

Integer Mapping Architectures for the Polynomial Ring Engine

S. S. Bizzan, G. A. Jullien, N.M. Wigley and W.C. Miller

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

Abstract

A novel finite polynomial ring structure, for mapping inner product computations to parallel independent ring computations over 3-bit moduli, has been introduced recently [10]. The main algorithmic computation architecture can be implemented using well established systolic array mapping principles [6], and a project to construct a Polynomial Ring Engine (PRE) is underway to exploit the VLSI implementation properties of such computations. This paper introduces a semi-systolic architecture for the input and output conversion mappings, that are required in the engine.

We show that the entire mapping procedure can be carried out with pipelined 6-input logic blocks and small, fast, binary adders. In addition, the paper discusses CMOS implementation techniques for the 6-input pipelined blocks, and illustrates the design procedure with results from a recently completed module generator.

1: Introduction:

Number theoretic architectures have traditionally been based on the Residue Number System (RNS) [7], but the disadvantages of RNS techniques (non-homogeneous data conversion architectures) outweigh the advantages of carry free computation. A recently introduced approach, based on a polynomial ring mapping strategy, eliminates many of the problems associated with conversion architectures from rings with disparate moduli [10]. Unlike the recently introduced algebraic integer [3] or PRNS approaches, the new technique allows simple, error-free, mapping of incoming integer streams, and homogeneous conversion architectures at the output. The main body of the computation is performed in identically replicated linear bit-level pipelines; this has important ramifications in terms of fault tolerance and testability when implemented in dense technologies, such as WSI and ULSI.

This paper is directly concerned with the implementation of conversion strategies associated with the polynomial ring mappings. We first present an

overview of the mapping technique, discuss architectural details of a semi-systolic mapping architecture and finally introduce a low-level dynamic pipelined CMOS synthesizer (module generator) that builds the special bit-level cells required for the small ring computations required by the conversion procedure.

2: Polynomial Ring Mapping

The PRE mapping strategy allows computations on Gaussian integers to be implemented in direct product rings with simple mapping procedures between signed digit binary representations and a direct product of identical 3-bit modulus rings. The theoretical mapping procedure requires several intermediate rings to be defined, and these are described in this section.

2.1: Encoding and Decoding of Integers

The integer mappings required by the PRE are shown in Fig. 1. The figure shows the progression of ring mappings required; the rings are denoted by boxes, and the mappings by arrows.

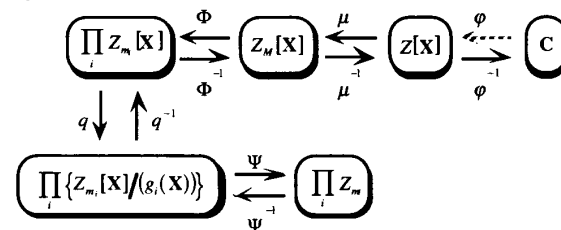


Fig 1. The Rings and Homomorphisms

For simplicity in the diagram we have represented the set of variables X_1, X_2, \dots, X_n , $2^i < m < 2^{i+1}$, with the symbol \mathbf{X} . Encoding of the data begins in the upper right at the complex numbers \mathbb{C} (Gaussian integers) and continues counter-clockwise around the diagram to the lower right. The algorithmic computation is performed in

the direct product ring $\prod_i Z_{m_i}$; the answers are then decoded in the reverse direction.

2.2 Representing integers as polynomials (the map ϕ)

ϕ^{-1} is a homomorphism, it preserves sums and products in the required algorithmic DSP computation. It is sufficient to find the polynomial in $Z[\mathbf{X}]$ which represents the final computation; this is the forward map, ϕ (this is not a homomorphism, and is shown hatched on Fig. 1).

