



*Large Dynamic Range  
Computations over Small  
Finite Rings*

*N.M. Wigley, G.A. Jullien and D. Reaume*

---

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*

---

# Large Dynamic Range Computations over Small Finite Rings

N. Wigley, G.A. Jullien and D. Reaume

VLSI Research Group, University of Windsor  
Windsor, Ontario, Canada N9B 3P4

## Abstract

This paper presents a new multivariate mapping strategy for the recently introduced Modulus Replication Residue Number System (MRRNS). This mapping allows computation over a large dynamic range using replications of extremely small rings. The technique maintains the useful features of the MRRNS, namely: the ease of input coding; the absence of a Chinese Remainder Theorem inverse mapping across the full dynamic range; the replication of identical rings; the natural integration of complex data processing.

The concepts are illustrated by a specific example of complex inner product processing associated with a radix-4 decimation in time FFT algorithm. A complete quantization analysis is performed and an efficient scaling strategy chosen based on the analysis. The example processor uses replications of three rings modulo 3, 5 and 7; the effective dynamic range is in excess of 32-bits.

The paper also includes VLSI implementation strategies for the processor architecture which consists of arrays of massively parallel linear bit-level pipelines.

**Key Words:** Quadratic residue rings; Polynomial rings; Residue Number Systems; Complex Arithmetic; Inner Product Computations; VLSI Signal Processors; Dynamic Logic.

# 1. Introduction

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) [8]. It derives from the classical form of the Chinese Remainder Theorem (CRT).

Normal techniques for arithmetic computation require some form of carry propagation; this occurs at regular intervals during the computation, and the mechanism for carry propagation is one of the main driving forces behind the various number representations used for arithmetic computation. In using residue systems, we are able to defer the entire carry propagation procedure until a considerable part of the computation has been completed. Since carry data flows orthogonally to word data, the finite ring approach, which removes the carry flow, allows easier clocking [5], testing[6] and fault tolerance strategies [9] to be used.

The RNS does, however, have drawbacks. Since the number of moduli bears a direct relationship on the size of the dynamic range, a need for an increased dynamic range is reflected in the need for ever-larger moduli. The structures of the finite rings associated with these larger moduli become increasingly complex to implement in hardware. Indeed, practical implementations of RNS devices have essentially been limited to five, and on occasion, six-bit moduli, with one fixed and one variable input at each stage of the computation.

Treatment of complex integers can lead to more restrictions on the choice of moduli. Use of the QRNS (Quadratic Residue Number System) can, on the one hand, simplify the design and decrease the area of hardware; on the other hand, the QRNS imposes yet another condition on the moduli (that they be of the form  $4k + 1$ ) and thus creates the need for even larger moduli.

The authors have recently introduced a modified version of the RNS called the MRRNS (Modulus Replication RNS) [12]. The MRRNS is based on a version of the CRT which holds for polynomial rings. The distinction between the two lies in the fact that in the RNS all of the moduli must be relatively prime; this requirement is completely relaxed in the MRRNS framework.

The MRRNS allows the repeated use of moduli in order to increase the dynamic range of the desired computation. Moreover, the QRNS requirement that the moduli be of the form  $4k + 1$  can be relaxed, albeit at the cost of some extra hardware (but at no increase in the complexity of design). These two facts together have enabled us to put together a system which will execute a 1024-point FFT with 17-bit input data, using the smallest possible ring moduli of 3, 5, and 7. To do this we introduce a new, multivariate version of the MRRNS.

We also coalesce the use of the replication of small modulus rings with a recently introduced complex dynamic CMOS dynamic logic design technique, referred to as *Switching Trees* [14]. This design technique converts look-up table storage into minimized complex transistor circuitry based on a binary tree structure. The technique allows input data up to a wordlength of 6 bits, and allows outputs of an arbitrary number of bits. In particular we illustrate how multiplications and additions in the finite rings can be accomplished through the use of these tools, and a high speed single phase clock pipeline structure [13].

We start the main body of the paper with the necessary definitions and theorems on which the theory is based. There follows a discussion of the preliminary encoding of integers as polynomials. We then give a treatment of the further encoding and decoding of polynomials as vectors. We then discuss the example FFT both in terms of architecture, including scaling requirements and its implementation. We conclude with a discussion of VLSI implementation strategies.

## 2. Definitions and Theorems

In this paper we introduce a multivariate approach to the MRRNS [12], whereby several indeterminates are used to stand for various radices, each a distinct power of 2. This use of several variables allows very large dynamic range computations to be performed over very small finite rings. We begin with the definitions of the rings needed, and continue with the principle theorems used.

### 2.1 Definitions

Let  $M$  be a positive integer. Let  $Z$  denote the ring of integers. Let  $Z_M$  denote the ring of integers modulo  $M$ . The elements of  $Z_M$  can be taken as residue classes of integers, but it is more convenient and customary to use a set of  $M$  consecutive integers. Addition and multiplication in these rings are the usual operations of modulo arithmetic.

Let  $X$  denote a variable. If  $R$  is any ring then we let  $R[X]$  denote the ring of all polynomials in the variable  $X$  with coefficients in  $R$ , with the usual addition and multiplication of polynomials. Thus  $Z[X]$  consists of all polynomials in  $X$  whose coefficients are integers, and  $Z_M[X]$  consists of all polynomials in  $X$  whose coefficients are in the ring  $Z_M$ . Addition and multiplication in  $Z_M[X]$  are defined by reducing the coefficients of a sum or product modulo  $M$ . Thus in  $Z_{13}[X]$ , if  $f(X) = 5X^2 + 4X$  and  $h(X) = 3X^2 - 6$ , then  $f(X) + h(X) = -5X^2 + 4X - 6$ , and  $f(X) * h(X) = 2X^4 - X^3 - 4X^2 + 2X$ .

#### 2.1.1 Quotient Rings

Let  $g(X)$  be a polynomial of degree  $m$  from  $Z_M[X]$ . We denote by  $Z_M[X]/(g(X))$  the ring of polynomials which have coefficients in  $Z_M$  and have degree less than  $m$ . This ring is called the ring of quotients of  $Z_M[X]$  with respect to  $g(X)$ . Addition is straightforward. Multiplication of two polynomials is defined by first multiplying them in  $Z[X]$ , and then reducing the product by

means of the polynomial  $g(X)$ . This is done by using the Division Algorithm for Polynomials. Finally we reduce the coefficients of the product (mod  $M$ ).

If, as above, we have  $f(X) = 5X^2 + 4X$  and  $h(X) = 3X^2 - 6$ , and if  $g(X) = X^3 - X$ , then by the Division Algorithm we have  $f(X)h(X) = g(X)(2X - 1) + (-2X^2 + X)$ , and thus in the ring  $Z_M[X]/(g(X))$  the product  $f(X)h(X) = -2X^2 + X$ .

In the case of two variables, with two fixed polynomials  $g(X)$  and  $h(Y)$  (with coefficients in  $Z_M$ ), we can define the ring  $Z_M[X, Y]/(g(X), h(Y))$  as the set of all polynomials  $f(X, Y)$  which have coefficients in  $Z_M$ , have degree less than  $m$  in the variable  $X$  and degree less than  $n$  in  $Y$ . Addition is straightforward, and multiplication is defined according to the rule of taking the least residue modulo  $g(X)$  and modulo  $h(Y)$  as well as reducing the coefficients (mod  $M$ ).

