

Homogeneous VLSI structures for high speed digital signal processing using number theoretic techniques

G. A. Jullien and W. C. Miller,

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

ABSTRACT

Exact computations, performed with residues, occur in Number Theoretic Transforms and Residue Number System implementations. Once thought awkward to implement with standard logic circuits, the application of efficient *small* lookup tables, constructed with pipelined dynamic ROM's, allows very efficient construction of hardware ideally suited to residue operations. Linear DSP operations that are compute bound (require many arithmetic computations per input/output sample), are best suited for implementation with systolic arrays. For very high throughput operations, bit-level systolic arrays appear to be ideally suited to their implementation. The dynamic ROM's used for the residue computations, are a perfect vehicle for implementing such operations at the bit-level in a systolic architecture.

This paper discusses VLSI architectures based on finite ring computations, using linear systolic arrays of small look-up tables. The advantage of this approach is that fixed coefficient multiplication requires no increase in hardware over that required for general addition, and the resulting structure is homogeneous in that only one cell type is required to implement all of the processing functions. Several advantages accrue in the VLSI setting, particularly clock rate increase, ease of testability and natural fault tolerance

Three different approaches to implementing the finite ring/field residue calculations are discussed. The first uses a bit-level steering mechanism around small dynamic ROM's; two applications of this technique to digital signal processing are outlined. The other two techniques are based on a redundant binary representation implemented with a pipelined adder configuration, and an iterative solution technique based on neural-like networks.

INTRODUCTION

Number theory was, until quite recently, mainly a rather esoteric subject in mathematics. In terms of practical benefits, coding theory appeared to be the main beneficiary. When the Number Theoretic Transform was introduced^{1,2}, the idea of performing digital computation over finite rings or fields became known to a wider audience in the DSP community. In the Computer Arithmetic community, this concept had been explored as *Residue Number Systems* (RNS) for at least the previous 15 years³. After a flurry of activity with NTT's, most of the interest has disappeared. A small group of researchers is still trying to exploit the inherent benefits of RNS implementations⁴, but there has been little commercial activity (except, perhaps, in proprietary defense products). What has remained somewhat a mystery, has been how to build efficient computers that can naturally perform the finite ring/field operations. Very few people have looked at this problem without resorting to the use of available binary arithmetic elements, either in commercial integrated circuits or as standard cells in a VLSI CAD library. This tends to lead to the restriction of the types of rings or fields that can be successfully implemented, and this is very evident in most published works on implementation of NTT's⁵ and many RNS hardware structures⁶.

One of the important things that came to light in work on RNS hardware, was the use of look-up tables to perform pre-stored computations⁷. Most of this work was carried out with the idea of using large commercial ROM chips for the tables, but recent work, by the author's group at Windsor⁸, has revealed that there is great efficiency in building small ROM elements from a single generic cell in a full-custom VLSI CAD setting. Given current (as yet unreported) work from the group, it seems quite possible that such an

approach can be used to efficiently implement even binary arithmetic elements. Based on these possibilities it seems that restricting finite field/ring computations to match, as closely as possible, binary implementations, is no longer necessary.

This paper considers the implementation of finite field/ring arithmetic systems, from the point of view of generating high performance VLSI digital signal processing systems with homogeneous architectures. Such architectures result from building the arrays with small generic lookup table cells and the use of massively parallel processing structures operating at the bit level. An important parallel processing architecture for digital signal processing (DSP) systems, is the systolic array. For maximum performance, systolic arrays operating at the bit level are prime candidates for high performance DSP.

Since their introduction^{9,10}, bit-level systolic arrays have been a major factor in the implementation of very high speed, computationally intensive DSP algorithms. For systems requiring very high throughput, at the expense of some latency, bit-level systolic arrays represent the optimum solution for many DSP requirements. Until recently, the work on bit-level systolic arrays concentrated on the use of binary arithmetic, which generates large 2-dimensional arrays because of the carry interconnection requirements¹¹. A cellular structure was recently introduced for implementing inner product step processors (IPSP), over finite rings, at the bit-level¹². The cell is able to compute a complete word level IPSP in B cells (connected in a linear systolic array) where B is the width of the word, in bits. The concept, as introduced, relies on the ability to build arrays where all the multipliers are known apriori. In this case the cell can be constructed with a very small ROM, along with steering switches and latches. The remarkable fact is that the cell is only twice the area of a pipelined, gated full adder, and runs at a greater throughput rate⁸. The cell can be designed with a certain measure of fault detection in a very simple manner¹³, and the replication of this cell over an entire computation allows complete fault detection of an entire chip. A further advantage with this approach is the ease of testing the array, and 100% coverage of single faults is possible with a very small set of test vectors¹⁴. For fixed coefficient arrays, the complexity associated with multiplication has been removed, and arrays can be designed for optimal structural properties, rather than the minimization of multiplication. There is some overhead with coding and decoding, but this can normally be amortized over a large number of IPSP operations used for computing the algorithm. In fact, the same IPSP cell can be used for the coding and decoding operations, resulting in truly homogeneous arrays.