2.3 The maps μ and Φ

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. 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 . The encoding then continues from $Z_M[\mathbf{X}]$ to the direct product ring $\prod_i Z_{m_i}[X]$. The map Φ reduces

coefficients with respect to the moduli $\{m_i\}$. This is equivalent to using a small RNS system to represent integers (for computation) over the replicated rings, mod M , and allows the use of very small moduli. For example, $105 = 3 \times 5 \times 7$, is an example of the smallest 3 modulus decomposition possible with this technique

2.4 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. It is clear that there is no actual 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 Ψ . We arrange the mapping of bits to indeterminates such that the output polynomial from a single product has degree 2. It therefore suffices to have ideals generated from polynomials of degree 3; such a polynomial is $g_i(T) = T(T^2 - 1)$.

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 $i \neq j$ [10]). We therefore have a way of mapping the complex operator that is indistinguishable from mapping of the redundant data sample digits.

2.5 The map Ψ

The map Ψ 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. This assumes that our algorithm only allows one cascaded multiplication between conversion mappings. Clearly inner products (e.g. FIR filters) have this property, but so do a number of other algorithms (e.g. [9]).

With the moduli set $\{3,5,7\}$, we can represent the components of Ψ with a 9×9 matrix for the modulus $m = 3$, and as a 6×6 matrix for the modulus $m = 5$. With $m = 3$, the polynomial $M A_1 + A_2 X + A_3 X^2 + T(A_4 + A_5 X + A_6 X^2) + T^2(A_7 + A_8 X + A_9 X^2)$ 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, and this fact is highly beneficial in the VLSI layout problem, where we may sometimes deal with four or more variables.

2.6 The DSP Algorithm Computation

Once the data have been converted to elements of this last ring, the DSP algorithmic computation is performed

in each component of the direct product ring $\prod_i Z_{m_i}$. The results are then mapped back to \mathbb{C} .

2.7 Reversing the maps Ψ and q

The map Ψ is an isomorphism, so Ψ^{-1} poses no problem. The matrix components of Ψ^{-1} can be obtained by inverting the matrices Γ and Γ_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.

2.8 Reversing the maps Φ , μ and φ

The classical Chinese Remainder Theorem (CRT) assures that the map Φ^{-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 μ has in general no inverse. Since μ 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 μ^{-1} . This

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

3: Conversion Architecture

It is clear, from section 2, that the main computation for the forward map is the tensor product associated with the Ψ mapping. The other mappings are either formalisms or simple wiring interconnections. The inverse map also invokes a tensor product with an inverse matrix, but also requires the summation of result polynomial coefficients with their appropriate bit shifts (or complex operator separation) based on the indeterminate powers associated with each coefficient.

3.1 General Architecture

The overall mapping and computation architecture is shown in Fig. 2 for a complex input data stream. This diagram makes it clear that the tensor product polynomial mapping is implemented over 3-bit data streams, which allows the use of small 6-input switching blocks as the general computing unit. For this example, we have assumed that the DSP algorithmic computation can be implemented with linear pipelines; this is certainly the case with inner product computations (e.g. [8]). A brief description of the blocks, and their computational requirements, is given in Table 1.

Table 1 Architectural requirements for forward and reverse mapping

Block	Description	Computing Blocks
Binary \rightarrow Polynomial	Map weight of redundant binary digits to coefficients of polynomial indeterminates	Simple interconnection
Modulo Reduction	Reduce coefficients by $\{m_i\}$	No action since coefficients are chosen to be less than $\{m_i\}$
Polynomial Mapping	Forward mapping tensor product on roots $\{-1, 0, +1\}_{m_i}$	Pipelined 3-bit weighted modulus addition (6-bit input 3-bit output blocks)
Data Processing	DSP algorithmic computation	Systolic array based on general function 3-bit modulo computations (6-bit input 3-bit output blocks)
Reverse Polynomial	Reverse mapping tensor product	Pipelined 3-bit weighted modulus addition
CRT	3-modulus CRT (simple mixed radix system)	Pipelined 6-bit input 3-bit output blocks.
Polynomial \rightarrow Binary	Mapping to weighted binary (shifts) and the complex operator (separation)	Pipelined shifted binary adders and subtractors.

