left[ \frac{(2l+1)m}{2N} \right] \quad 0 \leq l \leq N-1 \quad (5.1)$$

where  $\rho(l) = 1/\sqrt{2}$  if  $l = 0$  and  $\rho(l) = 1$  otherwise. We choose  $M = 29 \cdot 31$  as modulus; these are the largest five-bit moduli that satisfy the conditions given in section 4.5.

Let  $y(m)$  be represented by the variable  $h(m)$  in the section 4.6, and let the cosine factors of eqn. (5.1), after conversion into integers, be represented by  $c(m)$ . We take  $d_c = d_h = 2$ , together with  $N = 12$  and  $\gamma = 3$ . We also use nonnegative residues throughout the calculations. Thus the radix  $2^\gamma = 8$  and the sequences  $c(m)$  and  $h(m)$  are represented as nine-bit integers ( $9 = \gamma(1 + d_c)$ ). The cosine terms (including the term  $\rho(l)$ ) must, of course, be converted to integers in the range  $[0, 512)$ .

Thus for each  $l = 0, \dots, N-1$ , we compute an inner product of the form

$$Y(l) = \rho(l) \sum_{m=0}^{N-1} \sum_{i=0}^2 \sum_{j=0}^2 c_i^l(m) h_j(m) X^{i+j} \quad (5.2)$$

where  $X$  represents the radix  $2^3 = 8$ . Note that the coefficients  $c_i^l(m)$  and  $h_j(m)$  are simply the three-bit components of the binary representations of the nine-bit integers  $c(m)$  and  $h(m)$ . Since these three-bit numbers are already their own residues mod 29 and mod 31, no further treatment is required to map them to the rings  $Z_{29}$  and  $Z_{31}$ .

We now consider the mapping to the direct product rings. Since  $d_c = d_h = 2$ , we have  $d = 5$ , and thus we require the polynomial  $g(X)$  to have degree five. The condition given in section 4.5 above is obviously satisfied if the roots of  $g$  are taken to be consecutive integers, and thus we can take the polynomial  $g(X) = X(X^2-1)(X^2-4)$ . The terms  $c_i^l(m)$  are mapped to five copies each of the rings  $Z_{29}$  and  $Z_{31}$  by the map

$$\sum_{i=0}^2 c_i^l(m) X^i \rightarrow \sum_{i=0}^2 c_i^l(m) s^i \quad (5.3)$$

where  $s$  takes on the values  $s = -2, -1, 0, 1, 2$ . These are the five roots of  $g(X)$ , and thus we are evaluating the polynomial at these individual roots, one evaluation for each copy of  $Z_{29}$  (resp.  $Z_{31}$ ).

The same map is applied to the polynomials  $\sum_{j=0}^2 h_j(m) X^j$ .

The inner product is then performed by summing over  $m$ . Note that in actual practice the cosine data,  $c(m)$ , are assumed to be transformed to the finite rings and pre-programmed in the individual rings; the input data  $h(m)$ , which is assumed to be integer valued to begin with, is transformed first by diverting the respective bits (in threes) to the coefficients  $h_j(m)$ , and then evaluating the polynomials, for each modulus 29 and 31, under the five evaluation maps  $X \rightarrow s$ , where  $-2 \leq s$

2. The factor  $\rho(l)$  is included in the cosine coefficient mapping strategy as will be indicated in the array implementation details.

The inner product is therefore calculated in each of the five copies of each of the rings  $Z_{29}$  and  $Z_{31}$ . These numbers, five each for each modulus 29 and 31, can be mapped back to the polynomial ring by the inverse of the mapping given in section 4.5. This computation proceeds as a  $5 \times 5$  matrix multiplication *inside the ring*  $Z_{29}$  (resp.  $Z_{31}$ ).

Let these ten results, the coefficients of one polynomial in  $Z_{29}[X]$  and another in  $Z_{31}[X]$ , be denoted by  $A_k(l)$  and  $B_k(l)$  respectively. These numbers are then processed via the Chinese Remainder Theorem to obtain the final answers, which will be of the form

$$Y(l) = \sum_{k=0}^4 w_k(l)X^k \quad 0 \leq l \leq N-1 \quad (5.4)$$

Here  $X = 2^3 = 8$  and, according to (5.2),  $w_k(l)$  must be given, for  $0 \leq k \leq 4$ , by

$$w_k(l) = \sum_{m=0}^{N-1} \sum_{i+j=k} c_i^l(m)h_j(m) \quad (5.5)$$

## 5.2 Scaling

The output  $Y(l)$ , as given by (5.4), will be correct insofar as the coefficients  $w_k(l)$  obtained from the Chinese Remainder Theorem agree with those given by (5.5). These coefficients are guaranteed to be correct provided that, for  $l = 0, \dots, N-1$ , the following (worst-case) bound holds:

$$\begin{aligned} 899 = M &> (1 + \min(d_c, d_h)) \max(h_j(m)) \sum_{i=0}^2 \sum_{m=0}^{N-1} c_i^l(m) \\ &= 3 \cdot 7 \sum_{i=0}^2 \sum_{m=0}^{N-1} c_i^l(m) \end{aligned}$$

We have found, by computer simulation, that modular overflow is extremely uncommon with  $N = 12$  (it only occurs for a certain class of functions that behave essentially as constants).

Of course, the output (5.4) will consist of integers which are, for most purposes, far too large, and which have to be scaled. Scaling can also help remove the complexities of conversion, particularly when the polynomial ring system is to be entered again for further computation. Some test runs have suggested the scaling factor  $2^{13}$ , which we shall use for this example. We write the  $w_k(l)$  in the form

$$w_k(l) = \alpha_k(l) + 32\beta_k(l) \quad 0 \leq \alpha_k(l) < 31 \quad (5.6)$$

To find  $\alpha_k(l)$  and  $\beta_k(l)$  we use the known residues  $A_k(l)$  and  $B_k(l)$  of  $w_k(l)$ , with respect to the moduli 31 and 29 respectively. Using the mixed radix version of the CRT we can obtain the following:

$$w_k(l) = f_k(l) + 31g_k(l) \quad 0 \leq f_k(l) < 30, 0 \leq g_k(l) < 28$$

from the formulae

$$f_k(l) = A_k(l) \quad g_k(l) = \langle 15(B_k(l) - A_k(l)) \rangle_{29}$$

From these numbers we obtain  $\alpha_k(l)$  and  $\beta_k(l)$  by means of exact division scaling [6]

$$\alpha_k(l) = \langle f_k(l) - g_k(l) \rangle_{32} \quad \beta_k(l) = \begin{cases} g_k(l) & \text{if } f_k(l) \geq g_k(l) \\ g_k(l) - 1 & \text{if } f_k(l) < g_k(l) \end{cases}$$

With these numbers known we can now construct  $w_k(l)$  using (5.5). We dispense with  $w_0(l)$  and  $w_1(l)$  completely, since the maximum value of  $(w_0(l) + 8w_1(l))/2^{13}$  can be shown to be .868, with an average value of .434 (we assume here that the values of the coefficients  $\alpha_k(l)$  and  $\beta_k(l)$  are essentially random for small  $k$ ).

In addition to this expected loss of .434 we also have to consider the addition of .5 required to convert rounding into truncation. These two omissions can be taken care of statistically by setting the middle bit of  $\alpha_4(l)$  and the high bit of  $\alpha_2(l)$  equal to 1.

We are now left with the task of forming the integer which is the truncated expression:

$$\sum_{k=0}^2 w_{k+2}(l)8^k = \sum_{k=0}^2 (\alpha_{k+2}(l) + 32\beta_{k+2}(l))8^k$$

Since  $\alpha_{k+2}(l) + 32\beta_{k+2}(l)$  is at most a ten-bit integer, this computation can be achieved by the use of ordinary binary adders.

It is worth mentioning that overflow errors are most likely to occur for the coefficient  $c_2$ , with  $c_1$  and  $c_3$  sharing a much smaller percentage of the errors, and  $c_0$  and  $c_4$  never exhibiting an error. This is because the terms that have the greatest contribution are the ones in which  $i+j = k$  has the greatest number of solutions. As a result of this we find that *if* there is an overflow error, then it will most likely occur in  $c_2$ , where it will not affect the output by more than two or three percent in terms of relative error. This is an important point, since in a standard residue system, any error in calculating a residue has disastrous consequences, with the error appearing as a multiple of the product of the other residues (over the direct product ring). In the case of the polynomial ring strategy, by scaling we have moved the problem of overflow from the centre of the dynamic range to the least significant part of the dynamic range. The minor consequences of overflow can now be reflected in a more aggressive dynamic range policy, rather than using pessimistic dynamic range bounds to ensure avoidance of overflow conditions.

### 5.3 Using Multiple Indeterminates

Suppose that **a** and **b** are real integer sequences of length N whose components can be represented by 8 bits plus a sign bit. Assuming four-bit modulus components  $m_k \leq 15$ , we can say the

following about the maximum blocklength  $N$  that would be permitted by the method discussed in this chapter:

Assume  $n = 3$ ,  $d_i^a = d_i^b = 1$  and  $d_i = 3$  for  $i = 1, 2, 3$ , and  $\beta = 1$ . Then each  $a^l$  is expanded as a sum

$$a^l = \sum_{i,j,k=0}^1 a_{ijk}^l 2^{4i+2j+k} \quad (5.7)$$

with  $a_{ijk}^l = 0$  or  $1$ ; a similar formula holds for  $b^l$ . Thus the coefficients of the polynomials representing  $a^l$  and  $b^l$  are simply the digits of their binary expansions.

Let indeterminates,  $X, Y$  and  $Z$ , represent  $2^4, 2^2$ , and  $2^1$  in eqn, (5.7). We can take the polynomials  $g_1, g_2$ , and  $g_3$  of the variables  $X, Y$ , and  $Z$  to be the same:  $g_i(X) = X(X^2-1)$  for  $i = 1, 2, 3$ , so that  $g_i$  has roots at  $0$  and  $\pm 1$ . The polynomials  $h_{ij}$  of section 5, which give the polynomial portions of the inverse of the isomorphism, are easily computed.

Encoding thus consists of multiplying the binary digits of the components of the sequences by  $0$  or  $\pm 1$ . The only possible values, since the coefficients themselves are of the same form, are  $0$  and  $\pm 1$ . Thus no reduction modulo  $m_k$  is needed, and the data entries to the components of the direct-product ring can be read off from the binary digits of the input data.

