Small Moduli Replications in the MRRNS

N. Wigley, G.A. Jullien, D. Reaume, W.C. Miller

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

Abstract

The use of finite polynomial rings to design algorithms for the processing of digital signals has received considerable attention in recent years. The authors have recently introduced a method which utilizes the bit patterns of the input data to directly compute the polynomial coefficients; this technique is straightforward and preserves, in part, some of the magnitude information. The new magnitude information coding allows different scaling and conversion algorithms than those required for standard RNS decoding. This paper discusses new polynomial mapping strategies involving replications of *very small* rings modulo 3, 5 and 7 including a scaling and conversion algorithm for such a mapping.

1. Introduction

Finite rings can offer considerable advantages over binary arithmetic in performing integer arithmetic. The most visible use of such finite rings 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) [6]. 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 [3].

If M is a positive integer which factors as $M = \prod m_i$, with the $\{m_i\}$ all pairwise relatively prime, then an isomorphism between the ring Z_M and the direct-product ring:

$$Z_{m_1} \times Z_{m_2} \times \ldots \times Z_{m_K}$$

is well-known (the inverse isomorphism is known as the Chinese Remainder Theorem (CRT)). Any computation which can be embedded into Z_M can thus be considered as a computation in the direct-product ring. In the latter, the computations are performed simultaneously and independently, which leads to a great simplification in hardware and allows the pipelining of data through each of the arithmetic units.

Some disadvantages of the RNS are:

1. The necessarily large size imposed on the modulus M in order to obtain the embedding of the calculation (M must accommodate the full dynamic range of the calculation, even though the smaller moduli m_k need not).

2. The residues of the input data must be computed (mod m_k) for each k.

3. The resulting answers must be assembled, to yield the correct answer, by means of the CRT or the equivalent Mixed Radix Method.

4. Scaling can pose considerable difficulties, partly because magnitude information has been completely lost in the finite-ring representation.

An optional method of proceeding is based on the use of finite polynomial rings, where the polynomials have coefficients which are considered in Z_M, and the indeterminates are used to indicate bit information (or, occasionally, the complex unit). This method is known as the *Modulus Replication RNS* [10], or *MRRNS*, and consists of encoding integer data as polynomials in several indeterminates[9].

2. The MRRNS technique

In the *MRRNS* technique the integer data are mapped to a polynomial ring. This idea has been explored by other authors [1][2][7], but the work normally assumes fairly complicated mapping strategies between overspanned complex numbers and polynomial coefficients. In the original *MRRNS* technique, the data are first rewritten as polynomials in some fixed radix 2^{β}. For theoretical purposes we use the indeterminate X in place of this radix, thus enabling us to represent the data as polynomials in X. The coefficients of these polynomials are then integers which are smaller in magnitude than 2^{β}. This mapping (which is trivial) is then followed by a mapping to the direct-product ring $Z_m \times Z_m \times \ldots \times Z_m$. The computations can now be carried out using independent linear pipelines, each computing over the same ring, at the end of which the inverse of the original mapping is applied. The forward and inverse mappings (to and from the direct product ring) are themselves linear pipelines, and so we have rendered the complete system as a set of linear pipelines operating over a finite ring, modulo m. The output of the inverse mapping stage is a redundantly coded number (here the redundancy arises because of the magnitude overlap of the polynomial coefficients) which is converted back into a non-redundant representation using a combination of scaling and binary addition. For complex data we may use the *QRNS* mapping strategy [4] prior to the *MRRNS* mapping, and invert the *QRNS* mapping after the inverse *MRRNS* mapping. In this way complex computations are mapped to the direct product ring, again yielding to linear pipeline implementation.

