High Speed Signal Processing using Systolic Arrays Over Finite Rings

M. Taheri, G.A. Jullien, W.C. Miller

Department of Electrical Engineering
University of Windsor
Windsor, Ontario, Canada N9B 3P4

Abstract

This paper presents a simple, modular, architecture for very fast digital signal processing elements. The computation is performed over finite rings (or fields) and is able to emulate processing over the integer ring using residue number systems. The computations are restricted to closed operations (ring or field binary operators) with the ability to perform limited scaling operations. Computations naturally defined over finite mathematical systems (e.g. Number Theoretic Transforms, Quadratic Residue 'complex' calculations, Recursive FIR filters over finite fields) are also easily implemented using this new approach.

The technique evolves from the decomposition of each closed calculation using the ring/field associativity property. Linear systolic arrays, formed with multiple elements, each of a single generic form, are used for all calculations. The pipeline cycle is determined from the generic cell and is predicted to be very fast based on a critical path analysis. The cells are perfectly matched to the VLSI medium, and the resulting array structures are very dense indeed.

Examples of DSP applications are given to illustrate the technique, and sample cell and array VLSI layouts are presented for a 3μ CMOS process.

Key Words

Digital Signal Processing, Systolic Arrays, Finite Rings/Fields, ResidueNumber Systems, VLSI.

Introduction

The use of computations over finite fields are natural to communication researchers. Such computations form the backbone of many coding techniques in use today[13,15,4]. The use of finite field/ring operations for general computation has had some followers[1,5,6] but there has been a certain mistrust by a community that prefers the more natural weighted magnitude form of computations (e.g. Binary, BCD). For the special case of Digital Signal Processing applications, where inner product forms represent a large computational burden on the processor, some workers have turned their attention to finite ring/field processors as a possible alternative. A resurgence of interest[5,6] was generated when large memories became available at a relatively low cost. Multiplication, the scourge of binary arithmetic solutions, turned out to be as simple as addition, if the computations were performed over finite rings, with results being assembled to a weighted magnitude form at the end[6]. It now seems as though such computations also lend themselves admirably to the VLSI medium with dense modular, homogeneous, structures resulting[23]. Some of the problems related to VLSI systems, such as clockskew, testing and fault detection, may be mitigated via the use of this alternate computational medium.

It is the purpose of this paper to present an alternative approach for implementation of finite ring operations which requires less silicon area and at the same time providing higher speeds of operation than previously published works[18]. To start with we will present the mathematical preliminaries including the notation that has been adopted throughout the paper. In section I we will discuss the basics of inner product step processors[8] (IPSP), both at the word and bit level. We will then introduce a structure which implements a finite ring IPSP at the bit level. Several of these processing cells can be computed in parallel (over different ring moduli) and the results mapped to a larger modulus ring (residue number system) allowing suitable dynamic ranges for the total calculation but restricting each individual computation to a very small dynamic range. In section II we will discuss the implementation of fixed coefficient FIR filters, based on this

structure, for both bit parallel and bit serial inputs. In section III we will consider the interface problem between the finite ring processors and normal weighted magnitude fixed point (or integer) computations.

Mathematical Preliminaries

In residue systems we deal with rings, or fields, that are used for the actual implementation and rings that are isomorphic to direct sums of implementation rings or extensions of them. A notation already exists for indicating ring or field operations[1]:

$|\circ|_{m_k}$ where $\circ \in \{+, x\}$ and m_k is the modulus of the reduction operation.

For cascades of operations, mixed ring (field) operations (such as the Chinese Remainder Theorem mapping[1]) the notation can become confusing, with unwieldy nesting of modulo parentheses. We will use special symbols for operating over the different systems, so that such operations can be readily understood in an expression. We will deal with the following hierarchy:

Base Ring(Field): $R(m_k)$ or $GF(m_k) = \{S: \ddot{a}^a, \ddot{a}^o\}$ $S = \{0, 1, \ldots, m_i\text{-}1\}$

Direct Sum Ring: $R(M) = \{ \ddot{a}S : \ddot{a}^a, \ddot{a}^o \}$; $\ddot{a}S = \{0, 1, \ldots, M\text{-}1\}$; $M = \prod_{k=1}^{L} m_k$

Where it is not obvious by context which ring the operation is being performed over, a subscript will be employed. Thus $\ddot{a}^a_m$ 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. Results become available over these rings following the appropriate isomorphic mapping (e.g Chinese Remainder Theorem).

Residue Number System

This is a brief introduction using the above notation. A digit in the residue number system is represented by an L-tuple of residues[1]:

$$X = (x_0, x_1, ..., x_{L-1}) \quad (1)$$

where $x_i = (X) \text{Mod } m_i$ is the ith residue and m_i is the ith 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:

$A \ \ddot{a}^a \ B \Leftrightarrow \{a_0 {}^a b_0, a_1 {}^a b_1, \ .... \ , a_L {}^a b_L\}; \quad A \ \ddot{a}^o \ B \Leftrightarrow \{a_0 {}^o b_0, a_1 {}^o b_1, \ .... \ , a_L {}^o b_L\}$

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

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

The final mapping is found from the CRT:

$$X = \sum_{k=1}^{L} \ddot{a}^a \{ \ddot{o}m_k \ \ddot{a}^o [x_k {}^o (\ddot{o}m_k)^{-1}]\} \quad (2)$$

with $\ddot{o}m_k = M / m_k$, $X \in R(M)$, $x_k \in R(m_k)$ and $(\bullet)^{-1}$ the multiplicative inverse operator.

Note the mixture of ring operations (the summation is shown over R(M)). Since we are mapping to R(M), calculations $\ddot{a}^a$ and $\ddot{a}^o$ have to be carried out

using modulo M arithmetic.

Section I

Many matrix operations and DSP algorithms can be implemented using repeated multiply and add operations in a loop[10,16,14]. The operation is performed using the IPSP. This processing cell has 3 inputs: X_{in}, A_{in} and Y_{in} and three outputs: X_{out}, A_{out} and Y_{out}. The IPSP is described below:

$$Y_{out} = Y_{in} + (A_{in} \cdot X_{in}) \quad (3a)$$
$$X_{out} = X_{in} \quad (3b)$$
$$A_{out} = A_{in} \quad (3c)$$

and is shown, diagrammatically, in Fig 1. The inputs and outputs are word level signals and will contain the number of bits required by the dynamic range of the computation. The internal multiplier/adder has to process several bits per clock cycle resulting in large area and clock cycle time. The interconnection of several of these processors, in a regular form, is used to process the required task. The IPSP can be sliced into bits[9,11], each a gated full adder with four inputs: a_{in}, x_{in}, y_{in} and c_{in} (carry in) and four output bits: a_{out}, x_{out}, y_{out} and c_{out}. The operation of the bit-sliced IPSP (BIPSP) is described by:

$$y_{out} = y_{in} \ {}^{a}2 \ c_{in} \ {}^{a}2 \ (a_{in} \ {}^{o}2 \ x_{in}) \quad (4a)$$
$$c_{out} = Maj(y_{in}, a_{in}, c_{in}) \quad (4b)$$
$$a_{out} = a_{in} \quad (4c)$$
$$x_{out} = x_{in} \quad (4d)$$

where Maj({xi}) selects the boolean element in the majority in the set {ai}, ai $\in$ {0,1}.

The BIPSP is shown in Fig 2. Several of these slices can be interconnected to form a word level IPSP[9,11,17,]. Pipelining these simpler processors normally results in a much higher throughput compared to the word level IPSP. A finite ring counterpart for the binary IPSP can not, for a general ring, be constructed so easily. The most general approach is to use a ROM as a direct truth table implementer as shown in Fig 3[6].

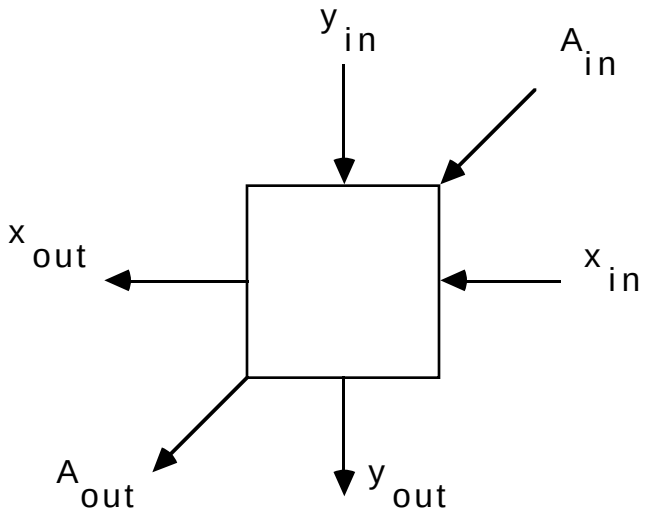


Fig 1 The IPSP

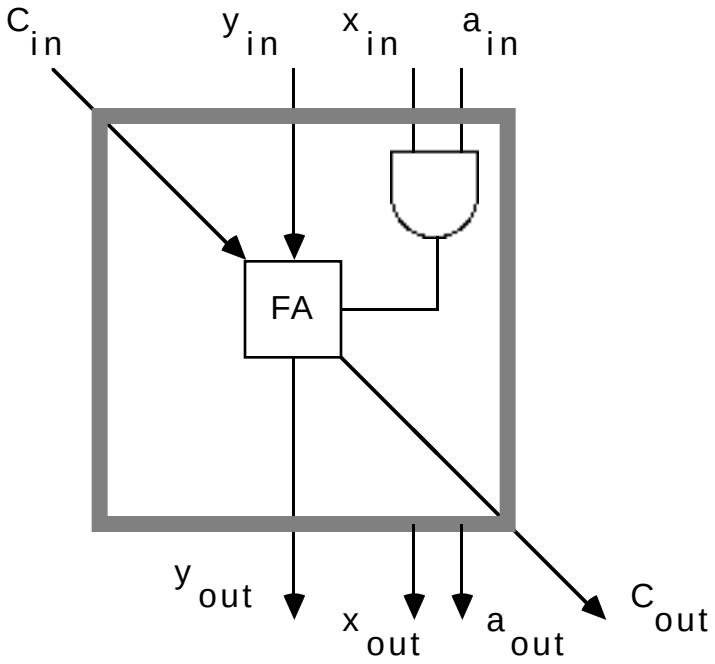


Fig 2 The BIPSP

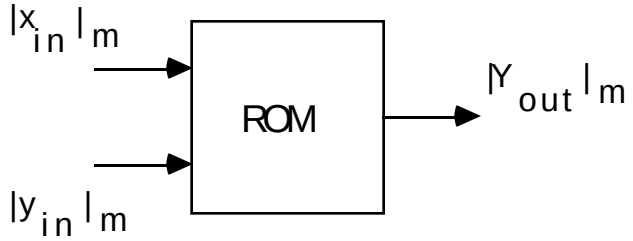


Fig 3 IPSP for Finite Ring Calculations

The ROM stores all the possible outcomes of the operation between all possible ring elements. A simple addressing operation, using the concatenated variables as the address, produces the result in the access time of the ROM. For an IPSP with a fixed multiplier we can write the relationship between input (address) variables and output (contents) variable as:

$$Y_{out} = Y_{in} \,{}^{a}_{m}\, [A_{in} \,{}^{o}_{m}\, X_{in}] \qquad (5)$$

All the inputs and outputs are B bit ring elements Y, A, X $\in$ R(m) with B=$\lceil \log_2 m \rceil$. By cascading 2 ROMs it is possible to implement a general, rather than fixed, multiplier. In this paper we will consider a structure equivalent to the BIPSP for finite ring calculations. We will use the symbol $BIPSP_m$ to indicate that the processor is operating over the finite ring, modulo m. It will be seen that the $BIPSP_m$ has similar advantages over the $IPSP_m$ that the BIPSP has over the IPSP. In addition the (Area.Period) product of VLSI implementations can be reduced considerably and dramatically reduced when the pipelining action is taken into account. The structure also satisfies the requirements of: modularity, homogeneity and local communication, sought after in 'good' VLSI designs. By repeatedly using a generic cell structure, we will show that all closed computations can be performed with parallel linear systolic structures, and that interfacing with normal fixed point computations can also be carried out using the same cell structure and linear systolic array concepts. We will finally illustrate this with an example of a FIR filter.