Other cellular approaches have also been developed for finite ring computations. A scheme based on a pipelined adder, first introduced for binary arithmetic, uses a redundant representation for the residues within a binary word¹⁵. This scheme allows calculations within large fields/rings and is inherently suited for the finite ring computations associated with Chinese Remainder Theorem (CRT) calculations¹⁶. The logic associated with the steps in the process is complex, but this complexity is removed when implemented with small tables. As with the IPSP cell, fixed coefficient multiplication is simplified. A totally new approach to finite ring computations has been recently discovered that is based on the architectures of neural networks. In this case the neurons are digital and, as with the two previous approaches, also operate at the bit-level¹⁷. The network has the feedback characteristics of neural networks, but the iterations to a stable result can be mapped to a feedforward structure, resulting in a pipelined architecture. As before, the complexity of the logic can be removed using an implementation based on lookup tables. One major advantage of this approach to the building of massively concurrent systems, operating over finite rings, is the removal of the second dimension in the connectivity. The removal occurs because we can perform independent calculations over several small rings, and yet still compute over a much larger ring that is isomorphic to the direct sum of the individual rings. There is an algebraic constraint that the rings have numbers of elements that are relatively prime to one another, but this can certainly be accommodated in most practical realizations. By removing the connectivity over a large dynamic range, we remove many of the problems of clock skew in terms of slowing down the system throughput, and yet allow data direction clock skew (which does not slow down throughput) to be used for averaging current spikes⁸. Systems that are essentially connected in only one dimension, are very easy to test; this is becoming a very important issue as the density of VLSI circuitry increases.

MATHEMATICAL PRELIMINARIES

Notation

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

Base Ring(Field):

$$R(m_k) \text{ or } GF(m_k) = \{S: \oplus_{m_k}, \otimes_{m_k}\}; S = \{0, 1, \dots, m_k-1\}$$

where the symbols \oplus_{m_k} and \otimes_{m_k} correspond to modulo m_k addition and multiplication, respectively.

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

Direct Sum Ring:

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

Where it is not obvious by context which ring the operation is being performed over, a subscript will be employed. Thus \oplus_m indicates addition over the ring with modulus m . Note that direct sum rings are not rings over which actual implementation of the digital signal processing algorithm takes place; they are isomorphic to the direct sum of the corresponding implementation rings. We have therefore used the symbols $\overline{\oplus}$ and $\overline{\otimes}$ rather than the expected symbols \oplus_M and \otimes_M . Results become available over these rings following the appropriate isomorphic mapping (e.g Chinese Remainder Theorem).

The quadratic residue ring

In the case of complex signal processing, we can make use of quadratic residue ring theory¹⁸. Operations over a quadratic residue ring are component-wise for both addition and multiplication. This leads to isolation between channels and reduced arithmetic operations, both major advantages of using finite ring computations.

We will introduce quadratic residue ring theory with an addition to the notation.

Quadratic Residue Ring:

$$QR(m_k) = \{S: \oplus, \otimes\}; S = \{A^\circ, A^*\} \text{ with } A^\circ, A^* \in R(m_k) \text{ and } A^\circ = a+jb, A^* = a-jb; j = -1;$$

$$a, b, j \in R(m_k), m_k = p_i^{e_i}, p_i = 4k+1, \text{ a prime. } A^\circ \text{ will be referred to as the } \textit{normal} \text{ component of element } \mathbf{A}$$

and A^* as the *conjugate* component of element \mathbf{A} . The multiplication and addition operators both compute component-wise.

Direct Sum Quadratic Ring:

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

Note that the actual computations take place over the base fields associated with the normal and conjugate

components of the quadratic residue elements.

Residue Number System

A digit in the residue number system is represented by an L-tuple of residues⁴: $X = (x_0, x_1, \dots, x_{L-1})$ where $x_i = (X) \text{Mod } m_i$ is the *i*th residue and m_i is the *i*th modulus. Closed computations (addition, multiplication) over the implementation rings map to the direct sum ring via the Chinese Remainder Theorem (CRT). We will write this succinctly as:

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

with $A, B \in R(M)$; $a_k, b_k \in R(m_k)$ and $R(M) \sim_{\oplus} R(m_k)$ (direct sum).

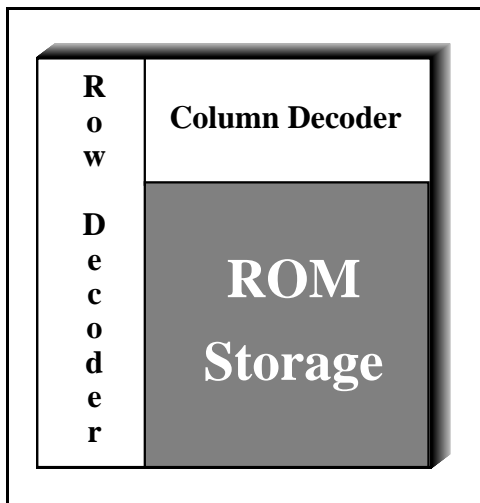
The final mapping is found from the CRT:
$$X = \sum_M^L \left\{ \hat{m}_k \otimes_M \left[x_k \otimes_{m_k} (\hat{m}_k)^{-1} \right] \right\}$$

with $\hat{m}_k = M / m_k$, $X \in R(M)$, $x_k \in R(m_k)$ and $(\cdot)^{-1}$ the multiplicative inverse operator. We have also used the notation \sum_M to indicate summation over the ring $R(M)$. Note that we use the expected ring operators, rather than the direct sum operators, since the CRT is the mapping to the ring $R(M)$.

ROM CELLS

Before we look at architectures that compute over finite rings, it will be constructive to examine the types of VLSI architecture that will be used for the generic cells.