A disadvantage of the technique is a large redundancy in the finite-ring representation of data, which results in a need for considerable replication of the hardware used to perform the computations. This replication is ameliorated, however, by the *repeated* use of very small moduli, allowing for the simple design of computational hardware. Moreover, linear pipelining and its attendant advantages in fault tolerance, testing and skewed clocking, make fabrication using wafer scale integration look encouraging. In this paper we use a different mapping strategy in which the data are written as polynomials in several variables, each representing a different power of 2. This will have the effect of increasing the dynamic range of the calculation, though it will increase the redundancy of the computational hardware. A major advantage of this scheme is that it allows us to utilize very small moduli, namely 3, 5, and 7. Thus all the data will be mapped to polynomials whose coefficients lie in the ring Z_{105}, and can thus be treated as elements of the three rings Z_3, Z_5, and Z_7. This in turn allows a very simple design of the hardware necessary for the computations within these small rings. Clearly the method will also work for larger moduli; in fact, the number of parallel channels will drastically reduce. Our concern in this paper, however, is to show that residue calculations over very small rings can still perform large dynamic range arithmetic. By way of example, we will use the *MRRNS* technique for the computation of a 1024-point *FFT* using a multiplexed radix 4 computational element and moduli 3, 5, and 7.

3. 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:

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

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 ±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 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. This will result in 81 parallel computations over small ring moduli (although the mapping is shown as 3-dimensional in Fig. 1, the implementation uses independent channels). An inner product on the bit mappings results in the 81 inputs to the computational inner product required by the algorithm.

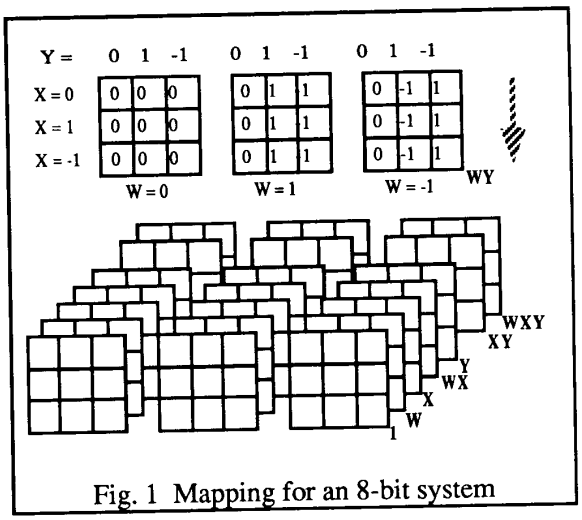


Fig. 1 Mapping for an 8-bit system

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. 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. 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 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.

4. Scaling and Decoding

Each coefficient in the inverse mapped polynomial represents a weighting by a specific power of 2. Table 1 shows the monomial equivalences for the first seven powers of 2 weightings.

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 1

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.

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:

$$\tilde{C}_i = \frac{\tilde{C}_{i-1}}{2^{\gamma}} + C_i; \quad \gamma = \begin{cases} 1 \text{ for } 0 \le i \le s \\ 0 \text{ for } i \ge s \end{cases}$$

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 recovery array is more clearly evident in Fig. 2. The total number of bits of the full dynamic range conversion is B, the number of scaling bits is s. The ⬬ 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.