The operation of the fixed multiplier $BIPSP_m$ can be defined by:

$$y^{(i+1)} = y^{(i)} \; a \; [A_{in} \; o \; x^{[i]} \; o \; 2^i] \qquad (6)$$

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

The ring operations are shown without the subscript. A possible implementation of the $BIPSP_m$ cell is shown in Fig 4. Inputs to the cell are $y^{(i)}$ and $x^{[i]}$; the output is $y^{(i+1)}$. The cell contains a ROM of size B.m bits and a set of steering switches.

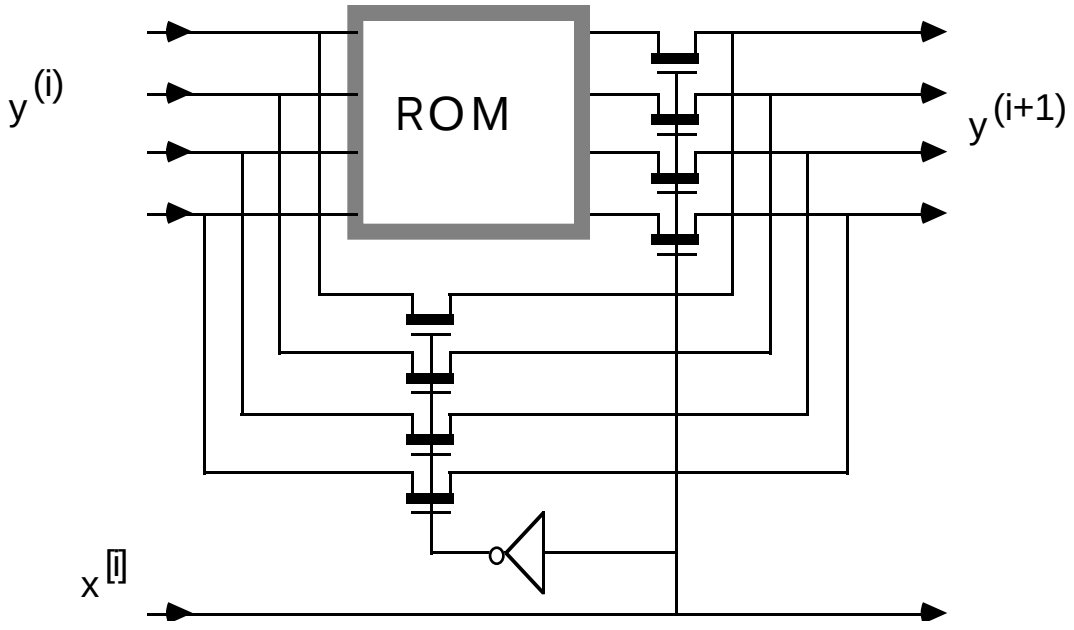


Fig 4 Implementation of the $BIPSP_m$ cell

The ROM stores the operation of $y^{(i)a}[2^{i}oA_{in}]$. The cell computes the following:

For $x^{[i]}=1$: $y^{(i+1)} = y^{(i)} \; a \; [2^i \; o \; A_{in}]$ (7a)

For $x^{[i]}=0$: $y^{(i+1)} = y^{(i)}$ (7b)

We can expand eqn (5) as:

$$Y_{out} = Y_{in} \,^{a}[\, \sum_{j=0}^{B-1}{}^{a} \{A_{in} \,^{o} x[j] \,^{o} 2j\} \,] \qquad (8)$$

where $\sum^a$ is the summation operator over the ring. It can be seen that the operator $IPSP_m$ is equivalent to a linear array containing B stages of $BIPSP_m$, as shown in Fig 5.

This technique only requires B^2m ROM bit locations compared to Bm^2 locations for the single ROM IPSP structure. This yields significant savings for large m. Without pipelining the array, a delay of Bë† is experienced compared to a delay of † for the single ROM implementation. ë† is the delay through the $BIPSP_m$ ROM and † the delay through the $IPSP_m$ ROM. When we consider the use of pipelining (as shown by the latches, ˙ , on the diagram), the throughput rate of the $IPSP_m$ cell is inversely proportional to $(\tau + t_L)$ compared to $(ë\tau + t_L)$ for the $BIPSP_m$ array. t_L is the latch delay.

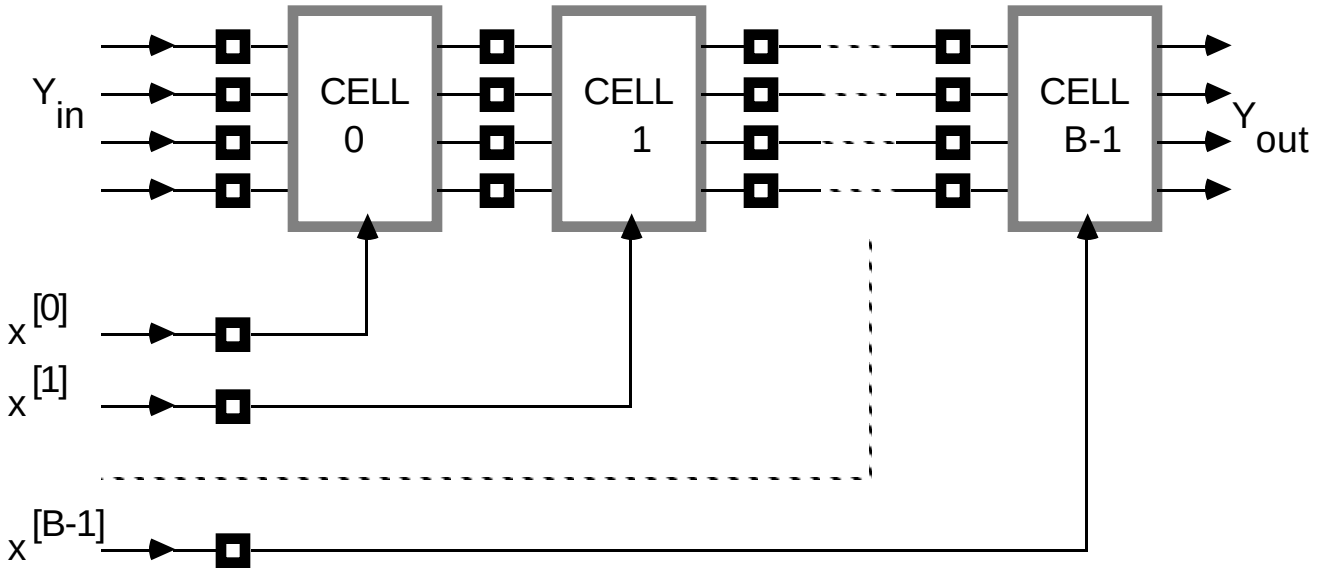


Fig 5 The $IPSP_m$ cell implemented using an array of $BIPSP_m$ cells

The ratio $(\tau + t_L)/(ë\tau + t_L)$ can be large for large m and so the $BIPSP_m$ array

can offer much larger throughput rates compared to the single $IPSP_m$ cell.

Although the processing section of the cells are uniform throughout the array, the geometry used for feeding the X bits, and their pipeline latches, cause a certain non-uniformity to the VLSI layout. Fig 6 shows the $BIPSP_m$ cell with all the X bits fed in parallel through each cell. The bits are circularly shifted by one position for each cell, automatically presenting the correct bit to the steering switches in each cell. This results in a regular, common, cell structure; the trade off is the requirement for extra latches. The multiplying factor is $2B/(B+1)$ which asymptotically approaches 2. For typical values of $B \leq 5$ we have a maximum increase of 66%. Although this appears considerable, the regularity of the modified structure mitigates the apparent increase in area and the availability of the entire X sequence at the output of the Bth cell is very useful.

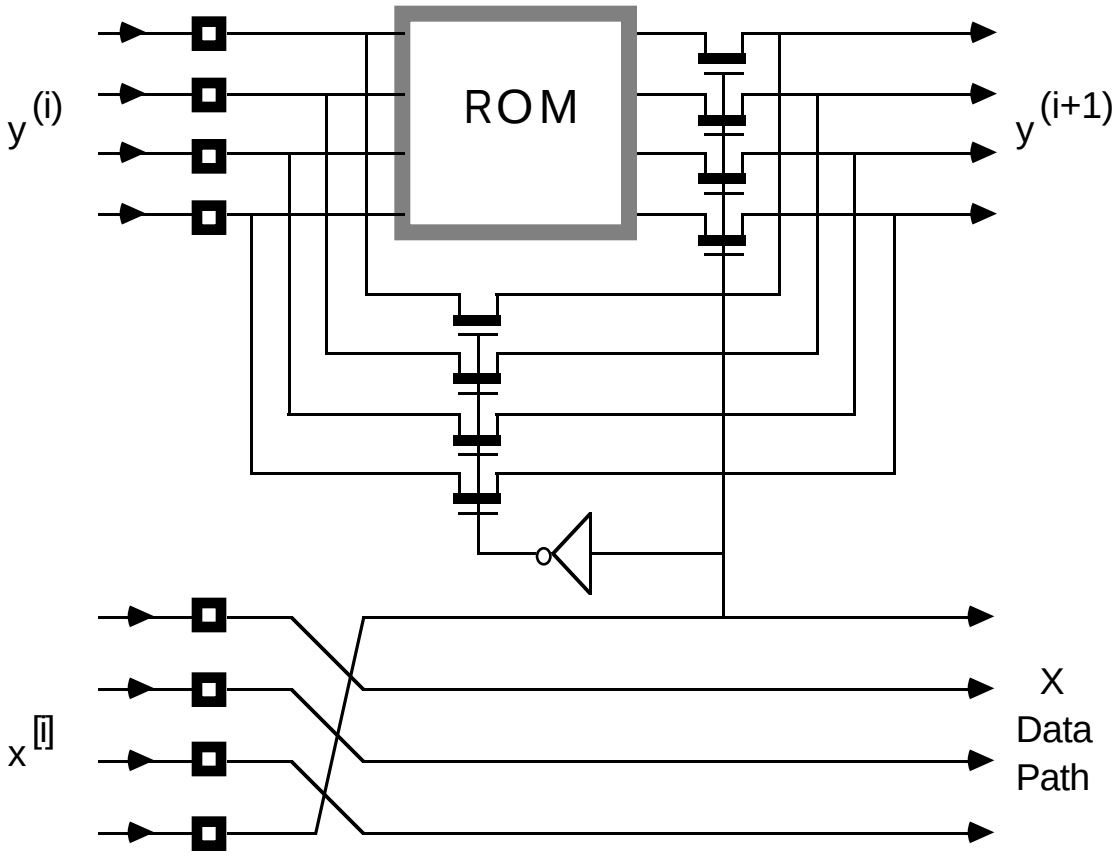


Fig 6 Systolic $BIPSP_m$ Cell

We have now created a universal systolic cell that only connects to its

immediate neighbours. The cell is shown with the pipeline latches connected to the input lines. An alternative scheme allows the pipeline latches at the output.

In order to obtain practical results from the cell architecture, several sample cells have been designed with different hardware structures. The designs are for a 3μ CMOS p-well, single metal fabrication process. The designs have been fabricated using a service offered by the Canadian MicroElectronics Corporation(CMC) to Canadian Universities.

As an example, Fig 7 shows the layout of a 5-bit cell using static logic decoders, single n-channel transistor memory elements and static latches. The design is compact and amenable to linear array implementations. The inputs enter at the top and exit at the bottom. The pipeline latches are formed at the output. The size of the cell is 845 x 624 μ^2. Many other different structures are possible, and we are currently investigating the optimization of the cell layout for various criteria.

The ROM structure is particularly efficient for a double metal process, and a cell is currently being designed for the 3μ double metal process recently introduced by the CMC.