Pipelined dynamic ROM cells



It is important to note that dynamic CMOS logic can be used not only to reduce the Area.Time complexity of logic gates, but can also be an important tool in pipelined systems. The natural storage element formed by the parasitic drain capacitances of dynamic output nodes, can be used as the memory elements for dynamic pipelines. The basic storage structure for the ROM is shown on the left. The most efficient layouts being obtained for 4 column select lines and 2^{n-2} row select lines, where *n* is the number of bits needed to represent all of the possible input states. These input states will either be elements of the ring or logic states associated with bit manipulations. We normally try to keep *n* 5 for efficiency, based on experimental comparisons with other techniques. The storage structure is simplicity itself, as can be seen by the schematic on the following page. The output line relies on the storage properties of the parasitic capacitance attached to the line. The pipeline cycle consists of a pre-charge phase, where the line is charged to the supply voltage, followed by an evaluate phase where the line is either discharged (if there is a path to ground) or not (no path to

ground). The path to ground is determined by the selection cross-point of the row and column decoders, and the presence, or absence, of a row select transistor. The cell is programmed by removing row select transistors. The two phase pre-charge/evaluate cycle is used to form a pipeline, where adjacent cells are in opposite phases. We have now combined both logic evaluation and pipelining function into an efficient,

and high speed, dynamic CMOS operation. There are a variety of choices as to what constitutes a single 2-phase cell. We can, for example, use the decoder structure as one cell and the ROM storage as its adjacent cell, or we may elect to combine both functions into a single cell. In the case of the first algorithmic approach to be discussed, we may elect to remove the decoders completely, and simply transfer cross-point selection information. The number of lines increases due to redundancy, but the structural elements reduce to a dynamic ROM storage cell with a buffer/driver¹⁹. Simulation and experimental results have confirmed that these cells pipeline at a high throughput rate (40MHz with 3 μ CMOS).

Finite Ring vs Binary

The most important VLSI considerations for using finite ring computations in large arrays are: the improvement of clock rate; the ease of testing the array; the fault tolerant capabilities. The improvement in clocking rate occurs because the dynamic range of the direct sum computation (which will be matched to an equivalent requirement for a binary implementation) is implemented with calculations over much smaller dynamic ranges; i.e. the range of elements in each of the individual rings/fields. This means that the connectivity requirements over the second (dynamic range) dimension on the 2-D chip are reduced. It is the action of synchronously clocking a large array of cells connected in 2-dimensions that slows down the clock rate. Our experiments show that large binary bit-level systolic arrays run considerably slower than do individual cells that make up the array. For the first technique, we will see that the 2nd dimension connectivity completely disappears, the other two techniques maintain limited 2nd dimension connectivity.

The testing issue also depends on the 2nd dimension connectivity. Finite ring implementations will always be easier to test than equivalently sized binary implementations. The first implementation technique allows testing of large arrays with only 34 test vectors²⁰.

RNS arithmetic has natural fault tolerant capability by the addition of redundant rings/fields to the direct sum mapping⁴. It is also possible to implement fault detection in linearly connected memory cells, much the same way as fault detection is implemented in communication systems with parity checks. The first implementation technique to be discussed allows such a technique to be used very efficiently²¹.

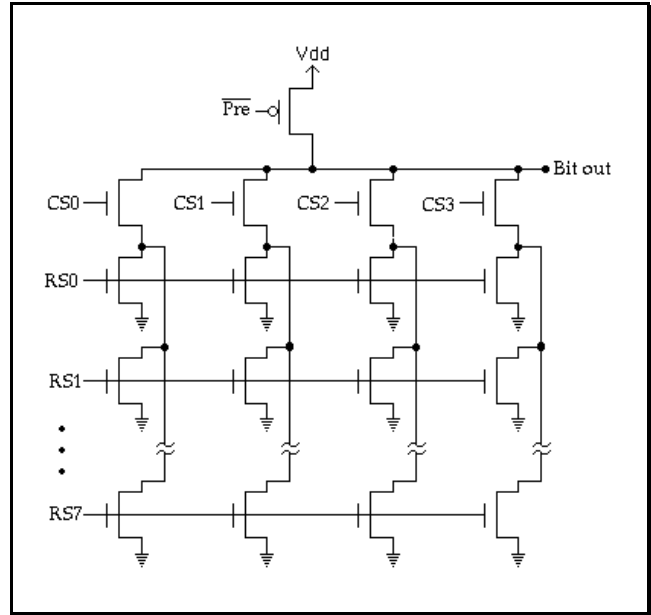
ROM STEERING TECHNIQUE

Many matrix operations and DSP algorithms can be implemented using repeated multiply and add operations in a loop. The operation is performed using the inner product step processor (IPSP). The basic arithmetic function of the IPSP, computed over R(m), is given in equation (1):

$$Y_{out} = Y_{in} \oplus_m [A_{in} \otimes_m X_{in}] \quad (1)$$

All the inputs and outputs are B bit ring elements $Y, A, X \in R(m)$ with $B = \lceil \log_2 m \rceil$. The operation of the fixed multiplier bit-sliced IPSP over R(m) (denoted the BIPSP_m) is defined by equation (2).