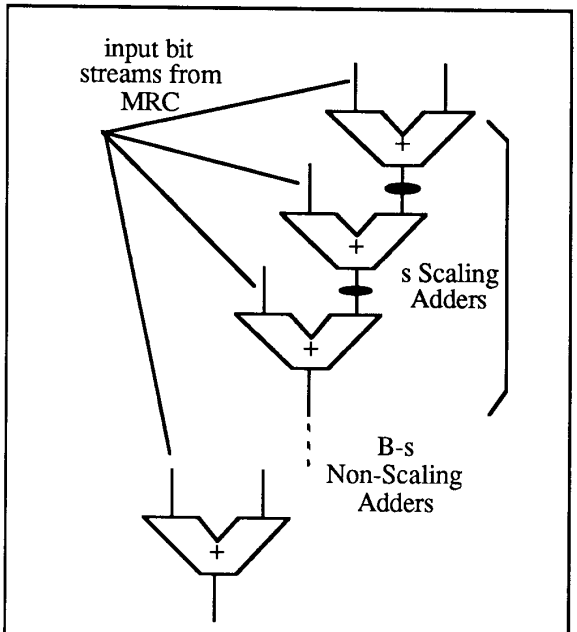
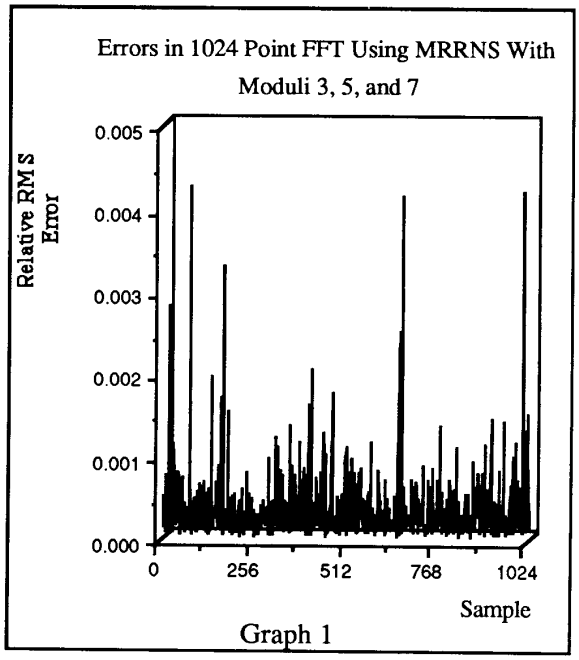


Fig. 2 Scaling and Binary Conversion

5. Experimental Error Analysis

For the purpose of measuring scaling error we have simulated the *MRRNS* 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 are given in Graph 1. The average relative root mean square error (*RRMS* error) was 1.96×10^{-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 1.08×10^{-3} *RRMS* error for the *QRNS* system versus a 1.96×10^{-4} *RRMS* error for the *MRRNS* 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 [8] (compared to fixed multiplication) and the very large overhead, and awkward structure, associated with converting 6-bit modulus systems [3] make the *MRRNS* technique much more attractive.



Graph 1

6. VLSI Implementation Considerations

Our preferred approach to the implementation is to use switching trees implementing dynamic pipelined logic. Because of space limitations we cannot discuss the concept in any detail. The reader is directed to [11] for an introduction to some of the basic concepts of dynamic logic. In brief, dynamic logic differs from static logic in that the parasitic capacitance of the evaluation (logic output) node is precharged to a logic '1' level. After pre-charge, a network of transistors is used to evaluate the node. For our purposes, if the network provides a path to ground for the evaluation node, then this is considered to be a logic '1' output. If the network provides an open circuit between the evaluation node and ground, this is considered to be a logic '0'. The logic function of the transistor network is therefore to discharge the evaluation node for every minterm

that is included in the boolean algebra expression of the switching function. The basic concept behind switching tree cells is to implement the transistor network as a look-up table, but to construct the table as a minimized tree. The minimization is based on the electrical characteristics required of the switching circuit (i.e. the discharge and charge sharing characteristics in evaluation and pre-charge, respectively) and does not involve either the usual Boolean algebra minimization or the concept of logic gate primitives. Because we will minimize the table directly, an initial choice has to be made regarding the dimensionality of the table decomposition (it is to be noted here that a normal ROM is decomposed into 2-dimensions - rows and columns).

The dimensionality of the decomposition directly leads to the number of series transistors in the switching circuit, and hence to the speed performance. The decoders represent the first stage of a two-stage circuit implementation process (decode, look-up) and their complexity will determine whether the two stages are to be treated as a single pipelined combinational logic circuit, or as a two-stage pipeline. Clearly the former is preferable, since the latency will be doubled if a two-stage pipeline look-up has to be performed. This determination is decided purely on the complexity of the decoders. For n input lines the decoders of the *n*-dimensional table are simply inverters and will not require an additional pipeline stage. It is interesting to note that the 1-dimensional table is multiplexer logic, and the n-dimensional table is a binary decision-tree.