The condition (4.16) coupled with the condition that  $m_k \leq 15$ , yields the inequality  $3 \leq m_k \leq 15$ .

To maximize the product  $M$ , we see that we should take the  $m_k$  from the set  $\{13, 11, 9, 7, 5\}$ .

Since  $d_1 = d_2 = d_3 = 3$ , the ring in which the computation is to be carried out will consist of  $27 = 3^3$  copies of  $Z_M$ , or what is equivalent,  $Z_{13} \times Z_{11} \times Z_9 \times Z_7 \times Z_5$ ; to maximize  $N$  we choose  $M = 13 \times 11 \times 9 \times 7 \times 5$ , and thus condition (4.15) reduces to  $13 \times 11 \times 9 \times 7 \times 5 > 16N$ , or  $N \leq 2815$ , and the computation will be carried out in 27 copies of the ring  $Z_{13} \times Z_{11} \times Z_9 \times Z_7 \times Z_5$ .

Obviously a choice of fewer of the  $\{m_k\}$  would result in a simpler ring structure as well as a smaller upper bound for the values of  $N$ .

Note that the decoding of the computational answer  $c$  consists of polynomial maps, which can be performed via the small moduli 5,7,9,11 and 13, followed by the Chinese Remainder Theorem for 5,7,9,11 and 13, which is not applied until the final step. Note too, that if scaling is required, then a simplification is afforded by the observation that many of the polynomial terms will map to small integers which will be considered insignificant. Such terms can therefore be ignored in the decoding process.

The multiple indeterminate procedure can lead to some very interesting results. The next section outlines the use of 3-bit moduli, 3, 5, and 7, to compute an FFT butterfly over a greater than 32-bit dynamic range. Conventional QRNS implementations require 6 moduli of 5- and 6-bit size. We will see, in section 6, that the VLSI implementation implications of 6-bit moduli require special pipelined structures for fixed multiplication, whereas 3-bit moduli implementations allow general multiplication with a single complex pipelined cell.

## 5.4 FFT Butterfly Computation

### 5.4.1. *The Mapping Strategy*

We write the integers representing the real and imaginary parts of the data, together with the coefficients of the FFT, as polynomials in the variables  $W, X, Y$  and  $Z$ , where  $W = 2, X = 4, Y = 16$ , and  $Z = 256$ . With this notation, any positive integer  $< 2^{16}$  can be written in a unique fashion as a sum:

$$\sum_{i_1, i_2, i_3, i_4 \in \{0,1\}} a_{i_1 i_2 i_3 i_4} W^{i_1} X^{i_2} Y^{i_3} Z^{i_4} \quad (5.8)$$

with the coefficients equal to 0 or 1. Similarly, any negative integer  $> -2^{16}$  can be written in the same form with coefficients 0 or  $-1$  (note that the use of 0 and  $\pm 1$  implies a signed bit

representation of the coefficients). To obtain representations for complex integers we should like to use the QRNS method, but we cannot inasmuch as the moduli 3 and 7 do not support a complex unit. To avoid this obstacle and yet still preserve a channel-independent mode of multiplication, we use an additional indeterminate, which we call  $T$ , to represent the complex unit  $j$ . The indeterminate  $T$  cannot satisfy the polynomial equation  $T^2 + 1 = 0$  (because 3 and 7 do not support roots of  $-1$ ). Instead, we use the polynomial  $T(T^2 - 1) = 0$  to define the mapping. This polynomial always has three roots in any finite ring  $Z_m$ , provided  $m > 2$ ; the penalty we pay for this modification is a 50% increase in the number of rings in the direct product; the advantage is that these rings are very small. Since each of the 'bit indeterminates' also form 1st order polynomials, we may use the same 3 root polynomial to form the direct product mapping. The amazing feature about this mapping is that the complex operator and the bit operators are interchangeable, allowing a variety of binary representations of complex numbers to be simply mapped to the direct product ring. This map is performed by evaluating each of the five variables  $W, X, Y, Z$  and  $T$  at each of the three roots 0, +1 and  $-1$ . This results in  $3^5 = 243$  results for each of the moduli 3, 5, and 7. Observe that the map is very simple, consisting of nothing more difficult than sign changes and additions.

As an illustration, Figure 5.1 depicts the forward mapping of an 8-bit integer map to three indeterminates:  $W, X, Y$ , producing 27 elements for each bit. The mapping elements for bit-5 are shown explicitly. Each mapping layer corresponds to a separate bit; the monomials corresponding to that bit position are shown alongside the map layer. Note that the mapping will be performed for each of the three moduli: 3, 5, 7. In terms of complex numbers we may map the  $Y$  indeterminate to  $j$  and treat the 8-bit number as a concatenation of a 4-bit real and a 4-bit imaginary Gaussian number.

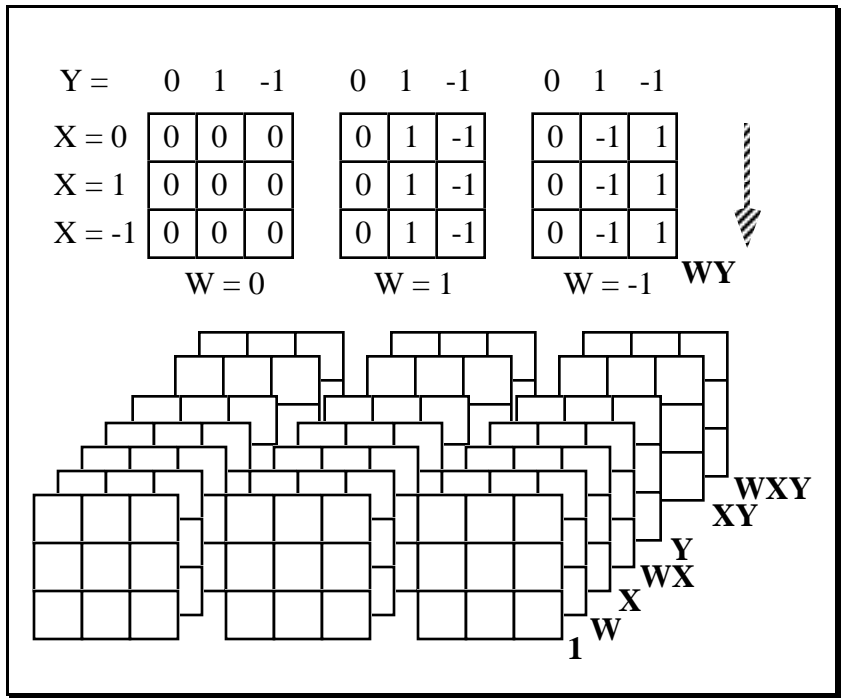


Fig. 5.1 Mapping for an 8-bit system

If we decide to map  $X$  to  $j$ , then the 8-bit sequence represents a 'digit division multiplex' type of decimation, between the real and imaginary parts. If we wish to code the sequence as an 8-bit real number, then each of the indeterminates will represent a power of 2. Only at the inverse map, when the indeterminates are replaced by the quantity they represent,

will their relevance become clear. If the forward map is formally treated as a map on three coefficients, namely the coefficients of  $1, W,$  and  $W^2$ , even though the coefficient of  $W^2$  is always zero, we can use this formalism to invert the above map; this inverse is performed after the computation of the required algorithm (in this paper a radix-4 DFT computational element) has been performed independently in each of the direct product rings. In a hardware implementation we need only consider the two coefficient forward map; the inverse map will, in general, use three coefficients.

We now set  $T = j$ , so that  $T^2 = -1$ . (This is allowed at this time since the complex multiplications have already been performed. Moreover, the isomorphism involving  $T$  has been accomplished both forward and backward.) This blends two of the three streams into one, each stream being the coefficients of the real and imaginary polynomials which represent the final result as polynomials over the rings  $Z_3, Z_5,$  and  $Z_7$ .

Weight	Equivalences			
$2^1$	$W$			
$2^2$	$W^2$	$X$		
$2^3$	$W^2X$	$X^2$	$Y$	
$2^4$	$W^2Y$	$XY$		
$2^5$	$W^2Z$	$XZ$	$W^2Y^2$	$XY^2$
$2^6$	$W^2XY$	$X^2Y$	$Y^2$	$Z$
$2^7$	$W^2XZ$	$X^2Z$	$YZ$	

Table 5.1

By using the CRT, and a combined scaling algorithm, we can finally combine these coefficients to give coefficients in the ring  $Z_{105}$ , the input wordlengths having been selected in such a way that modular overflow is either not possible or has very low probability. The scaling and conversion algorithm is presented in the next section.

#### 5.4.2. Scaling and Decoding

Each coefficient in the inverse mapped polynomial represents a weighting by a specific power of 2. Table 5.1 shows the monomial equivalences for the first seven powers of 2 weightings. The representation is redundant

since each of the coefficients is in the range  $[-52,52]$  while the weightings are in ascending powers of 2. Conversion is performed by summing coefficients that have the same power of 2; it turns out that these additions do not cause overflow (a proof of which will be presented in a later publication) and so the additions can be carried out over the rings, simply extending the inverse mapping computational array.

We now have polynomials in powers of 2 which have coefficients given by their residues modulo 3, 5, and 7, respectively. These coefficients are then decoded by means of the CRT (mixed radix conversion is preferred in our implementation), and will all lie in the interval  $[-52, 52]$ . Some rare exceptions to this are allowed as discussed in the error analysis below.