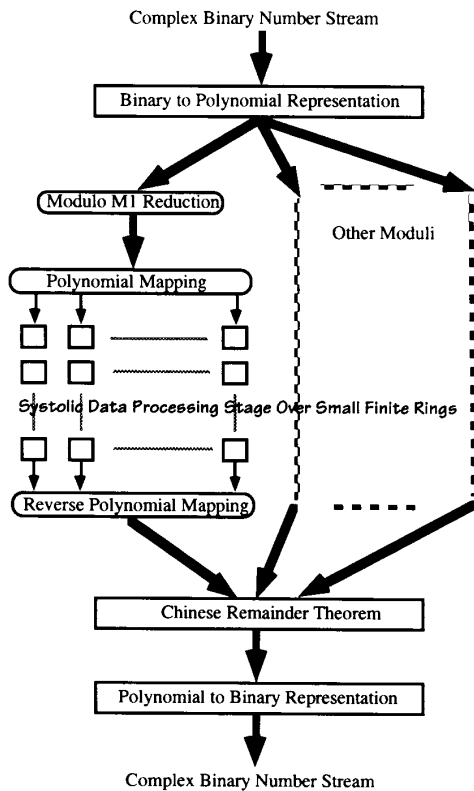


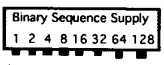




Fig. 2 PRE Computational Architecture

3.2 Architectural Details..An Example

Fig. 3 shows an example architecture that uses two indeterminates; one corresponds to a binary weight of 2 and the other represents the complex operator j . The RNS representation is based on the residue set $\{3, 5, 7\}$ which, as discussed above, keeps the data representation to 3 bits throughout the architecture. The dotted line indicates the placement of the DSP algorithmic computation (not included in this simulation) and the boxes contain the two modular structures for the forward and reverse mapping portions.

Pipeline latches are shown as  and general computational blocks (weighted modulo adders) as . Input blocks shown as  are used to generate the real and imaginary binary input data for the simulation. 3-bit 3-modulus CRT behavioural model block is shown as  and can be implemented by pipelined 6-bit input 3-bit output blocks using simple

mixed radix system. The output converters are represented with a behavioural model within the block  utilizing fast binary adders. Note that after the CRT block we use a stage of binary subtractors to separate the coefficients of $j^2 = -1$ from the real part polynomial coefficients. It is at this stage where we introduce a sign bit for the real part of the complex number output.

There are a total of 9 replications of each of the 3 residue rings (27 channels in total). Note that each of the three moduli computational channels are completely independent until the CRT stage is reached. The number of stages in both the forward and reverse mapping is equal to the number of indeterminates (in this example two). The reverse mapping illustrates the need for a full 2nd order polynomial inversion whereas the forward mapping is simplified because the input polynomial has order one. The forward basic tensor product structure for both the forward and inverse maps are contained in the shaded areas on Fig.3. The weighted modulo adder blocks will actually take no more hardware than the binary adders since we implement the blocks as minimized look-up tables (see section 4). The simulator includes a weight option within each modulo adder block (not shown in Fig. 3). The DSP computation is carried over each of the 27 channels with complete independence between the channels. This offers unique opportunities for fault detection/tolerance and simple testing [5] and clocking strategies.

This example architecture has an input dynamic range of 3 (for both real and imaginary parts of the Gaussian number). The maximum output dynamic range is 468 (from -234 to +234) for the real part and 468 for the imaginary part with one cascaded multiplication and 26 additions. We may increase the input dynamic range, output dynamic range, or/and the block length of the inner product (number of additions) by allowing the occasional overflow of the output polynomial coefficients and/or grouping the input data bits. In the next section we discuss the optimization of such an architecture.