### 2.1.2 Direct Product Rings

The *direct product* of two rings  $R$  and  $S$  is denoted by  $R \times S$  and consists of the set of all pairs of elements  $(r, s)$  where  $r \in R$  and  $s \in S$ . Addition and multiplication are performed within components. Thus  $(r_1, s_1) + (r_2, s_2) = (r_1 + r_2, s_1 + s_2)$ , and  $(r_1, s_1) \cdot (r_2, s_2) = (r_1 \cdot r_2, s_1 \cdot s_2)$ . The direct product of two rings is also a ring. Note that some authors write  $R \oplus S$  and use the terminology ‘direct sum’.

For example in the direct product ring  $Z_7 \times Z_9$  the elements  $(3, 2)$  and  $(-3, 4)$  have sum given by  $(3, 2) + (-3, 4) = (0, -3)$ , and their product is given by  $(3, 2) \cdot (-3, 4) = (-2, -1)$ .

If  $R_1, \dots, R_p$  are rings then we shall use the notation  $\prod_i R_i$  to indicate the direct product  $R_1 \times R_2 \times \dots \times R_p$ .

The benefit of direct product rings to the VLSI designer is that a stream of computations will be performed wholly in each individual channel, independently of computations in the other channels.

This is to be contrasted with the cross channel communication required by typical binary arithmetic systems.

### 2.1.3 Homomorphisms and Isomorphisms

If  $R$  and  $S$  are two rings then a *ring homomorphism* from  $R$  to  $S$  is a map  $\Phi$  from  $R$  to  $S$  which has the properties of being additive and multiplicative, i.e. for any two elements  $r_1, r_2 \in R$  we have  $\Phi(r_1 + r_2) = \Phi(r_1) + \Phi(r_2)$  and  $\Phi(r_1 \cdot r_2) = \Phi(r_1) \cdot \Phi(r_2)$ .

If  $R$  is any ring then another example of a homomorphism is the map from  $R[X]$  to  $R$  given by fixing an element  $r$  of  $R$  and then evaluating each polynomial  $f(X)$  at the value  $f(r)$ . We have  $\{f(X) + g(X)\}|_{X=r} = f(X)|_{X=r} + g(X)|_{X=r}$ , since both sides reduce to  $f(r) + g(r)$ . A similar result holds for multiplication. This map is called an *evaluation map*.

An *isomorphism* is a homomorphism  $\Phi$  from  $R$  to  $S$  for which the inverse map  $\Phi^{-1}$  from  $S$  to  $R$  exists. A necessary and sufficient condition for  $\Phi^{-1}$  to exist is that  $\Phi$  satisfy two conditions; the range of  $\Phi$  must be all of  $S$  ( $\Phi$  is *onto*), and  $\Phi$  must be 1-1. This means that if  $\Phi$  maps any element of  $R$  to the zero element of  $S$  then the element of  $R$  must itself have been zero. In other words, if  $r \in R$  is nonzero then  $\Phi(r)$  must be nonzero.

If there is an isomorphism from  $R$  to  $S$  then we say that  $R$  and  $S$  are isomorphic. It follows that any additions or multiplications that are performed in the one ring will have these operations echoed in the other ring. Hence computations can be performed in one ring and the results mapped to the other ring. We shall use this fact below to map data from a ring of polynomials to a direct product ring, in order to perform the computations in the direct product ring. This enables us to take advantage of the independent channel arithmetic discussed earlier.

## 2.2 The basic isomorphisms

We now lay the theoretical groundwork for our conversion of a given algorithm, originally stated in binary form, into a problem which can be implemented in a direct product ring. To do this we

give an isomorphism from a ring of polynomials to a direct product ring. Let  $g(X)$  be a fixed polynomial of degree  $m$  from  $Z_M[X]$ . We shall assume throughout that  $g(X)$  has all of its roots  $r_1, r_2, \dots, r_m$  in the ring  $Z_M$ , and that each of the differences  $r_i - r_j$ , when  $i \neq j$ , is invertible in the ring  $Z_M$ . Under this condition on the roots  $\{r_i\}$  we can find polynomials  $\varphi_i(X)$  satisfying the condition  $\varphi_i(r_j) = 0$  for  $j \neq i$ , and  $\varphi_i(r_i) = 1$ . Indeed, set

$$\varphi_i(X) = \prod_{\substack{j=1 \\ j \neq i}}^m (X - r_j) (r_i - r_j)^{-1}.$$

The term  $(r_i - r_j)^{-1}$  denotes the multiplicative inverse of  $(r_i - r_j)$  in the ring  $Z_M$ , which exists by the assumption above.

First we state a lemma which is valid for more general rings.

Lemma 1. Let  $f(X)$  be a polynomial in  $X$  with coefficients in a commutative ring  $R$ . Then an element  $a \in R$  is a root of  $f$  if and only if there exists a polynomial  $F(X)$  with coefficients in  $R$  such that

$$f(X) = (X - a)F(X).$$

Proof. [3, pg. 435].  $\square$

The next lemma is usually stated for a prime modulus  $M$ . With the conditions stated below, however, it also holds when  $M$  is not prime.

Lemma 2. Let  $f$  be a polynomial in  $X$  with coefficients in  $Z_M$ . Suppose that  $f$  vanishes at the points  $r_1, r_2, \dots, r_m \in Z_M$ . If each difference  $r_i - r_j, i \neq j$ , is invertible in  $Z_M$ , then  $f$  is either the zero polynomial or it has degree greater than or equal to  $m$ .

Proof. From lemma 1 it follows that  $f(X) = (X - r_1)F(X)$  for some polynomial  $F(X)$ . Setting  $X = r_j$  for any  $1 < j \leq m$  we obtain

$$0 = f(r_j) = (r_j - r_1)F(r_j).$$

Since  $r_j - r_1$  is assumed invertible it follows that  $F(r_j) = 0$ . Thus  $F(X)$  vanishes at each element of  $\{r_i; 1 < i \leq m\}$ . The lemma then follows by induction on  $m$ .  $\square$

Under these conditions we prove the following theorem, which is a form of the Chinese Remainder Theorem for polynomials. Note that the usual CRT for polynomials usually assumes coefficients in a field [1, pg. 60].

**Theorem 1.** Let  $f(X) \in Z_M[X]/(g(X))$ . The evaluation map of  $f(X)$  at all of the roots of  $g(X)$ , given by  $f(X) \rightarrow (f(r_1), f(r_2), \dots, f(r_m))$ , is not only a homomorphism from  $Z_M[X]/(g(X))$  to the direct product ring  $Z_M \times Z_M \times \dots \times Z_M$  (the direct product taken  $m$  times), but also an isomorphism  $Z_M[X]/(g(X)) \cong Z_M \times Z_M \times \dots \times Z_M$ .

**Proof.** Let  $f(X)$  have coefficients in  $Z_M$  and degree less than  $m$ . Then the map  $f(X) \rightarrow (f(r_1), f(r_2), \dots, f(r_m))$  is an evaluation map of  $f(X)$ , and hence it is a homomorphism.

We must show that it is 1-1 and onto.

To show that it is onto, let  $(a_1, a_2, \dots, a_m)$  be any element in the cross-product ring. Using the functions  $\{\varphi_i(X)\}$  from above, we define the polynomial  $f(X) = \sum_{i=1}^m \alpha_i \varphi_i(X)$  and we see that it maps to the element  $(\alpha_1, \alpha_2, \dots, \alpha_m)$ . Hence  $(\alpha_1, \alpha_2, \dots, \alpha_m)$  belongs to the range, and thus the map is onto.