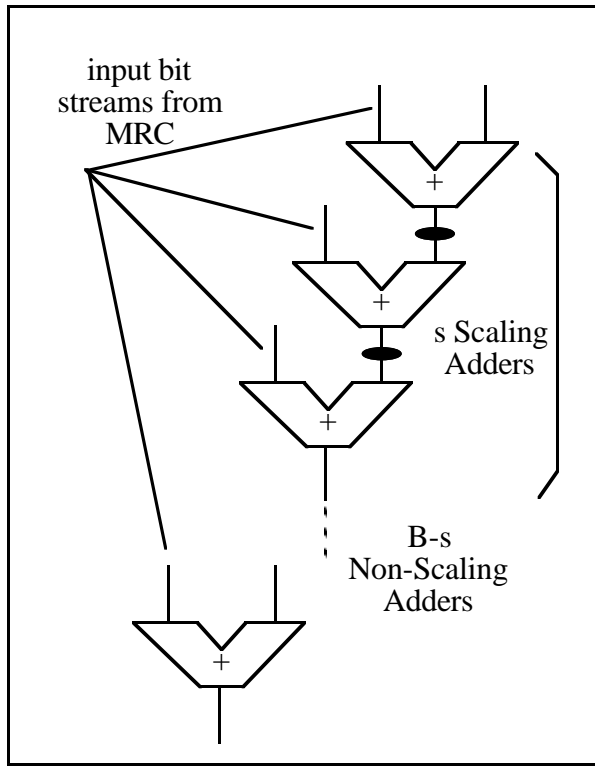


Fig. 5.2 Scaling and Binary Conversion

recovery array is more clearly evident in Fig. 5.2. The total number of bits of the full dynamic range conversion is  $B$ , the number of scaling bits is  $s$ . The  $\bullet$  blocks represent least significant bit removal (divide by 2).

A very important point is that we are doing most of the work in linear small ring pipelines with the final output generated by standard binary adders. The only RNS type of structure we require is the 3 ring converter to map each coefficient to a mod 105 ring. The 3 rings total only 8-bits, and the conversion can be performed with a single 8 input circuit. This circuit can also provide mapping to any weighted magnitude protocol. For example, a redundant mapping protocol could be implemented if redundant addition is desired for the conversion process. Our new technique of applying switching trees to a dynamic CMOS implementation yields efficient implementations of such small input bit circuits.

5.4.3. Experimental Error Analysis

We now perform the scaling by using a factor  $2^s$ . A low error method of applying this scale factor is to use the recursive relationship:

$$\tilde{C}_i = \frac{\tilde{C}_{i-1}}{2^\gamma} + C_i;$$

$$\gamma = \begin{cases} 1 & \text{for } 0 \leq i < s \\ 0 & \text{for } i \geq s \end{cases} \quad (5.9)$$

Using this method, the coefficients corresponding to the first  $s$  powers of 2 are processed using 5-bit additions. The error is

limited to  $\sum_{i=1}^s 2^{-i} = \frac{1 - 2^{-s}}{1 - 2^{-1}} - 1 < 1$ . The

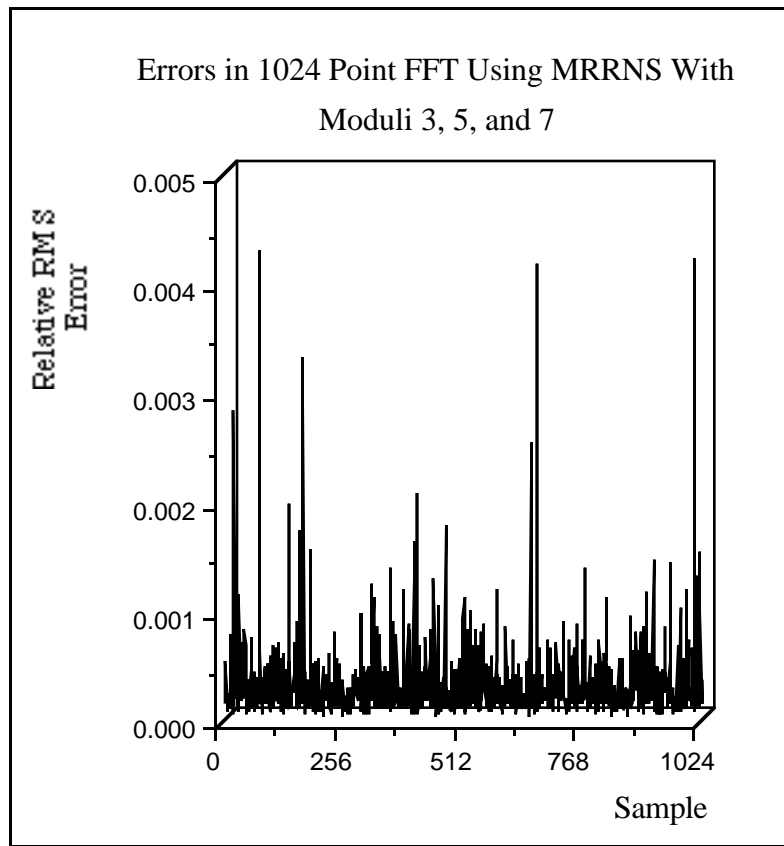


Fig. 5.3 Results from error analysis

are given in Fig. 5.3. The average relative root mean square error (RRMS error) was  $1.96 \times 10^{-4}$ .

A comparison was made against a QRNS system using moduli 61, 53, 41, 37, and 29 with twiddle factors quantized to 14-bits. This comparison yielded a  $1.08 \times 10^{-3}$  RRMS error for the QRNS system versus a  $1.96 \times 10^{-4}$  RRMS error for the new technique. This demonstrates the ability of a very small ring system to offer significant dynamic range. The hardware difference is considerable. The QRNS system has much smaller redundancy, but the lack of an efficient general multiplication structure [16] (compared to fixed multiplication) and the very large overhead, and awkward structure, associated with converting 6-bit modulus systems [3] make the new technique much more attractive.

For the purpose of measuring scaling error we have simulated the polynomial ring mapping technique with input data consisting of random complex integers. We assumed complex integers whose real and imaginary parts consist each of 14 bit random integers (plus an additional sign bit). For the twiddle factors we used approximations of 15 bits. The distribution of relative errors was measured in the sense of relative root mean square, and the results

## 6. VLSI Implementation of Pipelined Residue Computations

The thrust of digital signal processing (DSP) algorithms into the main stream of general signal processing has been due entirely to the advances in integrated circuit fabrication. With the advent of VLSI, just over a decade ago, DSP has witnessed a large increase in the number of applications of its theory and practice. VLSI implementations naturally adopted standard digital logic components that were developed for the 'binary world'. Such hardware includes ROM's, Adders, Multipliers, Microprocessors and special DSP processors. This is a perfectly reasonable approach when the custom design, and fabrication, of VLSI hardware is not practical, particularly for small production runs. Now, however, it is possible to both design and fabricate small production runs using the many 'silicon foundries' that have appeared over the last several years. Many are oriented to 'standard cell' implementations, but it is possible to produce small runs of full custom circuits. With the advent of ULSI (ultra large scale integration ..  $<1\mu\text{m}$  line widths) complete DSP systems will be available on a single chip.

In this section we will concentrate on the implementation of special finite ring high speed computational cells and architectures that are especially suited to custom VLSI implementations.

### 6.1. Modular Arithmetic Elements

Although ROM's play a central role in finite ring computational architectures [6][1], the use of direct boolean logic implementation, for modular addition and multiplication, is important for many implementations. We will start with an assumption that we are only interested in very high speed implementations. We are naturally lead to the use of bit-level systolic arrays [9] for the architecture of RNS arithmetic elements. These provide the minimum critical path for the pipelined cells and the local communication between adjacent cells that is sought after in high speed VLSI implementations. We will look at a recently published result that shows the efficiency of finite ring computations as a direct replacement for binary arithmetic [7].

We can identify 3 different generic techniques for implementing finite ring computations. All three are dependent upon the construction of switching tables rather than minimized logic gates. The reason for this is that modular computations do not exhibit the minimization patterns that we expect to see with conventional binary arithmetic elements (e.g. binary full adder).

The three techniques are identified in Fig. 6.1. Technique a) uses operations with constants to provide tables with small address spaces. The number of bits required to represent an element in the ring provides the address space  $2^B$ . DSP algorithms provide many opportunities to implement arithmetic operations between variables and constants (the basis set of a DSP transform can be used as a set of constants for example). Technique b) provides a method of generating an inner product step processor, at the bit-level, over a finite ring [16]. This is the least obvious of the three techniques and will be detailed in the next section. This technique also provides an address space that is quite modest ( $2^{B+1}$ ), for each table. The final technique, in c), provides general two variable arithmetic operations, with a consequently expensive address space of  $2^{2B}$ . The polynomial ring technique outlined in this chapter is ideally suited to the third technique, since very small moduli rings can be replicated to provide reasonable dynamic range. This technique is quite straight forward, and we will see, in a later section, that 3-bit moduli operators can be implemented in efficient dynamic logic CMOS circuitry.