$$y^{(i+1)} = y^{(i)} \oplus_m [A \otimes_m x^{[i]} \otimes_m 2^i] \quad (2a)$$

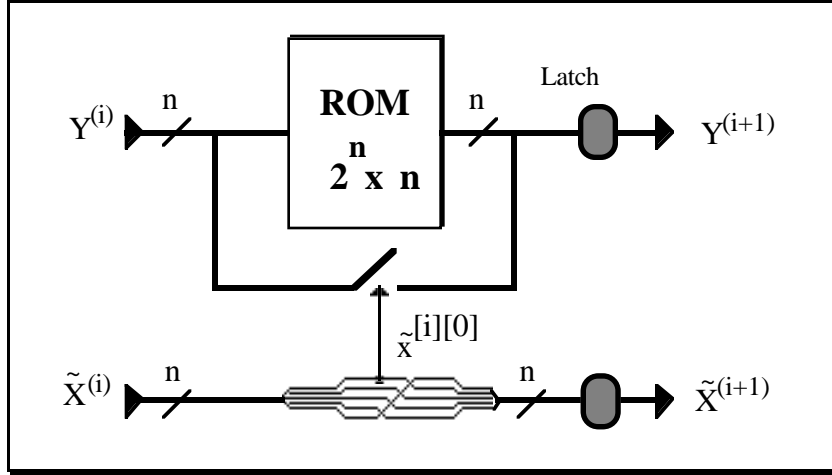


$$x^{(i+1)} = x^{(i)} \quad (2b)$$

where i is the spatial array index, $y^{(i+1)}, y^{(i)}, A \in \mathbb{R}(m)$, and $x^{[i]}$ is the i th bit of $X_{in} \in \mathbb{R}(m)$.

Systolic Implementation of the BIPSP_m cell

A particular implementation of the cell is shown on the left. Inputs to the cell are $y^{(i)}$ and $\tilde{x}^{(i)}$ (cyclically rotated mapping of $x^{(i)}$); the output is $y^{(i+1)}$. The mapping $\tilde{x}^{(i)} \Rightarrow \tilde{x}^{(i+1)}$



corresponds to a cyclic shift of one place in the binary coding of the x input. The cell contains a ROM of size $B \cdot m$ bits and a set of steering switches. The ROM stores the operation of $y^{(i)} \oplus_m [2^i \otimes_m A]$. The cell computes the following:

For $\tilde{x}^{(i)[0]} = 1$:

$$y^{(i+1)} = y^{(i)} \oplus_m [2^i \otimes_m A]$$

For $\tilde{x}^{(i)[0]} = 0$: $y^{(i+1)} = y^{(i)}$. If we expand eqn (1) as:

$$Y_{out} = Y_{in} \oplus_m \sum_{i=0}^{B-1} \{ A_{in} \otimes_m x^{[i]} \otimes_m 2^i \} \quad (3)$$

then it can be seen that the operator IPSP_m is equivalent to a linear array containing B consecutive stages of BIPSP_m cells.

Bit-Level Convolution

One of the important applications of this cellular structure is in bit-level FIR filtering. Using finite ring arithmetic, an N^{th} order fixed coefficient FIR filter can be expressed as:

$$|Y(n)|_m = \sum_{i=0}^{N-1} (A(i) \otimes_m X(n-i)) \quad (4)$$

where X, A, Y are input, filter coefficient, and output and A, X and $Y \in \mathbb{R}(m)$. Slicing (4) at the bit level yields:

$$|Y(n)|_m = \sum_{i=0}^{N-1} \sum_{b=0}^{B-1} 2^b \otimes (A(i) \otimes x^{[b]}(n-i)) \quad (5)$$

where $B = \lceil \log_2 m \rceil$. To implement this array we use a cell with an extra latch in the $\tilde{x}^{(i)[0]}$ bit line. This latch is used to provide the sliding action between the filter kernel and the signal being filtered. The algorithm embeds word level convolution with bit level fixed coefficient inner product computation in a perfectly homogeneous architecture. The effect of the application of our ROM cell on this algorithm is a set of low dynamic range finite ring linear systolic arrays, operating independently and yet effectively computing over a large dynamic range. The direct sum mapping in this case is performed using the same cellular structure but using a special mixed radix, binary conversion technique¹².

Bit-Level Systolic 2-D DFT

As a further example of the use of the fixed multiplier $BIPSP_m$, we present the results of implementing a 2-D DFT, computed over a direct sum of quadratic residue rings, as shown in equation (6).

$$\mathbf{X}_{k,l} = \bigoplus_{n=0}^1 \bigoplus_{m=0}^1 \{ \mathbf{x}_{n,m} \otimes \mathbf{j}^{2(nk+ml)} \} \quad (6)$$

with $\mathbf{j} = \sqrt{-1} \in QR(M)$, $k, l \in [0, 1)$, and $\mathbf{x}, \mathbf{X} \in QR(M)$. Using the QRNS decomposition we isolate the computations into $2L$ independent but structurally identical processors. The isolation of computations is important because we use QRNS in this approach for its structural, rather than arithmetic complexity reduction properties. The operations within the QRNS ring structure are base ring computations, even though the result will map to the extension ring. When embedded in an N point 2-D FFT structure, equation (6) is augmented with complex twiddle factor multipliers, $\{\alpha\}$. We will assume a pre-multiplication form for the butterfly. The operation of one of these processors (say the *normal* element) is shown in equation (7):

$$X^\circ_{k,l} = \bigoplus_{n=0}^1 \bigoplus_{m=0}^1 \{ x^\circ_{n,m} \otimes \mathbf{j}^{2(nk+ml)} \otimes \alpha^\circ_{n,m} \} \quad (7)$$

We take note of the fact that $\alpha^\circ_{0,0} = 1$, in any position of the $N \times N$ array. Equation (7) can be computed recursively (and a systolic structure obtained) using intermediate variables A, B as shown below.