General Multiplication

Although we are concentrating on fixed coefficient $BIPSP_m$ cells, it is possible to use similar cell structures for general multiplication. The finite ring general multiplier can be decomposed as shown in eqn (9).

$$A \circ B = \sum_{l=0}^{L-1} \sum_{k=0}^{L-1} \{ b^{[l]} \circ a^{[k]} \circ | 2k+l |_m \} \qquad (9)$$

Fig 7 $BIPSP_m$ cell using static decoders and latches

a) Plot of the composite mask layout

Fig 7 $BIPSP_m$ cell using static decoders and latches

b) Floor plan showing the separate cell components

For the purposes of generating a systolic array realization, we use the

following recursive formulation:

$$SëUM(i\text{-}1, j) = SëUM(i\text{-}1, j\text{-}1) \,^{a}_{m}\, \{b^{[i-1]} \circ_2 a^{[j]} \circ_2 |2^{i-1+j}|_m \qquad (10)$$

$$SUM(i) = SUM(i\text{-}1) \,^{a}_{m}\, SëUM(i\text{-}1, L\text{-}1) \qquad (11)$$

Eqn (10) represents the inner summation, SëUM, and eqn (11) the outer summation, SUM. We have used the ring operator, $\circ_2$, since $b^{[i-1]}, a^{[j]} \in \{0,1\}$ and the logical AND operation is clearly implied. Clearly there are several alternate realizations and both one and two dimensional systolic arrays (with two types of cells) can be constructed. For ROM based cells it is much better to use a linear array which results in a reduced input word width to the ROM element. In this case we can effectively remove the physical structure represented by eqn (9) by recognizing (over a linear array) that:

$$SUM(i) = SëUM(i\text{-}1, L\text{-}1) \qquad (12)$$

Using the same generic cell structure (with the addition of a second word input) we can implement the structure implied by eqns (10) and (11). For the purposes of this paper, however, we will concentrate on the fixed multiplier cell. In the following section we will discuss the implementation of a fixed coefficient FIR filter that uses the $BIPSP_m$ cell to great advantage.

Section II

In this section two different FIR filter structures will be considered. These will be based on, firstly, a bit parallel and, secondly, a bit serial input.

Bit Parallel input FIR Filter

A bit parallel filter is formed by processing the B bits of an input sample in B independent arrays, the outputs of these arrays are fed to a modulo adder structure for computing the filter output. The modulo adder structure is

shown in Fig 8.

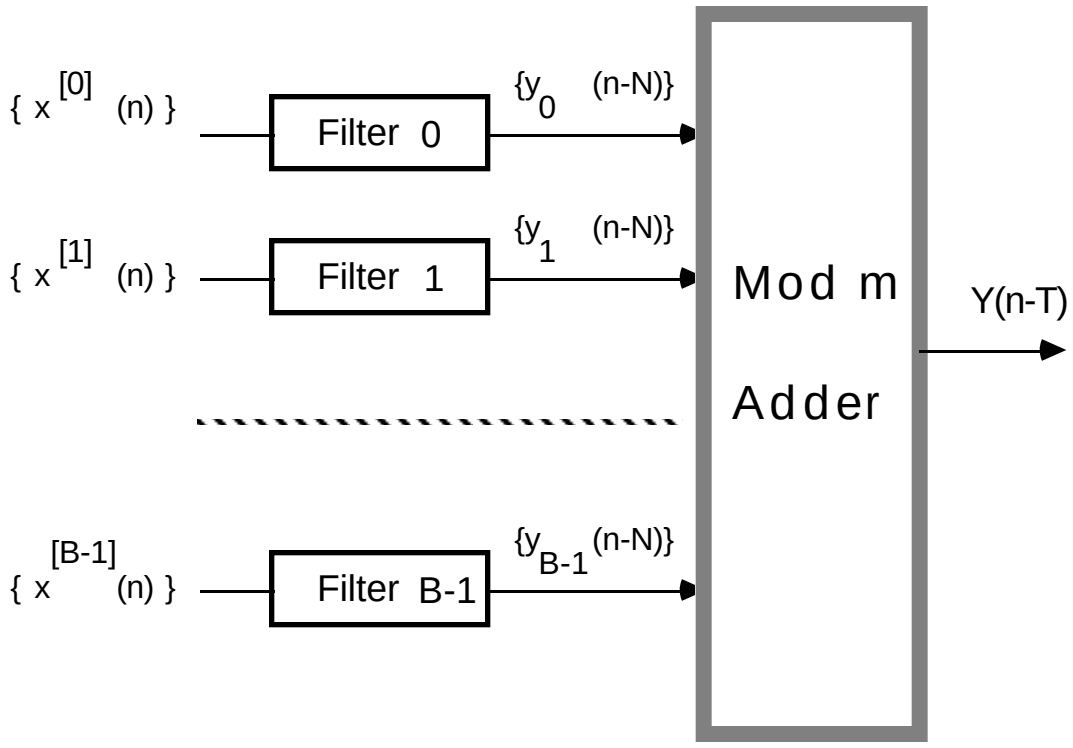


Fig 8 Bit parallel FIR filter structure

The entire filter structure can be implemented using the $BIPSP_m$ cell. The nth output sample of an Nth order fixed coefficient FIR filter can be expressed as:

$$|Y(n)|_m = \sum_{i=0}^{N-1}{}^{a} \{A(i)^{o} X(n-i)\} \qquad (13)$$

with A, X, Y ∈ R(m). Slicing (13) at the bit level yields:

$$|Y(n)|_m = \sum_{b=0}^{B-1}{}^{a} 2^{b\,o} \sum_{i=0}^{N-1}{}^{a} \{A(i)^{o} x^{[b]}(n-i)\} \qquad (14)$$

where $x^{[b]}$ is the bth bit of X. Eqn (12) can be written:

$$|Y(n)|_m = \sum_{b=0}^{B-1} {}^{a}\{2^{bo} |y_b(n)|_m\} \quad (15)$$

with:

$$|y_b(n)|_m = \sum_{i=0}^{N-1} {}^{a}\{A(i)^{o} x^{[b]}(n-i)\} \quad (16)$$

The elements of $|y_b(n)|_m$ (Filter b in Fig 9) can be computed by the following recursive relationship:

$$|y_b^{(0)}(n)|_m = 0 \quad (17a)$$

$$|y_b^{(i+1)}(n)|_m = |y_b^{(i)}(n)|_m {}^{a}\{A(i)^{o} x^{[b]}(n-i)\} \quad (17b)$$

$$|y_b(n)|_m = |y_b^{(N)}(n)|_m \quad (17c)$$

Eqn (17b) can be computed using the $BIPSP_m$ with a modified X data path, as illustrated in Fig 9.

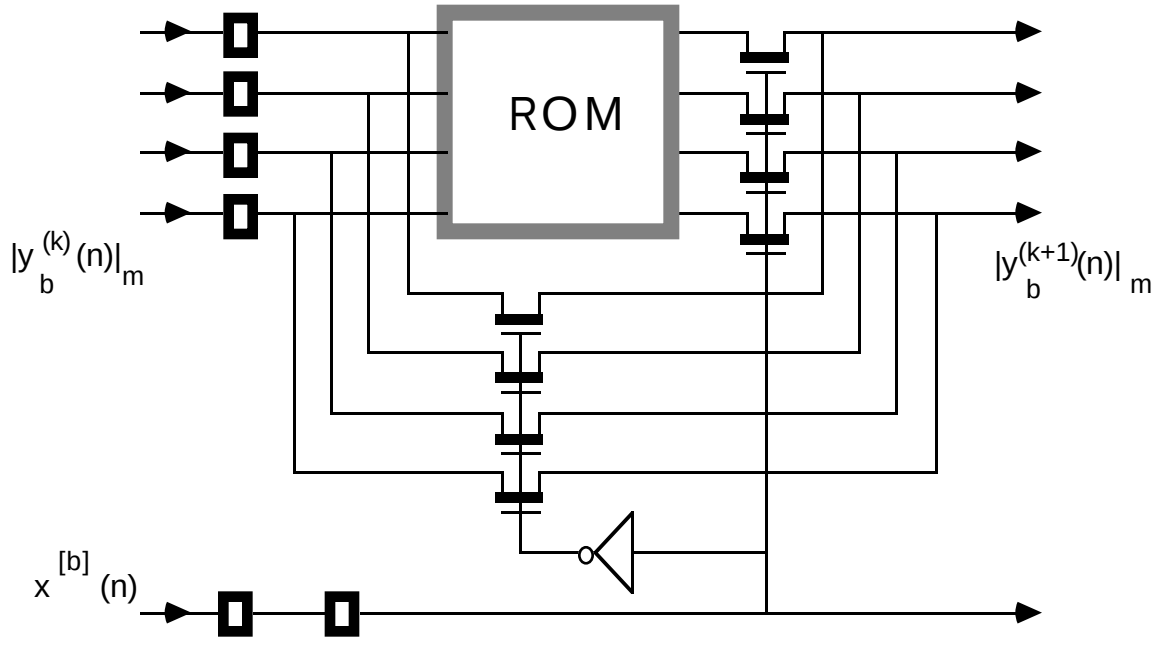


Fig 9 $BIPSP_m$ with a modified X data path

This structure is used in constructing the processing arrays of a filter with B=4. The ROM stores the function:

$$|y_b^{(i+1)}(n)|_m = |y_b^{(i)}(n)|_m \; a \; A(i) \qquad (18)$$

Multiplication by $x^{[b]}$ is performed by looking up the contents of the ROM ($x^{[b]}=1$) or steering the address to the output ($x^{[b]}=0$). B independent arrays are used to calculate $|y_b(n)|_m \; \forall \; b \in [0,B)$. Fig 10 shows the linear array for the bth bit of an Nth order filter.

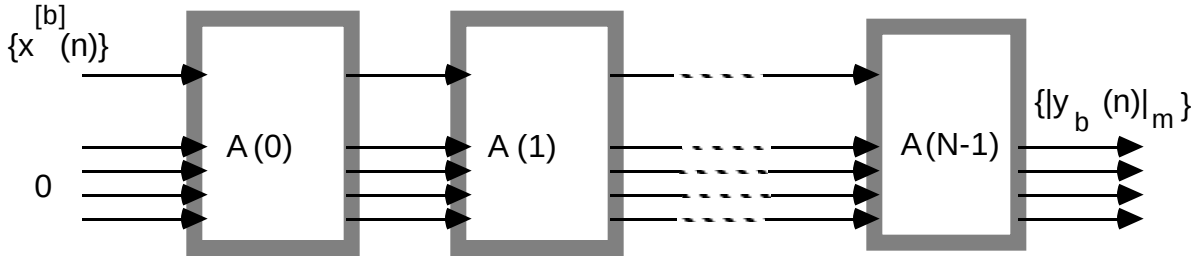


Fig 10 Linear array for an Nth order FIR filter

The overall computation scheme can be viewed as follows. The input sequence to the bth array is $x^{[b]}(n)$ which moves through the pipeline from left to right. The $\{|Y(k)|_m\}$, which are initially zero, are pipelined in the same direction as the {x} sequence. The {x} bits experience twice the delay of the {y} bits, this guarantees correct timing for a linear array with unidirectional data movement. One output word, $|Y^{(N)}|_m$, is computed per clock cycle in each one of the arrays (the cells in these arrays are 100% efficient with an area complexity of $O[N.n.B^2]$). Finally the elements of the sequence $\{|Y^{(N)}|_m\}$ are added in the modulo m adder to perform the summation of eqn (13).This operation is shown in Fig 11. The inputs to the array are skewed so that the correct terms of $\{|y_b(n)|_m, \forall \; b \in [0,B)\}$ are added together; the ROMs in the adder cells are programmed as:

$$SUM_{i,j}(out) = SUM_{i,j}(in) \; a \; (|2^{i+j+1+(B+2j)i}|_m \; o \; A(i) \qquad (19)$$

$$i \in [0,B-1) \; ; \; j \in [0,B)$$

The adder size and its structure only depends on B, as expected. The inputs to the adder are $\{|y_b(n)|_m \; ; \; b \in [0,B-1)\}$ and one sample, $|y(n)|_m$ is calculated every clock cycle. The cells in this systolic adder block are also 100% efficient.