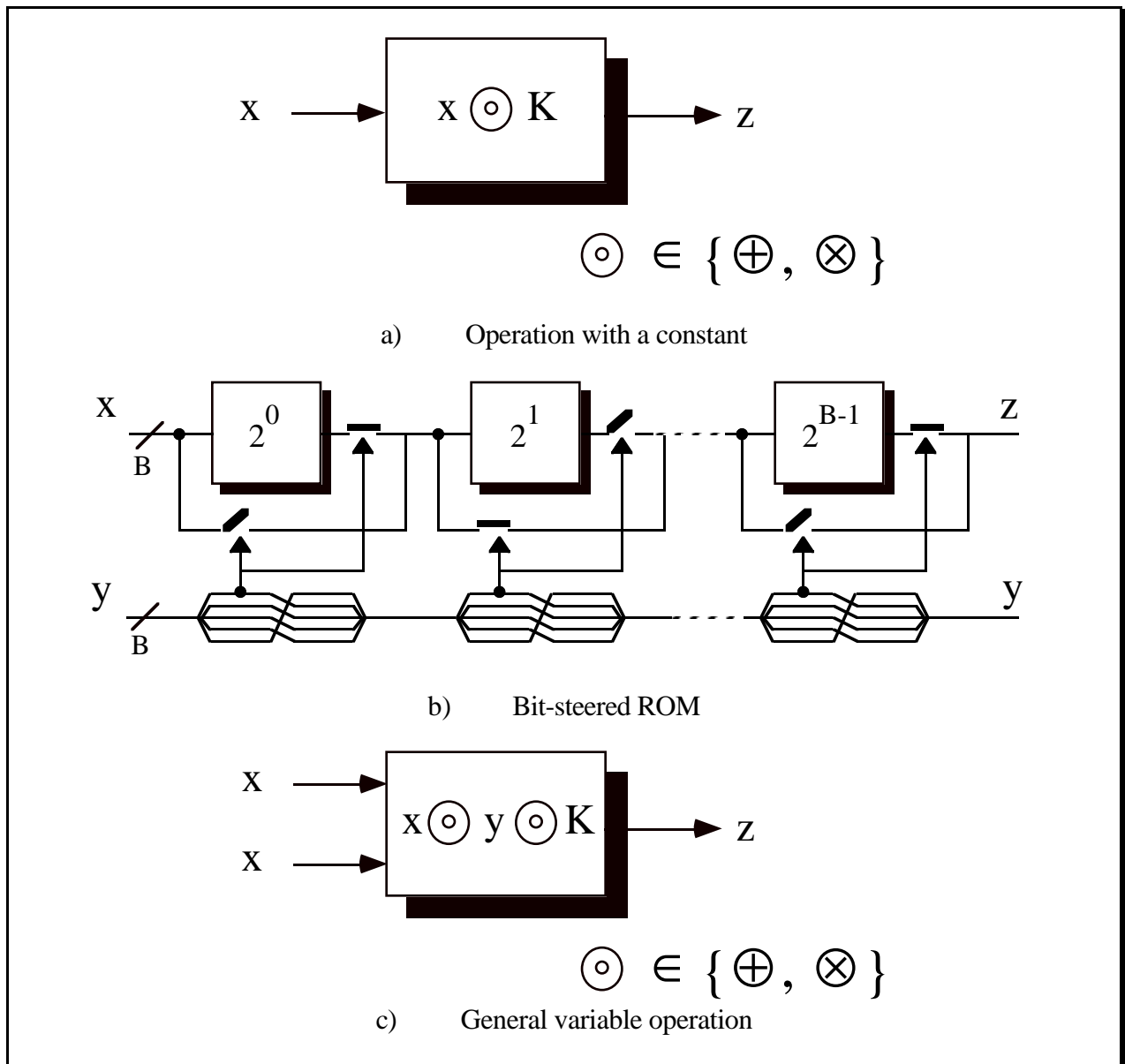


Fig. 6.1 Three finite ring computational techniques

## 6.2 Bit Steered ROM Structures

A pivotal element in the processing of most linear DSP algorithms is the Inner Product Step Processor (IPSP). This element is, in essence, a multiplier/accumulator cell. In this section we will discuss a recent development of a finite ring bit-level systolic cell realization of the IPSP that allows most RNS implementations of linear DSP algorithms to be constructed with large linear systolic chains of the cell [16]. This leads us to a true generic cell implementation of DSP algorithms.

### 6.2.1. The Finite Ring IPSP

Many matrix operations and DSP algorithms can be implemented using repeated multiply and add operations in a loop. The operation is performed using the IPSP. This processing cell computes:

$$Y_{\text{out}} = Y_{\text{in}} + (a \cdot X_{\text{in}}) \quad (6.1)$$

where  $Y$  is a running accumulator,  $A$  is a multiplier and  $X$  is the independent input data. By building a chain of such devices with an initial  $Y_{\text{in}} = 0$ , we are able to compute the inner product

$$Y = \mathbf{A} \cdot \mathbf{X}^T \quad (6.2)$$

We assume that the  $a$  in each IPSP changes accordingly. If we allow the restriction that the elements of vector  $A$  are fixed, then we can build a finite ring IPSP that has a complexity identical to simple addition, and a hardware cost function much lower than that of a binary implementation for many practical DSP applications.

The finite ring IPSP, we will give it the symbol  $\text{IPSP}_m$ , provides the relationship of equation (6.3):

$$Y_{\text{out}} = Y_{\text{in}} \oplus_m [a \otimes_m X_{\text{in}}] \quad (6.3)$$

All the inputs and outputs are  $B$  bit ring elements  $Y, a, X \in \mathfrak{R}(m)$  with  $B = \lceil \log_2 m \rceil$ . We can now breakdown the representation of the ring element of  $X$  into a binary form and generate the bit-level equivalent for the  $\text{IPSP}_m$ ; we will use the symbol  $\text{BIPSP}_m$ .

The operation of the  $\text{BIPSP}_m$ , with a fixed multiplier, can be defined by equation (6.4):

$$y_{i+1} = y_i \oplus [a \otimes x^{[i]} \otimes 2^i] \tag{6.4}$$

where  $i$  is the spatial array index,  $y_{i+1}, y_i, a \in \mathfrak{R}(m)$ , and  $x^{[i]}$  is the  $i$ th bit of  $X_{\text{in}} \in \mathfrak{R}(m)$ . Note that we have made the variables lower case to indicate operation at the bit level (note that both  $a$  and  $y$  are still assumed to be word variables within the ring). The ring operations are shown without the modulus subscript. The implementation of the  $\text{BIPSP}_m$  cell is shown in Fig.6.2.

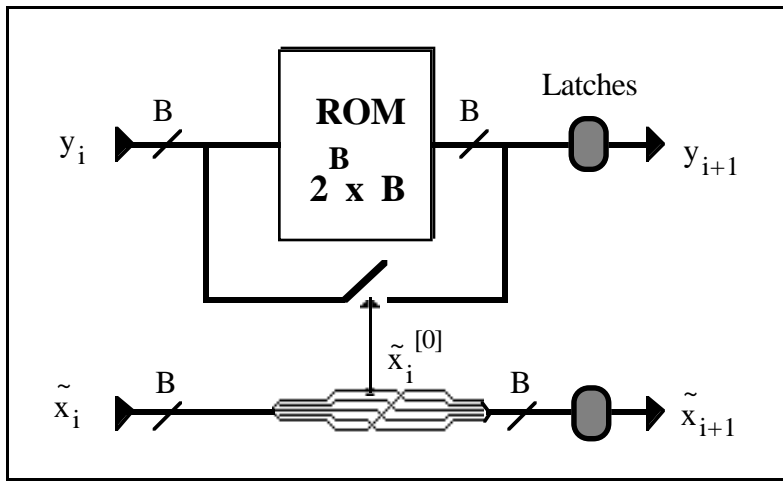


Fig 6.2 Implementation of the  $\text{BIPSP}_m$  cell

The cell contains a ROM of size  $B \cdot 2^B$  bits and a set of steering switches. Although we only need  $m$  words of storage, it is convenient to design a cell based on the largest value of  $m$ , namely  $2^B$ . Inputs to the cell are  $y_i$  and  $\tilde{x}_i$ , the outputs are  $y_{i+1}$  and  $\tilde{x}_{i+1}$ ; each output line is latched. The  $x$

input is given the new symbol,  $\tilde{x}$ , to indicate that 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 ROM stores the operation of  $y_i \oplus [2^i \otimes a]$ . The cell computes the relationship in equation (6.5):

$$\text{For } \tilde{x}_i^{[0]} = 1: \quad y_{i+1} = y_i \oplus [2^i \otimes a]$$

(6.5)

$$\text{For } \tilde{x}_i^{[0]}=0: \quad y_{i+1} = y_i$$

$\tilde{x}_i^{[0]}$  is known as the *steering bit*, since it is used to determine the direction the  $y$  data takes through the cell (either through the ROM or around it).

If we expand eqn (6.1) as:

$$Y_{\text{out}} = Y_{\text{in}} \oplus \sum_{j=0}^{B-1} \oplus_m \{ a \otimes x^{[j]} \otimes 2^j \} \quad (6.6)$$

then the output can be computed recursively using the recurrence:

$$y_0 = Y_{\text{in}}$$

$$y_{i+1} = y_i \oplus_m \{ 2^j \otimes_m x^{[j]} \} \quad (6.7)$$

$$Y_{\text{out}} = y_B$$

it can be seen that the operator  $\text{IPSP}_m$  is equivalent to a linear array containing  $B$  stages of  $\text{BIPSP}_m$  cells.

The addition of latches at the output of each cell allows adjacent cells to compute, *at the same time*, with stable input data. At the end of the computation period, the data is transferred to the latch (which is basically a single bit storage device), ready for the next computation period. This is known as *pipelining*, and in this case we have constructed a one-dimensional, or linear, pipeline. Pipelining is the essential ingredient of systolic arrays [8] along with local interconnection between processing cells. We do not have the space in this chapter to delve into systolic arrays, but it is clear from the 2 requirements just stated that an array of these  $\text{BIPSP}_m$  cells forms a linear systolic array.

The linear systolic array structure satisfies the requirements of modularity, homogeneity and local communication, sought after in VLSI designs. By repeatedly using the generic cell structure, all

closed computations can be performed with such parallel linear systolic structures, including interfacing with normal fixed point computations [16].

We will illustrate the use of the technique by taking an example of a linear pipelined FIR filter. An  $N^{\text{th}}$  order fixed coefficient FIR filter, computed over a finite ring, can be expressed as:

$$|Y(n)|_m = \sum_{i=0}^{N-1} {}_m (a_i \otimes_m X(n-i)) \quad (6.8)$$

We can now express the independent data,  $\{X\}$ , as a binary number within the ring, and thus expand eqn. (6.1) to the form of eqn.(6.2):

$$|Y(n)|_m = \sum_{i=0}^{N-1} {}_m \sum_{b=0}^{B-1} {}_m 2^b \otimes_m (a_i \otimes x^{[b]}(n-i)) \quad (6.9)$$

A single linear array of bit-steered ROM cells can be used to implement eqn (6.9), the addition being performed in a distributed manner as the partial results move along the array. The systolic cell required for this operation is similar to that shown in Fig. 6.2. An extra latch is provided on the  $\tilde{x}$  output. This single latch, forms a complete word shift after traversing  $B$  cells, because of the cyclic shift of data as each cell is traversed. This accomplishes the time shift indicated by the term,  $x^{[b]}(n-i)$ , in eqn.(6.9). It is interesting to note that, although the original FIR equation assumed word level operation, it is possible to have the individual bits embedded in the same structure. The use of single bit systolic arrays is not new and has been reported several times in the literature (for example [9]).

To visualize how individual groups of cells, forming  $\text{IPSP}_m$ s, can be used, along with a sliding latch, to form a convolution sum, let us take an example of a 2 coefficient FIR filter, where each coefficient is 3 bits long.

The word and bit level relationships are given in eqn. (6.10).

$$|Y(n)|_m = \sum_{i=0}^1 (a_i \otimes_m X(n-i)) = \sum_{i=0}^1 \sum_{b=0}^2 (a_i \otimes 2^b \otimes x(n-i)^{[b]}) \quad (6.10)$$

The filter coefficients  $a(0)$ ,  $a(1)$  are stored in a six cell array as shown in Fig. 6.3. Each cell is assumed to compute the operations defined by eqn. (6.10) and include latches on all of the outputs. The sliding latch on the steering bit line is explicitly shown as a filled square. The inputs are fed in sequentially (the  $y$  inputs are zero to initialize the accumulated products) and the partial results are also obtained sequentially. The inputs are assumed to be padded with zeros.