$X^\circ_{k,l} = A^{[2k+1]} \oplus (\alpha^\circ_{1,0} \otimes \mathbf{j}^{2k} \otimes B^{[2k+1]})$ where initial values are: $A^{[0]} = x^\circ_{0,0} \oplus x^\circ_{0,1}$, $B^{[0]} = x^\circ_{1,0} \oplus x^\circ_{1,1}$ with the recursive formulations given in equation (8):

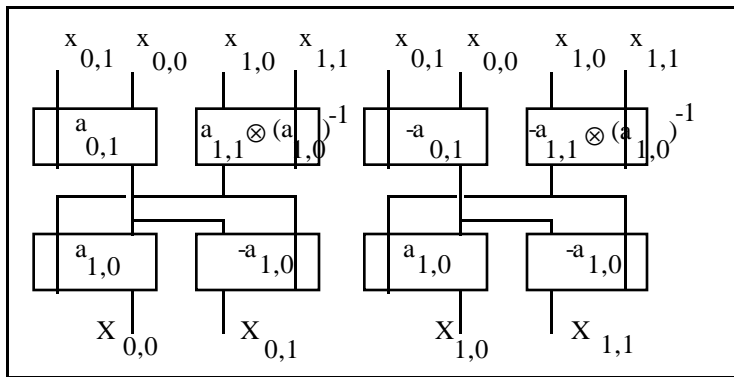
$$A^{[i+1]} = A^{[i]} \oplus \{ x^\circ_{0,1} \otimes \alpha^\circ_{0,1} \otimes (\mathbf{j}^{2(i+1)} \oplus [-\mathbf{j}^{2i}]) \} \quad (8)$$

$$B^{[i+1]} = B^{[i]} \oplus \{ x^\circ_{1,1} \otimes \alpha^\circ_{1,1} \otimes (\alpha^\circ_{1,0})^{-1} \otimes (\mathbf{j}^{2(i+1)} \oplus [-\mathbf{j}^{2i}]) \}$$

to compute $A^{[i]}$ and $B^{[i]}$ for $i=1,2,3$.

Note that we have defined an inverse, $(\alpha^\circ_{1,0})^{-1}$. Although we cannot rely on the QRNS mapped inverse for $\alpha^\circ_{1,0}$, the inverse exists for each component of $\alpha^\circ_{1,0}$ providing we compute over a base field.

By observing, and removing duplication of IPSP functions, we can make some simplifications in the structure. The semi-systolic (some broadcasting required) realization of the 2-D CE is shown on the left. The fact that multiplication does not involve extra hardware allows us to *spread out* the multipliers over the array. Contrast this to conventional structures that require fixed multiplication sites. The CE consists of two arrays, each with four IPSP modules. Note that there is some interchange of data position and some broadcasting between the top and bottom two pairs of arrays. The basic pipeline structure, however, maintains two linear arrays running concurrently.



and one with a binary weighted value m . The advantage of such a scheme is that there is no need to check whether the result is greater than m . Corrections need be applied only when there is a carry out of the MSB.

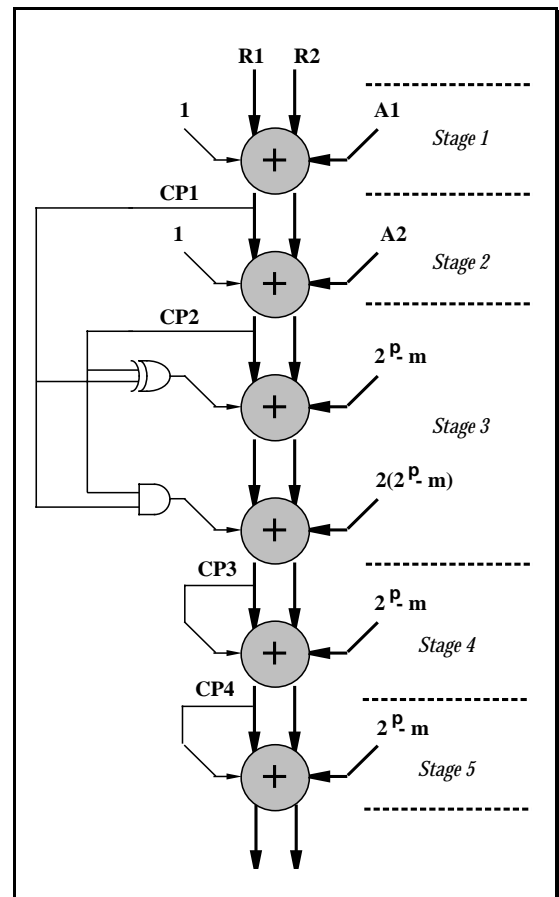
Design of a systolic residue adder

We consider two operands represented by two pairs of p -bit numbers ($A1, A2$) and ($R1, R2$). The result of adding the two operands, modulo m , will also be a pair of p bit numbers, and we may do this using two stages of addition, *each* arising from adding the R operand to the two components of the A operand in sequence. We apply an appropriate correction to the partial sum generated by the second stage (if required by the presence of a carry in stages 1, 2 or both); this correction requires, at most, three further addition stages, and the interesting feature is that this is independent of m . This is an important result for the problem of decoding using the Chinese Remainder Theorem where m is large.