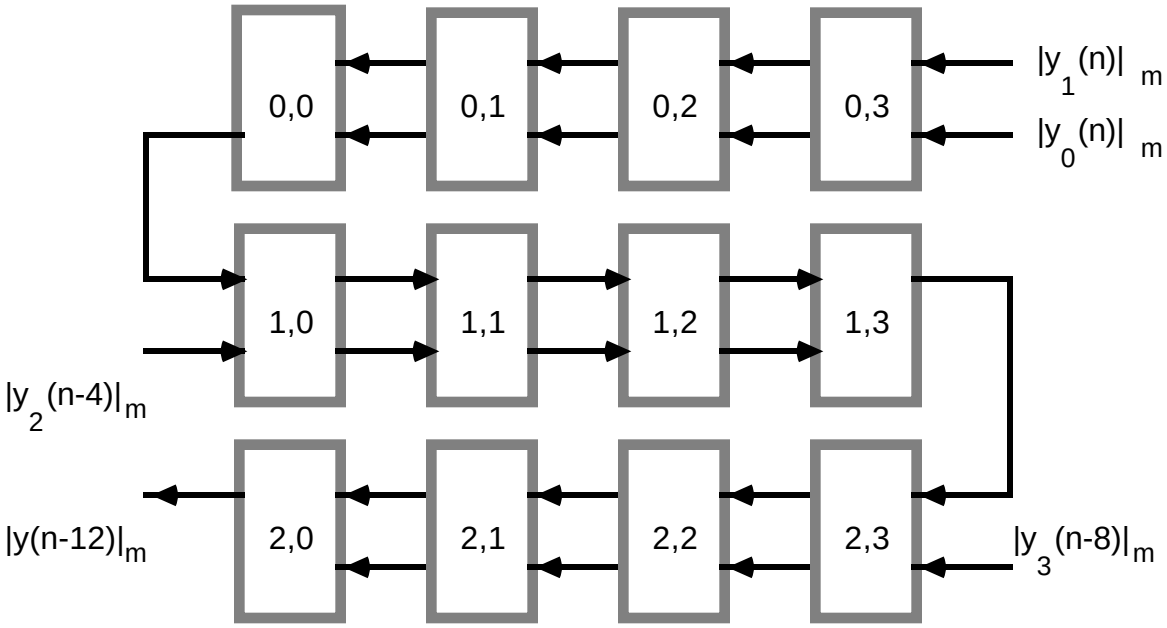


Fig 11 Modulo m adder for FIR filter

Bit Serial FIR filter

The cells used in this scheme perform the same basic function as the cells in Fig 9, but the number of delays on the input X path are increased from 2 to B+1, as shown in Fig 12 for B=4. This is equivalent to one extra delay for a serial B bit word.

The sequence $\{|y(n)|_m\}$ is computed sequentially in only one array using N cells. The array is similar to the bit parallel structure of Fig 10. The input bit sequence $\{x^{[b]}(n), b \in [0,B-1)\}$, is fed to the array one bit per clock cycle. One output word, $|y^{(N)}|_m$, is calculated per clock cycle. After B 'clock ticks', all the $\{|y^{(N)}|_m\}$ have been computed. The output sequence is fed to a serial adder block; this is shown in Fig 13.

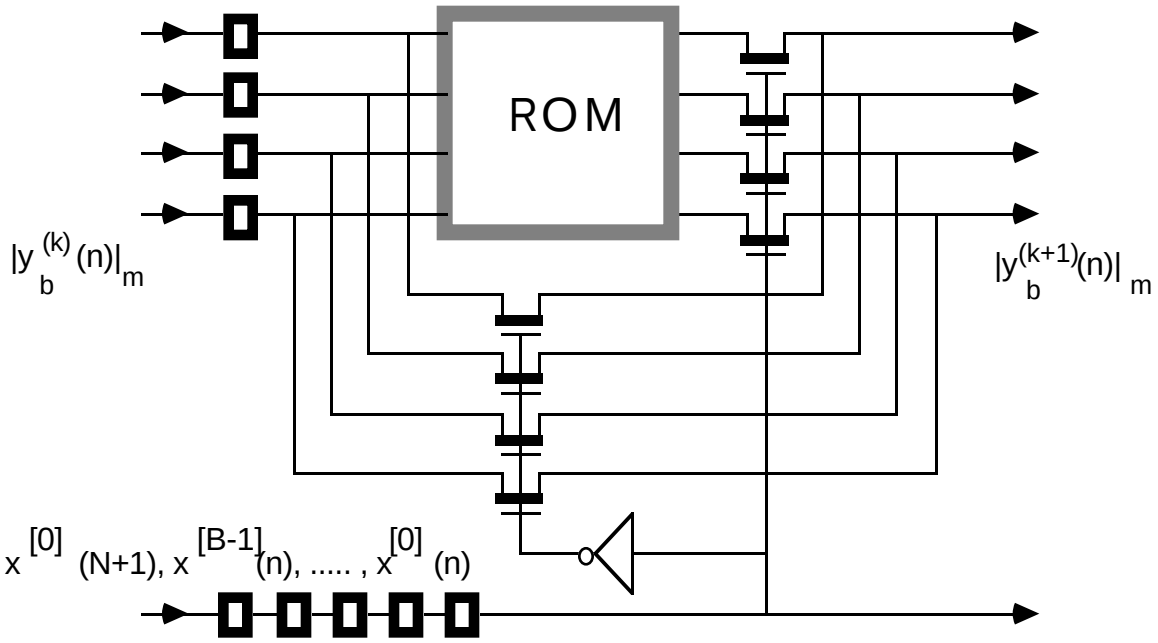


Fig 12 $BIPSP_m$ cell for the bit serial FIR filter

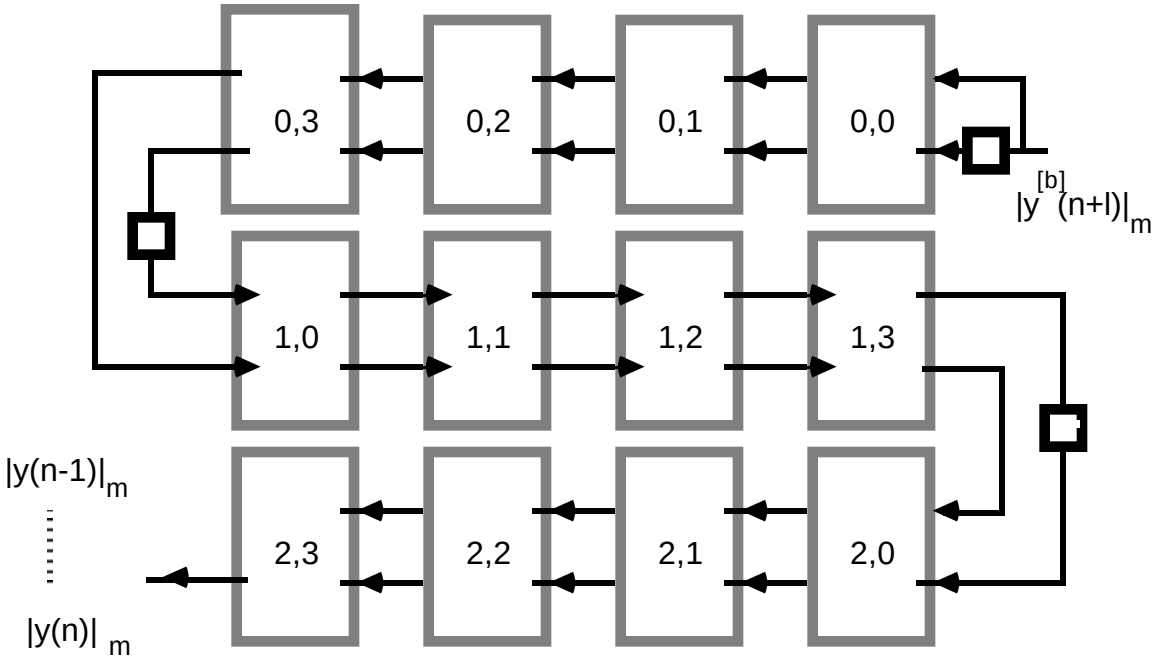


Fig 13 Adder block for bit serial FIR filter

The adder operates on the same basis as the bit parallel adder array. The

sequence $\{|y^{[b]}(n)|_m,\ b \in [0,B-1)\}$, enters the array at the top; data skewing is used to align the bit serial words in contrast to the use of direct connections in the parallel array of Fig 11. Clearly, complete output words are only available every B clock periods, yielding a 25% efficiency for a four bit ring; the separation of adjacent output samples by a dotted line is used to indicate this.

In order to show a typical VLSI structure using the cells connected in a linear systolic array, Fig 14 shows the bit serial array for a 20 tap FIR filter using a 5-bit modulus. The cells used are the static decoder/latch design of Fig 7.

Section III

In this section we will consider the problems of interfacing with binary arithmetic systems. Mathematically we perform a reversible mapping between the ring R(M) and its direct sum isomorphism $^{a}\sum R(m_i);\ i\in[1,L]$.

Forward Mapping

The mapping $R(M) \Rightarrow {}^{a}\sum R(m_i)$ is straightforward. The mapping operator is $\{|\bullet|_{m_i}\}$ and maps $X \Rightarrow \{x_1, x_2, \ldots, x_L\}$. The L bit binary number can be reduced, modulo m, as follows:

$$|X|_m = |\sum^{a} \{2^j o x[j]\}|_m \qquad (20)$$

Eqn (20) can be computed via the following recursion:

$$Y^{(0)} = 0 \qquad (21a)$$
$$Y^{(i+1)} = Y^{(i)}\ a\ \{2^i\ o\ x[i]\} \qquad (21b)$$
$$|X|_m = Y^{(L)} \qquad (21c)$$

Eqn (21b) can be calculated using the $BIPSP_m$ with L bits in the data path, as shown in Fig 6.

Fig 14 20 tap 5-bit modulus FIR filter
a) Composite Mask Layout

Fig 14 20 tap 5-bit modulus FIR filter

b) Floor plan

A linear array containing L stages of $BIPSP_m$ cells is capable of the modulo m

reduction. This is shown in Fig 15.

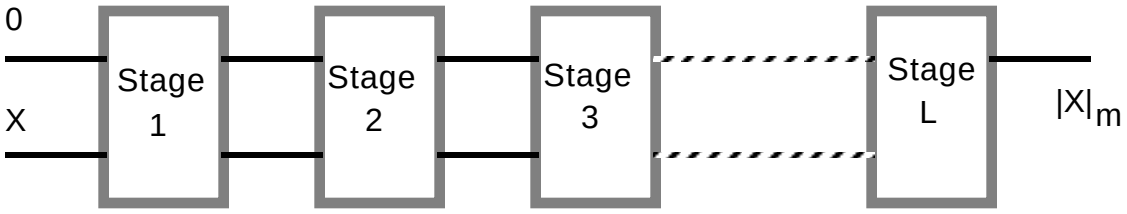


Fig 15 Modulo m reduction using an array of $BIPSP_m$ cells

If it is required to multiply the input by a fixed coefficient, A, this can be performed without any extra hardware. The stage can also correctly encode 2's complement form for the input binary sequence, by generating the additive inverse of an element in the ring, modulo m, if required. This is performed by mapping address to content within the final stage (MSB stage) ROM as follows:

$$\text{Address (i)} \rightarrow \{|-2^{L-1}|_m \circ A\} \Rightarrow \text{Content (i)} \qquad (22)$$

The procedure is illustrated in Fig 16. The columns represent ROM contents for a 16 bit binary to modulo 11 ring mapping. The mapping also includes a prestored multiplication of 13. An example of mapping -29x13 is shown, with active ROM contents outlined. The result is $|-29x13|_{11} = 8$. The ROM addresses are the column on the left, and the serial binary input is shown, starting at the LSB on the left, as the top row. Where the binary input is 0, no ROM contents are active (addresses steered past the ROMs).