The upper partial results are the accumulating  $y$  values for each cell, and the lower results are the steering bits for each cell. The effect of the sliding latch can be seen by the extra delay on the steering bits after the first three cells. The effect is to delay the entire word (all three bits) by one clock period. The final accumulated result ( $y_6$ ) is the output. In order to verify that this linear systolic array produces the correct filtered output, Table 6.1 is constructed, showing the 6 cell outputs for 9 clock periods. In order to limit space in the table we temporarily drop the special symbol for ring multiplication, except at the output cell. We also assume that the array latches are initially filled with zeros. When a complete word has been assembled, this is written explicitly as a word (not as a bit-level description).

Table 6.2 shows the result of applying the word level filter description (left hand summation in eqn. (6.10)) for  $0 \leq n \leq 3$  over the summation index  $0 \leq i \leq 1$ . We see that the systolic array generates an identical set of outputs to the equation evaluation; the only difference between the two evaluations is the latency of 6 clock periods in the systolic array output.

It is to be pointed out that the efficiencies inherent in having fixed coefficient multipliers can also be used to advantage in binary arithmetic [10], but the resulting structure is not as regular, and cannot be used at such high throughput rates, as the systolic array approach discussed here.

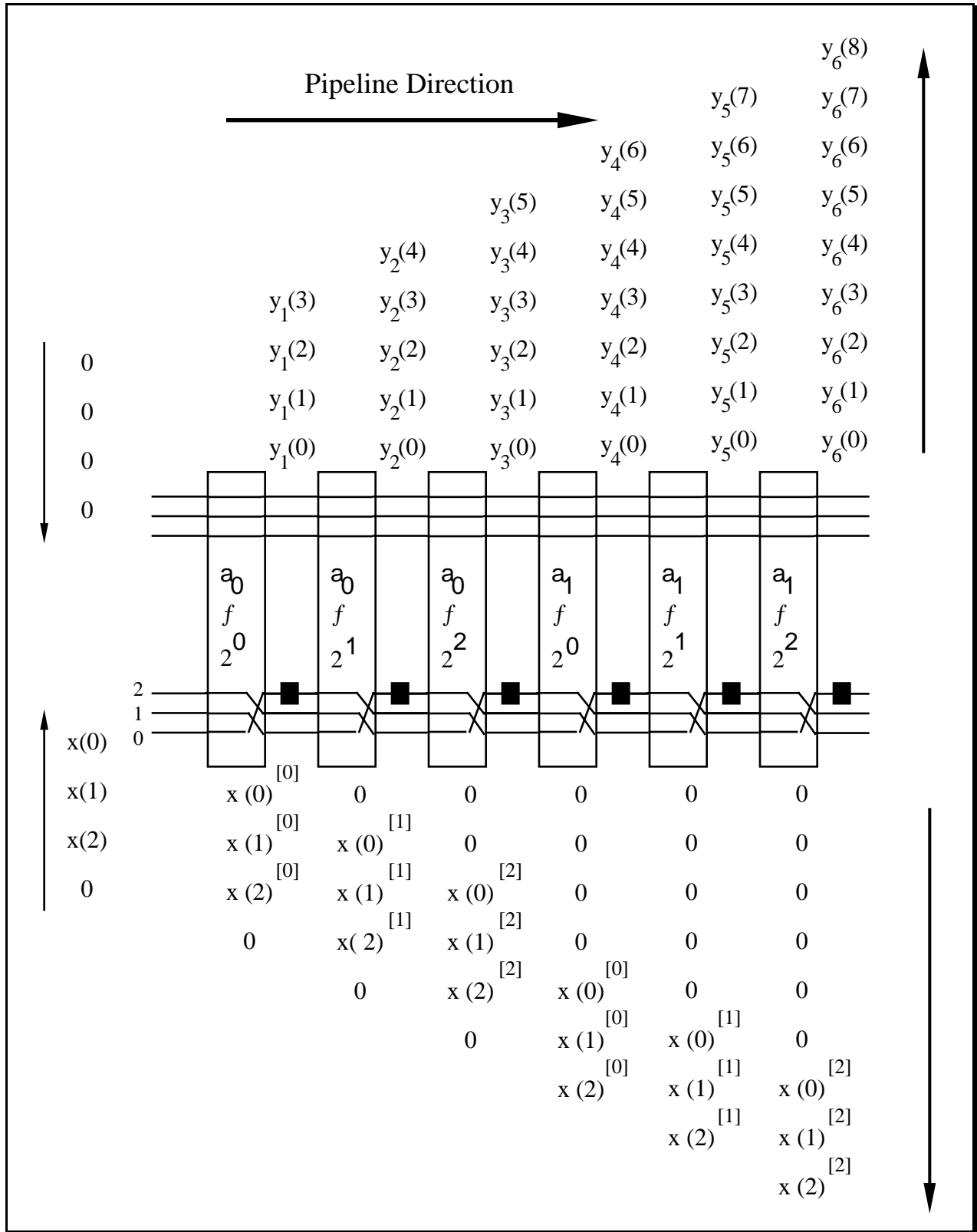


Fig 6.3 Data flow for the 2 coefficient linear systolic FIR filter

Table 6.1 Data output for the 2 coefficient systolic FIR filter

Clock Period ( $n$ )	$y_1(-n)$	$y_2(-n)$	$y_3(-n)$	$y_4(-n)$	$y_5(-n)$	$y_6(-n)$
0	$a_0 2^0 x(0)^{[0]}$	0	0	0	0	0
1	$a_0 2^0 x(1)^{[0]}$	$a_0 2^0 x(0)^{[0]}$ $\oplus$ $a_0 2^1 x(0)^{[1]}$	0	0	0	0
2	$a_0 2^0 x(2)^{[0]}$	$a_0 2^0 x(1)^{[0]}$ $\oplus$ $a_0 2^1 x(1)^{[1]}$	$a_0 x(0)$	0	0	0
3	0	$a_0 2^0 x(2)^{[0]}$ $\oplus$ $a_0 2^1 x(2)^{[1]}$	$a_0 x(1)$	$a_0 x(0)$	0	0
4	0	0	$a_0 x(2)$	$a_0 x(1) \oplus$ $a_1 2^0 x(0)^{[0]}$	$a_0 x(0)$	0
5	0	0	0	$a_0 x(2) \oplus$ $a_1 2^0 x(1)^{[0]}$	$a_0 x(1) \oplus$ $a_1 2^0 x(0)^{[0]}$ $\oplus$ $a_1 2^1 x(0)^{[1]}$	$a_0 \otimes x(0)$
6	0	0	0	$a_1 2^0 x(2)^{[0]}$	$a_0 x(2) \oplus$ $a_1 2^0 x(1)^{[0]}$ $\oplus$ $a_1 2^1 x(1)^{[1]}$	$a_0 \otimes x(1)$ $\oplus$ $a_1 \otimes x(0)$
7	0	0	0	0	$a_1 2^0 x(2)^{[0]}$ $\oplus$ $a_1 2^1 x(2)^{[1]}$	$a_0 \otimes x(2)$ $\oplus$ $a_1 \otimes x(1)$
8	0	0	0	0	0	$a_1 \otimes x(2)$

Table 6.2 Evaluation of the 2 coefficient FIR filter equation

n	0	1	2	3
$\sum_{i=0}^1 (a_i \otimes_m X(n-i))$	$a_0 \otimes x(0)$	$a_0 \otimes x(1) \oplus$ $a_1 \otimes x(0)$	$a_0 \otimes x(2) \oplus$ $a_1 \otimes x(1)$	$a_1 \otimes x(2)$

### 6.3 An Introduction to Switching Trees

Our preferred approach to the implementation of look-up table functions is to use switching trees implementing pipelined dynamic logic. The basic concept behind switching tree cells is to implement the switching function as a look-up table, but to construct the table as a minimized decision tree switching circuit. The minimization is based on the electrical characteristics required of the switching circuit, and does not involve either the usual Boolean algebra minimization or the concept of logic gate primitives. The switching blocks, unfortunately, do not have the benign decomposition properties of binary arithmetic, where it is possible to build complete arithmetic circuits from 3-bit input, 2-bit output full adders. The traditional approach for residue blocks has been to suggest the use of ROMs, and this still remains the preferred implementation procedure within the residue arithmetic community [15]. In an attempt to optimize current designs on silicon, our group has examined the construction of such ROMs and, in particular, the minimization of the ROM structure based on the specific ROM contents. In looking at ROM decomposition strategies, we can go beyond the normal 2-dimensional structure (row and column decoders) to a maximally decomposed structure consisting, essentially, of a binary tree. In this implementation the decoders reduce to inverters. The approach we have evolved is to generate a full binary tree, program the bottom of the tree (remove unwanted transistors) and then minimize the resulting structure based on three simple graph theory rules. In doing this we do not invoke any concepts from boolean algebra which may not yield the best transistor configuration (including PLA configurations).

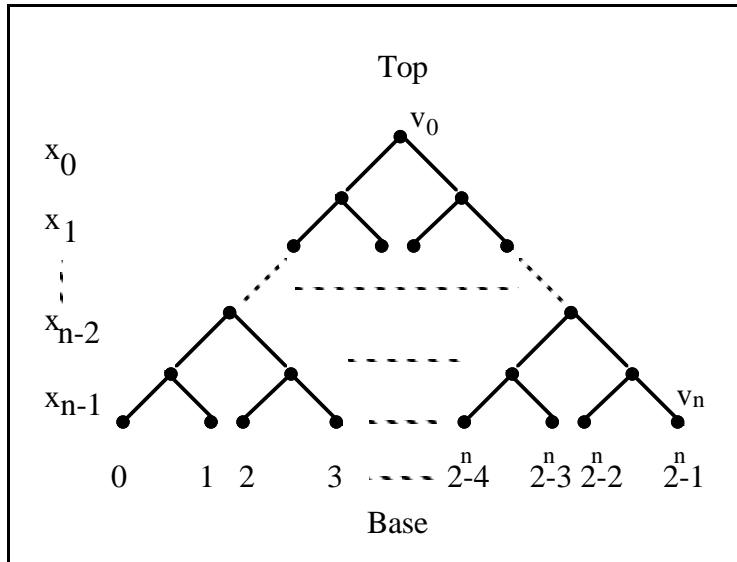


Fig 6.4 A Full Binary Tree

The use of binary, or decision trees, in logic minimization is obviously not new; Shannon used binary trees in relay logic in the late 1930s [13] and there were still refinements being published in the 60's. Since that time there have been many publications illustrating the use of binary trees to aid normal boolean decomposition techniques(e.g. [2]), and binary trees are regularly used for VLSI

architectures where each node contains a processing element [17]. Physically building binary trees as MOSFET logic blocks, and then minimizing the structure, does not appear to have been well explored in a dynamic CMOS setting.