The figure on the previous page shows how carry save residue addition may be performed using the nonunique representation. Each shaded circle in the figure represents a conditional carry save adder²² with four inputs and two outputs. The two vertical lines represent the carry/sum bits generated by the previous stage. The right horizontal input is a single p bit binary number. The left horizontal input is single bit which determines whether the addition operation will take place. If it is 0, the inputs received from the vertical inputs are merely passed on to the vertical outputs. If it is 1, carry save addition is carried out and the two vertical outputs correspond to the pair of p bits obtained by

REDUNDANT PIPELINED ADDER

In this approach we use redundancy in the binary representation of the finite ring elements. This redundancy is coupled with the inherent redundancy of the pipelined adder itself, in that the elements being summed are redundantly represented by a sum carry pair. The binary representation is unique for all residues, x , in the range $2^{P-m} < x < m$. Each residue in the range $0 < x < 2^{P-m}$ will have two representations, one with a binary weighted value $< m$



adding the input from the right horizontal line to the two vertical inputs. In the case of such an addition, there may be a carry out of the most significant bit depicted by the left horizontal line drawn from the output of each adder. The chain of adders may be directly implemented by a bit-level systolic array. The adders are realized, at the bit-level, by a ROM cell with 16 words of three bits each, as shown at the left. By simply changing the contents of the ROM, functions in the different levels can be fabricated with a single generic cell. The cell size is $400\mu \times 250\mu$ in 3μ CMOS, which is about the same size as a pipelined full adder cell⁸. In the case of fixed coefficient multiplication (such as the fixed kernel FIR filter), the addition steps in the multiplier require only 3

stages¹⁵.

NEURAL-LIKE FINITE RING COMPUTATIONS

The implementation procedure discussed here, is modeled after the massively parallel neural network architecture. Our computational structure consists of a set of bit-level adders for the neurons, and various finite ring mapped binary weights for the synapses. The computational technique relies on a convergent feedback system to perform modulo reduction on a binary representation of the result of an operation. It is assumed that this result has the form Z

$$= \sum_{i=0}^{n-1} 2^i \{z\} [i], \text{ where } \{z\} [i] \text{ is an}$$

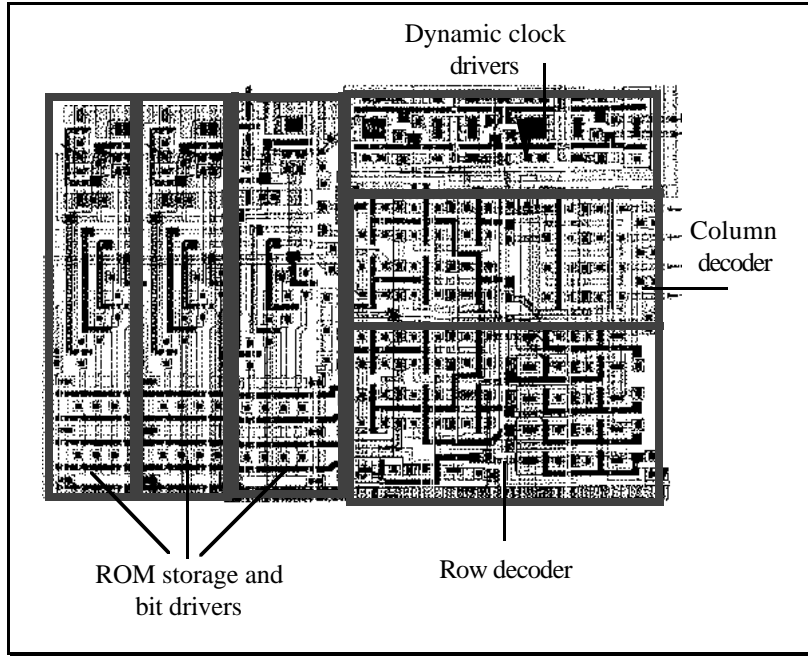
operator that extracts the i th bit of the binary representation of Z . We can show that all variable arithmetic finite ring operations are based on the computing model: $|Z|_m = \sum_{i=0}^{n-1} |2^i|_m \{z\} [i]$. An iteration in the computing model is defined as:

$$z(k+1) = \sum_{i=0}^{n-1} |2^i|_m \{z(k)\} [i] \tag{9}$$

The computing model can be iterated and the iterations converge. Since the iterations are guaranteed to be stable, the implementation can either be a stable synchronous feedback circuit, or these feedback iterations can be converted into feedforward pipelined paths, providing that the number of iterations required (as a maximum) can be determined. We also note that all finite ring arithmetic can be reduced to this computing model (addition, multiplication and any combination of these operations such as the Chinese Remainder Theorem).

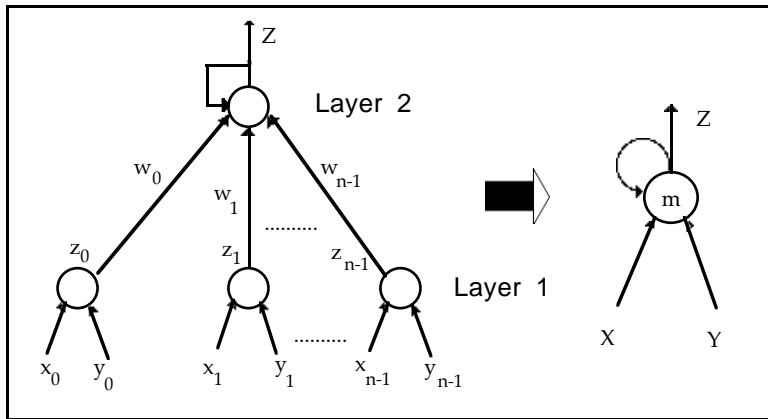
Finite ring neural network architecture

Based on the finite ring computing model, a basic operator can be extracted; the operator can be implemented using layered subnets, as shown on the left. The synaptic weight of z_i is $w_i = |2^i|_m$, ($i = 0, 1, \dots, n-1$).