	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1
0	2	4	8	5	10	9	7	3	6	1	2	4	8	5	10	2
1	3	5	9	6	0	10	8	4	7	2	3	5	9	6	0	3
2	4	6	10	7	1	0	9	5	8	3	4	6	10	7	1	4
3	5	7	0	8	2	1	10	6	9	4	5	7	0	8	2	5
4	6	8	1	9	3	2	0	7	10	5	6	8	1	9	3	6
5	7	9	2	10	4	3	1	8	0	6	7	9	2	10	4	7
6	8	10	3	0	5	4	2	9	1	7	8	10	3	0	5	8
7	9	0	4	1	6	5	3	10	2	8	9	0	4	1	6	9
8	10	1	5	2	7	6	4	0	3	9	10	1	5	2	7	10
9	0	2	6	3	8	7	5	1	4	10	0	2	6	3	8	0
10	1	3	7	4	9	8	6	2	5	0	1	3	7	4	9	1

Fig 16 ROM Contents for $|-29x13|_{11}$

This example also illustrates the cyclic nature of the ROM contents. The only information required to generate the entire ROM contents is the first location. The other locations follow in a cyclic fashion; this is direct evidence of the ROM contents forming an additive group under the ring addition operation. Although the contents always follow in this cyclic fashion, our efforts, so far, show that a general ROM structure is the most efficient implementation mechanism.

Reverse Mapping

The mapping ${}^{a}\sum R(m_i) \Rightarrow R(M)$ is always difficult, and has been the subject of many works on residue number systems[18].

The mathematical mapping is the Chinese Remainder Theorem (CRT) as shown in eqn (2). The CRT requires a modulo M adder, normally a drawback unless special moduli (close to a power of 2) are used[18]. For our purposes a scheme which uses the same small ring structures that we have been computing over would be ideal. The Mixed Radix Conversion procedure[1] is suitable, but produces a weighted magnitude output in an awkward form. A scheme recently disclosed[19], allows computation over the individual rings

and also produces the output in bit-sliced binary. The basic concept is to iterate around a loop that contains base extension (generating a new ring) to a power of 2 modulus, followed by scaling with that same ring modulus. The output is therefore produced in slices of bits, with the number of bits equal to the log base 2 of the extended ring modulus. If we use a ring modulus of, for example, 32, then the output will be in 5-bit slices. The operations required are all individual ring operations, and are associative, therefore we can use our linear systolic structure (with small modifications in data paths) to perform the mapping.

Often the mapped data is not required over the precision of the direct sum ring. Scaling strategies[6] can be used to great effect in this situation. The following example illustrates the point:

With four 5-bit moduli (say 32, 31, 29, 27) we have greater than 19 bits of dynamic range. If we only require the mapped output to have about 10 bits of dynamic range, then we can adopt an exact division scaling strategy[6]. By carefully arranging the order of the moduli, we will be able to produce the output in bit slices with only one iteration through the scaling array and no base extension. In the example used here, we arrange the moduli in the order {27, 31, 32, 29} and we plan to scale by 27x31. This will reduce the dynamic range to $32x29 \approx 2^{10}$.

A possible construction is shown in Fig 17 with the half tone blocks equivalent to 5 linearly connected cells and the solid block representing 5 cascades of 5 latches each. The data line shown superimposed on the half tone blocks represents the cyclically rotated parallel data path with access to the top serial bit The arrow between blocks 1 and 2 indicates that the serial bit for block 1 is obtained from the serial bit used for block 2, rather than its own serial bit. This allows existing data paths within each cell to be used more effectively. The output is obtained at the bottom of the array, with the ordering of bits as shown.

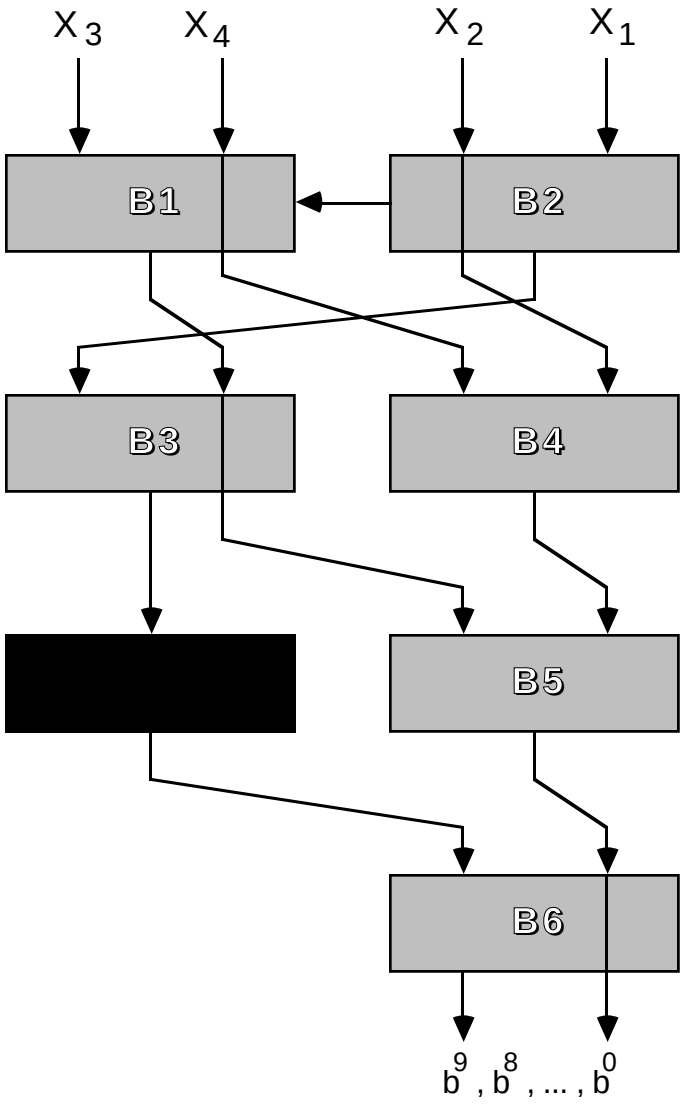


Fig 17 Scaling and reverse mapping array for four 5-bit moduli

The contents of each block have the form $R_i = \{(A\ {}^{a}_{m_k}\ B)\ {}^{o}_{m_k}\ K\}$, where A and B are variables and K is a constant multiplier. This is amenable to bit slicing (as for example in eqn (8)) and so each block in Fig 17 is capable of being sliced. The contents of the 6 blocks are given in eqn (23).

$$B1 = \{(X_3\ {}^{a}_{m_3}\ \gamma)\ {}^{a}_{m_3}\ [-(X_2\ {}^{a}_{m_2}\ \gamma)]\}\ {}^{o}_{m_3}\ \{|m_2^{-1}|_{m_3}\} \quad (23a)$$

$$B2 = \{(X_1\ {}^{a}_{m_1}\ \gamma)\ {}^{a}_{m_1}\ [-(X_2\ {}^{a}_{m_2}\ \gamma)]\}\ {}^{o}_{m_1}\ \{|m_2^{-1}|_{m_1}\} \quad (23b)$$

$$B3 = \{(B2\ {}^{a}_{m_3}\ [-B1]\}\ {}^{o}_{m_3}\ \{|m_1^{-1}|_{m_3}\} \quad (23c)$$

$$B4 = \{(X_4\ {}^{a}m_4\ \gamma)^{a}m_4\ [-(X_2\ {}^{a}m_2\ \gamma)]\}^{o}m_4\ \{|m_2^{-1}|_{m4}\} \quad (23d)$$

$$B5 = \{(B4\ {}^{a}m_4\ [-B1]\}^{o}m_4\ \{|m_1^{-1}|_{m4}\} \quad (23e)$$

$$B6 = \{(B3\ {}^{a}m_3\ [-B5]\}^{o}m_3\ \{|m_4^{-1}|_{m3}\} \quad (23f)$$

In eqn (23) the addition constant $\gamma = m_1.m_2/2$ is used to allow rounding of the estimate to the nearest integer rather than truncation, as normally happens with exact division scaling techniques.

Observations

We have demonstrated that typical multiply/add processing of integers (or fixed point numbers) can be implemented entirely by linear arrays of generic cells, consisting of a small ROM, steering switches and pipeline latches. The following points detail some of the advantages of using this approach.

i) For algorithms defined over finite rings or fields[2,3,20], the approach presented here is a natural one. This includes the use of quadratic residue number systems[7,20,21] where operations over the extension ring are mapped to base ring operations, and are amenable to bit slicing.

ii) Because of the ability to perform mapping operations to, and from, weighted magnitude representations (e.g. binary), the approach is also a natural replacement for conventional binary implementations.

iii) From a VLSI implementation standpoint, the use of parallel computations over small finite rings removes the problems of clock skew across large connected two dimensional bit-level systolic structures[11] that occur in, for example, pipelined or carry-save adders with large bit lengths. The cells presented in this paper have a very low '2nd dimension' to their structure, this dimension is only a function of the largest ring modulus used in the system. For example, we are able to emulate >24 bit computation using 5 moduli

each only 5 bits wide. Our structure will also allow the flexibility of deciding whether to segregate the individual rings within the same substrate or across different substrates. This can have important ramifications for both conventional multi-chip systems and wafer scale integrated systems.

iv) Because of the global use of a single generic cell, the use of fault detection circuitry for the cell (a fairly simple procedure[22]), instantly gives fault detection across an entire system. The use of residue systems for fault correction has been a strong-point in past published works[12,18], a combination of simple fault detection and natural correction facilities provides the basis for a robust implementation that is very fast.

Conclusions

In this paper we have introduced a general technique for implementing DSP operations, based on intensive multiply/add requirements, over finite rings. The implementation is based on bit-slicing the basic inner product sum processor (IPSP) to yield a BIPSP. The finite ring counterparts are $IPSP_m$ and $BIPSP_m$. The $BIPSP_m$ yields to a linear systolic implementation of any DSP algorithm that requires intensive multiply/add operations. An example of a cell structure for the $BIPSP_m$ has been given, including sample VLSI layouts using a 3μ single metal CMOS process offered by the Canadian Microelectronics Corporation. An example of the use of the technique to implement a fixed coefficient FIR filter, suitable for many communication requirements, has also been presented. Finally, it has been demonstrated that the basic cell can also be used in both forward and reverse mapping from the finite ring structure to more conventional representations, such as binary. An example is given illustrating the use of pre-scaling to limit the computational overhead in reverse mapping. The performance analysis and verification of submitted chip designs, and a detailed comparison between this technique

and conventional binary implementations has been intentionally omitted. This subject matter will be submitted for publication as a separate work, in order to afford appropriate attention to detail.

Acknowledgements

The authors wish to acknowledge the invaluable help from A. Yeung, J. Carr and P.V.R. Raja for the design of the VLSI cells and FIR filter layouts. Financial assistance for this research was provided by the Natural Sciences and Engineering Research Council of Canada, and the University of Windsor.

References

(1) N.S.Szabo and R.I.Tanaka, " Residue Arithmetic and its Applications to Computer Technology", McGraw-Hill, New York, 1967.

(2) I.S.Reed and T.K.Truong," The Use of Finite Fields to Compute Convolutions", IEEE trans. on Inform. Theory, Vol. IT-21, March 1975, pp.208-213.

(3) R.C.Agarwal and C.S.Burrus,"Number Theoretic Transforms to Implement Fast Digital Convolution", Proc. IEEE, Vol-63, April 1975, pp. 550-560.

(4) F.J. MacWilliams and N.J.A. Sloane, "The Theory of Error-Correcting Codes", New York: North Holland, 1977.

(5) W.K.Jenkins and B.J.Leon, "The Use of Residue Number Systems in the Design of Finite Impulse Response Digital Filters", IEEE Trans. Circuits and Systems, Vol. CAS-24, pp.191-201, April 1977

(6) G.A.Jullien, "Residue Number Scaling and Other Operations Using

ROM Arrays", IEEE Trans. on Computers, Vol. C-27, No.4, April 1978, pp.325-336.