To show that the map is 1-1, suppose that  $f(X)$  is a polynomial of degree less than  $m$  and it maps to zero. We must show that  $f(X)$  is the zero polynomial. Since the image of  $f(X)$  is the zero element of the direct product ring, this means that for each  $i = 1, \dots, m$  we have  $f(r_i) = 0$ . Thus  $f(X)$  has  $m$  distinct roots  $\{r_1, r_2, \dots, r_m\}$ . Since  $f(X)$  has degree  $< m$ , by lemma 2 it must be the zero polynomial.  $\square$

We now turn to functions of more than one variable. Although in the rest of the paper we are concerned with polynomials of more than two variables, the proofs are made clearer if we restrict ourselves to two variables. The extension to more than two variables is easily made.

We assume now that  $\{r_i:1 \leq i \leq m\}$  and  $\{s_j:1 \leq j \leq n\}$  are two sets of elements of  $Z_M$  with the property that each difference  $r_i - r_p (i \neq p)$  and  $s_j - s_q (j \neq q)$  is invertible in  $Z_M$ . Let  $\{\varphi_i(X):1 \leq i \leq m\}$  and  $\{\psi_j(Y):1 \leq j \leq n\}$  be polynomials with the property that  $\varphi_i(r_i) = 1 = \psi_j(s_j)$  and  $\varphi_i(r_p) = 0$  for  $p \neq i$  and  $\psi_j(s_q) = 0$  for  $q \neq j$ ; the construction of these polynomials follows exactly as in the case for one variable, given above.

Let  $g(X) = \prod_{i=1}^m (X - r_i)$ , and let  $h(Y) = \prod_{j=1}^n (Y - s_j)$ . Then we have the following theorem.

**Theorem 2.** Let  $f(X, Y) \in Z_M[X, Y] / (g(X), h(Y))$ , i.e.  $f(X, Y)$  is a polynomial with coefficients in  $Z_M$  whose degree in  $X$  is less than  $m$ , and whose degree in  $Y$  is less than  $n$ . The evaluation map of  $f(X, Y)$  at all pairs of roots of  $g(X)$  and  $h(Y)$ , given by

$$f(X, Y) \rightarrow (f(r_1, s_1), f(r_1, s_2), \dots, f(r_m, s_n)),$$

is not only a homomorphism from  $Z_M[X, Y] / (g(X), h(Y))$  to  $Z_M \times Z_M \times \dots \times Z_M$  (the direct product taken  $mn$  times), but also an isomorphism:

$$Z_M[X, Y] / (g(X), h(Y)) \cong Z_M \times Z_M \times \dots \times Z_M.$$

**Proof.** The map is an evaluation map and is thus a homomorphism. We must show that it is an isomorphism, i.e. that it is 1-1 and onto.

To show that the map is onto, suppose that  $(\alpha_{ij}:1 \leq i \leq m, 1 \leq j \leq n)$  is an element of the cross-product ring. Then the element  $f(X, Y) = \sum_{i=1}^m \sum_{j=1}^n \alpha_{ij} \varphi_i(X) \psi_j(Y)$  maps to  $(\alpha_{ij})$ , so the homomorphism is onto.

To show that it is 1-1, suppose that  $f(X, Y)$  is a polynomial of degree less than  $m$  in  $X$  and less than  $n$  in  $Y$ , and suppose that  $f(r_i, s_j) = 0$  for  $1 \leq i \leq m$  and  $1 \leq j \leq n$ . It is sufficient to prove that

$f(X, Y)$  is the zero polynomial. We write  $f(X, Y) = \sum_{j=0}^{n-1} f_j(X)Y^j$ , and then, for each  $i$ ,  $1 \leq i \leq m$ , we can write  $f(r_i, Y) = \sum_{j=0}^{n-1} f_j(r_i)Y^j$ .

By setting  $Y = s_j$  for  $1 \leq j \leq n$ , we see that the polynomial  $\sum_{j=0}^{n-1} f_j(r_i)Y^j$  has  $n$  roots; since it is of degree less than  $n$  in the variable  $Y$ , by lemma 1 it must be the zero polynomial. This says that each of its coefficients is zero. But this means that for  $1 \leq i \leq m$  we have  $f_j(r_i) = 0$ . Since  $i$  was arbitrary it follows that  $f_j(X)$  has  $m$  distinct roots. But  $f_j(X)$  is of degree less than  $m$ , and thus by lemma 2 it follows that  $f_j(X)$  is the zero polynomial. Hence  $f(X, Y)$  is also the zero polynomial, and we have proved that only the zero polynomial gets mapped to the zero element. This implies that the map from  $Z_M[X, Y] / (g(X), h(Y))$  to the cross-product ring is 1-1, and hence is an isomorphism.  $\square$

### 3. Encoding and Decoding of Integers.

We now show how to convert a DSP algorithm into a computation in a direct product ring. For simplicity let us assume that the algorithm to be solved consists of computing the inner product of two vectors of complex integers; this will cover a great number of DSP algorithms. We first give a schematic of the process, and accompany it with a simplified example. In Fig. 1 we have a diagram of all of the rings that we shall be using, together with the mappings between them. The rings are denoted by the boxes.

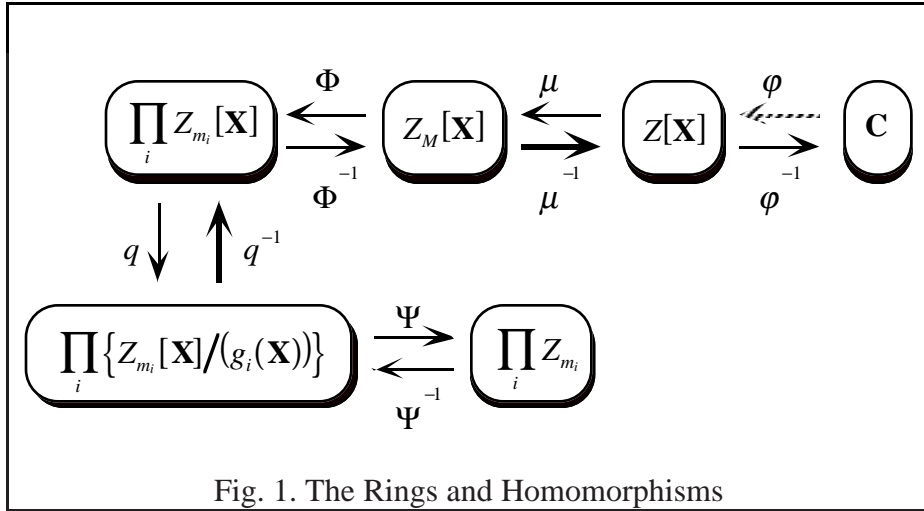


Fig. 1. The Rings and Homomorphisms

For simplicity in the diagram we have represented the set of variables  $X_1, X_2, \dots, X_n$  with the symbol  $\mathbf{X}$ . Encoding of the data begins in the upper right at the complex numbers

$2^i < m < 2^{i+1} \mathbb{C}$  and continues counter-clockwise around the diagram to the lower right. In the direct product ring  $\prod_i Z_{m_i}$  the computation of the inner product is performed. The answers are

then decoded, following the reverse direction. We shall give explanations of each step below, accompanied by a simple example. For the sake of definiteness in the example we take  $M = 15 = 3 \cdot 5$ , and  $\mathbf{X} = \{T, X\}$ , where  $T$  stands for the complex unit  $j$  and  $X$  stands for 4. We shall define the quotient rings later by specifying the polynomials  $\{g_1(T), g_2(T), g_1(X), g_2(X)\}$ .