3.3 Statistical Analysis

A software package, *MODULUS*, has been developed to relate the statistical distribution of the input data streams to the output polynomial coefficients using both the polynomial ring and RNS mapping techniques. The software uses the following input and output variables;

- Product of moduli
- Input data distribution
- Input bit representation
- Dynamic range of the input data
- Block length
- Probability of overflow (POF)

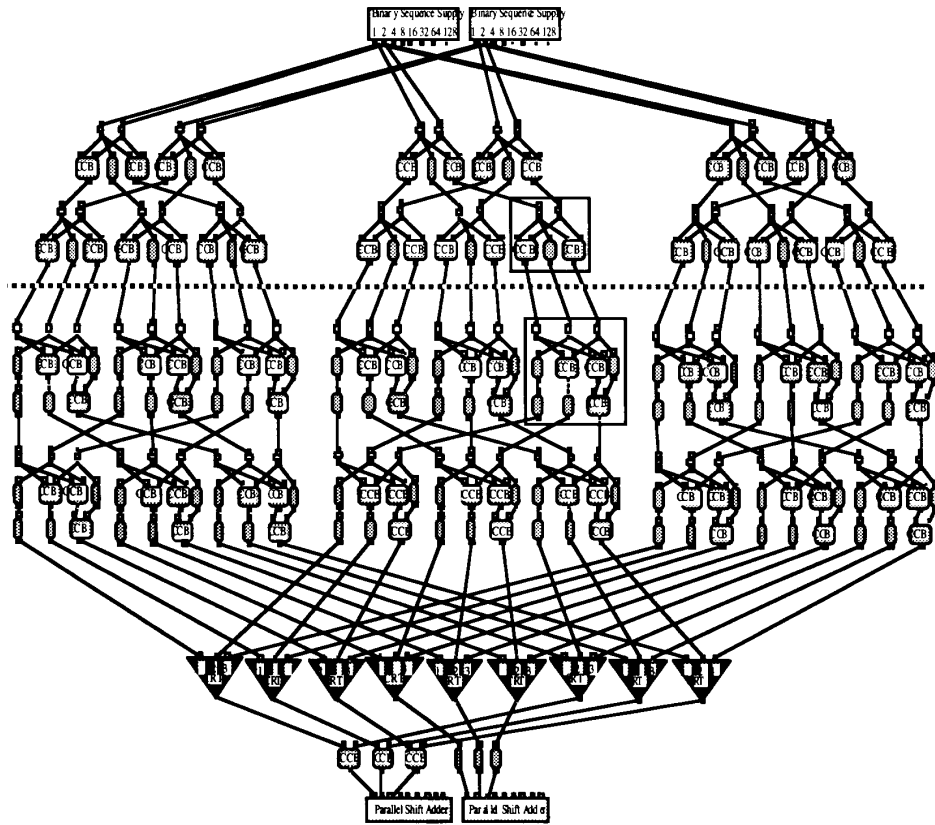


Fig. 3 Complex integer mapping simulation

As an example of using the software, it is found that the output dynamic range is increased to 490 by grouping 2 bits to the input polynomial coefficients X and XY (where $X = 2$ and $Y = j$). The input dynamic range also increased to 7 for both real and imaginary parts. However the block length decreases to 5 with the restriction of zero probability of overflow (POF).

If we allow the occasional overflow, Fig. 4 shows the block length versus the POF for the original problem (real and imaginary input dynamic range of 3). It is important to mention that for a block length of 26 the POF is zero where for block length of 27 the POF is less than 4.4×10^{-65} for the output polynomial coefficient that overflows (XY in this case). At a block length of 53 other coefficients will start to overflow. This means that up to block length of 52 the imaginary part will have a maximum error of 44.4% of the maximum imaginary value possible. If we scale the output, for instance, to half the number of bits the overflow error will vanish. It is clear that building a system with POF=0 is extremely

pessimistic, however, the application might restrict our choices.

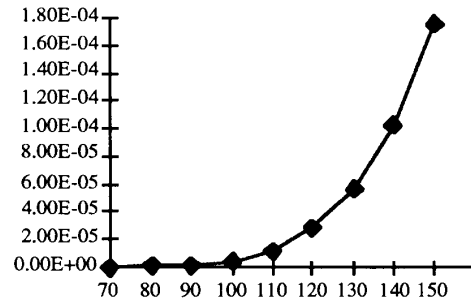


Fig. 4 POF versus block length

4: Synthesizing The Pipelined Blocks

The mapping architecture relies on pipelined 6-input blocks for the majority of the computational requirements. Appropriate circuitry for such blocks has been discussed

elsewhere [4]. Our interest, here, is to introduce a simple module generator for such blocks.