(7) W.K. Jenkins, C.F. Lee, "Complex Residue Number System for Digital Signal Processing", Proc. 14th Asilomar Conf. on Circuits, Systems and Computers, Pacific Grove, CA. Nov. 1980.

(8) C.E. Leiserson, "Area Efficient VLSI Computation", Ph.D. Dissertation, Dept. Computer Science, Carnegie-Mellon University, Oct. 1981.

(9) P.R. Capello and K. Steiglitz, "Digital Signal Processing Applications of Systolic Algorithms", in VLSI Systems and Computations, H.T. Kung, R. Sproull and G. Steele eds., Computer Science Press, 1981, pp.245-254.

(10) H.T. Kung, "Why Systolic Architectures", IEEE Computer Magazine, Vol. 15, No. 1, Jan. 1982, pp.37-46.

(11) J.V. McCanny and J.G. McWhirter, "Implementation of Signal Processing Functions using 1-bit Systolic Arrays", Electronics Letters, Vol. 18, No. 6, March 1982, pp.241.

(12) J.V.Krogmeier and W.K.Jenkins,"Error detection and Correction in Quadratic Residue Number Systems", Proceedings of the 1983 Mid-West Symposium on Circuits and Systems, Puebla, Mexico, August 1983.

(13) R.E. Blahut, "Theory and Practice of Error Control Codes", Addison-Wesly, Reading MA., 1983.

(14) S.Y. Kung and J. Annevelink, "VLSI Design for Massively Parallel Signal Processors", Microprocessors and Microsystems, Vol.7, No. 10, December 1983, pp.461-467.

(15) D.E.R. Denning, "Cryptography and Data Security", Addison-Wesly, Reading MA., 1983.

(16) D.I. Moldovan, "On the Design of Algorithms for VLSI Systolic Arrays", Proc. of IEEE, Vol. 71, No. 1, Jan. 1983.

(17) M. Hatamian and G.L. Cash, "High Speed Signal Processing, Pipelining, and Signal Processing", Proc. ICASSP, April 1986, pp.1173-1176.

(18) M.A.Soderstrand, W.K. Jenkins, G.A. Jullien, F.J. Taylor, "Residue Number System Arithmetic: Modern Applications in Digital Signal Processing", IEEE Press, 1986.

(19) A.P. Senoy, R. Kumaresan, "Residue to Binary Conversion for RNS Arithmetic Using only Modular Look-up Tables", Submitted for publication to IEEE Trans. Circuits and Systems.

(20) G.A.Jullien, R.Krishnan and W.C.Miller," Complex Digital Signal Processing Over Finite Rings", IEEE.Trans. Circuits and Systems,(invited paper), special issue on complex signal processing, May, 1987

(21) W.K. Jenkins and J.V. Krogmier, "The Design of Dual-Mode Complex Signal Processors Based on Quadratic Modular Number Codes", IEEE.Trans. Circuits and Systems,(invited paper), special issue on complex signal processing, May 1987

(22) M. Taheri, G.A. Jullien, W.C. Miller, " Fault Detection in RNS Systolic Arrays", IEE Electronics Letters, (in press).

(23) M.A. Bayoumi, G.A. Jullien, W.C. Miller, "A Look-up Table VLSI Design Methodology for RNS Structures Used in DSP Applications", IEEE Trans. on Circuits and Systems, July 1987.

Abstract

This paper presents a simple, modular, architecture for very fast digital signal processing elements. The computation is performed over finite rings (or fields) and is able to emulate processing over the integer ring using residue number systems. The computations are restricted to closed operations (ring or field binary operators) with the ability to perform limited scaling operations. Computations naturally defined over finite mathematical systems (e.g. Number Theoretic Transforms, Quadratic Residue 'complex' calculations, Recursive FIR filters over finite fields) are also easily implemented using this new approach.

The technique evolves from the decomposition of each closed calculation using the ring/field associativity property. Linear systolic arrays, formed with multiple elements, each of a single generic form, are used for all calculations. The pipeline cycle is determined from the generic cell and is predicted to be very fast based on a critical path analysis. The cells are perfectly matched to the VLSI medium, and the resulting array structures are very dense indeed.

Examples of DSP applications are given to illustrate the technique, and sample cell and array VLSI layouts are presented for a 3μ CMOS process.

Authors

Majid Taheri, Ph. D. Candidate
Department of Electrical Engineering
University of Windsor
Windsor, Ontario, Canada N9B 3P4

Graham A. Jullien, Professor and Director VLSI Research Group
Department of Electrical Engineering
University of Windsor
Windsor, Ontario, Canada N9B 3P4

William C. Miller, Professor and Director CAD/CAM Centre
Department of Electrical Engineering
University of Windsor
Windsor, Ontario, Canada N9B 3P4