A full binary tree possesses interesting qualities as far as a series chain discharge block in dynamic logic is concerned. In order to illustrate this, consider the full tree shown in Fig.6.4; although this is an uninteresting logic block (an all-true block) it demonstrates the circuit features of the tree. The tree directly corresponds to the transistor configuration of the logic block. Each node represents a net within the tree and each arc represents a transistor. Each level of the tree is controlled by the true and complement of a logic variable connected to the gates of the associated transistors. Our notation will assume that the arcs represented by the direction,  $\setminus$ , are transistors whose gates are driven by the true logic input. The other arc direction,  $/$ , represents transistors whose gates are driven by the complement of the logic input. In the full tree we see that, for stable logic inputs, only a single series path connects the top node to one of the bottom nodes, and that the capacitance at every node in each of the possible series paths is 3 source/drain capacitances in parallel. The gate load at the  $i$ 'th level of the tree is  $C_G 2^{i-1}$ , where the levels are numbered from 1 and a single

transistor gate load capacitance is  $C_G$ . The latter result is, perhaps, a disadvantage for the use of binary trees as logic blocks, but the former results are advantageous. Some of these qualities are modified when the tree is minimized.

Restricting the trees to be n-channel blocks and evaluating only a single node, we can build massively pipelined system, where every evaluation node is pipelined. The true single phase clocking system [18] recently introduced provides an excellent, stable, pipelining technique for quite complex trees. Using double trees evaluating both true and complement nodes, we can implement both domino and differential cascode voltage switch logic [3].

In this paper we will discuss basic minimization procedures of the tree itself, and give examples of trees for the small modulus multivariate polynomial ring mapping technique.

#### 6.4. Definitions and Rules

In order to present the three simple rules we use in our minimization procedure, the following definitions are used. We denote the tree as a graph,  $G=\{X, V, L\}$ , where  $X$  is the edge set (n-channel transistors) consisting of elements  $(x_{ij}, x'_{ij}) \Rightarrow (\setminus, /)$ , where  $i$  is the level and  $j \in [0, 2^i)$ ,  $V$  is the vertex set of nodes  $\{v_{ij}\}$ , and  $L$  is the link set (shorting links) of  $G$  consisting of elements  $L_{ij,kl}$  which represents a shorting link between a node  $v_{ij}$  and a node  $v_{kl}$ . In the full tree,  $L$  only contains links  $L_{nj,nl}, j,l \in [0, 2^n), j \neq l$ . The inputs to the tree are  $g_i \in \{0, 1\}$ ; if  $g_i = 1$ , then the path takes direction  $\setminus$  otherwise the path takes direction  $/$ .

A path,  $P_{ij,kl}$ , is a connection from a node  $v_{ij}$  to a node  $v_{kl}$ , constructed by edges and links. A full path connects node  $v_{00}$  to a node  $v_{nl}$ . The full paths will be divided into true paths, in which an edge  $x_{nj}$  or  $x'_{nj}$  is present, and a complement path in which an edge  $x_{nj}$  or  $x'_{nj}$  is removed. A truth table is mapped into a binary tree by removing a sub-set of edges  $\in \{x_{nj}, x'_{nj}\}, j \in [0, 2^n)$ , from a full tree based on the set of zeros in the truth table.

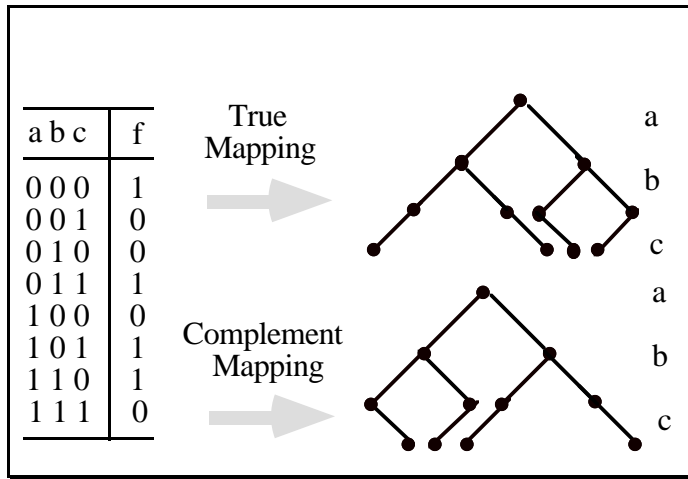


Fig. 6.5 Constructing the Tree

Fig. 6.5 illustrates the direct mapping of a truth table to both a true and complement tree. Note the correspondence between true and complement paths in the two mappings. We can also immediately identify common paths in both trees in order to minimize the generation of true and complement evaluation nodes. This can be very efficient for dynamic logic

that requires true and complement evaluation nodes e.g. domino and DCVS logic [Chu, 1986 #6]. Once the tree is obtained, it can be minimized; we refer to the minimized tree as a *switching tree*. The three main rules for removing edges (minimizing transistors) are given below, without proof but with appropriate examples.

Rule 1:

If a full sub-tree exists between node  $v_{ij}$  and all nodes  $v_{nl}$  where  $j \cdot 2^{n-i} \leq l < (j+1) \cdot 2^{n-i}$ , the edges can be replaced by a link  $L_{i,j,n} j 2^{n-i}$ .

Proof:

Since a full sub-tree connects  $v_{ij}$  to the ground plane, any set of inputs  $\{g_i, g_{i+1}, \dots, g_n\}$ ;  $g_j \in \{0, 1\}$  will provide a unique path to ground.

◆

An example is shown in Fig. 6.6.

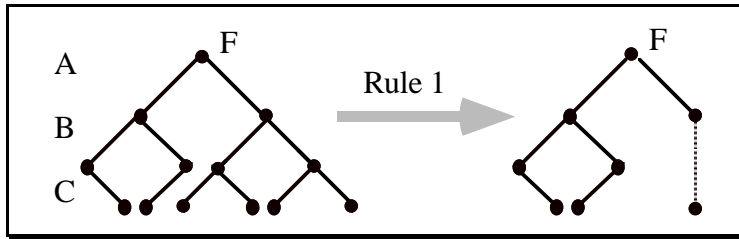


Fig.6.6 An example of Rule 1

Rule 2:

If paths from  $v_{ij}$  to  $v_{nl}$  and  $v_{ik}$  to  $v_{nm}$ ;  $j, k \in [0, 2^i)$ ,  $l, m \in [0, 2^n)$ , contain an identical sequence of edges, nodes at the same level in both sequences can be merged.

Proof:

Since the sequences are identical this implies that each sequence is driven from an identical input set:  $\{g_i, g_{i+1}, \dots, g_n\}$ ;  $g_j \in \{0, 1\}$ . Each input connects to an edge  $e_{mk} \in \{x_{mk}, x'_{mk}\}$ ,  $k \in [0, 2^m)$ . At level  $n$  let the two edges in the sequence be  $e_{ni}$ ,  $e_{nj}$ . Since the input,  $g_n$ , is common to both edges, both bottom nodes at level  $n-1$  will either be identically a path to ground or not a path to ground. Thus the two nodes at level  $n-1$  may be merged. If bottom nodes at level  $k$  are merged, then, by similar reasoning, the bottom nodes at level  $k-1$  may be merged.

By induction we see that all of the nodes at the same level in both sequences may be merged.

◆

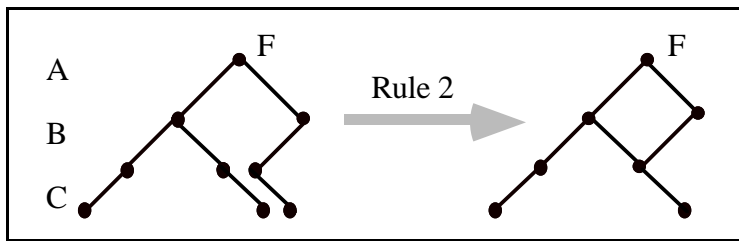


Fig. 6.7 An example of Rule 2

An example is shown in Fig.6.7 for  $i = n-1$ .

Rule 3:

Consider an  $r$ -edge path,  $P_1$ , connecting a node  $v_{ij}$  with a node  $v_{nl}$ , and an  $s$ -edge path  $P_2$ , connecting the node  $v_{ij}$  with a node  $v_{nm}$ , such that  $r < s$ . If the condition is satisfied that, except for the first edge, the first  $r$  edges are identical in both paths then the first edge in  $P_2$  can be replaced by a link.

Proof:

Since  $r < s$  the connection between the node at level  $i+r$  in  $P_1$  and the ground plane is a sequence of  $s-r$  links; thus the node at the  $(i+r)$ 'th level in  $P_1$  is connected to the ground plane. The input conditions that cause each path to conduct are, therefore,  $\{g_{ij_i}, g_{i+1j_{i+1}}, \dots, g_{i+rj_{i+r}}\}$  for  $P_1$  and  $\{g'_{ij_i}, g_{i+1j_{i+1}}, \dots, g_{i+rj_{i+r}}, \dots, g_{nj_n}\}$  for  $P_2$ , where  $g'$  is the logic complement of  $g$  and the values of  $g$  or  $g'$  in the sequences are such that the transistor turns on. Given a set of inputs  $\{g_{i+1j_{i+1}}, \dots, g_{i+rj_{i+r}}\}$  which cause a conducting path between nodes at level  $i+1$  and  $i+r$  for both paths  $P_1$  and  $P_2$ , then conduction between node  $v_{ij}$  and ground occurs for either condition:  $\{g_{ij_i}\}$  for  $P_1$  or  $\{g'_{ij_i} g_{i+r+1j_{i+r+1}}, \dots, g_{nj_n}\}$  for  $P_2$ . Conduction between node  $v_{ij}$  and ground is therefore dependent on either  $g_{ij_i}$  or  $\{g_{i+r+1j_{i+r+1}}, \dots, g_{nj_n}\}$ . Since the conduction is independent of  $g'_{ij_i}$  the edge can be replaced by a link.