We have determined that a discharge chain with 6 transistors in series yields dynamic pipeline speeds in excess of 50MHz for a conservative 3μm *CMOS* process. Since 6-inputs corresponds to the maximum general logic block for two 3-bit ring values, we can use a binary decision tree, and reduce the decoders to simple logic inverters. We therefore construct a binary tree of transistors in 6 variables and program the bottom of the tree according to the truth table entries (place a transistor for a '1' and remove the transistor for a '0'). The true and complement inputs to the tree are connected to the gates of the transistors such that only one path is available for a given input word. The presence, or absence, of a bottom transistor will determine the state of the evaluation node.

6.1 A 3-bit finite ring multiplier

As an example we show the minimized switching tree for bit zero of a mod 7 finite ring multiplier. The tree is minimized using simple rules and the final minimized tree is shown in Fig. 3. The tree has been minimized from the full 6-input binary tree by applying 3 simple graphical rules [12]. This results in some of the transistors being replaced by wire connections.

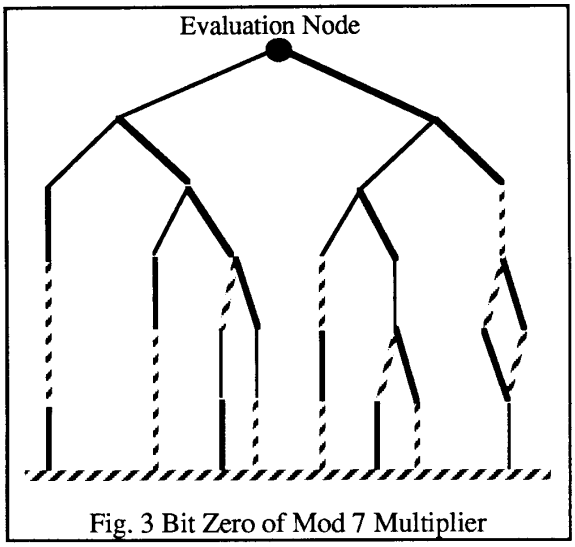


Fig. 3 Bit Zero of Mod 7 Multiplier

The thin lines indicate paths taken when the logic input at that particular level is a '0', the bold lines indicate paths taken when the logic level is a '1'. A wire is shown as a hatched line. Each level corresponds to one dimension of the table (accessed by one of the logic inputs) with the levels ordered hierarchically in correspondence with the logic input ordering on the truth table. It is important to note that our starting point for the minimization is a diagram that can be directly implemented in silicon by simply transforming paths into transistors. Minimization procedures will therefore be directly mappable to silicon. In the case of finite ring circuits there will always be don't care states providing the modulus is not a power of 2. We use these states to advantage in minimizing the tree, and have reduced the maximum tree length to 5.