### 3.1 Representing integers as polynomials (the map $\varphi$ )

The input data are assumed to be given as two sequences of complex integers of a given bitlength. They are then represented as polynomials in  $Z[X_1, \dots, X_n]$ , as shown below. Instead of calculating the inner product of two sequences of complex integers we thus perform the inner product of two sequences of polynomials and then apply the map  $\varphi^{-1}$ . Since  $\varphi^{-1}$  is a homomorphism, it preserves sums and products and thus it preserves inner products. Thus it is sufficient to find the polynomial in  $Z[\mathbf{X}]$  which represents the inner product; this is the forward map,  $\varphi$  (this is not a homomorphism, and is shown hatched on Fig. 1). The rest of this section is devoted to this problem.

For the sake of example and to show how the various homomorphisms work, we shall map the complex integer  $13 - 11j$  to the direct product ring. Using  $T$  in place of  $j$  as well as  $X$  in place of the radix 4, we have  $13 - 11j = 3 \cdot 4 + 1 - j(2 \cdot 4 + 3) \rightarrow 3X + 1 - T(2X + 3)$ . By the same means we can represent any integer  $A + jB$ , where  $|A|, |B| \leq 15$ , with a polynomial of the form

$$\sum_{i_1, i_2 \in \{0,1\}} a_{i_1 i_2} T^{i_1} X^{i_2} \quad (3.1)$$

with coefficients from the interval  $[-3, 3]$ . To allow a greater range for  $A$  and  $B$  we could introduce more variables  $Y, Z$ , etc., to stand for 16, 256, etc.

### 3.2 The maps $\mu$ and $\Phi$

The modulus  $M$  is selected in advance, and is assumed in general to factor in the form  $M = \prod_i m_i$ , where the  $\{m_i\}$  are relatively prime to one another. In our example we have taken

$M = 15 = 3 \cdot 5$ . In the FFT example of section 4, we take  $M = 105 = 3 \cdot 5 \cdot 7$ . The data are then mapped to the ring  $Z_M[\mathbf{X}]$  by mapping the coefficients of each polynomial in  $Z[\mathbf{X}]$  to the corresponding elements of  $Z_M$ . As we shall see, the coefficients used later in this paper lie in the interval  $[-1, 1]$ , and thus require no processing in the forward direction if  $M \geq 3$ . In the present example the coefficients lie in the interval  $[-3, 3]$  and the modulus is  $M = 15$ , so no forward processing is required.

The encoding then continues from  $Z_M[\mathbf{X}]$  to the direct product ring  $\prod_i Z_{m_i}[\mathbf{X}]$ . The map  $\Phi$  reduces coefficients with respect to the moduli  $\{m_i\}$ . In the FFT example to be given below, no processing is required for the coefficients of the DSP problem in the forward direction, since the coefficients are assumed to lie in the interval  $[-1, 1]$ . For our present example, the integer  $13 - 11j$  has the representation  $3X + 1 - T(2X + 3)$ . Converting to  $Z_{15}[T, X]$  requires no changes. To map further to the rings  $Z_3[T, X]$  and  $Z_5[T, X]$  we get, respectively,  $1 + TX$  and  $-2X + 1 - T(2X - 2)$ , which we can represent as  $[1 \ 0 \ 0 \ 1]$  for  $m = 3$  and  $[1 \ 2 \ -2 \ -2]$  for  $m = 5$ .

### 3.3 The map $q$

The map  $q$  takes each individual ring  $Z_{m_i}[\mathbf{X}]$  to the quotient ring  $Z_{m_i}[\mathbf{X}]/(g_i(\mathbf{X}))$ . This map reduces polynomials by calculating remainders by means of the Division Algorithm. In the MRRNS method, however, there is no reduction to be performed as the input polynomials already have degree low enough to require no reduction. Hence the map  $q$  plays no role in the encoding; it is merely a device to give the existence of the isomorphism  $\Psi$ .

In the present example the inputs are of degree one in each of the variables; since the output is going to be the inner product of such linear polynomials, the output will be a polynomial of degree two in each variable. For our example, therefore, it is sufficient to give each polynomial  $g_i$  degree 3. Thus we take  $g_i(T) = T(T^2 - 1)$ , and  $h_i(X) = X(X^2 - 1), i = 1, 2$ .

An exception to this occurs when the variable  $T$  represents the complex unit  $j$ . Since the complex unit  $j$  satisfies the equation  $T^2 = -1$ , we can lower the degree of  $g_i(T)$  from three to two whenever it is possible to use the polynomial  $g_i(T) = T^2 + 1$ . This requires  $g_i(T)$  to have its roots in the ring  $Z_{m_i}$ , and the roots must satisfy the root condition of the theorems (i.e. the difference  $r_i - r_j$  is invertible for  $i \neq j$ ). In the present example we have  $m_1 = 3$  and  $m_2 = 5$ . Clearly  $g(T) = T^2 + 1$  factors in  $Z_5$  as  $(T + 2)(T - 2)$ , but does not factor in  $Z_3$ . Hence we can take  $g_2(T) = T^2 + 1$  for the modulus  $m_2 = 5$ , but must take  $g_1(T) = T(T^2 - 1)$  for the modulus  $m_1 = 3$ . For the case of the variable  $X$  no simplification is possible, and thus  $h_i(X) = X(X^2 - 1)$  for both moduli 3 and 5.

### 3.4 The map $\Psi$

Finally, the map  $\Psi$  is the map of theorem 2, which evaluates all polynomials at all possible combinations of roots of the polynomials  $\{g_i(\mathbf{X})\}$ . Note that the direct product ring, which we have indicated as  $\prod_i Z_{m_i}$  in the diagram, consists of  $d_1 d_2 \dots d_n$  individual copies of each of the rings  $Z_{m_i}$ ; by  $d_i$  we mean the degree of  $g_i(\mathbf{X})$ .

Since inputs and outputs have different polynomial degrees, it is important to take care of the highest degree possible. Inputs in our examples have degree one in each variable, and outputs have degree two. Thus we shall represent the components of  $\Psi$  with a  $9 \times 9$  matrix for the modulus  $m = 3$ , and as a  $6 \times 6$  matrix for the modulus  $m = 5$ .

With  $m = 3$ , the polynomial  $A_1 + A_2X + A_3X^2 + T(A_4 + A_5X + A_6X^2) + T^2(A_7 + A_8X + A_9X^2)$  evaluates at the roots  $T = -1, 0, 1$  and  $X = -1, 0, 1$  to yield the matrix product:

$$\begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \\ A_6 \\ A_7 \\ A_8 \\ A_9 \end{bmatrix}$$

It is significant that this matrix is a tensor product  $\Gamma \otimes \Gamma$  of matrices. Indeed, let the polynomial  $f(X) = \alpha_1 + \alpha_2X + \alpha_3X^2$  be evaluated at the points  $X = -1, 0, 1$ . Then the matrix  $\Gamma$  of the transformation is shown in the following product, which represents the evaluation:

$$\begin{bmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}$$

It is easily seen that  $\Gamma \otimes \Gamma$  is the  $9 \times 9$  matrix above. This fact is highly beneficial in the VLSI layout problem, where we may sometimes deal with four or more variables.