Phone number for the University of Windsor (519) 253-4232

Figure Captions

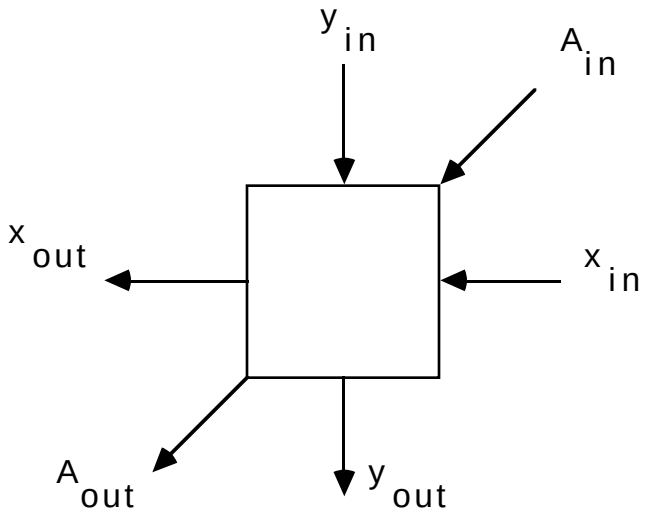


Fig 1 The IPSP

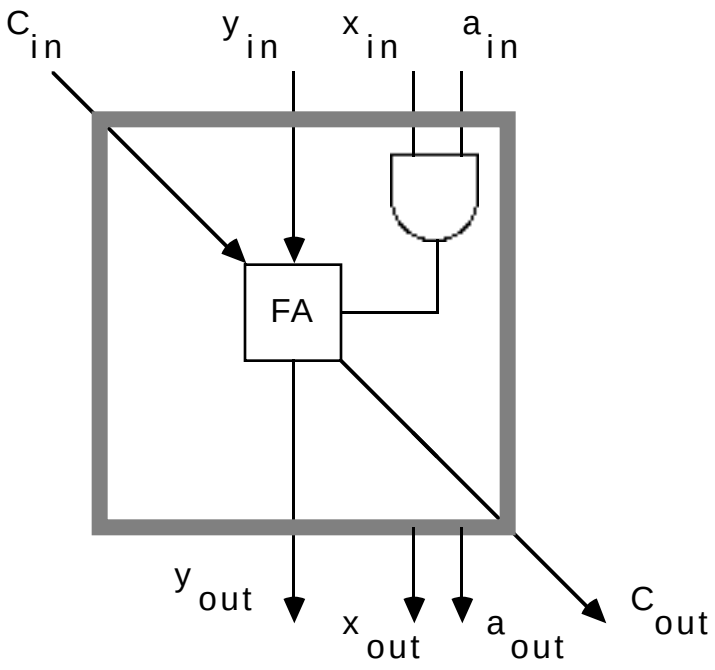


Fig 2 The BIPSP

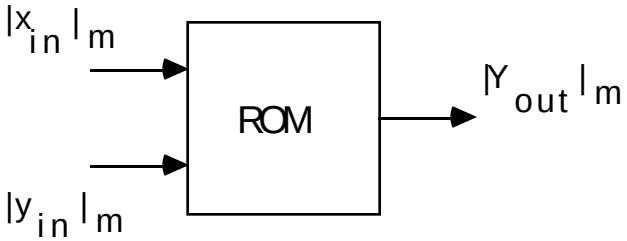


Fig 3 IPSP for Finite Ring Calculations

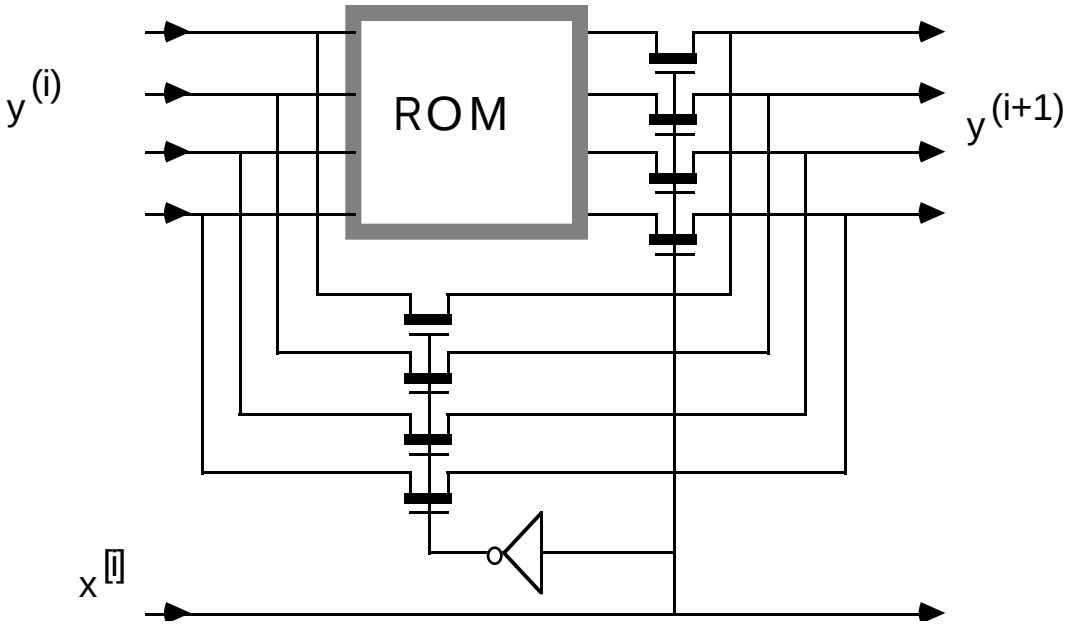


Fig 4 Implementation of the $BIPSP_m$ cell

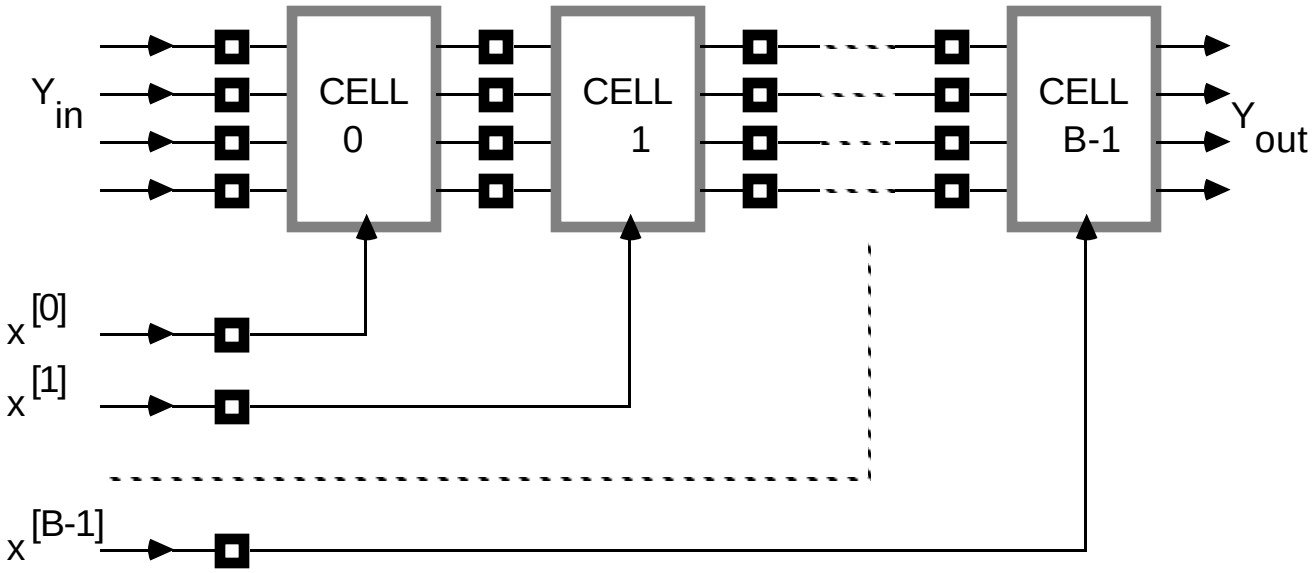


Fig 5 The $IPSP_m$ cell implemented using an array of $BIPSP_m$ cells

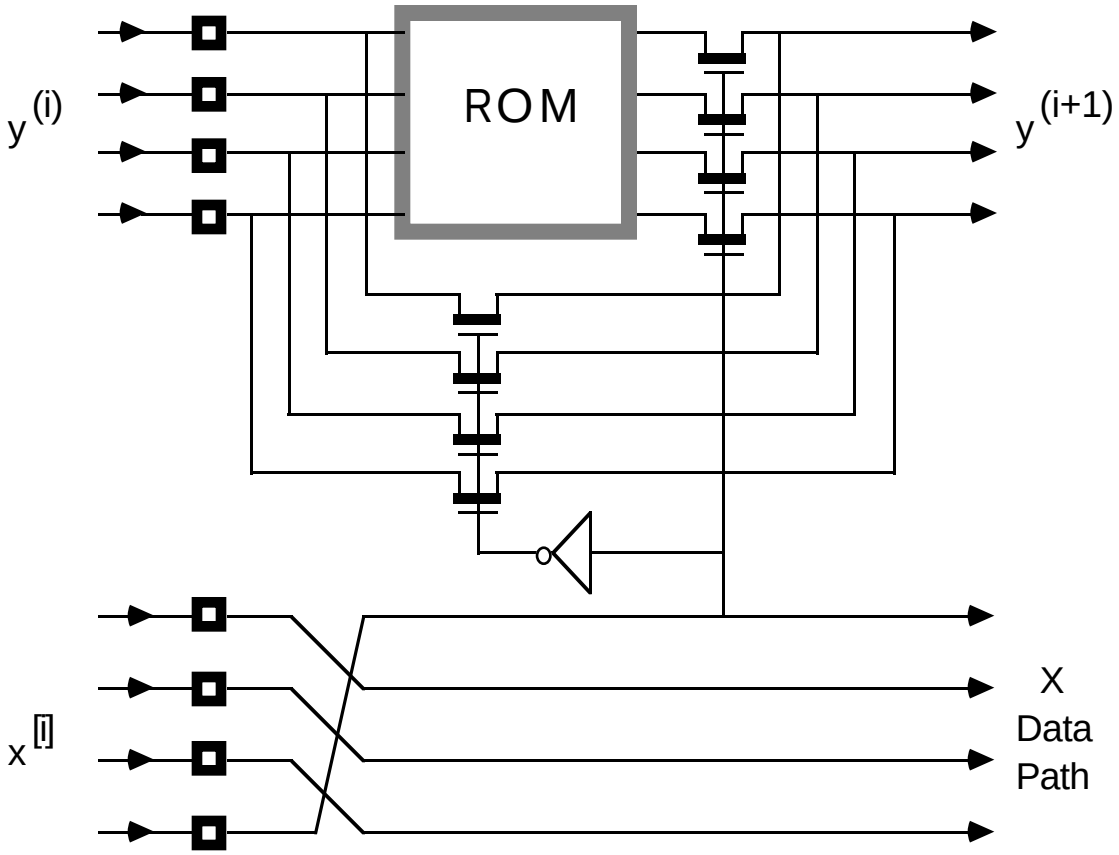


Fig 6 Systolic $BIPSP_m$ Cell

Fig 7 $BIPSP_m$ cell using static decoders and latches

a) Plot of the composite mask layout

Fig 7 $BIPSP_m$ cell using static decoders and latches
b) Floor plan showing the separate cell components