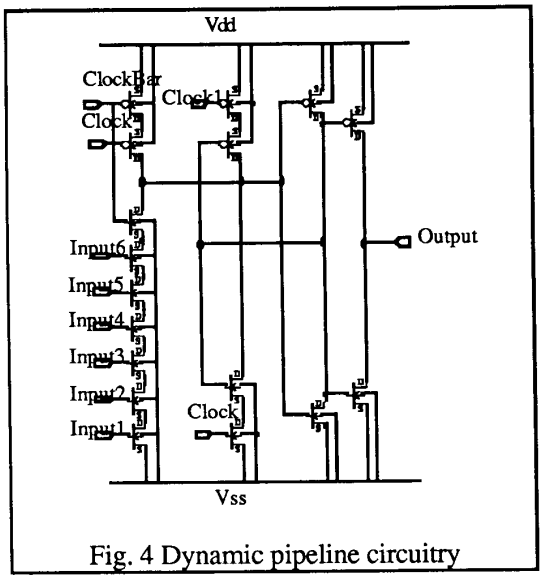


Fig. 4 Dynamic pipeline circuitry

For a linear pipeline implementation, the tree can be simply embedded in a dynamic pipeline structure [5]. Our preferred technique is to use an integrated evaluate/latch structure and to connect the evaluate node to a restoring latch. This eliminates the need for any charge sharing pull-up transistors. The pipeline structure with a single 6-transistor switching path is shown in Fig. 4. In the final implementation the 6-transistor series path is replaced by the switching tree. The electrical characteristics of the tree will be virtually the same as those of the series path shown.

The transistor circuit, and a metal only layout of the tree, implemented in a 3μm *CMOS* double metal technology are shown in Fig. 5. The tree replaces the single 6-transistor path shown in Fig. 4. A complete multiplier, including pipeline circuitry, and appropriate drivers has been sent for fabrication. The layout is shown in Fig. 6 along with SPICE simulation results for a 50*MHz* clock input. The 3 switching trees are at the bottom with pipeline circuitry and drivers placed above.

7. Conclusions

This paper has discussed mapping, scaling and conversion processes using a new mapping strategy for the modulus replication *RNS* (*MRRNS*).

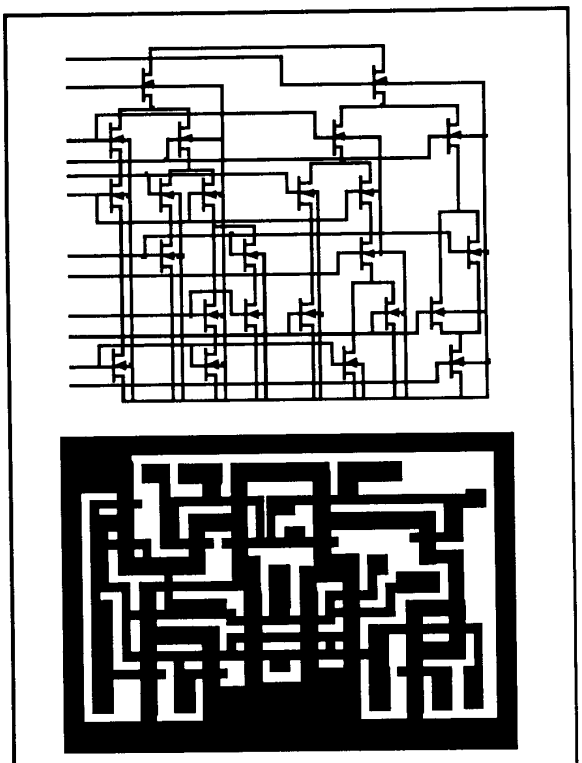


Fig. 5 Schematic and layout for bit 0 of the mod 7 multiplier

The strategy allows direct mapping of bits of either a purely real or multiplexed bit coded complex number to a set of independent rings, defined by moduli 3, 5 and 7. Although the use of such small rings in a traditional *RNS* system would yield an inadequate computational dynamic range, the *MRRNS* technique has been shown to be superior to a large *QRNS* system operating with a computational dynamic range of over 27 bits. A classical radix-4 implementation of a 1024 *FFT* was used for the comparison. The scaling and conversion procedure has been shown to be a set of finite ring calculations followed by an array of ordinary binary adders. The *VLSI* implementation of the most complex finite ring circuit required (a Mod 7 multiplier) has been shown to be easily implemented using the switching tree approach, and mask extracted simulations at 50*MHz* have been used to demonstrate the embedding of the switching trees in a dynamic pipeline/evaluate circuit with restoring latch.