For the modulus  $m = 5$  the situation is simpler. We treat polynomials which are quadratic in  $X$  and linear in  $T$ , say  $A_1 + A_2X + A_3X^2 + T(A_4 + A_5X + A_6X^2)$ . We set  $T = -2, 2$  and  $X = -1, 0, 1$ , and we obtain

$$\begin{bmatrix} 1 & -1 & 1 & -2 & 2 & -2 \\ 1 & 0 & 0 & -2 & 0 & 0 \\ 1 & 1 & 1 & -2 & -2 & -2 \\ 1 & -1 & 1 & 2 & -2 & 2 \\ 1 & 0 & 0 & 2 & 0 & 0 \\ 1 & 1 & 1 & 2 & 2 & 2 \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \\ A_6 \end{bmatrix}$$

The matrix is the tensor product  $\Gamma_1 \otimes \Gamma$ , where  $\Gamma$  is as before and  $\Gamma_1$  is the matrix corresponding to the evaluation of the polynomials  $A_1 + A_2 T$  at the points  $T = -2, 2$ .

### 3.5 The inner product

Once the data have been converted to elements of this last ring, the inner product is performed in each component of the direct product ring  $\prod_i Z_{m_i}$ . The results are then mapped back to  $\mathbf{C}$ .

### 3.6 Reversing the maps $\Psi$ and $q$

The map  $\Psi$  is an isomorphism, so  $\Psi^{-1}$  poses no problem. The matrix components of  $\Psi^{-1}$  can be obtained by inverting the matrices  $\Gamma$  and  $\Gamma_1$  above and forming the respective tensor products.

The map  $q$  reduces polynomials of higher degree  $(\text{mod } g_i(\mathbf{X}))$ ; as was stated before and will be shown below, we can choose the degrees of the polynomials  $\{g_i(\mathbf{X})\}$  high enough to ensure that the output data have degrees less than or equal to the degrees of the  $g_i$ , so that the map  $q^{-1}$  exists and requires no computation. This is shown as a bold line on Fig. 1.

### 3.7 Reversing the maps $\Phi$ , $\mu$ and $\varphi$

The classical CRT assures that the map  $\Phi^{-1}$  exists and is an isomorphism between  $\prod_i Z_{m_i}[\mathbf{X}]$  and  $Z_M[\mathbf{X}]$ . The next step, however, must be treated carefully. The map  $\mu$  has in general no inverse. Since  $\mu$  reduces coefficients by finding residues  $(\text{mod } M)$ , it is necessary that the coefficients of the answer polynomial be their own residues  $(\text{mod } M)$ . This is the crucial restriction on the size of  $M$  and clearly dictates the conditions on the existence of  $\mu^{-1}$ . This inverse mapping is indicated on

Fig. 1 as a bold line;  $\mu^{-1}$  and  $q^{-1}$  are both reverse mappings of homomorphisms under special constraints. Finally, the map  $\varphi^{-1}$  is the evaluation map, with each variable  $X_i$  taking on the value of some power of 2 or the complex unit  $j$ .

## 4. An Example: Implementing the FFT

In order to test the efficacy of embedding computations into rings of such small order, we will use the MRRNS technique to compute a 1024-point FFT using radix-4 computational elements and the moduli 3, 5, and 7. The implementation of a radix 4 FFT [7] in an RNS setting was studied comprehensively in [10], and we have used those results in the present project.

The input data are first represented as polynomials in the variables  $T, W, X, Y$  and  $Z$ , of degree 1 in each variable. Coefficients of the polynomials are from the set  $\{-1, 0, +1\}$ . The same type of representations holds for the twiddle factors. The four units of the DFT are represented by the polynomials  $\pm 1$  and  $\pm T$ . Hence the output of each stage of the FFT will consist of polynomials of degree 2 in each of the variables, and will have coefficients which could, in theory, be large. Since the computation is taken over the ring  $Z_{105}$ , modular overflow will take place if any of the coefficients is greater than  $52.5 = 105/2$  in absolute value.

In actual fact, of course, the computation will take place in the direct product ring, and the result will then be mapped back to the polynomial ring. However, this map is an isomorphism, and thus any difficulties that exist will appear, and thus can be studied in either setting. Modular overflow is easier to study in the framework of the polynomials.

After each stage of the FFT, i.e. after each DFT, the output must be converted to the same form as required for input data, since the output of one stage becomes the input for the next stage. Thus we require a means for transforming polynomials of degree two, with coefficients up to 52 in magnitude, into polynomials of degree 1 and coefficients from the set  $\{-1, 0, +1\}$ .

At the same time we must also be concerned with number growth. Inputs of 16 bits (plus a sign bit) will give rise to much larger outputs; these outputs must then be scaled down to 16 bit numbers. Thus both scaling and polynomial conversion are necessary.

During each stage of the FFT there is growth in the size of the coefficients of the transform, and normally scaling takes place after each such stage. In [10] a study was done on the growth of numbers, both from the theoretical and the practical standpoints. Experimental results were obtained by considering: pseudorandom numbers; sine waves plus pseudorandom numbers; and speech signals. Although the theoretical worst case upper bounds were 5.058, for  $N = 2048$ , a practical upper bound on number growth equal to 4 was considered quite reasonable for the input sequences under consideration. In the rest of this paper we shall make use of this empirical assumption. In addition to the growth of numbers arising through the FFT computations, we must also consider the scaling of the twiddle factors that occurs when performing the initial quantization.

Since the polynomial mapping will correctly map values in the interval  $(-2^{16}, 2^{16})$ , and since the twiddle factors are all equal to 1 in the first stage of the FFT, scaling may be disregarded provided we limit ourselves to inputs of 14 bits. In the following stages we quantize the twiddle factors with sixteen bits. Clearly scaling will be required after the 2nd, 3rd and 4th stages. For maximum accuracy, the scaling factor should be chosen so as to produce outputs which are as great as the input polynomial mapping scheme will allow. No scaling is performed at the end of the 5th stage.

## 5. Scaling, Polynomial Conversion and Error Analysis

The need for scaling at the end of each stage of the FFT can be carried out most efficiently by combining it with the requirement of conversion of polynomials. Reference [10] discusses both the A/D and twiddle factor quantization errors associated with the algorithm. We will specifically discuss the dynamic range overflow, as it pertains to our polynomial ring representation.

## 5.1 RNS Overflow

A potential source of error is overflow of the residue number system. This occurs when one of the coefficients of the output polynomials is greater in magnitude than  $52.5 = 105/2$ . We now show that, under normal circumstances, such an occurrence is extremely rare.