The organization of the subnet has two layers. Layer 1, the collecting layer, is used to collect inputs belonging to the same binary bit from both input and feedback sources. Layer 2 is the computing layer, and implements the computing model of equation (9). The computing result is fed back as the sole input for subsequent states after the initial state. The system stabilizes as discussed previously.

The system may stabilize to a non-valid representation, i.e. the range $[m, 2^n)$. This can either be corrected if the value is required as a final result, or it can be left in this invalid state; this will not affect the operation of subsequent stages that use this value.

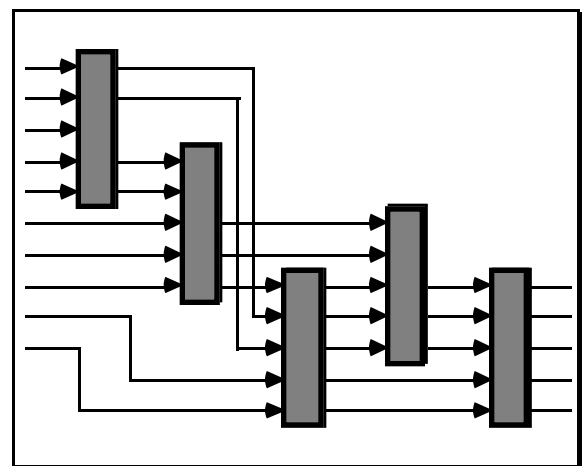
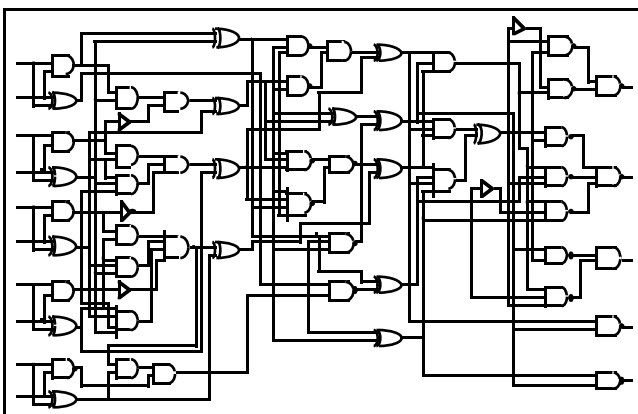


VLSI implementation

The implementation of this network can also be carried out using arrays of small ROM's. In this case the ROM's are used to implement *random* logic. The circuit contains many gates which have widely varying characteristics such as fan-in, fan-out, critical time-delay paths, power dissipation, silicon area

etc. The *random* topology itself creates varying conditions since, for example, gate delay depends on the number of gates being driven from its output.

The same circuit can be implemented by a composition of ROM logic cells in the configuration shown on the right. Although each cell is the equivalent size of several gates, the critical path through the cell is the same for all inputs and outputs; the topology is simplified; the fan-out usually small (in this case unity), and all cells are identical except for programming (which does not change the electrical characteristics). An example of the simplification of the circuit analysis is the determination of the critical time-delay path (the longest propagation delay through a connected network). With large combinations of basic logic gates, finding the critical path is complex. The *ROM logic* array has an identical delay for each cell since the fan-out is unity for all cell outputs. By inspection we see that the critical path is 5 cell delays! This final implementation procedure therefore introduces another dimension to the design of high performance arithmetic systems using the approach discussed here. The



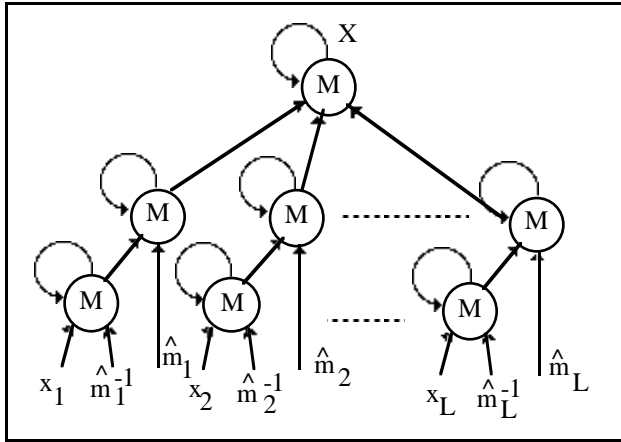
simplification of CAD requirements!

RNS to Binary converter (CRT)

For the general system, \hat{m}_i and \hat{m}_i^{-1} are treated as constants and the CRT may be rewritten as:

$$X = \sum_{i=1}^L Z_i; \quad Z_i = \hat{m}_i \otimes_M Q_i; \quad Q_i = x_i \otimes_M (\hat{m}_i)^{-1}$$

where $i = 1, 2, \dots, L$. The CRT can be implemented using the hierarchical network on the left, based on the operator subnet.



CONCLUSIONS

In this paper we have looked at the implementation of digital signal processing functions using finite ring/field computations. The theme running through the various implementations has been the use of generic ROM storage tables to implement the operations required to compute the modulo arithmetic required for finite rings/fields. We have shown that the use of residue number systems over normally implemented binary arithmetic, has great ramifications in clock rate, testing and fault tolerance for large VLSI computational arrays, operating at the bit-level. We have outlined the mechanism used to implement dynamic pipelined ROM

tables with very high throughput.