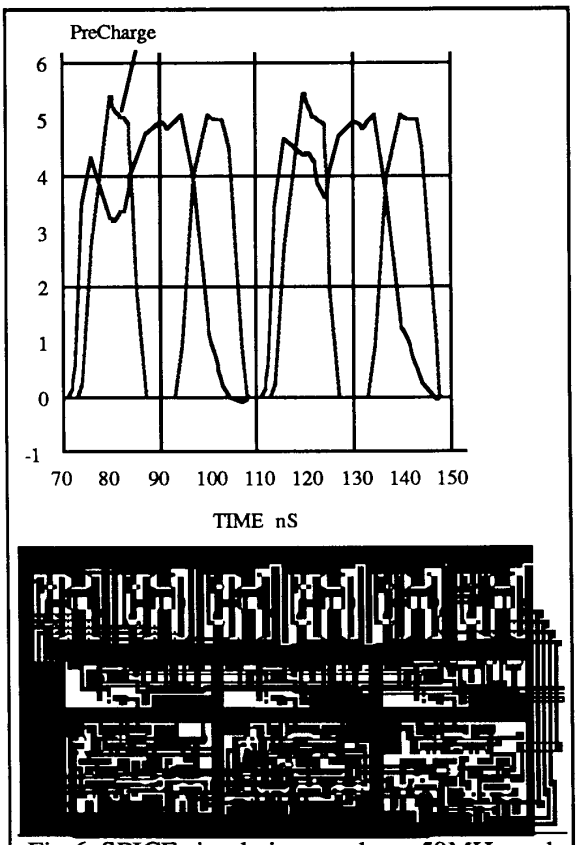


Fig.6 SPICE simulation results at 50MHz and Mod 7 multiplier layout

8. References

[1] Cozzens, J. H. and L. A. Finkelstein. "Computing the Discrete Fourier Transform Using Residue Number Systems in a Ring of Algebraic Integers." IEEE Trans. Inf. Th. **IT-31**: 580-587, 1985.

[2] Games, R. A. "An Algorithm for Complex Approximations in Z[e2πi/8]." IEEE Trans. Inform. Th. **IT-32**: 603-607, 1986.

[3] 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. **1**(3): 189-208, 1989.

[4] Jullien, G. A., R. Krishnan and W. C. Miller. "Complex Digital Signal Processing Over Finite Rings." IEEE Trans. on Circuits and Systems. **CAS-34**(No. 4 , Spe*cial Is*sue (Invited)): 365-377, 1987.

[5] Jullien, G. A. and W. C. Miller. Improved Cellular Structures for Bit-Steered ROM Finite Ring Systolic Arrays. Proceedings of the 1990 International Conference on Circuits and Systems, (Invited). , May, pp. 1414-1417., 1990.

[6] Soderstrand, M. A., W. K. Jenkins, G. A. Jullien and F. J. Taylor. Residue Number System Arithmetic: Modern Applications in Digital Signal Processing. 1986.

[7] Stouraitis, T. and A. Skavantzos. Parallel Decomposition of Complex Multipliers. 22nd. Asilomar Conf. Circ. Sys. Comp. 379-383, 1988.

[8] 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**(3): 504-512, 1988.

[9] Wigley, N. and G. A. Jullien. Array Processing on Finite Polynomial Rings. Proceedings of the International Conference on Application Specific Array Processors. 284-295, 1990.

[10] Wigley, N. M. and G. A. Jullien. "On Moduli Replication for Residue Arithmetic Computations of Complex Inner Products." IEEE Trans. Comp. **39**(No. 8): 1065-1076, 1990.

[11] Weste, N. and Eshraghian, K. "Principles of CMOS VLSI Design: A Systems Perspective." Addison-Wesley, 1985.

[12] Jullien, G.A., Zhang, D., Miller, W.C., DelPup, L. and Grondin, R. "Designing Complex Dynamic Logic Blocks Using Switching Trees." VLSI'91, submitted.

Acknowledgements

The authors acknowledge financial support from the Natural Sciences and Engineering Research Council of Canada to carry out this research work.