Let us consider the typical multiplication of polynomials that occurs in the MRRNS. We have two real integer multipliers, say  $A$  and  $B$ , which are represented as polynomials. For the FFT implementation we have used polynomials which are linear in each of their variables. Each integer  $A$  and  $B$  can then be written as such a polynomial with coefficients drawn from the set  $\{-1, 0, 1\}$  (see section 3.1):

$$A = \sum_{i_1, i_2, i_3, i_4 \in \{0,1\}} a_{i_1, i_2, i_3, i_4} W^{i_1} X^{i_2} Y^{i_3} Z^{i_4} \quad (5.1)$$

and

$$B = \sum_{i_1, i_2, i_3, i_4 \in \{0,1\}} b_{i_1, i_2, i_3, i_4} W^{i_1} X^{i_2} Y^{i_3} Z^{i_4} \quad (5.2)$$

Overflow occurs when one of the coefficients of the product polynomial  $C = AB$  is greater than 52 in magnitude. We have the identity:

$$c_{k_1 k_2 k_3 k_4} = \sum_{i_\mu + j_\mu = k_\mu} a_{i_1 i_2 i_3 i_4} b_{j_1 j_2 j_3 j_4} \quad (5.3)$$

where  $\mu$  takes on the values 1, ..., 4. Since  $i_n, j_m \in \{0,1\}; 1 \leq n, m \leq 4$ , we see that the maximum contribution to the sum on the right occurs when  $k_n = 1; 1 \leq n \leq 4$ . For example, if  $k_1 = 1$  then we can represent  $k_1$  in *two* ways, namely  $i_1 = 1, j_1 = 0$ , or vice versa. With  $k_1 = 0$  or 2 there is only one representation. Thus the coefficient with the greatest number of contributions is  $c_{1111}$ , which has 16 summands on the right hand side of (5.3). Thus, in theory, a value of 128 could occur in the implementation of a DFT of blocklength four (i.e. one stage of the FFT). In practice, of course, there is a great deal of cancellation of positive and negative terms, and typical values are much smaller.

## 5.2 Statistical Error Distribution

In order to examine the incidence of overflow we have simulated the multiplication of such polynomials using a program, *MODULUS*, developed within our group. Both the  $A$  and  $B$  multipliers are chosen to be polynomial representations of uniformly distributed random pseudointegers; the  $A$  multipliers are assumed to be of 14, 15 or 16 bits (plus sign bit), and the  $B$  multipliers have 16 bits.

The user specifies the statistical distributions of the input integers, the number and types of indeterminates to be used in the polynomial representations, and the blocklength of the inner product. Probability generating functions, PGFs, (which are themselves polynomials in some new, unrelated variable) are then computed for the coefficients of the input polynomials. From these, PGFs are computed for the output polynomials using standard statistical theories as well as the statistical independence of the distributions. These computations enable the user to determine requirements for the moduli of the RNS and MRRNS computations, and to determine overflow statistics.

Overflow for the example FFT is found to be exceedingly rare; the probability of an overflow error is less than  $10^{-9}$ . For a complex inner product with blocklength 8 (a radix 8 calculation), the statistical distribution of the coefficient most likely to overflow, namely  $c_{1111}$ , is given in Fig. 2. The distribution is essentially normal, and coefficient values exceeding 52 in absolute value still have very low probability; in the order of  $10^{-4}$ .

## 5.3 Scaling and conversion of output polynomials

It has been seen that, after all but the last stage of the FFT, outputs must be scaled and converted to polynomials which are appropriate for the next stage (i.e. multiplication by twiddle factors, followed by another 4-point DFT). The error introduced by scaling is, of course, dependent on the scaling algorithm employed. There are many techniques available.

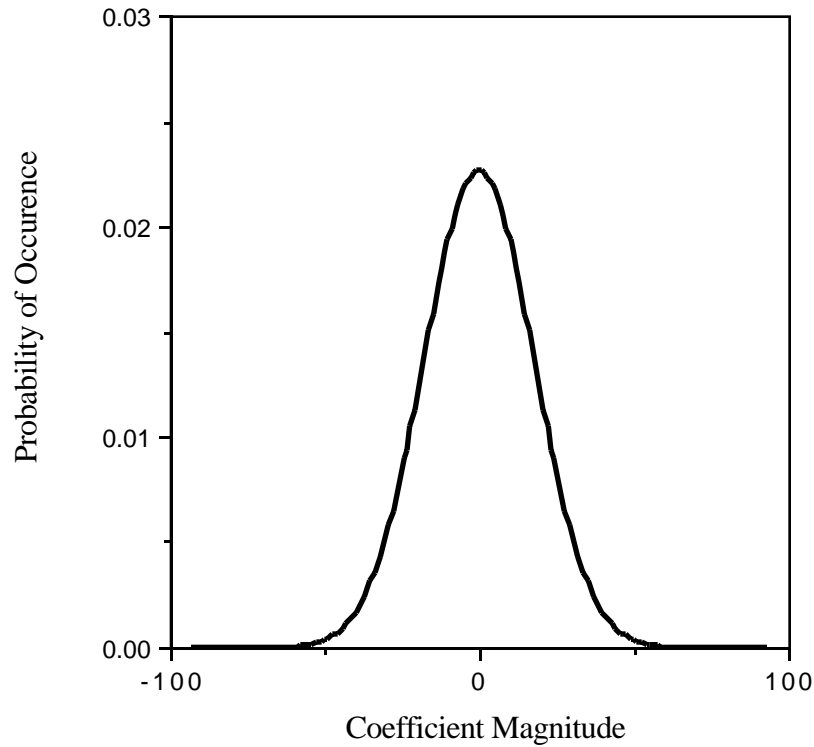


Fig. 2: Statistical distribution for  $c_{1111}$

Assuming twiddle factors to have  $B$  bits, and assuming a number growth factor of 4, the desired scaling factor is seen to be  $2^{B+2}$ .

The simplest technique for conversion of the output polynomials into input polynomials, which have been reduced by an appropriate scaling factor, is to convert the polynomials into binary integers, scale, and convert the results to input polynomials. This necessitates a large binary adder. The number of terms to be added may, however, be reduced by adding together the coefficients of the same power of two (i.e. coefficients of monomials which represent the same power of 2, e.g.  $W^2$  and  $X$ , each of which represents 4) and then treating this result as one coefficient. This permits the additions to be done using modular arithmetic, which has the combined effect of decreasing the number of additions to be performed and also decreases the number of values which the CRT must process. It also has the salutary effect of keeping down hardware overhead. The most efficient scaling algorithm we have found is given below:

**Scaling Algorithm:** Suppose that the desired scaling factor is  $2^s$  (in practice we have  $s = 18$ , since  $B = 16$ ). A low error method of applying this scale factor is to use the recursive relationship:

$$\tilde{C}_i = \frac{\tilde{C}_{i-1}}{2^\gamma} + C_i; \quad \gamma = \begin{cases} 1 & 0 \leq i < s \\ 0 & i \geq s \end{cases} \quad (5.4)$$

Using this method, the coefficients corresponding to the first  $s$  powers of 2 are processed using 5-bit (or fewer) additions. Absolute error, though roundoff errors occur  $s$  times, is limited to:

$$\sum_{i=1}^s 2^{-i} = \frac{1 - 2^{-s}}{1 - 2^{-1}} - 1 < 1 \quad (5.5)$$

Further hardware simplification is possible, without introducing significant error, by ignoring the lowest order terms.

## 5.4 Mean squared error

To compute mean squared error, we observe that after the first stage of scaling there is a probability  $\frac{1}{4}$  of an error of  $2^{-s}$ , a probability  $\frac{1}{4}$  of an error  $-2^{-s}$ , and a probability  $\frac{1}{2}$  of an error of 0. Hence the mean squared error after the first stage of scaling is

$$e(0) = \frac{1}{4}[2^{-s}]^2 + \frac{1}{4}[-2^{-s}]^2 = 2^{-1-2s} \quad (5.6)$$

After the second stage of scaling another error is introduced, whose mean squared error is:

$$e(1) = 2^{1-2s} \quad (5.7)$$

Continuing, we obtain  $e(s-1) = 2^{-3}$ . Since each term being added in the scaling operation is offset from the previous term by one bit, the errors calculated above are all independent. Thus the total mean squared scaling error is the sum of the individual mean squared errors, and the total mean squared scaling error is  $2^{-1-2s} + 2^{1-2s} + 2^{3-2s} + \dots + 2^{-3} < \frac{1}{6}$ . Accounting for both real and imaginary parts, we see that the total mean squared scaling error is  $< \frac{1}{3}$ . Thus algorithm III produces a mean squared error less than twice that of algorithm I, and hence less than twice the minimum possible mean squared error.