4.1 Embedded Single Phase Clocked Latch

The complete single phase clocked latch, with embedded switching tree, is shown in Fig. 5.

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.

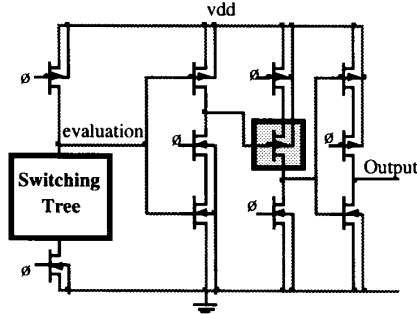


Fig. 5 Embedding a Switching Tree in a True Single Phase D-Latch

The tree is designed as an n -dimensional ROM (binary tree) where n is the number of input variables. Our minimization technique is based on the application of two simple graph reduction rules. We find this approach useful in that it allows a well established relationship between reduced tree structure and silicon layout that is essential for both hand custom layout and module generation approaches for complex multiple output trees. Pipelining each node is quite benign, because gate signals are guaranteed to remain constant over both precharge and evaluate cycles. Thus transistors that conduct when charge sharing occurs during evaluate also conduct during precharge, and this tends to precharge charge sharing nodes, resulting in a smaller charge sharing effect; in fact, as we slow down the throughput rate of the pipeline, the charge sharing problem disappears.

We can use the standard technique of internal tree p-channel pull-up transistors to reduce the charge sharing effect. Often a single pull-up transistor is sufficient because of the conducting transistors during precharge. We can also trade a reduction in precharge time for an increase in evaluate time, without reducing the throughput rate of the pipeline. Because we are only using a single pFET inverter for the p-logic block of the TSPC latch, we also have the flexibility of adjusting the precharge/evaluate duty cycle without being concerned about the effect on the pFET slave latch; i.e. the timing limitations are governed by the nFET latch circuitry.

A significant reduction in pull-down delay can be obtained by sizing the transistors. This is a complex issue when looking at the interaction between pull-down delay and charge sharing. We use an approximate analytical technique [1] that obtains very close to optimal results while allowing both on-the-fly calculations and algebraic manipulations for module generator applications.

4.2. Residue Module Generator

In this section we discuss a new approach to a module generator suitable for on-the-fly cell generation for arbitrary switching tree designs. This is essential for a design automation procedure, since it is impractical to pre-design a cell library based on the wide variety of truth table requirements that may have to be met in the conversion and DSP algorithmic computational blocks of the PRE.

Our approach is very similar to the gate matrix or PLA concept, where a two dimensional array of transistors is generated based on a transistor network mapped from a minimized Boolean function. In our case the network is a direct mapping from a minimized binary tree. In order to illustrate the procedure we have developed, an example of Mod 7 multiplication, $z = A \otimes_7 B$, will be used. The minimized tree [4] along with the matrix mapping is shown in Fig. 6.

The True and False edges on each row represent transistors whose gates are driven by the input signal or its complement. The mapping of primitives to the matrix is performed by either filling, or leaving empty, the table positions. The *Wire* and *Transistor* primitives are direct mappings from the switching tree, the shorting primitives are used to connect gate signals, propagated on metal 2 lines to polysilicon transistor gate lines. The metal 2 and polysilicon lines run horizontally across each row in the matrix with the metal 2 lines directly on top of the polysilicon lines. By shorting the metal 2 to the polysilicon at several places across the row (ideally near a transistor gate) we can eliminate the time constants associated with the large resistivity of polysilicon and transistor gate capacitances. We use space in the table to place the shorting primitives. Because these primitives are offset from the centre of the metal 2/polysilicon lines, they have two possible vertical directions; both directions have been used in Fig. 6.

The algorithm used to map the tree edges to the matrix primitives is given below:

- 1) Start at the top of the right hand tree; and map to the right most column in the matrix.
- 2) Move towards the bottom of the tree, taking either right hand edges or single merged edges, mapping the edges (vertical wire links or transistors) to matrix primitives in the column. Place horizontal wire matrix

primitives if a previously mapped edge (in the right hand adjacent matrix column) is connected to the currently mapped edge. The path will terminate when either a left hand link is reached, or when the bottom of the tree is reached.