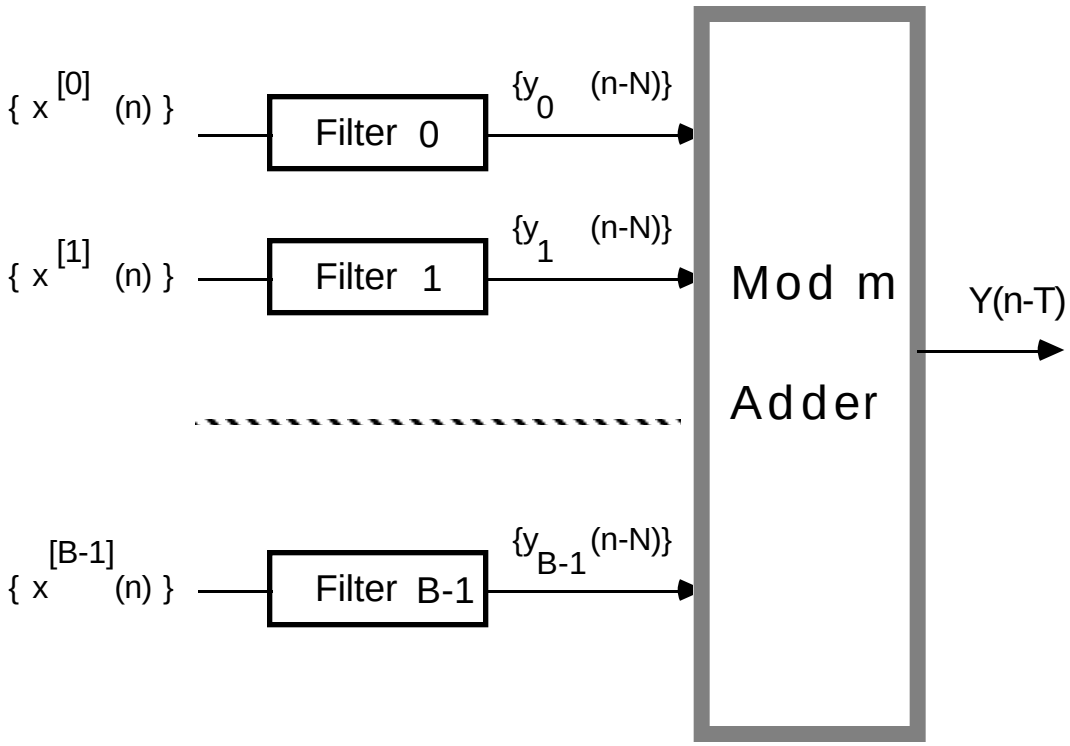


Fig 8 Bit parallel FIR filter structure

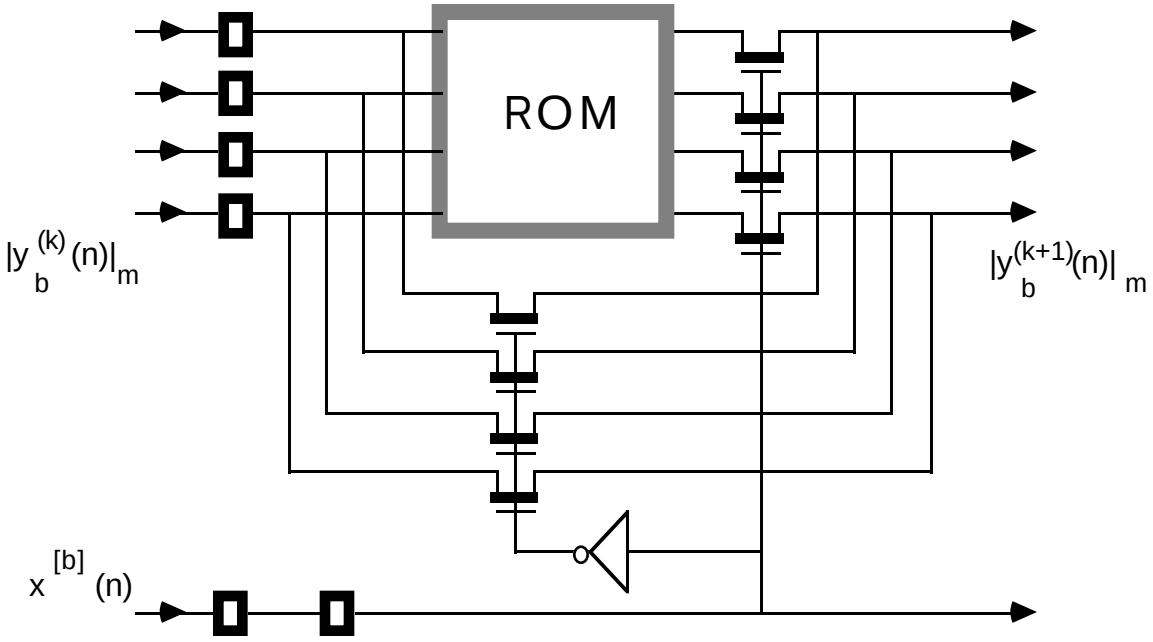


Fig 9 $BIPSP_m$ with a modified X data path

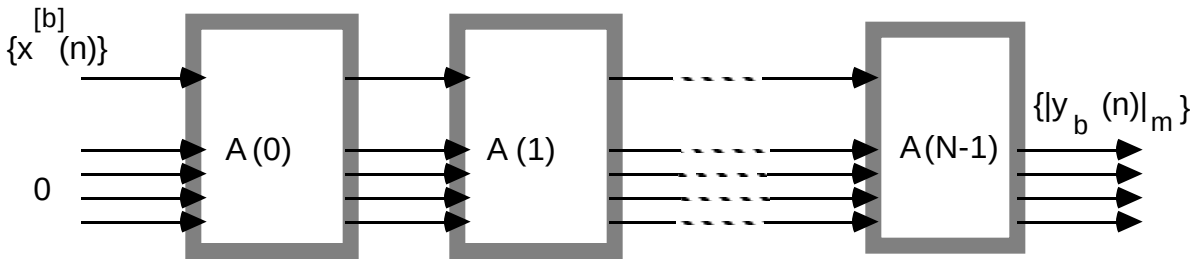


Fig 10 Linear array for an Nth order FIR filter

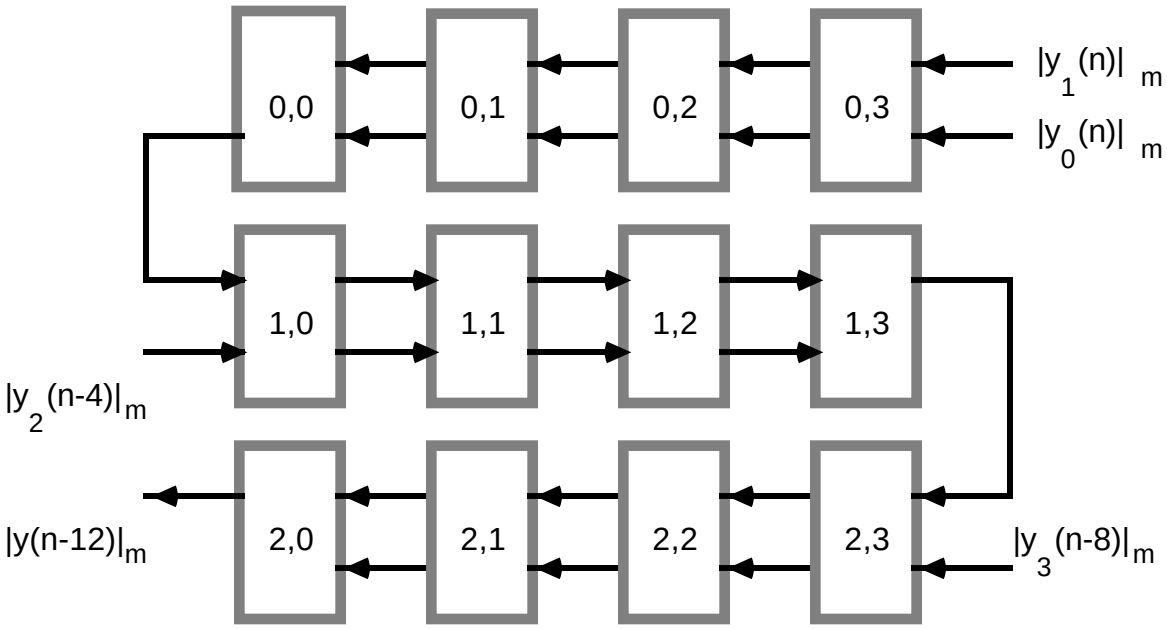


Fig 11 Modulo m adder for FIR filter

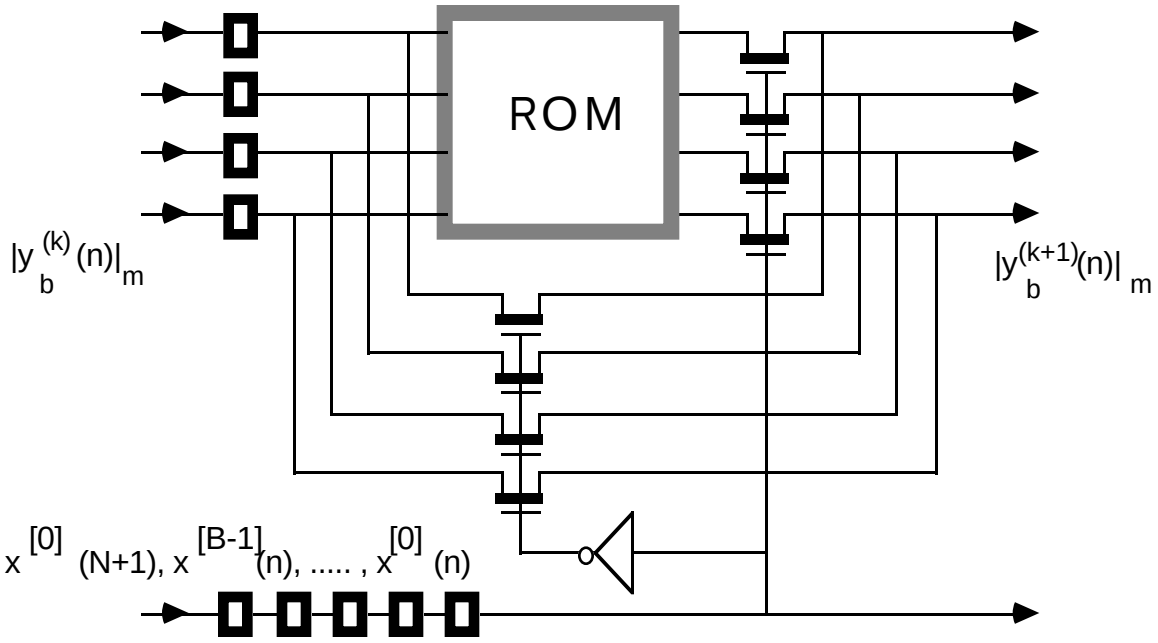


Fig 12 $BIPSP_m$ cell for the bit serial FIR filter

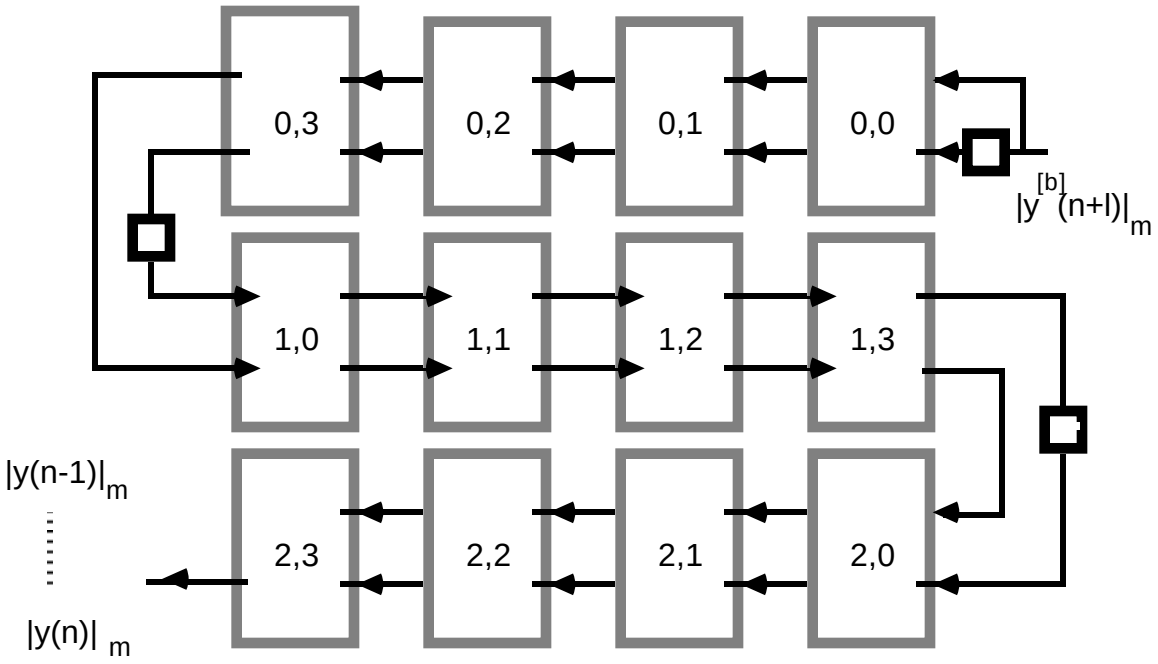


Fig 13 Adder block for bit serial FIR filter

Fig 14 20 tap 5-bit modulus FIR filter
a) Composite Mask Layout

Fig 14 20 tap 5-bit modulus FIR filter

b) Floor plan

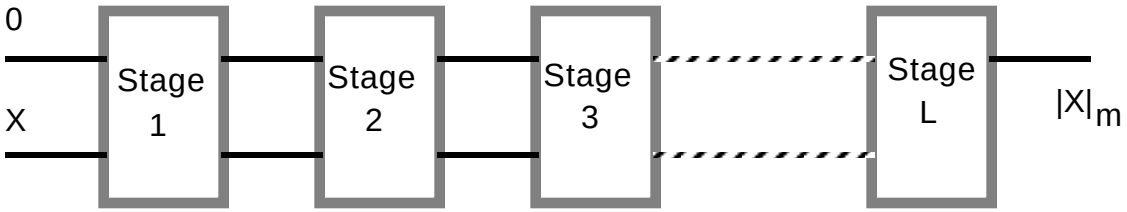
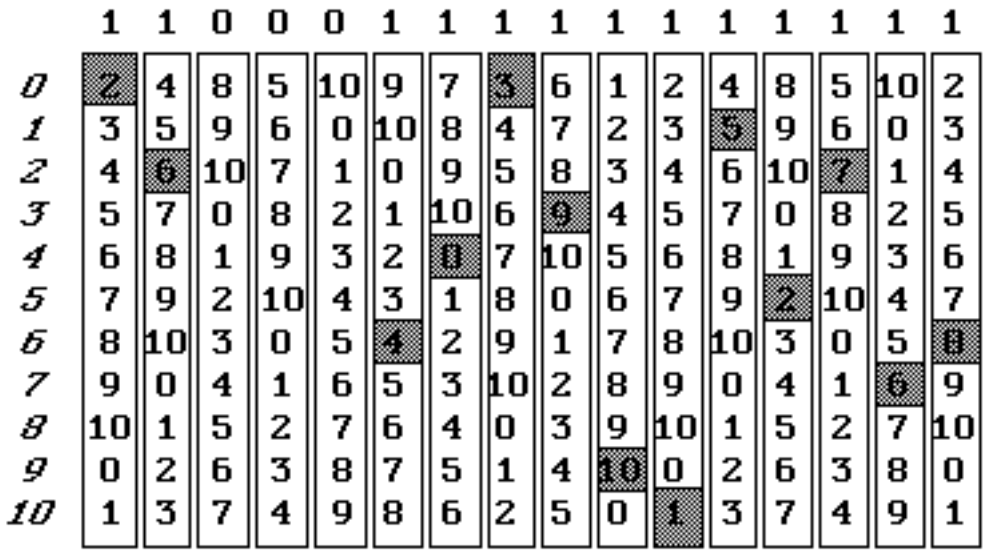


Fig 15 Modulo m reduction using an array of $BIPSP_m$ cells



	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1
0	2	4	8	5	10	9	7	3	6	1	2	4	8	5	10	2
1	3	5	9	6	0	10	8	4	7	2	3	5	9	6	0	3
2	4	6	10	7	1	0	9	5	8	3	4	6	10	7	1	4
3	5	7	0	8	2	1	10	6	9	4	5	7	0	8	2	5
4	6	8	1	9	3	2	0	7	10	5	6	8	1	9	3	6
5	7	9	2	10	4	3	1	8	0	6	7	9	2	10	4	7
6	8	10	3	0	5	4	2	9	1	7	8	10	3	0	5	8
7	9	0	4	1	6	5	3	10	2	8	9	0	4	1	6	9
8	10	1	5	2	7	6	4	0	3	9	10	1	5	2	7	10
9	0	2	6	3	8	7	5	1	4	10	0	2	6	3	8	0
10	1	3	7	4	9	8	6	2	5	0	1	3	7	4	9	1

Fig 16 ROM Contents for $|-29x13|_{11}$

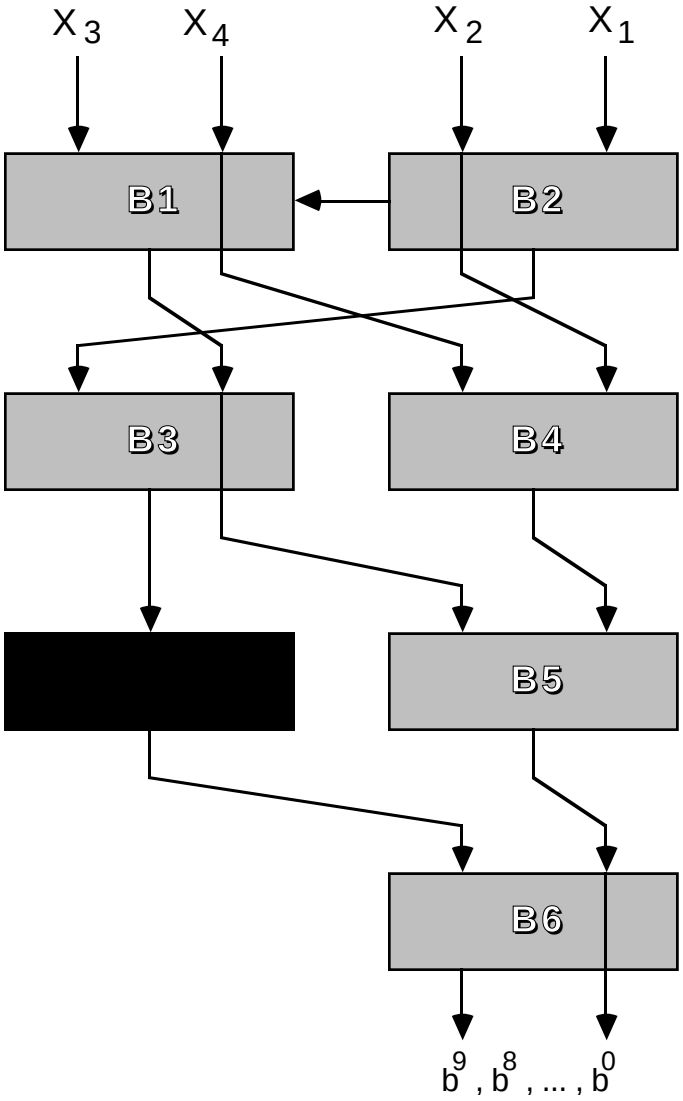


Fig 17 Scaling and reverse mapping array for four 5-bit moduli