## 6. VLSI Implementation

For the small ring computations allowed by our technique, we only require to build arbitrary switching blocks with a maximum of 6 logic inputs. The 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 [8].

### 6.1 Minimized ROM Structures (Switching Trees)

In an attempt to optimize current designs on silicon, we have examined the construction of such small address space 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 two simple graph theory rules [14]. In doing this we do not invoke any concepts from boolean algebra which may not yield the best transistor configurations (including PLA configurations).

Restricting the trees to be n-channel blocks and evaluating only a single node, we can build massively pipelined systems, where every evaluation node is pipelined. A recently introduced true single phase clocking system [13], 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 [2].

## 6.2 Embedded Single Phase Clocked Latch

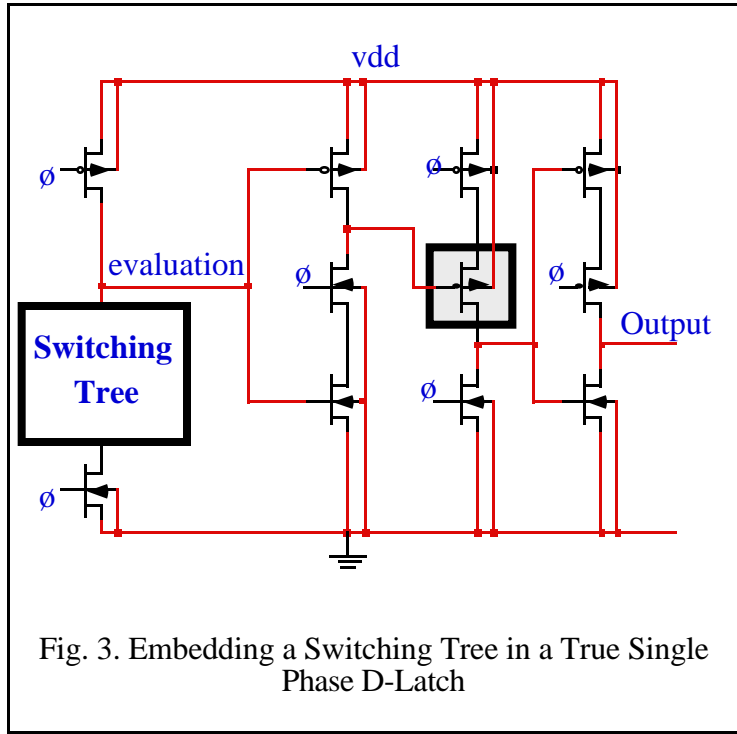


Fig. 3. Embedding a Switching Tree in a True Single Phase D-Latch

The complete single phase clocked latch, with embedded switching tree, is shown in Fig. 3. 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 fabricated 6 high trees that function at pipeline rates of 40MHz for a  $3\mu$  CMOS

double metal, p-well process.

## 6.4 Results

As an illustration of the application of the switching tree method we have used our CAD package, *WOODCHUCK* [15], to minimize the full tree structure of a modulo 7 multiplier (Fig. 4).

*WOODCHUCK* minimizes merged multiple output trees, including the use of don't care states for pattern matching reduction. This is particularly appropriate for this example since there are several

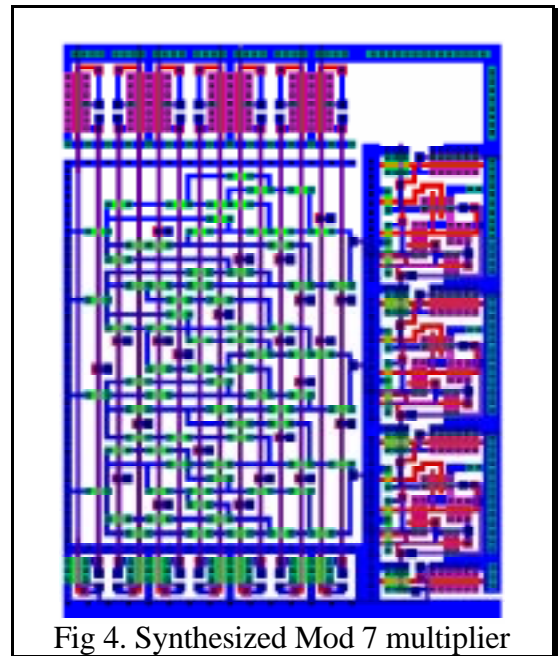


Fig 4. Synthesized Mod 7 multiplier

don't care states arising from the fact that some output states are not reachable in a modulo  $m$  computer where  $2^i < m < 2^{i+1}$ .

The size of the multiplier is  $528\mu \times 662\mu = 350 \times 10^3 \mu^2$  and if we allow a 10% overhead in routing (remember that the array is very regular) we are able to fit about 260 such cells on a  $1\text{cm}^2$  die. With a submicron (say  $0.8\mu$ ) process, this number will increase to over 2500 cells, and we expect pipeline rates in excess of 150MHz; well within speeds for applications such as Radar, HDTV etc.

## 7. Comments and Conclusions

In this paper we have discussed a new polynomial ring mapping technique which allows large dynamic range computations to be performed using massively parallel small finite ring computational elements. We have presented complete forward and inverse mapping details, along with suitable scaling procedures and the application of redundant binary representations; the computation of a radix-4 FFT has been used to illustrate the procedure. A complete quantization analysis, including coefficient growth peculiar to this mapping strategy, has been developed.

The technique allows direct mapping of bits of either a purely real or multiplexed bit coded complex number to a set of independent rings, defined by the smallest usable odd relatively prime moduli of 3, 5 and 7. Although the use of such small rings in a traditional *RNS* system would yield an inadequate computational dynamic range, our new technique allows usefully large dynamic range computations with such moduli.

A suitable *VLSI* implementation procedure, using the recently introduced *Switching Tree* approach, has been illustrated with the synthesis of a complete Mod 7 multiplier. Fabricated test cells operate at  $40\text{MHz}$ .

An important VLSI architectural point is the massive replication required for the small rings. We have to remember that a complete multiplication is performed within a single pipeline cycle using only 6-transistor high trees. Essentially we have changed the VLSI footprint of the computational elements from the roughly square footprint of standard binary multipliers to a narrow-long rectangular footprint. Since the narrow dimension is in the temporal direction, we achieve high speed, low latency implementations. Instead of the integrated two-dimensional data flow experienced with standard binary arithmetic elements (associated with carry propagation), our architecture only communicates across the dynamic range at relatively widely spaced intervals (scaling and conversion). Between these points we have linear independent pipelines using only 3-bit variables. At the conversion points we effectively have a corner turning procedure, where the computations across the entire dynamic range are computed; these computations are also linear pipelines. Testability, and fault tolerance advantages of such an architecture are not to be dismissed, particularly in critical applications; the architecture is ideally suited to current density ULSI fabrication processes.

## References

1. Blahut, R. E., "Fast Algorithms for Digital Signal Processing", 1985, Addison-Wesley, Reading, Massachusetts.
2. 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.
3. Godement, R., "Algebra", 1968, Paris: Hermann; Engl. transl. Boston: Houghton Mifflin.
4. Jullien, G. A. "Bit-Level Systolic Arrays for High Speed DSP." *Advances in VLSI Signal Processing*. 1991 Ablex Publishing.
5. 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, 189-208, 1989.