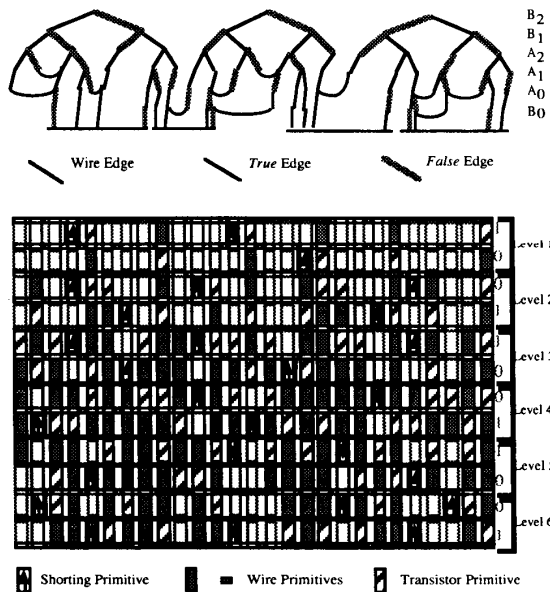


Fig. 6 Mod 7 multiplier tree and Table of primitives

3) Move to the left until the first unplaced left hand edge, at any vertical position, is reached. Terminate the algorithm if all edges have been placed.

4) Repeat from 2), mapping to a new column to the left of the previous column.

At the termination of the algorithm, the matrix is examined for suitable placement of shorting primitives. This is a somewhat heuristic procedure since there is a trade-off between reducing the resistance of the signal path to each transistor gate, and the extra capacitance load of the shorting primitive. There is often limited space for the shorting primitives, particularly near the dense central rows. We can see, from Fig. 8., that shorting primitives have been able to be placed within a short distance of every two or three transistors on a row; this will change with the particular function being implemented.

4.3 Floor Plan and Layout

Fig. 7 shows the floor plan and final layout of the Mod 7 multiplier, using a 3μ DLM p-well CMOS process [2].

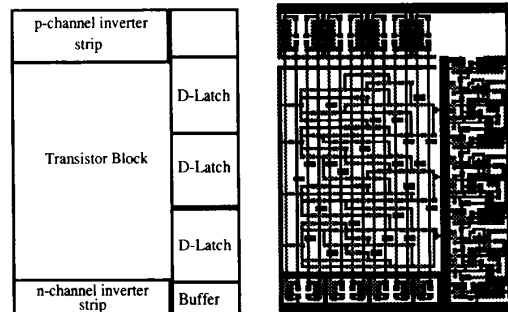


Fig. 7 Floor plan and layout for the Mod 7 multiplier

The transistor block contains the matrix of primitives mapped from the switching tree, and also the metal2/polysilicon signal wires. Note that the figures have been rotated by 90° . The inverters are formed by p-channel and n-channel strips, separated by the tree matrix. The matrix also includes the ground switch transistors, and the input clock signal to the switches is buffered by an inverter at the end of the inverter strip. The latch primitives are full custom layouts, and the clock signal to the latches is also buffered at the bottom of the latch column.

The transistor array governs the height of this particular example cell, but often the latches control the size, particularly for smaller numbers of inputs, or when there is a greater decomposition of the switching function (e.g. multi-bit binary adders). For such a cell, the area is now controlled only by the number of input bits (width) and number of output bits (height).

4.4 Comparison Study

Table 2 shows an area comparison study between the same Mod 7 multiplier cell between the synthesizer a hand layout and a hand layout of a TSPC pipelined PLA.

Design	Synthesized Switching Tree	Hand-Layout Switching Tree	PLA
Core Area	0.0514 mm^2	0.0747 mm^2	0.1659 mm^2
Relative %	100%	145%	323%

Table 2 Core Area Comparison

The PLA was not able to be folded because of the natural density of the interconnects with the Mod 7 function.