◆

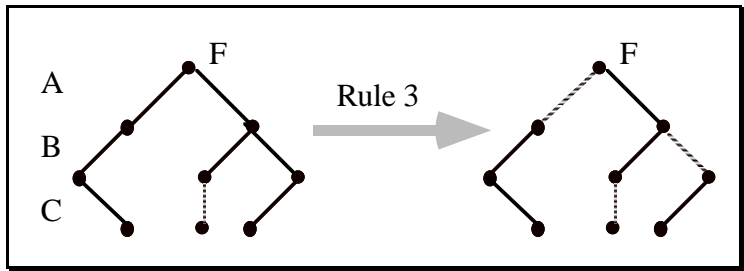


Fig. 6.8 An example of Rule 3

An example for Rule 3 is shown in Fig. 6.8. Note that the rule has been applied twice with  $r=1$  and  $r=2$  respectively.

A complete minimization strategy may be based on these three simple graphical rules. The complete procedure from truth table to switching tree is illustrated by the full-adder example in Fig. 6.9. The results obtained are not new [Chu, 1986 #6]; this is to be expected for an example which only uses 3 inputs, since most minimization techniques will converge to this solution. For larger trees, we obtain new results.

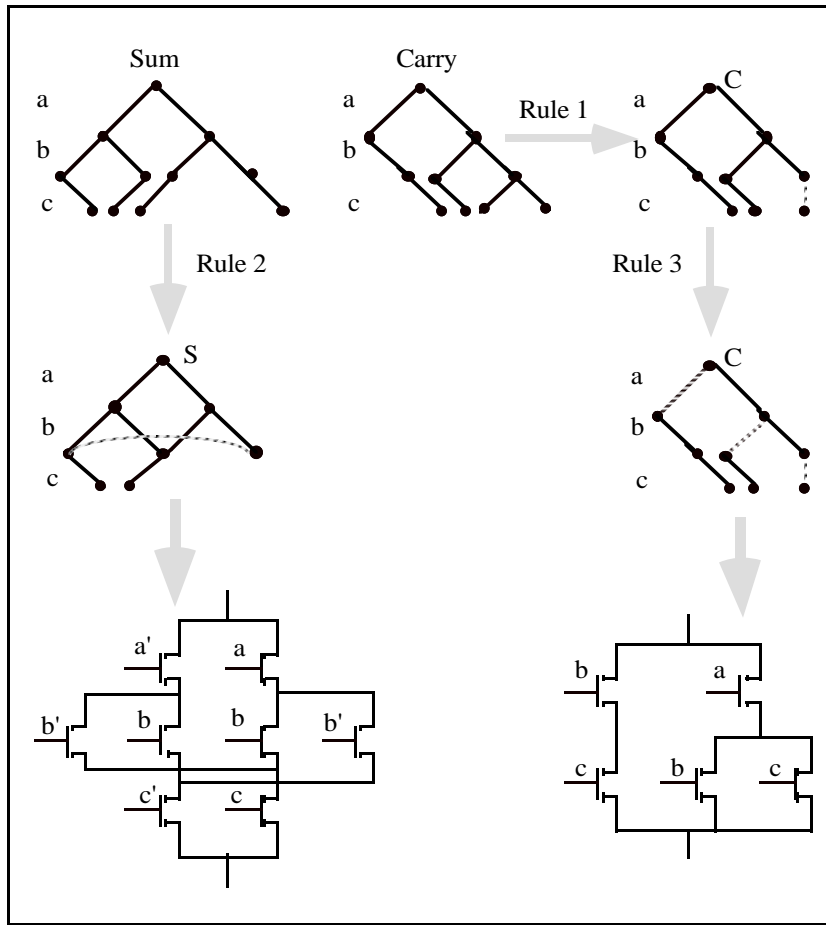


Fig. 6.9 A full-adder example to illustrate the entire procedure

In addition to the application of these three rules, we can also include the re-ordering of layers and allocation of don't care states to true or complement paths. We are currently generating an optimizing program to apply the rules within the heuristics obtained from a large set of full custom switching tree designs. These designs have been generated by the VLSI Research Group at the University of Windsor. The heuristics include transistor

merging techniques, and node capacitance and interconnect considerations. A transistor scaling feature, based on an approximate delay chain model [14]. The final goal is the minimizing of silicon area while maintaining a required pipeline throughput rate.

### 6.5. Switching Tree Circuits

Our initial target area for switching tree logic is in massively pipelined systems, where each evaluation node is pipelined.

A robust circuit technique is to embed a tree within the true single phase clock system of Yuan and Svensson [18]. This is shown in Fig. 6.10. Since we are only interested in implementing  $n$ -

channel logic blocks, we use a single inverter p-channel block at the output of each n-channel block.

Using this dynamic latch arrangement, we have simulated mask extracted circuits for up to 6 logic inputs at pipeline rates of 50MHz for a 3μ CMOS double metal, p-well process. These results have recently been verified by fabrication.

### 6.6 Modulo 7 Multiplier

Residue computations produce don't care states arising from the fact that some output states are not reachable in a modulo m computer where  $2^i < m < 2^{i+1}$ . We are therefore at liberty to either remove, or leave in place, the appropriate leaf transistor.

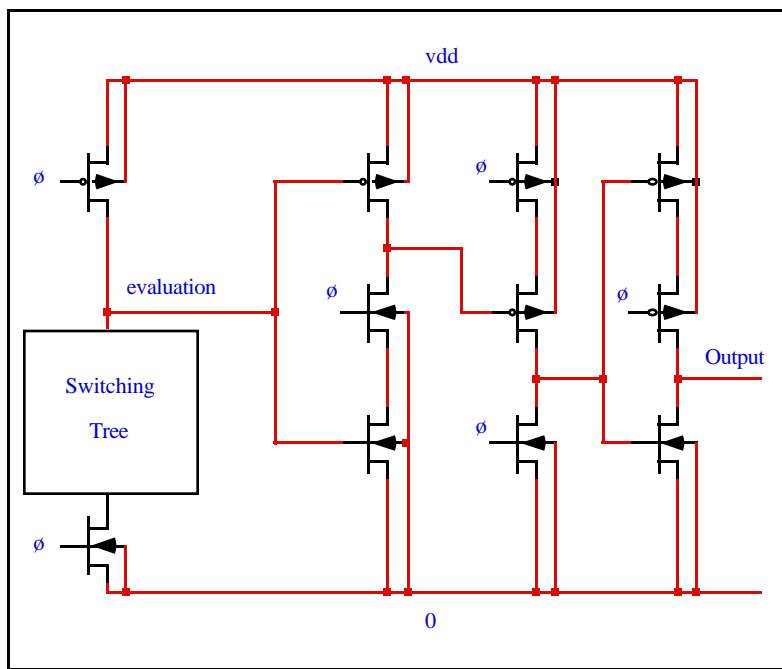


Fig 6.10. Embedding a Switching Tree in the Single Phase D-Latch

Clearly we make our selection based on a pattern that allows maximum edge reduction. The full programmed tree for bit 0 of the calculation  $(a \times b) \bmod 7$ , is shown in Fig. 6.11. The bold lines are the don't care states that have been programmed as 1's for minimized transistor count. Note the obvious symmetry in the pattern of don't care 1's that has been chosen. The pattern is selected for the direct application

of rule 1 or rule 2. Rule 1 is applied when a complete sub-tree is generated by the don't care state. Rule 2 is applied when common edges are placed beneath full sub-trees. The minimization

procedure reduced the original 126 transistor positions of the full tree to 24 transistors; the final switching tree transistor array is shown in Fig. 6.12.

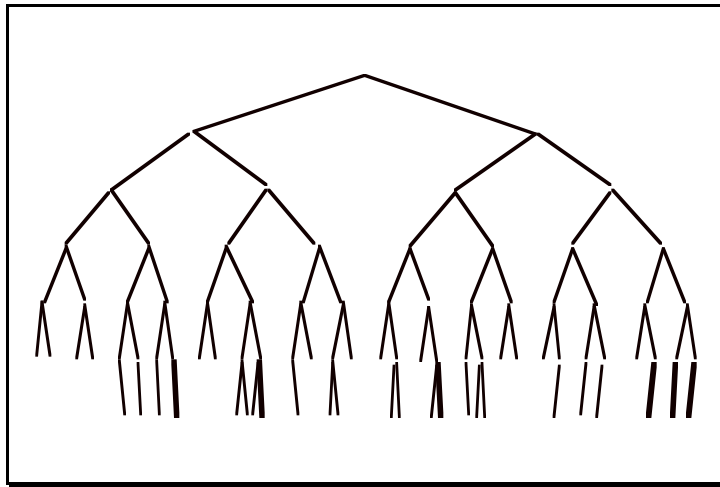


Fig. 6.11 Bit-0 of mod 7 multiplier-full programmed tree

Important points to note are that: the original bottom 64 transistor positions have been replaced by only 4 transistors; the maximum node capacitance is identical to the 3-transistor full binary tree node capacitance; the maximum gate load is 3 transistors; the maximum path length has been reduced from 6 to 5 transistors. It is also important to note

that the tree can be inverted for better charge sharing characteristics if required. All of our simulations have used the tree in its natural orientation.