The paper has discussed three totally different methods of implementing finite ring/field arithmetic, all using single generic look-up table cells. The methods covered are: ROM steering; redundant representation with pipelined adders; neural-like networks with an iterating computational model. The first technique is seen to offer very high efficiency with small ring moduli and fixed coefficient multiplication. The other two techniques are better suited to implementations over large moduli; the Chinese Remainder Theorem isomorphic mapping being a prime candidate.

In the case of the ROM steering technique, we have discussed 2 common signal processing functions; FIR filtering and FFT computation. Both were considered with the idea of using the fixed coefficient simplification to produce very efficient designs.

REFERENCES

1. P. J. Nicholson, "Algebraic Theory of Finite Fourier Transforms," Journal of Computer and Systems Science 5, pp. 524-547, 1971.
2. J. M. Pollard, "The Fast Fourier Transform in a Finite Field," Mathematics of Computation 25, pp. 365-374, April 1971.
3. A. Svoboda, "Rational numerical system of residual classes," Stroje Na Zpracovani Informaci, vol. 5, pp. 9-37, Prague, 1957.
4. M.A. Soderstrand, W.K. Jenkins, G.A. Jullien, and F.J. Taylor, (eds) "Residue Number System Arithmetic: Modern Applications in Digital Signal Processing," IEEE Press, New York, NY, 1986.
5. J.H. McClellan, "Hardware Realization of a Fermat Number Transform," IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-24, pp. 216-225, June 1976.

6. F.J. Taylor, "Single modulus ALU for signal processing," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-33, no. 5, pp. 1302-1315, October 1985.
7. G. A. Jullien, "Residue Number Scaling and Other Operations Using ROM Arrays," *IEEE Trans. on Computers*, vol. C-27, pp. 325-336, April, 1978.
8. G.A. Jullien, P.D. Bird, J.T. Carr, M. Taheri, W.C. Miller, "An Efficient Bit-Level Systolic Cell Design for Finite Ring Digital Signal Processing Applications," *Journal for VLSI Signal Processing*, (in print), 1989.
9. P. R. Capello, K. Steiglitz, "Digital Signal Processing Applications of Systolic Algorithms," *VLSI Systems and Computations*, H.T. Kung, R. Sproull and G. Steele eds., Computer Science Press, pp.245-254, 1981.
10. J.V. McCanny and J.G. McWhirter, "Implementation of Signal Processing Functions using 1-bit Systolic Arrays," *Electronic Letters*, Vol. 18, No. 6, March, pp. 241-243, 1982.
11. A. Corry, K. Patel, "Architecture of a CMOS Correlator," *Proceedings of the Int. Conf. on Circuits and Systems*, pp. 522-525, 1983.
12. M. Taheri, G.A. Jullien, and W.C. Miller, "High Speed Signal Processing Using Systolic Arrays Over Finite Rings," *IEEE Journal of Selected Areas in Communications*, April, 1988.
13. M. Taheri, G.A. Jullien, and W.C. Miller, "Fault Detection in RNS Systolic Arrays," *IEE Electronics Letters*, Vol. 23, No. 4, pp. 165-166, Feb. 1987.
14. G.A. Jullien, M. Taheri, S. Bandyopadhyay, "A Low-overhead Scheme for Testing a Bit Level Finite Ring Systolic Array," submitted to the *Journal of VLSI Signal Processing*, August 1988.
15. S. Bandyopadhyay, G.A. Jullien and M. Bayoumi, "Systolic arrays over finite rings with applications to digital signal processing," *Systolic Arrays*, edited by W.R. Moore. A. P. H. McCabe and R. B. Urquhart, Adam Hilger, Bristol Techno House, pp 123-131, 1987 .
16. S. Bandyopadhyay, G. A. Jullien and A. Sengupta, "A systolic architecture for implementing the Chinese Remainder Theorem", *Proc. International Conference on Parallel Processing*, pp 924-931, Aug 1987.
17. D. Zhang, G.A. Jullien, W.C. Miller, 1989, "VLSI Implementations of Neural-Like Networks for Finite Ring Computations", *Proceedings of the 32nd Mid-West Symposium on Circuits and Systems*, University of Illinois, Urbana Ill. August 14-16, paper number NN-386, (In Print)
18. W.K. Jenkins and J.J. Krogmeier, "Error Detection and Correction in Quadratic Residue Number Systems," *Proc. 26th MidWest symp. Circuits and Systems*, Puebla Mexico, Aug. 1983.
19. G.A. Jullien, W C. Miller, "A New High Speed Cellular Structure for VLSI Implementation of Systolic Digital Signal Processing Architectures," *Proc. 1989 Canadian Conference on Electrical and Computer Engineering*, Montreal, September 17-20, paper V47.
20. G.A. Jullien, M. Taheri, S. Bandyopadhyay, "A low-overhead scheme for testing a bit level finite ring systolic array", submitted to the *Journal on VLSI Signal Processing*, 1988.
21. M. Taheri, G.A. Jullien, and W.C. Miller, 1987, "Fault Detection in RNS Systolic Arrays," *IEE Electronics Letters*, Vol. 23, No. 4, Feb., pp. 165-166
22. Kai Hwang, "Computer arithmetic : principles, architecture and design", John Wiley, 1979.