A power consumption comparison was made between the synthesized tree structure and the PLA. The study was conducted using mask extracted SPICE files, with level 3 models based on tuning from many fabrication experiments. The results are shown in Table 3.

Design	Switching Tree	PLA
Max. Throughput	50MHz	70MHz
Peak Current(40MHz)	2.85mA	8.36mA
Average Dissipation at 40MHz	3.45mW	15.4mW

Table 3 Speed and Power Comparison

We note that the PLA is able to operate at almost 50% higher throughput rates than the switching tree design; the trade-off, however, is the almost 5 times increase in power dissipation and 3 times increase in the peak current spike. This latter result can be as important as the power dissipation result, since the current spike is effectively multiplied by the number of cells on the chip for perfectly synchronized clocking (no skew between clocks arriving at the cells). This also speaks for producing architectures that allow clocks to be skewed, and the PRE architecture is directly suitable for such skewed clocking, since the computations are carried out in independent pipelines.

We have partially verified the throughput rate predictions by fabricating 6-high test switching trees and observing successful operation at the bandwidth of the output drivers (40MHz). We have also verified, via SPICE simulations, that the concept can be extended to more aggressive technologies, such as sub-micron CMOS technologies, with the same area and power savings over more conventional implementation techniques.

5: Conclusions

In this paper we have discussed architectures for input/output conversion of integer data streams for the Polynomial Ring Engine. We have reviewed the mapping procedures for both input and output conversion and demonstrate that the architecture can be constructed with 6-input pipelined switching blocks and standard binary adders. We have demonstrated the use of a new software package, MODULUS, to optimize mapping parameters for a specific algorithm. Although we are able to design systems with zero probability of overflow, this is a very pessimistic design philosophy, and so we are able to more efficiently use the architecture.

We have also given details of a module generator that can be used for on-the-fly generation of the pipelined switching blocks required both by the converter and the

DSP algorithmic computation architecture. The generator produces very area efficient designs with much lower power consumption compared to a pipelined PLA circuit using similar dynamic pipelines and implementing the same function at the same throughput rate.

6: References

1. S. Bizzan, G.A. Jullien, W.C. Miller, 1992, "Analytical Approach to Sizing NFET Chains", IEE Electronics Letters, Vol. 28, No. 14, July, pp. 1334-1335..
2. "Guide to the Integrated Circuit Implementation Services of the Canadian Microelectronics Corporation." . 1986.
3. Games, R. A. "An Algorithm for Complex Approximations in $Z[e^{2\pi i/8}]$." *IEEE Trans. Inform. Th.* IT-32 603-607, 1986.
4. Jullien, G. A., W. C. Miller, R. Grondin, Z. Wang, D. Zhang, L. Del Pup and S. Bizzan. "WoodChuck: A Low-Level Synthesizer for Dynamic Pipelined DSP Arithmetic Logic Blocks." *IEEE International Symposium on Circuits and Systems.* 1 pp. 176-179, 1992.
5. 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.* 2, 3, pp. 131-138, 1990.
6. Kung, S. Y. "VLSI Array Processors." 1988 Prentice Hall.
7. Soderstrand, M. A., W. K. Jenkins, G. A. Jullien and F. J. Taylor. "Residue Number System Arithmetic: Modern Applications in Digital Signal Processing." . IEEE Press, 1986.
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 1988.
9. Wang, Z., G. A. Jullien and W. C. Miller. "Algorithms for Length 15 and 30 Discrete Cosine Transforms." *1991 Asilomar Conference on Circuits Systems and Computers.* November pp. 111-115, 1991.
10. Wigley, N. M. and G. A. Jullien. "Large Dynamic Range Computations Over Small Finite Rings." *IEEE Trans. Computers.* In Print 1992.

Acknowledgments

The authors acknowledge the financial support of grants from the Natural Sciences and Engineering Research Council of Canada, the Micronet Network of Centres of Excellence and the Canadian Microelectronics Corporation for providing design and test equipment and fabrication services. The authors are also indebted to Mr. D. Reaume for generating the concepts and code for the MODULUS package.