6. Jullien, G. A., M. Taheri, S. Bandyopadhyay and W. C. Miller. "A Low-Overhead Scheme for Testing a Bit Level Finite Ring Systolic Array." *Journal of VLSI Signal Processing*. 1.4, 1989.
7. Rabiner, L. R. and B. Gold. "Theory and Application of Digital Signal Processing." 1975 Prentice-Hall, Englewood Cliffs, N.J.
8. Soderstrand, M. A., W. K. Jenkins, G. A. Jullien and F. J. Taylor. Residue Number System Arithmetic: Modern Applications in Digital Signal Processing. 1986.
9. 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.
10. Tseng, B.-D., G. A. Jullien and W. C. Miller. "Implementation of FFT Structures Using the Residue Number System." *IEEE Trans. Comp.* C-28, 831-844, 1979.
12. Wigley, N. M. and G. A. Jullien. "On Moduli Replication for Residue Arithmetic Computations of Complex Inner Products." *IEEE Trans. Comp.* 39, 1065-1076, 1990.
13. Yuan, J. and C. Svensson. "High-Speed CMOS Circuit Technique." *IEEE. J. Solid-State Circuits*. vol. 24, pp. 62-70, 1989.
14. Zhang, D., G. A. Jullien, W. C. Miller and E. Swartzlander. Arithmetic for Digital Neural Networks. Proceedings of the 10th International Symposium on Computer Arithmetic. 58-63, 1991.
15. G.A. Jullien, W.C. Miller, R. Grondin, Z. Wang, D. Zhang, L. Del Pup, S. Bizzan, 1992, "WoodChuck: A Low-Level Synthesizer for Dynamic Pipelined DSP Arithmetic Logic Blocks." Proceedings of the 1992 IEEE Int. Symp. on Circuits and Systems, (*Invited*) 1, pp. 176-179.

## Acknowledgements

The authors acknowledge financial support from the Natural Sciences and Engineering Research Council of Canada, the Micronet Network of Centres of Excellence, and the fabrication and equipment loan programme of the Canadian Microelectronics Corporation, to carry out this research work.

## Author Affiliation

N. Wigley, G.A. Jullien, D. Reaume  
VLSI Research Group, University of Windsor  
Windsor, Ontario, Canada N9B 3P4

## Contact

Dr G.A. Jullien  
Department of Electrical Engineering  
University of Windsor  
Windsor, Ontario, Canada N9B 3P4  
Tel (519) 253-4232 Ext. 2574  
FAX (519) 973-7062  
e-mail: [jullien@engn.uwindsor.ca](mailto:jullien@engn.uwindsor.ca)

# *Figures*



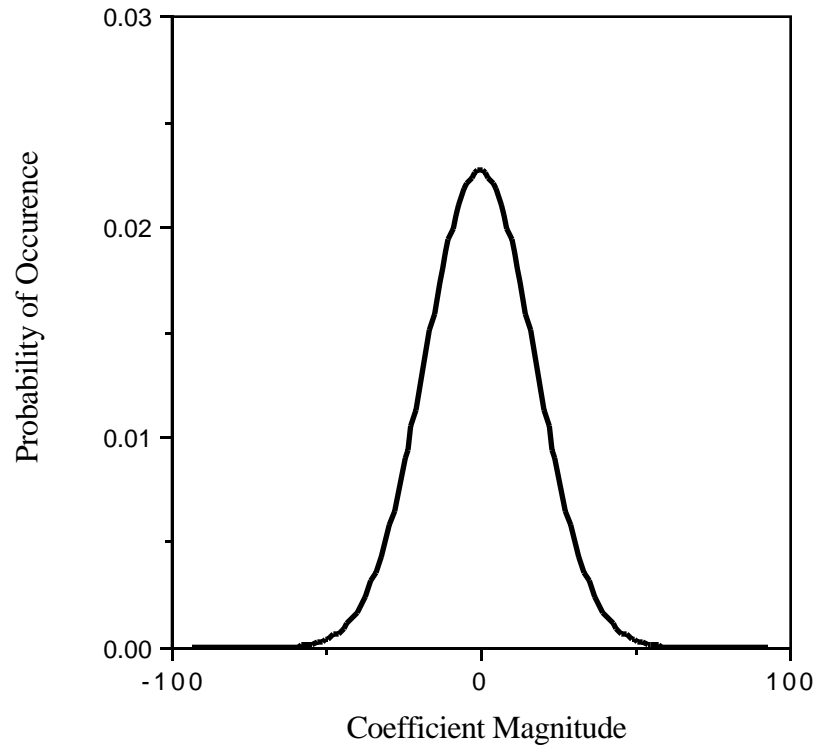


Fig. 2: Statistical distribution for  $c_{1111}$

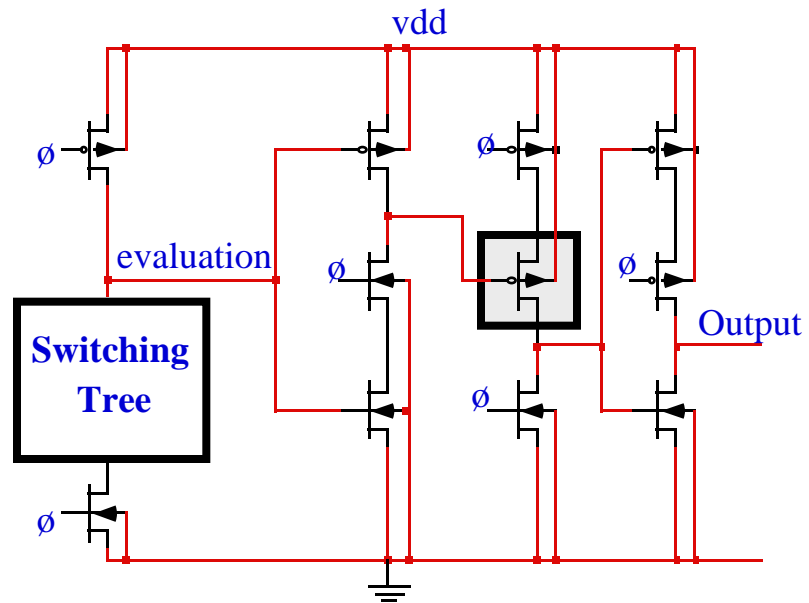


Fig. 3. Embedding a Switching Tree in a True Single Phase D-Latch

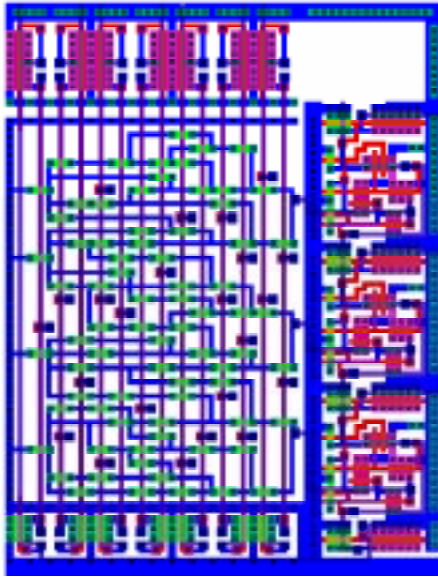


Fig 4. Synthesized Mod 7 multiplier