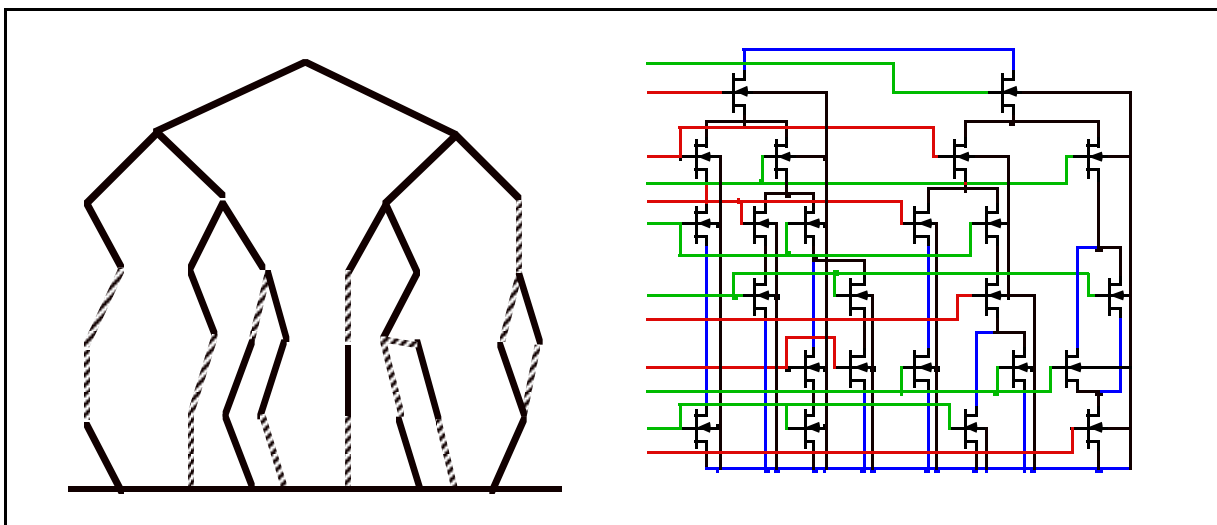


Fig. 6.12 minimized n-channel tree and transistor array

## 6.7 Results

Fabrication of a complete DSP system is still several months away as of writing this chapter. The operational characteristics of arrays of finite ring cells can be assessed, however, from initial fabrication results of test cells.

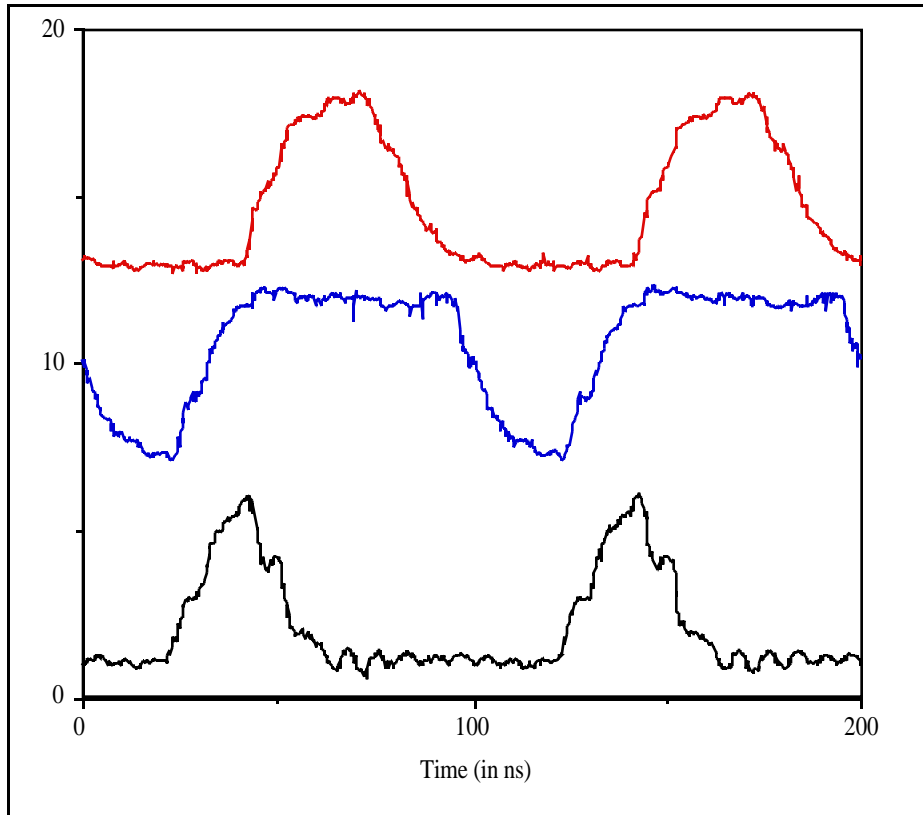


Fig 6.13 Results from fabricated test chip

We have fabricated several cells of 6 high switching trees to verify our simulation results of operation at 50MHz clock rate in a 3 $\mu$ m CMOS double level metal process<sup>1</sup>. The fabrication results verify the clock rate up to the bandwidth of our output pad drivers (40MHz) and low cycle rate

measurements indicate operation at the target rate. Charge sharing simulation results are verified by the fabrication, indicating that the switching tree complex block driving the true single phase clock system, provides sufficient voltage change at the evaluation node to solidly drive the pipeline buffer circuitry.

Results, obtained from a test chip, are shown in Fig. 6.13 for operation at the bandwidth limit of the output pad drivers.

<sup>1</sup> Northern Telecom CMOS 3 process available to the Canadian Microelectronics Corp.

## 7. Conclusions

This chapter has developed the theory and discussed the applications of number theoretic techniques for the design of high performance digital signal processing systems. The basic principles of residue number systems have been introduced along with a new finite polynomial ring mapping strategy for inner product computations. Both RNS and polynomial ring theories have been combined to illustrate the ability to perform finite ring computations through the use of look-up tables. The look-up tables are shown to be implemented using minimized binary trees (switching trees) embedded in a true single phase dynamic pipeline circuit structure. Test results from fabricated chips illustrate the operation and validity of our approach.

Although the mathematical theory of finite arithmetic is a very old subject, it has become the object of some study for over thirty years. The newly emerging digital computer industry, of thirty years ago, carried out particularly intensive research into the subject, but applications of the concepts have never been successful in general purpose digital computers where the deficiencies of non-binary techniques tend to counterbalance the advantages. However, the modern requirements of real-time digital signal processing and the capabilities of VLSI implementation form an ideal setting for number theoretic techniques. Today, these finite arithmetic concepts are being used primarily by engineers in research and development, and only occasionally find their way into practical solutions to engineering problems. It remains to be seen if finite arithmetic concepts will provide important solutions for signal processing problems of the 21st century.

The role of finite arithmetic in the implementation of high speed computational circuitry has taken on different forms over the past three decades. It has always been driven by technology, and the situation has not changed. The technology of today is VLSI and ULSI and of tomorrow GLSI, ..... We have seen, in this chapter, a new approach to implementing VLSI finite arithmetic circuits, and it is clear that our implementation techniques may have to change to fit this implementation vehicle. It is clear that in the field of DSP, where very fast computations are

required, that finite arithmetic has a very definite role to play. The secret to using the arithmetic correctly is to allow the requirements of the medium to drive the search for more efficient architectures, not the 'shoe-horning' of existing approaches, developed for discrete components, that so often marks our use of emerging technologies.

## 8. Acknowledgements

The authors wish to acknowledge the financial assistance of several grants from the Natural Sciences and Engineering Research Council of Canada. In addition they would also like to thank the Canadian Microelectronics Corp. for providing fabrication and design equipment support to the VLSI Research Group. Final thanks go to the many graduate students who produced the fabricated test chips.

## 9. References

1. Bayoumi, M. A., G. A. Jullien and W. C. Miller. "A VLSI Implementation of Residue Adders." IEEE Trans. on Circuits and Systems. CAS-34, 1987.
2. Bryant, R. E. "Graph-based algorithms for boolean function manipulation." IEEE Trans. on Comput. vol.35, pp .677-691, 1986.
3. Chu, M. K. and D. I. Pulfrey. "Design procedures for differential cascode-Voltage switch circuits." IEEE Trans. Solid-State Circuits. vol.SC-21, pp. 1082-1087, 1986.
4. Godement, R. "Algebra." 1968 Hermann. Paris.
5. Hatamian, M. and G. L. Cash. High Speed Signal Processing, Pipelining, and Signal Processing. Int. Conf. on Acoustics, Speech, and Signal Processing. April: pp.1173-1176., 1986.
6. Jullien, G. A. "Residue Number Scaling and Other Operations Using ROM Arrays." IEEE Trans. Comput. C-27, 325,336, 1978.

7. Jullien, G. A., P. D. Bird, J. T. Carr, M. Taheri and W. C. Miller. "An Efficient Bit-Level Systolic Cell Design for Finite Ring Digital Signal Processing Applications." *J. VLSI Sig. Proc.* I, (In Print), 1989.
8. Kung, H. T. and C. E. Leiserson. *Systolic Arrays (for VLSI)*. Sparse Matrix Proc. 256-282, 1978.
9. McCanny, J. V. and J. G. McWhirter. "Implementation of Signal Processing Functions using 1-bit Systolic Arrays." *Electronics Letters.* 18, 241, 1982.
10. Peled, A. and B. Liu. "A new hardware realization of digital filters." *IEEE Trans. on Acoustics, Speech, and Signal Processing.* ASSP-22, 456-462, 1974.
11. Rabiner, L. R. and B. Gold. "Theory and Application of Digital Signal Processing." 1975 Prentice-Hall. Englewood Cliffs, N.J.
12. Schilling, O. F. G. and W. S. Piper. "Basic Abstract Algebra." 1975 Allyn & Bacon. Boston.
13. Shannon, C. E. "A Symbolic Analysis of Relay and Switching Circuits." *Trans. AIEE.* vol. 57, pp. 713-723, 1938.
14. Shoji, M. "FET Scaling in domino CMOS Gates." *IEEE. J. Solid-State Circuits.* vol. 20, pp. 1067-1071, 1985.
15. Soderstrand, M. A., W. K. Jenkins, G. A. Jullien and F. J. Taylor. *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing.* 1986.
16. Taheri, M., G. A. Jullien and W. C. Miller. "High Speed Signal Processing Using Systolic Arrays Over Finite Rings." *IEEE Trans. Selected Areas in Comm.* 6, 504-512, 1988.
17. Youn, H. Y. and A. D. Singh. "On implementing large binary tree architectures in VLSI and WSI." *IEEE Trans. on Comput.* vol.38, pp .526-537, 1989.
18. Yuan, J. and C. Svensson. "High-Speed CMOS Circuit Technique." *IEEE. J. Solid-State Circuits.* vol. 24, pp. 62-70, 1989.
19. Zuckerman, H. and Niven, I. "An Introduction to the Theory of Numbers." 1972 John Wiley & Sons Inc. New York.